



scikit-learn API Reference

Frameworks and libraries | The estimator / transformer / predictor contract, Pipeline and ColumnTransformer, model selection and search, metrics and scoring strings, common estimators with the hyperparameters to tune, set_output for pandas (scikit-learn 1.x, as of 2026)

#scikit-learn #Pipeline #GridSearchCV #Tabular

The contract every object follows

Estimator API

In one sentence

●○○○

Practitioner

●●○○

Expert

●●●○

Under the hood

●●●●

Every model and preprocessing step has the same verbs: fit to learn, then predict or transform.

Constructor = hyperparameters only; fit(X, y) learns and returns self; learned values end with an underscore.

- X: 2-D array-like (n_samples, n_features)
- y: 1-D labels or targets

Uniform verbs let Pipeline, GridSearchCV and cross_validate treat any object the same way.

- get_params / set_params expose hyperparameters as step__param
- clone copies them without fitted state

Mixins add verbs: ClassifierMixin (score = accuracy), RegressorMixin (score = R²), TransformerMixin (fit_transform). Since 1.6 put mixins left of BaseEstimator.

- check_estimator tests a custom class against the contract

Analogy

Like a power socket standard: any appliance with the right plug (fit, predict, transform) works in any socket (Pipeline, search, CV).

The scikit-learn API at four depths

Before Using scikit-learn fluently, can you explain...

	Not yet	Partly	Yes
MUST KNOW NumPy arrays and pandas DataFrames Learn: programming 02-03	☐	☐	☐
MUST KNOW Train / validation / test, leakage Learn: machine-learning 08	☐	☐	☐
MUST KNOW Classification and regression metrics Learn: machine-learning 07	☐	☐	☐
HELPFUL Linear models, trees, SVM, k-NN Learn: machine-learning 02-06	☐	☐	☐
HELPFUL Python classes and inheritance Learn: python folder	☐	☐	☐

How the base classes compose. Your own estimator inherits the same way

Method	Who has it	Returns	Note
fit(X, y=None)	all	self	learns fitted_ attributes
transform(X)	transformers	new X	never refit on test data
fit_transform(X, y)	transformers	new X	train data only
predict(X)	predictors	labels / values	classes_ order for classifiers
predict_proba(X)	most classifiers	(n, n_classes)	SVC needs probability=True
decision_function(X)	linear, SVM	scores	ranking metrics accept it
score(X, y)	predictors	float	accuracy or R ² by default
get_feature_names_out()	transformers	names	after fit
partial_fit(X, y)	SGD, NB, MiniBatchK-Means	self	out-of-core streaming

Methods by role. "Self" means the method returns the estimator, so calls chain: model.fit(X, y).predict(X2)

→ Pipelines and column routing

What a full Pipeline does inside each cross-validation fold

Tadata | tadata.fr | contact@tadata.codes

```
pipeline.py: mixed columns, one leak-free estimator
import sklearn
from sklearn.compose import ColumnTransformer, make_column_selector as sel
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression

sklearn.set_config(transform_output="pandas") # DataFrames out
num = make_pipeline(SimpleImputer(strategy="median"),
                    StandardScaler())

cat = make_pipeline(SimpleImputer(strategy="most_frequent"),
                    OneHotEncoder(handle_unknown="ignore",
                                  sparse_output=False))

pre = ColumnTransformer([
    ("num", num, sel(dtype_include="number")),
    ("cat", cat, sel(dtype_include="object")),
], remainder="drop")
clf = Pipeline([("pre", pre),
                ("model", LogisticRegression(max_iter=1000))])
clf.fit(X_train, y_train)
clf.predict_proba(X_test)[:1, 1]
clf[:-1].get_feature_names_out() # num__age, cat__city_Lyon, ...
```

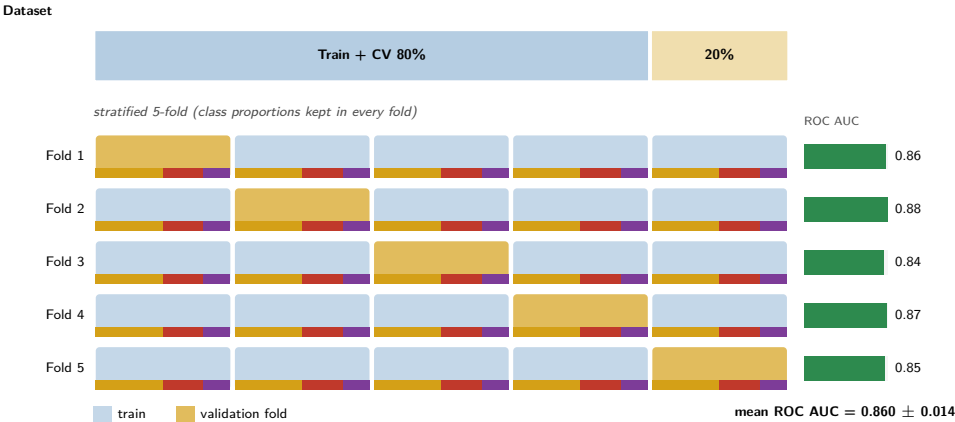
Need	How
Address a nested hyperparameter	step__param, e.g. pre__num__simpleimputer__strategy; make_pipeline names steps by lower-cased class
Skip or try without a step	set the step to "passthrough"; in a grid: {"pre__cat": [cat, "drop"]}
Keep untouched columns	remainder="passthrough" (default is "drop")
Custom function as a step	FunctionTransformer(np.log1p, feature_names_out="one-to-one")
Cache expensive fitted steps	Pipeline(..., memory="cache_dir")
Inspect a fitted step	clf.named_steps["model"].coef_ or clf["model"]; slicing clf[:-1] gives the preprocessing
Pandas or Polars output	.set_output(transform="pandas") per object, or sklearn.set_config(transform_output="pandas") globally; "polars" since 1.4
Transform the target	TransformedTargetRegressor(regressor=..., func=np.log1p, inverse_func=np.expml)

Pipeline and Column Transformer details you will need

fit_transform(X_train) learns and applies statistics Tell: Calling fit on test data is leakage	vs	transform(X_test) applies train statistics
Pipeline steps in sequence on all columns Tell: Sequence vs side by side	vs	Column Transformer steps in parallel on column subsets
cross_val_score one metric, array of scores Tell: Need more than one number? cross_validate	vs	cross_validate several metrics, fit times, train scores
GridSearchCV every combination Tell: Continuous parameters or many dims: randomized	vs	RandomizedSearchCV n_iter samples from distributions
predict hard label at threshold 0.5 Tell: ROC AUC and calibration need probabilities	vs	predict_proba probability per class

Pairs beginners mix up

Model selection and metrics



StratifiedKFold(n_splits=5, shuffle=True, random_state=0) on a held-out split; mean ± std of the fold scores is what you report

```
search.py: tune the whole pipeline, never the model alone
from sklearn.model_selection import (GridSearchCV, StratifiedKFold,
                                     cross_validate, train_test_split)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=0)
grid = {"model__C": [0.1, 1, 10],
        "model__class_weight": [None, "balanced"]}
cv = StratifiedKFold(5, shuffle=True, random_state=0)
search = GridSearchCV(clf, grid, cv=cv, scoring="roc_auc",
                      n_jobs=-1, refit=True)
search.fit(X_train, y_train)
search.best_params_, search.best_score_
search.score(X_test, y_test) # one final check
cross_validate(clf, X_train, y_train, cv=cv,
               scoring=["roc_auc", "f1"], return_train_score=True)
```

How many fits does a search cost?

Y

GridSearchCV above: 3 values of C × 2 class weights, cv = 5, refit=True. How many model fits, and how long at 4 s per fit on 8 cores?
Given: fits = G × k + R, where G = number of grid combinations, k = number of CV folds, R = 1 if refit else 0; time ≈ fits × t / j, where t = seconds per fit, j = parallel jobs (n_jobs)

Y

Step 1: Count combinations?

G = 3 × 2 = 6
grid is a Cartesian product

T

Y

Step 2: CV fits?

6 × 5 = 30
each combination fitted once per fold

T

Y

Step 3: Add the refit?

30 + 1 = 31 fits
best combination refitted on all of X_train

T

Y

Step 4: Wall time?

31 × 4 s / 8 ≈ 15.5 s (+ the refit runs alone, so ≈ 16-20 s)
n_jobs=-1 spreads CV fits across cores

T

31 fits, about 20 s of wall time

Check: RandomizedSearchCV(n_iter=10) with cv=5 would cost 10 × 5 + 1 = 51 fits regardless of grid size

T

Object	Use when	Key arguments
KFold / StratifiedKFold	iid rows; classification keeps class ratio	n_splits, shuffle, random_state
GroupKFold	several rows per patient, user, site	groups= in fit / split
TimeSeriesSplit	ordered data; validate on the future	n_splits, gap, test_size
RandomizedSearchCV	large or continuous spaces	param_distributions, n_iter
HalvingGridSearchCV	successive halving, cheap first rounds	needs enable_halving_search_cv import
TunedThresholdClassifierCV	pick the decision threshold by CV (1.5+)	scoring, cv
classification_report	precision, recall, F1 per class	y_true, y_pred, digits
roc_auc_score	ranking quality	probabilities, multi_class='ovr'
f1_score	imbalanced classes	average='macro' 'weighted' 'binary'
root_mean_squared_error	regression error in y units (1.4+)	y_true, y_pred
make_scorer	custom or parametrised metric	greater_is_better, response_method
ConfusionMatrixDisplay	plot from a fitted model	.from_estimator(clf, X, y)

Splitters, searches and metric functions (sklearn.model_selection, sklearn.metrics)

Scoring strings are 'higher is better'

Errors are negated: scoring="neg_root_mean_squared_error" returns -RMSE, so best_score_ = -3.2 means RMSE 3.2. List all names with sklearn.metrics.get_scorer_names().

Common estimators and the knobs to tune

Estimator	Key hyperparameters (first to tune first)	Notes
LogisticRegression	C (1/regularisation, 0.01-100 log), penalty, solver, class_weight, max_iter	scale features; raise max_iter if not converged
Ridge / Lasso / ElasticNet	alpha (regularisation, log grid), l1_ratio	RidgeCV, LassoCV tune alpha internally
SVC / SVR	C, kernel ('rbf'), gamma ('scale')	O(n ²) memory; > 50k rows use LinearSVC
KNeighborsClassifier	n_neighbors (5-50), weights, metric	always scale; slow predict on big data
DecisionTreeClassifier	max_depth, min_samples_leaf, criterion	ccp_alpha for cost-complexity pruning
RandomForestClassifier	n_estimators (200-1000), max_features, min_samples_leaf	n_jobs=-1; oob_score=True for free validation
HistGradientBoostingClassifier	learning_rate, max_iter, max_leaf_nodes, l2_regularization	native NaN and categorical_features; early_stopping='auto'
MLPClassifier	hidden_layer_sizes, alpha, learning_rate_init	small data only; use PyTorch beyond
KMeans	n_clusters, n_init, init='k-means++'	scale first; inertia_ and silhouette_score
DBSCAN / HDBSCAN	eps, min_samples / min_cluster_size	HDBSCAN in sklearn.cluster since 1.3
PCA	n_components (int or variance share 0.95)	explained_variance_ratio_
IsolationForest	contamination, n_estimators	predict gives -1 for outliers
OneHotEncoder	handle_unknown='ignore', min_frequency, sparse_output	drop='first' only for unregularised linear models
TargetEncoder	smooth, cv (internal cross-fitting)	high-cardinality categories (1.3+)

Start from defaults, tune the bold-faced first knob; ranges are usual starting grids

joblib.dump(clf, "model.joblib")	Save a fitted pipeline; load only files you trust (pickle executes code)
skops.io.dump(clf, "model.skops")	Safer format that checks types on load
skl2onnx.to_onnx(clf, X[:1])	Export to ONNX for C++, Java or browser runtimes (sheet 09)
sklearn.utils.estimator_checks.check_estimator(MyEst())	Test a custom estimator against the API contract
clf.get_params(deep=True).keys()	Every parameter name a grid may address

A custom transformer that plays well with Pipeline and set_output

```
import numpy as np
from sklearn.base import BaseEstimator, TransformerMixin

class Clip(TransformerMixin, BaseEstimator):
    # mixins first
    def __init__(self, q=0.99):
        # hyperparameters only
        self.q = q
    def fit(self, X, y=None):
        self.upper_ = np.quantile(X, self.q, axis=0)
        self.n_features_in_ = X.shape[1]
        return self
        # always return self
    def transform(self, X):
        return np.minimum(X, self.upper_)
    def get_feature_names_out(self, names=None):
        return names
        # one-to-one
```

✓ Recap and self-test

MYTH

✗ *Scale the whole dataset, then split*

✗ *best_score_ is the expected production score*

✗ *OneHotEncoder must drop a column*

✗ *scikit-learn cannot use pandas column names*

FACT

✓ **Split first, fit preprocessing inside a Pipeline**
The scaler's mean and std would otherwise include test rows: optimistic scores.

✓ **It is optimistic: it was used to choose**
Report search.score on the untouched test set, or use nested CV.

✓ **Only for unregularised linear models**
Trees and regularised models are fine with all columns; with drop='first' an unseen category looks like the dropped one.

✓ **set_output(transform='pandas') keeps them**
Plus feature_names_in_ checks column names at predict time.

Key takeaways

01 **fit learns, transform / predict apply**
learned values end with _

02 **Put every preprocessing step in a Pipeline**
ColumnTransformer routes columns

03 **Tune the pipeline with step__param**
fits = combinations × folds + 1

04 **Pick the splitter that matches the data**
Stratified, Group, TimeSeries

05 **Scoring is higher-is-better**
neg_ prefix for errors

06 **set_output for DataFrames**
get_feature_names_out after fit

Q1 What does fit return, and why does it matter?

A self, so calls chain (model.fit(X, y).predict(X2)) and Pipeline can treat every step alike.

Q2 How do you name the C of a LogisticRegression step called 'model' in a grid?

A model__C (double underscore between step and parameter).

Q3 Which transformer applies different preprocessing to numeric and categorical columns?

A ColumnTransformer (with make_column_selector to select by dtype).

Q4 Which splitter for several rows per customer?

A GroupKFold (pass groups=customer_id), so one customer never sits in train and validation.

Q5 GridSearchCV with 4×3 grid, cv=5, refit=True: how many fits?

A $4 \times 3 \times 5 + 1 = 61$.

Q6 best_score_ = -0.42 with scoring='neg_mean_absolute_error'. MAE?

A 0.42: errors are negated so that higher is better.

Q7 How do you get pandas DataFrames out of every transformer?

A sklearn.set_config(transform_output='pandas') or .set_output(transform='pandas') on the object.

Q8 Which estimator handles NaN and categorical columns natively?

A HistGradientBoostingClassifier / Regressor (categorical_features, missing values routed during splits).

Q9 Why is OneHotEncoder(handle_unknown='ignore') the safe default?

A A category unseen in training becomes all zeros at predict time instead of raising an error.