KD nuggets

CHEAT SHEET

SCIKIT-LEARN

THE FEATURE ENGINEERING PIPELINE

SERIES: PYTHON | LEVEL: INTERMEDIATE | UPDATED: 2026

Feature engineering in scikit-learn belongs **inside** the estimator, not in a notebook cell above it. A `Pipeline` chains transformers and a final model into one object with a single `fit`, so each step learns from training data only. That is what stops a scaler from seeing the validation fold and quietly inflating your score. Everything below is a step you can drop into that chain.

Pipeline

Chains transformers and a final estimator into one object. Fitting it fits each step in turn on the output of the one before.

```
from sklearn.pipeline import Pipeline

pipe = Pipeline([
    ("prep", preprocessor),
    ("model", LogisticRegression()),
])
pipe.fit(X_train, y_train)
```

```
pipe[:-1].transform(X_train).shape
(4500, 32)
```

ColumnTransformer

Applies different transformers to different column subsets, then glues the results back together side by side.

```
from sklearn.compose import ColumnTransformer

preprocessor = ColumnTransformer([
    ("num", num_pipe, ["age", "income"]),
    ("cat", cat_pipe, ["city", "plan"]),
], remainder="drop")
```

make_column_selector

Picks columns by dtype or name pattern instead of listing them, so new columns are handled without editing the pipeline.

```
from sklearn.compose import (
    make_column_selector as selector)

("num", num_pipe, selector(dtype_include="number")),
("cat", cat_pipe, selector(dtype_include=object)),
```

SimpleImputer

Fills gaps with a value learned from the training set. `add_indicator` keeps a flag marking what was missing, which is often signal in itself. See also `KNNImputer`.

```
from sklearn.impute import SimpleImputer

SimpleImputer(strategy="median",
              add_indicator=True)

# KNNImputer(n_neighbors=5)
```

StandardScaler · MinMaxScaler

Puts numeric columns on a common scale. Reach for `RobustScaler` when outliers would drag the mean and variance around.

```
from sklearn.preprocessing import StandardScaler

StandardScaler()   # mean 0, sd 1
MinMaxScaler()     # squeeze to [0, 1]
RobustScaler()     # median and IQR
```

OneHotEncoder

Turns categories into indicator columns. `handle_unknown` stops a category the model never saw from raising at predict time.

```
from sklearn.preprocessing import OneHotEncoder

OneHotEncoder(
    handle_unknown="ignore",
    min_frequency=0.01,
    sparse_output=False,
)
```

OrdinalEncoder

Maps categories to integers. Use it only when the order is real, or for tree models that split on the integer anyway.

```
from sklearn.preprocessing import OrdinalEncoder

OrdinalEncoder(
    categories=[["low", "mid", "high"]],
    handle_unknown="use_encoded_value",
    unknown_value=-1,
)
```

TargetEncoder

Replaces a high-cardinality category with a smoothed mean of the target. It cross-fits internally, so the encoding does not leak the label.

```
from sklearn.preprocessing import TargetEncoder

TargetEncoder(smooth="auto", cv=5)
```

PolynomialFeatures

Adds powers and pairwise products of numeric columns. Column count grows fast, so `interaction_only` is usually the safer setting.

```
from sklearn.preprocessing import PolynomialFeatures

PolynomialFeatures(degree=2,
                   interaction_only=True,
                   include_bias=False)
```

```
['age', 'income', 'age income']
```

FunctionTransformer

Wraps a stateless function as a transformer. Nothing is fitted, so it is safe anywhere in the chain.

```
from sklearn.preprocessing import FunctionTransformer
import numpy as np

FunctionTransformer(
    np.log1p, feature_names_out="one-to-one")
```

SelectKBest

Keeps the highest-scoring features by a univariate test. Inside a pipeline the selection is refit on every training fold.

```
from sklearn.feature_selection import (
    SelectKBest, f_classif, VarianceThreshold)

SelectKBest(f_classif, k=20)
VarianceThreshold(threshold=0.0)
```

set_output · get_feature_names_out

Returns DataFrames instead of arrays and recovers the names your transformers produced — the fastest way to see what the pipeline actually built.

```
pipe.set_output(transform="pandas")
pipe[:-1].get_feature_names_out()[:3]
```

```
array(['num__age', 'num__income',
       'cat__city_Lisbon'], dtype=object)
```

GridSearchCV

Tunes preprocessing and model together. Double underscores address nested steps, so an imputation strategy is a hyperparameter like any other.

```
from sklearn.model_selection import GridSearchCV

grid = {
  "prep__num__imputer__strategy": ["median", "mean"],
  "model__C": [0.1, 1, 10],
}
GridSearchCV(pipe, grid, cv=5).fit(X, y)
```

cross_val_score

Scores the whole pipeline, refitting every transformer on each fold. Scaling or encoding before this call is the classic source of leaked results.

```
from sklearn.model_selection import cross_val_score

cross_val_score(pipe, X, y, cv=5,
    scoring="roc_auc").mean()
```

```
0.8731
```

KBinsDiscretizer · SplineTransformer

Gives a numeric column a nonlinear basis so a linear model can bend along it. Splines keep the transition smooth where hard bin edges jump.

```
from sklearn.preprocessing import SplineTransformer

SplineTransformer(n_knots=5, degree=3)

# KBinsDiscretizer(n_bins=5, encode="onehot-dense")
```

SelectFromModel · RFECV

Selects on what a fitted estimator makes of each feature, so interactions a univariate test misses still count.

```
from sklearn.feature_selection import (
    SelectFromModel, RFECV)

SelectFromModel(
    LogisticRegression(penalty="l1",
                       solver="liblinear"),
    threshold="median")
```