# Exercise 1

尝试一个支持向量机回归器(**sklearn.svm.SVR**)，具有各种超参数，如 **kernel="linear"** （超参数 C 的不同值）或 **kernel="rbf"** （超参数 C 和 gamma 的不同值）。请注意，支持向量机不能很好地扩展到大型数据集，所以您应该只在训练集的前5000个实例上训练模型，并且只使用3折交叉验证，否则将需要几个小时。现在不要担心超参数的含义；我们将在第五章中讨论它们。最好的SVR预测器的表现如何？

**答案：**

In [50]:

```python
import sklearn
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import FunctionTransformer
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.utils.validation import check_array, check_is_fitted
from sklearn.compose import make_column_selector, make_column_transformer
from pathlib import Path
import pandas as pd
import tarfile
import urllib.request
from sklearn.metrics.pairwise import rbf_kernel
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
```

In [35]:

```python
def load_housing_data():
    tarball_path = Path("datasets/housing.tgz")
    if not tarball_path.is_file():
        Path("datasets").mkdir(parents=True, exist_ok=True)
        url = "https://github.com/ageron/data/raw/main/housing.tgz"
        urllib.request.urlretrieve(url, tarball_path)

    with tarfile.open(tarball_path) as housing_tarball:
        housing_tarball.extractall(path="datasets")

    return pd.read_csv(Path("datasets/housing/housing.csv"))

housing = load_housing_data()
```

In [36]:

```python
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3., 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])

strat_train_set, strat_test_set = train_test_split(
    housing,
    test_size=0.2,
    stratify=housing["income_cat"],
    random_state=42)

housing = strat_train_set.drop("median_house_value", axis=1)

housing_labels = strat_train_set["median_house_value"].copy()
```

In [37]:

```
from sklearn.cluster import KMeans

class ClusterSimilarity(BaseEstimator, TransformerMixin):
    def __init__(self, n_clusters=10, gamma=1.0, random_state=None):
        self.n_clusters = n_clusters
        self.gamma = gamma
        self.random_state = random_state

    def fit(self, X, y=None, sample_weight=None):
        self.kmeans_ = KMeans(self.n_clusters, random_state=self.random_state)
        self.kmeans_.fit(X, sample_weight=sample_weight)
        return self  # always return self!

    def transform(self, X):
        return rbf_kernel(X, self.kmeans_.cluster_centers_, gamma=self.gamma)

    def get_feature_names_out(self, names=None):
        return [f"Cluster {i} similarity" for i in range(self.n_clusters)]
```

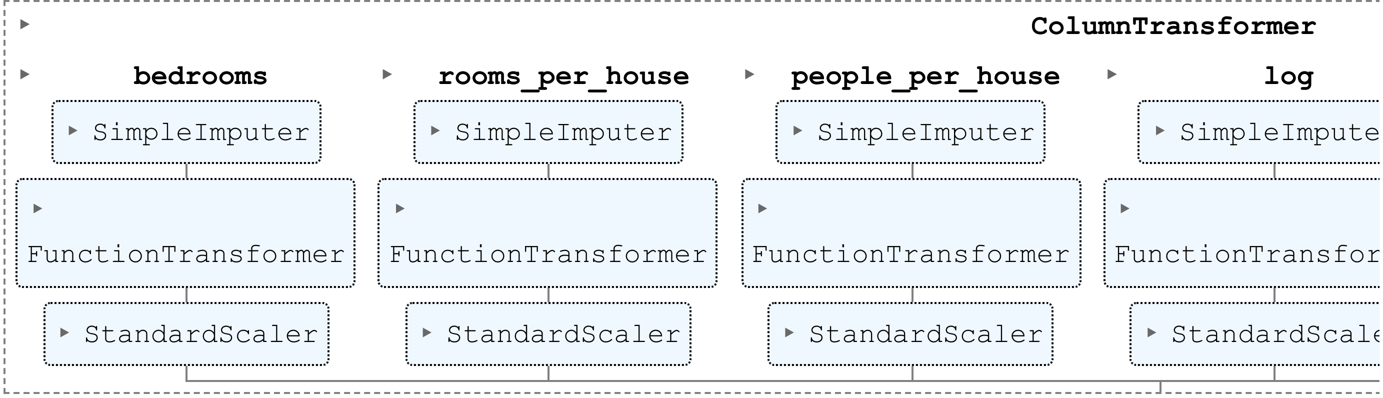
In [38]:

```
cat_pipeline = make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    OneHotEncoder(handle_unknown="ignore"))
```

In [39]:

```
def column_ratio(X):
    return X[:, [0]] / X[:, [1]]

def ratio_name(function_transformer, feature_names_in):
    return ["ratio"]  # feature names out

def ratio_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="median"),
        FunctionTransformer(column_ratio, feature_names_out=ratio_name),
        StandardScaler())

log_pipeline = make_pipeline(
    SimpleImputer(strategy="median"),
    FunctionTransformer(np.log, feature_names_out="one-to-one"),
    StandardScaler())

cluster_simil = ClusterSimilarity(n_clusters=10, gamma=1., random_state=42)

default_num_pipeline = make_pipeline(SimpleImputer(strategy="median"),
                                     StandardScaler())

preprocessing = ColumnTransformer([
        ("bedrooms", ratio_pipeline(), ["total_bedrooms", "total_rooms"]),
        ("rooms_per_house", ratio_pipeline(), ["total_rooms", "households"]),
        ("people_per_house", ratio_pipeline(), ["population", "households"]),
        ("log", log_pipeline, ["total_bedrooms", "total_rooms", "population",
                               "households", "median_income"]),
        ("geo", cluster_simil, ["latitude", "longitude"]),
        ("cat", cat_pipeline, make_column_selector(dtype_include=object)),
    ],
    remainder=default_num_pipeline)  # one column remaining: housing_median_age

preprocessing
```

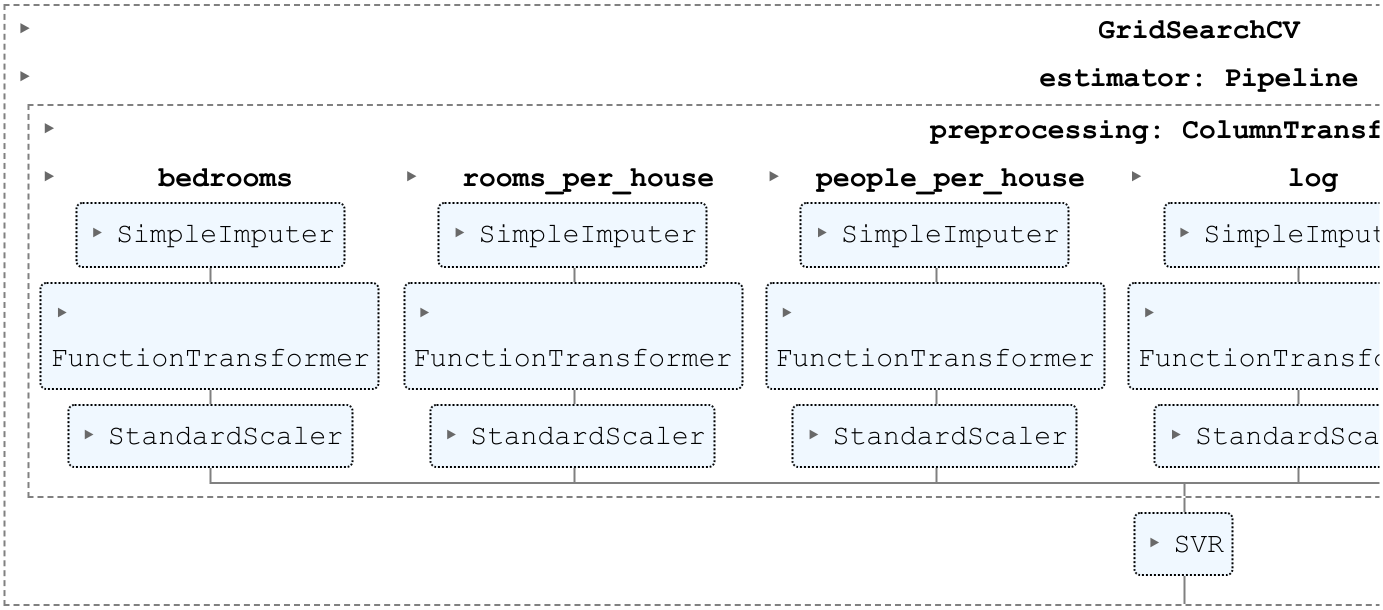
Out[39]:



In [40]:

```python
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

param_grid = [
        {'svr__kernel': ['linear'], 'svr__C': [10., 30., 100., 300., 1000.,
                                               3000., 10000., 30000.0]},
        {'svr__kernel': ['rbf'], 'svr__C': [1.0, 3.0, 10., 30., 100., 300.,
                                            1000.0],
         'svr__gamma': [0.01, 0.03, 0.1, 0.3, 1.0, 3.0]},
    ]

svr_pipeline = Pipeline([("preprocessing", preprocessing), ("svr", SVR())])

grid_search = GridSearchCV(svr_pipeline, param_grid, cv=3,
                           scoring='neg_root_mean_squared_error')

grid_search.fit(housing.iloc[:5000], housing_labels.iloc[:5000])
```

Out[40]:



In [41]:

```python
svr_grid_search_rmse = -grid_search.best_score_

svr_grid_search_rmse
```

Out[41]:

```
69062.06517312386
```

In [42]:

```python
grid_search.best_params_
```

Out[42]:

```
{'svr__C': 10000.0, 'svr__kernel': 'linear'}
```

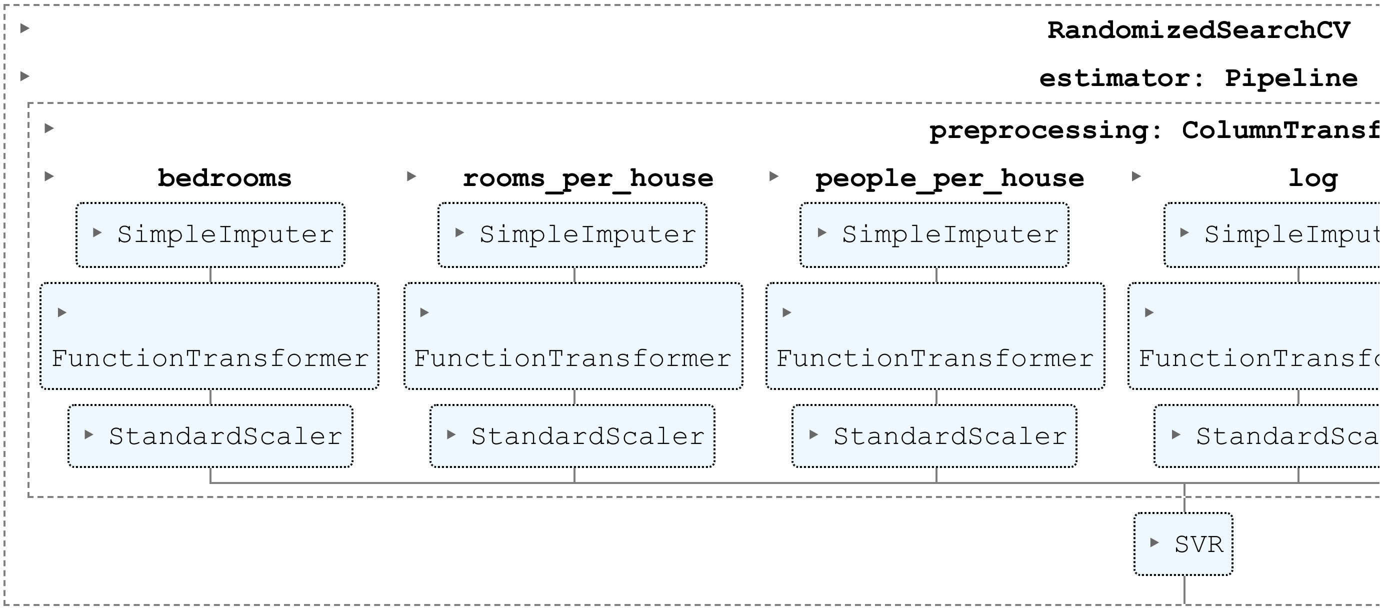
# Exercise 2

试着用 RandomizedSearchCV 替换 GridSearchCV 。

答案：

In [43]:
```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import expon, loguniform

# see https://docs.scipy.org/doc/scipy/reference/stats.html
# for `expon()` and `loguniform()` documentation and more probability distribution funct

# Note: gamma is ignored when kernel is "linear"
param_distribs = {
        'svr__kernel': ['linear', 'rbf'],
        'svr__C': loguniform(20, 200_000),
        'svr__gamma': expon(scale=1.0),
    }

rnd_search = RandomizedSearchCV(svr_pipeline,
                                param_distributions=param_distribs,
                                n_iter=50, cv=3,
                                scoring='neg_root_mean_squared_error',
                                random_state=42)

rnd_search.fit(housing.iloc[:5000], housing_labels.iloc[:5000])
```

Out[43]:



In [44]:
```python
svr_rnd_search_rmse = -rnd_search.best_score_

svr_rnd_search_rmse
```

Out[44]:
```
56313.77847635623
```

In [45]:
```python
rnd_search.best_params_
```

Out[45]:
```
{'svr__C': 157055.10989448498,
 'svr__gamma': 0.26497040005002437,
 'svr__kernel': 'rbf'}
```

# Exercise 3

尝试在准备 pipeline 中添加一个 **SelectFromModel** 转换器，以只选择最重要的属性。

答案：

In [48]:
```
from sklearn.feature_selection import SelectFromModel

selector_pipeline = Pipeline([
    ('preprocessing', preprocessing),
    ('selector', SelectFromModel(RandomForestRegressor(random_state=42),
                                 threshold=0.005)),  # min feature importance
    ('svr', SVR(C=rnd_search.best_params_["svr__C"],
                gamma=rnd_search.best_params_["svr__gamma"],
                kernel=rnd_search.best_params_["svr__kernel"])),
])
```

In [51]:
```
selector_rmses = -cross_val_score(selector_pipeline,
                                  housing.iloc[:5000],
                                  housing_labels.iloc[:5000],
                                  scoring="neg_root_mean_squared_error",
                                  cv=3)

pd.Series(selector_rmses).describe()
```

Out[51]:
```
count        3.000000
mean     56211.362085
std       1922.002802
min      54150.008629
25%      55339.929908
50%      56529.851186
75%      57242.038813
max      57954.226441
dtype: float64
```

# Exercise 4

尝试创建一个自定义转换器，在其 fit() 方法中训练 k-最近邻回归器 (sklearn.neighbors.KNeighborsRegressor)，并在其 transform() 方法中输出模型的预测。 然后将此功能添加到预处理管道，使用纬度和经度作为此转换器的输入。 这将在模型中添加一个与最近地区的住房中位数价格相对应的特征。

答案：

让我们创建一个接受任何回归器的转换器，而不是将我们自己限制在 k近邻回归器上。为此，我们可以扩展 **MetaEstimatorMixin** 并在构造函数中有一个必需的估计器参数。 **fit()** 方法必须在该估计器的克隆上工作，并且还必须保存 feature_names*in*。 MetaEstimatorMixin 将确保将估算器列为必需参数，并将更新 get_params() 和 set_params() 以使估算器的超参数可用于调整。最后，我们创建一个 get_feature_names_out() 方法。

In [56]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.base import MetaEstimatorMixin, clone

class FeatureFromRegressor(MetaEstimatorMixin, BaseEstimator, TransformerMixin):
    def __init__(self, estimator):
        self.estimator = estimator

    def fit(self, X, y=None):
        estimator_ = clone(self.estimator)
        estimator_.fit(X, y)
        self.estimator_ = estimator_
        self.n_features_in_ = self.estimator_.n_features_in_
        if hasattr(self.estimator, "feature_names_in_"):
            self.feature_names_in_ = self.estimator.feature_names_in_
        return self  # always return self!

    def transform(self, X):
        check_is_fitted(self)
        predictions = self.estimator_.predict(X)
        if predictions.ndim == 1:
            predictions = predictions.reshape(-1, 1)
        return predictions

    def get_feature_names_out(self, names=None):
        check_is_fitted(self)
        n_outputs = getattr(self.estimator_, "n_outputs_", 1)
        estimator_class_name = self.estimator_.__class__.__name__
        estimator_short_name = estimator_class_name.lower().replace("_", "")
        return [f"{estimator_short_name}_prediction_{i}"
                for i in range(n_outputs)]
```

让我们确保它符合 Scikit-Learn 的 API：

In [57]:

```
from sklearn.utils.estimator_checks import check_estimator

check_estimator(FeatureFromRegressor(KNeighborsRegressor()))
```

好的！ 现在让我们测试一下：

In [58]:

```
knn_reg = KNeighborsRegressor(n_neighbors=3, weights="distance")

knn_transformer = FeatureFromRegressor(knn_reg)

geo_features = housing[["latitude", "longitude"]]

knn_transformer.fit_transform(geo_features, housing_labels)
```

Out[58]:

```
array([[486100.66666667],
       [435250.        ],
       [105100.        ],
       ...,
       [148800.        ],
       [500001.        ],
       [234333.33333333]])
```

它的输出特征名称是什么样的？

In [60]:

```
knn_transformer.get_feature_names_out()
```

Out[60]:

```
['kneighborsregressor_prediction_0']
```

好的，现在让我们将这个转换器包含在我们的预处理管道中：

In [61]:

```
from sklearn.base import clone

transformers = [(name, clone(transformer), columns)
                for name, transformer, columns in preprocessing.transformers]
geo_index = [name for name, _, _ in transformers].index("geo")
transformers[geo_index] = ("geo", knn_transformer, ["latitude", "longitude"])

new_geo_preprocessing = ColumnTransformer(transformers)

new_geo_pipeline = Pipeline([
    ('preprocessing', new_geo_preprocessing),
    ('svr', SVR(C=rnd_search.best_params_["svr__C"],
                gamma=rnd_search.best_params_["svr__gamma"],
                kernel=rnd_search.best_params_["svr__kernel"])),
])

new_pipe_rmses = -cross_val_score(new_geo_pipeline,
                                  housing.iloc[:5000],
                                  housing_labels.iloc[:5000],
                                  scoring="neg_root_mean_squared_error",
                                  cv=3)

pd.Series(new_pipe_rmses).describe()
```

Out[61]:

```
count         3.000000
mean     105035.412299
std        2918.402445
min      101812.910219
25%      103802.972098
50%      105793.033978
75%      106646.663339
max      107500.292700
dtype: float64
```

# Exercise 5

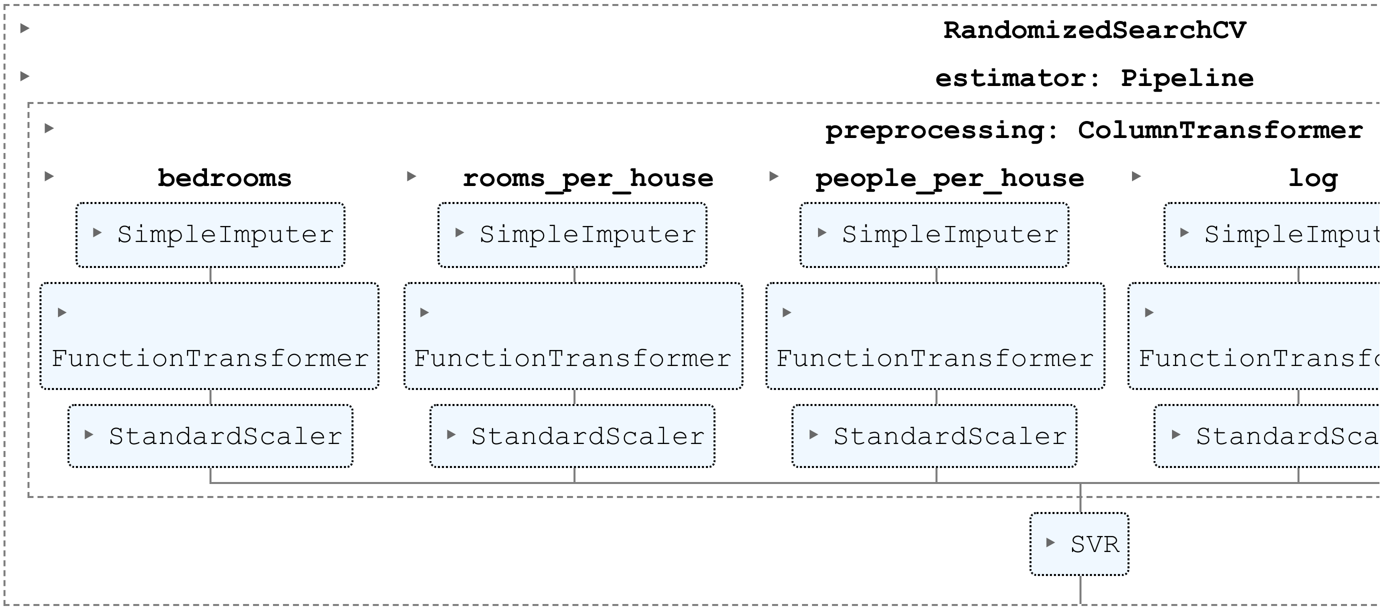
使用RandomizedSearchCV自动探索一些备选项。

答案：

In [62]:

```
param_distribs = {
    "preprocessing__geo__estimator__n_neighbors": range(1, 30),
    "preprocessing__geo__estimator__weights": ["distance", "uniform"],
    "svr__C": loguniform(20, 200_000),
    "svr__gamma": expon(scale=1.0),
}

new_geo_rnd_search = RandomizedSearchCV(new_geo_pipeline,
                                        param_distributions=param_distribs,
                                        n_iter=50,
                                        cv=3,
                                        scoring='neg_root_mean_squared_error',
                                        random_state=42)

new_geo_rnd_search.fit(housing.iloc[:5000], housing_labels.iloc[:5000])
```

Out[62]:



In [63]:
```
new_geo_rnd_search_rmse = -new_geo_rnd_search.best_score_

new_geo_rnd_search_rmse
```

Out[63]:
```
106867.28988103945
```

# Exercise 6

尝试从头开始再次实现 StandardScalerClone 类，然后添加对 inverse_transform() 方法的支持：执行 scaler.inverse_transform (scaler.fit_transform(X)) 应该返回一个非常接近 X 的数组。然后添加对特征名称的支持：如果输入是 DataFrame， 则在 fit() 方法中设置 feature_names*in* 。该属性应该是列名称的 NumPy 数组。最后，实现 get_feature_names_out() 方法：它应该有一个可选的 input_features=None 参数。如果通过，该方法应检查其长度是否与 n_features*in* 匹配，如果已定义，则应与 feature_names*in* 匹配； 然后应返回 input_features。 如果 input_features 为 None，则该方法应返回 feature_names*in*（如果已定义）否则返回长度为 n_features*in* 的 np.array(["x0", "x1", ...]) 。

答案：

In [64]:
```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.utils.validation import check_array, check_is_fitted

class StandardScalerClone(BaseEstimator, TransformerMixin):
    def __init__(self, with_mean=True):  # no *args or **kwargs!
        self.with_mean = with_mean

    def fit(self, X, y=None):  # y is required even though we don't use it
        X_orig = X
        X = check_array(X)  # checks that X is an array with finite float values
        self.mean_ = X.mean(axis=0)
        self.scale_ = X.std(axis=0)
        self.n_features_in_ = X.shape[1]  # every estimator stores this in fit()
        if hasattr(X_orig, "columns"):
            self.feature_names_in_ = np.array(X_orig.columns, dtype=object)
        return self  # always return self!

    def transform(self, X):
        check_is_fitted(self)  # looks for learned attributes (with trailing _)
        X = check_array(X)
        if self.n_features_in_ != X.shape[1]:
            raise ValueError("Unexpected number of features")
        if self.with_mean:
            X = X - self.mean_
        return X / self.scale_

    def inverse_transform(self, X):
        check_is_fitted(self)
        X = check_array(X)
        if self.n_features_in_ != X.shape[1]:
            raise ValueError("Unexpected number of features")
        X = X * self.scale_
        return X + self.mean_ if self.with_mean else X

    def get_feature_names_out(self, input_features=None):
        if input_features is None:
            return getattr(self, "feature_names_in_",
                           [f"x{i}" for i in range(self.n_features_in_)])
        else:
            if len(input_features) != self.n_features_in_:
                raise ValueError("Invalid number of features")
            if hasattr(self, "feature_names_in_") and not np.all(
                self.feature_names_in_ == input_features
            ):
                raise ValueError("input_features ≠ feature_names_in_")
            return input_features
```

让我们测试一下我们的自定义转换器：

In [65]:

```
from sklearn.utils.estimator_checks import check_estimator

check_estimator(StandardScalerClone())
```

没有错误，这是一个很好的开始，我们符合 Scikit-Learn API。

现在让我们确保转换按预期进行：

In [66]:

```
np.random.seed(42)
X = np.random.rand(1000, 3)

scaler = StandardScalerClone()
X_scaled = scaler.fit_transform(X)

assert np.allclose(X_scaled, (X - X.mean(axis=0)) / X.std(axis=0))
```

设置 with_mean=False 怎么样？

In [67]:

```
scaler = StandardScalerClone(with_mean=False)
X_scaled_uncentered = scaler.fit_transform(X)

assert np.allclose(X_scaled_uncentered, X / X.std(axis=0))
```

inverse 有效吗？

In [68]:

```
scaler = StandardScalerClone()
X_back = scaler.inverse_transform(scaler.fit_transform(X))

assert np.allclose(X, X_back)
```

特征名称怎么样？

In [69]:

```
assert np.all(scaler.get_feature_names_out() == ["x0", "x1", "x2"])
assert np.all(scaler.get_feature_names_out(["a", "b", "c"]) == ["a", "b", "c"])
```

如果我们 fit 一个 DataFrame，这个特征是否正常？

In [70]:

```
df = pd.DataFrame({"a": np.random.rand(100), "b": np.random.rand(100)})
scaler = StandardScalerClone()
X_scaled = scaler.fit_transform(df)

assert np.all(scaler.feature_names_in_ == ["a", "b"])
assert np.all(scaler.get_feature_names_out() == ["a", "b"])
```