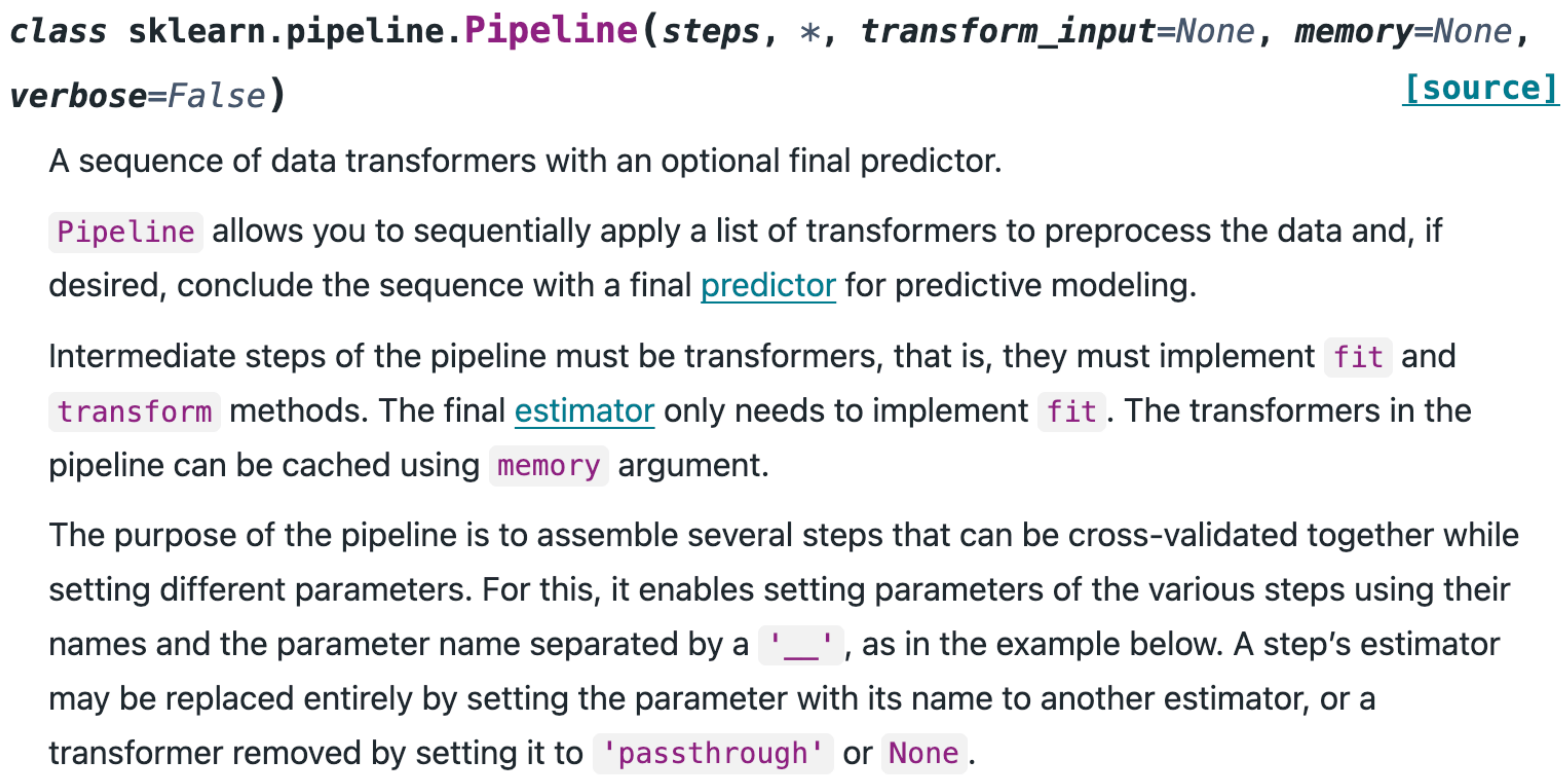
Chapter 10. Pipeline

SKLearn - Pipeline

- Scikit-learn 라이브러리의 단계별 모델들을 묶어 Pipeline을 구성할 수 있음.
- Pipeline은 fit/transform 메서드를 가진 객체들만 처리 가능
    - 사용가능: Imputere, Scaler, Encoder, PCA, Regression
    - 사용불가능: dropna, fallna 등
- 전처리, 모델학습 및 활용 과정을 묶어서 하나의 객체처럼 다룰 수 있음.

Pipeline



class `sklearn.pipeline.Pipeline(`*steps*, ***, *transform_input=None*, *memory=None*, *verbose=False*`)` [source]

A sequence of data transformers with an optional final predictor.

`Pipeline` allows you to sequentially apply a list of transformers to preprocess the data and, if desired, conclude the sequence with a final predictor for predictive modeling.

Intermediate steps of the pipeline must be transformers, that is, they must implement `fit` and `transform` methods. The final estimator only needs to implement `fit`. The transformers in the pipeline can be cached using `memory` argument.

The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters. For this, it enables setting parameters of the various steps using their names and the parameter name separated by a `'__'`, as in the example below. A step's estimator may be replaced entirely by setting the parameter with its name to another estimator, or a transformer removed by setting it to `'passthrough'` or `None`.

기본 모듈 로드

```python
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

데이터 분할 및 파이프라인 구성

```
X_train, X_test, y_train, y_test = train_test_split(df[['Age', 'Fare']]
                                                    ,df['Survived'],
                                                    test_size=0.2,
                                                    random_state=42)
```

```
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),    # 결측값 평균 대체
    ('scaler', StandardScaler()),                    # 정규화
    ('model', LogisticRegression())                  # 분류기
])
```

- 파이프라인 : List of (name of step, estimator) tuples 필요
- 파이프라인 내 각 단계를 순차적으로 진행

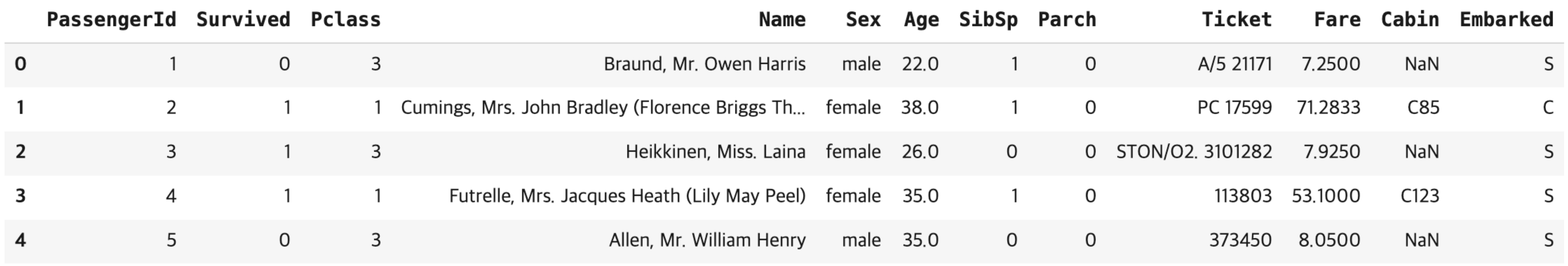
학습 이후는?

- 일반 scikit learn 모델처럼 활용함.
- pipeline.fit(X_train, y_train)
- pipeline.score(X_test, y_test)
- pipeline.predict(X_test)
- pipeline.predict_proba(X_test)

컬럼에 따른 전처리

- 변수 유형(범주형, 연속형)에 따라서 전처리 기법이 달라질 수 있음.
- 예시1) 범주형 변수에 대한 누락값 대체로 평균값을 쓰는 경우
- 예시2) 연속형 변수에 one-hot encoding을 쓰는 경우
- pipeline은 순차적으로 동작 >> Column transformer를 통해 컬럼별 전처리

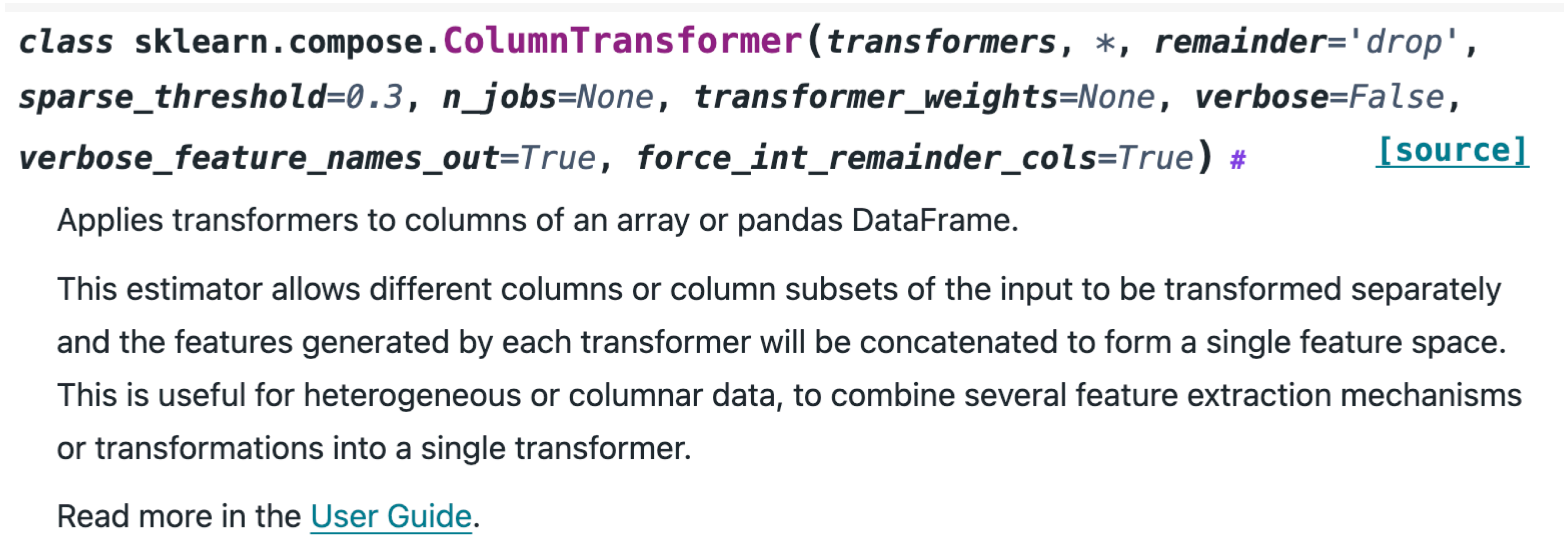
범주형/연속형 변수



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

- 범주형: Survived, Pclass, Sex, Embarked
- 연속형: Age, SibSp, Parch, Fare
- 제외 : PassengerId, Name, Ticket, Cabin

Column transformer



```
class sklearn.compose.ColumnTransformer(transformers, *, remainder='drop', sparse_threshold=0.3, n_jobs=None, transformer_weights=None, verbose=False, verbose_feature_names_out=True, force_int_remainder_cols=True) #
```

[source]

Applies transformers to columns of an array or pandas DataFrame.

This estimator allows different columns or column subsets of the input to be transformed separately and the features generated by each transformer will be concatenated to form a single feature space. This is useful for heterogeneous or columnar data, to combine several feature extraction mechanisms or transformations into a single transformer.

Read more in the User Guide.

개별 컬럼에 대한 전처리 과정 구성

```python
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder

numeric_features = ['Age', 'Fare']
numeric_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_features = ['Sex', 'Embarked']
categorical_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='constant', fill_value=-1)),
    ('ordinal', OrdinalEncoder())
])
```

Column transformer 적용

```
preprocessor = ColumnTransformer([
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])
```

```
from sklearn.ensemble import RandomForestClassifier
pipeline=Pipeline([
    ('preprocessor', preprocessor),
    ('model', RandomForestClassifier())
])
```

Column transformer 적용

```
train_test_split(df.drop(['PassengerId','Survived','Name','Ticket','Cabin'],axis=1),
                 df.Survived,
                 test_size=0.2, random_state=0)
```

```
pipeline.fit(X_train, y_train)
pipeline.score(X_test, y_test)
```

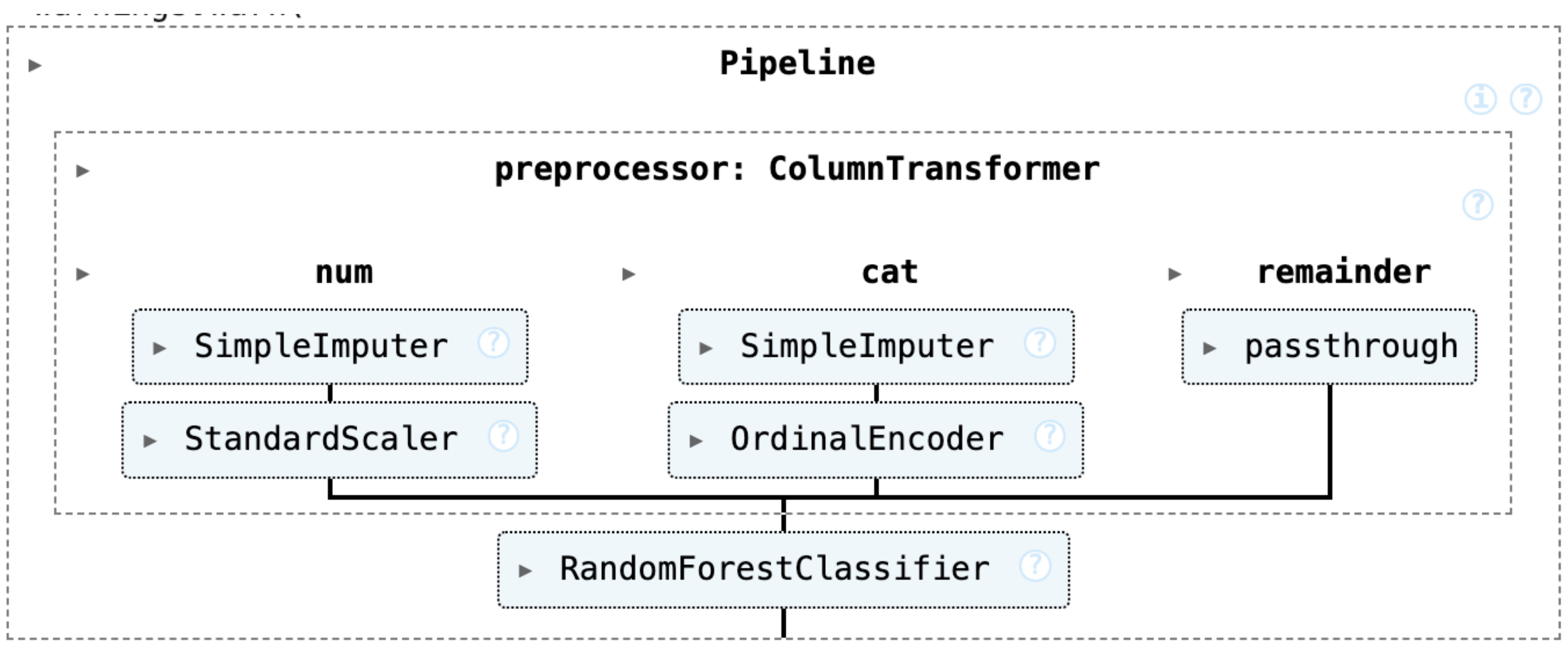
차이점은?

```
preprocessor1 = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ],
    remainder='passthrough'
)
```

- Remainder = ‘drop’, 주어진 feature 외의 컬럼들은 drop
- Remainder = ‘passthrough’, 주어지지 않은 feature들 그대로 사용

Pipeline

- 객체 호출시 아래와 같이 구성도 확인가능



Feature Union

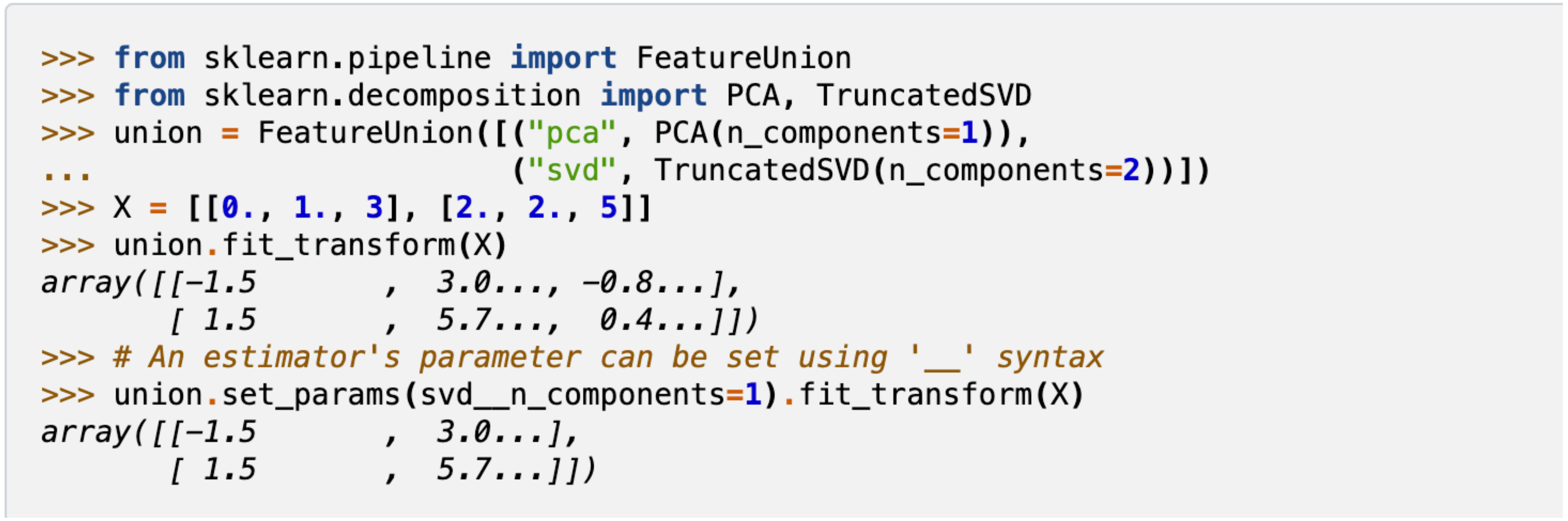
- 동일한 컬럼에 대해서 서로다른 전처리 방법을 병렬적으로 적용한다면?

```
numeric_features = ['Age', 'Fare']
numeric_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

numeric_features2 =['Age']
numeric_transformer2 = Pipeline([
    ('scaler',MinMaxScaler())
])
```

Feature Union

- 모든 컬럼에 대해서 다양한 transformer를 적용하고 싶은 경우
- e.g., PCA 추출 + polynomial feature



```
>>> from sklearn.pipeline import FeatureUnion
>>> from sklearn.decomposition import PCA, TruncatedSVD
>>> union = FeatureUnion([("pca", PCA(n_components=1)),
...                       ("svd", TruncatedSVD(n_components=2))])
>>> X = [[0., 1., 3], [2., 2., 5]]
>>> union.fit_transform(X)
array([[-1.5       ,  3.0..., -0.8...],
       [ 1.5       ,  5.7...,  0.4...]])
>>> # An estimator's parameter can be set using '__' syntax
>>> union.set_params(svd__n_components=1).fit_transform(X)
array([[-1.5       ,  3.0...],
       [ 1.5       ,  5.7...]])
```