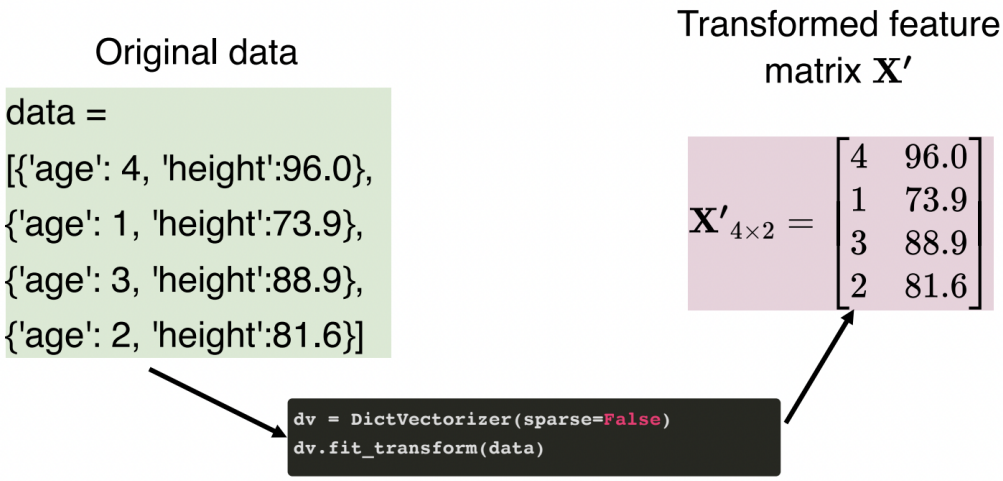
Week 2

Feature Extraction

💡 sklearn.feature_extraction

DictVectorizer

Converts lists of mappings of feature name and feature value, into a matrix.



FeatureHasher

- **High-speed, low-memory vectorizer** that uses feature hashing technique.
- Instead of building a hash table of the features, as the vectorizers do, it **applies a hash function to the features to determine their column index in sample matrices** directly.

- This results in **increased speed and reduced memory usage**, at the expense of inspectability; the hasher does not remember what the input features looked like and has **no inverse_transform** method.
- The **output is scipy.sparse matrix**.

Data Cleaning

Handling missing values

sklearn.impute

SimpleImputer

Fills missing values with one of the following strategies:
'mean', 'median', 'most_frequent' and 'constant'.

KNNImputer

- Uses the k-nearest neighbours approach to fill missing values in a dataset.
- The missing value of an attribute in a specific example is filled with the mean value of the same attribute of n_neighbors closest neighbours.
- The nearest neighbours are decided based on Euclidean distance.

Feature Scaling

StandardScaler

Transforms the original features vector into a new feature vector using following formula:

$$x' = \frac{x - \mu}{\sigma}$$

MinMaxScaler

It transforms the original feature vector into new feature vector so that all values fall within range [0, 1] using the formula:

$$x' = \frac{x - x.min}{x.max - x.min}$$

MaxAbsScaler

It transforms the original features vector into new feature vector so that all values fall within range [−1, 1] using the formula:

$$x' = \frac{x}{\text{Max absolute value}}$$

Feature transformations

Function transformation

Constructs transformed features by applying a user defined function.

```
ft = FunctionTransformer(numpy.log2)
ft.fit_transform(X)
```

Polynomial transformation

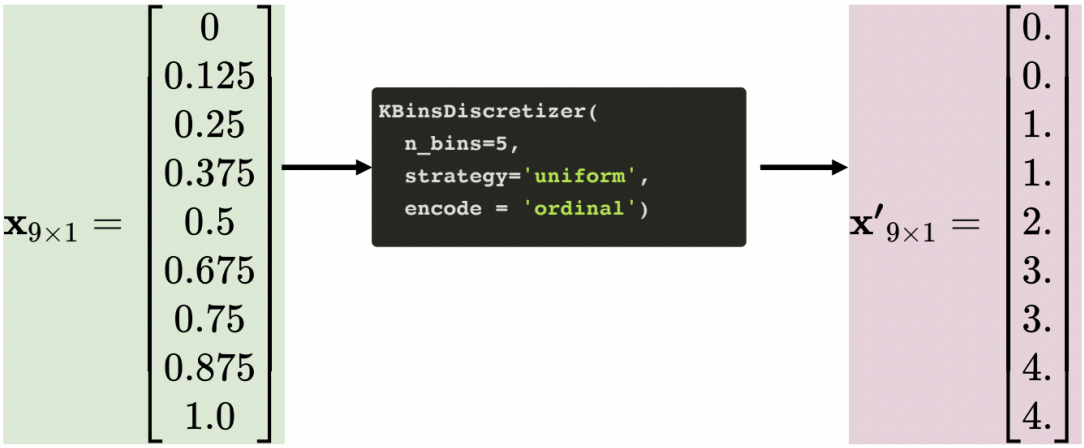
Generates a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree.

```
pf = PolynomialFeatures(degree=2)
pf.fit_transform(X)
```

Discretization

KBinsDiscretizer

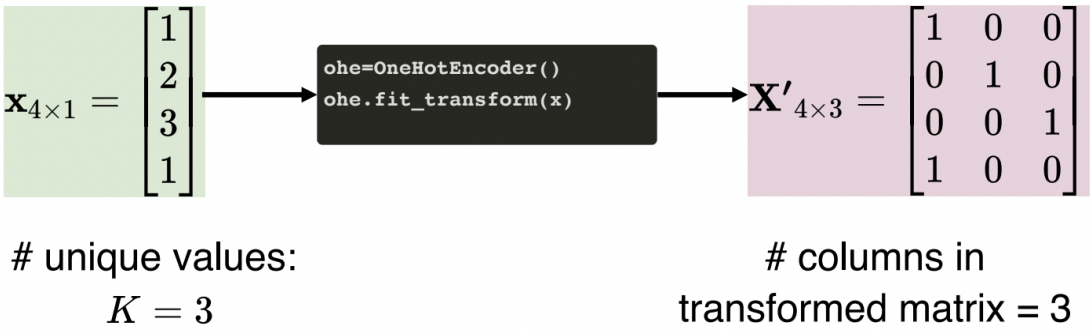
- Divides a continuous variable into bins.
- One hot encoding or ordinal encoding is further applied to the bin labels.



Categorical transformers

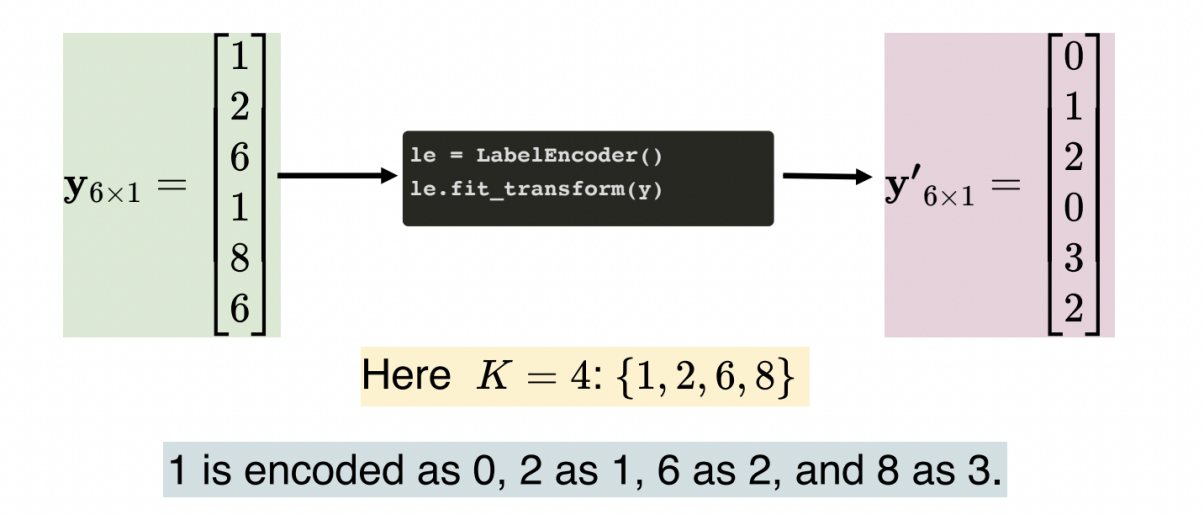
OneHotEncoder

- Encodes categorical feature or label as a one-hot numeric array.
- Creates one binary column for each of k unique values.
- Exactly one column has 1 in it and rest have 0.



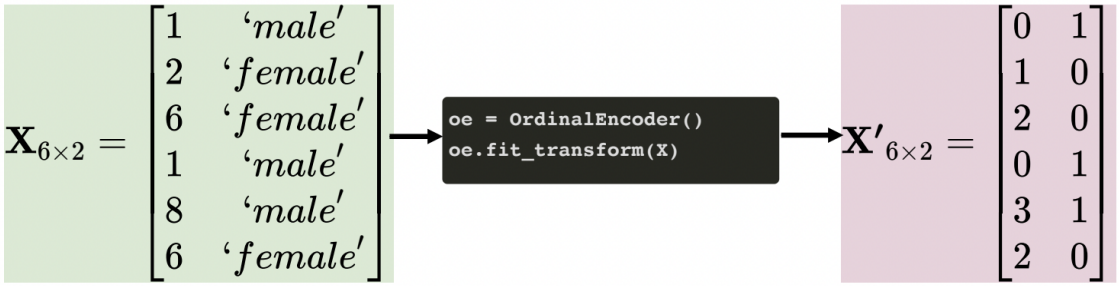
LabelEncoder

Encodes target labels with value between 0 and k-1, where k is number of distinct values.



OrdinalEncoder

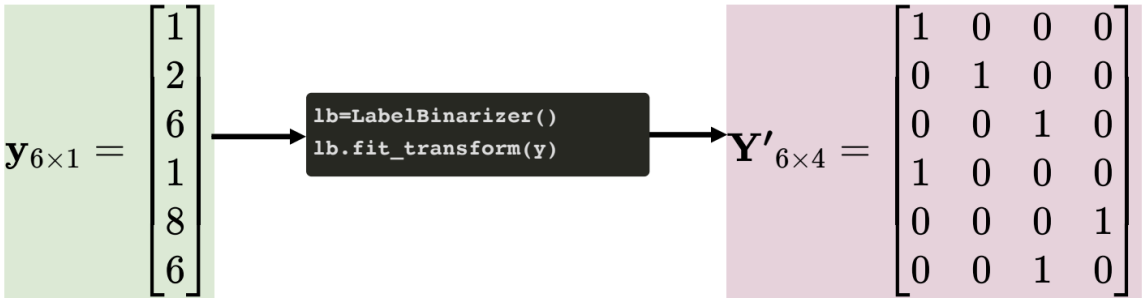
Encodes categorical features with value between 0 and K − 1, where K is number of distinct values.



OrdinalEncoder can operate multi dimensional data, while LabelEncoder can transform only 1D data.

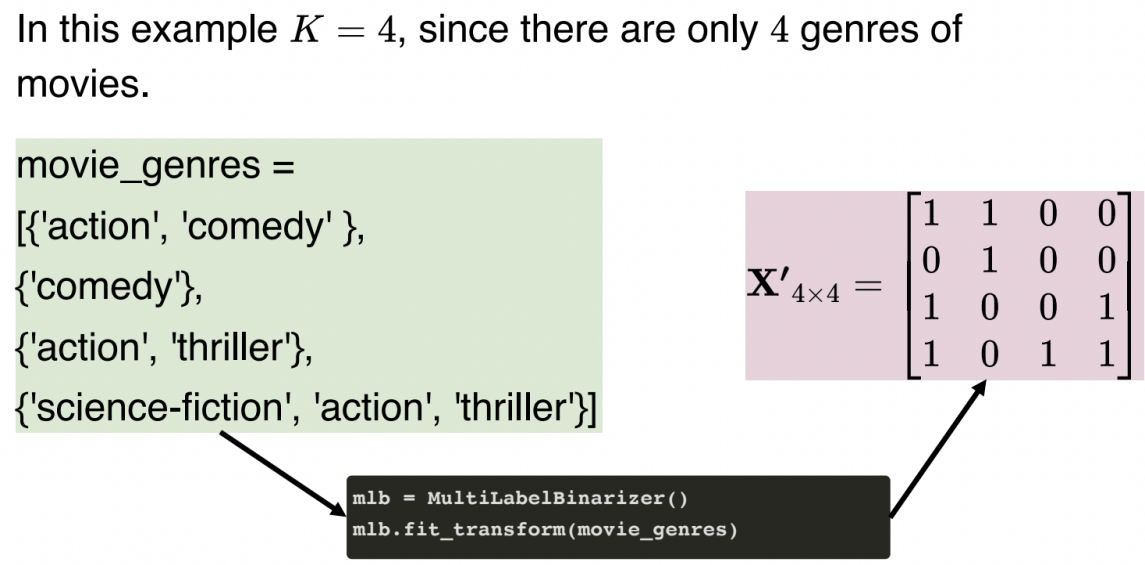
LabelBinarizer

- Several regression and binary classification can be extended to multi-class setup in one-vs-all fashion.
- This involves training a single regressor or classifier per class.
- For this, we need to convert multi-class labels to binary labels, and LabelBinarizer performs this task.



MultiLabelBinarizer

Encodes categorical features with value between 0 and K − 1, where K is number of classes.



add_dummy_feature

Augments dataset with a column vector, each value in the column vector is 1.

Composite Transformer

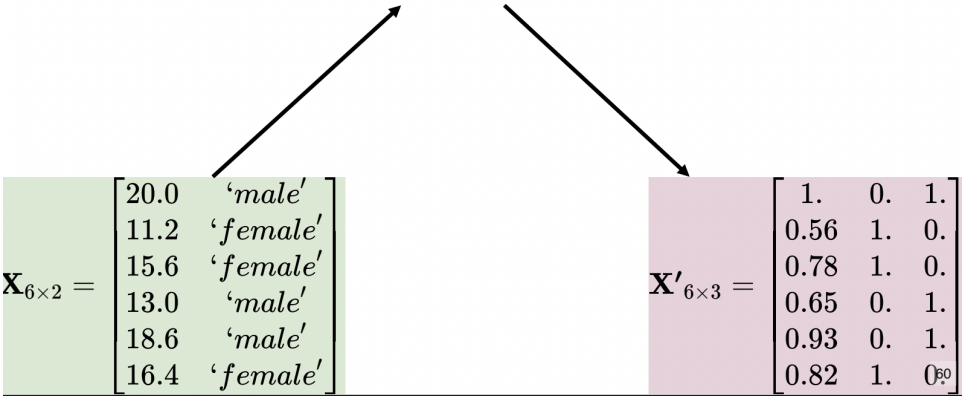
sklearn.compose

Column Transformer

- It applies a set of transformers to columns of an array or pandas.DataFrame, concatenates the transformed outputs from different transformers into a single matrix.
- It is useful for transforming heterogenous data by applying different transformers to separate subsets of features.
- It combines different feature selection mechanisms and transformation into a single transformer object.

In this example, lets apply MaxAbsScaler on the numeric column and OneHotEncoder on categorical column.

```
column_trans = ColumnTransformer(
    [('ageScaler', MaxAbsScaler(), [0]),
     ('genderEncoder', OneHotEncoder(dtype='int'), [1])],
    remainder='drop', verbose_feature_names_out=False)
column_trans.fit_transform(X)
```



TransformedTargetRegressor

- Transforms the target variable y before fitting a regression model.
- The predicted values are mapped back to the original space via an inverse transform.
- TransformedTargetRegressor takes regressor and transformer to be applied to the target variable as arguments.

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.compose import TransformedTargetRegressor
tt = TransformedTargetRegressor(regressor=LinearRegression(),
                                func=np.log, inverse_func=np.exp)
X = np.arange(4).reshape(-1, 1)
```

```
y = np.exp(2 * X).ravel()
tt.fit(X, y)
```

Feature Selection

💡 sklearn.feature_selection

Filter-based methods

VarianceThreshold

- Removes all features with variance below a certain threshold, as specified by the user, from input feature matrix.
- By default removes a feature which has same value, i.e. zero variance.

SelectKBest, SelectPercentile and GenericUnivariateSelect work on common univariate statistical tests:

- **SelectFpr** selects features based on a false positive rate test.
- **SelectFdr** selects features based on an estimated false discovery rate.
- **SelectFwe** selects featurfes based on family-wise error rate.

Each API need a scoring function to score each feature. Three classes of scoring functions are proposed:

- **Mutual Information (MI):** mutual_info_regression, mutual_info_classif
- **Chi-square:** chi2
- **F-statistics:** f_regression, f_classif

MI and F-statistics can be used in both classification and regression problems.

Chi-square can be used only in classification problems.

Mutual information (MI)

- Measures dependency between two variables.
- It returns a non-negative value.
  - MI = 0 for independent variables.
  - Higher MI indicates higher dependency.

Chi-square

- Measures dependence between two variables.
- Computes chi-square stats between non-negative feature (boolean or frequencies) and class label.
- Higher chi-square values indicates that the features and labels are likely to be correlated.

MI and chi-squared feature selection is recommended for sparse data.

SelectKBest

Removes all but the k highest scoring features

```
skb = SelectKBest(chi2, k=20)
X_new = skb.fit_transform(X, y)
```

SelectPercentile

Removes all but a user-specified highest scoring percentage of features

```
sp = SelectPercentile(chi2, percentile=20)
X_new = sp.fit_transform(X, y)
```

GenericUnivariateSelect

Performs univariate feature selection with a configurable strategy, which can be found via hyper-parameter search.

```
transformer = GenericUnivariateSelect(chi2, mode='k_best', param=20) #percentile mode
 is default
X_new = transformer.fit_transform(X, y)
```

Wrapper-based methods

Unlike filter based methods, wrapper based methods use estimator class rather than a scoring function.

Recursive Feature Elimination (RFE)

- Uses an estimator to recursively remove features.
- Initially fits an estimator on all features.
- Obtains feature importance from the estimator and removes the least important feature.
- Repeats the process by removing features one by one, until desired number of features are obtained.

Recursive Feature Elimination Cross Validation (RFECV)

- Use if we do not want to specify the desired number of features in RFE .
- It performs RFE in a cross-validation loop to find the optimal number of features.

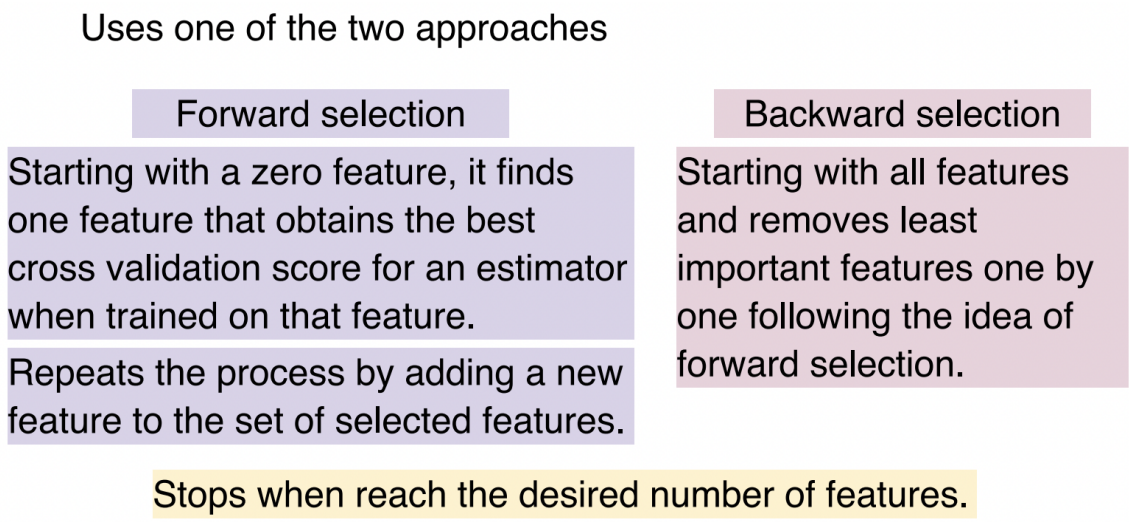
SelectFromModel

- Selects desired number of important features (as specified with max_features parameter) above certain threshold of feature importance as obtained from the trained estimator.
- The feature importance is obtained via coef_, feature_importances_ or an importance_getter callable from the trained estimator.
- The feature importance threshold can be specified either numerically or through string argument based on built-in heuristics such as 'mean', 'median' and float multiples of these like '0.1*mean'.

```
clf = LinearSVC(C=0.01, penalty="l1", dual=False)
clf = clf.fit(X, y)
clf.coef_
model = SelectFromModel(clf, prefit=True)
X_new = model.transform(X)
```

SequentialFeatureSelector

Performs feature selection by selecting or deselecting features one by one in a greedy manner.



- The direction parameter controls whether forward or backward SFS is used.
- In general, forward and backward selection do not yield equivalent results.
- Select the direction that is efficient for the required number of selected features.
- SFS does not require the underlying model to expose a coef_ or feature_importances_ attributes unlike in RFE and SelectFromModel.
- SFS may be slower than RFE and SelectFromModel as it needs to evaluate more models compared to the other two approaches.

Dimensionality Reduction

💡 sklearn.decomposition

Principal Component Analysis (PCA)

💡 sklearn.decomposition.PCA

- PCA, is a linear dimensionality reduction technique.
- It uses singular value decomposition (SVD) to project the feature matrix or data to a lower dimensional space.
- The first principle component (PC) is in the direction of maximum variance in the data.

- It captures bulk of the variance in the data.
- The subsequent PCs are orthogonal to the first PC and gradually capture lesser and lesser variance in the data.
- We can select first PCs such that we are able to capture the desired variance in the data.

Chaining Transformers

- The module provides utilities to build a composite estimator, as a chain of transformers and
  estimators.
- There are two classes: (i) Pipeline and (ii) FeatureUnion.

Class	Usage
Pipeline	Constructs a chain of multiple transformers to execute a fixed sequence of steps in data preprocessing and modelling.
FeatureUnion	Combines output from several transformer objects by creating a new transformer from them.

Pipeline

sklearn.pipeline.Pipeline

- Sequentially apply a list of transformers and estimators.
- 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 purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters.

Creating Pipelines

Two ways to create a pipeline object.

`Pipeline()`

- It takes a list of `('estimatorName', estimator(...))` tuples.
- The pipeline object exposes interface of the last step.

```
estimators = [
  ('simpleImputer', SimpleImputer()),
  ('standardScaler', StandardScaler()),
]
pipe = Pipeline(steps=estimators)
```

`make_pipeline`

- It takes a number of estimator objects only.

```
pipe = make_pipeline(SimpleImputer(),
                     StandardScaler())
```

Accessing individual steps in Pipeline

```
estimators = [
  ('simpleImputer', SimpleImputer()),
  ('pca', PCA()),
  ('regressor', LinearRegression())
]
pipe = Pipeline(steps=estimators)
```

Total # steps: 3

1. SimpleImputer
2. PCA
3. LinearRegression

The second estimator can be accessed in following 4 ways:

- `pipe.named_steps.pca`
- `pipe.steps[1]`
- `pipe[1]`
- `pipe['pca']`

Accessing parameters of each step in Pipeline

```
estimators = [
('simpleImputer', SimpleImputer()),
('pca', PCA()),
('regressor', LinearRegression())
]
pipe = Pipeline(steps=estimators)
pipe.set_params(pca__n_components = 2) #double underscore
```

GridSearch with Pipeline

```
param_grid = dict(imputer=['passthrough',
                  SimpleImputer(),
                  KNNImputer()],
clf=[SVC(), LogisticRegression()],
clf__C=[0.1, 10, 100])
grid_search = GridSearchCV(pipe, param_grid=param_grid)
```

Caching Transformers

- Transforming data is a computationally expensive step.
- For grid search, transformers need not be applied for every parameter configuration. They can be applied only once, and the transformed data can reused.
- This can be achieved by setting memory parameter of a pipeline object.
- memory can take either location of a directory in string format or joblib.Memory object.

```
estimators = [
              ('simpleImputer', SimpleImputer()),
              ('pca', PCA(2)),
              ('regressor', LinearRegression())
              ]
pipe = Pipeline(steps=estimators, memory = '/path/to/cache/dir')
```

Feature Union

sklearn.pipeline.FeatureUnion

- Concatenates results of multiple transformer objects.
- Applies a list of transformer objects in parallel, and their outputs are concatenated side-by-side into a larger matrix.
- FeatureUnion and Pipeline can be used to create complex transformers.

Combining Transformers and Pipelines

- FeatureUnion() accepts a list of tuples.

- Each tuple is of the format:
  ('estimatorName',estimator(...))

```
num_pipeline = Pipeline([('selector',
                    ColumnTransformer([('select_first_4','passthrough',
                    slice(0,4))])),
                    ('imputer', SimpleImputer(strategy="median")),
                    ('std_scaler', StandardScaler()),
                    ])
cat_pipeline = ColumnTransformer([('label_binarizer', LabelBinarizer()[4]),
                ])
full_pipeline = FeatureUnion(transformer_list=
                [("num_pipeline", num_pipeline),
                ("cat_pipeline", cat_pipeline),])
```

Visualising Composite Transformers

```
set_config(display='diagram')
# displays HTML representation in a jupyter context
full_pipeline
```