



# Lecture 14: Scikit Learn Pipelines

# Recap

- Kmeans algorithm
- Expectation Maximization
- Centroid initialization with kmeans++
- Kmeans decision boundary
- Kmeans limitations with oblong data clusters
- Kmeans limitations with outliers
- Using Kmeans and silhouette analysis for outlier detection



**Data  
Preprocessing  
(duplicate  
removal,  
imputation,  
encoding)**

## **2) Data understanding**

**Goal:** Understand the data



## **3) Data preparation**

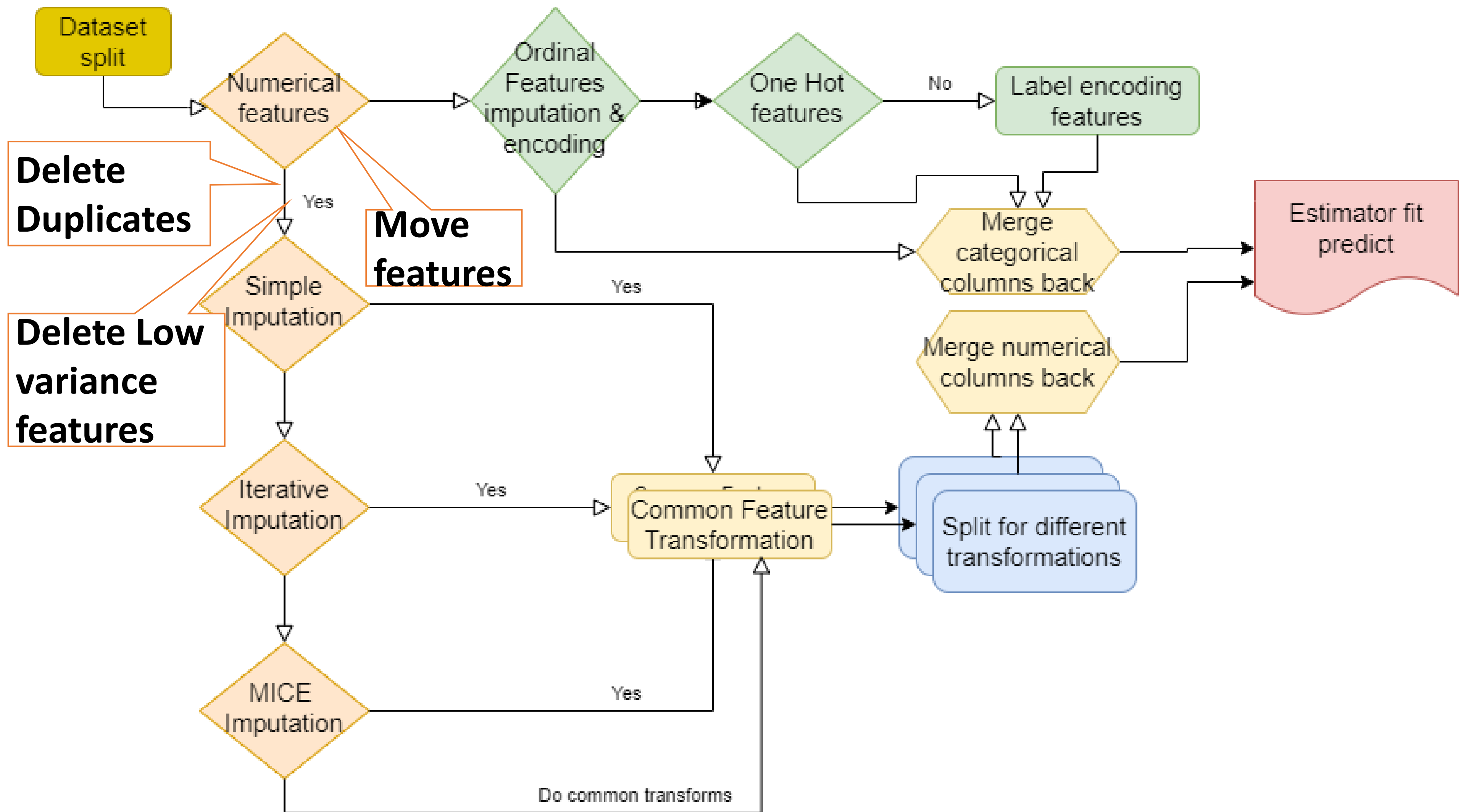
**Goal:** Prepare data for model



## **4) Modelling**

**Goal:** Build a model

- 1. Feature Selection**
- 2. Feature Transformation/Engineering**
- 3. Feature Extraction**







```
#imputer object for missing value
imputer = SimpleImputer(strategy="median")

# scales object the numerical feature
scaler = StandardScaler()

# one-hot object for the categorical features
one_hot=OneHotEncoder(handle_unknown='ignore',sparse=False)

# Applying the fit and transform methods to the Training data
imputer.fit(X_train[numeric_features])
imputed=imputer.transform(X_train[numeric_features])

scaler.fit(imputed)
scaled=scaler.transform(imputed)

one_hot.fit(X_train[categorical_features])
one_hotted_cat=one_hot.transform(X_train[categorical_features])

# Concatenating the scaled and one hot matrixes
Final_train=pd.DataFrame(np.concatenate((scaled,one_hotted_cat), axis=1))
lr.fit(Final_train, y_train)
```

- Preprocessing generally 500-1000 features
  - To be grouped in about 25-50 features
  - E.g. Diff numeric fields need diff imputation
  - Diff groups need diff encoding
  - Encoding as embedding
  - Feature Selection, Elimination, Importance, extraction etc.
  - Distribution transformation
  - Cross validation
- Output of one should be fed into next
- Dynamic code, rearranged often
- Error prone by design



```
# Predict on the test set-using the trained classifier-still need to do the transformations
X_test_filled = imputer.transform(X_test[numeric_features])
X_test_scaled = scaler.transform(X_test_filled)
X_test_one_hot = one_hot.transform(X_test[categorical_features])
X_test=pd.DataFrame(np.concatenate((X_test_scaled, X_test_one_hot), axis=1))
lr.score(X_test,y_test)
```

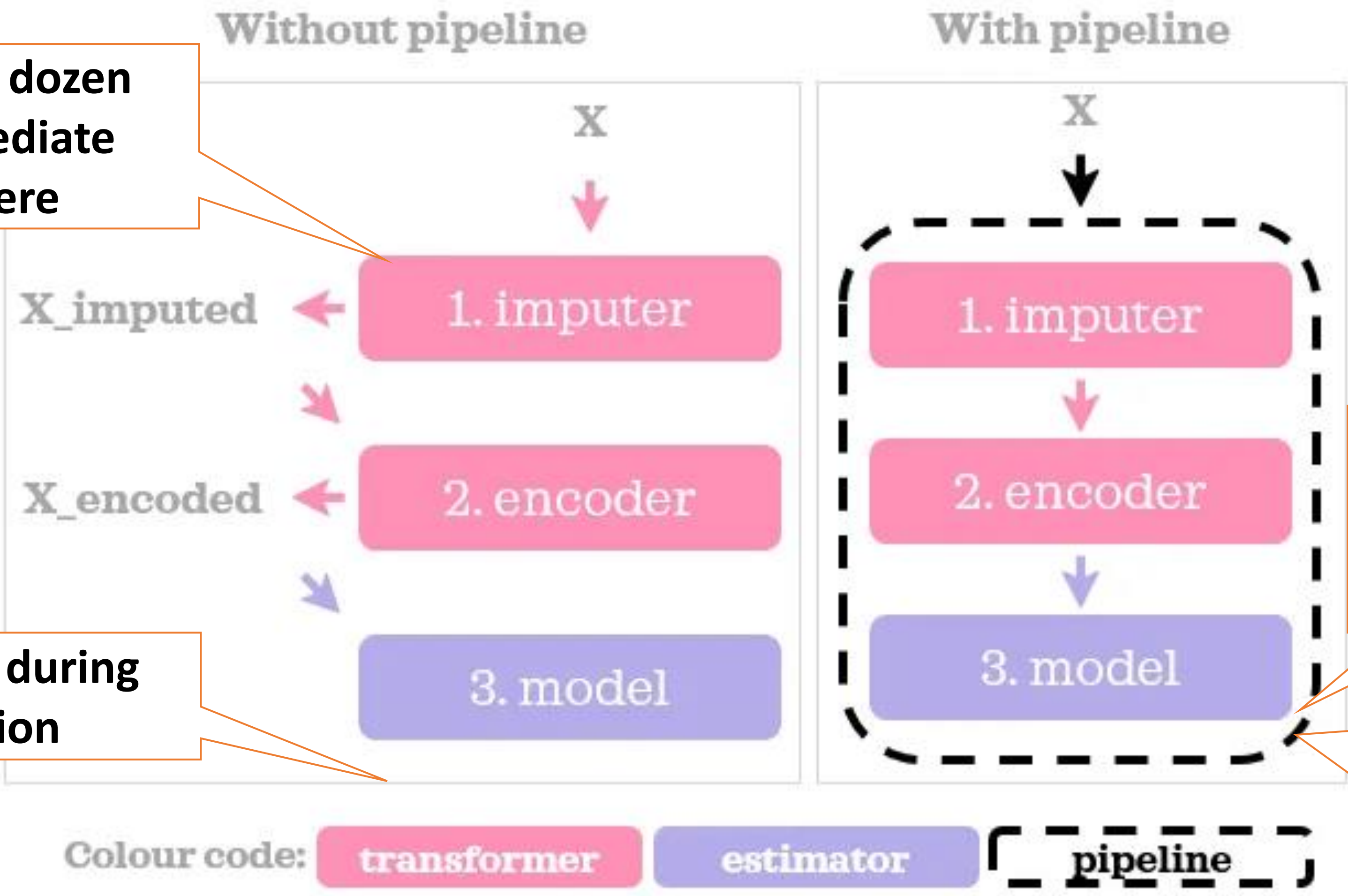
- Same order in predict
- Predict done by some other team
  - Two dozen models
  - Provide all configured imputers
  - High cost of ownership & maintenance
  - Brittle code - Touch and it breaks
  - Logistics nightmare
- Data leakage (What is it?)





Several dozen intermediate steps here

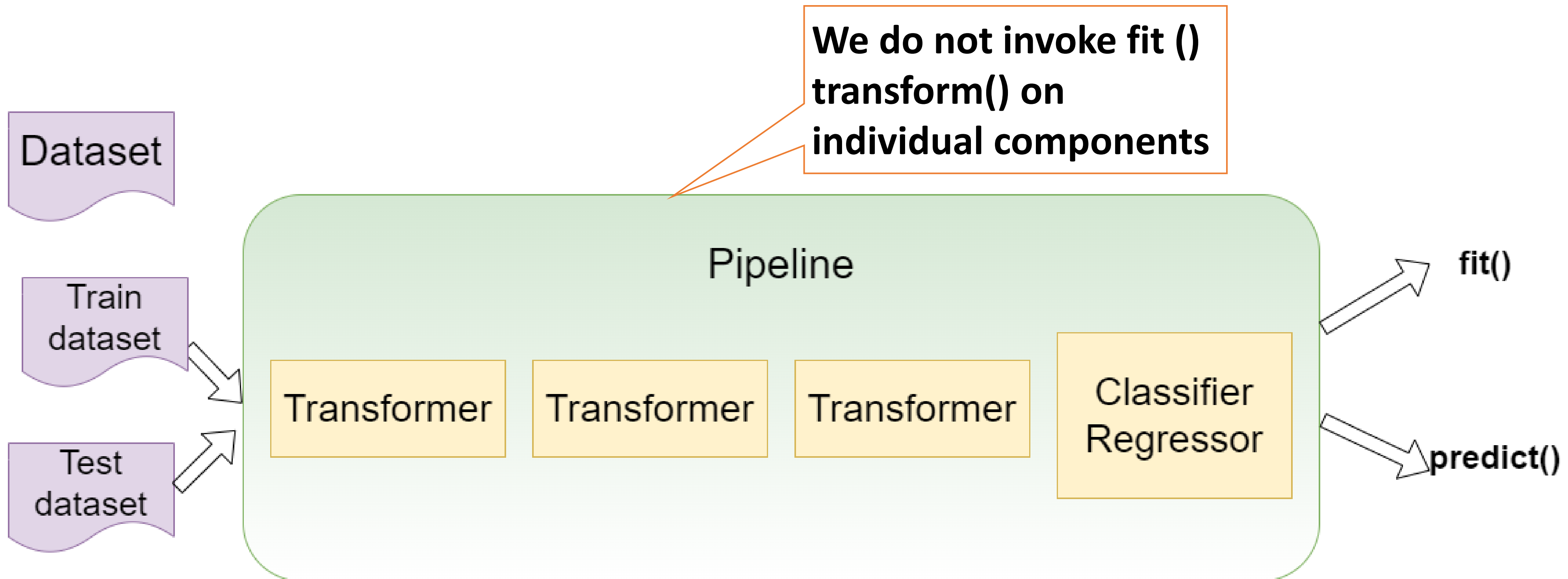
Repeat during prediction



Last component in pipeline is ML Mixin

Pickle the pipeline & share





```
BaseEstimator:

get_params(self, deep=True)
set_params(self, **params)
```

```
TransformerMixin:

def fit_transform(self, X, y=None, **fit_params):
    if y is None:
        # fit method of arity 1 (unsupervised transformation)
        return self.fit(X, **fit_params).transform(X)
    else:
        # fit method of arity 2 (supervised transformation)
        return self.fit(X, y, **fit_params).transform(X)
```

What do fit() and transform() do?

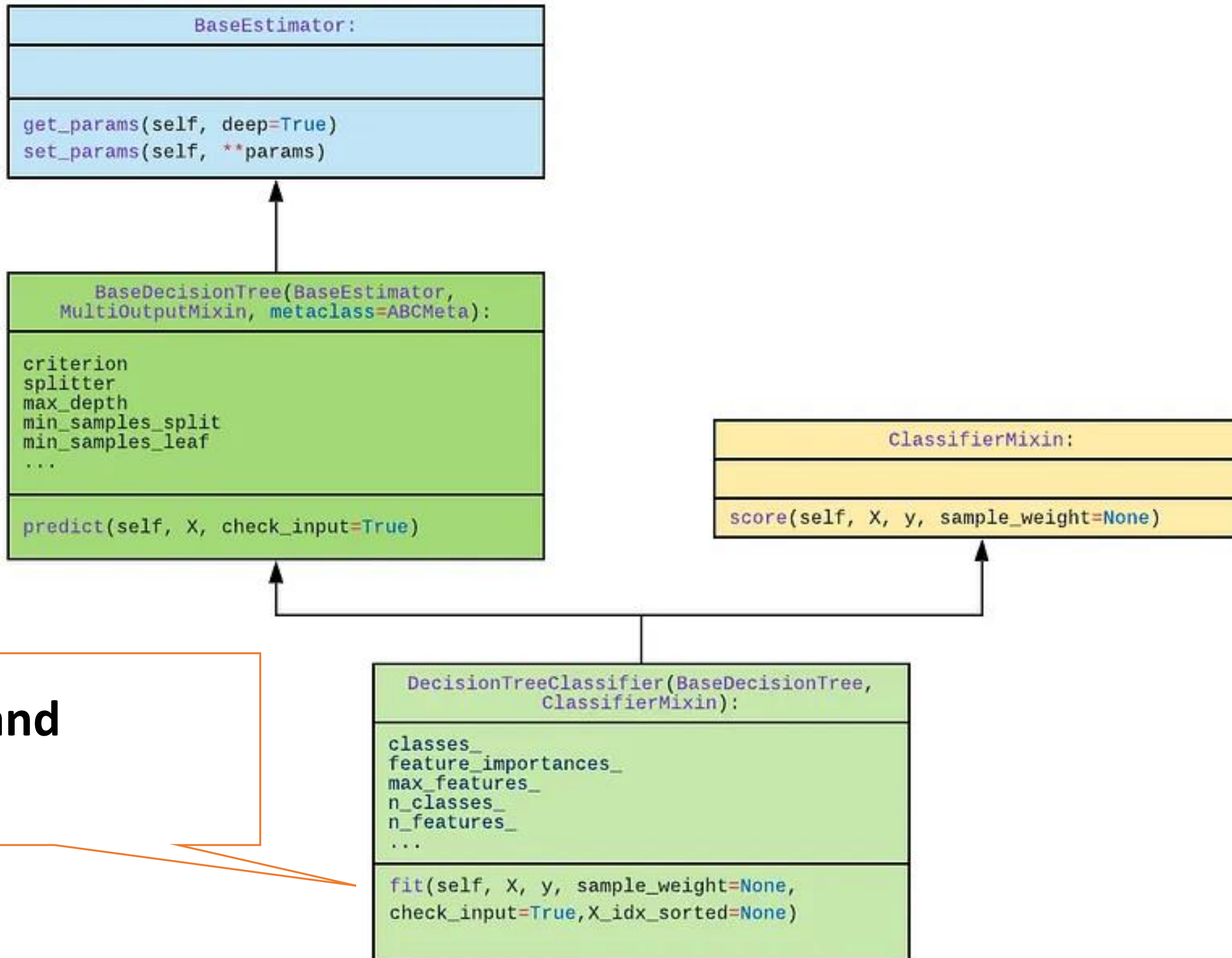
```
StandardScaler(BaseEstimator,
TransformerMixin):

scale_
mean_
var_
n_samples_seen_

fit(self, X, y=None):
transform(self, X):
```

fit\_transform() combines both fit() and transform()





Only fit() and  
predict()

# fit(), fit\_transform() and predict()

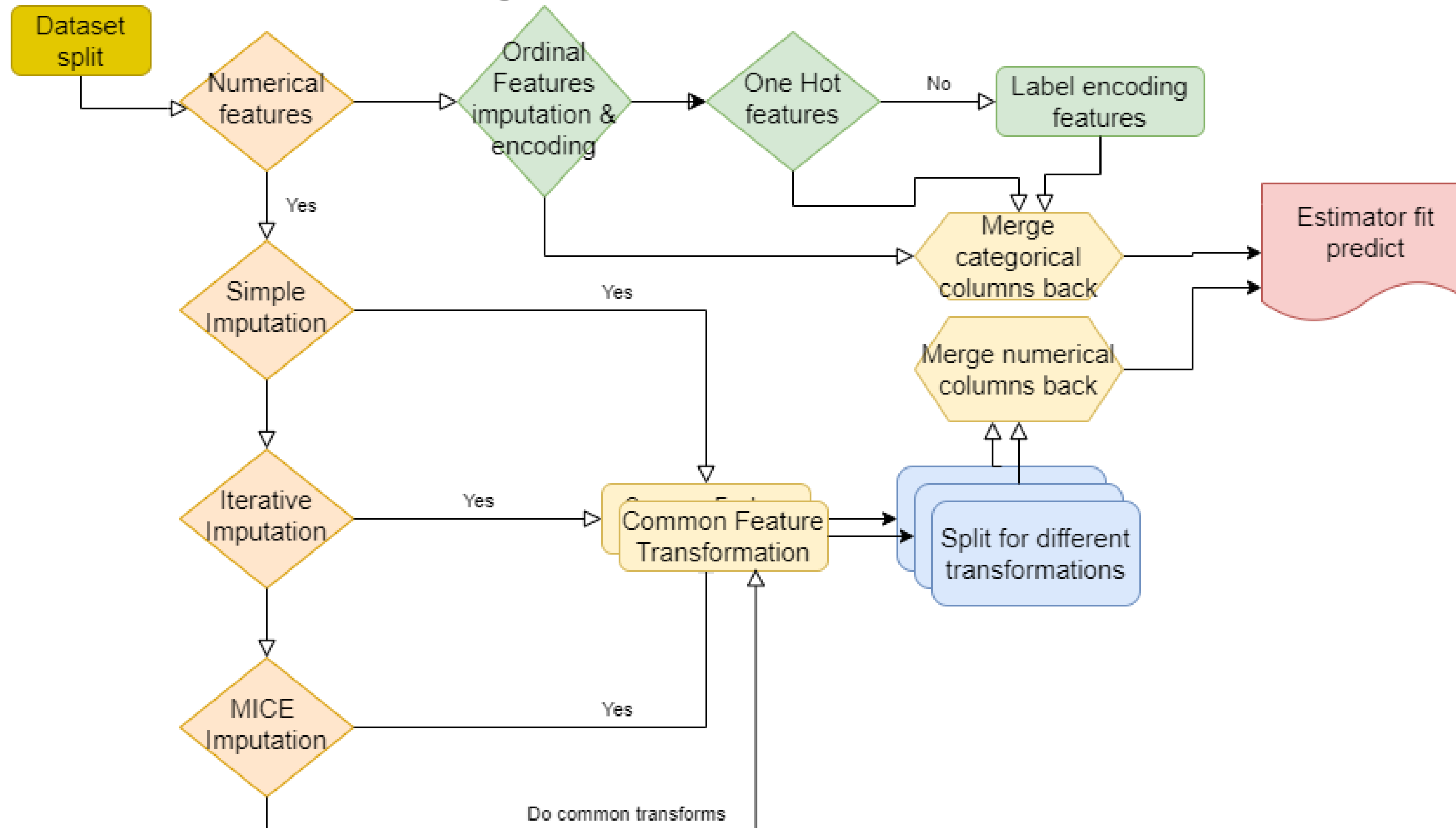
- Always configure pipeline with
  - Last component as Classifier/Regressor/Cluster Mixin
  - Other components are Transformers
- Invoke fit() on pipeline during training
  - Sklearn invokes fit\_transform() on all but last component
  - Sklearn invokes fit() on last component
- Invoke predict() on pipeline during test
  - Sklearn invokes transform() on all but last component
  - Sklearn invokes predict on last component
- Sklearn relies on python duck typing than class hierarchy



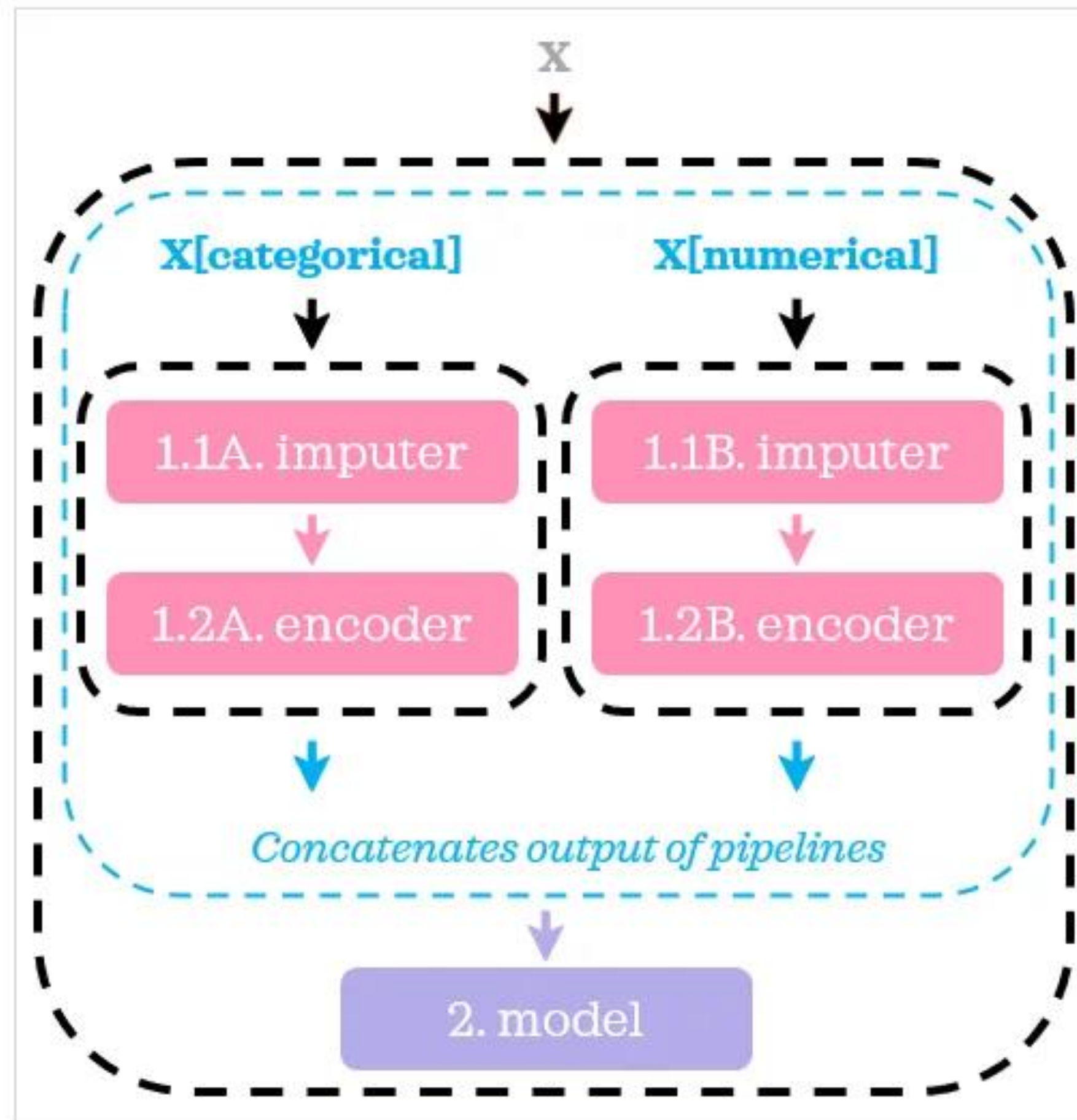


# Mixture Model clustering

- Apply different transformations on numerical & categorical features. Then merge back







Colour  
code:

**transformer**

**estimator**

**pipeline**

**column  
transformer**

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

numeric_transformer = Pipeline(steps=[
    ('meanimputer', SimpleImputer(strategy='mean')),
    ('stdscaler', StandardScaler())
])

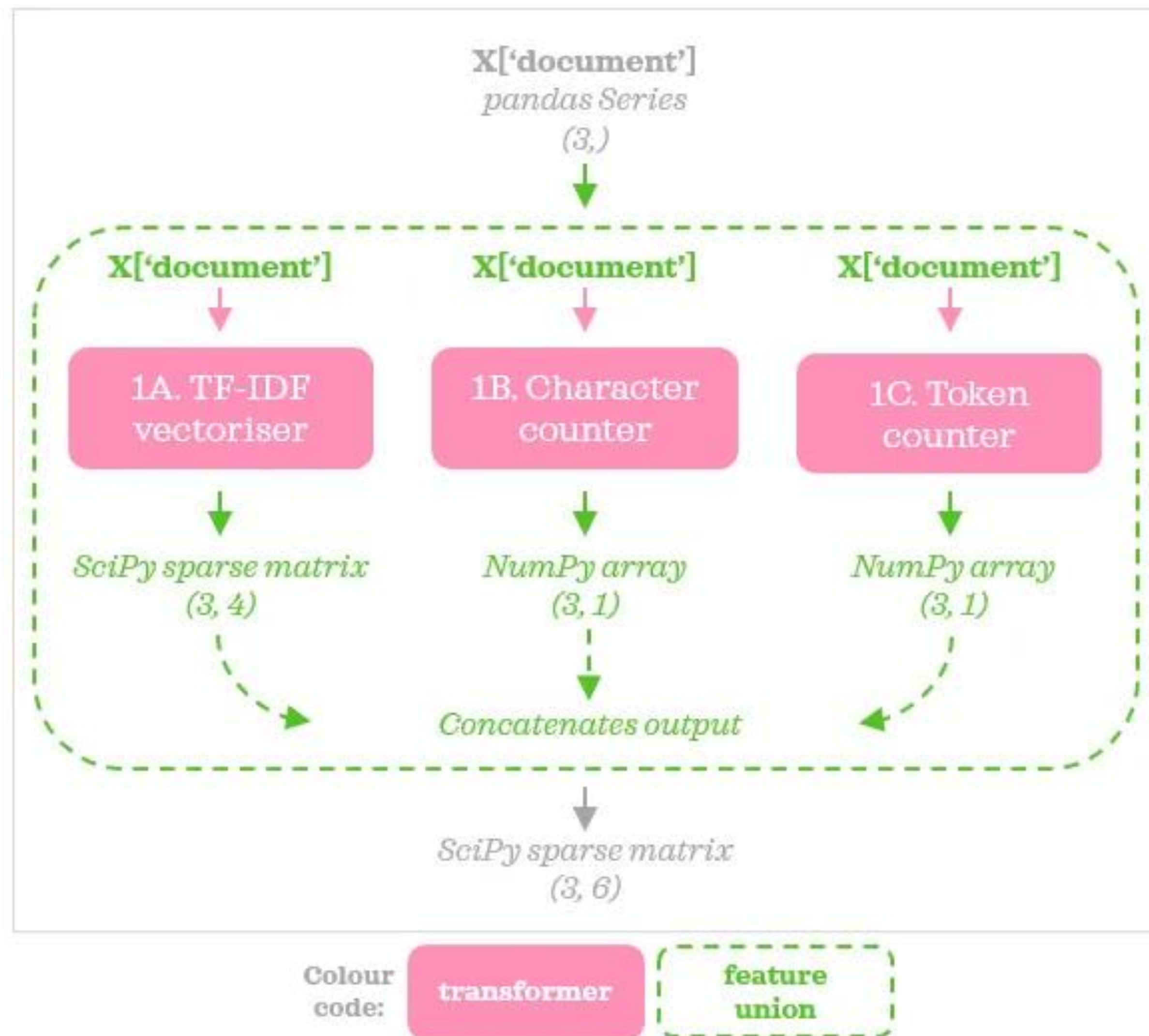
categorical_transformer = Pipeline(steps=[
    ('onehotenc', OneHotEncoder(handle_unknown='ignore'))
])

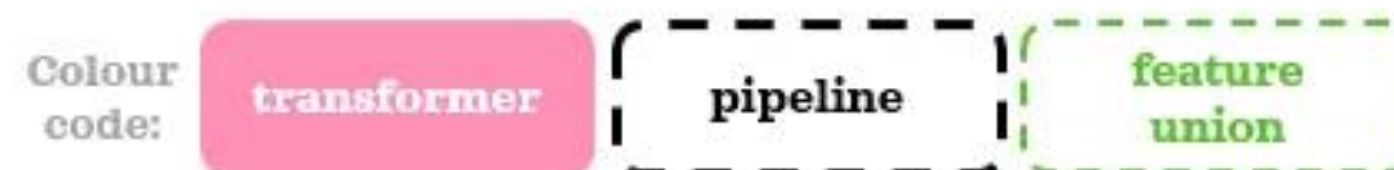
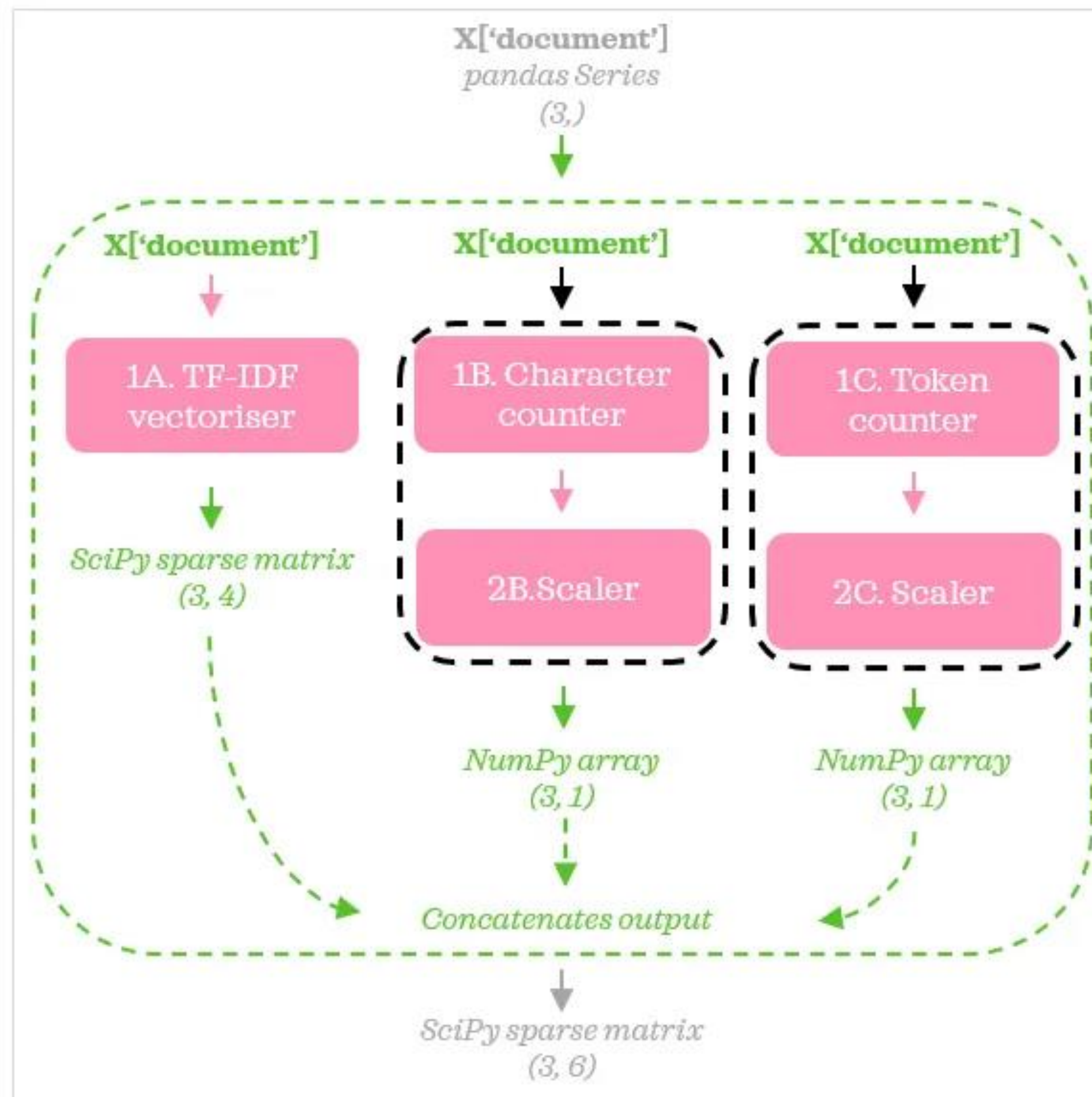
col_transformer = ColumnTransformer(transformers=[('numeric_processing', numeric_transformer,
                                                  numeric_features),
                                                  ('categorical_processing', categorical_transformer,
                                                  categorical_features)])

pipeline = Pipeline([
    ('transform_column', col_transformer),
    ('logistics', LogisticRegression())
])

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









# GridSearch with Pipeline

```
from sklearn.model_selection import GridSearchCV

X_train, X_test, y_train, y_test = train_test_split(df, y, random_state=42)

#create new a knn model
knn = KNeighborsClassifier()
#create a dictionary of all values we want to test for n_neighbors
param_grid = {"n_neighbors": np.arange(1, 25),
              "weights": ["uniform", "distance"]}

loocv = LeaveOneOut()

#use gridsearch to test all values for n_neighbors
knn_gscv = GridSearchCV(knn, param_grid, cv=loocv)
#fit model to data
knn_gscv.fit(X_train, y_train)
```

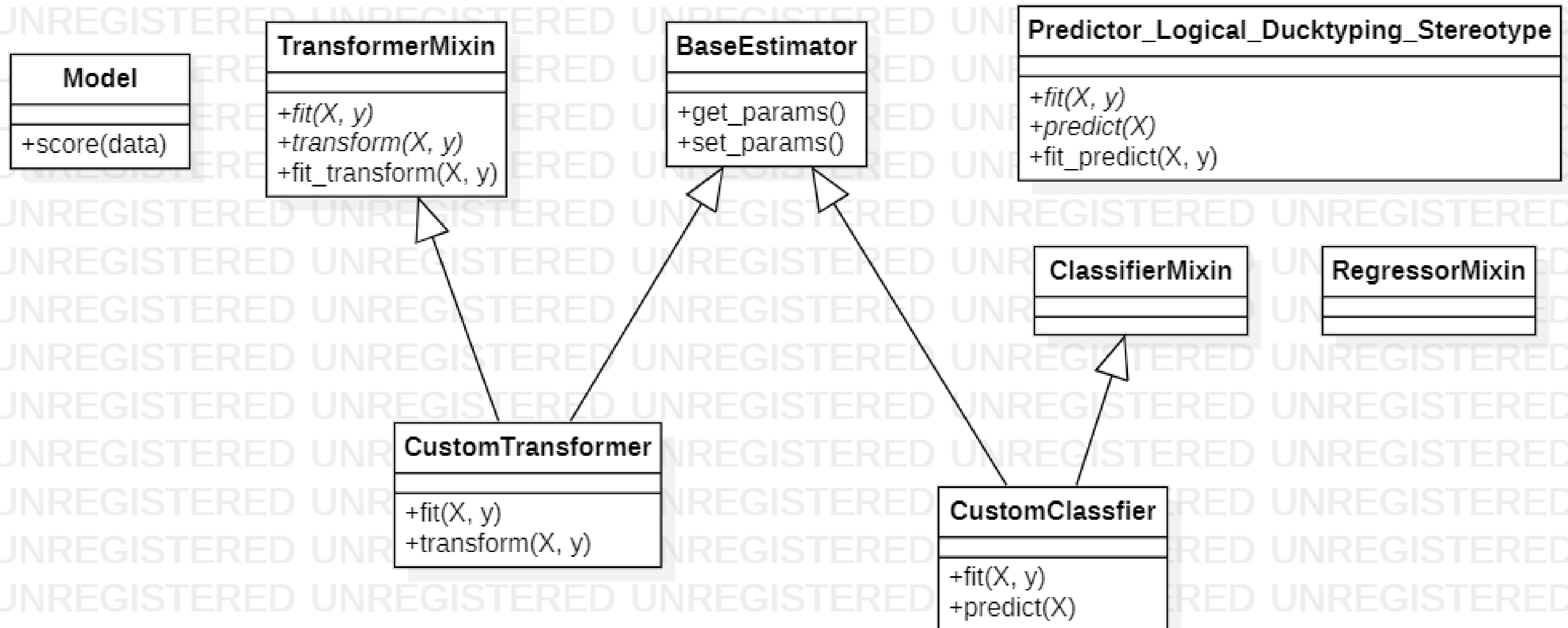
**Replace with  
pipeline**

# Odds and ends

- PowerTransformer
- FunctionTransformer
- BYOT – Bring Your Own Transformer
  - Custom Transformer, Custom Predictor



# Custom Transformer & Classifier/Regressor





QUESTIONS