1. Pipelines Introduction

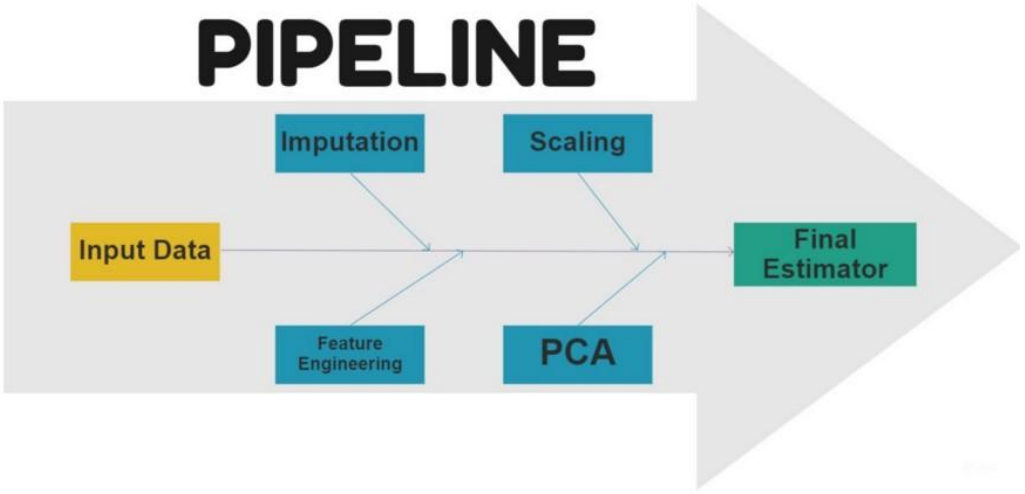
- A sequence of transformation (pre-processing, Feature Engineering) followed by estimators assembled and executed as a single entity.
- Transformers can be in parallel (FeatureUnion)

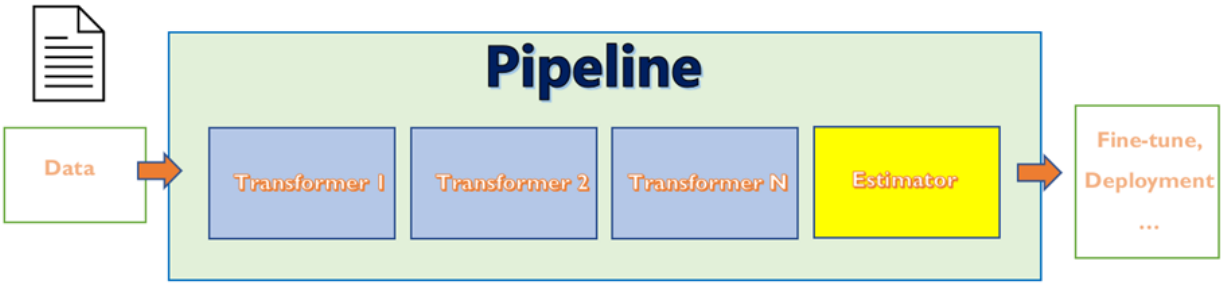
Advantages

Pipelines allows to encapsulate all preprocessing steps, feature selections, scaling, the encoding of features

- Convenience and encapsulation: Only have to call fit and transform/predict once on data to fit a whole sequence of estimators.
- Joint parameter selection: grid search over parameters of all estimators in the pipeline at once.

Generalised diagram of an end to end machine learning pipeline



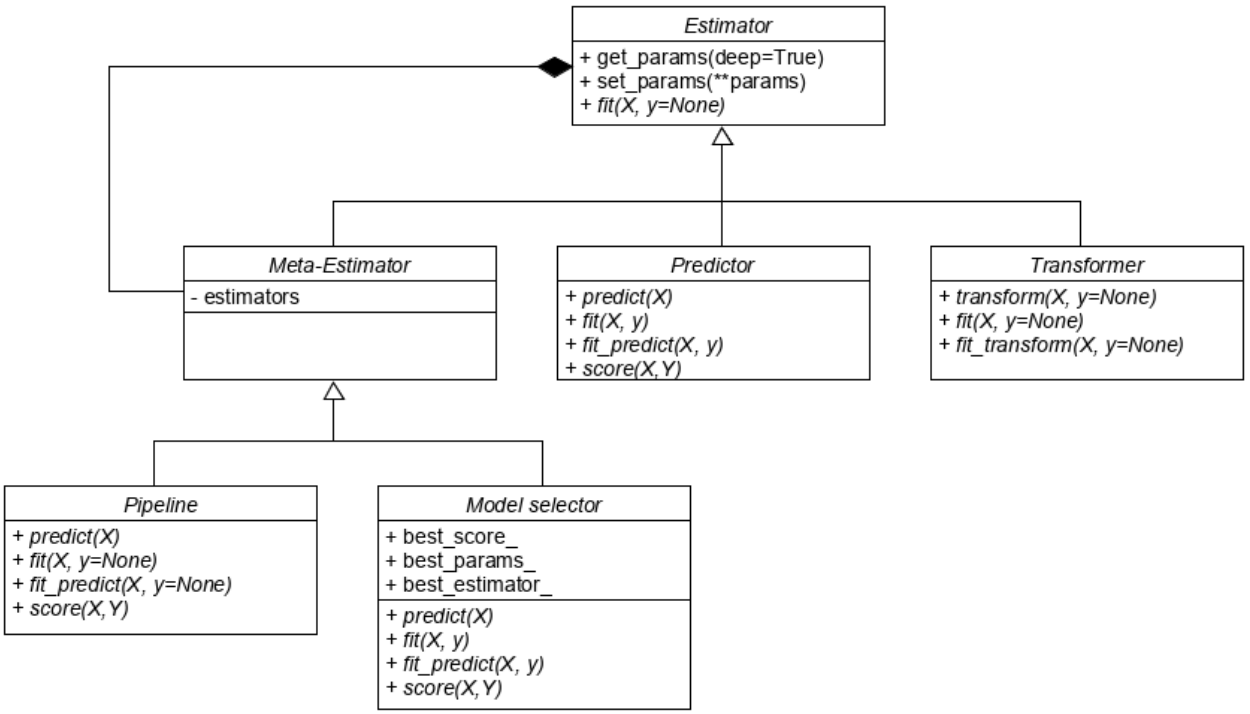


2. Workings of a Pipeline: Very Important

1. All but the last estimator ***must*** be transformers
2. When pipeline's fit() method is called, it calls fit_transform() sequentially on all transformers, passing the output of each call as the parameter to the next call until it reaches the final estimator, for which it calls the fit() method (with the following constraint)
3. ***The pipeline exposes the same methods as the final estimator***. If the last estimator is a StandardScaler, which is a transformer, then the pipeline has a transform() method that applies all the transforms to the data in sequence.
4. What to call on pipeline varies based on what the pipeline contains and whether train or test set is being used
    a. ***When pipeline has only transformers, call fit_transform() for training data set and transform() for test data set***
    b. ***When pipeline has both transformers and predictors, call fit() for training data set and predict() for test data set***

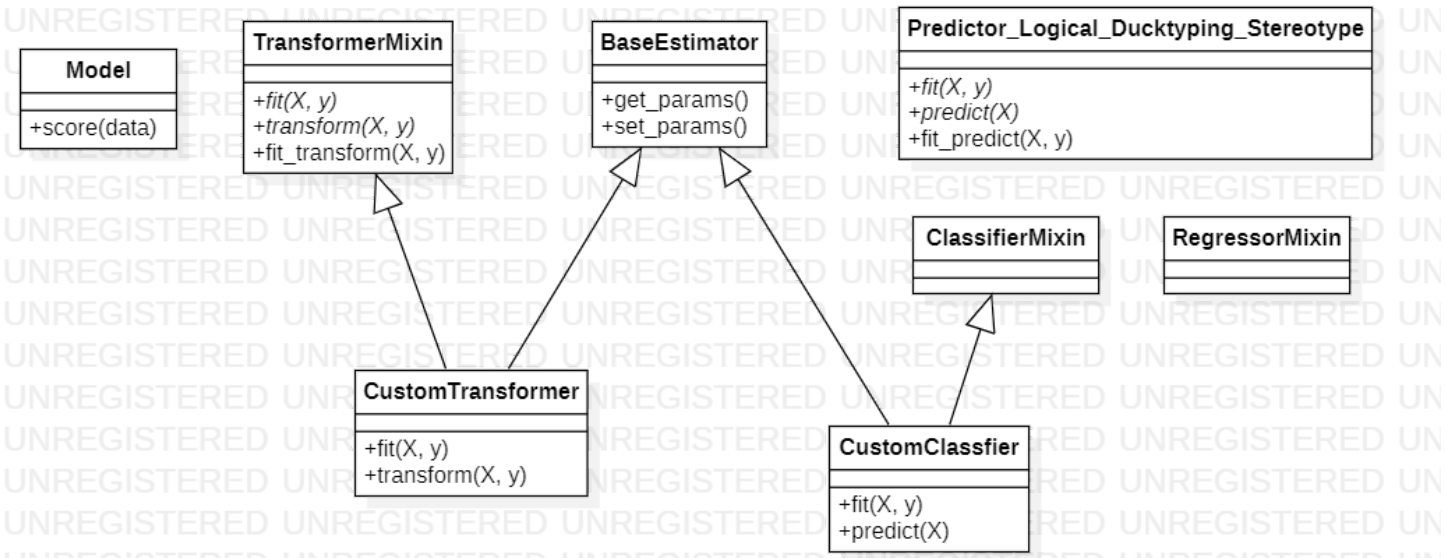
3. Scikit Learn Class & Package layout (wrt to Pipelines)

Logical view for comprehension[1]



[1] https://desosa.nl/projects/scikit-learn/2020/03/06/scikit-learn-what-does-it-want-to-be.html

Actual view for custom Transformer, Predictor creation



3.1 A deeper look at estimator, transformer and predictor[2]

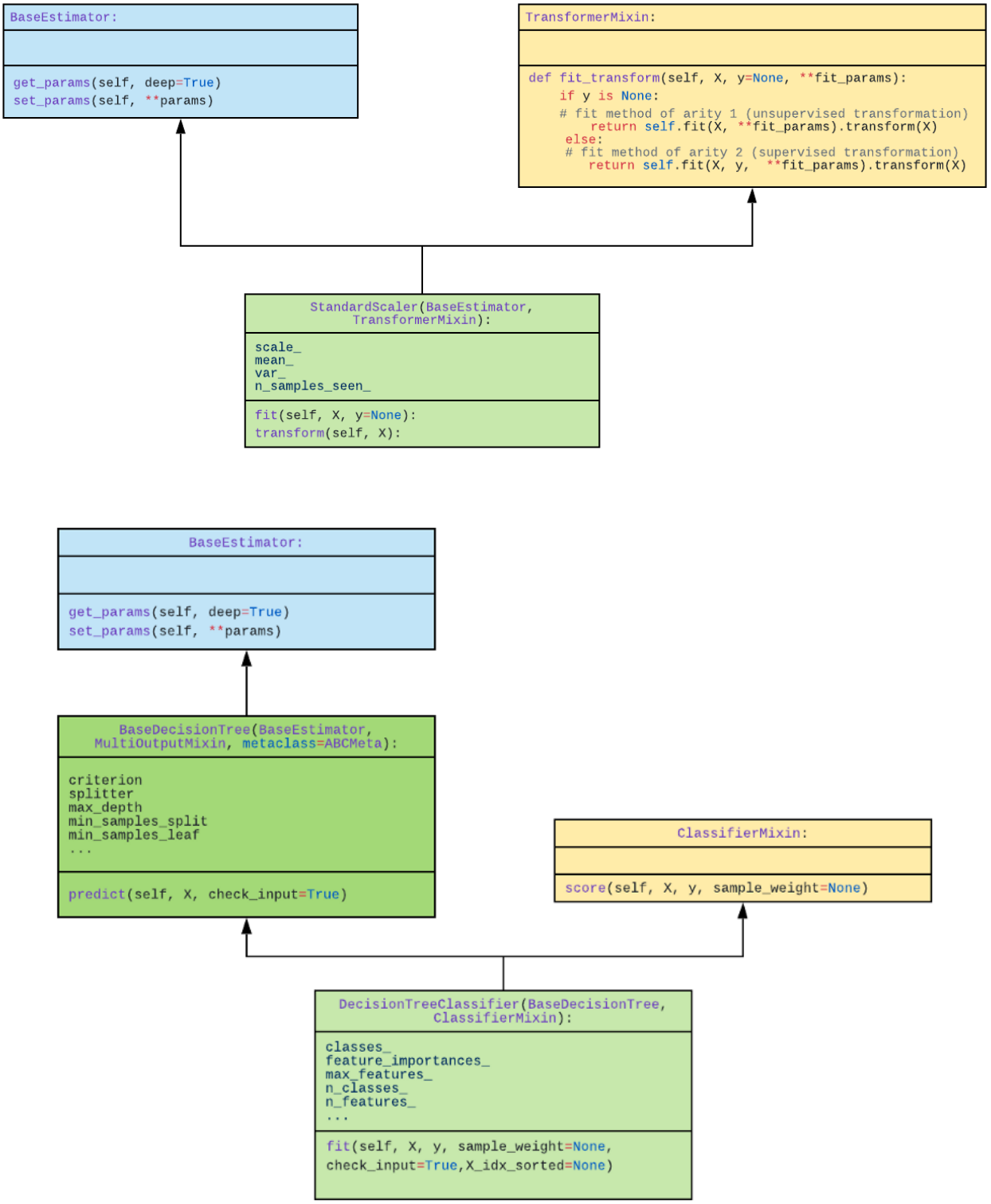
The library is organised around three fundamental APIs (interfaces): Estimator, Predictor, and Transformer. Importantly and crucially these interfaces are complementary — they do not represent hard boundaries between classes or precise semantic separation, but rather an overlap.

Sklearn is a library not a framework (except may be a little framework-ish with meta estimator and its subclass pipeline)

A very insightful and useful reference on sklearn API design can be found here[3].

[2] https://towardsdatascience.com/scikit-learn-design-principles-d1371958059b
[3] API design for machine learning software: experiences from sklearn project

3.2 A look at scikit learn Transformer and Predictor(Classifier) UML

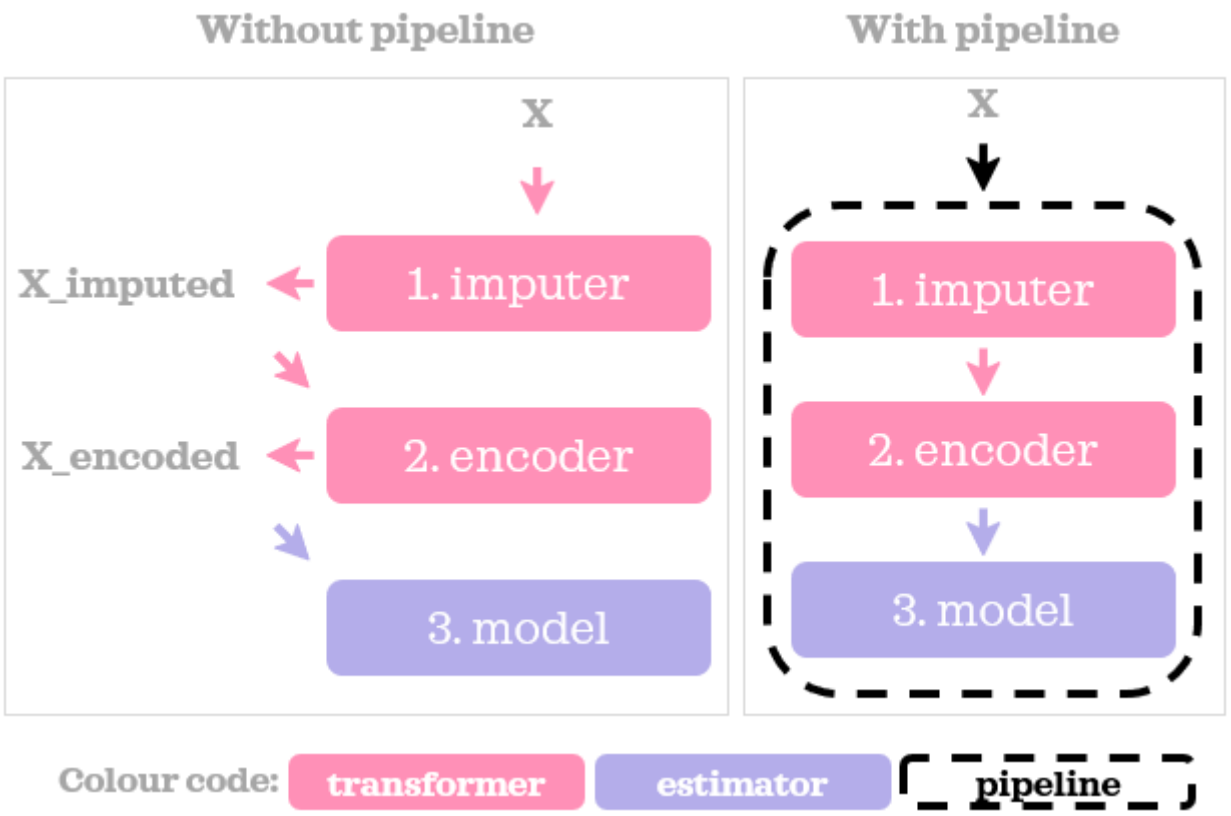


4. Creating a Pipeline

4.1 First Step: Preparation[4][5]

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=124)

#Numeric Feature
numeric_features = ['tenure']

#Categorical Features
categorical_features = ['SeniorCitizen', 'Partner', 'Dependents',
'PhoneService',

'InternetService','OnlineSecurity','OnlineBackup',
                        'DeviceProtection','TechSupport','StreamingTV',
                        'StreamingMovies','Contract']
```

4.2 Comparing with and without pipeline[6]



[4] Example from here: https://medium.com/swlh/how-to-use-scikit-learn-pipeline-for-your-ml-projects-e67d5a7e2c88
[5] Uses telco customer churn dataset
[6] https://towardsdatascience.com/pipeline-columntransformer-and-featureunion-explained-f5491f815f

4.2 Code without Pipeline

```
#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)

# Define the classifier
lr = LogisticRegression()

# 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)

# 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)
```

4.3 Code with Pipeline

```
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)
```

4.4 Important points when designing around ColumnTransformer & Pipeline

1. Column names are provided in ColumnTransformer
2. When transform() or fit_transform() are called (on the ColumnTransformer or predict called on the pipeline in which the Column Transformer resides along with the model), columns corresponding to those column names are extracted from input dataframe by the ColumnTransformer and transparently only those the DataFrame corresponding to those columns are passed to its constituent estimators. Even though the DataFrame is passed (when available), the column name information is not present. The columns are passed as indexes on the subset of columns passed into that respective estimator. (We could call this loose coupling of transformers to the dataset specific features thus making the code more reusable)
3. Output of all observed Transformers (sklearn or otherwise) used within ColumnTransformers (hence are likely to be generic) is always numpy array
4. Default is to drop all columns other than specified in transformers. Columns present in the output of the ColumnTransformer are in the order of processing (followed by remainder=passthrough if set). This means the order will be seriously messed up by ColumnTransformer by design, especially when

remainder=passthrough is used. Not knowing this can cause a lot of grief as seen in stackoverflow posts[7]

5. Many Custom Transformers (as in code provided on the net) accept col names as a list to the constructor. This is okay for very specific Custom Transformers meant just for single use. A better design for even a slightly generic Custom Transformer is to rely on CustomTransformer to pass in a data frame with only the respective columns (as described in 2) into the transform() by virtue of loose coupling of columns.
6. Even if a DataFrame is added within a custom transformer, its DataFrame nature will be stripped when it is invoked in a ColumnTransformer framework (i.e. input can be a numpy array-like or Pandas Dataframe [when provided so] but with no column names. Output is numpy array-like)
7. The above steps can make it a bit difficult to extract the final order of columns. ColumnTransformer provides API methods on member transformers and their steps to query this as follows[8] This can be used after column transformer execution for verifying the order

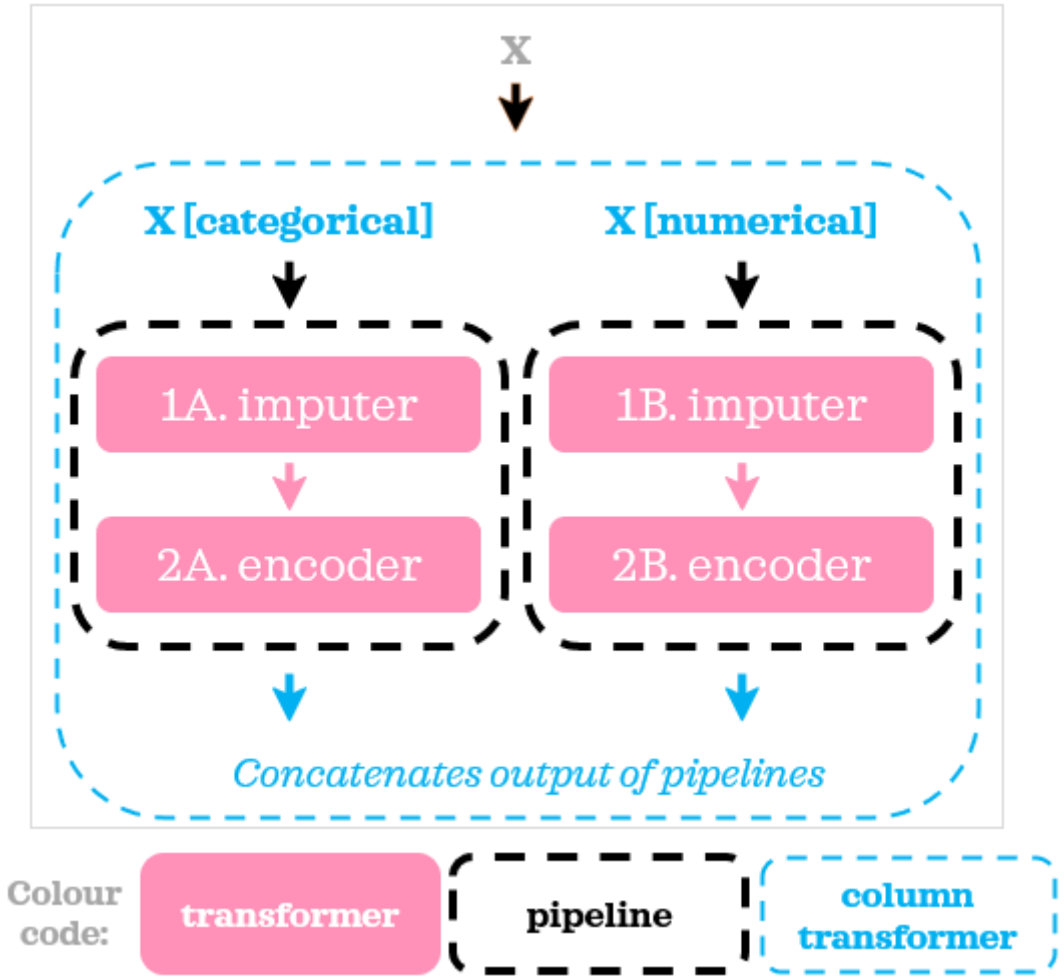
```
col_transformer.named_transformers_['ohe'].get_feature_names()

col_transformer.named_transformers_['cats'].named_steps['ohe'].get_feature_names()
```

8.

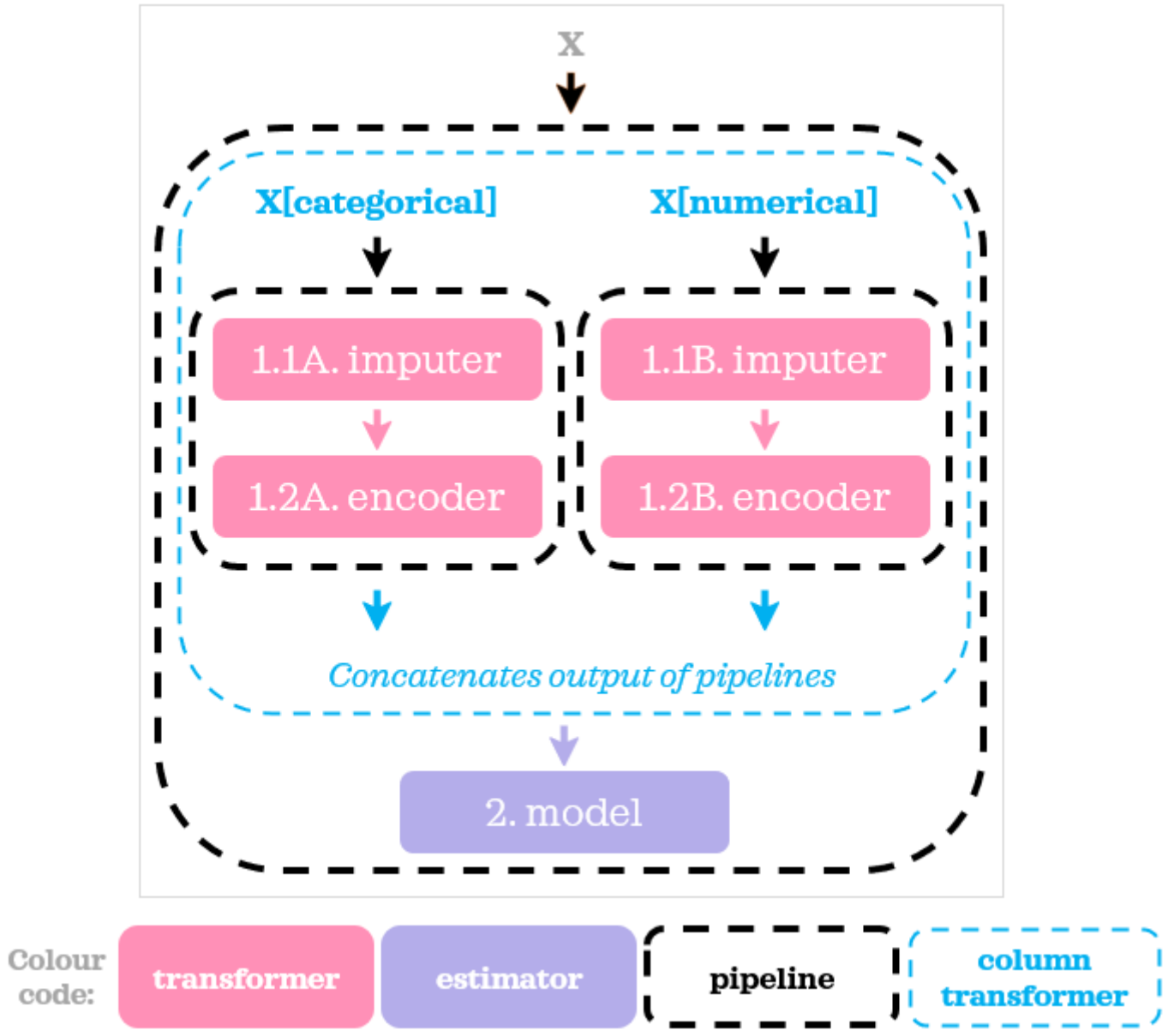
[7] https://stackoverflow.com/questions/68874492/preserve-column-order-after-applying-sklearn-compose-columntransformer

[8] https://towardsdatascience.com/using-columntransformer-to-combine-data-processing-steps-af383f7d5260

4.5 Schematic on ColumnTransformer usage

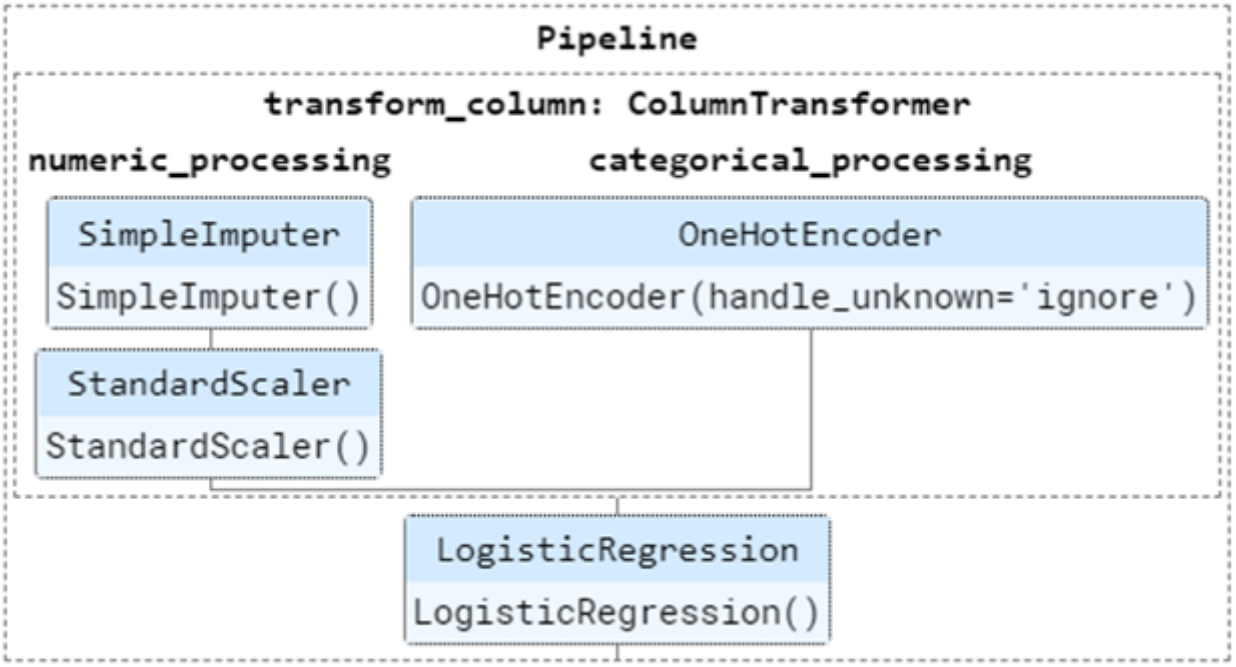


4.6 Schematic on Final Pipeline with Predictor



4.7 Visualising the pipeline

```
from sklearn import set_config
set_config(display='diagram')
pipeline
```



4.8 Pipeline with least code

Disadvantage of this least code pipeline approach is that steps cannot be named. Hence setting hyper parameters becomes hard (hyper parameter choice inside pipeline is in next section)

```
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline

numeric_transformer = make_pipeline((SimpleImputer(strategy='mean')),
                                    (StandardScaler()))

categorical_transformer = make_pipeline(OneHotEncoder(handle_unknown='ignore'))

col_transformer = make_column_transformer((numeric_transformer, numeric_features),
                                          (categorical_transformer, categorical_features))

pipeline = make_pipeline(col_transformer,LogisticRegression())

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

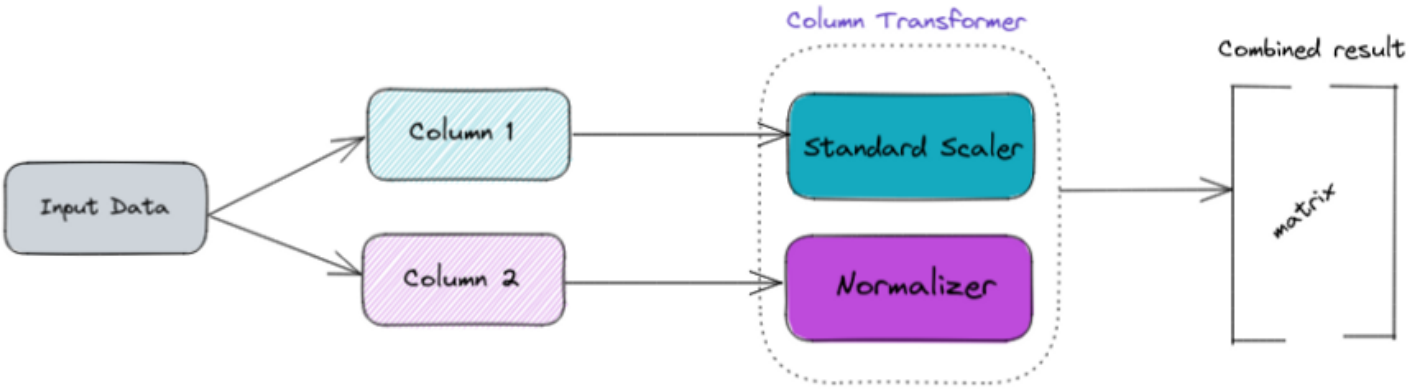
NOTE: Last line can be replaced with this

```
predictions = pipe.predict(X_test)

#checking pipeline's accuracy
accuracy_score(y_test, predictions)
```

5. Pipeline Estimators Details[9]

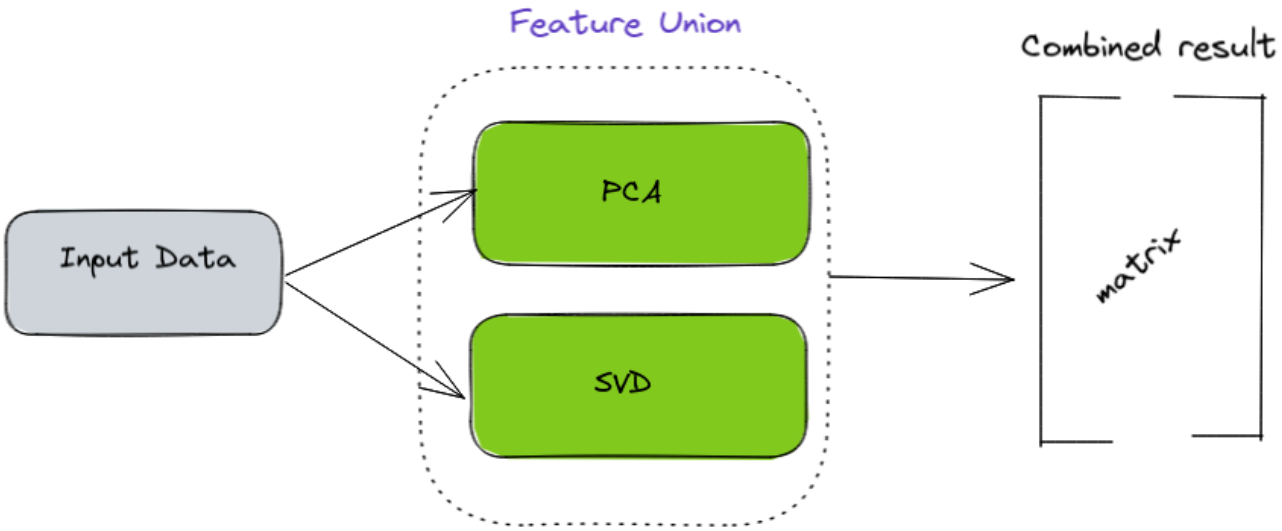
5.1 Column Transformer

Combines transformations applied to various columns & concatenates them.



5.2 Feature Union:

Data flow is not linear, but multiple things need to be preprocessed together. It executes all the functions in parallel and pastes the final results together. Use it when you want to apply different transformations to one column.



[9] https://towardsdatascience.com/make-a-rock-solid-ml-model-using-sklearn-pipeline-926f2ccf4706

TODO: Need to try out a FeatureUnion and see if it serves us better than ColumnTransformer in the project

5.3 Function Transformer

Converts user-defined function into a function that can be easily used and integrated with Pipeline.
<<TODO: Adding a function transformer example here. Real power in adding a quick and dirty short transformer as a code snippet, even as an inline lambda instead of having to create a full blown python class>>

5.4 Another simplistic variation

Wine data set with VarianceThreshold and KNN in pipeline[10]

6. GridSearchCV and Hyperparameters in Pipeline

This section shows additional mechanisms[11] using Kaggle's Adult Census Income dataset :

- options for specifying columns in the pipeline using data type.
- usage of GridSearch and specifying the hyper parameters

```
preprocessor = ColumnTransformer([
    ('numerical', numerical_pipe,
make_column_selector(dtype_include=['int', 'float'])),
    ('categorical', categorical_pipe,
make_column_selector(dtype_include=['object'])),
])
```

```
#defining the hyperparameter space for searching
parameters = {
    'column_transformer__numerical__imputer__strategy':
['mean', 'median'],
    'column_transformer__numerical__scaler':
[StandardScaler(), MinMaxScaler()],
    'model__n_neighbors': [3, 6, 10, 15],
    'model__weights': ['uniform', 'distance'],
    'model__leaf_size': [30, 40]
}
```

[10] https://towardsdatascience.com/introduction-to-scikit-learns-pipelines-565cc549754a
[11] https://towardsdatascience.com/unleash-the-power-of-scikit-learns-pipelines-b5f03f9196de

```
#defining a scorer and a GridSearchCV instance
my_scorer = make_scorer(accuracy_score,
greater_is_better=True)
search = GridSearchCV(pipe, parameters, cv=3,
scoring=my_scorer, n_jobs=-1, verbose=1)

#search for the best hyperparameter combination within our
defined hyperparameter space
search.fit(X_train, y_train)

#changing pipeline parameters
pipe.set_params(**search.best_params_)

#making predictions
predictions = pipe.predict(X_test)

#checking accuracy
accuracy_score(y_test, predictions)
```

The parameters dictionary is created as follows:[12]

`algorithm+double underscore + hyperparameter`

To see the all available hyperparameter that can be used for an transformer/predictor, use `<varname>.get_params().keys()`

E.g.

Logistic Regression	params=[{'lr__solver':['saga'],'lr__C':[0.1,1,10],'lr__penalty':['elastic net','l1','l2']}, {'lr__solver':['lbfgs'],'lr__C':[0.1,1,10],'lr__penalty':['l2']}]
Decision Tree	Params = ={'dt__max_depth':[2,3,4,5,6,7,8]}
anova = SelectPercentile() poly = PolynomialFeatures()	dict(poly__degree=[2,3,4], anova__percentile=[20, 30, 40, 50], lr__C=[0.01,0.1,1,10], lr__penalty=['l1','l2'])

12 https://towardsdatascience.com/easy-way-to-create-an-algorithm-chains-pipelines-with-grid-search-columntransformer-feature-24feb3d6aec5

7. Writing a Custom Transformer

A sample transformer is shown below from OReilly Hands on ML with Sklearn book chapter 2[13]
Please refer to section 3 and 4.4 for Custom Transformer design tips

```
from sklearn.base import BaseEstimator, TransformerMixin

rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]

        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

Please refer to these as well
https://www.section.io/engineering-education/custom-transformer/

https://machinelearningmastery.com/create-custom-data-transforms-for-scikit-learn/

Especially this one outlier transformer
https://www.kaggle.com/ksvmuralidhar/creating-custom-transformers-using-scikit-learn

[13] https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/ch02.html

8. Pipelines in H2O

All sklearn pipeline constructs can be used as is with H2O. A slightly old but useful material here: https://www.youtube.com/watch?v=KqLXFglgNqk
Libraries for GridSearch can also be used. Perhaps ColumnTransformer also can be used (did not try yet). It's just that the dataframe passed into them is H2oFrame. Sometimes if needed h2oframe can be converted to a pandas dataframe and used locally
Encoders, transformers and estimators are h2o specific since they have to operate on data in a different memory space (that of h2o cluster)
More details on this in the document
https://docs.google.com/document/d/1Ji5VzefwigQYJCe6txAggUl7cekNn6XFcPWp-2BDrqE/

There is a pipeline equivalent called H2OAssembly
And there are classes such as H2OColOp that are somewhat equivalent of operation on columns

9. Encoder gotchas

TODO: Make a new note/document out of this section

9.1 Ordinal Encoder gotchas

This is based on my findings with EDA on ICU data in the Survival Analysis project.
<<Link to Jupyter Notebook here>>

9.2 New category showing up in test input or production data.. Oops

Especially tip 4 in this article is a real problem that can occur very infrequently but there is a possibility[14]

10. Additional References

TODO: Please revise and refine this section

10.1 Intermediate and Revision of pipeline concepts

https://towardsdatascience.com/pipelines-custom-transformers-in-scikit-learn-ef792bbb3260

[14] https://towardsdatascience.com/using-columntransformer-to-combine-data-processing-steps-af383f7d5260

https://www.kaggle.com/baghern/a-deep-dive-into-sklearn-pipelines
https://www.kaggle.com/metadist/work-like-a-pro-with-pipelines-and-feature-unions

10.2 Advanced

https://towardsdatascience.com/customizing-sklearn-pipelines-transformermixin-a54341d8d624
https://towardsdatascience.com/pipeline-columntransformer-and-featureunion-explained-f5491f815f
https://towardsdatascience.com/featureunion-columntransformer-pipeline-for-preprocessing-text-data-9dcb233dbcb6
https://towardsdatascience.com/pipeline-for-text-data-pre-processing-a9887b4e2db3