INDIAN INSTITUTE OF TECHNOLOGY MADRAS • सिद्धिर्भवति कर्मजा •

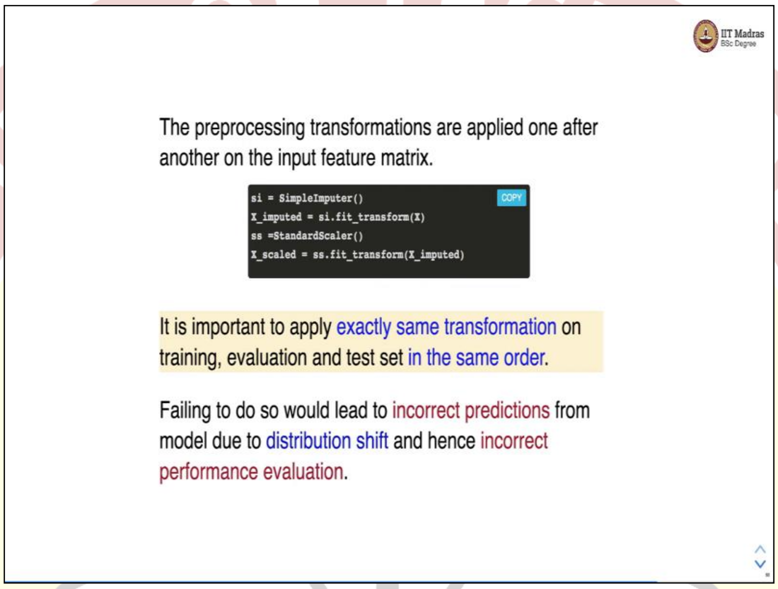
IIT Madras

ONLINE DEGREE

Machine Learning Practice
Online Degree Programme
B. Sc in Programming and Data Science
Diploma Level
Dr. Ashish Tendulkar
Indian Institute of Technology – Madras

Part 4: Chaining Transformers

(Refer Slide Time: 00:22)



The preprocessing transformations are applied one after another on the input feature matrix.

```
si = SimpleImputer()
X_imputed = si.fit_transform(X)
ss =StandardScaler()
X_scaled = ss.fit_transform(X_imputed)
```

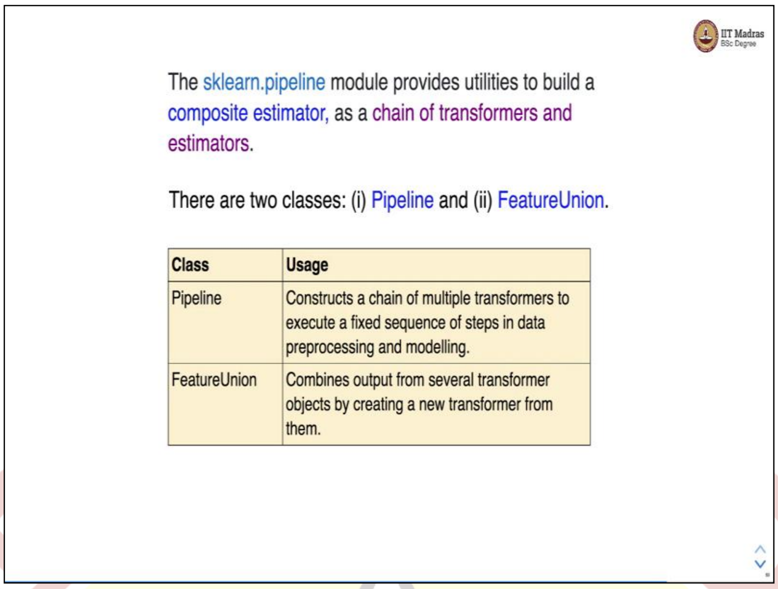
It is important to apply exactly same transformation on training, evaluation and test set in the same order.

Failing to do so would lead to incorrect predictions from model due to distribution shift and hence incorrect performance evaluation.

Namaste welcome to the next video of the machine learning practice course. In this video, we will study how to chain transformers one after the other in an efficient manner. The preprocessing transformations are applied one after another on the input feature matrix. Here is an example in this example we apply simple imputer and standard scalar one after the other on the feature matrix.

It is important to apply the same transformation on training evaluation and test set in the same order. Failing to do so would lead to incorrect predictions from the model due to distribution shift and hence incorrect performance evaluation.

(Refer Slide Time: 00:57)

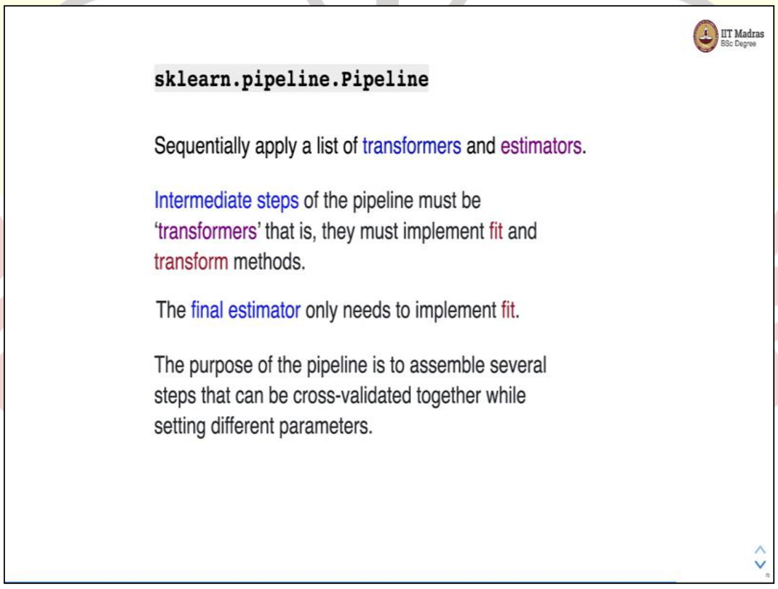

The sklearn.pipeline module provides utilities to build a composite estimator, as a chain of transformers and estimators.

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.

Sklearn. pipeline module provides utilities to build a composite estimator. The composite estimator is a chain of transformers and estimators. There are two classes pipeline and feature union. The pipeline class constructs a chain of multiple transformers to execute a fixed sequence of steps in data pre-processing and modeling. On the other hand, the feature union combines output from several transformer objects by creating a new transformer from them.

(Refer Slide Time: 01:32)


`sklearn.pipeline.Pipeline`

Sequentially apply a list of transformers and estimators.

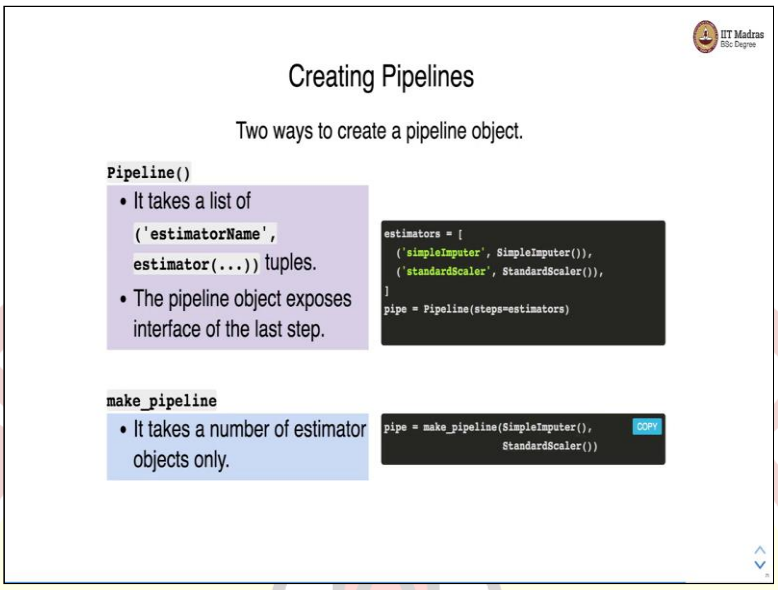
Intermediate steps of the pipeline must be 'transformers' that is, they must implement fit and transform methods.

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.

Let us first study the pipeline API from the sklearn.pipeline module. The pipeline API sequentially applies a list of transformers and estimators. The intermediate steps of the pipeline must be transformers that is they must implement the fit and transform method. The final estimator only needs to implement the fit. The purpose of the pipeline is to assemble several steps that can be cross-validated together while setting different parameters.

(Refer Slide Time: 02:10)



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())
```

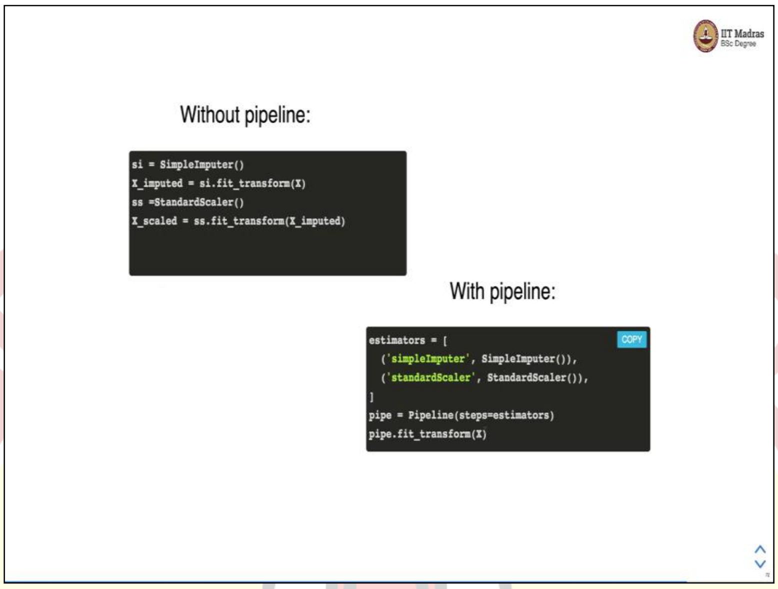
There are two ways to create pipeline objects either using the pipeline constructor or using the make_pipeline method. The pipeline constructor takes a list of tuples the first part of the tuple is some kind of an id of the transformation that we want to apply and the second is the transformer object itself. The pipeline object exposes the interface of the last step. Let us take an example here we are defining a pipeline object on several steps defined in this estimator object.

And the estimator object is a list of tuples the first apple is the simple imputer. So, we are using a simple imputer transformer and we are giving it a name called simple computer so that we can refer to this particular step through this particular name. So, the second tuple contains the second transformation which is a standard scalar. So, here we have a standard scalar transformer object and the name that we have given to this particular transformer object.

So, we put this list of tuples as an argument in the pipeline constructor and construct a pipeline object. Another way to construct a pipeline object is to make n _pipeline method it takes several estimator objects. So, we do not need to give tuples as in the case of the pipeline here we can just give the number of estimator objects. For example, we can specify the same thing with making n _pipeline by specifying the number of estimator object one after the other.

So, here we make a pipeline with a simple computer transformer as the first step and a standard scalar transformer as the second step.

(Refer Slide Time: 04:13)



Without pipeline:

```
si = SimpleImputer()
X_imputed = si.fit_transform(X)
ss =StandardScaler()
X_scaled = ss.fit_transform(X_imputed)
```

With pipeline:

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

So, let us try to understand what is the advantage of using pipelines by looking at two code snippets one without pipeline and the second is with the pipeline. You can see that in the first step without pipeline we have to write every line of the code and make sure that we appropriately pass the feature matrix. So, here we use the original feature matrix we perform a simple computer on this to handle the missing values then we get x_imputed as a transform matrix.

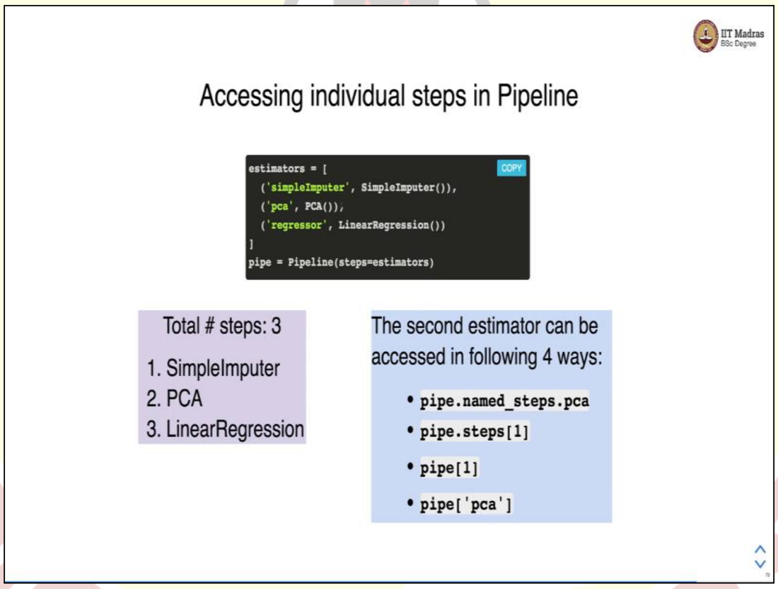
We pass this transform matrix to the standard scalar transformer which performs the feature scaling in this particular transform matrix and then we get another transfer matrix which is x_scaled. If you want to apply further transformers on this, we need to pass this x_scale to that particular transformer. So, here we have to make sure that we are passing these transform feature matrices appropriately and there is no mistake.

If there is any mistake what would happen is that we might miss some of the transformations in between and we have to also make sure that we apply the same piece of code on other datasets like the evaluation dataset as well as on the test dataset. If we fail to do that it will have disastrous consequences. So, now you look at the same piece of code with pipeline and you can immediately see that the code with pipeline looks much more elegant and manageable than this particular piece of code.

In the pipeline construct, we simply specify the list of tuples containing the transformer objects along with their ids and then we initialize the pipeline object and we simply call pipe.fit_transform on the feature matrix we do not have to pass around the intermediate transform feature matrix properly from one transformer to the other. So, you can see that here the final method is standard scalar hence this fit and so, we expose the pipeline object exposes this fit_transfer method of the final estimator or final transformation object.

So, here we apply pipe.fit transform bypassing feature matrix as an argument. So, what it will do is it will first apply simple imputer and then the standard scalar transformation to the original feature matrix, and then we will get a transform feature matrix with both the transformations applied to the original feature matrix.

(Refer Slide Time: 07:07)



IIT Madras
BSc Degree

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']`

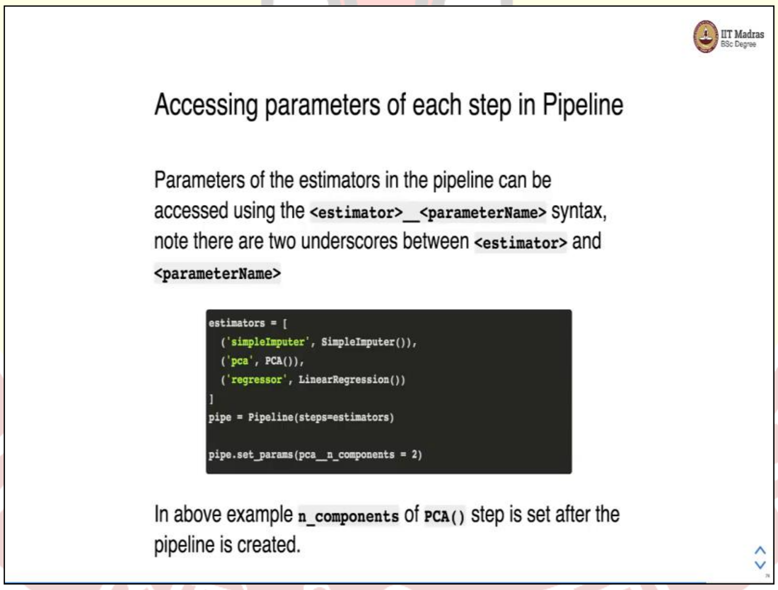
So, let us see how to access individual steps in the pipeline. So, in this particular example code, there is total of three steps the first step applies simple imputer transformation followed by principal component analysis, and then there is a linear regression estimator at the end. So, what it is doing is before applying the linear regression estimator or before applying linear regression on the feature matrix we are first filling up the missing values.

Then applying principal component analysis to select important features and then we apply this linear regression on these particular features that are obtained to principal component analysis. Now you can see that we have instantiated a pipeline object with these three steps. So, now let us say if you want to access the second step or the second estimator which is PCA.

So, there are total four ways to access it. We can either call pipe.named_steps.pca. So, when we are calling this named step we are using the ids that we are giving here. So, we can refer to PCA by this id by calling pipe.named_steps.PCA. In the same manner, the third step can be accessed as the pipe.named_space.regressor the first step can be accessed as .named_steps.simple imputer.

So, this is the id that is used in named steps. So, the second estimator can be also accessed by the pipe.Step one. So, steps are some kind of an array that stores the steps, and the first step is PCA. We can also call pipe1 to refer to the same thing which is the PCA step or we can call pipe PCA this is more like a dictionary notation, got it. So, there are four different ways in which we can access the second component.

(Refer Slide Time: 09:21)



Accessing parameters of each step in Pipeline

Parameters of the estimators in the pipeline can be accessed using the `<estimator>__<parameterName>` syntax, note there are two underscores between `<estimator>` and `<parameterName>`

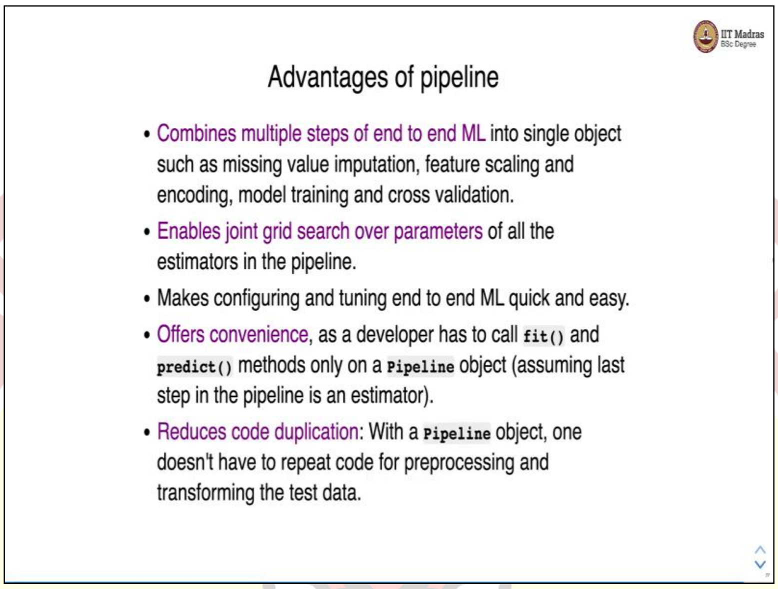
```
estimators = [
  ('simpleImputer', SimpleImputer()),
  ('pca', PCA()),
  ('regressor', LinearRegression())
]
pipe = Pipeline(steps=estimators)

pipe.set_params(pca__n_components = 2)
```

In above example `n_components` of `PCA()` step is set after the pipeline is created.

Sometimes we need to access parameters of each step in the pipeline and these parameters can be accessed by the name of the estimator double the name of the parameter. Let us take an example in this particular code we are applying simple imputer followed by principal component analysis followed by linear regression. And you can see that we are setting the parameter of this PCA step we are trying to set n _components parameter of the PCA.

And we are referring to this parameter as the name of this particular estimator which is PCA double and the name of the component name of the parameter and that name is end_component. So, we have PCA double_n_components that will refer to this particular parameter of the PCA and we are setting this parameter to 2.

(Refer Slide Time: 10:22)



Advantages of pipeline

- Combines multiple steps of end to end ML into single object such as missing value imputation, feature scaling and encoding, model training and cross validation.
- Enables joint grid search over parameters of all the estimators in the pipeline.
- Makes configuring and tuning end to end ML quick and easy.
- Offers convenience, as a developer has to call `fit()` and `predict()` methods only on a `Pipeline` object (assuming last step in the pipeline is an estimator).
- Reduces code duplication: With a `Pipeline` object, one doesn't have to repeat code for preprocessing and transforming the test data.

We can also perform grid search with pipeline here we are specifying first the parameter grid where we are trying to find out which computer is but better. So, we have three computers one is pass through second is simple imputer and the third is KNNimputer. Bypassing through we mean that we do not want to perform any imputation we want to expand with two classes one is SVC which is a support vector classifier and the logistic regression classifier.

And we want to try with three different values of the parameter C which is the inverse of regularization. The lower value of C indicates the stronger regularization. So, you want to try three values of C which are 0.1, 10, and 100, and what we do is specify this parameter grid. So, what will happen is it will try to find out the best combination of imputer clf that is the class and the value of c through the grid search cv.

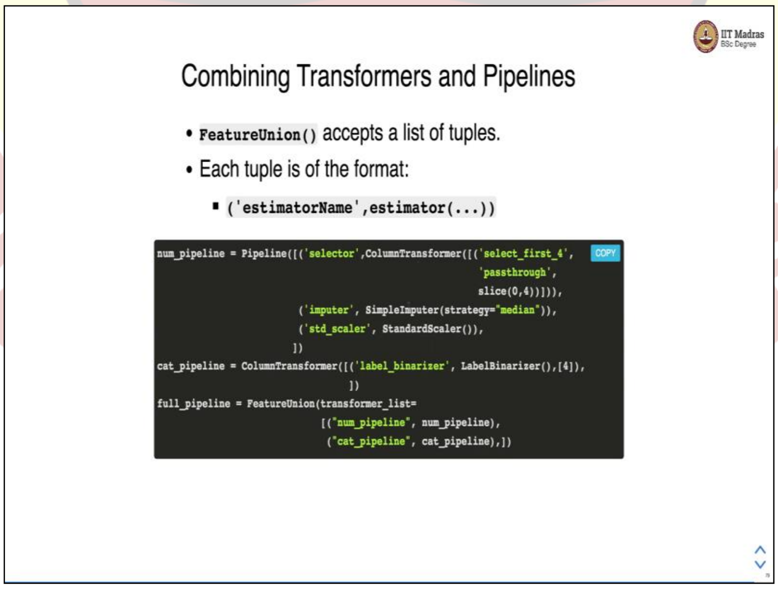
So, we call GridSearchCV on this particular parameter grid we will study GridSearchCV in detail probably in the next one or two weeks and GridSearchCV basically what it does is it tries to find out it first performs the grid search and finds out the values of the hyperparameter that maximizes the cross-validation score. So, we perform the GridSearchCV with cross-validation on the pipeline object with the specified parameter grid and at the end of the process, we get the best values for the imputer class and hyperparameter c.

So, transforming data is a computationally expensive task for grid search transformer need not be applied for every parameter configuration. They can be applied only once and the transformation data can be reused that can be achieved by setting the memory parameter of a pipeline object the memory parameter takes either a location of a directory in string format or joblab.memory object.

In this particular illustration, we are specifying the location of the directory in the string format. So, what are the advantages of pipeline helps us combine multiple steps of end-to-end machine learning pipeline into a single object it enables joint grid search or parameters of all the estimators in the pipeline it makes configuration and tuning of end-to-end machine learning pipeline quick and easy.

It offers convenience as the developer has to call only fit and predict methods on the pipeline object. It reduces code duplication with pipeline objects one does not have to repeat code for pre-processing and transforming on the test set.

(Refer Slide Time: 13:42)



Combining Transformers and Pipelines

- `FeatureUnion()` accepts a list of tuples.
- Each tuple is of the format:
  - `('estimatorName',estimator(...))`

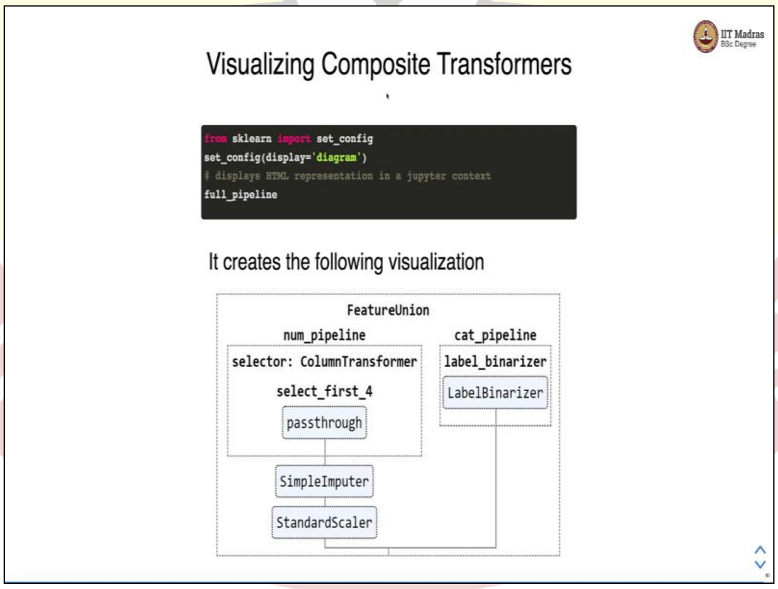
```
num_pipeline = Pipeline([('selector',ColumnTransformer([('select_first_4',
                                                          'passthrough',
                                                          slice(0,4))])),
                         ('imputer', SimpleImputer(strategy="median")),
  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

So, that was about pipeline construct the second construct is feature union. So, feature union concatenates the result of multiple transformer objects. It first applies a list of transformer objects in parallel and their outputs are concatenated side by side into a larger matrix. So, feature union and pipeline can be used to create complex transformers. So, let us see how to combine transformers and pipelines in the future union. So, future unions accept a list or tuple.

Each tuple has a format the estimator name followed by the estimator object this format is very similar to the arguments that we give in the pipeline object. So, here we are defining a feature union object with a list of transformers. So, this is the name of the transformer which is num pipeline we are giving some kind of an id followed by its object. So, here there are two pipelines one is on the one which is applied on the numerical features and the second one that is applied on the categorical features.

So, the num_pipeline has numerous or pipeline object which has got three steps one is the selector second is the imputer and third is the standard scalar. The selector selects selector basically selects the first four and then it performs a pass through. So, select the first four features and then it applies simple imputer on it with using the strategy minimum and then it applies or it does the scaling of the features. On the other hand, the column transformer applies LabelBinarizer.

(Refer Slide Time: 15:22)



Visualizing Composite Transformers

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

It creates the following visualization

So, let us visualize this composite transformer that we created using the pipeline and column transformers. So, here you can see that we have a feature union which has two pipeline num pipeline and cat pipeline. Num pipeline has a selector which is a column transformer with select first four features then there is a pass-through then we apply simple imputer.

On the resulting matrix on the resulting feature matrix followed by a standard scalar on the other hand a categorical pipeline applies LabelBinarizer and what feature union does it does is it concatenates output from these two pipelines and makes a single transform feature matrix. So, that is it from data preprocessing. So, we looked at different pre-processing steps like how to perform let us say scaling of the features.

How to handle missing values, how to perform feature selection feature reduction through PCA, and then how to use the pipeline construct pipeline and feature union construct for an efficient way of writing code for performing data preprocessing. Hope you use all these constructs in subsequent classes of machine learning practice courses. So, this was one of the most important topics that we learned.

And as I said in my first class that 80% to 85% of the work in machine learning that we do is in data pre-processing and constructing right kind of features in that sense the pipeline and feature union are two important tools in your toolkit.