the scikit-learn library provides us with three important methods, namely fit(), transform(), and fit\_transform() Sklearn, that are used widely in machine learning. The fit() method helps in fitting the data into a model, transform() method helps in transforming the data into a form that is more suitable for the model. Fit\_transform() method, on the other hand, combines the functionalities of both fit() and transform() methods in one step. Understanding the differences between these methods is very important to perform effective data preprocessing and feature engineering.

Key Takeaways

* The fit() method helps in fitting the training dataset into an estimator (ML algorithms).
* The transform() helps in transforming the data into a more suitable form for the model.
* The fit\_transform() method combines the functionalities of both fit() and transform().

**Now using standard scaler we first fit and then transform our dataset.**

from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()

X\_train\_fit=scaler.fit(X\_train)

X\_train\_scaled=scaler.transform(X\_train)

pd.DataFrame(X\_train\_scaled)

**Use fit\_transform() function directly and verify the results.**

X\_train\_scaled=scaler.fit\_transform(X\_train)

pd.DataFrame(X\_train\_scaled)

**Transform our test data.**

X\_test\_scaled=scaler.transform(X\_test)

pd.DataFrame(X\_test\_scaled)

**Conclusion**

**–** Here we observe that the **fit\_transform()** function gives the same result as the function **fit()** and the **transform()** function gives separately by combining the results.

**–**Remember **fit\_transform()** function only acts on training data, **transform()** acts on test data, and **predict()** acts on test data.

**–**In summary, **fit()** performs or completes the training step, **transform()**changes the data in the pipeline to pass it on to the next stage in the pipeline, and **fit\_transform()**does both the fitting and the transforming in one possibly short step.

|  |  |  |  |
| --- | --- | --- | --- |
| fit() | Learn and estimate the parameters of the transformation | estimator.fit(X) | estimator.fit(train\_data) |
| transform() | Apply the learned transformation to new data | transformed\_data = estimator.transform(X) | transformed\_data = estimator.transform(test\_data) |
| fit\_transform() | Learn the parameters and apply the transformation to new data | transformed\_data = estimator.fit\_transform(X) | transformed\_data = estimator.fit\_tran |

**Enumerate:**

The basic syntax is is *enumerate(sequence, start=0)*

The output object includes a counter like so: (0, thing[0]), (1, thing[1]), (2, thing[2]),

As input it takes a sequence like a list, tuple or iterator. The start parameter is optional.  
If the start parameter is set to one, counting will start from one instead of zero

Create a sequence and feed it to the enumerate function. This can be any type of sequence, in this example we use a list. Then we output the object.

Try the program below:

|  |
| --- |
| *# create a sequence* browsers = ['Chrome','Firefox','Opera','Vivaldi']  *# create an enumeratable and convert to list* x = list(enumerate(browsers)) print(x) |

Let’s see how you can enumerate a Python list. You can open the Python shell to try it out.

You can iterate over a Python list by using **enumerate()**. Lets see that in a simple example.

|  |
| --- |
| >>> fruits = [ "Apple","Berry","Cherry" ] >>> for i,j in enumerate(fruits): ... print(i,j) ...  0 Apple 1 Berry 2 Cherry >>> |

It outputs both the index (i) and the value (j).