

Kaggle Challenge - Data Transformation

WE'LL COVER THE FOLLOWING ^

- 3. Transformation Pipelines
 - Jupyter Notebook

3. Transformation Pipelines

As you can see, from imputing missing values to feature scaling to handling categorical attributes, we have many data transformation steps that need to be executed in the right order. Fortunately, Scikit-Learn is here to make our life easier: Scikit-Learn provides the `Pipeline` class to help with such sequences of transformations.

✎ **Note:** Creating transformation pipelines is optional. It is handy when dealing with a large number of attributes, so it is a good-to-know feature of Scikit-Learn. In fact, at this point we could directly move on to create our machine learning model. However, for learning how things are done, we are going to look at working with pipelines.

Some Scikit-Learn terminology:

- **Estimators:** An object that can estimate some parameters based on a dataset, e.g., an imputer is an estimator). The estimation itself is performed by simply calling the `fit()` method.
- **Transformers:** Some estimators (such as an imputer) can also transform a dataset; these are called transformers. The transformation is performed by the handy and easy to use `transform()` method with the dataset to transform as a parameter.

- **Predictors:** Some estimators are capable of making predictions given a dataset; they are called predictors. For example, the LinearRegression model is a predictor. A predictor has a `predict()` method that takes a dataset of new instances and returns a dataset of corresponding predictions. It also has a `score()` method that measures the quality of the predictions given a test set.

Based on some of the data preparation steps we have identified so far, we are going to create a transformation pipeline based on `SimpleImputer` (*) and `StandardScaler` classes for the numerical attributes and `OneHotEncoder` for dealing with categorical attributes.

(*)Scikit-Learn provides a very handy class, `SimpleImputer` to take care of missing values. You just tell it the type of imputation, e.g. by median, and voila, the job is done. We have already talked about the other two classes.

First, we will look at a simple example pipeline to impute and scale numerical attributes. Then we will create a full pipeline to handle both numerical and categorical attributes in one go.

The numerical pipeline:

```
# Import modules
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer

# Separate features and target variable
housing_X = housing_processed.drop("SalePrice", axis=1)
housing_y = housing_processed["SalePrice"].copy()

# Get the list of names for numerical and categorical attributes separately
num_attributes = housing_X.select_dtypes(exclude='object')
cat_attributes = housing_X.select_dtypes(include='object')

num_attribs = list(num_attributes)
cat_attribs = list(cat_attributes)

# Numerical Pipeline to impute any missing values with the median and scale attributes
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler()),
])
```

```

1: [48]: # Import modules
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer

# Separate features and target variable
housing_X = housing_preprocessed.drop('SalePrice', axis=1)
housing_y = housing_preprocessed['SalePrice'].copy()

# Get the list of names for numerical and categorical attributes separately
num_attributes = housing_X.select_dtypes(exclude='object')
cat_attributes = housing_X.select_dtypes(include='object')

num_attribs = list(num_attributes)
cat_attribs = list(cat_attributes)

# Assemble Pipeline to impute any missing values with the median and scale attributes
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('std_scaler', StandardScaler()),
])

```

Note that we have separated the *SalePrice* attribute into a separate variable, because for creating the machine learning model, we need to separate all the features, *housing_X*, from the target variable, *housing_y*.

The Pipeline constructor takes a list of name/estimator pairs defining a sequence of steps. The names can be whatever we want as long as they are unique and without double underscores, “__”.

The pipeline is run sequentially, one transformer at a time, passing the output of each call as the parameter to the next call. In this example, the last estimator is a *StandardScaler* (a transformer), and the pipeline applies all the transforms to the data in sequence.

So far, we have handled categorical and numerical attributes separately. It is more convenient and clean to have a single transformer handle all columns, applying the appropriate transformations to each column. Scikit-Learn comes to the rescue again by providing the *ColumnTransformer* for the very purpose. Let's use it to apply all the transformations to our data and create a complete pipeline.

(*num_pipeline* is the numerical pipeline from the previous step)

```

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OneHotEncoder(), cat_attribs),
])

# Description before applying transforms
print(housing_y.describe())

```

```
# Apply log-transform to SalePrice
housing_y_prepared = np.log(housing_y)

# Run the transformation pipeline on all the other attributes
housing_X_prepared = full_pipeline.fit_transform(housing_X)

# Description before applying transforms
print(housing_y_prepared.describe())

housing_X_prepared
```

```
In [41]: full_pipeline = ColumnTransformer([
    ('num', num_pipeline, num_attribs),
    ('cat', OneHotEncoder(), cat_attribs),
    ])

# Description before applying transforms
print(housing_y.describe())

# Apply log-transform to SalePrice
housing_y_prepared = np.log(housing_y)

# Run the transformation pipeline on all the other attributes
housing_X_prepared = full_pipeline.fit_transform(housing_X)

# Description before applying transforms
print(housing_y_prepared.describe())

housing_X_prepared

count      1462.000000
mean       12.42858400
std         0.261127
min        11.497200
25%        11.773200
50%        11.897200
75%        12.261100
max        13.322927
Name: SalePrice, dtype: float64

count      1462.000000
mean       12.42858400
std         0.261127
min        11.497200
25%        11.773200
50%        11.897200
75%        12.261100
max        13.322927
Name: SalePrice, dtype: float64

Out[41]: <1462x380 sparse matrix of type '<class 'numpy.float64'>'
      with 3800 stored elements in Compressed Sparse Row format>
```

What is happening in the ColumnTransformer?

1. We import the `ColumnTransformer` class.
2. We get the list of numerical column names and the list of categorical column names.
3. We construct a *ColumnTransformer*. The constructor requires a list of tuples, where each tuple contains a name, a transformer, and a list of names of columns that the transformer should be applied to.





In this example, we specify that the numerical columns should be transformed using the *num_pipeline* that we defined earlier, and the categorical columns should be transformed using a *OneHotEncoder*. Finally, we apply this *ColumnTransformer* to the housing data using *fit_transform()*.

And that's it! We have a preprocessing pipeline that takes the housing data and applies the appropriate transformations to each column.

Jupyter Notebook

You can see the instructions running in the Jupyter Notebook below:

How to Use a Jupyter Notebook?

- Click on “**Click to Launch**”  button to work and see the code running live in the notebook.
- You can click  to open the **Jupyter Notebook in a new tab**.
- Go to files and click *Download as* and then choose the format of the file to **download** . You can choose Notebook(.ipynb) to download the file and work locally or on your personal Jupyter Notebook.
-  The notebook **session expires after 30 minutes of inactivity**. It will reset if there is no interaction with the notebook for 30 consecutive minutes.

Your app can be found at: <https://1dgnrwwoynk5m-live-app.educative.run/notebooks/DataTransformation.ipynb>



Click to launch app!

