# Managing Diverse Data Types in a Dataset with COLUMNTRANSFER

In real-world applications, datasets often include a variety of feature types, such as unstructured text, categorical, and numerical columns. Preparing these mixed-feature datasets for machine learning requires a careful approach: first performing feature selection to identify the most relevant columns, and then applying distinct preprocessing methods for each data type. Managing different feature types effectively is essential to ensure that the final model can leverage the unique information each type provides.

In this article, I’ll demonstrate how to address such scenarios, sharing my experience working with a dataset that included categorical, numerical, and unstructured text columns after feature selection, and explaining the preprocessing steps used to ready the data for model training.

## **Data Preparation**

To prepare your data for machine learning, particularly when working with a dataset that includes unstructured text, categorical, and numerical features, using **ColumnTransformer** from SKLEARN.COMPOSE is an efficient approach.

To demonstrate a use case for ColumnTransformer, I’m going to use an existing Kaggle Problem (<https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews>). The reason I’m choosing this dataset is because it includes diverse datatypes in one dataset. As you can see in the highlighted columns of **Figure 1**, numerical columns are highlighted in yellow, categorical columns are in green, and orange is used to highlight unstructured text columns. Stopwords in the ReviewText column are also highlighted in bold purple.

A screenshot of a computer

Description automatically generated

**Figure 1:** Raw Dataset with different datatypes highlighted in color

To create an ML model for such a dataset without using ColumnTransformer, I’ll need to manually encode categorical features, scale numerical features, or preprocess text features. This can be complex and error-prone, especially as the dataset grows or changes. Also, when processing numerical and categorical features independently, it breaks the smooth flow of data transformation that a pipeline offers. Manual encoding creates a weak point in the ML workflow as transformed features need to be concatenated back together manually. Using ColumnTransformer for this use case is an ideal choice.

In the beginning of the Python code, import all the libraries in **Listing 1**.

### Feature Engineering

Feature engineering is the process of creating new inputs, called **features**, that help improve the performance of a machine learning model. This process involves transforming raw data into a more useful format that makes patterns easier for the model to recognize. Feature engineering includes cleaning, modifying, or adding features to a dataset so the model can understand it better.

Download this dataset to your local folder and read the CSV file using READ\_CSV command to load this dataset to a **DataFrame** in Python. Exact commands for you to use are shown in the following snippet and you can see the output in **Figure 2**.

path = r\\Womens Clothing E-Commerce Reviews.csv

df = pd.read\_csv(path)

A screenshot of a survey

Description automatically generated

**Figure 2:** Output of the READ\_CSV command snippet

Now, let’s perform common feature engineering steps to prepare the **DataFrame** for building the pipeline, as you can see in the snippet that follows.

1. Remove unnamed column 0.
2. Remove null rows from ‘Class name’, ‘Review Text’, and ‘Title’.

df.drop(df.columns[0], axis=1, inplace=True)

df = df[~df['Class Name'].isnull()]

df = df[~df['Review Text'].isnull()]

df = df[~df['Title'].isnull()]

After feature engineering, the DataFrame should look like **Figure 3**.

A screenshot of a computer

Description automatically generated

**Figure 3:** Output of feature engineering

Until now, you haven’t verified the datatypes included in the DataFrame. Let’s find out all the data types in this DataFrame using code from the next snippet. You can see the different datatypes as displayed in **Figure 4**.

df.dtypes

A computer screen shot of a computer code

Description automatically generated

**Figure 4:** The data types in the DataFrame

## **ColumnTransformer Class**

When working with a DataFrame that has a mix of data types, it’s essential to apply different preprocessing steps to numeric, categorical, and text features before feeding them into a machine learning model. Numeric features may require scaling, categorical features benefit from one-hot encoding or ordinal encoding, and text features typically need vectorization techniques like TF-IDF or word embeddings.

The **ColumnTransformer** class in Python’s SKLEARN.COMPOSE module offers an efficient way to handle datasets with heterogeneous data types, allowing you to apply multiple transformations to different feature types simultaneously within the same dataset. This approach ensures that each feature type transforms appropriately, helping better model performance. This class accepts up to six parameters, out of which, two key parameters are transformers and remainder.

### Transformers Parameter

One of the key parameters for **ColumnTransfer** class object, is a list of tuples specifying which transformer objects to apply to subsets of the data. Each tuple follows one of the below mentioned formats.

* For multiple columns: (Name, Transformer, [columns])
* For a single text column: (Name, Transformer, column)

**Name:** A string that acts as an identifier. It’s useful for setting parameters using **set\_params** and enables searching during grid search.

**Transformer:** Specifies the type of transformation. It can be an estimator supporting fit and transform, or the values **drop** or **passthrough**. In this example, I’ll be using the following transformations:

* **StandardScaler** for numeric columns
* **CountVectorizer** for unstructured text columns
* **OneHotEncoder**for categorical columns

**[*columns*]:** A list of columns from the dataset to which the transformations will apply.

For the dataset in present example, I’ll be applying transformations as explained below:

* **SimpleImputer** with constant filling for categorical columns (['Division Name', 'Department Name'])
* **CountVectorizer** with STOPWORDS for the 'Review Text' column
* **simple CountVectorizer** for the 'Title' column
* **SimpleImputer** with median filling for numerical columns (['Clothing ID', 'Age', 'Rating', 'Recommended IND', 'Positive Feedback Count'])

By using these transformations in the **ColumnTransformer** class object, you can streamline preprocessing and directly feed the transformed data into a ML model.

### Remainder Parameter

Now, let’s discuss the second key parameter, remainder. This parameter manages any remaining columns in the dataset for which you haven’t defined any specific transformations in the transformers parameter. By default, **remainder** is set to value **drop**, meaning that if you haven’t specified any value for this parameter, the pipeline won’t include the remaining columns in the final dataset. The other possible value is **passthroug**h, which allows these columns to pass through unchanged. Selecting between these options can have a significant impact on the ML model accuracy.

## **Pipeline Building**

Setting up the **ColumnTransformer** class for the Pipeline building involves the steps discussed in this next section.

### Text Preprocessing

Use **TfidfVectorizer** or **CountVectorizer** for vectorizing text data. In this case, you can remove stop words by specifying **stop\_words='english'** within the vectorizer. An English stopwords list usually includes simple words that don’t add any meaning and aren’t included in the text analysis. Examples of these words are **the**, **and** **is**, **in**, **for**, and **it**. These stopwords are frequently removed to focus on more meaningful terms when processing text data in natural language processing tasks such as text classification or sentiment analysis. You can remove these in the dataset using the **STOPWORDS.words** method.

### Categorical Encoding Using OneHotEncoder

One-Hot Encoding is a method in ML used to convert categorical data into numerical data to enhance the accuracy of the ML models. This encoding method is particularly useful when dealing with non-ordinal categorical features, where categories don’t have a natural ordering. In one-hot encoding, each unique category within a feature is transformed into a binary vector. For instance, if a feature **Fruit** has categories like **Pear**, **Apple**, and **Mango**, one-hot encoding creates three new binary columns: **Fruit\_Pear**, **Fruit\_Apple**, and **Fruit\_Mango**. When an instance has a specific category, the corresponding column is set to **1**, and the others are set to **0**.

### Numeric Scaling Using StandardScaler

StandardScaler is part of the **sklearn.preprocessing** module in Python that’s used to adjust each feature to have a mean of **0** and a standard deviation of **1**. The purpose of scaling is to ensure that all numerical features contribute equally to the model.

**Listing 2** explains the exact Python methods and commands used to define the object of the **ColumnTransformer** class to apply transformations in the example dataset.

In **Listing 2**, you’re splitting the data into training and test sets, with 80% of the data allocated for training and 20% reserved for testing, ensuring consistent results through a fixed random state.

Next, you’ll construct a pipeline to combine two main components: **ColumnTransformer** object and a **Random Forest classifier** for the prediction task. After building the pipeline, you’ll train it on the training data to create the prediction model. This structured approach ensures consistent preprocessing and model training while preventing data leakage between training and test sets, as shown in the next snippet.

X = df.drop('Class Name', axis='columns')

y=df['Class Name']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2,random\_state = 42)

pipeline = Pipeline(steps=[

('feature\_engineer', ct),

('RF', RandomForestClassifier (n\_jobs=-1,

class\_weight='balanced'))

])

pipeline.fit(X\_train, y\_train)

preds = pipeline.predict(X\_test)

print('accuracy %s' % accuracy\_score(preds, y\_test))

print(classification\_report(y\_test, preds))

## **Summary**

In this article, you saw the usage of **ColumnTransformer** in Python for applying customized transformations to numeric, categorical, and unstructured textual data separately. In turn, this allowed effective preprocessing of columns containing numeric data, categorical data, and unstructured textual data, with each type receiving diverse kinds of transformations. Further, I described how the parameter remainder is set to either pass through or drop the untransformed columns. These are flexible strategies that help prepare heterogeneous data for machine learning with the most appropriate preprocessing treatment according to each feature type, thus, increasing the effectiveness of the model.

A person in a suit

Description automatically generated

Gaurav Mittal

Gauravmittal1985@gmail.com

Gaurav Mittal is an accomplished author and international speaker, recognized for his published articles. He has spoken at global conferences and served on several judging panels. In addition to his professional achievements, Gaurav actively contributes to non-profit organizations through volunteer work. Outside of work, he enjoys spending time with his children and playing sports.

Sponsored Sidebar

Stay Connected with CODE Magazine on LinkedIn

Join our professional community on LinkedIn to stay ahead in the tech world.

Why Connect with Us?

**Exclusive Content**: Get the latest tech insights and articles.

**Industry Updates**: Stay informed with cutting-edge news.

**Networking:** Connect with peers and industry leaders.

Help us grow by sharing with colleagues and friends.

**Follow us today:** [www.linkedin.com/company/**code-magazine**](https://www.linkedin.com/company/code-magazine)

Listing 1: Importing Necessary Libraries

#Imports

import numpy as np

import pandas as pd # type: ignore

import re

import string

# Machine Learning Imports

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn import metrics

from sklearn.model\_selection import cross\_val\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

# Test Processing Imports

import nltk

from nltk.corpus import stopwords

from sklearn.preprocessing import OneHotEncoder

#word exrtaction

from sklearn.feature\_extraction.text import CountVectorizer

Listing 2: Steps for creating Data Preprocessing Pipeline

STOPWORDS = (stopwords.words('english'))

catTransformer = Pipeline(steps

'cat\_imputer',

SimpleImputer(

strategy='constant',

fill\_value='missing'

)),

('cat\_ohe',

OneHotEncoder(

handle\_unknown='ignore')

)]

)

textTransformer\_0 = Pipeline(steps=[

('text\_bow',

CountVectorizer(lowercase=True,

\token\_pattern=r"(?u)\b\w+\b",\

                              stop\_words=STOPWORDS))])

textTransformer\_1 = Pipeline(steps=[(

'text\_bow1',

CountVectorizer())

])

numeric\_features = [

'Clothing ID',

'Age',

'Rating',

'Recommended IND',

'Positive Feedback Count']

numTransformer = Pipeline(steps=[(

'imputer',

SimpleImputer(strategy='median')),

('scaler', StandardScaler())

])

ct = ColumnTransformer(

transformers=[

('cat', catTransformer,

['Division Name','Department Name']),

('num', numTransformer, numeric\_features),

('text1', textTransformer\_0, 'Review Text'),

('text2', textTransformer\_1, 'Title')

],remainder='passthrough')

# If no value is specified in remainder,

# columns without transformations will be

# removed from the dataset.

# 'drop' is the default value for remainder.