Introduction

**İşlem hatları (Pipelines)**, veri ön işleme ve modelleme kodunuzu düzenli tutmanın basit bir yoludur. Özellikle, bir işlem hattı ön işleme ve modelleme adımlarını tek bir adım gibi kullanabileceğiniz bir paket halinde birleştirir.

Birçok veri bilimcisi, işlem hatları olmadan modelleri bir araya getirse de, işlem hatlarının bazı önemli faydaları vardır. Bunlar:

* **Daha Temiz Kod**: Ön işlemenin her adımında verileri hesaba katmak karmaşık hale gelebilir. Bir işlem hattı ile her adımda eğitim ve doğrulama verilerinizi manuel olarak takip etmek zorunda kalmazsınız.
* **Daha Az Hata**: Bir adımı yanlış uygulama veya bir ön işleme adımını unutma olasılığı daha düşüktür.
* **Üretime Geçirmesi Daha Kolay**: Bir modeli prototipten büyük ölçekte konuşlandırılabilir bir şeye dönüştürmek şaşırtıcı derecede zor olabilir. Burada ilgili birçok konuya girmeyeceğiz, ancak işlem hatları yardımcı olabilir.
* **Model Doğrulaması için Daha Fazla Seçenek**: Bir sonraki eğitimde, çapraz doğrulamayı ele alan bir örnek göreceksiniz.

# Example

Önceki eğitimde olduğu gibi, Melbourne Konut veri kümesiyle çalışacağız.

Veri yükleme adımına odaklanmayacağız. Bunun yerine, X\_train, X\_valid, y\_train ve y\_valid veri kümelerinde eğitim ve doğrulama verilerine zaten sahip olduğunuzu hayal edebilirsiniz.

import pandas as pd

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

*# Read the data*

data = pd.read\_csv('../input/melbourne-housing-snapshot/melb\_data.csv')

*# Separate target from predictors*

y = data.Price

X = data.drop(['Price'], axis=1)

*# Divide data into training and validation subsets*

X\_train\_full, X\_valid\_full, y\_train, y\_valid = train\_test\_split(X, y, train\_size=0.8, test\_size=0.2,

random\_state=0)

*# "Cardinality" means the number of unique values in a column*

*# Select categorical columns with relatively low cardinality (convenient but arbitrary)*

categorical\_cols = [cname for cname **in** X\_train\_full.columns if X\_train\_full[cname].nunique() < 10 **and**

X\_train\_full[cname].dtype == "object"]

*# Select numerical columns*

numerical\_cols = [cname for cname **in** X\_train\_full.columns if X\_train\_full[cname].dtype **in** ['int64', 'float64']]

*# Keep selected columns only*

my\_cols = categorical\_cols + numerical\_cols

X\_train = X\_train\_full[my\_cols].copy()

X\_valid = X\_valid\_full[my\_cols].copy()

Aşağıda, head() metoduyla eğitim verilerine bir göz atıyoruz. Verilerin hem kategorik veriler hem de eksik değerler içeren sütunlar barındırdığını fark edin. Bir işlem hattıyla (pipeline) her ikisiyle de başa çıkmak çok kolay!

X\_train.head()

| Type | Method | Regionname | Rooms | Distance | Postcode | Bedroom2 | Bathroom | Car | Landsize | BuildingArea | YearBuilt | Lattitude | Longtitude | Propertycount |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 12167 | u | S | Southern Metropolitan | 1 | 5.0 | 3182.0 | 1.0 | 1.0 | 1.0 | 0.0 | NaN | 1940.0 | -37.85984 | 144.9867 | 13240.0 |
| 6524 | h | SA | Western Metropolitan | 2 | 8.0 | 3016.0 | 2.0 | 2.0 | 1.0 | 193.0 | NaN | NaN | -37.85800 | 144.9005 | 6380.0 |
| 8413 | h | S | Western Metropolitan | 3 | 12.6 | 3020.0 | 3.0 | 1.0 | 1.0 | 555.0 | NaN | NaN | -37.79880 | 144.8220 | 3755.0 |
| 2919 | u | SP | Northern Metropolitan | 3 | 13.0 | 3046.0 | 3.0 | 1.0 | 1.0 | 265.0 | NaN | 1995.0 | -37.70830 | 144.9158 | 8870.0 |
| 6043 | h | S | Western Metropolitan | 3 | 13.3 | 3020.0 | 3.0 | 1.0 | 2.0 | 673.0 | 673.0 | 1970.0 | -37.76230 | 144.8272 | 4217.0 |

Boru hattının tamamını üç aşamada inşa ediyoruz.

### **Step 1: Define Preprocessing Steps[¶](https://www.kaggle.com/code/alexisbcook/pipelines" \l "Step-1:-Define-Preprocessing-Steps" \t "_self)**

Tıpkı bir **pipeline**'ın (işlem hattı), ön işleme ve modelleme adımlarını bir araya getirmesi gibi, biz de farklı ön işleme adımlarını bir araya getirmek için **ColumnTransformer** sınıfını kullanıyoruz. Aşağıdaki kod:

* **Sayısal** verilerdeki eksik değerleri atar (impute eder) ve
* **Kategorik** verilerdeki eksik değerleri atar ve bunlara tekil sıcak kodlama (one-hot encoding) uygular.
* from sklearn.compose import ColumnTransformer
* from sklearn.pipeline import Pipeline
* from sklearn.impute import SimpleImputer
* from sklearn.preprocessing import OneHotEncoder
* *# Preprocessing for numerical data*
* numerical\_transformer = SimpleImputer(strategy='constant')
* *# Preprocessing for categorical data*
* categorical\_transformer = Pipeline(steps=[
* ('imputer', SimpleImputer(strategy='most\_frequent')),
* ('onehot', OneHotEncoder(handle\_unknown='ignore'))
* ])
* *# Bundle preprocessing for numerical and categorical data*
* preprocessor = ColumnTransformer(
* transformers=[
* ('num', numerical\_transformer, numerical\_cols),
* ('cat', categorical\_transformer, categorical\_cols)
* ])

### **Step 2: Define the Model**[**¶**](https://www.kaggle.com/code/alexisbcook/pipelines#Step-2:-Define-the-Model)

Daha sonra, bildiğimiz RandomForestRegressor sınıfıyla rastgele bir orman modeli tanımlıyoruz.

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n\_estimators=100, random\_state=0)

### **Step 3: Create and Evaluate the Pipeline[¶](https://www.kaggle.com/code/alexisbcook/pipelines" \l "Step-3:-Create-and-Evaluate-the-Pipeline" \t "_self)**

Son olarak, ön işleme ve modelleme adımlarını bir araya getiren bir pipeline (işlem hattı) tanımlamak için **Pipeline** sınıfını kullanıyoruz. Dikkat edilmesi gereken birkaç önemli nokta şunlardır:

* Pipeline ile, eğitim verilerini ön işleme tabi tutuyor ve modeli tek bir kod satırında eğitiyoruz. (Bunun aksine, bir pipeline olmadan, değer atama, tekil sıcak kodlama ve model eğitimini ayrı adımlarda yapmamız gerekir. Bu, hem sayısal hem de kategorik değişkenlerle uğraşmak zorunda kaldığımızda özellikle karmaşık bir hale gelir!)
* Pipeline ile, işlenmemiş özellikleri X\_valid değişkeniyle predict() komutuna sağlıyoruz ve pipeline, tahminler oluşturmadan önce özellikleri otomatik olarak ön işliyor. (Ancak, bir pipeline olmadan, tahmin yapmadan önce doğrulama verilerini ön işlemden geçirmeyi hatırlamamız gerekir.)
* from sklearn.metrics import mean\_absolute\_error
* *# Bundle preprocessing and modeling code in a pipeline*
* my\_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
* ('model', model)
* ])
* *# Preprocessing of training data, fit model*
* my\_pipeline.fit(X\_train, y\_train)
* *# Preprocessing of validation data, get predictions*
* preds = my\_pipeline.predict(X\_valid)
* *# Evaluate the model*
* score = mean\_absolute\_error(y\_valid, preds)
* print('MAE:', score)

MAE: 160679.18917034855

# Conclusion

İşlem hatları (Pipelines), makine öğrenmesi kodunu düzenlemek ve hataları önlemek için oldukça değerlidir; özellikle karmaşık veri ön işleme adımları içeren iş akışları için son derece faydalıdır.