# **Machine Learning Workflow**

## **PlanMachine Learning Workflow**

## **Plan**[**¶**](https://kitt.lewagon.com/camps/1917/lectures/content/05-ML_08-Workflow.html#Plan)

## **Model Selection Tips 💡**

## **Pipelines 🔥**

## **Preprocessing Pipes**

## **Pipelines → → →**

## **Column Transformers ⑂**

## **Custom Transformers →**

## **Feature Unions ||**

## **Full Pipes (Preprocessing + Models)**

## **Surprise 🤗**

## **1. Model Selection**

## **Let's take a step back: which models have we seen so far?**

## **1️⃣ Regression models are parametric**

## **^**

## **y**

## **=**

## **f**

## **β**

## **(**

## **X**

## **)**

## 

## **An arbitrarily large number**

## **n**

## **of datapoints can be modeled with few**

## **β**

## **parameters**

## ***Note: Neural Networks are also parametrics models (See Deep Learning)***

## **✅ Fast to train, even on large datasets with Stochastic Gradient Descent**

## **❗️ Requires prior assumptions**

## **f**

## **about the structure of the data; may not find complex patterns, unless given complex features**

## **2️⃣ KNN, kernel-SVM are non-parametric**

## **No prior assumptions about the data structure are needed**

## **Possibly many parameters to learn (not known beforehand)**

## **e.g. KNN .fit stores the *whole dataset***

## **e.g. rbf-SVM .fit must compute a Kernel between *each pair* of datapoints**

## ***Note: Trees are also non-parametric models (See Ensemble Methods)***

## **✅ Can find complex features for you!**

## **❗️ Harder to train on large datasets and prone to overfitting**

## 

# **2. Pipelines**

## **📚** [**sklearn - Pipeline and composite estimators**](https://scikit-learn.org/stable/modules/compose.html)

## **📚** [**sklearn.pipeline**](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.pipeline)

## 

## **A *Pipeline* is a chain of operations in a Machine Learning project (preprocessing, training, predicting, etc.)**

## **Pipelines are powerful because they:**

## **🏄🏻‍♀️ make your workflow much easier to read and understand**

## **💪 enforce the implementation and order of steps in your project**

## **⚙️ make your work reproducible and deployable**

## 

## **2.1 Preprocessing Pipelines**

## **🎯 We are going to predict the charges of a health insurance contract based on various features using the following dataset.**

## **💾 Download the dataset** [**here**](https://wagon-public-datasets.s3.amazonaws.com/data_workflow.csv)

## **data.head(5)**

## 

|  | **age** | **bmi** | **children** | **smoker** | **region** | **charges** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **19.0** | **27.900** | **0** | **True** | **southwest** | **16884.92400** |
| **1** | **18.0** | **33.770** | **1** | **False** | **southeast** | **1725.55230** |
| **2** | **NaN** | **33.000** | **3** | **False** | **southeast** | **4449.46200** |
| **3** | **33.0** | **22.705** | **0** | **False** | **northwest** | **21984.47061** |
| **4** | **32.0** | **28.880** | **0** | **False** | **northwest** | **3866.85520** |

## **data.shape**

## 

## **(1338, 6)**

## ***# Defining the features and the target***

## 

## **X = data.drop(columns='charges')**

## **y = data['charges']**

## 

## ***# Train-Test split***

## 

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

## 

## **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20)**

## **X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape**

## 

## **((1070, 5), (268, 5), (1070,), (268,))**

## **✏️ Today's challenges:**

## ***Impute* missing values**

## **Preprocessing:**

## ***Scale* numerical features**

## ***Encode* categorical features**

## ***Fine-tune* your ML model *and* the preprocessing steps...**

## **... 🔥 in one cell ! 🔥**

### **a) Pipeline → → →**

## **A Pipeline essentially chains multiple steps in sequence (e.g. *imputing* then *scaling*)**

## **📚** [**sklearn.pipeline.Pipeline**](https://scikit-learn.org/0.16/modules/generated/sklearn.pipeline.Pipeline.html)

## **from sklearn.pipeline import Pipeline**

## 

## ***# Preprocess "age"***

## **from sklearn.pipeline import Pipeline**

## **from sklearn.impute import SimpleImputer**

## **from sklearn.preprocessing import StandardScaler**

## 

## ***# Build the pipeline with the different steps***

## **pipeline = Pipeline([**

## **('imputer', SimpleImputer(strategy="median")),**

## **('standard\_scaler', StandardScaler())**

## **])**

## 

## **pipeline.fit(X\_train[['age']])**

## **pipeline.transform(X\_train[['age']])**

## 

## **array([[ 1.03287039],**

## **[-1.45497346],**

## **[ 1.1750329 ],**

## **...,**

## **[ 0.25097661],**

## **[-0.17551091],**

## **[-1.2417297 ]])**

## ***# Show the different steps of the pipeline***

## **pipeline**

## 

## **Pipeline**

## **SimpleImputer**

## **StandardScaler**

### **b) Column Transformer ⑂**

## **Column Transformers allow you to apply specific changes to specific columns in parallel**

## **📚** [**sklearn.compose.ColumnTransformer**](https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html)

## **from sklearn.compose import ColumnTransformer**

## 

## **💻 Let's perform the following operations in parallel:**

## **🔢 *Impute* then *scale* numerical values**

## **🔠 *Encode* categorical values**

## 

## **☝️ Notice how a Pipeline object can be passed into a ColumnTransformer!**

## **from sklearn.compose import ColumnTransformer**

## 

## **from sklearn.pipeline import Pipeline**

## **from sklearn.impute import SimpleImputer**

## **from sklearn.preprocessing import StandardScaler**

## 

## **from sklearn.preprocessing import OneHotEncoder**

## 

## 

## ***# Impute then scale numerical values:***

## **num\_transformer = Pipeline([**

## **('imputer', SimpleImputer(strategy="mean")),**

## **('standard\_scaler', StandardScaler())**

## **])**

## 

## ***# Encode categorical values***

## **cat\_transformer = OneHotEncoder(handle\_unknown='ignore')**

## 

## ***# Parallelize "num\_transformer" and "cat\_transfomer"***

## **preprocessor = ColumnTransformer([**

## **('num\_transformer', num\_transformer, ['age', 'bmi']),**

## **('cat\_transformer', cat\_transformer, ['smoker', 'region'])**

## **])**

## 

## ***# Visualizing Pipelines in HTML***

## **from sklearn import set\_config; set\_config(display='diagram')**

## **preprocessor**

## 

## **ColumnTransformer**

## **num\_transformer**

## **SimpleImputer**

## **StandardScaler**

## **cat\_transformer**

## **OneHotEncoder**

## **X\_train\_transformed = preprocessor.fit\_transform(X\_train)**

## 

## **print("Original training set")**

## **display(X\_train.head(3))**

## 

## **print("Preprocessed training set")**

## **display(pd.DataFrame(X\_train\_transformed).head(3))**

## 

## **Original training set**

## 

|  | **age** | **bmi** | **children** | **smoker** | **region** |
| --- | --- | --- | --- | --- | --- |
| **162** | **54.0** | **39.60** | **1** | **False** | **southwest** |
| **410** | **19.0** | **17.48** | **0** | **False** | **northwest** |
| **639** | **56.0** | **33.66** | **4** | **False** | **southeast** |

## **Preprocessed training set**

## 

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1.032979** | **1.456688** | **1.0** | **0.0** | **0.0** | **0.0** | **0.0** | **1.0** |
| **1** | **-1.454870** | **-2.170790** | **1.0** | **0.0** | **0.0** | **1.0** | **0.0** | **0.0** |
| **2** | **1.175141** | **0.482582** | **1.0** | **0.0** | **0.0** | **0.0** | **1.0** | **0.0** |

## **😱 Where are the columns' names?**

## **😉 Don't worry and stay tuned to scikit-learn updates!**

## [**scikit-learn.org/stable/whats\_new.html**](https://scikit-learn.org/stable/whats_new.html)

### **🚀 get\_feature\_names\_out() 🚀**

## **New in scikit-learn 1.0.2 *(September 2021)***

## **✅ This new method helps retrieve the names of the features which went through some transformations like StandardScaler or OheHotEncoder**

## **❌ Not all the transformers in Scikit-Learn have this new method**

## **New in scikit-learn 1.1.3: *(October 2022)***

## **✅ *ALL* the transformers have this method!**

## ***# Get your features' names***

## **preprocessor.get\_feature\_names\_out()**

## 

## **array(['num\_transformer\_\_age', 'num\_transformer\_\_bmi',**

## **'cat\_transformer\_\_smoker\_False', 'cat\_transformer\_\_smoker\_True',**

## **'cat\_transformer\_\_region\_northeast',**

## **'cat\_transformer\_\_region\_northwest',**

## **'cat\_transformer\_\_region\_southeast',**

## **'cat\_transformer\_\_region\_southwest'], dtype=object)**

## **pd.DataFrame(**

## **X\_train\_transformed,**

## **columns=preprocessor.get\_feature\_names\_out()**

## **).head()**

## 

|  | **num\_transformer\_\_age** | **num\_transformer\_\_bmi** | **cat\_transformer\_\_smoker\_False** | **cat\_transformer\_\_smoker\_True** | **cat\_transformer\_\_region\_northeast** | **cat\_transformer\_\_region\_northwest** | **cat\_transformer\_\_region\_southeast** | **cat\_transformer\_\_region\_southwest** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1.032979** | **1.456688** | **1.0** | **0.0** | **0.0** | **0.0** | **0.0** | **1.0** |
| **1** | **-1.454870** | **-2.170790** | **1.0** | **0.0** | **0.0** | **1.0** | **0.0** | **0.0** |
| **2** | **1.175141** | **0.482582** | **1.0** | **0.0** | **0.0** | **0.0** | **1.0** | **0.0** |
| **3** | **-0.815138** | **0.157880** | **0.0** | **1.0** | **0.0** | **0.0** | **1.0** | **0.0** |
| **4** | **-0.601894** | **-0.148783** | **0.0** | **1.0** | **0.0** | **0.0** | **1.0** | **0.0** |

## **🤔 What happened to the children column? What if we want to keep it untouched?**

#### **👉 remainder=passthrough**

## **preprocessor = ColumnTransformer([**

## **('num\_transformer', num\_transformer, ['age','bmi']),**

## **('cat\_transformer', cat\_transformer, ['region','smoker'])],**

## **remainder='passthrough'**

## **)**

## 

## **preprocessor**

## 

## **ColumnTransformer**

## **num\_transformer**

## **SimpleImputer**

## **StandardScaler**

## **cat\_transformer**

## **OneHotEncoder**

## **remainder**

## **passthrough**

## **pd.DataFrame(preprocessor.fit\_transform(X\_train),**

## **columns=preprocessor.get\_feature\_names\_out()).head(3)**

## 

|  | **num\_transformer\_\_age** | **num\_transformer\_\_bmi** | **cat\_transformer\_\_region\_northeast** | **cat\_transformer\_\_region\_northwest** | **cat\_transformer\_\_region\_southeast** | **cat\_transformer\_\_region\_southwest** | **cat\_transformer\_\_smoker\_False** | **cat\_transformer\_\_smoker\_True** | **remainder\_\_children** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1.032979** | **1.456688** | **0.0** | **0.0** | **0.0** | **1.0** | **1.0** | **0.0** | **1.0** |
| **1** | **-1.454870** | **-2.170790** | **0.0** | **1.0** | **0.0** | **0.0** | **1.0** | **0.0** | **0.0** |
| **2** | **1.175141** | **0.482582** | **0.0** | **0.0** | **1.0** | **0.0** | **1.0** | **0.0** | **4.0** |

### **c) Custom: Function Transformer →**

## **📚** [**sklearn.preprocessing.FunctionTransformer**](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.FunctionTransformer.html)

## **from sklearn.preprocessing import FunctionTransformer**

## 

## **⚒️ Function Transformers enable you to encapsulate a *Python* function within a scikitTransformer (→) Object**

## **💪 They can be used with either Pipelines (→ → →) or ColumnTransformers (⑂)**

## 

## **👆 If you want to use your own transformer in a Pipeline or a ColumnTransformer *(not one already available in Sklearn)*, you must encapsulate your function within a FunctionTransformer.**

## **from sklearn.preprocessing import FunctionTransformer**

## 

## ***# Create a transformer that compresses data to 2 digits (for instance!)***

## ***# rounder = FunctionTransformer(np.round)***

## 

## ***# We can use a lambda function for more customizable functions***

## **rounder = FunctionTransformer(lambda array: np.round(array, decimals=2))**

## 

## ***# Add it at the end of our numerical transformer***

## **num\_transformer = Pipeline([**

## **('imputer', SimpleImputer()),**

## **('scaler', StandardScaler()),**

## **('rounder', rounder)])**

## 

## ***# Encode categorical values***

## **cat\_transformer = OneHotEncoder(drop='if\_binary',**

## **handle\_unknown='ignore')**

## 

## **preprocessor = ColumnTransformer([**

## **('num\_transformer', num\_transformer, ['bmi', 'age']),**

## **('cat\_transformer', cat\_transformer, ['region', 'smoker'])],**

## **remainder='passthrough')**

## **preprocessor**

## 

## **ColumnTransformer**

## **num\_transformer**

## **SimpleImputer**

## **StandardScaler**

## **FunctionTransformer**

## **cat\_transformer**

## **OneHotEncoder**

## **remainder**

## **passthrough**

## **pd.DataFrame(preprocessor.fit\_transform(X\_train)).head(3)**

## 

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1.46** | **1.03** | **0.0** | **0.0** | **0.0** | **1.0** | **0.0** | **1.0** |
| **1** | **-2.17** | **-1.45** | **0.0** | **1.0** | **0.0** | **0.0** | **0.0** | **0.0** |
| **2** | **0.48** | **1.18** | **0.0** | **0.0** | **1.0** | **0.0** | **0.0** | **4.0** |

## **❗️ FunctionTransformer only works for stateless transformations ❗️**

## **👨🏻‍🏫 stateless transformations are transformations which cannot *store* information during .fit(X\_train) that would be used for the .transform(X\_test).**

## **👍 Since a stateless transformation doesn't learn anything, fitting it is impossible, it does nothing other than transform!**

## **✅ FunctionTransformer is compatible with stateless transformations.**

## ***Examples* of transformations which don't "learn" anything:**

## **X**

## **→**

## **l**

## **o**

## **g**

## **(**

## **X**

## **)**

## 

## **(**

## **X**

## **1**

## **,**

## **X**

## **2**

## **)**

## **→**

## **X**

## **1**

## **+**

## **5**

## **X**

## **2**

## 

## **👩‍🏫 stateful transformations are transformations which *store* information during .fit(X\_train). This information is re-used for .transform(X\_test).**

## ***Examples* of transformations which "learn" something:**

## **X**

## **t**

## **r**

## **a**

## **i**

## **n**

## **→**

## **S**

## **t**

## **a**

## **n**

## **d**

## **a**

## **r**

## **d**

## **S**

## **c**

## **a**

## **l**

## **e**

## **r**

## **(**

## **X**

## **t**

## **r**

## **a**

## **i**

## **n**

## **)**

## **learns**

## **μ**

## **t**

## **r**

## **a**

## **i**

## **n**

## **and**

## **σ**

## **t**

## **r**

## **a**

## **i**

## **n**

## **X**

## **t**

## **r**

## **a**

## **i**

## **n**

## **→**

## **M**

## **i**

## **n**

## **M**

## **a**

## **x**

## **S**

## **c**

## **a**

## **l**

## **e**

## **r**

## **(**

## **X**

## **t**

## **r**

## **a**

## **i**

## **n**

## **)**

## **learns**

## **X**

## **(**

## **m**

## **i**

## **n**

## **)**

## **t**

## **r**

## **a**

## **i**

## **n**

## **and**

## **X**

## **(**

## **m**

## **a**

## **x**

## **)**

## **t**

## **r**

## **a**

## **i**

## **n**

## **❌ FunctionTransformer is not compatible with stateful transformations**

## **🌶️ We will have to code our own Class to use FunctionTransformer with stateful transformations!**

#### **🕵🏻‍♂️ Transformers under the hood**

## **from sklearn.base import TransformerMixin, BaseEstimator**

## 

## **class MyCustomTranformer(TransformerMixin, BaseEstimator):**

## ***# BaseEstimator generates the get\_params() and set\_params() methods that all Pipelines require***

## ***# TransformerMixin creates the fit\_transform() method from fit() and transform()***

## 

## **def \_\_init\_\_(self):**

## **pass**

## 

## **def fit(self, X, y=None):**

## ***# Here you store what needs to be stored/learned during .fit(X\_train) as instance attributes***

## ***# Return "self" to allow chaining .fit().transform()***

## **pass**

## 

## **def transform(self, X, y=None):**

## ***# Return the result as a DataFrame for an integration into the ColumnTransformer***

## **pass**

## 

## **my\_transformer = MyCustomTranformer()**

## **my\_transformer.fit(X\_train)**

## **my\_transformer.transform(X\_train)**

## **my\_transformer.transform(X\_test)**

## 

## **💻 *More in today's challenges* 💻**

### **d) FeatureUnion | |**

## **FeatureUnion applies a list of transformer objects in parallel to the input data, then concatenates the results. This is useful to combine several feature extraction mechanisms into a single transformer**

## **📚** [**sklearn.pipeline.FeatureUnion**](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.FeatureUnion.html)

## **👌 Useful to create entirely new features!**

## ***Example*: let's build and add a new feature called bmi\_age\_ratio**

## **X\_train.head(3)**

## 

|  | **age** | **bmi** | **children** | **smoker** | **region** |
| --- | --- | --- | --- | --- | --- |
| **162** | **54.0** | **39.60** | **1** | **False** | **southwest** |
| **410** | **19.0** | **17.48** | **0** | **False** | **northwest** |
| **639** | **56.0** | **33.66** | **4** | **False** | **southeast** |

## **from sklearn.pipeline import FeatureUnion**

## 

## ***# Create a custom transformer that multiplies/divides two columns***

## ***# Notice that we are creating this new feature completely randomly just as an example***

## **bmi\_age\_ratio\_constructor = FunctionTransformer(lambda df: pd.DataFrame(df["bmi"] / df["age"]))**

## 

## **union = FeatureUnion([**

## **('preprocess', preprocessor), *# columns 0-7***

## **('bmi\_age\_ratio', bmi\_age\_ratio\_constructor) *# new column 8***

## **])**

## 

## **union**

## 

## **FeatureUnion**

## **preprocess**

## **num\_transformer**

## **SimpleImputer**

## **StandardScaler**

## **FunctionTransformer**

## **cat\_transformer**

## **OneHotEncoder**

## **remainder**

## **passthrough**

## **bmi\_age\_ratio**

## **FunctionTransformer**

## **pd.DataFrame(union.fit\_transform(X\_train)).head(1)**

## 

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1.46** | **1.03** | **0.0** | **0.0** | **0.0** | **1.0** | **0.0** | **1.0** | **0.733333** |

### **Building your preprocessor with make\_\*\*\* shortcuts ⚡️**

## **from sklearn.pipeline import Pipeline**

## **from sklearn.pipeline import FeatureUnion**

## **from sklearn.compose import ColumnTransformer**

## 

## **There are equivalent transformers using the syntax make\_\*\*\*👇**

## **from sklearn.pipeline import make\_pipeline**

## **from sklearn.pipeline import make\_union**

## **from sklearn.compose import make\_column\_transformer**

## 

## **Pipeline([**

## **('my\_name\_for\_the\_imputer', SimpleImputer()),**

## **('my\_name\_for\_the\_scaler', StandardScaler())**

## **])**

## 

## **⇔**

## **make\_pipeline(SimpleImputer(), StandardScaler())**

## 

## **Pipeline**

## **SimpleImputer**

## **StandardScaler**

## **num\_transformer = make\_pipeline(SimpleImputer(), StandardScaler())**

## **cat\_transformer = OneHotEncoder()**

## 

## **preproc\_basic = make\_column\_transformer(**

## **(num\_transformer, ['age', 'bmi']),**

## **(cat\_transformer, ['smoker', 'region']),**

## **remainder='passthrough'**

## **)**

## 

## **preproc\_full = make\_union(preproc\_basic, bmi\_age\_ratio\_constructor)**

## 

## **preproc\_full**

## 

## **FeatureUnion**

## **columntransformer**

## **pipeline**

## **SimpleImputer**

## **StandardScaler**

## **onehotencoder**

## **OneHotEncoder**

## **remainder**

## **passthrough**

## **functiontransformer**

## **FunctionTransformer**

#### **🍒 make\_column\_selector selects features automatically based on dtype**

## **from sklearn.compose import make\_column\_selector**

## 

## **num\_col = make\_column\_selector(dtype\_include=['float64'])**

## **cat\_col = make\_column\_selector(dtype\_include=['object','bool'])**

## 

## **X\_train.dtypes**

## 

## **age float64**

## **bmi float64**

## **children int64**

## **smoker bool**

## **region object**

## **dtype: object**

## **🎉 Complete preprocessing pipeline 🎉**

## **from sklearn.compose import make\_column\_selector**

## 

## **num\_transformer = make\_pipeline(SimpleImputer(), StandardScaler())**

## **num\_col = make\_column\_selector(dtype\_include=['float64'])**

## 

## **cat\_transformer = OneHotEncoder()**

## **cat\_col = make\_column\_selector(dtype\_include=['object','bool'])**

## 

## **preproc\_basic = make\_column\_transformer(**

## **(num\_transformer, num\_col),**

## **(cat\_transformer, cat\_col),**

## **remainder='passthrough'**

## **)**

## 

## **preproc\_full = make\_union(preproc\_basic, bmi\_age\_ratio\_constructor)**

## 

## **preproc\_full**

## 

## **FeatureUnion**

## **columntransformer**

## **pipeline**

## **SimpleImputer**

## **StandardScaler**

## **onehotencoder**

## **OneHotEncoder**

## **remainder**

## **passthrough**

## **functiontransformer**

## **FunctionTransformer**

## **2.2 Including models in Pipelines**

## **Model objects can be plugged into Pipelines**

## **Pipelines inherit the methods of the last object in the sequence**

## **Transformers: fit and transform**

## **Models: fit, score, predict, etc.**

## 

## **When executing the pipeline.fit method, the transformer's .fit and .transform methods will be called sequentially, and the model will be trained.**

## **At this stage, all transformers' variables are saved into the memory of the pipeline**

## **When executing the pipeline.predict method, only the transformer's .transform method will be called, using the variables learned during the original fit**

### **a) Full pipeline**

## **from sklearn.linear\_model import Ridge**

## 

## ***# Preprocessor***

## **num\_transformer = make\_pipeline(SimpleImputer(), StandardScaler())**

## **cat\_transformer = OneHotEncoder()**

## 

## **preproc = make\_column\_transformer(**

## **(num\_transformer, make\_column\_selector(dtype\_include=['float64'])),**

## **(cat\_transformer, make\_column\_selector(dtype\_include=['object','bool'])),**

## **remainder='passthrough'**

## **)**

## 

## ***# Add estimator***

## **pipeline = make\_pipeline(preproc, Ridge())**

## **pipeline**

## 

## **Pipeline**

## **columntransformer: ColumnTransformer**

## **pipeline**

## **SimpleImputer**

## **StandardScaler**

## **onehotencoder**

## **OneHotEncoder**

## **remainder**

## **passthrough**

## **Ridge**

## ***# Train Pipeline***

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

## 

## ***# Make predictions***

## **pipeline.predict(X\_test.iloc[0:1])**

## 

## ***# Score model***

## **pipeline.score(X\_test,y\_test)**

## 

## **0.7473478157212925**

### **b) Cross-validate a Pipeline**

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

## 

## ***# Cross-validate Pipeline***

## **cross\_val\_score(pipeline, X\_train, y\_train, cv=5, scoring='r2').mean()**

## 

## **0.7434317676218065**

### **c) Grid Search a Pipeline**

## ***Grid Searching* allows you to check which combination of preprocessing/modeling hyperparameters works best.**

## **It is possible to *Grid Search* the hyperparameters of any component of the Pipeline**

## **Typical Sklearn syntax: step\_name\_\_transformer\_name\_\_hyperparameter\_name**

## **To check which hyperparameters of the pipeline can be optimized: pipeline.get\_params()**

## ***# Which parameters of the pipeline are GridSearch-able?***

## **pipeline.get\_params()**

## 

## **from sklearn.model\_selection import GridSearchCV**

## 

## **grid\_search = GridSearchCV(**

## **pipeline,**

## **param\_grid={**

## ***# Access any component of the Pipeline***

## ***# and any available hyperparamater you want to optimize***

## **'columntransformer\_\_pipeline\_\_simpleimputer\_\_strategy': ['mean', 'median'],**

## **'ridge\_\_alpha': [0.1, 0.5, 1, 5, 10]**

## **},**

## **cv=5,**

## **scoring="r2")**

## 

## **grid\_search.fit(X\_train, y\_train)**

## 

## **grid\_search.best\_params\_**

## 

## **{'columntransformer\_\_pipeline\_\_simpleimputer\_\_strategy': 'mean',**

## **'ridge\_\_alpha': 1}**

## **💾 Let's save the pipelined model with the best hyperparameters.**

## **pipeline\_tuned = grid\_search.best\_estimator\_**

## **pipeline\_tuned**

## 

## **Pipeline**

## **columntransformer: ColumnTransformer**

## **pipeline**

## **SimpleImputer**

## **StandardScaler**

## **onehotencoder**

## **OneHotEncoder**

## **remainder**

## **passthrough**

## **Ridge**

## **🔮 We can use this "best" model for predictions without re-training it!**

## **pipeline\_tuned.predict(X\_test[0:1])**

## 

## **array([10216.56989159])**

### **d) Caching to avoid repeated computations**

## **😠 Are your preprocessing steps too long to run?**

## **🪄 You can use caching techniques!**

## **from tempfile import mkdtemp**

## **from shutil import rmtree**

## 

## ***# Create a temp folder***

## **cachedir = mkdtemp()**

## 

## ***# Instantiate the Pipeline with the cache parameter***

## **pipeline = Pipeline(steps, memory=cachedir)**

## 

## ***# Clear the cache directory after the cross-validation***

## **rmtree(cachedir)**

## 

## **With the parameter memory=cachedir, preproc parameters can be cached into memory.**

## **Avoid recalculating all of the parameters during CrossValidation or GridSearchCV on estimator hyperparams only**

## **Helpful only when the transformer's .fit time is long and the dataset is very large**

### **e) Debug your pipe**

## ***# Access the components of a Pipeline with `named\_steps`***

## **pipeline\_tuned.named\_steps.keys()**

## 

## **dict\_keys(['columntransformer', 'ridge'])**

## ***# Check intermediate steps***

## **print("Before preprocessing, X\_train.shape = ")**

## **print(X\_train.shape)**

## **print("After preprocessing, X\_train\_preprocessed.shape = ")**

## **pipeline\_tuned.named\_steps["columntransformer"].fit\_transform(X\_train).shape**

## 

## **Before preprocessing, X\_train.shape =**

## **(1070, 5)**

## **After preprocessing, X\_train\_preprocessed.shape =**

## 

## **(1070, 9)**

### **f) Exporting models/Pipelines**

## **💾 You can export your final model/pipeline as a pickle file**

## **👉The file can then be loaded back into a notebook or deployed on a server (see ML Opsmodule).**

## **import pickle**

## 

## ***# Export Pipeline as pickle file***

## **with open("pipeline.pkl", "wb") as file:**

## **pickle.dump(pipeline\_tuned, file)**

## 

## ***# Load Pipeline from pickle file***

## **my\_pipeline = pickle.load(open("pipeline.pkl","rb"))**

## 

## **my\_pipeline.score(X\_test, y\_test)**

## 

## **0.7473478157212925**

# **3. Surprise 🎉**

# **AutoML**

## **TPOT**

## **The Tree-based Pipeline Optimization Tool (TPOT) is an automated Machine Learning tool that optimizes Machine Learning Pipelines**

## 

## **📚 More details available in the** [**TPOT documentation**](http://epistasislab.github.io/tpot/)

## **Installation**

## 

## **pip install TPOT**

## 

## **import os**

## **from tpot import TPOTRegressor**

## 

## **X\_train\_preproc = preproc\_basic.fit\_transform(X\_train)**

## **X\_test\_preproc = preproc\_basic.transform(X\_test)**

## 

## ***# Instantiate TPOTClassifier***

## **tpot = TPOTRegressor(generations=4, population\_size=20, verbosity=2, scoring='r2', n\_jobs=-1, cv=2)**

## 

## ***# Process autoML with TPOT***

## **tpot.fit(X\_train\_preproc, y\_train)**

## 

## ***# Print score***

## **print(tpot.score(X\_test\_preproc, y\_test))**

## 

## **Generation 1 - Current best internal CV score: 0.8517440046999218**

## 

## **Generation 2 - Current best internal CV score: 0.853008927910814**

## 

## **Generation 3 - Current best internal CV score: 0.853008927910814**

## 

## **Generation 4 - Current best internal CV score: 0.8558530771102855**

## 

## **Best pipeline: RidgeCV(GradientBoostingRegressor(input\_matrix, alpha=0.9, learning\_rate=0.01, loss=ls, max\_depth=3, max\_features=0.6000000000000001, min\_samples\_leaf=14, min\_samples\_split=13, n\_estimators=100, subsample=0.55))**

## **0.872811877434264**

## 

## **/Users/davywai/.pyenv/versions/3.8.12/envs/lewagon-data/lib/python3.8/site-packages/sklearn/metrics/\_scorer.py:765: FutureWarning: sklearn.metrics.SCORERS is deprecated and will be removed in v1.3. Please use sklearn.metrics.get\_scorer\_names to get a list of available scorers and sklearn.metrics.get\_metric to get scorer.**

## **warnings.warn(**

## 

## ***# Export TPOT Pipeline to a Python file***

## **tpot.export(os.path.join(os.getcwd(),'tpot\_iris\_pipeline.py'))**

## 

## **! cat 'tpot\_iris\_pipeline.py'**

## 

## **import numpy as np**

## **import pandas as pd**

## **from sklearn.ensemble import GradientBoostingRegressor**

## **from sklearn.linear\_model import RidgeCV**

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

## **from sklearn.pipeline import make\_pipeline, make\_union**

## **from tpot.builtins import StackingEstimator**

## 

## **# NOTE: Make sure that the outcome column is labeled 'target' in the data file**

## **tpot\_data = pd.read\_csv('PATH/TO/DATA/FILE', sep='COLUMN\_SEPARATOR', dtype=np.float64)**

## **features = tpot\_data.drop('target', axis=1)**

## **training\_features, testing\_features, training\_target, testing\_target = \**

## **train\_test\_split(features, tpot\_data['target'], random\_state=None)**

## 

## **# Average CV score on the training set was: 0.8558530771102855**

## **exported\_pipeline = make\_pipeline(**

## **StackingEstimator(estimator=GradientBoostingRegressor(alpha=0.9, learning\_rate=0.01, loss="ls", max\_depth=3, max\_features=0.6000000000000001, min\_samples\_leaf=14, min\_samples\_split=13, n\_estimators=100, subsample=0.55)),**

## **RidgeCV()**

## **)**

## 

## **exported\_pipeline.fit(training\_features, training\_target)**

## **results = exported\_pipeline.predict(testing\_features)**

## 

## **Summary**

## **Pipeline**

## **→**

## **list of sequential steps**

## **ColumnTransformer**

## **→**

## **list of parallel steps**

## **remainder="passthrough": used to save untransformed columns**

## **FunctionTransformer**

## **→**

## **encapsulates a function as a Scikit-Learn transformer that you can plug into a Pipeline or a ColumnTransformer**

## **FeatureUnion**

## **→**

## **applies transformations in parallel and concatenates the results, quite useful for feature creation**

## **preproc\_full**

## 

## **FeatureUnion**

## **columntransformer**

## **pipeline**

## **SimpleImputer**

## **StandardScaler**

## **onehotencoder**

## **OneHotEncoder**

## **remainder**

## **passthrough**

## **functiontransformer**

## **FunctionTransformer**

## **You can chain a preprocessor pipeline with a Scikit Learn model**

## **A full pipeline can go through cross\_validate, GridSearchCV, RandomizedSearchCV**

## **pipeline\_tuned**

## 

## **Pipeline**

## **columntransformer: ColumnTransformer**

## **pipeline**

## **SimpleImputer**

## **StandardScaler**

## **onehotencoder**

## **OneHotEncoder**

## **remainder**

## **passthrough**

## **Ridge**

# **Your turn! 🚀**

## [**¶**](https://kitt.lewagon.com/camps/1917/lectures/content/05-ML_08-Workflow.html#Plan)

# **Model Selection Tips** 💡

# **Pipelines** 🔥

# Preprocessing Pipes

# Pipelines → → →

# Column Transformers ⑂

# Custom Transformers →

# Feature Unions ||

# Full Pipes (Preprocessing + Models)

# **Surprise** 🤗

## **1. Model Selection**

# **Let's take a step back: which models have we seen so far?**

# 1️⃣ Regression models are parametric

# ^

# y

# =

# f

# β

# (

# X

# )

# 

# An arbitrarily large number

# n

# of datapoints can be modeled with few

# β

# parameters

# *Note: Neural Networks are also parametrics models (See Deep Learning)*

# ✅ Fast to train, even on large datasets with Stochastic Gradient Descent

# ❗️ Requires prior assumptions

# f

# about the structure of the data; may not find complex patterns, unless given complex features

# 2️⃣ KNN, kernel-SVM are non-parametric

# No prior assumptions about the data structure are needed

# Possibly many parameters to learn (not known beforehand)

# e.g. KNN .fit stores the *whole dataset*

# e.g. rbf-SVM .fit must compute a Kernel between *each pair* of datapoints

# *Note: Trees are also non-parametric models (See Ensemble Methods)*

# ✅ Can find complex features for you!

# ❗️ Harder to train on large datasets and prone to overfitting

# 

# **2. Pipelines**

# 📚 [**sklearn - Pipeline and composite estimators**](https://scikit-learn.org/stable/modules/compose.html)

# 📚 [**sklearn.pipeline**](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.pipeline)

# 

# A ***Pipeline*** is a chain of operations in a Machine Learning project (preprocessing, training, predicting, etc.)

# Pipelines are powerful because they:

# 🏄🏻‍♀️ make your workflow much easier to read and understand

# 💪 enforce the implementation and order of steps in your project

# ⚙️ make your work reproducible and deployable

# 

## **2.1 Preprocessing Pipelines**

# 🎯 We are going to predict the **charges** of a health insurance contract based on various features using the following dataset.

# 💾 Download the dataset [here](https://wagon-public-datasets.s3.amazonaws.com/data_workflow.csv)

# data.head(5)

# 

|  | **age** | **bmi** | **children** | **smoker** | **region** | **charges** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 19.0 | 27.900 | 0 | True | southwest | 16884.92400 |
| **1** | 18.0 | 33.770 | 1 | False | southeast | 1725.55230 |
| **2** | NaN | 33.000 | 3 | False | southeast | 4449.46200 |
| **3** | 33.0 | 22.705 | 0 | False | northwest | 21984.47061 |
| **4** | 32.0 | 28.880 | 0 | False | northwest | 3866.85520 |

# data.shape

# 

# (1338, 6)

# *# Defining the features and the target*

# 

# X = data.drop(columns='charges')

# y = data['charges']

# 

# *# Train-Test split*

# 

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

# 

# X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20)

# X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape

# 

# ((1070, 5), (268, 5), (1070,), (268,))

# ✏️ **Today's challenges**:

# *Impute* missing values

# Preprocessing:

# *Scale* numerical features

# *Encode* categorical features

# *Fine-tune* your ML model ***and*** the preprocessing steps...

# ... 🔥 in one cell ! 🔥

### **a) Pipeline → → →**

# A Pipeline essentially **chains** multiple steps **in sequence** (e.g. *imputing* then *scaling*)

# 📚 [**sklearn.pipeline.Pipeline**](https://scikit-learn.org/0.16/modules/generated/sklearn.pipeline.Pipeline.html)

# **from** **sklearn.pipeline** **import** Pipeline

# 

# *# Preprocess "age"*

# **from** **sklearn.pipeline** **import** Pipeline

# **from** **sklearn.impute** **import** SimpleImputer

# **from** **sklearn.preprocessing** **import** StandardScaler

# 

# *# Build the pipeline with the different steps*

# pipeline = Pipeline([

# ('imputer', SimpleImputer(strategy="median")),

# ('standard\_scaler', StandardScaler())

# ])

# 

# pipeline.fit(X\_train[['age']])

# pipeline.transform(X\_train[['age']])

# 

# array([[ 1.03287039],

# [-1.45497346],

# [ 1.1750329 ],

# ...,

# [ 0.25097661],

# [-0.17551091],

# [-1.2417297 ]])

# *# Show the different steps of the pipeline*

# pipeline

# 

# **Pipeline**

# SimpleImputer

# StandardScaler

### **b) Column Transformer ⑂**

# Column Transformers allow you to apply specific changes to specific columns **in parallel**

# 📚 [**sklearn.compose.ColumnTransformer**](https://scikit-learn.org/stable/modules/generated/sklearn.compose.ColumnTransformer.html)

# **from** **sklearn.compose** **import** ColumnTransformer

# 

# 💻 Let's perform the following operations **in parallel**:

# 🔢 *Impute* then *scale* numerical values

# 🔠 *Encode* categorical values

# 

# ☝️ Notice how a Pipeline object can be passed into a ColumnTransformer!

# **from** **sklearn.compose** **import** ColumnTransformer

# 

# **from** **sklearn.pipeline** **import** Pipeline

# **from** **sklearn.impute** **import** SimpleImputer

# **from** **sklearn.preprocessing** **import** StandardScaler

# 

# **from** **sklearn.preprocessing** **import** OneHotEncoder

# 

# 

# *# Impute then scale numerical values:*

# num\_transformer = Pipeline([

# ('imputer', SimpleImputer(strategy="mean")),

# ('standard\_scaler', StandardScaler())

# ])

# 

# *# Encode categorical values*

# cat\_transformer = OneHotEncoder(handle\_unknown='ignore')

# 

# *# Parallelize "num\_transformer" and "cat\_transfomer"*

# preprocessor = ColumnTransformer([

# ('num\_transformer', num\_transformer, ['age', 'bmi']),

# ('cat\_transformer', cat\_transformer, ['smoker', 'region'])

# ])

# 

# *# Visualizing Pipelines in HTML*

# **from** **sklearn** **import** set\_config; set\_config(display='diagram')

# preprocessor

# 

# **ColumnTransformer**

# **num\_transformer**

# SimpleImputer

# StandardScaler

# **cat\_transformer**

# OneHotEncoder

# X\_train\_transformed = preprocessor.fit\_transform(X\_train)

# 

# print("Original training set")

# display(X\_train.head(3))

# 

# print("Preprocessed training set")

# display(pd.DataFrame(X\_train\_transformed).head(3))

# 

# Original training set

# 

|  | **age** | **bmi** | **children** | **smoker** | **region** |
| --- | --- | --- | --- | --- | --- |
| **162** | 54.0 | 39.60 | 1 | False | southwest |
| **410** | 19.0 | 17.48 | 0 | False | northwest |
| **639** | 56.0 | 33.66 | 4 | False | southeast |

# Preprocessed training set

# 

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.032979 | 1.456688 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| **1** | -1.454870 | -2.170790 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **2** | 1.175141 | 0.482582 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |

# 😱 Where are the columns' names?

# 😉 Don't worry and stay tuned to scikit-learn updates!

# [scikit-learn.org/stable/whats\_new.html](https://scikit-learn.org/stable/whats_new.html)

### **🚀 get\_feature\_names\_out() 🚀**

# New in scikit-learn 1.0.2 *(September 2021)*

# ✅ This new method helps retrieve the names of the features which went through some transformations like StandardScaler or OheHotEncoder

# ❌ Not all the transformers in Scikit-Learn have this new method

# New in scikit-learn 1.1.3: *(October 2022)*

# ✅ *ALL* the transformers have this method!

# *# Get your features' names*

# preprocessor.get\_feature\_names\_out()

# 

# array(['num\_transformer\_\_age', 'num\_transformer\_\_bmi',

# 'cat\_transformer\_\_smoker\_False', 'cat\_transformer\_\_smoker\_True',

# 'cat\_transformer\_\_region\_northeast',

# 'cat\_transformer\_\_region\_northwest',

# 'cat\_transformer\_\_region\_southeast',

# 'cat\_transformer\_\_region\_southwest'], dtype=object)

# pd.DataFrame(

# X\_train\_transformed,

# columns=preprocessor.get\_feature\_names\_out()

# ).head()

# 

|  | **num\_transformer\_\_age** | **num\_transformer\_\_bmi** | **cat\_transformer\_\_smoker\_False** | **cat\_transformer\_\_smoker\_True** | **cat\_transformer\_\_region\_northeast** | **cat\_transformer\_\_region\_northwest** | **cat\_transformer\_\_region\_southeast** | **cat\_transformer\_\_region\_southwest** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.032979 | 1.456688 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |
| **1** | -1.454870 | -2.170790 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **2** | 1.175141 | 0.482582 | 1.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| **3** | -0.815138 | 0.157880 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| **4** | -0.601894 | -0.148783 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 |

# 🤔 What happened to the children column? What if we want to keep it untouched?

#### **👉 remainder=passthrough**

# preprocessor = ColumnTransformer([

# ('num\_transformer', num\_transformer, ['age','bmi']),

# ('cat\_transformer', cat\_transformer, ['region','smoker'])],

# remainder='passthrough'

# )

# 

# preprocessor

# 

# **ColumnTransformer**

# **num\_transformer**

# SimpleImputer

# StandardScaler

# **cat\_transformer**

# OneHotEncoder

# **remainder**

# passthrough

# pd.DataFrame(preprocessor.fit\_transform(X\_train),

# columns=preprocessor.get\_feature\_names\_out()).head(3)

# 

|  | **num\_transformer\_\_age** | **num\_transformer\_\_bmi** | **cat\_transformer\_\_region\_northeast** | **cat\_transformer\_\_region\_northwest** | **cat\_transformer\_\_region\_southeast** | **cat\_transformer\_\_region\_southwest** | **cat\_transformer\_\_smoker\_False** | **cat\_transformer\_\_smoker\_True** | **remainder\_\_children** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.032979 | 1.456688 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 | 0.0 | 1.0 |
| **1** | -1.454870 | -2.170790 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **2** | 1.175141 | 0.482582 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 4.0 |

### **c) Custom: Function Transformer →**

# 📚 [**sklearn.preprocessing.FunctionTransformer**](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.FunctionTransformer.html)

# **from** **sklearn.preprocessing** **import** FunctionTransformer

# 

# ⚒️ Function Transformers enable you to encapsulate a *Python* function within a scikitTransformer (→) Object

# 💪 They can be used with either Pipelines (**→ → →**) or ColumnTransformers (**⑂**)

# 

# 👆 If you want to use your own transformer in a Pipeline or a ColumnTransformer *(not one already available in Sklearn)*, you must encapsulate your function within a **FunctionTransformer**.

# **from** **sklearn.preprocessing** **import** FunctionTransformer

# 

# *# Create a transformer that compresses data to 2 digits (for instance!)*

# *# rounder = FunctionTransformer(np.round)*

# 

# *# We can use a lambda function for more customizable functions*

# rounder = FunctionTransformer(**lambda** array: np.round(array, decimals=2))

# 

# *# Add it at the end of our numerical transformer*

# num\_transformer = Pipeline([

# ('imputer', SimpleImputer()),

# ('scaler', StandardScaler()),

# ('rounder', rounder)])

# 

# *# Encode categorical values*

# cat\_transformer = OneHotEncoder(drop='if\_binary',

# handle\_unknown='ignore')

# 

# preprocessor = ColumnTransformer([

# ('num\_transformer', num\_transformer, ['bmi', 'age']),

# ('cat\_transformer', cat\_transformer, ['region', 'smoker'])],

# remainder='passthrough')

# preprocessor

# 

# **ColumnTransformer**

# **num\_transformer**

# SimpleImputer

# StandardScaler

# FunctionTransformer

# **cat\_transformer**

# OneHotEncoder

# **remainder**

# passthrough

# pd.DataFrame(preprocessor.fit\_transform(X\_train)).head(3)

# 

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.46 | 1.03 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 |
| **1** | -2.17 | -1.45 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **2** | 0.48 | 1.18 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 4.0 |

# ❗️ **FunctionTransformer** only works for **stateless** transformations ❗️

# 👨🏻‍🏫 **stateless transformations** are transformations which cannot *store* information during .fit(X\_train) that would be used for the .transform(X\_test).

# 👍 Since a stateless transformation doesn't learn anything, fitting it is impossible, it does nothing other than transform!

# ✅ FunctionTransformer is compatible with **stateless** transformations.

# *Examples* of transformations which don't "learn" anything:

# X

# →

# l

# o

# g

# (

# X

# )

# 

# (

# X

# 1

# ,

# X

# 2

# )

# →

# X

# 1

# +

# 5

# X

# 2

# 

# 👩‍🏫 **stateful transformations** are transformations which *store* information during .fit(X\_train). This information is re-used for .transform(X\_test).

# *Examples* of transformations which "learn" something:

# X

# t

# r

# a

# i

# n

# →

# S

# t

# a

# n

# d

# a

# r

# d

# S

# c

# a

# l

# e

# r

# (

# X

# t

# r

# a

# i

# n

# )

# **learns**

# μ

# t

# r

# a

# i

# n

# and

# σ

# t

# r

# a

# i

# n

# X

# t

# r

# a

# i

# n

# →

# M

# i

# n

# M

# a

# x

# S

# c

# a

# l

# e

# r

# (

# X

# t

# r

# a

# i

# n

# )

# **learns**

# X

# (

# m

# i

# n

# )

# t

# r

# a

# i

# n

# and

# X

# (

# m

# a

# x

# )

# t

# r

# a

# i

# n

# ❌ FunctionTransformer is not compatible with **stateful** transformations

# 🌶️ We will have to code our own Class to use FunctionTransformer with stateful transformations!

#### **🕵🏻‍♂️ Transformers under the hood**

# **from** **sklearn.base** **import** TransformerMixin, BaseEstimator

# 

# **class** **MyCustomTranformer**(TransformerMixin, BaseEstimator):

# *# BaseEstimator generates the get\_params() and set\_params() methods that all Pipelines require*

# *# TransformerMixin creates the fit\_transform() method from fit() and transform()*

# 

# **def** \_\_init\_\_(self):

# **pass**

# 

# **def** fit(self, X, y=**None**):

# *# Here you store what needs to be stored/learned during .fit(X\_train) as instance attributes*

# *# Return "self" to allow chaining .fit().transform()*

# **pass**

# 

# **def** transform(self, X, y=**None**):

# *# Return the result as a DataFrame for an integration into the ColumnTransformer*

# **pass**

# 

# my\_transformer = MyCustomTranformer()

# my\_transformer.fit(X\_train)

# my\_transformer.transform(X\_train)

# my\_transformer.transform(X\_test)

# 

# 💻 *More in today's challenges* 💻

### **d) FeatureUnion | |**

# FeatureUnion applies a list of transformer objects **in parallel** to the input data, then **concatenates** the results. This is useful to combine several feature extraction mechanisms into a single transformer

# 📚 [**sklearn.pipeline.FeatureUnion**](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.FeatureUnion.html)

# 👌 Useful to **create entirely new features**!

# *Example*: let's build and add a new feature called bmi\_age\_ratio

# X\_train.head(3)

# 

|  | **age** | **bmi** | **children** | **smoker** | **region** |
| --- | --- | --- | --- | --- | --- |
| **162** | 54.0 | 39.60 | 1 | False | southwest |
| **410** | 19.0 | 17.48 | 0 | False | northwest |
| **639** | 56.0 | 33.66 | 4 | False | southeast |

# **from** **sklearn.pipeline** **import** FeatureUnion

# 

# *# Create a custom transformer that multiplies/divides two columns*

# *# Notice that we are creating this new feature completely randomly just as an example*

# bmi\_age\_ratio\_constructor = FunctionTransformer(**lambda** df: pd.DataFrame(df["bmi"] / df["age"]))

# 

# union = FeatureUnion([

# ('preprocess', preprocessor), *# columns 0-7*

# ('bmi\_age\_ratio', bmi\_age\_ratio\_constructor) *# new column 8*

# ])

# 

# union

# 

# **FeatureUnion**

# **preprocess**

# **num\_transformer**

# SimpleImputer

# StandardScaler

# FunctionTransformer

# **cat\_transformer**

# OneHotEncoder

# **remainder**

# passthrough

# **bmi\_age\_ratio**

# FunctionTransformer

# pd.DataFrame(union.fit\_transform(X\_train)).head(1)

# 

|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.46 | 1.03 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.733333 |

### **Building your preprocessor with make\_\*\*\* shortcuts ⚡️**

# **from** **sklearn.pipeline** **import** Pipeline

# **from** **sklearn.pipeline** **import** FeatureUnion

# **from** **sklearn.compose** **import** ColumnTransformer

# 

# There are equivalent transformers using the syntax **make\_\*\*\***👇

# **from** **sklearn.pipeline** **import** make\_pipeline

# **from** **sklearn.pipeline** **import** make\_union

# **from** **sklearn.compose** **import** make\_column\_transformer

# 

# Pipeline([

# ('my\_name\_for\_the\_imputer', SimpleImputer()),

# ('my\_name\_for\_the\_scaler', StandardScaler())

# ])

# 

# ⇔

# make\_pipeline(SimpleImputer(), StandardScaler())

# 

# **Pipeline**

# SimpleImputer

# StandardScaler

# num\_transformer = make\_pipeline(SimpleImputer(), StandardScaler())

# cat\_transformer = OneHotEncoder()

# 

# preproc\_basic = make\_column\_transformer(

# (num\_transformer, ['age', 'bmi']),

# (cat\_transformer, ['smoker', 'region']),

# remainder='passthrough'

# )

# 

# preproc\_full = make\_union(preproc\_basic, bmi\_age\_ratio\_constructor)

# 

# preproc\_full

# 

# **FeatureUnion**

# **columntransformer**

# **pipeline**

# SimpleImputer

# StandardScaler

# **onehotencoder**

# OneHotEncoder

# **remainder**

# passthrough

# **functiontransformer**

# FunctionTransformer

#### **🍒 make\_column\_selector selects features automatically based on dtype**

# **from** **sklearn.compose** **import** make\_column\_selector

# 

# num\_col = make\_column\_selector(dtype\_include=['float64'])

# cat\_col = make\_column\_selector(dtype\_include=['object','bool'])

# 

# X\_train.dtypes

# 

# age float64

# bmi float64

# children int64

# smoker bool

# region object

# dtype: object

# 🎉 **Complete preprocessing pipeline** 🎉

# **from** **sklearn.compose** **import** make\_column\_selector

# 

# num\_transformer = make\_pipeline(SimpleImputer(), StandardScaler())

# num\_col = make\_column\_selector(dtype\_include=['float64'])

# 

# cat\_transformer = OneHotEncoder()

# cat\_col = make\_column\_selector(dtype\_include=['object','bool'])

# 

# preproc\_basic = make\_column\_transformer(

# (num\_transformer, num\_col),

# (cat\_transformer, cat\_col),

# remainder='passthrough'

# )

# 

# preproc\_full = make\_union(preproc\_basic, bmi\_age\_ratio\_constructor)

# 

# preproc\_full

# 

# **FeatureUnion**

# **columntransformer**

# **pipeline**

# SimpleImputer

# StandardScaler

# **onehotencoder**

# OneHotEncoder

# **remainder**

# passthrough

# **functiontransformer**

# FunctionTransformer

## **2.2 Including models in Pipelines**

# Model objects can be plugged into Pipelines

# Pipelines inherit the methods of the **last** object in the sequence

# Transformers: fit and transform

# Models: fit, score, predict, etc.

# 

# When executing the pipeline.fit method, the transformer's .fit and .transform methods will be called sequentially, and the model will be trained.

# At this stage, all transformers' variables are saved into the memory of the pipeline

# When executing the pipeline.predict method, only the transformer's .transform method will be called, using the variables learned during the original fit

### **a) Full pipeline**

# **from** **sklearn.linear\_model** **import** Ridge

# 

# *# Preprocessor*

# num\_transformer = make\_pipeline(SimpleImputer(), StandardScaler())

# cat\_transformer = OneHotEncoder()

# 

# preproc = make\_column\_transformer(

# (num\_transformer, make\_column\_selector(dtype\_include=['float64'])),

# (cat\_transformer, make\_column\_selector(dtype\_include=['object','bool'])),

# remainder='passthrough'

# )

# 

# *# Add estimator*

# pipeline = make\_pipeline(preproc, Ridge())

# pipeline

# 

# **Pipeline**

# **columntransformer: ColumnTransformer**

# **pipeline**

# SimpleImputer

# StandardScaler

# **onehotencoder**

# OneHotEncoder

# **remainder**

# passthrough

# Ridge

# *# Train Pipeline*

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

# 

# *# Make predictions*

# pipeline.predict(X\_test.iloc[0:1])

# 

# *# Score model*

# pipeline.score(X\_test,y\_test)

# 

# 0.7473478157212925

### **b) Cross-validate a Pipeline**

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

# 

# *# Cross-validate Pipeline*

# cross\_val\_score(pipeline, X\_train, y\_train, cv=5, scoring='r2').mean()

# 

# 0.7434317676218065

### **c) Grid Search a Pipeline**

# *Grid Searching* allows you to check which combination of preprocessing/modeling **hyperparameters** works best.

# It is possible to *Grid Search* the hyperparameters of **any component of the Pipeline**

# Typical Sklearn syntax: step\_name\_\_transformer\_name\_\_hyperparameter\_name

# To check which hyperparameters of the pipeline can be optimized: pipeline.get\_params()

# *# Which parameters of the pipeline are GridSearch-able?*

# pipeline.get\_params()

# 

# **from** **sklearn.model\_selection** **import** GridSearchCV

# 

# grid\_search = GridSearchCV(

# pipeline,

# param\_grid={

# *# Access any component of the Pipeline*

# *# and any available hyperparamater you want to optimize*

# 'columntransformer\_\_pipeline\_\_simpleimputer\_\_strategy': ['mean', 'median'],

# 'ridge\_\_alpha': [0.1, 0.5, 1, 5, 10]

# },

# cv=5,

# scoring="r2")

# 

# grid\_search.fit(X\_train, y\_train)

# 

# grid\_search.best\_params\_

# 

# {'columntransformer\_\_pipeline\_\_simpleimputer\_\_strategy': 'mean',

# 'ridge\_\_alpha': 1}

# 💾 Let's save the pipelined model with the best hyperparameters.

# pipeline\_tuned = grid\_search.best\_estimator\_

# pipeline\_tuned

# 

# **Pipeline**

# **columntransformer: ColumnTransformer**

# **pipeline**

# SimpleImputer

# StandardScaler

# **onehotencoder**

# OneHotEncoder

# **remainder**

# passthrough

# Ridge

# 🔮 We can use this "best" model for predictions without re-training it!

# pipeline\_tuned.predict(X\_test[0:1])

# 

# array([10216.56989159])

### **d) Caching to avoid repeated computations**

# 😠 Are your preprocessing steps too long to run?

# 🪄 You can use caching techniques!

# **from** **tempfile** **import** mkdtemp

# **from** **shutil** **import** rmtree

# 

# *# Create a temp folder*

# cachedir = mkdtemp()

# 

# *# Instantiate the Pipeline with the cache parameter*

# pipeline = Pipeline(steps, memory=cachedir)

# 

# *# Clear the cache directory after the cross-validation*

# rmtree(cachedir)

# 

# With the parameter memory=cachedir, preproc parameters can be cached into memory.

# Avoid recalculating all of the parameters during CrossValidation or GridSearchCV on estimator hyperparams only

# Helpful only when the transformer's .fit time is long and the dataset is very large

### **e) Debug your pipe**

# *# Access the components of a Pipeline with `named\_steps`*

# pipeline\_tuned.named\_steps.keys()

# 

# dict\_keys(['columntransformer', 'ridge'])

# *# Check intermediate steps*

# print("Before preprocessing, X\_train.shape = ")

# print(X\_train.shape)

# print("After preprocessing, X\_train\_preprocessed.shape = ")

# pipeline\_tuned.named\_steps["columntransformer"].fit\_transform(X\_train).shape

# 

# Before preprocessing, X\_train.shape =

# (1070, 5)

# After preprocessing, X\_train\_preprocessed.shape =

# 

# (1070, 9)

### **f) Exporting models/Pipelines**

# 💾 You can export your final model/pipeline as a pickle file

# 👉The file can then be loaded back into a notebook or deployed on a server (see ML Opsmodule).

# **import** **pickle**

# 

# *# Export Pipeline as pickle file*

# **with** open("pipeline.pkl", "wb") **as** file:

# pickle.dump(pipeline\_tuned, file)

# 

# *# Load Pipeline from pickle file*

# my\_pipeline = pickle.load(open("pipeline.pkl","rb"))

# 

# my\_pipeline.score(X\_test, y\_test)

# 

# 0.7473478157212925

# **3. Surprise 🎉**

# **AutoML**

## **TPOT**

# The Tree-based Pipeline Optimization Tool (TPOT) is an automated Machine Learning tool that optimizes Machine Learning Pipelines

# 

# 📚 More details available in the [**TPOT documentation**](http://epistasislab.github.io/tpot/)

# **Installation**

# 

# pip install TPOT

# 

# **import** **os**

# **from** **tpot** **import** TPOTRegressor

# 

# X\_train\_preproc = preproc\_basic.fit\_transform(X\_train)

# X\_test\_preproc = preproc\_basic.transform(X\_test)

# 

# *# Instantiate TPOTClassifier*

# tpot = TPOTRegressor(generations=4, population\_size=20, verbosity=2, scoring='r2', n\_jobs=-1, cv=2)

# 

# *# Process autoML with TPOT*

# tpot.fit(X\_train\_preproc, y\_train)

# 

# *# Print score*

# print(tpot.score(X\_test\_preproc, y\_test))

# 

# Generation 1 - Current best internal CV score: 0.8517440046999218

# 

# Generation 2 - Current best internal CV score: 0.853008927910814

# 

# Generation 3 - Current best internal CV score: 0.853008927910814

# 

# Generation 4 - Current best internal CV score: 0.8558530771102855

# 

# Best pipeline: RidgeCV(GradientBoostingRegressor(input\_matrix, alpha=0.9, learning\_rate=0.01, loss=ls, max\_depth=3, max\_features=0.6000000000000001, min\_samples\_leaf=14, min\_samples\_split=13, n\_estimators=100, subsample=0.55))

# 0.872811877434264

# 

# /Users/davywai/.pyenv/versions/3.8.12/envs/lewagon-data/lib/python3.8/site-packages/sklearn/metrics/\_scorer.py:765: FutureWarning: sklearn.metrics.SCORERS is deprecated and will be removed in v1.3. Please use sklearn.metrics.get\_scorer\_names to get a list of available scorers and sklearn.metrics.get\_metric to get scorer.

# warnings.warn(

# 

# *# Export TPOT Pipeline to a Python file*

# tpot.export(os.path.join(os.getcwd(),'tpot\_iris\_pipeline.py'))

# 

# ! cat 'tpot\_iris\_pipeline.py'

# 

# import numpy as np

# import pandas as pd

# from sklearn.ensemble import GradientBoostingRegressor

# from sklearn.linear\_model import RidgeCV

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

# from sklearn.pipeline import make\_pipeline, make\_union

# from tpot.builtins import StackingEstimator

# 

# # NOTE: Make sure that the outcome column is labeled 'target' in the data file

# tpot\_data = pd.read\_csv('PATH/TO/DATA/FILE', sep='COLUMN\_SEPARATOR', dtype=np.float64)

# features = tpot\_data.drop('target', axis=1)

# training\_features, testing\_features, training\_target, testing\_target = \

# train\_test\_split(features, tpot\_data['target'], random\_state=None)

# 

# # Average CV score on the training set was: 0.8558530771102855

# exported\_pipeline = make\_pipeline(

# StackingEstimator(estimator=GradientBoostingRegressor(alpha=0.9, learning\_rate=0.01, loss="ls", max\_depth=3, max\_features=0.6000000000000001, min\_samples\_leaf=14, min\_samples\_split=13, n\_estimators=100, subsample=0.55)),

# RidgeCV()

# )

# 

# exported\_pipeline.fit(training\_features, training\_target)

# results = exported\_pipeline.predict(testing\_features)

# 

## **Summary**

# Pipeline

# →

# list of sequential steps

# ColumnTransformer

# →

# list of parallel steps

# remainder="passthrough": used to save untransformed columns

# FunctionTransformer

# →

# encapsulates a function as a Scikit-Learn transformer that you can plug into a Pipeline or a ColumnTransformer

# FeatureUnion

# →

# applies transformations in parallel and concatenates the results, quite useful for feature creation

# preproc\_full

# 

# **FeatureUnion**

# **columntransformer**

# **pipeline**

# SimpleImputer

# StandardScaler

# **onehotencoder**

# OneHotEncoder

# **remainder**

# passthrough

# **functiontransformer**

# FunctionTransformer

# You can chain a preprocessor pipeline with a Scikit Learn model

# A full pipeline can go through cross\_validate, GridSearchCV, RandomizedSearchCV

# pipeline\_tuned

# 

# **Pipeline**

# **columntransformer: ColumnTransformer**

# **pipeline**

# SimpleImputer

# StandardScaler

# **onehotencoder**

# OneHotEncoder

# **remainder**

# passthrough

# Ridge

# **Your turn! 🚀**

# 