

towards
data science

Sign in

Get started



Follow

612K Followers

·

Editors' Picks

Features

Deep Dives

You have **2** free member-only stories left this month. [Sign up for Medium and get an extra one](#)

Pre-Process Data like a Pro: Intro to Scikit-Learn Pipelines

Reusable Functions to Impute, Scale, Encode, and Transform Your Data



Guillermo Perez Sep 1, 2020 · 6 min read ★



Data Wrangling in the 20's Photo by [Mahir Uysal](#) on [Unsplash](#)

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.



“Without a systematic way to start and keep data clean, bad data

will happen.”

— *Donato Diorio*
Guillermo Perez

Data scientist sharing thoughts

You wrote all your queries, gathered all the data and are all fired up to implement the latest machine learning algorithm

you read about on Medium. Wait! You soon realize you need to

Follow



dealing with missing data, imputation, categorical data, standardization, etc.

GUILLERMO PEREZ FOLLOWS



Marco Cerliani



Paulo Haddad



Zhiluyev Tran






Jon Blosio



Yong Cui

See all (19)

In this article you will learn how to :

- Reproduce transformations easily on any dataset.
 112  
- Easily track all transformations you apply to your dataset.
- Start building your library of transformations you can use later on different projects.

tl;dr: Let's build a pipeline where we can impute, transform, scale, and encode like this:

```
from sklearn.compose import ColumnTransformer
```

```
data_pipeline = ColumnTransformer([
    ('numerical', num_pipeline, num_vars),
    ('categorical', OneHotEncoder(), cat_vars),
])

airbnb_processed =
data_pipeline.fit_transform(airbnb_data)
```

Without knowing much you can infer that different transformations are applied to numerical variables and categorical variables. Let's go into the proverbial weeds and see how we end up with this pipeline.

. . .

Data

For all of these examples, I will be using the airbnb NYC listings dataset from insideairbnb.com. This is a real dataset containing information scraped from airbnb and has all the information related to a listing on the site.

Let us imagine we want to predict the price of a listing given some variables like the property type and neighborhood.

```
raw_data =
pd.read_csv('http://data.insideairbnb.com/united-
states/ny/new-york-city/2020-07-07/data/listings.csv.gz',
            compression='gzip')
```

Let's start by getting our categorical and numerical variables that we want to work with. We will keep it simple by removing data with missing values in our categorical variables of interest and with no reviews.

```
1  #Categorical variables to use
2  cat_vars = ["instant_bookable", "is_business_travel_ready", "cancellati
3
4  # Numerical Variables to use
5  num_vars = ['price', 'square_feet', 'minimum_nights', 'weekly_price', 'mo
6              'cleaning_fee', "reviews_per_month", "host_listings_count",
7              "calculated_host_listings_count_shared_rooms", "host_respons
8              "review_scores_cleanliness", "review_scores_value", "review_sc
9
10  airbnb_data = raw_data
11
12  # Drop observations if categorical data is missing
13  before = airbnb_data.shape[0]
14  airbnb_data = airbnb_data.dropna(subset=cat_vars + ["number_of_reviews"]
15  airbnb_data = airbnb_data[airbnb_data['number_of_reviews']>0]
16  after = airbnb_data.shape[0]
17
18
```

Imputation

Is a dataset even real if it isn't missing data? The reality is that we have to deal with missing data all the time. You will have to decide how to deal with missing data for your specific use

- You can `dropna()` rows with missing data. Might drop too much data.
- Drop the variable that has missing data. What if you really

want that variable?

- Replace NAs with zero, the mean, median, or some other calculation.

Scikit-Learn provides us with a nice simple class to deal with missing values.

Let us impute numerical variables such as price or security deposit with the median. For simplicity, we do this for all numerical variables.

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")

# Num_vars is the list of numerical variables
airbnb_num = airbnb_data[num_vars]

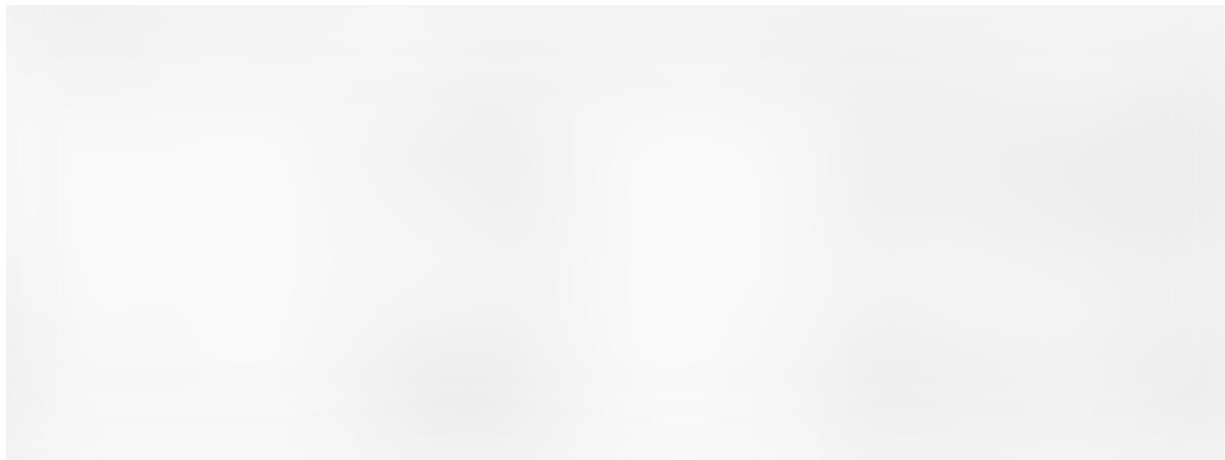
airbnb_num = imputer.fit_transform(airbnb_num)
```

The `SimpleImputer` class will replace all the missing values with the median. The `.fit_transform()` method will return a nice numpy array object that is ready to be used with any machine learning method. You can choose different metrics and pass it as an argument.

Encoding Categorical Variables

Numerical variables are pretty straightforward. Let us deal with categorical data that usually comes in strings. In this case, we have to deal with variables such as neighborhood, room_type, bed_type. Machine Learning algorithms work better with numbers, so we will convert our categorical variables.

How our categorical data looks:



We will convert all the text variables.

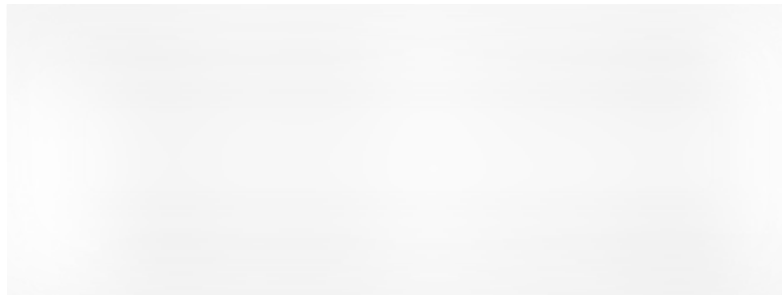
We use the `OrdinalEncoder` to convert our string data to numbers. Each unique value in the variables will be mapped to a number. E.g Apartment =0, Condominium=1, etc.

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()

airbnb_cat_encoded =
ordinal_encoder.fit_transform(airbnb_cat)
airbnb_cat_encoded[:,1:10]
```

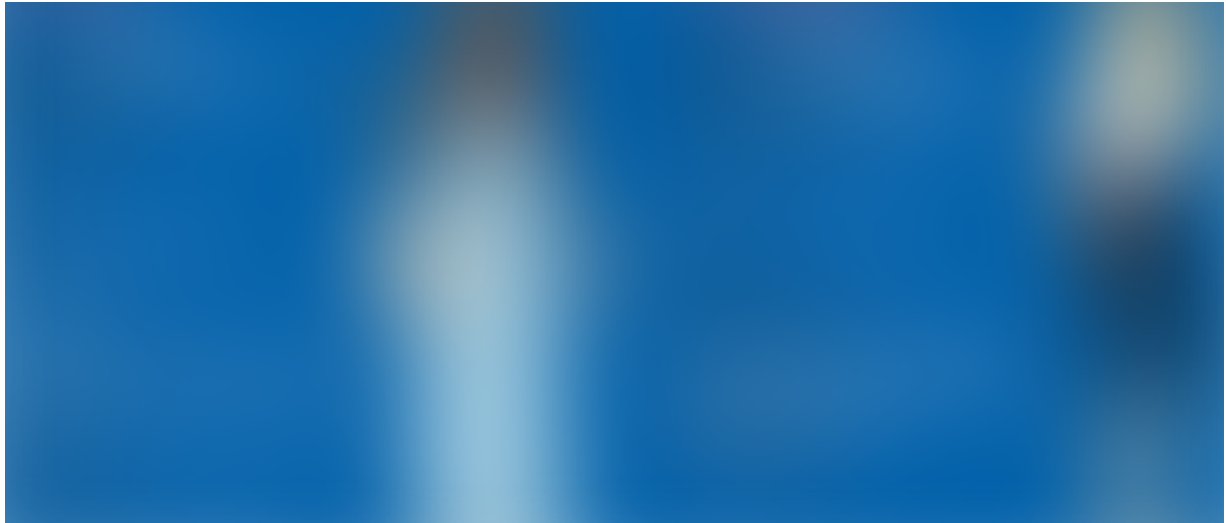
After encoding we get :



All our text data is now encoded.

Are we done? Not yet. ML algos can take things a little too literally and think that categories 1 and 2 are more similar than 1 and 19. This might be good for ratings or things where there is an order but how about neighborhoods? Can this simple encoder differentiate the vibes of SoHo and TriBeCa? Does it know all the normies work in Midtown?





Type of Manhattan Roommates via [6sqft](#)

We can transform our data to have a singular attribute per category. For Example, we create an attribute equal to 1 when the 'property_type' is 'House' and 0 otherwise. Repeat for all other categories.

This process is called one-hot encoding. If you come from a stats or econometrics background these are called dummy variables/attributes.

```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()

airbnb_cat_hot_encoded =
cat_encoder.fit_transform(airbnb_cat)
airbnb_cat_hot_encoded

<48563x281 sparse matrix of type '<class
'numpy.float64'>'
      with 388504 stored elements in Compressed Sparse
Row format>
```

A wild sparse matrix appears! Instead of a NumPy array the encoder returns a sparse matrix which is very convenient when we have several categorical attributes with hundreds of categories.

The matrix is full of 0s except for a single 1 per row. This would use a lot of memory to store but the sparse matrix is smart and stores only the location of the nonzero elements.

Feature Scaling

It is very important to scale our data when using machine learning algorithms. There are exceptions, but when the variables are in different scales we usually need to scale them. You can read all about it [here](#).

A couple of ways to do it:

- **Min-max scaling:** subtract the min value and divide by the range (max-min). Values will range from 0 to 1. The `MinMaxScaler` does this.
- **Standardization:** subtract mean and divide by standard deviation. You end up with 0 mean and unit variance. Values are not bounded in this case. Standardization is less affected by outliers. We can use `StandardScaler`.

```
from sklearn.preprocessing import StandardScaler
```

```
StandardScaler().fit_transform(airbnb_num)
```

That was easy!

Custom Transformations

Scikit-Learn API is very flexible lets you create your own custom “transformation” that you can easily incorporate into your process. You just need to implement the `fit()`, `transform()`, and `fit_transform()` methods.

Adding the `TransformerMixin` as a base class gets you the `fit_transform()` method automatically.

Here we have a very simple transformer that creates the ratio of rating to number of reviews. You can make these as complicated as you wish.

Checkout the Scikit-Learn documentation for more details and examples.

Pipelines

We can finally put everything together! One dataset can require several transformations which vary according to the variable type. These must also be performed in the right order.

Scikit-Learn gifts us the Pipeline class to help with this ubiquitous process to write clean and efficient code.

Putting it all together

We created a pipeline for our numerical variables. The Pipeline constructor takes in a list of ('Estimator Name', Estimator()) pairs. All estimators except the last one must have the `fit_transform()` method. They must be transformers. The names should be informative but you can put whatever you want.

Let us complete our pipeline with our categorical data and create our “master” Pipeline

We can combine different pipelines applied to different sets of variables. Here we are applying our numerical pipeline (Impute, Transform, Scale) to the numerical variables (num_vars is a list of column names) and do hot encoding to our categorical variables (cat_vars is a list of column names).

The `ColumnTransformer` class will do exactly what the name implies. It will apply the pipeline or transformer to a specified list of variables.

It takes in a list of tuples with ('name', transformer/pipeline, 'variables list')

Once we have set up and defined our pipeline process, we can

apply it to any dataset or fresh new data easily with the `fit_transform()` method.

• • •

Conclusion

Now you know the basics of implementing clean and efficient pipelines that can be easily organized and utilized on different datasets. For more detailed information reference the [Scikit-Learn documentation](#)

Why stop at pre-processing? In the pipeline you can finish with the training of your model. Here is a quick complete example:

Example from [Scikit-Learn documentation](#)

Once you have your data transformation setup, you can include the training as another “estimator” in your pipeline. Your pipeline can be used as any other estimator. `SVC()` is included at the end and will use the scaled data passed to it. You can then reuse this pipeline for many other machine learning algorithms or datasets.

Hopefully, this framework will help you write neat code that is easy to maintain. You can find this airbnb complete working example in a jupyter notebook in this [git repository](#).

Do clap and share if you found this worthy and helpful. Follow me or feel free to connect on [linkedin](#).

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)



Get this newsletter

Data Science

Data

Python

Data Preprocessing



[About](#) [Write](#) [Help](#) [Legal](#)