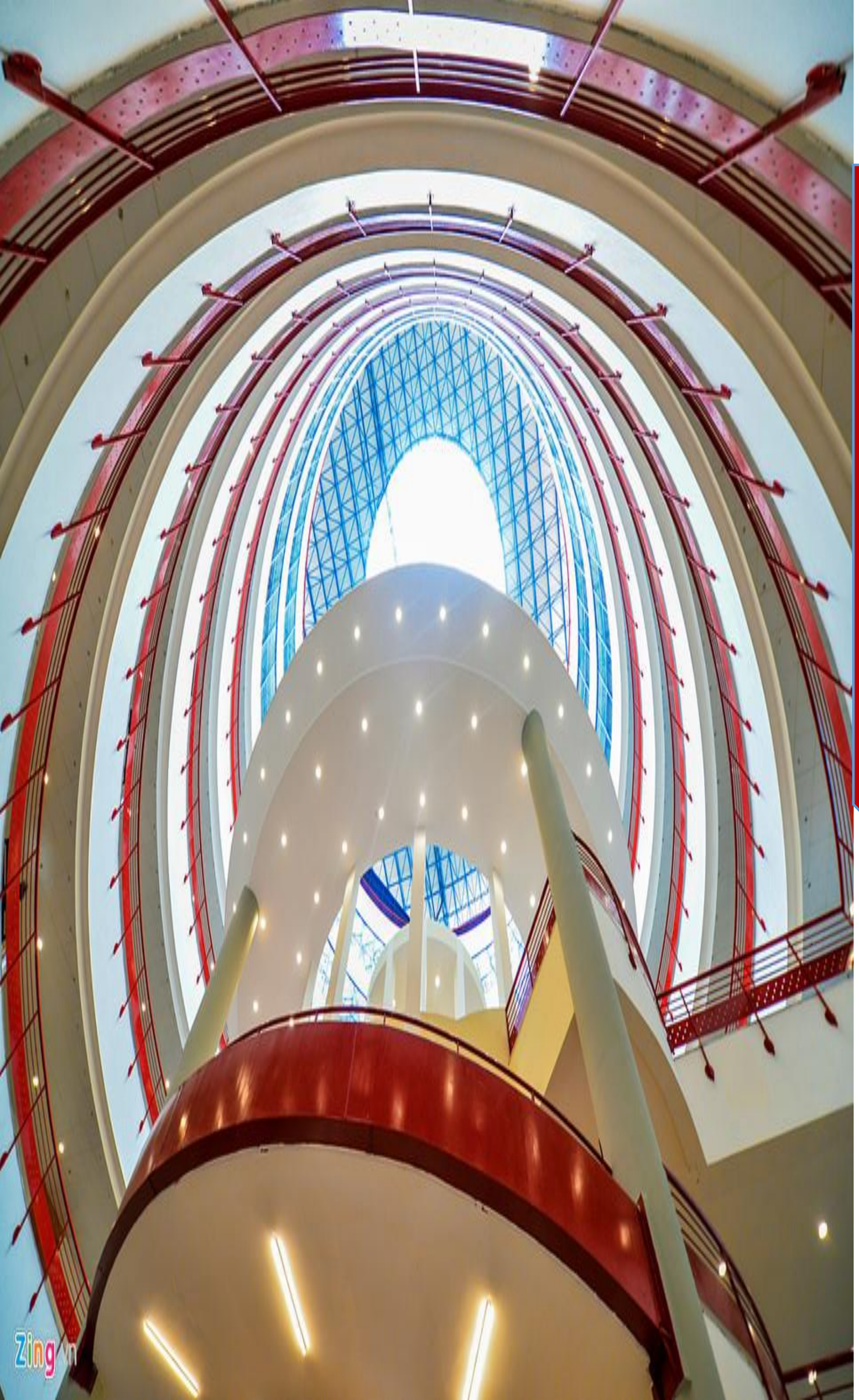

DATA PREPARATION AND VISUALIZATION

Department of Mathematical Economics

National Economics University
https://www.neu.edu.vn/

Introduction

Transformer Pipeline

- Scikit-Learn provides the Pipeline class to help with such sequences of transformations.
- The *Pipeline* constructor takes a list of name/estimator pairs defining a sequence of steps.
- When you call the pipeline's fit() method, it calls fit_tranforms() sequentially on all transformers, passing the output of each call as the parameter to the next call, until it reach the final estimator, for which it just calls the fit() method

Transformer Pipeline

- Ex:

```
from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([

                    ('imputer', Imputer(strategy = 'median')),\

                    ('std_scaler', StandardScaler()),\

                    ])
```

Custom Transformers

- Although Scikit-Learn provides many useful transformer, you will need to write your own for tasks such as custom cleanup operations or combining specific attribute
- You will want your transformer to work seamlessly with Scikit-Learn functionalities (such as pipelines), and since Scikit-Learn relies on duck typing (not inheritance), all you need is to create a class and implement three methods: fit() (returning self), transform(), and fit_transform().

Custom Transformers

- You can get the last one for free by simply adding TransformerMixin as a base class. Also, if you add BaseEstimator as a base class (and avoid *args and **kargs in your constructor)
- You will get two extra methods (get_params() and set_params()) that will be useful for automatic hyperparameter tuning.

Custom Transformers example

```
from sklearn.base import BaseEstimator, TransformerMixin
class AssertGoodHeader(BaseEstimator, TransformerMixin):

    def __init__(self, ):
        self.columns = None

    def fit(self, X, y=None):
        self.columns = X.columns
        return self

    def transform(self, X, y=None):
        return X[self.columns]

    def fit_transform(self, X, y=None, **fit_params):
        self.fit(X, y)
        return self.transform(X, y)
```

Exercise

READ *bank-additional-full.csv* FILE

Create a pipeline and test this pipeline

- Encode the *label* variable with numerical values in order to be able to build machine-learning models

- Encode the month, day_of_week, education, housing, loan and default attributes using the OrdinalEncoder class

- Encode the marital, poutcome, contact and job attribute using the OneHotEncoder class

- Transform the duration attribute using PowerTransformer and MinMaxscaler class

- Scale the age, cons.price.idx, cons.conf.idx, nr.employed attributes using MinMaxScaler class

- Use RandomForestClassifier() for modeling