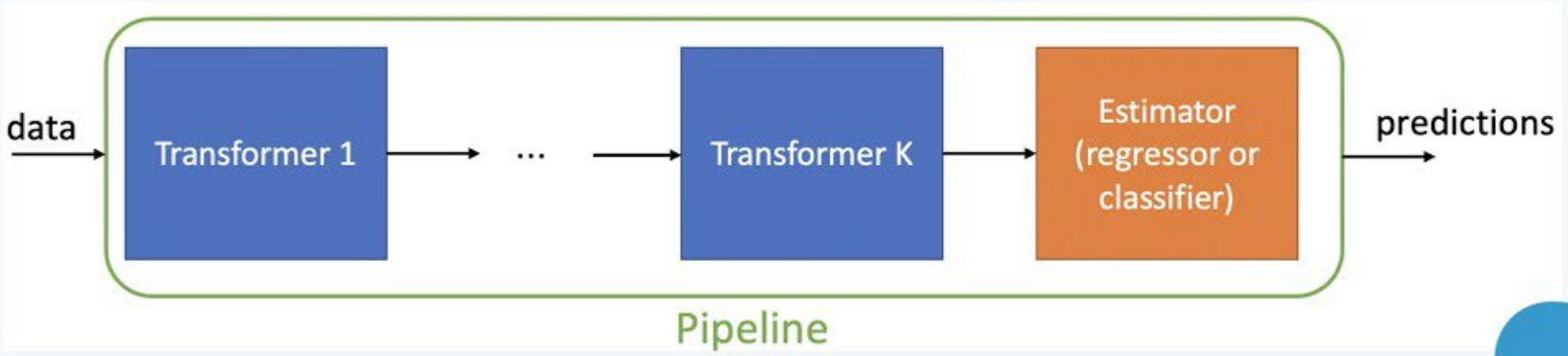
PIPELINES

PIPELINES

- Most data requires some preprocessing before modelling:
  - Categorical data usually needs to be encoded.
  - Numerical data usually needs to be scaled.
- Creating long series of steps to prep data can be cumbersome.
- Sklearn's pipeline can simplify and condense data prep into one function.
- Create a "pipe" where each step of prep and modelling is specified and executed together:
  - Fit the data to the pipe instead of the model.
  - The model is the last step of the pipe, the prep steps are listed in order.

Dr. Data Science

Understanding "Pipeline" in Machine Learning with Scikit-learn




```
from sklearn.pipeline import Pipeline
# creating pipeline and fitting it on data
pipe = Pipeline([('transformer',PolynomialFeatures(degree=2)),
                 ('estimator',LinearRegression())
                ])
```

scikit learn

USING PIPELINES

- Pipelines can contain most or all of our prep steps.
- Makes code easier to read and maintain.
- Can create custom transformers (we won't, but you can later) to apply other transformations.
- We want to structure all (if possible) prep steps into a pipeline, e.g.:
  - Encoding (e.g. one hot encoding of categorical variables).
  - Imputation (replacing missing values).
  - Feature selection of the useful features to keep.
  - Model (usually called the "estimator" in pipeline talk).

IN SKLEARN

- We make pipelines by defining a list of steps that we'll go through.
  - E.g. all the transformations and then the model.
- Once the pipeline exists, we use it "as" the model.
  - E.g. we fit the pipeline to the data, rather than the model.
- When the pipeline is "used" (fit, predict, score, etc..) it runs the data through the prep steps, then into the model. Just as if we did it piece by piece by hand.
  - The prep becomes "built-in" to the model.
- Helps us use many prep steps easily.

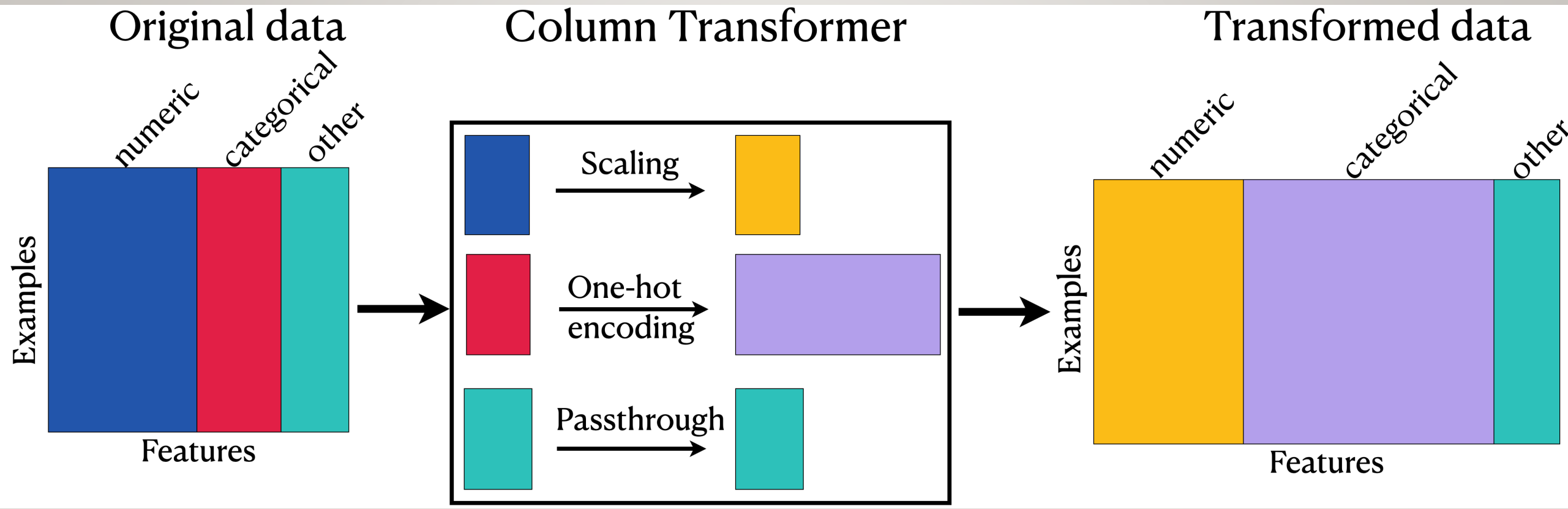
```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
x, y = make_classification(random_state=0)
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    random_state=0)
pipeline = Pipeline([('scaler', StandardScaler()), ('svc', SVC())])

pipeline.fit(x_train, y_train)
pipeline.score(x_test, y_test)
```

```
0.88
```

COLUMN TRANSFORMERS

- One issue with a pipeline is that everything goes through all the steps.
- What if we need to do different things to different data?
  - E.g. One-hot encode the categorical, scale the numerical?
- The universe has an answer – the Column Transformer!
- A column transformer is effectively a multi-route pipeline.
  - We specify different routes – each their own pipeline.
  - We define which features go through which route.
  - The column transformer sends data down its path and recombines it after.

```
si_cat = SimpleImputer(strategy='most_frequent')
ohe = OneHotEncoder(sparse=False, handle_unknown='ignore')

si_cont = SimpleImputer(strategy='mean')
ss = StandardScaler()

cat_cols = ['Neighborhood', 'LotShape', 'OverallQual', 'MasVnrType']
cont_cols = ['GrLivArea', 'GarageArea', 'LotFrontage']

cat_steps = [
    ('si', si_cat),
    ('ohe', ohe)
]
cat_pipe = Pipeline(cat_steps)

cont_steps = [
    ('si', si_cont),
    ('ss', ss)
]
cont_pipe = Pipeline(cont_steps)

transformers = [
    ('cat_cols', cat_pipe, cat_cols),
    ('cont_cols', cont_pipe, cont_cols)
]
ct = ColumnTransformer(transformers)

ls = Lasso()

steps = [
    ('ct', ct),
    ('ls', ls)
]

final_pipe = Pipeline(steps)
```


Original data
Column Transformer
Transformed data
numeric
categorical
other
Examples
Features
Scaling
One-hot
encoding
Passthrough
numeric
categorical
other
Examples
Features

CONCLUSION

- Pipelines and column transformers both make our code easier to use and maintain.
- These tools, and most other sklearn items, can be combined in almost any arrangement.
  - This goes beyond us a bit, but as long as a "step" meets the requirements that sklearn is expecting (take in an array/df, do some changes, return an array), it can be used at any step in a pipeline.
  - We can make a CT of pipelines, a pipeline of CTs, each with encoding/imputing, etc…
  - It is possible to make custom "steps" (transformers) to do whatever we want.
  - Once we get used to the structure, it is much easier to prepare data.
- Use pipelines, unless you have an explicit reason not to.