# Warsztaty modelowania

## 06 – optymalizacja i automatyzacja

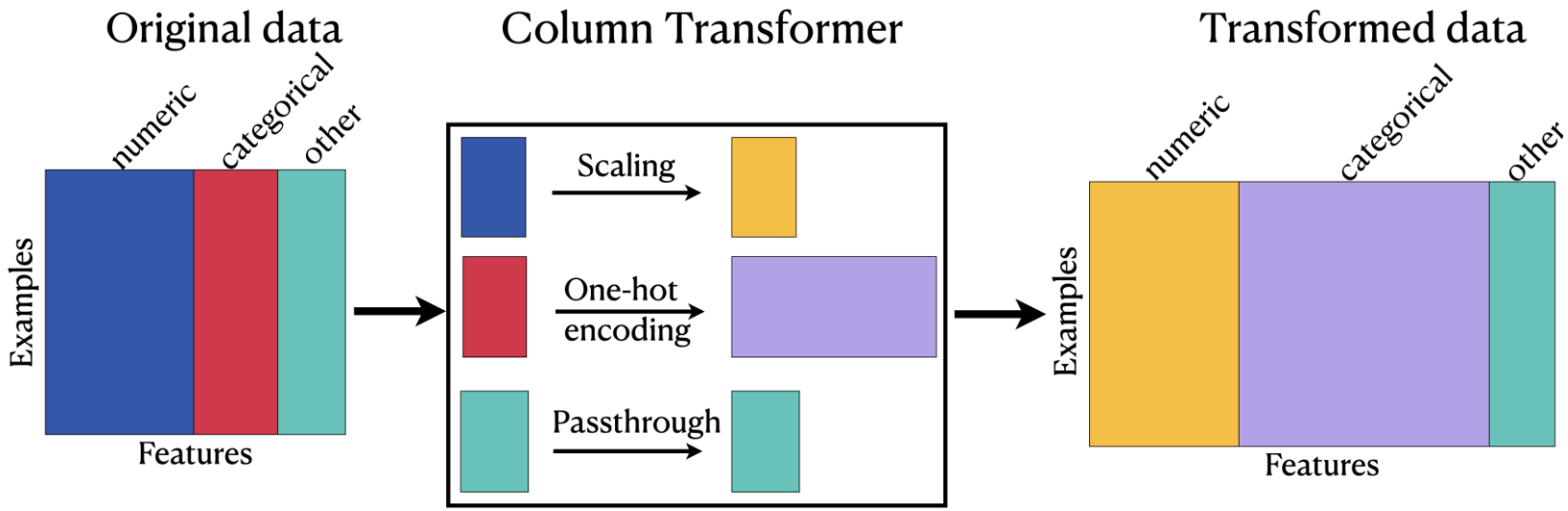
opracowała Patrycja Naumczyk

# Optymalizacja i automatyzacja

1. Automatyzacja cz. I:
   a) przekształcanie zmiennych
2. Optymalizacja:
   a) walidacja krzyżowa
   b) selekcja zmiennych wyjaśniających
3. Automatyzacja cz.II
   a) pipeline’y

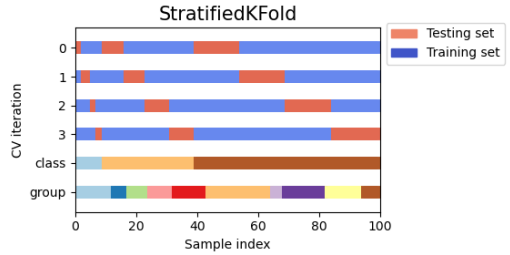
# Automatyzacja cz. I – przekształcanie zmiennych

1. Zbiorowe kodowanie i skalowanie – ColumnTransformer()
2. Ważne parametry:
   a) transformers – lista tupli:
      i. nazwa własna (str)
      ii. transformer - estymator (ew. ‚drop' lub ‚passthrough')
      iii. kolumny (np. lista)
   b) remainder – estymator (ew. ‚drop' lub ‚passthrough')
   c) sparse_threshold
   d) verbose
3. Ważne metody:
   a) .fit()
   b) .transform()
   c) .get_feature_names_out()
4. Ważne atrybuty:
   a) .transformers_
   b) .feature_names_in_

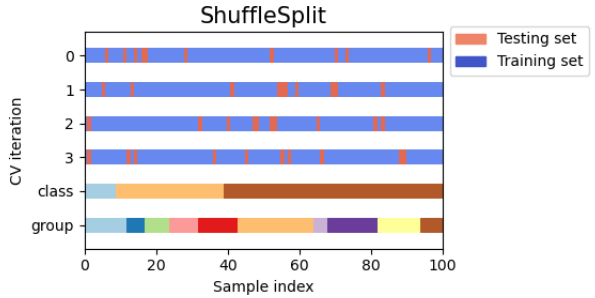


# Optymalizacja – walidacja krzyżowa (cross-validation)

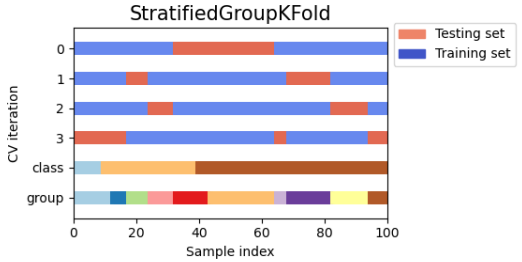
1. Optymalizacja uczenia modelu na secie TRENINGOWYM
2. Podstawowe rodzaje CV w scikit-learn:
   a) StratifiedKFold()

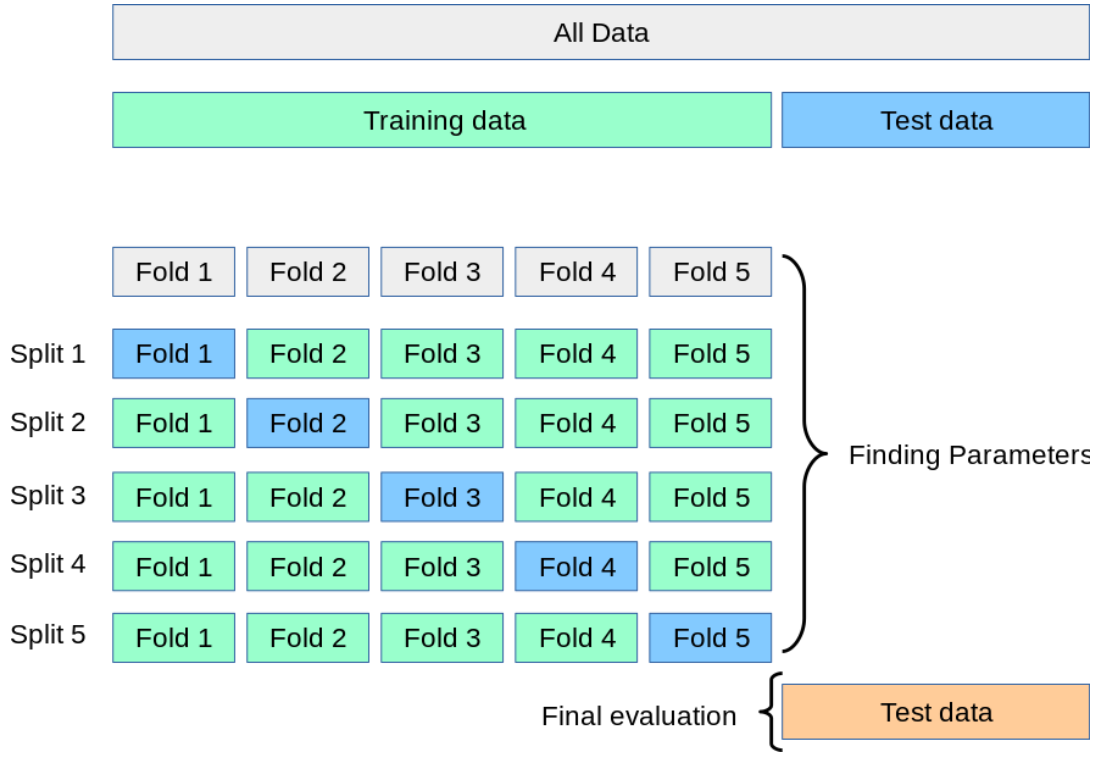



   b) ShuffleSplit()



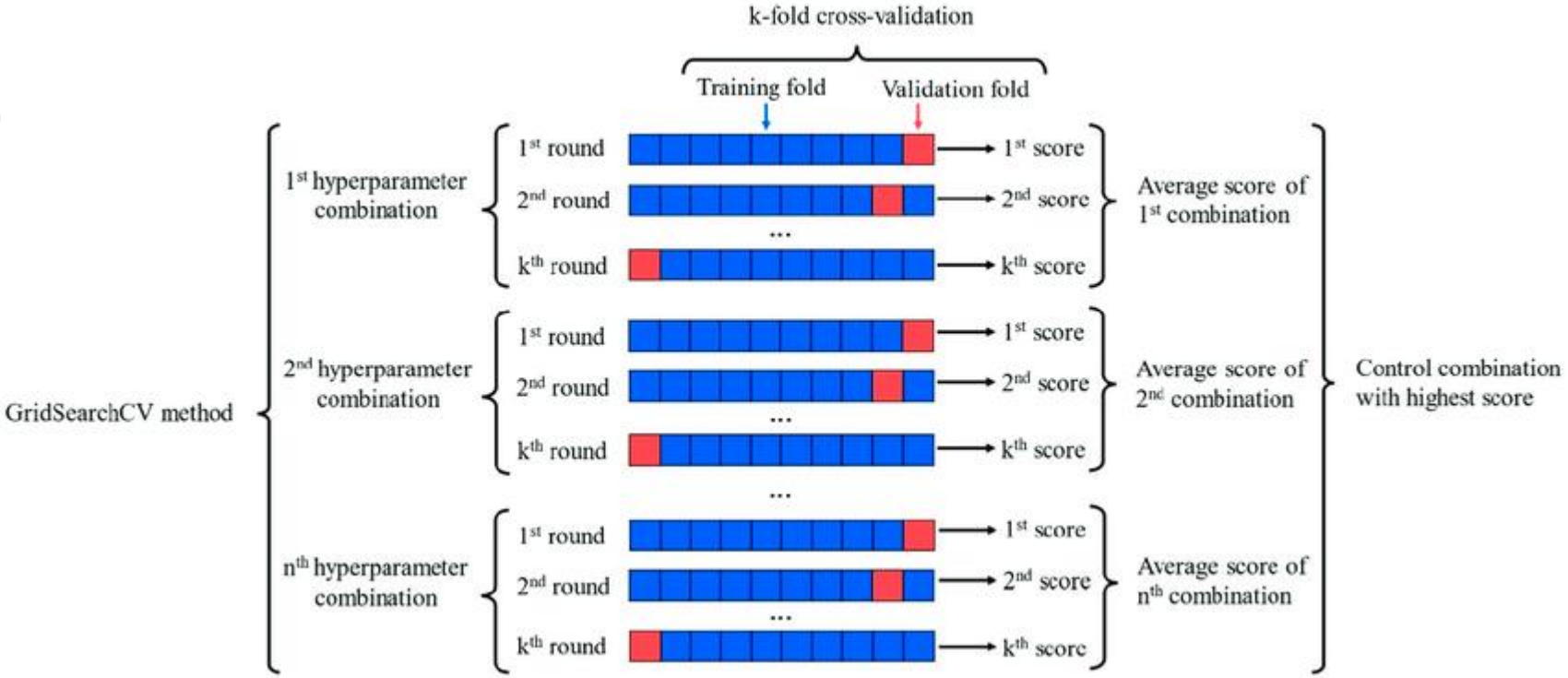

   c) StratifiedGroupKFold()



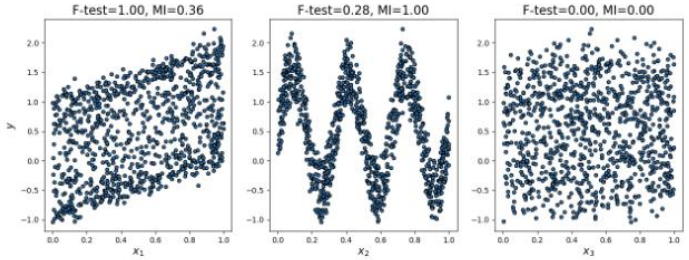
# Optymalizacja – walidacja krzyżowa (cross-validation)

3. Optymalizacja parametrów – GridSearchCV()
4. Ważne parametry:
   a) estimator
   b) param_grid (słownik !)
   c) scoring
   d) cv
5. Ważne atrybuty:
   a) .cv_results_
   b) .best_estimator_
   c) .best_params_
   d) .feature_names_in_
6. Ważne metody:
   a) .fit()
   b) .transform()
   c) .predict()



# Optymalizacja – selekcja zmiennych (feature selection)

1. Moduł feature_selection w scikit-learn
2. Wybór jednoczynnikowy (związek ze zmienną wyjaśnianą)
   a) SelectKBest() i SelectPercentile()
   b) r_regression(), f_regression(), mutual_info_regression()




**SelectKBest( mutual_info_regression, k=4)** → .fit() → **SelectKBest( mutual_info_regression, k=4)** → .transform()

```
[[1. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 1. ... 1. 0. 0.]
 ...
 [0. 0. 0. ... 0. 1. 0.]
 [1. 1. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 1. 0.]]
```

**Atrybuty**:
.scores_
.pvalues_
.feature_names_in_

**Metody**:
.get_feature_names_out()

3. Recursive Feature Elimination (RFE)
   a) metoda eliminacji wstecznej na podstawie współczynników modelu ( $\beta$ )

**RFE(PoissonRegressor(), n_features_to_select)** → .fit() → **RFE(PoissonRegressor(), n_features_to_select)** → .transform()

```
[[1. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 1. ... 1. 0. 0.]
 ...
 [0. 0. 0. ... 0. 1. 0.]
 [1. 1. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 1. 0.]]
```

**Atrybuty**:
.feature_names_in_
.support_
.ranking_

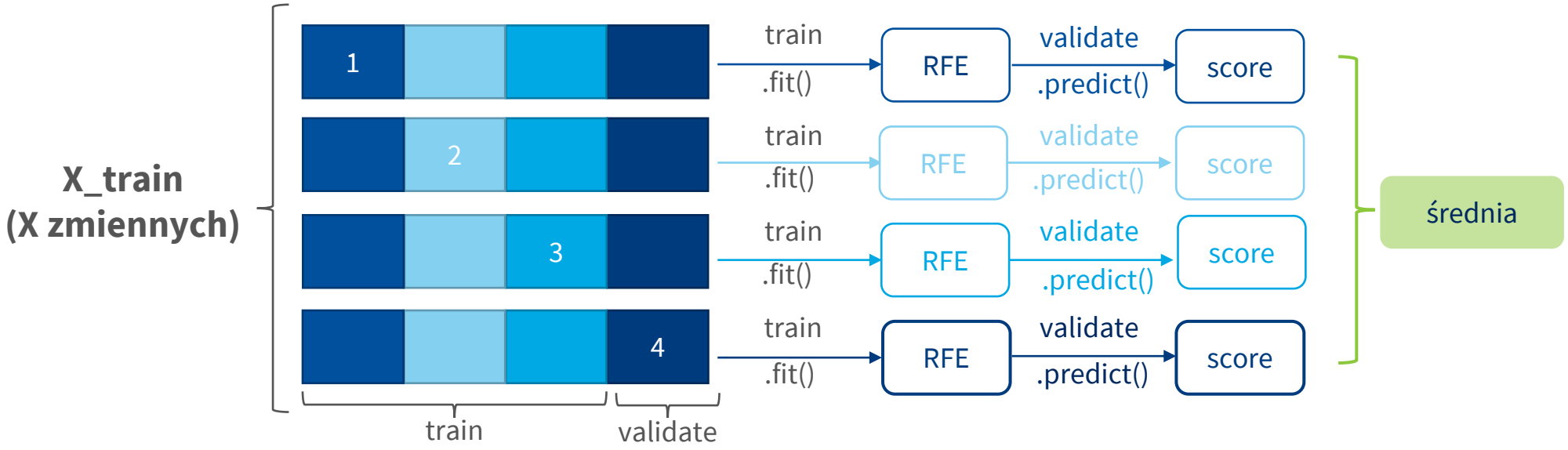
**Metody**:
.get_feature_names_out()

.predict()

```
array([0.40164084, 0.49078604, 0.46763855, 0.47875   , 0.41495275,
       0.47448992, 0.44240534, 0.4262428 , 0.37862659, 0.49555404,
       0.47602527, 0.37936182, 0.40186752, 0.38349234, 0.49078604,
       0.49729195, 0.42151945, 0.46085436, 0.38349234, 0.38956118,
       0.47547504, 0.47641806, 0.49729195, 0.48774034, 0.38956118,
       0.41096026, 0.3818036 , 0.37606936, 0.42151945, 0.45990434,
       0.49643638, 0.44736766, 0.47514809, 0.45711726, 0.49555404,
       0.41495275, 0.47221076, 0.47804511, 0.40749027, 0.4860358 ,
       0.49763413, 0.47875   , 0.37503606, 0.46955453, 0.48135939,
       0.41096026, 0.49729195, 0.45460398, 0.46853901, 0.39414875,
       0.48774034, 0.45711726, 0.44736766, 0.38370878, 0.51788313,
       0.37862659, 0.45711726, 0.44240534, 0.37503606, 0.42416317,
       0.38370878, 0.38956118, 0.40186752, 0.47736295, 0.43750289,
       0.47221076, 0.46893189, 0.49729195, 0.48058266, 0.47875   ,
       0.39469136, 0.45711726])
```

# Optymalizacja – selekcja zmiennych (feature selection)

4. Recursive Feature Elimination with Cross-Validation (RFECV)
   a) selekcja optymalnej liczby zmiennych na bazie RFE




**RFECV(PoissonRegressor(), cv, scoring)** → .fit() → **RFECV(PoissonRegressor(), cv, scoring)** → .transform()

**Atrybuty**:
.cv_results_
['mean_test_score']
['splitX_test_score']
['std_test_score']
.feature_names_in_
.support_
.ranking_

**Metody**:
.get_feature_names_out()

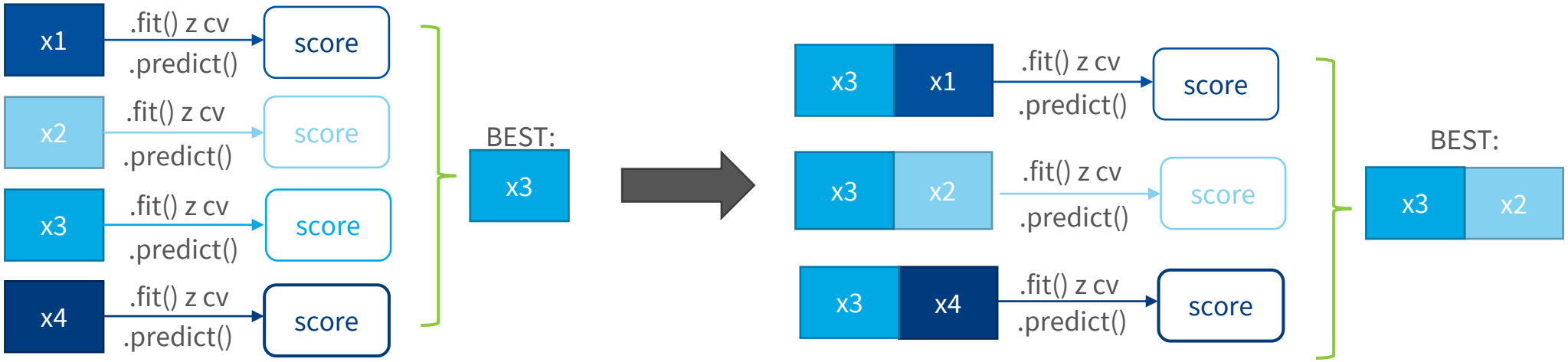
.predict()

```
[[1. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 1. ... 1. 0. 0.]
 ...
 [0. 0. 0. ... 0. 1. 0.]
 [1. 1. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 1. 0.]]
```

```
array([0.40164084, 0.49078604, 0.46763855, 0.47875   , 0.41495275,
       0.47448992, 0.44240534, 0.4262428 , 0.37862659, 0.49555404,
       0.47602527, 0.37936182, 0.40186752, 0.38349234, 0.49078604,
       0.49729195, 0.42151945, 0.46085436, 0.38349234, 0.38956118,
       0.47547504, 0.47641806, 0.49729195, 0.48774034, 0.38956118,
       0.41096026, 0.3818036 , 0.37606936, 0.42151945, 0.45990434,
       0.49643638, 0.44736766, 0.47514809, 0.45711726, 0.49555404,
       0.41495275, 0.47221076, 0.47804511, 0.40749027, 0.4860358 ,
       0.49763413, 0.47875   , 0.37503606, 0.46955453, 0.48135939,
       0.41096026, 0.49729195, 0.45460398, 0.46853901, 0.39414875,
       0.48774034, 0.45711726, 0.44736766, 0.38370878, 0.51788313,
       0.37862659, 0.45711726, 0.44240534, 0.37503606, 0.42416317,
       0.38370878, 0.38956118, 0.40186752, 0.47736295, 0.43750289,
       0.47221076, 0.46893189, 0.49729195, 0.48058266, 0.47875   ,
       0.39469136, 0.45711726])
```

# Optymalizacja – selekcja zmiennych (feature selection)

5. Sequential Feature Selector (SFS)
   a) regresja z eliminacją wsteczną lub **wprzód**




**SequentialFeatureSelector (PoissonRegressor(), cv, scoring)**

.fit()

**SequentialFeatureSelector (PoissonRegressor(), cv, scoring)**

**Atrybuty**:
.feature_names_in_
.support_

**Metody**:
.get_feature_names_out()

.transform()

```
[[1. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 1. ... 1. 0. 0.]
 ...
 [0. 0. 0. ... 0. 1. 0.]
 [1. 1. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 1. 0.]]
```

.predict()

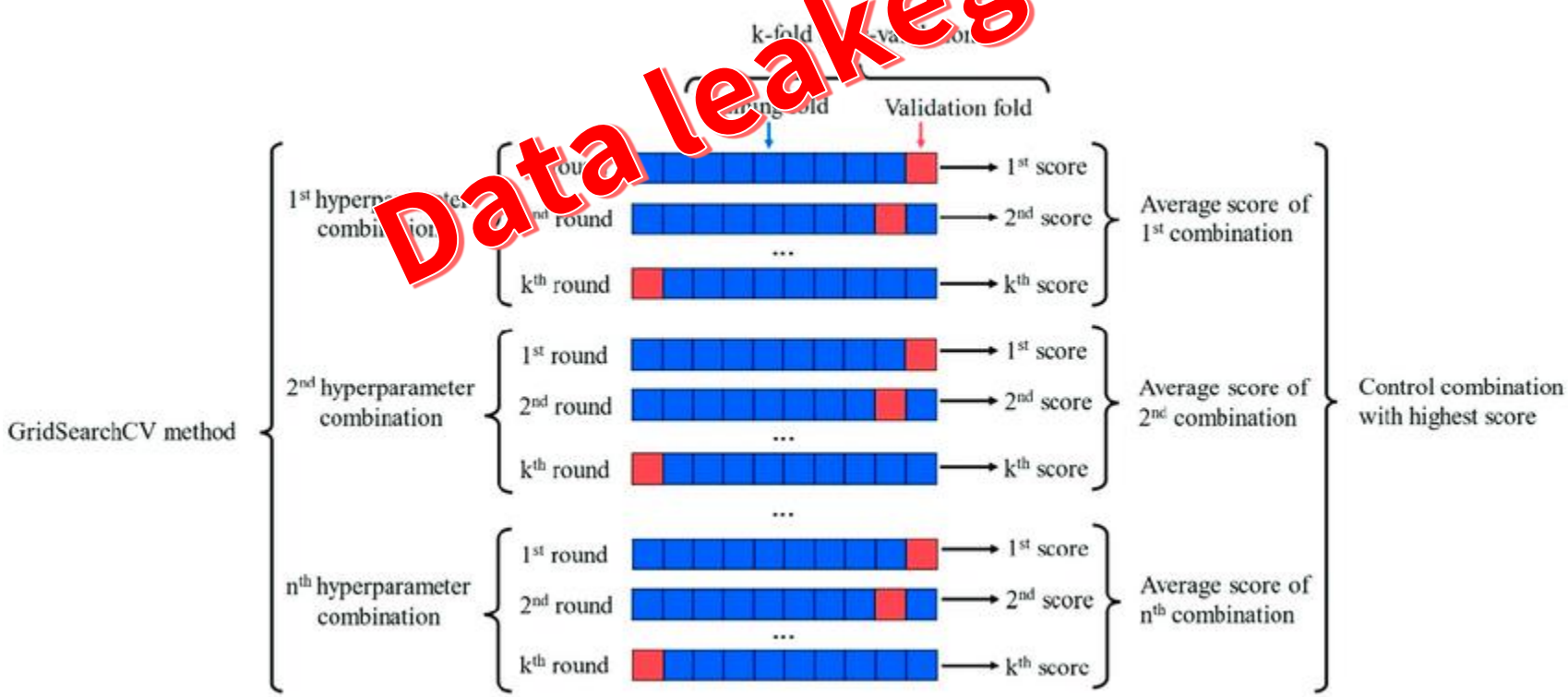
```
array([0.40164084, 0.49078604, 0.46763855, 0.47875   , 0.41495275,
       0.47448992, 0.44240534, 0.4262428 , 0.37862659, 0.49555404,
       0.47602527, 0.37936182, 0.40186752, 0.38349234, 0.49078604,
       0.49729195, 0.42151945, 0.46085436, 0.38349234, 0.38956118,
       0.47547504, 0.47641806, 0.49729195, 0.48774034, 0.38956118,
       0.41096026, 0.3818036 , 0.37606936, 0.42151945, 0.45990434,
       0.49643638, 0.44736766, 0.47514809, 0.45711726, 0.49555404,
       0.41495275, 0.47221076, 0.47804511, 0.40749027, 0.4860358 ,
       0.49763413, 0.47875   , 0.37503606, 0.46955453, 0.48135939,
       0.41096026, 0.49729195, 0.45460398, 0.46853901, 0.39414875,
       0.48774034, 0.45711726, 0.44736766, 0.38370878, 0.51788313,
       0.37862659, 0.45711726, 0.44240534, 0.37503606, 0.42416317,
       0.38370878, 0.38956118, 0.40186752, 0.47736295, 0.43750289,
       0.47221076, 0.46893189, 0.49729195, 0.48058266, 0.47875   ,
       0.39469136, 0.45711726])
```

# Automatyzacja – ścieżki przekształceń (pipeline)

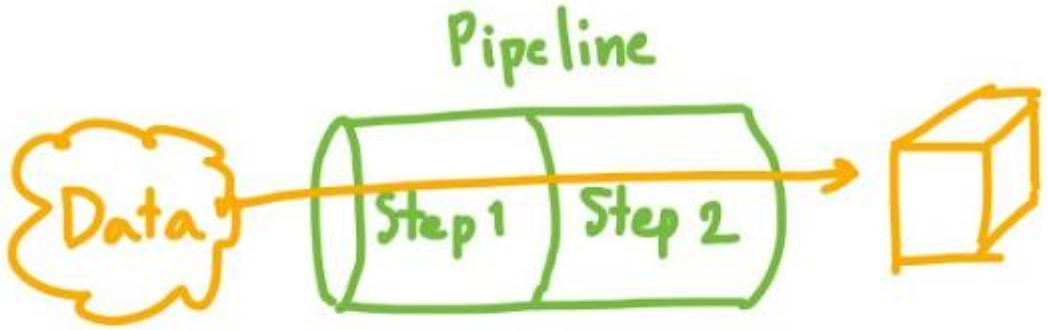
In [12]:

```
poisson_gscv = GridSearchCV(
    PoissonRegressor(),
    param_grid = params,
    scoring='neg_mean_poisson_deviance',
    cv=10
).fit(df_train_trans, df_train.czy_[illegible]zkoda)
poisson_gscv
```

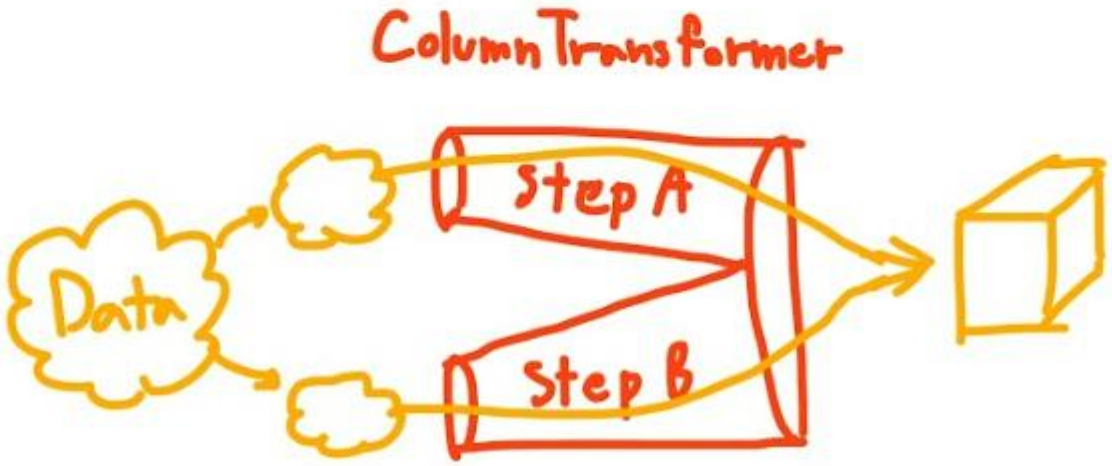
Data leakege!!!



# Automatyzacja – ścieżki przekształceń (pipeline)




```
Pipeline(steps=[
                (nazwa_str, transformer),
                (nazwa_str, transformer),
                …
])
```

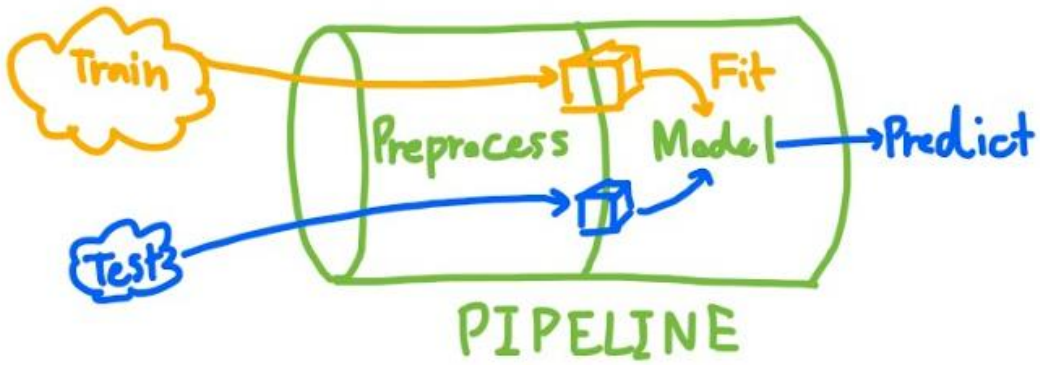


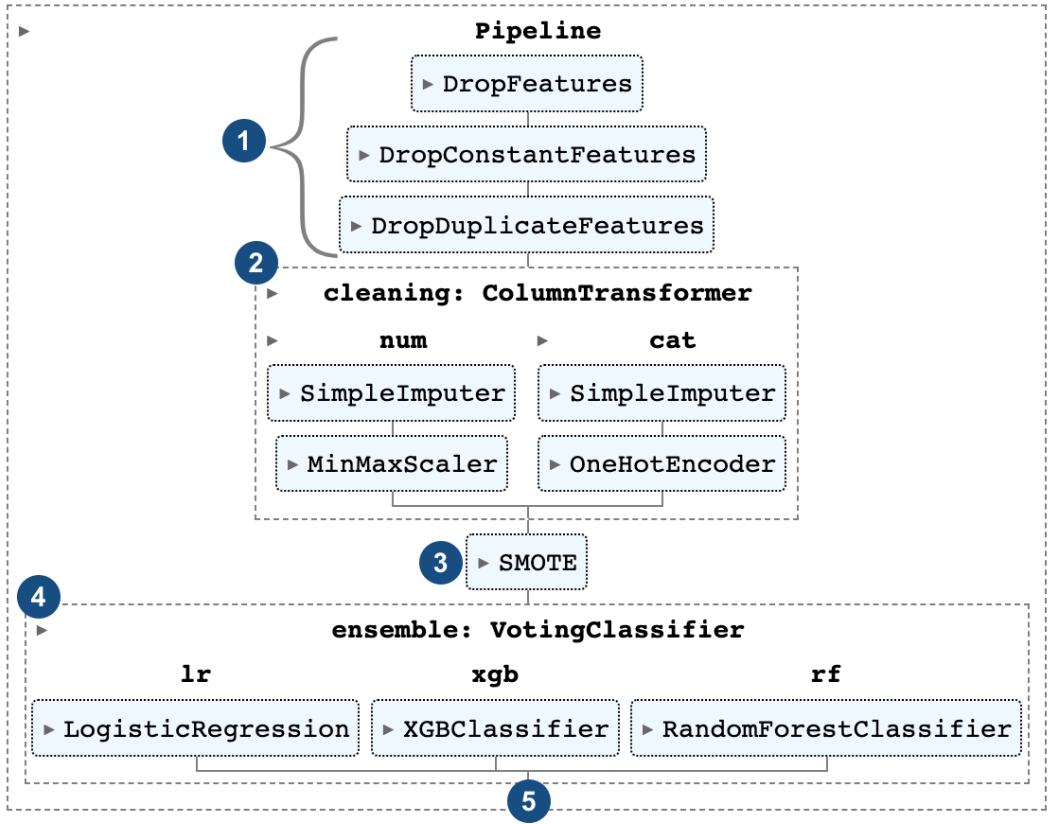

```
ColumnTransformer(transformers=[
                (nazwa_str, transformer, kolumny),
                (nazwa_str, transformer, kolumny),
                …
])
```

# Automatyzacja – ścieżki przekształceń (pipeline)

1. Ścieżki „krok po kroku” – Pipeline()
2. Ważne parametry:
   a) steps – lista tupli:
      i. nazwa własna (str)
      ii. transformer - estymator
   b) verbose
3. Ważne metody:
   a) .fit()
   b) .predict()
   c) .transform()
   d) .get_params()
   e) .get_feature_names_out()
4. Ważne atrybuty:
   a) .feature_names_in_
5. „Ładne” wyświetlanie:

```
from sklearn import set_config

set_config(display='diagram')
```



# Automatyzacja – ścieżki przekształceń (pipeline)

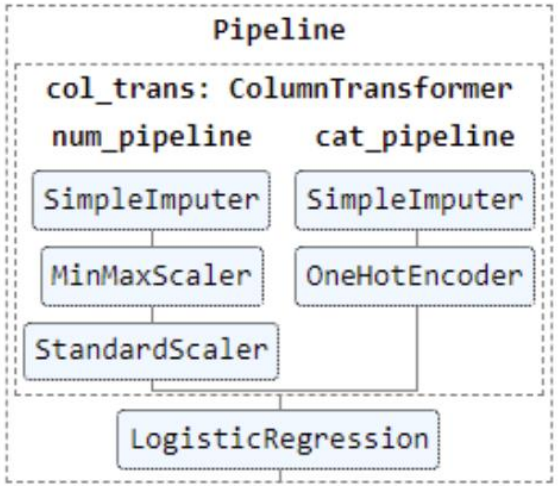
1. GridSearchCV() z pipeline:
   a) Optymalizacja parametrów bez data likeage

```
params = {
    "alpha": list(np.logspace(-20, 1, num=100)),
#     "solver" :['lbfgs', 'newton-cholesky']
}

poisson_gscv__data_leakage = GridSearchCV(
    PoissonRegressor(),
    param_grid = params,
    scoring=scores,
    cv=10,
    refit=False
).fit(df_train_trans, df_train.czy_szkoda)
```

```
params = {
    "model__alpha": list(np.logspace(-20, 1, num=100)),
#     "solver" :['lbfgs', 'newton-cholesky']
}

poisson_gscv__pipe = GridSearchCV(
    pipeline,
    param_grid = params,
    scoring=scores,
    cv=10,
    refit=False
).fit(df_train, df_train.czy_szkoda)
```

   b) Weryfikacja optymalnej metody przetwarzania

```
grid_step_params = [{'col_trans__num_pipeline__minmax_scale': ['passthrough']},
                    {'col_trans__num_pipeline__std_scale': ['passthrough']}]
```




   c) Oba ww. na raz:

```
merge_dict = {**dict_1,**dict_2}
```

```
step_params = [
    {**params, **{"col_trans__minmax_numeric": ['passthrough']}},
    {**params, **{"col_trans__standard_numeric": ['passthrough']}}
]
```