betsson group

Predicting future calls to the Customer Service

Business presentation

Cainã Max Couto da Silva

April 2024

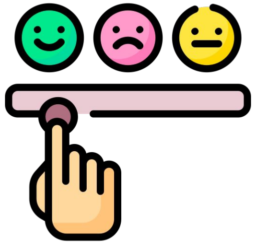
Business context

For efficiency purposes, Betsson is trying to predict which customers are going to call the Customer Service, based on their past behaviour.

For a given day, predict whether a customer will call the Customer Service in the following 14 days.

Why? → **Optimize** customer relationship →

therefore reducing churn



Data Insights

Solution

Model deployment

GIF

EDA

Data quality checks

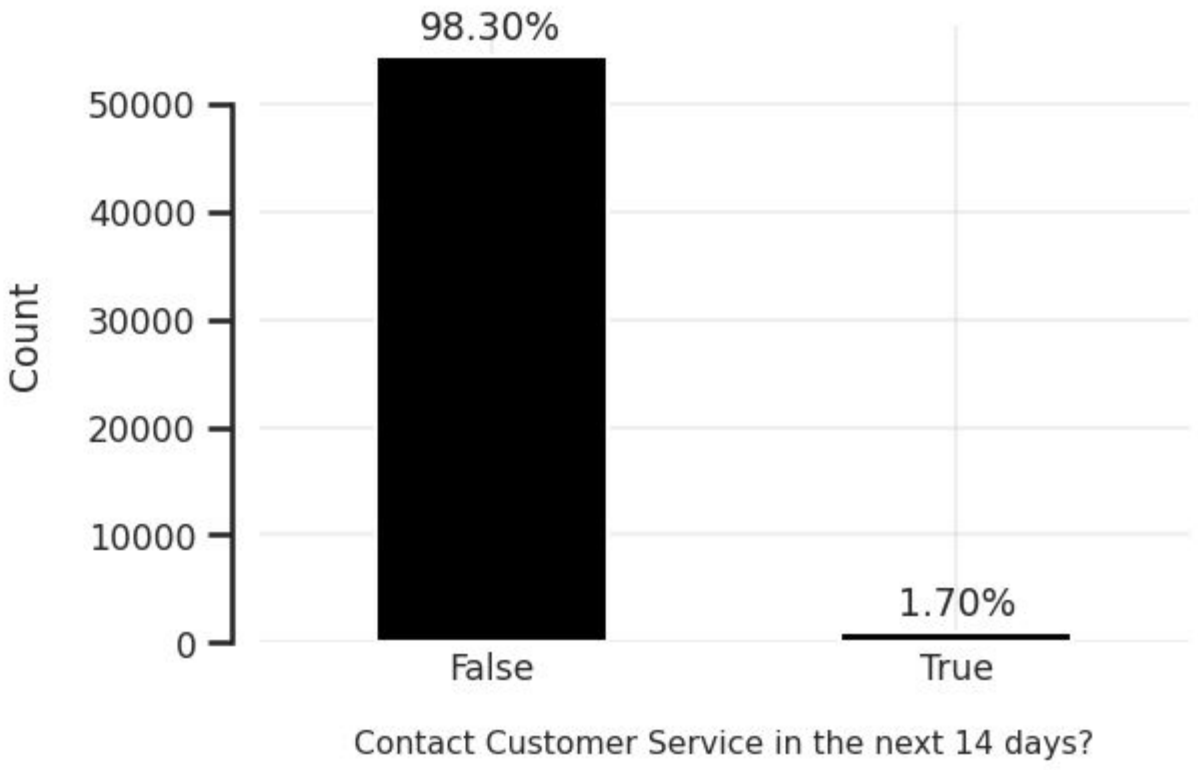
Duplicates

No missing values (probably they’ve been filled with -1)

Removal of constant features

Statistical Analysis

Feature vs target



Feature selection

There are almost 300 features in the provided dataset.

- Simple models are easier to interpret
- Shorter training times
- Enhanced generalization by reducing overfitting
- Easier to implement by software developers
- Variable redundancy
- Bad learning behaviour in high dimensional spaces

Model

Model selection

What will you learn?

- What data leakage is
- How to avoid data leakage
- How to identify data leakage

- Learn how to use machine learning pipelines
- Understand how machine pipeline works elsewhere

Note: This workshop is not for you if you are highly familiar with machine learning pipelines (*i.e.* column transformers & pipelines) and is confident about your ability to avoid data leakage.

Requirements

Theory (slides)

Good to have
- Basic machine learning knowledge
- Plus: familiarity with scikit-learn

Practice (hands-on examples)

Expected:
- Python (core functions & pandas)

Good to have
- Basic machine learning knowledge
- Basic familiarity with scikit-learn

Agenda

01 **Machine learning lifecycle**
Framework, validation & feature engineering

02 **Data Leakage**
Definition and common mistakes

03 **Scikit-learn pipelines (focus on pre-processing)**
Transformers, estimators, ColumnTransfomers & Pipelines

04 **Hands-on!**
Walkthrough our code!

Notebooks:

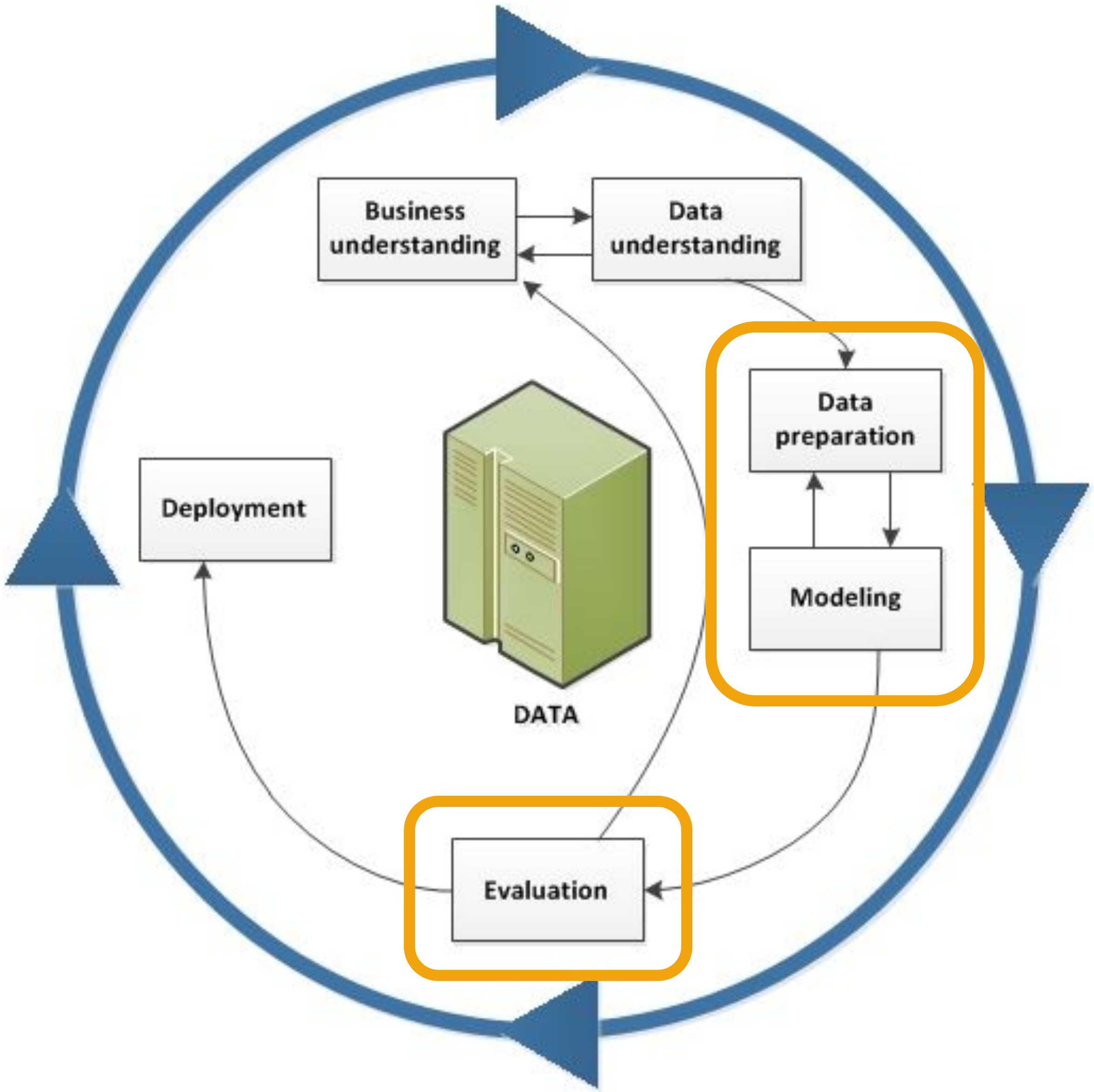
1. Pre-processing
   (manual & column transformers)
2. Scikit-learn and feature-engine pipelines
3. Imbalanced-learn pipelines
4. PyCaret AutoML pipelines
5. SparkML pipeline

cmcouto-silva/Talks/tree/main/PyDataGlobal

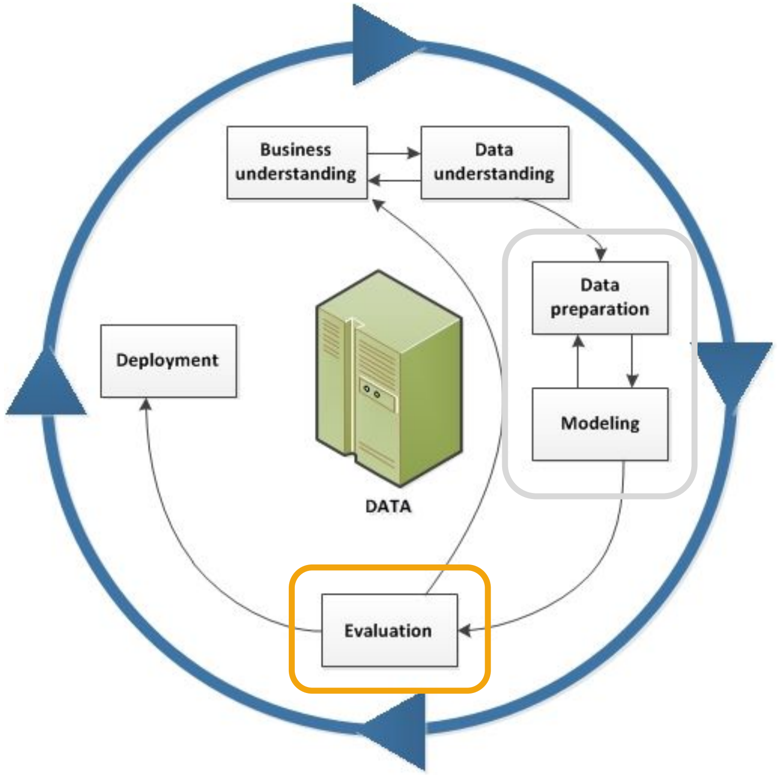
Let’s begin!

Machine learning lifecycle

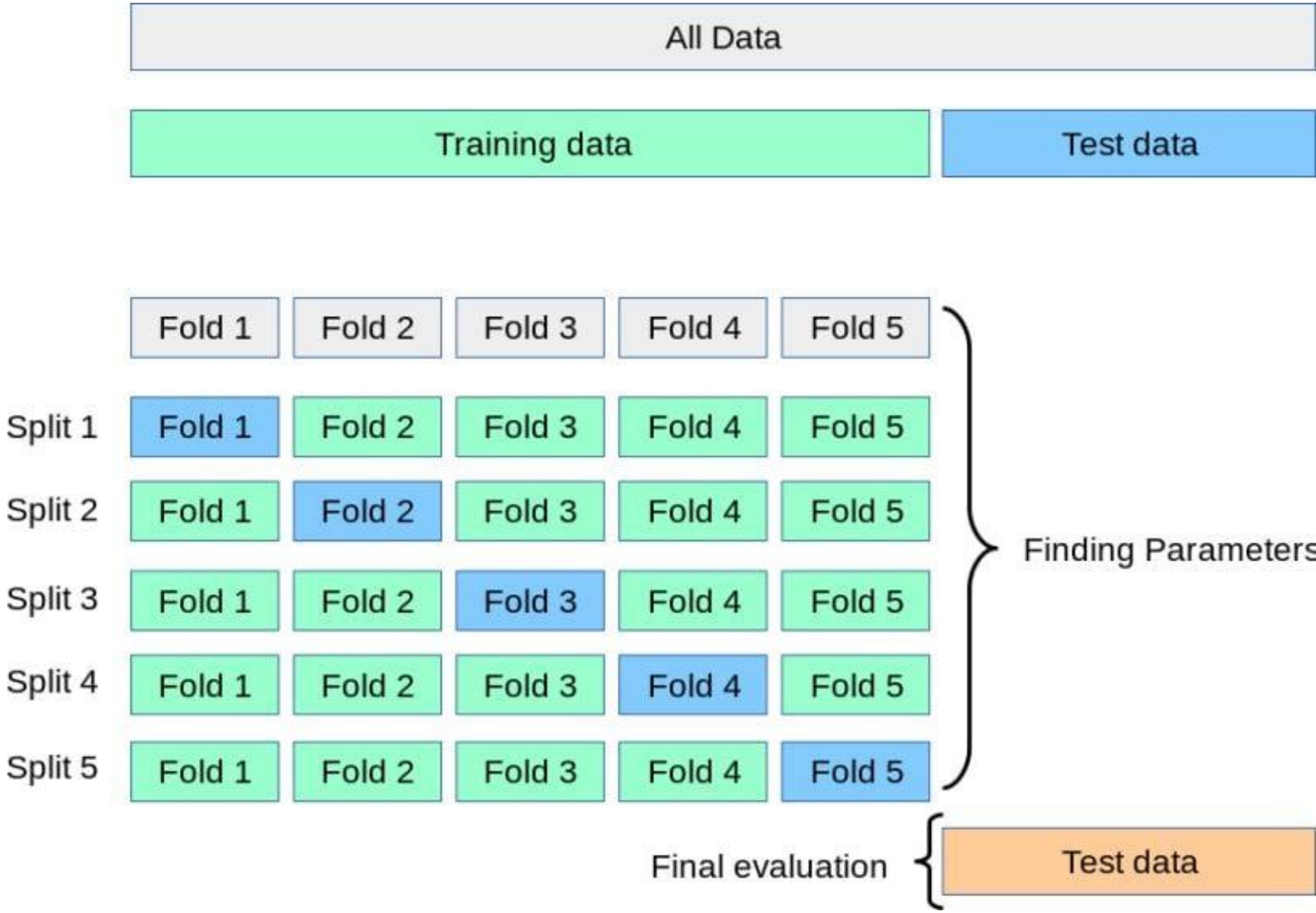
Data Scientist



Machine learning lifecycle

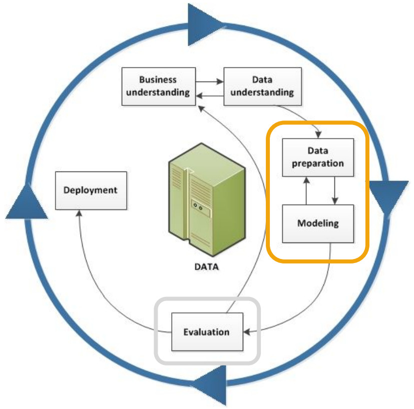


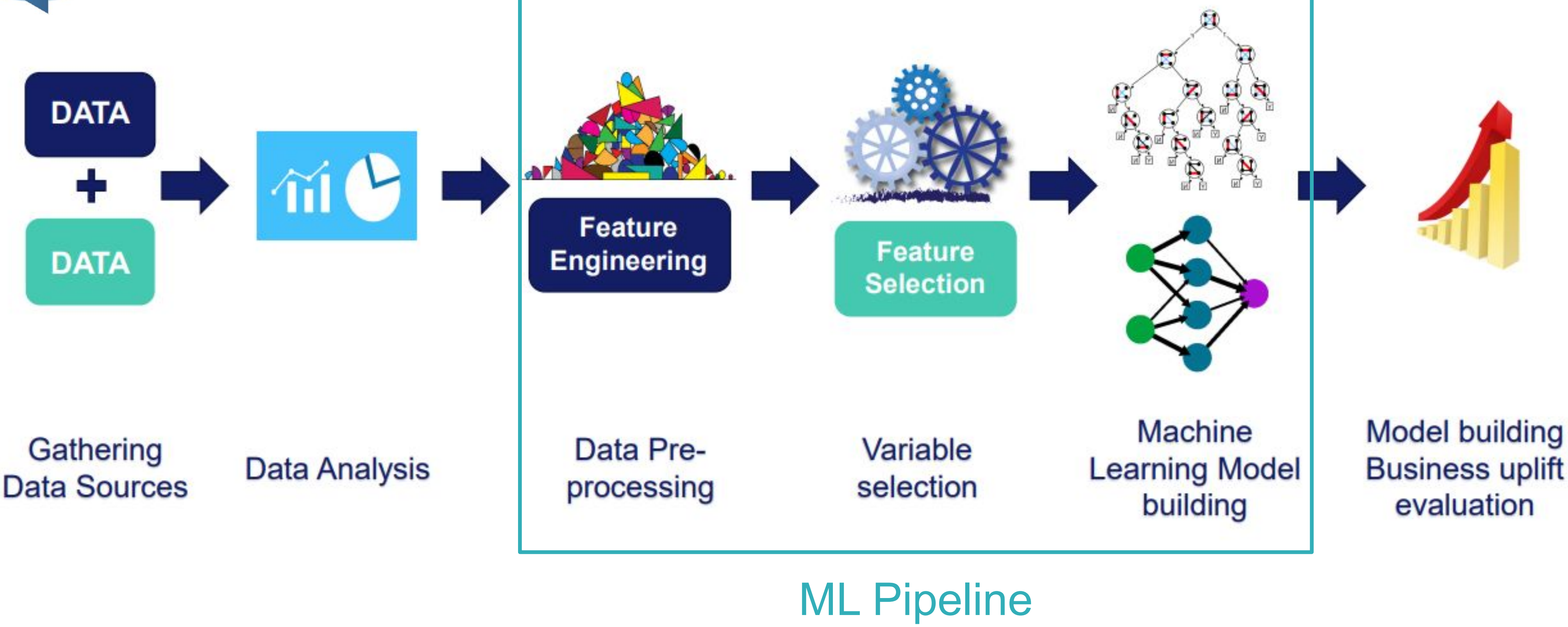
The test should know **nothing** about the training set!



We want a good generalist model!

Machine learning lifecycle





Feature engineering

Missing values

Models do not accept missing values.

Categorical variables

Models do not accept strings.

Cardinality: high number of labels
Rare labels: infrequent categories

Distribution

A better spread of values may benefit performance

Outliers

This can lead to tremendous weight and bad generalization

Feature engineering

Numeric feature scaling

Z-score and min-max normalization.

Categorical feature encoding

One-hot, ordinal frequency or target encoding.

Feature selection

Dimensionality reduction, variable importances and regularization.

Feature transformation

Logarithmic, polynomial, binning.

Feature interaction

Product of two variables, custom combinations.

Handling data imbalance

Random under- and upper-sampling, SMOTE.

Note: depending on the data type (text data, time-series, images), distinct pre-processing should be applied.

When does it break?

When the training data includes information that allows the model to make predictions that it cannot make in a real-world scenario. It usually leads to overoptimistic results.

Analogy: It’s like having the answers (or tips) to an exam in advance —
it doesn't truly test your knowledge!

Data leakage is a flaw in machine learning that leads to overoptimistic results

Data leakage can be a multi-million-dollar mistake in many data science applications 💸

Data leakage: common reasons

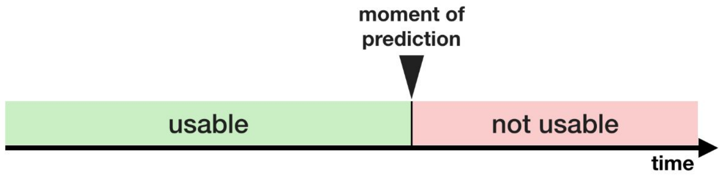
L1

Lack of clean separation of training and test dataset

- No test set
- Pre-processing on training and test set
- Feature selection on training and test set
- Duplicates in datasets

L2

Model uses features that are not legitimate



Handling unbalanced data

- Resampling test data

L3

Test set is not drawn from the distribution of scientific interest

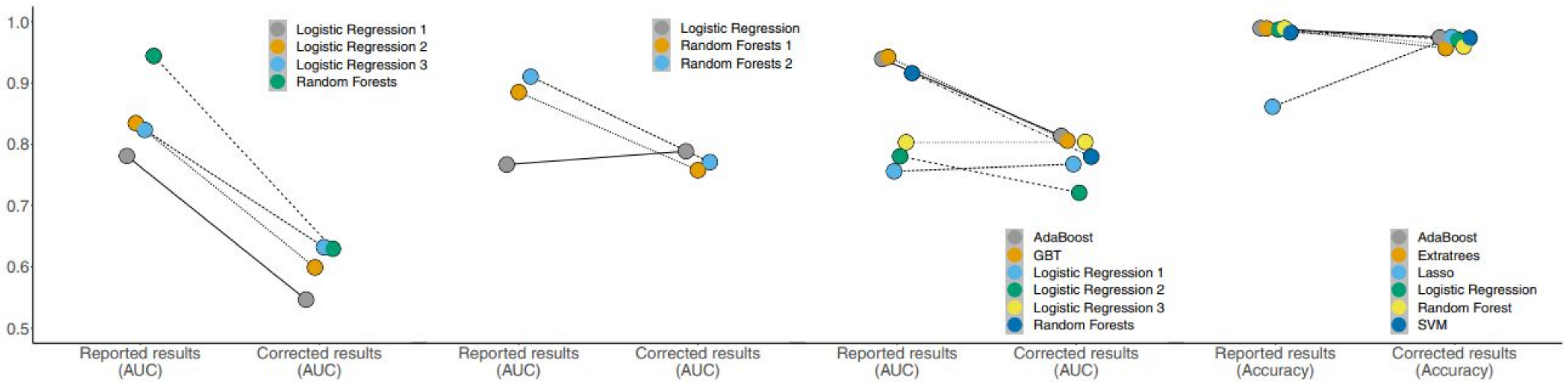
- Temporal leakage
- Nonindependence between training and test samples
- Sampling bias in test distribution

Article

Patterns

Leakage and the reproducibility crisis in machine-learning-based science

Sayash Kapoor & Arvind Narayanan (2023)



Paper	Muchlinski et al.	Colaresi and Mahmood	Wang	Kaufman et al.
Claim	Random Forests model drastically outperforms Logistic regression models	Random Forests models drastically outperform Logistic regression model	Adaboost and Gradient Boosted Trees (GBT) drastically outperform other models	Adaboost outperforms other models
Error	**[L1.2] Pre-proc. on train-test** (Incorrect imputation)	**[L1.2] Pre-proc. on train-test** (Incorrect reuse of an imputed dataset)	**[L1.2] Pre-proc. on train-test**. (Incorrect reuse of an imputed dataset) **[L3.1] Temporal leakage** (*k*-fold cross validation with temporal data)	**[L2] Illegitimate features** (Data leakage due to proxy variables) **[L3.1] Temporal leakage** (*k*-fold cross validation with temporal data)
Impact	Random Forests perform no better than Logistic Regression	Random Forests perform no better than Logistic Regression	Difference in AUC between Adaboost and Logistic Regression drops from 0.14 to 0.01	Adaboost no longer outperforms Logistic Regression. None of the models outperform a baseline model that predicts the outcome of the previous year
Discussion	Impact of the incorrect imputation is severe since 95% of the out-of-sample dataset is missing and is filled in using the incorrect imputation method	Re-use the dataset provided by Muchlinski et al., which uses an incorrect imputation method	Re-use the dataset provided by Muchlinski et al., which uses an incorrect imputation method	Use several proxy variables for the outcome as predictors (e.g., *colwars*, *cowwars*, *sdwars*, all proxies for civil war), leading to near perfect accuracy

Solution

Transformers & pipelines

Transformers and pipelines are specialized classes for machine learning frameworks that play a crucial role in the data preprocessing and model building process.

scikit learn

Pipeline

Transformers → Estimator

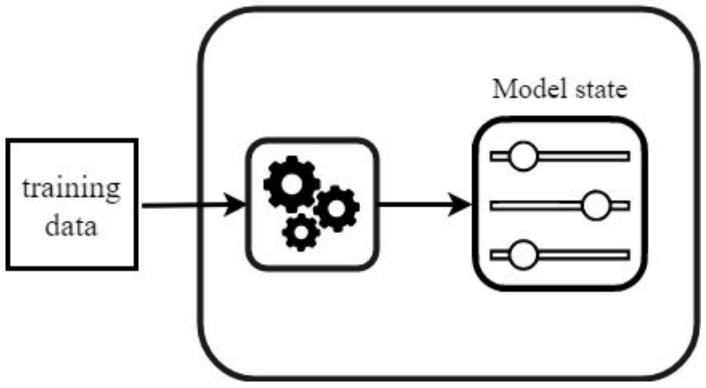
How does scikit-learn work?

- Transformer
- Estimators
- Pipelines & column transformers

Note: not only scikit-learn, many libraries, inspired by the scikit-learn framework, built a similar structure.

Scikit-learn transformer

`transformer.fit(data)`



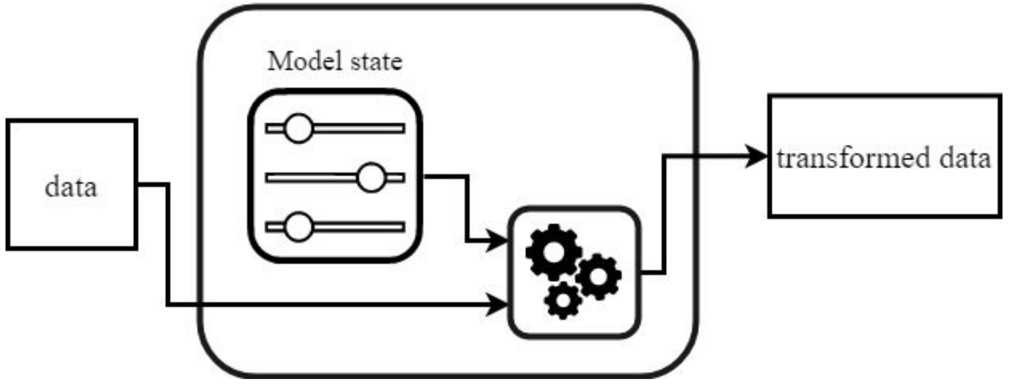

```python
imputer = SimpleImputer(strategy='median')
imputer.fit(X[target_columns])
imputer.statistics_
```

```
array([29.   , 70.35])
```

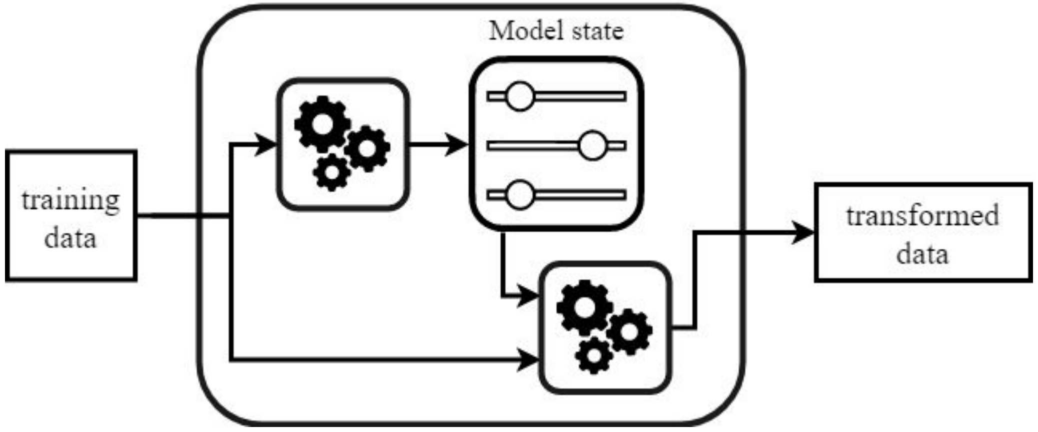
```python
scaler = StandardScaler()
scaler.fit(X[target_columns])
print(scaler.mean_, scaler.scale_)
```

```
[32.37114866 64.76169246] [24.55773742 30.08791085]
```

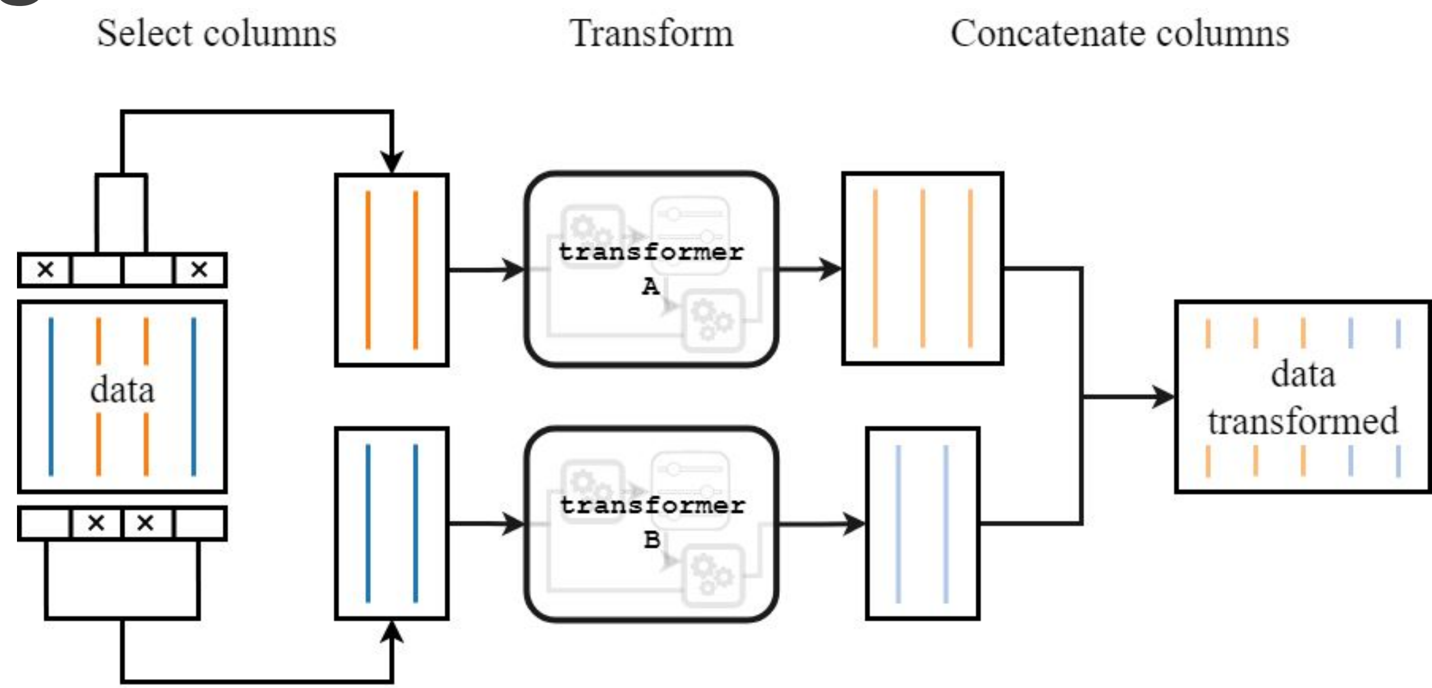
`transformer.transform(data)`



`transformer.fit_transform(data)`



Column transformers



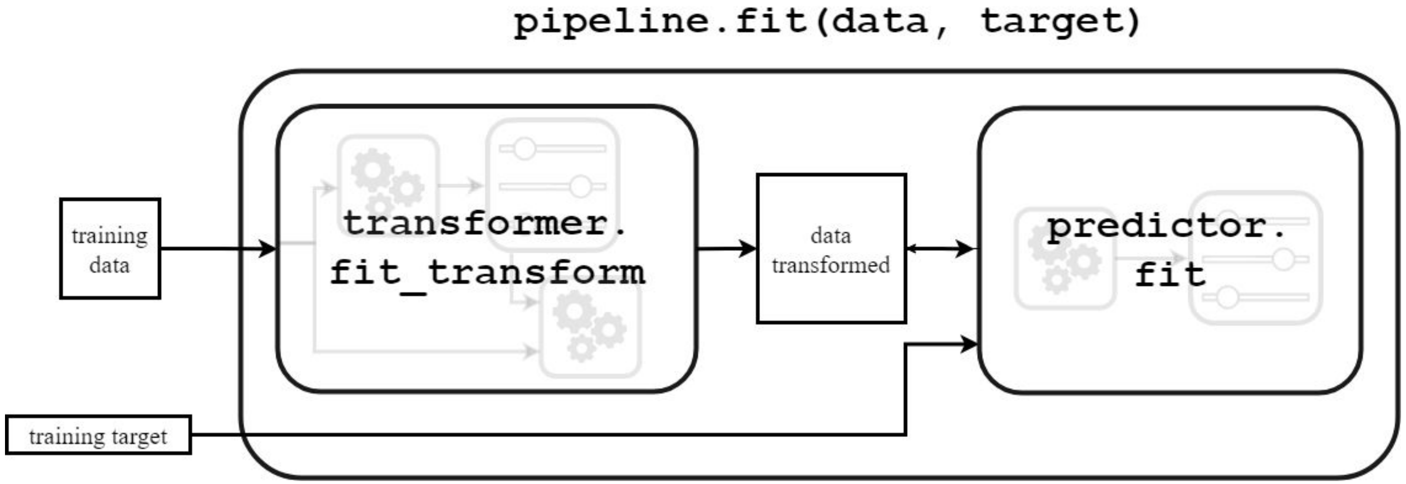
Input features

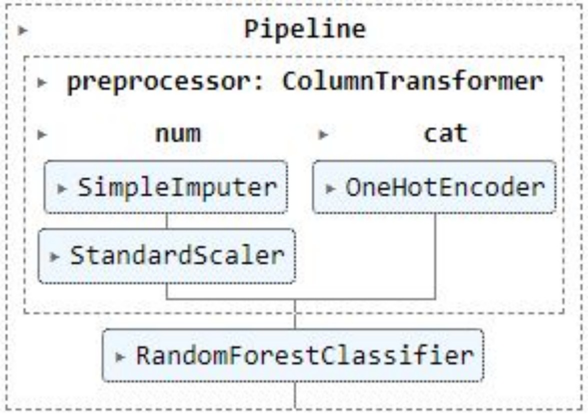
CustomerID	CLTV	Monthly Charges	Contract	Dependents
8695-WDYEA	4483	51.25	Month-to-month	No
3373-YZZYM	3007	19.20	Month-to-month	Yes
8748-HFWBO	5964	19.90	One year	Yes
2200-DSAAL	5914	80.65	Two year	No
0195-IESCP	3573	85.25	Month-to-month	No

Transformed features (output)

CustomerID	num__CLTV	num__Monthly Charges	cat__Contract_Month-to-month	cat__Contract_One year	cat__Contract_Two year	cat__Dependents_Yes
8695-WDYEA	0.062885	-0.443960	1.0	0.0	0.0	0.0
3373-YZZYM	-1.183375	-1.507622	1.0	0.0	0.0	1.0
8748-HFWBO	1.313367	-1.484390	0.0	1.0	0.0	1.0
2200-DSAAL	1.271150	0.531755	0.0	0.0	1.0	0.0
0195-IESCP	-0.705473	0.684418	1.0	0.0	0.0	0.0

Scikit-learn pipelines

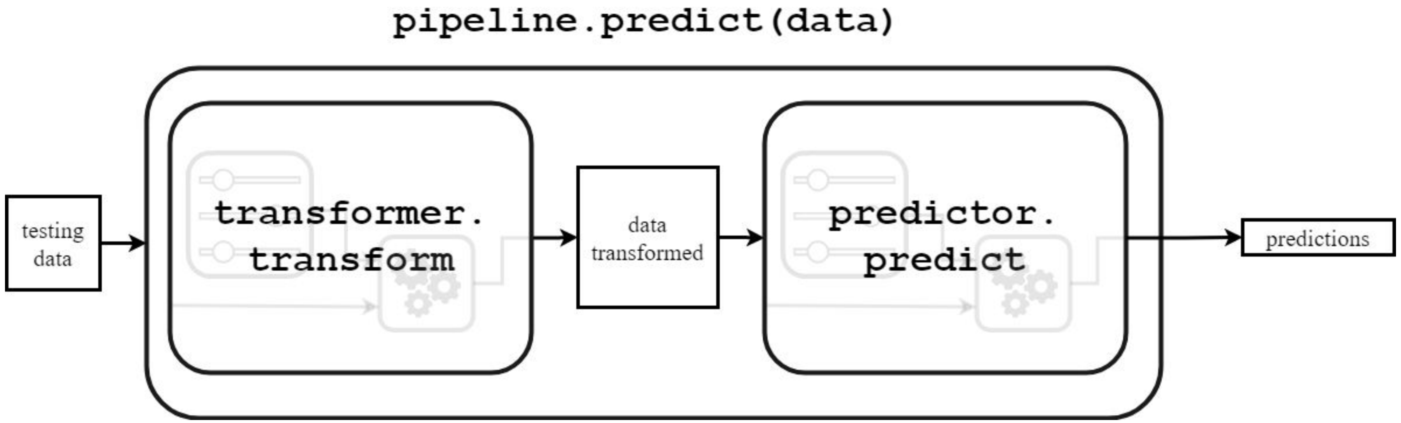




pipeline.fit(data, target):

```
imputer.mean_
scaler.mean_, scaler.scale_
encoder.categories_, encoder.encoder_drop_idx_

classifier.estimators_
```



Warning: Risk of data leak

Do not use `scale` unless you know what you are doing. A common mistake is to apply it to the entire data *before* splitting into training and test sets. This will bias the model evaluation because information would have leaked from the test set to the training set. In general, we recommend using `StandardScaler` within a Pipeline in order to prevent most risks of data leaking: `pipe = make_pipeline(StandardScaler(), LogisticRegression())`.

`sklearn.preprocessing`: Preprocessing and Normalization

The `sklearn.preprocessing` module includes scaling, centering, normalization, binarization methods.

User guide: See the Preprocessing data section for further details.

`preprocessing.Binarizer`(*[, threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold.
`preprocessing.FunctionTransformer`([func, ...])	Constructs a transformer from an arbitrary callable.
`preprocessing.KBinsDiscretizer`([n_bins, ...])	Bin continuous data into intervals.
`preprocessing.KernelCenterer`()	Center an arbitrary kernel matrix K.
`preprocessing.LabelBinarizer`(*[, neg_label, ...])	Binarize labels in a one-vs-all fashion.
`preprocessing.LabelEncoder`()	Encode target labels with value between 0 and n_classes-1.
`preprocessing.MultiLabelBinarizer`(*[, ...])	Transform between iterable of iterables and a multilabel format.
`preprocessing.MaxAbsScaler`(*[, copy])	Scale each feature by its maximum absolute value.
`preprocessing.MinMaxScaler`([feature_range, ...])	Transform features by scaling each feature to a given range.
`preprocessing.Normalizer`([norm, copy])	Normalize samples individually to unit norm.
`preprocessing.OneHotEncoder`(*[, categories, ...])	Encode categorical features as a one-hot numeric array.
`preprocessing.OrdinalEncoder`(*[, ...])	Encode categorical features as an integer array.
`preprocessing.PolynomialFeatures`([degree, ...])	Generate polynomial and interaction features.
`preprocessing.PowerTransformer`([method, ...])	Apply a power transform featurewise to make data more Gaussian-like.
`preprocessing.QuantileTransformer`(*[, ...])	Transform features using quantiles information.
`preprocessing.RobustScaler`(*[, ...])	Scale features using statistics that are robust to outliers.
`preprocessing.SplineTransformer`([n_knots, ...])	Generate univariate B-spline bases for features.
`preprocessing.StandardScaler`(*[, copy, ...])	Standardize features by removing the mean and scaling to unit variance.

Use in pipeline

`preprocessing.add_dummy_feature`(X[, value])	Augment dataset with an additional dummy feature.
`preprocessing.binarize`(X, *[, threshold, copy])	Boolean thresholding of array-like or scipy.sparse matrix.
`preprocessing.label_binarize`(y, *, classes)	Binarize labels in a one-vs-all fashion.
`preprocessing.maxabs_scale`(X, *[, axis, copy])	Scale each feature to the [-1, 1] range without breaking the sparsity.
`preprocessing.minmax_scale`(X[, ...])	Transform features by scaling each feature to a given range.
`preprocessing.normalize`(X[, norm, axis, ...])	Scale input vectors individually to unit norm (vector length).
`preprocessing.quantile_transform`(X, *[, ...])	Transform features using quantiles information.
`preprocessing.robust_scale`(X, *[, axis, ...])	Standardize a dataset along any axis.
`preprocessing.scale`(X, *[, axis, with_mean, ...])	Standardize a dataset along any axis.
`preprocessing.power_transform`(X[, method, ...])	Parametric, monotonic transformation to make data more Gaussian-like.

Do not use **unless you know what you are doing!**

Hands-on!

Section Break

Time to grab a coup of coffee ☕

Feel free to contact me

cmcouto-silva

Thank you!