HALMSTAD UNIVERSITY

# PyCaret
## Introduction to research in embedded and intelligent systems

Anna Vettoruzzo
23/11/2023

# Overview

1. Introduction
2. PyCaret for machine learning
3. Data preparation + Tutorial
4. Model training
5. Hyperparameters tuning + Tutorial
6. Ensemble a model

# 1- Introduction

PYCARET



python™

# 1- Introduction

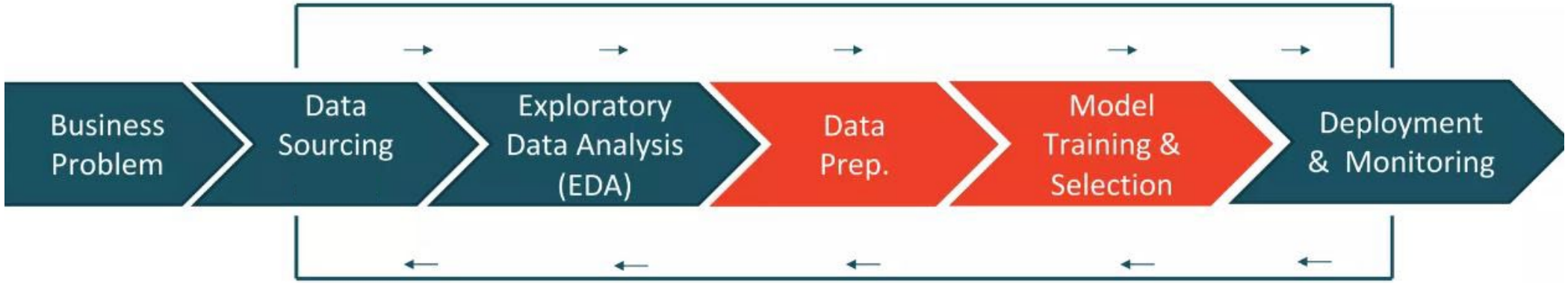
PYCARET

Hypothesis

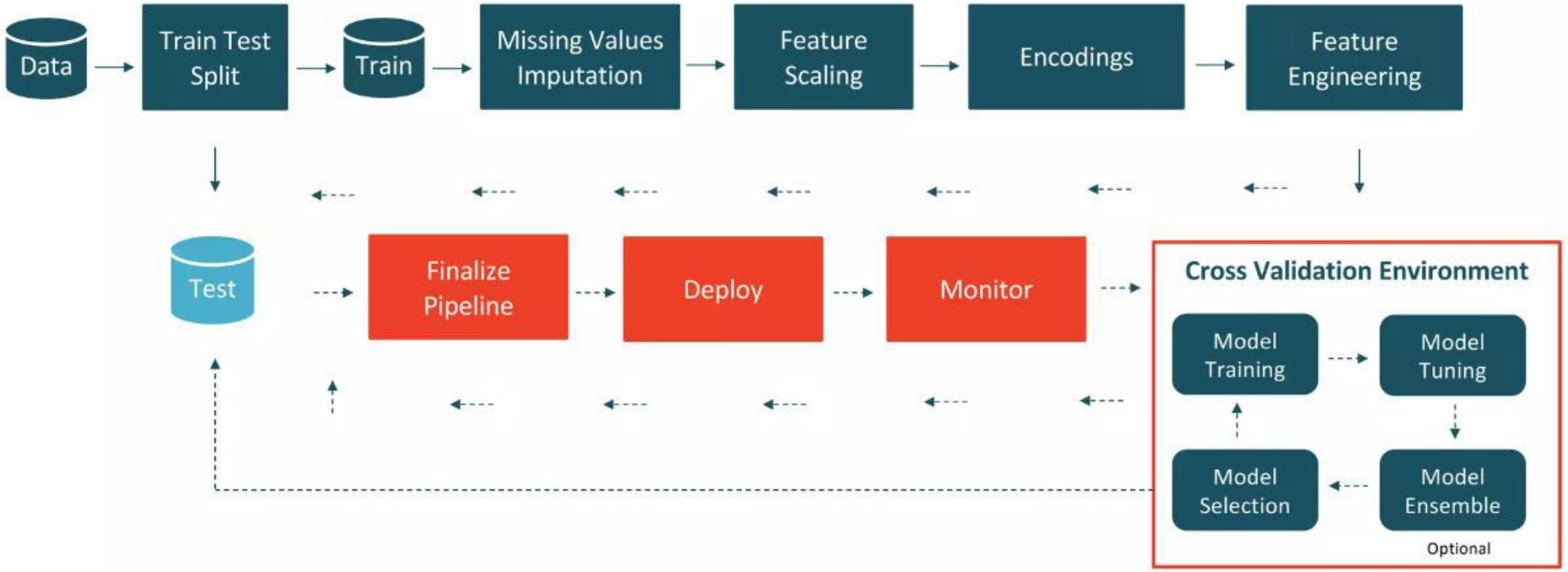
Results

***Goal**: automate the major steps for evaluating and comparing machine learning algorithms (for classification and regression).*

# 2- Machine Learning Life Cycle



# 2- Data Prep, Model Training & Selection



# 2- Features

Data Preparation

Model Training

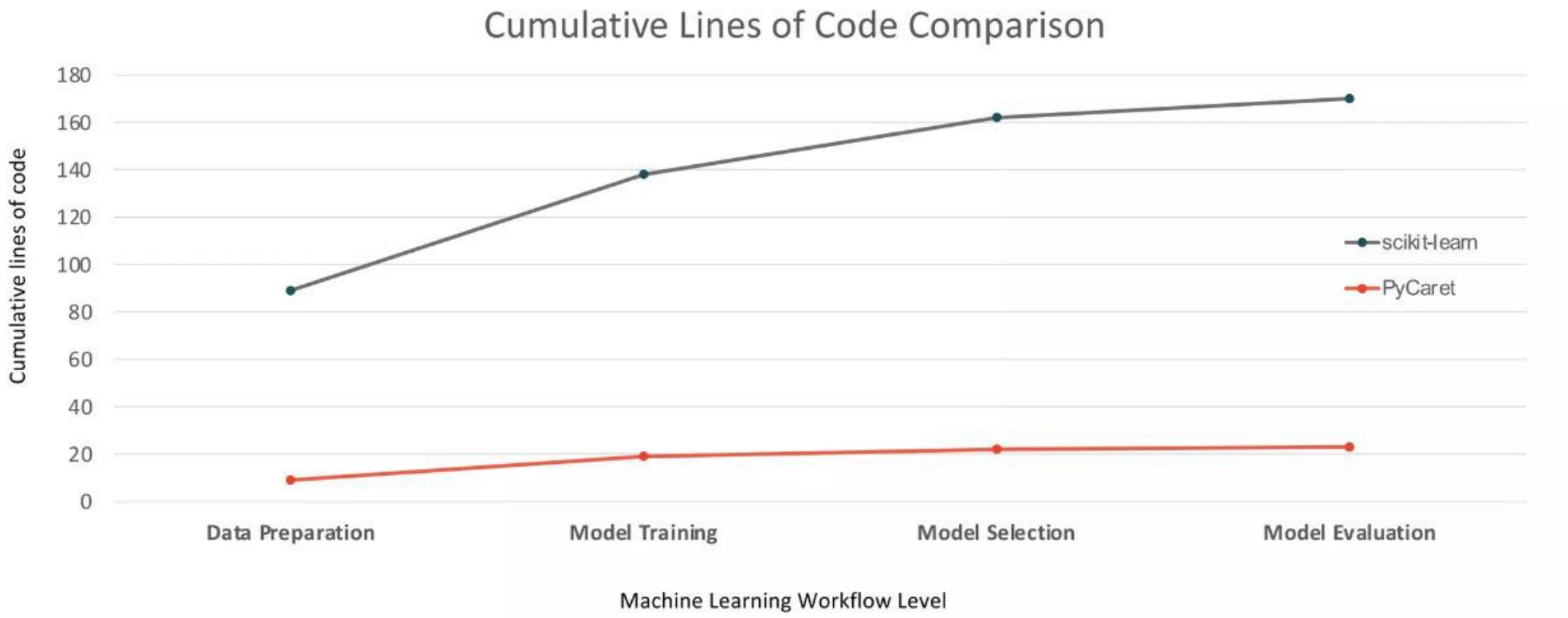
Hyperparameter Tuning

Analysis & Interpretability

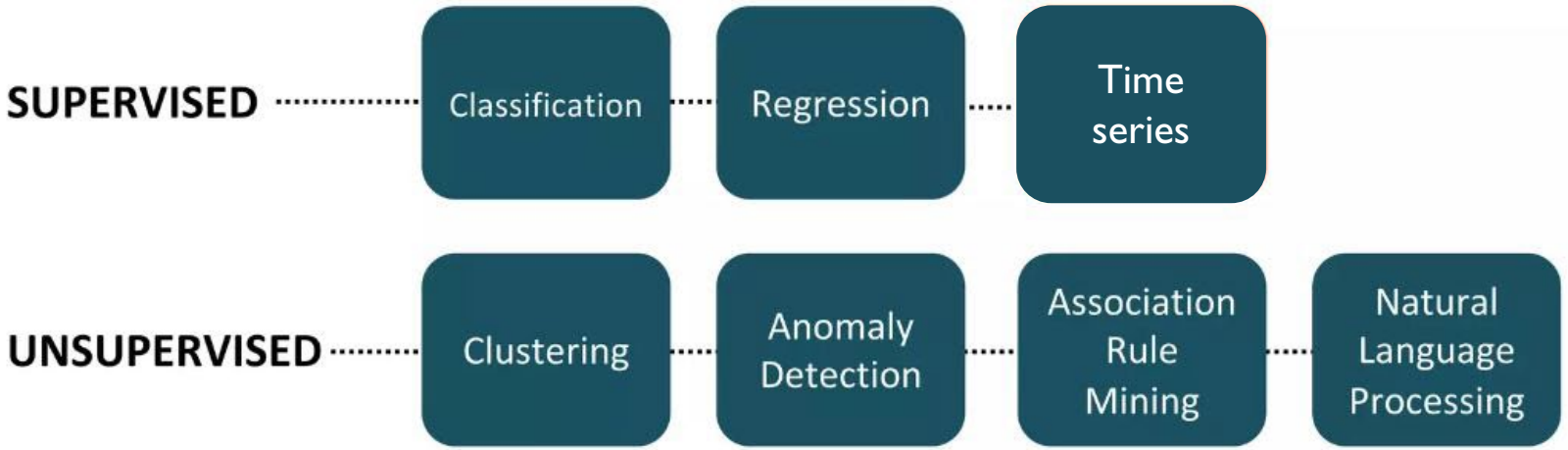
Model Selection

Experiment Logging

# 2- Impact of PyCaret (measured on # lines of code)



# 2- Machine Learning use-case supported

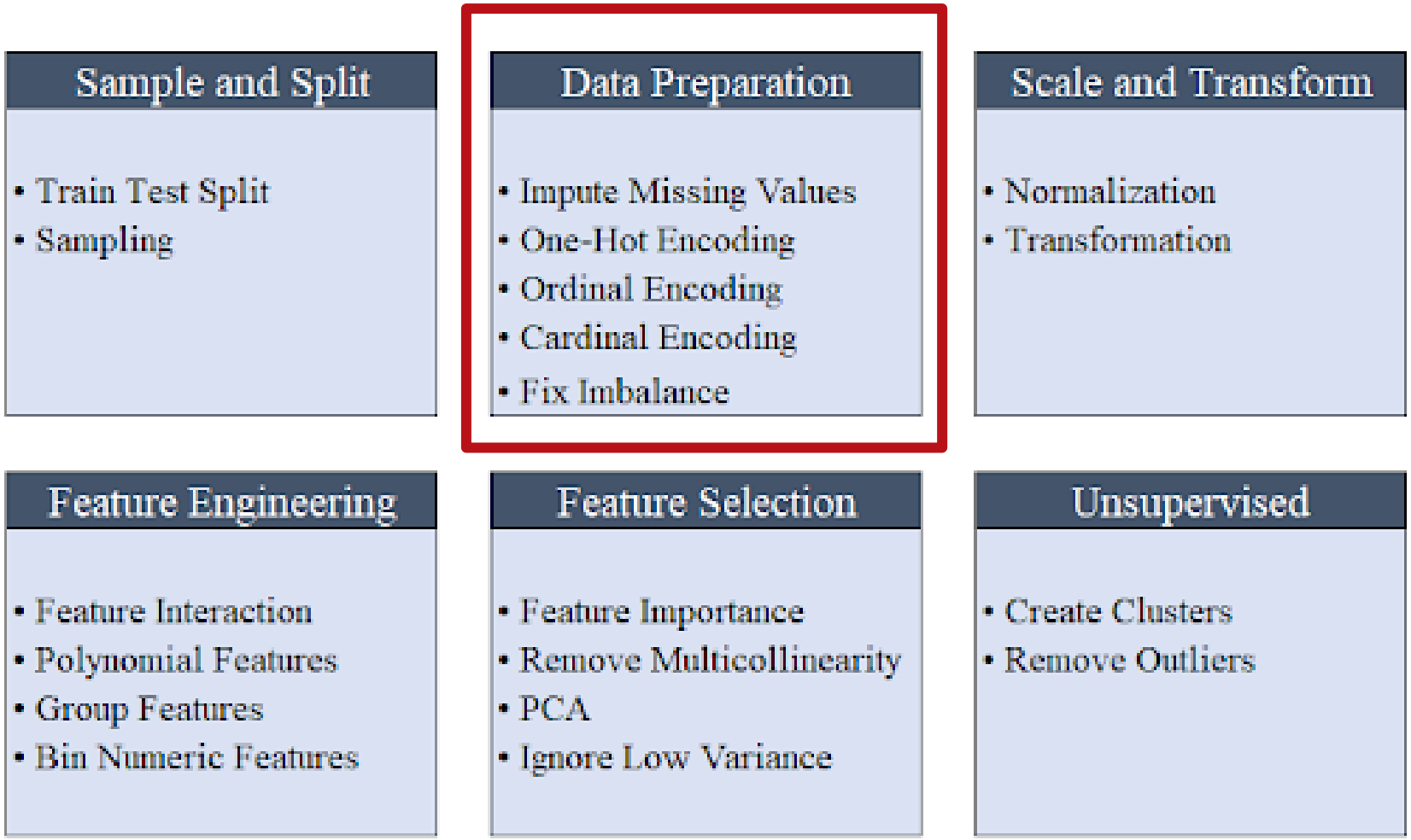


# 3- ML pipeline – Data loading

## Let's move to


https://colab.research.google.com/drive/1uQzh1HKrHJOprKCJeHovuSoOrMcWnIGK?usp=sharing

# 3- Data preprocessing




One function for everything: *pycaret.classification.setup(....)*

# 3- Data preparation

- Missing values (often encoded as blanks or NaN)
  - → we can remove the samples with missing values
  - → we can impute the missing values

By default:

*imputation_type: Optional[str] = 'simple'*

# 3- Data preparation

- Missing values (often encoded as blanks or NaN)

- Data types of the features
  - → PyCaret automatically detect the data type
  - → We can manually set the datatype of specific features

By default:

*imputation_type: Optional[str] = 'simple'*

*ordinal_features: dict, default = None*
*numeric_features: list of str, default = None*
*categorical_features: list of str, default = None*

# 3- Data preparation

- Missing values (often encoded as blanks or NaN)
- Data types of the features
- One-hot Encoding / Ordinal Encoding

By default:

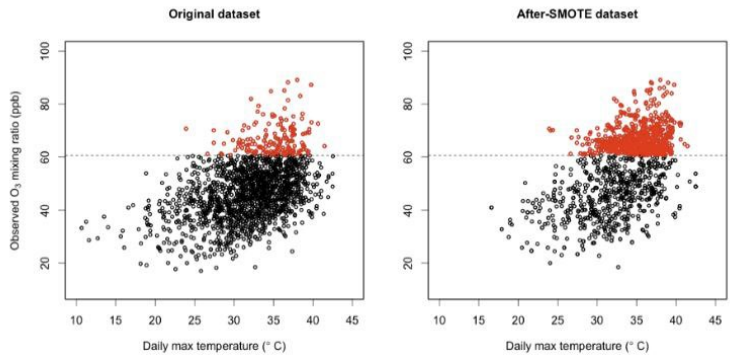
*imputation_type: Optional[str] = 'simple'*

*ordinal_features: dict, default = None*
*numeric_features: list of str, default = None*
*categorical_features: list of str, default = None*

*ordinal_features: Optional[Dict[str, list]] = None*

# 3- Data preparation

- Missing values (often encoded as blanks or NaN)
- Data types of the features
- One-hot Encoding / Ordinal Encoding
- Target imbalance




By default:

*imputation_type: Optional[str] = 'simple'*

*ordinal_features: dict, default = None*
*numeric_features: list of str, default = None*
*categorical_features: list of str, default = None*

*ordinal_features: Optional[Dict[str, list]] = None*

*fix_imbalance: bool, default = False*
fix_imbalance_method: str, default = "SMOTE"



HALMSTAD UNIVERSITY

# 3- Data preparation

- Missing values (often encoded as blanks or NaN)
- Data types of the features
- One-hot Encoding / Ordinal Encoding
- Target imbalance
- Remove outliers

By default:

*imputation_type: Optional[str] = 'simple'*

*ordinal_features: dict, default = None*
*numeric_features: list of str, default = None*
*categorical_features: list of str, default = None*

*ordinal_features: Optional[Dict[str, list]] = None*

*fix_imbalance: bool, default = False*
fix_imbalance_method: str, default = “SMOTE”

*remove_outliers: bool, default = False*
*outliers_method: str, default = “iforest”*
*outliers_threshold: float, default = 0.05*

# 3- Data preparation

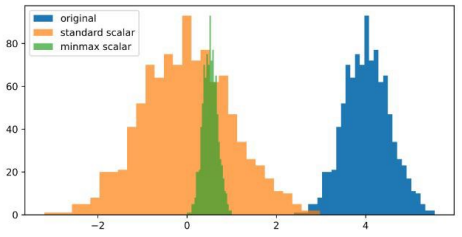
- Normalize
  - → Use to normalize the feature space
  - → Possible methods: *zscore, minmax, maxabs, robust*

By default:

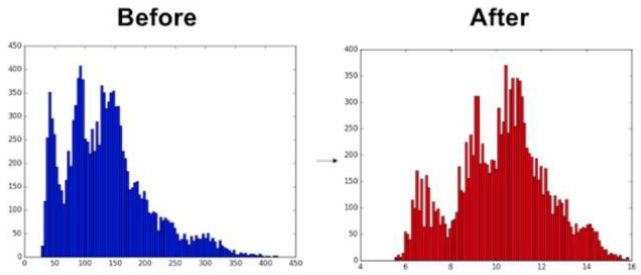
*normalize: bool = False*
*normalize_method: str = 'zscore'*

# 3- Data preparation

- Normalize




- Feature transform
  - → It changes the shape of the distribution




By default:

*normalize: bool = False*
*normalize_method: str = 'zscore'*

*transformation: bool = False,*
*transformation_method: str = 'yeo-johnson'*

# 3- Data preparation

- Normalize
- Feature transform
- Target transform
  - → It changes the shape of the target distribution
  - → Only with regression

By default:

*normalize: bool = False*
*normalize_method: str = 'zscore'*

*transformation: bool = False,*
*transformation_method: str = 'yeo-johnson'*

*Transform_target: bool = False,*
*Transform_target_method: str = 'yeo-johnson'*

# 3- Data preparation

- **Feature engineering:**
  usually we assume a linear relationship between the dependent and independent variables. When this is not the case we can create new polynomial features.

- **Group features:**
  Features that are related to each other. We can create them from existing features.

- **Bin numeric feature:**
  To turn continuous variables into categorical values using bins. It is effective when a continuous feature has too many unique values or few extreme values outside the expected range.

By default:

*polynomial_features: bool = False,*
*polynomial_degree: int = 2*

*group_features: Optional[dict] = None,*

*bin_numeric_features: Optional[List[str]] = None*

# 3- Data preparation

- Feature selection:
  → possible methods:
  *classic, univariate, sequential*
- Remove multicollinearity
- PCA:
  unsupervised technique that reduce the data dimensionality
- Ignore low variance

By default:

*feature_selection: bool = False,*
*feature_selection_method: str = 'classic',*
*feature_selection_estimator:*
*Union[str, Any] = 'lightgbm',*
*n_features_to_select: Union[int, float] = 0.2*

*remove_multicollinearity: bool = False,*
*multicollinearity_threshold: float = 0.9*

*pca: bool = False,*
*pca_method: str = 'linear',*
*pca_components:*
*Optional[Union[int, float, str]] = None*

*low_variance_threshold: Optional[float] = None*

# 3- Data preparation

- Experiments logging
  → the default use Mlflow but you can extend it to *wandb, cometml, dagshub*

- Parameters that influence model selection:
  - train_test split
  - fold_strategy: possibilities are *kfold, stratifiedkfold, groupkfold, timeseries, …*

- GPU usage

- Others…

By default:

*log_experiment: Union[bool, str, BaseLogger, List[Union[str, BaseLogger]]] = False*
*log_plots: Union[bool, list] = False,*
*log_profile: bool = False,*
*log_data: bool = False*

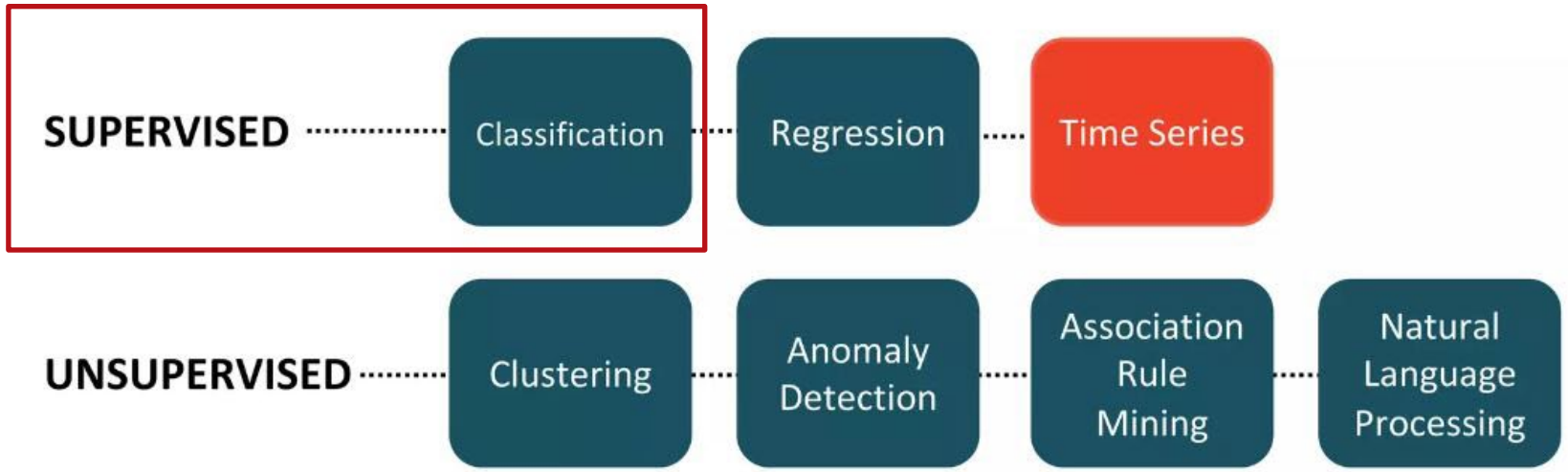
*train_size: float, default = 0.7*
*test_data: dataframe-like or None, default = None*
*data_split_shuffle: bool, default = True*
*data_split_stratify: bool or list, default = True*
*fold_strategy: str or scikit-learn CV generator object, default = 'stratifiedkfold'*
*fold: int, default = 10*
*fold_shuffle: bool, default = False*
*fold_groups: str or array-like, with shape (n_samples,), default = None*

*use_gpu = bool, default=False*

# 3- Summary

```
setup(data: Optional[Union[dict, list, tuple, ndarray, spmatrix, DataFrame]] = None, data_func:
Optional[Callable[[], Union[dict, list, tuple, ndarray, spmatrix, DataFrame]]] = None, target: Union[int, str, list,
tuple, ndarray, Series] = -1, index: Union[bool, int, str, list, tuple, ndarray, Series] = True, train_size: float =
0.7, test_data: Optional[Union[dict, list, tuple, ndarray, spmatrix, DataFrame]] = None, ordinal_features:
Optional[Dict[str, list]] = None, numeric_features: Optional[List[str]] = None, categorical_features:
Optional[List[str]] = None, date_features: Optional[List[str]] = None, text_features: Optional[List[str]] =
None, ignore_features: Optional[List[str]] = None, keep_features: Optional[List[str]] = None, preprocess: bool
= True, create_date_columns: List[str] = ['day', 'month', 'year'], imputation_type: Optional[str] = 'simple',
numeric_imputation: str = 'mean', categorical_imputation: str = 'mode', iterative_imputation_iters: int = 5,
numeric_iterative_imputer: Union[str, Any] = 'lightgbm', categorical_iterative_imputer: Union[str, Any] =
'lightgbm', text_features_method: str = 'tf-idf', max_encoding_ohe: int = 25, encoding_method:
Optional[Any] = None, rare_to_value: Optional[float] = None, rare_value: str = 'rare', polynomial_features:
bool = False, polynomial_degree: int = 2, low_variance_threshold: Optional[float] = None, group_features:
Optional[dict] = None, drop_groups: bool = False, remove_multicollinearity: bool = False,
multicollinearity_threshold: float = 0.9, bin_numeric_features: Optional[List[str]] = None, remove_outliers:
bool = False, outliers_method: str = 'iforest', outliers_threshold: float = 0.05, fix_imbalance: bool = False,
fix_imbalance_method: Union[str, Any] = 'SMOTE', transformation: bool = False, transformation_method: str
= 'yeo-johnson', normalize: bool = False, normalize_method: str = 'zscore', pca: bool = False, pca_method: str
= 'linear', pca_components: Optional[Union[int, float, str]] = None, feature_selection: bool = False,
feature_selection_method: str = 'classic', feature_selection_estimator: Union[str, Any] = 'lightgbm',
n_features_to_select: Union[int, float] = 0.2, custom_pipeline: Optional[Any] = None,
custom_pipeline_position: int = -1, data_split_shuffle: bool = True, data_split_stratify: Union[bool, List[str]] =
True, fold_strategy: Union[str, Any] = 'stratifiedkfold', fold: int = 10, fold_shuffle: bool = False, fold_groups:
Optional[Union[str, DataFrame]] = None, n_jobs: Optional[int] = -1, use_gpu: bool = False, html: bool = True,
session_id: Optional[int] = None, system_log: Union[bool, str, Logger] = True, log_experiment: Union[bool, str,
BaseLogger, List[Union[str, BaseLogger]]] = False, experiment_name: Optional[str] = None,
experiment_custom_tags: Optional[Dict[str, Any]] = None, log_plots: Union[bool, list] = False, log_profile:
bool = False, log_data: bool = False, engine: Optional[Dict[str, str]] = None, verbose: bool = True, memory:
Union[bool, str, Memory] = True, profile: bool = False, profile_kwargs: Optional[Dict[str, Any]] = None)
```

# 4- Classification experiments



# 4- Creating models

- **create_model()** will create a model and train it on the dataset that has been defined with the setup function.

  Models can be chosen from the models available in the library or pass an untrained model object consistent with scikit-learn API.
  Possible choices are:

- 'lr' - Logistic Regression
- 'knn' - K Neighbors Classifier
- 'nb' - Naive Bayes
- 'dt' - Decision Tree Classifier
- 'svm' - SVM - Linear Kernel
- 'rbfsvm' - SVM - Radial Kernel
- 'gpc' - Gaussian Process Classifier
- 'mlp' - MLP Classifier
- 'ridge' - Ridge Classifier
- 'rf' - Random Forest Classifier
- 'qda' - Quadratic Discriminant Analysis
- 'ada' - Ada Boost Classifier
- 'gbc' - Gradient Boosting Classifier
- 'lda' - Linear Discriminant Analysis
- 'et' - Extra Trees Classifier
- 'xgboost' - Extreme Gradient Boosting
- 'lightgbm' - Light Gradient Boosting Machine
- 'catboost' - CatBoost Classifier

# 4- Creating models

- **compare_models()** will compare all the estimators that are present in the library.

```
best = compare_models()
```

| | Model | Accuracy | AUC | Recall | Prec. | F1 | Kappa | MCC | TT (Sec) |
|---|---|---|---|---|---|---|---|---|---|
| **ridge** | Ridge Classifier | 0.8626 | 0.0000 | 0.7986 | 0.8964 | 0.8275 | 0.7165 | 0.7359 | 0.0310 |
| **lda** | Linear Discriminant Analysis | 0.8626 | 0.8948 | 0.7986 | 0.8964 | 0.8275 | 0.7165 | 0.7359 | 0.0520 |
| **lr** | Logistic Regression | 0.8520 | 0.8945 | 0.7986 | 0.8756 | 0.8181 | 0.6959 | 0.7145 | 1.0430 |
| **rf** | Random Forest Classifier | 0.8468 | 0.8942 | 0.7639 | 0.8931 | 0.8088 | 0.6849 | 0.7038 | 0.2030 |
| **et** | Extra Trees Classifier | 0.8415 | 0.9071 | 0.7764 | 0.8674 | 0.8093 | 0.6756 | 0.6898 | 0.4110 |
| **catboost** | CatBoost Classifier | 0.8415 | 0.8985 | 0.7861 | 0.8598 | 0.8125 | 0.6761 | 0.6887 | 1.2390 |
| **qda** | Quadratic Discriminant Analysis | 0.8257 | 0.8671 | 0.7875 | 0.8261 | 0.7920 | 0.6433 | 0.6590 | 0.0310 |
| **nb** | Naive Bayes | 0.8152 | 0.8745 | 0.7778 | 0.8224 | 0.7782 | 0.6245 | 0.6427 | 0.0320 |
| **xgboost** | Extreme Gradient Boosting | 0.7994 | 0.8745 | 0.7389 | 0.7943 | 0.7583 | 0.5886 | 0.5972 | 0.1270 |
| **lightgbm** | Light Gradient Boosting Machine | 0.7939 | 0.8711 | 0.7153 | 0.8067 | 0.7491 | 0.5763 | 0.5890 | 0.2230 |
| **gbc** | Gradient Boosting Classifier | 0.7781 | 0.8566 | 0.7139 | 0.7715 | 0.7333 | 0.5453 | 0.5551 | 0.1890 |
| **ada** | Ada Boost Classifier | 0.7573 | 0.8386 | 0.7056 | 0.7587 | 0.7151 | 0.5068 | 0.5259 | 0.1510 |
| **dt** | Decision Tree Classifier | 0.7202 | 0.7207 | 0.7278 | 0.6775 | 0.6903 | 0.4369 | 0.4523 | 0.0300 |
| **knn** | K Neighbors Classifier | 0.6775 | 0.7395 | 0.6111 | 0.6237 | 0.6062 | 0.3434 | 0.3462 | 0.0450 |
| **svm** | SVM - Linear Kernel | 0.6614 | 0.0000 | 0.6278 | 0.6568 | 0.5823 | 0.3200 | 0.3657 | 0.0300 |
| **dummy** | Dummy Classifier | 0.5556 | 0.5000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0470 |

# 4- Comparing models

- **compare_models()** will compare all the estimators that are present in the library.

- **get_metrics(), add_metric(), remove_metric()** can be used to access, add or remove specific metrics during the comparison.

Let's move to


https://colab.research.google.com/drive/1uQzh1HKrHJOprKCJeHovuSoOrMcWnIGK?usp=sharing

# 4- Analyze the model

Different function can be used to analyze the performance of a trained model on the test set.

- **plot_model(**model, plot=*'confusion_matrix'***)** return the confusion matrix (for classification)
- **plot_model(**model, plot=*metric_name***)**
- **plot_model(**model, plot='feature'**)** plot the feature importance
- **evaluate_model**(model) displays a user interface for analyzing performance
- **interpret_model**(model) returns an interpretation plot based on the test / hold-out set using SHAP (Shapley Additive exPlanations).

# 5- Tune the model

Let's move to


https://colab.research.google.com/drive/1uQzh1HKrHJOprKCJeHovuSoOrMcWnIGK?usp=sharing

# Saving the model

```
save_model(tuned_best,'Final RC Model')
```

```
Transformation Pipeline and Model Successfully Saved
(Pipeline(memory=Memory(location=None),
          steps=[('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['age', 'sex', 'chest pain type',
                                              'resting blood pressure',
                                              'serum cholestoral in mg/dl      ',
                                              'fasting blood sugar > 120 mg/dl ',
                                              'resting electrocardiographic '
                                              'results',
                                              'maximum heart rate achieved  ',
                                              'exercise induced angina    ',
                                              'oldpeak ', 'slope of peak',
                                              'number of major vessels ',...
                                                          strategy='most_frequent',
                                                          verbose='deprecated'))),
                 ('clean_column_names',
                  TransformerWrapper(exclude=None, include=None,
                                     transformer=CleanColumnNames(match='[\\]\\[\\,\\{\\}\\"\\:]+'))),
                 ('trained_model',
                  RidgeClassifier(alpha=5.62, class_weight=None, copy_X=True,
                                  fit_intercept=True, max_iter=None,
                                  positive=False, random_state=123,
                                  solver='auto', tol=0.0001))],
          verbose=False),
 'Final RC Model.pkl')
```

```
saved_final_rc = load_model('Final RC Model')
```

```
Transformation Pipeline and Model Successfully Loaded
```

# Advanced topics – Ensemble a model

Ensembling is a common technique to improve the performance of models.
Ensemble methods include:

- *Bagging*: learns each model in parallel and combine them with an averaging process.
- *Boosting*: learns models sequentially in an adaptive way and combines them with a deterministic strategy.
- *Stacking*: learns heterogeneous models in parallel and combines them by training a meta-model to output prediction.

In PyCaret (for bagging and boosting):

```
ensemble_model(model,
               method,
               fold,
               n_estimators,
               etc…
              )
```

In PyCaret (for stacking):

```
stack_models([models_name],
             method,
             fold,
             n_estimators,
             etc…
             )
```

# Regression experiment

## Now it's your turn!

Open Colab and setup a regression experiment with PyCaret:

1. Choose a dataset for regression.
2. Preprocess the dataset (remember you can automatically do it with *setup(…)* ).
3. Train the models and compare them (use *compare_models(…)* ).
4. Tune the best model.
5. Deploy it on the test data and compare the performance before and after tuning.
6. Save the model.

# References


- M. Ali, **PyCaret: An open source, low-code machine learning library in Python.**
  URL: https://www.pycaret.org

- M. Ali, **Machine Learning Made Easy With PyCaret.**
  URL: https://www.youtube.com/watch?v=k4zNjlHacr4

- J. Rocca, **Ensemble methods: bagging, boosting and stacking.**
  URL: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205