MachineLearnAthon - Microlecture
Low code I – Model creation

MachineLearnAthon
A project Co-funded by the Erasmus+ programme of the European Union

Low-Code approach

- Combines drag-and-drop tools with coding options.
- Allows for advanced customization.
- Ideal for semi-technical users and developers.

Key features:

- Visual tools and drag-and-drop interfaces.
- Pre-built templates for rapid development.
- Minimal technical expertise required.

Use Cases:

- Suitable for complex workflows requiring customization.
- Common in enterprise application development.
- Used for both internal and external software solutions.

Introduction to PyCaret

- PyCaret is an open-source, low-code machine learning library in Python.
- Simplifies the end-to-end machine learning lifecycle.
- Designed for both beginners and experienced users.

Key Features of PyCaret

- Automates repetitive ML tasks.
- Supports classification, regression, clustering, and NLP.
- Integrates seamlessly with Jupyter Notebook and IDEs.

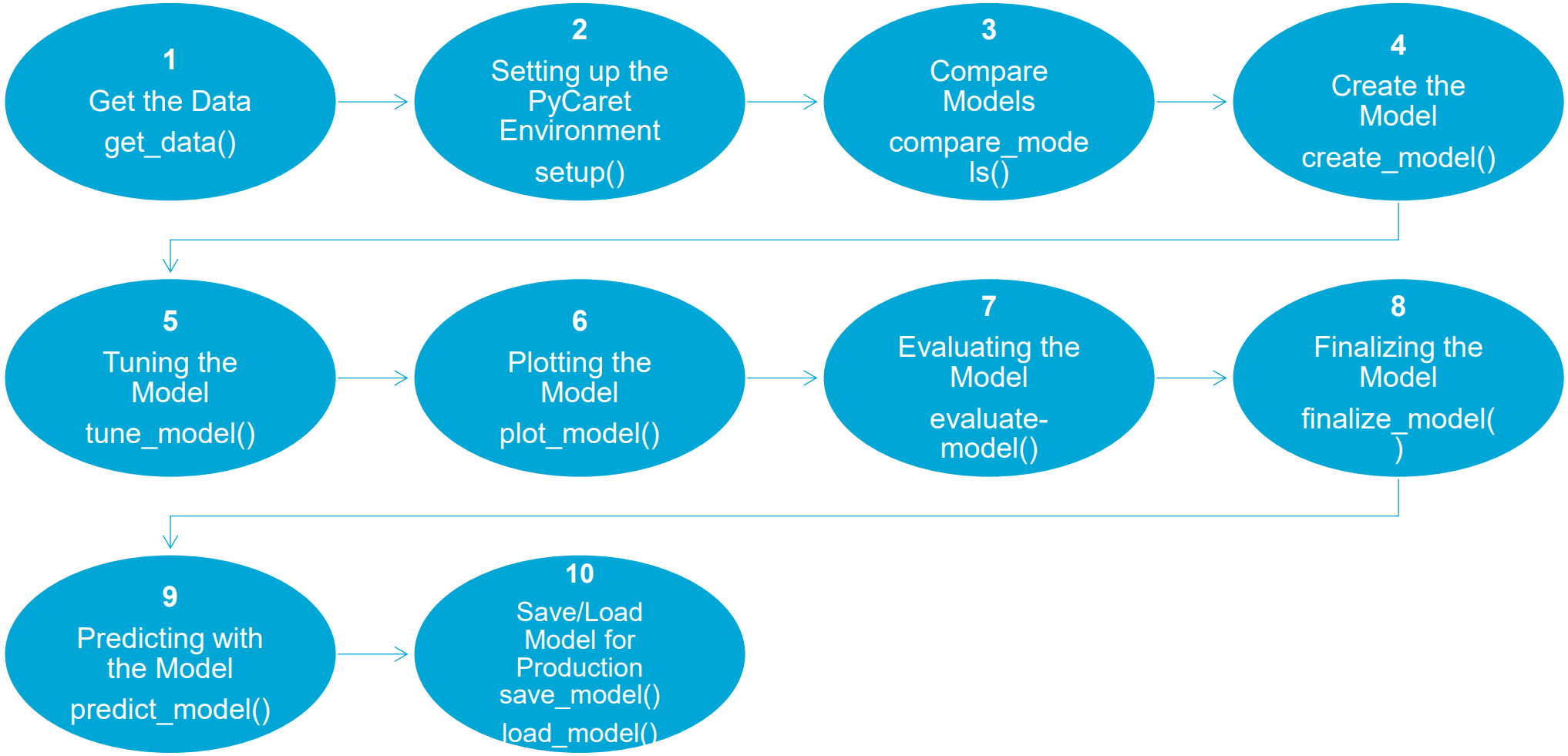
Applications of PyCaret

- Used in rapid prototyping and experimentation.
- Ideal for data exploration and quick model building.
- Supports deployment and monitoring of ML models.

Some of PyCaret modules

Module	Purpose	Example Use Case
`pycaret.classification`	Supervised classification tasks	Predicting customer churn
`pycaret.regression`	Supervised regression tasks	Forecasting sales revenue
`pycaret.clustering`	Unsupervised clustering tasks	Customer segmentation
`pycaret.anomaly`	Anomaly detection tasks	Fraud detection
`pycaret.nlp`	Natural Language Processing (NLP)	Topic modeling or text classification
`pycaret.time_series`	Time series analysis and forecasting	Stock price predictions
`pycaret.association`	Association rule mining	Market basket analysis
`pycaret.deep_learning`	Experimental: Deep learning integration	Utilizing TensorFlow or PyTorch for modeling
`pycaret.image`	Experimental: Image data analysis	Image classification
`pycaret.eda`	Experimental: Exploratory Data Analysis (EDA)	Visualizing data distributions and correlations

Machine Learning Pipeline with PyCaret



Example of Clasification Models

Classification Module in PyCaretDesigned for supervised machine learning tasks like binary and multi-class classification.

Includes

- 18+ algorithms and
- 14 performance plots
- stacking and
- ensembling

```
models()
```

[10] ✓ 0.1s

ID	Name	Reference	Turbo
lr	Logistic Regression	sklearn.linear_model._logistic.LogisticRegression	True
knn	K Neighbors Classifier	sklearn.neighbors._classification.KNeighborsCl...	True
nb	Naive Bayes	sklearn.naive_bayes.GaussianNB	True
dt	Decision Tree Classifier	sklearn.tree._classes.DecisionTreeClassifier	True
svm	SVM - Linear Kernel	sklearn.linear_model._stochastic_gradient.SGDC...	True
rbfsvm	SVM - Radial Kernel	sklearn.svm._classes.SVC	False
gpc	Gaussian Process Classifier	sklearn.gaussian_process._gpc.GaussianProcessC...	False
mlp	MLP Classifier	sklearn.neural_network._multilayer_perceptron....	False
ridge	Ridge Classifier	sklearn.linear_model._ridge.RidgeClassifier	True
rf	Random Forest Classifier	sklearn.ensemble._forest.RandomForestClassifier	True
qda	Quadratic Discriminant Analysis	sklearn.discriminant_analysis.QuadraticDiscrim...	True
ada	Ada Boost Classifier	sklearn.ensemble._weight_boosting.AdaBoostClas...	True
gbc	Gradient Boosting Classifier	sklearn.ensemble._gb.GradientBoostingClassifier	True
lda	Linear Discriminant Analysis	sklearn.discriminant_analysis.LinearDiscrimina...	True
et	Extra Trees Classifier	sklearn.ensemble._forest.ExtraTreesClassifier	True
lightgbm	Light Gradient Boosting Machine	lightgbm.sklearn.LGBMClassifier	True
dummy	Dummy Classifier	sklearn.dummy.DummyClassifier	True

technische universität dortmund
TU Delft
Co-funded by the Erasmus+ programme of the European Union
Machine LearnAthon

Step 1 Get the Data

Example: Default of Credit Card Clients (Yeh, I.-C. (2016))

```
import pandas as pd
df = pd.read_excel("C:/Users/mirka/Desktop/default of credit card clients.xls")
df.head()
```

[1] ✓ 3.6s Python

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X15	X16	X17	X18	X19	X20	X21	X22	X23	Y
0	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default payment next month
1	1	20000	2	2	1	24	2	2	-1	-1	...	0	0	0	0	689	0	0	0	0	1
2	2	120000	2	2	2	26	-1	2	0	0	...	3272	3455	3261	0	1000	1000	1000	0	2000	1
3	3	90000	2	2	2	34	0	0	0	0	...	14331	14948	15549	1518	1500	1000	1000	1000	5000	0
4	4	50000	2	2	1	37	0	0	0	0	...	28314	28959	29547	2000	2019	1200	1100	1069	1000	0

5 rows × 25 columns

```
from pycaret.datasets import get_data
dataset = get_data('credit')
```

[2] ✓ 5.8s Python

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default
0	20000	2	2	1	24	2	2	-1	-1	-2	...	0.0	0.0	0.0	0.0	689.0	0.0	0.0	0.0	0.0	1
1	90000	2	2	2	34	0	0	0	0	0	...	14331.0	14948.0	15549.0	1518.0	1500.0	1000.0	1000.0	1000.0	5000.0	0
2	50000	2	2	1	37	0	0	0	0	0	...	28314.0	28959.0	29547.0	2000.0	2019.0	1200.0	1100.0	1069.0	1000.0	0
3	50000	1	2	1	57	-1	0	-1	0	0	...	20940.0	19146.0	19131.0	2000.0	36681.0	10000.0	9000.0	689.0	679.0	0
4	50000	1	1	2	37	0	0	0	0	0	...	19394.0	19619.0	20024.0	2500.0	1815.0	657.0	1000.0	1000.0	800.0	0

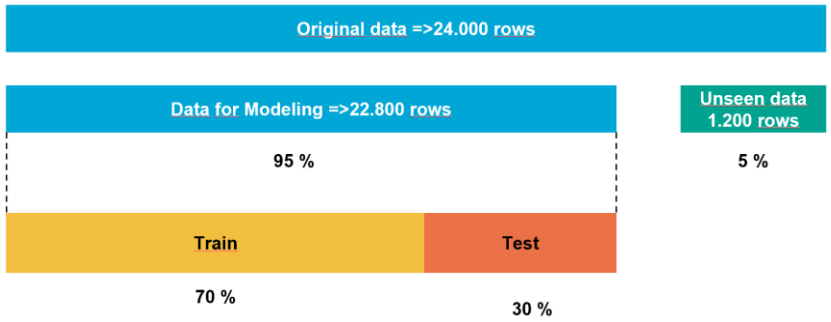
5 rows × 24 columns

```
#check the shape of data
dataset.shape
```

[3] ✓ 0.0s Python

```
(24000, 24)
```

Creating Random Sample



- Dividing the dataset ensures unbiased evaluation.
- Data is split into:
  - **Training Set**: For model learning.
  - **Test Set**: For tuning hyperparameters and frequent evaluation.
  - **Validation Set (Unseen Data)**: For final evaluation simulating real-world data.

```
## sample returns a random sample from an axis of the object. That would be 22,800 samples,
data = dataset.sample(frac=0.95, random_state=786)
data
```

[4] ✓ 0.0s Python

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default
20534	270000	2	1	2	34	0	0	2	0	0	...	44908.0	19508.0	15860.0	4025.0	5.0	34000.0	0.0	0.0	0.0	0
6885	160000	2	1	2	42	-2	-2	-2	-2	-2	...	0.0	741.0	0.0	0.0	0.0	0.0	741.0	0.0	0.0	0
1553	360000	2	1	2	30	0	0	0	0	0	...	146117.0	145884.0	147645.0	6000.0	6000.0	4818.0	5000.0	5000.0	4500.0	0
1952	20000	2	1	2	25	0	0	0	0	0	...	18964.0	19676.0	20116.0	1700.0	1300.0	662.0	1000.0	747.0	602.0	0
21422	70000	1	2	2	29	0	0	0	0	0	...	48538.0	49034.0	49689.0	2200.0	8808.0	2200.0	2000.0	2000.0	2300.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4516	130000	1	3	2	45	0	0	-1	0	-1	...	1261.0	390.0	390.0	1000.0	2522.0	0.0	390.0	390.0	390.0	0
8641	290000	2	1	2	29	0	0	0	0	-1	...	-77.0	8123.0	210989.0	1690.0	3000.0	0.0	8200.0	205000.0	6000.0	0
6206	210000	1	2	1	41	1	2	0	0	0	...	69670.0	59502.0	119494.0	0.0	5000.0	3600.0	2000.0	2000.0	5000.0	0
2110	550000	1	2	1	47	0	0	0	0	0	...	30000.0	0.0	0.0	10000.0	20000.0	5000.0	0.0	0.0	0.0	0
4042	200000	1	1	2	28	0	0	0	0	0	...	161221.0	162438.0	157415.0	7000.0	8016.0	5000.0	12000.0	6000.0	7000.0	0

22800 rows × 24 columns

Removing unseen data

```
# we remove from the original dataset this random data
data_unseen = dataset.drop(data.index)
data_unseen
```

[5] ✓ 0.0s Python

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default
5	100000	2	2	2	23	0	-1	-1	0	0	...	221.0	-159.0	567.0	380.0	601.0	0.0	581.0	1687.0	1542.0	0
39	380000	1	2	2	32	-1	-1	-1	-1	-1	...	32018.0	11849.0	11873.0	21540.0	15138.0	24677.0	11851.0	11875.0	8251.0	0
57	200000	2	2	1	32	-1	-1	-1	-1	2	...	5247.0	3848.0	3151.0	5818.0	15.0	9102.0	17.0	3165.0	1395.0	0
72	200000	1	1	1	53	2	2	2	2	2	...	144098.0	147124.0	149531.0	6300.0	5500.0	5500.0	5500.0	5000.0	5000.0	1
103	240000	1	1	2	41	1	-1	-1	0	0	...	3164.0	360.0	1737.0	2622.0	3301.0	0.0	360.0	1737.0	924.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23978	50000	1	2	1	37	1	2	2	2	0	...	2846.0	1585.0	1324.0	0.0	3000.0	0.0	0.0	1000.0	1000.0	1
23979	220000	1	2	1	41	0	0	-1	-1	-2	...	5924.0	1759.0	1824.0	8840.0	6643.0	5924.0	1759.0	1824.0	7022.0	0
23981	420000	1	1	2	34	0	0	0	0	0	...	141695.0	144839.0	147954.0	7000.0	7000.0	5500.0	5500.0	5600.0	5000.0	0
23985	90000	1	2	1	36	0	0	0	0	0	...	11328.0	12036.0	14329.0	1500.0	1500.0	1500.0	1200.0	2500.0	0.0	1
23999	50000	1	2	1	46	0	0	0	0	0	...	36535.0	32428.0	15313.0	2078.0	1800.0	1430.0	1000.0	1000.0	1000.0	1

1200 rows × 24 columns

```
## we reset the index of both datasets
data.reset_index(inplace=True, drop=True)
data_unseen.reset_index(inplace=True, drop=True)
print('Data for Modeling: ' + str(data.shape))
print('Unseen Data For Predictions: ' + str(data_unseen.shape))
```

[6] ✓ 0.0s Python

```
Data for Modeling: (22800, 24)
Unseen Data For Predictions: (1200, 24)
```

Step 2 Setting up the PyCaret environment Now

- The setup() function initializes PyCaret's environment and prepares the data pipeline for modeling and deployment.

Mandatory Parameters

- A pandas dataframe.
- The name of the target column.

- Must be called before other PyCaret functions.

Automatic Configurations

- Default train-test split: 70:30 (modifiable with train_size).
- K-fold cross-validation: 10 folds.
- Reproducibility: session_id acts as the seed (random_state)

```
## setting up the environment
from pycaret.classification import *
model_setup = setup(data=data, target='default', session_id=123)
```

[7] ✓ 2.8s

...

	Description	Value
0	Session id	123
1	Target	default
2	Target type	Binary
3	Original data shape	(22800, 24)
4	Transformed data shape	(22800, 24)
5	Transformed train set shape	(15959, 24)
6	Transformed test set shape	(6841, 24)
7	Numeric features	23
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	StratifiedKFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False
16	Log Experiment	False
17	Experiment Name	clf-default-name
18	USI	7c1c

Important Outputs of setup()

Data Characteristics

- Original Data: Shape of the raw dataset.
- Transformed Train/Test Sets: Includes expanded features after preprocessing.
- Numerical and Categorical Features: Counts of features by type.

Target Variable Information

- Binary or multiclass detection.
- Automatic label encoding for string-based targets (Yes/No → 1/0).

Preprocess

- Indicates that preprocessing were done.

	Description	Value
0	Session id	123
1	Target	default
2	Target type	Binary
3	Original data shape	(22800, 24)
4	Transformed data shape	(22800, 24)
5	Transformed train set shape	(15959, 24)
6	Transformed test set shape	(6841, 24)
7	Numeric features	23
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	StratifiedKFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False
16	Log Experiment	False
17	Experiment Name	clf-default-name
18	USI	7c1c

Step 3 Comparison of the Models

Key Function: compare_models()

- Trains and evaluates multiple models using stratified cross-validation (default 10-fold).
- Outputs a score grid with metrics like Accuracy, AUC, Recall, Precision, F1, Kappa, and MCC.

Gradient Boosting Classifier:

- highest accuracy (82.24%)
- AUC (0.7890),

Ridge Classifier: low Recall (15.58%),

Random Forest:

- good accuracy (81.93%)
- AUC (0.7748)

```
best_model = compare_models()
```

[8] ✓ 48.2s

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
gbc	Gradient Boosting Classifier	0.8224	0.7890	0.3756	0.6787	0.4833	0.3867	0.4115	0.9260
lightgbm	Light Gradient Boosting Machine	0.8202	0.7842	0.3816	0.6630	0.4842	0.3848	0.4064	0.1720
ada	Ada Boost Classifier	0.8200	0.7802	0.3476	0.6847	0.4604	0.3660	0.3966	0.2380
rf	Random Forest Classifier	0.8193	0.7748	0.3870	0.6552	0.4864	0.3855	0.4053	0.5000
lda	Linear Discriminant Analysis	0.8151	0.7212	0.2739	0.7152	0.3958	0.3114	0.3618	0.0210
et	Extra Trees Classifier	0.8130	0.7677	0.3793	0.6277	0.4728	0.3674	0.3847	0.2600
lr	Logistic Regression	0.8047	0.7060	0.2207	0.6945	0.3296	0.2505	0.3089	0.7730
ridge	Ridge Classifier	0.8007	0.7212	0.1558	0.7328	0.2567	0.1943	0.2735	0.0200
dummy	Dummy Classifier	0.7788	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0190
knn	K Neighbors Classifier	0.7504	0.6099	0.1796	0.3689	0.2415	0.1129	0.1234	0.4490
dt	Decision Tree Classifier	0.7295	0.6207	0.4244	0.3960	0.4096	0.2345	0.2348	0.0770
svm	SVM - Linear Kernel	0.6188	0.5846	0.3895	0.2528	0.2759	0.0585	0.0654	0.0670
qda	Quadratic Discriminant Analysis	0.5668	0.7254	0.7861	0.3143	0.4468	0.1910	0.2460	0.0210
nb	Naive Bayes	0.3889	0.6816	0.8765	0.2493	0.3881	0.0668	0.1273	0.0210

```
print(best_model)
```

[9] ✓ 0.0s

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='log_loss', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_leaf=0.0,
                           n_estimators=100, n_iter_no_change=None,
                           random_state=123, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

Step 4 Create the Model

Purpose

- Trains and evaluates a specific machine learning model using cross-validation.

Output

- Prints a scoring table with metrics like Precision, AUC, Recall, F1, Kappa, and MCC. Displays model hyperparameters, critical for further optimization.

Key Features of create_model

- **Granularity**
  - Most granular function in PyCaret.
  - Foundation for other PyCaret functions.
- **Cross-Validation**
  - Metrics are calculated for each fold and averaged across all folds.
  - Customizable using the fold parameter.

Random Forest Classifier

- The model achieves good accuracy (81.93%) and a solid AUC, indicating the model's ability to discriminate between classes.
- Recall (38.70%), however, is quite low, indicating that the model identifies only a minority of true positive cases.
- The results are consistent between folds, as evidenced by the low standard deviation.

```
rf = create_model('rf')
```

[61] ✓ 6.1s

```
Initiated  . . . . . 21:58:34
Status     . . . . . Loading Dependencies
Estimator  . . . . . Compiling Library
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8227	0.7717	0.4108	0.6591	0.5061	0.4051	0.4219
1	0.8271	0.7726	0.4023	0.6860	0.5071	0.4108	0.4323
2	0.8233	0.8079	0.4108	0.6621	0.5070	0.4065	0.4237
3	0.8120	0.7501	0.3711	0.6268	0.4662	0.3611	0.3794
4	0.8139	0.7623	0.3541	0.6443	0.4570	0.3560	0.3793
5	0.8221	0.7827	0.3909	0.6667	0.4929	0.3937	0.4144
6	0.8120	0.7607	0.3711	0.6268	0.4662	0.3611	0.3794
7	0.8258	0.7959	0.4023	0.6794	0.5053	0.4079	0.4286
8	0.8208	0.7602	0.3739	0.6701	0.4800	0.3821	0.4058
9	0.8138	0.7839	0.3824	0.6308	0.4762	0.3711	0.3883
Mean	0.8193	0.7748	0.3870	0.6552	0.4864	0.3855	0.4053
Std	0.0055	0.0170	0.0184	0.0207	0.0186	0.0208	0.0207

```
#trained model object is stored in the variable 'dt'.
print(rf)
```

[62] ✓ 0.0s

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='sqrt',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       monotonic_cst=None, n_estimators=100, n_jobs=-1,
                       oob_score=False, random_state=123, verbose=0,
                       warm_start=False)
```

Ridge Classifier

- Model performance: The model has high accuracy and prediction accuracy (Precision), but significantly low Recall, which means that it often ignores positive cases.
- Use of the Ridge Classifier: the model might be appropriate where it is important to minimize false positives, but it is not a priority to capture all positive cases.
- The model shows good Accuracy and Precision, but the low Recall suggests the need for further optimization, especially if capturing positive cases is a priority.

```
ridge = create_model('ridge')
```

[63] ✓ 0.6s

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8058	0.7360	0.1671	0.7867	0.2757	0.2148	0.3026
1	0.8076	0.7160	0.1700	0.8108	0.2810	0.2213	0.3133
2	0.8020	0.7334	0.1700	0.7229	0.2752	0.2086	0.2831
3	0.8001	0.7118	0.1700	0.6977	0.2733	0.2044	0.2740
4	0.8051	0.7124	0.1671	0.7763	0.2751	0.2134	0.2991
5	0.8058	0.7382	0.1615	0.8028	0.2689	0.2104	0.3024
6	0.7995	0.7082	0.1530	0.7200	0.2523	0.1895	0.2669
7	0.7932	0.7290	0.1161	0.6949	0.1990	0.1449	0.2236
8	0.8014	0.7176	0.1586	0.7368	0.2611	0.1982	0.2778
9	0.7862	0.7095	0.1246	0.5789	0.2051	0.1375	0.1927
Mean	0.8007	0.7212	0.1558	0.7328	0.2567	0.1943	0.2735
Std	0.0062	0.0111	0.0186	0.0651	0.0284	0.0279	0.0362

```
print(ridge)
```

[64] ✓ 0.0s

```
RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,
                max_iter=None, positive=False, random_state=123, solver='auto',
                tol=0.0001)
```

Gradient Boosting Classifier

- The model has high accuracy and good AUC, indicating good overall performance.
- However, the recall is quite low, which means that the model does not capture all positive cases.
- Precision and F1 scores show that the model is balanced but has room for improvement in capturing positive cases.
- The Gradient Boosting Classifier is a well-adjusted model with high precision and AUC, but the low Recall indicates the need for further optimization, especially if the priority is to capture more positive cases.

```
gbc = create_model ("gbc")
```

[44] ✓ 25.0s

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8239	0.7862	0.3853	0.6800	0.4919	0.3951	0.4184
1	0.8239	0.7967	0.3683	0.6915	0.4806	0.3862	0.4141
2	0.8208	0.8278	0.3938	0.6588	0.4929	0.3923	0.4115
3	0.8202	0.7770	0.3711	0.6684	0.4772	0.3792	0.4031
4	0.8258	0.7837	0.3654	0.7049	0.4813	0.3891	0.4194
5	0.8208	0.8065	0.3711	0.6718	0.4781	0.3806	0.4051
6	0.8177	0.7650	0.3598	0.6615	0.4661	0.3675	0.3923
7	0.8340	0.7963	0.3994	0.7268	0.5155	0.4254	0.4532
8	0.8214	0.7634	0.3569	0.6848	0.4693	0.3745	0.4032
9	0.8157	0.7876	0.3853	0.6385	0.4806	0.3767	0.3945
Mean	0.8224	0.7890	0.3756	0.6787	0.4833	0.3867	0.4115
Std	0.0048	0.0183	0.0137	0.0238	0.0133	0.0152	0.0164

```
print(gbc)
```

[46] ✓ 0.0s

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='log_loss', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_leaf=0.0,
                           n_estimators=100, n_iter_no_change=None,
                           random_state=123, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

Thank you very much for your attention!

Literature

- Alamin, M. A. A., & Uddin, G. (2018). Challenges and barriers of using low code software for machine learning. *ACM Transactions.*
- Data Valley. (n.d.). *Kaggle survey insights and analysis*. Retrieved December 1, 2024, from https://datavalley.technology/kaggle/
- Hasim, N., & Haris, N. A. (2015). A study of open-source data mining tools for forecasting. *Proceedings of the ACM International Conference.* https://doi.org/10.1145/2701126.2701152
- Kaggle. (2021). *Kaggle State of Machine Learning and Data Science Report 2021*. Retrieved from https://storage.googleapis.com/kaggle-media/surveys/Kaggle%20State%20of%20Machine%20Learning%20and%20Data%20Science%20Report%202021.pdf
- Kaggle. (2022). *Kaggle State of Machine Learning and Data Science Report 2022*. Retrieved from https://storage.googleapis.com/kaggle-media/surveys/Kaggle%20State%20of%20Machine%20Learning%20and%20Data%20Science%20Report%202022.pdf

Literature (cont.)

- Naik, A., & Samant, L. (2016). Correlation review of classification algorithm using data mining tools: WEKA, Rapidminer, Tanagra, Orange, and Knime. *Procedia Computer Science, 85,* 662-668. https://doi.org/10.1016/j.procs.2016.05.251
- Ng Mh. (2024). A comparison of the top 3 tools for machine learning beginners: Orange vs PyCaret vs Scikit. *Tinkercademy Build Log.* Retrieved from https://blog.tinkercademy.com/orange-v-s-pycaret-v-s-scikit-a-comparison-of-beginner-machine-learning-libraries-2d79c0e43e3f
- Rexer Analytics. (2023). *Data Science Survey 2023: Highlights from the TechCast (July 27, 2023)*. Retrieved December 1, 2024, from https://andouc.org/wp-content/uploads/2023/10/DSS-2023-Highlights-REXER-TechCast-7-27-23-CDChanges.pdf
- Tolios, G. (2023). *Simplifying Machine Learning with PyCaret: A Low-code Approach for Beginners and Experts!* Leanpub.
- Wahbeh, A. H., Al-Radaideh, Q. A., Al-Kabi, M. N., & Al-Shawakfa, E. (2011). A comparison study between data mining tools over some classification methods. *International Journal of Advanced Computer Science and Applications.* https://doi.org/10.14569/SpecialIssue.2011.010304
- Yan, Z. (2022). The impacts of low/no-code development on digital transformation and software development. *University of Toronto.* Retrieved from https://arxiv.org/pdf/2112.14073v1.pdf
- Yeh, I.-C. (2016). *Default of credit card clients dataset*. UCI Machine Learning Repository. University of California, Irvine. https://doi.org/10.24432/C55S3H

The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.