Last updated: 16 Feb 2023

👋 PyCaret Binary Classification Tutorial

PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows. It is an end-to-end machine learning and model management tool that exponentially speeds up the experiment cycle and makes you more productive.

Compared with the other open-source machine learning libraries, PyCaret is an alternate low-code library that can be used to replace hundreds of lines of code with a few lines only. This makes experiments exponentially fast and efficient. PyCaret is essentially a Python wrapper around several machine learning libraries and frameworks, such as scikit-learn, XGBoost, LightGBM, CatBoost, spaCy, Optuna, Hyperopt, Ray, and a few more.

The design and simplicity of PyCaret are inspired by the emerging role of citizen data scientists, a term first used by Gartner. Citizen Data Scientists are power users who can perform both simple and moderately sophisticated analytical tasks that would previously have required more technical expertise.

💻 Installation

PyCaret is tested and supported on the following 64-bit systems:

- Python 3.7 – 3.10
- Python 3.9 for Ubuntu only
- Ubuntu 16.04 or later
- Windows 7 or later

You can install PyCaret with Python's pip package manager:

`pip install pycaret`

PyCaret's default installation will not install all the extra dependencies automatically. For that you will have to install the full version:

`pip install pycaret[full]`

or depending on your use-case you may install one of the following variant:

- `pip install pycaret[analysis]`
- `pip install pycaret[models]`
- `pip install pycaret[tuner]`
- `pip install pycaret[mlops]`
- `pip install pycaret[parallel]`
- `pip install pycaret[test]`

In [1]:
```
# check installed version
import pycaret
pycaret.__version__
```

Out[1]: '3.0.0'

🚀 Quick start

PyCaret's Classification Module is a supervised machine learning module that is used for classifying elements into groups. The goal is to predict the categorical class labels which are discrete and unordered.

Some common use cases include predicting customer default (Yes or No), predicting customer churn (customer will leave or stay), the disease found (positive or negative).

This module can be used for binary or multiclass problems. It provides several pre-processing features that prepare the data for modeling through the setup function. It has over 18 ready-to-use algorithms and several plots to analyze the performance of trained models.

A typical workflow in PyCaret consist of following 5 steps in this order:

Setup ➡️ Compare Models ➡️ Analyze Model ➡️ Prediction ➡️ Save Model

In [2]:
```
# loading sample dataset from pycaret dataset module
from pycaret.datasets import get_data
data = get_data('diabetes')
```

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

Setup

This function initializes the training environment and creates the transformation pipeline. Setup function must be called before executing any other function in PyCaret. It only has two required parameters i.e. `data` and `target`. All the other parameters are optional.

In [3]:
```
# import pycaret classification and init setup
from pycaret.classification import *
s = setup(data, target = 'Class variable', session_id = 123)
```

	Description	Value
0	Session id	123
1	Target	Class variable
2	Target type	Binary
3	Original data shape	(768, 9)
4	Transformed data shape	(768, 9)
5	Transformed train set shape	(537, 9)
6	Transformed test set shape	(231, 9)
7	Numeric features	8
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	52db

Once the setup has been successfully executed it shows the information grid containing experiment level information.

- **Session id:** A pseudo-random number distributed as a seed in all functions for later reproducibility. If no `session_id` is passed, a random number is automatically generated that is distributed to all functions.

- **Target type:** Binary, Multiclass, or Regression. The Target type is automatically detected.

- **Label Encoding:** When the Target variable is of type string (i.e. 'Yes' or 'No') instead of 1 or 0, it automatically encodes the label into 1 and 0 and displays the mapping (0 : No, 1 : Yes) for reference. In this tutorial, no label encoding is required since the target variable is of numeric type.

- **Original data shape:** Shape of the original data prior to any transformations.

- **Transformed train set shape :** Shape of transformed train set

- **Transformed test set shape :** Shape of transformed test set

- **Numeric features :** The number of features considered as numerical.

- **Categorical features :** The number of features considered as categorical.

 can work with. (1) Functional (as seen above) and (2)

Object Oriented API.

With Object Oriented API instead of executing functions directly you will import a class and execute methods of class.

In [4]:
```
# import ClassificationExperiment and init the class
from pycaret.classification import ClassificationExperiment
exp = ClassificationExperiment()
```

In [5]:
```
# check the type of exp
type(exp)
```

Out[5]: pycaret.classification.oop.ClassificationExperiment

In [6]:
```
# init setup on exp
exp.setup(data, target = 'Class variable', session_id = 123)
```

	Description	Value
0	Session id	123
1	Target	Class variable
2	Target type	Binary
3	Original data shape	(768, 9)
4	Transformed data shape	(768, 9)
5	Transformed train set shape	(537, 9)
6	Transformed test set shape	(231, 9)
7	Numeric features	8
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	0071

Out[6]: <pycaret.classification.oop.ClassificationExperiment at 0x2e24286edc0>

You can use any of the two method i.e. Functional or OOP and even switch back and forth between two set of API's. The choice of method will not impact the results and has been tested for consistency.

Compare Models

This function trains and evaluates the performance of all the estimators available in the model library using cross-validation. The output of this function is a scoring grid with average cross-validated scores. Metrics evaluated during CV can be accessed using the `get_metrics` function. Custom metrics can be added or removed using `add_metric` and `remove_metric` function.

In [7]:
```
# compare baseline models
best = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.7689	0.8047	0.5602	0.7208	0.6279	0.4641	0.4736	1.3810
ridge	Ridge Classifier	0.7670	0.0000	0.5497	0.7235	0.6221	0.4581	0.4690	0.0370
lda	Linear Discriminant Analysis	0.7670	0.8055	0.5550	0.7202	0.6243	0.4594	0.4695	0.0500
rf	Random Forest Classifier	0.7485	0.7911	0.5284	0.6811	0.5924	0.4150	0.4238	0.1940
nb	Naive Bayes	0.7427	0.7955	0.5702	0.6543	0.6043	0.4156	0.4215	0.0400
catboost	CatBoost Classifier	0.7410	0.7993	0.5278	0.6630	0.5851	0.4005	0.4078	0.0890
gbc	Gradient Boosting Classifier	0.7373	0.7918	0.5550	0.6445	0.5931	0.4013	0.4059	0.0770
ada	Ada Boost Classifier	0.7372	0.7799	0.5275	0.6585	0.5796	0.3926	0.4017	0.0870
et	Extra Trees Classifier	0.7299	0.7788	0.4965	0.6516	0.5596	0.3706	0.3802	0.1280
qda	Quadratic Discriminant Analysis	0.7282	0.7894	0.5281	0.6558	0.5736	0.3785	0.3910	0.0510
lightgbm	Light Gradient Boosting Machine	0.7133	0.7645	0.5398	0.6036	0.5650	0.3534	0.3580	0.2440
knn	K Neighbors Classifier	0.7001	0.7164	0.5020	0.5982	0.5413	0.3209	0.3271	0.0570
dt	Decision Tree Classifier	0.6928	0.6512	0.5137	0.5636	0.5328	0.3070	0.3098	0.0460
xgboost	Extreme Gradient Boosting	0.6853	0.7516	0.4912	0.5620	0.5216	0.2887	0.2922	0.0520
dummy	Dummy Classifier	0.6518	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0380
svm	SVM - Linear Kernel	0.5954	0.0000	0.3395	0.4090	0.2671	0.0720	0.0912	0.0410

```
Processing:   0%|          | 0/69 [00:00<?, ?it/s]
```

In [8]:
```
# compare models using OOP
exp.compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.7689	0.8047	0.5602	0.7208	0.6279	0.4641	0.4736	0.0450
ridge	Ridge Classifier	0.7670	0.0000	0.5497	0.7235	0.6221	0.4581	0.4690	0.0330
lda	Linear Discriminant Analysis	0.7670	0.8055	0.5550	0.7202	0.6243	0.4594	0.4695	0.0370
rf	Random Forest Classifier	0.7485	0.7911	0.5284	0.6811	0.5924	0.4150	0.4238	0.1320
nb	Naive Bayes	0.7427	0.7955	0.5702	0.6543	0.6043	0.4156	0.4215	0.0360
catboost	CatBoost Classifier	0.7410	0.7993	0.5278	0.6630	0.5851	0.4005	0.4078	0.0340
gbc	Gradient Boosting Classifier	0.7373	0.7918	0.5550	0.6445	0.5931	0.4013	0.4059	0.0730
ada	Ada Boost Classifier	0.7372	0.7799	0.5275	0.6585	0.5796	0.3926	0.4017	0.0750
et	Extra Trees Classifier	0.7299	0.7788	0.4965	0.6516	0.5596	0.3706	0.3802	0.1320
qda	Quadratic Discriminant Analysis	0.7282	0.7894	0.5281	0.6558	0.5736	0.3785	0.3910	0.0380
lightgbm	Light Gradient Boosting Machine	0.7133	0.7645	0.5398	0.6036	0.5650	0.3534	0.3580	0.0390
knn	K Neighbors Classifier	0.7001	0.7164	0.5020	0.5982	0.5413	0.3209	0.3271	0.0490
dt	Decision Tree Classifier	0.6928	0.6512	0.5137	0.5636	0.5328	0.3070	0.3098	0.0390
xgboost	Extreme Gradient Boosting	0.6853	0.7516	0.4912	0.5620	0.5216	0.2887	0.2922	0.0440
dummy	Dummy Classifier	0.6518	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0330
svm	SVM - Linear Kernel	0.5954	0.0000	0.3395	0.4090	0.2671	0.0720	0.0912	0.0310

```
Processing:   0%|          | 0/69 [00:00<?, ?it/s]
```

Out[8]:

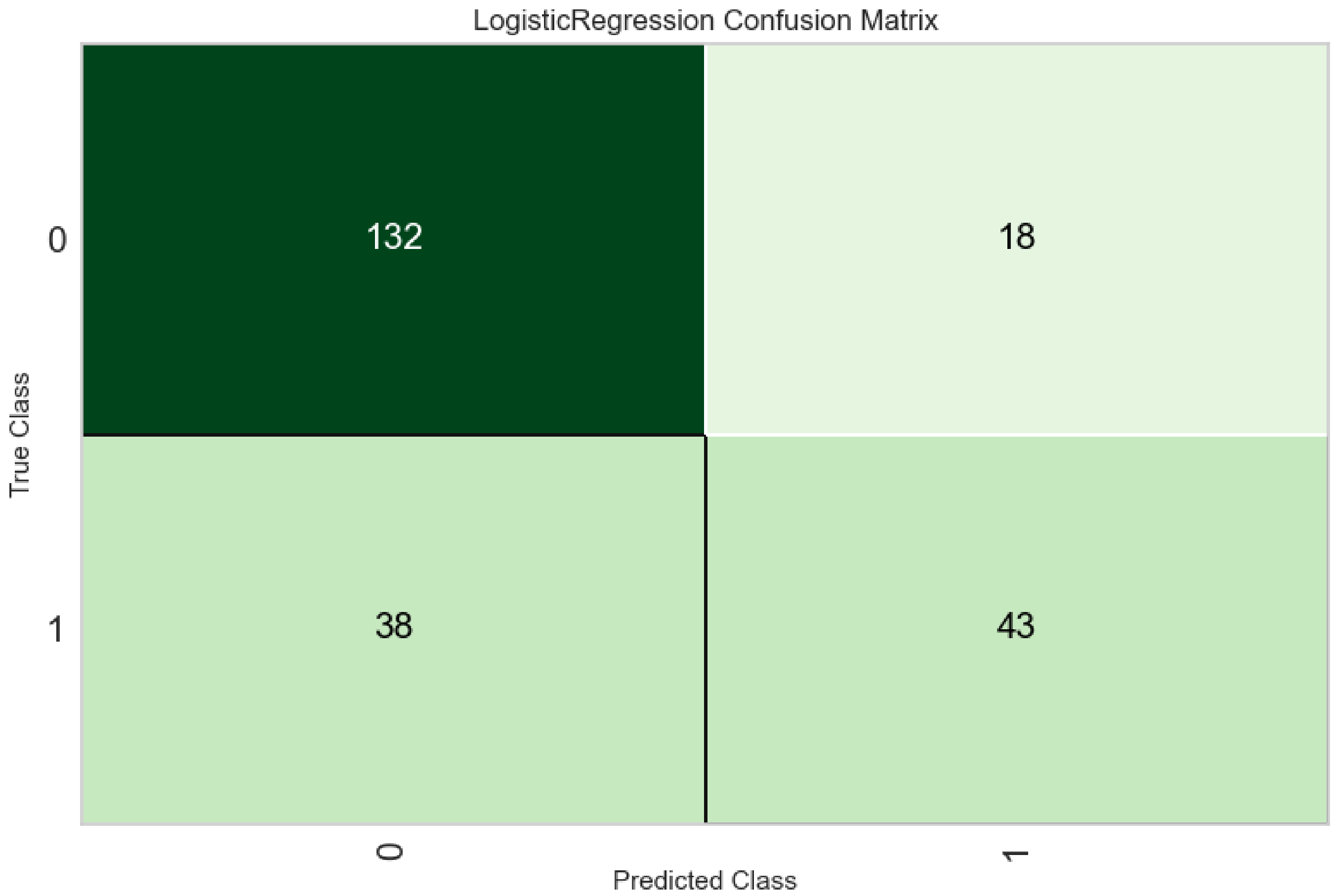
```
▾ LogisticRegression
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=123, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Notice that the output between functional and OOP API is consistent. Rest of the functions in this notebook will only be shown using functional API only.

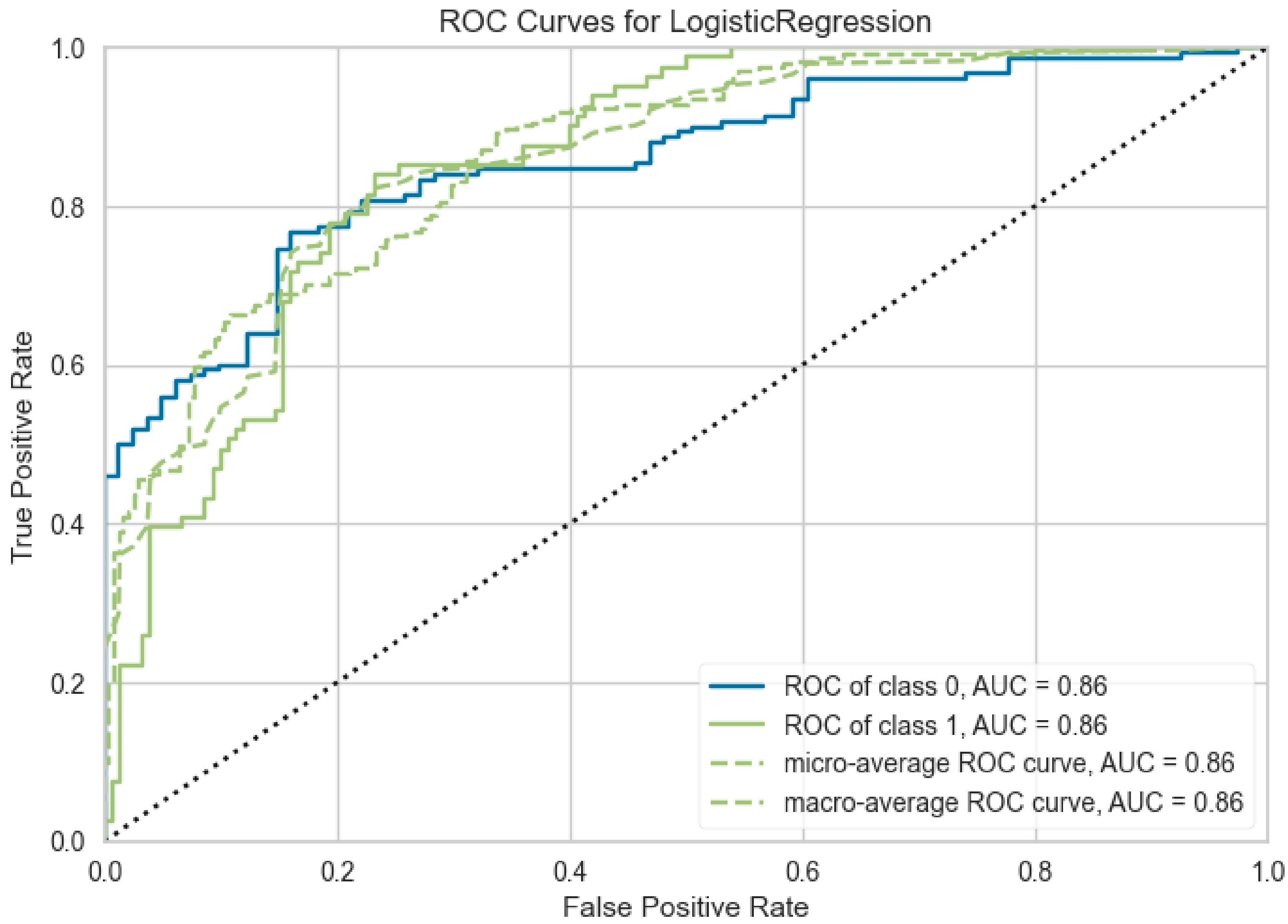
Analyze Model

You can use the `plot_model` function to analyzes the performance of a trained model on the test set. It may require re-training the model in certain cases.

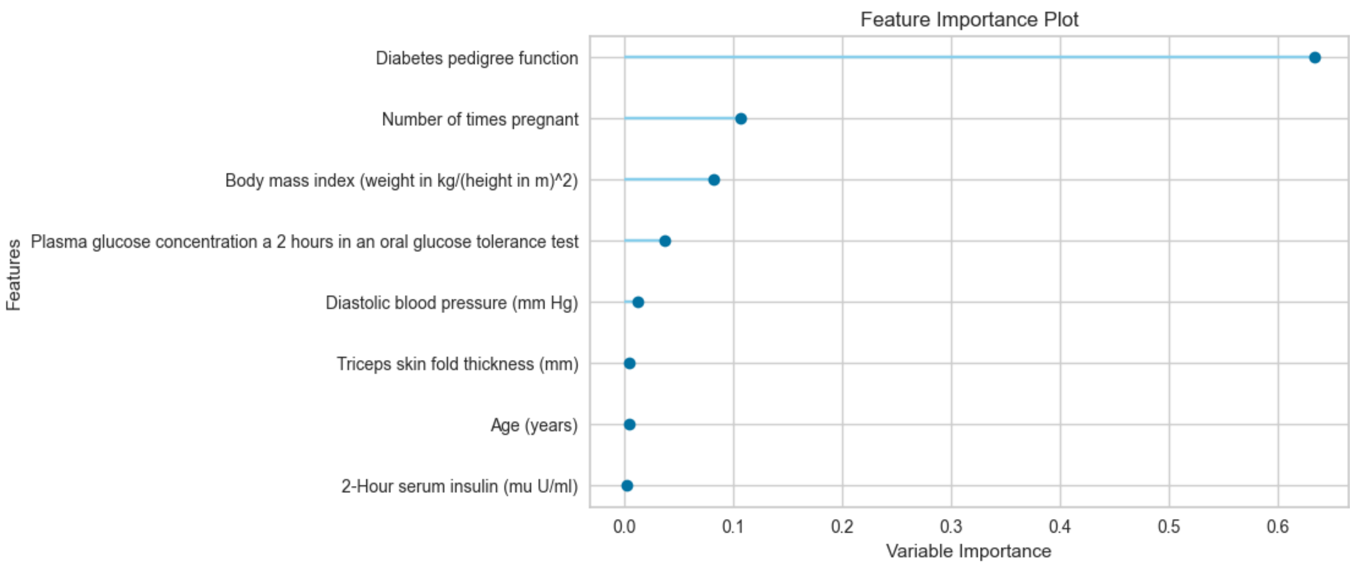
In [9]:
```
# plot confusion matrix
plot_model(best, plot = 'confusion_matrix')
```



In [10]:
```
# plot AUC
plot_model(best, plot = 'auc')
```



In [11]:
```
# plot feature importance
plot_model(best, plot = 'feature')
```



In [105…
```
# check docstring to see available plots
# help(plot_model)
```

An alternate to `plot_model` function is `evaluate_model`. It can only be used in Notebook since it uses ipywidget.

In [13]:
```
evaluate_model(best)
```

```
interactive(children=(ToggleButtons(description='Plot Type:', icons=('',), options=
(('Pipeline Plot', 'pipelin…
```

Prediction

The `predict_model` function returns `prediction_label` and `prediction_score` (probability of the predicted class) as new columns in dataframe. When data is `None` (default), it uses the test set (created during the setup function) for scoring.

In [14]:
```
# predict on test set
holdout_pred = predict_model(best)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.7576	0.8568	0.5309	0.7049	0.6056	0.4356	0.4447

In [15]:
```
# show predictions df
holdout_pred.head()
```

Out[15]:

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	A (year
537	6	114	88	0	0	27.799999	0.247	
538	1	97	70	15	0	18.200001	0.147	
539	2	90	70	17	0	27.299999	0.085	
540	2	105	58	40	94	34.900002	0.225	
541	11	138	76	0	0	33.200001	0.420	

The same function works for predicting the labels on unseen dataset. Let's create a copy of original data and drop the `Class variable`. We can then use the new data frame without labels for scoring.

In [16]:
```
# copy data and drop Class variable

new_data = data.copy()
new_data.drop('Class variable', axis=1, inplace=True)
new_data.head()
```

Out[16]:

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

In [17]:
```
# predict model on new_data
predictions = predict_model(best, data = new_data)
```

```
predictions.head()
```

Out[17]:

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)
0	6	148	72	35	0	33.599998	0.627	50
1	1	85	66	29	0	26.600000	0.351	31
2	8	183	64	0	0	23.299999	0.672	32
3	1	89	66	23	94	28.100000	0.167	21
4	0	137	40	35	168	43.099998	2.288	33

Save Model

Finally, you can save the entire pipeline on disk for later use, using pycaret's `save_model` function.

In [18]:
```
# save pipeline
save_model(best, 'my_first_pipeline')
```

```
Transformation Pipeline and Model Successfully Saved
```

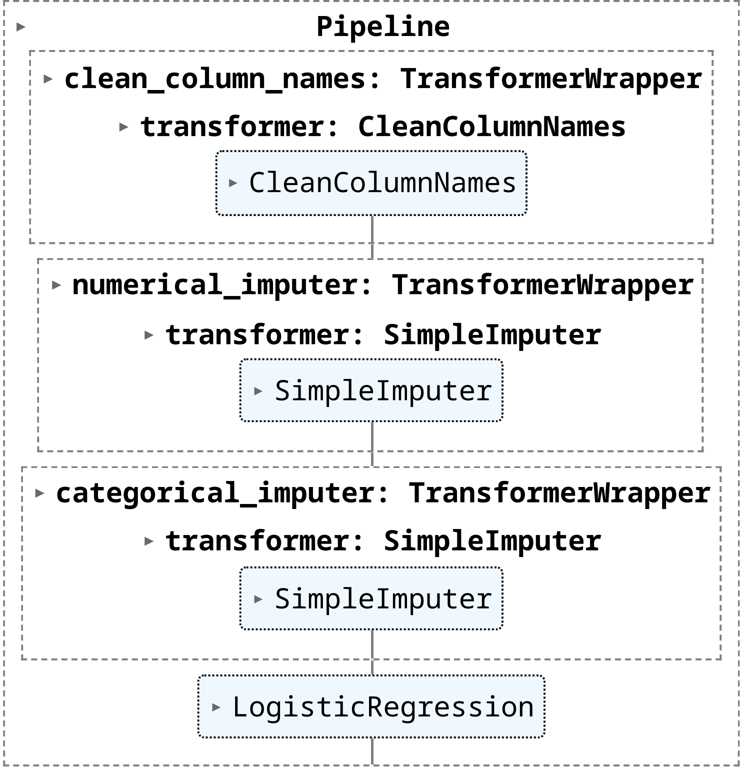
Out[18]:
```
(Pipeline(memory=FastMemory(location=C:\Users\owner\AppData\Local\Temp\joblib),
          steps=[('clean_column_names',
                  TransformerWrapper(exclude=None, include=None,
                                     transformer=CleanColumnNames(match='[\\]\\
[\\,\\{\\}\\"\\:]+'))),
                 ('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['Number of times pregnant',
                                              'Plasma glucose concentration a 2 '
                                              'hours in an oral glu...
                                                               fill_value=None,
                                                               missing_values=nan,
                                                               strategy='most_freque
nt',

                                                               verbose='deprecate
d'))),
                 ('trained_model',
                  LogisticRegression(C=1.0, class_weight=None, dual=False,
                                     fit_intercept=True, intercept_scaling=1,
                                     l1_ratio=None, max_iter=1000,
                                     multi_class='auto', n_jobs=None,
                                     penalty='l2', random_state=123,
                                     solver='lbfgs', tol=0.0001, verbose=0,
                                     warm_start=False))],
          verbose=False),
 'my_first_pipeline.pkl')
```

In [19]:
```
# load pipeline
loaded_best_pipeline = load_model('my_first_pipeline')
loaded_best_pipeline
```

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

Out[19]:



👇 Detailed function-by-function overview

✅ Setup

This function initializes the experiment in PyCaret and creates the transformation pipeline based on all the parameters passed in the function. Setup function must be called before executing any other function. It takes two required parameters: `data` and `target`. All the other parameters are optional and are used for configuring data preprocessing pipeline.

In [20]:

```
# init setup function
s = setup(data, target = 'Class variable', session_id = 123)
```

	Description	Value
0	Session id	123
1	Target	Class variable
2	Target type	Binary
3	Original data shape	(768, 9)
4	Transformed data shape	(768, 9)
5	Transformed train set shape	(537, 9)
6	Transformed test set shape	(231, 9)
7	Numeric features	8
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	038a

To access all the variables created by the setup function such as transformed dataset, random_state, etc. you can use `get_config` method.

In [21]:
```
# check all available config
get_config()
```

Out[21]:
```
{'USI',
 'X',
 'X_test',
 'X_test_transformed',
 'X_train',
 'X_train_transformed',
 'X_transformed',
 '_available_plots',
 '_ml_usecase',
 'data',
 'dataset',
 'dataset_transformed',
 'exp_id',
 'exp_name_log',
 'fix_imbalance',
 'fold_generator',
 'fold_groups_param',
 'fold_shuffle_param',
 'gpu_n_jobs_param',
 'gpu_param',
 'html_param',
 'idx',
 'is_multiclass',
 'log_plots_param',
 'logging_param',
 'memory',
 'n_jobs_param',
 'pipeline',
 'seed',
 'target_param',
 'test',
 'test_transformed',
 'train',
 'train_transformed',
 'variable_and_property_keys',
 'variables',
 'y',
 'y_test',
 'y_test_transformed',
 'y_train',
 'y_train_transformed',
 'y_transformed'}
```

In [22]:
```
# lets access X_train_transformed
get_config('X_train_transformed')
```

Out[22]:

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	A(yea
0	13.0	152.0	90.0	33.0	29.0	26.799999	0.731	43
1	0.0	104.0	64.0	37.0	64.0	33.599998	0.510	22
2	5.0	137.0	108.0	0.0	0.0	48.799999	0.227	37
3	0.0	111.0	65.0	0.0	0.0	24.600000	0.660	31
4	6.0	105.0	70.0	32.0	68.0	30.799999	0.122	37
...	...	...	...	...	...	...	...	
532	10.0	179.0	70.0	0.0	0.0	35.099998	0.200	37
533	0.0	100.0	88.0	60.0	110.0	46.799999	0.962	31
534	1.0	89.0	76.0	34.0	37.0	31.200001	0.192	23
535	1.0	121.0	78.0	39.0	74.0	39.000000	0.261	28
536	0.0	140.0	65.0	26.0	130.0	42.599998	0.431	24

537 rows × 8 columns

In [23]:

```
# another example: let's access seed
print("The current seed is: {}".format(get_config('seed')))

# now lets change it using set_config
set_config('seed', 786)
print("The new seed is: {}".format(get_config('seed')))
```

```
The current seed is: 123
The new seed is: 786
```

All the preprocessing configurations and experiment settings/parameters are passed into the `setup` function. To see all available parameters, check the docstring:

In [24]:

```
# help(setup)
```

In [25]:

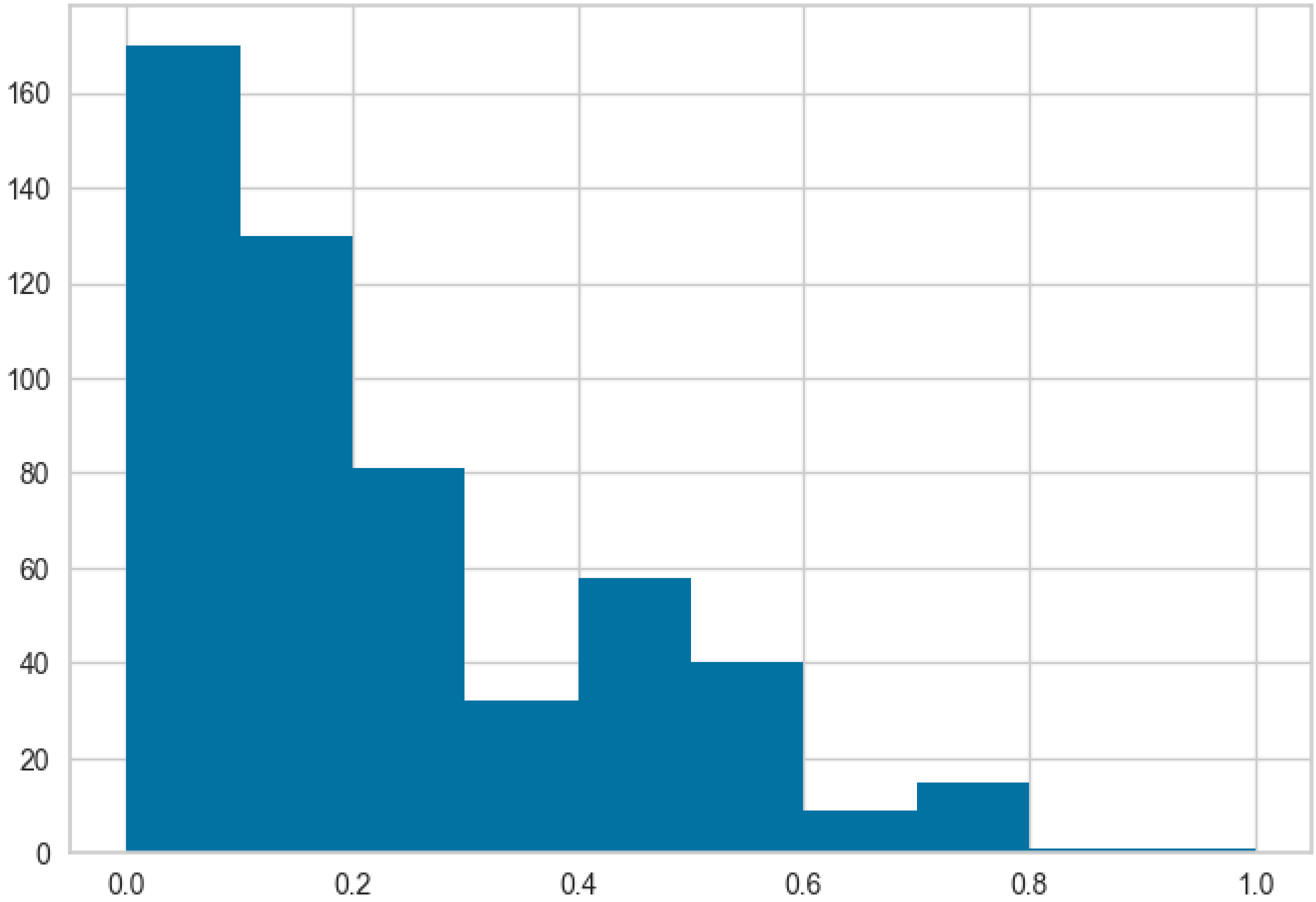
```
# init setup with normalize = True

s = setup(data, target = 'Class variable', session_id = 123,
          normalize = True, normalize_method = 'minmax')
```

	Description	Value
0	Session id	123
1	Target	Class variable
2	Target type	Binary
3	Original data shape	(768, 9)
4	Transformed data shape	(768, 9)
5	Transformed train set shape	(537, 9)
6	Transformed test set shape	(231, 9)
7	Numeric features	8
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Normalize	True
13	Normalize method	minmax
14	Fold Generator	StratifiedKFold
15	Fold Number	10
16	CPU Jobs	-1
17	Use GPU	False
18	Log Experiment	False
19	Experiment Name	clf-default-name
20	USI	f18d

In [26]:
```
# lets check the X_train_transformed to see effect of params passed
get_config('X_train_transformed')['Number of times pregnant'].hist()
```

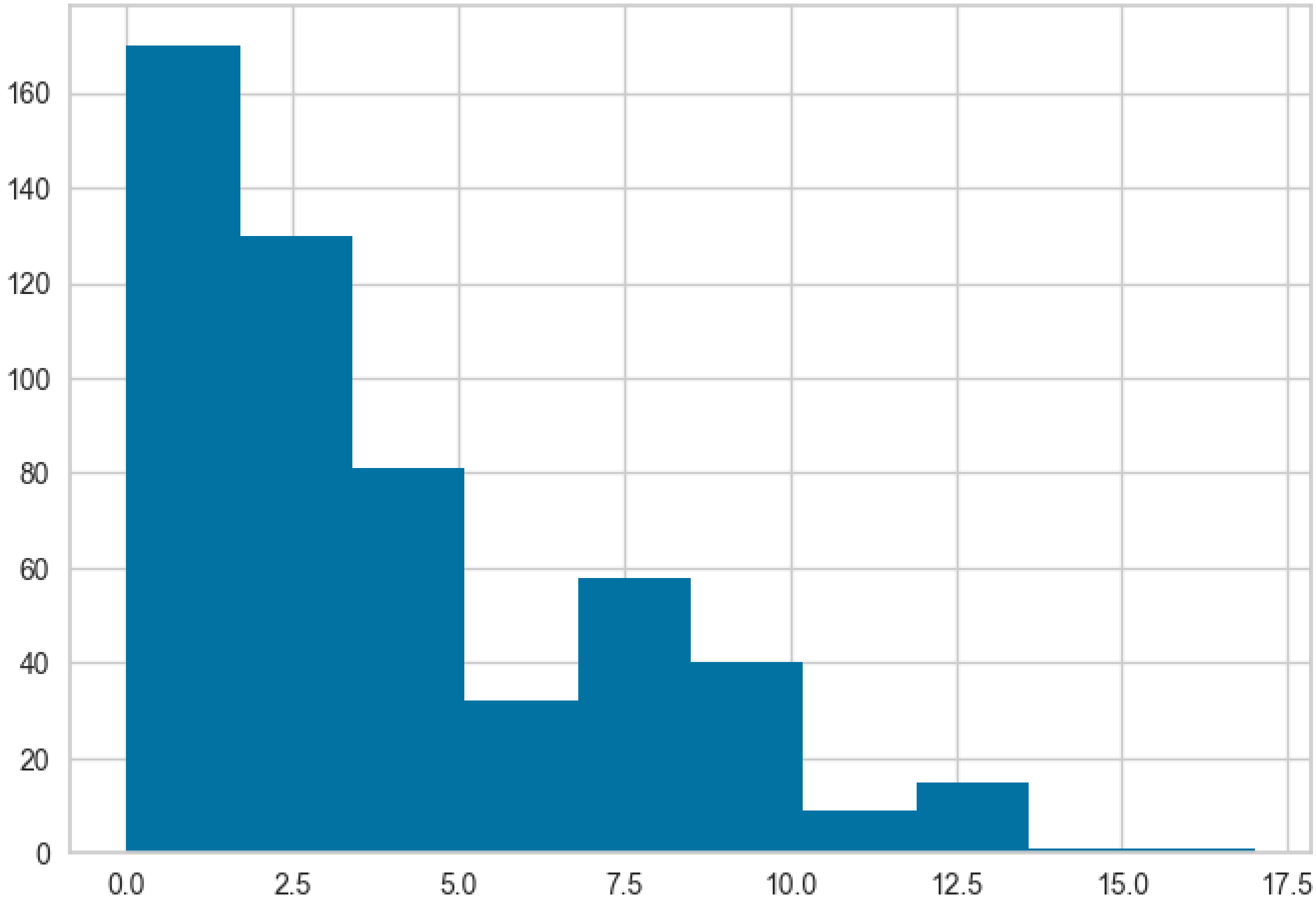
Out[26]: <AxesSubplot:>



Notice that all the values are between 0 and 1 - that is because we passed `normalize=True` in the `setup` function. If you don't remember how it compares to actual data, no problem - we can also access non-transformed values using `get_config` and then compare. See below and notice the range of values on x-axis and compare it with histogram above.

In [27]:
```
get_config('X_train')['Number of times pregnant'].hist()
```

Out[27]:
```
<AxesSubplot:>
```



✅ Compare Models

This function trains and evaluates the performance of all estimators available in the model library using cross-validation. The output of this function is a scoring grid with average cross-validated scores. Metrics evaluated during CV can be accessed using the `get_metrics` function. Custom metrics can be added or removed using `add_metric` and `remove_metric` function.

In [28]:
```
best = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
ridge	Ridge Classifier	0.7708	0.0000	0.5392	0.7353	0.6203	0.4618	0.4744	0.0340
lr	Logistic Regression	0.7689	0.8068	0.4959	0.7614	0.5968	0.4453	0.4673	0.0360
lda	Linear Discriminant Analysis	0.7670	0.8055	0.5550	0.7202	0.6243	0.4594	0.4695	0.0340
svm	SVM - Linear Kernel	0.7521	0.0000	0.5070	0.7363	0.5796	0.4154	0.4398	0.0340
rf	Random Forest Classifier	0.7485	0.7917	0.5336	0.6784	0.5946	0.4164	0.4245	0.1340
nb	Naive Bayes	0.7427	0.7957	0.5702	0.6543	0.6043	0.4156	0.4215	0.0390
catboost	CatBoost Classifier	0.7410	0.7994	0.5278	0.6630	0.5851	0.4005	0.4078	0.0430
gbc	Gradient Boosting Classifier	0.7373	0.7920	0.5550	0.6445	0.5931	0.4013	0.4059	0.0730
ada	Ada Boost Classifier	0.7372	0.7799	0.5275	0.6585	0.5796	0.3926	0.4017	0.0690
et	Extra Trees Classifier	0.7299	0.7788	0.4965	0.6516	0.5596	0.3706	0.3802	0.1330
qda	Quadratic Discriminant Analysis	0.7282	0.7894	0.5281	0.6558	0.5736	0.3785	0.3910	0.0360
lightgbm	Light Gradient Boosting Machine	0.7113	0.7653	0.5181	0.6036	0.5533	0.3427	0.3479	0.0480
knn	K Neighbors Classifier	0.7002	0.7433	0.4860	0.5965	0.5311	0.3142	0.3210	0.0570
dt	Decision Tree Classifier	0.6947	0.6526	0.5137	0.5665	0.5343	0.3103	0.3130	0.0380
xgboost	Extreme Gradient Boosting	0.6853	0.7522	0.4912	0.5620	0.5216	0.2887	0.2922	0.0390
dummy	Dummy Classifier	0.6518	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0380

```
Processing:   0%|          | 0/69 [00:00<?, ?it/s]
```

`compare_models` by default uses all the estimators in model library (all except models with `Turbo=False`) . To see all available models you can use the function `models()`

In [29]:
```
# check available models
models()
```

Out[29]:

	Name	Reference	Turbo
ID			
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
xgboost	Extreme Gradient Boosting	xgboost.sklearn.XGBClassifier	True
lightgbm	Light Gradient Boosting Machine	lightgbm.sklearn.LGBMClassifier	True
catboost	CatBoost Classifier	catboost.core.CatBoostClassifier	True
dummy	Dummy Classifier	sklearn.dummy.DummyClassifier	True

You can use the `include` and `exclude` parameter in the `compare_models` to train only select model or exclude specific models from training by passing the model id's in `exclude` parameter.

In [30]:
```
compare_tree_models = compare_models(include = ['dt', 'rf', 'et', 'gbc', 'xgboost', '
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.7485	0.7917	0.5336	0.6784	0.5946	0.4164	0.4245	0.1200
catboost	CatBoost Classifier	0.7410	0.7994	0.5278	0.6630	0.5851	0.4005	0.4078	0.0410
gbc	Gradient Boosting Classifier	0.7373	0.7920	0.5550	0.6445	0.5931	0.4013	0.4059	0.0780
et	Extra Trees Classifier	0.7299	0.7788	0.4965	0.6516	0.5596	0.3706	0.3802	0.1300
lightgbm	Light Gradient Boosting Machine	0.7113	0.7653	0.5181	0.6036	0.5533	0.3427	0.3479	0.0460
dt	Decision Tree Classifier	0.6947	0.6526	0.5137	0.5665	0.5343	0.3103	0.3130	0.0360
xgboost	Extreme Gradient Boosting	0.6853	0.7522	0.4912	0.5620	0.5216	0.2887	0.2922	0.0420

```
Processing:   0%|          | 0/33 [00:00<?, ?it/s]
```

In [31]:
```
compare_tree_models
```

Out[31]:

```
RandomForestClassifier

RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='sqr
t',
                       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,
                       n_estimators=100, n_jobs=-1, oob_score=False,
                       random_state=123, verbose=0, warm_start=False)
```

The function above has return trained model object as an output. The scoring grid is only displayed and not returned. If you need access to the scoring grid you can use `pull` function to access the dataframe.

In [32]:
```
compare_tree_models_results = pull()
compare_tree_models_results
```

Out[32]:

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.7485	0.7917	0.5336	0.6784	0.5946	0.4164	0.4245	0.120
catboost	CatBoost Classifier	0.7410	0.7994	0.5278	0.6630	0.5851	0.4005	0.4078	0.041
gbc	Gradient Boosting Classifier	0.7373	0.7920	0.5550	0.6445	0.5931	0.4013	0.4059	0.078
et	Extra Trees Classifier	0.7299	0.7788	0.4965	0.6516	0.5596	0.3706	0.3802	0.130
lightgbm	Light Gradient Boosting Machine	0.7113	0.7653	0.5181	0.6036	0.5533	0.3427	0.3479	0.046
dt	Decision Tree Classifier	0.6947	0.6526	0.5137	0.5665	0.5343	0.3103	0.3130	0.036
xgboost	Extreme Gradient Boosting	0.6853	0.7522	0.4912	0.5620	0.5216	0.2887	0.2922	0.042

By default `compare_models` return the single best performing model based on the metric defined in the `sort` parameter. Let's change our code to return 3 top models based on `Recall`.

In [33]:
```
best_recall_models_top3 = compare_models(sort = 'Recall', n_select = 3)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
nb	Naive Bayes	0.7427	0.7957	0.5702	0.6543	0.6043	0.4156	0.4215	0.0430
gbc	Gradient Boosting Classifier	0.7373	0.7920	0.5550	0.6445	0.5931	0.4013	0.4059	0.0710
lda	Linear Discriminant Analysis	0.7670	0.8055	0.5550	0.7202	0.6243	0.4594	0.4695	0.0330
ridge	Ridge Classifier	0.7708	0.0000	0.5392	0.7353	0.6203	0.4618	0.4744	0.0340
rf	Random Forest Classifier	0.7485	0.7917	0.5336	0.6784	0.5946	0.4164	0.4245	0.1190
qda	Quadratic Discriminant Analysis	0.7282	0.7894	0.5281	0.6558	0.5736	0.3785	0.3910	0.0370
catboost	CatBoost Classifier	0.7410	0.7994	0.5278	0.6630	0.5851	0.4005	0.4078	0.0400
ada	Ada Boost Classifier	0.7372	0.7799	0.5275	0.6585	0.5796	0.3926	0.4017	0.0670
lightgbm	Light Gradient Boosting Machine	0.7113	0.7653	0.5181	0.6036	0.5533	0.3427	0.3479	0.0450
dt	Decision Tree Classifier	0.6947	0.6526	0.5137	0.5665	0.5343	0.3103	0.3130	0.0340
svm	SVM - Linear Kernel	0.7521	0.0000	0.5070	0.7363	0.5796	0.4154	0.4398	0.0340
et	Extra Trees Classifier	0.7299	0.7788	0.4965	0.6516	0.5596	0.3706	0.3802	0.1290
lr	Logistic Regression	0.7689	0.8068	0.4959	0.7614	0.5968	0.4453	0.4673	0.0410
xgboost	Extreme Gradient Boosting	0.6853	0.7522	0.4912	0.5620	0.5216	0.2887	0.2922	0.0390
knn	K Neighbors Classifier	0.7002	0.7433	0.4860	0.5965	0.5311	0.3142	0.3210	0.0570
dummy	Dummy Classifier	0.6518	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0550

```
Processing:   0%|          | 0/71 [00:00<?, ?it/s]
```

In [34]:
```
# list of top 3 models by Recall
best_recall_models_top3
```

Out[34]:
```
[GaussianNB(priors=None, var_smoothing=1e-09),
 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),
 LinearDiscriminantAnalysis(covariance_estimator=None, n_components=None,
                            priors=None, shrinkage=None, solver='svd',
                            store_covariance=False, tol=0.0001)]
```

Some other parameters that you might find very useful in `compare_models` are:

- fold
- cross_validation
- budget_time
- errors
- probability_threshold
- parallel

You can check the docstring of the function for more info.

In [35]:
```
# help(compare_models)
```

✅ Set Custom Metrics

In [36]:
```
# check available metrics used in CV
get_metrics()
```

Out[36]:

ID	Name	Display Name	Score Function	
acc	Accuracy	Accuracy	<function accuracy_score at 0x000002E242711280>	
auc	AUC	AUC	<function roc_auc_score at 0x000002E24270B0D0>	make_scor needs
recall	Recall	Recall	<pycaret.internal.metrics.BinaryMulticlassScor...	make_sc av
precision	Precision	Prec.	<pycaret.internal.metrics.BinaryMulticlassScor...	make_scorer av
f1	F1	F1	<pycaret.internal.metrics.BinaryMulticlassScor...	make av
kappa	Kappa	Kappa	<function cohen_kappa_score at 0x000002E242711...	make_scorer(coh
mcc	MCC	MCC	<function matthews_corrcoef at 0x000002E242711...	make_scorer(ma

In [37]:
```
# create a custom function
import numpy as np

def custom_metric(y, y_pred):
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```python
    tp = np.where((y_pred==1) & (y==1), (100), 0)
    fp = np.where((y_pred==1) & (y==0), -5, 0)
    return np.sum([tp,fp])

# add metric to PyCaret
add_metric('custom_metric', 'Custom Metric', custom_metric)
```

Out[37]:

```
Name                                                    Custom Metric
Display Name                                            Custom Metric
Score Function       <function custom_metric at 0x000002E24B0EA430>
Scorer                                      make_scorer(custom_metric)
Target                                                           pred
Args                                                               {}
Greater is Better                                                True
Multiclass                                                       True
Custom                                                           True
Name: custom_metric, dtype: object
```

In [38]:

```python
# now let's run compare_models again
compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	Custom Metric
ridge	Ridge Classifier	0.7708	0.0000	0.5392	0.7353	0.6203	0.4618	0.4744	991.5000
lr	Logistic Regression	0.7689	0.8068	0.4959	0.7614	0.5968	0.4453	0.4673	915.0000
lda	Linear Discriminant Analysis	0.7670	0.8055	0.5550	0.7202	0.6243	0.4594	0.4695	1019.000(
svm	SVM - Linear Kernel	0.7521	0.0000	0.5070	0.7363	0.5796	0.4154	0.4398	929.5000
rf	Random Forest Classifier	0.7485	0.7917	0.5336	0.6784	0.5946	0.4164	0.4245	976.0000
nb	Naive Bayes	0.7427	0.7957	0.5702	0.6543	0.6043	0.4156	0.4215	1041.000(
catboost	CatBoost Classifier	0.7410	0.7994	0.5278	0.6630	0.5851	0.4005	0.4078	964.5000
gbc	Gradient Boosting Classifier	0.7373	0.7920	0.5550	0.6445	0.5931	0.4013	0.4059	1011.000(
ada	Ada Boost Classifier	0.7372	0.7799	0.5275	0.6585	0.5796	0.3926	0.4017	963.5000
et	Extra Trees Classifier	0.7299	0.7788	0.4965	0.6516	0.5596	0.3706	0.3802	904.5000
qda	Quadratic Discriminant Analysis	0.7282	0.7894	0.5281	0.6558	0.5736	0.3785	0.3910	961.0000
lightgbm	Light Gradient Boosting Machine	0.7113	0.7653	0.5181	0.6036	0.5533	0.3427	0.3479	937.5000
knn	K Neighbors Classifier	0.7002	0.7433	0.4860	0.5965	0.5311	0.3142	0.3210	877.5000
dt	Decision Tree Classifier	0.6947	0.6526	0.5137	0.5665	0.5343	0.3103	0.3130	923.5000
xgboost	Extreme Gradient Boosting	0.6853	0.7522	0.4912	0.5620	0.5216	0.2887	0.2922	883.0000
dummy	Dummy Classifier	0.6518	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

```
Processing:   0%|          | 0/69 [00:00<?, ?it/s]
```

Out[38]:

```
RidgeClassifier
RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,
                max_iter=None, normalize='deprecated', positive=False,
                random_state=123, solver='auto', tol=0.001)
```

In [39]:

```
# remove custom metric
remove_metric('custom_metric')
```

✅ Experiment Logging

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

PyCaret integrates with many different type of experiment loggers (default = 'mlflow'). To turn on experiment tracking in PyCaret you can set `log_experiment` and `experiment_name` parameter. It will automatically track all the metrics, hyperparameters, and artifacts based on the defined logger.

In [40]:
```
# from pycaret.classification import *
# s = setup(data, target = 'Class variable', log_experiment='mlflow', experiment_name
```

In [41]:
```
# compare models
# best = compare_models()
```

In [42]:
```
# start mlflow server on localhost:5000
# !mlflow ui
```

By default PyCaret uses `MLFlow` logger that can be changed using `log_experiment` parameter. Following loggers are available:

```
- mlflow
- wandb
- comet_ml
- dagshub
```

Other logging related parameters that you may find useful are:

- experiment_custom_tags
- log_plots
- log_data
- log_profile

For more information check out the docstring of the `setup` function.

In [43]:
```
# help(setup)
```

✅ Create Model

This function trains and evaluates the performance of a given estimator using cross-validation. The output of this function is a scoring grid with CV scores by fold. Metrics evaluated during CV can be accessed using the `get_metrics` function. Custom metrics can be added or removed using `add_metric` and `remove_metric` function. All the available models can be accessed using the models function.

In [44]:
```
# check all the available models
models()
```

Out[44]:

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
xgboost	Extreme Gradient Boosting	xgboost.sklearn.XGBClassifier	True
lightgbm	Light Gradient Boosting Machine	lightgbm.sklearn.LGBMClassifier	True
catboost	CatBoost Classifier	catboost.core.CatBoostClassifier	True
dummy	Dummy Classifier	sklearn.dummy.DummyClassifier	True

In [45]:

```
# train logistic regression with default fold=10
lr = create_model('lr')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8148	0.9023	0.5789	0.8462	0.6875	0.5624	0.5828
1	0.8333	0.7970	0.6316	0.8571	0.7273	0.6112	0.6260
2	0.8519	0.9383	0.6316	0.9231	0.7500	0.6499	0.6736
3	0.7222	0.7759	0.4211	0.6667	0.5161	0.3350	0.3524
4	0.8333	0.9083	0.5789	0.9167	0.7097	0.6010	0.6322
5	0.6852	0.6737	0.4211	0.5714	0.4848	0.2656	0.2720
6	0.7222	0.7820	0.4737	0.6429	0.5455	0.3520	0.3605
7	0.7547	0.8460	0.3333	0.8571	0.4800	0.3579	0.4263
8	0.7358	0.6952	0.4444	0.6667	0.5333	0.3592	0.3736
9	0.7358	0.7492	0.4444	0.6667	0.5333	0.3592	0.3736
Mean	0.7689	0.8068	0.4959	0.7614	0.5968	0.4453	0.4673
Std	0.0557	0.0857	0.0970	0.1236	0.1024	0.1353	0.1379

```
Processing:   0%|          | 0/4 [00:00<?, ?it/s]
```

The function above has return trained model object as an output. The scoring grid is only displayed and not returned. If you need access to the scoring grid you can use `pull` function to access the dataframe.

In [46]:
```python
lr_results = pull()
print(type(lr_results))
lr_results
```

```
<class 'pandas.core.frame.DataFrame'>
```

Out[46]:

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8148	0.9023	0.5789	0.8462	0.6875	0.5624	0.5828
1	0.8333	0.7970	0.6316	0.8571	0.7273	0.6112	0.6260
2	0.8519	0.9383	0.6316	0.9231	0.7500	0.6499	0.6736
3	0.7222	0.7759	0.4211	0.6667	0.5161	0.3350	0.3524
4	0.8333	0.9083	0.5789	0.9167	0.7097	0.6010	0.6322
5	0.6852	0.6737	0.4211	0.5714	0.4848	0.2656	0.2720
6	0.7222	0.7820	0.4737	0.6429	0.5455	0.3520	0.3605
7	0.7547	0.8460	0.3333	0.8571	0.4800	0.3579	0.4263
8	0.7358	0.6952	0.4444	0.6667	0.5333	0.3592	0.3736
9	0.7358	0.7492	0.4444	0.6667	0.5333	0.3592	0.3736
Mean	0.7689	0.8068	0.4959	0.7614	0.5968	0.4453	0.4673
Std	0.0557	0.0857	0.0970	0.1236	0.1024	0.1353	0.1379

In [47]:
```python
# train logistic regression with fold=3
lr = create_model('lr', fold=3)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8101	0.8526	0.5714	0.8372	0.6792	0.5510	0.5713
1	0.7486	0.7921	0.5000	0.6889	0.5794	0.4065	0.4172
2	0.7486	0.7804	0.4194	0.7429	0.5361	0.3815	0.4108
Mean	0.7691	0.8084	0.4969	0.7563	0.5983	0.4464	0.4664
Std	0.0290	0.0317	0.0621	0.0613	0.0599	0.0747	0.0742

```
Processing:   0%|          | 0/4 [00:00<?, ?it/s]
```

In [48]:
```
# train logistic regression with specific model parameters
create_model('lr', C = 0.5, l1_ratio = 0.15)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7963	0.8872	0.4737	0.9000	0.6207	0.4992	0.5472
1	0.8148	0.8030	0.5789	0.8462	0.6875	0.5624	0.5828
2	0.8519	0.9353	0.5789	1.0000	0.7333	0.6406	0.6865
3	0.7037	0.7684	0.3684	0.6364	0.4667	0.2812	0.3013
4	0.8519	0.9038	0.5789	1.0000	0.7333	0.6406	0.6865
5	0.6852	0.6737	0.4211	0.5714	0.4848	0.2656	0.2720
6	0.7222	0.7624	0.4737	0.6429	0.5455	0.3520	0.3605
7	0.7547	0.8302	0.3333	0.8571	0.4800	0.3579	0.4263
8	0.7358	0.6952	0.3333	0.7500	0.4615	0.3193	0.3654
9	0.7547	0.7587	0.4444	0.7273	0.5517	0.3961	0.4189
Mean	0.7671	0.8018	0.4585	0.7931	0.5765	0.4315	0.4647
Std	0.0561	0.0828	0.0922	0.1437	0.1039	0.1360	0.1440

```
Processing:   0%|          | 0/4 [00:00<?, ?it/s]
```

Out[48]:

▾ LogisticRegression

```
LogisticRegression(C=0.5, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=0.15, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=123, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [49]:
```
# train lr and return train score as well alongwith CV
create_model('lr', return_train_score=True)
```

Split	Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
CV-Train	0	0.7660	0.8146	0.5000	0.7434	0.5979	0.4417	0.4589
	1	0.7764	0.8259	0.5000	0.7778	0.6087	0.4623	0.4845
	2	0.7702	0.8138	0.5000	0.7568	0.6022	0.4499	0.4690
	3	0.7909	0.8296	0.5417	0.7913	0.6431	0.5025	0.5205
	4	0.7764	0.8142	0.5060	0.7727	0.6115	0.4640	0.4845
	5	0.7888	0.8403	0.5417	0.7845	0.6408	0.4983	0.5154
	6	0.7826	0.8242	0.5238	0.7788	0.6263	0.4812	0.5000
	7	0.7748	0.8185	0.5148	0.7632	0.6148	0.4641	0.4820
	8	0.7810	0.8387	0.5266	0.7739	0.6268	0.4796	0.4974
	9	0.7851	0.8340	0.5266	0.7876	0.6312	0.4879	0.5076
CV-Val	0	0.8148	0.9023	0.5789	0.8462	0.6875	0.5624	0.5828
	1	0.8333	0.7970	0.6316	0.8571	0.7273	0.6112	0.6260
	2	0.8519	0.9383	0.6316	0.9231	0.7500	0.6499	0.6736
	3	0.7222	0.7759	0.4211	0.6667	0.5161	0.3350	0.3524
	4	0.8333	0.9083	0.5789	0.9167	0.7097	0.6010	0.6322
	5	0.6852	0.6737	0.4211	0.5714	0.4848	0.2656	0.2720
	6	0.7222	0.7820	0.4737	0.6429	0.5455	0.3520	0.3605
	7	0.7547	0.8460	0.3333	0.8571	0.4800	0.3579	0.4263
	8	0.7358	0.6952	0.4444	0.6667	0.5333	0.3592	0.3736
	9	0.7358	0.7492	0.4444	0.6667	0.5333	0.3592	0.3736
CV-Train	Mean	0.7792	0.8254	0.5181	0.7730	0.6203	0.4731	0.4920
	Std	0.0075	0.0096	0.0156	0.0141	0.0149	0.0191	0.0188
CV-Val	Mean	0.7689	0.8068	0.4959	0.7614	0.5968	0.4453	0.4673
	Std	0.0557	0.0857	0.0970	0.1236	0.1024	0.1353	0.1379
Train	nan	0.7765	0.8248	0.5187	0.7638	0.6178	0.4680	0.4855

```
Processing:   0%|          | 0/4 [00:00<?, ?it/s]
```

Out[49]:

```
▾ LogisticRegression
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=123, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [50]:

```python
# change the probability threshold of classifier from 0.5 to 0.66
create_model('lr', probability_threshold = 0.66)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7222	0.9023	0.2105	1.0000	0.3478	0.2569	0.3839
1	0.7407	0.7970	0.2632	1.0000	0.4167	0.3165	0.4336
2	0.7037	0.9383	0.1579	1.0000	0.2727	0.1955	0.3292
3	0.7037	0.7759	0.2105	0.8000	0.3333	0.2188	0.2998
4	0.7037	0.9083	0.1579	1.0000	0.2727	0.1955	0.3292
5	0.6852	0.6737	0.2105	0.6667	0.3200	0.1818	0.2331
6	0.7222	0.7820	0.3158	0.7500	0.4444	0.2981	0.3477
7	0.6981	0.8460	0.1111	1.0000	0.2000	0.1417	0.2761
8	0.7170	0.6952	0.2222	0.8000	0.3478	0.2348	0.3138
9	0.6981	0.7492	0.1667	0.7500	0.2727	0.1703	0.2476
Mean	0.7095	0.8068	0.2026	0.8767	0.3228	0.2210	0.3194
Std	0.0152	0.0857	0.0554	0.1281	0.0690	0.0531	0.0575

```
Processing:   0%|          | 0/4 [00:00<?, ?it/s]
```

Out[50]:

```
▸ CustomProbabilityThresholdClassifier
  ▸ classifier: LogisticRegression
      ▸ LogisticRegression
```

Some other parameters that you might find very useful in `create_model` are:

- cross_validation
- engine
- fit_kwargs
- groups

You can check the docstring of the function for more info.

In [51]:

```python
# help(create_model)
```

✅ Tune Model

This function tunes the hyperparameters of the model. The output of this function is a scoring grid with cross-validated scores by fold. The best model is selected based on the metric defined in optimize parameter. Metrics evaluated during cross-validation can be accessed using the `get_metrics` function. Custom metrics can be added or removed using `add_metric` and `remove_metric` function.

In [52]:

```python
# train a dt model with default params
dt = create_model('dt')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7222	0.6774	0.5263	0.6250	0.5714	0.3682	0.3711
1	0.7222	0.7015	0.6316	0.6000	0.6154	0.3982	0.3985
2	0.7407	0.7038	0.5789	0.6471	0.6111	0.4176	0.4190
3	0.5926	0.5053	0.2105	0.3636	0.2667	0.0116	0.0125
4	0.7778	0.7684	0.7368	0.6667	0.7000	0.5242	0.5259
5	0.6296	0.5940	0.4737	0.4737	0.4737	0.1880	0.1880
6	0.6296	0.5699	0.3684	0.4667	0.4118	0.1469	0.1491
7	0.8302	0.7770	0.6111	0.8462	0.7097	0.5940	0.6098
8	0.6604	0.6079	0.4444	0.5000	0.4706	0.2219	0.2227
9	0.6415	0.6206	0.5556	0.4762	0.5128	0.2319	0.2336
Mean	0.6947	0.6526	0.5137	0.5665	0.5343	0.3103	0.3130
Std	0.0720	0.0834	0.1410	0.1310	0.1292	0.1714	0.1739

```
Processing:   0%|          | 0/4 [00:00<?, ?it/s]
```

In [53]:
```
# tune hyperparameters of dt
tuned_dt = tune_model(dt)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8519	0.8135	0.6842	0.8667	0.7647	0.6588	0.6686
1	0.7593	0.6940	0.4737	0.7500	0.5806	0.4236	0.4456
2	0.7593	0.7782	0.8421	0.6154	0.7111	0.5132	0.5318
3	0.7037	0.6511	0.4737	0.6000	0.5294	0.3175	0.3223
4	0.8333	0.7632	0.5263	1.0000	0.6897	0.5902	0.6470
5	0.6296	0.5820	0.4211	0.4706	0.4444	0.1680	0.1685
6	0.7222	0.6654	0.4737	0.6429	0.5455	0.3520	0.3605
7	0.7358	0.6246	0.2778	0.8333	0.4167	0.2973	0.3725
8	0.6604	0.5675	0.2778	0.5000	0.3571	0.1512	0.1633
9	0.7170	0.6643	0.5000	0.6000	0.5455	0.3424	0.3454
Mean	0.7372	0.6804	0.4950	0.6879	0.5585	0.3814	0.4026
Std	0.0653	0.0782	0.1608	0.1611	0.1258	0.1587	0.1654

```
Processing:   0%|          | 0/7 [00:00<?, ?it/s]
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

Metric to optimize can be defined in `optimize` parameter (default = 'Accuracy'). Also, a custom tuned grid can be passed with `custom_grid` parameter.

In [54]:
```
dt
```

Out[54]:

DecisionTreeClassifier

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=Non
e,
                       min_impurity_decrease=0.0, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       random_state=123, splitter='best')
```

In [55]:

```
# define tuning grid
dt_grid = {'max_depth' : [None, 2, 4, 6, 8, 10, 12]}

# tune model with custom grid and metric = F1
tuned_dt = tune_model(dt, custom_grid = dt_grid, optimize = 'F1')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7593	0.8008	0.7368	0.6364	0.6829	0.4906	0.4940
1	0.6667	0.7444	0.5263	0.5263	0.5263	0.2692	0.2692
2	0.7593	0.8241	0.5263	0.7143	0.6061	0.4384	0.4490
3	0.6667	0.6293	0.4211	0.5333	0.4706	0.2322	0.2357
4	0.8333	0.8962	0.6842	0.8125	0.7429	0.6209	0.6259
5	0.6667	0.6534	0.5789	0.5238	0.5500	0.2863	0.2872
6	0.6296	0.6759	0.3158	0.4615	0.3750	0.1248	0.1293
7	0.7736	0.7698	0.6111	0.6875	0.6471	0.4812	0.4830
8	0.6415	0.6817	0.4444	0.4706	0.4571	0.1899	0.1900
9	0.7547	0.7437	0.6111	0.6471	0.6286	0.4457	0.4461
Mean	0.7151	0.7419	0.5456	0.6013	0.5687	0.3579	0.3610
Std	0.0653	0.0796	0.1203	0.1100	0.1078	0.1508	0.1517

```
Processing:   0%|          | 0/7 [00:00<?, ?it/s]
Fitting 10 folds for each of 7 candidates, totalling 70 fits
```

In [56]:

```
# to access the tuner object you can set return_tuner = True
tuned_dt, tuner = tune_model(dt, return_tuner=True)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8519	0.8135	0.6842	0.8667	0.7647	0.6588	0.6686
1	0.7593	0.6940	0.4737	0.7500	0.5806	0.4236	0.4456
2	0.7593	0.7782	0.8421	0.6154	0.7111	0.5132	0.5318
3	0.7037	0.6511	0.4737	0.6000	0.5294	0.3175	0.3223
4	0.8333	0.7632	0.5263	1.0000	0.6897	0.5902	0.6470
5	0.6296	0.5820	0.4211	0.4706	0.4444	0.1680	0.1685
6	0.7222	0.6654	0.4737	0.6429	0.5455	0.3520	0.3605
7	0.7358	0.6246	0.2778	0.8333	0.4167	0.2973	0.3725
8	0.6604	0.5675	0.2778	0.5000	0.3571	0.1512	0.1633
9	0.7170	0.6643	0.5000	0.6000	0.5455	0.3424	0.3454
Mean	0.7372	0.6804	0.4950	0.6879	0.5585	0.3814	0.4026
Std	0.0653	0.0782	0.1608	0.1611	0.1258	0.1587	0.1654

```
Processing:   0%|          | 0/7 [00:00<?, ?it/s]
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

In [57]:
```
# model object
tuned_dt
```

Out[57]:

▾ DecisionTreeClassifier

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=1, max_features=1.0, max_leaf_nodes=None,
                       min_impurity_decrease=0.01, min_samples_leaf=6,
                       min_samples_split=5, min_weight_fraction_leaf=0.0,
                       random_state=123, splitter='best')
```

In [58]:
```
# tuner object
tuner
```

Out[58]:

```
▸ RandomizedSearchCV
▸ estimator: Pipeline
▸ clean_column_names: TransformerWrapper
▸ transformer: CleanColumnNames
▸ CleanColumnNames
▸ numerical_imputer: TransformerWrapper
▸ transformer: SimpleImputer
▸ SimpleImputer
▸ categorical_imputer: TransformerWrapper
▸ transformer: SimpleImputer
▸ SimpleImputer
▸ normalize: TransformerWrapper
▸ transformer: MinMaxScaler
▸ MinMaxScaler
▸ DecisionTreeClassifier
```

The default search algorithm is `RandomizedSearchCV` from `sklearn`. This can be changed by using `search_library` and `search_algorithm` parameter.

In [59]:

```python
# tune dt using optuna
tuned_dt = tune_model(dt, search_library = 'optuna')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7593	0.7820	0.5263	0.7143	0.6061	0.4384	0.4490
1	0.7778	0.7895	0.6316	0.7059	0.6667	0.5008	0.5025
2	0.7222	0.7880	0.3684	0.7000	0.4828	0.3170	0.3476
3	0.6852	0.5662	0.4211	0.5714	0.4848	0.2656	0.2720
4	0.7963	0.8233	0.6842	0.7222	0.7027	0.5479	0.5484
5	0.6667	0.6805	0.5263	0.5263	0.5263	0.2692	0.2692
6	0.6852	0.6940	0.4211	0.5714	0.4848	0.2656	0.2720
7	0.8302	0.8508	0.6667	0.8000	0.7273	0.6055	0.6108
8	0.6604	0.6389	0.6111	0.5000	0.5500	0.2816	0.2853
9	0.6415	0.6849	0.4444	0.4706	0.4571	0.1899	0.1900
Mean	0.7225	0.7298	0.5301	0.6282	0.5689	0.3681	0.3747
Std	0.0615	0.0859	0.1080	0.1073	0.0949	0.1356	0.1352

```
Processing:   0%|          | 0/7 [00:00<?, ?it/s]
[I 2023-02-16 14:23:58,902] Searching the best hyperparameters using 537 samples...
[I 2023-02-16 14:24:07,369] Finished hyperparemeter search!
```

 ch_library and search_algorithm please check the docstring. Some other parameters that you might find very useful in `tune_model` are:

- choose_better
- n_iter
- early_stopping
- groups

You can check the docstring of the function for more info.

In [60]:
```
# help(tune_model)
```

✅ Ensemble Model

This function ensembles a given estimator. The output of this function is a scoring grid with CV scores by fold. Metrics evaluated during CV can be accessed using the `get_metrics` function. Custom metrics can be added or removed using `add_metric` and `remove_metric` function.

In [61]:
```
# ensemble with bagging
ensemble_model(dt, method = 'Bagging')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7407	0.8383	0.5263	0.6667	0.5882	0.4028	0.4088
1	0.7963	0.7797	0.7368	0.7000	0.7179	0.5587	0.5591
2	0.7593	0.7669	0.4737	0.7500	0.5806	0.4236	0.4456
3	0.7222	0.7842	0.5263	0.6250	0.5714	0.3682	0.3711
4	0.8148	0.8421	0.7368	0.7368	0.7368	0.5940	0.5940
5	0.6852	0.6759	0.4211	0.5714	0.4848	0.2656	0.2720
6	0.7037	0.7677	0.5263	0.5882	0.5556	0.3344	0.3355
7	0.7925	0.8405	0.4444	0.8889	0.5926	0.4734	0.5245
8	0.6792	0.6659	0.5000	0.5294	0.5143	0.2751	0.2754
9	0.6792	0.6508	0.3333	0.5455	0.4138	0.2103	0.2224
Mean	0.7373	0.7612	0.5225	0.6602	0.5756	0.3906	0.4009
Std	0.0488	0.0695	0.1212	0.1060	0.0924	0.1195	0.1221

```
Processing:   0%|          | 0/6 [00:00<?, ?it/s]
```

Out[61]:
```
▸ BaggingClassifier
▸ base_estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

In [62]:
```
# ensemble with boosting
ensemble_model(dt, method = 'Boosting')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7222	0.6895	0.5789	0.6111	0.5946	0.3836	0.3839
1	0.7222	0.6774	0.5263	0.6250	0.5714	0.3682	0.3711
2	0.7593	0.7421	0.6842	0.6500	0.6667	0.4785	0.4788
3	0.6111	0.5436	0.3158	0.4286	0.3636	0.0928	0.0950
4	0.8148	0.8211	0.8421	0.6957	0.7619	0.6126	0.6201
5	0.5926	0.5654	0.4737	0.4286	0.4500	0.1278	0.1282
6	0.6667	0.6226	0.4737	0.5294	0.5000	0.2512	0.2520
7	0.7925	0.7484	0.6111	0.7333	0.6667	0.5178	0.5223
8	0.6604	0.6214	0.5000	0.5000	0.5000	0.2429	0.2429
9	0.6792	0.6357	0.5000	0.5294	0.5143	0.2751	0.2754
Mean	0.7021	0.6667	0.5506	0.5731	0.5589	0.3350	0.3370
Std	0.0698	0.0820	0.1342	0.1008	0.1117	0.1598	0.1613

```
Processing:   0%|          | 0/6 [00:00<?, ?it/s]
```

Out[62]:

```
AdaBoostClassifier
base_estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

Some other parameters that you might find very useful in `ensemble_model` are:

- choose_better
- n_estimators
- groups
- fit_kwargs
- probability_threshold
- return_train_score

You can check the docstring of the function for more info.

In [63]:

```
# help(ensemble_model)
```

✅ Blend Models

This function trains a Soft Voting / Majority Rule classifier for select models passed in the estimator_list parameter. The output of this function is a scoring grid with CV scores by fold. Metrics evaluated during CV can be accessed using the `get_metrics` function. Custom metrics can be added or removed using `add_metric` and `remove_metric` function.

In [64]:

```
# top 3 models based on recall
best_recall_models_top3
```

Out[64]:
```
[GaussianNB(priors=None, var_smoothing=1e-09),
 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),
 LinearDiscriminantAnalysis(covariance_estimator=None, n_components=None,
                            priors=None, shrinkage=None, solver='svd',
                            store_covariance=False, tol=0.0001)]
```

In [65]:
```
# blend top 3 models
blend_models(best_recall_models_top3)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7963	0.8932	0.6842	0.7222	0.7027	0.5479	0.5484
1	0.7778	0.8120	0.6316	0.7059	0.6667	0.5008	0.5025
2	0.8704	0.9338	0.6842	0.9286	0.7879	0.6976	0.7145
3	0.7037	0.7865	0.4737	0.6000	0.5294	0.3175	0.3223
4	0.8704	0.8962	0.6842	0.9286	0.7879	0.6976	0.7145
5	0.7037	0.6692	0.4737	0.6000	0.5294	0.3175	0.3223
6	0.7407	0.7805	0.6842	0.6190	0.6500	0.4449	0.4463
7	0.7736	0.8667	0.4444	0.8000	0.5714	0.4342	0.4688
8	0.6604	0.6889	0.4444	0.5000	0.4706	0.2219	0.2227
9	0.6981	0.7286	0.4444	0.5714	0.5000	0.2886	0.2933
Mean	0.7595	0.8056	0.5649	0.6976	0.6196	0.4469	0.4555
Std	0.0683	0.0868	0.1103	0.1407	0.1104	0.1575	0.1616

```
Processing:   0%|          | 0/6 [00:00<?, ?it/s]
```

Out[65]:

VotingClassifier

Naive Bayes	Gradient Boosting Classifier	Linear Discriminant Analysis
GaussianNB	GradientBoostingClassifier	LinearDiscriminantAnalysis

Some other parameters that you might find very useful in `blend_models` are:

- choose_better
- method
- weights
- fit_kwargs
- probability_threshold
- return_train_score

You can check the docstring of the function for more info.

In [66]:
```
# help(blend_models)
```

✅ Stack Models

This function trains a meta-model over select estimators passed in the estimator_list parameter. The output of this function is a scoring grid with CV scores by fold. Metrics evaluated during CV can be accessed using the `get_metrics` function. Custom metrics can be added or removed using `add_metric` and `remove_metric` function.

In [67]:
```
# stack models
stack_models(best_recall_models_top3)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8148	0.9023	0.6316	0.8000	0.7059	0.5735	0.5820
1	0.7963	0.7970	0.6316	0.7500	0.6857	0.5367	0.5410
2	0.8704	0.9233	0.6842	0.9286	0.7879	0.6976	0.7145
3	0.7037	0.7835	0.4737	0.6000	0.5294	0.3175	0.3223
4	0.8519	0.8992	0.6316	0.9231	0.7500	0.6499	0.6736
5	0.6852	0.6722	0.4211	0.5714	0.4848	0.2656	0.2720
6	0.7222	0.7910	0.5263	0.6250	0.5714	0.3682	0.3711
7	0.7547	0.8667	0.3889	0.7778	0.5185	0.3776	0.4184
8	0.6981	0.6810	0.4444	0.5714	0.5000	0.2886	0.2933
9	0.7358	0.7190	0.5000	0.6429	0.5625	0.3775	0.3836
Mean	0.7633	0.8035	0.5333	0.7190	0.6096	0.4453	0.4572
Std	0.0628	0.0879	0.0989	0.1300	0.1061	0.1479	0.1514

```
Processing:   0%|          | 0/6 [00:00<?, ?it/s]
```

Out[67]:

StackingClassifier

Naive Bayes | **Gradient Boosting Classifier** | **Linear Discriminant Analysis**

GaussianNB | GradientBoostingClassifier | LinearDiscriminantAnalysis

final_estimator

LogisticRegression

Some other parameters that you might find very useful in `stack_models` are:

- choose_better
- meta_model
- method
- restack
- probability_threshold
- return_train_score

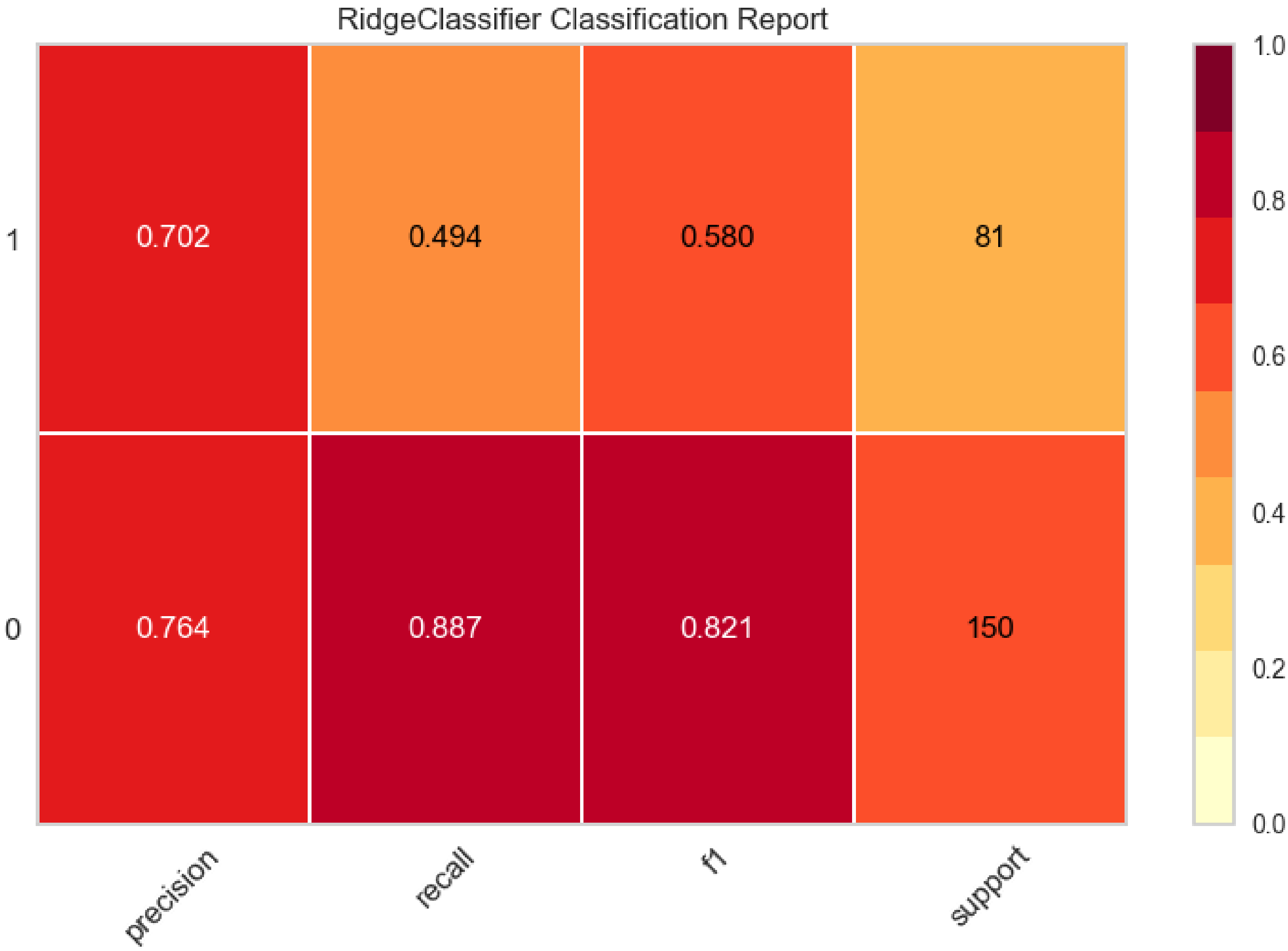
You can check the docstring of the function for more info.

In [68]:
```
# help(stack_models)
```

✅ Plot Model

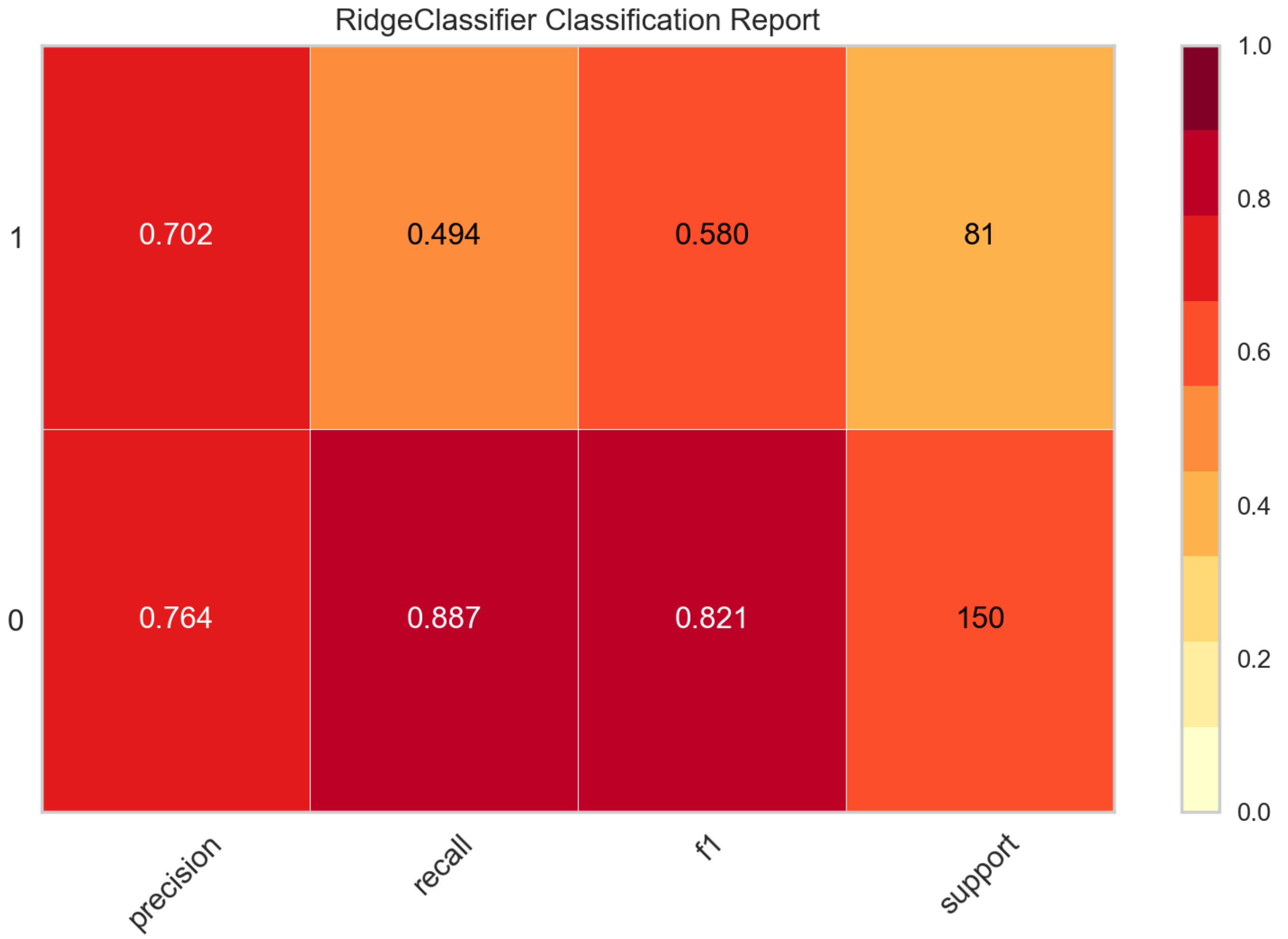
This function analyzes the performance of a trained model on the hold-out set. It may require re-training the model in certain cases.

In [69]:
```
# plot class report
plot_model(best, plot = 'class_report')
```



In [70]:
```
# to control the scale of plot
plot_model(best, plot = 'class_report', scale = 2)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [71]:
```
# to save the plot
plot_model(best, plot = 'class_report', save=True)
```

Out[71]: 'Class Report.png'

Some other parameters that you might find very useful in `plot_model` are:

- fit_kwargs
- plot_kwargs
- groups
- display_format

You can check the docstring of the function for more info.

In [72]:
```
# help(plot_model)
```

✅ Interpret Model

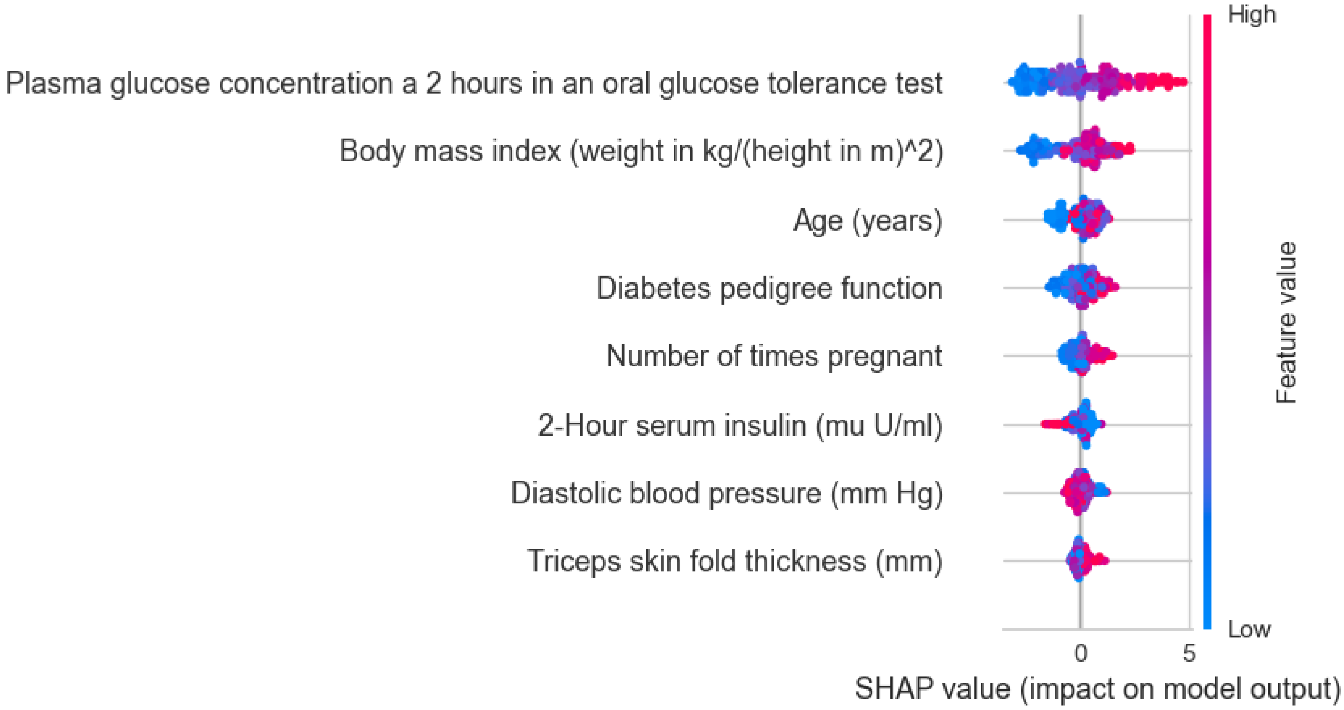
This function analyzes the predictions generated from a trained model. Most plots in this function are implemented based on the SHAP (Shapley Additive exPlanations). For more info on this, please see https://shap.readthedocs.io/en/latest/

In [73]:
```
# train lightgbm model
lightgbm = create_model('lightgbm')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7222	0.8376	0.4737	0.6429	0.5455	0.3520	0.3605
1	0.7593	0.7865	0.7368	0.6364	0.6829	0.4906	0.4940
2	0.6667	0.8301	0.4211	0.5333	0.4706	0.2322	0.2357
3	0.6852	0.7639	0.5263	0.5556	0.5405	0.3014	0.3016
4	0.7778	0.8406	0.6842	0.6842	0.6842	0.5128	0.5128
5	0.6481	0.6887	0.3684	0.5000	0.4242	0.1792	0.1835
6	0.7407	0.7338	0.5263	0.6667	0.5882	0.4028	0.4088
7	0.8491	0.8603	0.6111	0.9167	0.7333	0.6339	0.6592
8	0.6604	0.6952	0.5000	0.5000	0.5000	0.2429	0.2429
9	0.6038	0.6159	0.3333	0.4000	0.3636	0.0794	0.0801
Mean	0.7113	0.7653	0.5181	0.6036	0.5533	0.3427	0.3479
Std	0.0689	0.0766	0.1235	0.1346	0.1141	0.1610	0.1655

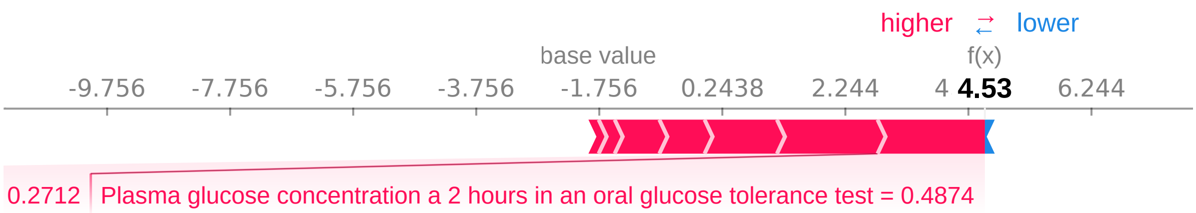
```
Processing:   0%|          | 0/4 [00:00<?, ?it/s]
```

In [74]:
```
# interpret summary model
interpret_model(lightgbm, plot = 'summary')
```



In [75]:
```
# reason plot for test set observation 1
interpret_model(lightgbm, plot = 'reason', observation = 1)
```

Out[75]:



Some other parameters that you might find very useful in `interpret_model` are:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

- plot
- feature
- use_train_data
- X_new_sample
- y_new_sample
- save

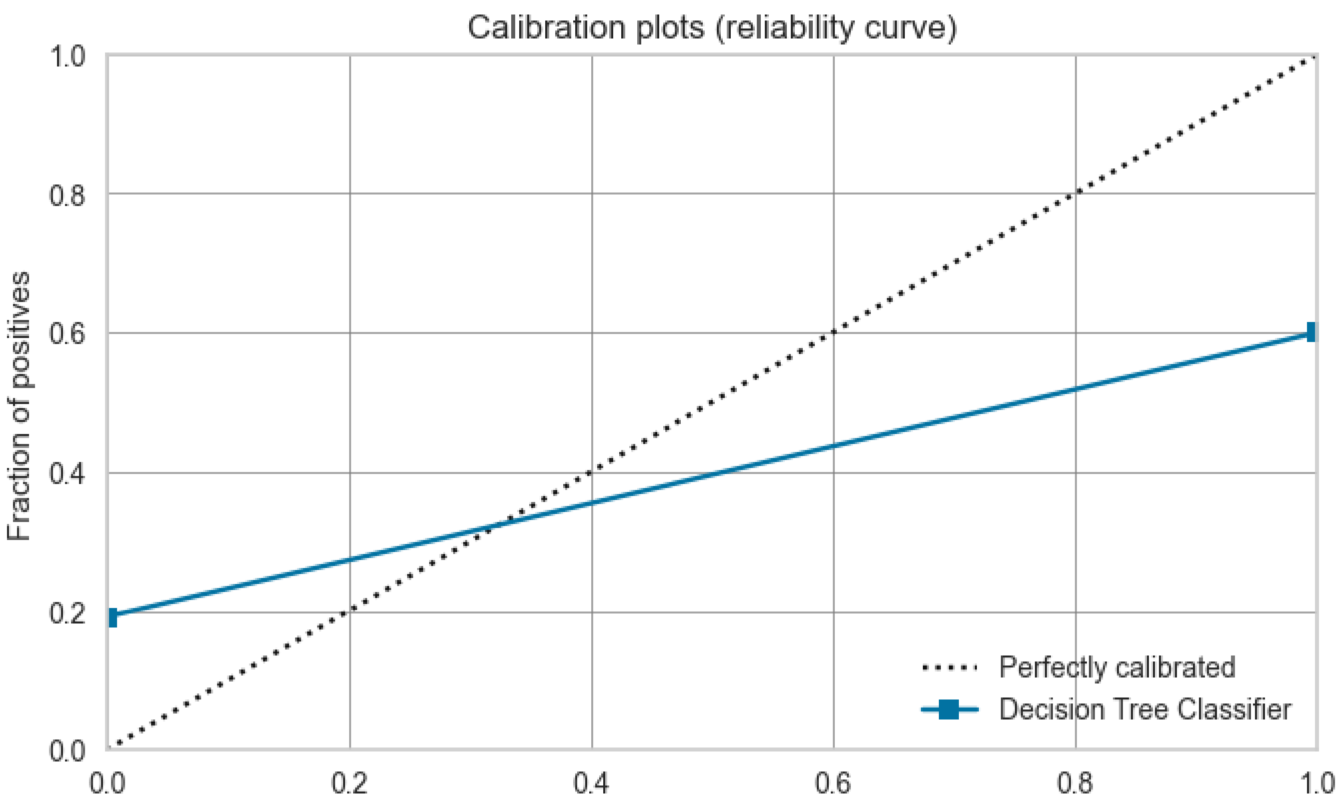
You can check the docstring of the function for more info.

In [76]:
```
# help(interpret_model)
```

✅ Calibrate Model

This function calibrates the probability of a given model using isotonic or logistic regression. The output of this function is a scoring grid with CV scores by fold. Metrics evaluated during CV can be accessed using the `get_metrics` function. Custom metrics can be added or removed using `add_metric` and `remove_metric` function.

In [77]:
```
# check calbiration of default dt
plot_model(dt, plot = 'calibration')
```



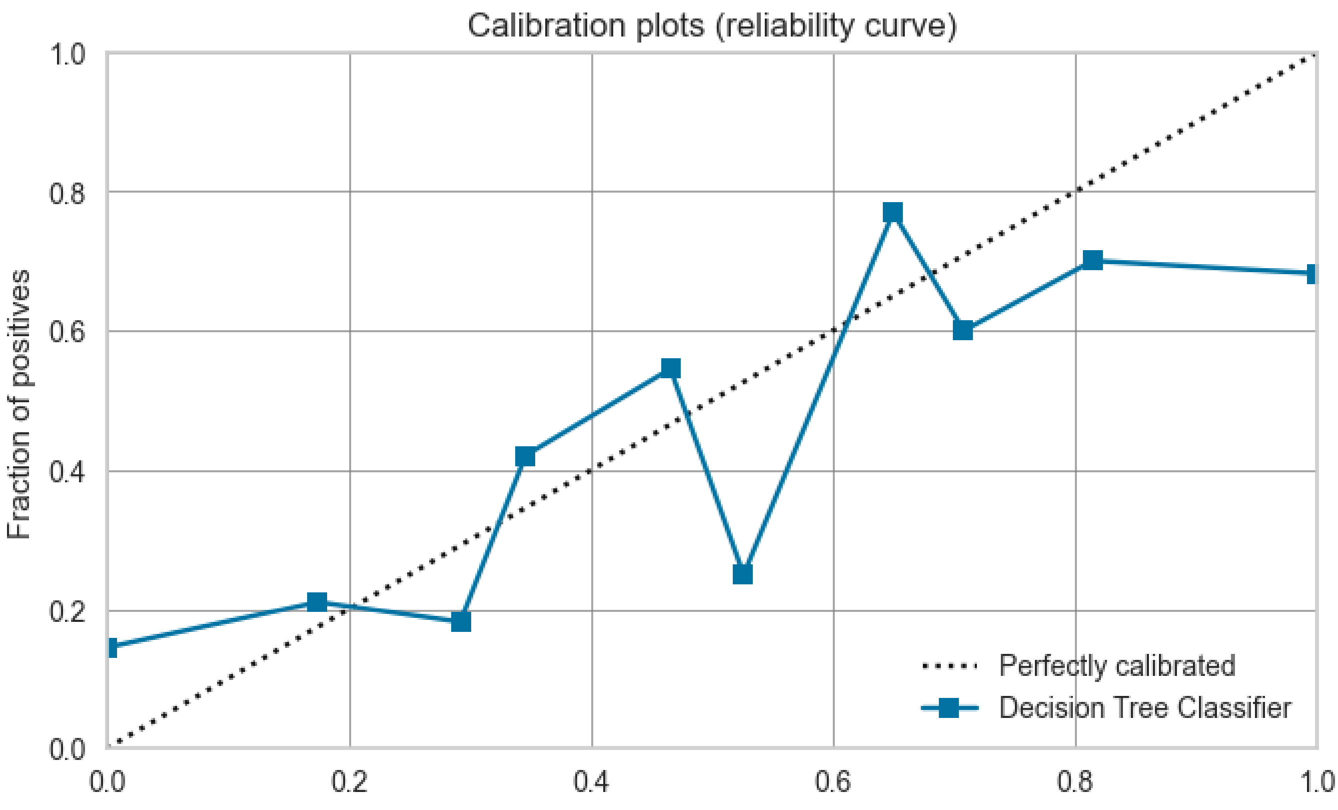
In [78]:
```
# calibrate default dt
calibrated_dt = calibrate_model(dt)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7037	0.7338	0.1579	1.0000	0.2727	0.1955	0.3292
1	0.6296	0.6767	0.1053	0.4000	0.1667	0.0235	0.0322
2	0.6667	0.7677	0.0526	1.0000	0.1000	0.0672	0.1864
3	0.6667	0.6940	0.2105	0.5714	0.3077	0.1459	0.1774
4	0.6481	0.7962	0.0000	0.0000	0.0000	0.0000	0.0000
5	0.5926	0.6045	0.1053	0.2857	0.1538	-0.0439	-0.0534
6	0.7222	0.6752	0.2105	1.0000	0.3478	0.2569	0.3839
7	0.7358	0.7476	0.2222	1.0000	0.3636	0.2740	0.3984
8	0.6226	0.6151	0.1111	0.3333	0.1667	-0.0038	-0.0047
9	0.6792	0.5683	0.1667	0.6000	0.2609	0.1328	0.1774
Mean	0.6667	0.6879	0.1342	0.6190	0.2140	0.1048	0.1627
Std	0.0431	0.0712	0.0692	0.3474	0.1103	0.1074	0.1586

```
Processing:   0%|          | 0/6 [00:00<?, ?it/s]
```

In [79]:

```
# check calbiration of calibrated dt
plot_model(calibrated_dt, plot = 'calibration')
```



Some other parameters that you might find very useful in `calibrate_model` are:

- calibrate_fold
- fit_kwargs
- method
- return_train_score
- groups

You can check the docstring of the function for more info.

In [80]:

```
# help(calibrate_model)
```

✅ Get Leaderboard

This function returns the leaderboard of all models trained in the current setup.

In [81]:
```
# get leaderboard
lb = get_leaderboard()
lb
```

```
Processing:   0%|          | 0/76 [00:00<?, ?it/s]
```

Out[81]:

Index	Model Name	Model	Accuracy	AUC	Recall	Prec.	
0	Logistic Regression	(TransformerWrapper(exclude=None, include=None...	0.7689	0.8068	0.4959	0.7614	0.
1	K Neighbors Classifier	(TransformerWrapper(exclude=None, include=None...	0.7002	0.7433	0.4860	0.5965	0.
2	Naive Bayes	(TransformerWrapper(exclude=None, include=None...	0.7427	0.7957	0.5702	0.6543	0.
3	Decision Tree Classifier	(TransformerWrapper(exclude=None, include=None...	0.6947	0.6526	0.5137	0.5665	0.
4	SVM - Linear Kernel	(TransformerWrapper(exclude=None, include=None...	0.7521	0.0000	0.5070	0.7363	0.
...	...	...	...	...	...	...	
70	Decision Tree Classifier	(TransformerWrapper(exclude=None, include=None...	0.7021	0.6667	0.5506	0.5731	0.
71	Voting Classifier	(TransformerWrapper(exclude=None, include=None...	0.7595	0.8056	0.5649	0.6976	0.
72	Stacking Classifier	(TransformerWrapper(exclude=None, include=None...	0.7633	0.8035	0.5333	0.7190	0.
73	Light Gradient Boosting Machine	(TransformerWrapper(exclude=None, include=None...	0.7113	0.7653	0.5181	0.6036	0.
74	Decision Tree Classifier	(TransformerWrapper(exclude=None, include=None...	0.6667	0.6879	0.1342	0.6190	0.

75 rows × 10 columns

In [82]:
```
# select the best model based on F1
lb.sort_values(by='F1', ascending=False)['Model'].iloc[0]
```

Out[82]:

```
▸ Pipeline
  ▸ clean_column_names: TransformerWrapper
      ▸ transformer: CleanColumnNames
          ▸ CleanColumnNames
  ▸ numerical_imputer: TransformerWrapper
      ▸ transformer: SimpleImputer
          ▸ SimpleImputer
  ▸ categorical_imputer: TransformerWrapper
      ▸ transformer: SimpleImputer
          ▸ SimpleImputer
  ▸ normalize: TransformerWrapper
      ▸ transformer: MinMaxScaler
          ▸ MinMaxScaler
  ▸ LinearDiscriminantAnalysis
```

Some other parameters that you might find very useful in `get_leaderboard` are:

- finalize_models
- fit_kwargs
- model_only
- groups

You can check the docstring of the function for more info.

In [83]:

```
# help(get_leaderboard)
```

✅ AutoML

This function returns the best model out of all trained models in the current setup based on the optimize parameter. Metrics evaluated can be accessed using the `get_metrics` function.

In [84]:

```
automl()
```

Out[84]:

```
▾ RidgeClassifier
RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=Tru
e,
                max_iter=None, normalize='deprecated', positive=False,
                random_state=123, solver='auto', tol=0.001)
```

✅ Check Fairness

There are many approaches to conceptualizing fairness. The check_fairness function follows the approach known as group fairness, which asks: which groups of individuals are at risk for experiencing harm. `check_fairness` provides fairness-related metrics between different groups (also called sub-population).

In [85]:

```
# check fairness
check_fairness(best, sensitive_features = ['Number of times pregnant'])
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Ridge Classifier	0.7489	0.6902	0.4938	0.7018	0.5797	0.4083	0.4211

Out[85]:

Number of times pregnant	Samples	Accuracy	Recall	Precision	F1	Kappa	MCC	Select R
0	27	0.814815	0.666667	0.571429	0.615385	0.494382	0.496929	0.259
1	49	0.734694	0.181818	0.333333	0.235294	0.091298	0.097443	0.122
2	27	0.888889	0.5	1.0	0.666667	0.608696	0.661438	0.111
3	21	0.761905	0.5	0.8	0.615385	0.455959	0.482382	0.238
4	23	0.652174	0.333333	0.333333	0.333333	0.098039	0.098039	0.26
5	21	0.714286	0.375	0.75	0.5	0.329787	0.36863	0.190
6	18	0.722222	0.571429	0.666667	0.615385	0.4	0.402911	0.333
7	13	0.692308	0.5	1.0	0.666667	0.434783	0.527046	0.307
8	12	0.666667	0.333333	1.0	0.5	0.333333	0.447214	0.166
9	5	0.8	0.75	1.0	0.857143	0.545455	0.612372	
10	7	0.714286	0.8	0.8	0.8	0.3	0.3	0.714
11	2	0.5	1.0	0.5	0.666667	0.0	0.0	
12	2	1.0	1.0	1.0	1.0	NaN	0.0	
13	2	1.0	1.0	1.0	1.0	1.0	1.0	
14	2	0.5	0.5	1.0	0.666667	0.0	0.0	

✅ Dashboard

The dashboard function generates the interactive dashboard for a trained model. The dashboard is implemented using `ExplainerDashboard`. For more information check out [Explainer Dashboard.]()

In [86]:
```
# dashboard function
dashboard(dt, display_format ='inline')
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Note: model_output=='probability', so assuming that raw shap output of DecisionTreeCla
ssifier is in probability space...
Generating self.shap_explainer = shap.TreeExplainer(model)
Building ExplainerDashboard..
The explainer object has no decision_trees property. so setting decision_trees=Fals
e...
Warning: calculating shap interaction values can be slow! Pass shap_interaction=False
to remove interactions tab.
Generating layout...
Calculating shap values...
Calculating prediction probabilities...
Calculating metrics...
Calculating confusion matrices...
Calculating classification_dfs...
Calculating roc auc curves...
Calculating pr auc curves...
Calculating liftcurve_dfs...
Calculating shap interaction values... (this may take a while)
Reminder: TreeShap computational complexity is O(TLD^2), where T is the number of tree
s, L is the maximum number of leaves in any tree and D the maximal depth of any tree.
So reducing these will speed up the calculation.
Calculating dependencies...
Calculating permutation importances (if slow, try setting n_jobs parameter)...
Calculating predictions...
Calculating pred_percentiles...
Reminder: you can store the explainer (including calculated dependencies) with explain
er.dump('explainer.joblib') and reload with e.g. ClassifierExplainer.from_file('explai
ner.joblib')
Registering callbacks...
Starting ExplainerDashboard inline (terminate it with ExplainerDashboard.terminate(805
0))
```

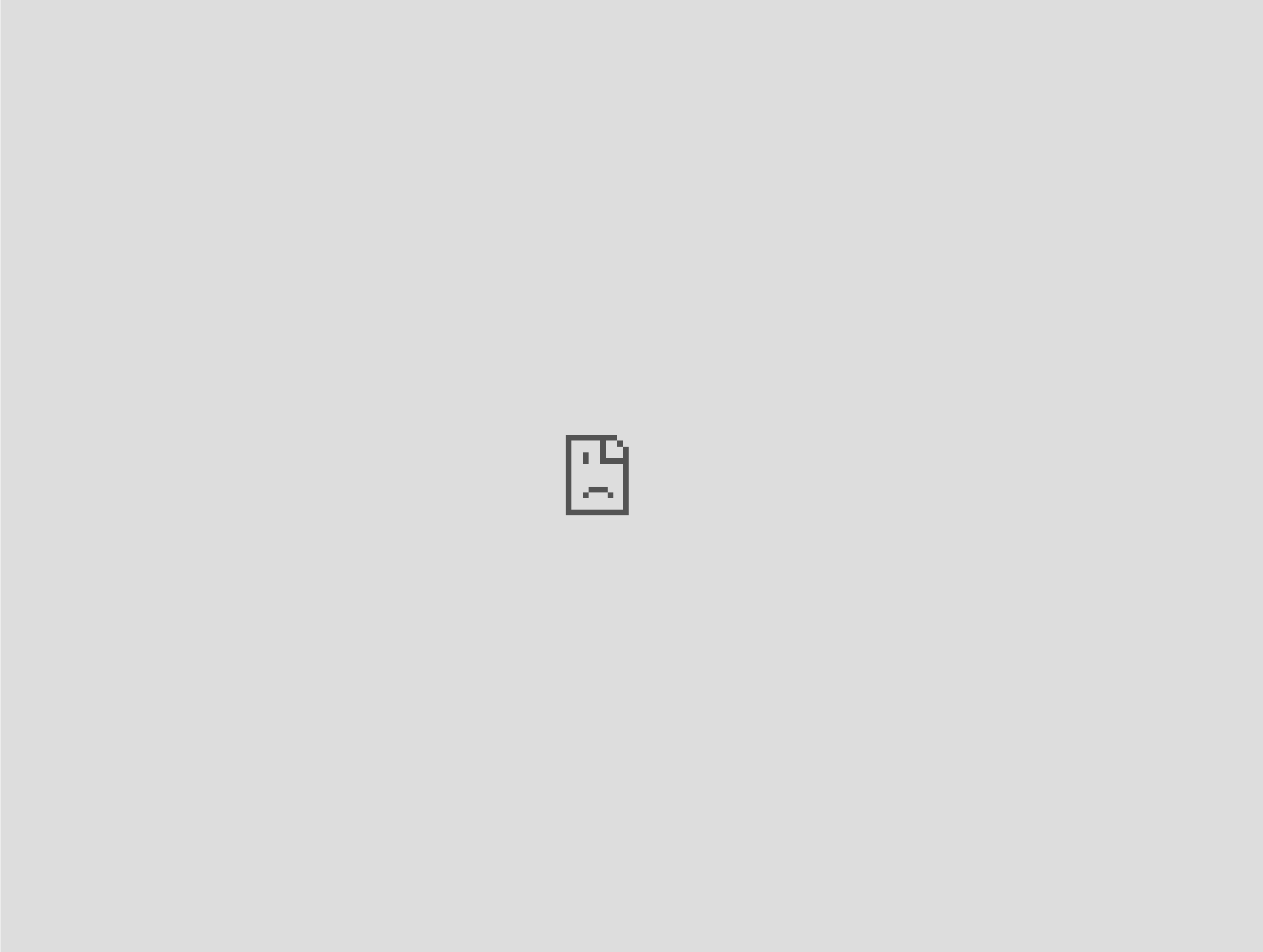
✅Create App

This function creates a basic gradio app for inference.

In [89]:
```python
# create gradio app
create_app(best)
```

```
Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
```



Out[89]:

✅ Create API

This function takes an input model and creates a POST API for inference.

In [90]:
```
# create api
create_api(best, api_name = 'my_first_api')
```

```
API successfully created. This function only creates a POST API, it doesn't run it aut
omatically. To run your API, please run this command --> !python my_first_api.py
```

In [91]:
```
# !python my_first_api.py
```

In [92]:
```
# check out the .py file created with this magic command
# %load my_first_api.py
```

✅ Create Docker

This function creates a `Dockerfile` and `requirements.txt` for productionalizing API end-point.

In [93]:
```
create_docker('my_first_api')
```

```
Writing requirements.txt
Writing Dockerfile
Dockerfile and requirements.txt successfully created.
    To build image you have to run --> !docker image build -f "Dockerfile" -t IMAGE_NA
ME:IMAGE_TAG .
```

In [94]:
```
# check out the DockerFile file created with this magic command
# %load DockerFile
```

In [95]:
```
# check out the requirements file created with this magic command
# %load requirements.txt
```

✅ Finalize Model

This function trains a given model on the entire dataset including the hold-out set.

In [96]:
```
final_best = finalize_model(best)
```

In [97]:
```
final_best
```

Out[97]:
```
Pipeline
  clean_column_names: TransformerWrapper
    transformer: CleanColumnNames
      CleanColumnNames
  numerical_imputer: TransformerWrapper
    transformer: SimpleImputer
      SimpleImputer
  categorical_imputer: TransformerWrapper
    transformer: SimpleImputer
      SimpleImputer
  normalize: TransformerWrapper
    transformer: MinMaxScaler
      MinMaxScaler
  RidgeClassifier
```

✅ Convert Model

This function transpiles the trained machine learning model's decision function in different programming languages such as Python, C, Java, Go, C#, etc. It is very useful if you want to deploy models into environments where you can't install your normal Python stack to support model inference.

In [98]:
```
# transpiles learned function to java
print(convert_model(best, language = 'java'))
```

```
public class Model {
    public static double score(double[] input) {
        return -2.4222329408494767 + input[0] * 0.5943492729771869 + input[1] * 2.3273
354603187455 + input[2] * -0.41637843900032867 + input[3] * 0.10259178891131746 + inpu
t[4] * -0.3134524281639536 + input[5] * 1.4903417391961826 + input[6] * 0.501968541379
2472 + input[7] * 0.12389520576261319;
    }
}
```

✅ Deploy Model

This function deploys the entire ML pipeline on the cloud.

AWS: When deploying model on AWS S3, environment variables must be configured using the command-line interface. To configure AWS environment variables, type `aws configure` in terminal. The following information is required which can be generated using the Identity and Access Management (IAM) portal of your amazon console account:

- AWS Access Key ID
- AWS Secret Key Access
- Default Region Name (can be seen under Global settings on your AWS console)
- Default output format (must be left blank)

GCP: To deploy a model on Google Cloud Platform ('gcp'), the project must be created using the command-line or GCP console. Once the project is created, you must create a service account and download the service account key as a JSON file to set environment variables in your local environment. Learn more about it: https://cloud.google.com/docs/authentication/production

Azure: To deploy a model on Microsoft Azure ('azure'), environment variables for the connection string must be set in your local environment. Go to settings of storage account on Azure portal to access the connection string required. AZURE_STORAGE_CONNECTION_STRING (required as environment variable) Learn more about it: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-python?toc=%2Fpython%2Fazure%2FTOC.json

In [99]:

```
# deploy model on aws s3
# deploy_model(best, model_name = 'my_first_platform_on_aws',
#              platform = 'aws', authentication = {'bucket' : 'pycaret-test'})
```

In [100…

```
# load model from aws s3
# loaded_from_aws = load_model(model_name = 'my_first_platform_on_aws', platform = 'a
#                              authentication = {'bucket' : 'pycaret-test'})

# loaded_from_aws
```

✅ Save / Load Model

This function saves the transformation pipeline and a trained model object into the current working directory as a pickle file for later use.

In [101…

```
# save model
save_model(best, 'my_first_model')
```

```
Transformation Pipeline and Model Successfully Saved
```

Out[101…
```
(Pipeline(memory=FastMemory(location=C:\Users\owner\AppData\Local\Temp\joblib),
          steps=[('clean_column_names',
                  TransformerWrapper(exclude=None, include=None,
                                     transformer=CleanColumnNames(match='[\\]\\
[\\,\\{\\}\\"\\:]+'))),
                 ('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['Number of times pregnant',
                                              'Plasma glucose concentration a 2 '
                                              'hours in an oral glu...
                                                                 verbose='deprecate
d'))),
                 ('normalize',
                  TransformerWrapper(exclude=None, include=None,
                                     transformer=MinMaxScaler(clip=False,
                                                              copy=True,
                                                              feature_range=(0,
                                                                             1)))),
                 ('trained_model',
                  RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True,
                                  fit_intercept=True, max_iter=None,
                                  normalize='deprecated', positive=False,
                                  random_state=123, solver='auto', tol=0.001))],
          verbose=False),
 'my_first_model.pkl')
```

In [102…
```python
# load model
loaded_from_disk = load_model('my_first_model')
loaded_from_disk
```

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

Out[102…

▸ **Pipeline**

▸ **clean_column_names: TransformerWrapper**

▸ **transformer: CleanColumnNames**

▸ CleanColumnNames

▸ **numerical_imputer: TransformerWrapper**

▸ **transformer: SimpleImputer**

▸ SimpleImputer

▸ **categorical_imputer: TransformerWrapper**

▸ **transformer: SimpleImputer**

▸ SimpleImputer

▸ **normalize: TransformerWrapper**

▸ **transformer: MinMaxScaler**

▸ MinMaxScaler

▸ RidgeClassifier

✅ Save / Load Experiment

This function saves all the experiment variables on disk, allowing to later resume without rerunning the setup function.

In [103…

```
# save experiment
save_experiment('my_experiment')
```

In [104…

```
# load experiment from disk
exp_from_disk = load_experiment('my_experiment', data=data)
```

	Description	Value
0	Session id	123
1	Target	Class variable
2	Target type	Binary
3	Original data shape	(768, 9)
4	Transformed data shape	(768, 9)
5	Transformed train set shape	(537, 9)
6	Transformed test set shape	(231, 9)
7	Numeric features	8
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Normalize	True
13	Normalize method	minmax
14	Fold Generator	StratifiedKFold
15	Fold Number	10
16	CPU Jobs	-1
17	Use GPU	False
18	Log Experiment	False
19	Experiment Name	clf-default-name
20	USI	3e8a

In [ ]: