🚀 Quickstart

Get Up and Running in No Time: A Beginner's Guide to PyCaret

🚀 Classification

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.

Setup

This function initializes the training environment and creates the transformation pipeline. Setup function must be called before executing any other function. It takes two required parameters: `data` and `target`. All the other parameters are optional.

```
# load sample dataset
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)	Class variable
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

PyCaret 3.0 has two API's. You can choose one of it based on your preference. The functionalities and experiment results are consistent.

Functional API

```
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	1124

OOP API

```
from pycaret.classification import ClassificationExperiment
s = ClassificationExperiment()
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	431c

```
<pycaret.classification.oop.ClassificationExperiment at 0x20e3da1a850>
```

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.

```
# functional API
best = compare_models()

# OOP API
best = s.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	1.5090
ridge	Ridge Classifier	0.7670	0.0000	0.5497	0.7235	0.6221	0.4581	0.4690	0.0360
lda	Linear Discriminant Analysis	0.7670	0.8055	0.5550	0.7202	0.6243	0.4594	0.4695	0.0680
rf	Random Forest Classifier	0.7485	0.7911	0.5284	0.6811	0.5924	0.4150	0.4238	0.1640
nb	Naive Bayes	0.7427	0.7955	0.5702	0.6543	0.6043	0.4156	0.4215	0.0490
catboost	CatBoost Classifier	0.7410	0.7993	0.5278	0.6630	0.5851	0.4005	0.4078	0.2030
gbc	Gradient Boosting Classifier	0.7373	0.7918	0.5550	0.6445	0.5931	0.4013	0.4059	0.0960
ada	Ada Boost Classifier	0.7372	0.7799	0.5275	0.6585	0.5796	0.3926	0.4017	0.1100
et	Extra Trees Classifier	0.7299	0.7788	0.4965	0.6516	0.5596	0.3706	0.3802	0.1560
qda	Quadratic Discriminant Analysis	0.7282	0.7894	0.5281	0.6558	0.5736	0.3785	0.3910	0.0460
lightgbm	Light Gradient Boosting Machine	0.7133	0.7645	0.5398	0.6036	0.5650	0.3534	0.3580	0.2690
knn	K Neighbors Classifier	0.7001	0.7164	0.5020	0.5982	0.5413	0.3209	0.3271	0.0590
dt	Decision Tree Classifier	0.6928	0.6512	0.5137	0.5636	0.5328	0.3070	0.3098	0.0500
xgboost	Extreme Gradient Boosting	0.6853	0.7516	0.4912	0.5620	0.5216	0.2887	0.2922	0.0840
dummy	Dummy Classifier	0.6518	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0410
svm	SVM - Linear Kernel	0.5954	0.0000	0.3395	0.4090	0.2671	0.0720	0.0912	0.0400

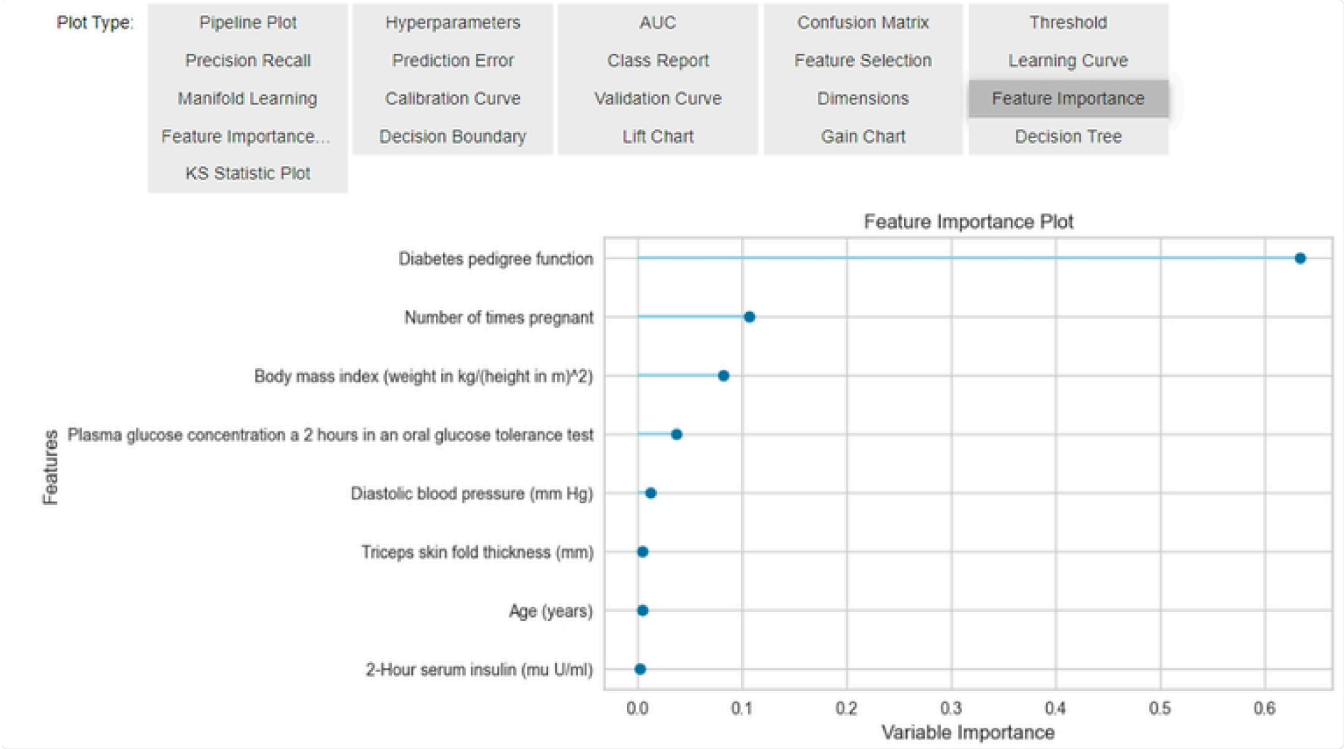
```
print(best)
```

```
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)
```

Analyze Model

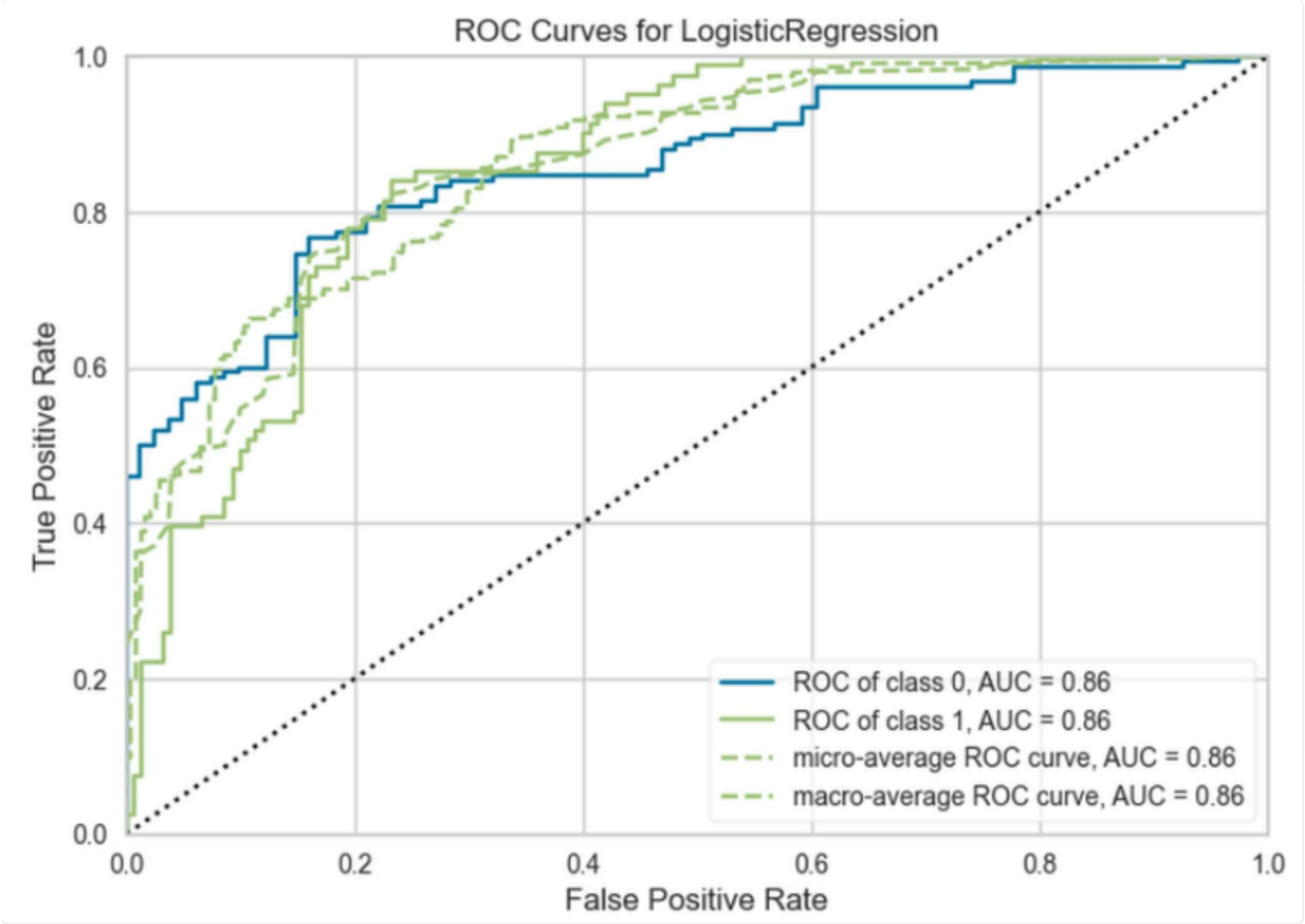
This function analyzes the performance of a trained model on the test set. It may require re-training the model in certain cases.

```
# functional API
evaluate_model(best)

# OOP API
s.evaluate_model(best)
```

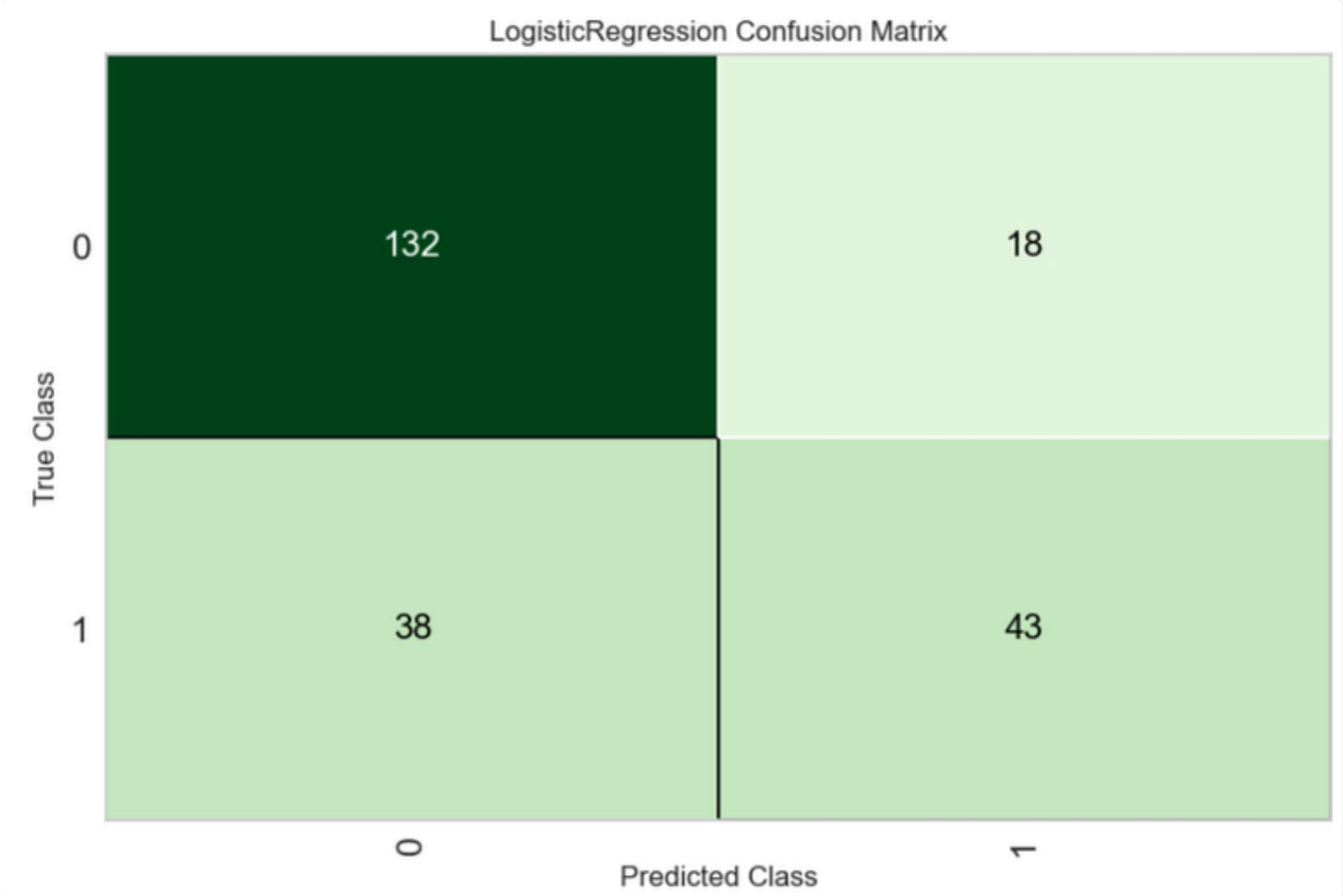


`evaluate_model` can only be used in Notebook since it uses `ipywidget` . You can also use the `plot_model` function to generate plots individually.

```
# functional API
plot_model(best, plot = 'auc')

# OOP API
s.plot_model(best, plot = 'auc')
```




```
# functional API
plot_model(best, plot = 'confusion_matrix')

# OOP API
s.plot_model(best, plot = 'confusion_matrix')
```



Predictions

This function scores the data and returns `prediction_label` and `prediction_score` probability of the predicted class). When `data` is None, it predicts label and score on the test set (created during the `setup` function).

```
# functional API
predict_model(best)

# OOP API
s.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

	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)	Class variable	prediction_label	prediction_score
537	6	114	88	0	0	27.799999	0.247	66	0	0	0.8037
538	1	97	70	15	0	18.200001	0.147	21	0	0	0.9648
539	2	90	70	17	0	27.299999	0.085	22	0	0	0.9393
540	2	105	58	40	94	34.900002	0.225	25	0	0	0.7998
541	11	138	76	0	0	33.200001	0.420	35	0	1	0.6391
...	...	...	...	...	...	...	...	...	...	...	...
763	2	110	74	29	125	32.400002	0.698	27	0	0	0.8002
764	10	115	0	0	0	35.299999	0.134	29	0	1	0.6229
765	14	100	78	25	184	36.599998	0.412	46	1	0	0.5986
766	1	95	74	21	73	25.900000	0.673	36	0	0	0.9243
767	0	119	66	27	0	38.799999	0.259	22	0	0	0.6797

231 rows × 11 columns

The evaluation metrics are calculated on the test set. The second output is the `pd.DataFrame` with predictions on the test set (see the last two columns). To generate labels on the unseen (new) dataset, simply pass the dataset in the `data` parameter under `predict_model` function.

```
# functional API
predictions = predict_model(best, data=data)
predictions.head()

# OOP API
predictions = s.predict_model(best, data=data)
predictions.head()
```

	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)	Class variable	prediction_label	prediction_score
0	6	148	72	35	0	33.599998	0.627	50	1	1	0.6939
1	1	85	66	29	0	26.600000	0.351	31	0	0	0.9419
2	8	183	64	0	0	23.299999	0.672	32	1	1	0.7975
3	1	89	66	23	94	28.100000	0.167	21	0	0	0.9453
4	0	137	40	35	168	43.099998	2.288	33	1	1	0.8393

`Score` means the probability of the predicted class (NOT the positive class). If `prediction_label` is 0 and `prediction_score` is 0.90, this means 90% probability of class 0. If you want to see the probability of both the classes, simply pass `raw_score=True` in the `predict_model` function.

```
# functional API
predictions = predict_model(best, data=data, raw_score=True)
predictions.head()

# OOP API
predictions = s.predict_model(best, data=data, raw_score=True)
predictions.head()
```

	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)	Class variable	prediction_label	prediction_score_0	prediction_score_1
0	6	148	72	35	0	33.599998	0.627	50	1	1	0.3061	0.6939
1	1	85	66	29	0	26.600000	0.351	31	0	0	0.9419	0.0581
2	8	183	64	0	0	23.299999	0.672	32	1	1	0.2025	0.7975
3	1	89	66	23	94	28.100000	0.167	21	0	0	0.9453	0.0547
4	0	137	40	35	168	43.099998	2.288	33	1	1	0.1607	0.8393

Save the model

```
# functional API
save_model(best, 'my_best_pipeline')

# OOP API
s.save_model(best, 'my_best_pipeline')
```

```
Transformation Pipeline and Model Successfully Saved

(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_frequent',
                                                          verbose='deprecated'))),
                 ('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_best_pipeline.pkl')
```

To load the model back in environment:

```python
# functional API
loaded_model = load_model('my_best_pipeline')
print(loaded_model)

# OOP API
loaded_model = s.load_model('my_best_pipeline')
print(loaded_model)
```

```
Transformation Pipeline and Model Successfully Loaded
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_frequent',
                                                          verbose='deprecated'))),
                ('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)
```

🚀 Regression

PyCaret's Regression Module is a supervised machine learning module that is used for estimating the relationships between a dependent variable (often called the 'outcome variable', or 'target') and one or more independent variables (often called 'features', 'predictors', or 'covariates').

The objective of regression is to predict continuous values such as predicting sales amount, predicting quantity, predicting temperature, etc.

Setup

This function initializes the training environment and creates the transformation pipeline. Setup function must be called before executing any other function. It takes two required parameters: `data` and `target`. All the other parameters are optional.

```
# load sample dataset
from pycaret.datasets import get_data
data = get_data('insurance')
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

PyCaret 3.0 has two API's. You can choose one of it based on your preference. The functionalities and experiment results are consistent.

Functional API

```
from pycaret.regression import *
s = setup(data, target = 'charges', session_id = 123)
```

	Description	Value
0	Session id	123
1	Target	charges
2	Target type	Regression
3	Original data shape	(1338, 7)
4	Transformed data shape	(1338, 10)
5	Transformed train set shape	(936, 10)
6	Transformed test set shape	(402, 10)
7	Ordinal features	2
8	Numeric features	3
9	Categorical features	3
10	Preprocess	True
11	Imputation type	simple
12	Numeric imputation	mean
13	Categorical imputation	mode
14	Maximum one-hot encoding	25
15	Encoding method	None
16	Fold Generator	KFold
17	Fold Number	10
18	CPU Jobs	-1
19	Use GPU	False
20	Log Experiment	False
21	Experiment Name	reg-default-name
22	USI	e95e

OOP API

```
from pycaret.regression import RegressionExperiment
s = RegressionExperiment()
s.setup(data, target = 'charges', session_id = 123)
```

	Description	Value
0	Session id	123
1	Target	charges
2	Target type	Regression
3	Original data shape	(1338, 7)
4	Transformed data shape	(1338, 10)
5	Transformed train set shape	(936, 10)
6	Transformed test set shape	(402, 10)
7	Ordinal features	2
8	Numeric features	3
9	Categorical features	3
10	Preprocess	True
11	Imputation type	simple
12	Numeric imputation	mean
13	Categorical imputation	mode
14	Maximum one-hot encoding	25
15	Encoding method	None
16	Fold Generator	KFold
17	Fold Number	10
18	CPU Jobs	-1
19	Use GPU	False
20	Log Experiment	False
21	Experiment Name	reg-default-name
22	USI	1da6

```
<pycaret.regression.oop.RegressionExperiment at 0x20e40275cd0>
```

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.

```python
# functional API
best = compare_models()

# OOP API
best = s.compare_models()
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	2701.9919	23548657.1177	4832.9329	0.8320	0.4447	0.3137	0.2400
rf	Random Forest Regressor	2771.4583	25416502.3827	5028.6343	0.8172	0.4690	0.3303	0.4590
catboost	CatBoost Regressor	2899.3783	25762701.9552	5057.5721	0.8163	0.4815	0.3522	0.9220
lightgbm	Light Gradient Boosting Machine	2992.1828	25521038.3331	5042.0978	0.8149	0.5378	0.3751	0.3880
et	Extra Trees Regressor	2833.3624	28427844.2412	5305.6516	0.7991	0.4877	0.3363	0.4770
ada	AdaBoost Regressor	4316.0568	29220505.6498	5398.4561	0.7903	0.6368	0.7394	0.1740
xgboost	Extreme Gradient Boosting	3443.6091	32824626.4000	5711.2140	0.7626	0.6224	0.4469	0.1960
llar	Lasso Least Angle Regression	4298.6038	38369142.0849	6174.9424	0.7309	0.5786	0.4424	0.1100
ridge	Ridge Regression	4317.6984	38396435.9578	6177.2329	0.7306	0.5891	0.4459	0.0870
br	Bayesian Ridge	4311.2349	38391950.0874	6176.8896	0.7306	0.5910	0.4447	0.0730
lar	Least Angle Regression	4303.5559	38388058.4578	6176.5920	0.7306	0.5949	0.4433	0.3910
lasso	Lasso Regression	4303.7697	38386797.6709	6176.4824	0.7306	0.5952	0.4434	0.0920
lr	Linear Regression	4303.5559	38388058.4578	6176.5920	0.7306	0.5949	0.4433	0.8420
huber	Huber Regressor	3463.2216	48801106.4612	6963.9984	0.6544	0.4927	0.2212	0.1060
dt	Decision Tree Regressor	3383.4916	47823199.0729	6895.7016	0.6497	0.5602	0.4013	0.1580
omp	Orthogonal Matching Pursuit	5754.7769	57503207.7233	7566.7086	0.5997	0.7418	0.8990	0.0950
par	Passive Aggressive Regressor	4537.0122	67346309.9218	8142.7826	0.5422	0.5276	0.3207	0.0920
en	Elastic Net	7372.5238	90450782.5713	9468.3193	0.3792	0.7342	0.9184	0.0840
knn	K Neighbors Regressor	8007.7997	131387268.8000	11425.3695	0.0859	0.8535	0.9232	0.1250
dummy	Dummy Regressor	9192.5418	148516792.8000	12132.4733	-0.0175	1.0154	1.5637	0.0900

```python
print(best)
```

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=123, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
```

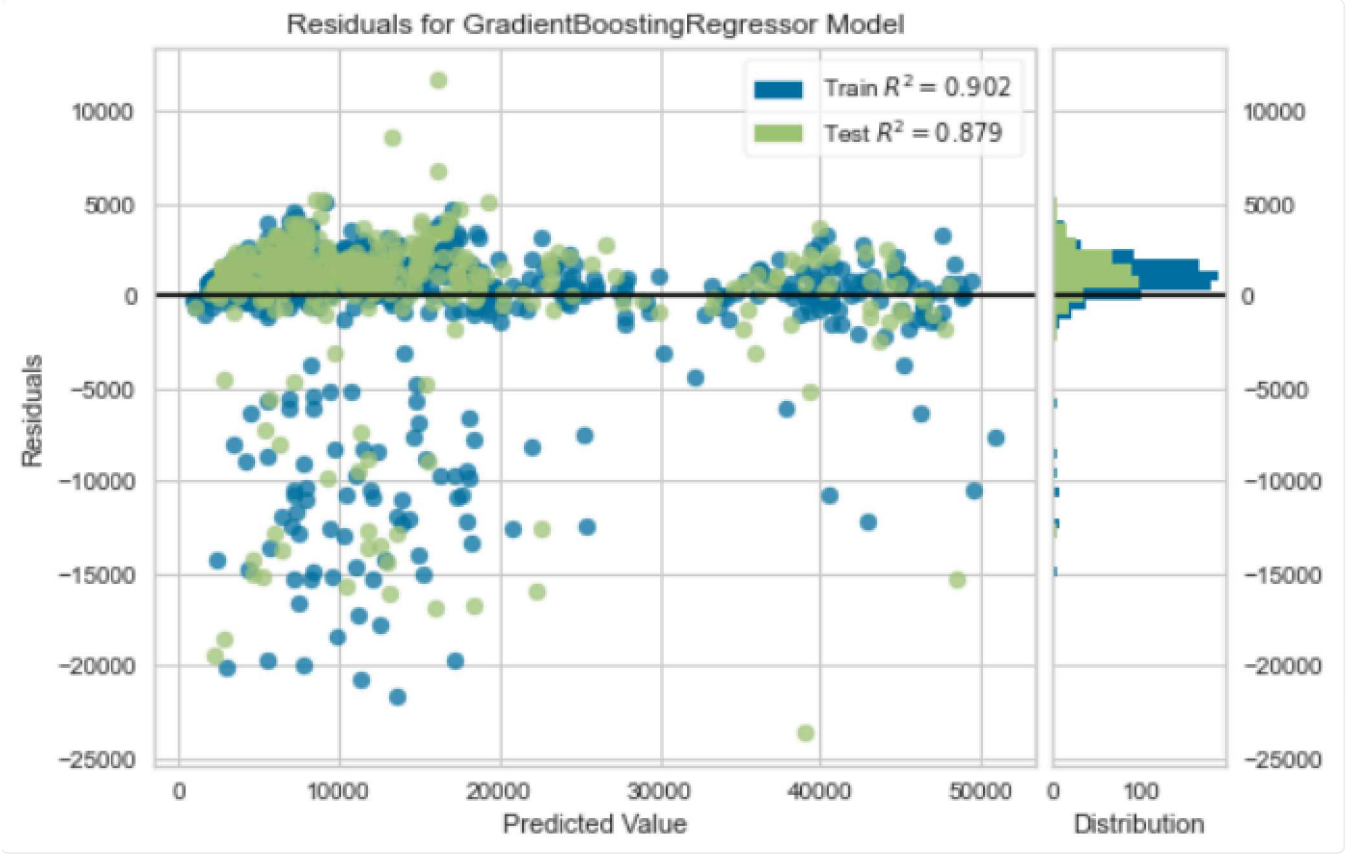
Analyze Model

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

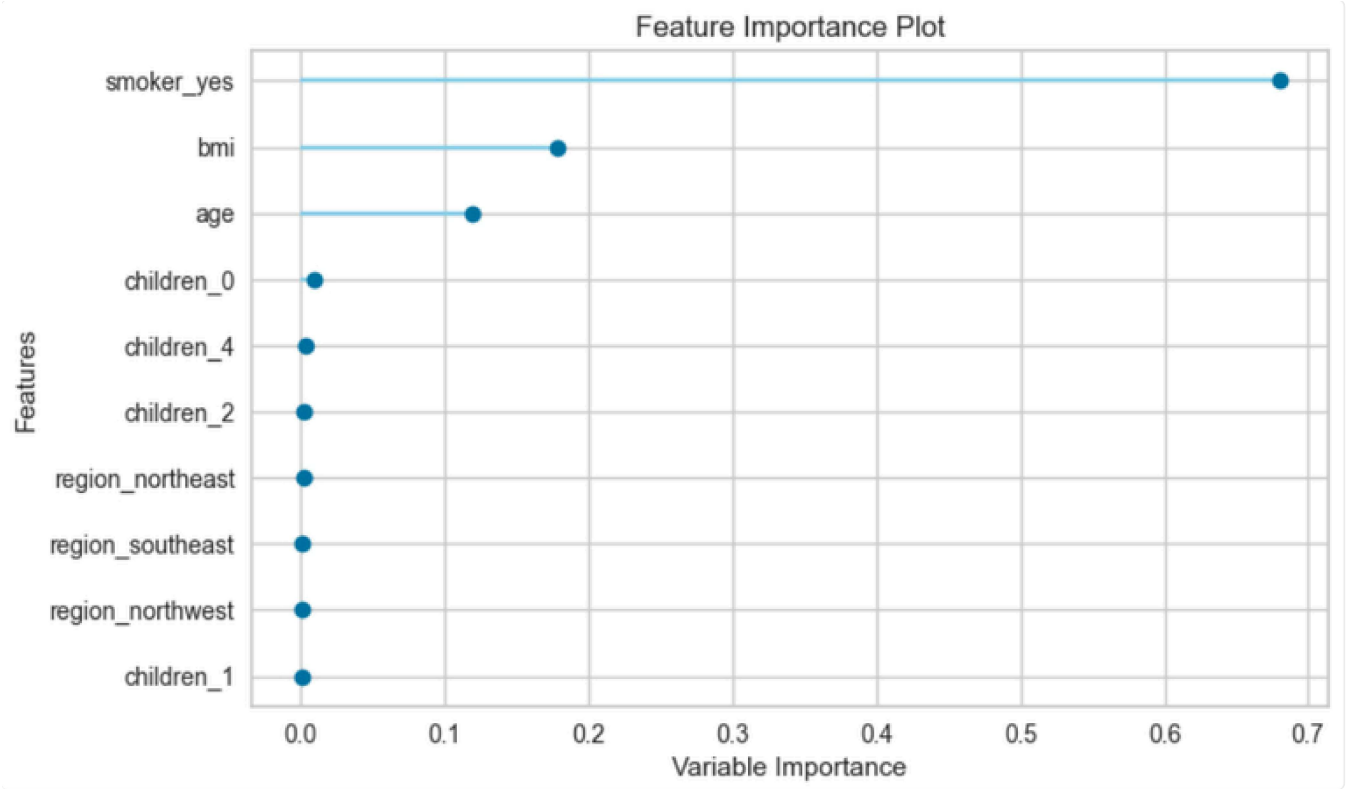
```
# functional API
evaluate_model(best)

# OOP API
s.evaluate_model(best)
```

`evaluate_model` can only be used in Notebook since it uses `ipywidget` . You can also use the `plot_model` function to generate plots individually.

```
# functional API
plot_model(best, plot = 'residuals')

# OOP API
s.plot_model(best, plot = 'residuals')
```




```
# functional API
plot_model(best, plot = 'feature')

# OOP API
s.plot_model(best, plot = 'feature')
```



Predictions

This function predicts `prediction_label` using the trained model. When `data` is None, it predicts label and score on the test set (created during the `setup` function).

```
# functional API
predict_model(best)

# OOP API
s.predict_model(best)
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Gradient Boosting Regressor	2392.5661	17148355.3169	4141.0573	0.8800	0.3928	0.2875

	age	sex	bmi	children	smoker	region	charges	prediction_label
936	49	female	42.680000	2	no	southeast	9800.888672	10681.513104
937	32	male	37.334999	1	no	northeast	4667.607422	8043.453463
938	27	female	31.400000	0	yes	southwest	34838.871094	36153.097686
939	35	male	24.129999	1	no	northwest	5125.215820	7435.516853
940	60	male	25.740000	0	no	southeast	12142.578125	14676.544334
...	...	...	...	...	...	...	...	...
1333	40	male	24.969999	2	no	southeast	6593.508301	9264.152792
1334	19	female	40.500000	0	no	southwest	1759.338013	2604.919960
1335	24	male	29.830000	0	yes	northeast	18648.421875	18639.490954
1336	36	female	30.020000	0	no	northwest	5272.175781	5117.185514
1337	18	female	35.625000	0	no	northeast	2211.130859	3146.534024

The evaluation metrics are calculated on the test set. The second output is the `pd.DataFrame` with predictions on the test set (see the last two columns). To generate labels on the unseen (new) dataset, simply pass the dataset in the `predict_model` function.

```
# functional API
predictions = predict_model(best, data=data)
predictions.head()

# OOP API
predictions = s.predict_model(best, data=data)
predictions.head()
```

	age	sex	bmi	children	smoker	region	charges	prediction_label
0	19	female	27.900000	0	yes	southwest	16884.923828	18464.334448
1	18	male	33.770000	1	no	southeast	1725.552246	4020.345384
2	28	male	33.000000	3	no	southeast	4449.461914	6555.388388
3	33	male	22.705000	0	no	northwest	21984.470703	9627.045725
4	32	male	28.879999	0	no	northwest	3866.855225	3325.531292
...	...	...	...	...	...	...	...	...
1333	50	male	30.969999	3	no	northwest	10600.547852	11824.542819
1334	18	female	31.920000	0	no	northeast	2205.980713	3044.638467
1335	18	female	36.849998	0	no	southeast	1629.833496	2507.539055
1336	21	female	25.799999	0	no	southwest	2007.944946	2606.628126
1337	61	female	29.070000	0	yes	northwest	29141.359375	28636.644913

1338 rows × 8 columns

Save the model

```
# functional API
save_model(best, 'my_best_pipeline')

# OOP API
s.save_model(best, 'my_best_pipeline')
```

```
Transformation Pipeline and Model Successfully Saved

(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_frequent',
                                                             verbose='deprecated'))),
                 ('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_best_pipeline.pkl')
```

To load the model back in the environment:

```python
# functional API
loaded_model = load_model('my_best_pipeline')
print(loaded_model)

# OOP API
loaded_model = s.load_model('my_best_pipeline')
print(loaded_model)
```

```
Transformation Pipeline and Model Successfully Loaded
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_frequent',
                                                             verbose='deprecated'))),
                ('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)
```

🚀 Clustering

PyCaret's Clustering Module is an unsupervised machine learning module that performs the task of grouping a set of objects in such a way that objects in the same group (also known as a cluster) are more similar to each other than to those in other groups.

Setup

This function initializes the training environment and creates the transformation pipeline. Setup function must be called before executing any other function. It takes only one required parameter: `data`. All the other parameters are optional.

```python
# load sample dataset
from pycaret.datasets import get_data
data = get_data('jewellery')
```

	Age	Income	SpendingScore	Savings
0	58	77769	0.791329	6559.829923
1	59	81799	0.791082	5417.661426
2	62	74751	0.702657	9258.992965
3	59	74373	0.765680	7346.334504
4	87	17760	0.348778	16869.507130

PyCaret 3.0 has two API's. You can choose one of it based on your preference. The functionalities and experiment results are consistent.

Functional API

```
from pycaret.clustering import *
s = setup(data, normalize = True)
```

	Description	Value
0	Session id	123
1	Original data shape	(505, 4)
2	Transformed data shape	(505, 4)
3	Numeric features	4
4	Preprocess	True
5	Imputation type	simple
6	Numeric imputation	mean
7	Categorical imputation	mode
8	Normalize	True
9	Normalize method	zscore
10	CPU Jobs	-1
11	Use GPU	False
12	Log Experiment	False
13	Experiment Name	cluster-default-name
14	USI	3988

OOP API

```
from pycaret.clustering import ClusteringExperiment
s = ClusteringExperiment()
s.setup(data, normalize = True)
```

	Description	Value
0	Session id	4594
1	Original data shape	(505, 4)
2	Transformed data shape	(505, 4)
3	Numeric features	4
4	Preprocess	True
5	Imputation type	simple
6	Numeric imputation	mean
7	Categorical imputation	mode
8	Normalize	True
9	Normalize method	zscore
10	CPU Jobs	-1
11	Use GPU	False
12	Log Experiment	False
13	Experiment Name	cluster-default-name
14	USI	801c

```
<pycaret.clustering.oop.ClusteringExperiment at 0x140a461a130>
```

Create Model

This function trains and evaluates the performance of a given model. Metrics evaluated can be accessed using the `get_metrics` function. Custom metrics can be added or removed using the `add_metric` and `remove_metric` function. All the available models can be accessed using the `models` function.

```
# functional API
kmeans = create_model('kmeans')

# OOP API
kmeans = s.create_model('kmeans')
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.7581	1611.2649	0.3743	0	0	0

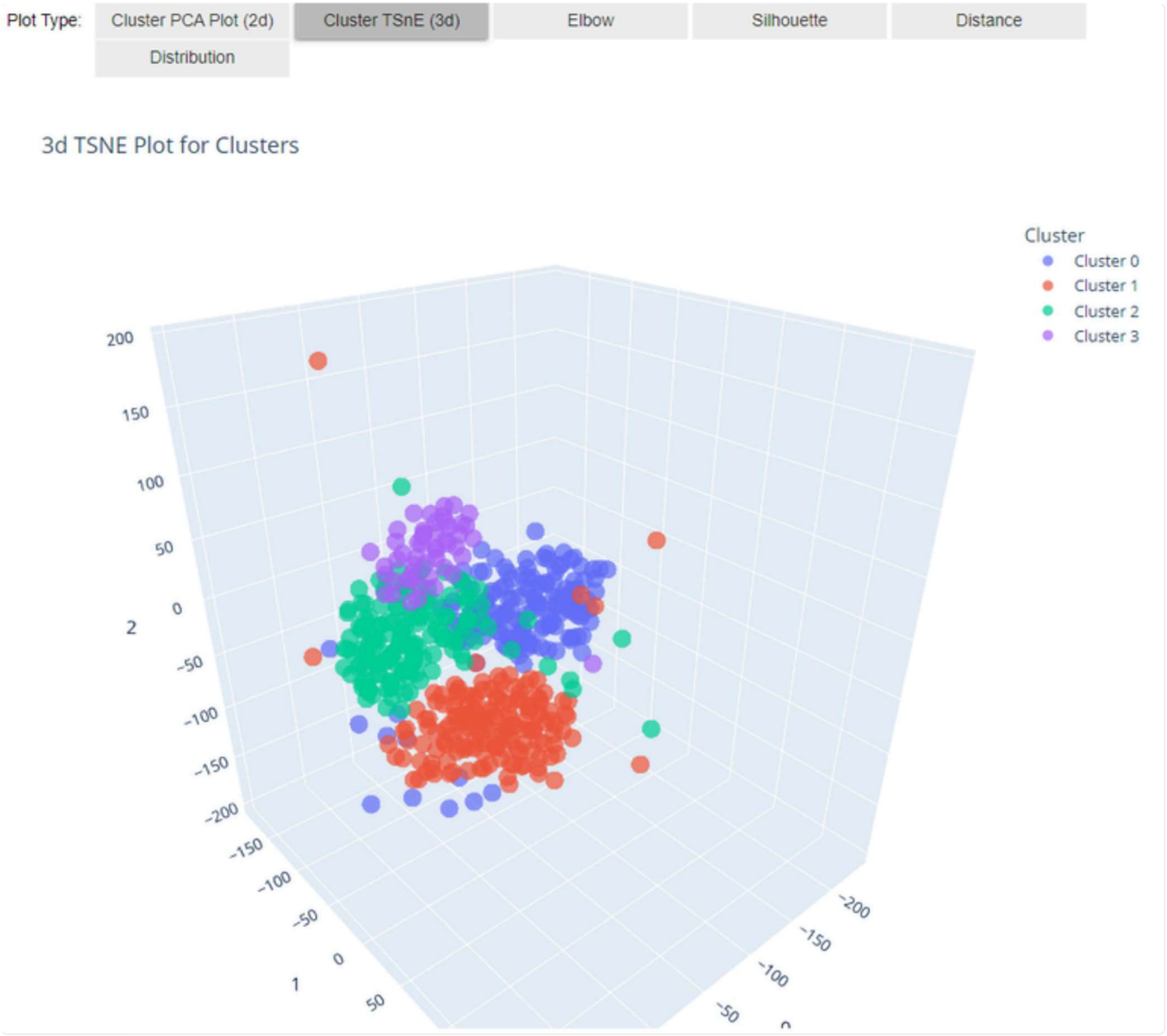
```
print(kmeans)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=-1, precompute_distances='deprecated',
       random_state=123, tol=0.0001, verbose=0)
```

Analyze Model

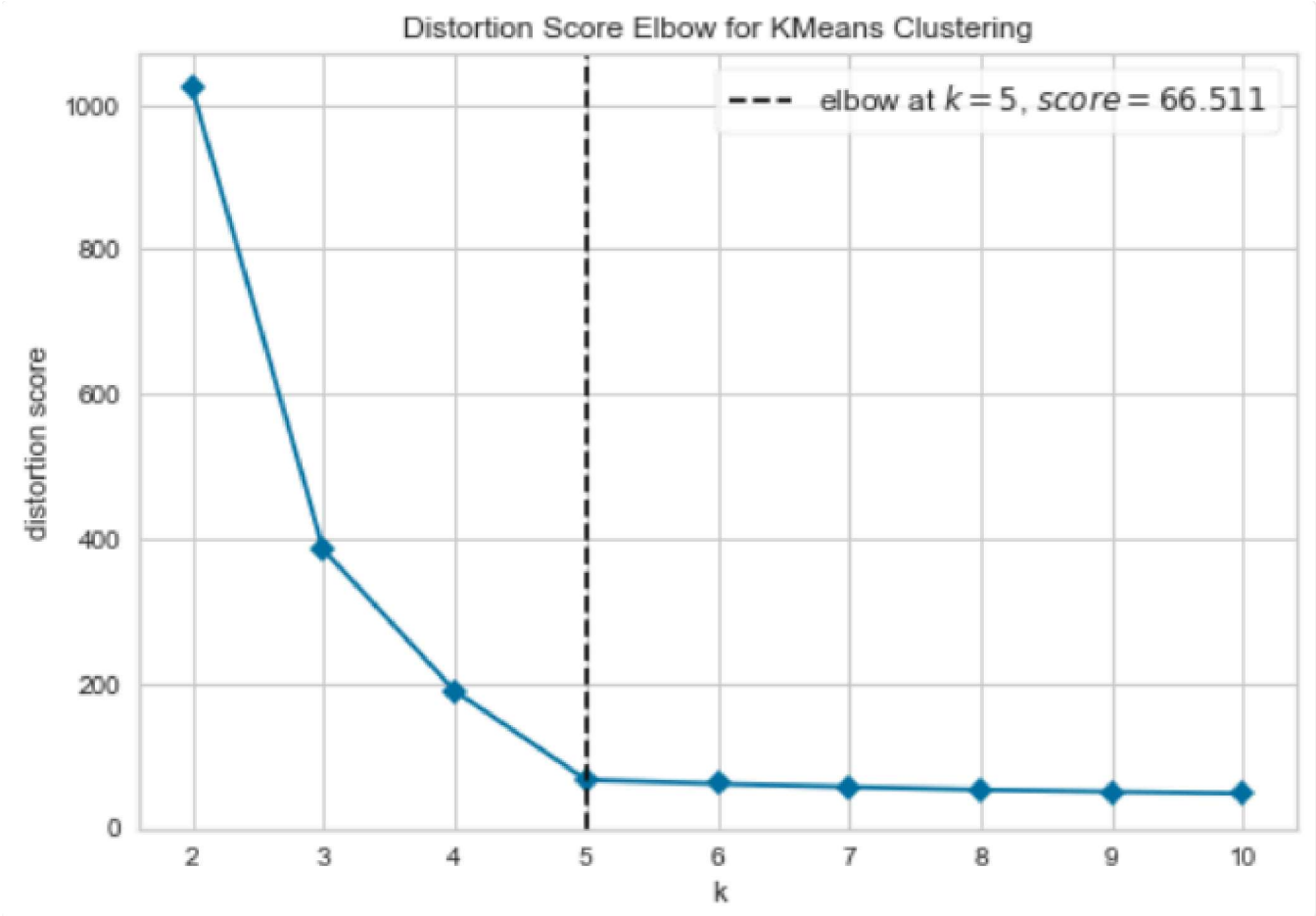
This function analyzes the performance of a trained model.

```
# functional API
evaluate_model(kmeans)

# OOP API
s.evaluate_model(kmeans)
```

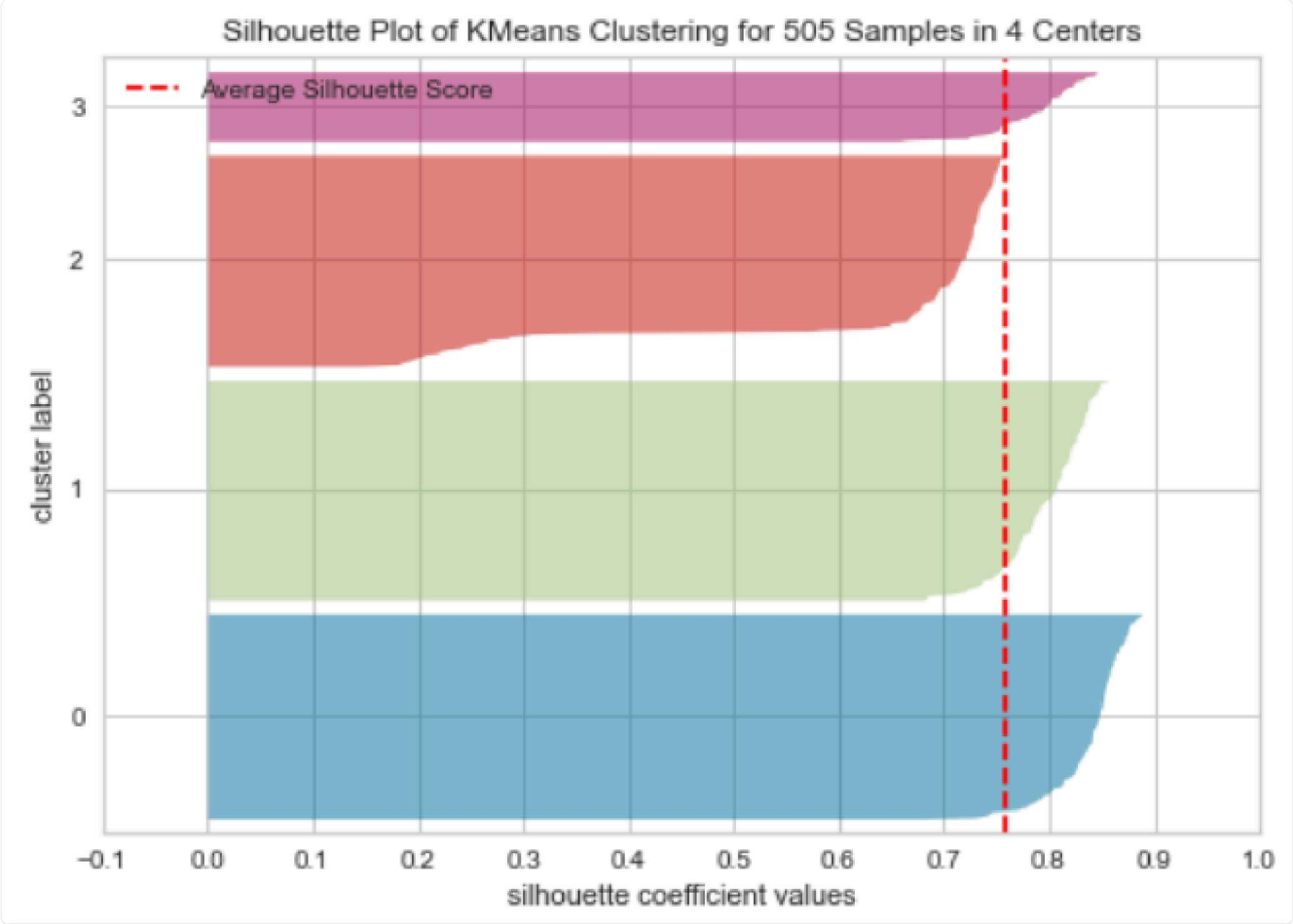


`evaluate_model` can only be used in Notebook since it uses `ipywidget` . You can also use the `plot_model` function to generate plots individually.

```
# functional API
plot_model(kmeans, plot = 'elbow')

# OOP API
s.plot_model(kmeans, plot = 'elbow')
```




```
# functional API
plot_model(kmeans, plot = 'silhouette')

# OOP API
s.plot_model(kmeans, plot = 'silhouette')
```



Assign Model

This function assigns cluster labels to the training data, given a trained model.

```
# functional API
result = assign_model(kmeans)
result.head()

# OOP API
result = s.assign_model(kmeans)
result.head()
```

	Age	Income	SpendingScore	Savings	Cluster
0	58	77769	0.791329	6559.829923	Cluster 1
1	59	81799	0.791082	5417.661426	Cluster 1
2	62	74751	0.702657	9258.992965	Cluster 1
3	59	74373	0.765680	7346.334504	Cluster 1
4	87	17760	0.348778	16869.507130	Cluster 0

Predictions

This function generates cluster labels using a trained model on the new/unseen dataset.

```
# functional API
predictions = predict_model(kmeans, data = data)
predictions.head()

# OOP API
predictions = s.predict_model(kmeans, data = data)
predictions.head()
```

	Age	Income	SpendingScore	Savings	Cluster
0	58	77769	0.791329	6559.829923	Cluster 1
1	59	81799	0.791082	5417.661426	Cluster 1
2	62	74751	0.702657	9258.992965	Cluster 1
3	59	74373	0.765680	7346.334504	Cluster 1
4	87	17760	0.348778	16869.507130	Cluster 0

Save the model

```
# functional API
save_model(kmeans, 'kmeans_pipeline')

# OOP API
s.save_model(kmeans, 'kmeans_pipeline')
```

```
Transformation Pipeline and Model Successfully Saved

(Pipeline(memory=FastMemory(location=C:\Users\owner\AppData\Local\Temp\joblib),
          steps=[('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['Age', 'Income', 'SpendingScore',
                                              'Savings'],
                                     transformer=SimpleImputer(add_indicator=False,
                                                               copy=True,
                                                               fill_value=None,
                                                               missing_values=nan,
                                                               strategy='mean',
                                                               verbose='deprecated'))),
                 ('categorical_imputer',
                  Transfo...
                                                               missing_values=nan,
                                                               strategy='most_frequent',
                                                               verbose='deprecated'))),
                 ('normalize',
                  TransformerWrapper(exclude=None, include=None,
                                     transformer=StandardScaler(copy=True,
                                                                with_mean=True,
                                                                with_std=True))),
                 ('trained_model',
                  KMeans(algorithm='lloyd', copy_x=True, init='k-means++',
                         max_iter=300, n_clusters=4, n_init=10,
                         random_state=4594, tol=0.0001, verbose=0))],
          verbose=False),
 'kmeans_pipeline.pkl')
```

To load the model back in the environment:

```python
# functional API
loaded_model = load_model('kmeans_pipeline')
print(loaded_model)

# OOP API
loaded_model = s.load_model('kmeans_pipeline')
print(loaded_model)
```

```
Transformation Pipeline and Model Successfully Loaded
Pipeline(memory=FastMemory(location=C:\Users\owner\AppData\Local\Temp\joblib),
         steps=[('numerical_imputer',
                 TransformerWrapper(exclude=None,
                                    include=['Age', 'Income', 'SpendingScore',
                                             'Savings'],
                                    transformer=SimpleImputer(add_indicator=False,
                                                              copy=True,
                                                              fill_value=None,
                                                              missing_values=nan,
                                                              strategy='mean',
                                                              verbose='deprecated'))),
                ('categorical_imputer',
                 Transfo...
                                                              missing_values=nan,
                                                              strategy='most_frequent',
                                                              verbose='deprecated'))),
                ('normalize',
                 TransformerWrapper(exclude=None, include=None,
                                    transformer=StandardScaler(copy=True,
                                                               with_mean=True,
                                                               with_std=True))),
                ('trained_model',
                 KMeans(algorithm='lloyd', copy_x=True, init='k-means++',
                        max_iter=300, n_clusters=4, n_init=10,
                        random_state=4594, tol=0.0001, verbose=0))],
         verbose=False)
```

🚀 Anomaly Detection

PyCaret's Anomaly Detection Module is an unsupervised machine learning module that is used for identifying rare items, events, or observations that raise suspicions by differing significantly from the majority of the data.

Typically, the anomalous items will translate to some kind of problems such as bank fraud, a structural defect, medical problems, or errors.

Setup

This function initializes the training environment and creates the transformation pipeline. The `setup` function must be called before executing any other function. It takes only one required parameter only: `data`. All the other parameters are optional.

```
# load sample dataset
from pycaret.datasets import get_data
data = get_data('anomaly')
```

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10
0	0.263995	0.764929	0.138424	0.935242	0.605867	0.518790	0.912225	0.608234	0.723782	0.733591
1	0.546092	0.653975	0.065575	0.227772	0.845269	0.837066	0.272379	0.331679	0.429297	0.367422
2	0.336714	0.538842	0.192801	0.553563	0.074515	0.332993	0.365792	0.861309	0.899017	0.088600
3	0.092108	0.995017	0.014465	0.176371	0.241530	0.514724	0.562208	0.158963	0.073715	0.208463
4	0.325261	0.805968	0.957033	0.331665	0.307923	0.355315	0.501899	0.558449	0.885169	0.182754

PyCaret 3.0 has two API's. You can choose one of it based on your preference. The functionalities and experiment results are consistent.

Functional API

```
from pycaret.anomaly import *
s = setup(data, session_id = 123)
```

	Description	Value
0	Session id	123
1	Original data shape	(1000, 10)
2	Transformed data shape	(1000, 10)
3	Numeric features	10
4	Preprocess	True
5	Imputation type	simple
6	Numeric imputation	mean
7	Categorical imputation	mode
8	CPU Jobs	-1
9	Use GPU	False
10	Log Experiment	False
11	Experiment Name	anomaly-default-name
12	USI	16b0

OOP API

```
from pycaret.anomaly import AnomalyExperiment
s = AnomalyExperiment()
s.setup(data, session_id = 123)
```

	Description	Value
0	Session id	123
1	Original data shape	(1000, 10)
2	Transformed data shape	(1000, 10)
3	Numeric features	10
4	Preprocess	True
5	Imputation type	simple
6	Numeric imputation	mean
7	Categorical imputation	mode
8	CPU Jobs	-1
9	Use GPU	False
10	Log Experiment	False
11	Experiment Name	anomaly-default-name
12	USI	dd5f

```
<pycaret.anomaly.oop.AnomalyExperiment at 0x140ac774910>
```

Create Model

This function trains an unsupervised anomaly detection model. All the available models can be accessed using the `models` function.

```
# functional API
iforest = create_model('iforest')
print(iforest)

# OOP API
iforest = s.create_model('iforest')
print(iforest)
```

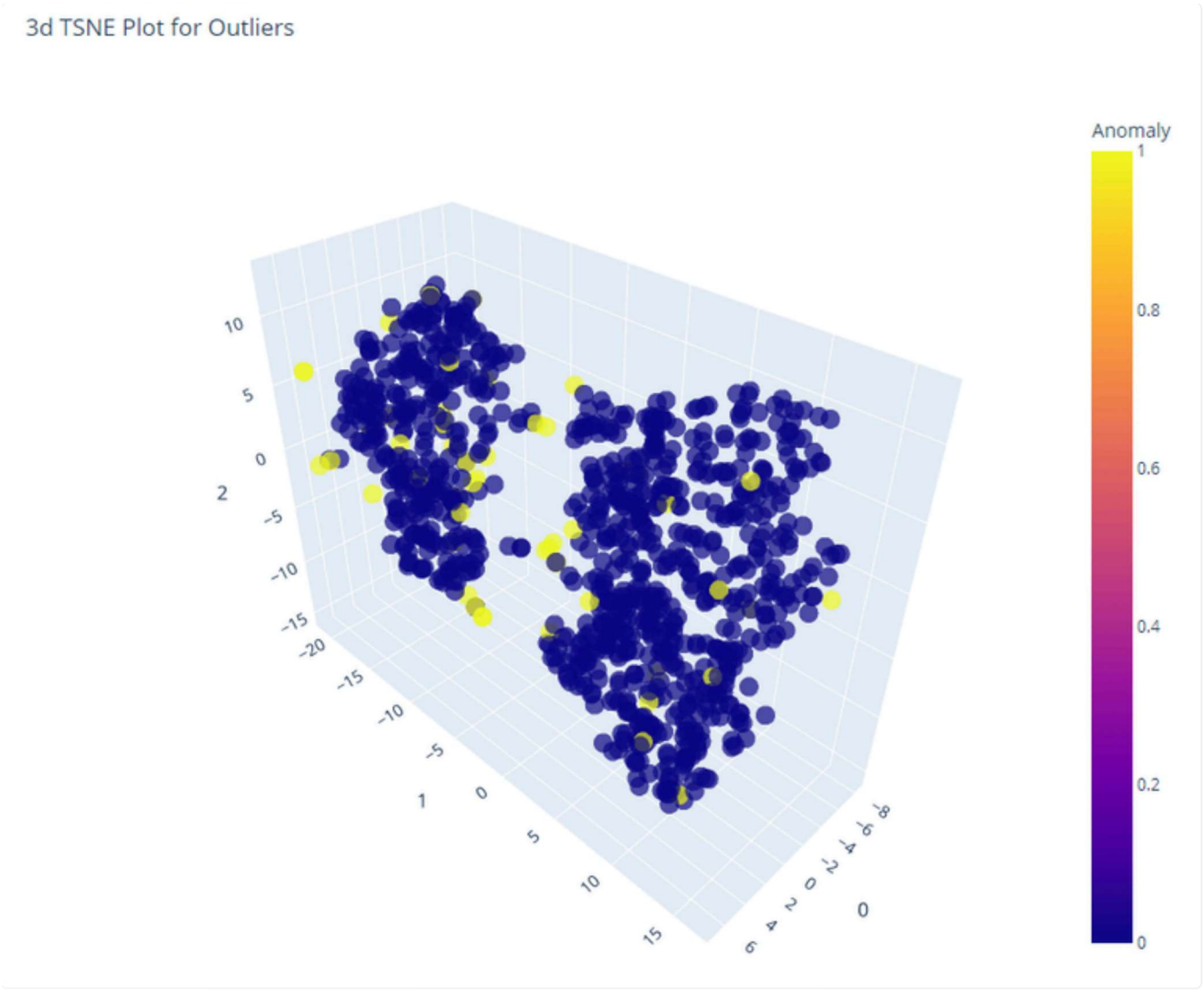
```
IForest(behaviour='new', bootstrap=False, contamination=0.05,
    max_features=1.0, max_samples='auto', n_estimators=100, n_jobs=-1,
    random_state=123, verbose=0)
```

```
# functional API
models()

# OOP API
s.models()
```

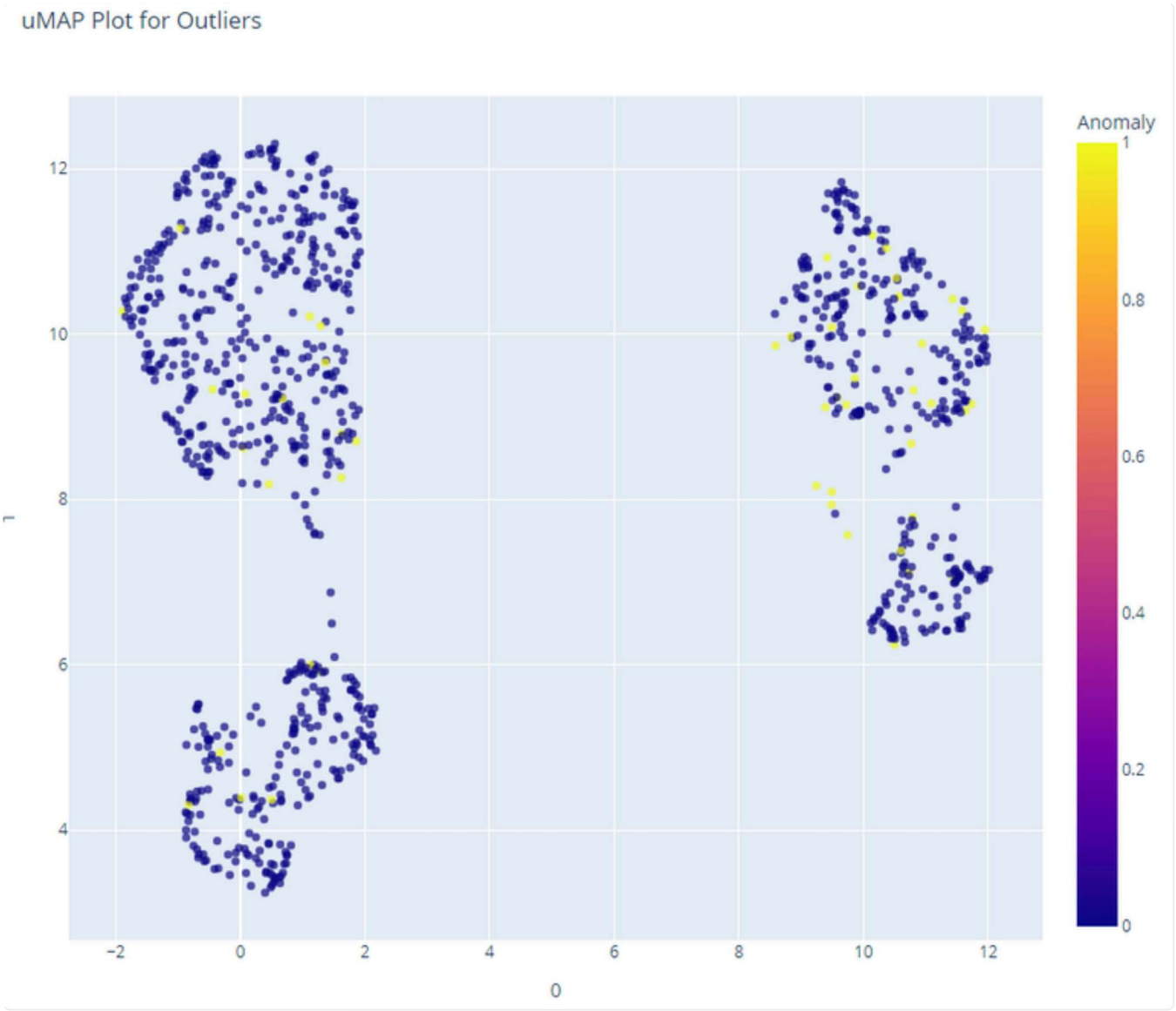
ID	Name	Reference
abod	Angle-base Outlier Detection	pyod.models.abod.ABOD
cluster	Clustering-Based Local Outlier	pyod.models.cblof.CBLOF
cof	Connectivity-Based Local Outlier	pyod.models.cof.COF
iforest	Isolation Forest	pyod.models.iforest.IForest
histogram	Histogram-based Outlier Detection	pyod.models.hbos.HBOS
knn	K-Nearest Neighbors Detector	pyod.models.knn.KNN
lof	Local Outlier Factor	pyod.models.lof.LOF
svm	One-class SVM detector	pyod.models.ocsvm.OCSVM
pca	Principal Component Analysis	pyod.models.pca.PCA
mcd	Minimum Covariance Determinant	pyod.models.mcd.MCD
sod	Subspace Outlier Detection	pyod.models.sod.SOD
sos	Stochastic Outlier Selection	pyod.models.sos.SOS

Analyze Model

```
# functional API
plot_model(iforest, plot = 'tsne')

# OOP API
s.plot_model(iforest, plot = 'tsne')
```




```
# functional API
plot_model(iforest, plot = 'umap')

# OOP API
s.plot_model(iforest, plot = 'umap')
```



Assign Model

This function assigns anomaly labels to the dataset for a given model. (1 = outlier, 0 = inlier).

```
# functional API
result = assign_model(iforest)
result.head()

# OOP API
result = s.assign_model(iforest)
result.head()
```

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Anomaly	Anomaly_Score
0	0.263995	0.764929	0.138424	0.935242	0.605867	0.518790	0.912225	0.608234	0.723782	0.733591	0	-0.035865
1	0.546092	0.653975	0.065575	0.227772	0.845269	0.837066	0.272379	0.331679	0.429297	0.367422	0	-0.084927
2	0.336714	0.538842	0.192801	0.553563	0.074515	0.332993	0.365792	0.861309	0.899017	0.088600	1	0.025356
3	0.092108	0.995017	0.014465	0.176371	0.241530	0.514724	0.562208	0.158963	0.073715	0.208463	1	0.042415
4	0.325261	0.805968	0.957033	0.331665	0.307923	0.355315	0.501899	0.558449	0.885169	0.182754	0	-0.023408

Predictions

This function generates anomaly labels using a trained model on the new/unseen dataset.

```
# functional API
predictions = predict_model(iforest, data = data)
predictions.head()

# OOP API
predictions = s.predict_model(iforest, data = data)
predictions.head()
```

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Anomaly	Anomaly_Score
0	0.263995	0.764929	0.138424	0.935242	0.605867	0.518790	0.912225	0.608234	0.723782	0.733591	0	-0.034955
1	0.546092	0.653975	0.065575	0.227772	0.845269	0.837066	0.272379	0.331679	0.429297	0.367422	0	-0.072194
2	0.336714	0.538842	0.192801	0.553563	0.074515	0.332993	0.365792	0.861309	0.899017	0.088600	1	0.008371
3	0.092108	0.995017	0.014465	0.176371	0.241530	0.514724	0.562208	0.158963	0.073715	0.208463	1	0.044704
4	0.325261	0.805968	0.957033	0.331665	0.307923	0.355315	0.501899	0.558449	0.885169	0.182754	0	-0.027766

Output from predict_model(iforest, data = data)

Save the model

```
# functional API
save_model(iforest, 'iforest_pipeline')

# OOP API
s.save_model(iforest, 'iforest_pipeline')
```

```
Transformation Pipeline and Model Successfully Saved

(Pipeline(memory=FastMemory(location=C:\Users\owner\AppData\Local\Temp\joblib),
          steps=[('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['Col1', 'Col2', 'Col3', 'Col4',
                                              'Col5', 'Col6', 'Col7', 'Col8',
                                              'Col9', 'Col10'],
                                     transformer=SimpleImputer(add_indicator=False,
                                                               copy=True,
                                                               fill_value=None,
                                                               missing_values=nan,
                                                               strategy='mean',
                                                               verbose='deprecated'))),...
                  TransformerWrapper(exclude=None, include=[],
                                     transformer=SimpleImputer(add_indicator=False,
                                                               copy=True,
                                                               fill_value=None,
                                                               missing_values=nan,
                                                               strategy='most_frequent',
                                                               verbose='deprecated'))),
                 ('trained_model',
                  IForest(behaviour='new', bootstrap=False, contamination=0.05,
    max_features=1.0, max_samples='auto', n_estimators=100, n_jobs=-1,
    random_state=123, verbose=0))],
          verbose=False),
 'iforest_pipeline.pkl')
```

To load the model back in the environment:

```python
# functional API
loaded_model = load_model('iforest_pipeline')
print(loaded_model)

# OOP API
loaded_model = s.load_model('iforest_pipeline')
print(loaded_model)
```

```
Transformation Pipeline and Model Successfully Loaded
Pipeline(memory=FastMemory(location=C:\Users\owner\AppData\Local\Temp\joblib),
         steps=[('numerical_imputer',
                 TransformerWrapper(exclude=None,
                                    include=['Col1', 'Col2', 'Col3', 'Col4',
                                             'Col5', 'Col6', 'Col7', 'Col8',
                                             'Col9', 'Col10'],
                                    transformer=SimpleImputer(add_indicator=False,
                                                              copy=True,
                                                              fill_value=None,
                                                              missing_values=nan,
                                                              strategy='mean',
                                                              verbose='deprecated'))),...
                 TransformerWrapper(exclude=None, include=[],
                                    transformer=SimpleImputer(add_indicator=False,
                                                              copy=True,
                                                              fill_value=None,
                                                              missing_values=nan,
                                                              strategy='most_frequent',
                                                              verbose='deprecated'))),
                ('trained_model',
                 IForest(behaviour='new', bootstrap=False, contamination=0.05,
    max_features=1.0, max_samples='auto', n_estimators=100, n_jobs=-1,
    random_state=123, verbose=0))],
         verbose=False)
```

🚀 Time Series

PyCaret Time Series module is a powerful tool for analyzing and predicting time series data using machine learning and classical statistical techniques. This module enables users to easily perform complex time series forecasting tasks by automating the entire process from data preparation to model deployment.

PyCaret Time Series Forecasting module supports a wide range of forecasting methods such as ARIMA, Prophet, and LSTM. It also provides various features to handle missing values, time series decomposition, and data visualizations.

Setup

This function initializes the training environment and creates the transformation pipeline. Setup function must be called before executing any other function.

```python
# load sample dataset
from pycaret.datasets import get_data
data = get_data('airline')
```

```
Period
1949-01    112.0
1949-02    118.0
1949-03    132.0
1949-04    129.0
1949-05    121.0
Freq: M, Name: Number of airline passengers, dtype: float64
```

PyCaret 3.0 has two API's. You can choose one of it based on your preference. The functionalities and experiment results are consistent.

Functional API

```
from pycaret.time_series import *
s = setup(data, fh = 3, fold = 5, session_id = 123)
```

	Description	Value
0	session_id	123
1	Target	Number of airline passengers
2	Approach	Univariate
3	Exogenous Variables	Not Present
4	Original data shape	(144, 1)
5	Transformed data shape	(144, 1)
6	Transformed train set shape	(141, 1)
7	Transformed test set shape	(3, 1)
8	Rows with missing values	0.0%
9	Fold Generator	ExpandingWindowSplitter
10	Fold Number	5
11	Enforce Prediction Interval	False
12	Splits used for hyperparameters	all
13	Seasonality Detection Algo	auto
14	Max Period to Consider	60
15	Seasonal Period(s) Tested	[12, 24, 36, 11, 48]
16	Significant Seasonal Period(s)	[12, 24, 36, 11, 48]
17	Significant Seasonal Period(s) without Harmonics	[48, 36, 11]
18	Remove Harmonics	False

Output truncated

OOP API

```
from pycaret.time_series import TSForecastingExperiment
s = TSForecastingExperiment()
```

	Description	Value
0	session_id	123
1	Target	Number of airline passengers
2	Approach	Univariate
3	Exogenous Variables	Not Present
4	Original data shape	(144, 1)
5	Transformed data shape	(144, 1)
6	Transformed train set shape	(141, 1)
7	Transformed test set shape	(3, 1)
8	Rows with missing values	0.0%
9	Fold Generator	ExpandingWindowSplitter
10	Fold Number	5
11	Enforce Prediction Interval	False
12	Splits used for hyperparameters	all
13	Seasonality Detection Algo	auto
14	Max Period to Consider	60
15	Seasonal Period(s) Tested	[12, 24, 36, 11, 48]
16	Significant Seasonal Period(s)	[12, 24, 36, 11, 48]
17	Significant Seasonal Period(s) without Harmonics	[48, 36, 11]
18	Remove Harmonics	False

Output truncated

Compare Models

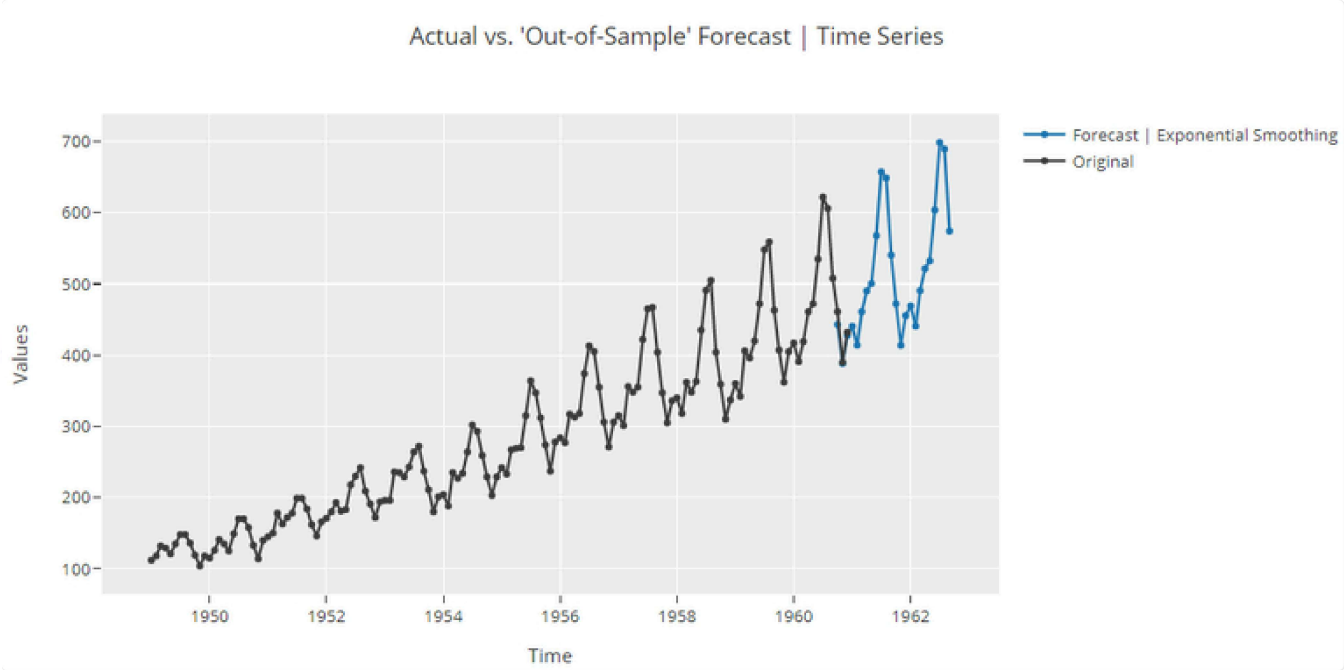
This function trains and evaluates the performance of all the estimators available in the model library using cross-validation. The output of this function is a scoring grid with average cross-validated scores. Metrics evaluated during CV can be accessed using the `get_metrics` function. Custom metrics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

```
# functional API
best = compare_models()

# OOP API
best = s.compare_models()
```

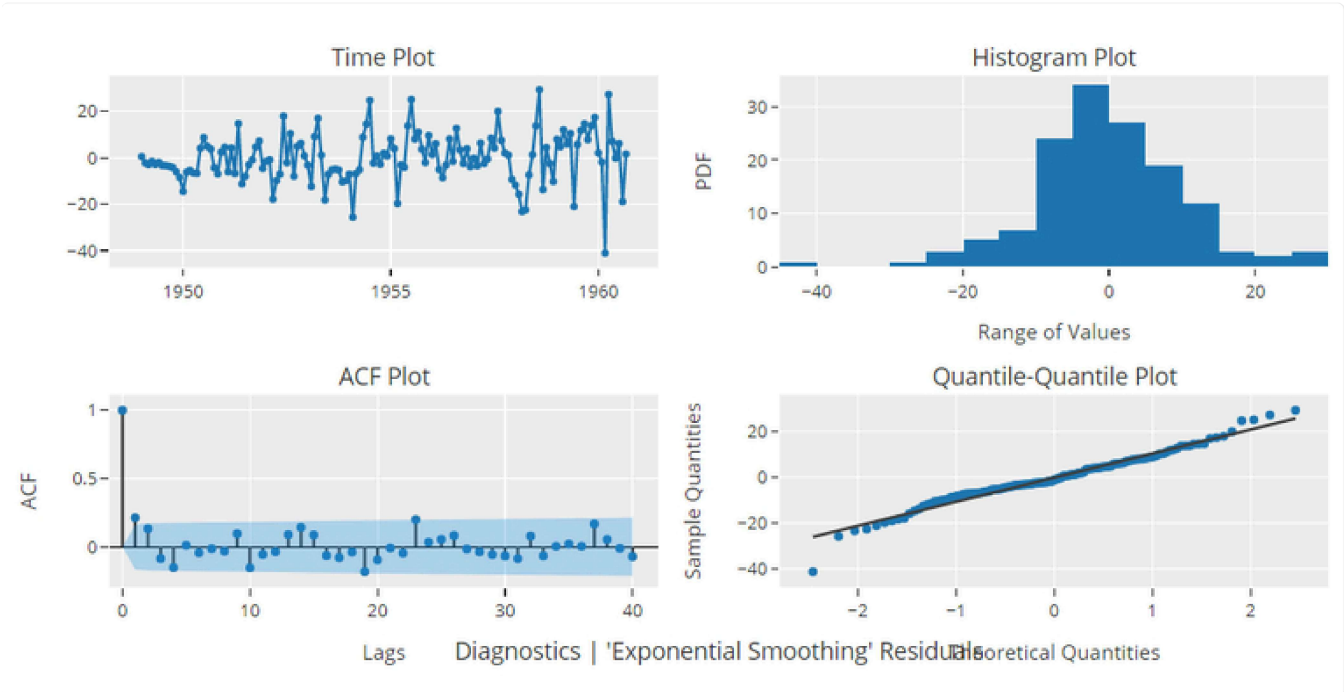
	Model	MAE	RMSE	MAPE	SMAPE	MASE	RMSSE	R2	TT (Sec)
exp_smooth	Exponential Smoothing	15.2893	18.4452	0.0334	0.0335	0.5063	0.5378	-0.0519	0.0920
ets	ETS	16.6159	19.8384	0.0354	0.0357	0.5520	0.5801	-0.0740	0.0960
arima	ARIMA	19.5728	22.3027	0.0412	0.0420	0.6480	0.6501	-0.0796	0.0260
auto_arima	Auto ARIMA	19.7329	21.6100	0.0414	0.0420	0.6524	0.6297	-0.0554	13.7600
par_cds_dt	Passive Aggressive w/ Cond. Deseasonalize & Detrending	21.0358	22.1139	0.0430	0.0441	0.6968	0.6457	0.2684	0.0080
theta	Theta Forecaster	25.7024	28.3332	0.0524	0.0541	0.8458	0.8223	-0.7710	0.0080
huber_cds_dt	Huber w/ Cond. Deseasonalize & Detrending	27.2568	30.5782	0.0550	0.0572	0.9002	0.8900	-0.0309	0.0140
knn_cds_dt	K Neighbors w/ Cond. Deseasonalize & Detrending	28.5678	30.5007	0.0555	0.0575	0.9381	0.8830	0.0908	0.0140
lar_cds_dt	Least Angular Regressor w/ Cond. Deseasonalize & Detrending	27.1821	30.2170	0.0556	0.0578	0.8989	0.8804	-0.1059	0.0100
lr_cds_dt	Linear w/ Cond. Deseasonalize & Detrending	28.6337	31.9163	0.0581	0.0605	0.9469	0.9297	-0.1620	0.0100
ridge_cds_dt	Ridge w/ Cond. Deseasonalize & Detrending	28.6340	31.9164	0.0581	0.0605	0.9469	0.9297	-0.1620	0.0100
br_cds_dt	Bayesian Ridge w/ Cond. Deseasonalize & Detrending	28.9018	32.1013	0.0582	0.0606	0.9551	0.9347	-0.1377	0.0100
en_cds_dt	Elastic Net w/ Cond. Deseasonalize & Detrending	28.7271	31.9952	0.0582	0.0606	0.9499	0.9320	-0.1579	0.0100
lasso_cds_dt	Lasso w/ Cond. Deseasonalize & Detrending	28.7941	32.0557	0.0583	0.0607	0.9521	0.9337	-0.1560	0.0100
et_cds_dt	Extra Trees w/ Cond. Deseasonalize & Detrending	31.0020	33.5933	0.0601	0.0627	1.0189	0.9735	-0.1143	0.0860
rf_cds_dt	Random Forest w/ Cond. Deseasonalize & Detrending	31.9684	34.6774	0.0624	0.0651	1.0506	1.0045	-0.3823	0.0980
dt_cds_dt	Decision Tree w/ Cond. Deseasonalize & Detrending	33.1479	36.0389	0.0664	0.0694	1.0916	1.0463	-0.4206	0.0120
lightgbm_cds_dt	Light Gradient Boosting w/ Cond. Deseasonalize & Detrending	34.5999	37.9918	0.0670	0.0701	1.1409	1.1040	-0.3994	0.0140
ada_cds_dt	AdaBoost w/ Cond. Deseasonalize & Detrending	35.3164	38.1248	0.0674	0.0708	1.1629	1.1065	-0.3512	0.0420
omp_cds_dt	Orthogonal Matching Pursuit w/ Cond. Deseasonalize & Detrending	35.7348	38.6755	0.0706	0.0732	1.1793	1.1250	-0.5095	0.0080
gbr_cds_dt	Gradient Boosting w/ Cond. Deseasonalize & Detrending	36.2416	38.8115	0.0716	0.0749	1.1962	1.1291	-0.5396	0.0280
llar_cds_dt	Lasso Least Angular Regressor w/ Cond. Deseasonalize & Detrending	39.5552	41.9705	0.0749	0.0790	1.3038	1.2194	-0.5759	0.0100
naive	Naive Forecaster	54.8667	59.8160	0.1135	0.1151	1.8145	1.7444	-3.7710	1.4660
snaive	Seasonal Naive Forecaster	53.5333	54.9143	0.1136	0.1211	1.7700	1.5999	-4.1630	0.0040
polytrend	Polynomial Trend Forecaster	70.1138	77.3400	0.1363	0.1468	2.3154	2.2507	-4.6202	0.0060
croston	Croston	79.3645	85.8439	0.1515	0.1684	2.6211	2.4985	-5.2294	0.0040
grand_means	Grand Means Forecaster	216.0214	218.4259	0.4377	0.5682	7.1261	6.3506	-59.2684	1.0980

Analyze Model

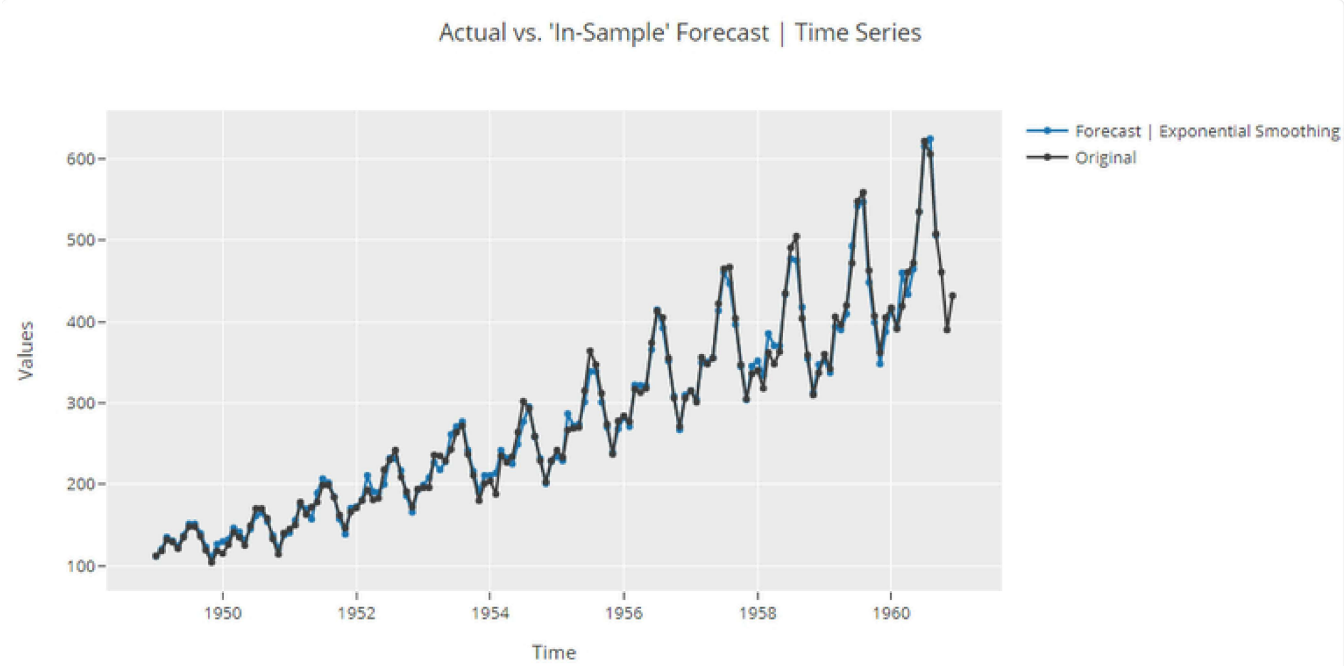
```
# functional API
plot_model(best, plot = 'forecast', data_kwargs = {'fh' : 24})

# OOP API
s.plot_model(best, plot = 'forecast', data_kwargs = {'fh' : 24})
```




```
# functional API
plot_model(best, plot = 'diagnostics')

# OOP API
s.plot_model(best, plot = 'diagnostics')
```




```
# functional API
plot_model(best, plot = 'insample')

# OOP API
s.plot_model(best, plot = 'insample')
```



Predictions

```
# functional API
final_best = finalize_model(best)
predict_model(best, fh = 24)

# OOP API
final_best = s.finalize_model(best)
s.predict_model(best, fh = 24)
```

	y_pred
1960-10	442.8886
1960-11	388.1647
1960-12	427.6613
1961-01	440.5849
1961-02	413.9441
1961-03	460.8743
1961-04	490.1254
1961-05	500.5098
1961-06	567.8292
1961-07	657.2934
1961-08	648.7543
1961-09	540.5014
1961-10	472.0858
1961-11	413.6145
1961-12	455.5483
1962-01	469.1593
1962-02	440.6465
1962-03	490.4450
1962-04	521.4056
1962-05	532.2838
1962-06	603.6872
1962-07	698.5836
1962-08	689.2960
1962-09	574.1031

Save the model

```
# functional API
save_model(final_best, 'my_final_best_model')

# OOP API
s.save_model(final_best, 'my_final_best_model')
```

```
Transformation Pipeline and Model Successfully Saved

(ExponentialSmoothing(damped_trend=False, initial_level=None,
                      initial_seasonal=None, initial_trend=None,
                      initialization_method='estimated', seasonal='mul', sp=12,
                      trend='add', use_boxcox=None),
 'my_final_best_model.pkl')
```

To load the model back in the environment:

```
# functional API
loaded_model = load_model('my_final_best_model')
print(loaded_model)

# OOP API
loaded_model = s.load_model('my_final_best_model')
print(loaded_model)
```

```
ExponentialSmoothing(damped_trend=False, initial_level=None,
                     initial_seasonal=None, initial_trend=None,
                     initialization_method='estimated', seasonal='mul', sp=12,
                     trend='add', use_boxcox=None)
```

Previous
Installation

Next
Tutorials

Last updated 1 year ago