
autotab

Release 0.1

Sara Iftikhar

May 04, 2022

CONTENTS

1	using pip	1
2	using github link	3
3	using setup.py file	5
4	installation options	7
5	quick start	9
5.1	Optimize pipeline for machine learning models (regression)	9
5.2	machine learning models (classification)	10
5.3	deep learning models (regression)	11
5.4	deep learning models (classification)	11
5.5	deep learning models (multi-class classification)	12
6	Frequently Asked Questions	13
6.1	What is difference between parent and child iterations/algorithm?	13
6.2	what splitting scheme is used	13
6.3	Is the pipeline optimized for test data or validation data?	13
6.4	I don't want to optimize preprocessing step	14
6.5	I don't want to optimize hyperparameters of the models	14
6.6	I don't want to optimize model selection	14
6.7	I want to optimize pipeline for only one model	14
6.8	I want to optimize pipeline for only selected models	15
6.9	Can I use different optimization algorithms for parent and child iterations	15
6.10	How to monitor more than one metrics	15
6.11	How to find best/optimized pipeline	15
6.12	Find best pipeline with respect to a specific (performance) metric	16
6.13	Find best pipeline with respect to a particular model	16
6.14	Change search space of a particular model	16
6.15	consider only selected transformations	16
6.16	do not optimize transformations for input data	17
6.17	change number of optimization iterations of a specific model	17
6.18	where are all the results stored	17
6.19	what if optimization stops in the middle	17
6.20	what is config.json file	18
6.21	How to include results from previous runs	18
6.22	What versions of underlying libraries do this package depends	18
6.23	how to use cross validation during pipeline optimization	18
6.24	how to change search space for batch_size and learning rate	19

7	OptimizePipeline	21
8	Examples	33
8.1	regression	33
9	Indices and tables	53
	Index	55

USING PIP

The most easy way to install autotab is using pip

```
pip install autotab
```

However, if you are interested in optimizing pipeline for deep learning models, you can choose to install tensorflow as well by using all option

```
pip install autotab[all]
```

For list of all options see *installation options* options

USING GITHUB LINK

You can use github link for install autotab.

```
python -m pip install git+https://github.com/Sara-Iftikhar/AutoTab.git
```

The latest code however (possibly with less bugs and more features) can be installed from dev branch instead

```
python -m pip install git+https://github.com/Sara-Iftikhar/AutoTab.git@dev
```

To install the latest branch (*dev*) with all requirements use all keyword

```
python -m pip install "AI4Water[all] @ git+https://github.com/Sara-Iftikhar/AutoTab.  
↪git@dev"
```


USING SETUP.PY FILE

go to folder where repository is downloaded

```
python setup.py install
```


INSTALLATION OPTIONS

The `all` option will install tensorflow 2.7 version along with autotab and h5py.

QUICK START

This page describes optimization of pipeline for different problems and using different models.

5.1 Optimize pipeline for machine learning models (regression)

This covers all scikit-learn models, catboost, lightgbm and xgboost

```
>>> from ai4water.datasets import busan_beach
>>> from autotab import OptimizePipeline

>>> data = busan_beach()
>>> input_features = data.columns.tolist()[0:-1]
>>> output_features = data.columns.tolist()[-1:]

>>> pl = OptimizePipeline(
...     inputs_to_transform=input_features,
...     outputs_to_transform=output_features,
...     models=["LinearRegression",
...             "LassoLars",
...             "Lasso",
...             "RandomForestRegressor",
...             "HistGradientBoostingRegressor",
...             "CatBoostRegressor",
...             "XGBRegressor",
...             "LGBMRegressor",
...             "GradientBoostingRegressor",
...             "ExtraTreeRegressor",
...             "ExtraTreesRegressor"
...     ],
...     parent_iterations=30,
...     child_iterations=12,
...     parent_algorithm='bayes',
...     child_algorithm='bayes',
...     eval_metric='mse',
...     monitor=['r2', 'mse'],
...     input_features=input_features,
...     output_features=output_features,
...     split_random=True,
... )
```

(continues on next page)

(continued from previous page)

```
>>> pl.fit(data=data)

>>> pl.post_fit(data=data)
```

5.2 machine learning models (classification)

This covers all scikit-learn models, catboost, lightgbm and xgboost

```
>>> from ai4water.datasets import MtropicsLaos
>>> from autotab import OptimizePipeline

>>> data = MtropicsLaos().make_classification(lookback_steps=1)
>>> input_features = data.columns.tolist()[0:-1]
>>> output_features = data.columns.tolist()[-1:]

>>> pl = OptimizePipeline(
...     mode="classification",
...     eval_metric="accuracy",
...     inputs_to_transform=input_features,
...     outputs_to_transform=output_features,
...     models=["ExtraTreeClassifier",
...             "RandomForestClassifier",
...             "XGBClassifier",
...             "CatBoostClassifier",
...             "LGBMClassifier",
...             "GradientBoostingClassifier",
...             "HistGradientBoostingClassifier",
...             "ExtraTreesClassifier",
...             "RidgeClassifier",
...             "SVC",
...             "KNeighborsClassifier",
...             ],
...     parent_iterations=30,
...     child_iterations=12,
...     parent_algorithm='bayes',
...     child_algorithm='bayes',
...     monitor=['accuracy'],
...     input_features=input_features,
...     output_features=output_features,
...     split_random=True,
... )

>>> pl.fit(data=data)

>>> pl.post_fit(data=data)
```

5.3 deep learning models (regression)

This covers MLP, LSTM, CNN, CNNLSTM, TFT, TCN, LSTMAutoEncoder for regression . Each model can consist of stacks of layers. For example MLP can consist of stacks of Dense layers. The number of layers are also optimized. When using deep learning models, also set the value fo epochs because the default value is 14 which is too small for a deep learning model. Also consider setting values for batch_size and lr.

```
>>> from ai4water.datasets import busan_beach
>>> from autotab import OptimizePipeline

>>> data = busan_beach()
>>> input_features = data.columns.tolist()[0:-1]
>>> output_features = data.columns.tolist()[-1:]

>>> pl = OptimizePipeline(
...     inputs_to_transform=input_features,
...     outputs_to_transform=output_features,
...     models=["MLP", "LSTM", "CNN", "CNNLSTM", "TFT", "TCN", "LSTMAutoEncoder"],
...     parent_iterations=30,
...     child_iterations=12,
...     parent_algorithm='bayes',
...     child_algorithm='bayes',
...     eval_metric='mse',
...     monitor=['r2', 'nse'],
...     input_features=input_features,
...     output_features=output_features,
...     split_random=True,
...     epochs=100,
... )

>>> pl.fit(data=data)

>>> pl.post_fit(data=data)
```

5.4 deep learning models (classification)

This covers MLP, LSTM, CNN, CNNLSTM, TFT, TCN, LSTMAutoEncoder for classification problem. Each model can consist of stacks of layers. For example MLP can consist of stacks of Dense layers. The number of layers are also optimized.

```
>>> from ai4water.datasets import MtropicsLaos
>>> from autotab import OptimizePipeline

>>> data = MtropicsLaos().make_classification(lookback_steps=1,)
>>> input_features = data.columns.tolist()[0:-1]
>>> output_features = data.columns.tolist()[-1:]

>>> pl = OptimizePipeline(
...     mode="classification",
...     eval_metric="accuracy",
...     inputs_to_transform=input_features,
```

(continues on next page)

(continued from previous page)

```
...     outputs_to_transform=output_features,
...     models=["MLP", "CNN"],
...     parent_iterations=30,
...     child_iterations=12,
...     parent_algorithm='bayes',
...     child_algorithm='bayes',
...     monitor=['f1_score'],
...     input_features=input_features,
...     output_features=output_features,
...     split_random=True,
...     epochs=100,
... )

>>> pl.fit(data=data)

>>> pl.post_fit(data=data)
```

5.5 deep learning models (multi-class classification)

For multi-class classification with neural networks, we must set `num_classes` argument to some value greater than 2.

FREQUENTLY ASKED QUESTIONS

6.1 What is difference between parent and child iterations/algorithm?

AutoTab operates based upon parent and child optimization iterations. The parent iteration is responsible for preprocessing step optimization and model optimization. During each parent iteration, when the preprocessing and model is selected/suggested by the algorithm for this iteration, the child optimization loops starts. The job of child optimization loop is to optimize hyperparameters of the selected/suggested model. The user can specify any algorithm from following algorithms for parent and child optimization algorithms.

- bayes
- random
- grid
- bayes_rf
- tpe
- atpe
- cmaes

6.2 what splitting scheme is used

By default it is supposed that the data is split into 3 sets i.e. training, validation and test sets. validation data is only used during pipeline optimization inside `.fit` method while the test data is only used after optimization. If you have only two sets i.e. training and validation, set `fit_on_all_train_data` to `False` during `post_fit`

6.3 Is the pipeline optimized for test data or validation data?

for validation data

6.4 I don't want to optimize preprocessing step

If you don't want any preprocessing steps, keep *inputs_to_transform* and *outputs_to_transform* arguments equal to None or an empty list. In this way transformations will not be optimized for both inputs and targets. As shown in below example,

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(
...     inputs_to_transform=[],
...     outputs_to_transform=[],
... )
>>> results = pl.fit(data=data)
```

6.5 I don't want to optimize hyperparameters of the models

If you don't want to optimize hyperparameters of the models, the child iterations needs to be set to zero. As shown in below example,

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(child_iterations=0)
>>> results = pl.fit(data=data)
```

6.6 I don't want to optimize model selection

If you don't want to optimize model selection, keep *models* argument equals to None or an empty list. As shown in below example,

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(models=[])
>>> results = pl.fit(data=data)
```

6.7 I want to optimize pipeline for only one model

You can set *models* parameter to the desired model. In this way, pipeline will be optimized by using only one model. For example, in the following code, only *AdaBoostRegressor* will be used in pipeline optimization.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(
>>> models=["AdaBoostRegressor"])
>>> results = pl.fit(data=data)
```

6.8 I want to optimize pipeline for only selected models

List the desired models in *models* as a list. In this way, pipeline will be optimized for the selected models.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(
>>> models=[
...     "GradientBoostingRegressor",
...     "HistGradientBoostingRegressor",
...     "DecisionTreeRegressor",
...     "CatBoostRegressor",
...     "ExtraTreeRegressor",
...     "ExtraTreesRegressor",
...     ])
>>> results = pl.fit(data=data)
```

6.9 Can I use different optimization algorithms for parent and child iterations

Different optimization algorithms can be set by *parent_algorithm* and *child_algorithm*.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(
...     parent_algorithm="bayes",
...     child_algorithm="bayes"
... )
>>> results = pl.fit(data=data)
```

6.10 How to monitor more than one metrics

The metrics you want to monitor can be given to *monitor* as a list. In this example, two metrics NSE and R^2 are being monitored.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(monitor=['r2', 'nse'])
>>> results = pl.fit(data=data)
```

6.11 How to find best/optimized pipeline

There are two functions to get best pipeline after optimization. They are *get_best_pipeline_by_metric* which returns optimized pipeline according to given metric. On the other hand, *get_best_pipeline_by_model* gives us best pipeline according to given model.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline()
>>> results = pl.fit(data=data)
```

(continues on next page)

(continued from previous page)

```
>>> pl.get_best_pipeline_by_metric(metric_name='nse')
>>> pl.get_best_pipeline_by_model(model_name='RandomForest_regressor')
```

6.12 Find best pipeline with respect to a specific (performance) metric

get_best_pipeline_by_metric function can be used to get best pipeline with respect to a specific (performance) metric.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline()
>>> results = pl.fit(data=data)
>>> pl.get_best_pipeline_by_metric(metric_name='nse')
```

6.13 Find best pipeline with respect to a particular model

get_best_pipeline_by_model returns the best pipeline with respect to a particular model and performance metric. The metric must be recorded i.e. must be given as *monitor* argument.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline()
>>> results = pl.fit(data=data)
>>> pl.get_best_pipeline_by_model(model_name='RandomForest_regressor')
```

6.14 Change search space of a particular model

update_model_space updates or changes the search space of an already existing model.

```
>>> pl = OptimizePipeline(...)
>>> rf_space = {'max_depth': [5,10, 15, 20],
>>>             'n_models': [5,10, 15, 20]}
>>> pl.update_model_space({"RandomForestRegressor": rf_space})
```

6.15 consider only selected transformations

Selected transformations can be given to *input_transformations* and *output_transformations*. In this way, the given transformations will be used for preprocessing steps.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(
...     input_transformations=['minmax', 'log', 'zscore'],
...     output_transformations=['quantile', 'box-cox', 'yeo-johnson']
... )
>>> results = pl.fit(data=data)
```

6.16 do not optimize transformations for input data

If you don't want to optimize transformations for input data, keep *inputs_to_transform* argument equal to *None* or an empty list. In this way transformations will not be optimized for input data.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(inputs_to_transform=[])
>>> results = pl.fit(data=data)
```

6.17 change number of optimization iterations of a specific model

Number of optimization iterations for a particular model can be changed by using *change_child_iteration* function after initializing the *OptimizePipeline* class. For example we may want to change the child hpo iterations for one or more models. We may want to run only 10 iterations for *LinearRegression* but 40 iterations for *XGBRegressor*. In such a case we can use this function to modify child hpo iterations for one or more models. The iterations for all the remaining models will remain same as defined by the user at the start.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(...)
>>> pl.change_child_iteration({"XGBRegressor": 10})
#If we want to change iterations for more than one models
>>> pl.change_child_iteration({"XGBRegressor": 30,
>>>                             "RandomForestRegressor": 20}))
```

6.18 where are all the results stored

The results are stored in folder named *results* in the current working directory. The exact path of stored results can be checked by printing *model.path*.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(...)
>>> print(pl.path)
```

6.19 what if optimization stops in the middle

If optimization stops in the middle due to an error, remaining results can be saved and analyzed by using these commands.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(...)
>>> pl.fit(data=data)
.. # if above command stops in the middle due to an error
>>> pl.save_results()
>>> pl.post_fit(data=data)
```

6.20 what is config.json file

config.json is a simply plain text file that stores information about pipeline such as parameters, pipeline configuration. The pipeline can be built again by using *from_config_file* method as shown below.

```
>>> from autotab import OptimizePipeline
>>> config_path = "path/to/config.json"
>>> new_pipeline = OptimizePipeline.from_config_file(config_path)
```

6.21 How to include results from previous runs

The path to *iterations.json* from previous pipeline results has to be given to fit function in order to include results from previous runs.

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(...)
>>> fpath = "path/to/previous/iterations.json"
>>> results = pl.fit(data=data, previous_results=fpath)
```

6.22 What versions of underlying libraries do this package depends

Currently *AutoTab* is strongly coupled with a ML python framework *AI4Water*, whose version should be 1.2 or greater. Another dependency is *h5py* which does not have any specific version requirement.

6.23 how to use cross validation during pipeline optimization

By default the pipeline is evaluated on the validation data according to *eval_metric*. However, you can choose to perform cross validation on child or parent or on both iterations. To perform cross validation at parent iterations set *cv_parent_hpo* to True. Similarly to perform cross validation at child iteration, set *cv_child_hpo* to True. You must pass the *cross_validator* argument as well to determine what kind of cross validation to be performed. Consider the following example

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(
...     cv_parent_hpo=True,
...     cross_validator={"KFold": {"n_splits": 5}},
... )
```

Instead of *KFold*, we also choose *LeaveOneOut*, or *ShuffleSplit* or *TimeSeriesSplit*.

6.24 how to change search space for batch_size and learning rate

The learning_rate and batch_size search space is only active for deep learning models i.e. when the category is “DL”. The default search space for learning rate is Real(low=1e-5, high=0.05, num_samples=10, name="lr") while for batch_size, the default search space is [8, 16, 32, 64]. We can change the default search space by making use of change_batch_size_space and change_lr_space methods after class initialization. For example we can achieve a different batch_size search space as below

```
>>> from autotab import OptimizePipeline
>>> pl = OptimizePipeline(
...     ...
...     category="DL"
... )
... pl.change_batch_size_space([32, 64, 128, 256, 512])
```


OPTIMIZEPIPELINE

```
class autotab.OptimizePipeline(inputs_to_transform, input_transformations: Optional[Union[list, dict]] =  
    None, outputs_to_transform=None, output_transformations: Optional[list]  
    = None, models: Optional[list] = None, parent_iterations: int = 100,  
    child_iterations: int = 25, parent_algorithm: str = 'bayes', child_algorithm:  
    str = 'bayes', eval_metric: Optional[str] = None, cv_parent_hpo:  
    Optional[bool] = None, cv_child_hpo: Optional[bool] = None, monitor:  
    Optional[Union[list, str]] = None, mode: str = 'regression', num_classes:  
    Optional[int] = None, category: str = 'ML', prefix: Optional[str] = None,  
    **model_kwargs)
```

optimizes model/estimator, its hyperparameters and preprocessing operation to be performed on input and output features. It consists of two hpo loops. The parent or outer loop optimizes preprocessing/feature engineering, feature selection and model selection while the child hpo loop optimizes hyperparameters of child hpo loop.

- **metrics_**
a pandas DataFrame of shape (parent_iterations, len(monitor)) which contains values of metrics being monitored at each parent iteration.
- **val_scores_**
a 1d numpy array of length equal to parent_iterations which contains value of evaluation metric at each parent iteration.
- **parent_suggestions_**
an ordered dictionary of suggestions to the parent objective function during parent hpo loop
- **child_val_scores_**
a numpy array of shape (parent_iterations, child_iterations) containing value of eval_metric at all child hpo loops
- **optimizer_**
an instance of ai4water.hyperopt.HyperOpt [1] for parent optimization
- **models**
a list of models being considered for optimization
- **model_space**
a dictionary which contains parameter space for each model

Example

```
>>> from autotab import OptimizePipeline
>>> from ai4water.datasets import busan_beach
>>> data = busan_beach()
>>> input_features = data.columns.tolist()[0:-1]
>>> output_features = data.columns.tolist()[-1:]
>>> pl = OptimizePipeline(input_features=input_features,
>>>                        output_features=output_features,
>>>                        inputs_to_transform=input_features)
>>> results = pl.fit(data=data)
```

Note: This optimization always solves a minimization problem even if the val_metric is $SR^{2\$}$.

Undoc-members

Show-inheritance

__init__ (inputs_to_transform, input_transformations: Optional[Union[list, dict]] = None, outputs_to_transform=None, output_transformations: Optional[list] = None, models: Optional[list] = None, parent_iterations: int = 100, child_iterations: int = 25, parent_algorithm: str = 'bayes', child_algorithm: str = 'bayes', eval_metric: Optional[str] = None, cv_parent_hpo: Optional[bool] = None, cv_child_hpo: Optional[bool] = None, monitor: Optional[Union[list, str]] = None, mode: str = 'regression', num_classes: Optional[int] = None, category: str = 'ML', prefix: Optional[str] = None, **model_kwargs)

initializes the class

Parameters

- **inputs_to_transform** (list) – Input features on which feature engineering/transformation is to be applied. By default all input features are considered. If you want to apply a single transformation on a group of input features, then pass this as a dictionary. This is helpful if the input data consists of hundred or thousands of input features.

- **input_transformations** (list, dict) – The transformations to be considered for input features. Default is None, in which case all input features are considered.

If list, then it will be the names of transformations to be considered for all input features. By default following transformations are considered

- minmax rescale from 0 to 1
- center center the data by subtracting mean from it
- scale scale the data by dividing it with its standard deviation
- zscore first performs centering and then scaling
- box-cox
- yeo-johnson
- quantile
- robust
- log

- log2
- log10
- sqrt square root

The user can however, specify list of transformations to be considered for each input feature. In such a case, this argument must be a dictionary whose keys are names of input features and values are list of transformations.

- **outputs_to_transform** (*list, optional*) – Output features on which feature engineering/transformation is to be applied. If None, then transformations on outputs are not applied.
- **output_transformations** – The transformations to be considered for outputs/targets. The user can consider any transformation as given for **input_transformations**
- **models** (*list, optional*) – The models/algorithms to consider during optimization. If not given, then all available models from sklearn, xgboost, catboost and lgbm are considered. For neural networks, following 6 model types are considered by default
 - MLP ^[1] multi layer perceptron
 - CNN² 1D convolution neural network
 - LSTM³ Long short term memory network
 - CNNLSTM⁴ CNN-> LSTM
 - LSTMAutoEncoder⁵ LSTM based autoencoder
 - TCN⁶ Temporal convolution networks
 - TFT⁷ Temporal fusion Transformer

However, in such cases, the **category** must be DL.

- **parent_iterations** (*int, optional (default=100)*) – Number of iterations for parent optimization loop
- **child_iterations** (*int, optional*) – Number of iterations for child optimization loop. It set to 0, the child hpo loop is not run which means the hyperparameters of the model are not optimized. You can customize iterations for each model by making using of :meth: *change_child_iterations* method.
- **parent_algorithm** (*str, optional*) – Algorithm for optimization of parent optimization
- **child_algorithm** (*str, optional*) – Algorithm for optimization of child optimization
- **eval_metric** (*str, optional*) – Validation metric to calculate val_score in objective function. The parent and child hpo loop optimizes/improves this metric. This metric is calculated on validation data. If cross validation is performed then this metric is calculated using cross validation.
- **cv_parent_hpo** (*bool, optional (default=False)*) – Whether we want to apply cross validation in parent hpo loop or not?. If given, the parent hpo loop will optimize the cross validation score. The model is fitted on whole training data (training+validation)

² <https://ai4water.readthedocs.io/en/latest/models/models.html#ai4water.models.CNN>

³ <https://ai4water.readthedocs.io/en/latest/models/models.html#ai4water.models.LSTM>

⁴ <https://ai4water.readthedocs.io/en/latest/models/models.html#ai4water.models.CNNLSTM>

⁵ <https://ai4water.readthedocs.io/en/latest/models/models.html#ai4water.models.LSTMAutoEncoder>

⁶ <https://ai4water.readthedocs.io/en/latest/models/models.html#ai4water.models.TCN>

⁷ <https://ai4water.readthedocs.io/en/latest/models/models.html#ai4water.models.TFT>

after cross validation and the metrics printed (other than `parent_val_metric`) are calculated on the based the updated model i.e. the one fitted on whole training (training+validation) data.

- **cv_child_hpo** (*bool, optional (default=False)*) – Whether we want to apply cross validation in child hpo loop or not?. If False, then `val_score` will be calculated on validation data. The type of cross validator used is taken from `model.config['cross_validator']`
- **monitor** (*Union[str, list], optional, (default=None)*) – Names of performance metrics to monitor in parent hpo loop. If None, then R2 is monitored for regression and accuracy for classification.
- **mode** (*str, optional (default="regression")*) – whether this is a regression problem or classification
- **num_classes** (*int, optional (default=None)*) – number of classes, only relevant if `mode=="classification"`.
- **category** (*str, optional (default="DL")*) – either “DL” or “ML”. If DL, the pipeline is optimized for neural networks.
- ****model_kwargs** – any additional key word arguments for ai4water’s Model

References

_build_model (*model: dict, val_metric: str, x_transformation, y_transformation, prefix: Optional[str], verbosity: int = 0, batch_size: int = 32, lr: float = 0.001*) → ai4water.main.Model

build the ai4water Model. When overwriting this method, the user must return an instance of ai4water’s **Model** class. `batch_size` : only used when category is “DL”. `lr` : only used when category is “DL”

add_dl_model (*model: Callable, space: Union[list, ai4water.hyperopt._space.Real, ai4water.hyperopt._space.Categorical, ai4water.hyperopt._space.Integer]*) → None

adds a deep learning model to be considered.

Parameters

- **model** (*callable*) – the model to be added
- **space** (*list*) – the search space of the model

add_model (*model: dict*) → None

adds a new model which will be considered during optimization.

Parameters model (*dict*) – a dictionary of length 1 whose value should also be a dictionary of parameter space for that model

Example

```
>>> pl = OptimizePipeline(...)
>>> pl.add_model({"XGBRegressor": {"n_estimators": [100, 200, 300, 400, 500]}})
```

baseline_results (*x=None, y=None, data=None, test_data=None, fit_on_all_train_data: bool = True*) → tuple

Returns default performance of all models.

It runs all the models with their default parameters and without any x and y transformation. These results can be considered as baseline results and can be compared with optimized model’s results. The model is trained on ‘training’+‘validation’ data.

Parameters

- **x** – the input data for training
- **y** – the target data for training
- **data** – raw unprepared and unprocessed data from which x,y pairs for both training and test will be prepared. It is only required if x, y are not provided.
- **test_data** – a tuple/list of length 2 whose first element is x and second value is y. The is the data on which the performance of optimized pipeline will be calculated. This should only be given if data argument is not given.
- **fit_on_all_train_data** (*bool, optional (default=True)*) – If true, the model is trained on (training+validation) data. This is based on supposition that the data is split into training, validation and test sets. The optimization of pipeline was performed on validation data. But now, we are training the model on all available training data which is (training + validation) data. If False, then model is trained only on training data.

Returns a tuple of two dictionaries. - a dictionary of val_scores on test data for each model - a dictionary of metrics being monitored for each model on test data.

Return type tuple

be_best_model_from_config(*x=None, y=None, data=None, test_data=None, metric_name: Optional[str] = None, model_name: Optional[str] = None, verbosity=1*) → ai4water.main.Model

Build and Evaluate the best model with respect to metric *from config*.

Parameters

- **x** – the input data for training
- **y** – the target data for training
- **data** – raw unprepared and unprocessed data from which x,y pairs for both training and test will be prepared. It is only required if x, y are not provided.
- **test_data** – a tuple/list of length 2 whose first element is x and second value is y. The is the data on which the performance of optimized pipeline will be calculated. This should only be given if data argument is not given.
- **metric_name** (*str*) – the metric with respect to which the best model is fetched and then built/evaluated. If not given, the best model is built/evaluated with respect to evaluation metric.
- **model_name** (*str, optional*) – If given, the best version of this model will be fetched and built. The ‘best’ will be decided based upon *metric_name*
- **verbosity** (*int, optional (default=1)*) – determines the amount of print information

Return type an instance of trained ai4water Model

bfe_all_best_models(*x=None, y=None, data=None, test_data=None, metric_name: Optional[str] = None, fit_on_all_train_data: bool = True, verbosity: int = 0*) → None

builds, trains and evaluates best versions of all the models. The model is trained on ‘training’+‘validation’ data.

Parameters

- **x** – the input data for training
- **y** – the target data for training

- **data** – raw unprepared and unprocessed data from which x,y pairs for both training and test will be prepared. It is only required if x, y are not provided.
- **test_data** – a tuple/list of length 2 whose first element is x and second value is y. The is the data on which the performance of optimized pipeline will be calculated. This should only be given if data argument is not given.
- **metric_name** (*str*) – the name of metric to determine best version of a model. If not given, parent_val_metric will be used.
- **fit_on_all_train_data** (*bool*, *optional* (*default=True*)) – If true, the model is trained on (training+validation) data. This is based on supposition that the data is split into training, validation and test sets. The optimization of pipeline was performed on validation data. But now, we are training the model on all available training data which is (training + validation) data. If False, then model is trained only on training data.
- **verbosity** (*int*, *optional* (*default=0*)) – determines the amount of print information

Return type None

bfe_best_model_from_scratch(*x=None, y=None, data=None, test_data=None, metric_name: Optional[str] = None, model_name: Optional[str] = None, fit_on_all_train_data: bool = True, verbosity: int = 1*) → ai4water.main.Model

Builds, Trains and Evaluates the **best model** with respect to metric from scratch. The model is trained on 'training'+ 'validation' data. Running this method will also populate `taylor_plot_data_` dictionary.

Parameters

- **x** – the input data for training
- **y** – the target data for training
- **data** – raw unprepared and unprocessed data from which x,y pairs for both training and test will be prepared. It is only required if x, y are not provided.
- **test_data** – a tuple/list of length 2 whose first element is x and second value is y. The is the data on which the performance of optimized pipeline will be calculated. This should only be given if data argument is not given.
- **metric_name** (*str*) – the metric with respect to which the best model is searched and then built/trained/evaluated. If None, the best model is chosen based on the evaluation metric.
- **model_name** (*str*, *optional*) – If given, the best version of this model will be found and built. The 'best' will be decided based upon *metric_name*
- **fit_on_all_train_data** (*bool*, *optional* (*default=True*)) – If true, the model is trained on (training+validation) data. This is based on supposition that the data is split into training, validation and test sets. The optimization of pipeline was performed on validation data. But now, we are training the model on all available training data which is (training + validation) data. If False, then model is trained only on training data.
- **verbosity** (*int*, *optional* (*default=1*)) – determines amount of information to be printed.

Return type an instance of trained ai4water Model

bfe_model_from_scratch(*iter_num: int, x=None, y=None, data=None, test_data=None, fit_on_all_train_data: bool = True*) → ai4water.main.Model

Builds, trains and evaluates the model from a specific iteration. The model is trained on 'training'+ 'validation' data.

Parameters

- **iter_num** (*int*) – iteration number from which to choose the model
- **x** – the input data for training
- **y** – the target data for training
- **data** – raw unprepared and unprocessed data from which x,y pairs for both training and test will be prepared. It is only required if x, y are not provided.
- **test_data** – a tuple/list of length 2 whose first element is x and second value is y. The is the data on which the performance of optimized pipeline will be calculated. This should only be given if data argument is not given.
- **fit_on_all_train_data** (*bool, optional (default=True)*) – If true, the model is trained on (training+validation) data. This is based on supposition that the data is split into training, validation and test sets. The optimization of pipeline was performed on validation data. But now, we are training the model on all available training data which is (training + validation) data. If False, then model is trained only on training data.

Return type an instance of trained ai4water Model

change_child_iteration(*model: dict*)

We may want to change the child hpo iterations for one or more models. For example we may want to run only 10 iterations for LinearRegression but 40 iterations for XGBRegressor. In such a case we can use this function to modify child hpo iterations for one or more models. The iterations for all the remaining models will remain same as defined by the user at the start. This method updated `_child_iters` dictionary

Parameters **model** (*dict*) – a dictionary whose keys are names of models and values are number of iterations for that model during child hpo

Example

```
>>> pl = OptimizePipeline(...)
>>> pl.change_child_iteration({"XGBRegressor": 10})
If we want to change iterations for more than one models
>>> pl.change_child_iteration({"XGBRegressor": 30,
>>>                             "RandomForestRegressor": 20}))
```

compare_models(*metric_name: Optional[str] = None, plot_type: str = 'circular', show: bool = False, **kwargs*) → `matplotlib.axes._axes.Axes`

Compares all the models with respect to a metric and plots a bar plot.

metric_name [str, optional] The metric with respect to which to compare the models.

plot_type [str, optional] if “circular” then `easy_mpl.circular_bar_plot` is drawn otherwise a simple `bar_plot` is drawn.

show [bool, optional] whether to show the plot or not

****kwargs** : keyword arguments for `easy_mpl.circular_bar_plot` or `easy_mpl.bar_chart`

Return type `matplotlib.pyplot.Axes`

config() → `dict`

Returns a dictionary which contains all the information about the class and from which the class can be created.

Returns a dictionary with two keys `init_paras` and `runtime_paras` and `version_info`.

Return type dict

dumbbell_plot(*x=None, y=None, data=None, test_data=None, metric_name: Optional[str] = None, fit_on_all_train_data: bool = True, figsize: Optional[tuple] = None, show: bool = True, save: bool = True*) → matplotlib.axes._axes.Axes

Generate **Dumbbell** plot as comparison of baseline models with optimized models. Not that this command will train all the considered models, so this can be expensive.

Parameters

- **x** – the input data for training
- **y** – the target data for training
- **data** – raw unprepared and unprocessed data from which x,y pairs for both training and test will be prepared. It is only required if x, y are not provided.
- **test_data** – a tuple/list of length 2 whose first element is x and second value is y. The is the data on which the performance of optimized pipeline will be calculated. This should only be given if **data** argument is not given.
- **metric_name** (*str*) – The name of metric with respect to which the models have to be compared. If not given, the evaluation metric is used.
- **fit_on_all_train_data** (*bool, optional (default=True)*) – If true, the model is trained on (training+validation) data. This is based on supposition that the data is split into training, validation and test sets. The optimization of pipeline was performed on validation data. But now, we are training the model on all available training data which is (training + validation) data. If False, then model is trained only on training data.
- **figsize** (*tuple*) – If given, plot will be generated of this size.
- **show** (*bool*) – whether to show the plot or not
- **save** – By default True. If False, function will not save the resultant plot in current working directory.

Return type matplotlib Axes

fit(*x: Optional[numpy.ndarray] = None, y: Optional[numpy.ndarray] = None, data: Optional[pandas.core.frame.DataFrame] = None, validation_data: Optional[Tuple[numpy.ndarray, numpy.ndarray]] = None, previous_results: Optional[dict] = None, process_results: bool = True*) → ai4water.hyperopt._main.HyperOpt

Optimizes the pipeline for the given data.

Parameters

- **x** (*np.ndarray*) – input training data
- **y** (*np.ndarray*) – output/target/label data. It must of same length as x.
- **data** – A pandas dataframe which contains input (x) and output (y) features Only required if x and y are not given. The training and validation data will be extracted from this data.
- **validation_data** – validation data on which pipeline is optimized. Only required if data is not given.
- **previous_results** (*dict, optional*) – path of file which contains xy values.
- **process_results** (*bool*) –

Returns

- an instance of `ai4water.hyperopt.HyperOpt` class which is used for
- optimization.

classmethod `from_config`(*config: dict*) → *autotab._main.OptimizePipeline*

Builds the class from config dictionary

Parameters `config` (*dict*) – a dictionary which contains *init_paras* key.

Return type an instance of `OptimizePipeline` class

classmethod `from_config_file`(*config_file: str*) → *autotab._main.OptimizePipeline*

Builds the class from config file.

Parameters `config_file` (*str*) – complete path of config file which has .json extension

Return type an instance of `OptimizePipeline` class

get_best_metric(*metric_name: str*) → float

returns the best value of a particular performance metric. The metric must be recorded i.e. must be given as *monitor* argument.

Parameters `metric_name` (*str*) – Name of performance metric

Returns the best value of performance metric achieved

Return type float

get_best_metric_iteration(*metric_name: Optional[str] = None*) → int

returns iteration of the best value of a particular performance metric.

Parameters `metric_name` (*str, optional*) – The metric must be recorded i.e. must be given as *monitor* argument. If not given, then evaluation metric is used.

get_best_pipeline_by_metric(*metric_name: Optional[str] = None*) → dict

returns the best pipeline with respect to a particular performance metric.

Parameters `metric_name` (*str, optional*) – The name of metric whose best value is to be retrieved. The metric must be recorded i.e. must be given as *monitor*.

Returns

a dictionary with following keys

- `path` path where the model is saved on disk
- `model_name` name of model
- `x_transformations` transformations for the input data
- `y_transformations` transformations for the target data
- `iter_num` iteration number on which this pipeline was achieved

Return type dict

get_best_pipeline_by_model(*model_name: str, metric_name: Optional[str] = None*) → tuple

returns the best pipeline with respect to a particular model and performance metric. The metric must be recorded i.e. must be given as *monitor* argument.

Parameters

- `model_name` (*str*) – The name of model for which best pipeline is to be found. The *best* is defined by `metric_name`.

- **metric_name** (*str*, *optional*) – The name of metric with respect to which the best model is to be retrieved. If not given, the best model is defined by the evaluation metric.

Returns

a tuple of length two

- **first value is a float which represents the value of** `metric`
- second value is a dictionary of pipeline with four keys

`x_transformation y_transformation model path iter_num`

Return type `tuple`

post_fit (*x=None, y=None, data=None, test_data: Optional[Union[list, tuple]] = None, fit_on_all_train_data: bool = True, show: bool = True*) → `None`

post processing of results to draw dumbbell plot and taylor plot.

Parameters

- **x** – the input data for training
- **y** – the target data for training
- **data** – raw unprepared and unprocessed data from which x,y pairs for both training and test will be prepared. It is only required if x, y are not provided.
- **test_data** – a tuple/list of length 2 whose first element is x and second value is y. The is the data on which the performance of optimized pipeline will be calculated. This should only be given if data argument is not given.
- **fit_on_all_train_data** (*bool, optional (default=True)*) – If true, the model is trained on (training+validation) data. This is based on supposition that the data is split into training, validation and test sets. The optimization of pipeline was performed on validation data. But now, we are training the model on all available training data which is (training + validation) data. If False, then model is trained only on training data.
- **show** (*bool, optional (default=True)*) – whether to show the plots or not

Return type `None`

remove_model (*models: Union[str, list]*) → `None`

removes an model/models from being considered. The following attributes are updated.

- `models`
- `model_space`
- `_child_iters`

Parameters **models** (*list, str*) – name or names of model to be removed.

Example

```
>>> pl = OptimizePipeline(...)
>>> pl.remove_model("ExtraTreeRegressor")
```

report(*write: bool = True*) → str

makes the report and writes it in text form

save_results() → None

saves the results. It is called automatically at the end of optimization. It saves tried models and transformations at each step as json file with the name `parent_suggestions.json`.

An `errors.csv` file is saved which contains validation performance of the models at each optimization iteration with respect to all metrics being monitored.

The performance of each model during child optimization iteration is saved as a csv file with the name `child_val_scores.csv`.

The global seeds for parent and child iterations are also saved in csv files with name `parent_seeds.csv` and `child_seeds.csv`. All of these results are saved in `pl.path` folder.

Return type None

taylor_plot(*x=None, y=None, data=None, test_data=None, fit_on_all_train_data: bool = True, plot_bias: bool = True, figsize: Optional[tuple] = None, show: bool = True, save: bool = True, verbosity: int = 0, **kwargs*) → matplotlib.figure.Figure

makes Taylor's plot using the best version of each model. The number of models in taylor plot will be equal to the number of models which have been considered by the model.

Parameters

- **x** – the input data for training
- **y** – the target data for training
- **data** – raw unprepared and unprocessed data from which x,y pairs for both training and test will be prepared. It is only required if x, y are not provided.
- **test_data** – a tuple/list of length 2 whose first element is x and second value is y. This is the data on which the performance of optimized pipeline will be calculated. This should only be given if data argument is not given.
- **fit_on_all_train_data** (*bool, optional (default=True)*) – If true, the model is trained on (training+validation) data. This is based on supposition that the data is split into training, validation and test sets. The optimization of pipeline was performed on validation data. But now, we are training the model on all available training data which is (training + validation) data. If False, then model is trained only on training data.
- **plot_bias** (*bool, optional*) – whether to plot the bias or not
- **figsize** (*tuple, optional*) – a tuple determining figure size
- **show** (*bool, optional*) – whether to show the plot or not
- **save** (*bool, optional*) – whether to save the plot or not
- **verbosity** (*int, optional (default=0)*) – determines the amount of print information
- ****kwargs** – any additional keyword arguments for taylor_plot function of `easy_mpl`.

Return type matplotlib.pyplot.Figure

update_model_space(*space: dict*) → None

updates or changes the search space of an already existing model

Parameters **space** – a dictionary whose keys are names of models and values are parameter space for that model.

Return type None

Example

```
>>> pl = OptimizePipeline(...)
>>> rf_space = {'max_depth': [5,10, 15, 20],
>>>              'n_models': [5,10, 15, 20]}
>>> pl.update_model_space({"RandomForestRegressor": rf_space})
```

EXAMPLES

Below is a gallery of examples

8.1 regression

```
from ai4water.datasets import busan_beach
from skopt.plots import plot_objective
from autotab import OptimizePipeline

data = busan_beach()

pl = OptimizePipeline(
    inputs_to_transform=data.columns.tolist()[0:-1],
    outputs_to_transform=data.columns.tolist()[-1:],
    parent_iterations=30,
    child_iterations=0, # don't optimize hyperparamters only for demonstration
    parent_algorithm='bayes',
    child_algorithm='random',
    eval_metric='mse',
    monitor=['r2', 'r2_score'],
    models=[ "LinearRegression",
             "LassoLars",
             "Lasso",
             "RandomForestRegressor",
             "HistGradientBoostingRegressor",
             "CatBoostRegressor",
             "XGBRegressor",
             "LGBMRegressor",
             "GradientBoostingRegressor",
             "ExtraTreeRegressor",
             "ExtraTreesRegressor"
           ],

    input_features=data.columns.tolist()[0:-1],
    output_features=data.columns.tolist()[-1:],
    split_random=True,
)

results = pl.fit(data=data, process_results=False)
```

Out:

```

/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳ packages/sklearn/experimental/enable_hist_gradient_boosting.py:17: UserWarning: Since
↳ version 1.0, it is not needed to import enable_hist_gradient_boosting anymore.
↳ HistGradientBoostingClassifier and HistGradientBoostingRegressor are now stable and
↳ can be normally imported from sklearn.ensemble.
"Since version 1.0, "
Iter   mse                r2                r2_score                mse
WARNING:tensorflow:From /home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/
↳ latest/lib/python3.7/site-packages/ai4water/utils/utils.py:1685: is_gpu_available
↳ (from tensorflow.python.framework.test_util) is deprecated and will be removed in a
↳ future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳ py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
0      2.73e+15          0.2340123          0.02512779
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳ packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
    x = um.multiply(x, x, out=x)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳ py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
1
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳ py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
2
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳ py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
3              0.4546905
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳ py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
4
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳ packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
    x = um.multiply(x, x, out=x)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳ packages/sklearn/linear_model/_base.py:138: FutureWarning: The default of 'normalize'
↳ will be set to False in version 1.2 and deprecated in version 1.4.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing
↳ stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LassoLars())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to
↳ each step of the pipeline as follows:

```

(continues on next page)

(continued from previous page)

```

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
FutureWarning,
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
5
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
6
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/sklearn/linear_model/_coordinate_descent.py:648: ConvergenceWarning:
↳Objective did not converge. You might want to increase the number of iterations, check
↳the scale of the features or consider increasing regularisation. Duality gap: 8.
↳223e+14, tolerance: 2.710e+11
    coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
7
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
    x = um.multiply(x, x, out=x)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/numpy/core/_methods.py:244: RuntimeWarning: overflow encountered in reduce
    ret = umr_sum(x, axis, dtype, out, keepdims=keepdims, where=where)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
    x = um.multiply(x, x, out=x)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
8
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
9    2.71e+15                                0.03148921
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
10   2.69e+15                                0.03712889
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
11
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/sklearn/linear_model/_base.py:138: FutureWarning: The default of 'normalize'
↳will be set to False in version 1.2 and deprecated in version 1.4.

```

(continues on next page)

(continued from previous page)

If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the previous behavior:

```
from sklearn.pipeline import make_pipeline
```

```
model = make_pipeline(StandardScaler(with_mean=False), LassoLars())
```

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to each step of the pipeline as follows:

```
kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)
```

```
Set parameter alpha to: original_alpha * np.sqrt(n_samples).
```

```
FutureWarning,
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
12
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
    x = um.multiply(x, x, out=x)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
packages/numpy/core/_methods.py:244: RuntimeWarning: overflow encountered in reduce
    ret = umr_sum(x, axis, dtype, out, keepdims=keepdims, where=where)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
    x = um.multiply(x, x, out=x)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
13
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
14    2.53e+15                                0.09695015
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
    x = um.multiply(x, x, out=x)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
packages/numpy/core/_methods.py:244: RuntimeWarning: overflow encountered in reduce
    ret = umr_sum(x, axis, dtype, out, keepdims=keepdims, where=where)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
15
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
16
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
py:2472: RuntimeWarning: All-NaN axis encountered
```

(continues on next page)

(continued from previous page)

```

    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
17         0.5653205
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
18
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/sklearn/preprocessing/_data.py:3218: RuntimeWarning: overflow encountered in
↳power
    out[pos] = (np.power(x[pos] + 1, lambda) - 1) / lambda
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
    x = um.multiply(x, x, out=x)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/numpy/core/_methods.py:244: RuntimeWarning: overflow encountered in reduce
    ret = umr_sum(x, axis, dtype, out, keepdims=keepdims, where=where)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
19
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/sklearn/base.py:444: UserWarning: X has feature names, but PowerTransformer
↳was fitted without feature names
    f"X has feature names, but {self.__class__.__name__} was fitted without"
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/sklearn/base.py:444: UserWarning: X has feature names, but PowerTransformer
↳was fitted without feature names
    f"X has feature names, but {self.__class__.__name__} was fitted without"
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
20
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/sklearn/base.py:444: UserWarning: X has feature names, but
↳QuantileTransformer was fitted without feature names
    f"X has feature names, but {self.__class__.__name__} was fitted without"
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/sklearn/base.py:444: UserWarning: X has feature names, but
↳QuantileTransformer was fitted without feature names
    f"X has feature names, but {self.__class__.__name__} was fitted without"
/home/docs/checkouts/readthedocs.org/user_builds/autotab/checkouts/latest/autotab/_main.
↳py:2472: RuntimeWarning: All-NaN axis encountered
    best_so_far = func(self.metrics_best_.loc[:self.parent_iter_, _metric])
21     5.4e+13         0.6369561         0.4878222         5.40484e+13
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
    x = um.multiply(x, x, out=x)
22
23
24         0.7440143
25
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳packages/sklearn/preprocessing/_data.py:3218: RuntimeWarning: overflow encountered in
↳power

```

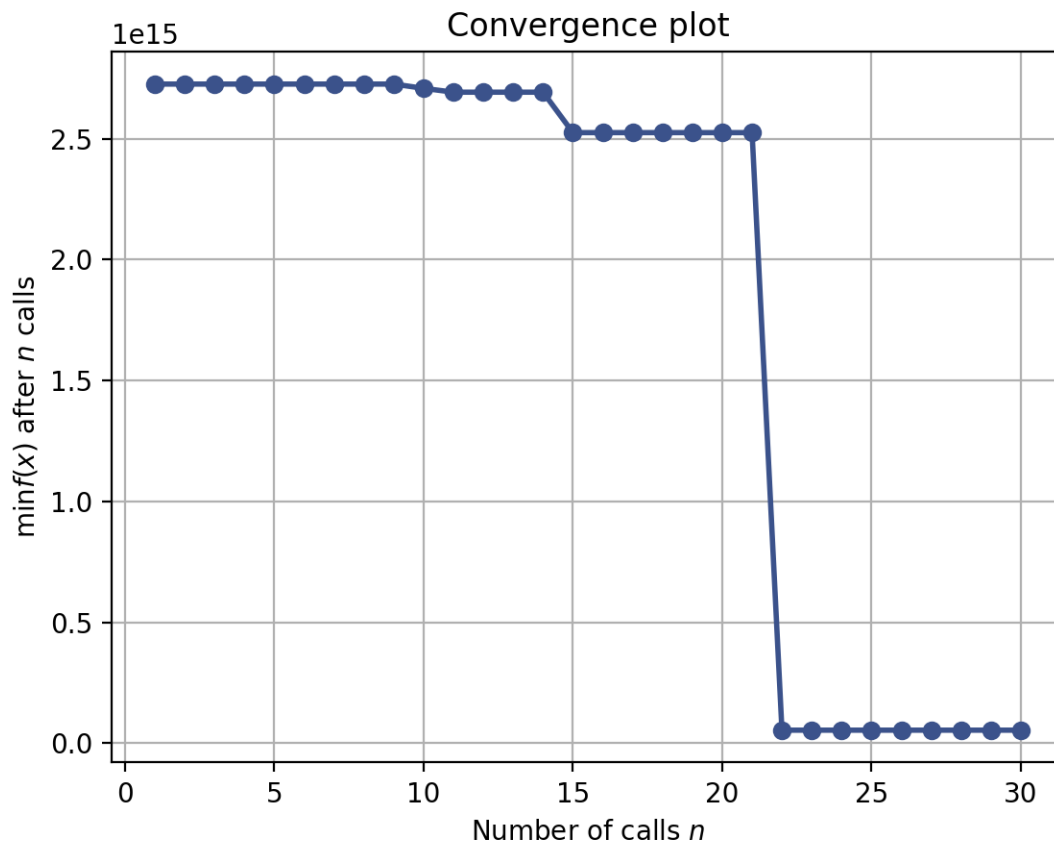
(continued from previous page)

```

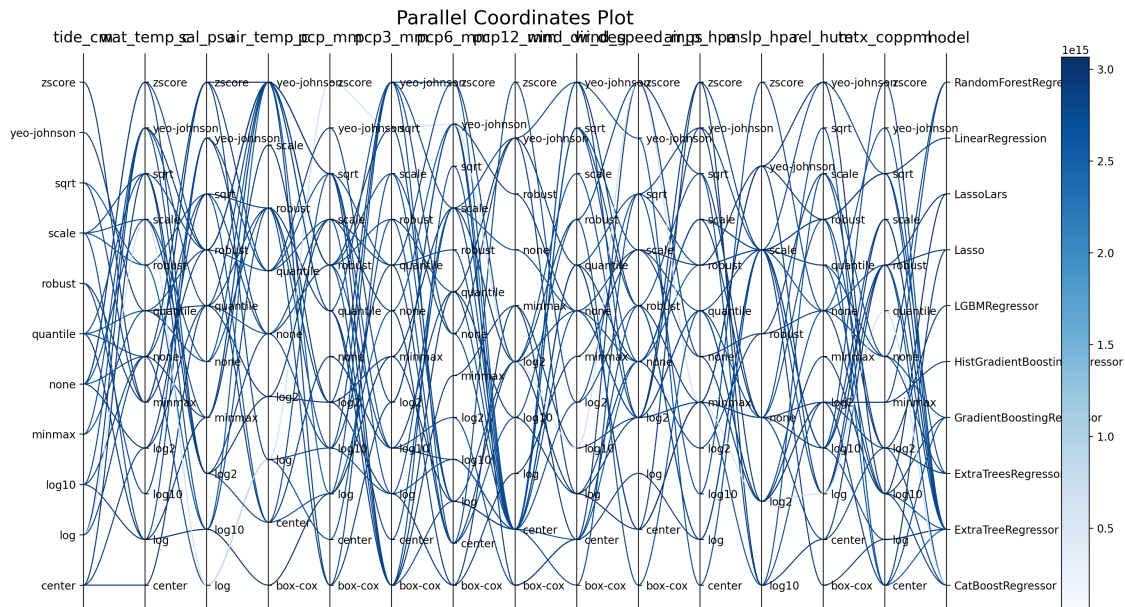
    out[pos] = (np.power(x[pos] + 1, lambda) - 1) / lambda
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳ packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
    x = um.multiply(x, x, out=x)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳ packages/numpy/core/_methods.py:244: RuntimeWarning: overflow encountered in reduce
    ret = umr_sum(x, axis, dtype, out, keepdims=keepdims, where=where)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳ packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
    x = um.multiply(x, x, out=x)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
↳ packages/numpy/core/_methods.py:244: RuntimeWarning: overflow encountered in reduce
    ret = umr_sum(x, axis, dtype, out, keepdims=keepdims, where=where)
26                                0.8295779
27
28
29

```

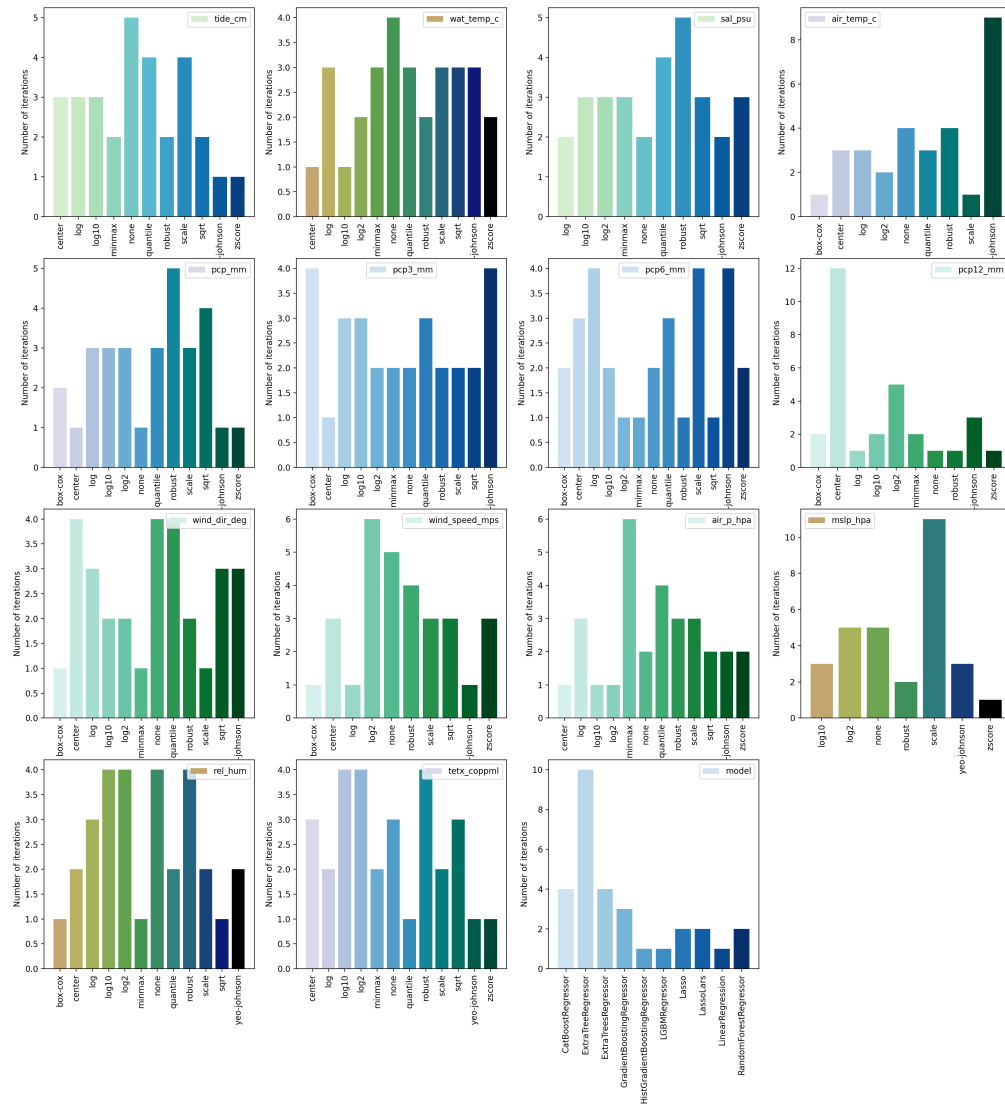
```
pl.optimizer._plot_convergence(save=False)
```



```
pl.optimizer._plot_parallel_coords(figsize=(16, 8), save=False)
```



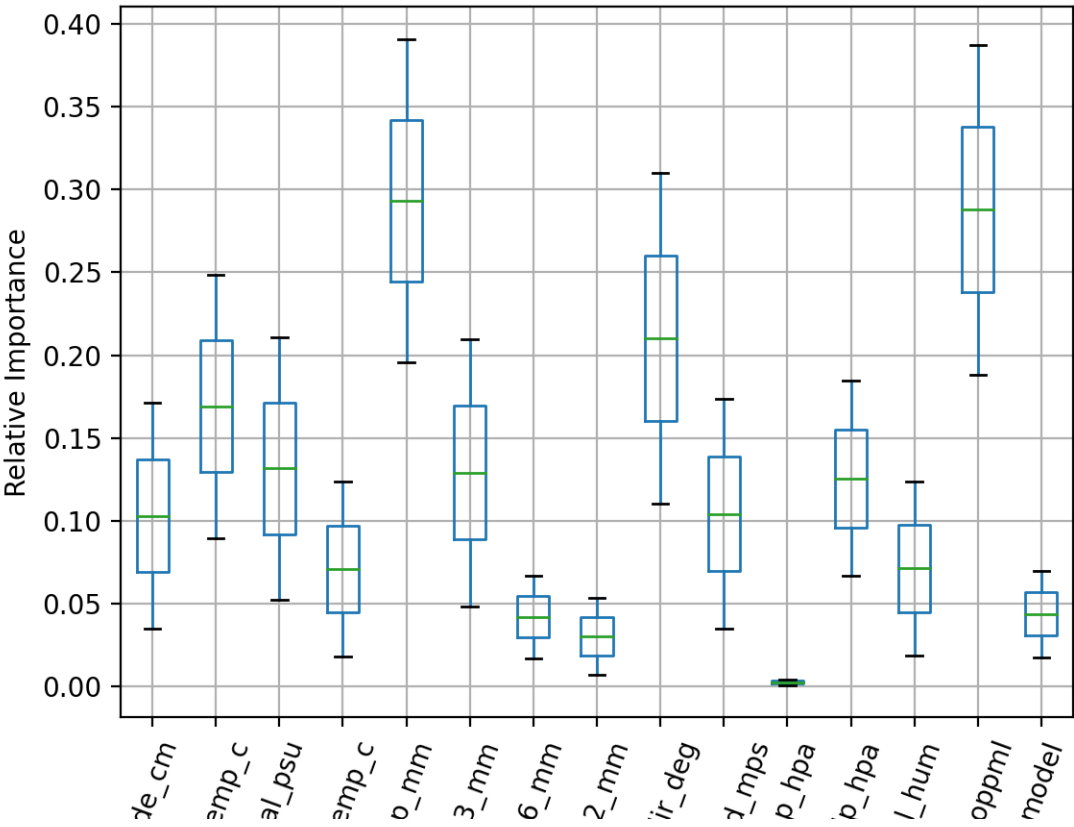
```
pl.optimizer._plot_distributions(save=False)
```



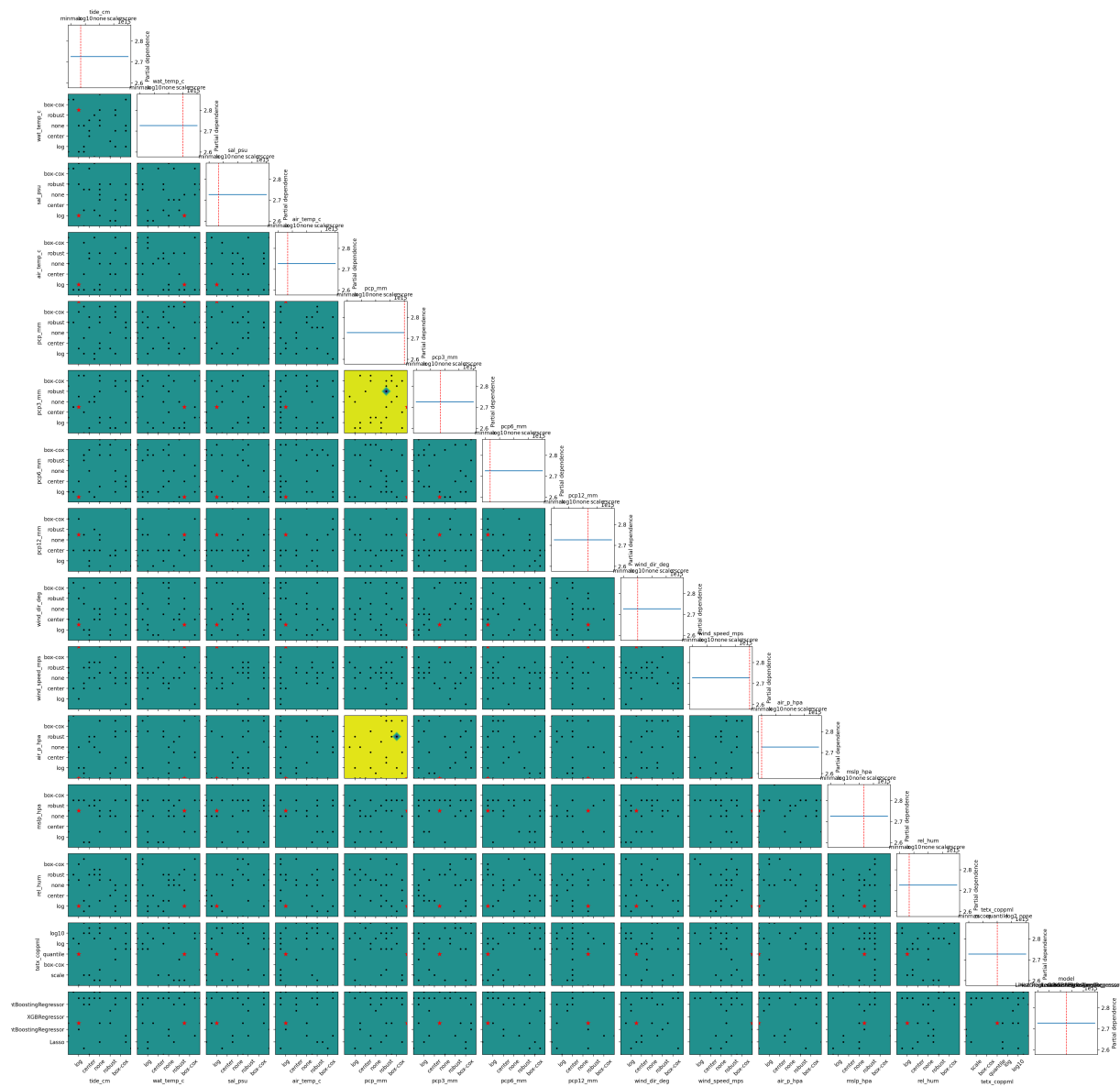
Out:

```
<Figure size 2100x2100 with 16 Axes>

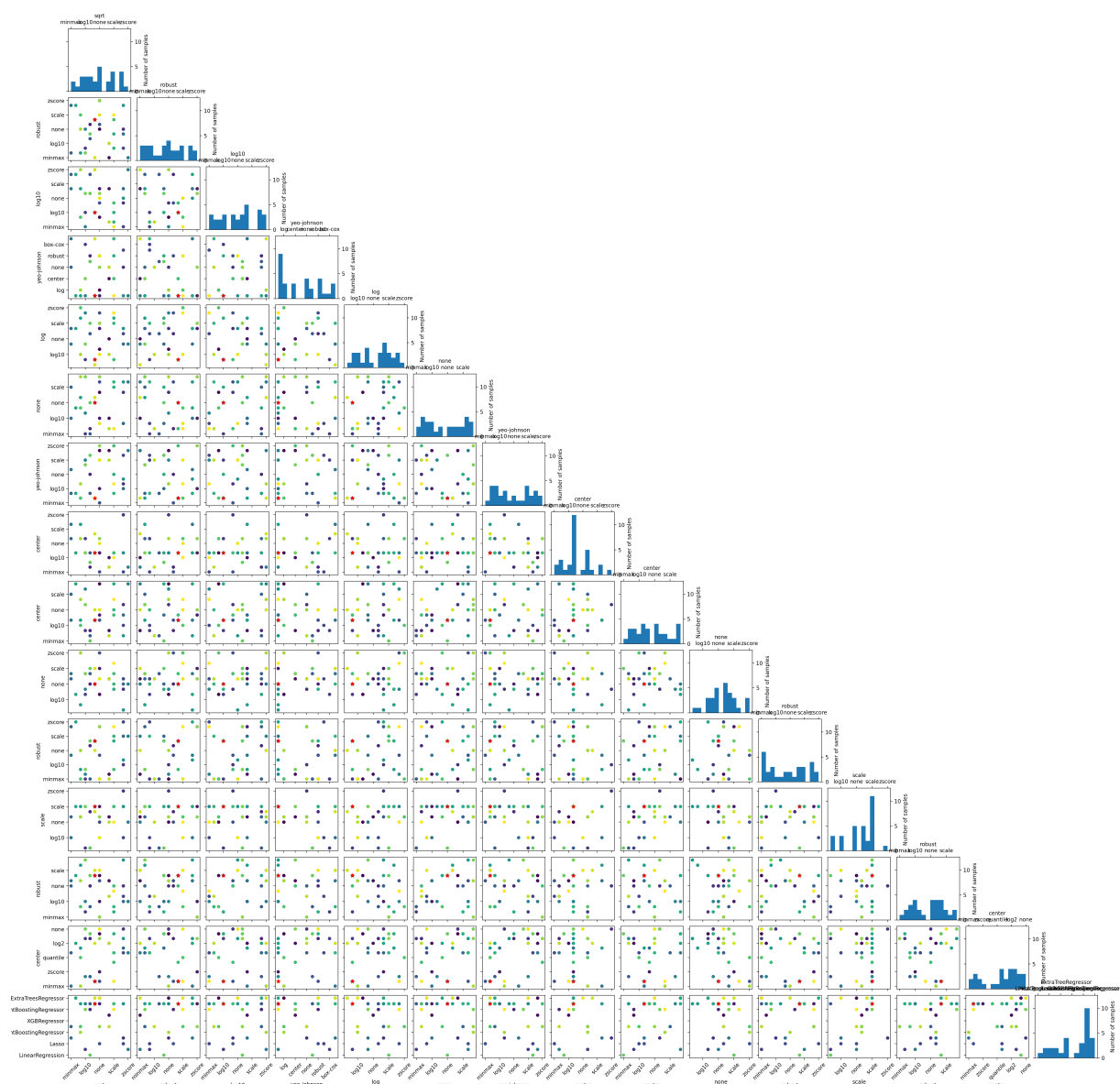
pl.optimizer_.plot_importance(save=False)
```



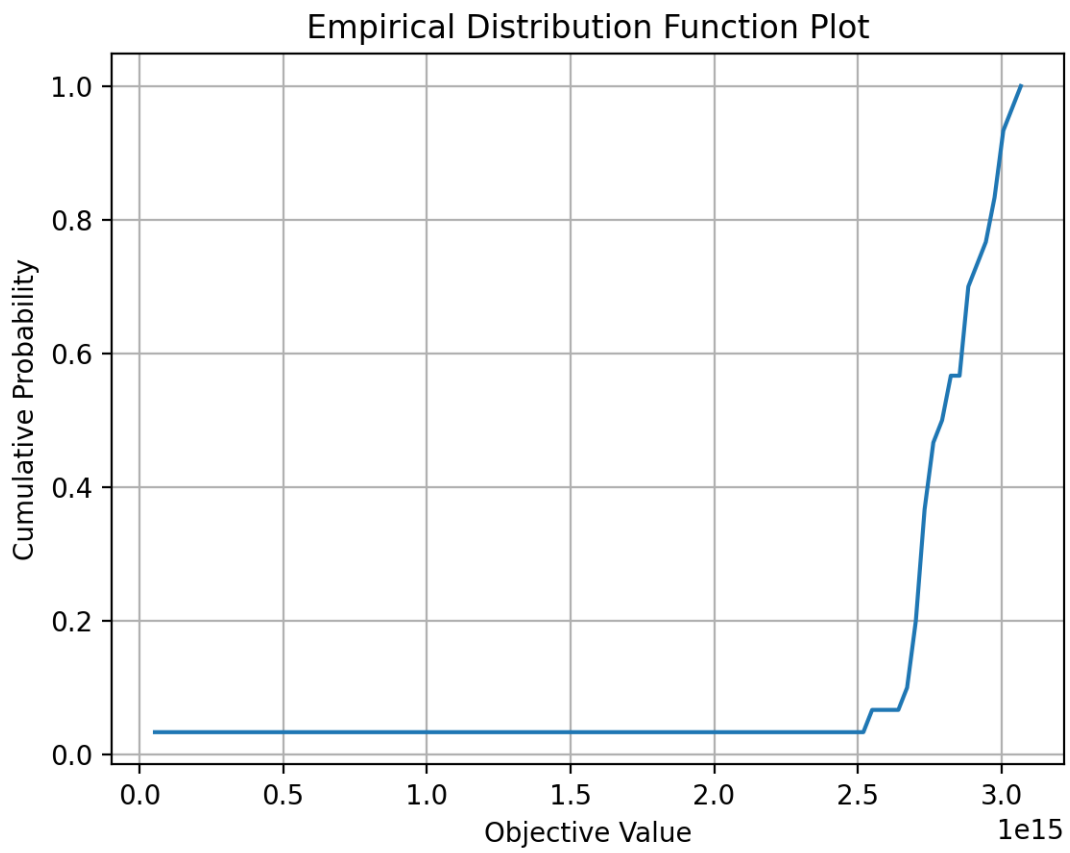
```
_ = plot_objective(results)
```



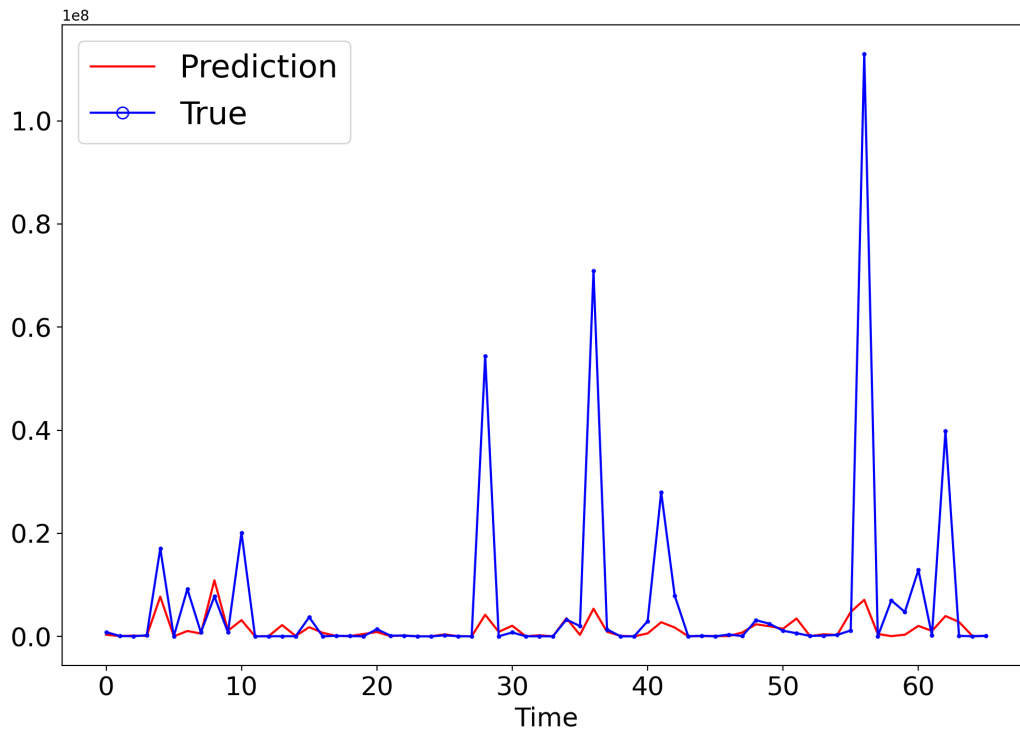
```
pl.optimizer._plot_evaluations(save=False)
```



```
pl.optimizer_._plot_edf(save=False)
```



```
pl.bfe_all_best_models(data=data)
```

Out:

```
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
packages/numpy/core/_methods.py:233: RuntimeWarning: overflow encountered in multiply
  x = um.multiply(x, x, out=x)
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
packages/sklearn/linear_model/_base.py:138: FutureWarning: The default of 'normalize'
will be set to False in version 1.2 and deprecated in version 1.4.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing
stage. To reproduce the previous behavior:
```

```
from sklearn.pipeline import make_pipeline
```

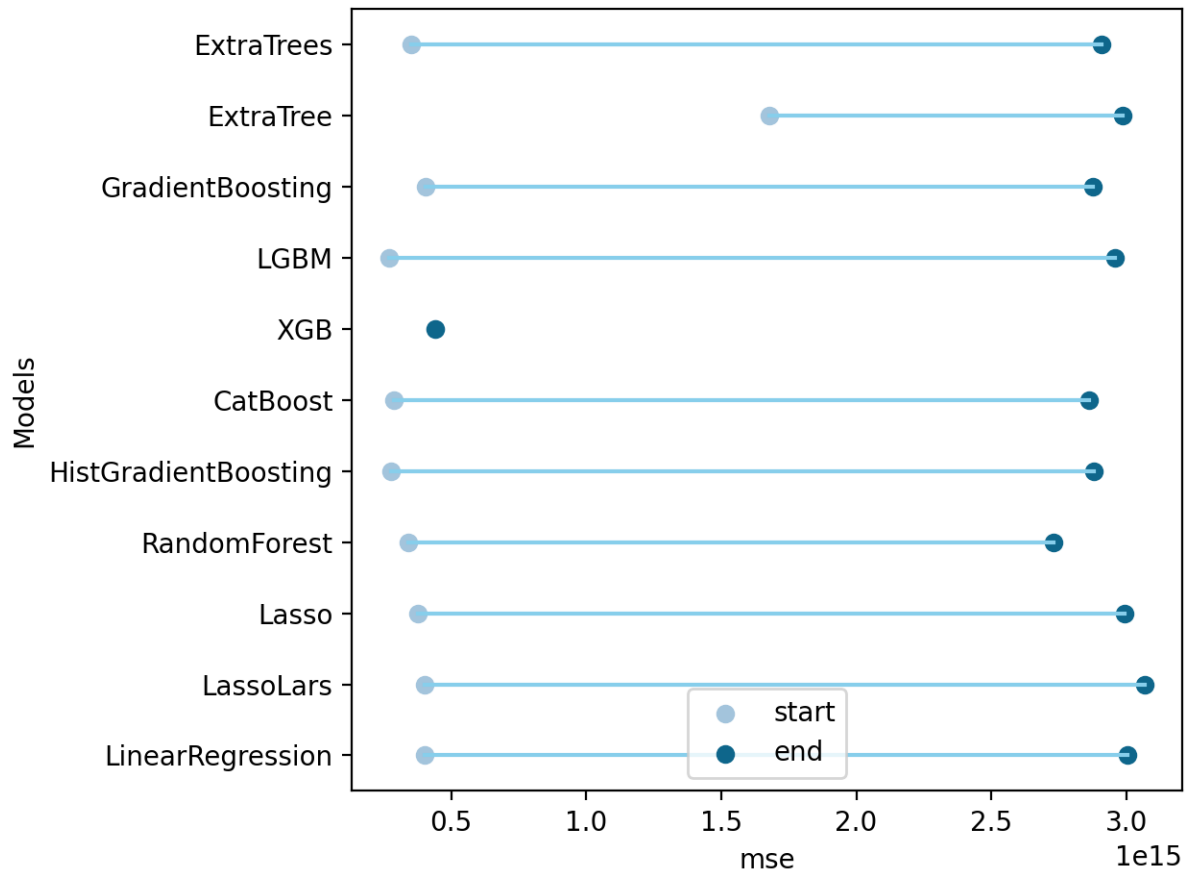
```
model = make_pipeline(StandardScaler(with_mean=False), LassoLars())
```

```
If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to
each step of the pipeline as follows:
```

```
kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)
```

```
Set parameter alpha to: original_alpha * np.sqrt(n_samples).
FutureWarning,
```

```
pl.dumbbell_plot(data=data, save=False)
```



Out:

```

/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
packages/sklearn/linear_model/_base.py:138: FutureWarning: The default of 'normalize'
will be set to False in version 1.2 and deprecated in version 1.4.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing
stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), LassoLars())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to
each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
FutureWarning,
/home/docs/checkouts/readthedocs.org/user_builds/autotab/envs/latest/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:648: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations, check
the scale of the features or consider increasing regularisation. Duality gap: 4.
075e+16, tolerance: 9.458e+12

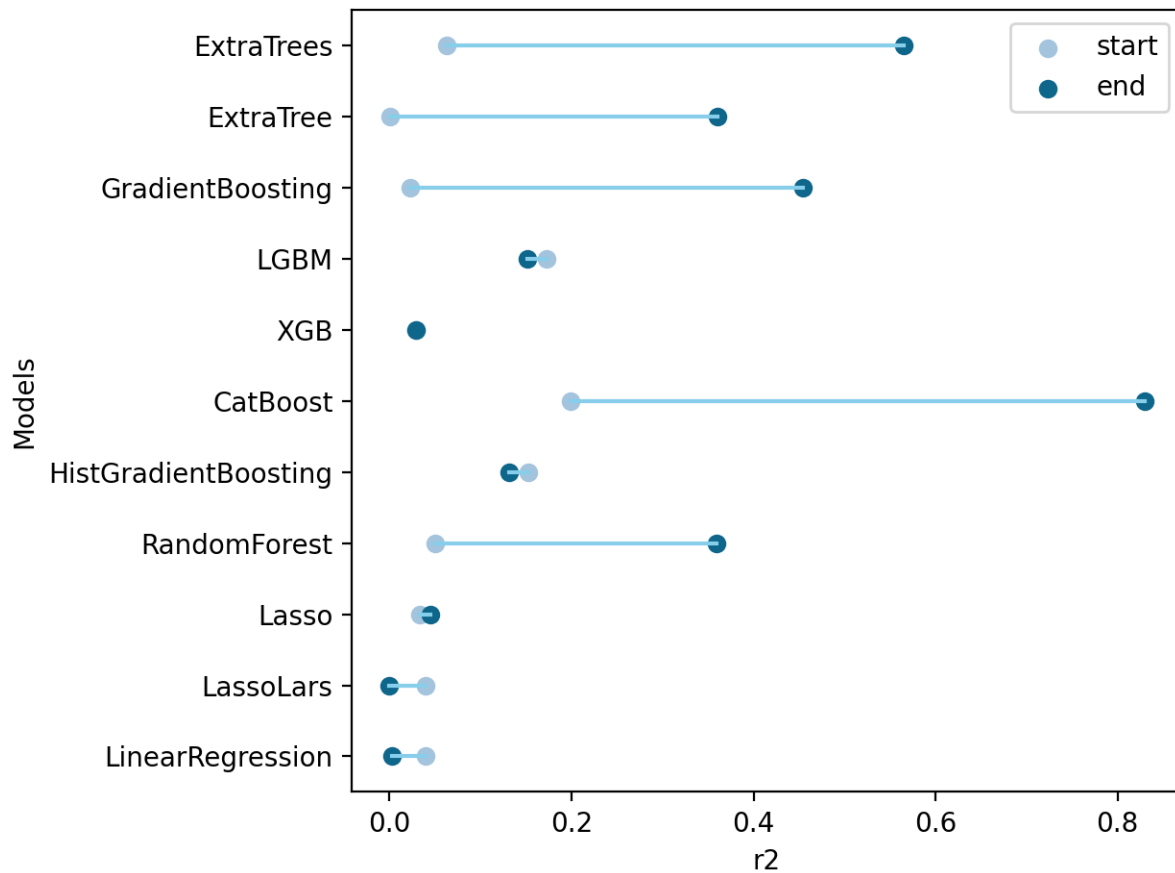
```

(continues on next page)

(continued from previous page)

```
coef_, l1_reg, l2_reg, X, y, max_iter, tol, rng, random, positive
<AxesSubplot:xlabel='mse', ylabel='Models'>
```

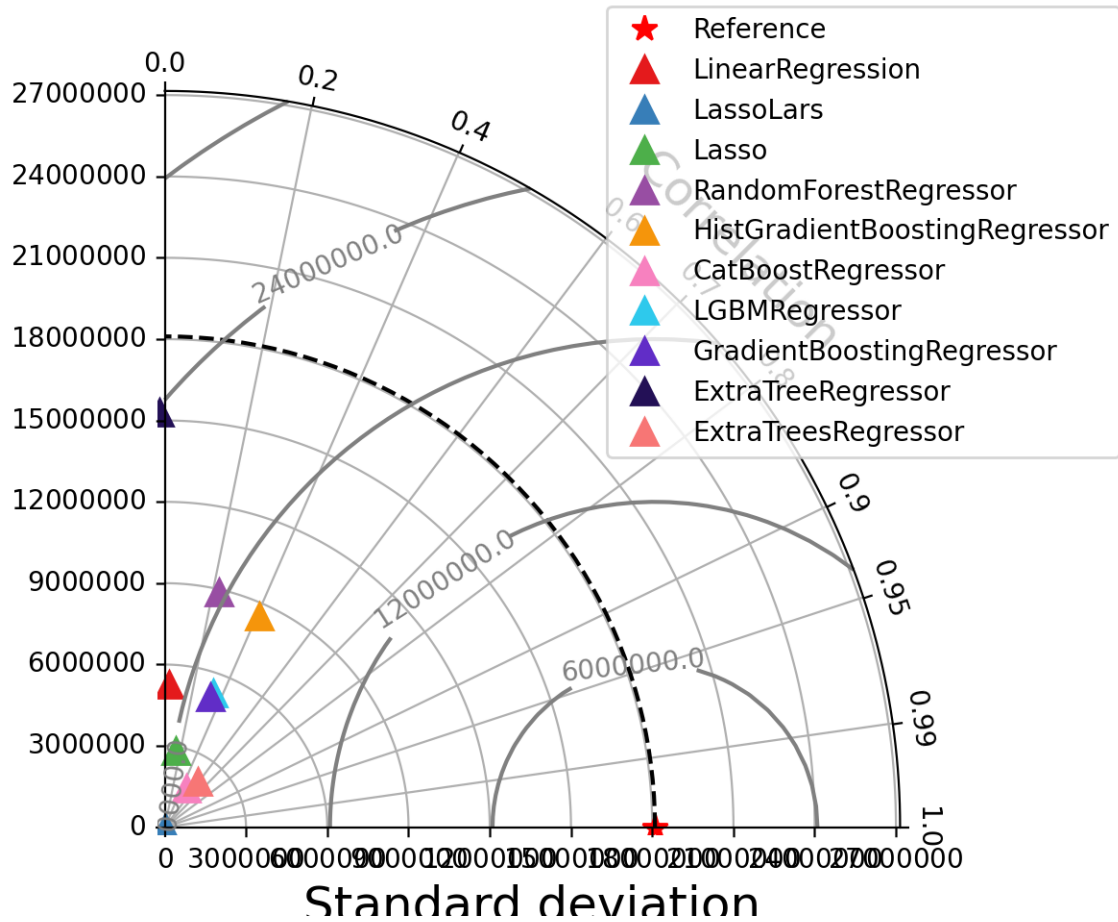
```
pl.dumbbell_plot(data=data, metric_name='r2', save=False)
```



Out:

```
<AxesSubplot:xlabel='r2', ylabel='Models'>
```

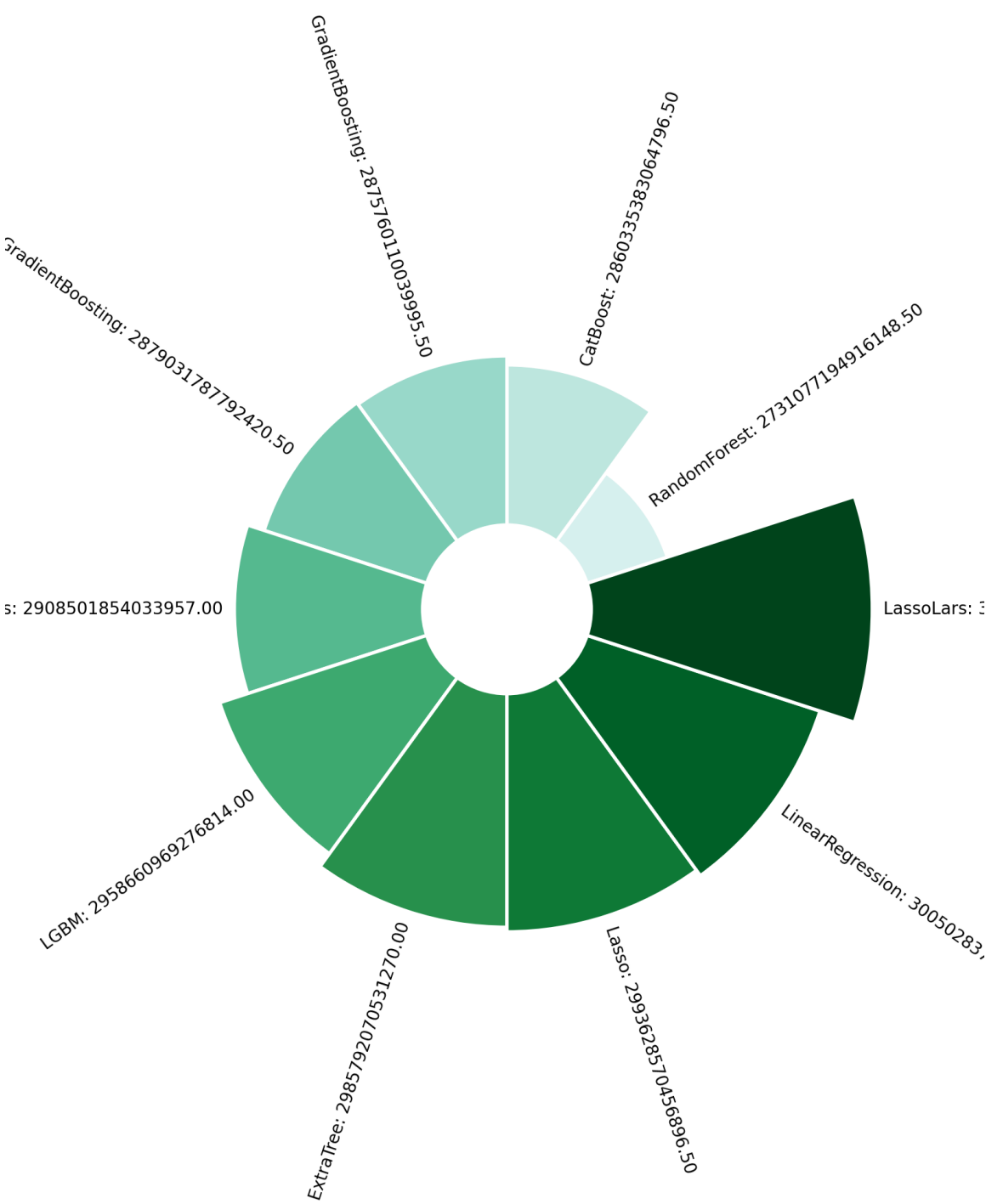
```
pl.taylor_plot(data=data, save=False)
```



Out:

```
<Figure size 640x480 with 1 Axes>
```

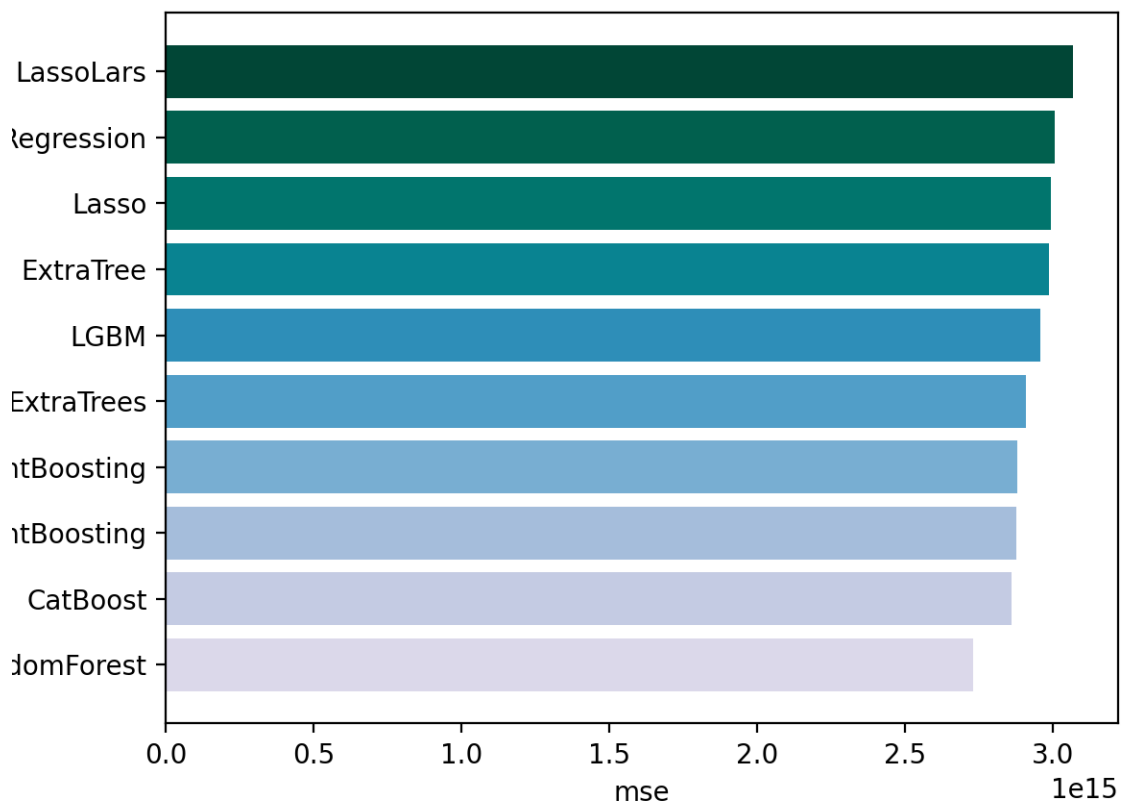
```
pl.compare_models()
```



Out:

```
<PolarAxesSubplot:>
```

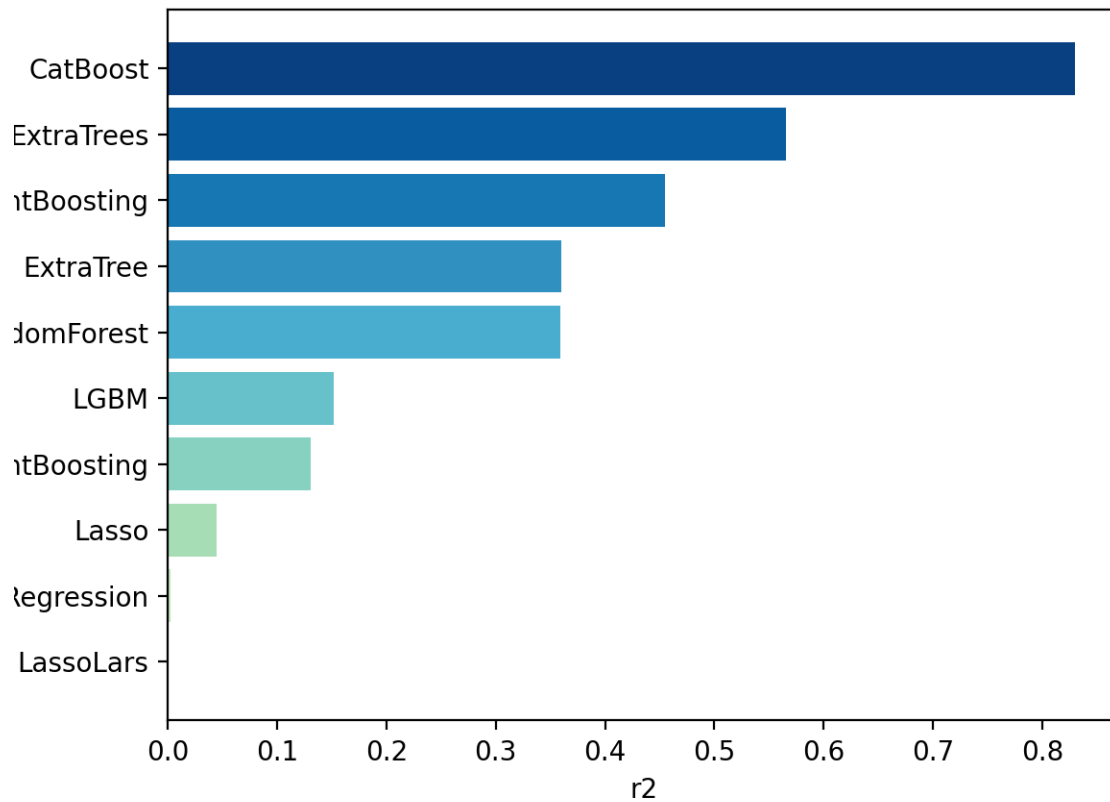
```
pl.compare_models(plot_type="bar_chart")
```



Out:

```
<AxesSubplot:xlabel='mse'>
```

```
pl.compare_models("r2", plot_type="bar_chart")
```



Out:

```
<AxesSubplot:xlabel='r2'>
```

```
print(f"all results are save in {pl.path} folder")
```

Out:

```
all results are save in /home/docs/checkouts/readthedocs.org/user_builds/autotab/
↳checkouts/latest/examples/results/pipeline_opt_20220504_162449 folder
```

```
pl.cleanup()
```

Total running time of the script: (3 minutes 7.167 seconds)

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

Symbols

`__init__()` (*autotab.OptimizePipeline method*), 22

`_build_model()` (*autotab.OptimizePipeline method*), 24

A

`add_dl_model()` (*autotab.OptimizePipeline method*), 24

`add_model()` (*autotab.OptimizePipeline method*), 24

B

`baseline_results()` (*autotab.OptimizePipeline method*), 24

`be_best_model_from_config()` (*autotab.OptimizePipeline method*), 25

`bfe_all_best_models()` (*autotab.OptimizePipeline method*), 25

`bfe_best_model_from_scratch()` (*autotab.OptimizePipeline method*), 26

`bfe_model_from_scratch()` (*autotab.OptimizePipeline method*), 26

C

`change_child_iteration()` (*autotab.OptimizePipeline method*), 27

`compare_models()` (*autotab.OptimizePipeline method*), 27

`config()` (*autotab.OptimizePipeline method*), 27

D

`dumbbell_plot()` (*autotab.OptimizePipeline method*), 28

F

`fit()` (*autotab.OptimizePipeline method*), 28

`from_config()` (*autotab.OptimizePipeline class method*), 29

`from_config_file()` (*autotab.OptimizePipeline class method*), 29

G

`get_best_metric()` (*autotab.OptimizePipeline method*), 29

`get_best_metric_iteration()` (*autotab.OptimizePipeline method*), 29

`get_best_pipeline_by_metric()` (*autotab.OptimizePipeline method*), 29

`get_best_pipeline_by_model()` (*autotab.OptimizePipeline method*), 29

O

`OptimizePipeline` (*class in autotab*), 21

P

`post_fit()` (*autotab.OptimizePipeline method*), 30

R

`remove_model()` (*autotab.OptimizePipeline method*), 30

`report()` (*autotab.OptimizePipeline method*), 31

S

`save_results()` (*autotab.OptimizePipeline method*), 31

T

`taylor_plot()` (*autotab.OptimizePipeline method*), 31

U

`update_model_space()` (*autotab.OptimizePipeline method*), 31