ExplainAI

by Feini Huang, Wei Shuangguan, Yongkun Zhang

Contact: huangfn3@mail2.sysu.edu.cn

Content

Overview

Installation

ExplainAI works in

Currently it requires

You can install ExplainAI using pip:

```
pip install ExplainAI
```

The latest version (ExplainAI 0.1.22)available for

```
pip install ExplainAI==0.1.22
```

Or clone codes from github:

https://github.com/HuangFeini/ExplainAI.git

git@github.com:HuangFeini/ExplainAI.git

gh repo clone HuangFeini/ExplainAI

In order to use the ExplainAI successfully, the following site-packages are required:

- pandas
- eli5
- packaging
- psutil
- numpy
- lime
- sklearn
- scipy
- seaborn
- shap
- matplotlib

The latest ExplainAI 0.1.22 can work in

linux-Ubuntu 20.04+

Window 7+

Features

ExplainAI is a Python package which helps to visualize black-box machine learning and explain their predictions. It provides support for the following machine learning frameworks and functions:

- scikit-learn. Currently ExplainAI allows to explain predictions of scikit-learn regressors including DecisionTreeRegressor, LinearRegression, svm.SVR, KNeighborsRegressor, RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor, BaggingRegressor, ExtraTreeRegressor， in order to show feature importances and feature effects.
- Post-hoc interpretation. Currently, ExplainAI integrated the following post-hoc methods: partial dependence plot (PDP), mean squared error (MSE)-based feature importance (MFI), permutation importance (PI), accumulated local effect (ALE), individual conditional expectation (ICE), local

interpretable model-agnostic explanations(LIME) and Shapley values. Details about each methods are given in the Feature effects section of the tutorial.
- Data preview. You can visualize observation and prediction distribution of feature or feature interaction, which are displayed in a figure of console.
- Two formats of explanation. You can upload your raw data (better in a csv) and after interpretation, you can get plot-based and text-based explanation in the console.
- Feature selection. The sequence backward selection (SBS) is provided. And some feature selection procedures specific for FLUXNET data also are available.

Basic usage

The basic usage involves following procedures:

1. upload your raw data and conduct data cleaning.
2. choose whether feature selection by sequential backward selection, if yes, a new input data is obtained, if no, you can use contrived work to select the features.
3. prediction, using sklearn model to train model and get prediction.
4. check the prediction and observation distribution.
5. interpretation. The trained model, input data matrix as input, the interpretation methods can be objectified.
6. display the results of interpretation (plot or text).

We recommend the users to accomplish step 1 to 3 due to their own requirements, and use the functions of step 4,5 provided in the ExpalinAI toolbox.

There are two main ways to interpret a black-box model:

1. inspect all the model predctions together and try to figure out how the model works globally;
2. inspect an individual prediction of a model, try to figure out why the model makes the decision it makes.

   For (1), ALE, PDP, MFI and PI, are all the avaliable "global" tools.

   For (2), ICE, Shapley values and LIME are all the avaliable "local" tools.

The interpretation are formatting in several ways, including figures, text, and a pandas Dataframe object. For example, a global interpretation are given as follows. You can also see these codes in the ExplainAI package/test.py

```
#1.upload your raw data and conduct data cleaning.
#here, we use the default dataset which has conducted data cleaning as an example.
from ExplainAI.flx_data.input import input_dataset
d=input_dataset(flag=0)
# d: the entire dataset (both input and output)




#2. choose whether feature selection by sequential backward selection, if yes, a new input
data is obtained, if no, you can use contrived work to select the features.
pass

#3.prediction, using sklearn model to train model and get prediction.
```

```python
#first, splite the data into training set and testing set
from ExplainAI.data_processing.split_data import split_data
xtr,ytr,xte,yte=split_data(d,target="SWC").split_xy()
#second, sklearn modeling
#here, you can use original sklearn function instead.
from ExplainAI.model.make_model import make_model
m,res,y_predict=make_model(modeltype='RandomForest',
                             x_train=xtr,
                             y_train=ytr,
                             x_test=xte,
                             y_test=yte)
print(res)
#m: sklearn model object
#res: sklearn metrics
#y_predict: prediction values of testing set



#4.check the prediction and observation distribution.
from ExplainAI.preview import info_plots
import matplotlib.pyplot as plt
# show distribution with feature of interest ("TS")
fig1, axes, summary_df = info_plots.actual_plot(model=m, X=xte, feature="TS",
feature_name="TS")
fig2, axes, summary_df = info_plots.target_plot(df=d, target="SWC", feature="TS",
feature_name="TS")
# # show distribution under two features' interaction
fig3, axes, summary_df = info_plots.actual_plot_interact(model=m, X=xte, features=["DOY",
"TS"], feature_names=["DOY", "TS"])
fig4, axes, summary_df = info_plots.target_plot_interact(df=d, target="SWC", features=
["DOY", "TS"], feature_names=["DOY", "TS"])

# plot-based results
plt.show()
fig4.savefig('fig4.jpg')
#text-based results
print(summary_df)


#5.interpretation. The trained model, input data matrix as input, the interpretation
methods can be objectified.

#first, get input dataset and features
from ExplainAI.utils import get_x,get_features
x=get_x(d,target="SWC")
f=get_features(x)
#second, interpretation (PI as an example)
from ExplainAI.explainers.pi.pi import permutation_importance
p=permutation_importance(model=m, features=f, save=True,plot=True,save_path='pi.jpg')

#6.display the results of interpretation (plot or text).
# plot-based results
#'pi.jpg' seemed in your save_path
```

```
#text-based results
print(p)
```

Why use ExplainAI?

At present, the post-hoc tools are widely used in many fields. However, it is not convenient to use different methods from different packages. Particularly, it leads to compatibility issues. To address this, ALE, PDP, ICE, Shapley, LIME, PI, MFI are integrated to one practical tool for ML developers and the decision-makers. Using ExplainAI, you can have a better experiences:

- you can call a ready-made function from ExplainAI and get a nicely formatted result immediately;
- formatting code can be reused between machine learning frameworks;
- algorithms like LIME try to explain a black-box model through a locally-fit simple, interpretable model. It means that with additional "simple" model supported algorithms like LIME will get more options automatically.

Tutorials

In this turoial, we will show how to use the ExplainAI using an example data set from a

site or other two fixed format csv files. Users who want to build their own machine learning model can just jump to the feature effects section for the functions available to interpret and visualize the model.

Task

With the increasing demand for machine learning application in hydrometeorological forecast, we face the urge to demystify the black-box of machine learning as the lack of interpretability hampers adaptation of machine learning.

Here, taking soil moisture (SM) prediction of one FLUXNET site (Haibei,China, named as CH-Ha2) as an example, we used air forcing variables, timekeeping, energy processing, net ecosystem exchange and partitioning, and sundown as input data. We aimed to predict the daily SM via historial dataset. We aimed to interpret the model via ExplainAI toolbox.

Dataset

All dataset used in this tutorial is in the flx_data" dictionary of ExplainAI toolbox. The meta data of FLUXNET site data (Haibei,China, named as CH-Ha2) is available at https://ftp.fluxdata.org/.fluxnet_downloads_86523/FLUXNET2015/FLX_CN-Ha2_FLUXNET2015_FULLSET_2003-2005_1-4.zip

If users want to change the dataset, please modify the flag value of input_dataset(flag).

Filename	Content	Function
FLX_CN-Ha2_ FLUXNET2015_ FULLSET_DD_ 2003-2005_1-4.csv	Raw site data downloaded from FLUXNET	data=input_dataset(flag=2)
dataset_process.csv	Data after entire data processing	data=input_dataset(flag=1)
dataset.csv	Data after data processing and contrived work	data=input_dataset(flag=0)

```
from ExplainAI.flx_data.input import input_dataset
d=input_dataset(flag=0)
d=input_dataset(flag=1)
d=input_dataset(flag=2)
```

`:param` flag: index of which data set, see Tutorials section Dataset `:return` data: read data input, pandas.Dataframe

Data processing

Since the FLUXNET raw data can not be used directly used in modeling, the data processing offers a feasible way to process the raw data.

Certainly, if the users have other time-relating and lagged-relating variables, the functions can be modified. And the dataset can be replaced.

Read data and add time-relating variables

The original FLUXNET data with time series only has time-relating variable "TIMESTAMP", whose formation is year%month%day%. It can not be used as a time-series variable.

Via the time_add function, the DAY (day sequence of whole time) and DOY (day of year) are added.

```
from ExplainAI.data_processing.add_variables import time_add
file='.\\flx_data\\FLX_CN-Ha2_FLUXNET2015_FULLSET_DD_2003-2005_1-4.csv' #change your path
data=pd.read_csv(file,header=0)
new_data=time_add(data)
```

`:parameter` file: data , from csv `:returns` data:pd.Dataframe

Add lagged-relating variables

In this example, the soil moisture has time "memory" and the lagged precipitation also has impact on soil moisture prediction, the 1 to 7 days lagged values of these two variables are added in the dataset.

```
from ExplainAI.data_processing.add_variables import lag_add
new_data=lag_add(data,sm_lag=7,p_lag=7)
#sm_lag and p_lag are the days of lagged soil moisture and precipitation. Defaults are 7.
```

`:param data:` pd.Dataframe, input data `:param sm_lag:` int, lagged days of soil moisture `:param p_lag:` int, lagged days of precipitation `:return data:` pd.Dataframe, input data with lagged variables

Data cleaning and feature selection

data_cleaning() offers several data cleaning functions:

- Eliminate the observation without target values
- Eliminate irrelevant records in FLUXNET, like percentiles, quality index, RANDUNC, se, sd...
- Eliminate the features with too many (30%) Nan.

```
from ExplainAI.data_processing.data_cleaning import data_cleaning
c1=data_cleaning(data)
d1=c1.elim_SM_nan()
c2=data_cleaning(d1)
d2=c2.drop_ir()
c3=data_cleaning(d2)
d3=c3.drop_nan_feature()
```

```
#Or only use one function
Newdata=data_cleaning(data).elim_SM_nan()
Newdata=data_cleaning(data).drop_ir()
Newdata=data_cleaning(data).drop_nan_feature()
```

`:parameter data:` pd.Dataframe,input data `:returns Newdata:` pd.Dataframe, output data

feature_selection() provides sequential backward selection (SBS) which based on random forest.

```
from ExplainAI.data_processing.feature_selection import feature_selection
fs=feature_selection(data=data,target="SWC_F_MDS_1")
sbs_result=fs.sbs_rf(n_estimators=100)
print(sbs_result)
#Be aware that, sbs_result is the SBS scores.
#If you want to select features, you can download the data and select the features
according to the SBS results.
```

`:parameter` `data:` pd.Dataframe,input data `target:` string, name of target `:returns NFeaFrame:` dataframe, including features and their importance calculated by MSE.

data_processing_main() integrates data cleaning and feature selection.

```
from ExplainAI.data_processing.data_processing_main import data_processing_main
d=data_processing_main(data=data,
                       target='SWC_F_MDS_1',
                       time_add=True,
                       lag_add=True,
                       elim_SM_nan=True,
                       drop_ir=True,
```

```
                              drop_nan_feature=True,
                              part=0.7,
                              n_estimator=200,
                              sbs=True,
                              split=2)
dd,ss=d.total()
dd.to_csv("dd.csv")
#dd is new_dataset after data processing
ss.to_csv('ss.csv')
#ss is sbs result
```

`:parameter`

`data:` pd.Dataframe,input data

`target:` string, predicted target column name

`time_add:` bool, whether time_add

`lag_add:` bool, whether lag_add

`elim_SM_nan:` bool, whether elim_SM_nan

`drop_ir:` eliminate data of irrelevant records in FLUXNET,like percentiles, quality index, RANDUNC, se, sd... `drop_nan_feature:` Eliminate the features with too many(30%) Nan. `part:` part of split_data `n_estimator:` sequential backward selection using random forest, n_estimator of random forest `sbs:` whether use sbs

`split:` int 2 or int 3, if 2, splite data to training set and testing set; if 3, training set, validating set and testing set

Split dataset

split_data offers a way to split dataset into training set and testing set, according to the time-sequence (using data of time-ahead to predict feature data).

```
from ExplainAI.data_processing.split_data import split_data
xtr,ytr,xte,yte=split_data(data,target="SWC",part=0.7).split_xy()

train,test=split_data(data,target="SWC",part=0.7).split()

#Or validating set is required.
train, valid, test=split_data(data,target="SWC",part3=[0.7,0.2,0.1]).split3()
```

`:parameter` `data[dataframe]:` A data set

`target[str]:` target feature name

`part[float]:` division proportion for two default as 0.7 `part3[list]:` A list division proportion for three, default as [0.7,0.2,0.1]

Black-box machine learning

For this version, sklean models are available.

```
from ExplainAI.model.make_model import make_model
model_list = ['DecisionTree', 'Linear', 'KNeighbors',
                  'RandomForest', 'AdaBoost',
                  'GradientBoosting', 'Bagging',
                  'BayesianRidge', 'SVR']
m,res,y_predict=make_model(modeltype='GradientBoosting',
                             x_train=xtr,
                             y_train=ytr,
                             x_test=xte,
                             y_test=yte)
print(res)
#res:R2,MSE,MAE,RMSE
#m:trained model object
#y_predction:series, precdiction of testing set
```

If USER wants to modify the parameters of sklear models, please use the original sklearn model instead, for example:

```
from sklearn.metrics import
r2_score,mean_absolute_error,mean_squared_error,mean_squared_log_error
from sklearn import ensemble
m=ensemble.RandomForestRegressor(n_estimators=500)
m.fit(xtr,ytr)
y_predict = m.predict(xte)
res={"r2":r2_score(y_predict,yte),
             "MSE":mean_squared_error(y_predict,yte),
             "MAE":mean_absolute_error(y_predict,yte),
             "RMSE":mean_squared_log_error(y_predict,yte)}
print(res)
```

Preview

Data preview. You can visualize observation and prediction distribution of feature or feature interaction, which are displayed in a figure of console.

```
info_plots.actual_plot()
info_plots.target_plot()
actual_plot_interact()
target_plot_interact()
```

`Parameters`

`df: pandas DataFrame` data set to investigate on, should contain at least the feature to investigate as well as the target `feature: string or list` feature or feature list to investigate, for one-hot encoding features, feature list is required `feature_name: string` name of the feature, not necessary a column name `target: string or list` column name or column name list for target value for multi-class problem, a list of one-hot encoding target column `num_grid_points:` integer, optional, default=10 number of grid points for numeric feature `grid_type: string, optional`, default='percentile' 'percentile' or 'equal' type of grid points for numeric feature `percentile_range: tuple or None,` optional, default=None percentile range to investigate for numeric feature when grid_type='percentile' `grid_range: tuple or None,` optional, default=None value range to investigate for numeric feature when grid_type='equal' `cust_grid_points: Series`, 1d-array, list or None, optional, default=None, customized list of grid points, for numeric feature `show_percentile:` bool, optional, default=False whether to display the percentile buckets for numeric feature when grid_type='percentile' `show_outliers:` bool, optional, default=False whether to display the out of range buckets for numeric feature when percentile_range or grid_range is not None `endpoint:` bool, optional, default=True If True, stop is the last grid point Otherwise, it is not included `figsize:` tuple or None, optional, default=None size of the figure, (width, height) `ncols:` integer, optional, default=2 number subplot columns, used when it is multi-class problem `plot_params:` dict or None, optional, default=None parameters for the plot

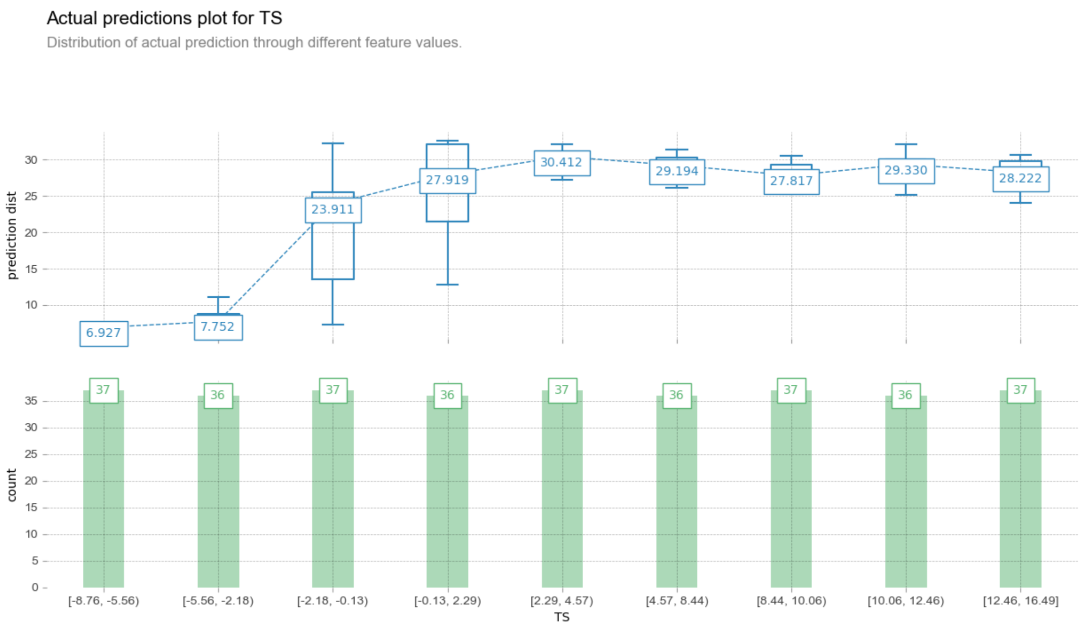
`Returns`

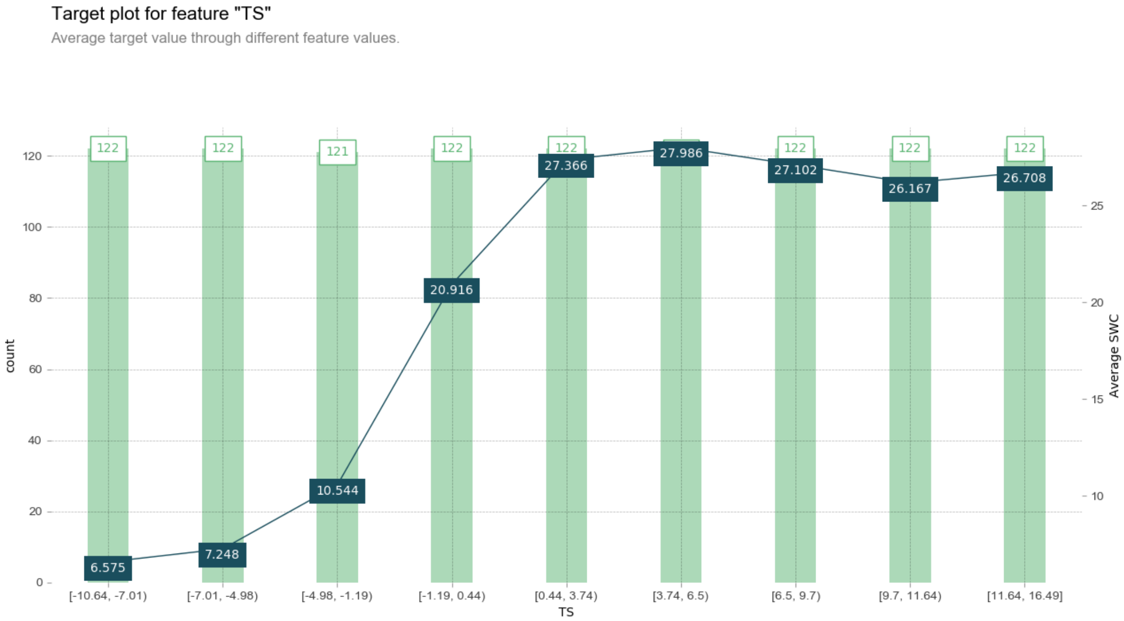
`fig:` matplotlib Figure `axes:` a dictionary of matplotlib Axes Returns the Axes objects for further tweaking `summary_df:` pandas DataFrame Graph data in data frame format

For example,

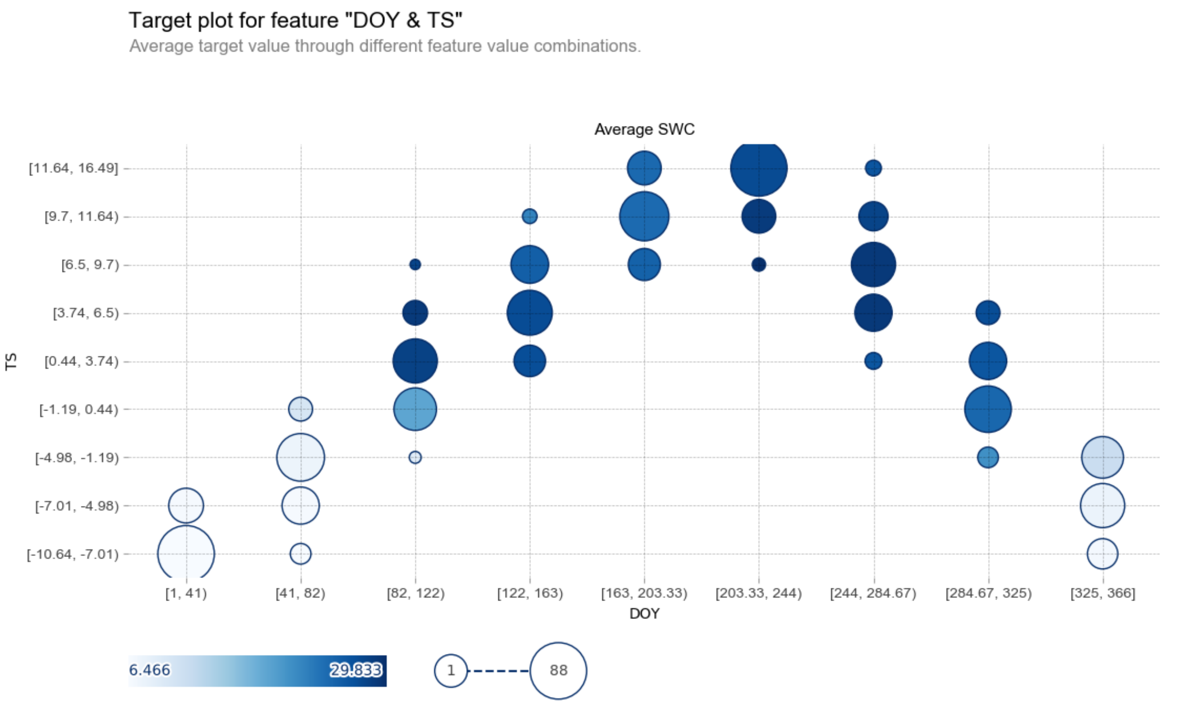
```
# from preview import info_plots
# import matplotlib.pyplot as plt
from ExplainAI.preview import info_plots
import matplotlib.pyplot as plt
# show distribution with feature of interest ("TS")
fig1, axes, summary_df = info_plots.actual_plot(model=m, X=xte, feature="TS",
feature_name="TS")

fig2, axes, summary_df = info_plots.target_plot(df=d, target="SWC", feature="TS",
feature_name="TS")
# show distribution under two features' interaction
fig3, axes, summary_df = info_plots.actual_plot_interact(model=m, X=xte, features=["DOY",
"TS"], feature_names=["DOY", "TS"])

fig4, axes, summary_df = info_plots.target_plot_interact(df=d, target="SWC", features=
["DOY", "TS"], feature_names=["DOY", "TS"])


# plot-based results
plt.show()  # for windows
fig4.savefig('fig4.jpg') #for windows and linux
#text-based results
print(summary_df)
```

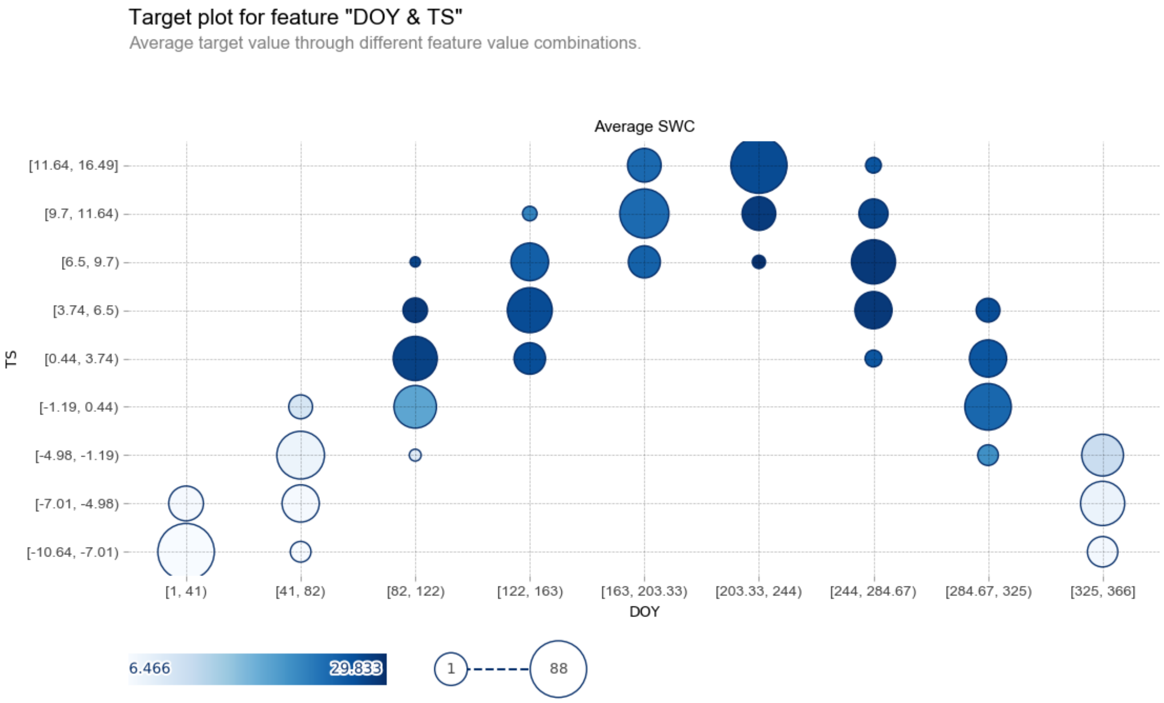
```
#print(summary_df)
x1  x2 display_column_1  ... value_upper_2  count       SWC
0     0   0          [1, 41)  ...      -7.009000   88.0  6.465909
1     0   1          [1, 41)  ...      -4.983333   32.0  6.775000
2     0   2          [1, 41)  ...      -1.191000    0.0  0.000000
3     0   3          [1, 41)  ...       0.436333    0.0  0.000000
4     0   4          [1, 41)  ...       3.735667    0.0  0.000000
..   ..  ..              ...  ...            ...    ...       ...
76    8   4       [325, 366]  ...       3.735667    0.0  0.000000
77    8   5       [325, 366]  ...       6.505000    0.0  0.000000
78    8   6       [325, 366]  ...       9.701000    0.0  0.000000
79    8   7       [325, 366]  ...      11.642333    0.0  0.000000
80    8   8       [325, 366]  ...      16.486000    0.0  0.000000
```





![Figure_3](/Figure_3.png





Feature effects

MSE-based Feature importance

Being one of the most pragmatic methods to quantify the feature importance, the Python package named as sklearn provides a specified importance evaluation for RF model. Note that R package named as randomForest also provides similar functions (Breiman, 2001). This method computes the importance from permuting out-of-bag data. First, for each tree, the MSE from prediction model on the out-of-bag portion of the training data is recorded. Next, this procedure is repeated for each feature.

Noted that, this method is specific-based, only for random forest.

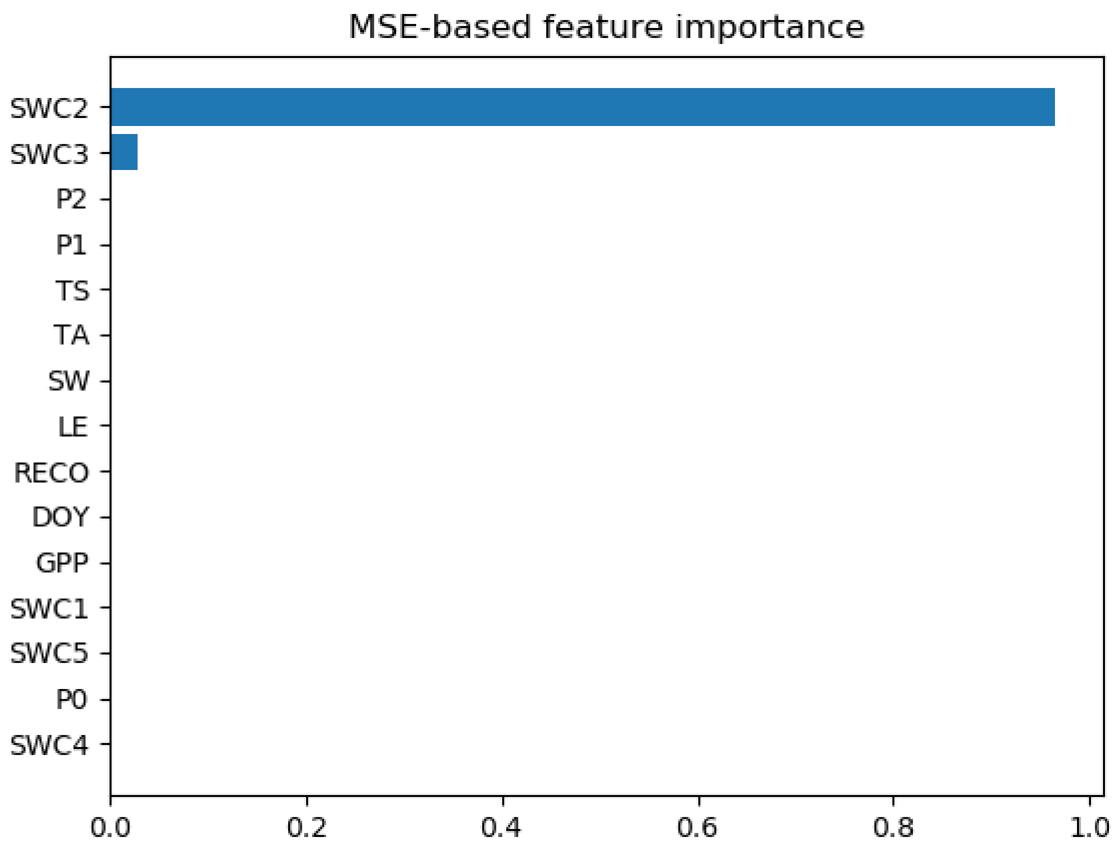
```
mse_feature_importance()
```

`:param model:` sklearn model object, trained model `:param data:` pd.Dataframe, input data `:param target:` string, predicted target column name `:param plot:` bool, if plt.show() `:param top:` int, number of top feature at list `:param save:` bool, if save the picture `:param save_path:` string, path of picture saved `:return:` `df:` pd.Dataframe, MFI of features

For example,

```python
from ExplainAI.explainers.mfi.mfi import mse_feature_importance
mfi=mse_feature_importance(model=m, data=d,
target="SWC",top=15,save=True,plot=True,save_path='mfi.jpg')
print(mfi)

# plot-based results
#plot=True # for windows
#save_path='mfi.jpg' #for windows and linux
#text-based results
print(mfi)
```

```
#print(mfi)
   feature       MFI
0      DOY  0.000193
1       TA  0.000397
2       SW  0.000378
3       TS  0.000657
4       LE  0.000357
5      GPP  0.000198
6     RECO  0.000237
7     SWC1  0.000219
8     SWC2  0.992778
9     SWC3  0.000774
10    SWC4  0.000148
11    SWC5  0.000108
12      P0  0.000129
13      P1  0.001396
14      P2  0.001866
```



Permutation importance

The PI of the observed importance provides a corrected measure of feature importance. PI computed with permutation importance are very helpful for deciding the significance of variables, and therefore improve model interpretability.

permutation_importance() offers an approach of PI storage in a dataframe and plot of ranking features.

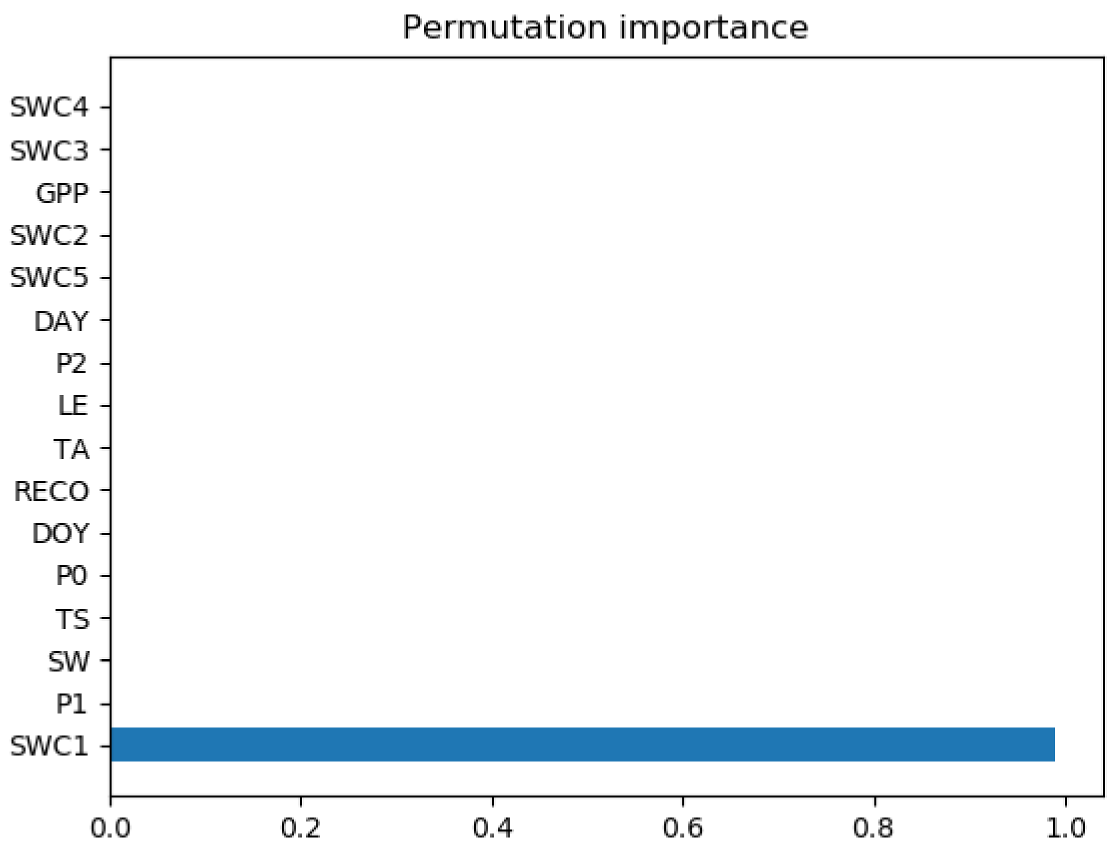
```
permutation_importance()
```

`:param model:` sklearn model object, trained model `:param features:` list or turple, fearture names storaged in a list or a turple `:param plot:` bool, if plt.show() `:param save:` bool, if save the picture `:param save_path:` string, path of picture saved `:return:` pd.Dataframe, PI of features

For example,

```
from ExplainAI.explainers.pi.pi import permutation_importance
p=permutation_importance(model=m, features=f, save=True,plot=True,save_path='pi.jpg')
# plot-based results
#plot=True # for windows
#save_path='pi.jpg' #for windows and linux
#text-based results
print(p)
```

```
#print(p)
   feature    weight       std
0     SWC1  0.983773  0.092478
1     SWC2  0.009759  0.092371
2       P1  0.001884  0.000815
3       P0  0.001252  0.000717
4       SW  0.000654  0.000705
5      DOY  0.000460  0.000392
6       TS  0.000404  0.000433
7       TA  0.000333  0.000310
8      GPP  0.000244  0.000244
9       LE  0.000213  0.000225
10     DAY  0.000208  0.000217
11    RECO  0.000200  0.000232
12    SWC5  0.000183  0.000215
13      P2  0.000162  0.000140
14    SWC3  0.000154  0.000172
15    SWC4  0.000115  0.000103
```



Partial dependence plot

The PDP demonstrates the relationships between the features and predicted variable (Friedman, 2001). The PDP for regression is defined as:

$$p(x(s,j))(x(s,j)) = 1/n \sum^{n} f[x(s,j), x_c(i)]$$

where x(s,j) is the set of the feature of interest (as j-th feature) for which the partial dependence function should be plotted, p(x(s,j)) is the partial dependence value of j-th feature, n is the number of elements in x_s, and x_c is subset of other actual features values. PDP estimates the average marginal effect of predictors on the predicted SM, which can be a determined value in regression.

partial_dependence_plot_1d() provides one-dimentinal PDP.

```
explainers.partial_dependence_plot_1d()
```

`Parameters`

`model:` a fitted sklearn model `data:` pandas DataFrame, data set on which the model is trained `model_features:` list or 1-d array, list of model features `feature:` string or list,feature or feature list to investigate, `num_grid_points:` integer, optional, default=10, number of grid points for numeric feature `grid_type:` string, optional, default='percentile', 'percentile' or 'equal', type of grid points for numeric feature `percentile_range:` tuple or None, optional, default=None percentile range to investigate, for numeric feature when grid_type='percentile' `grid_range:` tuple or None, optional, default=None,value range to investigate, for numeric feature when grid_type='equal' `cust_grid_points:` Series, 1d-array, list or None, optional, default=None, customized list of grid points for numeric feature `memory_limit:` float, (0, 1), fraction of memory to use `n_jobs:` integer, default=1

`pdp_isolate_out:` (list of) instance of PDPIsolate, for multi-class, it is a list `center:` bool, default=True, whether to center the plot `plot_pts_dist:`bool, default=False whether to show data points distribution `plot_lines:` bool, default=False whether to plot out the individual lines `frac_to_plot:` float or integer, default=1 how many lines to plot, can be a integer or a float `cluster:` bool, default=False whether to cluster the individual lines and only plot out the cluster centers `n_cluster_centers:` integer, default=None number of cluster centers `cluster_method:`string, default='accurate' cluster method to use, default is KMeans, if 'approx' is passed, MiniBatchKMeans is used `x_quantile:` bool, default=False whether to construct x axis ticks using quantiles `show_percentile:` bool, optional, default=False whether to display the percentile buckets, for numeric feature when grid_type='percentile' `figsize:` tuple or None, optional, default=None size of the figure, (width, height) `ncols:` integer, optional, default=2 number subplot columns, used when it is multi-class problem `plot_params:` dict or None, optional, default=None

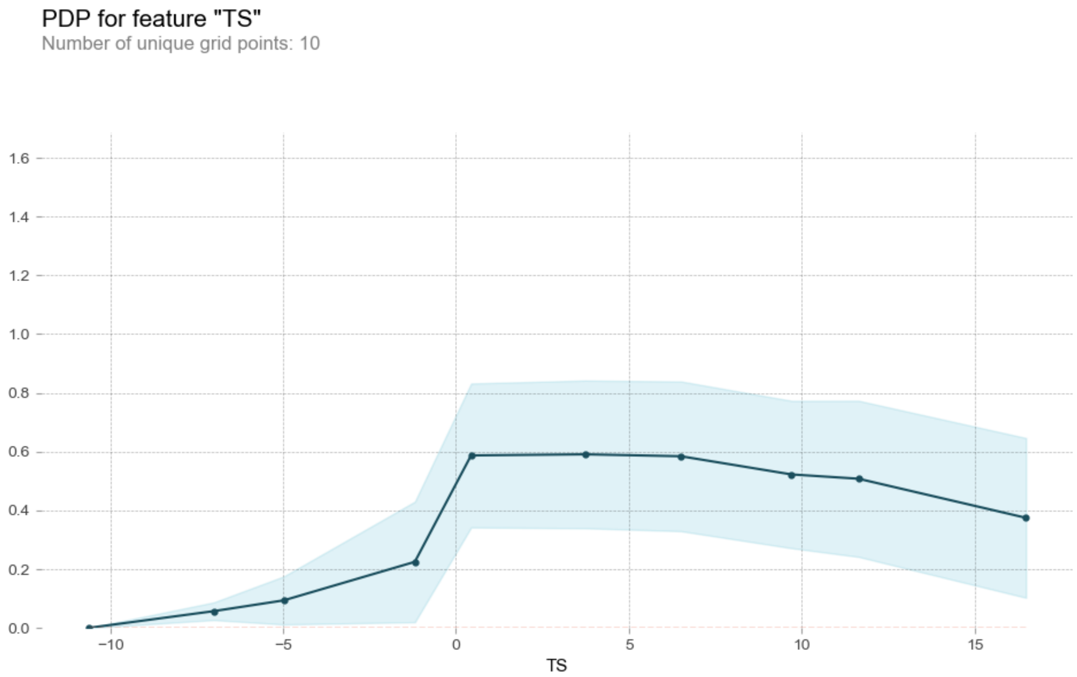
`plot:` bool, if plt.show() `save:` bool, if save the picture `save_path:` string, path of picture saved, default='pdp.jpg'

`return:` PDP dataframe

For example,

```
from ExplainAI.explainers.pdp.pdp import partial_dependence_plot_1d
pd1=partial_dependence_plot_1d(model=m,data=x,model_features=f,feature="TS",plot=True,save=
True)
print(pd1)
```

```
#print(pd1)
x      xticklabels  count  count_norm
0  0  [-10.64, -7.01)    122    0.111314
1  1   [-7.01, -4.98)    122    0.111314
2  2   [-4.98, -1.19)    121    0.110401
3  3    [-1.19, 0.44)    122    0.111314
4  4     [0.44, 3.74)    122    0.111314
5  5      [3.74, 6.5)    121    0.110401
6  6       [6.5, 9.7)    122    0.111314
7  7     [9.7, 11.64)    122    0.111314
8  8   [11.64, 16.49]    122    0.111314
```



partial_dependence_plot_2d() provides two-dimentinal PDP, of which the two features have no interactions.

```
explainers.partial_dependence_plot_2d()
```

`parameter:`

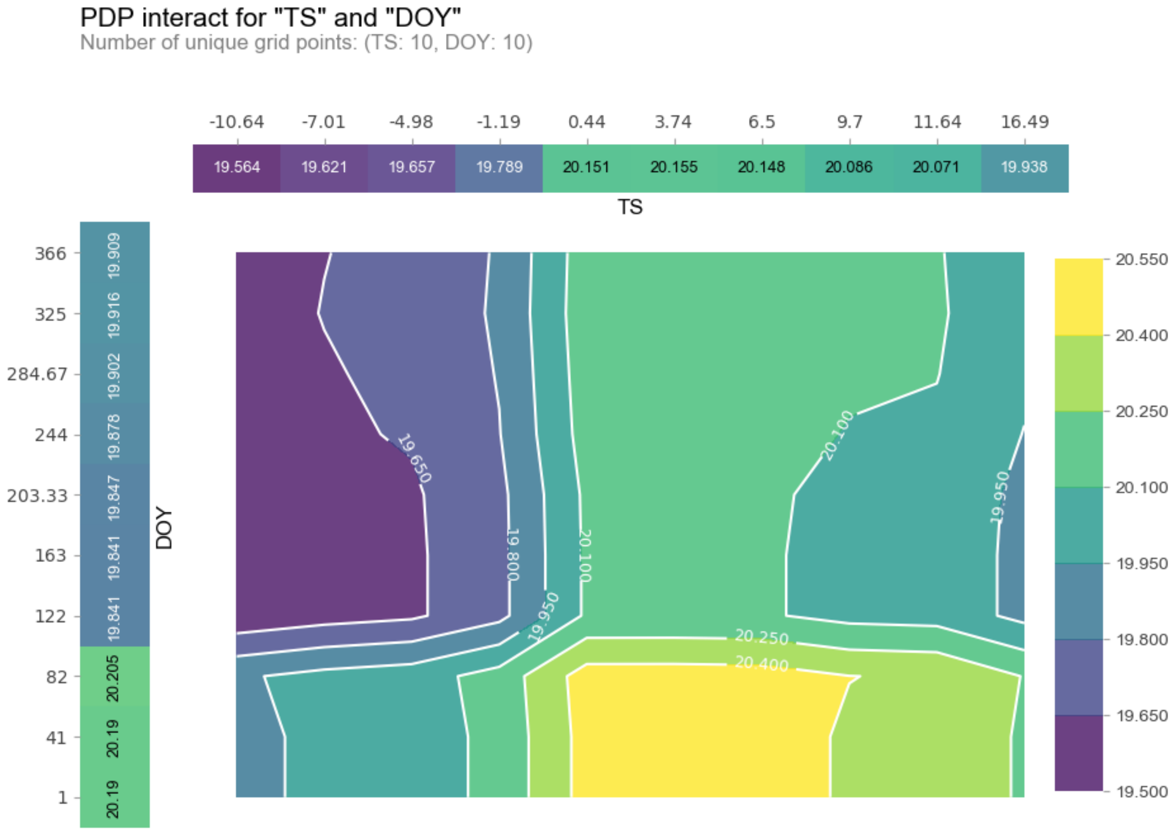
`model:` a fitted sklearn model `data:` pandas DataFrame data set on which the model is trained `model_features:` list or 1-d array list of model features `features:` list [feature1, feature2] `num_grid_points:` list, default=None [feature1 num_grid_points, feature2 num_grid_points] `grid_types:` list, default=None [feature1 grid_type, feature2 grid_type] `percentile_ranges:` list, default=None [feature1 percentile_range, feature2 percentile_range] `grid_ranges:` list, default=None [feature1 grid_range, feature2 grid_range] `cust_grid_points:` list, default=None [feature1 cust_grid_points, feature2 cust_grid_points] `memory_limit:` float, (0, 1) fraction of memory to use `n_jobs:` integer, default=1 number of jobs to run in parallel. pdp_interact_out: (list of) instance of PDPInteract for multi-class, it is a list `plot_type:` str, optional, default='contour' type of the interact plot, can be 'contour' or 'grid' `x_quantile:` bool, default=False whether to construct x axis ticks using quantiles `plot_pdp:` bool, default=False whether to plot pdp for each feature `which_classes:` list, optional, default=None which classes to plot, only use when it is a multi-class problem `figsize:` tuple or None, optional, default=None size of the figure, (width, height) `ncols:` integer, optional, default=2 number subplot columns, used when it is multi-class problem `plot_params:` dict or None, optional, default=None parameters for the plot

`plot:` bool, if plt.show() `save:` bool, if save the picture `save_path:` string, path of picture saved, default='pdp.jpg'

`return:` PDP dataframe

```
from ExplainAI.explainers.pdp.pdp import partial_dependence_plot_2d
pd2=partial_dependence_plot_2d(model=m,data=x,model_features=f,features=
["TS",'DOY'],plot=True,save=True)
print(pd2)
```

```
#print(pd2)
     TS         DOY      preds
0  -10.636    1.000000  19.985679
1  -10.636   41.000000  19.984508
2  -10.636   82.000000  19.987828
3  -10.636  122.000000  19.877804
4  -10.636  163.000000  19.864064
..     ...         ...        ...
95  16.486  203.333333  19.889768
96  16.486  244.000000  19.894714
97  16.486  284.666667  19.896580
98  16.486  325.000000  19.888302
99  16.486  366.000000  19.865036
```



Individual conditional expectation

ICE was proposed by Goldstein et al. (2015). The ICE concept is given by:

$$p(x(s,j)) = f(x(s,j), xc)$$

For a feature of interest, ICE plots highlight the variation in the fitted values across the range of covariate. In other words, the ICE provides the plots of dependence of the predicted response on a feature for each instance separately.

```
individual_conditional_exception()
```

`:param data:` pandas.DataFrame, the sample data from which to generate ICE curves

`:param column:` str, the name of the column in `data` that will be varied to generate ICE curves

`:param predict:` callable, the function that generates predictions from the model.

`:param num_grid_points:` None or int,the number of grid points to use for the independent

`:param frac_to_plot:` float, the fraction of ICE curves to plot.

`:param plot_points:` bool, whether or not to plot the original data points on the ICE curves.

`:param x_quantile:` bool, if `True`, the plotted x-coordinates are the quantiles of `ice_data.index`

`:param plot_pdp:` if `True`, plot the partial depdendence plot. In this case, `pdp_kwargs` is passed as keyword arguments to `plot`.

`:param centered:` if `True`, each ICE curve is centered to zero at the percentile closest to `centered_quantile`.

`:param color_by:` If a string, color the ICE curve by that level of the column index.

`:param cmap:` matplotlib Colormap

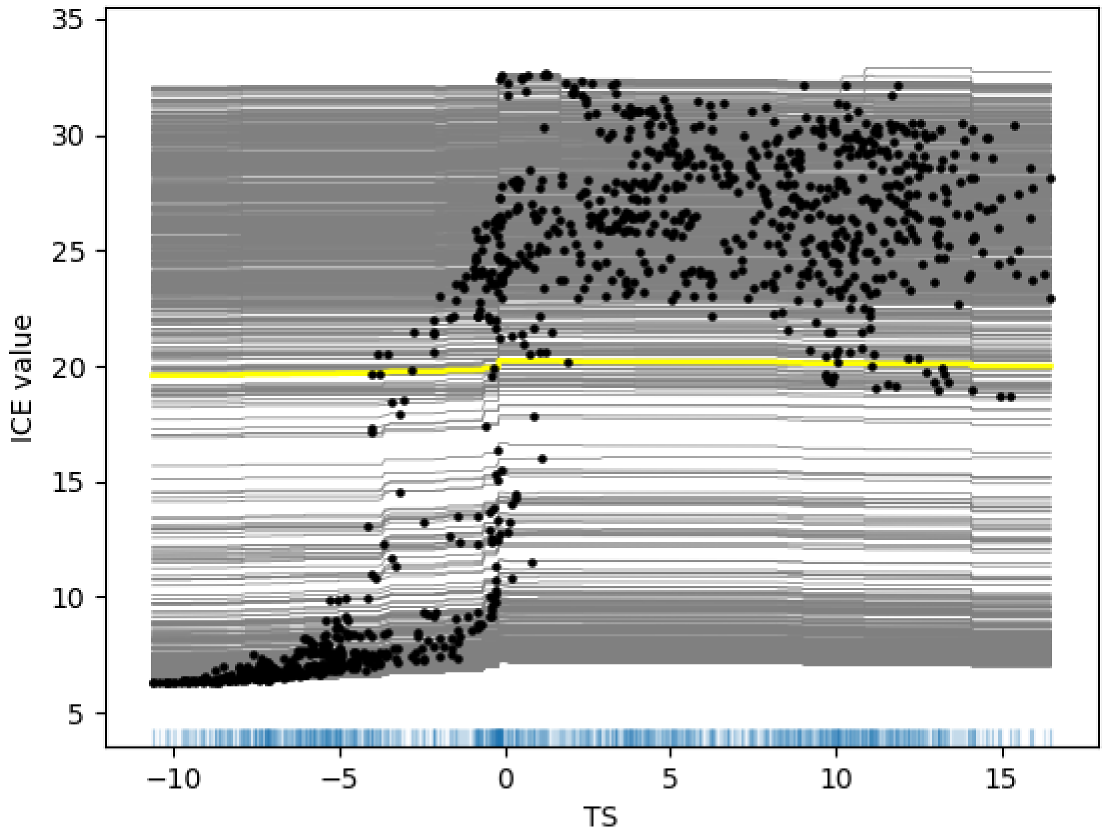
`:param ax:`: `None` or `matplotlib` `Axes`,the `Axes` on which to plot the ICE curves

`plot:` bool, if plt.show() `save:` bool, if save the picture `save_path:` string, path of picture saved, default='pdp.jpg'

`retuen:` Dataframe, ICE data

For example,

```
from ExplainAI.explainers.ice.ice import individual_conditional_exception
i=individual_conditional_exception(data=x, feature='TS',
model=m,plot=True,save=True,save_path='ice.jpg')
i.to_csv('ice.csv')
```



Accumulated Local Effect

The ALE is a more sophisticated method to evaluate the feature effects, owing to averaging the differences in the prediction model for conditional distribution (Apley et al., 2019). One-dimensional ALE (1D ALE) shows the dominate effects with the feature of interest variation.

$$f_j(x) = \sum^h 1/(n_j(k)) \sum_m [f(z(k,j), x_(i, \backslash j)]) - f(z(k-1,j), x(i, \backslash j)) - c$$

$$h = k_j(k)$$

$$m = (i : x(i,j) \in N_j(k))$$

And the constant c is calculated to make sure the following equation:

$$1/n \sum^n f_j(x) = 0$$

where fj donates the ALE values and it visualizes the main effect dependence of modelling on x_j, x(i,\j)=(x(i,l):l=1, ...,d;l≠j), where the subscript \j means all feature but the j-th. Similarly, Nj (k)=(z(k-1,j),z(k,j);k=1,2,...,K) donates a sufficiently fine partition of the sample range of x(i,j) into K intervals.

accumulated_local_effect_1d() offers one-dimentional ALE.

```
accumulated_local_effect_1d()
```

`parameter:`

`model :` object or function A Python object that contains 'predict' method. It is also possible to define a custom prediction function with 'predictor' parameters that will override 'predict' method of model. `train_set :` pandas DataFrame Training set on which model was trained. `features :` string or tuple of string A single or tuple of features' names. `bins :` int Number of bins used to split feature's space. `monte_carlo :` boolean Compute and plot Monte-Carlo samples. `predictor :` function Custom function that overrides 'predict' method of model.

`monte_carlo_rep :` int Number of Monte-Carlo replicas. `monte_carlo_ratio :` float Proportion of randomly selected samples from dataset at each Monte-Carlo replica.

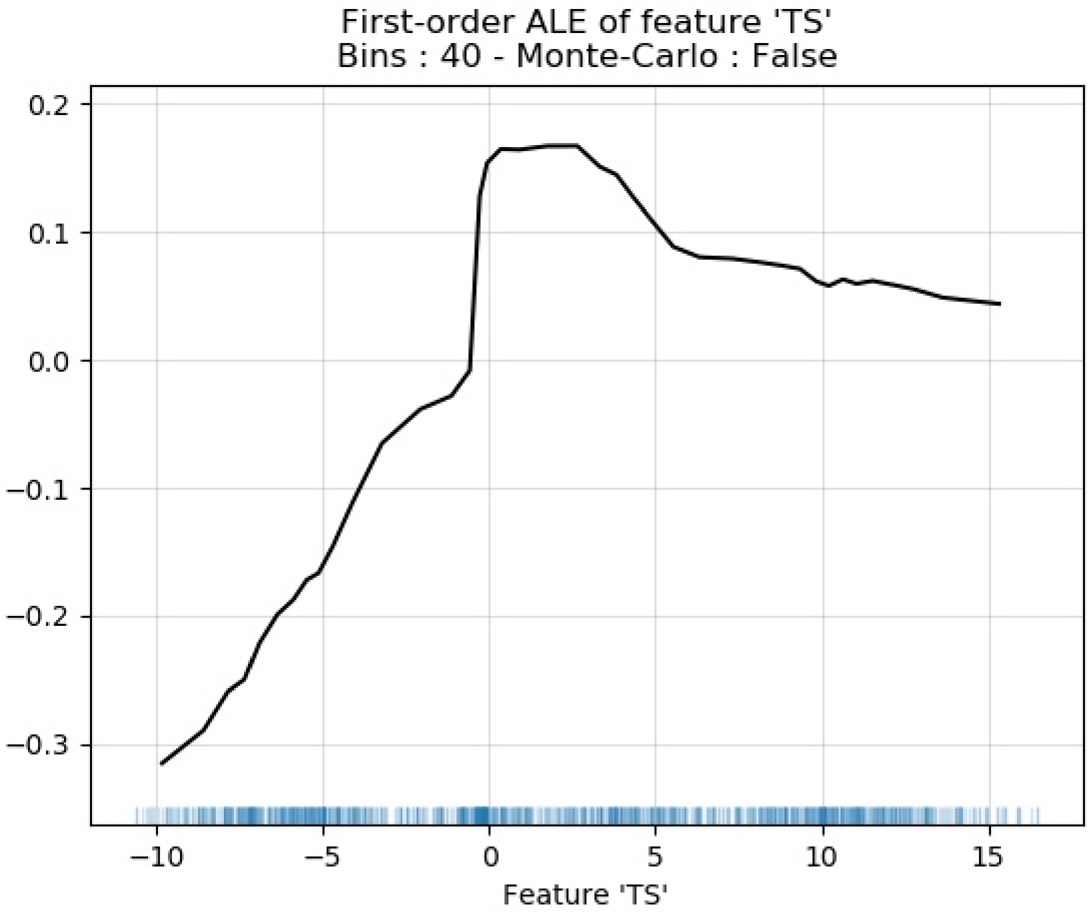
`plot:` bool, if plt.show() `save:` bool, if save the picture `save_path:` string, path of picture saved, default='pdp.jpg'

`retuen:` Dataframe, ALE data

For example,

```
from ExplainAI.explainers.ale.ale import accumulated_local_effect_1d
a1=accumulated_local_effect_1d(model=m, train_set=x,
features='TS',plot=False,save=True,monte_carlo=False)
print(a1)
```

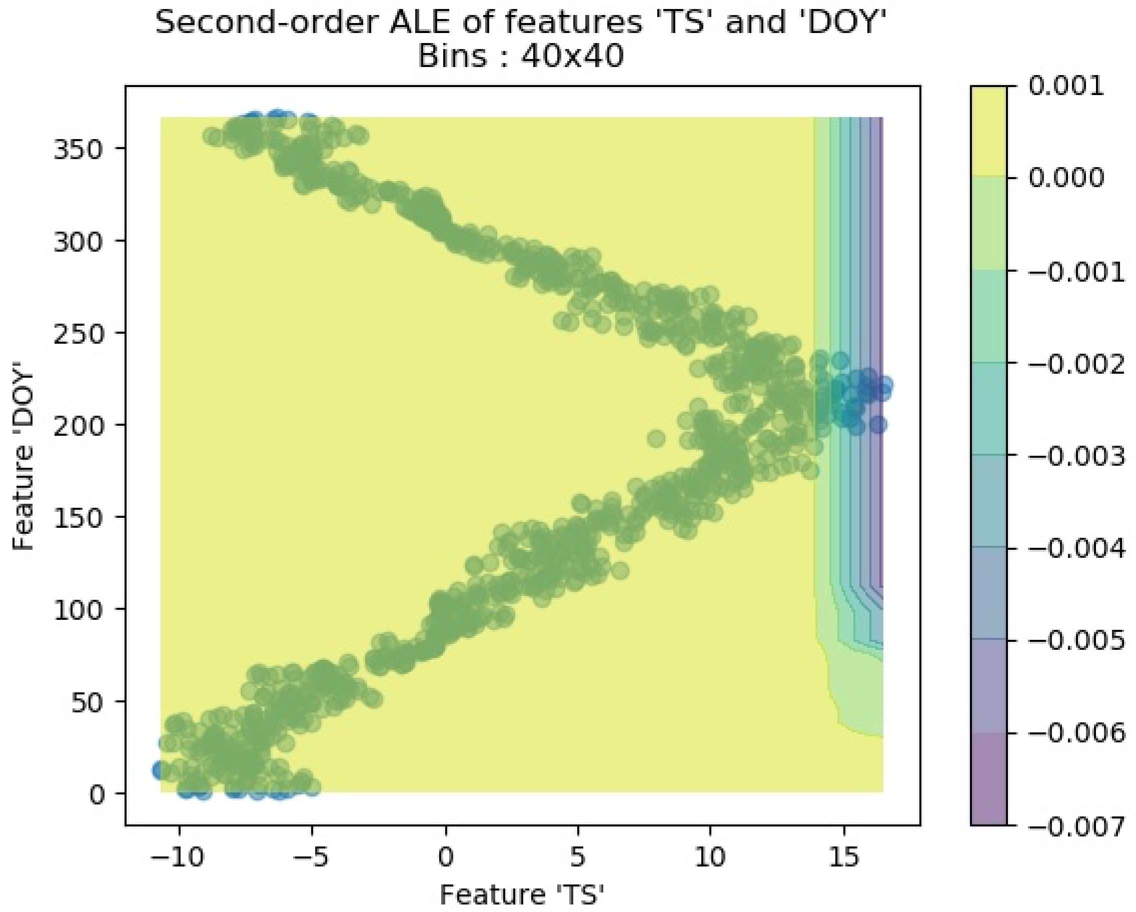
```
#print(a1)
    quantiles       ALE
0   -9.007500 -0.315112
1   -8.151000 -0.289409
2   -7.540875 -0.259159
3   -7.168500 -0.249974
4   -6.639375 -0.221270
5   -6.096750 -0.198948
6   -5.685750 -0.187578
...
37  13.109000  0.055397
38  14.142500  0.048841
39  16.486000  0.043952
```



Two-dimensional ALE (2D ALE) solely displays the additional effect of an interaction between two features, which does not contain the main effect of each feature.

```
from ExplainAI.explainers.ale.ale import accumulated_local_effect_2d
a2=accumulated_local_effect_2d(m, train_set=x, features=['TS', 'DOY'], plot=False,
bins=40,save=True)
```

```
         -10.6360  -9.0075   -8.1510   ...   13.1090   14.1425   16.4860
1.000     0.000119  0.000119  0.000119  ...  0.000119  0.000119  0.000119
10.000    0.000119  0.000119  0.000119  ...  0.000119  0.000119  0.000119
...
338.000   0.000119  0.000119  0.000119  ...  0.000119  0.000119 -0.006281
347.250   0.000119  0.000119  0.000119  ...  0.000119  0.000119 -0.006281
356.625   0.000119  0.000119  0.000119  ...  0.000119  0.000119 -0.006281
366.000   0.000119  0.000119  0.000119  ...  0.000119  0.000119 -0.006281
```



Shapley values

Considering the all-possible interactions and redundancies between features, all combinations of features are tested. Apart from the evaluation for the training set, the Shapley values method can be applied on any data subset or even a single instance (Shapley and Roth, 1988). The Shapley values of a feature value is its contribution to the predicted result, weighted and summed over all possible feature value combinations (Štrumbelj and Kononenko, 2013):

$$\varphi(i,j)(val) = \sum_S |S|!(p-|S|-1)!/p![val(S \cup x(i,j)) - val(S)]$$

$$S \subseteq [x(i,1),\ldots,x(i,p)] \setminus x(i,j)$$

where S is a subset of the features used in an alliance, x_i is the vector of feature value of interest of instance j, p donates the number of features, and val is the prediction for feature values in subset S that are marginalized over features that are not included in subset S.

Here are the two versions of Shapley values in different operating systems. For windows only, the pictures would be captured in the console which requires the manual saving. For Linux and Windows, you can choose your save path to save the pictures.

For Windows only

At first, the Shapley values function is treated as a vessel.

```
shap_obj=shap_func()
```

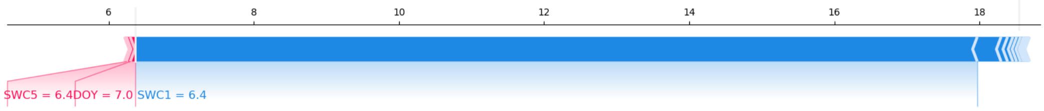
`:param model:` sklearn model object `:param x:` dataframe, input feature dataset `:param features:` list, feature names

record_shap() can export calculated shapley values in "shap.csv".

```
from ExplainAI.explainers.shap_func.shap_func import shap_func
ss=shap_func(m,x)
ss.record_shap()
```

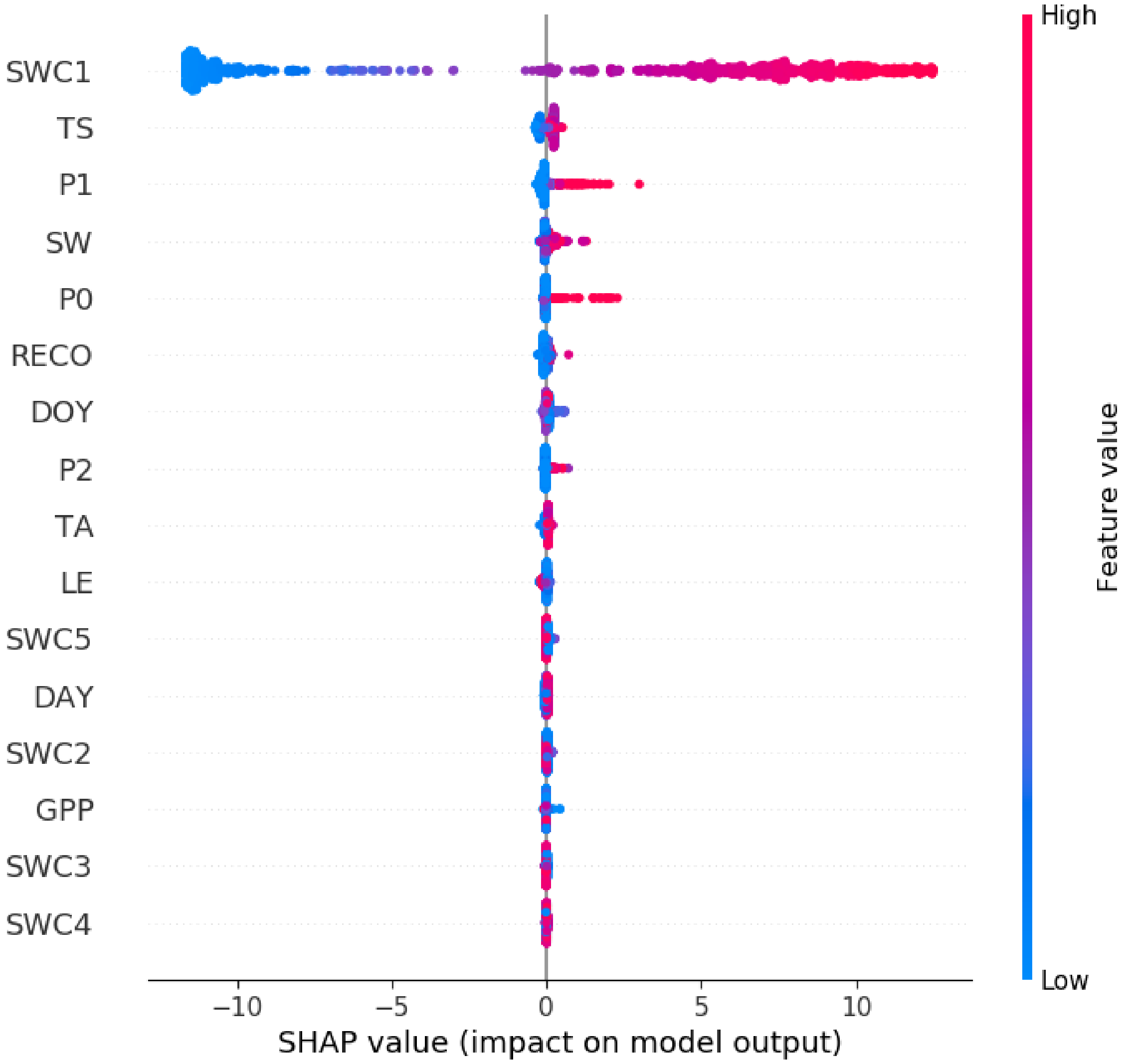
single_shap() offers shapley values for an individual instance.

```
ss.single_shap(nth=6)
#nth donates sequence of instance of interest.
```



feature_value_shap() provides shapley values distribution with feature values.

```
ss.feature_value_shap()
```

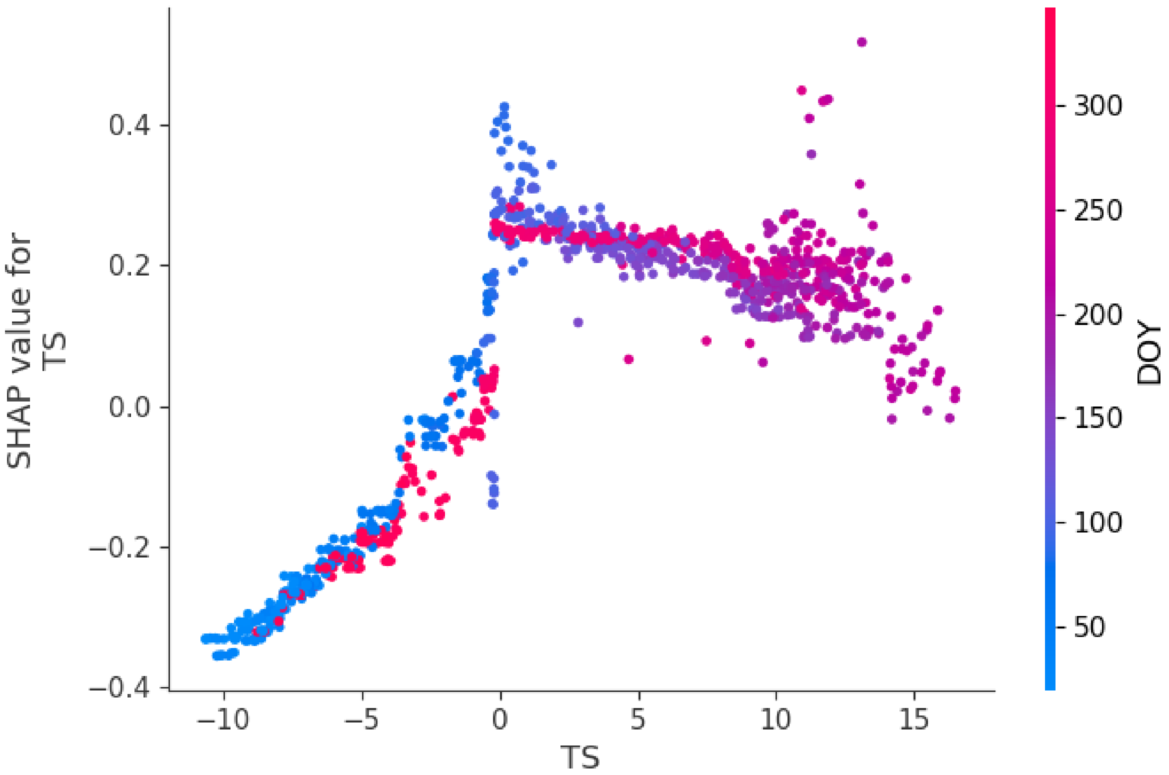


time_shap() provides Shapley values distribution with time series.

```
ss.time_shap()
```

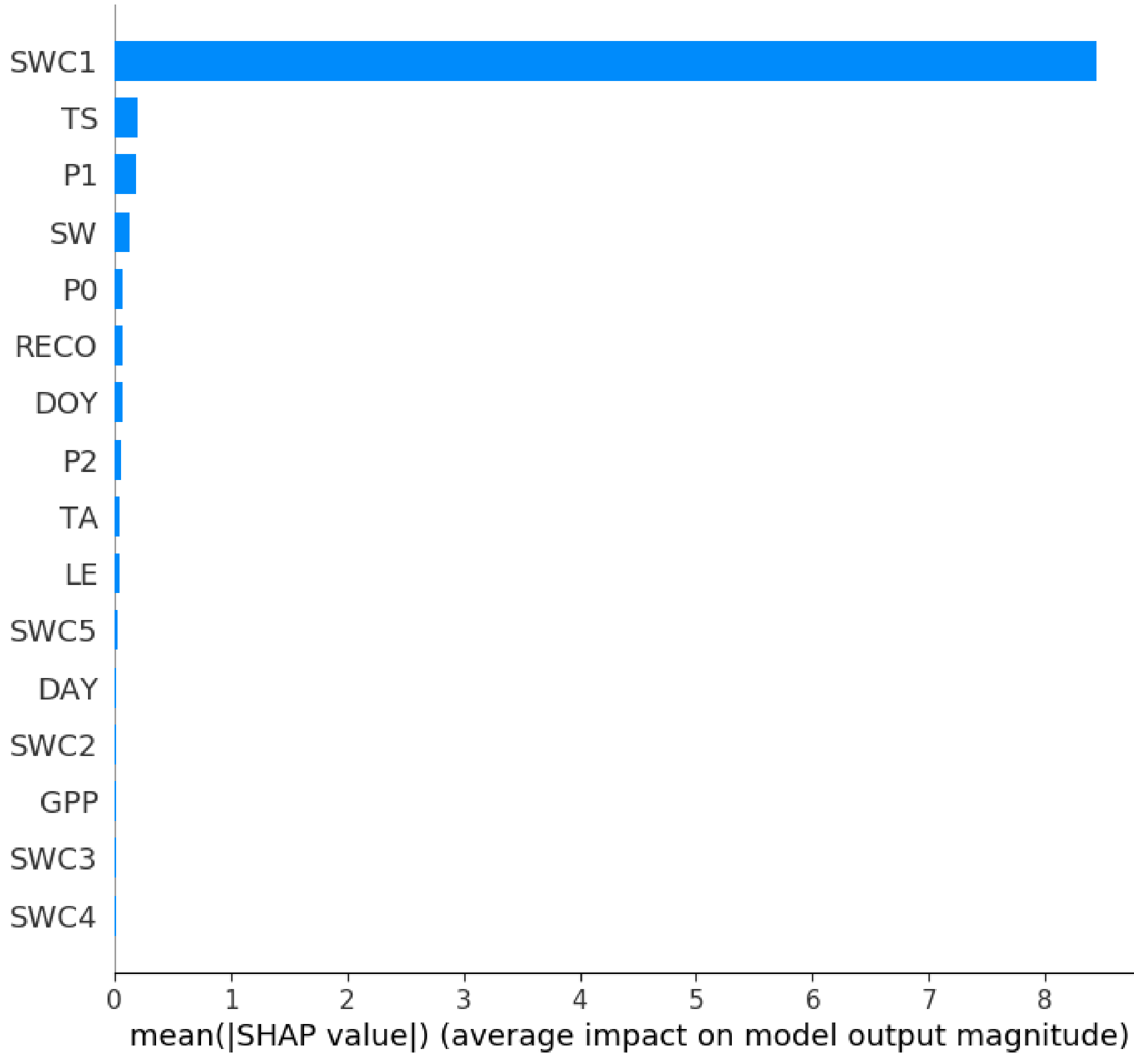
depend_shap() provides Shapley values distribution with feature variation.

```
ss.depend_shap(depend_feature='TS')
```



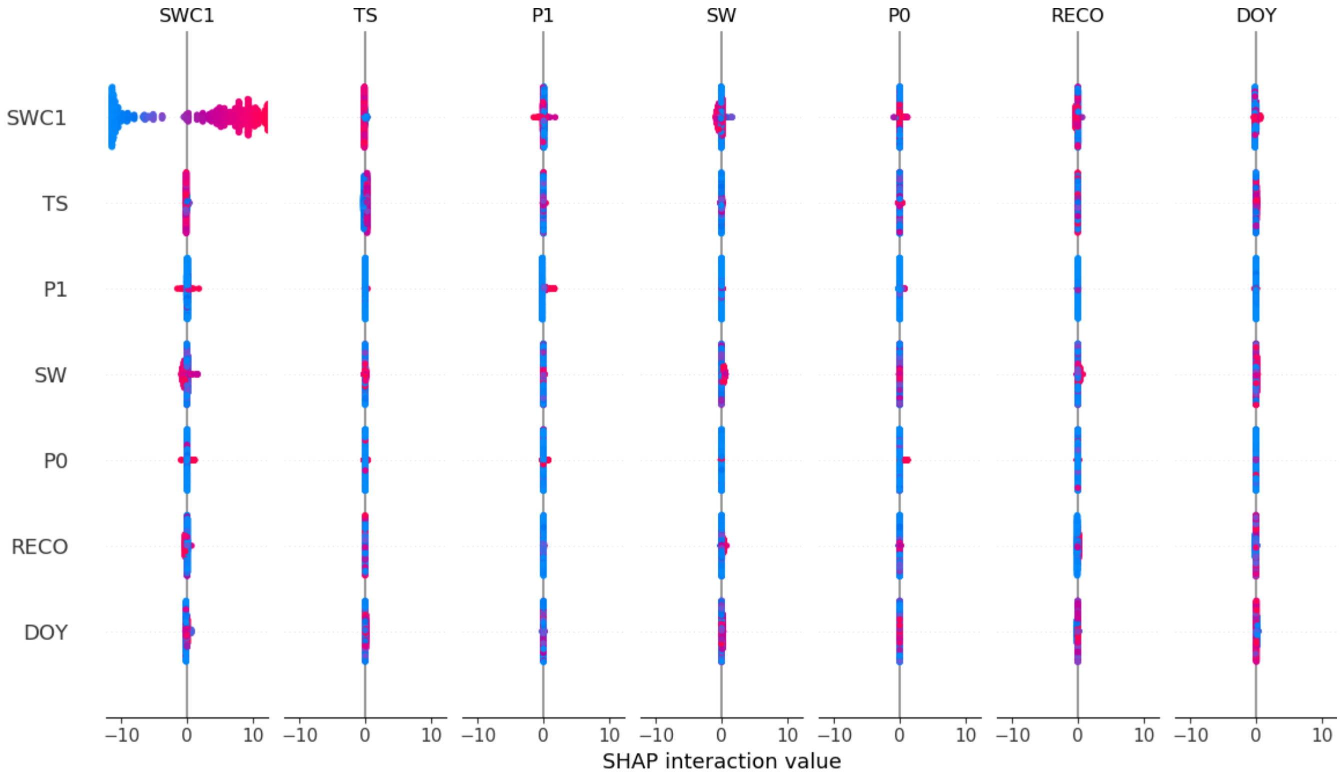
mean_shap() provides averaged Shapley values of features.

```
ss.mean_shap()
```



intera_shap() provides averaged Shapley values under two features' interaction.

```
ss.intera_shap()
```



For Linux and Windows

- shap_explainations_instance() offers shapley values for an individual instance.

```
shap_explainations_instance()
```

`:param model:` sklearn model object `:param x:` dataframe, input feature dataset `:param features:` list, feature names `:param nth:` int, sequence of instance of interest `:param plot:` bool, if plt.show() `:param save:` bool, if save the picture `:param save_path:` string, path of picture saved, default='pdp.jpg'

`:return:` dataframe, shap valure of the instance

For example,

```
from ExplainAI.explainers.shap_func.shap_func import shap_explainations_instance
s=shap_explainations_instance(m,x,f,nth=4)
print(s)
```

```
#print(s)
DAY       0.005810
DOY       0.057023
TA       -0.006199
SW       -0.017431
TS       -0.024737
LE        0.012880
GPP       0.022192
RECO      0.003033
SWC1    -12.311931
SWC2      0.105164
SWC3      0.013828
```

```
SWC4     0.005863
SWC5     0.004599
P0      -0.017164
P1      -0.051249
P2      -0.004497
```

- shap_explainations_instance() offers averaged Shapley values of features.

```
shap_explainations_mean()
```

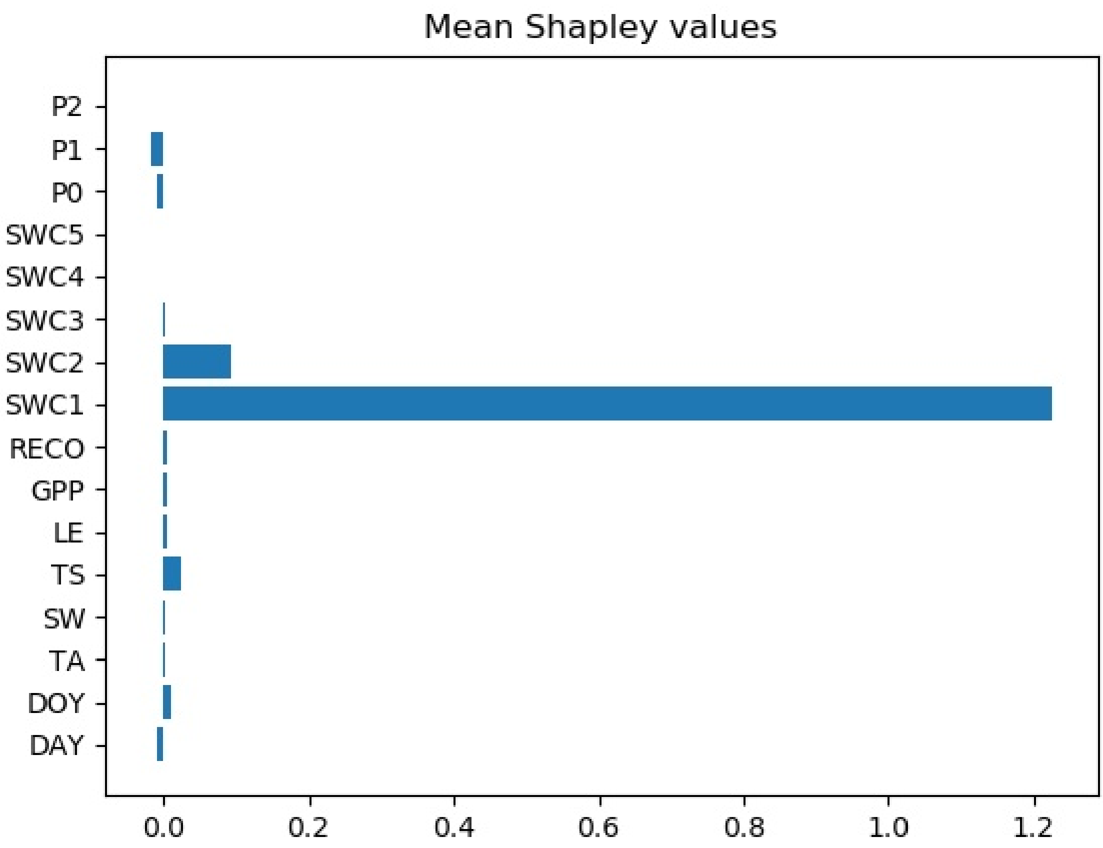
`:param model:` sklearn model object `:param x:` dataframe, input feature dataset `:param features:` list, feature names `:param plot:` bool, if plt.show() `:param save:` bool, if save the picture `:param save_path:` string, path of picture saved, default='shap_mean.jpg' `:param describe:` bool, if True, the shapley values will be described statistically witl mean, std, min, max and so on.

`:return:` dataframe, mean shapley values

For example,

```
from ExplainAI.explainers.shap_func.shap_func import shap_explainations_mean
sm=shap_explainations_mean(m,x,f)
print(sm)
```

```
features  Shapley_mean
DAY      -0.007492
DOY       0.009586
TA       0.002884
SW       0.001441
TS       0.023729
LE       0.005203
GPP       0.006358
RECO       0.006129
SWC1       1.226364
SWC2       0.092463
SWC3       0.003289
SWC4       0.000869
SWC5       0.000134
P0      -0.007938
P1      -0.018106
P2      -0.000417
```



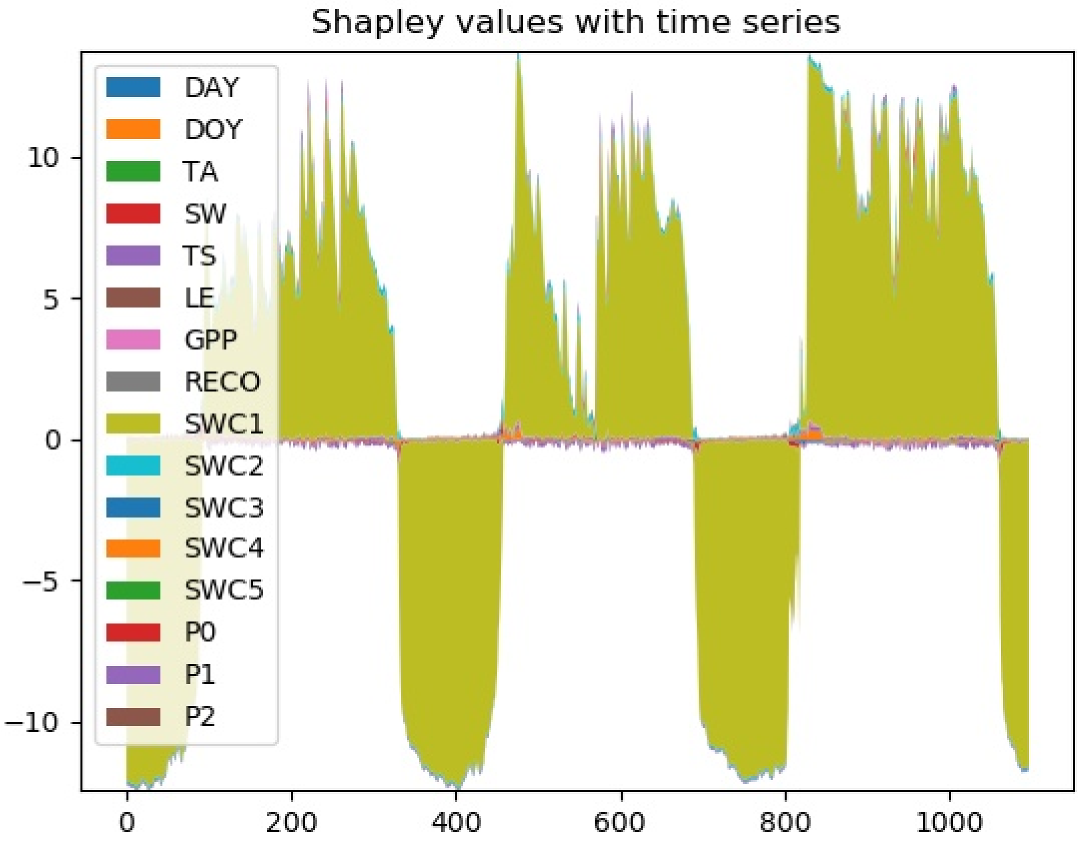
- shap_explainations_time() provides Shapley values distribution with time series.

  ```
  shap_explainations_time()
  ```

  `:param model:` sklearn model object `:param x:` dataframe, input feature dataset `:param features:` list, feature names `:param plot:` bool, if plt.show() `:param save:` bool, if save the picture `:param save_path:` string, path of picture saved, default='shap_mean.jpg' `:param describe:` bool, if True, the shapley values will be described statistically witl mean, std, min, max and so on.

  `:return:` dataframe, mean shapley values

```
from ExplainAI.explainers.shap_func.shap_func import shap_explainations_time
st=shap_explainations_time(m,x,f)
```



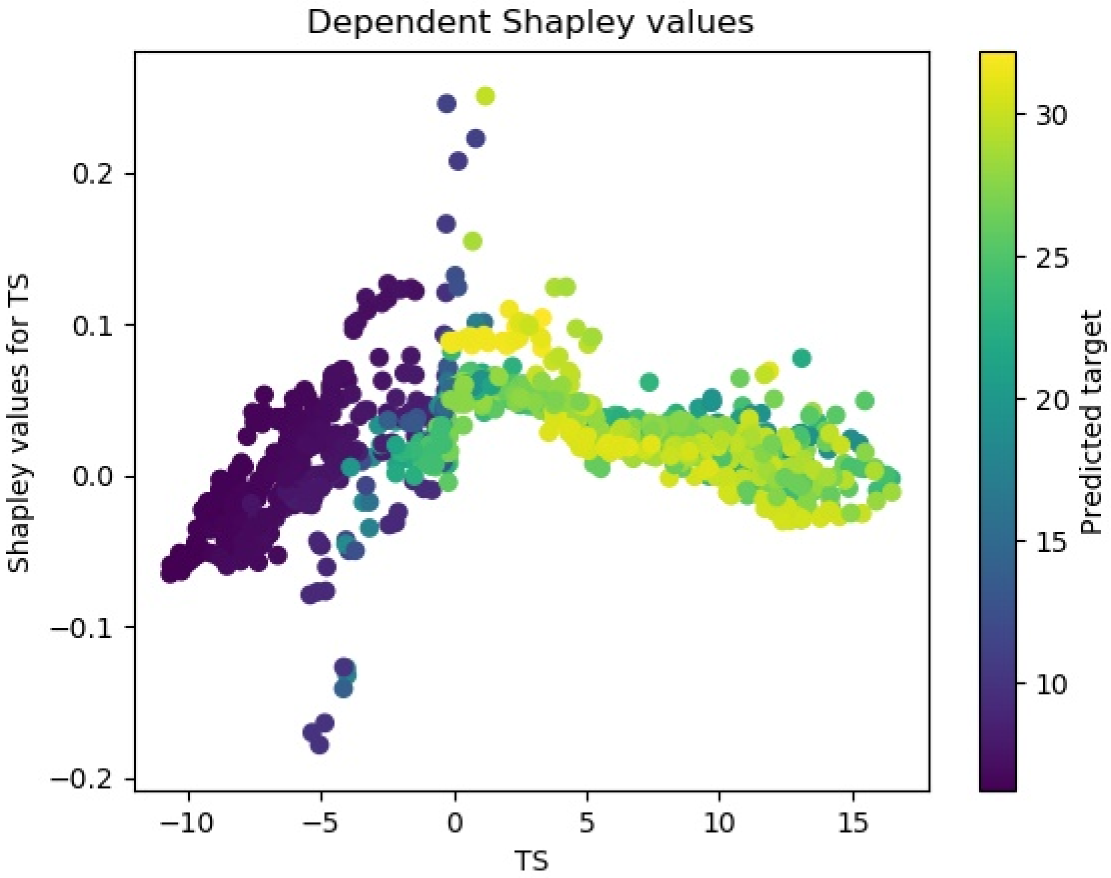
- shap_explainations_dependence() provides Shapley values distribution with feature variation.

```
shap_explainations_dependence()
```

`:param model:` sklearn model object `:param x:` dataframe, input feature dataset `:param features:` list, feature names `:param plot:` bool, if plt.show() `:param save:` bool, if save the picture `:param save_path:` string, path of picture saved, default='shap_time.jpg' `:param describe:` bool, if True, the shapley values will be described statistically witl mean, std, min, max and so on.

`:return:` dataframe, shapley values

```
from ExplainAI.explainers.shap_func.shap_func import shap_explainations_dependence
shap_explainations_dependence(m,x,f,dependence_feature='TS')
```



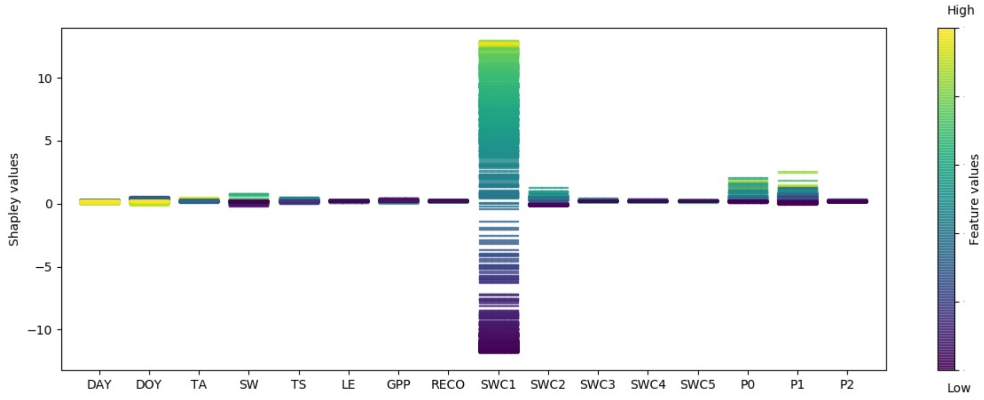
shap_explainations_feature_value() provides shapley values distribution with feature values.

```
shap_explainations_feature_value()
```

`:param model:` sklearn model object `:param x:` dataframe, input feature dataset `:param features:` list, feature names `:param plot:` bool, if plt.show() `:param save:` bool, if save the picture `:param save_path:` string, path of picture saved, default='shap_time.jpg' `:param describe:` bool, if True, the shapley values will be described statistically witl mean, std, min, max and so on.

`:return:` dataframe, shapley values

```
from ExplainAI.explainers.shap_func.shap_func import shap_explainations_feature_value
shap_explainations_feature_value(m,x,f)
```



Local Interpretable Model-Agnostic Explanations

Local Interpretable Model-Agnostic Explanations (LIME) is an attempt to make these complex models at least partly understandable (Ribeiro, et al., 2016). Generally, the surrogate model after training, aims to approximate the predictions of the underlying black box model.

```
lime_explainations()
```

`:param model:` sklearn model object `:param train_data:` dataframe, input feature dataset `:param features:` list, feature names `:param target:` string, target feature name `:param instance_sequence:` int, instance number `:param num_features:` int, number of features `:param plot:` bool, if plt.show() `:param save:` bool, if save the picture `:param save_path:` string, path of picture saved, default='lime.jpg'

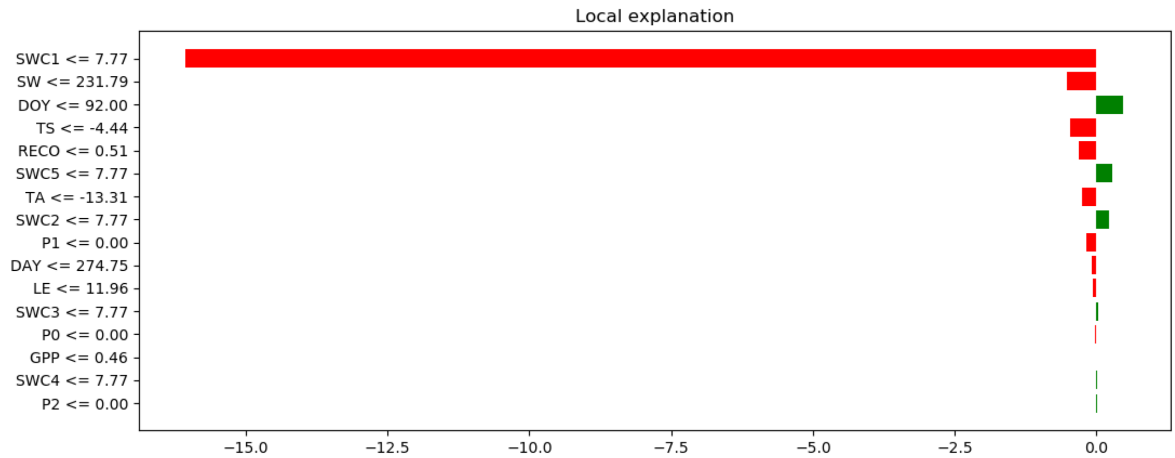
`:return:` dataframe, lime values

For example,

```
from ExplainAI.explainers.lime_func.lime_output import lime_explainations
lime=lime_explainations(m, train_data=x, features=f, target="TS",
instance_sequence=2,num_features=len(f),
plot=True,save=True,save_path='lime.jpg')
print(lime)
```

```
#print(lime)
    feature feature_upper_val feature_lower_val   lime_val
0      SWC1              7.77              7.77 -16.898005
1        LE             11.96             11.96  -0.437438
2       DOY             92.00             92.00   0.348370
3      SWC3              7.77              7.77   0.259343
4      SWC5              7.77              7.77   0.246198
5        P1              0.00              0.00  -0.239537
6       GPP              0.46              0.46   0.157023
7      SWC4              7.77              7.77   0.148707
8      RECO              0.51              0.51  -0.123809
9        SW            231.79            231.79  -0.109733
10      DAY            274.75            274.75   0.059360
```

```
11     SWC2           7.77           7.77   0.052277
12       TS          -4.44          -4.44  -0.048100
13       P0           0.00           0.00  -0.029209
14       TA         -13.31         -13.31  -0.016023
15       P2           0.00           0.00   0.011043
```



Contributing

ExplainAI uses MIT license; contributions are welcome!

- Source code: https://github.com/HuangFeini/ExplainAI.git

ExplainAI supports Python 3.6+ .

References

1. Apley, D. W., and Zhu, J.: Visualizing the effects of predictor variables in black box supervised learning models. arXiv.org. https://arxiv.org/abs/1612.08468, 2019.
2. Breiman, L.: Classification and regression based on a forest of trees using random inputs, Mach. Learn., 45(1), 5–32, doi:10.1023/a:1010933404324, 2001.
3. Friedman, J. H.: Greedy function approximation: A gradient boosting machine. The Annals of Statistics, 29(5), doi:10.1214/aos/1013203451, 2001.
4. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Comput. Surv., 51(5), 1–42, doi:10.1145/3236009, 2019.
5. Štrumbelj, E., and Kononenko, I.: Explaining prediction models and individual predictions with feature contributions. Knowl. Inf. Syst., 41(3), 647–665, doi:10.1007/s10115-013-0679-x, 2013.

6. Shapley, L.S., and Roth, A.E.: The Shapley value: essays in honor of Lloyd S. Shapley. Cambridge University Press. https://www.amazon.com/Shapley-Value-Essays-Honor-Lloyd-ebook/dp/B00IE6MSSY, 1988.

Citation

please cite the following for the usage of ExplainAI toolbox.

Copyright licence

Changelog

In this version, you might have some problems as follows. And you can try the sulotion to fix that.

1. if in the linux, 'display' issue still exists.

   import matplotlib.pyplot as plt

   -->plt.switch_backend('agg')

2. About the sklearn version.

   from sklearn.metrics import check_scoring

   --> vi sklearn/metrics/**init**.py

   -->from scorer import check_scoring

   -->all=['check_scoring']