**Manual**

LoadData Module

1. getTXTpv(filepath,filename)

TXT data format: one PV name in every row.

**Parameters:**

filepath: str

File absolute path.

filename: str

File name.

return: list

1. generate\_live\_data (duration, period, pvnames)

Get EPICS live data.

**Parameters**:

duration: int

Define scan time in second.

period: int

Define scan period in second.

pvnames: list

pv name list.

return: DataFrame

1. connectChanArch(ipaddr)

Connect Channel Archiver server.

**Parameters:**

ipaddr: str

Server IP address.

return: Channel Archiver server and engine.

1. getChanArchEngineKey(ipaddr,enginename)

Use Archiver engine name to find corresponding key number.

**Parameters**:

Ipaddr: str

Server IP address.

Enginename: str

Archiver engine name.

return: int

1. getChanArchEngineName(ipaddr,enginekey)

Use Archiver engine key number to find corresponding engine name.

**Parameters**:

Ipaddr: str

Server IP address.

enginekey: int

Archiver engine key number.

return: str

1. getChanArch(ipaddr, key, pvnames, start, end, how=0)

Get ChannelArchiver raw data in dictionary, not Dataframe type.

**Parameters**:

Ipaddr: str

Server IP address.

key: int

Engine key number.

pvnames: list

start,end: str, fomat: “%m/%d/%Y %H:%M:%S”

start and end time

how: Channel Archiver get data method: 0-raw, 1-spreadsheet, 2-avg, 3-plot-binning, 4-linear

return: dict

1. getKey(ipaddr,pvnames)

Print PV’s every engine name and key number.

**Parameters:**

Ipaddr: str

Server IP address.

pvnames: list

return: None

1. getKeyWithTime(server, engine,pvnames,start,end)

Get PV’s engine key number with start and end time.

**Parameter**s:

server: Connected Channel Archiver server.

engine: Connected Channel Archiver engine.

pvnames: list

start,end: str, fomat: “%m/%d/%Y %H:%M:%S”

return: list

1. getFormatChanArch(server, engine, pvnames, start, end, merge\_type='inner', interpolate\_type='linear', fillna\_type=None, how=0, dropna=True)

Get formatted Channel Achiver data in DataFrame.

**Parameters:**

server: Connected Channel Archiver server.

engine: Connected Channel Archiver engine.

pvnames: list

start,end: str, fomat: “%m/%d/%Y %H:%M:%S”

merge\_type: str

Define how to align PV data with different timestamp period. outer-interpolate for null data, keep smallest time period; inner-keep biggest time period, delete others; number-Define a time period to align data.

interpolate\_type: str

linear, time, index, values, nearest, zero, slinear, quadratic, cubic, barycentric, krogh, polynomial, spline, piecewise\_polynomial, from\_derivatives, pchip, akima, default linear.

fillna\_type: str:

Define how to handle null data. pad-propagate last valid observation forward to next valid backfill-use NEXT valid observation to fill gap, default None.

how: Channel Archiver get data method: 0-raw, 1-spreadsheet, 2-avg, 3-plot-binning, 4-linear

dropna: Boolean

Whether to remove rows with null data.

return: DataFrame

1. getArchAppl(data\_retrieval\_url,pvnames,start,end,merge\_type)

Get formatted ArchiverAppliance history data in Dataframe.

**Parameters:**

data\_retrieval\_url: ArchiverAppliance server url, like '192.168.44.168:17665/retrieval'.

pvnames: list

start,end: str, fomat: “%m/%d/%Y %H:%M:%S”

merge\_type: str

Define how to align PV data with different timestamp period. outer-interpolate for null data, keep smallest time period; inner-keep biggest time period, delete others; number-Define a time period to align data.

return: DataFrame

1. getLocalFile(filepath,filename,skiprows=0):

Get CSV file data

**Parameters:**

filepath: str

CSV file absolute path.

filename; str

File name.

skiprows: int

Define how many rows to skip when getting data from CSV file.

1. dataset2df (dataset):

sklearn dataset transform to pandas DataFrame type.

**Parameters:**

dataset: sklearn dataset

return: DataFrame

1. np2df(data, col="")

Transform numpy list, dict, ndarray to pandas DataFrame.

**Parameters:**

data: numpy data

col: str

Column name for list and ndarray type

return: DataFrame

1. df2np(data, datatype)

Transform DataFrame to numpy list, dict, ndarray.

**Parameters:**

data: DataFrame

datatype: str

list, dict, ndarray, default dict.

return: define data type.

1. df2other(data, type, path\_or\_buf=None, encoding='utf-8')

Transform DataFrame to CSV, HTML, Excel, clipboard type.

**Parameters:**

data: DataFrame

type:str

csv, html, excel, clipboard.

path\_or\_buf: str

File save path.

encoding: str

Default utf-8.

1. datetime2utc(datestr ,dtformat='%m/%d/%Y %H:%M:%S')

Transform formatted string type timestamp to unix time

**Parameters:**

datestr: str

dtformat: str

Time format.

return：float

1. mergeDF(left,right,merge\_type='left')

Merge two DataFrames.

**Parameters:**

left, right: DataFrame

merge\_type: str

left: use calling frame’s index (or column if on is specified)

right: use other’s index.

outer: form union of calling frame’s index (or column if on is specified) with other’s index, and sort it. lexicographically.

inner: form intersection of calling frame’s index (or column if on is specified) with other’s index, preserving the order of the calling’s one.

DisplayData Module

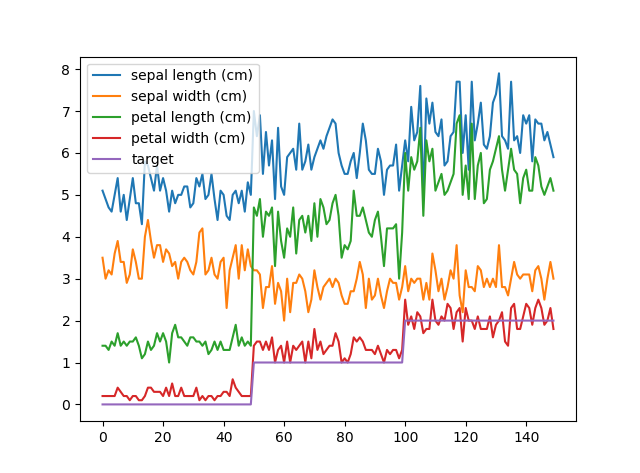
1. showPlot(data)

Show every column data in on plot.

**Parameters:**

data：DataFrame

Example：



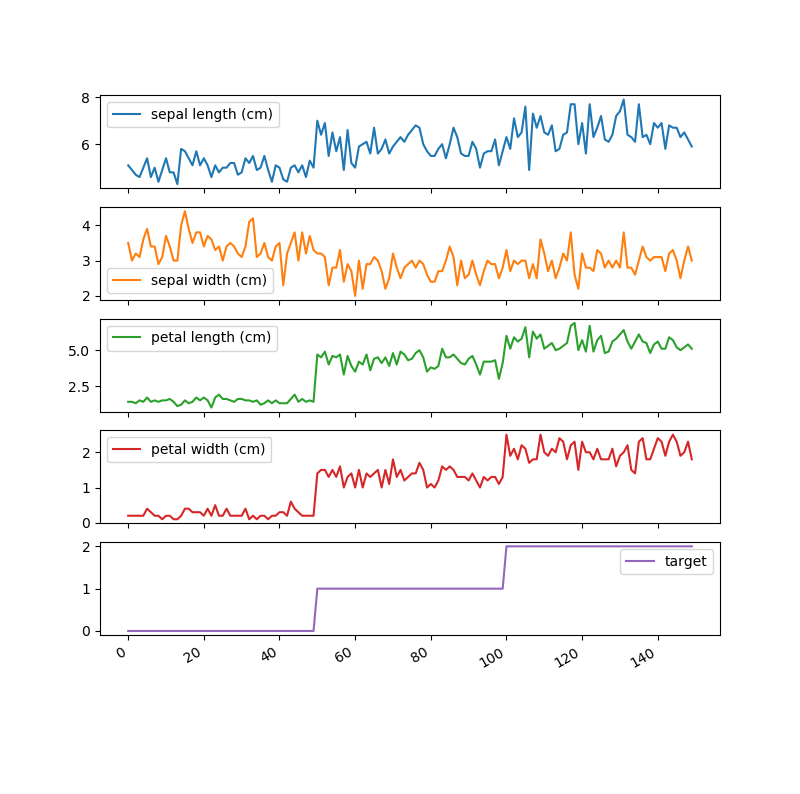
1. showSubPlot(data)

Show data in separate plots.

**Parameters:**

data: DataFrame

Example:



1. showHist(data，column=None)

Show histogram.

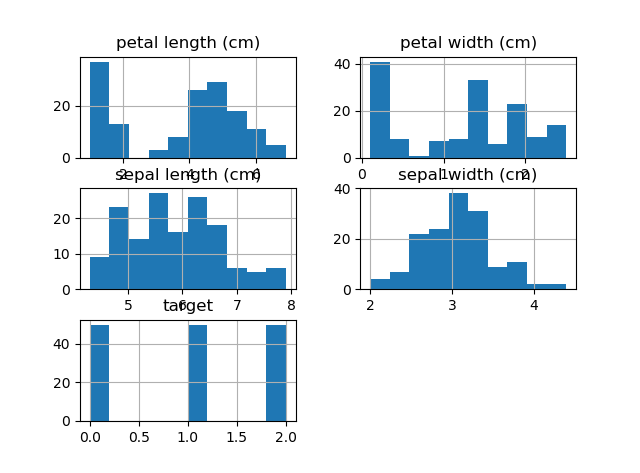
**Parameters:**

data: DataFrame

column: list

Default all column.

Example:



1. showStatistic(data)

Print statistical information of data, including NaN. (count, mean, std, min, 25%,50%, 75%,max)

**Parameters:**

data: DataFrame

Example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
| count | 150.000000 | 150.000000 | 150.000000 | 150.000000 | 150.000000 |
| mean | 5.843333 | 3.054000 | 3.758667 | 1.198667 | 1.000000 |
| std | 0.828066 | 0.433594 | 1.764420 | 0.763161 | 0.819232 |
| min | 4.300000 | 2.000000 | 1.000000 | 0.100000 | 0.000000 |
| 25% | 5.100000 | 2.800000 | 1.600000 | 0.300000 | 1.000000 |
| 50% | 5.800000 | 3.000000 | 4.350000 | 1.300000 | 1.000000 |
| 75% | 6.400000 | 3.300000 | 5.100000 | 1.800000 | 2.000000 |
| max | 7.900000 | 4.400000 | 6.900000 | 2.500000 | 2.000000 |

1. showCorr (data,method='pearson')

Print correlation between columns, including NaN.

**Parameters:**

data: DataFrame

method: str

pearson - standard correlation coefficient, kendall - Kendall Tau correlation coefficient, spearman - Spearman rank correlation

Example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
| sepal length (cm) | 1.000000 | -0.109369 | 0.871754 | 0.817954 | 0.782561 |
| sepal width (cm) | -0.109369 | 1.000000 | -0.420516 | -0.356544 | -0.419446 |
| petal length (cm) | 0.871754 | -0.420516 | 1.000000 | 0.962757 | 0.949043 |
| petal width (cm) | 0.817954 | -0.356544 | 0.962757 | 1.000000 | 0.956464 |
| target | 0.782561 | -0.419446 | 0.949043 | 0.956464 | 1.000000 |

1. showCorrMap (data,method='pearson')

Show correlation heat map.

**Parameters:**

data: DataFrame

method: str

pearson - standard correlation coefficient, kendall - Kendall Tau correlation coefficient, spearman - Spearman rank correlation

Example:



1. showSPLOM(data, hue=None, hue\_order=None, palette=None, vars=None, x\_vars=None, y\_vars=None, kind='scatter', diag\_kind='hist', markers=None, size=2.5, aspect=1, dropna=True, plot\_kws=None, diag\_kws=None, grid\_kws=None)

Plot pairwise relationships.

**Parameters:**

data: DataFrame

hue : string (variable name), optional

Variable in ``data`` to map plot aspects to different colors.

hue\_order : list of strings

Order for the levels of the hue variable in the palette

palette : dict or seaborn color palette

Set of colors for mapping the ``hue`` variable. If a dict, keys

should be values in the ``hue`` variable.

vars : list of variable names, optional

Variables within ``data`` to use, otherwise use every column with

a numeric datatype.

{x, y}\_vars : lists of variable names, optional

Variables within ``data`` to use separately for the rows and

columns of the figure; i.e. to make a non-square plot.

kind : {'scatter', 'reg'}, optional

Kind of plot for the non-identity relationships.

diag\_kind : {'hist', 'kde'}, optional

Kind of plot for the diagonal subplots.

markers : single matplotlib marker code or list, optional

Either the marker to use for all datapoints or a list of markers with

a length the same as the number of levels in the hue variable so that

differently colored points will also have different scatterplot

markers.

size : scalar, optional

Height (in inches) of each facet.

aspect : scalar, optional

Aspect \* size gives the width (in inches) of each facet.

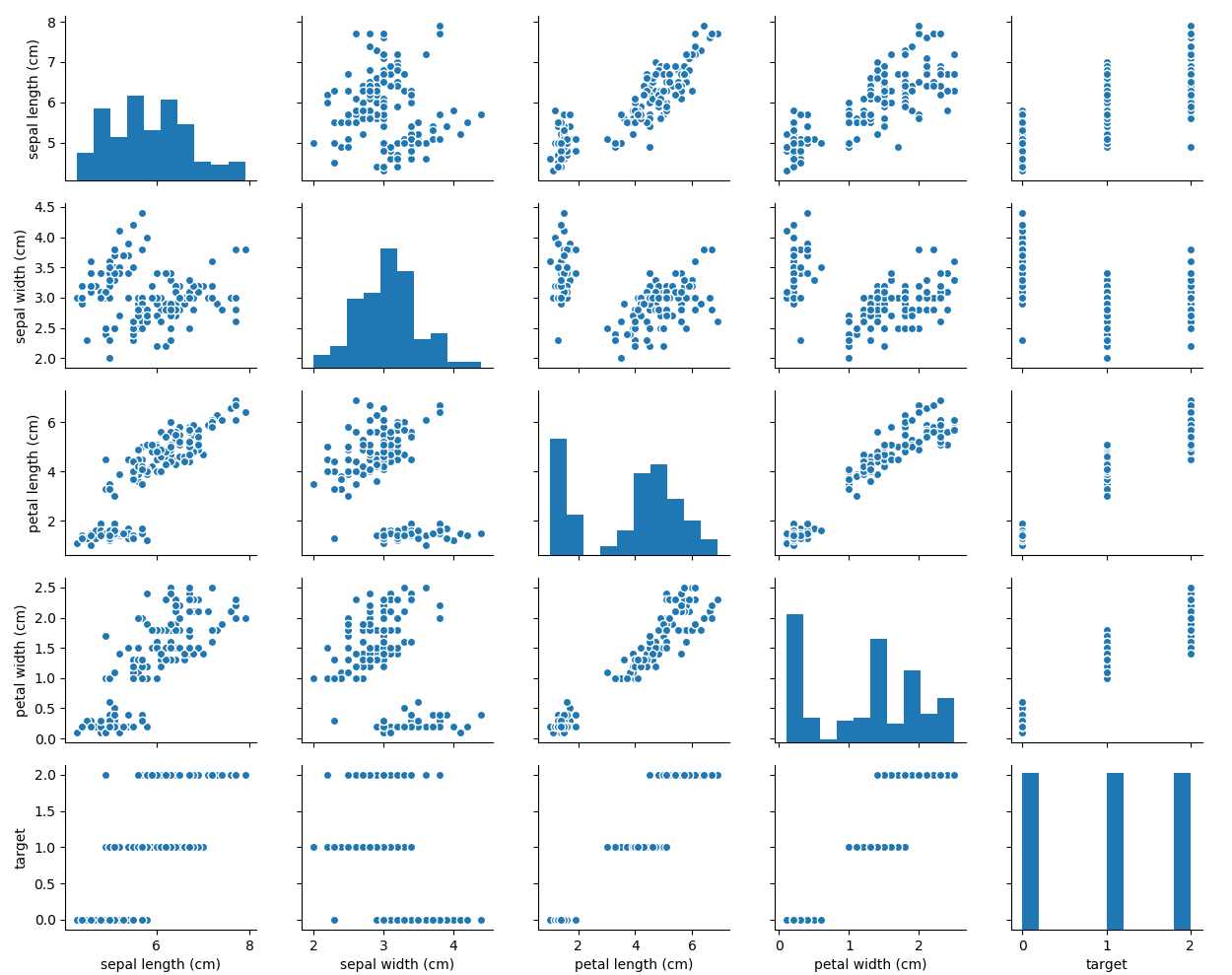
dropna : boolean, optional

Drop missing values from the data before plotting.

{plot, diag, grid}\_kws : dicts, optional

Dictionaries of keyword arguments.

Example:

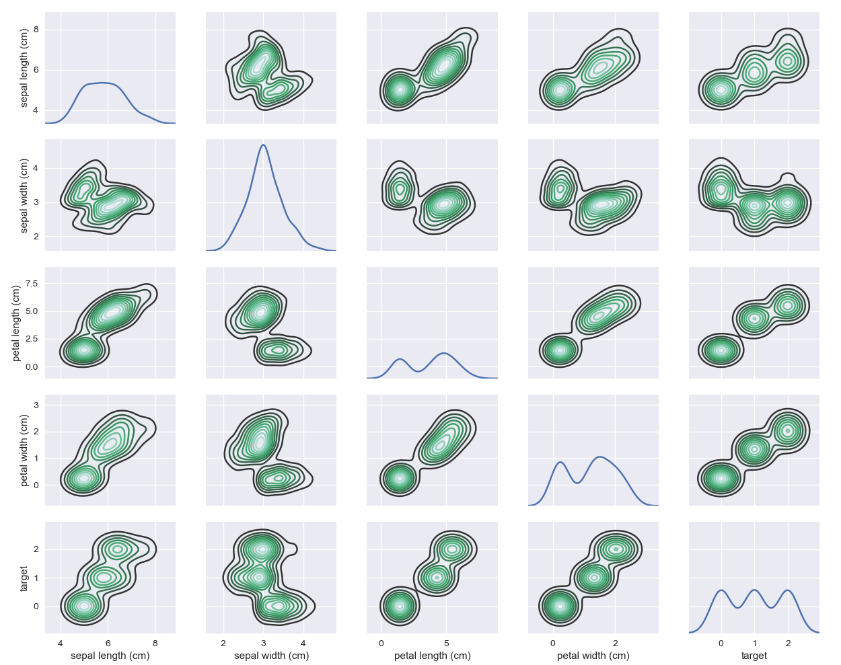


1. showSPLOM\_G(data, hue=None, hue\_order=None, palette=None, hue\_kws=None, vars=None, x\_vars=None, y\_vars=None, diag\_sharey=True, height=2.5, aspect=1, despine=True, dropna=True, size=None):

Subplot grid for plotting pairwise relationships in a dataset

**Parameters:**

Same as showSPLOM()



1. showStd(data,ddof=1):

Print std information.

**Parameters:**

data: DataFrame

ddof：Default 1. Delta Degrees of Freedom. The divisor used in calculations is N - ddof, where N represents the number of elements.

Example:

sepal length (cm) 0.828066

sepal width (cm) 0.433594

petal length (cm) 1.764420

petal width (cm) 0.763161

target 0.819232

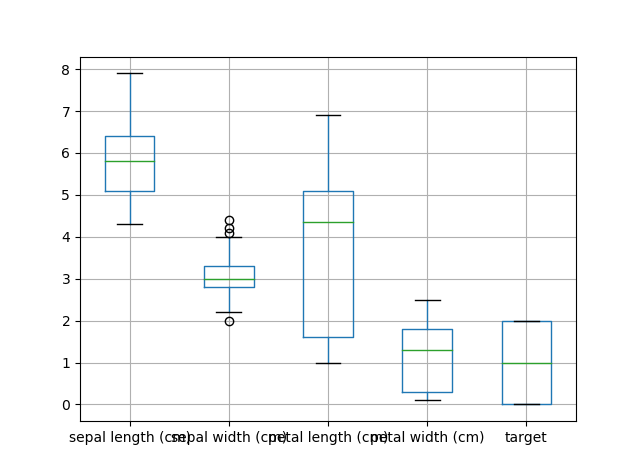
dtype: float64

1. showBins(data)

Show bin plot.

data: DataFrame

Example:



1. plot\_predict(y\_test,y\_pred,title='Figure'):

Show Predict-Test plot

**Parameters:**

y\_test: list

y\_pred: list

title:str

Title, default='Figure'

1. subplot\_predict(y\_test,y\_pred,title='Figure'):

Show Predict-Test plot for every 1000 data point.

**Parameters:**

y\_test: list

y\_pred: list

title:str

Title, default='Figure'

CleanData Module

1. NormData(data,norm\_type,ddof=0):

Normalize data.

**Parameters:**

data: DataFrame

norm\_type: str

min-max - Deviation standardization, linear transformation of the original data, which maps the result value to [0 - 1]. Z-score - The mean and standard deviation of the original data are standardized. The processed data conform to the standard normal distribution, that is, the mean value is 0 and the standard deviation is 1.

return: DataFrame

1. EnumData(data,pvname,cut\_ranges, right=True,enumnames=None):

Transform a column of DataFrame to Enum type

**Parameters:**

data: DatFrame

pvname: str

Column name.

cut\_ranges: divide range,[1,5,7]

right: Indicates whether bins includes the rightmost edge or not. If right == True (the default), then the bins [1, 2, 3, 4] indicate (1,2], (2,3], (3,4]. This argument is ignored when bins is an IntervalIndex.

enumnames：array or boolean, default None. Used as labels for the resulting bins. Must be of the same length as the resulting bins. If False, return only integer indicators of the bins.

1. fillempty (data,method='pad'):

**Parameters:**

data：DataFrame

fillna\_type: str

Define how to handle null data. pad-propagate last valid observation forward to next valid backfill-use NEXT valid observation to fill gap, default pad.

1. DataFrame filter

use of boolean vectors to filter the data. The operators are: | for or, & for and, and ~ for not. These must be grouped by using parentheses.

Example:

data[(data['pvname']>1000)&data['pvname']<20]

1. detect\_burr(data, pv, left=None, right=None, method=0, minimum\_peak\_distance=100)

Detect burr(peak) data, show pic and files.

**Parameters:**

data: DataFrame

pv:str

left, right:number

min and max of peak, if None, use bin boundaries.

method: number

detect peak method,0-scipy.signal. find\_peaks, 1- detect\_peaks function, 2- peakutils.peak.indexes.

minimum\_peak\_distance:number

TrainData Module

1. split\_data(data,t\_pvname,test\_size = 0.3)

Split DataFrame into test and train set.

**Parameters:**

data: DataFrame

t\_pvname: target column name

test\_size：test set percentage

return: X\_train, X\_test, y\_train, y\_test

1. MLLinearRegression(X\_train, X\_test, y\_train, y\_test)
2. MLRidgeRegression(X\_train, X\_test, y\_train, y\_test, alphas=[0.01, 0.1, 0.5, 1, 3, 5, 7, 10, 20, 100])
3. MLGaussianNB\_testmodel(X\_train, X\_test, y\_train, y\_test)
4. MLGaussianNB(data,target\_pv,predict\_data):
5. MLDesionTrees\_testmodel(X\_train, X\_test, y\_train, y\_test)
6. MLDecisionTrees(data,target\_pv,predict\_data)
7. MLPolynomialRegression(X\_train, X\_test, y\_train, y\_test,degree=2)
8. MLKNN\_Regression(X\_train, X\_test, y\_train, y\_test, type='k-fold')
9. MLKNN\_Classification(X\_train, X\_test, y\_train, y\_test, k=5, weights='uniform')
10. MLLogisticRegression\_testmodel(X\_train, X\_test, y\_train, y\_test,multinominal=False,c=1.0):
11. MLLogisticRegression(X,y,predict\_data,multinominal=False,c=1.0):
12. test\_LogisticRegression\_C(X\_train, X\_test, y\_train, y\_test,a=-2,b=4,n=100):
13. MLDBSCAN(data,target\_pv,eps=0.5,min\_samples=10):
14. MLKMeans(data,feature\_pv1,feature\_pv2,cluster=2):
15. MLMLPClassifier(X\_train, X\_test, y\_train, y\_test):
16. LSTMmodel
17. performance score, including r2, absolute\_error, quare\_error, explained\_viarance\_score：
18. performance\_metric\_r2(y\_true, y\_predict):
19. performance\_metric\_abstErr(y\_true, y\_predict):
20. performance\_metric\_sqrErr(y\_true, y\_predict):
21. performance\_metric\_expVia(y\_true, y\_predict):
22. fit\_model\_k\_fold(X, y):
23. fit\_model\_shuffle(X, y):