Documentation

Release alpha-2019.05.10

Najmeh Kaffashzadeh

May 10, 2019

CONTENTS

This is an introduction to the automated QC software. The source code of the software holds in the git repository (https://jugit.fz-juelich.de/n.kaffashzadeh/AutoQC4Env).

CHAPTER

ONE

TUTORIALS

1.1 Motivation

Environmental monitoring is an essential component in humanitys quest to pro- tect Earth and mitigate adversarial effects of climate change, air, water, and soil pollution, and other transformations that are directly or indirectly caused by human activities. Moreover, environmental monitoring drives the development of advanced information technology. This is due to the exponential growth of monitoring data, the open data attitude in major parts of the environmental science communities and of many governmental agencies, and the high degree of standardization that has been achieved with respect to geo-data and metadata. Interoperable data services make it possible to employ environmental data in different contexts from regulatory purposes to earth system science and public information. As the data are used in different contexts, users need to be able to assess the fitness-for-purpose of a dataset and they need to find information that allows them to use the dataset correctly.

Although, at some stages of the quality control workflow human involvement is important to ensure that solid scientific principles are applied to the quality control (QC) process, too much human intervention leads to subjective decisions, arbitrary rules, and adhoc procedures. The aim of an automated QC tool must be to achieve robust and objective decisions about the usefulness of a particular data point and data series for a given analysis task. This means that QC cannot be considered separately from science, and the first prerequisite is to understand the science and context within which the data are being analyzed. Nevertheless, robust statistical measures can help to achieve a certain level of objectivity in the QC process, and automation is the only way to cope with ever larger data sets, in particular in (near) realtime environments.

1.2 Methodology

Different tests may be applied to a given data series or a specific point of this series in order to detect different errors which may occur or be present in a series (eg. constant values, calibration errors, data out of range, . . .). The result of a single test is typically categorical (passed / failed). The aim of this framework, however, is, not only reporting this categorical result, but also taking the uncertainty of each test into account to finally assign a probability “that a value is valid” given the results of all tests.

We modify the binary result (passed (1) and zero failed (0) by making use of the uncertainty related to this tests in terms of subtracting it from one if the test is passed or adding it to zero if a test is not passed.

Further details will be provided later.

CHAPTER

TWO

REFERENCE GUIDE

This program will demo a factory class (function) for the automated quality control (auto-qc) of the air quality data.

qc-tests in this program require a set of input parameters that users are able to modify them in the input file. Although the input parameters set to a default value as a suggestion, the operator are in the best position to select an appropriate thresholds for their operations.

class auto_qc_tests.**QCBase**

QCBase is an abstract base class for the qc-tests.

name = 'abstract class for the QC-tests'

__init__()

It creates the variables associated with the QCBase class.

__repr__()

It returns representation of the object.

validate_inputs()

It validates the input time series.

run_test(*df*)

It runs the test.

Parameters **df** (*pd.DataFrame*) – time series

calculate_thresholds()

It calculates the thresholds.

make_conditions()

It makes the failing and passing conditions based on the QC-Test criteria.

make_modifier()

It makes the modifier_prob.

modify_modifier()

It modifies the modifier.

find_index_effecting_neighbor(*idx=None*)

It finds index of neighbors.

Parameters

- **idx** – the index of invalid values.
- **l_idx** – the index of farthest left side neighbor
- **r_idx** – the index of farthest right side neighbor

modify_neighbors()

It modifies the probability of the neighbors influencing from an invalid value.

normalize_modifier()
It normalizes modifier if the prob is larger than 1 or less than 0.

check_modifier()
It checks modifier.

It gives an error if it is larger than 1 or less than 0.

modify_probability()
It modifies the probability.

Returns modified input time series in which 'modify_prob' column: holds the modified probability resulting from a given qc-test 'qc_test_name_prob' column: holds the probability resulting from the qc_test_name

Return type self.df_input(pd.DataFrame)

class auto_qc_tests.**StatCalc**
StatCalc is a kind of abstract base class for statistical calculations of the qc-tests.

name = 'Statistical Calculation'

__init__()
It creates the variables associated with the StatCalc class.

__repr__()
It returns representation of the object.

static calculate_mean_with_cma(*data=None*)
It calculates the mean of the data excluding the center data point.

Parameters **data** (*np.array*) – values

Returns mean of the values(float)

static calculate_stdev_with_cma(*data=None*)
It calculates the standard deviation of the data excluding the center data point.

Parameters **data** (*np.array*) – values

Returns standard deviation of the values(float)

rolling_mean_with_cma()
It calculates the mean of the grouped data in which the center point is masked.

Returns mean of the grouped data

Return type grp_mean(pd.DataFrame)

Note: It will return Nan, if more than half of the values within the window size are missing.

rolling_stdev_with_cma()
It calculates the standard deviation of the grouped data in which the center point is masked.

Returns standard deviation of the grouped data

Return type grp_stdev(pd.DataFrame)

Note: It will return Nan, if more than half of the values within the window size are missing.

rolling_mean()
It calculates the mean of the grouped data.

Returns mean of the grouped data

Return type grp_mean(pd.DataFrame)

Note: It will return Nan, if more than half of the values within the window size are missing.

---

rolling_stdev()

It calculates the standard deviation of the grouped data.

Returns standard deviation of the grouped data

Return type grp_stdev(pd.DataFrame)

Note: It will return Nan, if more than half of the values within the window size are missing.

---

class auto_qc_tests.**RangeTest**(*range_min=None*, *range_max=None*, *range_neighboring_size=None*, *range_neighboring_side=None*, *qc_group_name=None*)

RangeTest finds data points those exceed the prescribed minimum and maximum.

Example:

```
>>> df_in = pd.DataFrame({'value2':[31,32,300,37,42], 'modify_prob':[1,1,1,1,1],
↪'base_prob':[1,1,1,1,1]}, index=pd.date_range(start='2010-1-1',end='2010-1-5',
↪freq='D'))  # create a dataframe containing a value out of range
>>> df_in    # input data-frame
            value2  modify_prob  base_prob
2010-01-01      31            1          1
2010-01-02      32            1          1
2010-01-03     300            1          1
2010-01-04      37            1          1
2010-01-05      42            1          1
>>> obj = RangeTest(range_min=-5, range_max=200, range_neighboring_size=1, range_
↪neighboring_side='left', qc_group_name='g0')    # create an object
>>> df_out = obj.run_test(df = df_in)    # run the qc-test
>>> df_out   # output data-frame
            value2  modify_prob  base_prob  g0_range_test_prob
2010-01-01      31          1.0          1                 1.0
2010-01-02      32          0.5          1                 0.5
2010-01-03     300          0.0          1                 0.0
2010-01-04      37          1.0          1                 1.0
2010-01-05      42          1.0          1                 1.0
```

name = 'range_test'

__init__(*range_min=None*, *range_max=None*, *range_neighboring_size=None*, *range_neighboring_side=None*, *qc_group_name=None*)

It creates the variables associated with the RangeTest class.

Parameters

- **range_min** (*float*) – minimum valid value
- **range_max** (*float*) – maximum valid value
- **range_neighboring_size** (*int*) – the numbers of neighbors influencing from an invalid value
- **range_neighboring_side** (*str*) – the side of the neighbors [‘left’,’right’, ‘both’]

- **qc_group_name** (*str*) – qc groups' name (e.g. g0, g1, g2, etc.)

validate_inputs()
It validates the input time series and input parameters of the RangeTest.

make_conditions()
It makes the failing and passing conditions based on the RangeTest criteria.

Returns boolean of valid data fail_data(pd.DataFrame): boolean of invalid data

Return type pass_data(pd.DataFrame)

run_test(*df*)
It runs the RangeTest.

Parameters **df** (*pd.DataFrame*) – time series of a variable

Returns modified df holds the qc-result

Return type df_output(pd.DataFrame)

__repr__()
It returns representation of the object.

calculate_thresholds()
It calculates the thresholds.

check_modifier()
It checks modifier.

It gives an error if it is larger than 1 or less than 0.

find_index_effecting_neighbor(*idx=None*)
It finds index of neighbors.

Parameters

- **idx** – the index of invalid values.
- **l_idx** – the index of farthest left side neighbor
- **r_idx** – the index of farthest right side neighbor

make_modifier()
It makes the modifier_prob.

modify_modifier()
It modifies the modifier.

modify_neighbors()
It modifies the probability of the neighbors influencing from an invalid value.

modify_probability()
It modifies the probability.

Returns modified input time series in which 'modify_prob' column: holds the modified probability resulting from a given qc-test 'qc_test_name_prob' column: holds the probability resulting from the qc_test_name

Return type self.df_input(pd.DataFrame)

normalize_modifier()
It normalizes modifier if the prob is larger than 1 or less than 0.

class auto_qc_tests.**NegativeValueTest**(*negative_value_first=-5*, *negative_value_second=-2*, *negative_value_neighboring_size=None*, *negative_value_neighboring_side=None*, *qc_group_name=None*)

NegativeValuesTest identifies the negative values.

Example:

```
>>> df_in = pd.DataFrame({'value2':[31,32,-30,37,42], 'modify_prob':[1,1,1,1,1],
↪'base_prob':[1,1,1,1,1]}, index=pd.date_range(start='2010-1-1',end='2010-1-5',
↪freq='D'))  # create a dataframe containing a negative value
>>> df_in    # input data-frame
            value2  modify_prob  base_prob
2010-01-01      31            1          1
2010-01-02      32            1          1
2010-01-03     -30            1          1
2010-01-04      37            1          1
2010-01-05      42            1          1
>>> obj = NegativeValueTest(negative_value_first=-5, negative_value_second=-2,␣
↪negative_value_neighboring_size=1, negative_value_neighboring_side='both', qc_
↪group_name='g1')   # create an object
>>> df_out = obj.run_test(df = df_in)   # run the qc-test
>>> df_out   # output data-frame
            value2  modify_prob  base_prob  g1_negative_value_test_prob
2010-01-01      31          1.0          1                          1.0
2010-01-02      32          0.5          1                          0.5
2010-01-03     -30          0.0          1                          0.0
2010-01-04      37          0.5          1                          0.5
2010-01-05      42          1.0          1                          1.0
```

name = 'negative_value_test'

__init__(*negative_value_first=-5*, *negative_value_second=-2*, *negative_value_neighboring_size=None*, *negative_value_neighboring_side=None*, *qc_group_name=None*)

It creates the variables associate with the NegativeValueTest class.

Parameters

- **negative_value_first** (*float*) – the first negative value
- **negative_value_second** (*float*) – the second negative value
- **negative_value_neighboring_size** (*int*) – the numbers of neighbors influencing from an invalid value
- **negative_value_neighboring_side** (*str*) – the side of the neighbors ['left','right', 'both']
- **qc_group_name** (*str*) – qc groups' name (e.g. g0, g1, g2, etc.)

validate_inputs()

It validates the input time series and input parameters of the NegativeValueTest.

count_negatives()

It counts the negative values.

make_conditions()

It makes the failing and passing conditions based on the NegativeValueTest criteria.

Note: if the number of negative values is less than two: they flag as an outlier (i.e. fails)

---

run_test(*df*)
It runs the NegativeValueTest.

Parameters **df** (*pd.DataFrame*) – time series of a variable

Returns modified df holds the qc-result

Return type df_new(pd.DataFrame)

__repr__()
It returns representation of the object.

calculate_thresholds()
It calculates the thresholds.

check_modifier()
It checks modifier.

It gives an error if it is larger than 1 or less than 0.

find_index_effecting_neighbor(*idx=None*)
It finds index of neighbors.

Parameters

- **idx** – the index of invalid values.
- **l_idx** – the index of farthest left side neighbor
- **r_idx** – the index of farthest right side neighbor

make_modifier()
It makes the modifier_prob.

modify_modifier()
It modifies the modifier.

modify_neighbors()
It modifies the probability of the neighbors influencing from an invalid value.

modify_probability()
It modifies the probability.

Returns modified input time series in which 'modify_prob' column: holds the modified probability resulting from a given qc-test 'qc_test_name_prob' column: holds the probability resulting from the qc_test_name

Return type self.df_input(pd.DataFrame)

normalize_modifier()
It normalizes modifier if the prob is larger than 1 or less than 0.

class auto_qc_tests.**ConstantValueTest**(*constant_value_stuck_points=None*, *constant_value_neighboring_size=None*, *constant_value_neighboring_side=None*, *qc_group_name=None*)
ConstantValueTest identifies where several data stuck at the a singular value. That means repeated occurrence of one value in a time series.

Example:

```
>>> df_in = pd.DataFrame({'value2':[31,30,30,30,42], 'modify_prob':[1,1,1,1,1],
↪'base_prob':[1,1,1,1,1]}, index=pd.date_range(start='2010-1-1',end='2010-1-5',
↪freq='D'))  # create a dataframe containing several constant values
>>> df_in    # input data-frame
            value2  modify_prob  base_prob
2010-01-01      31            1          1
2010-01-02      30            1          1
2010-01-03      30            1          1
2010-01-04      30            1          1
2010-01-05      42            1          1
>>> obj = ConstantValueTest(constant_value_stuck_points=3, constant_value_
↪neighboring_size=1, constant_value_neighboring_side='right', qc_group_name='g1
↪') # create an object
>>> df_out = obj.run_test(df = df_in)    # run the qc-test
>>> df_out   # output data-frame
            value2  modify_prob  base_prob  g1_constant_value_test_prob
2010-01-01      31          1.0          1                           1.0
2010-01-02      30          0.0          1                           0.0
2010-01-03      30          0.0          1                           0.0
2010-01-04      30          0.0          1                           0.0
2010-01-05      42          NaN          1                           NaN
```

name = 'constant_value_test'

__init__(*constant_value_stuck_points=None*, *constant_value_neighboring_size=None*, *constant_value_neighboring_side=None*, *qc_group_name=None*)

It creates the variables associate with the ConstantValueTest class.

Parameters

- **constant_value_stuck_points** (*int*) – number of stuck points
- **constant_value_neighboring_size** (*int*) – the numbers of neighbors influencing from an invalid value
- **constant_value_neighboring_side** (*str*) – the side of the neighbors ['left','right', 'both']
- **qc_group_name** (*str*) – qc groups' name (e.g. g0, g1, g2, etc.)

make_conditions()

It makes the failing and passing conditions based on the ConstantValueTest criteria.

Note: All n (i.e. self.stuck_points) values which stuck at the same value will be flagged.

---

run_test(*df*)

It runs the ConstantValueTest.

Parameters **df** (*pd.DataFrame*) – time series of a variable

Returns modified df holds the qc-result

Return type df_new(pd.DataFrame)

__repr__()

It returns representation of the object.

calculate_thresholds()

It calculates the thresholds.

check_modifier()
It checks modifier.

It gives an error if it is larger than 1 or less than 0.

find_index_effecting_neighbor(*idx=None*)
It finds index of neighbors.

Parameters

- **idx** – the index of invalid values.
- **l_idx** – the index of farthest left side neighbor
- **r_idx** – the index of farthest right side neighbor

make_modifier()
It makes the modifier_prob.

modify_modifier()
It modifies the modifier.

modify_neighbors()
It modifies the probability of the neighbors influencing from an invalid value.

modify_probability()
It modifies the probability.

Returns modified input time series in which 'modify_prob' column: holds the modified probability resulting from a given qc-test 'qc_test_name_prob' column: holds the probability resulting from the qc_test_name

Return type self.df_input(pd.DataFrame)

normalize_modifier()
It normalizes modifier if the prob is larger than 1 or less than 0.

validate_inputs()
It validates the input time series.

class auto_qc_tests.**OutlierZTest**(*outlier_z_grp_size=None*, *outlier_z_wid_stdev=None*, *outlier_z_neighboring_size=None*, *outlier_z_neighboring_side=None*, *qc_group_name=None*)
OutlierZTest identifies the data points those fall outside of the expectation ranges. This test is also called z-test.

Example:

```
>>> df_in = pd.DataFrame({'value2':[31,32,78,37,42], 'modify_prob':[1,1,1,1,1],
↪'base_prob':[1,1,1,1,1]}, index=pd.date_range(start='2010-1-1',end='2010-1-5',
↪freq='D'))  # create a dataframe containing an outlier
>>> df_in    # input data-frame
            value2  modify_prob  base_prob
2010-01-01      31            1          1
2010-01-02      32            1          1
2010-01-03      78            1          1
2010-01-04      37            1          1
2010-01-05      42            1          1
>>> obj = OutlierZTest(outlier_z_grp_size=3, outlier_z_wid_stdev=2, outlier_z_
↪neighboring_size=1, outlier_z_neighboring_side='both', qc_group_name='g2')  #␣
↪create an object
>>> df_out = obj.run_test(df = df_in)    # run the qc-test
>>> df_out    # output data-frame
            value2  modify_prob  base_prob  g2_outlier_z_test_prob
```

(continues on next page)

(continued from previous page)

```
2010-01-01      31          1.0          1                    1.0
2010-01-02      32          0.5          1                    0.5
2010-01-03      78          0.0          1                    0.0
2010-01-04      37          0.5          1                    0.5
2010-01-05      42          1.0          1                    1.0
```

name = 'outlier_z_test'

__init__(*outlier_z_grp_size=None*, *outlier_z_wid_stdev=None*, *outlier_z_neighboring_size=None*, *outlier_z_neighboring_side=None*, *qc_group_name=None*)

It creates the variables associate with the OutlierZTest class.

Parameters

- **outlier_z_grp_size** (*int*) – size of the moving window for grouping
- **outlier_z_wid_stdev** (*float*) – width of the standard deviation (i.e. z-value)
- **outlier_z_neighboring_size** (*int*) – the numbers of neighbors influencing from an invalid value
- **outlier_z_neighboring_side** (*str*) – the side of the neighbors ['left','right', 'both']
- **qc_group_name** (*str*) – qc groups' name (e.g. g0, g1, g2, etc.)

calculate_thresholds()

It calculates the upper and lower boundaries (thresholds) for the OutlierZTest class.

make_conditions()

It makes the failing and passing conditions based on the OutlierZTest criteria.

run_test(*df*)

It runs the OutlierZTest.

Parameters **df** (*pd.DataFrame*) – time series of a variable

Returns modified df holds the qc-result

Return type df_new(pd.DataFrame)

__repr__()

It returns representation of the object.

static calculate_mean_with_cma(*data=None*)

It calculates the mean of the data excluding the center data point.

Parameters **data** (*np.array*) – values

Returns mean of the values(float)

static calculate_stdev_with_cma(*data=None*)

It calculates the standard deviation of the data excluding the center data point.

Parameters **data** (*np.array*) – values

Returns standard deviation of the values(float)

check_modifier()

It checks modifier.

It gives an error if it is larger than 1 or less than 0.

find_index_effecting_neighbor(*idx=None*)

It finds index of neighbors.

Parameters

- **idx** – the index of invalid values.
- **l_idx** – the index of farthest left side neighbor
- **r_idx** – the index of farthest right side neighbor

make_modifier()
It makes the modifier_prob.

modify_modifier()
It modifies the modifier.

modify_neighbors()
It modifies the probability of the neighbors influencing from an invalid value.

modify_probability()
It modifies the probability.

Returns modified input time series in which 'modify_prob' column: holds the modified probability resulting from a given qc-test 'qc_test_name_prob' column: holds the probability resulting from the qc_test_name

Return type self.df_input(pd.DataFrame)

normalize_modifier()
It normalizes modifier if the prob is larger than 1 or less than 0.

rolling_mean()
It calculates the mean of the grouped data.

Returns mean of the grouped data

Return type grp_mean(pd.DataFrame)

Note: It will return Nan, if more than half of the values within the window size are missing.

---

rolling_mean_with_cma()
It calculates the mean of the grouped data in which the center point is masked.

Returns mean of the grouped data

Return type grp_mean(pd.DataFrame)

Note: It will return Nan, if more than half of the values within the window size are missing.

---

rolling_stdev()
It calculates the standard deviation of the grouped data.

Returns standard deviation of the grouped data

Return type grp_stdev(pd.DataFrame)

Note: It will return Nan, if more than half of the values within the window size are missing.

---

rolling_stdev_with_cma()
It calculates the standard deviation of the grouped data in which the center point is masked.

Returns standard deviation of the grouped data

Return type grp_stdev(pd.DataFrame)

Note: It will return Nan, if more than half of the values within the window size are missing.

---

validate_inputs()
It validates the input time series.

class auto_qc_tests.**OutlierQTest**(*outlier_q_grp_size=None*, *outlier_q_critical_value=None*, *outlier_q_neighboring_size=None*, *outlier_q_neighboring_side=None*, *qc_group_name=None*)
OutlierQTest identifies a single outlier. This test is also called DixonTest.

Example:

```
>>> df_in = pd.DataFrame({'value2':[31,32,138,37,42], 'modify_prob':[1,1,1,1,1],
↪'base_prob':[1,1,1,1,1]}, index=pd.date_range(start='2010-1-1',end='2010-1-5',
↪freq='D'))  # create a dataframe containing an outlier
>>> df_in    # input data-frame
            value2  modify_prob  base_prob
2010-01-01      31            1          1
2010-01-02      32            1          1
2010-01-03     138            1          1
2010-01-04      37            1          1
2010-01-05      42            1          1
>>> obj = OutlierQTest(outlier_q_grp_size=3, outlier_q_critical_value=0.45,␣
↪outlier_q_neighboring_size=0, outlier_q_neighboring_side='left', qc_group_name=
↪'g2')  # create an object
>>> df_out = obj.run_test(df = df_in)    # run the qc-test
>>> df_out   # output data-frame
            value2  modify_prob  base_prob  g2_outlier_q_test_prob
2010-01-01      31            1          1                       1
2010-01-02      32            0          1                       0
2010-01-03     138            0          1                       0
2010-01-04      37            0          1                       0
2010-01-05      42            1          1                       1
```

name = 'outlier_q_test'

__init__(*outlier_q_grp_size=None*, *outlier_q_critical_value=None*, *outlier_q_neighboring_size=None*, *outlier_q_neighboring_side=None*, *qc_group_name=None*)
This creates the variables associated with the OutlierQTest class.

Parameters

- **outlier_q_grp_size** (*int*) – size of the moving window for grouping
- **outlier_q_critical_value** (*float*) – the maximum acceptable threshold (i.e. dixon ratio)
- **outlier_q_neighboring_size** (*int*) – the numbers of neighbors influencing from an invalid value
- **outlier_q_neighboring_side** (*str*) – the side of the neighbors [‘left’,’right’, ‘both’]
- **qc_group_name** (*str*) – number of the qc group (e.g. g0, g1, g2, etc)

calculate_q(*data*)
It calculates the Q of the possible outliers, i.e. very large (small) value.

Parameters **data** (*float*) – input values

Returns the q of the possible outliers

Return type q(float)

is_step()
It checks either the largest (smallest) value is a step change or not.

Note: Here, the largest (smallest) value is step if is the first or the last value.

is_spike()
It checks either the largest (smallest) value is a spike change or not.

Note: Here, the largest (smallest) value is spike if its previous value is equal to its next value.

Here, the largest (smallest) value should not be the first or the last value.

make_conditions()
It makes the failing and passing conditions based on the OutlierZTest criteria.

run_test(*df*)
It runs the OutlierQTest.

Parameters **df** (*pd.DataFrame*) – time series of a variable

Returns modified df holds the qc-result

Return type df_new(pd.DataFrame)

__repr__()
It returns representation of the object.

calculate_thresholds()
It calculates the thresholds.

check_modifier()
It checks modifier.

It gives an error if it is larger than 1 or less than 0.

find_index_effecting_neighbor(*idx=None*)
It finds index of neighbors.

Parameters

- **idx** – the index of invalid values.
- **l_idx** – the index of farthest left side neighbor
- **r_idx** – the index of farthest right side neighbor

make_modifier()
It makes the modifier_prob.

modify_modifier()
It modifies the modifier.

modify_neighbors()
It modifies the probability of the neighbors influencing from an invalid value.

modify_probability()
It modifies the probability.

Returns modified input time series in which 'modify_prob' column: holds the modified probability resulting from a given qc-test 'qc_test_name_prob' column: holds the probability resulting from the qc_test_name

Return type self.df_input(pd.DataFrame)

normalize_modifier()
It normalizes modifier if the prob is larger than 1 or less than 0.

validate_inputs()
It validates the input time series.

class auto_qc_tests.**SpikeTest**(*spike_magnify_factor=None*, *spike_grp_size=None*, *spike_neighboring_size=None*, *spike_neighboring_side=None*, *qc_group_name=None*)
SpikeTest identifies spikes (jumps) within the data! That means a single data value deviates significantly from the surrounding data values.

Example:

```
>>> df_in = pd.DataFrame({'value2':[31,32,138,32,42], 'modify_prob':[1,1,1,1,1],
↪'base_prob':[1,1,1,1,1]}, index=pd.date_range(start='2010-1-1',end='2010-1-5',
↪freq='D'))  # create a dataframe containing an outlier
>>> df_in    # input data-frame
            value2  modify_prob  base_prob
2010-01-01      31            1          1
2010-01-02      32            1          1
2010-01-03     138            1          1
2010-01-04      32            1          1
2010-01-05      42            1          1
>>> obj = SpikeTest(spike_magnify_factor=2, spike_grp_size=3, spike_neighboring_
↪size=1, spike_neighboring_side='right', qc_group_name='g2')  # create an object
>>> df_out = obj.run_test(df = df_in)   # run the qc-test
>>> df_out   # output data-frame
            value2  modify_prob  base_prob  g2_spike_test_prob
2010-01-01      31          1.0          1                 1.0
2010-01-02      32          1.0          1                 1.0
2010-01-03     138          0.0          1                 0.0
2010-01-04      32          0.5          1                 0.5
2010-01-05      42          1.0          1                 1.0
```

name = 'spike_test'

__init__(*spike_magnify_factor=None*, *spike_grp_size=None*, *spike_neighboring_size=None*, *spike_neighboring_side=None*, *qc_group_name=None*)
This creates the variables associated with the SpikeTest class.

Parameters

- **spike_magnify_factor** (*int*) – the factor multiplies to the range for the threshold (max_delta)
- **spike_grp_size** (*int*) – size of the moving window for grouping
- **spike_neighboring_size** (*int*) – the numbers of neighbors influencing from an invalid value
- **spike_neighboring_side** (*str*) – the side of the neighbors ['left','right', 'both']
- **qc_group_name** (*str*) – number of the qc group (e.g. g0, g1, g2, etc)

Note: the larger spike_multiplier_factor, the less strict this test would be.

calculate_delta_max(*data*)
This calculates the maximum acceptable delta.

Parameters **data** (*np.array*) – input values

Returns the maximum acceptable delta

Return type __delta_max(float)

calculate_delta_max_with_cma(*data*)
This calculates the maximum acceptable delta with masking the center point (i.e. the possible spike or the outlier value).

Parameters **data** (*np.array*) – input values

Returns the maximum acceptable delta

Return type __delta_max(float)

make_conditions()
It makes the failing and passing conditions based on the SpikeTest criteria.

Note: Here, the points is an spike if its differences from its previous (i.e. df_delta['p']) and its next value (i.e. df_delta['n'] are larger than a maximum acceptable difference (i.e. max_delta). Furthermore, the multiplication of df_delta['p'] and df_delta['n'] must be larger than zero.

Here, two adjacent neighbors were flagged.

run_test(*df*)
It runs the SpikeTest.

Parameters **df** (*pd.DataFrame*) – time series of a variable

Returns modified df holds the qc-result

Return type df_new(pd.DataFrame)

__repr__()
It returns representation of the object.

calculate_thresholds()
It calculates the thresholds.

check_modifier()
It checks modifier.

It gives an error if it is larger than 1 or less than 0.

find_index_effecting_neighbor(*idx=None*)
It finds index of neighbors.

Parameters

- **idx** – the index of invalid values.
- **l_idx** – the index of farthest left side neighbor
- **r_idx** – the index of farthest right side neighbor

make_modifier()
It makes the modifier_prob.

modify_modifier()
It modifies the modifier.

modify_neighbors()
It modifies the probability of the neighbors influencing from an invalid value.

modify_probability()
It modifies the probability.

Returns modified input time series in which 'modify_prob' column: holds the modified probability resulting from a given qc-test 'qc_test_name_prob' column: holds the probability resulting from the qc_test_name

Return type self.df_input(pd.DataFrame)

normalize_modifier()
It normalizes modifier if the prob is larger than 1 or less than 0.

validate_inputs()
It validates the input time series.

class auto_qc_tests.**StepTest**(*step_magnify_factor=None*, *step_grp_size=None*, *step_neighboring_size=None*, *step_neighboring_side=None*, *qc_group_name=None*)
StepTest identifies an step within the data!

Example:

```
>>> df_in = pd.DataFrame({'value2':[32,32,138,138,42], 'modify_prob':[1,1,1,1,1],
↪'base_prob':[1,1,1,1,1]}, index=pd.date_range(start='2010-1-1',end='2010-1-5',
↪freq='D'))  # create a dataframe containing a step
>>> df_in    # input data-frame
            value2  modify_prob  base_prob
2010-01-01      32            1          1
2010-01-02      32            1          1
2010-01-03     138            1          1
2010-01-04     138            1          1
2010-01-05      42            1          1
>>> obj = StepTest(step_magnify_factor=2, step_grp_size=3, step_neighboring_
↪size=1, step_neighboring_side='right', qc_group_name='g2')  # create an object
>>> df_out = obj.run_test(df = df_in)   # run the qc-test
>>> df_out   # output data-frame
            value2  modify_prob  base_prob  g2_step_test_prob
2010-01-01      32          1.0          1                1.0
2010-01-02      32          0.0          1                0.0
2010-01-03     138          0.0          1                0.0
2010-01-04     138          0.5          1                0.5
2010-01-05      42          1.0          1                1.0
```

name = 'step_test'

__init__(*step_magnify_factor=None*, *step_grp_size=None*, *step_neighboring_size=None*, *step_neighboring_side=None*, *qc_group_name=None*)
This creates the variables associated with the StepTest class.

Parameters

- **step_magnify_factor** (*int*) – the factor multiplies to the range for the thresholds
- **step_grp_size** (*int*) – size of the moving window for grouping

- **step_neighboring_size** (*int*) – the numbers of neighbors influencing from an invalid value
- **step_neighboring_side** (*str*) – the side of the neighbors [‘left’,’right’, ‘both’]
- **qc_group_name** (*str*) – number of the qc group (e.g. g0, g1, g2, etc)

Note: the larger step_magnify_factor, the less strict this test would be.

calculate_delta_max(*data*)
This calculates the maximum acceptable delta.

Parameters **data** (*np.array*) – input values

Returns the maximum acceptable delta

Return type __delta_max(float)

make_conditions()
It makes the failing and passing conditions based on the StepTest criteria.

Note: Here, the points is an step if its difference from its next value are larger than a maximum acceptable difference (i.e. df_delta[‘n1’] > max_delta). Furthermore, to be a real stepped point, its difference from the previous value should be zero (i.e. df_delta[‘p’]=0), and also df_delta[‘n1’] = df_delta[‘n2’]. Nevertheless, these criteria might be changed in the future.

Here, three neighbors (2 next and 1 previous) of the stepped point were flagged.

run_test(*df*)
It runs the StepTest.

Parameters **df** (*pd.DataFrame*) – time series of a variable

Returns modified df holds the qc-result

Return type df_new(pd.DataFrame)

__repr__()
It returns representation of the object.

calculate_thresholds()
It calculates the thresholds.

check_modifier()
It checks modifier.

It gives an error if it is larger than 1 or less than 0.

find_index_effecting_neighbor(*idx=None*)
It finds index of neighbors.

Parameters

- **idx** – the index of invalid values.
- **l_idx** – the index of farthest left side neighbor
- **r_idx** – the index of farthest right side neighbor

make_modifier()
It makes the modifier_prob.

modify_modifier()
It modifies the modifier.

modify_neighbors()
It modifies the probability of the neighbors influencing from an invalid value.

modify_probability()
It modifies the probability.

Returns modified input time series in which 'modify_prob' column: holds the modified probability resulting from a given qc-test 'qc_test_name_prob' column: holds the probability resulting from the qc_test_name

Return type self.df_input(pd.DataFrame)

normalize_modifier()
It normalizes modifier if the prob is larger than 1 or less than 0.

validate_inputs()
It validates the input time series.

This program will demo a factory classes for showing the results of the implemented qc tests in the format 'png' and 'JSONFile'!

class auto_qc_outputs.**Output**(*output_format=None*, *output_template=None*, *output_type=None*, *output_file_name=None*, *output_path=None*)

name = 'abstract class for the qc-output'

__init__(*output_format=None*, *output_template=None*, *output_type=None*, *output_file_name=None*, *output_path=None*)
It creates the variables associated with the Plot class.

Parameters

- **output_format** (*str*) – the output file format, i.e. 'png'
- **output_template** – the output file format, i.e. ['all-years', 'each-year']
- **output_type** (*str*) – type of output file, i.e. 'summary'
- **output_file_name** (*str*) – the output file name
- **output_path** (*str*) – the output path

__repr__()
It returns representation of the object.

static df_col_drop(*df=None*, *col_name=None*)
It drops a column from the input data.

Args: df(pd.DataFrame): input data col_name(str): the name of the column to be removed

Returns: df(pd.DataFrame): data without the column['col_name']

save_qc_output()
It saves the qc outputs in the given (user-defined) format.

make_output_filename()
It makes the filename for the output files.

Returns filename (str)

find_output_path()
It finds the path for the output files.

Returns path for the output files

class auto_qc_outputs.**Plot**(*output_format=None*, *output_template=None*, *output_type=None*, *output_file_name=None*, *output_path=None*)

Plot is an abstract base class for the QC-plot.

name = 'abstract class for the qc-plot'

static make_subplots(*n_rows=None*, *n_cols=None*)

It makes subplots.

Parameters

- **n_rows** (*int*) – the number subplot rows
- **n_cols** (*int*) – the number of subplot columns

add_title(*fig=None*)

It adds title.

Parameters **fig** (*object*) – the figure

save_figure(*fig=None*)

It saves a figure.

Parameters **fig** (*object*) – the figure

save_qc_output(*df=None*, *station_metadata=None*)

It saves (plots) the qc-output.

Parameters

- **df** (*pd.DataFrame*) – time series
- **station_metadata** (*dict*) – the station metadata

__init__(*output_format=None*, *output_template=None*, *output_type=None*, *output_file_name=None*, *output_path=None*)

It creates the variables associated with the Plot class.

Parameters

- **output_format** (*str*) – the output file format, i.e. 'png'
- **output_template** – the output file format, i.e. ['all-years', 'each-year']
- **output_type** (*str*) – type of output file, i.e. 'summary'
- **output_file_name** (*str*) – the output file name
- **output_path** (*str*) – the output path

__repr__()

It returns representation of the object.

static df_col_drop(*df=None*, *col_name=None*)

It drops a column from the input data.

Args: df(pd.DataFrame): input data col_name(str): the name of the column to be removed

Returns: df(pd.DataFrame): data without the column['col_name']

find_output_path()

It finds the path for the output files.

Returns path for the output files

make_output_filename()
It makes the filename for the output files.

Returns filename (str)

class auto_qc_outputs.**TimeSeriesScatterPlot**(*df=None*, *pass_prob=None*, *fail_prob=None*)
It plots a scatter time series on the given axes.

name = 'a class for time series scatter plot'

__init__(*df=None*, *pass_prob=None*, *fail_prob=None*)
This creates the variables associated with the TimeSeriesScatterPlot class.

Parameters

- **data** (*pd.DataFrame*) – input timeseries
- **timeseriesscatter_template** (*str*) – the template of the plots where 'each-year' shows the results of each years 'all-years' shows the results of all years
- **timeseriesscatter_fig_format** (*str*) – format of the plot where 'summary' shows valid, suspected and erroneous values 'probability' shows probability of being a good value 'summary-probability' shows both summary and probability
- **timeseriesscatter_pass_prob** (*float*) – probability above it a value is a valid data point
- **timeseriesscatter_fail_prob** (*float*) – probability below it a value is a erroneous data point

make_xticks_label(*ax=None*, *x=None*)
It makes the x ticks labels.

Parameters

- **ax** (*object*) – axis
- **x** (*datetime*) – x-axis values

Returns

make_scatter_plot(*ax=None*, *x=None*, *y=None*, *c=None*, *marker_shape='x'*, *marker_size=3*, *color=None*, *my_cmap=None*, *scatter_label=''*, *colorbar_label=''*)
It makes scatter plot.

Parameters

- **ax** (*object*) – the axes
- **x** (*float or datetime*) – x-axis values
- **y** (*float*) – y-axis values
- **c** (*float*) – filled_marked values
- **marker_shape** (*str*) – shape of marker
- **marker_size** (*int*) – size of marker
- **color** (*str*) – the color of markers
- **my_cmap** (*obj*) – continuous color map
- **scatter_label()** – label of scatter
- **colorbar_label** (*str*) – colorbar label

static add_colorbar(*image*, *label*)
It add a colorbar to the images.

Parameters

- **image** (*object*) – the figure
- **label** (*str*) – label of colorbar

static set_labels(*ax=None*, *xlabel=''*, *ylabel=''*)
It sets labels for the given axes.

Parameters

- **ax** (*object*) – axes
- **xlabel** (*str*) – the x-axis label
- **ylabel** (*str*) – the y-axis label

static make_legend(*ax=None*)
It makes legend.

Parameters **ax** (*object*) – axes

Note: bbox_to_anchor(x, y) places the corner of the legend specified by loc at x, y e.g. right: (1.01, 0.95)

---

static make_colormap1(*my_seq=None*)
It returns a LinearSegmentedColormap.

Parameters **my_seq** (*list*) – floats number and RGB-tuples. The floats should be increasing and in the interval (0,1).

Reference: https://stackoverflow.com/questions/16834861/ create-own-colormap-using-matplotlib-and-plot-color-scale

static make_colormap2()
It returns a LinearSegmentedColormap.

Reference: https://matplotlib.org/examples/pylab_examples/custom_cmap.html

plot_summary(*ax=None*, *colorbar=None*)
It plots a scatter plot for the summary.

Parameters

- **ax** (*object*) – the axes
- **colorbar** (*str*) – the type of the colorbar [‘continuous, discrete’] the default is ‘RdYlBu’

make_plot(*ax=None*, *plt_type=None*)
It makes plot.

Parameters

- **ax** (*object*) – the axes
- **plt_type** (*str*) – the type of the plots

class auto_qc_outputs.**Table**(*df=None*, *pass_prob=None*, *fail_prob=None*)
Table is an class for the QC-table on the given axes (user defined).

name = 'a class for the qc-tables'

__init__(*df=None*, *pass_prob=None*, *fail_prob=None*)
This creates the variables associated with the Table class.

Parameters

- **df** (*pd.DataFrame*) –
- **pass_prob** (*float*) –
- **fail_prob** (*float*) –

describe_data()
It describes the data.

Returns the length and parentage of the values (i.e. valid, suspected, erroneous) test_names(str): the test names

Return type values(float)

make_table(*ax=None*)
It makes a table.

Parameters **ax** (*object*) – the axes for making the table

class auto_qc_outputs.**ScatterPlot**

make_scatter_plot(*ax=None*, *x=None*, *y=None*, *c=None*, *marker_shape='x'*, *marker_size=3*, *color=None*, *my_cmap=None*, *scatter_label=''*, *colorbar_label=''*)
It makes scatter plot.

Parameters

- **ax** (*object*) – the axes
- **x** (*float or datetime*) – x-axis values
- **y** (*float*) – y-axis values
- **c** (*float*) – filled_marked values
- **marker_shape** (*str*) – shape of marker
- **marker_size** (*int*) – size of marker
- **color** (*str*) – the color of markers
- **my_cmap** (*obj*) – continuous color map
- **scatter_label()** – label of scatter
- **colorbar_label** (*str*) – colorbar label

class auto_qc_outputs.**File**(*output_format=None*, *output_template=None*, *output_type=None*, *output_file_name=None*, *output_path=None*)
File is an abstract base class for the QC-output-file.

name = 'abstract class for the qc-output-file'

save_csv()
It saves the data-frame as a csv file.

save_json()
It saves the data-frame as a json file.

save_qc_output(*df=None*, *station_metadata=None*)
It saves the output-data as a file.

Parameters

- **df** (*pd.DataFrame*) – time series
- **station_metadata** (*dict*) – the station metadata

__init__(*output_format=None*, *output_template=None*, *output_type=None*, *output_file_name=None*, *output_path=None*)

It creates the variables associated with the Plot class.

Parameters

- **output_format** (*str*) – the output file format, i.e. 'png'
- **output_template** – the output file format, i.e. ['all-years', 'each-year']
- **output_type** (*str*) – type of output file, i.e. 'summary'
- **output_file_name** (*str*) – the output file name
- **output_path** (*str*) – the output path

__repr__()

It returns representation of the object.

static df_col_drop(*df=None*, *col_name=None*)

It drops a column from the input data.

Args: df(pd.DataFrame): input data col_name(str): the name of the column to be removed

Returns: df(pd.DataFrame): data without the column['col_name']

find_output_path()

It finds the path for the output files.

Returns path for the output files

make_output_filename()

It makes the filename for the output files.

Returns filename (str)

class auto_qc_outputs.**CSV**

CSV is class for the QC-output-file.

name = 'csv output file'

__init__()

It creates the variables associated with the CSV class.

make_csv_file()

It makes a csv file for the qc-output.

class auto_qc_outputs.**JSON**

JSON is class for the QC-output-file.

name = 'json output file'

__init__()

It creates the variables associated with the JSON class.

make_json_file()

It makes a json file for the output.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

C

D

F

I

J

M

N