DLMDWPMP01

Release 1.2.0

Georg Grunsky

Jan 12, 2023

CONTENTS

The **main effort** of this package is to gain basic python programming skills and therefore solve a simple classification task. This package is designed to fulfill the given task and complete the IU Python Module 'DLMDWPMP01'.

The **task** was to:

Evaluate ideal functions for a set of training data (1) and assign values of a test-dataset to those ideal functions (2)

Used criteria for evaluation:

(1) to match training data and ideal functions: minimum SummedSquaredError (SSE)

(2) to match ideal functions and test data: precalculated SSE (1) * SquareRoot(2)

After **installing** the package by typing 'pip install -e .' in your console, the program 'functionfinder' can be called from the console, using the CLI-command **'ff'**.

This documentation was automatically generated by Sphinx, using the docstrings of following modules:

CHAPTER

ONE

FUNCTIONFINDER

Modules

functionfinder.classes	Definition of object classes.
functionfinder.config	User configurable parameters.
functionfinder.datafunctions	Functions used for data handling.
functionfinder.exceptions	Userdefined exceptions.
functionfinder.ffrunner	Main program.
functionfinder.log	Definition of logging parameters.
functionfinder.setuplog	Definition of logging parameters for the installation process.

1.1 functionfinder.classes

Definition of object classes.

This module defines three classes used to store data information. The latter two are subclasses of the first class projectdata.

Classes

idealdata([dataname, ylabel, xlabel, ...])	Subclass of class 'projectdata' to store and handle data operations.
projectdata([dataname, ylabel, xlabel, ...])	Storage defined to handle necessary data operations.
testdata([dataname, ylabel, xlabel, ...])	Subclass of class 'projectdata' to store and handle data operations.

1.1.1 functionfinder.classes.idealdata

class functionfinder.classes.**idealdata**(*dataname='train'*, *ylabel='Y'*, *xlabel='X'*, *plottitle='no title'*, *plotfile='testplot.png'*)

Bases: *projectdata*

Subclass of class 'projectdata' to store and handle data operations.

Contains recurring methods for the ideal functions dataset.

Attributes

As in class 'projectdata' plus additionally: matched : dictionary

Results of matching training data to ideal functions

Methods

As in class 'projectdata' plus additionally: matched_functions(match_result=dict())

Assign dictionary to 'matched' attribute

__init__(*dataname='train'*, *ylabel='Y'*, *xlabel='X'*, *plottitle='no title'*, *plotfile='testplot.png'*)

Construct attributes for this class.

Parameters

dataname
[string] Provide SQLite table name to read data from. The default is "train".

ylabel
[string, optional] Provide y-axis title for plotting. The default is "Y".

xlabel
[string, optional] Provide x-axis title for plotting. The default is "X".

plottitle
[string] Provide plot-title for contained data. The default is "no title".

plotfile
[string] Provide name of PNG to save plot. The default is "testplot.png".

Methods

__init__([dataname, ylabel, xlabel, ...])	Construct attributes for this class.
draw()	Plot data and save to png file.
getdata()	Query SQLite database to read and store data in 'data' attribute.
matched_functions([match_result])	Assign dictionary to 'matched' attribute.

draw()

Plot data and save to png file.

getdata()

Query SQLite database to read and store data in 'data' attribute.

matched_functions(*match_result={}*)

Assign dictionary to 'matched' attribute.

Parameters

match_result
[dictionary, optional] Provide dictionary to store in 'matched' attribute. The default is dict().

1.1.2 functionfinder.classes.projectdata

class functionfinder.classes.**projectdata**(*dataname='train'*, *ylabel='Y'*, *xlabel='X'*, *plottitle='no title'*, *plotfile='testplot.png'*)

Bases: object

Storage defined to handle necessary data operations.

Contains recurring methods for the given types of datasets.

Attributes

_table
[string] Refered table name in SQLite Database

_style
[string] Matplotlib plotting style

ylabel
[string] Label of y-axis for plotting

xlabel
[string] Label of x-axis for plotting

title
[string] Title of plot

fname
[string] Name of png file to save plots

_dbcon
[string] Name and location of SQLite Database

data
[pandas.DataFrame] Data read in from SQLite Database

Methods

__init__(dataname="train", ylabel="Y", xlabel="X", plottitle = "no title",

plotfile="testplot.png")

Constructor method to specify relevant attributes.

getdata()
Query SQLite database to read and store data in 'data' attribute.

draw()
Plot data and save to png file.

__init__(*dataname='train'*, *ylabel='Y'*, *xlabel='X'*, *plottitle='no title'*, *plotfile='testplot.png'*)
Construct attributes for this class.

Parameters

dataname
[string] Provide SQLite table name to read data from. The default is "train".

ylabel
[string, optional] Provide y-axis title for plotting. The default is "Y".

xlabel
[string, optional] Provide x-axis title for plotting. The default is "X".

plottitle
[string] Provide plot-title for contained data. The default is "no title".

plotfile
[string] Provide name of PNG to save plot. The default is "testplot.png".

Methods

__init__([dataname, ylabel, xlabel, ...])	Construct attributes for this class.
draw()	Plot data and save to png file.
getdata()	Query SQLite database to read and store data in 'data' attribute.

draw()

Plot data and save to png file.

getdata()

Query SQLite database to read and store data in 'data' attribute.

1.1.3 functionfinder.classes.testdata

class functionfinder.classes.**testdata**(*dataname='train'*, *ylabel='Y'*, *xlabel='X'*, *plottitle='no title'*, *plotfile='testplot.png'*)

Bases: *projectdata*

Subclass of class 'projectdata' to store and handle data operations.

Contains recurring methods for the test dataset.

Attributes

As in class 'projectdata' plus additionally: off : pandas.DataFrame

Results of matching test data to ideal functions, which exceed the predefined limit.

Methods

As in class 'projectdata' plus additionally: off_limit()

Query SQLite database to read and store entries, which exceeded the predefined limit, to 'off' attribute.

__init__(*dataname='train'*, *ylabel='Y'*, *xlabel='X'*, *plottitle='no title'*, *plotfile='testplot.png'*)

Construct attributes for this class.

Parameters

dataname
[string] Provide SQLite table name to read data from. The default is "train".

ylabel
[string, optional] Provide y-axis title for plotting. The default is "Y".

xlabel
[string, optional] Provide x-axis title for plotting. The default is "X".

plottitle
[string] Provide plot-title for contained data. The default is "no title".

plotfile
[string] Provide name of PNG to save plot. The default is "testplot.png".

Methods

__init__([dataname, ylabel, xlabel, ...])	Construct attributes for this class.
draw()	Plot data and save to png file.
getdata()	Query SQLite database to read and store data in 'data' attribute.
off_limit()	Read data, exceeding predifined deviation limit, from SQLite.

draw()
Plot data and save to png file.

getdata()
Query SQLite database to read and store data in 'data' attribute.

off_limit()
Read data, exceeding predifined deviation limit, from SQLite.

1.2 functionfinder.config

User configurable parameters.

This script contains definitions of configurable elements for the functionfinder package and should be adjusted to your own needs before running the program.

It provides variables to configure
output paths, the name of the database, logging configuration, required files (checked at setup), a dictionary to match datafiles to specified keys, the functiondefinition for the errorcalculation of train and ideal data, the factor to modify the calculated error when evaluating test data.

1.2.1 Methods

error_calculation(trainvalue, idealvalue)
Calculate error rate between to given pandas-series objects of same length.

Functions

error_calculation(trainvalue, idealvalue)	Calculate an error-value between two pandas Series.

1.2.2 functionfinder.config.error_calculation

functionfinder.config.**error_calculation**(*trainvalue*, *idealvalue*)
Calculate an error-value between two pandas Series.

The method is called by datafunctions.min_error method and can modified to test different calculation algorithms.

Parameters

trainvalue
[pandas Series] Specific Y-Data column of a trainingdata DataFrame. (has to have the same length as idealvalue!)

idealvalue
[pandas Series] Specific Y-Data column of a idealdata DataFrame. (has to have the same length as idealvalue!)

Returns

calcerror
[float] Calculated error value of the provided two pandas Series

Notes

By default the provided algorithm is the summed squared error (SSE) of the provided pandas Series.

$$\sum_{i=1}^{n}(TrainingData_i - IdealFunction_i)^2$$

1.3 functionfinder.datafunctions

Functions used for data handling.

This script contains definitions for data related functions of the functionfinder package.

1.3.1 Methods

create_empty_sqlitedb(dbname)
Create empty SQLite Database.

csv2sql_directly(existing_db, csv_toadd, tablename)
Import csv files into existing SQLite database using subprocess.

csv2sql_pandas(existing_db, csv_toadd, tablename)
Import csv files into existing SQLite database using pandas.

min_error(train_set, ideal_df)
Select column of a pandas DataFrame with minimal error to a Series.

calculate_best_ideal(test_value, match_against, index, idealdata, function)
Select ideal function with minimal deviation to given point.

Functions

calculate_best_ideal(test_value, ...[, function])	Select ideal function with minimal deviation to given point.
checktypes(functionname, typedict)	Check if types of data passed to a function upon call meet requirements.
create_empty_sqlitedb(dbname)	Create empty SQLite Database.
csv2sql_directly(existing_db, csv_toadd, ...)	Import csv files into existing SQLite database using subprocess.
csv2sql_pandas(existing_db, csv_toadd, tablename)	Import csv files into existing SQLite database using pandas.
min_error(train_set, ideal_df)	Select column of a pandas DataFrame with minimal error to a Series.

1.3.2 functionfinder.datafunctions.calculate_best_ideal

functionfinder.datafunctions.**calculate_best_ideal**(*test_value*, *match_against*, *index*, *idealdata*, *function=<built-in function min>*)

Select ideal function with minimal deviation to given point.

Parameters

test_value
[list] Current row of test.csv file containing two values (x and y).

match_against
[dictionary] Matches of training data to selected ideal functions including calculated deviation.

index
[float] Read from x-value of test_value parameter. Used to index resulting DataFrame.

idealdata
[pandas.DataFrame] Ideal functions from ideal dataset. Used to find function with closest y-value at indexed position (x-value)

function
[method, optional] Desired method to filter resulting deviation values. The default is min (the observation with the lowest deviation will be returned). When passing 'raw' all rows of the resulting DataFrame will be returned (used to write all DataFrame lines to logfile for evaluating certain values).

Returns

result : pandas.DataFrame Containing the deviation (DeltaY), number of matched ideal function (Idealfunktion) and a boolean whether the deviation exceeded the given limit, filtered by passed 'function' method upon call.

1.3.3 functionfinder.datafunctions.checktypes

functionfinder.datafunctions.**checktypes**(*functionname*, *typedict*)

Check if types of data passed to a function upon call meet requirements.

Parameters

functionname
[string] Name of calling function, needed in case of error-logging.

typedict
[dictionary] Required parameters of functions and according datatypes. Format has to be {object-name as string: (object-name, object-type)}

Returns

Raise Error on failure and quit program.

1.3.4 functionfinder.datafunctions.create_empty_sqlitedb

functionfinder.datafunctions.**create_empty_sqlitedb**(*dbname*)

Create empty SQLite Database.

Name and location are specified by handed parameter and data output folder in config.py. Existing databases in same location of the same name are deleted before creation.

Parameters

dbname
[string] Name of database to create.

1.3.5 functionfinder.datafunctions.csv2sql_directly

functionfinder.datafunctions.**csv2sql_directly**(*existing_db*, *csv_toadd*, *tablename*)

Import csv files into existing SQLite database using subprocess.

Direct import using subprocess enhances importing large csv files.

Parameters

existing_db
[string] Name of existing SQLite database. Location/Directory is handled by parameter set in config.py.

csv_toadd
[string] Location and name of csv-file to import.

tablename
[string] Name of table in SQLite database in which to write the data.

1.3.6 functionfinder.datafunctions.csv2sql_pandas

functionfinder.datafunctions.**csv2sql_pandas**(*existing_db*, *csv_toadd*, *tablename*)

Import csv files into existing SQLite database using pandas.

No direct import. csv files are first read into a pandas dataframe and then fed into an existing SQLite database.

Parameters

existing_db
[string] Name of existing SQLite database. Location/Directory is handled by parameter set in config.py.

csv_toadd
[string] Location and name of csv-file to import.

tablename
[string] Name of table in SQLite database in which to write the data.

1.3.7 functionfinder.datafunctions.min_error

functionfinder.datafunctions.**min_error**(*train_set*, *ideal_df*)

Select column of a pandas DataFrame with minimal error to a Series.

Parameters

train_set
[pandas Series] Column of training dataset, that should be matched against DataFrame of ideal functions.

ideal_df
[pandas DataFrame] DataFrame of ideal functions.

Returns

selected, last_val : tupel selected : string

Column name of matched ideal function for this train_set.

last_val
[float] Deviation between train_set and matched ideal function according to the error_calculation method.

1.4 functionfinder.exceptions

Userdefined exceptions.

This module contains userdefined exceptions.

Exceptions

TypeError	Exception raised during check of datatypes of function parameters.

1.4.1 functionfinder.exceptions.TypeError

exception functionfinder.exceptions.**TypeError**

Exception raised during check of datatypes of function parameters.

1.5 functionfinder.ffrunner

Main program.

This script contains the main part of the programm which orchestrates calculations and can be called, after installation, by the CLI-command ff.

1.5.1 Methods

task()

Run the calculations to solve the given task.

Functions

task()	Run the calculations to solve the given task.

1.5.2 functionfinder.ffrunner.task

functionfinder.ffrunner.**task**()

Run the calculations to solve the given task.

Evaluate ideal functions for a set of training data (1) and assign values of a test-dataset to those ideal functions (2)

Used criteria for evaluation: (1) to match training data and ideal functions: minimum SummedSquaredError (SSE) (2) to match ideal functions and test data: precalculated SSE (1) * SquareRoot(2)

1.6 functionfinder.log

Definition of logging parameters.

This script defines the basic logging configuration for the main program. User configurable variables are read from 'config' module.

Functions

setlogging()	Set predefined parameters for logging.

1.6.1 functionfinder.log.setlogging

functionfinder.log.**setlogging**()

Set predefined parameters for logging.

1.7 functionfinder.setuplog

Definition of logging parameters for the installation process.

This script defines the basic logging configuration for the setup process of this program. User configurable variables are read from 'config' module.

Functions

set_setuplogging()	Set predefined parameters for logging setup process.

1.7.1 functionfinder.setuplog.set_setuplogging

functionfinder.setuplog.**set_setuplogging**()

Set predefined parameters for logging setup process.

CHAPTER

TWO

TESTS

Modules

tests.test_unit	Definition of UnitTest-Sets.

2.1 tests.test_unit

Definition of UnitTest-Sets.

This script contains sets of UnitTests to check the functionality of the functionfinder package and is called during the setup process by setup.py.

2.1.1 Notes

Tests of the calculation functions have not been written, due to the fact that calculation functions are, by the author, meant to be open for modification and therefore cannot be tested with specific values.

2.1.2 TestSets

test_sqlite
: Test data functions with regards to SQLite operations.

test_df
: Test basic data properties such as the structure of provided data files.

2.1.3 Methods

check_struct(structdata)
: Check structure of a passed DataFrame to certain needs. Used in test_df TestSet.

Functions

check_struct(structdata)	Check structure of passed DataFrame to certain needs.

2.1.4 tests.test_unit.check_struct

tests.test_unit.**check_struct**(*structdata*)

Check structure of passed DataFrame to certain needs.

Check if column of name 'x' exists and if the number of columns is at least two.

Parameters

structdata
[pandas.DataFrame] DataFrame to check for above criteria.

Returns

checkresult
[bool] If True, all checks were passed.

Classes

test_df([methodName])	
test_sqlite([methodName])	

2.1.5 tests.test_unit.test_df

class tests.test_unit.**test_df**(*methodName='runTest'*)

Bases: TestCase

__init__(*methodName='runTest'*)

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

Methods

__init__([methodName])	Create an instance of the class that will use the named test method when executed.
addClassCleanup(function, /, *args, **kwargs)	Same as addCleanup, except the cleanup items are called even if setUpClass fails (unlike tearDownClass).
addCleanup(function, /, *args, **kwargs)	Add a function, with arguments, to be called when the test is completed.
addTypeEqualityFunc(typeobj, function)	Add a type specific assertEqual style function to compare a type.
assertAlmostEqual(first, second[, places, ...])	Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is more than the given delta.
assertAlmostEquals(**kwargs)	
assertCountEqual(first, second[, msg])	Asserts that two iterables have the same elements, the same number of times, without regard to order.
assertDictContainsSubset(subset, dictionary)	Checks whether dictionary is a superset of subset.
assertDictEqual(d1, d2[, msg])	
assertEqual(first, second[, msg])	Fail if the two objects are unequal as determined by the '==' operator.
assertEquals(**kwargs)	
assertFalse(expr[, msg])	Check that the expression is false.
assertGreater(a, b[, msg])	Just like self.assertTrue(a > b), but with a nicer default message.
assertGreaterEqual(a, b[, msg])	Just like self.assertTrue(a >= b), but with a nicer default message.
assertIn(member, container[, msg])	Just like self.assertTrue(a in b), but with a nicer default message.
assertIs(expr1, expr2[, msg])	Just like self.assertTrue(a is b), but with a nicer default message.
assertIsInstance(obj, cls[, msg])	Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.
assertIsNone(obj[, msg])	Same as self.assertTrue(obj is None), with a nicer default message.
assertIsNot(expr1, expr2[, msg])	Just like self.assertTrue(a is not b), but with a nicer default message.
assertIsNotNone(obj[, msg])	Included for symmetry with assertIsNone.
assertLess(a, b[, msg])	Just like self.assertTrue(a < b), but with a nicer default message.
assertLessEqual(a, b[, msg])	Just like self.assertTrue(a <= b), but with a nicer default message.
assertListEqual(list1, list2[, msg])	A list-specific equality assertion.
assertLogs([logger, level])	Fail unless a log message of level *level* or higher is emitted on *logger_name* or its children.
assertMultiLineEqual(first, second[, msg])	Assert that two multi-line strings are equal.

continues on next page

Table 1 – continued from previous page

assertNoLogs([logger, level])	Fail unless no log messages of level *level* or higher are emitted on *logger_name* or its children.
assertNotAlmostEqual(first, second[, ...])	Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is less than the given delta.
assertNotAlmostEquals(**kwargs)	
assertNotEqual(first, second[, msg])	Fail if the two objects are equal as determined by the '!=' operator.
assertNotEquals(**kwargs)	
assertNotIn(member, container[, msg])	Just like self.assertTrue(a not in b), but with a nicer default message.
assertNotIsInstance(obj, cls[, msg])	Included for symmetry with assertIsInstance.
assertNotRegex(text, unexpected_regex[, msg])	Fail the test if the text matches the regular expression.
assertNotRegexpMatches(**kwargs)	
assertRaises(expected_exception, *args, **kwargs)	Fail unless an exception of class expected_exception is raised by the callable when invoked with specified positional and keyword arguments.
assertRaisesRegex(expected_exception, ...)	Asserts that the message in a raised exception matches a regex.
assertRaisesRegexp(**kwargs)	
assertRegex(text, expected_regex[, msg])	Fail the test unless the text matches the regular expression.
assertRegexpMatches(**kwargs)	
assertSequenceEqual(seq1, seq2[, msg, seq_type])	An equality assertion for ordered sequences (like lists and tuples).
assertSetEqual(set1, set2[, msg])	A set-specific equality assertion.
assertTrue(expr[, msg])	Check that the expression is true.
assertTupleEqual(tuple1, tuple2[, msg])	A tuple-specific equality assertion.
assertWarns(expected_warning, *args, **kwargs)	Fail unless a warning of class warnClass is triggered by the callable when invoked with specified positional and keyword arguments.
assertWarnsRegex(expected_warning, ...)	Asserts that the message in a triggered warning matches a regexp.
assert_(**kwargs)	
countTestCases()	
debug()	Run the test without collecting errors in a TestResult
defaultTestResult()	
doClassCleanups()	Execute all class cleanup functions.
doCleanups()	Execute all cleanup functions.
enterClassContext(cm)	Same as enterContext, but class-wide.
enterContext(cm)	Enters the supplied context manager.
fail([msg])	Fail immediately, with the given message.

continues on next page

Table 1 – continued from previous page

failIf(**kwargs)	
failIfAlmostEqual(**kwargs)	
failIfEqual(**kwargs)	
failUnless(**kwargs)	
failUnlessAlmostEqual(**kwargs)	
failUnlessEqual(**kwargs)	
failUnlessRaises(**kwargs)	
id()	
run([result])	
setUp()	Execute everytime before running a test of this class.
setUpClass()	Execute before running tests of this class.
shortDescription()	Returns a one-line description of the test, or None if no description has been provided.
skipTest(reason)	Skip this test.
subTest([msg])	Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters.
tearDown()	Execute everytime after running a test of this class.
tearDownClass()	Execute after running all tests of this class.
test_idealstruct()	Test structure of ideal dataset.
test_instancecheck()	Test functionality of datafunctions checktypes
test_teststruct()	Test structure of test dataset.
test_trainideal_length()	Test if train dataset and ideal dataset are of same length.
test_trainstruct()	Test structure of train dataset.

Attributes

longMessage
maxDiff

classmethod addClassCleanup(*function*, */*, **args*, ***kwargs*)

Same as addCleanup, except the cleanup items are called even if setUpClass fails (unlike tearDownClass).

addCleanup(*function*, */*, **args*, ***kwargs*)

Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)

Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Args:

typeobj: The data type to call this function on when both values
are of the same type in assertEqual().

function: The callable taking two arguments and an optional
msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertCountEqual(*first*, *second*, *msg=None*)

Asserts that two iterables have the same elements, the same number of times, without regard to order.

self.assertEqual(Counter(list(first)),
Counter(list(second)))

Example:

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertDictContainsSubset(*subset*, *dictionary*, *msg=None*)

Checks whether dictionary is a superset of subset.

assertEqual(*first*, *second*, *msg=None*)

Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*expr*, *msg=None*)

Check that the expression is false.

assertGreater(*a*, *b*, *msg=None*)

Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)

Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)

Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)

Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)

Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)

Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)

Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)

Included for symmetry with assertIsNone.

assertLess(*a*, *b*, *msg=None*)

Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)

Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)

A list-specific equality assertion.

Args:

list1: The first list to compare. list2: The second list to compare. msg: Optional message to use on failure instead of a list of

differences.

assertLogs(*logger=None*, *level=None*)

Fail unless a log message of level *level* or higher is emitted on *logger_name* or its children. If omitted, *level* defaults to INFO and *logger* defaults to the root logger.

This method must be used as a context manager, and will yield a recording object with two attributes: *output* and *records*. At the end of the context manager, the *output* attribute will be a list of the matching formatted log messages and the *records* attribute will be a list of the corresponding LogRecord objects.

Example:

```
with self.assertLogs('foo', level='INFO') as cm:
    logging.getLogger('foo').info('first message')
    logging.getLogger('foo.bar').error('second message')
self.assertEqual(cm.output, ['INFO:foo:first message',
                             'ERROR:foo.bar:second message'])
```

assertMultiLineEqual(*first*, *second*, *msg=None*)

Assert that two multi-line strings are equal.

assertNoLogs(*logger=None*, *level=None*)

Fail unless no log messages of level *level* or higher are emitted on *logger_name* or its children.

This method must be used as a context manager.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)

Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)

Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)

Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)

Included for symmetry with assertIsInstance.

assertNotRegex(*text*, *unexpected_regex*, *msg=None*)

Fail the test if the text matches the regular expression.

assertRaises(*expected_exception*, **args*, ***kwargs*)

Fail unless an exception of class expected_exception is raised by the callable when invoked with specified positional and keyword arguments. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with the callable and arguments omitted, will return a context object used like this:

```
with self.assertRaises(SomeException):
    do_something()
```

An optional keyword argument 'msg' can be provided when assertRaises is used as a context object.

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

```
with self.assertRaises(SomeException) as cm:
    do_something()
the_exception = cm.exception
self.assertEqual(the_exception.error_code, 3)
```

assertRaisesRegex(*expected_exception*, *expected_regex*, **args*, ***kwargs*)

Asserts that the message in a raised exception matches a regex.

Args:

expected_exception: Exception class expected to be raised. expected_regex: Regex (re.Pattern object or string) expected

to be found in error message.

args: Function to be called and extra positional args. kwargs: Extra kwargs. msg: Optional message used in case of failure. Can only be used

when assertRaisesRegex is used as a context manager.

assertRegex(*text*, *expected_regex*, *msg=None*)

Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)

An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Args:

seq1: The first sequence to compare. seq2: The second sequence to compare. seq_type: The expected datatype of the sequences, or None if no

datatype should be enforced.

msg: Optional message to use on failure instead of a list of
differences.

assertSetEqual(*set1*, *set2*, *msg=None*)

A set-specific equality assertion.

Args:
set1: The first set to compare. set2: The second set to compare. msg: Optional message to use on failure instead of a list of

differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)

Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)

A tuple-specific equality assertion.

Args:
tuple1: The first tuple to compare. tuple2: The second tuple to compare. msg: Optional message to use on failure instead of a list of

differences.

assertWarns(*expected_warning*, **args*, ***kwargs*)

Fail unless a warning of class warnClass is triggered by the callable when invoked with specified positional and keyword arguments. If a different type of warning is triggered, it will not be handled: depending on the other warning filtering rules in effect, it might be silenced, printed out, or raised as an exception.

If called with the callable and arguments omitted, will return a context object used like this:

```
with self.assertWarns(SomeWarning):
    do_something()
```

An optional keyword argument 'msg' can be provided when assertWarns is used as a context object.

The context manager keeps a reference to the first matching warning as the 'warning' attribute; similarly, the 'filename' and 'lineno' attributes give you information about the line of Python code from which the warning was triggered. This allows you to inspect the warning after the assertion:

```
with self.assertWarns(SomeWarning) as cm:
    do_something()
the_warning = cm.warning
self.assertEqual(the_warning.some_attribute, 147)
```

assertWarnsRegex(*expected_warning*, *expected_regex*, **args*, ***kwargs*)

Asserts that the message in a triggered warning matches a regexp. Basic functioning is similar to assertWarns() with the addition that only warnings whose messages also match the regular expression are considered successful matches.

Args:
expected_warning: Warning class expected to be triggered. expected_regex: Regex (re.Pattern object or string) expected

to be found in error message.

args: Function to be called and extra positional args. kwargs: Extra kwargs. msg: Optional message used in case of failure. Can only be used

when assertWarnsRegex is used as a context manager.

debug()

Run the test without collecting errors in a TestResult

classmethod doClassCleanups()

Execute all class cleanup functions. Normally called for you after tearDownClass.

doCleanups()

Execute all cleanup functions. Normally called for you after tearDown.

classmethod enterClassContext(*cm*)

Same as enterContext, but class-wide.

enterContext(*cm*)

Enters the supplied context manager.

If successful, also adds its __exit__ method as a cleanup function and returns the result of the __enter__ method.

fail(*msg=None*)

Fail immediately, with the given message.

failureException

alias of `AssertionError`

setUp()

Execute everytime before running a test of this class.

classmethod setUpClass()

Execute before running tests of this class.

shortDescription()

Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)

Skip this test.

subTest(*msg=<object object>*, ***params*)

Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters. A failure in the subtest marks the test case as failed but resumes execution at the end of the enclosed block, allowing further test code to be executed.

tearDown()

Execute everytime after running a test of this class.

classmethod tearDownClass()

Execute after running all tests of this class.

test_idealstruct()

Test structure of ideal dataset.

test_instancecheck()

Test functionality of datafunctions checktypes

test_teststruct()

Test structure of test dataset.

test_trainideal_length()

Test if train dataset and ideal dataset are of same length.

Same length is needed to execute error calculation by conducting a math operation to two pandas.Series objects.

test_trainstruct()

Test structure of train dataset.

2.1.6 tests.test_unit.test_sqlite

class tests.test_unit.**test_sqlite**(*methodName='runTest'*)

Bases: TestCase

__init__(*methodName='runTest'*)

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

Methods

__init__([methodName])	Create an instance of the class that will use the named test method when executed.
addClassCleanup(function, /, *args, **kwargs)	Same as addCleanup, except the cleanup items are called even if setUpClass fails (unlike tearDownClass).
addCleanup(function, /, *args, **kwargs)	Add a function, with arguments, to be called when the test is completed.
addTypeEqualityFunc(typeobj, function)	Add a type specific assertEqual style function to compare a type.
assertAlmostEqual(first, second[, places, ...])	Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is more than the given delta.
assertAlmostEquals(**kwargs)	
assertCountEqual(first, second[, msg])	Asserts that two iterables have the same elements, the same number of times, without regard to order.
assertDictContainsSubset(subset, dictionary)	Checks whether dictionary is a superset of subset.
assertDictEqual(d1, d2[, msg])	
assertEqual(first, second[, msg])	Fail if the two objects are unequal as determined by the '==' operator.
assertEquals(**kwargs)	
assertFalse(expr[, msg])	Check that the expression is false.
assertGreater(a, b[, msg])	Just like self.assertTrue(a > b), but with a nicer default message.

continues on next page

Table 2 – continued from previous page

assertGreaterEqual(a, b[, msg])	Just like self.assertTrue(a >= b), but with a nicer default message.
assertIn(member, container[, msg])	Just like self.assertTrue(a in b), but with a nicer default message.
assertIs(expr1, expr2[, msg])	Just like self.assertTrue(a is b), but with a nicer default message.
assertIsInstance(obj, cls[, msg])	Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.
assertIsNone(obj[, msg])	Same as self.assertTrue(obj is None), with a nicer default message.
assertIsNot(expr1, expr2[, msg])	Just like self.assertTrue(a is not b), but with a nicer default message.
assertIsNotNone(obj[, msg])	Included for symmetry with assertIsNone.
assertLess(a, b[, msg])	Just like self.assertTrue(a < b), but with a nicer default message.
assertLessEqual(a, b[, msg])	Just like self.assertTrue(a <= b), but with a nicer default message.
assertListEqual(list1, list2[, msg])	A list-specific equality assertion.
assertLogs([logger, level])	Fail unless a log message of level *level* or higher is emitted on *logger_name* or its children.
assertMultiLineEqual(first, second[, msg])	Assert that two multi-line strings are equal.
assertNoLogs([logger, level])	Fail unless no log messages of level *level* or higher are emitted on *logger_name* or its children.
assertNotAlmostEqual(first, second[, ...])	Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is less than the given delta.
assertNotAlmostEquals(**kwargs)	
assertNotEqual(first, second[, msg])	Fail if the two objects are equal as determined by the '!=' operator.
assertNotEquals(**kwargs)	
assertNotIn(member, container[, msg])	Just like self.assertTrue(a not in b), but with a nicer default message.
assertNotIsInstance(obj, cls[, msg])	Included for symmetry with assertIsInstance.
assertNotRegex(text, unexpected_regex[, msg])	Fail the test if the text matches the regular expression.
assertNotRegexpMatches(**kwargs)	
assertRaises(expected_exception, *args, **kwargs)	Fail unless an exception of class expected_exception is raised by the callable when invoked with specified positional and keyword arguments.
assertRaisesRegex(expected_exception, ...)	Asserts that the message in a raised exception matches a regex.
assertRaisesRegexp(**kwargs)	
assertRegex(text, expected_regex[, msg])	Fail the test unless the text matches the regular expression.
assertRegexpMatches(**kwargs)	

continues on next page

Table 2 – continued from previous page

assertSequenceEqual(seq1, seq2[, msg, seq_type])	An equality assertion for ordered sequences (like lists and tuples).
assertSetEqual(set1, set2[, msg])	A set-specific equality assertion.
assertTrue(expr[, msg])	Check that the expression is true.
assertTupleEqual(tuple1, tuple2[, msg])	A tuple-specific equality assertion.
assertWarns(expected_warning, *args, **kwargs)	Fail unless a warning of class warnClass is triggered by the callable when invoked with specified positional and keyword arguments.
assertWarnsRegex(expected_warning, ...)	Asserts that the message in a triggered warning matches a regexp.
assert_(**kwargs)	
countTestCases()	
debug()	Run the test without collecting errors in a TestResult
defaultTestResult()	
doClassCleanups()	Execute all class cleanup functions.
doCleanups()	Execute all cleanup functions.
enterClassContext(cm)	Same as enterContext, but class-wide.
enterContext(cm)	Enters the supplied context manager.
fail([msg])	Fail immediately, with the given message.
failIf(**kwargs)	
failIfAlmostEqual(**kwargs)	
failIfEqual(**kwargs)	
failUnless(**kwargs)	
failUnlessAlmostEqual(**kwargs)	
failUnlessEqual(**kwargs)	
failUnlessRaises(**kwargs)	
id()	
run([result])	
setUp()	Execute everytime before running a test of this class.
setUpClass()	Execute before running tests of this class.
shortDescription()	Returns a one-line description of the test, or None if no description has been provided.
skipTest(reason)	Skip this test.
subTest([msg])	Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters.
tearDown()	Execute everytime after running a test of this class.
tearDownClass()	Execute after running all tests of this class.
test_creation()	Test creation of empty sqlite database.
test_table_from_csv()	Test import of csv files to SQLite using subprocess.

continues on next page

Table 2 – continued from previous page

test_table_from_pandas()	Test import of csv files to SQLite using pandas.

Attributes

longMessage
maxDiff

classmethod addClassCleanup(*function*, /, **args*, ***kwargs*)

Same as addCleanup, except the cleanup items are called even if setUpClass fails (unlike tearDownClass).

addCleanup(*function*, /, **args*, ***kwargs*)

Add a function, with arguments, to be called when the test is completed. Functions added are called on a LIFO basis and are called after tearDown on test failure or success.

Cleanup items are called even if setUp fails (unlike tearDown).

addTypeEqualityFunc(*typeobj*, *function*)

Add a type specific assertEqual style function to compare a type.

This method is for use by TestCase subclasses that need to register their own type equality functions to provide nicer error messages.

Args:

typeobj: The data type to call this function on when both values
are of the same type in assertEqual().

function: The callable taking two arguments and an optional
msg= argument that raises self.failureException with a useful error message when the two arguments are not equal.

assertAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)

Fail if the two objects are unequal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is more than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

If the two objects compare equal then they will automatically compare almost equal.

assertCountEqual(*first*, *second*, *msg=None*)

Asserts that two iterables have the same elements, the same number of times, without regard to order.

self.assertEqual(Counter(list(first)),
Counter(list(second)))

Example:

- [0, 1, 1] and [1, 0, 1] compare equal.
- [0, 0, 1] and [0, 1] compare unequal.

assertDictContainsSubset(*subset*, *dictionary*, *msg=None*)

Checks whether dictionary is a superset of subset.

assertEqual(*first*, *second*, *msg=None*)

Fail if the two objects are unequal as determined by the '==' operator.

assertFalse(*expr*, *msg=None*)

Check that the expression is false.

assertGreater(*a*, *b*, *msg=None*)

Just like self.assertTrue(a > b), but with a nicer default message.

assertGreaterEqual(*a*, *b*, *msg=None*)

Just like self.assertTrue(a >= b), but with a nicer default message.

assertIn(*member*, *container*, *msg=None*)

Just like self.assertTrue(a in b), but with a nicer default message.

assertIs(*expr1*, *expr2*, *msg=None*)

Just like self.assertTrue(a is b), but with a nicer default message.

assertIsInstance(*obj*, *cls*, *msg=None*)

Same as self.assertTrue(isinstance(obj, cls)), with a nicer default message.

assertIsNone(*obj*, *msg=None*)

Same as self.assertTrue(obj is None), with a nicer default message.

assertIsNot(*expr1*, *expr2*, *msg=None*)

Just like self.assertTrue(a is not b), but with a nicer default message.

assertIsNotNone(*obj*, *msg=None*)

Included for symmetry with assertIsNone.

assertLess(*a*, *b*, *msg=None*)

Just like self.assertTrue(a < b), but with a nicer default message.

assertLessEqual(*a*, *b*, *msg=None*)

Just like self.assertTrue(a <= b), but with a nicer default message.

assertListEqual(*list1*, *list2*, *msg=None*)

A list-specific equality assertion.

Args:

list1: The first list to compare. list2: The second list to compare. msg: Optional message to use on failure instead of a list of

differences.

assertLogs(*logger=None*, *level=None*)

Fail unless a log message of level *level* or higher is emitted on *logger_name* or its children. If omitted, *level* defaults to INFO and *logger* defaults to the root logger.

This method must be used as a context manager, and will yield a recording object with two attributes: *output* and *records*. At the end of the context manager, the *output* attribute will be a list of the matching formatted log messages and the *records* attribute will be a list of the corresponding LogRecord objects.

Example:

```
with self.assertLogs('foo', level='INFO') as cm:
    logging.getLogger('foo').info('first message')
    logging.getLogger('foo.bar').error('second message')
self.assertEqual(cm.output, ['INFO:foo:first message',
                             'ERROR:foo.bar:second message'])
```

assertMultiLineEqual(*first*, *second*, *msg=None*)

Assert that two multi-line strings are equal.

assertNoLogs(*logger=None*, *level=None*)

Fail unless no log messages of level *level* or higher are emitted on *logger_name* or its children.

This method must be used as a context manager.

assertNotAlmostEqual(*first*, *second*, *places=None*, *msg=None*, *delta=None*)

Fail if the two objects are equal as determined by their difference rounded to the given number of decimal places (default 7) and comparing to zero, or by comparing that the difference between the two objects is less than the given delta.

Note that decimal places (from zero) are usually not the same as significant digits (measured from the most significant digit).

Objects that are equal automatically fail.

assertNotEqual(*first*, *second*, *msg=None*)

Fail if the two objects are equal as determined by the '!=' operator.

assertNotIn(*member*, *container*, *msg=None*)

Just like self.assertTrue(a not in b), but with a nicer default message.

assertNotIsInstance(*obj*, *cls*, *msg=None*)

Included for symmetry with assertIsInstance.

assertNotRegex(*text*, *unexpected_regex*, *msg=None*)

Fail the test if the text matches the regular expression.

assertRaises(*expected_exception*, **args*, ***kwargs*)

Fail unless an exception of class expected_exception is raised by the callable when invoked with specified positional and keyword arguments. If a different type of exception is raised, it will not be caught, and the test case will be deemed to have suffered an error, exactly as for an unexpected exception.

If called with the callable and arguments omitted, will return a context object used like this:

```
with self.assertRaises(SomeException):
    do_something()
```

An optional keyword argument 'msg' can be provided when assertRaises is used as a context object.

The context manager keeps a reference to the exception as the 'exception' attribute. This allows you to inspect the exception after the assertion:

```
with self.assertRaises(SomeException) as cm:
    do_something()
the_exception = cm.exception
self.assertEqual(the_exception.error_code, 3)
```

assertRaisesRegex(*expected_exception*, *expected_regex*, **args*, ***kwargs*)

Asserts that the message in a raised exception matches a regex.

Args:

expected_exception: Exception class expected to be raised. expected_regex: Regex (re.Pattern object or string) expected

to be found in error message.

args: Function to be called and extra positional args. kwargs: Extra kwargs. msg: Optional message used in case of failure. Can only be used

when assertRaisesRegex is used as a context manager.

assertRegex(*text*, *expected_regex*, *msg=None*)

Fail the test unless the text matches the regular expression.

assertSequenceEqual(*seq1*, *seq2*, *msg=None*, *seq_type=None*)

An equality assertion for ordered sequences (like lists and tuples).

For the purposes of this function, a valid ordered sequence type is one which can be indexed, has a length, and has an equality operator.

Args:

seq1: The first sequence to compare. seq2: The second sequence to compare. seq_type: The expected datatype of the sequences, or None if no

datatype should be enforced.

msg: Optional message to use on failure instead of a list of
differences.

assertSetEqual(*set1*, *set2*, *msg=None*)

A set-specific equality assertion.

Args:

set1: The first set to compare. set2: The second set to compare. msg: Optional message to use on failure instead of a list of

differences.

assertSetEqual uses ducktyping to support different types of sets, and is optimized for sets specifically (parameters must support a difference method).

assertTrue(*expr*, *msg=None*)

Check that the expression is true.

assertTupleEqual(*tuple1*, *tuple2*, *msg=None*)

A tuple-specific equality assertion.

Args:

tuple1: The first tuple to compare. tuple2: The second tuple to compare. msg: Optional message to use on failure instead of a list of

differences.

assertWarns(*expected_warning*, **args*, ***kwargs*)

Fail unless a warning of class warnClass is triggered by the callable when invoked with specified positional and keyword arguments. If a different type of warning is triggered, it will not be handled: depending on the other warning filtering rules in effect, it might be silenced, printed out, or raised as an exception.

If called with the callable and arguments omitted, will return a context object used like this:

```
with self.assertWarns(SomeWarning):
    do_something()
```

An optional keyword argument 'msg' can be provided when assertWarns is used as a context object.

The context manager keeps a reference to the first matching warning as the 'warning' attribute; similarly, the 'filename' and 'lineno' attributes give you information about the line of Python code from which the warning was triggered. This allows you to inspect the warning after the assertion:

```
with self.assertWarns(SomeWarning) as cm:
    do_something()
the_warning = cm.warning
self.assertEqual(the_warning.some_attribute, 147)
```

assertWarnsRegex(*expected_warning*, *expected_regex*, **args*, ***kwargs*)

Asserts that the message in a triggered warning matches a regexp. Basic functioning is similar to assertWarns() with the addition that only warnings whose messages also match the regular expression are considered successful matches.

Args:

expected_warning: Warning class expected to be triggered. expected_regex: Regex (re.Pattern object or string) expected

to be found in error message.

args: Function to be called and extra positional args. kwargs: Extra kwargs. msg: Optional message used in case of failure. Can only be used

when assertWarnsRegex is used as a context manager.

debug()

Run the test without collecting errors in a TestResult

classmethod doClassCleanups()

Execute all class cleanup functions. Normally called for you after tearDownClass.

doCleanups()

Execute all cleanup functions. Normally called for you after tearDown.

classmethod enterClassContext(*cm*)

Same as enterContext, but class-wide.

enterContext(*cm*)

Enters the supplied context manager.

If successful, also adds its __exit__ method as a cleanup function and returns the result of the __enter__ method.

fail(*msg=None*)

Fail immediately, with the given message.

failureException

alias of `AssertionError`

setUp()

Execute everytime before running a test of this class.

classmethod setUpClass()

Execute before running tests of this class.

shortDescription()

Returns a one-line description of the test, or None if no description has been provided.

The default implementation of this method returns the first line of the specified test method's docstring.

skipTest(*reason*)

Skip this test.

subTest(*msg=<object object>*, ***params*)

Return a context manager that will return the enclosed block of code in a subtest identified by the optional message and keyword parameters. A failure in the subtest marks the test case as failed but resumes execution at the end of the enclosed block, allowing further test code to be executed.

tearDown()

Execute everytime after running a test of this class.

classmethod tearDownClass()

Execute after running all tests of this class.

Try deleting created files during testing process. Pass if deletion is not possible. The existance of test-files can be accepted an has no negative effect on the result of the main program.

test_creation()

Test creation of empty sqlite database.

test_table_from_csv()

Test import of csv files to SQLite using subprocess.

Test csv2sql_directly function inside the module datafunctions.py. This Test is expected to fail, therefore a @expectedFailure decorator was placed. In case of success, a log-file entry gets printed during the setup process. Success would suggest, that the system is able to handle extremely large csv files, due to the possibility to directly write those files to the SQLite database.

test_table_from_pandas()

Test import of csv files to SQLite using pandas.

Test csv2sql_pandas function of module datafunction.py

PYTHON MODULE INDEX

f

t

INDEX

Symbols

A

C

D

E

F

G

I

M

O

P

S

T