Python 3.8.9 [default, Clang 13.1.6 (clang-1316.0.21.2.5)]
macOS-12.4

stat_functions

/Users/pranavdhruv/PycharmProje

Modules

numpy pandas scipy scipy.stats

Classes

builtins.object

BioStatistics
DFmaker

class **BioStatistics**(builtins.object)

`BioStatistics(array)`

This class has all the statistical functions that are required. Any future to add more features work should be added on top of this and then be manipulated inside the DFmaker class

Methods defined here:

RI(self)
Calculates reference interval in accordance to CLSI guidelines

__init__(self, array)
Initialize self. See help(type(self)) for accurate signature.

co_eff_kurtosis(self)
Calculates the coefficient of kurtosis

co_eff_skewness(self)
Calculates the coefficient of skewness

cube_root_transform(self)
Cube Root Transformation of the data

exp_transform(self)

Exponential transformation of the data

high_lim(self)
Returns lower limit of the Reference Range

iqr(self)
Calculates the inter-quartile range of an array/vector

log_transform(self)
Log transformation of the data

low_lim(self)
Returns lower limit of the Reference Range

max_val(self)
Calculates the max of an array/vector

mean(self)
Calculates the mean of an array/vector

median(self)
Calculates the median of an array/vector

min_val(self)
Calculates the min of an array/vector

normalize(self)
Normalizes the array

para_outlier(self)
Removes outliers accordance to CLSI guidelines

sd(self)
Calculates the standard deviation of an array/vector

shapiro_wilk_test(self)
Calculates the coefficient of kurtosis

square_root_transform(self)
Square Root Transformation of the data
remember to square data after processing

var(self)
Calculates the variance of an array/vector

z_score(self)
Calculates the z score

Data descriptors defined here:

__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

class **DFmaker**(builtins.object)

```
DFmaker(df_in, age=None, sex=None, biofluid=None)


Makes the DataFrame for processing by the Biostatists
@:param df_in is the input dataframe
@:param age is the age subcategory "Adult" or "Child"
@:param sex is the sex subcategory "Male" or "Female"
@:param biofluid is the type of fluid being analysed "serum" or "plasma"
```

Methods defined here:

__init__(self, df_in, age=None, sex=None, biofluid=None)
Initialize self. See help(type(self)) for accurate signature.

df_out(self)
This is where the processing for the Reference Range happens
Remember:Adding MY biomarkers to update the list add the Biomarker and ID in the MY Biomarkers file

---

Data descriptors defined here:

__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

Python 3.8.9 [default, Clang 13.1.6 (clang-1316.0.21.2.5)]
macOS-12.4

test

/Users/pranavdhruv/Pyc

Modules

numpy
pandas
scipy
scipy.stats
unittest

Classes

unittest.case.TestCase(builtins.object)

TestRI

class **TestRI**(unittest.case.TestCase)

```
TestRI(methodName='runTest')

A class whose instances are single test cases.

By default, the test code itself should be placed in a method named
'runTest'.

If the fixture may be used for many test cases, create as
many test methods as are needed. When instantiating such a TestCase
subclass, specify in the constructor arguments the name of the test method
that the instance is to execute.

Test authors should subclass TestCase for their own tests. Construction
and deconstruction of the test's environment ('fixture') can be
implemented by overriding the 'setUp' and 'tearDown' methods respectively.

If it is necessary to override the __init__ method, the base class
__init__ method must always be called. It is important that subclasses
should not change the signature of their __init__ method, since instances
of the classes are instantiated automatically by parts of the framework
in order to be run.
```

```
When subclassing TestCase, you can set these attributes:
* failureException: determines which exception will be raised when
    the instance's assertion methods fail; test methods raising this
    exception will be deemed to have 'failed' rather than 'errored'.
* longMessage: determines whether long messages (including repr of
    objects used in assert methods) will be printed on failure in *addition*
    to any explicit message passed.
* maxDiff: sets the maximum length of a diff in failure messages
    by assert methods using difflib. It is looked up as an instance
    attribute so can be configured by individual tests if required.
```

Method resolution order:

TestRI
unittest.case.TestCase
builtins.object

Methods defined here:

test_RI_al_combo(self)

```
    Testing the functions using the CLSI documentation
    URL:https://docs.ufpr.br/~taconeli/CE06219/Artigo_FR3.pdf
    Testing for Alanine (AlaAT) can refer pg 20 in the URL
```

test_RI_al_men(self)

test_RI_al_women(self)

test_RI_cal_combo(self)

```
    Testing the functions using the CLSI documentation
    URL:https://docs.ufpr.br/~taconeli/CE06219/Artigo_FR3.pdf
    Testing for Calcium can refer pg 20 in the URL
```

test_RI_cal_men(self)

test_RI_cal_women(self)

test_co_eff_skewness_kurt(self)

test_cub_root_trans(self)

test_exp_trans(self)

test_f_score(self)

test_lg_trans(self)

test_mean(self)

test_median(self)

test_min_max_val(self)

test_norm(self)

test_sd(self)

test_shap_wilk(self)

test_sq_root_trans(self)

test_var(self)

Methods inherited from unittest.case.TestCase:

__call__(self, *args, **kwds)
    Call self as a function.

__eq__(self, other)
    Return self==value.

__hash__(self)
    Return hash(self).

__init__(self, 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.

__repr__(self)
    Return repr(self).

__str__(self)
    Return str(self).

addCleanup(self, 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(self, 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(self, 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.
```

assertAlmostEquals = deprecated_func(*args, **kwargs)

assertCountEqual(self, 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(self, subset, dictionary, msg=None)

```
    Checks whether dictionary is a superset of subset.
```

assertDictEqual(self, d1, d2, msg=None)

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

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

assertEquals = deprecated_func(*args, **kwargs)

assertFalse(self, expr, msg=None)

```
    Check that the expression is false.
```

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

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

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

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

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

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

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

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

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

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

assertIsNone(self, obj, msg=None)

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

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

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

assertIsNotNone(self, obj, msg=None)

```
    Included for symmetry with assertIsNone.
```

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

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

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

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

assertListEqual(self, 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(self, 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(self, first, second, msg=None)

```
    Assert that two multi-line strings are equal.
```

assertNotAlmostEqual(self, 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.
```

assertNotAlmostEquals = deprecated_func(*args, **kwargs)

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

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

assertNotEquals = deprecated_func(*args, **kwargs)

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

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

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

```
    Included for symmetry with assertIsInstance.
```

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

```
    Fail the test if the text matches the regular expression.
```

assertNotRegexpMatches = deprecated_func(*args, **kwargs)

assertRaises(self, 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(self, 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.
```

assertRaisesRegexp = deprecated_func(*args, **kwargs)

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

```
    Fail the test unless the text matches the regular expression.
```

assertRegexpMatches = deprecated_func(*args, **kwargs)

assertSequenceEqual(self, 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(self, 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(self, expr, msg=None)

```
    Check that the expression is true.
```

assertTupleEqual(self, 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(self, 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(self, 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.
```

assert_ = deprecated_func(*args, **kwargs)

countTestCases(self)

debug(self)

```
    Run the test without collecting errors in a TestResult
```

defaultTestResult(self)

doCleanups(self)

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

fail(self, msg=None)

```
    Fail immediately, with the given message.
```

failIf = deprecated_func(*args, **kwargs)

failIfAlmostEqual = deprecated_func(*args, **kwargs)

failIfEqual = deprecated_func(*args, **kwargs)

failUnless = deprecated_func(*args, **kwargs)

failUnlessAlmostEqual = deprecated_func(*args, **kwargs)

failUnlessEqual = deprecated_func(*args, **kwargs)

failUnlessRaises = deprecated_func(*args, **kwargs)

id(self)

run(self, result=None)

setUp(self)
```
Hook method for setting up the test fixture before exercising it.
```

shortDescription(self)
```
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(self, reason)
```
Skip this test.
```

subTest(self, msg=<object object at 0x1286b6770>, **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(self)
```
Hook method for deconstructing the test fixture after testing it.
```

Class methods inherited from unittest.case.TestCase:

addClassCleanup(function, /, *args, **kwargs) from builtins.type
```
Same as addCleanup, except the cleanup items are called even if
setUpClass fails (unlike tearDownClass).
```

doClassCleanups() from builtins.type
```
Execute all class cleanup functions. Normally called for you after
tearDownClass.
```

setUpClass() from builtins.type
```
Hook method for setting up class fixture before running tests in the class.
```

tearDownClass() from builtins.type

Hook method for deconstructing the class fixture after running all tests in the class.

Data descriptors inherited from unittest.case.TestCase:

__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

Data and other attributes inherited from unittest.case.TestCase:

failureException = <class 'AssertionError'>
Assertion failed.

longMessage = True

maxDiff = 640

Data

al_arr_combo = array([5, 6, 6, 6, 7, 8, 8, 8, 8, 8, ..., 51, 51, 51, 53, 54, 55, 55, 62, 65, 69])
cal_arr_combo = array([8.8, 8.9, 8.9, 9. , 9.1, 9.1, 9.1,... 10.3, 10.3, 10.3, 10.3, 10.3, 10.3, 10.4, 10.6])
men_al_arr = array([9, 10, 10, 11, 11, 11, 11, 12, 12, 13, 1..., 49, 51, 51, 51, 53, 54, 55, 55, 62, 69])
men_cal_arr = array([9.1, 9.1, 9.2, 9.3, 9.3, 9.3, 9.3,... 10.3, 10.3, 10.3, 10.3, 10.3, 10.3, 10.4, 10.6])
women_al_arr = array([5, 6, 6, 6, 7, 8, 8, 8, 8, 8, ..., 29, 30, 30, 36, 37, 37, 39, 46, 47, 65])
women_cal_arr = array([8.8, 8.9, 8.9, 9. , 9.1, 9.1, 9.1,... 10. , 10.1, 10.1, 10.2, 10.2, 10.2, 10.3, 10.3])