`tdda library` PYTHON TEST-DRIVEN DATA ANALYSIS PACKAGE: REFERENCE TEST QUICK REFERENCE

INSTALLATION

STANDARD: **`pip install tdda`**

SOURCE: **`git clone https://github.com/tdda/tdda.git`**

COPY EXAMPLES: **`python -m tdda.referencetest.examples`**

DOCUMENTATION: **`http://pythonhosted.org/tdda/referencetest.html`**

REFERENCE TESTS FOR CSV FILES & PANDAS DATAFRAMES

TEST ITEM	REFERENCE ITEM	METHOD
Generated CSV File	Expected CSV File	**`assertCSVFileCorrect`**(*actual_path, ref_csv, . . .*)
Generated CSV Files	Expected CSV Files	**`assertCSVFilesCorrect`**(*actual_paths, ref_csvs, . . .*)
DataFrame	Expected DF in CSV	**`assertDataFrameCorrect`**(*df, ref_csv, . . .*)
DataFrame	Expected DataFrame	**`assertDataFramesEqual`**(*df, ref_df, . . .*)

COMMON OPTIONAL PARAMETERS

NAME	TYPE	DESCRIPTION
kind	string	*Optional classification; allows only selected outputs to be rewritten*
csv_read_fn	function	*Optional function for reading CSV file (default:* **`pd.read_csv`**) *
precision	int	*Number of decimal places to check (default: no rounding).*
**kwargs	(variable)	*Unknown parameters are passed straight through to CSV reading function*

* *Default CSV reader parameters same as for constraints; see overleaf.*

REFERENCE TESTS FOR STRINGS & TEXT FILES

TEST ITEM	REFERENCE ITEM	METHOD
Generated Text File	Expected text file	**`assertFileCorrect`**(*actual_path, ref_path, . . .*)
Generated Text	Expected text files	**`assertFilesCorrect`**(*actual_paths, ref_paths, . . .*)
String	File with expected string	**`assertStringCorrect`**(*string, ref_path, . . .*)

COMMON OPTIONAL PARAMETERS

NAME	TYPE	DESCRIPTION
kind	string	*Optional classification; allows only selected outputs to be rewritten*
lstrip	boolean	*If* `True`*, lines are left-stripped (of whitespace) before comparison*
rstrip	boolean	*If* `True`*, lines are right-stripped (of whitespace) before comparison*
ignore_substrings	list	*Lines containing any string in list are ignored*
ignore_patterns	list	*Lines matching any regular expressions in list are ignored*

COMMAND-LINE FLAGS

`--write-all`	*Re-write all reference result files*
`--write kind1 [kind2...]`	*Re-write reference results of specified kind(s) only*
`-s`	*(Pytest only): report names of rewritten files. (Not needed for unittest.)*

REFERENCE TESTS UNDER `unittest`

`test_my_module.py:` *(Run this to run the test under* `unittest`*)*

```
from tdda.referencetest.referencetestcase import ReferenceTestCase
import my_module

class MyTest(ReferenceTestCase):
    def test_my_csv_file(self):
        result = my_module.produce_a_csv_file(self.tmp_dir)
        self.assertCSVFileCorrect(result, 'result.csv')

MyTest.set_default_data_location('testdata')

if __name__ == '__main__':
    ReferenceTestCase.main()
```

REFERENCE TESTS UNDER `pytest`

`conftest.py:` *(Required to define fixtures for use in* `pytest`*)*

```
import pytest
from tdda.referencetest import referencepytest

def pytest_addoption(parser):
    referencepytest.addoption(parser)

@pytest.fixture(scope='module')
def ref(request):
    return referencepytest.ref(request)

referencepytest.set_default_data_location('testdata')
```

`test_my_module.py:` *(Run* `pytest`*, which should discover this file and run the test)*

```
from tdda.referencetest import referencepytest
import my_module

def test_my_csv_function(ref):
    resultfile = my_module.produce_a_csv_file(ref.tmp_dir)
    ref.assertCSVFileCorrect(resultfile, 'result.csv')

referencepytest.set_default_data_location('testdata')
```

`tdda library` PYTHON TEST-DRIVEN DATA ANALYSIS PACKAGE: CONSTRAINTS QUICK REFERENCE

INSTALLATION

STANDARD: **`pip install tdda`**

SOURCE: **`git clone https://github.com/tdda/tdda.git`**

COPY EXAMPLES: **`python -m tdda.constraints.examples`**

DOCUMENTATION: **`http://pythonhosted.org/tdda/constraints.html`**

COMMAND-LINE COMMANDS

DISCOVER: **`tdda discover input-file [constraints.tdda]`**

VERIFY: **`tdda verify [FLAGS] input-file [constraints.tdda]`**

If **`constraints.tdda`** *is not provided, it is assumed to be same as input file, with* **`.tdda`** *extension.*

`input-file` *is taken to be a CSV file unless the extension is* **`.feather`***, in which case it is assumed to be a Feather file (see* `https://pypi.python.org/pypi/feather-format/`*)*

VERIFY FLAGS:

`-a, --all`	*Report all fields, even if there are no failures*
`-f, --fields`	*Report only fields with failures*

CONSTRAINT TYPES & APPLICABILITY IN TDDA FILES

KIND	DESCRIPTION	BOOLEAN	NUMERIC	DATE	STRING
`min`	*Minimum allowed value; on verification interpreted with proportionate tolerance* `epsilon`.	✔	✔	✔	✗
`max`	*Maximum allowed value; on verification interpreted with proportionate tolerance* `epsilon`.	✔	✔	✔	✗
`sign`	`"positive"`, `"non-negative"`, `"zero"`, `"non-positive"` *or* `"negative"`.	✔	✔	✗	✗
`max_nulls`	`0` *if nulls not allowed. In principle, can be higher values (in particular,* `1`*), but* `discover` *function does not use these at present.*	✔	✔	✔	✔
`no_duplicates`	`true` *if duplicates are not allowed.*	✔	✔	✔	✔
`min_length`	*smallest allowed string length*	✗	✗	✗	✔
`max_length`	*largest allowed string length*	✗	✗	✗	✔
`rex`	*list of regular expressions; strings must match at least one. (Available from version 0.4.0 on.)*	✗	✗	✗	✔

In all cases, a constraint value of `null` is equivalent to not supplying a constraint.

CONSTRAINT DISCOVERY API

Given a Pandas DataFrame `df`*, this code will discover constraints and write them to* `constraints.tdda`*.*

```
from tdda.constraints.pdconstraints import discover_constraints
constraints = discover_constraints(df)
with open('constraints.tdda', 'w') as f:
    f.write(constraints.to_json())
```

The `constraints` *object returned is a* `DatasetConstraints` *object.*

CONSTRAINT VERIFICATION API

Given a Pandas DataFrame `df` *and a TDDA file* `constraints.tdda`*, this code will verify the dataframe against the constraints*

```
from tdda.constraints.pdconstraints import verify_df

verification = verify_df(df, 'constraints.tdda')

constraints_df = verification.to_frame())
```

The `verification` *object returned is a* `PandasVerification` *object, which has a* `__str__` *method, as well as the to_frame method shown. It also has attributes* `passes` *and* `failures`*, which count the number of passing and failing constraints respectively.*

OPTIONAL PARAMETERS FOR `verify_df`

NAME	TYPE	DESCRIPTION
epsilon	float	*Constraint tolerance (default* `0.01` *i.e. 1%)*
type-checking	string	`'strict'` *or* `'sloppy'` *(default* `'sloppy'`*, i.e. accept similar types)*
report	string	`'all'` *or* `'fields'` *(default* `'all'`*, i.e. show even fields for which all constraints pass*

DEFAULT CSV READER PARAMETERS

```
(index_col=None, infer_datetime_format=True,
 quotechar='"', quoting=csv.QUOTE_MINIMAL,
 escapechar='\\', na_values=['', 'NaN', 'NULL'],
 keep_default_na=False)
```