`tdda library`: REFERENCE TEST QUICK REFERENCE

COMMAND-LINE FLAGS

`--write-all`	*Re-write all reference result files*
`--write kind1 [kind2...]`	*Re-write reference results of specified kind(s) only*
`--tagged`	*Run only tests decorated with* `@tag`.

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*

REFERENCE TESTS UNDER `unittest`

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

```
from tdda.referencetest import ReferenceTestCase, tag
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, tag

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

def pytest_collection_modifyitems(session, config, items):
    referencepytest.tagged(config, items)

@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')
```

INSTALLATION

STANDARD:	**`pip install tdda`**
SOURCE:	**`git clone https://github.com/tdda/tdda.git`**
COPY EXAMPLES:	**`tdda examples`**
DOCUMENTATION:	**`http://tdda.readthedocs.io/en/latest/referencetest.html`**

`tdda library` : CONSTRAINTS QUICK REFERENCE

COMMAND-LINE

DISCOVER: `tdda discover input [constraints.tdda]`

Generates constraints satisfied by **`input`** *data, saving them to file* **`constraints.tdda`** *(else* **`stdout`***).*

`input` *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/`*). If prefixed with a known database, e.g.* **`postgres:table`***, input can be a database table, with connection details specified in* **`~/.tdda_db_conn_postgres`**.

VERIFY: `tdda verify [FLAGS] input constraints.tdda`

Reports any constraints from **`constraints.tdda`** *not satisfied by* **`input`** *data.*

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

DETECT: `tdda detect [FLAGS] input constraints.tdda output`

Like verify, but writes failing records (or their line numbers, index or keys) to **`output`**.

`--write-all`	*Write all records (passes as well as fails)*

REXPY: `rexpy [FLAGS] [input [output]]`

Generates one or more regular expressions that (together) characterize all the strings in input.

CONSTRAINT TYPES & APPLICABILITY IN TDDA FILES

KIND	DESCRIPTION	BOOLEAN	NUMERIC	DATE	STRING
`min / min-len`	*Minimum allowed value; on verification interpreted with proportionate tolerance* `epsilon`.	✔	✔	✔	✔ *le*
`max / max-len`	*Maximum allowed value; on verification interpreted with proportionate tolerance* `epsilon`.	✔	✔	✔	✔ *le*
`sign`	`"positive"`, `"non-negative"`, `"zero"`, `"non-positive"` *or* `"negative"`.	✔	✔	✗	✗
`max_nulls`	`0` *if nulls not allowed. Can be higher values (in particular,* `1`*).*	✔	✔	✔	✔
`no_duplicates`	`true` *if duplicates are not allowed.*	✔	✔	✔	✔
`allowed_values`	*list of values; every string must be one of these*	✗	✗	✗	✔
`rex`	*list of regular expressions; strings must match at least one.*	✗	✗	✗	✔

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

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
```

CONSTRAINT DISCOVERY API

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

```
from tdda.constraints import discover_df
constraints = discover_df(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 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* `to_frame` *method as 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* `0.0` *i.e. 0%)*
`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*

ANOMALY DETECTION API

Given a Pandas DataFrame `df` *and a TDDA file* `constraints.tdda`*, this code will verify the DataFrame against the constraints and construct a DataFrame* `bads_df` *containing failing records.*

```
from tdda.constraints import detect_df

v = detect_df(df, 'constraints.tdda')
bads_df = v.detected()
```

INSTALLATION

STANDARD: **`pip install tdda`**

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

COPY EXAMPLES: **`tdda examples`**

DOCUMENTATION: **`http://tdda.readthedocs.io/en/latest/constraints.html`**