petlx Documentation

Release 0.4

Alistair Miles

March 18, 2013

CONTENTS

`petlx` is a collection of extensions to `petl`, a Python module for extracting, transforming and loading tables of data.

- Documentation: http://petlx.readthedocs.org/
- Source Code: https://github.com/alimanfoo/petlx
- Download: http://pypi.python.org/pypi/petlx
- Mailing List: http://groups.google.com/group/python-etl

For an overview of all functions in the package, see the *genindex*.

CHAPTER

ONE

INSTALLATION

This module is available from the Python Package Index. On Linux distributions you should be able to do `easy_install petlx` or `pip install petlx`. On Windows or Mac you can download manually, extract and run `python setup.py install`.

Note that each submodule within the `petlx` package has dependencies on one or more third party modules which will need to be installed separately.

CHAPTER

TWO

MODULES

2.1 Excel files (openpyxl)

`petlx.xlsx.`**`fromxlsx`**(*filename*, *sheetname*, *checksumfun=None*)

Extract a table from a sheet in an Excel (.xlsx) file.

N.B., the sheet name is case sensitive, so watch out for, e.g., 'Sheet1'.

The package openpyxl is required. Instructions for installation can be found at https://bitbucket.org/ericgazoni/openpyxl/wiki/Home or try `pip install openpyxl`.

2.2 Arrays (numpy)

`petlx.array.`**`toarray`**(*table*, *dtype=None*, *count=-1*, *sample=1000*)

Convenience function to load data from the given *table* into a numpy structured array. E.g.:

```
>>> from petl import look
>>> from petlx.array import toarray
>>> look(table)
+-----------+-------+-------+
| 'foo'     | 'bar' | 'baz' |
+===========+=======+=======+
| 'apples'  | 1     | 2.5   |
+-----------+-------+-------+
| 'oranges' | 3     | 4.4   |
+-----------+-------+-------+
| 'pears'   | 7     | 0.1   |
+-----------+-------+-------+

>>> a = toarray(table)
>>> a
array([('apples', 1, 2.5), ('oranges', 3, 4.4), ('pears', 7, 0.1)],
      dtype=[('foo', '|S7'), ('bar', '<i8'), ('baz', '<f8')])
>>> a['foo']
array(['apples', 'oranges', 'pears'],
      dtype='|S7')
>>> a['bar']
array([1, 3, 7])
>>> a['baz']
array([ 2.5,  4.4,  0.1])
>>> a['foo'][0]
'apples'
```

```
>>> a['bar'][1]
3
>>> a['baz'][2]
0.10000000000000001
```

If no datatype is specified, *sample* rows will be examined to infer an appropriate datatype for each field.

The datatype can be specified as a string, e.g.:

```
>>> a = toarray(table, dtype='a4, i2, f4')
>>> a
array([('appl', 1, 2.5), ('oran', 3, 4.400000095367432),
       ('pear', 7, 0.10000000149011612)],
      dtype=[('foo', '|S4'), ('bar', '<i2'), ('baz', '<f4')])
```

The datatype can also be partially specified, in which case datatypes will be inferred for other fields, e.g.:

```
>>> a = toarray(table, dtype={'foo': 'a4'})
>>> a
array([('appl', 1, 2.5), ('oran', 3, 4.4), ('pear', 7, 0.1)],
      dtype=[('foo', '|S4'), ('bar', '<i8'), ('baz', '<f8')])
```

`petlx.array.`**`fromarray`**`(`*a*`)`
Extract rows from a numpy structured array. New in version 0.4.

2.3 Intervals (bx-python)

The package bx.intervals is required. Instructions for installation can be found at https://bitbucket.org/james_taylor/bx-python/wiki/Home or try `pip install bx-python`.

`petlx.interval.`**`intervallookup`**`(`*table*, *start='start'*, *stop='stop'*, *valuespec=None*, *proximity=0*`)`
Construct an interval lookup for the given table. E.g.:

```
>>> from petlx.interval import intervallookup
>>> table = [['start', 'stop', 'value'],
...          [1, 4, 'foo'],
...          [3, 7, 'bar'],
...          [4, 9, 'baz']]
>>> lkp = intervallookup(table, 'start', 'stop')
>>> lkp[1:2]
[(1, 4, 'foo')]
>>> lkp[2:4]
[(1, 4, 'foo'), (3, 7, 'bar')]
>>> lkp[2:5]
[(1, 4, 'foo'), (3, 7, 'bar'), (4, 9, 'baz')]
>>> lkp[9:14]
[]
>>> lkp[19:140]
[]
>>> lkp[1]
[]
>>> lkp[2]
[(1, 4, 'foo')]
>>> lkp[4]
[(3, 7, 'bar')]
>>> lkp[5]
[(3, 7, 'bar'), (4, 9, 'baz')]
```

Note that there must be a non-zero overlap between the query and the interval for the interval to be retrieved, hence *lkp[1]* returns nothing. Use the *proximity* keyword argument to find intervals within a given distance of the query.

Some examples using the *proximity* and *valuespec* keyword arguments:

```
>>> table = [['start', 'stop', 'value'],
...          [1, 4, 'foo'],
...          [3, 7, 'bar'],
...          [4, 9, 'baz']]
>>> lkp = intervallookup(table, 'start', 'stop', valuespec='value', proximity=1)
>>> lkp[1:2]
['foo']
>>> lkp[2:4]
['foo', 'bar', 'baz']
>>> lkp[2:5]
['foo', 'bar', 'baz']
>>> lkp[9:14]
['baz']
>>> lkp[19:140]
[]
>>> lkp[1]
['foo']
>>> lkp[2]
['foo']
>>> lkp[4]
['foo', 'bar', 'baz']
>>> lkp[5]
['bar', 'baz']
>>> lkp[9]
['baz']
```

New in version 0.2.

`petlx.interval.`**`intervallookupone`**(*table*, *start='start'*, *stop='stop'*, *valuespec=None*, *proximity=0*, *strict=True*)

Construct an interval lookup for the given table, returning at most one result for each query. If `strict=True` is given, queries returning more than one result will raise a *DuplicateKeyError*. If `strict=False` is given, and there is more than one result, the first result is returned.

See also `intervallookup()`. New in version 0.2.

`petlx.interval.`**`intervalrecordlookup`**(*table*, *start='start'*, *stop='stop'*, *proximity=0*)

As `intervallookup()` but return records (dictionaries of values indexed by field name). New in version 0.2.

`petlx.interval.`**`intervalrecordlookupone`**(*table*, *start='start'*, *stop='stop'*, *proximity=0*, *strict=True*)

As `intervallookupone()` but return records (dictionaries of values indexed by field name). New in version 0.2.

`petlx.interval.`**`facetintervallookup`**(*table*, *key*, *start='start'*, *stop='stop'*, *valuespec=None*, *proximity=0*)

Construct a faceted interval lookup for the given table. E.g.:

```
>>> from petl import look
>>> from petlx.interval import facetintervallookup
>>> look(table)
+----------+---------+--------+---------+
| 'type'   | 'start' | 'stop' | 'value' |
+==========+=========+========+=========+
| 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+--------+---------+
| 'apple'  | 3       | 7      | 'bar'   |
+----------+---------+--------+---------+
| 'orange' | 4       | 9      | 'baz'   |
+----------+---------+--------+---------+

>>> lkp = facetintervallookup(table, key='type', start='start', stop='stop')
>>> lkp['apple'][1:2]
[('apple', 1, 4, 'foo')]
>>> lkp['apple'][2:4]
[('apple', 1, 4, 'foo'), ('apple', 3, 7, 'bar')]
>>> lkp['apple'][2:5]
[('apple', 1, 4, 'foo'), ('apple', 3, 7, 'bar')]
>>> lkp['orange'][2:5]
[('orange', 4, 9, 'baz')]
>>> lkp['orange'][9:14]
[]
>>> lkp['orange'][19:140]
[]
>>> lkp['apple'][1]
[]
>>> lkp['apple'][2]
[('apple', 1, 4, 'foo')]
>>> lkp['apple'][4]
[('apple', 3, 7, 'bar')]
>>> lkp['apple'][5]
[('apple', 3, 7, 'bar')]
>>> lkp['orange'][5]
[('orange', 4, 9, 'baz')]
```

New in version 0.2.

`petlx.interval.`**`facetintervallookupone`**(*table, key, start='start', stop='stop', valuespec=None, proximity=0, strict=True*)

Construct a faceted interval lookup for the given table, returning at most one result for each query, e.g.:

```
>>> from petl import look
>>> from petlx.interval import facetintervallookupone
>>> look(table)
+----------+---------+--------+---------+
| 'type'   | 'start' | 'stop' | 'value' |
+==========+=========+========+=========+
| 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+--------+---------+
| 'apple'  | 3       | 7      | 'bar'   |
+----------+---------+--------+---------+
| 'orange' | 4       | 9      | 'baz'   |
+----------+---------+--------+---------+

>>> lkp = facetintervallookupone(table, key='type', start='start', stop='stop', valuespec='value
>>> lkp['apple'][1:2]
'foo'
>>> lkp['apple'][2:4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "petlx/interval.py", line 191, in __getitem__
    raise DuplicateKeyError
petl.util.DuplicateKeyError
```

```
>>> lkp['apple'][2:5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "petlx/interval.py", line 191, in __getitem__
    raise DuplicateKeyError
petl.util.DuplicateKeyError
>>> lkp['apple'][4:5]
'bar'
>>> lkp['orange'][4:5]
'baz'
>>> lkp['apple'][5:7]
'bar'
>>> lkp['orange'][5:7]
'baz'
>>> lkp['apple'][8:9]
>>> lkp['orange'][8:9]
'baz'
>>> lkp['orange'][9:14]
>>> lkp['orange'][19:140]
>>> lkp['apple'][1]
>>> lkp['apple'][2]
'foo'
>>> lkp['apple'][4]
'bar'
>>> lkp['apple'][5]
'bar'
>>> lkp['orange'][5]
'baz'
>>> lkp['apple'][8]
>>> lkp['orange'][8]
'baz'
```

If `strict=True` is given, queries returning more than one result will raise a *DuplicateKeyError*. If `strict=False` is given, and there is more than one result, the first result is returned.

See also `facetintervallookup()`. New in version 0.2.

`petlx.interval.`**`facetintervalrecordlookup`**`(`*table*, *key*, *start='start'*, *stop='stop'*, *proximity=0*`)`

As `facetintervallookup()` but return records (dictionaries of values indexed by field name). New in version 0.2.

`petlx.interval.`**`facetintervalrecordlookupone`**`(`*table*, *key*, *start*, *stop*, *proximity=0*, *strict=True*`)`

As `facetintervallookupone()` but return records (dictionaries of values indexed by field name). New in version 0.2.

`petlx.interval.`**`intervaljoin`**`(`*left*, *right*, *lstart='start'*, *lstop='stop'*, *rstart='start'*, *rstop='stop'*, *lfacet=None*, *rfacet=None*, *proximity=0*`)`

Join two tables by overlapping intervals. E.g.:

```
>>> from petl import look
>>> from petlx.interval import intervaljoin
>>> look(left)
+---------+-------+--------+
| 'begin' | 'end' | 'quux' |
+=========+=======+========+
| 1       | 2     | 'a'    |
+---------+-------+--------+
| 2       | 4     | 'b'    |
+---------+-------+--------+
| 2       | 5     | 'c'    |
+---------+-------+--------+
| 9       | 14    | 'd'    |
+---------+-------+--------+
| 9       | 140   | 'e'    |
+---------+-------+--------+
| 1       | 1     | 'f'    |
+---------+-------+--------+
| 2       | 2     | 'g'    |
+---------+-------+--------+
| 4       | 4     | 'h'    |
+---------+-------+--------+
| 5       | 5     | 'i'    |
+---------+-------+--------+
| 1       | 8     | 'j'    |
+---------+-------+--------+

>>> look(right)
+---------+--------+---------+
| 'start' | 'stop' | 'value' |
+=========+========+=========+
| 1       | 4      | 'foo'   |
+---------+--------+---------+
| 3       | 7      | 'bar'   |
+---------+--------+---------+
| 4       | 9      | 'baz'   |
+---------+--------+---------+

>>> result = intervaljoin(left, right, lstart='begin', lstop='end', rstart='start', rstop='stop'
>>> look(result)
+---------+-------+--------+---------+--------+---------+
| 'begin' | 'end' | 'quux' | 'start' | 'stop' | 'value' |
+=========+=======+========+=========+========+=========+
| 1       | 2     | 'a'    | 1       | 4      | 'foo'   |
+---------+-------+--------+---------+--------+---------+
| 2       | 4     | 'b'    | 1       | 4      | 'foo'   |
+---------+-------+--------+---------+--------+---------+
| 2       | 4     | 'b'    | 3       | 7      | 'bar'   |
+---------+-------+--------+---------+--------+---------+
| 2       | 5     | 'c'    | 1       | 4      | 'foo'   |
+---------+-------+--------+---------+--------+---------+
| 2       | 5     | 'c'    | 3       | 7      | 'bar'   |
+---------+-------+--------+---------+--------+---------+
| 2       | 5     | 'c'    | 4       | 9      | 'baz'   |
+---------+-------+--------+---------+--------+---------+
| 2       | 2     | 'g'    | 1       | 4      | 'foo'   |
+---------+-------+--------+---------+--------+---------+
| 4       | 4     | 'h'    | 3       | 7      | 'bar'   |
+---------+-------+--------+---------+--------+---------+
| 5       | 5     | 'i'    | 3       | 7      | 'bar'   |
+---------+-------+--------+---------+--------+---------+
| 5       | 5     | 'i'    | 4       | 9      | 'baz'   |
+---------+-------+--------+---------+--------+---------+
```

An additional key comparison can be added, e.g.:

```
>>> from petl import look
>>> from petlx.interval import intervaljoin
>>> look(left)
+----------+---------+-------+
| 'fruit'  | 'begin' | 'end' |
+==========+=========+=======+
| 'apple'  | 1       | 2     |
+----------+---------+-------+
| 'apple'  | 2       | 4     |
+----------+---------+-------+
| 'apple'  | 2       | 5     |
+----------+---------+-------+
| 'orange' | 2       | 5     |
+----------+---------+-------+
| 'orange' | 9       | 14    |
+----------+---------+-------+
| 'orange' | 19      | 140   |
+----------+---------+-------+
| 'apple'  | 1       | 1     |
+----------+---------+-------+
| 'apple'  | 2       | 2     |
+----------+---------+-------+
| 'apple'  | 4       | 4     |
+----------+---------+-------+
| 'apple'  | 5       | 5     |
+----------+---------+-------+

>>> look(right)
+----------+---------+--------+---------+
| 'type'   | 'start' | 'stop' | 'value' |
+==========+=========+========+=========+
| 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+--------+---------+
| 'apple'  | 3       | 7      | 'bar'   |
+----------+---------+--------+---------+
| 'orange' | 4       | 9      | 'baz'   |
+----------+---------+--------+---------+

>>> result = intervaljoin(left, right, lstart='begin', lstop='end', rstart='start', rstop='stop'
>>> look(result)
+----------+---------+-------+----------+---------+--------+---------+
| 'fruit'  | 'begin' | 'end' | 'type'   | 'start' | 'stop' | 'value' |
+==========+=========+=======+==========+=========+========+=========+
| 'apple'  | 1       | 2     | 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 2       | 4     | 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 2       | 4     | 'apple'  | 3       | 7      | 'bar'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 2       | 5     | 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 2       | 5     | 'apple'  | 3       | 7      | 'bar'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'orange' | 2       | 5     | 'orange' | 4       | 9      | 'baz'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 2       | 2     | 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 4       | 4     | 'apple'  | 3       | 7      | 'bar'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 5       | 5     | 'apple'  | 3       | 7      | 'bar'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'orange' | 5       | 5     | 'orange' | 4       | 9      | 'baz'   |
+----------+---------+-------+----------+---------+--------+---------+
```

New in version 0.2.

`petlx.interval.`**`intervalleftjoin`**(*left*, *right*, *lstart='start'*, *lstop='stop'*, *rstart='start'*, *rstop='stop'*, *lfacet=None*, *rfacet=None*, *proximity=0*, *missing=None*)

Like `intervaljoin()` but rows from the left table without a match in the right table are also included. E.g.:

```
>>> from petl import look
>>> from petlx.interval import intervalleftjoin
>>> look(left)
+----------+---------+-------+
| 'fruit'  | 'begin' | 'end' |
+==========+=========+=======+
| 'apple'  | 1       | 2     |
+----------+---------+-------+
| 'apple'  | 2       | 4     |
+----------+---------+-------+
| 'apple'  | 2       | 5     |
+----------+---------+-------+
| 'orange' | 2       | 5     |
+----------+---------+-------+
| 'orange' | 9       | 14    |
+----------+---------+-------+
| 'orange' | 19      | 140   |
+----------+---------+-------+
| 'apple'  | 1       | 1     |
+----------+---------+-------+
| 'apple'  | 2       | 2     |
+----------+---------+-------+
| 'apple'  | 4       | 4     |
+----------+---------+-------+
| 'apple'  | 5       | 5     |
+----------+---------+-------+

>>> look(right)
+----------+---------+--------+---------+
| 'type'   | 'start' | 'stop' | 'value' |
+==========+=========+========+=========+
| 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+--------+---------+
| 'apple'  | 3       | 7      | 'bar'   |
+----------+---------+--------+---------+
| 'orange' | 4       | 9      | 'baz'   |
+----------+---------+--------+---------+

>>> result = intervalleftjoin(left, right, lstart='begin', lstop='end', rstart='start', rstop='s
>>> look(result)
+----------+---------+-------+----------+---------+--------+---------+
| 'fruit'  | 'begin' | 'end' | 'type'   | 'start' | 'stop' | 'value' |
+==========+=========+=======+==========+=========+========+=========+
| 'apple'  | 1       | 2     | 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 2       | 4     | 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 2       | 4     | 'apple'  | 3       | 7      | 'bar'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 2       | 5     | 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 2       | 5     | 'apple'  | 3       | 7      | 'bar'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'apple'  | 2       | 5     | 'orange' | 4       | 9      | 'baz'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'orange' | 2       | 5     | 'apple'  | 1       | 4      | 'foo'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'orange' | 2       | 5     | 'apple'  | 3       | 7      | 'bar'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'orange' | 2       | 5     | 'orange' | 4       | 9      | 'baz'   |
+----------+---------+-------+----------+---------+--------+---------+
| 'orange' | 9       | 14    | None     | None    | None   | None    |
+----------+---------+-------+----------+---------+--------+---------+
```

New in version 0.2.

2.4 GFF3 Utilities

`petlx.gff3.`**`fromgff3`**`(`*filename*`)`
Extract feature rows from a GFF3 file. New in version 0.2.

`petlx.gff3.`**`gff3lookup`**`(`*features*, *facet='seqid'*`)`
Build a GFF3 feature lookup based on interval trees. See also `petlx.interval.facetintervallookup()`. New in version 0.2.

`petlx.gff3.`**`gff3join`**`(`*table*, *features*, *seqid='seqid'*, *start='start'*, *end='end'*, *proximity=1*`)`
Join with a table of GFF3 features. See also `petlx.interval.intervaljoin()`. New in version 0.2.

`petlx.gff3.`**`gff3leftjoin`**`(`*table*, *features*, *seqid='seqid'*, *start='start'*, *end='end'*, *proximity=1*`)`
Left join with a table of GFF3 features. See also `petlx.interval.intervalleftjoin()`. New in version 0.2.

2.5 HDF5 Files (pytables)

The package pytables is required. Instructions for installation can be found at http://pytables.github.com/usersguide/installation.html or try apt-get install python-tables.

`petlx.hdf5.`**`fromhdf5`**`(`*source*, *where=None*, *name=None*, *condition=None*, *condvars=None*, *start=None*, *stop=None*, *step=None*`)`
Provides access to an HDF5 table. E.g.:

```
>>> from petl import look
>>> from petlx.hdf5 import fromhdf5
>>> table1 = fromhdf5('test1.h5', '/testgroup', 'testtable')
>>> look(table1)
+-------+----------+
| 'foo' | 'bar'    |
+=======+==========+
| 1     | 'asdfgh' |
+-------+----------+
| 2     | 'qwerty' |
+-------+----------+
| 3     | 'zxcvbn' |
+-------+----------+
```

Some alternative signatures:

```
>>> # just specify path to table node
... table1 = fromhdf5('test1.h5', '/testgroup/testtable')
>>>
>>> # use an existing tables.File object
... import tables
>>> h5file = tables.openFile('test1.h5')
>>> table1 = fromhdf5(h5file, '/testgroup/testtable')
>>>
>>> # use an existing tables.Table object
... h5tbl = h5file.getNode('/testgroup/testtable')
>>> table1 = fromhdf5(h5tbl)
>>>
>>> # use a condition to filter data
... table2 = fromhdf5(h5tbl, condition="(foo < 3)")
>>> look(table2)
+-------+----------+
| 'foo' | 'bar'    |
+=======+==========+
| 1     | 'asdfgh' |
+-------+----------+
| 2     | 'qwerty' |
+-------+----------+
```

New in version 0.3.

petlx.hdf5.**fromhdf5sorted**(*source*, *where=None*, *name=None*, *sortby=None*, *checkCSI=False*, *start=None*, *stop=None*, *step=None*)

Provides access to an HDF5 table, sorted by an indexed column, e.g.:

```
>>> # set up a new hdf5 table to demonstrate with
... import tables
>>> h5file = tables.openFile("test1.h5", mode="w", title="Test file")
>>> h5file.createGroup('/', 'testgroup', 'Test Group')
/testgroup (Group) 'Test Group'
  children := []
>>> class FooBar(tables.IsDescription):
...     foo = tables.Int32Col(pos=0)
...     bar = tables.StringCol(6, pos=2)
...
>>> h5table = h5file.createTable('/testgroup', 'testtable', FooBar, 'Test Table')
>>>
>>> # load some data into the table
... table1 = (('foo', 'bar'),
...           (3, 'asdfgh'),
...           (2, 'qwerty'),
...           (1, 'zxcvbn'))
>>>
>>> for row in table1[1:]:
...     for i, f in enumerate(table1[0]):
...         h5table.row[f] = row[i]
...     h5table.row.append()
...
>>> h5table.cols.foo.createCSIndex() # CS index is required
0
```

```
>>> h5file.flush()
>>> h5file.close()
>>>
>>> # access the data, sorted by the indexed column
... from petl import look
>>> from petlx.hdf5 import fromhdf5sorted
>>> table2 = fromhdf5sorted('test1.h5', '/testgroup', 'testtable', sortby='foo')
>>> look(table2)
+-------+----------+
| 'foo' | 'bar'    |
+=======+==========+
| 1     | 'zxcvbn' |
+-------+----------+
| 2     | 'qwerty' |
+-------+----------+
| 3     | 'asdfgh' |
+-------+----------+
```

New in version 0.3.

`petlx.hdf5.`**`tohdf5`**(*table*, *source*, *where=None*, *name=None*, *create=False*, *description=None*, *title=''*, *filters=None*, *expectedrows=10000*, *chunkshape=None*, *byteorder=None*, *createparents=False*, *sample=1000*)

Write to an HDF5 table. If *create* is *False*, assumes the table already exists, and attempts to truncate it before loading. If *create* is *True*, any existing table is dropped, and a new table is created; if *description* is None, the datatype will be guessed. E.g.:

```
>>> from petl import look
>>> look(table1)
+-------+----------+
| 'foo' | 'bar'    |
+=======+==========+
| 1     | 'asdfgh' |
+-------+----------+
| 2     | 'qwerty' |
+-------+----------+
| 3     | 'zxcvbn' |
+-------+----------+

>>> from petlx.hdf5 import tohdf5, fromhdf5
>>> tohdf5(table1, 'test1.h5', '/testgroup', 'testtable', create=True, createparents=True)
>>> look(fromhdf5('test1.h5', '/testgroup', 'testtable'))
+-------+----------+
| 'foo' | 'bar'    |
+=======+==========+
| 1     | 'asdfgh' |
+-------+----------+
| 2     | 'qwerty' |
+-------+----------+
| 3     | 'zxcvbn' |
+-------+----------+
```

See also `appendhdf5()`. New in version 0.3.

`petlx.hdf5.`**`appendhdf5`**(*table*, *source*, *where=None*, *name=None*)

Like `tohdf5()` but don't truncate the table before loading. New in version 0.3.

2.6 Tabix (pysam)

petlx.tabix.**fromtabix**(*filename*, *reference=None*, *start=None*, *end=None*, *region=None*, *header=None*)

Extract rows from a tabix indexed file. E.g.:

```
>>> from petlx.tabix import fromtabix
>>> from petl import look
>>> t = fromtabix('test.bed.gz', region='Pf3D7_02_v3:100000-200000')
>>> look(t)
+---------------+----------+----------+-----------------------------+
| '#chrom'      | 'start'  | 'end'    | 'region'                    |
+===============+==========+==========+=============================+
| 'Pf3D7_02_v3' | '23100'  | '105800' | 'SubtelomericHypervariable' |
+---------------+----------+----------+-----------------------------+
| 'Pf3D7_02_v3' | '105800' | '447300' | 'Core'                      |
+---------------+----------+----------+-----------------------------+
```

New in version 0.4.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX