Table Classes

Each `Dataset` object is backed by a PyArrow Table.
A Table can be loaded from either the disk (memory mapped) or in memory.
Several Table types are available, and they all inherit from table.Table.

Tabledatasets.table.Table

class datasets.table.Tabledatasets.table.Tablehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L153[{"name": "table", "val": ": Table"}]

Wraps a pyarrow Table by using composition.
This is the base class for `InMemoryTable`, `MemoryMappedTable` and `ConcatenationTable`.

It implements all the basic attributes/methods of the pyarrow Table class except
the Table transforms:
`slice, filter, flatten, combine_chunks, cast, add_column, append_column, remove_column, set_column,`
and `drop`.

The implementation of these methods differs for the subclasses.

validatedatasets.table.Table.validatehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L178[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **full** (`bool`, defaults to `False`) --
If `True`, run expensive checks, otherwise cheap checks only.0- `pa.lib.ArrowInvalid` -- if validation fails`pa.lib.ArrowInvalid`

Perform validation checks. An exception is raised if validation fails.

By default only cheap validation checks are run. Pass `full=True`
for thorough validation checks (potentially `O(n)`).

equalsdatasets.table.Table.equalshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L194[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **other** (Table) --
Table to compare against.

- **check_metadata** `bool`, defaults to `False`) --
  Whether schema metadata equality should be checked as well.0 `bool`

Check if contents of two tables are equal.

to_batchesdatasets.table.Table.to_batcheshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L211[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]-
max_chunksize (`int`, defaults to `None`) --
Maximum size for `RecordBatch` chunks. Individual chunks may be
smaller depending on the chunk layout of individual columns.0 `List[pyarrow.RecordBatch]`

Convert Table to list of (contiguous) `RecordBatch` objects.

to_pydictdatasets.table.Table.to_pydicthttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L225[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}] `dict`

Convert the Table to a `dict` or `OrderedDict`.

to_pandasdatasets.table.Table.to_pandashttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L243[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]-
memory_pool (`MemoryPool`, defaults to `None`) --
Arrow MemoryPool to use for allocations. Uses the default memory
pool is not passed.

- **strings_to_categorical** (`bool`, defaults to `False`) --
  Encode string (UTF8) and binary types to `pandas.Categorical`.
- **categories** (`list`, defaults to `empty`) --
  List of fields that should be returned as `pandas.Categorical`. Only
  applies to table-like data structures.
- **zero_copy_only** (`bool`, defaults to `False`) --
  Raise an `ArrowException` if this function call would require copying
  the underlying data.
- **integer_object_nulls** (`bool`, defaults to `False`) --
  Cast integers with nulls to objects.
- **date_as_object** (`bool`, defaults to `True`) --
  Cast dates to objects. If `False`, convert to `datetime64[ns]` dtype.
- **timestamp_as_object** (`bool`, defaults to `False`) --

Cast non-nanosecond timestamps (`np.datetime64`) to objects. This is useful if you have timestamps that don't fit in the normal date range of nanosecond timestamps (1678 CE-2262 CE).
If `False` , all timestamps are converted to `datetime64[ns]` dtype.

- **use_threads** (`bool` , defaults to `True`) --
  Whether to parallelize the conversion using multiple threads.
- **deduplicate_objects** (`bool` , defaults to `False`) --
  Do not create multiple copies Python objects when created, to save on memory use. Conversion will be slower.
- **ignore_metadata** (`bool` , defaults to `False`) --
  If `True` , do not use the 'pandas' metadata to reconstruct the DataFrame index, if present.
- **safe** (`bool` , defaults to `True`) --
  For certain data types, a cast is needed in order to store the data in a pandas DataFrame or Series (e.g. timestamps are always stored as nanoseconds in pandas). This option controls whether it is a safe cast or not.
- **split_blocks** (`bool` , defaults to `False`) --
  If `True` , generate one internal "block" for each column when creating a pandas.DataFrame from a `RecordBatch` or `Table` . While this can temporarily reduce memory note that various pandas operations can trigger "consolidation" which may balloon memory use.
- **self_destruct** (`bool` , defaults to `False`) --
  EXPERIMENTAL: If `True` , attempt to deallocate the originating Arrow memory while converting the Arrow object to pandas. If you use the object after calling `to_pandas` with this option it will crash your program.
- **types_mapper** (`function` , defaults to `None`) --
  A function mapping a pyarrow DataType to a pandas `ExtensionDtype` . This can be used to override the default pandas type for conversion of built-in pyarrow types or in absence of `pandas_metadata` in the Table schema. The function receives a pyarrow DataType and is expected to return a pandas `ExtensionDtype` or `None` if the default conversion should be used for that type. If you have a dictionary mapping, you can pass `dict.get` as function.0 `pandas.Series` or

`pandas.DataFrame` `pandas.Series` or `pandas.DataFrame` depending on type of object

Convert to a pandas-compatible NumPy array or DataFrame, as appropriate.

to_stringdatasets.table.Table.to_stringhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L305[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]

fielddatasets.table.Table.fieldhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L324[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`Union[int, str]`) -- The index or name of the field to retrieve.0 `pyarrow.Field`

Select a schema field by its column name or numeric index.

columndatasets.table.Table.columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L337[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`Union[int, str]`) -- The index or name of the column to retrieve.0 `pyarrow.ChunkedArray`

Select a column by its column name, or numeric index.

itercolumnsdatasets.table.Table.itercolumnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L350[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}] `pyarrow.ChunkedArray`

Iterator over all columns in their numerical order.

schemadatasets.table.Table.schemahttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L359[] `pyarrow.Schema`

Schema of the table and its columns.

columnsdatasets.table.Table.columnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L369[] `List[pa.ChunkedArray]`

List of all columns in numerical order.

num_columnsdatasets.table.Table.num_columnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L379[]int

Number of columns in this table.

num_rowsdatasets.table.Table.num_rowshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L389[]int

Number of rows in this table.

Due to the definition of a table, all columns have the same number of
rows.

shapedatasets.table.Table.shapehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L402[] `(int, int)` Number of rows and number of columns.

Dimensions of the table: (#rows, #columns).

nbytesdatasets.table.Table.nbyteshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L412[]

Total number of bytes consumed by the elements of the table.

InMemoryTabledatasets.table.InMemoryTable

class datasets.table.InMemoryTabledatasets.table.InMemoryTablehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L638[{"name": "table", "val": ": Table"}]

The table is said in-memory when it is loaded into the user's RAM.

Pickling it does copy all the data using memory.
Its implementation is simple and uses the underlying pyarrow Table methods directly.

This is different from the `MemoryMapped` table, for which pickling doesn't copy all the data in memory. For a `MemoryMapped`, unpickling instead reloads the table from the disk.

`InMemoryTable` must be used when data fit in memory, while `MemoryMapped` are reserved for data bigger than memory or when you want the memory footprint of your application to stay low.

validatedatasets.table.InMemoryTable.validatehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L178[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **full** (`bool`, defaults to `False`) --
If `True`, run expensive checks, otherwise cheap checks only.0- `pa.lib.ArrowInvalid` -- if

validation fails `pa.lib.ArrowInvalid`

Perform validation checks. An exception is raised if validation fails.

By default only cheap validation checks are run. Pass `full=True`
for thorough validation checks (potentially `O(n)`).

equalsdatasets.table.InMemoryTable.equalshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L194[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **other** (Table) --
Table to compare against.

- **check_metadata** `bool`, defaults to `False`) --
  Whether schema metadata equality should be checked as well.0 `bool`

Check if contents of two tables are equal.

to_batchesdatasets.table.InMemoryTable.to_batcheshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L211[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **max_chunksize** (`int`, defaults to `None`) --
Maximum size for `RecordBatch` chunks. Individual chunks may be
smaller depending on the chunk layout of individual columns.0 `List[pyarrow.RecordBatch]`

Convert Table to list of (contiguous) `RecordBatch` objects.

to_pydictdatasets.table.InMemoryTable.to_pydicthttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L225[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}] `dict`

Convert the Table to a `dict` or `OrderedDict`.

to_pandasdatasets.table.InMemoryTable.to_pandashttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L243[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **memory_pool** (`MemoryPool`, defaults to `None`) --
Arrow MemoryPool to use for allocations. Uses the default memory
pool is not passed.

- **strings_to_categorical** (`bool`, defaults to `False`) --
  Encode string (UTF8) and binary types to `pandas.Categorical`.

- **categories** (`list` , defaults to `empty`) --
  List of fields that should be returned as `pandas.Categorical` . Only applies to table-like data structures.
- **zero_copy_only** (`bool` , defaults to `False`) --
  Raise an `ArrowException` if this function call would require copying the underlying data.
- **integer_object_nulls** (`bool` , defaults to `False`) --
  Cast integers with nulls to objects.
- **date_as_object** (`bool` , defaults to `True`) --
  Cast dates to objects. If `False` , convert to `datetime64[ns]` dtype.
- **timestamp_as_object** (`bool` , defaults to `False`) --
  Cast non-nanosecond timestamps (`np.datetime64`) to objects. This is useful if you have timestamps that don't fit in the normal date range of nanosecond timestamps (1678 CE-2262 CE).
  If `False` , all timestamps are converted to `datetime64[ns]` dtype.
- **use_threads** (`bool` , defaults to `True`) --
  Whether to parallelize the conversion using multiple threads.
- **deduplicate_objects** (`bool` , defaults to `False`) --
  Do not create multiple copies Python objects when created, to save on memory use. Conversion will be slower.
- **ignore_metadata** (`bool` , defaults to `False`) --
  If `True` , do not use the 'pandas' metadata to reconstruct the DataFrame index, if present.
- **safe** (`bool` , defaults to `True`) --
  For certain data types, a cast is needed in order to store the data in a pandas DataFrame or Series (e.g. timestamps are always stored as nanoseconds in pandas). This option controls whether it is a safe cast or not.
- **split_blocks** (`bool` , defaults to `False`) --
  If `True` , generate one internal "block" for each column when creating a pandas.DataFrame from a `RecordBatch` or `Table` . While this can temporarily reduce memory note that various pandas operations can trigger "consolidation" which may balloon memory use.
- **self_destruct** (`bool` , defaults to `False`) --
  EXPERIMENTAL: If `True` , attempt to deallocate the originating Arrow

memory while converting the Arrow object to pandas. If you use the object after calling `to_pandas` with this option it will crash your program.

- **types_mapper** (`function` , defaults to `None`) --
  A function mapping a pyarrow DataType to a pandas `ExtensionDtype` . This can be used to override the default pandas type for conversion of built-in pyarrow types or in absence of `pandas_metadata` in the Table schema. The function receives a pyarrow DataType and is expected to return a pandas `ExtensionDtype` or `None` if the default conversion should be used for that type. If you have a dictionary mapping, you can pass `dict.get` as function.0 `pandas.Series` or `pandas.DataFrame` `pandas.Series` or `pandas.DataFrame` depending on type of object

Convert to a pandas-compatible NumPy array or DataFrame, as appropriate.

to_stringdatasets.table.InMemoryTable.to_stringhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L305[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]

fielddatasets.table.InMemoryTable.fieldhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L324[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`Union[int, str]`) --
The index or name of the field to retrieve.0 `pyarrow.Field`

Select a schema field by its column name or numeric index.

columndatasets.table.InMemoryTable.columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L337[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`Union[int, str]`) --
The index or name of the column to retrieve.0 `pyarrow.ChunkedArray`

Select a column by its column name, or numeric index.

itercolumnsdatasets.table.InMemoryTable.itercolumnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L350[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}] `pyarrow.ChunkedArray`

Iterator over all columns in their numerical order.

schemadatasets.table.InMemoryTable.schemahttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L359[] `pyarrow.Schema`

Schema of the table and its columns.

columnsdatasets.table.InMemoryTable.columnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L369[] `List[pa.ChunkedArray]`

List of all columns in numerical order.

num_columnsdatasets.table.InMemoryTable.num_columnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L379[]int

Number of columns in this table.

num_rowsdatasets.table.InMemoryTable.num_rowshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L389[]int

Number of rows in this table.

Due to the definition of a table, all columns have the same number of
rows.

shapedatasets.table.InMemoryTable.shapehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L402[] `(int, int)` Number of rows and number of columns.

Dimensions of the table: (#rows, #columns).

nbytesdatasets.table.InMemoryTable.nbyteshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L412[]

Total number of bytes consumed by the elements of the table.

column_namesdatasets.table.InMemoryTable.column_nameshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L419[]

Names of the table's columns.

slicedatasets.table.InMemoryTable.slicehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L793[{"name": "offset", "val": " = 0"}, {"name": "length", "val": " = None"}]- **offset** (`int` , defaults to `0`) --

Offset from start of table to slice.

- **length** (`int` , defaults to `None`) --
  Length of slice (default is until end of table starting from
  offset).0 `datasets.table.Table`

Compute zero-copy slice of this Table.

filterdatasets.table.InMemoryTable.filterhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L810[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]

Select records from a Table. See `pyarrow.compute.filter` for full usage.

flattendatasets.table.InMemoryTable.flattenhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L816[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **memory_pool** (`MemoryPool` , defaults to `None`) --
For memory allocations, if required, otherwise use default pool.0 `datasets.table.Table`

Flatten this Table. Each column with a struct type is flattened
into one column per struct field. Other columns are left unchanged.

combine_chunksdatasets.table.InMemoryTable.combine_chunkshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L830[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **memory_pool** (`MemoryPool` , defaults to `None`) --
For memory allocations, if required, otherwise use default pool.0 `datasets.table.Table`

Make a new table by combining the chunks this table has.

All the underlying chunks in the `ChunkedArray` of each column are
concatenated into zero or one chunk.

castdatasets.table.InMemoryTable.casthttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L846[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **target_schema** (`Schema`) --
Schema to cast to, the names and order of fields must match.

- **safe** (`bool` , defaults to `True`) --
  Check for overflows or other unsafe conversions.0 `datasets.table.Table`

Cast table values to another schema.

replace_schema_metadatadatasets.table.InMemoryTable.replace_schema_metadatahttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L861[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **metadata** (`dict` , defaults to `None`)
--0 `datasets.table.Table` shallow_copy

EXPERIMENTAL: Create shallow copy of table by replacing schema
key-value metadata with the indicated new metadata (which may be `None` ,
which deletes any existing metadata).

add_columndatasets.table.InMemoryTable.add_columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L875[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`int`) --
Index to place the column at.

- **field_** (`Union[str, pyarrow.Field]`) --
  If a string is passed then the type is deduced from the column
  data.
- **column** (`Union[pyarrow.Array, List[pyarrow.Array]]`) --
  Column data.0 `datasets.table.Table` New table with the passed column added.

Add column to Table at position.

A new table is returned with the column added, the original table
object is left unchanged.

append_columndatasets.table.InMemoryTable.append_columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L896[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **field_** (`Union[str, pyarrow.Field]`) --
If a string is passed then the type is deduced from the column
data.

- **column** (`Union[pyarrow.Array, List[pyarrow.Array]]`) --
  Column data.0 `datasets.table.Table` New table with the passed column added.

Append column at end of columns.

remove_columndatasets.table.InMemoryTable.remove_columnhttps://github.com/huggingface/

datasets/blob/4.2.0/src/datasets/table.py#L913[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`int`) --
Index of column to remove.0 `datasets.table.Table` New table without the column.

Create new Table with the indicated column removed.

set_columndatasets.table.InMemoryTable.set_columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L927[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`int`) --
Index to place the column at.

- **field_** (`Union[str, pyarrow.Field]`) --
  If a string is passed then the type is deduced from the column
  data.
- **column** (`Union[pyarrow.Array, List[pyarrow.Array]]`) --
  Column data.0 `datasets.table.Table` New table with the passed column set.

Replace column in Table at position.

rename_columnsdatasets.table.InMemoryTable.rename_columnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L946[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]

Create new table with columns renamed to provided names.

selectdatasets.table.InMemoryTable.selecthttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L969[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **columns** (`Union[List[str], List[int]]`) --
The column names or integer indices to select.0datasets.table.TableNew table with the specified columns, and metadata preserved.

Select columns of the table.

Returns a new table with the specified columns, and metadata preserved.

dropdatasets.table.InMemoryTable.drophttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L952[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **columns** (`List[str]`) --
List of field names referencing existing columns.0 `datasets.table.Table` New table without the

columns.- `KeyError` -- : if any of the passed columns name are not existing.`KeyError`

Drop one or more columns and return a new table.

from_filedatasets.table.InMemoryTable.from_filehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L653[{"name": "filename", "val": ": str"}]

from_bufferdatasets.table.InMemoryTable.from_bufferhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L658[{"name": "buffer", "val": ": Buffer"}]

from_pandasdatasets.table.InMemoryTable.from_pandashttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L663[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **df** (`pandas.DataFrame`) --

- **schema** (`pyarrow.Schema` , *optional*) --
  The expected schema of the Arrow Table. This can be used to
  indicate the type of columns if we cannot infer it automatically.
  If passed, the output will have exactly this schema. Columns
  specified in the schema that are not found in the DataFrame columns
  or its index will raise an error. Additional columns or index
  levels in the DataFrame which are not specified in the schema will
  be ignored.
- **preserve_index** (`bool` , *optional*) --
  Whether to store the index as an additional column in the resulting
  `Table` . The default of None will store the index as a column,
  except for RangeIndex which is stored as metadata only. Use
  `preserve_index=True` to force it to be stored as a column.
- **nthreads** (`int` , defaults to `None` (may use up to system CPU count threads)) --
  If greater than 1, convert columns to Arrow in parallel using
  indicated number of threads.
- **columns** (`List[str]` , *optional*) --
  List of column to be converted. If `None` , use all columns.
- **safe** (`bool` , defaults to `True`) --
  Check for overflows or other unsafe conversions,0 `datasets.table.Table`

Convert pandas.DataFrame to an Arrow Table.

The column types in the resulting Arrow Table are inferred from the

dtypes of the pandas.Series in the DataFrame. In the case of non-object Series, the NumPy dtype is translated to its Arrow equivalent. In the case of `object`, we need to guess the datatype by looking at the Python objects in this Series.

Be aware that Series of the `object` dtype don't carry enough information to always lead to a meaningful Arrow type. In the case that we cannot infer a type, e.g. because the DataFrame is of length 0 or the Series only contains `None/nan` objects, the type is set to null. This behavior can be avoided by constructing an explicit schema and passing it to this function.

Examples:

```
>>> import pandas as pd
>>> import pyarrow as pa
>>> df = pd.DataFrame({
    ...     'int': [1, 2],
    ...     'str': ['a', 'b']
    ... })
>>> pa.Table.from_pandas(df)
<pyarrow.lib.Table object at 0x7f05d1fb1b40>
```

from_arraysdatasets.table.InMemoryTable.from_arrayshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L721[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **arrays** (`List[Union[pyarrow.Array, pyarrow.ChunkedArray]]`) -- Equal-length arrays that should form the table.

- **names** (`List[str]` , *optional*) -- Names for the table columns. If not passed, schema must be passed.
- **schema** (`Schema` , defaults to `None`) -- Schema for the created table. If not passed, names must be passed.
- **metadata** (`Union[dict, Mapping]` , defaults to `None`) -- Optional metadata for the schema (if inferred).0 `datasets.table.Table`

Construct a Table from Arrow arrays.

from_pydictdatasets.table.InMemoryTable.from_pydicthttps://github.com/huggingface/datasets/

blob/4.2.0/src/datasets/table.py#L741[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **mapping** (`Union[dict, Mapping]`) --
A mapping of strings to Arrays or Python lists.

- **schema** (`Schema` , defaults to `None`) --
  If not passed, will be inferred from the Mapping values
- **metadata** (`Union[dict, Mapping]` , defaults to `None`) --
  Optional metadata for the schema (if inferred).0 `datasets.table.Table`

Construct a Table from Arrow arrays or columns.

from_batchesdatasets.table.InMemoryTable.from_batcheshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L777[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **batches** (`Union[Sequence[pyarrow.RecordBatch], Iterator[pyarrow.RecordBatch]]`) --
Sequence of `RecordBatch` to be converted, all schemas must be equal.

- **schema** (`Schema` , defaults to `None`) --
  If not passed, will be inferred from the first `RecordBatch` .0 `datasets.table.Table`

Construct a Table from a sequence or iterator of Arrow `RecordBatches` .

MemoryMappedTabledatasets.table.MemoryMappedTable

class datasets.table.MemoryMappedTabledatasets.table.MemoryMappedTablehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L989[{"name": "table", "val": ": Table"}, {"name": "path", "val": ": str"}, {"name": "replays", "val": ": typing.Optional[list[tuple[str, tuple, dict]]] = None"}]

The table is said memory mapped when it doesn't use the user's RAM but loads the data from the disk instead.

Pickling it doesn't copy the data into memory.
Instead, only the path to the memory mapped arrow file is pickled, as well as the list of transforms to "replay" when reloading the table from the disk.

Its implementation requires to store an history of all the transforms that were applied to the underlying pyarrow Table, so that they can be "replayed" when reloading the Table from the disk.

This is different from the `InMemoryTable` table, for which pickling does copy all the data in memory.

`InMemoryTable` must be used when data fit in memory, while `MemoryMapped` are reserved for data bigger than memory or when you want the memory footprint of your application to stay low.

validatedatasets.table.MemoryMappedTable.validatehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L178[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **full** (`bool` , defaults to `False`) --
If `True` , run expensive checks, otherwise cheap checks only.0- `pa.lib.ArrowInvalid` -- if validation fails `pa.lib.ArrowInvalid`

Perform validation checks. An exception is raised if validation fails.

By default only cheap validation checks are run. Pass `full=True` for thorough validation checks (potentially `O(n)`).

equalsdatasets.table.MemoryMappedTable.equalshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L194[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **other** (Table) --
Table to compare against.

- **check_metadata** `bool` , defaults to `False`) --
  Whether schema metadata equality should be checked as well.0 `bool`

Check if contents of two tables are equal.

to_batchesdatasets.table.MemoryMappedTable.to_batcheshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L211[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **max_chunksize** (`int` , defaults to `None`) --
Maximum size for `RecordBatch` chunks. Individual chunks may be smaller depending on the chunk layout of individual columns.0 `List[pyarrow.RecordBatch]`

Convert Table to list of (contiguous) `RecordBatch` objects.

to_pydictdatasets.table.MemoryMappedTable.to_pydicthttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L225[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]`dict`

Convert the Table to a `dict` or `OrderedDict`.

to_pandasdatasets.table.MemoryMappedTable.to_pandashttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L243[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **memory_pool** (`MemoryPool` , defaults to `None`) --
Arrow MemoryPool to use for allocations. Uses the default memory
pool is not passed.

- **strings_to_categorical** (`bool` , defaults to `False`) --
  Encode string (UTF8) and binary types to `pandas.Categorical` .
- **categories** (`list` , defaults to `empty`) --
  List of fields that should be returned as `pandas.Categorical` . Only
  applies to table-like data structures.
- **zero_copy_only** (`bool` , defaults to `False`) --
  Raise an `ArrowException` if this function call would require copying
  the underlying data.
- **integer_object_nulls** (`bool` , defaults to `False`) --
  Cast integers with nulls to objects.
- **date_as_object** (`bool` , defaults to `True`) --
  Cast dates to objects. If `False` , convert to `datetime64[ns]` dtype.
- **timestamp_as_object** (`bool` , defaults to `False`) --
  Cast non-nanosecond timestamps (`np.datetime64`) to objects. This is
  useful if you have timestamps that don't fit in the normal date
  range of nanosecond timestamps (1678 CE-2262 CE).
  If `False` , all timestamps are converted to `datetime64[ns]` dtype.
- **use_threads** (`bool` , defaults to `True`) --
  Whether to parallelize the conversion using multiple threads.
- **deduplicate_objects** (`bool` , defaults to `False`) --
  Do not create multiple copies Python objects when created, to save
  on memory use. Conversion will be slower.
- **ignore_metadata** (`bool` , defaults to `False`) --
  If `True` , do not use the 'pandas' metadata to reconstruct the

DataFrame index, if present.

- **safe** (`bool` , defaults to `True`) --
  For certain data types, a cast is needed in order to store the data in a pandas DataFrame or Series (e.g. timestamps are always stored as nanoseconds in pandas). This option controls whether it is a safe cast or not.
- **split_blocks** (`bool` , defaults to `False`) --
  If `True` , generate one internal "block" for each column when creating a pandas.DataFrame from a `RecordBatch` or `Table` . While this can temporarily reduce memory note that various pandas operations can trigger "consolidation" which may balloon memory use.
- **self_destruct** (`bool` , defaults to `False`) --
  EXPERIMENTAL: If `True` , attempt to deallocate the originating Arrow memory while converting the Arrow object to pandas. If you use the object after calling `to_pandas` with this option it will crash your program.
- **types_mapper** (`function` , defaults to `None`) --
  A function mapping a pyarrow DataType to a pandas `ExtensionDtype` . This can be used to override the default pandas type for conversion of built-in pyarrow types or in absence of `pandas_metadata` in the Table schema. The function receives a pyarrow DataType and is expected to return a pandas `ExtensionDtype` or `None` if the default conversion should be used for that type. If you have a dictionary mapping, you can pass `dict.get` as function.0 `pandas.Series` or `pandas.DataFrame` `pandas.Series` or `pandas.DataFrame` depending on type of object

Convert to a pandas-compatible NumPy array or DataFrame, as appropriate.

to_stringdatasets.table.MemoryMappedTable.to_stringhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L305[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]

fielddatasets.table.MemoryMappedTable.fieldhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L324[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`Union[int, str]`) --
The index or name of the field to retrieve.0 `pyarrow.Field`

Select a schema field by its column name or numeric index.

columndatasets.table.MemoryMappedTable.columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L337[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`Union[int, str]`) --
The index or name of the column to retrieve.0 `pyarrow.ChunkedArray`

Select a column by its column name, or numeric index.

itercolumnsdatasets.table.MemoryMappedTable.itercolumnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L350[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}] `pyarrow.ChunkedArray`

Iterator over all columns in their numerical order.

schemadatasets.table.MemoryMappedTable.schemahttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L359[] `pyarrow.Schema`

Schema of the table and its columns.

columnsdatasets.table.MemoryMappedTable.columnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L369[] `List[pa.ChunkedArray]`

List of all columns in numerical order.

num_columnsdatasets.table.MemoryMappedTable.num_columnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L379[]int

Number of columns in this table.

num_rowsdatasets.table.MemoryMappedTable.num_rowshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L389[]int

Number of rows in this table.

Due to the definition of a table, all columns have the same number of
rows.

shapedatasets.table.MemoryMappedTable.shapehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L402[] `(int, int)` Number of rows and number of columns.

Dimensions of the table: (#rows, #columns).

nbytesdatasets.table.MemoryMappedTable.nbyteshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L412[]

Total number of bytes consumed by the elements of the table.

column_namesdatasets.table.MemoryMappedTable.column_nameshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L419[]

Names of the table's columns.

slicedatasets.table.MemoryMappedTable.slicehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1048[{"name": "offset", "val": " = 0"}, {"name": "length", "val": " = None"}]- **offset** (`int` , defaults to `0`) --
Offset from start of table to slice.

- **length** (`int` , defaults to `None`) --
  Length of slice (default is until end of table starting from offset).0`datasets.table.Table`

Compute zero-copy slice of this Table.

filterdatasets.table.MemoryMappedTable.filterhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1067[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]

Select records from a Table. See `pyarrow.compute.filter` for full usage.

flattendatasets.table.MemoryMappedTable.flattenhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1075[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **memory_pool** (`MemoryPool` , defaults to `None`) --
For memory allocations, if required, otherwise use default pool.0`datasets.table.Table`

Flatten this Table. Each column with a struct type is flattened
into one column per struct field. Other columns are left unchanged.

combine_chunksdatasets.table.MemoryMappedTable.combine_chunkshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1091[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **memory_pool** (`MemoryPool` , defaults to `None`) --
For memory allocations, if required, otherwise use default pool.0`datasets.table.Table`

Make a new table by combining the chunks this table has.

All the underlying chunks in the ChunkedArray of each column are
concatenated into zero or one chunk.

castdatasets.table.MemoryMappedTable.casthttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1109[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **target_schema** (`Schema`) --
Schema to cast to, the names and order of fields must match.

- **safe** (`bool` , defaults to `True`) --
  Check for overflows or other unsafe conversions.0 `datasets.table.Table`

Cast table values to another schema

replace_schema_metadatadatasets.table.MemoryMappedTable.replace_schema_metadatahttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1126[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **metadata** (`dict` , defaults to `None`)
--0 `datasets.table.Table` shallow_copy

EXPERIMENTAL: Create shallow copy of table by replacing schema
key-value metadata with the indicated new metadata (which may be None,
which deletes any existing metadata.

add_columndatasets.table.MemoryMappedTable.add_columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1142[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`int`) --
Index to place the column at.

- **field_** (`Union[str, pyarrow.Field]`) --
  If a string is passed then the type is deduced from the column
  data.
- **column** (`Union[pyarrow.Array, List[pyarrow.Array]]`) --
  Column data.0 `datasets.table.Table` New table with the passed column added.

Add column to Table at position.

A new table is returned with the column added, the original table
object is left unchanged.

append_columndatasets.table.MemoryMappedTable.append_columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1165[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **field_** (`Union[str, pyarrow.Field]`) --
If a string is passed then the type is deduced from the column
data.

- **column** (`Union[pyarrow.Array, List[pyarrow.Array]]`) --
  Column data.0 `datasets.table.Table` New table with the passed column added.

Append column at end of columns.

remove_columndatasets.table.MemoryMappedTable.remove_columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1184[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`int`) --
Index of column to remove.0 `datasets.table.Table` New table without the column.

Create new Table with the indicated column removed.

set_columndatasets.table.MemoryMappedTable.set_columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1200[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`int`) --
Index to place the column at.

- **field_** (`Union[str, pyarrow.Field]`) --
  If a string is passed then the type is deduced from the column
  data.
- **column** (`Union[pyarrow.Array, List[pyarrow.Array]]`) --
  Column data.0 `datasets.table.Table` New table with the passed column set.

Replace column in Table at position.

rename_columnsdatasets.table.MemoryMappedTable.rename_columnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1221[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]

Create new table with columns renamed to provided names.

selectdatasets.table.MemoryMappedTable.selecthttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1248[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]-

columns (`Union[List[str], List[int]]`) --
The column names or integer indices to select.0datasets.table.TableNew table with the specified columns, and metadata preserved.

Select columns of the table.

Returns a new table with the specified columns, and metadata preserved.

dropdatasets.table.MemoryMappedTable.drophttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1229[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]-
columns (`List[str]`) --
List of field names referencing existing columns.0 `datasets.table.Table` New table without the columns.- `KeyError` -- : if any of the passed columns name are not existing. `KeyError`

Drop one or more columns and return a new table.

from_filedatasets.table.MemoryMappedTable.from_filehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1015[{"name": "filename", "val": ": str"}, {"name": "replays", "val": " = None"}]

ConcatenationTabledatasets.table.ConcatenationTable

class datasets.table.ConcatenationTabledatasets.table.ConcatenationTablehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1273[{"name": "table", "val": ": Table"}, {"name": "blocks", "val": ": list"}]

The table comes from the concatenation of several tables called blocks.
It enables concatenation on both axis 0 (append rows) and axis 1 (append columns).

The underlying tables are called "blocks" and can be either `InMemoryTable`
or `MemoryMappedTable` objects.
This allows to combine tables that come from memory or that are memory mapped.
When a `ConcatenationTable` is pickled, then each block is pickled:

- the `InMemoryTable` objects are pickled by copying all the data in memory.
- the MemoryMappedTable objects are pickled without copying the data into memory.

Instead, only the path to the memory mapped arrow file is pickled, as well as the list of transforms to "replays" when reloading the table from the disk.

Its implementation requires to store each block separately.
The `blocks` attributes stores a list of list of blocks.
The first axis concatenates the tables along the axis 0 (it appends rows),
while the second axis concatenates tables along the axis 1 (it appends columns).

If some columns are missing when concatenating on axis 0, they are filled with null values.
This is done using `pyarrow.concat_tables(tables, promote=True)`.

You can access the fully combined table by accessing the `ConcatenationTable.table` attribute, and the blocks by accessing the `ConcatenationTable.blocks` attribute.

validatedatasets.table.ConcatenationTable.validatehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L178[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **full** (`bool`, defaults to `False`) --
If `True`, run expensive checks, otherwise cheap checks only.0- `pa.lib.ArrowInvalid` -- if validation fails`pa.lib.ArrowInvalid`

Perform validation checks. An exception is raised if validation fails.

By default only cheap validation checks are run. Pass `full=True`
for thorough validation checks (potentially `O(n)`).

equalsdatasets.table.ConcatenationTable.equalshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L194[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **other** (Table) --
Table to compare against.

- **check_metadata** `bool`, defaults to `False`) --
  Whether schema metadata equality should be checked as well.0`bool`

Check if contents of two tables are equal.

to_batchesdatasets.table.ConcatenationTable.to_batcheshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L211[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **max_chunksize** (`int`, defaults to `None`) --
Maximum size for `RecordBatch` chunks. Individual chunks may be

smaller depending on the chunk layout of individual columns.0 `List[pyarrow.RecordBatch]`

Convert Table to list of (contiguous) `RecordBatch` objects.

to_pydictdatasets.table.ConcatenationTable.to_pydicthttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L225[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}] `dict`

Convert the Table to a `dict` or `OrderedDict`.

to_pandasdatasets.table.ConcatenationTable.to_pandashttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L243[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **memory_pool** (`MemoryPool` , defaults to `None`) --
Arrow MemoryPool to use for allocations. Uses the default memory
pool is not passed.

- **strings_to_categorical** (`bool` , defaults to `False`) --
  Encode string (UTF8) and binary types to `pandas.Categorical` .
- **categories** (`list` , defaults to `empty`) --
  List of fields that should be returned as `pandas.Categorical` . Only
  applies to table-like data structures.
- **zero_copy_only** (`bool` , defaults to `False`) --
  Raise an `ArrowException` if this function call would require copying
  the underlying data.
- **integer_object_nulls** (`bool` , defaults to `False`) --
  Cast integers with nulls to objects.
- **date_as_object** (`bool` , defaults to `True`) --
  Cast dates to objects. If `False` , convert to `datetime64[ns]` dtype.
- **timestamp_as_object** (`bool` , defaults to `False`) --
  Cast non-nanosecond timestamps (`np.datetime64`) to objects. This is
  useful if you have timestamps that don't fit in the normal date
  range of nanosecond timestamps (1678 CE-2262 CE).
  If `False` , all timestamps are converted to `datetime64[ns]` dtype.
- **use_threads** (`bool` , defaults to `True`) --
  Whether to parallelize the conversion using multiple threads.
- **deduplicate_objects** (`bool` , defaults to `False`) --
  Do not create multiple copies Python objects when created, to save

on memory use. Conversion will be slower.

- **ignore_metadata** (`bool` , defaults to `False`) --
  If `True` , do not use the 'pandas' metadata to reconstruct the DataFrame index, if present.
- **safe** (`bool` , defaults to `True`) --
  For certain data types, a cast is needed in order to store the data in a pandas DataFrame or Series (e.g. timestamps are always stored as nanoseconds in pandas). This option controls whether it is a safe cast or not.
- **split_blocks** (`bool` , defaults to `False`) --
  If `True` , generate one internal "block" for each column when creating a pandas.DataFrame from a `RecordBatch` or `Table` . While this can temporarily reduce memory note that various pandas operations can trigger "consolidation" which may balloon memory use.
- **self_destruct** (`bool` , defaults to `False`) --
  EXPERIMENTAL: If `True` , attempt to deallocate the originating Arrow memory while converting the Arrow object to pandas. If you use the object after calling `to_pandas` with this option it will crash your program.
- **types_mapper** (`function` , defaults to `None`) --
  A function mapping a pyarrow DataType to a pandas `ExtensionDtype` . This can be used to override the default pandas type for conversion of built-in pyarrow types or in absence of `pandas_metadata` in the Table schema. The function receives a pyarrow DataType and is expected to return a pandas `ExtensionDtype` or `None` if the default conversion should be used for that type. If you have a dictionary mapping, you can pass `dict.get` as function.0 `pandas.Series` or `pandas.DataFrame` `pandas.Series` or `pandas.DataFrame` depending on type of object

Convert to a pandas-compatible NumPy array or DataFrame, as appropriate.

to_stringdatasets.table.ConcatenationTable.to_stringhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L305[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]

fielddatasets.table.ConcatenationTable.fieldhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L324[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i**

(`Union[int, str]`) --
The index or name of the field to retrieve.0 `pyarrow.Field`

Select a schema field by its column name or numeric index.

columndatasets.table.ConcatenationTable.columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L337[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`Union[int, str]`) --
The index or name of the column to retrieve.0 `pyarrow.ChunkedArray`

Select a column by its column name, or numeric index.

itercolumnsdatasets.table.ConcatenationTable.itercolumnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L350[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}] `pyarrow.ChunkedArray`

Iterator over all columns in their numerical order.

schemadatasets.table.ConcatenationTable.schemahttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L359[] `pyarrow.Schema`

Schema of the table and its columns.

columnsdatasets.table.ConcatenationTable.columnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L369[] `List[pa.ChunkedArray]`

List of all columns in numerical order.

num_columnsdatasets.table.ConcatenationTable.num_columnshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L379[]int

Number of columns in this table.

num_rowsdatasets.table.ConcatenationTable.num_rowshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L389[]int

Number of rows in this table.

Due to the definition of a table, all columns have the same number of
rows.

shapedatasets.table.ConcatenationTable.shapehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L402[] `(int, int)` Number of rows and number of columns.

Dimensions of the table: (#rows, #columns).

nbytesdatasets.table.ConcatenationTable.nbyteshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L412[]

Total number of bytes consumed by the elements of the table.

column_namesdatasets.table.ConcatenationTable.column_nameshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L419[]

Names of the table's columns.

slicedatasets.table.ConcatenationTable.slicehttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1482[{"name": "offset", "val": " = 0"}, {"name": "length", "val": " = None"}]- **offset** (`int` , defaults to `0`) --
Offset from start of table to slice.

- **length** (`int` , defaults to `None`) --
  Length of slice (default is until end of table starting from offset).0 `datasets.table.Table`

Compute zero-copy slice of this Table.

filterdatasets.table.ConcatenationTable.filterhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1513[{"name": "mask", "val": ""}, {"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]

Select records from a Table. See `pyarrow.compute.filter` for full usage.

flattendatasets.table.ConcatenationTable.flattenhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1524[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **memory_pool** (`MemoryPool` , defaults to `None`) --
For memory allocations, if required, otherwise use default pool.0 `datasets.table.Table`

Flatten this Table. Each column with a struct type is flattened
into one column per struct field. Other columns are left unchanged.

combine_chunksdatasets.table.ConcatenationTable.combine_chunkshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1542[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **memory_pool** (`MemoryPool` , defaults to `None`) --
For memory allocations, if required, otherwise use default pool.0 `datasets.table.Table`

Make a new table by combining the chunks this table has.

All the underlying chunks in the `ChunkedArray` of each column are
concatenated into zero or one chunk.

castdatasets.table.ConcatenationTable.casthttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1562[{"name": "target_schema", "val": ""}, {"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **target_schema** (`Schema`) --
Schema to cast to, the names and order of fields must match.

- **safe** (`bool` , defaults to `True`) --
  Check for overflows or other unsafe conversions.0 `datasets.table.Table`

Cast table values to another schema.

replace_schema_metadatadatasets.table.ConcatenationTable.replace_schema_metadatahttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1593[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **metadata** (`dict` , defaults to `None`)
--0 `datasets.table.Table` shallow_copy

EXPERIMENTAL: Create shallow copy of table by replacing schema
key-value metadata with the indicated new metadata (which may be `None` ,
which deletes any existing metadata).

add_columndatasets.table.ConcatenationTable.add_columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1611[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`int`) --
Index to place the column at.

- **field_** (`Union[str, pyarrow.Field]`) --
  If a string is passed then the type is deduced from the column
  data.
- **column** (`Union[pyarrow.Array, List[pyarrow.Array]]`) --

Column data.0 `datasets.table.Table` New table with the passed column added.

Add column to Table at position.

A new table is returned with the column added, the original table
object is left unchanged.

append_columndatasets.table.ConcatenationTable.append_columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1632[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **field_** (`Union[str, pyarrow.Field]`) --
If a string is passed then the type is deduced from the column
data.

- **column** (`Union[pyarrow.Array, List[pyarrow.Array]]`) --
  Column data.0 `datasets.table.Table` New table with the passed column added.

Append column at end of columns.

remove_columndatasets.table.ConcatenationTable.remove_columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1649[{"name": "i", "val": ""}, {"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`int`) --
Index of column to remove.0 `datasets.table.Table` New table without the column.

Create new Table with the indicated column removed.

set_columndatasets.table.ConcatenationTable.set_columnhttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1673[{"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **i** (`int`) --
Index to place the column at.

- **field_** (`Union[str, pyarrow.Field]`) --
  If a string is passed then the type is deduced from the column
  data.
- **column** (`Union[pyarrow.Array, List[pyarrow.Array]]`) --
  Column data.0 `datasets.table.Table` New table with the passed column set.

Replace column in Table at position.

rename_columnsdatasets.table.ConcatenationTable.rename_columnshttps://github.com/

huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1692[{"name": "names", "val": ""}, {"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]

Create new table with columns renamed to provided names.

selectdatasets.table.ConcatenationTable.selecthttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1726[{"name": "columns", "val": ""}, {"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **columns** (`Union[List[str], List[int]]`) --
The column names or integer indices to select.0datasets.table.TableNew table with the specified columns, and metadata preserved.

Select columns of the table.

Returns a new table with the specified columns, and metadata preserved.

dropdatasets.table.ConcatenationTable.drophttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1705[{"name": "columns", "val": ""}, {"name": "*args", "val": ""}, {"name": "**kwargs", "val": ""}]- **columns** (`List[str]`) --
List of field names referencing existing columns.0 `datasets.table.Table` New table without the columns.- `KeyError` -- : if any of the passed columns name are not existing. `KeyError`

Drop one or more columns and return a new table.

from_blocksdatasets.table.ConcatenationTable.from_blockshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1378[{"name": "blocks", "val": ": ~TableBlockContainer"}]

from_tablesdatasets.table.ConcatenationTable.from_tableshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1392[{"name": "tables", "val": ": list"}, {"name": "axis", "val": ": int = 0"}]- **tables** (list of `Table` or list of `pyarrow.Table`) --
List of tables.

- **axis** (`{0, 1}` , defaults to `0` , meaning over rows) --
  Axis to concatenate over, where `0` means over rows (vertically) and `1` means over columns
  (horizontally).
  0
  Create `ConcatenationTable` from list of tables.

Utilsdatasets.table.concat_tables

datasets.table.concat_tablesdatasets.table.concat_tableshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1746[{"name": "tables", "val": ": list"}, {"name": "axis", "val": ": int = 0"}]- **tables** (list of `Table`) --
List of tables to be concatenated.

- **axis** (`{0, 1}`, defaults to `0`, meaning over rows) --
  Axis to concatenate over, where `0` means over rows (vertically) and `1` means over columns
  (horizontally).
  0`datasets.table.Table`If the number of input tables is > 1, then the returned table is a `datasets.table.ConcatenationTable`.
  Otherwise if there's only one table, it is returned as is.

Concatenate tables.

datasets.table.list_table_cache_filesdatasets.table.list_table_cache_fileshttps://github.com/huggingface/datasets/blob/4.2.0/src/datasets/table.py#L1769[{"name": "table", "val": ": Table"}]`List[str]`A list of paths to the cache files loaded by the table.

Get the cache files that are loaded by the table.
Cache file are used when parts of the table come from the disk via memory mapping.