pantab

Release 2.1.1

Apr 14, 2022

Contents

1 Usage Examples **1**
- 1.1 Writing to a Hyper Extract . . . 1
- 1.2 Reading a Hyper Extract . . . 2
- 1.3 Working with Schemas . . . 2
- 1.4 Reading and Writing Multiple Tables . . . 2
- 1.5 Appending Data to Existing Tables . . . 3
- 1.6 Issuing SQL queries . . . 3
- 1.7 Providing your own HyperProcess . . . 4
- 1.8 Providing your own Hyper Connection . . . 5

2 Usage Notes **7**
- 2.1 Type Mapping . . . 7
- 2.2 Index / Column Handling . . . 8
- 2.3 Datetime Timezone Handling . . . 8
- 2.4 Timedelta Components . . . 8

3 API Reference **9**

4 Changelog **11**
- 4.1 Pantab 2.1.1 (2022-04-13) . . . 11
- 4.2 Pantab 2.1.0 (2021-07-02) . . . 11
- 4.3 Pantab 2.0.0 (2021-04-15) . . . 11
- 4.4 Pantab 1.1.1 (2020-11-02) . . . 12
- 4.5 Pantab 1.1.0 (2020-04-30) . . . 12
- 4.6 Pantab 1.0.1 (2020-02-03) . . . 12
- 4.7 Pantab 1.0.0 (2020-01-15) . . . 12
- 4.8 Pantab 0.2.3 (2020-01-02) . . . 13
- 4.9 Pantab 0.2.2 (2019-12-25) . . . 13
- 4.10 Pantab 0.2.1 (2019-12-23) . . . 13
- 4.11 Pantab 0.2.0 (2019-12-19) . . . 13
- 4.12 0.1.1 (2019-12-06) . . . 14
- 4.13 0.1.0 (2019-11-29) . . . 14
- 4.14 0.0.1.b5 (2019-11-05) . . . 14
- 4.15 0.0.1.b4 (2019-11-05) . . . 14
- 4.16 0.0.1.b3 (2019-11-01) . . . 14

5 Support **17**

6 What is it? 19

7 How do I get it? 21

8 Why should I use it? 23

Index 27

CHAPTER 1

Usage Examples

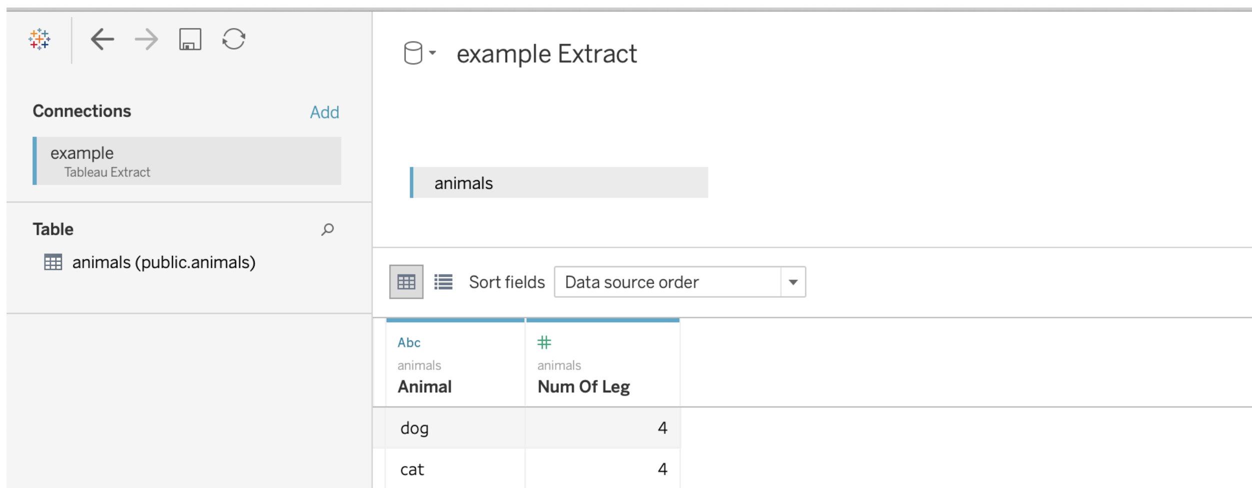
1.1 Writing to a Hyper Extract

```
import pandas as pd
import pantab

df = pd.DataFrame([
    ["dog", 4],
    ["cat", 4],
], columns=["animal", "num_of_legs"])

pantab.frame_to_hyper(df, "example.hyper", table="animals")
```

The above example will write out to a file named "example.hyper", which Tableau can then report off of.



1.2 Reading a Hyper Extract

```
import pantab

df = pantab.frame_from_hyper("example.hyper", table="animals")
print(df)
```

1.3 Working with Schemas

By default tables will be written to the "public" schema. You can control this behavior however by specifying a `tableauhyperapi.TableName` when reading / writing extracts.

```
import pandas as pd
import pantab
from tableauhyperapi import TableName

# Let's write somewhere besides the default public schema
table = TableName("not_the_public_schema", "a_table")

df = pd.DataFrame([
    ["dog", 4],
    ["cat", 4],
], columns=["animal", "num_of_legs"])

pantab.frame_to_hyper(df, "example.hyper", table=table)

# Can also be round-tripped
df2 = pantab.frame_from_hyper("example.hyper", table=table)
```

Note: If you want to publish a hyper file using the Tableau Server REST API and you're using using a version prior to 2020.1 you'll need to have a single table named `Extract` that uses the `Extract` schema (`Extract.Extract`).

1.4 Reading and Writing Multiple Tables

`frames_to_hyper` and `frames_from_hyper` can write and return a dictionary of DataFrames for Hyper extract, respectively.

```
import pandas as pd
import pantab
from tableauhyperapi import TableName

dict_of_frames = {
    "table1": pd.DataFrame([[1, 2]], columns=list("ab")),
    TableName("non_public_schema", "table2"): pd.DataFrame([[3, 4]], columns=list("cd
↪")),
}

pantab.frames_to_hyper(dict_of_frames, "example.hyper")
```

(continues on next page)

(continued from previous page)

```
# Can also be round-tripped
result = pantab.frames_from_hyper("example.hyper")
```

Note: While you can write using `str`, `tableauhyperapi.Name` or `tableauhyperapi.TableName` instances, the keys of the dict returned by `frames_from_hyper` will always be `tableauhyperapi.TableName` instances

1.5 Appending Data to Existing Tables

By default, `frame_to_hyper` and `frames_to_hyper` will fully drop and reloaded targeted tables. However, you can also append records to existing tables by supplying `table_mode="a"` as a keyword argument.

```
import pandas as pd
import pantab

df = pd.DataFrame([
    ["dog", 4],
    ["cat", 4],
], columns=["animal", "num_of_legs"])

pantab.frame_to_hyper(df, "example.hyper", table="animals")

new_data = pd.DataFrame([["moose", 4]], columns=["animal", "num_of_legs"])

# Instead of overwriting the animals table, we can append via table_mode
pantab.frame_to_hyper(df, "example.hyper", table="animals", table_mode="a")
```

Please note that `table_mode="a"` will create the table(s) if they do not already exist.

1.6 Issuing SQL queries

New in version 2.0.

With `frame_from_hyper_query`, one can execute SQL queries against a Hyper file and retrieve the resulting data as a DataFrame. This can be used, e.g. to retrieve only a part of the data (using a `WHERE` clause) or to offload computations to Hyper.

```
import pandas as pd
import pantab

df = pd.DataFrame([
    ["dog", 4],
    ["cat", 4],
    ["moose", 4],
    ["centipede", 100],
], columns=["animal", "num_of_legs"])

pantab.frame_to_hyper(df, "example.hyper", table="animals")

# Read a subset of the data from the Hyper file
```

(continues on next page)

(continued from previous page)

```
query = """
SELECT animal
FROM animals
WHERE num_of_legs > 4
"""
df = pantab.frame_from_hyper_query("example.hyper", query)
print(df)

# Let Hyper do an aggregation for us - it could also do joins, window queries, ...
query = """
SELECT num_of_legs, COUNT(*)
FROM animals
GROUP BY num_of_legs
"""
df = pantab.frame_from_hyper_query("example.hyper", query)
print(df)
```

1.7 Providing your own HyperProcess

New in version 2.0.

For convenience, pantab's functions internally spawn a HyperProcess. In case you prefer to spawn your own `HyperProcess`, you can supply it to pantab through the `hyper_process` keyword argument.

By using your own `HyperProcess`, you have full control over all its startup paramters. In the following example we use that flexibility to:

- enable telemetry, thereby making sure the Hyper team at Tableau knows about our use case and potential issues we might be facing
- disable log files, as we operate in some environment with really tight disk space
- opt-in to the new Hyper file format

By reusing the same `HyperProcess` for multiple operations, we also save a few milliseconds. While not noteworthy in this simple example, this might be a good optimization in case you call `frame_to_hyper` repeatedly in a loop.

```
import pandas as pd
import pantab
from tableauhyperapi import HyperProcess, Telemetry

df = pd.DataFrame([
    ["dog", 4],
    ["cat", 4],
], columns=["animal", "num_of_legs"])

parameters = {"log_config": "", "default_database_version": "1"}
with HyperProcess(Telemetry.SEND_USAGE_DATA_TO_TABLEAU, parameters=parameters) as␣
↪hyper:
    # Insert some initial data
    pantab.frame_to_hyper(df, "example.hyper", table="animals", hyper_process=hyper)

    # Append additional data to the same table using `table_mode="a"`
    new_data = pd.DataFrame([["moose", 4]], columns=["animal", "num_of_legs"])
    pantab.frame_to_hyper(df, "example.hyper", table="animals", table_mode="a", hyper_
↪process=hyper)
```

1.8 Providing your own Hyper Connection

New in version 2.0.

In order to interface with Hyper, pantab functions need a HyperAPI Connection to interface with Hyper. For convenience, pantab creates those connections implicitly for you. However, establishing a connection is not for free, and by reusing the same `Connection` for multiple operations, we can save time. Hence, pantab also allows you to pass in a HyperAPI connection instead of the name / location of your Hyper file.

```
import pandas as pd
import pantab
from tableauhyperapi import HyperProcess, Telemetry, Connection, CreateMode

df = pd.DataFrame([
    ["dog", 4],
    ["cat", 4],
    ["centipede", 100],
], columns=["animal", "num_of_legs"])
path = "example.hyper"

with HyperProcess(Telemetry.DO_NOT_SEND_USAGE_DATA_TO_TABLEAU) as hyper:
    pantab.frames_to_hyper({"animals": df}, path, hyper_process=hyper)

    with Connection(hyper.endpoint, path, CreateMode.NONE) as connection:
        query = """
        SELECT animal
        FROM animals
        WHERE num_of_legs > 4
        """
        many_legs_df = pantab.frame_from_hyper_query(connection, query)
        print(many_legs_df)

        all_animals = pantab.frame_from_hyper_query(connection, table="animals")
        print(all_animals)
```

CHAPTER 2

Usage Notes

2.1 Type Mapping

pantab maps the following dtypes from pandas to the equivalent column type in Tableau.

Dtype	Tableau Type	Nullability
int16	SMALLINT	NOT NULLABLE
int32	INT	NOT NULLABLE
int64	BIGINT	NOT NULLABLE
Int16	SMALLINT	NULLABLE
Int32	INT	NULLABLE
Int64	BIGINT	NULLABLE
float32	DOUBLE	NULLABLE
float64	DOUBLE	NULLABLE
Float32	DOUBLE	NULLABLE
Float64	DOUBLE	NULLABLE
bool	BOOL	NOT NULLABLE
boolean	BOOL	NULLABLE
datetime64[ns]	TIMESTAMP	NULLABLE
datetime64[ns, UTC]	TIMESTAMP_WITH_TZ	Nullable
timedelta64[ns]	INTERVAL	NULLABLE
object	TEXT	NULLABLE
string	TEXT	NULLABLE

Any dtype not explicitly listed in the above table will raise a ValueError if trying to write out data.

When reading data through `frame_from_hyper_query`, pantab will always assume that all result columns are nullable.

New in version 1.0.0: If using pandas 1.0 and above, text data will be read back into a `string` dtype rather than an `object` dtype.

Note: Most objects can maintain their type when "round-tripping" to/from Hyper extracts, with the exception of the float32 object as only DOUBLE is available for floating point storage in Hyper. After pandas 1.0 / pantab 1.0, object dtypes written will be read back in as string. Also, reading data back through `frame_from_hyper_query` will likely read back different dtypes, as `frame_from_hyper_query` assumes all columns to be nullable.

2.2 Index / Column Handling

A pandas DataFrame always comes with an `Index` used to slice / access rows. No such concept exists in Tableau, so this will be implicitly dropped when writing Hyper extracts.

With respect to columns, note that Tableau stores column labels internally as a string. You *may* be able to write non-string objects to the database (this is left to the underlying Hyper API to decide) but reading those objects back is not a lossless operation and will always return strings.

2.3 Datetime Timezone Handling

Timezones are not supported in Hyper Extracts. Attempting to write a timezone aware array to an extract will result in an error that the dtype is not supported. The only option to write dates with timezone information would be to make the data timezone naive. You may also consider storing the timezone in a separate column as part of the extract to avoid losing information.

2.4 Timedelta Components

The `pd.Timedelta` and the `Interval` exposed by the Hyper API have similar but different storage mechanisms that may cause inconsistencies. Specifically, a *pd.Timedelta* does not have a month component to it, so reading `Interval` objects from a Hyper Extract that have this component will raise a `TypeError`. The Hyper API's `Interval` only offers storage of days and microseconds (aside from months). pantab will convert hours, minutes, seconds, etc… into microseconds for you, but reading that information back from a Hyper extract is lossy and will only provide back the microsecond storage.

CHAPTER 3

API Reference

frame_to_hyper(*df: pd.DataFrame, database: Union[str, pathlib.Path], *, table: Union[str, tableauhyperapi.Name, tableauhyperapi.TableName], hyper_process: Optional[HyperProcess]*) → None:

Convert a DataFrame to a .hyper extract.

Parameters

- **df** – Data to be written out.
- **database** – Name / location of the Hyper file to write to.
- **table** – Table to write to. Must be supplied as a keyword argument.
- **hyper_process** – A *HyperProcess* in case you want to spawn it by yourself. Optional. Must be supplied as a keyword argument.

frame_from_hyper(*source: Union[str, pathlib.Path, tab_api.Connection], *, table: Union[str, tableauhyperapi.Name, tableauhyperapi.TableName], hyper_process: Optional[HyperProcess], use_float_na: bool = False*) → pd.DataFrame:

Extracts a DataFrame from a .hyper extract.

Parameters

- **source** – Name / location of the Hyper file to be read or Hyper-API connection.
- **table** – Table to read. Must be supplied as a keyword argument.
- **hyper_process** – A *HyperProcess* in case you want to spawn it by yourself. Optional. Must be supplied as a keyword argument.
- **use_float_na** – Flag indicating whether to use the pandas *Float32*/*Float64* dtypes which support the new pandas missing value *pd.NA*, default False

Return type pd.DataFrame

frames_to_hyper(*dict_of_frames: Dict[Union[str, tableauhyperapi.Name, tableauhyperapi.TableName], pd.DataFrame], database: Union[str, pathlib.Path], *, hyper_process: Optional[HyperProcess]*) → None:

Writes multiple DataFrames to a .hyper extract.

Parameters

- **dict_of_frames** – A dictionary whose keys are valid table identifiers and values are dataframes
- **database** – Name / location of the Hyper file to write to.
- **hyper_process** – A *HyperProcess* in case you want to spawn it by yourself. Optional. Must be supplied as a keyword argument.

frames_from_hyper(*source: Union[str, pathlib.Path, tab_api.Connection], *, hyper_process: Optional[HyperProcess]*) → Dict[tableauhyperapi.TableName, pd.DataFrame, use_float_na: bool = False]:

Extracts tables from a .hyper extract.

Parameters

- **source** – Name / location of the Hyper file to be read or Hyper-API connection.
- **hyper_process** – A *HyperProcess* in case you want to spawn it by yourself. Optional. Must be supplied as a keyword argument.
- **use_float_na** – Flag indicating whether to use the pandas *Float32*/*Float64* dtypes which support the new pandas missing value *pd.NA*, default False

Return type Dict[tableauhyperapi.TableName, pd.DataFrame]

frame_from_hyper_query(*source: Union[str, pathlib.Path, tab_api.Connection], query: str, *, hyper_process: Optional[HyperProcess], use_float_na: bool = False*) → pd.DataFrame:

New in version 2.0: Executes a SQL query and returns the result as a pandas dataframe

param source Name / location of the Hyper file to be read or Hyper-API connection.

param query SQL query to execute.

param hyper_process A *HyperProcess* in case you want to spawn it by yourself. Optional. Must be supplied as a keyword argument.

param use_float_na Flag indicating whether to use the pandas *Float32*/*Float64* dtypes which support the new pandas missing value *pd.NA*, default False

rtype Dict[tableauhyperapi.TableName, pd.DataFrame]

CHAPTER 4

Changelog

4.1 Pantab 2.1.1 (2022-04-13)

- Fixed a memory leak with `frame_to_hyper`
- Fixed issue where `pantab.__version__` was misreporting the version string

4.2 Pantab 2.1.0 (2021-07-02)

Special thanks to Caleb Overman for contributing to this release.

4.2.1 Enhancments

- A new `use_float_na` parameter has been added to reading functions, which will convert doubles from Hyper files to the pandas `Float64` Extension dtype rather than using the standard numpy float dtype (#131)
- Writing `Float32` and `Float64` dtypes is now supported (#131)
- Writing to a Hyper file is now up to 50% faster (#132)

4.3 Pantab 2.0.0 (2021-04-15)

Special thanks to Adrian Vogelsgesang for contributing to this release.

4.3.1 API Breaking Changes

- Users may now pass an existing connection as the first argument to pantab's read functions. As part of this, the first argument was renamed from `database` to `source` (#123)

4.3.2 Enhancements

- Added support for Python 3.9 while dropping support for 3.6 (#122)
- A new `frame_from_hyper_query` method has been added, providing support for executing SQL statements against a Hyper file (#118)
- Users may now create their own Hyper process and pass it as an argument to the reading and writing functions (#39, #51)
- The value 0001-01-01 will no longer be read as a NULL timestamp (#121)

4.4 Pantab 1.1.1 (2020-11-02)

4.4.1 Bugfixes

- Fixed issue where pantab would throw `TypeError: Column "COLUMN_NAME" has unsupported datatype TEXT` when reading Non-Nullable string columns from Hyper (#111)

4.5 Pantab 1.1.0 (2020-04-30)

Special thanks to Adrian Vogelsgesang for contributing to this release.

4.5.1 Features

- Added support for reading Hyper DATE columns as datetime64 objects in pandas (#94)

4.5.2 Bugfixes

- Fixed issue where Python would crash instead of throwing an error when reading invalid records from a Hyper file (#77)
- Fixed ImportError when building from source with tableauhyperapi versions 0.0.10309 and greater (#88)
- Attempting to read a Hyper extract with unsupported data types will now raise a `TypeError` (#92)

4.6 Pantab 1.0.1 (2020-02-03)

4.6.1 Features

- pantab will not automatically install the tableauhyperapi as a dependency when installing via pip (#83)
- Pre-built wheels for manylinux configurations are now available. (#84)

4.7 Pantab 1.0.0 (2020-01-15)

Special thanks to chillerno1 for contributing to this release.

4.7.1 Features

- pantab now supports reading/writing pandas 1.0 dtypes, namely the `boolean` and `string` dtypes. (#20)

Important: TEXT data read from a Hyper extract will be stored in a `string` dtype when using pandas 1.0 or greater in combination with pantab 1.0 or greater. Older versions of either tool will read the data back into a `object` dtype.

4.7.2 Bugfixes

- Fixed potential segfault on systems where not all addresses can be expressed in an unsigned long long. (#52)

4.8 Pantab 0.2.3 (2020-01-02)

4.8.1 Bugfixes

- Fixed issue where dates would roundtrip in pantab find but would either error or be incorrect in Tableau Desktop (#66)

4.9 Pantab 0.2.2 (2019-12-25)

4.9.1 Bugfixes

- Pantab now writes actual NULL values for datetime columns, rather than 0001-01-01 00:00:00 (#60)

4.10 Pantab 0.2.1 (2019-12-23)

4.10.1 Bugfixes

- Fixed issue where reading a datetime column containing `pd.NaT` values would throw an `OutOfBoundsDatetime` error (#56)
- Fixed issue where reading a timedelta column containing `pd.NaT` would throw a `ValueError` (#57)

4.11 Pantab 0.2.0 (2019-12-19)

4.11.1 Features

- Improved performance when reading data from Hyper extracts (#34)

4.12 0.1.1 (2019-12-06)

A special *thank you* goes out to the following contributors leading up to this release:

- chillerno1
- cedricyau

4.12.1 Bugfixes

- Fixed issue where source installations would error with *fatal error: tableauhyperapi.h: No such file or directory* (#40)

4.13 0.1.0 (2019-11-29)

pantab is officially out of beta! Thanks for all of the feedback and support of the tool so far.

Special thanks to Adrian Vogelsgesang and Jan Finis at Tableau, who offered guidance and feedback on performance improvements in this release.

- Improved error messaging when attempting to write invalid data. (#19)
- Write-performance of Hyper extracts has been drastically improved for larger datasets. (#31)
- Less memory is now required to write DataFrames to the Hyper format. (#33)

4.14 0.0.1.b5 (2019-11-05)

4.14.1 Bugfixes

- Fixed issue where failures during append mode (`table_mode="a"`) would delete original Hyper file. (#17)

4.15 0.0.1.b4 (2019-11-05)

4.15.1 Features

- frame_to_hyper and frames_to_hyper now support a table_mode keyword argument. `table_mode="a"` will append data to existing tables, or create them if they do not exist. The default operation of `table_mode="w"` will continue to fully drop / reload tables. (#14)

4.16 0.0.1.b3 (2019-11-01)

4.16.1 Features

- Added support for nullable integer types (i.e. the "Int*" types in pandas). Current integer types will now show as NOT_NULLABLE in Hyper extracts. (#7)
- Added support for reading / writing UTC timestamps, rather than only timezone-naive. (#8)

4.16.2 Bugfixes

- Fixed issue where certain versions of pantab in combination with certain versions of the Hyper API would throw "TypeError: __init__() got an unexpected keyword argument 'name'" when generating Hyper extracts. (#10)

CHAPTER 5

Support

pantab was built and is maintained by innobi, a Tableau Technology Partner. pantab is and *always will be* free.

In addition to free tools, innobi offers all of the following to corporate enterprises:

- Consulting Services for Integrating Tableau and Python
- Python and Tableau Training
- Dedicated support and maintenance contracts for open source tools

Contact sales@innobi.io for more details.

Important: pantab is currently incompatible with tableauhyperapi>=0.0.14567. For details and status updates see GitHub issue #145

CHAPTER 6

What is it?

`pantab` is a Python wrapper around Tableau's Hyper API which promotes usage of the pandas DataFrame to seamlessly generate and read hyper extracts.

CHAPTER 7

How do I get it?

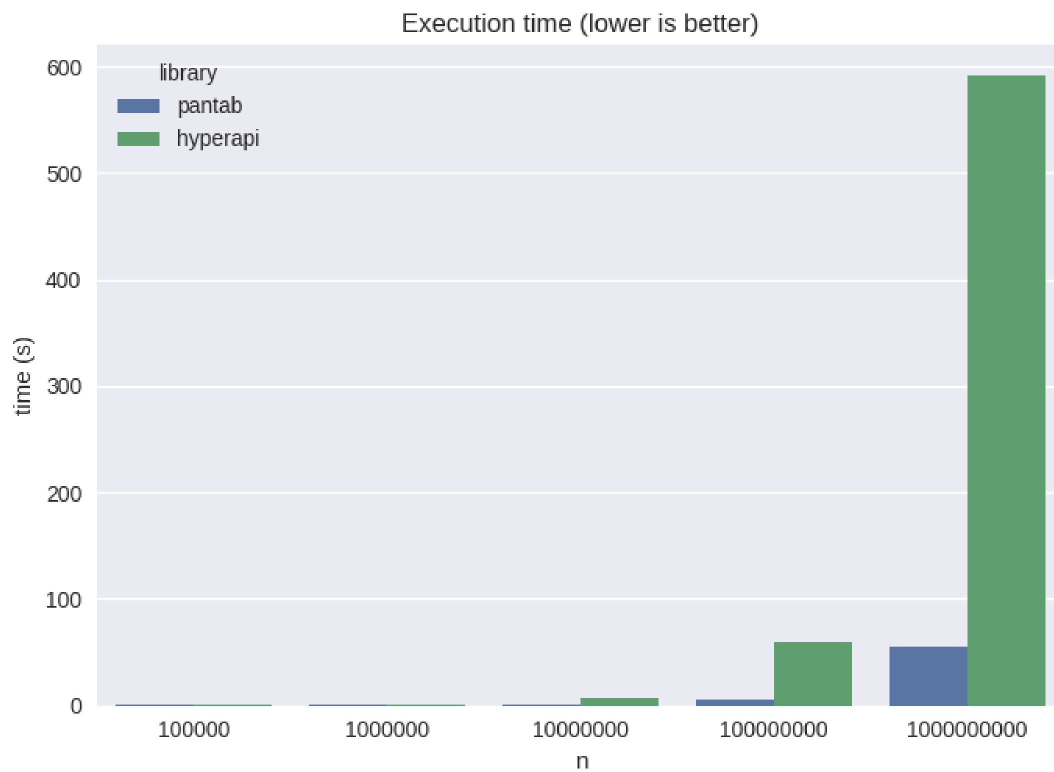
`pantab` requires Python 3.6+ and can run on any Python-supported OS. Installation is as easy as:

```
python -m pip install pantab
```

CHAPTER 8

Why should I use it?

`pantab` is faster than what you would write by hand, in less lines of code. Consider the below benchmark:



This was generated from the output of the below code on Manjaro Linux with 4.6 GHz CPU and 16 GB of memory.

```python
import itertools
import time

import numpy as np
import pandas as pd
import pantab
from tableauhyperapi import (
    Connection,
    CreateMode,
    HyperProcess,
    Inserter,
    SqlType,
    TableDefinition,
    TableName,
    Telemetry,
)


def write_via_pantab(proc: HyperProcess, n: int):
    df = pd.DataFrame({"column": np.ones(n, dtype=np.int16)})
    pantab.frame_to_hyper(df, "example.hyper", table="table", hyper_process=proc)


def write_via_hyperapi(proc: HyperProcess, n: int):
    data_to_insert = np.ones(n, dtype=np.int16)

    table = TableDefinition(
        table_name=TableName("table"),
        columns=[TableDefinition.Column(name="column", type=SqlType.int())],
    )

    with Connection(
        endpoint=proc.endpoint,
        database="example.hyper",
        create_mode=CreateMode.CREATE_AND_REPLACE,
    ) as conn:
        conn.catalog.create_table(table)

        with Inserter(conn, table) as inserter:
            for data in data_to_insert:
                inserter.add_row([data])
            inserter.execute()


ns = (100_000, 1_000_000, 10_000_000, 100_000_000, 1_000_000_000)
funcs = (write_via_pantab, write_via_hyperapi)
for n, func in itertools.product(ns, funcs):
    with HyperProcess(telemetry=Telemetry.DO_NOT_SEND_USAGE_DATA_TO_TABLEAU) as hyper:
        start = time.time()
        func(hyper, n)
        end = time.time()
        print(
            f"Total seconds for function {func.__name__} with {n} rows: {end - start}"
        )
```

Running this script may yield the below

```
Total seconds for function write_via_pantab with 100_000 rows: 0.026696205139160156
Total seconds for function write_via_hyperapi with 100000 rows: 0.08622312545776367
Total seconds for function write_via_pantab with 1000000 rows: 0.10160708427429199
Total seconds for function write_via_hyperapi with 1000000 rows: 0.6384274959564209
Total seconds for function write_via_pantab with 10000000 rows: 0.7043006420135498
Total seconds for function write_via_hyperapi with 10000000 rows: 6.226941108703613
Total seconds for function write_via_pantab with 100_000_000 rows: 5.718722820281982
Total seconds for function write_via_hyperapi with 100000000 rows: 59.03396558761597
Total seconds for function write_via_pantab with 1000000000 rows: 54.43922758102417
Total seconds for function write_via_hyperapi with 1000000000 rows: 591.2542216777802
```

Index

F

frame_from_hyper() (*built-in function*), 9
frame_from_hyper_query() (*built-in function*), 10
frame_to_hyper() (*built-in function*), 9
frames_from_hyper() (*built-in function*), 10
frames_to_hyper() (*built-in function*), 9