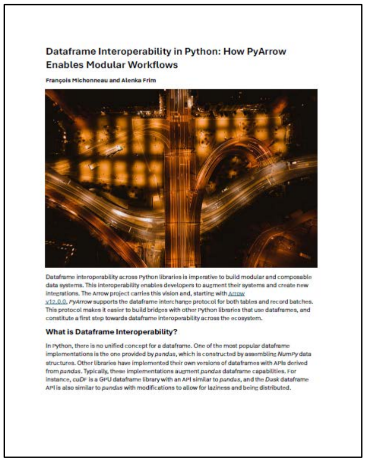
VOLTRON DATA

carahsoft.


Dataframe Interoperability in Python: How PyArrow Enables Modular Workflows

François Michonneau and Alenka Frim

Dataframe interoperability across Python libraries is imperative to build modular and composable data systems. This interoperability enables developers to augment their systems and create new integrations. The Arrow project carries this vision and, starting with Arrow v12.0.0, *PyArrow* supports the dataframe interchange protocol for both tables and record batches. This protocol makes it easier to build bridges with other Python libraries that use dataframes, and constitute a first step towards dataframe interoperability across the ecosystem.

What is Dataframe Interoperability?

In Python, there is no unified concept for a dataframe. One of the most popular dataframe implementations is the one provided by *pandas*, which is constructed by assembling *NumPy* data structures. Other libraries have implemented their own versions of dataframes with APIs derived from *pandas*. Typically, these implementations augment *pandas* dataframe capabilities. For instance, *cuDF* is a GPU dataframe library with an API similar to *pandas*, and the *Dask* dataframe API is also similar to *pandas* with modifications to allow for laziness and being distributed.

5 Ⅰ·ű Ⅰ·ŦNJⅠ·ƳŜ LyʃűŜNJƻLJŜNJⅠ·ōʎfʎűȅ ʎʃ
tȅűKƻyƳ Ⅰƻǿ tȅ!NJNJƻǿ 9yⅠ·ōfŜǎ
a ƻŘdzfⅠ·NJ ² ƻNJʅŦfƻǿǎ

Thank you for downloading this Voltron Data resource. Carahsoft is the distributor for Voltron Data's AI and ML solutions available via NASA SEWP V, ITES-SW2, NASPO ValuePoint, and many more contract vehicles.

To learn how to take the next step toward acquiring Voltron Data's solutions, please check out the following resources and information:

For Voltron Data overview:
carah.io/Voltron-Data

For upcoming events:
carah.io/Voltron-Events

For additional resources:
carah.io/Voltron-Resources

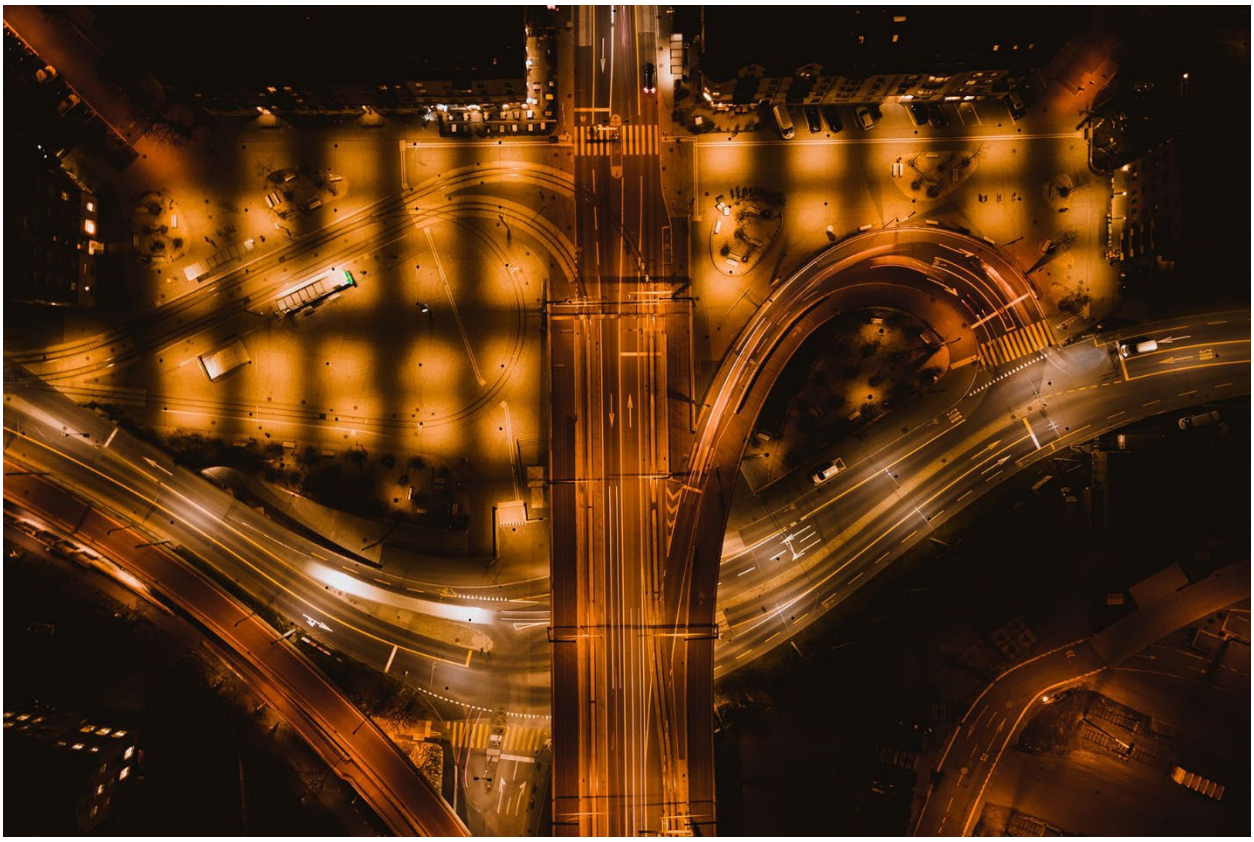
For additional Artificial Intelligence:
carah.io/ai-solutions

To set up a meeting:
VoltronData@carahsoft.com

To purchase, check out the contract vehicles available for procurement:
carah.io/procurement

Dataframe Interoperability in Python: How PyArrow Enables Modular Workflows

François Michonneau and Alenka Frim



Dataframe interoperability across Python libraries is imperative to build modular and composable data systems. This interoperability enables developers to augment their systems and create new integrations. The Arrow project carries this vision and, starting with Arrow v12.0.0, *PyArrow* supports the dataframe interchange protocol for both tables and record batches. This protocol makes it easier to build bridges with other Python libraries that use dataframes, and constitute a first step towards dataframe interoperability across the ecosystem.

What is Dataframe Interoperability?

In Python, there is no unified concept for a dataframe. One of the most popular dataframe implementations is the one provided by *pandas*, which is constructed by assembling *NumPy* data structures. Other libraries have implemented their own versions of dataframes with APIs derived from *pandas*. Typically, these implementations augment *pandas* dataframe capabilities. For instance, *cuDF* is a GPU dataframe library with an API similar to *pandas*, and the *Dask* dataframe API is also similar to *pandas* with modifications to allow for laziness and being distributed.

While the Arrow specification could be used to standardize how these dataframes are represented in memory, some libraries have developed APIs on top of these data structures which cannot be standardized by relying on Arrow alone.

Developing workflows or tools that take advantage of the features that each of these dataframe implementations provide is challenging: they all have different APIs, and there is no standard way of converting from one implementation into another. Developers need to write ad-hoc code to convert the data structures that each of these libraries provide.

To address this challenge, a consortium was established to create standards for dataframe interoperability. Dataframe interoperability means that you can pass your dataframe from one library to the next without having to worry about conversions. The conversion is taken care of internally by the libraries by relying on the predictability of the interchange dataframe structure.

The Dataframe Interchange Protocol

The first step towards interoperability is the development of the dataframe interchange protocol. Its main goal is to make it easier to pass dataframes across libraries. It also allows for the inspection of basic properties (e.g., number of columns, names of the columns, column data types).

The interchange protocol is implemented in the form of the __dataframe__ function on a given dataframe implementation within a library. This function defines the interchange objects and allows the consumer to construct the dataframe from these objects. The consumer library can then define the from_dataframe() function that uses these interchange objects to assemble its own dataframe.

If you are familiar with it, you might have realized that this dataframe interchange protocol is similar to the Arrow C Data Interface. The consortium considered using the Arrow C Data Interface for its standard, but some of its designs didn't align with everyone's needs. Writing the protocol in Python means that its implementation is directly accessible to Python developers without having to rely on another language.

The Example of the *PyArrow* Implementation

The dataframe interchange protocol is implemented for both *PyArrow* tables (pa.Table) and record batches (pa.RecordBatch). When converting dataframes from other libraries the from_dataframe function will always return a *PyArrow* table. Following the current specifications of the protocol, the *PyArrow* implementation supports the primitive data types and the dictionary type, as well as missing values. Nested data is however not included in the specifications.

You can find more details about the implementation of the dataframe interchange protocol in the *PyArrow* documentation.

Which Libraries Support the Dataframe Interchange Protocol?

In addition to *PyArrow*, libraries that currently support the Dataframe Interchange Protocol for their objects are: *Ibis*, *pandas*, *Modin*, *Vaex*, *CuDF*, and *Polars*. *Vega-Altair* (since 5.0.0) supports the protocol. It is also available in the development version of *Plotly* and is on the roadmap for *Seaborn*.

The Dataframe Interchange Protocol in Action

Having support for the Dataframe interchange protocol in visualization libraries means that you can wrangle your data with *PyArrow* and pass the resulting table directly to one of the visualization libraries that has implemented the protocol — without having to worry about the data conversion yourself.

To demonstrate this, we will use the PUMS dataset[1] and compute the average family income per state. First, we use *PyArrow* to read our PUMS stored as partitioned Parquet files and compute the average Family income (”FINCP”) per state (”ST”):

```python
import pyarrow.dataset as ds

h21 = ds.dataset(
    "/home/francois/datasets/pums/2021-5year-household/",
    format="parquet",
    partitioning="hive",
)
inc_mean = h21.to_table().group_by("ST").aggregate([("FINCP", "mean")])
```

In the PUMS dataset, the states are encoded numerically. To make the output more readable, we will join a table to our result that contains the full and abbreviated state names that are available online. To be able to read CSV files in Arrow hosted on a website, we can use the *fsspec* library:

```python
import fsspec.implementations.http

http = fsspec.implementations.http.HTTPFileSystem()

states = ds.dataset(
    "https://raw.githubusercontent.com/voltrondata-labs/2023-jonthebeach-ibis/main/data/pums_states.cs
    format="csv",
    filesystem=http,
    schema=pa.schema(
        [("pums_code", pa.int32()), ("state_name", pa.string()), ("state", pa.string())]
    ),
)

inc_mean = inc_mean.rename_columns(["pums_code", "FINCP_mean"])
res = inc_mean.join(states, keys = "pums_code")
```

We can now pass this *PyArrow* table directly to Vega-Altair to visualize the result of this data aggregation:

```python
import altair as alt

alt.Chart(res).mark_bar().encode(alt.X("ST:N").sort("-y"), alt.Y("FINCP_mean:Q"))
```

Conclusion

Beyond visualizations, dataframe interoperability enables developers to use features specific to certain libraries to develop new applications. This interoperability, enabled by standards and at the core of the Arrow ecosystem philosophy, allows you to build modular workflows to increase your productivity and take advantage of the best tool for your specific data analytics needs.

Voltron Data designs and builds composable data systems using open standards like Arrow to unlock interoperability and increase productivity. Visit our Product Page to learn about our approach.