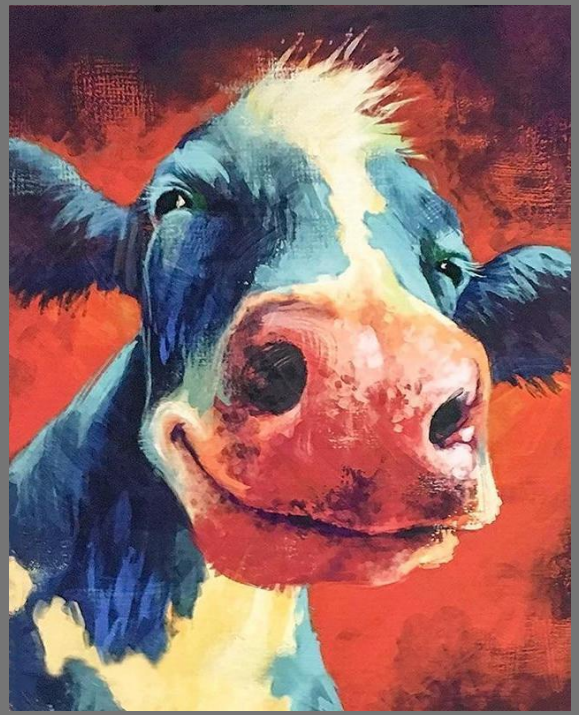
# Cheap
# Long-Term Storage of structured data

Using AWS Data Wrangler, S3 and Parquet files

@enricomarchesin
Principal Software Architect
@Cainthus



# User Requirements

- structured data
- long-term storage
- large* batch processing oriented
- standard & Open*
- cheap

# Ideas: data format

- structured data
- long-term storage
- large* batch processing oriented
- standard & Open*
- cheap

AVRO or PARQUET

# Ideas: storage

- structured data
- long-term storage
- large* batch processing oriented
- standard & Open*
- cheap

# Tools

- AVRO/Parquet
  - structured data
  - large* batch processing oriented
  - standard & Open*
- S3
  - long-term storage
  - standard & Open*
  - cheap

# Tools (in Python)

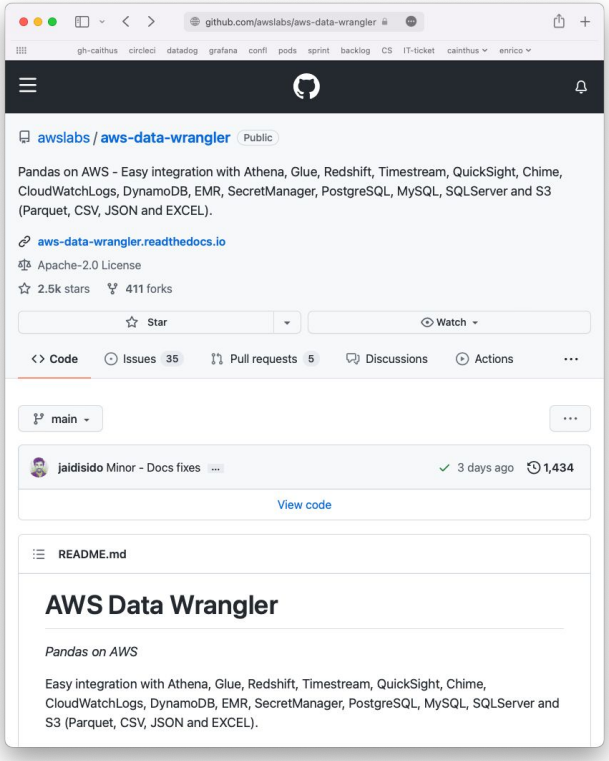
- AVRO/Parquet
  - structured data
  - large* batch processing oriented
  - standard & Open*
- S3
  - long-term storage
  - standard & Open
  - cheap

Pandas

# Tech requirements

- AVRO/Parquet
- S3
- the API is a Pandas DataFrame
- standard and Open*

AWS Data Wrangler



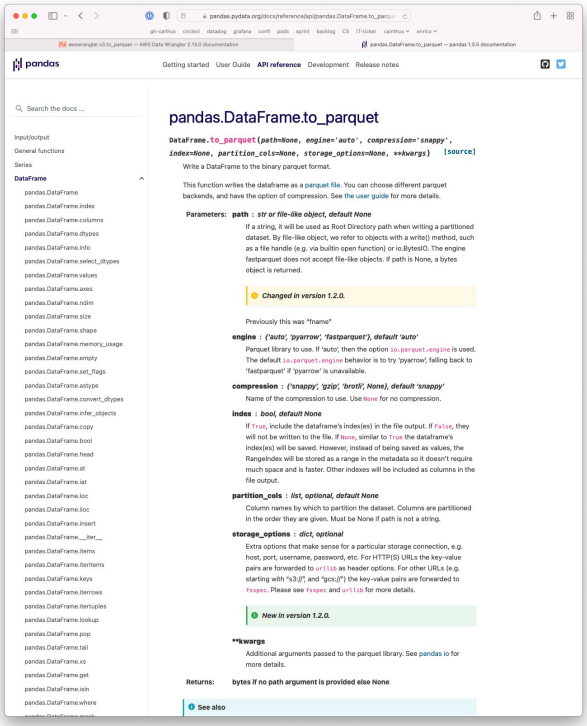

https://github.com/awslabs/aws-data-wrangler

# Let’s code!

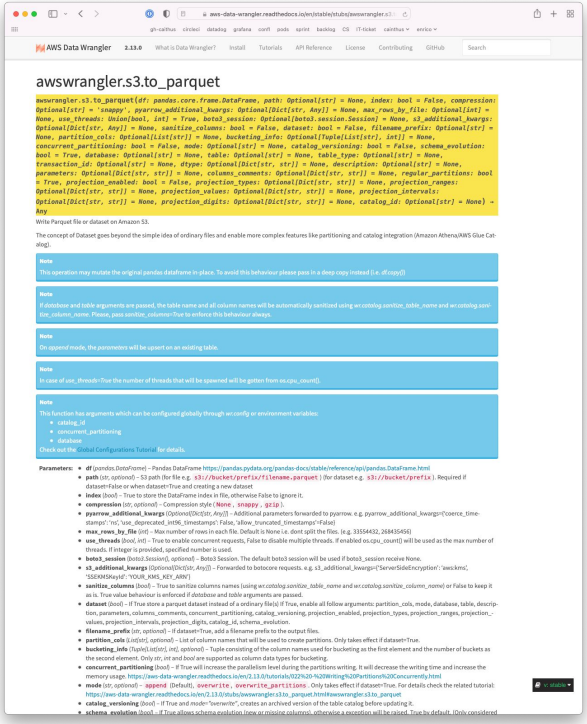
# Tenets Check

- AWS Data Wrangler
  - AVRO/Parquet
  - S3
  - the API is a Pandas DataFrame
  - standard and Open*

# From Pandas...


pandas
Getting started
User Guide
API reference
Development
Release notes
Search the docs ...
Input/output
General functions
Series
DataFrame
pandas.DataFrame
pandas.DataFrame.index
pandas.DataFrame.columns
pandas.DataFrame.dtypes
pandas.DataFrame.info
pandas.DataFrame.select_dtypes
pandas.DataFrame.values
pandas.DataFrame.axes
pandas.DataFrame.ndim
pandas.DataFrame.size
pandas.DataFrame.shape
pandas.DataFrame.memory_usage
pandas.DataFrame.empty
pandas.DataFrame.set_flags
pandas.DataFrame.astype
pandas.DataFrame.convert_dtypes
pandas.DataFrame.infer_objects
pandas.DataFrame.copy
pandas.DataFrame.bool
pandas.DataFrame.head
pandas.DataFrame.at
pandas.DataFrame.iat
pandas.DataFrame.loc
pandas.DataFrame.iloc
pandas.DataFrame.insert
pandas.DataFrame.__iter__
pandas.DataFrame.items
pandas.DataFrame.iteritems
pandas.DataFrame.keys
pandas.DataFrame.itertuples
pandas.DataFrame.lookup
pandas.DataFrame.pop
pandas.DataFrame.tail
pandas.DataFrame.xs
pandas.DataFrame.get
pandas.DataFrame.isin
pandas.DataFrame.where
pandas.DataFrame.to_parquet
DataFrame.to_parquet(path=None, engine='auto', compression='snappy', index=None, partition_cols=None, storage_options=None, **kwargs) [source]
Write a DataFrame to the binary parquet format.
This function writes the dataframe as a parquet file. You can choose different parquet backends, and have the option of compression. See the user guide for more details.
Parameters:
path : str or file-like object, default None
If a string, it will be used as Root Directory path when writing a partitioned dataset. By file-like object, we refer to objects with a write() method, such as a file handle (e.g. via builtin open function) or io.BytesIO. The engine fastparquet does not accept file-like objects. If path is None, a bytes object is returned.
Changed in version 1.2.0.
Previously this was "fname"
engine : {'auto', 'pyarrow', 'fastparquet'}, default 'auto'
Parquet library to use. If 'auto', then the option io.parquet.engine is used. The default io.parquet.engine behavior is to try 'pyarrow', falling back to 'fastparquet' if 'pyarrow' is unavailable.
compression : {'snappy', 'gzip', 'brotli', None}, default 'snappy'
Name of the compression to use. Use None for no compression.
index : bool, default None
If True, include the dataframe's index(es) in the file output. If False, they will not be written to the file. If None, similar to True the dataframe's index(es) will be saved. However, instead of being saved as values, the RangeIndex will be stored as a range in the metadata so it doesn't require much space and is faster. Other indexes will be included as columns in the file output.
partition_cols : list, optional, default None
Column names by which to partition the dataset. Columns are partitioned in the order they are given. Must be None if path is not a string.
storage_options : dict, optional
Extra options that make sense for a particular storage connection, e.g. host, port, username, password, etc. For HTTP(S) URLs the key-value pairs are forwarded to urllib as header options. For other URLs (e.g. starting with "s3://", and "gcs://") the key-value pairs are forwarded to fsspec. Please see fsspec and urllib for more details.
New in version 1.2.0.
**kwargs
Additional arguments passed to the parquet library. See pandas io for more details.
Returns:
bytes if no path argument is provided else None
See also

# ...to ADW



AWS Data Wrangler 2.13.0 What is Data Wrangler? Install Tutorials API Reference License Contributing GitHub Search

## awswrangler.s3.to_parquet

```
awswrangler.s3.to_parquet(df: pandas.core.frame.DataFrame, path: Optional[str] = None, index: bool = False, compression: Optional[str] = 'snappy', pyarrow_additional_kwargs: Optional[Dict[str, Any]] = None, max_rows_by_file: Optional[int] = None, use_threads: Union[bool, int] = True, boto3_session: Optional[boto3.session.Session] = None, s3_additional_kwargs: Optional[Dict[str, Any]] = None, sanitize_columns: bool = False, dataset: bool = False, filename_prefix: Optional[str] = None, partition_cols: Optional[List[str]] = None, bucketing_info: Optional[Tuple[List[str], int]] = None, concurrent_partitioning: bool = False, mode: Optional[str] = None, catalog_versioning: bool = False, schema_evolution: bool = True, database: Optional[str] = None, table: Optional[str] = None, table_type: Optional[str] = None, transaction_id: Optional[str] = None, dtype: Optional[Dict[str, str]] = None, description: Optional[str] = None, parameters: Optional[Dict[str, str]] = None, columns_comments: Optional[Dict[str, str]] = None, regular_partitions: bool = True, projection_enabled: bool = False, projection_types: Optional[Dict[str, str]] = None, projection_ranges: Optional[Dict[str, str]] = None, projection_values: Optional[Dict[str, str]] = None, projection_intervals: Optional[Dict[str, str]] = None, projection_digits: Optional[Dict[str, str]] = None, catalog_id: Optional[str] = None) → Any
```

Write Parquet file or dataset on Amazon S3.

The concept of Dataset goes beyond the simple idea of ordinary files and enable more complex features like partitioning and catalog integration (Amazon Athena/AWS Glue Catalog).

**Note**
This operation may mutate the original pandas dataframe in-place. To avoid this behaviour please pass in a deep copy instead (i.e. *df.copy()*)

**Note**
If *database* and *table* arguments are passed, the table name and all column names will be automatically sanitized using *wr.catalog.sanitize_table_name* and *wr.catalog.sanitize_column_name*. Please, pass *sanitize_columns=True* to enforce this behaviour always.

**Note**
On *append* mode, the *parameters* will be upsert on an existing table.

**Note**
In case of *use_threads=True* the number of threads that will be spawned will be gotten from os.cpu_count().

**Note**
This function has arguments which can be configured globally through wr.config or environment variables:
- catalog_id
- concurrent_partitioning
- database

Check out the Global Configurations Tutorial for details.

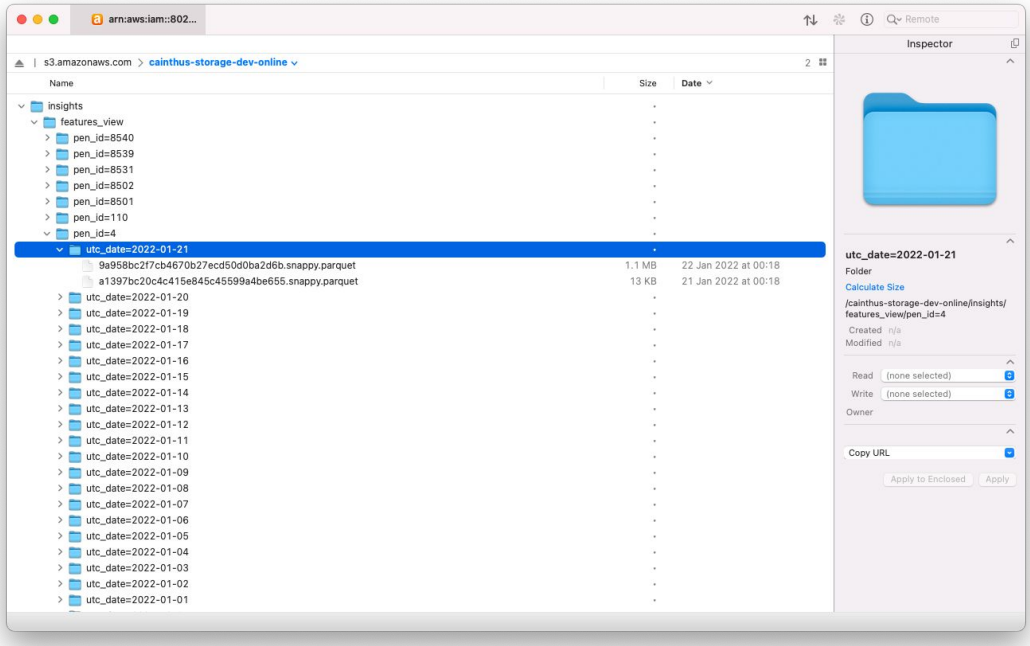
**Parameters:**
- **df** (*pandas.DataFrame*) – Pandas DataFrame https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html
- **path** (*str, optional*) – S3 path (for file e.g. `s3://bucket/prefix/filename.parquet`) (for dataset e.g. `s3://bucket/prefix`). Required if dataset=False or when dataset=True and creating a new dataset
- **index** (*bool*) – True to store the DataFrame index in file, otherwise False to ignore it.
- **compression** (*str, optional*) – Compression style (`None`, `snappy`, `gzip`).
- **pyarrow_additional_kwargs** (*Optional[Dict[str, Any]]*) – Additional parameters forwarded to pyarrow. e.g. pyarrow_additional_kwargs={'coerce_timestamps': 'ns', 'use_deprecated_int96_timestamps': False, 'allow_truncated_timestamps'=False}
- **max_rows_by_file** (*int*) – Max number of rows in each file. Default is None i.e. dont split the files. (e.g. 33554432, 268435456)
- **use_threads** (*bool, int*) – True to enable concurrent requests, False to disable multiple threads. If enabled os.cpu_count() will be used as the max number of threads. If integer is provided, specified number is used.
- **boto3_session** (*boto3.Session(), optional*) – Boto3 Session. The default boto3 session will be used if boto3_session receive None.
- **s3_additional_kwargs** (*Optional[Dict[str, Any]]*) – Forwarded to botocore requests. e.g. s3_additional_kwargs={'ServerSideEncryption': 'aws:kms', 'SSEKMSKeyId': 'YOUR_KMS_KEY_ARN'}
- **sanitize_columns** (*bool*) – True to sanitize columns names (using *wr.catalog.sanitize_table_name* and *wr.catalog.sanitize_column_name*) or False to keep it as is. True value behaviour is enforced if *database* and *table* arguments are passed.
- **dataset** (*bool*) – If True store a parquet dataset instead of a ordinary file(s) If True, enable all follow arguments: partition_cols, mode, database, table, description, parameters, columns_comments, concurrent_partitioning, catalog_versioning, projection_enabled, projection_types, projection_ranges, projection_values, projection_intervals, projection_digits, catalog_id, schema_evolution.
- **filename_prefix** (*str, optional*) – If dataset=True, add a filename prefix to the output files.
- **partition_cols** (*List[str], optional*) – List of column names that will be used to create partitions. Only takes effect if dataset=True.
- **bucketing_info** (*Tuple[List[str], int], optional*) – Tuple consisting of the column names used for bucketing as the first element and the number of buckets as the second element. Only *str*, *int* and *bool* are supported as column data types for bucketing.
- **concurrent_partitioning** (*bool*) – If True will increase the parallelism level during the partitions writing. It will decrease the writing time and increase the memory usage. https://aws-data-wrangler.readthedocs.io/en/2.13.0/tutorials/022%20-%20Writing%20Partitions%20Concurrently.html
- **mode** (*str, optional*) – `append` (Default), `overwrite`, `overwrite_partitions`. Only takes effect if dataset=True. For details check the related tutorial: https://aws-data-wrangler.readthedocs.io/en/2.13.0/stubs/awswrangler.s3.to_parquet.html#awswrangler.s3.to_parquet
- **catalog_versioning** (*bool*) – If True and *mode="overwrite"*, creates an archived version of the table catalog before updating it.
- **schema_evolution** (*bool*) – If True allows schema evolution (new or missing columns), otherwise a exception will be raised. True by default. (Only considered

# From `pd` to `wr`

```
df.to_parquet(
  path=f"s3://my-bucket/my-path/",
  partition_cols=[“date”, “obj_id”],
)
```

```
wr.s3.to_parquet(
  df,
  path=f"s3://my-bucket/my-path/",
  partition_cols=[“date”, “obj_id”],
  dataset=True,
)
```

# Partitions


arn:aws:iam::802...
Remote
s3.amazonaws.com
cainthus-storage-dev-online
2
Name
Size
Date
insights
features_view
pen_id=8540
pen_id=8539
pen_id=8531
pen_id=8502
pen_id=8501
pen_id=110
pen_id=4
utc_date=2022-01-21
9a958bc2f7cb4670b27ecd50d0ba2d6b.snappy.parquet
1.1 MB
22 Jan 2022 at 00:18
a1397bc20c4c415e845c45599a4be655.snappy.parquet
13 KB
21 Jan 2022 at 00:18
utc_date=2022-01-20
utc_date=2022-01-19
utc_date=2022-01-18
utc_date=2022-01-17
utc_date=2022-01-16
utc_date=2022-01-15
utc_date=2022-01-14
utc_date=2022-01-13
utc_date=2022-01-12
utc_date=2022-01-11
utc_date=2022-01-10
utc_date=2022-01-09
utc_date=2022-01-08
utc_date=2022-01-07
utc_date=2022-01-06
utc_date=2022-01-05
utc_date=2022-01-04
utc_date=2022-01-03
utc_date=2022-01-02
utc_date=2022-01-01
Inspector
utc_date=2022-01-21
Folder
Calculate Size
/cainthus-storage-dev-online/insights/features_view/pen_id=4
Created n/a
Modified n/a
Read (none selected)
Write (none selected)
Owner
Copy URL
Apply to Enclosed
Apply

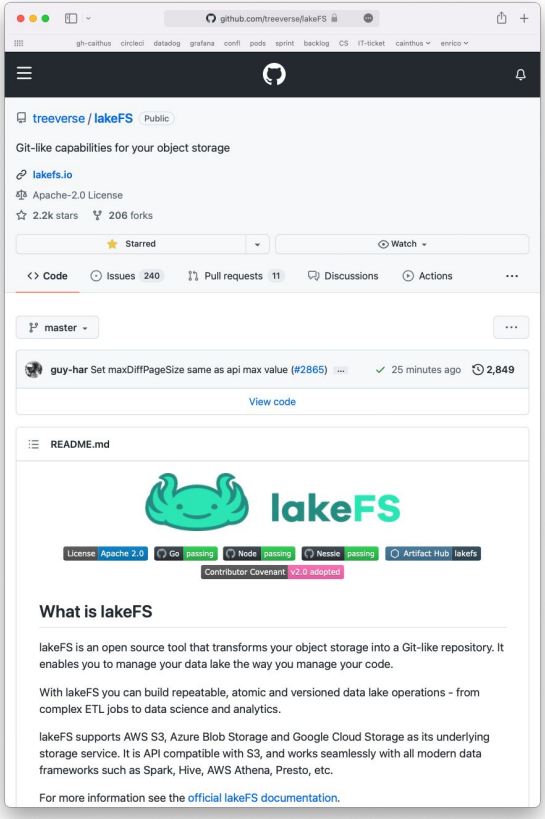
# AWS Glue?

*Maybe*

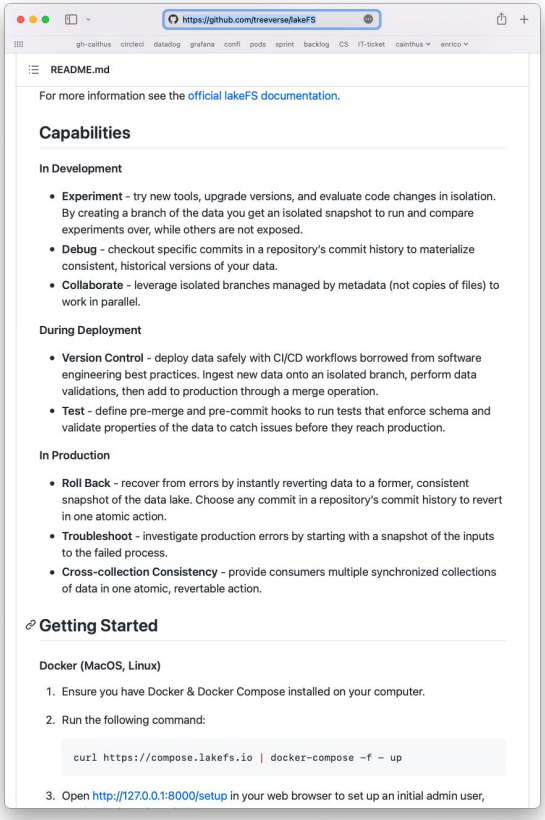
*But we don’t need it now*

*KISS* 😙

# What's next...




https://github.com/treeverse/lakeFS




https://github.com/treeverse/lakeFS