HELLO POLARS!

LIGHTNING-FAST DATAFRAME LIBRARY FOR RUST AND PYTHON

Antonio Bevilacqua
18/10/2022

HI!

ANTONIO BEVILACQUA
POSTDOCTORAL RESEARCHER @INSIGHT-CENTRE, UCD

thisisanton.io
github/b3by

CONTENT

BRIEF INTRODUCTION ABOUT POLARS
[key concepts]

POLARS IS **NOT** PANDAS*
[key differences]

PERFORMANCE & BENCHMARKS
[hands-on]
[H2O.ai]

* but close enough

WHAT IS POLARS?

WHAT IS POLARS?

DATAFRAME LIBRARY

WHAT IS POLARS?

DATAFRAME LIBRARY
IN-MEMORY QUERY ENGINE

WHAT IS POLARS?

DATAFRAME LIBRARY
IN-MEMORY QUERY ENGINE
DBMS-ESQUE LAYER

WHAT IS POLARS?

DATAFRAME LIBRARY
IN-MEMORY QUERY ENGINE
DBMS-ESQUE LAYER

BASED ON THE SAME PANDAS CONCEPT OF DATAFRAMES
FILLING THE GAPS BETWEEN PANDAS AND SPARK

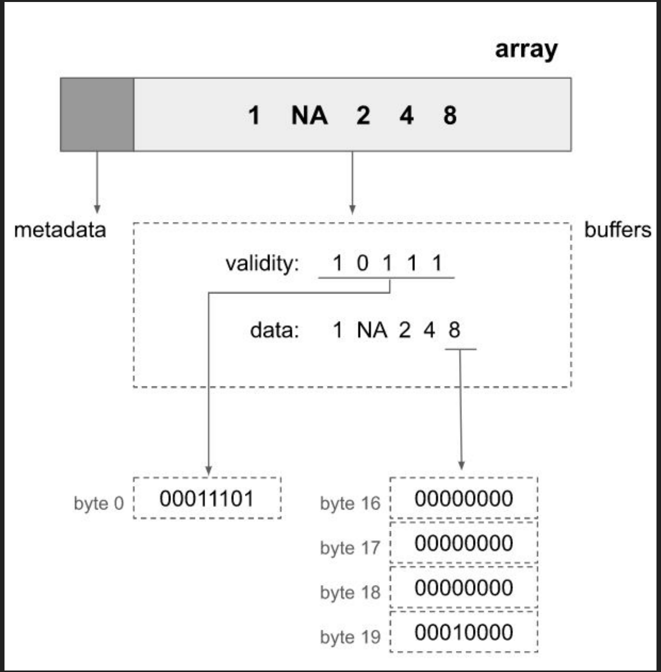
POLARS DATA STRUCTURES

POLARS USES **APACHE ARROW** COLUMNAR FORMAT
[in-memory optimizations]
[parallel by default]

DATA STRUCTURES BUILT IN CONTIGUOUS MEMORY LOCATIONS
IDEAL FOR BIT MASKING AND MULTITHREADING

POLARS DATA STRUCTURES

NUMERIC ARRAYS



POLARS DATA STRUCTURES

NUMERIC ARRAYS

VALIDITY ENDIANNESS IS A BIT CONFUSING

workbox ❐ 2 ● 1 python

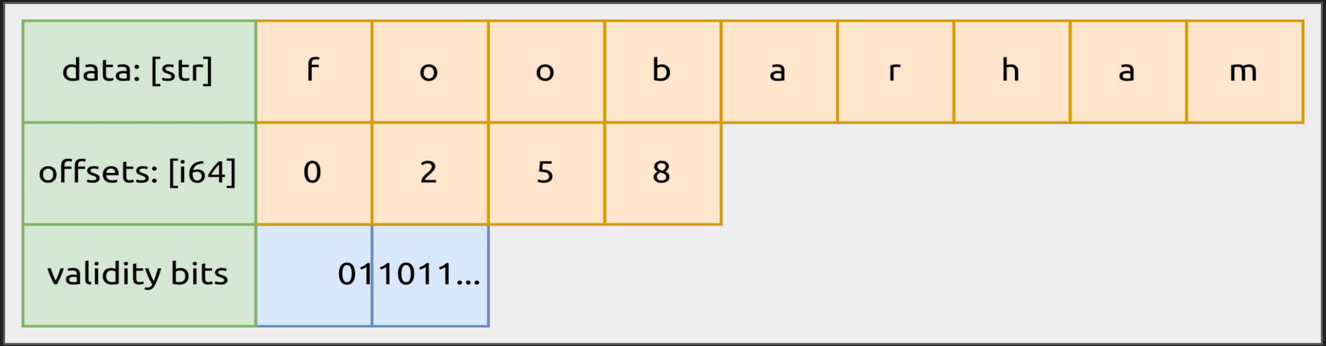
```
➜  ~ python                                                  (env: pandas_polars_cmp)
Python 3.9.8 (main, Nov 11 2021, 12:14:57)
[GCC 7.5.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import polars as pl
>>> array = pl.Series([1, None, 2, 4, 8])
>>> arrow = array.to_arrow()
>>> validity = arrow.buffers()[0]
>>> data = arrow.buffers()[1]
>>> bin(int(validity.hex(), base=16))
'0b11111101'
>>> data.hex()
b'01000000000000000000000000000000020000000000000004000000000000000800000000000000'
>>> 
```

家 2 ↑ 46d 42m 1 python 12:17 14 Oct anto workbox

(requires pyarrow)

POLARS DATA STRUCTURES

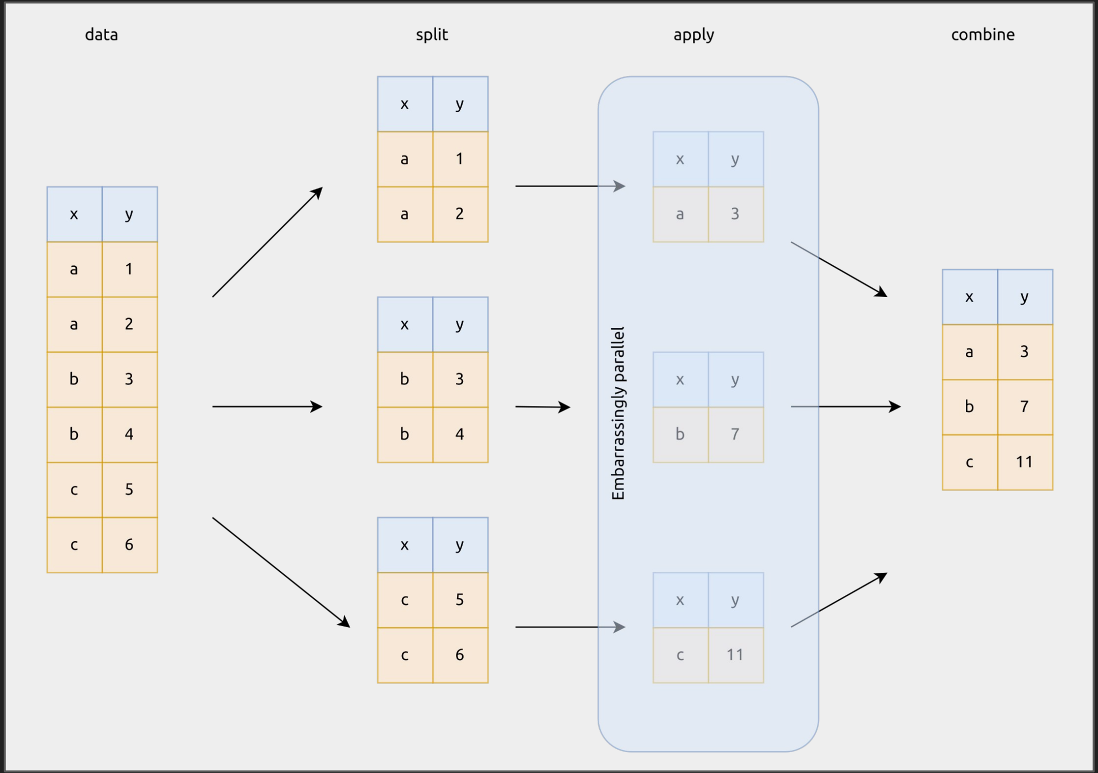
STRING ARRAYS



OFFSET BITS POINT TO VALUE INDEXES IN MEMORY

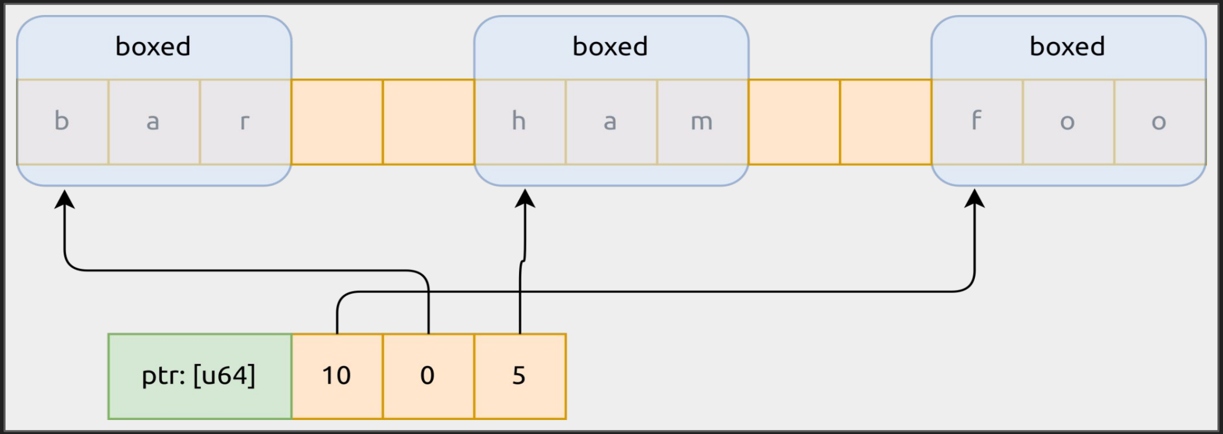
POLARS DATA STRUCTURES

STRING ARRAYS



NUMPY REPRESENTATION

STRING ARRAYS



PANDAS USES NUMPY OBJECTS ALLOCATED ON THE HEAP
SOME TIME WASTED ON LOOKUP & CACHE MISS :(

POLARS APIs

AVAILABLE FOR PYTHON & RUST
PROVIDES **EAGER** APIs AND **LAZY** APIs

	EAGER		LAZY	
	QUERY PIPELINES ARE EVALUATED ON THE FLY (LIKE PANDAS)		QUERY PIPELINES ARE OPTIMIZED FIRST AND EVALUATED ONLY WHEN COLLECTED	

POLARS APIs

```python
import polars as pl

pl.read_csv('iris.csv')
  .filter(pl.col('sepal_length') > 5)
  .groupby('species')
  .agg(pl.all().sum())
```

```python
import polars as pl

pl.read_csv('iris.csv')
  .lazy()
  .filter(pl.col('sepal_length') > 5)
  .groupby('species')
  .agg(pl.all().sum())
  .collect()
```

POLARS VS PANDAS

POLARS DOES NOT HAVE INDEXES
[indexing is considered to be an anti-pattern]

POLARS USES APACHE ARROWS vs. NUMPY NDARRAYS
[we already talked about that]

POLARS HAS LAZY APIs & IS MULTITHREADING-READY
[some frameworks try to fill the gap in pandas]

POPULARITY AND ADOPTION

35.6K	STARS	8.6K
89	RELEASES	10
30.3K	COMMITS	4.5K
15.2K	FORKS	471

DON'T RESIST THE HYPE!

LET’S LOOK AT SOME NUMBERS NOW!

BENCHMARK SETUP

KAGGLE DATASET OF TRANSACTIONS FROM ONLINE STORE*
(LARGE: 9GB + 5GB)

	event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
i64	str	str	i64	i64	str	str	f64	i64	str
0	"2019-10-01 00:...	"view"	44600062	2103807459595387724	null	"shiseido"	35.79	541312140	"72d76fde-8bb3-...
1	"2019-10-01 00:...	"view"	3900821	2053013552326770905	"appliances.env...	"aqua"	33.2	554748717	"9333dfbd-b87a-...
2	"2019-10-01 00:...	"view"	17200506	2053013559792632471	"furniture.livi...	null	543.1	519107250	"566511c2-e2e3-...
3	"2019-10-01 00:...	"view"	1307067	2053013558920217191	"computers.note...	"lenovo"	251.74	550050854	"7c90fc70-0e80-...
4	"2019-10-01 00:...	"view"	1004237	2053013555631882655	"electronics.sm...	"apple"	1081.98	535871217	"c6bd7419-2748-...

* https://www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store

BENCHMARK SETUP

PARTITIONS TESTED FOR BENCHMARKS (# ROWS)

1000 → ~130 kB
10_000
100_000
1_000_000
10_000_000
100_000_000 → ~13 GB

BENCHMARK SETUP

QUERIES TESTED FOR BENCHMARKS SPLIT INTO GROUPS

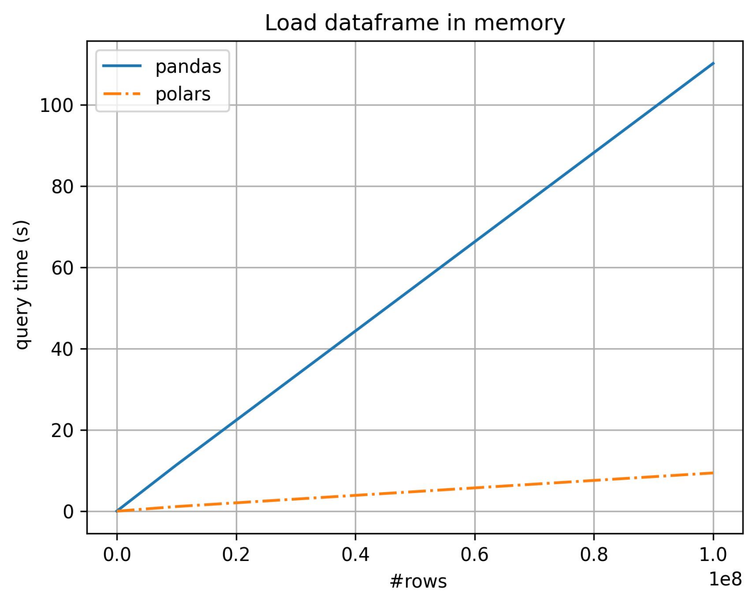
DATA LOAD IN MEMORY

DATA SUMMARY
[describe, count values, count uniques, count nans...]

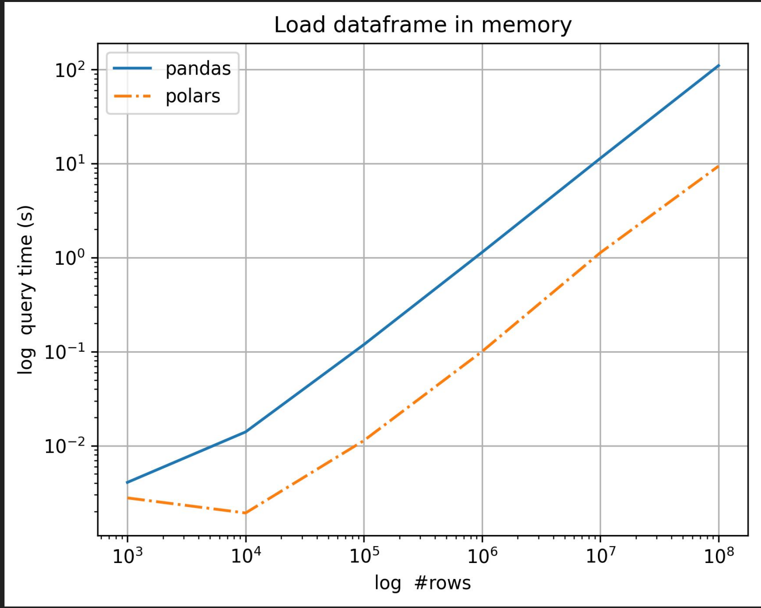
DATA ORDERING & FILTERING
[sorting columns, logical conditions...]

GROUPING

LOAD DATA

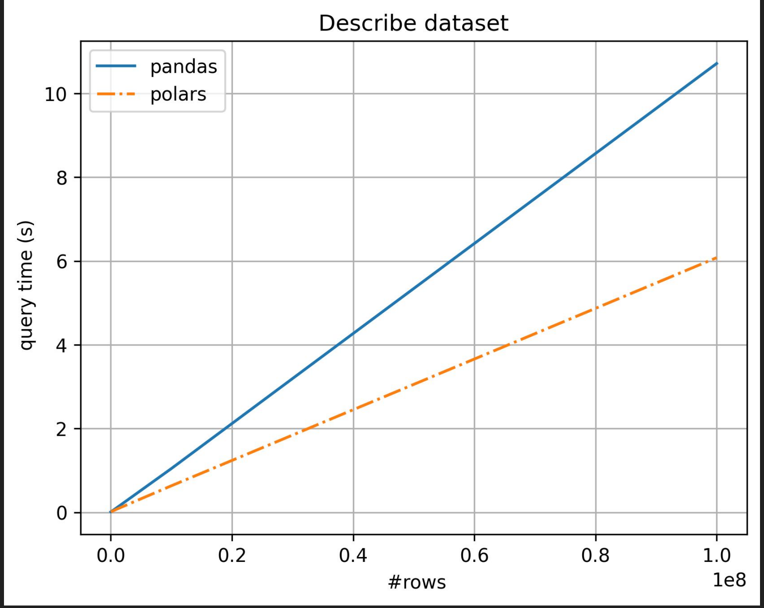


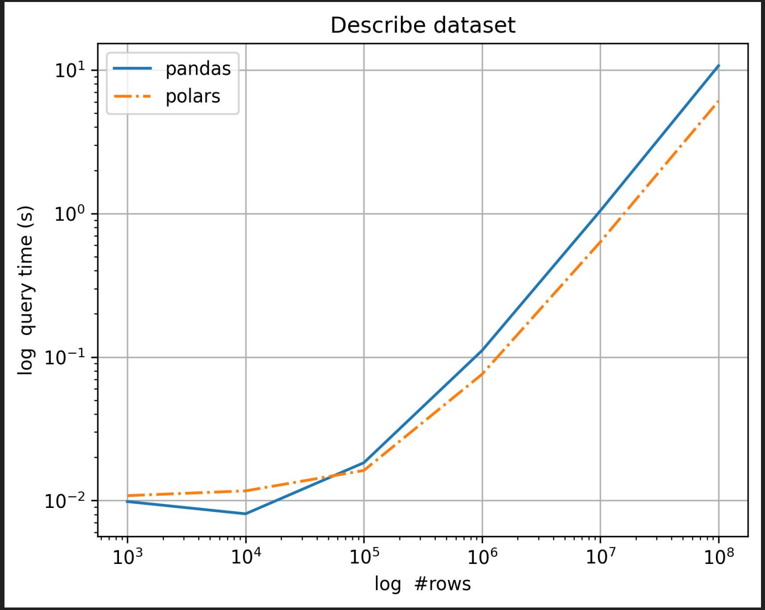
Load dataframe in memory
pandas
polars
query time (s)
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0
1e8
#rows



Load dataframe in memory
pandas
polars
log query time (s)
10^2
10^1
10^0
10^{-1}
10^{-2}
10^3
10^4
10^5
10^6
10^7
10^8
log #rows

DESCRIBE DATA



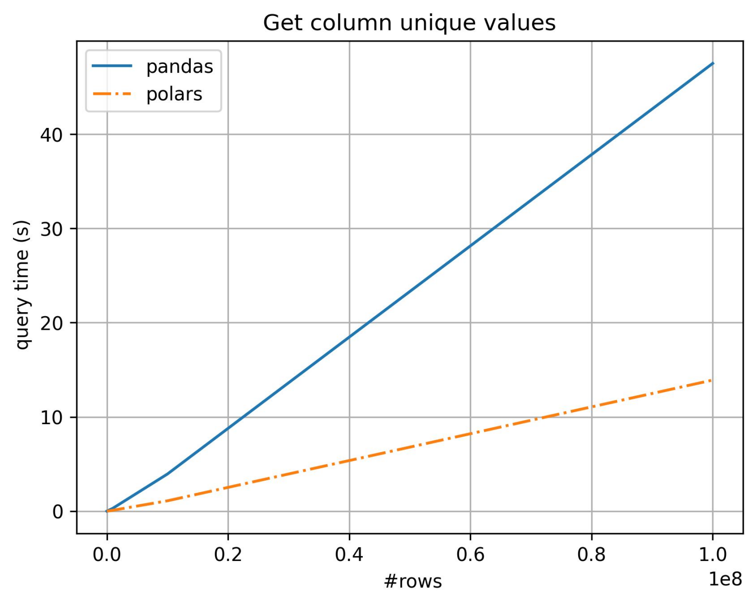


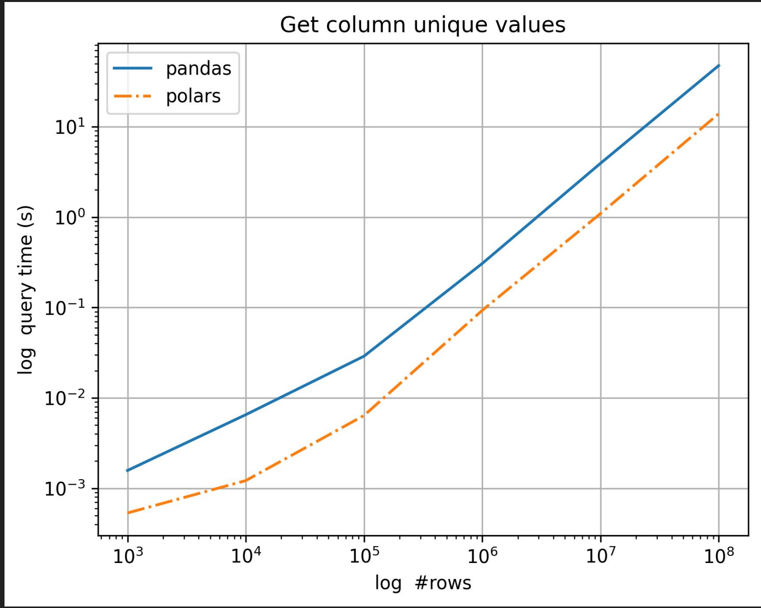
GET COLUMN UNIQUE VALUES

```
import polars as pl

pl.read_csv(data.csv’)
  .select([
    pl.col(‘*’).unique().count()
])
```

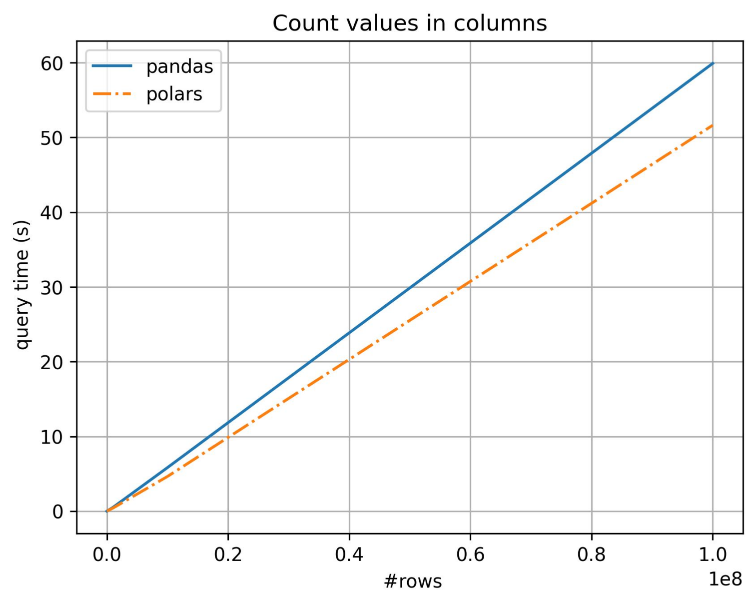
```
import pandas as pd

pd.read_csv(‘data.csv’)
  .apply(λ col: len(col.uniques())
```

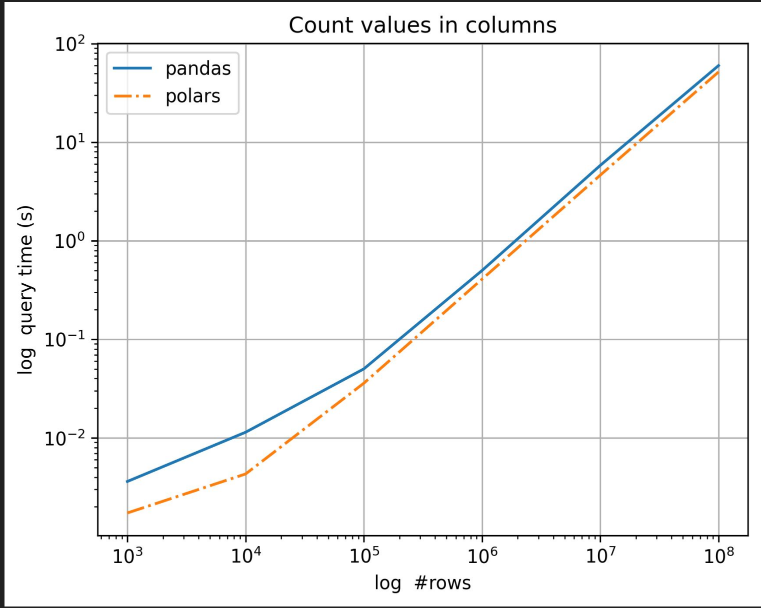
GET COLUMN UNIQUE VALUES



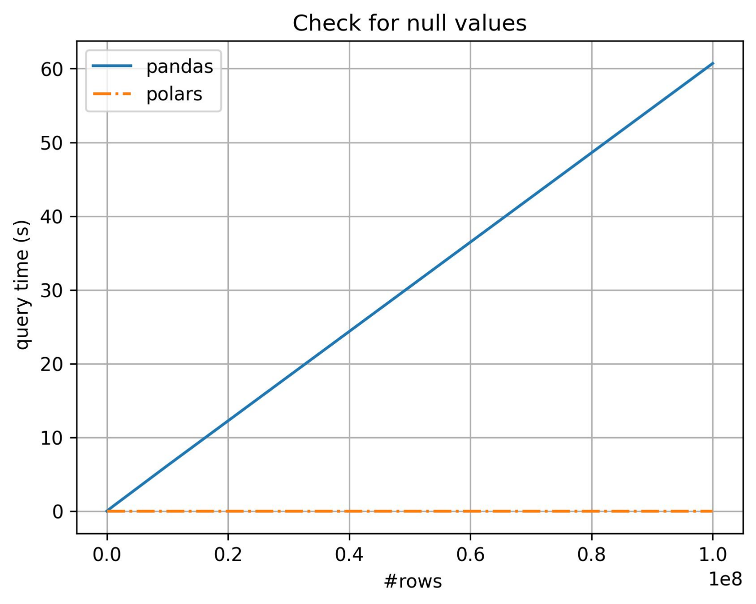


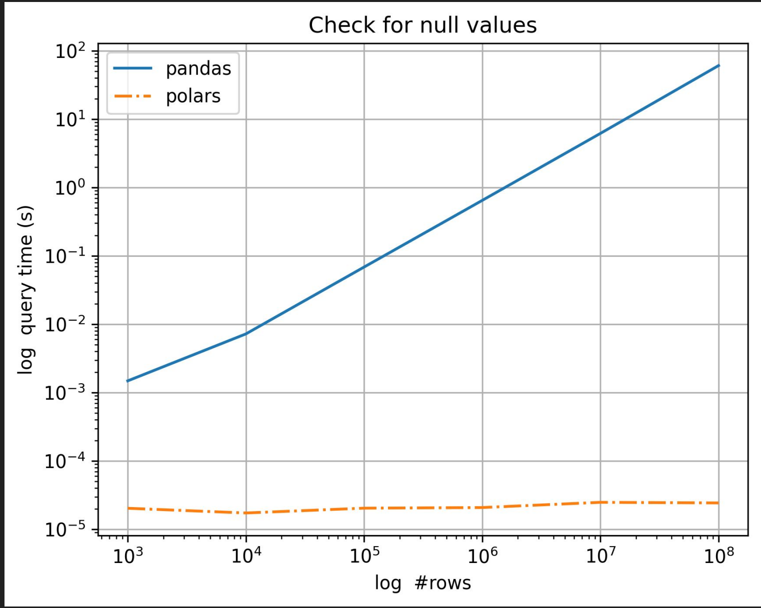
VALUE COUNTS



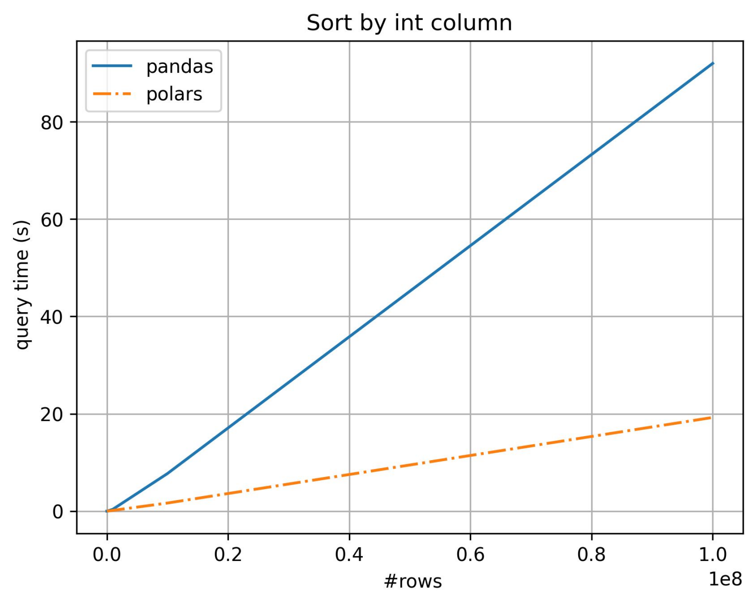


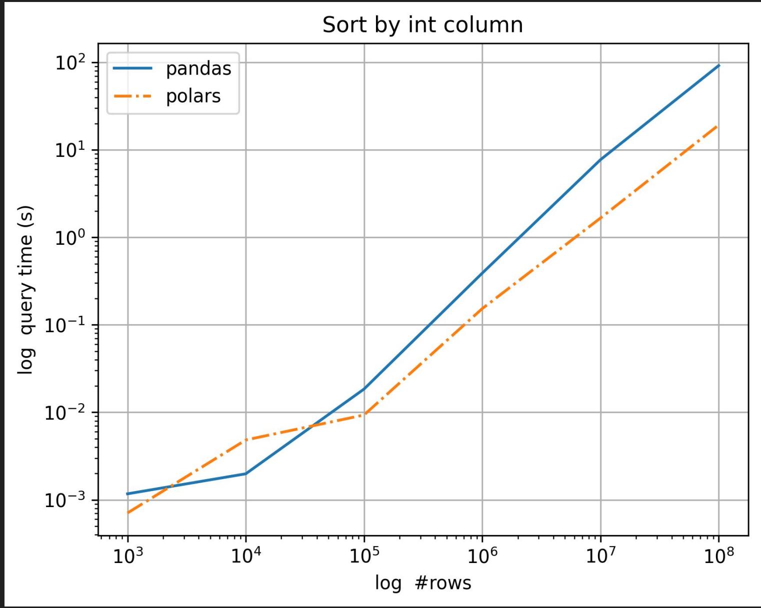
CHECK FOR NULL VALUES





SORT BY COLUMN (INT)



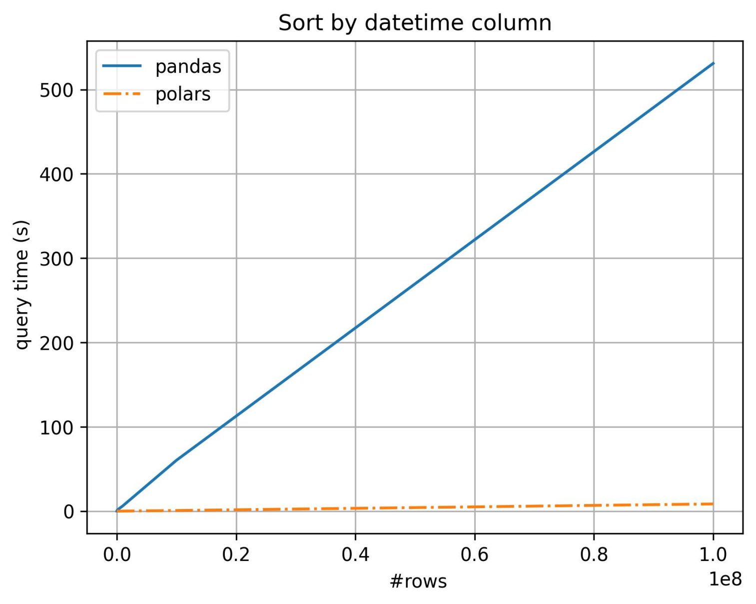


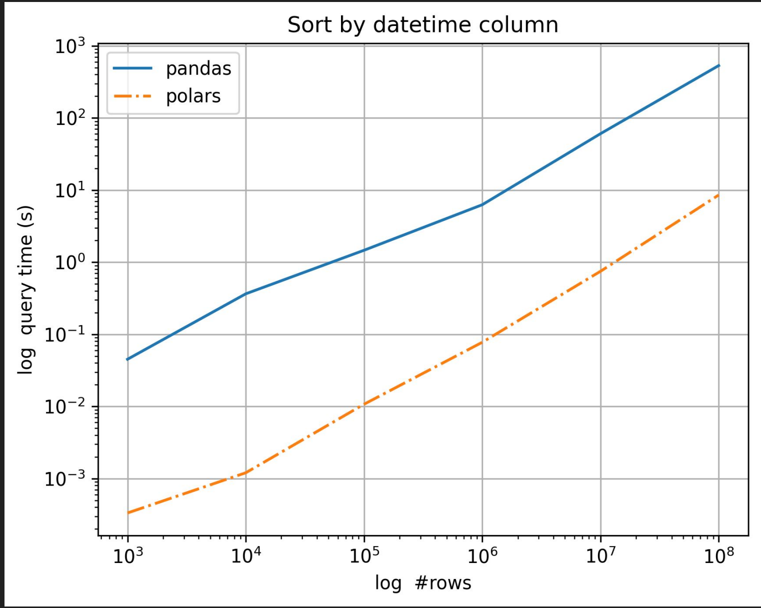
SORT BY COLUMN (DATETIME)

```
import polars as pl

pl.read_csv(data.csv’)
  .select([
    pl.col(‘event_time’)
      .str
      .strptime(pl.Datetime, fmt=’...’)
      .sort()
])
```

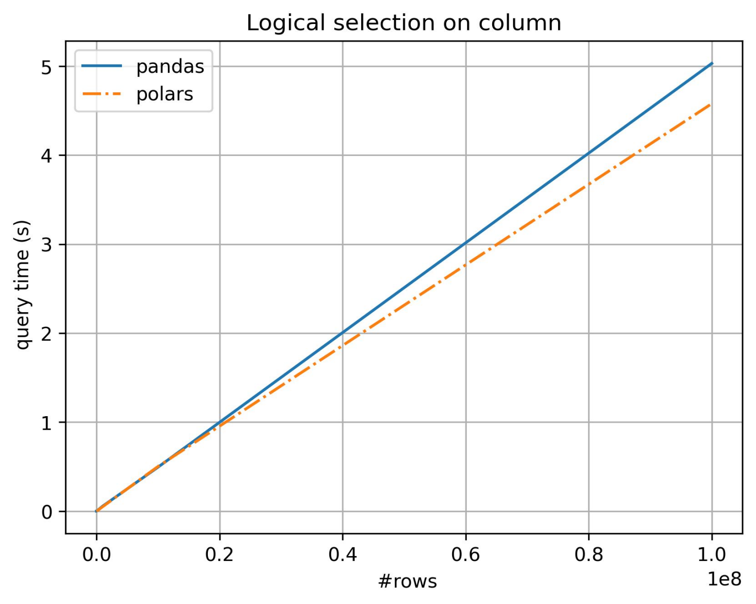
```
import pandas as pd

df = pd.read_csv(‘data.csv’)

pd.to_datetime(df[‘event_time’])
  .sort_values()
```

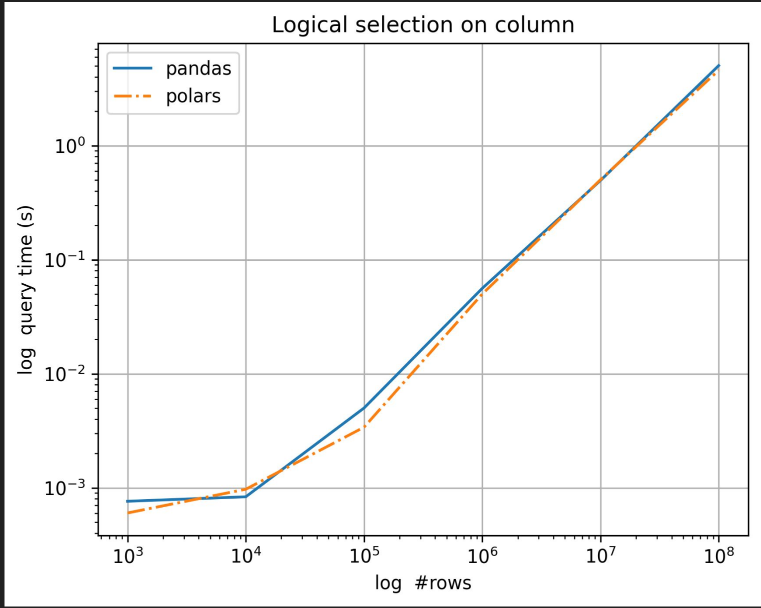
SORT BY COLUMN (DATETIME)





LOGICAL SELECTION



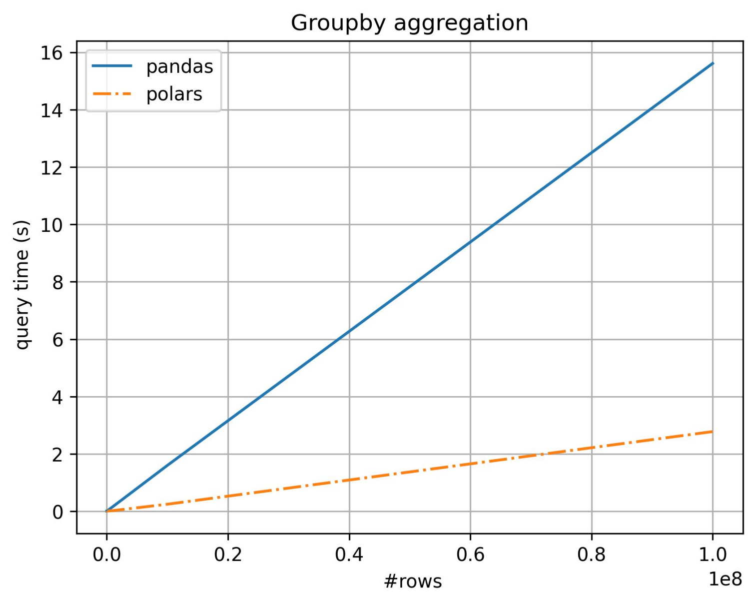


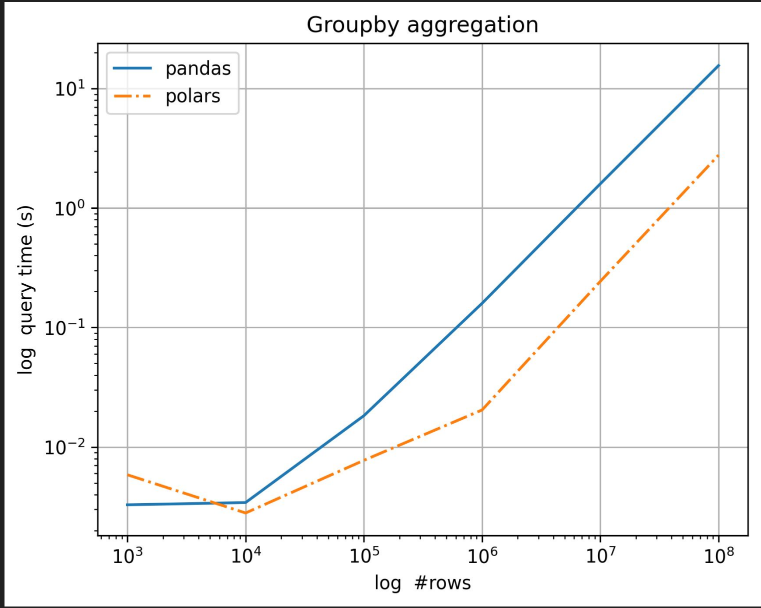
GROUPBY AGGREGATION

```python
import polars as pl

pl.read_csv(data.csv’)
  .groupby(‘brand’).agg([
    pl.col(‘price’).min(),
    pl.col(‘price’).max(),
    pl.col(‘price’).std(),
    pl.col(‘price’).mean()
])
```

```python
import pandas as pd

pd.read_csv(‘data.csv’)
  .groupby(‘brand’)[‘price’]
  .agg([‘min’, ‘max’, ‘std’, ‘mean’])
```

GROUPBY AGGREGATION





LINKS WHERE I STOLE MY STUFF

OFFICIAL POLARS WEBSITE
https://www.pola.rs/

APACHE ARROW FORMAT DOCS
https://arrow.apache.org/docs/format/Columnar.html

H2O BENCHMARKS
https://h2oai.github.io/db-benchmark/

MORE ON ARROWS
https://blog.djnavarro.net/posts/2022-05-25_arrays-and-tables-in-arrow/

THANK YOU!

?? || /* */