what are the main difference between pandas and polars ?

Pandas and Polars are both libraries used for data manipulation and analysis in Python, but they have some differences in terms of performance, memory usage, and functionality. Here are some key differences:

1. Performance

2. Memory Usage

3. Functionality
4. Language and Implementation

Performance

•**Pandas:** Pandas is a widely-used library known for its flexibility and ease of use. However, for large datasets, it can sometimes be slower due to its underlying implementation in Python, especially when dealing with operations on large data frames .

•**Polars:** Polars is designed to offer faster performance compared to Pandas, especially for operations on larger datasets. It's built on Rust, which provides it with a performance boost, making it more suitable for handling big data efficiently.

Memory Usage

•**Pandas:** Pandas can be memory-intensive, especially with large datasets. It creates copies of data, which can lead to high memory usage, impacting performance when dealing with substantial amounts of data.

•**Polars:** Polars is optimized for memory efficiency. It uses columnar memory layout and employs zero-copy techniques, which help reduce memory overhead. This makes it more efficient when working with large datasets, as it can handle them using less memory compared to Pandas.

Functionality

- **Pandas:** Pandas has a vast ecosystem of tools and libraries built around it. It offers a wide range of functions and methods for data manipulation, cleaning, merging, reshaping, and analysis. Its extensive documentation and community support make it a go-to choice for many data scientists and analysts.
- **Polars:** Polars is relatively newer compared to Pandas and might have a smaller ecosystem and fewer functionalities in comparison. However, it's growing and aims to offer a similar range of functionalities optimized for performance.

Language and Implementation

•**Pandas:** Pandas is written in Python and relies heavily on pandas DataFrames and Series for data manipulation.

•**Polars:** Polars is written in Rust, a language known for its performance and memory safety. It focuses on providing Data Frame structures for efficient data processing.

•Choosing between Pandas and Polars often depends on the specific use case. Pandas is more established, has a larger community, and is great for general-purpose data analysis and manipulation. On the other hand, Polars is ideal when dealing with larger datasets that require better performance and memory efficiency.

Choose Polars When

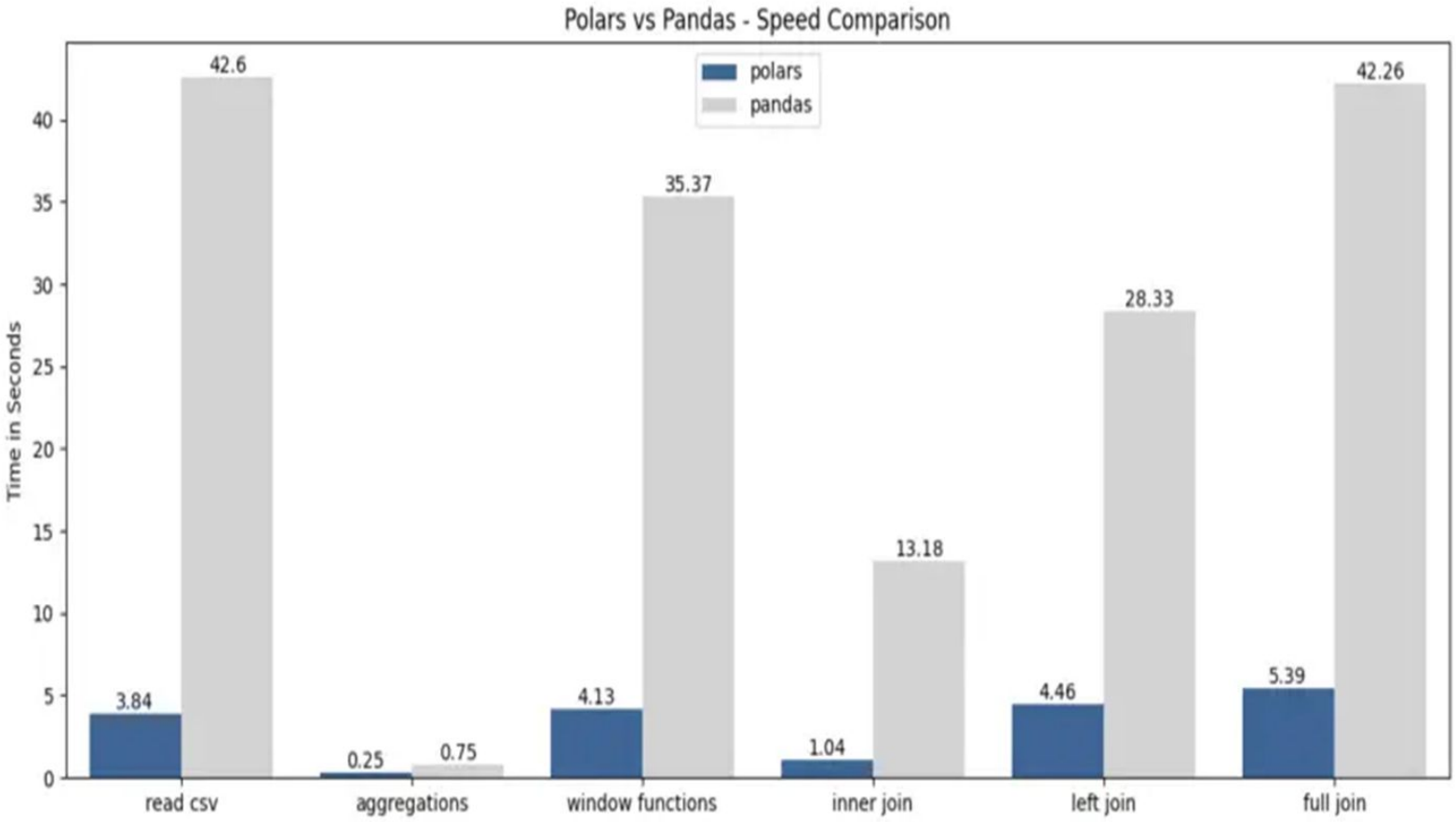
•**Performance at Scale**: For handling large-scale data processing tasks where performance and memory efficiency are critical, especially with big datasets, Polars shines due to its optimized performance and memory utilization.

•

•**Big Data Processing:** When dealing with huge datasets that might strain the memory or performance capabilities of Pandas, Polars' efficient memory handling and columnar operations become highly beneficial.

•

•**Advanced Optimizations:** If you're comfortable with a slightly smaller but rapidly growing ecosystem and want to leverage advanced optimizations for speed and memory, Polars can provide a significant advantage.


Polars vs Pandas - Speed Comparison
polars
pandas
Time in Seconds
0
5
10
15
20
25
30
35
40
3.84
42.6
0.25
0.75
4.13
35.37
1.04
13.18
4.46
28.33
5.39
42.26
read csv
aggregations
window functions
inner join
left join
full join

basic questions #

Input table: 100,000,000 rows x 7 columns (5 GB)

Polars	0.8.8	2021-06-30	43s
data.table	1.14.1	2021-06-30	92s
ClickHouse	21.3.2.5	2021-05-12	159s
spark	3.1.2	2021-05-31	332s
DataFrames.jl	1.1.1	2021-06-03	349s
dplyr	1.0.7	2021-06-20	370s
(py)datatable	1.0.0a0	2021-06-30	500s
pandas	1.2.5	2021-06-30	628s
DuckDB	0.2.7	2021-06-15	630s
dask	2021.04.1	2021-05-09	internal error
cuDF*	0.19.2	2021-05-31	internal error
Arrow	4.0.1	2021-05-31	not yet implemented
Modin		see README	pending

First time
Second time

PANDAS AND POLAR INSTALLATION

Pandas :

To install pandas : pip install pandas

To import pandas : import pandas as pd

Polar :

To install polar : pip install polar

To import polar : import polar

1. Selecting data

To select specific columns from a DataFrame, you can use the `select()` method

```
import polars as pl


# Load diamond data from a CSV file
df = pl.read_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/datasets/diamond.csv')


# Select specific columns: carat, cut, and price
selected_df = df.select(['Carat Weight', 'Cut', 'Price'])


# show selected_df head
selected_df.head()
```

Output:

shape: (5, 3)

Carat Weight	Cut	Price
f64	str	i64
1.1	"Ideal"	5169
0.83	"Ideal"	3470
0.85	"Ideal"	3183
0.91	"Ideal"	4370
0.83	"Ideal"	3171

2. Sorting and ordering data

Polars provides the `sort()` method to sort a DataFrame based on one or more columns.

```
import polars as pl


# Load diamond data from a CSV file
df = pl.read_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/datasets/diamond.csv')


# sort the df by price
sorted_df = df.sort(by='Price')


# show sorted_df head
sorted_df.head()
```

Output:

shape: (5, 8)

Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
f64	str	str	str	str	str	str	i64
0.77	"Good"	"I"	"VS1"	"VG"	"G"	"AGSL"	2184
0.77	"Good"	"I"	"SI1"	"EX"	"VG"	"GIA"	2241
0.78	"Very Good"	"I"	"SI1"	"EX"	"VG"	"GIA"	2348
0.75	"Ideal"	"I"	"SI1"	"VG"	"VG"	"GIA"	2383
0.76	"Very Good"	"H"	"SI1"	"G"	"G"	"GIA"	2396

3.Handling missing values

Polars provides convenient methods to handle missing values. The `drop_nulls()` method allows you to drop rows that contain any missing values:

```
import polars as pl


# Load diamond data from a CSV file
df = pl.read_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/datasets/diamond.csv')


# drop missing values
cleaned_df = df.drop_nulls()


# show cleaned_df head
cleaned_df.head()
```

Output:

shape: (5, 8)

Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
f64	str	str	str	str	str	str	i64
1.1	"Ideal"	"H"	"SI1"	"VG"	"EX"	"GIA"	5169
0.83	"Ideal"	"H"	"VS1"	"ID"	"ID"	"AGSL"	3470
0.85	"Ideal"	"H"	"SI1"	"EX"	"EX"	"GIA"	3183
0.91	"Ideal"	"E"	"SI1"	"VG"	"VG"	"GIA"	4370
0.83	"Ideal"	"G"	"SI1"	"EX"	"EX"	"GIA"	3171

4. Grouping data based on specific columns

To group data based on specific columns, you can use the `groupby()` method

```
import polars as pl


# Load diamond data from a CSV file
df = pl.read_csv('https://raw.githubusercontent.com/pycaret/pycaret/master/datasets/diamond.csv')


# group by cut and calc mean of price
grouped_df = df.groupby(by='Cut').agg(pl.col('Price').mean())


# show grouped_df head
grouped_df.head()
```

Output:

shape: (5, 2)

Cut	Price
str	f64
"Very Good"	11484.69687
"Signature-Idea…	11541.525692
"Ideal"	13127.331185
"Good"	9326.65678
"Fair"	5886.178295

5. Joining and combining Data Frames

To perform a join operation, you can use the `join()` method.

```
import polars as pl
# Create the first DataFrame
df1 = pl.DataFrame({
    'id': [1, 2, 3, 4],
    'name': ['Alice', 'Bob', 'Charlie', 'David']
})
# Create the second DataFrame
df2 = pl.DataFrame({
    'id': [2, 3, 5],
    'age': [25, 30, 35]
})

# Perform an inner join on the 'id' column
joined_df = df1.join(df2, on='id')

# Display the joined DataFrame
joined_df
```

Output:

shape: (2, 3)

id	name	age
i64	str	i64
2	"Bob"	25
3	"Charlie"	30

THANK YOU