✦ Member-only story

Pandas vs. Polars: A Syntax and Speed Comparison

Understanding the major differences between the Python libraries Pandas and Polars for Data Science

Leonie Monigatti · Follow
Published in Towards Data Science · 7 min read · Jan 11, 2023

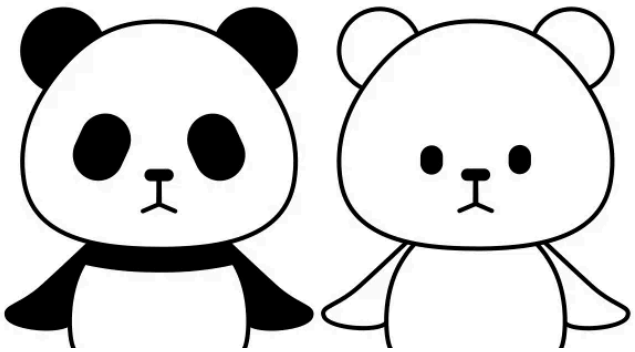


Image by the author

This article was revised based on the comments received from Jakob Ullmann, Dr. Robert Kübler, and Thiago Jaworski on January 19th, 2023. Thank you for your input!

Pandas is an essential Python library for Data Science. But its biggest downside is that it can be slow for operations on large datasets. Polars is a Pandas alternative designed to process data faster.

> Polars is a Pandas alternative designed to process data faster.

This article briefly introduces the Polars Python package and compares it to the popular Data Science library Pandas regarding syntax and speed.

You can find the related code for this article in my Kaggle Notebook.

What is Polars and Why is it Faster Than Pandas?

According to the Polars User Guide [1], its aim "is to provide a lightning-fast DataFrame library that utilizes all available cores on your machine."

In contrast to Polars, Pandas doesn't natively parallelize the processing across the cores of your computer. Other tools like Dask are built upon libraries like Pandas to try to parallelize them. Instead, Polars is designed for parallelization and built from the ground up. Although it is written in Rust, Polars has a Python package, which makes it a potential alternative to Pandas.

Polars has two different APIs: an eager API and a lazy API.

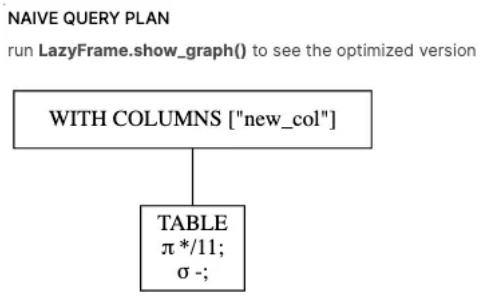
The **eager** execution is similar to Pandas. That means the code is run directly, and its results are returned immediately.

On the other hand, **lazy** execution is not run until you need the results. Because it avoids running unnecessary code, lazy execution can be more efficient than eager execution.

For lazy execution, you must begin your operation with the `.lazy()` method. Then you can write the code for whatever it is you want to do. In the end, you need to run the `.collect()` method to show the results.

```
df.lazy()
  .with_columns([(pl.col("col") * 10).alias("new_col")])
  #...
  .collect()
```

If you don't run the `.collect()` method, the operation is not executed right away. Instead, you will see the execution graph.



Example execution graph of Polars lazy execution (Image by the author via Kaggle)

Benchmark Setup

For benchmarking, we will borrow the benchmark setup from my previous article, which compared different file formats.

How to Handle Large Datasets in Python
A Comparison of CSV, Pickle, Parquet, Feather, and HDF5
towardsdatascience.com

This benchmark setup uses a fictional dataset containing one column of each data type. To reduce timing noise for comparability, this fictional dataset contains 4,000,000 rows and **is almost 1GB large**, as suggested in [2].

	col01	col02	col03	col04	col05	col06	col07	col08	col09	col10	col11
0	102	11589	1895654828	385440915577878085	0.262207	0.407871	0.495690	JJGYe2	7S7Qh2	False	1900-01-01 00:00:00
1	51	16851	1760676504	3329786743902062576	0.489014	0.208688	0.428361	g4zT3T	NsjC8D	False	1900-01-01 00:01:00
2	92	5635	750725288	2810594586424124441	0.067566	0.833029	0.871520	1x3MY7	9vK1Uv	False	1900-01-01 00:02:00
3	14	18750	2117869707	3883745346773623912	0.697754	0.978824	0.340163	KIU372	HzKZk1	False	1900-01-01 00:03:00
4	106	6534	1136936996	3988717031793496799	0.604492	0.511207	0.104959	amVWUu	uzM5wK	False	1900-01-01 00:04:00
...	...	...	...	...	...	...	...	...	...	...	...
3999995	33	19423	1955594573	2762876070612304967	0.880371	0.847194	0.286015	qtlmYP	4ep1fv	False	1907-08-10 18:35:00
3999996	84	27110	509925251	299107579274458622	0.652832	0.252367	0.260360	Ytkbbd	BCYPiy	False	1907-08-10 18:36:00
3999997	48	9947	1021389609	986230688143141101	0.529785	0.700320	0.060581	G1kLsN	zf8kSx	False	1907-08-10 18:37:00
3999998	39	11558	1427174450	6671986351224936730	0.509766	0.278563	0.052728	jGx7vs	GXaaJ8	False	1907-08-10 18:38:00
3999999	94	19471	734623919	222224577011008702	0.856934	0.944471	0.894911	UX0cjf	gLIlPk	False	1907-08-10 18:39:00

4000000 rows × 11 columns

Head of Fictional Dataset for Benchmarking (Image by the author via Kaggle)

In the following, we will time the execution with `%%timeit -n32 -r4`.

Getting Started with Polars

To set up Polars, you simply `pip` install it.

```
pip install polars
```

After that, you can import the Polars Python package just like you would do with Pandas.

```
import polars as pl
import pandas as pd
```

Now, you are all set!

Comparison between Pandas and Polars

At first glance, Pandas and Polars (eager API) are similar regarding syntax because of their shared main building blocks: Series and DataFrames.

Also, many expressions in Polars are similar to Pandas expressions:

```
# Example expressions that work both with Pandas and Polars
df.head() # Get the first n rows
df.tail() # Get the last n rows
df.unique() # Get unique values of this expression.
```

But, according to the Polars User Guide [1], "if your Polars code looks like it could be Pandas code, it might run, but it likely runs slower than it should."

This section explores the main aspects of how the Polars package differs from Pandas regarding syntax and execution time:

- Reading Data
- Selecting and Filtering Data
- Creating New Columns
- Grouping and Aggregation
- Missing Data

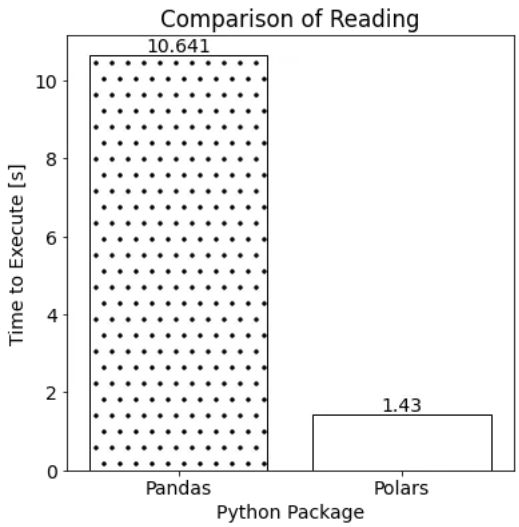
Reading Data

Reading a CSV file in Polars will feel familiar because you can use the `.read_csv()` method like in Pandas:

```
# Pandas
pd.read_csv('example.csv')

# Polars
pl.read_csv('example.csv')
```

The resulting execution times to read the sample dataset in Pandas and Polars are shown below:



Comparison of reading time between Pandas and Polars (Image by the author via Kaggle)

For our sample dataset, reading data takes about e**ight times longer with Pandas** than with Polars.

Selecting and Filtering Data

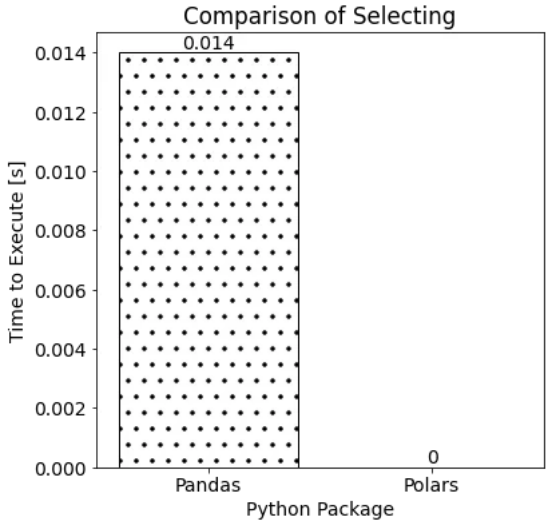
The first major difference between Pandas and Polars is that Polars does not use an index [1]. Instead, each row is indexed by its integer position in the DataFrame [1].

Although the same Pandas code will run with Polars, it is not the best practice. In Polars, you should use the `.select()` method to select data.

```
# Pandas
df[['col1', 'col2']]

# The above code will run with Polars as well,
# but the correct way in Polars is:
df.select(pl.col(['col1', 'col2']))
```

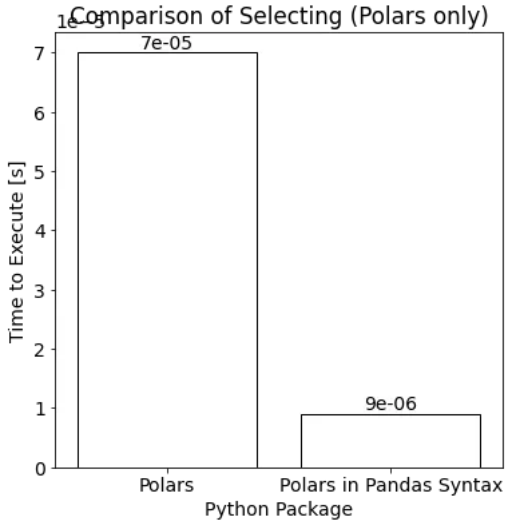
The resulting execution times to select data in Pandas and Polars are shown below:



Comparison of selecting time between Pandas and Polars (Image by the author via Kaggle)

For our sample dataset, selecting data takes about **15 times longer with Pandas** than with Polars (~70.3 µs).

Below you can see a comparison of the Polars operation in the syntax suggested in the documentation (using `.select()`, left) and in the Pandas syntax (using `df[['col1', 'col2']]`, right). *Unexpectedly*, the Pandas syntax is much faster than the suggested `.select()` method.



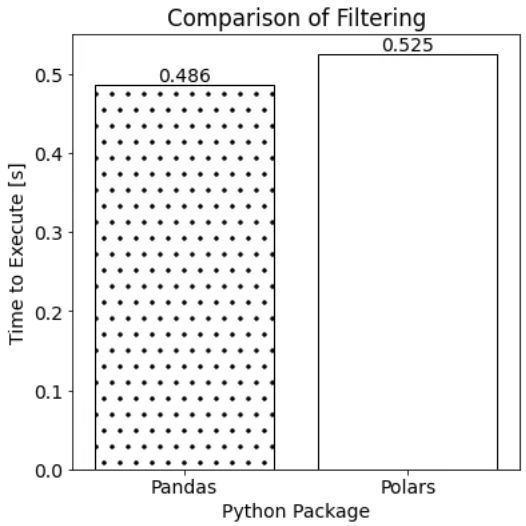
Comparison of selecting time between Polars and Polars with Pandas Syntax (df[['col1', 'col2']]) (Image by the author via Kaggle)

While you would use the `.query()` method in Pandas to filter data, you need to use the `.filter()` method in Polars.

```
# Pandas
df.query('col1 > 5')

# Polars
df.filter(pl.col('col') > 5)
```

The resulting execution times to filter data in Pandas and Polars are shown below:



Comparison of filtering time between Pandas and Polars (Image by the author via Kaggle)

For our sample dataset, filtering a dataframe takes a **similar amount of time** in Pandas and Polars.

In contrast to Pandas, Polars can run operations in `.select()` and `.filter()` in parallel.

Creating New Columns

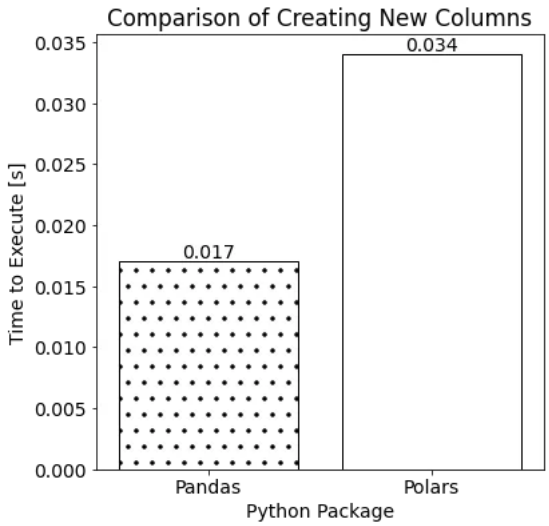
Creating a new column in Polars also differs from what you might be used to in Pandas. In Polars, you need to use the `.with_column()` or the `.with_columns()` method depending on how many columns you want to create.

```
# Pandas
df_pd["new_col"] = df_pd["col"] * 10

# Polars
df.with_columns([(pl.col("col") * 10).alias("new_col")])
```

```
# Polars for multiple columns
# df.with_columns([(pl.col("col") * 10).alias("new_col"), ...])
```

The resulting execution times to create a new column in Pandas and Polars are shown below:



Comparison of time to create a new column between Pandas and Polars (Image by the author via Kaggle)

For our sample dataset, creating a new column with **Polars takes roughly two times longer** than with Pandas.

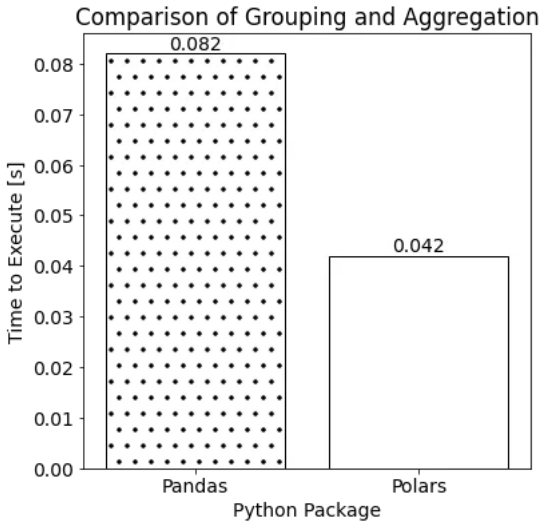
Grouping and Aggregation

Grouping and aggregation are slightly different between Pandas and Polars syntax-wise, but both use the `.groupby()` and `.agg()` methods.

```
# Pandas
df_pd.groupby('col1')['col2'].agg('mean')

# Polars
# df.groupby('col1').agg([pl.col('col2').mean()]) # As suggested in Polars docs
df.groupby('col1').agg([pl.mean('col2')]) # Shorter
```

The resulting execution times to group and aggregate data in Pandas and Polars are shown below:



Comparison of aggregation time between Pandas and Polars (Image by the author via Kaggle)

For our sample dataset, aggregating data takes about **two times longer with Pandas** than with Polars.

Missing Data

Another major difference between Pandas and Polars is that Pandas uses NaN values to indicate missing values, while Polars uses null [1].

	col1	col2	col3
0	1	NaN	3
1	4	5.0	6
2	7	NaN	9

Pandas DataFrame with missing values

col1	col2	col3
i64	i64	i64
1	null	3
4	5	6
7	null	9

Polars DataFrame with missing values

How Pandas and Polars indicate missing values in DataFrames (Image by the author)

Thus, instead of the `.fillna()` method in Pandas, you should use the `.fill_null()` method in Polars.

```
# Pandas
df['col2'].fillna(-999)

# Polars
# df_pd.with_column(pl.col('col2').fill_null(pl.lit(-999))) # As suggested in Po
df_pd.with_column(pl.col('col2').fill_null(-999)) # Shorter
```

Conclusion

Now, is Polars better than Pandas? Will Polars replace Pandas?

The main advantage of Polars over Pandas is its speed. If you need to do a lot of data processing on large datasets, you should definitely try Polars.

> The main advantage of Polars over Pandas is its speed.

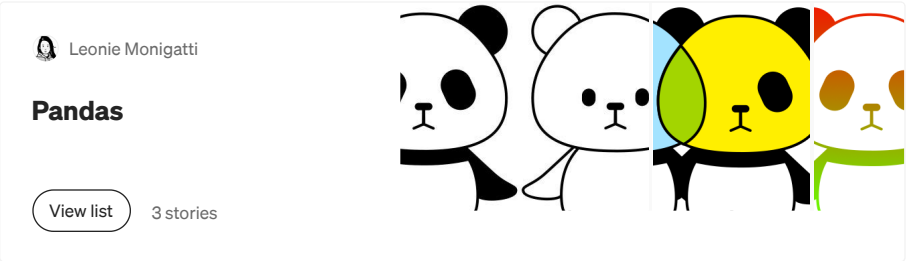
But as shown in this article, you would have to learn the new Polars syntax to switch from Pandas to Polars. Also, you have seen that the Polars code is usually a little longer than the Pandas code for the same operation. Last but not least, Polars doesn't cover the full range of functionality that Pandas has, e.g., for data exploration.

> Polars code is usually a little longer than the Pandas code

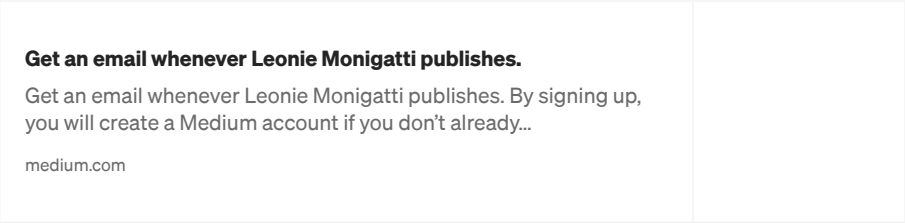
For further discussion, please refer to the comment section where more experienced Polars users have listed their insights.

Enjoyed This Story?

If you liked this article, you might also enjoy my other Pandas articles.



Subscribe for free to get notified when I publish a new story.



Find me on LinkedIn, Twitter, and Kaggle!

References

[1] Polars (2023): User Guide https://pola-rs.github.io/polars-book/user-guide/ (accessed 8. January 2023)

[2] “Stackoverflow”, “What are the differences between feather and parquet?”. stackoverflow.com. https://stackoverflow.com/questions/48083405/what-are-the-differences-between-feather-and-parquet (accessed July 25, 2022)

Data Science | Artificial Intelligence | Pandas | Python | Programming

Written by Leonie Monigatti

Follow

33K Followers · Writer for Towards Data Science

Developer Advocate @ Weaviate. Follow for practical data science guides - whether you're a data scientist or not. linkedin.com/in/804250ab

Leonie Monigatti in Towards Data Science

Evaluating RAG Applications with RAGAs

A framework with metrics and LLM-generated data to evaluate the performance...

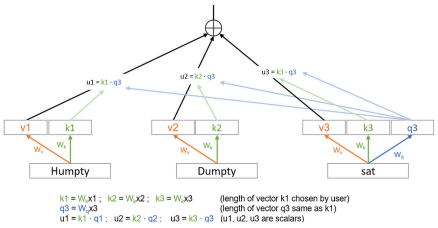
Dec 13, 2023 1.5K 9

Shaw Talebi in Towards Data Science

5 AI Projects You Can Build This Weekend (with Python)

From beginner-friendly to advanced

Oct 9 3.5K 60

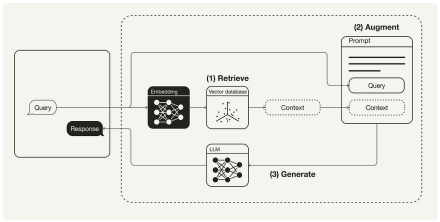


Rohit Patel in Towards Data Science

Understanding LLMs from Scratch Using Middle School Math

In this article, we talk about how LLMs work, from scratch—assuming only that you know...

Oct 19 1.8K 19



Leonie Monigatti in Towards Data Science

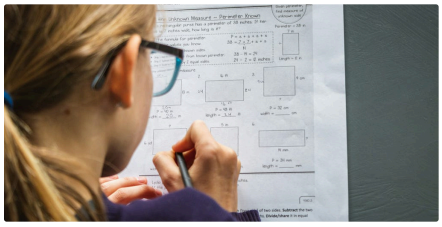
Retrieval-Augmented Generation (RAG): From Theory to LangChain...

From the theory of the original academic paper to its Python implementation with...

Nov 14, 2023 1.7K 14

See all from Leonie Monigatti | See all from Towards Data Science

Recommended from Medium



Egor Howell in Towards Data Science

4 Years of Data Science in 8 Minutes

What I have learned in my 4+ year journey of studying data science

5d ago 825 13

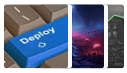
Abdur Rahman in Stackademic

Python is No More The King of Data Science

5 Reasons Why Python is Losing Its Crown

Oct 22 390 10

Lists



Predictive Modeling w/ Python
20 stories · 1626 saves

Coding & Development
11 stories · 878 saves

ChatGPT
21 stories · 853 saves

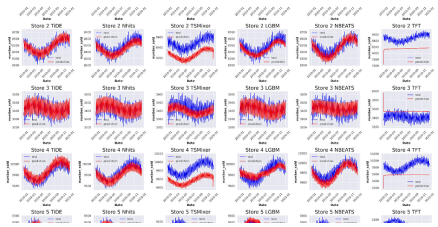
ChatGPT prompts
50 stories · 2157 saves

Bryson Meiling in Python in Plain English

What is Pydantic and how can it help your next python project?

For anyone that has been either living under a programming rock you may not have heard ...

Oct 6 467 9

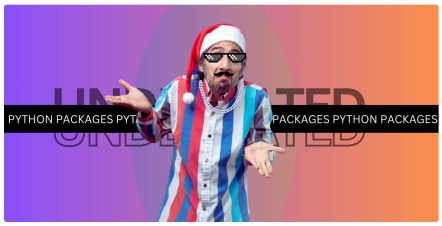


Esther Cifuentes

Comparing Time Series Algorithms

Evaluating Leading Time Series Algorithm with Darts.

Oct 16 83

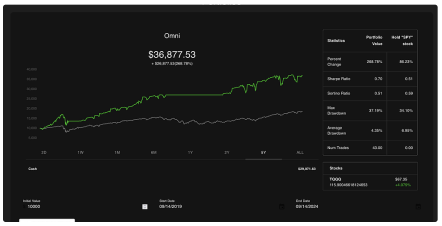


Abhay Parashar in The Pythoneers

23 Game-Changing Python Packages You Are Missing Out On

Make Your Life Easy By Exploring These Hidden Gems

Oct 21 1.3K 9



Austin Starks in DataDrivenInvestor

I used OpenAI's o1 model to develop a trading strategy. It is...

It literally took one try. I was shocked.

Sep 15 5K 132

See more recommendations