



Сын маминой подруги: может ли Polars заменить Pandas?



Data Fest²⁰²⁴



МИХАИЛ АРХИПОВ

Мир Plat.Form



ПАВЕЛ ЦВЕТОВ

Мир Plat.Form

Секция: Open Source
27 мая 2024 года

Мир Plat.Form – ЦТА



Подразделение

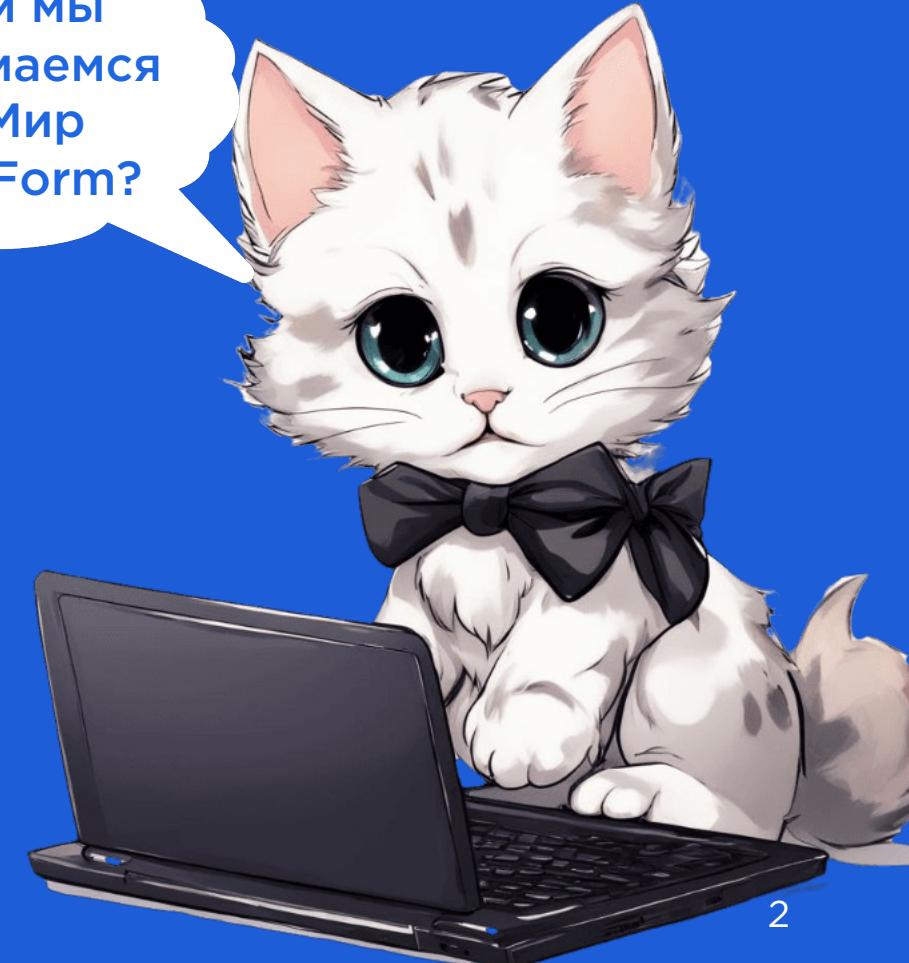
Центр транзакционной аналитики

Задачи

- Анализ больших данных
- Визуализация данных
- Разработка ETL-процессов
- Обучение и деплой ML-моделей
- RnD-задачи



Чем мы
занимаемся
в Мир
Plat.Form?



Наш стек технологий:



Airflow



pandas



PyTorch



Yandex
CatBoost



learn



NumPy



FineBI



Agenda

› Polars

Бенчмарки

Концепты в Polars

Синтаксис Polars

Использование памяти

Lazy API

Polars и ML

ПРЕРЕКВИЗИТЫ



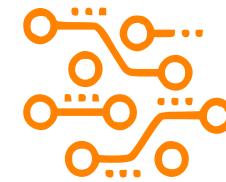
или



или



- ❑ И хорошее настроение!



Pandas – инструмент мощный,
С ним DataFrames создаем.
Анализировать данные
помогает,
В Python-е он незаменим.

Spark – система распределенных вычислений,
Для анализа служит, не боится сложностей.
Большие данные ему ни почем,
Он ускоряет процессы – и ночью, и днем.

Язык SQL так много значит для меня,
Его команды, словно танец у огня.
Они сплетаются в понятный алгоритм,
И в этом всем я вижу только позитив!



ЧТО-ТО ЕЩЕ?



- Быстрая и эффективная библиотека для обработки данных
- Ядро расчетов написано на Rust
- API для Rust, Python, R и NodeJS
- Open Source (MIT лицензия)

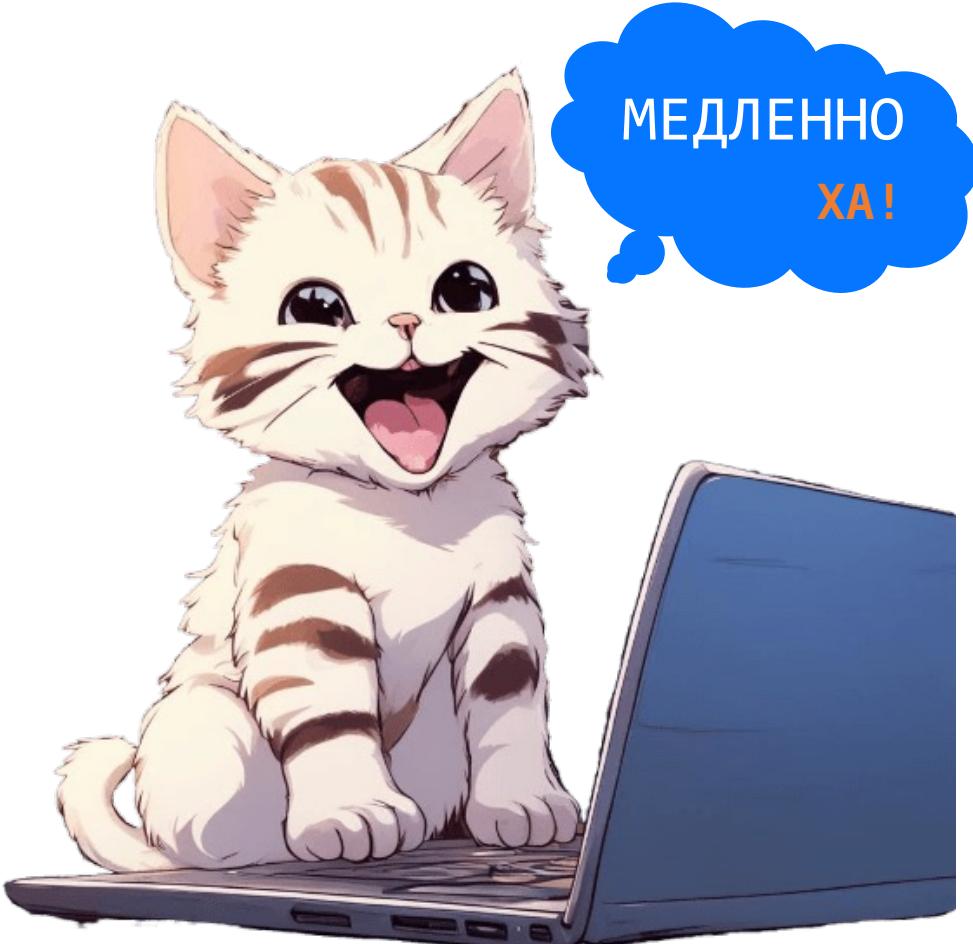
DataFrames for
the new era 

40M+ Downloads to date 26K Github stars

<https://pola.rs>



УТИЛИЗАЦИЯ СРУ



POLARS

0[99.4%
1[98.9%
2[99.4%
3[99.4%
4[99.4%
5[98.9%
6[98.9%
7[98.3%
8[99.4%
9[98.9%
10[98.3%
11[98.9%
12[99.4%
13[99.4%
14[98.9%
15[99.4%
16[99.4%
17[99.4%
18[99.4%
19[98.9%
20[99.4%
21[99.4%
22[98.9%
23[99.4%
24[99.4%
25[92.0%
26[99.4%
27[99.4%
28[99.4%
29[98.9%
30[98.3%
31[98.9%
32[98.9%
33[98.9%
34[99.4%
35[99.4%
36[99.4%
37[97.2%
38[99.4%
39[98.3%
40[95.5%
41[98.9%
42[98.9%
43[98.9%
44[99.4%
45[99.4%
46[100.0%
47[99.4%
48[99.4%
49[98.9%
50[99.4%
51[99.4%
52[98.9%
53[98.3%
54[97.1%
55[98.9%
56[98.9%
57[98.9%
58[98.9%
59[99.4%
60[99.4%
61[99.4%
62[99.4%
63[98.9%

PANDAS

0[0.6%
1[7.9%
2[8.5%
3[7.9%
4[0.0%
5[0.0%
6[0.6%
7[0.0%
8[0.0%
9[1.2%
10[0.0%
11[7.4%
12[0.0%
13[0.0%
14[0.0%
15[8.0%
16[0.0%
17[0.0%
18[0.0%
19[0.0%
20[0.0%
21[0.0%
22[0.0%
23[0.0%
24[0.6%
25[0.0%
26[0.0%
27[0.6%
28[0.6%
29[100.0%
30[0.0%
31[0.0%
32[0.6%
33[0.6%
34[8.0%
35[8.0%
36[0.6%
37[0.0%
38[0.0%
39[0.6%
40[8.5%
41[0.6%
42[0.0%
43[0.0%
44[0.0%
45[0.0%
46[0.6%
47[0.0%
48[7.9%
49[0.0%
50[0.0%
51[1.8%
52[0.0%
53[1.2%
54[0.0%
55[0.0%
56[0.0%
57[0.0%
58[0.0%
59[0.0%
60[0.0%
61[0.0%
62[0.0%
63[0.6%

Polars

› Бенчмарки

Концепты в Polars

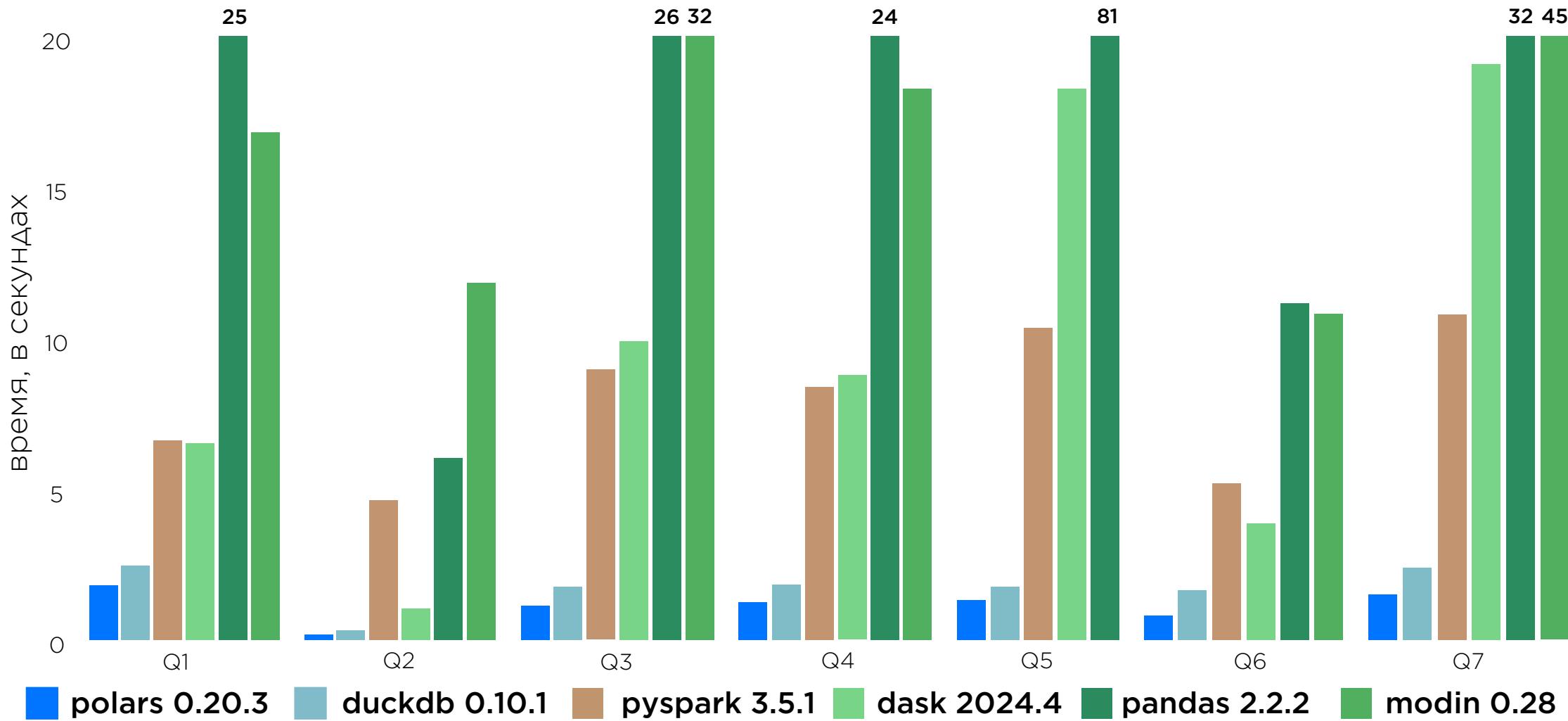
Синтаксис Polars

Использование памяти

Lazy API

Polars и ML

БЕНЧМАРК ТРС-Н'24



СОБСТВЕННЫЙ БЕНЧМАРК

int_1	int_2	int_3	int_4	float_1	float_2	bool	date	datetime	name
u32	u64	i32	i64	f32	f64	bool	date	datetime[ns]	str
16609	35812027	-7917	74732678	-7308.70996	8.6001e7	false	2024-05-27	2024-05-27 09:30:05	Victor
73457	14332514	35535	-2140453	6454.02241	9.9425e6	true	2024-05-27	2024-05-27 11:47:22	Stacy
99417	57378795	-69656	-69230944	-23446.21875	-3.1343e5	false	2024-05-27	2024-05-27 15:55:59	Paolo
27692	3803131	82179	18715942	42818.49218	-7.5345e8	true	2024-05-27	2024-05-27 17:00:00	Michael

100 млн. строк, ~5 Gb RAM

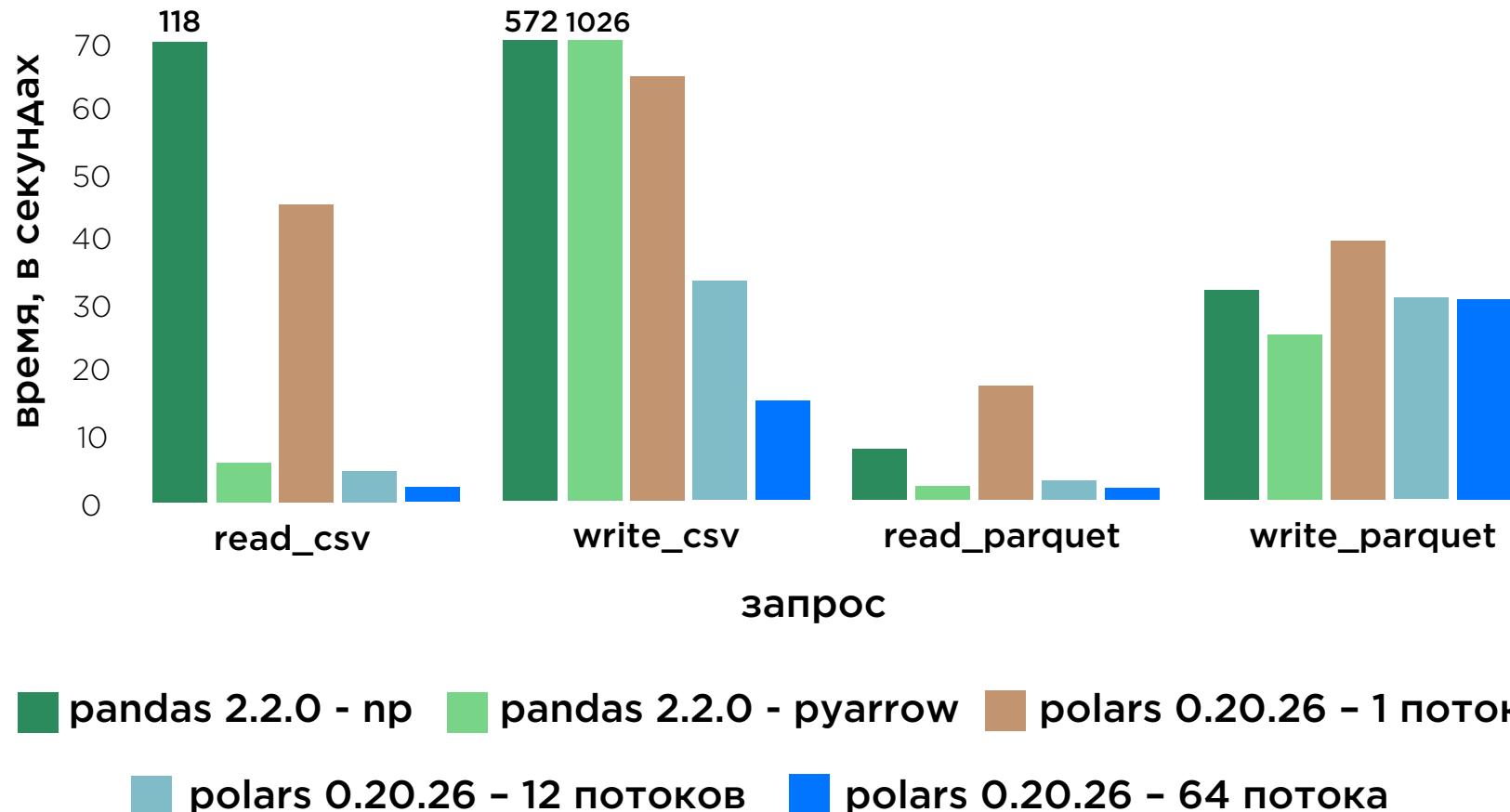
Были проведены замеры скоростей:

- Pandas v.2.2 NumPy бэкенд
- Pandas v.2.2 Pyarrow бэкенд
- Polars 0.20.26 - 1 поток
- Polars 0.20.26 - 12 потоков
- Polars 0.20.26 - 64 потока

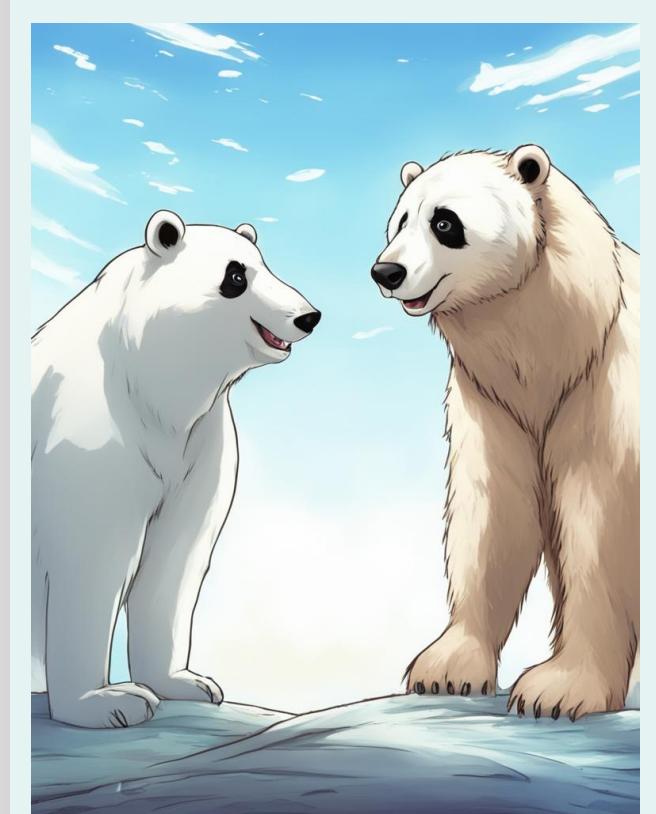


ВСЕГДА ПОБЕЖДАЮ ?

ЧТЕНИЕ И ЗАПИСЬ

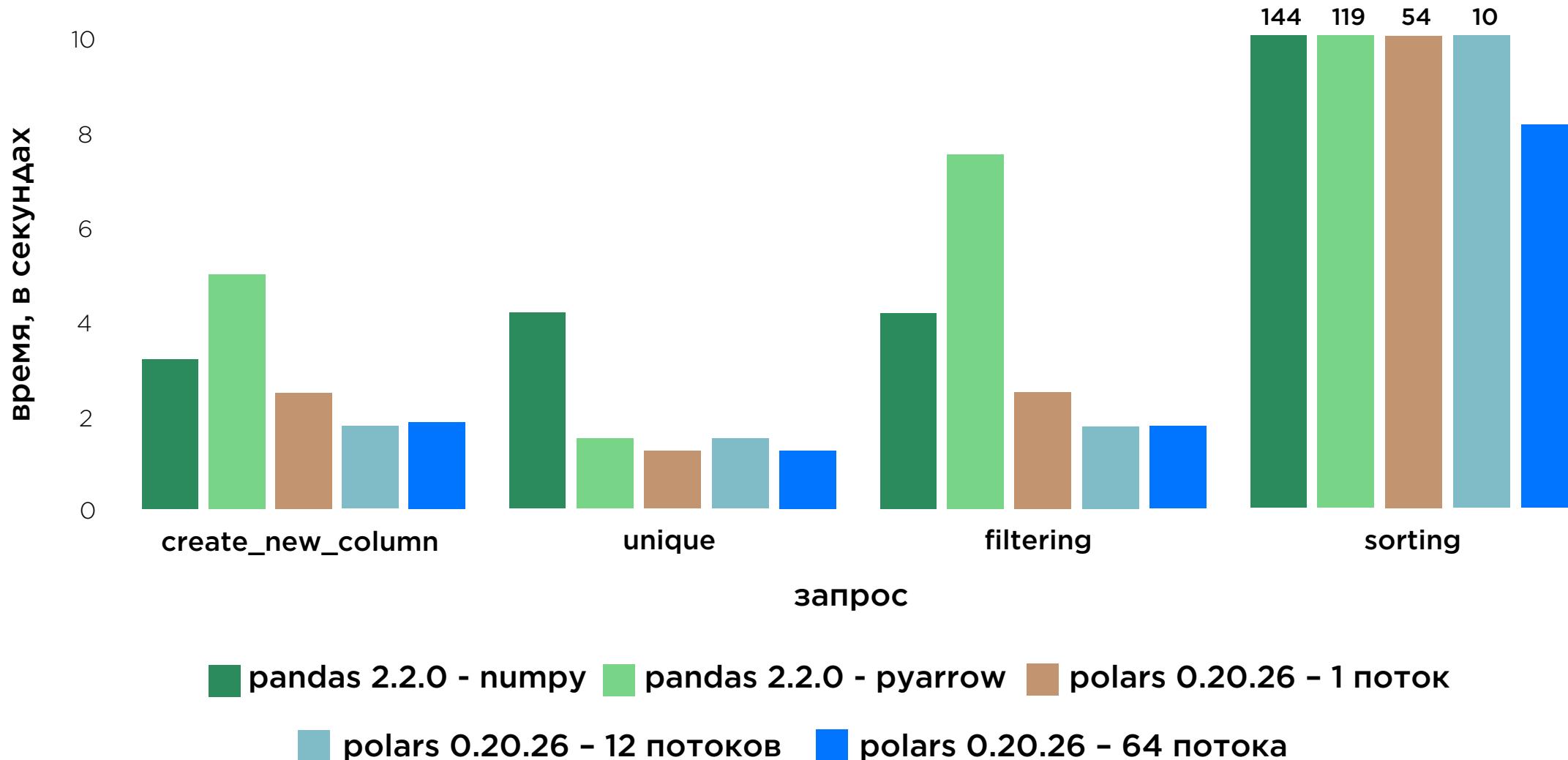


POLARS И PANDAS

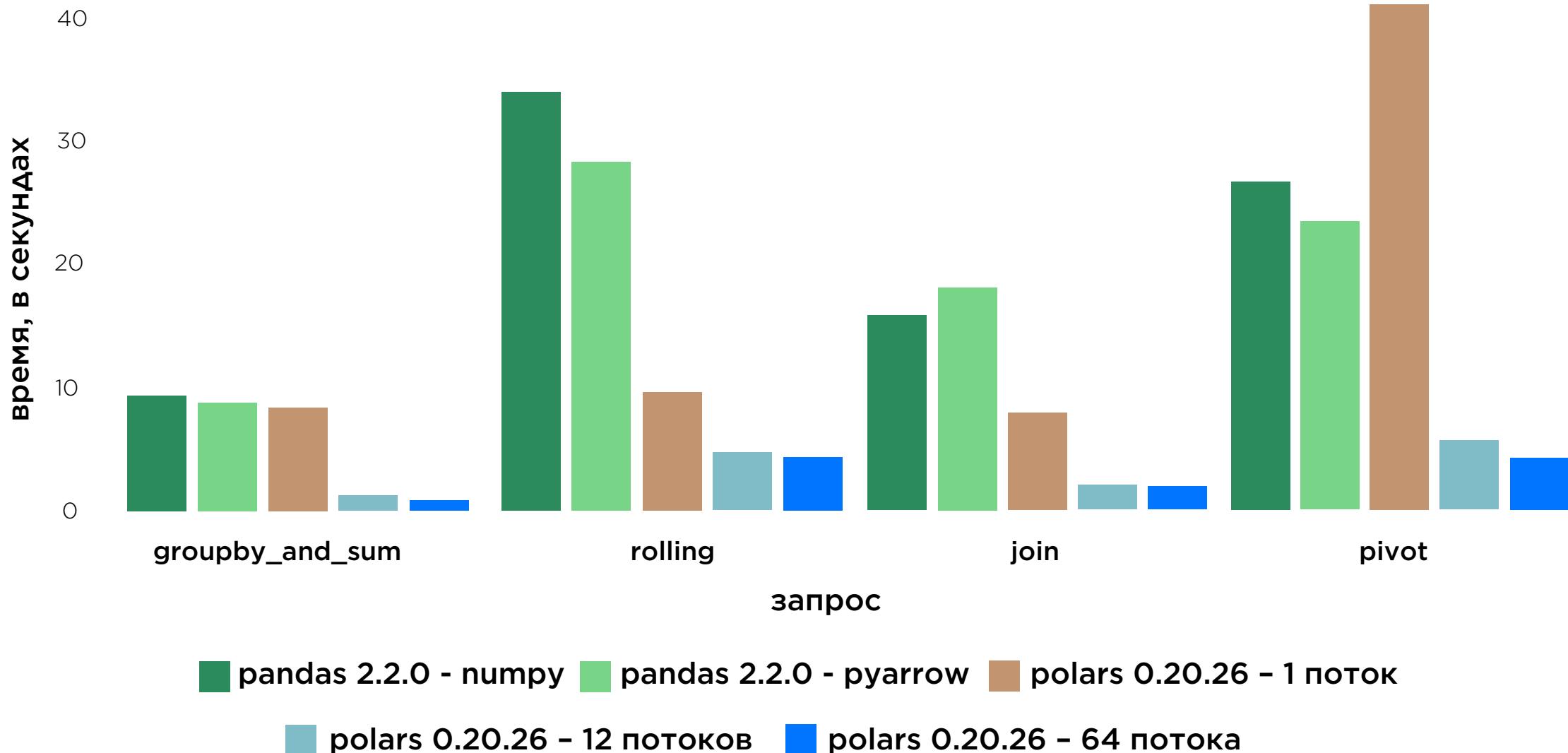


ПАНДУ ВЫРУЧАЮ

ПРОСТЫЕ ЗАПРОСЫ



СЛОЖНЫЕ ЗАПРОСЫ



Polars

Бенчмарки

› Концепты в Polars

Синтаксис Polars

Использование памяти

Lazy API

Polars и ML

ДЕФОЛТНЫЕ ИМОРТЫ



```
import polars as pl
import polars.selectors as cs
```

APACHE ARROW



**Формат хранения данных
в оперативной памяти**

**Open Source
Колоночный
Мультиязычный
Стандартизованный
Мультиплатформенный**

	session_id	timestamp	source_ip
Row 1	1331246660	5/25/2024 2:44 PM	99.155.155.225
Row 2	1331246351	5/25/2024 2:38 PM	65.87.165.114
Row 3	1331244570	5/25/2024 2:09 PM	71.10.106.181
Row 4	1331261196	5/25/2024 6:46 PM	76.102.156.138

	Arrow Memory Buffer	Traditional Memory Buffer
Row 1	1331246660	1331246660
Row 2	5/25/2024 2:44 PM	5/25/2024 2:44 PM
Row 3	99.155.155.225	99.155.155.225
Row 4	1331246351	1331246351
Row 5	5/25/2024 2:38 PM	5/25/2024 2:38 PM
Row 6	65.87.165.114	65.87.165.114
Row 7	1331244570	1331244570
Row 8	5/25/2024 2:09 PM	5/25/2024 2:09 PM
Row 9	71.10.106.181	71.10.106.181
Row 10	1331261196	1331261196
Row 11	5/25/2024 6:46 PM	5/25/2024 6:46 PM
Row 12	76.102.156.138	76.102.156.138

APACHE ARROW



Позволяет без затрат по памяти конвертировать DataFrame между Polars и Pandas

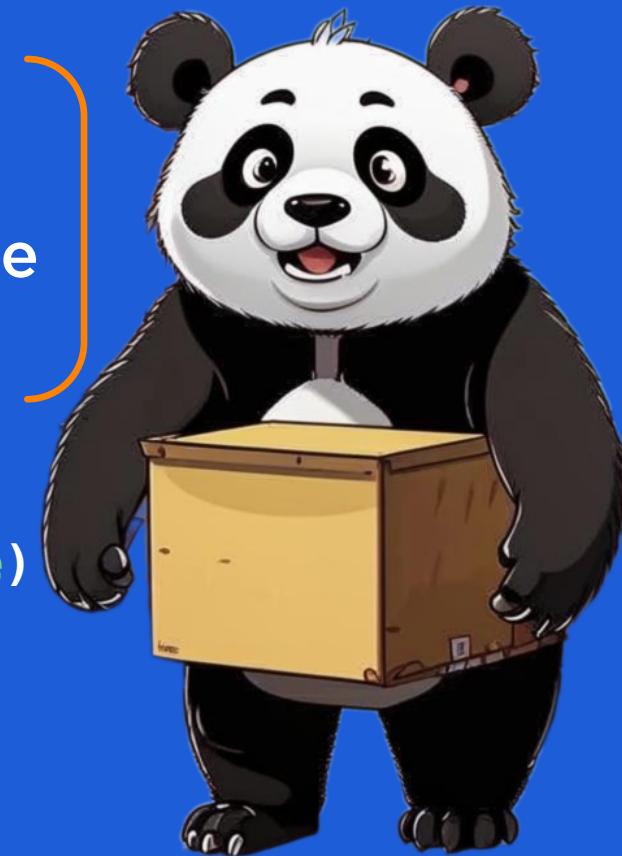
.from_pandas()

Polars
DataFrame

Pandas
DataFrame



.to_pandas(use_pyarrow_extension_array=True)



DSL POLARS

Domain Specific Language для трансформаций данных в Polars



EXPRESSION API

- Fn(Series) -> Series
- Method Chaining
- Abstract transformation



ВЫРАЖЕНИЯ



$Fn(Series) \rightarrow Series$

s

rub
i64
100
25
25
10000

s.sort()

rub
i64
25
25
100
10000

s.sqrt()

rub
f64
10.0
5.0
5.0
100.0

s.cum_sum()

rub
i64
100
125
150
10150

ВЫРАЖЕНИЯ

Method Chaining

s

rub
i64
100
25
25
10000

s.sort()

rub
i64
25
25
100
10000

s.sort().sqrt()

rub
f64
5.0
5.0
10.0
100.0

s.sort().sqrt().cum_sum()

rub
f64
5.0
10.0
20.0
120.0

ВЫРАЖЕНИЯ

Abstract Transformation

```
pl.col("rub").sort().sqrt().cum_sum()
```

- Fn(DataFrame) -> DataFrame
- Method Chaining
- Lazy transformations



КОНТЕКСТ

$\text{Fn(DataFrame)} \rightarrow \text{DataFrame}$

- Выборка: `df.select(...)`
- Добавление колонки: `df.with_columns(...)`
- Фильтрация: `df.filter(...)`
- Группировка / агрегация: `df.group_by(...).agg(...)`

КОНТЕКСТ: ПРИМЕР



card	ps	rub	type
u64	str	u64	str
1	МИР	100	Покупка
1	МИР	25	Перевод
2	Не МИР	25	Покупка
2	Не МИР	10000	АТМ

```
expr = (pl.col("rub") > 25).alias("rub_>_25")
```

КОНТЕКСТ: SELECT

`df.select(expr)`

card	ps	rub	type
u64	str	u64	str
1	МИР	100	Покупка
1	МИР	25	Перевод
2	Не МИР	25	Покупка
2	Не МИР	10000	АТМ



rub > 25
bool
true
false
false
true



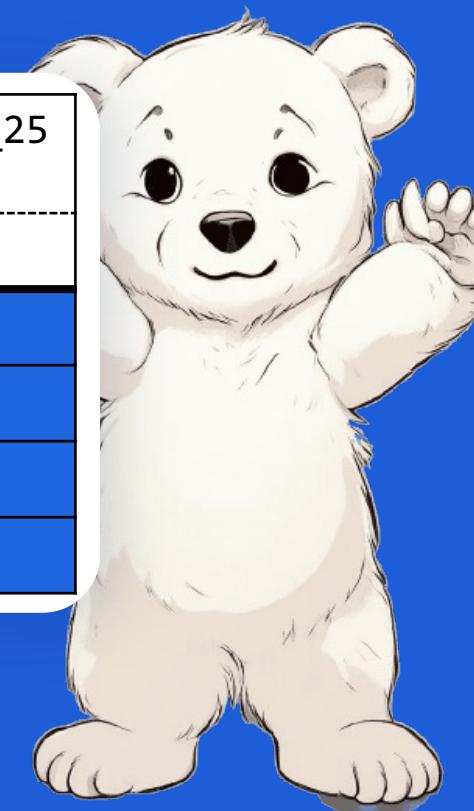
КОНТЕКСТ: WITH_COLUMNS

`df.with_columns(expr)`

card	ps	rub	type
u64	str	u64	str
1	МИР	100	Покупка
1	МИР	25	Перевод
2	Не МИР	25	Покупка
2	Не МИР	10000	АТМ



card	ps	rub	type	rub_>_25
u64	str	u64	str	bool
1	МИР	100	Покупка	true
1	МИР	25	Перевод	false
2	Не МИР	25	Покупка	false
2	Не МИР	10000	АТМ	true



КОНТЕКСТ: FILTER 

df.filter(expr)

card	ps	rub	type
u64	str	u64	str
1	МИР	100	Покупка
1	МИР	25	Перевод
2	Не МИР	25	Покупка
2	Не МИР	10000	АТМ



card	ps	rub	type
u64	str	u64	str
1	МИР	100	Покупка
2	Не МИР	10000	АТМ



КОНТЕКСТ: GROUP_BY

`df.group_by(expr).agg("card", "type")`

card	ps	rub	type
u64	str	u64	str
1	МИР	100	Покупка
1	МИР	25	Перевод
2	Не МИР	25	Покупка
2	Не МИР	10000	АТМ



rub_>_25	card	type
bool	list[u64]	list[str]
false	[1, 2]	["Перевод", "Покупка"]
true	[1, 2]	["Покупка", "АТМ"]



КОНТЕКСТ

Method Chaining

```
df.with_columns(expr)
```

```
df.with_columns(expr)\n    .filter(pl.col("ps") ==\n        "МИР")
```

```
df.with_columns(expr)\n    .filter(pl.col("ps") == "МИР")\n    .group_by("card").agg("rub_>_25")
```

card	ps	rub	type	rub_>_25
u64	str	u64	str	bool
1	МИР	100	Покупка	true
1	МИР	25	Перевод	false
2	Не МИР	25	Покупка	false
2	Не МИР	10000	АТМ	true

card	ps	rub	type	rub_>_25
u64	str	u64	str	bool
1	МИР	100	Покупка	true
1	МИР	25	Перевод	false

card	rub_>_25
u64	list[bool]
1	[true, false]

LAZY TRANSFORMATIONS

DataFrame.**lazy()** → LazyFrame

LAZY TRANSFORMATIONS

lf = df.lazy()

```
DF[  
    "card", "ps",  
    "rub", "type"  
];  
PROJECT */4 COLUMNS;  
SELECTION: "None"
```

Накопленные
трансформации



card	ps	rub	type
u64	str	u64	str
1	МИР	100	Покупка
1	МИР	25	Перевод
2	Не МИР	25	Покупка
2	Не МИР	10000	АТМ

Ссылка на данные

LAZY TRANSFORMATIONS

lf.with_columns(expr)

```
WITH_COLUMNS: [
    [(col("rub")) > (25)].alias("rub_>_25")
]
```

2

```
DF ["card", "ps", "rub", "type"];
PROJECT */4 COLUMNS;
SELECTION: None
```

1

LAZY TRANSFORMATIONS

```
lf.with_columns(expr)\\
    .filter(pl.col("ps") == "МИР")
```

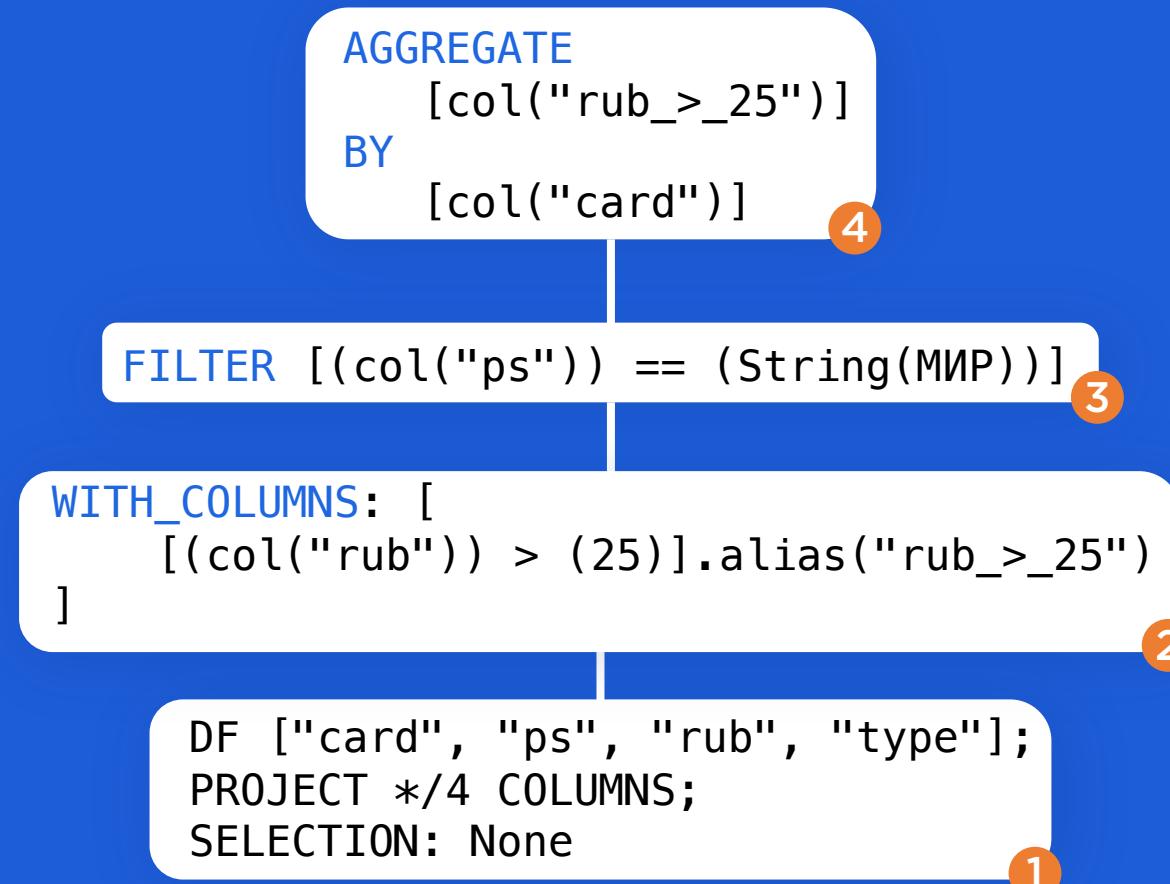
FILTER [(col("ps")) == (String(МИР))] 3

WITH_COLUMNS: [
 [(col("rub")) > (25)].alias("rub_>_25")
]

DF ["card", "ps", "rub", "type"];
PROJECT */4 COLUMNS;
SELECTION: None 1

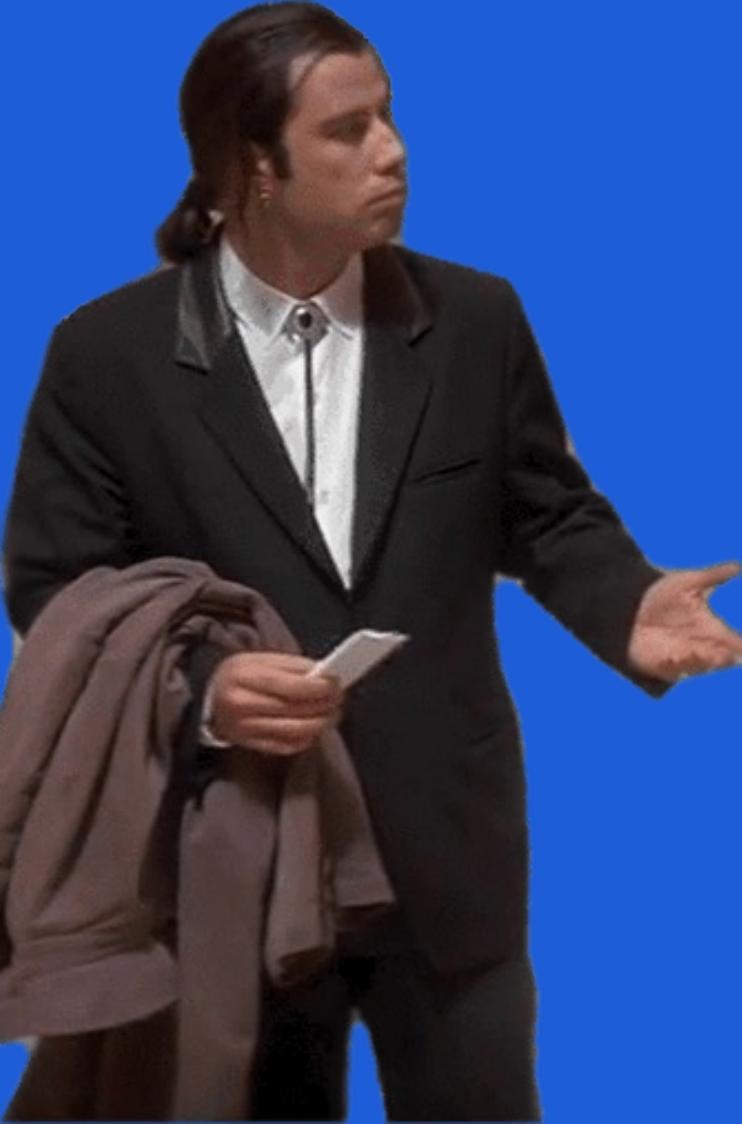
LAZY TRANSFORMATIONS

```
lf.with_columns(expr)\n    .filter(pl.col("ps") == "МИР")\n    .group_by("card").agg("rub_>_25")
```



ИНДЕКСЫ

PANDAS



```
import pandas as pd

# класс Индекса
pd.Index

# класс МультиИндекса
pd.MultiIndex

# Индексация
df.iloc[0, :]

# Сброс индекса
df.reset_index(drop=True)
```

ИНДЕКСЫ

POLARS

«Правда в том, что нет никакого ложки индекса»



`pip install polars`

`pip install polars-u64-idx`

Polars

Бенчмарки

Концепты в Polars

› Синтаксис Polars

Использование памяти

Lazy API

Polars и ML

POLARS: SERIES

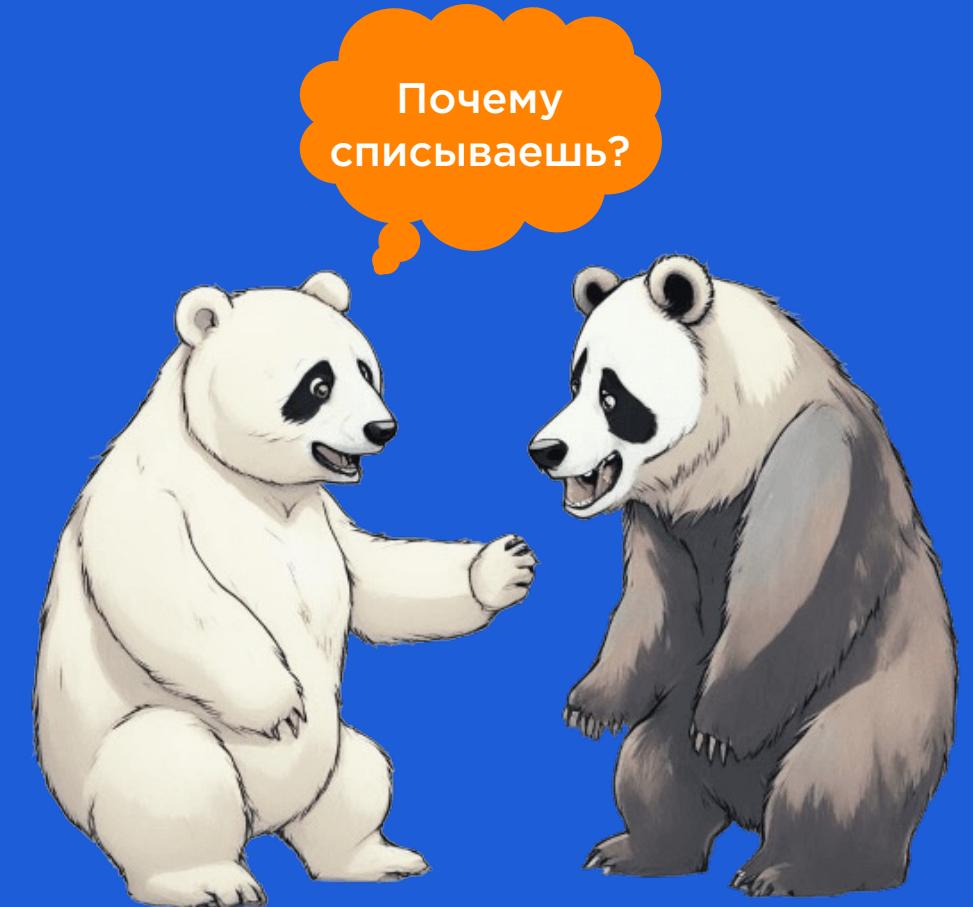
Сравнение инициализации класса Series

Pandas:

```
series_pd = pd.Series(  
    data=range(1,9),  
    name="ps_id"  
)
```

Polars:

```
series_pl = pl.Series(  
    values=range(1,9),  
    name="ps_id"  
)
```



POLARS: DATAFRAME

Сравнение инициализации класса DataFrame

Pandas:

```
df_pd = pd.DataFrame(  
    {  
        "ps_id": [1, 1, 2, 2],  
        "rub": [10, 25, 25, 10000],  
        "type": ["Покупка", "Перевод", "Покупка", "АТМ"]  
    }  
)
```

Polars:

```
df_pl = pl.DataFrame(  
    {  
        "ps_id": [1, 1, 2, 2],  
        "rub": [10, 25, 25, 10000],  
        "type": ["Покупка", "Перевод", "Покупка", "АТМ"]  
    }  
)
```



POLARS: READ/WRITE



Polars

```
data = pl.read_csv(  
    source="data_polars.csv"  
)
```

```
data = pl.write_csv(  
    file="data_polars.csv"  
)
```

Pandas

```
data = pd.read_csv(  
    path="data_pandas.csv"  
)
```

```
data = pd.to_csv(  
    path="data_pandas.csv"  
)
```

ДОСТУП к ДАННЫМ



Существует два способа доступа к данным таблиц:

1

Квадратные скобки []

- Просмотр значений в колонках, строках или ячейках
- Преобразование колонки `DataFrame` в `Series`

```
df["rub"]
```

2

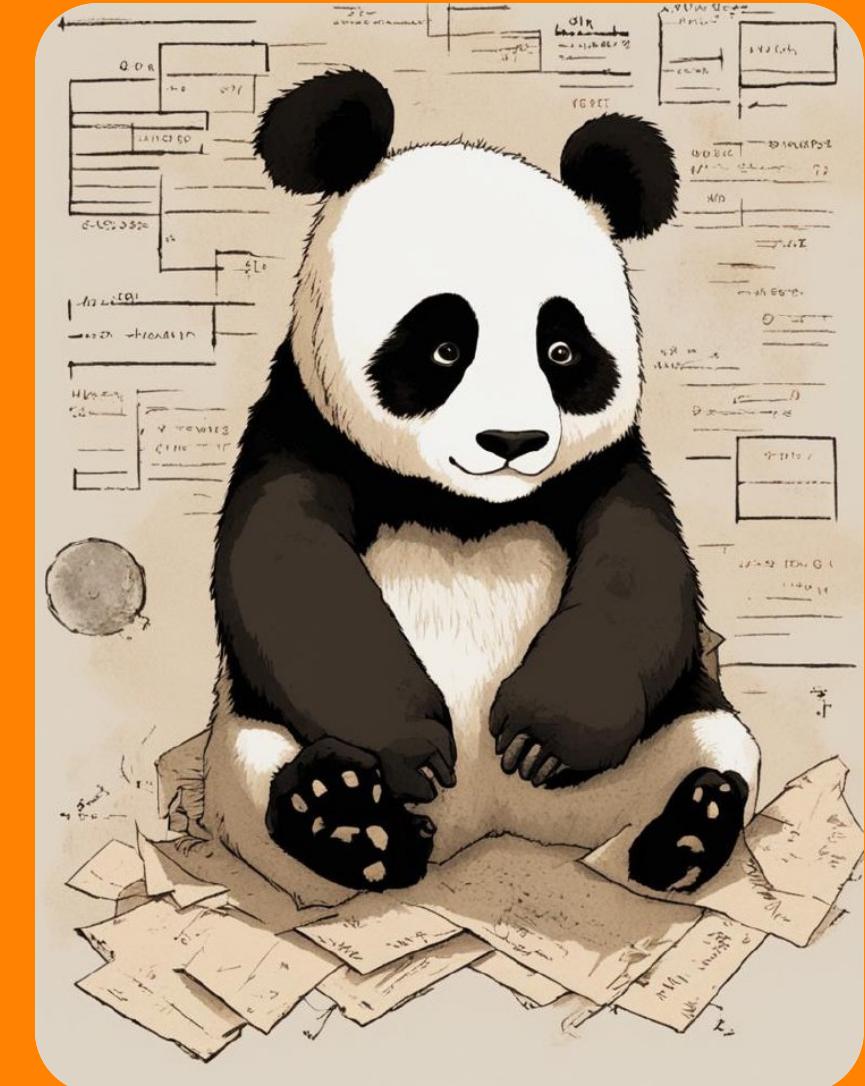
.filter(), .select()

- Контекстные операции выполняются параллельно
- Контекстные операции могут быть оптимизированы в Lazy API

```
df.filter(pl.col("rub") > 25)
```

СХОЖЕСТЬ

DataFrame	.value_counts()	.rolling()
Series	.read_parquet()	...
.shape	.min()	
.dtypes	.max()	
.columns	.sum()	
.describe()	.count()	
.head()	.mean()	
.tail()	.pivot()	
.drop()	.cut()	
.rename()	.melt()	



РАЗЛИЧИЯ

Pandas

.loc[]
.iloc[...]
[:, :]
.query()
.to_parquet()
.assign()
.as_type()

Polars

.select().filter()
.filter()
.write_parquet()
.with_columns()
.cast()
.group_by()
.is_in()
.sort()
.map_rows()
.fill_null()



POLARS: SELECTORS

Polars предоставляет удобный интерфейс по отбору нескольких колонок:

Все колонки

```
df.select(cs.all())
```

Все целочисленные колонки

```
df.select(cs.integer())
```

Все колонки с датой

```
df.select(cs.date())
```

С использованием regex выражений

```
df.select(cs.matches("ps\\|type"))
```

С использованием Str паттерна

```
df.select(cs.starts_with("rub"))
```

```
df.select(cs.contains("amount"))
```

Polars

Бенчмарки

Концепты в Polars

Синтаксис Polars

› Использование памяти

Lazy API

Polars и ML

КОПИИ и ПРЕДСТАВЛЕНИЯ

PANDAS

View

	A	B
0	1	2
1	3	4
2	5	6

←df2

df1

Copy

	A	B
0	1	2
1	3	4
2	5	6

df1

	A	B
0	1	2
1	3	4
2	5	6

df2

КОПИИ и ПРЕДСТАВЛЕНИЯ PANDAS

Modifying a View

	A	B
0	1	2
1	3	5
2	5	6

df1

←df2

	A	B
0	1	2
1	3	4
2	5	6

df1

	A	B
0	1	2
1	3	5

df2

КОПИИ и ПРЕДСТАВЛЕНИЯ PANDAS

SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:

https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

КОПИИ и ПРЕДСТАВЛЕНИЯ PANDAS

Когда возвращается копия, а когда представление?

Разработчики говорят:

“The exact behaviour is hard to predict”

Решение:

`df = df.copy()`

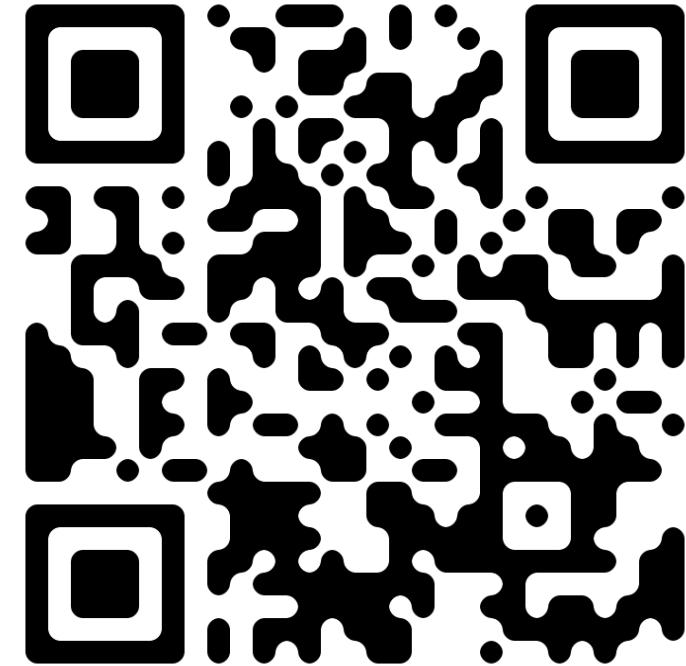
НОВЫЙ COPY ON WRITE ПОДХОД

Появился в 1.5

Полностью реализовали в 2.1

Работает по умолчанию с 3.0

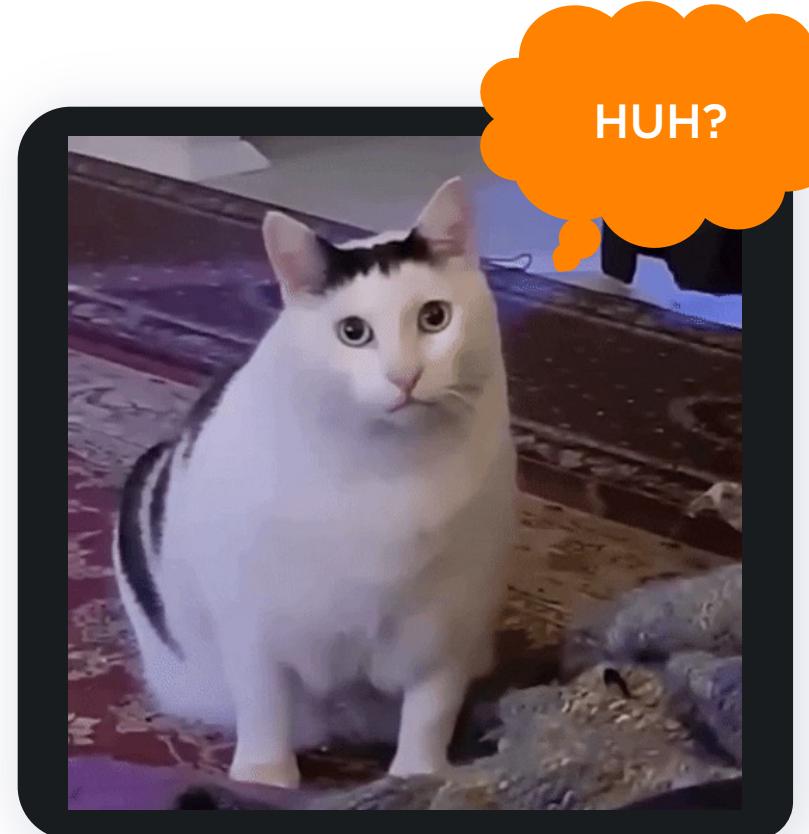
```
pd.options.mode.copy_on_write = True
```



YouTube

https://pandas.pydata.org/docs/dev/user_guide/copy_on_write.html#migrating-to-copy-on-write
<https://habr.com/ru/companies/wunderfund/articles/769176/>

КОПИИ и ПРЕДСТАВЛЕНИЯ POLARS



OUT-OF-PLACE ПОДХОД

Out-Of-Place подход:

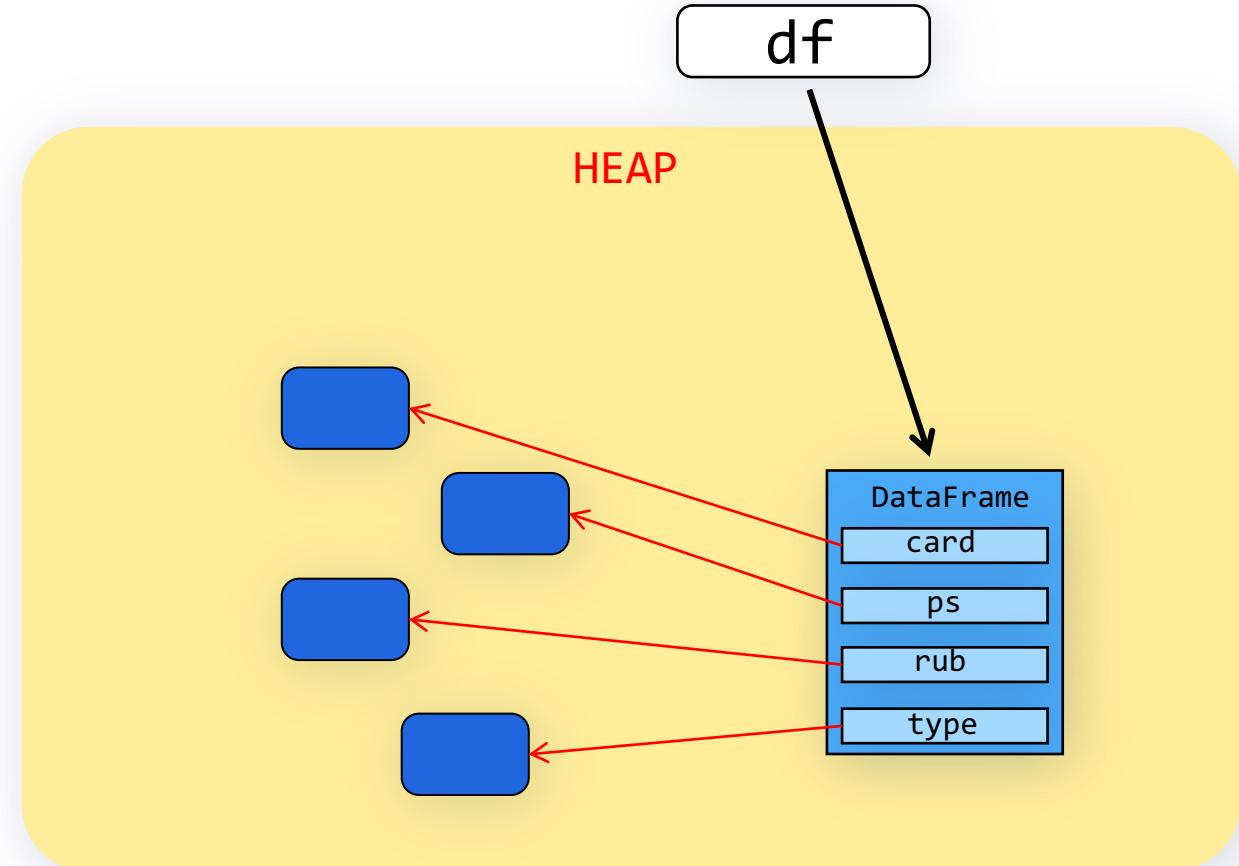
```
df = df.with_columns((pl.col("rub") > 25).alias("rub_>_25"))
```

In-Place подход:

```
df.drop_in_place("rub_>_25")
```

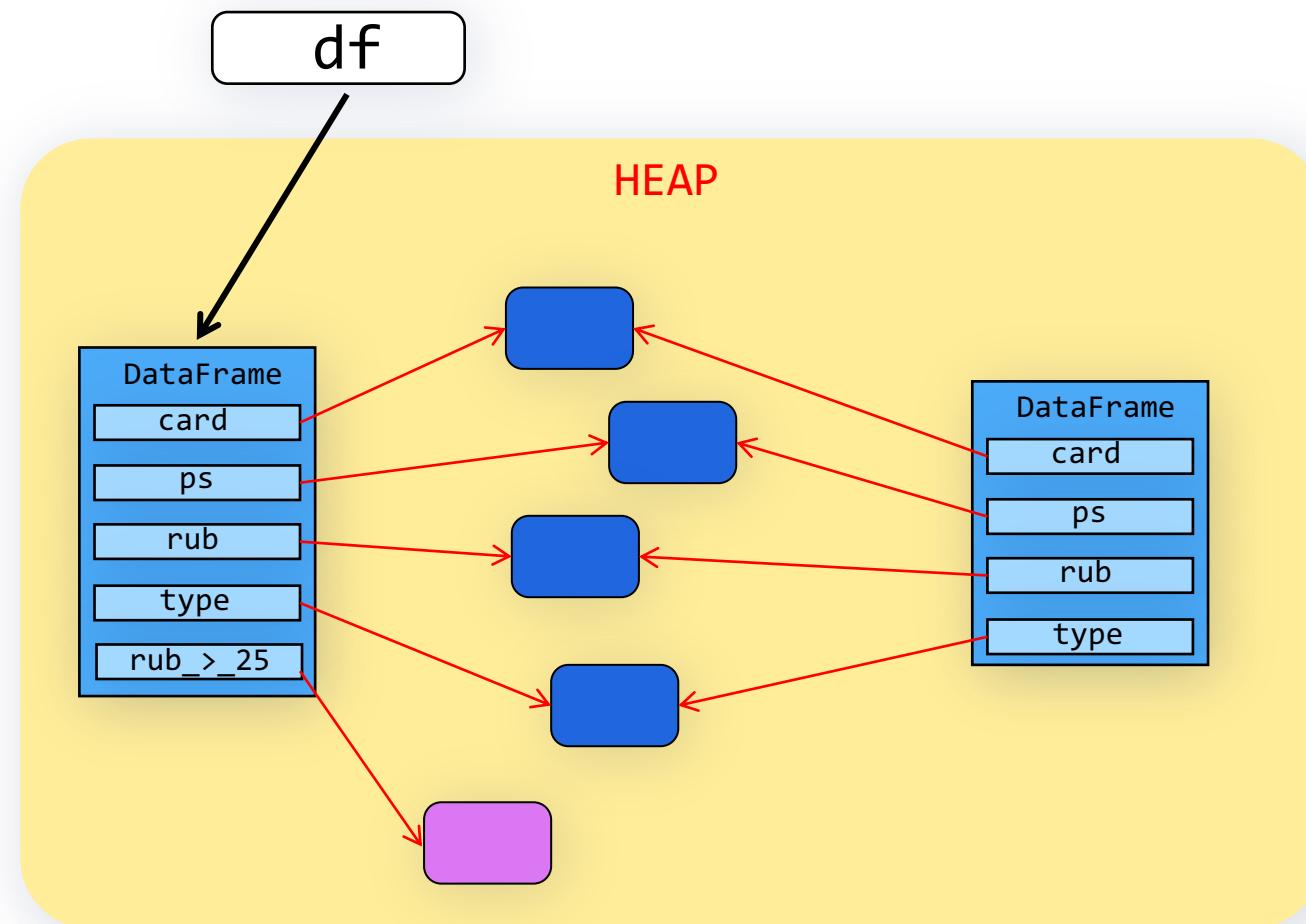
DATAFRAME в ПАМЯТИ

card	ps	rub	type
u64	str	u64	str
1	МИР	100	Покупка
1	МИР	25	Перевод
2	Не МИР	25	Покупка
2	Не МИР	10000	АТМ



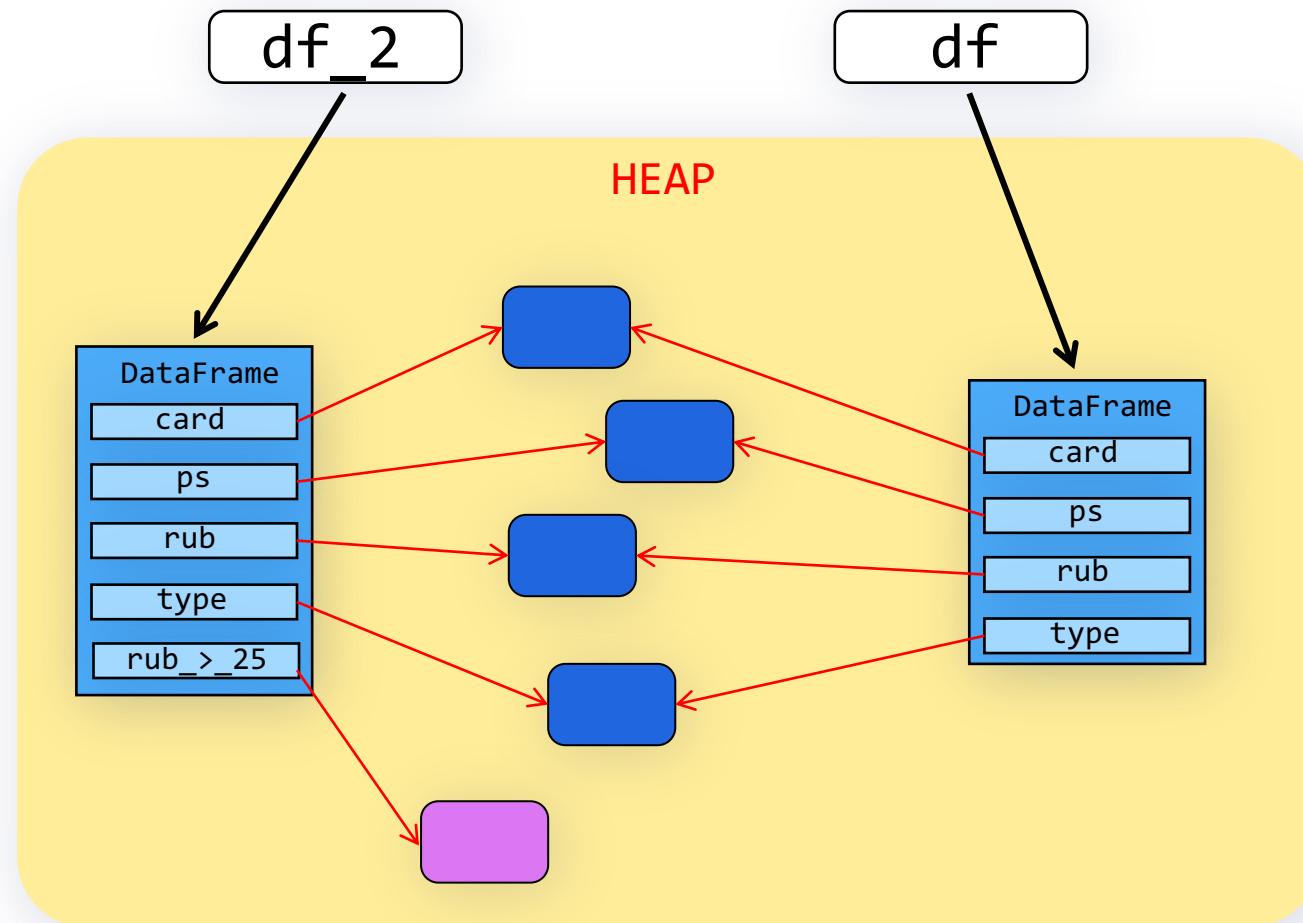
ДОБАВЛЕНИЕ КОЛОНКИ

```
df = df.with_columns((pl.col("rub") > 25).alias("rub_>_25"))
```



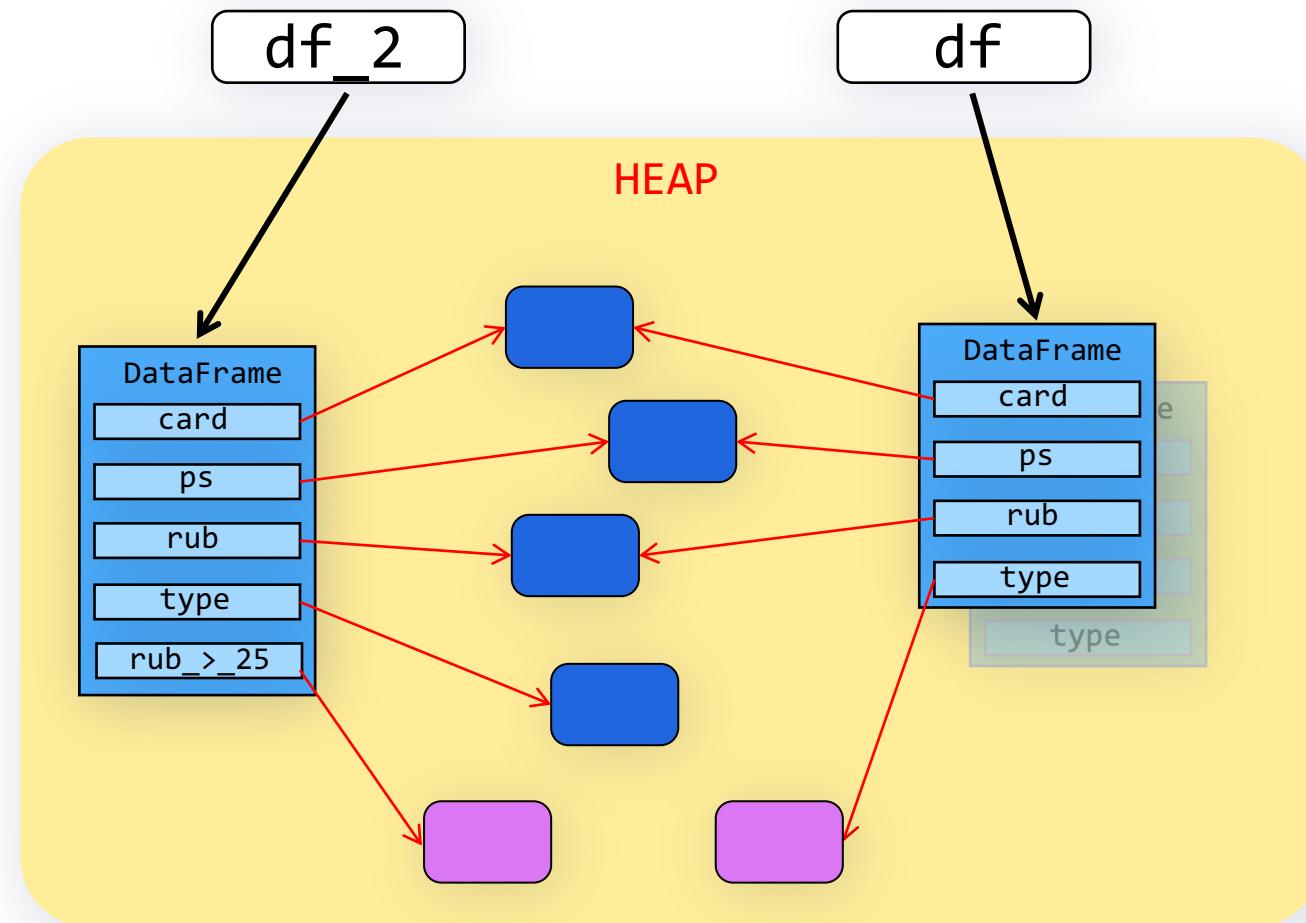
ДОБАВЛЕНИЕ КОЛОНКИ

```
df_2 = df.with_columns((pl.col("rub") > 25).alias("rub_>_25"))
```



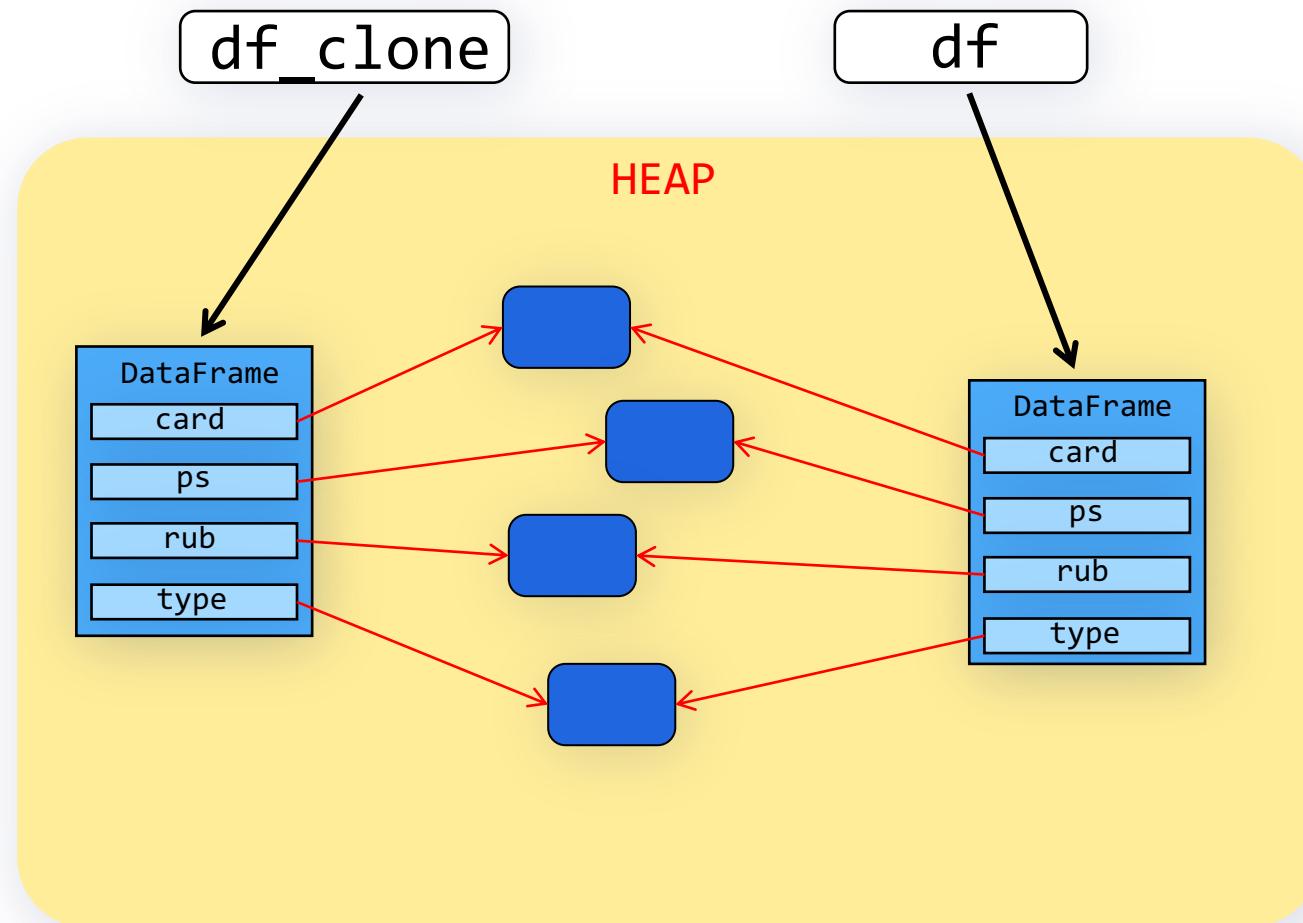
ДОБАВЛЕНИЕ КОЛОНКИ

```
df_2 = df.with_columns((pl.col("rub") > 25).alias("rub_>_25"))
df = df.with_columns(pl.col("type").cast(pl.Categorical))
```



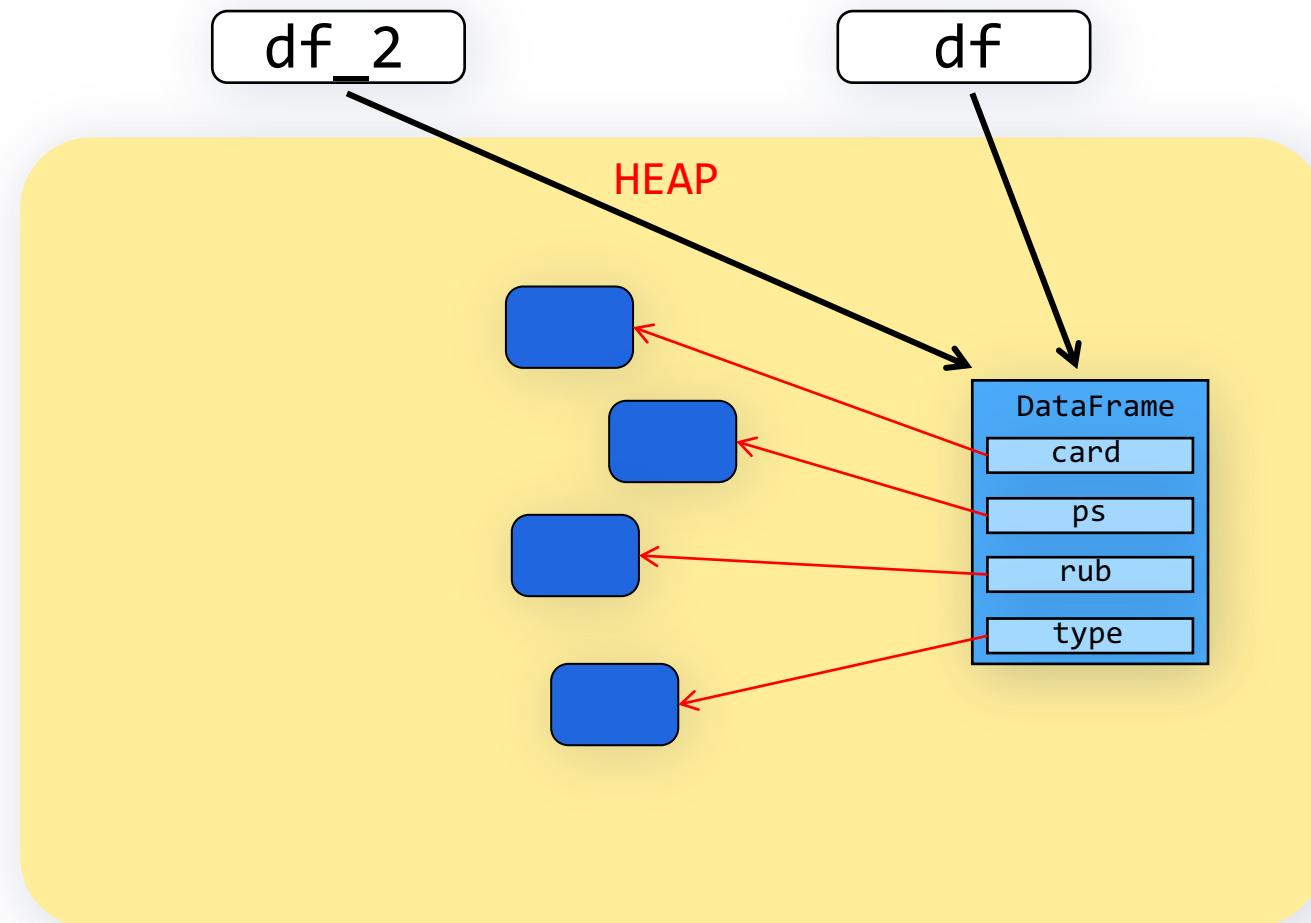
КЛОНИРОВАНИЕ DF

`df_clone = df.clone()`



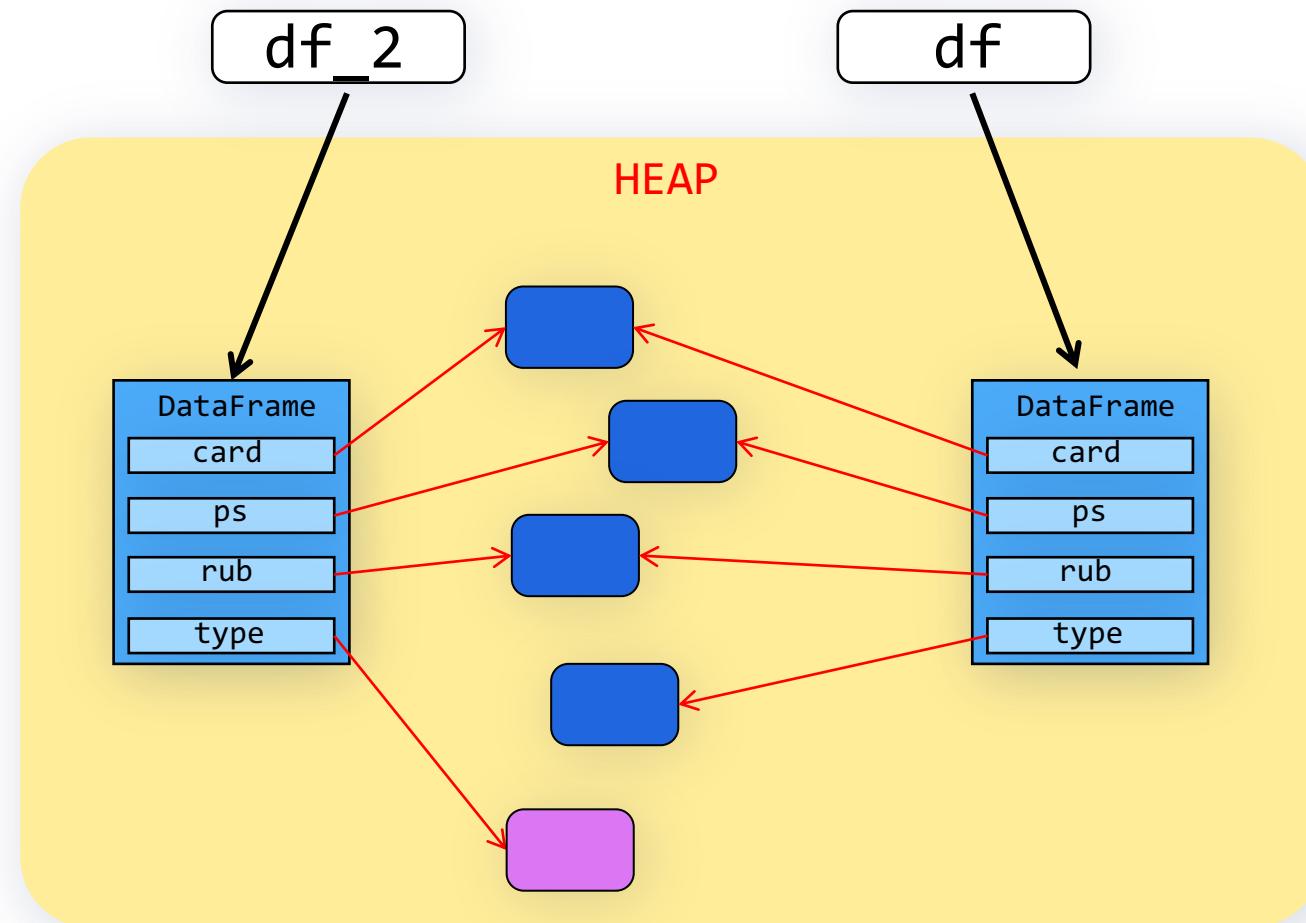
ДВЕ ПЕРЕМЕННЫЕ

`df_2 = df`



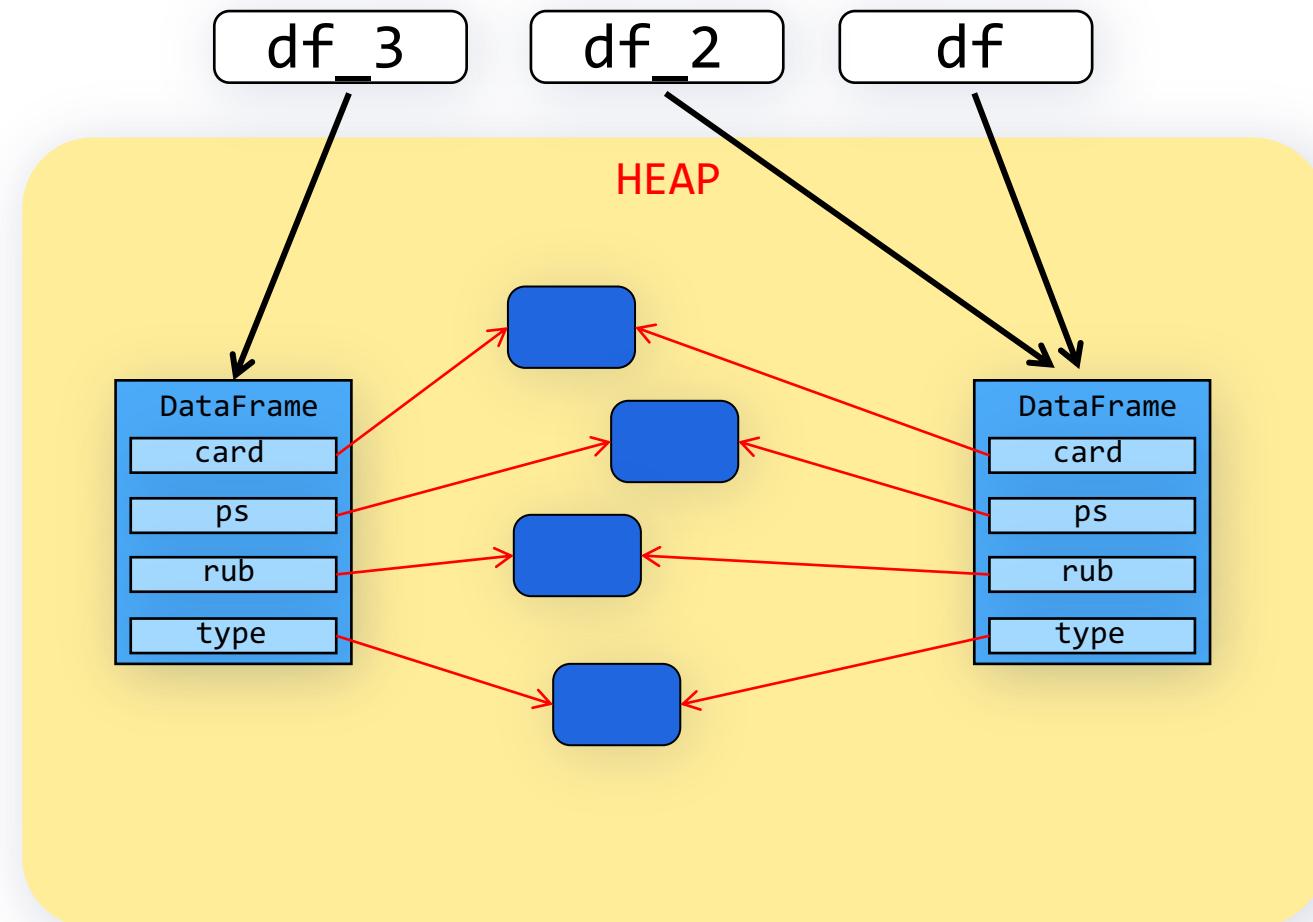
ДВЕ ПЕРЕМЕННЫЕ

```
df_2 = df  
df_2 = df_2.with_columns(pl.col("type").cast(pl.Categorical))
```



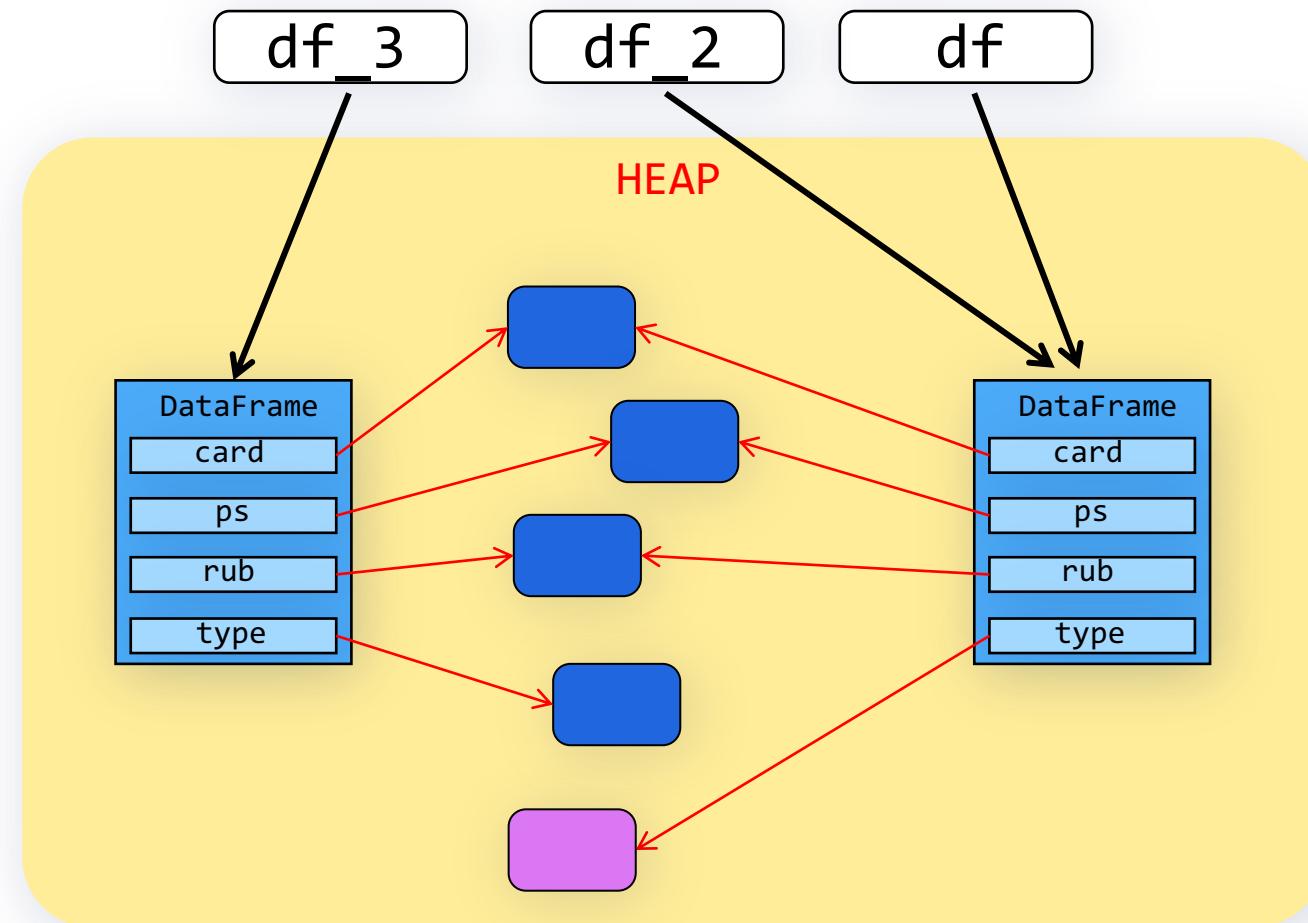
DF[...] = <VALUE>

```
df_2 = df  
df_3 = df.clone()
```



DF[...] = <VALUE>

df[0, "type"] = "ЕСОМ Покупка"



ДРУГИЕ IN-PLACE ОПЕРАЦИИ

In-place операции:

- `.drop_in_place(name: str) → Series`
- `.extend(other: DataFrame) → Self`
- `.replace_column(index: int, column: Series) → Self`

Операции с in-place параметром:

- `.hstack(columns: list[Series] | DataFrame, *, in_place: bool = False) → Self`
- `.vstack(other: DataFrame, *, in_place: bool = False) → Self`
- `.shrink_to_fit(*, in_place: bool = False) → Self`

ВЫВОДЫ ПО ПАМЯТИ

- Неизменяемые данные не копируются
- Изменение данных приводит к созданию новой колонки
- `df.clone()` – дешевая операция
- In-place операции не изменяют существующие колонки

Polars

Бенчмарки

Концепты в Polars

Синтаксис Polars

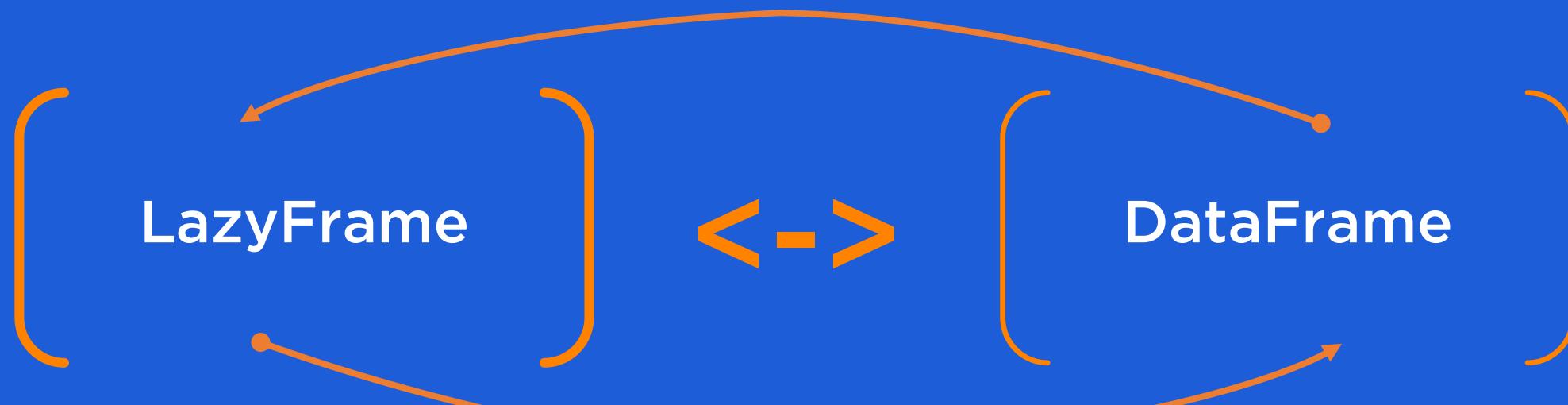
Использование памяти

› Lazy API

Прикладной Polars и ML

LAZYFRAME <-> DATAFRAME

`lf = df.lazy()`



`df = lf.collect()`

ОПТИМИЗАЦИЯ ЗАПРОСОВ



Если вы используете Lazy API, то Polars выполнит несколько оптимизаций вашего запроса.

Оптимизация	Описание
Predicate pushdown	Применяет фильтр как можно раньше, на уровне сканирования запроса
Projection pushdown	Выбирает только те столбцы, которые необходимы
Slice pushdown	Загружает только необходимый фрагмент на уровне сканирования
Common subplan elimination	Кэширует сканирования, которые используются несколькими поддеревьями в плане выполнения запроса
Simplify expressions	Различные оптимизации, такие как constant folding и замена дорогостоящих операций более быстрыми альтернативами
Join ordering	Оценивает ветки Join, которые должны быть выполнены в первую очередь, чтобы уменьшить нагрузку на память
Type coercion	Конвертирует типы, что позволяет операциям завершаться успешно и выполняться с минимально необходимой памятью
Cardinality estimation	Оценивает кардинальность, чтобы определить оптимальную стратегию

PREDICATE PUSHDOWN

predicate_pushdown=False

```
lf.with_columns(expr)\n    .filter(pl.col("ps") == "МИР")\n    .group_by("card").agg("rub_>_25")
```

AGGREGATE

[col("rub_>_25")]

BY

[col("card")]

4

FILTER [(col("ps")) == (String(МИР))]

3

WITH_COLUMNS:

[(col("rub")) > (25)].alias("rub_>_25")

]

2

```
DF ["card", "ps", "rub", "type"];\nPROJECT */4 COLUMNS;\nSELECTION: None
```

1

PREDICATE PUSHDOWN

`predicate_pushdown=True`

```
lf.with_columns(expr)\n    .filter(pl.col("ps") == "МИР")\n    .group_by("card").agg("rub_>_25")
```

AGGREGATE

```
[col("rub_>_25")]
BY
[col("card")]
```

4

WITH_COLUMNS: [
 [(col("rub")) > (25)].alias("rub_>_25")
]

2

```
DF ["card", "ps", "rub", "type"];
PROJECT */4 COLUMNS;
SELECTION: [(col("ps")) == (String(МИР))]
```

1

PROJECTION PUSHDOWN

projection_pushdown=False

```
lf.select("card", "ps", "rub")\
    .with_columns((pl.col("rub") > 25).alias("rub_>_25"))\
    .filter(pl.col("ps") == "МИР")\
    .group_by("card").agg("rub_>_25")
```

AGGREGATE
[col("rub_>_25")]
BY
[col("card")] 4

WITH_COLUMNS: [
 [(col("rub")) > (25)].alias("rub_>_25")
] 3

FAST_PROJECT: [card, ps, rub] 2

DF ["card", "ps", "rub", "type"];
PROJECT */4 COLUMNS;
SELECTION: [(col("ps")) == (String(МИР))]

1

PROJECTION PUSHDOWN

projection_pushdown=True

```
lf.select("card", "ps", "rub")\
    .with_columns((pl.col("rub") > 25).alias("rub_>_25"))\
    .filter(pl.col("ps") == "МИР")\
    .group_by("card").agg("rub_>_25")
```

AGGREGATE
[col("rub_>_25")]
BY
[col("card")] 4

WITH_COLUMNS: [
 [(col("rub")) > (25)].alias("rub_>_25")
] 3

FAST_PROJECT: [card, ps, rub] 2

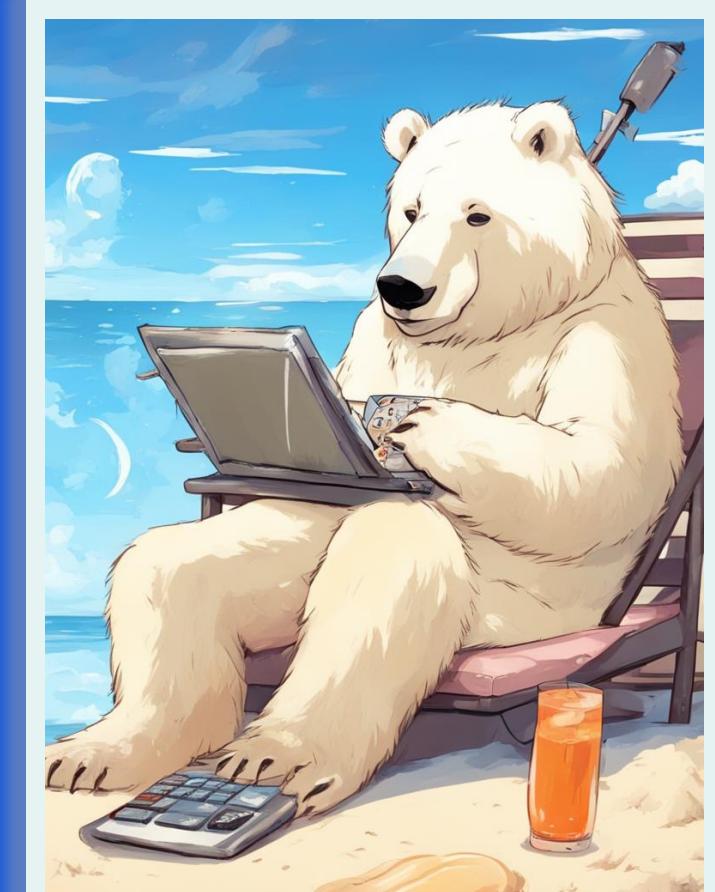
DF ["card", "ps", "rub", "type"];
PROJECT 3/4 COLUMNS;
SELECTION: [(col("ps")) == (String(МИР))]

1

STREAMING

`lf.collect(streaming=True)`

- `filter, slice, head, tail`
- `with_columns, select`
- `group_by`
- `join`
- `unique`
- `sort`
- `explode, melt`
- `scan_csv, scan_parquet, scan_ipc.`



ЛЕЖУ И ВЫЧИСЛЯЮ

STREAMING

streaming=True

```
lf.filter(pl.col("rub") > 25)
```

```
---- STREAMING
DF ["card", "ps", "rub", "type"];
PROJECT */4 COLUMNS;
SELECTION: [(col("rub")) > (25)]
---- END STREAMING
```

2

```
DF [];
PROJECT */0 COLUMNS;
SELECTION: "None"
```

1

STREAMING

streaming=True

```
lf.with_columns(pl.col("rub").sum().over("card").alias("rub_sum"))
```

```
WITH_COLUMNS: [
    col("rub").sum().over([col("card")]).alias("rub_sum")
]
```

3

```
---- STREAMING
DF ["card", "ps", "rub", "type"];
PROJECT */4 COLUMNS;
SELECTION: "None"
---- END STREAMING
```

2

```
DF [];
PROJECT */0 COLUMNS;
SELECTION: "None"
```

1

ДВА РЕЖИМА РАБОТЫ



Eager
API

VS



Lazy
API

- Немедленное исполнение кода
- Под капотом Lazy API (где возможно)

- Отложенное исполнение кода
- Оптимизация запросов
- Отсутствие промежуточных результатов
- Доступен стриминговый режим работы

Polars

Бенчмарки

Концепты в Polars

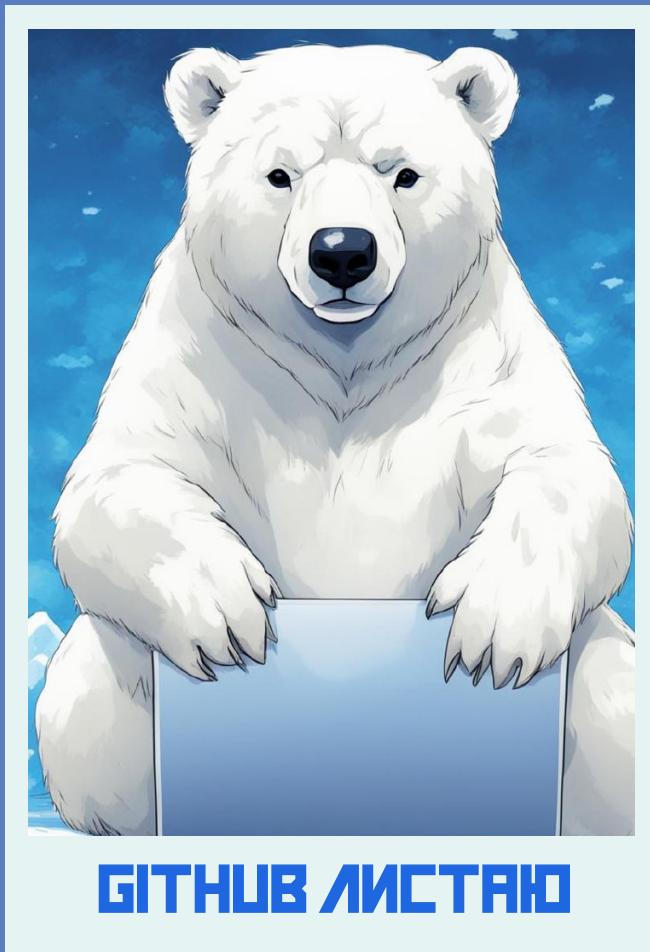
Синтаксис Polars

Использование памяти

Lazy API

› Polars и ML

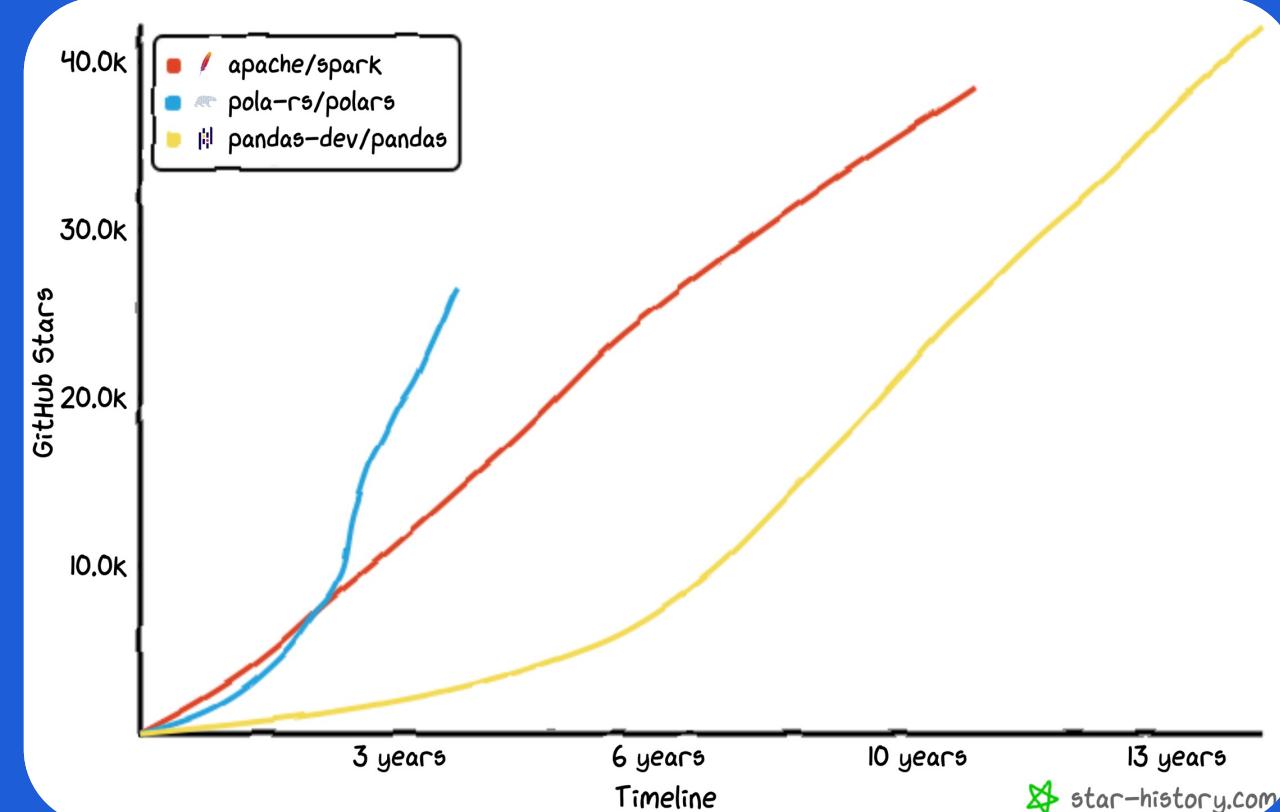
POLARS и GITHUB



Последняя версия: **0.20.26**

Разрабатывается с: **2020**

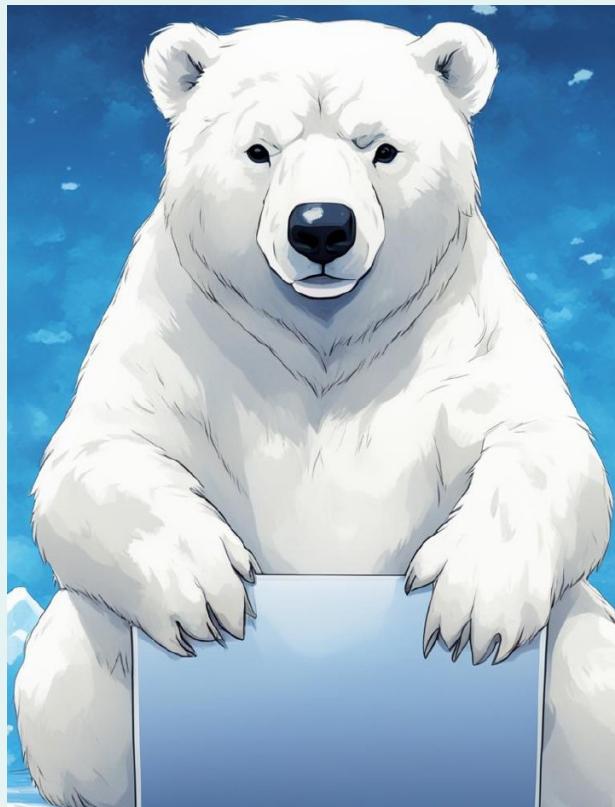
Распределение звезд среди популярных библиотек:



POLARS и GITHUB



Новый релиз – почти каждую неделю



GITHUB АИСТАЮ

[polars](#) Public

Dataframes powered by a multithreaded, vectorized query engine, written in Rust

Rust ⭐ 26,509 ⚡ 1,618 ⚡ 1,546 (5 issues need help) ⚡ 87 Updated 40 minutes ago



Поиск: [polars](#)
language: Jupyter

 Code

4.5k

 Repositories

531

 Issues

265

 Pull requests

588

 Discussions

389



Поиск: [polars](#)
language: Python

 Code

33.3k

 Repositories

936

 Issues

2k

 Pull requests

3k

 Discussions

389

POLARS и KAGGLE



Популярность в Соревновательном Анализе Данных

Поиск **polars**:

Last 90 days: **797**

Notebooks: **1,787**

Comments: **937**

This week: **46**

Topics: **148**

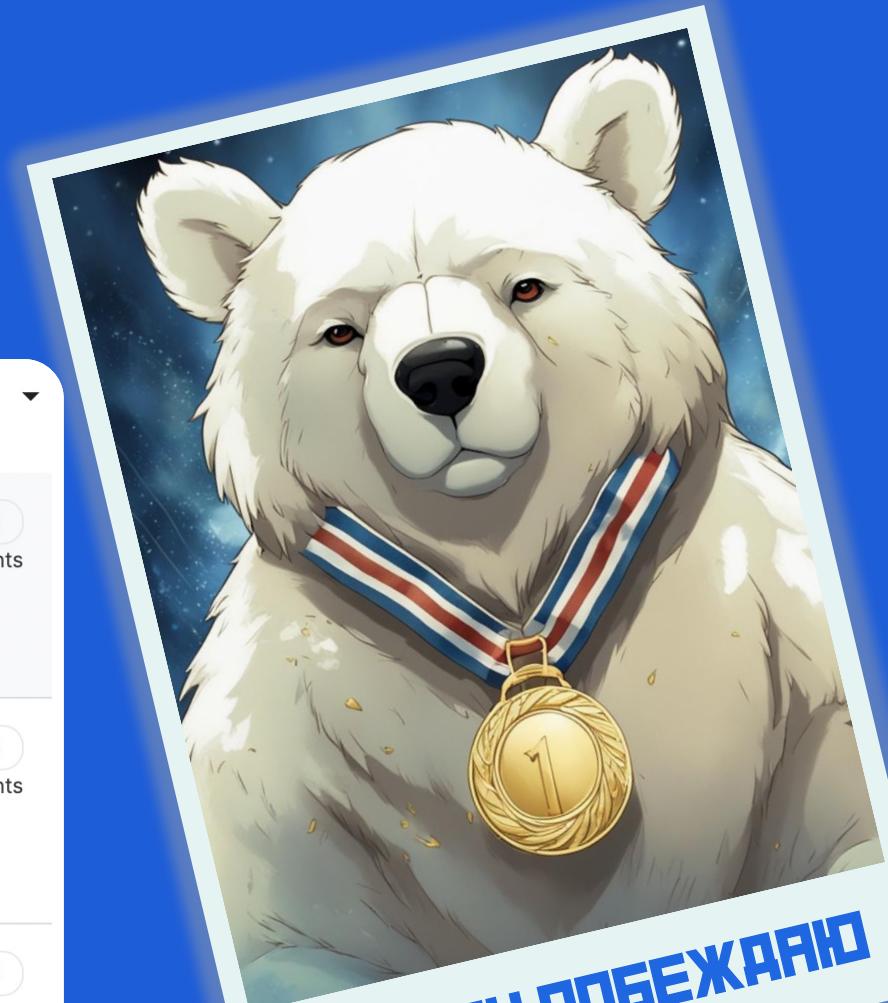
1,775 Results

Relevance ▾

 **Enefit|Polars-preprocessing**
Notebook · 6mo ago · by [MikeOMa](#)
Args: revealed_target: **polars** DF (not lazy) with all of the data.

 **[polars] Proof of concept: LGBM Ranker** 
Notebook · 1y ago · by [Radek Osmulski](#)
pip install **polars** import **polars** as pl train = pl.read_parquet(..

 **Carno** **CPU CatBoost Baseline (Using Polars) - Inference**
Notebook · 1y ago · by [Carno Zhao](#)
Using **Polars** I learnt '**polars**' from last OTTO competition, which helped our team to do fast feature



СОРЕВУ ПОБЕЖДАЮ

POLARS и SQL

```
query = "select * from df where rub > 25"  
sql_context = pl.SQLContext(df=df, eager_execution=True)  
df_sql = sql_context.execute(query)
```

card	ps	rub	type
u64	str	u64	str
1	МИР	100	Покупка
1	МИР	25	Перевод
2	Не МИР	25	Покупка
2	Не МИР	10000	ATM

execute

card	ps	rub	type
u64	str	u64	str
1	МИР	100	Покупка
2	Не МИР	10000	ATM

МЫ ЗНАЕМ, КАК ВСЕ ЛЮБЯТ SQL

POLARS и БД

```
uri = "postgresql://username:password@server:port/database"  
query = "SELECT * FROM table"  
pl.read_database_uri(query=query, uri=uri)
```



- Postgres
- MySQL
- MariaDB
- SQLite
- Redshift
- Clickhouse
- SQL Server
- Azure SQL Database
- Oracle
- Big Query

POLARS и SKLEARN

Использование в Пайплайнах:

UPD: Sklearn 1.4 – Январь 2024

Major Feature

Transformers now support polars output with
`.set_output(transform="polars")`

Использование в ML Алгоритмах:

Polars под капотом преобразуется
в Numpy Array

NdArray $2.81 \text{ ms} \pm 945 \mu\text{s}$

Polars DF $5.02 \text{ ms} \pm 854 \mu\text{s}$



POLARS и TORCH

UPD: Polars 0.20.24 – Апрель 2024

.to_torch()



Пример:

[1]

```
df = pl.DataFrame({
    "target": [0, 1, 1, 0],
    "rub": [100, 99, 30, 1000],
    "ps_id": [1, 2, 5, 1]
})
```

[2]

```
ds: torch.Tensor = df.to_torch(
    return_type="dataset",
    label="target"
)
dl = DataLoader(ds, batch_size=2)
batches = list(dl)
batches[0]
```

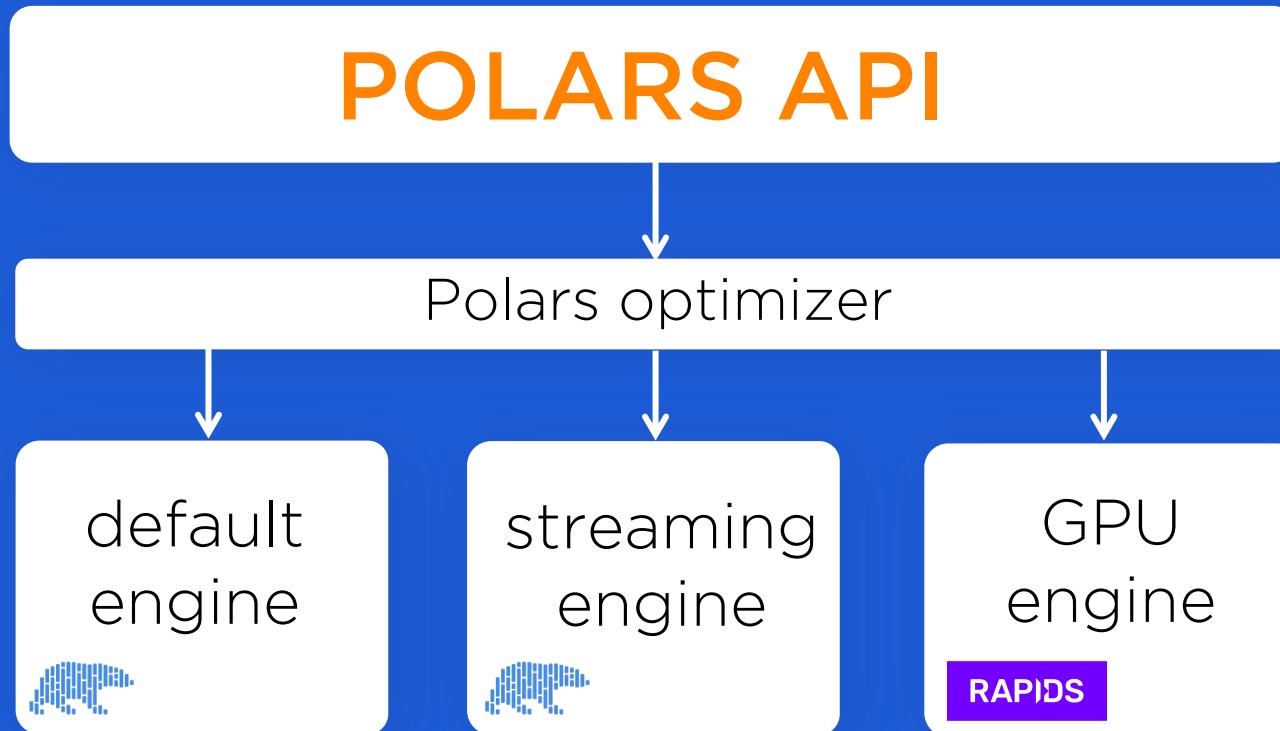
[Out]

```
[tensor([[100.0000, 1.0000],
[99.0000, 2.0000]], dtype=torch.float64),
tensor([0, 1])]
```

POLARS и RAPIDS

.collect(gpu=True)

TBD Polars and NVIDIA engineers are collaborating
to bring GPU acceleration to Polars



УСПЕХИ КОМАНДЫ с



Получили ускорение в рутинных задачах



Перенесли часть пайплайнов с Pandas на Polars



Получили ускорение на этапах

- препроцессинга для ML
- генерации признаков для ML
 - Агрегированные признаки
 - Оконные признаки



Выступили с докладом на внутреннем митапе

УСКОРЕНИЕ ПАРСИНГА ПРИМЕР PSYCHOHOUSE



PsycoHouse – для парсинга с Postgres

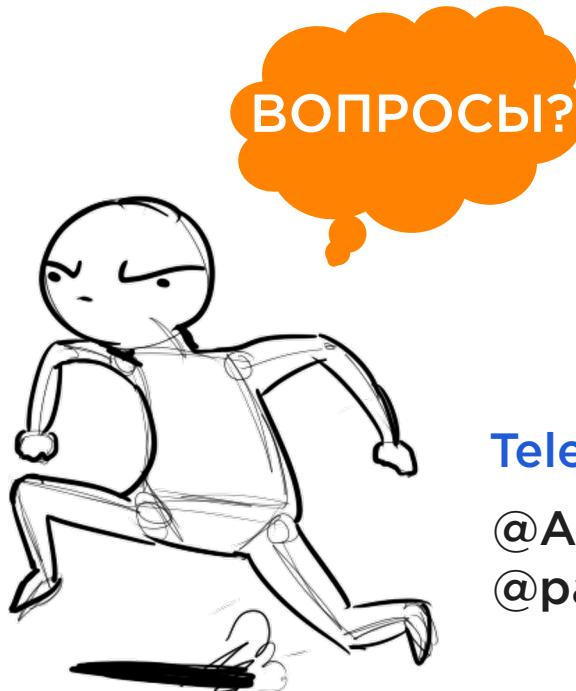
Обновили внутренние библиотеки для работы с БД в Python:

Получилось ускорить обработку запросов за счет парсера на Polars, использования Arrow

СПАСИБО ЗА ВНИМАНИЕ!

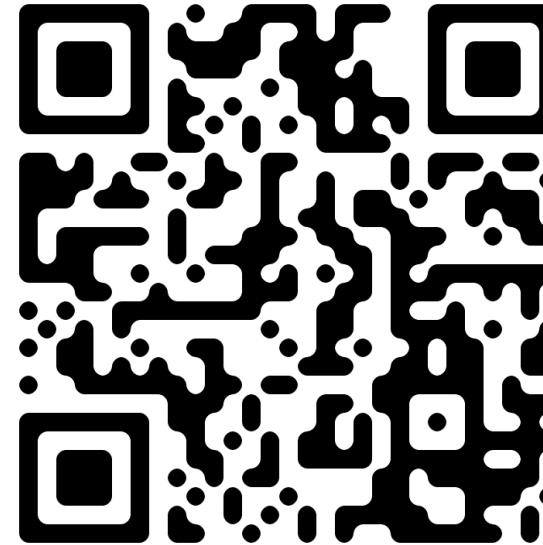
А ВЫ

ХОТИТЕ ПОПРОБОВАТЬ
POLARS?



Telegram:

@Arhimisha
@patsvetov



Репозиторий*
Github



Блог на Habr
Мир Plat.Form

*материалы к докладу