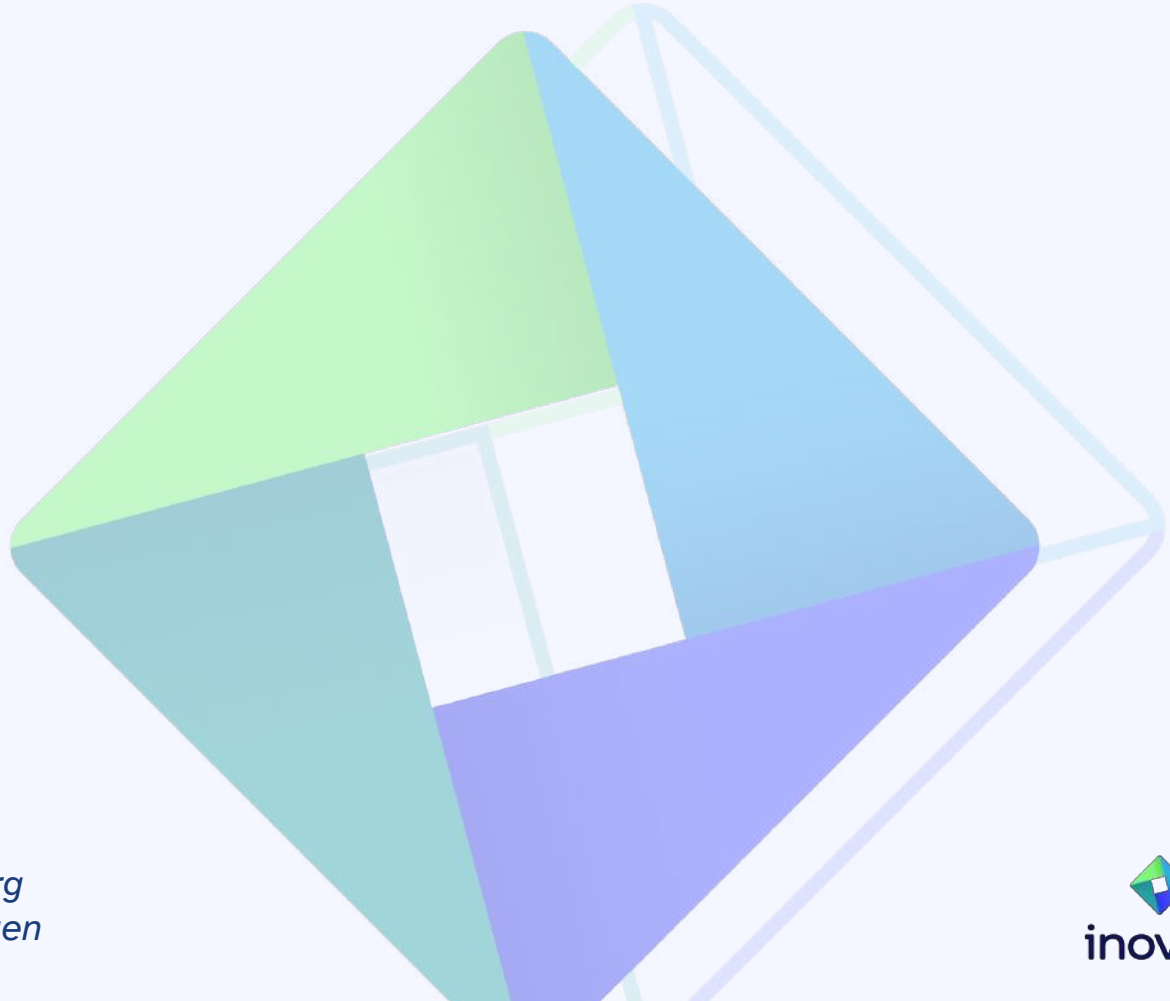
Polars

DataFrames for the new era



Team inovex
Karlsruhe · Köln · München · Hamburg
Berlin · Stuttgart · Pforzheim · Erlangen

inovex

Bernd Kaiser

Developer at inovex Erlangen

🎸🎢🏥🧾

🏦🃏📹🥁

Bernd Kaiser

@meldron

kaiser.land

JSCraftCamp.org Organizer

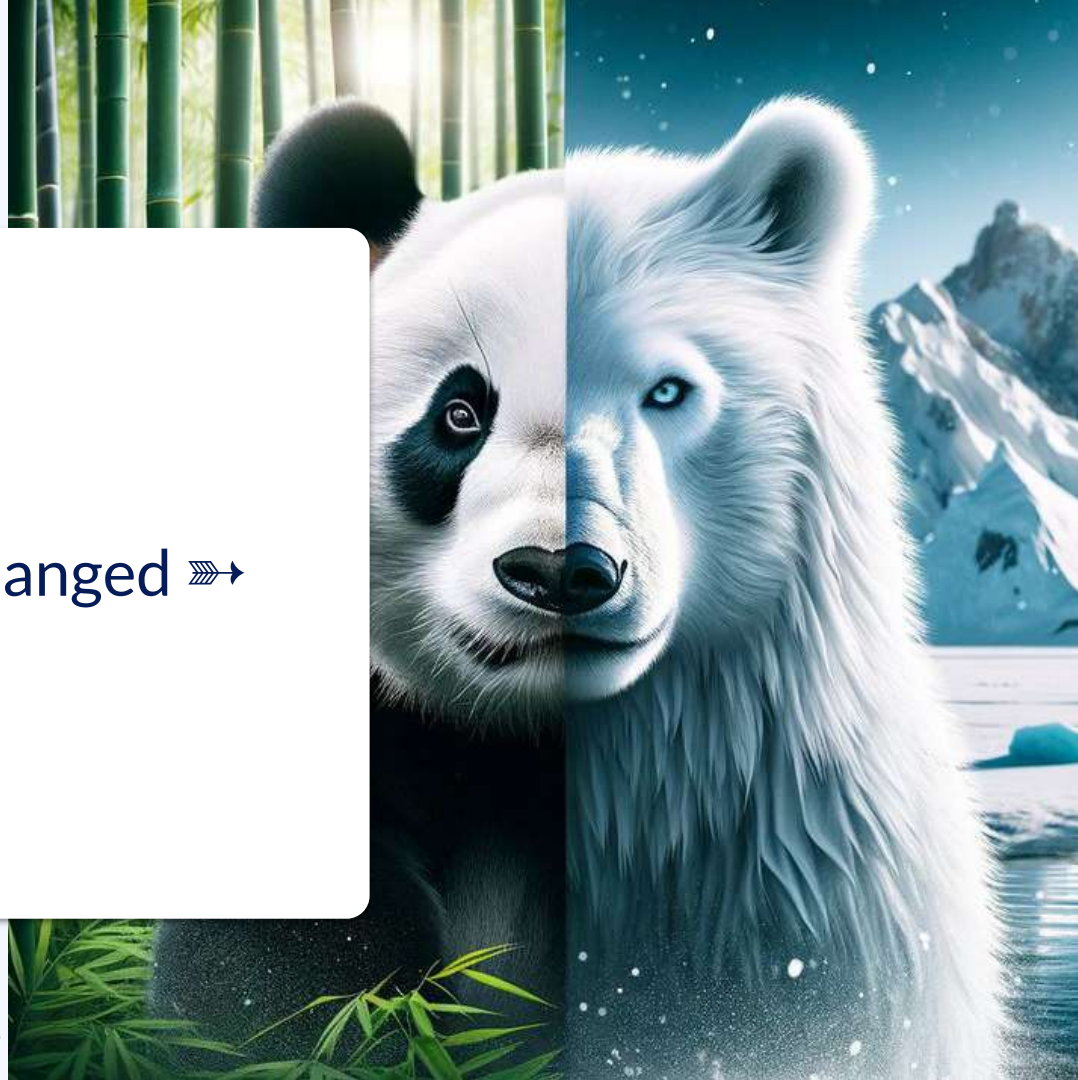
7. & 8. June 2024 in Munich

https://app.inovex.ai

inovex

Agenda

- How it has been 🐼
- How the landscape changed ➳
- Polars 🐻‍❄️



inovex

Pandas 🐼

Everyone’s favorite DataFrame Library

inovex

Pandas

- Python DataFrame for data manipulation and analysis
- Built upon NumPy
- Open Source (BSD License)
- Created in 2008
- Downloaded 191_780_336 last month[1]

1: pypistats

inovex

Pandas

Pros

- 🐼 Everyone knows it
- 🐼 Integrated into everything (Plotting, DBs, ...)
- 🐼 Large and active community support (docs, tutorials, ...)

Cons

- 😓 Inconsistent API
- 😓 Performance Issues
- 😓 No multi-core processing
- 😓 No zero-copy sharing

inovex

How the landscape changed

A new era?

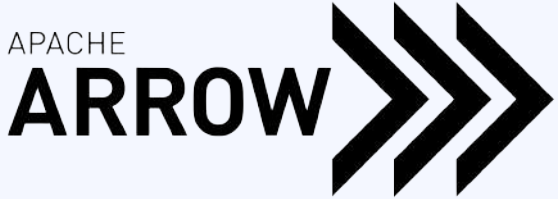
inovex

How the landscape changed

- Explosion in data volume and complexity
- Increased diversity of data processing frameworks
- Emphasis on leveraging multi-core CPU architectures
- Transition to in-memory computing for real-time insights
- Growth of machine learning and AI applications

inovex

Apache Arrow

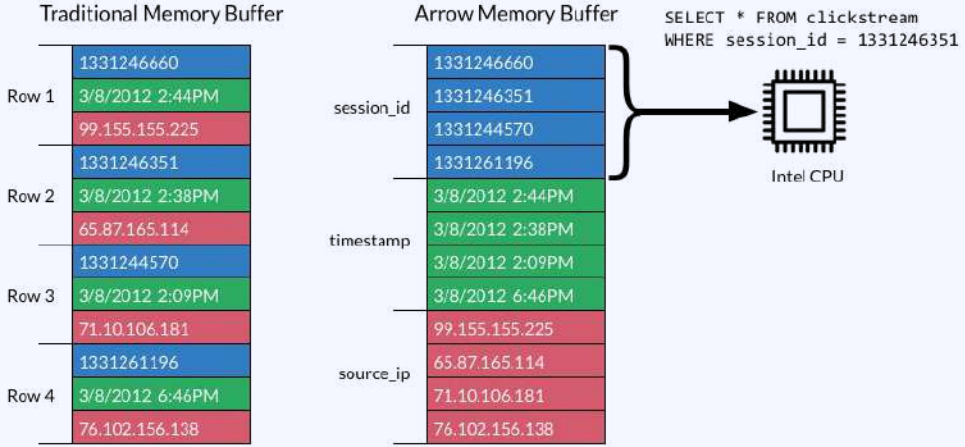


1. Language-independent columnar memory format for data
2. Designed for efficient analytics on CPUs/GPUs
3. Supports zero-copy reads, eliminating serialization overhead
4. Wide library support across multiple programming languages
5. Community-driven development with open consensus decision-making

inovex

Apache Arrow Format

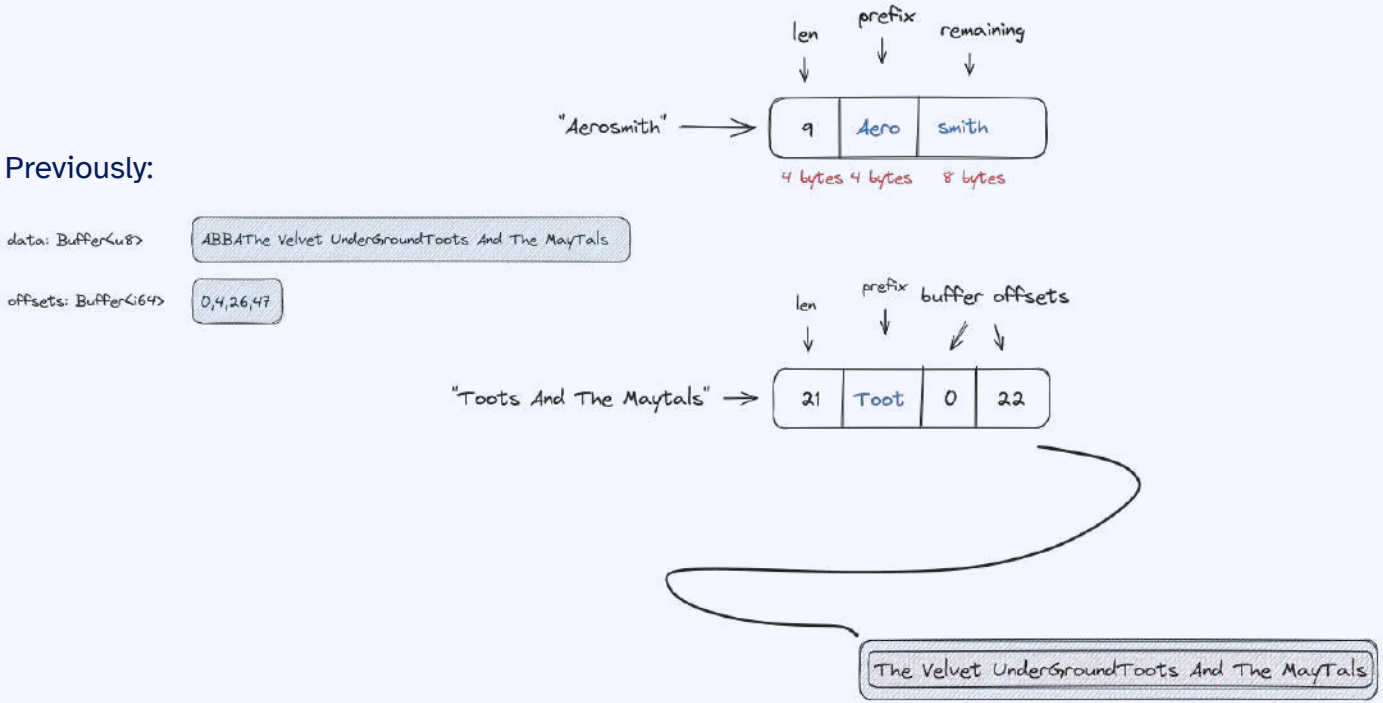
	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138



Source: https://arrow.apache.org/overview/

German Style Strings (Hyper/Umbra)

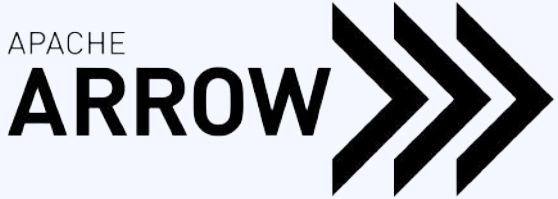
Previously:



Source: [pola.rs - Why we have rewritten the string data type]

inovex

Projects Powered by Apache Arrow



RAY

influxdb

https://arrow.apache.org/powered_by/

inovex

Polars 🐻‍❄️

DataFrames for the new era

inovex

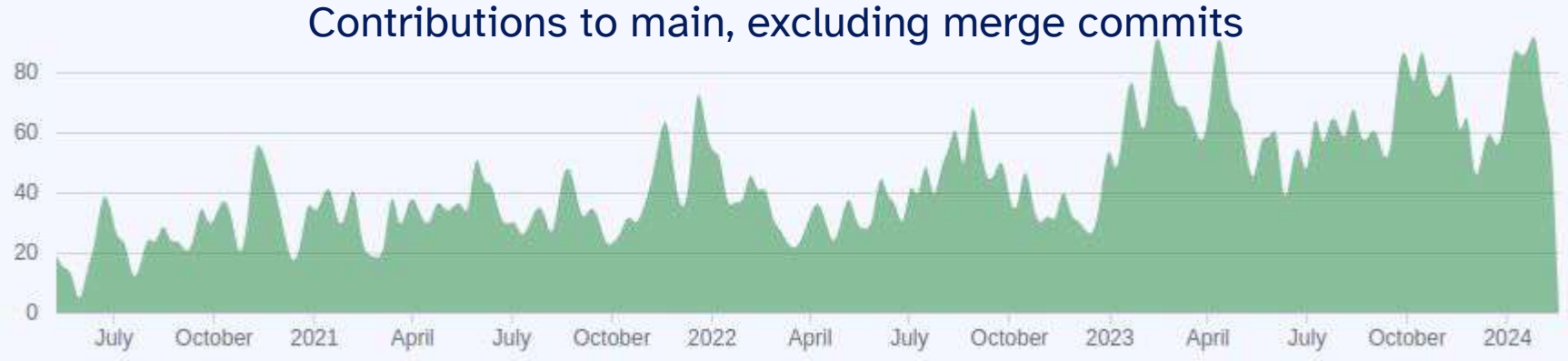
Polars - A Modern DataFrame Library

- 🦀 **Fast, Multi-threaded Query Engine**: Written in Rust for effective parallelism
- 📑 **Vectorized, Columnar Processing**: Enables high performance on modern processors
- 📈 **Intuitive Data Wrangling**: Expressions familiar to data scientists and analysts
- ⚡ **Readable and Performant Code**: Balance between clarity and efficiency
- 📖 **Committed to Open Source**: Active community contributions under MIT license

inovex

Project Overview

- Created by Ritchie Vink
- License: MIT
- Github: 25k ⭐
- Bindings (mthly DLs): Rust (75k), Python (3.5M), JavaScript (20k), R



Source: https://github.com/pola-rs/polars/graphs/contributors

inovex

Supports all common data formats

- **Text**: CSV & JSON
- **Binary**: Parquet, Delta Lake, AVRO & Excel
- **IPC**: Feather, Arrow
- **Databases**:
  - **read**: SQLAlchemy, connector-x & ADBC
  - **write**: SQLAlchemy, Arrow Database Connectivity (ADBC)
- **S3 Cloud storage**: AWS, Azure, GCP
- **Google Big Query**
- **Apache Iceberg tables**

inovex

Dataset: eCommerce behavior data from multi category store

- Kaggle Dataset, data collected by Open CDP project
- **Nov 2019**: 67,501,979 rows (CSV 8.4GB , Parquet: 2GB)
- Each row represents an event
- All events are related to products and users

File: /home/bernd/Downloads/ecommerce-archive/2019-Nov.csv

```
event_time,event_type,product_id,category_id,category_code,brand,price,user_id,user_session
2019-11-01 00:00:00 UTC,view,1003461,2053013555631882655,electronics.smartphone,xiaomi,489.07,520088904,4d3b30da-a5e4-49df-b1a8-ba5943f1dd33
2019-11-01 00:00:00 UTC,view,5000088,2053013566100866035,appliances.sewing_machine,janome,293.65,530496790,8e5f4f83-366c-4f70-860e-ca7417414283
2019-11-01 00:00:01 UTC,view,17302664,2053013553853497655,,creed,28.31,561587266,755422e7-9040-477b-9bd2-6a6e8fd97387
2019-11-01 00:00:01 UTC,view,3601530,2053013563810775923,appliances.kitchen.washer,lg,712.87,518085591,3bfb58cd-7892-48cc-8020-2f17e6de6e7f
2019-11-01 00:00:01 UTC,view,1004775,2053013555631882655,electronics.smartphone,xiaomi,183.27,558856683,313628f1-68b8-460d-84f6-cec7a8796ef2
2019-11-01 00:00:01 UTC,view,1306894,2053013558920217191,computers.notebook,hp,360.09,520772685,816a59f3-f5ae-4ccd-9b23-82aa8c23d33c
2019-11-01 00:00:01 UTC,view,1306421,2053013558920217191,computers.notebook,hp,514.56,514028527,df8184cc-3694-4549-8c8c-6b5171877376
2019-11-01 00:00:02 UTC,view,15900065,2053013558190408249,,rondell,30.86,518574284,5e6ef132-4d7c-4730-8c7f-85aa4082588f
2019-11-01 00:00:02 UTC,view,12708937,2053013553559896355,,michelin,72.72,532364121,0a899268-31eb-46de-898d-09b2da950b24
2019-11-01 00:00:02 UTC,view,1004258,2053013555631882655,electronics.smartphone,apple,732.07,532647354,d2d3d2c6-631d-489e-9fb5-06f340b85be0
2019-11-01 00:00:03 UTC,view,17200570,2053013559792632471,furniture.living_room.sofa,,437.33,518780843,aa806835-b14c-45af-9538-cd4d1849be3e
2019-11-01 00:00:03 UTC,view,2701517,2053013563911439225,appliances.kitchen.refrigerators,,155.11,518427361,c89b0d96-247f-4044-9c91-bb5f38c6af9b
2019-11-01 00:00:03 UTC,view,16700260,2053013559901684381,furniture.kitchen.chair,,31.64,566255262,173d7b72-1db7-4638-8207-be8148bf3c9f
2019-11-01 00:00:04 UTC,view,34600011,2060981320581986480,,,20.54,512416379,4dfe2c67-e537-4dc2-ae69-0b65263db091
2019-11-01 00:00:05 UTC,view,4600658,2053013563944993659,appliances.kitchen.dishwasher,samsung,411.83,526595547,aab33a9a-29c3-4d50-84c1-8a2bc9256104
2019-11-01 00:00:06 UTC,view,24900193,2053013562183385881,,,1.09,512651494,f603c815-f51a-46fe-9404-cb586e35edaf
2019-11-01 00:00:07 UTC,view,27400066,2053013563391345499,,,8.55,551061950,3f6112f1-5695-4e88-bb0a-49f9e36658ff
2019-11-01 00:00:07 UTC,view,5100503,2053013553375346967,,xiaomi,22.68,520037415,f54fa96a-f3f2-43ac-99a4-fcb2a449dd36
2019-11-01 00:00:07 UTC,view,1004566,2053013555631882655,electronics.smartphone,huawei,164.84,566265908,52c2c76c-b79e-4794-86ff-badc76d35f5a
```


inovex

Pandas vs Polars

Pandas DataFrame:

```python
import pandas as pd

df = pd.read_csv("2019-Nov.csv")

df_filtered = df[df["event_type"] == "purchase"]

df_grouped = df_filtered
  .groupby("brand")["price"]
  .mean()
  .reset_index()

df_sorted = df_grouped
              .sort_values("price", ascending=False)

print(df_sorted)
```

```
              brand        price
647         dynacord  1919.480000
1823           rado  1899.660000
1641       omabelle  1852.267248
33       aerosystem  1801.820000
1714           peda  1724.247857
...             ...          ...
969           heinz     1.180000
239           banbao     1.180000
703   enlightenbrick     0.970000
863           gerber     0.960385
585              dfz     0.900000

[2486 rows x 2 columns]
```

Elapsed time: 1m 56s
Percent CPU: 100%
Maximum resident set size: 18.5 GB

inovex

Pandas vs Polars

Polars DataFrame:

```python
import polars as pl

df = (
    pl.read_csv("2019-Nov.csv")
    .filter(pl.col("event_type") == "purchase")
    .group_by("brand")
    .agg(pl.col("price").mean())
    .sort("price", descending=True)
)

print(df)
```

```
shape: (2_488, 2)
┌─────────────────┬────────────┐
│ brand           ┆ price      │
│ ---             ┆ ---        │
│ str             ┆ f64        │
╞═════════════════╪════════════╡
│ dynacord        ┆ 1919.48    │
│ rado            ┆ 1899.66    │
│ omabelle        ┆ 1852.267248 │
│ aerosystem      ┆ 1801.82    │
│ peda            ┆ 1724.247857 │
│ …               ┆ …          │
│ banbao          ┆ 1.18       │
│ heinz           ┆ 1.18       │
│ enlightenbrick  ┆ 0.97       │
│ gerber          ┆ 0.960385   │
│ dfz             ┆ 0.9        │
└─────────────────┴────────────┘
```

Elapsed time: 13s
Percent CPU: 900%
Maximum resident set size: 21 GB

`POLARS_MAX_THREADS` env var can control CPU usage

Polars LazyFrame

```python
import polars as pl

lf = (
    pl.scan_csv("2019-Nov.csv")
    .filter(pl.col("event_type") == "purchase")
    .group_by("brand")
    .agg(pl.col("price").mean())
    .sort("price", descending=True)
)
print(lf.explain())
df = lf.collect()

print(df)
```

```
SORT BY [col("price")]
  AGGREGATE
        [col("price").mean()] BY [col("brand")]
        FROM Csv SCAN 2019-Nov.csv
      PROJECT 3/9 COLUMNS
      SELECTION: [(col("event_type")) == (String(purchase))]
```

```
shape: (2_488, 2)
┌──────────────┬──────────┐
│ brand        ┆ price    │
│ ---          ┆ ---      │
│ str          ┆ f64      │
╞══════════════╪══════════╡
│ dynacord     ┆ 1919.48  │
│ rado         ┆ 1899.66  │
│ omabelle     ┆ 1852.267248 │
│ aerosystem   ┆ 1801.82  │
│ peda         ┆ 1724.247857 │
│ …            ┆ …        │
│ banbao       ┆ 1.18     │
│ heinz        ┆ 1.18     │
│ enlightenbrick ┆ 0.97   │
│ gerber       ┆ 0.960385 │
│ dfz          ┆ 0.9      │
└──────────────┴──────────┘
```

Elapsed time: 5s
Percent CPU: 1033%
Maximum resident set size: 9 GB

inovex

Polars `LazyFrame` Info

`LazyFrame` doesn't run each query line-by-line,
but instead processes the full query end-to-end

- Apply automatic **query optimization**
- Catch schema errors **before** processing the data
- Work with larger than memory datasets using **streaming**

```python
import polars as pl
lf = (
      …
)
print(lf.explain(streaming=True))
df = lf.collect(streaming=True)
```

```
--- STREAMING
SORT BY [col("price")]
  AGGREGATE
        [col("price").mean()] BY [col("brand")]
        FROM Csv SCAN 2019-Nov.csv
      PROJECT 3/9 COLUMNS
      SELECTION: [(col("event_type")) == (String(purchase))]
--- END STREAMING

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


inovex

Polars Expressions & Transformations

- Expressions
  - Cache-Optimal String Functions
  - Folds
  - Window Functions
  - First-class support for Lists & Arrays
  - Structs to work with multiple columns
- Transformations
  - Joins
  - Pivots
  - Melts
  - Time series
  - Concatenation

inovex

Visualization

- hvPlot (used by built-in `plot`)
- Matplotlib (converts Series to numpy arrays)
- Seaborn (Python dataframe API standard)
- Plotly
- Altair

inovex

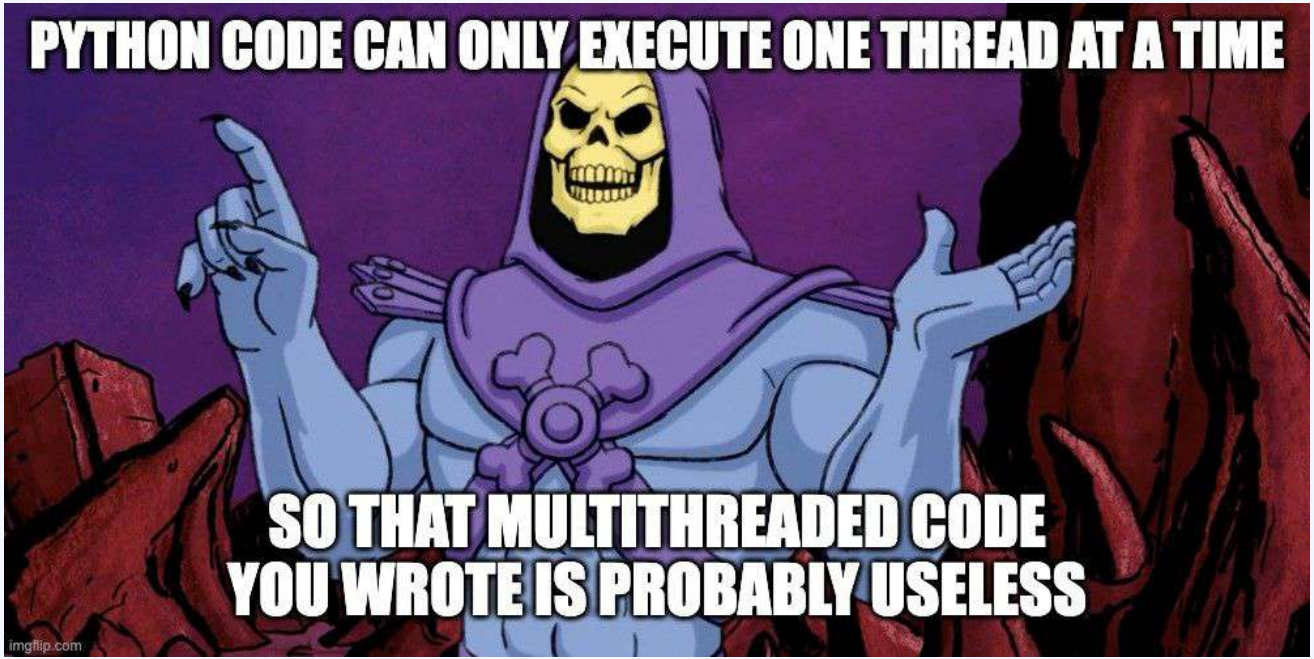
Python User Defined Functions

- `map_batches`: passing the Series in an expression to a third party library (do not use with `group_by`)
- `map_elements` (f.k.a. `apply`): passes elements of the column to the Python function

```
df.select(
    pl.col("tracking_data")
      .map_batches(lambda s: TrackingNeuralNetwork.predict(s.to_numpy()))
      .alias("wishes_and_desires"),
    pl.col("credit_card_encrypted")
      .map_elements(lambda v: decrypt(key, v))
      .alias("credit_card")
)
```

inovex

GIL explained:



inovex

https://github.com/meldron/polars-expedition

inovex

Rust Extensions with `pyo3-polars`

- Create Rust extensions to circumvent the GIL (😈)
- Uses PyO3 (11k⭐) and maturin (3k⭐)
- Outputs wheel files for specific architectures / targets

```toml
[package]
name = "polars_encrypt"
version = "0.1.0"
edition = "2021"
[lib]
name = "polars_encrypt"
crate-type = ["cdylib"]
[dependencies]
polars = { version = "0.38", features = [...] }
polars-core = { version = "0.38" }
polars-lazy = { version = "0.38" }
pyo3 = { version = "0.20", features = ["extension-module", "abi3-py310"] }
pyo3-polars = { version = "0.12", features = ["lazy"] }
```

```
$ maturin build --release
🔗 Found pyo3 bindings with abi3 support for Python ≥ 3.10
🐍 Not using a specific python interpreter
Compiling…
Finished release [optimized] target(s) in 49.26s
📦 Built wheel for abi3 Python ≥ 3.10 to
target/wheels/polars_encrypt-0.1.0-cp310-abi3-manylinux_2_34_x86_64.whl
```


inovex

PyO3 Polars Example Module


```rust
fn parallel_encrypt_data_frame(
    df: DataFrame,
    col: &str,
    password: &str,
) -> PolarsResult<DataFrame> { … }

#[pyfunction]
fn parallel_encrypt(pydf: PyDataFrame, col: &str, password: &str) -> PyResult<PyDataFrame> {
    let df: DataFrame = pydf.into();
    let df_encrytped = parallel_encrypt_data_frame(df, col, password).map_err(PyPolarsErr::from)?;
    Ok(PyDataFrame(df_encrytped))
}

#[pymodule]
fn extend_polars(_py: Python, m: &PyModule) -> PyResult<()> {
    m.add_function(wrap_pyfunction!(parallel_encrypt, m)?)?;
    m.add_function(wrap_pyfunction!(parallel_decrypt, m)?)?;
    m.add_function(wrap_pyfunction!(lazy_parallel_encrypt, m)?)?;
    m.add_function(wrap_pyfunction!(lazy_parallel_decrypt, m)?)?;
    Ok(())
}
```


inovex

`map_elements` vs PyO3

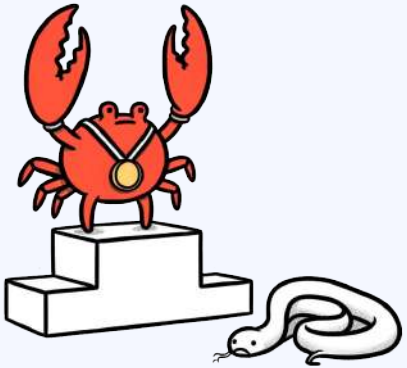
Encrypt & Decrypt **916_939** lines of 9 chars:

`map_elements`:

```
Elapsed time: 14s
Percent CPU: 101%
Maximum resident set size: 5GB
```

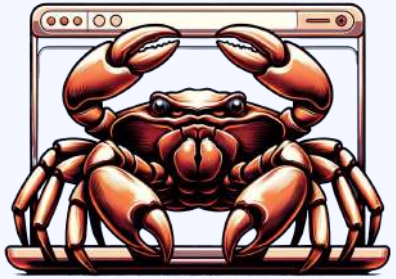
PyO3 module:

```
Elapsed time: 1s
Percent CPU: 500%
Maximum resident set size: 0.6GB
```



inovex

Polars in the Browser (WASM)



- Polars can be compiled to WebAssembly (WASM)
- Not fully supported (features have to be disabled)
- Basic DataFrame and CSV reader are working

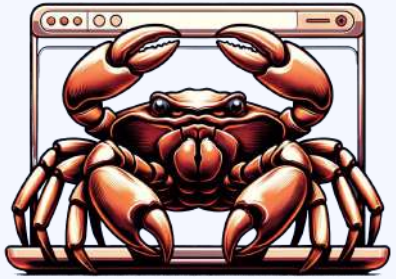
```
[dependencies.polars]
version = "0.37.0"
default-features = false
features = [
    "abs", "regex",
    "diff","dtype-full",
    "round_series", "lazy",
    "mode", "product",
    "rank", "reinterpret", "rows",
    "strings", "semi_anti_join",
    "unique_counts","csv",
    # "parquet", "ipc", "performant" :(
]
```

```
$ wasm-pack build
[INFO]: 🎯  Checking for the Wasm target...
[INFO]: 🌀  Compiling to Wasm…
Compiling…
Finished release [optimized] target(s) in 2m 32s
[INFO]: ⬇️  Installing wasm-bindgen...
[INFO]: ✨   Done in 2m 35s
[INFO]: 📦   Your wasm pkg is ready to publish at /tmp/polars-wasm/pkg.

$ ls /tmp/polars-wasm/pkg
README.md package.json
polars_wasm_bg.js polars_wasm_bg.wasm.d.ts
polars_wasm_bg.wasm (13 MB / 2 MB gzip)
polars_wasm.d.ts polars_wasm.js
```


inovex

Polars in the Browser (WASM)



```rust
#[wasm_bindgen]
pub fn describe(csv_data: &str, date_cols_js: JsValue) -> Result<String, String> {
    let cursor = Cursor::new(csv_data.as_bytes());
    let df_result = CsvReader::new(cursor).infer_schema(None).finish();
    let df = match df_result {
        Ok(df) => df,
        Err(err) => return Err(err.to_string()),
    };

    let date_cols: HashMap<String, String> = from_value(date_cols_js).map_err(|e| e.to_string())?;
    let mut map: HashMap<String, Stats> = HashMap::new();

    for series in df.get_columns() { … }

    let result = to_string(&map).map_err(|e| e.to_string())?;
    Ok(result)
}
```


inovex

SQL Context

- `SQLContext` allows mapping `DataFrames` and `LazyFrames` names to their datasets
- **No** separate SQL engine
- SQL queries become polars expressions
- Polars does not support the full SQL Language

inovex

SQL Context Support

Supports:

- Write a `CREATE` statements
- Write a `SELECT` statements with all generic elements (`GROUP BY`, `WHERE`,`ORDER`,...)
- Write Common Table Expressions (`WITH table AS`)
- Show an overview of all tables `SHOW TABLES`

Not Supported:

- `INSERT`, `UPDATE`, `DELETE`
- Table aliasing
- Meta queries (`ANALYZE`, `EXPLAIN`)

inovex

SQL Example

```
ctx = pl.SQLContext()
e_commerce_events = pl.scan_csv("2019-Nov.csv")
ctx.register("e_commerce_events", e_commerce_events)

lf = ctx.execute(
    """
    WITH purchases AS (
        SELECT
          product_id, brand, price FROM e_commerce_events
        WHERE event_type = 'purchase'
    )
    SELECT
      product_id, brand, count(*) AS purchase_count,
      min(price) AS price_min, avg(price) AS price_avg,
      max(price) AS price_max,
      stdev(price) AS price_stdev
    FROM purchases
    GROUP BY product_id, brand
    HAVING purchase_count > 10
    ORDER BY price_stdev DESC
    LIMIT 10
    """,
    eager=False,
)
```

```
SORT BY [col("price_stdev")]
  FAST_PROJECT: [product_id, brand, purchase_count, price_min,
price_avg, price_max, price_stdev]
    FILTER [(col("purchase_count")) > (10)] FROM

    AGGREGATE
        [len().alias("purchase_count"), col("price").min().alias("price_min"),
         col("price").mean().alias("price_avg"),
         col("price").max().alias("price_max"),
         col("price").std().alias("price_stdev")]
    BY [col("product_id"), col("brand")] FROM
      FAST_PROJECT: [product_id, brand, price]

        Csv SCAN 2019-Nov.csv
        PROJECT 4/9 COLUMNS
        SELECTION: [(col("event_type")) == (String(purchase))]
```

product_id	brand	purchase_count	price_min	price_avg	price_max	price_stdev
i64	str	u32	f64	f64	f64	f64
2200982	canon	33	1389.97	1670.702121	1976.92	286.925209
1004159	samsung	38	769.13	947.364211	1253.07	236.495904
15100141	null	22	493.96	1461.552727	1569.92	227.148752
15100239	ostamebel	35	308.86	650.714	823.7	220.342035
2200486	nikon	11	1261.27	1434.444545	1698.89	209.794349
1801795	sony	21	2059.0	2334.792857	2445.11	178.734199
1801938	sony	29	1157.79	1449.722069	1544.16	167.64585
1306571	acer	29	1801.82	2083.653448	2312.8	166.756888
2701627	lg	18	1645.81	1871.702778	2004.43	141.609503
1305215	apple	24	1209.55	1326.735833	1503.49	138.539943

inovex

SQL Example


```
ctx = pl.SQLContext()
e_commerce_events = pl.scan_csv("2019-Nov.csv")
ctx.register("e_commerce_events", e_commerce_events)

lf = ctx.execute(
    """
    WITH purchases AS (
        SELECT
          product_id, brand, price FROM e_commerce_events
        WHERE event_type = 'purchase'
    )
    SELECT
      product_id, brand, count(*) AS purchase_count,
      min(price) AS price_min, avg(price) AS price_avg,
      max(price) AS price_max,
      stdev(price) AS price_stdev
    FROM purchases
    GROUP BY product_id, brand
    HAVING purchase_count > 10
    ORDER BY price_stdev DESC
    LIMIT 10
    """,
    eager=False,
)
```

```
lf = pl.scan_csv("2019-Nov.csv")
purchases = lf
    .filter(pl.col("event_type") == "purchase")
    .select(
            [
            pl.col("product_id"),
            pl.col("brand"),
            pl.col("price")
        ]
    )
result = (
    purchases.group_by(["product_id", "brand"])
    .agg(
        [
            pl.len().alias("purchase_count"),
            pl.min("price").alias("price_min"),
            pl.mean("price").alias("price_avg"),
            pl.max("price").alias("price_max"),
            pl.std("price").alias("price_stdev"),
        ]
    )
    .filter(pl.col("purchase_count") > 10)
    .sort(by="price_stdev", descending=True)
    .limit(10)
)
```


inovex

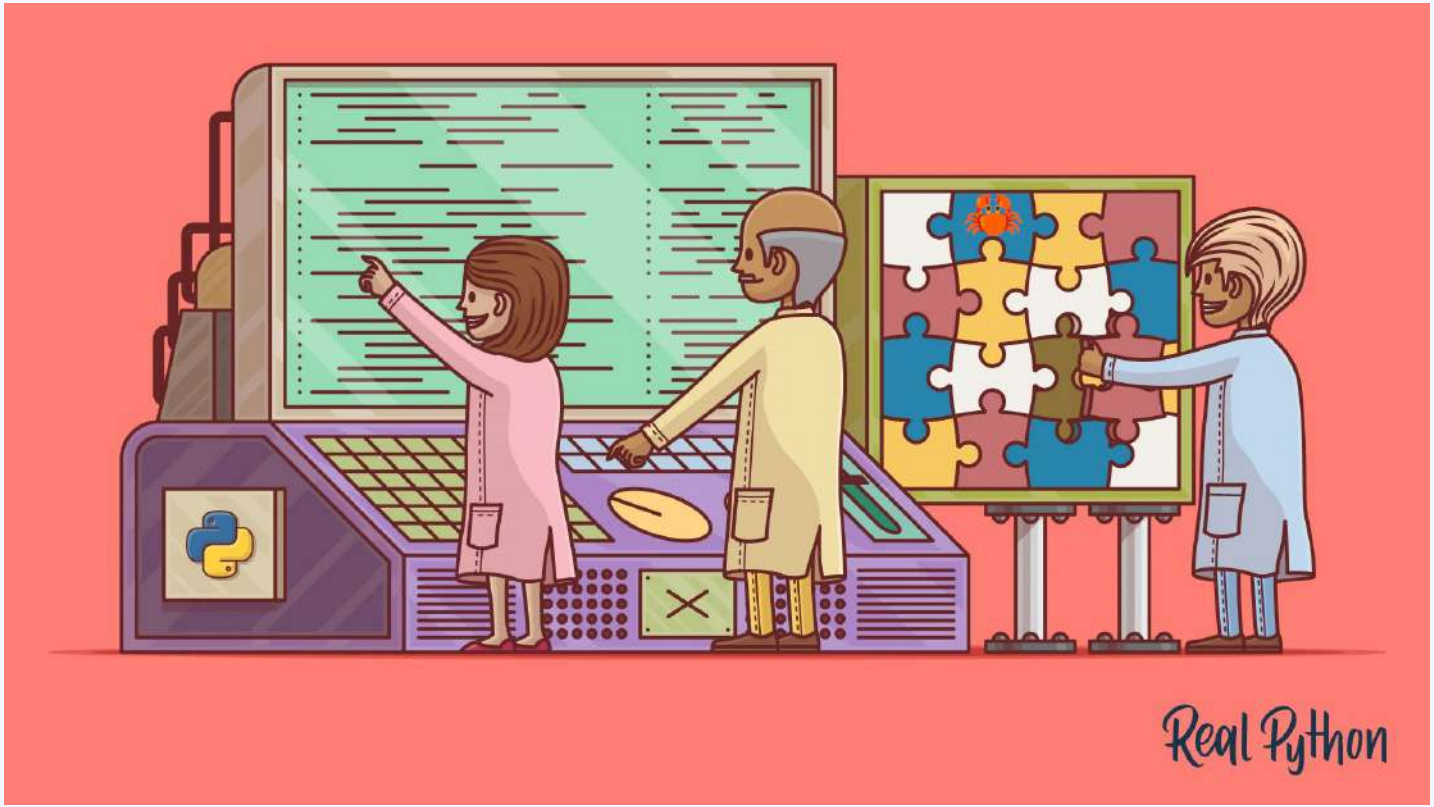
polars-cli

- Run SQL queries on datasets from your command line
- Input formats: `csv`, `json`, `parquet`, `IPC/Apache Arrow`
- Output formats: `csv`, `json`, `parquet`, `arrow`, `table`, `markdown`

```
polars -o csv -c \
  "WITH purchases AS (SELECT product_id, brand, price FROM
   read_parquet('2019-Nov.parquet') WHERE event_type = 'purchase')
   SELECT product_id, brand, count(*) AS c, avg(price) AS avg_price
   FROM purchases GROUP BY product_id, brand ORDER BY c DESC"
```


inovex

Why should I use this and not a Python script?



Source: realpython.com

inovex

Digression: DuckDB

DuckDB

- DuckDB is a fast in-process analytical database
- Github: 15k ⭐
- License: MIT
- Feature-Rich
  - ACID guarantees
  - Multi-Version Concurrency Control (MVCC)
  - Complex queries with vectorized execution
  - nested correlated subqueries
  - window functions
  - collations
  - complex types (arrays, structs)

inovex

Polars / DuckDB Integration

```python
import polars as pl
import duckdb

df = pl.DataFrame(
    {
        "A": [1, 2, 3, 4, 5],
        "fruits": ["🍌", "🍏", "🥭", "🍏", "🍌"],
        "B": [5, 4, 3, 2, 1],
    }
)
duckdb.sql("SELECT * FROM df").show()
```

```python
df = duckdb.sql("""
    SELECT 1 AS id, 'banana' AS fruit
    UNION ALL
    SELECT 2, 'apple'
    UNION ALL
    SELECT 3, 'mango'""""
).pl()
print(df)
```


inovex

Resources

- https://pola.rs/ - Project website
- What is Polars? - Polars explained in 90s by the creator
- Podcasts: TalkPython, RealPython, Rustacean Station
- https://github.com/meldron/polars-expedition

inovex

Thanks a lot!

Bernd Kaiser

bernd.kaiser@inovex.de

Allee am Röthelheimpark 11
91052 Erlangen

Bernd Kaiser

@meldron

kaiser.land

inovex