

Serpents and Ducks

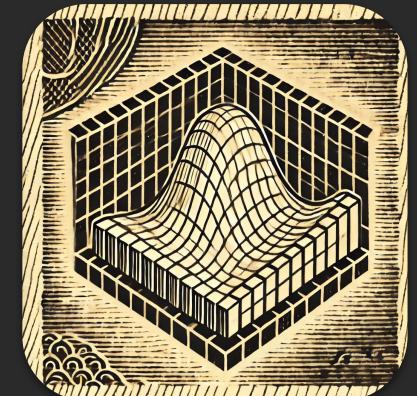
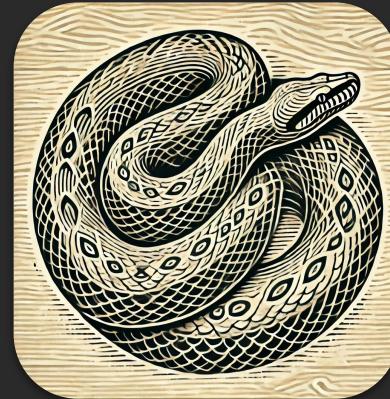
Wrangling data with
Python and DuckDB



Simon Aubury

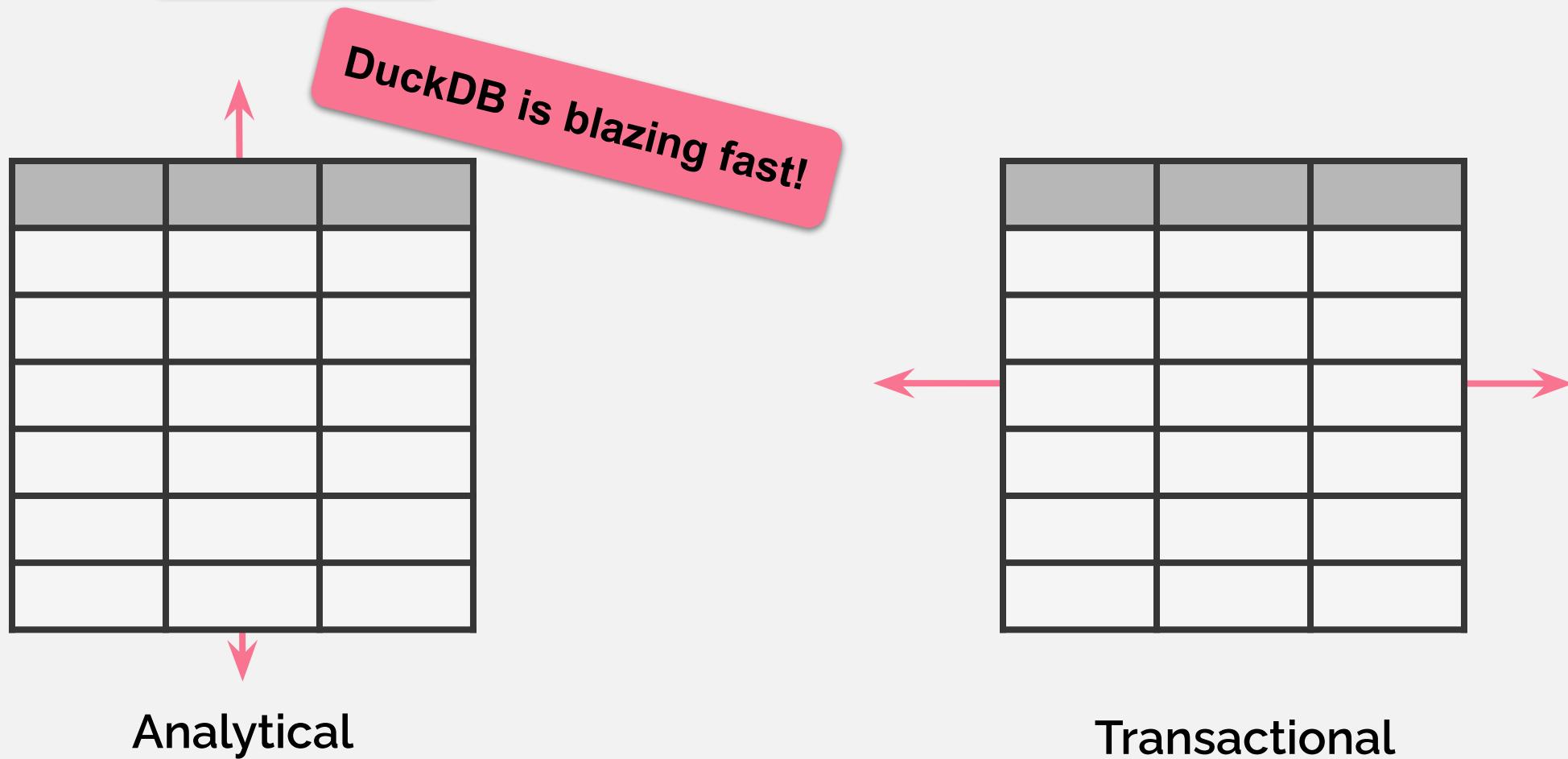


Ned Letcher



DuckDB is an analytical in-process SQL database management system.

DuckDB is an analytical in-process SQL database management system.



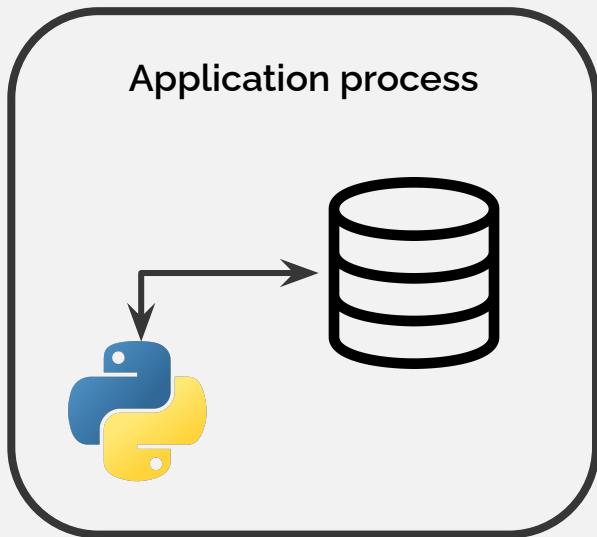
Google
Big Query



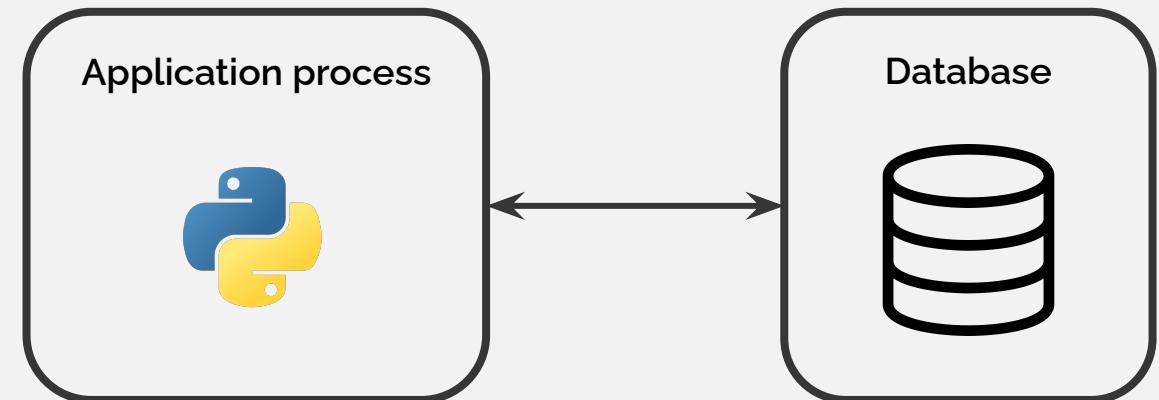
PostgreSQL



*DuckDB is an analytical **in-process** SQL database management system.*



In-process



Client-server

DuckDB is an analytical in-process SQL database management system.

Structured Query Language

Friendly SQL enhancements!

```
SELECT
    product_id,
    sale_month,
    SUM(sale_amount) AS total_sales,

    FROM product_sales

    WHERE region = 'Australia'

    GROUP BY product_id, sale_month

    ORDER BY sale_month, product_id;
```

DuckDB is an analytical in-process SQL database management system.



DuckDB is versatile

What does data analysis with DuckDB look like?



By Elekhh - Own work, CC BY-SA 3.0

```
[1]: import pandas as pd
```

```
df = pd.read_csv(  
    "data/melb_bike_share.csv",  
    usecols=["ID", "NAME", "NBBIKES", "RUNDATE", "LAT", "LONG"],  
    parse_dates=["RUNDATE"],  
    date_format="%Y%m%d%H%M%S",  
)
```

```
df
```

	ID	NAME	NBBIKES	RUNDATE	LAT	LONG
0	2	Harbour Town - Docklands Dve - Docklands	10	2017-04-22 13:45:06	-37.814022	144.939521
1	4	Federation Square - Flinders St / Swanston St ...	9	2017-04-22 13:45:06	-37.817523	144.967814
2	6	State Library - Swanston St / Little Lonsdale ...	1	2017-04-22 13:45:06	-37.810702	144.964417
3	7	Bourke Street Mall - 205 Bourke St - City	4	2017-04-22 13:45:06	-37.813088	144.967437
4	8	Melbourne Uni - Tin Alley - Carlton	8	2017-04-22 13:45:06	-37.796250	144.960858
...
5644549	11	MSAC - Aughtie Dve - Albert Park	25	2018-09-01 05:45:06	-37.842395	144.961868
5644550	12	Fitzroy Town Hall - Moor St - Fitzroy	10	2018-09-01 05:45:06	-37.801813	144.979209
5644551	14	Plum Garland Reserve - Beaconsfield Pde - Albe...	19	2018-09-01 05:45:06	-37.847795	144.948351
5644552	15	Coventry St / St Kilda Rd - Southbank	5	2018-09-01 05:45:06	-37.828887	144.970822
5644553	16	NAB - Harbour Esp / Bourke St - Docklands	5	2018-09-01 05:45:06	-37.818306	144.945923

5644554 rows × 6 columns



⌚ 9 seconds

```
[2]: import duckdb
```



```
bikes_rel = duckdb.sql(  
    """  
        SELECT ID, NAME, NBBIKES, RUNDATE, LAT, LONG  
        FROM read_csv(  
            'data/melb_bike_share.csv',  
            types={'RUNDATE': TIMESTAMP},  
            timestampformat='%Y%m%d%H%M%S'  
        )  
    """  
)  
  
bikes_rel
```

ID int64	NAME varchar	NBBIKES int64	RUNDATE timestamp	LAT double	LONG double
2	Harbour Town - Docklands Dve - Docklands	10	2017-04-22 13:45:06	-37.814022	144.939521
4	Federation Square - Flinders St / Swanston St - City	9	2017-04-22 13:45:06	-37.817523	144.967814
6	State Library - Swanston St / Little Lonsdale St - City	1	2017-04-22 13:45:06	-37.810702	144.964417
7	Bourke Street Mall - 205 Bourke St - City	4	2017-04-22 13:45:06	-37.813088	144.967437
8	Melbourne Uni - Tin Alley - Carlton	8	2017-04-22 13:45:06	-37.79625	144.960858
.
.
.
21	Bridport St / Montague St - Albert Park	11	2017-04-09 07:45:06	-37.840885	144.955303
22	Pickles St / Ingles St - Port Melbourne	9	2017-04-09 07:45:07	-37.835803	144.94852
23	Yarra's Edge - River Esp / Yarra River - Docklands	1	2017-04-09 07:45:07	-37.824468	144.946033
24	North Melbourne Station - Adderley St - North Melbourne	11	2017-04-09 07:45:07	-37.807021	144.941854
25	Sandridge Bridge - Southbank	14	2017-04-09 07:45:07	-37.820836	144.962266

? rows (>9999 rows, 10 shown) 6 columns

⌚ 0.2 seconds

```
[3]: monthly_bikes_rel = duckdb.sql(  
    """  
    SELECT  
        month(RUNDATE) AS MONTH,  
        round(avg(NBBIKES), 2) AS AVG_BIKES,  
    FROM bikes_rel  
    WHERE year(RUNDATE) = 2017  
    GROUP BY MONTH  
    ORDER BY MONTH  
    """  
)
```

```
monthly_bikes_rel
```

MONTH	AVG_BIKES
1	9.84
2	9.92
3	8.77
4	8.6
5	8.83
6	8.7
7	8.71
8	8.87
9	8.82
10	8.48
11	8.48
12	8.48
12 rows	

⌚ 0.1 seconds

```
avg_bikes_df = (  
    bikes_df[bikes_df["RUNDATE"].dt.year == 2017]  
    .groupby(bikes_df["RUNDATE"].dt.month)[ "NBBIKES" ]  
    .mean()  
    .round(2)  
    .reset_index(name="AVG_BIKES")  
    .sort_values("RUNDATE")  
)
```



Replacement scan

What's the average number of bikes per station by month, for 2017?

Exporting the results of our analysis



```
[4]: duckdb.sql("COPY monthly_bikes_rel TO 'data/monthly_avg_bikes.csv'")
```



```
[5]: duckdb.sql("COPY monthly_bikes_rel TO 'data/monthly_avg_bikes.parquet'")
```



```
[6]: duckdb.sql("COPY monthly_bikes_rel TO 'data/monthly_avg_bikes.json'")
```

Configuring the output format

```
[8]: duckdb.sql("COPY monthly_bikes_rel TO 'data/monthly_avg_bikes.json' (ARRAY true)")
```

```
[7]: duckdb.sql("COPY monthly_bikes_rel TO 'data/monthly_avg_bikes.csv' (DELIMITER '|')")
```

Python data-structure integrations

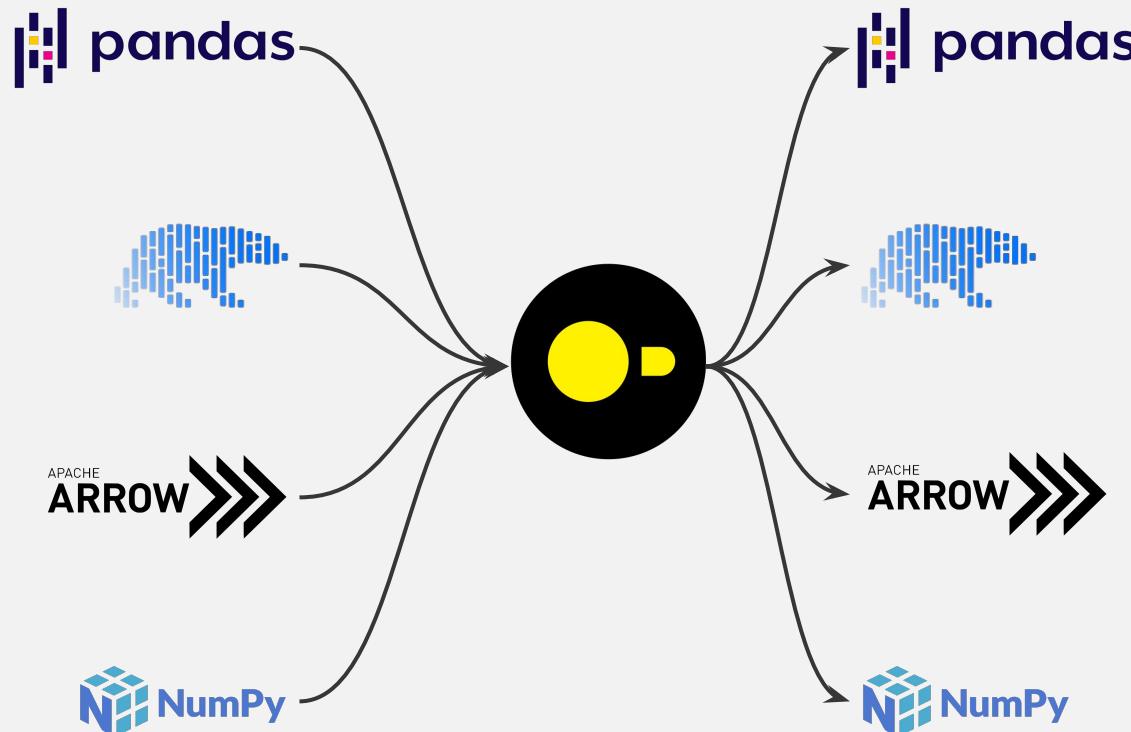
SQL on Dataframes!

```
duckdb.sql("SELECT * FROM pandas_df")
```

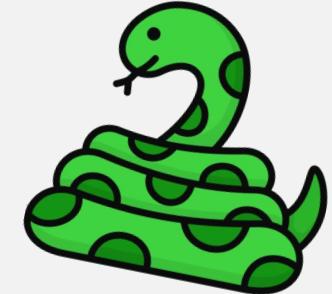
```
duckdb.sql("SELECT * FROM polars_df")
```

```
duckdb.sql("SELECT * FROM arrow_table")
```

```
duckdb.sql("SELECT * FROM numpy_array")
```

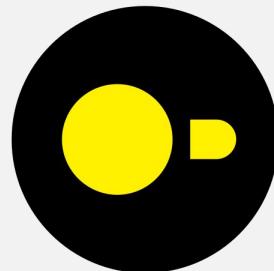


What if I don't want to use SQL?



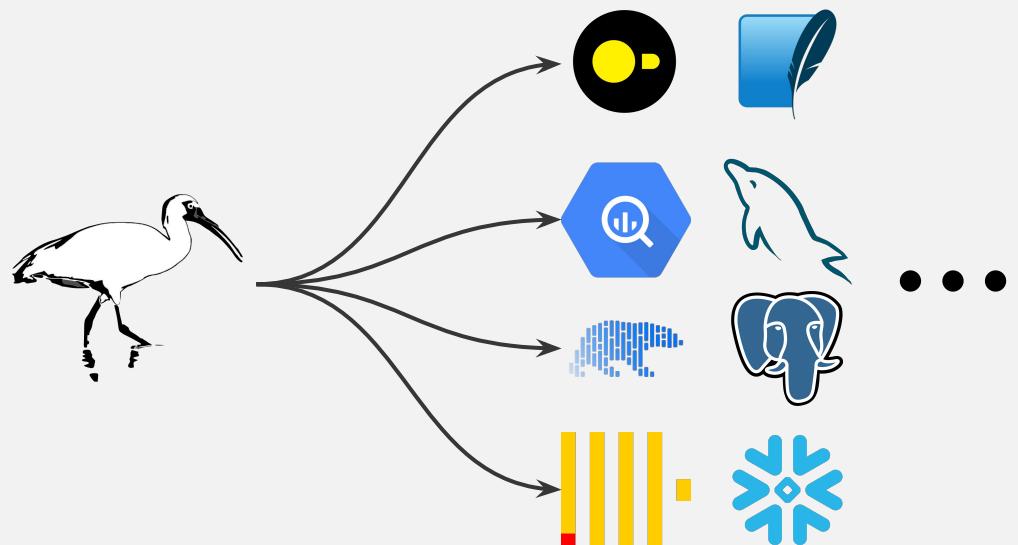
DuckDB Relational API

- ❖ Enables programmatic composition of queries
- ❖ Targets native DuckDB bindings
- ❖ Part of DuckDB Python client

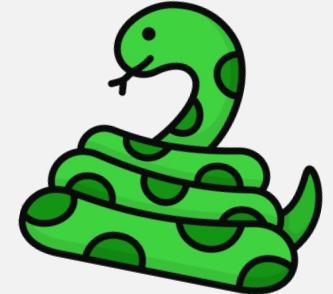


Ibis

- ❖ Dataframe-like API
- ❖ Targets a wide range of SQL dialects
- ❖ Third party Python library



What if I don't want to use SQL?



DuckDB Relational API

```
import duckdb

duckdb.read_csv(
    "data/melb_bike_share.csv",
    dtype={"RUNDAT": "TIMESTAMP"},
    timestamp_format="%Y%m%d%H%M%S",
).filter("year(RUNDAT) = 2017").count(*)
```

Ibis

```
import ibis
from ibis import _

ibis.read_csv(
    "data/melb_bike_share.csv",
    types={"RUNDAT": "TIMESTAMP"},
    timestampformat="%Y%m%d%H%M%S"
).filter(_.RUNDAT.year() == 2017).count()
```

What's the total number of readings for 2017?

DuckDB

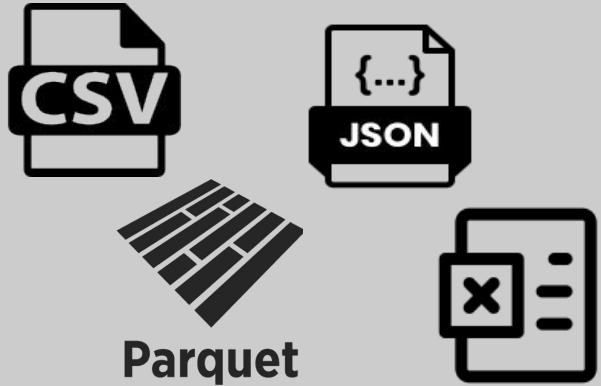
- the database



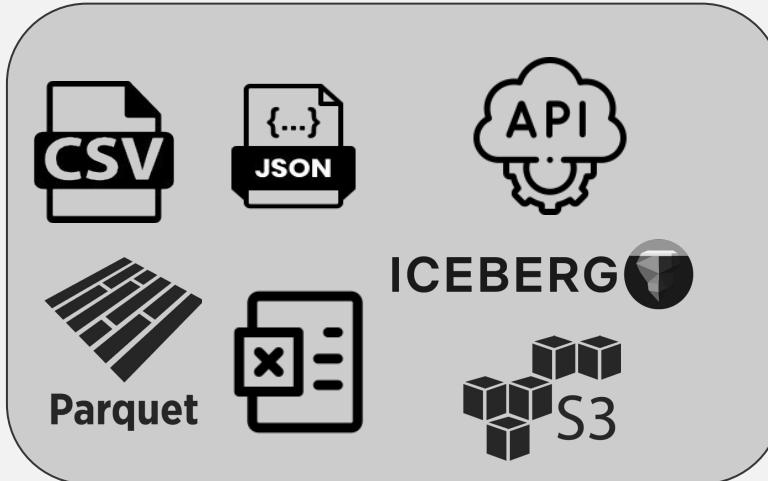
Data, data, everywhere!



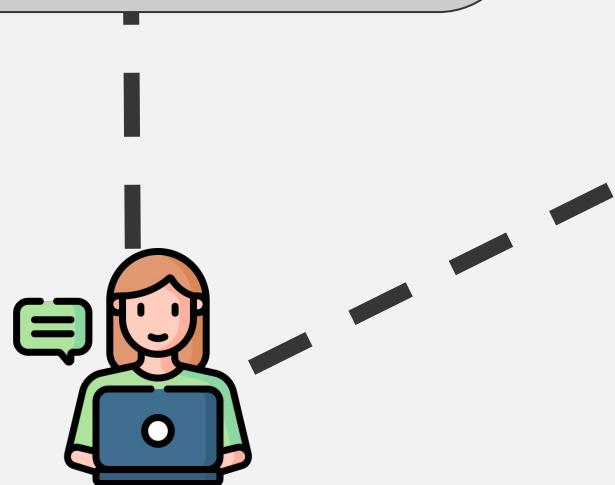
Local



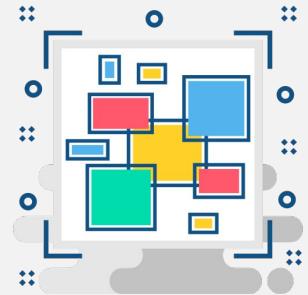
Cloud



Databases



Formats



Options for pesky data wrangling

```
SELECT *  
FROM read_json('exercise-*.json'  
, timestampformat='%m/%d/%y %H:%M:%S')
```



Credentials, compression, partitioning



```
SELECT *  
FROM read_parquet('cafes_encrypted.parquet',  
| encryption_config = {footer_key: 'key128'})
```

Even from Excel

```
SELECT *  
FROM st_read('data_excel.xlsx'  
, layer='Sheet1')
```



DuckDB extensions



Microsoft Azure
Blob Storage



Full text search



Hugging Face

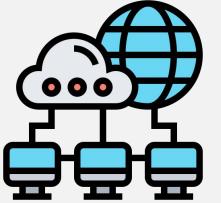


Spatial / H3



Spatial / PostGIS

Locations



Any s3 compatible blob storage

```
SELECT *
FROM read_parquet('s3://duckdb-s3-bucket-public/countries.parquet')
WHERE name SIMILAR TO '.*Republic.*';
```



HTTP / HTTPS



```
SELECT *
FROM read_parquet('https://example.net/yellow_tripdata_2024-01.parquet')
```

Rest API

```
SELECT json_extract(hourly, '$.temperature_2m[1]')
FROM read_json('https://api.open-meteo.com/v1/forecast?latitude=-33.8678&longitude=-70.6693&forecast_days=1');
```



It's smarter than it looks



```
SELECT count(*), max(ride_amt)
FROM read_parquet('s3://mybucket/taxi/*/*/*.parquet',
hive_partitioning = true)
WHERE year = 2024 AND month = 9
AND passengers > 4;
```

Projection pushdown

Filter pushdown zone maps

1	3	2	2
2	2	1	4
2	4	1	2
7	3	5	1

Hive partitioning

```
└── dt=2001-01-01/
    └── country=GB/
        └── file1
        └── file2
    └── country=US/
        └── file3
└── dt=2001-01-02/
    └── country=GB/
        └── file4
```

Wrangling data with Python and DuckDB





Good cafes?

CITY OF MELBOURNE OPEN DATA

Home Explore our Data Analysis and Insight Visualise our Data Data Stories About our Data Our API Help Contact Us

60,055 records

No active filters

Filters

Search records...

Census year

2017	3,569
2016	3,560
2018	3,547
2019	3,535
2015	3,447
2014	3,357
> More	

Block ID

85	1,286
15	1,053
1108	1,044
64	1,004
58	954
1109	945
> More	

CLUE small areas

Melbourne City	1,286
Carlton	1,053

Persistent database

Create table

```
import duckdb
conn = duckdb.connect('persistent.db')

cafe_source_url = 'https://data.melbourne.vic.gov.au/.../cafes-and-restaurants'

conn.execute(f"""
CREATE OR REPLACE TABLE all_cafes AS
SELECT *
FROM read_csv('{cafe_source_url}')
""");
```

<https://geojson.io>



Install spatial extension

Filter to points within area

```
conn.install_extension("spatial")
conn.load_extension("spatial")

region_geojson_file = 'boundary_region.geojson'

conn.sql(f"""
CREATE OR REPLACE VIEW nearby_cafes_view AS
SELECT *
FROM all_cafes
WHERE st_within(
    st_point(longitude, latitude),
    (
        SELECT geom
        FROM st_read('{region_geojson_file}')
    )
)""")
```

Good cafes?



Filter, order and limit the view

```
conn.view('nearby_cafes_view') \
    .filter("industry_description = 'Cafes and Restaurants'"") \
    .filter("seating_type = 'Seats - Indoor'"") \
    .order("number_of_seats") \
    .limit(10).df()
```

trading_name	address	seating_type	number_of_seats	industry_description	cou
Caffe Orr	Part 2 Convention Centre Place SOUTH WHARF VIC...	Seats - Outdoor	5	Cafes and Restaurants	
Café Orr	Part 2 Convention Centre Place SOUTH WHARF VIC...	Seats - Outdoor	8	Cafes and Restaurants	
Caffe Cino	Part 2 Convention Centre Place SOUTH WHARF 3006	Seats - Outdoor	9	Cafes and Restaurants	
Akachochin	33-65 South Wharf Promenade SOUTH WHARF 3006	Seats - Outdoor	12	Cafes and Restaurants	
Signature Pho Viet	20 Convention Centre Place SOUTH WHARF 3006	Seats - Outdoor	16	Cafes and Restaurants	
Akachochin	33 South Wharf Promenade SOUTH WHARF VIC 3006	Seats - Outdoor	20	Cafes and Restaurants	

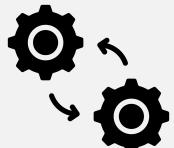
Consider adding DuckDB to your Python data toolkit!



Versatile



Powerful



Interoperable



Frictionless





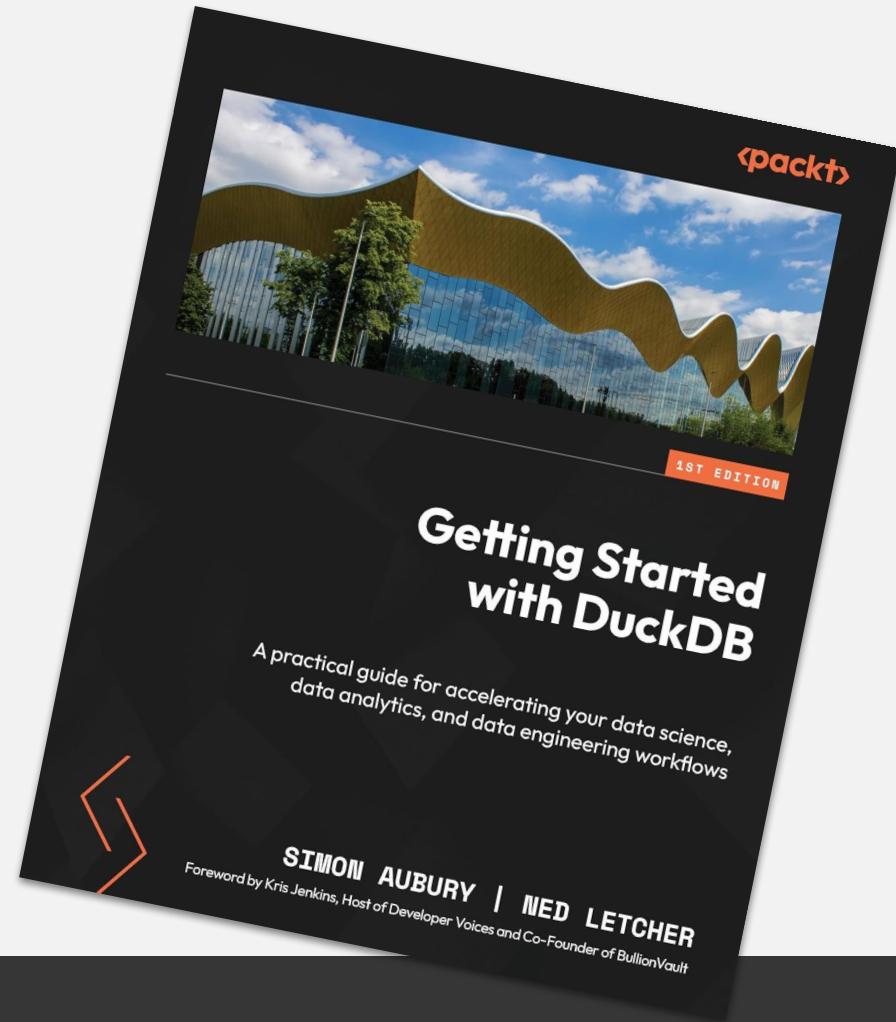
<https://github.com/ned2/duckdb-pyconau-2024>



Simon Aubury



Ned Letcher



Slides: SlidesMania

Fonts used: Raleway
Images: Freepik