\Orchestrating a brighter world NEC

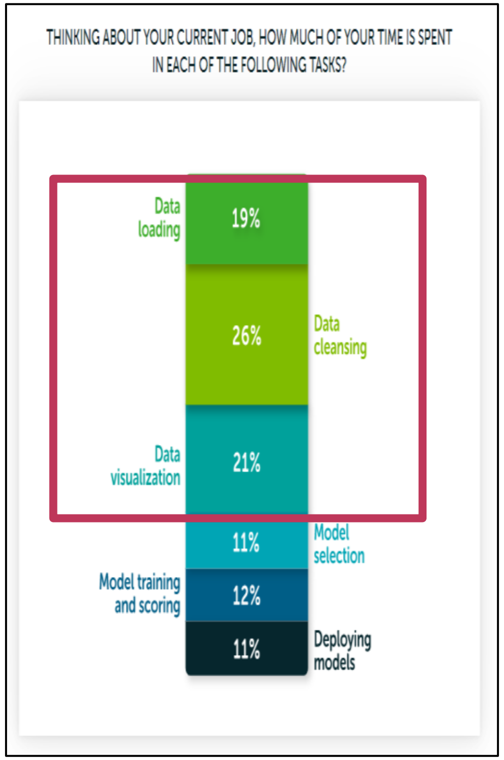
Accelerate your pandas workload using FireDucks

August 31, 2024

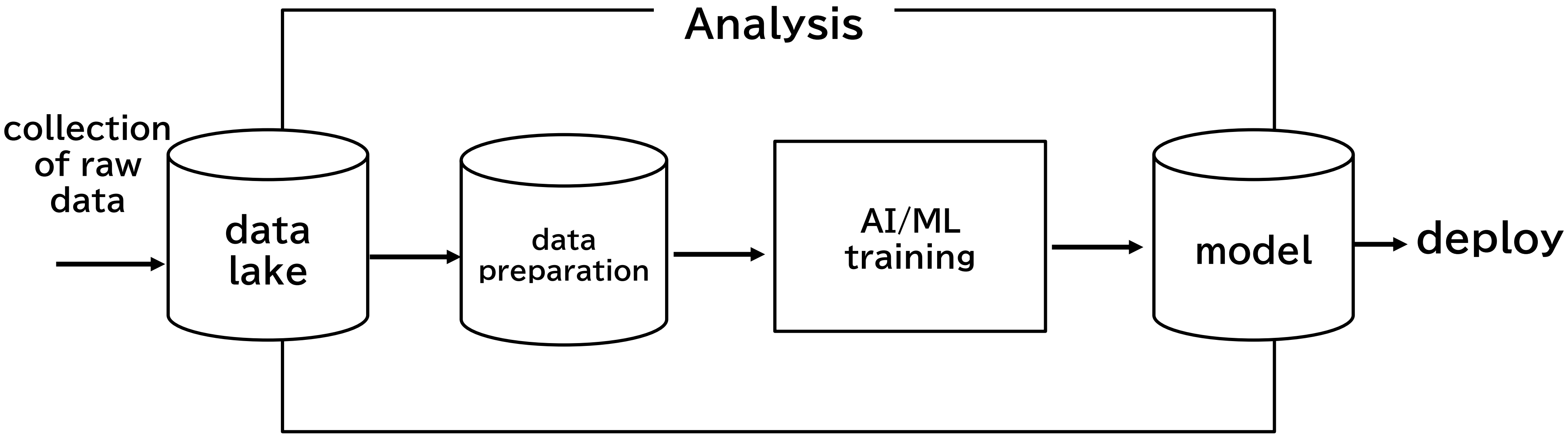
Sourav Saha (NEC)

Workflow of a Data Scientist

almost 75% efforts of a Data Scientist spent on data preparation

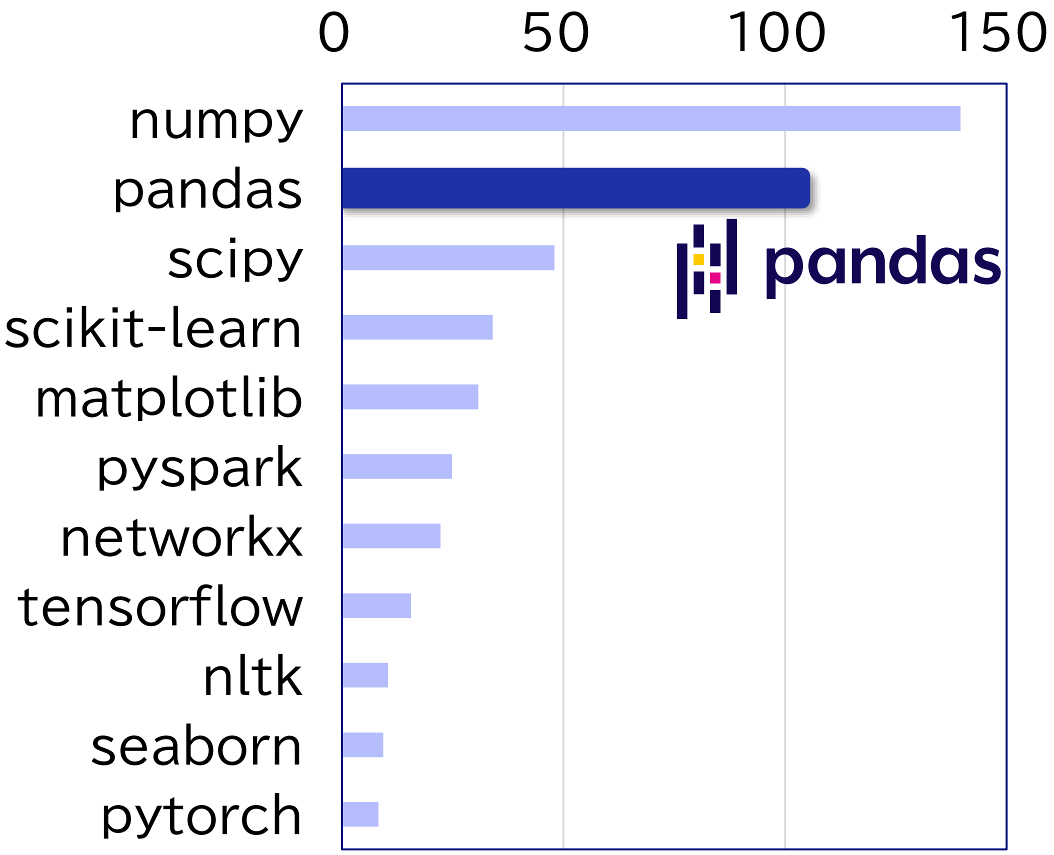


Anaconda:
The State of Data Science 2020



Pandas: Its Pros and Cons

◆ **Most popular Python library for data analytics.**



Monthly download from pypi.org
(Data Analytics Libraries)

- pandas drawbacks:

  - It (mostly) doesn't support parallel computation.
  - The choice of API heavily impacts the performance of a pandas application.
  - Very slow execution reduces the efficiency of a data analyst.
  - Long-running execution
    - produces higher cloud costs
    - attributes to higher CO2 emission

The way of implementing a query in pandas-like library (that does not support query optimization) heavily impacts its performance!!

Ice-Breaking Session

(test your pandas skill)

Quick check on basic pandas operations (1/5)

- **Which one of the following is the right method of getting top-2 rows based on the column "A" from table "df"?**

1. df.sort("A", ascending=True).head(2)
2. df["A"].top_k(2)
3. df.sort("A", ascending=False).first(2)
4. df.sort_values("A", ascending=False).head(2)

	A	B
0	2	10
1	5	30
2	1	20
3	3	70
4	7	60
5	8	40
6	4	80

	A	B
5	8	40
4	7	60

Quick check on basic pandas operations (2/5)

◆ **Which ones of the following are the right methods of performing inner-join of table “df1” with table “df2” on common key-column “x”?**

1. pd.merge(df1, df2, on=“x”, how=“inner”)
2. df1.inner_join(df2, on=“x”)
3. df1.merge(df2, on=“x”, how=“inner”)
4. df1.merge(df2, on=“x”)

df1

	x	y
0	0	10
1	1	20

df2

	x	z
0	0	100
1	1	200
2	2	300

	x	y	z
0	0	10	100
1	1	20	200

Quick check on basic pandas operations (3/5)

◆ Which one of the following is the right method of filtering rows where B > 20?

1. df.pipe(lambda t: t[t["B"] > 20])
2. df[df["B"] > 20]
3. df.query("B > 20")
4. All of the above

	A	B
0	2	10
1	5	30
2	1	20
3	3	70
4	7	60
5	8	40
6	4	80

➡

	A	B
1	5	30
3	3	70
4	7	60
5	8	40
6	4	80

```
df = df1.merge(df2, on="x")
res = df[df["B"] > 20]
```

```
res = df1.merge(df2, on="x").query("B > 20")
```

```
res = df1.merge(df2, on="x").pipe(lambda t: t[t["B"] > 20])
```

Quick check on basic pandas operations (4/5)

- **Which one of the following is the right method of selecting columns “A”, “D” and “E” from table “df”?**

1. df[[“A”, “D”, “E”]]
2. df.loc[:, [“A”, “D”, “E”]]
3. df.iloc[:, [0, 3, 4]]
4. All of the above

	A	B	C	D	E
0	2	10	10	g	9
1	5	30	69	a	2
2	1	20	31	g	8
3	3	70	45	f	3
4	7	60	59	e	1
5	8	40	66	f	1
6	4	80	97	h	8

	A	D	E
0	2	g	9
1	5	a	2
2	1	g	8
3	3	f	3
4	7	e	1
5	8	f	1
6	4	h	8

Quick check on basic pandas operations (5/5)

◆ **Select the options for appending a new column “F“ by doubling the column “B” from table “df”.**

1. df[“F”] = df[“B”] * 2
2. df.assign(F=lambda x: x[“B”] * 2)
3. df.with_columns(df.col(“B”) * 2).alias(“F”)
4. df.insert(5, "F", df["B"]*2)

	A	B	C	D	E
0	2	10	10	g	9
1	5	30	69	a	2
2	1	20	31	g	8
3	3	70	45	f	3
4	7	60	59	e	1
5	8	40	66	f	1
6	4	80	97	h	8

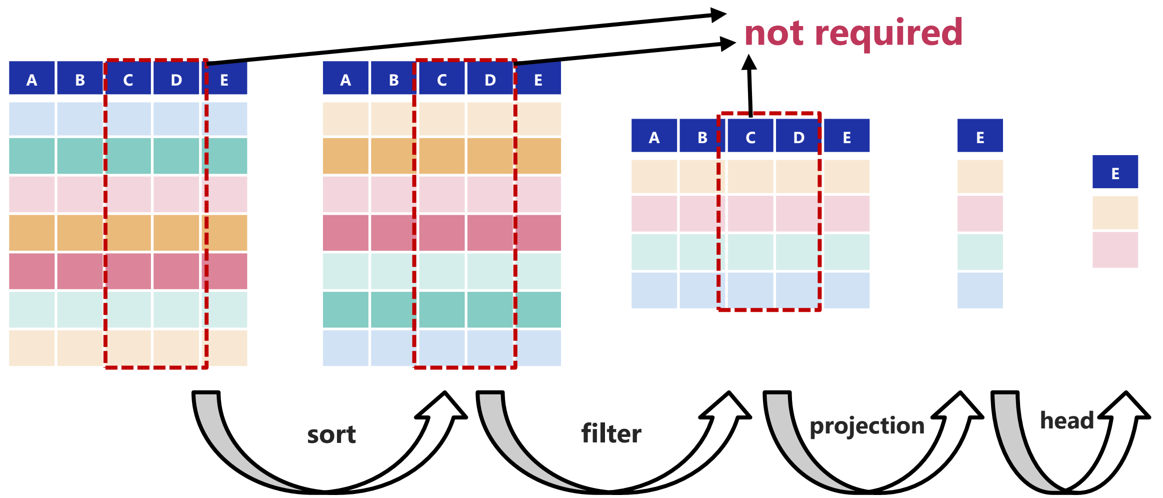
➡

	A	B	C	D	E	F
0	2	10	10	g	9	20
1	5	30	69	a	2	60
2	1	20	31	g	8	40
3	3	70	45	f	3	140
4	7	60	59	e	1	120
5	8	40	66	f	1	80
6	4	80	97	h	8	160

Execution order matters to boost the performance of a data analysis tool

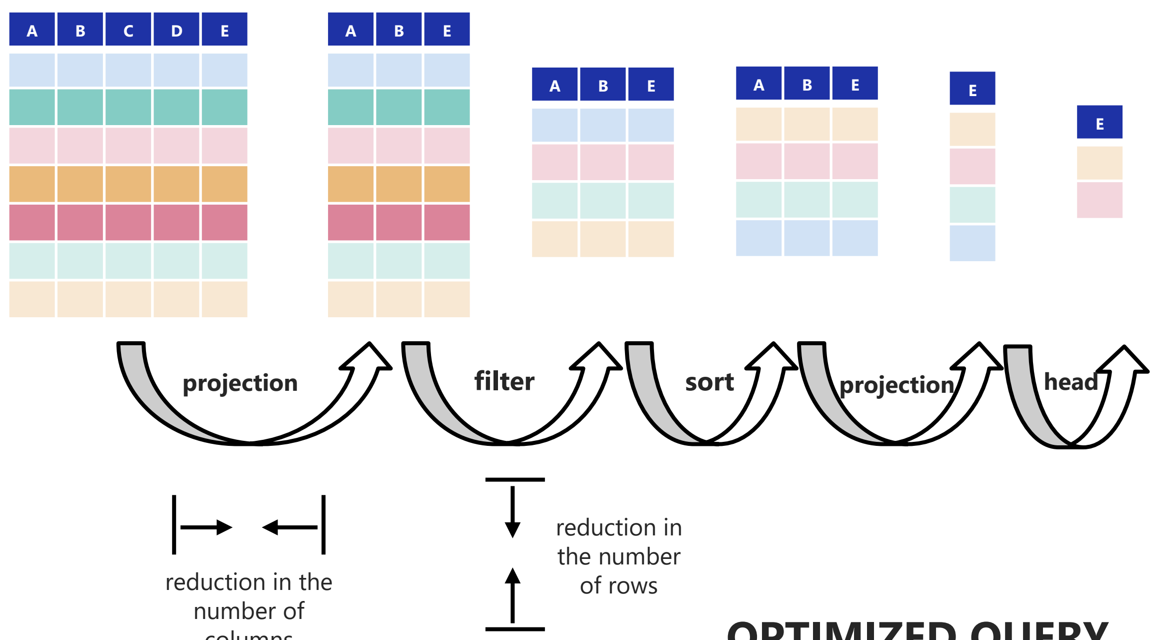
```
df.sort_values("A")
  .query("B > 1")["E"]
  .head(2)
```

※sort-order: yellow->red->green->blue
※B=1for darker shade, B=2 for lighter shade



SAMPLE QUERY

```
df.loc[:, ["A", "B", "E"]]
  .query("B > 1")
  .sort_values("A")["E"]
  .head(2)
```



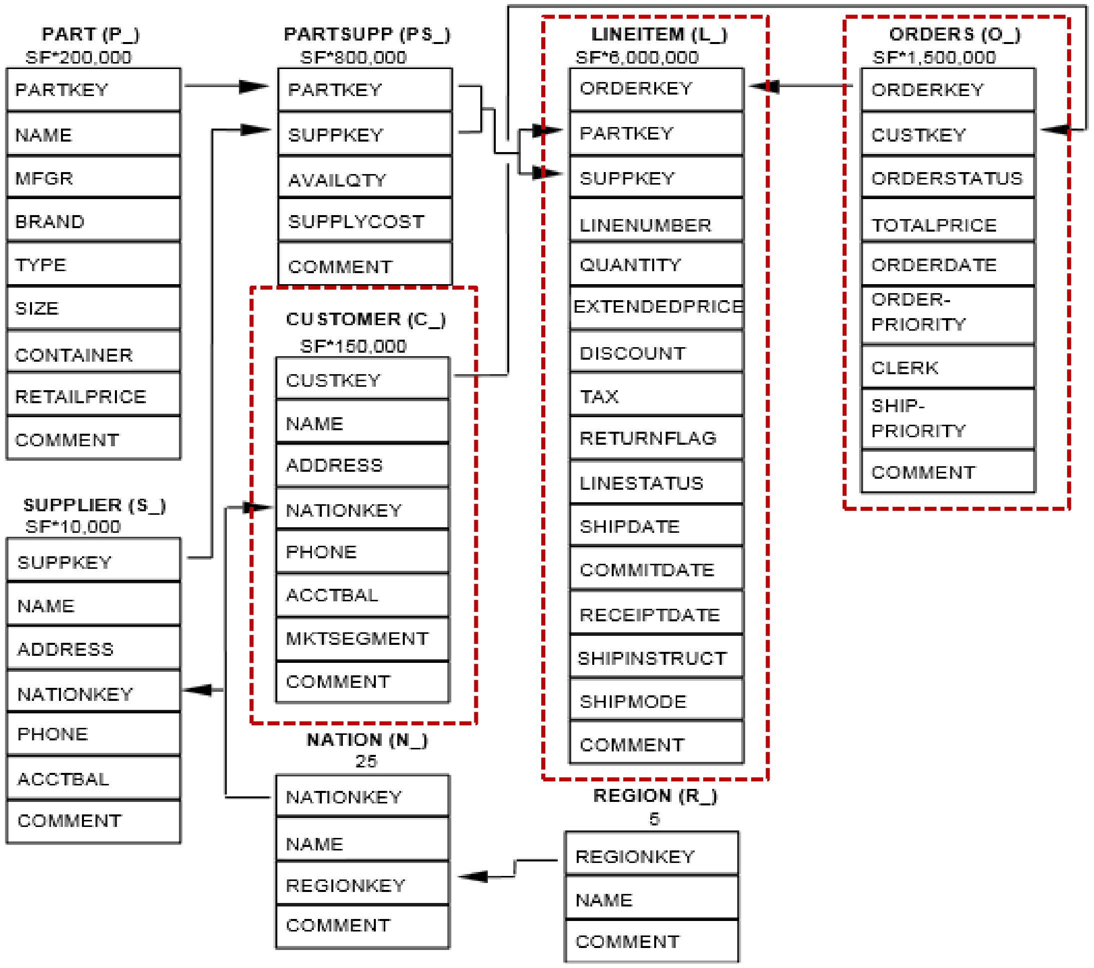
OPTIMIZED QUERY

Exercise: Query #3 from TPC-H Benchmark (SQL -> pandas)

- query to retrieve the 10 unshipped orders with the highest value.

```sql
SELECT l_orderkey,
               sum(l_extendedprice * (1 - l_discount)) as revenue,
               o_orderdate,
               o_shippriority
FROM customer, orders, lineitem
WHERE
    c_mktsegment = 'BUILDING' AND
    c_custkey = o_custkey AND
    l_orderkey = o_orderkey AND
    o_orderdate < date '1995-03-15' AND
    l_shipdate > date '1995-03-15'
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY revenue desc, o_orderdate
LIMIT 10;
```

```python
rescols = ["l_orderkey", "revenue", "o_orderdate", "o_shippriority"]
result = (
 customer.merge(orders, left_on="c_custkey", right_on="o_custkey")
  .merge(lineitem, left_on="o_orderkey", right_on="l_orderkey")
  .pipe(lambda df: df[df["c_mktsegment"] == "BUILDING"])
  .pipe(lambda df: df[df["o_orderdate"] < datetime(1995, 3, 15)])
  .pipe(lambda df: df[df["l_shipdate"] > datetime(1995, 3, 15)])
  .assign(revenue=lambda df: df["l_extendedprice"] * (1 - df["l_discount"]))
  .groupby(["l_orderkey", "o_orderdate", "o_shippriority"], as_index=False)
  .agg({"revenue": "sum"})[rescols]
  .sort_values(["revenue", "o_orderdate"], ascending=[False, True])
  .head(10)
)
```



Exercise: Query #3 from TPC-H Benchmark (pandas -> optimized pandas)

```
rescols = ["l_orderkey", “revenue”, "o_orderdate", "o_shippriority“]
result = (
 customer.merge(orders, left_on="c_custkey", right_on="o_custkey")
  .merge(lineitem, left_on="o_orderkey", right_on="l_orderkey")
  .pipe(lambda df: df[df[“c_mktsegment"] == "BUILDING"])
  .pipe(lambda df: df[df["o_orderdate"] < datetime(1995, 3, 15)])
  .pipe(lambda df: df[df["l_shipdate"] > datetime(1995, 3, 15)])
  .assign(revenue=lambda df: df["l_extendedprice"] * (1 - df["l_discount"]))
  .groupby(["l_orderkey", "o_orderdate", "o_shippriority"], as_index=False)
  .agg({"revenue": "sum”})[rescols]
  .sort_values(["revenue", "o_orderdate"], ascending=[False, True])
  .head(10)
)
```

Exec-time: 68.55 s

Scale Factor: 10

6.5x

Exec-time: 10.33 s

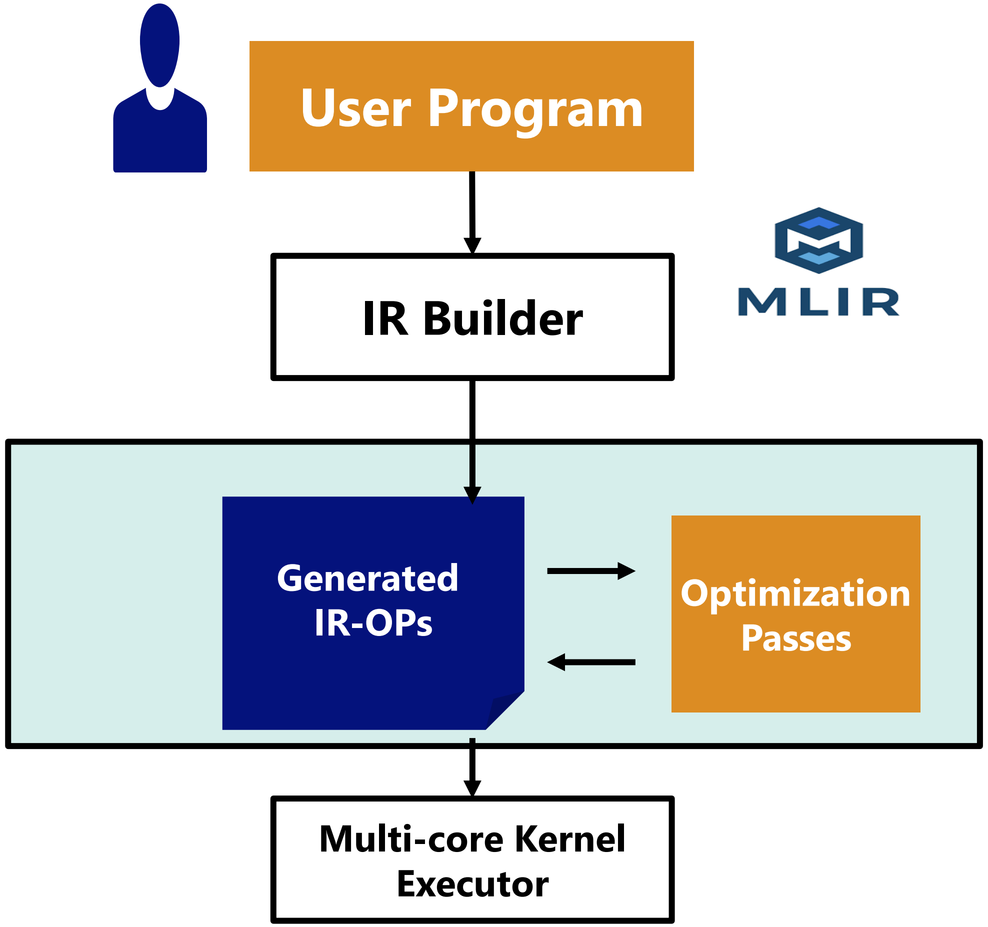
```
# projection-filter: to reduce scope of “customer” table to be processed
cust = customer[[“c_custkey”, “c_mktsegment"]] # (2/8)
f_cust = cust[cust[“c_mktsegment"] == "BUILDING"]

# projection-filter: to reduce scope of “orders” table to be processe
ord = orders[[“o_custkey”, “o_orderkey”, “o_orderdate”, "o_shippriority"]] (4/9)
f_ord = ord[ord["o_orderdate"] < datetime(1995, 3, 15)]

# projection-filter: to reduce scope of “lineitem” table to be processed
litem = lineitem[["l_orderkey“, "l_shipdate“, "l_extendedprice“, "l_discount"]] (4/16)
f_litem = litem[litem["l_shipdate"] > datetime(1995, 3, 15)]

rescols = ["l_orderkey", “revenue”, "o_orderdate", "o_shippriority“]
result = ( f_cust.merge(f_ord, left_on="c_custkey", right_on="o_custkey")
  .merge(f_litem, left_on="o_orderkey", right_on="l_orderkey")
  .assign(revenue=lambda df: df["l_extendedprice"] * (1 - df["l_discount"]))
  .pipe(lambda df: df[rescols])
  .groupby(["l_orderkey", "o_orderdate", "o_shippriority"], as_index=False)
  .agg({"revenue": "sum"})[rescols]
  .sort_values(["revenue", "o_orderdate"], ascending=[False, True])
  .head(10)
)
```

Automatic Optimization

Introducing FireDucks

※IR: Intermediate Representation

FireDucks (**F**lexible **IR E**ngine for DataFrame) is a high-performance compiler-accelerated DataFrame library with highly compatible pandas APIs.




```
result = df.sort_values("A")
   .query("B > 1")["E"]
   .head(2)
```

```
%v2 = "sort_values_op"(%v1, "A")
%v3 = "filter_op"(%v2, "B > 1")
%v4 = "project_op"(%v3, ["E"])
%v5 = "slice_op"(%v4, 2)
```

print (result)

```
%t1 = "project_op"(%v1, ["A", "B", "E"])
%t2 = "filter_op"(%t1, "B > 1")
%t3 = "sort_values_op"(%t2, "A")
%t4 = "project_op"(%t3, ["E"])
%t5 = "slice_op"(%t4, 2)
```

```
df.loc[:, ["A", "B", "E"]]
   .query("B > 1")
   .sort_values("A")["E"]
   .head(2)
```

Primary Objective: <u>Write Once, Execute Anywhere</u>

Why FireDucks?

FireDucks (**F**lexible **IR** **E**ngine for DataFrame) is a high-performance compiler-accelerated DataFrame library with highly compatible pandas APIs.

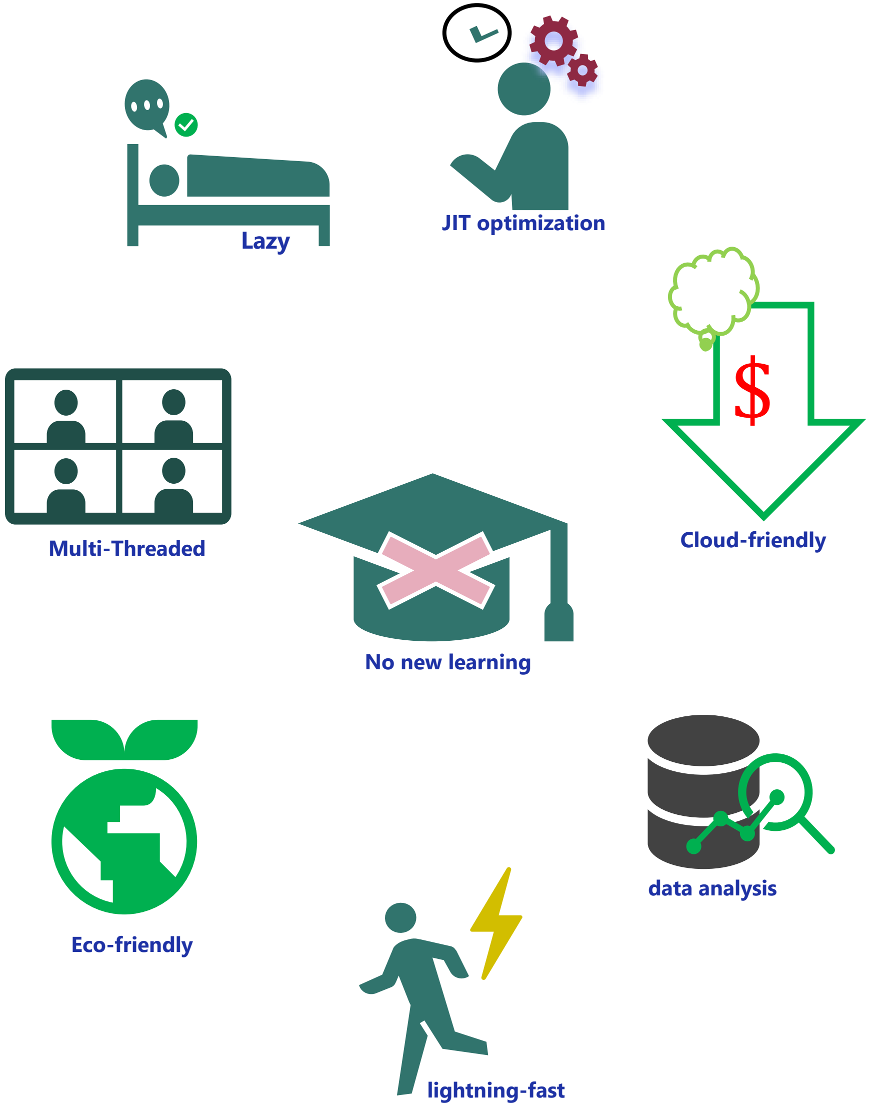
Speed: significantly faster than pandas

- FireDucks is multithreaded to fully exploit the modern processor
- Lazy execution model with Just-In-Time optimization using a defined-by-run mechanism supported by MLIR (a subproject of LLVM).
    - supports both lazy and non-lazy execution models without modifying user programs (same API).

MLIR

Ease of use: drop-in replacement of pandas

- FireDucks is highly compatible with pandas API
    - seamless integration is possible not only for an existing pandas program but also for any external libraries (like seaborn, scikit-learn, etc.) that internally use pandas dataframes.
- No extra learning is required
- No code modification is required



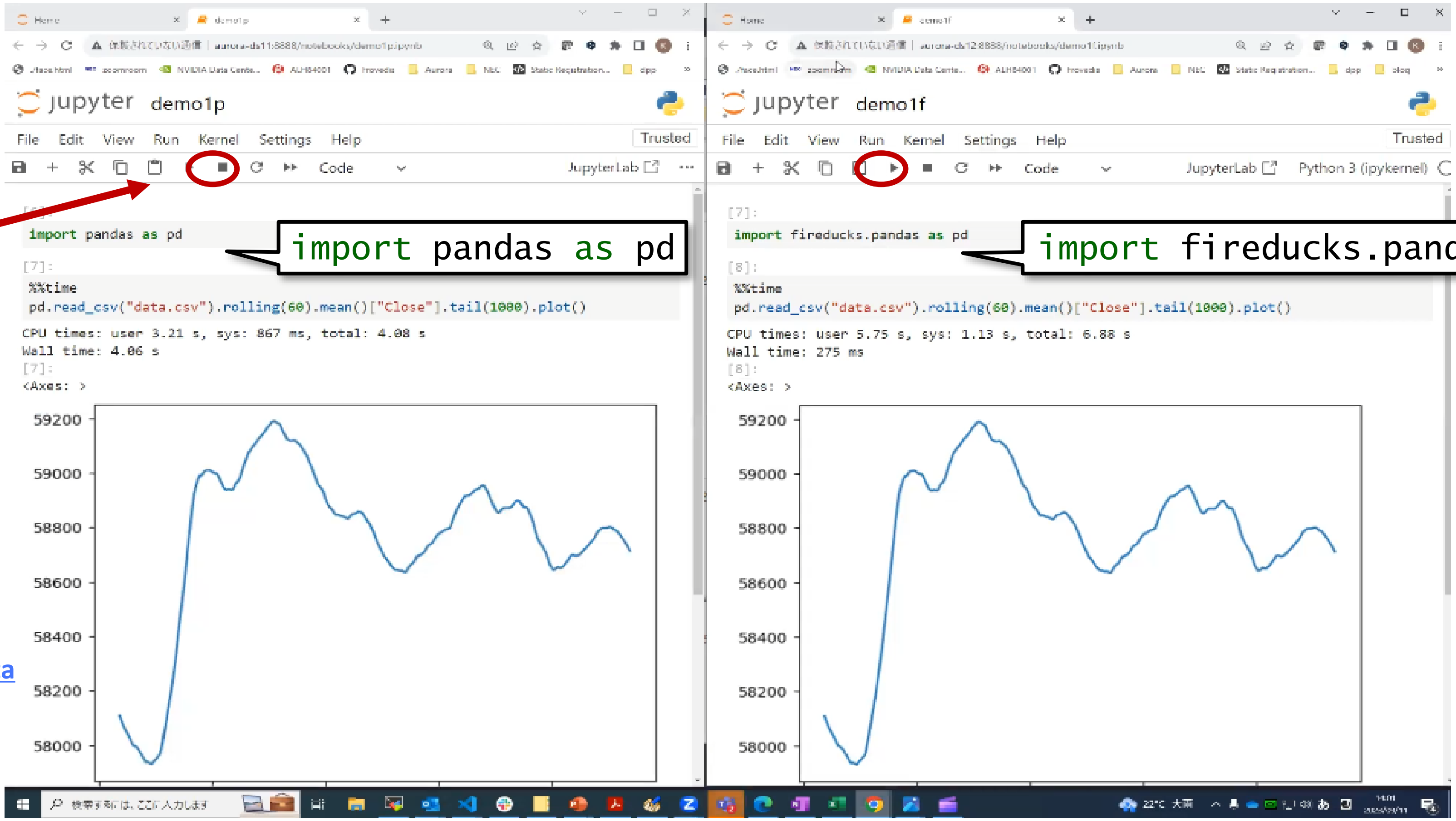
Let’s Have a Quick Demo!

```
pd.read_csv(“data.csv”).rolling(60).mean()[“Close”].tail(1000).plot()
```

Program to calculate moving average

pandas the difference is only in the import **FireDucks**

button to start execution



`import pandas as pd`

`import fireducks.pandas as pd`

pandas: 4.06s

~15x

FireDucks: 275ms

data.csv:
[Bitcoin Historical Data]

Usage of FireDucks

1. Explicit Import

easy to import

```
# import pandas as pd
import fireducks.pandas as pd
```

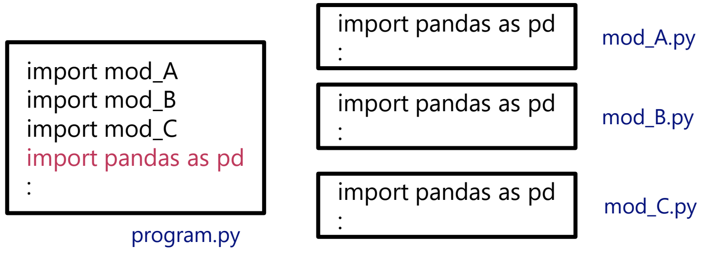
simply change the import statement

2. Import Hook

FireDucks provides command line option to automatically replace “**pandas**” with “**fireducks.pandas**”

```
$ python -m fireducks.pandas program.py
```

zero code modification



3. Notebook Extension

FireDucks provides simple import extension for interative notebooks.

```
%load_ext fireducks.pandas
import pandas as pd
```

simple integration in a notebook

Benchmark (1): DB-Benchmark

Database-like ops benchmark (https://duckdblabs.github.io/db-benchmark)

groupby | join

0.5 GB | 5 GB | 50 GB

basic questions

rank -1

Input table: 1,000,000,000 rows x 9 columns (50 GB)

FireDucks	0.12.5	2024-07-19	15s
DuckDB	1.0.0	2024-07-04	25s
ClickHouse	24.5.1.1763	2024-06-07	28s
Polars	1.1.0	2024-07-09	47s
Datafusion	38.0.1	2024-06-07	56s
data.table	1.15.99	2024-06-07	88s
DataFrames.jl	1.6.1	2024-06-07	91s
InMemoryDatasets	0.7.18	2023-10-17	218s
spark	3.5.1	2024-06-07	261s
R-arrow	16.1.0	2024-06-07	378s
collapse	2.0.14	2024-06-07	411s
(py)datatable	1.2.0a0	2024-06-07	1022s
dplyr	1.1.4	2024-06-07	1104s
pandas	2.2.2	2024-06-07	1126s
dask	2024.5.2	2024-06-07	out of memory
Modin		see README	pending

groupby

groupby | join

0.5 GB | 5 GB | 50 GB

basic questions

rank -1

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

FireDucks	0.12.5	2024-07-19	8s
DuckDB	1.0.0	2024-07-04	9s
Polars	1.1.0	2024-07-08	9s
Datafusion	38.0.1	2024-06-07	15s
InMemoryDatasets	0.7.18	2023-10-20	25s
ClickHouse	24.5.1.1763	2024-06-07	43s
data.table	1.15.99	2024-06-07	62s
collapse	2.0.14	2024-06-07	69s
DataFrames.jl	1.6.1	2024-06-07	77s
spark	3.5.1	2024-06-07	128s
dplyr	1.1.4	2024-06-07	214s
pandas	2.2.2	2024-06-07	244s
dask	2024.5.2	2024-06-07	635s
(py)datatable	1.2.0a0	2024-06-07	undefined exception
R-arrow	16.1.0	2024-06-07	out of memory
Modin		see README	pending

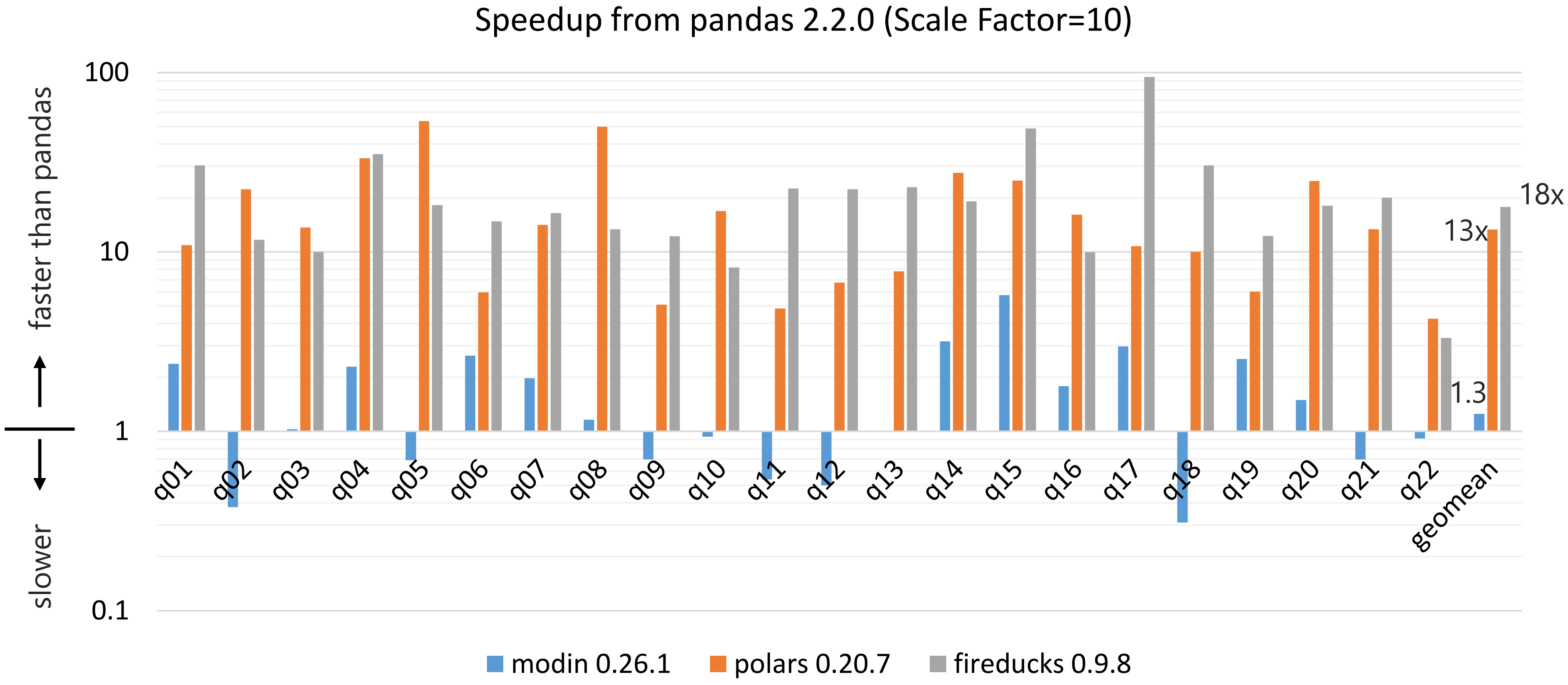
join

Benchmark (2): Speedup from pandas in TPC-H benchmark

FireDucks is 95x faster than pandas at max

Server

Xeon Gold 5317 x2 (24 cores), 256GB

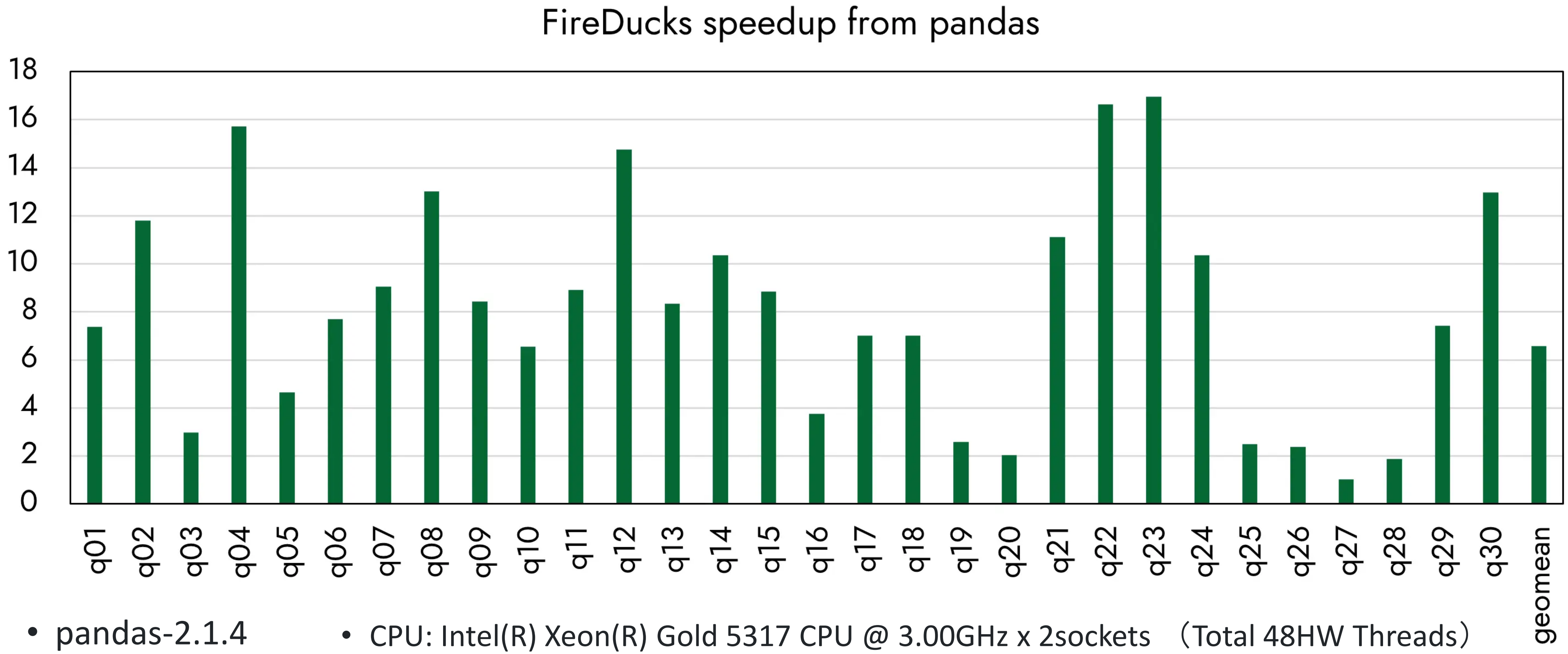


Comparison of DataFrame libraries (average speedup)

FireDucks	**18x**
Polars	13x
Modin	1.3x

Benchmark (3): Speedup from pandas in TPCx-BB benchmark

ETL(Extract, Transform, Load) and ML Workflow



- pandas-2.1.4
- fireducks-0.9.3

- CPU: Intel(R) Xeon(R) Gold 5317 CPU @ 3.00GHz x 2sockets （Total 48HW Threads）
- Main memory: 256GB

Resource on FireDucks

Web site (User guide, benchmark, blog)

https://fireducks-dev.github.io/

X(twitter) (Release information)

https://x.com/fireducksdev

Github (Issue report)

https://github.com/fireducks-dev/fireducks

slack Q/A, communication

https://join.slack.com/t/fireducks/shared_invite/zt-2j4lucmtj-IGR7AWlXO62Lu605pnBJ2w

FireDucks

Compiler Accelerated DataFrame Library for Python with fully-compatible pandas API

Get Started

```
import fireducks.pandas as pd
```

News

Release fileducks-0.12.4 (Jul 09, 2024)

Have you ever thought of speeding up your data analysis in pandas with a compiler?(blog) (Jul 03, 2024)

Evaluation result of Database-like ops benchmark with FireDucks is now available. (Jun 18, 2024)

Accelerate pandas without any manual code changes

Do you have a pandas-based program that is slow? FireDucks can speed-up your programs without any manual code changes. You can accelerate your data analysis without worrying about slow performance due to single-threaded execution in pandas.

Let’s go for a test drive!

https://colab.research.google.com/drive/1qpej-X7CZsIeOqKuhBg4kq-cbGuJf1Zp?usp=sharing

Thank You!

- Focus more on in-depth data exploration using "**pandas**".

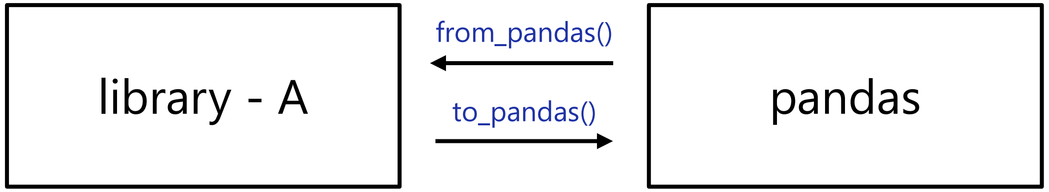
- Let the "**FireDucks**" take care of the optimization for you.

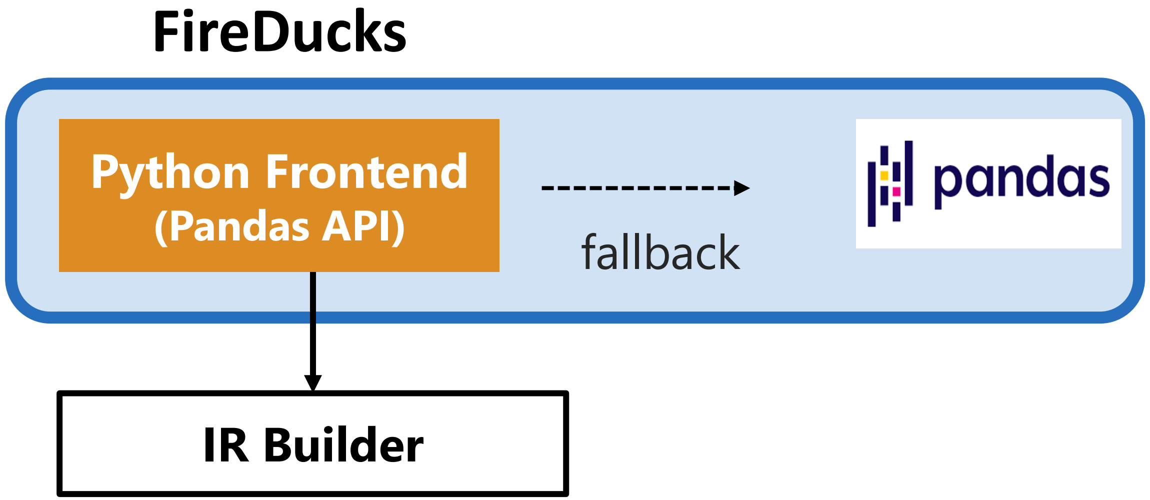
- Enjoy Green Computing!

https://www.linkedin.com/in/sourav-%E3%82%BD%E3%82%A6%E3%83%A9%E3%83%96-saha-%E3%82%B5%E3%83%8F-a5750259/
https://twitter.com/SouravSaha97589

Frequently Asked Questions

FAQ: Why FireDucks is highly compatible with pandas?






```
%load_ext fireducks.pandas      ← notebook extension for importhook
import pandas as pd
import numpy as np
```

```
%%fireducks.profile             ← notebook specific profiler
df = pd.DataFrame({
    "id": np.random.choice(list("abcdef"), 10000),
    "val": np.random.choice(100, 10000)
})

r1 =(
    df.sort_values("id")
      .groupby("id")
      .head(2)
      .reset_index(drop=True)
)
                pd.from_pandas(r1["val"].to_pandas().cumsum())
r1["val"] = r1["val"].cumsum()
r1.describe()
```

profiling-summary:: total: 42.4832 msec (fallback: 1.1448 msec)

	name	type	n_calls	duration (msec)
0	groupby_head	kernel	1	16.696805
1	sort_values	kernel	1	16.684564
2	from_pandas.frame.metadata	kernel	2	3.641694
3	to_pandas.frame.metadata	kernel	2	2.237987
4	describe	kernel	1	2.021135
5	DataFrame._repr_html_	fallback	1	1.021662
6	Series.cumsum	fallback	1	0.111802
7	setitem	kernel	1	0.010280
8	get_metadata	kernel	1	0.009650
9	reset_index	kernel	1	0.008050

When running a python script/program, you may like to set the environment variable to get fallback warning logs:
FIREDUCKS_FLAGS="-Wfallback"

[Raise](#) feature request when you encounter some expensive fallback hindering your program performance!

Directly [communicate](#) with us over our slack channel for any performance or API related queries!

FAQ: How to evaluate Lazy Execution?

```
def foo(employee, country):
    stime = time.time()
    m = employee.merge(country, on="C_Code")
    r = m[m["Gender"] == "Male"]
    print(f"fireducks time: {time.time() – stime} sec")
    return r
```

IR Builder

create_data_op(…)
merge_op(…)
filter_op(…)

fireducks time: 0.0000123 sec

```
def foo(employee, country):
    employee._evaluate()
    country._evaluate()
    stime = time.time()
    m = employee.merge(country, on="C_Code")
    r = m[m["Gender"] == "Male"]
    r._evaluate()
    print(f"fireducks time: {time.time() – stime} sec")
    return r
```

fireducks time: 0.02372143 sec

FIREDUCKS_FLAGS="--benchmark-mode"

Use this to disable lazy-execution mode when you do not want to make any changes in your existing application during performance evaluation.

FAQ: How to configure number of cores to be used?

OMP_NUM_THREADS=1

Use this to stop parallel execution, or configure this with the intended number of cores to be used

Alternatively, you can use the Linux taskset command to bind your program with specific CPU cores.

\Orchestrating a brighter world

NEC creates the social values of safety, security,
fairness and efficiency to promote a more sustainable world
where everyone has the chance to reach their full potential.

\Orchestrating a brighter world

NEC