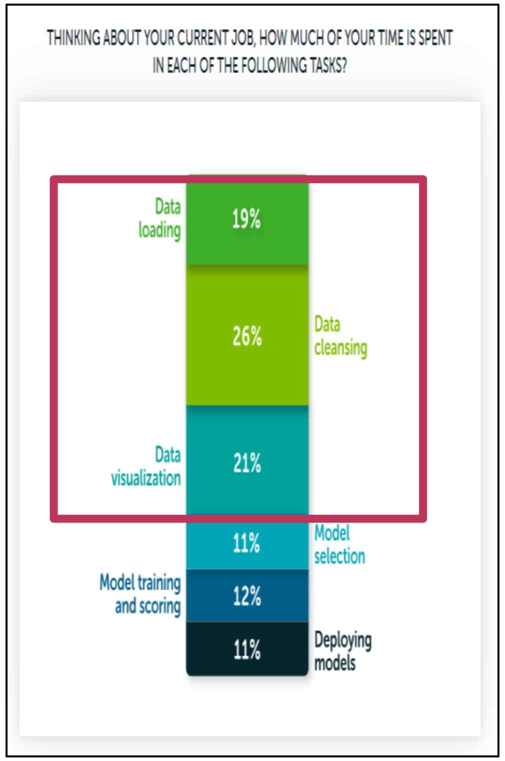
Introducing FireDucks: A must have DataFrame library to speedup your Pandas workload at zero manual cost

Sep 26, 2024, Thursday

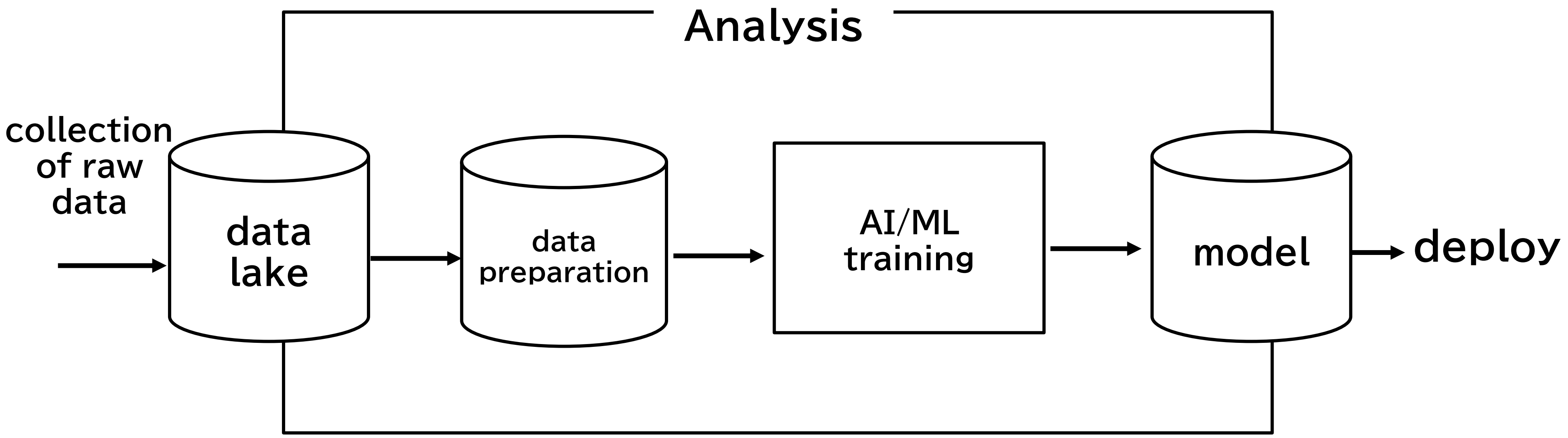
Sourav Saha (NEC)

Workflow of a Data Scientist

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

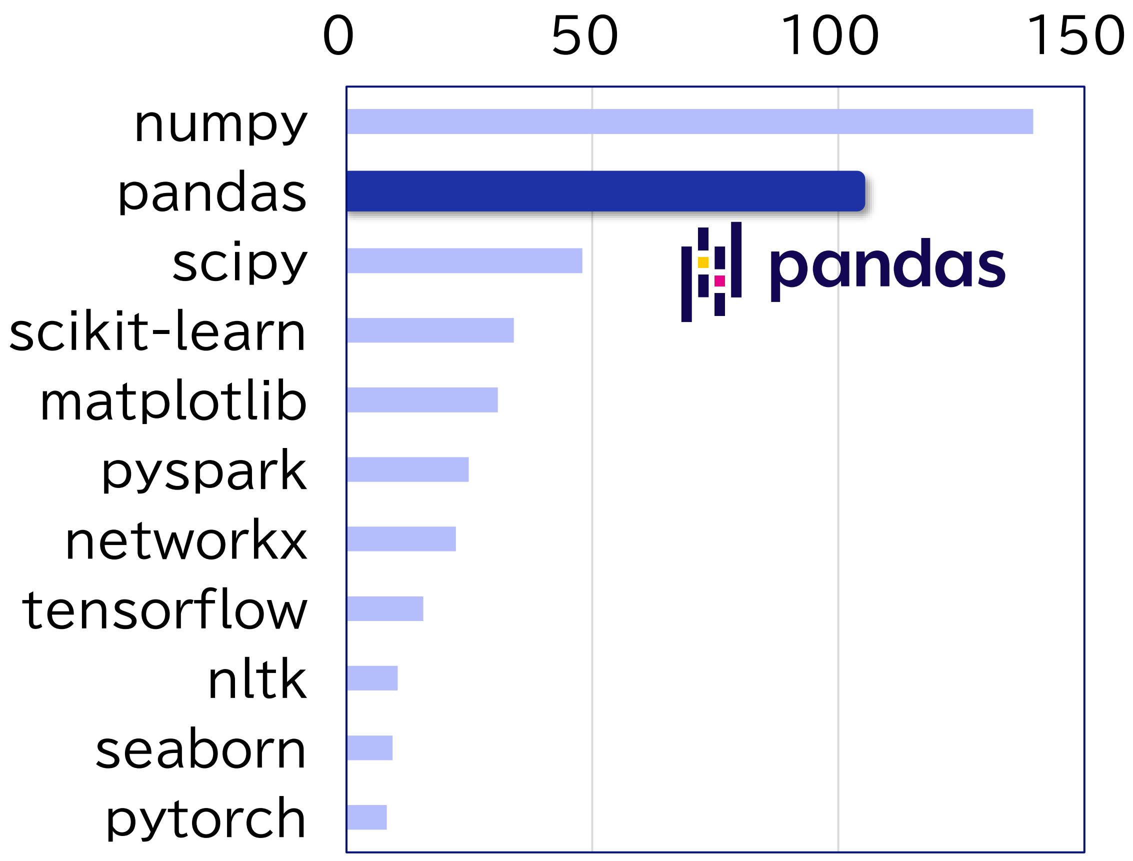


Anaconda:
The State of Data Science 2020



About Pandas

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



Monthly download from pypi.org
(Data Analytics Libraries)

- Most of its operations are single-threaded.
- The way of defining a query in pandas heavily impacts its performance!!

- Some of the high-performance pandas alternatives compel a user to learn completely new APIs
- Some of the others demand for paying extra hardware cost.

- We at NEC R&D Lab Japan, have developed a high-performance compiler-accelerated DataFrame library, named **FireDucks** with highly compatible pandas APIs to address the above issues.

Introducing FireDucks

※IR: Intermediate Representation

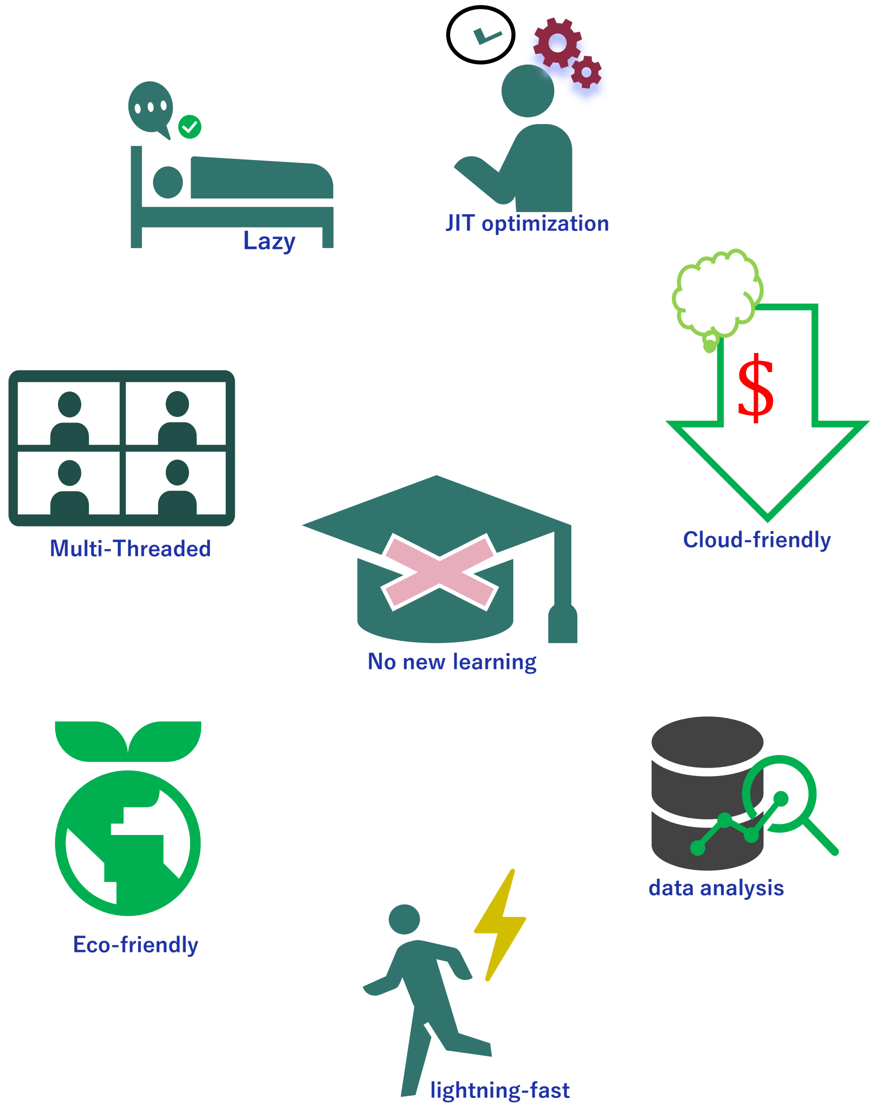
FireDucks (**F**lexible **IR** **E**ngine for DataFrame) is a high-performance compiler-accelerated DataFrame library with highly compatible pandas APIs.

Speed: significantly faster than pandas

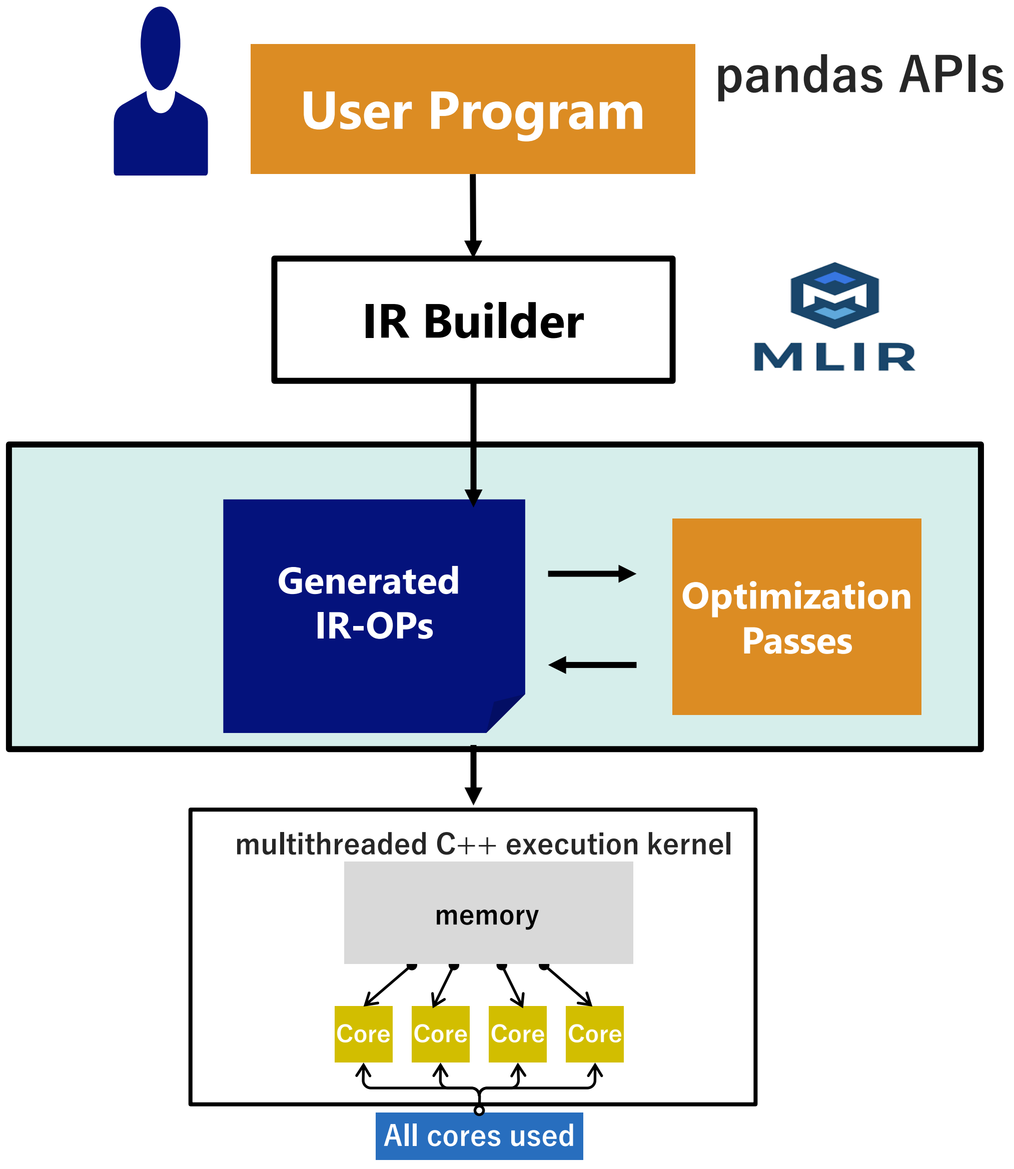
- FireDucks is multithreaded to fully exploit the modern processor
- Lazy execution model with Just-In-Time optimization using a defined-by-run mechanism supported by MLIR (a subproject of LLVM).
  - supports both lazy and non-lazy execution models without modifying user programs (same API).

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



How does it work?



```
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)
```

```
t1 = backend::project_columns(df, {"A", "B", "C"});
t2 = backend::filter_rows(t1, "B > 1");
t3 = backend::sort_values(t2, "A");
t4 = backend::project_columns(t3, {"E"});
return backend::slice_rows(t4, 2);
```

Primary Objective: Write Once, Execute Anywhere

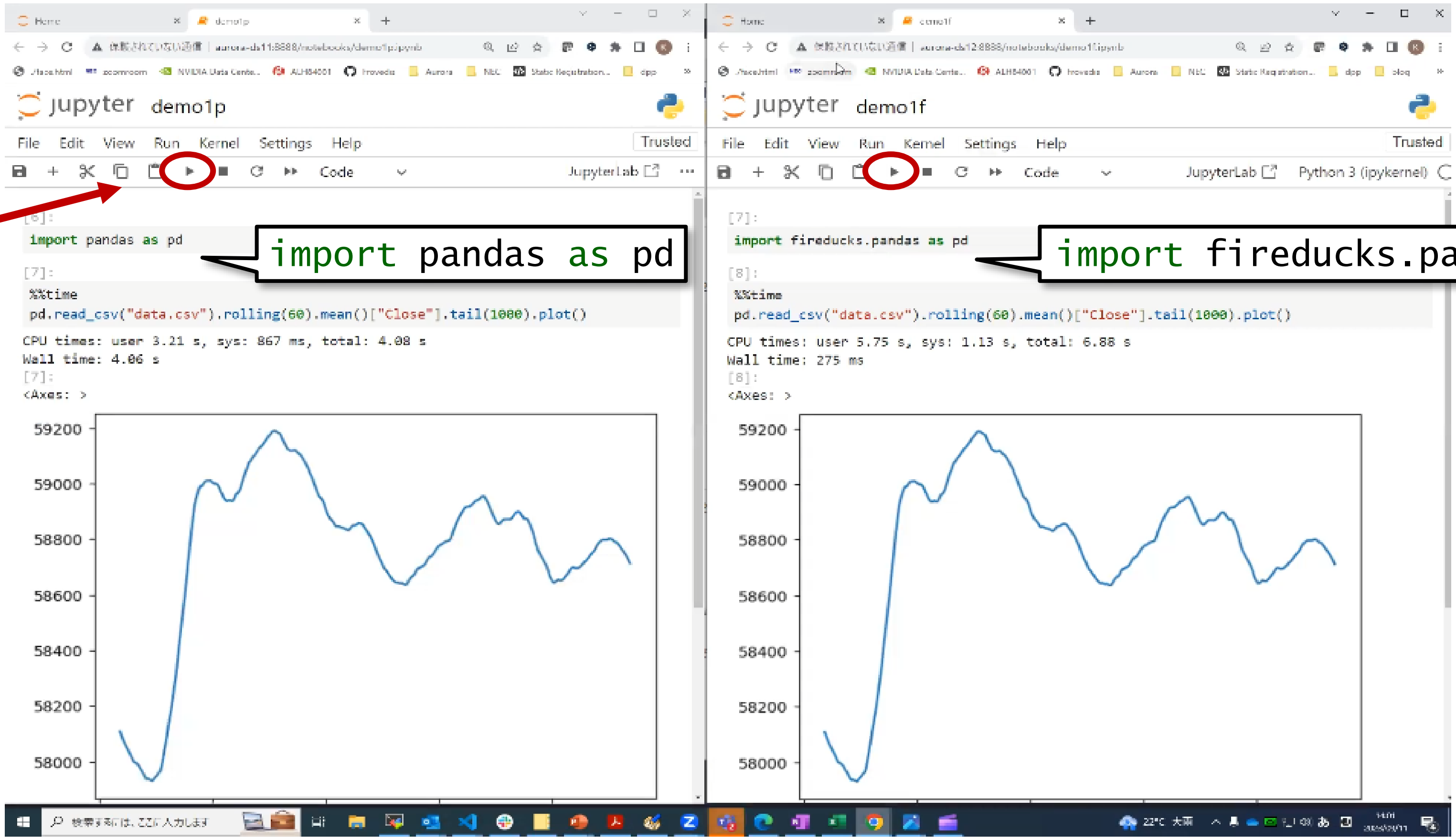
Let’s Have a Quick Demo!

```
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



```
[6]:
import pandas as pd
[7]:
%%time
pd.read_csv("data.csv").rolling(60).mean()["Close"].tail(1000).plot()
CPU times: user 3.21 s, sys: 867 ms, total: 4.08 s
Wall time: 4.06 s
[7]:
<Axes: >
```

```
[7]:
import fireducks.pandas as pd
[8]:
%%time
pd.read_csv("data.csv").rolling(60).mean()["Close"].tail(1000).plot()
CPU times: user 5.75 s, sys: 1.13 s, total: 6.88 s
Wall time: 275 ms
[8]:
<Axes: >
```

```
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
```

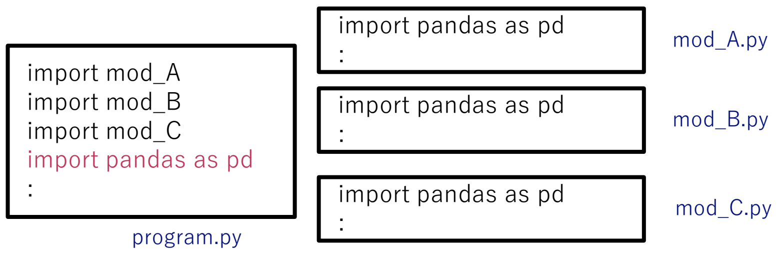
simply change the import statement

2. Import Hook (monkey-patch)

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

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

zero code modification



program.py
```
import mod_A
import mod_B
import mod_C
import pandas as pd
:
```

mod_A.py
```
import pandas as pd
:
```

mod_B.py
```
import pandas as pd
:
```

mod_C.py
```
import pandas as pd
:
```

3. Notebook Extension

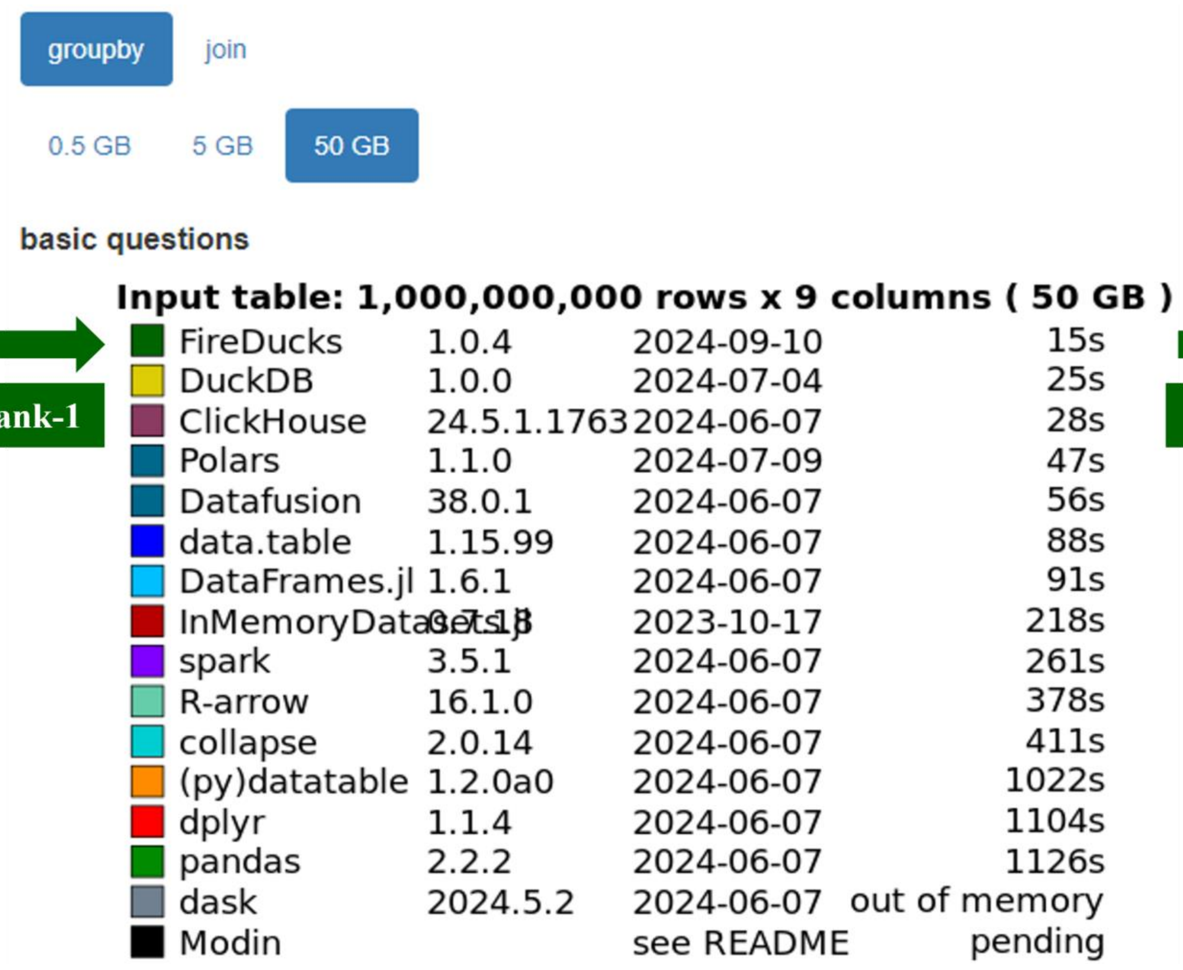
FireDucks provides simple import extension for interative notebooks.

```
%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)

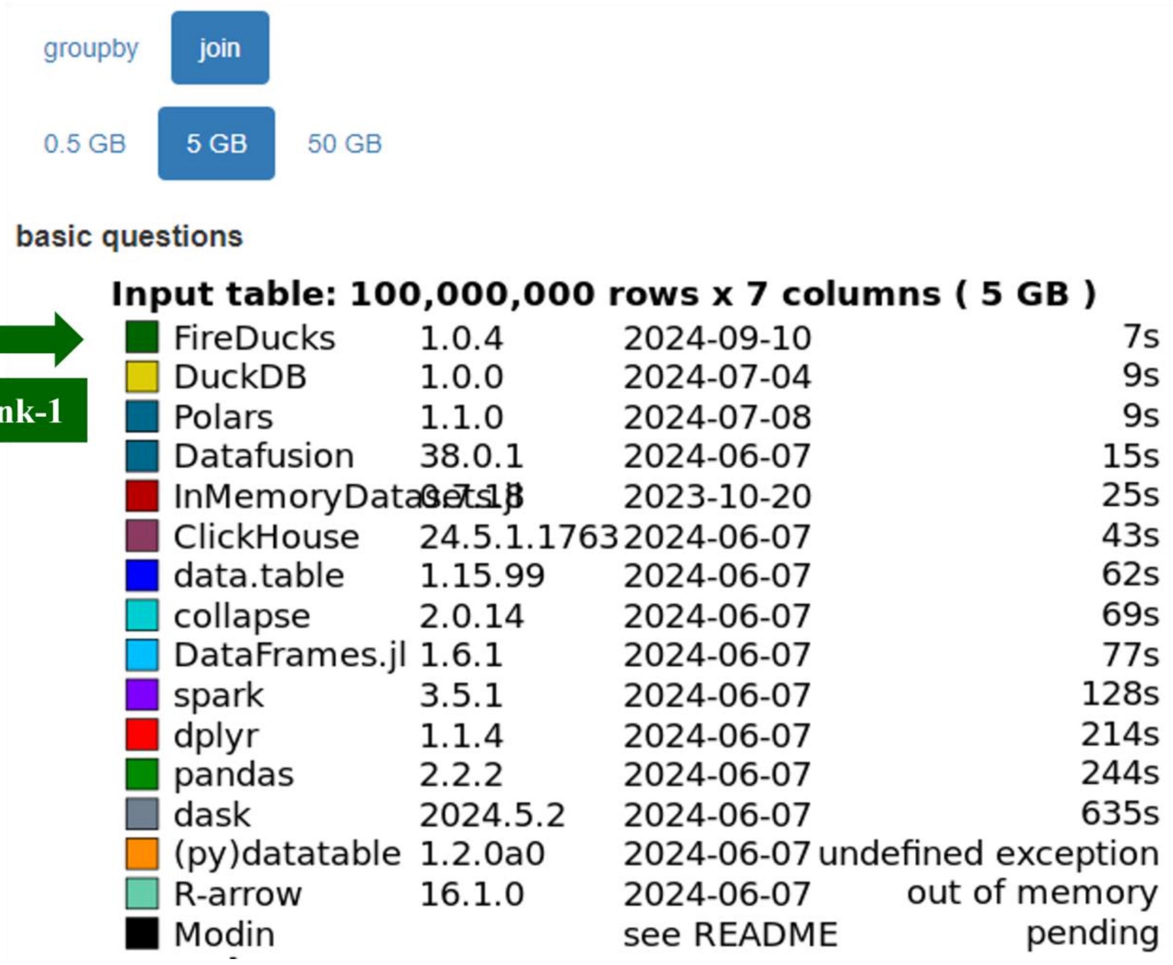


basic questions

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

rank-1

🟩	FireDucks	1.0.4	2024-09-10	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



basic questions

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

rank-1

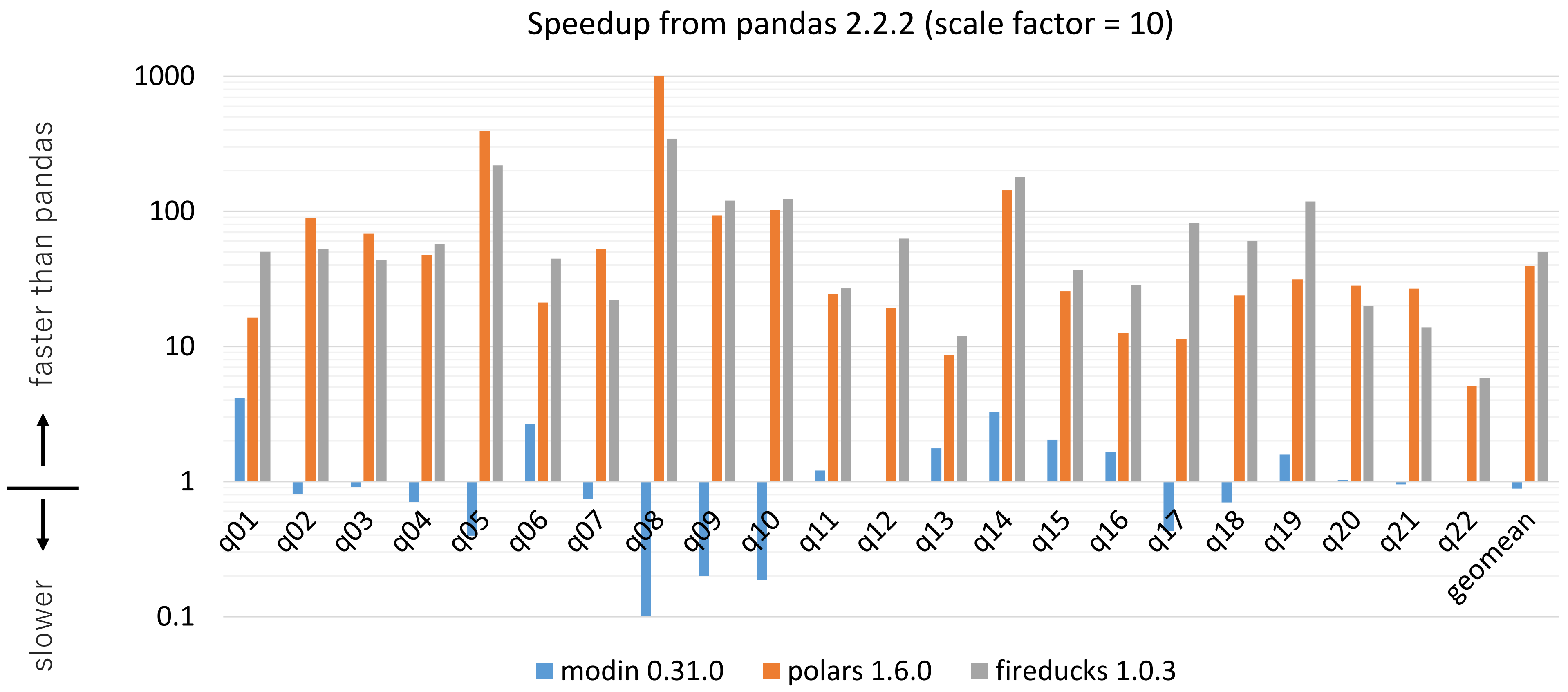
🟩	FireDucks	1.0.4	2024-09-10	7s
🟨	DuckDB	1.0.0	2024-07-04	9s
🟦	Polars	1.1.0	2024-07-08	9s
🟦	Datafusion	38.0.1	2024-06-07	15s
🟥	InMemoryDataSets	0.7.18	2023-10-20	25s
🟪	ClickHouse	24.5.1.1763	2024-06-07	43s
🟦	data.table	1.15.99	2024-06-07	62s
🟦	collapse	2.0.14	2024-06-07	69s
🟦	DataFrames.jl	1.6.1	2024-06-07	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

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

FireDucks is ~345x faster than pandas at max

Server

Xeon Gold 5317 x2
(24 cores), 256GB



Comparison of DataFrame libraries (average speedup)

FireDucks	**50x**
Polars	39x
Modin	0.9x

Resource on FireDucks

Web site (User guide, benchmark, blog)

https://fireducks-dev.github.io

X(twitter) (Release information)

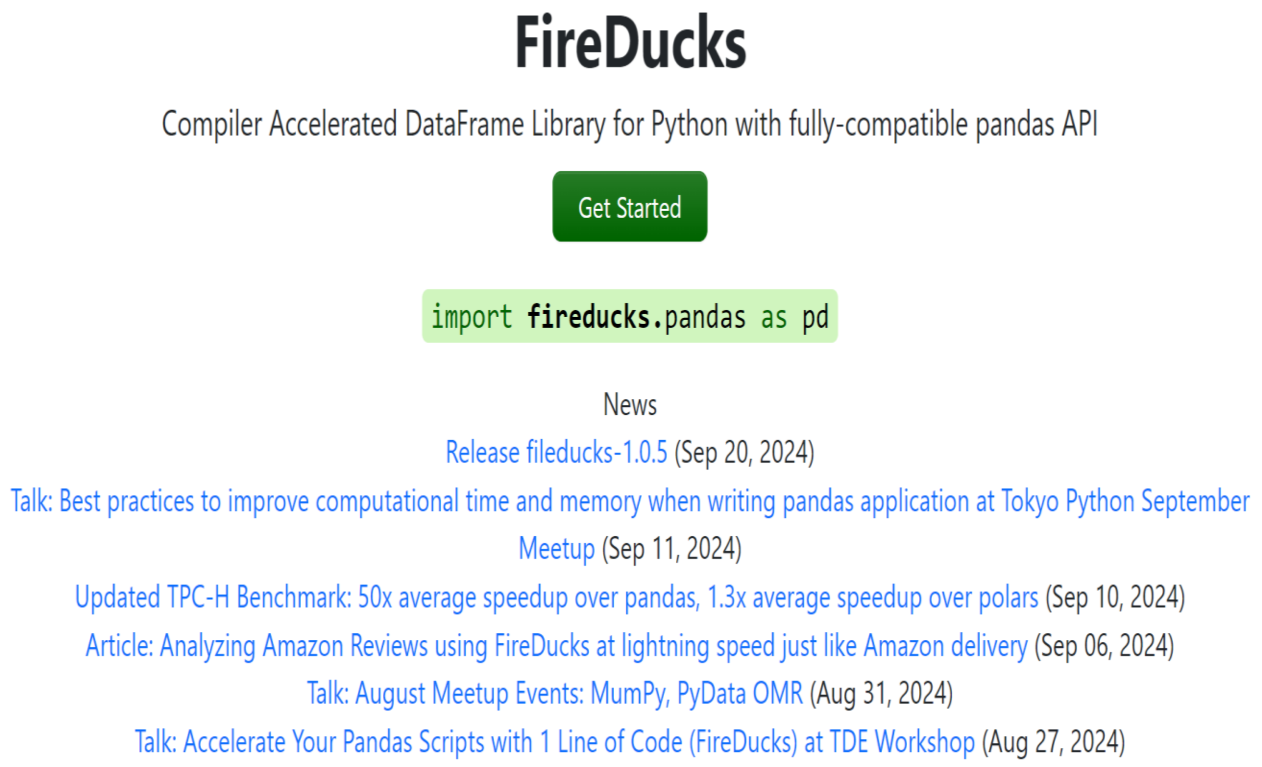
https://x.com/fireducksdev
(@fireducksdev)

Github (Issue report)

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

slack **Q/A, communication**

https://join.slack.com/t/fireducks/shared_invite/zt-2j4lucmtj-IGR7AWIXO62Lu605pnBJ2w



FireDucks

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

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

News

Release fileducks-1.0.5 (Sep 20, 2024)

Talk: Best practices to improve computational time and memory when writing pandas application at Tokyo Python September Meetup (Sep 11, 2024)

Updated TPC-H Benchmark: 50x average speedup over pandas, 1.3x average speedup over polars (Sep 10, 2024)

Article: Analyzing Amazon Reviews using FireDucks at lightning speed just like Amazon delivery (Sep 06, 2024)

Talk: August Meetup Events: MumPy, PyData OMR (Aug 31, 2024)

Talk: Accelerate Your Pandas Scripts with 1 Line of Code (FireDucks) at TDE Workshop (Aug 27, 2024)

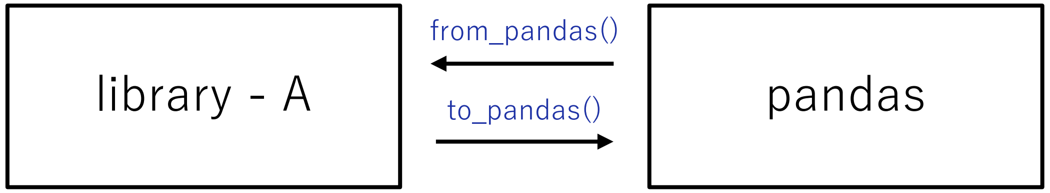
Thank You!

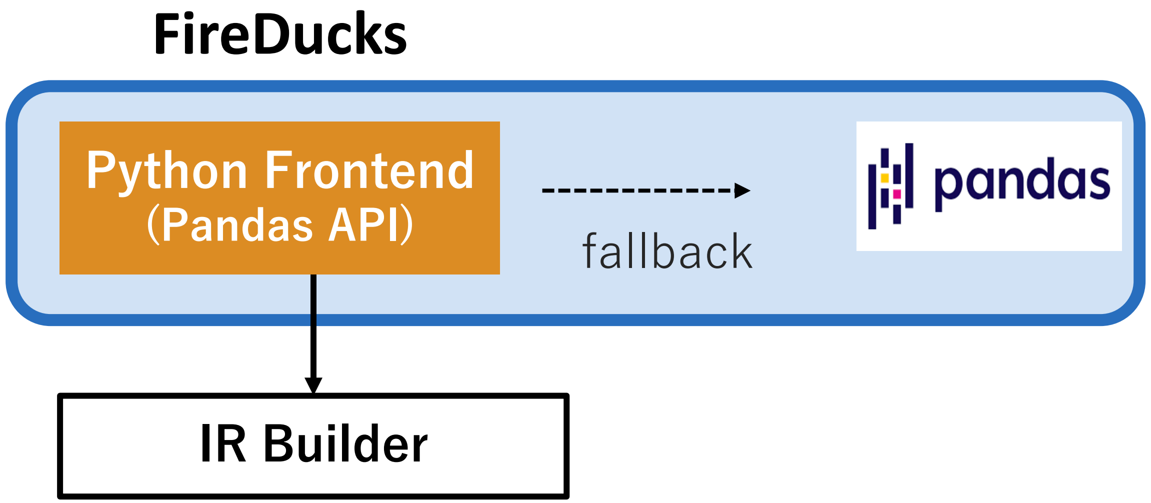
- Focus more on in-depth data exploration using **“Pandas”**.
- Let the **“FireDucks”** take care of the optimization for you.
- Enjoy Green Computing!

We would love to see you at our booth for any queries related to FireDucks.

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
```

fireducks time: 0.0000123 sec

⇨

IR Builder

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

```
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は、安全・安心・公平・効率という社会価値を創造し、

誰もが人間性を十分に発揮できる持続可能な社会の実現を目指します。

\Orchestrating a brighter world

NEC