\Orchestrating a brighter world

Best practices while processing large-scale data using Pandas-like libraries

Sep 28, 2024

Sourav Saha (NEC)

Quick check on basic pandas operations (1/5)

- **How to get top-2 rows based on the column “A” from table “df”?**

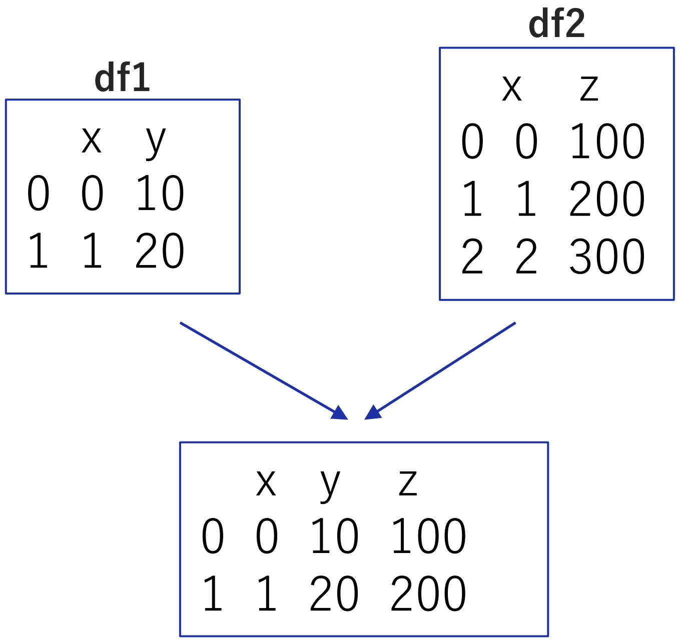
```
   A   B
0  2  10
1  5  30
2  1  20
3  3  70
4  7  60
5  8  40
6  4  80
```

```
   A   B
5  8  40
4  7  60
```

```
df.sort_values(“A”, ascending=False).head(2)
```

Quick check on basic pandas operations (2/5)

- **How to perform inner-join of table “df1” with table “df2” on common key-column “x”?**



df1

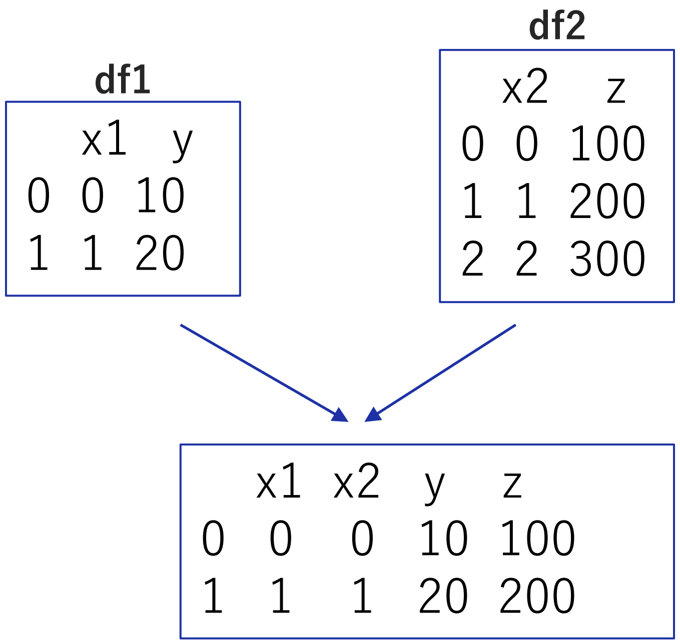
```
   x  y
0  0  10
1  1  20
```

df2

```
   x   z
0  0  100
1  1  200
2  2  300
```

```
   x  y   z
0  0  10  100
1  1  20  200
```

df1.merge(df2, on=“x”, how=“inner”)



df1

```
   x1  y
0  0  10
1  1  20
```

df2

```
   x2   z
0  0  100
1  1  200
2  2  300
```

```
   x1  x2  y   z
0  0    0  10  100
1  1    1  20  200
```

df1.merge(df2, left_on=“x1”, right_on=“x2”, how=“inner”)

Quick check on basic pandas operations (3/4)

- **How to perform Sum of “B” column based on different group of “A” column?**

```
   A   B
0  1  10
1  2  20
2  1  30
3  2  40
4  3  50
5  3  60
6  1  70
```

```
     B
A
1  110
2   60
3  110
```

df.groupby(“A”).agg(“sum”)

df.groupby(“A”)[“B”].agg(“sum”)

df.groupby(“A”).agg({“B”: “sum”})

df.groupby(“A”).agg(**b_sum** = (“B”, “sum”))

```
   b_sum
A
1    110
2     60
3    110
```

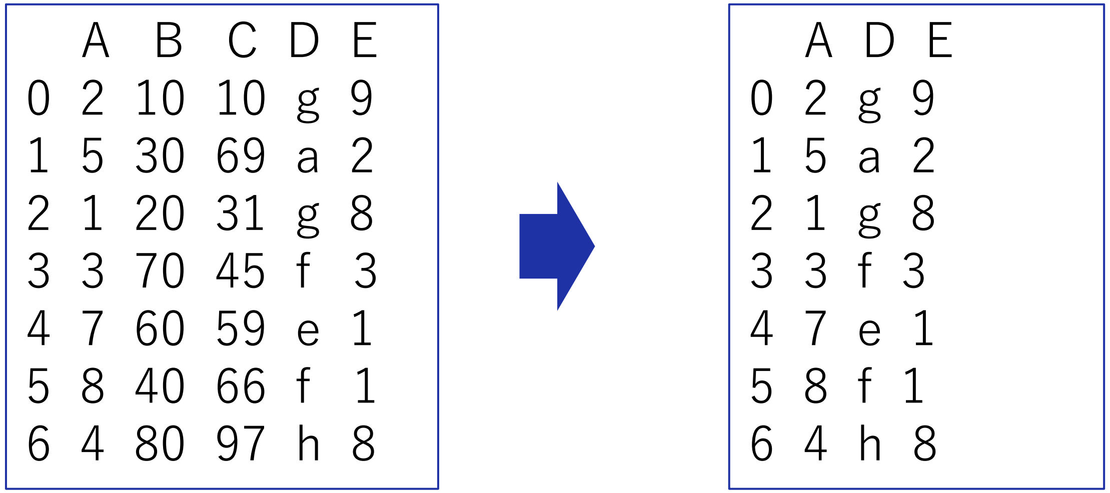
```
   A   B
0  1  10
1  2  20
2  1  30
3  2  40
4  3  50
5  3  60
6  1  70
```

```
   A    B
0  1  110
1  2   60
2  3  110
```

df.groupby(“A”, as_index=False).agg(“sum”)

Quick check on basic pandas operations (4/4)

- **How to select intended columns, e.g., “A”, “D” and “E” from table “df”?**



```
   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
```

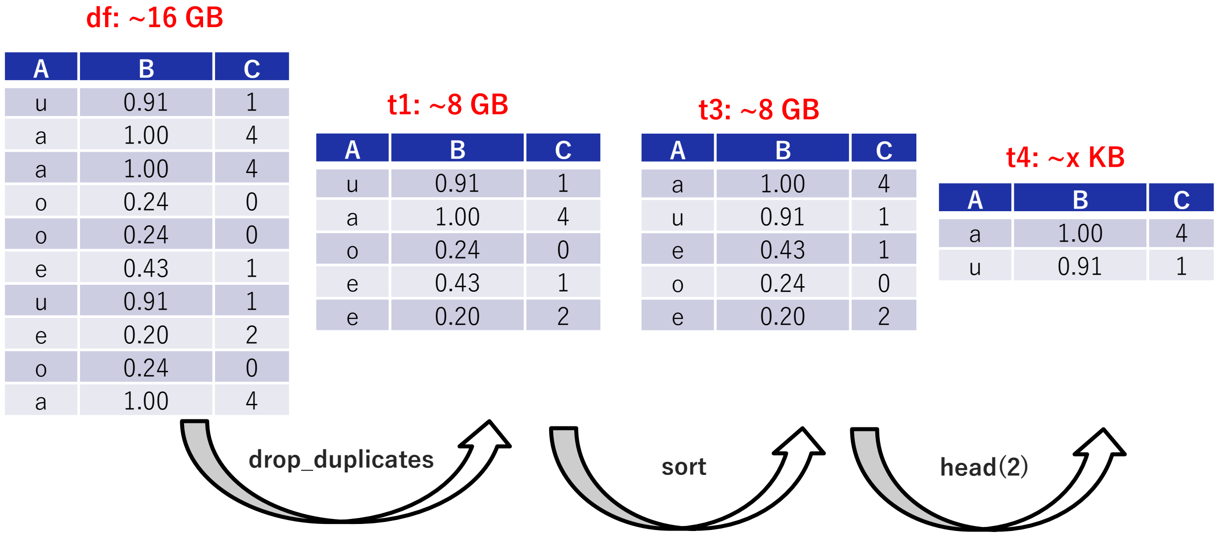
```
   A  D  E
0  2  g  9
1  5  a  2
2  1  g  8
3  3  f  3
4  7  e  1
5  8  f  1
6  4  h  8
```

```
df[[“A”, “D”, “E”]]

df.loc[:, [“A”, “D”, “E”]]

df.iloc[:, [0, 3, 4]]
```

Performance Challenges & Best Practices to follow

(1) importance of chained expression

```
def foo(filename):
  df = pd.read_csv(filename)
  t1 = df.drop_duplicates()
  t2 = t1.sort_values(“B”)
  t3 = t2.head(2)
  return t3
```



df: ~16 GB

A	B	C
u	0.91	1
a	1.00	4
a	1.00	4
o	0.24	0
o	0.24	0
e	0.43	1
u	0.91	1
e	0.20	2
o	0.24	0
a	1.00	4

t1: ~8 GB

A	B	C
u	0.91	1
a	1.00	4
o	0.24	0
e	0.43	1
e	0.20	2

t3: ~8 GB

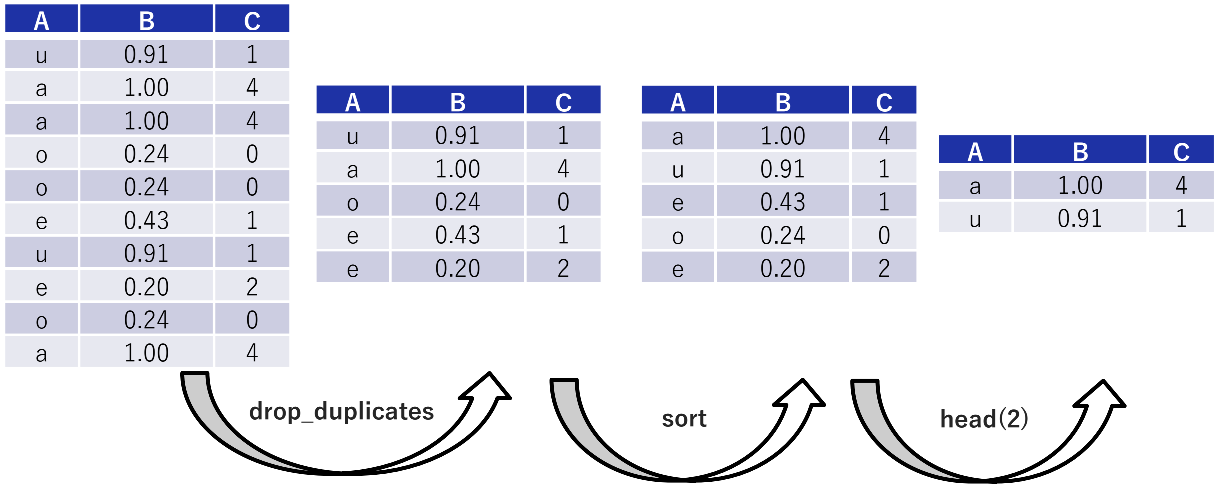
A	B	C
a	1.00	4
u	0.91	1
e	0.43	1
o	0.24	0
e	0.20	2

t4: ~x KB

A	B	C
a	1.00	4
u	0.91	1

re-write using chained expression

```
def foo(filename):
  return (
    pd.read_csv(filename)
      .drop_duplicates()
      .sort_values(“B”)
      .head(2)
  )
```



A	B	C
u	0.91	1
a	1.00	4
a	1.00	4
o	0.24	0
o	0.24	0
e	0.43	1
u	0.91	1
e	0.20	2
o	0.24	0
a	1.00	4

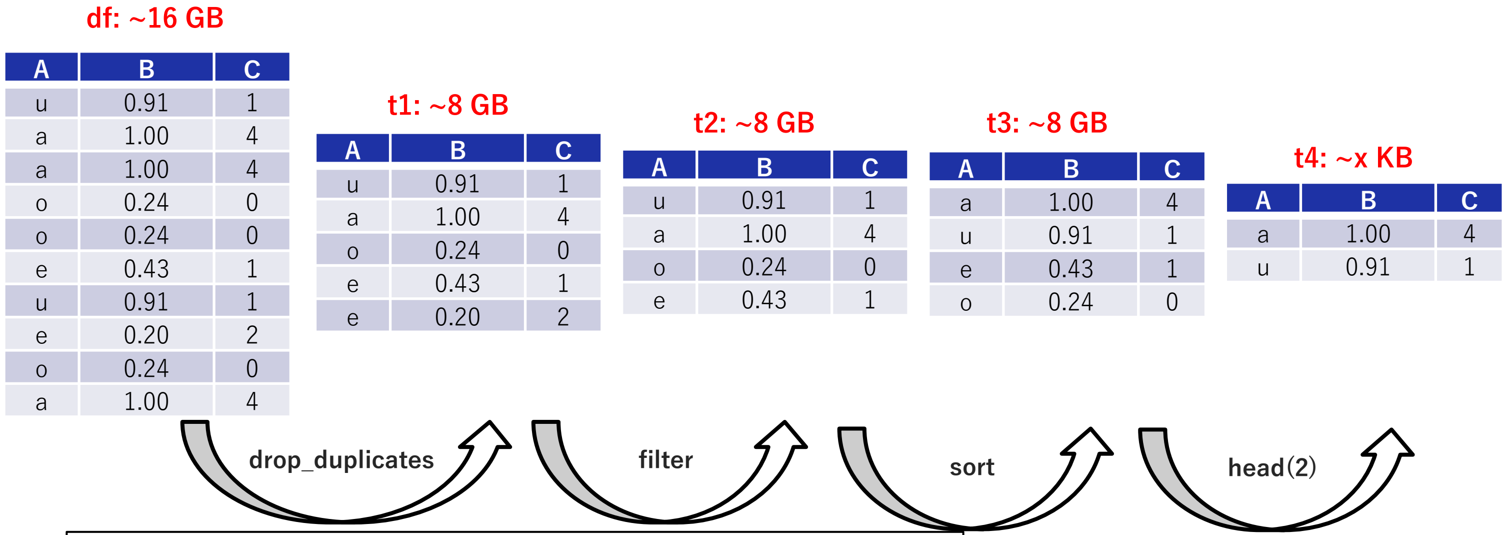
A	B	C
u	0.91	1
a	1.00	4
o	0.24	0
e	0.43	1
e	0.20	2

A	B	C
a	1.00	4
u	0.91	1
e	0.43	1
o	0.24	0
e	0.20	2

A	B	C
a	1.00	4
u	0.91	1

Use pipe() or query() for filter operation

```
def foo(filename):
  df = pd.read_csv(filename)
  t1 = df.drop_duplicates()
  t2 = t1[t1[“B”] > 0.20]
  t3 = t2.sort_values(“B”)
  t4 = t3.head(2)
  return t4
```



df: ~16 GB

A	B	C
u	0.91	1
a	1.00	4
a	1.00	4
o	0.24	0
o	0.24	0
e	0.43	1
u	0.91	1
e	0.20	2
o	0.24	0
a	1.00	4

t1: ~8 GB

A	B	C
u	0.91	1
a	1.00	4
o	0.24	0
e	0.43	1
e	0.20	2

t2: ~8 GB

A	B	C
u	0.91	1
a	1.00	4
o	0.24	0
e	0.43	1

t3: ~8 GB

A	B	C
a	1.00	4
u	0.91	1
e	0.43	1
o	0.24	0

t4: ~x KB

A	B	C
a	1.00	4
u	0.91	1

re-write using chained expression

```
def foo(filename):
  return (
    pd.read_csv(filename)
      .drop_duplicates()
      .??
      .sort_values(“B”)
      .head(2)
  )
```

```
def foo(filename):
  return (
    pd.read_csv(filename)
      .drop_duplicates()
      .query(“B > 0.20”)
      .sort_values(“B”)
      .head(2)
  )
```

query(): allows you to write SQL-like conditional expression, helping you to perform filter on the current state of the input frame, but its a little slower as it parses the input string to construct the filter mask.

```
def foo(filename):
  return (
    pd.read_csv(filename)
      .drop_duplicates()
      .pipe(lambda tmp: tmp[tmp[“B”] > 0.20]
      .sort_values(“B”)
      .head(2)
  )
```

pipe(): a convenient method allowing you to perform a given operation (like filter etc.) on the current state of the input frame without introducing computational overhead.

Use assign() for setting a new column

```
df = pd.read_csv(filename)
       .drop_duplicates()

df[“C”] = df[“A”] + df[“B”]
```

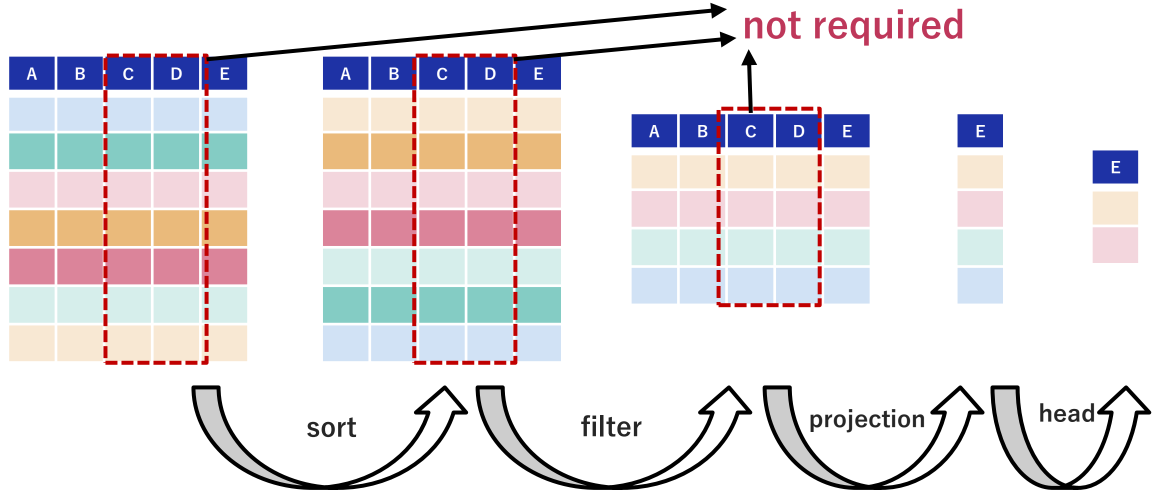
re-write using chained expression

```
df = pd.read_csv(filename)
       .drop_duplicates()
       .assign(C=lambda tmp: tmp[“A”] + tmp[“B”])
```

(2) importance of execution order

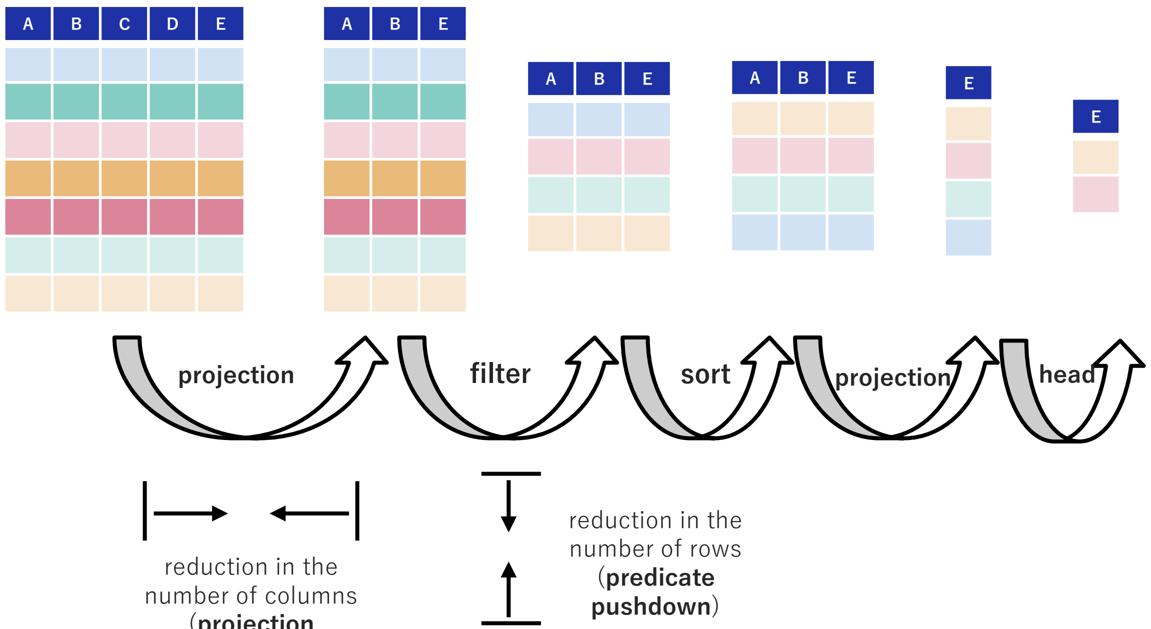
```
df.sort_values("A")
  .query("B > 1")["E"]
  .head(2)
```

※ 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)
```



reduction in the number of columns (**projection pushdown**)

reduction in the number of rows (**predicate pushdown**)

OPTIMIZED QUERY

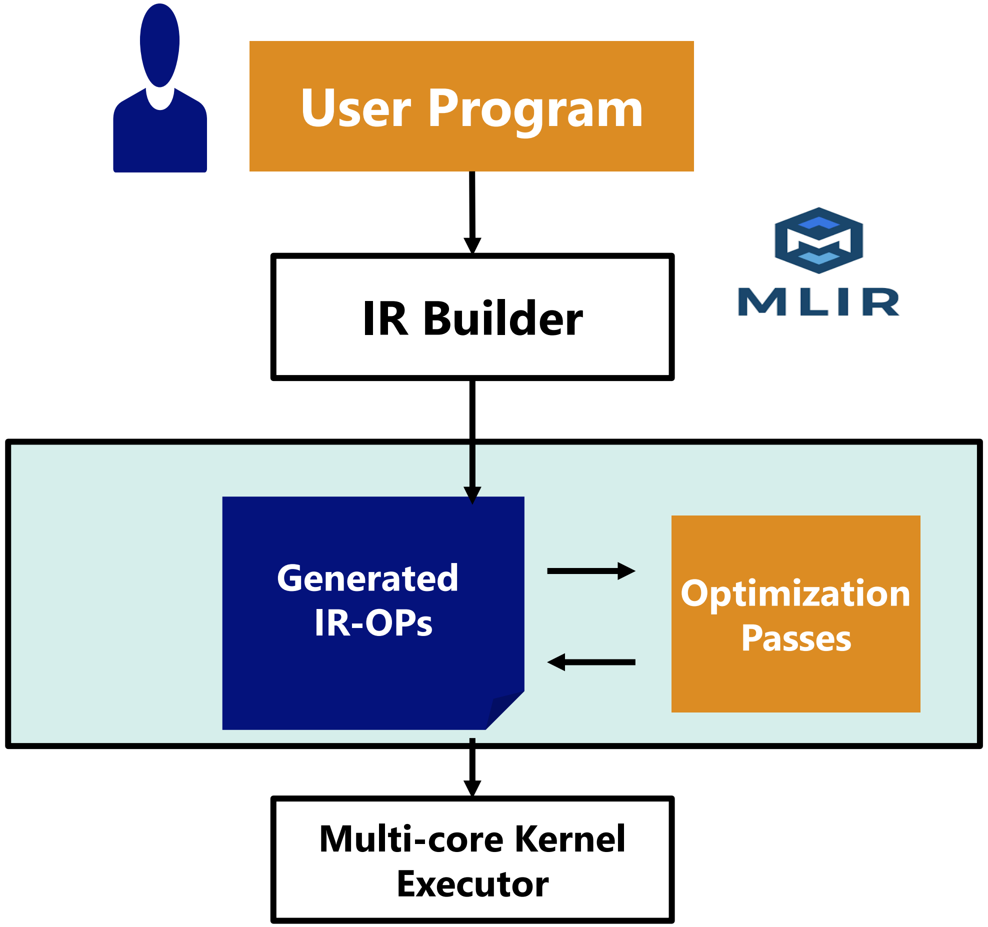
Let’s put our learning to exercise

1. Join “customer” and “orders” tables, where "c_custkey“=="o_custkey“
2. Join result with “lineitem”, where "o_orderkey“=="l_orderkey“
3. Filter result, where "c_mktsegment" == "BUILDING“
4. Filter result, where "o_orderdate" < 1995-03-15
5. Filter result, where "l_shipdate" > 1995-03-1
6. Add a new column, named “revenue” as: "l_extendedprice" * (1 - "l_discount")
7. Perform Groupby on: ["l_orderkey", "o_orderdate", "o_shippriority"]
8. Perform Aggregation to compute group-wise sum of “revenue” column.
9. Project columns as: ["l_orderkey", "revenue", "o_orderdate", "o_shippriority"]
10. Sort results by "revenue“ as descending order and "o_orderdate“ as ascending order.
11. Get top-10 from result

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)
```

```
result = df.loc[:, [“A”, “B”, “E”]]
   .query(“B > 1”)
   .sort_values(“A”)[“E”]
   .head(2)
```

Primary Objective: Write Once, Execute Anywhere

Usage of FireDucks

1. Explicit Import

easy to import

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

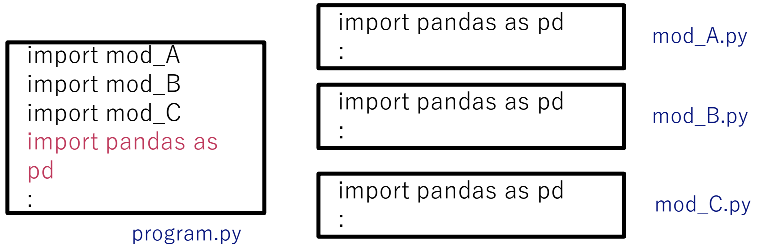
simply change the import statement

2. Import Hook

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

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

zero code modification



```
import mod_A
import mod_B
import mod_C
import pandas as pd
:
```

program.py

```
import pandas as pd
:
```

mod_A.py

```
import pandas as pd
:
```

mod_B.py

```
import pandas as pd
:
```

mod_C.py

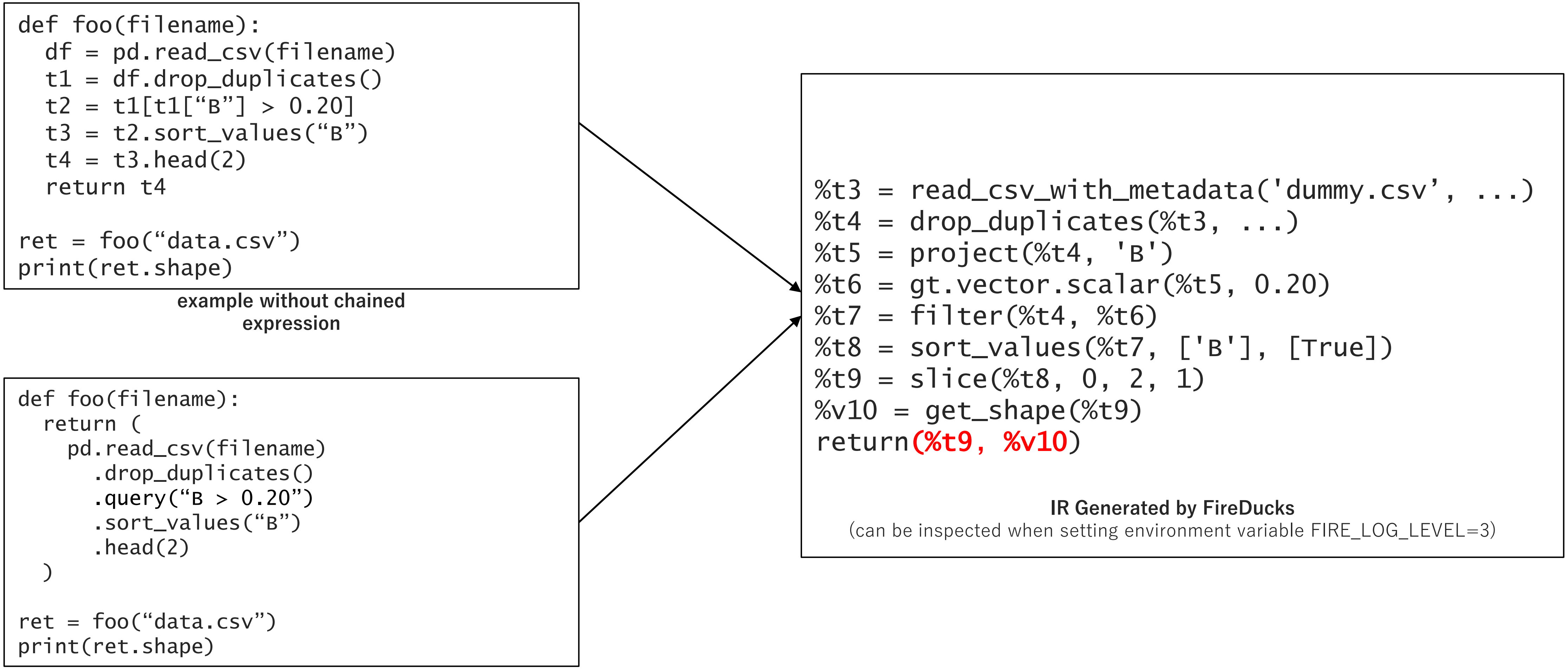
3. Notebook Extension

FireDucks provides simple import extension for interative notebooks.

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

simple integration in a notebook

IR-driven Lazy-execution addresses memory issue with intermediate tables



```
def foo(filename):
  df = pd.read_csv(filename)
  t1 = df.drop_duplicates()
  t2 = t1[t1[“B”] > 0.20]
  t3 = t2.sort_values(“B”)
  t4 = t3.head(2)
  return t4

ret = foo(“data.csv”)
print(ret.shape)
```

example without chained expression

```
def foo(filename):
  return (
    pd.read_csv(filename)
      .drop_duplicates()
      .query(“B > 0.20”)
      .sort_values(“B”)
      .head(2)
  )

ret = foo(“data.csv”)
print(ret.shape)
```

example with chained expression

```
%t3 = read_csv_with_metadata('dummy.csv’, ...)
%t4 = drop_duplicates(%t3, ...)
%t5 = project(%t4, 'B')
%t6 = gt.vector.scalar(%t5, 0.20)
%t7 = filter(%t4, %t6)
%t8 = sort_values(%t7, ['B'], [True])
%t9 = slice(%t8, 0, 2, 1)
%v10 = get_shape(%t9)
return(%t9, %v10)
```

IR Generated by FireDucks
(can be inspected when setting environment variable FIRE_LOG_LEVEL=3)

Resource on FireDucks

Web site (User guide, benchmark, blog)

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

X(twitter) (Release information)

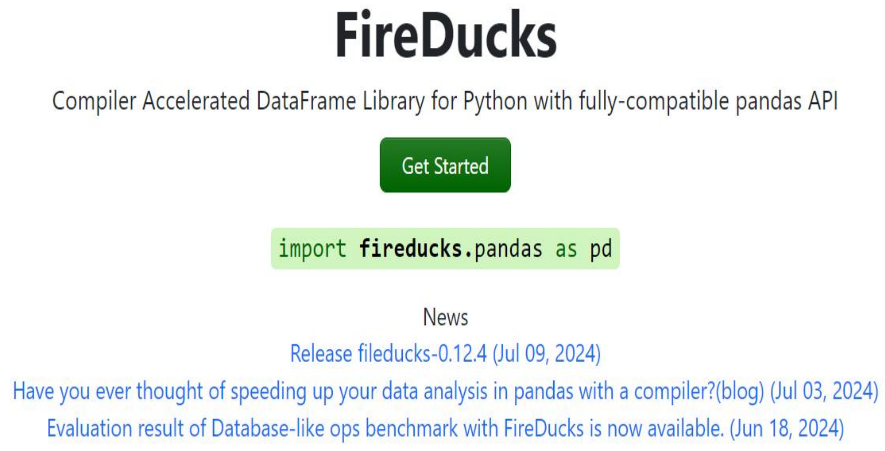
https://x.com/fireducksdev

Github (Issue report)

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

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

Get Started

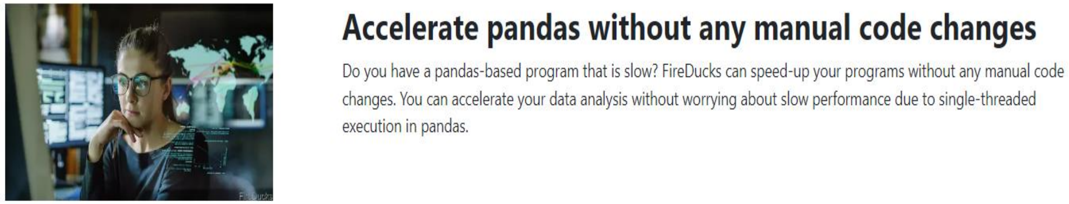
```
import fireducks.pandas as pd
```

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.

Thank You!

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

\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