dask compute() Deferred Computing

We're going to build a somewhat interesting workload and then run it a couple of different ways. Let's start by loading the NYC flight data.

This exercise will reinforce dask dataframe programming concepts by building a set of analyses. We will then use these type of `groupby` and aggregate queries to look at execution properties.

Code that you need to write is indicated with #TODO. I've left the output of the reference implementation in the cells so that you can refer to it for correctness. You can refer to the read-only shared version for this output.

```
%pip install fsspec
%pip install dask[dataframe]
%pip install gcsfs
```

```
Requirement already satisfied: fsspec in
/usr/local/lib/python3.10/dist-packages (2024.10.0)
Requirement already satisfied: dask[dataframe] in
/usr/local/lib/python3.10/dist-packages (2024.10.0)
Requirement already satisfied: click>=8.1 in
/usr/local/lib/python3.10/dist-packages (from dask[dataframe]) (8.1.7)
Requirement already satisfied: cloudpickle>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from dask[dataframe]) (3.1.0)
Requirement already satisfied: fsspec>=2021.09.0 in
/usr/local/lib/python3.10/dist-packages (from dask[dataframe])
(2024.10.0)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from dask[dataframe]) (24.1)
Requirement already satisfied: partd>=1.4.0 in
/usr/local/lib/python3.10/dist-packages (from dask[dataframe]) (1.4.2)
Requirement already satisfied: pyyaml>=5.3.1 in
/usr/local/lib/python3.10/dist-packages (from dask[dataframe]) (6.0.2)
Requirement already satisfied: toolz>=0.10.0 in
/usr/local/lib/python3.10/dist-packages (from dask[dataframe])
(0.12.1)
Requirement already satisfied: importlib-metadata>=4.13.0 in
/usr/local/lib/python3.10/dist-packages (from dask[dataframe]) (8.5.0)
Requirement already satisfied: pandas>=2.0 in
/usr/local/lib/python3.10/dist-packages (from dask[dataframe]) (2.2.2)
Collecting dask-expr<1.2,>=1.1 (from dask[dataframe])
  Downloading dask_expr-1.1.16-py3-none-any.whl.metadata (2.5 kB)
Requirement already satisfied: pyarrow>=14.0.1 in
/usr/local/lib/python3.10/dist-packages (from dask-expr<1.2,>=1.1-
>dask[dataframe]) (17.0.0)
Requirement already satisfied: zipp>=3.20 in
/usr/local/lib/python3.10/dist-packages (from importlib-
metadata>=4.13.0->dask[dataframe]) (3.20.2)
Requirement already satisfied: numpy>=1.22.4 in
```

```
/usr/local/lib/python3.10/dist-packages (from pandas>=2.0-
>dask[dataframe]) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=2.0-
>dask[dataframe]) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=2.0-
>dask[dataframe]) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas>=2.0-
>dask[dataframe]) (2024.2)
Requirement already satisfied: locket in
/usr/local/lib/python3.10/dist-packages (from partd>=1.4.0-
>dask[dataframe]) (1.0.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas>=2.0->dask[dataframe]) (1.16.0)
Downloading dask_expr-1.1.16-py3-none-any.whl (243 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 243.2/243.2 kB 4.3 MB/s eta
0:00:00
ent already satisfied: gcsfs in /usr/local/lib/python3.10/dist-
packages (2024.10.0)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in
/usr/local/lib/python3.10/dist-packages (from gcsfs) (3.10.10)
Requirement already satisfied: decorator>4.1.2 in
/usr/local/lib/python3.10/dist-packages (from gcsfs) (4.4.2)
Requirement already satisfied: fsspec==2024.10.0 in
/usr/local/lib/python3.10/dist-packages (from gcsfs) (2024.10.0)
Requirement already satisfied: google-auth>=1.2 in
/usr/local/lib/python3.10/dist-packages (from gcsfs) (2.27.0)
Requirement already satisfied: google-auth-oauthlib in
/usr/local/lib/python3.10/dist-packages (from gcsfs) (1.2.1)
Requirement already satisfied: google-cloud-storage in
/usr/local/lib/python3.10/dist-packages (from gcsfs) (2.8.0)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from gcsfs) (2.32.3)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!
=4.0.0a1->gcsfs) (2.4.3)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!
=4.0.0a1->gcsfs) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!
=4.0.0a1->gcsfs) (24.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!
=4.0.0a1->gcsfs) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
```

```
/usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!
=4.0.0a1->gcsfs) (6.1.0)
Requirement already satisfied: yarl<2.0,>=1.12.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!
=4.0.0a1->gcsfs) (1.17.0)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp!=4.0.0a0,!
=4.0.0a1->gcsfs) (4.0.3)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth>=1.2->gcsfs)
(5.5.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-auth>=1.2->gcsfs)
(0.4.1)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.10/dist-packages (from google-auth>=1.2->gcsfs)
(4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib-
>gcsfs) (1.3.1)
Requirement already satisfied: google-api-core!=2.0.*,!=2.1.*,!
=2.2.*,!=2.3.0,<3.0.0dev,>=1.31.5 in /usr/local/lib/python3.10/dist-
packages (from google-cloud-storage->gcsfs) (2.19.2)
Requirement already satisfied: google-cloud-core<3.0dev,>=2.3.0 in
/usr/local/lib/python3.10/dist-packages (from google-cloud-storage-
>gcsfs) (2.4.1)
Requirement already satisfied: google-resumable-media>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from google-cloud-storage-
>gcsfs) (2.7.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->gcsfs) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->gcsfs) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->gcsfs) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->gcsfs)
(2024.8.30)
Requirement already satisfied: googleapis-common-
protos<2.0.dev0,>=1.56.2 in /usr/local/lib/python3.10/dist-packages
(from google-api-core!=2.0.*,!=2.1.*,!=2.2.*,!
=2.3.0,<3.0.0dev,>=1.31.5->google-cloud-storage->gcsfs) (1.65.0)
Requirement already satisfied: protobuf!=3.20.0,!=3.20.1,!=4.21.0,!
=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0.dev0,>=3.19.5 in
/usr/local/lib/python3.10/dist-packages (from google-api-core!=2.0.*,!
=2.1.*,!=2.2.*,!=2.3.0,<3.0.0dev,>=1.31.5->google-cloud-storage-
>gcsfs) (3.20.3)
Requirement already satisfied: proto-plus<2.0.0dev,>=1.22.3 in
/usr/local/lib/python3.10/dist-packages (from google-api-core!=2.0.*,!
```

```
=2.1.*,!=2.2.*,!=2.3.0,<3.0.0dev,>=1.31.5->google-cloud-storage-
>gcsfs) (1.25.0)
Requirement already satisfied: google-crc32c<2.0dev,>=1.0 in
/usr/local/lib/python3.10/dist-packages (from google-resumable-
media>=2.3.2->google-cloud-storage->gcsfs) (1.6.0)
Requirement already satisfied: typing-extensions>=4.1.0 in
/usr/local/lib/python3.10/dist-packages (from multidict<7.0,>=4.5-
>aiohttp!=4.0.0a0,!=4.0.0a1->gcsfs) (4.12.2)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth>=1.2->gcsfs) (0.6.1)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests-
oauthlib>=0.7.0->google-auth-oauthlib->gcsfs) (3.2.2)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from yarl<2.0,>=1.12.0-
>aiohttp!=4.0.0a0,!=4.0.0a1->gcsfs) (0.2.0)
```

Read in the NYC Flights data from Google cloud storage and then print the dataframe metadata.

```
import dask.dataframe as dd

df = dd.read_csv('gcs://nycflights/*.csv',
                 storage_options={'token': 'anon'},
                 dtype={'TailNum': str,
                        'CRSElapsedTime': float,
                        'Cancelled': bool})
df
```

```
Dask DataFrame Structure:
                 Year  Month DayofMonth DayOfWeek  DepTime CRSDepTime
ArrTime CRSArrTime UniqueCarrier FlightNum TailNum ActualElapsedTime
CRSElapsedTime  AirTime ArrDelay DepDelay  Origin    Dest Distance
TaxiIn  TaxiOut Cancelled Diverted
npartitions=10

                int64  int64      int64     int64  float64      int64
float64      int64        string     int64  string           float64
float64  float64  float64  float64  string  string  float64  float64
float64      bool    int64
                  ...    ...        ...       ...      ...        ...
...         ...           ...       ...     ...               ...
...      ...      ...       ...     ...     ...      ...      ...
...       ...      ...
...                ...    ...        ...       ...      ...        ...
...         ...           ...       ...     ...               ...
...      ...      ...       ...     ...     ...      ...      ...
...       ...      ...
                  ...    ...        ...       ...      ...        ...
```

```
...        ...           ...       ...     ...                ...
...      ...      ...       ...     ...     ...      ...       ...
...       ...      ...
                  ...    ...        ...       ...      ...        ...
...        ...           ...       ...     ...                ...
...      ...      ...       ...     ...     ...      ...       ...
...       ...      ...
Dask Name: to_pyarrow_string, 2 graph layers
```

Let's build a set of queries around the performance of particular planes, identified by tail number. The pattern will be to groupby('TailNum') and then compute statistics.

Query: What is the average departure delay 'DepDelay' for each plane?

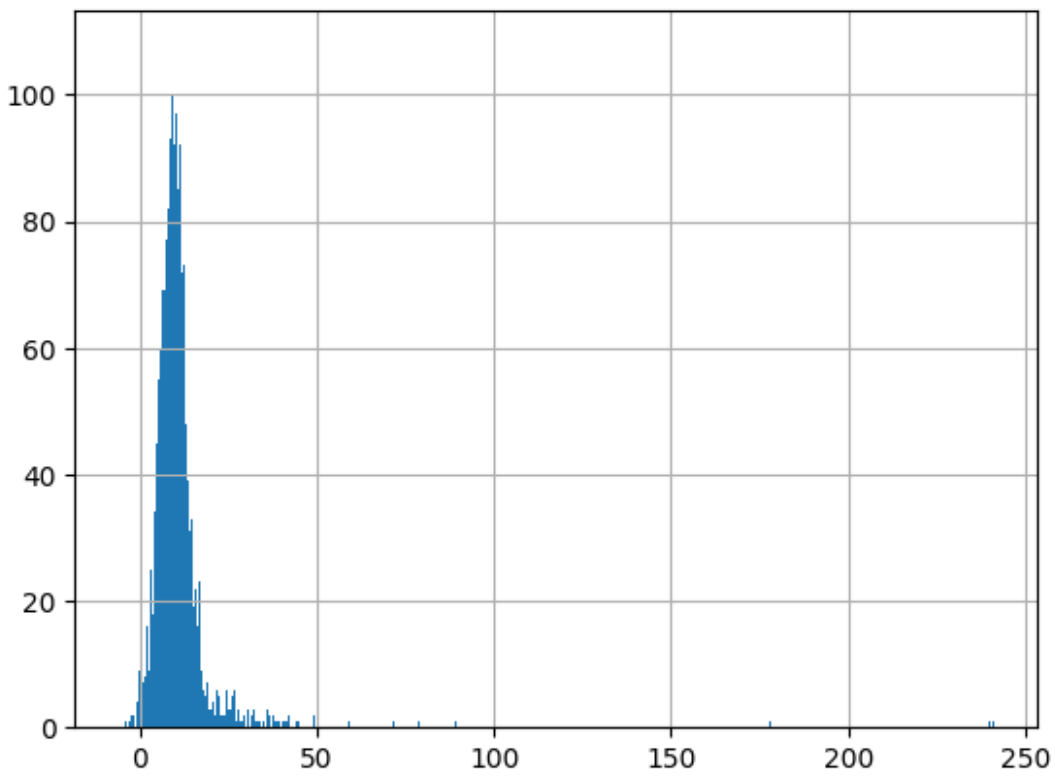
```
#TODO
df_delay = df.groupby('TailNum').DepDelay.mean().compute()
df_delay
```

```
TailNum
EI-BWD    11.213501
EI-CAL    23.846154
EI-CAM    26.611511
EI-CIW    12.918182
N050AA     9.180180
           ...
N976TW    -2.294118
N978TW     0.000000
N979TW     5.250000
N980TW     3.428571
N982TW    14.000000
Name: DepDelay, Length: 3712, dtype: float64
```

Interesting, some planes were early, lets plot a histrogram of the distribution with 1000 bins.

```
%matplotlib inline
df_delay.hist(bins=1000)
```

```
<Axes: >
```



OK, we have very few chronically bad planes. Let's find those that are 30 (or more) minutes late on average.

```
import numpy as np
lateplanes = df_delay[df_delay > 30].index
print(np.sort(lateplanes))
```

```
['N101UW' 'N102UW' 'N104UW' 'N106UW' 'N133JC' 'N133TW' 'N134TW'
'N14249'
 'N144JC' 'N147US' 'N151AW' 'N151UA' 'N152UA' 'N153US' 'N154AA'
'N154AW'
 'N155US' 'N156AW' 'N158AW' 'N161US' 'N168AW' 'N169AW' 'N17010'
'N17011'
 'N1738D' 'N1739D' 'N174AW' 'N174GM' 'N174UA' 'N175UA' 'N17789'
'N1854U'
 'N195UA' 'N199UA' 'N224DA' 'N224NW' 'N225NW' 'N235NW' 'N303TW'
'N304AW'
 'N305AW' 'N305TW' 'N307TW' 'N322AW' 'N328AW' 'N33021' 'N3310L'
'N331AW'
 'N375DA' 'N376DL' 'N379DL' 'N382DA' 'N53110' 'N53116' 'N534TW'
'N6700'
 'N701UW' 'N706UW' 'N708UW' 'N713DA' 'N713UW' 'N716DA' 'N719DA'
'N724DA'
 'N727UW' 'N733DS' 'N735D' 'N737D' 'N760DH' 'N78019' 'N787DL' 'N789DL'
 'N802DE' 'N805DE' 'N817AA' 'N8911E' 'N93104' 'N93107' 'N93108'
```

```
'N93109'
 'N93119' 'N96S' 'N971Z' 'N976UA' 'N993UA' 'NEIDLA' 'UNKNOW']
```

OK, this is a hard query. Build a dataframe that is a subset all the data associated with the late planes. There are many ways to solve this problem. I would recommend looking at the `isin()` function in dask.

```
df_late = df[df['TailNum'].isin(lateplanes)].compute()
df_late
```

```
{"type":"dataframe","variable_name":"df_late"}
```

Double check that the planes indexes in `df_late` match the answer to the `lateplanes` query.

```
latelist = df_late.TailNum.unique()
print(np.sort(latelist))

print(all(np.sort(latelist) == np.sort(lateplanes)))
```

```
['N101UW' 'N102UW' 'N104UW' 'N106UW' 'N133JC' 'N133TW' 'N134TW'
'N14249'
 'N144JC' 'N147US' 'N151AW' 'N151UA' 'N152UA' 'N153US' 'N154AA'
'N154AW'
 'N155US' 'N156AW' 'N158AW' 'N161US' 'N168AW' 'N169AW' 'N17010'
'N17011'
 'N1738D' 'N1739D' 'N174AW' 'N174GM' 'N174UA' 'N175UA' 'N17789'
'N1854U'
 'N195UA' 'N199UA' 'N224DA' 'N224NW' 'N225NW' 'N235NW' 'N303TW'
'N304AW'
 'N305AW' 'N305TW' 'N307TW' 'N322AW' 'N328AW' 'N33021' 'N3310L'
'N331AW'
 'N375DA' 'N376DL' 'N379DL' 'N382DA' 'N53110' 'N53116' 'N534TW'
'N6700'
 'N701UW' 'N706UW' 'N708UW' 'N713DA' 'N713UW' 'N716DA' 'N719DA'
'N724DA'
 'N727UW' 'N733DS' 'N735D' 'N737D' 'N760DH' 'N78019' 'N787DL' 'N789DL'
 'N802DE' 'N805DE' 'N817AA' 'N8911E' 'N93104' 'N93107' 'N93108'
'N93109'
 'N93119' 'N96S' 'N971Z' 'N976UA' 'N993UA' 'NEIDLA' 'UNKNOW']
True
```

Now, let's get a sense of what airports these planes fly out of. For the planes in the late_list, let's find out the total delay at these airports, the average delay by airport and the total number of flights at each airport.

```
#TODO total DepDelay for planes by Origin airport
late_airports_df = df_late.groupby('Origin').DepDelay.sum()
late_airports_df
```

```
Origin
EWR    16982.0
JFK    61684.0
LGA    27669.0
Name: DepDelay, dtype: float64
```

```
#TODO average DepDelay for planes by Origin airport
late_airports_df = df_late.groupby('Origin').DepDelay.mean()
late_airports_df
```

```
Origin
EWR    43.101523
JFK    41.763033
LGA    36.027344
Name: DepDelay, dtype: float64
```

```
#TODO number of late flights by Origin airport
late_airports_df = df_late.groupby('Origin').TailNum.count()
late_airports_df
```

```
Origin
EWR    18722
JFK     4564
LGA    13053
Name: TailNum, dtype: int64
```

Deferred computing

We are going to show the value of deferred computation by timing the following queries in two different ways:

```
df1 = df.groupby(['Origin','TailNum']).DepDelay.mean()
df2 = df.groupby(['TailNum','Origin']).DepDelay.mean()
df3 = df.groupby(['Origin','TailNum']).DepDelay.max()
df4 = df.groupby(['TailNum','Origin']).DepDelay.max()
```

1. In one cell, add these lines and then call `compute()` on every step.
2. In the next cell, add the lines and only call compute at the end.

First reload the data:

```
import dask.dataframe as dd

df = dd.read_csv('gcs://nycflights/*.csv',
                 storage_options={'token': 'anon'},
                 dtype={'TailNum': str,
                        'CRSElapsedTime': float,
                        'Cancelled': bool})
df
```

```
Dask DataFrame Structure:
                  Year  Month DayofMonth DayOfWeek  DepTime CRSDepTime
ArrTime CRSArrTime UniqueCarrier FlightNum TailNum ActualElapsedTime
CRSElapsedTime  AirTime ArrDelay DepDelay  Origin    Dest Distance
TaxiIn  TaxiOut Cancelled Diverted
npartitions=10

                int64  int64      int64     int64  float64      int64
float64      int64        string     int64  string           float64
float64  float64  float64  float64  string  string  float64  float64
float64      bool    int64
                  ...    ...        ...       ...      ...        ...
...        ...           ...       ...     ...               ...
...      ...      ...      ...     ...     ...      ...      ...
...       ...      ...
...                ...    ...        ...       ...      ...        ...
...        ...           ...       ...     ...               ...
...      ...      ...      ...     ...     ...      ...      ...
...       ...      ...
                  ...    ...        ...       ...      ...        ...
...        ...           ...       ...     ...               ...
...      ...      ...      ...     ...     ...      ...      ...
...       ...      ...
                  ...    ...        ...       ...      ...        ...
...        ...           ...       ...     ...               ...
...      ...      ...      ...     ...     ...      ...      ...
...       ...      ...
Dask Name: to_pyarrow_string, 2 graph layers
```

Run the workload calling `compute()` on every line.

```
%%time

#TODO
df1 = df.groupby(['Origin','TailNum']).DepDelay.mean().compute()
df2 = df.groupby(['TailNum','Origin']).DepDelay.mean().compute()
df3 = df.groupby(['Origin','TailNum']).DepDelay.max().compute()
df4 = df.groupby(['TailNum','Origin']).DepDelay.max().compute()

CPU times: user 45.6 s, sys: 3.97 s, total: 49.5 s
Wall time: 41.2 s
```

Load the data again to make sure that intermediate results are not cached and run the entire workload calling `compute()` just once.

```
import dask.dataframe as dd
df = dd.read_csv('gs://nycflights/*.csv',
                 storage_options={'token': 'anon'},
                 dtype={'TailNum': str,
```

```
                        'CRSElapsedTime': float,
                        'Cancelled': bool})

%%time

#TODO
df1 = df.groupby(['Origin','TailNum']).DepDelay.mean()
df2 = df.groupby(['TailNum','Origin']).DepDelay.mean()
df3 = df.groupby(['Origin','TailNum']).DepDelay.max()
df4 = df.groupby(['TailNum','Origin']).DepDelay.max()

dd.compute(df1, df2, df3, df4)

CPU times: user 12.6 s, sys: 1.26 s, total: 13.8 s
Wall time: 11.9 s

(Origin  TailNum
 EWR     EI-BWD      9.355140
         EI-CIW     16.283019
         N050AA      8.309677
         N051AA      5.949275
         N052AA     21.845070
                       ...
 LGA     N993UA     55.000000
         N994UA     -3.250000
         N995UA     14.600000
         N996UA      9.333333
         N998UA      1.750000
 Name: DepDelay, Length: 8861, dtype: float64,
 TailNum  Origin
 EI-BWD   EWR        9.355140
          JFK       11.575758
          LGA       11.626866
 EI-CAL   JFK       23.846154
 EI-CAM   JFK       26.611511
                       ...
 N993UA   LGA       55.000000
 N994UA   LGA       -3.250000
 N995UA   LGA       14.600000
 N996UA   LGA        9.333333
 N998UA   LGA        1.750000
 Name: DepDelay, Length: 8861, dtype: float64,
 Origin  TailNum
 EWR     EI-BWD     177.0
         EI-CIW     331.0
         N050AA     248.0
         N051AA     140.0
         N052AA     996.0
                    ...
 LGA     N993UA     227.0
```

```
        N994UA       0.0
        N995UA      59.0
        N996UA      17.0
        N998UA       8.0
Name: DepDelay, Length: 8861, dtype: float64,
TailNum  Origin
EI-BWD   EWR       177.0
         JFK       414.0
         LGA       208.0
EI-CAL   JFK       350.0
EI-CAM   JFK       225.0
                   ...
N993UA   LGA       227.0
N994UA   LGA         0.0
N995UA   LGA        59.0
N996UA   LGA        17.0
N998UA   LGA         8.0
Name: DepDelay, Length: 8861, dtype: float64)
```

Outcomes

- Wrestled with dataframes syntax and concepts. Good for you.
- Witnessed the benefit of deferred computation.

Questions

1. On computational reuse in execution graphs:

a. How much faster is it to defer the computation to the end versus calling `compute()` on every line?

Computing on every line took me 41 seconds (wall time), while computing once at the end took me only 12 seconds. This is a speedup of about 3.5.

b. What computations are shared in the workflow? Be specific, i.e. identify the code.

The groupby operation is shared across all four tasks in the workflow, with two tasks grouping by Origin first and then TailNum, and the other two grouping in the opposite order. Additionally, the indexing into DepDelay is shared across all four tasks. The only computations that are unique to all four tasks are the actual mean and max calculations, which depend on the specific dataframe generated for that task.

c. Explain the speedup realized in 1(a). Why is it faster?

The speedup is largely due to the dask execution graph being shared across two tasks - the tasks that group by Origin,TailNum have the same start for their execution paths, while the tasks that group by TailNum,Origin also share their starts. This means that dask can optimize the computation by doing these processes only once, which leads to a speedup of about 3.5.