**Dataset Overview**

| **Aspect** | **Value** |
| --- | --- |
| **States** | 50 |
| **NAICS 6‑digit codes** | 1 000 |
| **IDs per quarter** | 150 million |
| **Year‑quarters** | 20 |
| **Cohort size** | 1 000 IDs |
| **Total rows** | ~3 billion |

Tracking 1 000 IDs over 20 quarters across all states involves extracting around three billion records. A streaming filter can handle this in one pass.

**Complexity and Runtime Estimates**

| **Method** | **Complexity (big‑O)** | **Per‑quarter time (150 M rows)** | **20‑quarter time (3 B rows)** |
| --- | --- | --- | --- |
| **Pure Python loop** | O(N) | ≈8.75 min | ≈175 min (~3 h) |
| **PyPy loop** | O(N) | ≈5.8 min | ≈116 min (~2 h) |
| **Pandas (C engine)** | O(N) | ≈3.4 min | ≈68 min (~1 h) |
| **Pandas with PyArrow** | O(N) | ≈1.4 min | ≈27 min (~0.5 h) |

These estimates scale linearly with the number of rows. They are extrapolated from benchmarks on a 120 million‑row dataset: 7 min for plain Python, 4 min 40 s for PyPy[datapythonista.me](https://datapythonista.me/blog/pandas-with-hundreds-of-millions-of-rows#:~:text=With%20this%20method%20we%20solve,1%20Mb%20of%20heap%20memory), 2 min 45 s for Pandas[datapythonista.me](https://datapythonista.me/blog/pandas-with-hundreds-of-millions-of-rows#:~:text=This%20program%20completes%20in%20around,data%20into%20memory%2C%20and%20it), and 1 min 10 s for Pandas with PyArrow[datapythonista.me](https://datapythonista.me/blog/pandas-with-hundreds-of-millions-of-rows#:~:text=With%20the%20PyArrow%20engine%2C%20our,dataframes%20will%20be%20the%20same).

**Recommendations**

* **Stream and filter:** Use a set of the 1 000 IDs and stream the data once, checking membership for each row. This yields linear complexity with low memory usage.
* **Leverage optimized readers:** Pandas with the PyArrow engine or distributed tools like Dask can dramatically reduce I/O time. Avoid Python loops where possible.
* **Index for repeated queries:** If you will run many cohort queries, build an index on the ID column in a database. The one‑time cost is O(N), and subsequent queries run in O(k × t) where k is the number of IDs and t the number of time periods.
* **Mind hardware limits:** Disk I/O is usually the bottleneck[datapythonista.me](https://datapythonista.me/blog/pandas-with-hundreds-of-millions-of-rows#:~:text=We%20can%20see%20that%20more,focusing%20in%20the%20next%20sections). Use SSDs and process one quarter at a time when loading into memory‑intensive tools like Pandas.