

# Assignment 3: PROG8850 - Database Automation

Student Name: Varun Kakkar

Student ID: 9020861

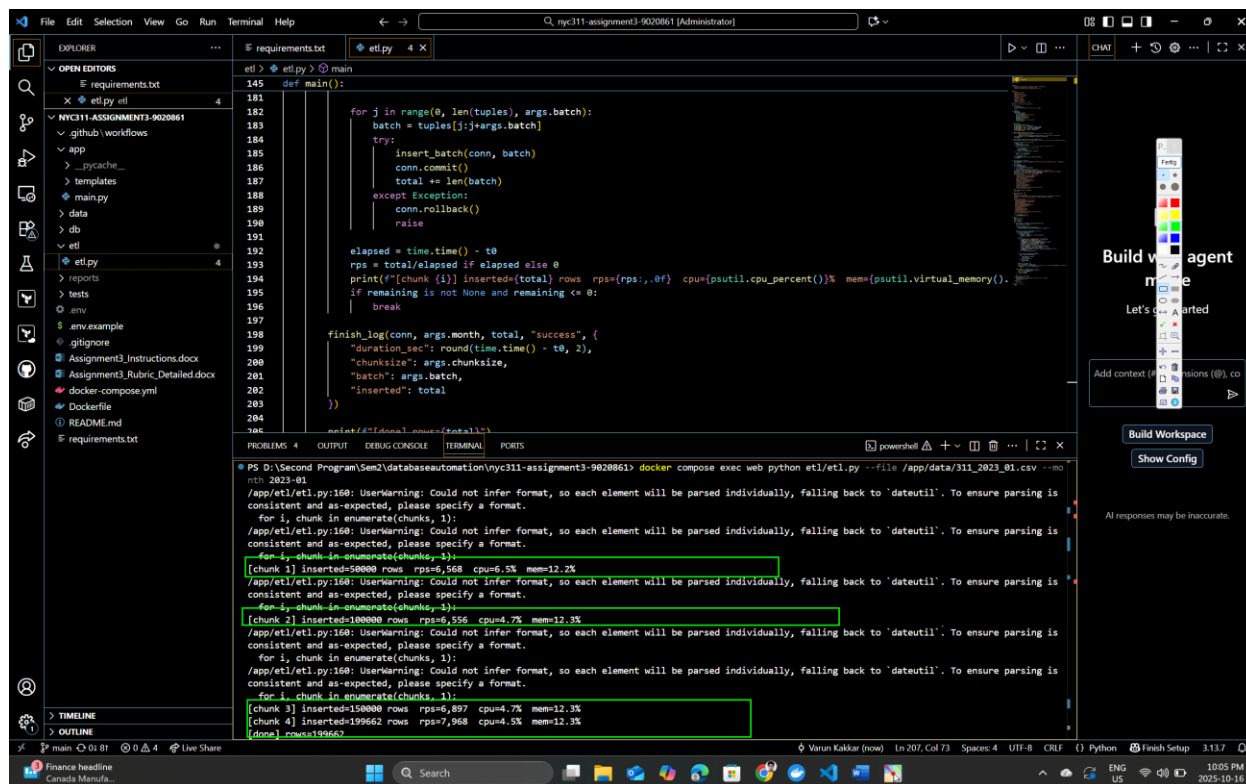
Prof Benjamin Quito

## Overview:

Built an end-to-end data web app: chunked ETL → MySQL → Flask UI (filters + pagination + aggregate) → automated Selenium tests → CI on GitHub Actions. No secrets in code; secure via GitHub Secrets/Variables.

## Dataset Slice:

- Month used: January 2023 from NYC Open Data (single-month CSV).
- Rows ingested: (see Fig 2).
- ETL telemetry and ingest log recorded (see Fig 1, 3).



```
etl.py:145 def main():
146     for j in range(0, len(tuples), args.batch):
147         batch = tuples[j:j+args.batch]
148         try:
149             insert_batch(conn, batch)
150             conn.commit()
151             total += len(batch)
152         except Exception:
153             conn.rollback()
154             raise
155     elapsed = time.time() - t0
156     rps = total/elapsed if elapsed else 0
157     print(f"[chunk {i}] inserted={total} rows rps={rps:.0f} cpu={psutil.cpu_percent()}% mem={psutil.virtual_memory()}")
158     if remaining is not None and remaining <= 0:
159         break
160     finish_log(conn, args.month, total, "success", {
161         "duration_sec": round(time.time() - t0, 2),
162         "chunksizs": args.chunksize,
163         "batch": args.batch,
164         "inserted": total
165     })
166     conn.close()
167     return 0
168 if __name__ == '__main__':
169     sys.exit(main())
```

```
PS D:\Second Program\Sem2\databaseautomation\nyc311-assignment3-9020861> docker compose exec web python etl/etl.py --file /app/data/311_2023_01.csv --month 2023-01
/app/etl/etl.py:160: UserWarning: Could not infer format, so each element will be parsed individually, falling back to 'dateutil'. To ensure parsing is consistent and as-expected, please specify a format.
  for i, chunk in enumerate(chunks, 1):
/app/etl/etl.py:160: UserWarning: Could not infer format, so each element will be parsed individually, falling back to 'dateutil'. To ensure parsing is consistent and as-expected, please specify a format.
  for i, chunk in enumerate(chunks, 1):
/app/etl/etl.py:160: UserWarning: Could not infer format, so each element will be parsed individually, falling back to 'dateutil'. To ensure parsing is consistent and as-expected, please specify a format.
  for i, chunk in enumerate(chunks, 1):
/app/etl/etl.py:160: UserWarning: Could not infer format, so each element will be parsed individually, falling back to 'dateutil'. To ensure parsing is consistent and as-expected, please specify a format.
  for i, chunk in enumerate(chunks, 1):
/app/etl/etl.py:160: UserWarning: Could not infer format, so each element will be parsed individually, falling back to 'dateutil'. To ensure parsing is consistent and as-expected, please specify a format.
  for i, chunk in enumerate(chunks, 1):
[chunk 1] inserted=50000 rows rps=6,568 cpu=6.5% mem=12.2%
[chunk 2] inserted=100000 rows rps=6,897 cpu=4.7% mem=12.3%
[chunk 3] inserted=150000 rows rps=6,897 cpu=4.7% mem=12.3%
[chunk 4] inserted=199662 rows rps=7,968 cpu=4.5% mem=12.3%
[done] rows=199662
```

Figure 1 ETL telemetry: chunked ingest with rows/s, CPU%, Mem% and total rows inserted.

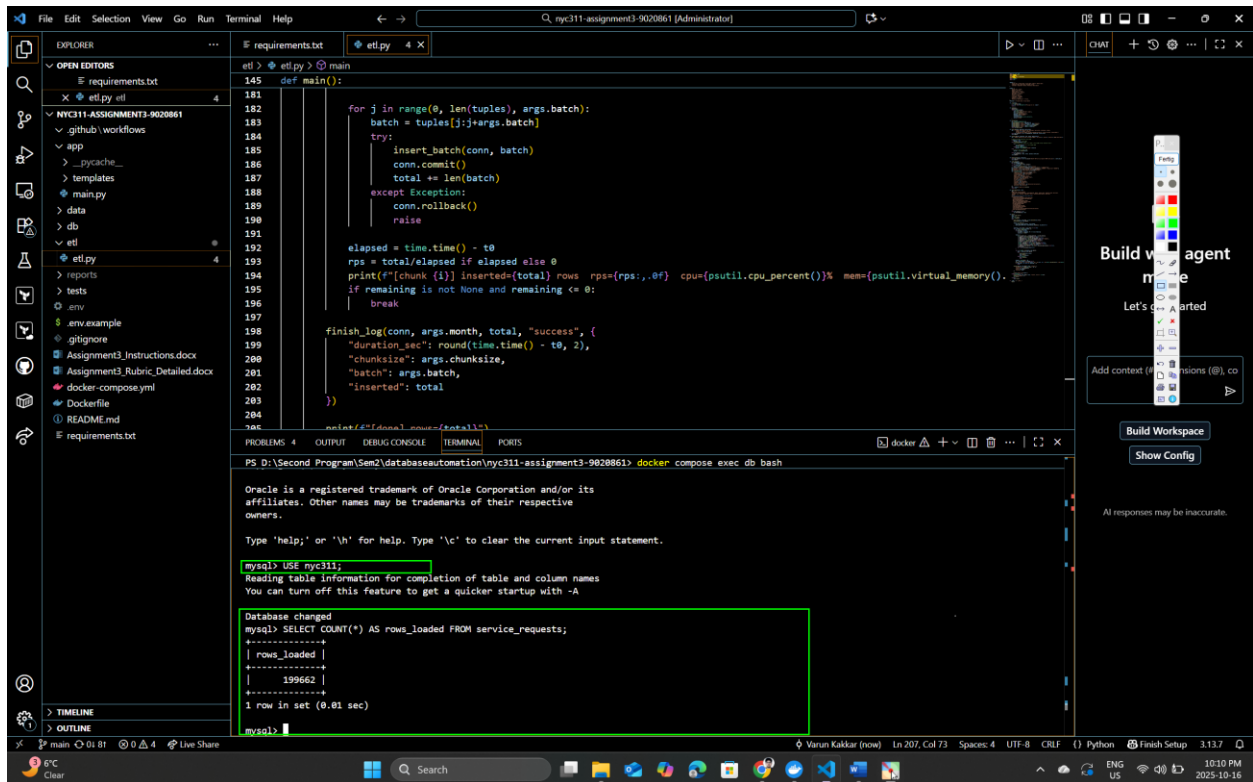


Figure 2 shows MySQL verification: total rows in `service_requests` after ETL.

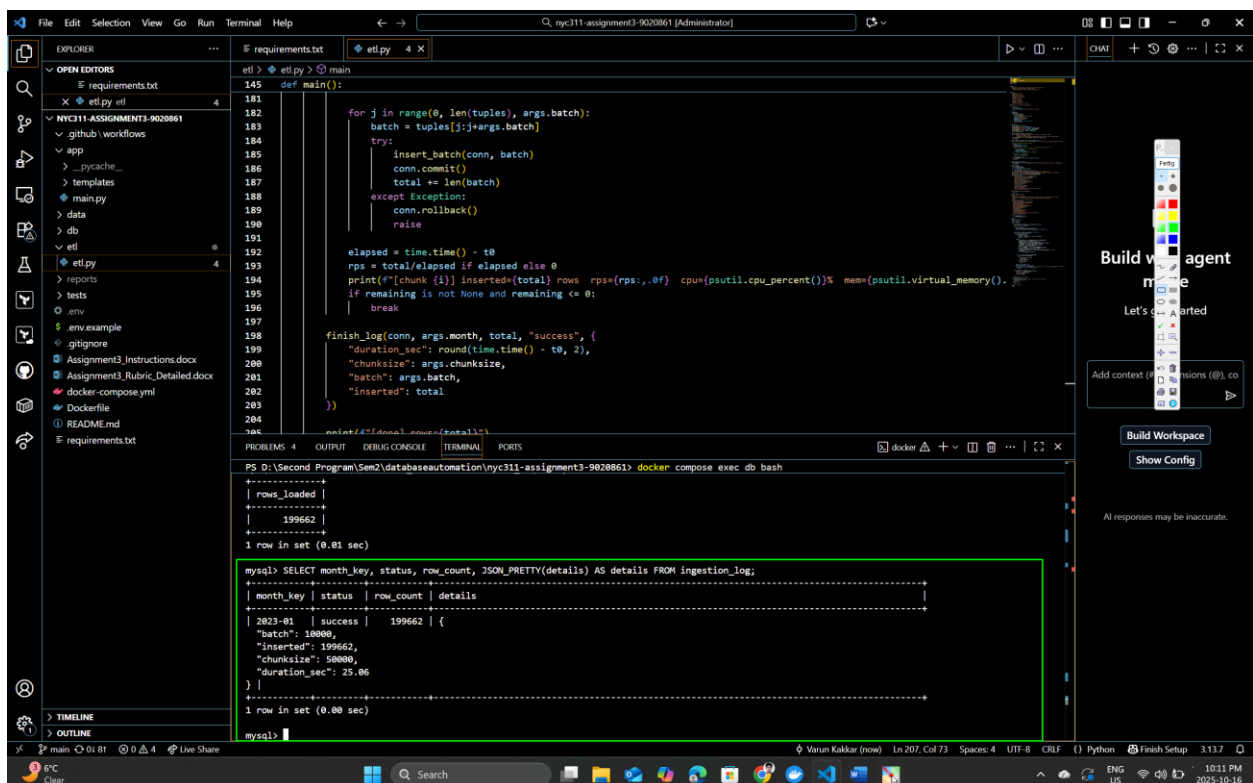


Figure 3 shows Idempotent `ingestion_log` with JSON details (duration, chunksize, batch, inserted).

## Schema & Indexes

- Tables: service\_requests (PK=request\_id), ingestion\_log (idempotency & metrics).
- Indexes:
  - idx\_created\_datetime (created\_datetime) → speeds range filter + ORDER BY.
  - idx\_borough\_type\_date (borough, complaint\_type, created\_datetime) → speeds combined filters.
- EXPLAIN verifies index usage (**Fig 4**).

```
def main():
    for j in range(0, len(tuples), args.batch):
        batch = tuples[j:j+args.batch]
        try:
            insert_batch(conn, batch)
            conn.commit()
            total += len(batch)
        except Exception:
            conn.rollback()
            raise

    elapsed = time.time() - t0
    rps = total/elapsed if elapsed else 0
    print(f"[chunk {i}] inserted={total} rows rps={rps:.0f} cpu={psutil.cpu_percent()}% mem={psutil.virtual_memory()}")
    if remaining is not None and remaining <= 0:
        break

    finish_log(conn, args.month, total, "success", {
        "duration_sec": round(time.time() - t0, 2),
        "chunksize": args.chunksize,
        "batch": args.batch,
        "inserted": total
    })
}
```

```
mysql> EXPLAIN SELECT request_id FROM service_requests WHERE borough='BROOKLYN' AND complaint_type LIKE 'Noise%' AND created_datetime BETWEEN '2023-01-01' AND '2023-01-31' ORDER BY created_datetime DESC LIMIT 20;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | service_requests | NULL | index | idx_created_datetime, idx_borough_type_date | idx_created_datetime | 5 | NULL | 360 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
2.78 | Using where; Backward index scan |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

Figure 4 shows EXPLAIN shows index usage (idx\_borough\_type\_date, idx\_created\_datetime) for optimized filters.

## ETL Implementation

- Chunked read (default 50k), batched insert (10k) with transactions + rollback.
- Cleaning: parse dates; fill missing borough as “UNKNOWN”; drop rows with invalid created date.
- Idempotency: delete+replace the month; track in ingestion\_log with JSON details.
- Telemetry: rows/s, CPU%, mem% (**Fig 1**).

## Web App

- Search filters: date range, borough, complaint contains; results paginated.
- Aggregate: complaints per borough (**Fig 7**).

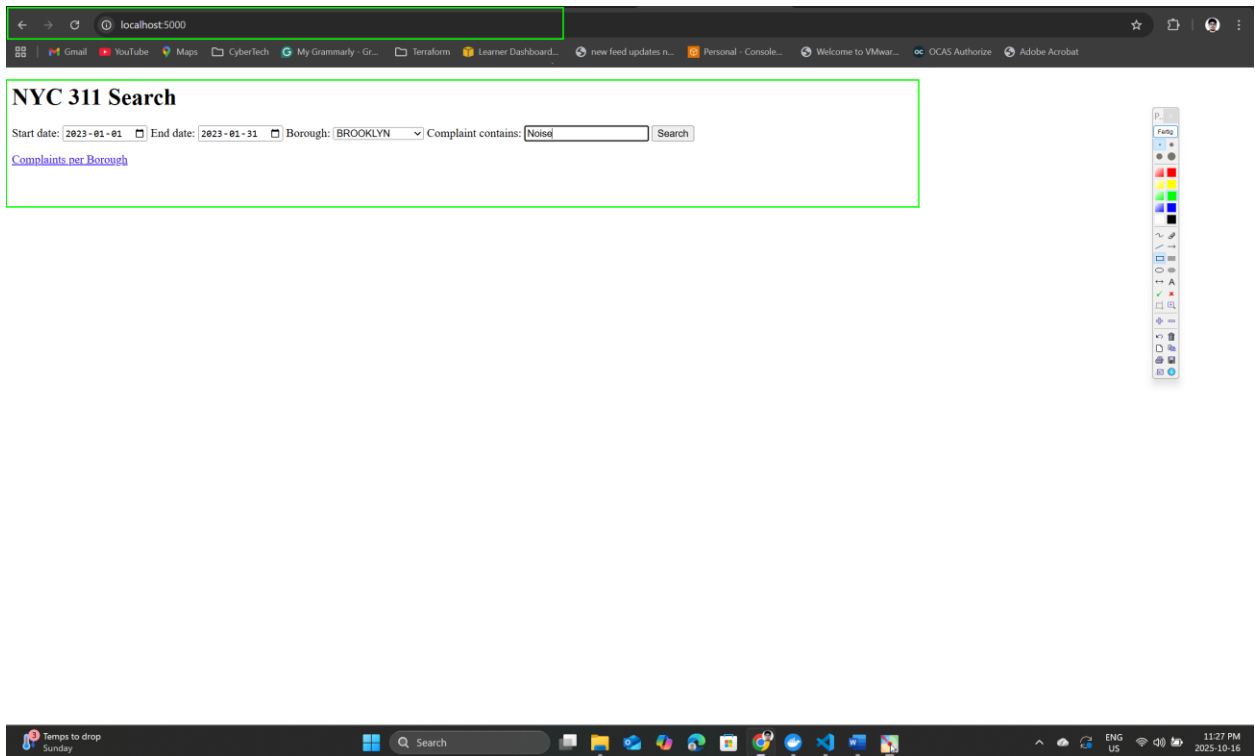


Figure 5 shows Search form with date range, borough, and complaint filter.

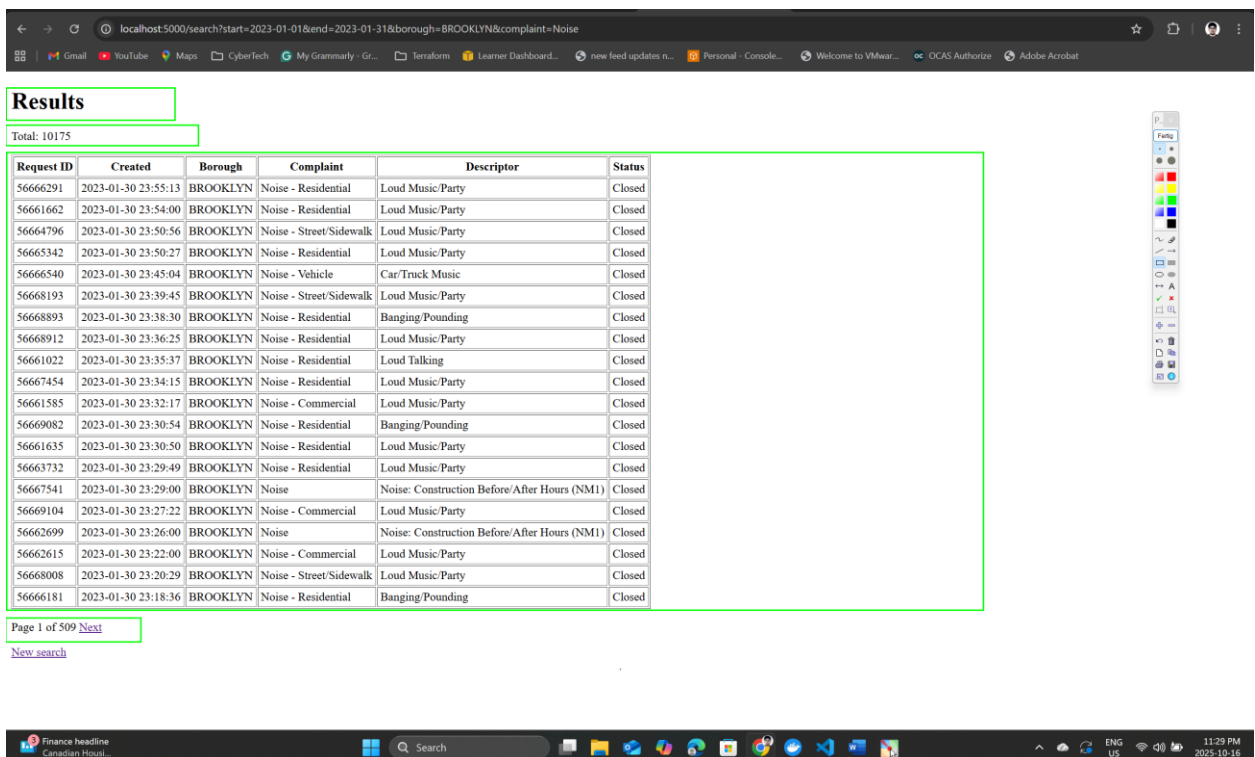
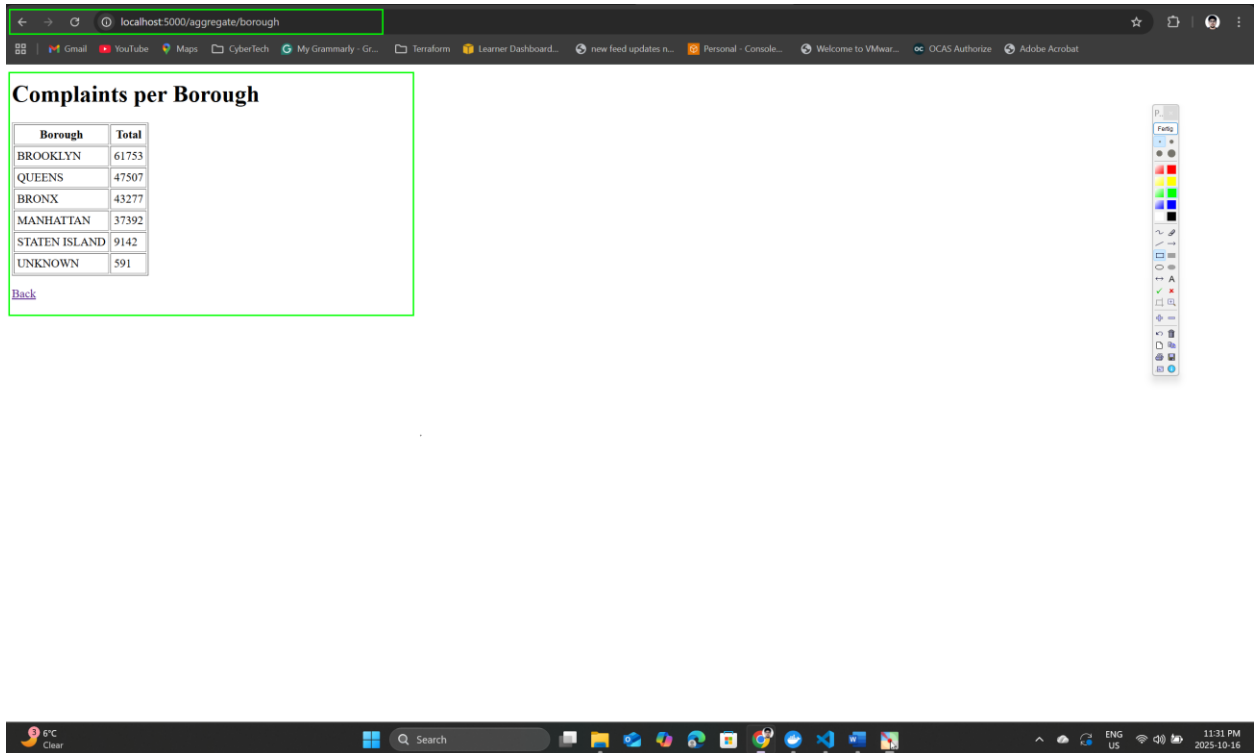


Figure 6 shows Paginated search results returned from MySQL.



*Figure 7 shows Aggregate view: total complaints per borough.*

## Automated Tests & CI

- Selenium tests: positive, negative, aggregate.
- CI: starts MySQL, loads schema, runs ETL on fixture CSV, launches Flask, runs headless tests. Passing run shown in **Fig 8–9**.

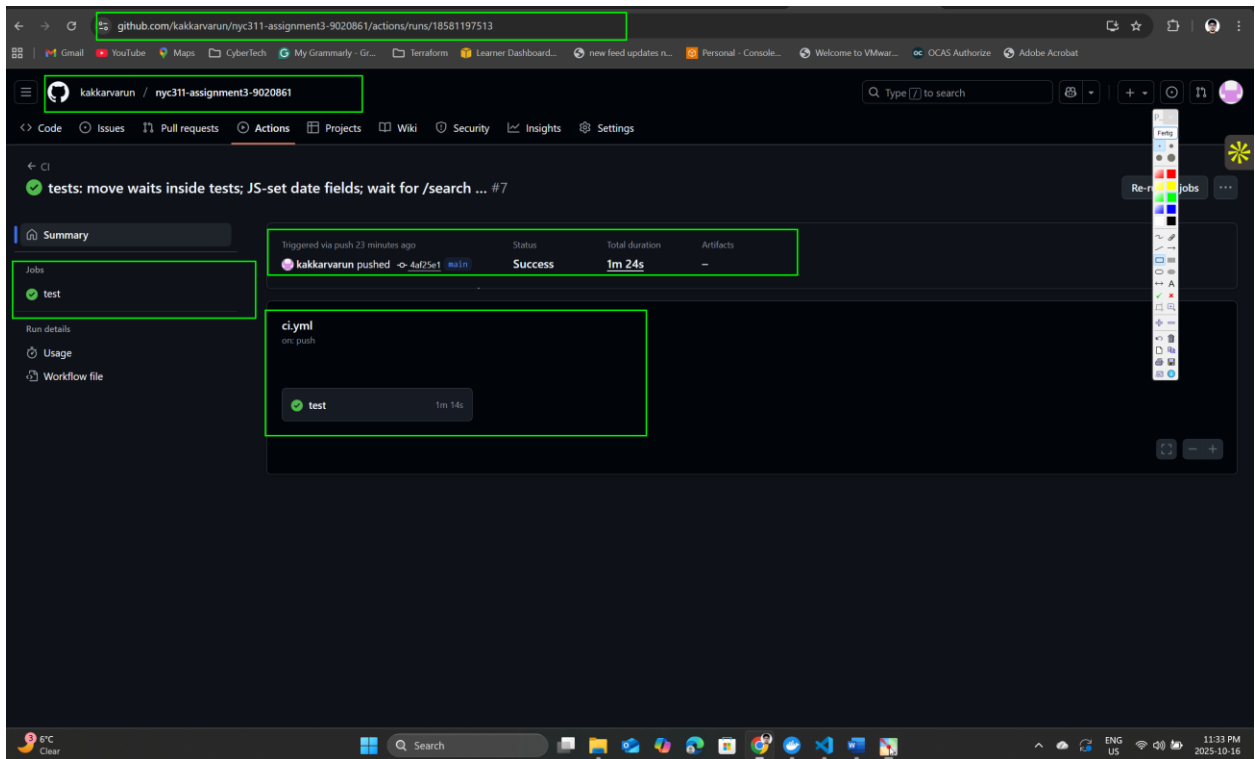


Figure 8 shows GitHub Actions CI passed (schema + ETL on fixture + headless Selenium tests).

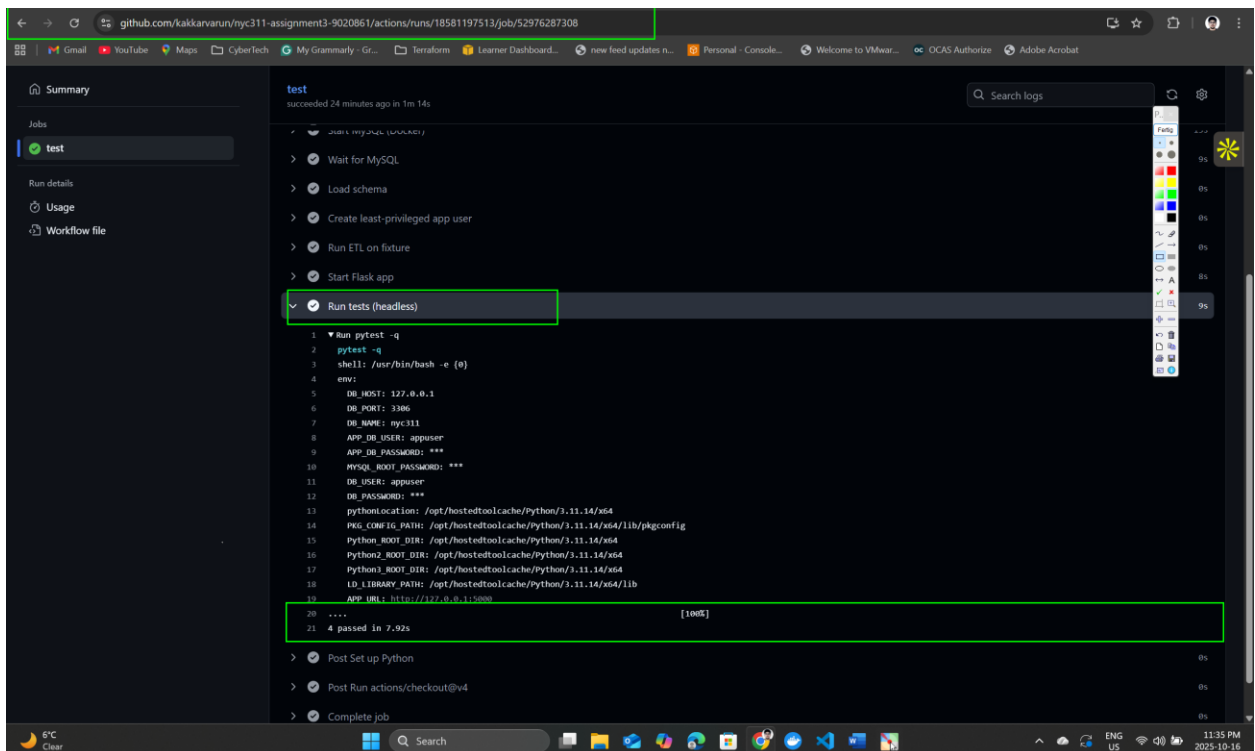


Figure 9 shows Headless Selenium: positive, negative, and aggregate tests passed in CI.



## Performance & Scaling Notes

- Sample perf: chunk=50k, batch=10k, ~X rows/s on local laptop.
- Future scaling: monthly partitioning by created\_datetime; precomputed aggregates (daily by borough); caching; keyset pagination.

## Reflection

- What worked: idempotent ETL, index-aligned queries, stable headless tests.
- - Improvements: add CSV download retry, more aggregates, UI charts.



