NYC Forensic Transit Audit System

Technical Architecture & Operational Guide

Generated: February 07, 2026
Author: Muhammad Umar Afzaal

1. System Overview

This project implements a forensic data audit pipeline designed to analyze New York City taxi data before and after the 2025 Congestion Pricing implementation. It leverages DuckDB for out-of-core processing (handling large datasets without memory crashes) and provides both interactive (Streamlit/Tkinter) and static (PDF) reporting layers.

2. Project Directory Structure

Directory/File	Description & Responsibility
audit_manager.bat	Master Control Script (CLI Entry Point).
audit_pipeline.py	Main Python Orchestrator. Coordinates phases 1-8.
src/	Core Source Code Modules.
■■■ raw_loader.py	Data Ingestion: Downloads Parquet files from NYC TLC.
■■■ data_definitions.py	Schema Unification: Maps Yellow/Green taxonomies.
■■■ ghost_trip_filter.py	Data Refinery: Removes anomalies (speed > 65mph).
■■■ missing_value_handler.py	Imputation: SI-Model for missing Dec 2025 data.
■■■ geo_mapping.py	Spatial Logic: Defines CBD zones & categorizes trips.
■■■ fleet_analytics.py	Analytics Engine: Calculates revenue, leakage, metrics.
■■■ chart_generator.py	Visualization: Generates Matplotlib/Seaborn plots.
■■■ document_builder.py	Reporting: Compiles the final PDF dossier.
dashboard/	UI Application Code.
■■■ web_dashboard.py	Streamlit Web App (Interactive Telemetry).
■■■ gui_dashboard.py	Tkinter Desktop App (Native Interface).
data/	Data Warehouse (Raw, Processed, Datamarts)
outputs/	Generated Artifacts (PDFs, Logs, Figures)

3. ETL Pipeline Architecture

Phase 1: Ingestion (raw_loader.py)

Function: `execute_full_data_harvest()`
Logic: Connects to the NYC TLC S3 bucket. It iterates through the target months (Jan-Mar 2024 & 2025). It uses `requests` to download the Parquet files and verifies integrity using file size checks.

Phase 2: Schema Alignmnent (data_definitions.py)

Function: `orchestrate_fleet_schema_alignment()`
Logic: Yellow and Green taxis have different column names (e.g., 'tpep_pickup_datetime' vs 'lpep_pickup_datetime'). This module maps them to a single canonical schema: `[pickup_time,

dropoff_time, pickup_loc, ...]`. It uses DuckDB to perform this transformation efficiently on disk.

Phase 3: Cleaning (ghost_trip_filter.py)

Function: `process_refinery_batch()`
Logic: Removes 'Ghost Trips'. Applies detection heuristics:
- Velocity > 65 MPH (Impossible in NYC)
- Duration < 60s with Fare > $20 (Suspicious)
- Negative Fares (System errors)

Clean data is saved to `data/processed/purified/`.

Phase 4: Missing Data (missing_value_handler.py)

Function: `run_comprehensive_data_recovery()`
Logic: Simulates the future (Dec 2025) if data is missing. It uses an SI-Model (Synthesis-Imputation) which takes 70% of Dec 2024 patterns and blends them with a 30% growth trend from early 2025 to create a realistic synthetic dataset.

4. Analytics & Visualization

Geospatial Logic (geo_mapping.py)

Identifies trips entering/exiting the Manhattan CBD. It queries the taxi zone shapefiles. If a pickup is Outside CBD and Dropoff is Inside CBD, it tags the trip as 'entering_zone' (Liable for Toll).

Visualization Engine (chart_generator.py)

Generates the static PNGs found in `outputs/figures/`. Uses Matplotlib and Seaborn. It reads the aggregated metrics from DuckDB and plots:
- **Time Series:** Daily trip volume trends.
- **Fiscal Trajectory:** Revenue collected vs Leakage.
- **Spatial Load:** Which zones have high traffic.

5. Execution Flow Execution

When you run `audit_manager.bat` -> Option [1]:

- **Start:** `audit_pipeline.py` initializes logging.
- **Step 1:** Downloads Raw Data (`raw_loader`).
- **Step 2:** Unifies Schemas (`data_definitions`).
- **Step 3:** Filters Ghost Trips (`ghost_trip_filter`).
- **Step 4:** Imputes Missing Dec 2025 (`missing_value_handler`).
- **Step 5:** Maps Spatial Zones (`geo_mapping`).
- **Step 6:** Calculates Metrics (`fleet_analytics`).
- **Step 7:** Generates Charts (`chart_generator`).

- **Step 8:** Compiles PDF (`document_builder`).
- **End:** Pipeline finishes. Output available in `outputs/`.