SYNMAX DATA AGENT

Take-Home Coding Assignment — Final Report

September 2025

COMPANY: SynMax

Project Track: Data Agent for Tabular Analytics (CLI)

Shreyas Mysore Narayana

Abstract

I built a small command-line tool that loads a big table (CSV, Excel, or Parquet), lets me ask questions in plain English, and gives back short tables and quick notes. It can show counts, averages, trends by year, correlations, and basic anomaly checks. It runs fully on my laptop. If I turn on an optional 'insights' mode, it also prints a few careful bullets with possible reasons behind patterns.

1. Introduction

The SynMax take-home assignment requires a Python agent that can load the provided dataset at runtime and answer both deterministic and analytical questions via a conversational CLI. The system must handle pattern recognition (e.g., trends and correlations), anomaly detection, and plausible causal hypotheses framed with limitations.

Primary objectives:

1. Build a CLI agent that loads data from a local path or URL (CSV/XLSX/Parquet).
2. Implement preprocessing to coerce types, derive a `year` column from a specified date field, and summarize missingness.
3. Translate natural-language questions into a deterministic plan (group_by, ops, filters, sort/top-N, anomalies).
4. Execute the plan in Pandas and return concise, evidence-backed answers.
5. Optionally append 3–5 brief Insights bullets when an API key is available.

2. Literature Review

2.1 Natural-Language Data Agents

Natural-language interfaces for data analysis have evolved from keyword-based query builders to hybrid systems that combine rule-based planners with large language models. Deterministic planning ensures repeatability and control, while LLMs can improve coverage for ambiguous phrasing when used with strong guardrails.

2.2 Deterministic Query Planning

A plan schema (type, group_by, ops, filters, top-N) creates a stable contract between language parsing and execution. It avoids the brittleness of pure free-form prompts and preserves transparency of the steps taken.

2.3 DataFrame Analytics with Pandas

The current standard for tabular analytics in Python is still the Pandas DataFrame. A significant portion of exploratory analysis requirements are met by grouped aggregations, value counts, sorting, and correlation matrices. For huge data, Parquet with PyArrow offers quick input/output.

2.4 Anomaly Detection

Univariate outliers can be flagged using standardized z-scores; multivariate outliers can be identified using IsolationForest, which estimates anomality via random partitioning in feature space. These methods are interpretable starting points but should be validated against domain context to avoid false positives.

2.5 LLM Insights & Causality Caveats

Short, bullets can help summarize observations and propose plausible drivers while explicitly stating uncertainty. We stress that correlation does not imply causation; insights are framed as hypotheses with limitations and potential confusion makers.

3. Methodology and Tools

Environment: Python 3.10+ in a virtual environment; dependencies pinned in requirements.txt.

- Core: pandas, pyarrow (Parquet), numpy
- Analytics: scikit-learn (IsolationForest), scipy (optional)
- CLI & formatting: rich
- Optional insights: openai or anthropic SDKs (only if `--insights` is enabled)

Reproducible setup (PowerShell):

```
python -m venv venv
.\venv\Scripts\Activate.ps1
python -m pip install --upgrade pip
pip install -r requirements.txt
```

Run on dataset:

```
python -m src.agent --data-path "C:\path\to\pipeline_data.parquet" --date-col
eff_gas_day
```

Note: Change the data path as per your local system.

Enable insights (optional):

```
$env:OPENAI_API_KEY = "sk-..."
python -m src.agent --data-path "C:\path\to\pipeline_data.parquet" --date-col
eff_gas_day --insights
```

4. System Design & Techniques Used

4.1 Data Ingestion & Preprocessing

- Loaders: CSV/XLSX/Parquet from local path or URL (Drive links supported).
- Type inference: numeric and datetime coercion; drop fully null columns to improve speed.
- Date handling: derive `year` from a chosen date column (e.g., `eff_gas_day`).
- Missingness: summary table with counts and percentages.

4.2 Natural-Language Planning

- Plan schema: `type`, `group_by`, `ops`, `filters`, `top_n`, `ascending`.
- Deterministic mapping for phrases like "count rows by …", "sum/mean by …", "top N by … where …", "correlations", and "outliers".

4.3 Deterministic Execution Layer

- Aggregates via `groupby().agg()` with safe ops: count/sum/mean/median/min/max/std.
- Row counts either global or grouped (fast path for `{'*':'count'}`).
- Sorting + top-N with optional filters.
- Pearson correlation matrix for numeric columns.

4.4 Evidence Renderer

- Every answer includes: Plan (JSON-ish dict), Method (human label), Preview (up to 10 rows).
- This keeps outputs concise, auditable, and fast to grade.

4.5 Optional LLM Insights

- If enabled, generates 3–5 brief, caveated bullets grounded on column names/types and a small preview.
- Always reiterates that correlation ≠ causation; calls out missingness and potential confounders.

5. Results

5.1 Dataset Overview

Observed shape: 23,854,855 rows × 14 columns. Date column used for year derivation: `eff_gas_day`.

Year distribution (row counts):

year	row_count
2022	6,504,649
2023	6,664,545

2024	6,550,441
2025	4,135,220

Missing-value snapshot (selected columns):

column	missing	missing_pct
longitude	23,854,855	100.000000
latitude	23,854,855	100.000000
connecting_pipeline	18,662,066	78.231731
connecting_entity	2,975,770	12.474484
county_name	2,477,210	10.384511
state_abb	591,309	2.478778
scheduled_quantity	47,758	0.200202
category_short	33,259	0.139422
pipeline_name	0	0.000000

5.2 Representative Analyses

- Distributions: `value counts for state_abb top 10` highlights concentration across states.
- Aggregates: `sum/average scheduled_quantity by state_abb` and `by year` show drivers and trends.
- Correlations: `correlations` to inspect linear relationships among numeric columns.
- Anomalies: `outliers in scheduled_quantity` (z-score) and `find multivariate anomalies using isolation forest` (if enabled).

5.3 Performance

- Parquet with PyArrow provides fast IO; previews capped at 10 rows to reduce print overhead.
- Group-bys on low-cardinality columns (e.g., `state_abb`) are responsive even at large scale.

5.4 Quality Checks

- Derived `year` validated by `value counts for year top 10`.
- Missingness reviewed; fully null columns pruned for speed.
- Evidence block ensures transparency for graders.

6. General Discussion

6.1 If Starting Over

I would define a simple, typed plan schema first, then implement a thin planner around it. I would add a YAML-based type override for edge columns earlier and script a micro-benchmark for wide CSV vs Parquet.

6.2 Lessons Learned

- Deterministic plans reduce ambiguity and speed up iteration.
- A minimal but strict evidence block avoids over-verbose logs while keeping outputs auditable.
- Parquet (columnar) significantly improves load times over CSV for multi-million-row datasets.

6.3 With More Time

- Lightweight segmentation (KMeans centroids) with profile tables.
- Seasonality-aware trends if sub-annual timestamps are available.
- A tiny config file to persist human-approved type overrides.

6.4 Fulfillment of the Project

The agent meets core requirements: dataset-agnostic ingestion, NL planning, deterministic execution, patterns and anomalies, and evidence-backed answers. Optional LLM insights provide caveated hypotheses in line with the spec.

6.5 Findings & Limitations

- Patterns: year-over-year changes and state-level concentration of scheduled quantities.
- Limitations: correlation ≠ causation; partial-year effects (2025); missingness in geo fields; anomaly ≠ error.

7. Conclusion

The SynMax Data Agent delivers a robust, reproducible CLI for fast exploratory analytics at scale. It balances determinism (for grading and reliability) with optional LLM insights (for interpretability), and adheres to the assignment's constraints on inputs, outputs, and environment.

8. References

- Pandas Documentation (DataFrame, groupby, IO).
- PyArrow Documentation (Parquet IO).
- scikit-learn Documentation (IsolationForest).
- OpenAI / Anthropic SDK docs (if Insights enabled).

9. Appendices

9.1 Quick Start Commands

```
python -m venv venv
.\venv\Scripts\Activate.ps1
pip install -r requirements.txt
```

```
python -m src.agent --data-path "C:\path\to\pipeline_data.parquet" --date-col
eff_gas_day
```

9.2 Command Catalog

- Meta: dataset shape; list columns; show dtypes; head N; tail N; missing values; duplicate rows.
- Value counts: value counts for <col> top K.
- Aggregates: sum/mean/median/min/max/std <metric> by <group>.
- Row count: count rows by <group> | count rows.
- Sorting: top N rows by <metric> where <filter>.
- Correlation: correlations.
- Anomalies: outliers in <metric>; find multivariate anomalies using isolation forest.