Data Analyst Agent – Project Blueprint

Last updated: 2025-08-12 22:26 UTC

## 1) Purpose A **generic, prompt-first, parallel-processing** API agent that can **source, prepare, analyze, and visualize** data from uploaded files, web pages, or S3 parquet—returning **only the requested JSON** within **3 minutes**.

- **No hardcoding** for datasets. - **Uploaded data is primary**; scrape/duckdb run in **parallel** if links exist. - **Deterministic-first**; LLMs act as planners/coders/adjudicators.

## 2) API Contract **Endpoint**: `POST /api/` (multipart form-data) **Required**: `questions.txt` (always present) **Optional**: attachments (`.csv`, `.xlsx`, `.json`, images, etc.)

Response: - Default: **JSON array** with 1 element per question. - If the question explicitly requests an object schema, return that exact **object** shape.

Example ```bash curl "http://127.0.0.1:8000/api/" \ -F "questions.txt=@tests/question1.txt" \ -F "data.csv=@tests/edges.csv" ```

## 3) Output Rules - Exact **shape** decided by **Contract** stage (array by default). - **No envelopes** (no `ok/meta`), return **only** the payload. - Plots as `data:image/png;base64,...` **< 100kB**. - Type-checked per qtype: `count(int)`, `earliest(str)`, `corr(float -1..1)`, `scatter(png b64)`, `graph_*` (see pipeline).

## 4) Pipeline (high level) 1. **Contract** → decide output shape (array/object), normalized questions, and inferred qtypes. 2. **Inputs & Catalog** → load uploaded CSV/Excel/JSON; build data catalog. 3. **Link Planner** → extract HTTP and **S3** links; optionally ask LLM to pick **up to 4** canonical URLs; persist to `link_plan_*/links.txt`. 4. **Parallel Forks** (no early stop): - **Scraper** → fetch HTML, parse tables to CSVs (`scrape_run_*/scrape/`). - **DuckDB** → if `s3://` present, materialize parquet **LIMITed** samples to CSV (`duckdb_run_*/`). - **Deterministic (pass 1)** → compute answers from uploaded + any existing tables. - **LLM-on-snippet (opt)** → tiny CSV snippets for fast numeric answers. - **Codegen (opt)** → safe Python for plots/corr/graph; prints one JSON array. 5. **Deterministic (pass 2)** → re-run quickly after scraper/duckdb to consume fresh tables. 6. **Adjudication** → type-validate & merge per qtype (priority: det_p2 > det_p1 > codegen > llm_snippet). 7. **Emit** → return final payload only.

Time budget ≤ 175s overall (global SLA 180s).

## 5) Qtypes (examples) - **count**: integer ≥ 0 (supports currency thresholds & date filters). - **earliest**: string title/name (by year or metric threshold). - **corr**: Pearson correlation between two numeric columns. - **scatter**: PNG with dotted **red** regression line (<100kB). - **graph_***: `edge_count`, `highest_degree`, `average_degree`, `density`, `shortest_path`, `plot`, `degree_histogram` (requires edges CSV).

## 6) Intermediates (always written) - `link_plan_*/links.txt` and `link_plan_*/link_plan.json` - `scrape_run_*/scrape/table_*.csv|.md` - `duckdb_run_*/table_*.csv` - `deterministic_run_*/answers.json` - `adjudicator_run_*/candidates.json`, `final.json`

## 7) Env & Secrets - `OPENAI_API_KEY` (optional; enables LLM-assisted link selection & codegen/snippet tracks) - `OPENAI_MODEL` (default `gpt-4o-mini`) - `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN` (optional for protected S3) - `AWS_S3_ENDPOINT` (optional for S3-compatible stores)

## 8) Dependencies `fastapi uvicorn httpx beautifulsoup4 pandas numpy matplotlib pillow duckdb networkx`

## 9) Runbook ```bash python -m venv .venv . .venv/Scripts/activate # Windows PowerShell: .venv\Scripts\Activate.ps1 pip install -r requirements.txt uvicorn app.api:app --host 127.0.0.1 --port 8000 --reload ```

## 10) Design Tenets - **Generic over bespoke** (no dataset-specific code). - **Prompt-first** (decisions in prompts; code is thin, safe). - **Parallel all the time** (no early stop). - **Deterministic-first**, LLMs as planners/coders/adjudicators. - **Reproducible** (persist intermediates).

Parallel Pipeline

Last updated: 2025-08-12 22:26 UTC

## Stage 0 — Init - Create `workdir/run_<ts>/` and subfolders: `contract/`, `link_plan/`, `scrape/`, `duckdb/`, `snippets/`, `candidates/`, `final/`. - SLA guard: **hard cap 175s**.

## Stage 1 — Contract (format + qtypes) - Prompt: `prompts/format_decider.txt` - Output JSON envelope (illustrative): ``` { "output": {"type": "array", "keys": []}, "questions": ["Q1", "Q2", "..."], "qtypes": ["count", "scatter", "..."] } ```

## Stage 2 — Inputs & Catalog - Load uploaded CSV/Excel/JSON → profile columns (numeric/text hints). - Build `inputs/catalog.json`.

## Stage 3 — Link Planner - Extract HTTP + **S3** from `questions.txt` (regex). - If `OPENAI_API_KEY`, ask LLM to pick **up to 4** canonical links. - Persist **authoritative** list to `link_plan_*/links.txt`.

## Stage 4 — Forks (all parallel) - **Scraper** (HTTP): concurrent fetch → HTML → tables → CSV (`scrape_run_*/scrape/`). - **DuckDB** (S3): `read_parquet('s3://...') LIMIT 50k` → CSV (`duckdb_run_*/`). - **Deterministic Pass 1**: compute from uploaded + any existing tables. - **LLM-on-snippet (opt)**: tiny CSV + prompt → quick numeric answers. - **Codegen (opt)**: safe Python for corr/plots/graph; prints one JSON array.

## Stage 5 — Deterministic Pass 2 - If scraper/duckdb produced tables OR p1 had nulls → run fast pass-2.

## Stage 6 — Adjudication - Priority per answer position: **det_p2 > det_p1 > codegen > llm_snippet**. - **Type validation** by qtype (reject invalid types). - Output conforms exactly to contract shape (array | object).

## Budgets (example, ms) - Contract: 7000 - Inputs: 8000 - Link planning: 10000 - Scraper: 50000 - DuckDB: 35000 - Deterministic P1: 55000 - LLM Snippet: 25000 - Codegen: 45000 - Deterministic P2: 25000 - Adjudication: 20000 - **Total ≤ 175000 ms** (overlapping).

## Plot Constraints - PNG base64 data URI `< 100kB` - Visible axes; **dotted red** regression line for scatter.

## Graph Suite - Inputs: edges CSV with two columns (`source/target` auto-detected; fallback first two columns). - Outputs: `edge_count`, `highest_degree`, `average_degree`, `density`, `shortest_path`,

`plot`, `degree_histogram`.

Prompts Reference (summary)

Last updated: 2025-08-12 22:26 UTC

## format_decider.txt - Decide: output shape (array/object), normalized questions, and qtypes. - Default to array if not specified; no commentary; strict JSON.

## link_selector.txt - Given question + candidate URLs (+ uploaded preview), pick up to 4 authoritative links. - Prefer canonical sources; may add links not present in candidates. - Return strict JSON: `{"links": [], "notes": ""}`.

## schema_map_prompt.txt (optional) - Map semantic names (`rank, peak, year, title, gross, source, target`) → actual column headers. - Strict JSON: `{"column_map": {"rank": "Rank", ...}}` (null if not found).

## direct_on_snippet_prompt.txt (optional) - Input: tiny CSV snippet (≤500 rows, selected columns). - Answer strictly **from the snippet**; return JSON array; no plots.

## codegen_prompt.txt (optional) - Generate **safe Python** using whitelisted libs: pandas, numpy, matplotlib, pillow, networkx. - No network or file deletion; print exactly **one JSON array**; plots as data URI <100kB. - Handles qtypes: `count, earliest, corr, scatter, graph_*`.

## finalize_prompt.txt (optional) - As a tie-breaker if numeric candidates disagree; must return strict JSON array of same length, or pass-through.

Troubleshooting

Last updated: 2025-08-12 22:26 UTC

## Windows build errors (pandas/meson/vswhere) - Prefer official wheels: `pip install --upgrade pip` then `pip install pandas` (or pin to a wheel version). - Our requirements avoid `lxml`—`beautifulsoup4` with `html.parser` is used.

## PowerShell curl - Use `curl.exe` for multipart: `curl.exe "http://127.0.0.1:8000/api/" -F "questions.txt=@tests/question1.txt"`

## Null answers - Check `link_plan_*/links.txt` (were links planned?) - Check `scrape_run_*/scrape/*.csv` and `duckdb_run_*/table_*.csv` - Ensure **Deterministic Pass 2** ran (look for `deterministic_run_*/answers.json`).

## PNG too large - Deterministic compresses to `<100kB`. If still large, reduce point size or DPI.

## S3 access - Public buckets: only region is required (`s3_region=...` hint in question). - Private: set `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, optional `AWS_SESSION_TOKEN`. - Non-AWS S3: set `AWS_S3_ENDPOINT`.

## Exact output shape - Contract decides shape; orchestrator enforces **no extra elements/keys**. - If you see envelopes, ensure `api.py` returns `final_payload` directly.
