InsightDash

KPI Intelligence Platform

Owen

October 1, 2025

Contents

1 Introduction

InsightDash is an interactive analytics application that aligns curated SQL datasets, reusable Plotly Dash components, and optional Large Language Model (LLM) insights into a single, leadership-ready command centre. The project ingests KPI extracts from a SQL Server instance, harmonises them across business lenses (outlet performance, regional comparisons, growth trends), and renders multi-tab dashboards. Each chart can be summarised via Gemini-powered narratives, giving business users quantified explanations without leaving the UI.

2 System Architecture

The runtime is orchestrated from the Dash entry point (`app.py`) and flows as follows:

1. Configure structured logging and read environment-driven settings (database, LLM, feature switches).

2. Fetch per-tab datasets by executing templated SQL through SQLAlchemy engines.

3. Normalise and remap query outputs to match the figure contracts expected by each tab.

4. Initialise Dash layouts, set up shared colour palettes, and register interactive callbacks within `app.py`.

5. Serve interactive charts that support brushing, filtering, and multi-select synchronisation.

6. On demand, send chart data to Gemini for map-reduce style summarisation and cross-chart synthesis.

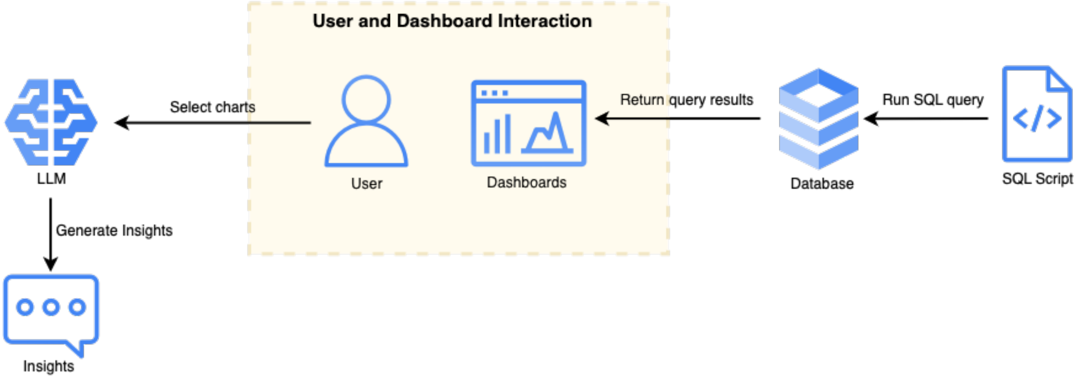
7. Persist usage logs and optional insight markdown while keeping the UI responsive.

2.1 Project Structure

```
InsightDash ............................ Root for the interactive analytics app
├── app.py ............................. Dash bootstrapper and callback wiring
└── app_tabs
    ├── tab1 ........................................... Outlet performance view
    │   ├── __init__.py
    │   ├── figures.py .................................... Plotly graph builders
    │   └── layout.py ..................................... Tab layout components
    ├── tab2 ........................................... Benchmarking scatter view
    │   ├── __init__.py
    │   ├── figures.py
    │   └── layout.py
    ├── tab3 ........................................... Trend and ranking view
    │   ├── __init__.py
    │   ├── figures.py
    │   └── layout.py
    └── tab_compare .................................... Outlet comparison tooling
```

```
│     ├── figures.py
│     └── layout.py
├── assets
│   ├── styles.css ........................ Global styling overrides
│   └── sidebar_controls.css .............. Resizable panel theming
├── config
│   ├── settings.py ....... Environment flags and connection URIs
│   └── logging.py ............................... Loguru sink setup
├── data_layer
│   ├── base.py ..................... Shared SQL execution helpers
│   ├── tab_1.py .................. Tab-specific query orchestration
│   ├── tab_2.py
│   └── tab_3.py
├── services
│   ├── llm.py .......................... Gemini client abstraction
│   ├── insights.py .................. Chunked summarisation logic
│   └── prompts.py ..................... Prompt builders for Gemini
├── sql_queries .............................. Templated SQL per tab
├── utils
│   ├── data.py .................. Deduplication and packing helpers
│   ├── colors.py ................. Palette utilities and colour maps
│   ├── dataframe.py ....... NaN handling and concatenation helpers
│   └── df_summary.py ........ Descriptive statistics for LLM payloads
├── logs ........................... Usage and diagnostic output
└── Documentation ................. Project documentation (this file)
```

2.2 Component Diagram



The primary components collaborate as follows:

1. **Dash UI**: Hosts the tabbed layout, callback graph, and controls.
2. **Data Layer**: Executes parameterised SQL templates and reshapes results for plotting.
3. **Services**: Provides LLM summarisation, prompt construction, and statistical preprocessing.
4. **Users**: Trigger filters, generate insights, and export findings.

5. **External Systems**: SQL Server supplies KPI tables; Gemini API enriches narratives when configured.

2.3 Data Sources

InsightDash targets the `cr_kpi` schema within SQL Server, expecting monthly tables (e.g., `kpi_april`, `kpi_may`) that expose outlet-level KPI metrics such as `rate_performance`, `rate_quality`, revenue shares, and intake volumes. Each data layer module wires its own SQL map:

- **Tab 1**: Outlet segmentation and region/category splits.
- **Tab 2**: Benchmark scatter plots with configurable axes and KPI filters.
- **Tab 3**: Time-sliced KPIs with ranking overlays.
- **Compare Tab**: Alias-based outlet lookup for side-by-side stat blocks.

3 Implementation

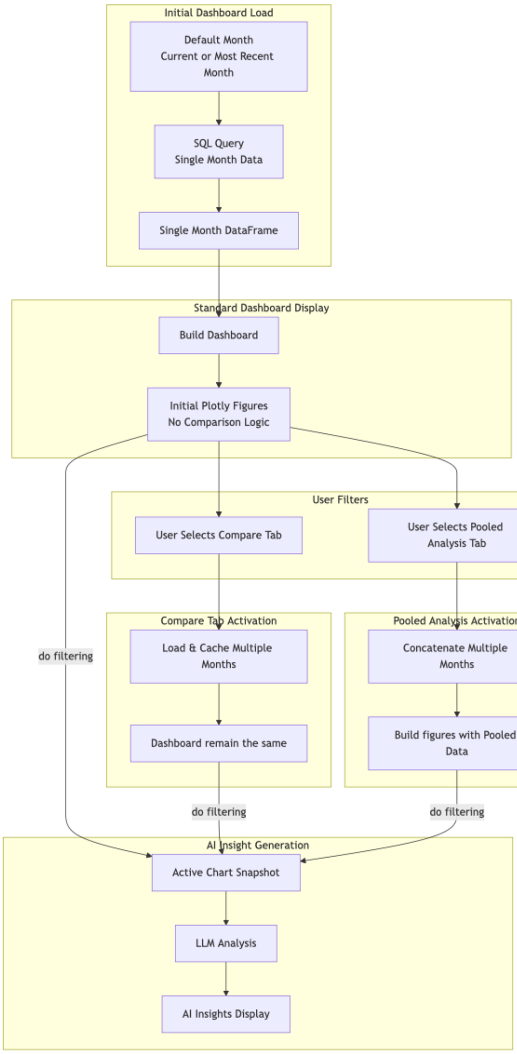
3.1 Runtime Algorithm

The Dash application follows the control flow summarised in Algorithm 1. Each stage is idempotent, enabling hot-reload in development and reliable deployments.

Algorithm 1 InsightDash Runtime Lifecycle

```
Require: Environment variables, SQL templates, Plotly Dash runtime
Ensure: Responsive dashboard with optional LLM summaries
CONFIGURE_LOGGING
data_tab1 ← GET_TAB1_RESULTS(”kpi_april”)
data_tab2 ← GET_TAB2_RESULTS(”kpi_april”)
data_tab3 ← GET_TAB3_RESULTS(”kpi_april”)
monthly ← LOAD_ADDITIONAL_MONTHS([”kpi_may”, ...])
app ← CREATE_DASHBOARD(data_tab1, data_tab2, data_tab3, monthly)
while app is running do
    Process callback inputs and recompute filtered DataFrames
    if user requests insight then
        slices ← CHUNK_DATAFRAME(selection)
        chunk_notes ← SUMMARIZE_CHART_VIA_CHUNKS(slices, context)
        markup ← SYNTHESIZE_ACROSS_CHARTS(chunk_notes)
        Update markdown panel with markup
    end if
    Append usage metrics to logs/usage.log
end while
```

3.2 Flow Chart



3.3 Key Modules

- `app.py`: Entry point that builds the Dash instance, injects tab layouts, and defines callback wiring for filters, graph selections, and insight generation.

- `data_layer/*`: Encapsulates SQL execution, result validation, and schema remediation so downstream charts always receive the expected columns.

- `app_tabs/*/figures.py`: Houses Plotly figure factories and table builders, isolating complex styling logic from callbacks.

- `services/insights.py`: Implements the map-reduce summarisation pipeline, including descriptive statistics to ground LLM outputs in verifiable numbers.

- `utils/colors.py` & `utils/data.py`: Provide consistent colour assignments, deduplication helpers, and DataFrame serialisation utilities shared across tabs.

- `assets/*.css`: Defines the visual identity (typography, spacing, panel controls) to keep charts and sidebars visually cohesive.

4 Key Features and Advantages

4.1 Interactive Cross-Filtering

Linked controls propagate selection state across tabs, allowing users to drill into outlet types, regions, and KPI ranges without reloading the app.

4.2 Stable Visual Identity

Colour maps and plot templates are centrally managed, so repeated visits guarantee consistent legend ordering and palette usage, aiding comparative analysis.

4.3 LLM-Assisted Narratives

When configured with a Gemini API key, business users can request per-chart insights or aggregated briefings that cite observed metrics and recommend next steps.

4.4 Operational Observability

Loguru sinks (stderr optional, file-based usage logs by default) capture database execution health, LLM token consumption, and user-triggered insight events.

5 Limitations and Risks

5.1 Database Connectivity

The dashboards rely on live SQL Server access. Network outages, credential drift, or schema changes will surface as empty charts despite defensive fallbacks.

5.2 LLM Configuration

LLM features degrade gracefully when `GOOGLE_API_KEY` is absent, but any misconfiguration (quota limits, model name changes) results in insight errors surfaced to the UI.

5.3 Data Schema Assumptions

Many figures expect specific KPI columns (e.g., `rate_performance`, `total_score`). Missing or renamed fields require corresponding updates to SQL templates and remap logic.

6 Related Technologies and References

- Plotly Dash documentation for component APIs and layout patterns.
- SQLAlchemy for engine creation and query execution against SQL Server.
- Google Gemini API for configuring the LLM insight features.
- Loguru for advanced logging patterns used by the project.

7 Frequently Asked Questions

- **How do monthly KPI tables map to dashboard tabs?**
  Each tab module builds a SQL map using the configured month (e.g., `kpi_april`). The keys (`q1`, `q2`, etc.) line up with figure factories that expect specific schemas.

- **Can I run InsightDash without Gemini?**
  Yes. The LLM services detect missing credentials and return informative fallback text while leaving all visual analytics functional.

- **What ensures colour consistency across tabs?**
  `utils.colors.color_map_from_list` generates deterministic palette assignments based on sorted category lists, and the results are cached per session.

- **Where are usage analytics stored?**
  Loguru writes structured entries to `logs/usage.log`, tagging events with `usage=True` so downstream tooling can aggregate adoption metrics.

- **How do I add a new KPI tab?**
  Scaffold a new folder under `app_tabs`, implement `layout.py` and `figures.py`, then wire new callbacks inside `app.py` (or a dedicated module) and extend `data_layer` to supply the required SQL results.