



Implementation Plan for ACE Quick Summary Feature

Overview

To give ACE (Autonomous Custom Intelligence) a more approachable, **Powerdrill-like** mode, we will create a *Quick Summary* pathway that delivers a concise dataset overview, basic statistics, simple visualisations and suggested questions. The new pathway should exist alongside the existing full intelligence pipeline, so users can choose between an in-depth report and a fast, easy-to-understand summary. The following plan outlines the steps required to design, implement, test and deploy this capability.

Phase 1: Requirements and Planning

1. **Define user personas** – Identify target audiences for the quick summary mode (e.g., analysts seeking a high-level overview, business users with minimal technical background). Clarify how this mode differs from the full report.
2. **Scope features** – Confirm the core capabilities to include: schema parsing, basic statistics, correlation analysis, suggested questions and simple charts. Deliberately exclude advanced modules (personas, clusters, deep narrative) to keep the scope focused.
3. **Review existing architecture** – Examine the current FastAPI backend and React/Vite front-end integration. Ensure the new functionality aligns with environment-variable-driven configuration ¹ and does not disrupt existing endpoints ².
4. **Security and compliance** – Verify that the summary pipeline respects existing file-size limits, data privacy policies and authentication/authorization mechanisms. Plan for robust error handling and user feedback.

Phase 2: Backend Implementation

1. **Extend the `/run` endpoint**
2. Update the endpoint's signature to accept an optional `mode` query parameter (default `"full"`). When `mode="summary"`, the request bypasses the full intelligence pipeline.
3. Validate that only supported modes are accepted; return a clear error message for invalid values.
4. **Implement data ingestion and schema parsing**
5. Leverage existing file-upload handling on the `/run` endpoint ² to read CSV/Excel content.
6. Infer data types (numeric, categorical, date/time) for each column. Use Python libraries such as **pandas** for robust type inference.
7. **Compute summary statistics**

8. For numeric columns, calculate count, mean, median, standard deviation, minimum and maximum values. For categorical columns, list the top categories and counts.
9. Detect and report missing values per column.
10. Compute simple pairwise correlations (e.g., Pearson correlation coefficients) between numeric columns.

11. Generate suggested questions

12. Based on the detected schema, create templated questions (e.g., "What is the distribution of X ?", "How does X relate to Y ?"). Use heuristics such as: every numeric column gets a distribution question; every numeric pair with a non-trivial correlation gets a correlation question.
13. Package these questions in the response so the front-end can display them in the sidebar.

14. Create `/runs/{run_id}/summary` endpoint

15. Persist the summary results (statistics, correlations, questions) using the same run ID mechanism as the full pipeline. This allows asynchronous polling and consistent state management.
16. Return a JSON payload structured as follows:

```
{
  "run_id": "abc123",
  "schema": [{"name": "age", "type": "numeric"}, ...],
  "statistics": {"age": {"count": ..., "mean": ..., ...}, ...},
  "correlations": [{"x": "age", "y": "height", "corr": 0.75}, ...],
  "questions": ["What is the distribution of age?", ...]
}
```

17. Expose this endpoint in the OpenAPI schema for discoverability.

18. Refactor common code

19. Extract shared functions (file parsing, basic stat computation) into reusable modules. This avoids duplication with existing analytics modules and facilitates unit testing.

Phase 3: Front-End Implementation

1. Upload interface and mode toggle

2. On the file-upload screen, add a checkbox or toggle labeled "**Quick View Mode**". When enabled, the front-end sends the `mode=summary` parameter to the `/run` endpoint.
3. Store the user's choice in local state so subsequent navigations (e.g., to the results page) can respect the selected mode.

4. Two-panel layout

5. Create a new page component (e.g., `SummaryView.tsx`) using React and your existing UI framework. Divide the page using CSS grid or Flexbox:
 - **Left sidebar:** displays dataset metadata (row/column counts, column types) and the list of suggested questions. Each question acts as a clickable item.
 - **Main content pane:** initially displays an overall summary card; when a question is clicked, this pane shows the corresponding chart and interpretation.
6. Ensure the sidebar is scrollable and collapsible for smaller screens; maintain sufficient white space and consistent typography ³.

7. Reusable summary cards

8. Build card components that encapsulate a chart, a short textual summary and a conclusion. Each card should accept props such as title, chart data, statistics and narrative.
9. For numeric distributions, display histograms or boxplots; for correlations, use scatter plots. Keep colour palettes harmonious and accessible ⁴.
10. Include bullet lists of key metrics where appropriate to aid quick comprehension ³.

11. Interactive controls

12. For correlation plots, provide dropdowns or selectors to switch the variables on the x- and y-axes. When the user selects a different pair, fetch or compute the relevant correlation metrics via the summary API.
13. Add hover tooltips for charts to display precise data values; ensure tooltips are keyboard-accessible ⁵.

14. Data fetching and state management

15. After initiating a summary run, poll the `/runs/{run_id}/summary` endpoint until the data is ready. Show a loading indicator during computation.
16. Once the summary is retrieved, store it in a context or state management layer (e.g., Redux or React Context) for easy access across components.

Phase 4: Integration and Configuration

1. Environment variables

2. Introduce a flag (e.g., `ACE_ENABLE_SUMMARY_MODE=true`) to enable or disable the quick summary feature at deployment time. This aligns with the existing practice of configuring API base URLs via environment variables ¹.
3. Expose the summary API URL through an environment variable (e.g., `VITE_ACE_SUMMARY_API_URL`).

4. Preserve existing workflows

5. The full report generation should remain the default when `mode` is not provided. Ensure that existing API clients and user interfaces continue to function without modification.
6. Where appropriate, reuse existing components (e.g., charting libraries) but encapsulate them so the summary view remains lightweight.

7. Backward compatibility

8. Version the API if necessary (e.g., `/v2/run`) to prevent breaking existing integrations. Provide migration notes in the documentation.

Phase 5: Testing and Quality Assurance

1. Unit tests

2. Write tests for schema inference, statistical computations and correlation calculations using representative sample datasets (including edge cases like missing values and mixed types).
3. Test question generation logic to ensure it produces sensible prompts for different schemas.

4. Integration tests

5. Use API testing frameworks (e.g., `pytest` with `starlette test client`) to verify that the `/run?` endpoint behaves correctly, including proper error responses for invalid inputs.

6. Test the end-to-end flow: uploading a file, polling the summary endpoint and rendering results.

7. Front-end tests

8. Implement unit tests for the new React components using **React Testing Library**. Verify that the toggle updates the run mode, the sidebar lists questions, and clicking a question updates the main pane.
9. Conduct manual usability tests to evaluate clarity, responsiveness and accessibility. Use tools like `axe-core` to check for accessibility issues ⁵.

10. Performance and security

11. Benchmark summary computation time on datasets of various sizes. Ensure the quick view remains responsive and does not block other backend tasks.
12. Validate that user data is not exposed to unauthorized parties; confirm that file uploads are properly sanitized.

Phase 6: Deployment and Documentation

1. Documentation updates

2. Update the main integration guide to describe the new `mode` parameter and summary endpoints, including example `curl` commands and expected responses ⁶.
3. Provide a user guide explaining how to select Quick View Mode and interpret the summary cards. Emphasize that the quick view offers a high-level overview, while the full report remains available for deeper analysis.

4. Release management

5. Deploy the backend changes behind a feature flag. Roll out the front-end components gradually, gathering user feedback.
6. Communicate the new capability to stakeholders and provide training sessions or demo videos to encourage adoption.

7. Monitoring and iteration

8. Monitor usage statistics (e.g., how often users choose Quick View Mode, which questions are clicked). Use this data to refine the question generator and chart templates.
9. Collect qualitative feedback from users about clarity, usefulness and desired enhancements. Iterate on the summary pipeline to improve accuracy and coverage.

Milestones and Timeline (Example)

Week	Milestone	Deliverables
1–2	Planning & Architecture	Requirements defined, API changes designed, UI wireframes created
3–4	Backend summarization	<code>/run</code> mode parameter implemented; summary computation functions completed; unit tests passing
5–6	Front-end summary view	Quick View toggle added; two-panel layout implemented with sample data; card components built
7–8	API integration	<code>/runs/{run_id}/summary</code> endpoint finalized; front-end connected to backend; integration tests passing
9	Testing & QA	Comprehensive test suite complete; performance benchmarks done; accessibility reviewed
10	Documentation & Launch	Updated docs published; environment variables configured; feature flag enabled in staging
11	Production Rollout	Deploy to production; monitor usage; collect feedback
12+	Iteration	Address feedback; refine question generator and visualisations

Additional Considerations

- **Design consistency:** Follow existing design guidelines for clarity, accuracy and aesthetics when creating new components [3](#) [4](#).
- **Accessibility:** Ensure all new UI elements are keyboard navigable and use colour palettes suitable for colour-blind users [5](#).
- **Extensibility:** Leave room for additional summary statistics (e.g., outlier detection) or future analysis modes.

This plan provides a comprehensive roadmap for integrating a Quick Summary feature into ACE. By extending the backend API, reusing and simplifying visualisation code, and implementing an intuitive two-panel UI, ACE can offer an accessible, Powerdrill-like experience while maintaining its powerful existing capabilities.

[1](#) [2](#) [6](#) INTEGRATION.md

<https://github.com/jashshah854-a11y/intelligent-insight-engine-9926c6ff/blob/170311c39eb582182da84296fdef71297f4b4385/INTEGRATION.md>

[3](#) [4](#) [5](#) Data visualization guide: principles and examples

<https://www.justinmind.com/blog/data-visualization-examples-principles/>