Comparative Analysis Dashboard - Project Documentation

Abstract

This document describes the Comparative Analysis Dashboard — a web application designed to collect, analyze, and visualize project cost metrics to compare Agile and Waterfall software development methodologies. The dashboard centralizes project data, runs statistical tests (e.g., Welch's t-test), computes effect sizes, and presents actionable recommendations to stakeholders. The project demonstrates a combination of modern frontend practices (React + TypeScript + Vite), cloud-backed persistence (Firebase Auth & Firestore), and statistical analysis tailored for project management decisions.

Introduction

Software development organizations continually evaluate delivery methodologies to determine which practices yield better outcomes for cost, schedule, quality, and team satisfaction. Agile and Waterfall represent two contrasting approaches. Agile emphasizes iterative delivery, customer feedback, and adaptability; Waterfall emphasizes linear planning, defined phases, and comprehensive upfront requirements. This dashboard helps teams make evidence-based decisions by aggregating historical project data, running hypothesis tests, and surfacing insights through charts and KPIs.

Project Objectives

- Provide a secure, multi-tenant web application for storing project-level cost and metadata.
- Offer comparative statistical analysis between Agile and Waterfall projects, including significance testing and effect size estimation.
- Present results using interactive visualizations (Chart.js) and downloadable reports (CSV/PDF export features).
- Enable easy data import (CSV) and basic admin capabilities to seed or manage sample data.
- Be developer-friendly: simple local setup using Node.js and Vite, with a modular codebase using React + TypeScript.

Methodologies

This section explains both the software engineering stack and the statistical methodologies used.

Software Stack and Architecture

- Frontend: React 18 + TypeScript. Vite is used for fast local development and production builds. Components are organized under `src/components` with pages under `src/pages`.
- Styling: Tailwind CSS provides utility-first styling; `index.css` contains Tailwind directives.
- Routing: React Router manages navigation across pages (Dashboard, Projects, Analysis, Settings).
- State & Context: `AuthContext` and `ProjectsContext` provide authentication state and project data across the app.
- Backend / Persistence: Firebase (Authentication and Firestore) is used for user management and storing project documents. Firestore stores project documents with timestamps for createdAt/updatedAt.

Statistical Methods

- Welch's t-test: Used to compare mean costs between Agile and Waterfall groups when variances may be unequal. Implemented in `src/utils/statistics.ts`.
- Cohen's d: Measures effect size to quantify the magnitude of difference between two groups.
- Confidence Intervals: Approximate 95% intervals around means or effect size estimates.
- Plain-English Interpretation: Results are translated into actionable recommendations (e.g., "Agile projects have significantly lower mean cost with a medium effect size").

Data Model

The `projects` collection in Firestore contains documents with the following structure:

Field	Type	Description	
name	string	Project name	
methodology	string	'Agile'	'Waterfall' 'Hybrid'
industry	string	Industry vertical	
size	string	Small/Medium/Large	
teamSize	number	Number of team members	
status	string	active/completed/archived	
plannedCost	number	Planned budget in USD	
actualCost	number	Actual cost in USD	
startDate	Timestamp	Project start date	
endDate	Timestamp	Project end date (optional)	

userId	string	Owner user id (sub)
createdAt	Timestamp	Record created timestamp
updatedAt	Timestamp	Record updated timestamp

Analysis & Discussion

This section provides guidance on how the dashboard produces analyses, how to interpret them, and example interpretations.

Typical Analysis Flow

1. Data collection: Projects are imported or entered manually. Each project must include `methodology` and cost fields.
2. Filtering & cohort selection: Analysts filter by industry, size, date range, or team size to compare similar projects.
3. Statistical test selection: The default test is Welch's t-test for mean differences. For non-normal or ordinal data, Mann-Whitney may be used (not implemented in the demo but the app provides an option in configuration).
4. Results: The dashboard computes summary statistics (means, variances), test statistic and p-value, effect sizes (Cohen's d), and 95% confidence intervals.
5. Interpretation: The application converts numeric outputs into plain-English statements and a recommended action (e.g., "Consider piloting Agile for small teams in Healthcare based on reduced rework costs").

Example Output and Interpretation (Hypothetical)

Suppose we compare `actualCost` between Agile (n=40) and Waterfall (n=35):

Comparative Analysis Dashboard - Project Documentation

Abstract

This document provides a comprehensive, technical and non-technical description of the "Comparative Analysis Dashboard" — a web application that collects project-level cost and metadata, performs statistical comparisons between development methodologies (Agile vs Waterfall), and presents results as interactive visualizations and plain-English recommendations. The documentation covers design rationale, architecture, data model, statistical methodology, UX flows, deployment, testing, security, limitations, and recommended next steps.

Table of Contents

Introduction

Organizations often face the question: which delivery methodology produces better outcomes for a given project profile? This dashboard provides a repeatable, data-driven process to compare cost, rework, overhead, and other financial metrics across projects that used Agile and Waterfall. By combining visualizations, hypothesis testing, and effect size measures, the dashboard helps stakeholders make empirically-grounded decisions.

Project Objectives

Primary objectives:

- Collect project-level cost and metadata in a secure, multi-tenant application.
- Provide cohort-based comparisons (e.g., compare small Agile vs small Waterfall projects in Finance).
- Implement robust statistical tests and interpret results in plain language.
- Offer import/export capabilities for CSV and PDF reporting.

- Keep the project lightweight and easy to run locally for evaluation and development.

Success criteria:

- Users can import project data and run an analysis within 5 minutes of setup.
- The analysis includes p-values, effect sizes, and confidence intervals, plus clear recommendations.
- The UI displays charts and tables, and provides CSV export of results.

Scope and Constraints

In-scope:

- Frontend-only application with Firebase for Auth and Firestore as the datastore.
- A statistical analysis engine implemented client-side (Welch's t-test, Cohen's d).
- CSV import functionality for bulk data ingestion.

Out-of-scope for v1:

- Server-side aggregation or heavy compute jobs.
- Advanced non-parametric tests (except where noted) and resampling methods (bootstrap/permutation tests).
- Enterprise-grade RBAC and SSO integrations (beyond Firebase Auth providers).

System Architecture

High-level overview:

- Client: React + TypeScript + Vite; components and pages are under `src/`.
- Backend: Firebase Auth (Users) and Firestore (projects, analysis results). No custom server in v1.
- Build & Dev: Vite for dev server and bundling.

Architecture diagram (logical):

- Browser (React SPA)
    - Auth flow (Firebase Auth SDK)
    - Firestore read/write (Projects, Analysis)
    - Analysis module (statistics.ts)
    - Charts (Chart.js)

Key files & responsibilities

- `src/lib/firebase.ts` — Initialize Firebase app using Vite env variables.
- `src/lib/firestoreService.ts` — Data-access layer for projects/analysis.
- `src/contexts/AuthContext.tsx` — Provides user state and helper auth methods.
- `src/contexts/ProjectsContext.tsx` — Loads/syncs projects for the signed-in user.
- `src/utils/statistics.ts` — Statistical functions (t-test, effect size, formatting).
- `src/pages/Analysis.tsx` — UI for running comparisons and visualizing results.

Data Model

Firestore collections used:

1. `projects` — stores project-level records.
2. `analysis` — stores saved analysis configurations and results.

`projects` document schema:

Field	Type	Example	Notes
name	string	"Website Redesign"	Project title
methodology	string	"Agile"	Agile/Waterfall/Hybrid
industry	string	"Finance"	Industry vertical
size	string	"Medium"	Size bucket
teamSize	number	8	Integer
status	string	"completed"	active/completed
plannedCost	number	120000	USD
actualCost	number	130000	USD
reworkCost	number	5000	Optional
overhead	number	2000	Optional
startDate	Timestamp	2024-01-01	Stored as Firestore Timestamp
endDate	Timestamp	2024-06-01	Optional

userId	string	"uid_abc123"	Owner UID
createdAt	Timestamp	Firestore server timestamp	
updatedAt	Timestamp	Firestore server timestamp	

Notes:

- Dates are stored as Firestore Timestamps; in UI they are formatted as YYYY-MM-DD.
- Numeric fields are stored as numbers (USD) for straightforward aggregation.

Statistical Methodology (Detailed)

This section explains each statistic produced by the application, the formulas used, and interpretation guidance.

1) Summary statistics

For any cohort (e.g., Agile projects where size=Small):

- n (sample size)
- mean = $\bar{x} = \frac{1}{n}\sum_{i=1}^n x_i$
- variance = $s^2 = \frac{1}{n-1}\sum_{i=1}^n (x_i - \bar{x})^2$
- standard deviation = $s = \sqrt{s^2}$

2) Welch's t-test for difference in means

Welch's t-test is used because it does not assume equal variances between groups. Given two samples (Agile and Waterfall):

- Test statistic:
  $ t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} $
- Degrees of freedom (Welch–Satterthwaite approximation):
  $ df = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{(s_1^2/n_1)^2}{n_1 - 1} + \frac{(s_2^2/n_2)^2}{n_2 - 1}} $
- p-value computed from t-distribution two-sided.

3) Cohen's d (effect size)

Cohen's d quantifies the magnitude of the mean difference:
$ d = \frac{\bar{x}*1 - \bar{x}2}{s{pooled}} $
where pooled standard deviation for unequal sample sizes is typically:
$ s{pooled} = \sqrt{\frac{(n_1 - 1) s_1^2 + (n_2 - 1) s_2^2}{n_1 + n_2 - 2}} $
Interpretation guidance:

- |d| < 0.2 — negligible
- 0.2 ≤ |d| < 0.5 — small
- 0.5 ≤ |d| < 0.8 — medium
- |d| ≥ 0.8 — large

4) Confidence intervals

Approximate 95% confidence intervals for mean difference use the t-distribution with computed df.

5) Assumptions & robustness

- Welch's t-test assumes samples are independent and approximately normally distributed; for moderate-to-large sample sizes (n > 30) the t-test is robust to non-normality.
- For small samples or heavily skewed data, the UI suggests using non-parametric tests or bootstrap methods (future work).

UX and User Flows

This section details key user flows and UI behavior.

Primary user personas

- Analyst: imports data, runs cohort comparisons, interprets results.
- Project lead: reviews dashboard visualizations to inform methodology decisions.
- Admin: seeds sample data and manages user accounts.

Key flows

1. Sign up / Sign in

    - Email/password or Google sign-in via Firebase Auth.
2. Import projects
    - CSV upload in `Projects` page. The importer validates required columns and shows a preview.
3. Run analysis
    - Choose cohorts (methodology, industry, size, date-range).
    - Choose metric (actualCost, plannedCost, reworkCost).
    - Click `Run Analysis`; the UI shows summary table, statistical results, charts, and a plain-English interpretation.
4. Export results
    - CSV export of results and PDF export of charts (future implementation notes included).

Analysis Workflow and Examples

This section gives step-by-step examples with hypothetical numbers and expected UI output.

Example: Comparing `actualCost` for Small projects in Finance

1. Filter: industry=Finance, size=Small, dateRange=2019-2024
2. Group by methodology: Agile vs Waterfall
3. Suppose results are:
    - Agile: n=25, mean=80000, sd=15000
    - Waterfall: n=20, mean=95000, sd=20000
4. Compute Welch's t-test → t, p-value
5. Compute Cohen's d → d ≈ -0.75 (medium-large)
6. UI shows chart with bars + error bars, a table with numeric results, and text: "Agile projects show a statistically significant lower actual cost with medium-to-large effect size; consider piloting Agile for similar projects."

Testing and Validation

Unit & Integration tests

- `src/utils/statistics.ts` should have unit tests for:
    - mean, variance, sd computations
    - t-test against known examples
    - Cohen's d calculation
- Integration: run the app, import the sample CSV, run analysis, and compare results to a known R/Python computation.

Manual QA checklist

- Import CSV with valid rows → projects appear correctly
- Run analysis with small sample sizes → ensure algorithm handles NaN/empty gracefully
- Try sign-in flows (email/password and Google)
- Ensure Firestore writes/reads are authorized and errors are surfaced in UI

Deployment and Environment

Local development

- Node.js 18+ and npm
- Commands:

```
npm install
npm run dev
```

The app will be served at `http://localhost:5173` by default.

Production

- Build with `npm run build` and host the `dist` directory on static hosting (Firebase Hosting / Vercel / Netlify) with environment variables set for production Firebase project.

Security and Privacy

- Authentication: Firebase Auth protects access to user data.
- Firestore rules: Use rules to ensure users can only access their own `projects` and `analysis` documents.
- Secrets: Service account keys must never be committed; store them securely and rotate if exposed.

Limitations and Assumptions

- This prototype runs statistics client-side; for large datasets or production workloads, move heavy computation to a backend.
- The statistical methods implemented are classical parametric tests — users should validate assumptions before acting.

Conclusion

The Comparative Analysis Dashboard is a lightweight, practical tool for exploratory analysis of Agile vs Waterfall project outcomes. It combines an accessible UI with rigorous statistical outputs and can be extended to include more tests, backend compute, or enterprise authentication.

Appendix: Sample Data & CSV Format

CSV header expected:

```
teamId,ownerEmail,projectName,methodology,industry,size,startDate,endDate,teamSize,plannedCost,actualCost,
reworkCost,overhead,defectCost,toolingCost,date,notes
```

Sample row:

```
team-001,owner@example.com,Website Redesign,Agile,Finance,Small,2023-01-01,2023-03-
15,6,50000,48000,2000,500,300,200,2023-03-16,Initial pilot
```