PROJECT PLAN: AI-POWERED TABLEAU DASHBOARD GENERATOR

Abhijit Ubale, Industry Advisory Board
abhijit.ubale@gmail.com

ABSTRACT

Build an AI-powered system that automatically generates professional Tableau dashboards from raw datasets. Using Azure OpenAI(or any other LLM of your choice/availability), you'll create a pipeline that analyzes data, recommends KPIs and visualizations, and produces publication-ready workbooks. This is a BUILD-FROM-SCRATCH project combining software engineering, data science, and AI/ML skills.

1 INTRODUCTION

PROBLEM: Data analysts spend 40% of time manually designing dashboards instead of analyzing. Creating dashboards requires expertise and takes weeks.

SOLUTION: Automate dashboard creation using AI to analyze data and recommend visualizations, charts, and KPIs.

LEARNING: You'll master modern software architecture, LLM integration, data engineering, and full-stack development—skills in high industry demand.

2 BASELINE OR INITIAL ANALYSIS

Weeks 1-2 GOALS: Understand the problem space and technology stack

Tasks:
□ Study similar systems: How Tableau, Power BI, Looker work
□ Analyze sample datasets: Understand quality issues, distributions, relationships
□ Learn key technologies:
  * Pydantic: Data validation and schema definition
  * Streamlit: Web UI framework
  * Langgraph: Workflow orchestration
  * Azure OpenAI: LLM API integration
  * Tableau format: XML structure of .twb files
□ Design your system architecture (draw diagrams)
□ Plan component interfaces and data flow

Deliverables:
- Architecture diagram showing component relationships
- Pydantic schema definitions for key data structures
- Technical notes on each component responsibility
- Sample EDA (exploratory data analysis) of test datasets

Weeks 3-8 GOALS: Build 8 core components. Each component should be independently testable.

3.1 DATA PROCESSING PIPELINE
Load, validate, and assess quality of CSV/Excel files.

Must do:
- Load data from files

- Detect column types (numeric, categorical, datetime)
- Assess quality: missing values, duplicates, outliers
- Generate profile report with statistics
- Validate data meets minimum requirements

Deliverables:
- DataProcessor class with methods: load(), validate(), assess_quality(), profile()
- Pydantic schemas: DatasetSchema, QualityReport, ProfileReport
- Unit tests covering normal cases and edge cases

3.2 AI ANALYSIS ENGINE
Use Azure OpenAI to analyze datasets and generate recommendations.

Must do:
- Setup Azure OpenAI API integration
- Design prompts to analyze dataset characteristics
- Generate structured recommendations (use Pydantic models)
- Recommend KPIs and metrics from the data
- Handle API failures gracefully (fallback to rule-based recommendations)
- Track API usage and costs

Deliverables:
- DashboardAnalyzer class with methods: analyze(), recommend_kpis()
- LLM prompt templates for different analysis types
- Error handling with retries and fallbacks
- Unit tests with mocked API responses

3.3 VISUALIZATION RECOMMENDATION ENGINE
Recommend chart types based on data characteristics.

Must do:
- Analyze data properties (dimensions, measures, cardinality)
- Apply heuristics to recommend visualizations:
  * 1 metric → Gauge, KPI card
  * 1 category + 1 metric → Bar chart
  * Time + metric → Line chart
  * 2 continuous → Scatter plot
  * Hierarchical → Treemap
  * Geographic → Map
- Ensure accessibility (color-blind friendly)
- Support 8+ visualization types

Deliverables:
- VisualizationRecommender class with recommendation rules
- Pydantic schemas: VisualizationSpec, RecommendationResult
- Tests covering diverse data scenarios

3.4 TABLEAU WORKBOOK GENERATOR
Convert recommendations into actual Tableau workbook files (.twb/.twbx).

Must do:
- Generate valid Tableau XML structure
- Create data source definitions
- Create worksheets with visualization specs
- Create dashboards assembling worksheets
- Generate Tableau calculated fields and formulas

- Export as .twb (XML) or .twbx (packaged)
- Validate generated files

Deliverables:
- TableauWorkbookGenerator class
- XML generation using lxml or ElementTree
- Pydantic schemas: WorkbookSpec, DashboardSpec, WorksheetSpec
- Tests: Generated files are valid Tableau workbooks

3.5 WORKFLOW ORCHESTRATION
Connect all components in a multi-step pipeline using Langgraph.

Must do:
- Define workflow state (data moving through pipeline)
- Create workflow nodes: validate → analyze → generate → finalize
- Implement conditional routing based on data type
- Handle errors and implement recovery
- Track progress and state
- Enable checkpoint/resume

Deliverables:
- DashboardGenerationWorkflow class using Langgraph
- State schema with Pydantic
- Tests: Complete workflow end-to-end

3.6 WEB USER INTERFACE
Build Streamlit app for user interaction.

Must do:
- File upload with validation
- Data preview and quality display
- Configuration: business goals, preferences
- Progress tracking during generation
- Download generated workbook
- Error messages and logging

Deliverables:
- Streamlit app with pages: Upload → Configure → Process → Results
- Reusable UI components
- Tests: File handling, state management

3.7 LOGGING & MONITORING
Add production-grade observability.

Must do:
- Structured logging (JSON format)
- Performance metrics (API latency, processing time)
- Error tracking and debugging
- Token usage monitoring
- User-friendly error messages

Deliverables:
- Logger configuration with context
- Performance decorators
- Monitoring dashboard (optional)

3.8 COMPREHENSIVE TESTING

Build test suite with unit, integration, and E2E tests.

Must do:
- Unit tests (60%): Test components in isolation, mock external APIs
- Integration tests (20%): Test component interactions
- E2E tests (10%): Full pipeline with real data
- Edge cases (10%): Empty data, missing columns, API failures
- Target: >80% code coverage

Deliverables:
- pytest configuration and fixtures
- Test data generators
- Mock API responses
- CI/CD pipeline (optional)

3 Final Analysis

BUILD STRATEGY

WEEK 1-2: Design & Learn
- Architecture diagram
- Technology research
- Schema definitions

WEEK 3: Data Layer
- DataProcessor implementation
- Tests

WEEK 4: AI Layer
- Analyzer with LLM integration
- Mocked API tests

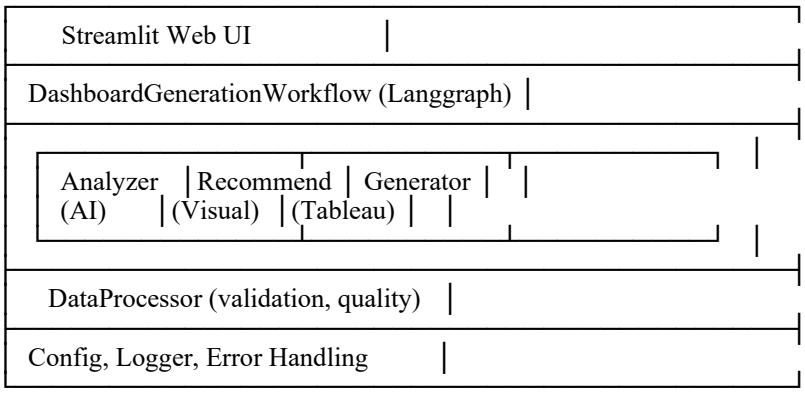
WEEK 5: Generation Layer
- Tableau XML generation
- Tests

WEEK 6: Orchestration
- Langgraph workflow
- End-to-end tests

WEEK 7: UI
- Streamlit interface
- Full pipeline tests

WEEK 8: Polish
- Logging & monitoring
- Final testing

ARCHITECTURE GUIDANCE
Build layered architecture:



Data flows: CSV → Process → Analyze → Recommend → Generate → TWB

You will decide:

1. ERROR HANDLING: How to handle LLM failures? Cache? Fallback rules?
2. TESTING STRATEGY: Unit vs integration test ratio?
3. RECOMMENDATION LOGIC: Rule-based? ML-based? Hybrid?
4. DATA VALIDATION: What makes data "good enough"?
5. ARCHITECTURE: Any deviations from suggested structure?

Document your choices and reasoning.

4 Final Goals & Evaluation

Your project is successful if:

FUNCTIONALITY:
□ Load CSV/Excel and generate valid Tableau workbooks
□ AI recommends sensible KPIs and visualizations
□ System handles errors gracefully
□ Generated workbooks open in Tableau Desktop

CODE QUALITY:
□ Components have clear responsibilities
□ Code is readable and well-documented
□ Error handling is comprehensive
□ Logging enables debugging
□ Type hints throughout (Pydantic models)

TESTING:
□ >80% code coverage
□ Unit tests for each component
□ Integration tests for workflows
□ Edge cases handled

DOCUMENTATION:
□ README with setup and usage
□ Architecture diagrams
□ Design decisions documented
□ Code comments where needed

5 Related Work

*Please cite research papers, reports, blog posts, projects, and other works you have consulted or plan to consult later. Please provide a sentence or two description that conveys the sources relevance to your project. You should have at least **5** sources.*

6 Data & Technical Requirements

Required:
- Python 3.10+
- pydantic (data validation)
- pandas, numpy (data processing)
- streamlit (web UI)
- langgraph (workflow orchestration)
- azure openai SDK (LLM integration)
- lxml (XML generation)

Recommended:
- scipy, scikit-learn (statistical analysis, outlier detection)
- pytest (testing)
- mypy (type checking)
- loguru (logging)