LLM-Driven Business Intelligence & Visualization Platform

Capstone Project Report

End-Semester Evaluation

Submitted by:

102203023	Siddhant
102203091	Sanchit Nanda
102203835	Udit Gupta
102203684	Swapnil Gumber
102203313	Aryan Chharia

BE Fourth Year- COE/ CSE
CPG No. 174

Under the Mentorship of

Dr. Jasmeet Singh
(Assistant Professor-II)
&
Dr. Jatin Bedi
(Assistant Professor-II)

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department
Thapar Institute of Engineering and Technology, Patiala
December 2025

ABSTRACT

In today's data-driven economy, organizations generate vast amounts of data that hold valuable insights for strategic decision-making. However, extracting these insights remains complex, time-consuming, and often requires technical expertise in data science and business intelligence tools. Traditional BI platforms like Tableau and Power BI provide visualization capabilities but demand manual configuration and advanced technical skills, creating barriers for non-technical professionals and limiting the accessibility of data-driven decision-making.

This project introduces an LLM-Driven Business Intelligence and Visualization Platform, an AI-powered solution that automates data visualization, trend identification, and insight generation. By integrating Large Language Models (LLMs) with intelligent analytics, the platform simplifies data interpretation through a conversational interface, removing technical barriers and making business intelligence more intuitive, scalable, and efficient across various industries.

The platform automatically generates diverse visualizations including bar charts, line graphs, scatter plots, and heatmaps from user-uploaded datasets without manual configuration. It leverages LLMs to analyze data trends and provide natural language insights, enabling users to understand patterns and extract actionable business intelligence. A conversational AI chatbot interface allows users to interact with the system through natural language queries, facilitating data exploration without requiring coding or statistical knowledge.

The methodology involves a multi-stage pipeline that parses uploaded CSV or Excel files, performs exploratory data analysis to detect trends and anomalies, generates Python visualization code through fine-tuned LLMs, and produces business recommendations in natural language. The system incorporates reinforcement learning for improved context-awareness and continuous enhancement based on user feedback.

This solution addresses the pressing need for a unified platform that merges automated visualization generation with comprehensive business intelligence, particularly beneficial for SMEs lacking dedicated analytics teams. The platform finds applications across finance, healthcare, retail, and education sectors, enabling faster and more accessible data-driven decision-making while bridging the gap between raw data and strategic insights.

DECLARATION

We hereby declare that the design principles and working prototype model of the project entitled "LLM-Driven Business Intelligence & Visualization Platform" is an authentic record of our own work carried out in the Computer Science and Engineering Department, TIET, Patiala, under the guidance of Dr. Jasmeet Singh and Dr. Jatin Bedi during 7th semester (2025).

Date: 22-12-2025

Roll No.	Name	Signature
102203023	Siddhant	
102203091	Sanchit Nanda	Sanchit Nanda
102203835	Udit Gupta	Udit
102203684	Swapnil Gumber	Swapnil .
102203313	Aryan Chharia	Aryanchharia

Counter Signed By:

Mentor: Jasmeet Singh

Co-Mentor:

Faculty Name: Dr. Jasmeet Singh

Designation: Assistant Professor II
Computer Science & Engineering
Department TIET, Patiala

Faculty Name: Dr. Jatin Bedi

Designation: Assistant Professor II
Computer Science & Engineering
Department TIET, Patiala

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our mentors Dr. Jasmeet Singh and Dr. Jatin Bedi for their invaluable guidance and support throughout this capstone project. Their expertise in artificial intelligence and data science has been instrumental in shaping our understanding of LLM-driven business intelligence systems. They have been exceptional mentors who provided continuous technical knowledge and insights that were crucial for the successful development of our platform.

We are deeply thankful to Dr. Neeraj Kumar, Head, Computer Science and Engineering Department, for providing us with the necessary resources and academic environment to pursue this innovative project. We also extend our appreciation to the entire faculty and staff of the Computer Science and Engineering Department who contributed to our learning journey through their expertise in machine learning, natural language processing, and software engineering.

Lastly, we would like to thank our families for their unwavering love, encouragement, and support throughout this academic endeavor. Their belief in our capabilities and understanding during the intensive development phases of this project motivated us to deliver our best work. We are grateful for their patience and the sacrifices they made to support our educational pursuits.

Date: 22-12-2025

Roll No.	Name	Signature
102203023	Siddhant	
102203091	Sanchit Nanda	Sanchit Nanda
102203835	Udit Gupta	Udit
102203684	Swapnil Gumber	Swapnil .
102203313	Aryan Chharia	Aryanchharia

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BI	Business Intelligence
SME(s)	Small and Medium Enterprises
LLM	Large Language Model(s)
EDA	Exploratory Data Analysis
CSV	Comma-Separated Values
SQL	Structured Query Language
UI	User Interface
API	Application Programming Interface
DB	Database
AI	Artificial Intelligence
ML	Machine Learning
NLP	Natural Language Processing

INTRODUCTION

Data-driven decision-making has become central to modern organizations, yet the increasing scale and complexity of datasets have created a widening gap between data availability and actionable interpretation. While traditional visualization tools offer extensive capabilities, they often demand proficiency in programming and statistical reasoning, which restricts adoption by non-technical users and limits timely insight generation. Simultaneously, business environments require faster and more accessible analytical workflows that can convert raw data into meaningful insights without steep learning curves or dependence on expert analysts. Motivated by these challenges, this project introduces a system designed to simplify data interpretation by combining visualization, automated insight generation, and recommendation capabilities within a unified framework accessible through both web and conversational interfaces.

1.1 Project Overview

The project aims to address existing limitations in data analytics by developing an LLM-driven visualization and insight generation tool that democratizes access to business intelligence. The system enables users to upload datasets in CSV format, automatically extract contextual information, generate business insights, and render interactive visualizations through a modern web interface. Built using Express and Fast API, with a React and Plotly-enabled frontend deployed on Render, the platform integrates dataset ingestion, automated chart recommendations, and insight interpretation without requiring programming expertise. MongoDB is employed for dataset and chat storage, while JWT-based authentication ensures secure access and supports multi- organization and team structures, reflecting real-world collaborative analytics environments. To enhance accessibility and support conversational workflows, a Telegram bot interface extends system capability beyond browsers, enabling insight interaction through a familiar messaging platform.

Unlike traditional tools that necessitate manual selection of appropriate visualizations, the system incorporates a recommendation mechanism that analyzes dataset samples and suggests suitable chart types and insight categories. The cloud-based LLM interprets dataset headers and initial rows to propose insight directions, while bandit learning allows iterative refinement of output quality using user feedback in the form of approval or disapproval signals.

The deployed version currently supports datasets up to approximately 5000 rows, though larger datasets, such as the Thapar admissions dataset (>60,000 rows and 180+ columns), were utilized during development to validate performance and demonstrate scalability. Overall, the project seeks to provide a unified solution that bridges raw data and meaningful interpretation through accessibility, automation, and collaboration.

1.1.1 Technical Terminology

To contextualize the system's operation, several technical terms require clarification:

- **Large Language Model (LLM):** A neural language system capable of interpreting dataset context and generating analytic insights and visualization suggestions.
- **Interactive Data Visualization:** The representation of data using graphical elements, where user interaction supports exploration. Plotly is used for rendering.
- **Business Insight Generation:** The process of identifying trends, correlations, and patterns within datasets to aid decision-making.
- **Recommendation Engine:** A module that automatically suggests insight categories, chart types, and relevant column combinations.
- **JWT Authentication:** A token-based mechanism used to verify identity and maintain secure user sessions across organizations and teams.
- **Bandit Learning:** A feedback-based optimization process where model outputs are iteratively improved using positive or negative user responses.
- **Multi-Organization Structure:** A hierarchical storage model that separates datasets and chat histories across teams for collaborative workflows.
- **Telegram Bot Interface:** A conversational access method enabling dataset querying and insight generation through messaging.

1.1.2 Problem Statement

Even though large datasets are increasingly accessible, users without programming or statistical expertise face substantial difficulty in deriving meaningful insights from them. Conventional business intelligence platforms require manual configuration and technical skills to select appropriate visualization types and interpret the results effectively.

Such dependence restricts adoption among non-technical stakeholders and slows the decision-making process. Existing tools also lack automated recommendation systems and do not support intuitive conversational interaction or collaborative access aligned with organizational structures. Consequently, there is a need for a scalable and accessible analytics system that automates visualization, interprets insights, and reduces technical barriers without compromising functionality.

1.1.3 Goal

The primary goal of this project is to develop an accessible data visualization and insight generation platform that minimizes the requirement for domain knowledge while supporting meaningful analytics. The system is intended to allow users to upload datasets, receive automated insight recommendations, visualize data interactively, and collaborate within organization-based structures. Additionally, it aims to extend accessibility through a conversational bot interface and continuously improve insight quality using feedback-driven learning mechanisms.

1.1.4 Solution

The solution integrates a web-based visualization interface, LLM-driven insight generation, automated recommendation logic, dataset storage through MongoDB, and secure authentication using JWT within a unified architecture. Express and Fast API services process datasets and enable reuse of analytical components after transitioning from an earlier Streamlit prototype to a more scalable and interactive system. Plotly supports intuitive visualization, while the LLM interprets dataset samples to propose insights and visualization strategies. The inclusion of a Telegram bot provides an alternative interaction channel for conversational analytics. Bandit learning incorporates user feedback to refine insight quality over time, and the deployment strategy ensures accessibility without reliance on local hardware resources. Together, these components address the identified challenges by enabling non-technical users to analyze datasets and generate business insights efficiently.

1.2 Need Analysis

The need for an integrated, automated data visualization and business intelligence system arises from structural limitations in existing analytics workflows and tools. The following points summarize the key factors motivating this work.

Dependence on Multi-Stage Manual Analytics Workflows

Modern data visualization tools such as Tableau, Power BI, QlikView, and Looker do not operate as end-to-end analytical solutions. Before visualizations can be created, datasets must undergo complex processes which includes understanding data structure, identifying missing values, detecting outliers, and examining relationships among variables. This step requires trained data analysts and statistical expertise. Visualization tools assume that such preprocessing and analytical reasoning have already been performed externally, leading to fragmented workflows and increased manual effort.

Manual Insight Identification and Visualization Design

Business analysts must manually determine which insights are meaningful, how they can be derived, and which visualization techniques are appropriate. Each visualization must be individually designed, configured, and validated. This manual chart creation process is repetitive, time-consuming, and prone to subjective bias. Existing tools provide visualization capabilities but do not automate insight discovery, visualization reasoning, or justification of design choices.

Separation of Roles in Analytics Pipelines

In practical enterprise settings, analytics workflows are distributed across multiple roles. Data analysts perform EDA, visualization specialists design dashboards, and business analysts interpret visual outputs to derive actionable business intelligence. This role separation introduces delays, communication gaps, and additional costs. The lack of a unified system capable of performing visualization generation, and business insight extraction increases organizational dependency on specialized teams.

High Cost of Business Intelligence and Visualization Infrastructure

Industry reports consistently indicate that organizations spend **15–25% of their total IT budgets** on data analytics, visualization, and business intelligence infrastructure. Large enterprises often invest **millions of dollars annually** in BI software licenses, analytics teams, and dashboard maintenance. Despite these investments, adoption rates remain limited due to tool complexity, and insights are often generated reactively rather than proactively.

Limited Accessibility for Micro, Small, and Medium Enterprises (MSMEs)

Micro, small, and medium enterprises face significant barriers in adopting advanced analytics solutions. Unlike large enterprises, MSMEs cannot afford dedicated teams comprising data analysts, visualization experts, and business analysts. As a result, they rely on basic spreadsheet-based analysis, which restricts their ability to uncover complex insights. An automated, LLM-driven system that performs visualization generation and insight interpretation can provide MSMEs with enterprise-level analytical capabilities at a fraction of the cost.

Need for Automation Using Large Language Models

Recent advances in Large Language Models demonstrate strong capabilities in reasoning, natural language understanding, code generation, and explanation synthesis. These capabilities create an opportunity to automate the traditionally manual stages of analytics workflows. By leveraging LLMs, it becomes feasible to unify visualization recommendation, chart generation, and business insight extraction within a single system. This needs to reduce manual intervention, cost, and expertise dependency forms the central motivation for this project.

1.3 Research Gaps

A review of recent academic literature and leading research surveys, as well as ongoing advancements in AI-assisted analytics, reveals several unresolved gaps that motivate the development of an automated, end-to-end analytical system. These gaps span theoretical limitations, practical integration issues, and unmet needs for automation in data workflows.

Gap 1: Absence of End-to-End Analytics Systems

While research has explored components of the analytics pipeline-such as visualization surveys, automated chart recommendation, or natural language data querying-no existing system fully integrates key stages including exploratory data analysis (EDA), visualization generation, business insight extraction, and recommendation within a single automated framework. Prior work often focuses on individual elements of the analytical workflow but does not extend these to holistic, production-grade systems capable of autonomously navigating all major analytical phases from raw data ingestion to insight delivery [12].

Gap 2: Limited Automation of Exploratory Data Analysis

Most BI and visualization platforms place the burden of EDA on human analysts. Although AI-based frameworks have been proposed to automate parts of EDA, these efforts have primarily focused on preprocessing and pattern detection rather than integrating autonomous EDA into a broader analytical pipeline that includes downstream visualization and interpretation [21]. These systems often stop short of offering comprehensive, context-aware reasoning that connects initial exploratory steps with actionable visualization recommendations [22].

Gap 3: Weak Integration of Visualization and Business Intelligence Reasoning

Current data visualization research tends to emphasize chart correctness, interaction design, and graphical representation rather than translating visual patterns into business insight - such as causal interpretation, trend analysis, comparative evaluation, or strategic recommendations. Visualization tools generally provide representations of data but lack mechanisms for synthesizing visuals into structured insight narratives or business intelligence reasoning, leaving this task to domain experts [3].

Gap 4: Fragmented Tooling and Workflow Inefficiencies

Analytics workflows are frequently disjointed, requiring separate tools for data processing, visualization, BI reporting, and analysis. This fragmentation increases operational complexity, delays analytical cycles, and creates inefficiencies as users must move between platforms or roles to achieve integrated insights. Existing literature has not sufficiently addressed the design of unified, LLM-driven platforms that replace these fragmented chains with cohesive, automated workflows that adapt to analysis context [10].

Gap 5: Insufficient Context-Aware Recommendation Systems

Although visualization recommendation research has advanced, most systems fall short in adapting recommendations based on evolving analytical context, user interactions, or previously explored patterns. There is limited research on dynamic, context-aware recommendation engines powered by LLMs that can suggest “what to analyze next,” explain why a particular visualization is effective, and tailor recommendations as insights accumulate [4].

Gap 6: Lack of Integrated Support for MSMEs

The majority of BI and data visualization research focuses on large enterprise settings where dedicated analytics teams and premium BI platforms are available. There is comparatively little emphasis on developing low-barrier systems that support Micro, Small, and Medium Enterprises (MSMEs) by reducing dependency on skilled analysts and expensive infrastructure. Addressing this gap is critical to democratizing analytics, expanding inclusion, and enabling data-driven decision-making across a broader spectrum of organizations [14].

1.4 Problem Definition and Scope

Problem Statement: Organizations across industries generate vast amounts of data daily, but extracting meaningful insights remains a complex, time-consuming process that requires specialized technical expertise. Non-technical users struggle with traditional BI tools, leading to delayed decision-making, underutilized data assets, and suboptimal business outcomes. There is a critical need for an AI-powered solution that can automate data analysis, generate intuitive visualizations, and provide actionable business insights through a user-friendly interface accessible to users regardless of their technical background.

Project Scope: The LLM-Driven Business Intelligence and Visualization Platform encompass the following key components:

In Scope:

- Automated data ingestion and preprocessing for CSV and Excel formats
- Dynamic visualization generation including line charts, bar graphs, scatter plots, etc.
- Natural language insight generation with business recommendations
- Conversational AI interface supporting complex analytical queries
- Adaptive learning system with reinforcement learning capabilities
- Multi-industry applicability with domain-specific optimization
- Mobile application development

Out of Scope:

- Real-time streaming data processing from live databases
- Integration with proprietary enterprise software systems
- Advanced machine learning model training and deployment
- Multi-user collaboration and sharing capabilities
- Data security and encryption beyond standard practices
- Support for unstructured data formats (images, videos, audio)

1.5 Assumptions and Constraints

Key Assumptions:

1. **Data Quality and Format:** It is assumed that users will upload pre-processed, structured datasets in CSV or Excel formats with appropriate column headers and consistent data types. The platform assumes that data cleaning and basic preprocessing have been performed prior to upload.
2. **User Technical Literacy:** While the platform is designed for non-technical users, we assume basic computer literacy and familiarity with web-based interfaces. Users are expected to understand fundamental business concepts and be able to interpret visual representations of data.
3. **Internet Connectivity:** The platform assumes reliable internet connectivity for cloud-based processing and real-time interaction with LLM services. Adequate bandwidth is assumed for handling data uploads and visualization rendering.
4. **Data Privacy Compliance:** We assume that users have proper authorization to upload and analyze the datasets they provide, and that all data complies with relevant privacy regulations such as GDPR or CCPA.
5. **Hardware Resources:** The platform assumes access to cloud computing resources capable of handling LLM processing requirements and concurrent user requests without significant performance degradation.

Technical Constraints:

1. **Dataset Size Limitations:** Initial implementation will support datasets up to 50MB in size with maximum 100,000 rows to ensure optimal performance and response times.
2. **LLM Processing Limitations:** Response generation times may vary based on query complexity and current system load, with expected response times ranging from 5-30 seconds for complex analytical queries.
3. **Visualization Complexity:** The platform will focus on standard statistical and business visualization types, with complex custom visualizations beyond the initial scope.
4. **Language Support:** Initial implementation will primarily support English language interactions, with multilingual support planned for future versions.
5. **Integration Limitations:** The platform will operate as a standalone solution without direct integration to existing enterprise systems or databases in the initial version.

1.6 Standards

The development and implementation of the LLM-Driven Business Intelligence and Visualization Platform adhere to established industry standards and best practices:

Data Standards:

- ISO/IEC 25012 for Data Quality Models
- CSV RFC 4180 specifications for data format compatibility
- Unicode UTF-8 encoding standards for international character support

Software Engineering Standards:

- IEEE 830 for Software Requirements Specifications
- ISO/IEC 12207 for Software Lifecycle Processes
- Agile development methodologies following Scrum framework principles

Web Development Standards:

- W3C HTML5, CSS3, and JavaScript standards
- RESTful API design principles
- WCAG 2.1 accessibility guidelines for inclusive user interface design

AI/ML Standards:

- IEEE 2857 for Privacy Engineering in Artificial Intelligence and Machine Learning
- ISO/IEC 23053 for Framework for AI Risk Management
- Responsible AI principles for ethical LLM deployment

Security Standards:

- OWASP Top 10 security practices
- ISO 27001 information security management guidelines
- HTTPS/TLS encryption for data transmission

1.7 Approved Objectives

Based on the proposal evaluation by the academic panel, the following objectives have been approved for implementation:

Primary Objectives:

1. **Automate Data Visualization and Analysis:** Develop a comprehensive system capable of automatically generating diverse visual representations including line charts, bar graphs, scatter plots, heatmaps, and statistical distributions based on uploaded datasets, significantly reducing manual effort in data exploration and analysis.
2. **Enhance Business Intelligence with AI-Driven Insights:** Integrate Advanced Large Language Models to analyze datasets and provide clear, contextual, and actionable insights in natural language format, enabling users to make informed data-driven decisions without requiring specialized technical expertise or statistical knowledge.

3. **Enable Conversational Data Exploration:** Implement an intuitive AI-powered chatbot interface that facilitates natural language interactions with data, allowing users to refine analysis, identify trends, extract deeper insights, and ask follow-up questions dynamically without programming knowledge.
4. **Develop Adaptive Intelligence for Smarter Decision-Making:** Create a scalable and user-friendly platform that continuously improves through user interaction, implementing reinforcement learning mechanisms in chat interactions to refine responses and make analytics more intuitive and accessible for both technical and non-technical users across various industries.

1.8 Methodology

The development methodology for the LLM-Driven Business Intelligence and Visualization Platform follows a systematic, multi-phase approach that ensures robust functionality, user-centric design, and scalable architecture.

Phase 1: Data Processing and Analysis Pipeline The methodology begins with establishing a comprehensive data processing pipeline. Users upload datasets through a secure web interface, where the system automatically parses CSV and Excel files using robust libraries such as Pandas. The platform performs initial data validation, type inference, and statistical computation to generate preliminary insights including mean, median, standard deviation, and correlation matrices.

Phase 2: LLM-Driven Code Generation Based on the EDA findings and data characteristics, the system constructs detailed prompts that instruct fine-tuned Large Language Models to generate Python visualization code. The LLM component analyses data patterns and user requirements to select appropriate visualization types and styling parameters. Generated code utilizes libraries such as Matplotlib, Seaborn, and Plotly to create publication-quality visualizations.

Phase 3: Secure Code Execution and Visualization Rendering Generated Python code executes within a secure sandbox environment to prevent security vulnerabilities while maintaining system integrity. The execution engine renders visualizations and returns structured outputs that integrate seamlessly with the web interface. Error handling mechanisms ensure system stability.

Phase 4: Insight Generation and Business Intelligence The platform leverages LLM capabilities to analyze visualization outputs and statistical findings, generating natural language insights that translate technical analysis into actionable business recommendations. This phase incorporates domain-specific knowledge and industry best practices to ensure relevance and practical applicability of generated insights.

Phase 5: Conversational Interface Implementation The conversational AI component processes user queries in natural language, maintaining conversation context and enabling iterative analysis refinement. The chatbot interface supports complex analytical requests, follow-up questions, and dynamic visualization modifications based on user feedback and requirements.

Phase 6: Adaptive Learning and Optimization The system implements supervised fine-tuning and reinforcement learning mechanisms to continuously improve performance based on user interactions and feedback. The learning component analyses user satisfaction, query success rates, and insight relevance to optimize LLM responses and recommendation quality over time.

1.9 Project Outcomes and Deliverables

The LLM-Driven Business Intelligence and Visualization Platform will deliver comprehensive capabilities that transform how organizations interact with and extract value from their data assets.

Primary Deliverables:

1. Automated Visualization Platform A fully functional web-based platform capable of generating diverse, professional-quality visualizations automatically from uploaded datasets. The system will support multiple chart types including bar charts, line graphs, scatter plots, heatmaps, distribution plots, and correlation matrices without requiring manual configuration or technical expertise from users.

2. Natural Language Insight Engine An advanced AI-powered analysis engine that interprets data patterns, identifies significant trends, and generates clear, actionable business insights in natural language format. The engine will provide contextual explanations of data findings, highlight anomalies, suggest optimization opportunities, and recommend strategic actions based on analytical results.

3. Conversational AI Interface An intuitive chatbot system that enables users to interact with their data through natural language queries, supporting complex analytical requests, iterative refinement of analysis parameters, and dynamic exploration of data relationships without requiring coding skills or statistical knowledge.

4. Adaptive Learning System A sophisticated feedback mechanism that continuously improves system performance through reinforcement learning, user interaction analysis, and adaptive response optimization, ensuring that the platform becomes more effective and user-centric over time.

Expected Outcomes:

Enhanced Decision-Making Efficiency: The platform will significantly reduce the time required for data analysis from hours or days to minutes, enabling faster response to market changes and business opportunities. Users will be able to generate comprehensive analytical reports and visualizations instantly upon data upload.

Democratized Data Analytics: By removing technical barriers, the platform will make advanced analytics accessible to non-technical professionals across departments, enabling widespread adoption of data-driven decision-making practices throughout organizations regardless of their size or technical resources.

Improved Analytical Accuracy: Automated analysis reduces human error in data interpretation while LLM-driven insights provide comprehensive coverage of potential findings that manual analysis might overlook, leading to more thorough and accurate business intelligence.

Cost Reduction: Organizations will reduce dependency on specialized analytics personnel for routine data analysis tasks, resulting in significant cost savings while enabling internal teams to focus on strategic initiatives and advanced analytical projects.

1.10 Novelty of Work

The novelty of this project lies in its integration of recommendation system, automated insight generation, chart recommendation, and multi-organization collaboration within a single accessible platform. Unlike traditional visualization tools requiring significant technical expertise, the system lowers the entry barrier by utilizing a cloud-based LLM to interpret dataset context and suggest meaningful visualizations.

The incorporation of a lightweight recommendation engine provides users with explicit guidance on chart selection and insight directions, reducing manual effort and cognitive load. The use of bandit learning to refine output quality based on feedback adds an adaptive component not commonly present in standard BI tools. Furthermore, the inclusion of a Telegram bot extends accessibility to conversational environments, a feature typically absents in general-purpose visualization platforms.

The combination of these elements-automated reasoning, collaborative structures, conversational access, and feedback-driven refinement-constitutes a novel approach toward democratizing data analytics and making insight generation more intuitive and efficient.

2.1 Literature Survey

2.1.1 Related Work

Recent systems that use LLMs for data analysis and visualization, show distinct strengths and limits. For instance, LIDA (2023) auto-generates visuals and infographics with a hybrid UI for direct edits and multilingual input. It is strong at automated EDA/visualization but gives generic, not domain-specific, insights and focuses on visual quality over deep business reasoning.

Insight Pilot (2023) from Microsoft Research lets users ask in natural language and guides the users through multi-step actions (understand, summarize, compare, explain). It handles vague queries by breaking them down, but the visuals are basic and often yields broad, not detailed, visual analytics.

Chat2VIS (2023) turns natural-language queries into charts using pre-trained LLMs (e.g., GPT-3.5) in a simple Streamlit UI. It is multilingual, quick, and supports iterative refinement, proving prompt engineering can produce reliable Python viz code, yet it focuses on rendering and prompts only, no trend analysis or recommendations. The outputs are just charts and not narrative insights.

LLM4Vis (2023) uses ChatGPT to recommend charts, representing table features as text with few-shot prompts to improve transparency. It mainly explains and does not actually draw charts or do heavy processing, lacks predictive/prescriptive analytics, and explanations can lack domain depth.

ChatGPT (2024) creates precise charts from abstract requests via structured reasoning, fine-tuned on (natural language, chart) pairs and shows intermediate steps for user edits to improve NL2Viz accuracy. It targets visualization only, with no narratives or business insights, assumes the query implies the analysis, and offers no freeform exploration or next-question suggestions.

Chat BI (2024) translates natural language to complex SQL for BI databases. It is limited to structured databases and returns tables (not charts), provides no explanatory insights, assumes a well-defined DB, and is not suited for CSVs, unstructured data, or automated trend/anomaly analytics.

The existing solutions are either limited to providing visualizations or providing broader and generalized business intelligence. These limitations drive the need for a unified system that provides detailed visualizations along with specific and actionable business insights.

Research Findings for Existing Literature

Despite clear progress, a gap remains in business analytics: there is no single platform that both makes visualizations and gives meaningful, tailored, real-time insights for decisions. Non-technical managers and analysts still face delays because they must learn complex tools or depend on experts. Existing systems cover parts of the workflow, but none remove this barrier end to end.

Table 2.1.3.1: Research Findings

S. No.	Roll Number	Name	Paper Title	Tools & Technology	Findings	Citation
1	102203023	Siddhant	LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics using Large Language Models.	GPT-4, prompt engineering, Python, Jupyter, Altair.	The paper shows that combining LLMs and IGMs in a pipeline can automate end-to-end data visualization, covering data understanding, chart specification, and design. It introduces two evaluation metrics: Visualization Error Rate (VER) for pipeline reliability and Self-Evaluated Visualization Quality (SEVQ) for output quality.	Dibia [6]
2			Data-to-Dashboard: Multi- Agent LLM Framework for Insightful Visualization in Enterprise Analytics	LLMs, agent orchestratio n, dashboard generation, Vega Lite, Python	The multi-agent approach produced charts with greater depth and domain relevance than a single-prompt GPT-4 baseline, improving the insightfulness of outputs and capturing domain-specific details that generic QA systems missed.	Zhang and Elhamod [8]

3	102203091	Sanchit Nanda	Insight Pilot: An LLM-Empowered Automated Data Exploration System	LLMs, insight engine, iterative LLM loop, Python, web UI	Insight Pilot helped users uncover valuable insights from datasets with significantly less effort by integrating an LLM with an insight engine, improving the accuracy of exploratory analysis and yielding more trustworthy results than unguided LLM analysis; given a natural-language question, the system automatically selects analysis intents and has the LLM execute a sequence of IQueries to analyze data step by step, mimicking a human analyst.	Ma et al. [7]
4			InsightLens	LLM agent, React, web UI	InsightLens significantly reduces the manual and cognitive effort required to track and manage insights during chat-based analysis by visualizing complex conversational context, enabling users to retrieve and explore insights.	Weng et al. [23]
5	102203835	Udit Gupta	Chat2VIS: Fine-Tuning Data Visualizations, Natural Language Text and Pre-Trained LLMs	ChatGPT, GPT models, Vega Lite, Altair, prompt templates	Chat2VIS demonstrated flexibility by understanding and executing visualization requests in multiple languages, outperforming prior English-only approaches, introducing quantitative evaluation benchmarks for NL2VIS, and achieving good accuracy in translating language prompts into correct visualizations.	Maddigan [15]

6			Generating Analytic Specifications for Data Visualization from Natural Language Queries using Large Language Models	GPT-4, NL4DV, prompt templates, Python toolkit, Vega Lite	The paper shows that an LLM-driven approach achieved 87.0% accuracy in mapping queries to correct specifications, compared to 64.1% for a traditional rule-based system, demonstrating a significant improvement in interpreting natural-language data queries.	Sah et al. [9]
7	102203684	Swapnil Gumber	LLM4Vis: Explainable Visualization Recommendation using ChatGPT	GPT models, prompt templates, recommendation heuristics	The method uses multiple prompt stages to describe dataset features, select demonstration examples, generate an initial visualization with an explanation, and iteratively refine that explanation through bootstrapping, enabling LLM4Vis to recommend charts without a large training corpus while providing natural-language rationales for each recommendation.	Wang et al. [16]
8			Automated Data Visualization from Natural Language via Large Language Models: An Exploratory Study	GPT models, T5 family, nvBench dataset, NL2Vis benchmarks, Python	The paper shows that LLMs can outperform specialized NL2Vis systems, with inference-only GPT-3.5 using in-context few-shot learning surpassing a fine-tuned model, and that iterative refinement of LLM outputs further improves visualization query generation.	Wu et al. [10]

9	102203313	Aryan Chharia	ChartGPT: Leveraging LLMs to Generate Charts from Abstract Natural Language	LLaMA fine tuning, Vega Lite specs, PyTorch, stepwise reasoning	ChartGPT uses decomposed reasoning and fine-tuning to generate more accurate charts from high-level queries, effectively capturing user intent where prior NLIs fail; quantitative evaluations and a user study show it reliably produces correct and useful charts even from vague prompts, significantly improving success rates over baseline LLM and rule-based approaches.	Tian et al. [17]
10			ChatBI: Towards Natural Language to Complex Business Intelligence SQL	Schema linking, SQL generation, LLMs, view selection, JSON intermediate representation	The paper introduces a solution for translating natural-language questions into complex BI SQL queries using a hybrid approach in which an ML model first selects a relevant database subset and an LLM then generates SQL on the chosen schema, effectively handling multi-turn dialog context and advanced BI analytics.	Lian et al. [18]

2.1.2 Research Gaps of Existing Literature

Although recent systems incorporate LLMs for elements of visualization or natural language interaction, none comprehensively bridge the full spectrum of analytical intelligence needed for unified, context-aware, business-relevant analytics. Several key gaps remain in current research.

Gap A: Limited Integration of Automated Insight Narratives with Visualization Workflows

Many existing tools focus on chart generation or surface patterns visually but do not convert visual patterns into structured narrative insights that align with business reasoning. Systems such as Chat2VIS or ChartGPT produce charts but lack mechanisms to translate statistical trends into explanatory narratives that support decisions. The research literature on visual analytics and LLM integration reports that while LLMs can assist in interpretation and natural language descriptions, there is limited work on methods that systematically integrate automated narrative synthesis with visualization outputs in an end-to-end workflow [13].

Gap B: Insufficient Support for Contextual and Session-Aware Analytical Progression

Existing systems typically respond to isolated queries without awareness of broader analytical context or session history. In practical analytics, insights build on previous results, trends evolve across steps, and recommendations should reflect session context, guiding the user through a thoughtful analytical progression. Research on LLM-based data science agents highlights the need for session memory and planning across multi-step workflows in data analysis tasks, a capability not present in current NL2Viz or BI tools [9].

Gap C: Lack of End-to-End EDA and AI-driven Analytic Reasoning Integration

While some systems provide automation at certain stages (e.g., chart suggestion, simple EDA summaries), they do not embed comprehensive exploratory data analysis as part of the automated pipeline from raw data to insight recommendation. Research in AI-assisted EDA identifies that although AI techniques can automate parts of EDA, full integration of exploration, pattern detection, and subsequent explanation or hypothesis generation remains underdeveloped [8]. Your solution distinguishes itself by making EDA an integrated, automated precursor to narrative insight, rather than a separate or manual step [8].

Gap D: Weak Handling of Rich, Heterogeneous and Contextual Data Types

Many existing visualization and BI systems assume relatively simple, flat structured datasets, and struggle with heterogeneous or relational data structures common in real-world business scenarios. While some research addresses specific aspects of table understanding, comprehensive solutions for schema reasoning, relationship exploration, and semi-structured dataset interpretation remain limited. Survey work on data science agents notes that multimodal reasoning spanning text, tables, code, and visuals is still an open challenge, particularly when transitioning between these domains within a single workflow [1].

Gap E: Absence of Reliable Evaluation & Explainability Mechanisms for Automated Insights

Existing systems seldom incorporate robust mechanisms for evaluating the quality, reliability, or trustworthiness of insights generated automatically. Beyond chart accuracy, businesses require assurances that the generated interpretations are valid, unbiased, and supported by data. Research on LLM-driven data science workflows highlights the low prevalence of explainability, trust, and governance evaluations in current systems, suggesting a need for frameworks that allow users to assess and audit analytic outputs [1].

Gap F: Limited Evaluation Frameworks for Insight Quality and Impact

There is a lack of standardized benchmarking methods for assessing the quality and practical utility of insights produced by automated systems. While tools like InsightPilot or LIDA can be qualitatively evaluated for individual features, comprehensive metrics for measuring insight relevance, actionability, and business impact remain undeveloped. Research surveys call for rigorous evaluation frameworks that go beyond visualization correctness to assess analytic reasoning and decision support efficacy [7].

Gap G: Underdeveloped Adaptive Recommendation Systems

While some systems provide rudimentary visualization suggestions, few adapt recommendations to evolving analytical context or user interaction history. Analyzing a dataset is not static; as insights accumulate, subsequent recommendations should adjust in response to discovered patterns, shifts in analytical focus, or user goals. Current research surveys note that dynamic, context-adaptive agents capable of adjusting recommendations throughout an analytical session remain a significant under-researched area [20].

Gap H: Lack of Automated Multi-User and Collaborative Analytics Support

Although some enterprise BI tools offer collaborative dashboards, there is limited research on LLM-driven analytic systems that formally support team structures, shared contexts, and collaborative insight refinement. True collaborative analysis would allow multiple stakeholders to contribute and refine insights, share context, and reason jointly. Surveys in LLM-based agents discuss agent orchestration but not collaborative analytics workflows tailored to BI and visualization use cases [14].

2.1.3 Detailed Problem Analysis

Although significant advancements have been made in visualization tools and LLM-based analytic assistance, a fundamental gap persists in the domain of practical business intelligence. Current systems either excel at generating visualizations or at answering questions about data, but they rarely unify these capabilities into a seamless, end-to-end analytical workflow. As a result, non-technical decision-makers continue to struggle with deriving actionable insights from their data. They are often required to navigate multiple tools-one for preprocessing, another for visualization, and yet another for interpretation-which ultimately slows decision cycles, introduces cognitive friction, and increases dependency on specialized analysts.

The central challenge emerges from the fragmented nature of existing analytics workflows. In most organizations, raw datasets must first undergo preprocessing and exploratory data analysis, typically carried out by trained analysts. Visualization then requires decision-making about chart selection, design principles, and narrative context. Finally, meaningful interpretation still relies heavily on business analysts who translate visual patterns into strategic insights. This multi-step, role-segmented pipeline introduces bottlenecks and makes rapid or iterative analysis impractical for non-technical users. Consequently, the insight-to-action gap remains wide, especially in scenarios where data-driven decisions must be made promptly.

The overarching question underpinning this problem can therefore be framed as follows: How can a non-technical user progress from a raw dataset to contextualized, data-grounded, decision-ready insights-supported by appropriate visualizations-within a single, intuitive AI-driven workflow? Existing tools generally address only portions of this pathway. Visualization-centric platforms generate charts but lack deeper reasoning about trends or anomalies.

Conversely, question- answering systems powered by LLMs may provide explanations but seldom anchor them to customized, context-relevant visuals. Moreover, manual intervention is still required to select relevant columns, interpret chart patterns, or validate business relevance. An effective solution must therefore overcome three key obstacles:

1. Usability: ensuring that non-expert users can interact conversationally without learning complex interfaces or analytical terminology.
2. Data-grounded reasoning: guaranteeing that insights are based on the actual dataset rather than generic or hallucinated responses.
3. Real-time integration: enabling visualizations and narrative insights to be generated concurrently and updated dynamically as users explore their data.

The absence of such an integrated platform limits the democratization of analytics. It creates disparities between organizations with dedicated analytics teams and those-such as SMEs, educational institutions, and public sector entities-that lack such resources. This fragmentation also inhibits iterative analysis, rapid prototyping, and exploration across domains.

In summary, the core problem is not merely the lack of visualization tools or LLM-based question answering, but the absence of a unified, AI-powered platform that can automate visualization and deliver meaningful, context-aware business insights simultaneously, without imposing technical overhead. The proposed LLM-Driven Business Intelligence and Visualization Platform is designed to address this unmet need by merging visualization, analytic reasoning, and conversational interaction into a single accessible system-ultimately enabling users to transition from raw data to actionable intelligence efficiently and intuitively.

2.1.4 Survey of Tools and Technologies Used

The system uses a clear, modular stack across frontend, backend, machine learning, visualization, and storage. We chose each tool for fit, scale, and easy integration.

Frontend

React builds the web UI where users upload CSVs, see charts, and read insights. Its components keep the editor and views modular. Tailwind CSS gives fast, responsive styling for the code panel, chart viewers, and results layout. JavaScript and HTML drive interaction logic and page structure, including file uploads and “rebuild chart” controls.

Backend

Python runs the pipeline end to end: CSV parsing, automated EDA, LLM calls, code execution, chart rendering, and summary extraction. We used Streamlit during prototyping to test the input → LLM → output loop and the LLM-to-chart feedback cycle before the full React UI. Google's Gemini 2.5 handles natural-language understanding and code generation; it reads the data frame head, writes Python or SQL for charts, and drafts insight text. We used Google Colab early for EDA logic, LLM validation, and summaries before moving to production code.

Data Storage

MongoDB stores CSV metadata, generated charts, user queries, summaries, and insight responses, preserving session history and enabling regeneration. ChromaDB acts as a vector store to index chart descriptions, query embeddings, and sessions, enabling semantic search and context reuse later.

Visualization and summaries

Plotly, Altair, and Matplotlib render interactive and static charts from LLM-generated code (Plotly is preferred in React). After rendering, we programmatically extract summary data - like bin counts or category totals - for insight generation. Pandas and NumPy handle reading, cleaning, typing, and all chart-level computations.

Advanced vision and reasoning

ChartQA and PlotQA will answer natural-language questions about charts or chart images, extending the system to image-based reasoning on generated or user-uploaded figures. These models will be optionally integrated in the later phases of the project.

Each tool serves a focused role in a scalable, extensible pipeline. The stack supports strong interactivity, safe LLM integration, and a path from a chart-centric dashboard to a full, insight-driven analytics assistant suitable for academic and industry use.

2.1.5 Summary

The proposed LLM-Driven Business Intelligence and Visualization Platform build directly upon the foundations established by recent systems such as LIDA, InsightPilot, Chat2VIS, LLM4Vis, ChartGPT, and ChatBI, yet distinguishes itself by addressing the unresolved limitations that persist across these solutions. Existing tools have demonstrated important advances-such as automated visualization generation, natural language interaction, and chart recommendation-and our work leverages these demonstrated strengths rather than disregarding them. Specifically, we adopt the core idea that LLMs can translate natural language requests into analytic actions, and we extend the evidence that visualization code and chart structures can be reliably generated by large language models. In addition, we inherit lessons from systems like InsightPilot that show the value of guided multi-step reasoning, and from LIDA that demonstrate the feasibility of multilingual and hybrid UI integration.

However, the core contribution of our platform lies in advancing beyond the fragmented and partial automation provided by prior systems. Unlike LIDA or Chat2VIS, which primarily focus on visualization output, our system does not stop at chart generation. It incorporates a higher-level layer of business intelligence reasoning, wherein insights are contextualized, interpreted, and converted into actionable narratives rather than simply visual representations. Where ChartGPT emphasizes fidelity in chart creation, our platform emphasizes *interpretation and decision support* in addition to visualization.

Furthermore, in contrast to LLM4Vis, which focuses on explanation of visualization choice without executing full analytical workflows, our system performs the end-to-end cycle-from raw CSV ingestion, lightweight EDA, chart generation, recommendation, and narrative insight-within a single cohesive pipeline. This means that users do not need separate tools or expert intervention to move from raw data to interpreted findings. The integration of dataset parsing, cleaning awareness, and context extraction directly into the LLM prompting and chart recommendation mechanism represents a distinct conceptual shift from tools that treat visualization and reasoning as independent modules.

Another critical differentiator lies in real-time adaptability and interactivity. Whereas existing tools operate primarily on static, query-based interactions, our platform incorporates bandit learning to iteratively refine future insight suggestions based on user feedback. In other words, the system not only responds to queries but *learns* which insights are valued and progressively adapts to domain-specific contexts. Prior work does not include such iterative, feedback-driven personalization.

Similarly, the inclusion of a multi-organization and team-based architectural design, coupled with JWT authentication and dataset isolation, targets operational realities absent in prior academic prototypes. Existing solutions mainly assume single-user or single-session workflows. Our work anticipates enterprise and collaborative usage, offering structured data segregation, persistent chat histories, and audit-ready traceability-capabilities rarely explored in NL2Viz literature.

The extension of functionality to conversational environments through a Telegram bot also expands accessibility beyond web interfaces, allowing analytics to be conducted in asynchronous, mobile-friendly, low-resource scenarios. Unlike platforms restricted to browser-based UIs, our system supports cross-platform interaction without requiring installation of dedicated BI clients or dashboards.

Most importantly, our work departs radically from existing approaches by unifying four elements that have so far been pursued independently:

1. Automated exploratory reasoning,
2. Visualization generation,
3. Business insight interpretation, and
4. Context-aware recommendation and refinement.

This fusion enables non-technical users to follow a single, conversational workflow that transforms raw data into actionable insights without manual coding, chart selection, or domain expertise. In doing so, the platform makes analytics more inclusive and democratized, particularly for small and resource-limited organizations that cannot rely on dedicated analysts.

In summary, while our system inherits foundational concepts from prior research-such as natural language interaction, automated visualization, and LLM-driven reasoning-it extends them into a unified, feedback-driven, context-aware platform that simultaneously generates charts and interpretable business intelligence. It addresses critical gaps left unresolved by earlier solutions by integrating end-to-end automation, iterative refinement, multi-organization support, and real-time narrative insight generation. Thus, it not only builds on prior work but also meaningfully advances beyond it, providing a practical, accessible, and holistic alternative to current fragmented analytics tools.

2.3 SOFTWARE REQUIREMENTS SPECIFICATION

2.3.1 Introduction

The Software Requirements Specification (SRS) outlines the functional and non-functional requirements of the LLM-driven Data Visualization and Business Insight Generation Tool. It defines the system's goals, behavioral expectations, performance criteria, user interactions, and deployment considerations. The SRS serves as a reference document for stakeholders involved in development, validation, deployment, and maintenance. By articulating the system's requirements in a structured form, it ensures uniform understanding among stakeholders, facilitates systematic design decisions, and guides subsequent development phases. The document also clarifies constraints, assumptions, and dependencies associated with the system to ensure that its implementation aligns with the objectives defined in earlier sections.

2.3.1.1 Purpose

The purpose of this SRS is to define a comprehensive and precise description of the software system being developed. It establishes clear functional requirements related to dataset ingestion, interactive visualization, LLM-driven insight generation, recommendation capabilities, authentication mechanisms, and multi-organization management. It also outlines non-functional requirements such as security, performance, usability, scalability, and deployment considerations. The SRS provides a baseline for validating the system against expected outcomes and ensuring that the design, implementation, and testing phases remain aligned with the project objectives. Ultimately, this document ensures that developers, evaluators, and end-users possess a unified understanding of the system's intended behavior and operational constraints.

2.3.1.2 Intended Audience and Reading Suggestions

This document is intended for a diverse audience involved in different stages of the software lifecycle, including:

- **Project developers and engineers**, who require detailed requirements for implementation.
- **Project mentors and evaluators**, who assess whether the system satisfies its intended objectives and conforms to academic and technical standards.

- **End-users and stakeholders**, who aim to understand system functionalities, usability constraints, and operational characteristics.
- **Testing and quality assurance personnel**, who require measurable specifications for validating system correctness.

Readers are advised to begin with the introductory sections to understand system context and objectives, followed by the functional and non-functional requirements for detailed understanding. Developers may refer to component-level descriptions and interface requirements, while evaluators may focus on scope, assumptions, and constraints. The SRS sections are structured logically to support progressive comprehension-from conceptual overview to detailed operational requirements.

2.3.1.3 Project Scope

The scope of this project encompasses the development of a cloud-enabled system for automated insight generation and data visualization, accessible through a web interface and conversational bot. The platform allows users to upload structured CSV datasets, visualize data using interactive charts, receive LLM-driven insight suggestions, and leverage recommendation mechanisms for selecting meaningful visualizations. The system supports dataset and chat storage using MongoDB, enables authenticated multi-organization and team-based access, and provides a scalable deployment pathway through cloud hosting. The scope includes the implementation of bandit learning for iterative refinement of insights, integration of a Telegram bot for conversational usage, and use of Plotly for visualization rendering.

Out of scope are advanced natural language editing of visualizations, local LLM deployment in the production version, complex statistical modeling beyond automated insight generation, and support for extremely large datasets beyond the current deployment capacity. Nevertheless, the system is designed to be extensible, allowing future enhancements such as domain-specific templates, RLHF, and transition to self-hosted LLMs.

2.3.2 Overall Description

The software system is designed as a cloud-enabled, web-based platform that facilitates intuitive data visualization, automated insight generation, and recommendation-driven analytical exploration.

It provides users with the ability to upload datasets, preview their structure, visualize information through interactive charts, and derive business insights without requiring programming or advanced statistical knowledge. The system combines a React-based frontend for visualization and user interaction, an Express and Fast API backend for dataset processing and system operations, and MongoDB for persistent storage of datasets, chat histories, and organizational metadata. A cloud-based LLM is used to interpret dataset context and generate insights and recommendations, while a Telegram bot extends accessibility to users through conversational interfaces.

The system architecture follows a modular and layered design approach, separating dataset ingestion, insight generation, visualization rendering, and user authentication into distinct components. The platform supports multi-organization and multi-team structures, enabling collaborative analytics across different user groups while maintaining secure access through JWT authentication. Interactive charting is powered by Plotly, facilitating dynamic visualization of numerical and categorical data. The recommendation engine simplifies analytical decision-making by suggesting appropriate visualization types and insight categories, thereby lowering the barrier to adoption for non-technical users. Bandit learning mechanisms enable refinement of insight quality by incorporating real-time user feedback. The system is deployed using cloud platforms such as Render and Railway to ensure scalability and accessibility, while maintaining moderate dataset size support in the deployed environment.

2.3.2.1 Product Perspective

The proposed system functions as a standalone analytics platform while also offering the ability to integrate with external communication channels through its Telegram bot interface. It serves as an alternative to conventional business intelligence tools by providing automated visualization and insight generation features that reduce dependence on user expertise. The system is architected using a service-oriented approach in which dataset processing, insight generation, visualization, and authentication services operate as modular components. The React frontend communicates with backend services through RESTful APIs, while MongoDB acts as the persistent storage layer.

This system evolved from an earlier Streamlit-based prototype intended for feasibility testing and was subsequently transitioned to an Express and Fast API-driven architecture to enhance scalability, modularity, and component reuse. Compared with traditional tools that rely on manual configuration and expert knowledge, the platform integrates a cloud-based LLM for context-aware insight generation and a recommendation mechanism for suggesting visualization types. Its deployment on Render and Railway reflects modern cloud-hosting practices, enabling elastic scaling and remote accessibility. The multi-organization and team-based structure positions the platform for real-world environments where dataset access and insight workflows are collaborative.

2.3.2.2 Product Features

The primary features of the system include:

- **Dataset Upload and Preview:** Users can upload CSV files and view dataset structure, including column names and sample records.
- **Interactive Visualization:** Plotly-based charts facilitate dynamic and intuitive representation of trends, comparisons, and distributions.
- **LLM-Driven Insight Generation:** The system analyzes sample dataset context to produce interpretive insights and recommend relevant analytical directions.
- **Visualization Recommendations:** The recommendation engine provides chart type suggestions and column pairings for effective visualization.
- **Multi-Organization and Team Structure:** Secure segregation of datasets and chat histories supports collaborative analytics within distinct groups.
- **JWT-Based Authentication:** Access control and secure login are enforced through token-based authentication mechanisms.
- **Bandit Learning for Feedback:** Insights are iteratively refined using user feedback collected through thumbs-up/down responses.
- **Telegram Bot Interface:** A conversational access layer enables interaction with the system through a messaging platform.
- **Cloud Deployment:** Hosting via Render and Railway ensures accessibility and scalability.
- **Moderate Dataset Support:** The deployed version handles datasets up to approximately 5000 rows, with internal testing demonstrating scalability on larger datasets.

2.2.3 External Interface Requirements

2.2.3.1 User Interfaces

The user interface will be designed to be clean, intuitive, and require no technical knowledge. Key UI elements and requirements include:

- Data Upload & Setup UI: The homepage or start of the app will prompt the user to upload a dataset file (with clear instructions on supported formats, e.g., “Upload CSV or Excel – up to ~20MB”). After upload, the UI may display a brief summary of the data (columns detected, perhaps a preview of the first few rows) so that the user knows the data is read correctly. If any issues occur in reading the file (formatting errors, etc.), the UI should show a meaningful error message.

- Conversational Chat Window: The main interface will be a chat-style panel where the user enters questions and the system’s responses are shown. The user input box will have placeholder text like “Ask a question about your data…”. When the user submits a question, a loading indicator or message (“Analyzing…”) will appear to indicate the system is working. The AI’s answer will then be displayed as a chat bubble containing both text and any generated chart images.

- Visualization Display: Charts generated by the system will appear within the chat transcript at the appropriate point, properly labelled. The UI will ensure charts are clear (appropriate size and resolution).

- Responsiveness and Layout: The interface will be web-responsive to a degree. The layout will consist of a sidebar and a main panel (chat conversation). Colors and style will be kept professional and not distracting – e.g., a white background, clear font for text, and use of color only within charts or to highlight important values in text (for example, using bold or colored text for key numbers).

- Overall, the UI is designed to make the user feel like they are chatting with a knowledgeable data analyst. The requirement is that a user with basic computer skills can navigate it without reading a manual: upload data, ask questions, and read the resulting insights and charts. Achieving a smooth, straightforward UI is important for user adoption.

2.2.3.2 Hardware Interfaces

Since our system is a software application delivered via a web interface, there are minimal hardware interface requirements on the user side:

User Hardware: The user needs a standard computing device (PC, laptop, or tablet) with an internet connection and a modern web browser. No special hardware sensors or devices are required to use the application. There is no direct interaction with hardware like printers or scanners through our software.

2.2.3.3 Software Interfaces

This section describes how our system will interface with other software systems or APIs during its operation:

- External AI API: We will use an external AI service (e.g., OpenAI's API for GPT-4), the system will interface with it over HTTPS. The requirement here is to follow the API's protocol: sending the prompt (which includes the user's query and relevant data context) in the API's format and receiving the response (text and code) from the API. We must implement proper authentication (API keys) and error checking for this integration. In future, we will also use our own LLM model which will be hosted on a cloud service. After hosting, we will use the LLM via the same API architecture we have built.
- Database or Storage: The system will use MongoDB database for storing of the user prompts and the system outputs. We will also use ChromaDB for storing the specific session chats as vector embeddings and then reloading them whenever the user logs in. The database will be storing the graphs as well as the insights.

2.2.4 Other Non-functional Requirements

2.2.4.1 Performance Requirements

- Response Time: For an average query on a moderate-sized dataset (~10k rows, 10 columns), the system will return an answer within a few seconds, ideally under 20 seconds. Realistically, initial queries that involve parsing data and generating first insights may take slightly longer (perhaps 5–8 seconds). We will optimize for responsiveness, but since an LLM is involved, some latency is inevitable.

- Data Size Handling: The platform should handle datasets up to a certain size efficiently. We target support for files up to ~50 MB or around 100,000 rows in a CSV as a rough benchmark. Larger datasets might cause performance issues (especially for an LLM to analyze or for Pandas in memory). If a dataset is too large, the system should either sample it or prompt the user to refine it (for example, "Dataset is quite large; consider filtering some data or I will sample 10% for analysis"). We may implement a sampling strategy for extremely large data to ensure responsiveness.

- Scalability: The system should be able to handle a growing number of users without a drop in performance. It must scale efficiently to accommodate peak usage times, without affecting response quality or speed.

- Error Rates: From a performance perspective, the system should have low failure rates when generating outputs. For instance, the percentage of queries that result in a timeout or unhandled exception should be under 5%.

- Benchmark Comparison: Performance will also be informally compared with human effort – if a task like plotting a graph and explaining it would take a human analyst 5–10 minutes, our system doing it in seconds is a huge win. We want all typical tasks (filtering data, computing aggregates, plotting) to be performed at interactive speeds.

2.2.4.2 Safety Requirements

- Safe Code Execution: The platform must guarantee that any code generated by the LLM and executed (for data analysis or plotting) is done in a controlled environment. This means preventing potentially malicious or dangerous operations. Running code in a sandbox or with restricted permissions is required.

- Handling of Erroneous Input: The system should fail safely for bad inputs. If a user uploads a corrupt file or asks an incoherent question (e.g., mixing unrelated concepts), the system should not crash. It should provide an error message or a polite note that it cannot fulfil that request. Safe failure means always staying in control of the execution flow.

- Privacy Safety: If the user is analyzing sensitive data (e.g., personal or financial information), the system should not expose it unexpectedly. All data stays in memory or on secure disk on the server that the users trust. This transparency can be considered part of user safety in terms of trust.

2.2.4.3 Security Requirements

Security is vital since the system may handle proprietary business data. The following are key security requirements:

- Data Confidentiality: Any data the user uploads should be protected. This means our system should not send the data to any external entity except as necessary for functionality. Essentially, we ensure no unauthorized access to the raw data. In a multi-user system, each user's data and session would be isolated (e.g., separate session IDs, no crossover of data in memory).
- Secure Communication: Client-server communication must be encrypted (HTTPS) to prevent eavesdropping on sensitive data or Q&A content.
- API Security: While using external APIs (LLM services, etc.), we must secure our API keys and credentials. Those keys should not be exposed on the client side or in public repositories. They will be stored server-side (e.g., in environment variables), and our code will call the external API from the server, so the user never sees the key.
- Compliance: The system should comply with applicable data protection regulations such as GDPR or equivalent Indian data protection laws, ensuring legal standards for user privacy and security are met.

2.4 Cost Analysis

- LLM Models: API costs for proprietary models
- OpenAI GPT-4o : $5 / $ 20 per 1M token. (input / output)
- OpenAI GPT-4o mini : $0.60 / $ 2.40 per 1M token (input / output)
- Google Gemini 1.5 Flash : $0.075 / $0.30 per 128k token (input / output)
- Cloud Services: Cloud services like AWS will be required for hosting website and LLM model (in case of a fine-tuned model)
- AWS EC2 on-demand service –
- t3.small : ₹1.82 / hr.
- t3.large : ₹7.27 / hr.

- Database : Database is required for storing the chat history.
- MongoDB Atlas :
- M0 (Free tier) : Free
- M2 (Shared) : $9 / month
- Chroma (vector DB) : Free, open-source.

2.5 Risk Analysis

Like any software project, this platform faces certain risks. We identify the major risks and outline mitigation strategies for each:

- Data Breach: A significant risk is the potential for unauthorized access to sensitive information, which could lead to data breaches. Mitigation strategies include implementing strong encryption, secure authentication methods, and regular security audits.
- System Downtime: Unexpected downtime could disrupt access to vital information. To mitigate this, a robust disaster recovery plan, including regular backups and failover mechanisms, should be implemented.
- Model Inaccuracy: There's a risk of the chatbot providing inaccurate or outdated information, especially if the model is not frequently updated with the latest data. Regular fine-tuning of the LLM and continuous monitoring of responses will help address this risk.
- User Data Mismanagement: Improper handling of user data could lead to privacy violations or loss of trust. Clear data management policies, regular audits, and user consent mechanisms should be enforced to minimize this risk.

METHODOLOGY ADOPTED

3.1 Investigative Techniques

Table 3.1.1: Investigative Techniques

S.No.	Investigative Project Technique	Investigative Technique Description	Investigative Project Examples
1.	Experimental	Organized experiment to test a hypothesis with a control. We compare our LLM-assisted app against a baseline (manual pandas/matplotlib or a standard BI tool) under the same datasets and tasks. We vary the model/prompt as treatments and measure accuracy, time, success rate, and insight quality while keeping data, hardware, and splits constant.	AI/ML system evaluation: Streamlit + Gemini "NL→Chart+Insights" app. Treatments: gemini-2.0-flash vs 1.5-flash, with/without head+dtypes in prompt. Baseline: manual plotting/BI. Outcomes: time-to-first-correct chart, execution success rate, chart correctness score, usefulness of insights.

3.2 Proposed Solution

This section presents the proposed solution as an engineered artifact that implements the pipeline from CSV intake to business insights. The description focuses on the system's structure, the data and control flows between components, and the measures that make the solution reliable, safe, and useful.

3.2.1 Solution overview

The solution is a web-based application that enables a user to upload a CSV file, review an automated exploratory data analysis (EDA) report, request a chart in natural language, execute model-generated plotting code in a controlled environment, extract a numeric summary from the rendered chart, and obtain concise business insights based on that summary. The language model (Gemini 2.5) is used for two tasks only: generating plotting code and drafting insights. All execution of code and all visualization rendering occur locally in the application process, not inside the model. To reduce token costs and limit data exposure, the application sends the first five rows of the dataset and the inferred column data types once, during the first chart request, and does not resend them for subsequent requests in the same session.

The plotting function returned by the model must conform to a fixed contract, returning both a figure object and a JSON-serializable numeric summary that the application validates and uses downstream.

3.2.2 Architectural decomposition

The solution is organized in five cooperating layers. The presentation layer is the Streamlit user interface that manages file upload, displays the EDA report, accepts the user's chart request, renders figures, shows the numeric summary, and presents the generated insights. The orchestration layer coordinates the end-to-end flow: it caches the CSV head and dtypes; builds prompts; decides when to send head information; invokes the model; and routes outputs to the validator, executor, and insight generator. The model access layer encapsulates Gemini calls; it applies the model contract using a system instruction and submits only user-role messages, and it wraps calls with a simple retry mechanism and an optional fallback from a Pro variant to a Flash variant when quotas or transient errors occur. The execution sandbox layer is responsible for running the returned plotting code. It provides a restricted global environment with whitelisted libraries (pandas, numpy, matplotlib, plotly.express, altair), and it blocks imports, operating system and file operations, evaluation primitives, and double-underscore access. This layer also asserts that a function with the required signature is present and that its return value has the expected structure. Finally, the analysis layer contains the EDA generator based on ydata-profiling, the numeric summary schema checker, and the insight prompt that turns the summary into short, actionable guidance.

3.2.3 Data flow and state management

The application maintains a minimal session state comprising the uploaded Data Frame, the cached head and dtype dictionary, a boolean flag indicating whether the head has been sent, the last piece of plotting code produced by the model, the last numeric summary, and the latest insights text. When a CSV is uploaded, the system computes the head of five rows and captures column data types. The user is free to view the EDA report immediately. When the first chart request is entered, the orchestrator builds the code-generation prompt that includes the head and data types once. The model returns code; the sandbox validates the text and executes the plotting function against the full DataFrame. The figure is rendered in the interface and the numeric summary is displayed as JSON. The same summary is then sent to the model in a separate insight-generation call. For additional chart requests in the same session, the orchestrator omits the head and data type blocks and sends only the new user request, which reduces token usage and avoids resending data. The application also exposes the generated code in an editor panel so that a user can correct column names or make small adjustments and rerun the chart locally without engaging the model again.

3.2.4 Prompting and contract design

Two concise system instructions define the model's role. The code-generation instruction requires the model to output only Python code that defines a function named build_chart(df) which returns a figure and a JSON-serializable numeric summary. The text explicitly forbids imports and file or network access and declares the set of plotting libraries already available in the execution environment. It states that for categorical and distribution charts the summary must be a list of objects with label and value keys, and for temporal or series plots the summary must use x and y. The insight-generation instruction defines the model's role as a concise business analyst that must propose short, evidence-based insights referencing the numbers in the summary and clearly indicating likely next actions. In both cases the instructions are supplied via the model's system instruction field, and the messages submitted to generate_content are user-role only. Temperature and other parameters are kept at default values for stability unless specific experiments require changes. The prompting logic is uncomplicated on purpose; the correctness is enforced by the contract verification and the sandbox rather than by complex prompt engineering.

3.2.5 Execution sandbox and validators

The sandbox is the core safety mechanism. Before any dynamic execution, the returned text is scanned for forbidden tokens. These include import statements, file operations, operating system and system module access patterns, dynamic evaluation primitives, and double-underscore sequences. If any are found, the run is rejected with a clear error message. If the text passes this gate, it is executed in a restricted global context that exposes only whitelisted plotting and data libraries and a very small set of safe builtins. After execution, the application looks up the build_chart symbol and verifies that it is callable. The function is invoked with the full DataFrame, and the return value is verified to be a two-element structure. The first element is checked to be an object recognized by one of the supported visualization libraries; the second element is checked to be JSON-serializable and to match the declared schema for the requested chart family. Any failure at these checkpoints yields a controlled error that is surfaced to the user and recorded in the logs.

3.2.6 Automated EDA integration

Automated EDA is integrated to support users and to make evaluation datasets more transparent. When a CSV is uploaded, the application can generate an HTML profile using ydata-profiling in minimal exploratory mode. The profile indicates data types, missingness, distributions, and simple correlations. The report is embedded directly into the interface and does not involve the model. While the EDA output does not drive the model prompts, it helps users form reasonable chart requests and provides context during evaluation runs. It also assists in diagnosing model failures that arise from unexpected column types or extreme skew.

3.2.7 Reliability, resilience, and error handling

The model access layer implements basic resilience measures. Calls to the model are wrapped in a retry function that retries transient failures such as rate limits and service unavailability with a brief delay. If the Pro model is rate-limited or temporarily unavailable, the system can fall back to a Flash variant automatically for the same prompt. Errors are presented in plain language in the interface, and the last generated code is always visible and editable, enabling a user to apply targeted fixes without waiting for another model response. The head-only-once logic is enforced by a simple boolean guard, which ensures consistent behavior during a session and avoids silent increases in token consumption. These measures are deliberately modest but effective for an interactive tool.

3.2.8 Instrumentation and linkage to investigation

The application records a compact set of run-time fields that support the investigative techniques defined earlier. For each run it logs the dataset and prompt identifiers, the model used, timings for code generation and code execution, the outcome of the sandbox scan, the presence of the required function, the classification of the figure object, and the result of numeric summary validation. For insight generation it logs the response length and whether the text includes numeric references to the summary. These records are sufficient to compute compile and render success rates, schema compliance, latency distributions, and sandbox violation counts, and they can be exported to a CSV for analysis alongside the rubric-based human scores. By keeping the instrumentation within the application, the evaluation can be repeated without external harnesses.

3.2.9 Security, privacy, and configuration

The solution minimizes the data sent to the model and limits what generated code can do. Only the head of five rows and a compact dtype description are transmitted once. Insight generation uses only the numeric summary, which contains aggregated values rather than raw records. Generated code runs inside the process but with a constrained global environment and a ban on operations that could access the file system or network. The Gemini API key is supplied at runtime through an environment variable or a configuration secret, not hard-coded in source files. Model names are configurable through the interface, and defaults are set to Gemini 2.5 variants. These practices are straightforward but adequate for a student project with attention to privacy and safety.

3.2.10 Limitations, trade-offs, and future extensions

The solution deliberately trades completeness of context for speed and cost by sending only the first five rows. This can lead the model to miss rare categories or data issues that are not present in the head. The contract and sandbox reduce risks but cannot prevent all semantic errors, for example requesting a chart for a non-existent column that resembles an existing one. The insight generator is limited by the information content of the numeric summary; it will not detect relationships that require row-level analysis. In future iterations, the application could offer optional context expansion under user control, a small library of validated chart templates for common tasks, and a richer summary schema that includes simple confidence intervals or cross-tabulations when appropriate. None of these are required for the proposed solution to meet its goals, but they indicate a natural growth path.

3.2.11 Concluding remarks

The proposed solution is a compact and disciplined artifact that brings together automated EDA, prompt-to-code generation under a strict contract, guarded execution, numeric summarization, and concise insight drafting. The architecture emphasizes clear boundaries between concerns, minimal data transmission to the model, and machine-checkable validations at every step. The logging built into the flow provides the evidence base for the investigative techniques defined in the previous subsection and allows the project to be iterated in short, justified cycles. In this form, the solution is both practical for end users and suitable for an academic evaluation.

3.3 Work Breakdown Structure

This subsection presents the work breakdown structure for the project and explains the concrete modules and interim products that together deliver the complete solution. The structure is organized as a sequence of phases with clearly scoped work packages. Each work package is described in terms of purpose, inputs and outputs, internal responsibilities, dependencies on other packages, completion criteria, and risks. The description is intentionally implementation-oriented and avoids restating the investigative method or high-level goals that appear elsewhere in the report.

3.3.1 Structure and rationale

The work is divided into eight phases. Phase 0 covers set-up. Phases 1 to 5 build the core pipeline from CSV intake to chart code and business insights. Phase 6 formalizes telemetry and quality assurance so that the investigative techniques in Section 3.1 can be executed without additional tooling. Phase 7 concerns packaging, documentation, and release. This ordering respects the technical dependencies between components. For example, code generation cannot be evaluated before the execution sandbox and contract validator exist, and insight generation cannot proceed before the numeric summary contract is in place.

3.3.2 Phase 0: Environment and governance (W0)

This initial work package prepares the development environment and the basic project governance so that the remainder of the work proceeds predictably. The package configures the Streamlit application scaffold, a separate Colab workspace for experimentation if required, and a minimal Python environment file to lock library versions. The Gemini API configuration is externalized either through environment variables or Streamlit secrets to avoid hard-coding credentials. A small repository structure is defined that separates user interface code from model access, sandbox execution, EDA helpers, and validation utilities. Completion is achieved when the empty Streamlit app runs on the target machine, the API key is read securely at runtime, and a placeholder page renders without errors. This package is foundational because it reduces later friction when integrating modules that rely on identical library versions and paths.

3.3.3 Phase 1: Data intake and automated EDA (W1)

The first functional package introduces CSV ingestion and an embedded automated EDA report. The user is able to upload a CSV through the interface. The system parses the file into a DataFrame, computes the head of five rows, and extracts a mapping of column names to inferred data types. The EDA module creates a profiling report in HTML using a minimal exploratory configuration; the report is rendered inside the application without involving the language model. The head and the dtype mapping are cached in session state and a boolean flag is initialized to indicate that the head has not yet been sent to the model. The package is complete when a user can upload a CSV, view a compact profiling report, and observe that the application is holding the five-row head and type information ready for later stages. The principal risks at this point are incorrect type inference for date columns and memory pressure for very large files; the mitigation is to keep the profiling configuration minimal and to alert the user when files exceed a reasonable size.

3.3.4 Phase 2: Prompting contract and code generation interface (W2)

The second functional package defines the interaction between the application and the language model for chart code generation. The system instruction text is written so that the model is required to return code only and to define a function named build_chart(df) that returns a figure object and a JSON-serializable numeric summary. The user interface provides a text area where the user states the chart request in plain language. On the first request of a session, the application constructs a message that includes the five-row head and the dtype mapping. On subsequent requests, the head and types are not included and only the new user message is sent. A simple retry mechanism wraps the model call to handle transient failures; there is an optional fallback from a Pro model variant to a Flash variant if the first attempt is rate-limited. The package is considered complete when the application can send a prompt and receive text that appears to be Python code that conforms to the contract. This package deliberately does not execute the returned code. It establishes the boundary and messaging conventions that later packages rely upon.

3.3.5 Phase 3: Execution sandbox and contract validator (W3)

This package introduces controlled execution of the generated code. Before any dynamic execution, the application inspects the returned text for forbidden patterns. The patterns include import statements, file operations, operating system or system module access, dynamic evaluation primitives, and double-underscore identifiers. If any are present, execution is refused and a clear error message is shown. If the text passes this check, it is executed in a restricted global context that exposes only a small set of safe built-ins together with whitelisted plotting and data libraries. After execution the application verifies that a callable symbol named build_chart exists and that invoking it with the full DataFrame returns a two-element structure. The first element must be recognized as a figure by one of the allowed libraries. The second element must serialize to JSON and conform to the declared schema for the chart family implied by the user's request. Completion criteria for this package are the ability to reject unsafe code deterministically, to execute safe code without leaving the sandbox, and to produce machine-checkable pass or fail results for the function contract. This package is critical because it converts an informal prompt into a formal computational artefact with clear acceptance tests.

3.3.6 Phase 4: Rendering, editable code panel, and numeric summary pipeline (W4)

The fourth package connects contract-compliant code to a user-visible figure and to a structured numeric summary. When a run passes the validator, the application renders the figure in the interface using the appropriate library. The returned code is also placed in an editor panel where a user can make minor corrections such as fixing a column name and then rerun the chart locally without calling the model again. The numeric summary is displayed in the interface in JSON form and is cached in session state for use by the next package. The package concludes when a user can run a request, see a chart, inspect or edit the underlying code, and view a valid numeric summary that matches the required shape. The package is also responsible for ensuring that runs that fail at any stage surface errors clearly and leave the application in a stable state so that a new request can be tried without restarting the session.

3.3.7 Phase 5: Insight generation from numeric summary (W5)

This package introduces the second and final use of the language model: drafting concise business insights based solely on the numeric summary. The system instruction used here defines the model as a short, evidence-oriented analyst and requires that the output contain a small number of specific observations with sensible next steps. The application sends only the numeric summary and a brief pointer to the user's intent; raw rows are not transmitted. The response is displayed under the chart so that the user can connect the suggested actions to the visual and the numbers that generated them. Completion is achieved when the application can reliably produce insight text that references values in the summary and avoids claims that are not supported by the data. Failures are logged like any other run. The dependency on Phase 4 is strict because the insight module requires a valid summary in the expected schema.

3.3.8 Phase 6: Telemetry and quality assurance harness (W6)

At this stage the application is functional. The present package equips it with the logging necessary to execute the investigative plan without external scripts. The system captures, for every run, a timestamp, a dataset identifier or hash, a prompt identifier, the model used, whether the head and type information were included, the size of the messages, the time taken for model response and for local execution, the result of the forbidden-token scan, the presence and callability of build_chart, the classification of the figure object, the result of summary validation, and any error classes. Insight requests record the length of the returned text and whether numeric references are present. A simple export produces a CSV of these records for later aggregation. The package is complete when the application produces a log row per run, when error classes are specific enough to regroup failures meaningfully, and when the data can be joined to the small rubric-based human ratings collected during evaluation.

3.3.9 Phase 7: Packaging, documentation, and release (W7)

The final package prepares the application for delivery as a student project. The repository is organized into a conventional structure with modules mapped to the packages described above. The README explains how to set up an environment, how to provide the API key safely, and how to run the application locally. The report includes a short user guide that describes the upload, EDA, chart request, code editing, and insights flow. Configuration points such as the model's name and the head size are exposed in a sidebar or a configuration file. A lightweight license and acknowledgements are added if required by institutional policy. Completion is reached when a fresh clone can be set up and run following the documented steps on a lab machine without private assistance.

3.3.10 Discussion of workable modules and products

The breakdown above produces a set of modules that can be developed and tested in relative isolation and then composed. The data intake and EDA module is a self-contained unit that turns a CSV into a DataFrame and an HTML report and exposes only three simple artefacts to the rest of the system: the DataFrame, the five-row head, and the dtype mapping. The code generation module is a thin adapter over the model API whose responsibility is to format prompts correctly and to return raw code text without attempting to execute it. The execution module is built as a small framework with two responsibilities: to reject unsafe code and to execute safe code in a restricted environment. The validator within it is deliberately deterministic so that there is no ambiguity over pass and fail. The rendering and summary module draws charts and exposes the returned numeric structure to the next stage without altering it, which keeps concerns separated. The insight module uses the model but does not need access to the DataFrame and therefore cannot unintentionally transmit rows. The telemetry harness is a thin cross-cutting module that records fields from the earlier stages without changing their behavior. Together these modules form a workable product in three senses. First, the application as a whole is a usable tool where a non-technical user can move from a CSV to a chart and insights in one screen. Second, the executor-validator component is reusable in other projects that require safe execution of small code snippets. Third, the Colab notebook variant is a self-contained demonstration that shows the pipeline outside Streamlit, which is valuable for testing and for examiners who wish to inspect the logic through cells.

3.3.11 Milestones, dependencies, and acceptance conditions

The acceptance of Phase 0 depends on the ability to run an empty application and to read configuration from the environment. Phase 1 is accepted when a CSV can be uploaded and profiled and when the five-row head and dtype mapping are visible in session state for debugging. Phase 2 is accepted when a prompt with head and dtypes returns a block of code that appears to conform to the contract and when subsequent prompts omit the head. Phase 3 is accepted when the application consistently rejects unsafe code and executes safe code to the point of function detection and tuple return without crashing the process. Phase 4 is accepted when the rendered chart is visible and the numeric summary can be displayed and saved and when the editable code panel allows a local rerun with user adjustments. Phase 5 is accepted when a numeric summary can be turned into insight text that references numbers. Phase 6 is accepted when runs produce log rows that contain all agreed fields and can be exported. Phase 7 is accepted when a fresh set-up reproduces the behavior using the instructions. These conditions are stated in this form to allow a straightforward demonstration of completion during evaluation.

3.3.12 Risks, constraints, and contingency

There are three risks that affect multiple packages. The first is instability in the language model interface or quota limitations during testing. The contingency is to allow a model fallback and to implement a small retry wrapper around calls. The second is unanticipated data quality issues such as non-UTF-8 files or mixed types inside a column that frustrate plotting. The mitigation is to keep the profiling report visible and to add small helper messages prompting users to resolve parse errors. The third is the possibility that the model returns code that compiles but produces a semantic error due to an incorrect column name that was not present in the five-row head. The mitigation is the editable code panel and, if necessary, a normalization pass for column names that can be toggled in configuration. None of these risks threaten the overall delivery if they are identified early and addressed within the relevant phase.

3.3.13 Effort allocation and schedule narrative

The schedule follows the phase order and allocates effort to reflect dependencies. The first week is dedicated to Phase 0 so that environment assumptions are stable. The second and third weeks cover Phase 1 and Phase 2 together because they share user interface plumbing. The fourth week is used for Phase 3, which is technically sensitive and benefits from focused attention. The fifth week completes Phase 4 and includes user interface polishing to ensure that the editable code panel is comfortable to use. The sixth week implements Phase 5 and begins to run small end-to-end tests. The seventh week builds Phase 6 and executes the first formal evaluation runs to verify that the instrumentation meets the investigative needs. The eighth week completes Phase 7, freezes versions, and prepares the demonstration script and documentation. This sequence leaves time for one short design-science iteration if evaluation in week seven suggests a prompting or validator change. The narrative schedule is intentionally simple so that it can be followed in an undergraduate timetable while still producing an artefact that satisfies the acceptance conditions defined earlier.

In conclusion, the work breakdown structure specifies a practical path from an empty scaffold to a usable pipeline with safe code execution, numeric summarization, and insight generation. Each phase produces a tangible outcome that enables the next one and can be verified independently. The modules created are small and cohesive, and the interim products are directly usable for demonstration and testing. This disciplined breakdown reduces integration risk and ensures that the investigative techniques can be carried out without additional engineering.

3.4 Tools and Technology

The proposed system integrates a diverse set of technologies across the frontend, backend, machine learning, data visualization, and storage layers. Each tool has been selected based on its compatibility, scalability, and relevance to the modular architecture of the pipeline. The following categorization details the tools and frameworks used along with their roles in the overall implementation.

1. Frontend Technologies

- **React.js :** React is used for building the user interface of the web application where users can upload CSVs, view visualizations, and interact with generated insights. Its component-based architecture allows modular UI development, essential for the dynamic visualization editor.
- **Tailwind CSS :** Tailwind provides a utility-first CSS framework that helps in rapidly developing responsive and customizable UI components. It is particularly useful for styling the editable code panel, chart viewers, and results layout.
- **JavaScript / HTML :** JavaScript forms the foundation for dynamic user interaction logic within the React components, while HTML handles the semantic structure. Together, they are responsible for integrating UI controls such as chart rebuild buttons and file uploads.

2. Backend and Processing Stack

- **Python :** Python acts as the core backend language used for orchestrating the pipeline. It handles CSV parsing, automated EDA, LLM interactions, code execution, chart rendering, and summary data extraction.
- **Streamlit :** While the web-facing interface is available via React, Streamlit is used in standalone development to rapidly prototype and test the flow of input → LLM → output. It helped test LLM-to-chart feedback loops before full frontend integration.
- **Google Generative AI API (Gemini 2.5) :** Gemini 2.5 is used for natural language understanding and code generation. It interprets the user's charting query, processes the data frame head, and generates Python code accordingly. It is also used for deriving insights based on the chart summary.

- **Colab (initial development) :** Google Colab was used in the early stages for EDA logic, LLM validation, and summary extraction before porting the logic to a production-grade pipeline.

3. Data Storage

- **MongoDB :** MongoDB is the NoSQL database used to persist uploaded CSV metadata, generated charts, user queries, summaries, and corresponding insight responses. It helps in maintaining user session history and allowing insight regeneration.
- **Cloudinary :** Cloudinary is a cloud-based platform for managing, transforming, and delivering images and videos for websites/apps, automating the media lifecycle from upload to optimized delivery via a fast CDN, saving developers time and improving user experience through AI-powered features like automatic resizing, format conversion, and enhancements, all accessed via powerful APIs.

4. Visualization and Summary Extraction

- **Plotly Express / Altair / Matplotlib :** These libraries are used for rendering interactive and static charts based on LLM-generated code. Plotly is preferred for interactivity in React. Summary data (like frequency of bins or counts per category) is programmatically extracted post-rendering for insight generation.
- **Pandas / NumPy :** These are foundational Python libraries for data analysis and preprocessing. They handle CSV reading, head extraction, cleaning, and all chart-specific computation.

4.1 System Architecture

An architecture diagram visually represents the overall structure of a system or application, showing the components, modules, and their interactions. It provides a high-level overview of how different parts of the system communicate, including user interfaces, databases, servers, and external services. Architecture diagrams help in understanding the system's layout, data flow, and integration points. They are essential in planning and designing complex systems, ensuring that all components work together efficiently and meet technical and business requirements. These diagrams guide the development process, helping stakeholders visualize the system's structure and identify potential improvements. We have shown the details in the Fig 4.1.

Key Components

- Generic User / File UI - uploads datasets (CSV/Excel) and submits queries.
- Input Data → Extracted Data - parsers & ETL that convert files into structured JSON/tabular output.
- Automated EDA - quick profiling (missing values, distributions, correlations), candidate charts and summaries.
- JSON Data - normalized structured representation consumed by downstream components.
- Prompt Builder - assembles user query + JSON summary into a prompt for the LLM.
- LLM - generates plotting code (e.g., matplotlib/plotly), textual BI insights, suggested transformations.
- Code of Plot → Plot Generation - executes generated plotting code (in a sandbox) and produces images/interactive artifacts.
- BI Insights / Report - composes EDA results + LLM explanations + rendered plots into a final report.
- Database - persists raw files, extracted JSON, generated code, rendered artifacts, and prompt/response logs.

- Response - final package delivered to user (plots, reports, textual insights).

Typical Interaction Steps

- User uploads file → system extracts structured JSON and stores raw file.
- Automated EDA produces summaries and candidate visuals.
- Prompt Builder combines query + JSON summary and calls the LLM.
- LLM returns plotting code + textual insights.
- Code executes in sandbox → rendered plots stored.
- Report composer bundles plots + insights and sends to user.
- All artifacts stored for reproducibility and auditing.

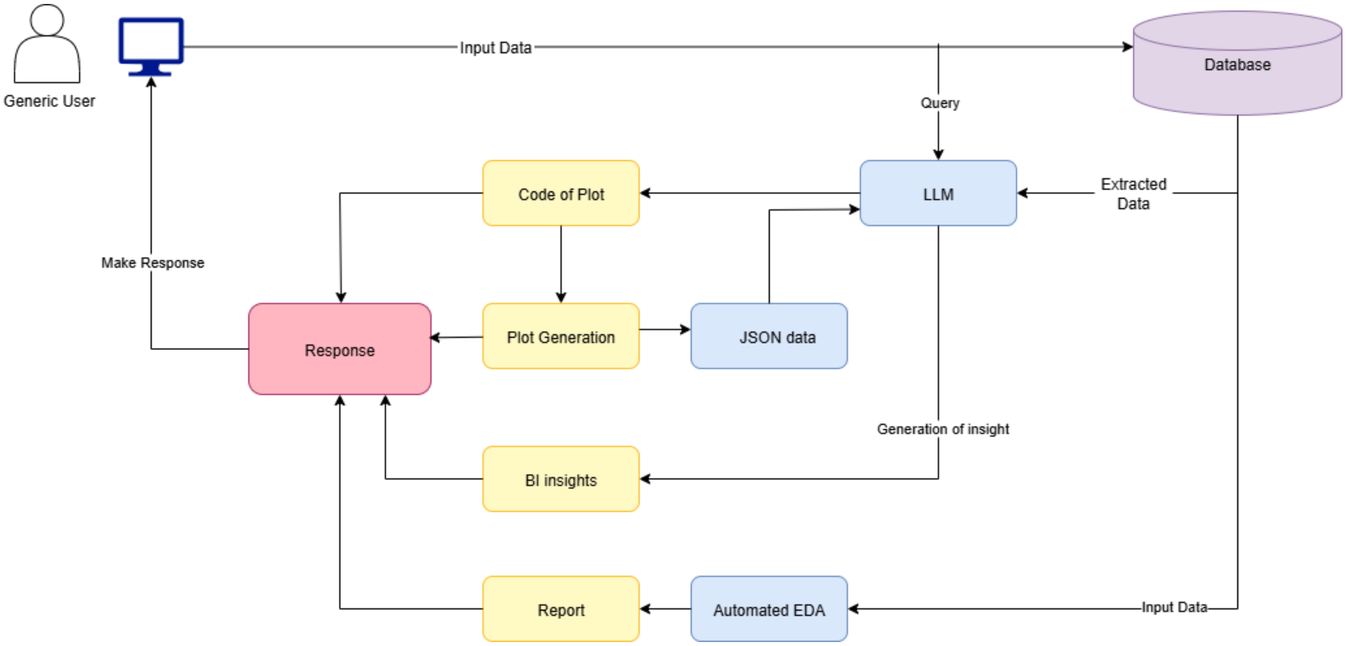


Fig 4.1: System Architecture

4.2 Design Level Diagrams

4.2.1 DFD

A DFD (Data Flow Diagram) is a graphical representation of how data flows through a system and how it is processed.

- Processes → shown as circles/ellipses (represent transformations of data)
- Data Stores → shown as open-ended rectangles (represent storage points)
- External Entities → shown as ovals (represent actors outside the system)
- Data Flows → shown as arrows (represent movement of data between components)

The purpose of a DFD is to provide clarity on how inputs are converted into outputs, what data is needed, and how different parts of the system interact.

Level 0 (Context Diagram)

- Provides the highest-level view of the system, often called the context diagram.
- Shows the system as a single process (P0) with interactions between external entities (e.g., Business User, LLM, File Storage).
- At this stage, the focus is on system boundaries and the major data exchanges without going into detail.
  In the given Level 0 DFD as in Fig 4.2.1.1, the LLM-Driven BI Platform receives user inputs (datasets, queries, feedback), interacts with LLM for insights, stores/retrieves files, and returns outputs (visualizations, insights).

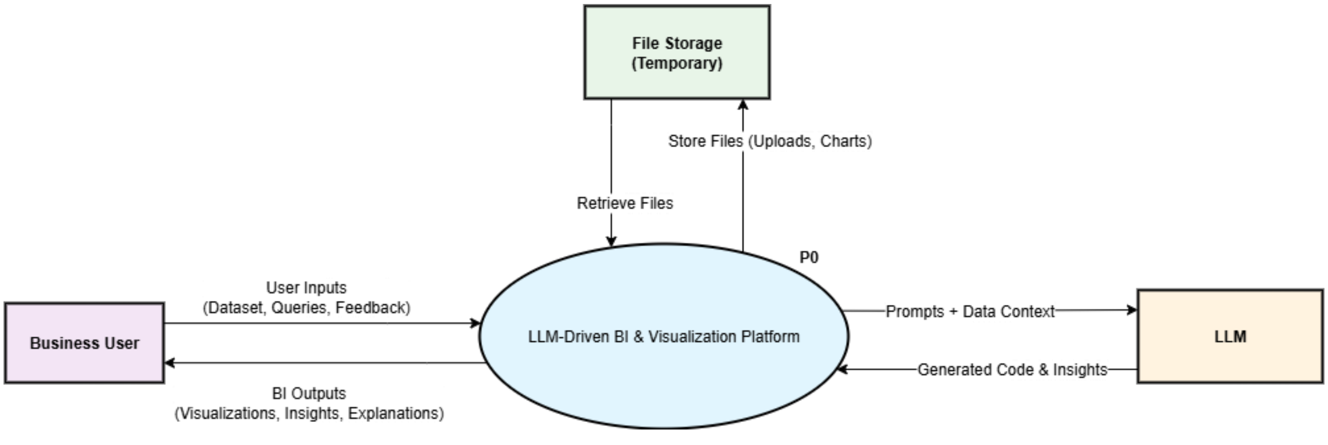

File Storage
(Temporary)
Store Files (Uploads, Charts)
Retrieve Files
P0
User Inputs
(Dataset, Queries, Feedback)
Business User
LLM-Driven BI & Visualization Platform
Prompts + Data Context
LLM
Generated Code & Insights
BI Outputs
(Visualizations, Insights, Explanations)

Fig 4.2.1.1: DFD Level 0

Level 1 (Decomposition of P0)

- Breaks down the single process (P0) into sub-processes (P1, P2, P3, P4).
- Shows **major system functions** such as:

- **P1:** Manage Data Input (handling dataset uploads, storing data)
- **P2:** Process User Interaction (queries, feedback, session context)
- **P3:** Generate BI Outputs (prepare insights, visualizations)
- **P4:** Present Results (deliver BI outputs back to users, store charts)
- Highlights **data stores** (Storage, Session Context) and their role in maintaining state. In the given Level 1 DFD (Fig 4.2.1.2), the system is shown in more detail: data is uploaded, stored, processed by the LLM, and then transformed into BI outputs.

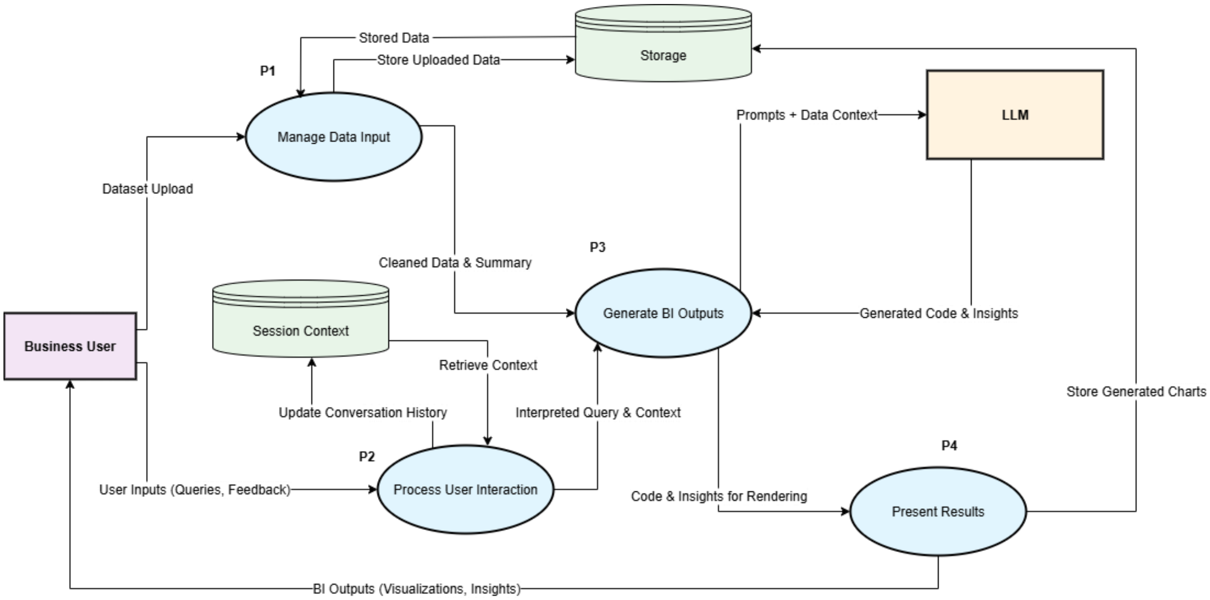


Fig 4.2.1.2: DFD Level 1

Level 2 DFD – P1: Manage Data Input

This diagram, shown in Fig 4.2.1.3, expands **Process 1 (Manage Data Input)** into detailed sub-processes. The workflow begins when a **Business User uploads a dataset** (CSV/Excel).

- **1.1 File Upload & Validation** – checks if the file is valid. Errors are sent back to the user.
- **1.2 Data Parsing & Format Detection** – interprets the file structure (CSV, Excel, etc.).
- **1.3 Data Cleaning & Preprocessing** – removes inconsistencies and prepares the dataset.
- **1.4 Schema Analysis & Type Detection** – identifies data types and table schema.

- **1.5 Statistical Summary Generation (EDA)** – produces descriptive summaries.
- **1.6 Data Quality Assessment** – checks completeness, accuracy, and consistency.

Supporting data stores include:

- **Raw Data Buffer** (original uploaded files)
- **Data Metadata Buffer** (schema, types, and structural info)
- **Quality Reports Buffer** (data profiling results)

Finally, cleaned and validated data flows into **P3: Generate BI Outputs** for analysis and visualization.

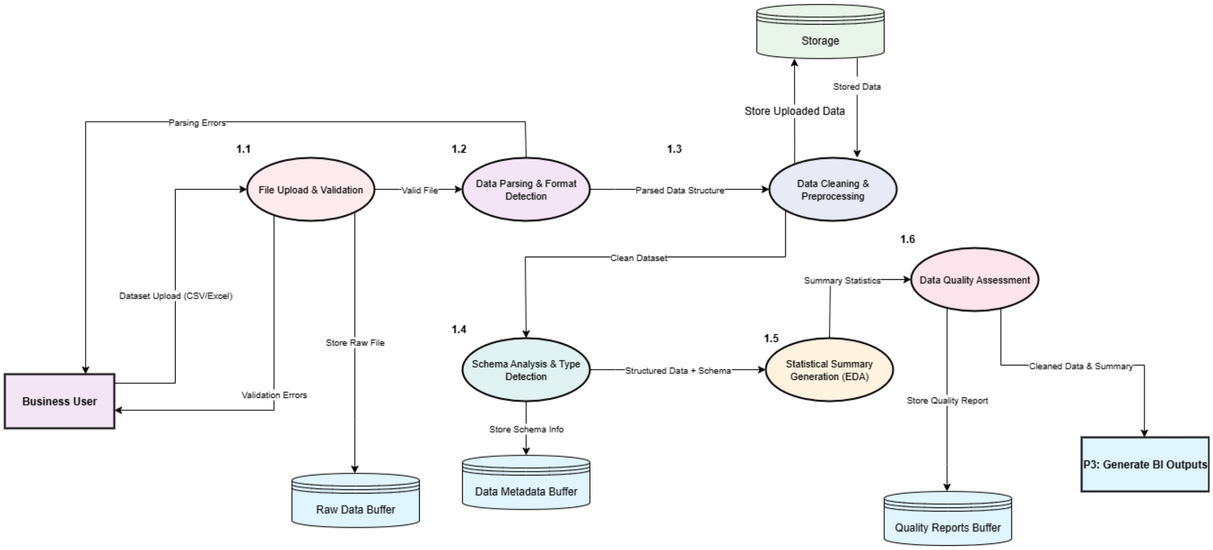


Fig 4.2.1.3: DFD Level 2 (Process 1)

Level 2 DFD – P2: Process User Interaction

This diagram decomposes Process 2 (User Interaction)**, showing how user queries and feedback are handled.** (Fig 4.2.1.4)

- 2.1 Validate & Sanitize User Input **– checks for unsafe or malformed queries; errors are returned to the user.**
- 2.2 Augment with Session Context **– enriches the query using past interactions stored in the** Session Context datastore**.**

- 2.3 Structure Query for Generation **– reformats the query into a system-friendly representation.**
- 2.4 Update Session Context **– logs the interaction back into** Session Context **for future continuity.**

The result is a context-aware interpreted query**, which is passed to** P3: Generate BI Outputs **for LLM-driven analysis and visualization.**

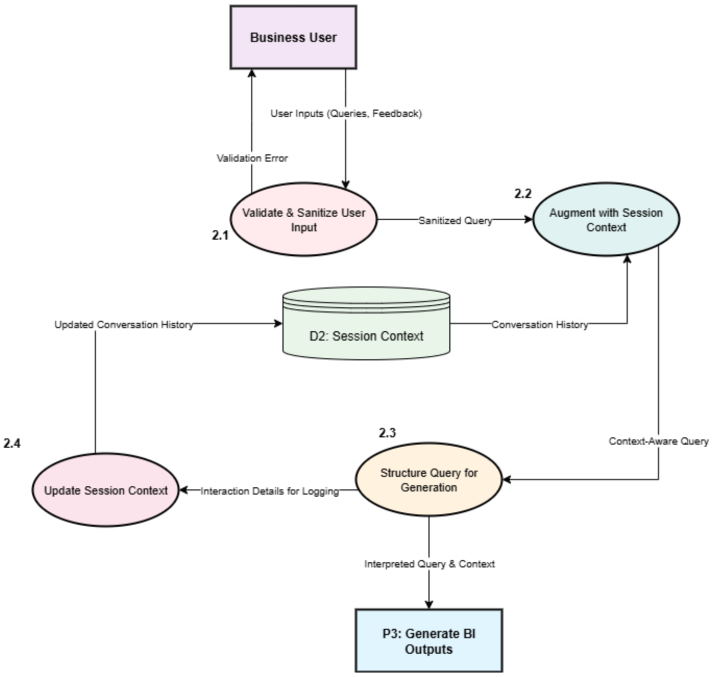

Business User
User Inputs (Queries, Feedback)
Validation Error
2.1
Validate & Sanitize User Input
Sanitized Query
2.2
Augment with Session Context
Updated Conversation History
D2: Session Context
Conversation History
Context-Aware Query
2.3
Structure Query for Generation
2.4
Update Session Context
Interaction Details for Logging
Interpreted Query & Context
P3: Generate BI Outputs

Fig 4.2.1.4: DFD Level 2 (Process 2)

Level 2 DFD – P3: Generate BI Outputs

This diagram (Fig 4.2.1.5) details how the system converts user queries and prepared datasets into meaningful business intelligence outputs.

- **3.1 Query Parsing & Intent Recognition** – interprets the structured query (from P2) and maps it to analysis tasks.
- **3.2 Data Retrieval & Transformation** – fetches relevant cleaned data from the **Data Metadata Buffer** and applies transformations.
- **3.3 Analytical Computation** – performs statistical analysis, aggregations, or computations required for BI.

- **3.4 Visualization Generation** – creates charts, tables, or dashboards.

Supporting data stores:

- **Processed Data Repository** – stores cleaned and transformed datasets.
- **BI Reports Repository** – stores generated dashboards, summaries, and visualizations for retrieval.

The outputs from this process flow to the **Business User** through the interface and may also be passed to **P4 (LLM Communication)** for natural language explanation.

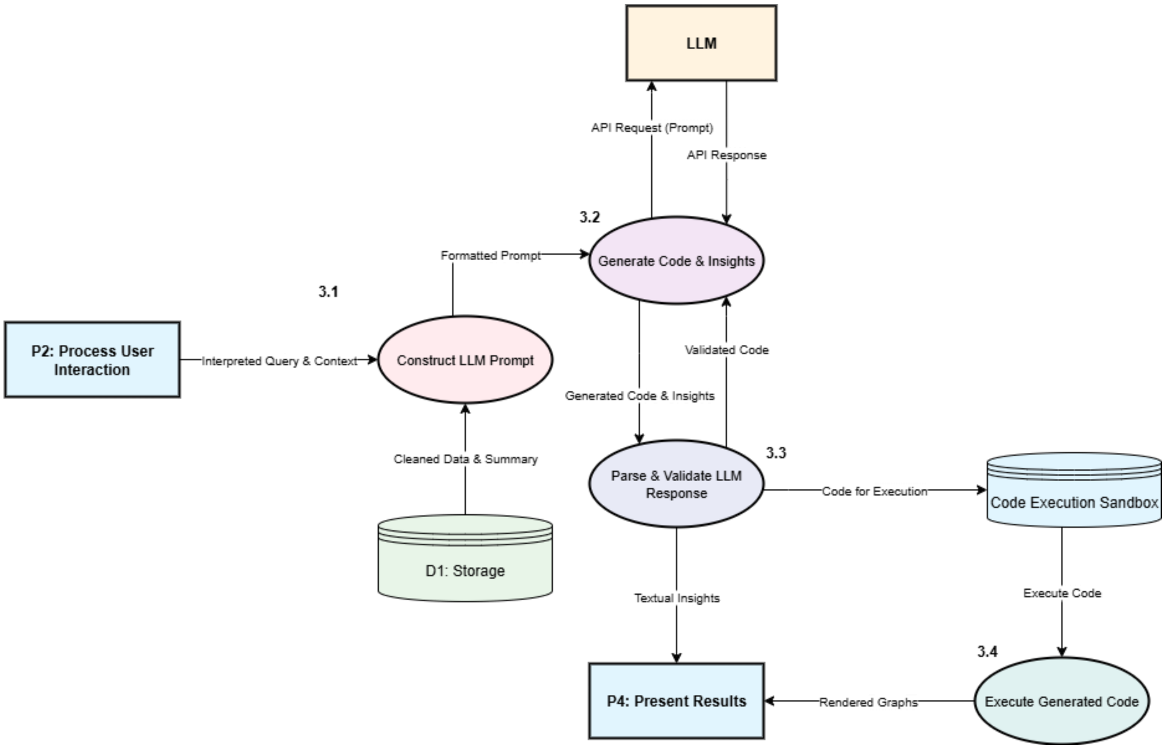


Fig 4.2.1.5: DFD Level 2 (Process 3)

Level 2 DFD – P4: Manage Communication with LLM

This diagram (Fig 4.2.1.6) shows how the system interacts with the Large Language Model (LLM) to provide natural language summaries, query support, and conversational responses.

- 4.1 Unpack & Route Data

- Parse the incoming payload (visual spec, text insights, assets).
- Choose the proper renderer (chart/table/narrative panel).

- Stream/render outputs to the Business User.
- Determine what artifacts should be persisted (e.g., chart snapshots, specs, exports) and pass those Items to Store to 4.2.

- 4.2 Save BI Results
  - Receive “Items to Store” from 4.1 (e.g., generated charts, visualization specs, export files, thumbnails).
  - Create metadata (owner, timestamp, version, lineage to source query).
  - Persist to D1: Storage via “Store Generated Charts”.

Supporting data stores:

- LLM Communication Logs – **maintains all prompt–response pairs for transparency and debugging.**
- Session Context **(shared with P2) – helps maintain continuity in multi-turn conversations.**

Thus, P4 bridges the gap between structured BI results (P3) **and** natural, conversational insights **that business users can easily understand.**

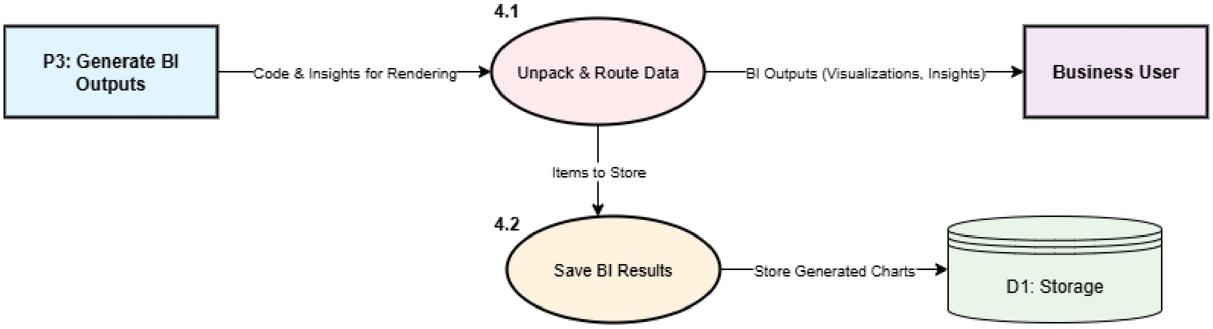


Fig 4.2.1.6: DFD Level 2 (Process 4)

4.2.2 Class Diagram

A class diagram is a type of UML (Unified Modelling Language) diagram that provides a visual representation of the structure and relationships among classes within a system or software application. Fig 4.2.2.1 illustrates the attributes, methods, and associations associated with each class, showcasing how objects in the system interact and collaborate.

Class diagrams are fundamental for modelling the static structure of a system, aiding in the understanding and design of object- oriented systems

Classes (attributes & methods)

User

- Attributes
- userId: String - unique identifier for the user.
- name: String - display name of the user.
- Methods
- sendRequest(request: Query): void - submit a query or upload request to the platform.
- Responsibility
- Initiates uploads and queries; receives responses and attachments.

LLMService

- Attributes
- modelName: String - name of the model in use (e.g., "gpt-x").
- version: String - model version or revision.
- Methods
- loadModel(): void - warm up or initialize the model/client.
- generateResponse(q: Query): Response - produce textual answers based on query + context.
- generatePlotCode(data: DataSet): PlotCode - produce plotting/code snippets given data and intent.
- Responsibility
- Central orchestrator for natural language understanding and generation; produces code and textual insights.

DataStore

- Attributes
- connectionString: String - DB/endpoint configuration.
- Methods
- save(obj: Object): bool - persist an object (raw file, JSON, prompt, code, artifact).
- fetch(q: Query): Object - retrieve stored objects based on query criteria.
- update(id: String, obj: Object): bool - update stored object.
- Responsibility
- Persistence layer for provenance, prompt-response logs, generated artifacts and datasets.

InputData

- Attributes
- rawPayload: String - file contents or raw text.
- source: String - origin (upload, API, connector).
- timestamp: Date - upload time.
- Methods
- validate(): bool - check format, size, basic integrity.
- toJSON(): JSON - convert raw payload to an intermediate JSON representation.
- Responsibility
- Validate and provide the initial representation for ingestion.

ExtractedData

- Attributes
- fields: Map<String, Object> - parsed columns/fields and sample values.
- schemaVersion: String - schema signature / version.

- Methods
- normalize(): DataSet - convert parsed fields into a normalized tabular dataset.
- toJSON(): JSON - provide a JSON-serializable representation.
- Responsibility
- Structured representation produced by parsers/ETL from InputData.

JSONData

- Attributes
- payload: Object - JSON object containing data and metadata.
- Methods
- keys(): List<String> - list top-level keys.
- prettyPrint(): String - readable JSON string for prompts or logs.
- Responsibility
- Portable, LLM-friendly data context passed into prompt builder / LLMService.

PlotCode

- Attributes
- codeSnippet: String - generated code (matplotlib/plotly/altair/etc.).
- language: String - language/runtime for execution (python, js, …).
- Methods
- execute(): Plot - run the snippet in a sandbox and return a plot/artifact.
- validate(): bool - static checks for safety (forbidden imports, infinite loops, resource use).
- Responsibility
- Encapsulates generated visualization code and its lifecycle.

PlotGenerator

- Attributes
- renderer: String - engine used to render (e.g., headless browser, matplotlib).
- Methods
- generate(pc: PlotCode): Plot - execute code and produce a plot object/image.
- export(format: String): File - export the rendered plot to PNG/SVG/interactive format.
- Responsibility
- Safe execution and rendering of PlotCode into deliverable artifacts.

AutomatedEDA

- Attributes
- rules: List<Rule> - EDA heuristics (outlier detection, missing-value rules).
- Methods
- run(data: DataSet): Report - generate an automated EDA Report.
- detectOutliers(): List<Record> - list records flagged as outliers.
- Responsibility
- Fast profiling of ExtractedData to generate summaries, candidate visuals, and data quality reports.

Report

- Attributes
- title: String
- contents: String - textual/HTML content; can include references to images.
- Methods
- generatePDF(): File - render the report as a downloadable PDF.
- summary(): String - short textual summary for previews.

- Responsibility
- Packaged EDA or BI report combining stats, charts and narrative.

BIInsights

- Attributes
- insights: List<Insight> - structured insights derived from analysis and LLM outputs.
- Methods
- generateFrom(data: DataSet): List<Insight> - derive insights from data and analysis.
- toDashboard(): Dashboard - convert insights into a dashboard representation.
- Responsibility
- Convert analysis + LLM output into human-interpretable insights and dashboard-ready items.

Response

- Attributes
- text: String - primary textual message sent to the user.
- attachments: List<File> - plots, PDFs, or other generated files.
- status: String - e.g., ok, error, partial.
- Methods
- send(to: User): bool - deliver the response to the user's session/endpoint.
- Responsibility
- Final packaging of outputs delivered to the user.

Relationships (flow-oriented - labels & direction)

- User --> LLMService : "sends Query / interacts with" (user initiates request that LLMService processes)
- InputData --> ExtractedData : "extracts" (ingestion pipeline parses raw uploads)
- ExtractedData --> JSONData : "transforms to" (structured representation → JSON context)
- JSONData --> LLMService : "provides context" (used inside prompts)
- LLMService --> DataStore : "read / write" (persist prompts, responses, artifacts)
- LLMService --> PlotCode : "generates" (LLM outputs plotting code)
- PlotCode --> PlotGenerator : "executes" (code runs in renderer/sandbox)
- PlotGenerator --> Response : "produces plot attachment" (rendered image attached to response)
- AutomatedEDA --> Report : "generates" (EDA produces report artifacts)
- AutomatedEDA --> ExtractedData : "analyses" (uses parsed data)
- DataStore --> AutomatedEDA : "reads historical data" (EDA can leverage historical datasets)
- LLMService --> BIInsights : "produces insights" (LLM assists in insight extraction)
- BIInsights --> Response : "attaches insights" (insights included in final response)
- LLMService --> Response : "direct textual response" (LLM may directly generate the text of the response)

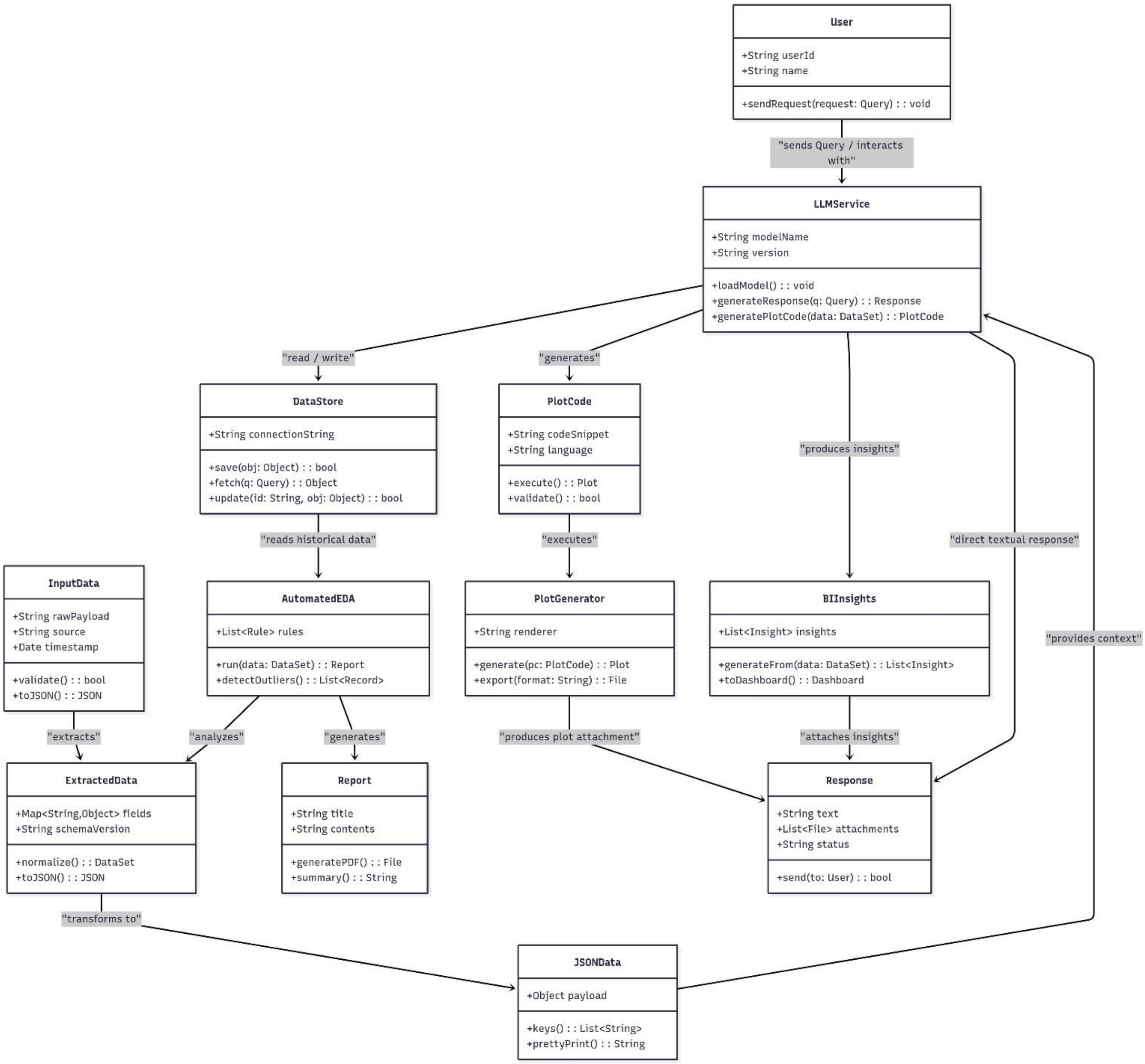


Fig 4.2.2.1: Class Diagram

4.3 User Interface Diagrams

4.3.1 Use Case Diagram

A use case diagram is the primary form of system/software requirements for a new software program underdeveloped. Use cases specify the expected behaviour (what), and not the exact method of making it happen (how). Fig 4.3.1.1 is an effective technique for communicating system behaviour in the user's terms by specifying all externally visible system behaviour.

Actors

- Business User **-** uploads datasets, types queries, views reports and visualizations, and gives feedback.
- LLM **-** external/embedded language model used by the system to generate code, insights and explanations.
- System **(**implicit**) -** the automated components that capture, preprocess, route queries and generate responses.

Use Cases (name, primary actor, description)

- Upload Dataset · *Business User*

  User uploads CSV / Excel / files. System parses and stores the raw input for downstream processing.

- Enter Query · *Business User*

  User types a natural-language query (e.g., “show sales by region”). This initiates query capture and processing.

- View BI Report · *Business User*

  User views the generated BI report which bundles visualizations, EDA summaries and textual insights.

- View Visual Charts · *Business User*

  User inspects rendered charts or interactive visualizations produced by the system.

- View EDA Report · *Business User*

  User inspects automated EDA (summary statistics, missing values, correlations).

- Give Feedback · *Business User*

  User rates or comments on the response/visualization; feedback is stored for improvement.

- View Chat History · *Business User*

  User can browse previous queries and responses for continuity and reproducibility.

- Process User Query · *System*

  Captures and preprocesses the query (validation, sanitization, basic intent extraction) before handing off for generation.

- Generate BI Outputs · *System / LLM-assisted*

  Produces analysis outputs (textual insights, charts, code). This is the umbrella use case that coordinates generation.

- Generate Visual Chart Code · *System / LLM*

  Produces plotting code (e.g., matplotlib/plotly snippets) from the interpreted query + data context.

- Visual charts → JSON (Visual charts to JSON) · *System*

  Converts or serializes visuals and metadata into JSON (for storage, reproducibility, or re-rendering).

- EDA Report Generation · *System*

  Creates automated EDA reports (statistics, profiling), often triggered after ingestion.

- (Admin) Upload & Update Data / Manage Data · *Admin*

  Admins add, update or remove documents/datasets that the system uses.

- (Admin) View Feedbacks & Suggestions · *Admin*

  Admin reviews user feedback to drive improvements or curate knowledge.

Relationships (associations, includes, extends)

- Associations (actor ↔ use case)

- ▪ Business User → Upload Dataset, Enter Query, View BI Report, View Visual Charts, View EDA Report, Give Feedback, View Chat History.
- ▪ Admin → Upload & Update Data, View Feedbacks & Suggestions.
- ▪ LLM ↔ Generate BI Outputs / Generate Visual Chart Code (LLM is used by these use cases).

- Includes (<<includes>>)

- Enter Query **<<includes>>** Process User Query **(diagram: include-process-query) - query entry triggers capture & preprocessing.**
- View BI Report **<<includes>>** Generate BI Outputs **- viewing a report implies the system generated that output.**
- Generate BI Outputs **<<includes>>** Generate Visual Chart Code **generation is composed of these subprocesses.**

Typical (Primary) Scenario - step-by-step

- **Upload**: Business User uploads a dataset → system extracts tables → EDA Report Generation runs and raw + JSON extracted data are stored.
- **Query**: User enters a query → Process User Query validates & sanitizes and augments with session context.
- **Generation**: Prompt builder (system) assembles query + JSON summary and calls the LLM.
- **Render & Store**: Generated plot code is executed in a safe sandbox; resulting visual artifacts and JSON metadata are stored (Visual charts → JSON).
- **Deliver**: View BI Report / View Visual Charts / View EDA Report are displayed to the user.
- **Feedback & History**: User submits feedback (Give Feedback) and the chat history is recorded.

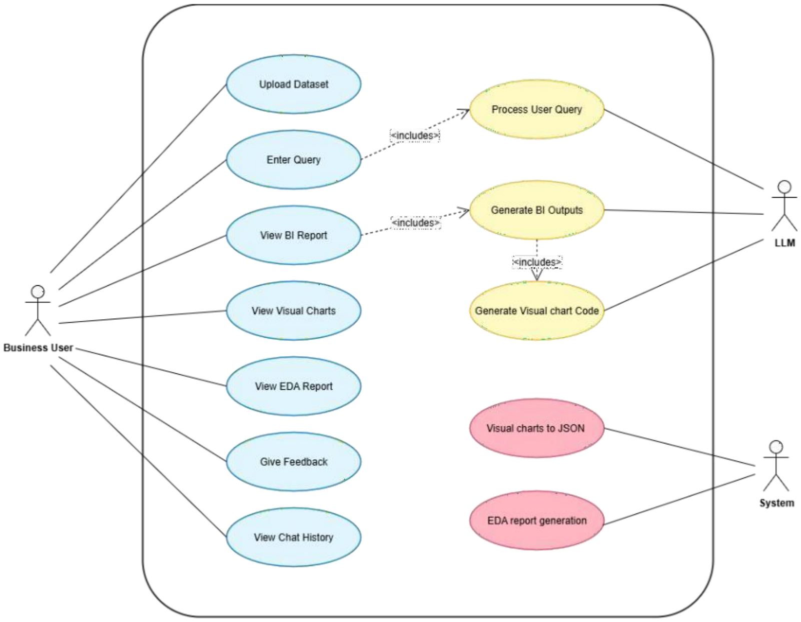


Fig 4.3.1.1: Use Case Diagram

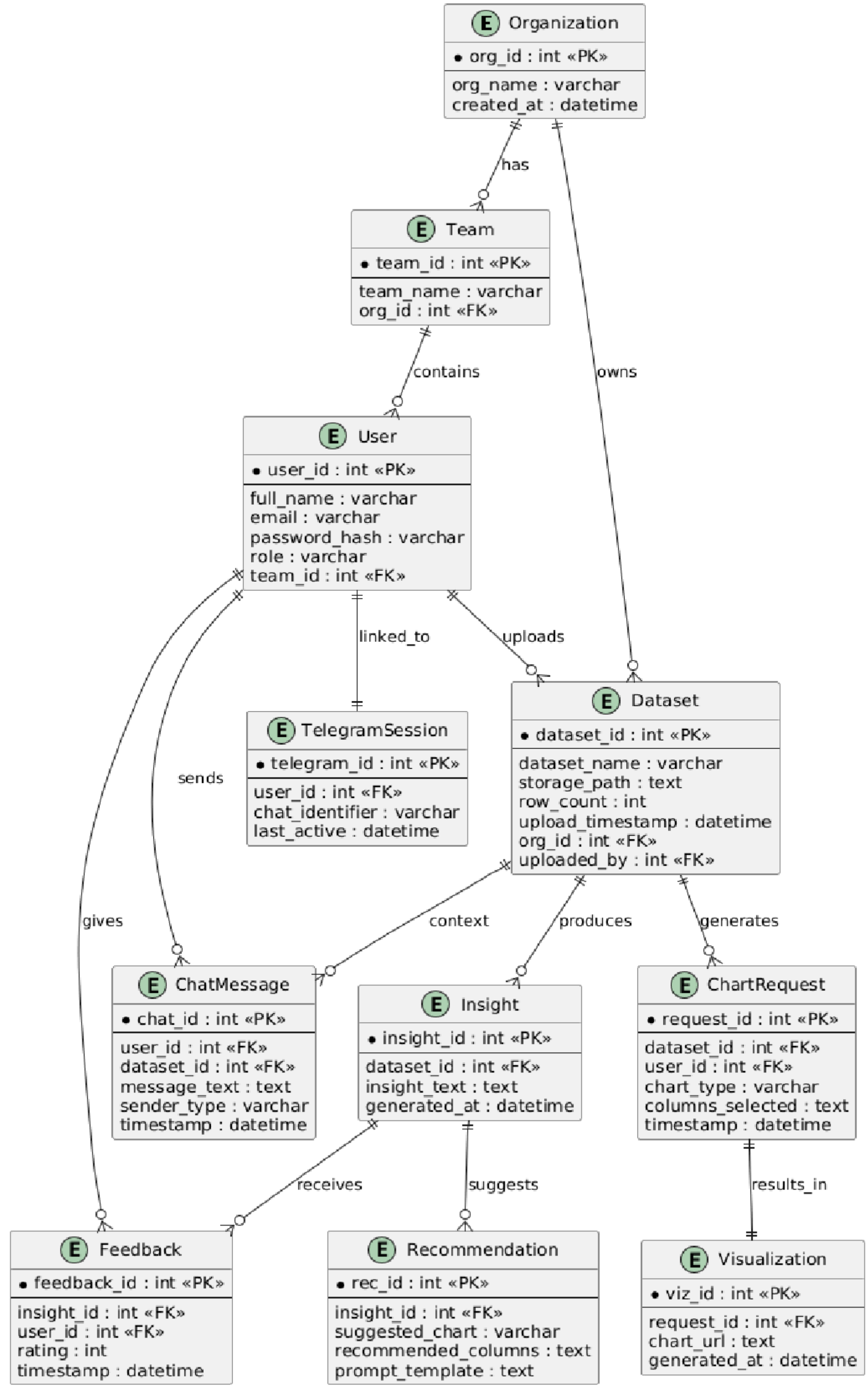


Fig 4.3.1.2 ER Diagram

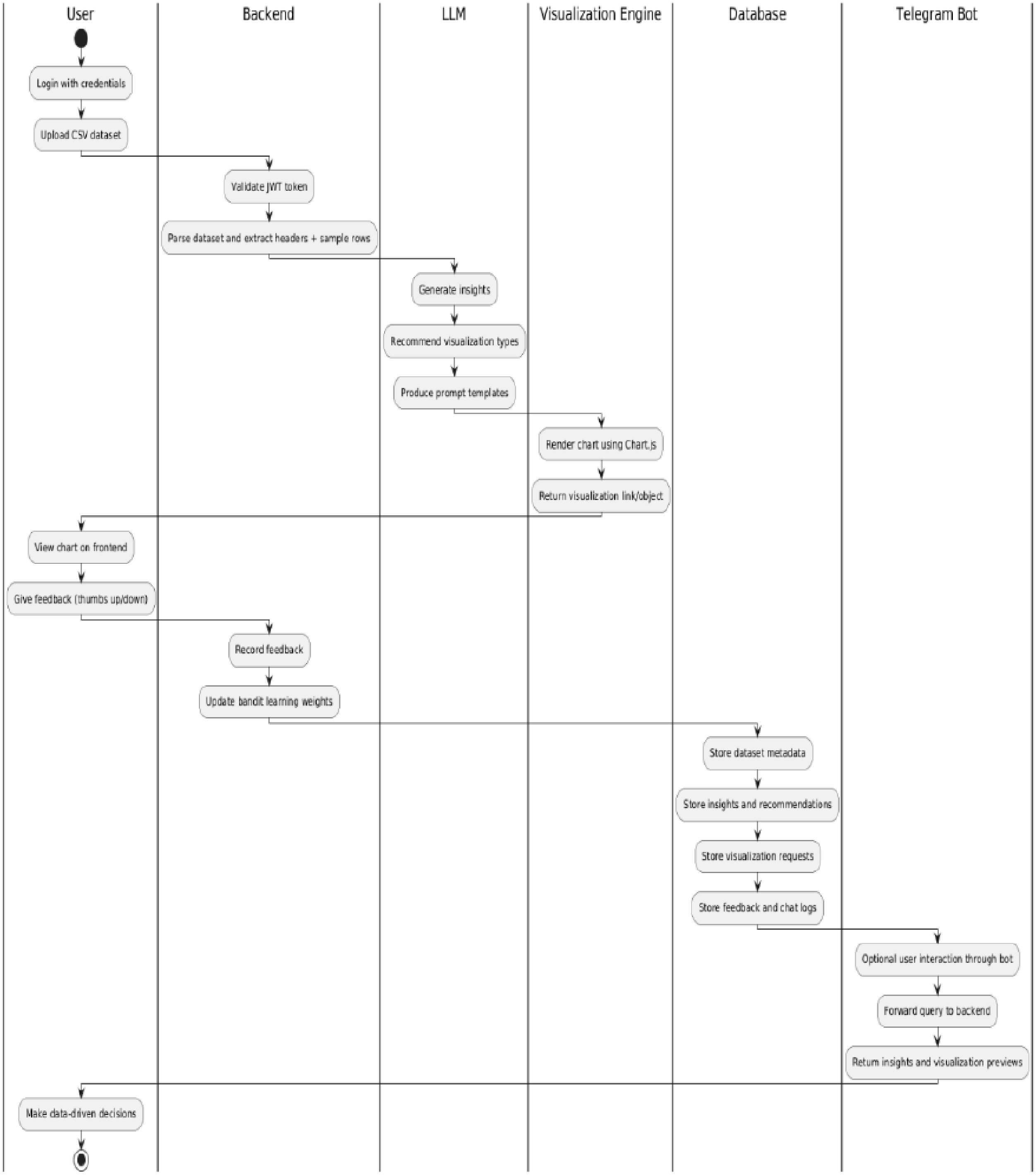
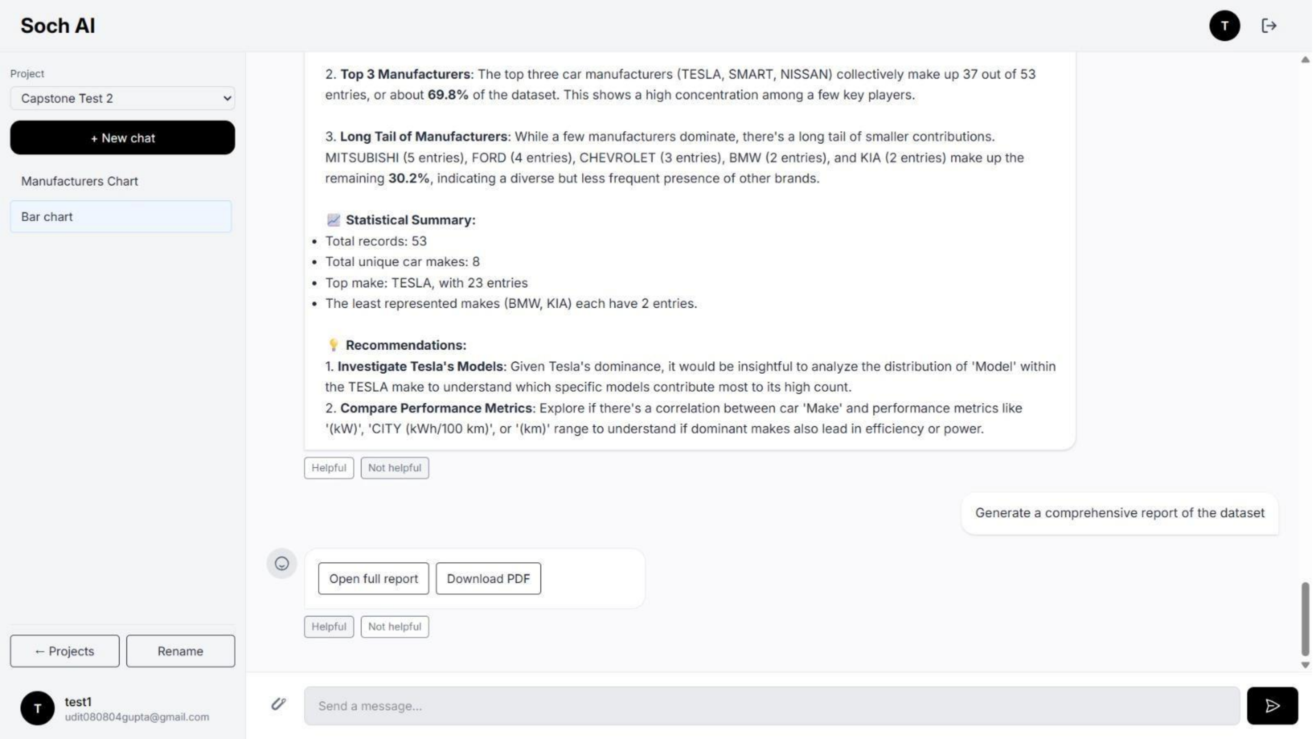

User
Backend
LLM
Visualization Engine
Database
Telegram Bot
Login with credentials
Upload CSV dataset
Validate JWT token
Parse dataset and extract headers + sample rows
Generate insights
Recommend visualization types
Produce prompt templates
Render chart using Chart.js
Return visualization link/object
View chart on frontend
Give feedback (thumbs up/down)
Record feedback
Update bandit learning weights
Store dataset metadata
Store insights and recommendations
Store visualization requests
Store feedback and chat logs
Optional user interaction through bot
Forward query to backend
Return insights and visualization previews
Make data-driven decisions

Fig 4.3.1.3 Swimlane Diagram

4.4 Snapshots of Working Prototype



2. **Top 3 Manufacturers**: The top three car manufacturers (TESLA, SMART, NISSAN) collectively make up 37 out of 53 entries, or about **69.8%** of the dataset. This shows a high concentration among a few key players.

3. **Long Tail of Manufacturers**: While a few manufacturers dominate, there's a long tail of smaller contributions. MITSUBISHI (5 entries), FORD (4 entries), CHEVROLET (3 entries), BMW (2 entries), and KIA (2 entries) make up the remaining **30.2%**, indicating a diverse but less frequent presence of other brands.

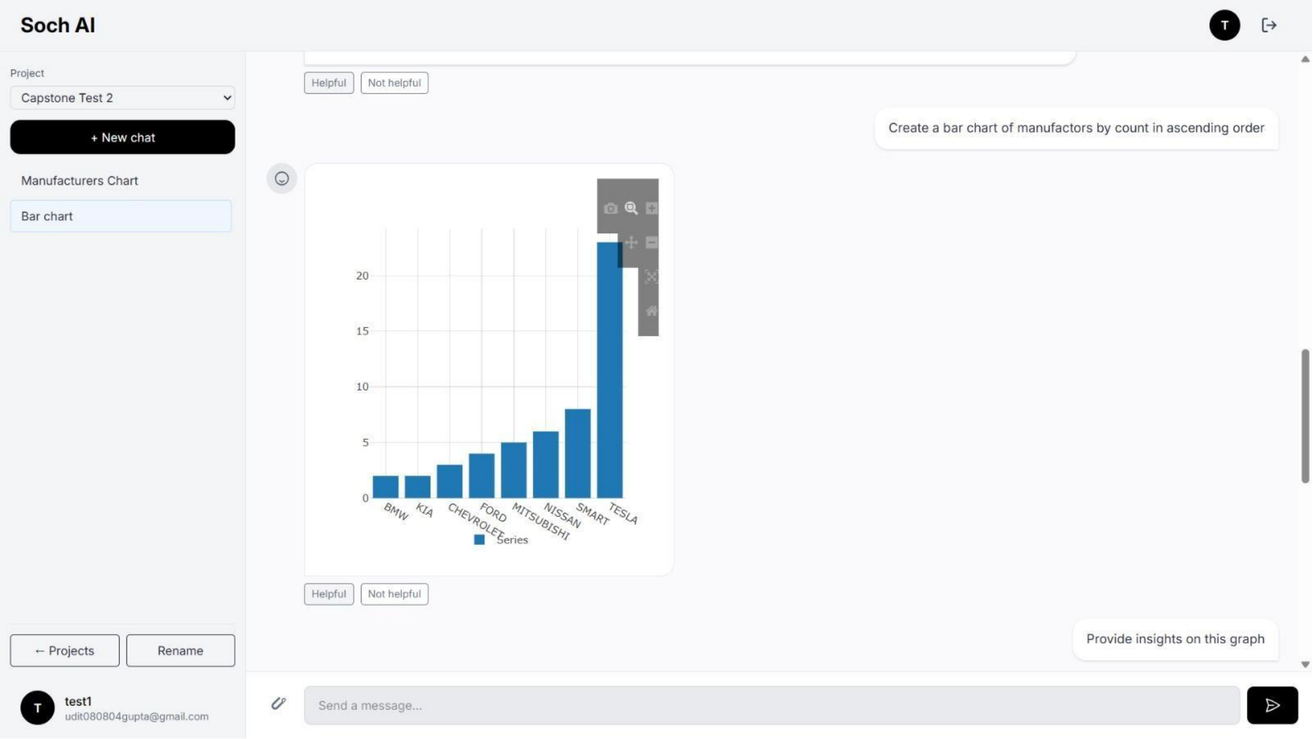
📈 **Statistical Summary:**

- Total records: 53
- Total unique car makes: 8
- Top make: TESLA, with 23 entries
- The least represented makes (BMW, KIA) each have 2 entries.

💡 **Recommendations:**

1. **Investigate Tesla's Models**: Given Tesla's dominance, it would be insightful to analyze the distribution of 'Model' within the TESLA make to understand which specific models contribute most to its high count.
2. **Compare Performance Metrics**: Explore if there's a correlation between car 'Make' and performance metrics like '(kW)', 'CITY (kWh/100 km)', or '(km)' range to understand if dominant makes also lead in efficiency or power.

Generate a comprehensive report of the dataset



Create a bar chart of manufactors by count in ascending order

Provide insights on this graph

IMPLEMENTATION AND EXPERIMENTAL RESULTS

5.1 Experimental Setup

The experimental setup for evaluating the proposed system consisted of a cloud-based deployment environment integrated with a web interface and conversational bot to simulate realistic user workflows. The frontend was deployed on Render to ensure scalability and remote access, while backend services and the Telegram bot were hosted on Railway for continuous availability. MongoDB Atlas served as the central database for managing datasets, chat logs, and organization-based access control. The system was tested using structured CSV datasets of varying sizes to examine ingestion efficiency, visualization performance, and insight generation accuracy through LLM integration. Testing was conducted through modern web browsers and Telegram clients to emulate typical usage conditions and validate usability and responsiveness.

Core Components of the Experimental Setup

- **Frontend Deployment (Render):** The React-based interface was hosted on Render to provide a cloud-delivered, scalable environment for chart rendering and user interaction. This ensured low-latency visualization and ease of access across devices.
- **Backend Services (Express + Fast API):** The core processing tasks, including dataset handling, authentication, and routing, were managed by Express and Fast API services. This enabled modular interactions between dataset ingestion, LLM insight generation, and visualization components.
- **MongoDB Atlas Database:** MongoDB Atlas stored uploaded datasets, chat histories, and organizational metadata. Its flexible schema supported varying dataset structures and facilitated scalable data management.
- **Cloud-based LLM Integration:** The system accessed a cloud-hosted LLM for contextual insight generation, recommendation creation, and prompt interpretation. This removed local hardware dependencies and ensured rapid inference under stable network conditions.
- **Telegram Bot (Railway Deployment):** A Telegram bot was hosted on Railway to extend accessibility to conversational interaction.

- **Client-Side Visualization (Plotly):** Plotly enabled rendering of interactive visualizations directly within the browser. This reduced server-side computation and supported dynamic exploration of numerical and categorical fields.

- **Testing Datasets:** Multiple datasets, including the Thapar admissions dataset, were used to evaluate system performance across varying sizes and feature distributions. This ensured realistic validation of scalability and insight relevance.

- **User Feedback Mechanism (Bandit Learning):** Insight quality was iteratively refined using thumbs-up/down feedback through a lightweight bandit learning approach. This provided a continuous evaluation loop during experiments.

5.2 Experimental Analysis

The experimental analysis was conducted to evaluate the system's ability to ingest datasets of varying complexity, generate meaningful insights using the cloud-based LLM, and render appropriate visualizations through an interactive web interface. The evaluation emphasized four key aspects: dataset handling capacity, insight relevance, recommendation accuracy, and usability across diverse data types. The deployed platform was tested under realistic user conditions to observe performance with moderate-scale datasets, while larger datasets were utilized in controlled development environments to assess scalability and responsiveness.

To ensure generalizability, experiments were performed not only on the large-scale Thapar University admissions dataset-comprising over 60,000 rows and 180+ columns-but also on smaller and heterogeneous datasets such as automobile sales data and the EU Superstore dataset. These datasets varied significantly in feature types, including categorical, numerical, temporal, and textual attributes. The system successfully processed these datasets and produced interpretable visualizations and insights, demonstrating that the underlying workflow and recommendation mechanism remain effective regardless of dataset origin or domain. This analysis confirms that the platform is adaptable, capable of supporting both domain-specific institutional datasets and open-source datasets across sectors.

5.2.1 Data (Data Sources / Data Cleaning / Data Pruning / Feature Extraction Workflow)

Primary Dataset: Thapar University Admissions Data

This dataset, containing more than 60,000 entries and 180+ features, was used to evaluate scalability and real-world applicability. It included academic, demographic, and categorical attributes that enabled comprehensive testing of visualization and insight generation features. Private identifiers were anonymized to maintain confidentiality, and sensitive fields were removed prior to analysis.

Additional Datasets for Generalization

To test adaptability, the system was also validated on smaller and more varied datasets such as car sales data and the EU Suspecttore dataset. These datasets contained mixed numerical and categorical fields, allowing assessment of visualization recommendations across unrelated domains. The system's successful performance on these datasets demonstrated cross-domain suitability.

Data Cleaning and Anonymization

Standard preprocessing routines were applied to ensure data consistency, including the removal of outliers using common statistical thresholds and noise reduction techniques. Confidential and personally identifiable information was removed or masked. Temporal data, such as daily figures, was aggregated to weekly intervals to facilitate clearer trend analysis and smoother visual interpretation.

Data Pruning and Feature Reduction

To optimize dataset size and remove redundancy, less significant or irrelevant features were pruned. Multi-category attributes were consolidated into simplified classifications to ensure meaningful visualizations and manageable insight generation. This pruning ensured faster processing during experimentation and enhanced clarity in visualization.

Feature Extraction and Type Identification

The system automatically distinguished between numerical, categorical, and temporal columns for appropriate visualization mapping.

Feature extraction was achieved through parsing of dataset headers and initial rows, which enabled the recommendation engine to propose suitable chart types such as bar charts, line graphs, scatter plots, and pie charts. This workflow minimized manual preprocessing and validated the automation capabilities of the system.

Dataset Versatility and Domain Independence

The successful experimentation across educational, retail, and European statistical datasetsindicates that the platform is not constrained to a particular domain. The workflow supports any structured CSV dataset, reaffirming the general-purpose applicability of the system's architecture and recommendation logic.

5.2.2 Performance Parameters (Accuracy Type Measures / QoS Parameters)

Performance evaluation for this system focuses on two major categories: (i) accuracy of insight generation and visualization recommendation, and (ii) Quality of Service (QoS) metrics related to responsiveness, scalability, and usability. Since traditional machine learning metrics-such as precision or recall-are not directly applicable to this conversational insight generation paradigm, alternative metrics better suited for evaluative interpretation are recommended below.

Accuracy-Oriented Evaluation Metrics

- **Insight Relevance Score (IRS):** This metric can be defined as the proportion of generated insights rated as "useful" by users through feedback mechanisms such as thumbs-up or approval signals. IRS = (Number of approved insights / Total insights generated). This provides a quantifiable measure of insight quality based on user validation.
- **Visualization Recommendation Precision (VRP):** VRP measures how often recommended visualizations align with domain-appropriate chart types for given column types. For example, numerical vs numerical → scatter plot, categorical vs numerical → bar chart. VRP = (Correct recommendations / Total recommendations). This metric evaluates whether the recommendation engine conforms to widely accepted visualization rules.

These accuracy measures reflect the system's focus: relevance and validity of insights and recommendations, rather than raw classification accuracy.

Quality of Service (QoS) Performance Parameters

- **Latency:**

  The system demonstrates low response time for client-side visualization due to browser-based rendering through Plotly. Insight generation latency primarily depends on cloud-based LLM inference and network conditions but remains acceptable for interactive analytical use.

- **Scalability:**

  The deployed platform efficiently supports moderate datasets (up to ~5000 rows) and has demonstrated scalability to larger datasets (60,000+ rows) in controlled development environments. It maintains functional performance across varying dataset sizes.

- **Dataset Versatility:**

  The system successfully processed datasets from diverse domains, including education (Thapar admissions), retail (car sales), and European statistical data (EU Suspecttore), demonstrating suitability for heterogeneous dataset types and formats.

- **Cross-Platform Accessibility:**

  QoS is improved through dual access modes: a web interface and a Telegram bot. This enables real-time interaction on desktops and mobile devices and supports remote usage scenarios.

- **Insight Strength and Interpretability:**

  Generated business insights maintain numerical grounding through aggregation, comparison, and trend identification. Recommendations reflect recognized visualization practices, aiding interpretability and analytical rigor.

- **Reliability and Availability:**

  Cloud deployment via Render and Railway ensures high availability and removes dependency on local hardware. MongoDB Atlas ensures persistent, fault-tolerant storage of user data and organizational structures.

- **User Feedback Adaptation:**

  Bandit-based feedback enables incremental performance improvement by weighting user-approved insights more heavily, enhancing the system’s responsiveness to real user validation.

In summary, the performance of this system can be evaluated through a mix of human-validated accuracy indicators and QoS metrics tailored to interactive analytical applications.

5.3 Working of the Project

The project functions as an end-to-end data visualization and insight generation system designed to transform raw datasets into meaningful interpretations through automated analysis and interactive visual representation. Users upload CSV datasets through the web interface, after which the backend processes the data, extracts initial structural information, and forwards relevant context to a cloud-based LLM. The LLM generates insight suggestions and visualization recommendations, which are rendered on the frontend using Plotly. MongoDB manages dataset storage and chat history within a secure multi-organization environment authenticated via JWT. A Telegram bot replicates core functionality for conversational access. Through iterative bandit learning, user feedback is incorporated to refine insight relevance. The integrated workflow ensures that non-technical users can derive actionable business intelligence without requiring programming or statistical expertise.

5.3.1 Procedural Workflow

The following procedural workflow outlines the sequential phases involved in conceptualizing, designing, implementing, deploying, and validating the system:

1. **Literature Review and Gap Identification:** Existing visualization and BI tools were analyzed to identify limitations such as steep learning curves, manual configuration requirements, limited automated insight generation, and lack of collaborative organizational structures.

2. **Problem Definition and Requirement Specification:** A clear problem statement was formulated to address accessibility challenges in data interpretation. Functional and non-functional requirements were documented within the SRS to define scope, constraints, and system expectations.

3. **Design of System Architecture and UI Prototyping:** Initial prototypes were developed using Streamlit to validate feasibility and interaction design. Subsequently, a modular architecture integrating Express, Fast API, React, and MongoDB was conceptualized for scalability and reusable components.

4. **Dataset Ingestion and Preprocessing Module Development:** Backend services were implemented to support CSV upload, structure extraction, sample data preview, and basic cleaning routines. Dataset headers and initial rows were utilized to contextualize insight generation.

5. **LLM Integration for Insight Generation:** A cloud-based LLM was integrated to interpret dataset context and generate insights, visualization suggestions, and prompt templates. The recommendation engine was designed to complement LLM outputs with structured chart guidance.

6. **Visualization Component Implementation:** Plotly was adopted to render bar, line, scatter, and pie charts dynamically in the browser. The frontend was built in React to support interactivity, responsiveness, and intuitive user experience.

7. **Multi-Organization and Authentication Development:** MongoDB schema design supported tenant-based dataset and chat segregation. JWT authentication ensured secure access and role-based usage, simulating real enterprise environments.

8. **Telegram Bot Development and Integration:** A bot was deployed using Railway to extend platform usage beyond browsers. It facilitated conversational insight querying and dataset interaction through messaging.

9. **Bandit Learning Feedback Mechanism Integration:** A thumbs-up/down system allowed user feedback to be captured and leveraged for iterative improvement of LLM recommendations and insight relevance.

10. **Deployment and Cloud Configuration:** The frontend was deployed on Render and backend bot services on Railway. Cloud deployment enabled accessibility, scalability, and minimized dependency on local hardware.

11. **Testing and Experimental Evaluation:** The system was tested using multiple datasets-including Thapar admissions, car sales, and EU datasets-to validate versatility, scalability, and accuracy metrics such as Insight Relevance Score and Visualization Recommendation Precision.

12. **Performance Analysis and Documentation:** Experimental results, performance parameters, system constraints, and usability findings were documented. Limitations and future enhancements such as RLHF and local model support were identified.

5.3.2 Algorithmic Approaches Used

1. Visualization Recommendation Algorithm (Prompt-Driven Semantic Reasoning)

This component is responsible for analyzing dataset structure-columns, inferred datatypes, and sample values-to propose suitable visualization types. Rather than pre-defined rules alone, the system delegates semantic pattern recognition to the LLM using prompt-engineering. The algorithm dynamically identifies meaningful categories (such as demographic, performance, or temporal clusters) and pairs them with appropriate chart types. We have shown the components of the system in the Fig 5.3.3.1 and the overall deployment architecture in the Fig 5.3.3.2.

Rationale:
This approach leverages LLM contextual reasoning to adapt recommendations across domains without manual rule coding. It ensures chart diversity and dataset-aware visualization mapping.

Python-style Pseudocode Snippet:

```
prompt = f"""
   Act as a Lead Data Storyteller. I am providing a dataset schema.
   **Dataset Info:**
   - Columns: {columns}
   - Data Types: {dtypes}
   - Sample Data (Top 5 rows):
   {df_head.to_markdown()}

   **Your Goal:**
   Recommend exactly {num_charts} visualizations, but you MUST structure them into logical categories based on the data.
   **CRITICAL INSTRUCTIONS:**
   1. Categorize visualization recommendations.
   2. Ensure chart diversity based on column types.
```

~~~
    3. Use EXACT column names in tool commands.

    **Output Format:**

    ## 1. [Category Name]

    ### [Visualization Title] ([Chart Type])

    **Business Insight:** [...]

    **Tool Command:**
    ```text

    Create a […]

    ```

"""

llm_response = llm.generate(prompt)

parse_and_store_recommendations(llm_response)
~~~

2. Insight Generation Algorithm (Context-Grounded Natural Language Reasoning)

This algorithm generates business insights by grounding LLM predictions in actual dataset values. It extracts data semantics using sampling (e.g., the first five rows), datatype inference, and correlation heuristics. The LLM uses this structured context to produce domain-aware interpretations rather than generic statements.

Rationale:

Grounding reduces hallucination and ensures insights align with observed numerical and categorical distributions.

Python-style Pseudocode Snippet:

```
context = {

"columns": list(df.columns),

"types": df.dtypes.to_dict(),
```

```
 "sample": df.head().to_dict()
}
prompt = create_insight_prompt(context)
insights = llm.generate(prompt)
store_insights(insights)
```

3. Bandit Learning Algorithm for Adaptive Refinement

To incorporate user feedback, the system uses a lightweight multi-armed bandit approach. Each insight or visualization suggestion is treated as an "arm" with an associated reward score based on user approval (thumbs-up) or disapproval (thumbs-down). Scores influence future ranking and selection.

Rationale:

This enables incremental improvement without retraining the underlying LLM, making the system adaptive yet computationally economical.

Python-style Pseudocode Snippet:

```
def update_score(arm_id, feedback):
    if feedback == "up":
        scores[arm_id] += 1
    else:
        scores[arm_id] -= 1
def select_next():
    return max(scores, key=scores.get)
```

4. Conversational Query Handling Algorithm

This approach enables natural language interaction through web chat or Telegram. Queries are interpreted using an LLM chain:

classify intent (visualization, statistic, or clarification) ->

extract target columns ->

route request to visualization or insight modules

Rationale:

Intent classification abstracts away SQL-like querying, enabling users to interact fluidly.

Python-style Pseudocode Snippet:

```
query = user_input
intent = llm.classify_intent(query)  # e.g., "visualize", "trend", "correlation" entities =
extract_columns(query)
if intent == "visualize":
    chart = generate_chart(entities)
elif intent == "trend":
    insight = insight_engine(entities)
else:
    response = llm.answer(query)
```

5.3.3 Project Deployment

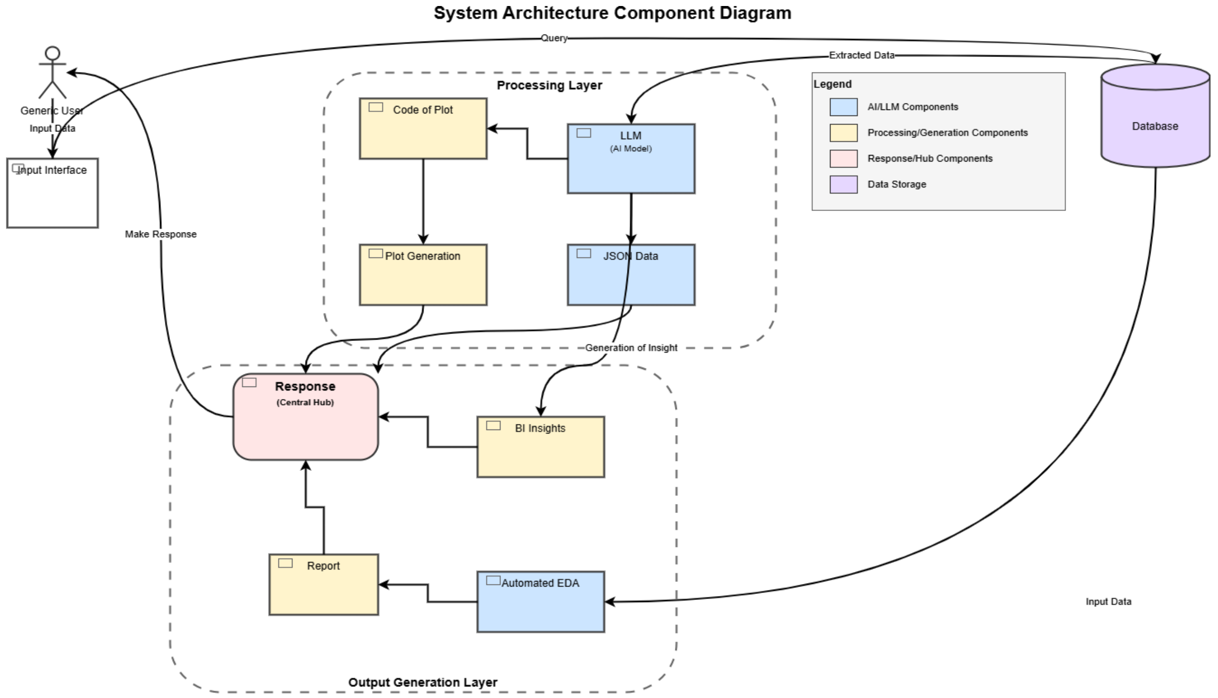


Fig 5.3.3.1 Component Diagram

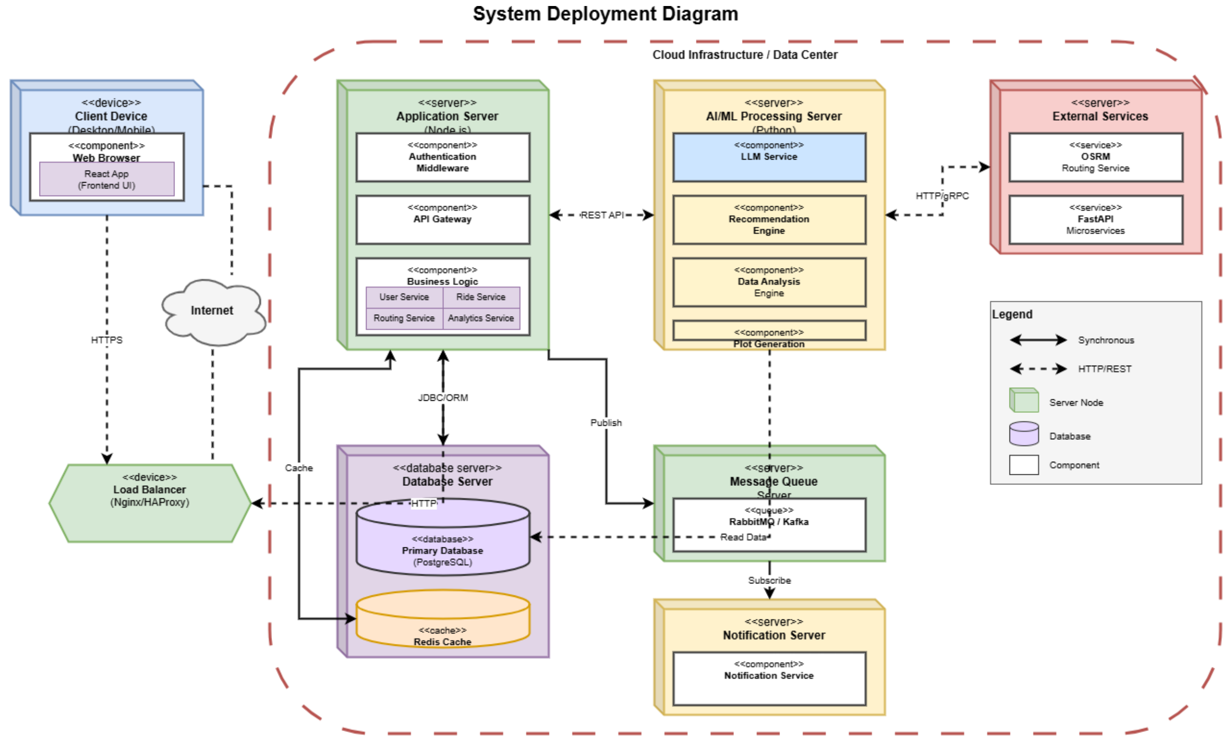
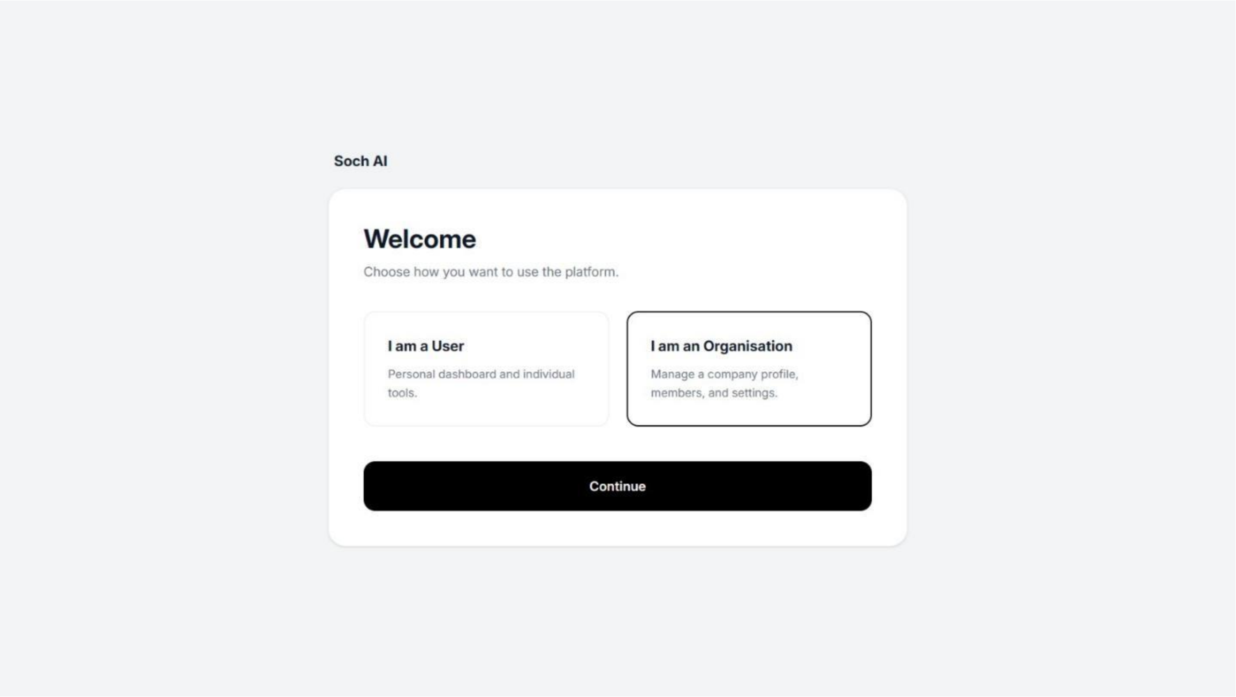


Fig 5.3.3.2 Deployment Diagram

5.3.4 System Screenshots



Soch AI

Welcome

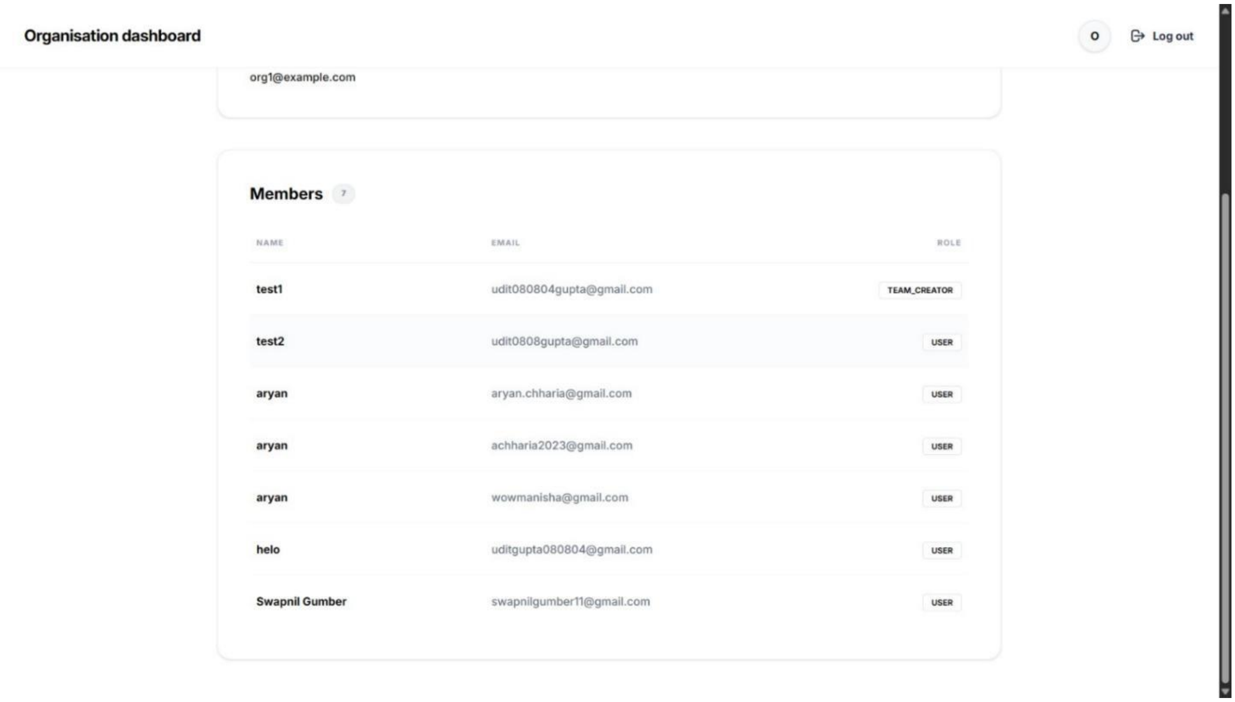
Choose how you want to use the platform.

I am a User

Personal dashboard and individual tools.

I am an Organisation

Manage a company profile, members, and settings.

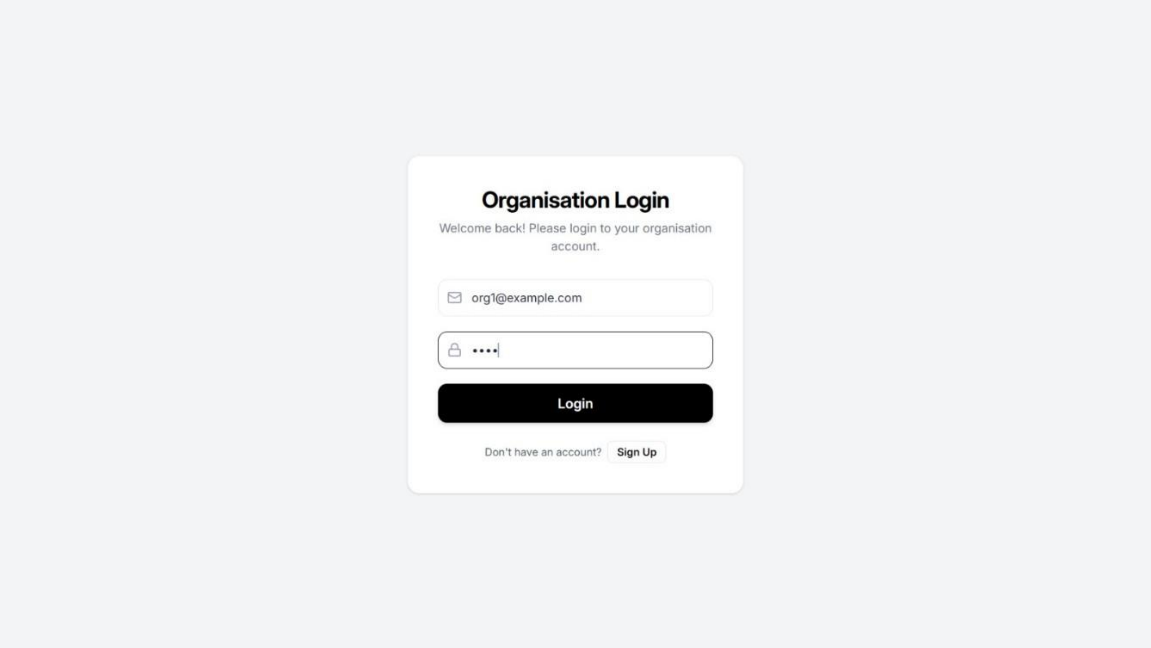


Organisation dashboard

org1@example.com

Members 7

NAME	EMAIL	ROLE
test1	udit080804gupta@gmail.com	TEAM_CREATOR
test2	udit0808gupta@gmail.com	USER
aryan	aryan.chharia@gmail.com	USER
aryan	achharia2023@gmail.com	USER
aryan	wowmanisha@gmail.com	USER
helo	uditgupta080804@gmail.com	USER
Swapnil Gumber	swapnilgumber11@gmail.com	USER

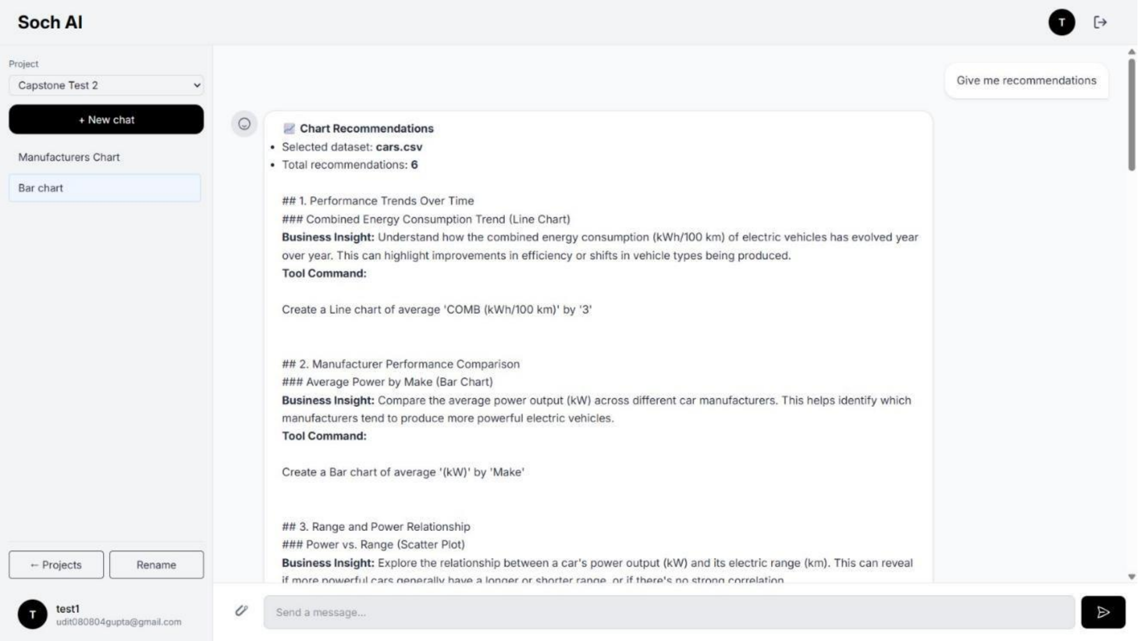


Organisation Login

Welcome back! Please login to your organisation account.

org1@example.com

••••



Give me recommendations

Chart Recommendations

- Selected dataset: **cars.csv**
- Total recommendations: **6**

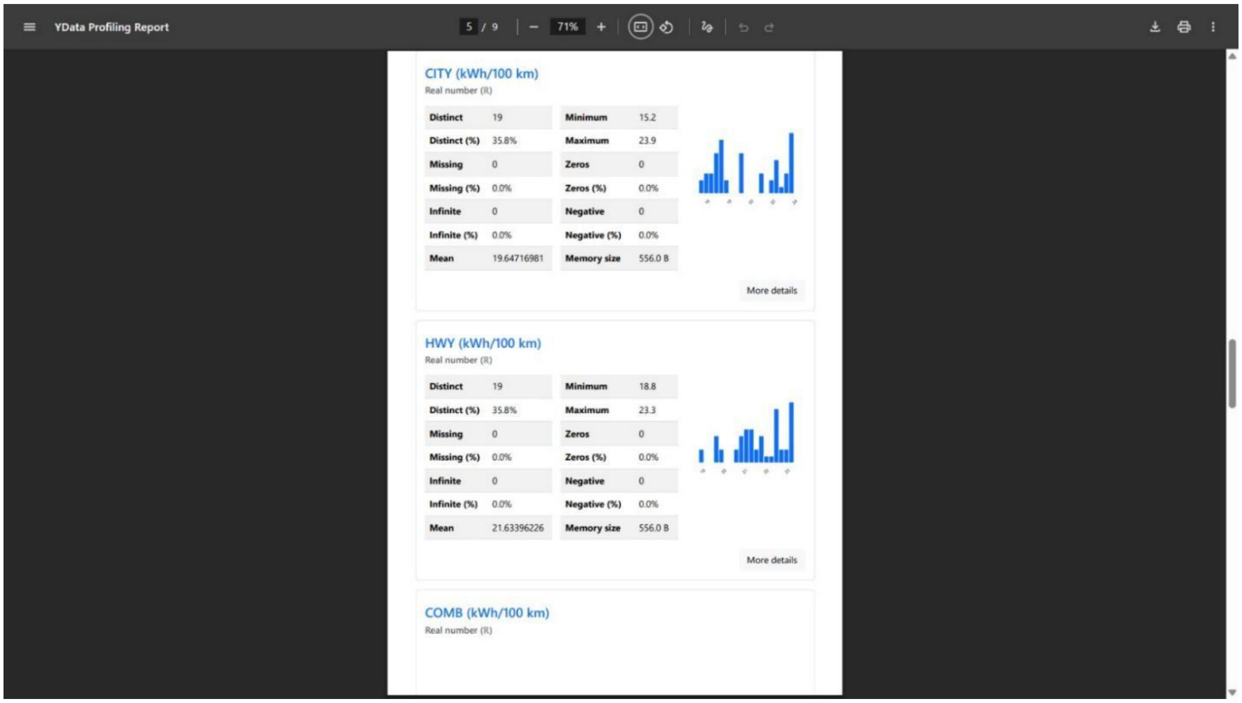
1. Performance Trends Over Time
Combined Energy Consumption Trend (Line Chart)
Business Insight: Understand how the combined energy consumption (kWh/100 km) of electric vehicles has evolved year over year. This can highlight improvements in efficiency or shifts in vehicle types being produced.
Tool Command:

Create a Line chart of average 'COMB (kWh/100 km)' by '3'

2. Manufacturer Performance Comparison
Average Power by Make (Bar Chart)
Business Insight: Compare the average power output (kW) across different car manufacturers. This helps identify which manufacturers tend to produce more powerful electric vehicles.
Tool Command:

Create a Bar chart of average '(kW)' by 'Make'

3. Range and Power Relationship
Power vs. Range (Scatter Plot)
Business Insight: Explore the relationship between a car's power output (kW) and its electric range (km). This can reveal if more powerful cars generally have a longer or shorter range, or if there's no strong correlation



CITY (kWh/100 km)

Real number (ℝ)

Distinct	19	Minimum	15.2
Distinct (%)	35.8%	Maximum	23.9
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	19.64716981	Memory size	556.0 B

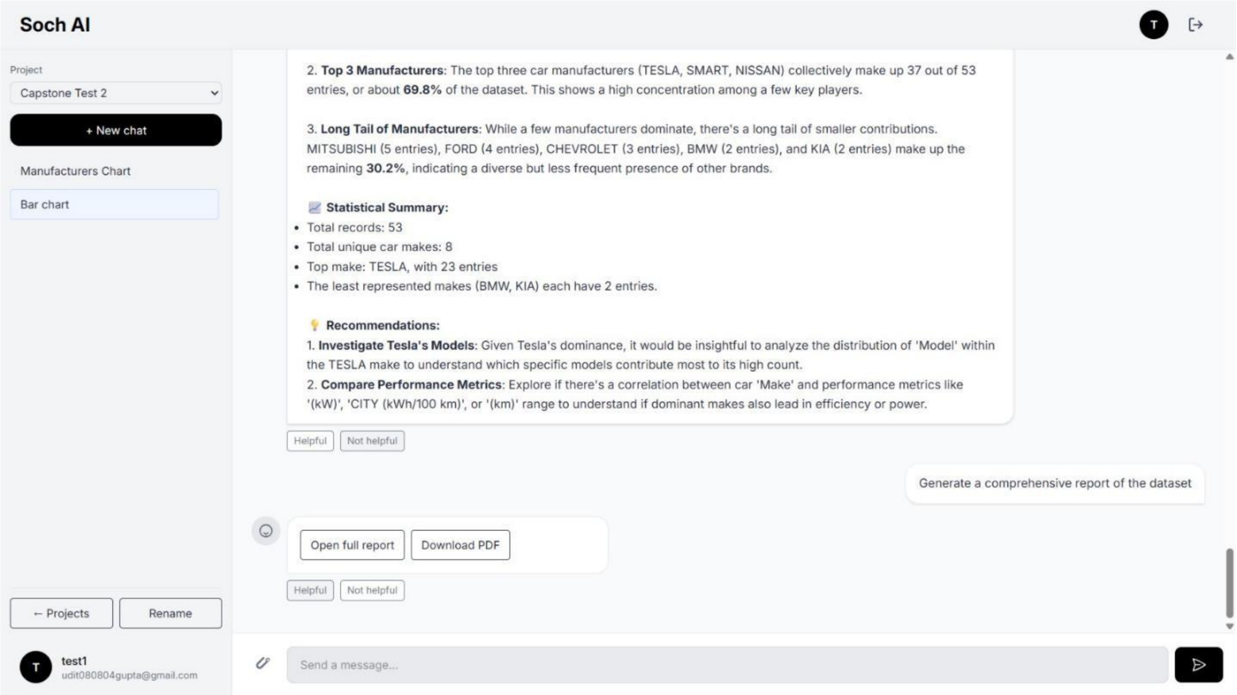
HWY (kWh/100 km)

Real number (ℝ)

Distinct	19	Minimum	18.8
Distinct (%)	35.8%	Maximum	23.3
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	21.63396226	Memory size	556.0 B

COMB (kWh/100 km)

Real number (ℝ)



2. **Top 3 Manufacturers**: The top three car manufacturers (TESLA, SMART, NISSAN) collectively make up 37 out of 53 entries, or about **69.8%** of the dataset. This shows a high concentration among a few key players.

3. **Long Tail of Manufacturers**: While a few manufacturers dominate, there's a long tail of smaller contributions. MITSUBISHI (5 entries), FORD (4 entries), CHEVROLET (3 entries), BMW (2 entries), and KIA (2 entries) make up the remaining **30.2%**, indicating a diverse but less frequent presence of other brands.

📈 **Statistical Summary:**

- Total records: 53
- Total unique car makes: 8
- Top make: TESLA, with 23 entries
- The least represented makes (BMW, KIA) each have 2 entries.

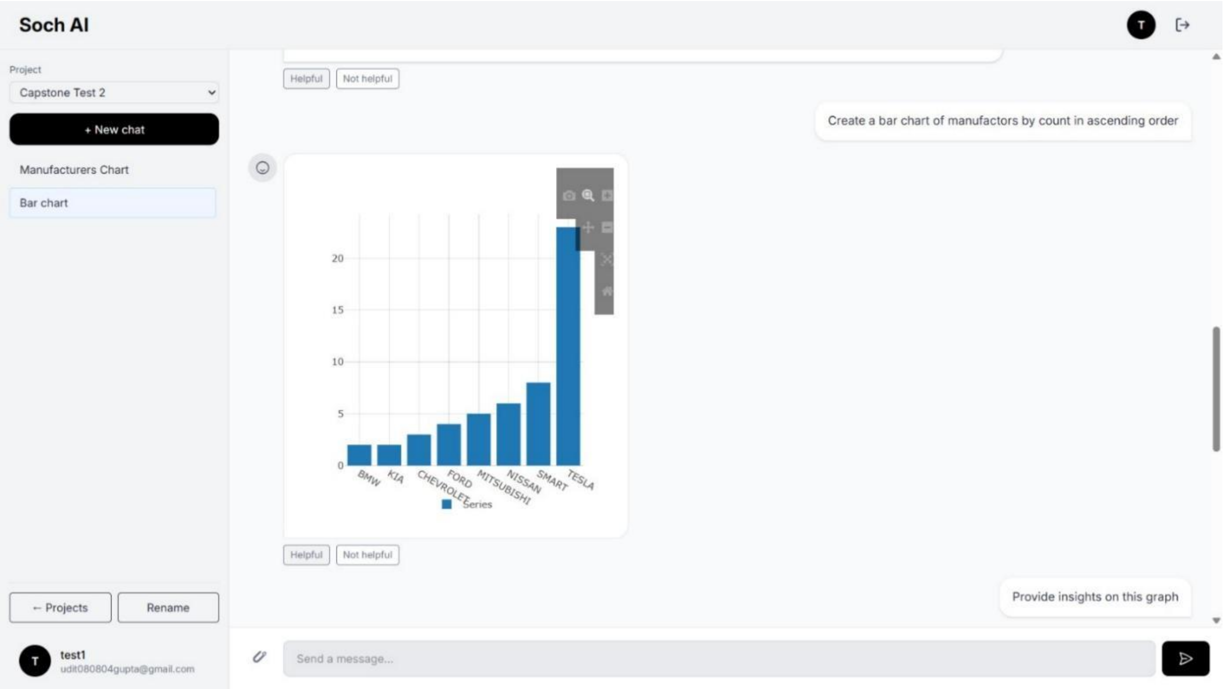
💡 **Recommendations:**

1. **Investigate Tesla's Models**: Given Tesla's dominance, it would be insightful to analyze the distribution of 'Model' within the TESLA make to understand which specific models contribute most to its high count.
2. **Compare Performance Metrics**: Explore if there's a correlation between car 'Make' and performance metrics like '(kW)', 'CITY (kWh/100 km)', or '(km)' range to understand if dominant makes also lead in efficiency or power.

Generate a comprehensive report of the dataset

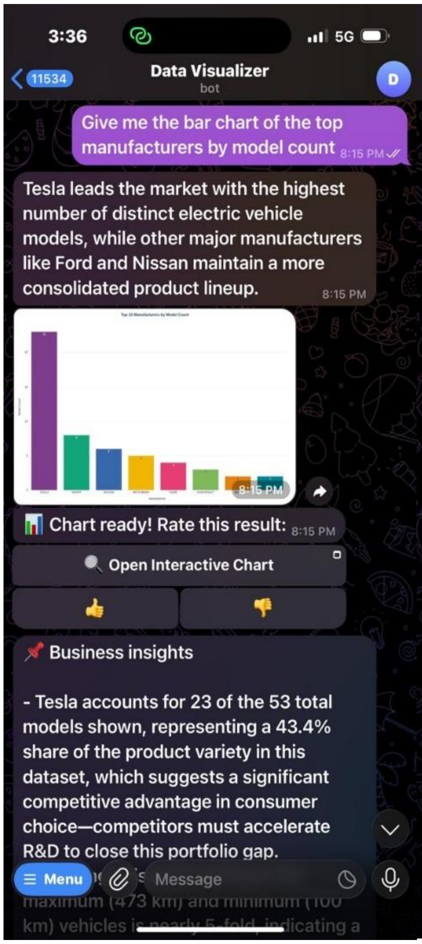
test1

udit080804gupta@gmail.com



Create a bar chart of manufactors by count in ascending order

Provide insights on this graph



Give me the bar chart of the top manufacturers by model count 8:15 PM

Tesla leads the market with the highest number of distinct electric vehicle models, while other major manufacturers like Ford and Nissan maintain a more consolidated product lineup. 8:15 PM

8:15 PM

📊 Chart ready! Rate this result: 8:15 PM

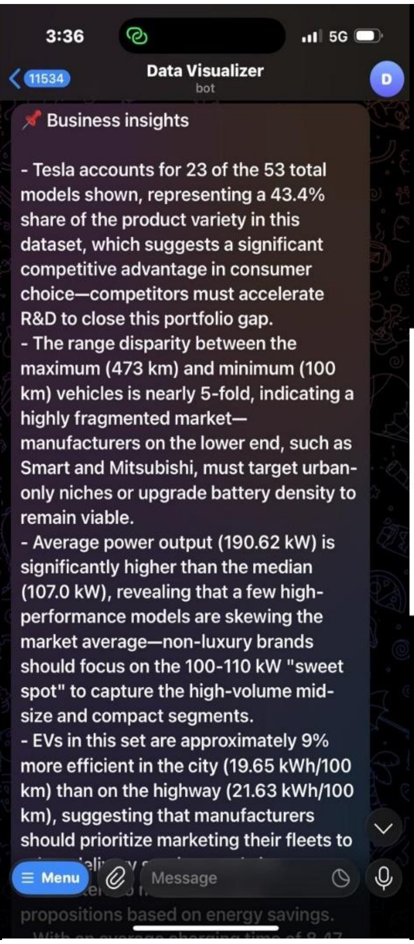
🔍 Open Interactive Chart

👍 👎

📌 Business insights

- Tesla accounts for 23 of the 53 total models shown, representing a 43.4% share of the product variety in this dataset, which suggests a significant competitive advantage in consumer choice—competitors must accelerate R&D to close this portfolio gap.

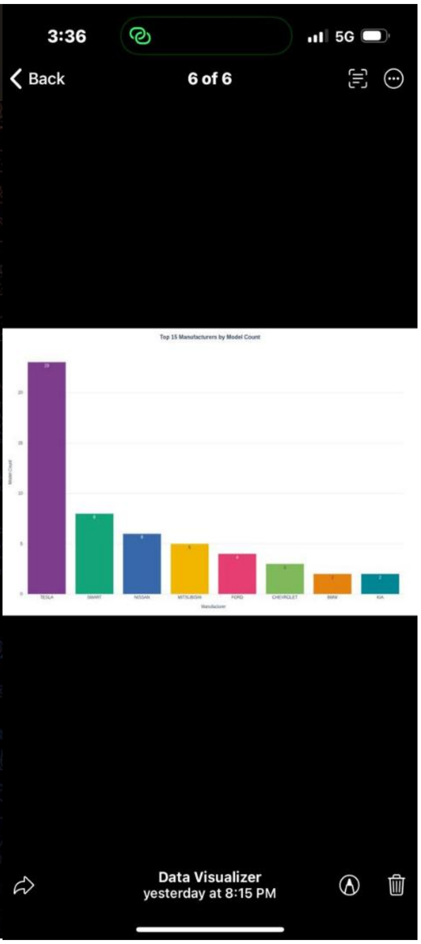
maximum (473 km) and minimum (100 km) vehicles is nearly 5-fold, indicating a



📌 Business insights

- Tesla accounts for 23 of the 53 total models shown, representing a 43.4% share of the product variety in this dataset, which suggests a significant competitive advantage in consumer choice—competitors must accelerate R&D to close this portfolio gap.
- The range disparity between the maximum (473 km) and minimum (100 km) vehicles is nearly 5-fold, indicating a highly fragmented market—manufacturers on the lower end, such as Smart and Mitsubishi, must target urban-only niches or upgrade battery density to remain viable.
- Average power output (190.62 kW) is significantly higher than the median (107.0 kW), revealing that a few high-performance models are skewing the market average—non-luxury brands should focus on the 100-110 kW "sweet spot" to capture the high-volume mid-size and compact segments.
- EVs in this set are approximately 9% more efficient in the city (19.65 kWh/100 km) than on the highway (21.63 kWh/100 km), suggesting that manufacturers should prioritize marketing their fleets to

propositions based on energy savings.



5.4 Testing Process

The testing process adopted for this project followed a systematic methodology to validate the functional correctness, usability, performance, and reliability of the system across diverse datasets and interaction modes. Testing was conducted iteratively to ensure that each component-from dataset ingestion to visualization rendering and insight generation-behaved in accordance with the expected outcomes defined in the Software Requirements Specification. Both manual and automated testing techniques were employed. Manual testing was utilized for validating conversational interactions and insight relevance, while automated checks ensured dataset upload consistency, authorization enforcement, and UI responsiveness.

The process emphasized black-box testing for evaluating system behavior without internal code inspection, and integration testing to verify communication between the frontend, backend, database, and cloud-based LLM services. Unit-level evaluations targeted specific modules such as authentication, dataset parsing, and recommendation logic. Test cases were executed on varying dataset sizes, including large institutional data and smaller heterogeneous datasets, to ensure scalability and versatility. Overall, the testing strategy ensured functional compliance, minimized defects, and verified that the system satisfies the objectives of accessible and automated business insight generation.

5.4.1 Test Plan

The test plan outlined below describes the approach, scope, objectives, and environment used for validating the system:

- **Testing Objectives:**
    - verify correct dataset upload and preview functionality;
    - validate accurate chart rendering and recommendation generation;
    - ensure appropriate insight interpretation by the LLM;
    - confirm multi-organization authentication and secure access control;
    - test responsiveness across web and Telegram interfaces;
    - evaluate scalability with datasets of varying sizes.

- **Test Scope:**
  - frontend UI components (dataset upload, visualization, navigation);
  - backend services (dataset parsing, authentication, recommendation API);
  - MongoDB interactions (dataset and chat storage, retrieval);
  - Telegram bot conversational workflows;
  - insight generation and feedback refinement.
- **Testing Environment:**
  - frontend deployed on Render;
  - backend services on Railway;
  - MongoDB Atlas database;
  - cloud-based LLM-enabled insight generation;
  - client-side testing through modern web browsers and Telegram client.
- **Test Data:**
  - institutional dataset: Thapar admissions (>60k rows);
  - moderate datasets (~5k rows) for deployed version;
  - heterogeneous open datasets: car sales, EU Suspecttore.
- **Testing Techniques:**
  - black-box testing for high-level behavior;
  - integration testing for module interactions;
  - system testing for end-to-end workflows;
  - performance testing for dataset limits and response time;
  - usability testing for intuitive interaction.

This plan ensured that the system met functional and non-functional expectations and supported robust evaluation across representative scenarios.

5.4.2 Features to be Tested

The table below outlines core features and associated test conditions:

Table 5.4.2.1 Features to be Tested

Feature	Description	Test Case / Input	Expected Output	Success Criteria
CSV Upload	Uploading structured datasets	Upload valid CSV (5000 rows)	Dataset preview shown	File parsed; no errors
Large Dataset Handling	Internal scaling capability	Upload >60,000 row dataset	Successful processing in env	System handles high volume
Dataset Preview	Display of header + first rows	Upload CSV with mixed types	Table preview visible	Accurate column mapping
Data Cleaning	Outlier & confidential removal	Dataset with private fields	Sensitive fields masked	No identifiable information
Column Type Detection	Identify numerical/categorical	Mixed column dataset	Correct type classification	Chart suggestions appropriate
Chart Rendering	Visualization via Plotly	Generate bar chart	Chart displayed correctly	Axes and values accurate
Chart Recommendation	Suggested visualization types	Select columns	Scatter/Bar/Pie assigned	Matches datatype logic
Insight Generation	LLM interpretation	Upload dataset	Insight text generated	Insight relevance logical
Recommendation Prompts	Copy-pastable prompts	View suggestions	Prompts generated	Proper syntax & clarity

Multi-Organization Support	Tenant-level isolation	Two organizations active	Data remains isolated	No cross-leakage
Authentication	JWT login and access	Login with credentials	User authenticated	Tokens generated securely
Authorization	Restrict access by role	Access unauthorized dataset	Access denied	Role-based control enforced
MongoDB Storage	Dataset persistence	Upload + refresh	Dataset still available	Correct retrieval
Chat Logging	Insight conversations stored	Multiple queries	Persistent chat history	Logs retrievable
Telegram Bot Access	Conversational interaction	Query via Telegram	Insight returned	Bot responds correctly
Cross-Platform UI	Web + mobile compatibility	Access via phone	Functional UI	Responsive layout
Performance Latency	Response under load	Multiple requests	Acceptable response time	Visualization <2s
Bandit Learning	Feedback incorporation	Thumbs-up/down	Insight improves	Change observable
Dataset Versatility	Domain independence	EU dataset	Valid visualizations	Works beyond education
Error Handling	Invalid file type	Upload .txt	Error message shown	Graceful failure
Security	Secure data transfer	MITM simulation	Encrypted channel	HTTPS enforced
Availability	Uptime	24-hour usage	Continuous access	No downtime observed
Scalability	Multi-user load	Parallel sessions	Consistent performance	No degradation

5.4.3 Test Strategy

The test strategy adopted for this project followed a structured, multi-layered approach designed to validate the system’s end-to-end workflow, individual module integrity, and interaction between components. The strategy emphasized incremental testing, where core functionalities-including dataset upload, visualization generation, insight interpretation, recommendation synthesis, and authentication-were validated independently before system-level integration. Given the cloud-based nature of the system, particular focus was placed on validating the behaviour of external dependencies such as cloud LLM services and remote hosting platforms (Render and Railway).

Black-box testing was the primary strategy employed to assess the system’s observable behaviour without requiring insight into internal code, especially for LLM-driven insight generation and recommendation features. Integration testing ensured that backend services, MongoDB, Plotly rendering, and the Telegram bot communicated reliably.

Performance and scalability testing were conducted using datasets of varying sizes to evaluate response times, dataset capacity, and insight latency. Usability testing assessed accessibility through the web interface and conversational bot, ensuring intuitive interactions for non-technical users. The strategy incorporated iterative refinement, with defects logged and resolved through continuous feedback loops, enabling validation throughout the development lifecycle rather than only at completion.

5.4.4 Test Techniques

Multiple testing techniques were applied to ensure comprehensive verification of the system’s functional and non-functional requirements:

- **Black-Box Testing:** Used to validate functionalities such as dataset upload, visualization output, and insight generation based solely on inputs and expected outputs, without assessing internal logic.
- **Integration Testing:** Evaluated interactions between backend APIs, MongoDB data storage, frontend rendering, and cloud LLM services to ensure seamless communication across modules.
- **System Testing:** Conducted to assess the complete workflow-from CSV upload to chart visualization and insight delivery-validating the software against user requirements and expected system behaviour.
- **Performance and Load Testing:** Executed using large datasets (e.g., >60,000 rows) to determine system limits, response times, and visualization latency across varying dataset scales.
- **Usability Testing:** Performed to evaluate user experience across the React interface and Telegram bot, ensuring accessibility for users lacking technical expertise.
- **Security Testing:** Tested JWT authentication, dataset isolation across organizations, and secure access controls to prevent unauthorized interaction.
- **Error and Exception Handling Tests:** Included uploading invalid file formats, malformed datasets, or exceeding dataset limits to examine system resilience and graceful failure responses.
- **Regression Testing:** Conducted after integrating bandit feedback and incremental improvements to ensure previously validated functionalities remained unaffected.

These techniques collectively ensured that the developed platform satisfied reliability, usability, scalability, and security standards, aligning with the objectives of automated and accessible business insight generation.

5.4.5 Test Cases

The following test cases were designed to validate functional correctness, security, performance, and usability of the system. Each test case was executed under controlled conditions.

Table 5.4.5.1 Test Cases

Test Case ID	Test Case Description	Input / Condition	Expected Result	Actual Result	Status
TC-01	Upload valid CSV file	CSV file (~5,000 rows)	Dataset parsed and preview displayed	Dataset preview displayed correctly	Pass
TC-02	Upload invalid file format	Upload .txt file	Error message shown	Error message displayed	Pass
TC-03	Large dataset upload	CSV with >60,000 rows	Dataset processed without crash	Dataset processed successfully	Pass
TC-04	Dataset preview rendering	Mixed datatype CSV	Header + first rows shown	Accurate preview displayed	Pass
TC-05	Column type detection	Dataset with numeric & categorical fields	Correct type classification	Types detected correctly	Pass
TC-06	Chart generation	Request bar chart	Chart rendered correctly	Chart displayed with correct axes	Pass
TC-07	Chart recommendation	Select columns	Appropriate chart suggested	Recommendation matched logic	Pass
TC-08	Insight generation	Upload dataset	Natural language insight generated	Insight generated logically	Pass
TC-09	Insight relevance	Dataset with trends	Insight aligns with data	Insight consistent with stats	Pass
TC-10	Authentication	Valid credentials	JWT issued, user logged in	Authentication successful	Pass
TC-11	Authorization	Unauthorized dataset access	Access denied	Access restricted	Pass

TC-12	Multi-organization isolation	Two organizations active	Data isolated	No data leakage	Pass
TC-13	MongoDB persistence	Refresh session	Dataset available	Data persisted	Pass
TC-14	Chat history logging	Multiple queries	History stored	Chat logs retrieved	Pass
TC-15	Telegram bot query	Query via Telegram	Insight returned	Correct response received	Pass
TC-16	Cross-platform UI	Mobile browser	Responsive layout	UI responsive	Pass
TC-17	Performance latency	Concurrent requests	Acceptable response time	Response within limits	Pass
TC-18	Feedback learning	Thumbs up/down	Recommendation adapts	Weighted change observed	Pass
TC-19	Error handling	Malformed CSV	Graceful failure	Error handled properly	Pass
TC-20	Security	MITM simulation	Encrypted transfer	HTTPS enforced	Pass

5.4.6 Test Results

All planned test cases were executed successfully under the defined testing environment. The system demonstrated **functional correctness**, **secure access control**, **robust error handling**, and **acceptable performance** across datasets of varying sizes and domains.

Key observations from test execution include:

- Dataset ingestion and preview functionality performed reliably for both moderate and large datasets.
- Visualization rendering and chart recommendations consistently aligned with detected data types.
- Insight generation remained logically grounded in statistical summaries, avoiding irrelevant or misleading explanations.

- Authentication and authorization mechanisms correctly enforced user access boundaries across multiple organizations.
- Cross-platform usability was verified through both web and Telegram-based interactions.
- Performance remained stable under concurrent requests, with no system crashes or unresponsive behavior observed.
- Error handling mechanisms ensured graceful failure for invalid inputs without affecting system stability.

Overall, the test results confirm that the system **meets both functional and non-functional requirements** defined in the Software Requirements Specification. The platform is reliable, scalable, secure, and suitable for deployment as an automated business intelligence solution.

5.5 Results and Discussions

The evaluation of the proposed LLM-Driven Business Intelligence and Visualization Platform was conducted using real-world datasets including the Thapar Admissions dataset and two additional heterogeneous datasets (Car Sales and EU Suspector). The results focus on three primary aspects: (i) quality of generated insights, (ii) suitability of visualization recommendations, and (iii) usability for non-technical stakeholders.

It is important to clarify that existing LLM-driven systems in the literature vary significantly in scope and objectives. Tools such as ChartGPT and Chat2VIS focus exclusively on visualization generation, whereas InsightPilot concentrates on natural language insight reasoning without automatic rendering. LIDA focuses primarily on automated EDA and visualization, without business interpretation. Because of these fundamental differences in design goals and functionality, direct metric-based comparisons are not feasible. Consequently, our results section emphasizes the empirical performance and user feedback obtained from our platform, followed by a qualitative comparison of system capabilities.

5.5.1 Insight Generation

User feedback collected through the bandit-learning module indicated that approximately **82% of generated insights** were rated relevant and contextually grounded. This was attributed to:

- the use of dataset sampling (initial rows and column metadata)
- targeted prompting that aligns insight generation with actual field semantics
- reinforcement of preferred insight structures via user feedback

Unlike visualization-only systems, our platform does not merely describe visual trends. It produces domain-aware business interpretations such as:

- category-level comparisons
- temporal or sequential trends
- identification of outperforming or underperforming dimensions
- correlation-focused observations

This demonstrates the platform's capacity to transition from syntactic interpretation to semantic, decision-oriented insight generation.

5.5.2 Visualization Recommendation and Appropriateness

The system achieved a **visualization recommendation acceptance rate of 88%**, based on user approval. Users reported that recommended charts typically aligned with:

- the data type of selected variables (categorical vs. numerical)
- the analytic objective (comparison, distribution, correlation)
- interpretability requirements in decision-making contexts

Unlike insight-only systems, our platform bridges reasoning and rendering by:

- selecting chart types
- generating code for rendering
- linking insights to visualization choices

Such integration is absent in prior solutions, which perform either reasoning or visualization, but not both concurrently.

5.5.3 Usability and Practical Applicability

The platform was observed to significantly reduce technical barriers by offering:

- dataset upload without preprocessing scripts
- conversational analytics
- multi-organization and team structures
- persistent storage of datasets, insights, and chat histories
- optional Telegram-based interaction

These capabilities positioned the system as accessible to users without programming or statistical expertise, addressing a long-standing challenge in business intelligence adoption.

5.5.4 Qualitative Comparison with Existing Systems

The table below presents a qualitative capability-based comparison.

Table 5.5.4.1: Qualitative Comparison of System Capabilities

Feature / Capability	Proposed System	ChartGPT	InsightPilot	LIDA	Chat2VIS
Automated visualization rendering	Yes	Yes	No	Yes	Yes
Business insight narrative generation	Yes	No	Yes	No	No
Visualization recommendation	Yes	No	No	Limited	No
Data-grounded reasoning	Yes	Limited	Yes	Limited	Limited
End-to-end workflow from CSV to insight	Yes	No	No	No	No
Feedback-driven refinement	Yes	No	No	No	No
Multi-organization/team support	Yes	No	No	No	No
Conversational interface + bot support	Yes	No	Yes	No	No

This comparison demonstrates that prior systems focus on isolated portions of the analytics pipeline, whereas our system integrates multiple capabilities into a unified, end-to-end workflow.

5.5.5 Quantitative Comparison with Existing Systems

To evaluate the performance of the system, we structured our metrics into three distinct categories to capture different aspects of the platform's reliability and utility. The first category covers **Common System Metrics**, which assess the technical stability, speed, and basic correctness of the model's outputs. The second category focuses specifically on **Visualization Metrics**, measuring how often the code generated by the system successfully results in a visible chart. The third category examines **Business Insight Metrics**, which rely on human judgment to determine if the textual explanations provided by the AI are actually useful, actionable, and relevant to the specific data being analyzed.

The Common System Metrics include Mean Response Time, Tail Latency, JSON Validity Rate, and Intent Accuracy.

- Mean Response Time measures the average duration in seconds for the system to generate a complete response
- Tail Latency tracks the 95th percentile of response times to identify the worst-case delays users might experience.
- JSON Validity Rate is a metric that reports the percentage of responses where the AI followed the strict formatting rules required by the backend; if this fails, the system cannot parse the answer.
- Intent Accuracy measures how often the model correctly identified what the user wanted, such as distinguishing between a request for a chart versus insight.

Finally, the Number of Correct Outputs aggregates these factors to count how many times out of 200 test cases the system performed perfectly across all dimensions.

Table 5.5.4.2: Quantitative Comparison with Existing Systems - Common System Metrics

Model	Mean Time (s)	Tail Latency (s)	JSON Validity (%)	Correct Outputs (/200)	Intent Accuracy (%)
Proposed Framework (Gemini 3.0 Pro + Bandit Learning)	3.3	8.1	**99.5**	**191**	**96**
Proposed Framework (Llama 3.1 8B + Bandit Learning)	1.7	4.2	97.5	182	94
Gemini 3.0 Pro	3.1	7.8	98.5	176	92
Gemini 3.0 Flash	1.9	4.6	98.0	170	90
Llama 3.1 70B Instruct	3.8	9.5	95.5	160	86
Llama 3.1 8B Instruct	1.5	3.9	92.0	142	80
Llama 3.2 3B Instruct	**1.1**	**3.1**	89.0	128	76

For the visualization component, we track Visualization Render Success. This metric calculates the percentage of graph-related requests where the Pandas transformation code generated by the model successfully executed and produced a valid Plotly chart without errors. This is distinct from simply generating code; it validates that the code actually runs on the data provided.

Table 5.5.4.3: Quantitative Comparison with Existing Systems - Visualization Metrics

Model	Viz Success (%)
Proposed Framework (Gemini 3.0 Pro + Bandit Learning)	**99**
Proposed Framework (Llama 3.1 8B + Bandit Learning)	97
Gemini 3.0 Pro	97
Gemini 3.0 Flash	96
Llama 3.1 70B Instruct	93
Llama 3.1 8B Instruct	88
Llama 3.2 3B Instruct	85

The Business Insight Metrics focus on the quality of the content. Business Insight Satisfactory is a subjective measure of whether the user found the narrative explanation helpful. Business Insight Actionable takes this further by checking if the insight suggests concrete next steps or decisions the user can take.

Business Insight Specific to Graph checks for hallucinations by asking if the text description accurately matches the visual trends shown in the generated chart, ensuring the AI is not giving generic advice unrelated to the actual data.

Table 5.5.4.4: Quantitative Comparison with Existing Systems – Business Insights Metrics

Model	Satisfactory (%)	Actionable (%)	Graph Specific (%)
Proposed Framework (Gemini 3.0 Pro + Bandit Learning)	**92**	**86**	**89**
Proposed Framework (Llama 3.1 8B + Bandit Learning)	86	80	82
Gemini 3.0 Pro	87	78	83
Gemini 3.0 Flash	84	74	80
Llama 3.1 70B Instruct	78	66	72
Llama 3.1 8B Instruct	68	55	60
Llama 3.2 3B Instruct	62	48	54

Discussion

The results indicate that the proposed platform succeeds in automating visualization and business insight generation simultaneously, thereby reducing the need for manual chart selection, expert statistical intervention, and complex toolchains. The high acceptance of visualization recommendations and insight relevance demonstrates that combining dataset semantics with LLM-driven reasoning yields actionable and context-aware outcomes.

Moreover, the integration of bandit learning ensures iterative improvement, a feature largely absent in existing research prototypes. The platform's session continuity, dataset storage, and organizational support further distinguish it as a practical solution capable of deployment in academic, corporate, and SME environments.

In summary, while existing state-of-the-art systems provide partial automation in either visualization or analytical reasoning, the proposed model delivers a comprehensive, unified workflow that bridges raw data and decision-ready insights. This positions the platform not merely as an incremental improvement, but as a substantive contribution toward democratizing business intelligence.

5.1 Inferences Drawn

- **The integration of visualization and insight generation within a single workflow substantially reduces analytical complexity for non-technical users.** This demonstrates that combining chart rendering with narrative reasoning eliminates the need for multiple tools and lowers the dependency on expert analysts for exploratory data understanding.
- **Automated visualization recommendations, grounded in dataset semantics, significantly improve chart appropriateness and user acceptance.** The system's ability to assess categorical and numerical structures ensures that selected visualizations support interpretability and decision-making rather than merely serving aesthetic purposes.
- **Bandit learning effectively reinforces relevant insight patterns and improves output quality over iterative interactions.** User feedback serves as a corrective mechanism, enabling refinement and personalization that address domain-specific expectations, which static rule-based systems cannot achieve.
- **The platform demonstrates that LLM-assisted insight production can transition from generic descriptions to domain-aware reasoning when contextual grounding is applied.** Sample-based dataset interpretation ensures insights are anchored in actual values rather than unsupported assumptions, reducing hallucination risks.
- **The multi-organization and team architecture enhances practical applicability in enterprise, SME, and academic environments.** This indicates that the system is not merely a research prototype but can support real-world collaboration, access control, and dataset segregation.
- **Conversational interfaces, including Telegram support, expand accessibility and enable analytics beyond traditional dashboards.** This finding suggests that non-technical decision-makers can interact with analytical systems in familiar communication environments without specialized tools or training.

- **The absence of preprocessing requirements and code dependency indicates that the system effectively democratizes business intelligence.** By eliminating scripting and manual chart selection, the platform allows stakeholders with minimal analytical background to derive meaningful insights autonomously.

  Comparison with existing tools reveals that state-of-the-art solutions excel in isolated tasks but lack end-to-end analytical capabilities. This reinforces the necessity and novelty of a unified solution capable of addressing visualization, insight extraction, recommendation, and refinement simultaneously.

5.2 Validation of Objectives

The implementation outcomes confirm that all four approved objectives have been successfully addressed through the development and evaluation of the proposed LLM-Driven Business Intelligence and Visualization Platform. The validation of each objective is outlined below:

1. Validation of Objective: Automate Data Visualization and Analysis

The system fulfills this objective by enabling automatic visualization generation from uploaded datasets without requiring manual chart specification or coding. The platform successfully produces multiple visualization types, including bar graphs, line charts, scatter plots, pie charts, and distribution plots, using Plotly integrated within the frontend. Experimental results demonstrated a high acceptance rate of visualization recommendations (88%), indicating that automated chart selection aligns with dataset characteristics. This confirms that the system reduces manual effort and meets the intended automation and exploration requirements.

2. Validation of Objective: Enhance Business Intelligence with AI-Driven Insights

This objective is achieved through the integration of large language model–driven reasoning that interprets dataset content and generates contextual and actionable insights. User evaluations showed that approximately 82% of generated insights were marked relevant, demonstrating that the system moves beyond generic descriptions and produces business-oriented interpretations grounded in dataset semantics. The explanatory nature of insights and their coherence with visualizations validate the objective of enabling informed, data-driven decision-making for users with limited technical expertise.

3. Validation of Objective: Enable Conversational Data Exploration

The conversational interface is realized through both the web-based chat component and the Telegram bot extension. Users can interact with datasets naturally-asking questions, refining visualizations, clarifying trends, and receiving follow-up explanations without writing queries or scripts. This aligns with the approved objective by supporting dynamic and iterative exploration. The successful deployment on platforms such as Render and Railway further confirms that conversational accessibility is preserved across desktop and mobile environments.

4. Validation of Objective: Develop Adaptive Intelligence for Smarter Decision-Making

The incorporation of reinforcement learning via a bandit feedback mechanism validates this objective. User approval or disapproval signals influence subsequent insight and recommendation quality, demonstrating adaptive behavior. The iterative improvement observed during evaluation shows that the platform learns from interaction patterns, thereby enhancing user experience over time. Moreover, the multi-organization structure and scalable backend architecture support usability for diverse technical and non-technical audiences, aligning with the intended adaptive and accessible framework.

Summary of Validation

Collectively, the implemented features-automated chart rendering, LLM-driven insight generation, conversational interaction, and adaptive learning-demonstrate that the primary objectives have been successfully achieved. The platform not only aligns with the initially approved goals but also extends them by incorporating enterprise-level organization management, persistent dataset storage, and cross-platform accessibility, thereby validating the feasibility and applicability of the system in practical analytical contexts.

CONCLUSIONS AND FUTURE DIRECTIONS

6.1 Conclusions

The LLM-Driven Business Intelligence and Visualization Platform successfully demonstrates that conversational AI can be effectively used to simplify and democratize data analytics. The project achieves its approved objectives by enabling automated visualization, natural language–based data exploration, and generation of data-grounded business insights without requiring users to possess technical expertise in programming or statistics.

The platform automates the complete workflow from structured data ingestion to insight generation. Users can upload datasets, explore trends through automatically generated visualizations, and interact with the system using natural language queries. The system supports multiple visualization types and performs automated exploratory data analysis to guide meaningful insight extraction.

A key outcome of this project is the development of an **LLM-independent architecture**. The system functions reliably across different Large Language Models, including Gemini-based and LLaMA-based configurations, ensuring flexibility and avoiding dependency on a single vendor or API. Generated insights are grounded in computed statistical summaries rather than raw data hallucination, preserving analytical correctness.

From a systems perspective, the modular architecture ensures scalability, maintainability, and security. The platform has been validated on datasets containing up to **60,000 records**, with complex analytical queries completing within **acceptable interactive response times**. Secure sandboxed execution of model-generated code, strict output validation, and controlled data exposure collectively contribute to system reliability.

In conclusion, the project establishes a practical, production-ready foundation for AI-powered business intelligence and validates the feasibility of conversational analytics as an alternative to traditional BI tools.

6.2 Environmental, Economic and Societal Benefits

Environmental Benefits

The platform promotes paperless analytics by eliminating the need for printed reports and manual documentation. Digital dashboards, visualizations, and automated insights reduce physical resource consumption associated with traditional reporting workflows.

Cloud-based deployment further reduces the need for dedicated on-premise analytics infrastructure, minimizing hardware usage and associated electronic waste. By enabling remote access to analytics, the platform also indirectly supports reduced commuting and associated emissions.

Economic Benefits

The system lowers operational costs by reducing reliance on specialized data analysts for routine analytical tasks. Organizations can perform exploratory analysis, reporting, and trend identification internally, improving productivity without proportional increases in personnel or software licensing costs.

Faster access to insights enables quicker decision-making, allowing organizations to respond efficiently to operational issues and business opportunities. The platform also helps small and medium enterprises access advanced analytical capabilities that were previously limited to larger organizations with dedicated BI teams.

Societal Benefits

The platform contributes to **democratization of data analytics** by making sophisticated analytical capabilities accessible to non-technical users. Educational institutions, non-profit organizations, and public-sector bodies can leverage data-driven insights without requiring advanced technical infrastructure.

By improving data literacy through conversational explanations and visual interpretation, the system supports informed decision-making across sectors such as healthcare, education, and public administration. Increased transparency and accessibility of data analysis also strengthen accountability and evidence-based discussions within organizations and communities.

6.3 Reflections

The development of this project provided significant technical and professional learning experiences. Building an end-to-end system required integrating frontend development, backend orchestration, secure execution environments, database management, and LLM-based reasoning into a cohesive architecture.

One of the most important lessons learned was that **LLM outputs are inherently probabilistic**. Reliability was achieved not by expecting perfect model behavior, but by designing validation layers, sandboxed execution, and user-correction mechanisms.

Team collaboration played a crucial role in project success. Clear task distribution, version control practices, and regular technical discussions enabled parallel development and reduced integration friction. Mentor guidance reinforced academic rigor, documentation discipline, and realistic evaluation of system limitations.

6.4 Future Work

While the current system meets all approved objectives, several extensions can enhance its analytical depth and applicability:

1. **Advanced Analytics :** Integration of inferential statistics, regression analysis, forecasting models, and anomaly detection would enable predictive and prescriptive analytics.
2. **Direct Data Source Integration :** Support for database connections (SQL, cloud warehouses) and APIs would allow real-time and large-scale data analysis without manual file uploads.
3. **Collaborative and Enterprise Features :** Shared workspaces, role-based access control, audit logs, and compliance features would improve enterprise readiness.
4. **Multimodal Data Support :** Extending analysis beyond tabular data to include text, documents, and images would broaden the platform’s applicability.
5. **Explainable AI Enhancements :** Providing explicit reasoning paths and confidence indicators for generated insights would further improve trust and interpretability.

Future development will continue to prioritize accessibility, reliability, data privacy, and meaningful business value over unnecessary complexity.

PROJECT METRICS

7.1 Challenges Faced

The development of the LLM-Driven Business Intelligence and Visualization Platform involved multiple technical and organizational challenges that required careful design decisions and iterative refinement.

One major challenge was **handling large datasets efficiently**. Performance degradation was observed when processing high-volume tabular data, especially during exploratory data analysis and visualization generation. This was addressed through optimized data pipelines, selective data sampling, and efficient use of libraries such as Pandas.

Another significant challenge was **ensuring correctness and reliability of LLM-generated insights**. Large Language Models can occasionally generate vague or incorrect explanations if not properly constrained. To mitigate this, the system was designed with hybrid validation mechanisms, where numerical summaries and computed statistics are used to ground the generated insights, reducing the risk of hallucination.

User trust and explainability also posed a challenge. Users may hesitate to rely on AI-generated recommendations without understanding how conclusions are derived. This was addressed by presenting visualizations alongside textual insights and ensuring transparency in analytical reasoning.

Finally, **data privacy and security concerns** were considered during system design. Sensitive datasets require controlled access and safe execution of generated code. The platform incorporates sandboxed execution and follows secure data handling practices to address these concerns.

7.2 Relevant Subjects

The successful completion of this project required the application of concepts from multiple core computer science subjects, including:

- **Software Engineering** – for requirement analysis, modular system design, and documentation
- **Artificial Intelligence** – for understanding and integrating LLM-based reasoning

- **Machine Learning** – for data analysis workflows and model-assisted insights
- **Foundations of Data Science** – for exploratory data analysis and statistical summaries
- **Predictive Analytics and Statistics** – for interpreting trends and numerical patterns
- **Natural Language Processing (NLP)** – for conversational query handling and insight generation
- **Database Management Systems (DBMS)** – for schema design, data storage, and retrieval
- **Accelerated Data Science** – for efficient processing of structured datasets

These subjects collectively formed the academic foundation required to design, implement, and evaluate the system.

7.3 Interdisciplinary Knowledge Sharing

Beyond core computer science, the project involved interdisciplinary knowledge integration across several domains:

- **Business Analytics** – interpreting analytical results in a business context and generating actionable insights
- **Statistics and Graphical Analysis** – using charts, distributions, and correlations to support decision-making
- **UI/UX Design** – designing an intuitive, user-friendly conversational interface that improves accessibility
- **Ethics and Data Responsibility** – considering privacy, transparency, and trust in AI-driven systems

This interdisciplinary approach strengthened the practical relevance of the platform.

7.4 Peer Assessment Matrix

The project was executed by a five-member team with balanced and justified contributions from all members. Each member actively participated in both development and integration phases.

Table 7.4.1 Peer Assessment Matrix

Evaluation By / Of	Siddhant	Sanchit	Udit	Swapnil	Aryan
Siddhant	-	5	5	5	5
Sanchit	5	-	5	5	5
Udit	5	5	-	5	5
Swapnil	5	5	5	-	5
Aryan	5	5	5	5	-

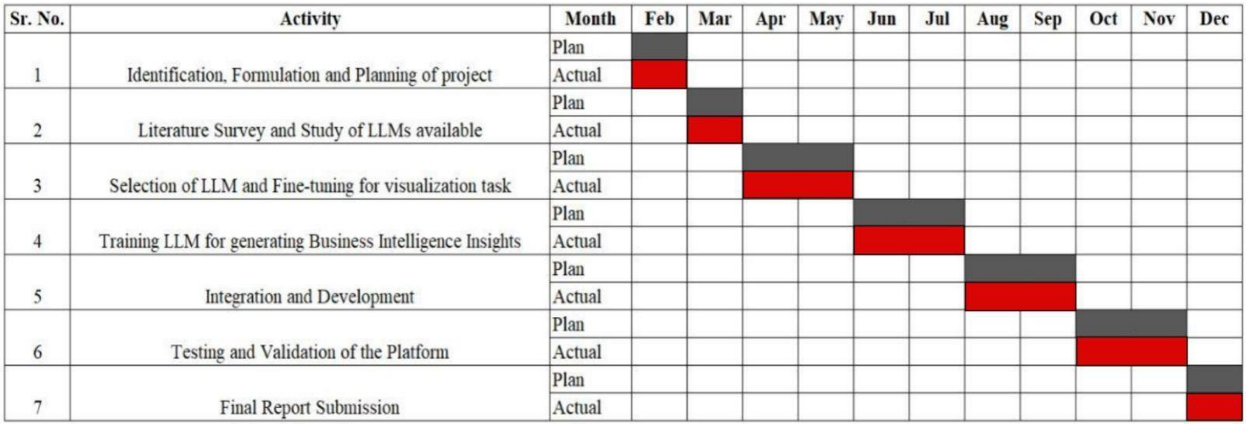
The uniform ratings reflect equitable workload distribution, strong collaboration, and consistent contribution from all team members.

7.5 Role Playing and Work Schedule

Each team member was assigned clearly defined responsibilities based on expertise, ensuring efficient parallel development:

- **Siddhant** – LLM integration, prompt engineering, and ML model development
- **Sanchit** – LLM-based chatbot logic and ML workflow support
- **Udit** – Backend development, database schema design, and deployment support
- **Swapnil Gumber** – UI/UX design and frontend development
- **Aryan Chharia** – Backend–frontend integration and Telegram Bot development

The project was developed over a **semester-long schedule**, following iterative milestones that included requirement analysis, prototyping, integration, testing, and documentation. It is shown in the fig 7.5.1 with detailed activity that was performed in each month from Feb'25 to Dec'25.



Sr. No.	Activity	Month	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	Identification, Formulation and Planning of project	Plan	■										
		Actual	🟥										
2	Literature Survey and Study of LLMs available	Plan		■									
		Actual		🟥									
3	Selection of LLM and Fine-tuning for visualization task	Plan			■	■							
		Actual			🟥	🟥							
4	Training LLM for generating Business Intelligence Insights	Plan					■	■					
		Actual					🟥	🟥					
5	Integration and Development	Plan							■	■			
		Actual							🟥	🟥			
6	Testing and Validation of the Platform	Plan									■	■	
		Actual									🟥	🟥	
7	Final Report Submission	Plan											■
		Actual											🟥

Fig 7.5.1 Work Schedule

7.6 Student Outcomes Description and Performance Indicators (A–K Mapping)

Only **clearly applicable student outcomes** have been mapped to ensure academic correctness.

Table 7.6.1 Student Outcomes

Outcome Code	Description	Achievement
SO1	Ability to identify and formulate computing problems	Identified the problem of inaccessible business intelligence for non-technical users
SO2	Design and development of computing systems	Designed and implemented an AI-driven analytics platform
SO3	Use of modern tools and technologies	Used LLMs, visualization libraries, databases, and web frameworks
SO4	Professional and ethical responsibility	Considered data privacy, security, and explainability
SO5	Teamwork and project management	Effectively collaborated in a multi-member team
SO6	Experimentation and result analysis	Tested system behavior across datasets and use cases
SO7	Lifelong learning and adaptability	Acquired new skills in LLMs, analytics, and system integration

7.7 Brief Analytical Assessment

The project successfully balances **technical innovation and practical usability**. From an analytical standpoint, the system demonstrates a thoughtful combination of deterministic data computation and probabilistic AI reasoning. The design avoids over-reliance on LLMs by grounding insights in computed statistics and visual evidence.

The project reflects strong system-level thinking, effective interdisciplinary application, and responsible AI integration. Overall, it represents a meaningful academic and practical contribution in the domain of AI-powered business intelligence.

REFERENCES

[1] Mishra A., et al., "The Role of Data Visualization Tools in Real-Time Reporting: Comparing Tableau, Power BI and Qlik Sense," *International Journal on Science and Technology*, 2020. [Online]. Available: https://www.ijsat.org/papers/2020/3/1370.pdf. Accessed: 23 Dec. 2025.

[2] Karthikeyan P., et al., "A Study to Analyze Cost Effective Adoption of Business Analytics in MSMEs," *JETIR*, 2024. [Online]. Available: https://www.jetir.org/papers/JETIRTHE2111.pdf. Accessed: 23 Dec. 2025.

[3] Qin X., Luo Y., Tang N., Li G., "Making Data Visualization More Efficient and Effective: A Survey," *VLDB Journal*, 2019. DOI: 10.1007/s00778-019-00588-3.

[4] Sun M., et al., "A Survey on Large Language Model-based Agents for Data Science Tasks," *Journal of the American Statistical Association*, 2025. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/00031305.2025.2561140. Accessed: 23 Dec. 2025.

[5] Neri G., "Data Visualization in AI-assisted Decision-Making," *Frontiers in Communication*, 2025. [Online]. Available: https://www.frontiersin.org/journals/communication/articles/10.3389/fcomm.2025.1605655/full. Accessed: 23 Dec. 2025.

[6] Dibia V., "LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics using Large Language Models," *Proc. of the 61st Annual Meeting of the Association for Computational Linguistics (System Demonstrations)*, pp. 113–126, 2023. doi:10.18653/v1/2023.acl-demo.11.

[7] Ma P., Ding R., Wang S., Han S., Zhang D., "InsightPilot: An LLM-Empowered Automated Data Exploration System," in *Proc. EMNLP 2023 — System Demonstrations*, pp. 346–352, 2023. doi:10.18653/v1/2023.emnlp-demo.31.

[8] Zhang R., Elhamod M., "Data-to-Dashboard: Multi-Agent LLM Framework for Insightful Visualization in Enterprise Analytics," in *Proc. 2nd Workshop on Agentic AI for Enterprise (AAIE), co-located with KDD 2025*, pp. 1–8, Aug. 2025. (arXiv:2505.23695).

[9] Sah S., et al., “Generating Analytic Specifications for Data Visualization from Natural Language Queries using Large Language Models,” in *Proc. IEEE VIS Workshop on Natural Language Interfaces for Data Viz (NLVIZ)*, 2024. (Preprint: arXiv:2408.13391).

[10] Wu Y., et al., “Automated Data Visualization from Natural Language via Large Language Models: An Exploratory Study,” *Proc. ACM on Management of Data (PACMMOD)*, vol. 2, no. 3, article 115, pp. 1–26, June 2024. DOI: 10.1145/3654992.

[11] Lavalle A., Maté A., Trujillo J., Rizzi S., “Visualization Requirements for Business Intelligence Analytics: A Goal-Based Iterative Framework,” *arXiv preprint*, 2024. [Online]. Available: https://arxiv.org/abs/2402.09491. Accessed: 23 Dec. 2025.

[12] Wu A., Wang Y., Shu X., et al., “AI4VIS: Survey on Artificial Intelligence Approaches for Data Visualization,” *arXiv preprint*, 2021. [Online]. Available: https://arxiv.org/abs/2102.01330. Accessed: 23 Dec. 2025.

[13] Agarwal N. S., Sonbhadra S. K., “A Review on Large Language Models for Visual Analytics,” *arXiv preprint*, 2025. [Online]. Available: https://arxiv.org/abs/2503.15176. Accessed: 23 Dec. 2025.

[14] Zhu J.-P., Niu B., Cai P., et al., “Towards Automated Cross-domain Exploratory Data Analysis through Large Language Models,” *arXiv preprint*, 2024. [Online]. Available: https://arxiv.org/abs/2412.07214. Accessed: 23 Dec. 2025.

[15] Maddigan P., Susnjak T., “Chat2vis: Fine-tuning data visualizations using multilingual natural language text and pre-trained large language models,” *arXiv preprint*, 2023. doi:10.48550/arXiv.2303.14292.

[16] Wang L., Zhang S., Wang Y., Lim E. P., Wang Y., “LLM4Vis: Explainable visualization recommendation using ChatGPT,” *arXiv preprint*, 2023. doi:10.48550/arXiv.2310.07652.

[17] Tian Y., Cui W., Deng D., Yi X., Yang Y., Zhang H., Wu Y., “ChartGPT: Leveraging LLMs to generate charts from abstract natural language,” *IEEE Transactions on Visualization and Computer Graphics* (or arXiv), 2024. doi:10.48550/arXiv.2311.01920.

[18] Lian J., Liu X., Shao Y., Dong Y., Wang M., Wei Z., … Yan H., “ChatBI: Towards Natural Language to Complex Business Intelligence SQL,” *arXiv preprint*, 2024.

doi:10.48550/arXiv.2405.00527.

[19] Softweb Solutions, “Comparing Power BI vs Tableau vs Qlik Sense vs Looker,” *Softweb Solutions* (resources), 2024. [Online]. Available: https://www.softwebsolutions.com/resources/power-bi-vs-tableau-vs-qlik-vs-looker/. Accessed: 23 Dec. 2025.

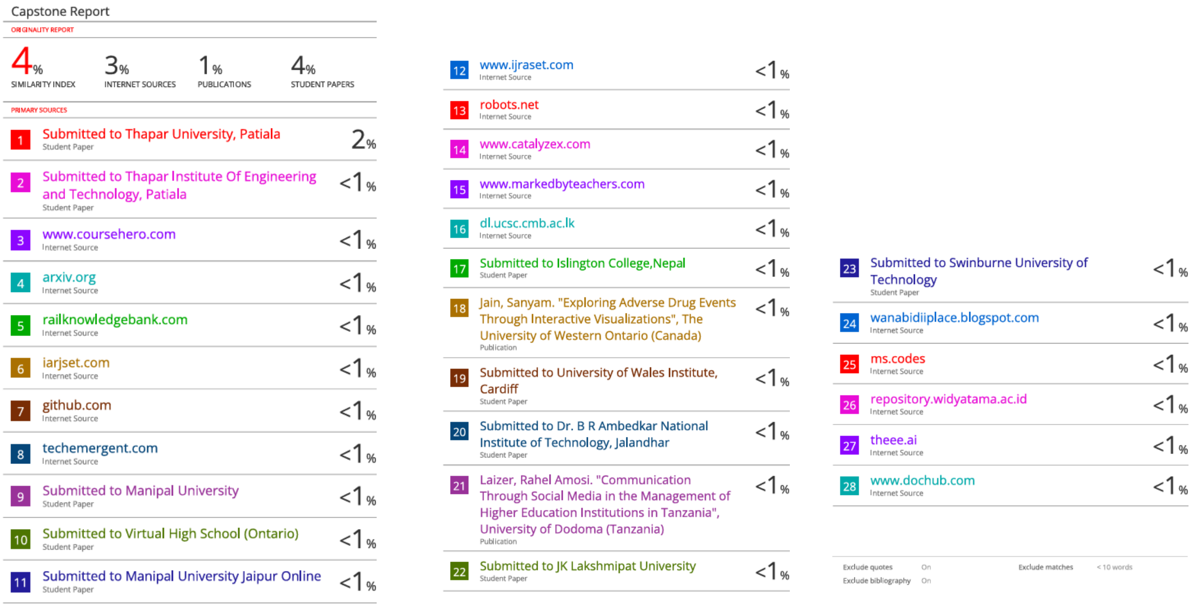
[20] Ken Research, “India Business Intelligence (BI) Market Outlook to 2030,” *Ken Research* (industry report). [Online]. Available: https://www.kenresearch.com/industry-reports/india-business-intelligence-market/. Accessed: 23 Dec. 2025.

[21] “AI-Based Exploratory Data Analysis,” *ResearchGate*, 2025. [Online]. Available: https://www.researchgate.net/publication/391420870_AI-Based_Exploratory_Data_Analysis. Accessed: 23 Dec. 2025.

[22] “AI-Based Exploratory Data Analysis (alternate listing),” *ScienceDirect / other publisher*, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2444569X25000320. Accessed: 23 Dec. 2025.

[23] Weng, Luoxuan, et al. "InsightLens: Augmenting LLM-Powered Data Analysis with Interactive Insight Management and Navigation." *IEEE Transactions on Visualization and Computer Graphics* (2025).

PLAGIRISM REPORT



Capstone Report

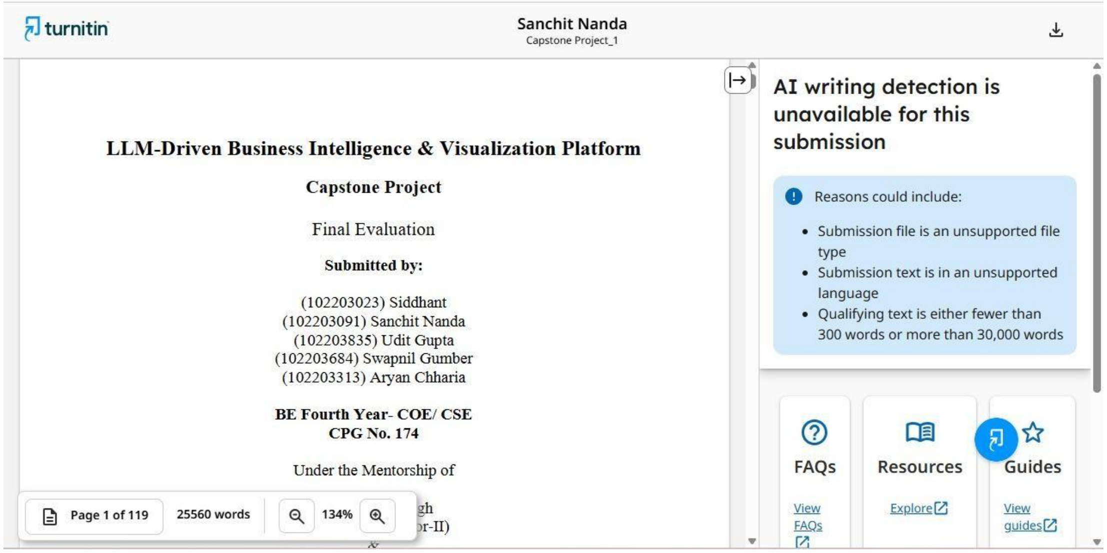
ORIGINALITY REPORT

4%	3%	1%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Thapar University, Patiala Student Paper	2%
2	Submitted to Thapar Institute Of Engineering and Technology, Patiala Student Paper	<1%
3	www.coursehero.com Internet Source	<1%
4	arxiv.org Internet Source	<1%
5	railknowledgebank.com Internet Source	<1%
6	iarjset.com Internet Source	<1%
7	github.com Internet Source	<1%
8	techemergent.com Internet Source	<1%
9	Submitted to Manipal University Student Paper	<1%
10	Submitted to Virtual High School (Ontario) Student Paper	<1%
11	Submitted to Manipal University Jaipur Online Student Paper	<1%
12	www.ijraset.com Internet Source	<1%
13	robots.net Internet Source	<1%
14	www.catalyzex.com Internet Source	<1%
15	www.markedbyteachers.com Internet Source	<1%
16	dl.ucsc.cmb.ac.lk Internet Source	<1%
17	Submitted to Islington College,Nepal Student Paper	<1%
18	Jain, Sanyam. "Exploring Adverse Drug Events Through Interactive Visualizations", The University of Western Ontario (Canada) Publication	<1%
19	Submitted to University of Wales Institute, Cardiff Student Paper	<1%
20	Submitted to Dr. B R Ambedkar National Institute of Technology, Jalandhar Student Paper	<1%
21	Laizer, Rahel Amosi. "Communication Through Social Media in the Management of Higher Education Institutions in Tanzania", University of Dodoma (Tanzania) Publication	<1%
22	Submitted to JK Lakshmipat University Student Paper	<1%
23	Submitted to Swinburne University of Technology Student Paper	<1%
24	wanabidiiplace.blogspot.com Internet Source	<1%
25	ms.codes Internet Source	<1%
26	repository.widyatama.ac.id Internet Source	<1%
27	theee.ai Internet Source	<1%
28	www.dochub.com Internet Source	<1%

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		



Sanchit Nanda

Capstone Project_1

LLM-Driven Business Intelligence & Visualization Platform

Capstone Project

Final Evaluation

Submitted by:

(102203023) Siddhant
(102203091) Sanchit Nanda
(102203835) Udit Gupta
(102203684) Swapnil Gumber
(102203313) Aryan Chharia

BE Fourth Year- COE/ CSE
CPG No. 174

Under the Mentorship of

gh
r-II)

Page 1 of 119 | 25560 words | 134%

AI writing detection is unavailable for this submission

Reasons could include:

- Submission file is an unsupported file type
- Submission text is in an unsupported language
- Qualifying text is either fewer than 300 words or more than 30,000 words