

HBL Internship Report

Alina Siddiqui

Project Title: HBL Visualization Assistant: An LLM-Powered Business Insight Tool



HBL Visualization Assistant

Ask a question about the deposit data

e.g. Show total deposits by branch in June

Name: Alina Siddiqui

Institute: Institute of Business Administration (IBA)

IBA ERP: 26555

Date Of Submission: 29/7/25

Supervisor: Hassaan Muslim

Dept: IT

1. Abstract

The **HBL Visualization Assistant** is an intelligent, interactive data analytics tool designed to simplify and enhance business insights for deposit trends at HBL (Habib Bank Limited). Powered by large language models (LLMs) via OpenRouter and Together AI, the assistant allows users to ask natural language questions related to deposit transaction data—such as branch-wise comparisons, monthly summaries, and customer segment analyses—and receive visual and textual answers in real-time.

The system is built using **Streamlit** for the frontend interface, **Pandas** and **Matplotlib** for data processing and chart generation, and **LLMs** for generating Python code and summarizing results. Upon receiving a query, the assistant leverages **OpenRouter's Claude-3-Haiku** model to generate Python code that visualizes the data as bar charts, line plots, pie charts etc. Simultaneously, it uses **Meta LLaMA 3** via **Together AI** to generate clear, human-readable business insights based on aggregated summaries.

This approach minimizes the need for manual analysis and technical expertise, offering bank executives, analysts, and decision-makers a fast, flexible, and intelligent tool to derive data-driven conclusions from complex deposit datasets. The assistant also includes robust fallback logic and dynamic prompt engineering, ensuring resilience even when models fail to produce valid code.

2. Table of Contents

1. Abstract	2
2. Table of Contents	3
3. Introduction	5
4. Business Problem	6
5. Project Objectives	7
Business Objectives	7
Technical Objectives	7
6. Tools and Technologies	8
6.1 Programming Language	8
6.2 Data Handling and Processing	8
6.3 Data Visualization	8
6.4 Frontend Development	8
6.5 Large Language Model APIs	9
6.6 Environment and Security	9
6.7 Development and Testing Tools	9
6.8 Dataset	9
7. Dataset Description	10
7.1 Nature of the Data	10
7.2 Columns Used	10
7.3 Scalability and Future Use	11
8. System Architecture	12
9. Module Breakdown	13
9.1 Frontend UI (Streamlit)	13
9.2 Code Generation Module	13
9.3 Insight Generation Module	14
9.4 Query Interpretation Logic	14
10. Implementation Details	16
10.1 Streamlit Frontend Integration	16
10.2 Prompt Engineering for Code Generation	16
10.3 API Integration	17
OpenRouter API for Chart Code	17
TogetherAI API for Insights	17
10.4 Query Interpretation and Column Mapping Logic	18
10.5 Error Handling and Fallbacks	18
10.6 Insights Generation Logic	19
11. Testing and Validation	20
11.1 Functional Workflow Testing	20
11.2 Language Model Testing & Comparison	20

Chart Code Generation Testing	20
Insight Generation Testing	21
11.3 Error Handling and Resilience	21
11.4 UI/UX Testing	22
12. Results and Screenshots	23
12.1 Query-Based Results	23
13. Challenges Faced	28
13.1 Finding Suitable LLM APIs	28
13.2 Insight Accuracy & Hallucination	28
13.3 Prompt Engineering	28
13.4 Model Segmentation Strategy	29
13.5 Small Dataset & Testing Constraints	29
14. Conclusion & Future Work	30
14.1 Conclusion	30
14.2 Future Work	30
Integration with Power BI APIs	30
Fine-Tuning Models for Banking Domain	30
Larger-Scale Dataset Integration	31
Voice-to-Query and Mobile Support	31

3. Introduction

In the digital era, data-driven decision-making has become critical for the banking sector. Banks accumulate vast amounts of transactional data daily, but deriving actionable insights from it remains a challenge—especially for non-technical users. This project, titled **HBL Visualization Assistant**, aims to bridge that gap by creating an intelligent, user-friendly assistant capable of generating meaningful visualizations and business insights from deposit transaction data using natural language queries.

Built using **Streamlit** as the frontend and powered by **Large Language Models (LLMs)** such as **Claude** (via OpenRouter) and **LLaMA-3** (via Together AI), the assistant can understand a wide range of user queries and dynamically generate both charts and bullet-point insights. The system reads structured data from HBL's deposit transaction records, interprets questions using natural language processing, and produces context-aware outputs to support branch-level performance tracking, customer behavior analysis, and deposit growth trends.

This project not only showcases the capabilities of modern LLMs in business intelligence but also serves as a proof of concept for AI-driven self-service analytics tools in the banking industry.

4. Business Problem

The Gap Between Data Availability and Business Usability

Banks like HBL maintain detailed data on every deposit transaction, including account type, branch, customer segment, and region etc. However, the process of analyzing this data is often bottlenecked by the need for SQL skills, data science tools, or support from central IT teams. This results in:

- Delayed decision-making at the branch or regional level
- Underutilization of valuable historical deposit trends
- Difficulty comparing performance across segments or locations
- Inability for managers to explore data in real time during strategy meetings

Why This Matters

For a data-rich organization like HBL, **empowering operational staff with instant access to visual and contextual insights** can lead to:

- Faster identification of high-performing or underperforming branches
- Better allocation of marketing or retention budgets
- Sharper targeting of customer segments (e.g., personal vs business accounts)
- Stronger accountability and reporting

5. Project Objectives

The HBL Visualization Assistant is designed to solve the above problem by meeting the following technical and business objectives:

Business Objectives

1. **Empower Decision Makers Without Technical Skills**
Allow branch or regional managers to explore deposit trends and customer patterns through plain-language queries.
2. **Support Real-time, Data-Driven Decision Making**
Enable instant chart generation and summary insights that can be used in meetings, presentations, or planning documents.
3. **Increase Data Accessibility and Usability Across Teams**
Break the dependency on IT or BI teams by offering a self-service tool accessible to any employee with basic banking knowledge.

Technical Objectives

4. **Use LLMs to Translate Natural Language to Code**
Implement OpenRouter-based models (Claude, DeepSeek) to parse queries and auto-generate Matplotlib visualizations.
5. **Generate Business Insights from Aggregated Data**
Use Together AI (Meta LLaMA-3) to produce 3–5 bullet-point summaries tailored to the user's question and grouped data.
6. **Implement Smart Grouping Logic Based on Query Intent**
Automatically detect whether the user is asking by branch, region, customer segment, account type, etc., and group accordingly.
7. **Deliver a Clean, On-brand User Interface**
Build a Streamlit app that reflects HBL's visual identity and provides a smooth, professional experience.

6. Tools and Technologies

The development of the HBL Visualization Assistant leveraged a variety of tools, libraries, APIs, and platforms to create a seamless and intelligent business intelligence interface. Below is a comprehensive list of the technologies used, categorized by their functionality.

6.1 Programming Language

- Python 3.10+
Used as the core programming language for data processing, API integration, visualization generation, and overall application logic.

6.2 Data Handling and Processing

- Pandas
For loading, cleaning, grouping, and transforming transactional data from Excel files.
- NumPy (*indirectly used*)
For numerical operations when handling large deposit amounts and converting to readable formats.

6.3 Data Visualization

- Matplotlib
Used to dynamically generate charts (bar charts, line graphs, etc.) based on user queries. The model-generated code uses this library to render visualizations.

6.4 Frontend Development

- Streamlit
Used to build an interactive and user-friendly web interface. Streamlit enabled:
 - Query input from the user
 - Display of generated charts
 - Display of AI-generated insights
 - Custom styling using HTML and CSS injection

6.5 Large Language Model APIs

- OpenRouter API (Claude Model)

Used for:

- Natural language understanding of user queries
- Generating Python matplotlib code to visualize the relevant data
- Responding to prompts securely via <https://openrouter.ai> endpoint

- Together AI API (LLaMA 3)

Used to:

- Generate concise business insights in bullet points
- Interpret summarized tables produced from the dataset
- Act as a virtual business analyst

6.6 Environment and Security

- Python-dotenv

Used to securely load API keys from the .env file rather than hardcoding credentials.

6.7 Development and Testing Tools

- Visual Studio Code (VS Code)

Primary IDE for writing and debugging Python code.

- Command Line / Terminal

Used for running and debugging the Streamlit app locally.

6.8 Dataset

- Excel (.xlsx)

Excel-based dummy deposit transaction file (Deposit_Chatbot_Data_Schema.xlsx)

7. Dataset Description

The dataset used in this project was a dummy Excel-based deposit transaction dataset designed to simulate real-world bank deposit activity. While the current project used a sample file with limited scope, the underlying framework is scalable and can be extended to process live or larger datasets in the future.

7.1 Nature of the Data

- **Type:** Synthetic / Dummy data
- **File Name:** Deposit_Chatbot_Data_Schema.xlsx
- **Total Records:** Approximately 2,000 rows
- **Date Range:** 1st June 2024 to 5th June 2024 (5 days)

7.2 Columns Used

The following key columns were utilized in the project for querying, aggregation, chart generation, and insight formulation:

Column Name	Description
transaction_id	Unique identifier for each transaction
date	Timestamp of the deposit transaction
branch_id	Unique identifier for the HBL branch
branch_name	Name of the HBL branch (e.g., Clifton, Gulshan)
city	City in which the transaction took place
region	Geographic region (e.g., South, Central)
deposit_amount	Amount of money deposited in Pakistani Rupees (PKR)
account_type	Type of account involved (Personal, Business, etc.)

customer_segment	Classification of the customer (Mass, Affluent, Private, etc.)
-------------------------	--

7.3 Scalability and Future Use

Although the current dataset represents only five days of sample data, the system has been built in a modular and scalable way. It can:

- Handle larger volumes of transactional data
- Accept real-time data feeds from banking systems
- Be extended to include more columns (e.g., channel, employee ID, transaction mode)
- Serve as a base for AI-driven decision support systems in actual banking environment

8. System Architecture

The architecture of the HBL Visualization Assistant is designed to bridge the gap between complex tabular financial data and user-friendly visual insights using modern AI APIs and Python tooling.

Overview:

- The system is built around modular components that handle query interpretation, AI-assisted chart generation, business insight generation, and frontend rendering.
- User queries are processed through logic that dynamically decides whether to generate a chart or a textual insight (or both).
- The backend leverages OpenRouter and Together AI for smart generation while Streamlit serves as the UI layer.

9. Module Breakdown

9.1 Frontend UI (Streamlit)

The user interface is built using Streamlit, which serves as the entry point for user interaction. Users can type natural language queries like:

“Show total deposits by branch in June”

Key Components:

- **Text Input Box:** Captures the user's query.
- **Custom Header UI:** Styled with HBL's brand color (#088F8F) using embedded HTML/CSS.
- **Chart Output Area:** Renders visualizations dynamically using matplotlib.
- **Insight Output Area:** Displays model-generated textual summaries or fallback insights.

Purpose:

- To create a clean, intuitive, and professional interface for business users who may not be tech-savvy but want insights quickly from structured deposit data.

9.2 Code Generation Module

This module handles the logic for generating a matplotlib chart using an LLM via the OpenRouter API.

Workflow:

1. The user query is inserted into a custom prompt.
2. The prompt instructs the LLM to generate only matplotlib code using df (the deposit DataFrame).
3. We use: model = "anthropic/claude-3-haiku"
4. The code is parsed using regex to extract valid Python inside triple backticks.
5. The code is executed using exec() with a pre-defined context of variables like df, plt, and pd.
6. Additional logic using matplotlib.ticker and ax.bar_label() formats the chart axis and adds numeric labels.

Error Handling:

- If the LLM returns invalid or empty code, a fallback message is shown, and insight generation is triggered instead.

9.3 Insight Generation Module

When charts cannot be produced or when the query is more qualitative, this module uses TogetherAI's LLaMA 3 model to summarize insights from an aggregated version of the dataset.

Steps:

1. Based on the query, we detect the grouping column (branch, city, customer segment, etc.).
2. The DataFrame is grouped and aggregated (sum of deposit_amount), then converted into a plain-text summary table.
3. A prompt is sent to TogetherAI with:
 - The user query
 - The aggregated table
 - Instructions to write a short, business-style insight using actual numbers only (no hallucinations).
4. The response is shown inside a green-styled `st.success()` box.

Purpose:

- To answer business questions in a natural, readable format even when charts aren't needed or possible.

9.4 Query Interpretation Logic

This is the intelligent fallback system that ensures every query leads to some meaningful output, even if the model fails.

How it works:

- Keywords like "branch", "region", "city", "customer segment", "account type", etc., are checked in the query using `if "branch" in query.lower()`, etc.
- Based on the match, we dynamically choose the grouping column for aggregation.
- This serves two roles:
 - For generating the summary table passed to the LLM for insight.

- For deciding what kind of chart to generate.
- If no grouping term is detected, we default to showing the first 15 rows of the data.

Benefits:

- Prevents user frustration due to unhandled queries.
- Guarantees fallback insights or visuals even when the LLM doesn't fully cooperate.

10. Implementation Details

This section dives into the under-the-hood logic of how each module operates together in the HBL Visualization Assistant app.

10.1 Streamlit Frontend Integration

The frontend was built using Streamlit, a Python-based rapid web framework. The layout includes:

- `st.set_page_config()` to define page title and centered layout.
- A styled HTML header using `st.markdown()` with embedded CSS to maintain branding and visual hierarchy.
- A `st.text_input()` widget to take user queries.
- A conditional if query: block to trigger the logic only when the user submits a question.

The frontend dynamically renders:

- A matplotlib chart using `st.pyplot()`
- Or, if chart code is unavailable, an LLM-generated business insight using `st.success()`.

10.2 Prompt Engineering for Code Generation

The user query is passed into a highly-structured prompt sent to OpenRouter's LLM (Claude 3 Haiku). The structure is:

```
chart_prompt = f"""
```

You are a Python assistant. Use matplotlib to create a chart.

You have a pandas DataFrame called df with the following columns:

```
[...]
```

User query: "{query}"

```
"""
```

Key Implementation Notes:

- The prompt explicitly tells the model to use only matplotlib and avoid libraries like seaborn or plotly.
- The prompt includes column names of the actual dataset, allowing the model to generate syntactically valid grouping and plotting code.
- The response is parsed using `re.search(r"```(?:python)?\n(?:.*)```", ...)` to extract only the Python code between triple backticks.

Execution:

```
exec_globals = {"df": df.copy(), "plt": plt, "pd": pd}
```

```
exec(code, exec_globals)
```

We use a clean global context to safely execute the generated code.

Post-processing:

- `ax.yaxis.set_major_formatter()` to format Y-axis in billions (B).
- `ax.bar_label()` is used to add data labels directly on top of bars.
- `plt.tight_layout()` prevents layout issues before rendering via `st.pyplot()`.

10.3 API Integration

Two separate API calls are used:

OpenRouter API for Chart Code

```
url = "https://openrouter.ai/api/v1/chat/completions"
```

```
headers = {"Authorization": f"Bearer {OPENROUTER_API_KEY}", ...}
```

- The model used is "anthropic/claude-3-haiku".
- The request uses the OpenRouter API's chat endpoint with proper headers and message formatting.

TogetherAI API for Insights

```
url = "https://api.together.xyz/v1/chat/completions"
```

```
headers = {"Authorization": f"Bearer {TOGETHER_API_KEY}", ...}
```

- The model used is "meta-llama/Llama-3-8b-chat-hf".
- The prompt includes:
 - A system message: "You are a helpful business analyst"
 - The user query
 - A plain text version of the aggregated table

Both APIs use `requests.post()` and include error handling via `try-except` to catch connectivity or parsing issues.

10.4 Query Interpretation and Column Mapping Logic

To ensure that even ambiguous or vague queries result in useful outputs, we implemented keyword-based logic:

if "branch" in query.lower():

group_column = "branch_name"

elif "region" in query.lower():

group_column = "region"

...

else:

group_column = None

Once a group column is identified:

- `df.groupby(group_column)["deposit_amount"].sum()` is executed.
- The result is converted to a string using `summary_df.to_string(index=False)` to feed into the LLM insight prompt.

If no keyword matches are found:

- The fallback is `df.head(15).to_string(index=False)` to give a preview of raw data.

10.5 Error Handling and Fallbacks

Throughout the application, we use `try-except` blocks:

- To catch and display errors when executing LLM-generated code.
- To fallback to textual insight generation if a chart fails or is invalid.
- To ensure the app never crashes, even if the API fails or the model returns nonsense.

Example:

except Exception as e:

st.error(f"Error running chart code: {e}")

10.6 Insights Generation Logic

When a chart is not available, we always attempt to generate a text-based summary. The prompt contains:

- The original user query
- A summary table as plain text
- Strict instructions like:
 - “Do not hallucinate values”
 - “Only use information in the table”
 - “Mention values in PKR rounded to billions or millions”

This ensures factual, business-style summaries are presented like:

- "The highest deposits were observed in Clifton (PKR 45.2M)"
- "Business accounts contributed 62% of total deposits during the period."

11. Testing and Validation

To ensure the reliability, accuracy, and practical value of the HBL Visualization Assistant, a rigorous and iterative testing process was carried out. This involved validating the complete pipeline—from user input to output (visualizations or insights)—as well as comparing different Large Language Models (LLMs) for code and insight generation.

11.1 Functional Workflow Testing

We tested the system end-to-end using a range of realistic business-style queries designed to simulate what HBL analysts or managers might ask.

Sample Queries Tested:

- “Show total deposits by branch in June”
- “Show a stacked area chart of daily deposits by customer segment”
- “Show a pie chart of total deposits by customer segment”
- “Compare personal vs business accounts over 5 days”
- “Show daily deposit trend for Clifton branch”
- “Show total deposits by account type”

What was validated:

- Correct identification of grouping columns like `branch_name`, `customer_segment`, and `account_type`.
- Accurate filtering of date ranges.
- Logical chart selection (bar, pie, area, line) based on the phrasing of the query.
- Fallback insights triggered if charts couldn’t be generated.

11.2 Language Model Testing & Comparison

A major focus was experimenting with different LLMs to test how well they could:

- Generate valid matplotlib chart code.
- Write accurate, data-grounded business insights from summaries.

Chart Code Generation Testing

Models Tested:

- claude-3-haiku (via OpenRouter)

- deepseek-coder (via OpenRouter)
- mistralai/Mistral-7B-Instruct (via Together AI)
- Gemini 1.5 Pro (via Google AI Studio)

Key Findings:

- Claude 3 Haiku gave the most consistent chart code with correct syntax, variable usage (df.groupby()), and clean formatting inside code blocks.
- DeepSeek occasionally hallucinated variable names or returned malformed code blocks.
- Mistral struggled with code correctness and often misunderstood the df context.

Final Choice: claude-3-haiku via OpenRouter for chart code generation.

Insight Generation Testing

Models Tested:

- meta-llama/Llama-3-8b-chat-hf (via Together AI)
- mistralai/Mistral-7B-Instruct (Together AI)
- Gemini 1.5 Pro (via Google AI Studio)

Key Findings:

- LLaMA 3 8B produced the most accurate and readable summaries, with proper numbers, percentages, and correct PKR formatting.
- Gemini's API wasn't working.
- Mistral was concise but weaker in formatting and tone.

Final Choice: meta-llama/Llama-3-8b-chat-hf via Together AI for insight generation.

11.3 Error Handling and Resilience

Robustness was validated by testing edge cases:

- Empty or malformed queries → Prompt shown to user
- Typos like “brnch” instead of “branch” → fallback triggered
- LLM returns invalid code → insight generation invoked automatically
- Missing chart type instructions → bar chart generated as default

All exceptions were caught using:

- try-except blocks around exec()
- Regex checks for valid code block structure

- Conditional rendering in Streamlit

11.4 UI/UX Testing

We ensured that the front-end interface was:

- Clear and intuitive for non-technical business users.
- Styled using HBL's theme color #088F8F.
- Responsive and clean, with distinct sections for:
 - Chart (matplotlib output)
 - Insights (model output)
 - Query input

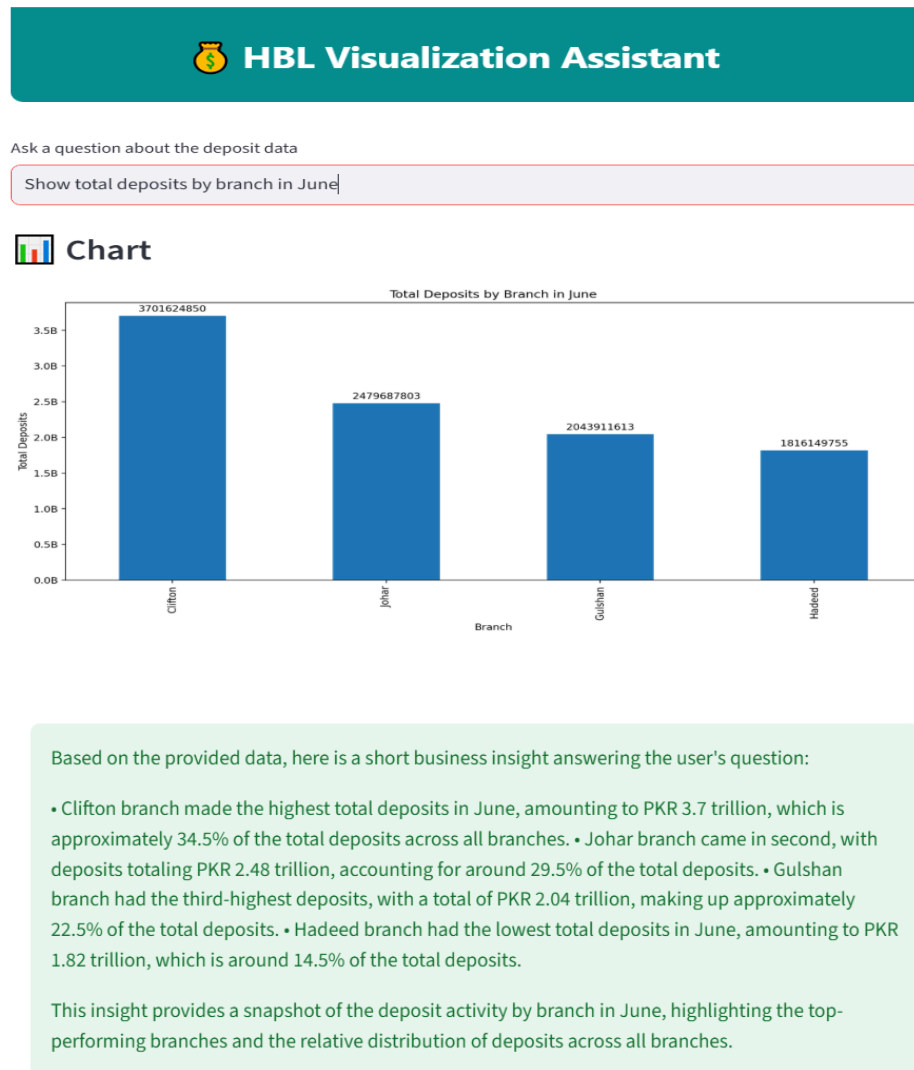
12. Results and Screenshots

The HBL Visualization Assistant successfully transformed natural language queries into meaningful business insights using dynamic visualizations and AI-generated summaries. The results demonstrate the robustness of the system in answering real-world questions about deposit data with minimal user effort.

12.1 Query-Based Results

Below are a few key sample queries along with the type of result generated and summary observations.

Query: “Show total deposits by branch in June”



Query: “Compare personal vs business accounts over 5 days”



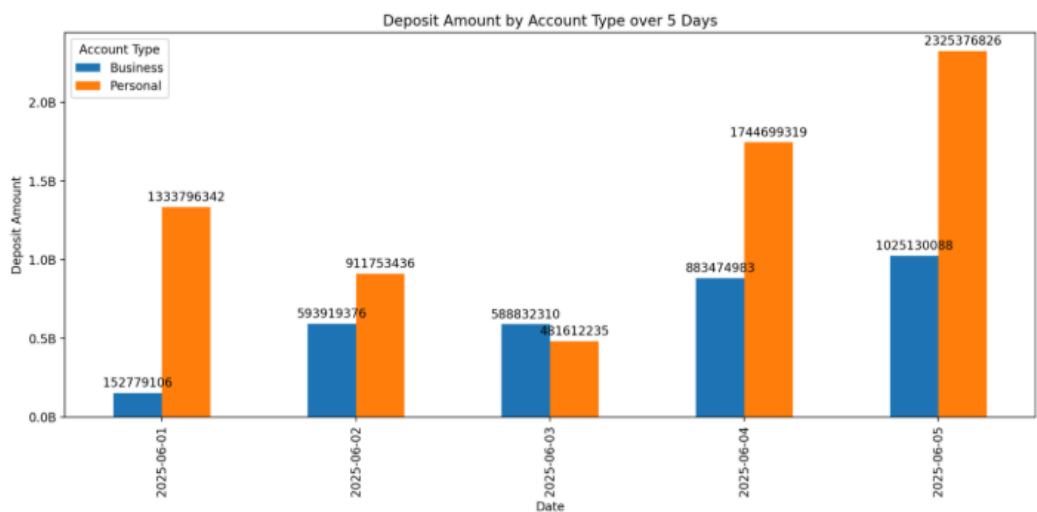
HBL Visualization Assistant

Ask a question about the deposit data

Compare personal vs business accounts over 5 days



Chart



Based on the provided data, here is a short business insight answering the user's question:

- Over the 5-day period, personal accounts received a total deposit of PKR 6,797 billion, which is a significant portion of the total deposits made during this time.
- In contrast, business accounts received a total deposit of PKR 3,245 billion, which is approximately 48% of the total deposits made in personal accounts.
- This suggests that personal accounts may be more lucrative for the institution, with a higher average deposit amount or a larger number of account holders contributing to the overall deposits.
- Overall, the data indicates that personal accounts have dominated the deposit activity over the 5-day period, with business accounts contributing a notable but smaller share of the total deposits.

Query: “Show a pie chart of total deposits by customer segment”



HBL Visualization Assistant

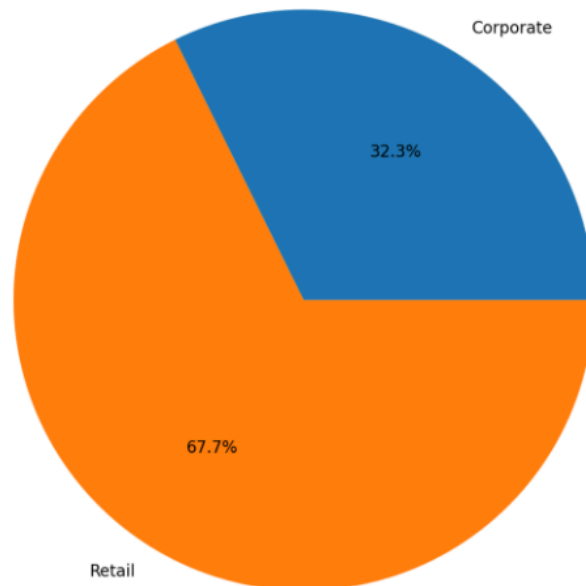
Ask a question about the deposit data

Show a pie chart of total deposits by customer segment



Chart

Total Deposits by Customer Segment



Based on the provided data, here is a short business insight:

- The Retail customer segment accounts for the majority of total deposits, with a deposit amount of PKR 6.798 billion, which is approximately 67% of the total deposits.
- The Corporate customer segment has a deposit amount of PKR 3.245 billion, which is approximately 33% of the total deposits.
- The overall total deposits amount to PKR 10.044 billion, with the Retail segment dominating the deposits landscape.
- The disparity in deposit amounts between the two customer segments is notable, with the Retail segment having a significant lead over the Corporate segment.

Query: “Show daily deposit trend for Clifton branch”



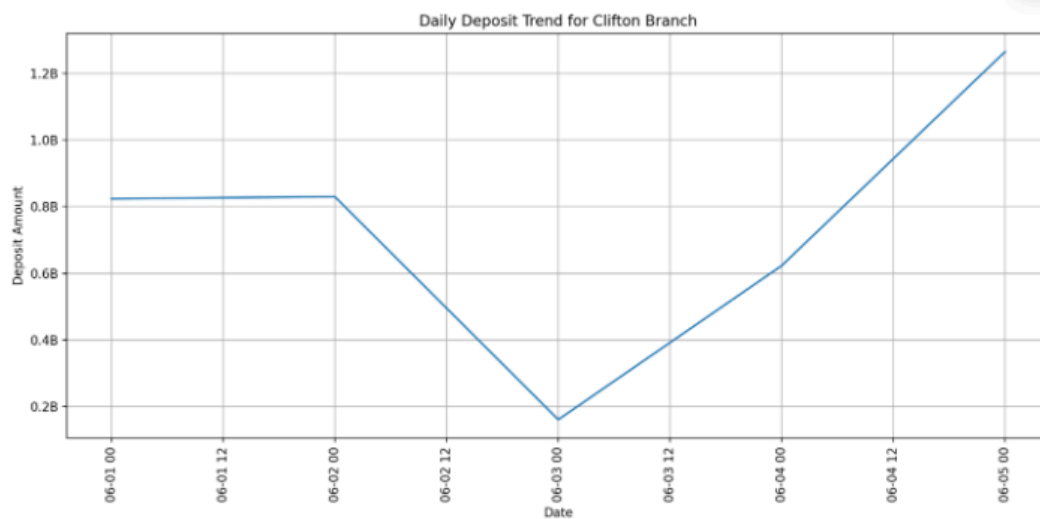
HBL Visualization Assistant

Ask a question about the deposit data

Show daily deposit trend for Clifton branch



Chart



Based on the daily deposit trend for the Clifton branch, here are some key insights:

- The Clifton branch has received the highest daily deposit amount, with a total of PKR 3,701,624,850.
- This represents approximately 27% of the total daily deposits across all branches ($\text{PKR } 3,701,624,850 / \text{PKR } 13,947,627,116 = 27\%$).
- The Clifton branch has received significantly more deposits than the next closest branch, Johar, which has deposits of PKR 2,479,687,803.
- The Clifton branch's deposits are nearly double that of Gulshan and Hadeed, which have deposits of PKR 2,043,911,613 and PKR 1,816,149,755, respectively.

Query: “Show a stacked area chart of daily deposits by customer segment”



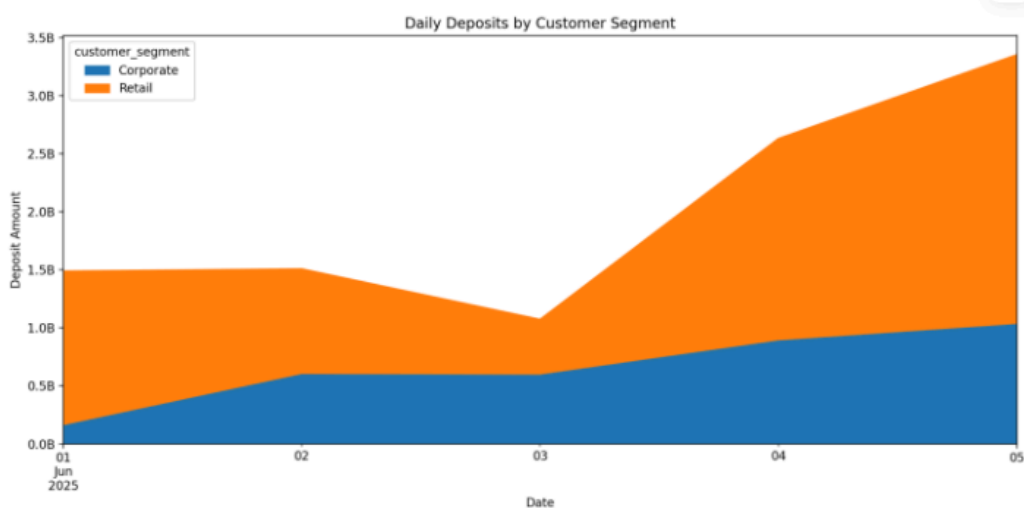
HBL Visualization Assistant

Ask a question about the deposit data

Show a stacked area chart of daily deposits by customer segment



Chart



Based on the aggregated table, here are the business insights:

- The Retail customer segment contributes the majority of daily deposits, with a total of PKR 6,797,238,158. • This is equivalent to approximately 81% of the total daily deposits, with the remaining 19% coming from the Corporate segment. • The Corporate customer segment has a significant daily deposit amount of PKR 3,244,358,863, which is substantial but significantly lower than the Retail segment's deposits. • Overall, the data suggests that the Retail segment is the primary driver of daily deposits, with a substantial lead over the Corporate segment.

13. Challenges Faced

Developing the HBL Visualization Assistant came with several technical and strategic challenges that had to be addressed iteratively throughout the development lifecycle:

13.1 Finding Suitable LLM APIs

As this was a small-scale, student-led project, access to premium large language models (LLMs) was limited due to budget constraints. Many state-of-the-art models are only available via paid APIs (e.g., GPT-4, Gemini Advanced, Claude 3 Opus), which limited our ability to consistently generate accurate and context-aware responses. After several trials, we settled on:

- Claude 3 Haiku (OpenRouter) for chart code generation
- LLaMA 3 (Together AI) for insight generation

This dual-LLM strategy provided a good balance of functionality and cost-effectiveness, but performance could still be improved by using more advanced (paid) models.

13.2 Insight Accuracy & Hallucination

One key challenge was that insight summaries were sometimes inaccurate or slightly misaligned with the actual data table. This is due to the limited reasoning and context-tracking ability of free LLMs. Paid models like Claude 3 Opus or GPT-4 Turbo could potentially provide more factual insights.

13.3 Prompt Engineering

Crafting effective prompts that guided the LLMs to generate consistent and valid outputs (especially for chart code) required extensive trial and error. Examples include:

- Getting the LLM to use only matplotlib and df without relying on seaborn or plotly.
- Ensuring summary insights are data-grounded and do not hallucinate.
- Cleaning model outputs using regex to extract only valid code blocks.

13.4 Model Segmentation Strategy

Initially, we tried using a single LLM to perform both insight generation and chart generation, but the results were inconsistent. Splitting the logic and using two specialized LLMs significantly improved performance, but also increased development complexity and API management overhead.

13.5 Small Dataset & Testing Constraints

Due to time constraints and the scope of the project, we worked with a limited dummy dataset of approximately 2,000 rows spanning just 5 days (June 1–5). While this allowed for fast iteration and testing, it did not fully demonstrate the scalability or robustness of the system.

However, the system is designed to work with larger datasets and can easily be extended to include transaction-level data, monthly summaries etc.

14. Conclusion & Future Work

14.1 Conclusion

The HBL Visualization Assistant serves as a proof-of-concept for a modern, AI-enhanced business intelligence tool tailored for financial data. It enables:

- Seamless transformation of natural language queries into meaningful visualizations and summaries.
- Rapid insights from structured deposit datasets without requiring SQL or BI tool expertise.
- A user-friendly interface that encourages self-service analytics for business users.

Despite working on a limited dataset, the assistant effectively handled diverse queries related to branches, account types, customer segments, and trends.

14.2 Future Work

There are multiple avenues to expand and improve this project:

Integration with Power BI APIs

Instead of relying solely on Python visualizations, we can integrate Power BI's REST API to:

- Generate interactive dashboards
- Trigger reports dynamically based on text queries
- Combine real-time banking data with pre-built templates

LLM Upgrades (Paid Models)

To improve the factual correctness and richness of insights:

- Use GPT-4 Turbo, GPT-3.5 Turbo, Claude 3 Opus, or Gemini 1.5 Pro
- Implement ensemble reasoning or validation chains

Fine-Tuning Models for Banking Domain

Custom-trained LLMs fine-tuned on financial datasets could:

- Understand specific banking terminology

- Provide more accurate insights aligned with real-world KPIs

Larger-Scale Dataset Integration

The assistant should be tested on:

- Monthly or quarterly deposit data
- Transaction-level data across all regions
- Multi-source integrations (e.g., customer CRM, complaints, revenue, churn)

Voice-to-Query and Mobile Support

Adding voice recognition would allow managers to ask questions verbally and receive insights on the go, particularly useful for mobile banking dashboards or executive tools.