LLM Report

This report outlines the development of an Information Retrieval (IR) pipeline designed to handle Machine Learning (ML) related queries. The system retrieves and summarizes relevant information based on user queries, using data scraped from books and various websites like **javatpoint, geeks for geek, builtin,** etc.. A web portal allows users to submit queries, view retrieved results, and provide feedback. The pipeline also integrates a generative AI model, LLaMA 3.1 70B (as part 2), for creating summaries from the retrieved content, enhancing the user experience with more concise and relevant answers.

Objective:
To develop an IR pipeline that retrieves relevant ML information based on user queries and generates summaries using a generative AI model. Current search engines often lack the ability to provide context-specific answers. We are trying to address this challenge by building an efficient IR system focused on ML specialised queries.

IR systems have advanced significantly, especially in the context of NLP and Machine Learning. Traditional IR methods focused on keyword matching, while modern systems, like the one developed here, utilize embeddings and similarity search to improve relevance. The integration of generative AI models for summarization includes an innovative layer to this IR process.

Github : Link

Methodology (How we worked)

1. **Data Collection:**

   Data was gathered through web scraping from various sources, including books and ML-related websites as mentioned above. The collected data was vast and diverse, ensuring that the system would cover a broad range of topics relevant to ML.

2. **Data Cleaning:**

   The collected data was thoroughly cleaned to remove irrelevant text, duplicates, and noise. Techniques like **tokenization**, **stop-word removal**, and **stemming** were applied to prepare the data for further processing using NLTK and RE.

3. **Embedding Creation:**

Using sentence transformers, embeddings were created for each piece of text. These embeddings capture the **semantic** meaning of the content, allowing the system to retrieve information that is contextually similar to user queries.

4. **Database Storage:**

   The embeddings were stored in a PostgreSQL database, for efficient querying. In this way, the system could quickly retrieve relevant information based on similarity to the user query.

5. **QueryTransformation and Handling:**

   When a user submits a query, it is rewritten for clarity, converted into an embedding, and matched against stored embeddings. The system retrieves the most relevant results with a similarity percentage to indicate relevance.

6. **Prompt Tuning:**

   The chatbot's responses were optimized by placing the most important data chunks at the **start and end** of the prompt. If the system couldn't find an answer in the stored data, it generated a response using its internal knowledge.

7. **User Feedback and Analysis:**

   Users can interact with the retrieved information by liking, disliking, or providing textual feedback. This feedback is used to improve the accuracy of the system over time, with an admin dashboard displaying graphs to monitor system performance and identify patterns in user interactions.

Web Portal Design

The web portal provides simple and user-friendly design that makes it easy to use for both users and administrators.

1. **User Features**:
   - Users can interact with search results through **copy**, **like**, **dislike**, and **feedback** options, enhancing engagement.
   - An **audio feature** allows users to listen to the responses, making the portal accessible to diverse audiences.
   - A **chat history section** on the left side tracks previous queries, enabling users to revisit earlier searches effortlessly.
2. **Admin Dashboard**:

- The admin interface includes **graphical tools and analytics** to evaluate system performance.
- Key data like query patterns, user ratings, and feedback trends are shown in charts to help make decisions and improve the system.

BONUS Part

Retrieval Augmented Generation (RAG) with LLaMA 3.1 70B

The integration of the **LLaMA 3.1 70B model** (Groq api) adds a generative AI layer to the retrieval process, creating a comprehensive **RAG** pipeline.

- **Enhanced Summarization**:
  - After retrieving the most relevant documents, the model generates **summaries** designed to user queries.
  - In any case, if we didn't get the similar answer then we are rephrasing the query by Llama and resending it again.

This feature combines traditional search methods with advanced AI to provide relevant results. It shows how RAG can offer accurate, easy-to-use, and specialized information.

Results and Evaluation

The system was thoroughly tested with various ML-related queries, focusing on the following key aspects for evaluation:

- **Relevance**: The system was evaluated based on its ability to retrieve documents that closely match user queries, ensuring both accuracy and precision.
- **User Feedback**: User-provided like/dislike and comments were analyzed to understand the **quality and effectiveness** of the results.
- **Summarization Quality**: The summaries generated by the LLaMA model were reviewed for **clarity, conciseness, and relevance** in addressing user needs.
- **Query Transformation:** This approach simplified complex or mixed questions, making them easier for the chatbot to process. In this way, improved the accuracy of the chatbot's responses.
- **Prompt Tuning:** Presenting data carefully at the start and end of prompts improved the chatbot's answers, ensuring it focused on the most relevant information.

Analysis

1. **Cluster Density**

- Clusters with closely grouped data show that users in those groups have similar questions or needs. Spread-out clusters mean users' questions are very different, so the chatbot needs improvement in handling them.

2. **Distance Relationship**

- Large distances between clusters show distinct topics, while small distances highlight overlaps. Overlapping clusters may indicate the need for better categorization or differentiation in chatbot understanding.

3. **User Feedback Sentiment Trend**

- Sentiment analysis shows how users feel about the chatbot over time. Positive feedback highlights strengths, while negative feedback points to areas that need improvement.

4. **Generative AI Impact:**

- Integrating LLaMA improved user experience by providing concise and summarized answers. However, the quality of responses is highly dependent on the clarity of user queries and input context.

Challenges we faced

Handling Diverse Queries:

- Queries with ambiguous, incomplete, or overly specific phrasing required advanced embedding techniques for improved matching that we have tackled with LLama to rephrase it for now.

When users ask multiple question in one ask:

- We used api to differentiate and asked in multiple go to retrieve the correct answer.

Embedding Management:

- Generating and storing high-dimensional embeddings in the database required careful optimization to balance storage and retrieval performance.

User Feedback Interpretation:

- Processing qualitative user feedback (e.g., textual comments) to derive actionable insights was a challenging task.

Security and Authentication:

- Ensuring secure handling of user data and preventing unauthorized access required robust session management and password hashing.

Prompt Tuning:

If we get the order of data wrong, the chatbot might not give the best answers. For now, we have given a prompt to Llama to correct it.

Reranking:

We tried but weren't able to apply due to lack of proper time. Relying on specific tools like Cohere Reranker might limit flexibility for other systems.

Real-Time Performance::

Generating embeddings and performing similarity searches in real-time is computationally hard, impacting response time under heavy loads.

Future Work:

Reranking

After the system retrieved initial results:

a. **Reranking** was used to reorder the chunks based on their relevance to the query.
b. Tools like the **Cohere Reranker** ensured that the most relevant answers were presented first.

Advanced Feedback Loop:

- **Dynamic Learning**: Automate feedback integration, enabling the system to adapt dynamically to user preferences and refine retrieval accuracy in real time.

Enhanced Summarization

- **Generative Model Exploration**: Experiment with alternative generative AI models like GPT-based systems or custom fine-tuned LLMs for clearer and more contextual summaries.
- **Segmented Summarization**: Develop methods to split responses from long documents for better comprehensibility.

Real-Time Processing:

- **Parallelization**: Implement parallel processing or embedding caching mechanisms to accelerate query handling and improve response times.

Interactive Features:

- **Custom Query Options**: Introduce filters (e.g., by date, topic, source) for personalized query refinement, enhancing user control and precision.

Security Enhancements:

- **Two-Factor Authentication**: Strengthen security by adding robust authentication layers, including two-factor methods for user accounts.

Improved Visualization:

- **Advanced Tools**: Utilize modern visualization libraries for interactive, intuitive representations of embeddings and clusters.

Exploring Alternative Reranking Tools

- Experiment with other reranking tools to evaluate their impact on result accuracy and relevance.
- This will help identify better options for improving the system's performance further.

Automating Data Cleaning and Chunking

- Implement automated methods for data cleaning and chunking to reduce manual effort and save time.
- This will ensure faster processing and consistency across datasets.

Tools Used

- **Flask:** Used for backend development, including route management, session handling, and integrating the frontend with the server-side logic.
- **Sentence-Transformers:** Generated semantic embeddings for textual data to enable similarity-based search and efficient retrieval.
- **PostgreSQL:** Served as the database for storing user data, embeddings, and feedback, integrated with the pgvector extension for vector operations.
- **Pgvector:** Enables efficient storage and similarity search of embeddings directly within the PostgreSQL database.

- **Scikit-learn:** Implemented machine learning algorithms like KMeans for clustering, PCA for dimensionality reduction, and Naive Bayes for query classification.
- **Matplotlib:** Created visualizations for embedding clusters, user activities, and sentiment trends to analyze and present data effectively.
- **Pandas:** Handled CSV data uploads, preprocessing, and manipulation for embedding generation and database storage.
- **VADER Sentiment Analyzer:** Performed sentiment analysis on user feedback to measure satisfaction and identify trends.
- **Cohere Reranker:** Helped rank the answers by relevance.
- **Fine-Tuned Prompts:** Improved how the chatbot handled input and gave better responses.
- **LLaMA 3.1 70B:** Integrated as a generative AI model to provide concise summaries for user queries, enabling advanced Retrieval-Augmented Generation (RAG).
- **Werkzeug:** Used for password hashing to securely store user credentials and enhance authentication mechanisms.
- **HTML, CSS, and JavaScript:** Built the user-friendly frontend interface for query input, displaying results, collecting feedback, and showcasing query history.
- **APIs:** Integrated external APIs to connect the system with the generative AI model for advanced response generation.

Learnings

- **Full-Stack Web Development:** Built a functional web application integrating backend and frontend features using Flask.
- **NLP Techniques:** Utilized embeddings and similarity search to enhance information retrieval performance.
- **Database Optimization:** Learned efficient handling of embeddings and large datasets using advanced database extensions like pgvector.
- **User Feedback Integration:** Integrated user feedback for system improvement and used sentiment analysis to measure satisfaction.
- **Generative AI Integration:** Connected the IR pipeline with a generative model for advanced Retrieval-Augmented Generation (RAG) capabilities.
- **Data Clustering and Visualization:** Implemented clustering algorithms and PCA for meaningful data organization and insights.
- **IR Techniques:** Explored embedding generation and vectorized similarity search for accurate and efficient retrieval.
- **Machine Learning Applications:** Leveraged clustering, dimensionality reduction, and classification techniques to enhance functionality.

- **Analytics and Visualization:** Developed insightful graphs and metrics for data-driven decision-making.
- **Feedback and Sentiment Analysis:** Incorporated user feedback to iteratively improve performance and applied sentiment analysis for better system evaluation.

Team Members & Contribution:

Jitendra Kumar

Contributions:

- Data Scraping and Cleaning:
    - Scraped data from diverse sources such as pen source book, Javatpoint, GeeksforGeeks, BuiltIn, and two ML-related books.
    - Applied data cleaning techniques, including tokenization, stop-word removal, and stemming, to prepare the dataset for embedding generation.
- Feedback Analysis:
    - Conducted sentiment analysis on user feedback using the VADER Sentiment Analyzer to gauge user satisfaction.
    - Created visualizations of feedback trends and sentiment distribution for the admin dashboard using Matplotlib.
- LLaMA API Integration:
    - Integrated the LLaMA 3.1 70B generative AI model into the system, enabling advanced Retrieval-Augmented Generation (RAG).
    - Fine-tuned prompts to enhance the summarization quality for user queries.

Techniques Used:

- Web scraping tools (e.g., BeautifulSoup, Selenium).
- Preprocessing methods like tokenization and stemming.
- Sentiment analysis with VADER.
- Data visualization with Matplotlib.
- Generative AI integration and prompt engineering.

Nikhil Yadav

Contributions:

- Chatbot Features:
    - Designed and implemented chat history functionality, enabling users to view and delete past queries.
    - Built a feedback system where users can like, dislike, or provide textual feedback for chatbot responses.
    - Added additional user-friendly features, such as a text-to-speech option and copy functionality for chat responses.
- Admin Dashboard:
    - Enabled admins to input new data into the LLaMA database, automatically generating embeddings and performing clustering using KMeans and PCA.
    - Created user feedback analytics, including graphs and trends displayed on the admin dashboard.
    - Supported the system's scalability by building tools for future dynamic user management.
- Book Scraping and Cleaning:
    - Scraped content from books and cleaned it paragraph-wise for embedding generation.

Techniques Used:

- Full-stack development with Flask (backend) and HTML, CSS, JavaScript (frontend).
- Clustering and dimensionality reduction with KMeans and PCA.
- Embedding generation with Sentence Transformers.
- Text-to-speech integration for accessibility.

Nikhil Raj Soni

Contributions:

- Authentication and User Alignment:
    - Implemented robust authentication mechanisms using Werkzeug for password hashing.
    - Aligned user-specific chat history with their accounts, ensuring personalized interaction and enabling feedback tracking.
- Data Scraping and Cleaning for Websites:

    - Cleaned and preprocessed website data for embedding generation, ensuring that irrelevant or duplicate content was removed.
    - Implemented and tried different techniques to improve retrieval answers.

Techniques Used:

- Authentication and session management with Flask and Werkzeug.
- Data preprocessing techniques like duplicate removal and semantic filtering.
- Integration of chat history storage with the PostgreSQL database.

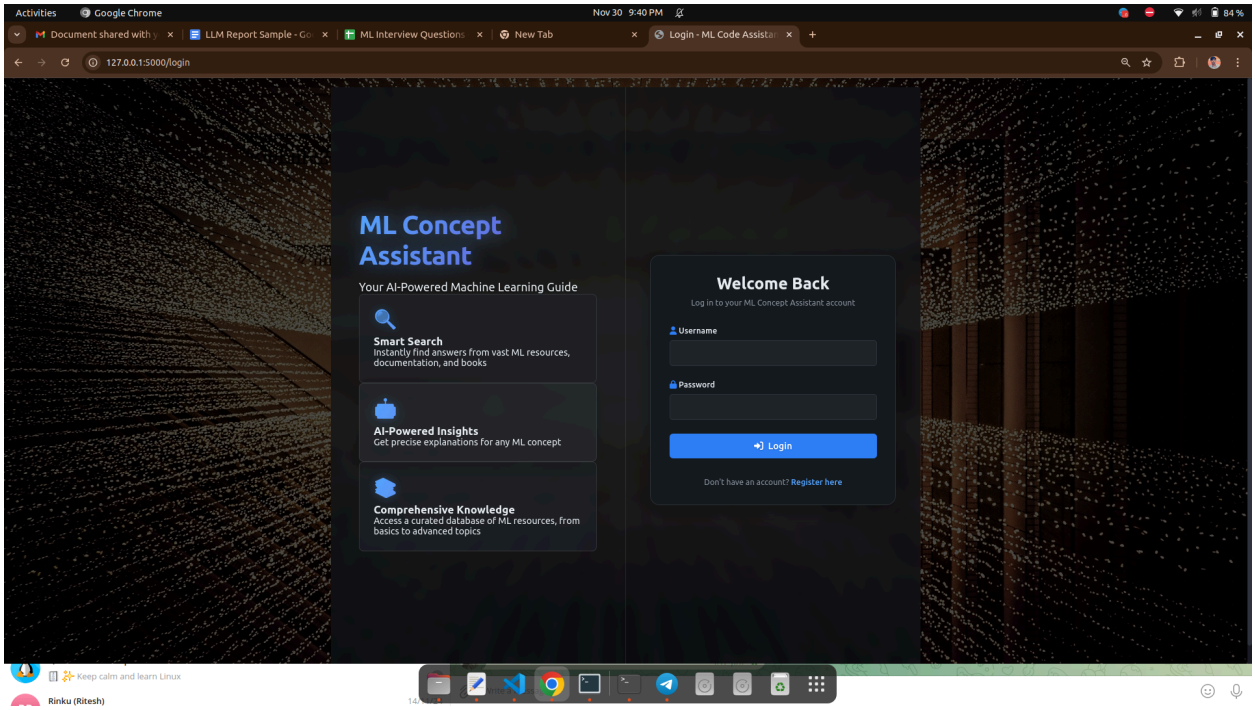
Rashmi Kumari

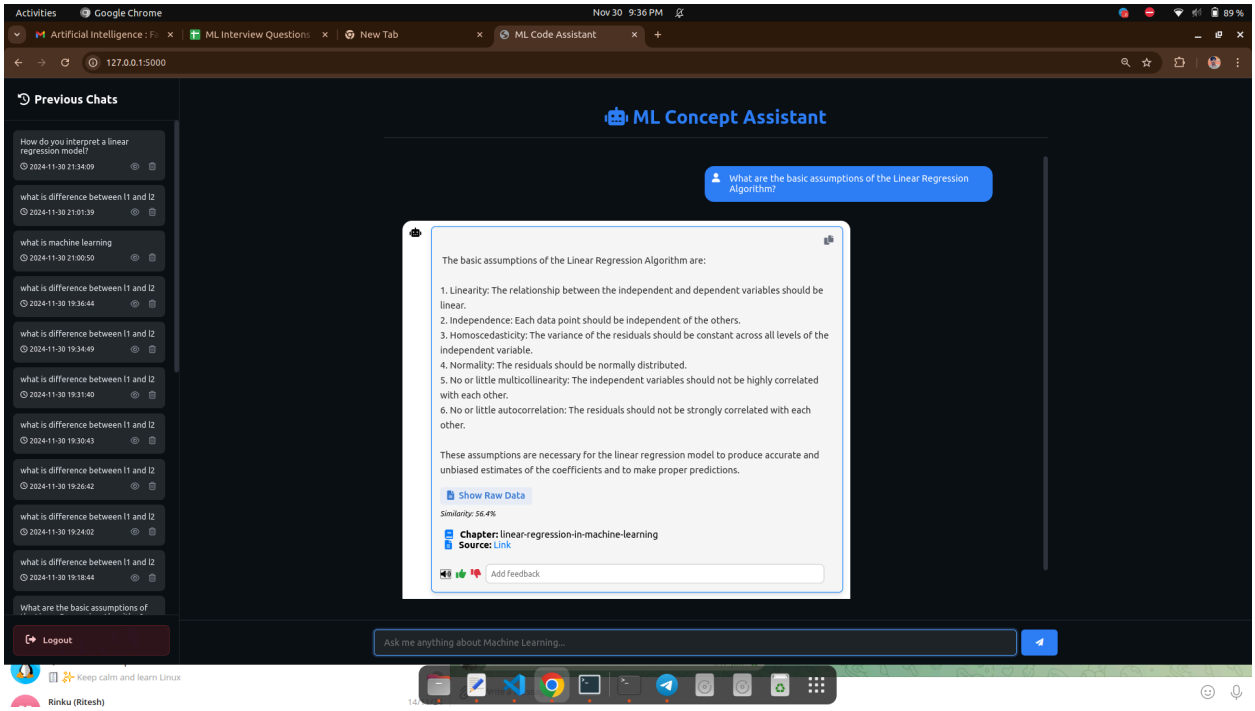
Contributions:

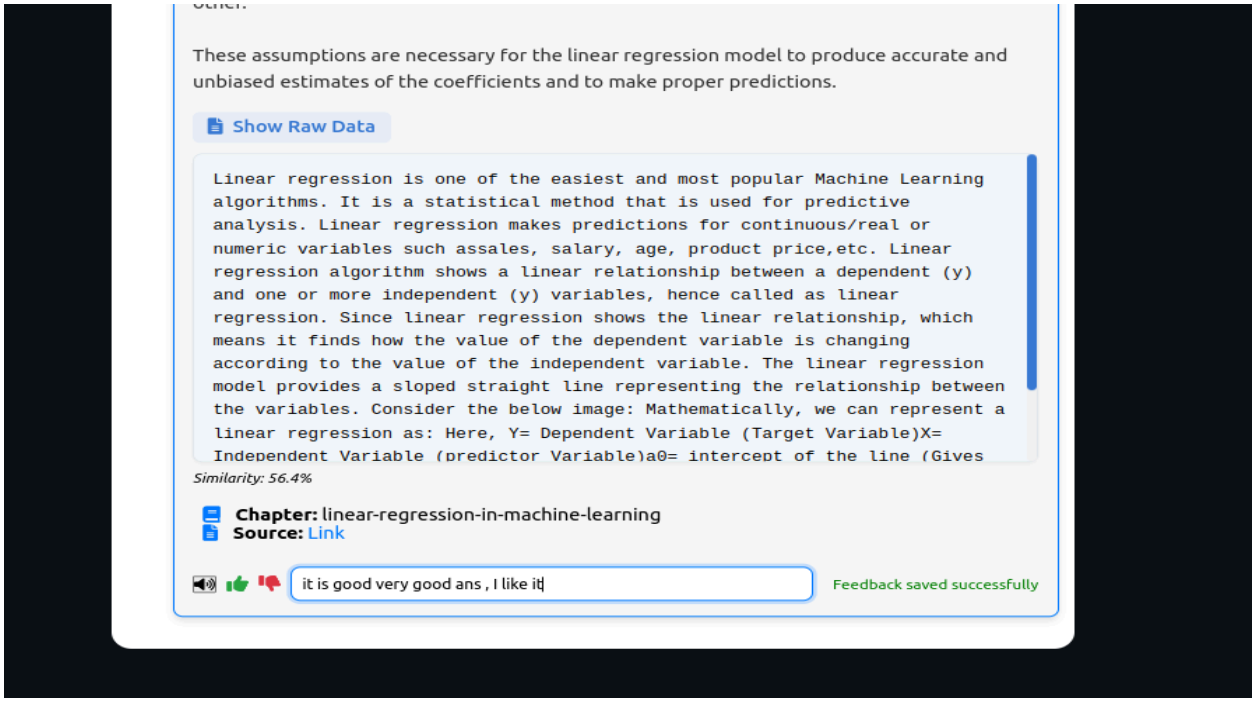
- Book Data Cleaning:
    - Supported the preprocessing of scraped book content by cleaning it paragraph-wise to make it ready for embedding generation.
- Data Scraping:
    - Scraped and contributed data from one ML-related book, ensuring diversity in the dataset.
    - Helped in integration of backend to frontend code.

Techniques Used:

- Web scraping and content extraction from books.
- Data cleaning processes such as noise removal and semantic segmentation.

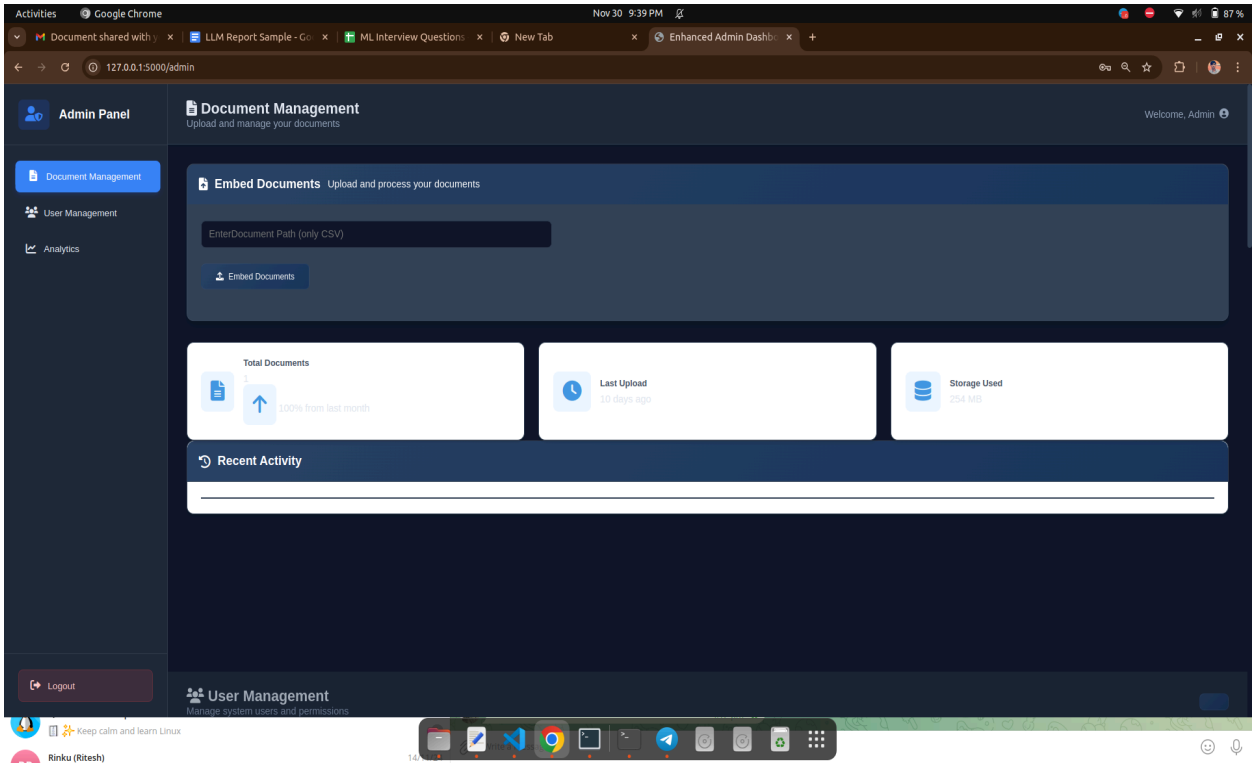

Activities
Google Chrome
Nov 30 9:40 PM
84 %
Document shared with
LLM Report Sample - Go
ML Interview Questions
New Tab
Login - ML Code Assistan
127.0.0.1:5000/login
ML Concept Assistant
Your AI-Powered Machine Learning Guide
Smart Search
Instantly find answers from vast ML resources, documentation, and books
AI-Powered Insights
Get precise explanations for any ML concept
Comprehensive Knowledge
Access a curated database of ML resources, from basics to advanced topics
Welcome Back
Log in to your ML Concept Assistant account
Username
Password
Login
Don't have an account? Register here
Keep calm and learn Linux
Rinku (Ritesh)


Activities
Google Chrome
Nov 30 9:36 PM
89 %
Artificial Intelligence : Fo
ML Interview Questions
New Tab
ML Code Assistant
127.0.0.1:5000
Previous Chats
How do you interpret a linear regression model?
2024-11-30 21:34:09
what is difference between l1 and l2
2024-11-30 21:01:39
what is machine learning
2024-11-30 21:00:50
what is difference between l1 and l2
2024-11-30 19:36:44
what is difference between l1 and l2
2024-11-30 19:34:49
what is difference between l1 and l2
2024-11-30 19:31:40
what is difference between l1 and l2
2024-11-30 19:30:43
what is difference between l1 and l2
2024-11-30 19:26:42
what is difference between l1 and l2
2024-11-30 19:24:02
what is difference between l1 and l2
2024-11-30 19:18:44
What are the basic assumptions of
Logout
ML Concept Assistant
What are the basic assumptions of the Linear Regression Algorithm?
The basic assumptions of the Linear Regression Algorithm are:
1. Linearity: The relationship between the independent and dependent variables should be linear.
2. Independence: Each data point should be independent of the others.
3. Homoscedasticity: The variance of the residuals should be constant across all levels of the independent variable.
4. Normality: The residuals should be normally distributed.
5. No or little multicollinearity: The independent variables should not be highly correlated with each other.
6. No or little autocorrelation: The residuals should not be strongly correlated with each other.
These assumptions are necessary for the linear regression model to produce accurate and unbiased estimates of the coefficients and to make proper predictions.
Show Raw Data
Similarity: 56.4%
Chapter: linear-regression-in-machine-learning
Source: Link
Add feedback
Ask me anything about Machine Learning...
Keep calm and learn Linux
Rinku (Ritesh)

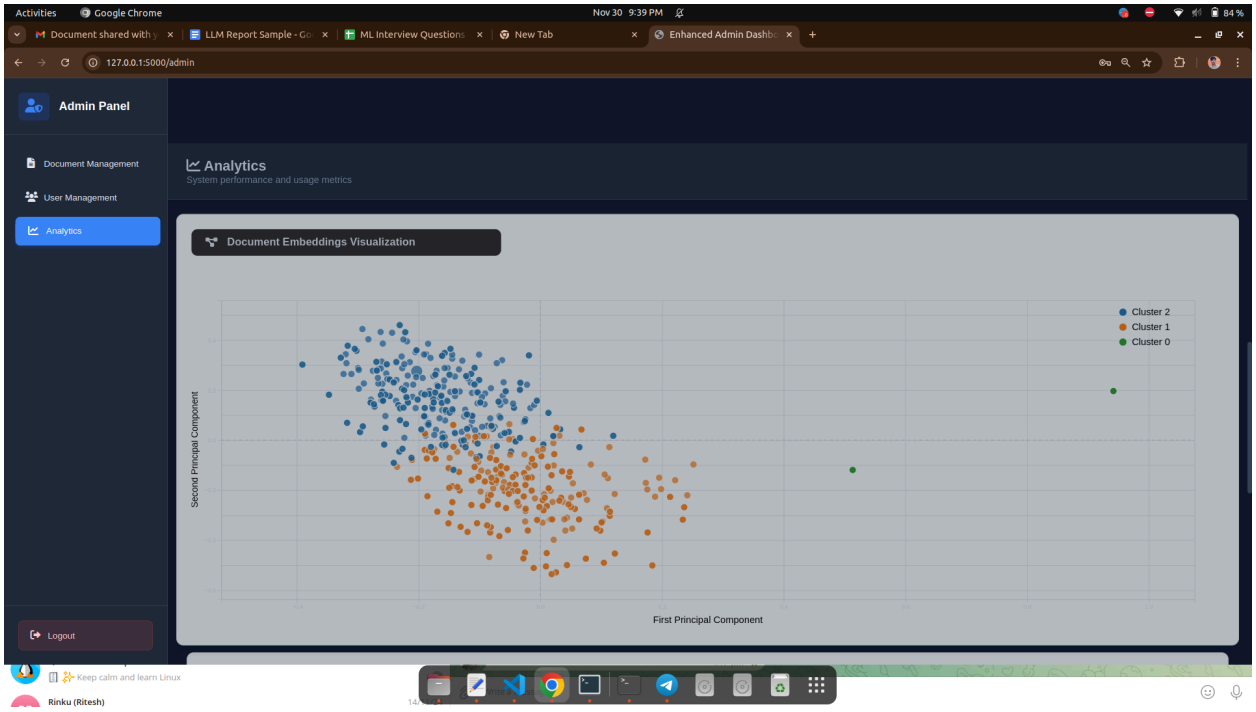


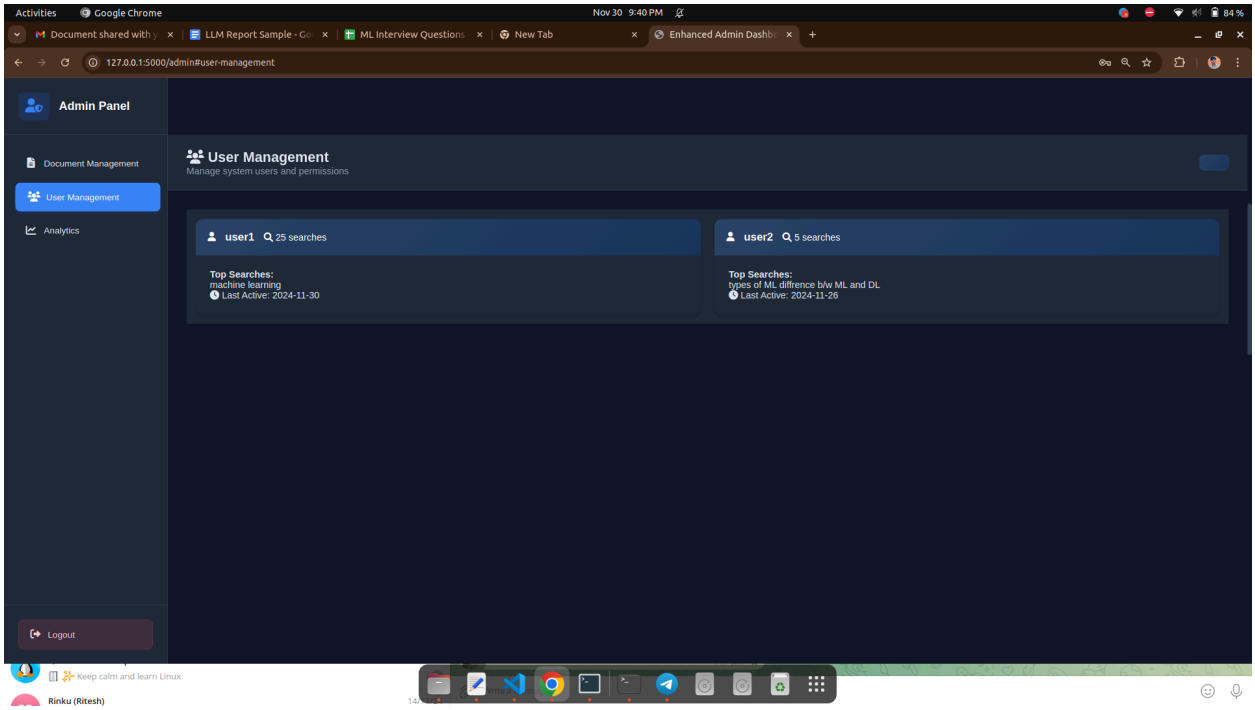
After viewing raw data and for feedback submission

2.) Admin Dashboard



Note: we've also integrated ML analysis part in frontend in admin desk as shown in following figs.




Activities
Google Chrome
Nov 30 9:40 PM
84 %
Document shared with
LLM Report Sample - Go
ML Interview Questions
New Tab
Enhanced Admin Dashbo
127.0.0.1:5000/admin#user-management
Admin Panel
Document Management
User Management
Analytics
Logout
User Management
Manage system users and permissions
user1
25 searches
Top Searches:
machine learning
Last Active: 2024-11-30
user2
5 searches
Top Searches:
types of ML diffrence b/w ML and DL
Last Active: 2024-11-26
Keep calm and learn Linux
Rinku (Ritesh)