AI Lead Scoring Dashboard - Project Report

Name: Pratham Vyas
LinkedIn: https://www.linkedin.com/in/prathamvyas3183/
GitHub: https://github.com/PRATO3183/Cleardeal_Internship_Assignment_II
Live App URL: https://lead-generation-project.netlify.app/

1. Solution Overview

This project successfully implements a full-stack AI Lead Scoring Dashboard designed to help sales brokers at ClearDeals prioritize high-intent leads. The application provides an "Intent Score" from 0 to 100, calculated by a machine learning model and refined by a rule-based re-ranker.

The solution consists of a responsive frontend for lead data submission, a table for viewing scored leads in real-time, and a robust FastAPI backend. The core objective—to create a functional prototype that demonstrates the lead scoring engine's logic and potential—has been fully met.

2. Architecture

The application is built on a modern, decoupled three-tier architecture to ensure scalability and maintainability.

- **Frontend (Client on Netlify):** A static single-page application built with vanilla HTML, CSS, and JavaScript. It is responsible for rendering the UI, capturing user input, and communicating with the backend via asynchronous API calls.
- **Backend (Server on Render):** A Python API built with the FastAPI framework. It exposes RESTful endpoints, validates incoming data using Pydantic, and orchestrates the entire scoring process.
- **Model & Logic Layer:** A pre-trained GradientBoostingClassifier model (.pkl file) is loaded into the FastAPI application at startup. A dedicated ScoringService encapsulates all business logic, including making predictions with the model and applying the re-ranking rules.

This separation of concerns allows each part of the application to be developed, tested, and deployed independently.

3. Machine Learning Model & Dataset

Model Justification: GradientBoostingClassifier

I chose Scikit-learn's GradientBoostingClassifier for several key reasons:

1. **High Performance:** Gradient Boosting models are powerful ensemble methods that consistently deliver high accuracy on structured, tabular data like the features in this dataset.
2. **Efficiency:** The model is fast to train and, more importantly, provides low-latency predictions, which is crucial for meeting the <300ms API response time requirement.
3. **Interpretability:** While more complex than simpler models, it still allows for the analysis of feature importances, providing insights into which factors are most predictive of lead intent.

Dataset Justification: Synthetic Data

A synthetic dataset of 10,000 leads was programmatically generated. This approach was chosen over sourcing a public dataset because it offered several advantages:

1. **Control & Relevance:** It allowed for the creation of features directly relevant to the problem scope (e.g., CreditScore, Income, TimeOnPage).
2. **Pattern Creation:** I could embed meaningful relationships between the features and the target variable (High_Intent). For example, the logic ensures that leads with higher income and credit scores have a higher probability of being high-intent, making the model's task realistic.
3. **Data Quality:** Generating the data myself ensured it was clean, well-formatted, and free of missing values, allowing me to focus on modeling rather than extensive cleaning.

4. LLM-Inspired Re-ranker

To incorporate the valuable context from the unstructured Comments field without the complexity and cost of a true LLM, a rule-based re-ranker was implemented. This system simulates the behavior of an LLM by parsing comments for specific keywords.

Re-ranker Logic:

- The system scans comments for predefined keywords that indicate high or low intent.
- **Positive Keywords:** Words like "urgent", "immediate", and "ready to buy" increase the initial ML score by a set amount (e.g., +20, +25).
- **Negative Keywords:** Words like "not interested", "spam", and "just browsing" decrease the score significantly (e.g., -30, -50).
- The final score is capped between 0 and 100 to maintain a consistent output range.

This pragmatic approach provides a fast, deterministic, and effective way to leverage

text data, directly addressing the project's unique requirements.

5. Compliance & Data Quality

Compliance Measure (DPDP-Ready):
To align with data privacy principles like those in India's DPDP Act, the application implements explicit user consent. The frontend form includes a mandatory "I consent to data processing" checkbox. This checkbox must be ticked for the form to be submitted, ensuring that data is only processed after clear and affirmative consent is given by the user.

Data Quality Challenge & Mitigation:

- **Challenge:** A common data quality issue is inconsistent or invalid categorical data entry (e.g., typos like "Singel" instead of "Single").
- **Mitigation:** This challenge was proactively mitigated on the frontend. Instead of using free-text fields for categorical features like AgeGroup and FamilyBackground, the UI uses HTML <select> dropdowns. This forces users to choose from a predefined, valid list of options, ensuring the data sent to the backend is always clean, standardized, and ready for the model.

6. Metrics & Implementation Challenges

Key Success Metrics:

- **Technical Metric: Precision.** For this use case, precision is the most important classification metric. It measures the proportion of "high-intent" predictions that were actually correct. A high precision score ensures that the time sales brokers invest in prioritized leads is not wasted on false positives.
- **Business Metric: Lead Conversion Lift.** The ultimate business goal is to increase the rate at which leads convert to sales. Success would be measured by comparing the conversion rate of leads prioritized by this system against a baseline. The target is to achieve a 2-3x lift in conversions for the prioritized group.

Implementation Challenge & Solution:

- **Challenge:** The most significant technical challenge encountered was resolving **CORS (Cross-Origin Resource Sharing) errors**. By default, web browsers block requests from a frontend hosted on one domain (e.g., Netlify) to a backend on another (e.g., Render) for security reasons.
- **Solution:** I resolved this by implementing and correctly configuring the CORSMiddleware in the FastAPI application. I explicitly added the live Netlify frontend URL to the list of allowed origins, which instructs the backend server to trust and respond to requests from the deployed frontend, thereby enabling seamless communication between the two services.