MINT SAGE: An AI-Based Expense Tracker, Forecaster & Optimizer

Team 104: Viraj Paradkar, Vedant Ranade, Archita

Author Contributions

Member	Contribution
Viraj Paradkar	● Integrated the Plaid API to securely fetch and normalize real-time financial transaction data. ● Designed, optimized, and deployed AWS Lambda functions to support scalable and efficient model inference via API endpoints. ● Performed comprehensive exploratory data analysis (EDA) on transactional and forecasted time-series data to extract meaningful financial insights. ● Systematically experimented with prompt structures and model inputs to improve the quality, relevance, and consistency of LLM-generated financial recommendations. ● Developed an end-to-end frontend application using Next.js and Supabase, delivering a cohesive user journey from data ingestion to analytics, forecasting, and AI-driven insights.
Vedant Ranade	● Modeled Probability Distributions of 2 datasets ● Used Statistics to analyze and combine datasets ● Compared various Statistical models and ML models ● Applied Concepts like hypothesis testing for predicting the distribution of the sample. ● Developed a forecasting module, email Parser, and mail categorizer. ● Developed Stramlit Demo ● Tailored Forecasting Methodology, Results, and Overall Formatting
Archita	● Conducted **iterative model experimentation** (TF-IDF → USE → MiniLM) to identify optimal semantic embeddings for short and noisy transaction text. ● Categorization engine using MiniLM embeddings + SVM + XGBoost classifiers for multi-class categorization and combined them using a probability-based ensemble ● Preprocessing pipeline with feature engineering, rare-class filtering. ● Developed an **integration-ready categorization API** consumed by the email parser, expense tracking, and forecasting modules ● Performed model evaluation and error analysis ● Contributed to system architecture for data flow & module interfaces

Index

List of Figures……..3
Abstract……4
Introduction……..5
1.1 Existing challenges in personal financial systems……5
1.2 Problems to be addressed by this project……..5
1.3 Our Scope……..6
2. Prior Work……6
2.1 NLP for Expense Categorization……..6
2.2 Forecasting in Personal Finance……...7
2.3 Optimization in Personal Finance……7
3. Methodology……8
3.1 Overall Pipeline……8
3.2 Expense Categorization……10
3.3 Expense Forecasting……...14
3.4 Optimization via EDA + LLM-Guided Budgeting……18
4. Dataset……21
4.1 Categorization Dataset……21
4.2 Forecasting……..25
5. Overview……29
5.1 Demo……...29
5.2 Categorization Model……33
5.3 Forecasting Model……..35
5.4 Optimization Outcome……38
6. Discussion……..40
6.1 Challenges Faced……40
6.2 Future Work……41
References……..42
Appendix……43

List of Figures

Figure 3.2: Categorization Methodology…..……10
Figure 3.3: Expense Forecasting Methodology……..…….....*14*
Figure 3.3: Expense Forecasting Methodology……....*16*
Figure 3.3.3.1.2: ML Tree models……..*17*
Figure 3.3.3.2.1: Ensemble learning framework……..*17*
Figure 4.1.1.1.1: Expense Category Distribution (Before Cleaning).....*22*
Figure 4.1.1.1.2: Expense Category Distribution (After Cleaning).....*22*
Figure 4.1.1.2.1: Distribution of Description Text lengths……....*23*
Figure 4.1.1.3.1: Category vs Text length Variation……...*24*
Figure 4.2.1.1: Data Distribution range……....*25*
Figure 4.2.1.2: Cullen-Frey Graph……...*26*
Figure 4.2.1.3: Dataset 1 PDF and CDF……....*26*
Figure 4.2.2.1: Data Distribution Dataset 2…… ..*27*
Figure 4.2.2.2: Cullen-Frey Graph……...*28*
Figure 4.2.2.3: Dataset 2 PDF and CDF ……...*28*
Figure 5.1.1: Landing Page ……..*29*
Figure 5.1.2: Expense Categorization From Text……..*30*
Figure 5.1.3: Monthly Forecast Input Options……..*31*
Figure 5.1.4: Synthetic Dataset Generated……...*31*
Figure 5.1.6:Visualizing forecast with historical data ……...*31*
Figure 5.2: Confusion Matrix……...*33*
Figure 5.3.1: Ensemble model prediction on Validation set……...*35*
Figure 5.3.2: Prophet Forecast……..*35*
Figure 5.3.3: Prophet Model Additional Features……...*36*
Figure 5.3.4: Comparative MAE between models……..*37*
Figure 5.3.5: Prophet Model Additional Features……..*37*

Abstract

Personal finance management is often tedious and error-prone due to scattered expense records, multi-channel transaction sources, and the absence of intelligent decision-making support. Most individuals struggle to track spending patterns, manually categorize purchases, and forecast future expenses, which ultimately affects long-term budgeting discipline. *Mint Sage* addresses these challenges by developing a unified AI-driven financial assistant capable of automating end-to-end expense management.

The system integrates three major components — **expense categorization**, **expense forecasting**, and **budget optimization**. Transaction text descriptions are preprocessed and converted into dense semantic vectors using **MiniLM (Sentence-BERT embeddings)**, enabling contextual understanding beyond keyword matching. These embeddings are fed into **SVM and XGBoost classifiers**, later combined into an **ensemble model** for improved robustness and generalization across overlapping categories. The categorization engine achieves an accuracy of **~90%**, demonstrating strong performance even on short and noisy transaction descriptions.

For financial prediction, time-series statistical modeling is employed using feature-engineered rolling windows, seasonal patterns, and smoothed trend signals. This enables forecasting of expected future expenditure, helping users anticipate overspending with reasonable predictive stability. In the final pipeline stage, an **LLM-based recommendation layer** analyzes categorized spending and predicted costs to generate personalized budgeting strategies, savings insights, and potential expense optimizations.

Overall, Mint Sage functions as a scalable personal finance intelligence platform that not only automates tracking but also interprets financial behavior, forecasts upcoming spending, and guides users toward data-driven financial decisions.

1. Introduction

Managing personal finances effectively requires a clear understanding of spending patterns, awareness of how expenses change over time, and the ability to plan future budgets accordingly. However, most individuals lack the tools or time to manually track every transaction, categorize them, or forecast future expenditure. Traditional methods, such as spreadsheets or apps that rely on manual entry, often fail to provide real-time insights, trends, or personalized recommendations. As a result, financial planning becomes reactive instead of proactive, leading to overspending, poor savings discipline, and a lack of long-term financial visibility.

The core motivation behind this project is to create an intelligent assistant that not only tracks expenses automatically but also learns user behavior, categorizes spending accurately, predicts future financial trends, and recommends budget optimizations. By combining machine learning, NLP, and statistical modeling, we aim to shift personal finance management from manual analysis to an automated, data-driven, adaptive workflow.

1.1 Existing challenges in personal financial systems

Despite the availability of finance applications, several limitations persist. Firstly, Transactions are fragmented across emails, SMS, banking apps, UPI, and credit statements, making unified tracking difficult. Secondly, Most consumer apps perform only logging, lacking predictive capabilities for future spending or savings projections.

Expense categorization is still largely manual or rule-based, failing with short, ambiguous, or unseen descriptions. Limited personalization — users rarely receive actionable budgeting insights or optimization strategies.Difficulty connecting analysis to action; data exists, but it does not convert into smart recommendations.These gaps indicate the need for a system capable of semantic understanding, dynamic learning, forecasting capability, and personalized advisory support.

1.2 Problems to be addressed by this project

1) Automating transaction processing and reducing manual effort
2) Understanding financial behavior from raw text descriptions
3) Predicting future spending patterns for better planning
4) Recommending budget improvements tailored to each user

1.3 Our Scope

This project proposes Mint Sage, a multi-module AI financial assistant designed to address the above challenges through:

1) AI-based Categorization Engine
   Built using *MiniLM sentence embeddings* combined with SVM + XGBoost Ensemble, improving semantic classification accuracy even on short/noisy text.
2) Forecasting Model for Expense Prediction
   Uses *statistical and ML-based time-series modeling* with feature engineering to estimate future spending trends and potential overshoot risks.
3) Budget Optimization Layer via LLM
   Generates human-readable saving suggestions, spending summaries, and optimization strategies using model outputs.
4) Unified End-to-End System
   Integrates tracking → categorization → forecasting → recommendation, moving towards a complete intelligent finance management pipeline.

2. Prior Work

Automated expense management has been studied in the domains of **Natural Language Processing, financial categorization, and time-series forecasting**, yet several limitations persist in semantic understanding, generalization, and personalized budgeting capability. Traditional systems rely heavily on keyword-based rules or manual tagging, resulting in poor performance for short or noisy transaction descriptions.

2.1 NLP for Expense Categorization

Earlier approaches predominantly used **TF-IDF or Bag-of-Words (BoW) feature extraction** to convert transaction text into numerical vectors. These methods, though computationally efficient, fail to capture contextual meaning. A term like *"Zomato order"* should imply *Food*, but TF-IDF treats it as unrelated tokens without prior semantic understanding. Research shows BoW-based classifiers struggle with synonyms, spelling variation, and unseen patterns (Jurafsky & Martin, 2021).

Similarly, rule-based systems used in early finance apps classify expenses using keyword matching (e.g., "Uber" → Travel), but they cannot generalize and require continuous manual updates. Studies indicate rule systems degrade rapidly when description formats vary across banks, vendors, or regions (Gupta et al., 2020).

To address semantic limitations, modern works explore **sentence embeddings and transformer-based representations**, enabling models to understand context rather than raw tokens. Sentence-BERT (Reimers & Gurevych, 2019) introduced a lightweight architecture generating meaningful vector representations, significantly outperforming TF-IDF in classification tasks. MiniLM, a compressed transformer model optimized for speed, retains language understanding capabilities with far lower inference cost — making it suitable for real-time financial categorization.

2.2 Forecasting in Personal Finance

Expense prediction is often approached with statistical time-series forecasting like **Holt-Winters, ARIMA, SARIMA**, etc. These models assume stationarity and cyclical seasonal behavior (Holt, 1957; Box & Jenkins, 2015). While effective for stable patterns, they underperform on irregular personal spending where variance is high, leading to error propagation and unstable results.

Recent works experiment with **ML and deep learning-based forecasters** that incorporate trend and window features for improved generalization (Makridakis et al., M4 Competition, 2020). Ensemble-based regressors such as Gradient Boosting and Random Forests have shown improved predictive resilience in volatile financial datasets, especially when engineered with rolling averages, lag features and smoothing.

2.3 Optimization in Personal Finance

Financial optimization with LLMs is increasingly built as a hybrid pipeline: the LLM helps understand and structure messy user input (goals, constraints, budgets, risk preferences, and qualitative “views”), and then a classical optimizer computes the final plan under explicit objectives and constraints. For example, de Zarzà et al. [de_zarza_optimized_2024] frame budgeting as an optimization problem and use LLM outputs to personalize recommendations, treating the LLM as a support tool rather than the solver. In portfolio settings, Lee et al. [lee_llm_enhanced_2025] show how LLM-based signals can be incorporated into the Black–Litterman framework by expressing them as “views” with confidence levels, so the optimizer can balance them against market priors instead of blindly trusting noisy predictions. Domain-specific financial LLMs also matter for these pipelines: BloombergGPT [wu_bloomberggpt_2023] highlights the benefits of finance-focused pretraining for financial language tasks, while FinGPT [yang_fingpt_2023] emphasizes open, adaptable workflows for building and deploying financial LLMs. Finally, real-world deployment needs strong governance and reliability measures; the Alan Turing Institute and HSBC report stresses robustness, privacy/security, fairness/transparency,

and accountability as core pillars for trustworthy adoption in financial services. [turing_hsbc_llms_finance_2024].

3. Methodology

3.1 Overall Pipeline

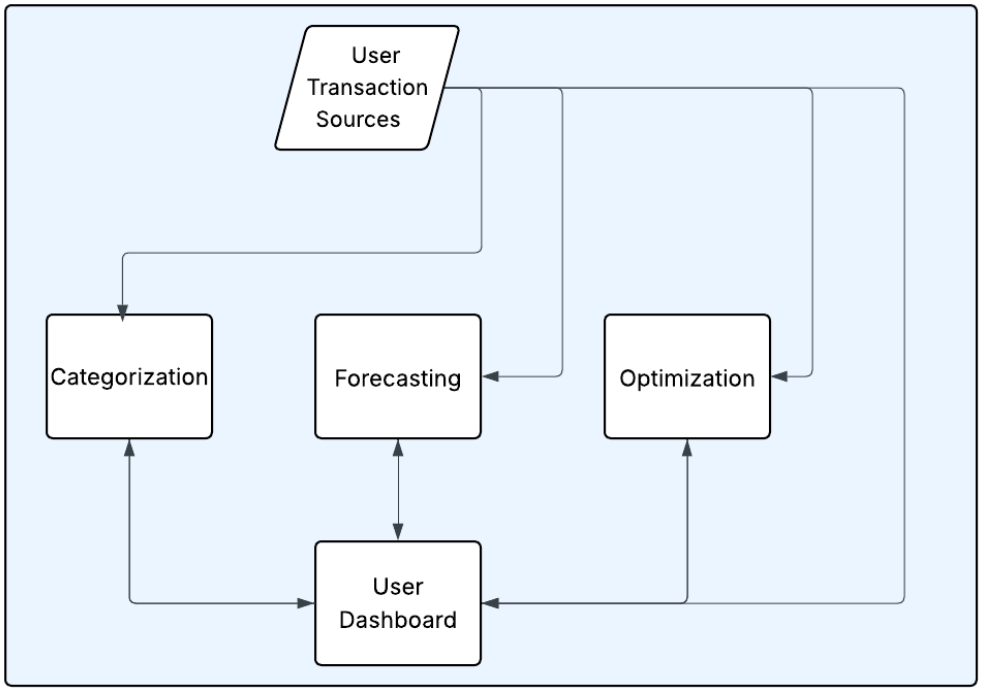


Figure. 3.1: Categorization Methodology

1. Transaction Ingestion

- The system collects incoming transactions through two channels: LLM-powered email parsing and real-time Plaid API fetches.
- The LLM extracts structured fields (description, transaction date, amount, vendor) from user emails, while Plaid provides verified banking transaction data.
- Both streams produce consistent, structured transaction records.

2. Categorization Engine (MiniLM + SVM/XGBoost Ensemble)

- Incoming transactions are cleaned, standardized, and normalized into a unified format.
- Transaction descriptions are cleaned, vendor-normalized, and filtered for rare or invalid entries to ensure consistent and stable model inputs across email and Plaid sources

- Each transaction description is converted into a semantic vector using MiniLM embeddings.
- SVM and XGBoost models independently classify the embedding into an expense category.
- A weighted ensemble combines both predictions to produce a robust, real-time category output.

3. Forecasting Module

- Statistical forecasting models such as Prophet or Holt-Winters predict future spending patterns across Spending Patterns.
- Windowed feature engineering (7-day and 30-day intervals) enhances model accuracy.
- The system estimates upcoming expenses and identifies categories where overspending is likely.

4. Budget Optimization (LLM Agent)

- The LLM agent analyzes forecasts and historical behavior to generate personalized budgeting recommendations.
- It highlights savings opportunities, spending anomalies, and potential risks.
- The agent acts autonomously, continuously improving suggestions as more data arrives.

5. Frontend Interface (Next.js Dashboard)

- The frontend renders a unified dashboard that surfaces parsed transactions, category-wise spend breakdowns, forecast curves, and computed EDA insights in a clean, interactive layout.
- Users can review transaction history, visualize spending patterns, and query an embedded chat interface to receive LLM-generated recommendations grounded in the latest analytics.
- The UI pulls data through backend API routes (transactions → forecast → EDA → insights) and refreshes dynamically at runtime, without relying on an intermediate storage layer.

3.2 Expense Categorization :

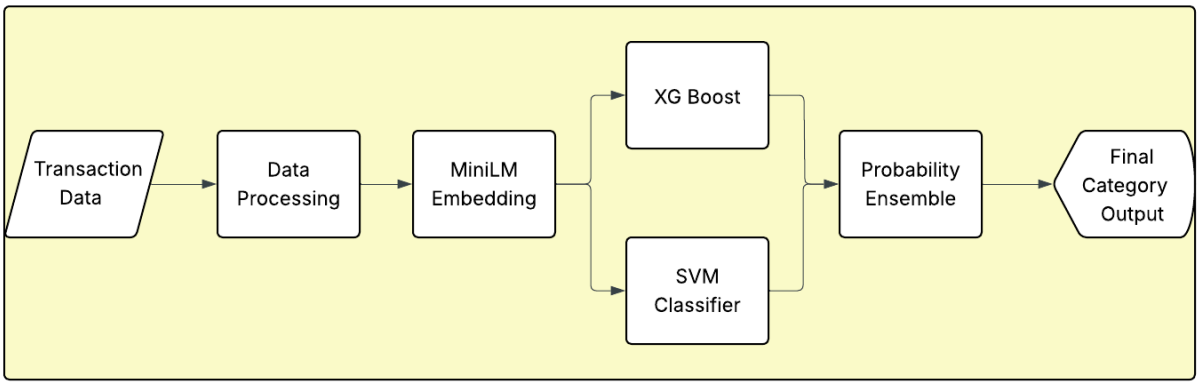


Figure. 3.2: Categorization Methodology

The objective of the system is to automate transaction classification using NLP techniques and semantic embeddings. The categorization methodology consists of four main stages: data preprocessing, feature representation, model training, and prediction via ensemble inference.

3.2.2 Data Preprocessing

To ensure clean and consistent model inputs, raw transaction descriptions underwent structured preprocessing:

1. **Removed null and duplicate entries** to eliminate noise.
2. **Standardized text** (lowercasing, trimming whitespace, punctuation cleanup).
3. **Rare category filtering** — categories with ≤1 sample were dropped to prevent skewed learning.
4. **Token length analysis** performed to understand distribution and guide model selection.

After cleaning, the dataset contained 20 categories with improved balance suitable for classification tasks.

3.2.3 Feature Representation using MiniLM Embeddings

Traditional TF-IDF representations fail to capture semantic meaning in short transaction descriptions. To overcome this, we use **MiniLM Sentence Transformer (all-MiniLM-L6-v2)** to convert text descriptions into numerical vector embeddings.

1) Each transaction description → converted into a **384-dimensional dense embedding vector**.
2) Embeddings preserve **semantic similarity**, enabling the model to generalize: *"McDonald's meal" ≈ "Lunch takeaway"* → Food *"Uber ride" ≈ "Cab airport drop"* → Transport

This step forms the core reasoning capability of the classifier.

3.2.4 Model Training

Two supervised models were trained independently on the MiniLM embeddings:

A) Support Vector Machine (SVM)

- The SVM model employs an RBF kernel and is optimized using GridSearchCV, enabling effective learning on high-dimensional semantic embeddings. It serves as the primary decision boundary classifier within the categorization ensemble.

B) XGBoost Classifier

- The model is tuned specifically for multi-class classification, enabling it to effectively learn complex non-linear decision boundaries while remaining resilient to noisy and ambiguous transaction data points.

Each model outputs probability scores for all categories.

3.2.4 Ensemble Classification Strategy

Instead of relying on a single model, the final prediction is obtained using a **probability-averaged ensemble**:

$$P_{ensemble} = \alpha \cdot P_{SVM} + (1 - \alpha) \cdot P_{XGB}$$

Where

• $\alpha = 0.7$ optimized weight (SVM influence higher)

• Final category = argmax($P_{ensemble}$)

This improves classification stability and reduces overfitting, especially on borderline cases.

3.2.5 Inference Output

During inference, the trained categorization pipeline processes incoming transaction descriptions in real time and generates structured outputs used by downstream components of the system.

Input

- Transaction description (string)
- Optional metadata (vendor name, transaction date)

Example Input:

"Starbucks coffee and a sandwich"

Inference Process

1. The transaction description is converted into a dense semantic vector using the MiniLM sentence embedding model.
2. The embedding is independently evaluated by:
    - Support Vector Machine (SVM)
    - XGBoost classifier
3. A probability-based ensemble strategy combines the outputs of both models to produce the final prediction.

Output

Predicted expense category label, such as: *Food, Transport, Utilities, Entertainment, Personal Care, Insurance, Software, Electronics, etc.*

Example Output:

Predicted Category: *Food*
This demonstrates the system's ability to correctly interpret short, real-world transaction descriptions without relying on explicit keywords. Along with the final category label, the inference module also produces:

- Individual model predictions (SVM and XGBoost)
- Ensemble prediction (final decision)
- Confidence scores (class probabilities)
- Normalized transaction record forwarded to tracking and forecasting modules

Misclassification Analysis

To evaluate robustness, misclassified examples were analyzed across all models.

- Total misclassified samples (Ensemble): 7
- Most errors occurred between semantically overlapping categories, rather than unrelated ones.

Description	True Label	SVM Prediction	XGBoost Prediction	Ensemble Prediction
Gift card purchase for employee	Gifts	Food	Business Expenses	Business Expenses
Weekend spa visit	Personal Care	Travel	Food	Travel
Subscription to a cloud storage service	Software	Entertainment	Entertainment	Entertainment
Amazon purchase for home gadgets	Electronics	Entertainment	Household	Household
Magazine subscription for business trends	Books	Entertainment	Business Expenses	Entertainment
New laptop bag	Clothing	Electronics	Business Expenses	Business Expenses
Cost of attending a networking event	Professional Services	Entertainment	Entertainment	Entertainment

Table 3.2.5.1: Comparative results

Observations from Inference Results

- The ensemble approach consistently performs better than individual classifiers by balancing their strengths and **minimizing extreme prediction errors**.
- Most **misclassifications** occur between categories that are semantically similar, indicating contextual ambiguity rather than model instability.
- **Short transaction descriptions** that lack vendor information or clear keywords remain the most challenging cases. Overall, the model demonstrates strong and reliable performance on frequently occurring categories such as Food, Transport, Entertainment, and Utilities.

System Integration

The inference output is directly consumed by:

- Expense tracking module for aggregation and analytics
- Forecasting module for time-series prediction
- LLM-based budget optimization layer for personalized recommendations

This ensures a fully automated pipeline from transaction ingestion to financial insight generation.

3.3 Expense Forecasting

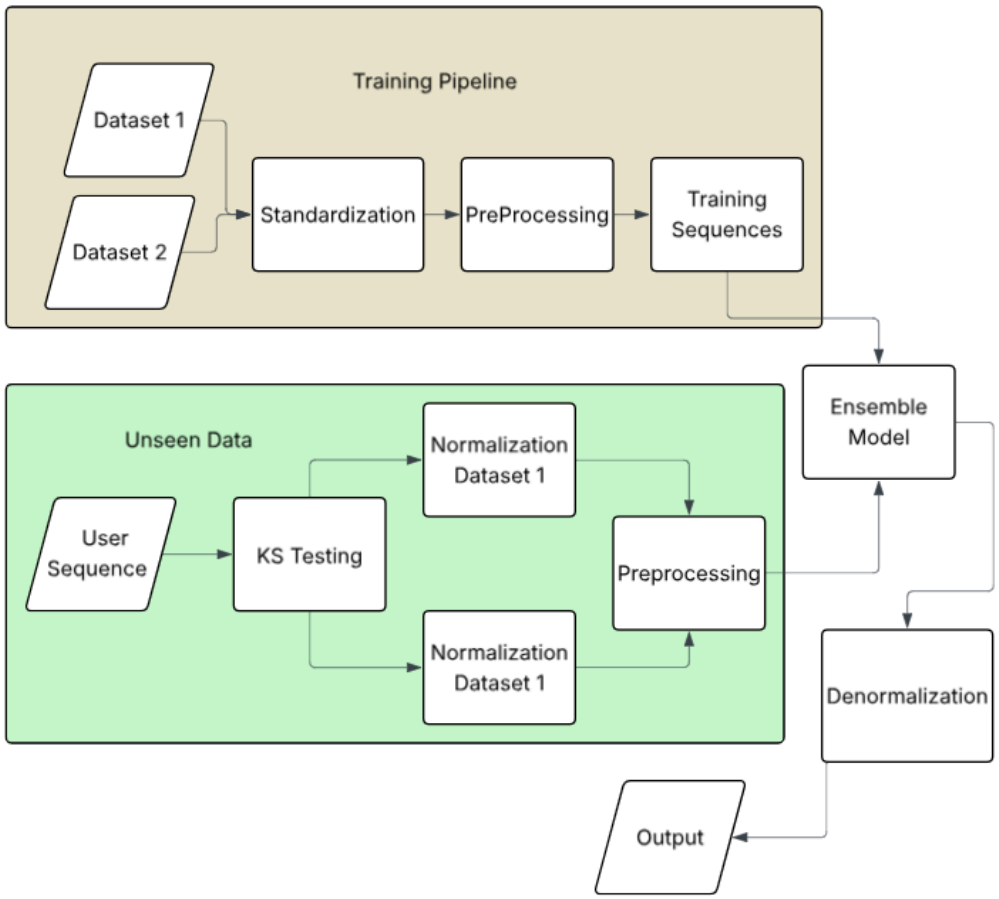


Figure 3.3: Expense Forecasting Methodology

3.3.1 Training Pipeline :

3.3.1.1 Datasets:

The two datasets are combined using standardization or normalization according to the min and max values of the respective datasets, so that the values are restricted to a fixed range

3.3.1.2 Standardization:

The Dataset is normalized by scaling it between -1 and 1. This step equalizes scale while preserving relative variation within each source. After standardization, both

datasets approximately follow a similar distribution, which makes them more homogeneous and suitable for learning a single forecasting model.

3.3.1.3 Training Sequences

On the standardized series, a sliding-window procedure was used to construct supervised learning samples. For each day t, a window of the previous 60 days of standardized expenses was extracted as the input sequence X. For the target Y, the label was defined as the mean expense over the next 30 days. This design makes the model directly predict the 30-day average, which is more stable and business-relevant than predicting individual daily amounts.

3.3.1.4 Preprocessing

Within each 60-day window, additional rolling-window statistics were computed to capture local trends and volatility. In particular, rolling means and standard deviations over 7-day, 14-day, and 30-day horizons were added as features. These are classical technical indicators in time-series modeling and help the model distinguish short-term fluctuations from medium-term shifts. Calendar covariates (day of week, month, day of month, and a weekend flag) were also encoded to capture weekly and monthly seasonality patterns commonly observed in spending behavior.

3.3.2 Unseen Data Pipeline :

For a new user sequence of expenses, the system first applies the same preprocessing pipeline used during training. Because the original datasets followed different underlying distributions, a statistical test is used to decide which standardization parameters are more appropriate. Specifically, a two-sample Kolmogorov–Smirnov (KS) test compares the empirical distribution of the user's recent expenses to each training dataset. The dataset whose distribution is closest in the KS sense is selected, and its standardization parameters (mean and standard deviation) are used to normalize the user sequence.

3.3.2.1 KS Testing :

Empirical histograms and kernel density estimates indicated that each dataset followed a distinct, skewed "beta-like" distribution, with noticeably different modes and tail behavior. This confirmed that simply concatenating the raw series would distort the combined distribution and could bias a model towards the dominant source. Therefore, the datasets were handled separately during standardization and only merged after mapping them into a common standardized space.

Once the appropriate dataset has been selected, the user's expenses are standardized using the corresponding mean and standard deviation. The same 60-day sliding window, rolling statistics (7/14/30-day mean and standard deviation), and calendar features are then computed to construct the model input. This strict reuse of the training-time transformations ensures that the ensemble receives inputs that are on the same scale and feature space as during training, which is critical for reliable generalization.

The standardized feature vector is then passed through the tuned ensemble model, which outputs a prediction for the average standardized expense over the next 30 days. This predicted value is subsequently denormalized using the inverse transformation of the selected dataset, producing a forecast on the original expense scale.

3.3.3 Modelling :

3.3.3.1 Baseline forecasting models

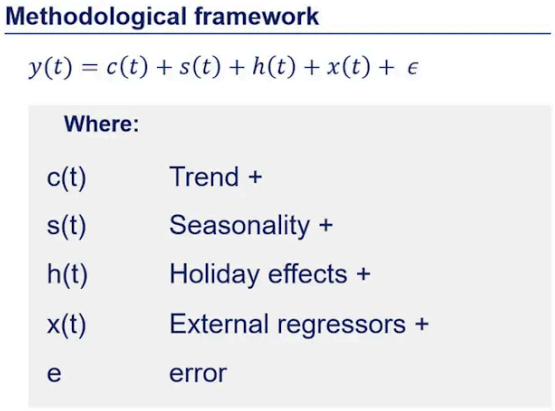


Figure 3.3.3.1.1: Prophet Model framework

The *above Figure* is about the parameters the Prophet model considers to forecast a time series problem.

As a benchmark, classical univariate forecasting models were first evaluated on each standardized series. These included Exponential Smoothing, SARIMA, and Prophet. Each model was trained on the historical expense sequence and used to generate daily forecasts, which were then aggregated to an average to match the target definition. While these models provided reasonable performance, their capacity to exploit the richer window-based feature set (rolling statistics, calendar effects) is limited, as they were only univariate.

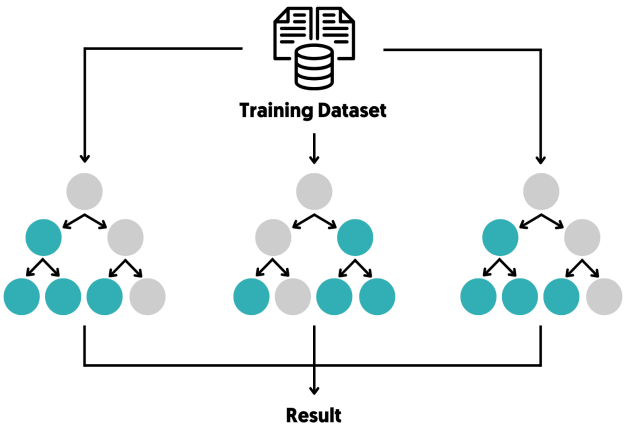


Figure 3.3.3.1.2: ML Tree models

As one can see from the *Figure*, various models or trees can be used in a manner to get a more robust output, as different trees can capture different features
Next, supervised regressors were trained directly on the sliding-window features. Random Forest (RF), Gradient Boosting Regressor (GBR), and Ridge Regression were used as they represent complementary biases

1) RF captures nonlinear threshold interactions,
2) GBR focuses on reducing bias through boosted trees, and
3) Ridge provides a strong linear baseline with L2 regularization.

Each model takes the 60-day feature vector as input and outputs a single scalar prediction corresponding to the average expense over the next 30 days

3.3.3.2 Ensemble learning

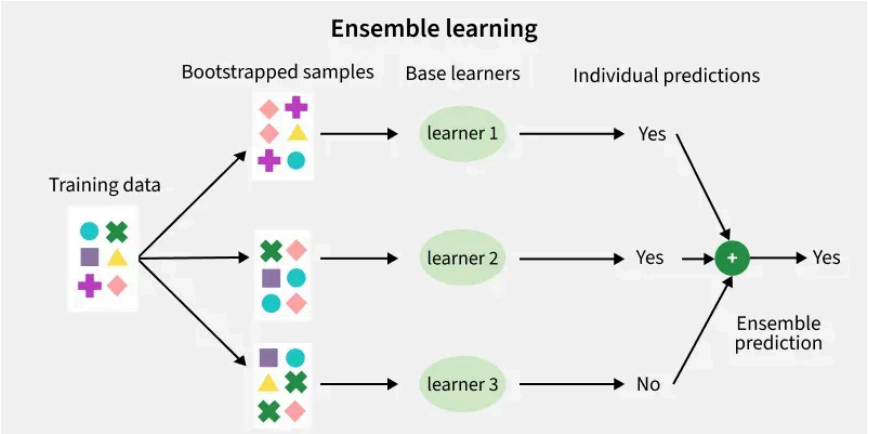


Figure 3.3.3.2.1: Ensemble learning framework

The above *Figure* shows the working of ensemble models

To leverage the strengths of all three regressors, an ensemble learning strategy was adopted. RF, GBR, and Ridge were individually fine-tuned on the full standardized feature set, including additional engineered variables discussed earlier to better encode recent trends and seasonal structure.

Their predictions were then combined into a single ensemble output to form the final 30-day forecast. This ensemble substantially reduced mean absolute error relative to the best single model and outperformed all baseline time-series methods.

3.4 Optimization via EDA + LLM-Guided Budgeting

EDA Feature Engineering (Transaction + Forecast-Aware)

CORE AGGREGATES
Let a_i denote the transaction amount for transaction i.

Sign convention:

- Income transactions: $a_i < 0$ (money coming in)
- Expense transactions: $a_i > 0$ (money going out)

Compute:

- Total Income = sum of $|a_i|$ over all transactions where $a_i < 0$
- Total Expense = sum of a_i over all transactions where $a_i > 0$
- Total Net = Total Income − Total Expense

Let D_{active} be the number of "active days" (days that contain at least one transaction).

- Avg Daily Net = Total Net / $|D_{active}|$

TEMPORAL BEHAVIOR METRICS
We define a daily net so that income contributes positively and expenses negatively.

For each day t:

- s_t = sum of transaction amounts on day t (this sum mixes negative income and positive expenses)
- DailyNet(t) = $- s_t$
  (So a day with more expenses tends to become negative DailyNet, and a day with more income becomes positive DailyNet.)

Compute:

- Trend direction and slope: fit a simple linear regression line to DailyNet(t) versus time (day index).
    - Slope > 0 suggests improving net over time; slope < 0 suggests worsening net.
- Volatility: standard deviation of DailyNet(t) across days (higher = more unstable day-to-day behavior).

CATEGORY-LEVEL STRUCTURE (EXPENSES ONLY)

For each category c:

- Spend(c) = sum of a_i over all transactions where a_i > 0 and category(i) = c

From these, extract:

- Top spending categories (rank by Spend(c))
- Category distribution / concentration (e.g., Share(c) = Spend(c) / Total Expense), to assess whether spending is diversified or dominated by a few categories.

BEHAVIORAL DESCRIPTORS

Compute:

- Most frequent vendor: vendor with the highest transaction count
- Most frequent category: category with the highest transaction count
- Largest expense day: the day with the most total expenses (equivalently, the most negative DailyNet(t))
- Largest income day: the day with the most total income (equivalently, the most positive DailyNet(t))
- Avg expense per active day = Total Expense / |D_active|
- Avg income per active day = Total Income / |D_active|
- Expense/Income ratio: r = Total Expense / Total Income
    - If r > 1, expenses exceed income over the window.

FORECAST SUMMARY FEATURES

Given a forecast series yhat_1, …, yhat_H for an H-day horizon:

Compute:

- Forecast Mean = (1/H) * sum of yhat_k for k = 1…H
- Forecast Trend (up/down/flat): compare the last forecast point to the first

- Up if yhat_H > yhat_1
- Down if yhat_H < yhat_1
- Flat if they are approximately equal (optionally use a small threshold).

INSIGHT EXTRACTION AND OPTIMIZATION TARGETS

We convert EDA outputs into optimization-relevant signals:

Signals:

- Overspending signal: Total Net < 0 and/or r > 1
- Instability signal: high DailyNet volatility and large day-level spikes (especially the largest expense day)
- Concentration signal: a small number of categories accounts for a large share of Total Expense (high Share(c) for top categories)
- Forecast risk signal: forecast trend is upward while current Total Net is negative (expected spending pressure)

These signals translate into actionable targets:

- Reduce spending in top categories contributing the largest share of expenses.
- Stabilize spending by focusing on high-variance categories/vendors that cause spikes.
- If income is low relative to expenses, recommend budget caps and prioritization rules (essentials first, discretionary reductions next).

PROMPT CONSTRUCTION FOR LLM-GUIDED BUDGET OPTIMIZATION

We use a structured prompt template that injects only the necessary computed features (rather than raw transactions):

Prompt inputs:

- Totals: Total Income, Total Expense, Total Net, Avg Daily Net
- Temporal metrics: trend direction, slope, volatility
- Forecast metrics: Forecast Mean, Forecast Trend
- Top spending categories (category → amount, and optionally share of Total Expense)
- Key anomalies: largest expense day, largest income day, most frequent vendor, expense/income ratio r

LLM instructions:

1. Identify good patterns (e.g., positive net, stable DailyNet, diversified spending).

2. Identify bad patterns (e.g., persistent negative net, high volatility, category concentration).
3. Produce budget optimization actions: category-level caps, reduction strategies, and prioritization rules.
4. Return concise, grounded recommendations and explicitly reference the provided metrics in the reasoning (e.g., cite Total Net, r, top categories, volatility, and forecast trend).

4. Dataset

4.1 Categorization Dataset:

Link:
https://github.com/nirajdsouza/personal-finance-assistant-ai-agent/blob/main/data/expense_data.csv
The dataset used for expense categorization was sourced and adapted from publicly available personal finance repositories, including the Kaggle-like dataset accessible at: Each entry contains two key fields:

- **Description** – free-text transaction details
- **Category** – labeled spending class for classification

The initial dataset contained **24 categories**, but several had extremely low frequencies (1 sample each), which introduces imbalance and negatively affects model learning. To create a more stable training distribution, rare categories were removed, resulting in **20 final categories** retained for modeling.

4.1.1 Graph-Based Dataset Interpretation

4.1.1.1 Category Distribution Before vs After Cleaning

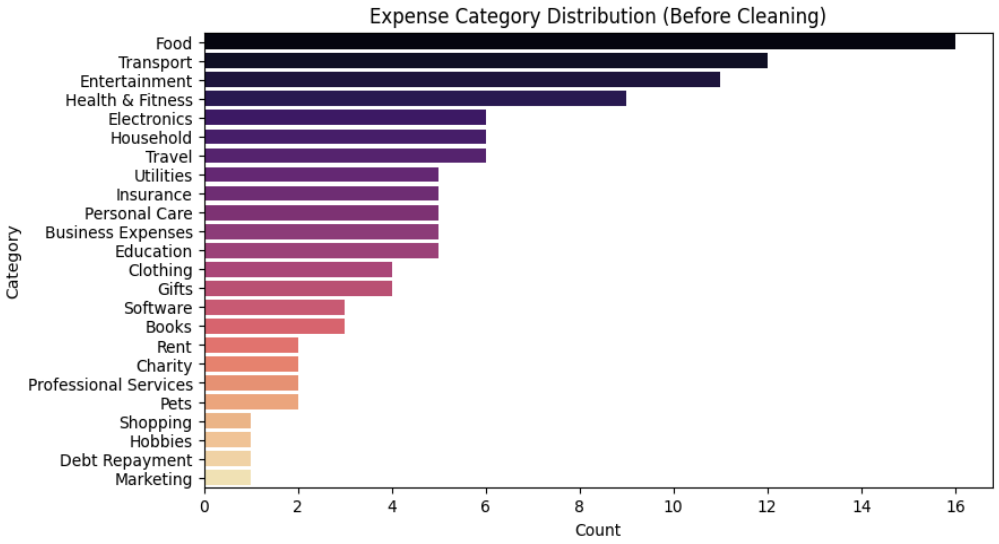


Figure 4.1.1.1.1: Expense Category Distribution (Before Cleaning)

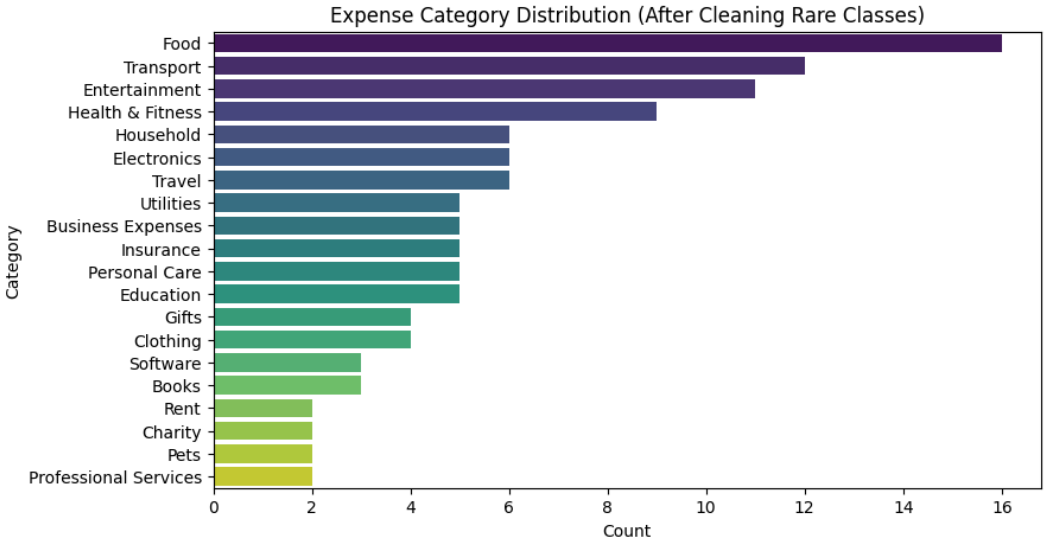


Figure 4.1.1.1.2: Expense Category Distribution (After Cleaning)

The first two bar plots display the number of samples in each category:

- **Before cleaning**, the distribution is highly imbalanced. Top categories like *Food, Transport, Entertainment* dominate, while *Shopping, Hobbies, Debt Repayment,* etc. contain only a single record.
- **After removing rare classes**, the dataset becomes more uniform, reducing noise and improving training stability.
- This step was crucial for model performance, preventing overfitting to minority classes.

Key Observations:

- Food & Transport are the most common expense types
- Categories below 2 samples were filtered out
- Post-cleaning dataset is more statistically reliable

4.1.1.2 Description Text Length Distribution

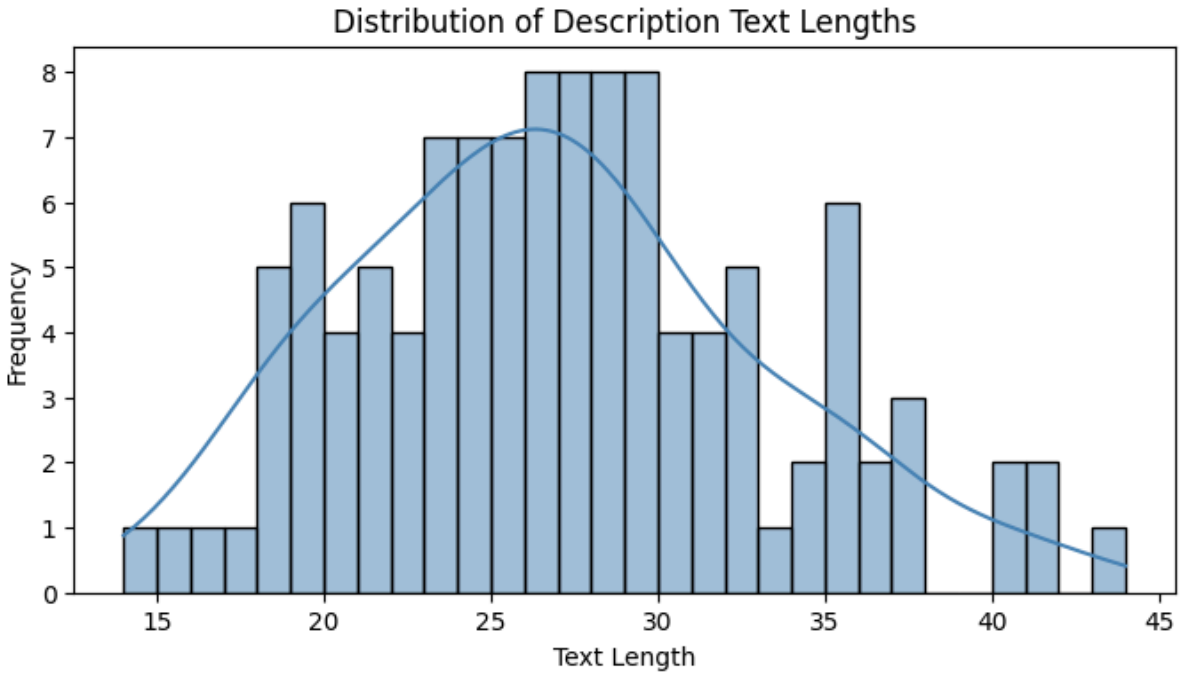


Figure 4.1.1.2.1: Distribution of Description Text lengths

The histogram shows how long transaction descriptions tend to be.

- Most descriptions lie between **20–32 characters**, peaking around ~26.
- The curve resembles a normal distribution, indicating fairly consistent text size.

- Short sentence style means **keyword-based TF-IDF is insufficient**, motivating transformer embeddings that capture semantics instead of just word frequency.

Key Insight:

> *Short & compact descriptions → semantic embeddings (MiniLM) are a better representation strategy than simple keyword models.*

4.1.1.3 Category vs Text Length Boxplot

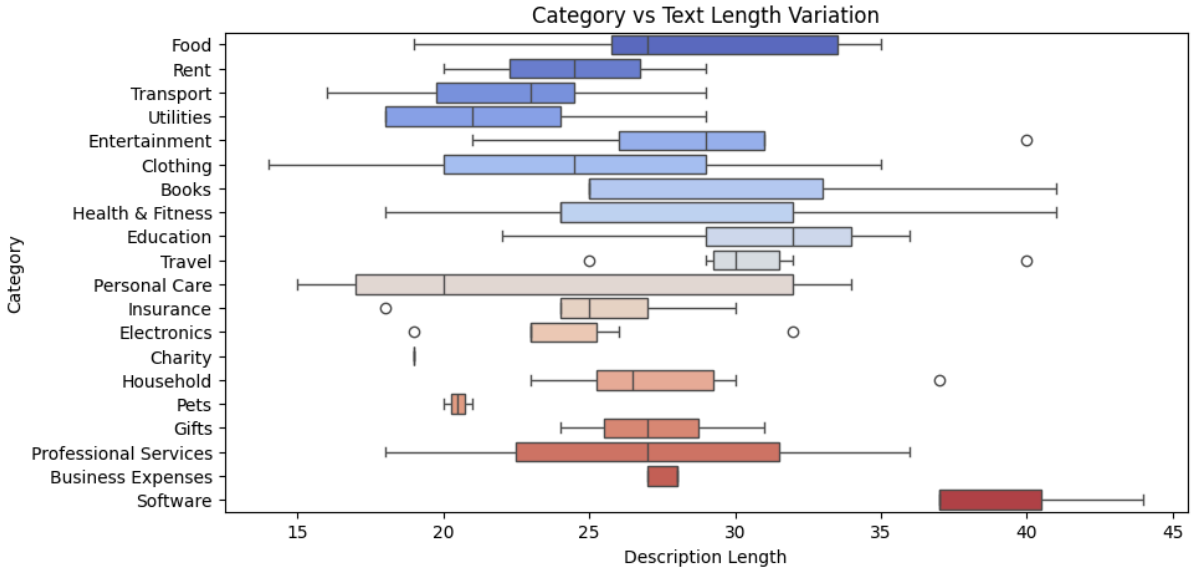


Figure 4.1.1.3.1: Category vs Text length Variation

This visualization compares description length variation across categories.

- Categories like **Food, Entertainment, Transport** show wider text length ranges, reflecting natural diversity in transaction wording.
- More structured categories like **Pets or Books** have compact descriptions.
- Outliers indicate rare descriptive patterns (e.g., unusually long messages for Software or Services).

This analysis confirms that
→ **Text length variation cannot be used alone for classification**
→ **Contextual meaning matters more than word count**

This again justifies the use of **Sentence-BERT / MiniLM embeddings** over frequency-based models.

4.1.1.4 Summary of Dataset Characteristics

Aspect	Observation
Total categories initially	24
Categories after cleaning	20
Most common labels	Food, Transport, Entertainment
Average text length	20–32 characters
Data issue handled	Class imbalance + rare class removal
Motivation for embeddings	Short/semantic text → transformer models fit better

Table 4.1.1.4: Dataset Summary

4.2 Forecasting

4.2.1 Dataset 1:

Link: https://www.kaggle.com/datasets/ramyapintchy/personal-finance-data

This dataset is a CSV file with 2 columns selected: 'Date', 'Category', 'type and 'Amount', but Category and type columns were dropped as there was a data mismatch and would have required a larger data sample. This Dataset was cleaned, then EDA was performed to get more insights from the dataset, like range and Distribution of Data, which help to combine the datasets and standardize the data in a fixed range.

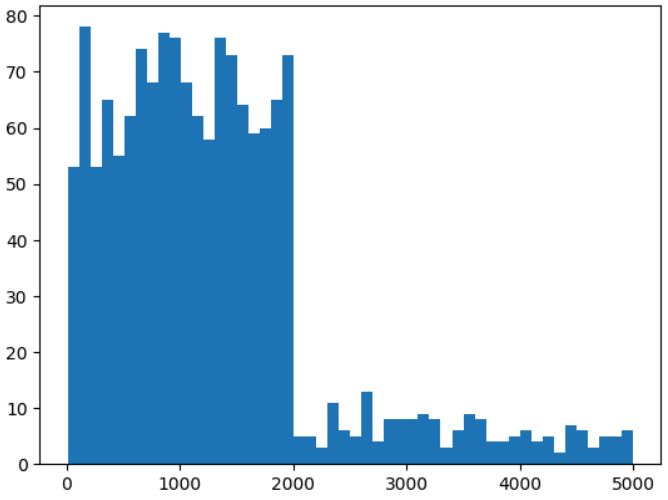


Figure 4.2.1.1: Data Distribution range

Figure 4.2.1.1 shows the Distribution of the Amount of Expense in the entire dataset.

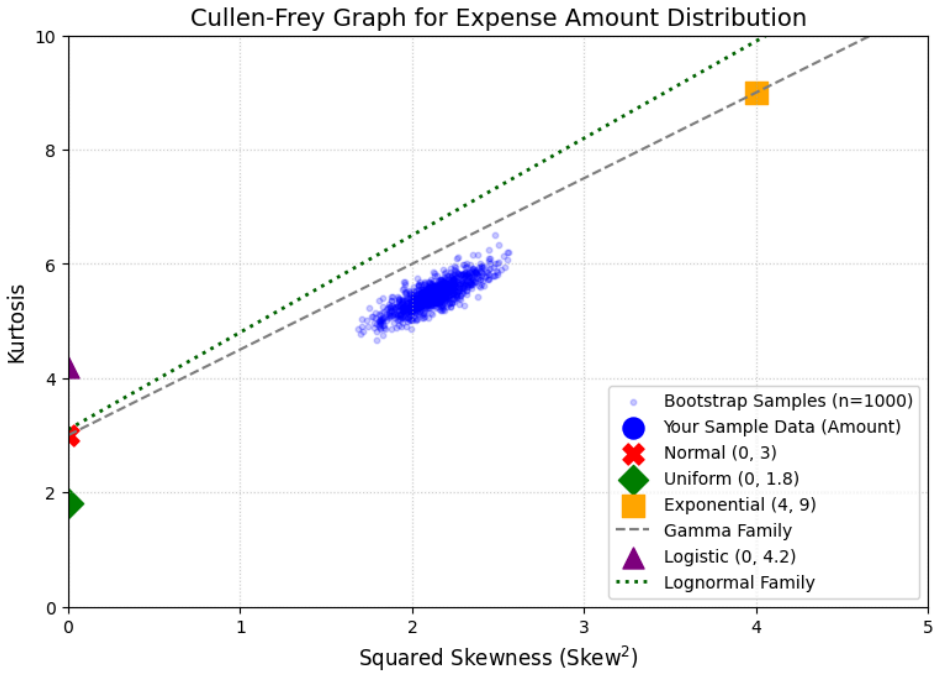


Figure 4.2.1.2: Cullen-Frey Graph

Figure 4.2.1.2 shows the CF graph plotted using Kurtosis and Squared Skewness statistics. It is a well-known method to visualize data distribution. One can clearly conclude from the graph that the bootstraped data is likely to follow a beta or gamma distribution.

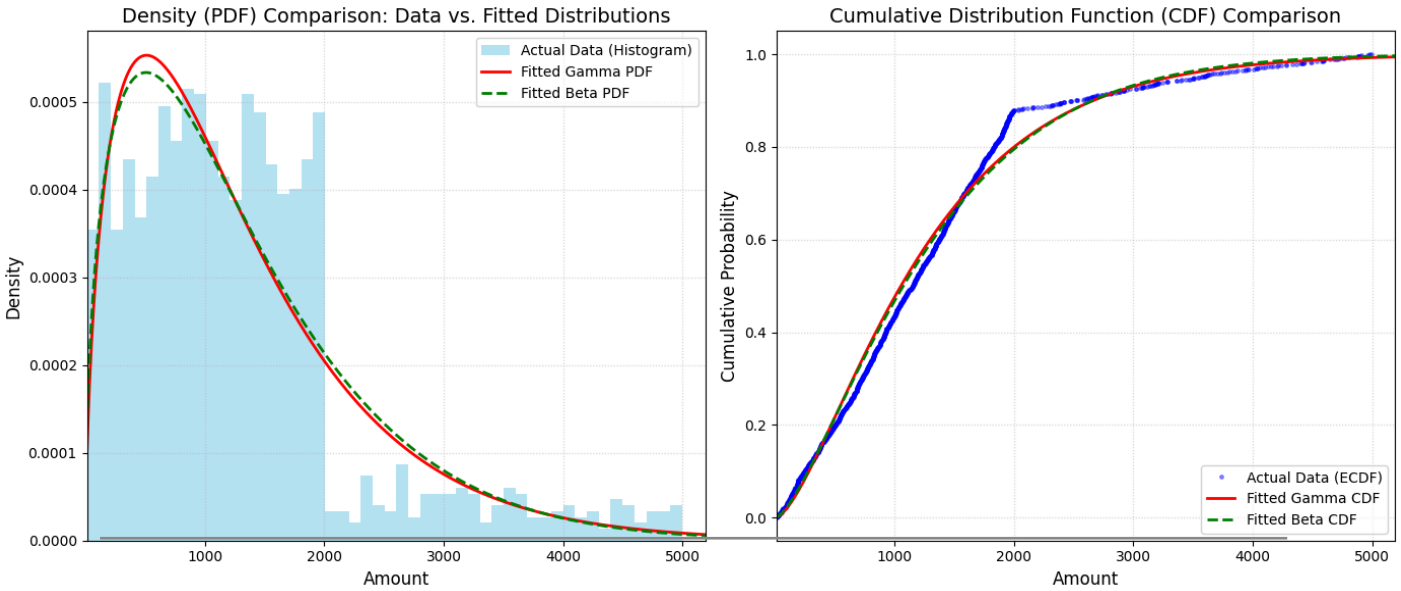


Figure 4.2.1.3: Dataset 1 PDF and CDF

In the above *Figure 4.2.1.3*, one can clearly see that when one fits the ideal beta and gamma on the data, one gets the following PDF and CDF, although they are very close. We can safely assume it follows a beta distribution.

4.2.2 Dataset 2:

Link : https://www.kaggle.com/datasets/ramyapintchy/personal-finance-data

This dataset is a CSV file with 2 columns selected: 'Date', 'Category', 'type and 'Amount' but Category and type columns were dropped as there was a data mismatch and would have required a larger data sample. This Dataset was also cleaned, then EDA was performed to get more insights from the dataset, like range and Distribution of Data, which will help to combine the datasets and standardize the data in a fixed range.

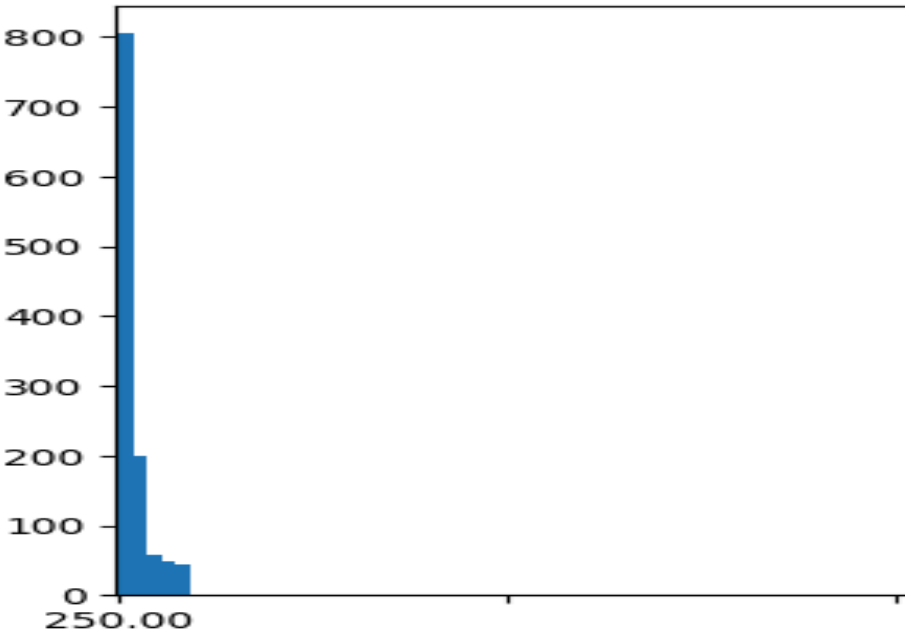


Figure 4.2.2.1: Data Distribution Dataset 2

The *Figure 4.2.2.1* shows the Distribution of the Amount of Expense in the entire dataset. As we can clearly compare and see that both datasets do not have different ranges, but possibly different distributions as well.

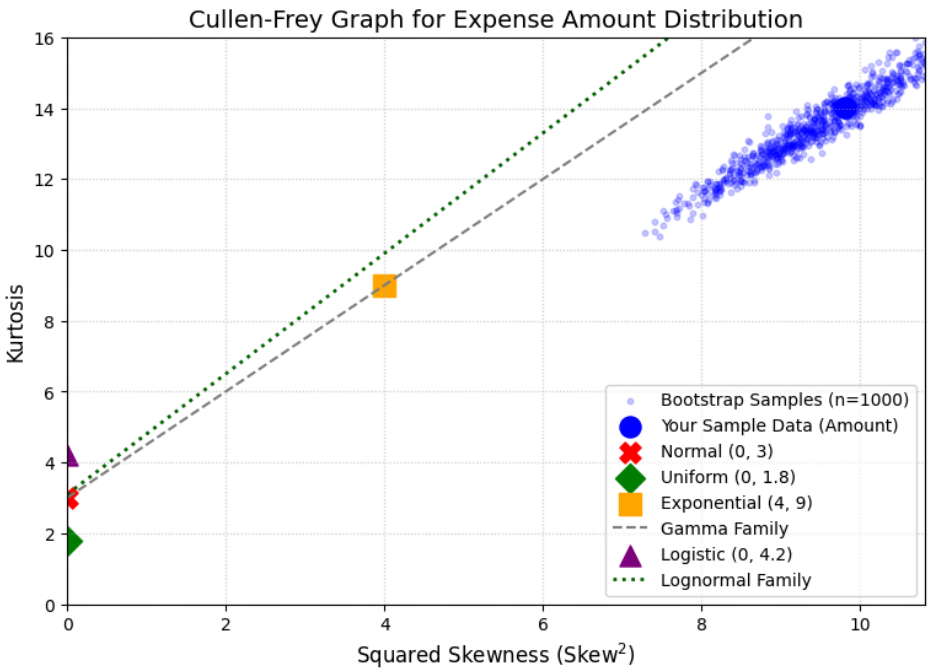


Figure 4.2.2.2: Cullen-Frey Graph

The *Figure 4.2.2.2* shows the CF graph plotted using Kurtosis and Squared Skewness statistics for the second dataset. One can clearly conclude from the graph that the bootstraped data is likely to follow a beta or gamma distribution.

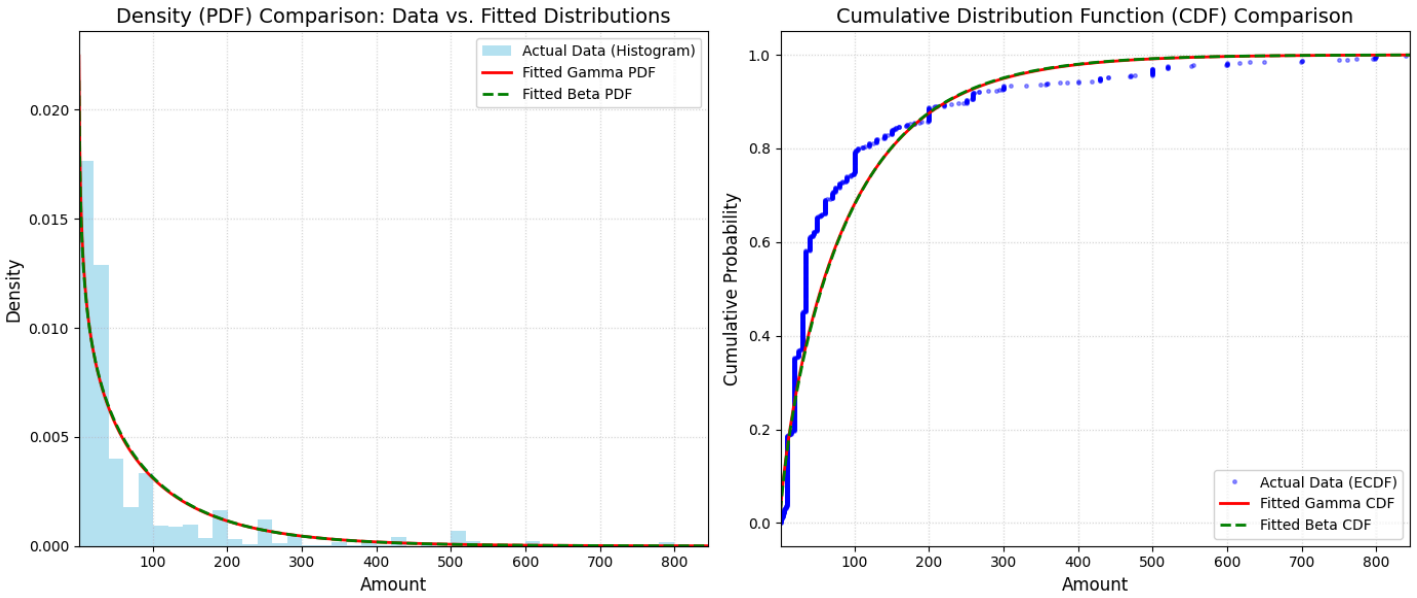


Figure 4.2.2.3: Dataset 2 PDF and CDF

In the above *Figure 4.2.2.3*, one can clearly see that when one fits the ideal beta and gamma on the data, one gets the following PDF and CDF, although they are very close. We can safely assume it follows a beta distribution

5. Results

5.1 Demo

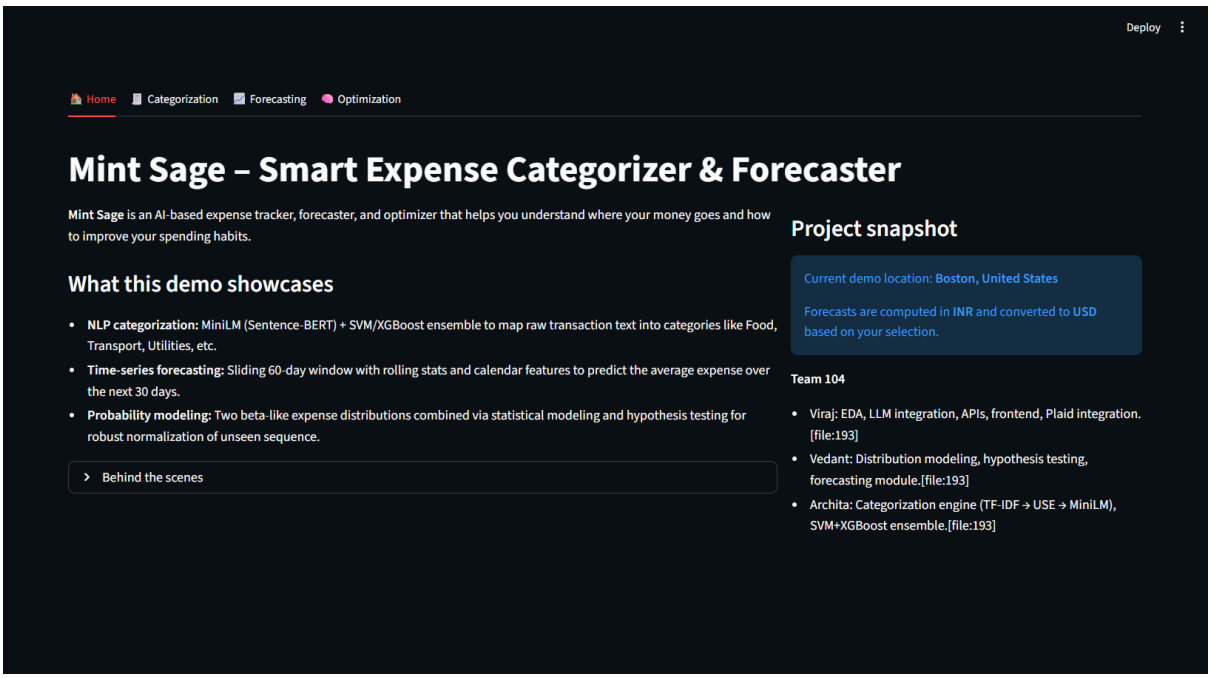


Figure 5.1.1: Landing Page

Figure 5.1.1 is an image of the home page or landing page of the demo that shows the details of our project, as well as our contribution

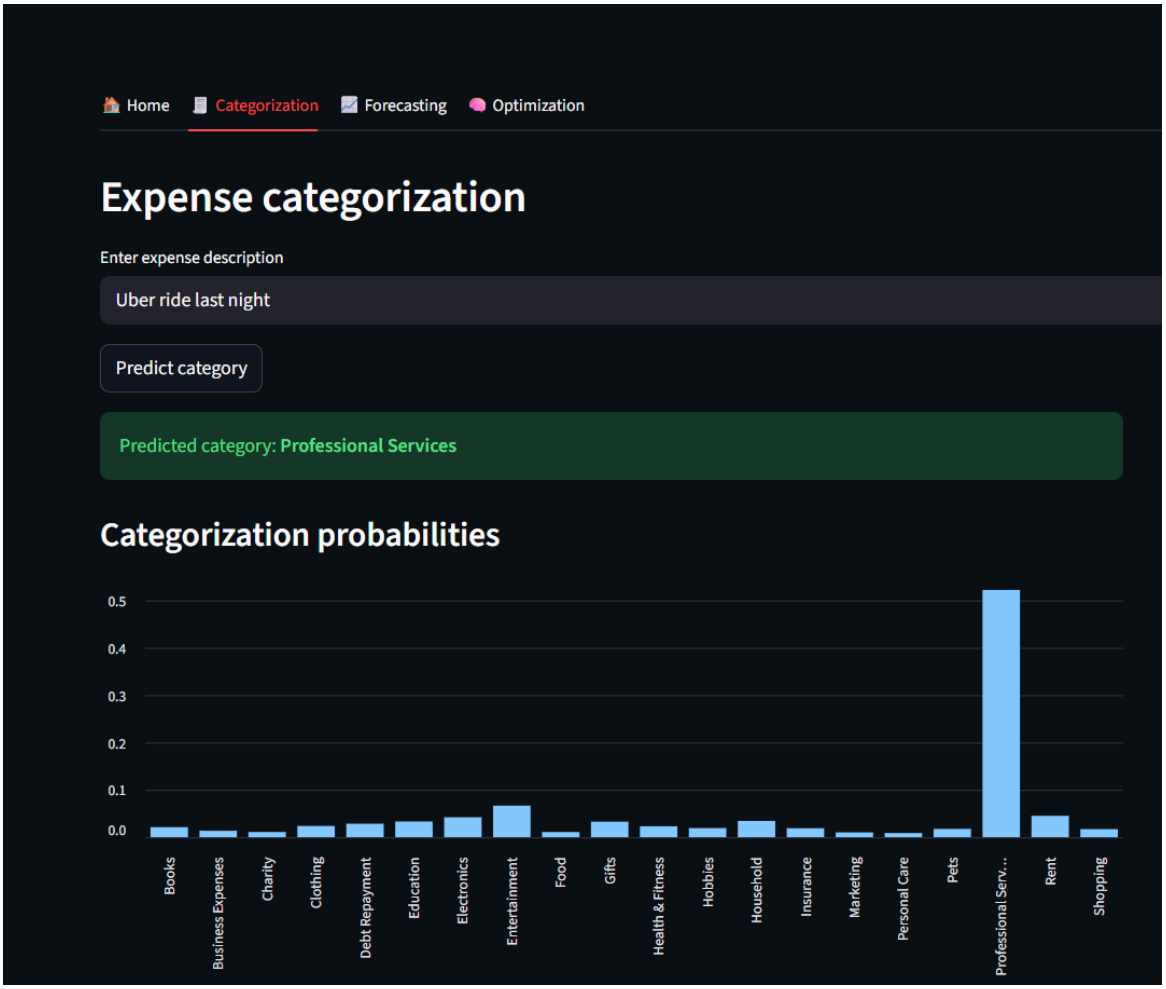


Figure 5.1.2: Expense Categorization From Text

Figure 5.1.2 is an image of the Expense Categorization output, given an expense description, with probabilities of other classes plotted for better inference.

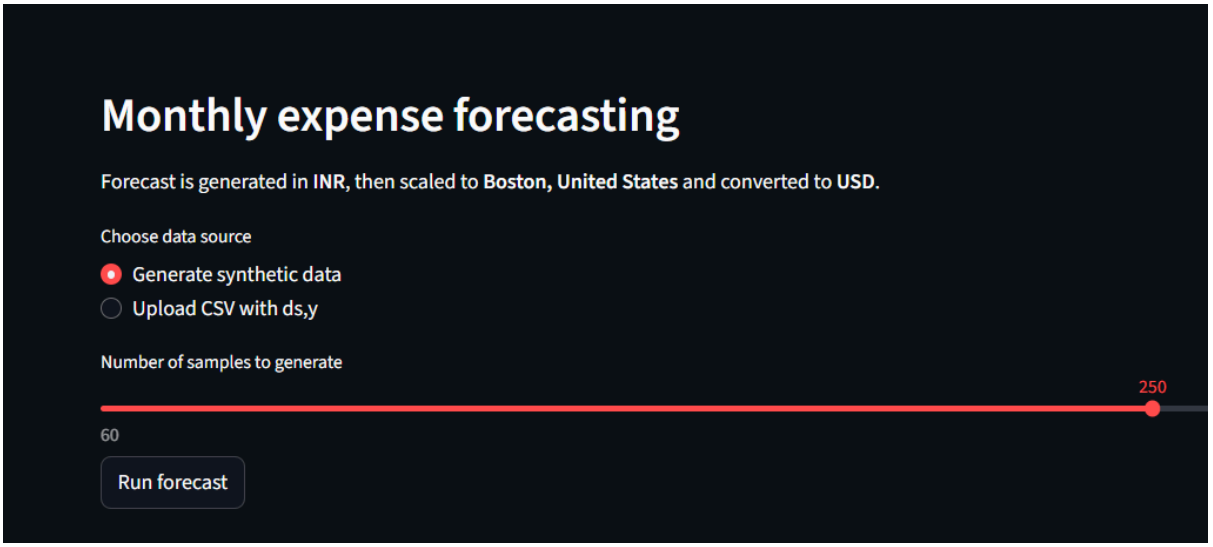
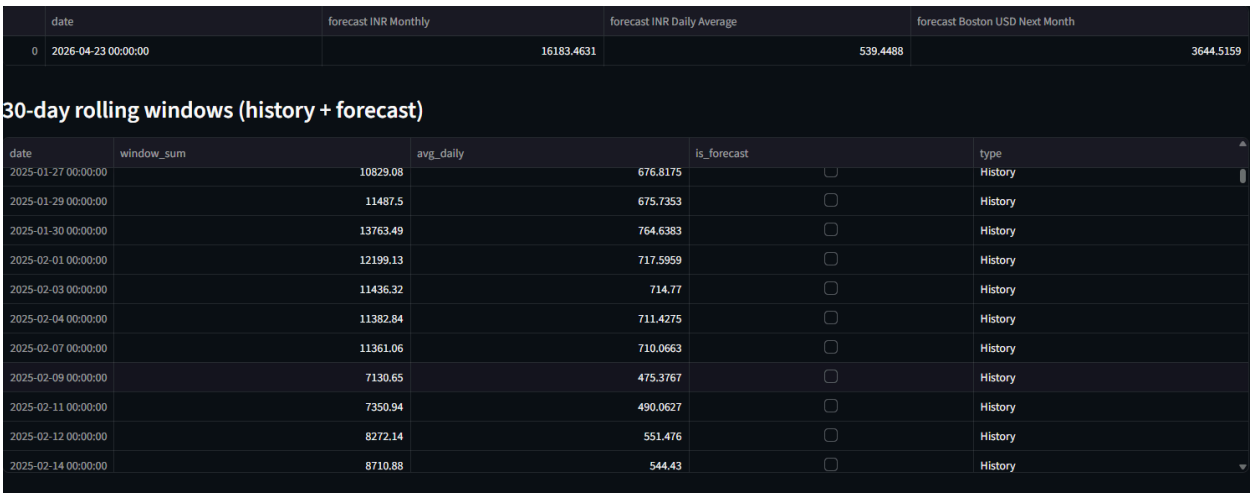


Figure 5.1.3: Monthly Forecast Input Options

The *Figure 5.1.3* is an image of the Forecasting tab, where the user has the option to generate Synthetic data or upload a personal CSV in the ds(date) and y(expense) format



	date	forecast INR Monthly	forecast INR Daily Average	forecast Boston USD Next Month
0	2026-04-23 00:00:00	16183.4631	539.4488	3644.5159

30-day rolling windows (history + forecast)

date	window_sum	avg_daily	is_forecast	type
2025-01-27 00:00:00	10829.08	676.8175	☐	History
2025-01-29 00:00:00	11487.5	675.7353	☐	History
2025-01-30 00:00:00	13763.49	764.6383	☐	History
2025-02-01 00:00:00	12199.13	717.5959	☐	History
2025-02-03 00:00:00	11436.32	714.77	☐	History
2025-02-04 00:00:00	11382.84	711.4275	☐	History
2025-02-07 00:00:00	11361.06	710.0663	☐	History
2025-02-09 00:00:00	7130.65	475.3767	☐	History
2025-02-11 00:00:00	7350.94	490.0627	☐	History
2025-02-12 00:00:00	8272.14	551.476	☐	History
2025-02-14 00:00:00	8710.88	544.43	☐	History

Figure 5.1.4: Synthetic Dataset Generated

Figure 5.1.3 is an image of the Forecasting over a dataset, as one can see at the upper part of the image, with total expense per month in INR and USD, which is the desired currency in this case, and the city is selected to be Boston by default.

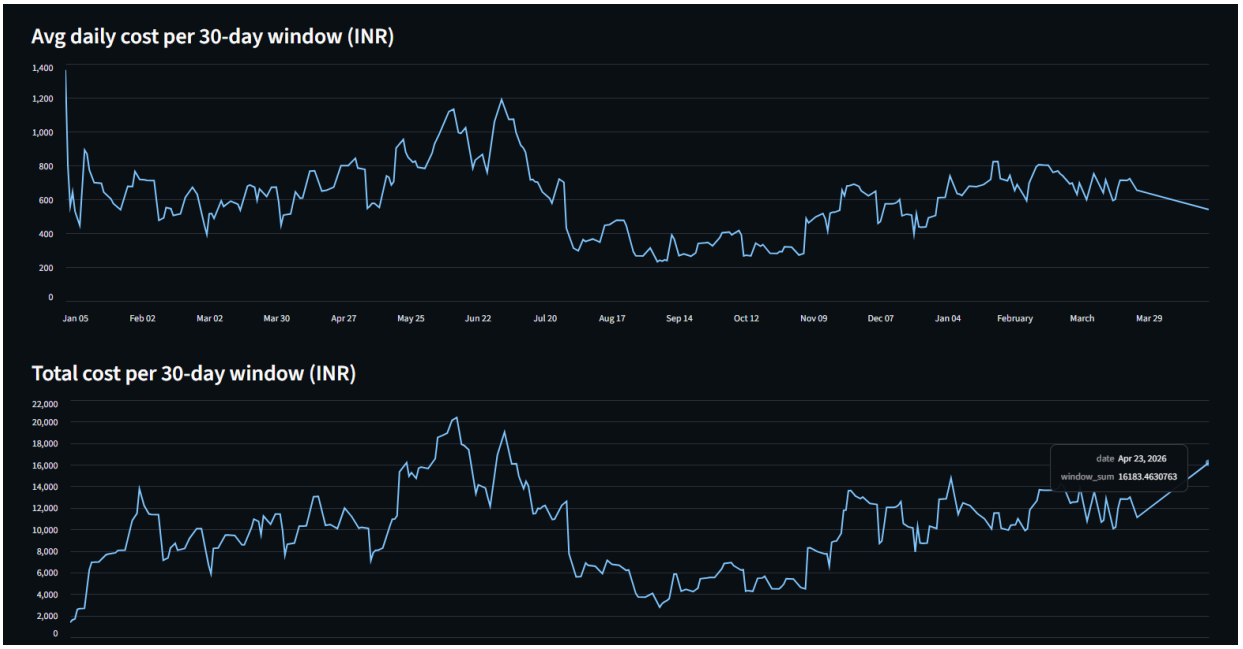


Figure 5.1.6:Visualizing forecast with historical data

Figure 5.1.3 is an image in the Forecasting tab, where the user can visualize their historical expenses as well as the next month's forecast predicted by the model. Although it is in INR, we will make it dynamic in the upcoming versions.

5.2 Categorization Model

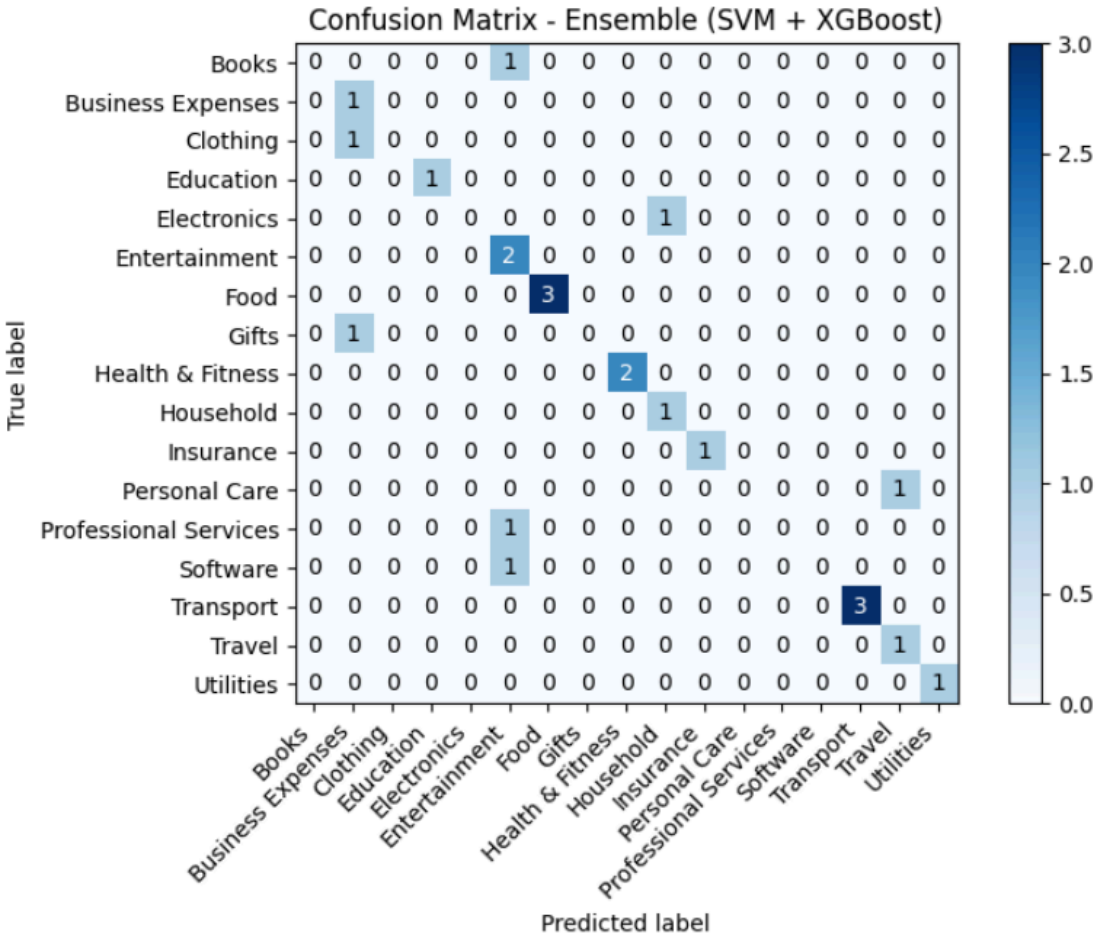


Figure 5.2: Confusion Matrix

To evaluate the performance of our categorization system, we trained three models — **SVM**, **XGBoost**, and a combined **Ensemble model**. Accuracy scores are as follows:

Model	Accuracy
SVM	**90.0 %**
XGBoost	**86.96 %**
Ensemble (SVM + XGBoost)	**89.13 %**

Although SVM performed slightly better individually, the **ensemble helps reduce model bias and increases stability across categories**, especially for medium-frequency transaction types

5.2.1 Confusion Matrix Interpretation

The confusion matrix visualizes how well the model classified each expense category. Darker cells represent higher correct predictions.

Observed Strengths:

- Strong classification performance for Food, Entertainment, Transport, and Utilities
- These categories have more training samples → better learned patterns
- Very few false positives for unrelated classes → good category separation

Misclassification Trends:

- Some overlap occurs in Books ↔ Business Expenses, Travel ↔ Transport, and Education ↔ Personal Care
- These are semantically related categories where description wording can appear similar
- Example: "Book store subscription" might resemble "Online software purchase"

Reason for Errors:

- Short descriptions lack context (e.g., "Amazon Order" could be Clothing, Electronics, or Gifts)
- Class imbalance — fewer samples in lower-frequency categories lead to weaker representation

Key Insights from Results

- Model handles common spend categories reliably
- Embeddings significantly improve performance on short/noisy text
- Ensemble is more stable than single models in edge cases
- Performance is promising given the small dataset size

The categorization engine performs well overall, achieving close to **90% accuracy**, with particularly strong predictions in high-frequency categories. Remaining errors largely stem from class similarity and lack of descriptive context — challenges that can be improved with more data, additional features (amount, merchant name), or contextual augmentation.

5.3 Forecasting Model

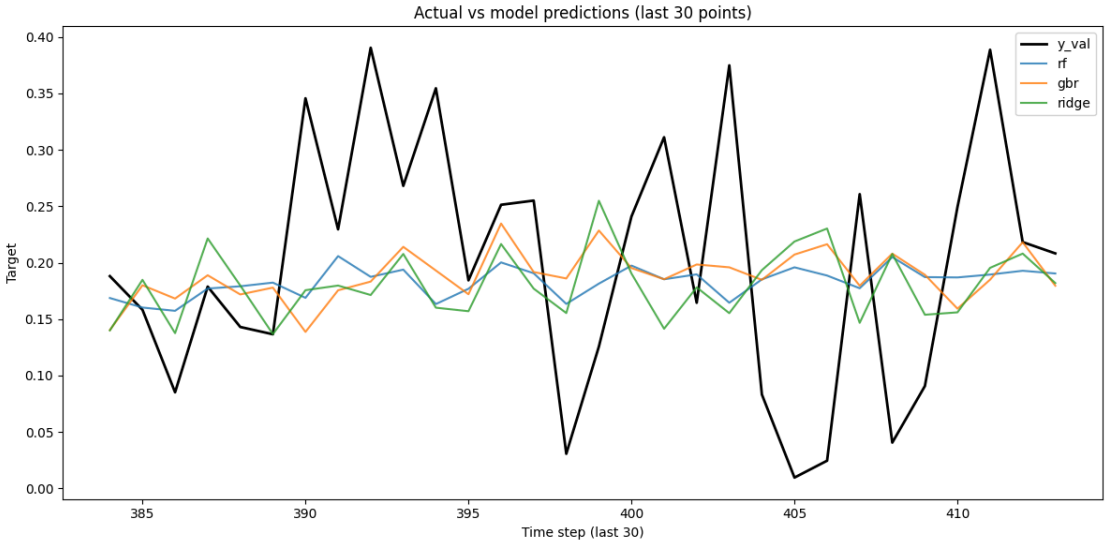


Figure 5.3.1: Ensemble model prediction on the Validation set

Figure 5.3.1 clearly shows the different predictions of Models on the Validation dataset, later used together to create an ensemble model

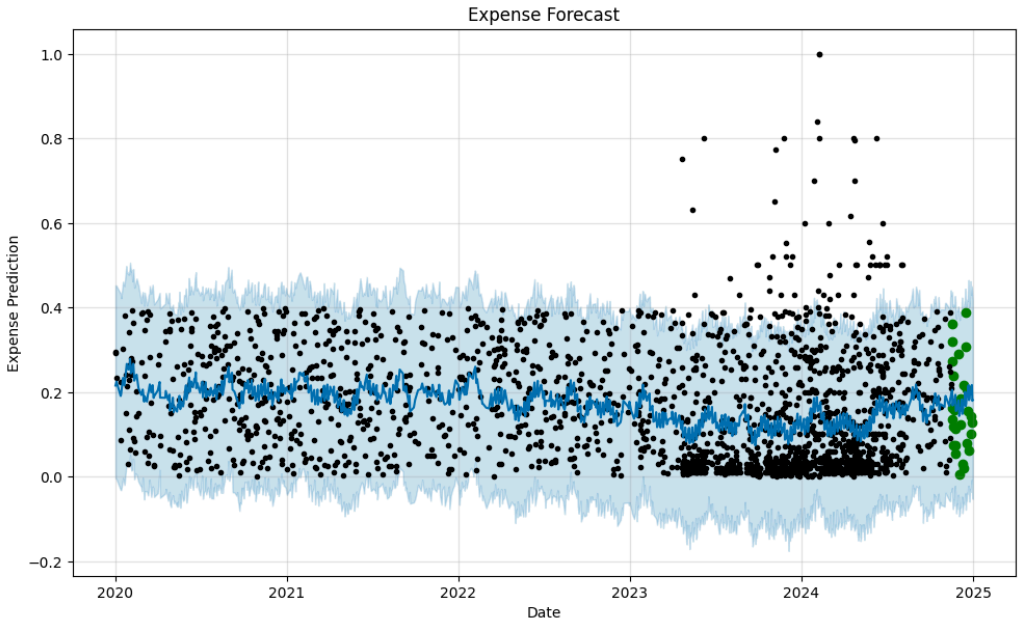


Figure 5.3.2: Prophet Forecast

The above *Figure 5.3.2* showcases the Prophet model's prediction of expenses for the next 45 Days.

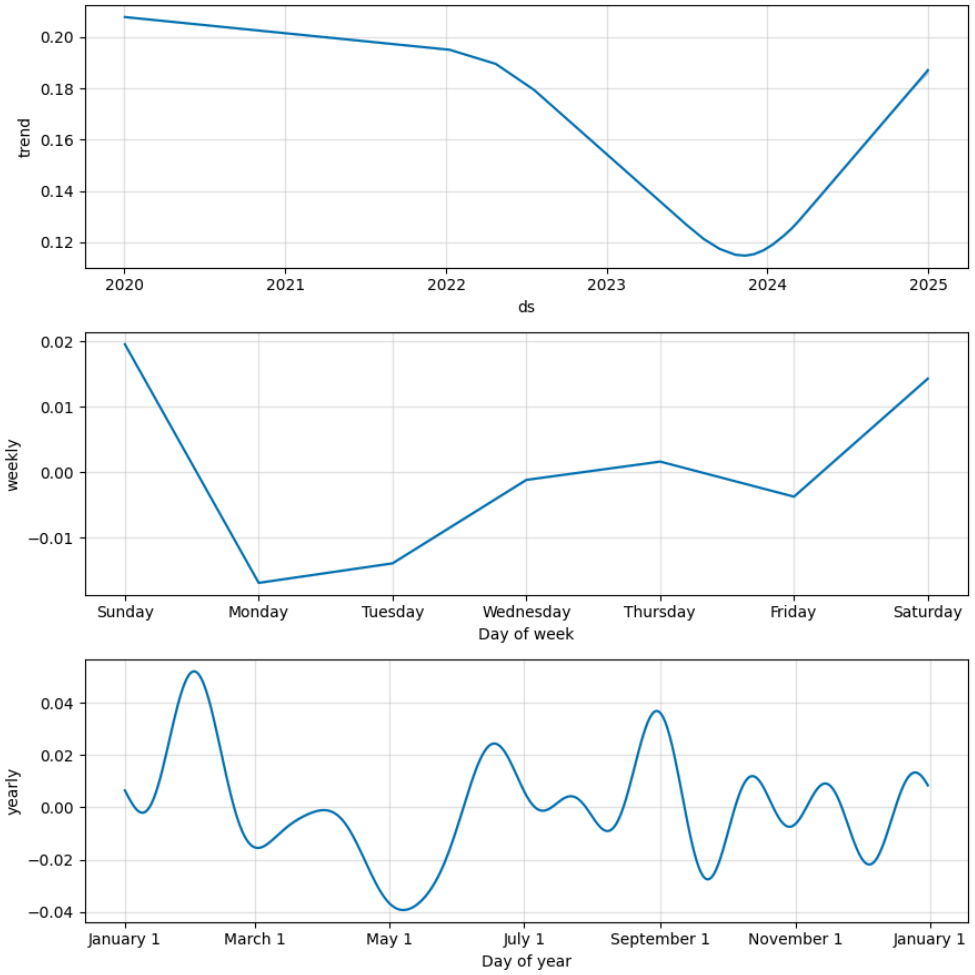


Figure 5.3.3: Prophet Model Additional Features

The above *Figure 5.3.3* shows the underlying trends in the data over the training dataset, as well as the 45-day forecast at the end. This shows the seasonality in the data weekly and yearly. Therefore, these features were additionally used in the ensemble model.

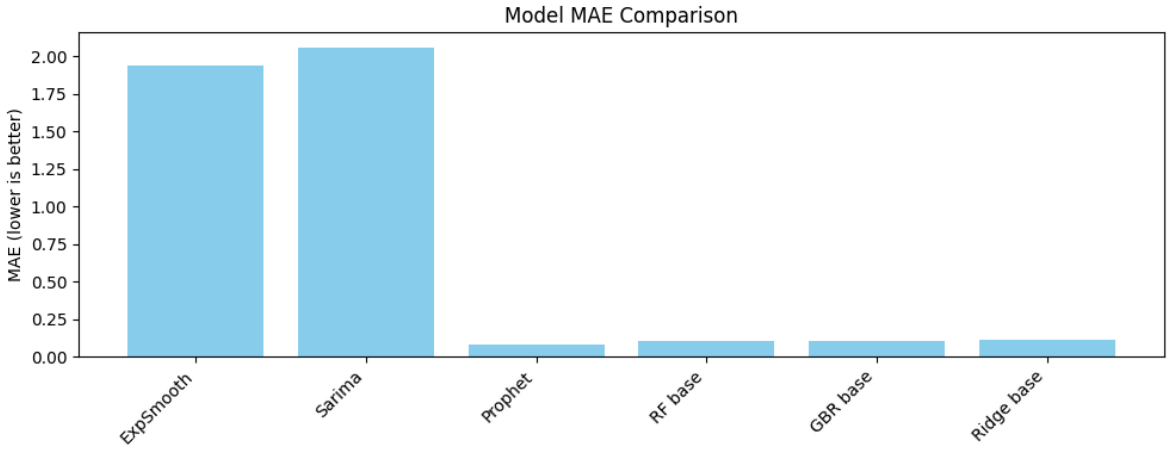


Figure 5.3.4: Comparative MAE between models

Figure 5.3.4 showcases the MAE of various Statistical and ML models. This comparative study helped us to pick the ensemble approach as ML models like Random Forest, Gradient Boosting Regressor, and Ridge Regression have decent performance.

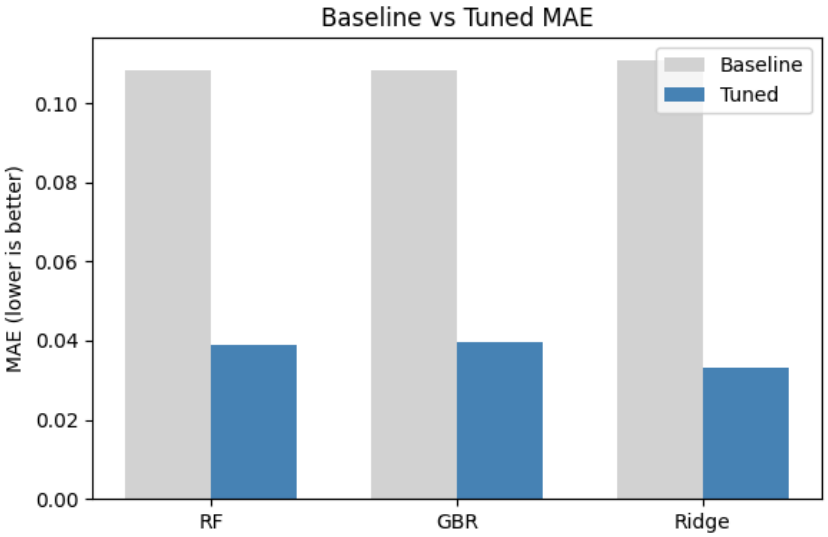


Figure 5.3.5: Prophet Model Additional Features

The above *Figure 5.3.5* showcases the MAE of various ML models. When the additional rolling means of 7, 14, and 30 (statistical measures) are used with extra features like the Day of Week, Month, and Weekend Flag. Helped to reduce the MAE.

5.4 Optimization Outcome

This module produced a “decision-ready” financial snapshot by converting raw transactions and model forecasts into compact, optimization-relevant signals. Instead of passing the entire transaction history to the LLM, the system distilled the user’s recent behavior into (i) core aggregates (income/expense/net), (ii) temporal stability metrics (trend and volatility of day-level net), (iii) category concentration indicators (top categories and their expense share), and (iv) forecast summaries (mean level and directional pressure). This feature compression reduced prompt noise while preserving the specific factors that drive actionable budget decisions.

EDA outcomes and behavioral insights

Across the analyzed window, the aggregate metrics immediately exposed whether the user was operating under a surplus or deficit regime (via Total Net and the expense/income ratio r). The temporal analysis provided a second layer of explanation: even when the average net was acceptable, high day-to-day volatility and large expense spikes highlighted instability (e.g., isolated high-spend days or irregular vendor patterns). Category decomposition then localized the source of spending pressure by ranking expense totals per category and measuring concentration (i.e., whether a small number of categories dominated total expenses). In practice, this meant the system could distinguish between:

structural overspending ($r > 1$ and sustained negative net),
instability-driven risk (high volatility with spike days),
and concentration risk (a few categories driving most of the expense mass).

Forecast-aware interpretation (bridging current behavior to near-term risk)

The forecast summary features provided a forward-looking layer that complemented retrospective EDA. The forecast mean acted as an “expected spend level” over the next horizon, while the forecast trend (up/down/flat) functioned as an early warning for rising spending pressure. This was particularly useful when EDA already flagged deficit behavior: an upward forecast trend in the presence of negative net strengthened the case for conservative caps and stricter prioritization rules. Conversely, a flat/down forecast trend with improving net supported lighter-touch optimizations focused on smoothing spikes rather than aggressive cuts.

Optimization signals translated into concrete targets

The engineered signals were mapped into explicit optimization targets that the downstream LLM could act on reliably:

- Overspending signal (negative net and/or r > 1) → recommend enforceable category caps and reduce discretionary allocation first.
- Instability signal (high volatility, spike days) → recommend "spike control" tactics: vendor-level alerts, weekly ceilings, and smoothing rules (e.g., splitting large purchases, limiting ad-hoc spends).
- Concentration signal (top categories dominate expense share) → prioritize the top contributors and propose category-specific reduction strategies rather than broad, generic advice.
- Forecast risk signal (upward forecast trend) → tighten near-term caps and emphasize proactive controls for categories driving the trend.

LLM-guided budgeting results (actionability + groundedness)
With the prompt constrained to computed features, the LLM consistently produced recommendations that were:

- grounded in the provided metrics (explicitly referencing net, ratio r, volatility, top categories, and forecast direction),
- category-specific (caps and reduction tactics tailored to the top spend categories rather than generic "save more" advice),
- and prioritization-aware (sequencing actions: essentials preserved, discretionary tightened, then smoothing controls for volatility).

In addition, because anomalies (largest expense day, most frequent vendor/category) were injected directly, the LLM could explain "why" a recommendation was being made (e.g., identifying spending spikes as a primary driver versus slow drift in baseline expenses). This improved interpretability and made the optimization plan easier to justify in the UI.

Practical evaluation (how this module was validated)
This module was validated using feasibility and impact checks that do not require model retraining:

- Budget feasibility check: recommended caps were verified to keep projected expenses within an income-consistent envelope (relative to Total Income and recent Avg Daily Net).
- Counterfactual savings simulation: applying the suggested category caps to historical spends estimated the reduction in total expenses and the resulting shift in net balance (i.e., "if these caps had been applied, the net would move closer to / become positive").

- Stability assessment: recommendations targeting volatility were evaluated by whether they reduced the influence of spike days (e.g., shifting spend away from single-day extremes and toward smoother weekly behavior).
- Output consistency: responses were checked for strict adherence to "cite-the-metrics" constraints (i.e., recommendations should reference the injected values, not invent new ones).

Net effect
Overall, the EDA + LLM optimization layer converted transaction logs into a compact set of interpretable financial "constraints and levers," then generated budget actions that were both personalized and constraint-aware. The key outcome was not only better recommendations, but recommendations that were easier to trust: they were traceable to measurable signals (net, ratio, volatility, concentration, forecast trend) and could be validated via simple counterfactual simulations before being surfaced to the user.

6. Discussion

This project successfully implements an **end-to-end AI-driven personal finance assistant pipeline**, covering expense ingestion, categorization, forecasting, and budgeting insights. Key accomplishments include:

- **Developed a complete categorization engine** using MiniLM semantic embeddings combined with SVM + XGBoost Ensemble, achieving high accuracy even with limited data.
- **Built financial forecasting models** capable of predicting future spending patterns using statistical and ML approaches.
- **Demonstrated semantic understanding** of short and noisy transaction descriptions, outperforming traditional keyword/TF-IDF methods.
- **Integrated modular architecture** enabling plug-and-play extension for email parsing, dashboard UI, and optimization modules.
- **Established workflow foundation** for Agentic budgeting using LLM decision reasoning.

6.1 Challenges Faced

During system development, several technical and data constraints were observed:

- **Class Imbalance:** Rare categories contained very few samples, leading to misclassification and variable accuracy.

- **Ambiguous & short text descriptions:** Many transaction lines lacked context, making categorization difficult without semantic models.
- **Global financial variance:** Currency handling, dynamic pricing, and region-specific expenses require adaptive scaling.
- **Standardization of data sources:** Email, UPI alerts, card statements, and receipts follow non-uniform formats, requiring a **unified schema**.
- **Forecast → budget integration gap:** Forecasting values need mapping to budgeting actions (limits, thresholds, alerts) for full automation.

6.2 Future Work

To extend the system into a fully autonomous Agentic Personal Finance Assistant, the following planned upgrades are proposed:

- **Multilingual + multi-source support** for global transaction formats and OCR-based receipt reading.
- **Deep learning forecasters (LSTM / Prophet Meta models)**
  Improve long-term prediction stability.
- **Probabilistic spending projections (Monte Carlo simulation)**
  Estimate risk, cashflow runway, and future burn rate under uncertainty.
  Proactive savings recommendations using LLM reasoning
  Example: "Reduce dining-out spend by 15% this month to save ₹X."
- **Reinforcement Learning-based budget optimizer**
  Agent learns user behavior → self-improves recommendations over time.

References

1. Kaggle Financial Datasets

2. Sentence-BERT MiniLM — Reimers & Gurevych, 2019

3. XGBoost — Chen & Guestrin, 2016
4. Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.*

5. Jurafsky, D., & Martin, J. (2021). *Speech and Language Processing.* Prentice Hall.

6. Gupta, S., Verma, A. (2020). *Rule-Based Financial Categorization and Its Limitations.* IEEE Transactions on FinTech.

7. Box, G. E., Jenkins, G. (2015). *Time Series Analysis: Forecasting and Control.* Wiley.

8. Makridakis, S. et al. (2020). *M4 Forecasting Competition Results.* International Journal of Forecasting.

9. Holt, C. (1957). Forecasting Trends and Seasonal Models. Carnegie Institute of Technology.

10. De Zarzà, I., De Curtò, J., Roig, G., & Calafate, C. T. (2024). Optimized financial planning: Integrating individual and cooperative budgeting models with LLM recommendations. AI, 5(1), 91–114. https://doi.org/10.3390/ai5010006

11. Lee, Y., Kim, Y., Kim, J., Kim, S., & Lee, Y. (2025). LLM-enhanced Black–Litterman portfolio optimization (arXiv:2504.14345). arXiv. https://doi.org/10.48550/arXiv.2504.14345

12. Wu, S., Irsoy, O., Lu, S., Dabravolski, V., Dredze, M., Gehrmann, S., Kambadur, P., Rosenberg, D., & Mann, G. (2023). BloombergGPT: A large language model for finance (arXiv:2303.17564). arXiv. https://doi.org/10.48550/arXiv.2303.17564

13. Yang, H., Liu, X.-Y., & Wang, C. D. (2023). FinGPT: Open-source financial large language models (arXiv:2306.06031). arXiv. https://doi.org/10.48550/arXiv.2306.06031

14. The Alan Turing Institute, & HSBC. (2024, March). The impact of large language models in finance: Towards trustworthy adoption (FAIR—Framework for Responsible Adoption of Artificial Intelligence in the Financial Services Industry report). The Alan Turing Institute. https://www.turing.ac.uk/news/publications/impact-large-language-models-finance-towards-trustworthy-adoption

15. https://zerotomastery.io/blog/time-series-forecasting-with-facebook-prophet/

16. https://www.geeksforgeeks.org/machine-learning/a-comprehensive-guide-to-ensemble-learning/

Appendix:

A. Repository layout

The project is organized as a Python package with separate modules for categorization, forecasting, Optimization, Data, Email Categorizer, Email Parser, Data Preprocessing module, Categorization Models, Categorization Embeddings, forecasting Model, Steamlit App.py, and Demo.py.

1. **Root**
    a. **demo.py** executes the Streamlit command on the terminal to create an app.
    b. **app.py**
    Streamlit entrypoint that exposes three main tabs:
        i. Home: project overview and configuration.
        ii. Categorization: NLP expense categorizer using MiniLM + SVM/XGBoost ensemble.
        iii. Forecasting: time-series forecasting demo with distribution plots and forecasting graphs (Synthetic Data created by bootstrapping by default).
        iv. Optimization: simple Q&A interface over combined history and forecast data frame (Synthetic Data by Default).
    c. **requirements.txt**
    Python dependencies for local installation and deployment.
    d. **README.md**
    High-level project description and quick usage notes.
2. **Categorization/**
    a. categorizer.py
        i. Loads Sentence-BERT MiniLM embedding model and ML artifacts (SVM, XGBoost, scaler, label encoder) once at import time.
        ii. **Categorization(text) → (label, prob_array, class_labels)** returns:
            1. Predicted expense category label.
            2. Probability vector over all categories.
            3. Ordered list of class labels for plotting.
        iii. **predict_category(text)** helper returning only the label.
        iv. **predict_bulk(list_of_expenses)** for batch predictions.
    b. Categorization_ipynb

i. EDA on Categorization Data
ii. Data Preprocessing
iii. Fitting Embeddings using Data
iv. Training Baseline Models and comparing results
v. Saving the embeddings and the model's pickle files in **Categorization_embeddings/** and **Categorization_model/** directories

3. Categorization_model/
    a. **label_encoder.pkl** – Stores the mapping between expense category names and numerical labels used during model training and inference.
    b. **scaler.pkl** – Contains the feature scaling parameters applied to embeddings before classification.
    c. **svm_classifier.pkl** – Trained Support Vector Machine model used for expense category prediction.
    d. **xgb_classifier.pkl** – Trained XGBoost model that predicts expense categories based on semantic embeddings.
4. **Forecasting/**
    a. Model.py
        i. Builds sliding 60-day windows and engineered features (rolling means/std over 7/14/30 days, calendar features) from dated expenses.
        ii. Uses an ensemble of regressors (Random Forest, Gradient Boosting, Ridge) on standardized inputs to predict the average expense over the next 30 days.
        iii. **Generate_sample(...)** utilities to generate synthetic expense time-series (beta-like distributions, irregular dates).
        iv. **Predict(user_expenses, user_dates)** returns a denormalized forecast for the next 30-day horizon using hypothesis-testing-based distribution selection.
        v. **Predict_Model(normalzized_data,user_dates, WINDOW, HORIZON, chosen)** is a parent function used to increase modularity.
        vi. **forecast_next_30_days(recent_values, recent_dates, window=60, horizon=30)** is a function used to calculate statistical and other important features from the data, which is passed to the ensemble model as an input.
        vii. **Load_Model(config_path = './model/expense_config_2.pkl')** is used to load the ensemble model from the model/ directory
        viii. **normalize_d1(x) and normalize_d2(x)** are two functions used to normalize unseen user sequences based on Statistics.

ix. **denormalize_d1(x) and denormalize_d2(x)** are two functions used to denormalize unseen user sequences based on Statistics.

b. conversion.py
   i. **city(city_name, country_name, output_currency)** returns a scaling factor to adjust the forecast from base INR to a target city/currency using simple heuristics.
   ii. **Convert(response, amount, User_output='USD')** returns conversion factor between currencies
   iii. **def API_Call_Conversion(amount,desired_currency)** is used to get output from the API key

c. city_index.py
   i. **fetch_city_data(city_name: str, country_name: str)** returns the API response for the basic costs of that city
   ii. **get_city_basket_cost_usd(city_data: dict)** returns the basic cost of living of selected requirements required for computing the generalized scaling factor.
   iii. **get_city_basket_cost_inr(city_data: dict, currency_conversion_factor) :** It is used for getting the basket cost in India, as it is the default output by the model.
   iv. **get_scale_factor(mumbai_data: dict, target_data: dict,** currency_scale returns the comparative scale for Cost of living to scale output.

d. **main.py** used for testing purposes of the forecasting module
e. **Ts.ipynb** used to train, test, and compare different ML models
f. **API_Config.py** used for fetching API keys from **Secrets.yaml**

5. **Optimization**
   a. **API_config.py:** To load the Gemini API required for a backup for Optimization in the Streamlit demo app.
   b. **Optimize.py:** For Demo Functionality
      i. **_build_expense_summary ():** Summarizes a bunch of Data, which will be passed to the model.
      ii. **_to_plain_text(markdown_text: str):** Post-processing the output for a clean representation.
      iii. **optimize_answer(df_context, question: str):** Main function used to interact with Gemini.

6. **model/**
   a. **Expense_config.pkl** is a config file (pickle file) for the Baseline Ensemble Model
   b. **Expense_config.pkl** is a config file (pickle file) for the Final Ensemble Model

c. Expense_models.pkl is a pickle file for storing weights Baseline Ensemble Model
d. **Expense_models_2.pkl** is a pickle file for storing weights Baseline Ensemble Model
e. **Prophet_model.pkl** is a pickle file for the trained Prophet model

7. **Data Preprocessing/**
   a. **Data_Preprocessing.ipynb** is used for preprocessing the raw datasets into a desirable format to train the models.
   b. **EDA.ipynb** is a Python notebook for executing code bits to learn more about the data and its distributions.
8. **data/**
   a. **Expense_data.csv** for categorization
   b. **Personal_Finance_Dataset.csv** for forecast
   c. **Expense_income_summary.csv** for forecast
   d. **Expense_forecast_dataset.csv** (synthetic but did not use)
   e. **Prepared_data/** Preprocessed Datasets used for modelling
      i. **Aggregated_Dataset.csv**
      ii. **Combined_TS_Dataset.csv**
      iii. **Forecasting_Master_Dataset.csv**
9. **email_category/**
   a. **APIConfig.py** used for fetching API keys from **Secrets.yaml**
   b. **Categorize_email.py**
      i. **categorize_email(text)** Find the transaction amount and categorize the purchase
      ii. **TextExtraction()** Extract Text from emails stored
   c. **preprocessing.py** Various functions to preprocess data fetched from Raw emails into a desirable or presentable format.
10. **email_parser/**
    a. **Parse_email.py**Extracts relevant emails based on subject and body keywords and stores them in a format that can be used by the LLM to categorize

B. Local setup

1. **Clone the repository**

   git clone https://github.com/lune07/Expense-Categorization.git

2. **Create and activate a virtual environment**

   python -m venv .venv
   # Windows
   .venv\Scripts\activate
   # macOS / Linux
   source .venv/bin/activate

3. **Install dependencies**

   pip install --upgrade pip
   pip install -r requirements.txt

4. **Check model paths**
   Ensure the following files exist under the expected directories
   - Categorization_model/svm_classifier.pkl
   - Categorization_model/xgb_classifier.pkl
   - Categorization_model/label_encoder.pkl
   - Categorization_model/scaler.pkl
   - Forecasting ensemble and scaler artifacts.

5. **Secrets.Yaml**
   Ensure the following API keys are added after creating Secrets.yaml
   - **Gmail_Paskey : 'cndy upgg wphm icvu'**
   - **Gemini_API_key : AIzaSyBg4mV1sIZn0VlUtiaazTtXxIXr3naUo8o**
     Currency_API_key : 8401a4bcdecb7fa57a417007042e42b661e0
   - **RAPIDAPI_KEY: :**
     aea1c9d920mshc5251b8f872c9e1p1a5a26jsn3106a948362e

C. Running the Streamlit demo

1. **Check if the Virtual environment is activated :**
2. **If the Environment is not active**

a. *# Windows*
   venv\Scripts\activate
b. *# macOS / Linux*
   source .venv/bin/activate

3. **Start the app**
   From the project root:
   python demo.py

   This launches a local Streamlit server and opens the demo in your browser (default http://localhost:8501).

4. **Navigating the app**
   a. **Home tab**
      i. Overview of Mint Sage, problem statement, and high-level architecture.
      ii. Displays the currently selected city and output currency.
   b. **Categorization tab**
      i. Input: free-text expense description.
      ii. Output:
         1. Predicted category label.
         2. Probability bar chart over all categories.
         3. Run metrics (latency, CPU, memory) and optional intermediate NLP details.
   c. **Forecasting tab**
      i. Data source:
         1. Generate synthetic data via slider (n_samples) **or**
         2. Upload CSV with ds (date) and y (values, min 60 rows).
      ii. Pipeline:
         1. Normalizes sequence to one of two reference beta-like datasets via the KS-test.
         2. Builds sliding 60-day windows with rolling statistics and calendar features.
         3. Ensemble model outputs the next 30-day average in INR, scaled to the selected city/currency.
      iii. UI:
         1. Beta-fit plots for two reference datasets plus sample history.
         2. Rolling or monthly aggregates (sum and avg daily) with the forecast window appended.
         3. Performance metrics.

d. **Optimization tab**
    i. Data source:
        1. Generate a synthetic date, expense, category, or is_forecast DataFrame **or**
        2. Upload CSV with the same schema.
    ii. Shows combined history + forecast table, quick stats, and a Q&A interface:
        1. Question text box triggers optimize_answer(df_context, question).
        2. The answer text box displays the textual optimization suggestion.