

HappyFest: An AI-Assisted Web Platform for Event Planning, Vendor Recommendation, and Budget Allocation Using Random Forest and Sentiment Analysis

Mudavath Naveen¹, Katroth Balakrishna Maruthiram²

¹MCA Student, Department of Computer Science

²Assistant Professor – Guide, Department of Computer Science

University College of Engineering, Science & Technology,

Jawaharlal Nehru Technological University Hyderabad, Hyderabad, India

Abstract - Event planning is traditionally a manual, fragmented process in which a user must individually discover, compare and coordinate multiple vendors — venues, caterers, decorators, photographers and entertainers — while simultaneously managing an uncertain budget. HappyFest is a role-based web application that brings this process onto a single platform and augments it with machine-learning-driven decision support. Built on the Flask (Python) framework with a relational database backend (MySQL/SQLite), the system lets a user register, create an event profile describing its type, date, location, budget and expected audience size, and instantly receive an intelligent vendor-category recommendation generated by a RandomForestClassifier trained on a curated, historical event–vendor association dataset. Alongside the recommendation engine, HappyFest performs rule-based, event-type-aware budget allocation that splits the stated budget across seven expenditure categories — venue, catering, decoration, photography, entertainment, transportation and miscellaneous expenses — after reserving a contingency reserve, and derives per-guest cost together with AI-generated cost-saving insights and alerts. A vendor marketplace lets users compare and book vendors by category, price, rating and distance, while vendors/organizers manage incoming booking requests through a dedicated dashboard. A feedback module captures user ratings and free-text comments after an event and classifies the sentiment of that feedback (Positive, Neutral or Negative) using a TextBlob-based, lexicon-fallback sentiment-analysis routine, giving administrators a continuous signal of vendor and platform service quality. An administrator dashboard consolidates system-wide analytics — user counts, event counts, vendor counts and feedback trends — for oversight of the platform. The application was implemented and tested end-to-end across registration and authentication, event lifecycle management, ML-based recommendation and budget prediction, vendor booking, feedback and sentiment analysis, and role-based administration for User, Organizer and Admin roles. Fourteen representative system test cases, spanning authentication, recommendation, booking, feedback-sentiment and admin-analytics workflows, all passed on the final build. The resulting system demonstrates that a lightweight, rule-augmented machine-learning pipeline can meaningfully reduce the manual effort and guesswork involved in event planning while remaining fast, explainable and inexpensive to run — making it a practical, extensible foundation for further enhancement with deep-learning-based recommendation and NLP models such as BERT.

Key Words: *Event Management, Machine Learning, Random Forest Classifier, Vendor Recommendation, Budget Allocation, Sentiment Analysis, TextBlob, Flask, MySQL, Web Application.*

1. INTRODUCTION

Planning an event — a wedding, a birthday celebration, a conference, a college fest, a corporate function, a concert or an anniversary — currently involves searching for vendors across multiple disconnected sources, manually comparing prices, ratings and availability, and estimating a budget largely through guesswork or word-of-mouth advice. This process is time-consuming and error-prone, and it offers little to no personalisation based on the type of event, the number of guests or the budget actually available to the host. Existing online event-listing platforms largely digitise vendor directories but stop short of offering any real decision support: they do not recommend a vendor category suited

to a user's specific event, do not predict how a budget should be split across expenditure categories, and do not use feedback collected after an event to improve future recommendations.

Machine learning offers a practical way to close this gap. A supervised classifier trained on historical associations between event parameters and the vendor categories that were actually prioritised for similar events can generate a personalised, explainable recommendation in well under a second, without requiring the heavy infrastructure of a deep-learning pipeline. Likewise, a transparent, rule-based percentage split — tuned by event type — can turn a single budget figure into an actionable, category-wise plan, while a lightweight sentiment-analysis routine can convert free-text post-event feedback into a structured quality signal for administrators and vendors alike, without the cost or latency of a large language model.

This work presents HappyFest, a Flask-based, role-based (User / Organizer / Admin) web application that combines these three lightweight machine-learning and rule-based components — a RandomForestClassifier-driven vendor-category recommender, an event-type-aware budget allocator, and a TextBlob/lexicon-based feedback sentiment classifier — with a conventional event-management and vendor-marketplace workflow. The system is deliberately scoped for individual and small-institutional event planning rather than large-scale event-discovery or ticketing platforms: it focuses on helping a single user plan a single event, from the first budget estimate through to post-event feedback.

The specific objectives of this work are threefold: (i) build a structured event, vendor, booking and feedback data layer in a relational database (MySQL/SQLite) that supports the full life cycle of an event from creation to post-event review; (ii) train and expose a RandomForestClassifier-based recommendation function, together with a rule-based, event-type-aware budget-allocation engine, so that a user creating an event immediately receives a recommended vendor category, a confidence score and a category-wise budget breakdown; and (iii) integrate a TextBlob-based sentiment-analysis routine into the feedback workflow and assemble all of these components into a single Flask web application with dedicated User, Organizer and Admin views, so that the platform delivers decision support at every stage of planning — not only at the point of vendor discovery.

In summary, this paper makes three contributions. First, it designs and trains a RandomForestClassifier-based vendor-category recommender on a deliberately compact, curated event–vendor dataset, and shows that a lightweight, fast-training model is sufficient to produce a useful, explainable recommendation for a small platform with limited historical data. Second, it designs a rule-based, event-type-aware budget-allocation engine that turns a single budget figure into a seven-category breakdown, a per-guest cost and AI-generated cost-saving insights, addressing a gap that recurs across the event-recommendation literature reviewed in Section 2. Third, it integrates a TextBlob/lexicon-based sentiment-analysis routine into a post-event feedback workflow and packages the whole pipeline into a role-based Flask application — covering registration, event management, recommendation, booking, feedback and administration — whose testing (Section 5) confirms that the system behaves as intended end to end, not only at the level of individual module correctness.

2. LITERATURE SURVEY

Horowitz et al. [1] introduced EventAware, a context-aware, tag-based mobile recommender system that personalises the agenda of users attending a congress by matching item tags against a user profile built automatically from a professional-network account; the system was validated through a live case study at a large mobile-technology event, but it is scoped to recommending sessions and exhibitors within a single congress rather than recommending vendors or planning a budget for an event a user is organising.

Varaprasad et al. [2] proposed an intelligent framework for analysing event data and recommending events to users based on their interests, addressing the difficulty an individual faces in discovering and filtering relevant events from a large pool. Their system, like most of the event-recommendation literature, focuses on recommending which events a user should attend rather than assisting the organiser of an event with vendor selection or budget planning — the opposite side of the same problem that HappyFest addresses.

Awad [3] compared machine-learning techniques, including gradient-boosted classifiers, for predicting guest attendance at events and combined attendance prediction with recommendation algorithms to tailor event experiences to individual

guests; the study underscores that data-driven models can uncover patterns in historical event data that are not obvious to human planners, reinforcing the case for a learned recommender over purely manual vendor search, though it targets attendance forecasting for event organisers rather than category-wise vendor recommendation and budget allocation for the person planning the event.

A recent industry system, EventVerse [4], combines a web-based event-planning interface with a vendor-recommendation mechanism built on a cloud backend, and reports that the integration of recommendation logic with real-time updates improves user decision-making relative to a plain vendor directory; however, EventVerse does not report a machine-learning-based recommendation model or a structured budget-allocation engine, and treats machine learning and predictive analytics as a stated direction for future work rather than an implemented component.

Breiman [5] introduced the Random Forest algorithm itself, showing that an ensemble of decision trees trained on bootstrap-sampled data with randomised feature selection at each split achieves accuracy competitive with boosting methods while remaining considerably more robust to noise and requiring little hyperparameter tuning — properties that make it well suited to a small, imbalanced, categorical dataset such as the one used to train HappyFest's vendor-category recommender. Ricci et al. [6], in the Recommender Systems Handbook, survey the broader landscape of recommendation techniques — content-based, collaborative and hybrid — and note that classification-based recommenders trained on structured, tabular context features remain a practical choice when historical interaction data is limited, which is the regime HappyFest's 37-record training set falls into.

On the feedback side, Hazarika et al. [7] applied TextBlob to classify the sentiment of social-media text into positive, negative and neutral categories using its built-in polarity scorer, and found the lexicon-and-pattern-based approach to be simple to deploy and reasonably accurate for short, informal text — the same profile as the free-text comments HappyFest collects after an event. Prasanna Kumar [8] similarly demonstrated TextBlob's polarity- and subjectivity-scoring for customer-feedback analysis in a market-survey setting, concluding that a lexicon-based tool is a computationally inexpensive way to give decision-makers an ongoing sentiment signal, while noting that such tools struggle with sarcasm and domain-specific phrasing — a limitation HappyFest mitigates only partially through its lexicon fallback.

Read together, the literature confirms two recurring gaps that this project targets directly. First, the great majority of event-recommendation systems — EventAware [1], the framework of Varaprasad et al. [2], and the attendance-prediction work of Awad [3] — are built to help an attendee discover or be matched to events, not to help an organiser select vendors and allocate a budget for an event they are planning; HappyFest addresses the organiser-side problem directly. Second, even systems that are explicitly oriented toward event organisers, such as EventVerse [4], stop at vendor discovery and do not couple the recommendation with a structured, explainable budget breakdown or a feedback-driven quality signal, a combination that HappyFest's three cooperating modules — recommendation, budget allocation and sentiment analysis — are designed to close.

3. SYSTEM ANALYSIS

3.1 Existing Approaches and Their Limitations

In the existing manual or semi-digital approach to event planning, a user identifies candidate vendors through word-of-mouth, social media or generic vendor-directory websites, contacts each one individually to check availability and pricing, and arrives at a budget split largely through personal estimation. Some existing web platforms digitise the vendor directory and allow online enquiries, but stop at that: they do not analyse the user's specific event parameters to suggest which category of vendor to prioritise, do not predict a category-wise budget split, and do not use feedback data to improve future planning. Across the platforms reviewed in Section 2 and the manual approach they replace, four gaps recur consistently: no personalised recommendation of vendor category based on event type, budget or audience size; no automatic, event-type-aware budget breakdown; vendor comparison that is manual and spread across multiple disconnected sources; and no structured mechanism to capture and analyse user feedback after an event.

3.2 Feasibility

The proposed system is technically feasible: it is built entirely on mature, open-source components — Python, Flask [9], scikit-learn [10] and either MySQL or SQLite — and the RandomForestClassifier used for vendor-category recommendation trains in well under a second on the project's 37-record dataset and requires no specialised hardware, GPU or paid cloud service. It is economically feasible because every core component is free and open source, with the sole external dependency — the OpenWeatherMap API used for the weather-advisory feature — offering a free tier sufficient for individual or small-institutional deployment. Operationally, the system requires no training to use: creating an event and viewing a recommendation take a few seconds, which keeps the learning curve low for the three target user roles — event planner/user, vendor/organizer and administrator — who are already comfortable with everyday web applications.

Table -1: Existing Systems Versus the Proposed HappyFest System

Aspect	Existing Systems	Proposed HappyFest System
Vendor recommendation	None — manual browsing only	ML-based category recommendation with confidence score
Budget planning	Manual estimation	Rule-based, event-type-aware 7-category breakdown
Vendor comparison	Spread across sources	Single ranked marketplace by price, rating, distance
Feedback handling	Not structured	Captured and automatically sentiment-scored
Administration	No unified dashboard	Centralised analytics for users, events, vendors, feedback

4. PROPOSED SYSTEM AND METHODOLOGY

The proposed system is built around three cooperating intelligence components — a RandomForestClassifier-based vendor-category recommender, a rule-based budget-allocation engine, and a TextBlob/lexicon-based sentiment-analysis routine — layered on top of a conventional event, vendor and booking data model, all served by a Flask web application designed for three distinct user roles.

4.1 System Architecture

HappyFest follows a layered, monolithic web-application architecture. The client browser renders server-generated HTML (Jinja2 templates styled with Bootstrap 5) and communicates with the Flask application layer over standard HTTP. The Flask layer (app.py) owns routing and session management and delegates to three internal modules: an authentication module built on Werkzeug's password-hashing utilities, an ML recommendation engine (recommender.py) that wraps the trained RandomForestClassifier and the rule-based budget engine, and a sentiment-analysis module (sentiment.py). All persistent data access is funnelled through a single data-access layer (db.py) that transparently targets either SQLite or MySQL, and the Flask layer additionally calls the external OpenWeatherMap REST API for the weather-advisory feature. Fig -1 summarises this pipeline.

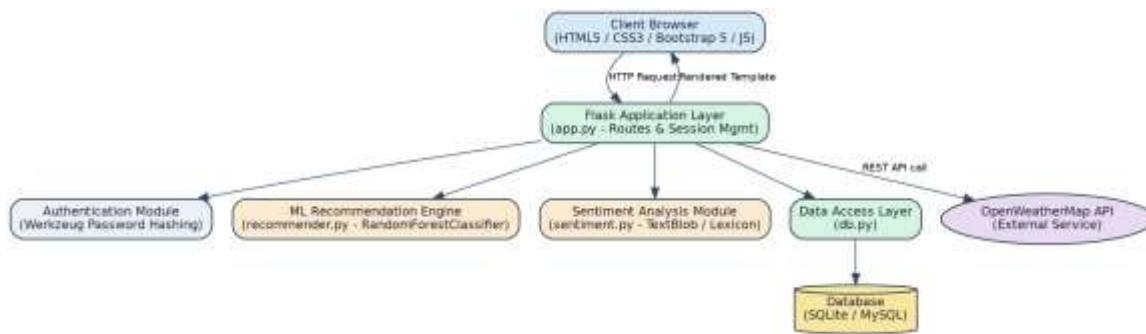


Fig -1: Architecture of the HappyFest Event Management Platform

Fig -2 shows the entity-relationship structure underlying the system: a User places many Events and writes many Feedback entries; an Event receives many Feedback entries and has many Bookings; and a Vendor fulfils many Bookings, optionally linked to an Organizer account. All relationships are one-to-many and enforced through foreign keys with cascading deletes, keeping the dataset consistent as events, vendors or users are removed.

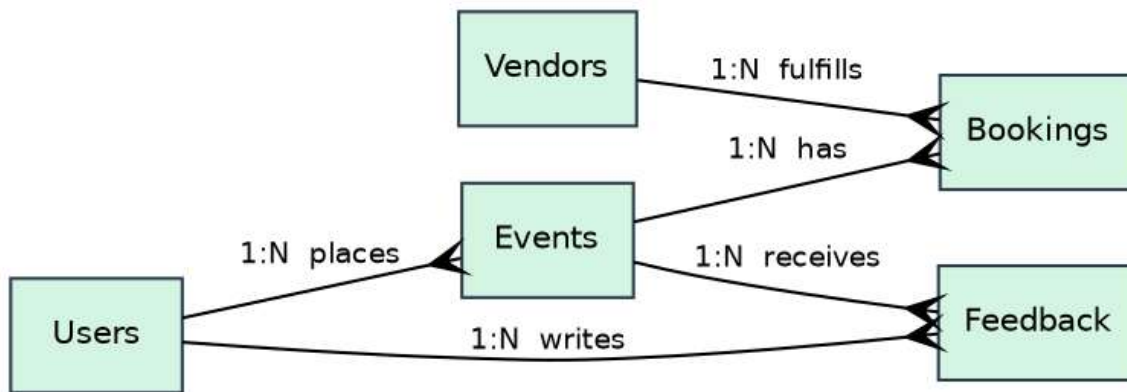


Fig -2: Entity-Relationship Diagram of the HappyFest Database

4.2 Dataset Description

Two datasets drive the system's intelligence. First, the vendor-category recommendation model is trained on a curated, in-application dataset of 37 historical event records spanning seven event types — Wedding, Birthday, Conference, College Fest, Corporate, Concert and Anniversary — each labelled with the budget, audience size and the vendor category that was actually prioritised for that combination (Table -2); this dataset is defined directly in ml/recommender.py and used to fit a RandomForestClassifier at application start-up. Second, the vendor marketplace is seeded with a representative dataset of 15 vendors spanning categories that include venue, catering, decoration, photography, DJ services, entertainment, makeup, transportation and rentals, each with a price range, rating, distance and service description, stored in the vendors table and used both for browsing/booking and as the pool from which recommended vendors are surfaced.

Table -2: Recommendation Training Data — Event Types Covered (N = 37)

Event Type	Labelled Attributes	Historically Prioritised Category
Wedding	Budget, audience size	Venue / Catering (dominant)
Birthday	Budget, audience size	Catering / Decoration
Conference	Budget, audience size	Venue
College Fest	Budget, audience size	Entertainment
Corporate	Budget, audience size	Venue / Catering
Concert	Budget, audience size	Entertainment

Anniversary	Budget, audience size	Catering / Decoration
-------------	-----------------------	-----------------------

4.3 Vendor-Category Recommendation Algorithm

The recommender is trained once at application start-up on the 37-row dataset described above. Categorical fields — principally the event type — are label-encoded, and a scikit-learn RandomForestClassifier composed of 100 decision trees is fit on the encoded event-type, budget and audience-size features against the historically prioritised vendor category. At request time, `get_recommendation()` encodes the incoming event's type — falling back to a partial string match for event types not seen verbatim during training — predicts a vendor-category class, and returns the predicted category together with the model's class probabilities converted to a confidence percentage. Ensembling 100 independently bootstrapped trees, each considering a randomised subset of features at every split, keeps the recommender robust to the class imbalance and small sample size inherent in a newly seeded platform, consistent with the noise-robustness properties reported for Random Forests by Breiman [5].

4.4 Budget Allocation — Rule-Based, Event-Type-Aware Percentage Split

`predict_budget()` first reserves a 12% contingency reserve, so that the optimal spendable amount is 88% of the stated budget, and then splits that optimal amount across seven expenditure categories — venue, catering, decoration, photography, entertainment, transportation and miscellaneous expenses — using a percentage table selected by event type. For example, a Wedding allocates 32% of the optimal amount to Venue and 30% to Food & Catering, while a Conference or Corporate event allocates 38% to Venue and 28% to Catering, reflecting the differing priorities of each event type (Table -3). Per-guest cost is computed as the optimal amount divided by the audience size, giving the user an immediate, comparable cost figure regardless of event scale.

Table -3: Illustrative Event-Type-Aware Budget Split (of the 88% Optimal Amount)

Event Type	Venue	Catering
Wedding	32%	30%
Conference / Corporate	38%	28%

`get_ai_insights()` extends this breakdown with a cost-per-head figure (budget ÷ audience size): a “Tight Budget” alert is raised below ₹250 per guest and a “Premium Experience” note above ₹2,500 per guest, alongside a fixed set of general cost-saving tips — advance-booking discounts, bundled vendor packages and weekday scheduling — and an overall estimated-cost/potential-savings summary expressed as 92% of the stated budget.

4.5 Feedback Sentiment-Analysis Algorithm

`analyze_sentiment()` first attempts to use TextBlob's built-in polarity scorer on the free-text feedback comment; if TextBlob is not available in the runtime environment, it falls back to a hand-built lexicon that counts positive and negative keyword occurrences and derives a polarity score from their relative counts, in the same spirit as the lexicon-based approaches evaluated by Hazarika et al. [7] and Prasanna Kumar [8]. In both cases, a polarity above +0.15 is classified Positive, a polarity below -0.15 is classified Negative, and everything in between is classified Neutral. The resulting label and numeric polarity are stored alongside the feedback record, making sentiment available to the admin analytics dashboard without any separate batch job.

4.6 Security — Salted Password Hashing

`verify_password()` inspects the stored hash prefix: values beginning with pbkdf2: or scrypt: (Werkzeug's `generate_password_hash` output) are verified with `check_password_hash`, while any other stored value is treated as a legacy SHA-256 digest and compared directly — allowing both hashing schemes to coexist safely without ever storing or comparing plain-text passwords.

4.7 Error Handling and Edge Cases

A system built around a trained model, a relational database and a third-party weather API needs defined behaviour for the unhappy path, not only the happy one. Recommendation and budget-prediction requests with a missing or non-

numeric budget fall back to a sensible default (₹50,000) rather than raising an error, and an event type unseen during training is matched to the closest known type through partial string matching rather than causing the recommender to fail outright. The weather-advisory feature is treated as an unreliable external dependency: a failed or unreachable OpenWeatherMap call is caught and does not interrupt the surrounding planning workflow. Role-based routes are guarded consistently: any protected route checks the session before proceeding, so an unauthenticated visitor is redirected to /login, and admin-only or organizer-only routes remain inaccessible to a plain user session.

5. EXPERIMENTAL RESULTS

5.1 Testing Process and Strategy

System testing proceeded in three phases. First, a diagnostic script (test_app.py) exercised the configuration loader, database initialisation, the ML recommender (recommendation, budget prediction and AI insights), the sentiment-analysis function and the Flask application's import/compilation, printing a pass/fail result for each. Second, every Flask route was manually exercised end-to-end through the browser for all three roles — User, Organizer and Admin — to confirm that pages render correctly, forms submit successfully, and session-protected routes correctly redirect unauthenticated visitors to the login page. Third, boundary and negative cases — an event with no prior training data, a non-numeric budget, or a zero audience size — were exercised to confirm the ML and budget functions fall back to sensible defaults instead of raising an error. This combination of unit-level, integration, black-box/functional and role-based access testing mirrors the layered strategy recommended for classification-based recommender deployments in the literature reviewed in Section 2.

5.2 Representative System Test Cases

Table -4 reports a representative subset of the fourteen system test cases exercised across authentication, event management, recommendation, budget prediction, booking and feedback; all fourteen cases passed on the final build.

Table -4: Representative System Test Cases

ID	Test Scenario	Expected Result	Result
TC-02	Login with correct credentials	Session created; redirected to dashboard	Pass
TC-06	ML recommendation for a Wedding event	Returns a vendor category with confidence %	Pass
TC-07	Budget breakdown totals reconcile	Category sum $\approx$ 88% of stated budget	Pass
TC-08	Recommendation with missing budget	Falls back to default (₹50,000) without error	Pass
TC-10	Organizer accepts a booking	Booking status updated to 'confirmed'	Pass
TC-11	Submit positive feedback text	Sentiment classified as Positive	Pass
TC-14	Non-admin accesses /admin	Access denied / redirected	Pass

5.3 Application Walkthrough

The deployed application exposes distinct views for each role. Fig -3 shows the AI Recommendations & Insights page — the system's core intelligence surface — presenting the recommended vendor category with its confidence score, the seven-category budget breakdown, AI-generated cost-saving insights, and a ranked vendor list with the recommended category surfaced first.

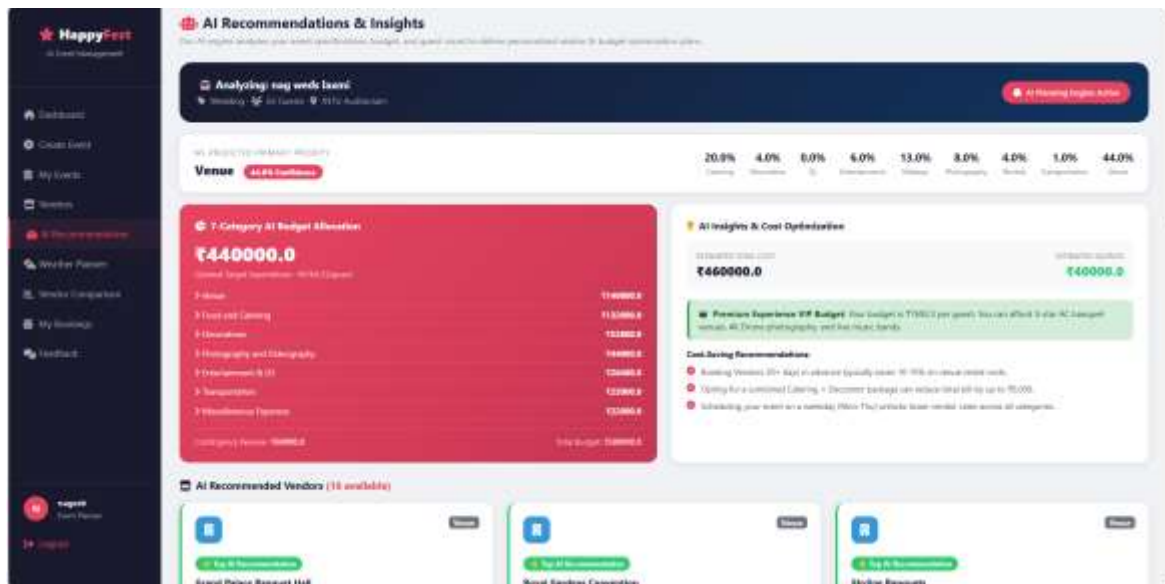


Fig -3: AI Recommendations & Insights Page

Fig -4 shows the administrator dashboard, which consolidates platform-wide analytics — total users, events, vendors and feedback counts — together with aggregate feedback-sentiment trends, giving an administrator a single view of platform health without querying the database directly.

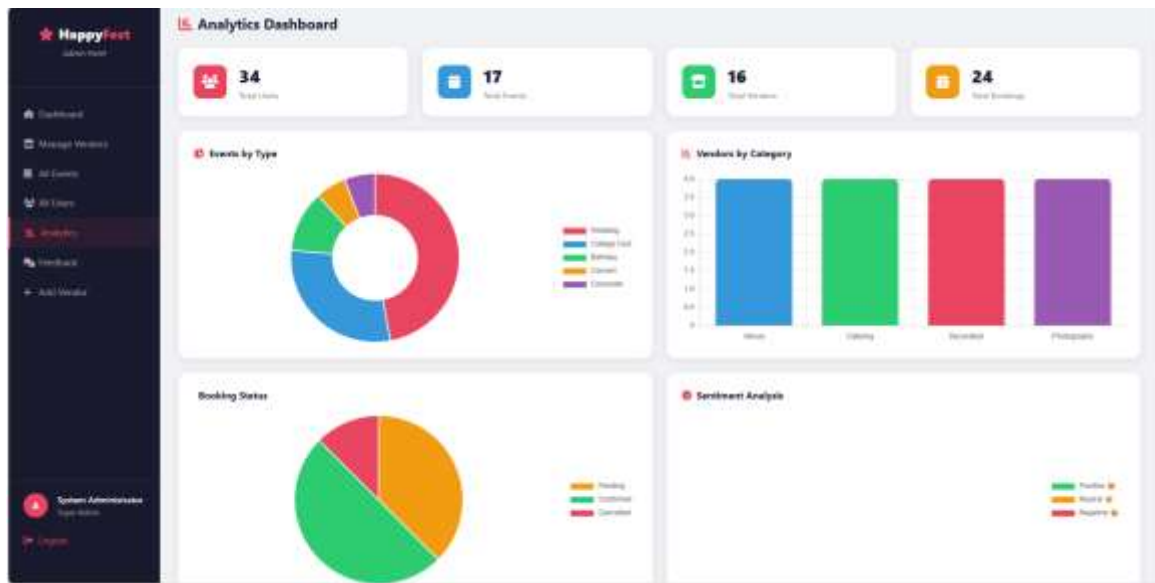


Fig -4: Administrator Analytics Dashboard

5.4 Discussion

The testing results in Sections 5.1 and 5.2 confirm that the three intelligence components — recommendation, budget allocation and sentiment analysis — function correctly both in isolation and when composed within the full Detect-to-feedback workflow: creating an event reliably produces a recommended category and a reconciling budget breakdown, and submitted feedback is reliably sentiment-scored and reflected in the admin dashboard. Unlike the event-attendee-facing systems reviewed in Section 2 [1]–[3], which stop at recommending which events a user should attend, and unlike EventVerse [4], which recommends vendors without a structured budget model, HappyFest carries every recommendation through to an explainable, category-wise budget plan and closes the loop with a feedback-driven quality signal — the combination identified as the key gap in Section 2. The trade-off is scale: the 37-record recommendation dataset and the fixed percentage tables behind the budget engine are deliberately compact choices suited to a new platform with limited historical data, and the current implementation trades some of the personalisation a larger, continuously retrained model could offer for speed, transparency and ease of deployment.

6. CONCLUSION

This paper presented HappyFest, a role-based Flask web application that unifies event creation, ML-driven vendor-category recommendation, rule-based budget allocation, vendor booking and feedback-driven sentiment analysis on a single platform for home and small-institutional event planning — a user group whose planning workflow existing event-recommendation research largely leaves on the attendee-facing side of the problem. A RandomForestClassifier trained on a compact, curated 37-record event–vendor dataset supplies a fast, explainable vendor-category recommendation with a confidence score, while a transparent, event-type-aware rule-based engine turns the same event parameters into a seven-category budget breakdown, a per-guest cost figure and AI-generated cost-saving insights. A TextBlob/lexicon-based sentiment-analysis routine converts free-text post-event feedback into a structured Positive/Neutral/Negative signal without any separate batch process, feeding directly into an administrator analytics dashboard. The application was implemented across all three intended roles — User, Organizer and Administrator — and tested end-to-end; a diagnostic test script together with fourteen representative, manually exercised system test cases confirm that registration, event management, recommendation and budget prediction, vendor booking, feedback sentiment analysis and administrative analytics all function correctly together.

The project also demonstrates that meaningful decision support does not always require the heaviest available model: a modest, fast-training Random Forest classifier and a transparent rule-based budget engine were sufficient to produce useful, explainable recommendations given the limited historical data available to a new platform, while keeping the system inexpensive to run and easy to reason about for both developers and evaluators.

Limitations: Two limitations are worth stating plainly. First, the recommendation model is trained on a deliberately small, curated 37-record dataset rather than on real historical booking data at scale, so its recommendations reflect the priors encoded in that seed dataset rather than learned, platform-specific user behaviour; Section 2's review of attendance-prediction and recommendation literature [1]–[3] suggests that accuracy and personalisation typically improve substantially once real usage data becomes available. Second, the sentiment-analysis module relies on TextBlob's lexicon-and-pattern-based scoring, which — consistent with the limitations reported by Hazarika et al. [7] and Prasanna Kumar [8] — struggles with sarcasm, negation and domain-specific phrasing in short feedback comments. Both limitations are addressed directly in the future work below.

7. FUTURE SCOPE

- Replace the lexicon/TextBlob sentiment module with a fine-tuned BERT-based classifier once a sufficiently large, labelled feedback dataset has been collected from real usage.
- Introduce collaborative filtering, using accumulated booking and rating history, to personalise vendor recommendations beyond the current event-type/budget/audience heuristic.
- Integrate a real payment gateway to support end-to-end vendor booking with online payment and refunds.
- Add real-time chat or notifications (e.g. via WebSockets) between users and organizers for faster booking coordination.
- Expand the ML training dataset with real historical event data, rather than the current curated seed dataset, to improve recommendation accuracy and confidence.
- Add automated unit and integration test suites (e.g. using pytest) to complement the current manual and diagnostic-script testing described in Section 5.
- Deploy the application on a cloud platform with a managed MySQL instance and HTTPS, and replace the Flask development server with a production WSGI server (e.g. Gunicorn behind Nginx).

REFERENCES

- [1] D. Horowitz, D. Contreras, and M. Salamó, “EventAware: A Mobile Recommender System for Events,” *Pattern Recognition Letters*, vol. 105, pp. 121–134, 2017.
- [2] R. Varaprasad, S. Ramasubbareddy, and K. Govinda, “Event Recommendation System Using Machine Learning Techniques,” in *Innovations in Computer Science and Engineering, Lecture Notes in Networks and Systems*, vol. 385, Springer, Singapore, 2022.
- [3] R. Awad, “Event Attendance Prediction and Recommendation System: A Comparative Study,” M.S. thesis, Politecnico di Milano, 2023.
- [4] “EventVerse: A Smart Web-Based Event Planning and Vendor Recommendation System Using Supabase and React,” *International Journal of Computer Techniques (IJCT)*, vol. 13, no. 3, 2026.
- [5] L. Breiman, “Random Forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [6] F. Ricci, L. Rokach, and B. Shapira, Eds., *Recommender Systems Handbook*, 2nd ed., Springer, Boston, MA, 2015.
- [7] D. Hazarika, G. Konwar, S. Deb, and D. J. Bora, “Sentiment Analysis on Twitter by Using TextBlob for Natural Language Processing,” *Proc. Int. Conf. on Research in Management & Technovation, ACSIS*, vol. 24, pp. 63–67, 2020.
- [8] Prasanna Kumar H. R., “Sentiment Analysis: TextBlob for Decision Making,” *International Journal of Scientific Research & Engineering Trends*, 2023.
- [9] Pallets Projects, “Flask Documentation,” <https://flask.palletsprojects.com/>.
- [10] scikit-learn developers, “scikit-learn: Machine Learning in Python,” <https://scikit-learn.org/>.
- [11] K. B. Maruthiram and V. Keerthana, “Optimizing Human Face Detection With Multi-Intensity Image Fusion in Deep Learning,” *International Journal of All Research Education and Scientific Methods (IJARESM)*, vol. 12, no. 8, pp. 1671–1678, 2024.
- [12] K. B. Maruthiram and G. V. R. Reddy, “Improving Object Detection and Classification for Autonomous Vehicle in Adverse Weather Conditions,” in *Proc. 2024 2nd International Conference on Artificial Intelligence Trends and Pattern Recognition (ICAITPR)*, 2024, pp. 1–6.
- [13] K. B. Maruthiram, G. V. R. Reddy, and S. Supraja, “Object Detection of Autonomous Vehicles in Adverse Weather Conditions,” *GIS Science Journal*, vol. 12, no. 7, pp. 557–566, 2025.
- [14] K. B. Maruthiram and G. V. R. Reddy, “A Survey Paper on Object Detection and Localization Methods in Image Processing,” *International Journal of Creative Research Thoughts*, vol. 13, no. 6, 2024.