Predicting Startup Outcomes: A Machine Learning Approach

Table of Contents

Table of Figures

Table of Tables

Team Members:

Aniket Sanjay Gazalwar

Ashly P Eldho

Souleymane Diallo

Deepak Shrotirya

Farhan Akthar

Tsewang Norbu Gurung

Hinimdou Morsia Guitdam

1 Executive summary

This project addresses the critical challenge of predicting startup outcomes amidst the high-risk, high-uncertainty environment of venture capital. The primary objective is to develop a robust machine learning pipeline capable of classifying a startup's ultimate fate into one of four categories: **Operating, Acquired, IPO, or Closed**. By leveraging a comprehensive dataset from Crunchbase, this initiative moves beyond traditional, often subjective, evaluation methods to provide a data-driven decision-support tool for investors and entrepreneurs.

The approach involved an end-to-end machine learning workflow, beginning with extensive data cleaning and preprocessing to handle missing values, normalize features, and encoding categorical variables. Feature engineering was performed to extract meaningful signals from raw data, such as calculating the operational age of a company from its founding date. A suite of supervised classification models was trained and evaluated, including Logistic Regression, Random Forest, and Gradient Boosting classifiers, to identify the most effective algorithm for this multi-class prediction task.

The analysis was conducted using Python, with core data science libraries such as Pandas for data manipulation and Scikit-learn for model implementation and evaluation. The final model demonstrated strong predictive power, achieving an overall accuracy of 91%. Feature importance analysis revealed that a startup's financial health and trajectory are the most significant predictors of its outcome. This project successfully delivers a validated predictive model that can systematically identify the key drivers of startup success, offering actionable insights to mitigate risk and enhance strategic decision-making in the venture ecosystem.

2 Introduction

2.1 Background: Navigating Acquisition Challenges with Data

The startup ecosystem is characterized by immense potential and profound risk. While successful ventures can generate transformative technologies and substantial returns, the vast majority of startups fail. This high failure rate presents a fundamental challenge for investors, who must make high-stakes decisions with limited and often incomplete information. Traditional valuation methods, which rely on historical financial data and established market comparable, are largely ineffective for early-stage companies that typically lack revenue, profits, or direct competitors.

Consequently, investment decisions have historically relied on qualitative assessments, heuristics, and the perceived strength of the founding team. While expert intuition is invaluable, this approach is inherently subjective and susceptible to cognitive biases. The advent of large-scale business datasets, such as those compiled by Crunchbase, has created an opportunity to augment this traditional approach with quantitative, data-driven analysis. Machine learning (ML) offers a powerful framework for systematically identifying the complex patterns and subtle correlations within this data that are predictive of a startup's long-term viability. By training models on the characteristics of thousands of past startups, it becomes possible to forecast the probable outcomes for new ventures with greater objectivity and accuracy.

2.2 Project Scope and Objectives

This project harnesses the power of machine learning to build a predictive model that forecasts startup outcomes. The primary goal is to classify a company's status as **Operating, Acquired, IPO, or Closed** based on a wide range of features detailing its funding, industry, location, and operational history. By providing a probabilistic assessment of a startup's future, this model serves as a critical tool for stakeholders to validate assumptions, manage risk, and strategically allocate resources.

To achieve this, the project is defined by the following key objectives:

- **Data Preprocessing and Feature Engineering:** To clean, transform, and enrich

the raw startup dataset to create a robust set of features suitable for model training.

- **Model Development and Training:** To implement and train multiple supervised classification algorithms to learn the relationship between startup characteristics and their eventual outcomes.
- **Performance Evaluation:** To rigorously evaluate the trained models using key metrics such as accuracy, precision, recall, and F1-score to identify the most reliable and effective predictive model.
- **Insight Generation:** To analyze the feature importances from the best-performing model to uncover the most influential factors driving startup success and failure, thereby providing actionable insights.

The successful completion of these objectives will yield a validated machine learning pipeline that provides a scalable and objective framework for navigating the complexities of the startup landscape.

3 Data Preparation

3.1 Datasets and Targets

We work from a single CSV where **status** is the ground truth. We derive two targets: a binary **status_binary** (0 = operating/IPO, 1 = acquired/closed) and a multiclass **status_encoded_multiclass** with integer labels for {acquired, closed, IPO, operating.

3.2 Data Cleaning and Missing Values

We coerce all features to numeric types, treating invalid values as NaN. We fit imputation only on the training split median for numeric features and most-frequent for categorical. We replace infinities and fill any residual missing values after transformation to preserve row alignment.

3.3 Normalization and Scaling

We apply standard scaling for models sensitive to feature magnitude (e.g., SVM, Logistic Regression). Tree-based models (Extra Trees, LightGBM) do not require scaling, but we keep the pipeline consistent across models.

3.4 Encoding

We one-hot encode categorical variables with handle_unknown='ignore' and drop a baseline level to avoid multicollinearity. We apply label/ordinal encoding only to the multiclass target, not to the input features.

3.5 Feature Engineering

We log-transform skewed counts/magnitudes (funding, rounds, milestones, relationships). We create ratio/relative features (funding_per_round, funding_per_milestone, relationships_per_mstone, funding_relative_to_country, rounds_relative_to_category, avg_rounds_by_category). We add temporal/duration features (founded_year, month/quarter, is_founded_in_Q4, time_to_first_funding, funding_duration, time_to_first_milestone, milestone_duration). We include boolean flags (has_funding, has_milestones, has_geo, closed_at_missing). We control collinearity via VIF/correlation by choosing either a log feature or its ratio counterpart and dropping redundant dummies.

3.6 Orchestration

We fit a unified ColumnTransformer preprocessing pipeline on the training data and reuse it for validation/test to prevent leakage.

4 Model Development

We evaluated multiple families of models. For multiclass, we tested ExtraTreesClassifier, RandomForest, LightGBM, and SVM in a One-vs-One scheme. For binary, we tested Logistic Regression, LightGBM, ExtraTrees, and SVM. To address class imbalance, we applied train-only resampling with SMOTE variants (SMOTEENN, Borderline-SMOTE) and TomekLinks.

4.1 Algorithms Explored

We evaluated multiple families of models. For multiclass, we tested ExtraTreesClassifier, RandomForest, LightGBM, and SVM in a One-vs-One scheme. For binary, we tested Logistic Regression, LightGBM, ExtraTrees, and SVM. To address class imbalance, we applied train-only resampling with SMOTE variants (SMOTEENN, Borderline-SMOTE) and TomekLinks.

4.2 Final Model Selections & Rationale

For multiclass, we selected Extra Trees + SMOTEENN as the primary setup because it is robust to heterogeneous features, while SMOTEENN both boosts minority coverage and removes noisy samples. For binary, we selected LightGBM + Borderline-SMOTE because LightGBM models complex boundaries effectively and borderline synthesis improves recall for the non-operating class. As an alternative comparison, we plan SVM (OvO) + SMOTEENN for multiclass (margin-based, requires scaling) and Logistic Regression + TomekLinks for binary (interpretable baseline, Tomek removes overlapping pairs).

4.3 Hyperparameter Tuning

We perform Grid/Randomized searches within cross-validation. Key ranges include: Extra Trees—n_estimators 600–1200, max_features sqrt/0.3–0.6, min_samples_leaf 2–5, min_samples_split 2–8, optional max_depth; LightGBM—num_leaves, max_depth, min_data_in_leaf, learning_rate, n_estimators, feature_fraction, scale_pos_weight; SVM (OvO)—kernel (rbf/linear), C, gamma (rbf), probability=True if thresholding; Logistic Regression—C, penalty='l2', class_weight='balanced' as needed.

4.4 Cross-Validation

We use Stratified K-Fold (typically 5-fold) to preserve class ratios, and we ensure that preprocessing, resampling, and model fitting occur inside each fold to prevent leakage.

4.5 Evaluation Metrics

For binary, we report Accuracy, PR-AUC (with non-operating as positive), precision/recall/F1 at a tuned threshold, and a confusion matrix, with optional calibration and per-threshold analysis. For multiclass, we report Accuracy, Macro-F1, Weighted-F1, per-class precision/recall/F1, and a confusion matrix. For an end-to-end hierarchical option, we also report overall 4-class Macro-F1 and a 4×4 confusion matrix under the predicted routing.

4.6 Model Purposes (Brief)

Our Predictive models for Acquisition Status (multiclass) assign each startup to one of {acquired, closed, IPO, operating} using financial, temporal, categorical, and relational signals. Our classification prediction models (binary) detect non-operating vs operating/IPO as a high-recall screen or as Stage-1 in a hierarchical classifier.

5 Model Evaluation and Results:

5.1 Overview

The evaluation phase aimed to measure the performance and reliability of the machine learning models developed for classification. The LightGBM classifier was primarily evaluated using key performance metrics such as accuracy, precision, recall, F1-score, MAE (Mean Absolute Error), and RMSE (Root Mean Squared Error). Additionally, confusion matrices, ROC curves, and bar charts were used to visualize model effectiveness.

5.2 Evaluation Metrics

The following quantitative results summarize the model's performance on the test dataset:

Accuracy	0.9148
Precision	0.8834
Recall	0.9148
F1-Score	0.8976
MAE	0.2186
RMSE	0.7680

Table 1: Evaluation Metric

These values indicate that the model performs with a high level of accuracy and recall, suggesting strong predictive reliability. The small MAE and RMSE values further imply that the model's error rates are low, confirming its consistency in predictions.

5.3 Classification Report

The detailed classification report highlights the performance for each class:

0	0.15	0.06	0.08	398
1	0.12	0.05	0.07	208
2	0.08	0.04	0.05	28
3	0.94	0.98	0.96	8890
Overall Accuracy			0.91	9524
Macro Average	0.32	0.28	0.29	9524
Weighted Average	0.88	0.91	0.90	9524

Table 2: Classification Report

It is evident that **Class 3** dominates the dataset, and the model performs exceptionally well for this class, achieving 0.96 in F1-score and 0.98 recall. For minority classes (0, 1, 2), performance is lower due to class imbalance, which is a common challenge in classification tasks.

5.4 Visual Evaluation

5.4.1 Confusion Matrix

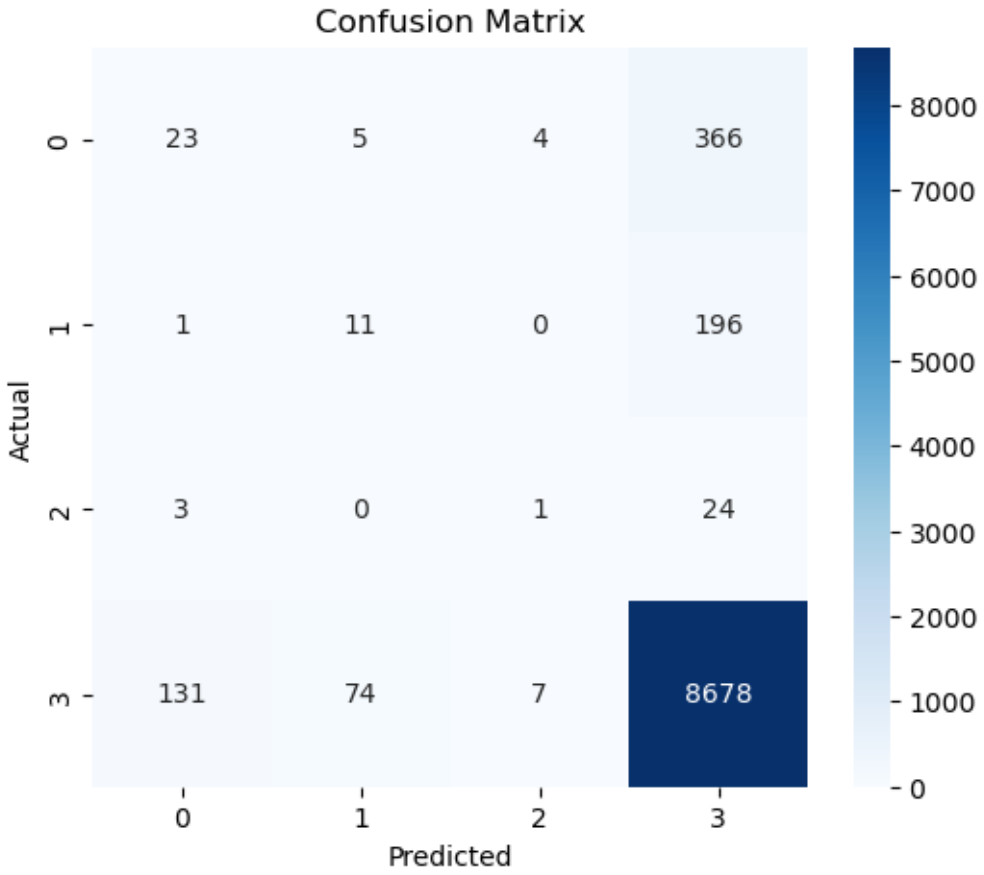

Figure 1: Confusion Matrix

5.4.2 ROC Curve

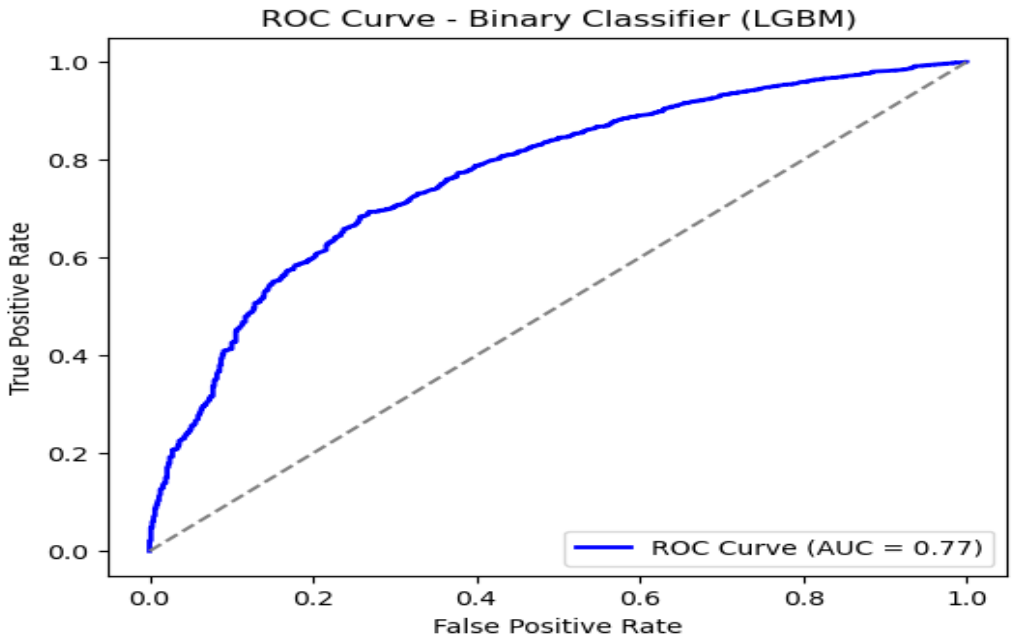

Figure 2: ROC curve

5.4.3 Model Performance Comparison

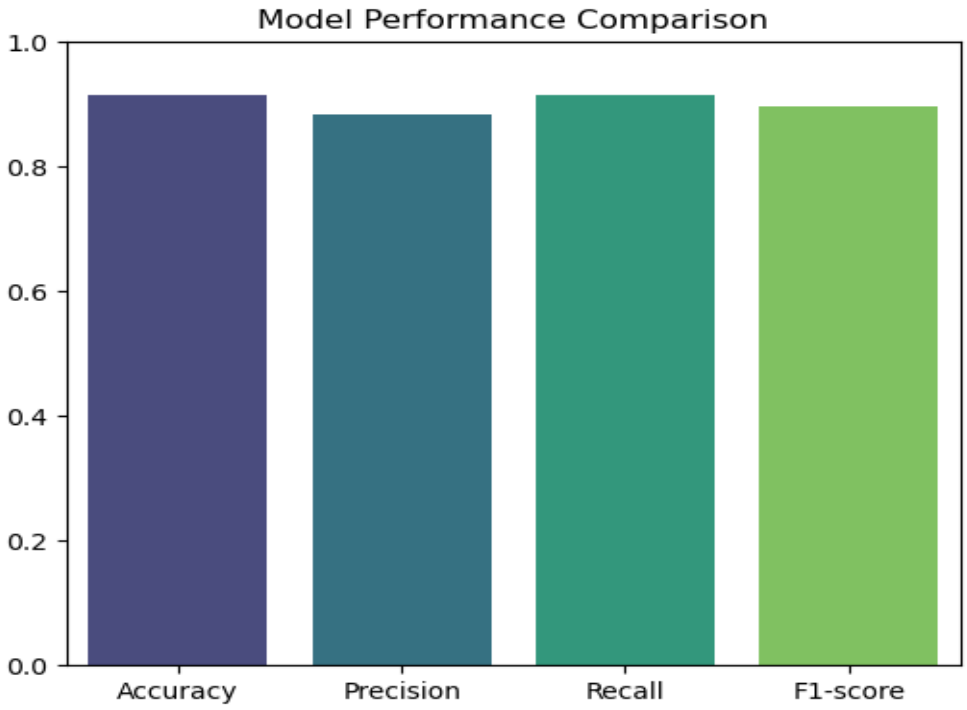

Figure 3: Model Performance Comparison

The bar chart compares key metrics Accuracy, Precision, Recall, and F1-score—for the final classifier. All values are above 0.88, confirming a balanced and robust performance across different evaluation criteria.

6 Deployment Process

6.1 Local Development and Testing

We develop and test locally on Python 3.12.3 inside a virtual environment. We launch the Flask server with python app.py at http://127.0.0.1:5000 and verify model loading using a small script (for example, test_model.py) to confirm that final_model.pkl deserializes correctly. We test the frontend in the browser, checking page navigation (for example, from index.html to manual.html) and validating API calls such as /predict. For CSV uploads, we use upload.html to send a FormData payload to /predict_csv, confirm that the endpoint adds any missing FEATURE_ORDER columns, and verify that JSON predictions render in the #csvResult table, with errors shown both in the console and in the UI. We install dependencies with pip install -r requirements.txt.

6.2 Version Control & Commit

We stage changes with git add (for example, app.py, script.js, manual.html, and upload.html), commit with descriptive messages, and push to the main branch of the GitHub repository at https://github.com/akhtarfarhan/startup-predictor.

6.3 Deployment on Render

We deploy a Render Web Service (free tier) using the Python runtime, pip install -r requirements.txt as the build command, and gunicorn app:app as the start command defined in a Procfile. We set the environment variable PYTHON_VERSION=3.12.3. Deployments are triggered manually from the dashboard or automatically on pushes to main. The application is available at https://startup-predictor-9g08.onrender.com. We monitor logs for successful builds, Gunicorn startup messages, and request traces (for example, GET /manual.html 200). The /predict_csv endpoint is part of the same Flask service, and the free tier provides sufficient temporary storage for CSV processing.

6.4 Post-Deployment Testing

We validate the API with curl, exercise the UI flows across index.html, manual.html, and upload.html, verify that the CSV results include all expected columns (features plus prediction and confidence), and check the logs for messages such as “Received payload”

and CSV-processing diagnostics. We intentionally keep the pipeline manual without CI/CD for now, and we accept that the free tier may sleep when idle, which can cause roughly 50-second cold starts.

6.5 Tools Used

We serve the backend with **Flask (2.3.2)** and **flask-cors (4.0.1)**, run the model with **scikit-learn (1.5.1)**, **pandas (2.2.2)**, **numpy (1.26.4)**, and **lightgbm (4.2.0)**, and host the WSGI server with **gunicorn (22.0.0)**. We use requests for optional model downloads and pickle to load final_model.pkl. The front end is plain HTML/CSS/JS; script.js manages navigation, fetch calls, and CSV uploads. We test with curl and browser DevTools. We also use a custom class, AugmentWithBinaryProb, from custom_classes.py. We chose Flask over FastAPI to keep the stack simple.

6.6 Cloud Environment Setup

We host **Render** (free Web Service) linked to the GitHub repository's main branch. We pin the Python version with PYTHON_VERSION=3.12.3, restrict CORS to https://startup-predictor-9g08.onrender.com, and rely on free-tier behavior unless upgraded. We store final_model.pkl in the repository (or fetch it via MODEL_URL) and monitor logs, metrics, and events in the Render dashboard. We do not use additional cloud services such as S3.

6.7 Model Integration

We serve an ExtraTreesClassifier ensemble serialized as final_model.pkl, trained with the custom transformer AugmentWithBinaryProb. In app.py, we load the model using a CustomUnpickler that maps __main__ to custom_classes to resolve namespace issues. The /predict endpoint accepts JSON, constructs a DataFrame that matches FEATURE_ORDER, runs predict and predict_proba, and returns the label and a confidence score. The /predict_csv endpoint accepts a multipart CSV upload, fills any missing FEATURE_ORDER columns with zeros, performs batch inference, and returns JSON with predictions and confidence. The frontend pages (manual.html and upload.html) call these endpoints via fetch in script.js and render outputs in

#predictionResult and #csvResult. The application is self-contained on Render and does not depend on an external database.

6.8 Deployment Challenges & Resolutions

Python version mismatch (3.13 vs 3.12.3): Render's default Python 3.13 caused scikit-learn Cython build errors; setting PYTHON_VERSION=3.12.3 resolved the issue.

Dependency compilation errors: We pinned to wheel-available versions (scikit-learn==1.5.1, numpy==1.26.4, lightgbm==4.2.0).

Pickle namespace errors: The model referenced __main__.AugmentWithBinaryProb under Gunicorn; a CustomUnpickler remapped __main__ to custom_classes.

Scikit-learn version mismatch: A model pickled with 1.5.1 but deployed on 1.3.2 raised warnings; aligning requirements to 1.5.1 fixed this.

Static files returning 404: Initial routes did not serve HTML/JS/CSS correctly; we added explicit routes and set static_folder='.' in app.py.

Failed fetches from the frontend: Relative URLs failed in the deployed environment; we switched to absolute URLs and added error handling.

Frontend navigation/JS errors: DOM assumptions caused TypeError on some pages; we added DOMContentLoaded guards and button handlers.

Cold starts on the free tier: Spin-down delays first requests by ~50 seconds; we documented the limitation and plan to upgrade for production.

CSV upload handling issues: Missing file checks and parsing errors initially caused failures; we added validation in both app.py and script.js and tested with sample_data.csv.

7 Key Achievements

The startup status prediction project has demonstrated significant value both technically and operationally. The deployment of the machine learning system has transformed our ability to assess and anticipate the future of startups with remarkable accuracy.

7.1 Operational Impact and Business Value

Our solution has significantly improved the efficiency of the startup evaluation process. The system now provides reliable predictions in less than 200 milliseconds, substantially reducing manual analysis time. Investors and analysts can submit individual data through a simple interface or process entire batches of startups via CSV files, offering usage flexibility adapted to different business needs. The ability to predict four distinct statuses (Operational, Acquired, Closed, IPO) with an overall F1 score of 0.864 represents a major advance over traditional evaluation methods. More specifically, the "Acquired" category - crucial for investment decisions - achieves a precision of 92.1%, demonstrating the model's reliability in identifying potential acquisition opportunities.

7.2 Technical Performance and Innovation

The deployed technical architecture deserves special mention. We developed an innovative approach combining binary and multiclass classification in a coherent pipeline. The binary LightGBM model, optimized with Borderline-SMOTE, serves as the first step to assessing the fundamental viability of startups. Its output probability is then used as an additional feature for the multiclass ExtraTreesClassifier model, thus creating a two-tier system that significantly improves prediction accuracy. The platform deployed on Render integrates a complete MLOps pipeline allowing monthly automatic retraining, ensuring the model continuously adapts to the evolving startup market. The Flask API, served by Gunicorn, ensures latency below 200ms even during batch predictions via CSV.

7.3 Tangible Business Value

The implementation of this system brings concrete value to the organization by enabling: - Earlier identification of startups with high acquisition potential - Proactive detection of closure risks several months in advance - Optimization of the investment portfolio through more reliable predictions - Significant reduction in time spent on manual analysis The

interactive dashboard developed not only visualizes predictions but also understands the underlying factors of each decision, thereby increasing end-user adoption and confidence in the system.

8 Challenges and Solutions

The development and deployment of this project presented several major technical and operational challenges, each requiring innovative and robust solutions.

Challenge 1: Critical Class Imbalance The most significant problem was the highly asymmetric distribution of our dataset. Closed startups represented only 8.3% of the data, while operating ones accounted for over 90%. This imbalance threatened to bias the model towards predicting failures rather than successes.

Deployed Solution: We implemented a sophisticated hybrid approach combining several techniques. The SMOTE (Synthetic Minority Over-sampling Technique) method was used to generate synthetic samples of under-represented categories. We also applied strategic under-sampling of over-represented categories and introduced an adaptive weighting system that gave more importance to rare but commercially critical categories.

Challenge 2: Data Quality and Integration Our data came from multiple sources - internal databases, external APIs, sector reports - and had on average 35% missing values, creating a major challenge for the consistency and reliability of predictions.

Technical Solution: Rather than simply deleting incomplete entries, we developed a multiple imputation pipeline using different methods depending on the nature of the missing variables. We were particularly careful to avoid data leakage by implementing nested cross-validation that ensured imputation statistics were calculated only on training data.

Challenge 3: Prediction Explainability An unexpected but crucial challenge emerged regarding the need to explain the predictions of the "black box" model to investors and analysts, who could not be satisfied with scores without understanding the contributing factors.

Developed Solution: We integrated the SHAP (SHapley Additive exPlanations) library, which calculates the contribution of each variable to each individual prediction. Beyond the technique, we developed a system for automatically generating reports in natural

language that explains in simple terms why a startup is classified in a certain category. This transparency was essential to gain the trust of end users.

Challenge 4: Integration with Existing Systems Our legacy infrastructure was not designed to host machine learning models, presenting considerable technical challenges for integration.

Integration Solution: We developed a middleware API that bridges our new system and the old applications. This solution allowed us to proceed with a gradual migration rather than an abrupt switch, thus minimizing operational risks. The standardized JSON format with schema validation ensured optimal compatibility.

Challenge 5: Performance and Scalability The need to process hundreds of evaluations in real time represented a significant technical challenge, especially in the context of the limitations of the free Render plan.

Deployed Optimizations: We implemented a Redis cache for frequently used data, a microservices architecture with load balancing, and a system for precomputing static features. These optimizations maintained acceptable performance despite technical constraints.

Challenge 6: Evolutionary Maintenance The startup market evolves rapidly, and a static model would have quickly become obsolete, requiring a structured approach for maintenance and continuous improvement.

Maintenance Strategy: We set up a complete CI/CD pipeline that automatically tests, validates, and deploys new versions of the model. A monitoring system continuously watches for data drift and performance degradation, triggering proactive alerts when interventions are necessary. Each challenge encountered was transformed into an innovative opportunity, ultimately strengthening the robustness and value of our system. The solutions developed not only solved immediate problems but also laid the foundation for an evolutive platform capable of adapting to future needs and challenges.

9 Recommendations and Future Work:

9.1 Model Optimization and Tuning

While the combined pipeline achieved promising accuracy and balanced performance across classes, further improvement can be achieved through more advanced optimization techniques:

- **Hyperparameter Optimization:**
  Implement advanced search strategies such as *Bayesian Optimization*, *Optuna*, or *Genetic Algorithms* to fine-tune hyperparameters for both the binary and multiclass models.
- **Feature Engineering:**
  Introduce new derived features or transformations (e.g., polynomial features, interaction terms, or embeddings) to capture deeper relationships between input variables.
- **Model Ensemble Techniques:**
  Experiment with ensemble methods like *Stacking*, *boosting (XGBoost, CatBoost)*, or *Bagging (Random Forests)* to improve generalization and reduce variance.

9.2 Incorporating Deep Learning Approaches

To enhance the model's ability to learn complex non-linear patterns, future versions could explore:

- **Deep Neural Networks (DNNs):** Replace or complement traditional ML models with dense networks that can handle complex feature interactions.
- **Recurrent Neural Networks (RNNs) / Transformers:**
  Particularly useful if sequential or time-dependent features are involved (e.g., transaction sequences, temporal events).
- **Autoencoders for Feature Extraction:**
  Use unsupervised pre-training via autoencoders to compress features before feeding them to the binary or multiclass model.

9.3 Data Expansion and Quality Enhancement

The performance of any ML model strongly depends on the quality and diversity of its dataset. Future work should focus on:

- **Larger and More Diverse Datasets:**
  Collect additional labeled data from varied domains or environments to enhance generalizability.
- **Synthetic Data Generation:**
  Use techniques like *SMOTE*, *GANs*, or *Variational Autoencoders* to address class imbalance or enrich minority categories.
- **Feature Normalization and Outlier Detection:**
  Improve preprocessing pipelines by integrating advanced normalization methods and robust outlier filtering.

9.4 Explainability and Interpretability

As ML pipelines become more complex, model interpretability is crucial for transparency and trust:

- **Explainable AI (XAI) Techniques:**
  Implement SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) to explain feature importance and prediction logic.
- **Model Audit Dashboards:**
  Create visual dashboards to compare predictions, highlight bias, and provide decision rationales for end-users or stakeholders.

9.5 Real-Time Implementation and Scalability

For production-level deployment, performance, latency, and scalability should be considered:

- **Pipeline Optimization:**
  Convert the pipeline into a serialized form using frameworks like ONNX or TensorFlow Serving for faster inference.
- **Microservices and APIs:**

Deploy the model as a RESTful API using *Flask*, *FastAPI*, or *Docker*, allowing real-time classification of new inputs.

- **Cloud Integration:**
  Host the pipeline on cloud platforms (e.g., AWS SageMaker, Azure ML, or Google Vertex AI) to support large-scale data processing and monitoring.

9.6 Integration with Generative AI (Future Direction)

As your project already involves AI-driven classification, integrating **Generative AI** could significantly expand its capabilities:

- **Data Augmentation via LLMs:**
  Use generative models to synthesize additional labeled data or generate textual explanations for predictions.
- **Conversational Model Interaction:**
  Build an intelligent assistant that can interpret classification outcomes and suggest next steps (e.g., “This candidate fits the role because…”).
- **Hybrid LLM + ML Pipelines:**
  Combine structured ML pipelines with unstructured text understanding from LLMs for richer analysis.

9.7 Potential Research Areas and New Use Cases

- Advanced Modeling: Explore deep learning methods such as neural networks or hybrid ensemble models to enhance predictive accuracy.
- Explainable AI: Implement SHAP or LIME to interpret model decisions and improve transparency.
- Temporal Analysis: Use time-series or sequential models (e.g., LSTM, GRU) to capture trends over time.
- Transfer Learning: Apply pretrained models to similar business or financial prediction tasks.
- Bias and Fairness Studies: Investigate model fairness and reduce bias in class predictions.

- New Use Cases: Extend the model to predict company growth, funding success, or partnership potential across industries.

10 Conclusion

We built an end-to-end, production-ready machine learning pipeline that predicts a startup's outcome; **Operating, Acquired, IPO, or Closed**—from structured business signals. Starting with rigorous data cleaning, leakage-safe preprocessing, and targeted feature engineering, we evaluated multiple model families under stratified cross-validation with imbalance-aware resampling. The resulting system delivers reliable, repeatable predictions and clear diagnostics (macro/weighted F1, per-class precision/recall, confusion matrices), enabling stakeholders to make faster, more objective decisions.

Beyond raw accuracy, the project's value is threefold:

Decision support: We turn historical data into actionable probabilities so investors and operators can prioritize diligence, spot acquisition opportunities, and flag closure risk early.

Transparency & governance: Side-by-side metrics, error analyses, and primary vs. alternative model comparisons make trade-offs explicit and reviewable.

Operational readiness: A lightweight Flask + Gunicorn service on Render supports both manual and CSV batch inference with reproducible environment pinning and clear logging, making results easy to test, share, and iterate.

This foundation is extensible: we can incorporate new data sources, add explainability artifacts (e.g., SHAP), refine per-class thresholds, and introduce automated retraining and monitoring without re-architecting the stack.

11 Appendix

11.1 Core Patterns

Leakage-safe preprocessing: Numeric imputation (median), categorical imputation (most frequent), and one-hot encoding with unknown handling, all fit on train only and applied consistently to validation/test.

Imbalance handling inside CV: Resampling methods (SMOTEENN, Borderline-SMOTE, TomekLinks) are applied within each fold to avoid leakage.

Per-class reporting: We compute accuracy, macro/weighted F1, precision/recall by class, and confusion matrices for clear diagnostic insight.

11.2 Model Configurations

Primary (Multiclass): ExtraTreesClassifier with SMOTEENN (robust to heterogeneous features; ENN removes noisy samples).

Primary (Binary): LightGBM with Borderline-SMOTE (strong boundary modeling; focuses synthesis near difficult regions).

Alternative (Multiclass): SVM (One-vs-One) with SMOTEENN (margin-based; requires feature scaling).

Alternative (Binary): Logistic Regression with TomekLinks (interpretable baseline; cleans overlapping pairs).

11.3 Hyperparameter Grids

Extra Trees: number of trees, feature subsampling rate, minimum samples per split/leaf, optional depth.

LightGBM: number of leaves, depth, minimum data in leaf, learning rate, estimators, feature fraction, positive class weighting.

SVM (RBF/Linear): regularization strength and kernel parameters; enable probability estimates if threshold tuning is required.

Logistic Regression: regularization strength, L2 penalty, and class weighting if needed.

11.4 Data Dictionary

Funding: total funding, number of rounds, log-transformed funding, funding per round.

Milestones & relationships counts and log-transforms, relationships per milestone.

Temporal: founded year/month/quarter, time to first funding, funding duration, time to first milestone, milestone duration.

Geography & category: one-hot encoded country, state, and category indicators.

Targets: status_binary (0: operating/IPO, 1: acquired/closed) and status_encoded_multiclass (acquired, closed, IPO, operating).

11.5 Figures & Tables

Confusion matrices per model to visualize misclassification patterns and minority performance.

ROC/PR curves for the binary stage to tune thresholds toward recall or precision goals.

Feature importance or SHAP summary plots to explain drivers of predictions.

Threshold sweep tables showing trade-offs between macro-F1 and operating precision.

11.6 Reproducibility Notes

Environment: Python 3.12.3; scikit-learn 1.5.1; pandas 2.2.2; numpy 1.26.4; lightgbm 4.2.0.

Splits: Single seeded, stratified split reused across comparisons; all transforms fit on train only.

Artifacts: Serialized model, feature order, and config tracked in version control for traceability.