ML Project Report: Predicting **Startup Founder Retention**

Nipun Verma (IMT2023591)
Sahil Singh (IMT2023521)
Mohammed Ibrahim (IMT2023112)

GitHub Repository

Background and Problem Statement

Startup ecosystems are built around founders. A founder who stays with the company provides continuity of vision, better investor confidence and more stable team dynamics. A founder who exits early can trigger a cascade of problems: stalled roadmaps, new leadership struggles and renegotiated funding terms.

The dataset used in this project comes from the Kaggle competition *"Startup Founder Retention Prediction"*. The task is to predict whether a founder will `Stay` with the venture or `Leave`, based on demographic, financial and organizational information.

Unlike the earlier transport–cost regression task (where the target was a continuous cost value and the modelling focus was linear vs tree vs boosting regressors, this project is a **binary classification** problem. The primary goal is not only to get a high score but also to understand which factors matter for retention and why certain model families behave the way they do on this kind of structured data.

Three model families were used:

1. A **Logistic Regression** pipeline as a fast, interpretable baseline.
2. An **SVM with RBF kernel** as a strong non–linear classical model.
3. A **Neural Network (Tabular MLP)** as a flexible high–capacity model.

For fairness, all three models share a common preprocessing philosophy, but they deliberately use slightly different engineering details to suit their strengths. The report mirrors

the structure of the earlier medical transport project: starting from data description and cleaning, moving through feature engineering and finally giving an in–depth analysis of each model and the overall comparison.

Dataset Description

The training file contains 59,611 rows and 24 columns. The test file contains 14,900 rows. The target variable is:

- `retention_status` $\in \{\texttt{Stayed}, \texttt{Left}\}$

The target distribution in the training set is:

$$\texttt{Stayed} : 31{,}265, \quad \texttt{Left} : 28{,}346$$

which is reasonably balanced but not perfectly symmetric. This mild imbalance motivates the use of **macro F1** rather than accuracy.

The remaining features can be grouped naturally into numeric, nominal and ordinal types:

Numeric features `founder_age`, `years_with_startup`, `monthly_revenue_generated`, `funding_rounds`, `distance_from_investor_hub`, `num_dependents`, `years_since_founding`.

Nominal features `founder_gender`, `founder_role`, `working_overtime`, `personal_status`, `remote_operations`, `leadership_scope`, `innovation_support`.

Ordinal features These started life as strings like "Excellent", "Poor", "Very High" and had to be carefully mapped to integers:

`work_life_balance_rating`, `venture_satisfaction`, `startup_performance_rating`, `startup_stage`, `team_size_category`, `startup_reputation`, `founder_visibility`, `education_background`.

A recurring theme in this project is that most modelling difficulties come not from the algorithms, but from making these heterogeneous columns numerically meaningful without destroying information.

EDA and Preprocessing

Handling Missing Values and Data Types

The raw CSV contained missing values in both numeric and categorical columns, and multiple columns that looked numeric but were actually stored as strings. The cleaning strategy was:

- For **numeric and ordinal** columns, missing values were imputed using the median of each column. Median is more robust than the mean in the presence of skewed distributions, which is important because revenue and distance variables have long right tails.
- For **nominal** columns, missing values were filled with the most frequent category (mode). This is defensible because these features represent membership in discrete groups, and mode imputation avoids inventing artificial categories.

For all models built directly with scikit–learn pipelines (the Logistic Regression and SVM pipelines), imputation was handled via `SimpleImputer`. For the neural–network–oriented code, imputation was performed manually during preprocessing but followed the same median/mode philosophy.

The final sanity check after preprocessing printed the remaining number of NaNs in train and test; both were zero.

Boolean Normalization

Columns such as `working_overtime`, `remote_operations`, and `leadership_scope` contained values like "Yes", "No", "True", "False" and sometimes even numeric flags like "1" and "0".

Instead of treating them as generic nominal categories, they were normalised to integer indicators:

$$\texttt{Yes},\ \texttt{True},\ 1 \to 1, \qquad \texttt{No},\ \texttt{False},\ 0 \to 0$$

This choice has two benefits. First, it prevents a simple binary decision from exploding into multiple one–hot columns. Second, it conveys the natural ordering (0 vs 1) directly to models like Logistic Regression and SVM that expect numeric features.

Ordinal Encoding

The more interesting part of preprocessing was the manual mapping of ordinal categories. For example:

$$\text{work\_life\_balance: } \texttt{Poor} \to 0,\ \texttt{Fair} \to 1,\ \texttt{Good} \to 2,\ \texttt{Excellent} \to 3.$$

Similarly:

$$\text{venture\_satisfaction:} \texttt{Low} = 0,\ \texttt{Medium} = 1,\ \texttt{High} = 2,\ \texttt{Very High} = 3.$$

$$\text{startup\_stage:} \texttt{Entry} = 0,\ \texttt{Mid} = 1,\ \texttt{Senior} = 2.$$

$$\text{education\_background:} \texttt{High School} = 0,\ \texttt{Associate Degree} = 1,\ \texttt{Bachelor's Degree} = 2,\ \texttt{Master's D}$$

The key design decision here was to treat these as genuinely ordered scales, not just categories. One–hot encoding them would have hidden the natural progression (for example, between "Low" and "Very High") and would have created unnecessary dimensionality. Instead, the models can now use the numeric gap between levels as signal.

As a safety check, the preprocessing function also printed out any category values that did not appear in the mapping, so that unexpected labels (due to typos or hidden categories) could be identified early.

Outlier Treatment via IQR Capping

During EDA, boxplots of numeric features, especially `monthly_revenue_generated`, revealed extreme outliers, with some entries far above the bulk of the data. Leaving these untouched would make scaling unstable and could bias models that rely on distances or linear combinations.

Rather than deleting outlier rows—which might remove genuinely informative edge cases—we applied IQR-based capping. For each numeric column:

$$Q_1 = 25\text{th percentile}, \quad Q_3 = 75\text{th percentile}, \quad \text{IQR} = Q_3 - Q_1,$$

$$\text{lower cap} = Q_1 - 1.5 \cdot \text{IQR}, \quad \text{upper cap} = Q_3 + 0.7 \cdot \text{IQR}.$$

Values outside this band are clipped. The asymmetric choice (1.5 below, 0.7 above) reflects the observation that extreme high values are more common than extreme low values, especially for revenue and distance. The cap is therefore more conservative on the upper side to avoid distorting genuine high–performing founders.

The motivation is similar to the earlier transport–cost project, where heavy–tailed cost variables motivated robust handling rather than aggressive trimming.

Scaling and Encoding for Modelling

After imputation and encoding, different models needed slightly different final feature transformations:

- For the **Logistic Regression** and **SVM** pipelines written with scikit–learn's `ColumnTransformer`, numeric features were log–transformed (for highly skewed variables like revenue and distance) and then standardised with `StandardScaler`. Categorical features were one–hot encoded.
- For the **Neural Network** code path, after one–hot encoding of nominal features, a `RobustScaler` was applied to all features at once. This matches the NN's preference for roughly standardised, but outlier–robust, inputs.

A deliberate design choice was to keep the NN preprocessing relatively simple and uniform (one big numeric matrix), while the classical pipelines used a more explicit per–column treatment using transformers.

Feature Importance and Mutual Information

To understand which features actually carry information about retention, we computed mutual information between each input feature and the encoded target.

The top-ranked features by mutual information were:

- `startup_stage`
- `personal_status`
- `remote_operations`
- `work_life_balance_rating`
- `startup_reputation`
- `funding_rounds_led`

Lower-ranked but still nonzero contributions came from revenue, dependents, age and the various leadership and innovation indicators.

The fact that `startup_stage` and `personal_status` appear at the top is intuitive: founders at senior stages and with stable personal circumstances are more likely to stay. Interestingly, raw financial numbers such as monthly revenue are not as dominant as one might expect, suggesting that retention is driven by structural and lifestyle factors at least as much as short-term income.

Even features with small individual mutual information were not dropped immediately. In the earlier regression project, low–contributing features were pruned aggressively using Lasso and then removed for linear and ensemble models. Here, the task is classification, and certain models (particularly NNs and RBF–SVMs) can benefit from interactions between many weak features. Therefore, manual feature removal was kept to a minimum.

Modelling Strategy

Train–Validation Strategy

All three models share the same basic data split:

$$\text{Training set} : 80\% \ (47{,}688 \text{ rows}), \qquad \text{Validation set} : 20\% \ (11{,}923 \text{ rows}),$$

with **stratification** on the target label to preserve the Stayed/Left proportions in both splits.

For some models, an additional level of cross–validation was used:

- Logistic Regression: hyperparameters tuned with 3–fold cross–validation on the training split.
- SVM: performance estimated with 5–fold StratifiedKFold on the held–out validation set.

This "outer" validation set is used for the final fair comparison across models, mirroring the structure of the previous project where a fixed public test set was evaluated after tuning on the training folds.

Evaluation Metrics

The primary metric throughout is the **macro F1 score**. Unlike accuracy, macro F1 gives equal weight to both classes, even though "Stayed" is slightly more frequent than "Left".

This discourages trivial strategies that over–predict the majority class.

For the Logistic Regression and SVM pipelines we also monitored ROC–AUC across folds, mainly to check calibration and ranking performance, but the leaderboard score and main discussion focus on macro F1.

Confusion matrices are used to analyse which class each model struggles with, and whether misclassifications are symmetric.

Model 1: Logistic Regression

Motivation

Logistic Regression is the simplest baseline capable of handling thousands of one–hot encoded features while remaining interpretable. In the earlier regression project, linear models (Lasso, Ridge, ElasticNet) performed surprisingly well because the underlying relationship between features and cost was largely linear.

For founder retention, it is not clear a priori whether the decision boundary is linear. Running Logistic Regression first provides a sanity check: if it performs close to more complex models, the problem may be mostly linear; if it lags significantly, we need non–linear classifiers.

Pipeline and Hyperparameters

The Logistic Regression model was implemented as a scikit–learn `Pipeline`:

1. **Preprocessing** with a `ColumnTransformer` that:
    - imputes and scales numeric columns (with log transform for skewed variables),
    - imputes and one–hot encodes categorical columns,
    - passes through binary columns as numeric indicators.
2. A **LogisticRegression** classifier with:
    - penalty: L2,
    - solvers considered: `lbfgs`, `saga`,
    - class_weight: `balanced`,
    - max_iter: 500.

The regularisation strength C was tuned over $\{0.01, 0.1, 1, 3, 10\}$ using 3–fold cross–validation with macro F1 as the scoring metric.

The best configuration was:

$$C = 1, \quad \text{solver} = \texttt{saga}.$$

Validation Performance

On the held–out validation set of 11,923 founders, the tuned Logistic Regression achieved:

$$\text{Macro F1} = 0.7364, \quad \text{Accuracy} \approx 0.74.$$

The confusion matrix was:

$$\begin{bmatrix} 4199 & 1471 \\ 1669 & 4584 \end{bmatrix}$$

where rows correspond to true labels (`Left`, `Stayed`) and columns to predicted labels.

The model correctly identifies most of the "Stayed" founders, but it is slightly more inclined to predict "Stayed" than "Left", as shown by the 1,471 false "Stayed" predictions for founders who actually left. This mild bias is expected given the near–balanced but slightly skewed target and the use of class weights rather than explicit threshold tuning.

Model 2: Support Vector Machine (RBF Kernel)

Motivation

SVMs with RBF kernels are classic strong performers on medium–sized tabular datasets. They are particularly appealing when:

- The non–linear boundary is important.
- The number of features is moderate after encoding.
- The dataset is not so large that training becomes impossible.

In this project, training almost 60k samples with an RBF kernel is computationally heavy but still feasible on Kaggle hardware. The expectation was that SVM would outperform Logistic Regression if the retention decision truly depends on non–linear combinations of features.

Preprocessing and Model Setup

The SVM model used a similar preprocessing pipeline as Logistic Regression, with a few differences:

- Numeric columns were log–transformed where needed and then scaled.
- Ordinal columns were scaled as numeric.
- Nominal columns were one–hot encoded.

The classifier itself was an `SVC` with:

$$\text{kernel} = \texttt{rbf}, \quad C = 1.5, \quad \gamma = \texttt{scale}, \quad \text{probability} = \text{True}, \quad \text{class\_weight} = \texttt{balanced}.$$

Instead of a large grid–search (which would be prohibitively slow), a good configuration was chosen and evaluated using 5–fold StratifiedKFold on the validation split.

Performance and Behaviour

Across 5 folds on the validation set, the SVM achieved:

$$\text{Average Macro F1} \approx 0.7508, \quad \text{Average AUC} \approx 0.824.$$

The combined confusion matrix over all folds was:

$$\begin{bmatrix} 4117 & 1553 \\ 1562 & 4691 \end{bmatrix}$$

compared to Logistic Regression, SVM:

- slightly improves both true positives for "Stayed" and true negatives for "Left",
- reduces the asymmetry between the two classes,
- produces a smoother probability distribution for AUC.

The downside is training time: the SVM model took tens of minutes to fit, whereas Logistic Regression completed in a fraction of that time. In a production setting, this cost must be weighed against the roughly 1–1.5 percentage point improvement in macro F1.

Model 3: Neural Network for Tabular Data

Motivation

Neural Networks are attractive when we suspect many weak interactions between features, and when we want a single model family that can be scaled up later (deeper, wider, or with embeddings). While tree–based methods like XGBoost were central in the transport–cost regression project, here the focus is deliberately on a clean comparison between linear models, kernel methods, and dense NNs.

Architecture and Training Setup

The final NN used in the main experiments was a relatively simple **Tabular MLP**:

- Input: 33 features after encoding.
- Hidden layers: sizes 128 and 64.
- Each hidden layer uses Batch Normalisation and ReLU activation, followed by dropout (0.2).
- Output: 2 logits corresponding to the two classes.

Training hyperparameters:

- Optimiser: AdamW with learning rate 10^{-3}.
- Weight decay: 10^{-2} for regularisation.
- Batch size: 256.
- Loss: Cross–entropy.
- Early stopping: patience of 10 epochs based on validation macro F1.

The NN was trained on the same 80% training split, with 20% used for validation within the PyTorch training loop.

Validation Performance

The best validation performance across epochs was:

$$\text{Macro F1} \approx 0.7466, \quad \text{Accuracy} \approx 0.75.$$

The corresponding confusion matrix on the validation set was:

$$\begin{bmatrix} 4218 & 1452 \\ 1572 & 4681 \end{bmatrix}$$

This is very similar to the SVM confusion matrix. The NN tends to make slightly fewer "Left $\rightarrow$ Stayed" mistakes than Logistic Regression, but the difference is subtle.

An interesting observation from the training logs is that the NN improves over the logistic baseline within the first few epochs and then plateaus. Deeper or more complex variants (with residual blocks and cosine–annealing learning rate schedules) were also tested experimentally, but the straightforward two–layer network already matched or slightly exceeded them once early stopping and regularisation were tuned.

Model Comparison and Discussion

Quantitative Comparison

Table 1 summarises the main macro F1 scores on the validation data:

Model	Macro F1 (Validation)	Notes
Logistic Regression	0.736	Linear baseline, very fast
SVM (RBF Kernel)	**0.751**	Best overall, slowest to train
Neural Network (MLP)	0.747	Close to SVM, GPU-friendly

Table 1: Summary of model performances on the validation set.

The gap between Logistic Regression and the non–linear models is noticeable but not dramatic. This suggests that a significant portion of the signal is linear, with additional gains coming from modelling more complex patterns.

Qualitative Behaviour

A few qualitative patterns emerged from confusion matrices and score distributions:

- All models are slightly better at predicting "Stayed" than "Left". This might reflect true asymmetries in the data: staying behaviour could be more homogeneous, while leaving may be driven by diverse reasons, some of which are not captured in the features.
- Logistic Regression tends to under–predict the "Left" class more than SVM and NN, because a linear decision boundary in the transformed space still cannot carve out certain small pockets of "at–risk" founders.
- SVM and the NN tend to disagree on a small fraction of borderline cases. In those instances, SVM is often more conservative (predicting "Stayed"), whereas the NN sometimes predicts "Left" when certain non–linear combinations of features suggest risk.

Design Choices and Alternatives

Throughout the pipeline, several fork–in–the–road decisions were made. A few of these are worth emphasising:

Median vs Mean Imputation

Mean imputation is common but fragile in the presence of skewed numeric distributions. Because we observed long–tailed revenue and distance variables, median imputation was used consistently. This reduces the influence of a few very high–revenue founders on the imputed values.

Manual Ordinal Mapping vs Automatic Encoders

Libraries such as `category_encoders` offer ordinal encoders, but here the ordering had to be explicitly reasoned about. For example, the distance between "High" and "Very High" is conceptually smaller than between "Low" and "High".

Additional Model: CatBoost Classifier

To complement the baseline models (Logistic Regression, SVM and Neural Networks), an additional experiment was conducted using **CatBoost**. Since the founder–retention dataset is tabular and contains several categorical attributes, CatBoost is well-suited due to its native

handling of categorical features through ordered target statistics, minimal preprocessing requirements, and strong performance on structured data.

Training Configuration

The model was trained using 5-fold cross-validation on the 20% validation split. Key settings included:

- Loss function: `Logloss`
- Task: binary classification (`Stayed` vs. `Left`)
- Iterations: 4000 (with early stopping)
- Depth: 6
- Learning rate: adaptive (CatBoost default)

CatBoost was trained using the preprocessed feature matrix with redundant columns removed. The model reached strong separation early in training (typically at 300–600 iterations), after which early stopping prevented overfitting.

Cross-Validation Results

CatBoost achieved consistently strong performance across all folds. The validation F1 and AUC values for the 5 folds were:

$$\begin{aligned}
&\text{Fold 1: } F1 = 0.7601, \quad AUC = 0.8408,\\
&\text{Fold 2: } F1 = 0.7616, \quad AUC = 0.8367,\\
&\text{Fold 3: } F1 = 0.7667, \quad AUC = 0.8471,\\
&\text{Fold 4: } F1 = 0.7656, \quad AUC = 0.8447,\\
&\text{Fold 5: } F1 = 0.7752, \quad AUC = 0.8545.
\end{aligned}$$

The combined metrics across all folds were:

$$\text{Average F1} = \mathbf{0.7658}, \qquad \text{Average AUC} = \mathbf{0.8448}.$$

These values place CatBoost among the top-performing models for this prediction task, achieving both high discrimination (AUC) and balanced performance across classes (F1 score).

Confusion Matrix

Aggregating predictions from all validation folds gives the combined confusion matrix:

$$\begin{bmatrix} 4178 & 1490 \\ 1448 & 4804 \end{bmatrix}$$

This indicates strong recall for both classes, with well-balanced prediction behaviour—especially notable given the natural imbalance in founder retention outcomes.

Conclusion

This project shows that startup founder retention can be predicted reasonably well using standard machine learning methods, provided that the preprocessing and encoding steps are carefully designed.

The main conclusions are:

- Clean preprocessing — especially correct ordinal mapping, boolean normalisation and sensible outlier handling — is essential. The models depend critically on the quality of this numeric representation.
- Logistic Regression already provides a strong and interpretable baseline with macro F1 around 0.736 and is suitable when training time or transparency is more important than squeezing out the last few percentage points.
- SVM with an RBF kernel achieves the best performance (macro F1 around 0.751) but at a high computational cost. It is best viewed as a high–accuracy classical reference model.
- A modest two–layer Neural Network matches SVM–level performance within a small margin and opens the door for more advanced deep–learning approaches tailored to tabular data.