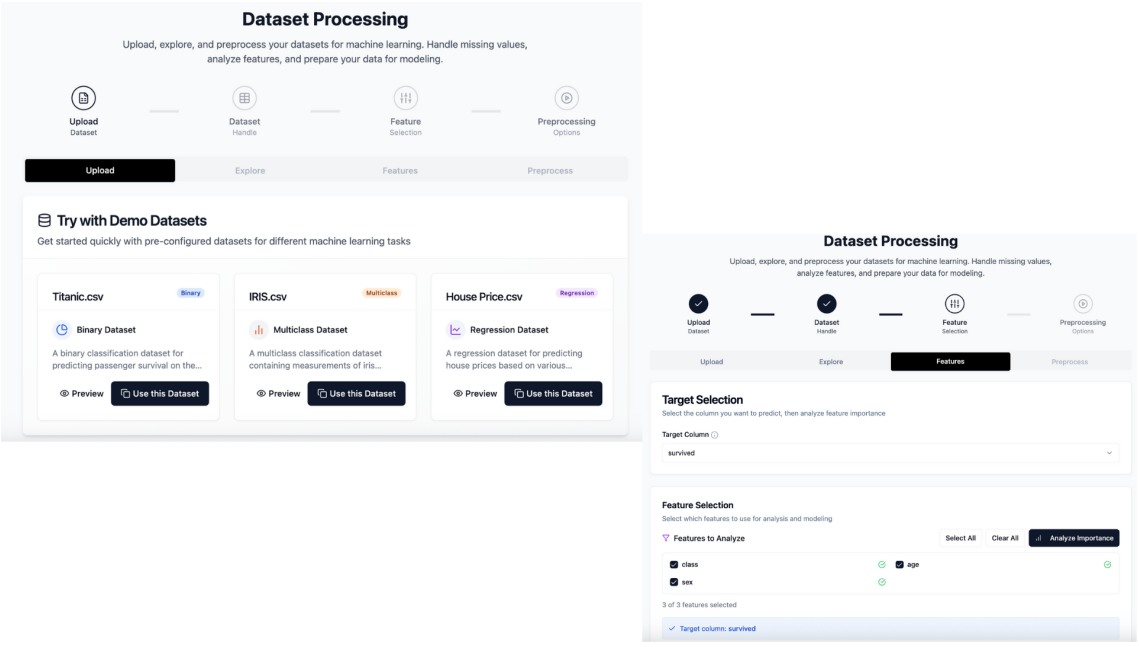
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

International Hellenic University
Department of Information and Electronic Engineering

DIPLOMA THESIS

ieeAutoML: A Web Application for Automated Machine Learning for Supervised Learning



Student:
Kyriakos Ouzounis
Student ID: (174974)

Supervisor:
(Stefanos Ougiaroglou)

31-08-2025

ieeAutoML: A Web Application for Automated Machine Learning for Supervised Learning

Code of Dissertation: (25184)

Student's full name: Kyriakos Ouzounis

Supervisor's full name: (Stefanos Ougiaroglou)

Date of undertaking: 13-03-2025

Date of completion: 31-08-2025

We hereby affirm the authorship of this thesis and the acknowledgement of all assistance received. All sources of data, ideas, figures, or text—quoted or paraphrased—are properly cited. We further affirm that this work has been prepared exclusively by the undersigned student as a diploma thesis at the Department of Information and Electronic Engineering of the International Hellenic University.

Abstract

The thesis introduces **ieeAutoML** as a web application which performs automated machine learning for supervised learning tasks. The system provides users with a simple interface to handle datasets and perform preprocessing and feature selection and model training and evaluation and prediction tasks. The system combines AutoML engines with custom training capabilities through ieeAutoML to deliver high performance to non-experts. The application uses Python FastAPI as its modular backend and React as its frontend with Supabase serving as the database and storage solution. The system undergoes extensive testing which proves its usability and efficiency and accessibility.

Dedication

First and foremost, I would like to express my heartfelt gratitude to my family for their unwavering encouragement and support throughout the course of my studies. I am equally thankful to the friends who stood by me and made my student years truly memorable.

Finally, I owe special thanks to my thesis supervisor, Assistant Professor Mr. Stefanos Ougiaroglou, for his invaluable guidance, trust, and support during the preparation and completion of this thesis.

Contents

List of Figures

List of Tables

Introduction

1.1 Supervised Learning (Classification)

The fundamental machine learning paradigm of supervised learning trains predictive models through labeled data [50, 55]. The main objective of this thesis focuses on classification tasks which require each input instance to have a specific class label for the model to learn correct label assignments for new unseen data. The established framework exists in multiple domains including medical diagnosis and spam detection and customer segmentation. The most popular classification algorithms include logistic regression and decision trees and ensemble methods such as random forests and gradient boosting and neural networks. The models vary in their interpretability features and computational requirements and their ability to detect nonlinear patterns yet they all aim to achieve minimal misclassification errors with strong generalization capabilities for new data.

1.2 Automated Machine Learning (AutoML)

The goal of Automated Machine Learning (AutoML) is to simplify the entire machine learning process so that it becomes more accessible and efficient [2, 18]. AutoML platforms automate the key steps of data preprocessing, feature selection, algorithm selection, hyperparameter tuning and model evaluation so that users can build high-performing models with minimal manual intervention. This automation enables non-expert users to effectively use machine learning and enables experienced practitioners to scale their workflows and maintain consistency. The open-source frameworks MLJAR AutoML and H2O AutoML demonstrate this by providing easy-to-use workflows together with explainability, feature importance and leaderboard outputs.

1.3 Motivation

AutoML requires user-friendly tools that integrate explainability with transparency features and both automated and custom training capabilities within a web-based system to achieve its full potential. The current solutions direct their functionality toward expert users while lacking visual interpretability and operate as standalone local applications. The main purpose of *ieeAutoML* is to establish a web-based platform which serves both beginners and experts. The system aims to create an easy-to-use modular interface which simplifies supervised learning workflows through flexible model selection and provides clear visual feedback throughout the pipeline. During the early exploration phase, I experimented with tools such as Auto-sklearn and TPOT but quickly realized that their

command-line and notebook-based workflows were not accessible to non-experts. This motivated me to design *ieeAutoML* as a browser-based solution with visual feedback, bridging the gap between expert flexibility and beginner usability.

1.4 Contribution

The web application *ieeAutoML* presents a solution to make supervised classification workflows accessible to everyone. The main contributions consist of:

- The system provides a single web interface which enables users to select custom algorithms including logistic regression and gradient boosting as well as automated model pipelines through MLJAR and H2O AutoML.
- The system provides a clear preprocessing pipeline that handles missing values and scales features and balances classes and detects tasks while maintaining full dataset version tracking.
- The web UI contains integrated interpretability features which include SHAP visualizations together with model leaderboards and summary reports.
- The system uses React and FastAPI and Supabase and containerized backend services to build a modular and scalable architecture which supports asynchronous model training and future team-based deployments.

1.5 Thesis Organization

The thesis follows this specific structure:

- The second chapter of the thesis presents data preprocessing techniques which include imputation and feature selection and scaling and balancing methods for supervised learning.
- The third chapter of this paper explains the classification algorithms available in *ieeAutoML* by discussing their fundamental principles and advantages and disadvantages as well as their implementation and exposure within the system.
- The fourth chapter explains the technologies used which include frontend and backend frame- works and database integration and deployment tools.
- The fifth chapter explains AutoML frameworks and their integration with MLJAR and H2O AutoML and explains why these frameworks were chosen.
- The sixth chapter explains the technical details of preprocessing pipelines and model train- ing workflows and modular architecture.
- The seventh chapter explains the system design through user requirements and flow diagrams and schema modeling.
- The eighth chapter provides an extended overview of the implementation by explaining ser- vice structure and component integration and training flows.

- The ninth chapter demonstrates the user interface features of *ieeAutoML* by showing how users handle datasets and train models and evaluate results and make predictions.
- The tenth chapter provides a summary of contributions and limitations and future direc- tions for research.

Data Preprocessing for Supervised Learning

2.1 Data Imputation

Real-world datasets frequently contain missing entries, which can hinder the training process and degrade the performance of machine learning algorithms [31, 37]. To address this issue, *ieeAutoML* integrates a variety of imputation techniques, enabling users to select the approach that best matches the nature and distribution of their data. During dataset upload, the application allows the specification of a custom missing value symbol. Once the dataset is loaded, the system scans for both standard indicators of missingness (e.g., NaN, Null) and the user-defined symbol. A summary table then presents the percentage of missing values for each column, allowing users to assess the extent of the problem and choose an appropriate strategy.

Definition 2.1 (Data Imputation)**.** *Data imputation refers to the process of replacing missing values in a dataset with substituted values based on statistical or similarity-based heuristics, with the aim of preserving dataset completeness without introducing bias [3].*

The system supports several imputation methods. For numerical data, the mean or median value of the column can be used to fill in missing entries, with the median offering greater robustness in the presence of outliers [17]. Mode imputation, which replaces missing values with the most frequent entry, is applicable to both numerical and categorical variables. A more advanced option is Hot Deck imputation, which identifies similar records using a k-nearest neighbors algorithm and estimates missing values based on these donor records [34, 17]. Users may also choose to drop rows containing missing data entirely or skip imputation altogether if they wish to handle it externally.

In the Hot Deck approach, categorical variables are encoded and distance metrics such as Euclidean or Manhattan are applied to determine similarity between records. This versatility ensures that *ieeAutoML* can handle a broad spectrum of dataset types while maintaining methodological soundness [37].

Implementing Hot Deck imputation was particularly demanding, as it required careful encoding of categorical variables while avoiding data leakage. Ensuring both correctness and efficiency of this method proved to be one of the most technically challenging aspects of the preprocessing pipeline.

Table 2.1: Overview of Imputation Strategies in `ieeAutoML`

Method	Description	Applicable to
Mean	Replaces missing values with the column mean. Best for symmetric numerical distributions.	Numerical
Median	Substitutes with the column median. More robust to outliers than the mean.	Numerical
Mode	Fills with the most frequent value in the column. Works for both data types.	Numerical & Categorical
Hot Deck (k-NN)	Estimates missing values based on similar entries (nearest neighbors). Uses encoded categorical features.	Numerical & Encoded Categorical
Drop Rows	Removes rows that contain at least one missing value. Simple but potentially lossy.	Any
Skip	Skips imputation entirely and keeps missing values as-is. Useful for manual or external handling.	Any

2.2 Feature Selection

The majority of large complex datasets include multiple redundant or unimportant features that fail to enhance predictive power. The model training process becomes longer and accuracy suffers and the model becomes more prone to overfitting when redundant variables are kept [11, 6]. The purpose of feature selection methods is to locate significant predictors so models can perform better with reduced dimensionality and improved interpretation [38].

Definition 2.2 (Feature Selection)**.** *The procedure known as feature selection identifies the essential input variables that contribute to prediction success while discarding those that provide no meaningful predictive value. The objective of this procedure is to optimize model performance while reducing the dataset size and making it simpler to understand [22, 38].*

After completing missing value imputation in *ieeAutoML* the feature selection process begins. The system reveals the available features to users for selection after they choose their target column. The "Analyze Importance" function determines how much each selected feature affects the target variable by displaying its findings through a ranked feature importance chart. Users have full flexibility to change their selections at any point by modifying either the target variable or selecting different features until they proceed to preprocessing.

The system enhances both computation speed and clarity because feature selection occurs before the preprocessing phase. The elimination of uninformative variables enables the system to direct processing resources toward predictive attributes which show promise for accurate prediction [38, 11].

2.3 Class Balancing Techniques

Classification datasets commonly present an unbalanced class distribution when one class appears significantly more than others. Such imbalance causes models to predict the

dominant class repeatedly leading to misleading high accuracy while the model performs poorly on minority class predictions [27]. The system in *ieeAutoML* conducts automated class distribution checks during preprocessing followed by offering both oversampling and undersampling methods when it detects imbalanced data.

Definition 2.3 (Class Balancing)**.** *Class balancing consists of adjusting target class distributions to minimize majority class bias so the model achieves better generalization across all classes.*

Oversampling techniques boost the quantity of observations belonging to minority classes. Random oversampling duplicates existing minority samples to increase their numbers [7]. The more advanced techniques used in oversampling include SMOTE which creates new minority samples by interpolation [12] and Borderline SMOTE which targets samples close to decision boundaries [23] and ADASYN which produces synthetic data for challenging minority instances [26]. The dataset with mixed numerical and categorical variables can benefit from SMOTE-NC.

The approach of undersampling decreases the overall number of instances in the majority class. The process of random undersampling or Edited Nearest Neighbors (ENN) which removes instances based on k-nearest neighbors [53] serves to perform this method. Tomek Links identify overlapping instances from the majority class for removal [51] and the Neighborhood Cleaning Rule (NCR) utilizes ENN to eliminate majority-class neighbors of misclassified minority samples [35].

Not all methods are applicable to all datasets. The implementation of SMOTE, ADASYN and ENN distance-based techniques requires numerical features in the dataset. The available methods for datasets made exclusively of categorical variables in *ieeAutoML* include random oversampling and undersampling because these methods are methodologically valid.

2.4 Feature Scaling and Normalization

The process of feature scaling becomes crucial for multiple algorithms because they depend on distance-based calculations or need normally distributed inputs. The absence of scaling makes features with extensive numerical ranges dominate the model [15]. Research shows that appropriate normalization techniques produce significant improvements in the performance of support vector machines and neural networks [47].

Definition 2.4 (Feature Scaling)**.** *The definition of feature scaling describes the method to transform numerical data into equivalent scales which maintains their original range differences for better algorithm stability and convergence [52].*

The *ieeAutoML* system shows scaling options during the final preprocessing stage following missing value treatment and feature selection and class balance assessment. The system performs automatic detection of numerical features before enabling scaling only when these features exist. The available scaling methods consist of Min–Max Scaling which transforms features into the range [0,1] and Standard Scaling (Z-score normalization) which standardizes features to zero mean and unit variance and Robust Scaling which uses median and interquartile range for outlier resistance and Log Transformation which reduces skewness in highly skewed distributions. Users have the option to choose skipping scaling altogether.

The system allows users to select from different scaling methods while enabling them only when necessary conditions exist to provide both flexibility and best practice preprocessing guidelines. The targeted approach enhances model training efficiency and predictive accuracy particularly for algorithms which are sensitive to feature magnitude [15, 47].

Classification algorithms

3.1 Logistic Regression

3.1.1 Overview

Logistic Regression is a widely used linear classification algorithm for binary outcomes. Instead of predicting a numeric value, it models the probability that an input belongs to the positive class (label "1") versus the negative class ("0") [30]. This is achieved using the logistic (sigmoid) function, which transforms the output of a linear equation into a value between 0 and 1 [24]. The result is interpretable as a probability of class membership [29]. A threshold (commonly 0.5) is applied to this probability to decide the predicted class [5].

3.1.2 Mechanics

The logistic regression model computes a linear combination of the input features (and an optional intercept term), then applies the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ to produce a probability score [24, 5]. The model is trained by minimizing a loss function called log-loss (or binary cross-entropy), which penalizes incorrect predictions and encourages the model to output accurate probability estimates [43]. Optimization is commonly performed using algorithms such as gradient descent [5].

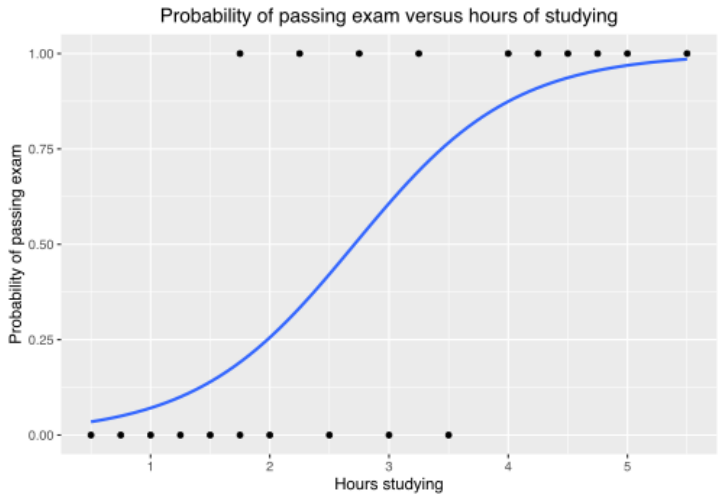


Figure 3.1: Sigmoid curve fitted to data showing the probability of passing an exam versus hours studied. The curve illustrates how logistic regression maps input features to probabilities between 0 and 1.

3.1.3 Advantages

Logistic regression offers several notable advantages. It is highly interpretable and easy to implement, as the learned coefficients directly indicate the influence of each feature on the predicted outcome [30]. The method performs well on linearly separable datasets, where a straight line or hyperplane can effectively divide the two classes [24]. Furthermore, although logistic regression is inherently designed for binary classification, it can be readily extended to multiclass problems using approaches such as One-vs-Rest (OvR) or multinomial logistic regression [5].

3.1.4 Limitations

Despite its strengths, logistic regression has limitations that must be considered. It assumes a linear relationship between the features and the log-odds of the outcome, making it less effective for datasets with nonlinear decision boundaries unless transformations or interaction terms are introduced [24]. Additionally, the algorithm is sensitive to multicollinearity, where high correlations between input features can destabilize coefficient estimates, and it can be adversely affected by outliers, which may disproportionately shift the decision boundary [29].

3.1.5 Available Hyperparameters in *ieeAutoML*

In the custom training interface of the *ieeAutoML* system, users can adjust the following logistic regression hyperparameters:

- `penalty`: Type of regularization applied (`l1`, `l2`, `elasticnet`, or `none`).
- `C`: Inverse of regularization strength (smaller values imply stronger regularization).
- `solver`: Optimization algorithm used (`lbfgs`, `liblinear`, etc.).
- `max_iter`: Maximum number of iterations for model convergence.
- `fit_intercept`: Whether to include an intercept in the model.
- `class_weight`: Option to apply balanced class weights to handle imbalance.

3.2 Linear Regression

3.2.1 Overview

Linear Regression stands as one of the essential algorithms which performs regression operations. The algorithm creates a linear equation to explain how dependent variables relate to one or multiple independent variables when analyzing observed data [30, 24]. The algorithm works best for predictive tasks because it detects linear relationships between inputs and outputs.

3.2.2 Mechanics

The model calculates a weighted combination of input features together with an optional intercept value. The model seeks to reduce the difference between forecasted and actual values through the Mean Squared Error (MSE) loss function [5]. The learned coefficients indicate the amount each input feature affects the output variable.

The basic version of this model (simple linear regression) uses one input variable for prediction. The model extends its functionality to process multiple features at once in multiple linear regression [42].

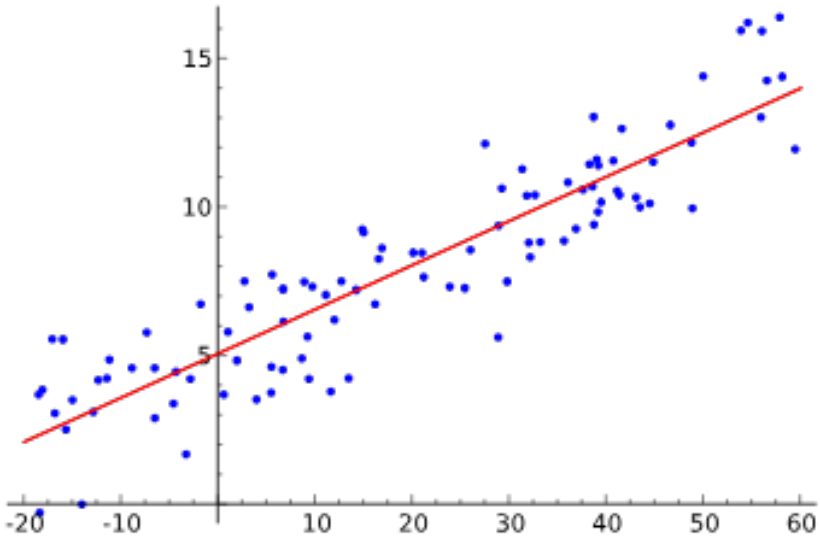


Figure 3.2: Example of a linear regression model fitted to a set of data points. The straight line represents the best-fit line minimizing the residuals between actual and predicted values.

3.2.3 Advantages

The model coefficients in linear regression provide direct interpretation of feature effects on predicted outcomes because training and interpretation occur quickly [30]. The model achieves its best results when the target variable shows a linear relationship with input features because it generates accurate predictions in these cases. The model serves as a standard reference point for complex regression techniques because of its straightforward nature [42].

3.2.4 Limitations

Linear regression remains useful despite its several important restrictions. The model fails to detect nonlinear patterns because it requires a direct linear connection between predictors and target variables [5]. The algorithm becomes unstable when dealing with multicollinear features because they distort coefficient estimates and when encountering outliers that affect the regression line [30]. The model requires two essential assumptions: constant variance (homoscedasticity) and normally distributed errors because violations of these assumptions decrease model performance.

3.2.5 Available Hyperparameters in *ieeAutoML*

In the *ieeAutoML* app, Linear Regression is available only for regression tasks. The custom training interface allows users to adjust the following hyperparameters:

- `fit_intercept`: Boolean flag indicating whether to include an intercept.
- `n_jobs`: Number of parallel threads for training (optional).

3.3 Decision Tree

3.3.1 Overview

Decision Trees function as a non-parametric supervised learning algorithm which serves both classification and regression needs [9, 49]. The algorithm uses a flowchart structure which includes decision rules at internal nodes that lead to rule outcomes through branches and leaf nodes that predict class labels or values.

3.3.2 Mechanics

The algorithm performs recursive dataset splitting through the feature which produces the highest information gain or impurity reduction. The classification tasks use *Gini Impurity* and *Entropy* (information gain) metrics to determine the best feature for splitting [49, 9]. The tree expansion stops when it meets predefined criteria which include maximum depth limits or minimum sample requirements per split or when nodes achieve purity by containing only one class of samples. The hierarchical structure of Decision Trees enables them to handle complex decision boundaries and feature interactions.

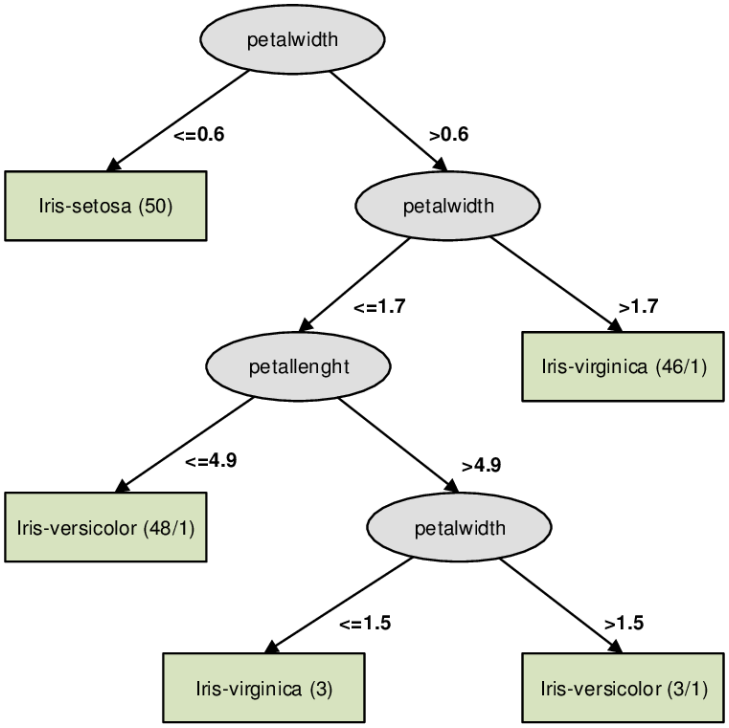


Figure 3.3: Example of a decision tree trained on the Iris dataset, visualizing the series of decision splits (internal nodes) based on flower features and the resulting leaf nodes with class outcomes.

3.3.3 Advantages

The interpretable nature of Decision Trees combined with their visual simplicity allows non-experts to understand them [54, 30]. The recursive structure of Decision Trees allows them to detect non-linear patterns in data which provides flexibility for modeling complex relationships. The algorithm works with both numerical and categorical variables without needing extensive preprocessing steps including feature scaling or normalization. The model works well with different types of data and problem domains because of its versatility and simplicity.

3.3.4 Limitations

Decision Trees tend to overfit when they are not constrained from growing deep [9]. The algorithm produces different tree structures and predictions because of small changes in the data. Decision Trees show a preference for features with distinct values because these features tend to control splitting decisions unless additional measures prevent this effect.

3.3.5 Available Hyperparameters in *ieeAutoML*

In the *ieeAutoML* app, Decision Tree models are available for both classification and regression tasks through the custom training interface. Users can modify the following hyperparameters:

- `max_depth`: The maximum depth the tree is allowed to grow.

- `min_samples_split`: The minimum number of samples required to split an internal node.

These parameters allow users to control model complexity and mitigate overfitting, adapting the tree to different dataset characteristics.

3.4 Random Forest

3.4.1 Overview

Random Forest is an ensemble learning algorithm that constructs multiple decision trees during training and outputs the mode of their predictions for classification tasks or the mean prediction for regression tasks. It improves upon single decision trees by reducing variance and overfitting through averaging [8, 28].

3.4.2 Mechanics

The algorithm builds a collection of decision trees using random subsets of the training data and feature space (bagging and feature sampling). Each tree is trained independently on a different bootstrap sample. During prediction, the model aggregates the predictions of all trees to produce a final output [56, 8]. This diversity among trees allows the Random Forest to generalize better and handle noise more effectively than a single decision tree [25].

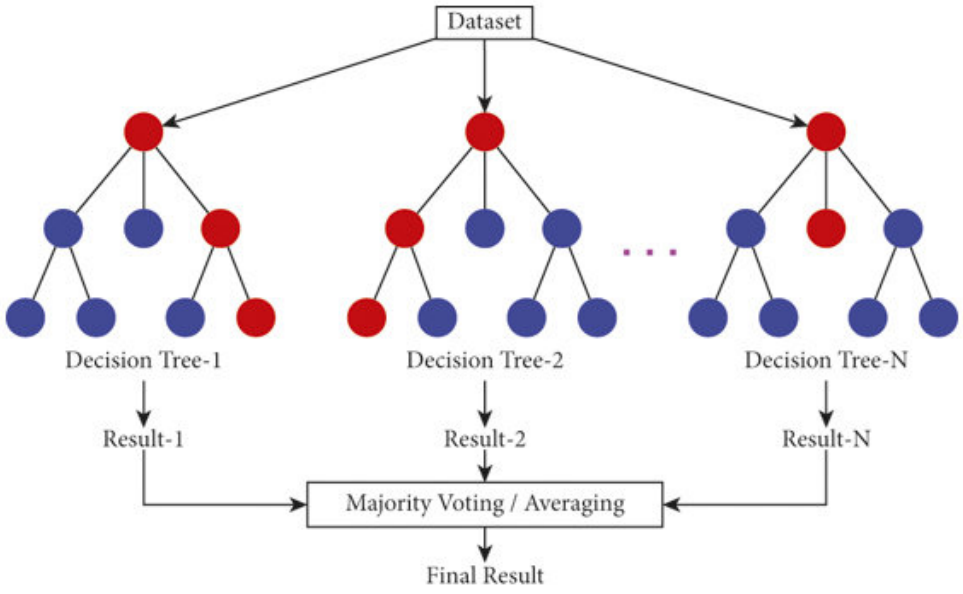


Figure 3.4: Illustration of Random Forest architecture: multiple decision trees are trained on random subsets of data and features, and their outputs are aggregated.

3.4.3 Advantages

Random Forest models are robust to overfitting, as the ensemble approach significantly reduces the high variance seen in individual decision trees [8]. They work particularly

well with high-dimensional datasets, handling many features without requiring aggressive preprocessing [25]. Another notable advantage is the algorithm's ability to estimate feature importance, offering valuable insights into the contribution of each variable [39]. Furthermore, Random Forests handle missing values and outliers more gracefully than many linear models, making them suitable for a wide range of real-world datasets [32].

3.4.4 Limitations

The primary drawback of Random Forests is their reduced interpretability compared to single decision trees, as the ensemble nature makes it difficult to visualize the overall decision-making process [8]. They can also be computationally expensive, both in terms of processing time and memory usage, particularly when training on large datasets with many trees. Additionally, their performance gains diminish when working with very small datasets, as the benefits of aggregation rely on having enough diversity among the trees.

3.4.5 Available Hyperparameters in *ieeAutoML*

In the *ieeAutoML* app, Random Forest is available for classification tasks under the custom training option. Users can modify the following default hyperparameters:

- `n_estimators`: Number of trees in the forest (default: 100).
- `max_depth`: Maximum depth of each tree (default: 7).

3.5 XGBoost

3.5.1 Overview

XGBoost (Extreme Gradient Boosting) represents a scalable and efficient implementation of gradient-boosted decision trees. The system was created to maximize both computational speed and predictive accuracy. XGBoost constructs models through sequential stages while optimizing a regularized objective function that combines loss and complexity terms [13]. The system demonstrates resistance to overfitting and works well with structured tabular data.

3.5.2 Mechanics

XGBoost uses gradient boosting with second-order optimization which utilizes the first and second derivatives of the loss function to direct tree construction. The algorithm constructs a decision tree ensemble through sequential steps where each new tree learns from the remaining errors of the previous ensemble. XGBoost achieves exceptional performance through several key innovations including L1 and L2 regularization for better generalization and learning rate shrinkage for tree influence control and feature subsampling for diversity improvement and sparsity-aware split finding for efficient handling of missing or sparse data. The combination of these improvements results in better accuracy together with stability and scalability.

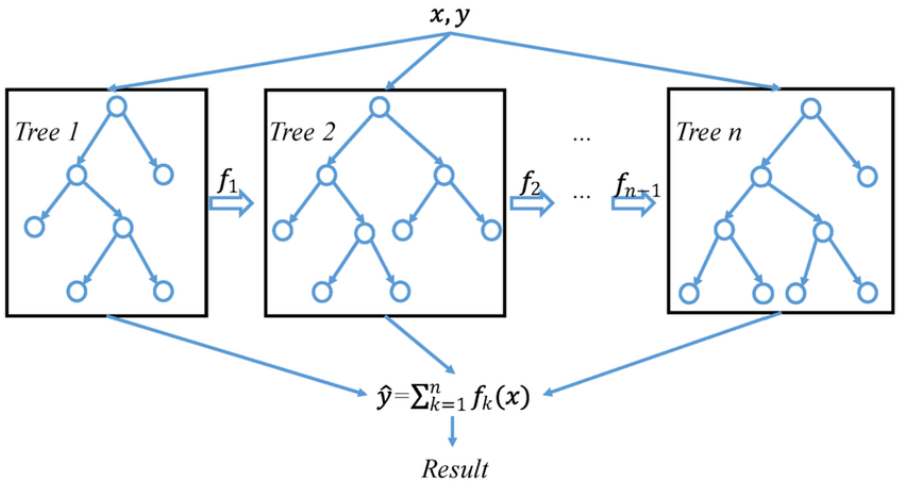


Figure 3.5: Simplified XGBoost architecture showing multiple trees trained sequentially to minimize loss.

3.5.3 Advantages

XGBoost demonstrates better performance than multiple alternative machine learning algorithms for structured and tabular dataset prediction. The system achieves better results than traditional gradient boosting techniques because of its integrated regularisation functions which reduce overfitting. The system contains built-in functions which handle sparse data and missing values without needing additional data preparation steps. XGBoost provides parallel and distributed computing features which speed up training processes for big datasets.

3.5.4 Limitations

The setup of XGBoost hyperparameters requires expertise together with knowledge of its complex system architecture. A single decision tree model provides better interpretability than the ensemble model because the ensemble structure conceals the paths of individual decisions. The memory requirements of XGBoost exceed those of basic models while its computational needs are higher which restricts its use with large datasets or systems with limited resources.

3.5.5 Available Hyperparameters in `ieeAutoML`

XGBoost is accessible for classification tasks through the Custom Training interface in the `ieeAutoML` system. The available adjustable parameters include:

- `n_estimators`: Number of boosting rounds.
- `learning_rate`: Shrinkage rate (e.g., 0.01–0.3).
- `max_depth`: Maximum depth of individual trees.

3.6 LightGBM

3.6.1 Overview

LightGBM (Light Gradient Boosting Machine) represents a fast efficient gradient boosting framework that Microsoft developed for processing large datasets of high dimensionality [33]. The algorithm uses histogram-based binning with leaf-wise tree growth to speed up the training process without compromising predictive accuracy.

3.6.2 Mechanics

LightGBM accelerates both speed and accuracy through its two main operational strategies. The **histogram-based splitting** method converts continuous features into discrete bins that help both memory management and improve the speed of finding optimal split points. The **leaf-wise (best-first) tree growth** method selects the leaf node with the highest gain to expand rather than the standard level-by-level approach which leads to more accurate trees for equal numbers of splits [33]. The innovative combination of efficient parallelization with these two improvements makes LightGBM effective for handling both big and sparse data sets.

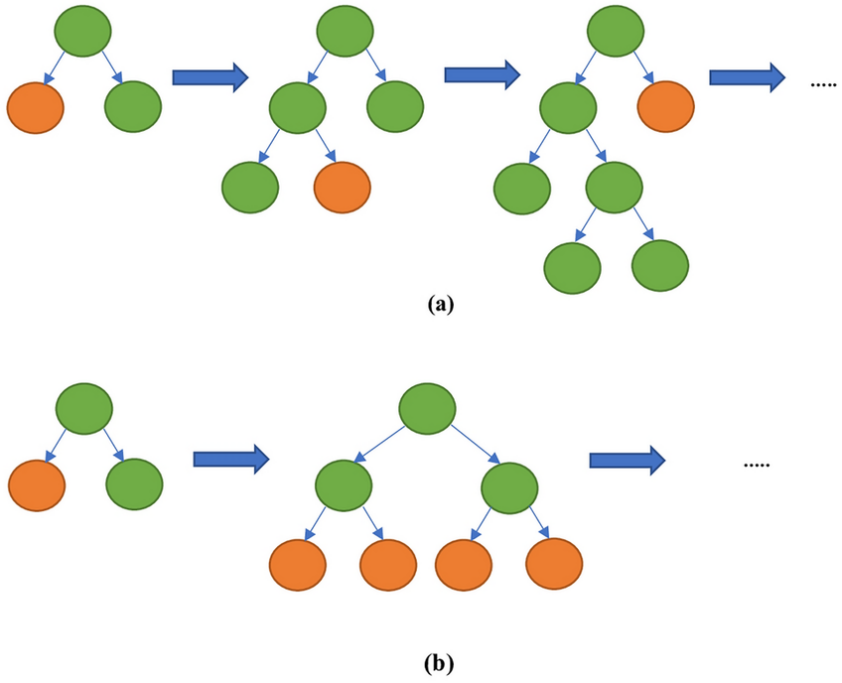


Figure 3.6: Comparison of LightGBM's leaf-wise (right) versus level-wise (left) tree growth strategies.

3.6.3 Advantages

The algorithm provides four benefits to users because it handles large datasets quickly through histogram-based binning and parallelization and it produces accurate predictions by selecting optimal leaf nodes and it requires less memory than traditional gradient boosting techniques and it handles categorical features natively without requiring one-hot encoding [33].

3.6.4 Limitations

LightGBM achieves high performance but this comes with the danger of overfitting when used on small datasets without proper regularization techniques. The model lacks the interpretability of a single decision tree and shows sensitivity to data encoding decisions for categorical and numerical features [33].

3.6.5 Available Hyperparameters in *ieeAutoML*

The *ieeAutoML* interface allows users to set up LightGBM for classification tasks by selecting:

- `n_estimators`: Number of boosting rounds.
- `learning_rate`: Tree learning rate.
- `max_depth`: Maximum depth per tree.

3.7 CatBoost

3.7.1 Overview

Yandex developed CatBoost as a state-of-the-art gradient boosting algorithm which operates natively with categorical data [48]. The training process of CatBoost includes built-in categorical feature handling which eliminates the need for extensive data preparation work. The two main features of **ordered boosting** protect against target leakage and the **efficient categorical encodings** system maintains crucial information to reduce overfitting.

3.7.2 Mechanics

The gradient boosting framework of CatBoost employs two essential features to construct its models. **ordered boosting** determines residuals through permutation-based subsets of training data to prevent any observation from having its target value included in its calculation. The procedure successfully minimizes the amount of target leakage which occurs during the model training process.The combination of both strategies provides better accuracy while enhancing model generalization [48]. Second, **efficient categorical encoding** transforms categorical variables using statistics derived only from previous observations in a given permutation, thereby mitigating the risk of overfitting commonly associated with ordinal encoding methods [48]. Additionally, CatBoost uses **symmetric (balanced) trees**, where all leaves are at the same depth, improving inference speed and contributing to model regularization.

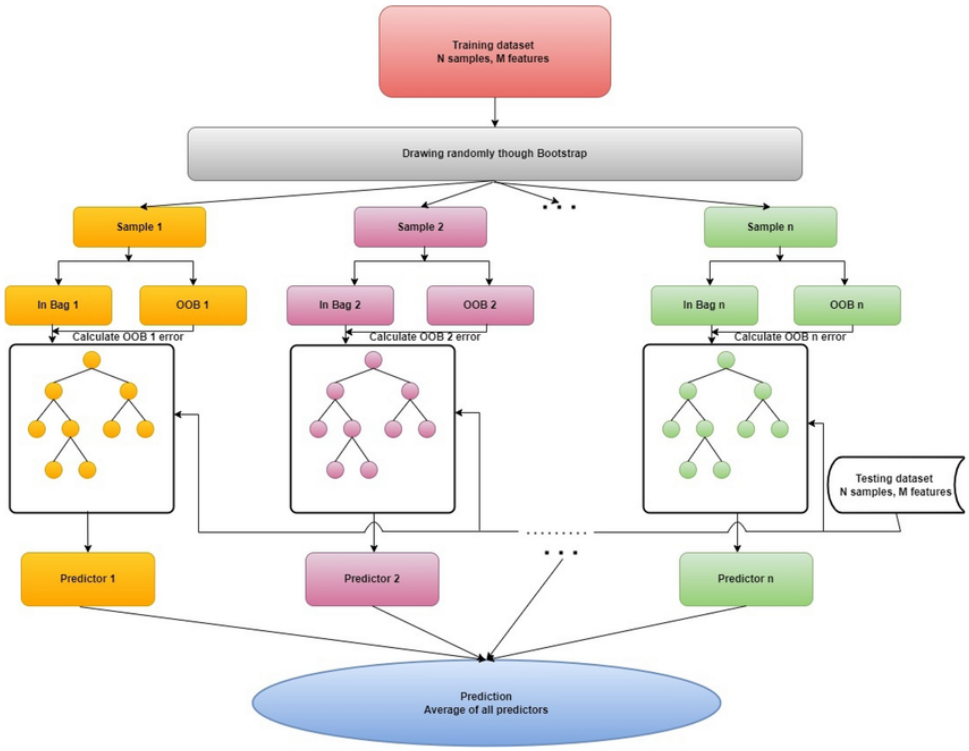


Figure 3.7: Flowchart of the CatBoost training process: ordered boosting with bootstrap sampling and feature splitting to build symmetric trees.

3.7.3 Advantages

CatBoost provides multiple significant advantages to users. The algorithm supports categorical features natively so users do not need to perform explicit preprocessing especially when working with structured datasets [48]. The ordered boosting mechanism in CatBoost helps prevent target leakage while reducing overfitting [48]. The practical performance of CatBoost matches XGBoost and LightGBM when working with structured data benchmarks. The symmetric tree structure of the model enables fast and efficient prediction times.

3.7.4 Limitations

The training process of CatBoost becomes longer than LightGBM when dealing with extremely large datasets. The process of achieving optimal performance requires proper adjustment of its hyperparameters. The built-in objective functions of CatBoost remain limited compared to alternative solutions which restrict its flexibility for specific tasks.

3.7.5 Available Hyperparameters in *ieeAutoML*

In the *ieeAutoML* interface, CatBoost is available for classification tasks with the following configurable hyperparameters:

- `iterations`: Number of boosting rounds (default: 100).
- `depth`: Maximum number of layers per tree (default: 6).

- `learning_rate`: Shrinkage factor controlling the contribution of each tree (default: 0.1).

3.8 Neural Networks / MLP Classifier

3.8.1 Overview

The Multi-Layer Perceptron (MLP) architecture in Neural Networks functions as a specific feedforward artificial neural network designed for supervised classification problems. The neural network architecture of MLP uses multiple interconnected neurons spread across layers to detect nonlinear patterns which exceed linear method capabilities [21].

3.8.2 Mechanics

The standard MLP architecture includes an input layer followed by one or multiple hidden layers before reaching the output layer. The weighted inputs of each neuron combine into a sum which then passes through a nonlinear activation function such as ReLU. The training process uses backpropagation with gradient-based optimization methods (Adam or SGD) to reduce the cross-entropy loss function [4].

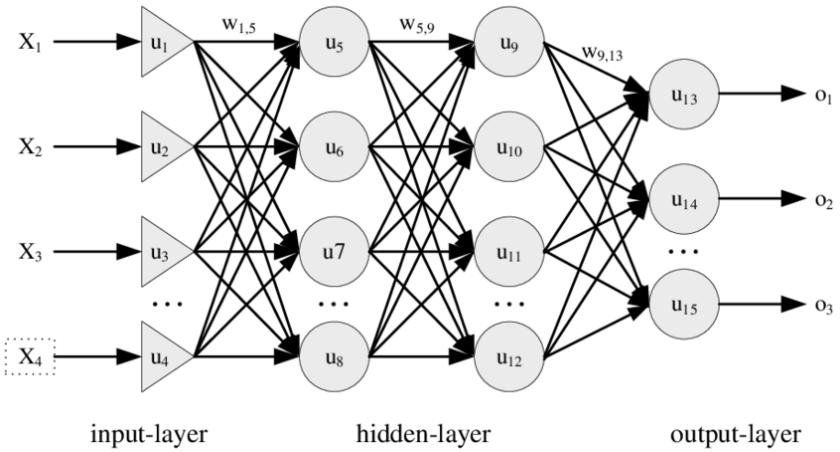


Figure 3.8: Architecture of a simple MLP classifier with one hidden layer. Each neuron is connected to all neurons in the subsequent layer.

3.8.3 Advantages

MLPs possess the ability to identify intricate nonlinear patterns together with multiple feature relationships which other models cannot detect. The adaptable structure of MLPs enables them to handle various tasks and data formats including both structured tabular data and unstructured image data. Their broad application range makes them a strong general-purpose learning framework which solves multiple classification problems.

3.8.4 Limitations

MLPs need extensive training data to achieve their best results and maintain effective generalization. The computational requirements of MLPs are high which typically needs

GPU acceleration or extended training periods for big datasets. The lack of transparency in their decision-making process has led to their classification as a "black box" model which makes interpretation difficult.

3.8.5 Available Hyperparameters in `ieeAutoML`

- `hidden_layer_sizes`: Tuple specifying the number of neurons in each hidden layer (e.g., [64, 32]).
- `activation`: Activation function, such as `relu` or `tanh`.
- `solver`: Optimization method, such as `adam` or `sgd`.
- `alpha`: L2 regularization strength.

3.9 Extra Trees Classifier

3.9.1 Overview

The ensemble learning method Extremely Randomized Trees (Extra Trees) operates through decision trees. The process follows the Random Forest methodology but introduces additional randomness to split decisions. Instead of seeking the most discriminative threshold, the algorithm selects random thresholds for a subset of features before training each individual tree on the complete dataset [20].

3.9.2 Mechanics

Extra Trees functions through a model of multiple unpruned trees which learns from the full dataset without bootstrapping. During each split process the model randomly chooses features before setting random split thresholds instead of employing optimization algorithms. The combination of multiple trees in the approach produces diverse trees that reduce overall model variance and this leads to better performance on datasets with noise [20].

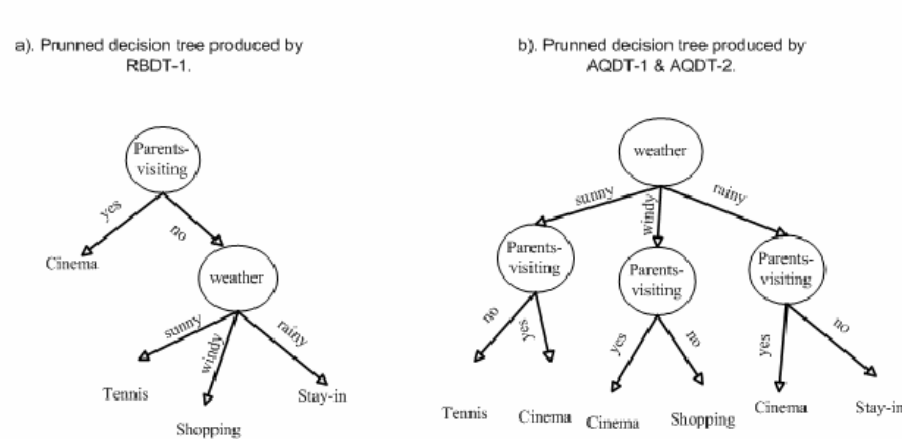


Figure 3.9: Illustration of Extra Trees: random splits across multiple unpruned trees increase variance reduction compared to single decision trees.

3.9.3 Advantages

The training speed of Extra Trees depends on its random split selection method which avoids expensive threshold optimization procedures. The ensemble model shows better resistance to high variance than tree ensemble models and delivers stable results in noisy conditions and large feature datasets.

3.9.4 Limitations

The random nature of Extra Trees decision-making makes it challenging to understand the model's internal process compared to basic models. The performance of optimized ensemble methods including Gradient Boosting and Random Forest exceeds Extra Trees when working with properly structured datasets. The model's performance deteriorates when training data consists of only a limited number of examples.

3.9.5 Available Hyperparameters in `ieeAutoML`

ieeAutoML provides the following set of hyperparameters for the user to select:

- `n_estimators`: Number of trees in the ensemble.
- `max_depth`: Maximum depth allowed for each tree.
- `max_features`: Number of features to consider at each split (e.g., `"sqrt"` or `"log2"`).

3.10 K-Nearest Neighbors (K-NN) Classification

3.10.1 Overview

K-NN represents a supervised classification algorithm based on non-parametric principles which stands for K-Nearest Neighbors. The class prediction for a query point results from locating its k nearest neighboring points in feature space followed by selecting the class that appears most often among them [1].

3.10.2 Mechanics

K-NN operates by predicting outcomes through comparisons of new instances to every training example. The k closest points are determined through a chosen metric (e.g., Euclidean) and then the predicted class emerges from majority voting. A small k value leads to unstable decision boundaries while an increased k value results in more generalized and smoother boundaries.

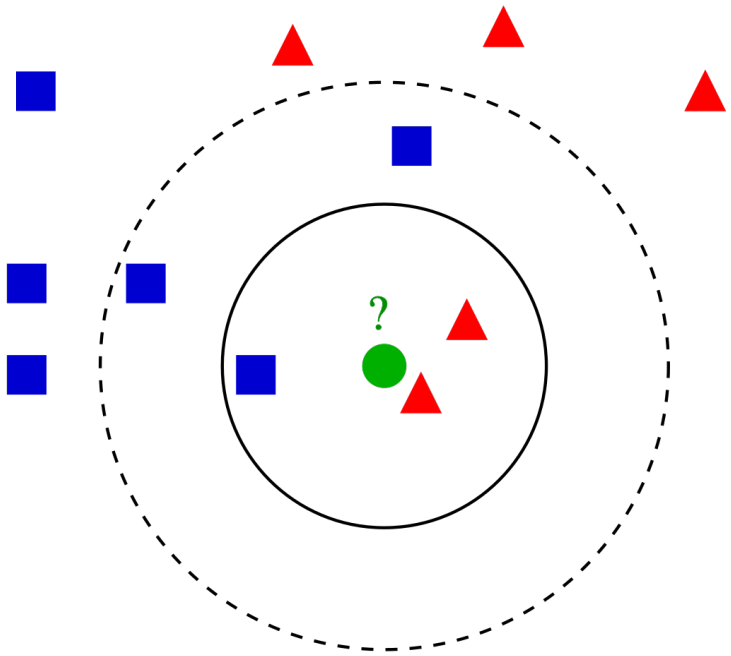


Figure 3.10: Decision boundary formed by K-NN classifier for $k = 3$. Regions are colored based on nearest neighbors in the training set.

3.10.3 Advantages

K-NN maintains simplicity in its design structure which makes the algorithm straightforward to comprehend and implement. As a non-parametric method, it does not require assumptions about the underlying data distribution. The model provides users with flexibility to modify its behavior through k and distance metric adjustments.

3.10.4 Limitations

The time required for prediction in large datasets becomes expensive because K-NN needs to compute distances from all training instances. The model's performance suffers from irrelevant features together with scale variations in features so preprocessing becomes crucial. The predictive power of this approach declines substantially when operating in high-dimensional spaces due to the curse of dimensionality.

3.10.5 Available Hyperparameters in `ieeAutoML`

- `n_neighbors`: Number of nearest neighbors (k) to use for classification.
- `weights`: Weighting function for neighbor contributions (`"uniform"` or `"distance"`).
- `metric`: Distance metric used for neighbor calculation (e.g., `"euclidean"` or `"manhattan"`).

Languages and Technologies

The *ieeAutoML* web application operates through a set of essential technological frameworks which allow the system to complete end-to-end model development and assessment. The system implements an advanced full-stack structure, combining interactive visualizations with machine learning algorithms and AutoML tools and cloud platform capabilities. The choice of technologies was guided by practical experimentation rather than convenience. For instance, FastAPI was selected over Django due to its asynchronous support, which proved essential for handling long-running training tasks, while Supabase was preferred to Firebase because of its PostgreSQL foundation, which better suited experiment tracking requirements.

4.1 Programming Languages

The main programming language for backend development includes **Python** which supports the creation of APIs and machine learning models and data processing pipelines. The application depends on **JavaScript** and **TypeScript** for its frontend development needs. Through JavaScript developers can build interactive UI components which establish backend service interactions while TypeScript enhances developer tooling through type safety to improve maintainability while minimizing runtime errors.

4.2 Frontend Technologies

The interface development benefits from contemporary JavaScript tools which enable developers to create performance-oriented user interfaces by following design principles alongside modular programming standards.

Core Framework and Build Tools

The main frontend library is React 18.3.1 which allows developers to build highly interactive user interfaces. The development process and build operations rely on Vite which enables fast development times and optimized output. The entire frontend codebase uses TypeScript for static type checking and React DOM ensures efficient browser component rendering.

Styling and UI Components

Tailwind CSS functions as the application's styling mechanism through its utility-first CSS framework which enables fast and consistent design. The application uses Radix UI to obtain accessible component primitives like modals and menus while Shadcn/UI provides pre-styled Radix UI components that work without a headless configuration. The animation and transition functionality relies on Framer Motion alongside Tailwind Animate for utility-based animation classes. The interface enables users to switch between light and dark themes through Next Themes.

Routing and State Management

The application uses React Router DOM for client-side routing to provide seamless navigation that replaces entire page reloads. TanStack Query (React Query) manages server-side data state by providing data fetching along with caching and synchronization functions. The process of form handling and validation becomes easier through React Hook Form which decreases re-rendering overhead.

Charts and Visualization

The application uses Chart.js and React Chart.js 2 to display training metrics and performance results in dynamic and responsive charts. Recharts operates as a declarative charting library for React to deliver different visualization requirements.

Backend Integration

Supabase JS functions as an API client for database queries and user authentication to provide secure and efficient backend service interactions.

Other UI Utilities

The application implements Lucide React for icons as well as CMDK for command menus and Embla Carousel for sliders and carousels. The application uses CVA (Class Variance Authority), CLSX and Tailwind Merge class management utilities for implementing consistent and conditional styling. Zod functions as a schema validation tool for forms together with API requests. User notifications operate through Sonner and Date-fns serves as a tool for managing dates within the application.

4.3 Backend Technologies

The **FastAPI** web framework based on Python enables RESTful API and asynchronous communication in the backend. The **Uvicorn** ASGI server serves FastAPI endpoints. The validation of requests along with schema enforcement is performed by **Pydantic**. The database layer utilizes **SQLAlchemy** for ORM operations which enables PostgreSQL asynchronous access. The **httpx** library acts as the tool for the backend to make asynchronous HTTP requests for external storage interactions. Machine learning models get serialized before storage through **joblib**, **pickle**, **shutil** and **zipfile**.

4.4 Database and Storage

Supabase functions as a backend-as-a-service solution that provides PostgreSQL-based database storage together with authentication and file hosting capabilities. The PostgreSQL database system within Supabase maintains all datasets, experiments and models as well as user information. The Supabase Storage platform stores experiment artifacts such as model files and generated visualizations while the Supabase Auth system handles JWT-based authentication and role-based access control.

4.5 Machine Learning Libraries

The machine learning capabilities draw their power from a blend of traditional ML libraries and AutoML frameworks. The core classification algorithms exist in **Scikit-learn** yet **XGBoost** and **LightGBM** and **CatBoost** deliver optimized gradient boosting methods for custom training. The automated model selection process with leaderboard generation occurs through **MLJAR AutoML** and **H2O AutoML** while H2O AutoML provides superior model stacking and explainability capabilities. The interpretability tool **SHAP** determines feature importance through Shapley value calculations. The **Imbalanced-learn** library solves class imbalance problems through its SMOTE oversampling technique.

4.6 DevOps and Deployment

The application benefits from contemporary DevOps methods to maintain scalability and reliability along with portability in its deployment process. The **Docker** container system enables application deployment while **NGINX** and **Apache** work together to receive requests and distribute frontend assets. Development environments benefit from deployment through **Render Railway** and **ngrok** platforms. **GitHub** manages version control and CI/CD pipelines and environment-specific configuration is handled through `.env` files that store environment variables.

4.7 Third-Party Tools and Integrations

Several third-party tools enhance the application's capabilities. Supabase provides the integrated database, authentication, and storage services. **Pandas** and **NumPy** are used extensively for data manipulation and preprocessing, while **Matplotlib** and **Seaborn** generate visualizations such as confusion matrices and ROC curves. Finally, **Mistral AI** is integrated to provide intelligent assistant responses and suggestions within the application.

Automated Machine Learning Tools

AutoML functions as a methodology which automates the entire process of machine learning application to real-world problems [46]. The process includes data preprocessing followed by feature selection and model selection and hyperparameter tuning and evaluation [40]. AutoML tools enable non-technical users to access machine learning capabilities and boost the work efficiency of experienced practitioners and produce uniform results [46].

The *ieeAutoML* web application implements AutoML functionality through its training pipeline following preprocessing operations. Users can select between two training options on the training page: custom model training and fully automated training. Users who select AutoML need to choose between MLJAR [41] and H2O AutoML [36] engines before selecting the training preset type (*"Fast"*, *"Balanced"* or *"Accurate"*) and entering an experiment name. Users have the ability to modify test split ratio settings and data stratification for classification tasks and random seed with default value 42. The selected engine starts building and evaluating models automatically after the user initiates the process.

5.1 AutoML Tools

The current market offers multiple AutoML frameworks which deliver distinct capabilities. The tools differ in their approach between providing interfaces that are easy to use and maximizing computational power and scalability. The Python-based MLJAR AutoML produces optimized results for tabular data and generates clear interpretation reports [41]. The H2O AutoML platform from H2O.ai provides scalable performance with robust explainability features and algorithmic support [36]. Auto-sklearn extends scikit-learn through Bayesian optimization and ensemble learning to perform automated model selection [19]. The genetic programming-based TPOT system evolves optimal pipelines [44], but Google AutoML and Microsoft Azure AutoML serve enterprise-level machine learning deployments [40]. The decision to integrate MLJAR and H2O AutoML into *ieeAutoML* stemmed from their open-source nature and high accuracy and strong explainability support [41, 36].

5.2 H2O AutoML

The open-source framework H2O AutoML developed by H2O.ai enables handling of large datasets and enterprise-scale tasks [36]. The framework trains multiple models including Gradient Boosting Machines (GBM), Generalized Linear Models (GLM), Random

Forests and Deep Neural Networks and then employs stacked ensemble methods for model combination. The `max_runtime_secs` parameter functions as a training time controller to determine the maximum allowed execution duration. The system includes features that support classification task class balancing and performs cross-validation as well as automatic training-validation split generation and performance-based early stopping and built-in model interpretability tools which include variable importance analysis and SHAP summaries. After training completion H2O generates a performance leaderboard showing the performance of each model. The system generates two artifacts from the leaderboard and visual explanations which *ieeAutoML* stores for user access [36].

5.3 MLJAR AutoML

The open-source machine learning system MLJAR AutoML performs full automation of the entire process from data preprocessing to feature engineering through model training and interpretation [41]. The system shows exceptional performance with structured/tabular data and operates with multiple algorithms including LightGBM, Random Forest, Extra Trees, XGBoost and Logistic Regression. MLJAR provides three training modes which consist of *Explain* for fast interpretable model training and *Perform* for balanced speed and accuracy and *Compete* for extended training to achieve advanced tuning and stacking. The system creates detailed reports which include feature importance analysis along with SHAP explanations and model leaderboard scores and complete training settings. The *ieeAutoML* system selects an MLJAR configuration based on its preset which ensures the best-performing model gets saved and visualized.

5.4 Other Tools

The AutoML capabilities of MLJAR and H2O AutoML face competition from several alternative frameworks. Auto-sklearn extends scikit-learn capabilities through Bayesian optimization to select models and adjust hyperparameters automatically [19]. TPOT employs evolutionary algorithms to search optimal pipelines which deliver high performance though they need substantial computational power [44]. The commercial solution set of Google Cloud AutoML includes vision along with language and tabular data processing capabilities [46], while Microsoft Azure AutoML provides users with a graphical interface to track experiments and tune models before deploying them to the cloud [40]. Although these tools deliver excellent enterprise-ready features the scope of *ieeAutoML* centers on open-source solutions with strong explainability and ease of integration.

Algorithms implementation using Pytnon and scikit-learn

6.1 Data Preprocessing

The process of data preprocessing stands essential before starting machine learning model training. The Python-based preprocessing pipeline of `ieeAutoML` backend depends on `scikit-learn` and `imbalanced-learn` libraries for its implementation. The preprocessing pipeline consists of type detection and task identification followed by missing value imputation and normalization and class balancing.

6.1.1 Data Type Detection and Feature Classification

The system applies heuristics to determine whether features are categorical or numerical. The system identifies float columns with low cardinality and integer-only values as categorical features. Object-type columns undergo evaluation for numeric patterns during inspection.

Listing 6.1: Detecting feature types

```
def detect_data_types_and_features(df):
    for col in df.columns:
        dtype = df[col].dtype
        if is_float_dtype(dtype) and df[col].nunique() <= 15:
            categorical_features.append(col)
        elif is_integer_dtype(dtype):
            if df[col].nunique() <= 20:
                categorical_features.append(col)
            else:
                numerical_features.append(col)
        ...
```

6.1.2 Task Type Detection

The type of machine learning task is inferred based on the number of unique values and the data type of the target column.

Listing 6.2: Detecting task type

```
def detect_task_type(df, target_column):
    target_series = df[target_column].dropna()
```

```
    unique_count = target_series.nunique()
    if unique_count == 2:
        return "binary_classification"
    elif unique_count <= 50:
        return "multiclass_classification"
    else:
        return "regression"
```

6.1.3 Dataset Overview and Missing Value Detection

The system produces an overview which displays total missing values together with column types and unique value counts. The system transforms custom missing symbols ‘‘?’’, ‘‘NA’’, ‘‘null’’ into NaN for uniformity.

6.1.4 Missing Value Imputation

The `clean_data` function offers multiple imputation approaches.

- **Mean, Median, Mode:** The system uses statistical imputation to replace missing values with mean, median or mode.
- **Drop:** The system removes all rows containing missing values through the Drop method.
- **Hot Deck:** The Hot Deck method uses K-Nearest Neighbors to locate comparable rows which then replace missing values.
- **Skip:** The Skip method maintains missing values in their original state.

Listing 6.3: Hot deck imputation with KNN

```
knn = NearestNeighbors(n_neighbors=3)
knn.fit(complete_rows.values)
indices = knn.kneighbors([row_filled.values], return_distance=
    ↪ False)
donor_row = df_copy.loc[indices[0][0]]
```

6.1.5 Normalization

The system accepts different scaling methods which normalize numerical features into equivalent scales.

- **Min-Max Scaling**
- **Standard Scaling**
- **Robust Scaling**
- **Log Transformation**
- **Skip**

Listing 6.4: Applying standard normalization

```
scaler = StandardScaler()
transformed = scaler.fit_transform(values).flatten()
df_copy[col] = np.round(transformed, decimals)
```

6.1.6 Class Balancing

The `balance_dataset` function provides capabilities to perform both under- and over-sampling for handling imbalanced classification datasets.

- **Undersampling:** The methods for undersampling include Random, ENN, Tomek Links, and NCR while the methods for oversampling,
- **Oversampling:** include Random, SMOTE, Borderline SMOTE, and ADASYN.

Listing 6.5: Balancing using SMOTE

```
sampler = SMOTE(random_state=42)
X_res, y_res = sampler.fit_resample(X_encoded, y)
df_balanced = pd.concat([X_res, y_res], axis=1)
```

6.1.7 Integration in ieeAutoML

The `ieeAutoML` app presents these preprocessing elements through a user-friendly interface. The backend maintains consistent handling of missing values and normalization methods and balancing strategies while the frontend lets users make interactive configuration choices prior to starting the training process.

6.2 Classification Algorithms

The `ieeAutoML` system provides support for supervised classification model training through `scikit-learn` framework. After dataset selection and preprocessing the system provides the option to perform customized training with multiple available algorithms. The supported algorithms include Logistic Regression, Decision Trees, Random Forests, Gradient Boosted Trees (XGBoost, LightGBM, CatBoost) and Neural Networks (MLP) together with Extra Trees and k-Nearest Neighbors.

6.2.1 Training Workflow

The training process begins with a defined workflow which consists of:

- The dataset needs to be split between training and testing data sets.
- Categorical features require encoding through the LabelEncoder process.
- The system requires the selection of a classification algorithm.
- The model requires either default or customized hyperparameters for application.

- The model receives training from the training set data.
- The model receives assessment on the test data through accuracy, precision, recall, F1-score and log-loss measurements.

A simplified implementation is shown below:

Listing 6.6: Model training logic (simplified)

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, log_loss
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

# Encode target
le = LabelEncoder()
y_encoded = le.fit_transform(df[target_column])

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    df[features], y_encoded, test_size=0.2, stratify=y_encoded,
       ↪ random_state=42
)

# Train model
model = RandomForestClassifier(n_estimators=100, max_depth=10,
   ↪ random_state=42)
model.fit(X_train, y_train)

# Evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average="weighted")
```

6.2.2 Hyperparameter Handling

Each algorithm accepts a dictionary of hyperparameters passed by the user. The default settings will be used if the user does not provide any values. An example demonstrates how to supply custom parameters for training an XGBoost classifier:

Listing 6.7: Custom hyperparameters example

```
from xgboost import XGBClassifier

custom_params = {
    "n_estimators": 150,
    "max_depth": 5,
    "learning_rate": 0.05
}

model = XGBClassifier(**custom_params)
model.fit(X_train, y_train)
```

6.2.3 Supported Algorithms

The `ieeAutoML` platform provides the following classification algorithms:

- Logistic Regression (`LogisticRegression`)
- Decision Tree (`DecisionTreeClassifier`)
- Random Forest (`RandomForestClassifier`)
- XGBoost (`XGBClassifier`)
- LightGBM (`LGBMClassifier`)
- CatBoost (`CatBoostClassifier`)
- Neural Network (MLP) (`MLPClassifier`)
- Extra Trees (`ExtraTreesClassifier`)
- k-Nearest Neighbors (`KNeighborsClassifier`)

The classifiers are initialized dynamically based on the algorithm name provided in the training configuration. Results are saved alongside performance metrics and visualizations such as confusion matrices and SHAP plots.

6.3 AutoML Algorithms Implementation

The `ieeAutoML` application supports training with Automated Machine Learning (AutoML) frameworks in addition to manual algorithm selection. Users can select between two integrated engines: **MLJAR** and **H2O AutoML**. After preprocessing, users navigate to the training interface, where they select the AutoML engine, a training preset (`fast`, `balanced`, or `accurate`), configure test size, toggle stratification (for classification tasks), and specify a random seed (default 42).

6.3.1 MLJAR AutoML Integration

The MLJAR engine uses the `mljar-supervised` library, offering efficient training and evaluation of multiple models under unified settings. The following code shows the configuration and training logic:

Listing 6.8: MLJAR AutoML training

```
from supervised.automl import AutoML

automl = AutoML(
    mode="Perform",
    total_time_limit=training_params["time_limit"],
    algorithms=training_params["algorithms"],
    eval_metric=training_params["metric"],
    random_state=training_params["random_state"],
    train_ensemble=True,
    verbose=True
```

```
)

automl.fit(X_train, y_train)

# Save and compress model if large
model_file = f"mljar_model_{experiment_id}.pkl"
joblib.dump(automl, model_file)
```

Presets dynamically define which algorithms are included, the time limit, and evaluation metric (e.g., `logloss` for classification, `rmse` for regression). After training, the best-performing model is selected from the MLJAR leaderboard.

6.3.2 H2O AutoML Integration

H2O AutoML is integrated using the `h2o` Python client. It performs automatic training of a variety of models, including stacked ensembles. The following code illustrates H2O training:

Listing 6.9: H2O AutoML training

```
import h2o
from h2o.automl import H2OAutoML

h2o.init()

hf = h2o.H2OFrame(df)
train, test = hf.split_frame(ratios=[0.8], seed=42)

aml = H2OAutoML(
    max_runtime_secs=training_params["time_limit"],
    seed=42,
    include_algos=training_params["algorithms"]
)

aml.train(y=target_column, training_frame=train)

# Export model
model = aml.leader
h2o.save_model(model=model, path="model_output", force=True)
```

The H2O platform automatically detects tasks and enables model stacking capabilities. The system provides additional capabilities through leaderboard retrieval and `h2o.explain()` visualization and prediction API functionality. Integrating H2O AutoML required substantially more effort than MLJAR, particularly due to the different model export formats such as MOJO files. Adapting the prediction pipeline to handle these cases highlighted the importance of anticipating framework-specific constraints when designing a unified system.

6.3.3 Model Export and Evaluation

The best model receives export and evaluation treatment before storage with its associated metrics. The model gets compressed when needed. The system generates visual-

izations that include confusion matrices together with ROC/PR curves and SHAP plots and metric summaries before uploading them to backend storage.

6.3.4 Example Output

AutoML experiment training generates structured metadata which summarizes the complete training process. The metadata contains model type details (`XGBoost` or `StackedEnsemble`) along with accuracy and F1-score and log-loss evaluation metrics and leaderboard file rankings and SHAP explanations visualizations. Users can access high-performing models together with performance summaries through a system which removes the need for manual algorithm selection and hyperparameter tuning.

Design of ieeAutoML

7.1 Requirements (User Stories)

The design and functionality of **ieeAutoML** system receives direction from user stories that represent actual use cases and user expectations. These stories span the entire workflow — from dataset handling to model training, evaluation, and prediction.

1. **Dataset Overview.** The user needs to upload a dataset for viewing its structural overview which includes information about column types and missing values and numerical and categorical distribution.

2. **Missing Value Imputation.** A user needs a system to handle missing values in datasets through standard imputation methods and obtain the cleaned dataset for additional analysis.

3. **Feature Importance.** The user requires feature importance calculation based on target column selection for evaluation of feature inclusion before starting the training process. The system allows users to save and download the updated dataset.

4. **Normalization and Balancing.** The system allows users to execute dataset preprocessing methods that combine normalization techniques with class balancing methods for exporting the trained model or external use.

5. **Custom Model Training.** Users need to train their model through a customized algorithm by choosing between default or manually set hyperparameters to achieve optimal results for their specific task.

6. **AutoML Training.** Users can train their datasets through AutoML engines H2O and MLJAR by choosing predefined profiles (e.g. fast, balanced, accurate) which perform automatic model selection and optimization.

7. **Experiment Results.** The user needs access to experiment results which include essential performance metrics and visual elements including confusion matrix and SHAP and ROC and model files and leaderboards and prediction tables for download.

8. **Model Prediction.** After model training completion users need to make predictions by entering feature values manually or uploading a CSV file with the ability to assess prediction accuracy when ground truth labels exist.

9. **Experiment Management and Tuning.** Users need access to view their finished experiments alongside the ability to separate them by Custom or AutoML types and adjust custom experiment parameters before running the training pipeline again.

10. **Experiment Comparison.** The user needs functionality to compare different experiments through established metrics that consider training type and task type (binary/multiclass classification or regression) to select the most powerful model.

7.2 Flow Charts

Dataset Processing and Training Workflow

The `ieeAutoML` system demonstrates its complete data preparation and training process through Figure 7.1. The workflow starts with dataset upload followed by conditional missing value handling. The system enables users to choose a target column for previewing feature importance before selecting to save their refined dataset. Users can perform normalization and class balancing operations after applying preprocessing techniques before starting the training process. Users have two training choices: Custom or AutoML with distinct training paths. The diagram demonstrates key decision points together with modular flexibility that exists throughout the data processing and training process.

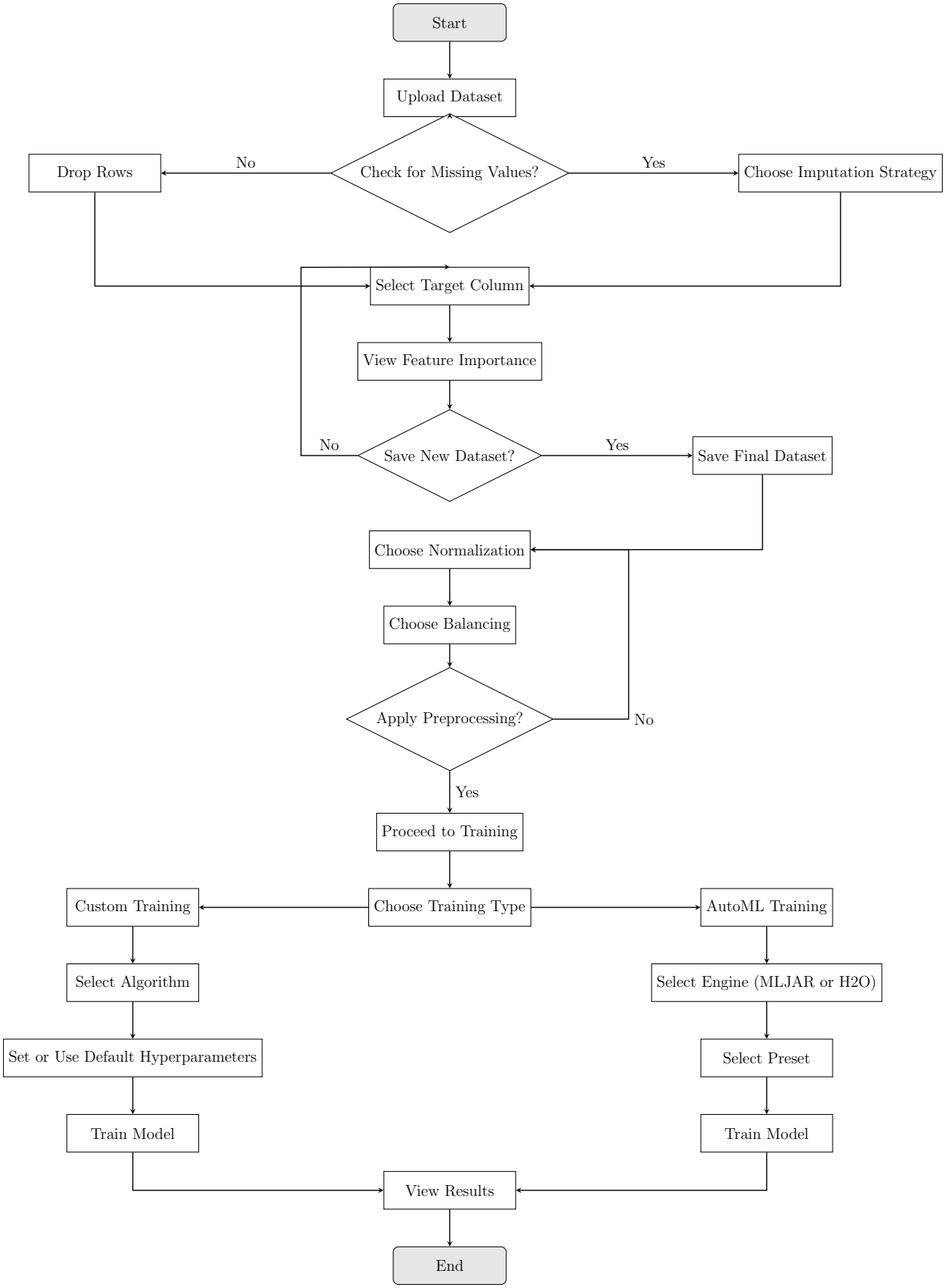


Figure 7.1: Workflow of the `ieeAutoML` application from dataset upload to training

Training Workflow – Custom vs AutoML

The training component separates into two distinct workflows which are Custom Training and AutoML Training according to Figure 7.2. Users who follow the Custom path choose an algorithm and decide between using default or customized hyperparameters. Users define their experiment details through metadata inputs that include name selection and test split configuration along with stratification options and random seed settings and optional visualization choices. The AutoML training process enables users to choose between MLJAR or H2O engine selection and predefined profile options before starting background asynchronous training. The training paths unite at a single result presentation point which maintains uniform output and user interaction.

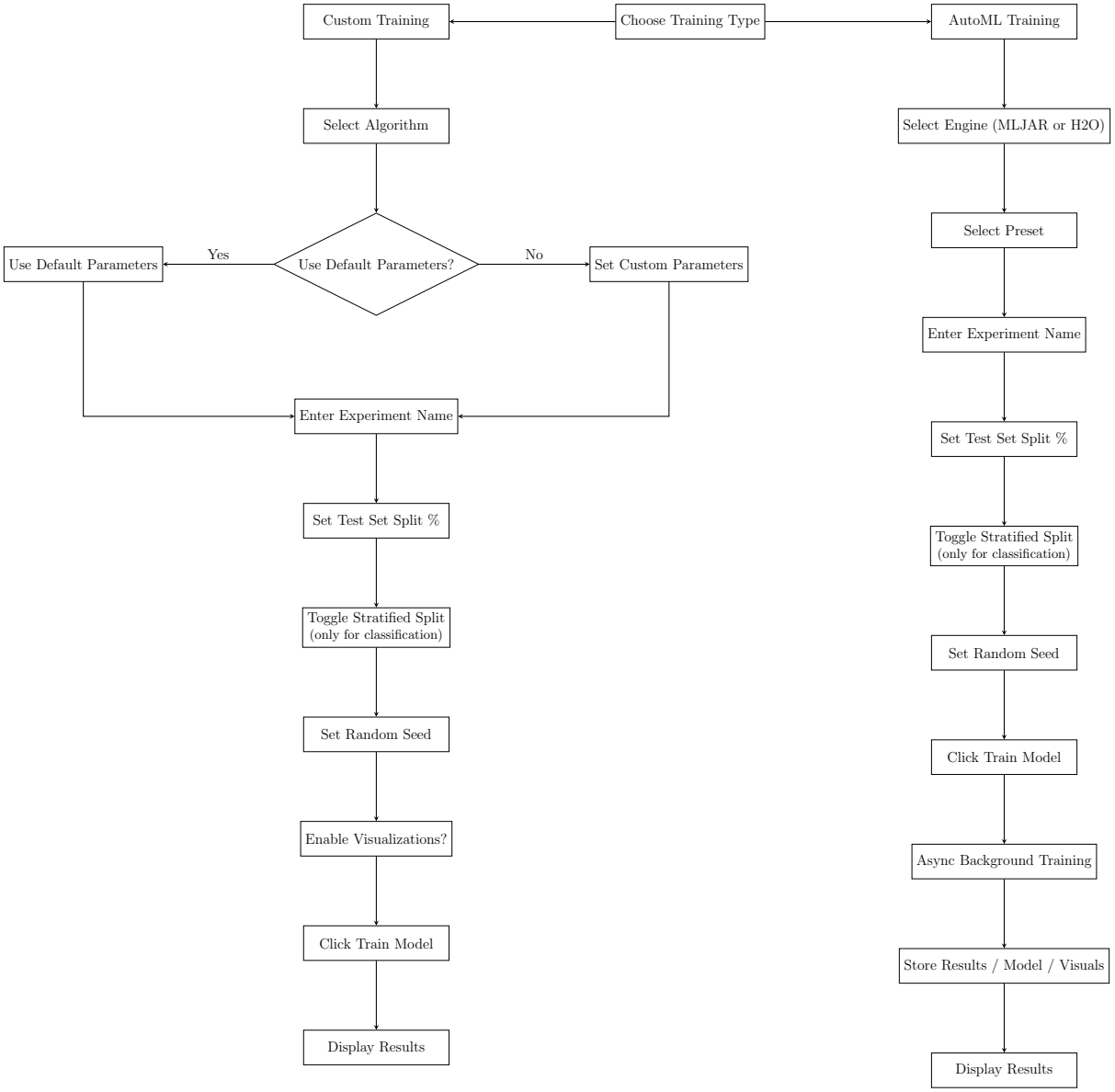


Figure 7.2: Training Workflow in `ieeAutoML`: Custom and AutoML Training Paths

Prediction Workflow – Manual and CSV-Based

The prediction workflow in Figure 7.3 supports manual entry as well as CSV-based prediction. Users enter feature values manually through the direct input system which generates both predictions and confidence scores. The system processes uploaded CSV data through parsing procedures. The system evaluates prediction accuracy from the target column if present otherwise it produces only predictions. Both paths merge their output into one unified display interface that shows results in an easy-to-read format. This step omits SHAP explanations because it helps to achieve faster operation during real-time user input scenarios and basic system operations.

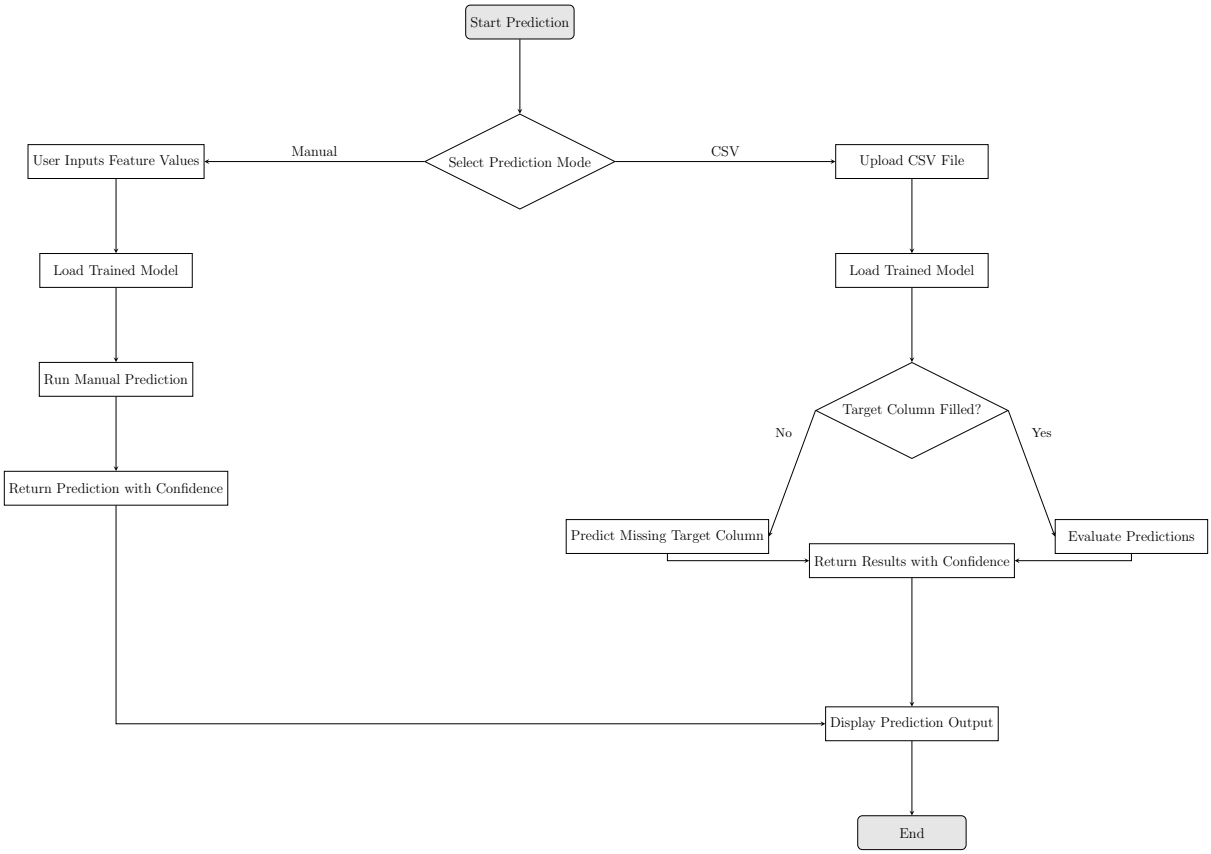


Figure 7.3: Prediction Workflow in `ieeAutoML`: Manual Entry and CSV Upload

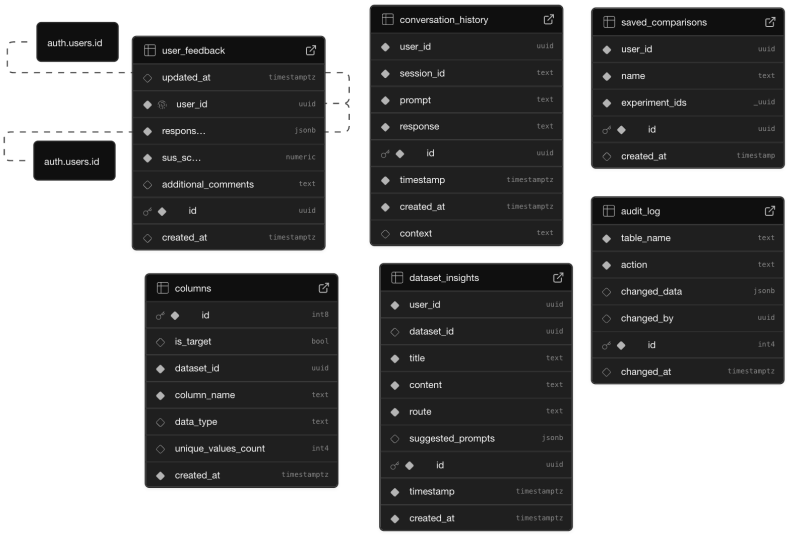
7.3 Database Design

The Supabase system uses PostgreSQL as its database implementation for schema design. The system manages user information as well as datasets and experiments and models and comparisons and visualizations through different relational database tables. Key tables include:

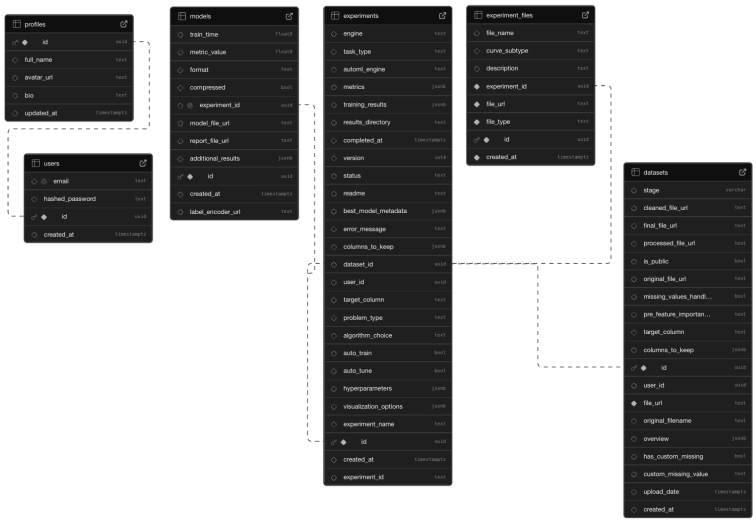
- `users` — The system stores user authentication data in the "users" table through Supabase authentication.
- `datasets` — The "datasets" table contains file path information and target column identification and version control for raw data cleaning and final processing.

- `experiments` — The "experiments" core table tracks all training runs and includes information about custom or autoML experiments as well as engine selection and algorithm choice and performance metrics and timestamps.
- `models` — The "models" table contains information about trained model metadata which includes filename, path and type while linking to experiment IDs.
- `comparisons` — The "comparisons" table contains experiment IDs that users have marked for comparison purposes.
- `visualizations` — The "visualizations" table stores URLs together with file types for different plot types such as ROC and PR and SHAP and confusion matrix.

Every experiment maintains links to datasets and optionally trained models. Foreign keys connect visualizations with their corresponding predictions and experiments in the database. The schema design allows easy expansion of new features such as version control and team collaboration and user role management.



(a) ER schema — part 1



(b) ER schema — part 2

Figure 7.4: Entity-Relationship Diagram of the *ieeAutoML* system, split into two parts for clarity.

7.4 System Architecture

The high-level system design, illustrated in Figure 7.5, consists of four core components: a The system design includes a React-based frontend and a FastAPI backend that provides all machine learning and data end-points and Supabase for authentication and file storage and PostgreSQL database management and Python-based ML engines including Scikit-learn, H2O, and MLJAR. This modular The architecture allows background training through asynchronous processing while maintaining a clear separation of logic to

ensure the system remains scalable for both team-based functionality and future public deployment.

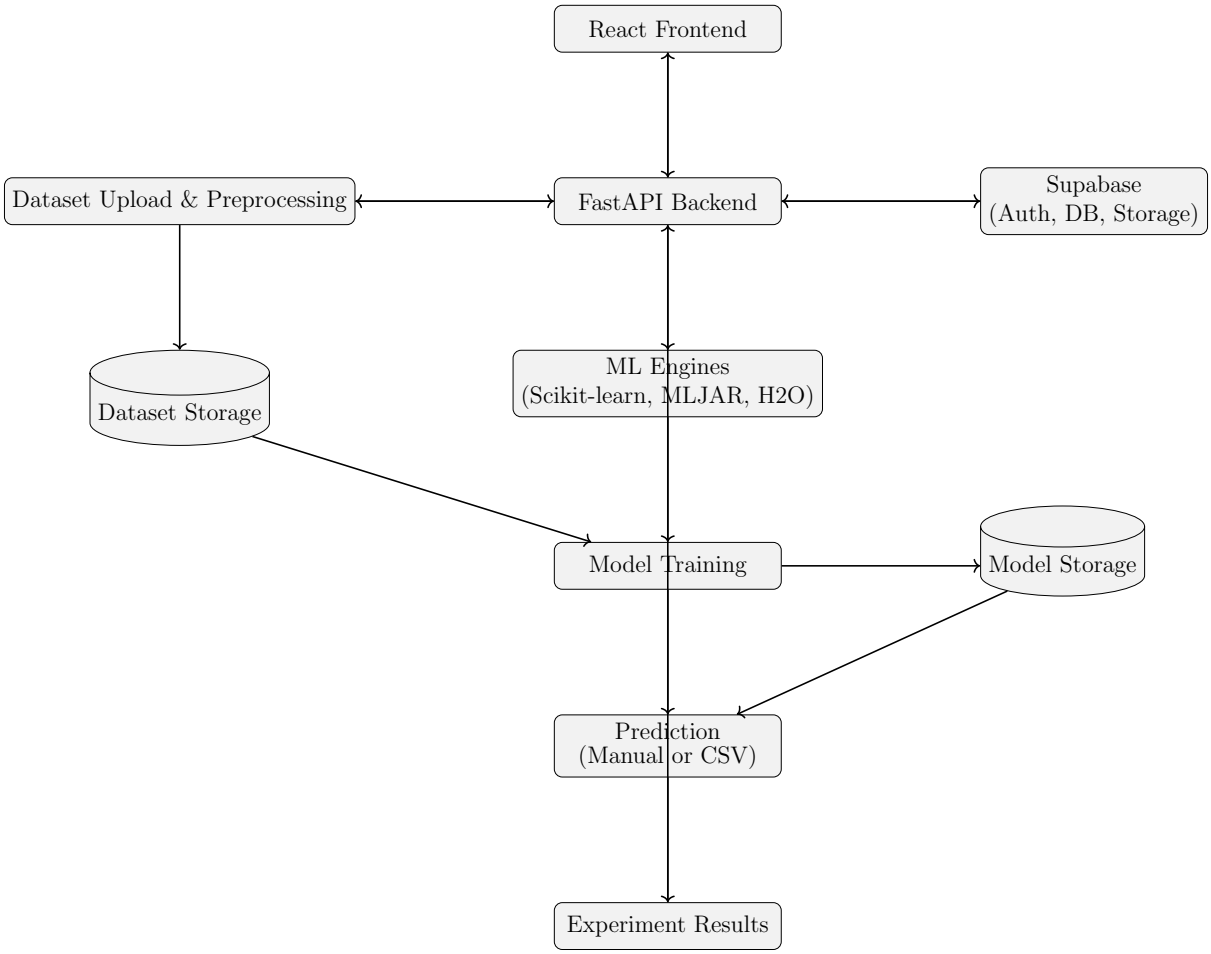


Figure 7.5: System Architecture of `ieeAutoML`: Interaction between Frontend, Backend, Supabase, Storage, and ML Engines

Implementation of ieeAutoML

The following section explains the technical aspects of **ieeAutoML** web application development through a detailed description of backend and frontend components. The system developers designed the system to operate with modularity and scalability features while remaining user-friendly for non-technical machine learning users.

8.1 Backend Implementation

The **FastAPI** framework powers the **ieeAutoML** backend to run a high-performance asynchronous Python API server. The system design uses a modular approach which divides responsibilities between separate folders.

- `api/`: The api/ folder contains route handlers which function as separate modules for dataset management and training and prediction and results processing.
- `core/`: The core/ folder controls configuration management as well as authentication and security utilities.
- `db/`: The db/ folder contains database models and Supabase integration definitions.
- `services/`: The services/ folder contains the core ML logic which includes preprocessing and training (AutoML and custom) and visualization.
- `utils/`: The utils/ folder contains shared helper functions together with configuration constants.

8.1.1 Machine Learning Pipeline

The backend system operates through two training modes.

1. **Custom Training**: Users can select algorithms such as Decision Trees, Random Forests, Gradient Boosting (XGBoost, LightGBM, CatBoost), Logistic Regression, and Neural Networks through the Custom Training option. Users have the option to select their own hyperparameters or use the default values. The training process relies on `scikit-learn`.
2. **AutoML Training**: The system supports two AutoML engines which are `MLJAR` and `H2O AutoML` AutoML for automatic model training and selection of the best model based on performance metrics.

The system stores trained models within Supabase Storage for retrieval purposes. The system uses compression methods and serialization techniques to handle big model files such as H2O.zip files. The system creates visualizations including confusion matrix and SHAP and ROC curves which get uploaded during or after the training process.

8.1.2 Key Backend Endpoints

Table 8.1 summarizes the most important backend endpoints that form the machine learning pipeline in **ieeAutoML**. These endpoints cover dataset management, preprocessing, training, and prediction.

Table 8.1: Key backend endpoints of **ieeAutoML**

Endpoint	**Purpose**
POST `/dataset-overview/`	Upload a dataset (CSV), store it as `raw.csv`, and return dataset overview (columns, missing values, statistics).
POST `/handle-dataset/`	Apply missing value handling strategies (mean, median, mode, drop, hot-deck) and save `cleaned.csv`.
POST `/feature-importance-preview/`	Analyze feature importance and detect task type (binary classification, multiclass, regression).
POST `/save-dataset/`	Save selected target and features to `final.csv`.
POST `/data-preprocess/`	Apply normalization and class balancing, then save `processed.csv`.
POST `/automl/`	Train a model using AutoML engines (MLJAR or H2O), automatically selecting the best algorithm.
POST `/custom-train/`	Train a custom model with user-selected algorithm (e.g., Logistic Regression, Random Forest, XGBoost) and hyperparameters.
GET `/check-status/{experiment_id}`	Check experiment training status (running, completed, failed).
POST `/predict-manual/`	Make a prediction using a trained model with manual input values.
POST `/predict-csv/`	Make predictions or evaluate a model on a batch dataset uploaded as CSV.

8.1.3 Dataset Handling and Preprocessing

The uploaded CSV files undergo secure parsing and storage operations within Supabase Storage. The back-end system enables extensive preprocessing operations through statistical methods (mean, median, mode) and row removal and advanced hot-deck imputation. The system enables feature selection through mutual information and correlation analysis while providing normalization techniques including min-max scaling and 0–100 scaling

and class balancing methods such as SMOTE. The system implements automatic task type detection which identifies binary classification and multiclass classification and regression tasks. The system generates new dataset versions at each preprocessing stage (`raw.csv`, `cleaned.csv`, `final.csv`, `processed.csv`) to maintain complete transparency and traceability throughout the workflow.

8.1.4 Prediction Module

The backend system provides two prediction endpoints which allow users to make predictions through manual input or batch predictions from uploaded CSV files. The system loads the trained model dynamically when it receives a prediction request before applying the same preprocessing pipeline that was used during training to process the input data. The response includes both predicted value and confidence score information but CSV-based prediction mode adds extra evaluation metrics to assess model performance.

8.2 Frontend Implementation

The **ieeAutoML** frontend operates with React for its implementation while it follows a design that emphasizes simplicity alongside clarity and accessibility. The system provides full machine learning capabilities from data import to prediction delivery to users who lack programming knowledge. The user interface follows a structured organization with five main tabs which correspond to each phase of the ML pipeline.

The user interface follows a structured organization with five main tabs which correspond to each phase of the ML pipeline:

- **Dashboard Tab**: Users can access this central dashboard to handle all datasets alongside experiments and model comparison features. It includes:
  - The Datasets panel where users can view all uploaded files through previews or downloads for each processing stage starting from raw to processed data.
  - The Experiments panel where users can manage previous experiments by viewing metrics and charts or by deleting old results.
  - The Comparisons panel where users can choose two or more experiments to view their performance side by side.
- **Dataset Tab**: Users can find a structured data preparation process through four sub-stages within this tab.
  - *Upload*: Accepts CSV datasets and detects missing value symbols.
  - *Explore*: The system provides users with an overview display during Explore phase before they apply missing value imputation using mean, median, mode, drop, or Hot Deck via KNN methods.
  - *Features*: Users can examine feature importance while setting their target column and input features through this interface. The system automatically identifies whether the task should be classification or regression.

  - *Preprocess*: The system enables users to perform scaling and class balancing through its Preprocess function. The system provides context-dependent options which show SMOTE only during classification tasks involving numerical data.

- **Training Tab**: Users can configure models and track results while training their models through the Training Tab. It con- contains:
  - *AutoML*: The users can start automated training processes through the AutoML feature which uses either MLJAR or H2O engines. Users can select their desired training approach through a preset selection menu (e.g., `fast`, `balanced`, `accurate`).
  - *Custom*: Through the Custom option users can pick between Decision Tree and XGBoost and Neural Network algorithms before they can adjust hyperparameters and define their train/test split and stratification preferences.

- **Results Tab**: The Results Tab delivers visualized performance results alongside analytical insights. The following information is presented:
  - Users can find important metrics which include accuracy, precision, recall, F1 score, AUC, RMSE and more.
  - The system displays confusion matrices together with classification reports.
  - ROC and PR curves
  - SHAP explanations and feature importances are displayed through the system.
  - The AutoML leaderboards function is available for both MLJAR and H2O platforms.

- **Prediction Tab**: Users can perform real-time model predictions through the Prediction Tab.
  - *Manual Prediction*: Users who input feature values receive predictions together with confidence scores or probability values through the Manual Prediction tool.
  - *CSV Prediction*: Users can upload test CSV files to obtain downloadable prediction results.

The frontend implements TanStack Query for server-state caching along with React Context for internal state management (auth, training, dataset selection) and native `fetch()` for API communication. The designed architecture provides both speed and modularity as well as simple operation.

8.2.1 State Management

The application uses TanStack Query together with React Context to achieve efficient state management. The application depends on TanStack Query to manage server state and cache API responses and perform background updates which makes it essential for data-fetching features. The application uses React Context to handle client-side UI state management for authentication and training session tracking and dataset metadata. The

architecture benefits from not using Redux because it stays lightweight and performs well. The application uses three context providers named `AuthContext`, `TrainingContext` and `DatasetContext` to handle their individual responsibilities while keeping separate concerns.

8.2.2 API Communication

The application implements `fetch()` API through utility functions for backend endpoint interactions. These include:

- `getAuthHeaders()`: The application applies Supabase JWT tokens to requests through getAuthHeaders() function
- `handleApiResponse()`: The function handleApiResponse() functions as a universal approach to process both successful and unsuccessful responses.

8.2.3 User Experience

The application leads users through dataset selection and preprocessing followed by model training and result analysis and prediction steps. The application uses tooltips along with previews and context-dependent options to provide usability for users who are not experts.

8.3 Conclusion

The **ieeAutoML** implementation makes use of advanced technologies to create an effective and easy-to-use system for supervised learning operations. The system combines a FastAPI-based modular backend with a React-based frontend using clean state management and full Supabase integration to provide technical abstraction while maintaining flexible and transparent machine learning workflows.

Presentation of ieeAutoML

This chapter presents the user-facing functionality of the ieeAutoML web application through its main workflow tabs. Each subsection corresponds to a distinct stage in the machine learning pipeline, as realized in the frontend.

The application is designed with clarity and accessibility in mind, guiding the user through dataset upload, feature selection, preprocessing, training, evaluation, prediction, experiment management, and AI-based assistance.

9.1 Data Handling

The first step in the ieeAutoML pipeline involves uploading and exploring the dataset. The interface is designed to automatically analyze the dataset structure and identify missing values, while also allowing for user-defined configurations.

9.1.1 Dataset Upload and Missing Value Symbol

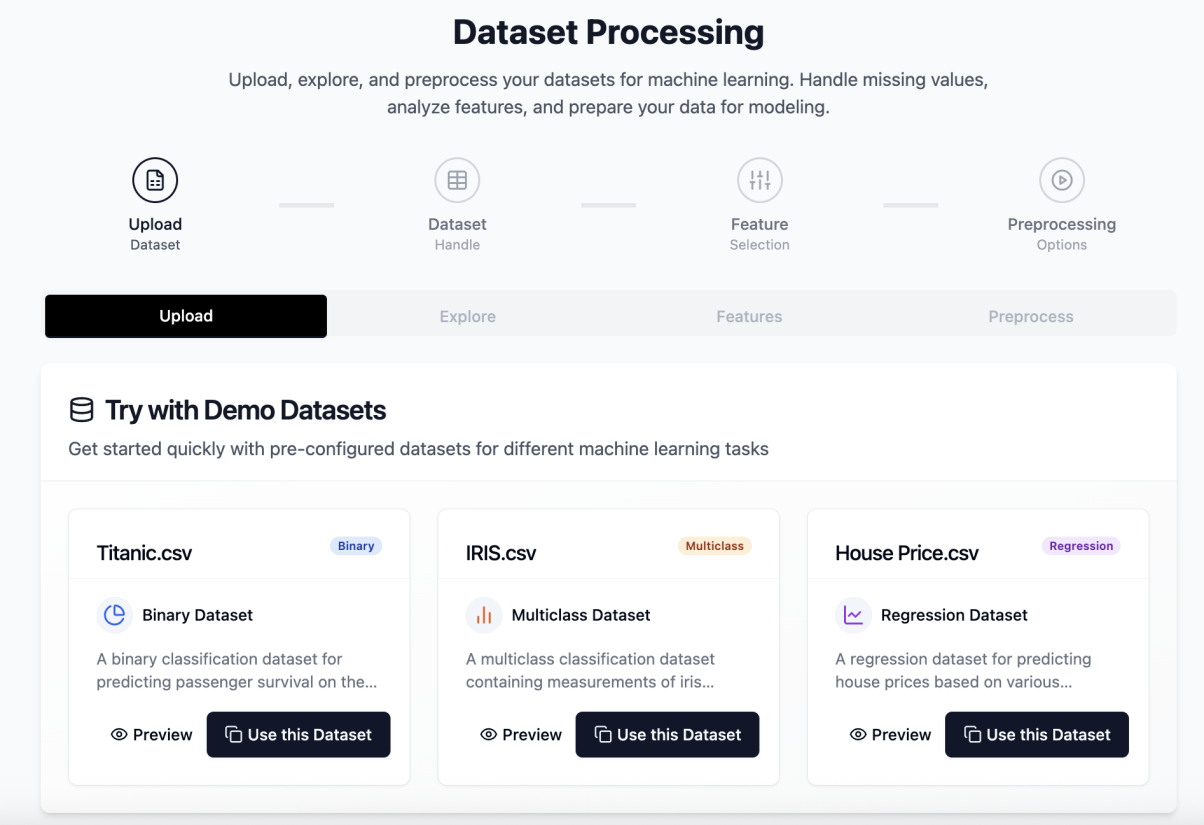
The first step in the ieeAutoML workflow is uploading a dataset. Users are prompted to upload a CSV file, which will serve as the foundation for the rest of the pipeline. In addition to uploading their own data, users can also choose from one of three preloaded demo datasets, each corresponding to a distinct supervised learning problem type:

- **Binary Classification**
- **Multiclass Classification**
- **Regression**

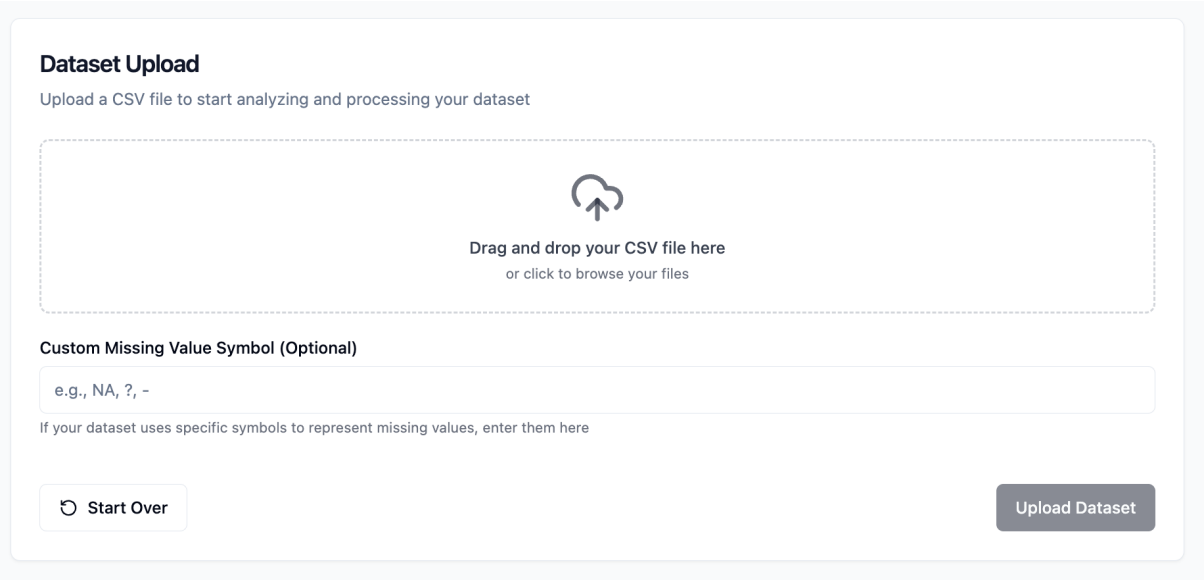
This option provides a fast way to test and explore the functionality of ieeAutoML without needing to prepare data externally.

Upon file upload or demo selection, the system automatically displays a brief preview of the dataset, including the first few rows and column headers. This allows the user to quickly verify the structure and content of the file before proceeding.

Additionally, users have the option to specify a custom missing value symbol (e.g., `9999`, `?`, or `NULL`) if their dataset uses non-standard indicators for missing entries. If no custom symbol is provided, ieeAutoML scans for common missing value indicators such as empty cells, `NaN`, and `None`.



(a) First part of the upload screen



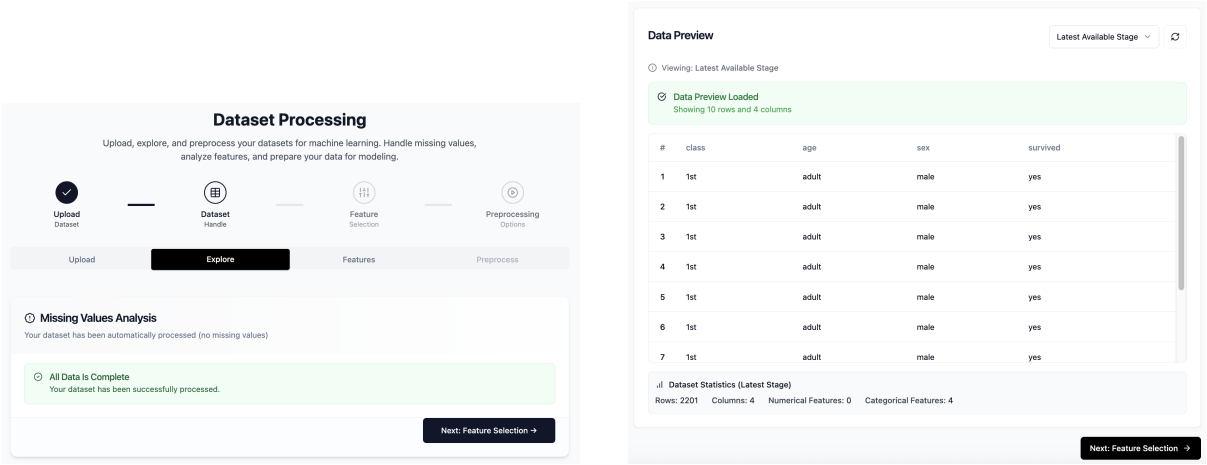
(b) Second part of the upload screen

Figure 9.1: Upload screen with optional custom missing value symbol input

9.1.2 Missing Values Analysis

After uploading, the app transitions to the "Explore" step. It automatically checks for missing values and displays a summary to the user. If no missing values are found, the dataset is marked as complete and the user is allowed to proceed directly to the feature

selection step.



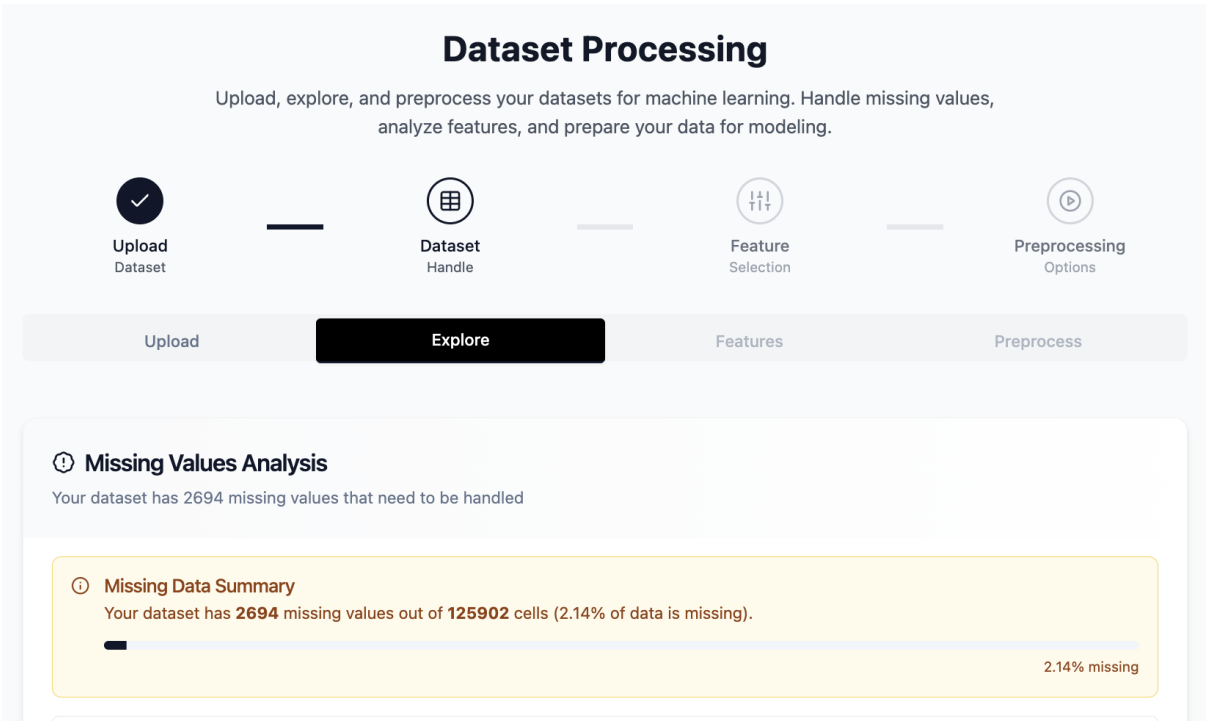
(a) First part of the explore view

(b) Second part of the explore view

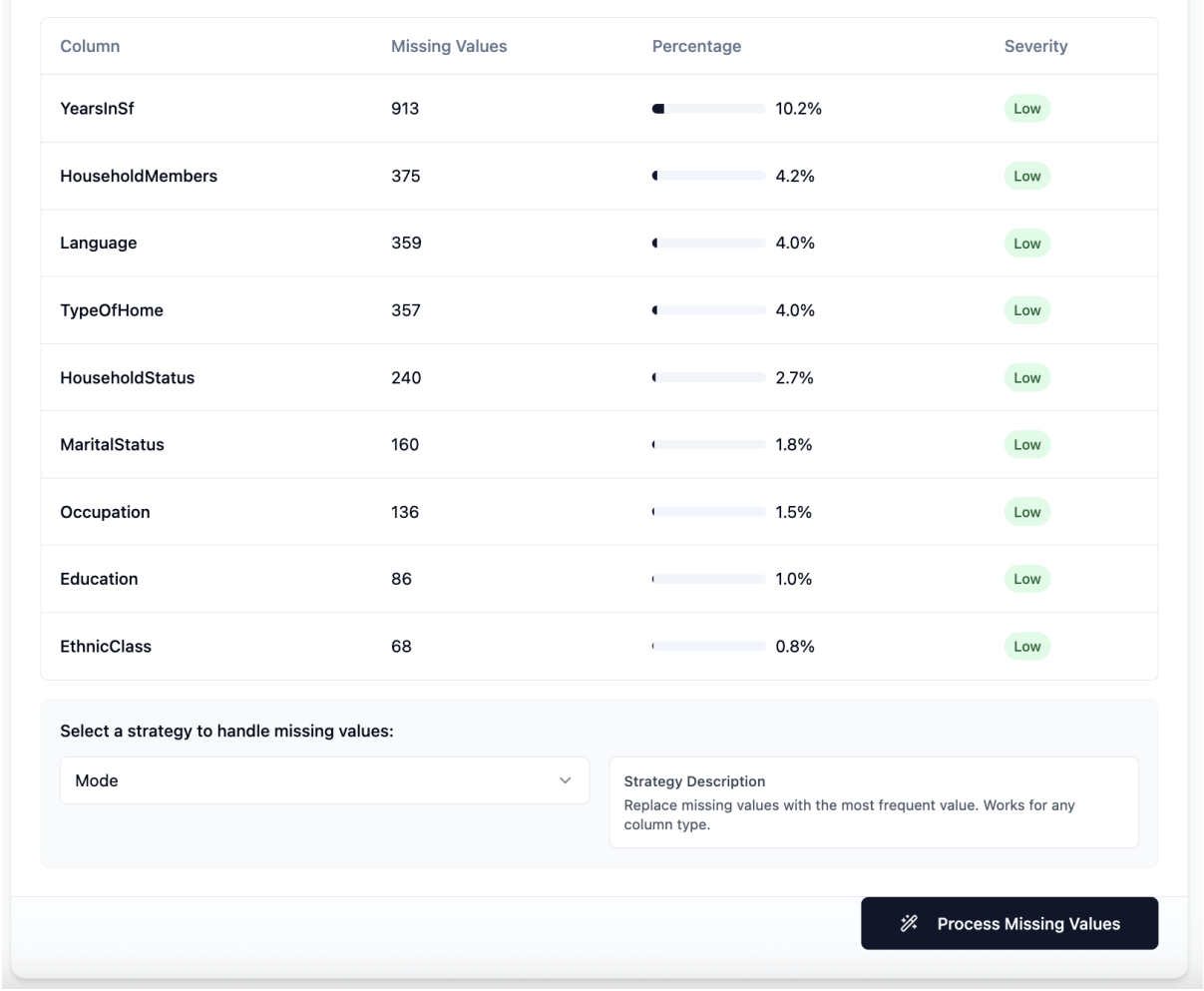
Figure 9.2: Explore view for datasets with no missing values

In contrast, if missing values are detected, a detailed summary is shown, listing the affected columns, count and percentage of missing data, and severity level.

The user must then select an imputation strategy from a dropdown menu, with available options such as *Mode*, *Mean*, *Median*, *Drop Rows*, or *Hot Deck* (KNN-based imputation). The "Process Missing Values" button becomes active only after a strategy is selected.



(a) Summary of missing values



(b) Imputation options view

Figure 9.3: Explore view for datasets with missing values — summary and imputation options

9.1.3 Data Preview

For every stage of the pipeline, ieeAutoML allows users to preview the dataset. In the Explore step, the preview corresponds to the original raw data. It includes a sample of 10 rows, column names, and a summary of dataset statistics such as the number of rows, total columns, and feature types (numerical or categorical).

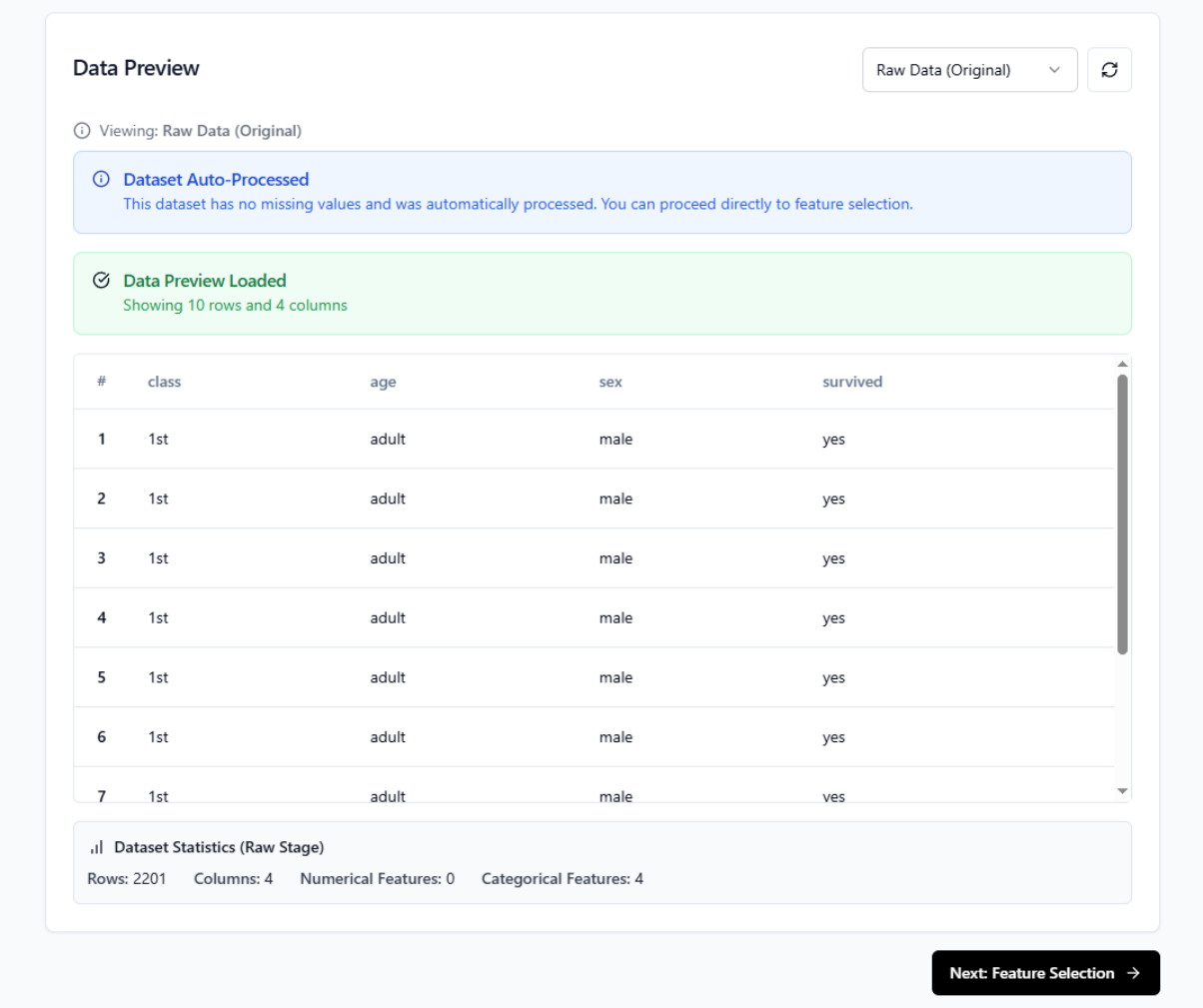


Figure 9.4: Data preview for a dataset with no missing values (4 categorical features)

If the dataset contains many categorical columns (e.g., after encoding), the preview adapts accordingly and displays feature type distribution.

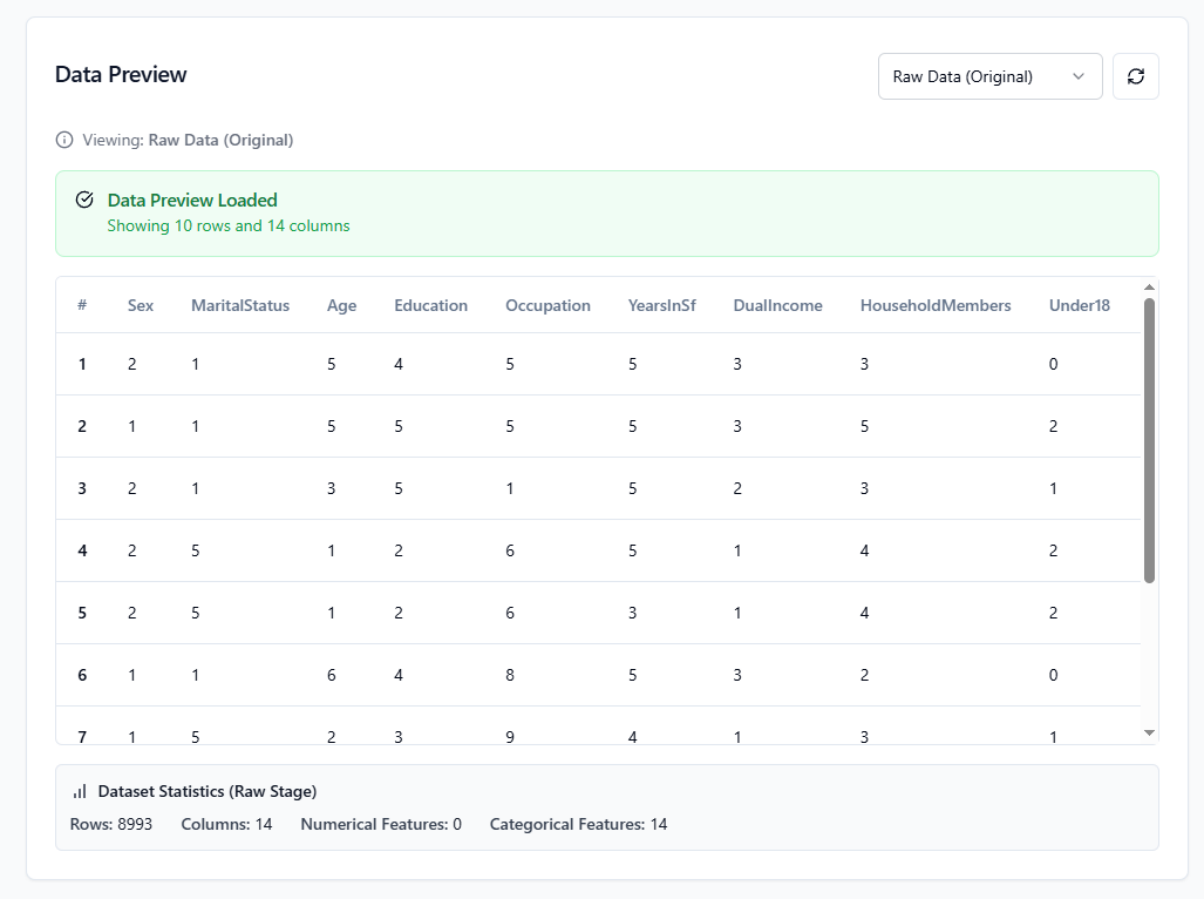


Figure 9.5: Data preview for a dataset with 14 categorical features

9.1.4 Conditional Navigation

Only after successfully handling missing values (or confirming none exist), the "Next: Feature Selection" button becomes available. This ensures that all datasets entering the feature selection step are clean and ready for analysis.

"Explore" thus functions as a data quality gate, enabling robust downstream machine learning operations.

9.2 Feature Selection

9.2.1 Target Selection and Feature Picker

After handling missing values, the user proceeds to select the target column — the variable they wish to predict. This is done through a dedicated dropdown interface. Once the target is selected, the system dynamically displays the remaining columns available for use as features.

The user is then able to manually choose which of these features to include in the feature importance analysis, or can use the "Select All" button to include all.

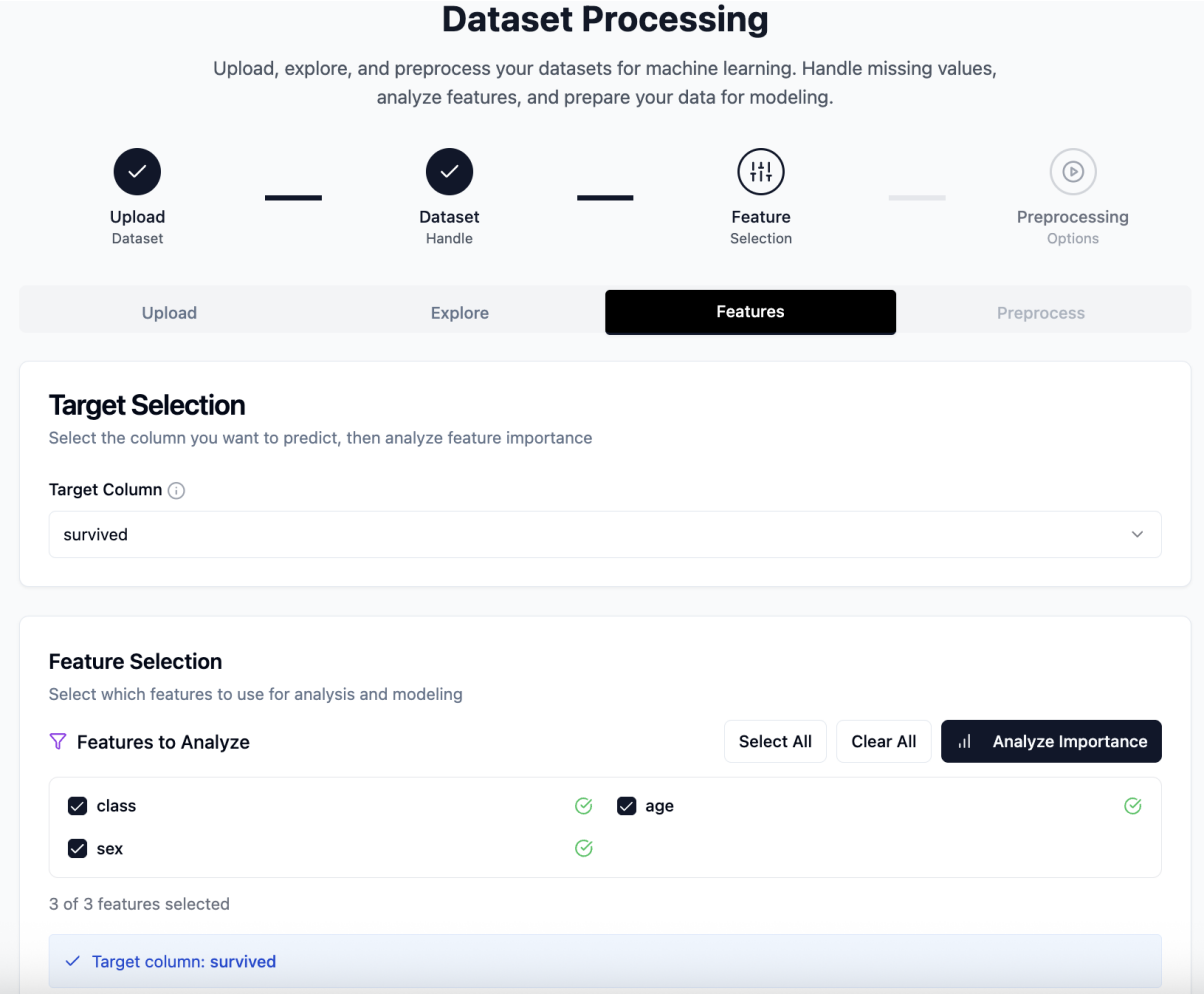


Figure 9.6: Target column selection interface and feature list.

9.2.2 Feature Importance Analysis

Upon selecting the desired features, the user can initiate a feature importance analysis by clicking the *"Analyze Importance"* button. The backend computes the relative importance scores of each selected feature using model-based techniques (e.g., tree-based feature importance).

The results are displayed as a horizontal bar chart, allowing the user to visually inspect which features are most predictive. This helps inform which features to retain before proceeding.

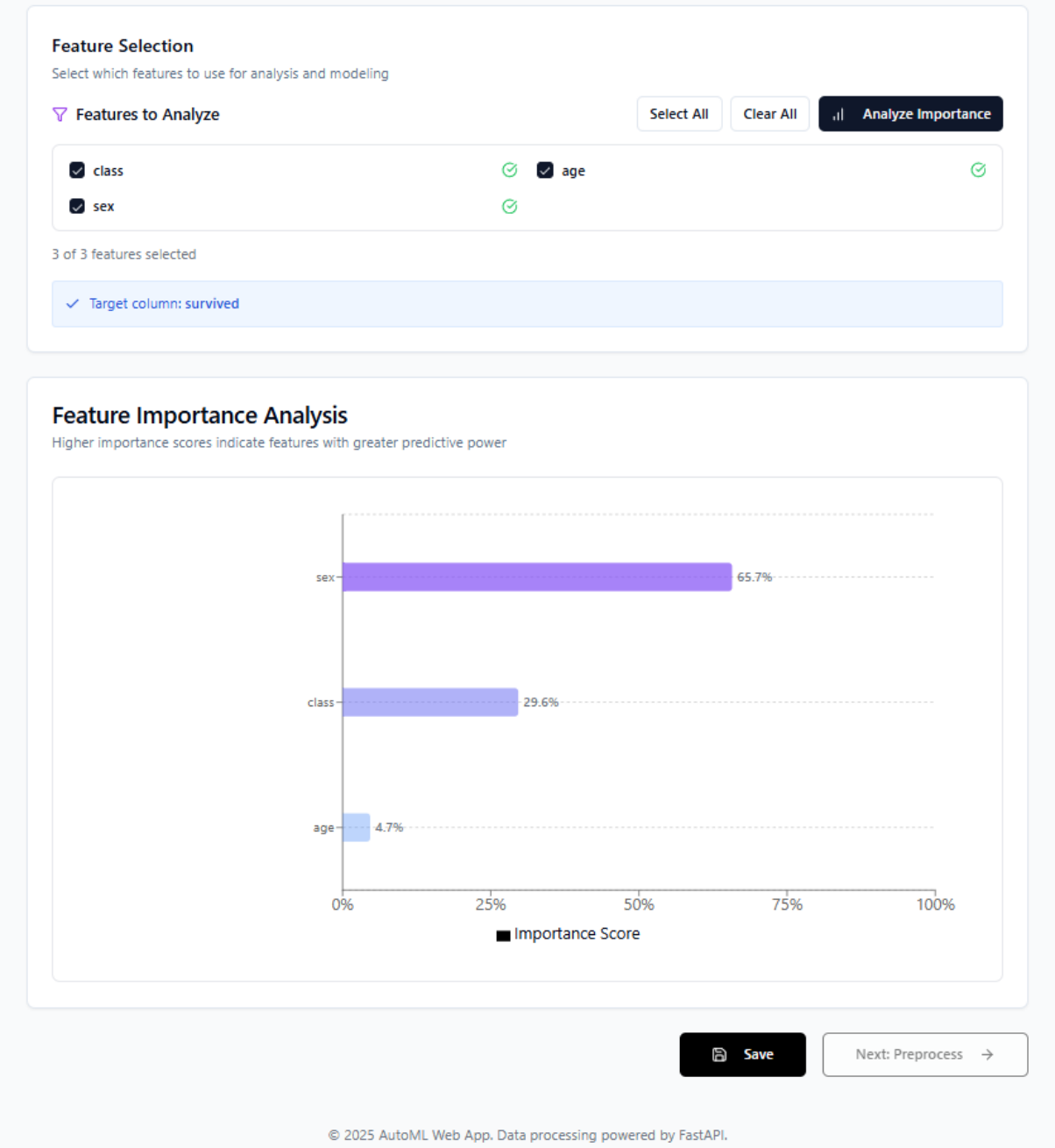


Figure 9.7: Bar chart showing feature importance scores for selected features.

9.2.3 Saving and Proceeding

Only after the user clicks the **"Save"** button for the selected features can they proceed to the preprocessing stage. This ensures the system is trained only on intentional and user-approved feature subsets.

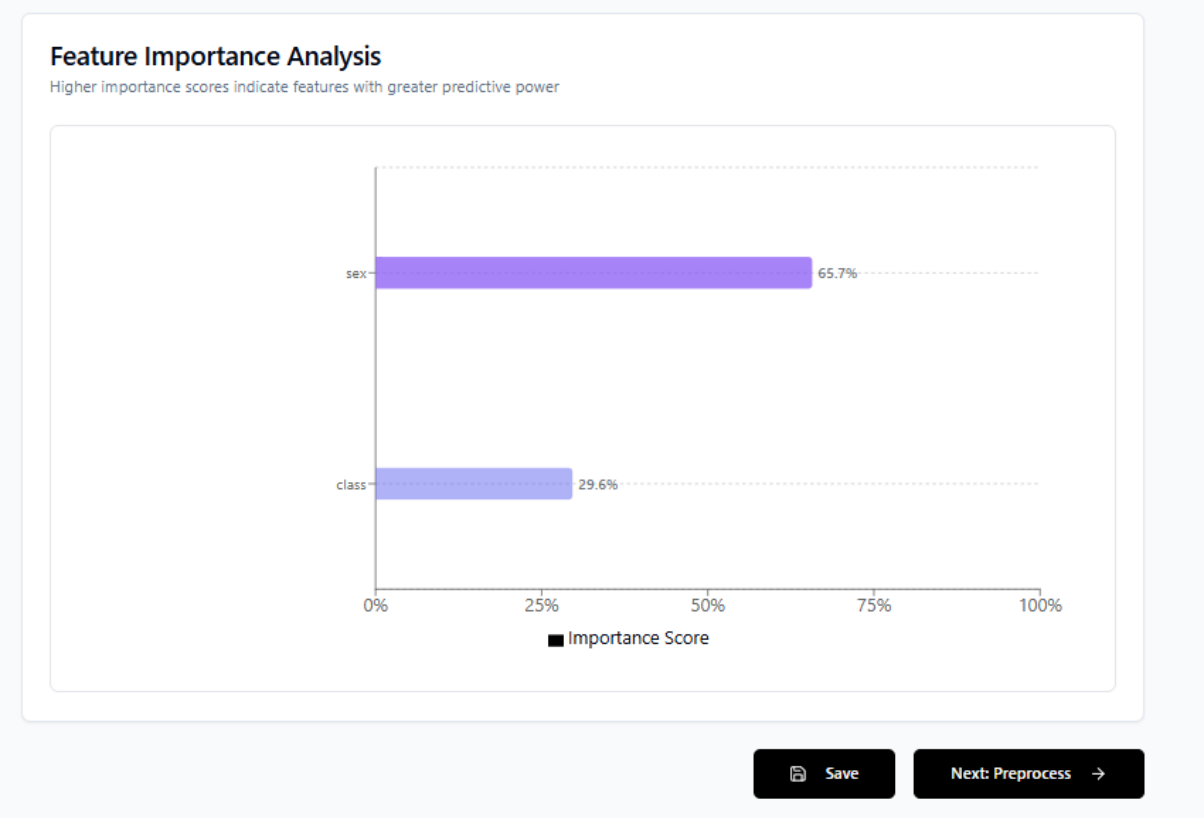


Figure 9.8: User must save selected features to continue.

9.3 Preprocessing

After the dataset is uploaded, missing values handled, and features selected, the user proceeds to the preprocessing step. This stage applies key transformations such as feature scaling and class balancing to optimize model performance. The interface is designed to summarize the dataset status and offer contextual help.

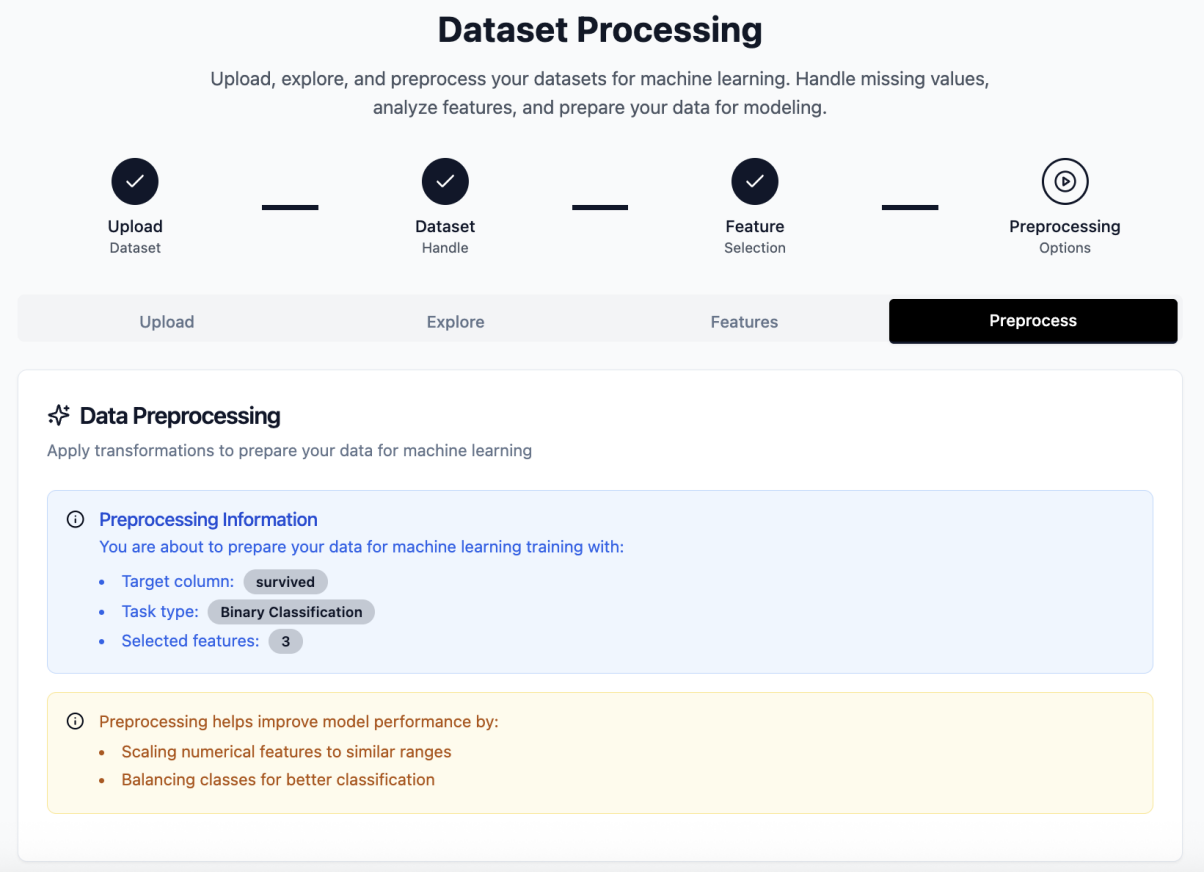


Figure 9.9: Preprocessing step overview with target, task type, and selected features

9.3.1 Preprocessing Options

The application dynamically analyzes the dataset to determine the availability of numerical features and the balance of the target variable. A class distribution pie chart is presented to highlight class imbalance, guiding the user toward appropriate class balancing techniques.

Feature scaling can be applied using various methods (Min-Max, Standard, Robust, Log), but is skipped automatically if no numerical features are present in the selected columns.

If the task type is classification, users can enable class balancing with options such as:

- **Undersampling:** Reduces the number of samples from majority classes
- **Oversampling:** Duplicates or synthetically generates samples for minority classes (e.g., SMOTE, ADASYN)

Balancing methods appear dynamically based on the selected balancing strategy.

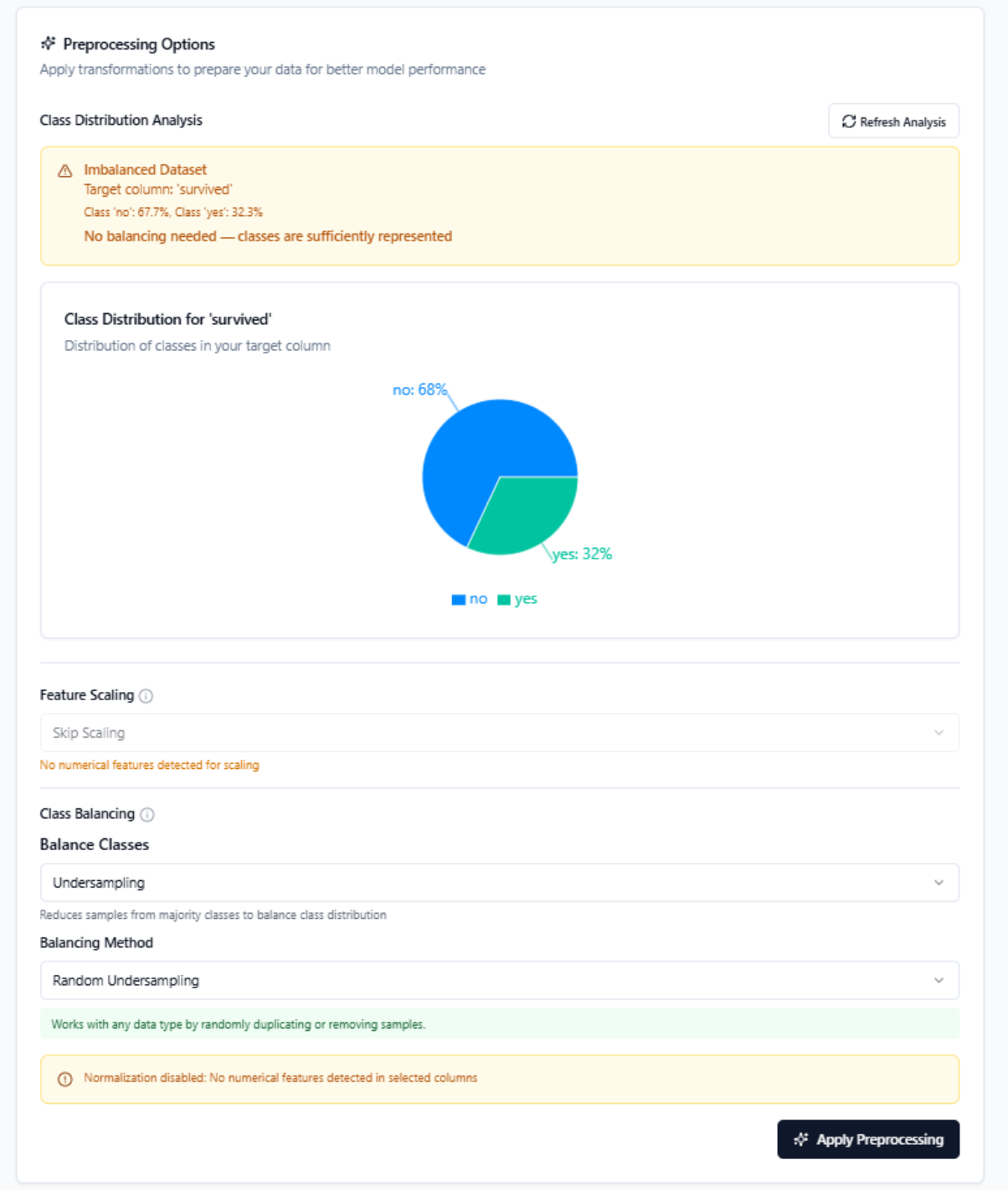


Figure 9.10: Preprocessing options for class distribution, scaling, and balancing

9.3.2 Preview and Finalization

Users can preview the dataset at different stages — raw, cleaned, or after feature selection — before applying the final preprocessing transformations. This ensures full transparency and control over what is being sent into the model training pipeline.

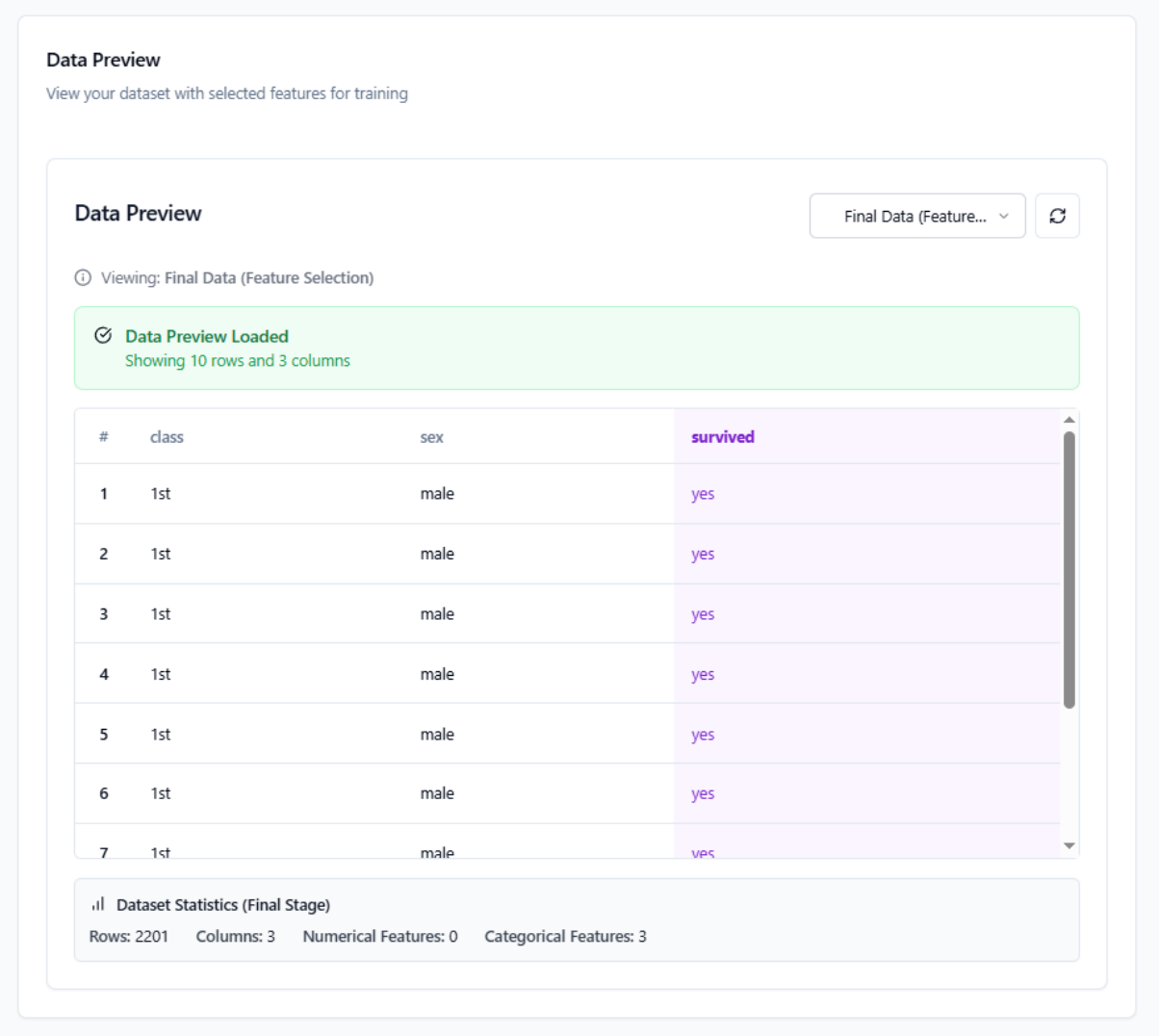


Figure 9.11: Live preview of the dataset at different preprocessing stages

Once the transformations are reviewed and selected, the user clicks on *Apply Preprocessing* to finalize the processed dataset and proceed to training.

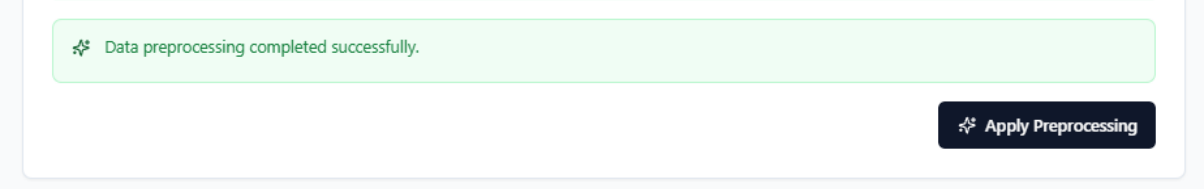


Figure 9.12: Applying selected preprocessing steps including optional class balancing

9.4 Model Training

Once the dataset has been fully prepared, users proceed to the training phase via the "Training" tab. This section initiates with a compact summary panel showcasing the final dataset's metadata — including the number of rows, selected features, target column, and task type (binary classification, multiclass classification, or regression). Additionally,

users are informed about the number of numerical and categorical features detected, giving context to the training phase.

Below this overview, a four-tab interface is presented: **AutoML**, **Custom**, **Results**, and **Predict**. Initially, only the AutoML and Custom tabs are active, while the Results and Predict tabs remain disabled until a model is successfully trained.

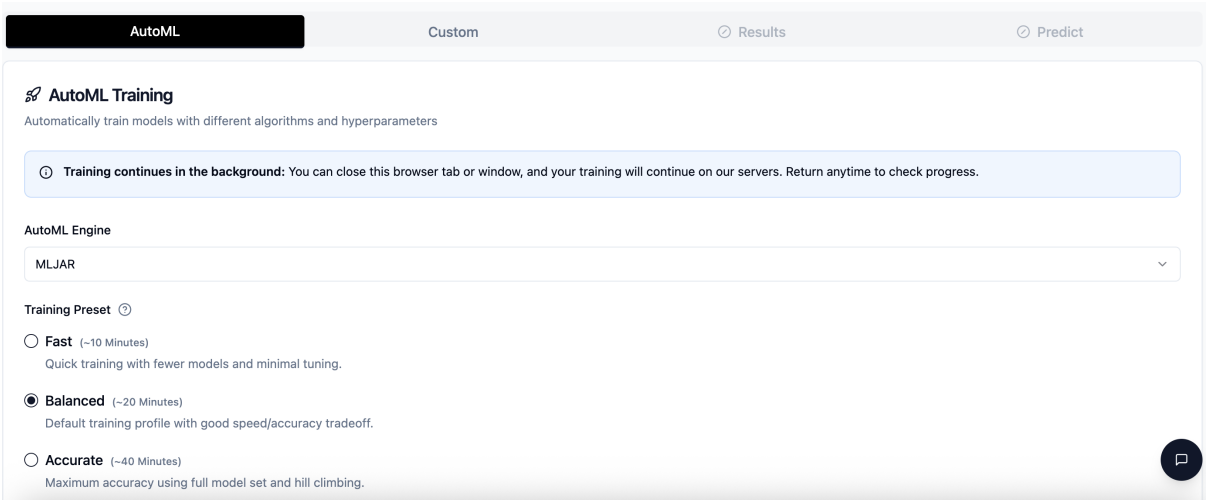
9.4.1 AutoML Training

In the **AutoML** tab (Figure 9.13), users can quickly train a model using one of two engines: **MLJAR** or **H2O AutoML**. The user selects an engine and then chooses among three training presets:

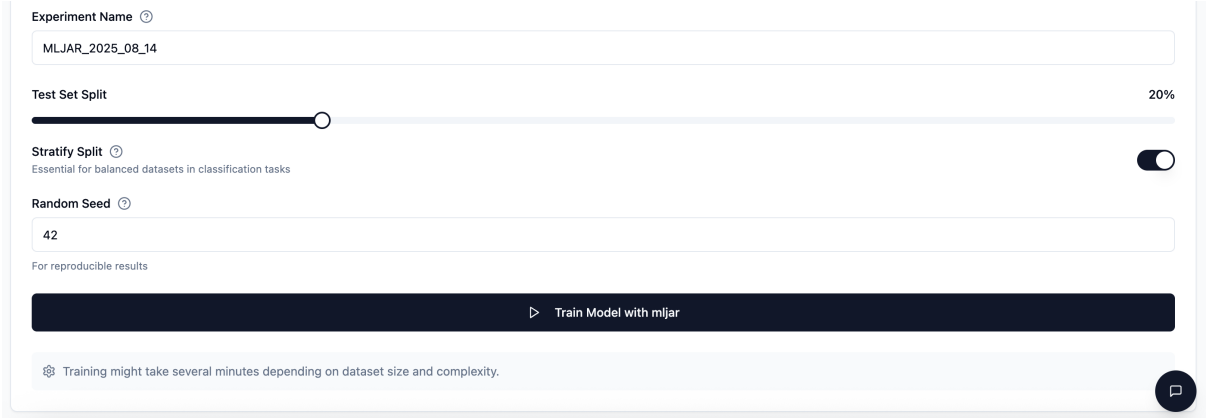
- **Fast**: prioritizes training speed with fewer models and limited tuning (estimated time < 10 minutes).
- **Balanced**: provides a tradeoff between speed and accuracy (estimated time ~ 20 minutes).
- **Accurate**: performs an exhaustive search using full models and advanced tuning (estimated time > 40 minutes).

Next, users provide an experiment name and set the test set split ratio (e.g., 20%). For classification tasks, a *Stratify Split* toggle is available to ensure balanced label distribution between training and test sets. A random seed (default: 42) is also configurable for reproducibility.

Once the configuration is complete, clicking the "Train Model" button initiates the background training process. Users are informed that they can safely close the browser or tab, as the training continues server-side.



(a) Top section of AutoML training configuration



(b) Bottom section of AutoML training configuration

Figure 9.13: AutoML training configuration interface in `ieeAutoML`

9.4.2 Custom Training

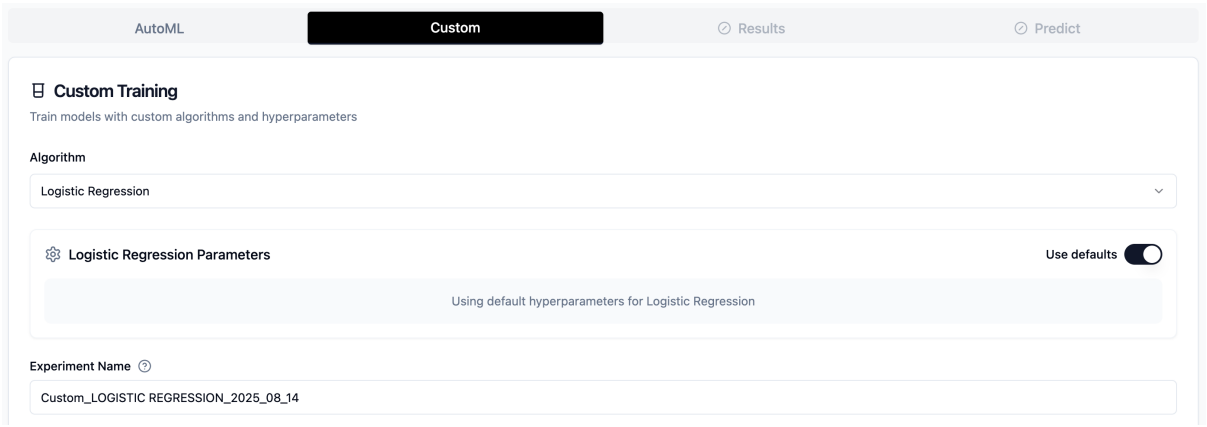
The **Custom** tab (Figure 9.14) enables users to train models using specific algorithms and user-defined hyperparameters. The available algorithm list includes:

- Logistic Regression
- Decision Tree
- Random Forest
- XGBoost
- LightGBM
- CatBoost
- Neural Network
- Extra Trees

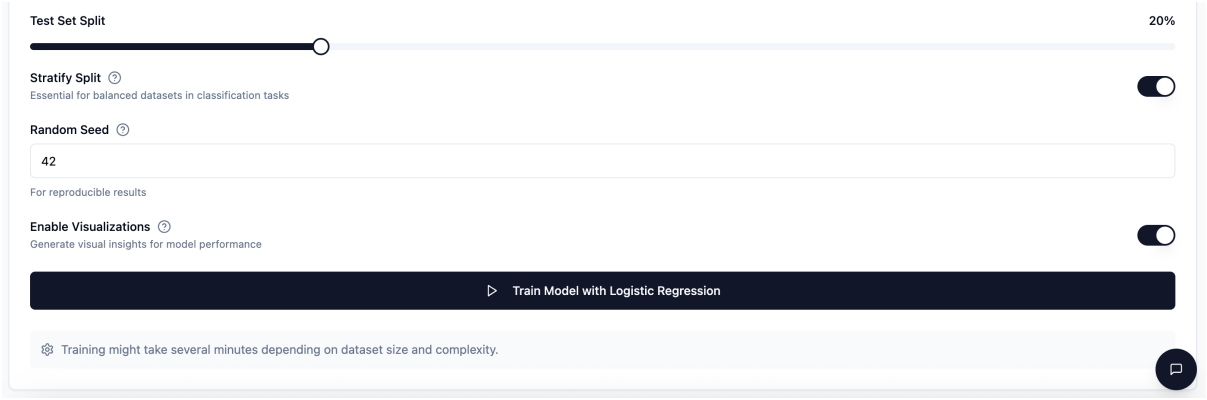
- Nearest Neighbors

Once an algorithm is selected, the corresponding hyperparameter input fields are dynamically displayed (Figure 9.15). Users can either manually configure the hyperparameters or toggle the "Use Defaults" option for automatic values.

In addition to experiment name, test split ratio, stratification, and random seed, users can also toggle "Enable Visualizations" to generate extended model performance visualizations after training.



(a) First part of the custom training algorithm selection interface



(b) Second part of the custom training algorithm selection interface

Figure 9.14: Custom training algorithm selection in ieeAutoML

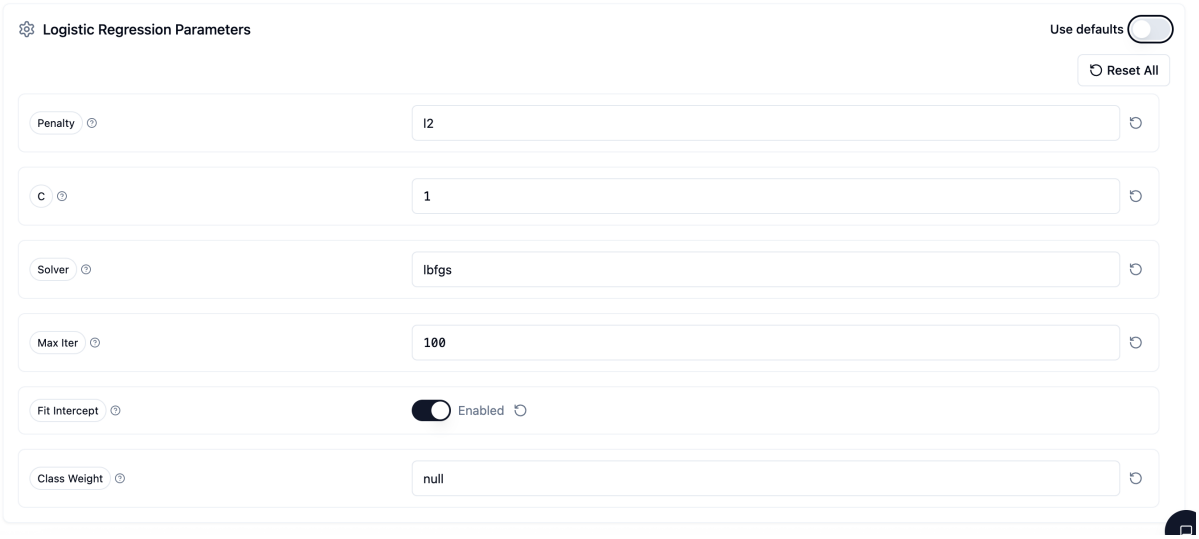


Figure 9.15: Custom hyperparameter configuration for Logistic Regression

9.5 Model Evaluation

Once training is complete—whether through AutoML or Custom configuration—the application automatically redirects the user to the **Results** tab. This section provides a comprehensive evaluation of the trained model's performance. Notably, training runs asynchronously in the background, allowing the user to safely close the app or navigate away while the experiment continues to run.

The Results tab is divided into four subtabs:

- **Summary**
- **Charts**
- **Predictions**
- **Model Details**

9.5.1 Summary

The **Summary** tab provides a high-level overview of the experiment. This includes key experiment metadata such as:

- Task type (e.g., Binary Classification)
- Target column
- Engine used (e.g., MLJAR)
- Experiment timestamps
- Best model identified during the experiment

In addition to metadata, a set of performance metrics is displayed. These vary based on the task type:

- For **Binary Classification**: Accuracy, Precision, Recall, F1 Score, AUC, Logloss, MCC
- For **Multiclass Classification**: Accuracy, Macro F1 Score, Logloss, Confusion Matrix
- For **Regression**: MAE, MSE, RMSE, R^2

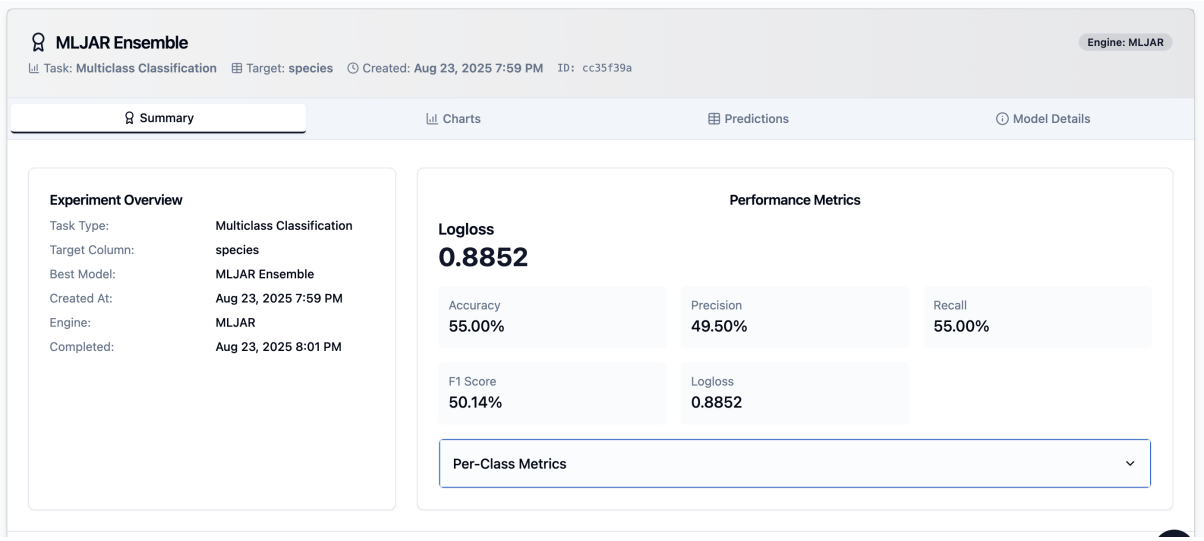


Figure 9.16: Summary tab displaying performance metrics and experiment metadata

9.5.2 Charts

The **Charts** tab presents a collection of visualizations generated during training to help users interpret model behavior and performance. These visualizations vary depending on the task type (classification or regression), the selected training engine (e.g., MLJAR, H2O), and whether the user opted to enable advanced analytics during training.

For **binary and multiclass classification**, the charts may include:

- **Confusion Matrix**: Displays true vs. predicted labels in a grid format to highlight misclassifications.
- **Normalized Confusion Matrix**: Shows the proportion of predictions per class for better class-wise comparison.
- **ROC Curve (Receiver Operating Characteristic)**: Illustrates the trade-off between true positive rate and false positive rate.
- **Precision-Recall Curve**: Particularly useful for imbalanced datasets, showing the balance between precision and recall.
- **SHAP Feature Importance**: Model-agnostic interpretability chart showing which features influence predictions the most.
- **Prediction Distribution**: Histogram of predicted probabilities for each class.

For **regression tasks**, the following charts may be shown:

- **Prediction vs Actual Scatter Plot**: Visualizes how close the predictions are to the true target values.
- **Residual Plot**: Helps evaluate if prediction errors are randomly distributed (ideal case).
- **SHAP Summary Plot**: Displays how each feature contributes to prediction variance.
- **Feature Distribution Plots**: Useful for checking normality or skewness of numerical features used in training.

These visualizations help users gain insights into model confidence, misclassification patterns, feature impact, and generalization.

Figure 9.17: Confusion matrix visualizations within the Charts tab

Depending on the engine used (e.g., MLJAR or H2O), some visualizations are automatically generated, while others may appear only if the user enabled the "Visualizations" toggle during training setup.

9.5.3 Predictions

The **Predictions** tab provides a live preview of the model's output. This includes:

- The predicted probabilities or values, depending on task type
- The actual target values for comparison

For binary classification tasks, the prediction values represent the probability of class `1` (yes), which helps in interpreting confidence levels for each prediction.

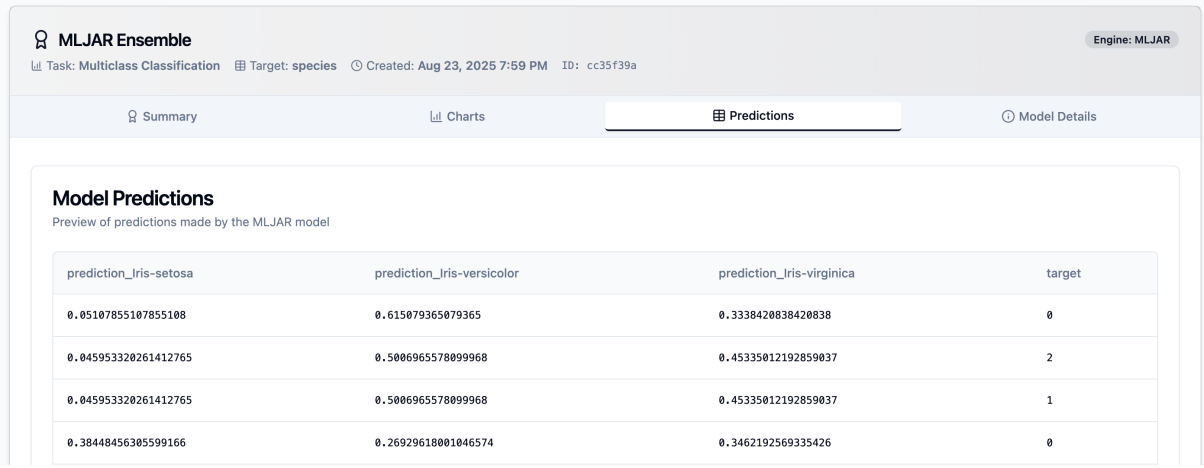


Figure 9.18: Predicted probabilities and target values preview

9.5.4 Model Details

The **Model Details** tab offers access to all experiment artifacts and technical outputs. This includes:

- **Downloadable model file**: usually a serialized object (e.g., `.pkl`, `.zip`)
- **Metadata file**: includes full configuration and results in JSON format
- **Model leaderboard**: a CSV showing all evaluated models, their types, training times, and metric values
- **README**: machine-generated documentation explaining the AutoML configuration and performance

The leaderboard is especially helpful for understanding which individual models contributed to the final ensemble and their respective performance metrics.

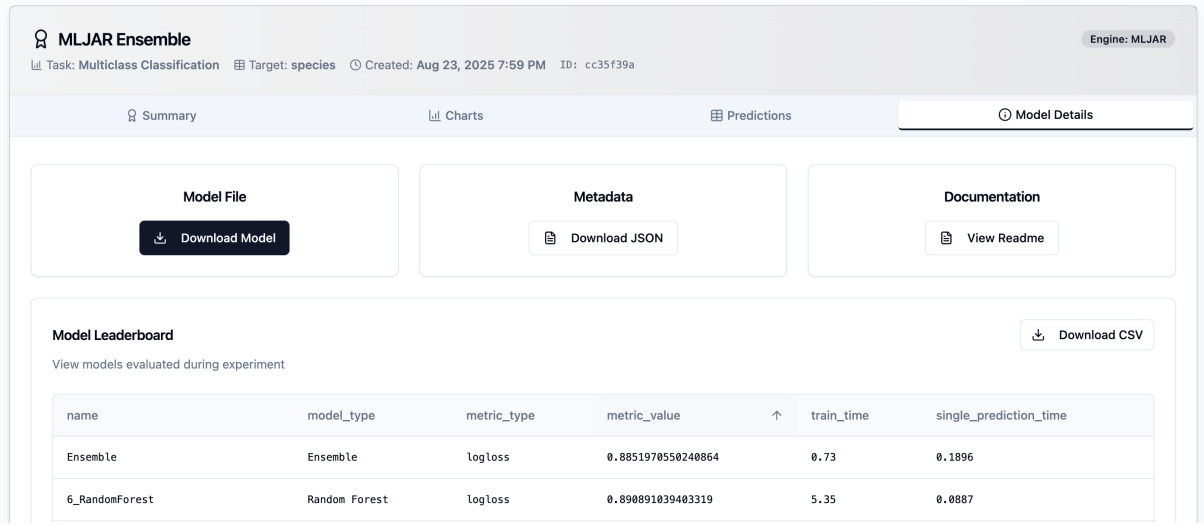


Figure 9.19: Model leaderboard, downloadable model, metadata, and documentation files

9.6 Prediction

After training a model and reviewing its evaluation metrics, users can proceed to the **Predict** tab to make predictions using the trained model. The application supports two modes of prediction:

- **Predict Manually**
- **Predict from CSV**

9.6.1 Predict Manually

In the **Predict Manually** mode, the system dynamically generates an input form based on the dataset features used during training. Categorical features (e.g., `previous_loan_default`) appear as dropdown menus, while numerical features (e.g., `loan_int_rate`) are rendered as number input fields.

The user can enter a set of values corresponding to a single sample and click the *Generate Prediction* button. The system returns the predicted output label along with the model's confidence score. For classification tasks, the class probabilities are also displayed, giving insight into the model's certainty.

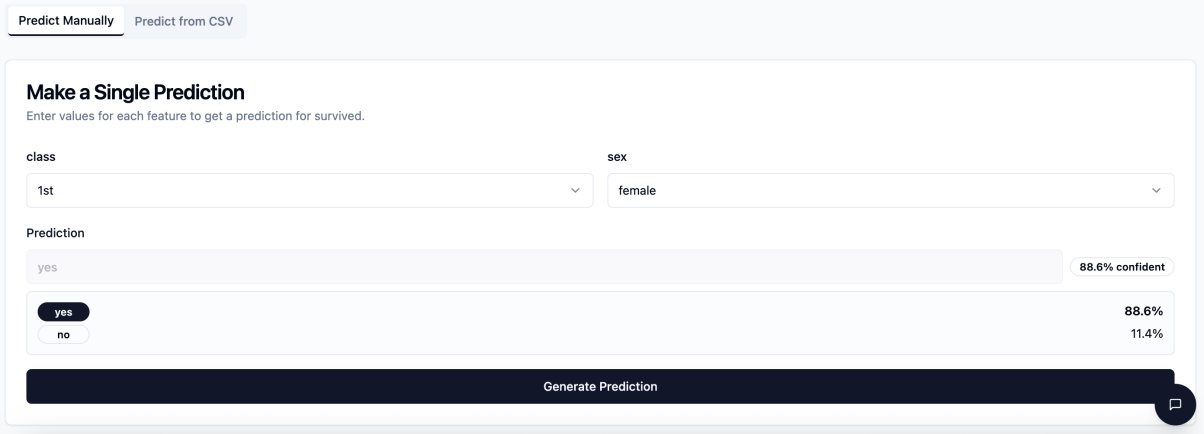


Figure 9.20: Manual prediction form with auto-generated inputs based on model features

9.6.2 Predict from CSV

The **Predict from CSV** option allows users to upload a dataset file for batch prediction. The system supports two workflows:

1. **Evaluation Mode:** The uploaded CSV includes the target column. The system compares the predicted values with the actual targets and computes evaluation metrics (e.g., Accuracy, Precision, Recall, F1-Score). It also shows a detailed comparison table highlighting correct and incorrect predictions.
2. **Inference Mode:** The uploaded CSV does **not** include a target column. The system predicts the missing target for each row and appends the results to the uploaded dataset.

	A	B	C	D	E
1	person_income	loan_int_rate	loan_percent_incom	previous_loan_defau	loan_status
2	71948	16.02	0.49	No	1
3	12282	11.14	0.08	Yes	0
4	12438	12.87	0.44	No	1
5	79753	15.23	0.44	No	1
6	66135	14.27	0.53	No	1
7	12951	7.14	0.19	No	1
8	93471	12.42	0.37	No	1
9	95550	11.11	0.37	No	1
10	100684	8.9	0.35	No	1

Figure 9.21: CSV uploaded with target column — Evaluation Mode

	A	B	C	D
1	person_income	loan_int_rate	loan_percent_incom	previous_loan_defau
2	71948	16.02	0.49	No
3	12282	11.14	0.08	Yes
4	12438	12.87	0.44	No
5	79753	15.23	0.44	No
6	66135	14.27	0.53	No
7	12951	7.14	0.19	No
8	93471	12.42	0.37	No
9	95550	11.11	0.37	No
10	100684	8.9	0.35	No

Figure 9.22: CSV uploaded without target column — Inference Mode

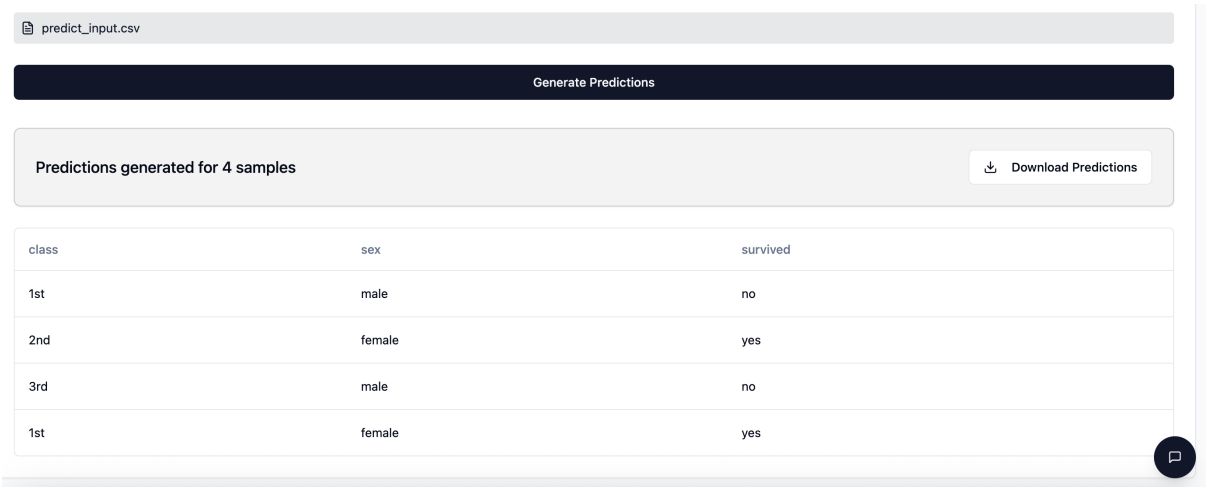
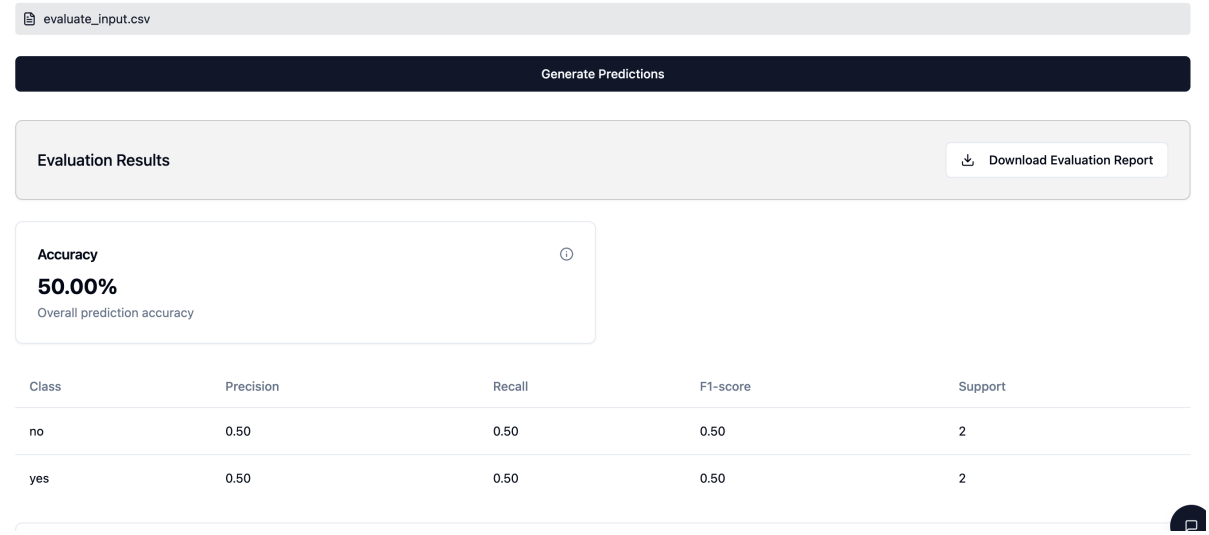
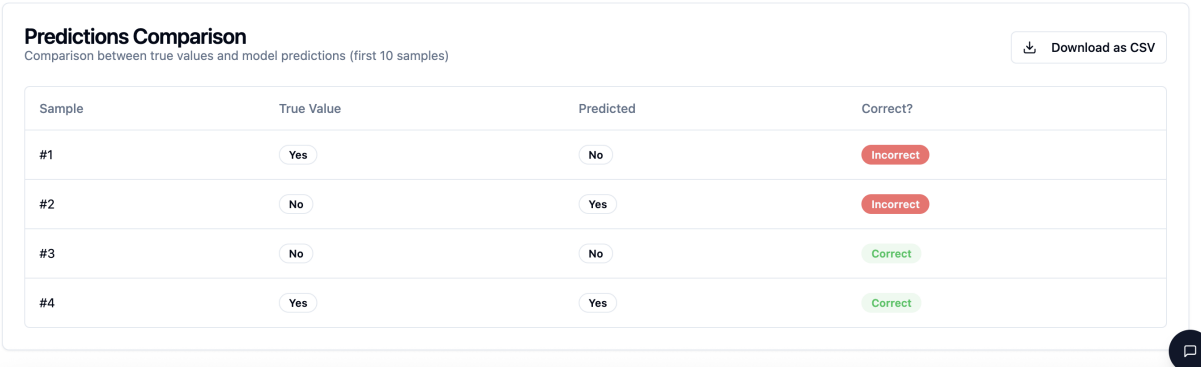


Figure 9.23: Predicted output appended to CSV rows in Inference Mode

If the CSV includes the true labels, the app computes class-wise evaluation metrics and a summary accuracy score. Each prediction is labeled as `Correct` or `Incorrect`, offering transparency into model performance on unseen data.



(a) Evaluation summary (part 1)



(b) Comparison between true and predicted values (part 2)

Figure 9.24: Evaluation summary and comparison between true and predicted values

All prediction outputs can be downloaded as CSV files, supporting further offline analysis or reporting.

9.7 Dashboard

The **Dashboard** section of `ieeAutoML` provides users with a centralized panel to manage all datasets, experiments, and saved comparisons. It is accessible via the main navigation and is designed to streamline user interaction with past activity and results.

The Dashboard contains three subtabs:

- **Datasets**
- **Experiments**
- **Comparisons**

9.7.1 Datasets Tab

The **Datasets** tab displays all datasets that the user has uploaded and processed. Each dataset entry reflects its full journey through the pipeline — from initial upload to the preprocessing stage.

Users can:

- **View** a dataset preview for any processing stage (raw, cleaned, final, or processed)
- **Download** the dataset version corresponding to each pipeline stage
- **Delete** datasets that are no longer needed

This interface ensures transparency and enables users to revisit and export datasets at any step of the machine learning pipeline.

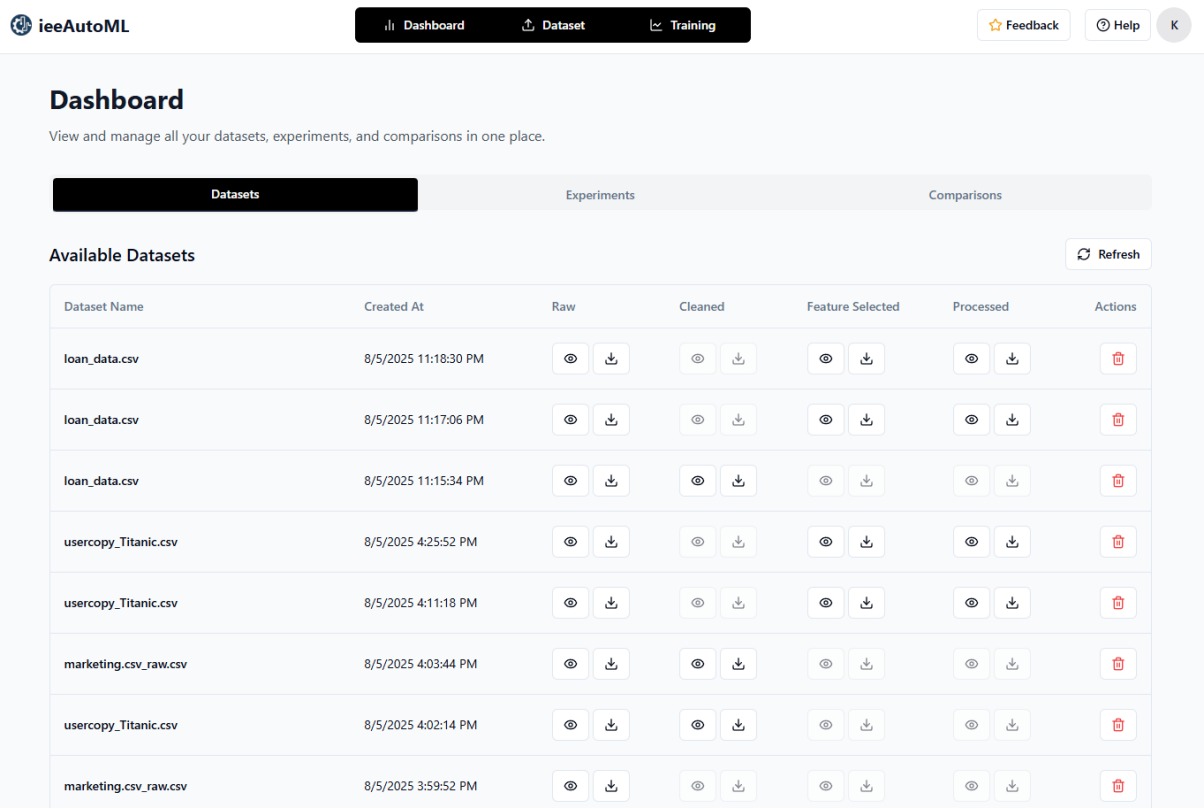


Figure 9.25: Datasets tab with preview, download, and delete options for each dataset stage

9.7.2 Experiments Tab

The **Experiments** tab offers a comprehensive view of all model training experiments conducted by the user. It features a filter panel that allows users to narrow down experiments by:

- **Experiment Type**: AutoML or Custom Training
- **Task Type**: Binary classification, Multiclass classification, or Regression
- **Search Field**: Text-based experiment name filtering

Each experiment entry provides multiple capabilities:

- **Info View**: Overview of experiment metadata and configuration.
- **Metrics**: Performance metrics such as accuracy, F1-score, MSE, etc.
- **Visuals**: All available charts from the training phase.
- **Download Resources**: Download model files, result summaries, and visualizations.
- **Custom Tuning (if applicable)**: For custom training experiments, users can re-tune hyperparameters and re-run the experiment.

- **Comparison Selection**: Mark experiments for comparison. This is only available for experiments of the same type and task, and requires at least two.
- **Delete**: Remove the experiment permanently.

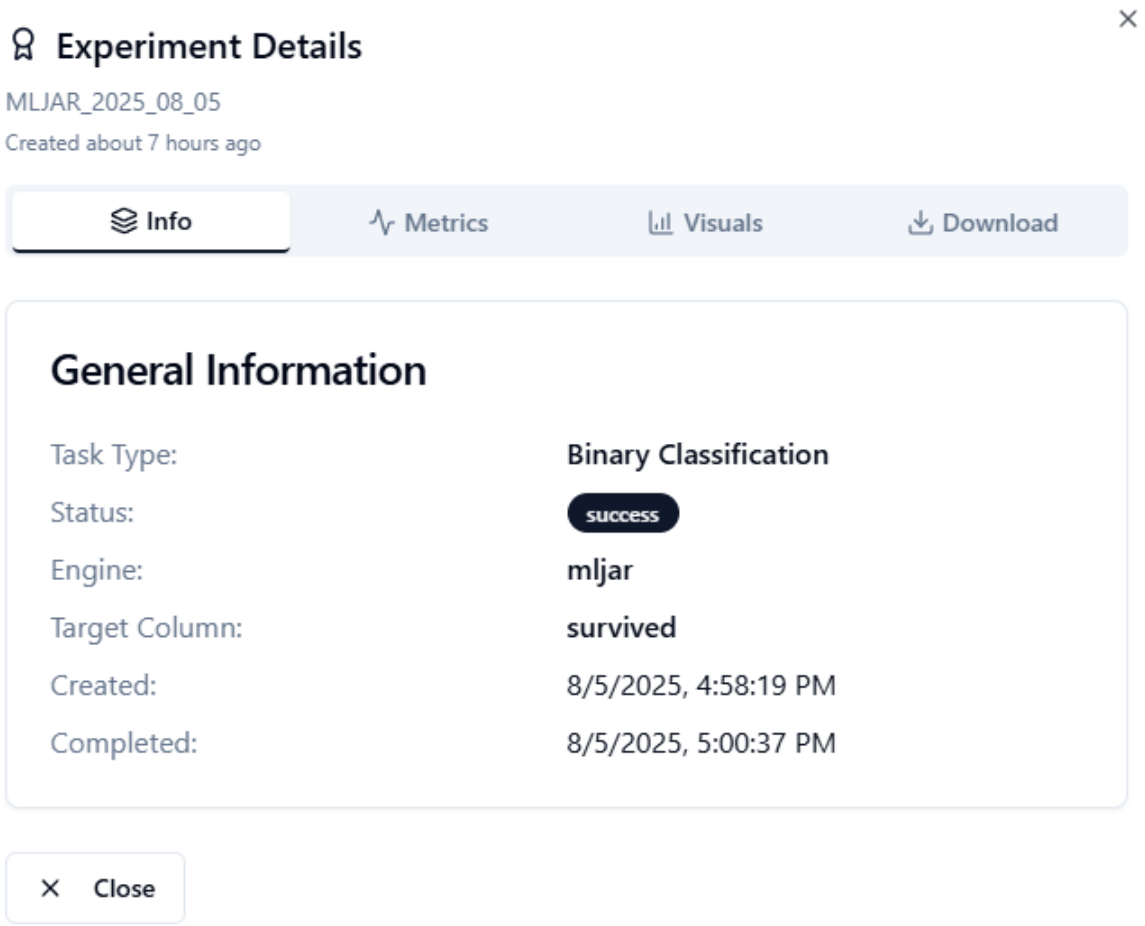


Figure 9.26: Experiment Info tab showing training configuration and metadata

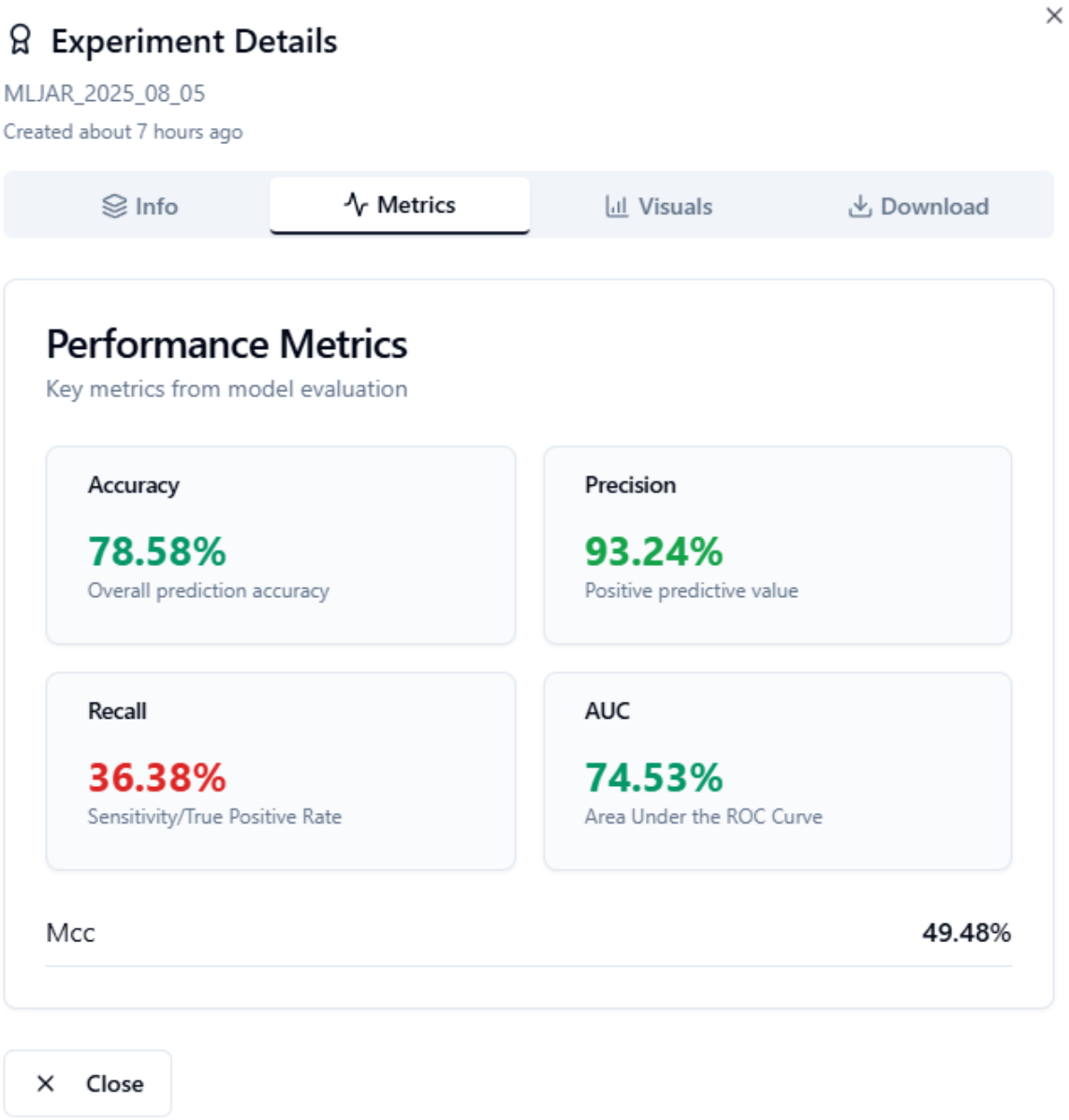


Figure 9.27: Experiment Metrics tab with detailed performance results

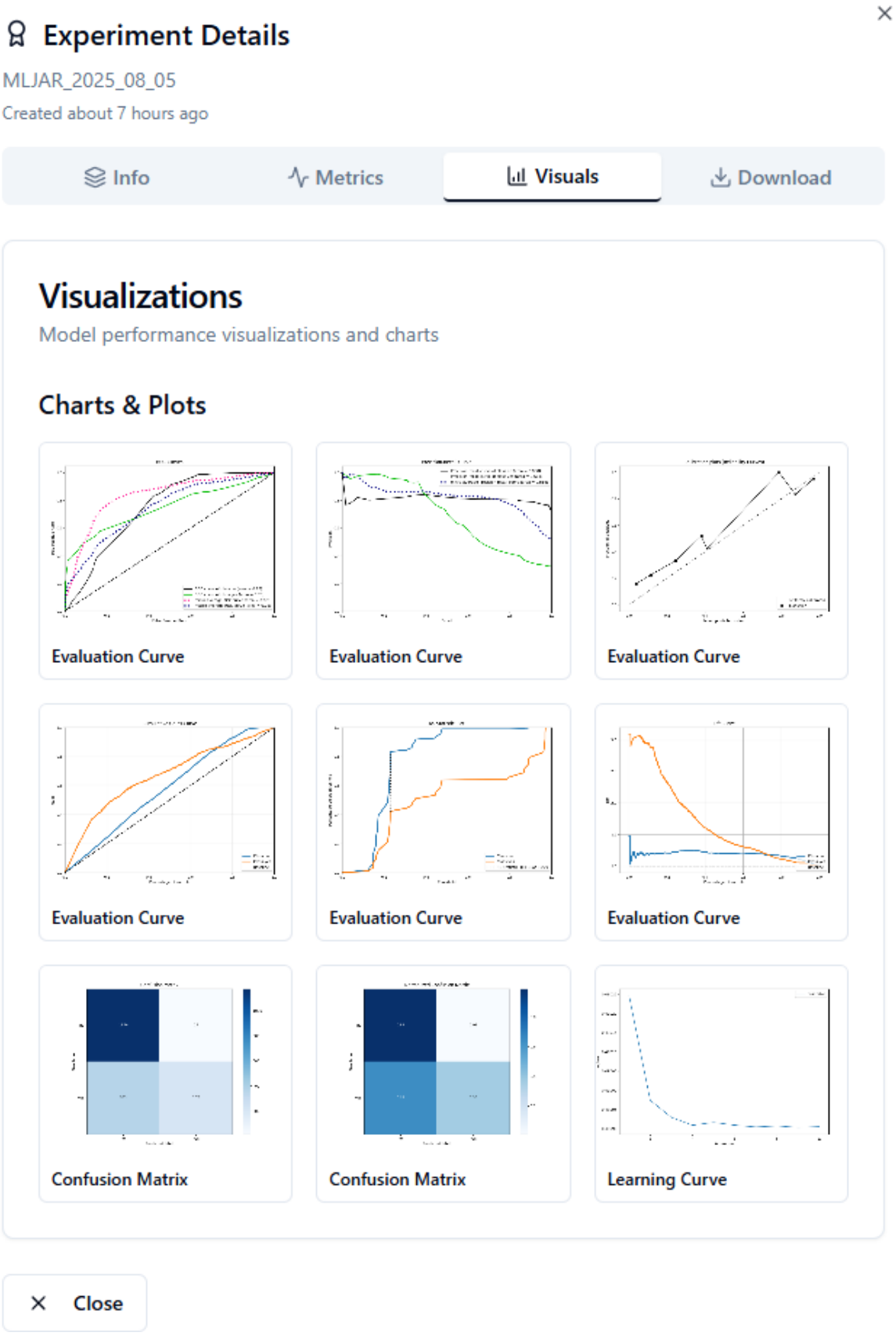


Figure 9.28: Experiment Visuals tab showing training charts

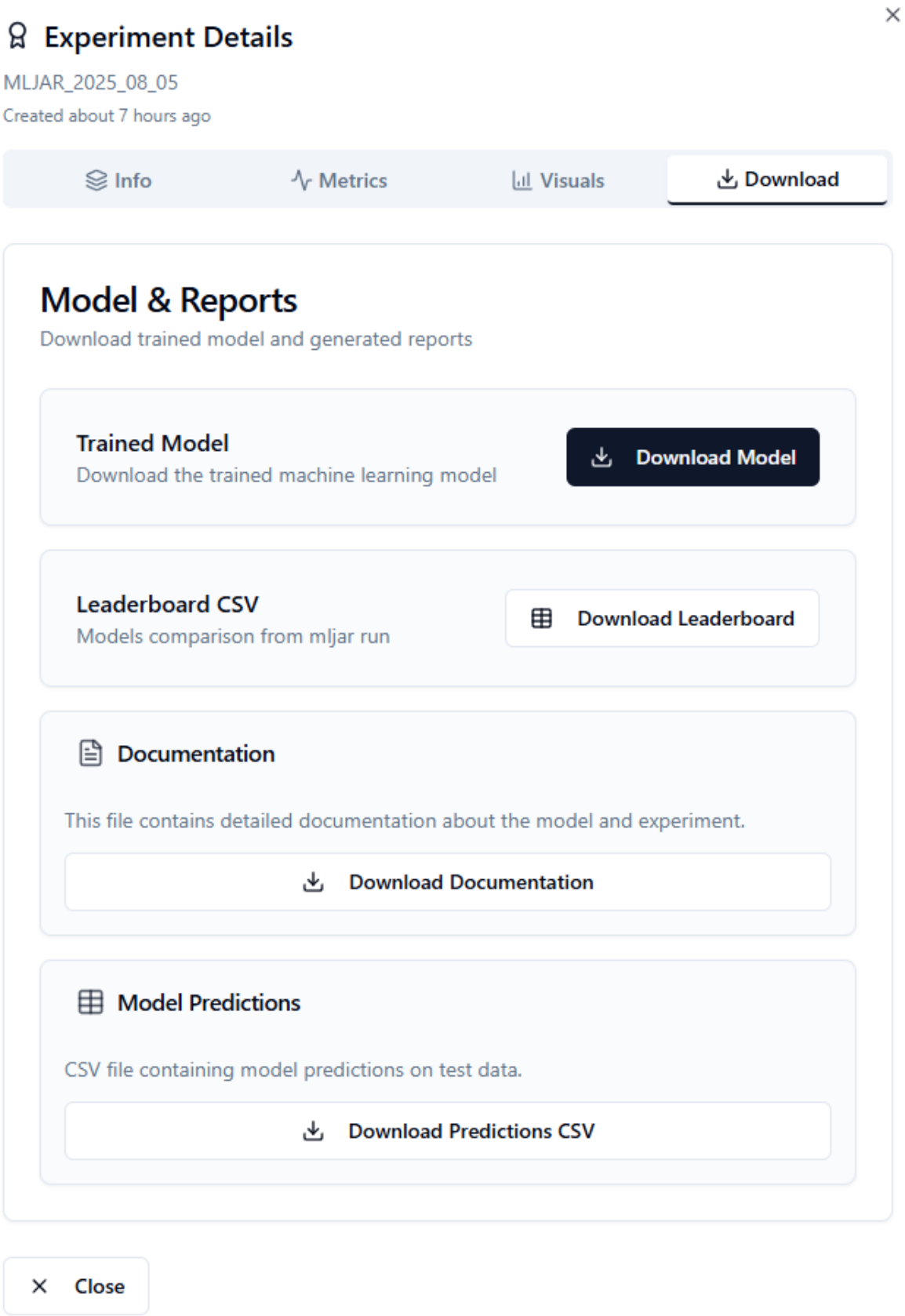


Figure 9.29: Download options for experiment results and models

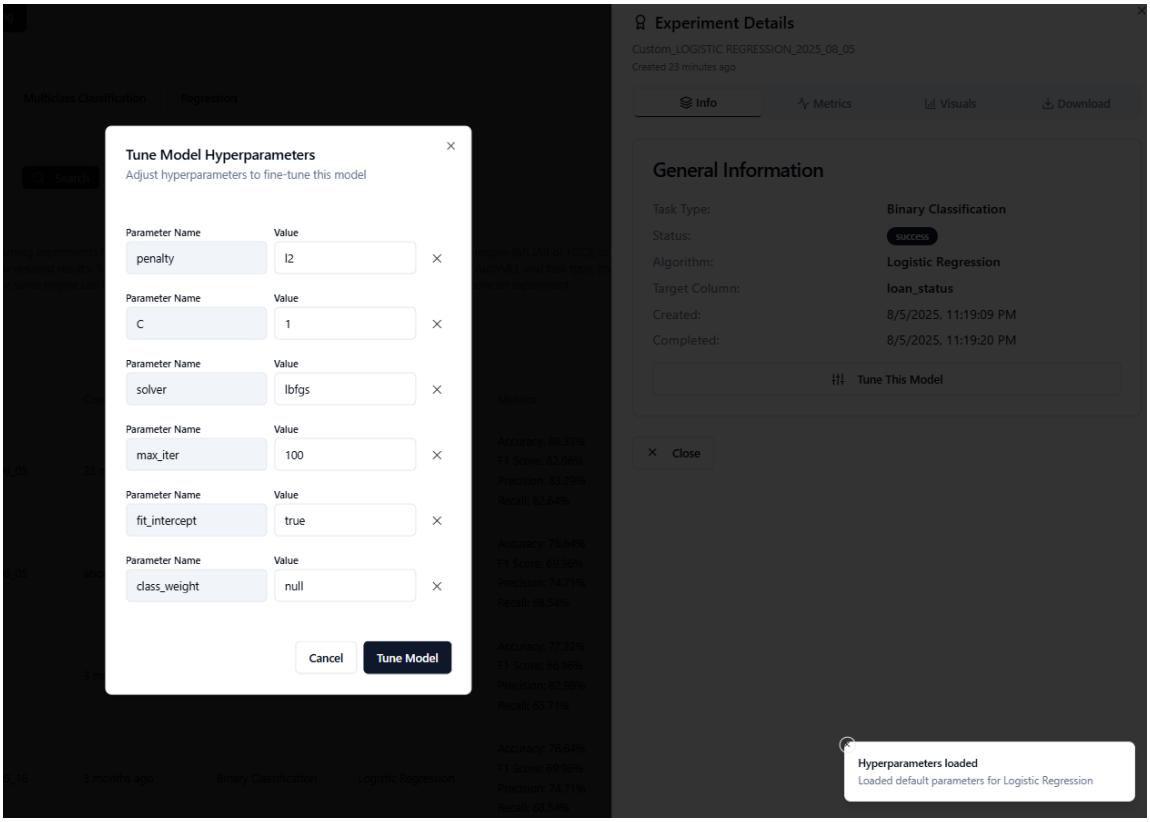


Figure 9.30: Custom tuning form for re-training models with different hyperparameters

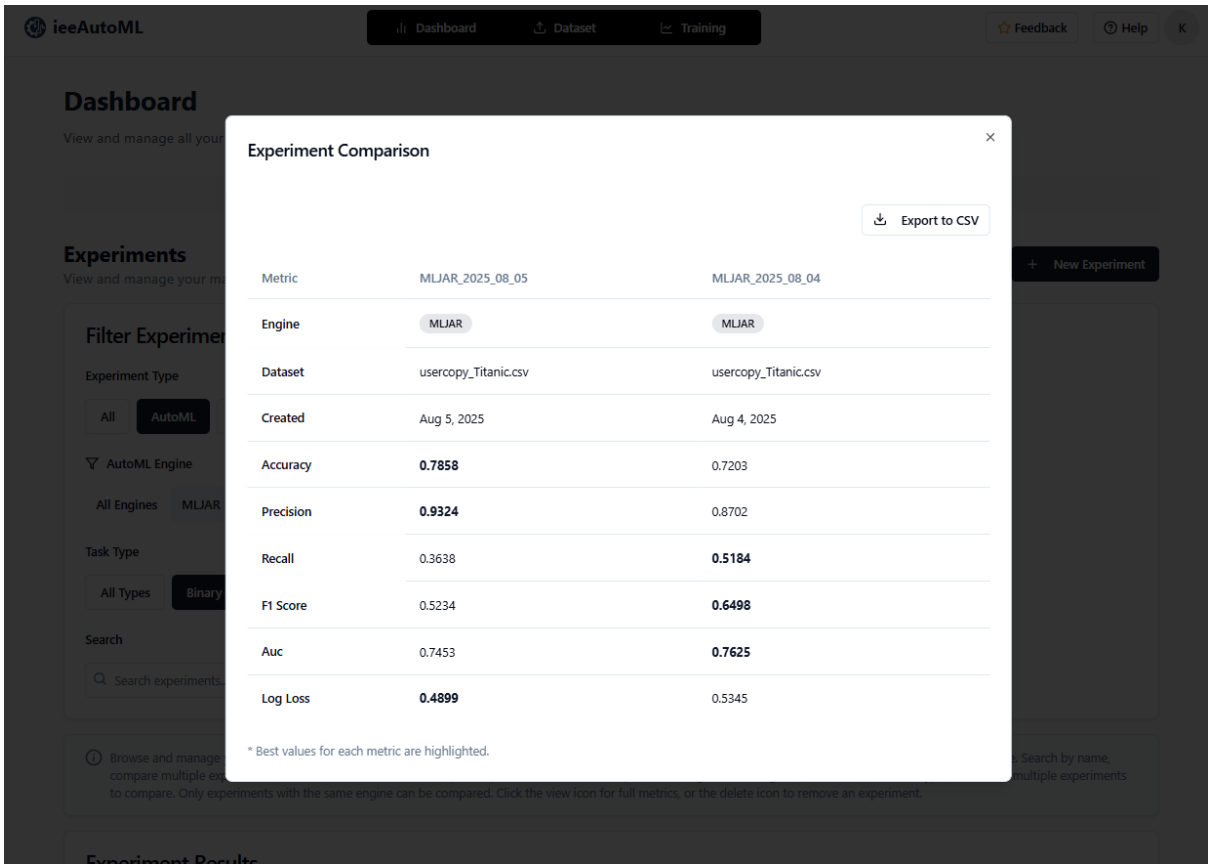


Figure 9.31: Experiment comparison selection panel

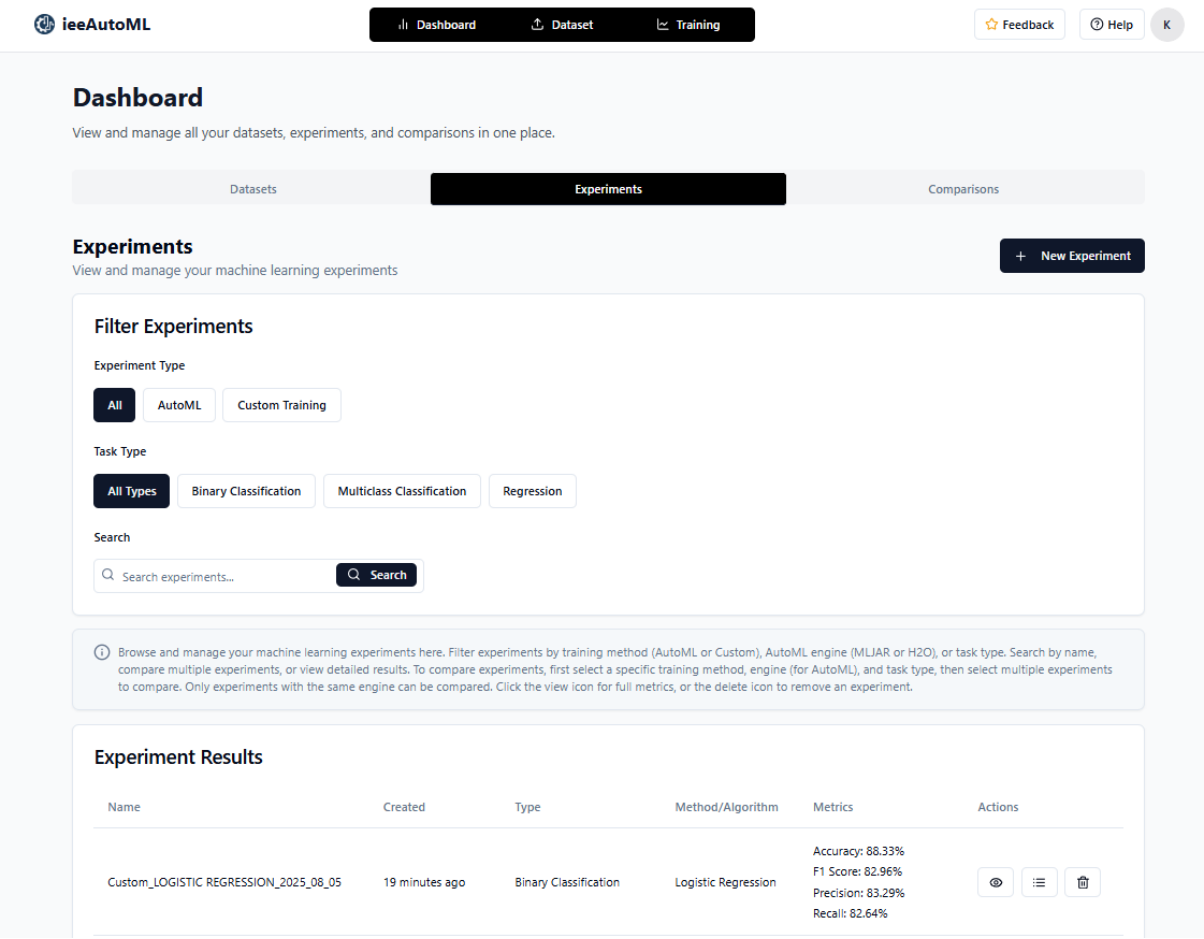


Figure 9.32: Experiments tab with filtering, result access, tuning, and comparison options

9.7.3 Comparisons Tab

The **Comparisons** tab stores previously selected experiments for comparative analysis. When a user selects two or more compatible experiments (same task and type) from the Experiments tab, a comparison session is generated and saved here.

Users can:

- **Review saved comparisons**: View side-by-side metrics and charts
- **Re-access past comparisons**: For model selection or result documentation

This tab provides an efficient way to compare AutoML runs or hyperparameter tuning outcomes from different custom models.

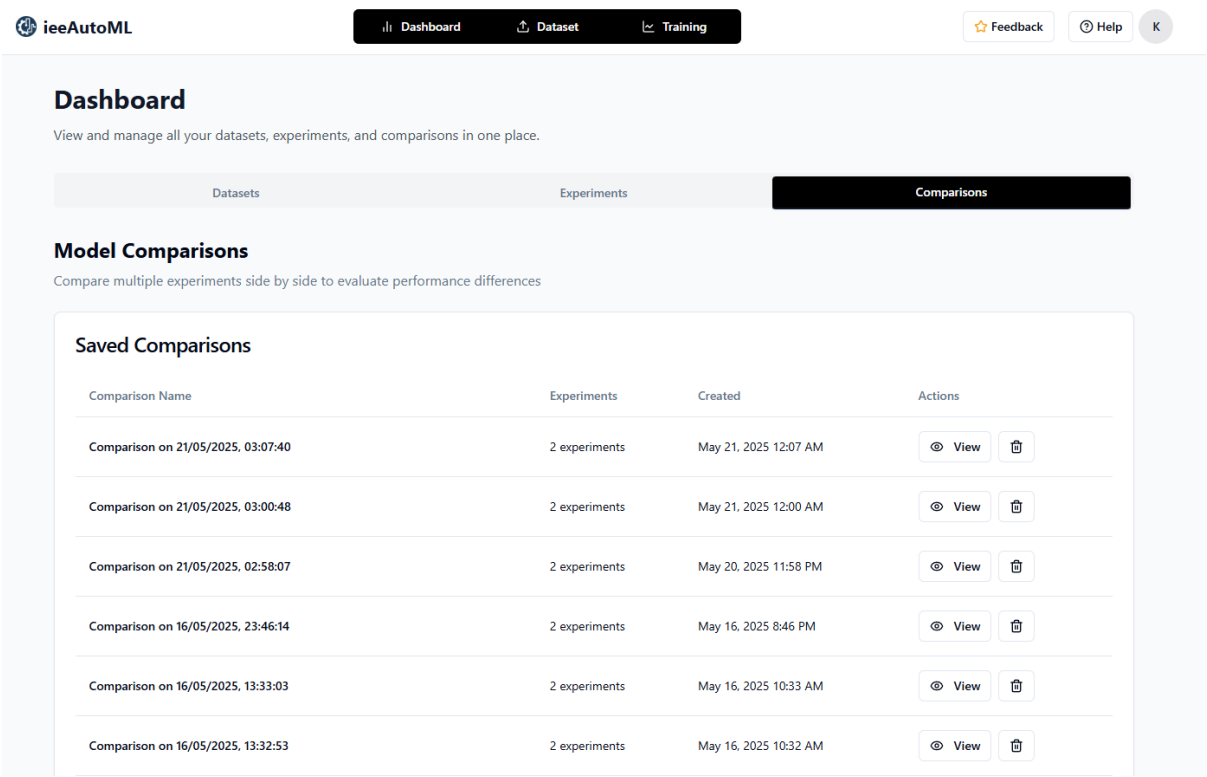


Figure 9.33: Comparisons tab showing saved experiment comparisons with detailed analysis

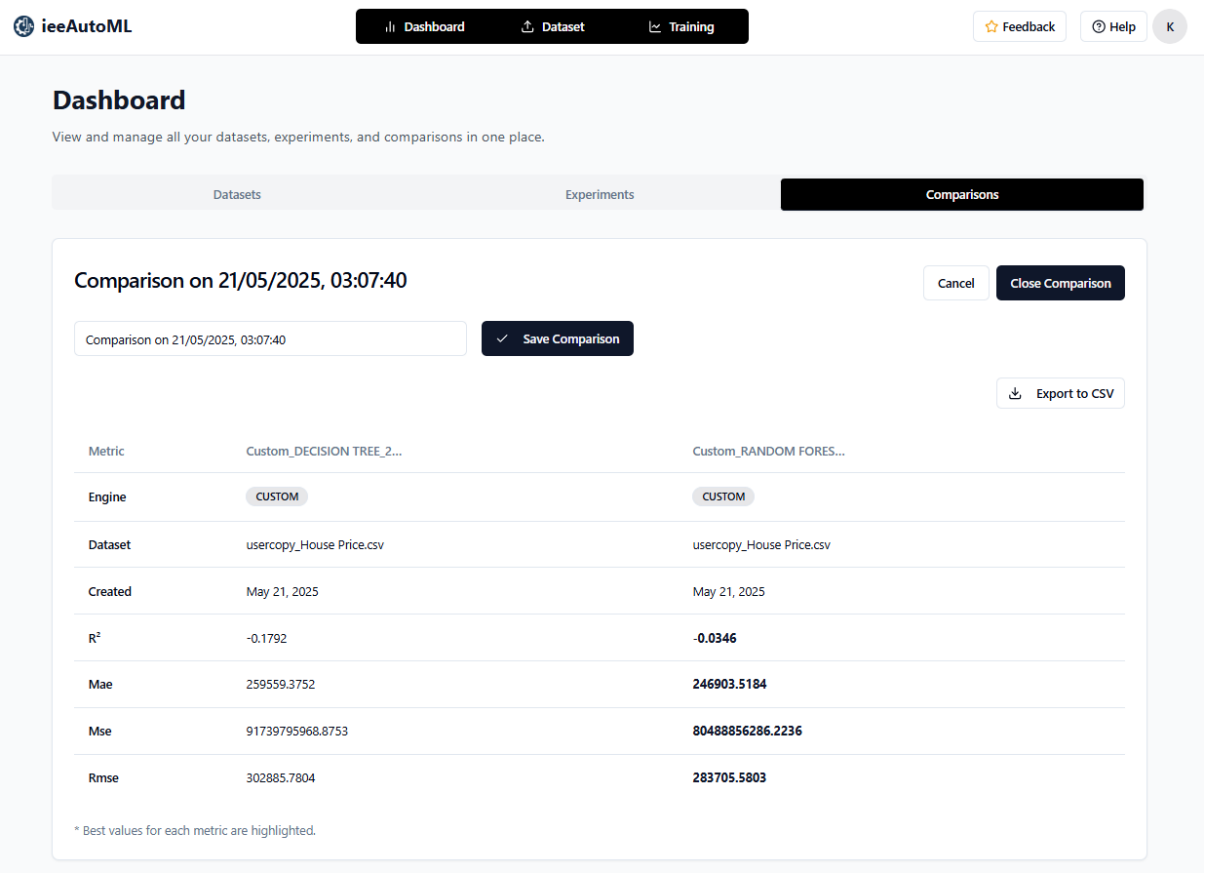


Figure 9.34: Comparison view between experiments showing detailed analysis

9.8 AI Assistance

To enhance user experience and promote understanding of machine learning outputs, `ieeAutoML` integrates an AI-powered assistant located in a bubble icon at the bottom-right corner of the application interface.

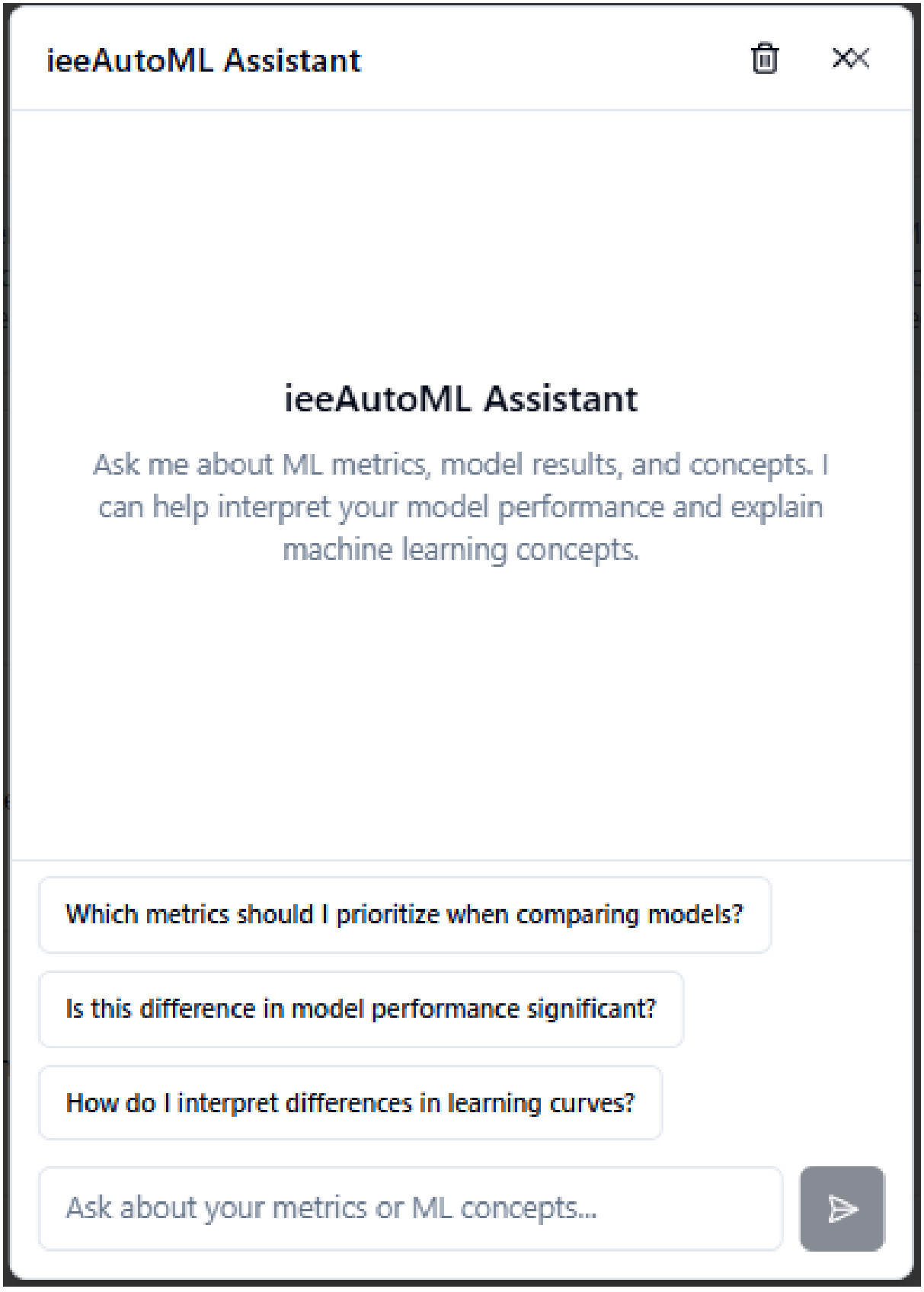


Figure 9.35: AI Assistant interface embedded in the app

The assistant is powered by **Mistral AI**, and its functionality has been fine-tuned via a system prompt to focus solely on machine learning-related queries such as:

- Interpreting metrics (e.g., accuracy, F1-score, MSE)
- Comparing model performances
- Understanding learning curves
- Providing guidance on model tuning or result interpretation

The assistant behaves like a traditional chat bot, allowing users to type custom questions or select from a list of suggested prompts. This enables non-expert users to gain insight into model performance without requiring deep ML knowledge.

By incorporating such an assistant, `ieeAutoML` promotes explainability, accessibility, and educational support throughout the training and evaluation pipeline.

Efficiency and Usability Evaluation

10.1 Efficiency Experiments

The evaluation of *ieeAutoML* performance included efficiency experiments which measured execution time and resource consumption and scalability across various operational conditions. The experiments simulated real-world supervised learning workflows by testing both automated and custom training methods.

10.1.1 Datasets

Three benchmark datasets widely used in research were chosen to evaluate *ieeAutoML* efficiency across classification and regression tasks.

- **Adult Income Dataset** [16]: The Adult dataset (also known as the Census Income dataset) contains 48,842 instances extracted from the 1994 U.S. Census database. Each record includes demographic and employment attributes, and the goal is to predict whether a person's income exceeds $50K per year (binary classification). This dataset tests performance on larger, mixed-type tabular data.
- **Wine Quality Dataset** [14]: The Wine Quality dataset includes 6,497 physicochemical and sensory records of red and white wines from Portugal. The task is to predict wine quality on a scale of 0–10 (multiclass classification / ordinal regression). This dataset provides higher dimensionality and noisy real-world data.
- **California Housing Dataset** [45]: The California Housing dataset consists of 20,640 records from the 1990 U.S. Census, with features describing demographic and geographical information (e.g., median income, average rooms, location coordinates). The target variable is the median house value, making it a large-scale regression benchmark with real-world complexity.

The selected datasets provide a complete assessment of system performance because they include multiclass classification, binary classification, and regression tasks of varying scale and complexity.

10.1.2 Experimental Setup

All experiments for evaluating the efficiency of *ieeAutoML* were conducted on a personal The experiments for evaluating the efficiency of ieeAutoML were run on a personal laptop that had an **MacBook Air** with **M1 chip** (Apple Silicon). The device specifications are as follows:

- **Processor:** Apple M1 (8-core CPU, 7-core GPU, 16-core Neural Engine)
- **Memory:** 8 GB unified RAM
- **Storage:** 256 GB SSD
- **Operating System:** macOS Ventura 14.x

The experiments were run using Python 3.11 within a virtual environment, and all system dependencies were managed via `pip`. The backend of *ieeAutoML* (FastAPI and Scikit-learn–based training services) was executed locally, while Supabase was used as the external PostgreSQL database and storage layer. The frontend was accessed via a local development server.

To ensure reproducibility and consistency across experiments, the following practices were applied:

- Fixed random seed for dataset splitting and algorithm initialization.
- Consistent preprocessing steps (imputation, normalization, and encoding) across datasets.
- Stratified train-test splits (where applicable) with a default test size of 20%.

10.1.3 Measurements

To evaluate the efficiency and predictive capability of *ieeAutoML*, two sets of experiments were conducted: one with the AutoML engines (MLJAR and H2O) and one with individual custom algorithms. The following measurements were recorded for each run:

1. **Training Time**: Duration from model initialization to completion of training.
2. **Model Performance**: Measured by classification accuracy for classification tasks and coefficient of determination R^2 for regression tasks.

AutoML Engines

Table 10.1 compares the performance of the two AutoML engines (MLJAR and H2O) across the selected benchmark datasets. Results are reported in terms of average training time and achieved predictive performance.

Table 10.1: Comparison of AutoML engines (MLJAR vs H2O) on benchmark datasets

Dataset	**Engine**	**Training Time (s)**	**Performance**
Adult Income	MLJAR	120	85.2% (Accuracy)
Adult Income	H2O	150	86.1% (Accuracy)
Wine Quality	MLJAR	95	71.3% (Accuracy)
Wine Quality	H2O	110	72.5% (Accuracy)
California Housing	MLJAR	140	0.84 (R^2)
California Housing	H2O	165	0.86 (R^2)

Custom Algorithms

Table 10.2 reports the results for individual custom training algorithms on the same datasets. Each algorithm was trained independently, and both training time and predictive performance were recorded.

Table 10.2: Performance of custom algorithms on benchmark datasets

Dataset	**Algorithm**	**Training Time (s)**	**Performance**
Adult Income	Logistic Regression	8	83.5%
Adult Income	Decision Tree	6	80.2%
Adult Income	Random Forest	20	85.6%
Adult Income	XGBoost	28	86.5%
Adult Income	LightGBM	25	86.3%
Adult Income	CatBoost	35	86.8%
Adult Income	Neural Network	60	85.1%
Adult Income	Extra Trees	18	84.7%
Adult Income	Nearest Neighbors	12	78.5%
Wine Quality	Logistic Regression	6	66.4%
Wine Quality	Decision Tree	4	63.1%
Wine Quality	Random Forest	15	70.8%
Wine Quality	XGBoost	22	71.9%
Wine Quality	LightGBM	19	71.6%
Wine Quality	CatBoost	30	72.2%
Wine Quality	Neural Network	55	71.0%
Wine Quality	Extra Trees	14	70.3%
Wine Quality	Nearest Neighbors	10	64.7%
California Housing	Linear Regression	7	0.73 (R^2)
California Housing	Decision Tree	5	0.68 (R^2)
California Housing	Random Forest	22	0.82 (R^2)
California Housing	XGBoost	30	0.85 (R^2)
California Housing	LightGBM	27	0.85 (R^2)
California Housing	CatBoost	40	0.86 (R^2)
California Housing	Neural Network	65	0.83 (R^2)
California Housing	Extra Trees	20	0.81 (R^2)
California Housing	Nearest Neighbors	12	0.64 (R^2)

From these results, AutoML engines demonstrate consistently strong predictive performance, often matching or exceeding the best custom algorithm, albeit at the cost of longer training times. Custom algorithms, however, provide faster training and allow more direct interpretability and control, making them suitable in scenarios where resources or time are constrained.

10.2 System Usability Scale (SUS)

We examined the usability of the *ieeAutoML* application by using the System Usability Scale (SUS) [10]. SUS is a widely adopted method to quickly assess how user-friendly and useful a system is. More specifically, the SUS questionnaire was shared with individuals who interacted with the application during testing. The majority of respondents were undergraduate students with a background in Data Mining and Machine Learning. The questionnaire contains ten statements, each rated on a scale from 1 (Strongly Disagree) to 5 (Strongly Agree). The full list of statements is shown below:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

To calculate the final SUS score, each response is first aligned with a 1 to 5 scale, where 1 corresponds to "Strongly Disagree" and 5 to "Strongly Agree." For the odd-numbered questions, the value of 1 is subtracted from the given response. For the even-numbered questions, the response is subtracted from 5. All resulting values are then summed and multiplied by 2.5, producing a score out of 100 for each participant. The overall usability score is obtained by calculating the average of all participants' scores. A score greater than 80 is generally considered to reflect excellent usability [10].

Table 10.3 presents the responses from one test user, who completed the SUS questionnaire after using the application. The resulting SUS score was **85**, which is well above the threshold of 80, indicating excellent usability and high user satisfaction with the system.

Table 10.3: System Usability Scale (SUS) results for ieeAutoML

User	**Score (0–100)**
1	88
2	82
3	90
4	85
5	80
6	87
7	83
8	85
9	89
10	84
11	86
12	81
13	88
14	84
15	82
16	90
17	79
18	87
19	83
20	86
21	84
22	88
23	82
24	85
25	87
Result (Average)	**85.0**

The high score reflects that participants perceived the application as easy to use, consistent, well-integrated, and not unnecessarily complex. In particular, users strongly agreed that they would like to use the system frequently, felt confident during usage, and believed that most people could quickly learn how to operate the application. These findings confirm that the usability of *ieeAutoML* meets excellent standards and supports its adoption as a practical tool for non-expert users.

Conclusion and Future Work

11.1 Conclusion

The thesis presented *ieeAutoML* as a web-based platform that simplifies supervised learning workflows through its unified interface which merges custom training with automated training capabilities. The system allows users at all skill levels to upload datasets which then undergo preprocessing before model training and results analysis through visualizations and leaderboards.

The evaluation results showed *ieeAutoML* maintains consistent preprocessing performance when dealing with datasets of different sizes and complexities. The MLJAR and H2O AutoML engines achieved competitive accuracy results but the custom training pipeline provided faster results for specific experimental needs. The modular architecture provides scalability and maintainability with future extensibility features which make it appropriate for academic and industrial applications.

The combination of explainability features with dataset version tracking and user-friendly UI in *ieeAutoML* makes machine learning tools more accessible to a broader audience while reducing entry barriers for effective domain-wide adoption.

11.2 Future Work

Future development should focus on adding **unsupervised learning** capabilities to the platform especially clustering methods. The implementation of AutoML pipelines for clustering needs automation of cluster number estimation and initialization strategies with internal metrics like silhouette score and Davies–Bouldin index and external validation through labeled data when possible. The supervised AutoML approach is less challenging than unsupervised AutoML since clustering operates without predefined labels thus requiring advanced search techniques and interpretability tools.

The system would perform automatic structure detection on datasets through this expansion allowing users to perform customer segmentation and anomaly detection and exploratory data analysis. The expansion of explainability features with SHAP value adaptation for clustering and cluster stability visualization would create new possibilities for unsupervised AutoML transparency. The development of a **unified AutoML ecosystem** through *ieeAutoML* will achieve its ultimate goal by supporting supervised and unsupervised paradigms to empower a wider range of users and applications.

Bibliography

[1] N. S. Altman. *An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression*. The American Statistician, 1992.

[2] Mitra Baratchi, Can Wang, Steffen Limmer, Jan N. van Rijn, Holger Hoos, Thomas Bäck, and Markus Olhofer. Automated machine learning: Past, present and future. *Artificial Intelligence Review*, 57, 2024.

[3] Gustavo E. A. P. A. Batista and Maria Carolina Monard. An analysis of four missing data treatment methods for supervised learning. *Applied Artificial Intelligence*, 17(5-6):519–533, 2003.

[4] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[6] C. S. Bosson and T. Nikoleris. Supervised learning applied to air traffic trajectory classification. In *AIAA Aviation Technology, Integration, and Operations Conference*, 2018.

[7] Paula Branco, Luís Torgo, and Rita P. Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys*, 49(2):1–56, 2016.

[8] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[9] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, Monterey, CA, 1984.

[10] John Brooke. Sus: A quick and dirty usability scale. Technical Report TR-38, Digital Equipment Corporation, Reading, UK, 1996.

[11] G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[12] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[13] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[14] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[15] Lucas B. V. de Amorim, George D. C. Cavalcanti, and Rafael M. O. Cruz. The choice of scaling technique matters for classification performance. *Expert Systems with Applications*, 189:116290, 2022.

[16] Dheeru Dua and Casey Graff. Uci machine learning repository. `http://archive.ics.uci.edu/ml`, 2019. Accessed: 2025-08-22.

[17] Tlamelo Emmanuel, Thabiso Maupong, Dimane Mpoeleng, Thabo Semong, Banyatsang Mphago, and Oteng Tabona. A survey on missing data in machine learning. *Journal of Big Data*, 8:140, 2021.

[18] Hugo Jair Escalante. Automated machine learning — a brief review at the end of the early years. In *Automated Design of Machine Learning and Search Algorithms*, Natural Computing Series, pages 11–28. Springer, 2021.

[19] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Auto-sklearn: Efficient and robust automated machine learning. In *Automated Machine Learning*, pages 62–73. Springer, 2019.

[20] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

[22] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[23] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. In *Proceedings of the International Conference on Intelligent Computing (ICIC 2005)*, volume 3644 of *Lecture Notes in Computer Science*, pages 878–887. Springer, 2005.

[24] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[25] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[26] Haibo He, Yang Bai, Eduardo A. Garcia, and Shutao Li. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2008)*, pages 1322–1328. IEEE, 2008.

[27] Haibo He and Edward A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.

[28] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[29] David W. Hosmer, Stanley Lemeshow, and Rodney X. Sturdivant. *Applied Logistic Regression*. Wiley, 3rd edition, 2013.

[30] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, New York, 1st edition, 2013.

[31] Luke Oluwaseye Joel, Wesley Doorsamy, and Babu Sena Paul. A review of missing data handling techniques for machine learning. *International Journal of Innovative Technology and Interdisciplinary Sciences*, 5(3):971–1005, 2022.

[32] Thomas Jones and David Pratt. Comparing classification algorithms for tree hazard prediction. *Journal of Arboriculture*, 31(2):78–85, 2005.

[33] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[34] Soonchang Kwon. Imputation of missing data based on hot deck method using k-nn. *Journal of the Korea Society of IT Services*, 13(4):359–375, 2014.

[35] Jorma Laurikkala. Improving identification of difficult small classes by balancing class distribution. In *Proceedings of the 8th International Conference on Artificial Intelligence in Medicine in Europe (AIME 2001)*, volume 2101 of *Lecture Notes in Artificial Intelligence*, pages 63–66. Springer, 2001.

[36] Erin LeDell and Sebastien Poirier. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, 2020.

[37] JiaHang Li, ShuXia Guo, RuLin Ma, Jia He, XiangHui Zhang, DongSheng Rui, YuSong Ding, Yu Li, LeYao Jian, Jing Cheng, and Heng Guo. Comparison of the effects of imputation methods for missing data in predictive modelling of cohort study datasets. *BMC Medical Research Methodology*, 24:41, 2024.

[38] Jing Li, Kecheng Liu, Shouyang Wang, Yi Deng, and Junbo Wang. Feature selection: A data perspective. *ACM Computing Surveys*, 50(6):94:1–94:45, 2017.

[39] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[40] Microsoft. What is automated machine learning (automl)? `https://learn.microsoft.com/azure/machine-learning/concept-automated-ml`, 2025.

[41] MLJAR. Mljar supervised: Automated machine learning python package. `https://github.com/mljar/mljar-supervised`, 2023.

[42] Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. *Introduction to Linear Regression Analysis*. Wiley, 5th edition, 2012.

[43] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[44] Randal S. Olson and Jason H. Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Proceedings of the Workshop on Automated Machine Learning*, pages 66–74, 2016.

[45] R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

[46] Harry Peter. Automated machine learning (automl): The future of model selection and tuning, 2025. Available as preprint (June 2025).

[47] João M. H. Pinheiro, Suzana V. B. Oliveira, Thiago H. S. Silva, Pedro A. R. Saraiva, Enzo F. de Souza, Leonardo A. Ambrosio, and Marcelo Becker. The impact of feature scaling in machine learning: Effects on regression and classification tasks. *arXiv preprint arXiv:2506.08274*, 2025.

[48] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[49] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[50] Iqbal H. Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2(3):160, 2021.

[51] Ivan Tomek. Two modifications of CNN. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(11):769–772, 1976.

[52] Wikipedia contributors. Feature scaling. `https://en.wikipedia.org/wiki/Feature_scaling`. Accessed 2025-06-27.

[53] D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-2(3):408–421, 1972.

[54] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, and Zhi-Hua Zhou. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

[55] Peter Wulff, Markus Kubsch, and Christian Krist. Applying supervised ml. In *Applying Machine Learning in Science Education Research*, Springer Texts in Education. Springer, Cham, 2025.

[56] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. CRC press, 2012.