DIY AUTOML PLATFORM

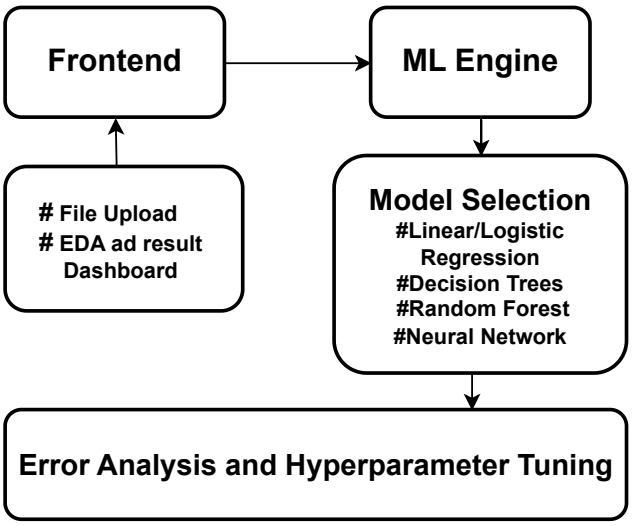
Introduction

In today's data-driven landscape, businesses generate substantial volumes of structured information—from daily sales figures and customer feedback to inventory levels and website analytics. Yet many organizations lack the technical expertise or resources to harness machine learning effectively. The DIY AutoML Platform addresses this gap by providing a lightweight, web-based solution that automates the end-to-end workflow: from exploratory data analysis (EDA) and model recommendation to training, prediction, and interpretability—all without relying on high-level libraries like scikit-learn or TensorFlow.

By building core algorithms from first principles (e.g., normal equations for linear regression, entropy-based splits for decision trees, and backpropagation for neural networks), the platform not only offers transparency into each computational step but also fosters deeper understanding of the underlying mathematics. Users can simply upload a CSV, select their target variable, and let the system intelligently recommend and train the best-fit model. Rich visualizations and custom feature-importance metrics then clarify why the model makes certain predictions, empowering users to make data-backed decisions with confidence.

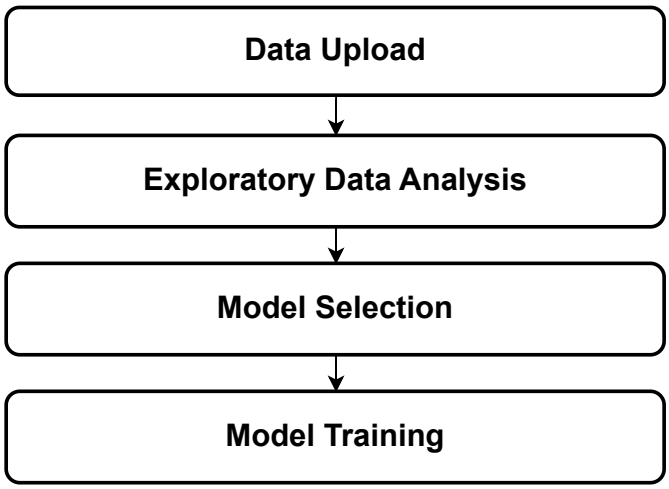
This report details the system's architecture, core components, and implementation strategies, illustrating how a math-first AutoML engine can be both accessible and powerful for non-technical stakeholders. The following sections will explore the platform's design, algorithmic foundations, and real-world use cases, culminating in a discussion of potential enhancements and future directions.

Core Components



DIY AutoML Platform is structured around five core modules. Frontend UI enables users to upload CSV data, select target columns, and explore results via automated EDA visualizations. ML Engine implements algorithms from scratch —linear and logistic regression, decision trees, random forests, neural networks (with optional PCA/K-means)—and computes custom metrics like R^2, accuracy, and F1-score. Model Selection Logic uses heuristics based on data types and tasks to recommend optimal models, while Hyperparameter Tuning automates grid or random search. Interpretability Module delivers global feature importance and local sensitivity analyses. Utilities handle model persistence, export predictions to CSV, and deploy with minimal setup.

High Level System Architecture



At the highest level, the DIY AutoML Platform consists of a client-side interface (React or Streamlit) that communicates via RESTful APIs with a backend server (Flask/FastAPI). The server handles data ingestion and preprocessing, then delegates to the ML Engine, which runs custom linear/logistic regression, decision trees, random forests, and neural networks. A Model Selection service picks appropriate algorithms and hyperparameters, feeding tasks into the Training Pipeline. Trained models and evaluation metrics flow into the Interpretability Module for feature-importance and sensitivity analyses. All artifacts—models, metrics, and prediction outputs—are stored in a persistent datastore and exposed for download or deployment. Monitoring and logging included.

Implementation Details

1. Data Ingestion & Preprocessing

i. CSV Loader

- **Uses Pandas to read user-uploaded files (`pd.read_csv`).**
- **Validates schema: checks for missing headers, enforces a minimum row count.**

ii. Data Cleaning

- **Missing values**: options to drop rows or impute (mean/median for numeric, mode for categorical).
- **Type inference:** auto-detect numeric vs. categorical via `df.dtypes`; allow manual overrides.
- **Encoding:**
  - Label-encode small-cardinality categorical.
  - One-hot encode larger sets via `pd.get_dummies`.

iii. Train/Test Split

- **Default 80/20 split using a random seed for reproducibility (`np.random.permutation`).**
- **Stratification for classification tasks (preserve class ratios).**

2. Exploratory Data Analysis (EDA)

i. Univariate Analysis

- **Histograms and boxplots (Matplotlib) for distributions and outliers.**
- **Summary table (mean, median, std, missing %) generated with df.describe() augmented by custom Pandas aggregations.**

ii. Bivariate Analysis

- **Correlation matrix heatmap (df.corr() → plt.imshow).**
- **Scatter plots of top correlated features vs. target.**

iii. Automated Report

- **Jinja2 template (backend) renders an HTML snippet of figures and tables.**
- **Frontend displays these via an embedded iframe or React component.**

3. Model Selection & Hyperparameter Tuning

i. Custom ML Engine

- **Linear Regression**
- **Logistic Regression**
- **Decision Trees**
- **Random Forest**
- **Neural Network**

ii. Task & Data Heuristics

- **If target is continuous → regression; else → classification.**
- **If features > 100 and rows > 10k, suggest tree-based or neural net for speed.**

iii. Grid/Random Search

- **User-configurable hyperparameter grids (e.g., learning_rate, max_depth, n_estimators).**
- **RandomSearch picks k random combinations; GridSearch exhaustively evaluates all.**
- **Performance measured on validation split; best model persisted.**

4. Exploratory Data Analysis (EDA)

i. Global Feature Importance

- **Linear/Logistic: absolute value of coefficients.**
- **Tree-based: average information gain per feature across all splits.**
- **Permutation importance: shuffle each feature; measure change in validation score.**

ii. Local Interpretability

- **Finite differences: perturb each input feature by ±ε and record Δ prediction.**
- **Present results in a table/chart for user-selected instances.**

5. Persistence, Export & Deployment

i. Model Persistence

- **Custom serialization: save NumPy arrays (parameters) and metadata (feature order, hyperparameters) to JSON + .npy.**

ii. Export Predictions

- **Builds a DataFrame of inputs + predicted output; df.to_csv() triggered by user click.**

iii. API Endpoints (Flask/FastAPI)

- **POST /upload → returns dataset schema.**
- **POST /train → triggers training, streams logs via Server-Sent Events.**
- **GET /predict → returns JSON of predictions for new data.**

6. Logging, Monitoring & Testing

i. Logging: Python's logging module captures pipeline events and exceptions; logs written to rotating files.

ii. Monitoring: lightweight Prometheus client tracks request latency and error rates.

iii. Unit Tests: PyTest suite covering edge cases—for example, zero-variance features, perfectly separable classes, and convergence failures.