
KÜHNE
LOGISTICS
UNIVERSITY

# Project: Automated Insight Engine

Fundamentals of Data Science

Prof. Dr. Henrik Leopold

Submitted by:

*Jitesh Gharat*

*Simran Singh*

Program: MS BADS (1st year)

Date of Submission: 21 December2025

## 1. Objectives

The primary goal of this project is to develop a comprehensive, end-to-end Automated Machine Learning (AutoML) pipeline. The objective was to build a robust application capable of processing raw, unseen datasets and guiding users through the entire data science lifecycle—from initial exploration to model evaluation without the need for manual coding for every new task.

## 2. Project Overview

The Automated Insight Engine is a web-based application developed using Python and Streamlit. Rather than being a static script designed for one specific problem, this engine serves as a dynamic framework that handles both Classification and Regression tasks.
The system follows a modular architecture that mirrors industry-standard workflows:

- Data Understanding: Automatically detects data types, schemas, and the specific prediction task.
- Diagnosis: Performs a "health check" to identify issues like missing values, duplicates, outliers, and data leakage.
- Cleaning & Feature Engineering: Automatically repairs data (imputation, outlier capping) and generates new features (date extraction, hashing) to boost model performance.
- Model Selection: Trains and ranks multiple algorithms (e.g., Random Forest, XG Boost, Logistic Regression) using cross-validation to find the best-performing model.
- Training & Evaluation: Conducts a final assessment of the winning model, providing detailed metrics, confusion matrices, and decision plots.

## 3. Motivation

The decision to build an AutoML tool instead of a traditional single-dataset project was driven by three main factors:

- Moving Beyond "Single Dataset" Constraints: Most projects focus on famous, static datasets (like Titanic or Housing Prices). By building a tool that handles *any* uploaded CSV, I had to write generalized, robust code that adapts to different data shapes and edge cases. This significantly strengthened my programming and logic skills.
- Broad Model Exploration: I wanted to go beyond learning just one or two algorithms. This project allowed me to compare a wide variety of models side-by-side. Building the "Leaderboard" logic helped me understand the practical trade-offs between model complexity, speed, and accuracy.
- Focusing on Data Quality: Since data cleaning often takes up 80% of a data scientist's time, I prioritized the Diagnosis and Cleaning modules. Automating these steps required a deep understanding of the statistical impact of missing

data and class imbalances, turning data preparation into a systematic, high-value process.

# 4. Technical Infrastructure

## 4.1 Technology Stack

The application is built on a modern Python-based stack, utilizing industry-standard libraries for data science and web deployment.

- Programming Language: * Python: The core language powering the entire application logic.
- Web Framework (UI): * Streamlit: Used to create the interactive web interface, managing the frontend rendering, file uploads, and multi-page state management.
- Data Manipulation & Analysis: * Pandas: The primary tool for data ingestion, cleaning, and statistical processing across the pipeline.
- Machine Learning Engine: * Scikit-Learn (sklearn): Handles the core ML operations, including algorithm implementation (Random Forest, Logistic Regression), data splitting, and performance metrics (F1-score, RMSE, etc.).
- Visualization & Reporting: * Plotly Express: Generates interactive charts, such as correlation heatmaps and the performance leaderboard.
- Matplotlib: Used for specialized visualizations, specifically rendering Decision Tree structures.
- Custom Modular Architecture: * Automl Package: A custom-built local library containing dedicated modules for cleaning, diagnostics, feature engineering, and model evaluation.

## 4.2 Supported Machine Learning Models

The Automated Insight Engine dynamically adapts to the data provided, supporting both Classification (predicting discrete labels) and Regression (predicting continuous numbers).

Classification Models

Used when the target variable is categorical (e.g., Yes/No, Spam/Not Spam).

- Linear/Statistical: Logistic Regression, Naive Bayes.
- Tree-Based: Decision Tree, Random Forest, XGBoost, LightGBM.
- Distance/Network Based: K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Neural Networks (MLP).

Regression Models

Used when the target variable is a numerical value (e.g., Price, Temperature).

- Linear: Linear Regression.
- Tree-Based: Decision Tree, Random Forest, XGBoost, LightGBM.
- Advanced: K-Nearest Neighbors (KNN), Support Vector Regressor (SVR), Neural Networks (MLP).

## 5. System Architecture (With Instance)

Taking a dataset is a comprehensive fitness and nutrition log containing 20,000 records. It integrates personal health metrics, detailed workout data, and nutritional tracking. Here are the features categorized for easier understanding:

1. Personal Health & Biometrics
    - Age, Gender, Weight (kg), Height (m): Standard demographic and physical profile data.
    - BMI & BMI_calc: Body Mass Index (both provided and calculated from height/weight).
    - Fat_Percentage: The percentage of body fat.
    - lean_mass_kg: Calculated lean body weight excluding fat.
    - Resting_BPM, Avg_BPM, Max_BPM: Heart rate metrics (Beats Per Minute) tracking cardiovascular health and intensity.
2. Exercise & Workout Details
    - Workout_Type: Categories such as HIIT, Strength, Cardio, or Yoga.
    - Name of Exercise: Specific movements (e.g., Decline Push-ups, Mountain Climbers).
    - Sets & Reps: Quantitative measures of the intensity of a specific exercise.
    - Session_Duration (hours): Total time spent exercising.
    - Workout_Frequency (days/week): How often the individual exercises weekly.
    - Experience_Level: Skill level (e.g., Beginner, Intermediate, Advanced).
    - Target Muscle Group, Body Part, Type of Muscle: Detailed anatomical focus of the exercises.
    - Equipment Needed: Tools required (e.g., Cable Machine, Dumbbells, Wall).
    - Difficulty Level: Perceived or set difficulty of the workout.
3. Performance & Energy Metrics
    - Calories_Burned: The amount of energy expended during the session.
    - Burns Calories (per 30 min): A rate metric for how efficient an exercise is at burning energy.
    - expected_burn: A predicted value for calorie expenditure.
    - pct_HRR & pct_maxHR: Percentage of Heart Rate Reserve and Heart Rate Max used, indicating effort level.
    - cal_balance: The difference between calories consumed and calories burned.
4. Nutrition & Diet
    - Calories: Total caloric intake from meals.
    - Carbs, Proteins, Fats: Macronutrient breakdown in grams.
    - cal_from_macros: Calculated calories based on the macronutrient profile.
    - diet_type: Specific dietary preference (e.g., Vegan, Vegetarian, Paleo).
    - meal_name & meal_type: Details about specific food items and when they were eaten (e.g., Breakfast, Lunch).
    - Water_Intake (liters): Daily hydration levels.

- sugar_g, sodium_mg, cholesterol_mg: Micronutrient and additive tracking.
- cooking_method, prep_time_min, cook_time_min: Information on food preparation.

5. Subjective & Derived Data
   - rating: User-provided score (likely for the meal or workout quality).
   - Benefit: The specific health advantage of an exercise (e.g., "Improves posture").
   - Burns_Calories_Bin: A categorical classification of calorie burn (e.g., Low, Medium, High).

## 5.1 Data Understanding

The understanding.py module performs an automated audit of the dataset to define the project scope and identify data integrity issues before model training.

1. Schema Inference (Inventory Check)
   - Action: Identifies the data type of every column in the data frame.
   - Logic: Uses Pandas dtype attributes.
     - Numeric: int or float.
     - Categorical: object, string, or bool.
     - Convertible Text: String columns containing only numeric characters are flagged for type conversion.
2. Target & Task Inference (Defining the Mission)
   - Action: Determines if the machine learning problem is Classification or Regression.
   - Logic: Analyzes the Target Column unique value count ($n_{unique}$).
     - Classification: Target has low cardinality (few unique values) or is non-numeric.
     - Regression: Target is numeric with high cardinality.
3. Class Imbalance (Checking the Terrain)
   - Action: Detects if classification categories are disproportionately represented.
   - Logic: Calculates frequency using value_counts(normalize=True).
     - Flag: Triggered if the majority class exceeds a set threshold (e.g., >90%).
4. Skewness (Measuring the Slant)
   - Action: Measures the asymmetry of numeric feature distributions.
   - Logic: Calculates the Fisher-Pearson coefficient of skewness.
     - Result: A value of 0 indicates a normal distribution. High positive or negative values suggest the need for Log Transformation to normalize the data.
5. Leakage Detection (Spotting the Mole)
   - Action: Identifies "cheater" variables that contain information about the target that wouldn't be available in production.

- Logic: Computes the Correlation Coefficient (Pearson/Spearman) between features and the target.
    - Flag: Features with a correlation $> 0.95$ are flagged as Suspicious and usually removed to prevent overfitting.

Preview

| | Age | Gender | Weight (kg) | Height (m) | Max_BPM | Avg_BPM | Resting_BPM | Session_Duration (hours) | Calories_Burned | Workout_Type | Fat_Percentage | Water_Intake (liters) | Worko |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 34.91 | Male | 65.27 | 1.62 | 188.58 | 157.65 | 69.05 | 1 | 1080.9 | Strength | 26.8004 | 1.5 | |
| 1 | 23.37 | Female | 56.41 | 1.55 | 179.43 | 131.75 | 73.18 | 1.37 | 1809.91 | HIIT | 27.655 | 1.9 | |
| 2 | 33.2 | Female | 58.98 | 1.67 | 175.04 | 123.95 | 54.96 | 0.91 | 802.26 | Cardio | 24.3208 | 1.88 | |
| 3 | 38.69 | Female | 93.78 | 1.7 | 191.21 | 155.1 | 50.07 | 1.1 | 1450.79 | HIIT | 32.8136 | 2.5 | |
| 4 | 45.09 | Male | 52.42 | 1.88 | 193.58 | 152.88 | 70.84 | 1.08 | 1166.4 | Strength | 17.3073 | 2.91 | |

## Target & Task


```
{
  "target" : "Burns_Calories_Bin"
  "target_reason" : "Heuristic: last non-ID column with >1 unique value."
  "task" : "classification"
  "task_reason" : "Non-numeric target."
  "target_dtype" : "object"
  "target_cardinality" : 4
}
```


## Schema


```
{
  "numeric" : [...]
  "categorical" : [
    0 : "Gender"
    1 : "Workout_Type"
    2 : "meal_name"
    3 : "meal_type"
    4 : "diet_type"
    5 : "cooking_method"
    6 : "Name of Exercise"
    7 : "Benefit"
    8 : "Target Muscle Group"
    9 : "Equipment Needed"
    10 : "Difficulty Level"
    11 : "Body Part"
    12 : "Type of Muscle"
    13 : "Workout"
    14 : "Burns_Calories_Bin"
  ]
  "datetime" : []
  "boolean" : []
  "convertible_text_numeric" : []
}
```


## Imbalance


```
{
  "is_imbalanced" : false
  "ratio" : 0.24985
  "class_counts" : {
    "Low" : 5003
    "High" : 5000
    "Very High" : 5000
    "Medium" : 4997
  }
  "threshold" : 0.2
}
```


## Potential Leakage


```
{
  "method" : "mutual_info_classif top 5"
  "suspicious_features" : [
    0 : "Burns Calories (per 30 min)"
    1 : "Burns Calories (per 30 min)_bc"
    2 : "expected_burn"
    3 : "cal_from_macros"
    4 : "BMI_calc"
  ]
}
```

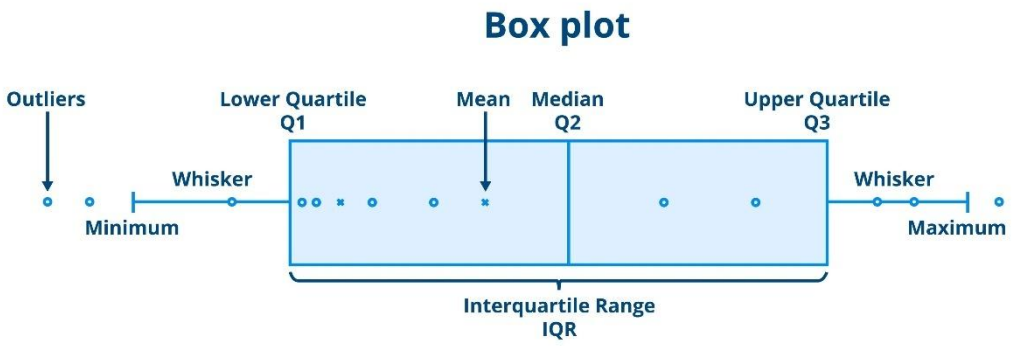
## 5.2 Data Diagnosis

The diagnosis.py module performs a technical "physical exam" on the dataset to detect anomalies, redundancies, and statistical outliers that could degrade model performance.

1. Missing Value Analysis (The X-Ray)
    - Action: Identifies "holes" or null entries within the dataset.
    - Logic: Scans every column for NaN or None values.
    - Metric: Calculated as a percentage of the total rows:

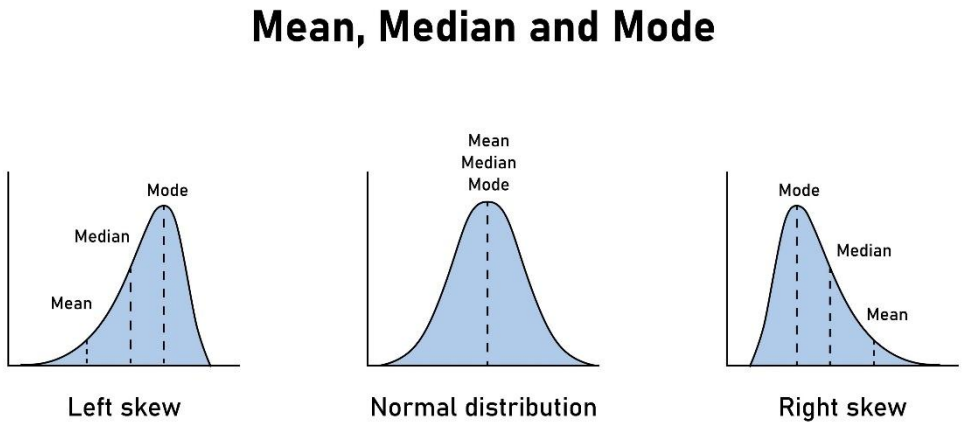
$$\text{Missing } \% = \left( \frac{\text{Count of Nulls}}{\text{Total Rows}} \right) \times 100$$

2. Duplicate Detection (The Echo Test)
    - Action: Finds identical rows that may cause the model to over-represent specific patterns (bias).
    - Logic: Uses df.duplicated() to compare row signatures across all features.
    - Metric: Reports the absolute count and percentage of redundant records.
3. Outlier Detection (The Fever Check)
    - Action: Flags extreme values that deviate significantly from the norm.
    - Logic: Employs the Interquartile Range (IQR) method to define the "Safe Zone."
        - IQR Calculation: IQR = Q3 - Q1
        - Lower Bound: Q1 - 1.5 \times IQR
        - Upper Bound: Q3 + 1.5 \times IQR
    - Result: Any value outside these bounds is flagged as an outlier.

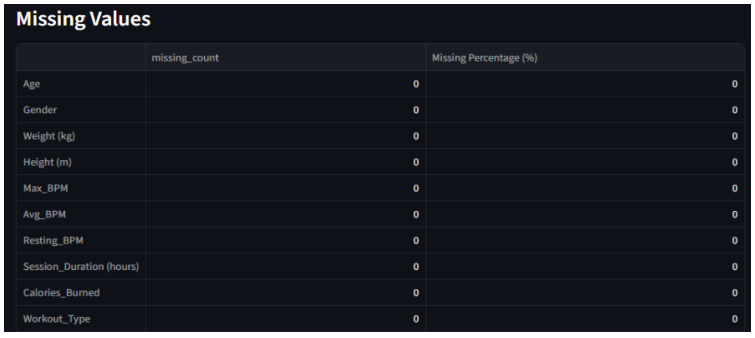


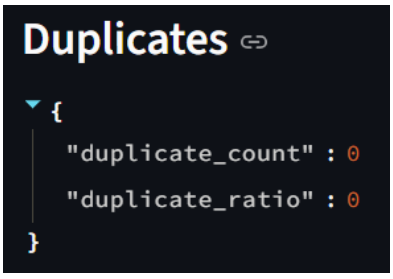

4. Numeric Correlation (The Pulse Check)
    - Action: Identifies relationships between numeric variables to detect Multicollinearity (redundancy).
    - Logic: Calculates the Pearson Correlation Coefficient (r), ranging from -1.0 to 1.0.
    - Threshold: Feature pairs with $|r| > 0.8$ are flagged as "Hotspots," suggesting one can likely be removed without losing information.
5. Categorical Association (Pattern Recognition)
    - Action: Measures the link between non-numeric features (e.g., "Gender" and "Purchase History").

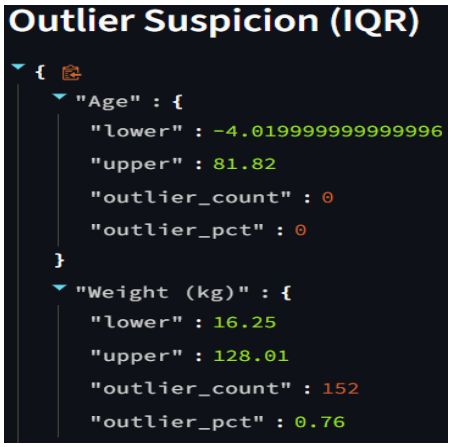
- Logic: Uses Cramer's V, a statistical measure based on the Chi-Square test.
    - Scale: 0 (no association) to 1(perfect association).
- Constraint: To optimize performance, columns with High Cardinality (e.g., Unique IDs, hashes) are excluded from this check.

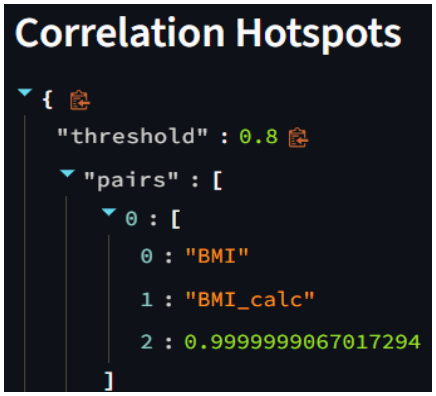



- Logic: Replaces values outside the IQR Safe Zone with the boundary values themselves:
    - If Value > (Q3 + 1.5 \times IQR), it is capped at the Upper Limit.
    - If Value < (Q1 - 1.5 \times IQR), it is floored at the Lower Limit.

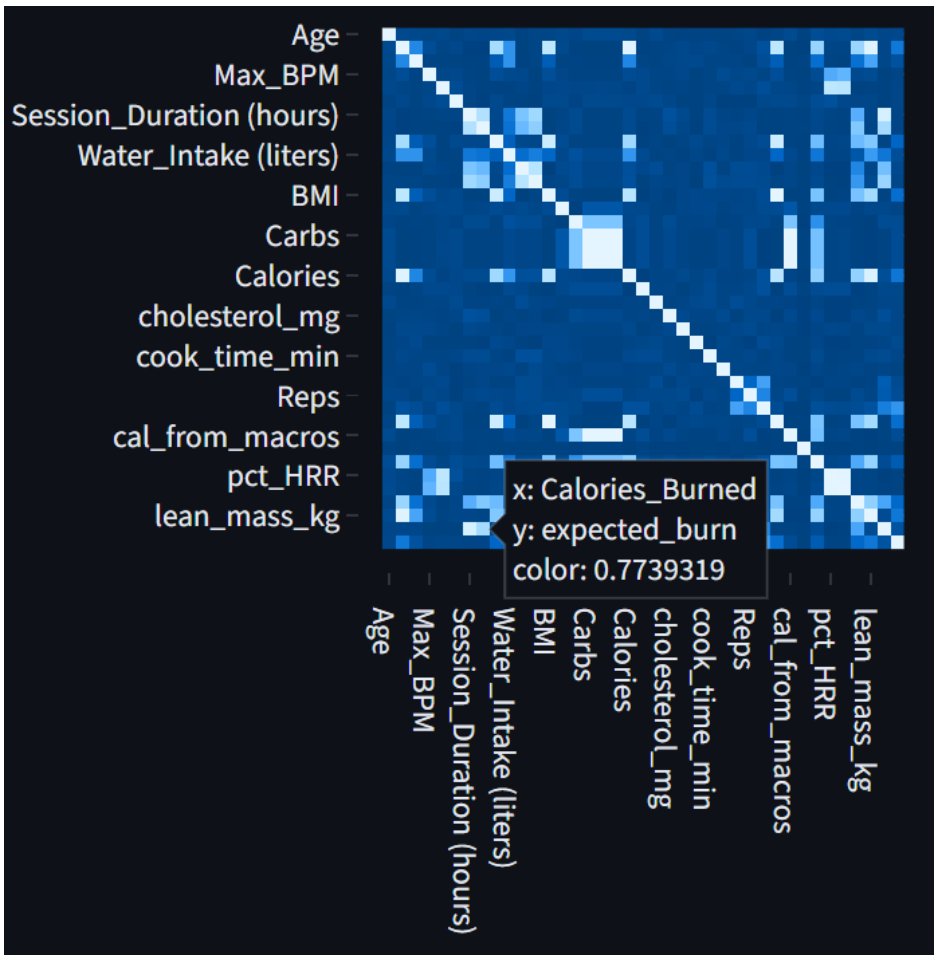


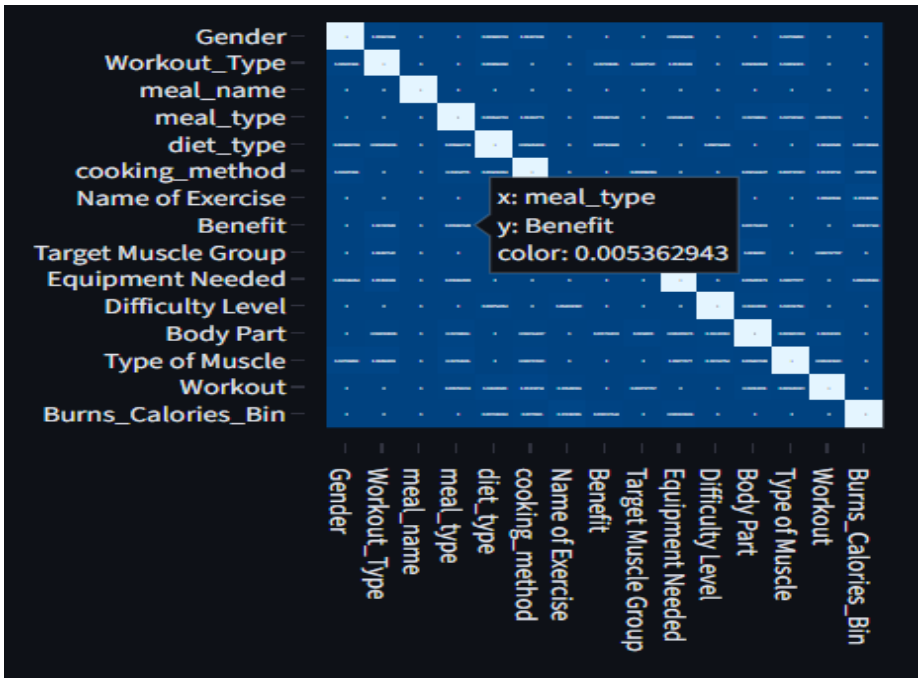



## Correlation Matrix (abs)



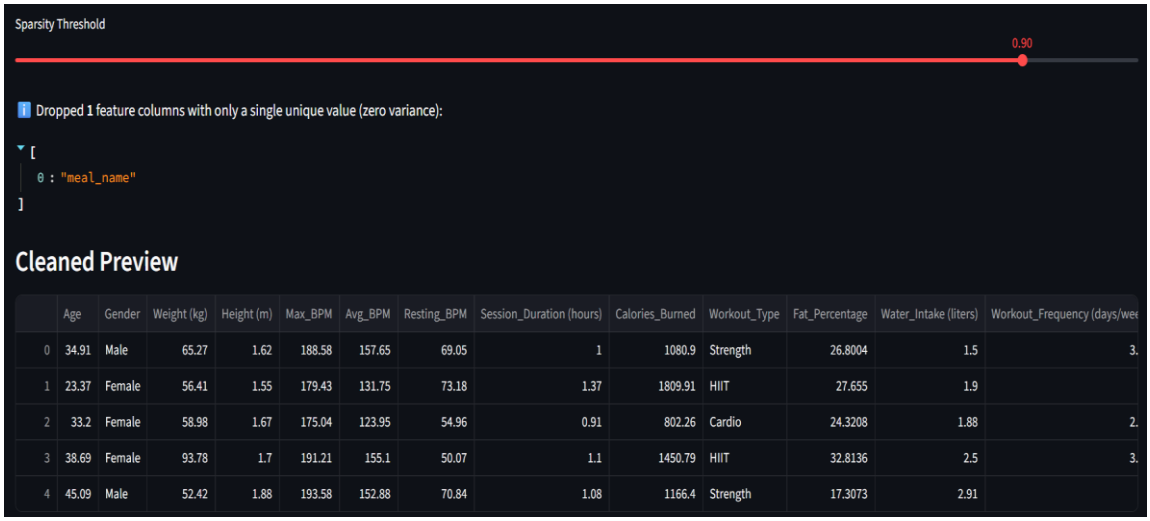

## Categorical Correlation (Cramer's V)



## 5.3 Data Cleaning

The cleaning.py module executes the remediation plan developed during the diagnosis phase. It prepares the data for modelling by removing noise, handling missingness, and stabilizing extreme values.

1. Feature Selection (Dropping Columns)
    - Action: Removes features that provide zero predictive value or are statistically unusable.
    - Logic:
        - Sparsity Check: Columns missing more than a specific threshold (e.g., >99%) are discarded as they lack enough information to form patterns.
        - Zero Variance Check: Columns with only one unique value are removed because they offer no contrast for the model to learn from.
2. Redundancy Removal (Dropping Duplicates)
    - Action: Deletes exact row replicas to prevent the model from over-weighting specific observations.
    - Logic: Uses drop_duplicates() to retain the first instance of a record and purge all subsequent identical matches.
3. Imputation (Patching the Holes)
    - Action: Fills missing data points using statistically sound substitutes.
    - Logic:
        - Numeric Data: Uses the Median. The median is preferred over the mean because it is "robust," meaning it isn't pulled away by extreme outliers.
        - Categorical Data: Replaces nulls with a new string label: "Unknown". This treats "missingness" as a distinct feature rather than guessing a value.
4. Winsorization (Outlier Capping)
    - Action: Constrains extreme values to a defined statistical boundary to prevent them from distorting the model's coefficients




| | Age | Gender | Weight (kg) | Height (m) | Max_BPM | Avg_BPM | Resting_BPM | Session_Duration (hours) | Calories_Burned | Workout_Type | Fat_Percentage | Water_Intake (liters) | Workout_Frequency (days/wee |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 34.91 | Male | 65.27 | 1.62 | 188.58 | 157.65 | 69.05 | 1 | 1080.9 | Strength | 26.8004 | 1.5 | 3. |
| 1 | 23.37 | Female | 56.41 | 1.55 | 179.43 | 131.75 | 73.18 | 1.37 | 1809.91 | HIIT | 27.655 | 1.9 | |
| 2 | 33.2 | Female | 58.98 | 1.67 | 175.04 | 123.95 | 54.96 | 0.91 | 802.26 | Cardio | 24.3208 | 1.88 | 2. |
| 3 | 38.69 | Female | 93.78 | 1.7 | 191.21 | 155.1 | 50.07 | 1.1 | 1450.79 | HIIT | 32.8136 | 2.5 | 3. |
| 4 | 45.09 | Male | 52.42 | 1.88 | 193.58 | 152.88 | 70.84 | 1.08 | 1166.4 | Strength | 17.3073 | 2.91 | |

## 5.4 Feature Engineering

The engineering.py module transforms raw, cleaned data into high-signal features. Its purpose is to expose hidden patterns and relationships, providing the model with more descriptive "clues" for prediction.

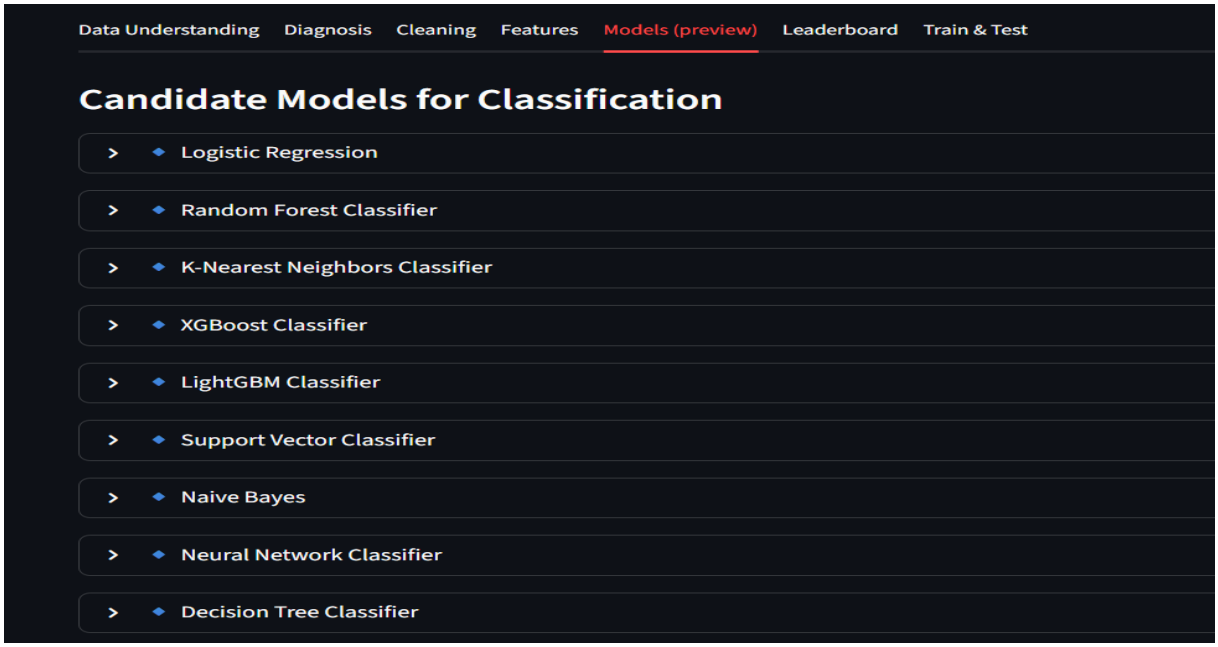
---

1. Date Extraction
    - Action: Deconstructs raw timestamps into cyclical and seasonal components that models can interpret numerically.
    - Logic: Uses Pandas dt accessors to "explode" a single date column into multiple features:
        - Year: Identifies long-term growth or decline.
        - Month: Captures seasonal cycles (e.g., summer vs. winter behaviour).
        - Day of Week: Captures weekly rhythms (e.g., weekday vs. weekend spikes).
        - Is Weekend: A binary flag (1or 0) to simplify weekend-specific logic.
2. Interaction Hashing
    - Action: Combines two categorical features to capture "synergy" without causing a massive explosion in data dimensions.
    - Logic: Employs the Hashing Trick to handle high-cardinality interactions:
        - Concatenation: Merges two strings (e.g., "Red" + "Sedan" = "Red_Sedan").
        - Hash Function: Converts the merged string into a large integer.
        - Modulo Operation: Compresses that integer into a fixed range (e.g., 0 to 511).
    - Benefit: Represents complex feature intersections (e.g., Colour * Model) in a fixed-width format, avoiding the Curse of Dimensionality.
3. Feature Selection (The Critic)
    - Action: Audits all existing and newly created features to remove "noise" and retain only the most predictive variables.
    - Logic: Ranks features based on statistical relevance to the Target:
        - Classification: Uses Chi-Square or ANOVA tests to see if feature distributions differ significantly across classes.
        - Regression: Uses Mutual Information or Correlation to measure the strength of the relationship between feature and target.
    - Selection: Retains only the Top K highest-scoring features, significantly reducing training time and preventing overfitting.

## 5.5 Model Preview

The models.py module evaluates the dataset's dimensions and objectives to shortlist the most efficient and accurate algorithms for the training phase.

1. Dimension Analysis (Reading the Script)
    - Action: Measures the scale and complexity of the dataset to filter out incompatible models.
    - Logic: Analyzes the shape of the processed dataframe.
        - Sample Count (n): High row counts trigger the exclusion of computationally expensive models like K-Nearest Neighbors (KNN).
        - Feature Count (p): High column counts suggest models with built-in regularization to prevent overfitting.
        - Categorical Density: Determines if the selected models require encoding or can handle text-based categories natively.
2. Task Identification (Defining the Genre)
    - Action: Filters the algorithm library based on the target variable's nature.
    - Logic: Uses the task variable identified by the Surveyor.
        - Classification: Restricted to "Classifiers" (e.g., Logistic Regression, Random Forest) for predicting discrete categories.
        - Regression: Restricted to "Regressors" (e.g., Linear Regression, XGBoost) for predicting continuous numerical quantities.
3. Model Shortlisting (The Audition)
    - Action: Curates a "Squad" of candidate algorithms tailored to the data profile via the get_available_models() function.
    - Logic: * Baseline: Always includes a Decision Tree for speed and as a performance benchmark.
        - Ensemble Specialists: Selects Random Forest or XGBoost for complex data requiring high accuracy.
        - Interpretability Specialists: Selects Linear/Logistic Regression when explainability is prioritized.
        - Deep Learning: Includes Neural Networks only if the row count is sufficient to avoid training failure.
4. Candidate Mapping (The Call Sheet)
    - Action: Translates technical shorthand into human-readable report names.
    - Logic: Performs a dictionary lookup using MODEL_DISPLAY_NAMES.
    - Result: Converts internal keys (e.g., rf, xgb) into clear titles like "Random Forest Classifier" or "XGBoost Regressor.


Data Understanding
Diagnosis
Cleaning
Features
Models (preview)
Leaderboard
Train & Test
Candidate Models for Classification
Logistic Regression
Random Forest Classifier
K-Nearest Neighbors Classifier
XGBoost Classifier
LightGBM Classifier
Support Vector Classifier
Naive Bayes
Neural Network Classifier
Decision Tree Classifier


K-Nearest Neighbors Classifier
Estimator: Pipeline
{
"memory" : NULL
"steps" : [
0 : [
0 : "standardscaler"
1 : "StandardScaler()"
]
1 : [
0 : "kneighborsclassifier"
1 : "KNeighborsClassifier()"
]
]
"transform_input" : NULL
"verbose" : false
"standardscaler" : "StandardScaler()"
"kneighborsclassifier" : "KNeighborsClassifier()"
"standardscaler__copy" : true
"standardscaler__with_mean" : true
"standardscaler__with_std" : true
"kneighborsclassifier__algorithm" : "auto"
"kneighborsclassifier__leaf_size" : 30
"kneighborsclassifier__metric" : "minkowski"
"kneighborsclassifier__metric_params" : NULL
"kneighborsclassifier__n_jobs" : NULL
"kneighborsclassifier__n_neighbors" : 5
"kneighborsclassifier__p" : 2
"kneighborsclassifier__weights" : "uniform"
}

## 5.6 Model Leaderboard

The leaderboard.py module executes a head-to-head competition between the shortlisted algorithms. It uses rigorous statistical validation to crown a "Champion" model for deployment.

1. The Competitors (Algorithm Styles)

Models are selected based on their specific mathematical strengths:

- Linear & Logistic Regression: Fast, simple models that establish a baseline using straight-line relationships.
- Decision Trees: Uses hierarchical "If-Then" logic to capture non-linear patterns.
- Random Forest: An ensemble of hundreds of trees that uses "wisdom of the crowd" to reduce errors.
- XGBoost & LightGBM: High-performance gradient boosting models that learn sequentially from previous errors.
- K-Nearest Neighbors (KNN): Makes predictions by finding the most similar historical data points.
- Support Vector Machines (SVM): Finds the optimal boundary (hyperplane) with the widest possible margin between classes.
- Neural Networks (MLP): Uses interconnected layers to detect high-level abstract patterns.

2. The Arena (3-Fold Cross-Validation)

To ensure results are not due to luck, models undergo K-Fold Cross-Validation.

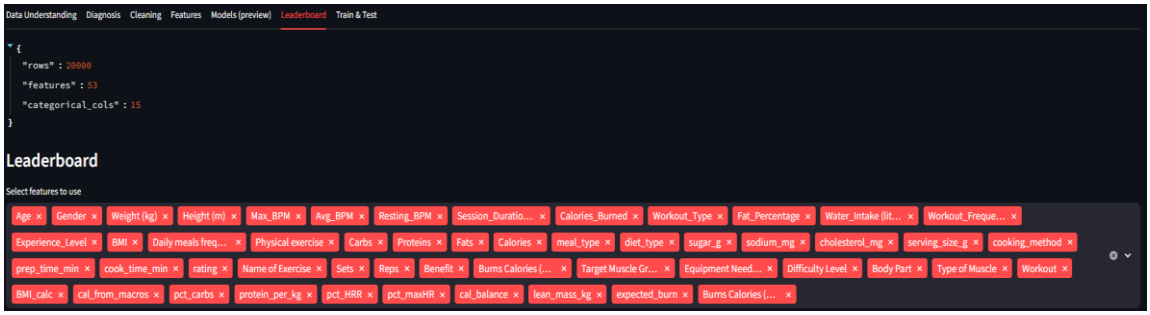
- Logic: The data is split into three equal segments (folds).
- Process: The model trains on two folds and is tested on the third. This is repeated three times so every data point is used for both training and testing.
- Metric: The "Leaderboard Score" is the average performance across all three rounds, ensuring the model is robust and generalizes well.

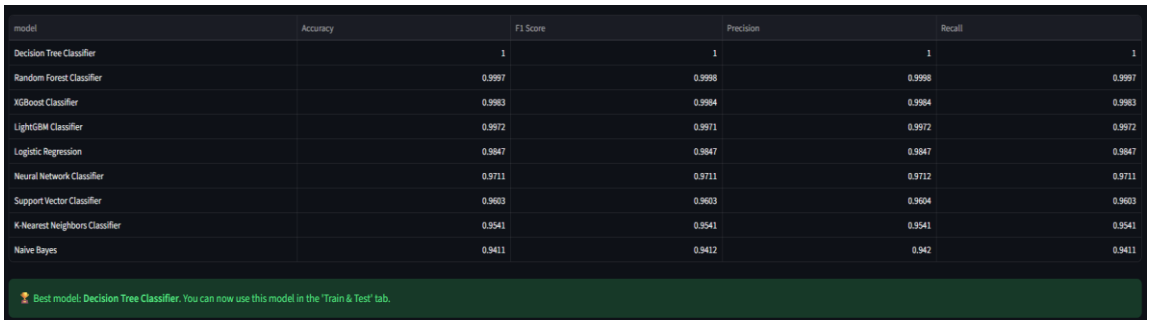
3. The Scoreboard (Performance Metrics)

Success is measured differently depending on the project type:

| Task Type | Primary Metric | Description |
|---|---|---|
| Classification | Accuracy | Percentage of total correct predictions. |
| | F1-Score | Harmonic mean of Precision and Recall; vital for imbalanced data. |
| | Precision/Recall | Measures "Exactness" vs. "Completeness" of findings. |
| Regression | RMSE | Root Mean Squared Error; punishes large errors heavily. |

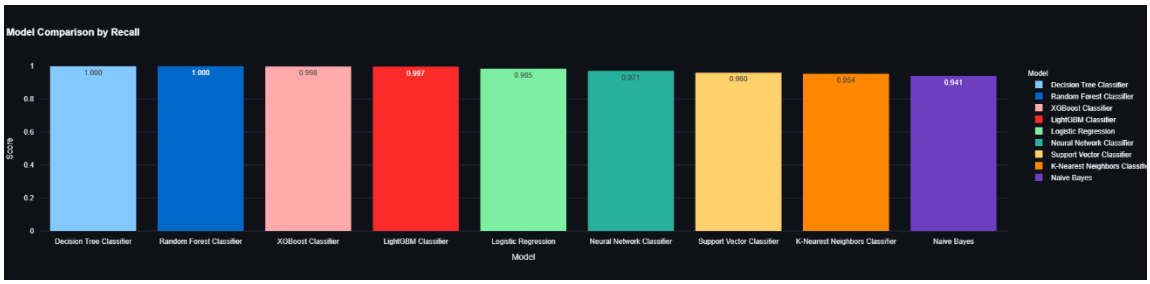
| Task Type | Primary Metric | Description |
|---|---|---|
| | MAE | Mean Absolute Error; average distance between prediction and truth. |
| | R^2 Score | The "Goodness of Fit"; measures what % of variance the model explains. |

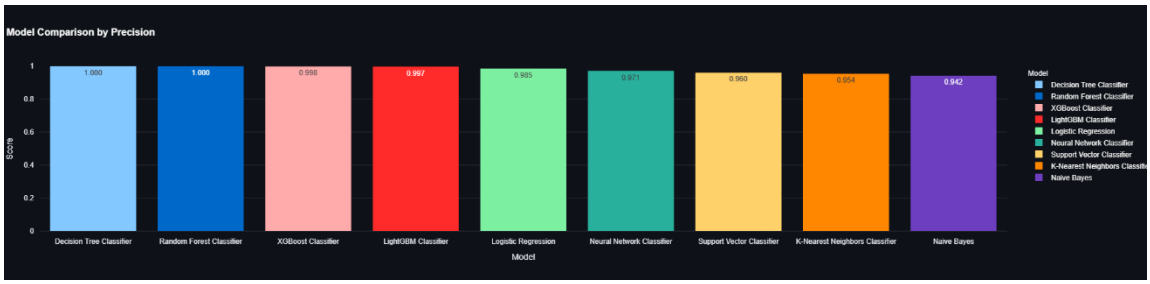


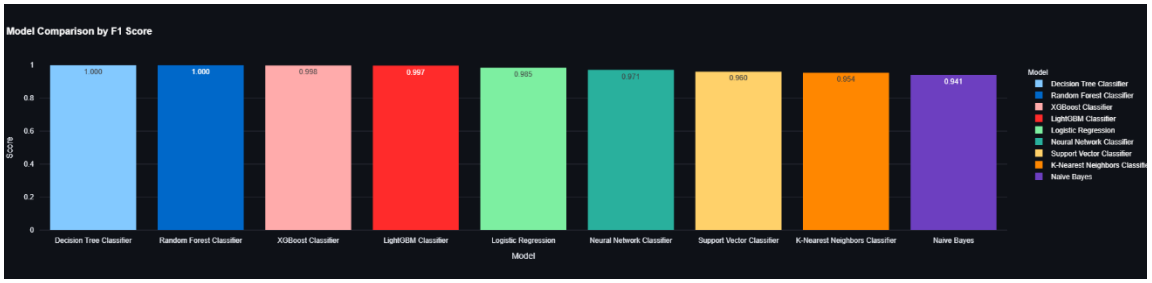



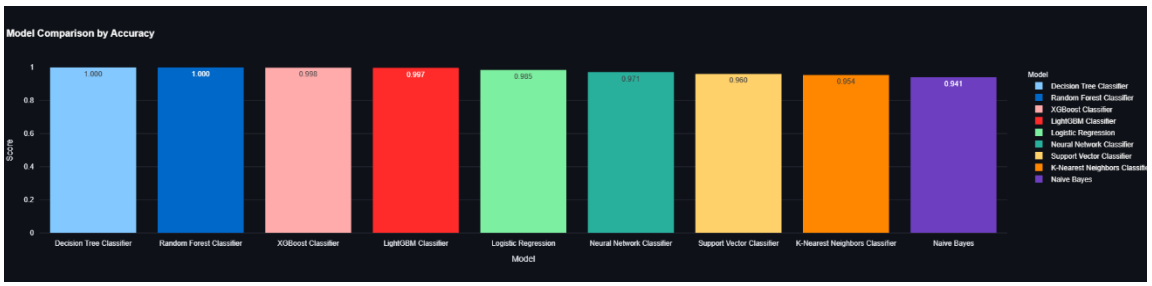
| model | Accuracy | F1 Score | Precision | Recall |
|---|---|---|---|---|
| Decision Tree Classifier | 1 | 1 | 1 | 1 |
| Random Forest Classifier | 0.9997 | 0.9998 | 0.9998 | 0.9997 |
| XGBoost Classifier | 0.9983 | 0.9984 | 0.9984 | 0.9983 |
| LightGBM Classifier | 0.9972 | 0.9971 | 0.9972 | 0.9972 |
| Logistic Regression | 0.9847 | 0.9847 | 0.9847 | 0.9847 |
| Neural Network Classifier | 0.9711 | 0.9711 | 0.9712 | 0.9711 |
| Support Vector Classifier | 0.9603 | 0.9603 | 0.9604 | 0.9603 |
| K-Nearest Neighbors Classifier | 0.9541 | 0.9541 | 0.9541 | 0.9541 |
| Naive Bayes | 0.9411 | 0.9412 | 0.942 | 0.9411 |

Best model: **Decision Tree Classifier**. You can now use this model in the 'Train & Test' tab.





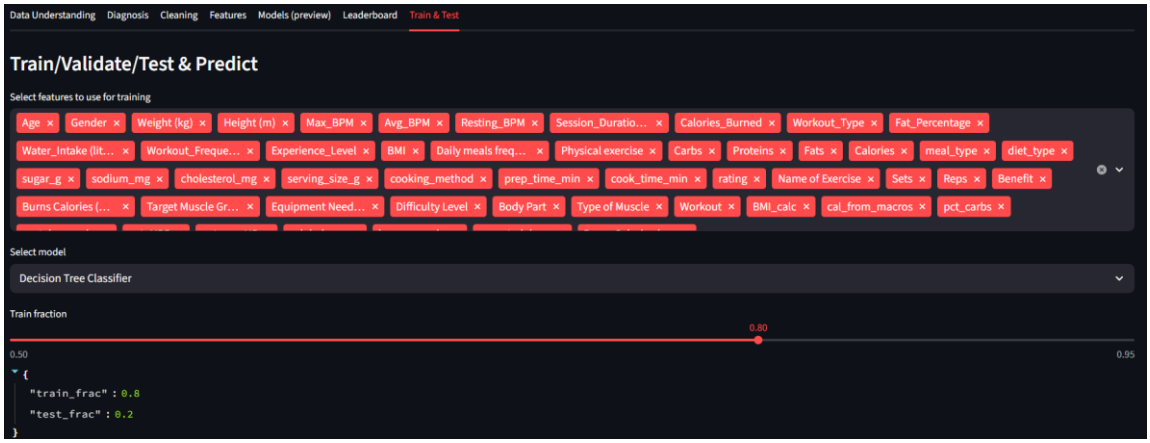

## 5.7 Train & Test

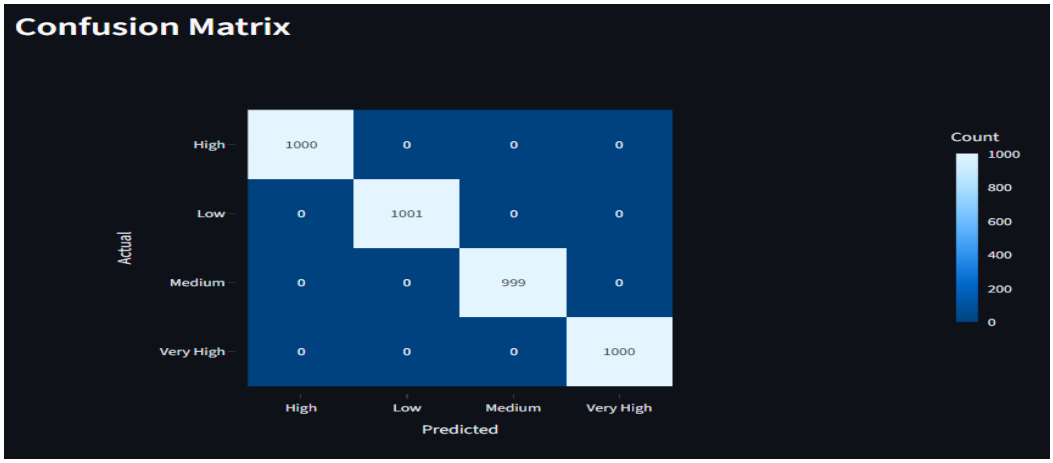
The train_test.py module conducts the final evaluation of the selected champion model. It simulates real-world deployment by training the model on historical data and testing it against a "hidden" exam set.

1. Feature Selection (Setting the Curriculum)
    - Action: Finalizes which variables the model is permitted to use for learning.
    - Logic: The user filters the feature_cols. This step ensures that "distractions" (irrelevant noise) or data not available at the time of prediction are excluded.
2. Data Partitioning (Train/Test Split)
    - Action: Divides the dataset into two strictly separate sets to prevent "cheating" (data leakage).
    - Logic: Uses a standard split ratio (e.g., 80/20).
        - Training Set (80%): The "Study Guide" where the model sees both features and targets to learn patterns.
        - Test Set (20%): The "Final Exam" where targets are hidden; the model must predict based on features alone.
    - Stratification: For classification, the split ensures that the ratio of classes (e.g., % of "Yes" vs. "No") remains identical in both sets to maintain statistical consistency.
3. Model Training (The Study Session)
    - Action: Executes the mathematical "fitting" of the algorithm.
    - Logic: Calls the .fit(X_train, y_train) method. The algorithm adjusts its internal parameters—such as tree branches, weights, or thresholds—to minimize error on the training data.
4. Inference (Taking the Test)
    - Action: Generates predictions on the unseen test data.

- Logic: Calls .predict(X_test). The model applies the rules it learned during training to generate a list of predicted values (y_pred).

5. Grading (Metrics Calculation)
   - Action: Compares the model's guesses (y_pred) against the actual ground truth (y_test).
   - Metrics:
     - Classification: Generates a Confusion Matrix to identify specific errors (False Positives vs. False Negatives).
     - Regression: Calculates Residuals (Actual - Predicted) to determine the average margin of error (RMSE/MAE).
6. Results Visualization (The Report Card)
   - Action: Provides a graphical representation of the model's logic and performance.
   - Visuals:
     - Confusion Matrix Heatmap: Highlights where the model is most likely to get confused.
     - Decision Tree Plot: (For Tree models) Visualizes the actual "If-Then" flowchart logic.
     - Actual vs. Predicted Table: A side-by-side comparison of the first few rows to allow for manual spot-checking.



Sample predictions (test split):

| | y_true | y_pred |
|---|---|---|
| 0 | Very High | Very High |
| 1 | High | High |
| 2 | Medium | Medium |
| 3 | High | High |
| 4 | Low | Low |
| 5 | Very High | Very High |
| 6 | High | High |
| 7 | High | High |
| 8 | High | High |
| 9 | Medium | Medium |

## 6. Conclusion: The Final Summary

The Automated Insight Engine serves as a digital "General Contractor" for data science, managing the end-to-end construction of machine learning models through a transparent and rigorous pipeline.

1. The Journey Summary

- Understanding: Maps the terrain and identifies if the task is Classification or Regression.
- Diagnosis: Performs a "health check" to flag missing values, duplicates, and outliers.
- Cleaning/Engineering: Repairs structural data issues and crafts new features (e.g., date extraction and hashing).
- Leaderboard: Competes multiple algorithms (Linear, Trees, Boosting) to find the top performer.
- Train & Test: Validates the "Champion" on unseen data to ensure real-world readiness.

2. Core Value Proposition

- Speed: Reduces days of manual boilerplate coding into minutes of automated execution.
- Transparency: Replaces "Black Box" AI with visual heatmaps, correlation matrices, and logic trees.
- Reliability: Standardizes best practices like Stratified Splitting and Cross-Validation to prevent human error.

## 7. Technical Challenges & Solutions

Navigating real-world data required solving several "edge-case" hurdles to ensure application stability:

| Challenge | Impact | Technical Solution |
|---|---|---|
| Integer Ambiguity | House prices (integers) mistakenly flagged as categories. | Cardinality Check: If unique values > 50 and represent > 25% of data, force Regression. |
| Memory Overload | High-cardinality columns (e.g., User IDs) crashing RAM. | Limit Enforcement: Columns with > 1,000 unique categories are excluded from heavy calculations. |
| "Zombie" State | Residual data from a previous upload causing KeyErrors. | State Reset: Implemented a unique file_id check to wipe st.session_state upon new uploads. |
| Encoding Failures | Non-UTF-8 files (Excel exports) failing to load. | Encoding Fallback: Wrapped pd.read_csv in a try-except to retry with latin1 if UTF-8 fails. |
| Silent Target Drop | Cleaning scripts accidentally removing the target column. | Guardrail: Added an explicit check to prevent the removal of the Target, even if it has zero variance. |

## 8. Future Scope: From Workshop to Factory

The roadmap for the next iteration focuses on moving from model discovery to production-grade deployment:

- Hyperparameter Tuning: Integrating Optuna or GridSearchCV to move beyond default settings and find the mathematical "sweet spot."
- Explainable AI (XAI): Implementing SHAP or LIME to explain *why* a specific prediction was made (e.g., "Risk is high because 'Last Login' > 30 days").
- Ethical AI: Adding a Fairness Audit to detect disparate impact or historical bias against specific demographics.
- Live Connectors: Moving beyond CSVs to direct integrations with SQL, Snowflake, or BigQuery.
- MLOps & Monitoring: Tracking Data Drift to alert users when a model's performance begins to "rot" due to changing real-world patterns.
- Unstructured Data: Expanding into NLP (Sentiment Analysis) and Computer Vision using BERT or Word Embeddings.