

Aim:

1. From the given x (independent variable) space, which is limited data (relative to experiment), explore the x space.
2. Explore the “blank space”

Objectives:

1. To infer the underlying distribution of a given set of samples.
2. To build a model that estimates the function approximating the relationship between the input variables and the dependent variable.
3. To utilize the trained model to extract insights and guide exploration of the input space.

Methodology:

Given the limited availability of data relative to the potential complexity of the experiment, two general strategies are proposed to explore and augment the input space:

1. **Bayesian Optimization-Based Sampling ([LINK](#))**
This approach uses an acquisition function to select new data points for experimentation or simulation. It is especially suitable when running experiments is relatively inexpensive.
2. **Model-Based Distribution Learning([LINK](#))**
In this method, a generative model is trained to approximate the distribution of the existing data, enabling the generation of synthetic data through optimization. This is particularly useful when experimentation is costly.
 - a.
3. **Non-Training Oversampling Technique (Imbalance Handling)**
 - a. When computational resources are limited or class imbalance is significant, oversampling methods like **SMOTE**, **GSMOTENC**, or **CSBBoost** can be used to synthetically balance the dataset without training a generative model. These methods introduce **diverse synthetic samples** through interpolation or extrapolation, helping to correct skewed label distributions efficiently. They are lightweight,

effective, and particularly useful in **classification tasks** where imbalance hampers model performance

4. **Simplest approach :**

This is most resource efficient and simple.

Overview

In this work, the goal is to build a model that captures the underlying relationship within a dataset – essentially fitting the function that governs the data – using a minimal number of observations. This trained model can then be used to explore the relationship between dependent and independent variables, or to "explore the blank space" using the SHAP (SHapley Additive exPlanations) library for interpretability.

Model performance is evaluated using a holdout test set. However, a validation set with k-fold cross-validation can also be employed to assess the model's robustness and generalization ability.

These strategies are detailed below.

Method 1: Bayesian Optimization for Data Augmentation

Objective: Use Bayesian Optimization to efficiently guide laboratory experiments by selecting optimal x points in the input space, instructing lab personnel to evaluate these points to obtain true y values, and augmenting the dataset to improve a surrogate model.

Context: When acquiring new data through physical experiments or simulations is feasible but costly, BO helps prioritize which x points to evaluate. The process iteratively trains the model, smartly suggests the points for which label to be collected from the lab, and then uses it to refine the surrogate model.

Step 1: Surrogate Modeling

- **Purpose:** Build a model to predict $y=f(x)$ and quantify uncertainty based on the current dataset.
 - **Process:**
 - Train a surrogate model on the existing dataset
 - Common choices:
 - **Gaussian Process (GP):** Ideal for smooth functions, low-to-moderate dimensionality, and noisy data.
 - **Neural Network:** Suitable for high-dimensional or complex data.
 - **Random Forest:** Robust to noise and non-smooth functions.
 - Select the model based on:
 - Function smoothness (e.g., GP for smooth, RF for non-smooth).
 - Noise level (e.g., GP with noise kernel for noisy data).
 - Input dimensionality (e.g., NN for high-dimensional $x \times x$).
 - Domain knowledge (e.g., physics-based priors for scientific experiments).
 - **Output:** A surrogate model providing:
 - Predicted mean $\mu(x)$: Expected y at x .
 - Predicted standard deviation $\sigma(x)$: Uncertainty at x .
-

Step 2: Acquisition Function

- **Purpose:** Identify x points that are most valuable for lab evaluation, balancing exploration (high uncertainty) and exploitation (high predicted y).
- **Process:**
 - Use the **Upper Confidence Bound (UCB)** acquisition function:

$$UCB(x) = \mu(x) + K \cdot \sigma(x)$$
 - Where:
 - $\mu(x)$: Predicted mean from the surrogate.
 - $\sigma(x)$: Predicted standard deviation (uncertainty).
 - K : Hyperparameter controlling exploration vs. exploitation.
 - Higher K values prioritize exploration (uncertain regions), while lower values favor exploitation (regions with high predicted y).
 - Alternative acquisition functions (if applicable):
 - **Expected Improvement (EI):** Focuses on expected gain over the current best y .
 - **Probability of Improvement (PI):** Prioritizes points likely to exceed the current best.
- **Output:** A score $UCB(x)$ for each candidate x , indicating its priority for evaluation.

Step 3: Point Selection Strategy

- **Purpose:** Select specific x points to send to the laboratory for experimental evaluation.
- **Process:**
 - **Optimization-Based Search** (Recommended):
 - Formulate the selection as an optimization problem:

$$x^* = \arg \max_x \text{UCB}(x)$$
 - Use gradient-free optimization (e.g., Bayesian optimization libraries like scikit-optimize or GPyOpt) or gradient-based methods (if the surrogate is differentiable, e.g., neural networks).
 - This is efficient for high-dimensional or complex spaces and directly identifies the most promising x .
 - **Alternative Strategies** (if optimization is infeasible):
 - **Grid Search:**
 - Discretize the input space into a grid.
 - Evaluate $\text{UCB}(x)$ at each grid point and select those with the highest scores.
 - Suitable for low-dimensional spaces but scales poorly.
 - **Threshold-Based Filtering** (Optional):
 - Only select x points where $\text{UCB}(x) > \text{threshold}$ (e.g., 0.9) to ensure high-value points.
 - Adjust the threshold based on lab constraints (e.g., number of experiments feasible).
- **Output:** A small set of x points (e.g., 1–10, depending on lab capacity) to be evaluated.

Step 4: Laboratory Evaluation

- **Purpose:** Obtain true y values for the selected x points through experiments.
- **Process:**

- Provide the selected x points to laboratory personnel with clear instructions (e.g., specific input conditions like temperature, pressure, or concentrations).
 - Lab personnel conduct experiments or simulations to measure the corresponding y values (e.g., reaction yield, material strength).
 - Collect the (x,y) pairs from the lab.
 - **Output:** New (x,y) pairs to augment the dataset.
-

Step 5: Dataset Augmentation and Model Retraining

- **Purpose:** Incorporate new data to improve the surrogate model.
 - **Process:**
 - Add the new (x,y) pairs to the existing dataset.
 - Retrain the surrogate model on the expanded dataset.
 - Evaluate the updated model on a holdout test set to monitor performance (e.g., RMSE, R^2).
 - **Output:** An improved surrogate model with better predictive accuracy and reduced uncertainty in targeted regions.
-

Step 6: Iteration and Convergence

- **Purpose:** Repeat the process to progressively refine the model until it meets performance goals or resources are exhausted.
- **Process:**
 - Iterate Steps 2–5, selecting new x points, obtaining y values, and retraining the model.
 - Stop when one of the following convergence criteria is met:
 - **Performance Threshold:** The surrogate's performance on the test set exceeds a predefined metric (e.g., $R^2 > 0.95$).
 - **Stagnation:** No significant improvement in performance over multiple iterations (e.g., $<1\%$ change in RMSE).
 - **Resource Limit:** Budget or lab capacity is exhausted (e.g., maximum number of experiments reached).
 - **UCB Convergence:** The maximum $UCB(x)$ falls below a threshold, indicating low potential for further improvement.
- **Output:** A final surrogate model optimized for the target function.

Method 2: Model-Based Distribution Learning

Objective

- The core idea behind model-based distribution learning is to efficiently augment a dataset without requiring additional costly experiments.
- We aim to *learn* the distribution of inputs from an existing dataset and then *generate* synthetic candidates that are both structurally valid and potentially high-performing, as determined by a surrogate model.

This process involves the following pipeline(details are discussed after pipeline):

1. Train a Generative Model

Train a GAN or VAE on known high-quality data to model the input distribution.

2. Select Sampling Strategy

Choose one of the following based on the problem setting:

- **Plan 1:** Gradient-Based Latent Optimization (for continuous, differentiable settings)
- **Plan 2:** Adaptive Grid Search (for non-differentiable or discrete settings)
- Both the strategies are explained in `sampling_strategy.txt`

3. Generate Candidates

Use the selected strategy to explore the latent space and generate diverse candidates:

- For **Plan 1:** Optimize latent vectors using gradient ascent on UCB
- For **Plan 2:** Evaluate UCB over an adaptively refined grid

4. **Memory-Based Filtering**

- Maintain a hash table H to store previously seen candidates
- Discard candidates that are too similar to any in H
- Retain only *novel, high-UCB* candidates

5. **Retrain Surrogate Model**

- Augment the training set with diverse, high-UCB synthetic candidates
- Retrain the surrogate to improve predictive accuracy

6. **Repeat**

- Periodically reapply the sampling strategy to continuously improve candidate diversity and model performance

Plan A: Using Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are powerful generative models consisting of two neural networks: a **Generator** $G(z)$ and a **Discriminator** $D(x)$, trained in a two-player minimax game. The **Generator** takes random noise $z \sim p_z(z)$ (usually from a normal distribution) and maps it to the data space, aiming to generate samples that resemble real data. The **Discriminator** tries to correctly classify inputs as real (from the dataset) or fake (from the generator).

Mathematically, GANs solve the following optimization problem:

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Here:

- $D(x)$: Probability that x is real
- $G(z)$: Generator output given noise z
- p_{data} : True data distribution
- p_z : Prior distribution over the latent space (e.g., Gaussian)

The **Discriminator** D is trained to maximize its ability to distinguish real vs. generated data, while the **Generator** G is trained to fool the Discriminator. When the model converges, $G(z)$ produces realistic data, and $D(x) \approx 0.5$ for both real and generated inputs, indicating it can no longer tell the difference.

1) Training:

- The GAN is trained on existing data such that the Generator learns to produce data samples that resemble the training distribution – not identical replicas, but structurally and statistically similar samples.
- Unlike probabilistic models such as VAEs, GANs do not assume an explicit likelihood function or probabilistic formulation of the data, which allows them to generate sharper and more realistic outputs.

The training objective is a minimax game:

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))]$$

2) Sampling and model training :

Data will be sampled in batches in which all the data points will have the value for acquisition function higher than threshold. The sampling strategy is explained in detail in `sampling_strategy.txt`

Then the model will be trained on the additional data. It is similar to a typical Bayesian optimization . it is just that an artificial data generator is kept in loop.

Each iteration saves the surrogate model and logs training loss on test data, enabling performance comparison across sampling iterations.

Limitation: GANs are prone to *mode collapse* and *instability*, especially in high-dimensional or sparse regimes.

Future Exploration: Conditional GANs (cGANs) offer a promising extension. In cGANs, the generator receives both a latent vector and a conditioning variable (e.g., age, gender, class label), enabling controlled data generation

This could allow for **targeted exploration** of the input space and incorporation of domain-specific constraints into the data generation process.

Plan B: Variational Autoencoders (VAEs)

When training instability or mode collapse in GANs limits performance, Variational Autoencoders (VAEs) offer a compelling and robust alternative for data generation in surrogate-guided optimization.

A VAE is a generative model based on probabilistic inference, composed of:

- Encoder $q\phi(z|x)$: Approximates the posterior distribution over latent variables z given input data x .
- Decoder $p\theta(x|z)$: Reconstructs the input data from latent variables z , sampled from a prior $p(z)$, typically $N(0,I)$.

The model is trained by maximizing the **Evidence Lower Bound (ELBO)**:

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q_{\phi}(z|x)} [\log p_{\theta}(x | z)] - \text{KL}(q_{\phi}(z | x) \| p(z))$$

- The **first term** promotes accurate reconstruction.
- The **second term** (KL-divergence) regularizes the latent space to be close to the prior, encouraging **smooth interpolation** and **diversity**.

Although VAEs tend to generate outputs with lower visual fidelity (slightly blurrier than GANs), they ensure structurally valid and diverse outputs. This makes them well-suited for surrogate-based optimization, where capturing the underlying structure and generalizability is more important than sharpness.

In the GAN Pipeline we will use VAE instead of GAN.

Method 3: Non-Training Oversampling Technique

In many real-world scenarios, the performance degradation of machine learning models is often rooted in the **imbalanced distribution of labels or classes**. For classification tasks, such imbalance leads to models that are biased toward the majority class, failing to generalize well to minority classes which are typically of higher importance.

To address this, several **resampling strategies** have been developed, broadly categorized as:

- **Data-Level Methods:** Modify the training data by either **over-sampling** the minority class (e.g., SMOTE) or **under-sampling** the majority class (e.g., random under-sampling).
- **Algorithm-Level Methods:** Modify the learning algorithm to be **cost-sensitive**, assigning higher misclassification penalties to minority class samples.

- **Hybrid Methods:** Combine resampling and algorithm-level changes for improved balance and robustness.
-

Challenges in Existing Imbalance Handling Techniques

Despite their effectiveness in some cases, these techniques are not without limitations:

- **Over-sampling** increases the dataset size, which may lead to **computational overhead**.
 - **Under-sampling** may discard useful majority samples, causing **information loss**.
 - Repeated **duplication** of minority samples during over-sampling can lead to **overfitting**.
 - Due to the **stochastic nature** of sampling, the newly constructed datasets may **fail to reflect** original distribution patterns.
 - **Algorithm-level solutions alone** (including ensembles) may be **unstable**, especially under extreme imbalance conditions.
-

Proposed Technique: CSBBoost with GSMOTENC

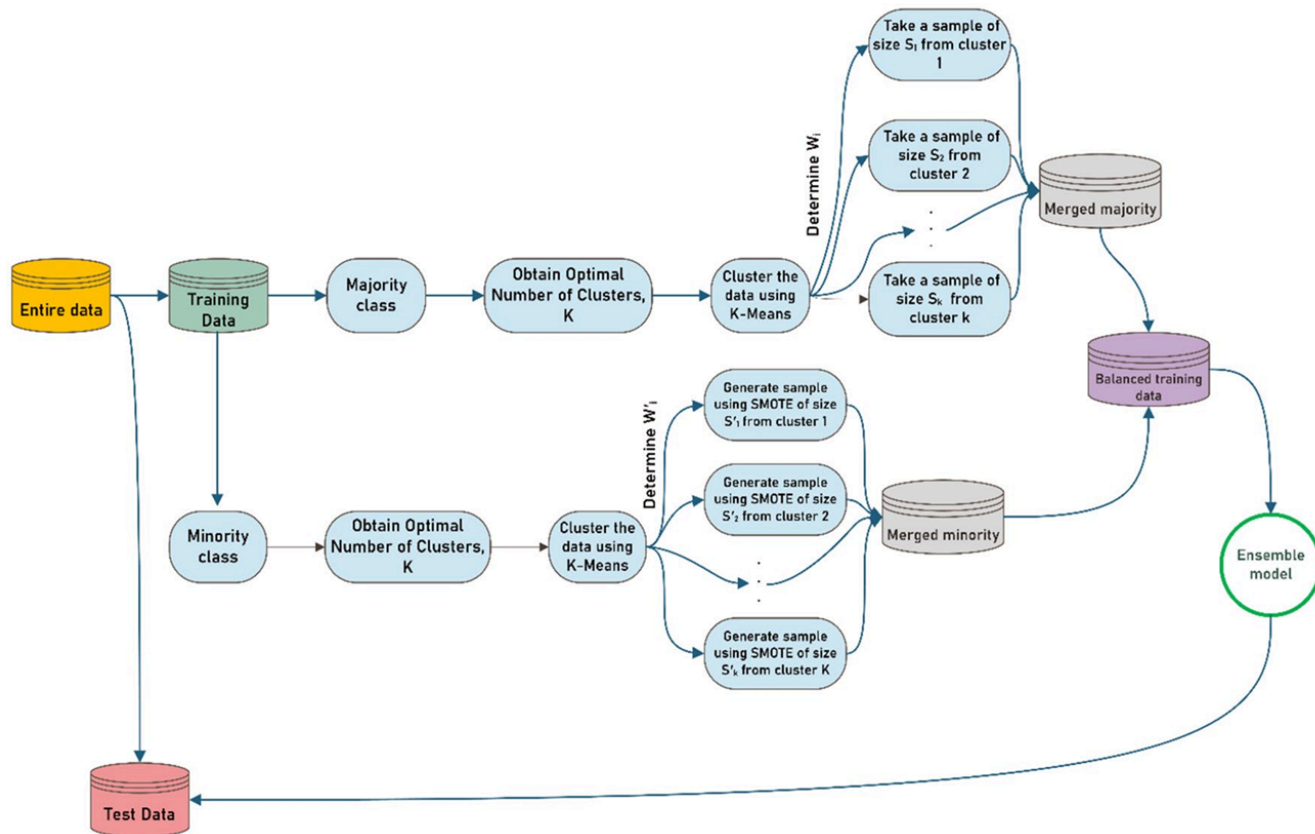
As a robust solution, we explore a recently published technique from *Nature*, called **CSBBoost** – a **cost-sensitive boosting algorithm** that integrates an advanced over-sampling strategy. Unlike traditional SMOTE, it is compatible with **nonlinear and categorical features**, thanks to the use of **GSMOTENC**.

♦ Why GSMOTENC?

GSMOTENC (Geometric SMOTE for Nominal and Continuous features) introduces **nonlinear extrapolation**, generating synthetic samples **without simple interpolation** or duplication. This preserves **feature interactions**, especially in

mixed-type datasets, making it ideal for structured, tabular data with categorical variables.

(The pseudo-code of CSBBoost is provided here) : [LINK](#)



Evaluation and Justification

The reliability and effectiveness of CSBBoost are demonstrated through comparisons against **eight state-of-the-art algorithms** from the literature. The evaluation involves:

- **Multiple datasets** with varying **imbalance ratios**.

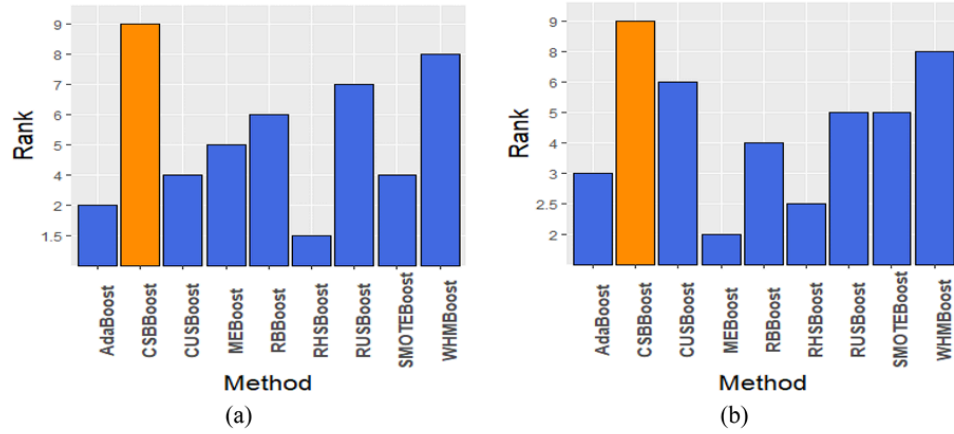


Figure 2. The ascending median rank of the proposed CSBBoost algorithm compared to other algorithms for (a) AUC, and (b) F1.

- Comprehensive performance metrics including **F1-score**, **G-mean**, **Precision**, and **Recall**.

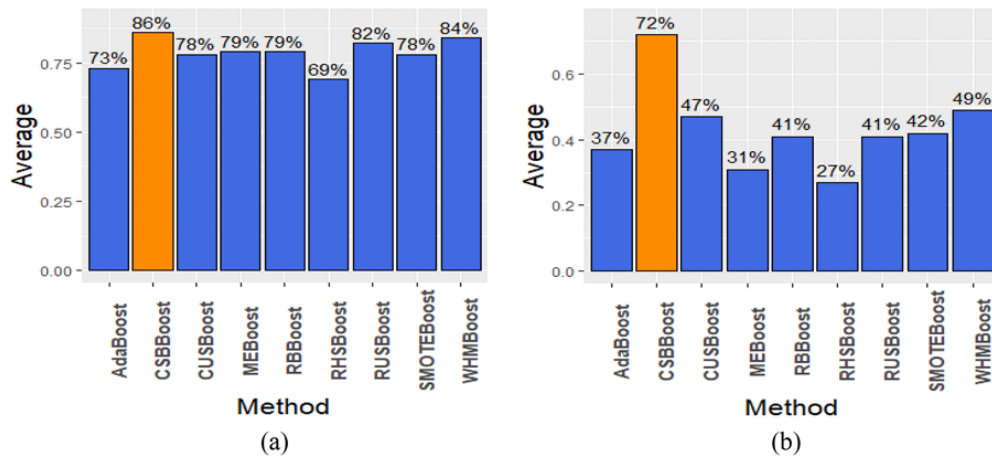


Figure 3. The Average of (a) AUC, and (b) F1 for proposed CSBBoost algorithm compared to other algorithms.

These results show that CSBBoost achieves **more stable and consistent performance** across imbalanced scenarios.

For full performance benchmarks, refer to the original paper: [\(LINK\)](#)

Post-Sampling Training Pipeline

Once the synthetic samples are generated using **GSMOTENC** under the CSBBoost framework, they are appended to the training set. The pipeline then continues as follows:

1. **Train a classifier** (e.g., LightGBM, XGBoost, or any surrogate model) on the augmented dataset.
2. **Evaluate model generalization** using stratified k-fold cross-validation.

This plan ensures that in case generative models fail to produce stable improvements (due to mode collapse or poor latent structure), **a non-training, purely data-level augmentation pipeline is still viable**

Simplest Plan : Simplest Approach for Resource-Constrained Scenarios

In scenarios where **Bayesian Optimization (BO)** becomes computationally infeasible—especially with large datasets—the following lightweight yet effective pipeline can be deployed:

1. Skewness Correction via Oversampling

The first step involves **correcting the imbalance in the dataset**. This can be done using one of the previously discussed techniques such as:

- **GSMOTENC** (for categorical + numerical features),
- **SMOTE** (for continuous features),
- or **CSBBoost** (a boosting-based approach incorporating cost sensitivity and oversampling).

These approaches aim to reduce label bias, increase classifier sensitivity to the minority class, and improve generalization on real-world class distributions.

2. Model Training & Tuning using H2O.ai

Once the data is balanced, the next step is to leverage **H2O.ai**, an **automated machine learning (AutoML)** framework that enables efficient model training and hyperparameter tuning.

♦ What is H2O.ai?

H2O.ai is a scalable, open-source machine learning platform designed for **parallel, distributed model training**. The cloud-based AutoML system was originally introduced with support from Google Cloud and supports both **CPU and GPU acceleration**.

Key advantages include:

- **Model selection** from a library of algorithms (e.g., XGBoost, GLM, Deep Learning, Random Forest).
- **Hyperparameter tuning** using random/grid search and stacked ensembles.
- **Automatic preprocessing** (handling missing values, standardization, encoding).
- **Parallel processing** to train and evaluate **multiple models simultaneously**, significantly reducing time-to-result.

This plan trades off the **exploration-driven** nature of Bayesian Optimization for **speed and simplicity**, making it suitable for real-time, low-resource deployments or prototyping phases.

Exploring Relationships Using Models

Finally after we get a model which generalizes well over our data we will try to understand the relationship between features and the target variable is critical for both **model interpretability** and **domain understanding**. This can be achieved using the model's internal logic, external interpretation libraries like **SHAP**, and thorough **Exploratory Data Analysis (EDA)**.


1. Using In-Built Model Functionalities

Many machine learning models provide built-in tools for evaluating feature importance:

Random Forests and Gradient Boosting (e.g., XGBoost, LightGBM)

These tree-based models expose:

- **Gini Importance (Mean Decrease in Impurity)**: Measures how much a feature reduces uncertainty (impurity) when used in a decision tree split.
- **Gain/Weight/Cover**: Specific to boosting models; they represent how valuable a feature is in decision-making during boosting rounds.

 These metrics help identify which features most strongly influence model predictions.

2. SHAP (SHapley Additive exPlanations)

What is SHAP?

SHAP assigns **each feature an importance value for a specific prediction**, grounded in **game theory** (Shapley values). It provides **both global and local explanations**, making it one of the most robust interpretation tools.

Why is SHAP Useful?

- Works with **any model** (tree, linear, neural net).

- **Considers feature interaction**—unlike traditional importance scores.
- Visualizations like:
 - **SHAP Summary Plot**: Overall impact of each feature.
 - **SHAP Dependence Plot**: Non-linear effects and interactions.
 - **Force Plot**: Local explanation for individual predictions.

⚙️ SHAP Formula:

For a model f and feature set x :

$$f(x) = \phi_0 + \sum_{i=1}^n \phi_i$$

Where:

- ϕ_0 : Base value (average prediction),
- ϕ_i : Contribution of feature i .

3. Exploratory Data Analysis (EDA)

Before or alongside model training, EDA is essential to uncover relationships and spot potential biases or correlations.

Key EDA Methods:

A. Univariate Analysis

- Histogram / KDE plots for distributions.
- Box plots to detect outliers.

- Value counts for categorical data.

B. Bivariate Analysis

- **Scatter plots:** For continuous variables (with hue for class).
- **Boxplots/Violin plots:** To compare distributions across classes.
- **Correlation Matrix** (heatmap):

$$\text{Corr}(X, Y) = \frac{\text{Cov}(X, Y)}{\sigma_X \cdot \sigma_Y}$$

- Reveals linear dependencies.

C. Multivariate Analysis

- **Pair plots:** Multiple scatter plots across features.
- **PCA (Principal Component Analysis):** To visualize clusters and reduce dimensionality.
- **t-SNE / UMAP:** For high-dimensional data projection.