# ASASF 2026 Student Data Challenge – Prediction Track

**Team Name:** Team V.I.P
**Competition Level:** Graduate

## Objective

Our goal was to predict `LBDHDD_outcome` for the 200-row test dataset using the 1,000-row training dataset. Since the initial competition ranking is based on RMSE, we designed the workflow for strong out-of-sample prediction while keeping it robust to common survey-data issues such as missingness, coded nonresponse values, skewed predictors, and mixed data types.

We used a structured pipeline: data cleaning, model-specific preprocessing, repeated cross-validation for tuning, and a stacked ensemble for final prediction generation.

## Data Overview

The dataset includes 97 NHANES-style variables for 1,200 individuals. Predictors cover dietary recall, demographics, anthropometrics, alcohol indicators, and diet behavior variables. The target variable, `LBDHDD_outcome`, is a noise-perturbed version of Direct HDL-Cholesterol (mg/dL).

The training dataset contains the outcome and was used for model development and validation. The test dataset excludes the outcome and was used only for final predictions. We verified that the outcome appeared only in the training set and that predictor columns aligned between train and test.

## Modeling Approach and Preprocessing Choices

We treated preprocessing as part of the modeling strategy because data quality decisions materially affect prediction performance.

### Sentinel values and missing-data handling

Survey datasets often contain coded nonresponse values (e.g., 7, 9, 77, 99, 777, 999). We screened numeric predictors for suspicious value patterns and recoded common sentinel values to missing (`NA`) in flagged variables to avoid false numeric signal.

Instead of dropping incomplete rows, we preserved all observations and handled missingness in preprocessing using:

- missingness indicators,
- median imputation for numeric predictors,
- mode imputation for categorical predictors.

This allowed models to use both the imputed value and the missingness pattern, which can be informative in survey data.

### Encoding, filtering, and outlier control

Categorical predictors were one-hot encoded to create a consistent numeric feature space across model families. We also removed zero-variance and near-zero-variance predictors to reduce noise and improve computational efficiency.

To limit the influence of extreme values, numeric predictors were winsorized using training-derived quantile bounds, and the same bounds were applied to the test set. This improved stability without introducing leakage.

### Model-specific preprocessing pipelines

We used separate preprocessing recipes for different model classes:

- **Tree-based models** (XGBoost, Random Forest, Cubist, BART): imputation, missingness indicators, dummy encoding, and variance filtering.
- **Scale-sensitive models** (Ridge, SVM-RBF, MARS, Neural Networks): the same steps plus Yeo-Johnson transformation and normalization for numeric predictors.

This setup matched preprocessing to model assumptions and improved compatibility across the candidate model set.

## Validation and/or Tuning Strategy

We estimated generalization performance using repeated cross-validation with RMSE as the tuning metric.

### Cross-validation design

- 5-fold cross-validation
- 2 repeats
- 10 total resamples
- RMSE for model comparison and hyperparameter tuning

We used repeated cross-validation instead of a single validation split to obtain more stable performance estimates and reduce sensitivity to one random partition.

### Hyperparameter tuning and execution

Each model was tuned using a model-specific hyperparameter grid (e.g., regularization strength, tree depth, learning rate, number of trees, kernel parameters, hidden units, dropout). We saved cross-validated predictions/workflows during tuning to support stacking and used parallel processing to improve runtime.

We also used a fault-tolerant tuning workflow so that if one model encountered an implementation issue, the rest of the pipeline continued.

## Candidate Models Evaluated

We used a diverse set of models to capture different signal structures in mixed-type tabular data with likely nonlinearities and interactions:

- **Ridge Regression** (`glmnet`) as a regularized linear baseline for correlated predictors.
- **XGBoost** (`xgboost`) as a primary nonlinear learner for tabular prediction.
- **Random Forest** (`ranger`) as a robust nonparametric benchmark.
- **Cubist** (`Cubist`) for rule-based partitioning with local linear fits.
- **MARS** (`earth`) for nonlinear basis expansion modeling.
- **SVM-RBF** (`kernlab`) for flexible nonlinear regression after scaling.
- **Neural Network** (`brulee`) for additional nonlinear modeling capacity.
- **BART** (`dbarts` via `parsnip`) as a core Bayesian tree-ensemble model.

BART was included as part of the main candidate pool because it captures nonlinearities and high-order interactions with a different inductive bias than XGBoost, Random Forest, and Cubist. This improved model diversity and strengthened the ensemble stage.

## Final Model Used to Generate Predictions

Our final submission was generated using a **stacked ensemble** built from the tuned base models, including Ridge Regression, XGBoost, Random Forest, Cubist, MARS, SVM-RBF, Neural Network, and BART.

We chose stacking rather than selecting a single model because the cross-validated results indicated that different learners captured different parts of the signal. Combining them improved robustness and predictive performance under the competition RMSE criterion.

### Final modeling process

1. Tune each base model using repeated cross-validation and RMSE.
2. Retain tuned models for ensemble construction.
3. Build a stacked ensemble using out-of-fold predictions.
4. Fit the final ensemble on the training data.
5. Generate predictions for the test set in the original row order.

The final predictions were exported in the required submission format (`pred.csv`) with one column named `pred`, and the complete analysis code (including preprocessing, validation, tuning, ensembling, and test-set prediction generation) is available in our GitHub repository: GitHub link.