# Statistical Modeling and Learning

## Final Presentation – Summer Internship


**Indian Institute of Technology Madras**

**Saurabh Kumar Gupta**
*Department of Mathematics & Computing, NIT Mizoram*

Supervisor: **Prof. Neelesh Shankar Upadhye**,
Department of Mathematics, IIT Madras

July 14, 2025

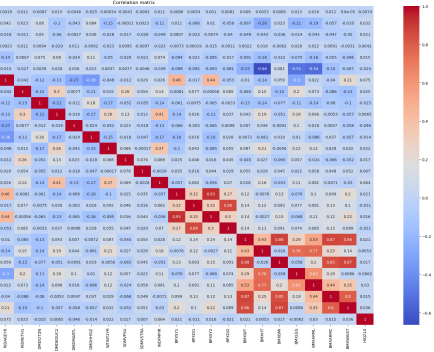
# Outline

# Introduction

- In this project, we aim to understand how different regression techniques perform in predicting Body Mass Index (BMI) using the NHANES dataset.
- We begin with classical linear models and move toward advanced techniques including regularization, probabilistic modeling, and ensemble learning.
- We'll also explore how introducing interaction terms and performing data cleaning affects model performance.
- Finally, we compare hand-coded models against in-built implementations and visualize performance gains.

# Dataset Overview: NHANES

- We use the **NHANES** dataset (National Health and Nutrition Examination Survey), a US-based public health dataset.
- It includes demographic, behavioral, and health-related variables.
- Our target variable is **BMI (Body Mass Index)**.
- Features include age, gender, blood pressure, education, income, alcohol consumption, and physical measuremen



# Data Cleaning and Feature Engineering

- **Handled Missing Values:** Replaced with column-wise mean using ‘fill-mean‘ strategy.
- **Dropped Redundant Columns:** Like IDs and uninformative variables (e.g., survey station codes SEQN).
- **Standardized Numeric Features:** Ensured consistent scales across features.
- **Interaction Terms:** Created new variables like Age $\times$ BP to capture multiplicative effects.
- **Feature Selection:** Retained features with meaningful correlation to BMI and relevance to health.

*All steps are modularized in the custom script:* `cleaning_module.py`

# Ordinary Least Squares (OLS)

- **What it does:** Minimizes squared errors to find best-fit coefficients.
- **Why it's used:** Simple baseline with full interpretability.
- **Mathematics:** $\hat{\beta} = (X^T X)^{-1} X^T y$
- **Strength:** Direct insight into relationships.
- **Limitation:** Can overfit, assumes linearity.
- **Use Case:** When data is linearly distributed.

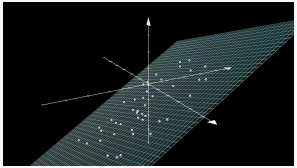


**Fig 1:** Linear Regression Algorithm

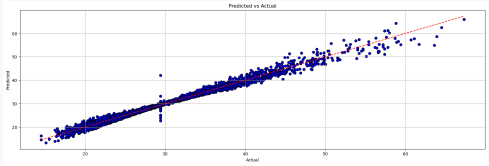



**Fig 2:** Actual vs Predicted BMI (OLS)

**OLS Performance**

Train $R^2$: 0.9848 Test $R^2$: 0.9853

# Ridge Regression (L2 Regularization)

- **Idea:** Adds penalty on large coefficients.
- **Math:** $\hat{\beta}^{ridge} = \arg\min \|y - X\beta\|^2 + \lambda\|\beta\|^2$
- **Strength:** Prevents overfitting, especially in multicollinearity.
- **Limitation:** May bias coefficients.
- **Use Case:** High-dimensional or noisy datasets.

**Ridge Performance**

Train $R^2 = 0.9419$
Test $R^2 = 0.9515$

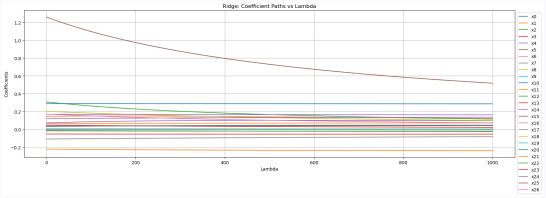



**Fig 1:** Ridge Coefficient Shrinkage

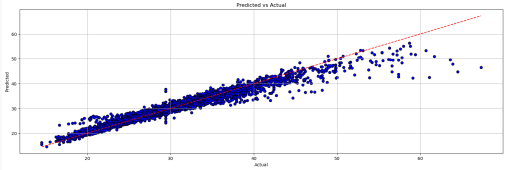



**Fig 2:** Actual vs Predicted BMI (Ridge)

# Lasso Regression (L1 Regularization)

- **Idea:** Adds L1 penalty to encourage sparsity.
- **Math:** $\hat{\beta}^{lasso} = \arg\min \|y - X\beta\|^2 + \lambda\|\beta\|_1$
- **Strength:** Performs variable selection (some weights become 0).
- **Limitation:** May underperform when features are correlated.
- **Use Case:** High-dimensional data with irrelevant features.

**Lasso Performance**

Train $R^2 = 0.9848$

Test $R^2 = 0.9855$

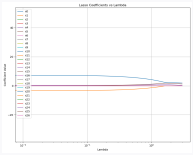



**Fig 1:** Lasso Coefficient Shrinkage

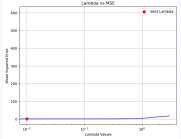



**Fig 2:** Cross-Validation for Best $\lambda$

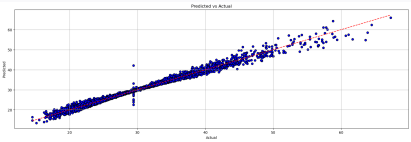



**Fig 3:** Actual vs Predicted BMI (Lasso)

# Regression Tree

- **Idea:** Split data recursively into homogeneous regions.
- **Math:** Greedy minimization of mean squared error at splits.
- **Strength:** Easy to interpret, non-linear patterns captured.
- **Limitation:** Can overfit without pruning or depth control.
- **Use Case:** When decision rules matter or data is non-linear.

**Regression Tree Performance**

Train $R^2 = 0.9206$

Test $R^2 = 0.9235$

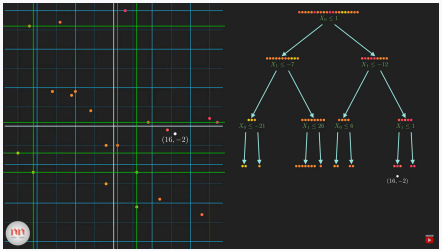



**Fig 1:** Regression Tree Structure (How splits happen)

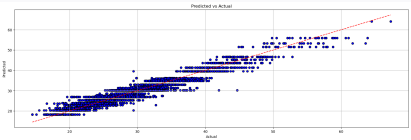



**Fig 2:** Actual vs Predicted BMI (Regression Tree)

# Random Forest Regression

- **Idea:** Ensemble of decision trees via bagging.
- **Math:** Averages predictions from multiple trees.
- **Strength:** Reduces overfitting, robust on noisy data.
- **Limitation:** Harder to interpret than single trees.
- **Use Case:** Complex, high-dimensional datasets.

**Random Forest Performance**

Train $R^2 = 0.9698$

Test $R^2 = 0.9627$

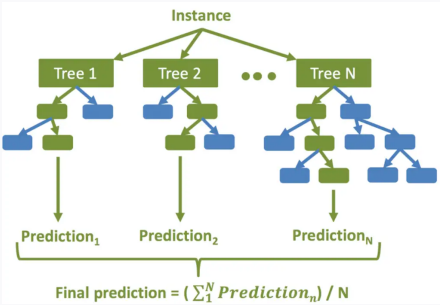



**Fig 1:** Random Forest Workflow

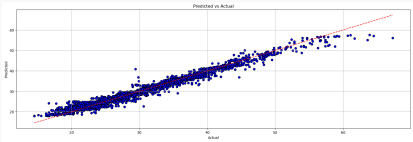



**Fig 2:** Actual vs Predicted BMI (Random Forest)

# XGBoost Regression

- **Idea:** Gradient Boosting with optimized speed and regularization.
- **Math:** Sequential trees minimize regularized loss.
- **Strength:** Handles missing data, overfitting-resistant.
- **Limitation:** Hyperparameter tuning is complex.
- **Use Case:** High-dimensional data with noise or imbalance.

**XGBoost Performance**

Train $R^2 = 0.9866$

Test $R^2 = 0.9827$

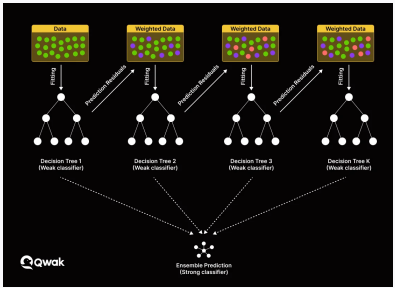



**Fig 1:** XGBoost Algorithm Overview

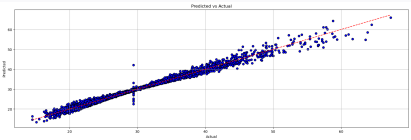



**Fig 2:** Actual vs Predicted BMI (XGBoost)

# Bayesian Linear Regression

- **Idea:** Adds uncertainty to regression.
- **Math:** Posterior $\propto$ Likelihood $\times$ Prior
- **Strength:** Confidence-aware predictions.
- **Limitation:** Computationally intensive.
- **Use Case:** Small data or when uncertainty matters.

**Bayesian Performance**

Train $R^2 = 0.9659$

Test $R^2 = 0.9644$

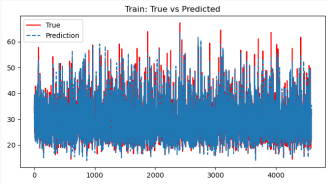



**Fig 1:** True vs Predicted (Bayesian)

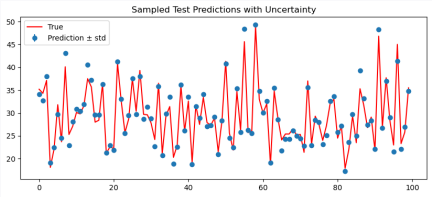


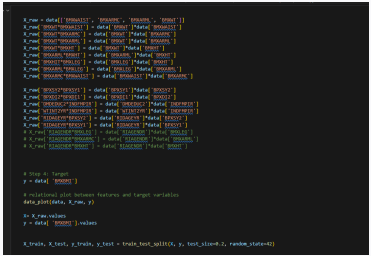

**Fig 2:** Bayesian Regression Fit

# Interaction Terms in Linear Models

- Interaction terms = pairwise feature combinations.
- Capture joint effects (e.g., Weight × Waist).
- Significantly boost linear models accuracy.
- Tree models already learn interactions inherently.

**$R^2$ Scores with Interaction Terms**

| Model | Train | Test |
|---|---|---|
| Linear | 0.9959 | 0.9967 |
| Ridge | 0.9435 | 0.9494 |
| Lasso | 0.9937 | 0.9945 |
| Bayesian | 0.9956 | 0.9960 |



```
X_raw = data[['BMXARMC', 'BMXARML', 'BMXWT']]
X_raw['BMXWT*BMXARMC'] = data['BMXWT']*data['BMXARMC']
X_raw['BMXWT*BMXARML'] = data['BMXWT']*data['BMXARML']
X_raw['BMXWT*BMXWAIST'] = data['BMXWT']*data['BMXWAIST']
X_raw['BMXWT*BMXWT'] = data['BMXWT']*data['BMXWT']
X_raw['BMXARML*BMXWT'] = data['BMXARML']*data['BMXWT']
X_raw['BMXWAIST*BMXWAIST'] = data['BMXWAIST']*data['BMXWAIST']

X_raw['BMXBMI*BMXWAIST'] = data['BMXBMI']*data['BMXWAIST']
X_raw['BMXWAIST*BMXWAIST'] = data['BMXWAIST']*data['BMXWAIST']
X_raw['SMOKEDCAT*NSMOKERS'] = data['SMOKEDCAT']*data['NSMOKERS']
X_raw['WAISTCAT*NSMOKERS'] = data['WAISTCAT']*data['NSMOKERS']
X_raw['RIDAGEYR*RIDAGEYR'] = data['RIDAGEYR']*data['RIDAGEYR']
X_raw['RIDAGEYR*BMXWAIST'] = data['RIDAGEYR']*data['BMXWAIST']
# X_raw['RIAGENDR*BMXBMI'] = data['RIAGENDR']*data['BMXBMI']
# X_raw['RIAGENDR*BMXARML'] = data['RIAGENDR']*data['BMXARML']
# X_raw['RIAGENDR*BMXWT'] = data['RIAGENDR']*data['BMXWT']


# Step 4: Target
y = data['BMXWT']

# relational plot between features and target variables
data_plot(data, X_raw, y)

X= X_raw.values
y = data['BMXWT'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

**Fig 1:** Pairwise Feature Interactions

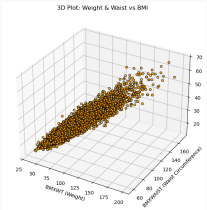



**Fig 2:** BMI vs BMXWT × BMXWAIST

# Model Performance Comparison

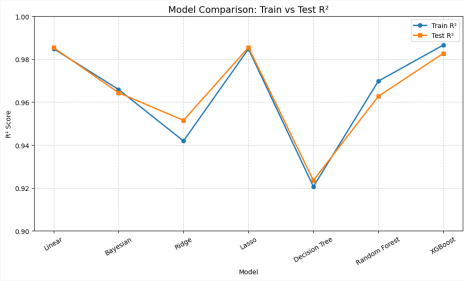
## Model Performance Comparison ($R^2$ Scores)

| Model | Train $R^2$ (Full) | Test $R^2$ (Full) | Train $R^2$ (Interaction) | Test $R^2$ (Interaction) | Test $R^2$ (Inbuilt) |
|---|---|---|---|---|---|
| **Linear Regression (OLS)** | 0.9849 | 0.9854 | 0.9960 | 0.9967 | 0.9854 |
| **Bayesian Linear Reg.** | 0.9659 | 0.9644 | 0.9956 | 0.9960 | 0.9967 |
| **Ridge Regression** | 0.9419 | 0.9515 | 0.9435 | 0.9494 | 0.9854 |
| **Lasso Regression** | 0.9848 | 0.9855 | 0.9937 | 0.9945 | 0.9854 |
| **Regression Tree** | 0.9206 | 0.9235 | 0.9206 | 0.9235 | 0.9068 |
| **Random Forest** | 0.9698 | 0.9627 | 0.9698 | 0.9627 | 0.9447 |
| **XGBoost** | 0.9866 | 0.9827 | 0.9866 | 0.9827 | 0.9943 |

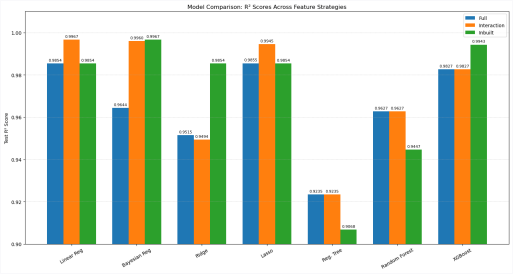
**Takeaway**

Feature engineering (interaction terms) can make simple models outperform complex algorithms

# Model $R^2$ Comparison (Test vs Train)




Train vs Test $R^2$ Curve




Bar Chart of Model Comparison

- Linear + Interaction Terms achieved highest test $R^2$ (0.9967)
- Bayesian and XGBoost close, but more complex
- Simpler linear models beat tree-based models with feature engineering

**Best Trade-off:** Linear Regression with Interaction Terms

# Overall Summary & Takeaways

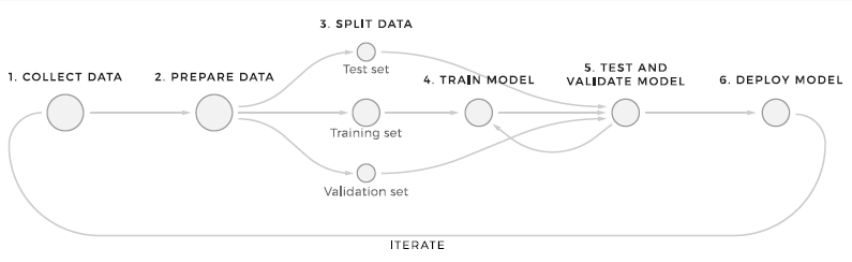



**Fig:** Model Pipeline Summary or Key Learning Visual

- Explored multiple regression models to predict BMI.
- Linear models gave high accuracy after adding interaction terms.
- Bayesian Linear Regression provided uncertainty-aware predictions.
- Regularization (Ridge/Lasso) helped handle multicollinearity and sparsity.
- Tree-based models captured non-linearities but slightly underperformed.
- Feature engineering (cleaning + interactions) was a game-changer.

**Key Insight:**

# From Code to Care: Real-World Applications

**Dataset Used:** NHANES — real-world health records

**Goal:** Predict BMI from easy body metrics (waist, height, weight)

**Why it Matters:**

- Early screening without lab tests
- Works in rural/low-resource areas
- Enables mobile health diagnostics

**Other Uses:**

- School health tracking
- Insurance profiling
- Hospital triage systems

**Future Scope:**

- Integration with wearables (e.g., Fitbit)
- BMI prediction from camera/mobile images
- Personalized lifestyle + diet guidance
- Embedded in telemedicine platforms

**Vision:**
*Smart, low-cost, scalable health tools for all.*



# Thank You!

## Questions and Discussions Welcome

*Presented by: Saurabh Kumar Gupta*
*Summer Internship – Statistical Modeling and Learning*
*National Institute of Technology, Mizoram*