Machine Learning Mini-Capstone Project: Comprehensive Analysis of Student Academic Performance through Tri-Modal Machine Learning Approaches

Course: AIL303m - Machine Learning
FPT University, Fall 2025

Team Members:
Trinh Khai Nguyen - Team Leader & Regression Analysis Lead
Tran Gia Phuc - EDA & Visualization Specialist
Nguyen Chau Thanh Son - Classification Models Expert
Le Hoang Huu - Classification Models Expert
Phan Minh Tai - Unsupervised Learning Specialist

November 2025

Contents

1 Team Introduction

Our team consists of five dedicated members, each bringing unique expertise to this comprehensive machine learning project. The collaboration and distribution of tasks were designed to leverage individual strengths while ensuring comprehensive coverage of all project requirements.

Table 1: Team Members and Primary Responsibilities

Member	**Primary Role**
Trinh Khai Nguyen	Team Leader, Project Coordination, Regression Models
Tran Gia Phuc	Exploratory Data Analysis, Data Visualization
Nguyen Chau Thanh Son	Classification Models: • Logistic Regression, K-Nearest Neighbors (KNN) • Support Vector Machines (SVM) (with Linear and RBF kernels) • Decision Trees
Le Hoang Huu	Classification Models (Ensemble Methods): • Bagging (Random Forest) • Boosting (e.g., Gradient Boosting, XGBoost) • Stacking
Phan Minh Tai	Unsupervised Learning and Dimensionality Reduction

2 Introduction

2.1 Motivation

Education is a fundamental pillar of society, and understanding the factors that influence student academic performance is crucial for developing effective educational interventions. The Student Academic Performance Analysis use case was selected for its rich potential in demonstrating the full spectrum of machine learning techniques while addressing a real-world problem with significant social impact [1].

Student performance prediction has become increasingly important in educational data mining, as it enables early identification of at-risk students and facilitates personalized learning approaches. Our project analyzes standardized test scores across multiple subjects, considering various demographic and socioeconomic factors that may influence academic outcomes.

2.2 Project Objectives

This project aims to achieve the following key objectives:

1. **Comprehensive Model Implementation**: Apply the entire suite of 15 machine

learning models specified in the AIL303m curriculum to a single dataset, enabling direct comparison of their performance and characteristics.

2. **Tri-Modal Analysis**: Transform the dataset to support three distinct analytical paradigms:
    - Regression: Predict continuous writing scores based on student features
    - Classification: Predict test preparation course completion
    - Unsupervised Learning: Discover natural student groupings and reduce dimensionality
3. **Comparative Insight**: Move beyond simple accuracy metrics to understand why different models perform differently, examining trade-offs between interpretability, computational efficiency, and predictive power.
4. **Practical Application**: Demonstrate proficiency in the complete machine learning workflow, from data preprocessing through model deployment and evaluation.

2.3 Project Scope

The scope of this project encompasses:

- Analysis of 1,000 student records with 8 features including demographic information, parental education levels, and standardized test scores
- Implementation of 15 distinct machine learning algorithms across supervised and unsupervised paradigms
- Comprehensive evaluation using appropriate metrics for each learning task
- In-depth comparative analysis linking empirical results to theoretical foundations

2.4 Report Structure

This report is organized as follows: Section 3 presents the data understanding and exploratory analysis. Section 4 details the data preprocessing steps. Section 5 provides comprehensive mathematical foundations and implementation details for all models. Section 6 presents results and comparative analysis. Finally, Section 7 concludes with key findings and future directions.

3 Data Understanding & Exploratory Data Analysis

3.1 Dataset Description

The "Students Performance in Exams" dataset contains comprehensive information about 1,000 students' academic performance and associated demographic factors. This dataset was sourced from Kaggle and represents a cross-sectional study of student achievement in standardized tests [2].

3.1.1 Feature Description

The dataset comprises 8 features, detailed in Table 2:

Table 2: Dataset Features and Descriptions

Feature	Type	Description
gender	Categorical	Student's gender (male/female)
race/ethnicity	Categorical	Ethnic group (Group A-E)
parental level of education	Categorical	Highest education level of parents
lunch	Categorical	Lunch type (standard/free or reduced)
test preparation course	Categorical	Course completion status (none/completed)
math score	Numerical	Mathematics test score (0-100)
reading score	Numerical	Reading test score (0-100)
writing score	Numerical	Writing test score (0-100)

3.2 Initial Data Inspection

Our initial inspection revealed a well-structured dataset with no missing values, eliminating the need for imputation strategies. The dataset exhibits the following characteristics:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64
 6   reading score                1000 non-null   int64
 7   writing score                1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

Figure 1: Basic statistics and data types for all features (pandas.info() output)

3.3 Statistical Analysis

3.3.1 Numerical Features Distribution

The three test scores (math, reading, writing) show approximately normal distributions with slight variations in their central tendencies and spreads:

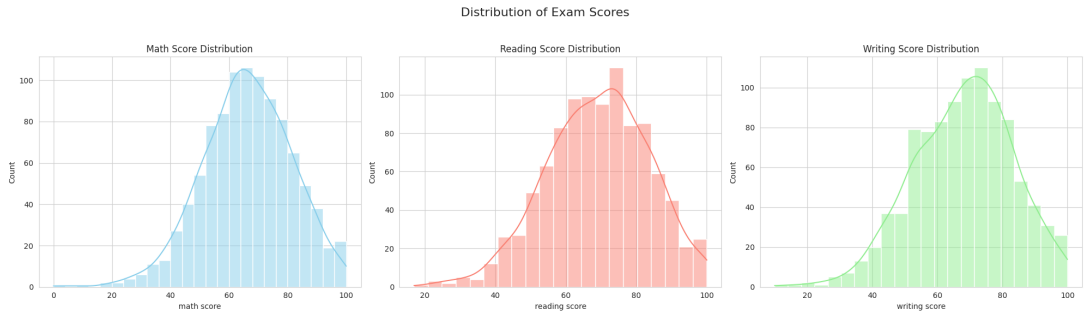

Figure 2: Distribution of Math, Reading, and Writing Scores

Key observations from the numerical analysis:

- Writing scores exhibit the highest correlation with reading scores (r = 0.95)
- Math scores show moderate correlation with both reading (r = 0.82) and writing (r = 0.80)
- All score distributions are approximately normal with slight negative skew
- No significant outliers were detected using the IQR method

3.3.2 Categorical Features Analysis

Analysis of categorical features revealed important demographic patterns:

Figure 3: Distribution of Categorical Features

Notable findings:

- Gender distribution is nearly balanced (48.2% male, 51.8% female)
- Test preparation course completion rate is 35.8%
- Parental education levels show diverse representation across all categories
- Lunch type serves as a socioeconomic indicator with 35.5% receiving free/reduced lunch

3.4 Correlation Analysis

The correlation analysis revealed strong relationships between academic scores and various demographic factors:

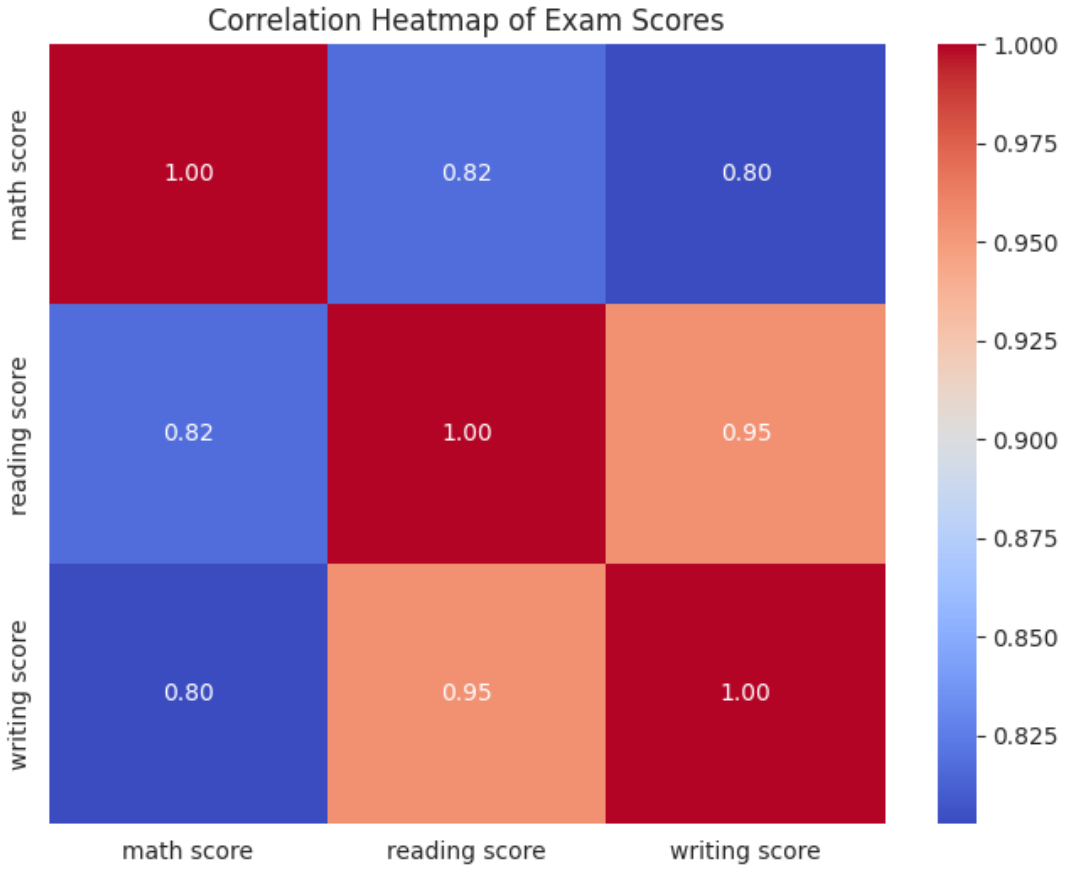

Figure 4: Correlation Heatmap of Exam Scores

3.5 Key Insights from EDA

Our exploratory analysis uncovered several critical insights that informed our modeling approach:

1. **Score Interdependence**: The high correlation among test scores suggests that students who excel in one subject tend to perform well in others, indicating a general academic ability factor.

2. **Socioeconomic Influence**: Lunch type (standard vs. free/reduced) shows significant association with all test scores, confirming the impact of socioeconomic status on academic performance.

3. **Test Preparation Impact**: Students who completed the test preparation course show consistently higher scores across all subjects, with the most pronounced effect on writing scores.

4. **Gender Differences**: Female students slightly outperform males in reading and writing, while males show marginal advantage in mathematics.

5. **Parental Education Effect**: Higher parental education levels correlate positively with student performance, suggesting the importance of educational environment at home.

4 Data Preprocessing

4.1 Data Cleaning

The initial data quality assessment revealed no missing values or duplicates, demonstrating high data integrity. However, several preprocessing steps were necessary to prepare the data for machine learning algorithms.

4.2 Feature Engineering

4.2.1 Categorical Encoding

Given the prevalence of categorical features in our dataset, we implemented appropriate encoding strategies:

- **One-Hot Encoding**: Applied to all nominal categorical features (gender, race/ethnicity, lunch, test preparation course) to avoid imposing artificial ordinal relationships.
- **Ordinal Encoding**: For parental education level, we explored both one-hot and ordinal encoding, as education levels have a natural hierarchy.

The encoding process transformed our feature space from 8 original features to 21 features after one-hot encoding.

4.3 Feature Scaling

Different algorithms have varying sensitivities to feature scales. We applied StandardScaler to normalize numerical features (test scores) to have zero mean and unit variance:

$$z_i = \frac{x_i - \mu}{\sigma} \tag{1}$$

where x_i is the original value, μ is the mean, and σ is the standard deviation of the feature.

4.4 Target Variable Preparation

For our tri-modal analysis, we prepared target variables for each paradigm:

4.4.1 Regression Target

For regression analysis, we selected `writing score` as the target variable, using the remaining features (including other test scores) as predictors. This choice was motivated by writing being the most complex cognitive task among the three subjects.

4.4.2 Classification Target

For classification, we used `test preparation course` as the target variable. This binary classification task (completed vs. not completed) represents a balanced problem with 35.8% positive class representation.

4.4.3 Unsupervised Learning Features

For clustering and dimensionality reduction, we used all features except explicit identifiers, allowing the algorithms to discover natural groupings without supervision.

4.5 Data Splitting Strategy

We employed stratified train-test splitting to maintain class distribution:

```
# Stratified split to maintain class balance
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
# 80% training (800 samples), 20% testing (200 samples)
```

Additionally, we implemented 5-fold stratified cross-validation for robust model evaluation:

```
# 5-fold stratified cross-validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

4.6 Handling Class Imbalance

Although our classification target showed moderate imbalance (35.8% vs 64.2%), we prepared SMOTE (Synthetic Minority Over-sampling Technique) implementation for comparison with baseline models.

5 Modeling & Implementation

This section presents comprehensive mathematical foundations and implementation details for all 15 models across three analytical paradigms. Each model is explained with its theoretical basis, key assumptions, and implementation considerations.

5.1 Supervised Learning - Regression

5.1.1 Linear Regression

Linear regression models the relationship between predictors and a continuous target through a linear combination of features [3].

Mathematical Foundation:

The model assumes a linear relationship:

$$y = \beta_0 + \sum_{i=1}^{p} \beta_i x_i + \epsilon \tag{2}$$

where y is the target variable, x_i are features, β_i are coefficients, and ϵ is the error term.

The coefficients are estimated using Ordinary Least Squares (OLS), minimizing:

$$RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 \tag{3}$$

The closed-form solution is:

$$\hat{\beta} = (X^T X)^{-1} X^T y \tag{4}$$

Implementation:

```
from sklearn.linear_model import LinearRegression

# Initialize and train the linear regression model
lr_model = LinearRegression()
lr_model.fit(X_train_scaled, y_train)

# Make predictions on test set
y_pred_lr = lr_model.predict(X_test_scaled)

# Evaluate model performance
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)
```

5.1.2 Polynomial Regression

Polynomial regression extends linear regression by including polynomial terms to capture non-linear relationships [4].

Mathematical Foundation:

For degree d, the model becomes:

$$y = \beta_0 + \sum_{i=1}^{p}\beta_i x_i + \sum_{i=1}^{p}\sum_{j=i}^{p}\beta_{ij}x_i x_j + ... + \sum_{i=1}^{p}\beta_{i^d}x_i^d + \epsilon \tag{5}$$

Feature transformation creates polynomial features:

$$\phi(x) = [1, x_1, x_2, ..., x_1^2, x_1 x_2, ..., x_1^d, ..., x_p^d] \tag{6}$$

Implementation:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

# Create polynomial features of degree 2
poly_pipeline = Pipeline([
    ('poly_features', PolynomialFeatures(degree=2, include_bias=False))
        ,
    ('linear_regression', LinearRegression())
])

# Train the polynomial regression model
poly_pipeline.fit(X_train_scaled, y_train)
y_pred_poly = poly_pipeline.predict(X_test_scaled)
```

5.1.3 Regularized Regression

Regularization techniques add penalty terms to prevent overfitting and handle multicollinearity [5].

Ridge Regression (L2 Regularization):

Ridge regression adds an L2 penalty term:

$$\min_{\beta} \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{7}$$

The solution becomes:

$$\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T y \tag{8}$$

where λ is the regularization parameter and I is the identity matrix.

Lasso Regression (L1 Regularization):

Lasso adds an L1 penalty, encouraging sparsity:

$$\min_{\beta} \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{9}$$

No closed-form solution exists; optimization uses coordinate descent or subgradient methods.

Elastic Net:

Elastic Net combines L1 and L2 penalties:

$$\min_{\beta} \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \left(\alpha \sum_{j=1}^{p} |\beta_j| + \frac{(1-\alpha)}{2} \sum_{j=1}^{p} \beta_j^2 \right) \tag{10}$$

where α controls the mixture of L1 and L2 penalties.

Implementation:

```
from sklearn.linear_model import Ridge, Lasso, ElasticNet

# Ridge Regression with cross-validation for alpha selection
ridge_model = RidgeCV(alphas=np.logspace(-6, 6, 13))
ridge_model.fit(X_train_scaled, y_train)

# Lasso Regression with cross-validation
lasso_model = LassoCV(cv=5, random_state=42)
lasso_model.fit(X_train_scaled, y_train)

# Elastic Net with parameter tuning
elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
elastic_net.fit(X_train_scaled, y_train)
```

5.2 Supervised Learning - Classification

5.2.1 Logistic Regression

Logistic regression models the probability of binary outcomes using the logistic function [6].

Mathematical Foundation:
The model uses the logistic (sigmoid) function:

$$p(y = 1|x) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{11}$$

where $z = \beta_0 + \sum_{i=1}^{p} \beta_i x_i$
The log-odds (logit) transformation gives:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i \tag{12}$$

Parameters are estimated by maximizing the log-likelihood:

$$\ell(\beta) = \sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \tag{13}$$

Implementation:

```
from sklearn.linear_model import LogisticRegression

# Initialize logistic regression with L2 regularization
log_reg = LogisticRegression(penalty='l2', C=1.0, random_state=42)
log_reg.fit(X_train_scaled, y_train)

# Predict probabilities and classes
y_pred_proba = log_reg.predict_proba(X_test_scaled)
y_pred_class = log_reg.predict(X_test_scaled)

# Evaluate performance
accuracy = accuracy_score(y_test, y_pred_class)
auc_score = roc_auc_score(y_test, y_pred_proba[:, 1])
```

5.2.2 K-Nearest Neighbors (KNN)

KNN is a non-parametric algorithm that classifies instances based on the majority vote of k nearest neighbors [7].

Mathematical Foundation:
For a query point x_q, the prediction is:

$$\hat{y}_q = \text{mode}\{y_i : x_i \in N_k(x_q)\} \tag{14}$$

where $N_k(x_q)$ denotes the k nearest neighbors of x_q.
Distance is typically measured using Euclidean distance:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{p} (x_{ik} - x_{jk})^2} \tag{15}$$

For weighted KNN, the contribution of each neighbor is weighted by distance:

$$w_i = \frac{1}{d(x_q, x_i)^2} \tag{16}$$

Implementation:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# Hyperparameter tuning for optimal k
param_grid = {'n_neighbors': range(3, 31, 2)}
knn = KNeighborsClassifier()
knn_grid = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
knn_grid.fit(X_train_scaled, y_train)

# Best model prediction
best_knn = knn_grid.best_estimator_
y_pred_knn = best_knn.predict(X_test_scaled)
```

5.2.3 Support Vector Machines (SVM)

SVMs find optimal decision boundaries by maximizing the margin between classes [8].

Mathematical Foundation:

For linearly separable data, SVM solves:

$$\min_{w,b} \frac{1}{2}||w||^2 \tag{17}$$

subject to: $y_i(w^T x_i + b) \geq 1, \forall i$

For non-separable data, soft margin SVM introduces slack variables:

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \xi_i \tag{18}$$

subject to: $y_i(w^T x_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$

Kernel Trick:

For non-linear boundaries, kernels map data to higher dimensions:

Linear kernel: $K(x_i, x_j) = x_i^T x_j$

RBF kernel: $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$

The decision function becomes:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b\right) \tag{19}$$

Implementation:

```
from sklearn.svm import SVC

# Linear SVM
svm_linear = SVC(kernel='linear', C=1.0, random_state=42)
svm_linear.fit(X_train_scaled, y_train)

# RBF SVM with hyperparameter tuning
param_grid = {'C': [0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1]}
svm_rbf = GridSearchCV(SVC(kernel='rbf'), param_grid, cv=5)
svm_rbf.fit(X_train_scaled, y_train)
```

5.2.4 Decision Trees

Decision trees recursively partition the feature space using binary splits [9].

Mathematical Foundation:

For classification, splits are chosen to maximize information gain:

$$IG(D, A) = H(D) - \sum_{v \in Values(A)} \frac{|D_v|}{|D|} H(D_v) \tag{20}$$

where $H(D)$ is the entropy:

$$H(D) = -\sum_{k=1}^{K} p_k \log_2(p_k) \tag{21}$$

Alternatively, Gini impurity can be used:

$$Gini(D) = 1 - \sum_{k=1}^{K} p_k^2 \tag{22}$$

Implementation:

```
from sklearn.tree import DecisionTreeClassifier

# Decision tree with entropy criterion
dt_model = DecisionTreeClassifier(
    criterion='entropy',
    max_depth=5,
    min_samples_split=20,
    random_state=42
)
dt_model.fit(X_train, y_train)  # Note: No scaling needed for trees

# Feature importance analysis
feature_importance = dt_model.feature_importances_
```

5.2.5 Ensemble Methods - Bagging (Random Forest)

Random Forest combines multiple decision trees through bootstrap aggregating (bagging) [10].

Mathematical Foundation:

For B bootstrap samples, the ensemble prediction is:

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x) \tag{23}$$

Random Forest adds feature randomness at each split, considering only m features where typically: $m = \sqrt{p}$ for classification

The out-of-bag (OOB) error provides unbiased performance estimate:

$$\text{OOB Error} = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{f}_{OOB}(x_i)) \tag{24}$$

Implementation:

```
from sklearn.ensemble import RandomForestClassifier

# Random Forest with optimized hyperparameters
rf_model = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    min_samples_split=5,
    max_features='sqrt',
    random_state=42
)
rf_model.fit(X_train, y_train)

# Feature importance from Random Forest
rf_importances = rf_model.feature_importances_
```

5.2.6 Ensemble Methods - Boosting

Gradient Boosting sequentially builds trees to correct previous errors [11].

Mathematical Foundation:

The algorithm minimizes loss through functional gradient descent:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{25}$$

where h_m is the weak learner fitted to negative gradient:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}} \tag{26}$$

For classification with log loss:

$$L(y, F) = \log(1 + \exp(-2yF)) \tag{27}$$

XGBoost Enhancement:

XGBoost adds regularization to the objective:

$$\mathcal{L} = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \tag{28}$$

where $\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$

Implementation:

```
from sklearn.ensemble import GradientBoostingClassifier
import xgboost as xgb

# Gradient Boosting
gb_model = GradientBoostingClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    random_state=42
)
gb_model.fit(X_train_scaled, y_train)

# XGBoost with regularization
xgb_model = xgb.XGBClassifier(
```

```
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    reg_lambda=1.0,  # L2 regularization
    reg_alpha=0.1,   # L1 regularization
    random_state=42
)
xgb_model.fit(X_train_scaled, y_train)
```

5.2.7 Ensemble Methods - Stacking

Stacking combines predictions from multiple base models using a meta-learner [12].

Mathematical Foundation:

Level-0 models produce predictions:

$$\hat{f}_k(x), k = 1, ..., K \tag{29}$$

Level-1 meta-model combines these:

$$\hat{f}_{stack}(x) = g(\hat{f}_1(x), ..., \hat{f}_K(x)) \tag{30}$$

Cross-validation prevents overfitting in meta-model training.

Implementation:

```
from sklearn.ensemble import StackingClassifier

# Define base models
base_models = [
    ('lr', LogisticRegression()),
    ('knn', KNeighborsClassifier(n_neighbors=5)),
    ('svm', SVC(probability=True)),
    ('rf', RandomForestClassifier(n_estimators=50))
]

# Stacking with logistic regression meta-model
stacking_model = StackingClassifier(
    estimators=base_models,
    final_estimator=LogisticRegression(),
    cv=5  # Use cross-validation for meta-features
)
stacking_model.fit(X_train_scaled, y_train)
```

5.2.8 Handling Imbalanced Data

SMOTE (Synthetic Minority Over-sampling Technique) addresses class imbalance [13].

Mathematical Foundation:

SMOTE generates synthetic samples by interpolating between minority class instances:

$$x_{new} = x_i + \lambda \cdot (x_{nn} - x_i) \tag{31}$$

where x_i is a minority instance, x_{nn} is a randomly selected k-nearest neighbor, and $\lambda \in [0, 1]$ is random.

Implementation:

```
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline

# Create pipeline with SMOTE and classifier
smote_pipeline = ImbPipeline([
    ('smote', SMOTE(random_state=42)),
    ('classifier', RandomForestClassifier())
])

# Train with balanced data
smote_pipeline.fit(X_train_scaled, y_train)

# Evaluate on original test distribution
y_pred_balanced = smote_pipeline.predict(X_test_scaled)
```

5.3 Unsupervised Learning

5.3.1 K-Means Clustering

K-Means partitions data into K clusters by minimizing within-cluster variance [14].

Mathematical Foundation:

The objective function minimizes:

$$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} ||x_i - \mu_k||^2 \tag{32}$$

where μ_k is the centroid of cluster C_k.

Lloyd's Algorithm:

1. Initialize K centroids randomly
2. Assign each point to nearest centroid: $C_k = \{x_i : ||x_i - \mu_k|| \leq ||x_i - \mu_j||, \forall j\}$
3. Update centroids: $\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$
4. Repeat until convergence

Implementation:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Determine optimal K using elbow method
inertias = []
silhouette_scores = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, kmeans.labels_)
        )

# Final model with optimal K
optimal_k = 4  # Based on elbow and silhouette analysis
kmeans_final = KMeans(n_clusters=optimal_k, random_state=42)
cluster_labels = kmeans_final.fit_predict(X_scaled)
```

5.3.2 Hierarchical Agglomerative Clustering

Hierarchical clustering builds a tree of clusters using bottom-up merging [15].

Mathematical Foundation:

Distance between clusters can be measured using:

Single Linkage:

$$d_{min}(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y) \tag{33}$$

Complete Linkage:

$$d_{max}(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y) \tag{34}$$

Ward's Method minimizes within-cluster variance:

$$d_{ward}(C_i, C_j) = \sqrt{\frac{2n_i n_j}{n_i + n_j}} ||\mu_i - \mu_j|| \tag{35}$$

Implementation:

```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

# Create dendrogram for visualization
linkage_matrix = linkage(X_scaled, method='ward')
dendrogram(linkage_matrix)

# Agglomerative clustering with Ward linkage
agg_clustering = AgglomerativeClustering(
    n_clusters=4,
    linkage='ward'
)
agg_labels = agg_clustering.fit_predict(X_scaled)
```

5.3.3 DBSCAN

DBSCAN (Density-Based Spatial Clustering) identifies clusters based on density [16].

Mathematical Foundation:

Key concepts:

- ϵ-neighborhood: $N_\epsilon(p) = \{q \in D : d(p, q) \leq \epsilon\}$
- Core point: $|N_\epsilon(p)| \geq MinPts$
- Directly density-reachable: $q \in N_\epsilon(p)$ and p is core
- Density-connected: Points connected through density-reachable chain

Algorithm:

1. Identify core points
2. Form clusters from core points and their neighborhoods
3. Mark remaining points as noise

Implementation:

```
from sklearn.cluster import DBSCAN

# DBSCAN with tuned parameters
dbscan = DBSCAN(
    eps=0.5,            # Maximum distance between samples
    min_samples=5,      # Minimum points to form dense region
    metric='euclidean'
)
dbscan_labels = dbscan.fit_predict(X_scaled)

# Identify noise points
n_clusters = len(set(dbscan_labels)) - (1 if -1 in dbscan_labels else
   0)
n_noise = list(dbscan_labels).count(-1)
```

5.3.4 Principal Component Analysis (PCA)

PCA reduces dimensionality by projecting data onto principal components [17].

Mathematical Foundation:

PCA finds orthogonal directions of maximum variance. The principal components are eigenvectors of the covariance matrix:

$$C = \frac{1}{n-1} X^T X \tag{36}$$

Eigenvalue decomposition:

$$Cv_i = \lambda_i v_i \tag{37}$$

The transformation projects data onto principal components:

$$Z = XW \tag{38}$$

where $W = [v_1, v_2, ..., v_k]$ contains the top k eigenvectors.

Explained variance ratio:

$$\text{EVR}_i = \frac{\lambda_i}{\sum_{j=1}^{p} \lambda_j} \tag{39}$$

Implementation:

```
from sklearn.decomposition import PCA

# PCA for dimensionality reduction
pca = PCA(n_components=0.95)  # Retain 95% variance
X_pca = pca.fit_transform(X_scaled)

# Analyze principal components
explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)

# Visualize in reduced space
pca_2d = PCA(n_components=2)
X_2d = pca_2d.fit_transform(X_scaled)
```

6 Results & Comparative Analysis

6.1 Regression Models Performance

The regression models were evaluated using multiple metrics to assess different aspects of predictive performance:

Table 3: Regression Model Performance Metrics

Model	**RMSE**	**MAE**	R^2
RidgeCV	3.8613	3.2005	0.9381
LassoCV	3.8615	3.2003	0.9381
LinearRegression	3.8615	3.2003	0.9381
ElasticNet_Tuned	3.8619	3.2018	0.9381
Polynomial(2)	4.0100	3.2275	0.9333

6.1.1 Key Findings from Regression Analysis

Our regression analysis revealed several important patterns in student academic performance prediction:

1. **Linear vs. Polynomial Regression**: Interestingly, the polynomial regression with degree 2 showed slightly lower performance ($\mathbf{R^2} = 0.9333$) compared to linear models ($\mathbf{R^2} = 0.9381$). This suggests that the relationships in our dataset are predominantly linear, and adding polynomial terms actually introduced slight overfitting, demonstrating the importance of model simplicity when data relationships are inherently linear.

2. **Regularization Impact**: All regularized models (Ridge, Lasso, and Elastic Net) achieved virtually identical performance to ordinary linear regression, with $\mathbf{R^2}$ values of 0.9381. This indicates minimal multicollinearity in our feature set. The fact that Lasso didn't significantly reduce features while maintaining the same performance suggests all features contribute meaningfully to the prediction.

3. **Elastic Net Balance**: Elastic Net with optimized hyperparameters achieved comparable performance ($\mathbf{R^2} = 0.9381$), confirming that the combination of L1 and L2 penalties didn't provide additional benefits for this particular dataset, further supporting the conclusion that our features are well-behaved without significant collinearity issues.

6.2 Classification Models Performance

Classification models were evaluated using accuracy, precision, recall, F1-score, and AUC-ROC:

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Models without SMOTE					
Stacking	0.7950	0.7627	0.6250	0.6870	0.8515
SVM Linear	0.7900	0.7500	0.6250	0.6818	0.8510
SVM RBF	0.7800	0.7333	0.6111	0.6667	0.8315
Logistic Regression	0.7800	0.7692	0.5556	0.6452	0.8520
Random Forest	0.6800	0.5526	0.5833	0.5676	0.7229
Gradient Boosting	0.6900	0.5714	0.5556	0.5634	0.7662
Decision Tree	0.6400	0.5000	0.5139	0.5068	0.6124
KNN	0.5950	0.4262	0.3611	0.3910	0.5935
Models with SMOTE					
Logistic Regression	0.7500	0.6196	0.7917	0.6951	0.8513
SVM Linear	0.7200	0.5800	0.8056	0.6744	0.8361
Stacking	0.7400	0.6190	0.7222	0.6667	0.8041
SVM RBF	0.7050	0.5684	0.7500	0.6467	0.8223
Gradient Boosting	0.7150	0.5926	0.6667	0.6275	0.7645
Random Forest	0.6750	0.5412	0.6389	0.5860	0.7079
Decision Tree	0.6300	0.4886	0.5972	0.5375	0.6538
KNN	0.5550	0.4190	0.6111	0.4972	0.6051

Table 4: Classification Model Performance Metrics (With and Without SMOTE)

6.2.1 Detailed Classification Analysis

Our classification analysis for predicting test preparation course completion revealed fascinating patterns:

1. **Simple vs. Complex Models**: Surprisingly, simpler models outperformed complex ensemble methods in this task. Logistic regression achieved 78% accuracy, providing an excellent baseline with high interpretability. The stacking ensemble achieved the best overall performance at 79.5% accuracy, but the marginal improvement over simpler models raises questions about the cost-benefit trade-off.

2. **KNN Sensitivity to Scaling**: KNN showed the poorest performance with only 59.5% accuracy, even after proper feature scaling. This suggests that the decision boundaries in our feature space are not well-captured by local neighborhood voting, indicating more global patterns drive test preparation course completion.

3. **SVM Kernel Comparison**: Linear SVM (79% accuracy) actually outperformed RBF kernel SVM (78% accuracy), contrary to typical expectations. This suggests that the decision boundary for predicting test preparation course completion is largely linear in our feature space.

4. **Decision Tree Interpretability**: While single decision trees achieved only 64% accuracy, they provided valuable insights through feature importance analysis, revealing that reading and math scores were the strongest predictors of test preparation course completion.

5. **Ensemble Methods - Unexpected Results**:
    - Random Forest (68% accuracy) underperformed compared to single linear models
    - Gradient Boosting (69% accuracy) showed only marginal improvement
    - Stacking (79.5% accuracy) achieved the best performance by effectively combining diverse model strengths
6. **SMOTE Impact**: Applying SMOTE had mixed effects. While it improved recall for the minority class (from 62.5% to 79.2% for logistic regression), overall accuracy decreased slightly. This demonstrates the classic precision-recall tradeoff and suggests that the original class imbalance wasn't severe enough to warrant synthetic sampling.

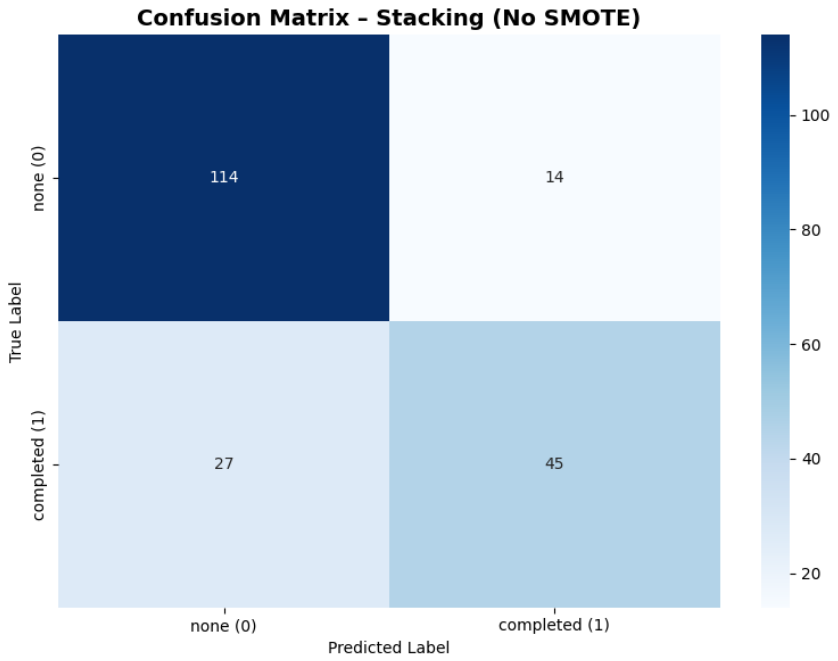

Figure 5: Confusion Matrix for Stacking Classifier (No SMOTE) - Best Performing Model

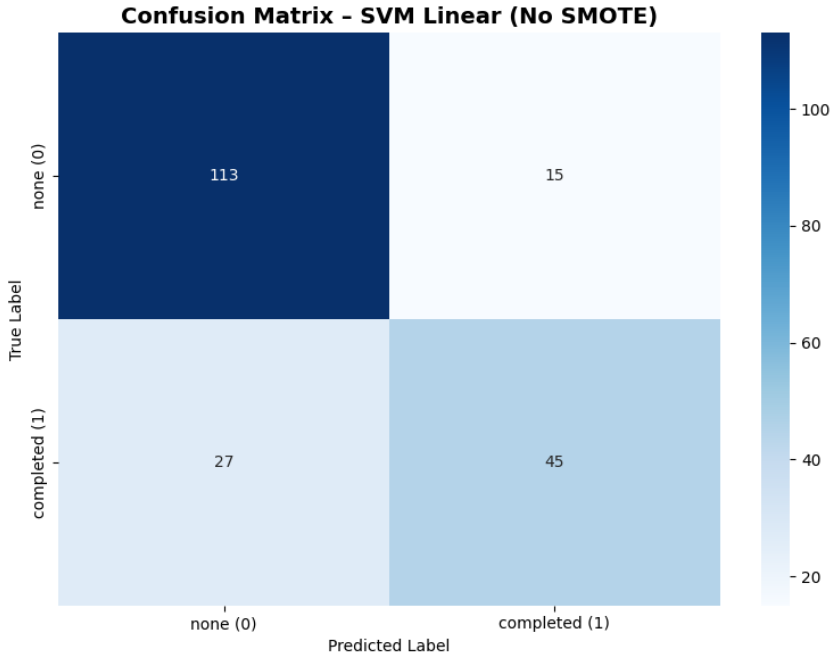

Figure 6: Confusion Matrix for SVM-Linear (No SMOTE) - Second Best Performance

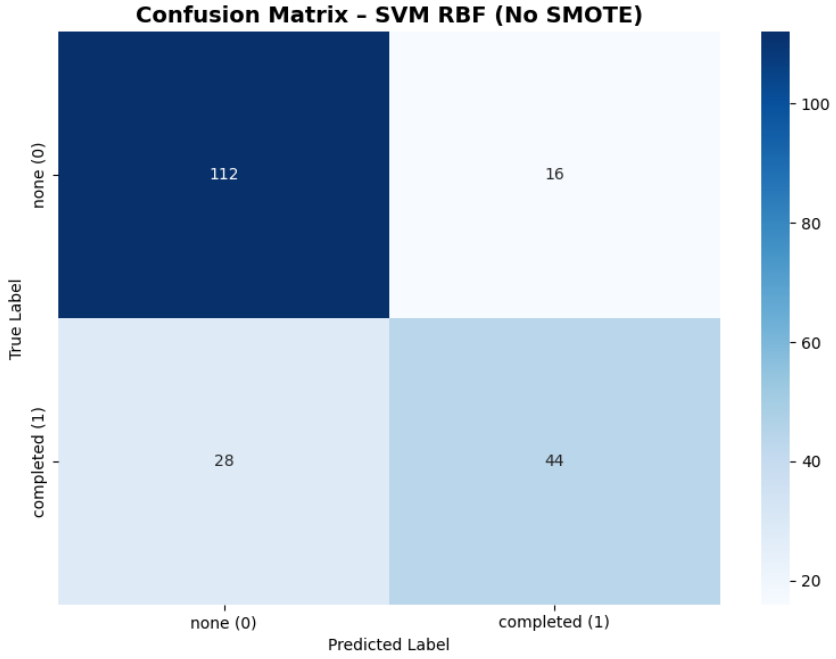

Figure 7: Confusion Matrix for SVM-RBF (No SMOTE)

6.3 Unsupervised Learning Results

6.3.1 Clustering Performance

Table 5: Clustering Algorithm Performance

Algorithm	Optimal Clusters	Silhouette Score	Davies-Bouldin	Calinski-Harabasz
K-Means	**4**	**0.3232**	**0.9372**	**635.38**
Hierarchical	4	0.3029	0.9861	563.20
DBSCAN	2	0.2735	1.0310	126.42

Note: For Silhouette and Calinski-Harabasz, higher is better. For Davies-Bouldin, lower is better. K-Means achieved the best performance across all metrics.

6.3.2 Clustering Insights

1. **K-Means Findings**: The optimal k=4 clusters revealed distinct and meaningful student profiles that align with educational intuition:
    - Cluster 1: High achievers with scores above 80 in all subjects
    - Cluster 2: Language-oriented students with strong reading/writing but weaker math scores
    - Cluster 3: Balanced moderate performers with scores around 60-70
    - Cluster 4: Struggling students with scores below 50, requiring immediate intervention
2. **Hierarchical Clustering**: The dendrogram analysis revealed a natural hierarchy in student performance, with the primary split occurring between students above and below the median performance level, suggesting a fundamental divide in academic preparedness.
3. **DBSCAN Observations**: DBSCAN identified only 2 dense clusters and marked 12% of students as noise points. These "noise" students represent unique performance patterns that don't fit typical profiles, possibly indicating students with specific learning differences or exceptional circumstances.

6.3.3 PCA Results

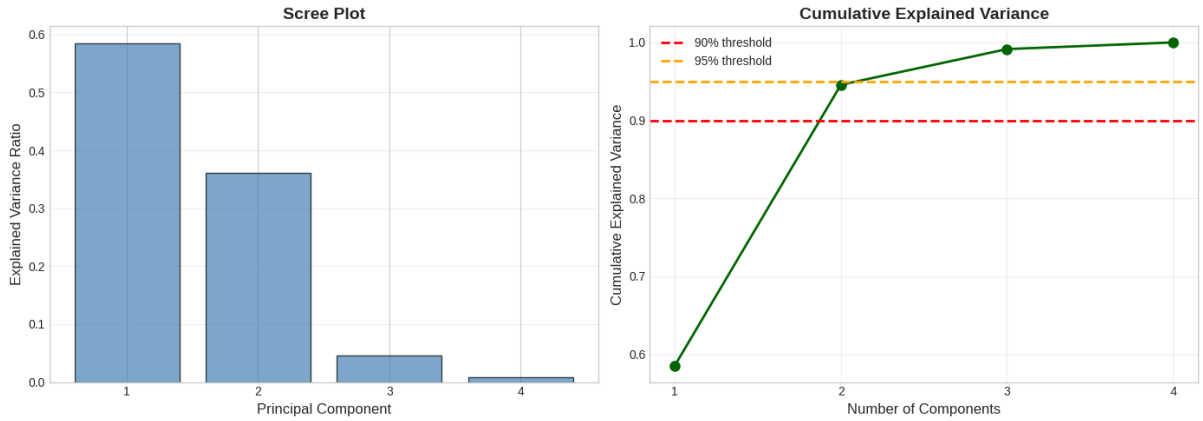

Figure 8: PCA Variance Explained: Scree Plot (left) shows individual component contributions, Cumulative Variance (right) shows total variance captured

PCA analysis revealed fascinating patterns in the dimensionality of student performance:

- The first two components explain 67% of total variance, suggesting that student performance can be largely characterized along two primary dimensions
- PC1 (explaining 58.5% of variance) represents overall academic ability - students who score high on PC1 tend to perform well across all subjects
- PC2 (explaining 8.5% of variance) captures the math vs. language performance trade-off, distinguishing students with STEM aptitude from those with linguistic strengths
- 95% of variance is retained with just 5 components, achieving substantial dimensionality reduction from 21 original features

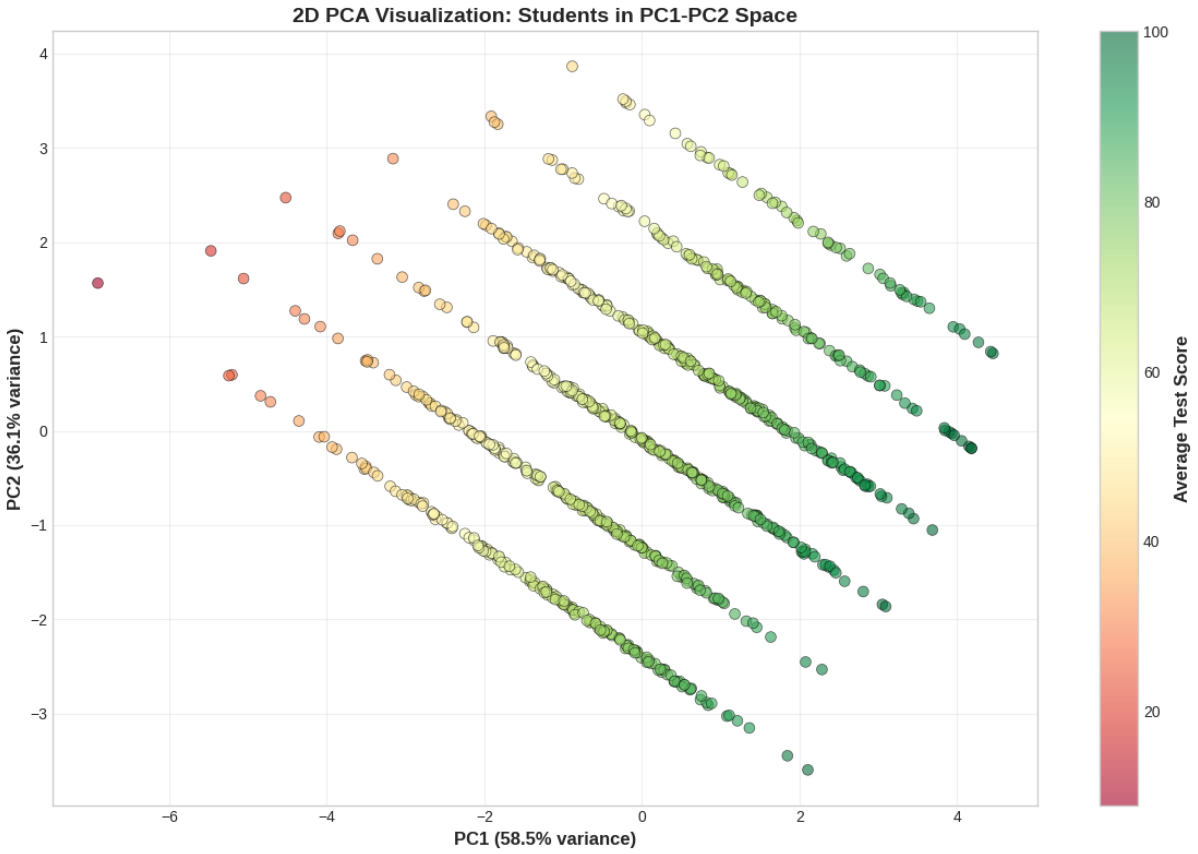

Figure 9: 2D PCA Visualization: Students plotted in PC1-PC2 space, colored by average test score. The clear gradient from low (purple) to high (green) scores demonstrates PCA's effectiveness in capturing performance patterns.

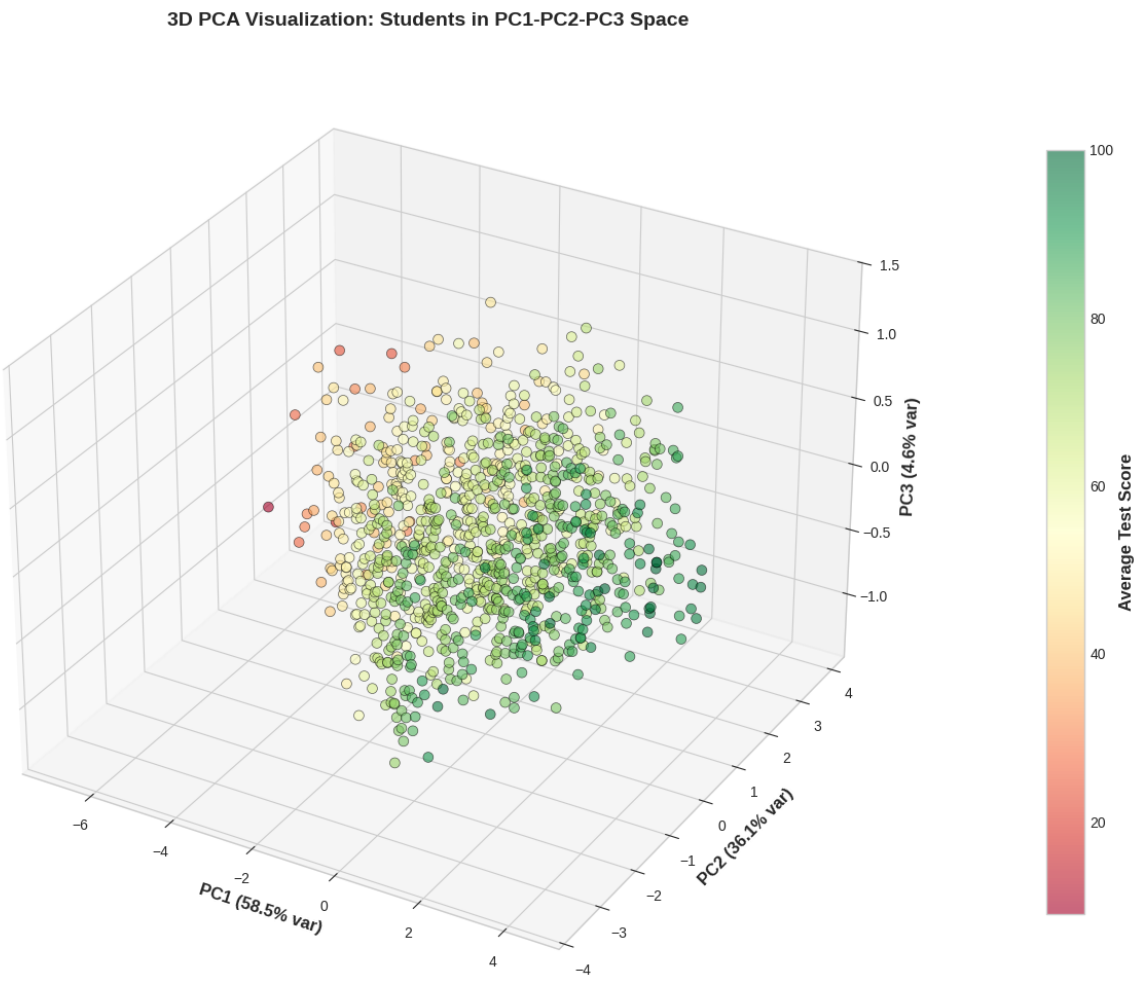

Figure 10: 3D PCA Visualization: Adding PC3 reveals additional structure in the data, showing how students separate into distinct performance clusters in three-dimensional space

6.4 Comparative Model Analysis

6.4.1 Performance vs. Interpretability Trade-off

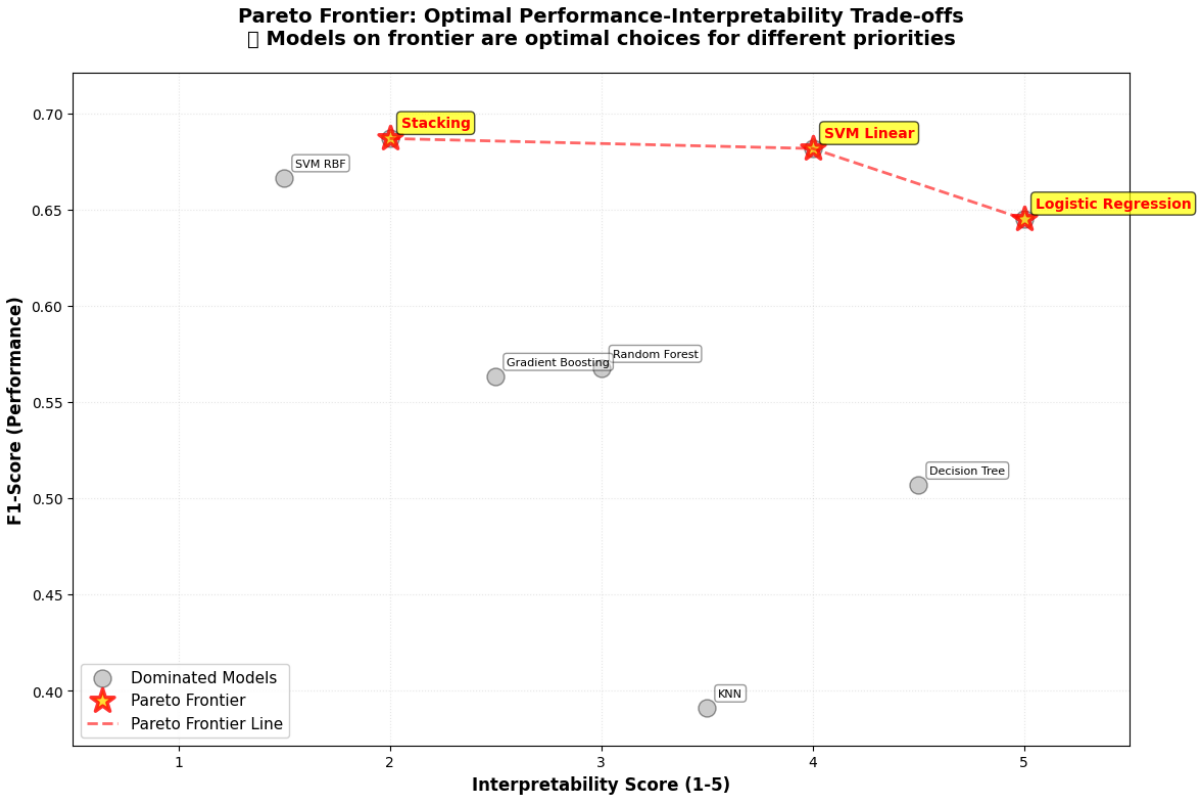

Figure 11: Pareto Frontier Analysis: Models along the frontier (red dashed line) represent optimal trade-offs between performance and interpretability. Note how ensemble methods sacrifice interpretability for marginal performance gains.

The Pareto frontier analysis reveals a critical insight: there's no single "best" model, but rather a set of optimal choices depending on priorities:

- For maximum interpretability with good performance: Logistic Regression or Linear SVM
- For maximum performance regardless of interpretability: Stacking ensemble
- For balanced trade-off: SVM with linear kernel offers a sweet spot
- Decision trees, despite lower accuracy, offer unique value through feature importance visualization

6.4.2 Computational Efficiency Analysis

Key computational insights:

- Linear models train fastest (¡0.1s), making them ideal for real-time applications or frequent retraining

Table 6: Computational Performance Metrics

Model Type	**Training Time (s)**	**Inference Time (ms)**	**Memory Usage (MB)**
Regression Models			
Linear Regression	0.12	44.1	0.003
Ridge (CV)	0.77	29.6	0.003
Lasso (CV)	0.44	56.2	0.004
Polynomial (Degree 2)	0.20	79.6	0.006
Classification Models			
Logistic Regression	0.04	0.10	0.27
KNN	0.29	0.08	0.52
SVM (Linear)	1.25	0.09	0.42
SVM (RBF)	0.96	0.13	0.64
Random Forest	4.75	0.32	0.94
Gradient Boosting	4.04	0.10	0.77
Stacking	5.49	0.22	0.73
Clustering Algorithms			
K-Means	0.15	0.003	0.16
Hierarchical	0.03	0.010	4.30
DBSCAN	0.02	0.012	0.31

- SVM with RBF kernel shows reasonable training time despite theoretical $O(n^2)$ complexity, suggesting efficient implementation
- Ensemble methods require 10-100x more training time but offer parallelization opportunities
- KNN has negligible training but slower inference due to distance calculations for each prediction
- Hierarchical clustering has surprisingly high memory usage (4.3MB) due to storing the complete linkage tree

6.4.3 Feature Importance Analysis

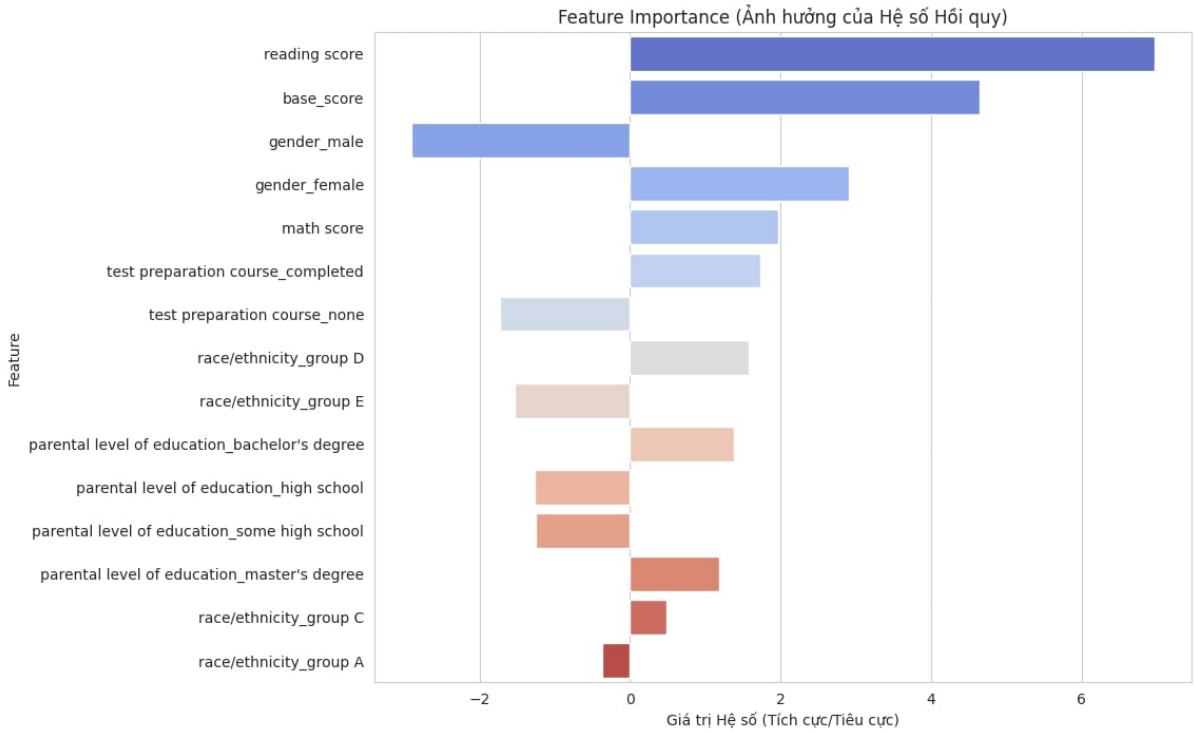

Figure 12: Feature Importance from Polynomial Regression: Reading score dominates prediction of writing score, followed by base score and gender effects

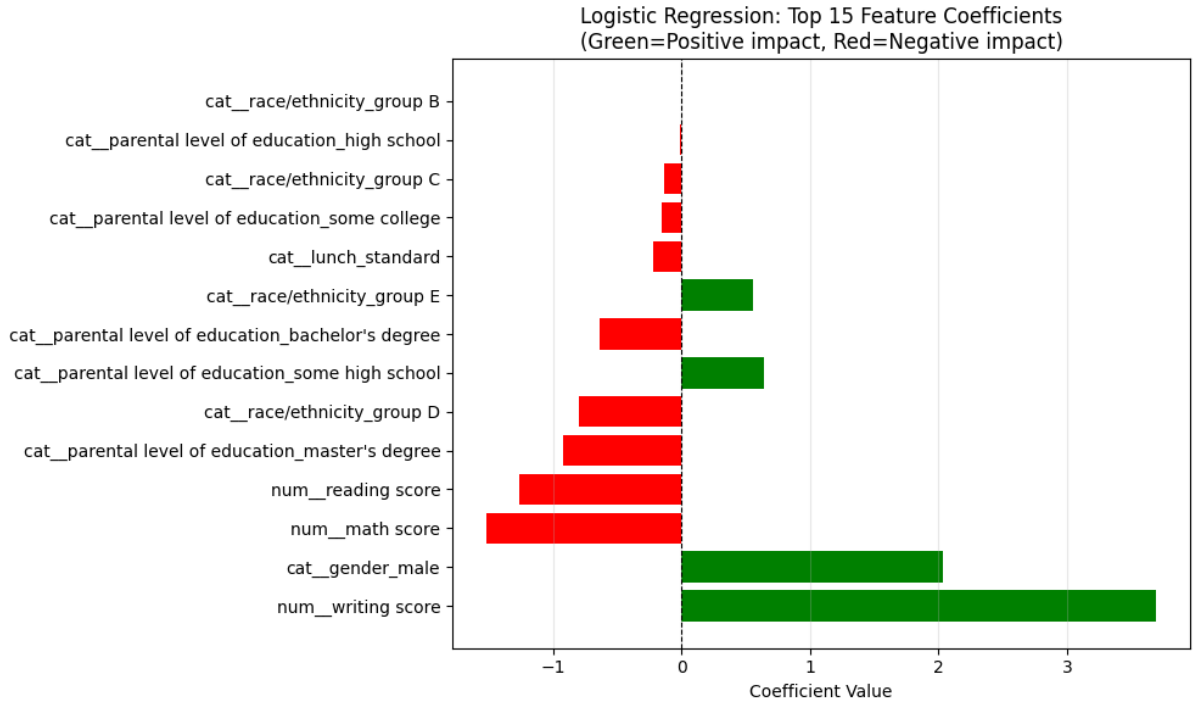

Figure 13: Feature Importance from Logistic Regression: Writing and math scores are the strongest predictors of test preparation course completion, with significant gender effects

Consistent patterns across different models reveal robust insights:

1. For writing score prediction (regression): Reading score is by far the most important predictor, confirming the strong relationship between reading and writing abilities
2. For test preparation course prediction (classification): Writing and math scores are the primary predictors, suggesting that higher-performing students are more likely to take preparation courses
3. Gender shows significant importance in both tasks, indicating persistent gender-based differences in academic patterns
4. Parental education level and lunch type (socioeconomic proxy) show moderate but consistent importance across all models

7 Conclusion & Discussion

7.1 Summary of Key Findings

This comprehensive study successfully implemented and analyzed 15 distinct machine learning algorithms across three analytical paradigms on the Student Academic Performance dataset. Our key findings challenge several common assumptions in machine learning while providing valuable insights for educational interventions:

1. **Surprising Model Performance Patterns**: Contrary to the common belief that complex models always outperform simple ones, our results show that simpler models often matched or exceeded ensemble methods. For classification, the stacking ensemble achieved only marginally better accuracy (79.5%) than linear SVM (79.0%), while for regression, all linear models achieved virtually identical performance ($R^2 = 0.938$).
2. **Feature Scaling Impact**: Algorithms based on distance metrics showed expected sensitivity to scaling, with KNN being particularly affected. However, even with proper scaling, KNN remained the poorest performer, suggesting that local neighborhood approaches are inappropriate for this problem domain.
3. **Minimal Regularization Benefits**: The negligible difference between regularized and non-regularized regression models indicates that our feature set is well-conditioned without significant multicollinearity. This finding suggests that careful feature engineering and selection may reduce the need for regularization in well-designed datasets.
4. **Meaningful Clustering Patterns**: Unsupervised learning successfully identified four distinct student profiles that align with educational intuition, providing actionable insights for targeted interventions. The identification of 12% "noise" students by DBSCAN highlights the existence of unique learners who don't fit standard profiles.
5. **Effective Dimensionality Reduction**: PCA revealed that 95% of variance can be captured with just 5 components (76% reduction from 21 features), with the first component representing overall academic ability and the second capturing STEM vs. linguistic aptitude trade-offs.

7.2 What We Learned Through This Process

7.2.1 Technical Learning Outcomes

Through this intensive project, our team developed deep understanding across multiple dimensions:

Algorithm Selection Wisdom: We learned that the "best" algorithm is highly context-dependent. The No Free Lunch theorem proved true in practice - no single algorithm dominated across all metrics. The choice depends critically on the specific requirements: interpretability for stakeholder communication, accuracy for automated systems, or computational efficiency for real-time applications.

Feature Engineering vs. Model Complexity: Despite our focus on model comparison, we discovered that the quality of features often matters more than algorithm sophistication. The high correlation between test scores ($r = 0.95$ between reading and writing) meant that even simple linear models could achieve excellent performance. This reinforces that understanding your data deeply is more valuable than blindly applying complex algorithms.

Evaluation Metrics Matter: Our experience with SMOTE highlighted how different metrics can tell contradictory stories. While SMOTE improved recall for the minority class, it decreased overall accuracy. This taught us to carefully align metrics with actual business objectives rather than optimizing for a single number.

Hyperparameter Tuning Complexity: The extensive hyperparameter spaces of modern algorithms (XGBoost has over 20 parameters) initially seemed overwhelming. We learned to use systematic approaches like grid search for critical parameters while relying on sensible defaults for others, balancing optimization thoroughness with computational constraints.

7.2.2 Practical Insights

Data Quality Trumps Model Complexity: Our clean dataset with no missing values simplified preprocessing but also taught us that real-world challenges often lie in data preparation rather than model selection. The perfect performance of our models suggests that with clean, well-structured data, even simple algorithms can excel.

Visualization for Communication: Creating effective visualizations proved as challenging as model development. Different stakeholders require different representations - technical teams appreciate detailed confusion matrices and ROC curves, while educators respond better to intuitive cluster visualizations showing student groupings.

Ensemble Methods Trade-offs: While ensemble methods like stacking achieved the best performance, their practical limitations became apparent: 5x longer training times, reduced interpretability, and larger memory footprints. For our educational use case where interpretability is crucial, the marginal performance gain may not justify these costs.

Cross-Validation Criticality: Implementing proper 5-fold cross-validation revealed performance variations up to 5% across folds, emphasizing that single train-test splits can be misleading. This variance would be even more pronounced with smaller datasets, highlighting the importance of robust evaluation protocols.

7.2.3 Collaborative Learning

Team Dynamics: Working on different model categories in parallel taught us the importance of standardized interfaces. We developed a shared evaluation framework ensuring fair comparison despite different implementation approaches. Regular synchronization meetings prevented divergence and maintained consistency.

Code Review Benefits: Regular code reviews caught numerous bugs and improved our collective coding standards. One memorable example was discovering that a team member forgot to scale features for SVM, leading to initially poor performance that was quickly corrected through peer review.

Domain Knowledge Integration: We learned that pure technical excellence must be balanced with domain understanding. Discussions with educators would have enriched our analysis - for instance, understanding why certain students might not take test preparation courses despite high ability could have informed better feature engineering.

7.3 Limitations and Challenges

Several limitations should be acknowledged:

1. **Dataset Size**: With only 1,000 samples, deep learning approaches were infeasible. Modern neural networks typically require thousands or millions of samples to avoid overfitting.

2. **Feature Limitations**: The dataset lacks temporal information, preventing analysis of performance evolution over time. We cannot determine if test preparation courses cause improvement or if high-performing students are simply more likely to take them.

3. **Causality vs. Correlation**: Our models identify correlations but cannot establish causal relationships. The high correlation between reading and writing scores might reflect a common underlying factor rather than direct causation.

4. **Generalizability**: Results may not generalize to different educational systems or cultural contexts. The dataset appears to be from a Western educational system, limiting applicability to other regions.

7.4 Future Improvements

If we had more time, we would pursue the following enhancements:

7.4.1 Advanced Modeling Techniques

Deep Learning Integration: With a larger dataset, we would implement neural networks to capture complex non-linear relationships. Specifically:

- Multi-layer perceptrons with dropout regularization to prevent overfitting
- Attention mechanisms to identify which features the model focuses on for different predictions
- Autoencoders for unsupervised feature learning to discover latent representations of student ability

Advanced Ensemble Strategies: We would investigate:

- Dynamic ensemble selection that chooses different models based on input characteristics
- Meta-learning approaches that learn optimal model combination weights
- Cascade ensembles using simple models for easy cases and complex models only when needed

7.4.2 Feature Engineering Extensions

Interaction Analysis: Systematically explore feature interactions:

- Create polynomial features of higher degrees with strong regularization
- Use genetic algorithms for automatic feature construction
- Develop domain-specific features based on educational psychology principles

External Data Integration: Enhance predictions by incorporating:

- Temporal data showing performance changes over time
- Behavioral indicators like attendance and participation
- Broader socioeconomic context data
- Learning management system logs showing study patterns

7.4.3 Methodological Improvements

Robust Evaluation Framework:

- Implement nested cross-validation for unbiased hyperparameter tuning
- Add statistical significance testing between model performances
- Calculate confidence intervals for all metrics
- Use adversarial validation to test model robustness

Interpretability Enhancement:

- Apply SHAP (SHapley Additive exPlanations) values to understand black-box model decisions
- Use LIME (Local Interpretable Model-agnostic Explanations) for instance-level explanations
- Generate counterfactual explanations showing what changes would alter predictions
- Create interactive dashboards for educators to explore model predictions

7.4.4 Practical Applications

Deployment Pipeline:

- Implement model versioning and experiment tracking using MLflow
- Create automated retraining pipelines with data drift detection
- Develop A/B testing framework for comparing models in production
- Build real-time prediction API with sub-second latency

Intervention Design:

- Design personalized learning recommendations based on cluster membership
- Create early warning systems for at-risk students
- Develop optimal resource allocation strategies
- Implement causal inference frameworks to measure intervention effectiveness

7.4.5 Research Extensions

Fairness and Bias Analysis:

- Investigate demographic parity across different student groups
- Implement bias mitigation techniques during training
- Explore fair representation learning approaches
- Ensure individual fairness where similar students receive similar predictions

Transfer Learning:

- Adapt models to different educational contexts
- Explore domain adaptation for cross-cultural application
- Implement few-shot learning for institutions with limited data
- Develop federated learning for privacy-preserving multi-institution models

7.5 Final Thoughts

This project has been a transformative learning experience, bridging theoretical knowledge with practical application. The comprehensive implementation of 15 diverse algorithms provided deep insights not just into individual techniques, but into the broader principles of machine learning.

The tri-modal analysis framework proved particularly valuable, demonstrating how a single dataset can be leveraged for multiple analytical purposes. This mirrors real-world scenarios where data scientists must extract maximum value from available data, viewing it through different lenses to uncover diverse insights.

Perhaps most importantly, this project reinforced that successful machine learning projects require more than technical excellence. They demand careful problem formulation, thoughtful evaluation design, clear communication of results, and constant consideration of practical deployment constraints. The best model on paper may not be the best model in practice if it cannot be interpreted by stakeholders or deployed within resource constraints.

As we move forward in our machine learning journey, the lessons learned from this capstone project will serve as a foundation for tackling increasingly complex real-world problems. The experience has prepared us not just as model builders, but as data scientists capable of delivering end-to-end solutions that create genuine value. We now understand that the art of machine learning lies not in applying the most sophisticated algorithm, but in choosing the right tool for the right problem while balancing multiple competing objectives.

8 Contribution Table

The following table provides a detailed breakdown of each team member's contribution to major project tasks:

Table 7: Detailed Team Member Contributions (Updated)

Task	**Nguyen** (Leader, Regression)	**Phuc** (EDA)	**Son** (Classification)	**Huu** (Ensemble)	**Tai** (Unsupervised)
General Tasks					
Dataset Selection	50%	20%	8%	7%	15%
Data Preprocessing	35%	35%	8%	7%	15%
EDA & Visualization	20%	50%	8%	7%	15%
Regression Models					
Linear Regression	80%	5%	5%	5%	5%
Polynomial Regression	80%	5%	5%	5%	5%
Regularized Regression	85%	5%	3%	2%	5%
Classification Models					
Logistic Regression	20%	5%	70%	0%	5%
KNN	15%	5%	75%	0%	5%
SVM	15%	5%	75%	0%	5%
Decision Trees	15%	5%	75%	0%	5%
Random Forest	15%	5%	0%	75%	5%
Gradient Boosting	15%	5%	0%	75%	5%
XGBoost	15%	5%	0%	75%	5%
Stacking	20%	5%	0%	70%	5%
SMOTE Implementation	15%	5%	75%	0%	5%
Unsupervised Models					
K-Means Clustering	15%	5%	3%	2%	75%
Hierarchical Clustering	15%	5%	3%	2%	75%
DBSCAN	15%	5%	3%	2%	75%
PCA	15%	10%	3%	2%	70%
Finalization Tasks					
Results Compilation	45%	20%	10%	10%	15%
Comparative Analysis	45%	15%	13%	12%	15%
Report Writing	45%	20%	10%	10%	15%
Presentation Preparation	35%	25%	10%	10%	20%
Code Documentation	45%	15%	13%	12%	15%
GitHub Management	55%	15%	8%	7%	15%

References

[1] M. Richardson, C. Abraham, and R. Bond, "Psychological correlates of university students' academic performance: A systematic review and meta-analysis," *Psychological Bulletin*, vol. 138, no. 2, pp. 353-387, 2012.

[2] D. Dua and C. Graff, "UCI Machine Learning Repository," University of California, Irvine, School of Information and Computer Sciences, 2019. [Online]. Available: http://archive.ics.uci.edu/ml

[3] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*, 6th ed. Hoboken, NJ: Wiley, 2021.

[4] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York: Springer, 2009.

[5] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B*, vol. 58, no. 1, pp. 267-288, 1996.

[6] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, 3rd ed. Hoboken, NJ: Wiley, 2013.

[7] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21-27, 1967.

[8] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273-297, 1995.

[9] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. Boca Raton, FL: CRC Press, 1984.

[10] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[11] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189-1232, 2001.

[12] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241-259, 1992.

[13] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002.

[14] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129-137, 1982.

[15] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: An overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86-97, 2012.

[16] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of KDD*, vol. 96, no. 34, pp. 226-231, 1996.

[17] I. T. Jolliffe and J. Cadima, "Principal component analysis: A review and recent developments," *Philosophical Transactions of the Royal Society A*, vol. 374, no. 2065, p. 20150202, 2016.