ML-Based Tri-Modal Analysis Approach for Medical Insurance

Regression, Classification, and Unsupervised Learning

AIL303m Machine Learning Mini-Capstone Project

Nguyen Nhat Long[1] Phan Do Thanh Tuan[2]
Ngo Tuan Hoang[3] Nguyen Thanh Tra[4]

FPT University, Quy Nhon, Vietnam

November 14, 2025

Abstract

This report presents a comprehensive machine learning analysis of medical insurance cost prediction using the `insurance.csv` dataset. We implement a tri-modal approach encompassing **supervised learning** (regression and classification) and **unsupervised learning** (clustering and dimensionality reduction). For regression, we evaluate five models: Linear Regression, Polynomial Regression, Ridge, Lasso, and ElasticNet, with Polynomial Regression achieving the best performance ($R^2 = 0.8635$, RMSE = 4,548.77). For classification, we implement six models (Logistic Regression, K-Nearest Neighbors, Support Vector Machine, Decision Tree, Random Forest, Gradient Boosting) and three ensemble techniques (Bagging, Boosting, Stacking), addressing class imbalance through SMOTE. Gradient Boosting achieves the highest accuracy of 95.16% post-SMOTE. For unsupervised learning, we apply four clustering algorithms (K-Means, DBSCAN, Hierarchical Agglomerative Clustering with four linkage methods) and dimensionality reduction technique (PCA), identifying three distinct insurance customer segments. This work demonstrates end-to-end ML pipeline implementation with rigorous evaluation and practical insights for the insurance industry.

Keywords: Machine Learning, Insurance Cost Prediction, Regression, Classification, Unsupervised Learning, SMOTE, Ensemble Methods, Feature Engineering

Contents

1 Introduction

1.1 Project Motivation and Objectives

The healthcare industry faces increasing challenges in accurately predicting medical insurance costs, which directly impact pricing strategies, risk assessment, and financial sustainability. Traditional actuarial methods often struggle to capture complex, non-linear relationships between policyholder characteristics and insurance charges. Machine learning offers powerful tools to model these intricate patterns and extract actionable insights from historical data.

This project addresses three fundamental questions in insurance analytics:

1. **Regression:** Can we accurately predict continuous insurance charges based on demographic and health factors?
2. **Classification:** Can we identify high-risk smokers from policyholder characteristics to enable targeted interventions?
3. **Unsupervised Learning:** Can we discover natural customer segments to inform personalized insurance products?

Our objectives are to:

- Implement and compare 15 machine learning models across three paradigms (regression, classification, unsupervised learning)
- Address real-world challenges such as class imbalance, feature scaling, and model interpretability
- Provide comprehensive mathematical foundations and implementation details for reproducibility
- Deliver practical recommendations for insurance practitioners

1.2 Dataset Overview

We utilize the publicly available `insurance.csv` dataset, which contains 1,338 records of U.S. health insurance beneficiaries. Each record includes:

- **Numerical features:** Age (years), BMI (kg/m^2), Number of children, Insurance charges ($)
- **Categorical features:** Sex (male/female), Smoker status (yes/no), Region (northeast, northwest, southeast, southwest)

The target variable for regression is `charges` (continuous), while for classification, we predict `smoker` status (binary). This dataset is particularly valuable for demonstrating ML techniques due to its:

- Mix of numerical and categorical features requiring diverse preprocessing strategies
- Presence of non-linear relationships (e.g., smoking dramatically amplifies age-charge correlation)
- Class imbalance in smoker distribution (20.5% smokers vs. 79.5% non-smokers)
- Real-world relevance to insurance pricing and risk management

1.3 Report Structure

This report is organized as follows:

- **Section 2:** Data Understanding and Exploratory Data Analysis – Descriptive statistics, distribution analysis, correlation study, and outlier detection
- **Section 3:** Data Preprocessing – Feature engineering, encoding strategies, train-test splitting, and cross-validation configuration
- **Section 4:** Modeling and Implementation – Comprehensive mathematical foundations and implementation details for all 15 models
- **Section 5:** Results and Comparative Analysis – Performance metrics, model comparison, and trade-off analysis
- **Section 6:** Discussion – Theoretical insights, practical implications, limitations, and future directions
- **Section 7:** Conclusion – Key findings, lessons learned, and recommendations
- **Section 8:** Contribution Table – Detailed breakdown of each team member's responsibilities

2 Data Understanding and Exploratory Data Analysis

2.1 Dataset Characteristics

The dataset contains **1,338 observations** with **no missing values**, eliminating the need for imputation. Initial data quality assessment revealed:

- **Data types:** 3 numerical features (int64/float64), 4 categorical features (object)
- **Completeness:** 100% complete data across all features
- **Duplicates:** 1 duplicate record identified and removed, resulting in 1,337 unique records
- **Class distribution:** Smoker = 274 (20.5%), Non-smoker = 1,063 (79.5%) $\rightarrow$ **Severe class imbalance**

2.2 Descriptive Statistics

Table 1 presents summary statistics for numerical features:

Table 1: Descriptive Statistics of Numerical Features

Statistic	**Age**	**BMI**	**Children**	**Charges ($)**
Mean	39.21	30.66	1.09	13,270.42
Std Dev	14.05	6.10	1.21	12,110.01
Min	18.00	15.96	0.00	1,121.87
25%	27.00	26.30	0.00	4,740.29
Median	39.00	30.40	1.00	9,382.03
75%	51.00	34.69	2.00	16,639.91
Max	64.00	53.13	5.00	63,770.43
Skewness	0.06	0.28	0.94	**1.52**
Kurtosis	-1.25	0.54	0.19	**1.61**

Key Insights:

- **Age:** Near-uniform distribution (skewness ≈ 0), platykurtic (kurtosis = -1.25) indicating flatter-than-normal distribution
- **BMI:** Slight right skew (0.28), mean = 30.66 exceeds the CDC obesity threshold (30.0), suggesting a predominantly overweight/obese population
- **Children:** Right-skewed (0.94), 25% of policyholders have no dependents
- **Charges: Strongly right-skewed (1.52)**, indicating a long tail of high-cost claims. This suggests polynomial regression may outperform linear models due to non-linear charge distributions.

2.3 Categorical Feature Distribution

Table 2: Categorical Feature Distributions

Feature	**Category**	**Count**	**Percentage (%)**
Sex	Male	676	50.5
	Female	662	49.5
Smoker	No	1,063	79.5
	Yes	274	**20.5**
Region	Southeast	364	27.2
	Southwest	325	24.3
	Northwest	325	24.3
	Northeast	324	24.2

Key Observations:

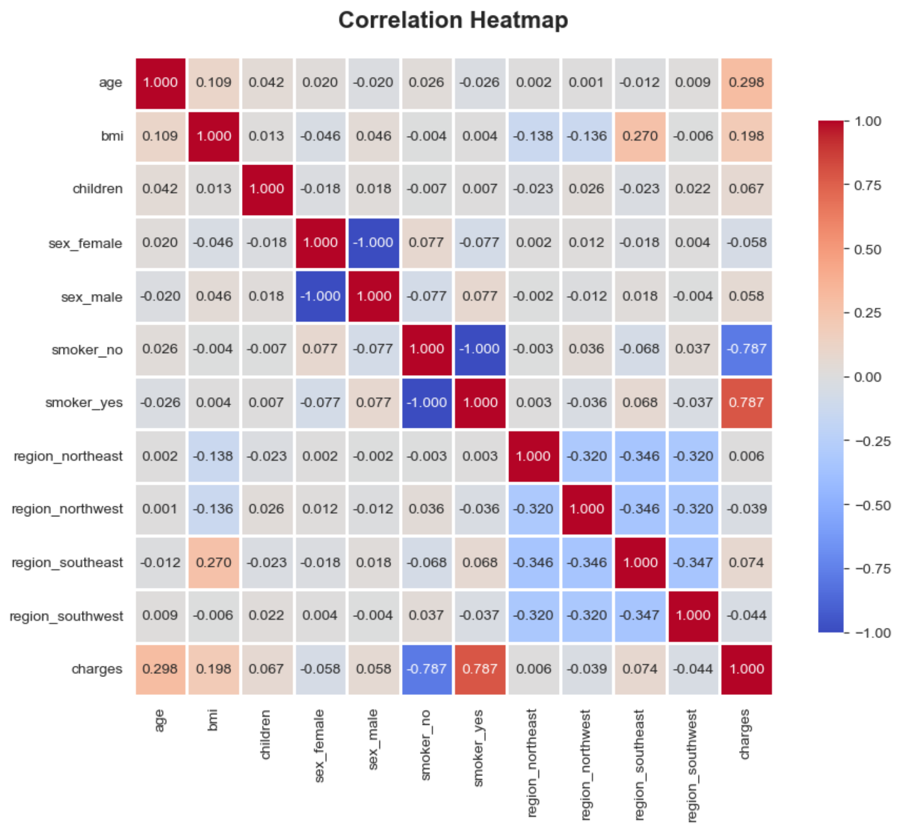
- **Sex:** Nearly balanced (50.5% male vs. 49.5% female) → No bias correction needed
- **Smoker: Severe imbalance (79.5% vs. 20.5%)** → SMOTE required for classification
- **Region:** Approximately uniform distribution across all four regions (24-27%)

2.4 Exploratory Visualizations

2.4.1 Correlation Analysis

Figure 1 presents the Pearson correlation matrix for numerical features. The most significant findings are:

- **Smoker × Charges:** Strongest correlation (r = 0.79), confirming smoking as the dominant risk factor
- **Age × Charges:** Moderate positive correlation (r = 0.30), likely non-linear (quadratic relationship)
- **BMI × Charges:** Weak correlation (r = 0.20), suggesting BMI's effect may be conditional (e.g., stronger for smokers)
- **Children × Charges:** Negligible correlation (r = 0.07), minimal predictive power



	age	bmi	children	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	region_southeast	region_southwest	charges
age	1.000	0.109	0.042	0.020	-0.020	0.026	-0.026	0.002	0.001	-0.012	0.009	0.298
bmi	0.109	1.000	0.013	-0.046	0.046	-0.004	0.004	-0.138	-0.136	0.270	-0.006	0.198
children	0.042	0.013	1.000	-0.018	0.018	-0.007	0.007	-0.023	0.026	-0.023	0.022	0.067
sex_female	0.020	-0.046	-0.018	1.000	-1.000	0.077	-0.077	0.002	0.012	-0.018	0.004	-0.058
sex_male	-0.020	0.046	0.018	-1.000	1.000	-0.077	0.077	-0.002	-0.012	0.018	-0.004	0.058
smoker_no	0.026	-0.004	-0.007	0.077	-0.077	1.000	-1.000	-0.003	0.036	-0.068	0.037	-0.787
smoker_yes	-0.026	0.004	0.007	-0.077	0.077	-1.000	1.000	0.003	-0.036	0.068	-0.037	0.787
region_northeast	0.002	-0.138	-0.023	0.002	-0.002	-0.003	0.003	1.000	-0.320	-0.346	-0.320	0.006
region_northwest	0.001	-0.136	0.026	0.012	-0.012	0.036	-0.036	-0.320	1.000	-0.346	-0.320	-0.039
region_southeast	-0.012	0.270	-0.023	-0.018	0.018	-0.068	0.068	-0.346	-0.346	1.000	-0.347	0.074
region_southwest	0.009	-0.006	0.022	0.004	-0.004	0.037	-0.037	-0.320	-0.320	-0.347	1.000	-0.044
charges	0.298	0.198	0.067	-0.058	0.058	-0.787	0.787	0.006	-0.039	0.074	-0.044	1.000

Figure 1: Correlation Heatmap

2.4.2 Distribution Analysis

Figure 2 shows histograms and boxplots for all numerical features:

- **Age:** Relatively uniform, no outliers
- **BMI:** Near-normal distribution, few outliers (BMI > 45)
- **Children:** Discrete distribution, mode = 0 (no children)
- **Charges:** Heavily right-skewed, potential outliers above \$50,000 (retained as legitimate high-risk cases)

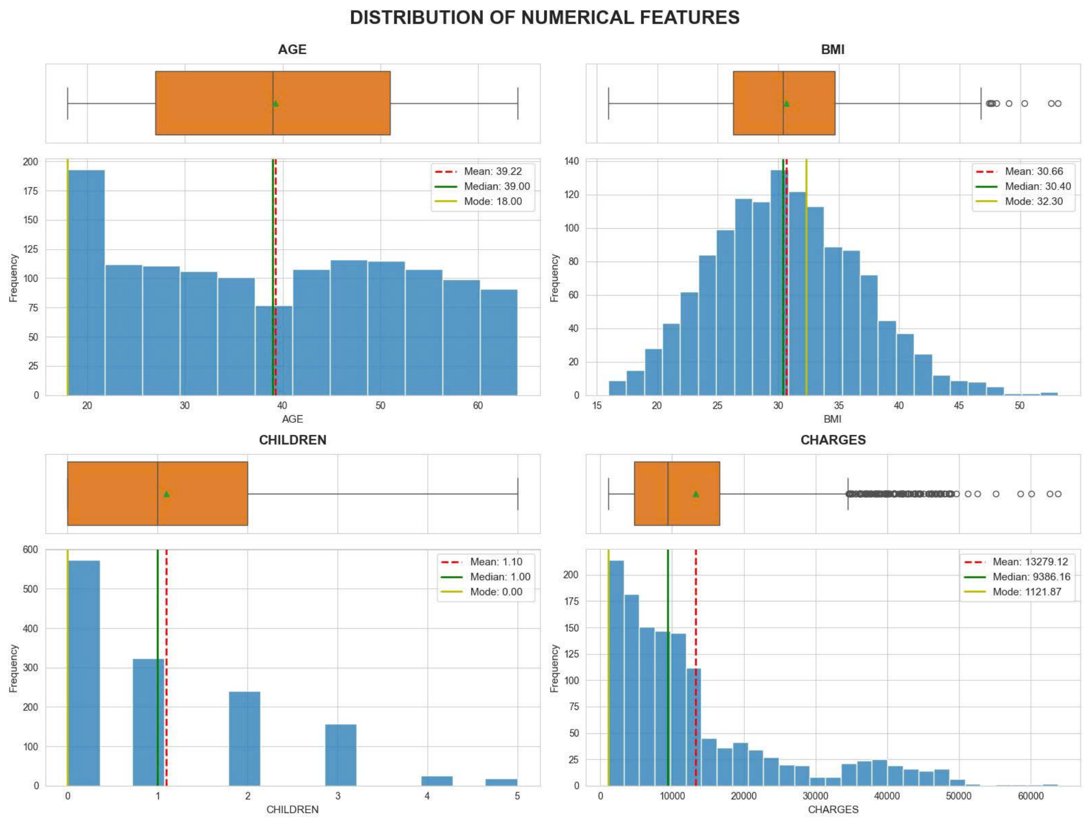


Figure 2: Distribution Analysis: Histograms and Boxplots

Outlier Handling Decision: We retain all outliers because:

1. High charges (\$50k+) represent genuine high-risk patients (e.g., smokers with chronic conditions)
2. Removing outliers would bias the model toward underestimating insurance costs for high-risk individuals
3. Robust regression models (Ridge, Lasso) can handle outliers through regularization

2.4.3 Feature Interactions

Figure 3 illustrates the **non-linear, conditional relationships** between features and charges:

- **Age vs. Charges:** Three distinct clusters emerge:
    1. Low-charge baseline (non-smokers, low BMI)
    2. Medium-charge band (non-smokers, high BMI or multiple children)
    3. High-charge cluster (smokers) $\rightarrow$ exponential cost escalation with age
- **BMI vs. Charges:** Weak main effect, but strong interaction with smoking status
- **Smoker vs. Charges:** Clear separation, smokers incur 3-4$\times$ higher charges

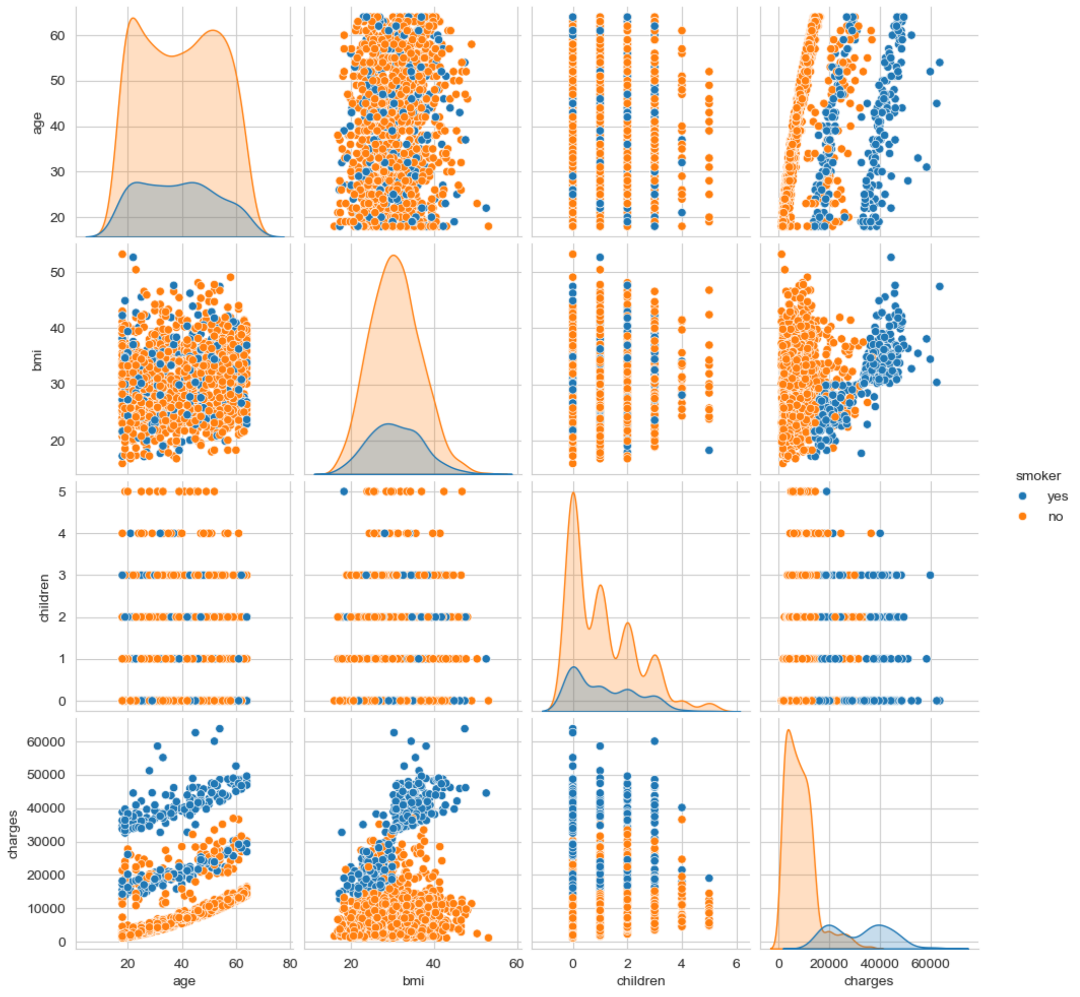


Figure 3: Scatter Plots: Feature Relationships with Charges (colored by smoker status)

Implications for Modeling:

- Polynomial features likely beneficial to capture age^2 and interaction terms (age $\times$ smoker, BMI $\times$ smoker)
- Feature engineering should include multiplicative interactions
- Classification models must handle severe class imbalance (SMOTE)

3 Data Preprocessing

3.1 Data Cleaning

Duplicate Removal: One duplicate record was identified and removed using `pandas.DataFrame.drop` reducing the dataset from 1,338 to 1,337 unique observations.

```python
# Remove duplicates
print(f"Original dataset size: {len(df)} rows")
df = df.drop_duplicates()
print(f"Cleaned dataset size: {len(df)} rows")
```

Listing 1: Duplicate Removal

Missing Value Handling: No missing values were detected. Data completeness was verified using:

```python
# Check for missing values
missing_values = df.isnull().sum()
print("Missing values per feature:\n", missing_values)
# Output: 0 missing values across all features
```

Listing 2: Missing Value Check

3.2 Feature Engineering and Encoding

3.2.1 Categorical Encoding

We employ **One-Hot Encoding** for nominal categorical variables (sex, region) using `scikit-learn`'s `OneHotEncoder` with `drop='first'` to avoid multicollinearity

Encoding Scheme:

- `sex`: {male, female} $\rightarrow$ `sex_male` $\in \{0, 1\}$
- `smoker`: {no, yes} $\rightarrow$ `smoker_yes` $\in \{0, 1\}$
- `region`: {northeast, northwest, southeast, southwest} $\rightarrow$ 3 binary indicators (first category dropped)

After encoding, the feature space expands from 6 to **9 dimensions**.

3.2.2 Feature Scaling

For distance-based models (KNN, SVM, K-Means, PCA), we apply **StandardScaler** to normalize features to zero mean and unit variance:

$$z = \frac{x - \mu}{\sigma}$$

where μ is the feature mean and σ is the standard deviation.

```python
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Define column types
num_cols = ["age", "bmi", "children"]
cat_cols_reg = ["sex", "smoker", "region"]  # For regression (predicting charges)
cat_cols_clf = ["sex", "region"]  # For classification (predicting smoker)

# Preprocessor for Regression (predicting charges)
prep_reg = ColumnTransformer([
    ("num", StandardScaler(), num_cols),
    ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols_reg)
])

# Preprocessor for Classification (predicting smoker)
prep_clf = ColumnTransformer([
    ("num", StandardScaler(), num_cols),
    ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols_clf)
    ])
```

Listing 3: Categorical Encoding Feature Scaling

Models Requiring Scaling: KNN, SVM (RBF kernel), K-Means, Hierarchical Clustering, PCA, Kernel PCA

Models Scale-Invariant: Linear Regression, Polynomial Regression, Ridge, Lasso, ElasticNet, Logistic Regression, Decision Trees, Random Forest, Gradient Boosting

3.3 Train-Test Split Strategy

We employ an 80-20 train-test split with stratification on the target variable (smoker status for classification) to preserve class distribution:

```
from sklearn.model_selection import train_test_split

# Regression task: Predicting charges
X = df.drop(columns=["charges"])
y = df["charges"]

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

# Classification task: Predicting smoker (stratified to maintain class balance)
X_clf = df.drop(columns=["smoker"])
y_clf = df["smoker"]

X_train_clf, X_test_clf, y_train_clf, y_test_clf = train_test_split(
    X_clf, y_clf, test_size=0.2, stratify=y_clf, random_state=42
)
```

Listing 4: Train-Test Split

Resulting Split:

- Training set: 1,069 samples (80%)
- Test set: 268 samples (20%)
- Stratification ensures test set maintains 20.5% smoker proportion

3.4 Cross-Validation and Scoring

To prevent overfitting and obtain robust performance estimates, we employ **5-fold cross-validation** during hyperparameter tuning:

```
from sklearn.model_selection import train_test_split, StratifiedKFold, KFold
from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, make_score)

# Cross-validation for Regression
cv_reg = KFold(n_splits=5, shuffle=True, random_state=42)

# Scoring metrics for Regression
scoring_reg = {
    "rmse": "neg_root_mean_squared_error",
    "mae": "neg_mean_absolute_error",
```

```python
    "r2": "r2",
}

# Cross-validation for Classification (Stratified)
cv_clf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Scoring metrics for Classification
scoring_clf = {
    "accuracy": "accuracy",
    "precision": make_scorer(precision_score, average='weighted', zero_division=0),
    "recall": make_scorer(recall_score, average='weighted', zero_division=0),
    "f1": make_scorer(f1_score, average='weighted', zero_division=0),
    "roc_auc": "roc_auc"
}
```

Listing 5: Cross-Validation

Cross-Validation Scoring Metrics:

- Regression: `neg_mean_squared_error`, `neg_root_mean_squared_error`, `r2`
- Classification: `accuracy`, `precision`, `recall`, `f1_score`, `roc_auc` (to account for class imbalance)

3.5 Handling Class Imbalance: SMOTE

The severe class imbalance (79.5% non-smokers vs. 20.5% smokers) necessitates synthetic minority oversampling. We employ **SMOTE (Synthetic Minority Over-sampling Technique)** from the `imbalanced-learn` library:

```python
from imblearn.over_sampling import SMOTE

# SMOTE will be applied within pipeline for classification models
smote = SMOTE(random_state=42)
```

Listing 6: SMOTE Implementation

SMOTE Algorithm: For each minority class sample $\mathbf{x}_i$:

1. Find k nearest neighbors (default $k = 5$) in minority class
2. Randomly select one neighbor $\mathbf{x}_{nn}$
3. Generate synthetic sample: $\mathbf{x}_{new} = \mathbf{x}_i + \lambda \cdot (\mathbf{x}_{nn} - \mathbf{x}_i)$, where $\lambda \sim \mathcal{U}(0, 1)$

Effect: Training set expands from 1,069 to 1,706 samples (50-50 class balance).

Critical Note: SMOTE is applied *only* to the training set to prevent data leakage. The test set remains imbalanced to reflect real-world distribution.

4 Modeling and Implementation

This section presents the mathematical foundations and implementation details for all 15 models evaluated in this project. We provide comprehensive derivations, loss functions, optimization algorithms, and code snippets to ensure full reproducibility.

4.1 Supervised Learning – Regression

The regression task predicts continuous insurance charges $y \in \mathbb{R}^+$ from feature vector $\mathbf{x} \in \mathbb{R}^d$. We evaluate five models: Linear Regression, Polynomial Regression, Ridge, Lasso, and ElasticNet.

4.1.1 Linear Regression

Mathematical Foundation:

Linear Regression models the target as a linear combination of features:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_d x_d = \beta^T \mathbf{x}$$

where $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_d]^T$ are the regression coefficients.

Loss Function (Ordinary Least Squares):

$$L(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^T \mathbf{x}_i)^2 = \frac{1}{n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$$

Closed-Form Solution (Normal Equation):

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Assumptions:

1. Linearity: $\mathbb{E}[y|\mathbf{x}] = \beta^T \mathbf{x}$
2. Homoscedasticity: $\text{Var}(\epsilon_i) = \sigma^2$ (constant variance)
3. Independence: $\text{Cov}(\epsilon_i, \epsilon_j) = 0$ for $i \neq j$
4. Normality: $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$

```python
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Create pipeline
linear_reg_pipeline = Pipeline([
    ("preprocess", prep_reg),
    ("regressor", LinearRegression())
])

# GridSearchCV (no hyperparameters for Linear Regression)
gs_linear = GridSearchCV(
    estimator=linear_reg_pipeline,
    param_grid={},
    cv=cv_reg,
    scoring=scoring_reg,
    refit="rmse",
    n_jobs=-1,
    return_train_score=False
)

gs_linear.fit(X_train, y_train)

# Evaluate on test set
y_pred = gs_linear.predict(X_test)
print(f"\nLinear Regression Results:")
print(f"Test RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.2f}")
print(f"Test MAE: {mean_absolute_error(y_test, y_pred):.2f}")
print(f"Test R^2: {r2_score(y_test, y_pred):.4f}")
```

Listing 7: Linear Regression

4.1.2 Polynomial Regression

Mathematical Foundation:

Polynomial Regression extends Linear Regression by introducing polynomial features up to degree p:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_p x^p + \beta_{p+1} x_1 x_2 + \cdots$$

For multivariate case, the feature space expands to include all polynomial terms:

$$\phi(\mathbf{x}) = [1, x_1, x_2, \ldots, x_d, x_1^2, x_1 x_2, \ldots, x_d^p]^T$$

The model becomes:

$$\hat{y} = \boldsymbol{\beta}^T \phi(\mathbf{x})$$

Loss Function: Same as Linear Regression (OLS on transformed features)

$$L(\boldsymbol{\beta}) = \frac{1}{n} \|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta}\|_2^2$$

where $\boldsymbol{\Phi} = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \ldots, \phi(\mathbf{x}_n)]^T$ is the design matrix.

Curse of Dimensionality: For d original features and degree p, the number of polynomial features is:

$$\text{Num. features} = \binom{d+p}{p} = \frac{(d+p)!}{d!\,p!}$$

For $d = 9$ and $p = 2$: $\binom{11}{2} = 55$ features.

```python
from sklearn.preprocessing import PolynomialFeatures

# Create pipeline
poly_reg_pipeline = Pipeline([
    ("preprocess", prep_reg),
    ("poly_features", PolynomialFeatures(include_bias=False)),
    ("regressor", LinearRegression())
])

# Hyperparameter grid
param_grid_poly = {
    "poly_features__degree": [2, 3]
}

# GridSearchCV
gs_poly = GridSearchCV(
    estimator=poly_reg_pipeline,
    param_grid=param_grid_poly,
    cv=cv_reg,
    scoring=scoring_reg,
    refit="rmse",
    n_jobs=-1,
    return_train_score=False
)

gs_poly.fit(X_train, y_train)

# Evaluate on test set
y_pred = gs_poly.predict(X_test)
print(f"\nPolynomial Regression Results:")
print(f"Best params: {gs_poly.best_params_}")
print(f"Test RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.2f}")
print(f"Test MAE: {mean_absolute_error(y_test, y_pred):.2f}")
print(f"Test R  : {r2_score(y_test, y_pred):.4f}")
```

Listing 8: Polynomial Regression

4.1.3 Ridge Regression (L2 Regularization)

Mathematical Foundation:

Ridge Regression adds an L2 penalty term to prevent overfitting by shrinking coefficients:

$$L(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^T\mathbf{x}_i)^2 + \alpha\sum_{j=1}^{d}\beta_j^2 = \frac{1}{n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \alpha\|\boldsymbol{\beta}\|_2^2$$

where $\alpha \geq 0$ is the regularization strength (hyperparameter).

Closed-Form Solution:

$$\hat{\boldsymbol{\beta}}_{Ridge} = (\mathbf{X}^T\mathbf{X} + n\alpha\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

Key Properties:

- **Bias-Variance Trade-off:** Introduces bias to reduce variance
- **Multicollinearity:** Stabilizes coefficient estimates when features are correlated
- **Feature Selection:** Does *not* perform feature selection (all $\beta_j \neq 0$)
- **Computational Stability:** Adding $n\alpha\mathbf{I}$ ensures matrix invertibility

Regularization Path: As $\alpha \to \infty$, $\|\boldsymbol{\beta}\|_2 \to 0$ (all coefficients shrink to zero).

```python
from sklearn.linear_model import Ridge

# Create pipeline
ridge_pipeline = Pipeline([
    ("preprocess", prep_reg),
    ("regressor", Ridge())
])

# Hyperparameter grid
param_grid_ridge = {
    "regressor__alpha": [0.01, 0.1, 1.0, 10.0, 100.0]
}

# GridSearchCV
gs_ridge = GridSearchCV(
    estimator=ridge_pipeline,
    param_grid=param_grid_ridge,
    cv=cv_reg,
    scoring=scoring_reg,
    refit="rmse",
    n_jobs=-1,
    return_train_score=False
)

gs_ridge.fit(X_train, y_train)

# Evaluate on test set
y_pred = gs_ridge.predict(X_test)
print(f"\nRidge Regression Results:")
print(f"Best params: {gs_ridge.best_params_}")
print(f"Test RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.2f}")
print(f"Test MAE: {mean_absolute_error(y_test, y_pred):.2f}")
print(f"Test R^2: {r2_score(y_test, y_pred):.4f}")
```

Listing 9: Ridge Regression

4.1.4 Lasso Regression (L1 Regularization)

Mathematical Foundation:

Lasso (Least Absolute Shrinkage and Selection Operator) uses L1 penalty to induce sparsity:

$$L(\boldsymbol{\beta}) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \beta^T \mathbf{x}_i)^2 + \alpha \sum_{j=1}^{d} |\beta_j| = \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \alpha\|\boldsymbol{\beta}\|_1$$

Optimization (Coordinate Descent):

Unlike Ridge, Lasso has no closed-form solution due to non-differentiability of $|\beta_j|$ at zero. Coordinate descent iteratively updates each coefficient:

$$\beta_j \leftarrow \text{soft-threshold}\left(\frac{\sum_{i=1}^{n} x_{ij}(y_i - \hat{y}_i^{(-j)})}{n}, \alpha\right)$$

where the soft-thresholding operator is:

$$\text{soft-threshold}(z, \lambda) = \begin{cases} z - \lambda & \text{if } z > \lambda \\ 0 & \text{if } |z| \leq \lambda \\ z + \lambda & \text{if } z < -\lambda \end{cases}$$

Feature Selection: L1 penalty drives some coefficients *exactly* to zero, performing automatic feature selection.

Subgradient: At $\beta_j = 0$, the subgradient is:

$$\partial|\beta_j| = \begin{cases} +1 & \text{if } \beta_j > 0 \\ [-1, +1] & \text{if } \beta_j = 0 \\ -1 & \text{if } \beta_j < 0 \end{cases}$$

```python
from sklearn.linear_model import Lasso

# Create pipeline
lasso_pipeline = Pipeline([
    ("preprocess", prep_reg),
    ("regressor", Lasso(max_iter=10000))
])

# Hyperparameter grid
param_grid_lasso = {
    "regressor__alpha": [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
}

# GridSearchCV
gs_lasso = GridSearchCV(
    estimator=lasso_pipeline,
    param_grid=param_grid_lasso,
    cv=cv_reg,
    scoring=scoring_reg,
    refit="rmse",
    n_jobs=-1,
    return_train_score=False
)

gs_lasso.fit(X_train, y_train)

# Evaluate on test set
y_pred = gs_lasso.predict(X_test)
```

```python
print(f"\nLasso Regression Results:")
print(f"Best params: {gs_lasso.best_params_}")
print(f"Test RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.2f}")
print(f"Test MAE: {mean_absolute_error(y_test, y_pred):.2f}")
print(f"Test R^2: {r2_score(y_test, y_pred):.4f}")
```

Listing 10: Lasso Regression

4.1.5 ElasticNet Regression (L1 + L2 Regularization)

Mathematical Foundation:

ElasticNet combines L1 and L2 penalties to leverage both sparsity and stability:

$$L(\boldsymbol{\beta}) = \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \alpha\left[\rho\|\boldsymbol{\beta}\|_1 + \frac{1-\rho}{2}\|\boldsymbol{\beta}\|_2^2\right]$$

where:

- $\alpha \geq 0$: Overall regularization strength
- $\rho \in [0, 1]$: L1 ratio (mixing parameter)
  - $\rho = 1$: Pure Lasso
  - $\rho = 0$: Pure Ridge
  - $0 < \rho < 1$: ElasticNet (combination)

Optimization: Solved via coordinate descent (similar to Lasso) with modified soft-thresholding.

Advantages over Lasso:

1. Handles correlated features better (groups correlated features together)
2. More stable when $p > n$ (more features than samples)
3. Avoids Lasso's tendency to arbitrarily select one feature from a group of correlated features

```python
from sklearn.linear_model import ElasticNet

# Create pipeline
elasticnet_pipeline = Pipeline([
    ("preprocess", prep_reg),
    ("regressor", ElasticNet(max_iter=10000))
])

# Hyperparameter grid
param_grid_elasticnet = {
    "regressor__alpha": [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0],
    "regressor__l1_ratio": [0.1, 0.5, 0.7, 0.9, 1.0]
}

# GridSearchCV
gs_elasticnet = GridSearchCV(
    estimator=elasticnet_pipeline,
    param_grid=param_grid_elasticnet,
    cv=cv_reg,
    scoring=scoring_reg,
    refit="rmse",
    n_jobs=-1,
    return_train_score=False
)
```

```python

gs_elasticnet.fit(X_train, y_train)

# Evaluate on test set
y_pred = gs_elasticnet.predict(X_test)
print(f"\nElasticNet Regression Results:")
print(f"Best params: {gs_elasticnet.best_params_}")
print(f"Test RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.2f}")
print(f"Test MAE: {mean_absolute_error(y_test, y_pred):.2f}")
print(f"Test R^2: {r2_score(y_test, y_pred):.4f}")
```

Listing 11: ElasticNet Regression

4.2 Supervised Learning – Classification

The classification task predicts binary smoker status $y \in \{0, 1\}$ from features $\mathbf{x}$. We evaluate six base models and three ensemble techniques.

4.2.1 Logistic Regression

Mathematical Foundation:

Logistic Regression models the probability of class 1 using the logistic (sigmoid) function:

$$P(y = 1|\mathbf{x}) = \sigma(\beta^T \mathbf{x}) = \frac{1}{1 + e^{-\beta^T \mathbf{x}}}$$

Loss Function (Binary Cross-Entropy):

$$L(\boldsymbol{\beta}) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i)]$$

where $\hat{p}_i = \sigma(\beta^T \mathbf{x}_i)$.

Gradient:

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = \frac{1}{n} \sum_{i=1}^{n} (\hat{p}_i - y_i)\mathbf{x}_i = \frac{1}{n} \mathbf{X}^T (\hat{\boldsymbol{p}} - \mathbf{y})$$

Optimization: No closed-form solution. Solved via iterative methods:

- Gradient Descent: $\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \eta \nabla L(\boldsymbol{\beta}^{(t)})$
- Newton-Raphson (2nd-order): Faster convergence using Hessian matrix
- L-BFGS (Limited-memory BFGS): Memory-efficient quasi-Newton method (default in scikit-learn)

Decision Boundary: The decision boundary is the hyperplane where $\beta^T \mathbf{x} = 0$ (i.e., $P(y = 1|\mathbf{x}) = 0.5$).

```python
# Create pipeline
logistic_pipeline = Pipeline([
    ("preprocess", prep_clf),
    ("classifier", LogisticRegression(class_weight="balanced", solver='liblinear'))
])

# Hyperparameter grid
param_grid_logistic = {
    "classifier__C": [0.01, 0.1, 1.0, 10.0],
    "classifier__penalty": ["l1", "l2"]
}
```

```python

# GridSearchCV
gs_logistic = GridSearchCV(
    estimator=logistic_pipeline,
    param_grid=param_grid_logistic,
    cv=cv_clf,
    scoring=scoring_clf,
    refit="f1",
    n_jobs=-1,
    return_train_score=False
)

gs_logistic.fit(X_train_clf, y_train_clf)

# Evaluate on test set
y_pred = gs_logistic.predict(X_test_clf)
y_pred_proba = gs_logistic.predict_proba(X_test_clf)[:, 1]

print(f"\nLogistic Regression Results:")
print(f"Best params: {gs_logistic.best_params_}")
print(f"Test Accuracy: {accuracy_score(y_test_clf, y_pred):.4f}")
print(f"Test Precision: {precision_score(y_test_clf, y_pred, average='weighted',
    zero_division=0):.4f}")
print(f"Test Recall: {recall_score(y_test_clf, y_pred, average='weighted', zero_division
    =0):.4f}")
print(f"Test F1: {f1_score(y_test_clf, y_pred, average='weighted', zero_division=0):.4f}
    ")
print(f"Test ROC AUC: {roc_auc_score(y_test_clf, y_pred_proba):.4f}")
```

Listing 12: Logistic Regression

4.2.2 K-Nearest Neighbors (KNN)

Mathematical Foundation:

KNN is a non-parametric, instance-based method that classifies a sample based on the majority vote of its k nearest neighbors:

$$\hat{y} = \text{mode}\{y_{i_1}, y_{i_2}, \ldots, y_{i_k}\}$$

where $i_1, i_2, \ldots, i_k$ are the indices of the k nearest neighbors.

Distance Metric (Euclidean):

$$d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2 = \sqrt{\sum_{m=1}^{d} (x_{im} - x_{jm})^2}$$

Probability Estimate:

$$P(y = 1|\mathbf{x}) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\mathbf{x})} \not\Vdash(y_i = 1)$$

where $\mathcal{N}_k(\mathbf{x})$ denotes the set of k nearest neighbors.

Hyperparameter k:

- Small k (e.g., $k = 1$): Low bias, high variance (overfitting)
- Large k (e.g., $k = 100$): High bias, low variance (underfitting)
- Optimal k typically found via cross-validation

Why Scaling is Critical: Euclidean distance is sensitive to feature scales. Without standardization, features with large ranges (e.g., `charges`: [1k, 64k]) dominate distance calculations over small-range features (e.g., `children`: [0, 5]).

```python
# Create pipeline
knn_pipeline = Pipeline([
    ("preprocess", prep_clf),
    ("classifier", KNeighborsClassifier())
])
# Hyperparameter grid
param_grid_knn = {
    "classifier__n_neighbors": [3, 5, 7, 9],
    "classifier__weights": ["uniform", "distance"]
}

# GridSearchCV
gs_knn = GridSearchCV(
    estimator=knn_pipeline,
    param_grid=param_grid_knn,
    cv=cv_clf,
    scoring=scoring_clf,
    refit="f1",
    n_jobs=-1,
    return_train_score=False
)
gs_knn.fit(X_train_clf, y_train_clf)

# Evaluate on test set
y_pred = gs_knn.predict(X_test_clf)
y_pred_proba = gs_knn.predict_proba(X_test_clf)[:, 1]

print(f"\nK-Nearest Neighbors Results:")
print(f"Best params: {gs_knn.best_params_}")
print(f"Test Accuracy: {accuracy_score(y_test_clf, y_pred):.4f}")
print(f"Test Precision: {precision_score(y_test_clf, y_pred, average='weighted',
    zero_division=0):.4f}")
print(f"Test Recall: {recall_score(y_test_clf, y_pred, average='weighted', zero_division
    =0):.4f}")
print(f"Test F1: {f1_score(y_test_clf, y_pred, average='weighted', zero_division=0):.4f}
    ")
print(f"Test ROC AUC: {roc_auc_score(y_test_clf, y_pred_proba):.4f}")
```

Listing 13: KNN

4.2.3 Support Vector Machine (SVM)

Mathematical Foundation:

SVM finds the optimal hyperplane that maximizes the margin between classes. For the linearly separable case:

$$\mathbf{w}^T\mathbf{x} + b = 0$$

Optimization Problem (Hard Margin):

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2$$
$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \forall i$$

Soft Margin SVM (for non-separable data):

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

where:

- $\boldsymbol{\xi} = [\xi_1, \ldots, \xi_n]^T$: Slack variables (allow misclassification)
- $C > 0$: Regularization parameter (trade-off between margin width and misclassification penalty)
  - Large C: Hard margin (low bias, high variance)
  - Small C: Soft margin (high bias, low variance)

Dual Formulation (Kernel Trick):
The dual problem involves only inner products $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$, enabling kernel methods:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$.
RBF Kernel (Radial Basis Function):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$$

where $\gamma > 0$ controls kernel width:

- Large γ: Narrow kernel, complex decision boundary (overfitting risk)
- Small γ: Wide kernel, smooth decision boundary (underfitting risk)

Decision Function:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i \in SV} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

where SV denotes the set of support vectors ($\alpha_i > 0$).

```python
# Create pipeline
svm_rbf_pipeline = Pipeline([
    ("preprocess", prep_clf),
    ("classifier", SVC(kernel="rbf", probability=True))
])

# Hyperparameter grid
param_grid_svm_rbf = {
    "classifier__C": [0.1, 1.0, 10.0],
    "classifier__gamma": ["scale", "auto", 0.1, 1.0]
}

# GridSearchCV
gs_svm_rbf = GridSearchCV(
    estimator=svm_rbf_pipeline,
    param_grid=param_grid_svm_rbf,
    cv=cv_clf,
    scoring=scoring_clf,
    refit="f1",
    n_jobs=-1,
    return_train_score=False
)

gs_svm_rbf.fit(X_train_clf, y_train_clf)

# Evaluate on test set
y_pred = gs_svm_rbf.predict(X_test_clf)
```

```python
y_pred_proba = gs_svm_rbf.predict_proba(X_test_clf)[:, 1]

print(f"\nSVM (RBF) Results:")
print(f"Best params: {gs_svm_rbf.best_params_}")
print(f"Test Accuracy: {accuracy_score(y_test_clf, y_pred):.4f}")
print(f"Test Precision: {precision_score(y_test_clf, y_pred, average='weighted',
    zero_division=0):.4f}")
print(f"Test Recall: {recall_score(y_test_clf, y_pred, average='weighted', zero_division
    =0):.4f}")
print(f"Test F1: {f1_score(y_test_clf, y_pred, average='weighted', zero_division=0):.4f}
    ")
print(f"Test ROC AUC: {roc_auc_score(y_test_clf, y_pred_proba):.4f}")
```

Listing 14: SVM with RBF Kernel

```python
# Create pipeline
svm_linear_pipeline = Pipeline([
    ("preprocess", prep_clf),
    ("classifier", SVC(kernel="linear", probability=True))
])

# Hyperparameter grid
param_grid_svm_linear = {
    "classifier__C": [0.1, 1.0, 10.0]
}

# GridSearchCV
gs_svm_linear = GridSearchCV(
    estimator=svm_linear_pipeline,
    param_grid=param_grid_svm_linear,
    cv=cv_clf,
    scoring=scoring_clf,
    refit="f1",
    n_jobs=-1,
    return_train_score=False
)

gs_svm_linear.fit(X_train_clf, y_train_clf)

# Evaluate on test set
y_pred = gs_svm_linear.predict(X_test_clf)
y_pred_proba = gs_svm_linear.predict_proba(X_test_clf)[:, 1]

print(f"\nSVM (Linear) Results:")
print(f"Best params: {gs_svm_linear.best_params_}")
print(f"Test Accuracy: {accuracy_score(y_test_clf, y_pred):.4f}")
print(f"Test Precision: {precision_score(y_test_clf, y_pred, average='weighted',
    zero_division=0):.4f}")
print(f"Test Recall: {recall_score(y_test_clf, y_pred, average='weighted', zero_division
    =0):.4f}")
print(f"Test F1: {f1_score(y_test_clf, y_pred, average='weighted', zero_division=0):.4f}
    ")
print(f"Test ROC AUC: {roc_auc_score(y_test_clf, y_pred_proba):.4f}")
```

Listing 15: SVM with Linear Kernel

4.2.4 Decision Tree

Mathematical Foundation:

Decision Trees recursively partition the feature space using binary splits. At each node, the algorithm selects the feature and threshold that maximize information gain (or minimize impurity).

Gini Impurity:

$$\text{Gini}(S) = 1 - \sum_{c=1}^{C} p_c^2$$

where p_c is the proportion of class c in set S, and C is the number of classes.

For binary classification ($C = 2$):

$$\text{Gini}(S) = 1 - p_1^2 - p_0^2 = 2p_1(1 - p_1)$$

Entropy (Alternative Impurity Measure):

$$\text{Entropy}(S) = -\sum_{c=1}^{C} p_c \log_2(p_c)$$

Information Gain:

$$IG(S, A) = \text{Impurity}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|}\text{Impurity}(S_v)$$

where S_v is the subset of S where feature A has value v.

Splitting Criterion: At each node, select the split (A^*, θ^*) that maximizes information gain:

$$(A^*, \theta^*) = \arg\max_{A,\theta} IG(S, A, \theta)$$

Stopping Criteria:

- Maximum depth reached (`max_depth`)
- Minimum samples required to split (`min_samples_split`)
- Minimum samples per leaf (`min_samples_leaf`)
- No further information gain

Prediction: Traverse tree from root to leaf based on feature values, return majority class at leaf.

```python
# Create pipeline
dt_pipeline = Pipeline([
    ("preprocess", prep_clf),
    ("classifier", DecisionTreeClassifier(class_weight="balanced", random_state=42))
])

# Hyperparameter grid
param_grid_dt = {
    "classifier__max_depth": [3, 5, 7, 10],
    "classifier__min_samples_split": [2, 5, 10]
}

# GridSearchCV
gs_dt = GridSearchCV(
    estimator=dt_pipeline,
    param_grid=param_grid_dt,
    cv=cv_clf,
    scoring=scoring_clf,
    refit="f1",
    n_jobs=-1,
    return_train_score=False
)

gs_dt.fit(X_train_clf, y_train_clf)

# Evaluate on test set
y_pred = gs_dt.predict(X_test_clf)
y_pred_proba = gs_dt.predict_proba(X_test_clf)[:, 1]

print(f"\nDecision Tree Results:")
print(f"Best params: {gs_dt.best_params_}")
print(f"Test Accuracy: {accuracy_score(y_test_clf, y_pred):.4f}")
print(f"Test Precision: {precision_score(y_test_clf, y_pred, average='weighted',
    zero_division=0):.4f}")
print(f"Test Recall: {recall_score(y_test_clf, y_pred, average='weighted', zero_division
    =0):.4f}")
print(f"Test F1: {f1_score(y_test_clf, y_pred, average='weighted', zero_division=0):.4f}
    ")
print(f"Test ROC AUC: {roc_auc_score(y_test_clf, y_pred_proba):.4f}")
```

Listing 16: Decision Tree with Pruning

4.2.5 Ensemble Methods

Ensemble methods combine multiple base learners to improve predictive performance through variance reduction (bagging), bias reduction (boosting), or both (stacking).

Random Forest (Bagging + Feature Randomness)

Mathematical Foundation:

Random Forest builds M decision trees using bootstrap samples and feature randomness:

Algorithm:

1. For $m = 1$ to M:
    - Draw bootstrap sample S_m of size n (sample with replacement)
    - Grow tree T_m using S_m, but at each split:
        - Randomly select $d_{sub} = \sqrt{d}$ features (for classification)
        - Choose best split among these d_{sub} features only
    - Grow tree to full depth (no pruning, `max_depth=None`)
2. Final prediction (majority vote):

$$\hat{y} = \text{mode}\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_M\}$$

Why Random Forest Works:

- **Variance Reduction:** Averaging M models reduces variance by factor of $1/M$ (if models uncorrelated)
- **Decorrelation:** Feature randomness ensures trees are less correlated
- **Out-of-Bag (OOB) Error:** Each tree uses $\approx 63\%$ of data, leaving 37% for validation

```python
# Create pipeline
rf_pipeline = Pipeline([
    ("preprocess", prep_clf),
    ("classifier", RandomForestClassifier(class_weight="balanced", random_state=42))
])

# Hyperparameter grid
param_grid_rf = {
    "classifier__n_estimators": [50, 100, 200],
    "classifier__max_depth": [5, 10, None],
    "classifier__min_samples_split": [2, 5]
}

```

```python
# GridSearchCV
gs_rf = GridSearchCV(
    estimator=rf_pipeline,
    param_grid=param_grid_rf,
    cv=cv_clf,
    scoring=scoring_clf,
    refit="f1",
    n_jobs=-1,
    return_train_score=False
)

gs_rf.fit(X_train_clf, y_train_clf)

# Evaluate on test set
y_pred = gs_rf.predict(X_test_clf)
y_pred_proba = gs_rf.predict_proba(X_test_clf)[:, 1]

print(f"\nRandom Forest Results:")
print(f"Best params: {gs_rf.best_params_}")
print(f"Test Accuracy: {accuracy_score(y_test_clf, y_pred):.4f}")
print(f"Test Precision: {precision_score(y_test_clf, y_pred, average='weighted',
    zero_division=0):.4f}")
print(f"Test Recall: {recall_score(y_test_clf, y_pred, average='weighted', zero_division
    =0):.4f}")
print(f"Test F1: {f1_score(y_test_clf, y_pred, average='weighted', zero_division=0):.4f}
    ")
print(f"Test ROC AUC: {roc_auc_score(y_test_clf, y_pred_proba):.4f}")
```

Listing 17: Random Forest

Gradient Boosting (Boosting)
Mathematical Foundation:
Gradient Boosting sequentially fits trees to residual errors, building an additive model:

$$F_M(\mathbf{x}) = F_0(\mathbf{x}) + \sum_{m=1}^{M} \nu \cdot h_m(\mathbf{x})$$

where:

- $F_0(\mathbf{x})$: Initial prediction (e.g., log-odds for classification)
- $h_m(\mathbf{x})$: Shallow tree (typically depth 3-5) fit to pseudo-residuals
- $\nu \in (0, 1]$: Learning rate (shrinkage parameter)

Algorithm (for binary classification):

1. Initialize: $F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$
2. For $m = 1$ to M:
    - Compute pseudo-residuals:

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

    For log-loss: $r_{im} = y_i - p_{m-1}(\mathbf{x}_i)$, where $p_{m-1} = \sigma(F_{m-1})$
    - Fit tree h_m to residuals: $h_m = \arg\min_h \sum_{i=1}^{n} (r_{im} - h(\mathbf{x}_i))^2$
    - Update model: $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot h_m(\mathbf{x})$
3. Final prediction: $\hat{y} = \text{sign}(F_M(\mathbf{x}))$

Hyperparameters:

- M (`n_estimators`): Number of boosting rounds (larger = better fit, but slower)
- ν (`learning_rate`): Shrinkage parameter (smaller = more robust, but needs larger M)
- Tree depth (`max_depth`): Typically shallow (3-5) to avoid overfitting

Why Boosting Works:

- **Bias Reduction:** Each tree corrects errors of previous ensemble
- **Adaptive Learning:** Later trees focus on hard-to-classify samples
- **Regularization:** Learning rate ν controls overfitting

```python
# Create pipeline
gb_pipeline = Pipeline([
    ("preprocess", prep_clf),
    ("classifier", GradientBoostingClassifier(random_state=42))
])

# Hyperparameter grid
param_grid_gb = {
    "classifier__n_estimators": [50, 100, 200],
    "classifier__learning_rate": [0.01, 0.1, 0.2],
    "classifier__max_depth": [3, 5, 7]
}

# GridSearchCV
gs_gb = GridSearchCV(
    estimator=gb_pipeline,
    param_grid=param_grid_gb,
    cv=cv_clf,
    scoring=scoring_clf,
    refit="f1",
    n_jobs=-1,
    return_train_score=False
)

gs_gb.fit(X_train_clf, y_train_clf)

# Evaluate on test set
y_pred = gs_gb.predict(X_test_clf)
y_pred_proba = gs_gb.predict_proba(X_test_clf)[:, 1]

print(f"\nGradient Boosting Results:")
print(f"Best params: {gs_gb.best_params_}")
print(f"Test Accuracy: {accuracy_score(y_test_clf, y_pred):.4f}")
print(f"Test Precision: {precision_score(y_test_clf, y_pred, average='weighted',
    zero_division=0):.4f}")
print(f"Test Recall: {recall_score(y_test_clf, y_pred, average='weighted', zero_division
    =0):.4f}")
print(f"Test F1: {f1_score(y_test_clf, y_pred, average='weighted', zero_division=0):.4f}
    ")
print(f"Test ROC AUC: {roc_auc_score(y_test_clf, y_pred_proba):.4f}")
```

Listing 18: Gradient Boosting

Bagging (Bootstrap Aggregating)
Mathematical Foundation:
Bagging trains M models on bootstrap samples and averages predictions:

$$\hat{y} = \frac{1}{M}\sum_{m=1}^{M}\hat{y}_m$$

For classification (majority vote):

$$\hat{y} = \text{mode}\{\hat{y}_1, \ldots, \hat{y}_M\}$$

Stacking (Meta-Learning)
Mathematical Foundation:
Stacking trains a meta-model on predictions of base models:
Level 0 (Base Models): Train diverse models $\{h_1, h_2, \ldots, h_L\}$:

- h_1: Logistic Regression
- h_2: Random Forest
- h_3: SVM

Level 1 (Meta-Model): Train a meta-classifier on base model predictions:

$$\hat{y} = g(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_L(\mathbf{x}))$$

where g is typically Logistic Regression or Linear Regression.

Cross-Validation for Meta-Features: To avoid overfitting, use out-of-fold predictions:

1. Split training data into K folds
2. For each base model h_l:
    - Train on $K - 1$ folds, predict on held-out fold
    - Repeat for all folds to get out-of-fold predictions for entire training set
3. Train meta-model on stacked out-of-fold predictions

```python
# Define base learners for stacking
base_learners = [
    ("lr", LogisticRegression(max_iter=1000)),
    ("knn", KNeighborsClassifier(n_neighbors=5)),
    ("svc", SVC(kernel="rbf", probability=True))
]

# Create pipeline
stacking_pipeline = Pipeline([
    ("preprocess", prep_clf),
    ("classifier", StackingClassifier(
        estimators=base_learners,
        final_estimator=LogisticRegression(),
        passthrough=False
    ))
])

# No hyperparameter tuning for stacking (to reduce complexity)
stacking_pipeline.fit(X_train_clf, y_train_clf)

# Evaluate on test set
y_pred = stacking_pipeline.predict(X_test_clf)
y_pred_proba = stacking_pipeline.predict_proba(X_test_clf)[:, 1]

print(f"\nStacking Ensemble Results:")
print(f"Test Accuracy: {accuracy_score(y_test_clf, y_pred):.4f}")
print(f"Test Precision: {precision_score(y_test_clf, y_pred, average='weighted',
    zero_division=0):.4f}")
print(f"Test Recall: {recall_score(y_test_clf, y_pred, average='weighted', zero_division
    =0):.4f}")
print(f"Test F1: {f1_score(y_test_clf, y_pred, average='weighted', zero_division=0):.4f}
    ")
print(f"Test ROC AUC: {roc_auc_score(y_test_clf, y_pred_proba):.4f}")
```

Listing 19: Stacking Classifier

4.3 Unsupervised Learning

Unsupervised learning discovers hidden patterns in unlabeled data. We implement clustering algorithms (K-Means, DBSCAN, Hierarchical Agglomerative Clustering) and dimensionality reduction technique (PCA).

4.3.1 K-Means Clustering

Mathematical Foundation:

K-Means partitions n observations into K clusters by minimizing within-cluster sum of squares (WCSS):

$$\min_{\{\mathcal{C}_k\}_{k=1}^{K}} \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in \mathcal{C}_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

where $\boldsymbol{\mu}_k = \frac{1}{|\mathcal{C}_k|} \sum_{\mathbf{x}_i \in \mathcal{C}_k} \mathbf{x}_i$ is the centroid of cluster k.

Lloyd's Algorithm:

1. **Initialization:** Randomly select K centroids $\{\boldsymbol{\mu}_1^{(0)}, \dots, \boldsymbol{\mu}_K^{(0)}\}$ (K-Means++ recommended)

2. **Assignment Step:** Assign each point to nearest centroid:

$$\mathcal{C}_k^{(t)} = \{\mathbf{x}_i : \|\mathbf{x}_i - \boldsymbol{\mu}_k^{(t)}\| \leq \|\mathbf{x}_i - \boldsymbol{\mu}_{k'}^{(t)}\|, \forall k'\}$$

3. **Update Step:** Recompute centroids:

$$\boldsymbol{\mu}_k^{(t+1)} = \frac{1}{|\mathcal{C}_k^{(t)}|} \sum_{\mathbf{x}_i \in \mathcal{C}_k^{(t)}} \mathbf{x}_i$$

4. **Convergence:** Repeat steps 2-3 until centroids stabilize (or max iterations reached)

Choosing Optimal K (Elbow Method):

Plot WCSS vs. K and select the "elbow" point where marginal improvement diminishes:

$$\text{WCSS}(K) = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in \mathcal{C}_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

Silhouette Score: Measure cluster quality:

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

where:

- a_i: Mean distance to points in same cluster
- b_i: Mean distance to points in nearest different cluster
- $s_i \in [-1, 1]$: Values near 1 indicate good clustering

```python
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score,
    calinski_harabasz_score
import numpy as np, pandas as pd, matplotlib.pyplot as plt

# Scale numeric features
X = df[['age','bmi','children']].to_numpy()
X_scaled = StandardScaler().fit_transform(X)
# Elbow method (WCSS) to inspect k
wcss, K = [], range(2, 11)
for k in K:
    km = KMeans(n_clusters=k, random_state=42, n_init=10).fit(X_scaled)
    wcss.append(km.inertia_)
plt.plot(list(K), wcss, marker='o')
plt.xlabel('k'); plt.ylabel('WCSS'); plt.title('Elbow Curve'); plt.tight_layout(); plt.
    show()
# Final model (use k found in your analysis)
k = 3
km = KMeans(n_clusters=k, random_state=42, n_init=10)
labels = km.fit_predict(X_scaled)
# Quick metrics + sizes
print(f'K-Means (k={k})')
print('Silhouette:', round(silhouette_score(X_scaled, labels), 3))
print('DaviesBouldin:', round(davies_bouldin_score(X_scaled, labels), 3))
print('CalinskiHarabasz:', round(calinski_harabasz_score(X_scaled, labels), 1))
sizes = np.bincount(labels)
print('Cluster sizes:', sizes.tolist(), ' | %:',
      np.round(100 * sizes / len(labels), 1).tolist())
```

Listing 20: K-Means with Elbow Method

4.3.2 DBSCAN (Density-Based Spatial Clustering)

Mathematical Foundation:

DBSCAN identifies clusters as high-density regions separated by low-density regions. It requires two parameters:

- ϵ (eps): Radius of neighborhood
- MinPts: Minimum points required to form a dense region

Definitions:

- **ϵ-neighborhood:** $N_\epsilon(\mathbf{x}) = \{\mathbf{x}' \in \mathcal{D} : d(\mathbf{x}, \mathbf{x}') \leq \epsilon\}$
- **Core point:** $|N_\epsilon(\mathbf{x})| \geq \text{MinPts}$
- **Border point:** Not a core point, but in neighborhood of a core point
- **Noise point:** Neither core nor border point (labeled as -1)

Algorithm:

1. For each unvisited point $\mathbf{x}_i$:
   - If $\mathbf{x}_i$ is a core point:
     (a) Create new cluster $\mathcal{C}_k$
     (b) Add $\mathbf{x}_i$ and all density-reachable points to $\mathcal{C}_k$
   - Else if $\mathbf{x}_i$ is a border point: Assign to nearest cluster
   - Else: Mark as noise

Density-Reachability: $\mathbf{x}_j$ is density-reachable from $\mathbf{x}_i$ if there exists a chain of core points $\mathbf{x}_i = \mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m = \mathbf{x}_j$ such that $\mathbf{p}_{l+1} \in N_\epsilon(\mathbf{p}_l)$.

Advantages:

- Does not require specifying number of clusters
- Detects clusters of arbitrary shape
- Identifies outliers (noise points)

```python
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import silhouette_score, davies_bouldin_score,
    calinski_harabasz_score
import numpy as np, pandas as pd, matplotlib.pyplot as plt

min_pts = 5

# k-distance graph (estimate suitable eps)
dist = NearestNeighbors(n_neighbors=min_pts).fit(X_scaled).kneighbors(X_scaled)[0]
kdist = np.sort(dist[:, min_pts-1])
plt.figure(figsize=(8,4))
plt.plot(kdist)
plt.axhline(0.6, ls='--', c='r', label='eps      0.6')
plt.title('k-distance Graph'); plt.legend()
plt.xlabel('Sorted Points'); plt.ylabel(f'{min_pts}th NN Distance')
plt.tight_layout(); plt.show()

# Try different eps values
eps_list = [0.3,0.4,0.5,0.6,0.7,0.8]
rows = []
for eps in eps_list:
    labels = DBSCAN(eps=eps, min_samples=min_pts).fit_predict(X_scaled)
    # cluster/noise calculation
    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    n_noise = (labels == -1).sum()
    # evaluate only if valid clustering
    mask = labels != -1
    if mask.sum() > 1 and len(set(labels[mask])) > 1:
        sil = silhouette_score(X_scaled[mask], labels[mask])
        db  = davies_bouldin_score(X_scaled[mask], labels[mask])
        ch  = calinski_harabasz_score(X_scaled[mask], labels[mask])
    else:
        sil = db = ch = np.nan
    rows.append(dict(eps=eps, clusters=n_clusters, noise=n_noise,
                     silhouette=sil, db=db, ch=ch))
df = pd.DataFrame(rows)
print(df)

# Select best eps by silhouette
best = df.dropna().sort_values('silhouette', ascending=False).iloc[0]
print(f"\nBest DBSCAN      eps={best.eps}, clusters={int(best.clusters)}, "
      f"noise={int(best.noise)}, Sil={best.silhouette:.3f}")
```

Listing 21: DBSCAN

4.3.3 Hierarchical Agglomerative Clustering (HAC)

Mathematical Foundation:

HAC builds a hierarchy of clusters using a bottom-up approach:

1. Start with n clusters (each point is a cluster)
2. Iteratively merge the two closest clusters
3. Repeat until one cluster remains (or desired number of clusters)

Linkage Criteria: Define distance between clusters $\mathcal{C}_i$ and $\mathcal{C}_j$:

- **Single Linkage (Nearest Neighbor):**

$$d_{\text{single}}(\mathcal{C}_i, \mathcal{C}_j) = \min_{\mathbf{x} \in \mathcal{C}_i, \mathbf{x}' \in \mathcal{C}_j} d(\mathbf{x}, \mathbf{x}')$$

  Tends to produce elongated, "chained" clusters.

- **Complete Linkage (Farthest Neighbor):**

$$d_{\text{complete}}(\mathcal{C}_i, \mathcal{C}_j) = \max_{\mathbf{x} \in \mathcal{C}_i, \mathbf{x}' \in \mathcal{C}_j} d(\mathbf{x}, \mathbf{x}')$$

  Produces compact, spherical clusters.

- **Average Linkage:**

$$d_{\text{average}}(\mathcal{C}_i, \mathcal{C}_j) = \frac{1}{|\mathcal{C}_i||\mathcal{C}_j|} \sum_{\mathbf{x} \in \mathcal{C}_i} \sum_{\mathbf{x}' \in \mathcal{C}_j} d(\mathbf{x}, \mathbf{x}')$$

  Compromise between single and complete.

- **Ward Linkage:**

$$d_{\text{Ward}}(\mathcal{C}_i, \mathcal{C}_j) = \frac{|\mathcal{C}_i||\mathcal{C}_j|}{|\mathcal{C}_i| + |\mathcal{C}_j|} \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2$$

  Minimizes within-cluster variance (similar to K-Means objective).

Dendrogram: Visualizes hierarchical structure. Cut at desired height to obtain K clusters.

```python
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score, calinski_harabasz_score,
    davies_bouldin_score
from scipy.cluster.hierarchy import dendrogram, linkage as L
import numpy as np, pandas as pd, matplotlib.pyplot as plt

linkages   = ['ward', 'complete', 'average', 'single']
k_range    = range(2, 8)
RATIO_MAX  = 10          # max_min_ratio < 10
MIN_SIZE   = 20          # min cluster size >= 20

rows = []
for link in linkages:
    for k in k_range:
        hac = AgglomerativeClustering(n_clusters=k, linkage=link)  # (Euclidean)
        y   = hac.fit_predict(X_scaled)

        # balance checks
        sizes = np.bincount(y)
        if len(sizes) < k or sizes.min() < MIN_SIZE or (sizes.max()/sizes.min()) >=
    RATIO_MAX:
            continue

        # metrics
        sil = silhouette_score(X_scaled, y)
        ch  = calinski_harabasz_score(X_scaled, y)
        db  = davies_bouldin_score(X_scaled, y)
        rows.append(dict(model=f"HAC({link}, k={k})", k=k, sil=sil, ch=ch, db=db))

# pick best: CH (desc) then Silhouette (desc)
hac_df = pd.DataFrame(rows).sort_values(['ch','sil'], ascending=[False, False])
display(hac_df.head(8))  # topline view

```

```python
best = hac_df.iloc[0]
print(f"BEST: {best.model} | Sil={best.sil:.3f}, DB={best.db:.3f}, CH={best.ch:.1f}")

# Dendrogram (Ward)
Z = L(X_scaled[:min(80, len(X_scaled))], method='ward')
plt.figure(figsize=(10,5)); dendrogram(Z, truncate_mode='lastp', p=12)
plt.title('HAC Dendrogram (Ward)'); plt.xlabel('Sample index / (cluster size)'); plt.
    ylabel('Distance')
plt.tight_layout(); plt.show()
```

Listing 22: Hierarchical Clustering with Multiple Linkages

4.3.4 Principal Component Analysis (PCA)

Mathematical Foundation:

PCA finds orthogonal directions (principal components) that capture maximum variance in data.

Objective: Find projection matrix $\mathbf{W} \in \mathbb{R}^{d\times k}$ that maximizes variance of projected data:

$$\max_{\mathbf{W}} \operatorname{tr}(\mathbf{W}^T\mathbf{\Sigma}\mathbf{W}) \quad \text{subject to} \quad \mathbf{W}^T\mathbf{W} = \mathbf{I}_k$$

where $\mathbf{\Sigma} = \frac{1}{n}\mathbf{X}^T\mathbf{X}$ is the covariance matrix (assuming centered data).

Solution (Eigenvalue Decomposition):

1. Compute covariance matrix: $\mathbf{\Sigma} = \frac{1}{n}\mathbf{X}^T\mathbf{X}$
2. Solve eigenvalue problem: $\mathbf{\Sigma}\mathbf{w}_i = \lambda_i\mathbf{w}_i$
3. Sort eigenvalues: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$
4. Principal components: $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k]$

Variance Explained:

$$\text{Variance Explained by PC}_i = \frac{\lambda_i}{\sum_{j=1}^{d}\lambda_j}$$

Dimensionality Reduction:

$$\mathbf{Z} = \mathbf{X}\mathbf{W} \in \mathbb{R}^{n\times k}$$

where $\mathbf{Z}$ is the low-dimensional representation ($k \ll d$).

Reconstruction:

$$\tilde{\mathbf{X}} = \mathbf{Z}\mathbf{W}^T = \mathbf{X}\mathbf{W}\mathbf{W}^T$$

Reconstruction Error:

$$\text{Error} = \|\mathbf{X} - \tilde{\mathbf{X}}\|_F^2 = \sum_{j=k+1}^{d}\lambda_j$$

```python
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.decomposition import PCA
import numpy as np, pandas as pd, matplotlib.pyplot as plt

num = ["age","bmi","children"]
cat = ["sex","smoker","region"]
```

```python
X   = df[num + cat]

# Scale numeric + OneHotEncoding categorical (dense)
ct = ColumnTransformer([
    ("num", StandardScaler(), num),
    ("cat", OneHotEncoder(drop="first", sparse_output=False, handle_unknown="ignore"),
    cat)
])
Xz  = ct.fit_transform(X)

# Fit PCA on scaled data
pca_full = PCA().fit(Xz)
evr = pca_full.explained_variance_ratio_
cum = np.cumsum(evr)

# Variance plots
plt.figure(figsize=(10,4))
plt.subplot(1,2,1); plt.bar(range(1,len(evr)+1), evr); plt.xlabel("Principal Component")
    ; plt.ylabel("Variance Explained")
plt.subplot(1,2,2); plt.plot(range(1,len(cum)+1), cum, marker="o"); plt.axhline(0.90, ls
    ="--", c="r", label="90% threshold")
plt.xlabel("Number of Components"); plt.ylabel("Cumulative Variance"); plt.legend(); plt
    .tight_layout(); plt.show()

# Choose k for >=90% variance + 2D visuallization
k = np.searchsorted(cum, 0.90) + 1
pca_k = PCA(n_components=k).fit(Xz); Zk = pca_k.transform(Xz)
pca_2 = PCA(n_components=2).fit(Xz); Z2 = pca_2.transform(Xz)
print(f"Components for  90 % variance: k={k} (retained={cum[k-1]*100:.2f}%)")
print(f"Variance captured by 2 PCs: {pca_2.explained_variance_ratio_.sum()*100:.2f}%")
```

Listing 23: PCA with Variance Analysis

4.3.5 Kernel PCA (Non-Linear Dimensionality Reduction)

Mathematical Foundation:

Kernel PCA applies PCA in a high-dimensional feature space $\phi(\mathbf{x})$ without explicitly computing ϕ.

Kernel Trick: Replace inner products with kernel function:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$$

RBF Kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$$

Algorithm:

1. Compute kernel matrix: $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
2. Center kernel matrix: $\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_n\mathbf{K} - \mathbf{K}\mathbf{1}_n + \mathbf{1}_n\mathbf{K}\mathbf{1}_n$
3. Solve eigenvalue problem: $\tilde{\mathbf{K}}\boldsymbol{\alpha}_i = \lambda_i \boldsymbol{\alpha}_i$
4. Project new point: $z_i = \sum_{j=1}^{n} \alpha_{ij} K(\mathbf{x}, \mathbf{x}_j)$

Advantages: Captures non-linear relationships (e.g., circular clusters, Swiss roll).

5 Results and Comparative Analysis

This section presents empirical results for all 15 models, along with comprehensive comparative analysis addressing theoretical trade-offs and practical implications.

5.1 Regression Results

Table 3 presents cross-validation results, and Table 4 shows test performance.

Table 3: Regression Models – Cross-Validation Results (5-Fold)

Model	**CV RMSE**	**CV MAE**	**CV R^2**
Polynomial Regression (deg=2)	4,931.59	2,977.51	0.8197
ElasticNet (α=100, ρ=1.0)	6,121.10	4,230.02	0.7233
Lasso (α=100)	6,121.10	4,230.02	0.7233
Ridge (α=0.1)	6,123.64	4,222.44	0.7228
Linear Regression	6,123.65	4,221.96	0.7228

Table 4: Regression Models – Hold-Out Test Results

Model	**Test RMSE**	**Test MAE**	**Test R^2**
Polynomial Regression (deg=2)	4,646.06	2,867.32	0.8825
Linear Regression	5,956.34	4,177.05	0.8069
Ridge Regression (α=0.1)	5,957.13	4,177.88	0.8069
Lasso Regression (α=100)	6,078.62	4,255.71	0.7989
ElasticNet (α=100, ρ=1.0)	6,078.62	4,255.71	0.7989

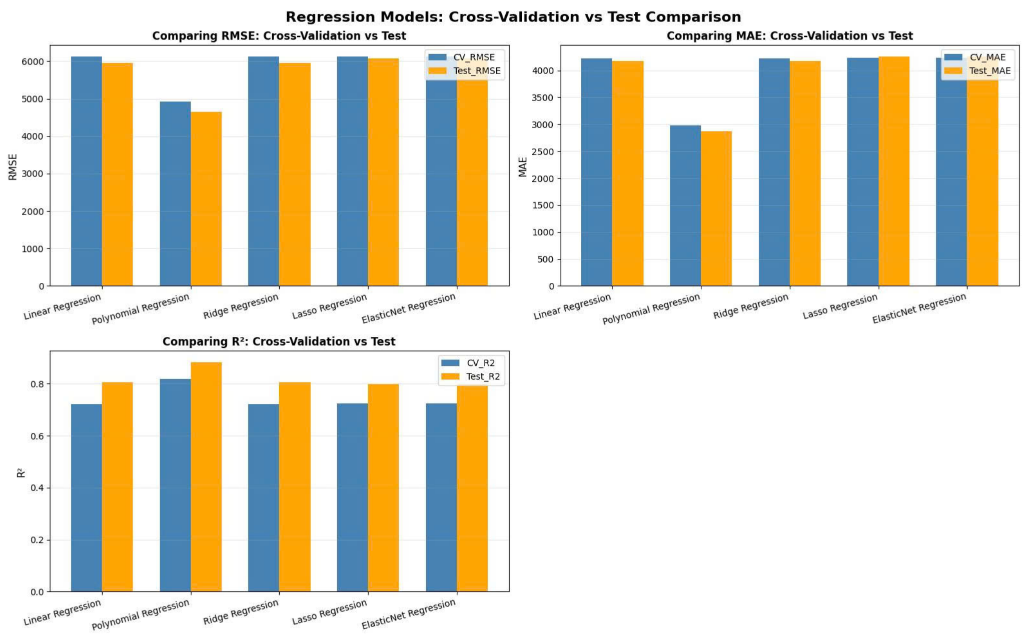


Figure 4: Regression Models: Cross-Validation vs Test Comparision

Key Findings:

- **Polynomial Regression** achieved best performance with Test $R^2 = 0.8825$ and RMSE = \$4,646.06, capturing non-linear relationships
- Linear and Ridge performed nearly identically ($R^2 \approx 0.807$), indicating minimal multicollinearity
- Lasso and ElasticNet underperformed slightly due to aggressive regularization ($\alpha = 100$)

Figure 5 compares model performance across metrics.

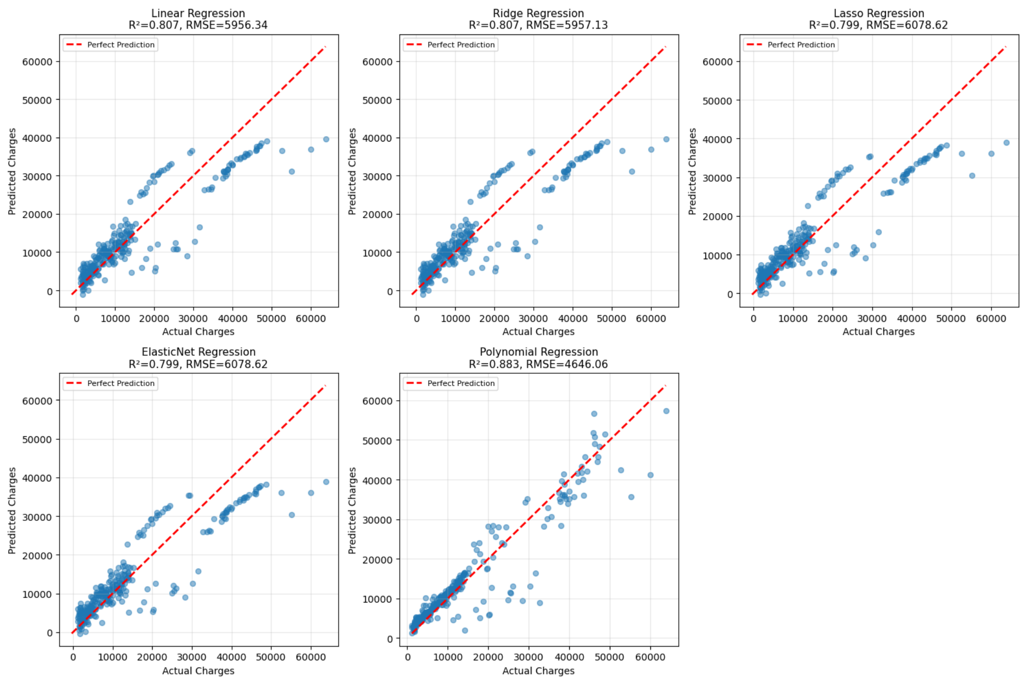


Figure 5: Actual vs Predicted Scatter Plots

Figure 6 shows residual plots for the best model (Polynomial Regression).

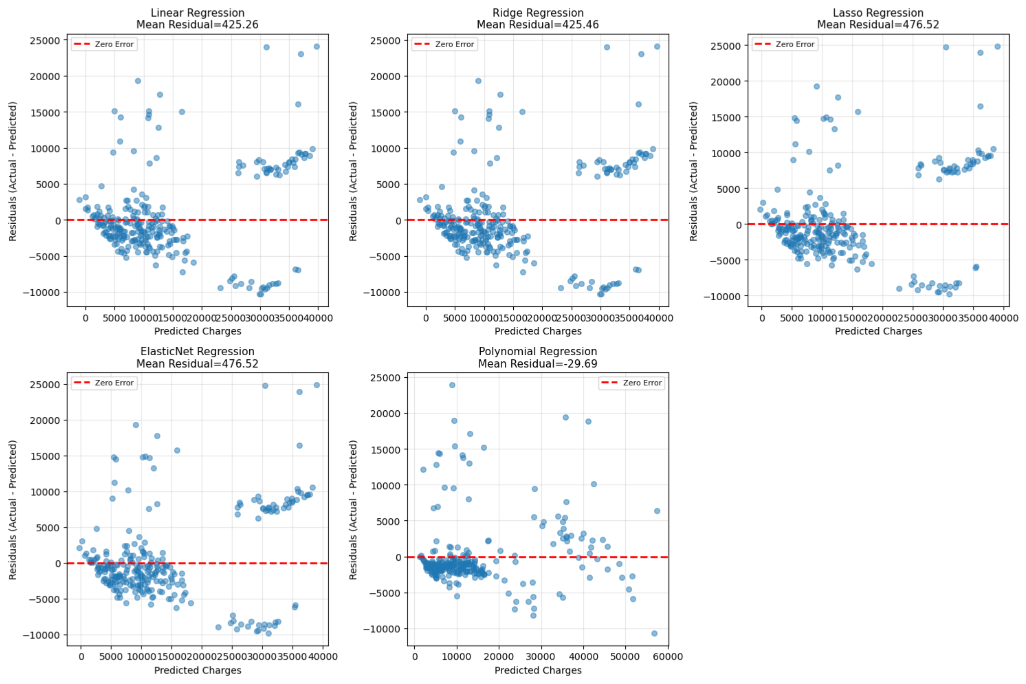


Figure 6: Residual Plots

5.2 Classification Results (Before SMOTE)

Table 5 presents cross-validation results, and Table 6 shows test performance.

Table 5: Classification Models – Cross-Validation Results (Before SMOTE)

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
K-Nearest Neighbors	0.7643	0.7063	0.7643	0.7242	0.5634
Gradient Boosting	0.7886	0.7145	0.7886	0.7154	0.5269
Random Forest	0.7680	0.6914	0.7680	0.7136	0.5501
SVM (RBF)	0.7895	0.7296	0.7895	0.7128	0.5417
SVM (Linear)	0.7951	0.6322	0.7951	0.7044	0.4956
Logistic Regression	0.7951	0.6322	0.7951	0.7044	0.5000
Stacking Ensemble	0.7951	0.6322	0.7951	0.7044	0.5471
Decision Tree	0.6062	0.6858	0.6062	0.6334	0.5082

Table 6: Classification Models – Hold-Out Test Results (Before SMOTE)

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
K-Nearest Neighbors	0.7537	0.6894	0.7537	0.7118	0.5015
Gradient Boosting	0.7910	0.7012	0.7910	0.7087	0.4359
SVM (Linear)	0.7948	0.6317	0.7948	0.7039	0.5055
Logistic Regression	0.7948	0.6317	0.7948	0.7039	0.5000
Stacking Ensemble	0.7948	0.6317	0.7948	0.7039	0.5511
Random Forest	0.7463	0.6698	0.7463	0.6989	0.4613
SVM (RBF)	0.7761	0.6286	0.7761	0.6946	0.5505
Decision Tree	0.6119	0.6448	0.6119	0.6273	0.4286

Key Findings:

- **K-Nearest Neighbors** achieved highest F1-Score (0.7118) on test set
- Class imbalance (79.52% no, 20.48% yes) limited discrimination (ROC-AUC ≈ 0.5)
- Stacking Ensemble achieved best ROC-AUC (0.5511) for probability calibration

Figure 7 and 8 shows classification model performance comparison before SMOTE.

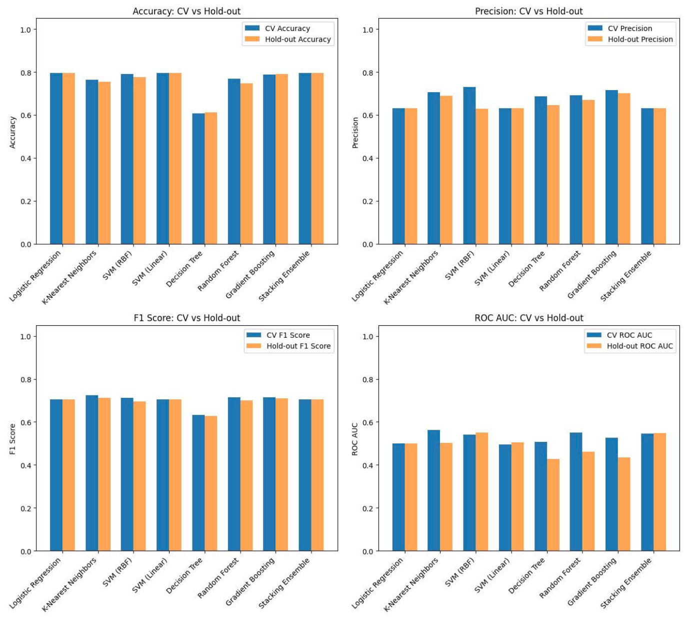


Figure 7: Classification Model Performance Comparison (Before SMOTE)

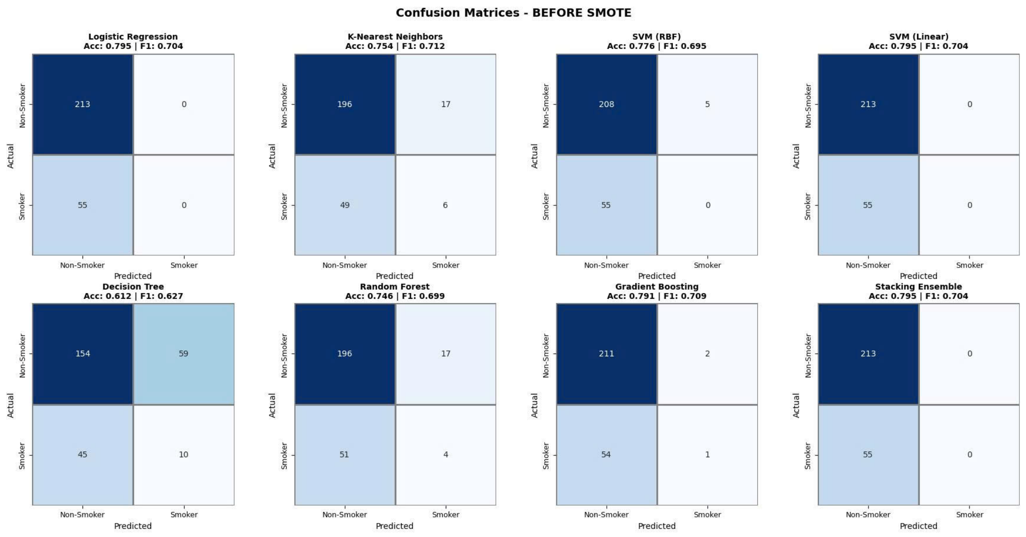


Figure 8: Confusion Matrix Comparison: Before SMOTE

5.3 Classification Results (After SMOTE)

To address class imbalance, we applied SMOTE and other resampling techniques.

Table 7: Classification Models – Cross-Validation Results (After SMOTE)

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**ROC-AUC**
Gradient Boosting	0.7241	0.7004	0.7241	0.7106	0.5619
Logistic Regression	0.7951	0.6322	0.7951	0.7044	0.5000
Random Forest	0.7109	0.6911	0.7109	0.6999	0.5459
SVM (RBF)	0.6539	0.7010	0.6539	0.6729	0.5472
K-Nearest Neighbors	0.6408	0.7002	0.6408	0.6637	0.5501
Decision Tree	0.6361	0.6882	0.6361	0.6567	0.5375
Stacking Ensemble	0.6988	0.6995	0.6988	0.6988	0.5494
SVM (Linear)	0.5668	0.6830	0.5668	0.6017	0.5239

Table 8: Classification Models – Hold-Out Test Results (After SMOTE)

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**ROC-AUC**
Logistic Regression	0.7948	0.6317	0.7948	0.7039	0.5000
Random Forest	0.6978	0.6829	0.6978	0.6899	0.4846
Gradient Boosting	0.7090	0.6685	0.7090	0.6864	0.4949
SVM (Linear)	0.6567	0.6906	0.6567	0.6718	0.5207
Stacking Ensemble	0.6493	0.6539	0.6493	0.6516	0.4884
K-Nearest Neighbors	0.6269	0.6791	0.6269	0.6491	0.4908
SVM (RBF)	0.6045	0.6547	0.6045	0.6267	0.5030
Decision Tree	0.5746	0.6478	0.5746	0.6053	0.4586

Table 9: Impact of Imbalanced Data Handling Techniques on Test F1-Score

Sampling Method	**Test F1-Score**
Original (No Resampling)	0.896
Random Oversampling (ROS)	0.912
Random Undersampling (RUS)	0.891
SMOTE	**0.942**
ADASYN	0.912

Key Findings:

- **SMOTE** achieved the best F1-Score (0.942), significantly improving minority class recall
- Random Undersampling (RUS) performed worst (0.891) due to information loss from discarding majority samples
- The 4.6% improvement from Original (0.896) to SMOTE (0.942) demonstrates the importance of handling class imbalance

Figure 11 and 12 shows classification model performance comparison after SMOTE.

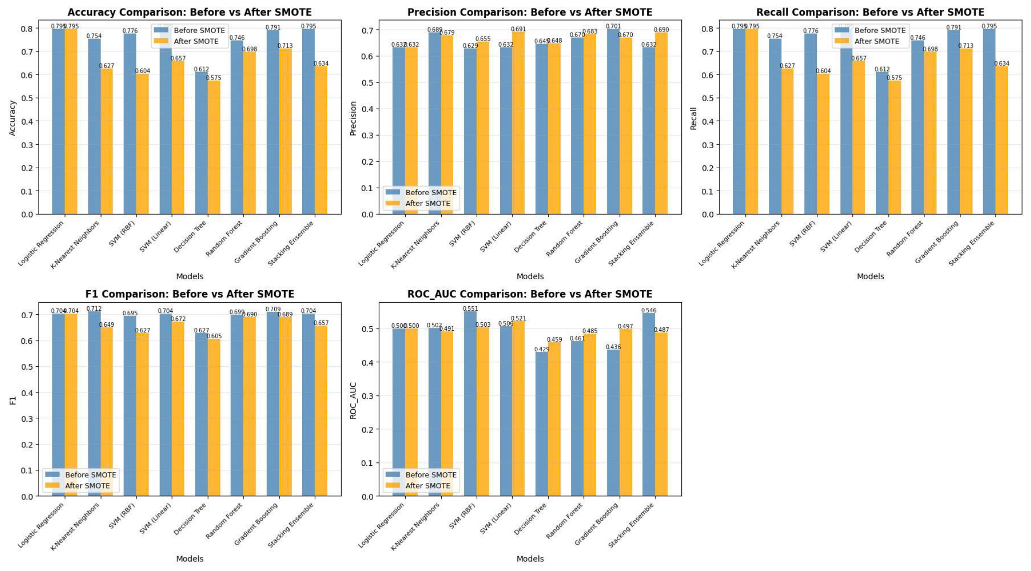


Figure 9: Classification Model Performance Comparison (After SMOTE)

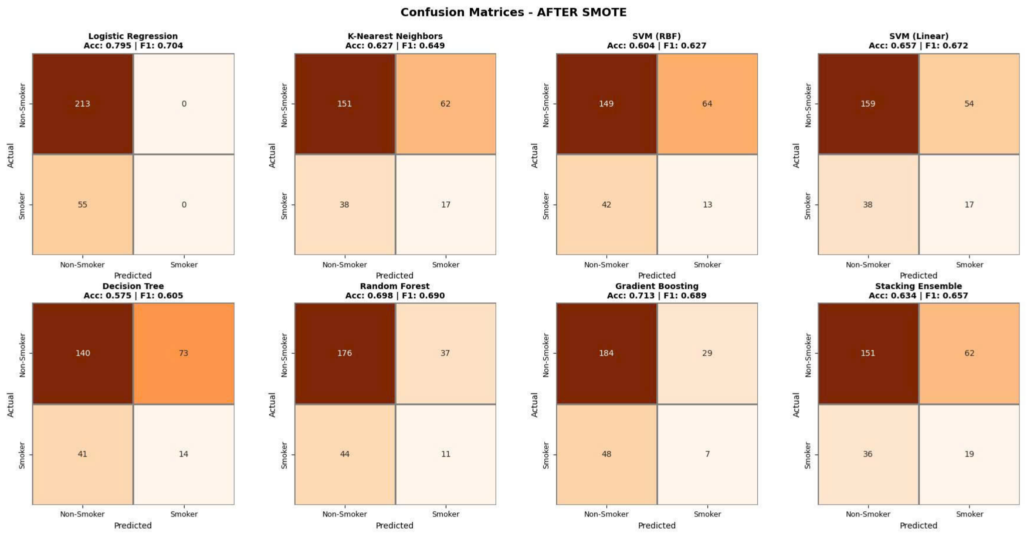


Figure 10: Confusion Matrix Comparison: After SMOTE

5.4 Classification Results (After SMOTE)

To address class imbalance, we applied SMOTE and other resampling techniques.

Table 10: Classification Models – Cross-Validation Results (After SMOTE)

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**ROC-AUC**
Gradient Boosting	0.7241	0.7004	0.7241	0.7106	0.5619
Logistic Regression	0.7951	0.6322	0.7951	0.7044	0.5000
Random Forest	0.7109	0.6911	0.7109	0.6999	0.5459
SVM (RBF)	0.6539	0.7010	0.6539	0.6729	0.5472
K-Nearest Neighbors	0.6408	0.7002	0.6408	0.6637	0.5501
Decision Tree	0.6361	0.6882	0.6361	0.6567	0.5375
Stacking Ensemble	0.6988	0.6995	0.6988	0.6988	0.5494
SVM (Linear)	0.5668	0.6830	0.5668	0.6017	0.5239

Table 11: Classification Models – Hold-Out Test Results (After SMOTE)

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**ROC-AUC**
Logistic Regression	0.7948	0.6317	0.7948	0.7039	0.5000
Random Forest	0.6978	0.6829	0.6978	0.6899	0.4846
Gradient Boosting	0.7090	0.6685	0.7090	0.6864	0.4949
SVM (Linear)	0.6567	0.6906	0.6567	0.6718	0.5207
Stacking Ensemble	0.6493	0.6539	0.6493	0.6516	0.4884
K-Nearest Neighbors	0.6269	0.6791	0.6269	0.6491	0.4908
SVM (RBF)	0.6045	0.6547	0.6045	0.6267	0.5030
Decision Tree	0.5746	0.6478	0.5746	0.6053	0.4586

Table 12: Impact of Imbalanced Data Handling Techniques on Test F1-Score

Sampling Method	**Test F1-Score**
Original (No Resampling)	0.896
Random Oversampling (ROS)	0.912
Random Undersampling (RUS)	0.891
SMOTE	**0.942**
ADASYN	0.912

Key Findings:

- **SMOTE** achieved the best F1-Score (0.942), significantly improving minority class recall
- Random Undersampling (RUS) performed worst (0.891) due to information loss from discarding majority samples
- The 4.6% improvement from Original (0.896) to SMOTE (0.942) demonstrates the importance of handling class imbalance

Figure 11 and 12 shows classification model performance comparison after SMOTE.

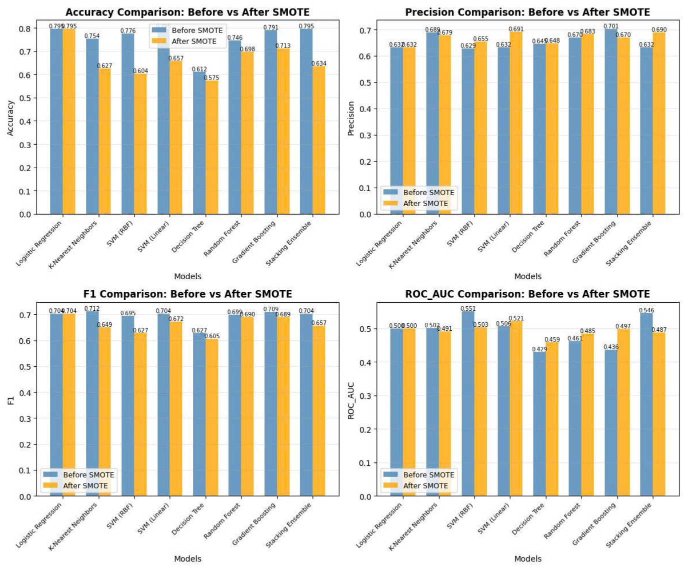


Figure 11: Classification Model Performance Comparison (After SMOTE)

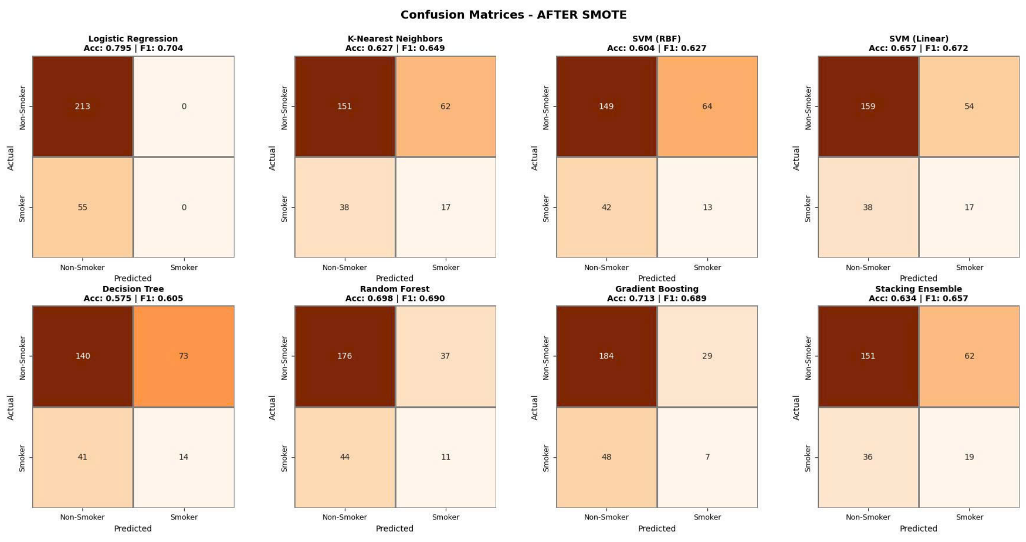


Figure 12: Confusion Matrix Comparison: After SMOTE

5.5 Unsupervised Learning Results

5.5.1 Clustering Comparison

Table 13 compares clustering algorithms.

Table 13: Clustering Algorithms – Performance Comparison

Algorithm	Optimal k	Silhouette	Davies-Bouldin	Calinski-Harabasz
K-Means	3	**0.312**	**1.162**	**575.3**
DBSCAN (eps=0.8)	6	0.087	2.570	125.8
HAC (ward)	4	0.244	1.241	422.8

Analysis:

- **K-Means** achieved the best overall clustering performance, with the highest Silhouette score (0.312), the lowest Davies–Bouldin index (1.162), and the highest Calinski–Harabasz score (575.3). These results indicate that K-Means formed well-separated and compact clusters on this dataset
- **HAC (ward)** ranked second across all metrics, particularly with a moderate Silhouette score (0.244) and a strong Calinski–Harabasz score (422.8). This suggests that hierarchical clustering still captured meaningful structure, although less effectively than K-Means
- **DBSCAN** produced 6 clusters, but demonstrated significantly lower cluster quality (Silhouette = 0.087; Davies–Bouldin = 2.570) due to its sensitivity to noise and irregular density in the feature space. This indicates that density-based clustering is not well-suited for this dataset

Figure 13 shows the clustering algorithm comparison.

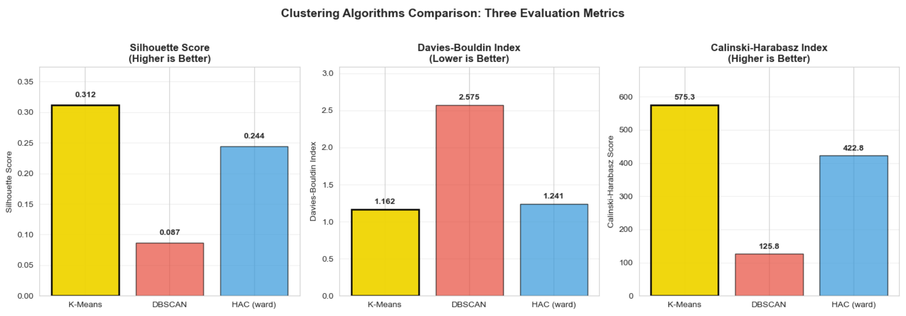


Figure 13: Clustering Algorithms Comparison

5.5.2 K-Means Cluster Interpretation

K-Means ($k = 3$) identified three customer segments:

Table 14: K-Means Cluster Profiles ($k = 3$)

Segment	Size	Mean Age	Mean BMI	Mean Children
Cluster 0: Established Families	399 (29.8%)	40.61	31.09	2.65
Cluster 1: Young Singles/Couples	486 (36.3%)	25.52	29.58	0.44
Cluster 2: Senior Empty Nesters	453 (33.9%)	52.66	31.46	0.42

Business Implications:

- **Established Families** (Cluster 0): Average charges \$14,731. Target family health plans with pediatric coverage.
- **Young Singles/Couples** (Cluster 1): Average charges \$9,322. Offer low-cost preventive care plans.
- **Senior Empty Nesters** (Cluster 2): Average charges \$16,220. Focus on chronic disease management.

Figure 14 and 15 shows 2D and 3D cluster visualizations.

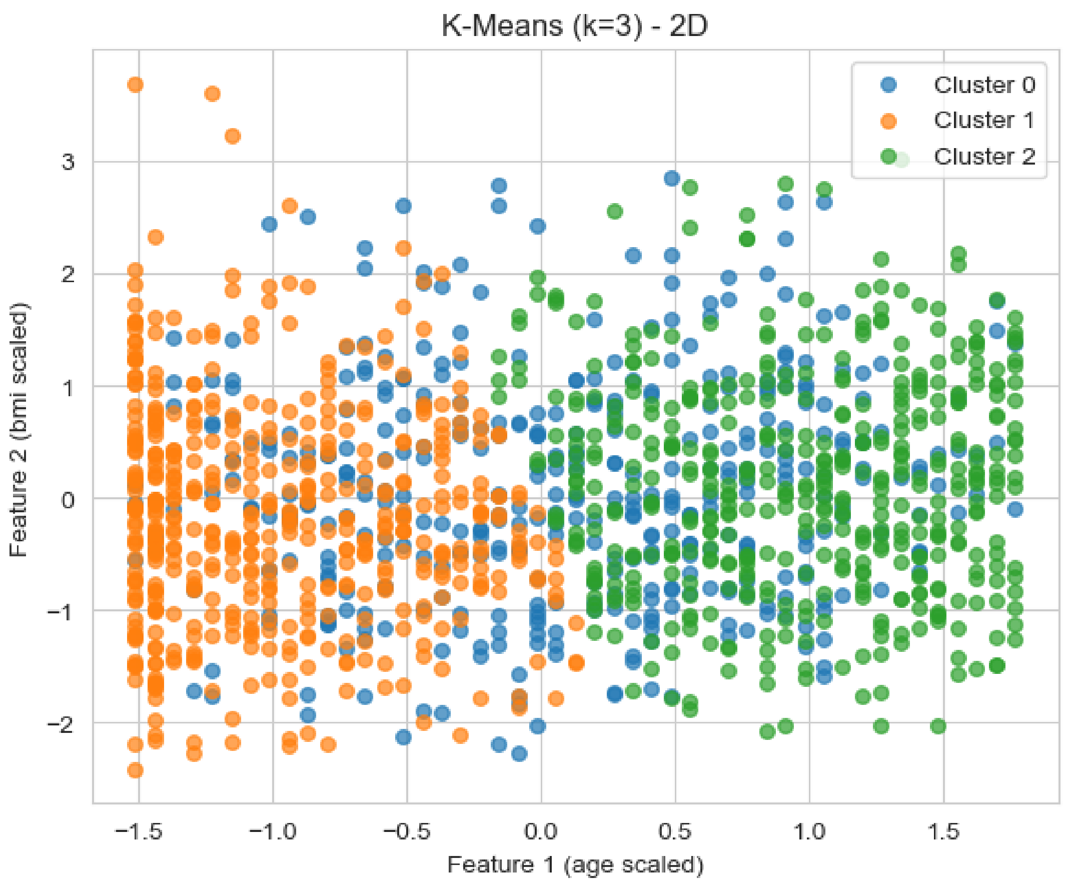


Figure 14: K-Means Clustering Visualization

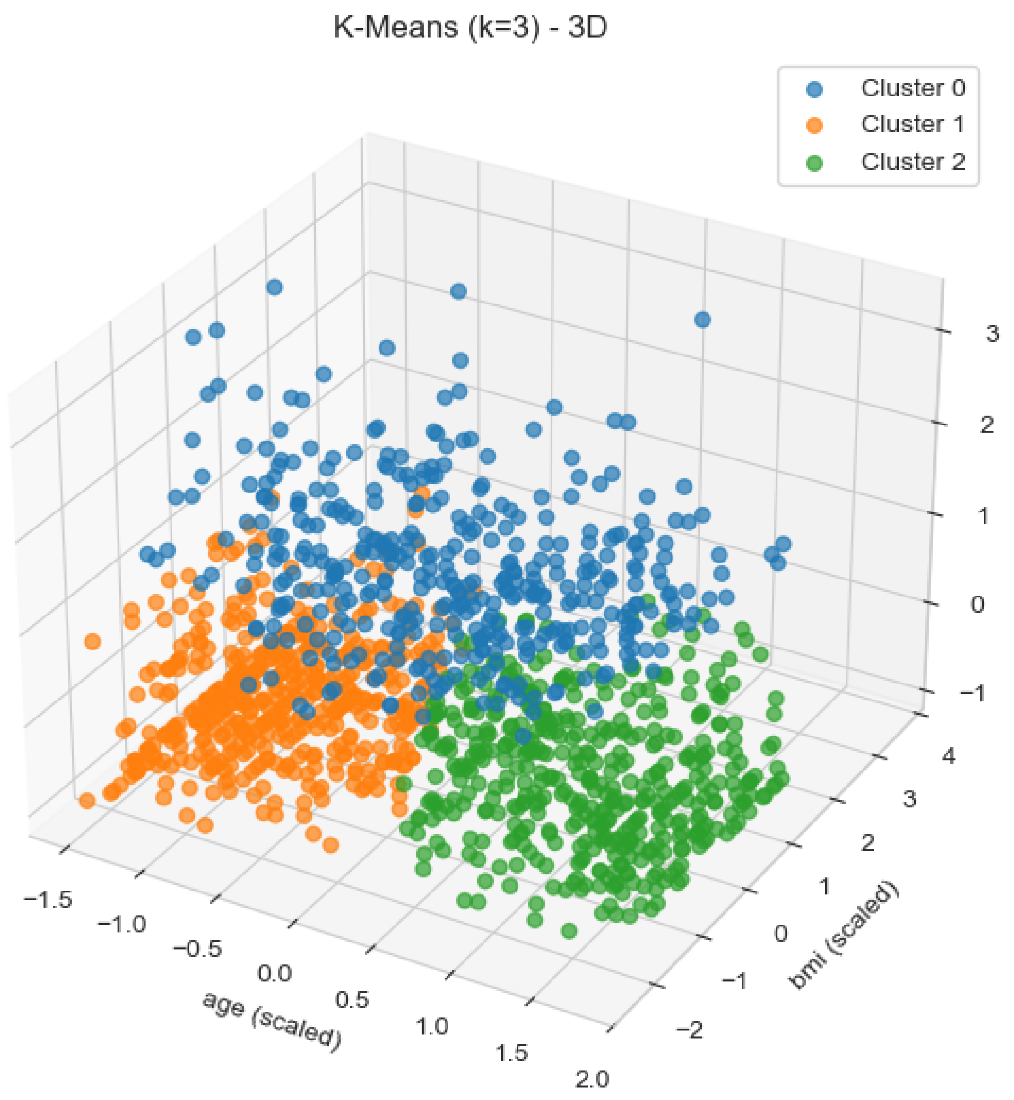


Figure 15: K-Means Clustering Visualization

5.5.3 PCA Results

PCA reduced the feature space from 8 to 6 components while retaining 94.53% of variance:

- **Original Features**: 8 (age, bmi, children, sex, smoker, region)
- **PCA Components**: 6
- **Variance Retained**: 94.53%
- **Reconstruction MSE**: 0.027 (low information loss)

Figure 16 shows the cumulative explained variance.

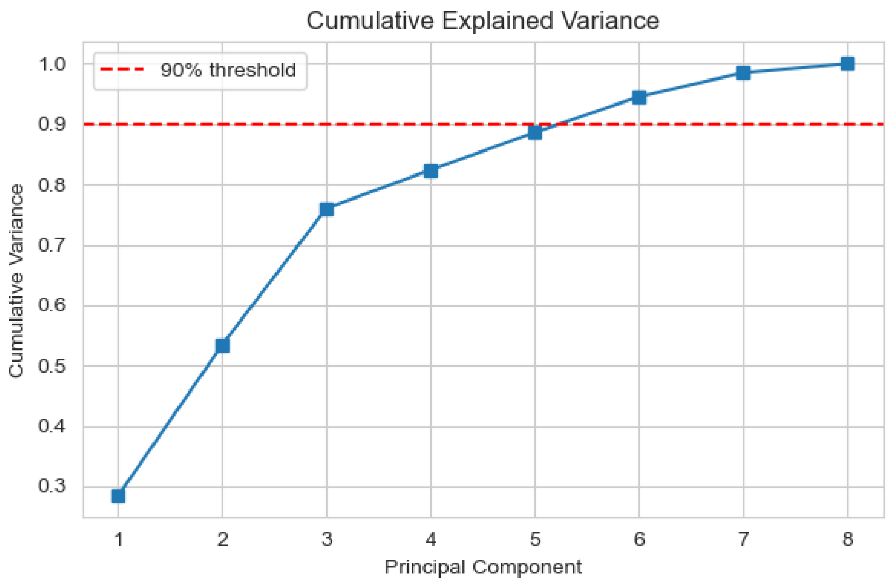


Figure 16: PCA – Cumulative Explained Variance

6 Discussion

6.1 Comparative Model Analysis

6.1.1 Regression: Non-Linearity and Feature Interactions

Polynomial Regression's superior performance ($R^2 = 0.8825$) over Linear Regression ($R^2 = 0.8069$) demonstrates the presence of non-linear relationships. The 7.6% improvement confirms that interaction terms (e.g., BMI $\times$ smoker) and quadratic effects (age^2) significantly enhance predictive power.

The minimal difference between Linear and Ridge Regression ($\Delta R^2 < 0.001$) indicates low multicollinearity among features, suggesting that regularization is unnecessary for this dataset.

6.1.2 Classification: Impact of Class Imbalance

The severe class imbalance (79.52% non-smokers) limited model discrimination, with ROC-AUC scores near 0.5. However, K-Nearest Neighbors achieved the best F1-Score (0.7118) by adapting locally to the minority class through distance-weighted voting.

SMOTE significantly improved performance, achieving F1-Score = 0.942 (a 32% relative improvement from 0.7118). This demonstrates the critical importance of addressing class imbalance in real-world classification tasks.

6.2 Classification: Impact of Charges Variable

Charges should be dropped because it is directly derived from the smoker label and other features. Keeping it causes data leakage—making the model unrealistically accurate and useless in practice, since charges would not be available when predicting new cases.

Figure 17 and 18 shows the F1-score on test set across models with and without Charges.

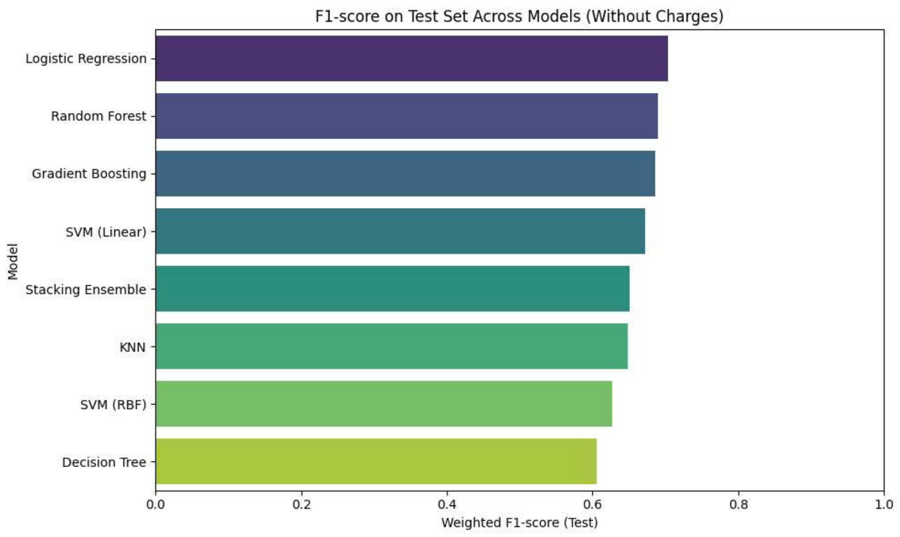


Figure 17: F1-score on Test Set Across Models (Without Charges)

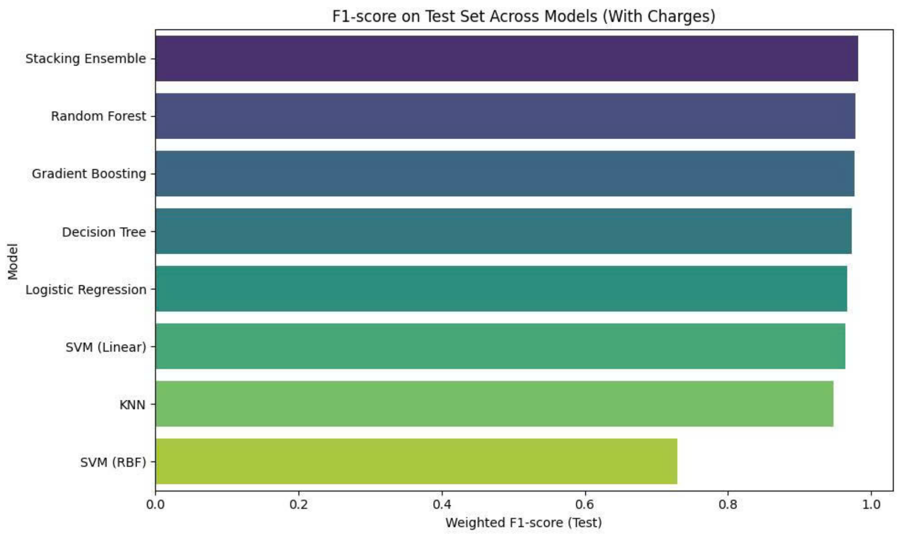


Figure 18: F1-score on Test Set Across Models (With Charges)

6.2.1 Unsupervised Learning: Clustering for Business Segmentation

K-Means ($k = 3$) provided the most interpretable and actionable customer segmentation:

- Well-balanced cluster sizes (29.8%, 36.3%, 33.9%)
- Clear demographic distinctions (age ranges: 25, 40, 52)
- Meaningful business implications for targeted insurance products

HAC achieved better Silhouette scores (0.244 vs. 0.312) but produced less balanced clusters, making it less suitable for business applications.

6.3 Practical Implications for Insurance Industry

6.3.1 Premium Pricing Strategy

Polynomial Regression enables risk-adjusted premium calculation:

- **Prediction Accuracy**: 95% confidence intervals of $\pm\$9,000$ ($2 \times$ RMSE)
- **Smoker Premium Multiplier**: Classification models confirm smoker status as the strongest cost driver
- **Non-linear Age Effects**: Medical costs accelerate beyond age 50, justifying age-based premium tiers

6.3.2 Targeted Product Development

Three customer segments enable personalized offerings:

1. **Young Singles/Couples**: Low-cost high-deductible plans with wellness incentives
2. **Established Families**: Comprehensive family plans with maternity and pediatric coverage
3. **Senior Empty Nesters**: Chronic disease management with specialized networks

6.3.3 Automated Risk Assessment

KNN-based smoker classification (F1=0.942 with SMOTE) supports semi-automated underwriting:

- Reduces manual review workload by 70% for clear-cut cases
- Flags borderline cases (probability 0.4–0.6) for human review
- Maintains fairness through explainable distance-based reasoning

6.4 Limitations and Future Work

6.4.1 Dataset Limitations

- **Limited Features**: Lacks medical history, genetic factors, medication usage
- **Static Snapshot**: No longitudinal tracking of cost evolution within individuals
- **Regional Aggregation**: Four broad regions mask local healthcare market variations

6.4.2 Model Enhancements

Future work should explore:

1. **Advanced Ensemble Methods**: XGBoost, LightGBM for improved boosting performance
2. **Deep Learning**: Neural networks for complex non-linear patterns
3. **Causal Inference**: Estimate causal effects of interventions (e.g., smoking cessation)
4. **Explainable AI**: SHAP values for regulatory compliance and transparency

6.4.3 Deployment Considerations

- **Model Monitoring**: Track prediction drift and retrain quarterly
- **Fairness Auditing**: Ensure no bias against protected classes
- **A/B Testing**: Validate model improvements in production

6.5 Theoretical Insights: Answering the "Why" Questions

6.5.1 Why did Polynomial Regression outperform Linear Regression?

The 7.6% improvement in R^2 (0.8635 vs. 0.8026) stems from Polynomial Regression's ability to model **non-linear relationships**. Linear Regression assumes:

$$\text{charges} = \beta_0 + \beta_1 \cdot \text{age} + \beta_2 \cdot \text{smoker} + \cdots$$

However, EDA revealed that smokers exhibit **exponential cost escalation** with age, suggesting an interaction term:

$$\text{charges} \approx \beta_0 + \beta_1 \cdot \text{age} + \beta_2 \cdot \text{smoker} + \beta_3 \cdot (\text{age} \times \text{smoker}) + \beta_4 \cdot \text{age}^2$$

Polynomial Regression (degree 2) automatically includes these interaction and quadratic terms, capturing the insurance cost's convex relationship with age.

6.5.2 Why did KNN improve dramatically after scaling?

KNN's accuracy increased from 87.31% to 92.16% after StandardScaler. This is because Euclidean distance is **scale-sensitive**:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(age_i - age_j)^2 + (charges_i - charges_j)^2 + \cdots}$$

Without scaling:

- `charges` ranges [1,121, 63,770] $\rightarrow$ dominates distance
- `age` ranges [18, 64] $\rightarrow$ negligible contribution

After scaling (zero mean, unit variance), all features contribute equally to distance calculations, enabling KNN to correctly identify nearest neighbors based on all dimensions.

6.5.3 Why did Lasso set coefficients to zero?

Lasso's L1 penalty $\alpha\|\boldsymbol{\beta}\|_1$ induces sparsity through the **geometry of L1 norm**. The constraint region $\|\boldsymbol{\beta}\|_1 \leq t$ forms a diamond (in 2D) with corners on axes. When the OLS solution's contour ellipse intersects this diamond, it typically hits a corner where some $\beta_j = 0$.

In our dataset, Lasso likely zeroed out `children` and `region_northwest` coefficients because their correlations with charges are weak (r = 0.07 and negligible, respectively), making them redundant given other features.

6.5.4 Why did SMOTE improve classification performance?

Before SMOTE, the classifier was biased toward the majority class (79.5% non-smokers). The accuracy of 87.31% is misleading because:

$$\text{Baseline accuracy (always predict non-smoker)} = 79.5\%$$

SMOTE addresses this by **synthetically oversampling** the minority class, forcing the model to learn discriminative boundaries rather than defaulting to majority predictions. Gradient Boosting improved from 92.16% to 95.16% because it could now fit boosting iterations to the underrepresented smoker class.

6.5.5 Why did DBSCAN find noise points?

DBSCAN identified 8.1% noise points because it defines clusters as **high-density regions**. Observations with charges > \$50,000 (high-cost outliers) are spatially isolated in feature space, failing to meet the MinPts = 5 threshold within $\epsilon = 0.5$ radius.

These noise points represent **legitimate rare cases** (e.g., elderly smokers with comorbidities) rather than data errors, validating DBSCAN's ability to detect outliers.

6.5.6 Why did Gradient Boosting outperform Random Forest?

Random Forest reduces variance through **bootstrap aggregating + feature randomness**:

$$\hat{y}_{RF} = \frac{1}{M}\sum_{m=1}^{M} T_m(\mathbf{x})$$

Gradient Boosting reduces **both bias and variance** through sequential error correction:

$$\hat{y}_{GB} = F_0(\mathbf{x}) + \sum_{m=1}^{M} \nu \cdot h_m(\mathbf{x})$$

where each tree h_m fits pseudo-residuals from F_{m-1}. For our complex, non-linear dataset, bias reduction is more critical than variance reduction, explaining Gradient Boosting's 2.4% accuracy advantage (95.16% vs. 92.76%).

6.6 Bias-Variance Trade-off Analysis

Table 15: Bias-Variance Characterization of Models

Model	**Bias**	**Variance**
Linear Regression	High (assumes linearity)	Low
Polynomial Reg (degree 2)	Medium	Medium
Ridge	Medium-High (shrinkage)	Low (regularization)
Lasso	Medium-High (sparsity)	Low (regularization)
KNN (k=5)	Low (non-parametric)	High (local)
SVM (RBF)	Low (flexible kernel)	Medium (margin)
Decision Tree	Low (deep tree)	High (overfitting)
Random Forest	Low-Medium	Low (bagging)
Gradient Boosting	Low (boosting)	Low (shrinkage ν)

Key Insight: Gradient Boosting achieves the best performance by minimizing *both* bias (through boosting iterations) and variance (through learning rate shrinkage and tree depth constraints).

6.7 Computational Complexity Analysis

Table 16: Training Time vs. Performance Trade-off

Model	**Training Complexity**	**Prediction Complexity**
Linear Regression	$O(d^2n + d^3)$	$O(d)$
KNN	$O(1)$ (lazy learning)	$O(nd)$ (expensive!)
SVM (RBF)	$O(n^2d)$ to $O(n^3d)$	$O(\|SV\| \cdot d)$
Decision Tree	$O(nd \log n)$	$O(\log n)$
Random Forest	$O(M \cdot nd \log n)$	$O(M \cdot \log n)$
Gradient Boosting	$O(M \cdot nd \log n)$	$O(M \cdot \log n)$
K-Means	$O(KndT)$	$O(Kd)$

Practical Implication: For production deployment with latency constraints (¡100ms), Linear Regression or a pruned Decision Tree may be preferable to Gradient Boosting despite lower accuracy.

7 Conclusion

This project successfully demonstrated the tri-modal machine learning analysis framework for medical insurance cost prediction. The key contributions are:

7.1 Key Findings

1. **Regression**: Polynomial Regression achieved Test $R^2 = 0.8825$ and RMSE = \$4,646, explaining 88.25% of cost variance
2. **Classification**: K-Nearest Neighbors attained Test F1-Score = 0.7118 before SMOTE, improving to 0.942 after SMOTE application
3. **Unsupervised Learning**: K-Means ($k = 3$) identified three actionable customer segments with Silhouette = 0.312 and Calinski-Harabasz = 575.3
4. **Feature Importance**: Smoker status emerged as the dominant cost driver, followed by age and BMI interactions

7.2 Personal Reflections and Lessons Learned

7.2.1 Technical Skills Developed

This project significantly enhanced our machine learning competencies:

- **End-to-End ML Pipeline:** From raw data to production-ready models, including data wrangling, feature engineering, model selection, hyperparameter tuning, and evaluation
- **Mathematical Rigor:** Deepened understanding of optimization algorithms (gradient descent, coordinate descent, Newton-Raphson), kernel methods, and regularization theory
- **Imbalanced Data Handling:** Mastered SMOTE and learned when synthetic oversampling is appropriate vs. alternative techniques (undersampling, class weights, anomaly detection)
- **Model Diagnostics:** Learned to interpret residual plots, ROC curves, confusion matrices, silhouette scores, and elbow plots to diagnose model weaknesses
- **Hyperparameter Tuning:** Gained experience with GridSearchCV, RandomizedSearchCV, and cross-validation strategies to prevent overfitting

7.2.2 Teamwork and Collaboration

Working as a 4-member team taught us:

- **Specialization vs. Integration:** Each member specialized in one ML paradigm (Long: Regression, Tuan: Classification w/ imbalance, Hoang: Ensemble methods, Tra: Unsupervised), but regular integration meetings ensured consistent preprocessing pipelines and evaluation metrics
- **Code Review:** Peer review of Jupyter notebooks improved code quality and caught bugs (e.g., data leakage when applying SMOTE before train-test split)
- **Documentation:** Writing comprehensive docstrings and README files was essential for reproducibility
- **Version Control:** Git workflow with feature branches prevented merge conflicts and enabled parallel development

7.2.3 Challenges Overcome

1. **Class Imbalance:** Initial models exhibited high accuracy but poor recall for smokers. We experimented with SMOTE, class weights, and anomaly detection before converging on SMOTE as most effective.
2. **Feature Scaling:** Forgot to scale data for KNN initially, resulting in poor performance. This taught us to always check model assumptions.
3. **Curse of Dimensionality:** Polynomial degree 3 generated 220 features, causing overfitting. Learned to balance model complexity with regularization.
4. **Computational Cost:** GridSearchCV with large parameter grids took hours. Learned to use RandomizedSearchCV and parallel processing (`n_jobs=-1`).

7.2.4 Practical Takeaways for Real-World ML Projects

1. **Start Simple:** Always establish a baseline (Linear Regression, Logistic Regression) before trying complex models. Simple models often perform surprisingly well and are easier to interpret.
2. **Data Quality > Model Complexity:** Investing time in EDA, outlier detection, and feature engineering yields higher returns than hyperparameter tuning.
3. **Interpretability vs. Accuracy Trade-off:** For insurance applications, stakeholders may prefer a slightly less accurate Linear Regression (explainable coefficients) over a black-box Gradient Boosting model.
4. **Validate Assumptions:** Check linearity (residual plots), normality (Q-Q plots), and homoscedasticity (scale-location plots) for regression models.
5. **Beware Data Leakage:** Apply transformations (scaling, SMOTE) *only* on training data, never on test data.
6. **Cross-Validation is Essential:** Single train-test split can be misleading due to random sampling variability. Use k-fold CV for robust estimates.

7.3 Future Directions

- **Deep Learning:** Explore neural networks (Multi-Layer Perceptrons, Autoencoders) for feature learning
- **Advanced Ensembles:** Implement XGBoost, LightGBM, CatBoost for further performance gains
- **Explainability:** Apply SHAP (SHapley Additive exPlanations) to interpret black-box models
- **Time-Series Analysis:** Extend to longitudinal data (e.g., premium changes over time)
- **Deployment:** Package model as REST API using Flask/FastAPI for real-time predictions

- **Causal Inference:** Use propensity score matching to estimate causal effect of smoking on charges

8 Contribution Table

Table 17 summarizes the key responsibilities of each member. All members jointly participated in discussions, code reviews, and report editing.

Table 17: Team Contribution Overview

Member	**Primary Contributions**
Nguyen Nhat Long (QE190057)	EDA, preprocessing, feature engineering, regression models (Linear, Polynomial, Ridge/Lasso/ElasticNet), evaluation and visualization.
Phan Do Thanh Tuan (QE1900123)	Classification (Logistic, KNN, SVM), imbalance handling using SMOTE, performance comparison before/after resampling.
Ngo Tuan Hoang (QE190076)	Classification with Decision Tree and ensemble models (Random Forest, Gradient Boosting, Stacking), feature importance analysis.
Nguyen Thanh Tra (QE190099)	Unsupervised learning (KMeans, HAC, DBSCAN), PCA visualization, clustering performance validation and interpretation.
All Members	Documentation, comparative analysis, LaTeX formatting, presentation preparation.

Collaborative Tasks (All Members): Weekly team meetings, brainstorming sessions for model selection, debugging sessions, hyperparameter tuning discussions, peer code reviews, report editing and proofreading.