Wine Quality Prediction

Parsa Hajiannejad

Summary

- In this project, I worked on predicting wine quality by classifying wines as either "good" or "bad" based on 11 properties. The dataset contained 6497 samples, and I handled issues like outliers and class imbalance using SMOTE. I scaled the features and applied transformations like Box-Cox to stabilize variance and reduce skewness.

- I implemented two models from scratch: Logistic Regression and Support Vector Machine (SVM), testing both in linear and polynomial forms. Polynomial features helped capture non-linear relationships which improved the models' performance. After implying 5-fold cross-validation, I found that Logistic Regression with polynomial features performed the best, with an accuracy of 0.76 on the test set, while SVM with polynomial features performed slightly lower but had higher precision.

Steps

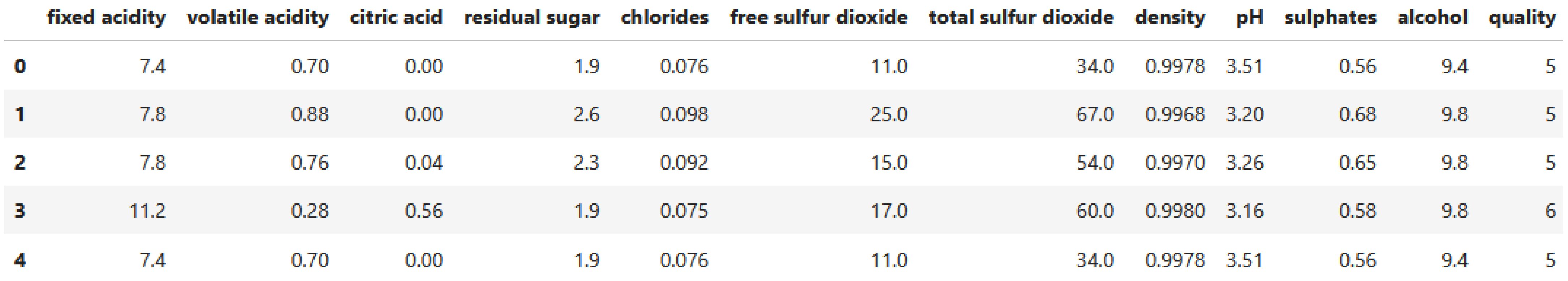
1. **Data Preparation:**
   - I cleaned the dataset by removing duplicate rows and ensuring there were no missing values.
   - I addressed the class imbalance problem by applying SMOTE to balance the classes, ensuring fair model training.
2. **Feature Engineering and Scaling:**
   - I used all 11 physicochemical features without any additional transformations, except for scaling.
   - I applied StandardScaler to standardize the features and Box-Cox PowerTransformer to handle skewness and stabilize variance.
   - Winsorization was used to cap outliers at the 1st and 99th percentiles instead of removing them.
3. **Model Selection:**
   - I implemented Logistic Regression (LR) and Support Vector Machine (SVM) models, testing both linear and polynomial forms.
   - Polynomial features were added to capture non-linear relationships in the data.

Steps

4. **Model Evaluation:**
    - I used 5-fold cross-validation to assess the performance of both models and avoid overfitting or underfitting.
    - I compared the performance of the models based on accuracy, precision, recall, and F1 score.
5. **Final Model Performance:**
    - After testing, Logistic Regression with polynomial features was the best-performing model, achieving the highest test accuracy and balanced evaluation metrics.
    - SVM with polynomial features performed similarly but had slightly lower accuracy, though it had higher precision.

Wine Quality Dataset Overview

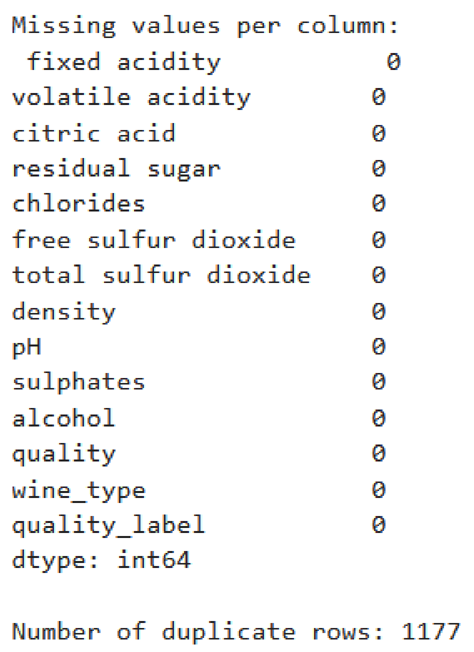
- **Dataset size**: 6497 wine samples (1599 red + 4898 white)
- **Source**: UCI Machine Learning Repository
- **Features**: 11 numeric physicochemical attributes
- **Target**: Quality score (int between 3 and 9)
- **Goal**: Classify wine as “good” or “bad” based on chemical properties



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

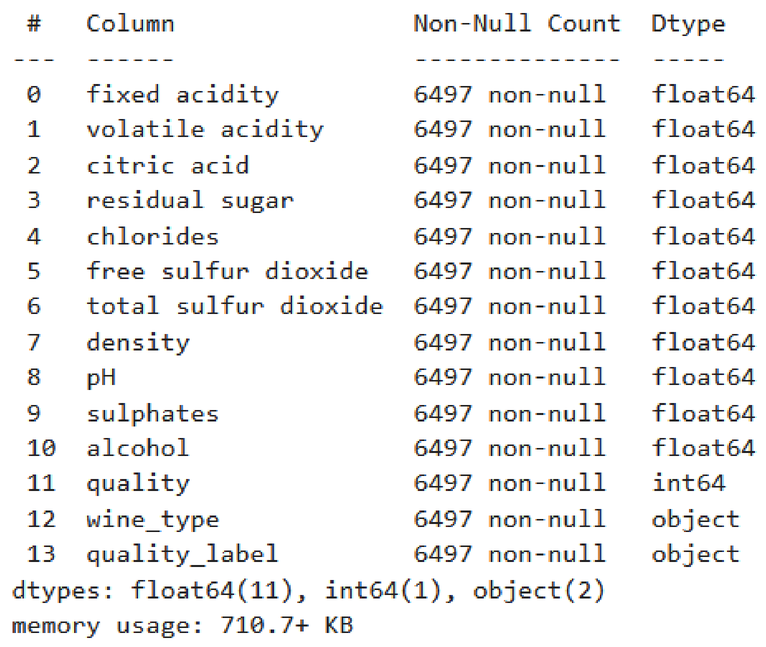
Data Integrity Checks

- **No missing values**
- **Duplicates found and removed:**
  - Red wine: 240 duplicates
  - White wine: 937 duplicates
  - Kept only unique rows
- **Data types checked and consistent**
  - All features are numerical (floats/integers)
  - No need for categorical encoding
- **Label harmonization**
  - Combined datasets added a type column: "red" or "white"



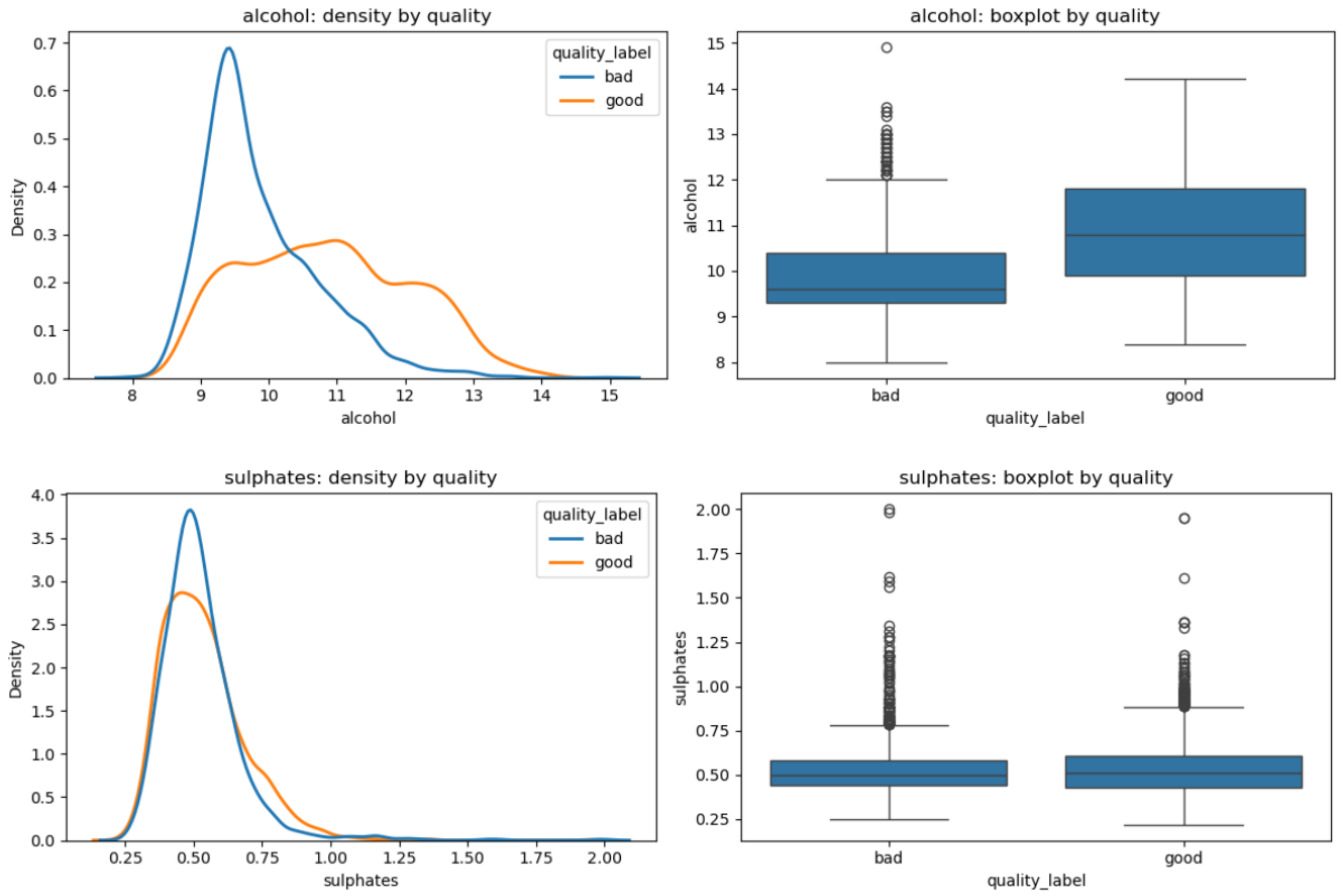
```
Missing values per column:
 fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
wine_type               0
quality_label           0
dtype: int64

Number of duplicate rows: 1177
```

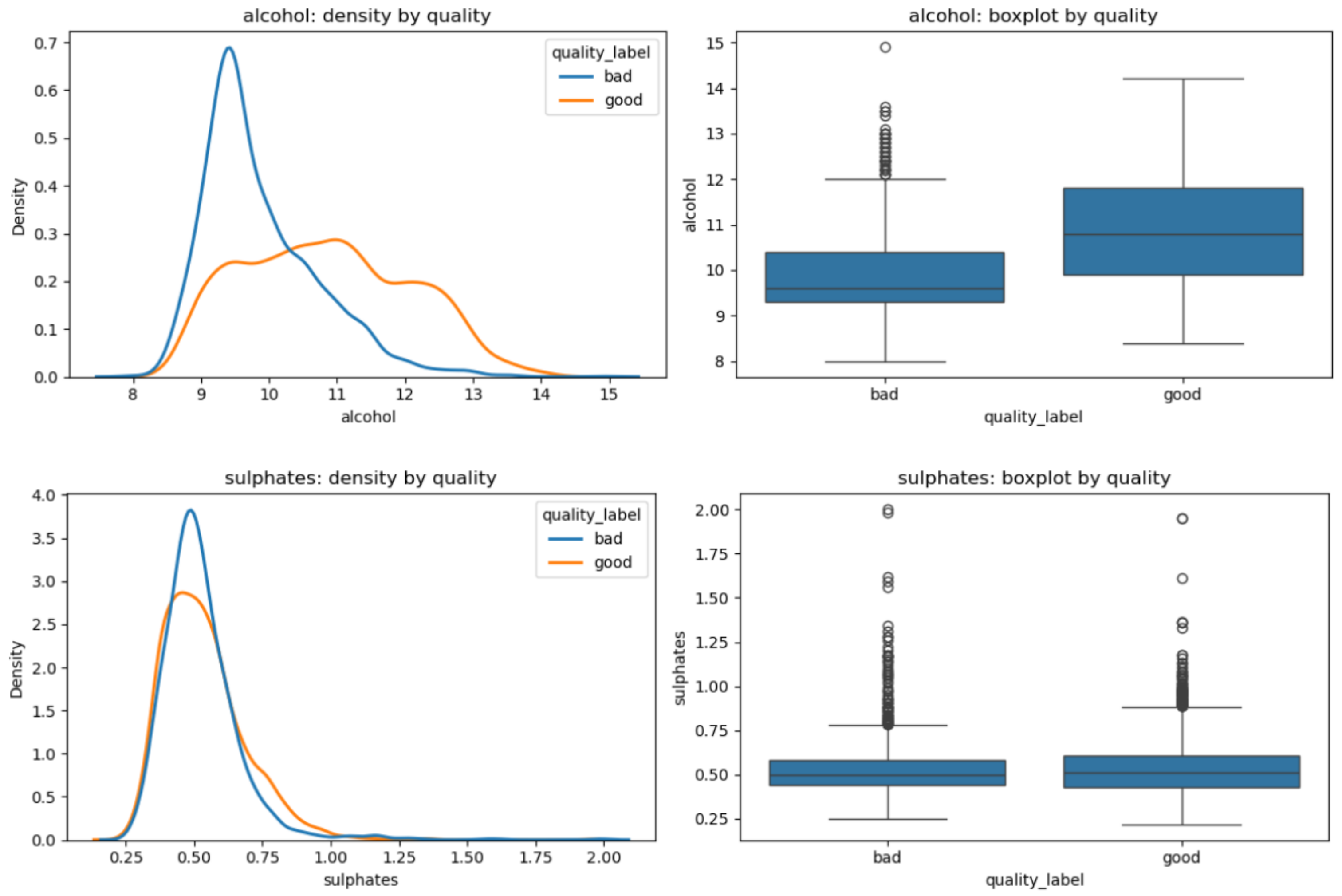


```
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         6497 non-null   float64
 1   volatile acidity      6497 non-null   float64
 2   citric acid           6497 non-null   float64
 3   residual sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free sulfur dioxide   6497 non-null   float64
 6   total sulfur dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64
 12  wine_type             6497 non-null   object
 13  quality_label         6497 non-null   object
dtypes: float64(11), int64(1), object(2)
memory usage: 710.7+ KB
```

Feature Distributions(example)



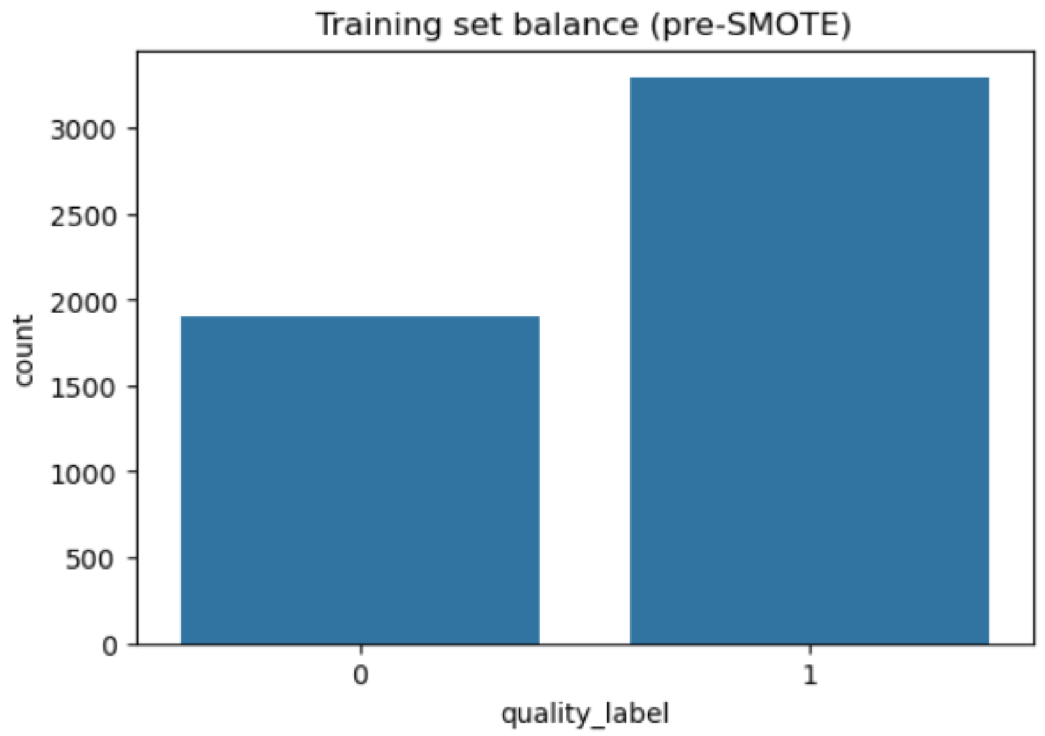
Feature Distributions(example)



Target Variable and Class Imbalance

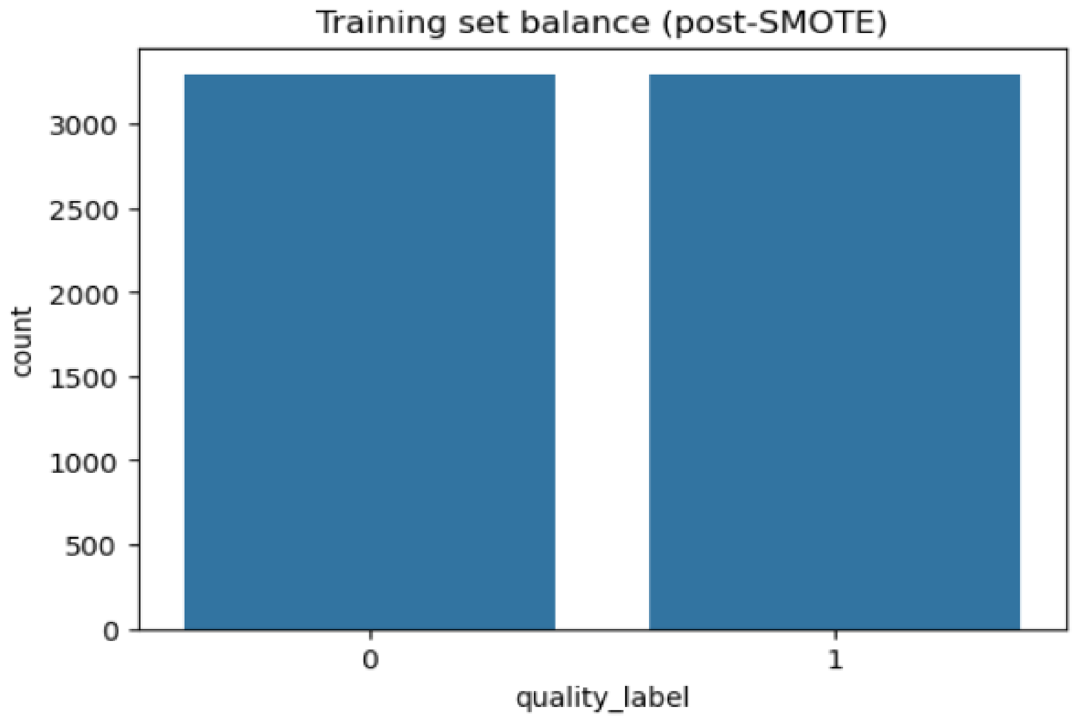
- **Target definition**
  - Quality scores (integer: 3–9) grouped into binary classes
  - Label 1 (“good”): quality ≥ 6
  - Label 0 (“bad”): quality < 6

- **Class imbalance issue:**
  - Majority of wines labeled “good” → imbalance may bias models



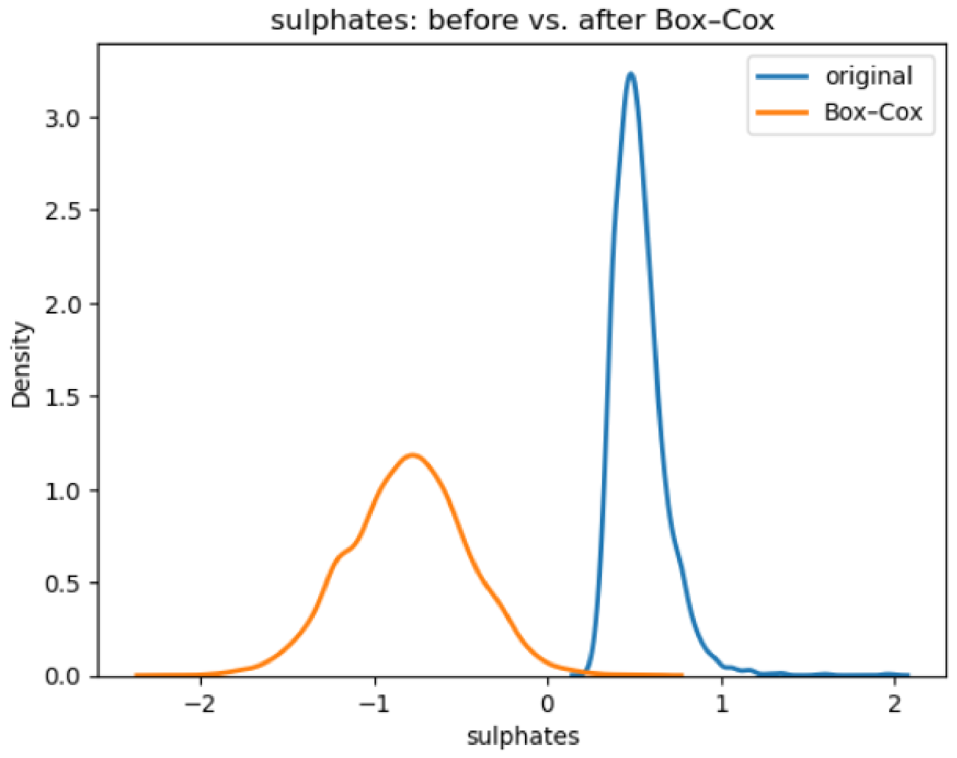
Addressing Class Imbalance (SMOTE)

- **Technique used:** SMOTE (Synthetic Minority Oversampling Technique)
  - Only applied on the **training set**, not test set
- **Oversampling parameters:**
  - k_neighbors = 5
  - Balanced both classes to the majority class size
- **Effectiveness:**
  - Transformed the skewed distribution into a balanced one
  - Essential for fair model training and evaluation



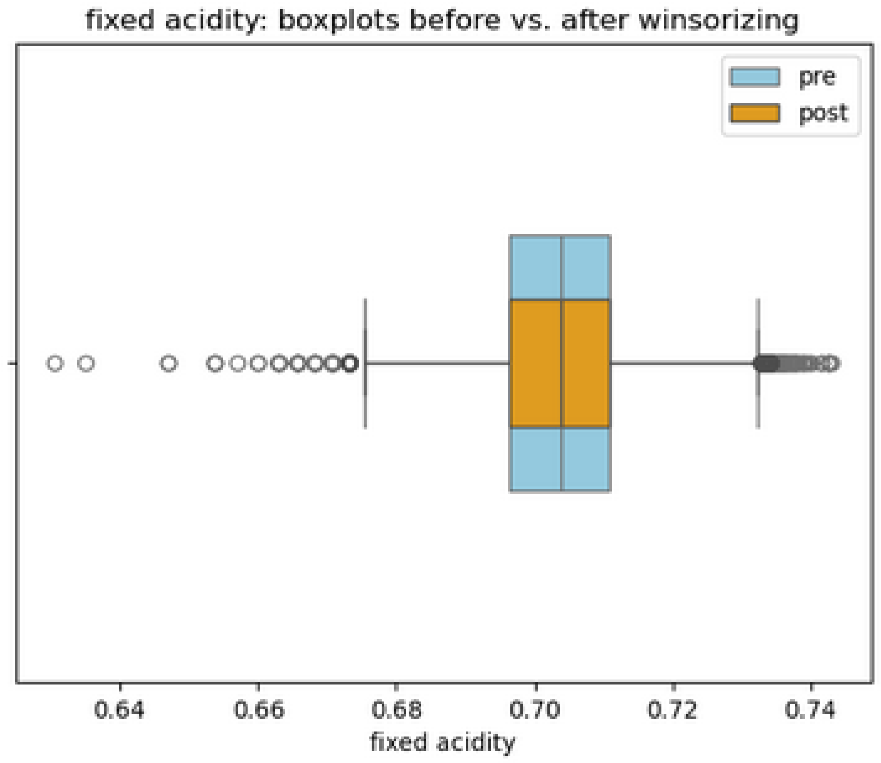
Feature Engineering and Scaling Pipeline

- **Feature selection:**
  - Used all 11 physicochemical variables (no derived features or domain-specific encoding)
- **Feature scaling strategy:**
  - Built a pipeline
  - StandardScaler: to center and scale features (mean = 0, std = 1)
  - Box-Cox PowerTransformer: Reduces skewness and stabilizes variance
- **Why this matters:**
  - Logistic regression and SVM are sensitive to feature scales
  - Helps gradient-based optimization converge faster
- **Implementation:**
  - Combined via make_pipeline() and fit only on training data (no leakage)



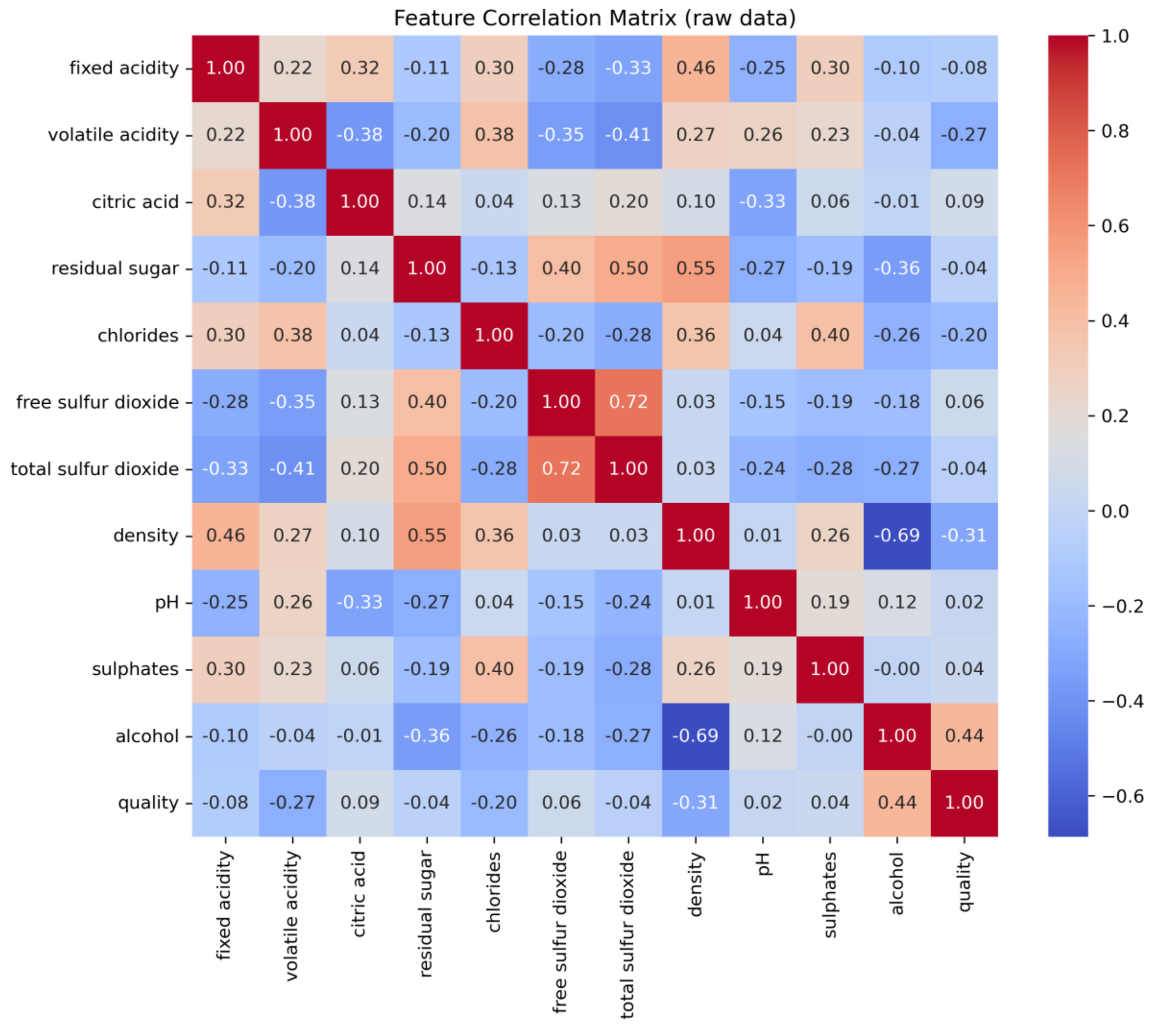
Outlier handling

- **Winsorization** is a technique where extreme values are capped at a certain percentile to reduce their influence on the dataset.(I have chosen 1st and 99st percentile)
- As an example, I have shown the impact of this method on a feature(fixed acidity)
- winsorization successfully reduces outliers, resulting in a more stable and consistent distribution without removing data points



Correlation Matrix & Feature Insights

- **Correlation heatmap** computed using Pearson coefficients
- Detected **strong correlations** among several features
- **Multicollinearity** observed in certain clusters
- Considered for potential feature reduction, but kept all 11 for interpretability



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.00	0.22	0.32	-0.11	0.30	-0.28	-0.33	0.46	-0.25	0.30	-0.10	-0.08
volatile acidity	0.22	1.00	-0.38	-0.20	0.38	-0.35	-0.41	0.27	0.26	0.23	-0.04	-0.27
citric acid	0.32	-0.38	1.00	0.14	0.04	0.13	0.20	0.10	-0.33	0.06	-0.01	0.09
residual sugar	-0.11	-0.20	0.14	1.00	-0.13	0.40	0.50	0.55	-0.27	-0.19	-0.36	-0.04
chlorides	0.30	0.38	0.04	-0.13	1.00	-0.20	-0.28	0.36	0.04	0.40	-0.26	-0.20
free sulfur dioxide	-0.28	-0.35	0.13	0.40	-0.20	1.00	0.72	0.03	-0.15	-0.19	-0.18	0.06
total sulfur dioxide	-0.33	-0.41	0.20	0.50	-0.28	0.72	1.00	0.03	-0.24	-0.28	-0.27	-0.04
density	0.46	0.27	0.10	0.55	0.36	0.03	0.03	1.00	0.01	0.26	-0.69	-0.31
pH	-0.25	0.26	-0.33	-0.27	0.04	-0.15	-0.24	0.01	1.00	0.19	0.12	0.02
sulphates	0.30	0.23	0.06	-0.19	0.40	-0.19	-0.28	0.26	0.19	1.00	-0.00	0.04
alcohol	-0.10	-0.04	-0.01	-0.36	-0.26	-0.18	-0.27	-0.69	0.12	-0.00	1.00	0.44
quality	-0.08	-0.27	0.09	-0.04	-0.20	0.06	-0.04	-0.31	0.02	0.04	0.44	1.00

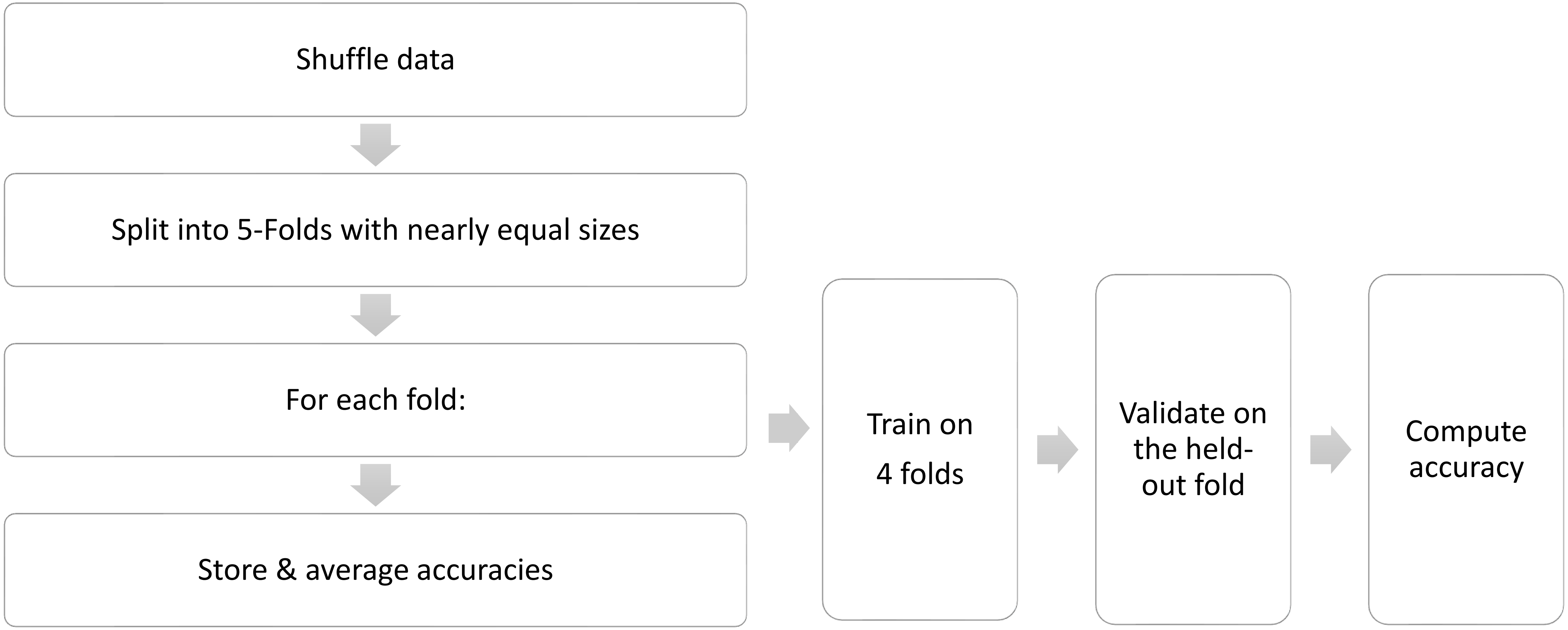
Logistic Regression: Implemented steps

- **Sigmoid:** $\sigma(z) = \dfrac{1}{1+e^{-z}}$
- **Prediction Probabilities:** $\hat{y}^{(i)} = \sigma(z^{(i)}) = \dfrac{1}{1+e^{-z^{(i)}}}$
- **Gradients with L2 Regularization:** $\dfrac{\partial L}{\partial w} = \dfrac{1}{n}\left(X^T(\hat{y}-y)+\lambda w\right)$ $\quad \dfrac{\partial L}{\partial b} = \dfrac{1}{n}\sum_{i=1}^{n}(\hat{y}^{(i)}-y^{(i)})$
- **Gradient Descent Updates:** $w := w - \eta \cdot \dfrac{\partial L}{\partial w}, \quad b := b - \eta \cdot \dfrac{\partial L}{\partial b}$
- **Log-loss with Regularization:** $\mathcal{L} = -\dfrac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right] + \dfrac{\lambda}{2n}\sum_{j=1}^{d} w_j^2$
- **Probabilities & Binary Prediction:** $P(y=1 \mid \mathbf{x}) = \sigma(\mathbf{x}\cdot\mathbf{w}+b)$ $\quad \hat{y} = \begin{cases} 1 & \text{if } \hat{p} \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases}$

Support Vector Machine: Implemented steps

- **Label Conversion:** $y \in \{0, 1\} \Rightarrow y' \in \{-1, +1\}$
- **Margin Computation:** $\text{margin}_i = y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b)$
- **Hinge Loss + Regularization:** $\mathcal{L} = \frac{1}{2}\|\mathbf{w}\|^2 + C \cdot \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y^{(i)}(\mathbf{w} \cdot \mathbf{x}^{(i)} + b))$
- **Gradient:** $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \frac{C}{n}\sum_{i \in \mathcal{I}} y^{(i)}\mathbf{x}^{(i)} \quad \frac{\partial \mathcal{L}}{\partial b} = -\frac{C}{n}\sum_{i \in \mathcal{I}} y^{(i)}$
- **Parameter Update:** $\mathbf{w} := \mathbf{w} - \eta \cdot \nabla_w \mathcal{L}, \quad b := b - \eta \cdot \nabla_b \mathcal{L}$

5-Fold Cross-Validation Diagram


Shuffle data
Split into 5-Folds with nearly equal sizes
For each fold:
Store & average accuracies
Train on 4 folds
Validate on the held-out fold
Compute accuracy

5-Fold Cross-Validation Utility

Why Cross Validation?

- **Reliable performance**

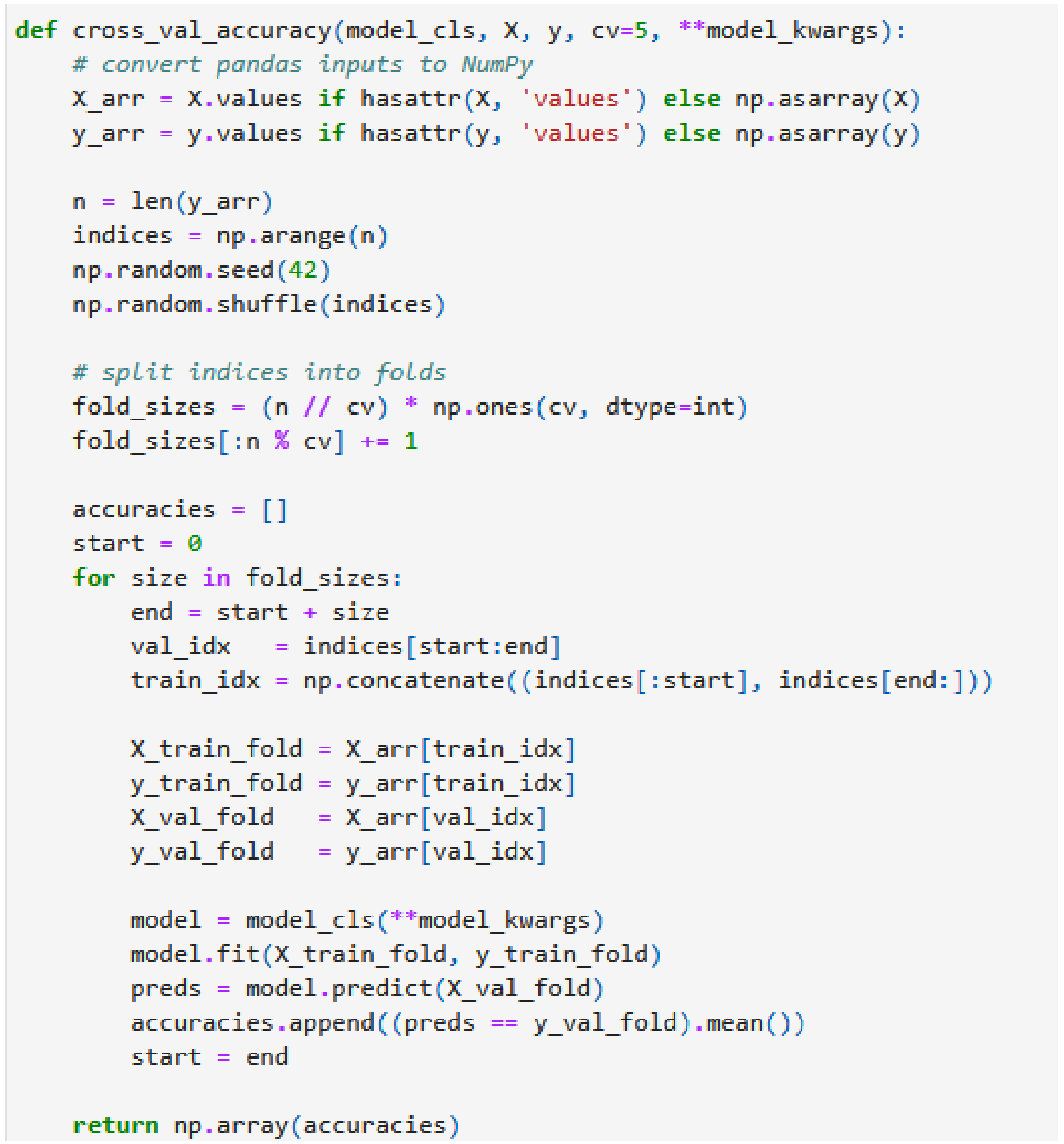
  Reduces variance vs a single train/test split
- **Hyperparameter guard**

  Prevents overfitting our tuning to one held-out set
- **Data efficiency**

  Each sample is used for validation exactly once
- **Reproducibility**

  Used np.random.seed(42) for consistent fold splits

Tuning metric: $mean(cv\_acc) = \frac{1}{5}\sum_{i=1}^{5} acc_i$



```python
def cross_val_accuracy(model_cls, X, y, cv=5, **model_kwargs):
    # convert pandas inputs to NumPy
    X_arr = X.values if hasattr(X, 'values') else np.asarray(X)
    y_arr = y.values if hasattr(y, 'values') else np.asarray(y)

    n = len(y_arr)
    indices = np.arange(n)
    np.random.seed(42)
    np.random.shuffle(indices)

    # split indices into folds
    fold_sizes = (n // cv) * np.ones(cv, dtype=int)
    fold_sizes[:n % cv] += 1

    accuracies = []
    start = 0
    for size in fold_sizes:
        end = start + size
        val_idx   = indices[start:end]
        train_idx = np.concatenate((indices[:start], indices[end:]))

        X_train_fold = X_arr[train_idx]
        y_train_fold = y_arr[train_idx]
        X_val_fold   = X_arr[val_idx]
        y_val_fold   = y_arr[val_idx]

        model = model_cls(**model_kwargs)
        model.fit(X_train_fold, y_train_fold)
        preds = model.predict(X_val_fold)
        accuracies.append((preds == y_val_fold).mean())
        start = end

    return np.array(accuracies)
```

Hyperparameter Grid Search

Logistic Regression Grid

- **Learning rates (α):** {0.1, 0.01}
  - 0.1 for faster convergence
  - 0.01 for more stable, smaller steps
- **Regularization (λ):** {0.0, 0.1}
  - 0.0 (no penalty) to test baseline fit
  - 0.1 to prevent overfitting
- **Fixed:**
  - epochs = 500

Linear SVM Grid

- **Learning rates (α):** {0.01, 0.001}
  - 0.01 to converge quickly on hinge-loss
  - 0.001 to ensure stability with large C
- **Penalty (C):** {0.1, 1.0, 10.0}
  - 0.1 enforces strong regularization (wider margin)
  - 1.0 is default balance
  - 10.0 allows more margin violations (tighter fit)
- **Fixed:**
  - epochs = 500

Procedure: evaluate mean CV accuracy over each combo; pick the highest

Logistic Regression: Mean CV Accuracy per (α, λ)

Table of Results

Chosen LR hyperparameters: α = 0.1, λ = 0.0

Learning Rate (α)	Regularization (λ)	Fold Accuracies (5-fold)	Mean CV Accuracy
0.1	**0.0**	0.745, 0.750, 0.742, 0.739, 0.744	**0.7439**
0.1	0.1	0.745, 0.750, 0.742, 0.739, 0.744	0.7439
0.01	0.0	0.728, 0.731, 0.727, 0.726, 0.727	0.7277
0.01	0.1	0.728, 0.731, 0.727, 0.726, 0.727	0.7277

- **Best setting**: α = 0.1, λ = 0.0 (Mean CV = 0.7439).
- **Effect of regularization**: Identical scores at λ = 0.0 and λ = 0.1 → small L_2 penalty has no impact at this learning rate.
- **Learning rate impact**: α = 0.1 outperforms α = 0.01 by ~1.6 , so larger step size converges more effectively for our problem.
- **Model behavior**: No signs of over-regularization here (λ up to 0.1), but too small α underfits

Linear SVM: Mean CV Accuracy per (α, C)

Chosen SVM hyperparameters: α = 0.01, C = 10.0

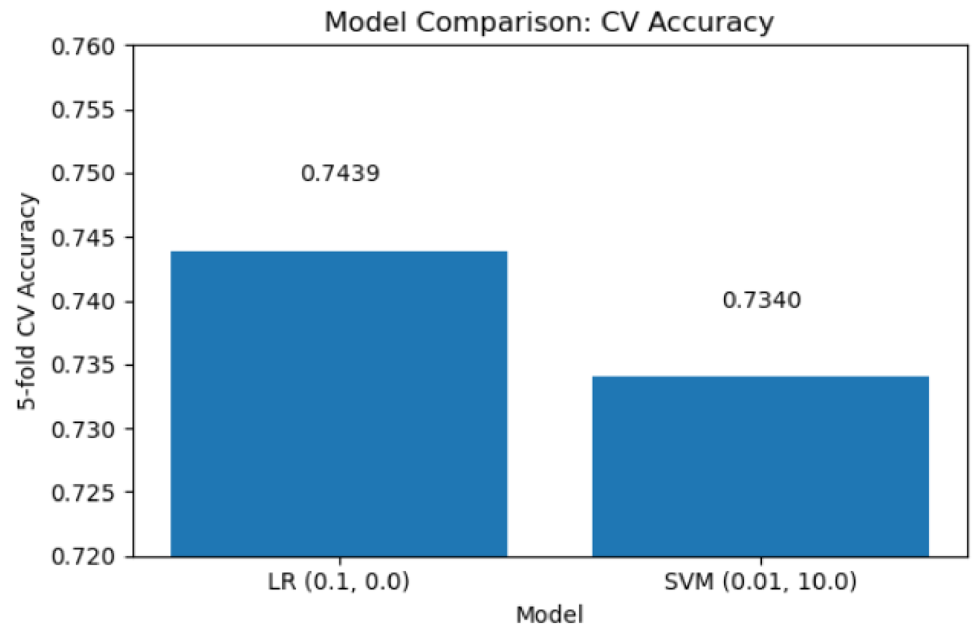
Table of Results

Learning Rate (α)	Penalty(C)	Mean CV Accuracy
0.01	0.1	0.6909
0.01	1.0	0.7040
0.01	**10.0**	**0.7340**
0.001	0.1	0.6910
0.001	1.0	0.6912
0.001	10.0	0.7277

- **Best setting:** α = 0.01, C = 10.0 (Mean CV = 0.7340).
- **Effect of C:**
  - Increasing C from 0.1 → 1.0 → 10.0 improves accuracy at α = 0.01
  - At α = 0.001, the boost from C=0.1→10.0 is smaller
- **Learning rate impact:**
  - α = 0.01 consistently outperforms α = 0.001 across C values by ~0.02–0.03 points, so a larger step size is needed to properly optimize the hinge-loss.
- **Bias–variance trade-off:**
  - Low C (0.1) underfits (strong regularization → lower scores), very high C (10.0) gives best fit without clear overfitting in CV

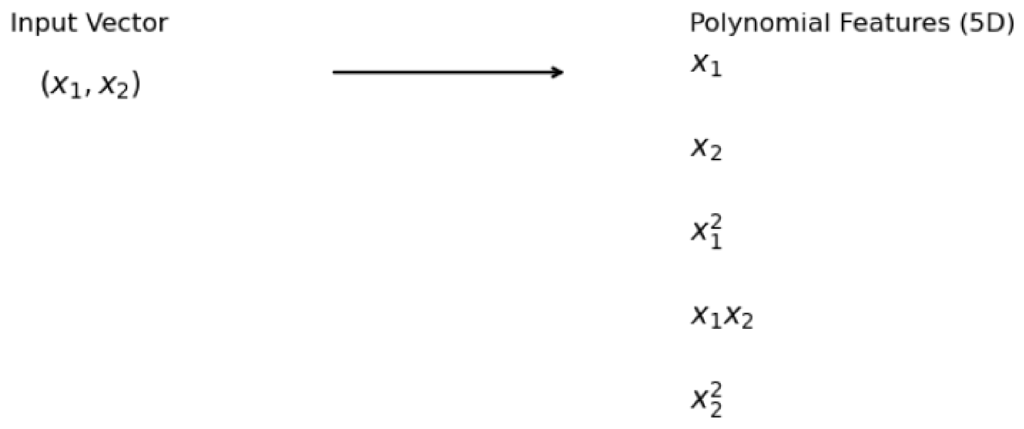
Chosen Hyperparameters

Model	Mean CV accuracy
LR (lr=0.1, λ=0.0)	0.7439
SVM (lr=0.01, C=10.0)	0.7340



Polynomial Feature Mapping

- **Motivation:** allow linear models to capture non-linear relationships
- Map original features X to higher-degree polynomial space $\varphi_d(X)$
- Example for degree 2: $\phi_2(x_i, x_j) = \{\, x_i,\ x_j,\ x_i^2,\ x_i x_j,\ x_j^2 \}$
- Implemented via PolynomialFeatures(degree=2, include_bias=False)



Logistic Regression on Polynomial Features

- Pipeline: PolynomialFeatures → StandardScaler → LogisticRegressionScratch
- Captures non-linear decision boundary
- Best CV parameters stayed at lr = 0.1, λ = 0.0

Model	CV Accuracy
LR (poly)	0.7608

SVM on Polynomial Features

- Kernel trick: compute $(\mathbf{X}^T\mathbf{X}')^d$ without explicit expansion
- I used degree 2 polynomial kernel in SVMScratch by manually mapping features
- Best CV params: lr = 0.01, C = 10.0

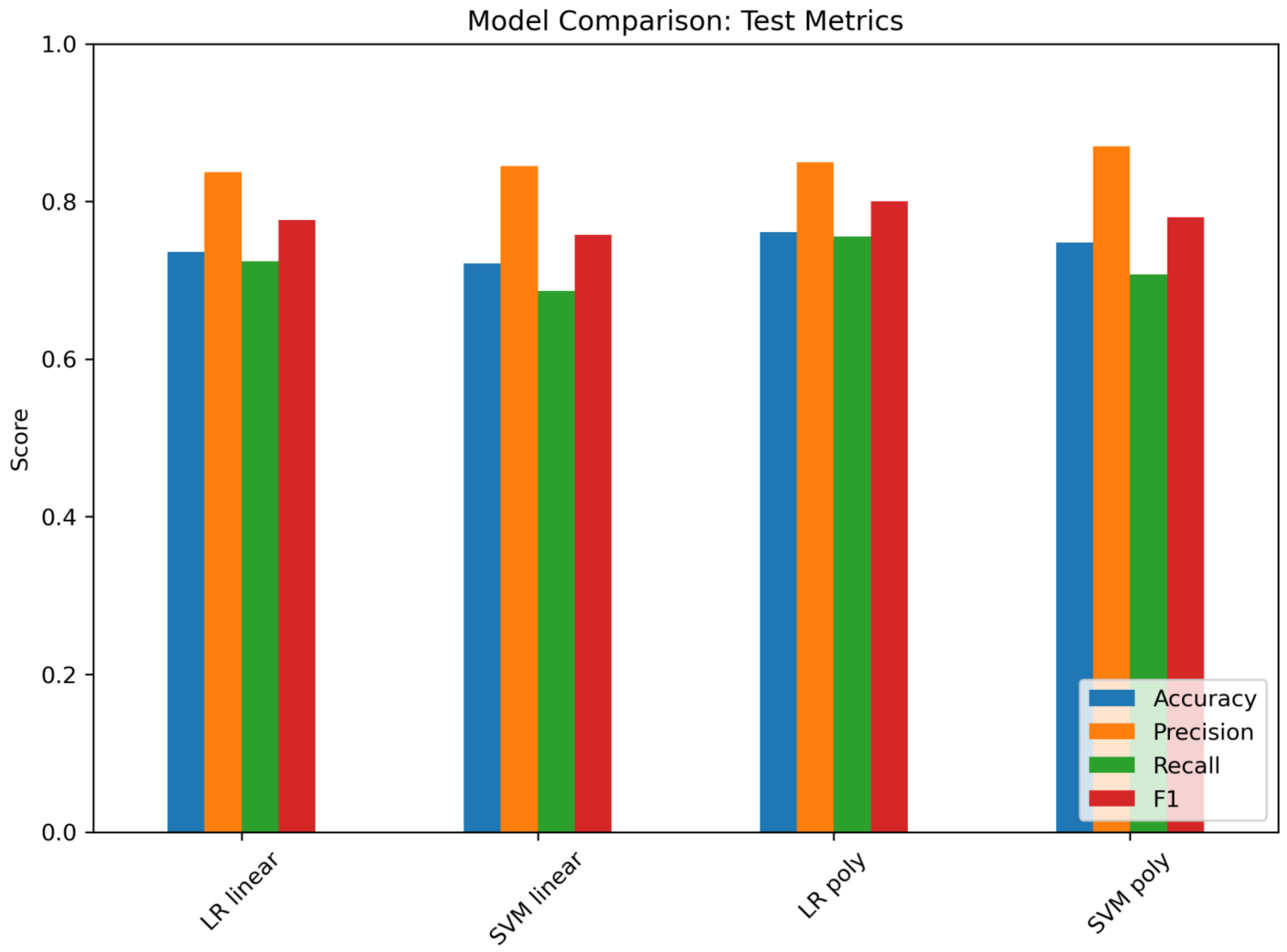
Model	CV Accuracy
SVM (poly)	0.7477

Kernel Models Comparison

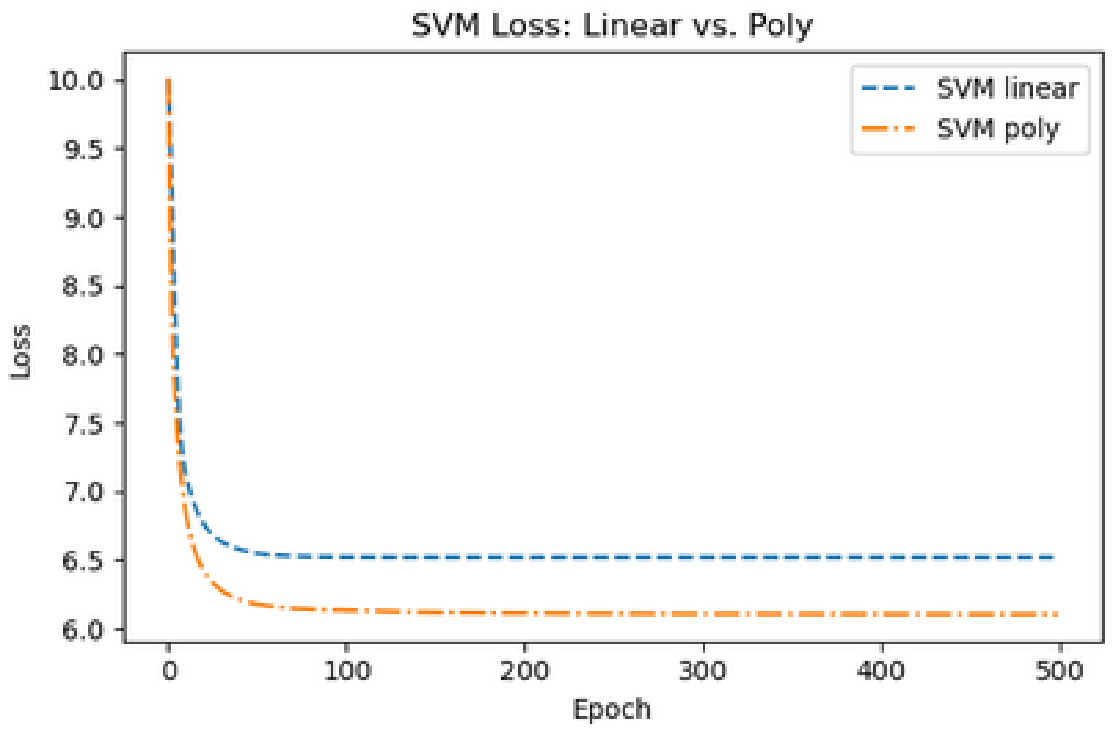
Model	CV Accuracy	Test Accuracy
LR (linear)	0.7439	0.7362
SVM (linear)	0.7340	0.7215
LR (poly)	0.7608	0.7608
SVM (poly)	0.7477	0.7477

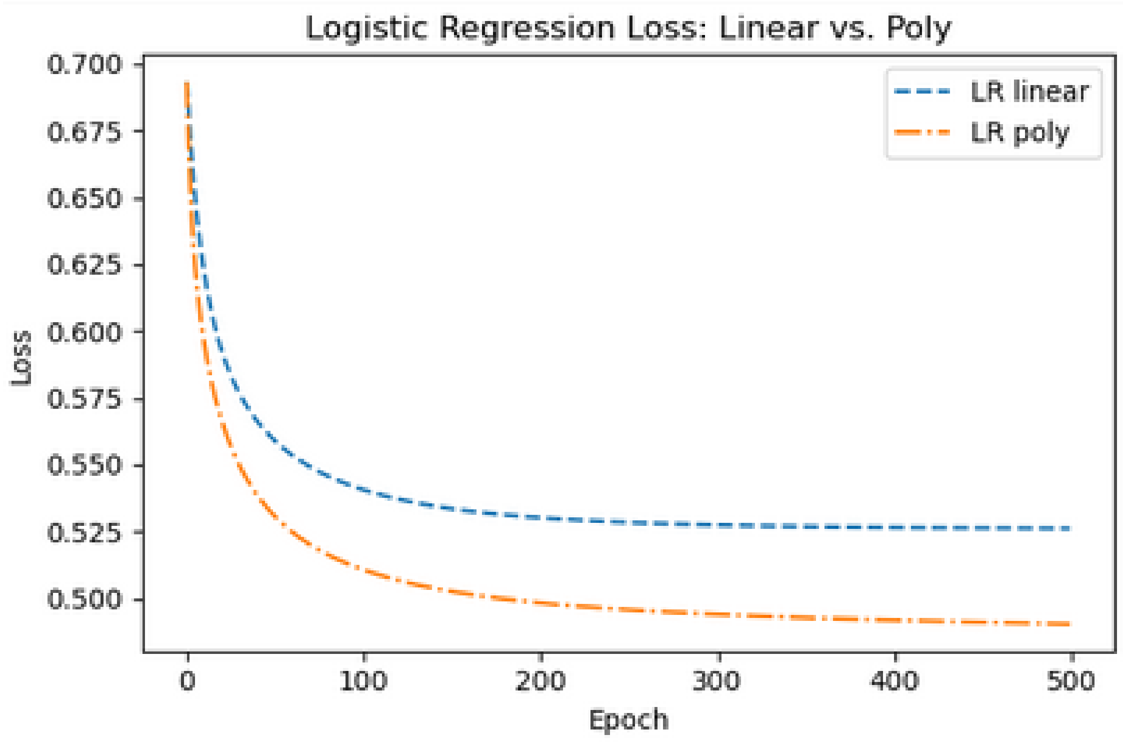
Final Model Fitting on Full Training Data

on test set	**Accuracy**	**Precision**	**Recall**	**F1 score**
LR linear	0.73	0.83	0.72	0.77
SVM Linear	0.72	0.84	0.68	0.75
LR poly	0.76	0.84	0.75	0.80
SVM poly	0.74	0.86	0.70	0.78

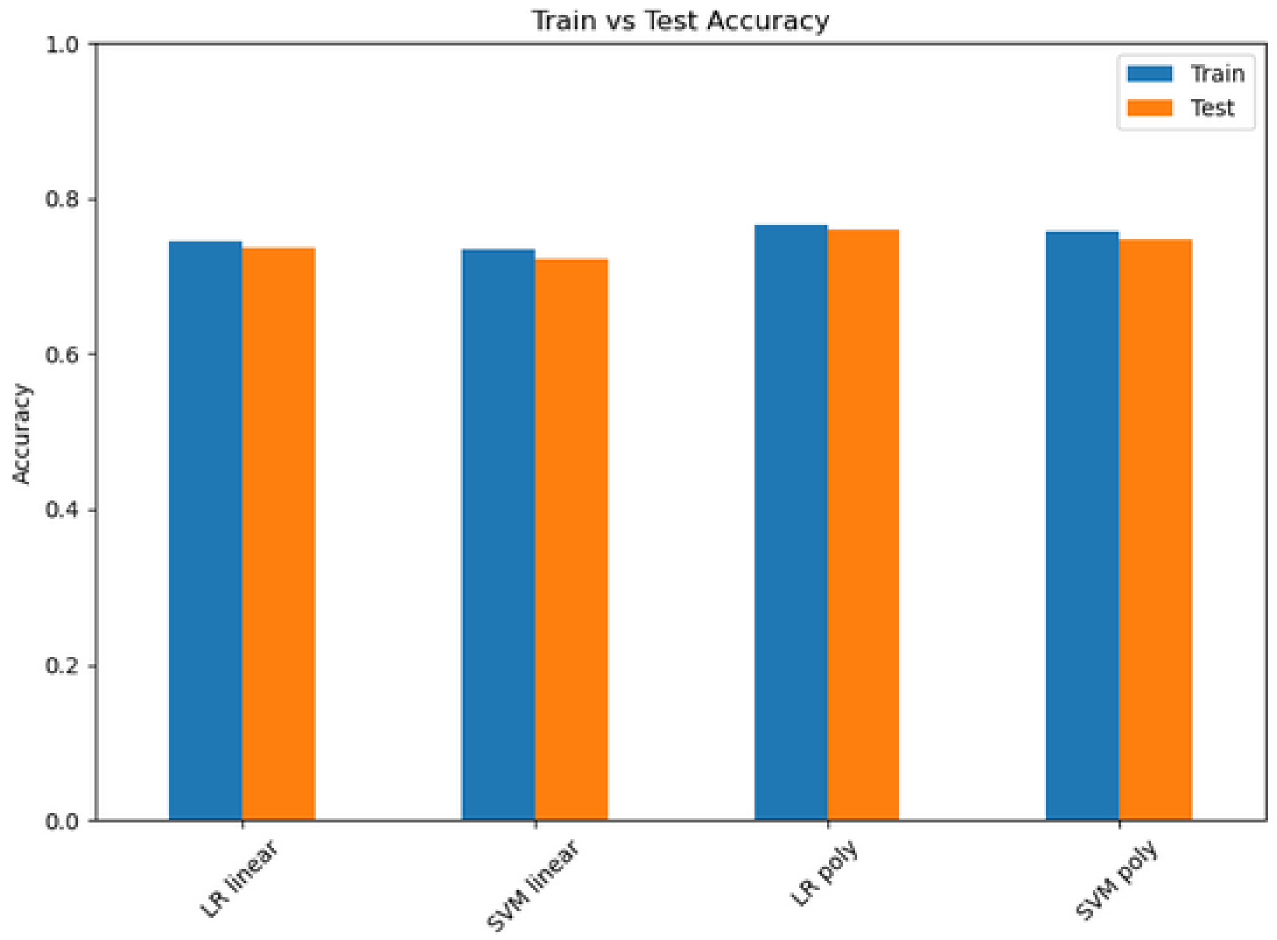


LR & SVM Losses





No significant Overfitting or Underfitting



	Train	Test
LR linear	0.74	0.73
SVM linear	0.73	0.72
LR poly	0.76	0.76
SVM poly	0.75	0.74

Conclusion

- **Data Preparation:**
  - Cleaned the dataset by removing duplicates and checking for missing values. Used SMOTE to balance the classes and address the class imbalance.
- **Feature Engineering:**
  - Used all 11 features, applied StandardScaler for scaling, and Box-Cox PowerTransformer to reduce skewness. Outliers were handled with Winsorization.
- **Model Selection:**
  - Tested both Logistic Regression (LR) and Support Vector Machine (SVM), both with linear and polynomial features. Polynomial features helped capture non-linear relationships.
- **Evaluation:**
  - Used 5-fold cross-validation to assess the models. Compared accuracy, precision, recall, and F1 score.
- **Final Model Performance:**
  - Logistic Regression (poly): Achieved 0.76 accuracy, 0.84 precision, 0.75 recall, and 0.80 F1 score.
  - SVM (poly): Achieved 0.74 accuracy, 0.86 precision, 0.70 recall, and 0.78 F1 score.

Declaration

- *I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.*