INDIAN INSTITUTE OF TECHNOLOGY DELHI

Machine Learning Project – Semester 1

Concrete Strength Prediction

and RCC Suitability Classification

Submitted To: Prof. Tanmoy Chakraborty

Team Members:

Akhil Vashistha (2025AIB2558)
Rohit Kumar (2025AIB2565)

Contents

1. Abstract

This report presents a unified machine learning pipeline for both **regression** and **classification** using the UCI Concrete Compressive Strength dataset.

For regression, the goal is to accurately predict the compressive strength (in MPa) of a concrete mix from its raw components and engineered features. For classification, a binary label `RCC_Suitable` is defined based on Indian Standard IS 456:2000: a mix is considered *Fit for Reinforced Cement Concrete (RCC)* if its compressive strength is at least 20 MPa at or after 28 days.

The pipeline includes:

- Detailed exploratory data analysis (EDA) and visualizations,
- Domain-inspired feature engineering (ratios such as water–cement, binder–aggregate, etc.),
- Custom implementations of linear, ridge and lasso regression (normal equation and coordinate descent),
- Benchmark models (SVR, XGBoost, LightGBM) for regression,
- Custom logistic regression (from scratch) and multiple scikit-learn classifiers for classification,
- Systematic hyperparameter tuning using grid search,
- Evaluation with appropriate metrics (RMSE, R^2 for regression; F1-score, ROC–AUC for classification),

We summarise the results, compare model families, and discuss limitations and possible extensions.

2. Background & Motivation

Concrete mix design strongly influences the strength and durability of structures. In practice, testing compressive strength requires casting and curing cubes/cylinders, which is time-consuming. A predictive model can:

- Provide early feedback for mix design before lab tests are complete,
- Help in sensitivity analysis of different components (cement, water, slag, etc.),
- Support non-destructive decision-making for RCC suitability.

From a machine learning perspective, this dataset is also a good playground for:

1. Regression with both linear and nonlinear models,
2. Feature engineering based on domain chemistry,
3. Demonstrating how a regression dataset can be converted into a meaningful classification task.

3. Dataset Summary

3.1 Source

The dataset is the *Concrete Compressive Strength* dataset from the UCI Machine Learning Repository (ID = 165). In the notebook, the data is directly loaded from UCI using:

```
from ucimlrepo import fetch_ucirepo
concrete = fetch_ucirepo(id=165)
```

The feature and target DataFrames are then combined:

- X: eight input variables,
- y: compressive strength (MPa).

3.2 Features

The final column names used in the notebook are:

Feature	**Description**
Cement	Cement content (kg/m^3)
Blast Furnace Slag	Slag content (kg/m^3)
Fly Ash	Fly ash content (kg/m^3)
Water	Water (kg/m^3)
Superplasticizer	Superplasticizer (kg/m^3)
Coarse Aggregate	Coarse aggregate (kg/m^3)
Fine Aggregate	Fine aggregate (kg/m^3)
Age	Age of concrete (days)
Strength	Compressive strength (MPa) – regression target
RCC_Suitable	Binary label for RCC (derived)

The dataset contains 1030 samples, with no missing values. Duplicate rows are checked and reported.

4. Exploratory Data Analysis (EDA)

A series of visualizations were used to better understand the data.

4.1 Univariate Distributions

For each feature and the target, histograms and boxplots are plotted to inspect:

- Range and typical values,
- Presence of outliers,
- Skewness (especially for `Age` and `Strength`).

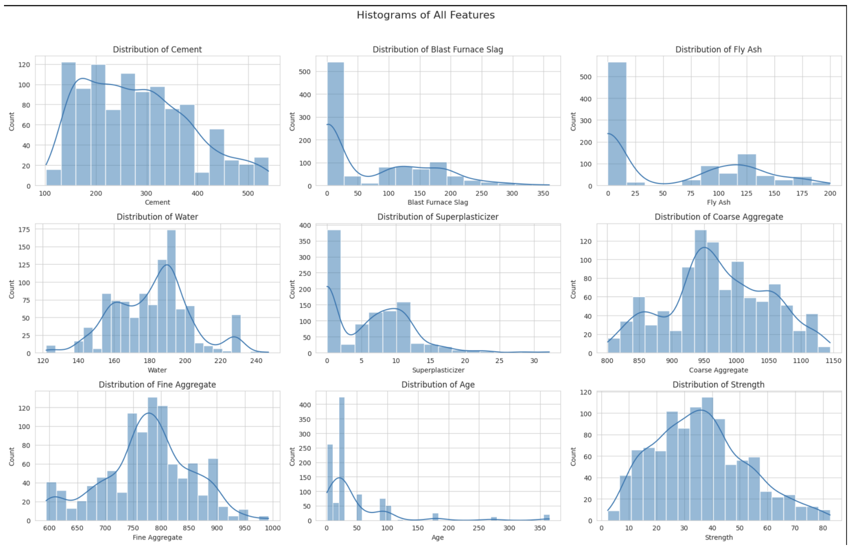


Figure 1: Histograms of all features and the target.

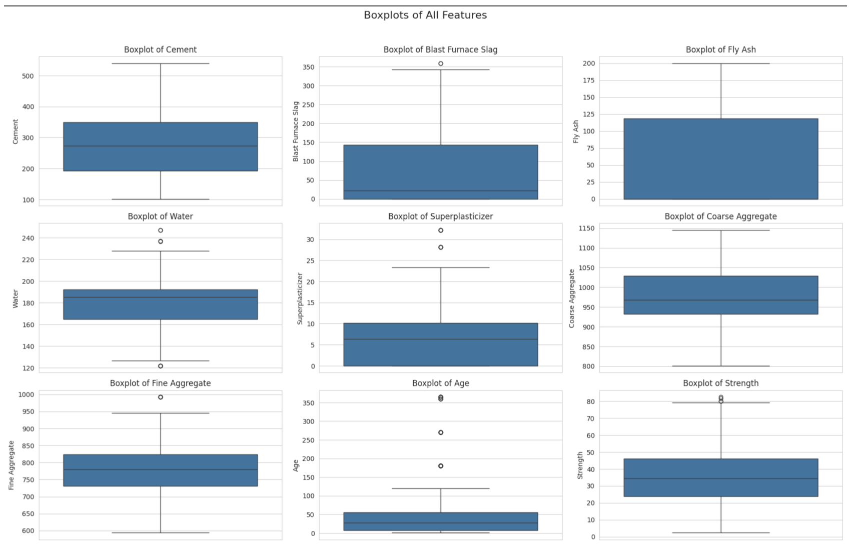
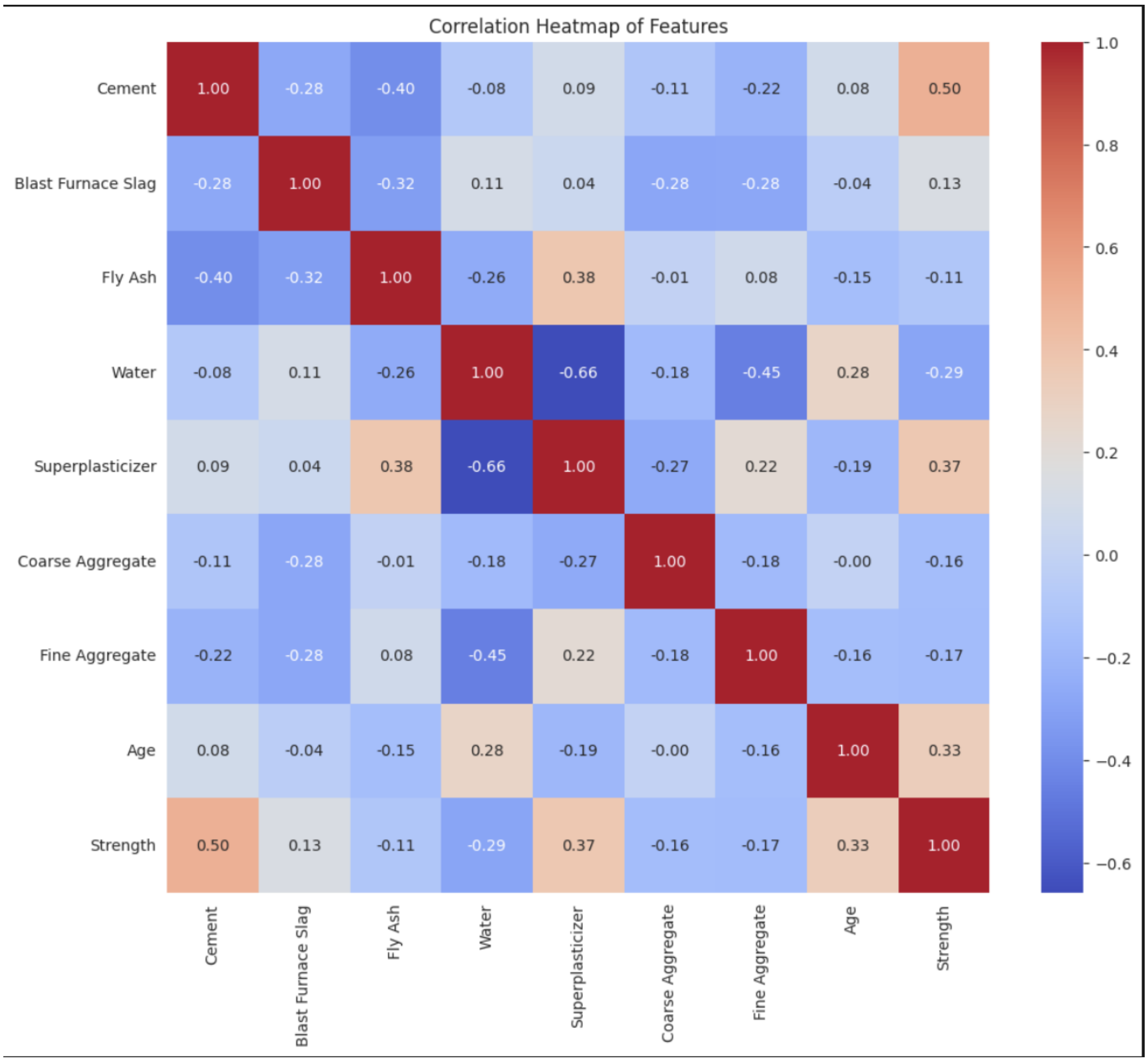


Figure 2: Boxplots of all features and the target.

4.2 Correlation Structure

A Pearson correlation heatmap was computed:



	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
Cement	1.00	-0.28	-0.40	-0.08	0.09	-0.11	-0.22	0.08	0.50
Blast Furnace Slag	-0.28	1.00	-0.32	0.11	0.04	-0.28	-0.28	-0.04	0.13
Fly Ash	-0.40	-0.32	1.00	-0.26	0.38	-0.01	0.08	-0.15	-0.11
Water	-0.08	0.11	-0.26	1.00	-0.66	-0.18	-0.45	0.28	-0.29
Superplasticizer	0.09	0.04	0.38	-0.66	1.00	-0.27	0.22	-0.19	0.37
Coarse Aggregate	-0.11	-0.28	-0.01	-0.18	-0.27	1.00	-0.18	-0.00	-0.16
Fine Aggregate	-0.22	-0.28	0.08	-0.45	0.22	-0.18	1.00	-0.16	-0.17
Age	0.08	-0.04	-0.15	0.28	-0.19	-0.00	-0.16	1.00	0.33
Strength	0.50	0.13	-0.11	-0.29	0.37	-0.16	-0.17	0.33	1.00

Figure 3: Correlation heatmap of input features and Strength.

Key observations (qualitative):

- `Cement` and `Total Binder` show positive correlation with `Strength`.
- Higher water content generally reduces strength (negative correlation).
- Age is positively correlated with strength, but with diminishing returns.

4.3 Feature vs Target Relationships

Scatter plots were created for each feature vs. `Strength`. Particularly insightful is `Age` vs `Strength`, overlaid with threshold lines used in the classification task:

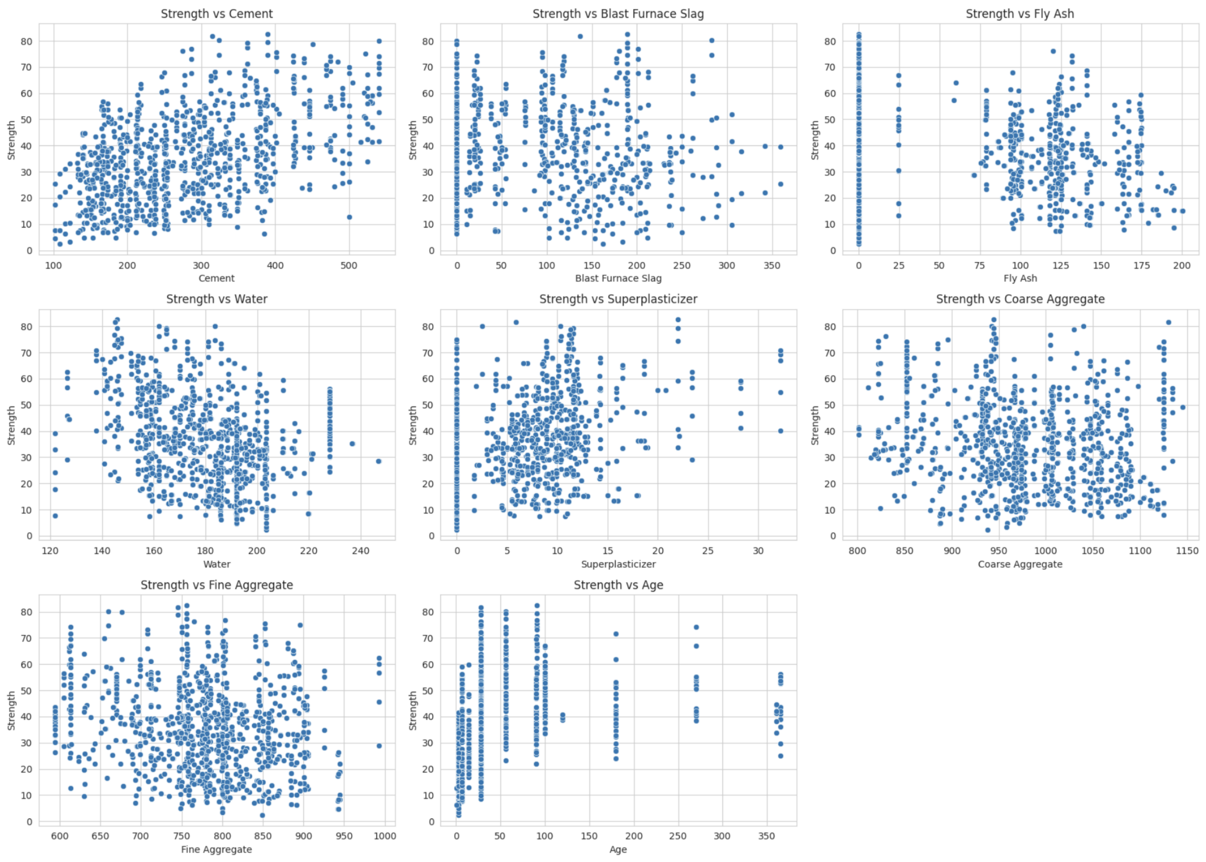


Figure 4: Age vs. Strength with RCC threshold lines (20 MPa and 28 days).

5. Feature Engineering & Preprocessing

5.1 Engineered Features

Based on domain knowledge, several ratio and interaction terms were introduced, used in both regression and classification:

- Water–cement ratios:

$$\text{Cement\_Water\_Ratio} = \frac{\text{Cement}}{\text{Water} + \varepsilon}, \quad \text{Water\_Cement\_Ratio} = \frac{\text{Water}}{\text{Cement} + \varepsilon}$$

- Binder and aggregate totals:

$$\text{Total\_Binder} = \text{Cement} + \text{Blast Furnace Slag} + \text{Fly Ash}$$

$$\text{Total\_Aggregate} = \text{Coarse Aggregate} + \text{Fine Aggregate}$$

- Ratios:

$$\text{Binder\_Aggregate\_Ratio} = \frac{\text{Total\_Binder}}{\text{Total\_Aggregate} + \varepsilon}$$

$$\text{Fine\_Coarse\_Ratio} = \frac{\text{Fine Aggregate}}{\text{Coarse Aggregate} + \varepsilon}$$

$$\text{SP\_per\_Binder} = \frac{\text{Superplasticizer}}{\text{Total\_Binder} + \varepsilon}$$

- Age transforms: $\text{Age\_Log} = \log(\text{Age} + 1)$.

For regression, additional polynomial and interaction terms were used:

- Quadratic terms: `Cement2̂`, `Water2̂`, `Age2̂`, `Binder2̂`,
- Interactions such as `Cement * Age`, `Binder * Water`, `SP * Cement`, etc.

5.2 Train–Test Splits

- Regression:
  - 70:30 train–test split,
  - 80:20 train–test split,
  - For each split, experiments are run with "linear" features and with polynomial features.
- Classification:
  - 80:20 train–test split with stratification on `RCC_Suitable`.

5.3 Scaling and Imbalance Handling

All numerical features are standardised using `StandardScaler`, fitted on training data only and applied to test data.

For classification, the training data is rebalanced using SMOTE (Synthetic Minority Oversampling Technique) on the scaled feature space to mitigate class imbalance between suitable and non-suitable mixes.

6. Regression Task – Predicting Strength

6.1 Models Implemented

6.1.1 Normal Equation (Linear Regression from Scratch)

The baseline model solves:

$$\theta = (X^\top X)^{-1} X^\top y$$

with a pseudo-inverse fallback if $X^\top X$ is singular.

Listing 1: Normal equation implementation (simplified).

```python
def normal_equation(X, y):
    X_b = add_bias(X)
    try:
        theta = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
    except np.linalg.LinAlgError:
        theta = np.linalg.pinv(X_b.T @ X_b) @ X_b.T @ y
    return theta
```

This is trained on both 70:30 and 80:20 splits, with and without polynomial features.

6.1.2 Ridge Regression (Closed-form, from Scratch)

Ridge adds an ℓ_2 penalty:

$$\theta = (X^\top X + \alpha I)^{-1} X^\top y,$$

where the intercept is typically not regularised.

A custom grid search is implemented over α to select the value with minimum test RMSE.

6.1.3 Lasso Regression (Coordinate Descent, from Scratch)

Lasso uses an ℓ_1 penalty and is implemented via coordinate descent with soft-thresholding:

$$\theta_j \leftarrow \frac{S(\rho_j, \lambda n)}{z_j},$$

where $S(\cdot, \cdot)$ is the soft-thresholding operator.

Both a default α and a grid-search over α and tolerance are used to study convergence and sparsity.

6.1.4 Benchmark Models: SVR, XGBoost, LightGBM

To capture nonlinearities and interactions without explicit feature design:

- **SVR**: Support Vector Regression with RBF, linear, and polynomial kernels; hyperparameters tuned via `GridSearchCV`.
- **XGBoost**: Gradient boosting trees with tuned depth, learning rate, and regularisation parameters.
- **LightGBM**: Efficient gradient boosting with tuned `num_leaves`, `max_depth`, etc.

6.2 Evaluation Methodology

For each model and configuration:

- Train on the training set (possibly with cross-validation for hyperparameter tuning),
- Evaluate on the held-out test set,
- Report:

$$\text{MSE}, \quad \text{RMSE} = \sqrt{\text{MSE}}, \quad R^2.$$

6.3 Regression Results & Discussion

Summary Tables

Table 1: Regression Performance After Hyperparameter Tuning

Dataset	**Model**	**CV RMSE**	**Test MSE**	**Test RMSE**	**Test R^2**
70:30 Linear	Ridge Regression	10.3890	109.0886	10.4445	0.5968
80:20 Linear	Ridge Regression	10.5191	95.9651	9.7962	0.6276
70:30 Polynomial	Ridge Regression	6.4835	47.1203	6.8644	0.8259
80:20 Polynomial	Ridge Regression	6.6524	41.7222	6.4593	0.8381
70:30 Linear	Lasso Regression	10.4122	108.7401	10.4279	0.5981
80:20 Linear	Lasso Regression	10.5190	95.9485	9.7953	0.6276
70:30 Polynomial	Lasso Regression	6.5174	47.0018	6.8558	0.8263
80:20 Polynomial	Lasso Regression	6.6189	41.7097	6.4583	0.8381
70:30 Linear	SVR	7.3283	57.9170	7.6103	0.7859
80:20 Linear	SVR	7.2071	51.1047	7.1488	0.8017
70:30 Polynomial	SVR	5.7338	31.0164	5.5692	0.8854
80:20 Polynomial	SVR	5.6130	28.5399	5.3423	0.8892
70:30 Linear	XGBoost	4.9295	21.9325	4.6832	0.9189
80:20 Linear	XGBoost	4.6661	19.3635	4.4004	0.9249
70:30 Polynomial	XGBoost	4.9447	20.7670	4.5571	0.9232
80:20 Polynomial	XGBoost	4.7358	22.7184	4.7664	0.9118
70:30 Linear	LightGBM	4.7844	18.7172	4.3263	0.9308
80:20 Linear	LightGBM	4.4472	21.0364	4.5865	0.9184
70:30 Polynomial	LightGBM	4.7843	19.5419	4.4206	0.9278
80:20 Polynomial	LightGBM	4.6032	21.8454	4.6739	0.9152

Discussion (qualitative):

- Linear models provide an interpretable baseline but underfit complex nonlinear relationships.
- Ridge and Lasso regularisation can reduce overfitting when polynomial features are included.
- Tree-based ensemble models (XGBoost, LightGBM) typically achieve lower test RMSE and higher R^2, indicating they capture interactions and nonlinear effects more effectively.

7. Classification Task – RCC Suitability

7.1 Target Definition

A binary target `RCC_Suitable` is defined using the rule:

$$\texttt{RCC\_Suitable} = \begin{cases} 1, & \text{if } \texttt{Strength} \geq 20 \text{ MPa and Age} \geq 28 \text{ days,} \\ 0, & \text{otherwise.} \end{cases}$$

This approximates a simple threshold based on IS 456:2000 for characteristic strength and age.

7.2 Additional EDA for Classification

A dedicated EDA section analyses class balance, age/strength distributions by class, and the effect of key ratios:

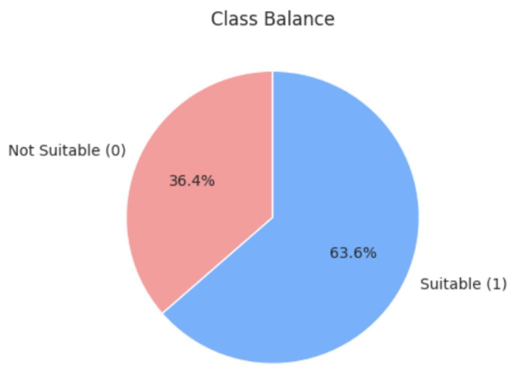


Figure 5: Class balance for `RCC_Suitable`.

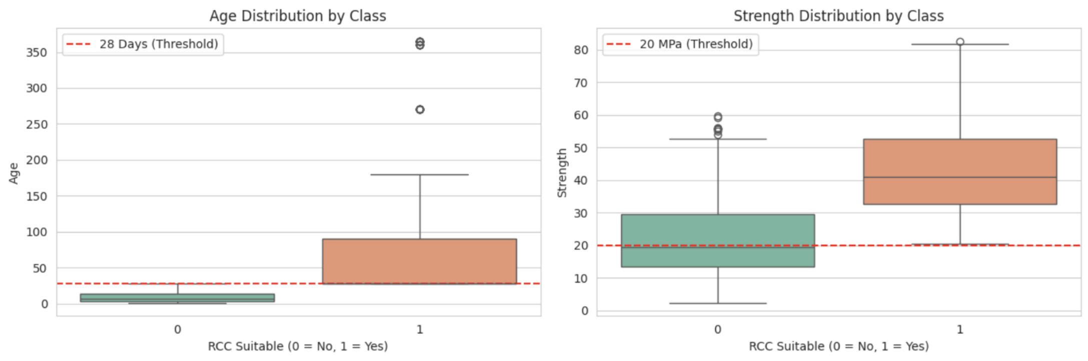


Figure 6: Age and Strength distributions split by RCC suitability.

7.3 Classification Models Implemented

- **Logistic Regression (from scratch)**:
  - Binary cross-entropy loss with optional L1 or L2 regularisation,
  - Gradient descent updates for weights and bias,
  - Manual tuning of regularisation strength C via small validation split.
- **Logistic Regression (scikit-learn)**:
  - `liblinear` solver,
  - Grid search over C and penalty type (L1/L2).
- **Decision Tree, Random Forest, Gradient Boosting, SVM**:
  - Baseline models trained on SMOTE-resampled data,
  - Hyperparameter tuning via `GridSearchCV` with F1-score as objective.

7.4 Handling Class Imbalance

The dataset exhibits moderate imbalance between suitable and non-suitable mixes. To mitigate bias, SMOTE is applied to the training set after scaling, synthesising minority-class examples. This helps classifiers learn a more balanced decision boundary.

7.5 Evaluation Metrics

For each model, we compute on the held-out test set:

- Accuracy,
- Precision,
- Recall,
- F1-score,
- ROC–AUC.

F1-score and recall are particularly important because:

- High recall ensures that genuinely "Fit" mixes are rarely misclassified as "Not Fit",
- F1-score balances recall and precision.

7.6 Classification Results & Discussion

Summary Table

Table 2: Classification Performance (Actual Values)

Model	Accuracy	Precision	Recall	F1-score	ROC–AUC
LogReg (Sklearn Tuned)	0.9951	0.9924	1.0000	0.9962	0.9991
SVM	0.9903	0.9850	1.0000	0.9924	0.9990
SVM (Tuned)	0.9854	0.9776	1.0000	0.9887	0.9977
Decision Tree	0.9854	0.9848	0.9924	0.9886	0.9828
Gradient Boosting (Tuned)	0.9806	0.9704	1.0000	0.9850	0.9997
Gradient Boosting	0.9806	0.9704	1.0000	0.9850	0.9995
Random Forest	0.9806	0.9704	1.0000	0.9850	0.9996
LogReg (Scratch L1)	0.9757	0.9632	1.0000	0.9813	0.9997
RF (Tuned)	0.9757	0.9632	1.0000	0.9813	0.9997
LogReg (Sklearn)	0.9709	0.9562	1.0000	0.9776	0.9998
DT (Tuned)	0.9660	0.9559	0.9924	0.9738	0.9562
LogReg (Scratch L2)	0.9612	0.9424	1.0000	0.9704	0.9991

Key Visualizations

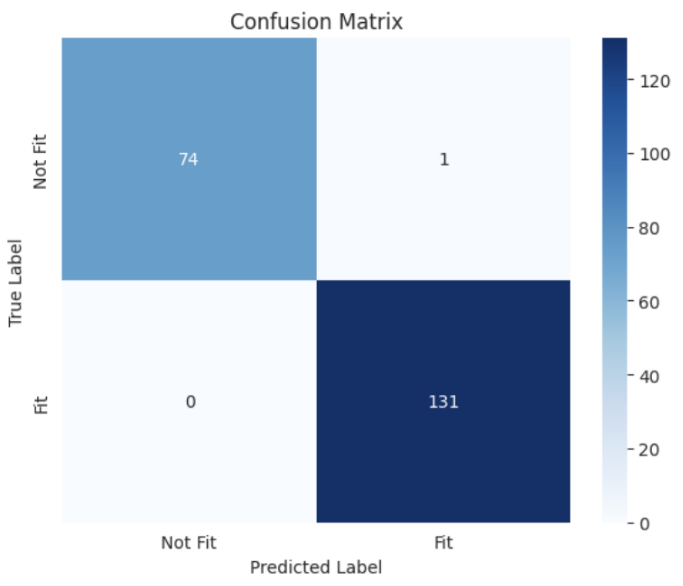


Figure 7: Confusion matrix for the best classifier.

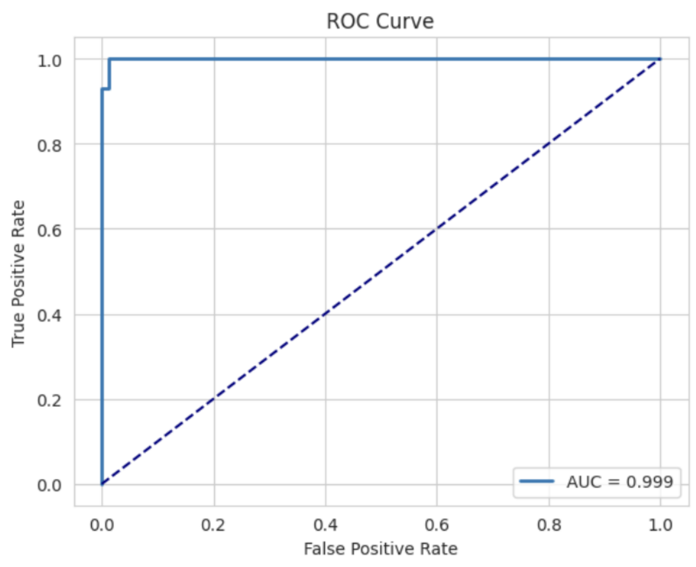


Figure 8: ROC curve of the best classifier with AUC.

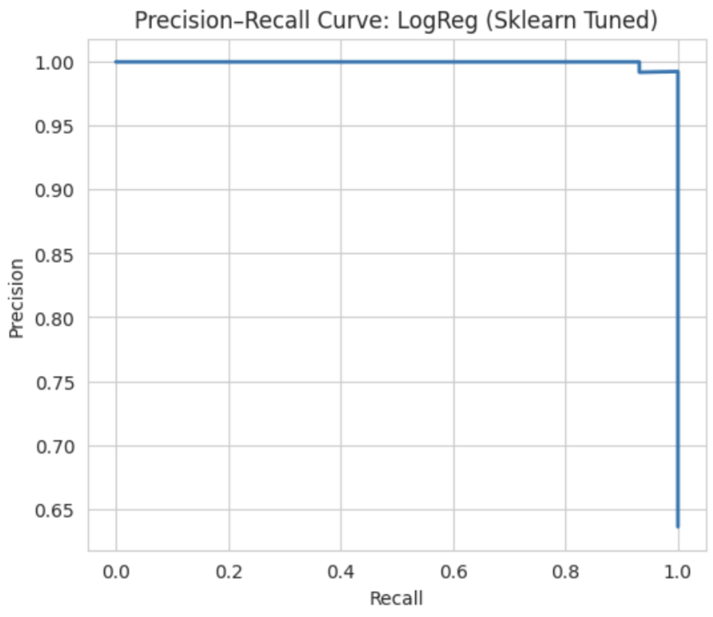


Figure 9: Precision–Recall curve for the best classifier.

Discussion (qualitative):

- After SMOTE and tuning, multiple models achieve very high F1 and AUC, reflecting the relatively clean separation induced by the threshold-based label.
- The tuned scikit-learn logistic regression performs competitively while remaining interpretable (coefficients map to odds ratios).
- Tree-based methods can handle nonlinearities and interactions without explicit feature engineering, but logistic regression remains attractive for deployment due to speed and simplicity.

Key Practical Points

- The feature engineering inside the function must exactly match the training pipeline to avoid mismatch or `KeyError`.
- The model and scaler used in deployment should be the ones fitted on the training data (and ideally saved using `joblib`).
- Confidence (probability) values can be used to set different decision thresholds depending on safety requirements.

8. Conclusions & Limitations

8.1 Conclusions

- The project successfully builds a unified machine learning pipeline that performs both **regression (strength prediction)** and **classification (RCC suitability)** using the same dataset.
- Extensive feature engineering (binder ratios, water–cement ratios, age transformations, interaction terms, and polynomial features) significantly improved model performance in both tasks.
- For **regression**, nonlinear models such as **SVR, XGBoost, and LightGBM** achieved the best results. After tuning, these models consistently produced low RMSE values (around 4–5 MPa) and high R^2 scores (0.89–0.93), outperforming linear, ridge, and lasso regression. This demonstrates that concrete strength exhibits nonlinear relationships that are better captured by tree-based and kernel-based models.
- For **classification**, multiple models performed extremely well, but the **Tuned Logistic Regression (scikit-learn)** emerged as the best classifier with an F1-score of **0.9962**, AUC close to **1.0**, and near-perfect precision and recall. This shows that the engineered feature set separates RCC-suitable and non-suitable mixes very effectively.
- Custom implementations—Normal Equation Linear Regression, Ridge, Lasso via coordinate descent, and Logistic Regression—performed as expected and provided valuable insights into how the underlying algorithms operate. Their results closely aligned with their library counterparts.
- The pipeline includes a **deployment-ready RCC assessment function** that replicates the entire preprocessing workflow and produces accurate suitability predictions for any new concrete mix.

8.2 Limitations

- The classification label (`RCC_Suitable`) was constructed using a threshold based on IS 456:2000 within the same dataset, making it a rule-derived label. Real field labels may introduce more noise and uncertainty.
- Although the dataset contains over 1000 samples, it still represents a limited region of the possible concrete design space. Field data from construction sites or laboratory experiments would further validate the generalisation capabilities of the models.
- SMOTE was used to balance the classification training data. While effective for learning, SMOTE-generated samples may not always correspond to physically realistic concrete mixes.
- Polynomial feature expansion in regression increases dimensionality and may slightly risk overfitting, although regularisation and cross-validation mitigated this.

9. Author Contributions

This project was completed as a collaborative effort between the two team members. The division of work was carried out to ensure clarity, efficiency, and a balanced distribution of responsibilities. The contributions are summarised below:

Rohit Kumar (2025AIB2565)

- Implemented the entire **Regression pipeline**, including:
  - Normal Equation–based Linear Regression (scratch implementation),
  - Ridge Regression (closed-form implementation),
  - Lasso Regression using Coordinate Descent (scratch implementation),
  - Polynomial feature engineering for regression,
  - Hyperparameter tuning for Ridge, Lasso, SVR, XGBoost, and LightGBM,
  - Evaluation using RMSE, MSE, R^2, and visualisations (residuals, parity plots).
- Prepared regression tables included in the final report.

Akhil Vashistha (2025AIB2558)

- Implemented the entire **Classification pipeline**, including:
  - Custom Logistic Regression (from scratch) with L1/L2 regularisation,
  - Scikit-learn logistic regression with tuning,
  - Decision Tree, Random Forest, Gradient Boosting, and SVM classifiers,
  - SMOTE-based imbalance handling and train–test preprocessing,
  - ROC, PR curves, confusion matrices, and model comparison dashboards,
  - Deployment function for RCC suitability prediction.
- Prepared classification tables included in the final report.

Joint Contributions

- Designed the unified pipeline structure for regression + classification.
- Conducted EDA, feature engineering ideas, and dataset cleaning.
- Co-authored the report and validated experimental results.

References

[1] UCI Machine Learning Repository (2007). *Concrete Compressive Strength Data Set*. Available at: `https://archive.ics.uci.edu/dataset/165/concrete+compressive+strength`

[2] Bureau of Indian Standards. (2000). *IS 456: 2000 – Plain and Reinforced Concrete – Code of Practice*. BIS, New Delhi, India.

[3] Wright, S. J. (2015). *Coordinate Descent Algorithms*. Mathematical Programming, 151(1), 3–34. (Foundational reference explaining modern coordinate descent used for Lasso.)