ML Assignment-1

Shobhit Raj (2022482)

1 Section A

1.1 Q1. a)

As the complexity of the model is increased, it may lead to overfitting, i.e. the model learns not only the underlying patterns of the training data, but also the noise present in it. So, it will perform very well on the training data, hence reducing bias, but as it becomes sensitive to noise & irrelavant details in the training data, it may generalize poorly on unseen test data, leading to poor performance on test data, hence reuslting in high variance. Therefore, increasing complexity of the model will lead to low bias & high variance.
The following Bias Variance Tradeoff graph [fig. 1] represents the above situation :-

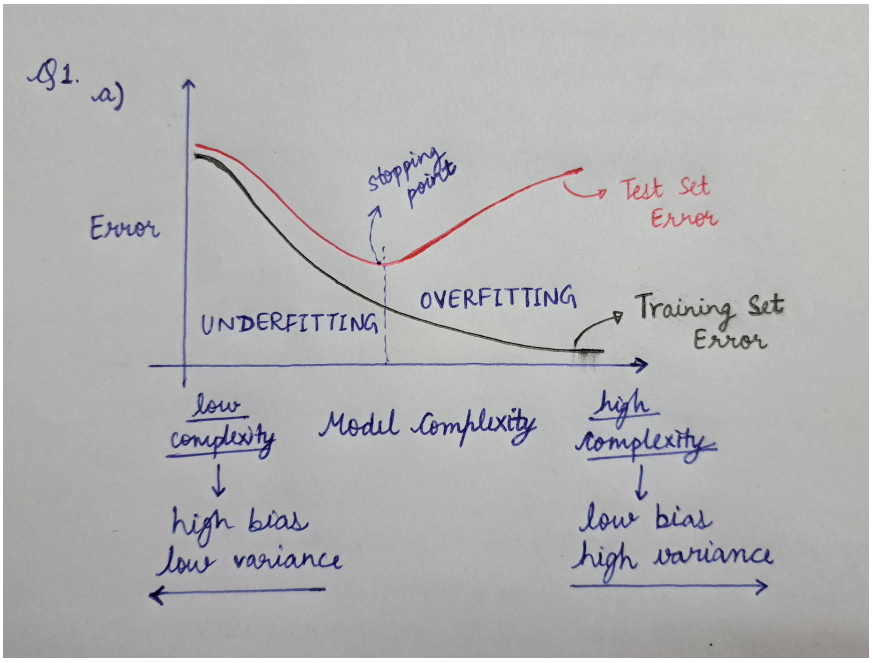


Figure 1: Bias Variance Tradeoff Graph

1.2 Q1. b)

The confusion matrix & various performance metrics [fig. 2] for this classification problem is :-

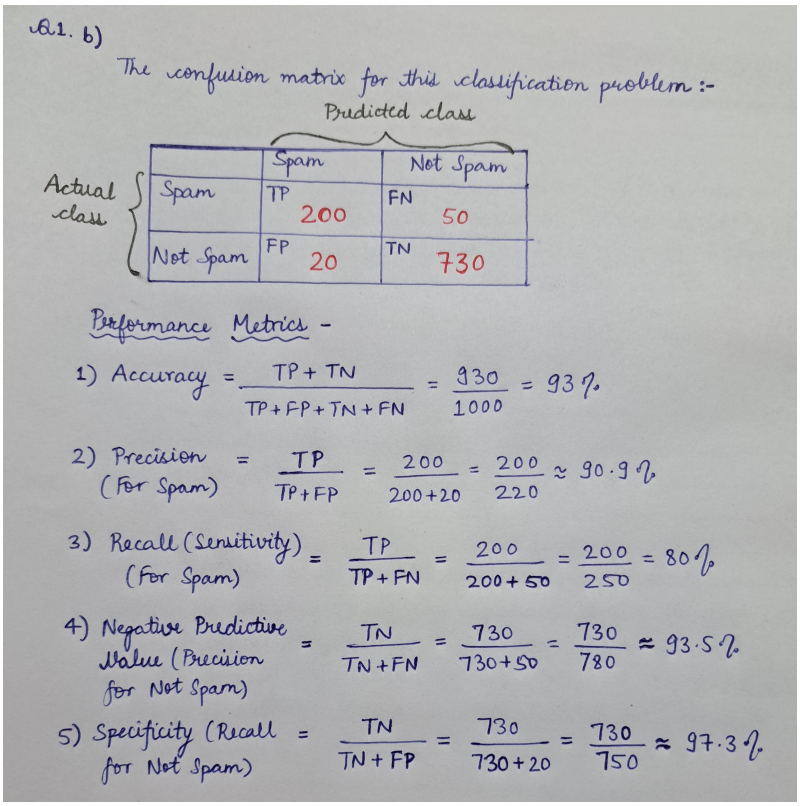


Figure 2: Confusion Matrix for Spam Detection

Here, 2 types of errors are there :-
1) False Positives (FP): Legitimate emails that are misclassified as spam and diverted to the spam folder. This is undesirable because it can result in important emails being lost or missed by the user. If minimizing false positives (legitimate emails misclassified as spam) is more important, Precision should be the focus.
2) False Negatives (FN): Spam emails that slip through the filter and reach the user's inbox. This is also undesirable but might be less problematic compared to losing important legitimate emails. If minimizing false negatives (spam emails slipping through) is more important, Recall should be the focus.

The first error, False Positive is more undesirable, so "minimizing false positives" (legitimate emails misclassified as spam) is more important here. Hence, Precision is likely to be the most appropriate metric because users are more likely to be concerned about losing legitimate emails to the spam folder. "Precision for spam emails" measures how many emails classified as spam are truly spam. A higher precision means that fewer legitimate emails are mistakenly flagged as spam (i.e., lower False Positives). For this model, the "Precision for spam category" as stated above is 90.9%. "Precision for legitimate emails" measures how many emails classified as legitimate are truly legitimiate. For this model, the "Precision for not spam/legitimate category" as stated above is 93.5%.
Hence, to find the overall performance, average precision for both categories is (90.9 + 93.5)/2 = 92.2%.

If both types of error are equally undesirable (will depend on business requirements & user needs), then we can take the average of Precision & Recall or use another metric called F1-score which provides a good balance between precision and recall. It is given as :-
F1 Score = 2 × (Precision×Recall)/(Precision+Recall)
F1-Score for Spam (Positive Class) = 2×(90.9×80)/(90.9+80) = 85.1%.
F1-Score for Legitimate Emails (Negative Class) = 2×(93.5×97.3)/(93.5+97.3) = 95.4%.
To find the overall performance, we calculate the average of the F1-scores for both classes: (85.1 + 95.4)/2 = 90.3%.

1.3 Q1. c)

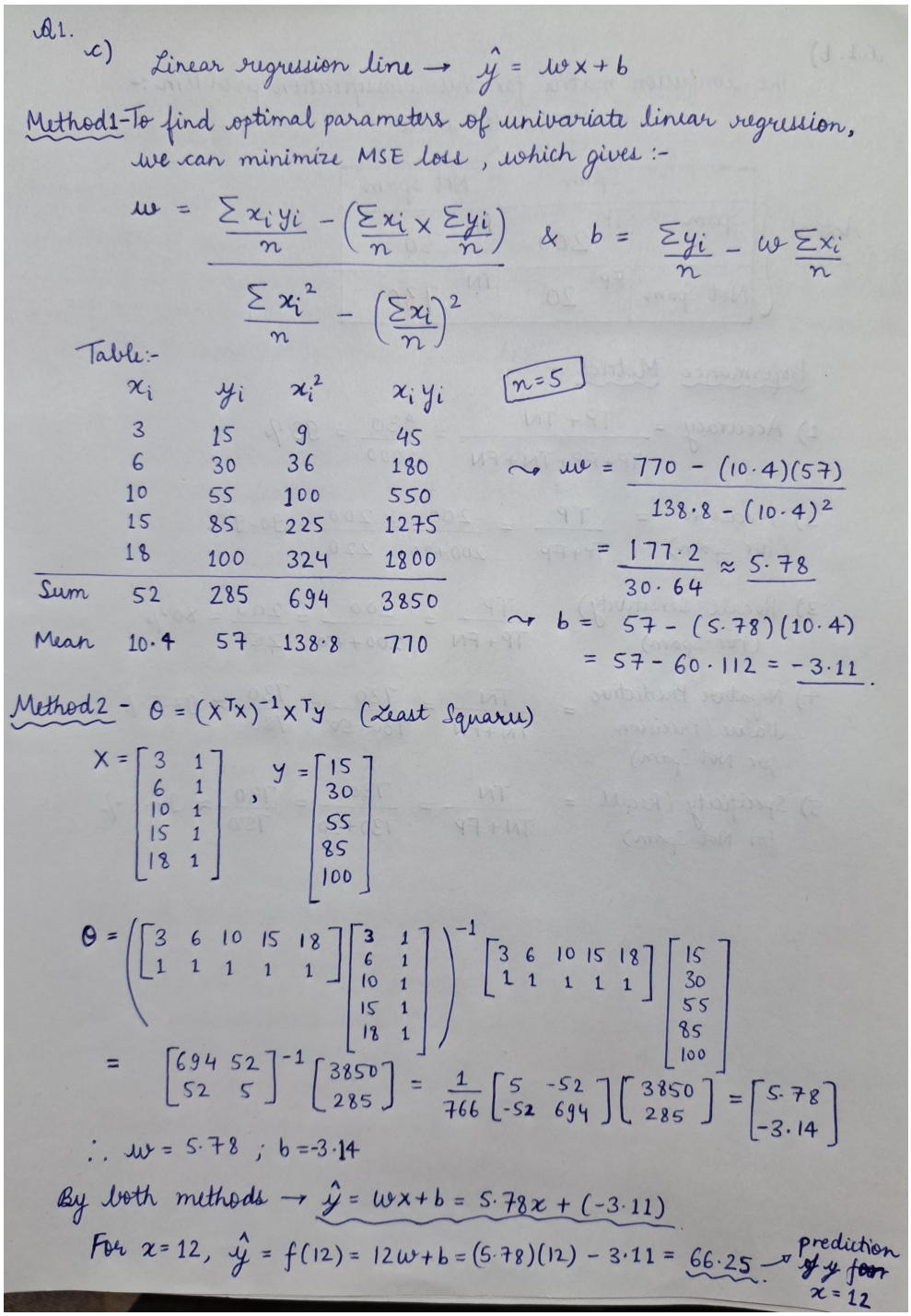


Figure 3: Q1 (c) Calculations

The equation of the regression line is obtained as 5.78x - 3.11.
The prediction for x=12 will be 66.25.

1.4 Q1. d)

We have to provide such an example of datasets, models & loss function such that Model f1 has a lower empirical risk (training error) but overfits the data, leading to poor generalization. Model f2 has a slightly higher empirical risk but generalizes better because it avoids overfitting.

Here, empirical risk refers to the average loss on the training set, whereas generalization refers to how well the model performs on unseen data (test set). A model with lower empirical risk on the training set does not always generalize better, especially if it overfits the training data (the reason & graph stated in part (a) of this question too).

The toy example is as follows [fig. 4 & 5] :-

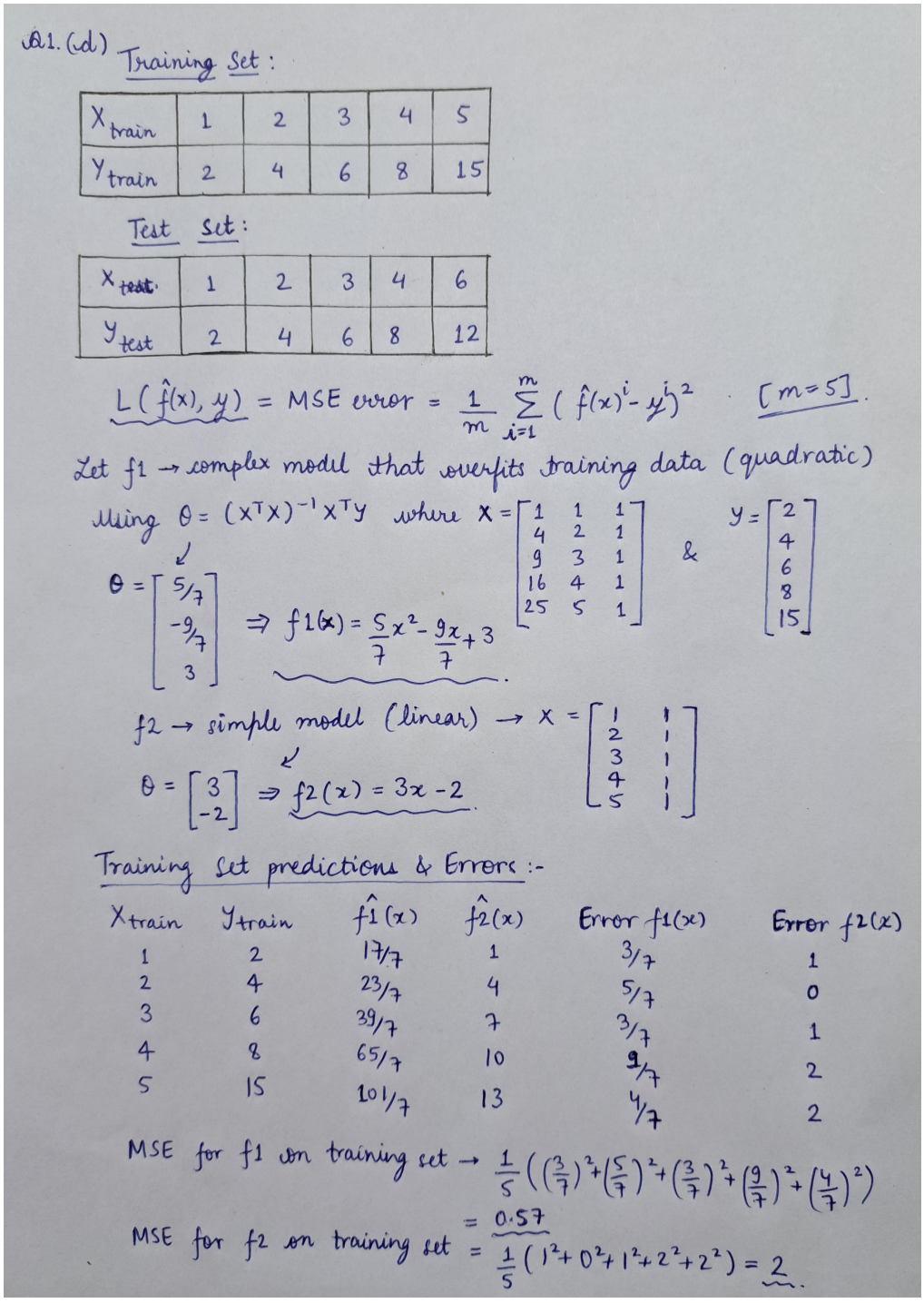


Figure 4: Q1 (d) Training Set Empirical Risk

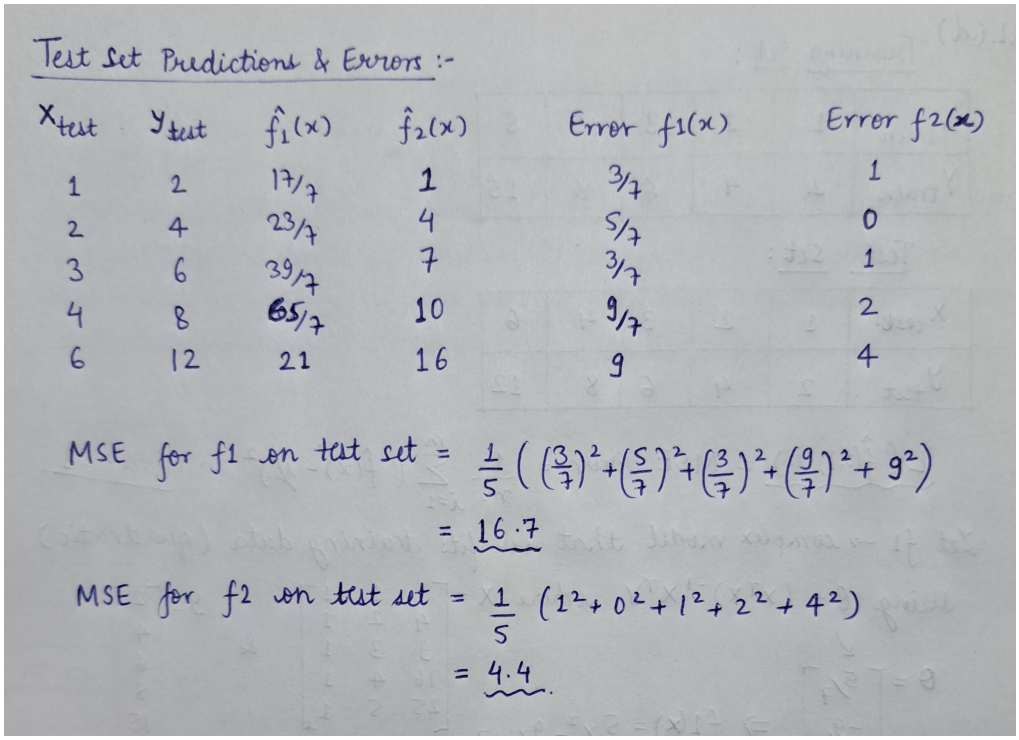


Figure 5: Q1 (d) Test Set Empirical Risk

Training Set: Model f1 has a lower empirical risk (MSE = 0.57) than f2 (MSE = 2) because f1 overfits the data, including the outlier at x = 5.

Test Set: Model f1 performs poorly on unseen data, with a much higher test error (MSE = 16.7) compared to f2 (MSE = 4.4), showing that f1 overfits and does not generalize well.

Thus, while f1 performs better on the training set, its poor generalization ability leads to worse performance on the test set compared to f2. This happens because the model f1 learns not only the underlying patterns of the training data, but also the noise & irrelavant details present in it. So, it will perform very well on the training data, but as it becomes sensitive to the noise, it performs poorly on the test data. This highlights the importance of balancing model complexity to avoid overfitting.

2 Section B

We have been given a 'Heart Disease' dataset, which contains 4238 entries and 16 columns. The columns are: 'male', 'age', 'education', 'currentSmoker', 'cigsPerDay', 'BPMeds', 'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose', and the label 'HeartDisease'.
This is a classification problem.

Handling Missing Values
The dataset contained missing values in several columns, particularly in the features 'education', 'cigsPerDay', 'BPMeds', 'totChol', 'BMI', 'heartRate', and 'glucose'. Out of these, the 'BPMeds' feature has a categorical value, either 0 or 1, while the others have floating numerical values.
For the categorical value, the most frequent value (mode) was used to fill in the missing values. This strategy ensures that the mode, being the most common value, has the least likelihood of introducing bias in the model's performance.
For the numerical values, missing values were imputed using the mean of each feature. Mean imputation is appropriate for continuous variables as it retains the general distribution of the data while handling missing entries without introducing extreme values or outliers.

Splitting the Dataset
Once the missing values were handled, the dataset was split into training, testing, and validation sets using a 70:15:15 ratio.
Training Set (70%): Used to train the model.
Validation Set (15%): Used during training to monitor the model's performance and prevent overfitting.
Test Set (15%): Set aside to evaluate the model's final performance.
Stratified Sampling: The data was split in a stratified manner, ensuring that both the training and testing datasets retained the original proportion of the 'HeartDisease'target variable. This helps prevent any bias in the distribution of the target class during the splits.
After splitting, the data was converted to NumPy arrays for compatibility with the gradient descent implementation.

2.1 Q2. a) Implementing Logistic Regression using Batch Gradient Descent

The code implements logistic regression using batch gradient descent, which means the entire training dataset is used to compute the gradient and update the model parameters (w and b) at each iteration. The code uses vectorized operations for efficient computation.
Sigmoid Function - It transforms the linear combination of input features and weights into a probability between 0 and 1. The probability is then used to make binary classifications (0 or 1).
Cost Function (Cross Entropy Loss) - The cost function used is the binary cross-entropy loss.
Gradient Calculation - The gradients with respect to w (weights) and b (bias) are computed to determine how much to adjust the parameters during each iteration. This is essential for minimizing the cost function.
Prediction - The model predicts the class label by applying the sigmoid function and thresholding the output (if ≥ 0.5, classify as 1; otherwise, classify as 0).
Batch Gradient Descent Algorithm - It updates the parameters w and b over several iterations (epochs). The learning rate alpha controls the step size of each parameter update. In each iteration, the cost & accuracy is computed for both the training and validation sets and the parameters w and b are updated based on the computed gradients.

I implemented Logistic Regression using Batch Gradient Descent and explored the impact of different learning rates on the convergence behavior of the model. For each learning rate, I tracked both the training and validation loss, as well as the training and validation accuracy across 10,000 epochs. The goal was to visualize how the model's cost and performance evolved with each learning rate and to determine which rate led to the most efficient convergence. The plots [fig. 6] are given below :-

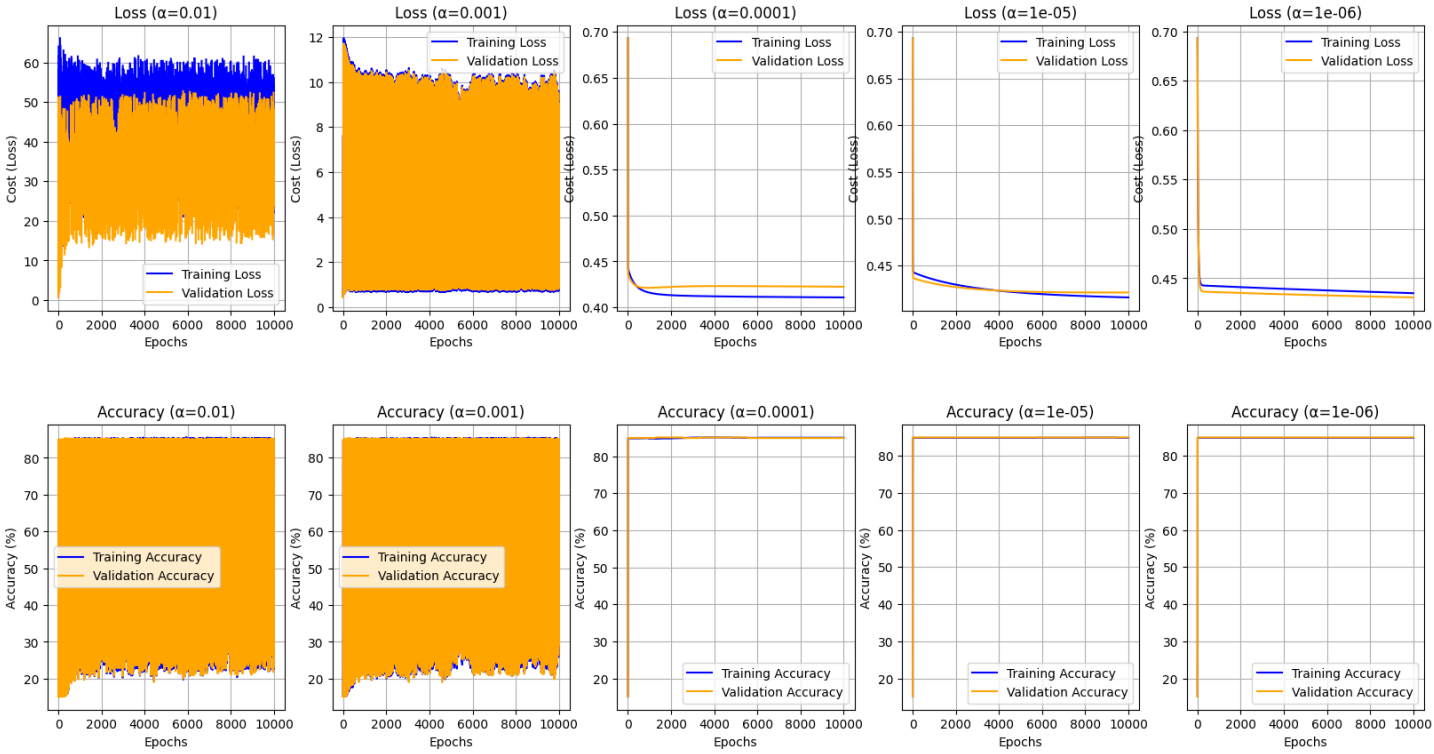


Figure 6: Loss & Accuracy v/s Epochs for different learning rates

The convergence of the model depends heavily on the learning rate (α), as evidenced by the plots.

For $\alpha = 0.01$, the model experiences instability during training. Both the training and validation loss fluctuate significantly, indicating that the learning rate is too high. The model fails to converge to a stable minimum, and the accuracy metrics remain low and erratic.

For $\alpha = 0.001$, the model shows more stability. The training and validation loss gradually decrease and plateau, indicating convergence. However, there are still some minor fluctuations, particularly in the validation loss, suggesting that the model may be slightly overfitting or underfitting.

For $\alpha = 0.0001$, the model converges smoothly and steadily. The loss values drop consistently without any significant fluctuations, and both the training and validation accuracy approach 80%, indicating successful convergence. This learning rate provides a good balance between speed and stability.

For $\alpha = 0.00001$, and even smaller $\alpha = 0.000001$, the model also converges, but very slowly. The loss and accuracy graphs show that the learning rate is too small, causing the model to take a very long time to improve. Despite converging, this would not be practical for real-world applications due to the time inefficiency.

In conclusion, the model converges well at moderate learning rates (0.001 and 0.0001), but it either fails to converge at very high learning rates (0.01) or converges too slowly at very low learning rates (0.00001 and 0.000001).

2.2 Q2. b) Using Min-Max Feature Scaling

Performance of model without applying any feature scaling is evaluated in part (a).
Min-Max Scaling - Scaling features using this method brings the features into a range between 0 and 1. This method helps models converge faster and can often improve performance, especially when features have different scales. The plots of the performance [fig. 7] are given below :-

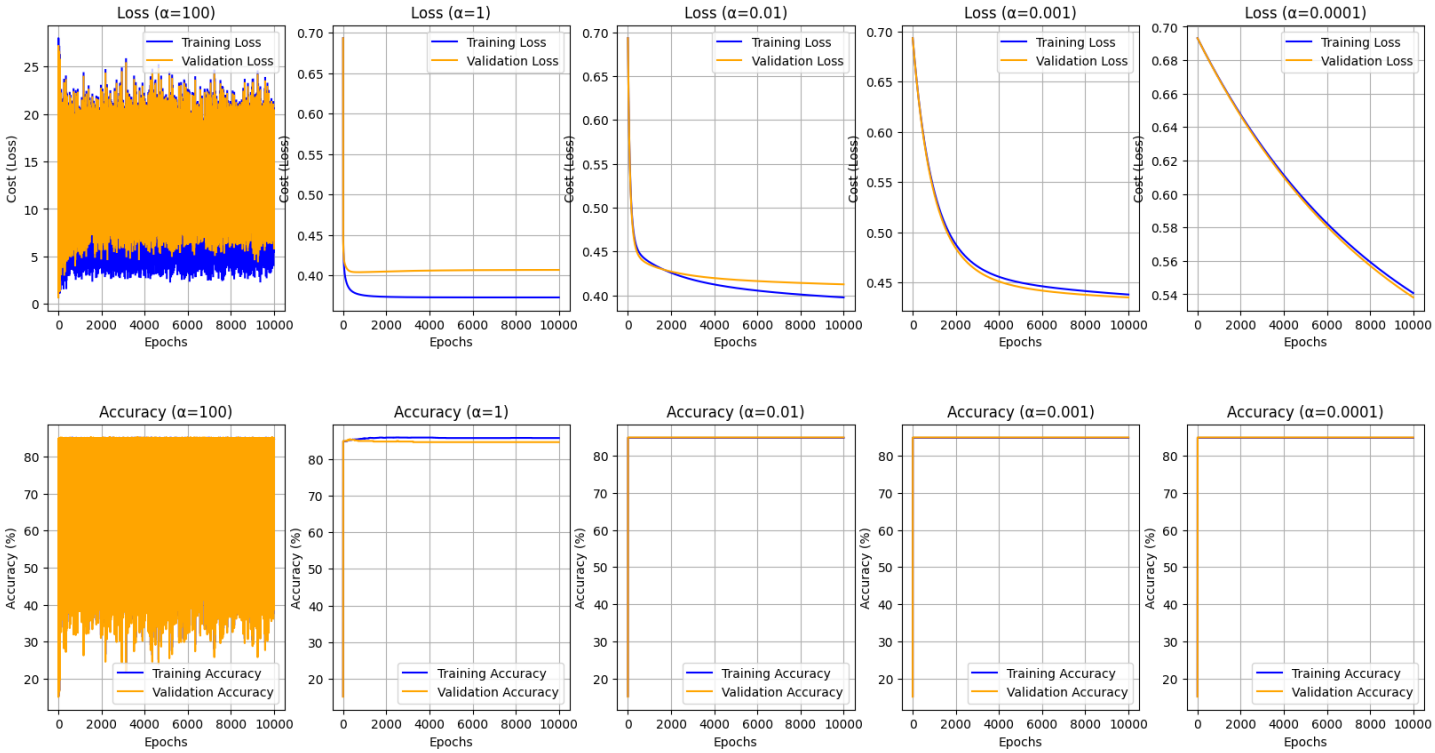


Figure 7: Loss & Accuracy v/s Epochs after applying Min-Max Scaling

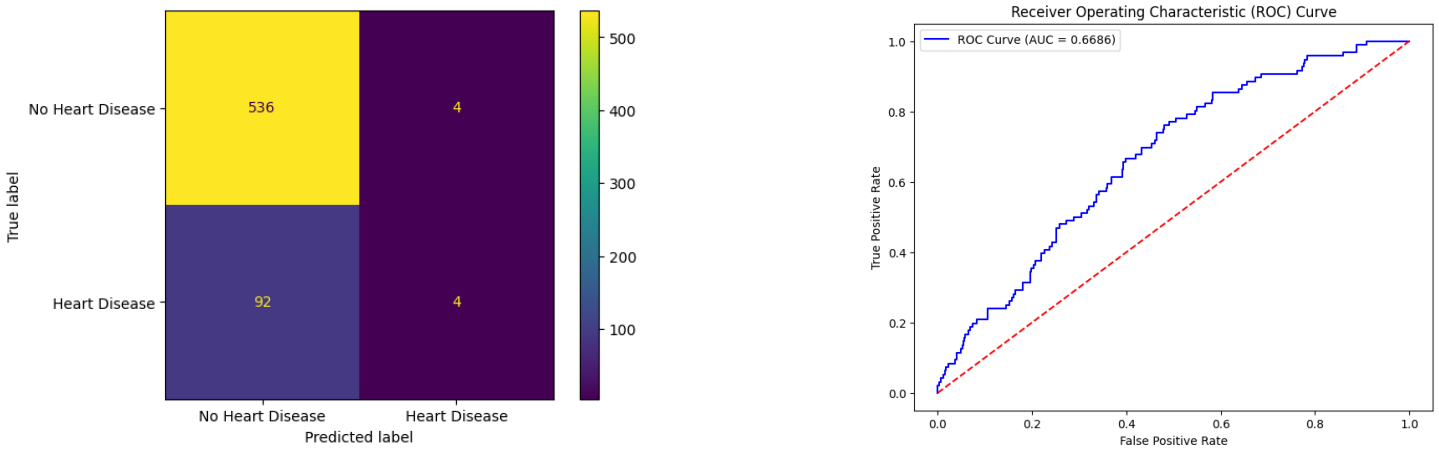
Without scaling, the model behaved erratically, particularly at $\alpha = 0.01$ and larger learning rates. The loss fluctuates wildly, making it hard for the model to converge. For lower learning rates like $\alpha = 0.0001$, the model does eventually converge, but much more slowly compared to the scaled version.

The impact of feature scaling on the convergence of the logistic regression model can be observed clearly. The loss converges smoothly for learning rates of $\alpha = 1, 0.01, 0.001, 0.0001$, with the model showing consistent reductions in loss as the epochs progress. The loss function flattens after about 2000 epochs, showing efficient & quicker convergence. Similarly, the validation loss aligns well with the training loss, indicating that the model generalizes properly across the validation set. There is no significant overfitting visible.

Without scaling, gradient descent struggles, as features with larger magnitudes dominate the updates. For example, a feature representing age may range from 20 to 70, while a feature representing cholesterol might range from 100 to 300. This causes oscillations in the loss function, leading to poor & slower convergence behavior. With Min-Max scaling applied, features are normalized to a common range, which helps ensure that each feature contributes proportionally to the parameter updates, which stabilizes gradient descent, allowing smoother convergence for both training and validation loss.

2.3 Q2. c) Performance on Validation Set

I used the trained model to make predictions on the validation set. The metrics of performance on validation set is given below [fig. 8]:-



(a) Confusion Matrix

(b) ROC Curve

Figure 8: Performance Metrics on Validation Set

Confusion Matrix:

$$\begin{bmatrix} 536 & 4 \\ 92 & 4 \end{bmatrix}$$

True Negatives (TN): 536
False Positives (FP): 4
False Negatives (FN): 92
True Positives (TP): 4

Accuracy: 84.906%

Precision: 0.5000

Recall: 0.0417

F1 Score: 0.0769

ROC-AUC Score: 0.6686

Model's Performance :-
The model's performance is highly skewed towards predicting the negative class, which results in high true negatives but poor true positives. The very low recall suggests that the model is not suitable for identifying heart disease effectively. While the precision indicates that some correct positive predictions are made, however, this can be misleading because the model only predicted 8 positive instances in total (4 true positives and 4 false positives). The low recall and F1 score highlight a significant issue with sensitivity. The ROC-AUC score further reinforces that the model needs improvement to balance its predictions for both classes. This is because the class imbalance makes it challenging for the model to learn to predict the minority class (Heart Disease present) effectively.

2.4 Q2. d) Implementing Stochastic and Mini-Batch Gradient Descent

I implemented Stochastic Gradient Descent in which model parameters are updated based on a single training example at a time and explored the impact of different learning rates on the convergence behavior of the model. For each learning rate, I tracked the training, validation & test loss, as well as the training, validation & test accuracy across 10,000 epochs. The plots [fig. 9] are given below :-

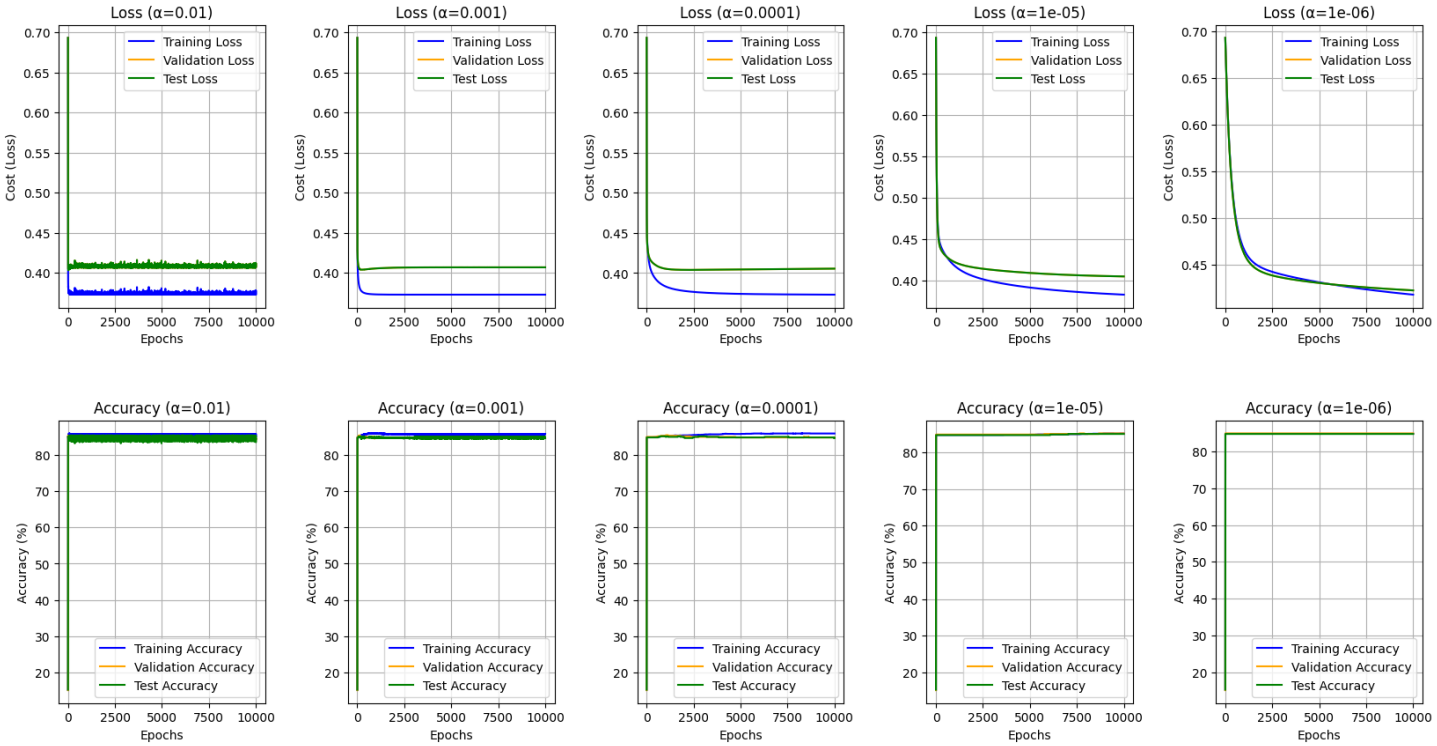


Figure 9: Loss & Accuracy v/s Epochs for Stochastic Gradient Descent

I also implemented Mini-Batch Gradient Descent which splits the training dataset into small batches and updates the parameters based on these mini-batches and explored the impact of different learning rates & different batch sizes on the convergence behavior of the model. The plots [fig. 10] are given on the next page.

Based on the plots :-

Batch gradient descent is the most stable but typically the slowest in terms of parameter updates since it computes the gradients based on the entire dataset. This means it can take longer to converge initially, but once it does, it remains stable i.e shows minimal fluctuations.

SGD has the potential to converge faster in the initial stages but the primary trade-off with SGD is its lack of stability, especially with higher learning rates. This instability arises because each update is based on each data point, leading to fluctuations in the loss and accuracy. Lowering the learning rate can help, but it may slow down convergence.

Mini-batch gradient descent offer a good trade-off in terms of stability. While they introduce some noise due to partial updates, the size of the batch helps average out extreme fluctuations, leading to smoother convergence than SGD. Smaller batch sizes still experience some oscillations in loss and accuracy, but as the batch size increases, the model becomes more stable, resembling the behavior of full-batch gradient descent.

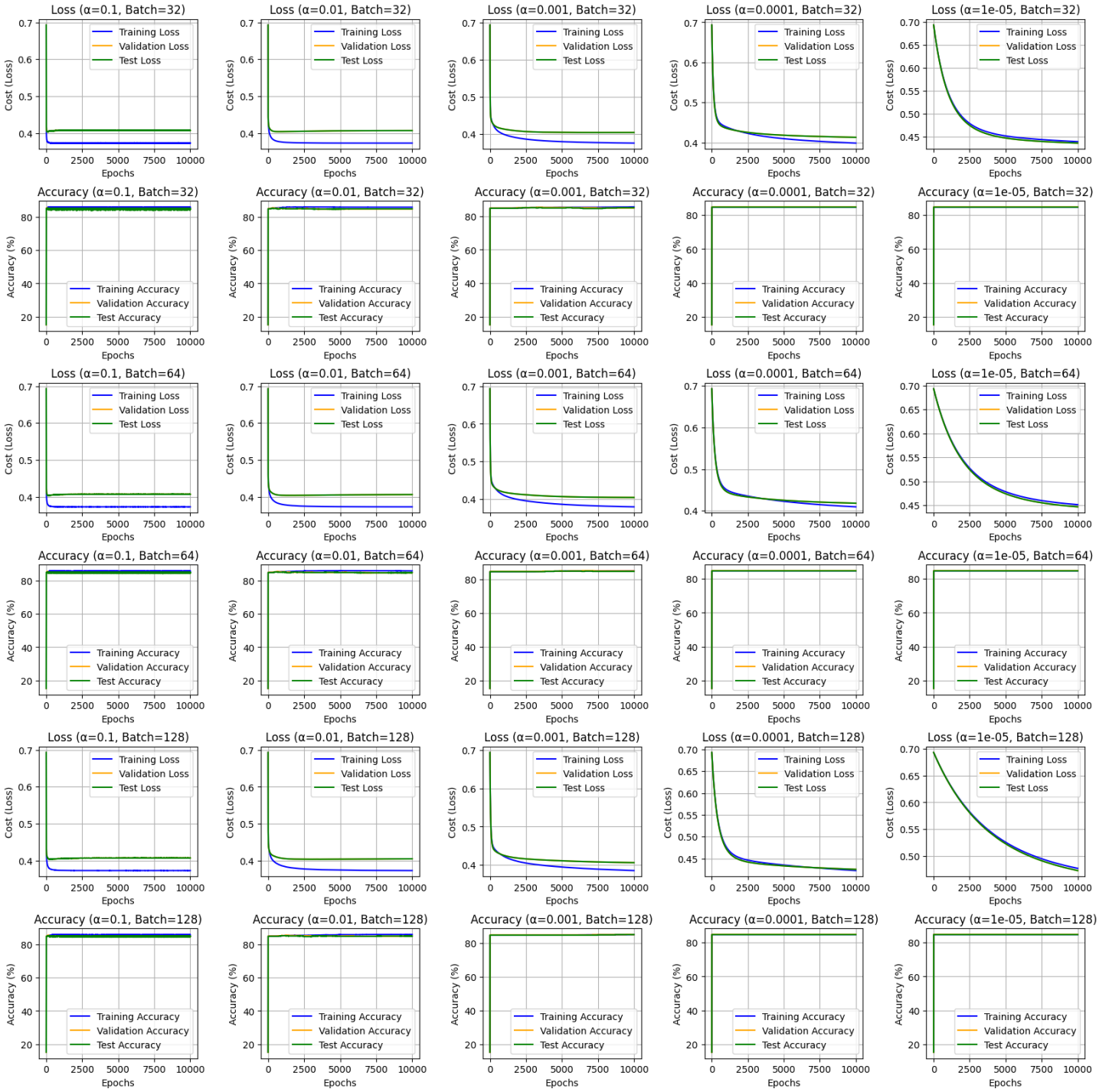


Figure 10: Loss & Accuracy v/s Epochs for Mini-Batch Gradient Descent

2.5 Q2. e) Implementing K-Fold Cross Validation

The K-fold Cross Validation using $k = 5$ folds gave the following performance metrics (Average ± Variance):

- **Accuracy:** 0.8515 ± 0.0109
- **Precision:** 0.7001 ± 0.2015
- **Recall:** 0.0498 ± 0.0128
- **F1 Score:** 0.0923 ± 0.0218

The accuracy across the five folds is quite stable, with a mean of 85.15% and a standard deviation of 1.09%. This indicates that the model consistently classifies most instances correctly, and its overall performance is not highly dependent on which subset of the data is used for training or testing.

The precision varies significantly between folds, with a mean of 0.7001 and a standard deviation of 0.2015. This indicates that the model's ability to correctly identify positive predictions fluctuates, likely due to the relatively low number of positive cases (heart disease) in the dataset. In some folds, the model exhibits a perfect precision (1.0), while in others it drops considerably (e.g., fold 3 at 0.44), highlighting the model's inconsistency in minimizing false positives.

The recall & F1 score, are very low across all folds which further indicates the instability of the model when it comes to handling the minority class.

While the model demonstrates consistent accuracy, it struggles significantly with precision and recall, leading to high variance in these metrics. This suggests the model is reliable for predicting the majority class (no heart disease) but is unstable and unreliable for identifying the minority class (heart disease).

2.6 Q2. f) Implementing Early Stopping & Regularization Techniques

To prevent overfitting and improve generalization, I implemented L1 (Lasso) and L2 (Ridge) regularization techniques within the gradient descent framework. L1 regularization adds a penalty proportional to the absolute value of the weights, while L2 regularization penalizes the square of the weights, encouraging smaller but non-zero values across the weights. These regularization terms modify the gradient updates, adding the respective regularization term, which helps control the complexity of the model, thereby reducing overfitting.

Early Stopping Implementation

For early stopping, I introduced two criteria :-

1) Validation loss increases: Training halts if the validation loss starts increasing, indicating the model is beginning to overfit.

2) Consecutive validation loss difference threshold: Training stops when the difference between two consecutive validation losses becomes smaller than a threshold, set to 1e-6, ensuring minimal improvement.

I experimented with different learning rates and regularization (L1 and L2), visualizing the training and validation losses along with accuracy across epochs, without any stopping criteria & then with using the 2 stopping criteria mentioned above. The plots [fig. 11-13] are given below :-

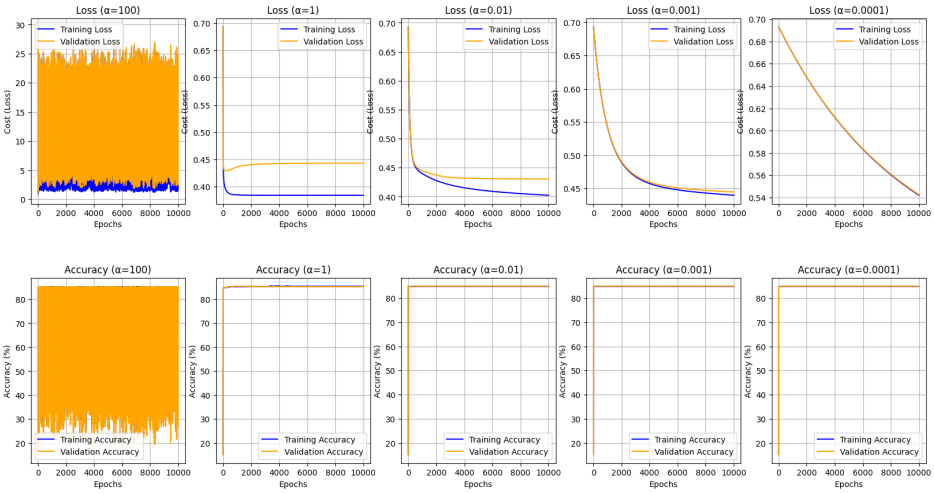


Figure 11: Loss & Accuracy v/s Epochs for L1 regularization with No Early Stopping

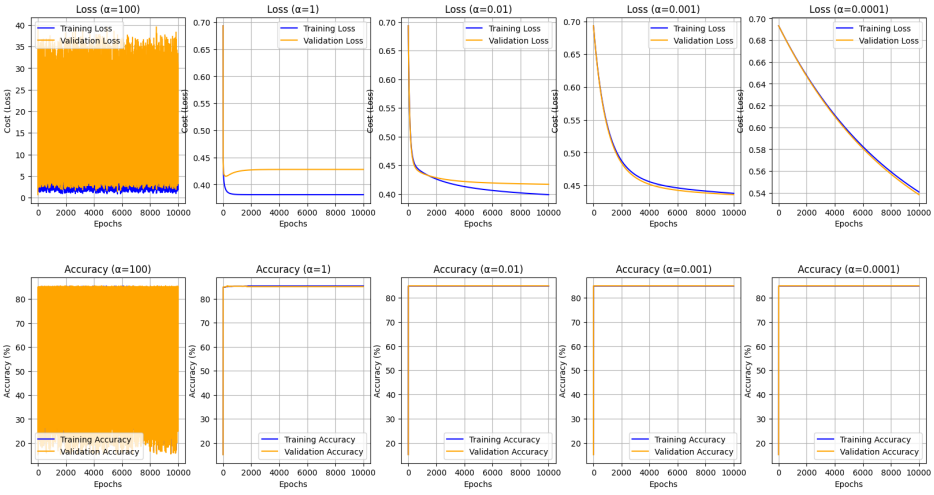
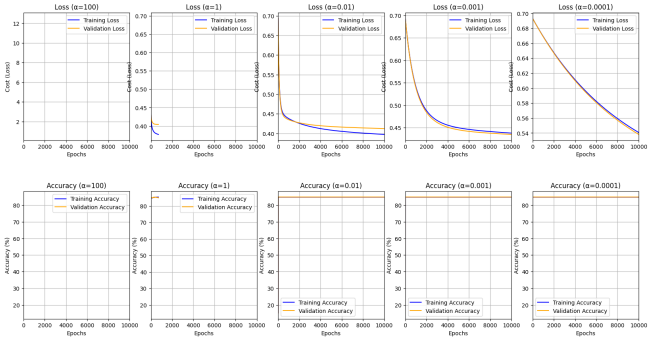
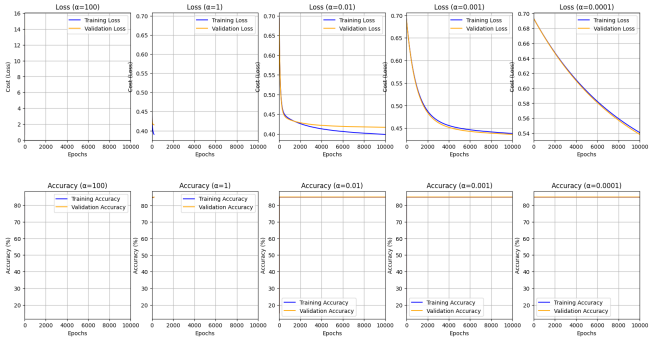


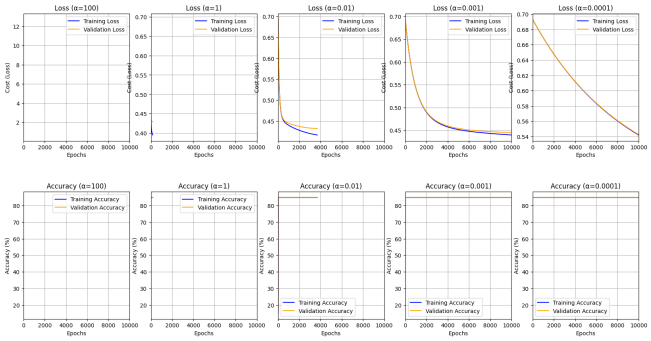
Figure 12: Loss & Accuracy v/s Epochs for L2 regularization with No Early Stopping



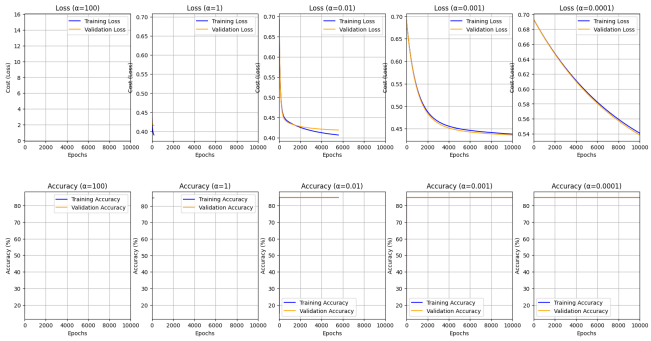
(a) Loss & Accuracy v/s Epochs for L1 regularization with Stopping Criteria 1



(b) Loss & Accuracy v/s Epochs for L2 regularization with Stopping Criteria 1



(c) Loss & Accuracy v/s Epochs for L1 regularization with Stopping Criteria 2



(d) Loss & Accuracy v/s Epochs for L2 regularization with Stopping Criteria 2

Figure 13: Comparison of Loss & Accuracy v/s Epochs for L1 and L2 regularization with different stopping criteria

In the attached plots, early stopping is shown to effectively stop training when the validation loss starts increasing, helping avoid overfitting.

Early stopping serves as a simple but effective method to control overfitting. By halting training once the validation loss begins to increase or fails to improve significantly, we prevent the model from memorizing the training data and help it generalize better. It also indicates that the model has reached convergence.

In summary, early stopping prevents overfitting by halting training when the model starts to diverge in validation performance. This ensures a better balance between bias and variance, leading to a more robust and generalizable model. Regularization (L1/L2) combined with early stopping further enhances this effect by controlling the magnitude of the weights, leading to simpler and more interpretable models.

3 Section C

We have been given an "Electricity BILL.csv" dataset, which contains 1250 entries and 16 columns. The columns are: 'Building_Type', 'Construction_Year', 'Number_of_Floors', 'Energy_Consumption_Per_SqM', 'Water_Usage_Per_Building', 'Waste_Recycled_Percentage', 'Occupancy_Rate', 'Indoor_Air_Quality', 'Smart_Devices_Count', 'Green_Certified', 'Maintenance_Resolution_Time', 'Building_Status', 'Maintenance_Priority', 'Energy_Per_SqM', 'Number_of_Residents', and 'Electricity_Bill'.

There are no null values in the dataset.

The dataset is split into training & testing sets using 80:20 ratio.

3.1 Q3. a) EDA

The plots [fig. 14-16] are as given below :-

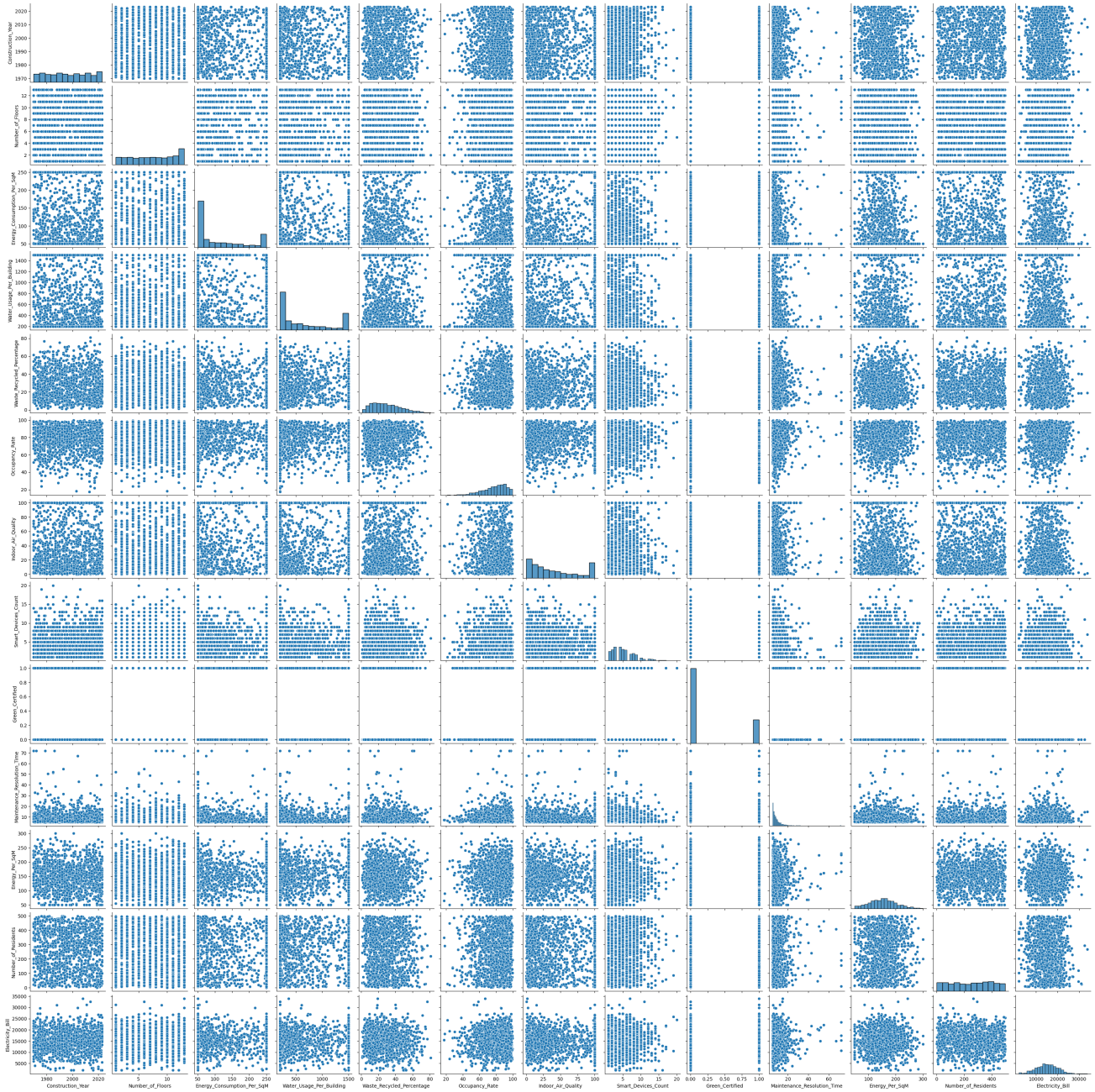
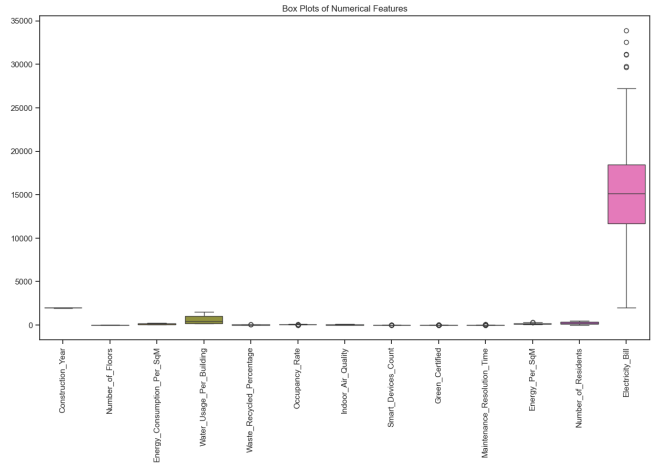
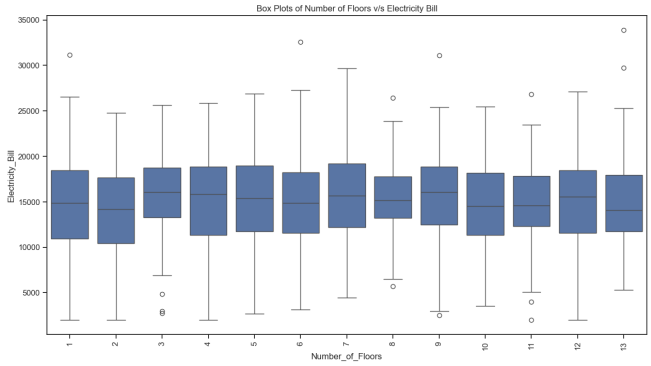


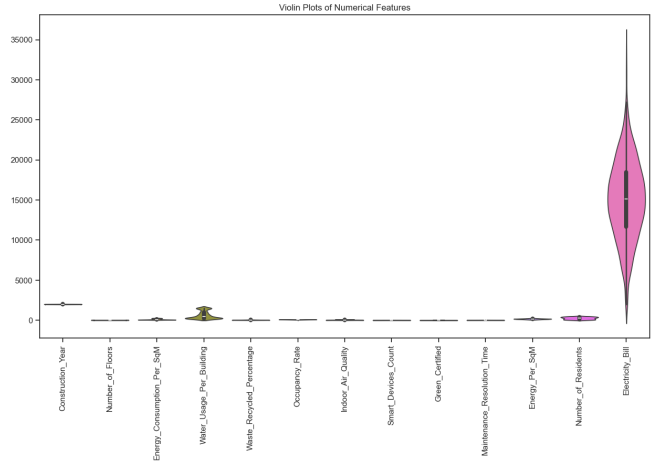
Figure 14: Pair Plot



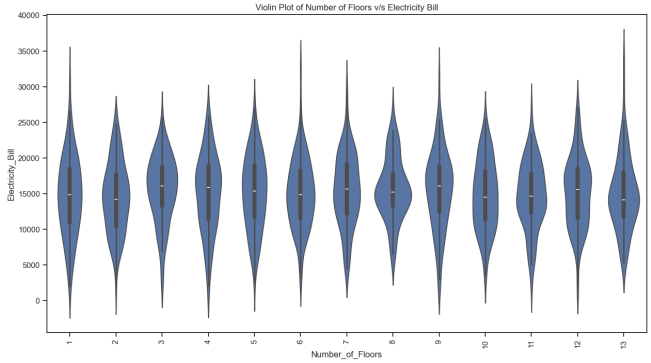
(a) Box Plots of Numerical Features



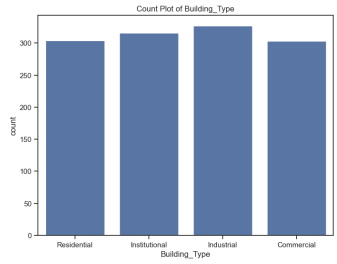
(b) Box Plots of Number of Floors



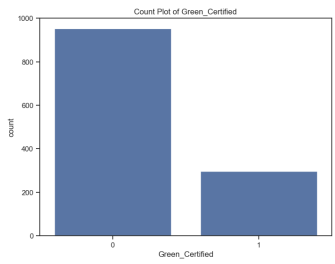
(c) Violin Plots of Numerical Features



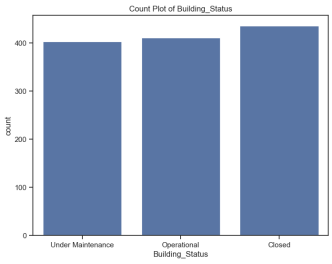
(d) Violin Plots of Number of Floors



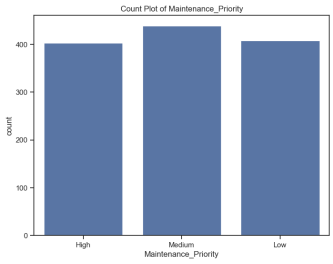
(e) Count Plot of Building_Type



(f) Count Plot of Green_Certified



(g) Count Plot of Building_Status



(h) Count Plot of Maintenance_Priority

Figure 15: Box Plots, Violin Plots, and Count Plots

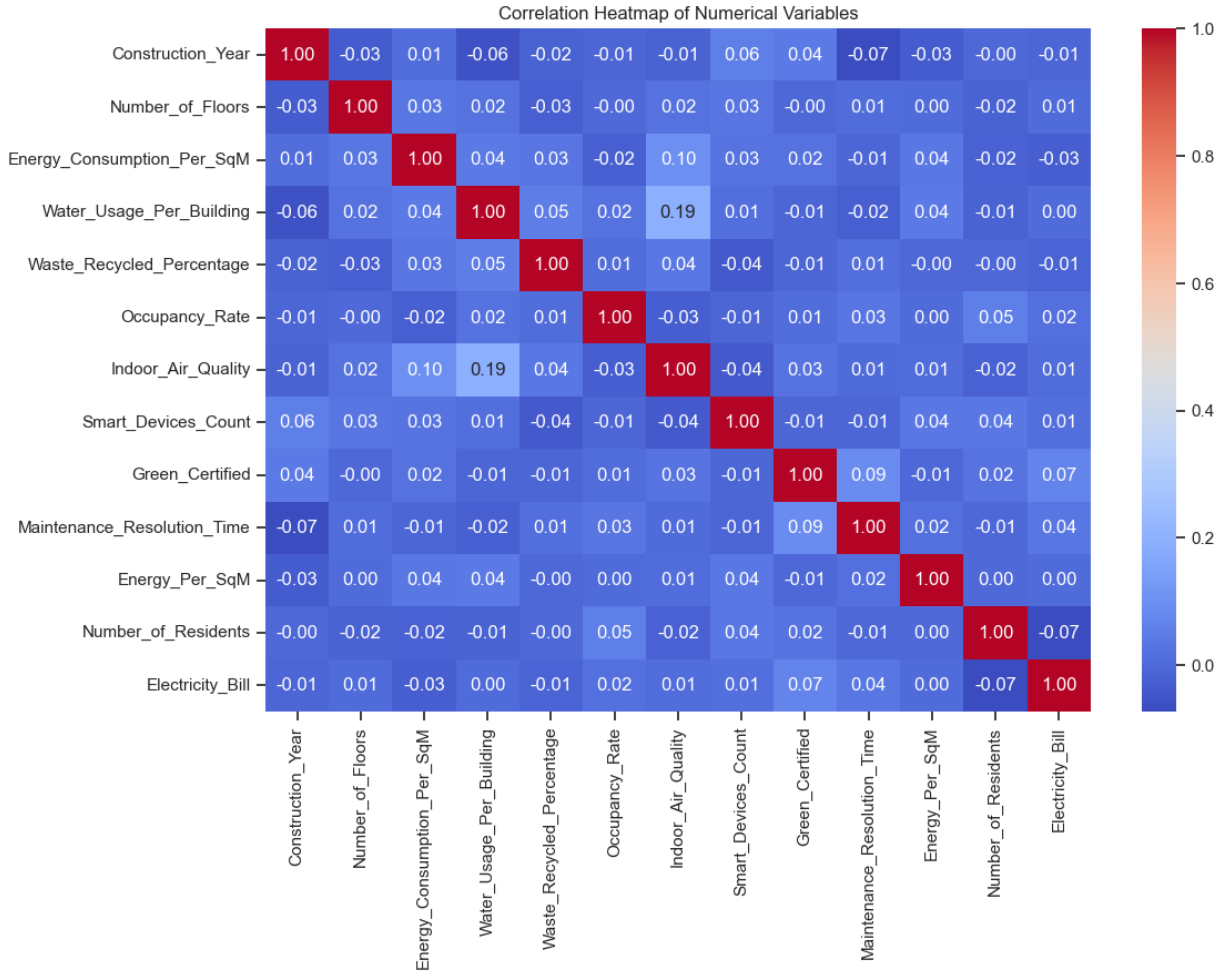


Figure 16: Correlation Heatmap

Some insights that can be drawn from the plots :-

1) In the pair plot, it can be observed that the Number_of_Floors does not appear to significantly influence Energy_Consumption_Per_SqM or Electricity_Bill, which suggests that smaller buildings can be just as energy-intensive as larger ones.

2) Number_of_Residents have no correlation with the Energy_Per_SqM, indicating that the total consumption does not significantly change with occupancy and is not dependent on it.

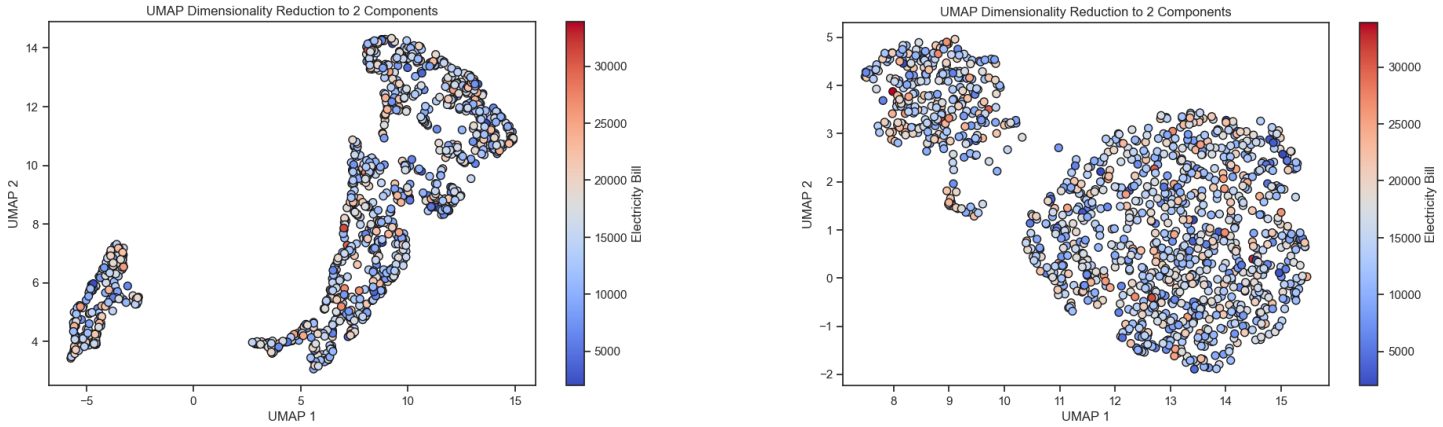
3) The correlation heatmap reveals that Construction_Year has little to no correlation with energy consumption or the electricity bill. This suggests that newer buildings are not necessarily more energy-efficient or cost-effective in terms of electricity usage.

4) From the Box Plot, it is observed that the Maintenance_Resolution_Time has some potential outliers, indicating that certain buildings experience much longer maintenance times compared to others.

5) There is negative correlation between Green_Certified and Waste_Recycled_Percentage, even though green-certified buildings might be expected to recycle more, but it is contrary.

3.2 Q3. b) Dimensionality Reduction using UMAP

The Scatter Plot [fig. 17] of dataset after performing UMAP :-



(a) Scatter Plot with No Scaling

(b) Scatter Plot with Standard Scaling

Figure 17: Scatter Plots after performing UMAP

The data points form distinct clusters or groups in the scatter plot which indicates that there are groups of data points with similar characteristics or patterns.
When applied Standard Scaling, the clusters are well-defined and dense suggesting that more similar types of points after scaling, hence increasing cluster density.
With no scaling, the clusters are separated from each other & good separability means that different groups of data points are well-distinguished from each other. With scaling, it is not that separable indicating that the features used do not fully capture the differences between the groups.

3.3 Q3. c) Linear Regression

Model Evaluation Metrics :-

Train Metrics:

- Train MSE: 24,475,013.168
- Train RMSE: 4,947.223
- Train MAE: 4,006.328
- Train R^2 Score: 0.014
- Train Adjusted R^2 Score: -0.001

Test Metrics:

- Test MSE: 24,278,016.156
- Test RMSE: 4,927.273
- Test MAE: 3,842.409
- Test R^2 Score: 0.000
- Test Adjusted R^2 Score: -0.064

3.4 Q3. d) RFE Feature Selection

Feature Selection and Model Evaluation :-

Selected Features: Building_Type, Green_Certified, Number_of_Residents

Train Metrics:

- Train MSE: 24,569,032.907
- Train RMSE: 4,956.716
- Train MAE: 4,006.473
- Train R2 Score: 0.010
- Train Adjusted R2 Score: 0.007

Test Metrics:

- Test MSE: 23,941,409.063
- Test RMSE: 4,892.996
- Test MAE: 3,813.948
- Test R2 Score: 0.014
- Test Adjusted R2 Score: 0.002

On comparing with (c) part, it can be concluded that the feature selection process with RFE and scaling did not lead to significant improvements in the model's performance.

3.5 Q3. e) One Hot Encoding & Ridge Regression

Model Evaluation Metrics :-

Train Metrics:

- Train MSE: 24,188,936.715
- Train RMSE: 4,918.225
- Train MAE: 3,976.694
- Train R2 Score: 0.025
- Train Adjusted R2 Score: 0.007

Test Metrics:

- Test MSE: 24,129,257.316
- Test RMSE: 4,912.154
- Test MAE: 3,797.612
- Test R2 Score: 0.006
- Test Adjusted R2 Score: -0.076

On comparing with (c) part, it can be concluded that the Ridge Regression with One-Hot Encoding provides a slight improvement over the original model with slight decrease in MSE & MAE and slight increase in R2-Score showing a slight improvement in explanatory power

3.6 Q3. f) ICA

Model Evaluation Metrics :-

Results for 4 Components:

- Train MSE: 24,701,058.762
- Train RMSE: 4,970.016
- Train MAE: 4,010.984
- Train R2 Score: 0.005
- Train Adjusted R2 Score: 0.001
- Test MSE: 24,167,219.342
- Test RMSE: 4,916.017
- Test MAE: 3,818.894
- Test R2 Score: 0.005
- Test Adjusted R2 Score: -0.012

Results for 5 Components:

- Train MSE: 24,683,781.380
- Train RMSE: 4,968.278
- Train MAE: 4,008.436
- Train R2 Score: 0.006
- Train Adjusted R2 Score: 0.001
- Test MSE: 24,261,490.341
- Test RMSE: 4,925.595
- Test MAE: 3,831.495
- Test R2 Score: 0.001
- Test Adjusted R2 Score: -0.020

Results for 6 Components:

- Train MSE: 24,682,728.731
- Train RMSE: 4,968.172
- Train MAE: 4,009.399
- Train R2 Score: 0.006
- Train Adjusted R2 Score: -0.000
- Test MSE: 24,253,810.365
- Test RMSE: 4,924.816
- Test MAE: 3,829.859
- Test R2 Score: 0.001
- Test Adjusted R2 Score: -0.024

Results for 8 Components:

- Train MSE: 24,674,426.131
- Train RMSE: 4,967.336
- Train MAE: 4,009.038
- Train R2 Score: 0.006
- Train Adjusted R2 Score: -0.002
- Test MSE: 24,222,147.565
- Test RMSE: 4,921.600
- Test MAE: 3,830.143
- Test R2 Score: 0.002
- Test Adjusted R2 Score: -0.031

There is no significant difference in performance on both train and test datasets for all the components. On comparing with (c) & (d) part, it can be concluded that Ridge Regression without reducing the number of components performs better than ICA.

3.7 Q3. g) ICA

Test Evaluation Metrics :-

Results for alpha=0.05:

- Test MSE: 24,272,612.870
- Test RMSE: 4,926.724
- Test MAE: 3,841.703
- Test R2 Score: 0.000
- Test Adjusted R2 Score: -0.064

Results for alpha=0.1:

- Test MSE: 24,267,836.217
- Test RMSE: 4,926.240
- Test MAE: 3,841.133
- Test R2 Score: 0.000
- Test Adjusted R2 Score: -0.064

Results for alpha=5:

- Test MSE: 24,277,056.627
- Test RMSE: 4,927.175
- Test MAE: 3,834.644
- Test R2 Score: 0.000
- Test Adjusted R2 Score: -0.064

Results for alpha=10:

- Test MSE: 24,307,831.389
- Test RMSE: 4,930.297
- Test MAE: 3,837.645
- Test R2 Score: -0.001
- Test Adjusted R2 Score: -0.065

Based on the evaluation metrics for different values of the mixing parameter (alpha), the results on the test dataset are quite consistent, showing minimal variation across different values of alpha. The MSE remains fairly stable across all alpha values, ranging from approximately 24,267,836 to 24,307,831. Same with RMSE & MAE. The R2 score remains near 0 across all alpha values, indicating that the model's performance, in terms of explained variance, is weak regardless of the value of alpha.

3.8 Q3. h) GBR

Model Evaluation Metrics :-

Train Metrics:

- Train MSE: 14,926,446.257
- Train RMSE: 3,863.476
- Train MAE: 3,092.748
- Train R2 Score: 0.399
- Train Adjusted R2 Score: 0.389

Test Metrics:

- Test MSE: 24,392,500.901
- Test RMSE: 4,938.876
- Test MAE: 3,815.703
- Test R2 Score: -0.005
- Test Adjusted R2 Score: -0.069

It is observed that Gradient Boosting Regressor performs significantly better on the training set with a higher R^2 score and lower MSE and RMSE compared to the ElasticNet and ICA methods. However, on the test set, its performance is somewhat worse compared to the ElasticNet and ICA, particularly in terms of R^2 score and Adjusted R^2 score, indicating some overfitting on the training data.