CSE343 Machine Learning Assignment-1

Aditya Sharma (2022038)

Table Of Contents

(Doing Section A and B, Section C in Bonus)

Section A (Theoretical)

(a)

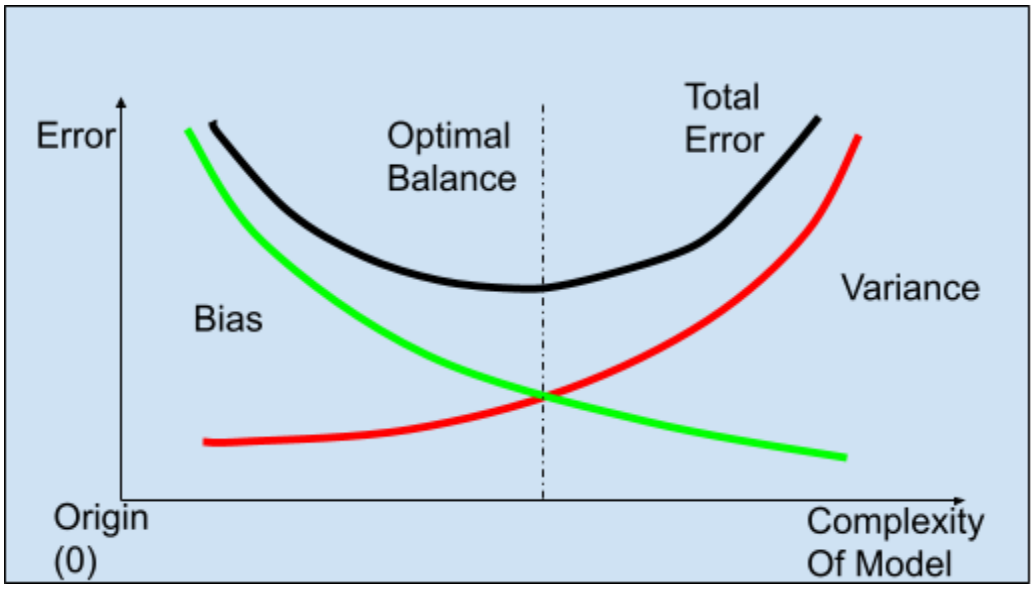
First, let us understand what is bias and variance in terms of a Machine learning Model.

- **Bias :** It is associated with the error in predictions of the training data i.e the data that the model has seen.
- **Variance:** It is associated with how the model performs on unseen data. That is how the model's performance (in this case, predictions made by regression model) varies over different datasets.

Increasing model's complexity :

As, we increase the model's complexity that could be including higher-order polynomial terms in the regression model, we would have a better "fitting model" of the training data i.e bias for the model would decrese. But, because of overly complex model we may run into issue of overfitting.

Overfitting : When the model "rote" learns from the training data instead of leveraging usefull patterns.



In the above figure we can see Bias-Variance Tradeoff curve.

Initially, when the model is simple, it has high bias because it cannot capture all the underlying patterns in the data. As model complexity increases, bias decreases since the model can better fit the training data. However, after a certain point, increasing model complexity doesn't significantly reduce bias because the model starts to overfit the noise in the data.

As the model complexity increases, variance increases because the model becomes too

sensitive to fluctuations in the training data. A highly complex model may fit the training data almost perfectly, but it will not generalize well to unseen data, leading to high variance.

b)

Confusion Matrix:

Let's break down the results:

- **True Positives (TP)**: 200 spam emails were correctly classified as spam.
- **False Negatives (FN)**: 50 spam emails were incorrectly classified as legitimate.
- **True Negatives (TN)**: 730 legitimate emails were correctly classified as legitimate.
- **False Positives (FP)**: 20 legitimate emails were incorrectly classified as spam.

	Actual Positive	**Acutally Negative**
Predicted Postive	(TP) 200	(FP) 20
Predicted Negative	(FN) 50	(TN) 730

Key Observations:

- We note that accuracy is high i.e 93% that means false positive are reduced.
- Since, the number of spam emails is data set is less that legitimate emails.
- Recall can be improved as it is only 80%. Having more training of spam emails will better improve models performance on identifying emails.

1. **Accuracy**: Measures overall correctness of model.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{200 + 730}{200 + 730 + 20 + 50}$$

$$= \frac{930}{1000} = 0.93$$

So, accuracy is 93%

2. **Precision**: Measures how many predicted spam emails are actually [spam.]

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{200}{200 + 20} = 0.909$$

So, precision is 90.9%

3. **Recall (Sensitivity)**: How many actual spam emails were correctly classified

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{200}{200 + 50} = \frac{200}{250} = 0.8$$

So, recall is 80%

4. **F1 score**: Harmonic mean of precision and recall. Balances both.

$$\text{F1 score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$= 2 \times \frac{0.909 \times 0.8}{0.909 + 0.8}$$

$$= \frac{1.4544}{1.709} = 0.851$$

So, F1 score is 85.1%

c)

	x_i	y_i	x_i^2	$x_i y_i$
	3	15	9	45
	6	30	36	180
	10	55	100	550
	15	85	225	1275
	18	100	324	1800
sum	52	285	694	3850
mean	10.4	57	138.8	770
	$\bar{x}$	$\bar{y}$	$\overline{x^2}$	$\overline{xy}$

$y = mx + b$

$$m = \frac{n\Sigma xy - \Sigma x \Sigma y}{n\Sigma x^2 - (\Sigma x)^2}$$

$$m = \frac{5 \times 3850 - 52 \times 285}{5 \times 694 - 52^2}$$

$$m = \frac{4430}{766} \approx 5.78$$

(Slope)

b (Intercept) $= \bar{y} - m\bar{x} = 57 - (5.78)(10.4) = -3.11$

Regression line : $\boxed{y = 5.78x - 3.11}$

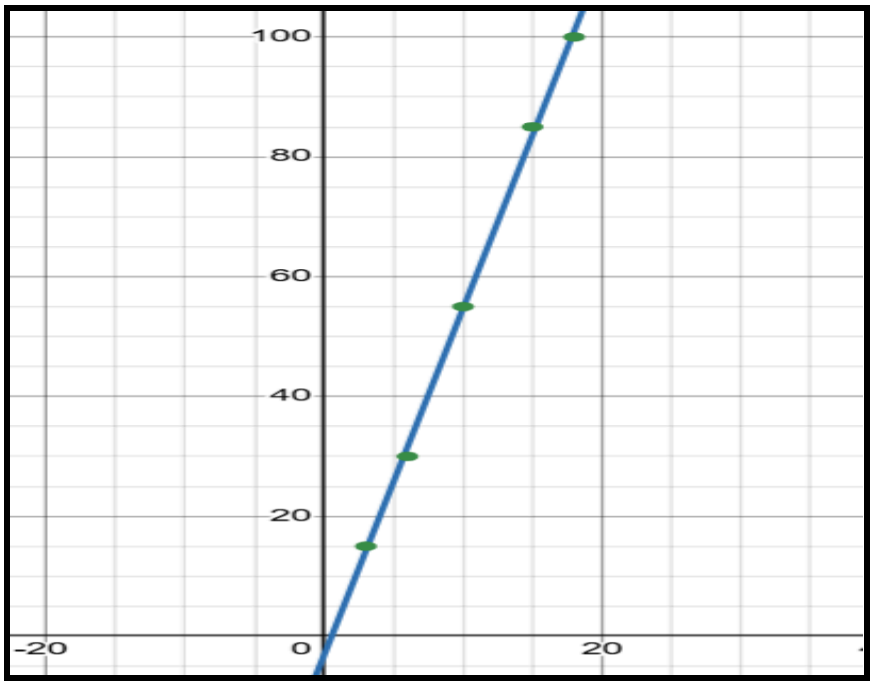
when $x = 12$,

$y = 5.78\ (12) - 3.11$

$y = 69.36 - 3.11$

$\boxed{y = 66.25}$

Regression line plotted also with given data points in green



d)

Let us consider a data set like this ➡️

Feature (X)	Label(Y)
1	2
2	3
3	5
4	7
5	11
6	13
7	14
8	16
9	18
10	20

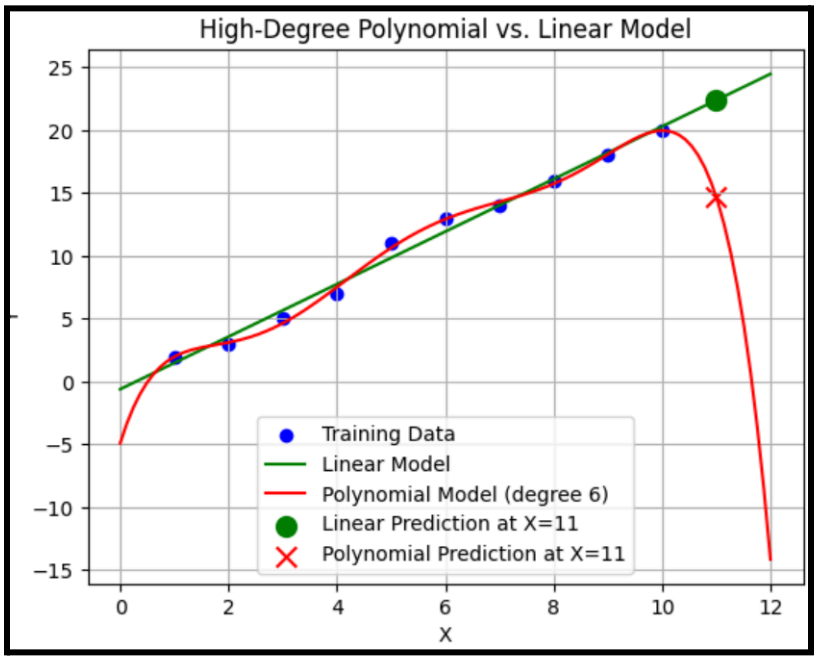
Model 1 ➡️ Fit a High-Degree Polynomial Mode

- Let's fit a polynomial model of degree 6 to this dataset. A high-degree polynomial can perfectly fit the training data but may lead to extreme predictions outside the training range.
- **Weight:** [0.0e+00, 1.38934848e+01 -1.00120542e+01, 3.55462558e+00 -5.99599359e-01, 4.78685897e-02 -1.45833333e-03]
  Bias: [-4.899999998224656]
-

Model 2 ➡️ Fit a Linear Model

- We will fit a linear regression model to the same dataset. This model will capture the overall trend without fitting the noise.
- Using OLS estimates to find m, c for regression line we get **Weight:** [2.09]**, Bias:** [-0.6]

Let use now try to predict value at X = 11 for both models.



Linear Model: The linear model provides a reasonable prediction for

- **`Linear Model Prediction at X=11: 22.40`**, staying consistent with the trend observed in the training data.

High-Degree Polynomial Model: The polynomial model, while fitting the training data perfectly, may produce an extreme prediction for

- **`Polynomial Model Prediction at X=11: 14.699999994989758`**, that is far from the expected trend.
  This is due to the nature of polynomial functions, where high degrees can lead to large fluctuations between points.

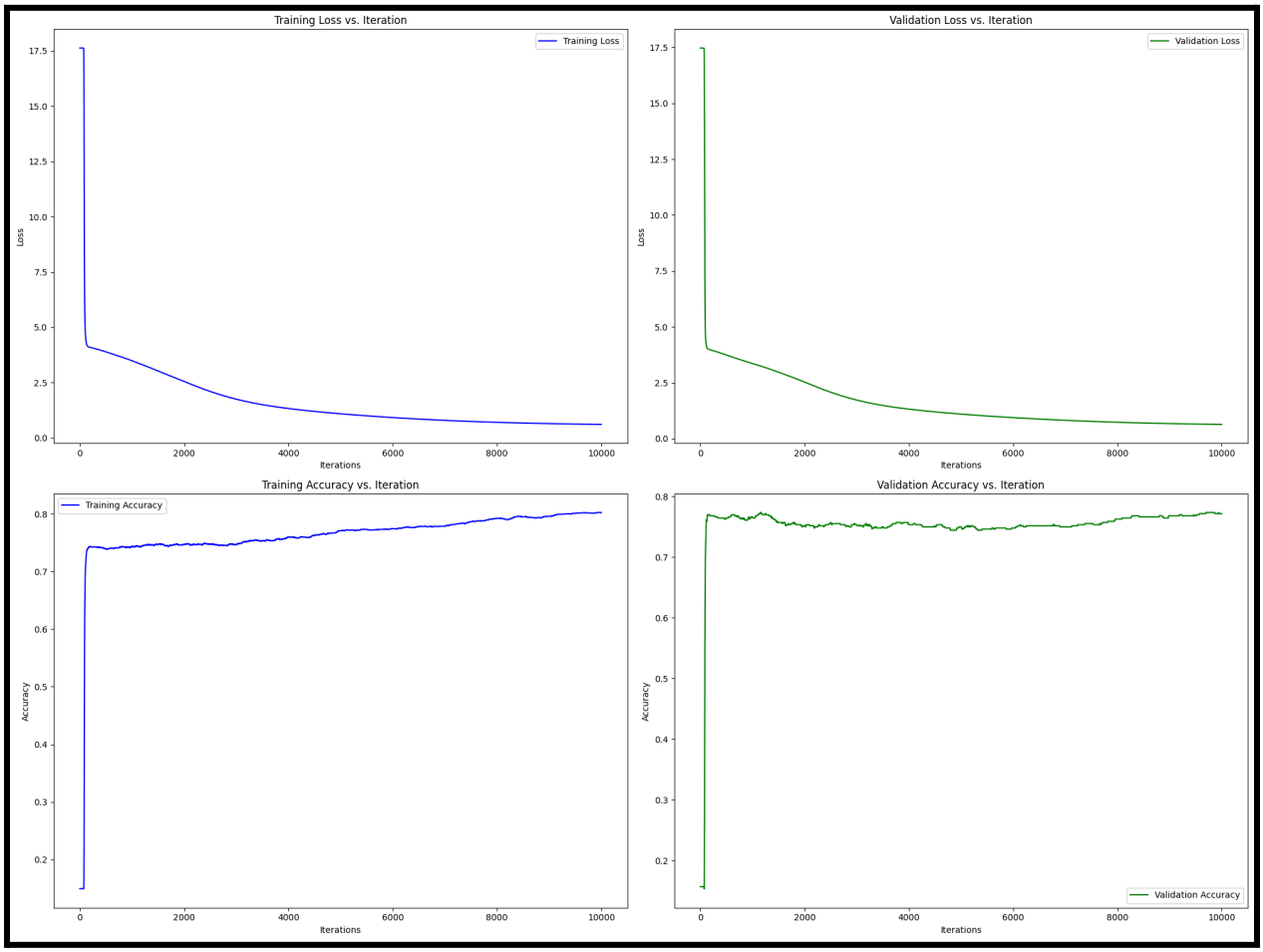
Section B (Scratch Implementation)

- Splitted the data set into 70:15:15 (train: test: validation). Used Cross entropy loss function for gradient descent.
- For some data points we don't have data we have NA written in the file
  - Common Approaches for Handling NA Values:
    - **Remove rows with missing values** This is useful when you have only a small number of missing values and removing them won't significantly reduce the dataset size.
    - **Impute missing values** Fill the missing data with an appropriate value. Common imputation strategies include:
    - **Mean/Median imputation**: Replace NA with the mean or median value of the column.
    - **Mode imputation**: Replace NA with the most frequent value in the column (useful for categorical features).

        - **Forward/backward fill**: Replace NA using neighboring values.
    - As, we didn't know the nature of data collection, and the amount of null data points were less I went with **removing null values**.

a)

With non-scaled features in this part, we had to use very low learning rate (`learning_rate = 0.0001`) for gradient decent that resulted in very slow convergence on the model. Validation accuracy converges after 8000 iterations. Taking `learning_rate > order(10^-3)` would result in `loss_explosion` i.e causing the loss to oscillate wildly or even diverge instead of converging to a minimum.



`Test Accuracy: 0.8069`

b)

Feature Scaling And Normalization

- Many machine learning algorithms can run into problems when scales of the features are very different. Some features can overplay others.

- For example in gradient descent we have a single common parameter **η** for all features . This can be problematic if features are of different orders but are changed with the same parameter.
- One solution for this problem is to rescale the features so that they have the same order.

Different ways to scale features

Sure, here are the first three feature scaling methods in Markdown code format:

1. **Min-Max Scaling (Normalization)**

x_scaled = (x - x_min) / (x_max - x_min)
where `x_min` and `x_max` are the minimum and maximum values of the feature, respectively. Making data points between 0 and 1

2. **Standardization**

x_scaled = (x - μ) / σ Distribute the data points normally with μ = 0, σ = 1

Impact of Feature Scaling on Gradient Descent

When training machine learning models, particularly with datasets like `heatdisease.csv`, feature scaling plays a crucial role in the effectiveness of optimization algorithms such as gradient descent.

1. **Different Feature Ranges**

In the `heatdisease.csv` dataset, features such as `age`, `totChol`, and `sysBP` can have significantly different ranges. For example:

- **Age**: 39 to 63
- **Total Cholesterol (totChol)**: 195 to 285
- **Systolic Blood Pressure (sysBP)**: 106 to 180

2. **Effects of No Scaling**

Without scaling, features with larger ranges (e.g., `totChol`) can disproportionately influence the model's predictions. This can lead to:

- **Slow Convergence**: The gradient descent algorithm may take longer to converge as it struggles to balance updates across features.
- **Instability**: The model may overshoot the optimal solution, resulting in erratic training behavior.

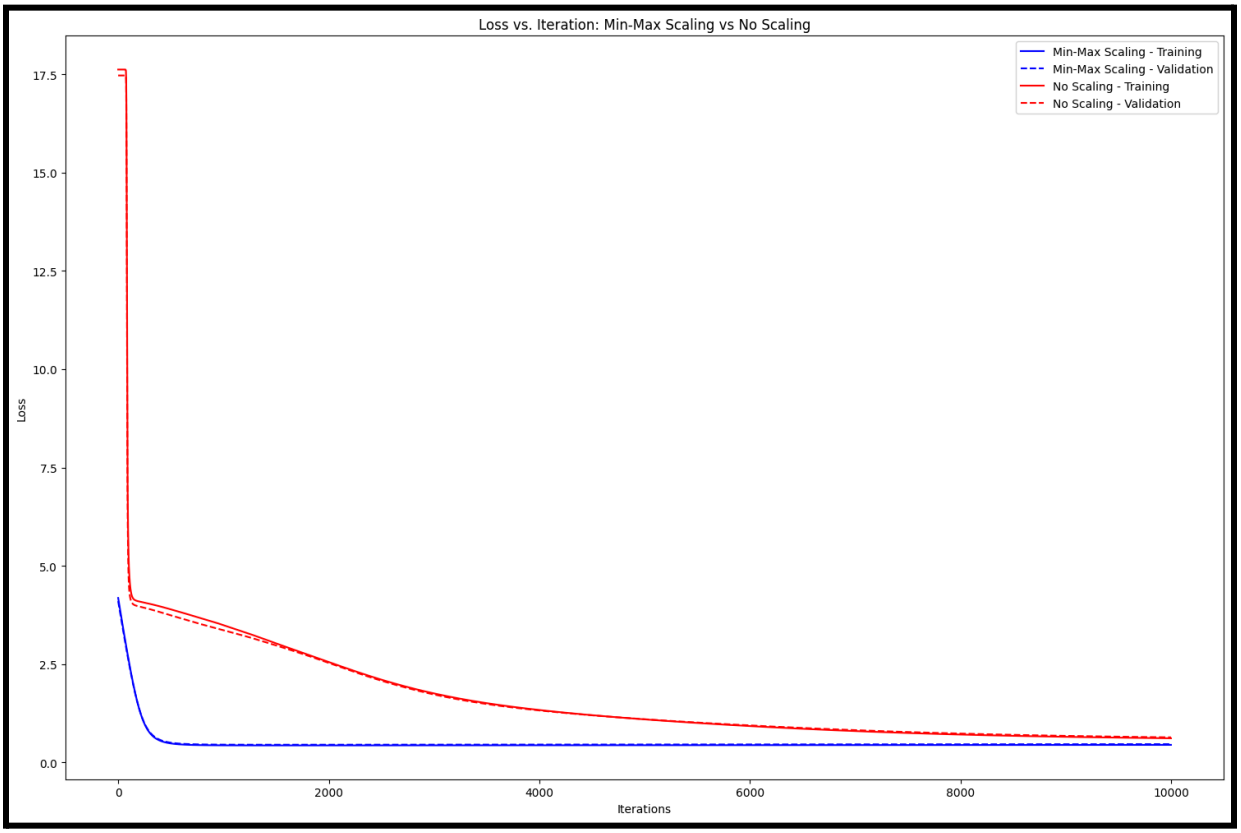
3. **Benefits of Min-Max Scaling**

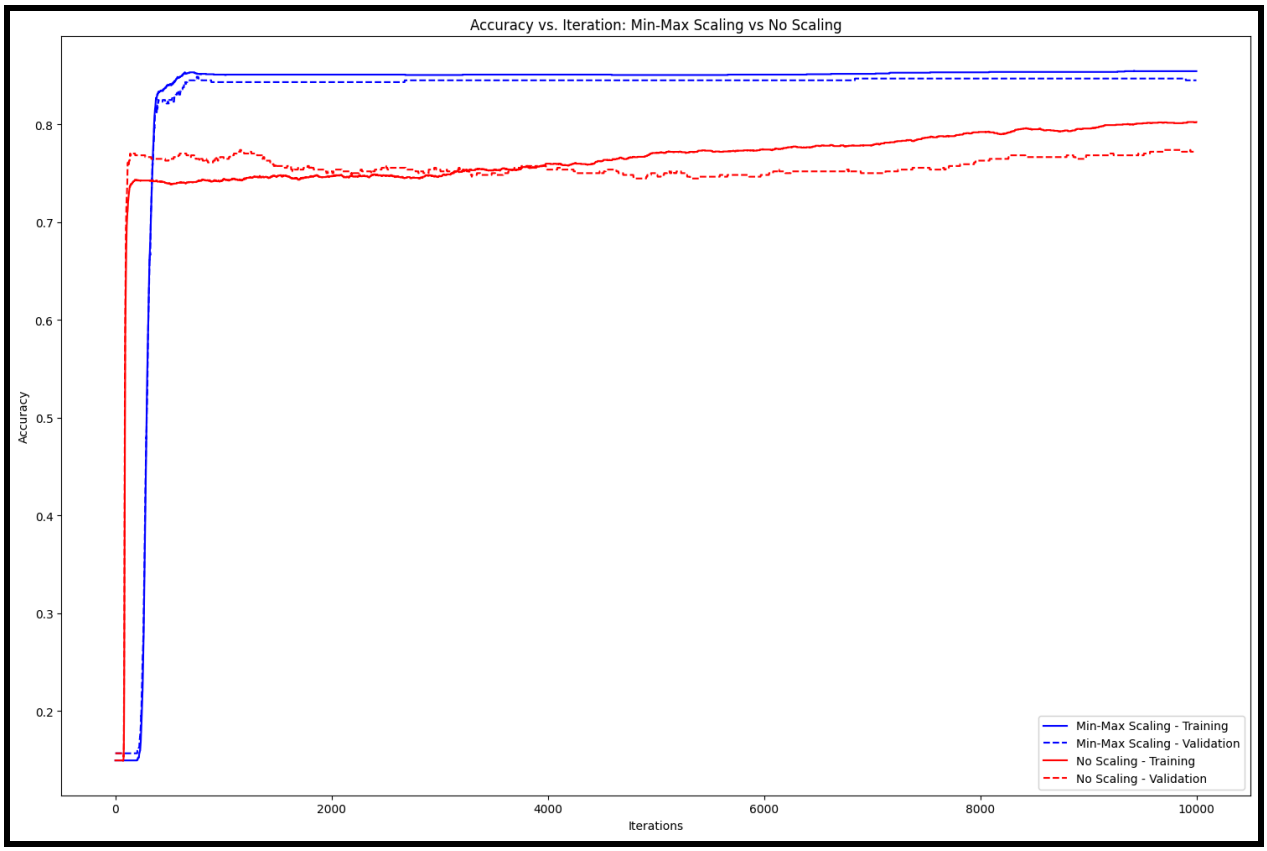
Min-Max scaling transforms features to a common range, typically [0, 1], ensuring that all features contribute equally to the model's learning. Key benefits include:

- **Faster Convergence**: With scaled features, the gradient descent algorithm can converge more quickly and smoothly.
- **Improved Stability**: The updates to model parameters are more balanced, reducing the risk of overshooting the optimal solution.
- **Larger Learning Rates**: Scaling allows for the use of larger learning rates, which can further expedite the training process.

Conclusion

In summary, applying Min-Max scaling to features in datasets like `heatdisease.csv` is essential for enhancing the performance of gradient descent. It ensures that all features are treated equally, leading to faster convergence and more stable training outcomes.





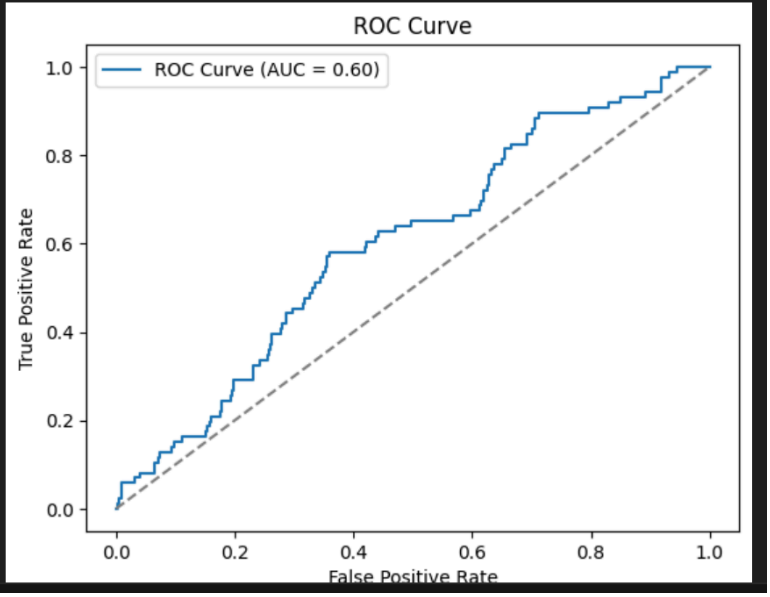
We can see that model using scaled features converges faster and gets higher performance

c)

This is the confusion matrix for the
For the validation set for the model

1. **Confusion Matrix**: A 2x2 matrix summarizing the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). The confusion matrix provides a comprehensive view of the model's performance, highlighting the types of errors made and the overall accuracy.

```
Confusion Matrix:
[[  0 462]
 [  0  86]]
Precision: 0.15693430656934307
Recall: 1.0
F1 Score: 0.27129337539432175
ROC-AUC Score: 0.6009765428370079
```



2. **Precision**: The ratio of correctly predicted positive observations to the total predicted positives:

   Precision = TP / (TP + FP)

   Precision measures the model's ability to avoid false positives, indicating how many of the predicted positive cases are actually positive. It is useful when the cost of false positives is high.

3. **Recall**: The ratio of correctly predicted positive observations to all actual positives:

   Recall = TP / (TP + FN)

   Recall measures the model's ability to identify all positive cases, indicating how many of the actual positive cases were correctly identified. It is useful when the cost of false negatives is high.

4. **F1 Score**: The harmonic mean of precision and recall:

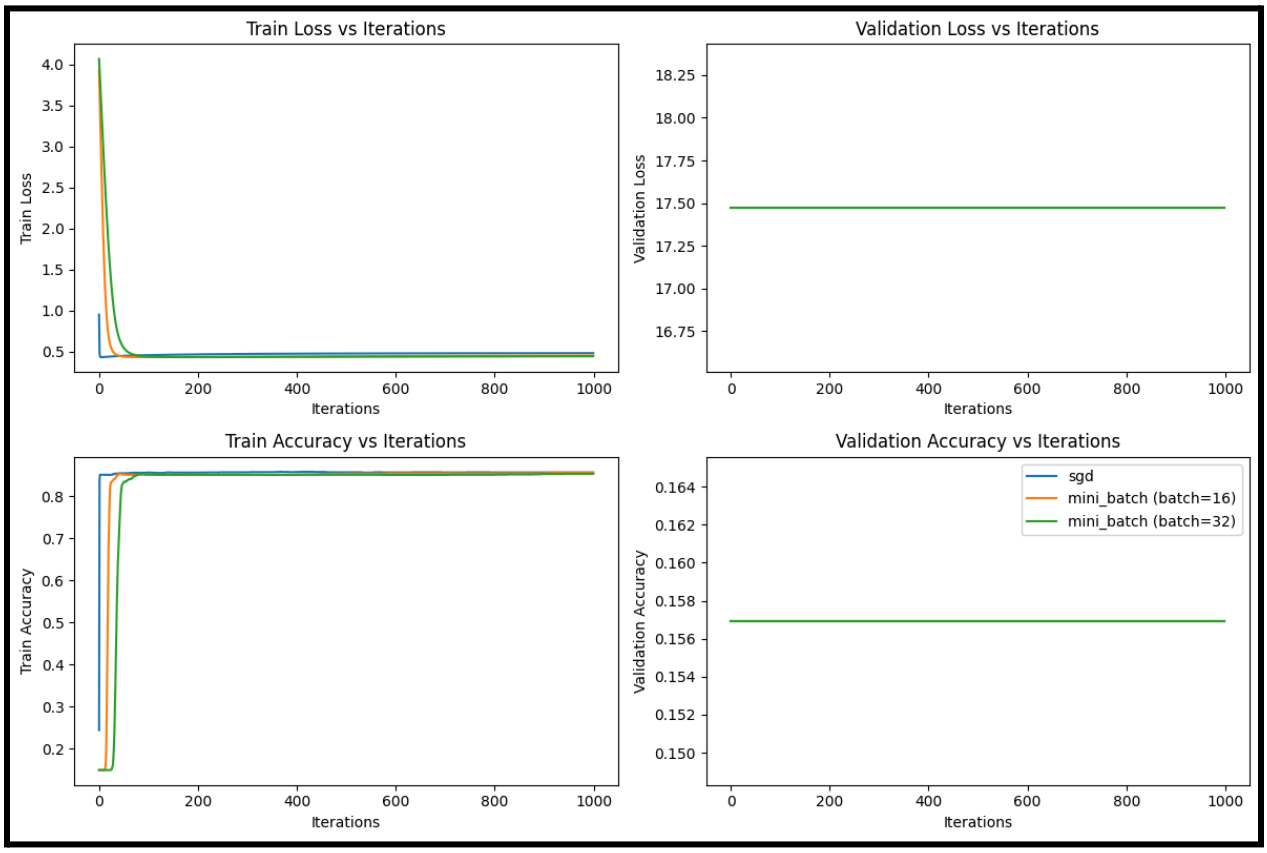
   F1 = 2 × (Precision × Recall) / (Precision + Recall)

   The F1 score provides a **balanced measure of the model's performance**, considering both precision and recall. It is useful when there is a need to balance the trade-off between precision and recall.

5. **ROC-AUC**:

- The area under the Receiver Operating Characteristic curve, which plots the true positive rate (recall) against the false positive rate.
- ROC-AUC measures the model's ability to distinguish between positive and negative classes across different classification thresholds. It is useful for evaluating the overall performance of the model and comparing different models.

d)



To compare Stochastic Gradient Descent (SGD) and Mini-Batch Gradient Descent, we'll modify your gradient descent function to support both methods. Then, we can analyze and plot the results for each optimization algorithm.

Gradient Descent

- Using previous implementation

Stochastic Gradient Descent (SGD)

- In SGD, we update the model's weights after computing the gradient from a single data point. This makes it faster per iteration but more noisy (less stable) in convergence.

Mini-Batch Gradient Descent

- Mini-Batch Gradient Descent is a compromise between Batch Gradient Descent (uses all data points) and SGD (uses one data point). It processes small batches of data, providing a balance between speed and stability.

Key Points:

- **SGD:** We perform an update after each data point.

- **Mini-Batch Gradient Descent:** We perform updates in batches of size 16 and 32 for comparison.
- **Batch Gradient Descent:** all the data is taken into consideration while making a single update to the parameters of our model.

Discussion on Trade-offs:

Stochastic Gradient Descent (SGD):

- **Speed:** Faster for each iteration but can be noisy and less stable in convergence.
- **Convergence:** May oscillate around the optimal solution due to the high variance in gradients.

Mini-Batch Gradient Descent:

- **Speed:** More stable than SGD, while faster than full-batch gradient descent.
- **Convergence:** With the right batch size (e.g., 16 or 32), it offers a good balance between speed and stability.

Batch Gradient Descent:

- **Speed:** Slower for each iteration since it uses the entire dataset.
- **Convergence:** More stable but can take longer to converge due to processing the entire dataset each time.

Best Overall Method:

Stochoatic Gradient Descent performed the best overall:

- It had the lowest validation loss (~0.4750).
- It converged smoothly without fluctuations.
- Although it was slower than Mini-Batch GD initially, it ultimately provided a more stable and better-performing model.

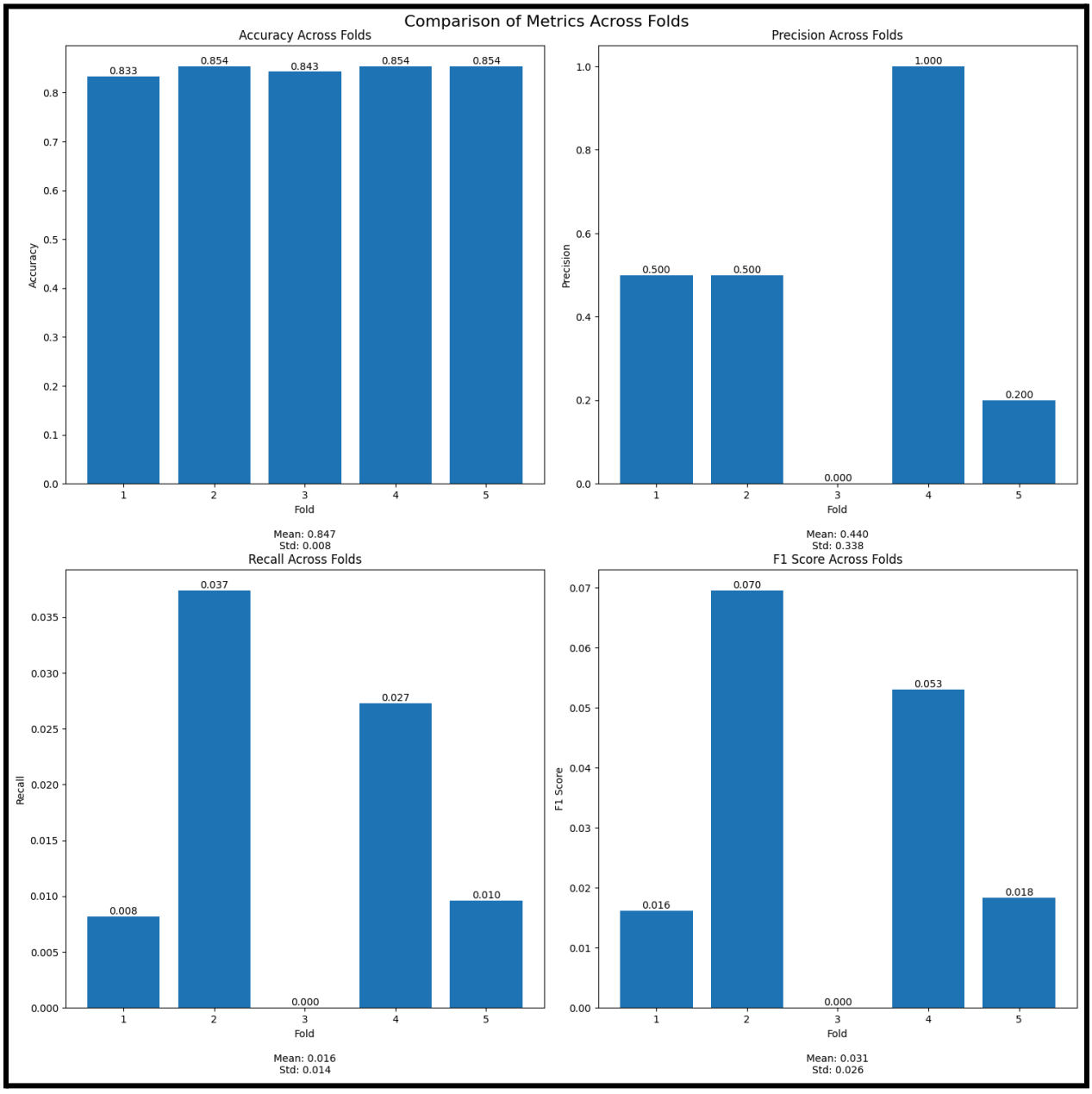
e)

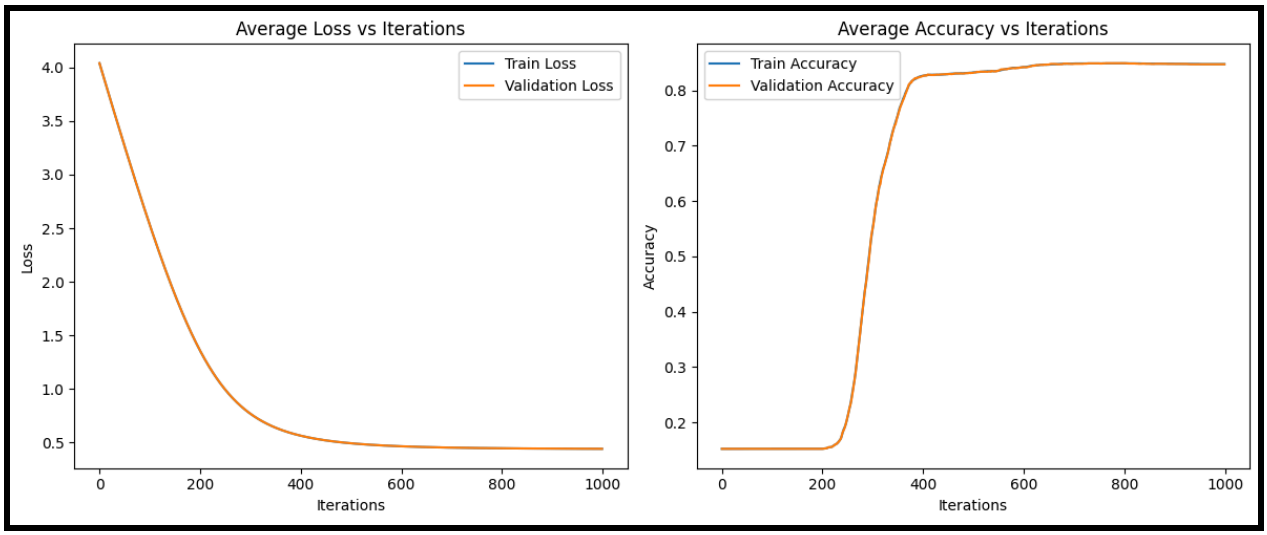
- **Splitted the data set into 5 folds using using train_test_split which helps in finding consistent and robust performance estimates for the models.**

```
--- Cross-Validation Results ---
Average Accuracy: 0.8474 ± 0.0082
Average Precision: 0.4400 ± 0.3382
Average Recall: 0.0165 ± 0.0137
Average F1 Score: 0.0314 ± 0.0258
```

```
Fold1 : Accuracy: 0.8333, Precision: 0.5000, Recall: 0.0082, F1 Score: 0.0161
Fold 2 : Accuracy: 0.8536, Precision: 0.5000, Recall: 0.0374, F1 Score: 0.0696
Fold 3 : Accuracy: 0.8427, Precision: 0.0000, Recall: 0.0000, F1 Score: 0.0000
Fold 5 : Accuracy: 0.8536, Precision: 1.0000, Recall: 0.0273, F1 Score: 0.0531
```

Fold 6 : Accuracy: 0.8536, Precision: 0.2000, Recall: 0.0096, F1 Score: 0.0183





Discussion on Stability and Variance:

- **Stability**: The mean of the metrics across the folds gives you an idea of how the model performs on average.

- **Variance**: The standard deviation shows how much the model's performance fluctuates between different folds. A low standard deviation indicates stable performance, while a high standard deviation suggests the model may not generalize well across different subsets of the data.

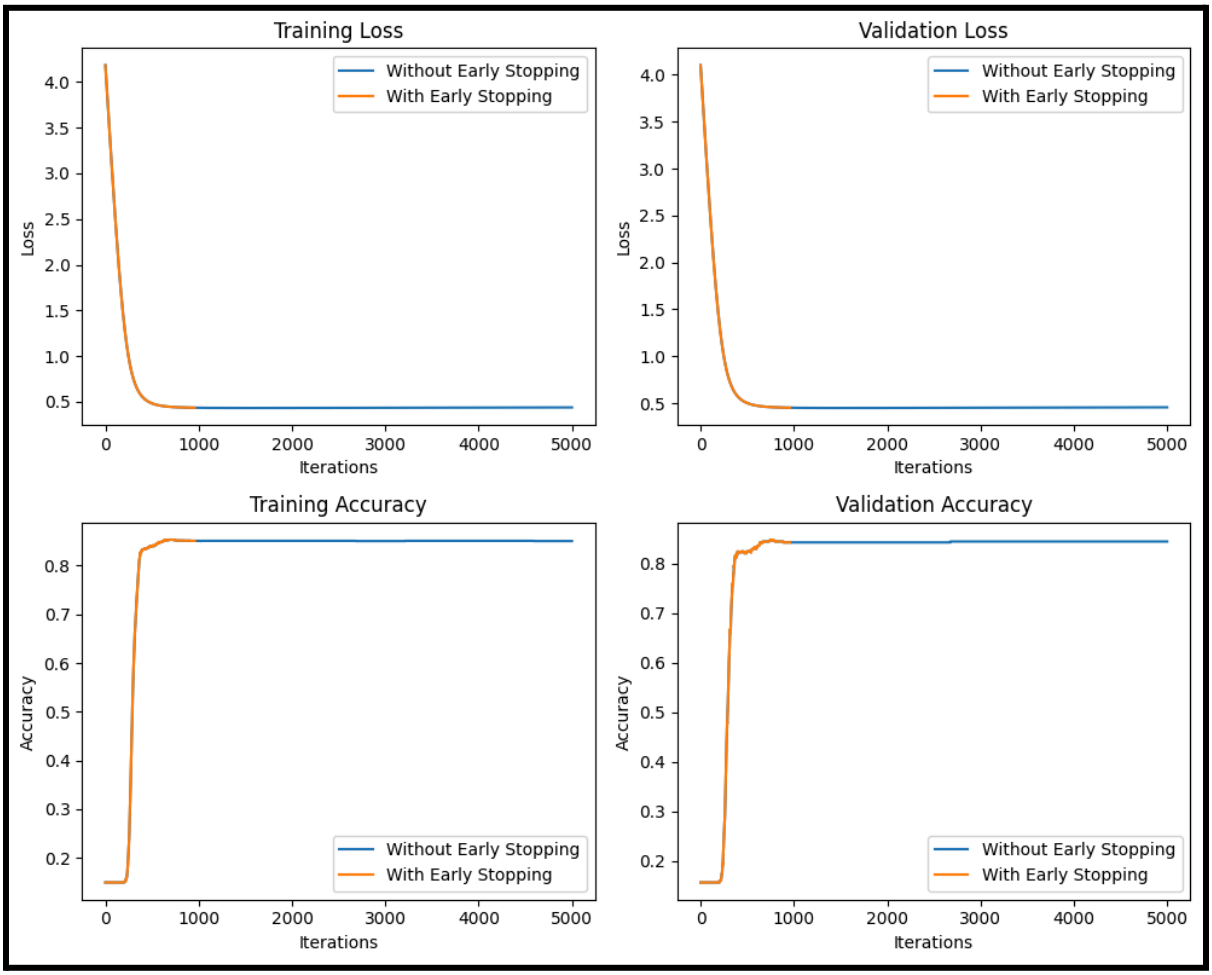
This approach ensures that the model is robust and not overly dependent on the particular train-test split used.

f)

- We will stop training the model when there is no change in model performance after certain number of iterations (Δ) delta.
- **patience:** Number of iterations to wait for improvement before stopping
- **min_delta:** Minimum change in validation loss to qualify as an improvement

```
if val_loss < best_val_loss:
    best_val_loss = val_loss
    best_W = W
    best_b = b
    patience_counter = 0
else:
    patience_counter += 1
```

```
if patience_counter >= patience:
    print(f"Early stopping at epoch {epoch}")
    break
```



Test Accuracy without Early Stopping: 0.8335207912382507
Test Accuracy with Early Stopping: 0.8372334531073221

Using Regularization

- Loss Function without Regularization

$$L_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

L1 Regularization (Lasso)

- L1 regularization adds a penalty equal to the absolute value of the magnitude of the coefficients to the loss function.

- It encourages sparsity in the model by driving some coefficients exactly to zero, effectively performing feature selection.
- Experimenting with different L1 regularization strengths (lambda values) allows you to change penalty strength and control the trade-off between model complexity and sparsity.

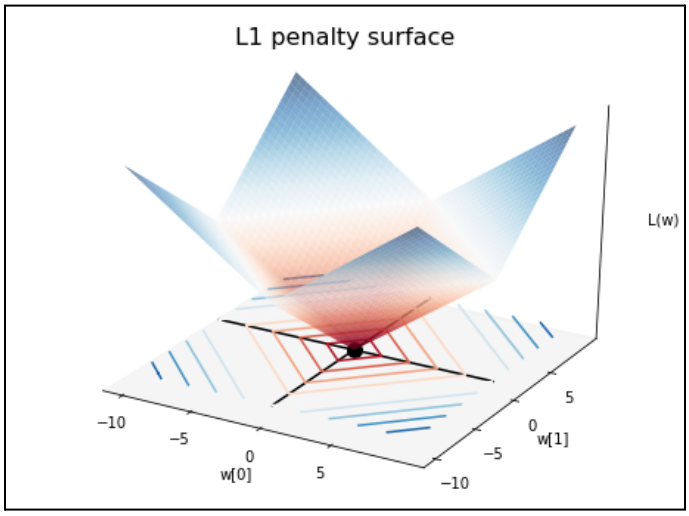
$$L_{\text{L1}} = L_{\text{MSE}} + \lambda \sum_{j=1}^{p} |w_j|$$

L2 Regularization (Ridge)

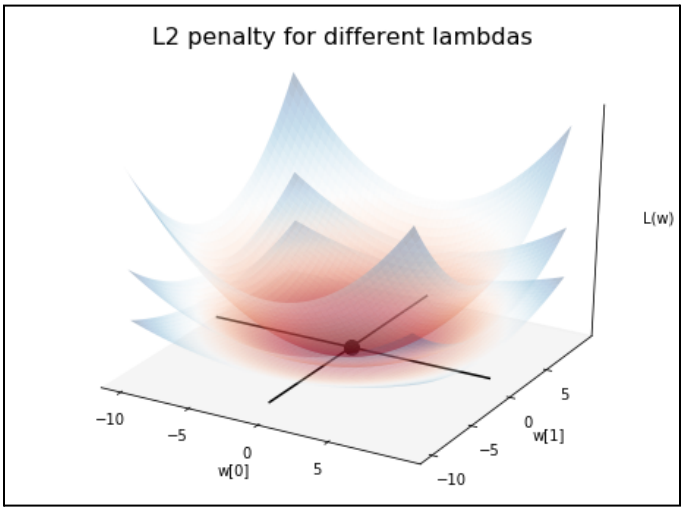
- L2 regularization adds a penalty equal to the square of the magnitude of the coefficients to the loss function.
- It shrinks the coefficients towards zero but does not make them exactly zero.
- Experimenting with different L2 regularization strengths (lambda values) helps control overfitting by limiting the magnitude penalty.

$$L_{\text{L2}} = L_{\text{MSE}} + \lambda \sum_{j=1}^{p} w_j^2$$

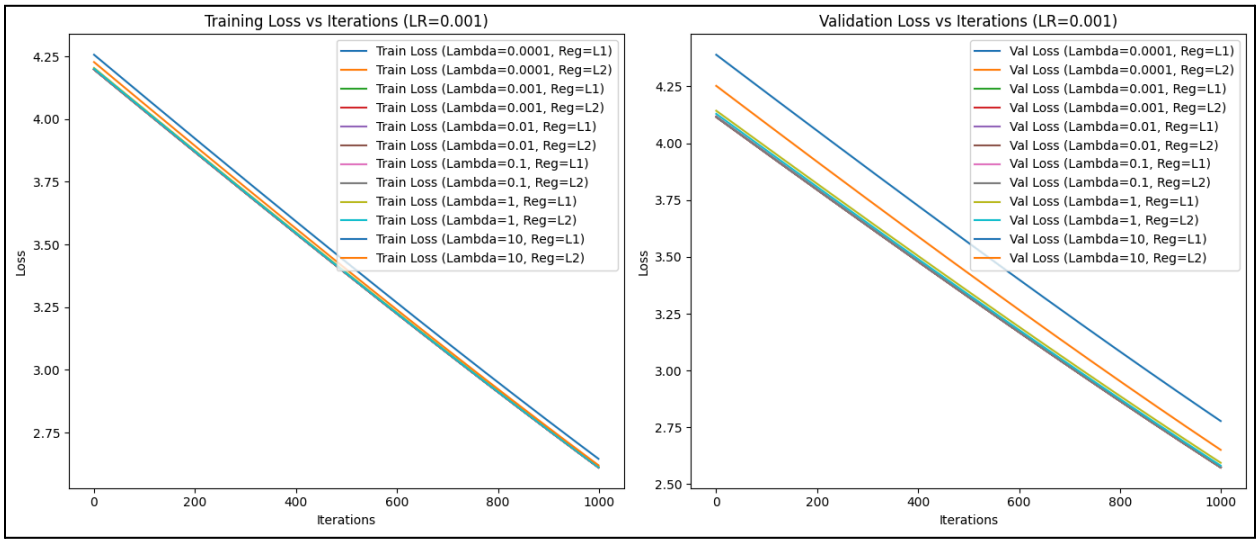
- **`Using regularization will avoid the model to overfit be using complex models that don't generalize well. Hence having low bias but high variance.`**

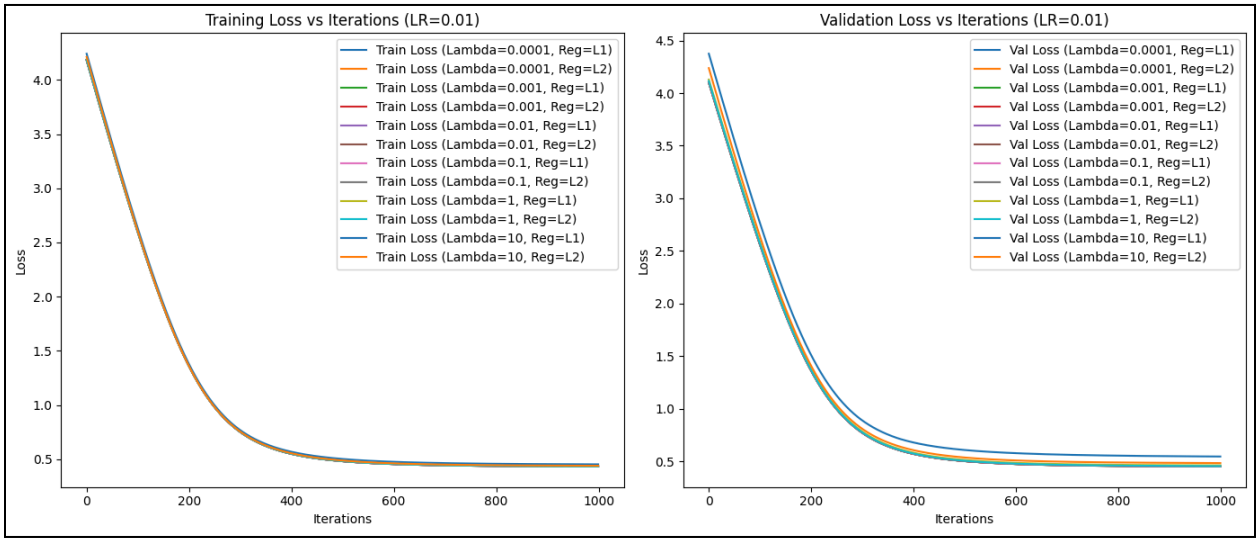


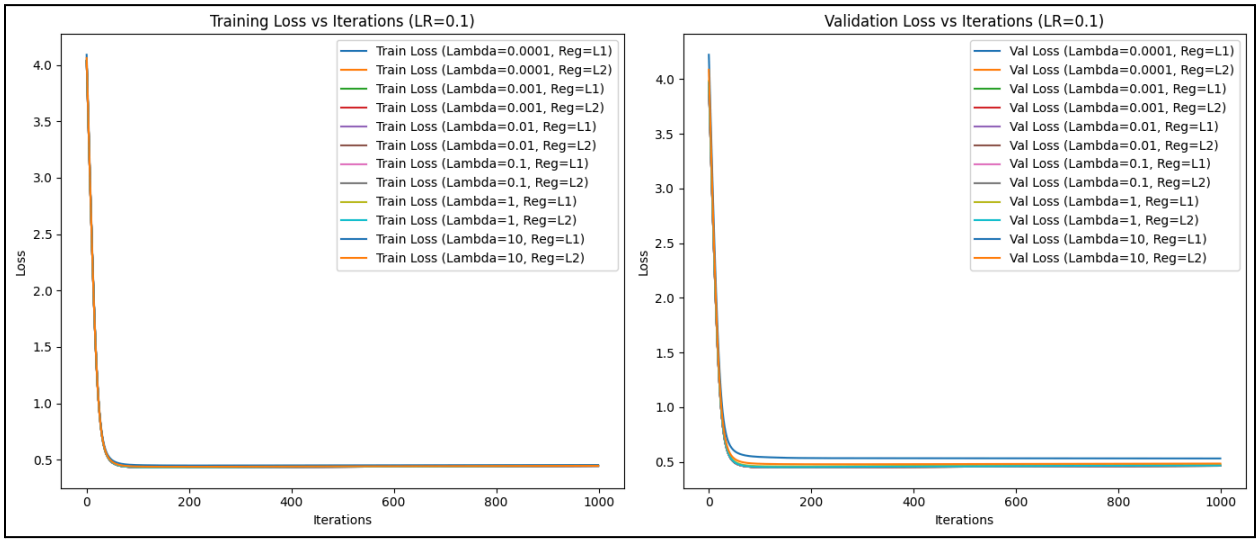
(source: [source])

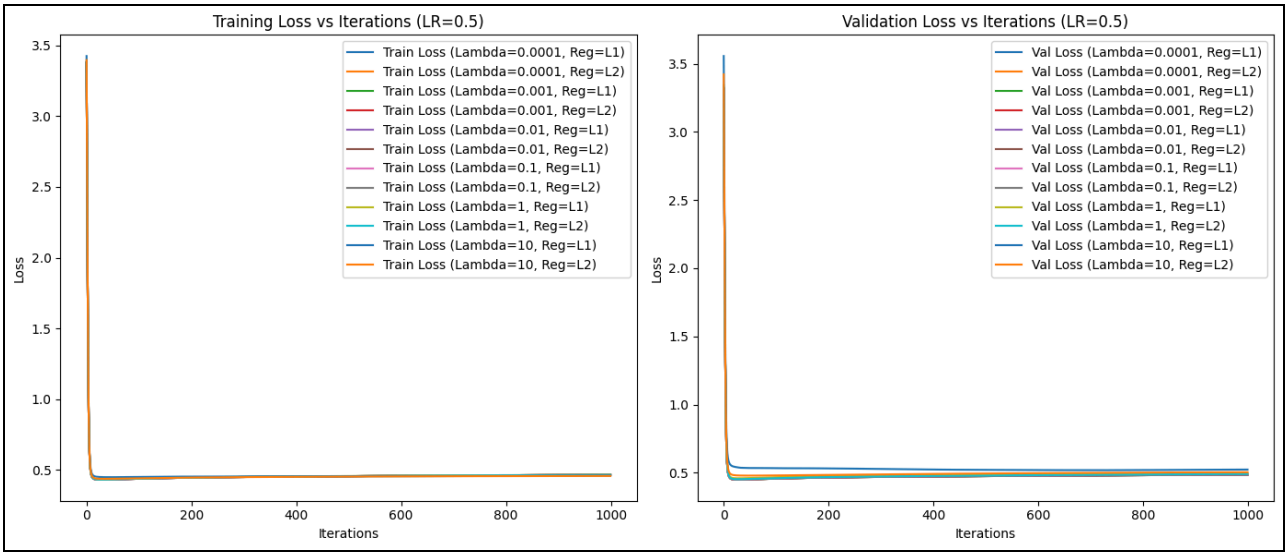


(source: [source])


Training Loss vs Iterations (LR=0.001)
Validation Loss vs Iterations (LR=0.001)
Loss
Iterations
Train Loss (Lambda=0.0001, Reg=L1)
Train Loss (Lambda=0.0001, Reg=L2)
Train Loss (Lambda=0.001, Reg=L1)
Train Loss (Lambda=0.001, Reg=L2)
Train Loss (Lambda=0.01, Reg=L1)
Train Loss (Lambda=0.01, Reg=L2)
Train Loss (Lambda=0.1, Reg=L1)
Train Loss (Lambda=0.1, Reg=L2)
Train Loss (Lambda=1, Reg=L1)
Train Loss (Lambda=1, Reg=L2)
Train Loss (Lambda=10, Reg=L1)
Train Loss (Lambda=10, Reg=L2)
Val Loss (Lambda=0.0001, Reg=L1)
Val Loss (Lambda=0.0001, Reg=L2)
Val Loss (Lambda=0.001, Reg=L1)
Val Loss (Lambda=0.001, Reg=L2)
Val Loss (Lambda=0.01, Reg=L1)
Val Loss (Lambda=0.01, Reg=L2)
Val Loss (Lambda=0.1, Reg=L1)
Val Loss (Lambda=0.1, Reg=L2)
Val Loss (Lambda=1, Reg=L1)
Val Loss (Lambda=1, Reg=L2)
Val Loss (Lambda=10, Reg=L1)
Val Loss (Lambda=10, Reg=L2)


Training Loss vs Iterations (LR=0.01)
Validation Loss vs Iterations (LR=0.01)
Loss
Iterations
Train Loss (Lambda=0.0001, Reg=L1)
Train Loss (Lambda=0.0001, Reg=L2)
Train Loss (Lambda=0.001, Reg=L1)
Train Loss (Lambda=0.001, Reg=L2)
Train Loss (Lambda=0.01, Reg=L1)
Train Loss (Lambda=0.01, Reg=L2)
Train Loss (Lambda=0.1, Reg=L1)
Train Loss (Lambda=0.1, Reg=L2)
Train Loss (Lambda=1, Reg=L1)
Train Loss (Lambda=1, Reg=L2)
Train Loss (Lambda=10, Reg=L1)
Train Loss (Lambda=10, Reg=L2)
Val Loss (Lambda=0.0001, Reg=L1)
Val Loss (Lambda=0.0001, Reg=L2)
Val Loss (Lambda=0.001, Reg=L1)
Val Loss (Lambda=0.001, Reg=L2)
Val Loss (Lambda=0.01, Reg=L1)
Val Loss (Lambda=0.01, Reg=L2)
Val Loss (Lambda=0.1, Reg=L1)
Val Loss (Lambda=0.1, Reg=L2)
Val Loss (Lambda=1, Reg=L1)
Val Loss (Lambda=1, Reg=L2)
Val Loss (Lambda=10, Reg=L1)
Val Loss (Lambda=10, Reg=L2)


Training Loss vs Iterations (LR=0.1)
Validation Loss vs Iterations (LR=0.1)
Loss
Iterations
Train Loss (Lambda=0.0001, Reg=L1)
Train Loss (Lambda=0.0001, Reg=L2)
Train Loss (Lambda=0.001, Reg=L1)
Train Loss (Lambda=0.001, Reg=L2)
Train Loss (Lambda=0.01, Reg=L1)
Train Loss (Lambda=0.01, Reg=L2)
Train Loss (Lambda=0.1, Reg=L1)
Train Loss (Lambda=0.1, Reg=L2)
Train Loss (Lambda=1, Reg=L1)
Train Loss (Lambda=1, Reg=L2)
Train Loss (Lambda=10, Reg=L1)
Train Loss (Lambda=10, Reg=L2)
Val Loss (Lambda=0.0001, Reg=L1)
Val Loss (Lambda=0.0001, Reg=L2)
Val Loss (Lambda=0.001, Reg=L1)
Val Loss (Lambda=0.001, Reg=L2)
Val Loss (Lambda=0.01, Reg=L1)
Val Loss (Lambda=0.01, Reg=L2)
Val Loss (Lambda=0.1, Reg=L1)
Val Loss (Lambda=0.1, Reg=L2)
Val Loss (Lambda=1, Reg=L1)
Val Loss (Lambda=1, Reg=L2)
Val Loss (Lambda=10, Reg=L1)
Val Loss (Lambda=10, Reg=L2)

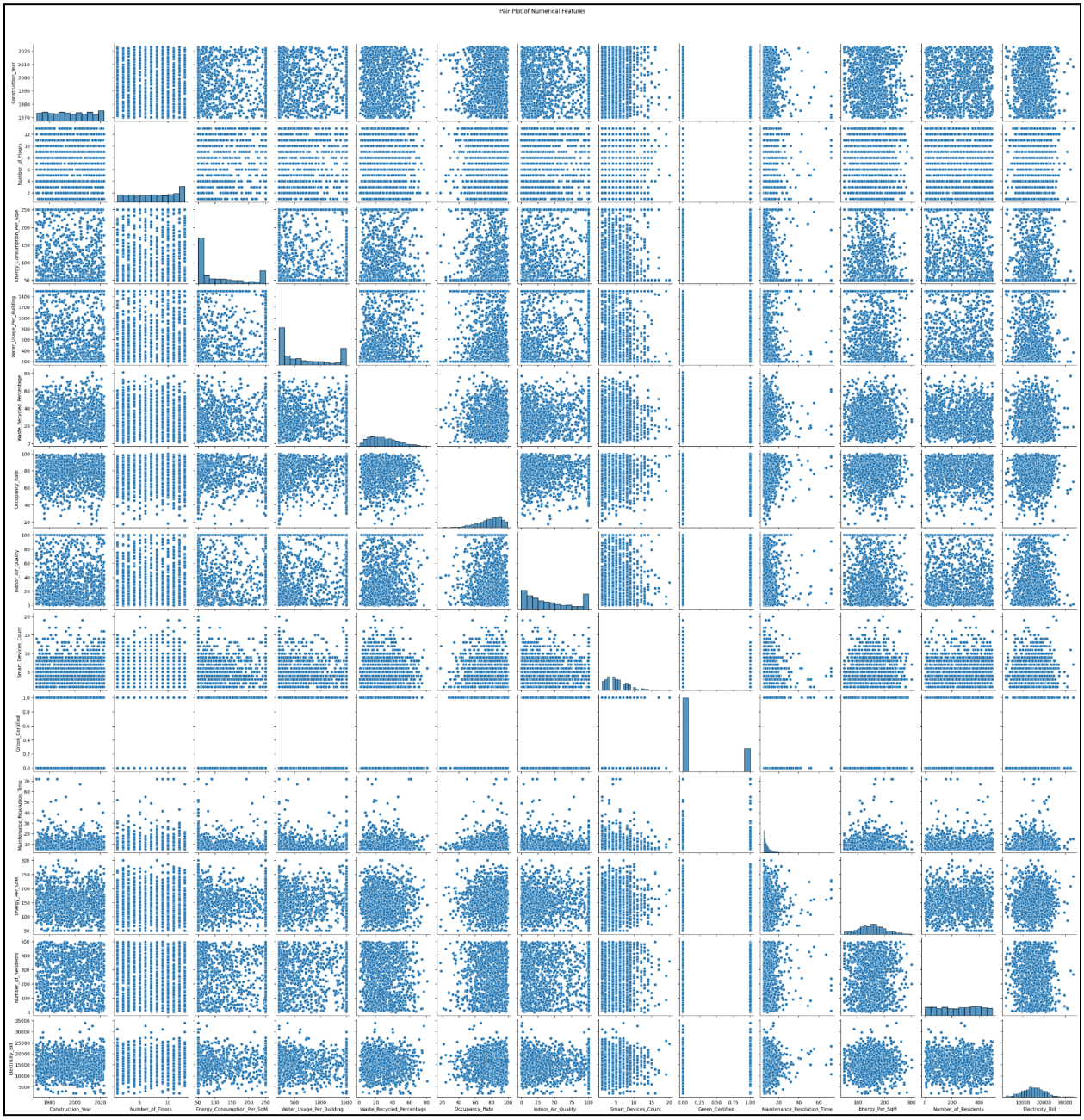


We can see that with high learning rates regularization doesn't play much role as the model converges very quickly.
But for lower learning rates, the model with (↑)lambdas shows little slower decrease in training error.

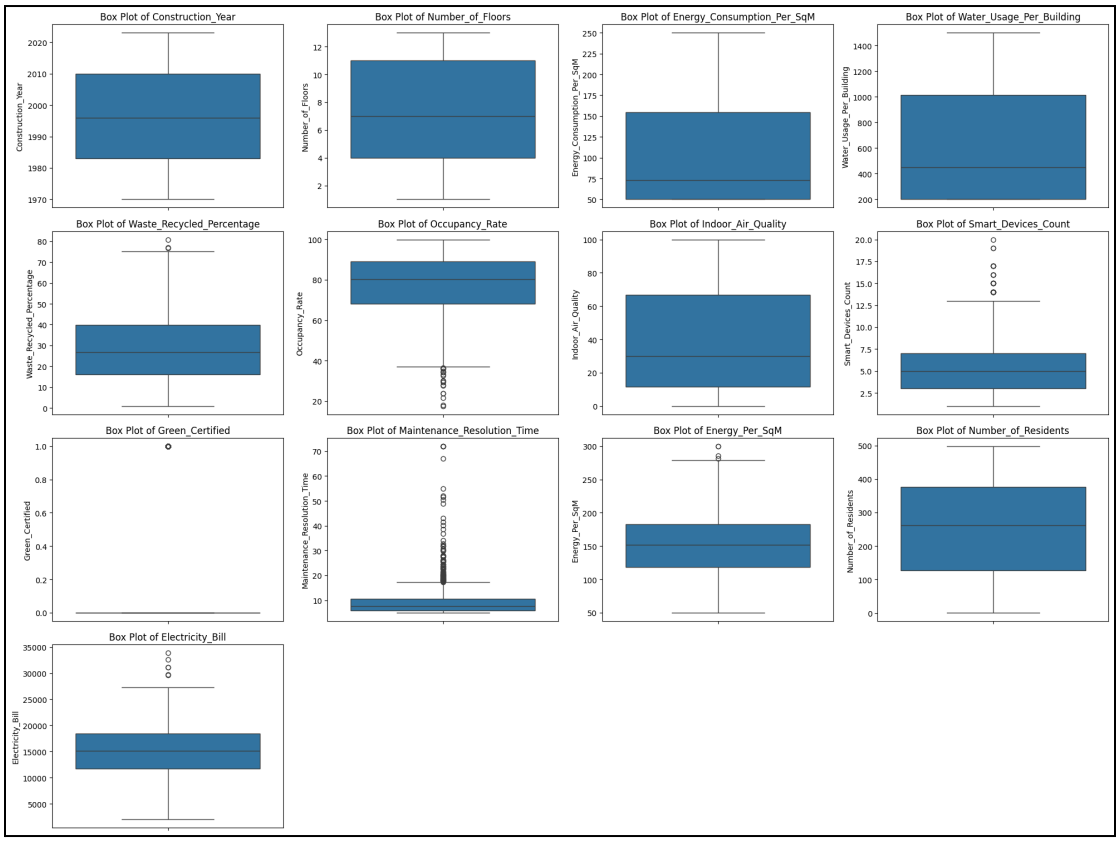
Section C (Algorithm implementation using packages)

a)

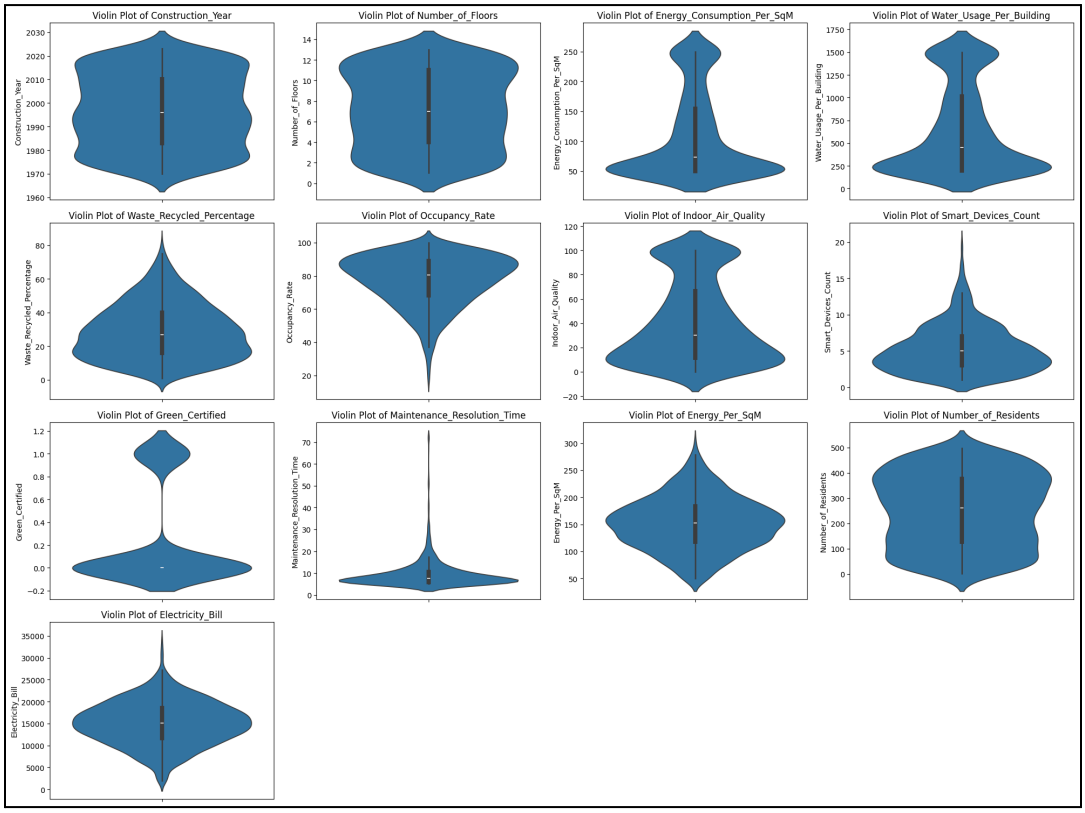
Pair Plots



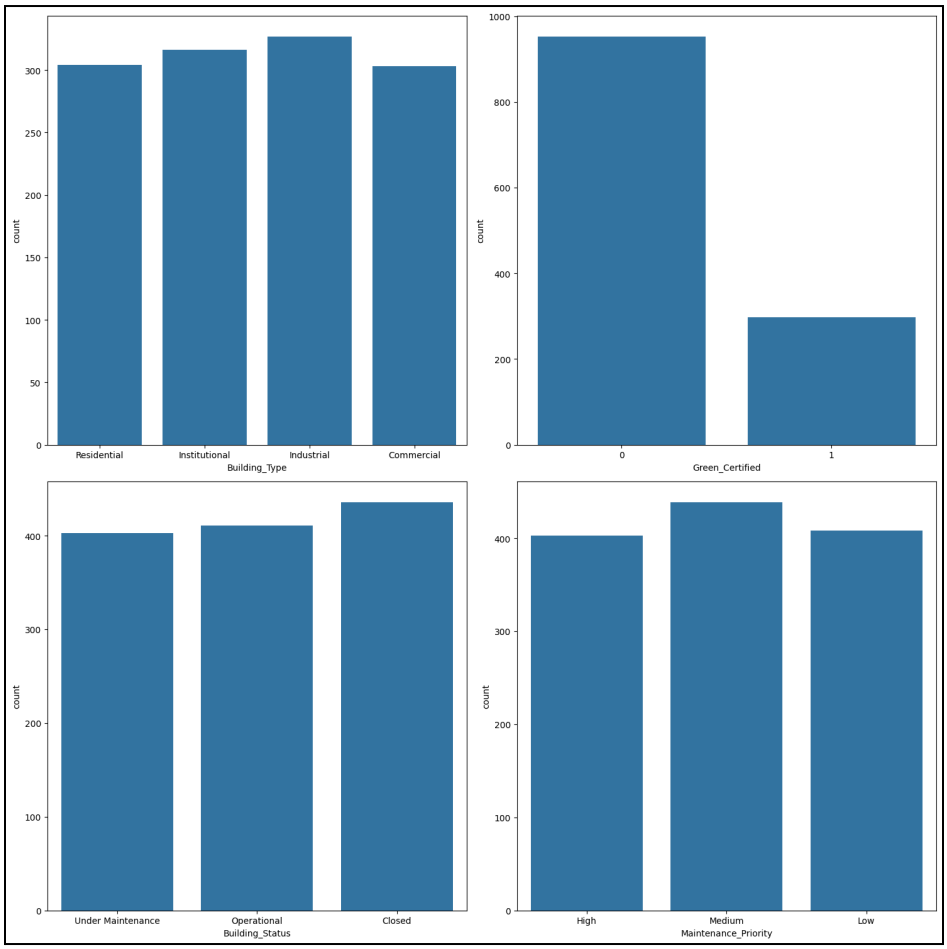
Box Plots



Violin Plots

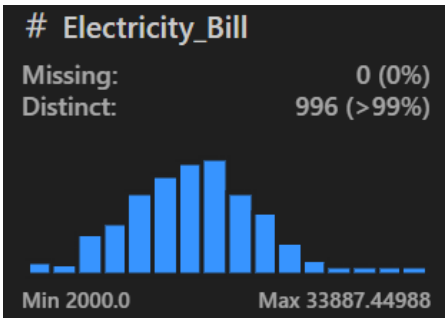


Count Plots

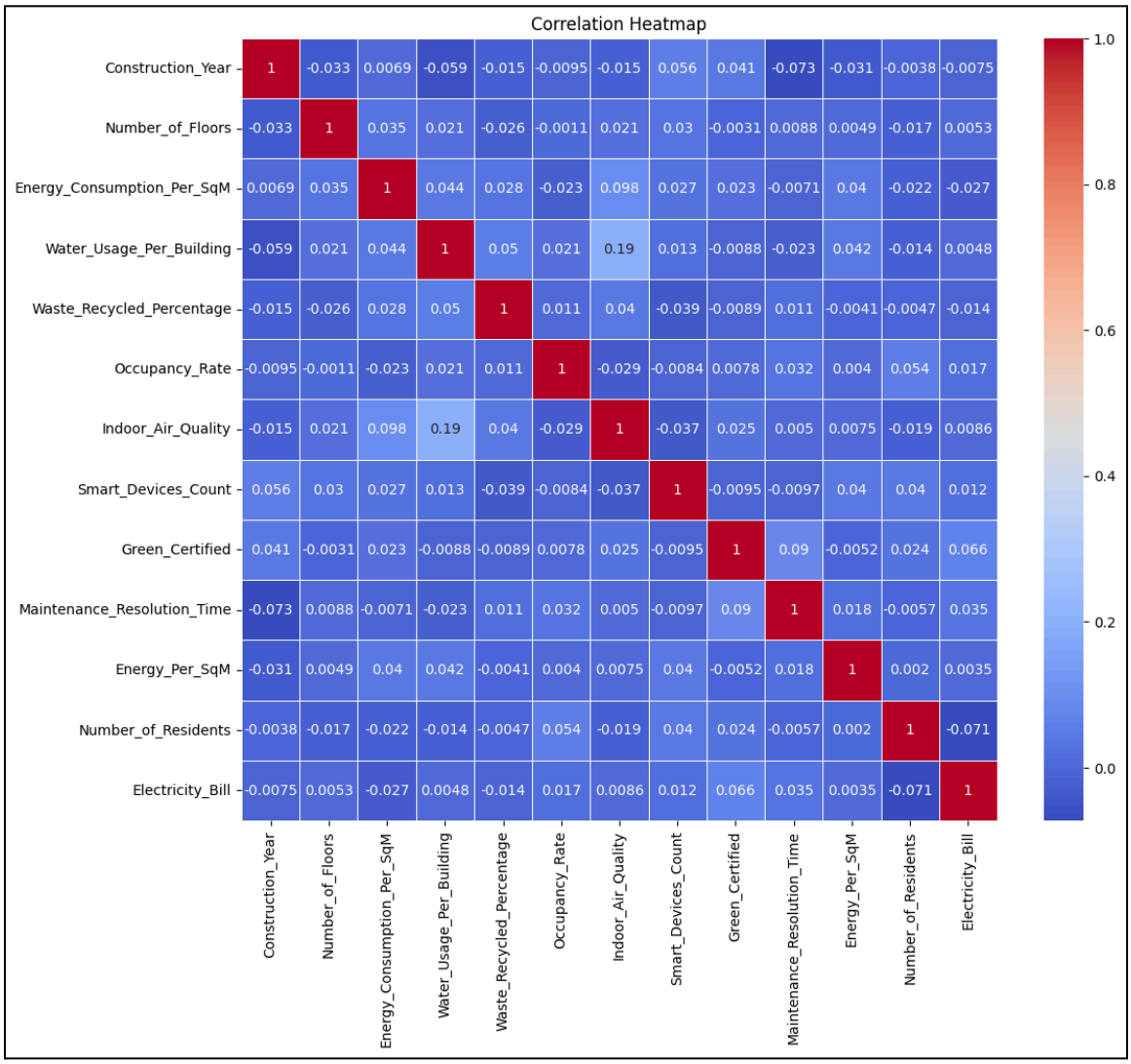


Insights On Data:

1. **Pair Plots :** The pair plots reveal that most relationships between features and the target variable (Electricity_Bill) are not straightforwardly linear. This implies that the interactions between variables might be more intricate, and simple linear models may not effectively capture these patterns.
2. **Correlation Analysis:** The majority of features exhibit low correlation values (close to 0), indicating that they are largely independent of one another. For instance, Construction_Year shows minimal correlation with most other features, suggesting that the building's age does not significantly influence energy consumption, water usage, occupancy rates, or electricity bills.
3. **Count Plot :** The categorical features are generally well-distributed across the dataset, except for the Green_Certified column. The count plot indicates that a significant number of buildings are not green certified, which could impact electricity bills.

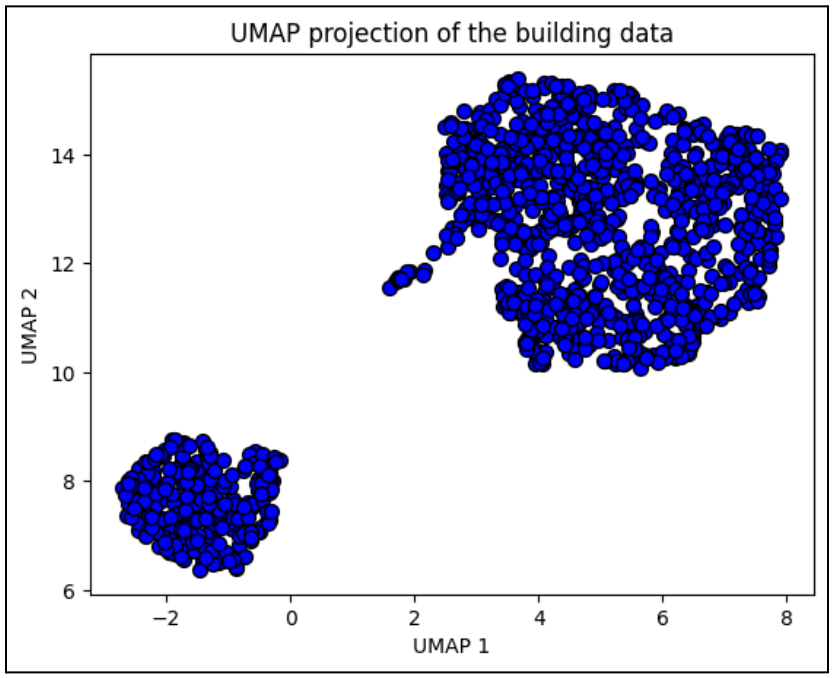


4. The **Electricity_Bill column** follows a **normal distribution** but includes some outliers. These outliers might result from various factors such as unusually high energy consumption or billing errors.
5. The Energy_Consumption_Per_SqM feature displays a broad distribution, with a **median** around **150-200 units.** The **presence of a long tail** suggests that **some buildings have significantly higher energy consumption**, indicating potential inefficiencies or greater operational demands.



b)

- Separated out target feature (electricity_bill) from other features
- we select only the numeric columns. These numeric features are standardized using StandardScaler to ensure that each feature contributes equally in distance calculations.
- Next, we apply UMAP (Uniform Manifold Approximation and Projection) to reduce the dimensionality of the standardized numeric features to two dimensions. Finally, we plot the resulting 2D data using a scatter plot



Separability : The data nicely gets separated into two distinct groups. This means that some buildings share some common characteristics.
Clustering: The clusters are well separated. One cluster is smaller than the other inferring more close relationships. For the biggest cluster some points are drifting away from the cluster indicating outlaw behaviour.

c)

Step 1: preprocessing

- Preprocessing Handling Missing Values (There aren't any):
- We use LabelEncoder to convert categorical features into numerical values.We use **StandardScaler to standardize numerical features** so that they have a mean of 0 and a standard deviation of 1. Splitting the Data:

```
Missing values in each column:
 Building_Type                  0
Construction_Year               0
Number_of_Floors                0
Energy_Consumption_Per_SqM      0
Water_Usage_Per_Building        0
Waste_Recycled_Percentage       0
Occupancy_Rate                  0
Indoor_Air_Quality              0
Smart_Devices_Count             0
Green_Certified                 0
Maintenance_Resolution_Time     0
Building_Status                 0
Maintenance_Priority            0
Energy_Per_SqM                  0
Number_of_Residents             0
Electricity_Bill                0
dtype: int64
```

- We split the data into **training and testing sets using train_test_split**.

Step 2: Initializing and Fitting the Model:

- We initialize a LinearRegression model and fit it to the training data. Printing Model Coefficients and Intercept:
- We print the coefficients and intercept of the fitted model to understand the learned relationships.

Feature	Coefficient
Construction_Year	-60.9494
Number_of_Floors	87.0814
Energy_Consumption_Per_SqM	-79.7104
Water_Usage_Per_Building	-80.3788
Waste_Recycled_Percentage	-85.1549
Occupancy_Rate	45.9943
Indoor_Air_Quality	119.091
Smart_Devices_Count	-117.211
Green_Certified	280.017
Maintenance_Resolution_Time	167.072
Energy_Per_SqM	-29.6458
Number_of_Residents	-349.736
Building_Type_Industrial	257.656
Building_Type_Institutional	261.945
Building_Type_Residential	-317.913
Building_Status_Operational	-5.26472
Building_Status_Under Maintenance	-121.883
Maintenance_Priority_Low	-250.645
Maintenance_Priority_Medium	-67.9845
Intercept	15031.7

Step 3:

We use the fitted model to make predictions on both the training and testing data.

Step 3 : Calculating Evaluation Metrics

- **Mean Squared Error (MSE)**: Measures the average squared difference between actual and predicted values.
- **Root Mean Squared Error (RMSE)**: The square root of MSE, providing error in the same units as the target variable.
- **Mean Absolute Error (MAE)**: Measures the average absolute difference between actual and predicted values.
- **R-squared (R2)**: Indicates the proportion of variance in the target variable explained by the model.
- **Adjusted R-squared**: Adjusts the R2 score based on the number of predictors, providing a more accurate measure for models with multiple predictors.

Metric	Train Set	Test Set
Mean Squared Error (MSE)	24188925.2503	24130184.6421
Root Mean Squared Error (RMSE)	4918.2238	4912.2484
Mean Absolute Error (MAE)	3976.6982	3797.4850
R2 Score	0.0254	0.0061
Adjusted R2 Score	0.0066	-0.0760

d)

Feature Selection using RFE

Initialize and Fit RFE with a LinearRegression Model to Select the Top 3 Features

We get these three top features :

`[ Building_Type', 'Green_Certified', 'Number_of_Residents ]`

Feature	Coefficient
Building_Type	-224.374
Green_Certified	280.453
Number_of_Residents	-344.846
Intercept	15366.3

After retraining the model with with selected features we get these results

Metric	Train	Test
Mean Squared Error	2.4569e+07	2.39414e+07
Root Mean Squared Error	4956.72	4893
Mean Absolute Error	4006.47	3813.95
R-squared	0.0101345	0.0139015
Adjusted R-squared	0.00715302	0.00187592

Comparing with c) :

- MeanSquaredError and MeanAbsoluteError remains almost same for both Train and Test set for both the models.
- Although, there is changes in R-squared for model with three features R-squared is less than R-squared when using all features indicating that we were better able to explain the data using only top thee features.
- The top three features although ae important but only able to explain the data significantly better than using all features.

e)

Performing Ridge Regression(L2) on the one-hot encoded dataset. We get these results

Metric	Train	Test
Mean Squared Error	2.47549e+07	2.17193e+07
Root Mean Squared Error	4975.43	4660.4
Mean Absolute Error	3975.41	3811.05
R-squared	0.0260932	0.0103171
Adjusted R-squared	0.00721135	-0.0714393

Comparing with c) :

- There is a big increase is R-sqaured in Test set from part c) which shows that One-Hot Encoding enabled better use of categorical features that helped better explain the data using the model.

f)

ISA with 4 components:

Metric	Value
n_components	4
MSE (Train)	2.52237e+07
RMSE (Train)	5022.32
MAE (Train)	3992.41
R2 (Train)	0.00765176
Adjusted R2 (Train)	0.00366242
MSE (Test)	2.17705e+07
RMSE (Test)	4665.89
MAE (Test)	3779.17
R2 (Test)	0.00798605
Adjusted R2 (Test)	-0.00821009

ISA with 5 components:

Metric	Value
n_components	5
MSE (Train)	2.5222e+07
RMSE (Train)	5022.15
MAE (Train)	3991.58
R2 (Train)	0.0077169
Adjusted R2 (Train)	0.00272554
MSE (Test)	2.17635e+07
RMSE (Test)	4665.13
MAE (Test)	3774.77
R2 (Test)	0.00830542
Adjusted R2 (Test)	-0.0120162

ISA with 6 components:

Metric	Value
n_components	6
MSE (Train)	2.51224e+07
RMSE (Train)	5012.23
MAE (Train)	3983.2
R2 (Train)	0.0116345
Adjusted R2 (Train)	0.00566252
MSE (Test)	2.19335e+07
RMSE (Test)	4683.32
MAE (Test)	3797.74
R2 (Test)	0.000557007
Adjusted R2 (Test)	-0.0241206

ISA with 8 components:

Metric	Value
n_components	8
MSE (Train)	2.50712e+07
RMSE (Train)	5007.12
MAE (Train)	3983.88
R2 (Train)	0.0136484
Adjusted R2 (Train)	0.00568592
MSE (Test)	2.17112e+07
RMSE (Test)	4659.53
MAE (Test)	3780.68
R2 (Test)	0.0106873
Adjusted R2 (Test)	-0.022153

- Increasing component sizess:
  - Shows constant decrease(⬇) in Mean Squared Error, Root Mean Squared Error (both in *test* and *train*)
  - R-squared *train* also increases, But for *test* there is a sudden drop is train R-squared but increases after that.But that could have happened because of unfortunate test set split, overfitting, component interaction etc.

g)

Alpha	MeanSquaredError (MSE)	RootMeanSquaredError (RMSE)	MeanAbsoluteError (MAE)	R-squared (R2)	AdjustedR-squared (Adj R2)
0.001	2.17259e+07	4661.11	3811.72	0.0100159	-0.0717654
0.01	2.17227e+07	4660.77	3811.4	0.010162	-0.0716073
0.1	2.16954e+07	4657.84	3808.51	0.0114056	-0.0702609
1	2.16215e+07	4649.89	3797.12	0.0147744	-0.0666138
10	2.18105e+07	4670.18	3808.34	0.00616111	-0.0759386
100	2.19427e+07	4684.3	3818.42	0.000139207	-0.082458

- Using preprocessed dataset from part c) with `StandardScaled` features.
- We can notice that with increase in α (*alpha*) :
  - MeanSquaredError first decreases then increases. Taking a minimum value for *alpha = 1* indicating that we can play with α to obtain optimal values.
  - R-squared(R) also obtains maximum value for **α** = 1 `(0.01477)` then decreases inferring model not being able to explain significant portion of dependent variable's variance.

h)

After running Gradient Boosting Regressor we get these results

Metric	Value
Mean Squared Error (MSE)	2.45261e+07
Root Mean Squared Error (RMSE)	4952.38
Mean Absolute Error (MAE)	4092
R-squared	-0.117578
Adjusted R-squared	-0.2099

- We notice that there is a significant **decrease**(⬇) in Mean Squared Error(MSE) and Root Mean Squared Error(RMSE) both suggesting better fitting model.
- But R-squared has decreased from previous results of ElasticNet regularization which means that model is **overfitting** the data.