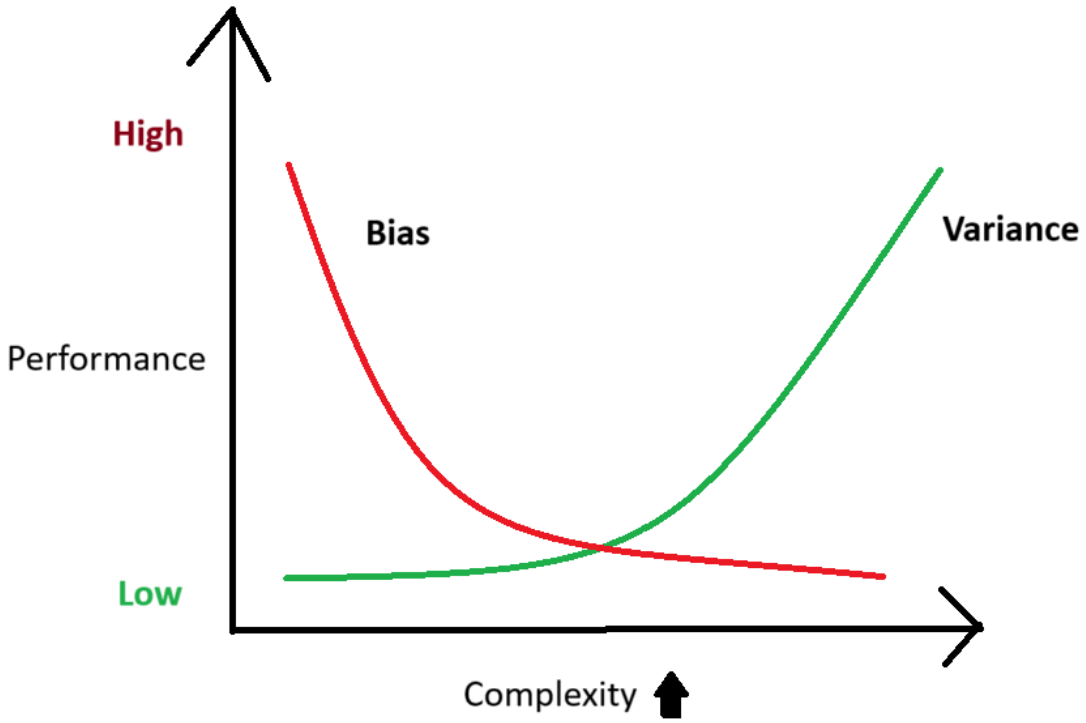
Section A

1.(a) As we increase the complexity of the model, by adding more features or by including higher-order polynomial terms in a regression model, model keeps trying to improve its performance on the train set. This will lead to the model trying to perfect its prediction on the train set, leading to **less bias** but inherently, on new data, it will get weaker, as its trying to memorize the train results, and thus is **overfitting**, so it will have **higher variance**.



1.(b) I am using 2 metrics for the classification performance, precision and recall. Also, I am assuming that the question wants us to find the average of all the performance metrics for spam mail, and then separately, find the average of all the performance metrics for legitimate emails.

		Predicted	
		Spam	Not Spam
Actual	Spam	200	50
	Not Spam	20	730

Precision -> Out of all the cases where your model predicts it to be positive, how many are actually positive.

Recall -> Out of all the cases where it is actually positive, does your model predict it to be positive.

Spam Mail :

Precision = TP / (TP + FP) = 200 / (200 + 20) = 0.909
Recall = TP / (TP + FN) = 200 / (200 + 50) = 0.8
Average = (0.909 + 0.8) / 2 = 0.8545

Legitimate Mail:

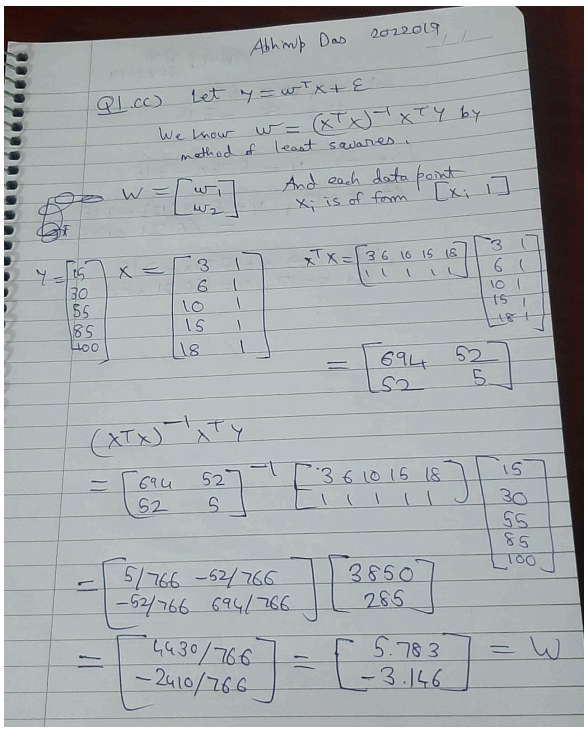
Precision = TP / (TP + FP) = 730 / (730 + 50) = 0.936
Recall = TP / (TP + FN) = 730 / (730 + 20) = 0.973
Average = (0.936 + 0.973) / 2 = 0.9545

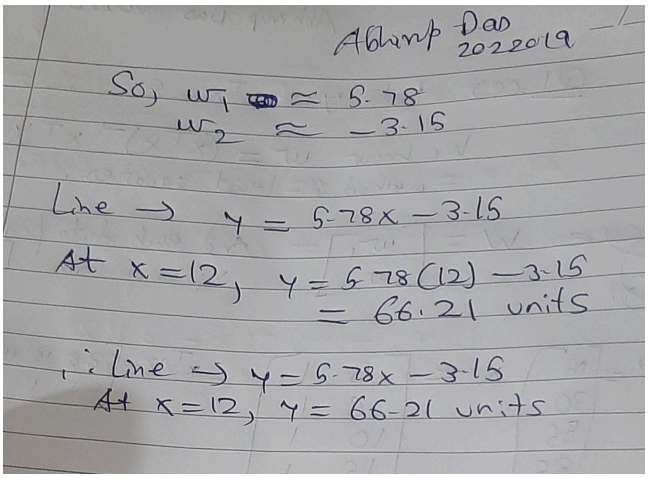
Accuracy = (TP + TN) / (TP + TN + FP + FN) = 930 / 1000 = 0.93

In the above 2 cases, TP, FP and FN are based on what our key objective is. When we are trying to find metrics for which mail are spam, positive is when the mail is found to be spam. Similarly, when we try to find which mail are legitimate, positive is when the mail is found to be legitimate.

Results:- Average for Spam -> 0.8545
Average for Legitimate -> 0.9545
Overall Accuracy -> 0.93

1.(c)





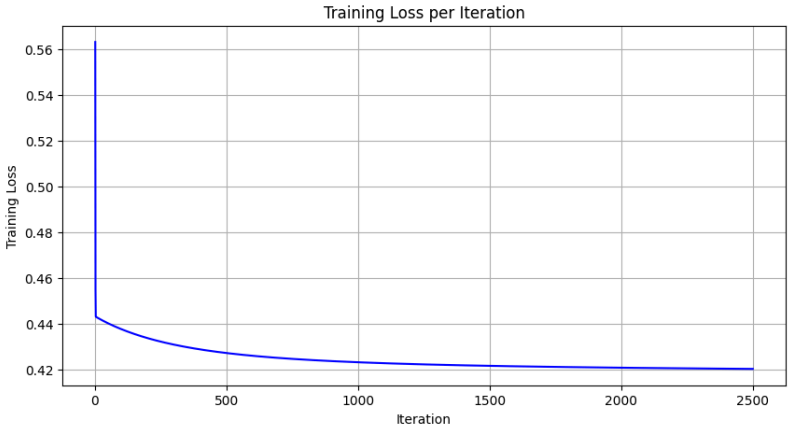
Line -> y = 5.78x - 3.15
At x = 12 units, y = 66.21 units

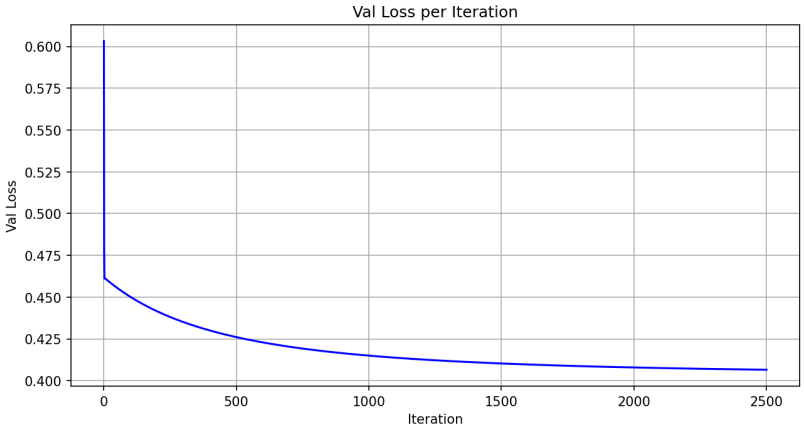
1.(d) Let the train Dataset D = {(1,1), (2,2), (3,4), (4,3)} and the test set that comes from the same distribution as D be {(5,5), (0,0)} where the first coordinate in each tuple is x, and its actual value is the 2nd coordinate y.

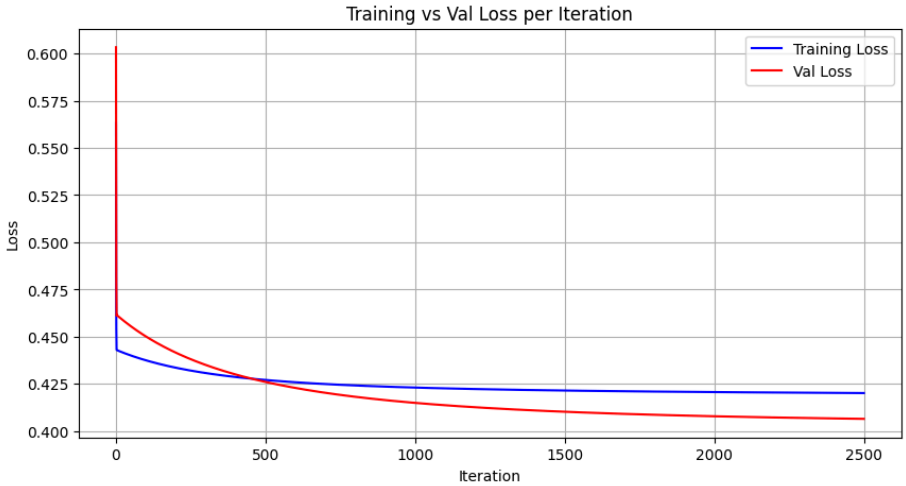
f1 could be a model that fits the 4 points of D like a glove and predicts a curve which is slightly complex. But perhaps due to the noise, the distribution just follows the straight line y = x. Thus, if f2 predicts y = x, it would generalize better than f1, as future datapoints would be coming from this distribution, so the lesser complexity helps. And of course, f1 has lower empirical risk on the train set, since it fits the 4 points exactly, but f2 has a small loss of 0.5*(1^2 + 1^2) = 1, by MSE loss.

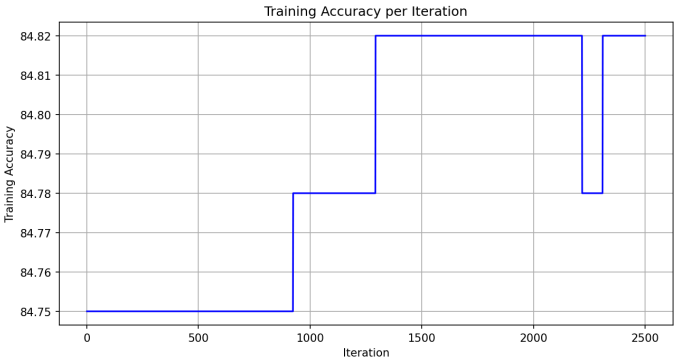
Section B

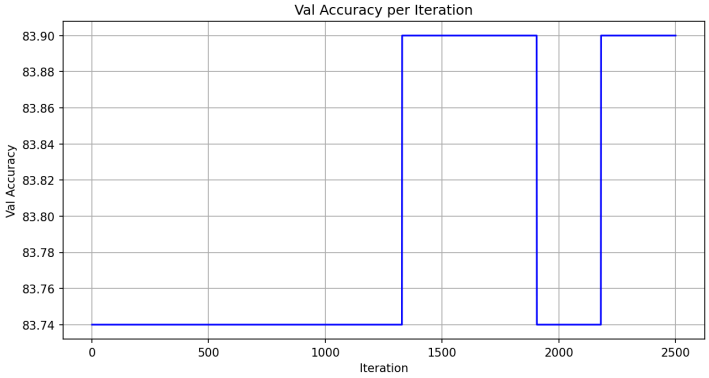
2.(a)

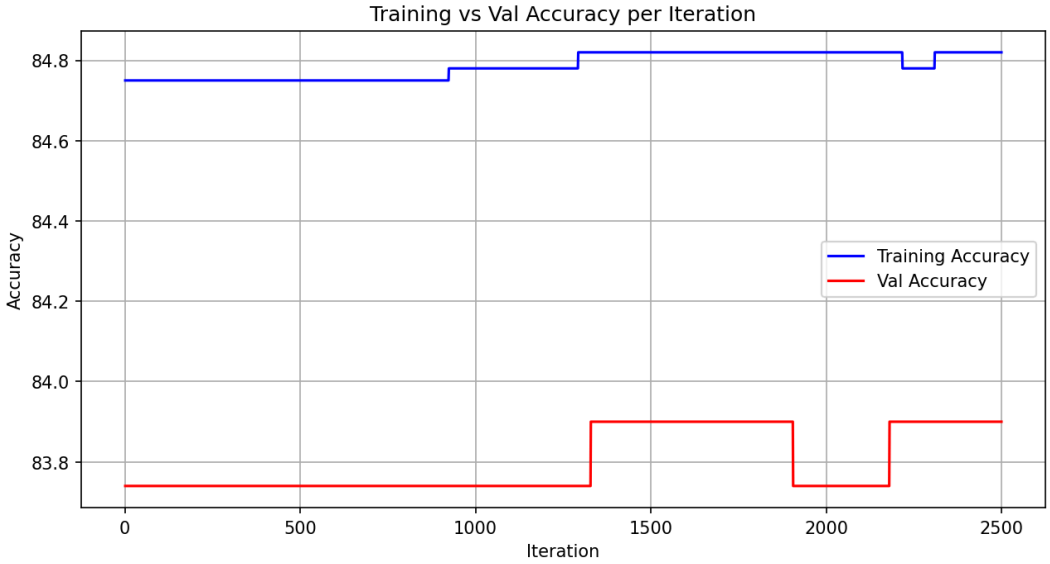










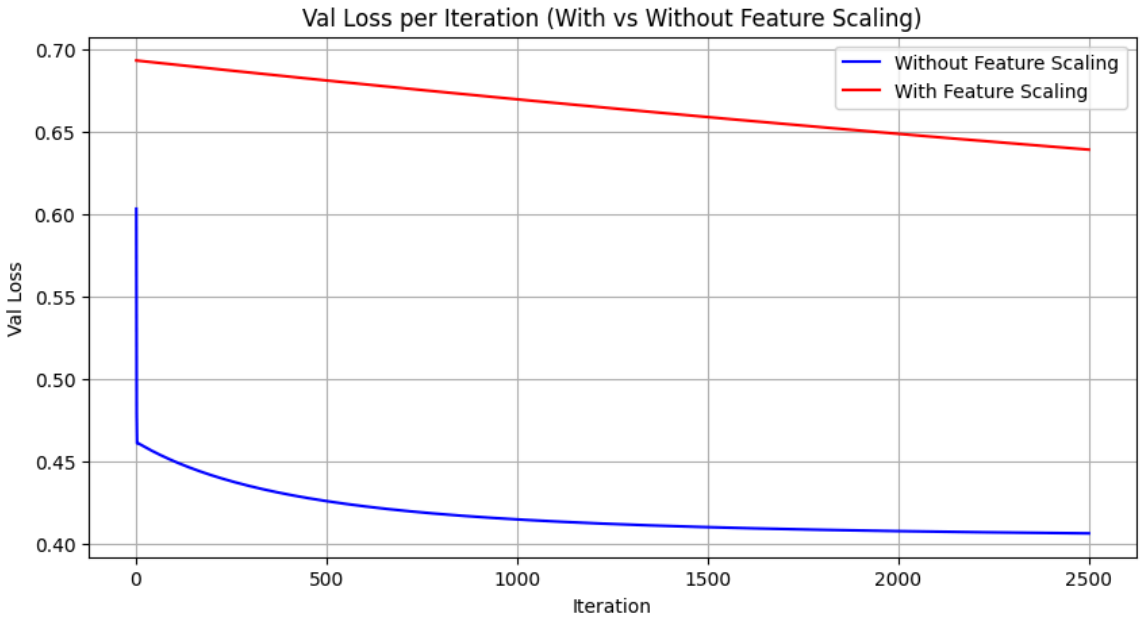


Convergence :- The model converges after 2000 iterations in terms of val loss and also doesn't have much shifts in the accuracy over time.

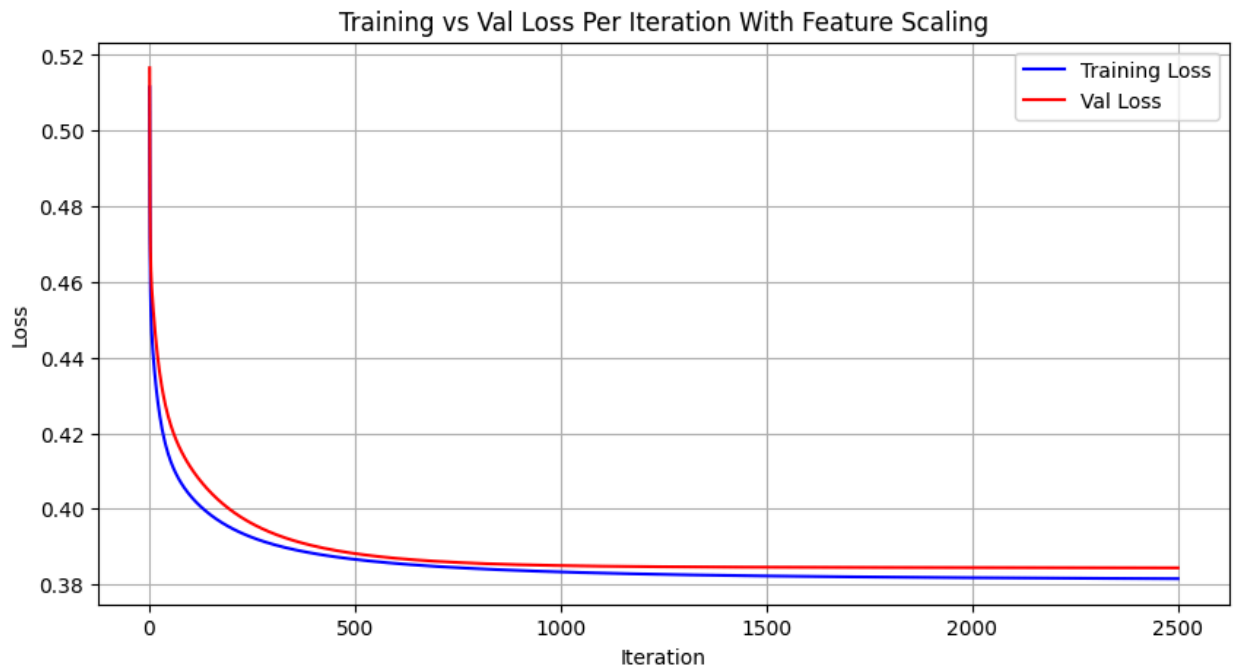
Comparison of Plots :- Generally, training loss should be better than val loss, but in this case the val loss gets better after the initial stages of training. This indicates an imbalance in the splitting of data, or that the model in general isn't overfitting at all, which is a very good sign. In terms of the accuracy, the val accuracy is about 1% less than train accuracy which is normal.

Analysis :- The model decreases pretty quickly and doesn't seem to be overfitting. Even after 2000 iterations, the loss decreases, so the gradient isn't quite at 0 yet, but small enough.

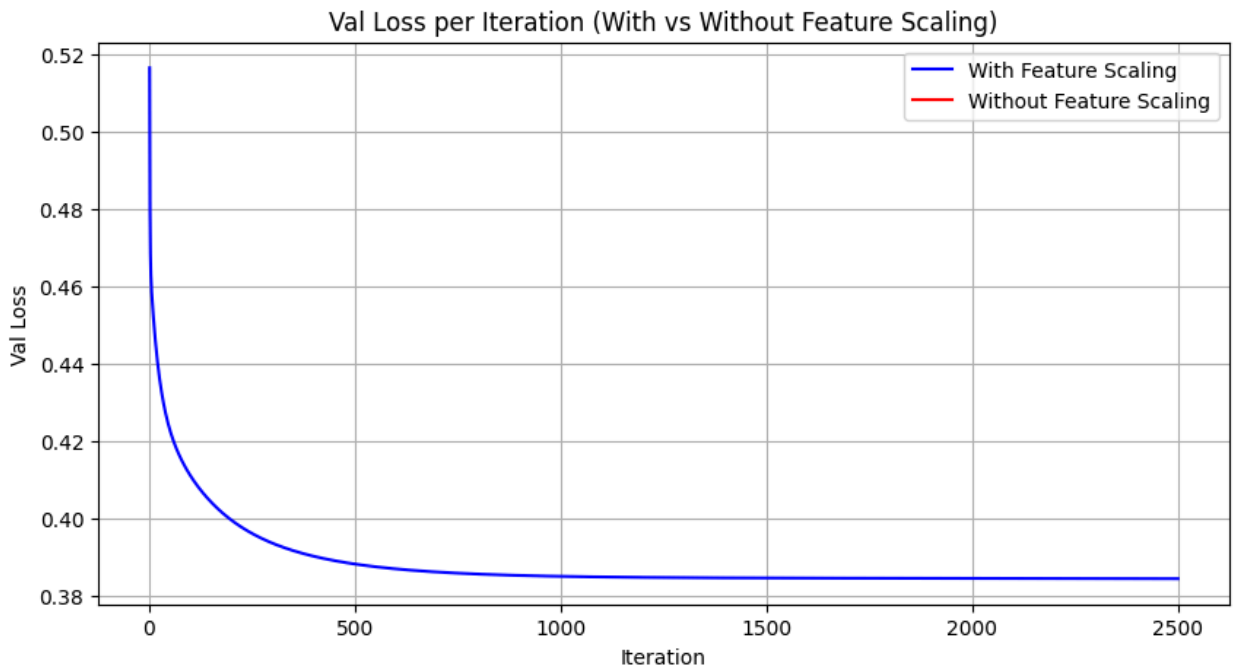
(b)



With a low learning rate like 0.0001, the convergence in regular gradient descent is much faster. With feature scaling, the convergence is really slow with such a learning rate. But with a higher learning rate, the regular gradient descent is really bad, as there is extremely high oscillations due to the learning rate. So, convergence is horrible, in fact, it doesn't converge. But, with a learning rate of 1 for feature scaling, the convergence is quick and leads to a better loss even. For a learning rate of 1 for regular gradient descent, the loss is in fact infinite, so you can't see it on the graph.


Training vs Val Loss Per Iteration With Feature Scaling
Training Loss
Val Loss
Loss
Iteration
0.52
0.50
0.48
0.46
0.44
0.42
0.40
0.38
0
500
1000
1500
2000
2500


Training Loss per Iteration (With vs Without Feature Scaling)
With Feature Scaling
Without Feature Scaling
Training Loss
Iteration
0.50
0.48
0.46
0.44
0.42
0.40
0.38
0
500
1000
1500
2000
2500



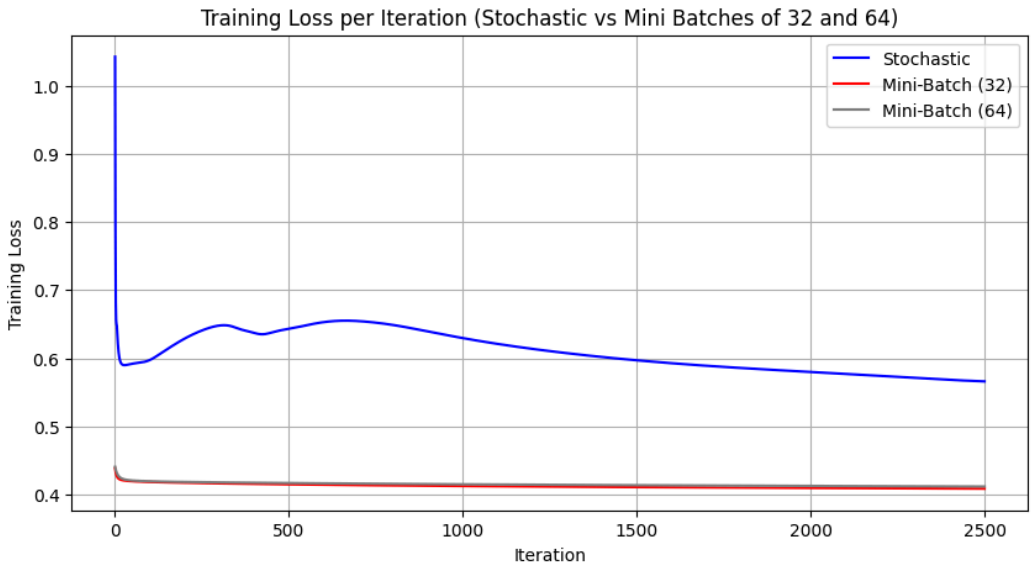
(c) **Confusion Matrix:**

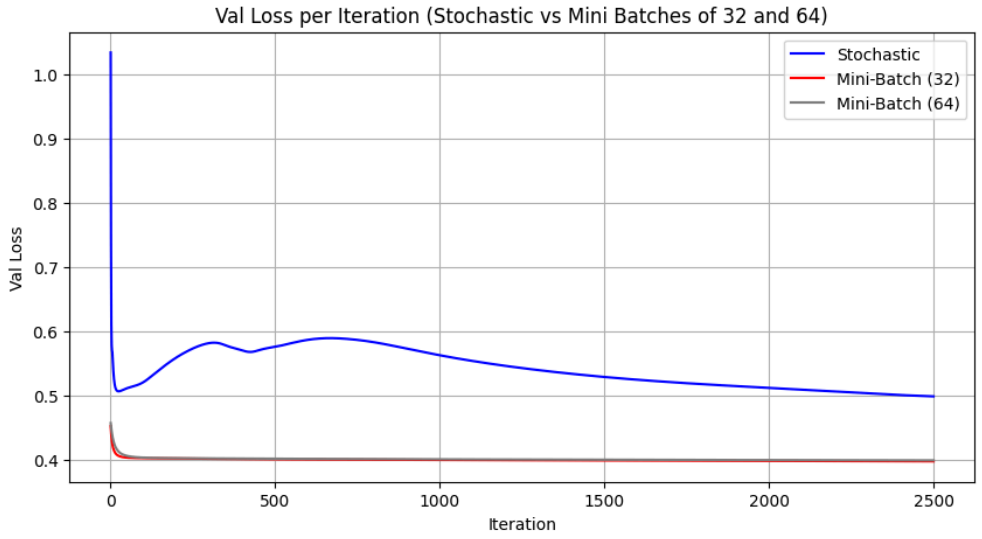
		Predicted Class	
		Heart Disease	No Heart Disease
Actual Class	Heart Disease	7	94
	No Heart Disease	5	515

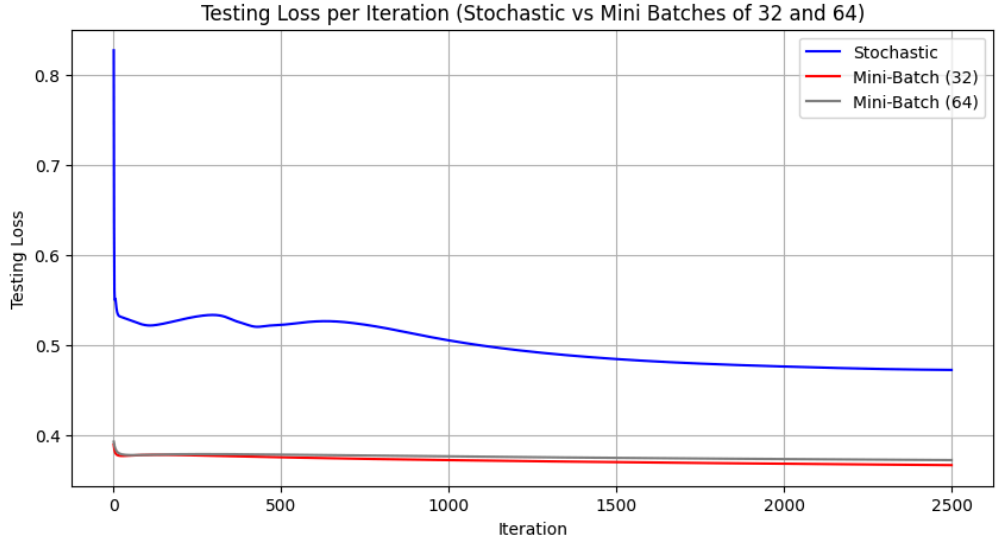
Recall -> 0.06930693069306931
Precision -> 0.5833333333333334
F1 Score -> 0.12389380530973453
ROC-AUC Score -> 0.5298457730388424

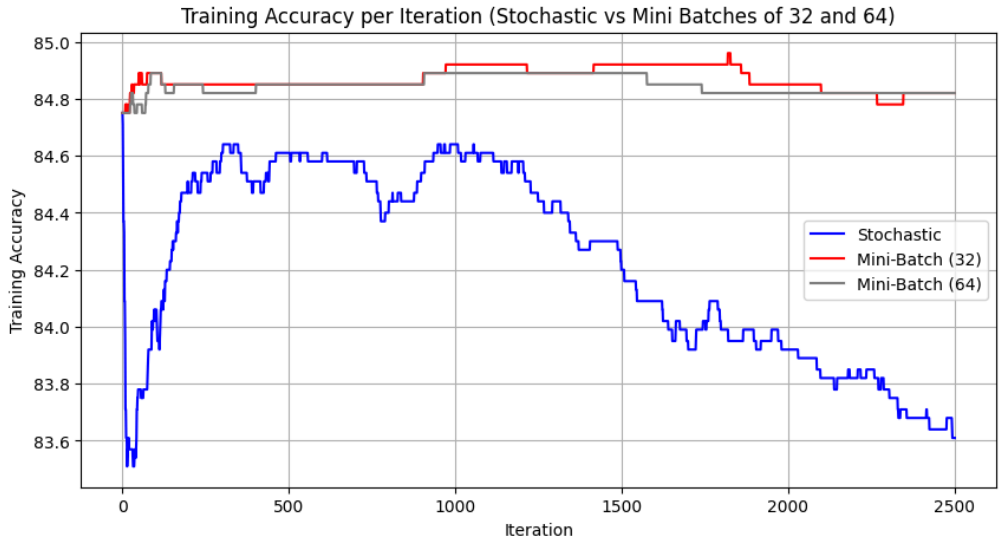
I am using the featured scaled model with learning rate 1 since it had better val loss. With it, the above metrics arise. This shows that there is a dramatic imbalance in the predictions, where the model predicts that the person doesn't have heart disease most of the time. So, the recall is very low, which is very bad for a model that's predicting heart disease(you don't want to tell a person with a heart disease that he doesn't have it but a false positive is comparatively better). The precision doesn't really matter in this case, because there are so few cases where heart disease is predicted, but it's still not great. The F1 Score and ROC-AUC Scores are horrible, which makes sense since the recall is really low, so overall it has really poor performance. The ROC-AUC scores should generally be above 0.9 to be a great model. Even though the accuracies were so high in the past diagrams, yet these scores emerge, indicating that the imbalance in labels has led to the model not being particularly great.

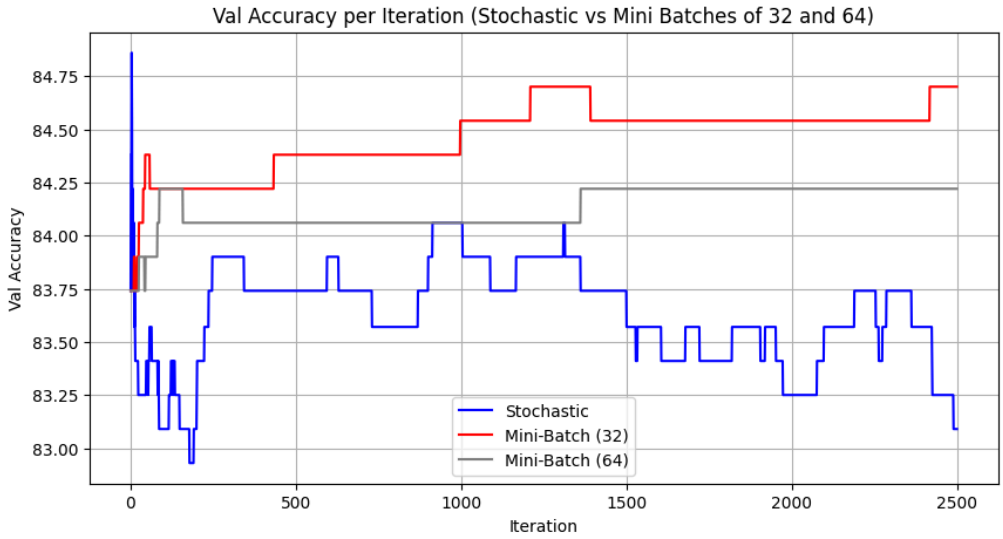
(d)

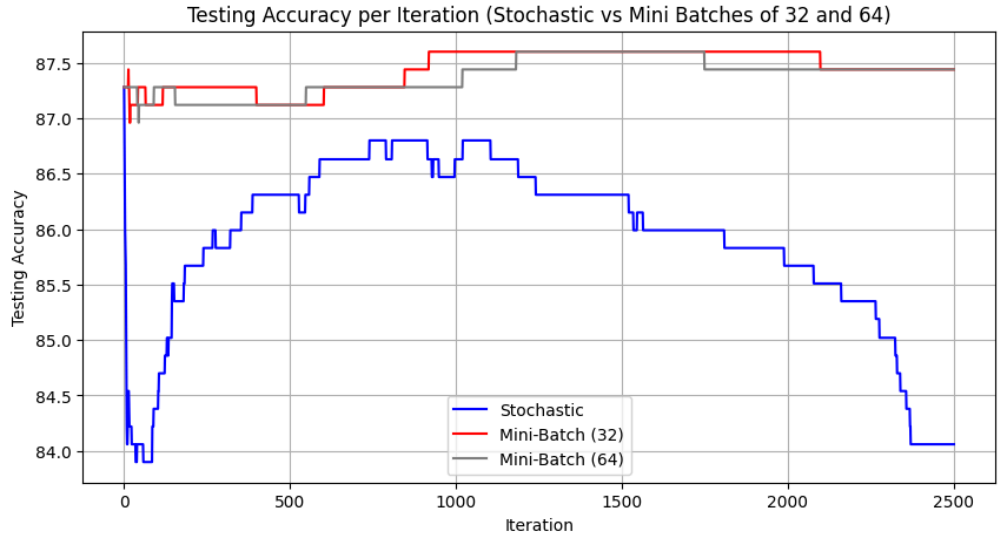

Training Loss per Iteration (Stochastic vs Mini Batches of 32 and 64)
Stochastic
Mini-Batch (32)
Mini-Batch (64)
Training Loss
Iteration
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
500
1000
1500
2000
2500


Val Loss per Iteration (Stochastic vs Mini Batches of 32 and 64)
Stochastic
Mini-Batch (32)
Mini-Batch (64)
Val Loss
Iteration
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
500
1000
1500
2000
2500


Testing Loss per Iteration (Stochastic vs Mini Batches of 32 and 64)
Stochastic
Mini-Batch (32)
Mini-Batch (64)
Testing Loss
Iteration
0.8
0.7
0.6
0.5
0.4
0
500
1000
1500
2000
2500


Training Accuracy per Iteration (Stochastic vs Mini Batches of 32 and 64)
Training Accuracy
Iteration
Stochastic
Mini-Batch (32)
Mini-Batch (64)


Val Accuracy per Iteration (Stochastic vs Mini Batches of 32 and 64)
Val Accuracy
Iteration
Stochastic
Mini-Batch (32)
Mini-Batch (64)


Testing Accuracy per Iteration (Stochastic vs Mini Batches of 32 and 64)
Testing Accuracy
Iteration
Stochastic
Mini-Batch (32)
Mini-Batch (64)

In terms of loss, Mini-Batch Gradient Descent with batchsize of 32 had much lower loss than that of batchsize of 64 and stochastic gradient descent. Both minibatches had similar loss, but stochastic gradient descent had much worse. Based on the graph, it looked like stochastic gradient descent entered a region of local minima which was much worse than other regions and thus the initial increase in loss, and then subsequent decrease. In terms of accuracy, mini-batch with batchsize of 32 was better for val accuracy while for test, they were very similar. But, stochastic gradient descent failed to have good accuracy.

Convergence Speed :- Stochastic gradient descent took a long time to converge at a learning rate of 0.0001. Whereas, the 2 mini-batch gradient descents were much faster at converging. Perhaps, with a much larger number of epochs, it would converge and be better than the mini-batches.

Stability :- Stochastic was also more unstable. The variation in accuracy and loss at times indicates a lack of stability even at such a low learning rate. Whereas the loss and accuracy of the mini-batch gradient descents were more consistent.

(e) Assuming that the question wanted us to test cross-validation on the model with batch gradient descent and nothing else.

```
Iteration 1 ->
      Recall -> 0.007936507936507936
      Precision -> 0.5
      Accuracy -> 84.78%
      F1 score -> 0.015625

Iteration 2 ->
      Recall -> 0.014925373134328358
      Precision -> 0.5
      Accuracy -> 83.82%
      F1 score -> 0.02898550724637681

Iteration 3 ->
      Recall -> 0.008064516129032258
      Precision -> 1.0
      Accuracy -> 85.14%
      F1 score -> 0.016

Iteration 4 ->
      Recall -> 0.017391304347826087
      Precision -> 0.5
      Accuracy -> 86.11%
      F1 score -> 0.03361344537815126

Iteration 5 ->
      Recall -> 0.016260162601626018
```

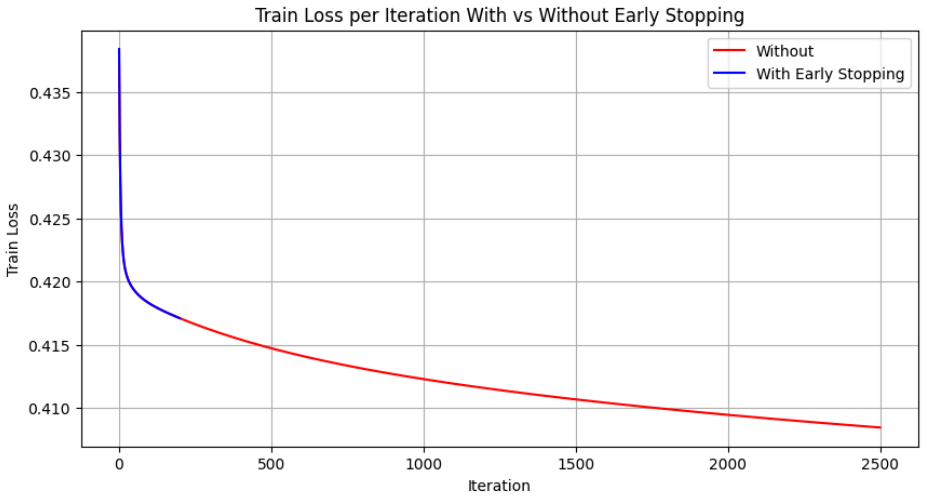
```
Precision -> 0.5
Accuracy -> 85.14%
F1 score -> 0.03149606299212599


Avg recall -> 0.012915572829864132
Std Dev of Recall -> 0.0040885593783878136
Variance of Recall -> 1.6716317790602946e-05

Avg precision -> 0.6
Std Dev of Precision -> 0.2
Variance of Precision -> 0.04000000000000001

Avg accuracy -> 84.998%
Std Dev of Accuracy -> 0.7364889680097068
Variance of Accuracy -> 0.5424160000000029

Avg F1 score -> 0.02514400312333081
Std Dev of F1 Score -> 0.00775965829393719
Variance of F1 Score -> 6.0212296838668214e-05
```

The Average Recall, F1 score are really bad. The precision is average and the accuracy is rather decent, but that is mainly because of a imbalance in classes in the dataset. There is a much larger number of cases which are not ‘heart disease’ so, you can see why the average is high, but it leads to the model not really grasping the balance between ‘non-heart disease’ and ‘heart disease’.

Across the folds, the accuracy and precision vary a bit, but not by much, and so, overall, the model is rather stable, and there’s not much variance in the performance of the model.

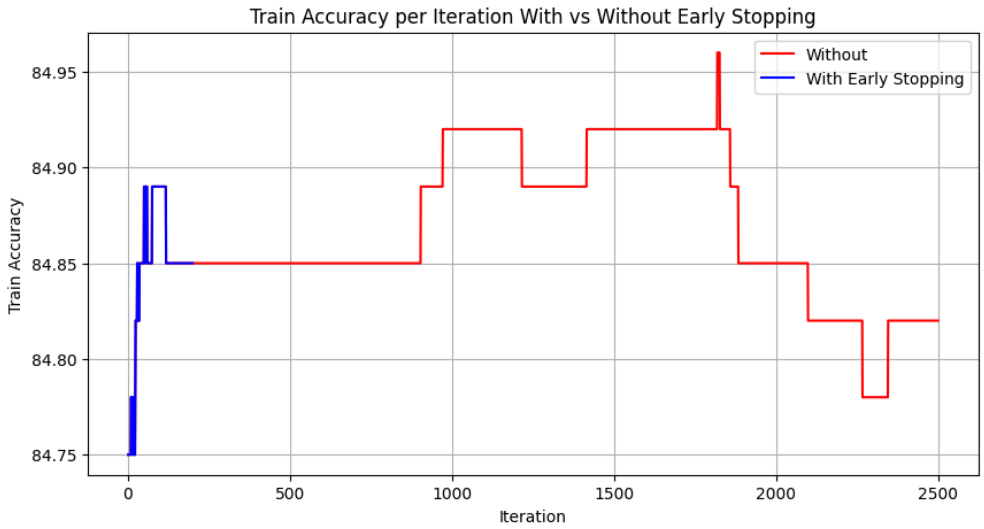
(f) Best Method Observed so far is Mini Batch with size 32. I am not doing feature scaling with it, because not mentioned in question to combine the various optimizations. The reason I use Mini Batch is because while it has worse loss, the F1 score is better and it does slighly better at predicting both heart disease and not heart disease, and not just 'not heart disease'.

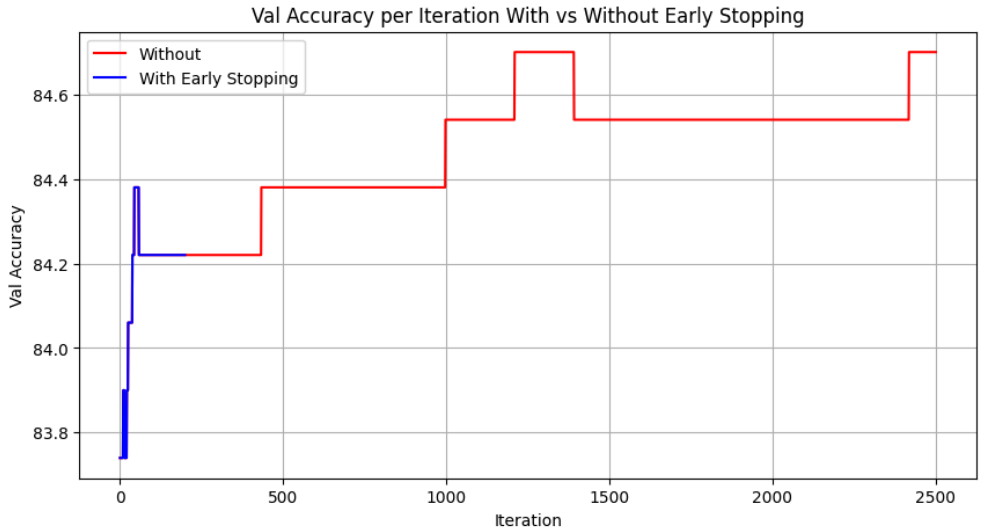
My Stopping criteria is when the val loss doesn’t decrease by a certain amount, then I stop the gradient descent.

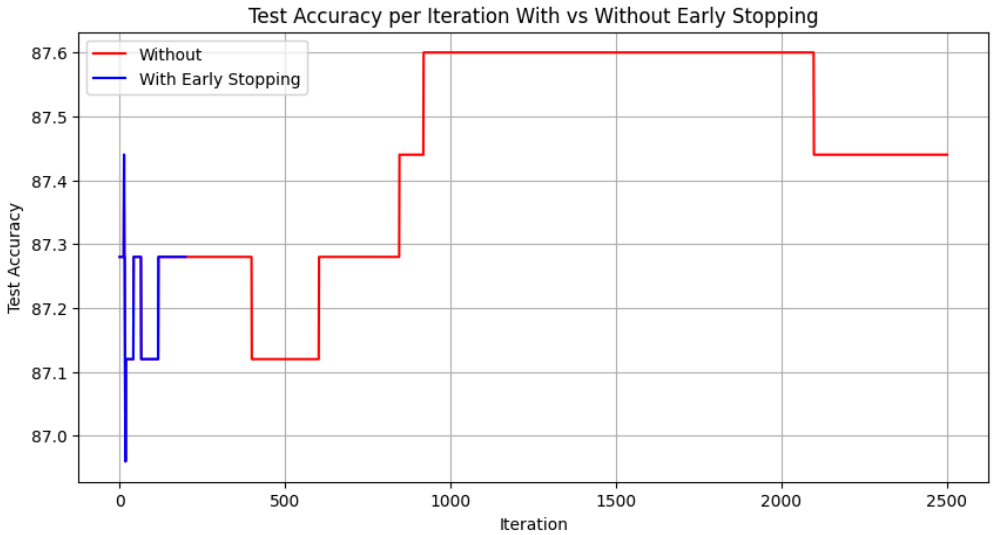

Train Loss per Iteration With vs Without Early Stopping
Without
With Early Stopping
Train Loss
0.435
0.430
0.425
0.420
0.415
0.410
0
500
1000
1500
2000
2500
Iteration


Val Loss per Iteration With vs Without Early Stopping
Without
With Early Stopping
Val Loss
0.45
0.44
0.43
0.42
0.41
0.40
0
500
1000
1500
2000
2500
Iteration

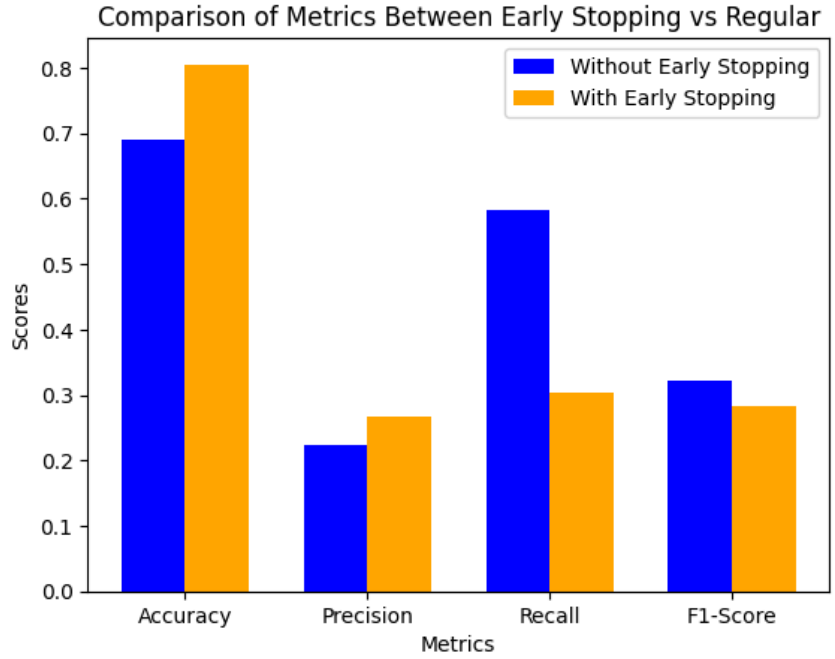

Test Loss per Iteration With vs Without Early Stopping
Without
With Early Stopping
Test Loss
0.390
0.385
0.380
0.375
0.370
0
500
1000
1500
2000
2500
Iteration


Train Accuracy per Iteration With vs Without Early Stopping
Without
With Early Stopping
Train Accuracy
84.95
84.90
84.85
84.80
84.75
0
500
1000
1500
2000
2500
Iteration


Val Accuracy per Iteration With vs Without Early Stopping
Without
With Early Stopping
Val Accuracy
84.6
84.4
84.2
84.0
83.8
0
500
1000
1500
2000
2500
Iteration


Test Accuracy per Iteration With vs Without Early Stopping
Without
With Early Stopping
Test Accuracy
87.6
87.5
87.4
87.3
87.2
87.1
87.0
0
500
1000
1500
2000
2500
Iteration

Based on the test loss, it feels like the early stopping was done too early. The accuracy was better but more importantly, the test loss was decreasing quite a lot even after stopping. There is not much overfitting at all, as we don't see an increase in val or test loss. So, it isn't memorizing the train data and not generalizing at all.



Confusion Matrix without Early Stopping:

		Predicted Class	
		Heart Disease	No Heart Disease
Actual Class	Heart Disease	46	33
	No Heart Disease	160	382

Confusion Matrix with Early Stopping:

		Predicted Class	
		Heart Disease	No Heart Disease
Actual Class	Heart Disease	24	55
	No Heart Disease	66	476

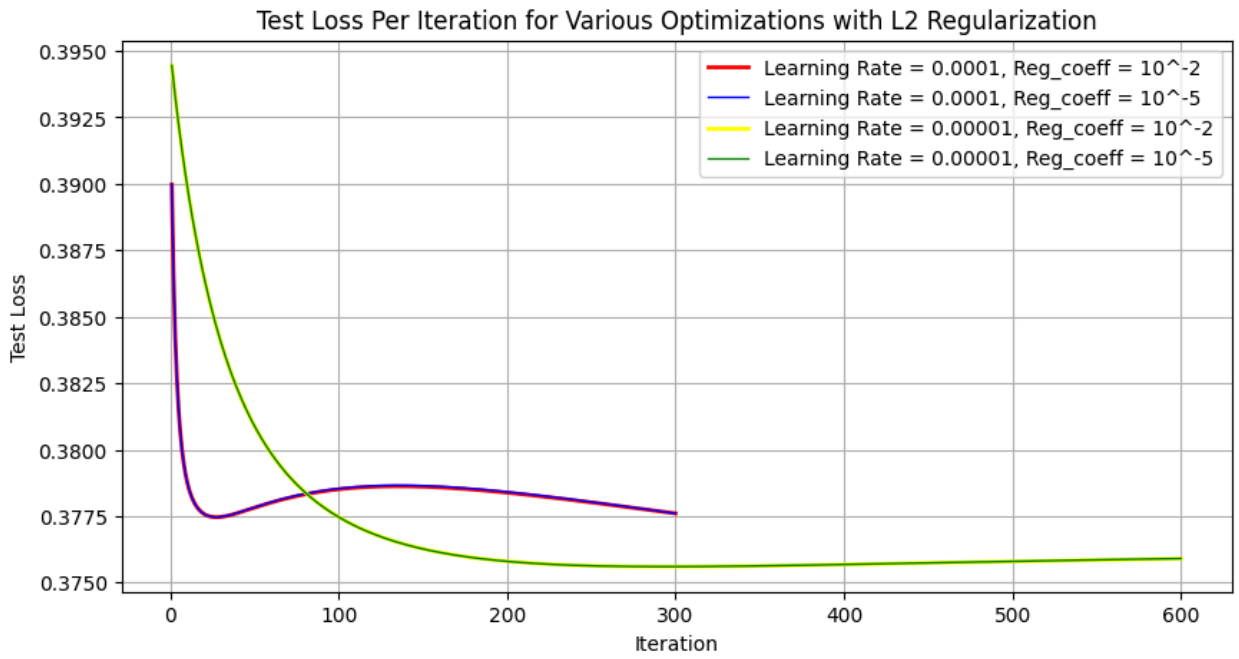
Accuracy Without -> 0.6892109500805152. With -> 0.8051529790660226
Precision Without -> 0.22330097087378642. With -> 0.26666666666666666
Recall Without -> 0.5822784810126582. With -> 0.3037974683544304
F1 Score Without -> 0.3228070175438597. With -> 0.28402366863905326

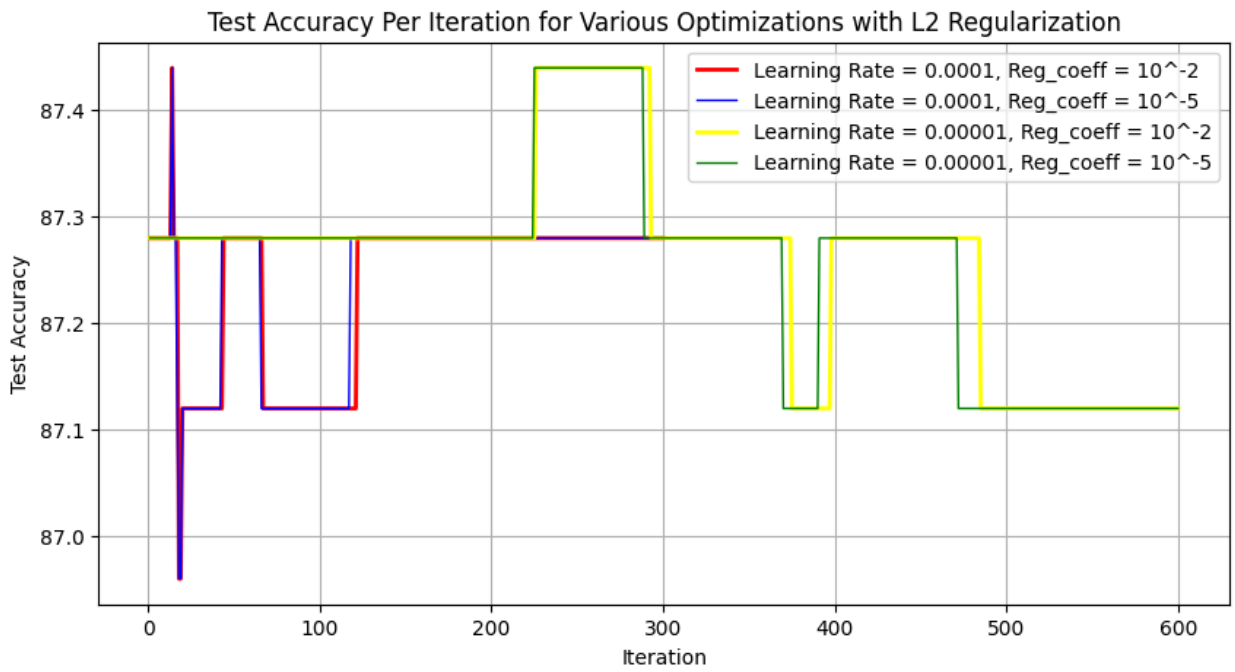
The above are scores on the test set, with early stopping. Clearly this shows the difference between early stopping and without. With more iterations, it trains itself to predict that it's not a

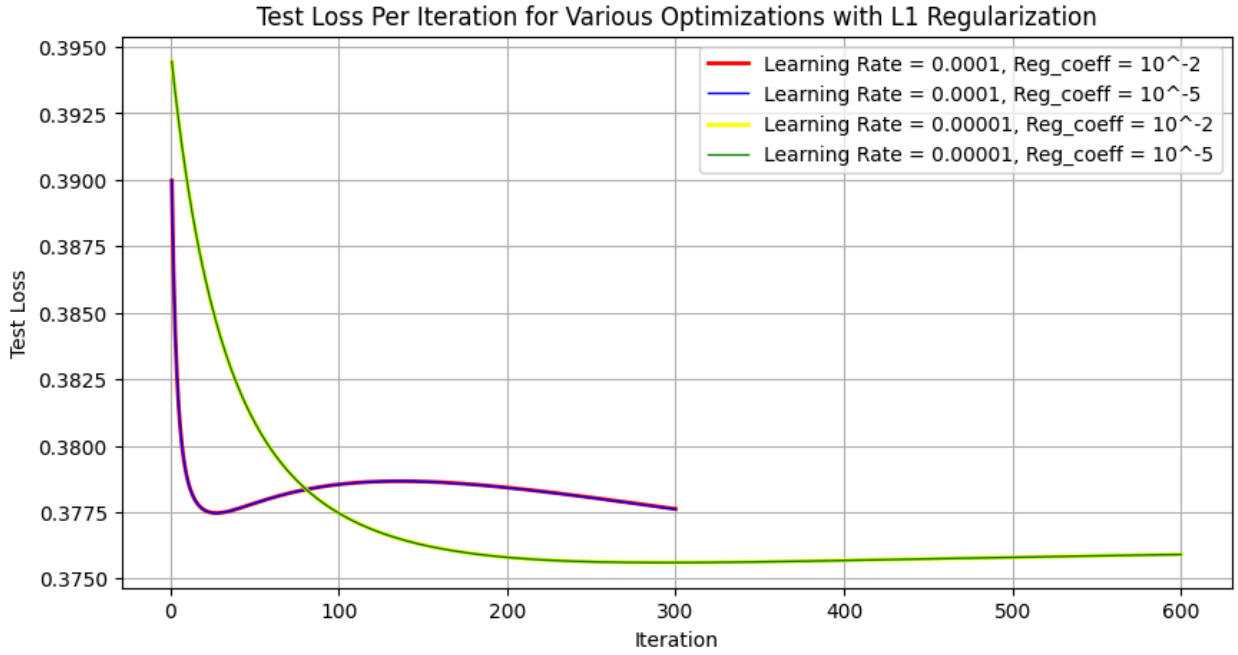
heart disease regardless of the person's condition, which leads to greater accuracy because of the label imbalance. But it leads to a worse recall, worse f1 score and just a bit better precision. So, it generalizes better though the label imbalance still is the main thing that needs to be resolved.

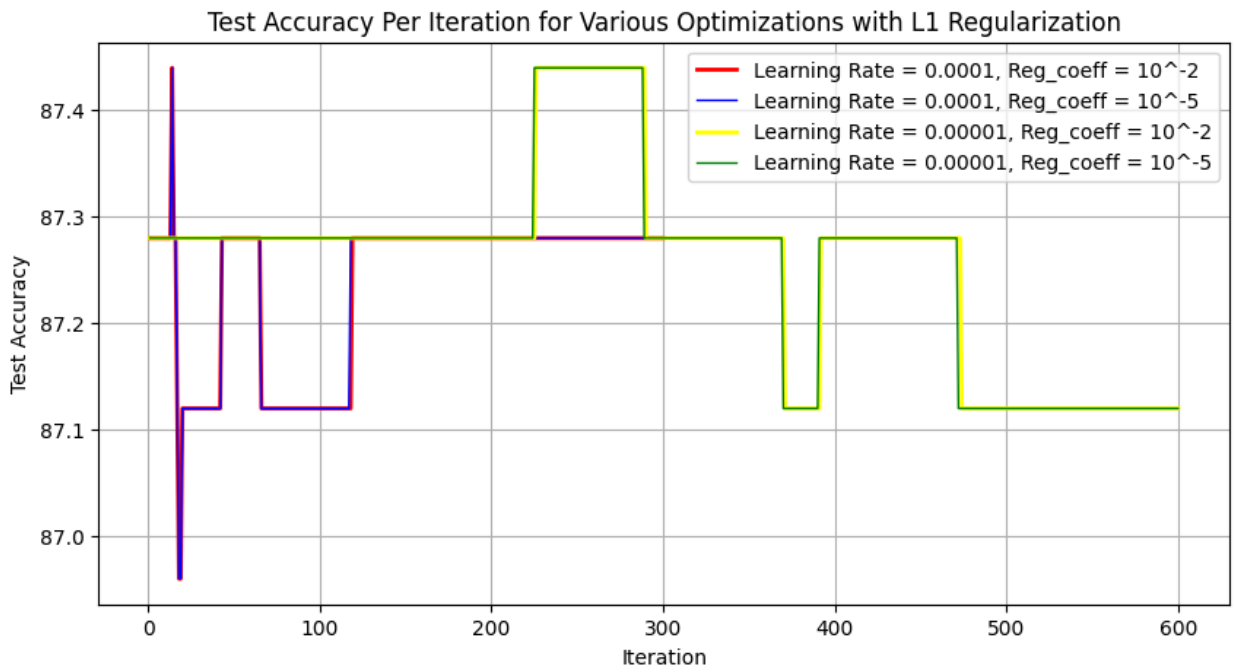
Experimenting with Learning Rates and Regularization (Using early stopping as well since no info in question):-

Learning Rates - 0.0001 and 0.00001 (Anything higher and it oscillates too much)
Regularization - 0.0001 and 0.00001




Test Accuracy Per Iteration for Various Optimizations with L2 Regularization
Learning Rate = 0.0001, Reg_coeff = 10^-2
Learning Rate = 0.0001, Reg_coeff = 10^-5
Learning Rate = 0.00001, Reg_coeff = 10^-2
Learning Rate = 0.00001, Reg_coeff = 10^-5
Test Accuracy
87.4
87.3
87.2
87.1
87.0
0
100
200
300
400
500
600
Iteration

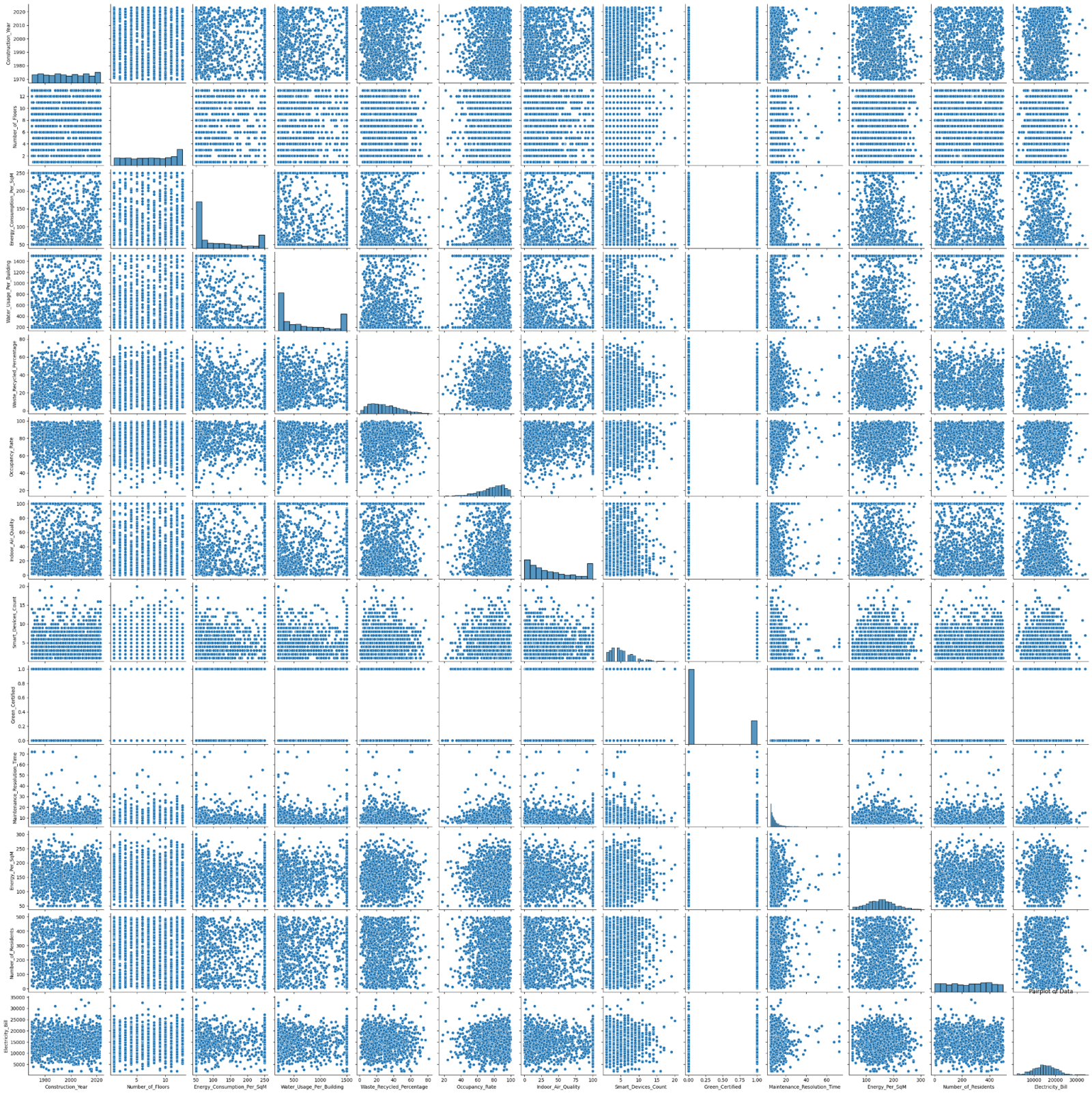

Test Loss Per Iteration for Various Optimizations with L1 Regularization
Learning Rate = 0.0001, Reg_coeff = 10^-2
Learning Rate = 0.0001, Reg_coeff = 10^-5
Learning Rate = 0.00001, Reg_coeff = 10^-2
Learning Rate = 0.00001, Reg_coeff = 10^-5
Test Loss
0.3950
0.3925
0.3900
0.3875
0.3850
0.3825
0.3800
0.3775
0.3750
0
100
200
300
400
500
600
Iteration



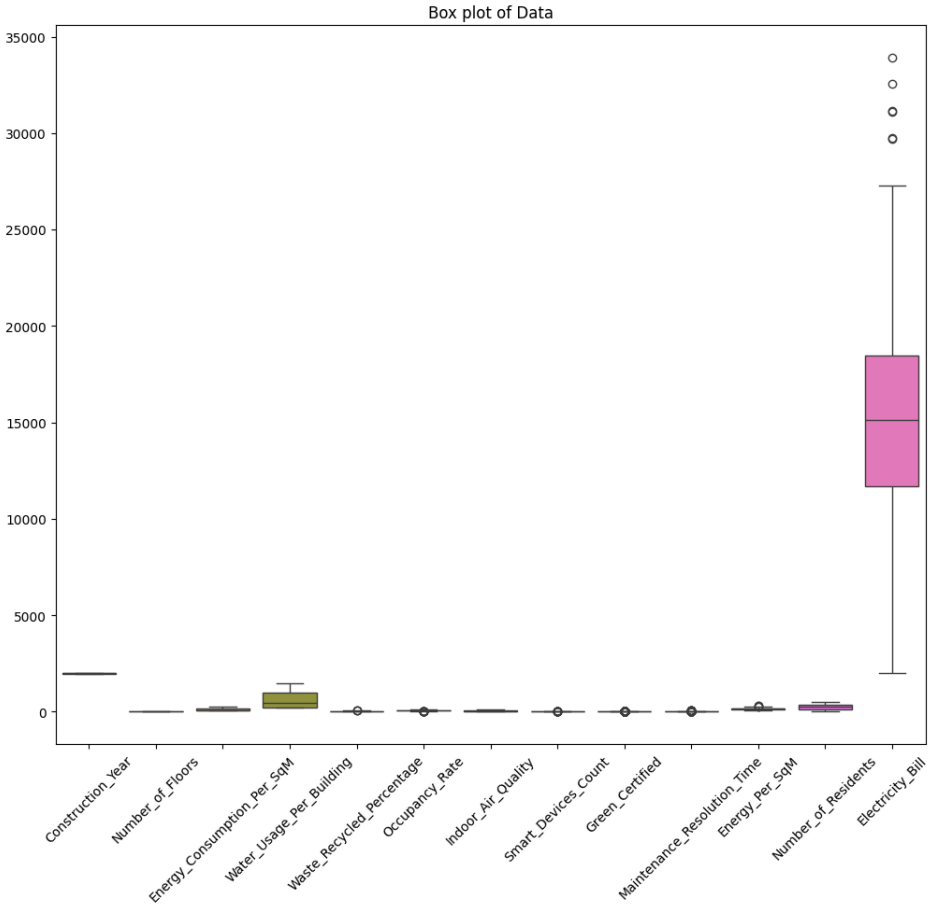
In general, the regularization coefficient doesn't change much. But a smaller learning rate has led to quicker convergence, which generally doesn't happen so perhaps it found the right local minima initially whereas with a larger learning rate, it entered the wrong one. The accuracies too are very similar. If I went any higher with learning rate, it oscillates.

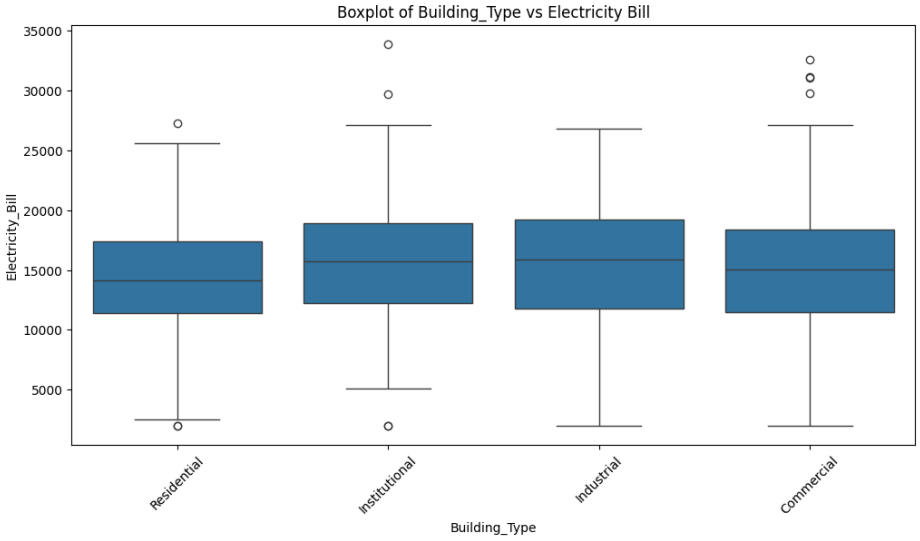
Section C

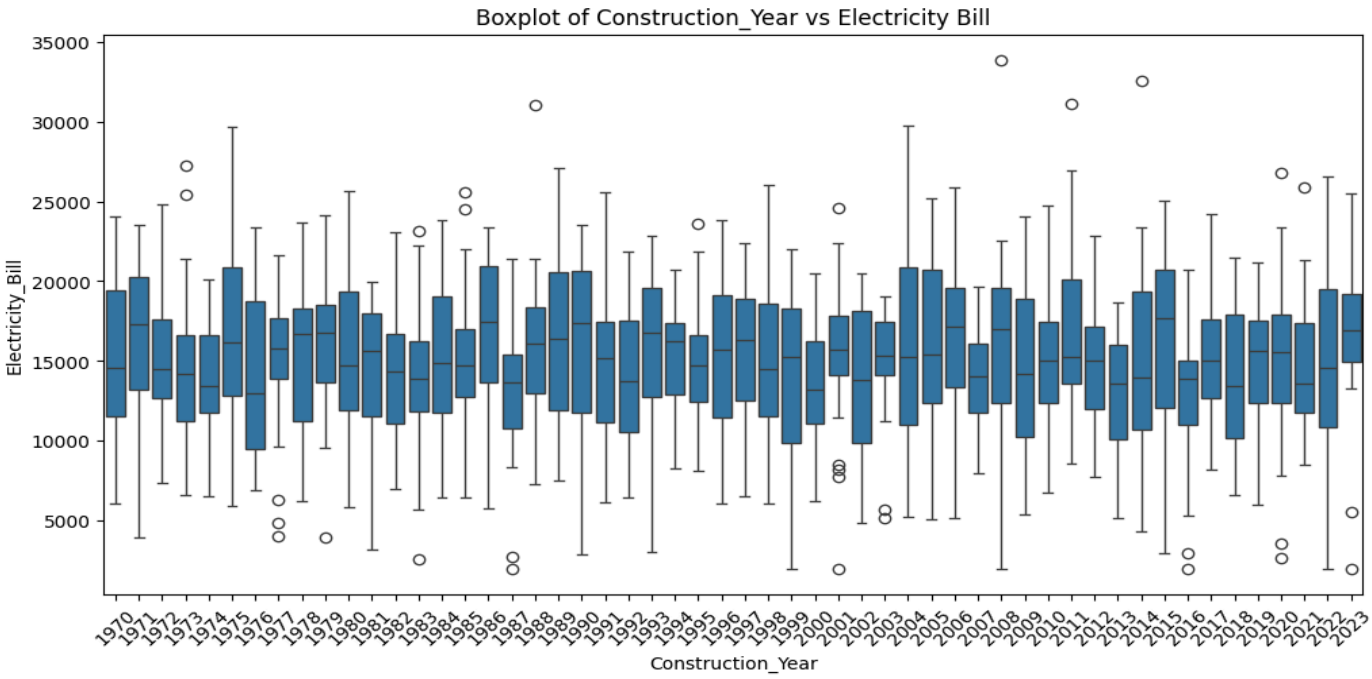
(a) Pair Plot:-

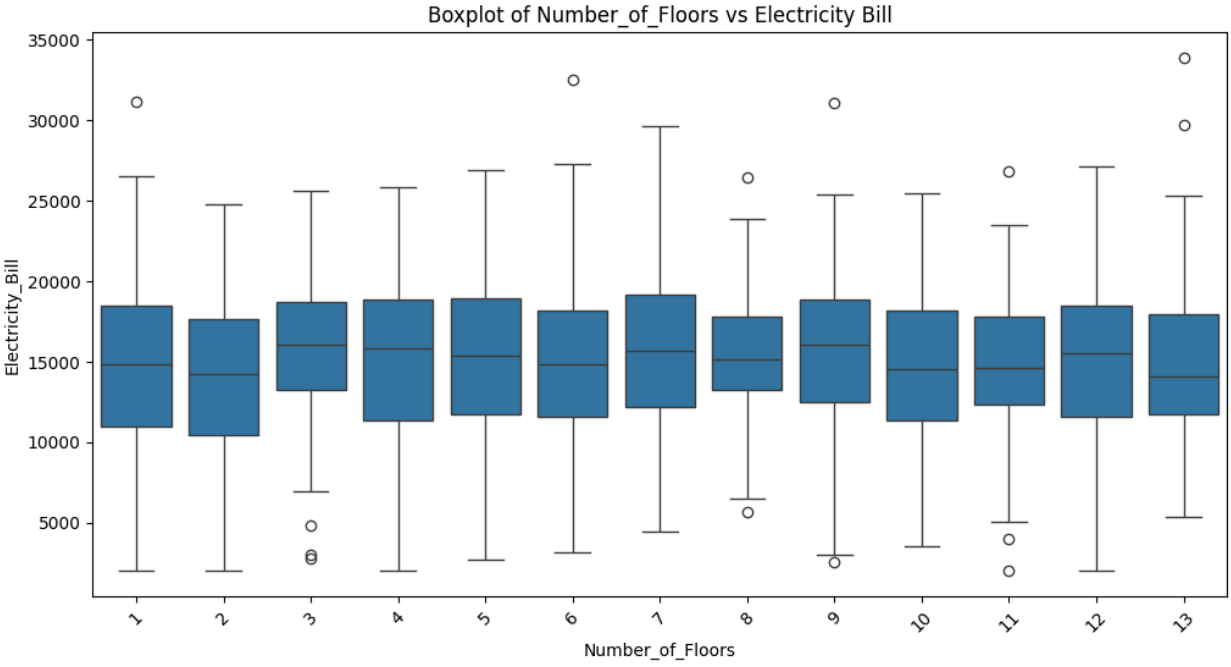


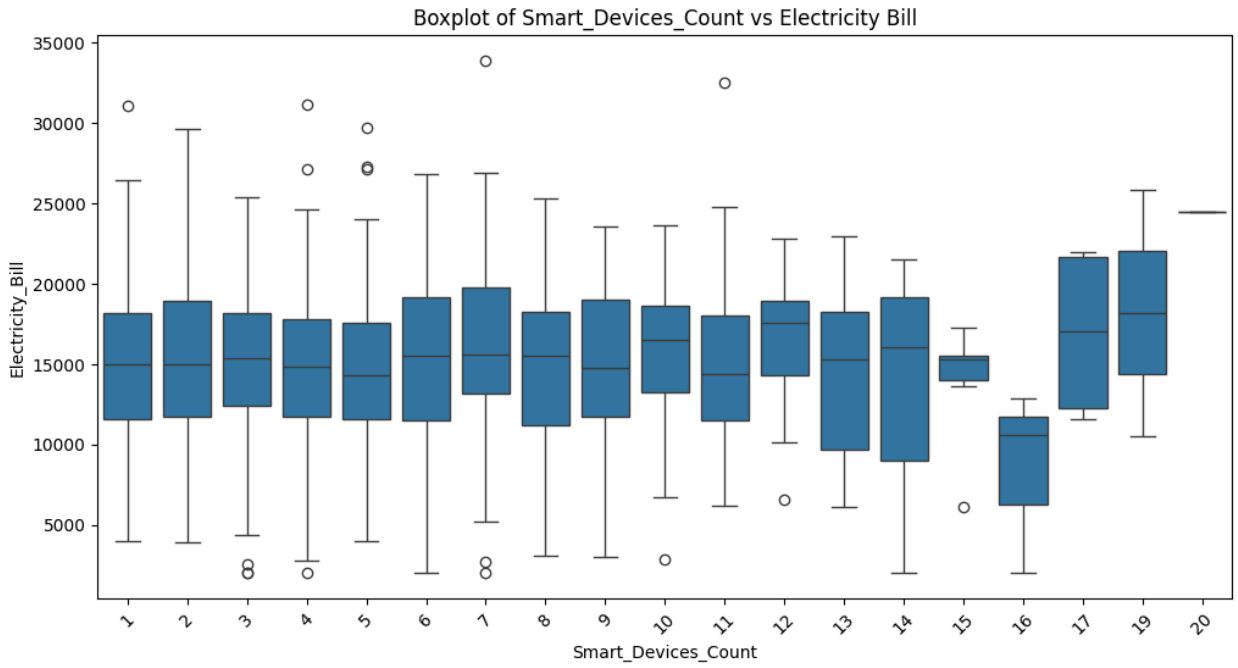
Link to full image:- https://imgur.com/a/350aXrX

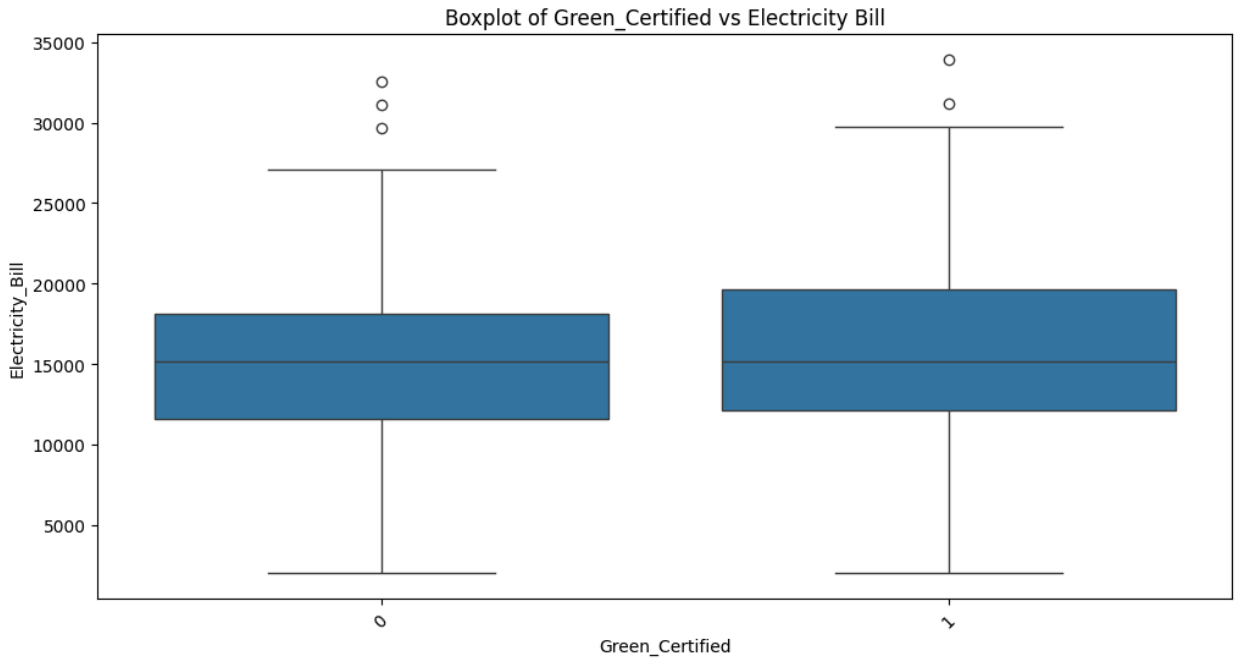

Box plot of Data
35000
30000
25000
20000
15000
10000
5000
0
Construction_Year
Number_of_Floors
Energy_Consumption_Per_SqM
Water_Usage_Per_Building
Waste_Recycled_Percentage
Occupancy_Rate
Indoor_Air_Quality
Smart_Devices_Count
Green_Certified
Maintenance_Resolution_Time
Energy_Per_SqM
Number_of_Residents
Electricity_Bill

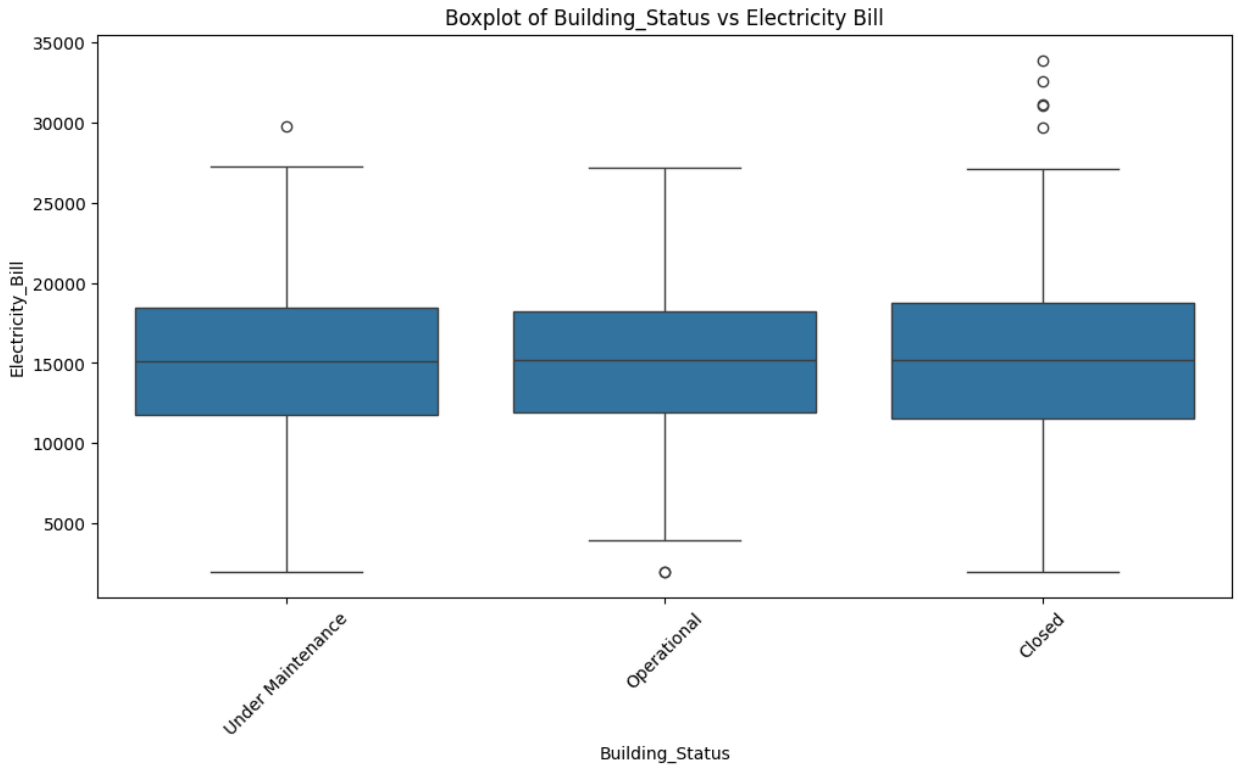

Boxplot of Building_Type vs Electricity Bill
Electricity_Bill
35000
30000
25000
20000
15000
10000
5000
Residential
Institutional
Industrial
Commercial
Building_Type

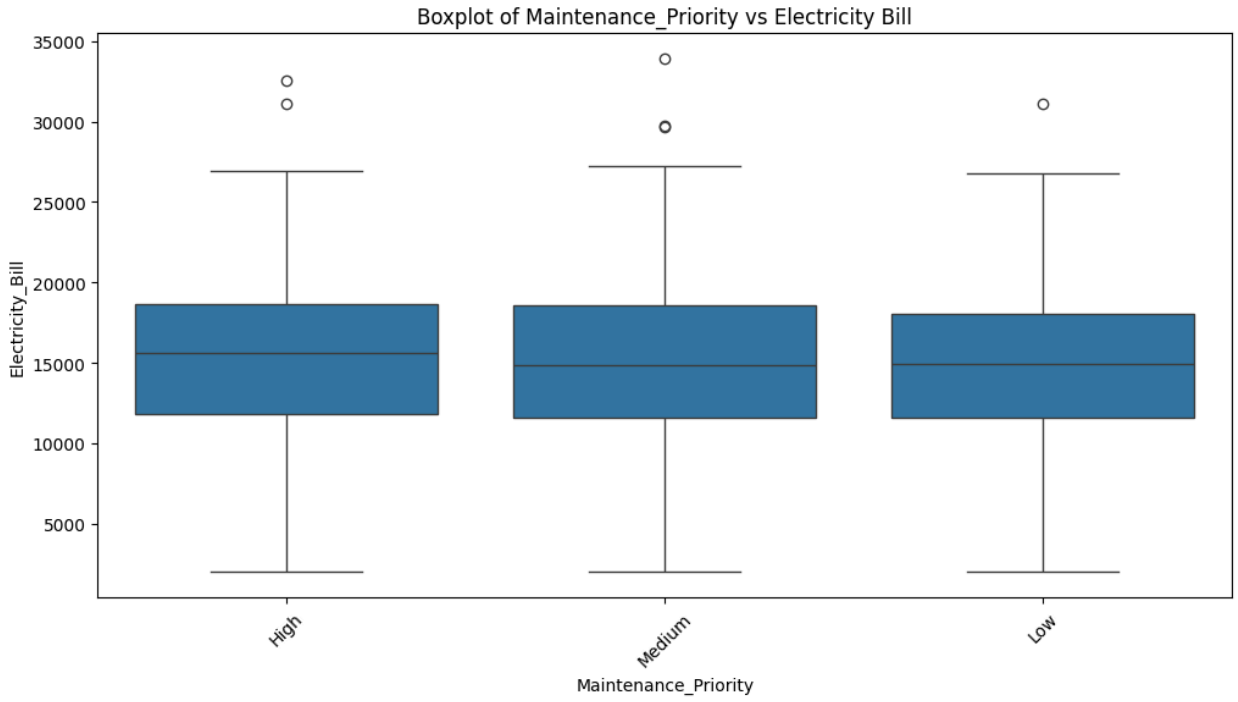

Boxplot of Construction_Year vs Electricity Bill
Electricity_Bill
Construction_Year

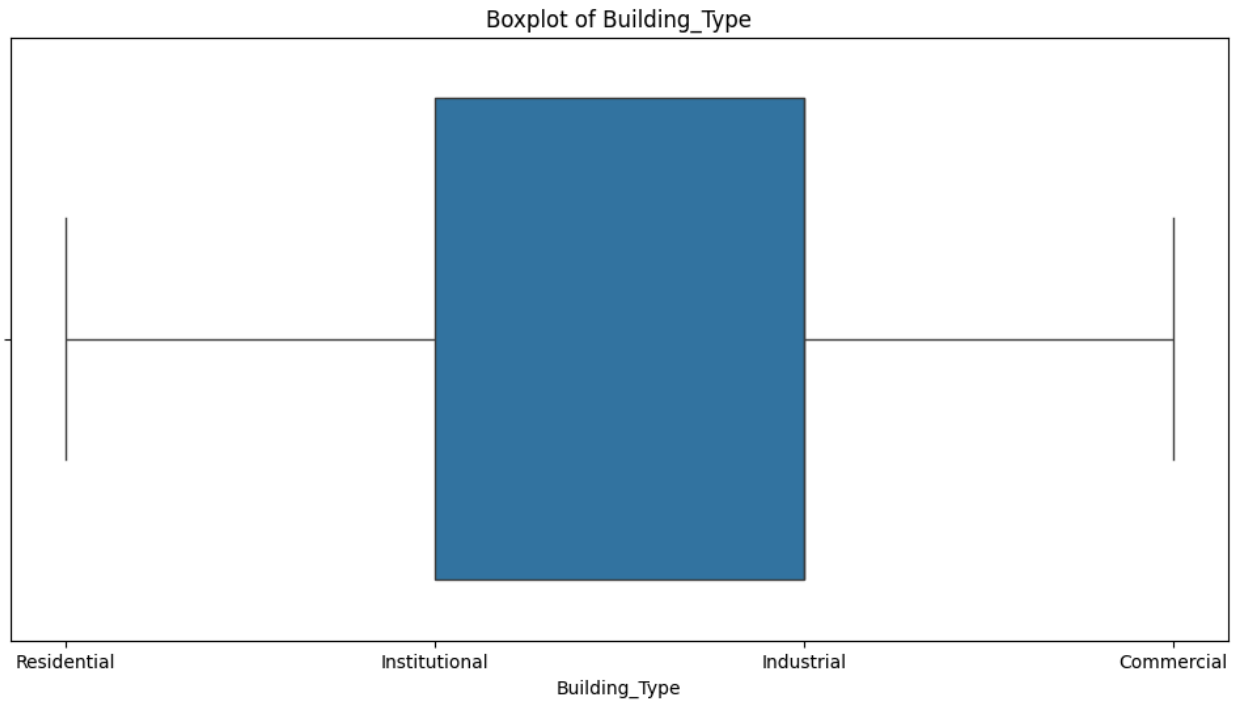

Boxplot of Number_of_Floors vs Electricity Bill
Electricity_Bill
Number_of_Floors

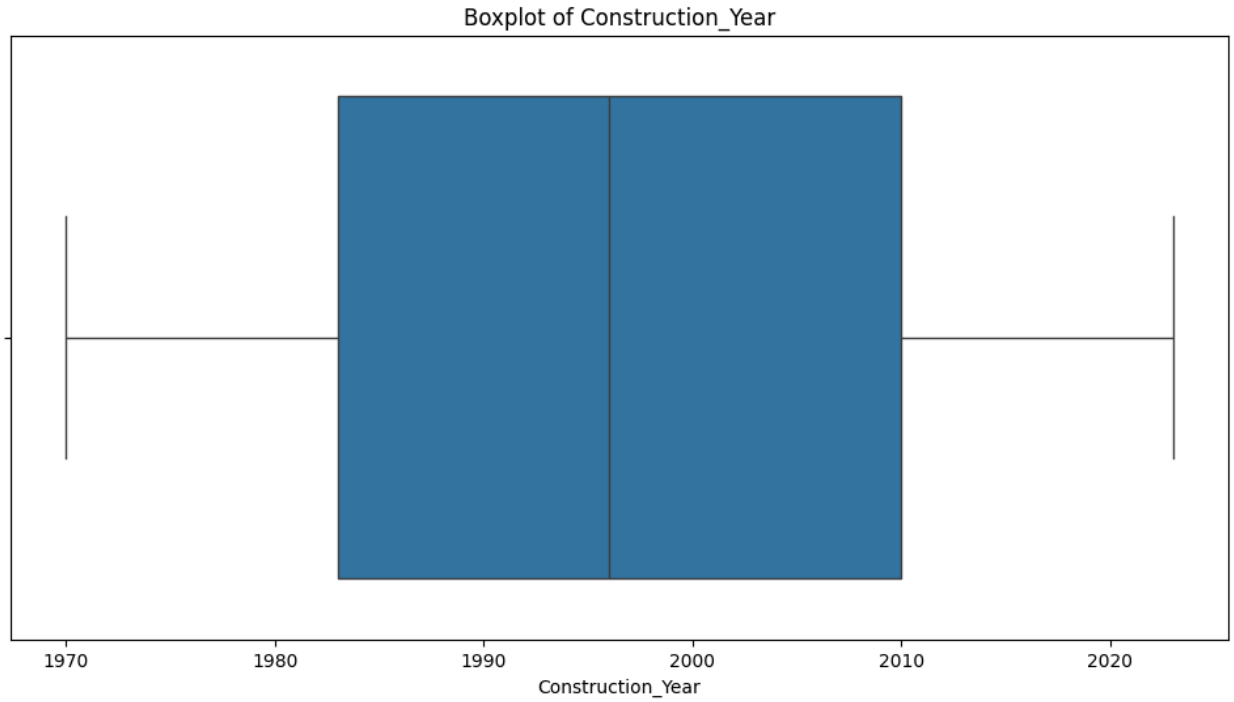

Boxplot of Smart_Devices_Count vs Electricity Bill
Electricity_Bill
35000
30000
25000
20000
15000
10000
5000
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
19
20
Smart_Devices_Count

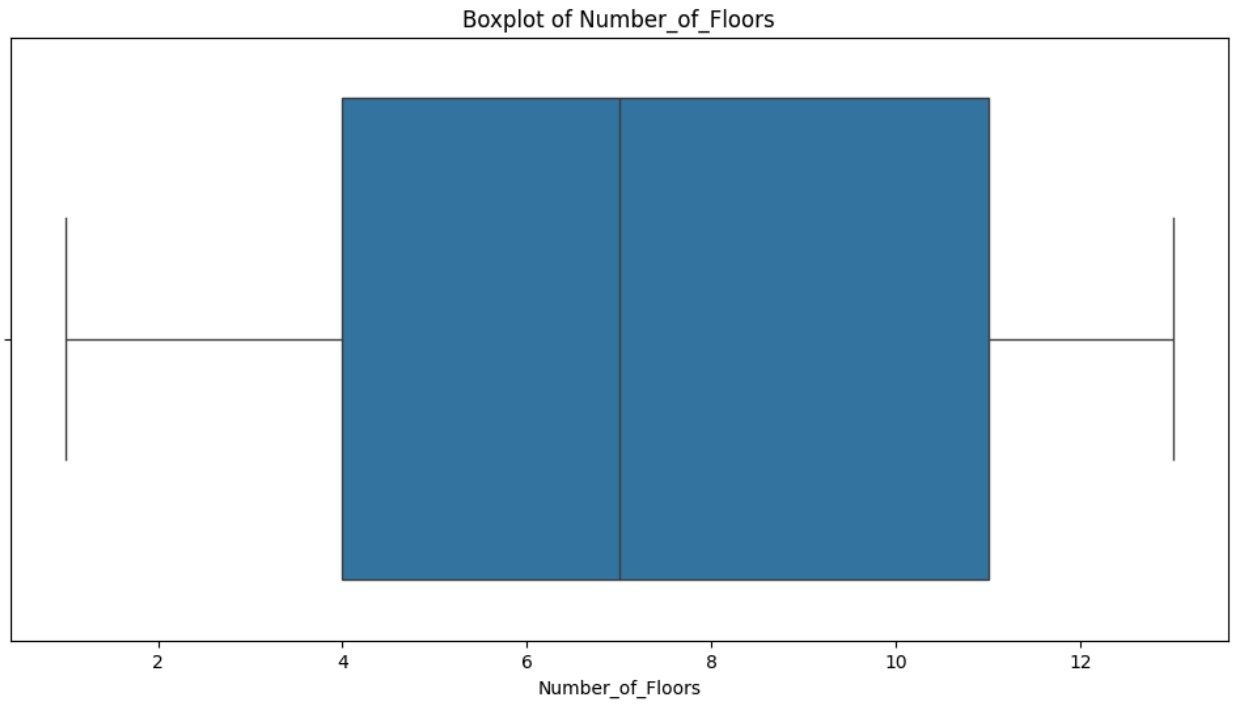

Boxplot of Green_Certified vs Electricity Bill
Electricity_Bill
35000
30000
25000
20000
15000
10000
5000
0
1
Green_Certified

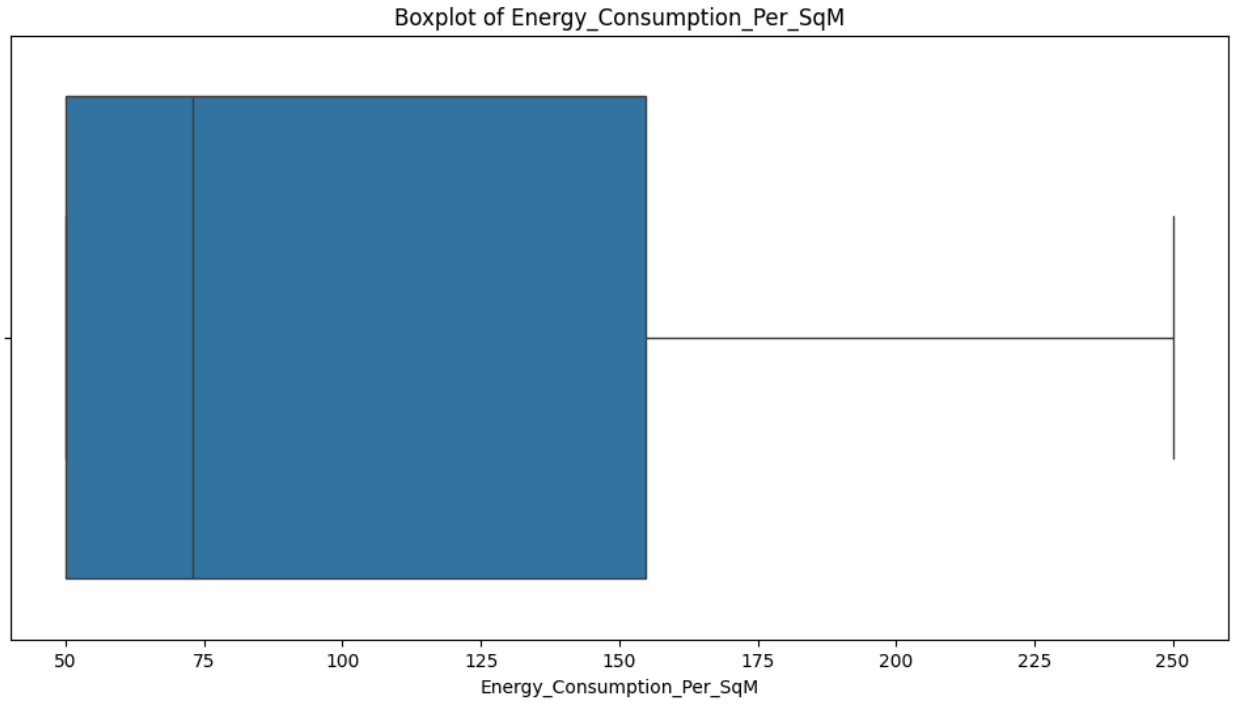

Boxplot of Building_Status vs Electricity Bill
Electricity_Bill
35000
30000
25000
20000
15000
10000
5000
Under Maintenance
Operational
Closed
Building_Status

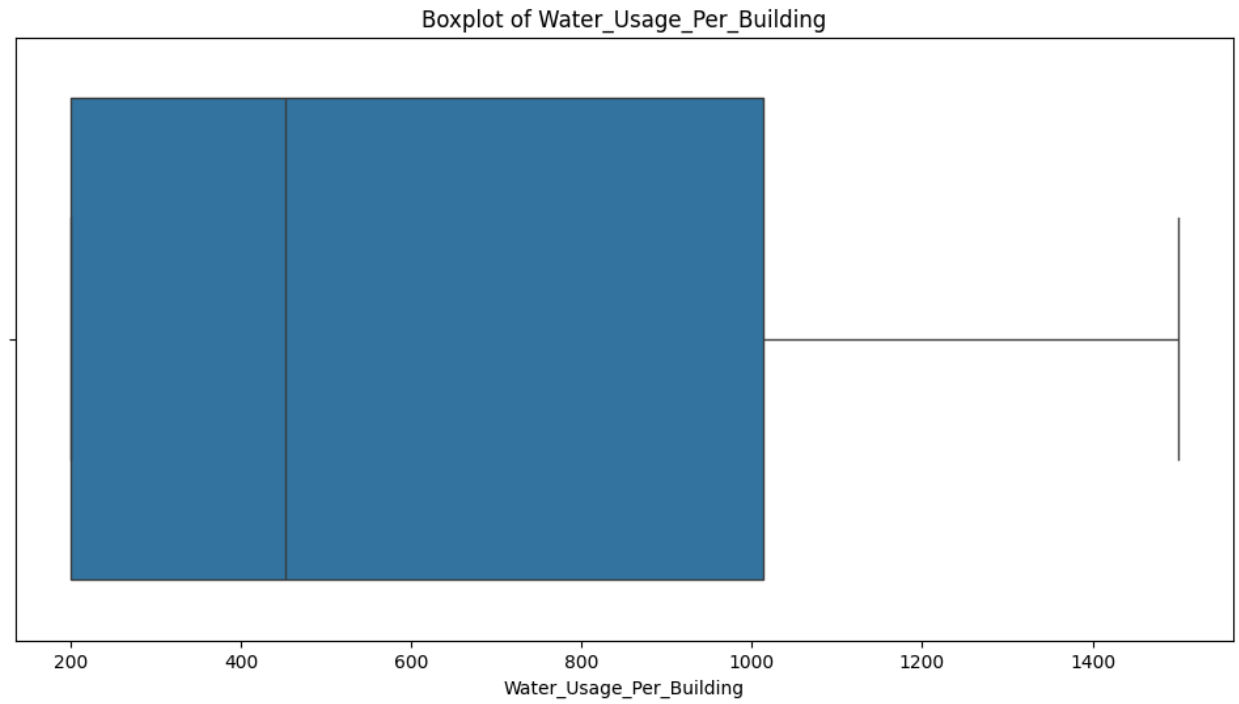

Boxplot of Maintenance_Priority vs Electricity Bill
Electricity_Bill
35000
30000
25000
20000
15000
10000
5000
High
Medium
Low
Maintenance_Priority

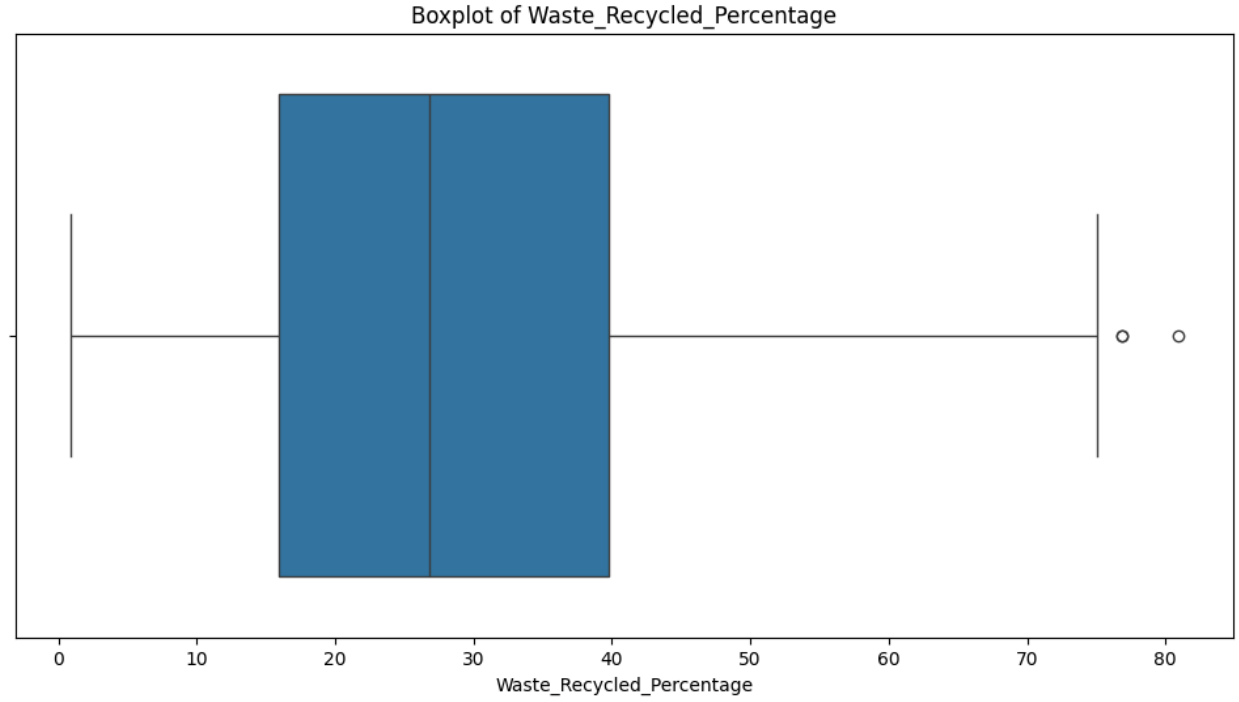

Boxplot of Building_Type
Residential
Institutional
Industrial
Commercial
Building_Type

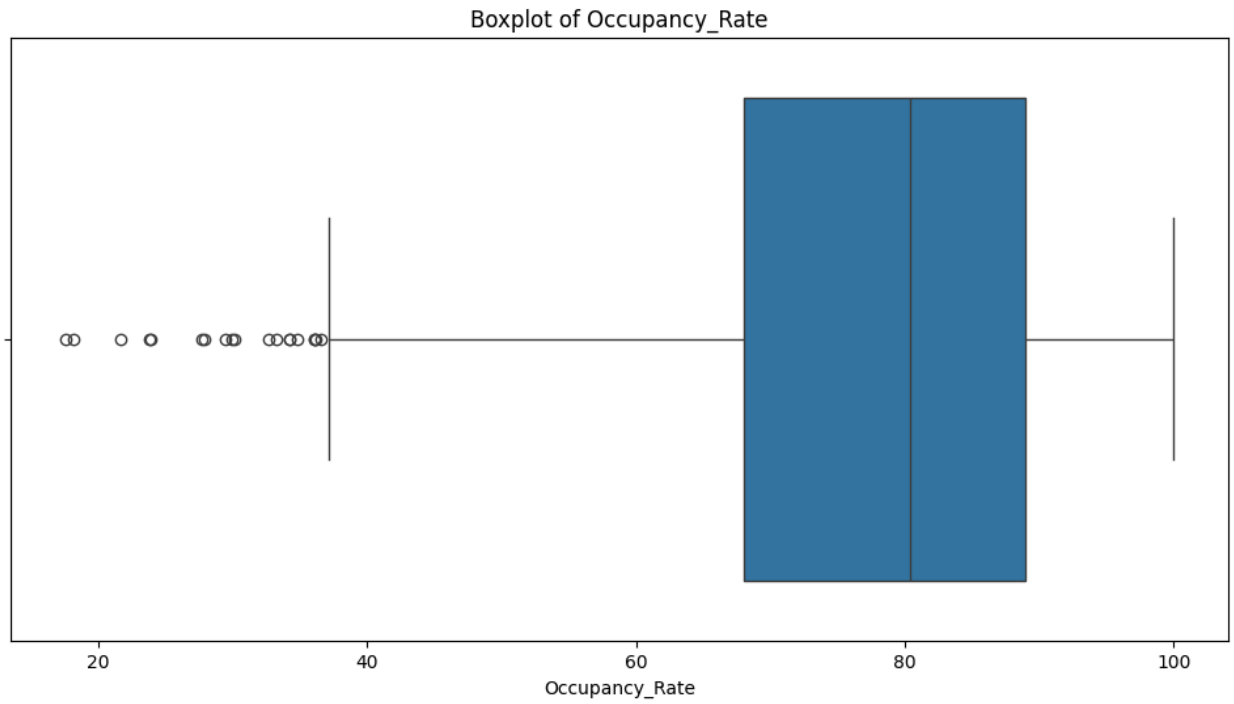

Boxplot of Construction_Year
1970
1980
1990
2000
2010
2020
Construction_Year

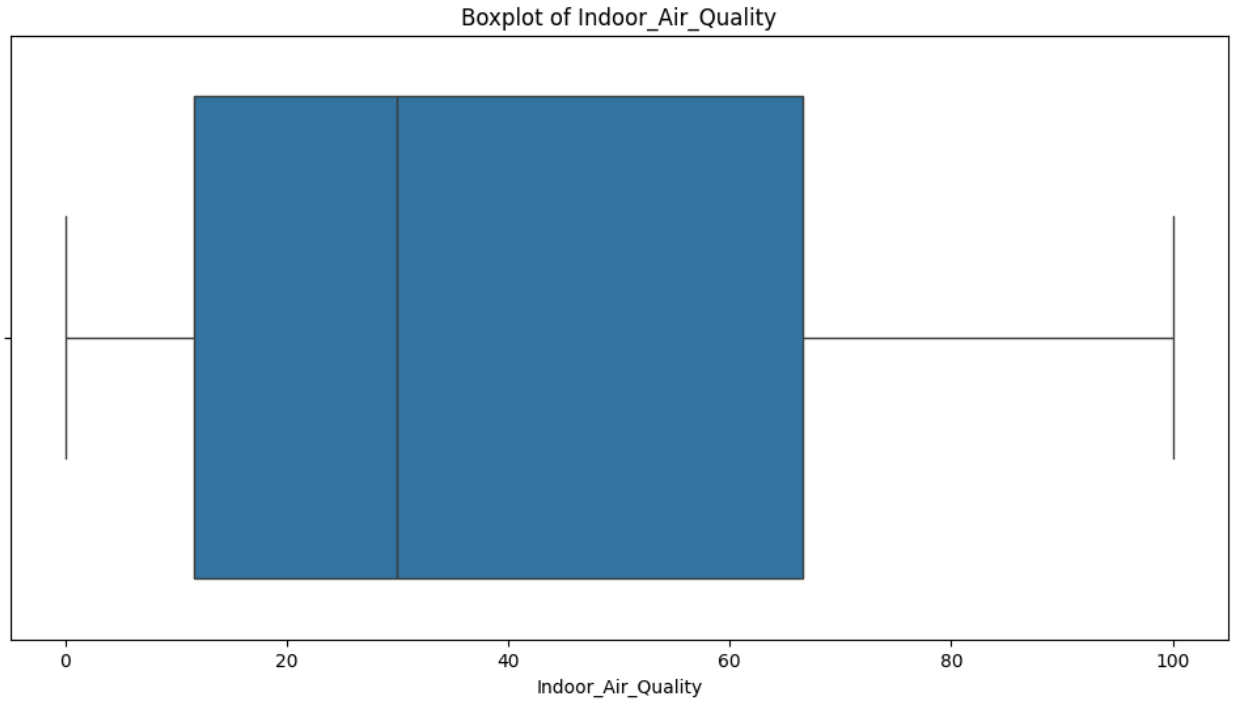

Boxplot of Number_of_Floors
2
4
6
8
10
12
Number_of_Floors

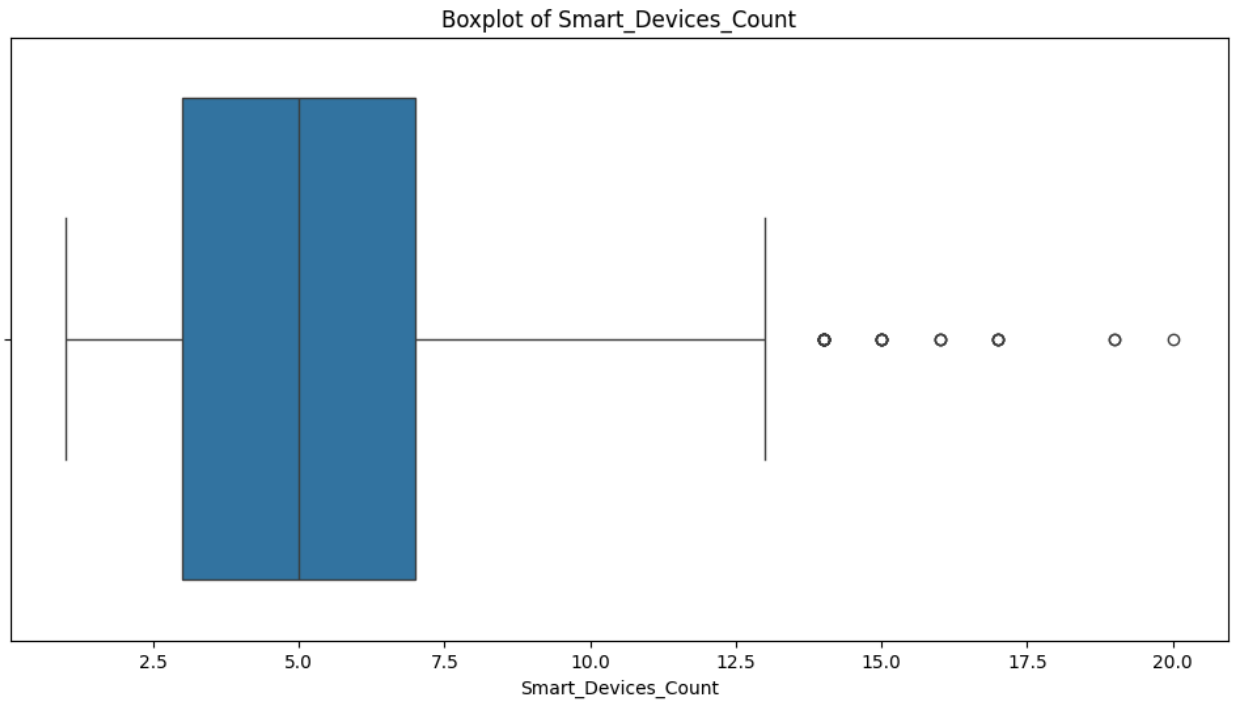

Boxplot of Energy_Consumption_Per_SqM
50
75
100
125
150
175
200
225
250
Energy_Consumption_Per_SqM

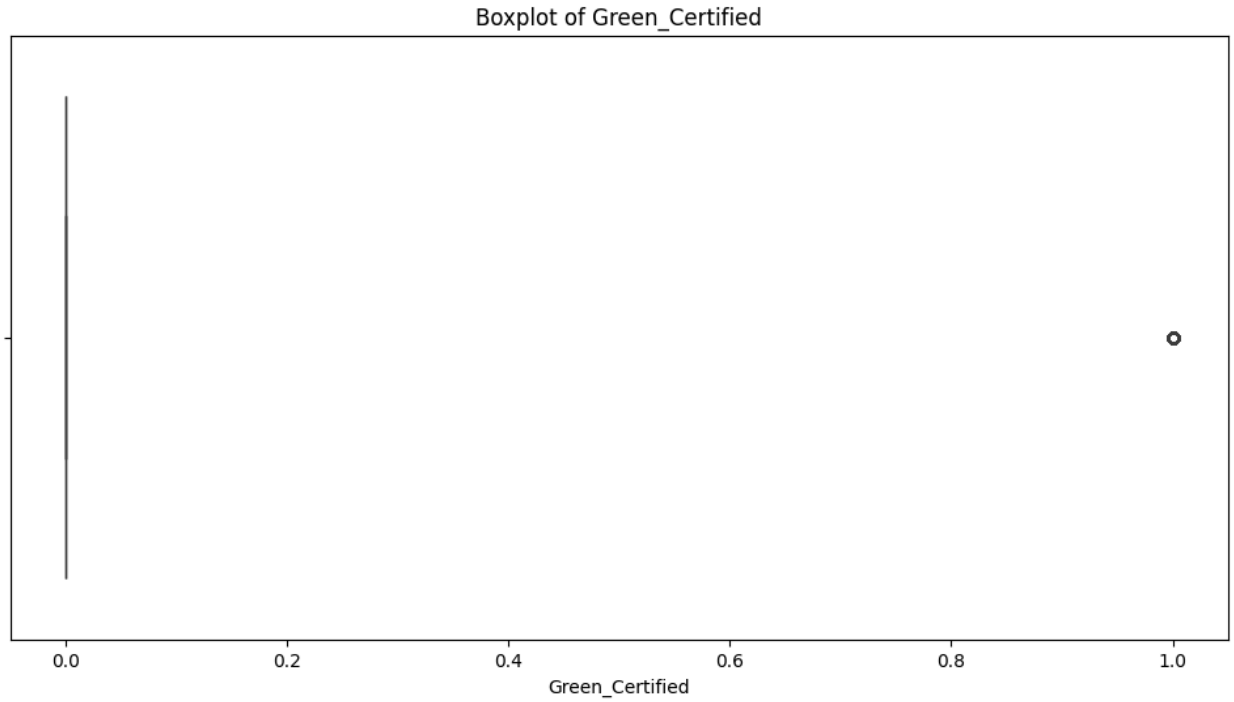

Boxplot of Water_Usage_Per_Building
200
400
600
800
1000
1200
1400
Water_Usage_Per_Building

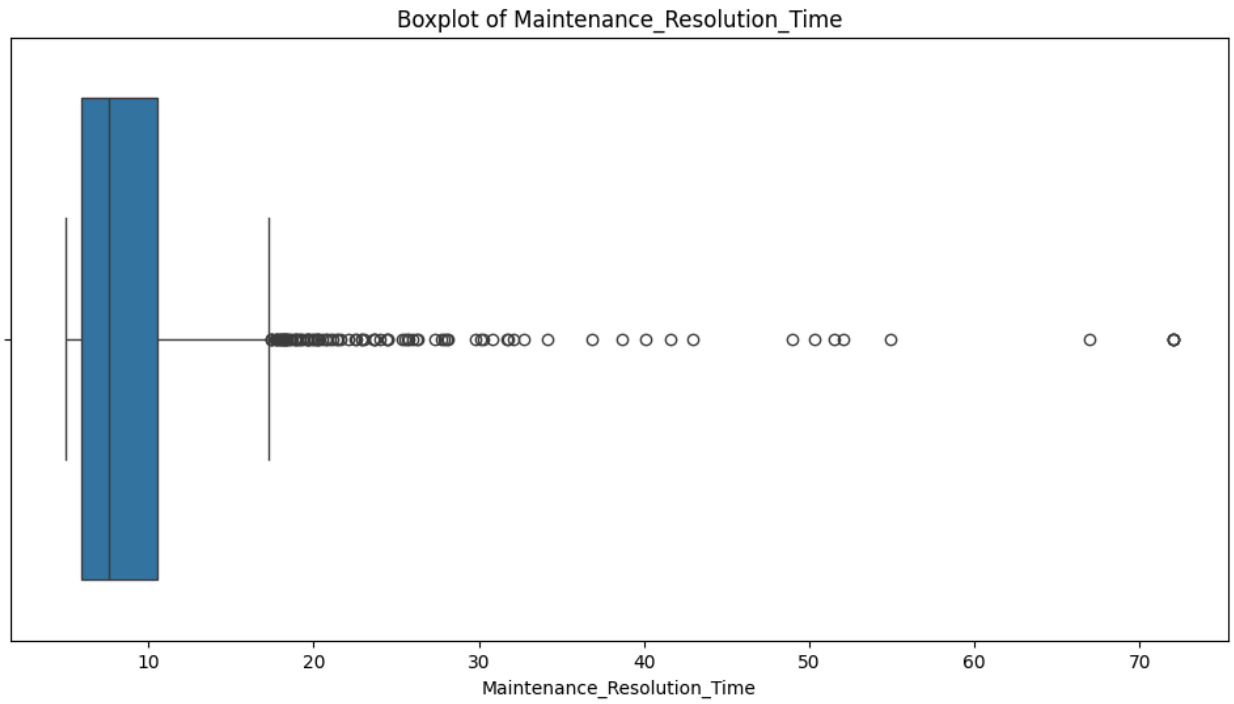

Boxplot of Waste_Recycled_Percentage
0
10
20
30
40
50
60
70
80
Waste_Recycled_Percentage

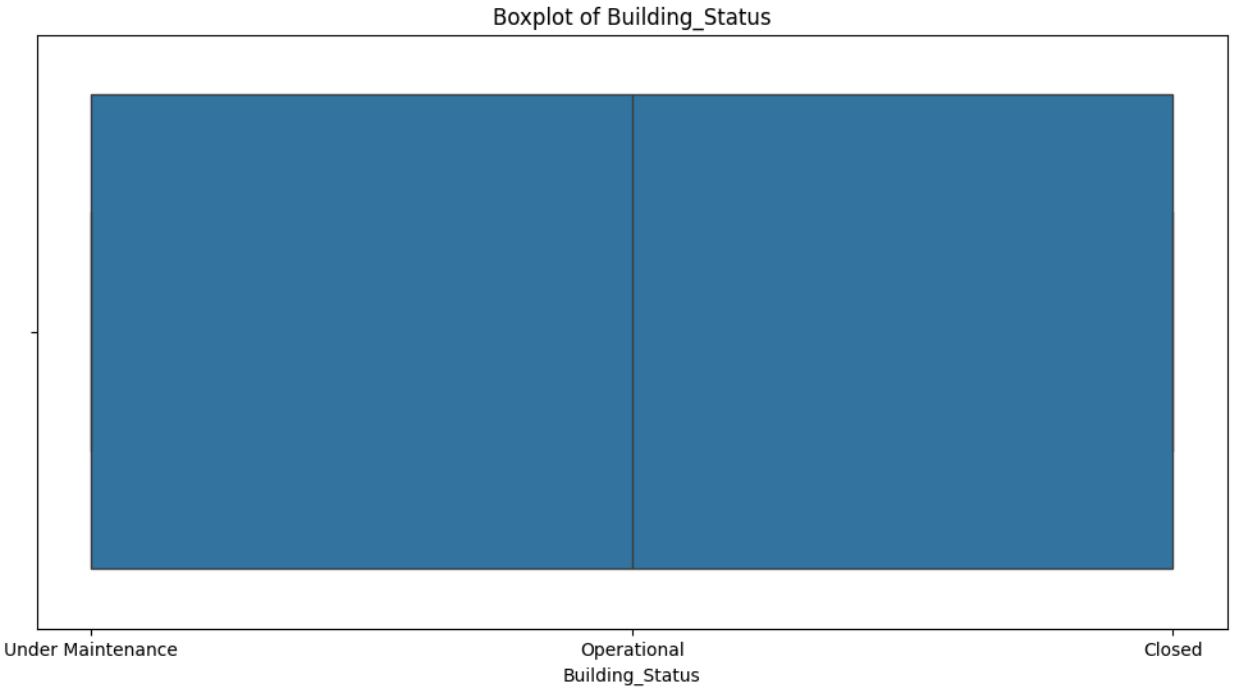

Boxplot of Occupancy_Rate
20
40
60
80
100
Occupancy_Rate

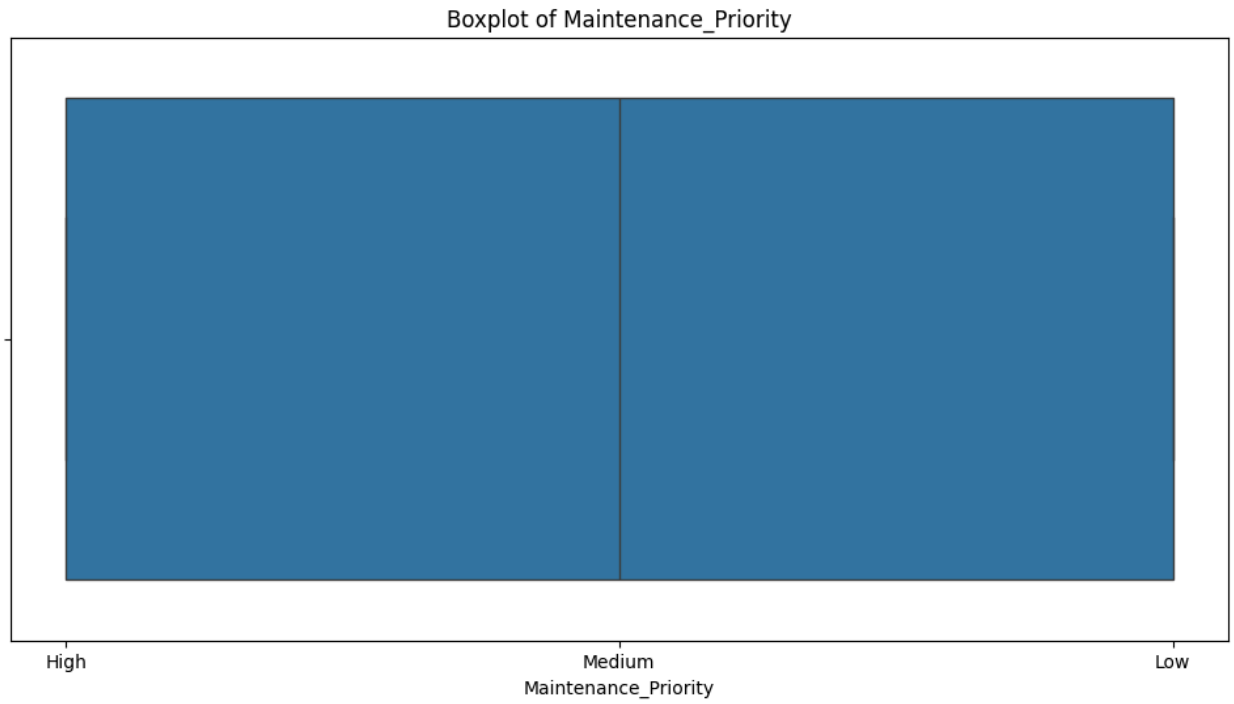

Boxplot of Indoor_Air_Quality
0
20
40
60
80
100
Indoor_Air_Quality

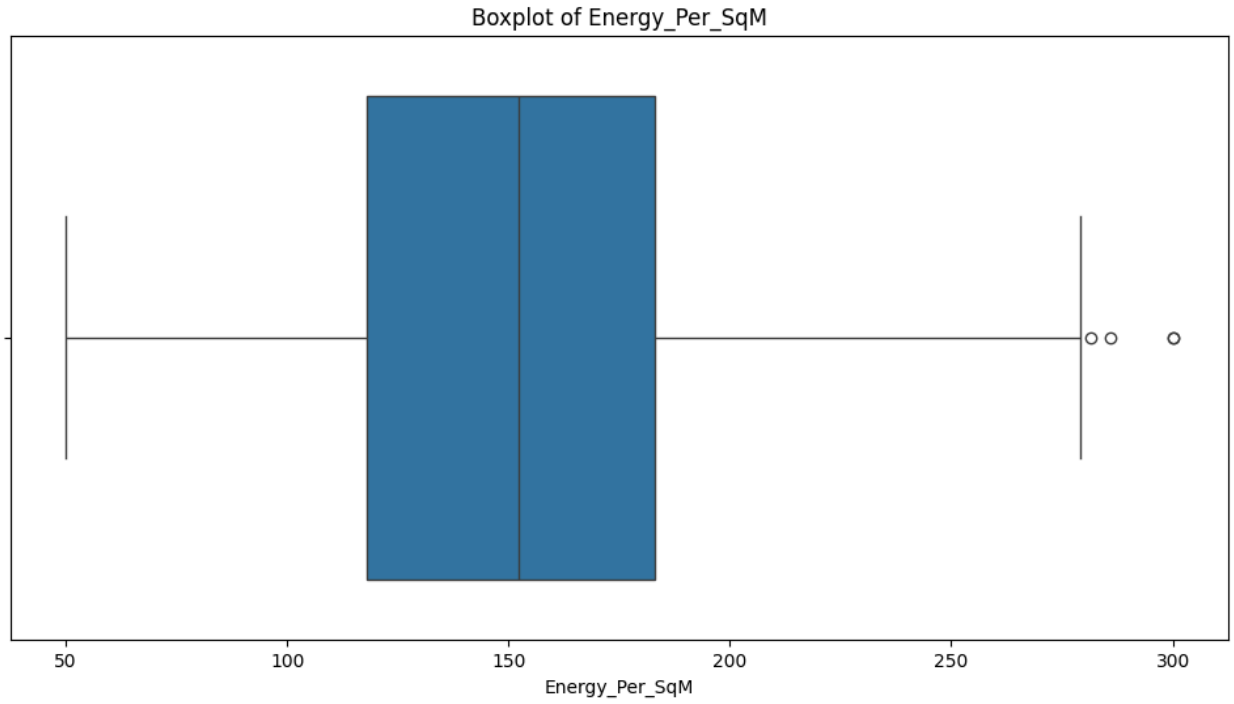

Boxplot of Smart_Devices_Count
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Smart_Devices_Count

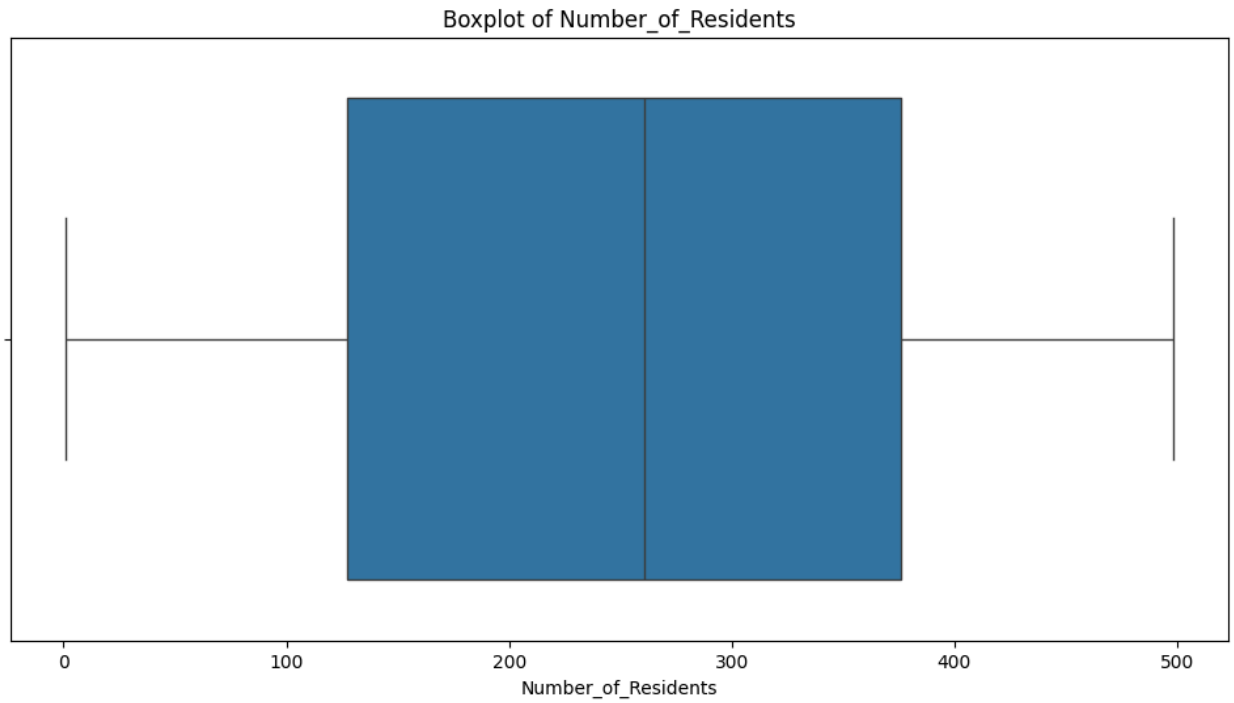

Boxplot of Green_Certified
0.0
0.2
0.4
0.6
0.8
1.0
Green_Certified

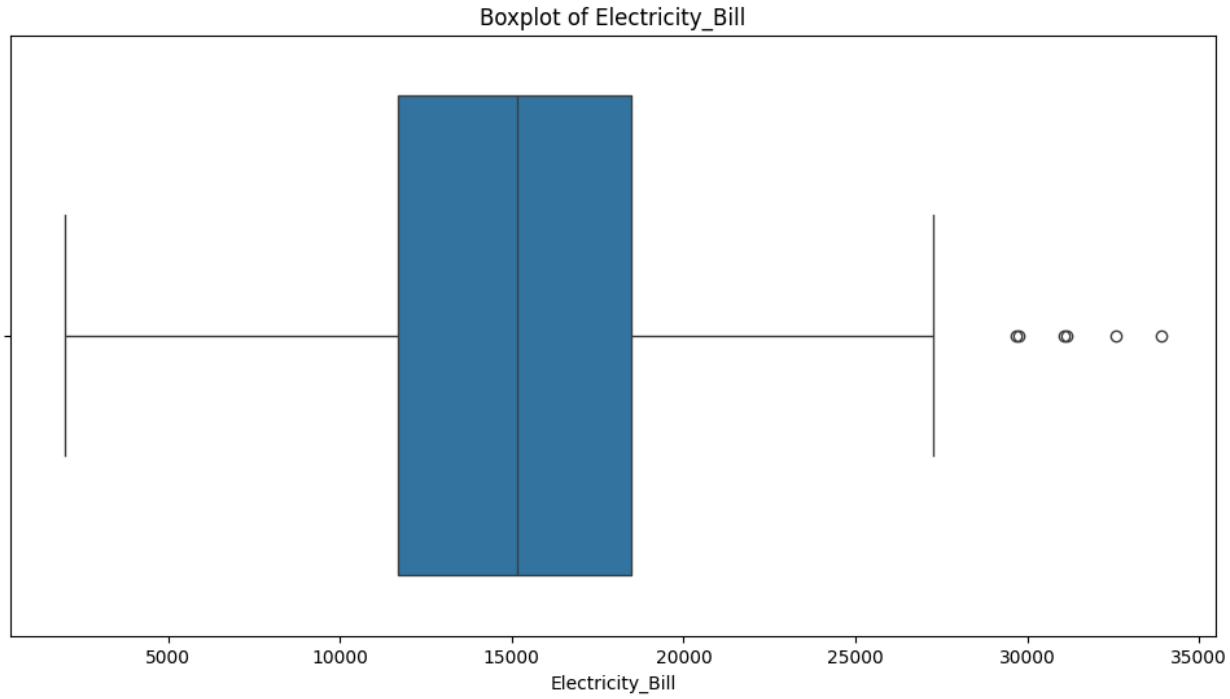

Boxplot of Maintenance_Resolution_Time
10
20
30
40
50
60
70
Maintenance_Resolution_Time

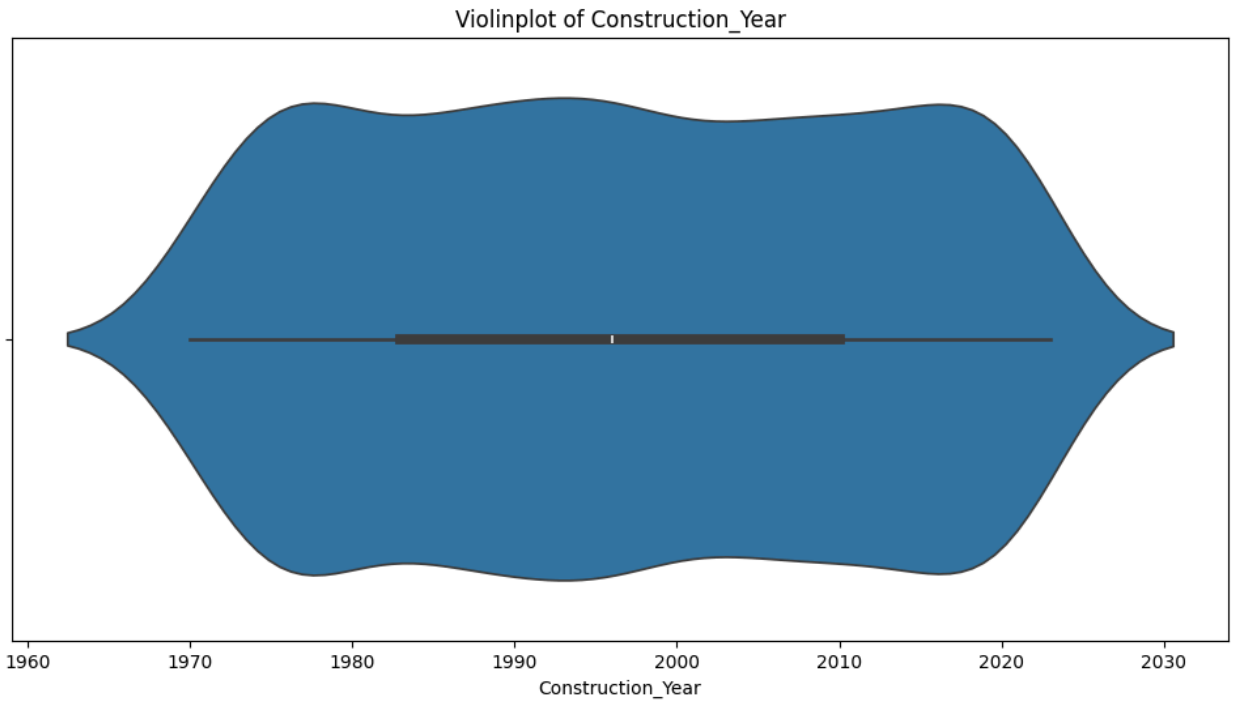

Boxplot of Building_Status
Under Maintenance
Operational
Closed
Building_Status

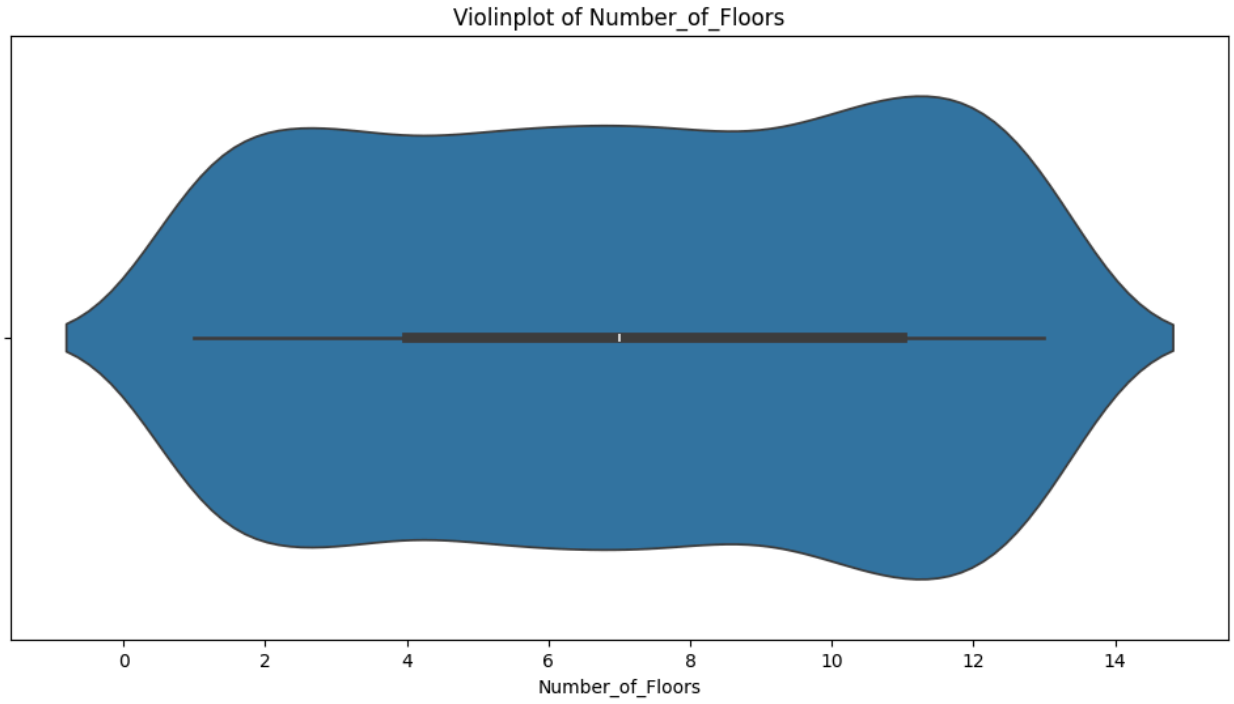

Boxplot of Maintenance_Priority
High
Medium
Low
Maintenance_Priority


Boxplot of Energy_Per_SqM
50
100
150
200
250
300
Energy_Per_SqM

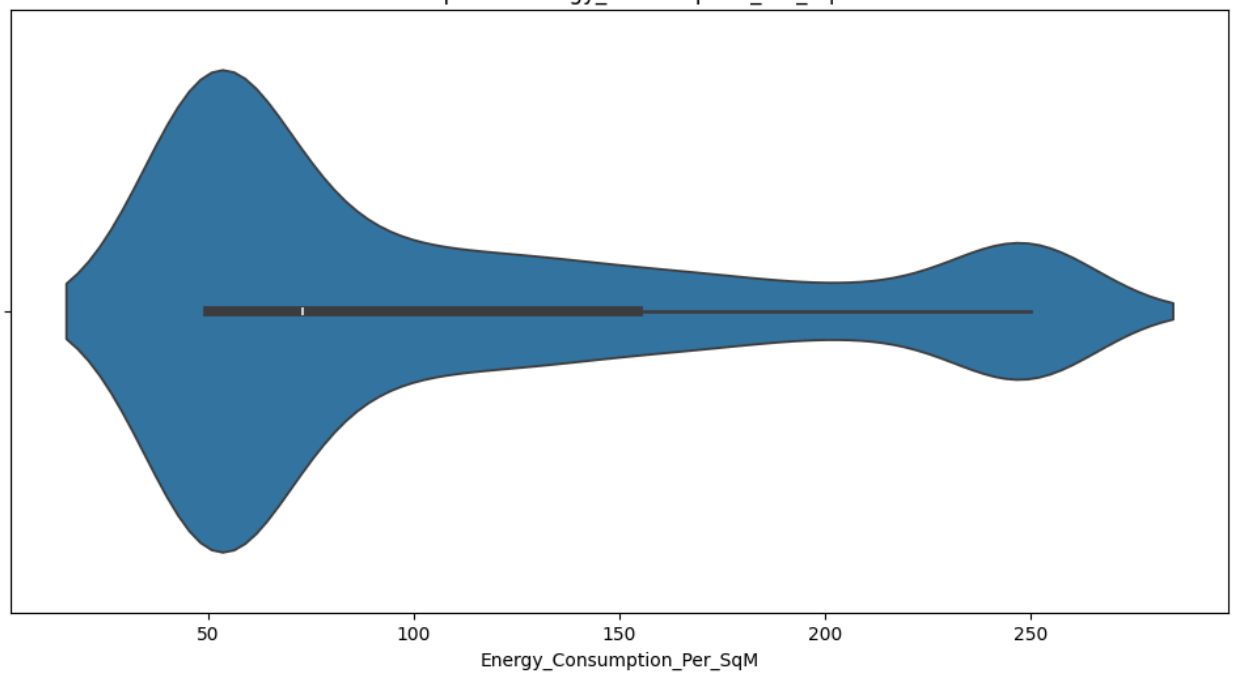

Boxplot of Number_of_Residents
0
100
200
300
400
500
Number_of_Residents


Boxplot of Electricity_Bill
5000
10000
15000
20000
25000
30000
35000
Electricity_Bill

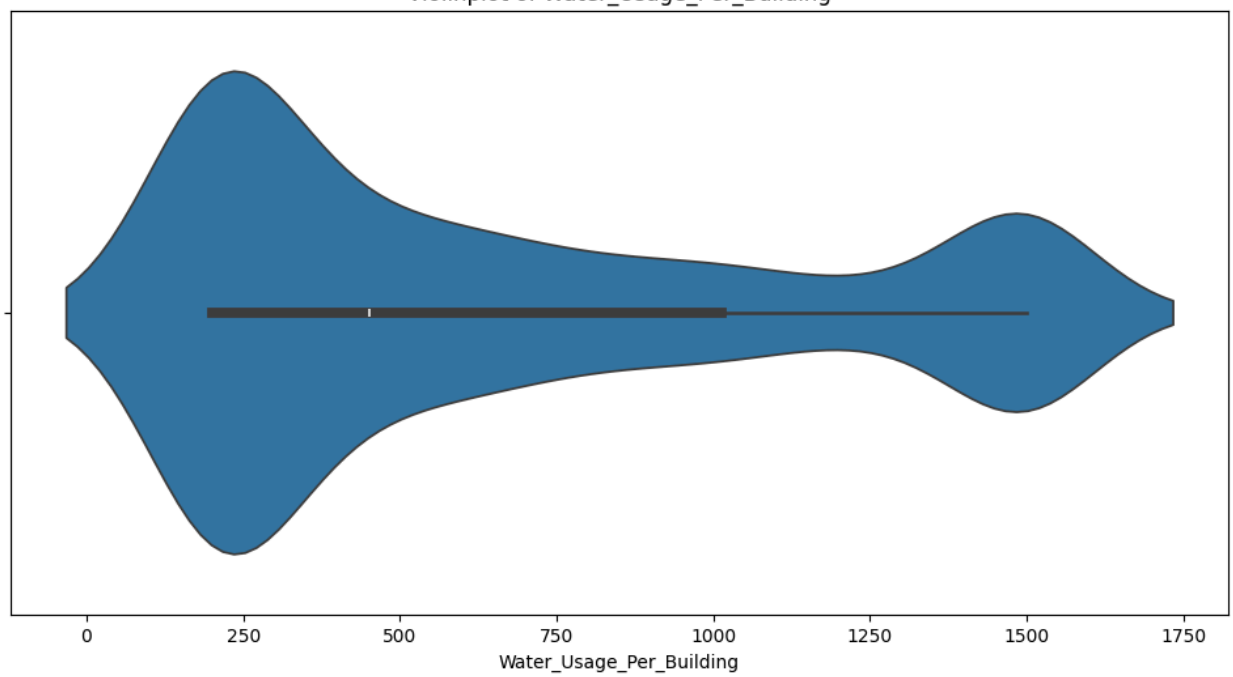

Violinplot of Construction_Year
1960
1970
1980
1990
2000
2010
2020
2030
Construction_Year

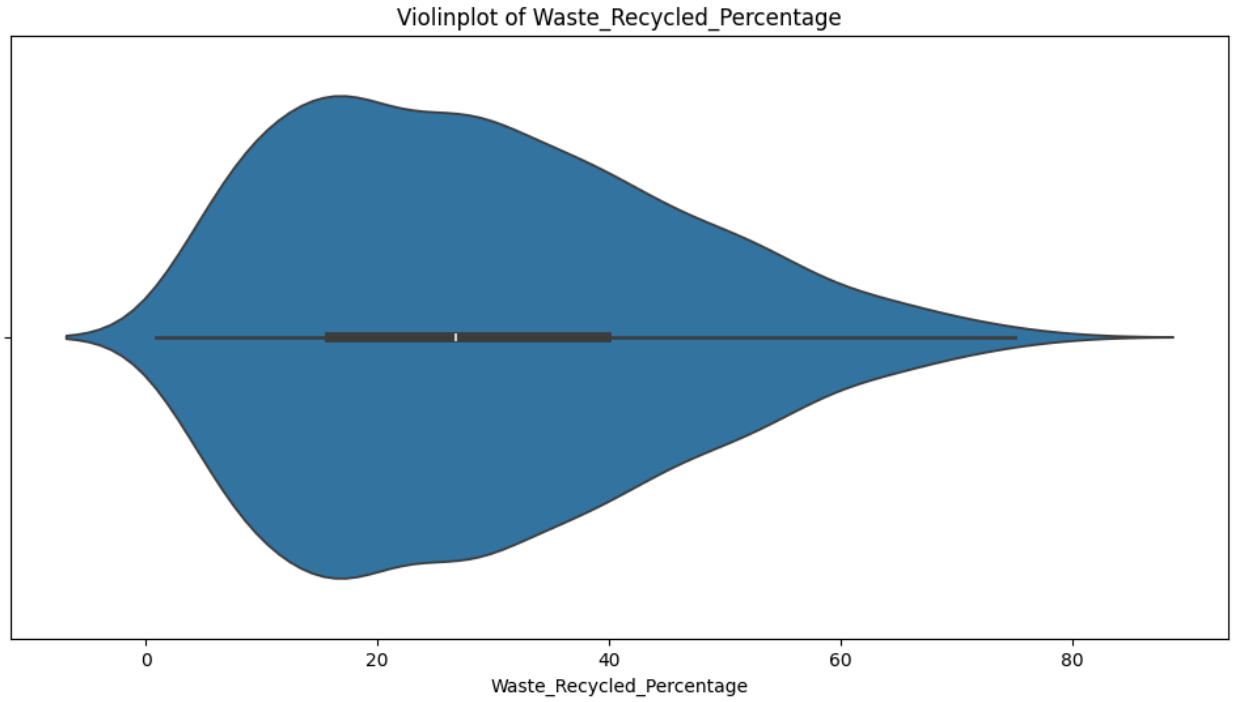

Violinplot of Number_of_Floors
0
2
4
6
8
10
12
14
Number_of_Floors

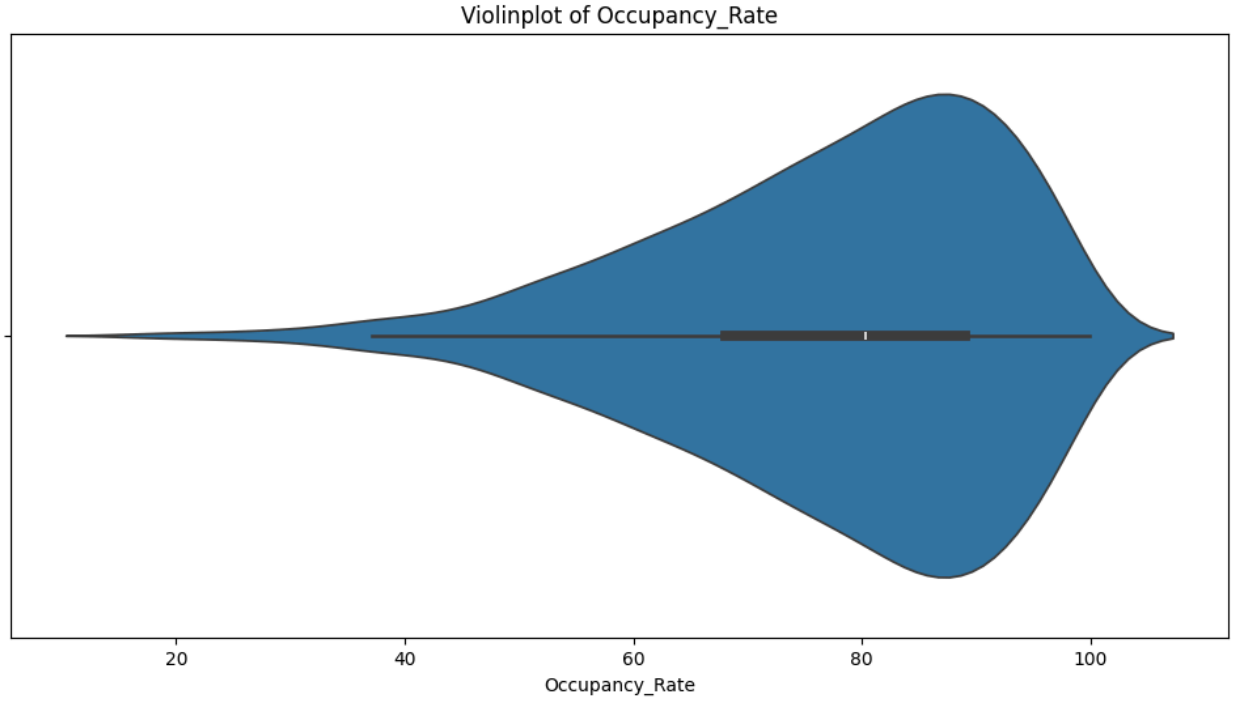
Violinplot of Energy_Consumption_Per_SqM

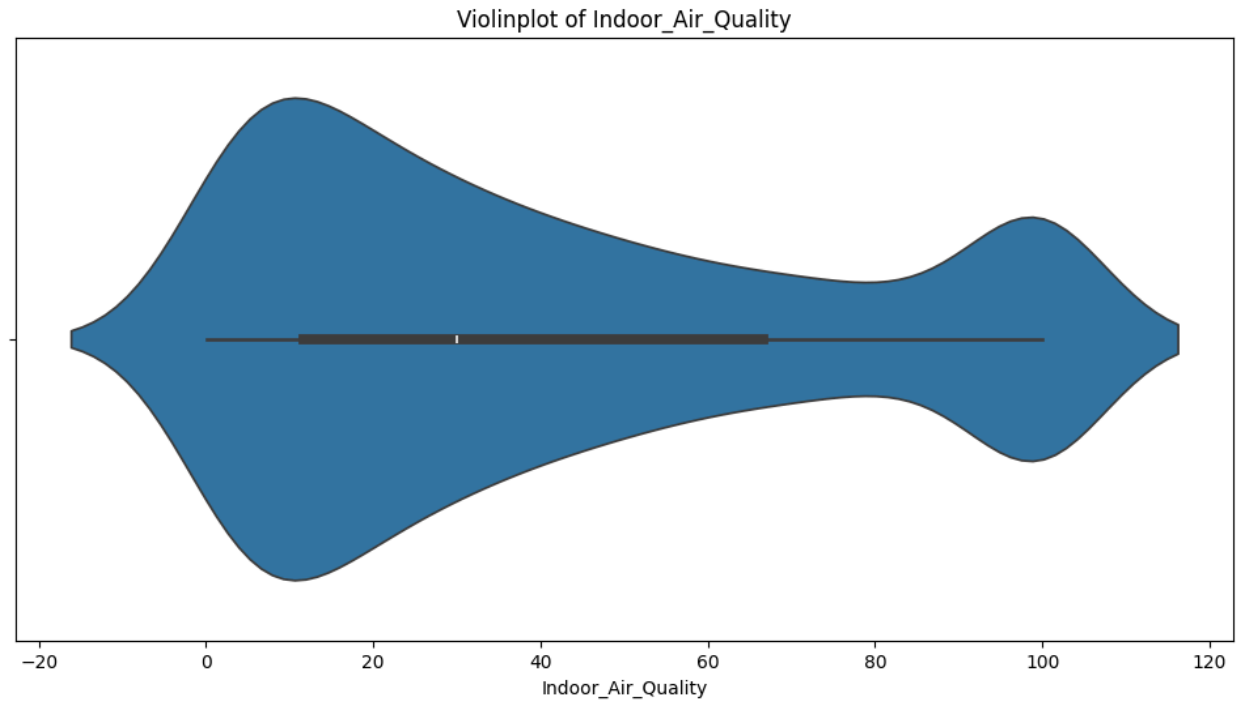


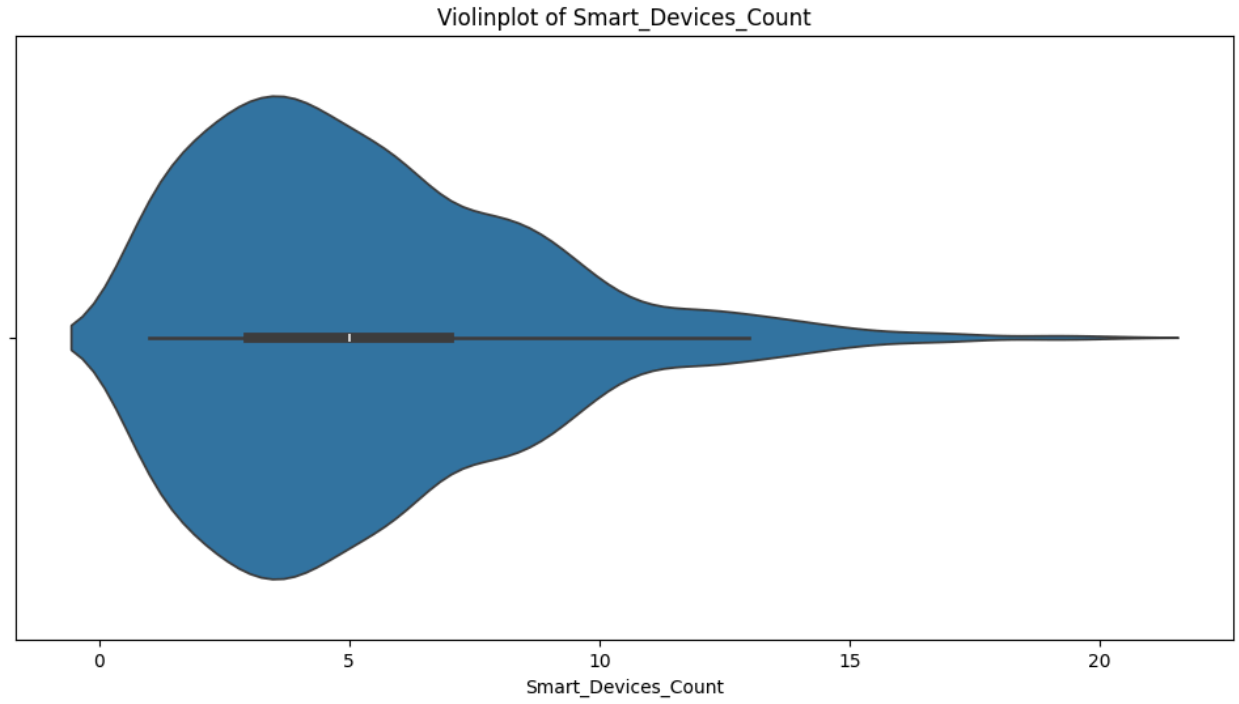
Violinplot of Water_Usage_Per_Building

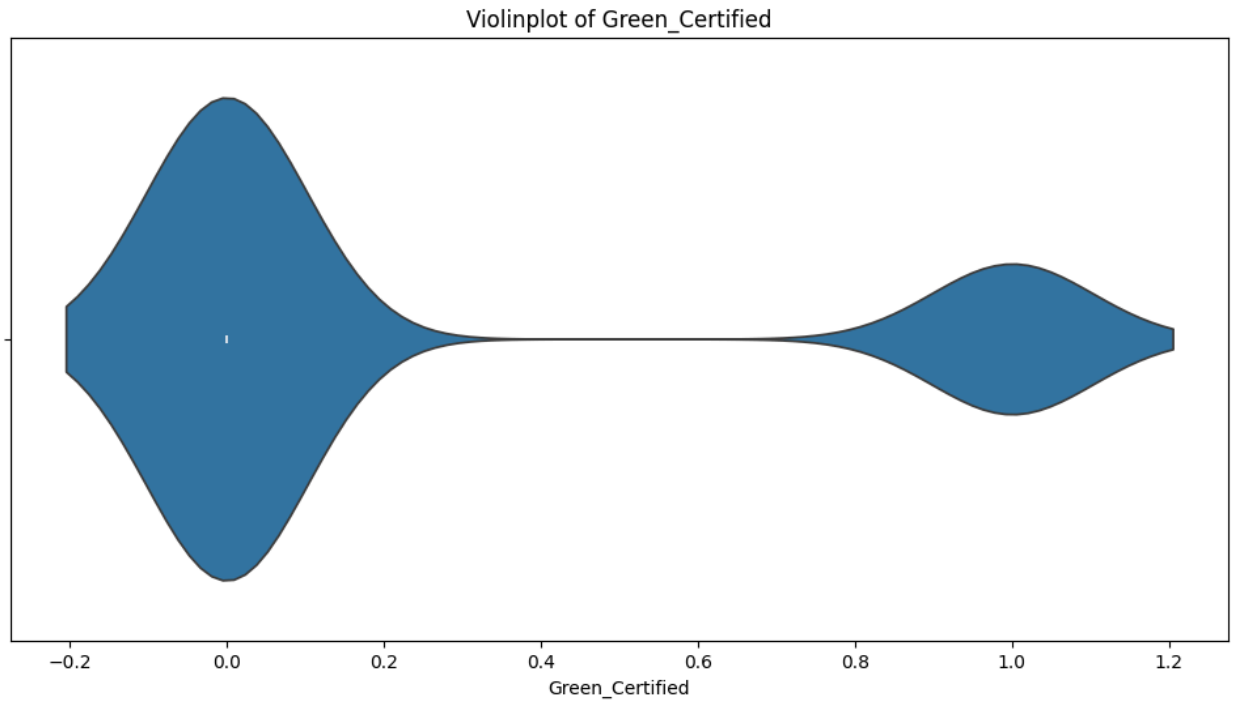


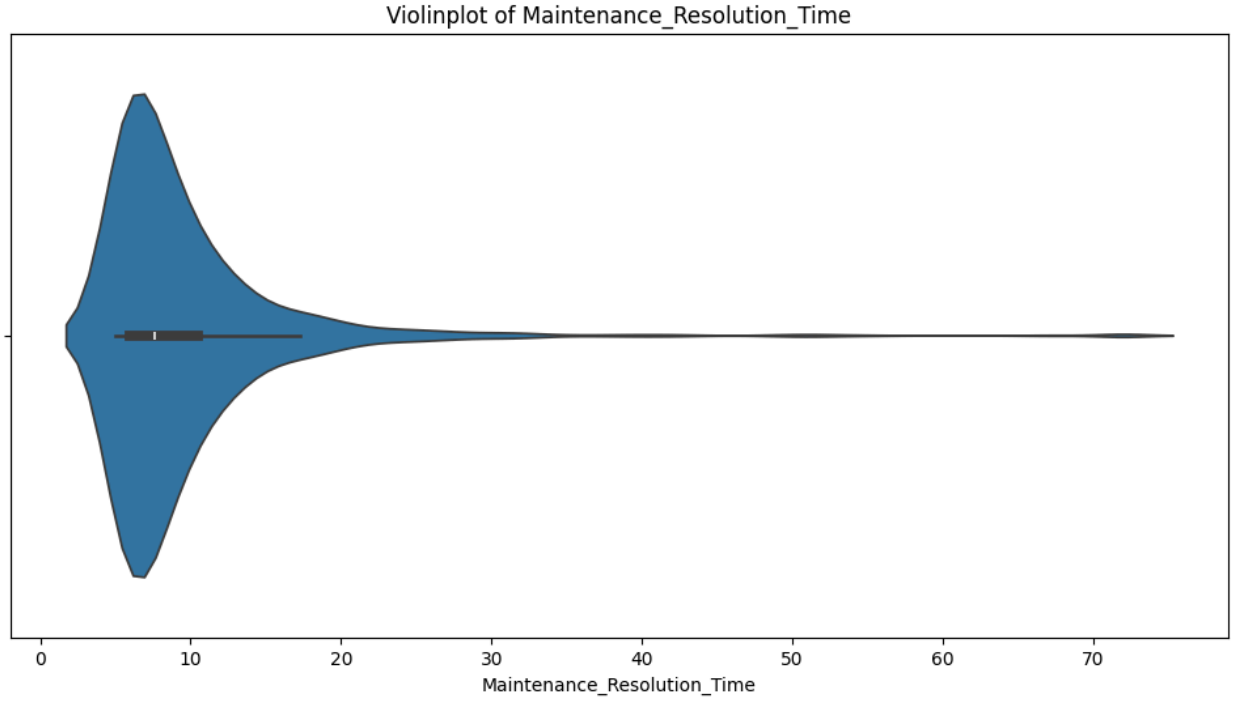

Violinplot of Waste_Recycled_Percentage
0
20
40
60
80
Waste_Recycled_Percentage


Violinplot of Occupancy_Rate
20
40
60
80
100
Occupancy_Rate

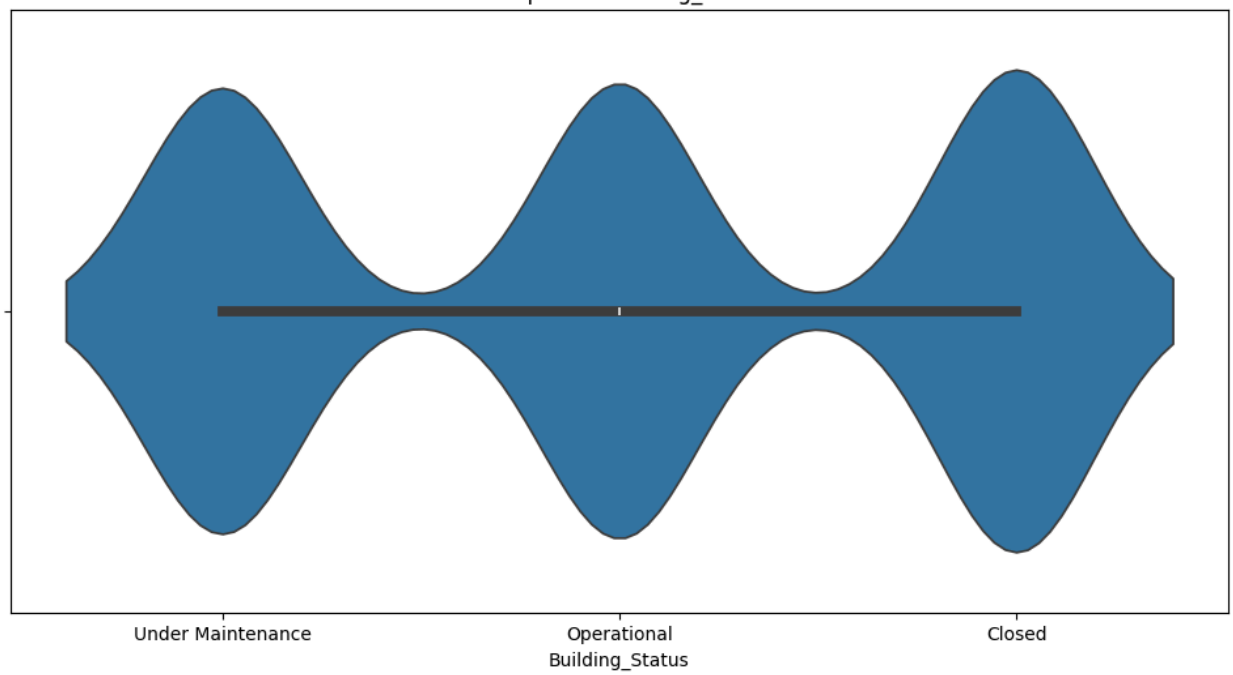

Violinplot of Indoor_Air_Quality
−20
0
20
40
60
80
100
120
Indoor_Air_Quality


Violinplot of Smart_Devices_Count
0
5
10
15
20
Smart_Devices_Count

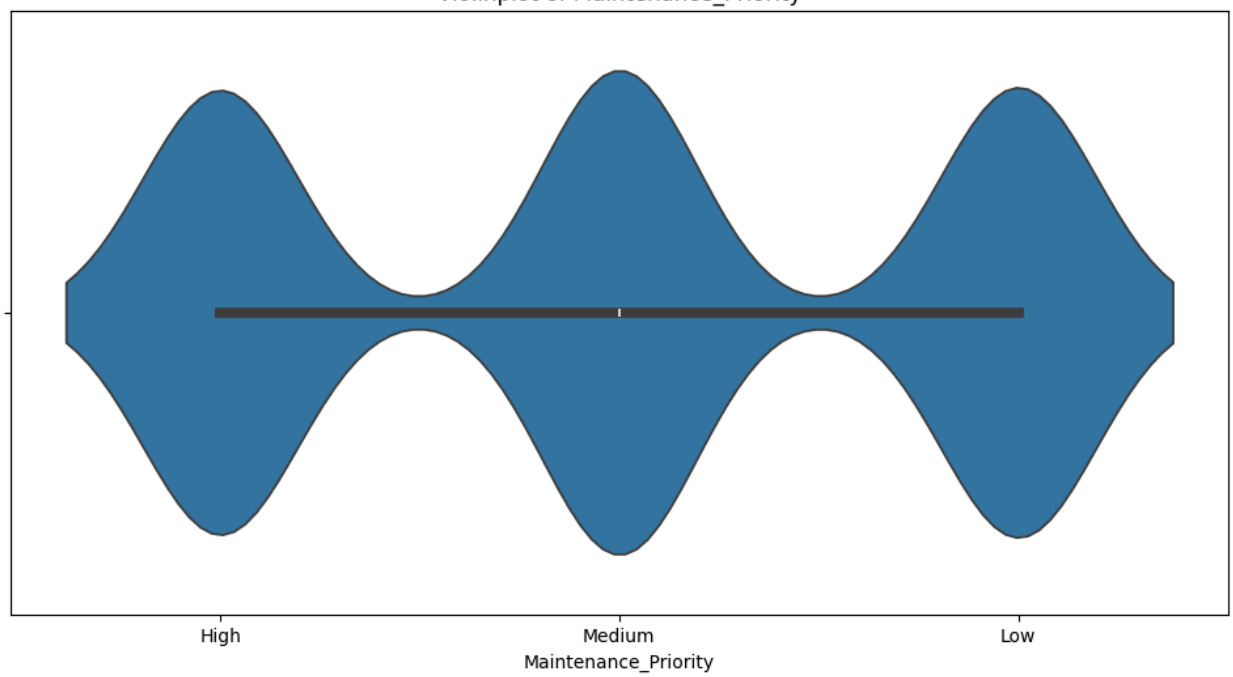

Violinplot of Green_Certified
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Green_Certified

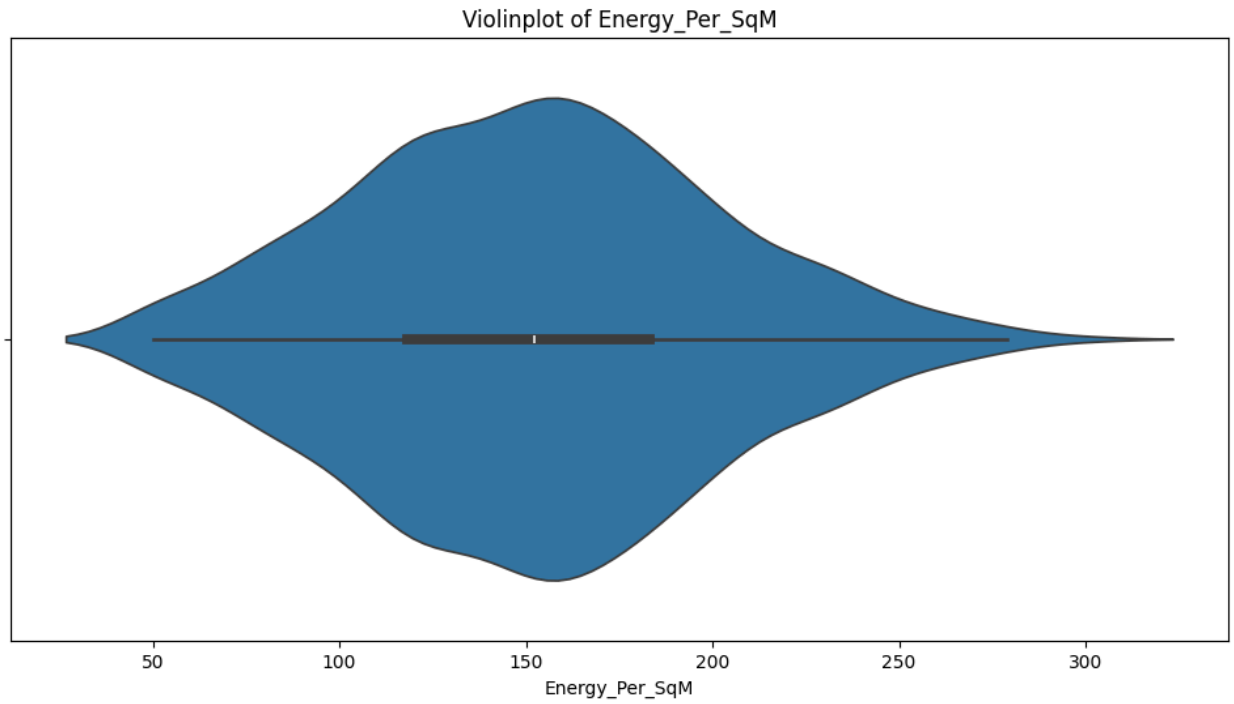

Violinplot of Maintenance_Resolution_Time
0
10
20
30
40
50
60
70
Maintenance_Resolution_Time

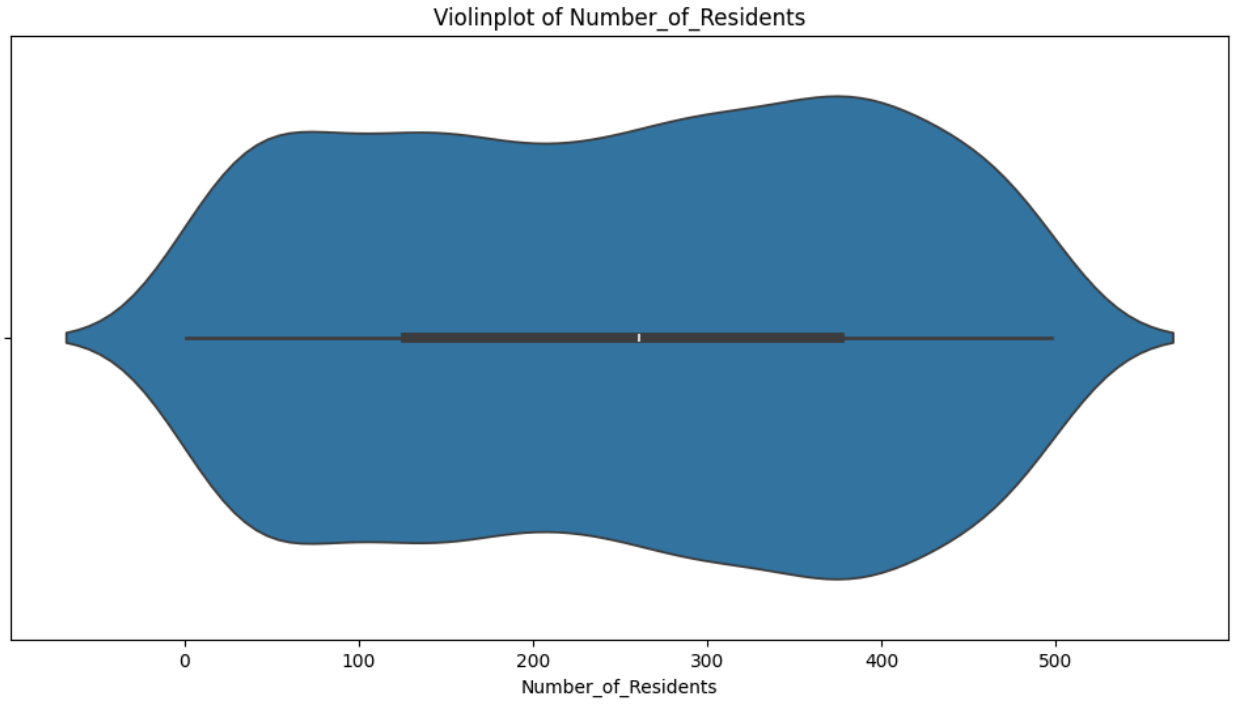
Violinplot of Building_Status

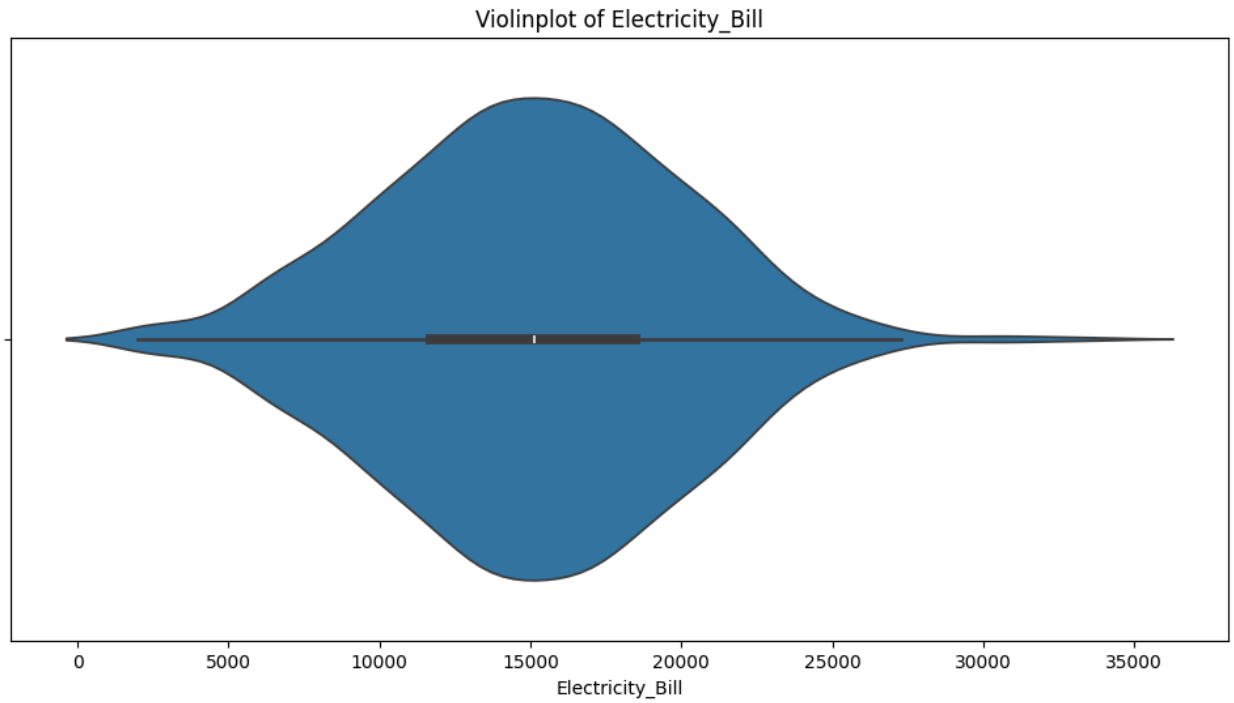


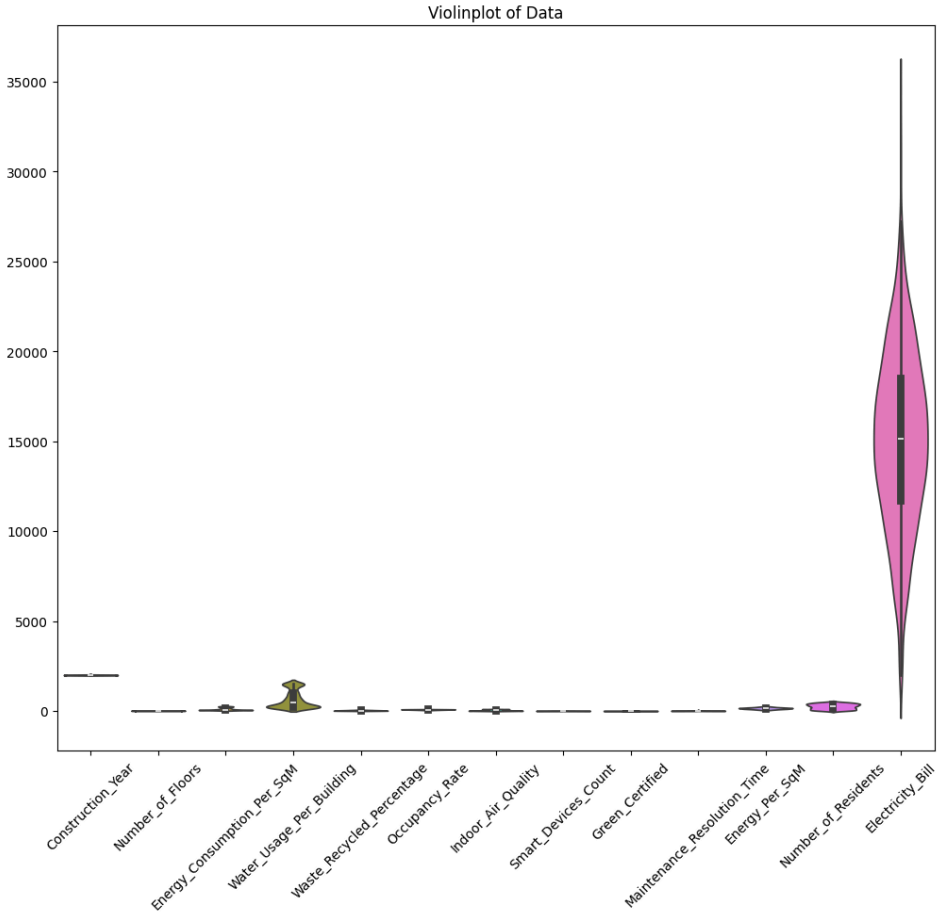
Violinplot of Maintenance_Priority

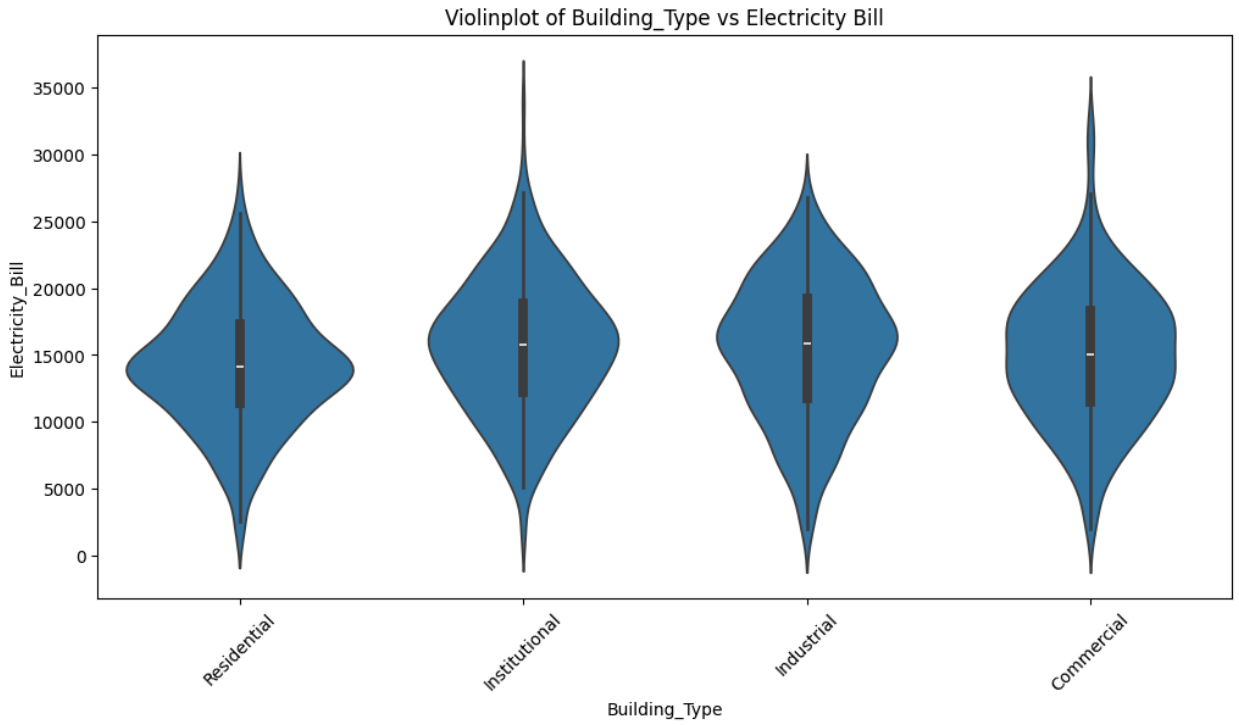


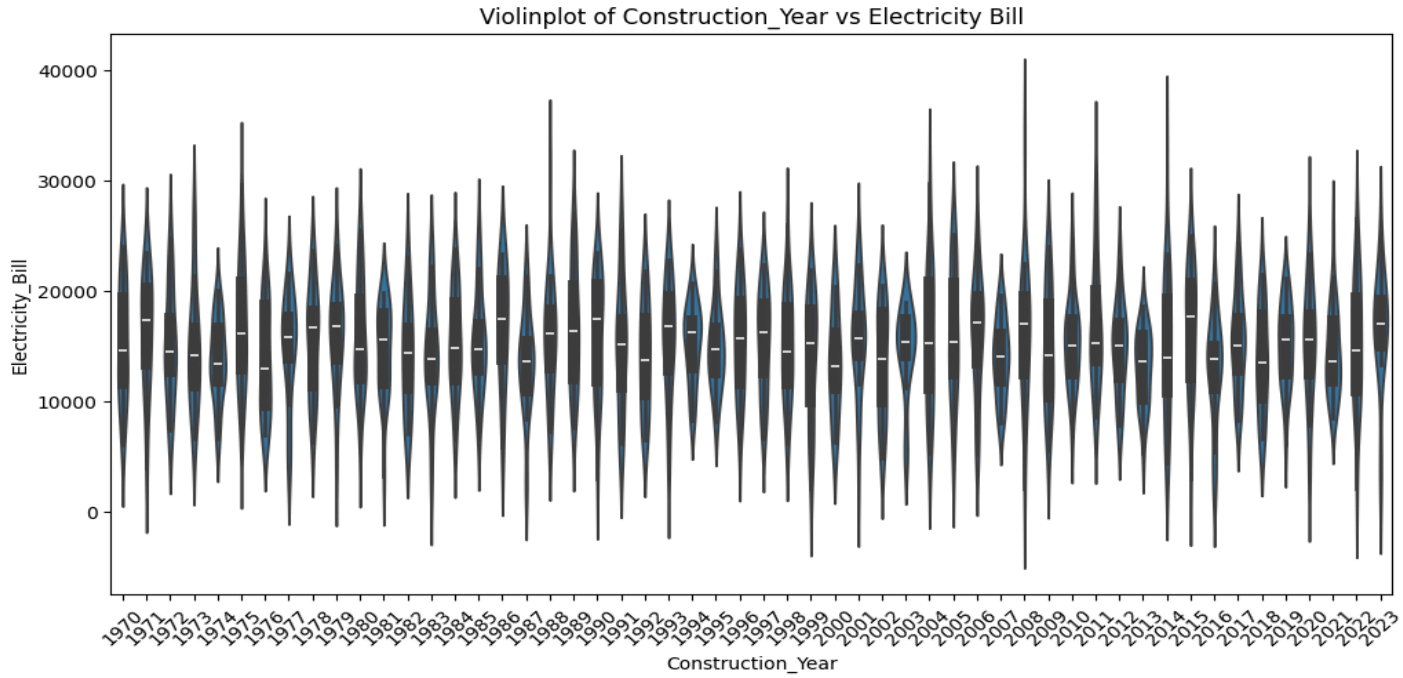

Violinplot of Energy_Per_SqM
50
100
150
200
250
300
Energy_Per_SqM

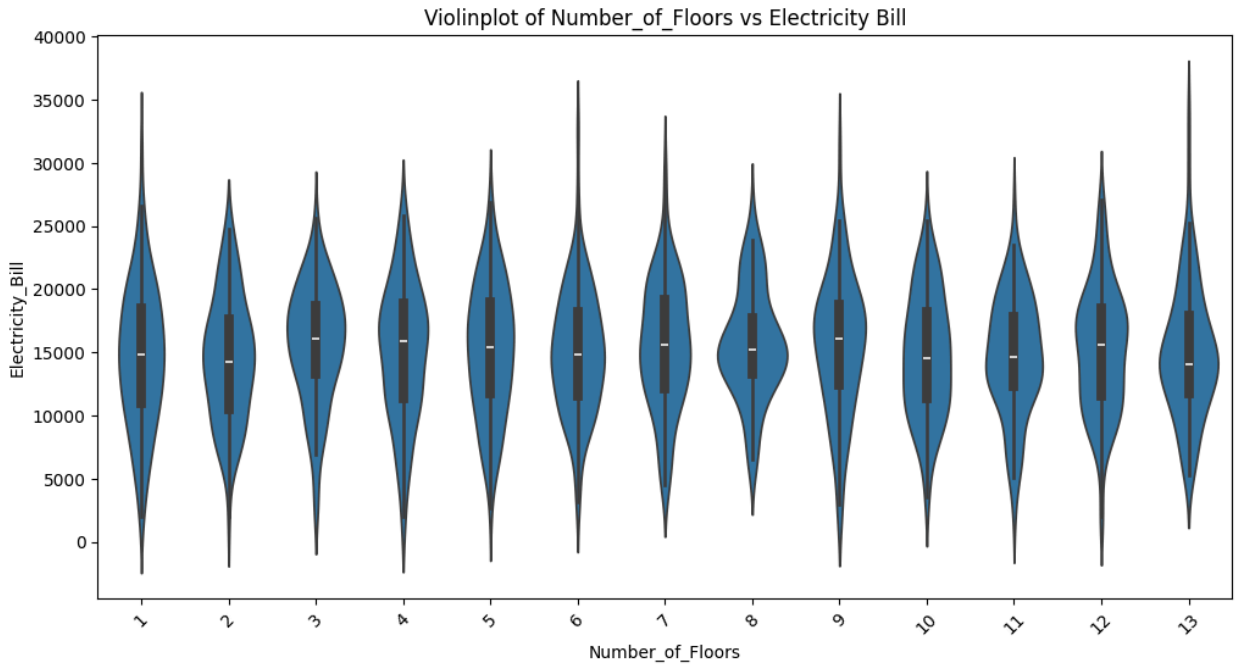

Violinplot of Number_of_Residents
0
100
200
300
400
500
Number_of_Residents

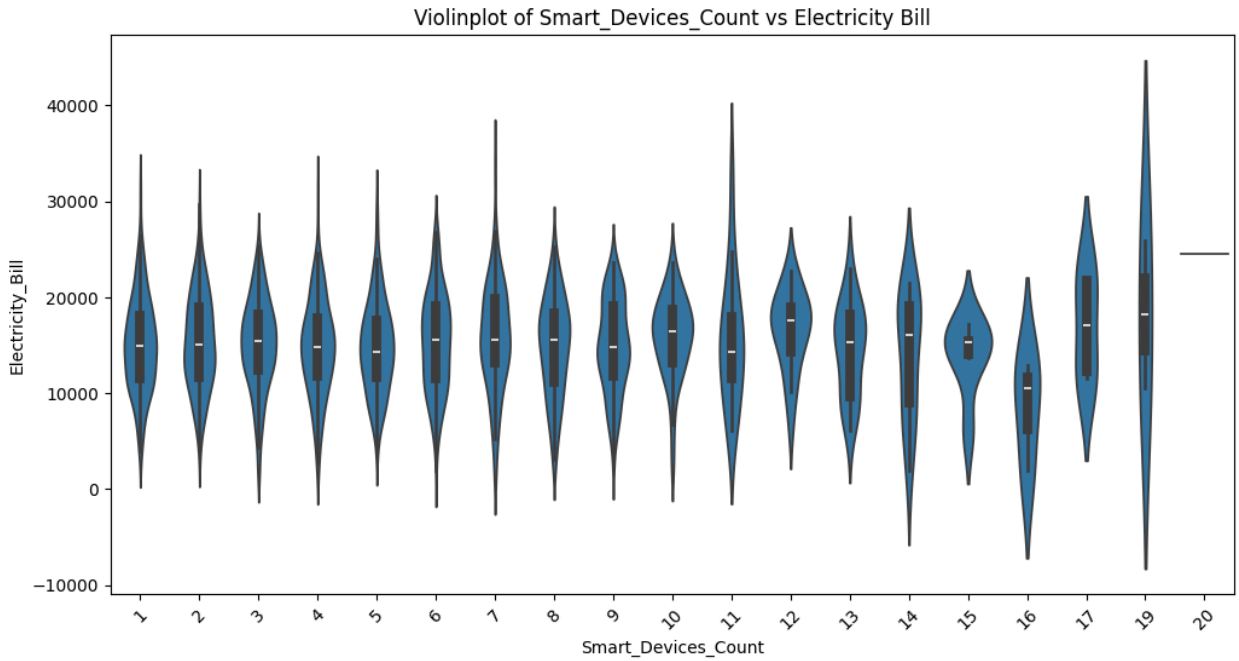

Violinplot of Electricity_Bill
0
5000
10000
15000
20000
25000
30000
35000
Electricity_Bill

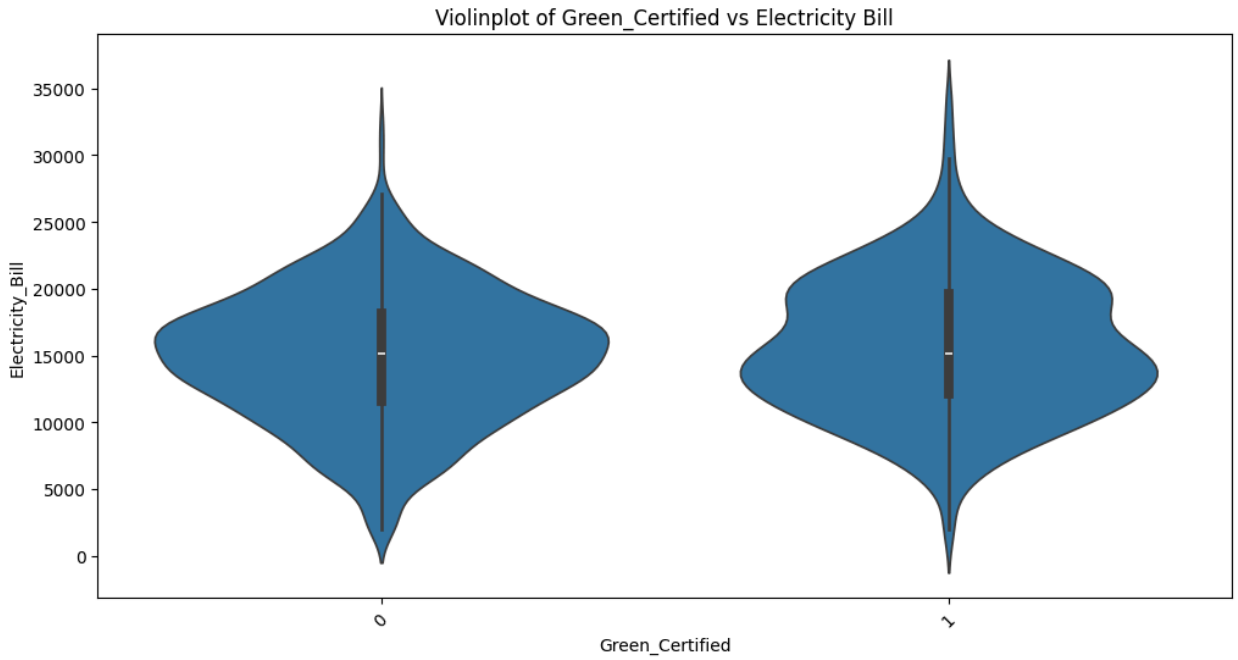

Violinplot of Data
35000
30000
25000
20000
15000
10000
5000
0
Construction_Year
Number_of_Floors
Energy_Consumption_Per_SqM
Water_Usage_Per_Building
Waste_Recycled_Percentage
Occupancy_Rate
Indoor_Air_Quality
Smart_Devices_Count
Green_Certified
Maintenance_Resolution_Time
Energy_Per_SqM
Number_of_Residents
Electricity_Bill

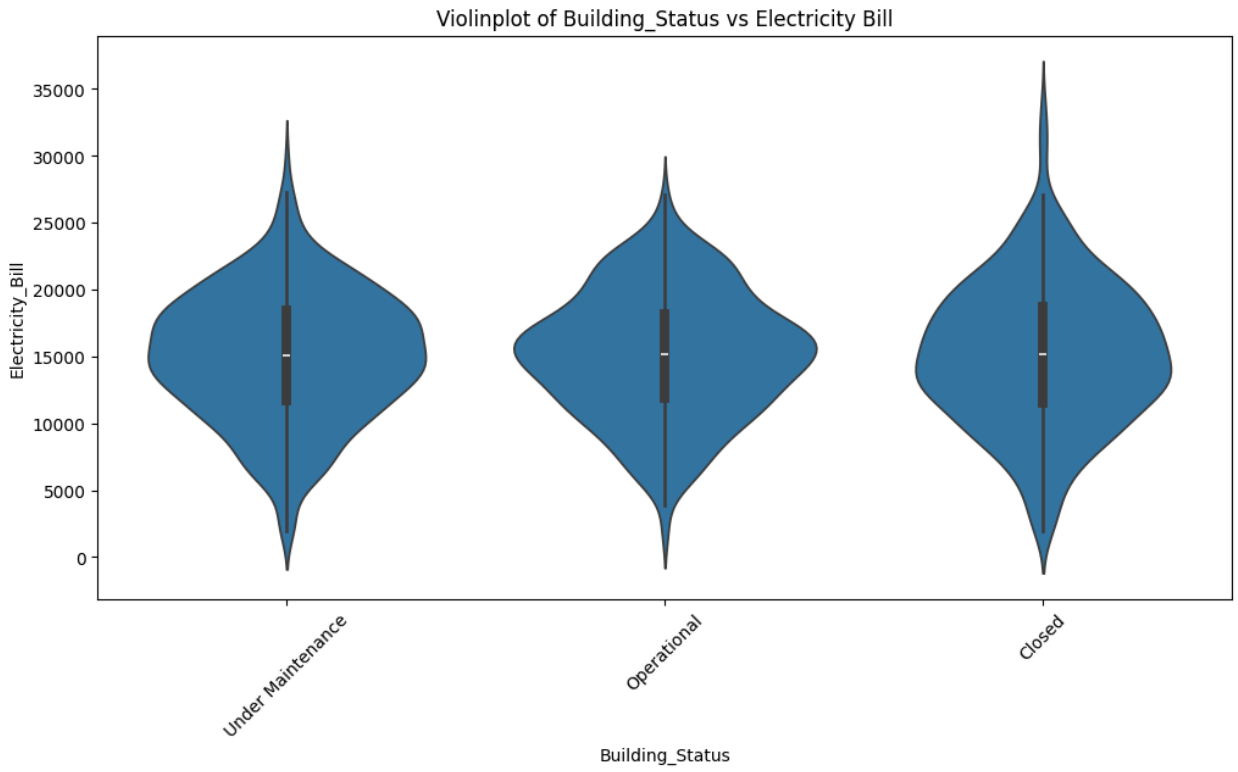

Violinplot of Building_Type vs Electricity Bill
Electricity_Bill
35000
30000
25000
20000
15000
10000
5000
0
Residential
Institutional
Industrial
Commercial
Building_Type

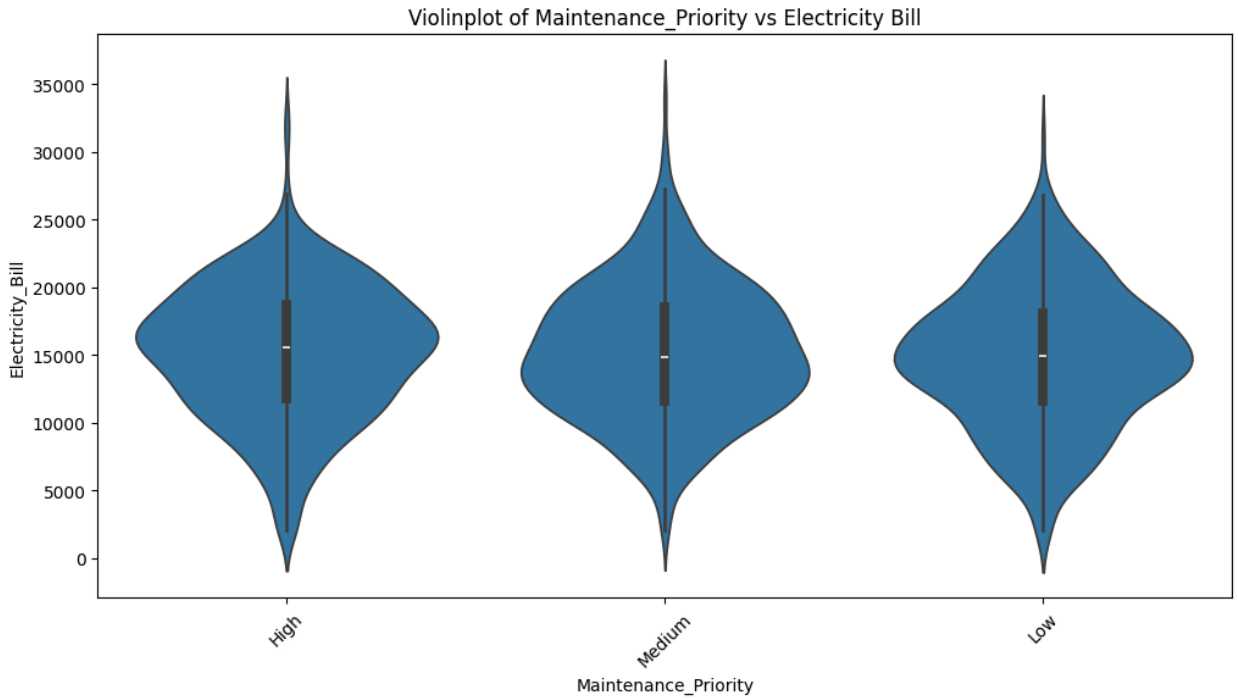

Violinplot of Construction_Year vs Electricity Bill
Electricity_Bill
40000
30000
20000
10000
0
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
Construction_Year

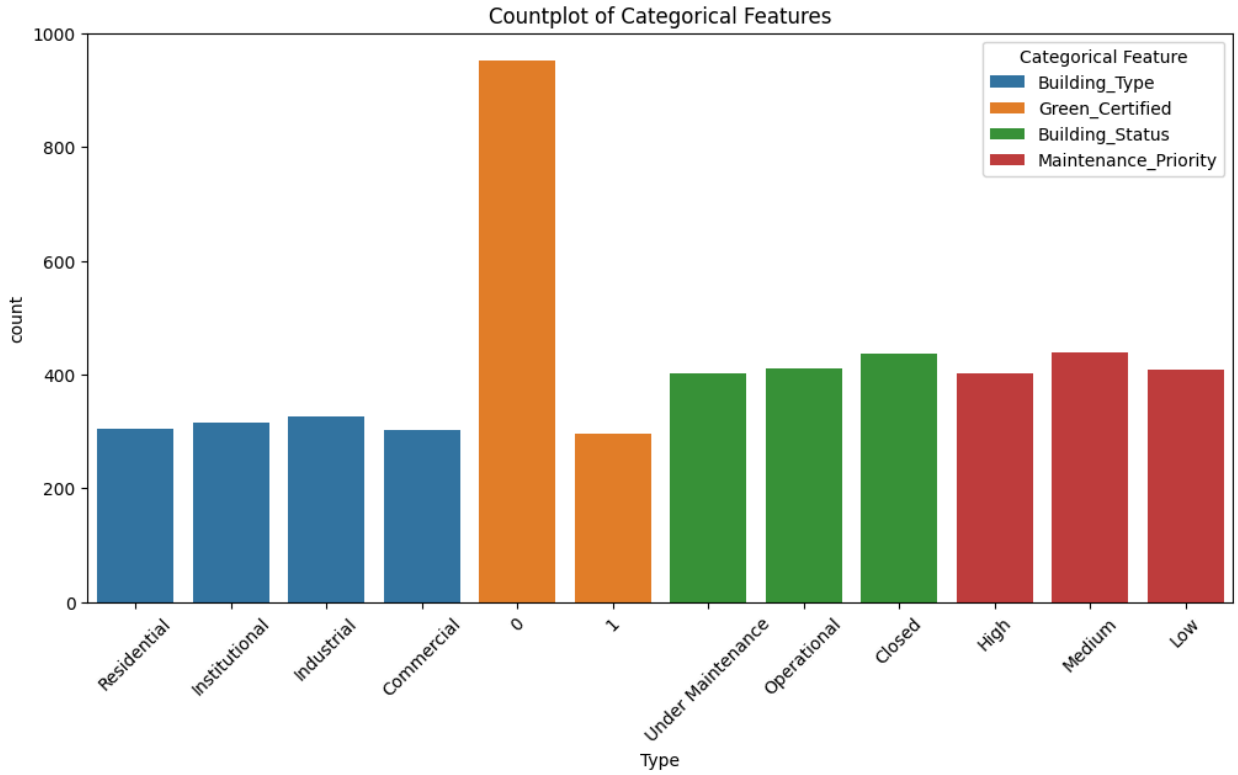

Violinplot of Number_of_Floors vs Electricity Bill
Electricity_Bill
40000
35000
30000
25000
20000
15000
10000
5000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
Number_of_Floors

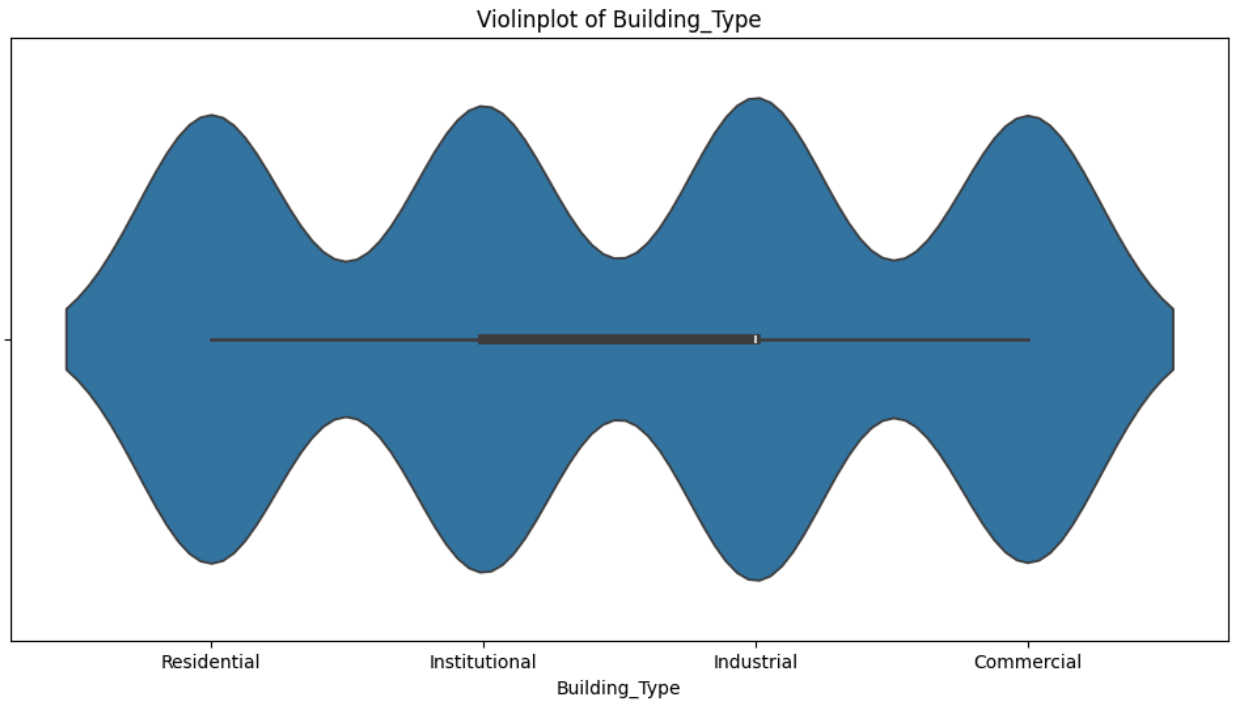

Violinplot of Smart_Devices_Count vs Electricity Bill
Electricity_Bill
40000
30000
20000
10000
0
−10000
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
19
20
Smart_Devices_Count

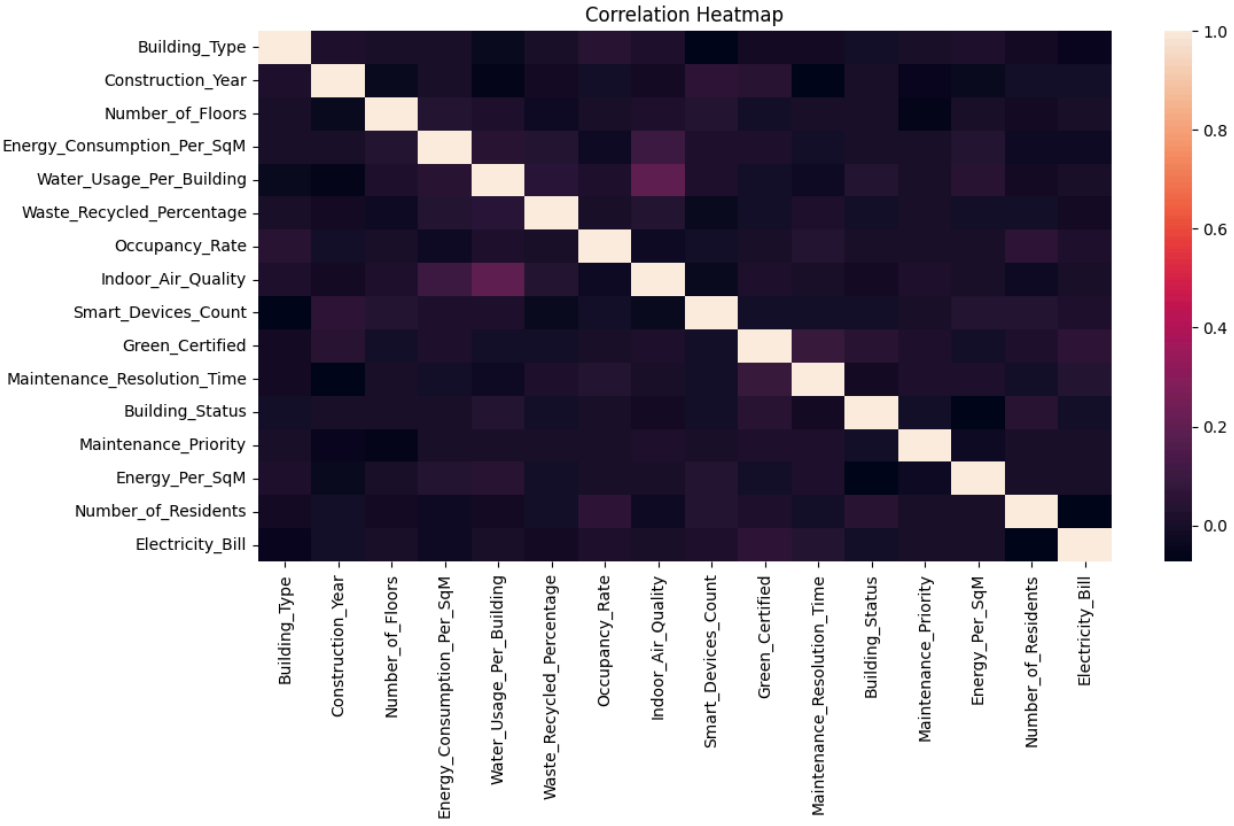

Violinplot of Green_Certified vs Electricity Bill
Electricity_Bill
35000
30000
25000
20000
15000
10000
5000
0
0
1
Green_Certified


Violinplot of Building_Status vs Electricity Bill
Electricity_Bill
35000
30000
25000
20000
15000
10000
5000
0
Under Maintenance
Operational
Closed
Building_Status


Violinplot of Maintenance_Priority vs Electricity Bill
Electricity_Bill
35000
30000
25000
20000
15000
10000
5000
0
High
Medium
Low
Maintenance_Priority


Countplot of Categorical Features
Categorical Feature
Building_Type
Green_Certified
Building_Status
Maintenance_Priority
count
1000
800
600
400
200
0
Residential
Institutional
Industrial
Commercial
0
1
Under Maintenance
Operational
Closed
High
Medium
Low
Type

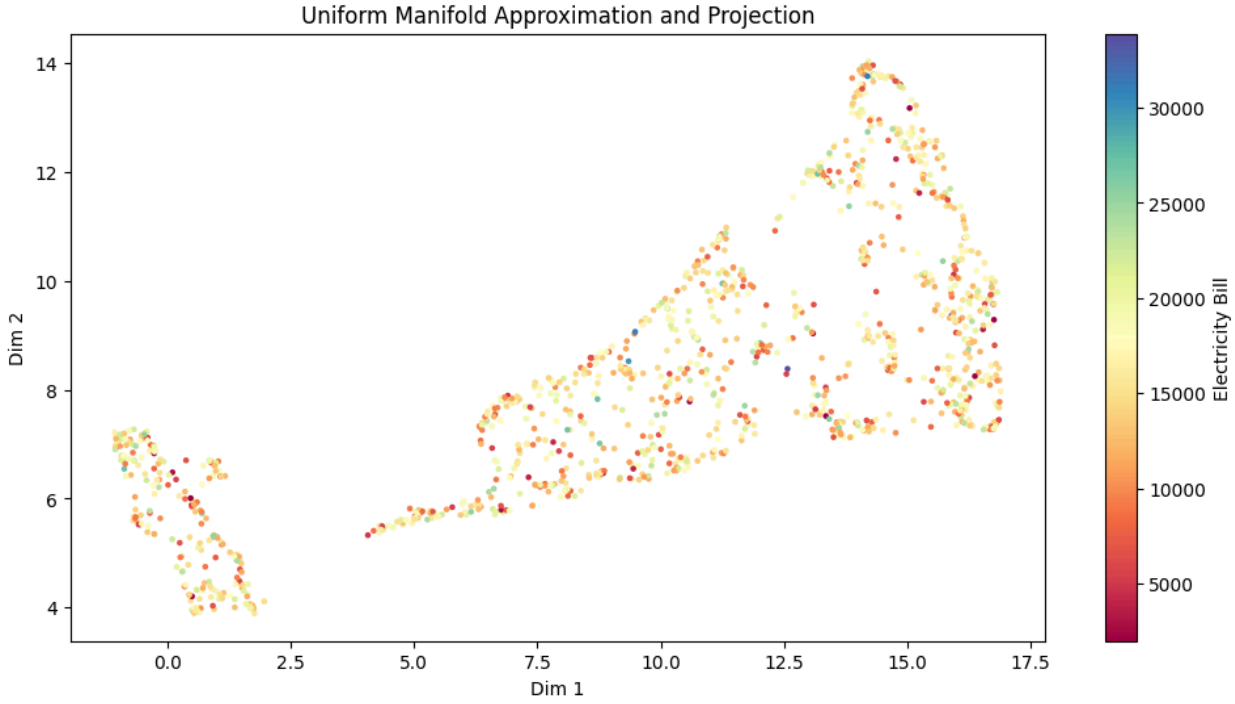

Violinplot of Building_Type
Residential
Institutional
Industrial
Commercial
Building_Type


Correlation Heatmap
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
Electricity_Bill
1.0
0.8
0.6
0.4
0.2
0.0

Insights:-

1) Buildings that are green certified have a higher electricity bill than those that don't on average based on the box plot
2) The dataset itself contains far more buildings that aren't green certified than those that are green certified.
3) As the maintenance resolution rate increases, the occupancy rate increases (from the pair plot).
4) With a larger number of smart devices, the maintenance resolution time decreases (from the pair plot).
5) The dataset has more samples that have a higher occupancy rate and also, most of its samples have electricity bills that are in the midrange rather than in the extreme, it is not evenly skewed across all values of bills.

(b)



2 (or maybe 3) main clusters have been formed. In terms of the separability between different ranges of electricity bill, there isn't much. Though, the points with very low bill are clumped together generally. So, not much separability.

(c) `MSE Error -> Train : 24655148.257489815, Test : 23366790.69266435`
`RMSE Error -> Train : 4965.395075670194, Test : 4833.9208405459385`
`R2 Score -> Train : 0.018294250744540497, Test : -0.008382098814608385`
`Adjusted R2 Score -> Train : 0.0033292240790609995, Test : -0.0730219769437499`

`MAE Error -> Train : 4002.8327810662854, Test : 3846.6264309277353`

MSE error is so high because the spread in the range of electricity bills is quite a bit, and any small deviation and the error can go to the power of 10^6, so such a high MSE is expected. The RMSE and MAE take care of this since the squared term no longer has the power in those cases, and it's either rooted or just the absolute error is taken.

For each of these cases, Test is performing worse than train, which is as expected. We can analyze 2 models by looking at their R2 scores and the one with the higher one has better performance, similar to the Adjusted R2 score. R2 scores are usually a bit higher, so there's not much variation in the dependent variable that is predictable from the independent variable.

(d) 3 most important features :- ['Building_Type', 'Green_Certified', 'Number_of_Residents']

These make sense to an extent as Building Type and Green Certification could reduce the bill by a bit. And the more the residents, you would expect to have a higher bill.

```
MSE Error -> Train : 24766070.56264958, Test : 23136682.2904276
RMSE Error -> Train : 4976.552075749793, Test : 4810.060528769633
R2 Score -> Train : 0.013877604632386964, Test : 0.0015481135390811307
Adjusted R2 Score -> Train : 0.010907356453568773, Test :
-0.010628128978735107
MAE Error -> Train : 4008.6773071820185, Test : 3814.805127504474
```

Metrics				
	Without Feature Elimination		With Feature Elimination	
	Train	Test	Train	Test
MSE Error	24655148.257489815	23366790.69266435	24766070.56264958	23136682.2904276
RMSE Error	4965.395075670194	4833.9208405459385	4976.552075749793	4810.060528769633
R2 Score	0.018294250744540497	-0.008382098814608385	0.013877604632386964	0.0015481135390811307
Adj. R2 Score	0.0033292240790609995	-0.0730219769437499	0.010907356453568773	-0.010628128978735107
MAE Error	4002.8327810662854	3846.6264309277353	4008.6773071820185	3814.805127504474

With Feature Elimination, the MSE, RMSE and MAE errors are lower on test set but higher on train set. So, with feature elimination, the model generalizes better. And the R2 as well as the Adjusted R2 scores are higher for test set for Feature Eliminated model, which means that the fit is better.

(e)
```
MSE Error -> Train : 24292870.059935816, Test : 23495181.827085067
RMSE Error -> Train : 4928.779773933485, Test : 4847.182875349873
R2 Score -> Train : 0.032719253817098726, Test : -0.013922753639620034
Adjusted R2 Score -> Train : 0.009924728036149189, Test :
-0.11710958254984671

MAE Error -> Train : 3959.326160703784, Test : 3843.5596807397287
```

Metrics

	Label Encoding		One Hot Encoding	
	Train	Test	Train	Test
MSE Error	24655148.257489815	23366790.69266435	24292870.059935816	23495181.827085067
RMSE Error	4965.395075670194	4833.9208405459385	4928.779773933485	4847.182875349873
R2 Score	0.018294250744540497	-0.008382098814608385	0.032719253817098726	-0.013922753639620034
Adj. R2 Score	0.0033292240790609995	-0.0730219769437499	0.009924728036149189	-0.11710958254984671
MAE Error	4002.8327810662854	3846.6264309277353	3959.326160703784	3843.5596807397287

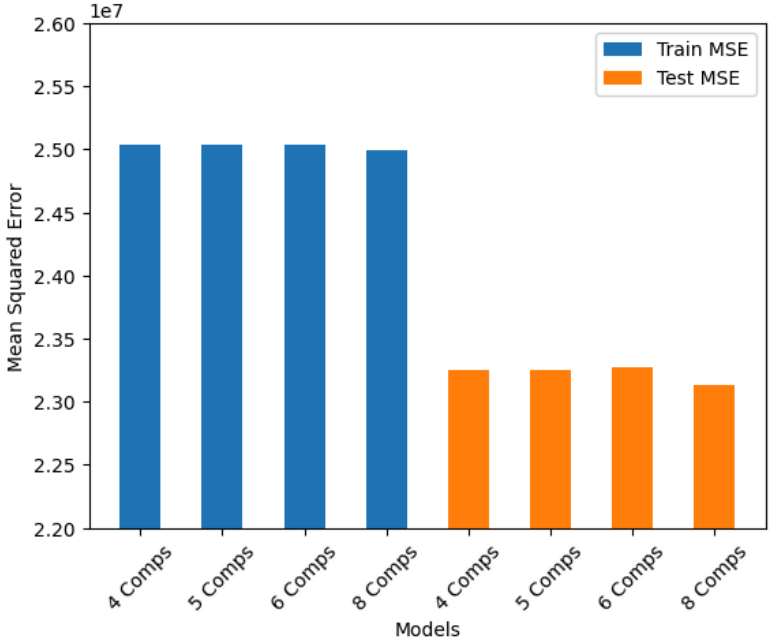
With One Hot Encoding, the MSE, the RMSE and both the R2 scores are worse than Label Encoding on the Test Set whereas they are much better for the train set. So, Label Encoding generalizes better and is the better performing model whereas One Hot Encoding memorizes the train data more.

(f) Since it isn't mentioned whether to use linear or ridge regression, I have tried for both:-
Linear:-

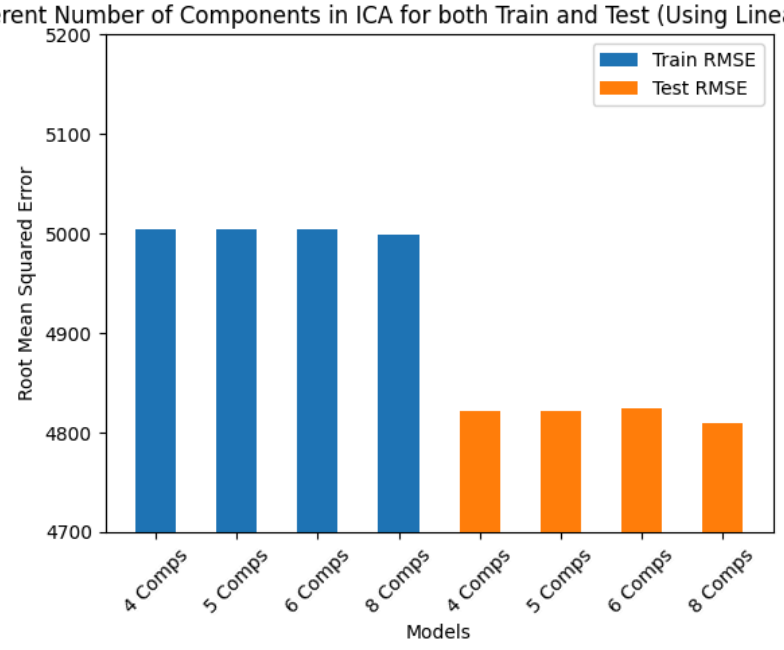
```
NO OF COMPONENTS -> 4

MSE Error -> Train : 25037304.292789903, Test : 23250324.30036188
RMSE Error -> Train : 5003.729038706023, Test : 4821.859008760198
R2 Score -> Train : 0.003077762364560921, Test : -0.003356050237547814
Adjusted R2 Score -> Train : -0.0009299652239231904, Test :
-0.019737373506732325
MAE Error -> Train : 4016.5298893470813, Test : 3830.5786896652025

NO OF COMPONENTS -> 5

MSE Error -> Train : 25037264.18551833, Test : 23251685.132673226
RMSE Error -> Train : 5003.725030966263, Test : 4822.000117448488
R2 Score -> Train : 0.003079359334843712, Test : -0.003414776271443598
Adjusted R2 Score -> Train : -0.001935332016590685, Test :
-0.023976554473727374
MAE Error -> Train : 4016.739494535791, Test : 3831.0940288782604

NO OF COMPONENTS -> 6

MSE Error -> Train : 25035442.200514406, Test : 23273726.03685368
RMSE Error -> Train : 5003.542964791489, Test : 4824.28502856679
R2 Score -> Train : 0.003151906177180086, Test : -0.004365940404746471
Adjusted R2 Score -> Train : -0.002871345145012194, Test :
-0.029165099427085783
MAE Error -> Train : 4017.005756825942, Test : 3833.582403103146

NO OF COMPONENTS -> 8
```

```
MSE Error -> Train : 24989124.36917531, Test : 23131201.918811798
RMSE Error -> Train : 4998.912318612451, Test : 4809.490817000465
R2 Score -> Train : 0.004996165268374875, Test : 0.0017846162195267468
Adjusted R2 Score -> Train : -0.003036156303626081, Test :
-0.03135116415492867
MAE Error -> Train : 4019.6788938033897, Test : 3826.440385501537
```

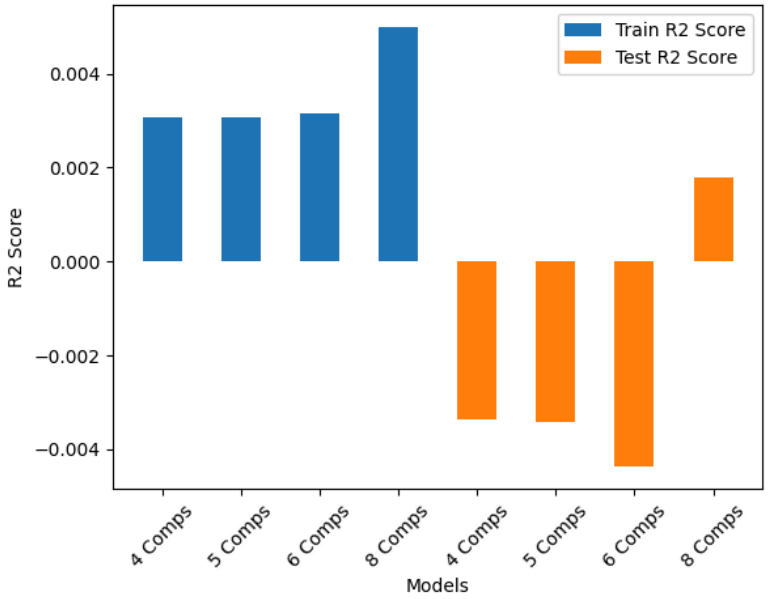
MSEs for Different Number of Components in ICA for both Train and Test (Using Linear Regression as Model)



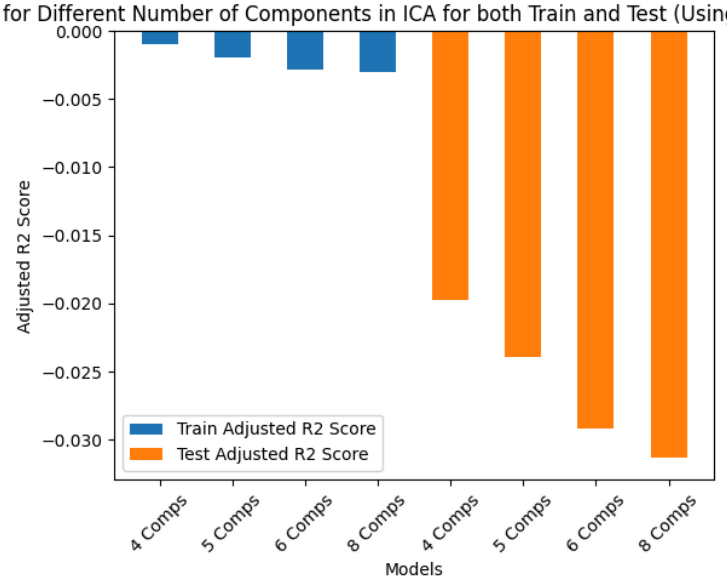
RMSEs for Different Number of Components in ICA for both Train and Test (Using Linear Regression as Model)

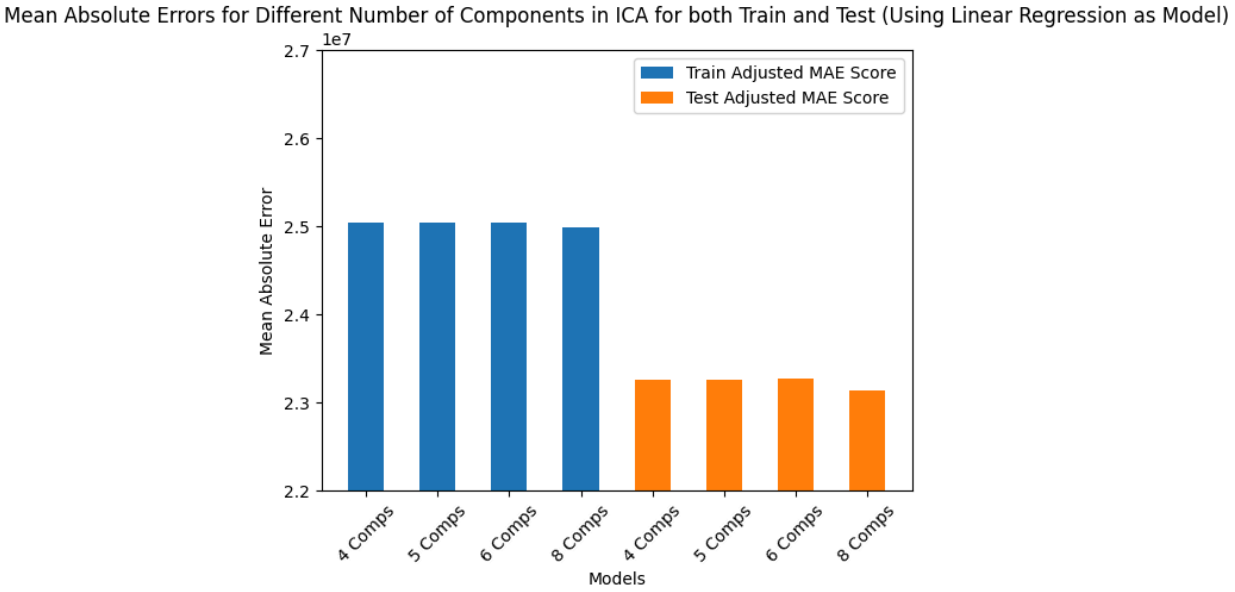


R2 Scores for Different Number of Components in ICA for both Train and Test (Using Linear Regression as Model)



Adjusted R2 Scores for Different Number of Components in ICA for both Train and Test (Using Linear Regression as Model)



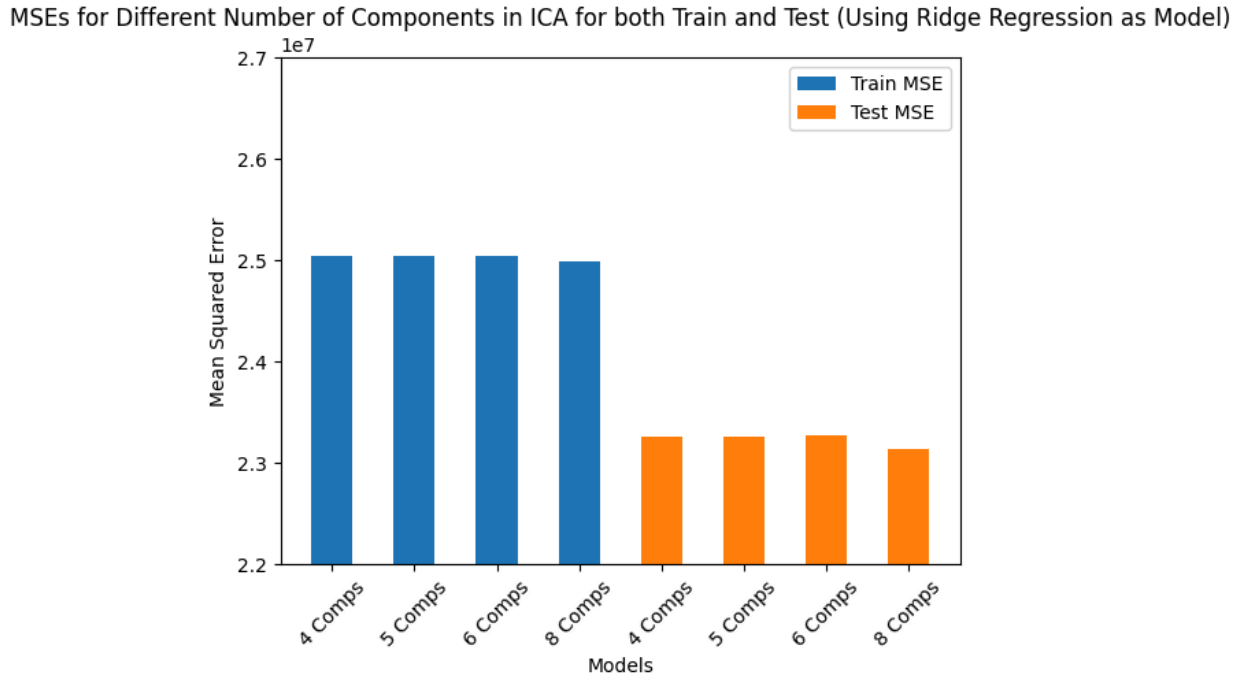


ICA with 8 components has the least errors and best R2 score for test set, but doesn't have a good adjusted R2 score. 4 components has the best adjusted R2 score. For the train set, it is also 8 components that has the least errors and best R2 score but 4 components has the best adjusted R2 score.

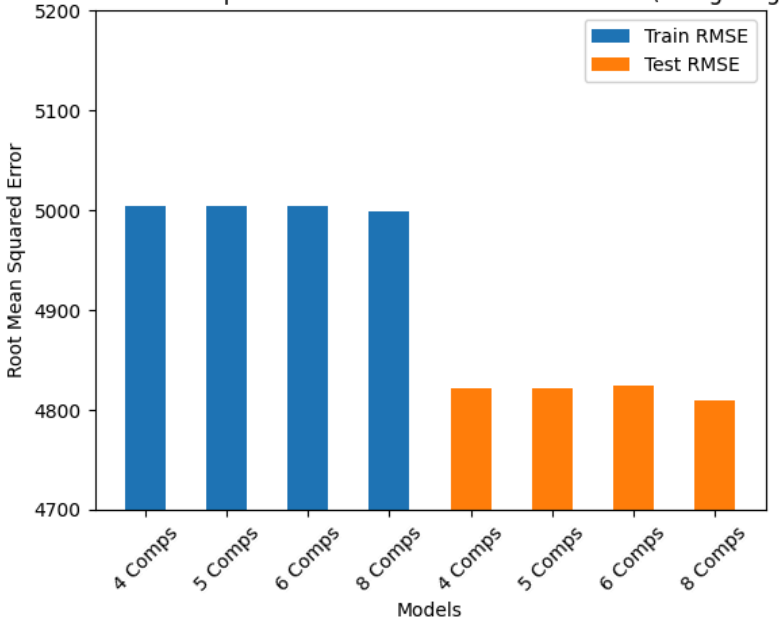
Ridge Regression:-

```
NO OF COMPONENTS -> 4

MSE Error -> Train : 25037304.369034607, Test : 23250171.996765994
RMSE Error -> Train : 5003.729046324812, Test : 4821.843215697291
R2 Score -> Train : 0.0030777593286892557, Test : -0.003349477652476507
Adjusted R2 Score -> Train : -0.0009299682719994262, Test :
-0.019730693614149475
MAE Error -> Train : 4016.5216111919294, Test : 3830.553651950557

NO OF COMPONENTS -> 5

MSE Error -> Train : 25037264.262147732, Test : 23251528.586014874
RMSE Error -> Train : 5003.725038623498, Test : 4821.983884877144
R2 Score -> Train : 0.003079356283654633, Test : -0.0034080205791455764
Adjusted R2 Score -> Train : -0.0019353350831279315, Test :
-0.02396966034511161
MAE Error -> Train : 4016.731240497645, Test : 3831.067659301856

NO OF COMPONENTS -> 6

MSE Error -> Train : 25035442.27885859, Test : 23273545.11039176
RMSE Error -> Train : 5003.54297262036, Test : 4824.266276895561
R2 Score -> Train : 0.003151903057712402, Test : -0.0043581326142885235
```

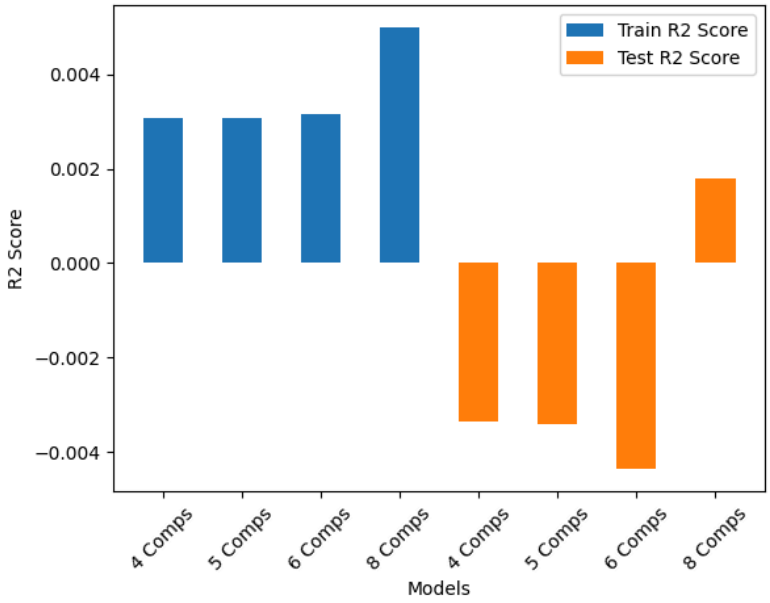
```
Adjusted R2 Score -> Train : -0.0028713482833286896, Test :
-0.029157098851678498
MAE Error -> Train : 4016.9987069661056, Test : 3833.5535275298853

NO OF COMPONENTS -> 8

MSE Error -> Train : 24989124.490464494, Test : 23131121.36519101
RMSE Error -> Train : 4998.912330744009, Test : 4809.482442549407
R2 Score -> Train : 0.004996160438946018, Test : 0.0017880924705133694
Adjusted R2 Score -> Train : -0.0030361611720413073, Test :
-0.031347572509718535
MAE Error -> Train : 4019.665441711154, Test : 3826.420420817365
```



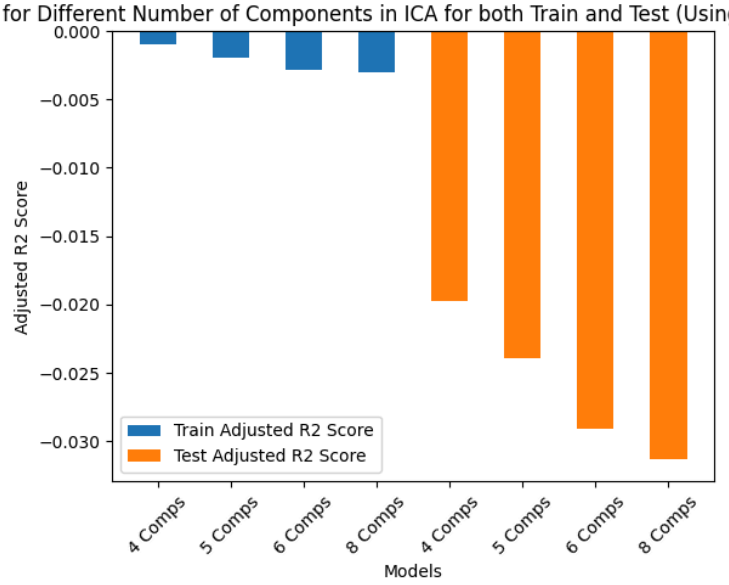
RMSEs for Different Number of Components in ICA for both Train and Test (Using Ridge Regression as Model)



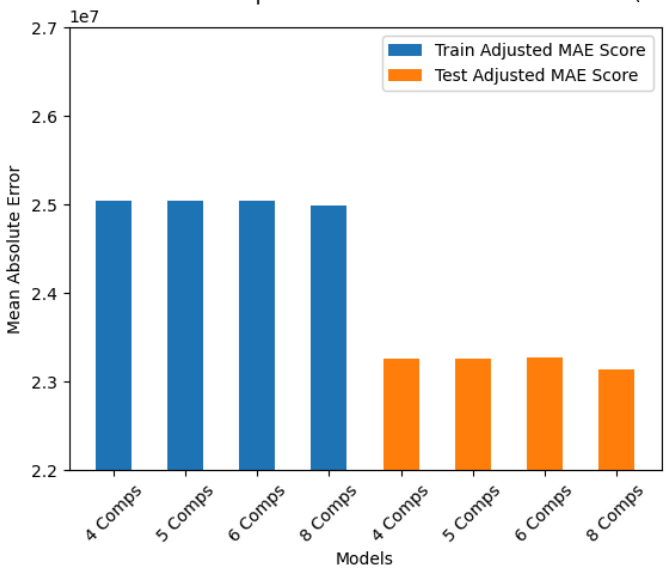
R2 Scores for Different Number of Components in ICA for both Train and Test (Using Ridge Regression as Model)



Adjusted R2 Scores for Different Number of Components in ICA for both Train and Test (Using Ridge Regression as Model)



Mean Absolute Errors for Different Number of Components in ICA for both Train and Test (Using Ridge Regression as Model)



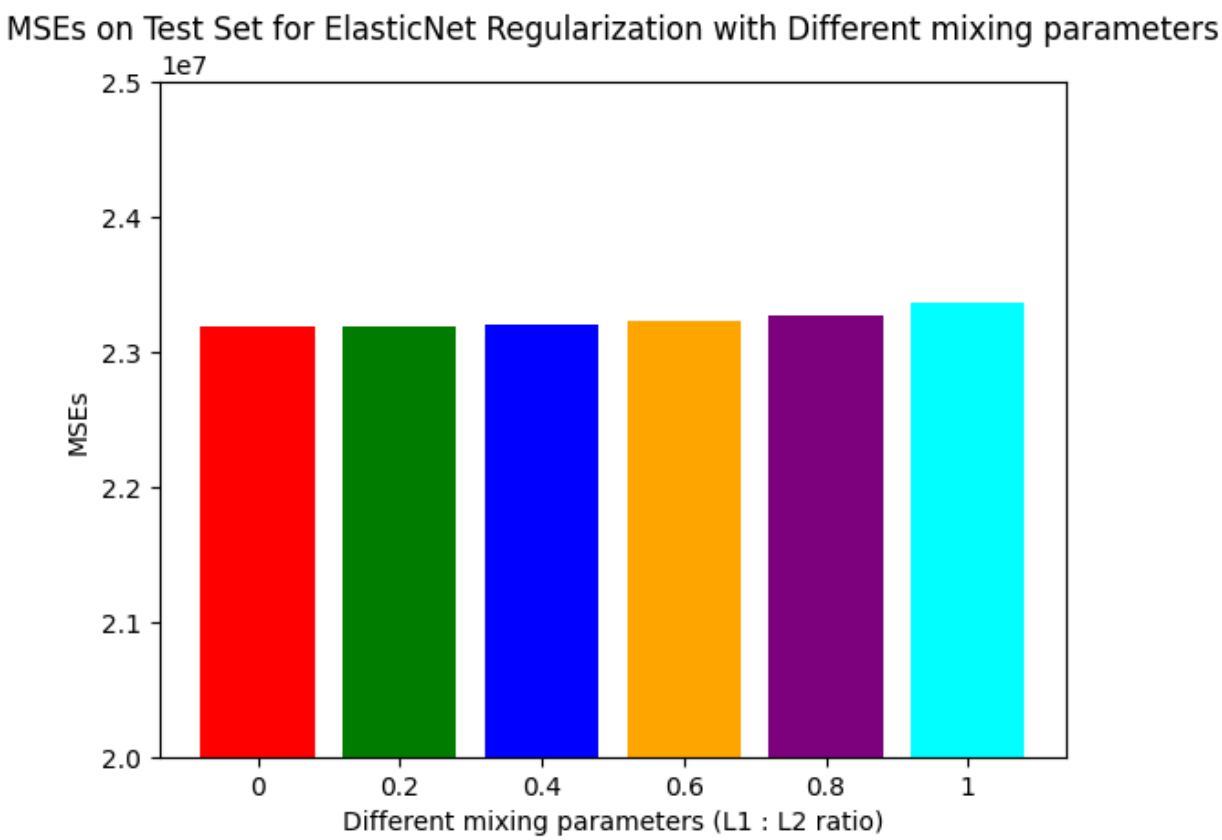
ICA with 8 components has the least errors and best R2 score for test set, but doesn't have a good adjusted R2 score. 4 components has the best adjusted R2 score. For the train set, it is also 8 components that has the least errors and best R2 score but 4 components has the best adjusted R2 score.

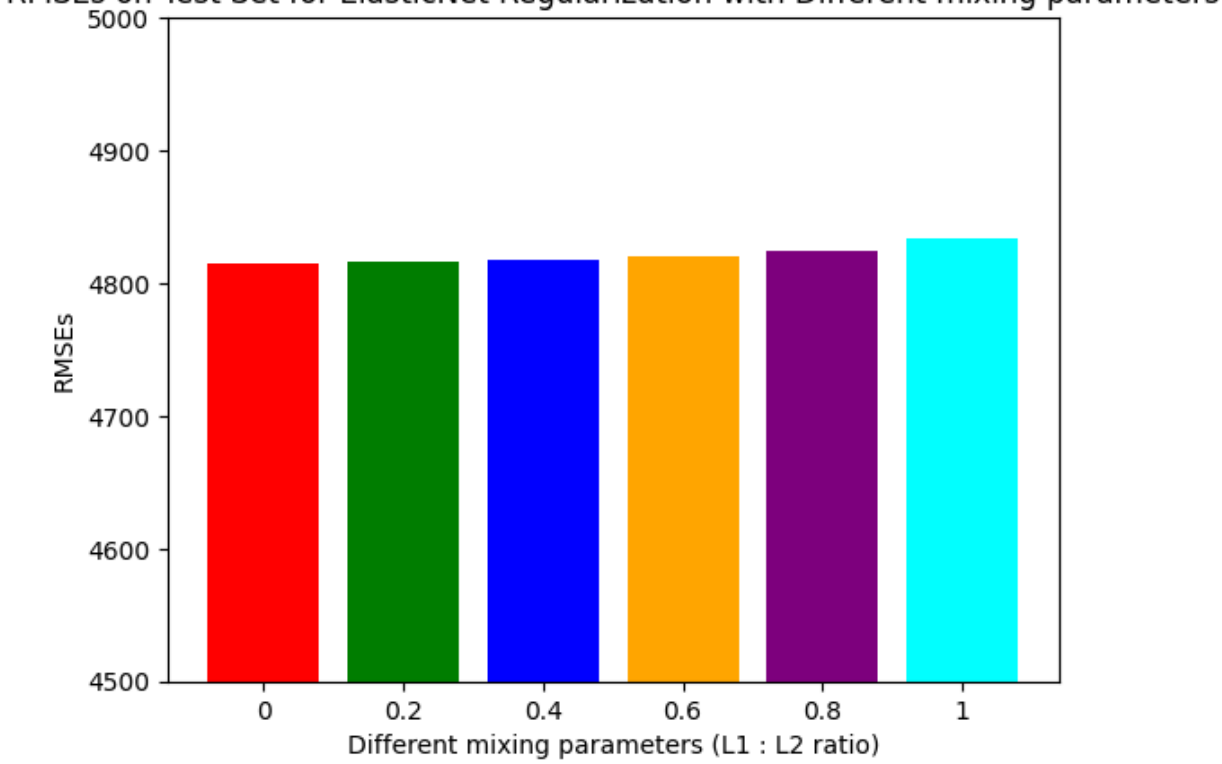
Both ridge and linear regression have very similar results and trends.

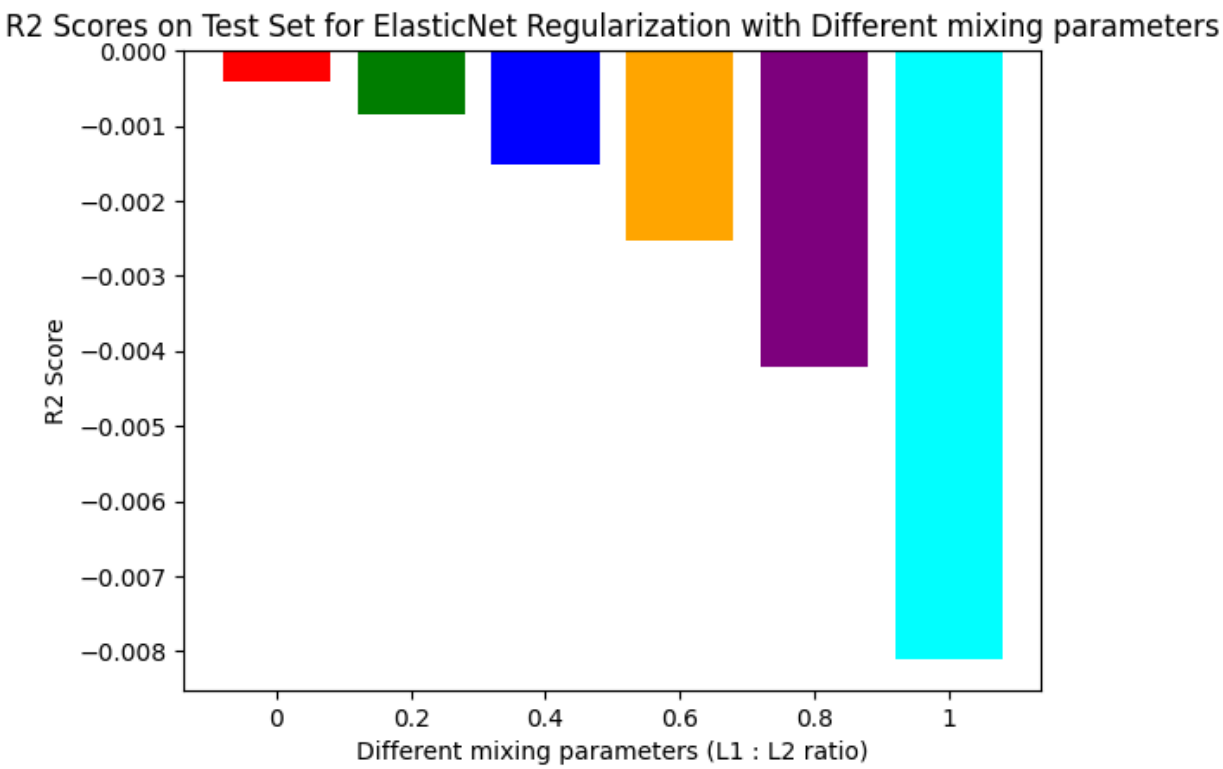
(g) `For l1 : l2 ratio of 0 : 1`

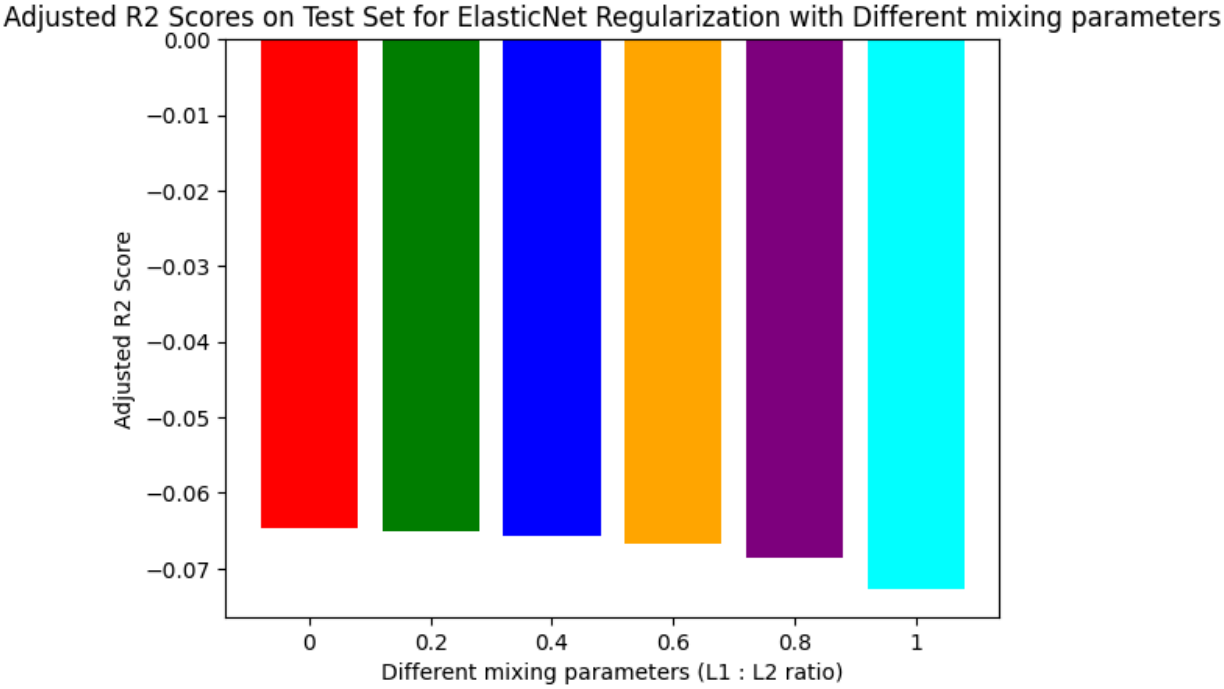
```
Test MSE Error : 23182314.233489905
Test RMSE Error : 4814.801577790086
Test R2 Score : -0.0004211100108435506
Test Adjusted R2 Score : -0.064550668344872
```

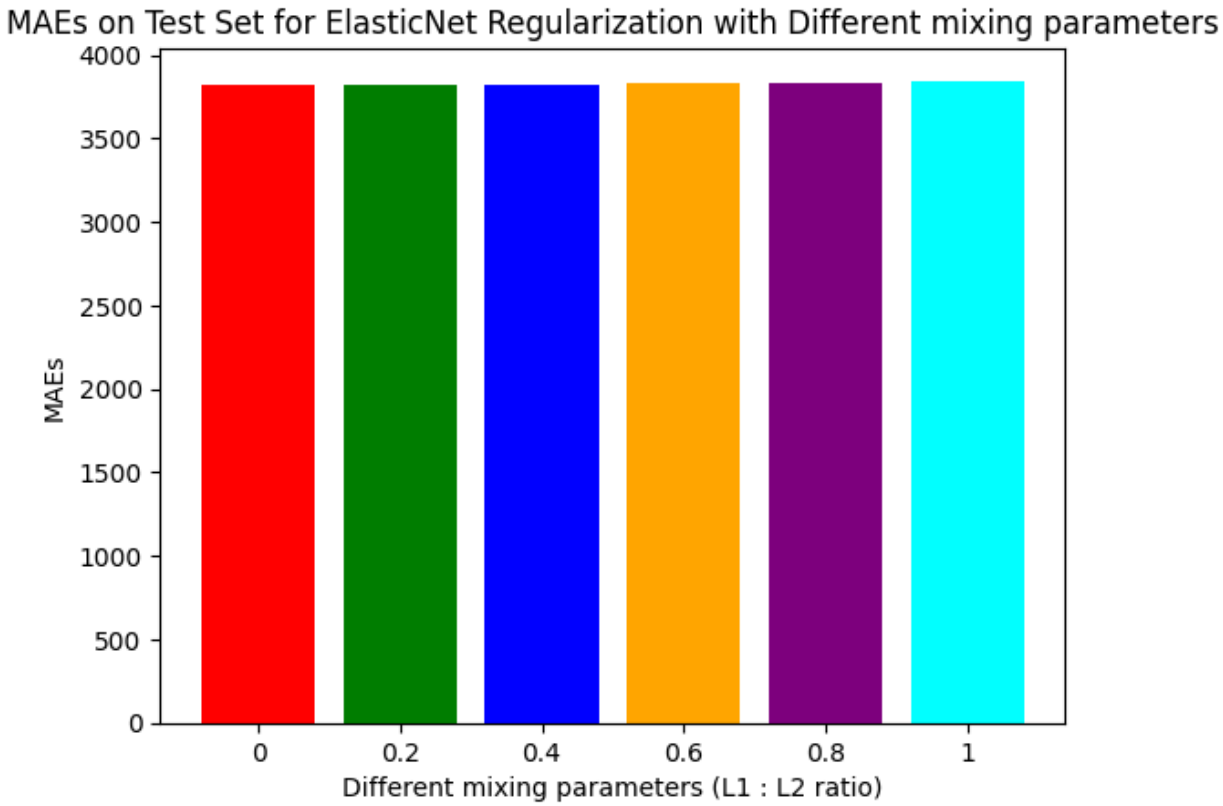
```
Test MAE Error : 3846.6264309277353

For l1 : l2 ratio of 0.2 : 0.8

Test MSE Error : 23192497.183252867
Test RMSE Error : 4815.85892476647
Test R2 Score : -0.0008605500858267501
Test Adjusted R2 Score : -0.06501827765543089
Test MAE Error : 3846.6264309277353

For l1 : l2 ratio of 0.4 : 0.6

Test MSE Error : 23207677.422006853
Test RMSE Error : 4817.434734587159
Test R2 Score : -0.0015156456535740936
Test Adjusted R2 Score : -0.06571536652880328
Test MAE Error : 3846.6264309277353

For l1 : l2 ratio of 0.6 : 0.4

Test MSE Error : 23231164.02237823
Test RMSE Error : 4819.871784848455
Test R2 Score : -0.002529198078809136
Test Adjusted R2 Score : -0.06679389026334825
Test MAE Error : 3846.6264309277353

For l1 : l2 ratio of 0.8 : 0.19999999999999996

Test MSE Error : 23270066.22851197
Test RMSE Error : 4823.905702696931
Test R2 Score : -0.004208003216648493
Test Adjusted R2 Score : -0.0685803111151515
Test MAE Error : 3846.6264309277353

For l1 : l2 ratio of 1 : 0

Test MSE Error : 23360503.511732955
Test RMSE Error : 4833.2704778165435
Test R2 Score : -0.008110778683974473
Test Adjusted R2 Score : -0.07273326449704975
Test MAE Error : 3846.6264309277353
```


MSEs on Test Set for ElasticNet Regularization with Different mixing parameters
1e7
2.5
2.4
2.3
2.2
2.1
2.0
MSEs
0
0.2
0.4
0.6
0.8
1
Different mixing parameters (L1 : L2 ratio)


RMSEs on Test Set for ElasticNet Regularization with Different mixing parameters
5000
4900
4800
4700
4600
4500
RMSEs
0
0.2
0.4
0.6
0.8
1
Different mixing parameters (L1 : L2 ratio)


R2 Scores on Test Set for ElasticNet Regularization with Different mixing parameters
R2 Score
0.000
−0.001
−0.002
−0.003
−0.004
−0.005
−0.006
−0.007
−0.008
0
0.2
0.4
0.6
0.8
1
Different mixing parameters (L1 : L2 ratio)


Adjusted R2 Scores on Test Set for ElasticNet Regularization with Different mixing parameters
Adjusted R2 Score
0.00
−0.01
−0.02
−0.03
−0.04
−0.05
−0.06
−0.07
0
0.2
0.4
0.6
0.8
1
Different mixing parameters (L1 : L2 ratio)



It is very evident from the graphs that when L1:L2 is 0:1, the best results emerge. So, when it's completely an L2 penalty and no L1 penalty is there, the model is the best. This is also called Ridge Regression.

(h) For gradient boosting:-

MSE Error -> Train : 14771770.828076074, Test : 24940708.828536887
RMSE Error -> Train : 3843.4061492478354, Test : 4994.067363235791
R2 Score -> Train : 0.41182538441231875, Test : -0.07630374428523656
Adjusted R2 Score -> Train : 0.4028593079551894, Test : -0.14529757404711074
MAE Error -> Train : 3071.687613637683, Test : 3914.8076485021365

Comparison with (c) and (g):-

Comparison of Metrics for Different Models For Train DataSet on Pre-Processed Data

Model	MSE	RMSE	R2 Score	Adjusted R2 Score	MAE Score
Linear Regression Train	24655148.257489815	4965.395075670194	0.018294250744540497	0.0033292240790609995	4002.8327810662854
Gradient Boosting Train	14771770.828076074	3843.4061492478354	0.41182538441231875	0.4028593079551894	3071.687613637683
ElasticNet_0:1_Train	24825916.55652792	4982.56124463392	0.011494688267549291	-0.0035739902649576383	4000.2301032618243
ElasticNet_0.2:0.8_Train	24802176.904793352	4980.178400900248	0.012439941256894849	-0.0026143279312622614	3999.183665291543
ElasticNet_0.4:0.6_Train	24774179.755248655	4977.366749120327	0.013554717058838928	-0.0014825585957518594	3998.221010452745
ElasticNet_0.6:0.4_Train	24740456.349794324	4973.97792011528	0.01489749791228323	-0.00011930852198083208	3997.554387889752
ElasticNet_0.8:0.2_Train	24698441.838088375	4969.75269385594	0.016570409681587295	0.0015791049511237887	3998.2915715097515
ElasticNet_1:0_Train	24655169.545732804	4965.3972193302725	0.018293403100457684	0.0033283635135744616	4002.7434544241296

Comparison of Metrics for Different Models For Test DataSet on Pre-Processed Data

Model	MSE	RMSE	R2 Score	Adjusted R2 Score	MAE Score
Linear Regression Test	23366790.69266435	4833.9208405459385	-0.008382098814608385	-0.0730219769437499	3846.6264309277353
Gradient Boosting Test	24940708.828536887	4994.067363235791	-0.07630374428523656	-0.14529757404711074	3914.8076485021365
ElasticNet_0:1_Test	23182314.233489905	4814.801577790086	-0.0004211100108435506	-0.064550668344872	3817.7507019728487
ElasticNet_0.2:0.8_Test	23192497.183252867	4815.85892476647	-0.0008605500858267501	-0.06501827765543089	3819.69943107461
ElasticNet_0.4:0.6_Test	23207677.422006853	4817.434734587159	-0.0015156456535740936	-0.06571536652880328	3822.7534232754306
ElasticNet_0.6:0.4_Test	23231164.02237823	4819.871784848455	-0.0025291980788091 36	-0.06679389026334825	3827.3777265651847
ElasticNet_0.8:0.2_Test	23270066.22851197	4823.905702696931	-0.004208003216648493	-0.0685803111151515	3834.0896522404405
ElasticNet_1:0_Test	23360503.511732955	4833.2704778165435	-0.008110778683974473	-0.07273326449704975	3846.003445204405

ElasticNet with a complete L2 penalty has the best scores and least errors for the test set, and so that is the best model. For train, gradient boosting has ridiculously good stats compared to the others but its test performance shows that it is overfitting immensely. So, ElasticNet beats it in that regard.