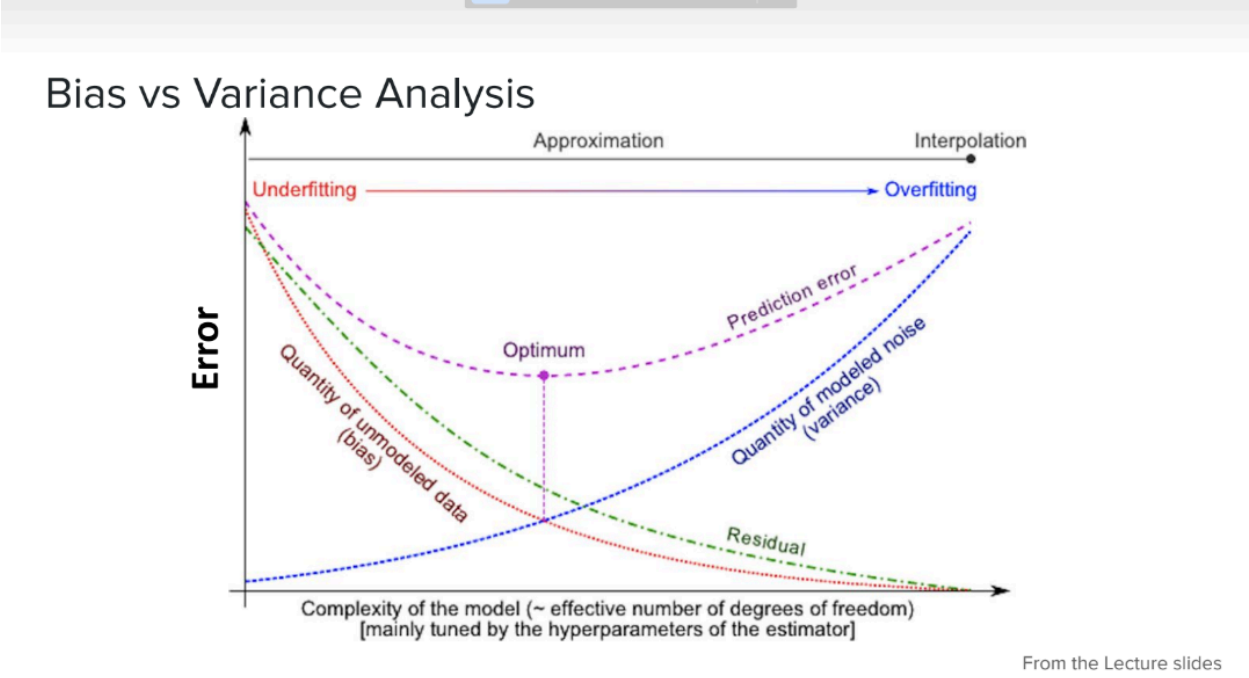
Assignment-1

Question 1:

As we increase the complexity of a machine learning model, we typically observe a characteristic pattern in terms of bias and variance. Let's see this in detail:
Bias Increases:
As we increase model complexity (e.g., adding more features or higher-order terms), the model becomes more flexible and can better fit the training data resulting in overfitting of the data.
Variance Increases:
As the model becomes more complex, it tends to fit the noise in the training data more closely.
It is true that the bias decreases with increasing model complexity. This is so that a more sophisticated model can learn the patterns in the training data more precisely by fitting the data more closely. The model learns from the training dataset extremely accurately as complexity increases. That's the exact reason the bias goes down.Nevertheless, a highly complex model frequently produces high variance when exposed to unknown data.



The image is taken from the slide as a reference,this graph effectively illustrates that as we increase model complexity, we're trading off bias for variance.

The model gets closer to fitting the training set as you increase its complexity. The model becomes more sensitive to minute variations or noise in the training set as a result, which increases variance but decreases bias and causes overfitting. The blue curve, which is labelled "Quantity of modeled noise (variance)," illustrates this and rises in complexity.

Question 2:

Given:
Emails correctly identified as spam (True Positives): 200
Spam emails incorrectly classified as legitimate (False Negative): 50
Legitimate emails correctly identified(True Negatives):730
Legitimate emails incorrectly classified as spam(False Positive):20

$$Precision = TP/(TP + FP) = 200/(200 + 20) = 0.909$$

$$Recall = TP/(TP + FN) = 200/(200 + 50) = 0.8$$

$$Accuracy = (TP + TN)/(TP + TN + FP + FN)$$
$$= (200 + 730)/(200 + 730 + 20 + 50) = 930/1000 = 0.93$$

$$F1 - Score = 2.\ (Precision.Recall)/(Precision + Recall)$$
$$= 2.(0.9091.0.8)/(0.90.91 + 0.8)$$
$$= 2 * 0.7273/1.7091 = 0.851$$

Question 3:

The equation of linear regression is: y = mx+b
Where m is the slope of the line and b is the y-intercept.
Formula to calculate m is:

$$m = (n * \sum xy - \sum x \sum y)/(n * \sum x^2 - (\sum x)^2)$$

Formula to calculate b is:

$$b = \bar{y} - m * (\bar{x})$$

$\sum x$ = 52
$\sum y$ = 285
$\sum x \sum y$= 3850
$\sum x^2$ = 694
n = 5

$\bar{y}$ = 57
$\bar{x}$ = 10.4

$$m = (5 * (3850) - (52)(285))/(5 * (694) - (52)^2)$$
$$m = (19250 - 14820)/(3470 - 2704)$$
$$m = 5.78$$

$$b = \bar{y} - m * (\bar{x})$$
$$b = 57 - 5.78 * 10.4$$
$$b = -3.112$$

When $x = 12$ then y is

y = 5.78*12 - 3.112 = 69.36 - 3.112 = 66.248

Question 4:

Hours studied (x_1)	Hours squared (x_2)	Pass or fail
2	4	0
4	16	0
6	36	1
7	49	1

If hours > 5 then pass otherwise fail

Overfitting logistic regression model.

$$P(Y=1 \mid x_1, x_2) = \frac{1}{1+e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

$w_0 = -15$, $w_1 = 2$, $w_2 = 0.5$

for $x_1 = 2$

$$P(Y=1 \mid 2,4) = \frac{1}{1+e^{-(-15 + (2)(2) + 0.5(4))}} \approx \frac{1}{1+e^{-(-7)}} \times 0.0$$

$$\approx 0.009$$

Predicts $Y = 0$

Testing set (overfitting issue)

x_1	x_2	y
5	25	0

$$P(Y=1 \mid 5, 25) = \frac{1}{1+e^{-(z)}} = \frac{1}{1+e^{-25}} \approx 0.924$$

$z = (-15 + 25 + 0.5 x$

Model Predict $Y=1$ which is wrong it should Predict Predict $y = 0$

X_1	X_2
150	50
160	60
170	70

eg. $y = B_0 + B_1 \cdot X$, $B_0 = -57$, $B_1 = 0.75$

Now for training set

$y = -57 + 0.75 \times 150$

$y = 55.5$ close to actual weight

$y = -57 + 0.75 \times 160$

$= 63$ close to actual weight

Now on testing set

190	90

$y = -57 + 0.75 \times 190 = 85.5$ close enough.

Section C

a) I conducted preliminary data analysis on the dataset and plotted various graphs for features. Firstly, I described the data using the describe function available under pandas library; this gave various statistical results about the dataset, which are as follows:-

	Construction_Year	Number_of_Floors	Energy_Consumption_Per_SqM
count	1250.000000	1250.000000	1250.000000
mean	1996.735200	7.118400	108.745712
std	15.696515	3.773456	72.180593
min	1970.000000	1.000000	50.000000
25%	1983.000000	4.000000	50.000000
50%	1996.000000	7.000000	72.868797
75%	2010.000000	11.000000	154.658612
max	2023.000000	13.000000	250.000000

	Water_Usage_Per_Building	Waste_Recycled_Percentage	Occupancy_Rate
count	1250.000000	1250.000000	1250.000000
mean	647.021156	28.831580	77.285182
std	485.194992	16.250126	15.093523
min	200.000000	0.908321	17.600798
25%	200.000000	15.948639	68.047227
50%	451.638391	26.828853	80.355416
75%	1013.701569	39.807403	88.977348
max	1500.000000	80.886835	99.923707

	Indoor_Air_Quality	Smart_Devices_Count	Green_Certified
count	1250.000000	1250.000000	1250.000000
mean	40.517956	5.244000	0.237600
std	33.705393	3.248524	0.425783

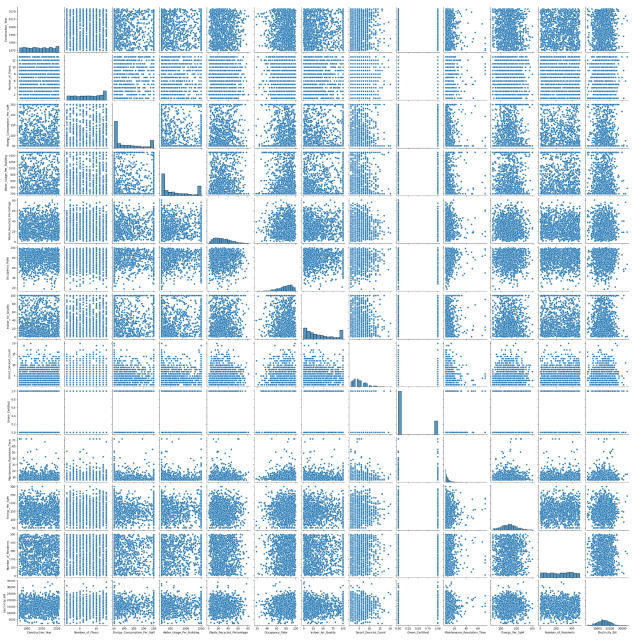
	Maintenance_Resolution_Time	Energy_Per_SqM	Number_of_Residents
count	1250.000000	1250.000000	1250.000000
mean	9.617933	152.156367	251.764800
std	6.874172	48.567837	144.082406
min	5.000417	50.000000	1.000000
25%	5.942027	118.117412	127.250000
50%	7.607172	152.234583	261.000000
75%	10.514598	183.061704	375.750000
max	72.000000	300.000000	498.000000

	Electricity_Bill
count	1250.000000
mean	15089.459765
std	4974.536190
min	2000.000000
25%	11690.482925
50%	15148.928890
75%	18471.215980
max	33887.449880

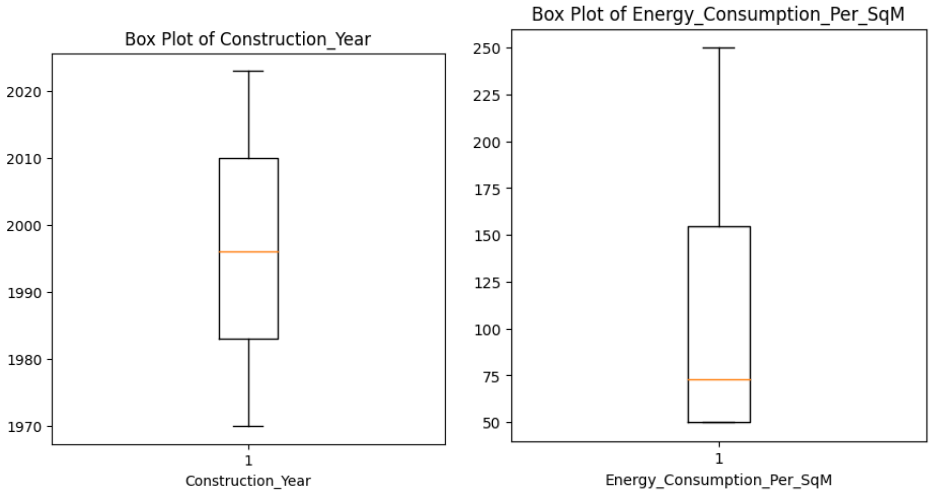
After this I found out the various features and their data types so that we could plot the data accordingly, i.e- count plot for categorical features and other plot for numerical features. Below is the output:-

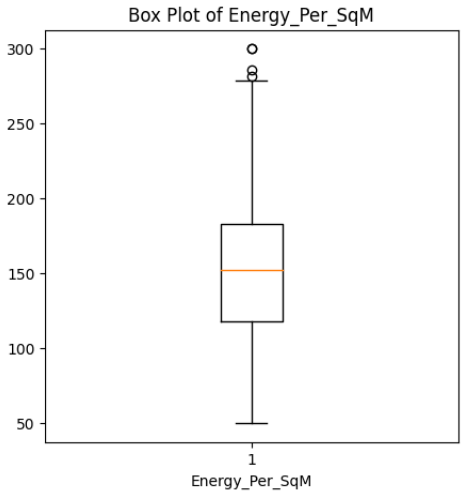
```
RangeIndex: 1250 entries, 0 to 1249
Data columns (total 16 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Building_Type               1250 non-null   object
 1   Construction_Year           1250 non-null   int64
 2   Number_of_Floors            1250 non-null   int64
 3   Energy_Consumption_Per_SqM  1250 non-null   float64
 4   Water_Usage_Per_Building    1250 non-null   float64
 5   Waste_Recycled_Percentage   1250 non-null   float64
 6   Occupancy_Rate              1250 non-null   float64
 7   Indoor_Air_Quality          1250 non-null   float64
 8   Smart_Devices_Count         1250 non-null   int64
 9   Green_Certified             1250 non-null   int64
 10  Maintenance_Resolution_Time 1250 non-null   float64
 11  Building_Status             1250 non-null   object
 12  Maintenance_Priority        1250 non-null   object
 13  Energy_Per_SqM              1250 non-null   float64
 14  Number_of_Residents         1250 non-null   int64
 15  Electricity_Bill            1250 non-null   float64
dtypes: float64(8), int64(5), object(3)
memory usage: 156.4+ KB
```

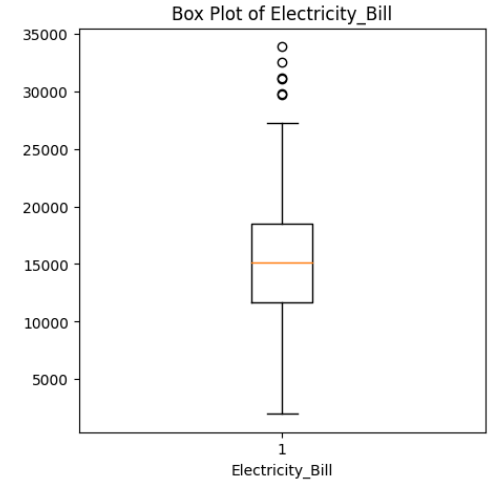
After this I plotted the various graph for each feature, below is the output:-
Pairplot:-

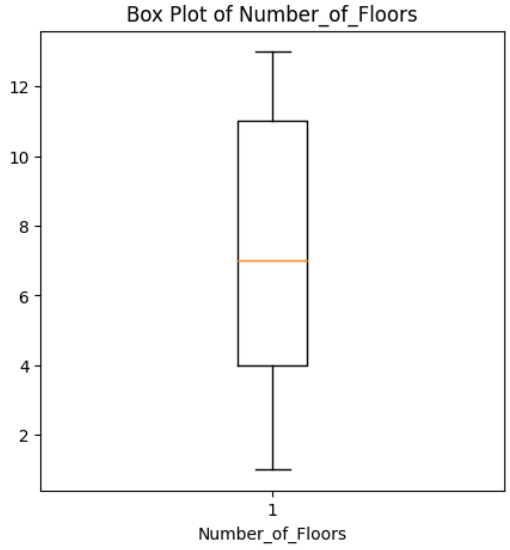


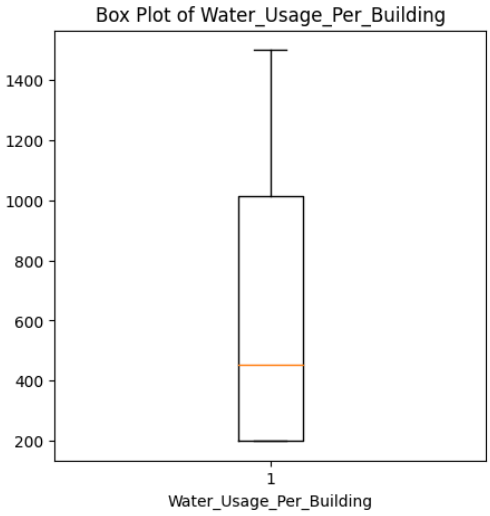
Box plot (Only for numerical features):-





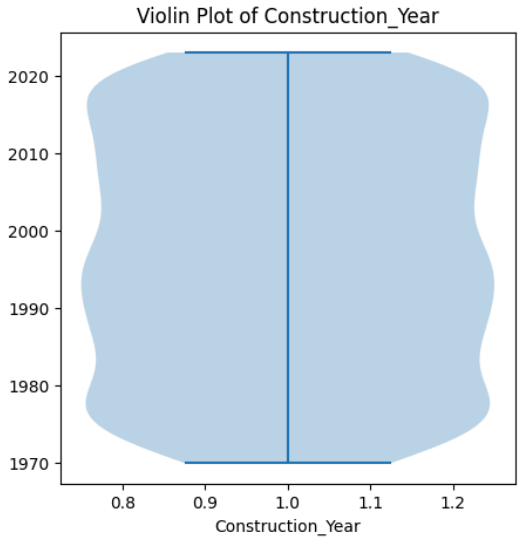


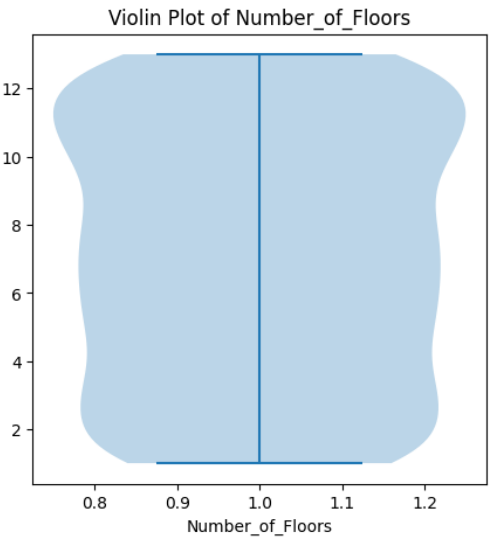


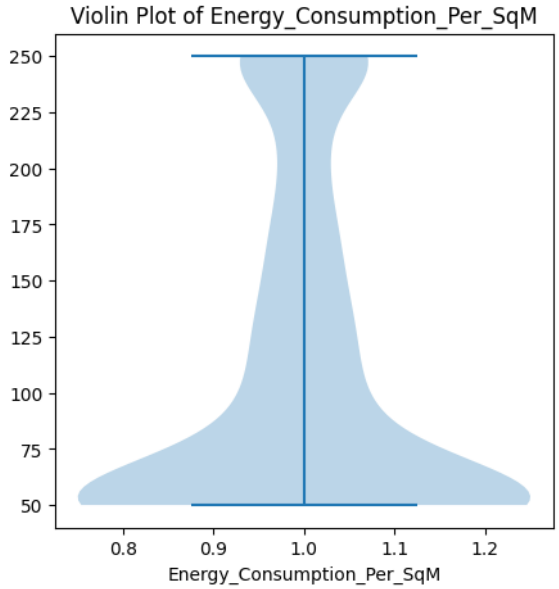


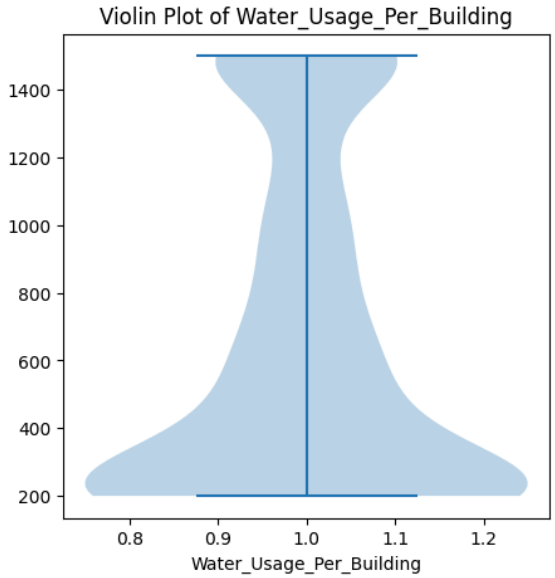
(There are more graphs for this which are in the graphs folder)

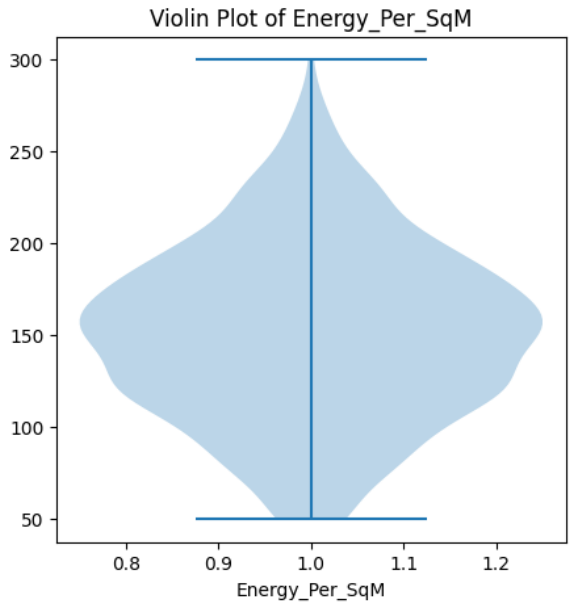
Violin Plot (only for numerical features):-

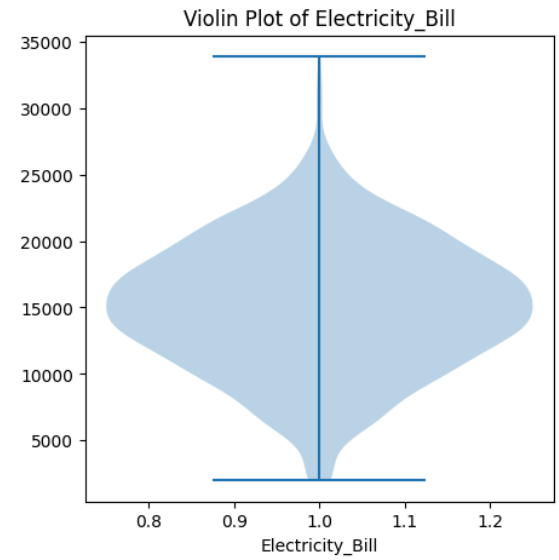






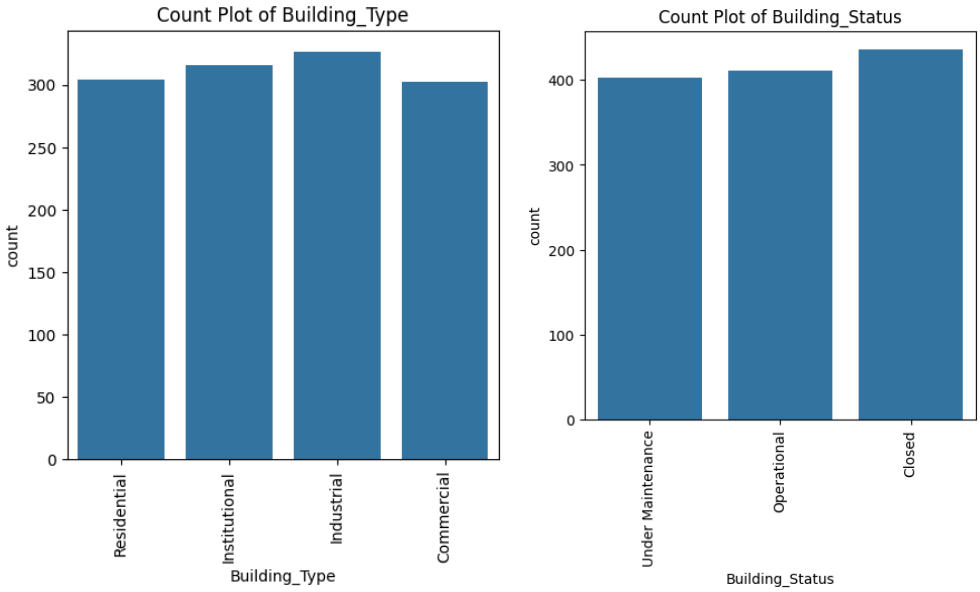


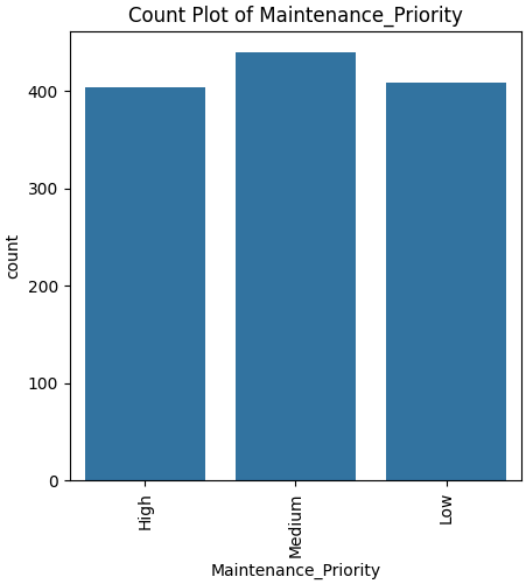




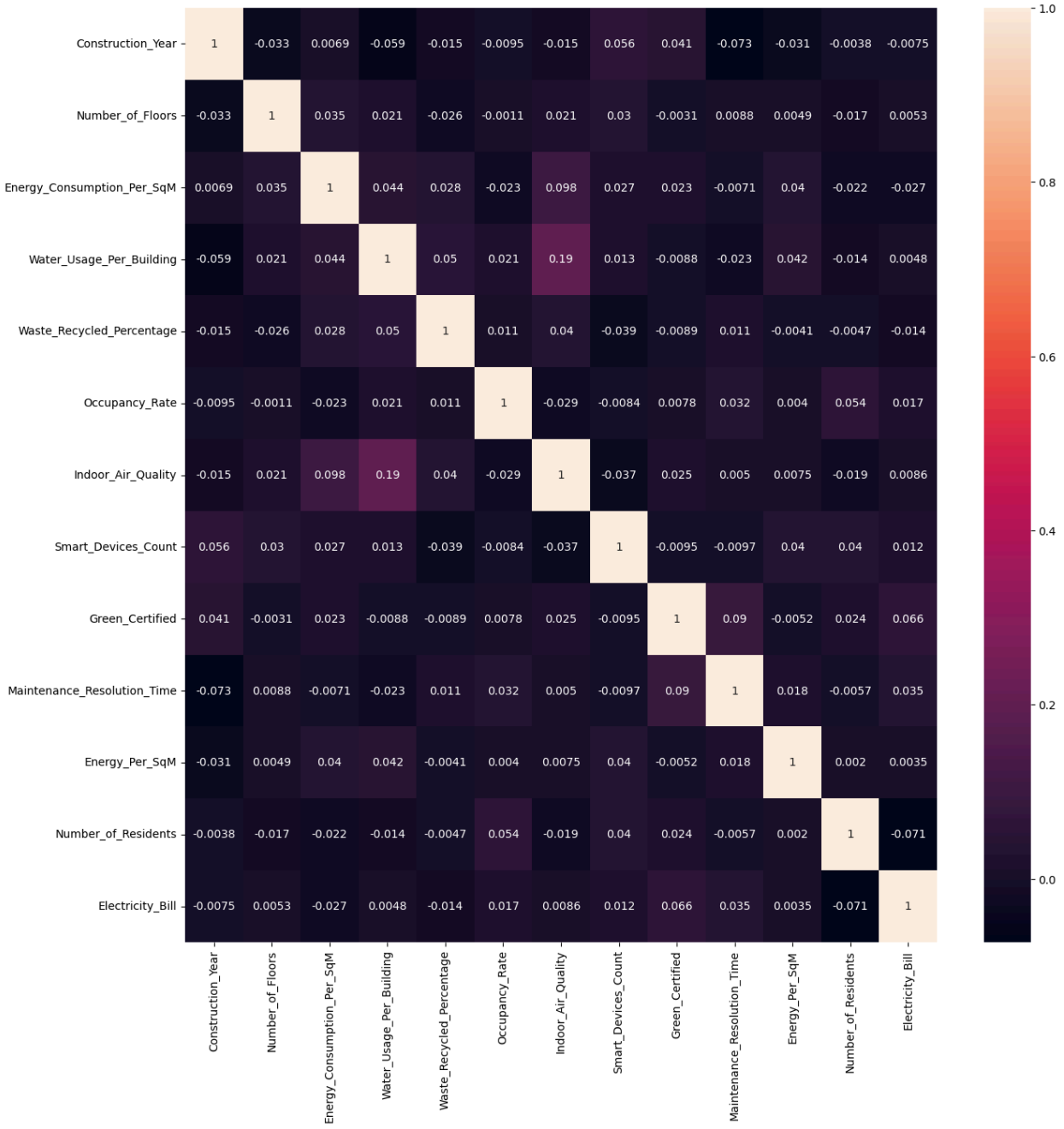
(There are more graphs for this, which are in the graphs folder)

Count plot (only for categorical features):-





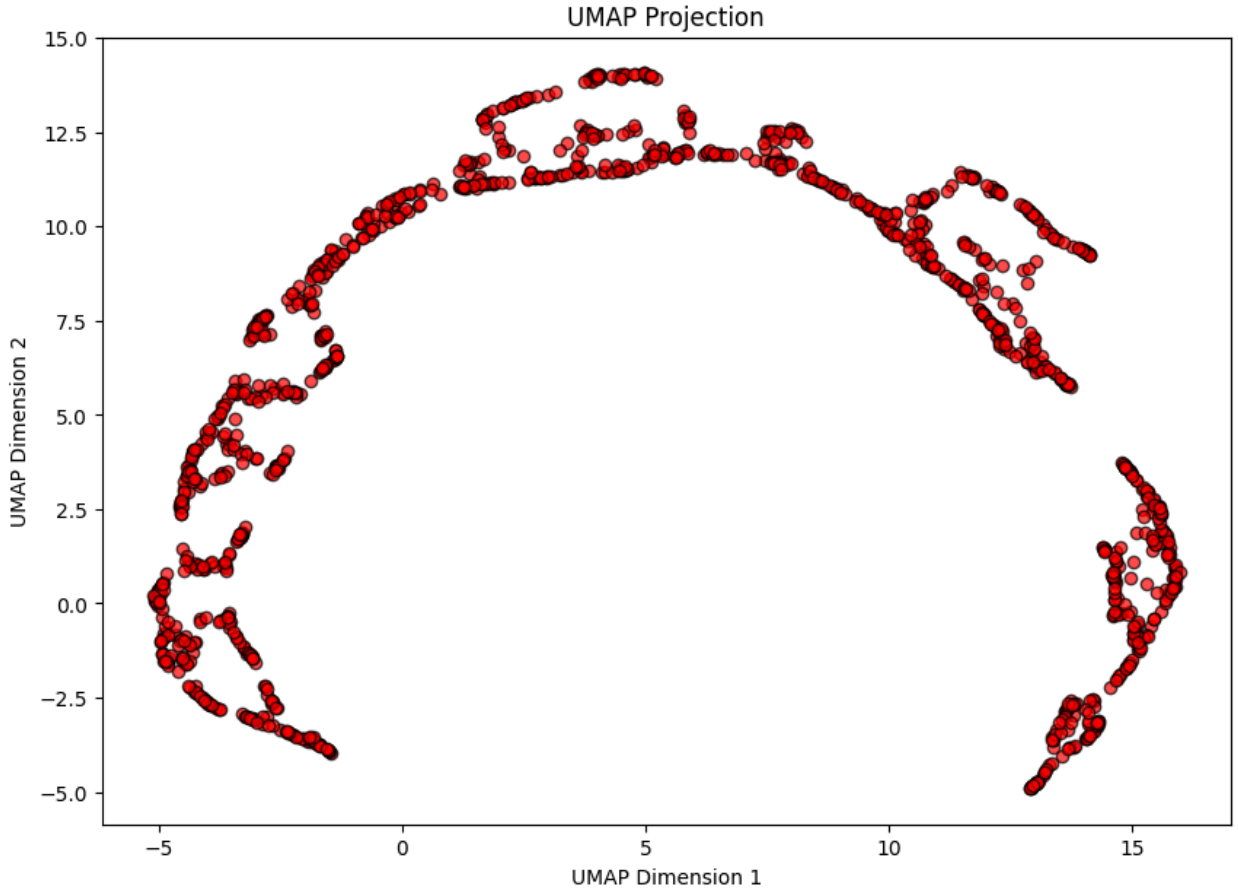
Correlation Heatmap:-



Five Insights from these graphs:-

1) The maximum number of building were from Industrial Category.
2) There is no as such significant difference between buildings under maintenance and operational (Both have almost same count).
3) Maximum Buildings have medium maintenance priority, with low and high maintenance priority being almost the same.
4) There are no strong positive correlations (close to +1) in the heatmap, but the highest correlation which is 0.19, is between Water_Usage_Per_Building and Indoor_Air_Quality. While not a strong relationship, it indicates that there might be a slight link between the two.
5) Energy_Per_SqM shows a more spread-out distribution in violin plot than Energy_Consumption_Per_SqM, which means that there's more variability in how energy is utilized per square meter than consumed.

b) Using UMAP algorithm to reduce the data dimension to 2, below is the scatter plot generated:-



The overall structure of the curve resembles a partial circle; this non-linear structure suggests that the original data has intricate relationships that simple linear methods might not capture well. Also, the UMAP has preserved some structure in the data, as shown by the non-random distribution of points.
There are no clearly separated, distinct clusters visible. Instead, the data points form a continuous curve with varying densities along its length and some areas have higher density which might represent subgroups or related data points.
For separability, there are no clear gaps between groups.

c) After loading the dataset into a dataframe, I found out whether there are any null values present for any feature along with this, i also checked for unique values for each feature. Below is the output for how many null values are there for each feature:-

```
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
```

We can easily infer from this that there are no null values for any of the features. Hence we do not have to handle any missing values for any of the features. Along with this, the following is the output for the number of unique values of each feature:-

```
Building_Type                   4
Construction_Year              54
Number_of_Floors               13
Energy_Consumption_Per_SqM     624
Water_Usage_Per_Building       710
Waste_Recycled_Percentage      1250
Occupancy_Rate                 1250
Indoor_Air_Quality             1080
Smart_Devices_Count            19
Green_Certified                2
Maintenance_Resolution_Time    1247
Building_Status                3
Maintenance_Priority           3
Energy_Per_SqM                 1229
Number_of_Residents            460
Electricity_Bill               1244
```

This clearly shows that Building_Type, Building_Status, Maintenance_Priority, and Green certified are categorical values.
After this, I used label encoding to convert the categorical values to numerical values, and after that the data became like this, (5 data samples):-

	Building_Type	Construction_Year	Number_of_Floors	\
0	3	1989	12	
1	2	1980	6	
2	1	2006	10	
3	0	1985	1	
4	1	2006	12	

	Energy_Consumption_Per_SqM	Water_Usage_Per_Building	\
0	50.000000	477.681762	
1	225.759107	200.000000	
2	98.755927	200.000000	
3	68.470692	200.000000	
4	50.000000	347.093644	

	Waste_Recycled_Percentage	Occupancy_Rate	Indoor_Air_Quality	\
0	48.417266	81.814110	13.780635	
1	46.405516	45.109000	10.177771	
2	17.247585	48.526225	27.757026	
3	57.719480	76.335921	1.519812	
4	26.936862	96.511319	1.874753	

	Smart_Devices_Count	Green_Certified	Maintenance_Resolution_Time	\
0	6	0	7.211768	
1	10	0	34.139492	
2	2	1	6.224612	
3	2	0	5.327398	
4	19	0	7.060854	

	Building_Status	Maintenance_Priority	Energy_Per_SqM	Number_of_Residents	\
0	2	0	174.835708	360	
1	2	0	143.086785	91	
2	1	2	182.384427	115	
3	0	1	226.151493	159	
4	0	2	138.292331	258	

	Electricity_Bill
0	16408.008770
1	15230.425070
2	8117.632795
3	13030.707160
4	25848.853740

After doing all of this, I trained the model on preprocessed data using linear regression, and the following were the different error metrics for training and testing set.

```
---------------Training Metrics---------------
Mean Squared Error (MSE):  24243994.339784507
Root Mean Squared Error (RMSE):  4923.819080732405
R2 Score:  0.018234994807995286
Adjusted R2 score:  0.0032690648508001408
Mean Absolute Error (MAE):  3955.449072991739


---------------Testing Metrics---------------
Mean Squared Error (MSE):  25173787.24730298
Root Mean Squared Error (RMSE):  5017.348627243574
R2 Score:  -0.013124586582368 37
Adjusted R2 score:  -0.07806847033764841
Mean Absolute Error (MAE):  4047.42434750498
```

This shows that model is underfitting the data, meaning it is not complex enough to capture the relationships between the features and the target variable.
The low R^2 scores, high errors, and negative adjusted R^2 score in the testing set suggest the model is not performing well on either the training or the testing sets.

d) Doing the correlation analysis on the dataset, I found the top 3 features- :
['Number_of_Residents', 'Green_Certified', 'Building_Type']

```
Top 3 features based on correlation analysis: Index(['Number_of_Residents', 'Green_Certified', 'Building_Type']
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1250 entries, 0 to 1249
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Number_of_Residents  1250 non-null   int64
 1   Green_Certified      1250 non-null   int64
 2   Building_Type        1250 non-null   int32
dtypes: int32(1), int64(2)
memory usage: 24.5 KB
```

Training the linear regression on these features gave the following error metrics:-

```
---------------Training Metrics---------------
Mean Squared Error (MSE):  24389721.059852477
Root Mean Squared Error (RMSE):  4938.595049186811
R2 Score:  0.0123337645041518
Adjusted R2 score:  0.0093588662044655 77
Mean Absolute Error (MAE):  3966.5451872698636


---------------Testing Metrics---------------
Mean Squared Error (MSE):  24647007.851224776
Root Mean Squared Error (RMSE):  4964.575294143979
R2 Score:  0.0080757656979193 46
Adjusted R2 score:  -0.0040208713057645 62
Mean Absolute Error (MAE):  3992.2211193795347
```

Using the correlation analysis and selecting the top most 3 features and training the model on these features performed slightly better in terms of generalization to the test as compared to the part(c) model, this is evident from the from the positive R^2 score and lower testing MSE, RMSE, and MAE. It also avoids the negative R^2 score observed in the part(c) model.
For training performance, the part(c) model has a slightly better fit to the training data, but the difference is minimal. Reducing the model complexity by using only the top 3 features from correlation analysis leads to better performance on unseen test data.
Thus, feature selection based on correlation has helped improve the generalization of the model without sacrificing much in terms of training performance.

e) After encoding the categorical features using one-hot encoding and then performing ridge regression over the preprocessed data, gave the below error metrics:-

```
---------------Training Metrics---------------
Mean Squared Error (MSE):  24109316.890044343
Root Mean Squared Error (RMSE):  4910.123917992737
R2 Score:  0.023688782879807 47
Adjusted R2 score:  0.0017042928320651907
Mean Absolute Error (MAE):  3926.4658779184697


---------------Testing Metrics---------------
Mean Squared Error (MSE):  24762405.30104465
Root Mean Squared Error (RMSE):  4976.183809009133
R2 Score:  0.0034315700314947 46
Adjusted R2 score:  -0.0931521544588449 5
Mean Absolute Error (MAE):  4005.879274683752
```

Comparing this with the model trained in part(c), we can easily see that its performance has increased over both the training and testing sets, earlier the MSE for part(c) was 24243994 (ignoring the fractional part) and now it is 24109316, though the MSE has decreased by 134678 which is not that big but the model performed better using ridge regression. Also, in part(c) the R2 score was 0.01 and now it is 0.02 on training set, and for testing set model has performed better in terms of R2 score as in part(c) it was negative which means model is performing worse than a simple mean-based model and now the r2 score is positive which means model performance improves.
Same thing is observed with the Mean Absolute error which is 3926 whereas earlier in part(c) it was 3955 for training data, and for testing the same pattern is observed.

f) Using Independent Component Analysis (ICA) on the one-hot encoded dataset with varying number of components, we observe the following metrics:-

	Train MSE	Train RMSE	Train R2	Train Adjusted R2	Train MAE	Test MSE	\
4	182.009640	13.491095	0.180049	0.160726	5.312063	190.352137	
5	82.697411	9.093812	0.224123	0.205839	3.634735	78.298256	
6	35.098445	5.924394	0.265952	0.248653	2.341424	34.376635	
8	13.272890	3.643198	0.355306	0.340113	1.375422	11.038195	

	Test RMSE	Test R2	Test Adjusted R2	Test MAE
4	13.796816	0.189016	0.106483	5.495518
5	8.848630	0.235892	0.158129	3.672609
6	5.863159	0.278355	0.204913	2.376111
8	3.322378	0.370842	0.306813	1.277830

The ICA regression model shows consistent improvement in performance as the number of components increases. For the training set, the MSE decreases from 182.01 (with 4 components) to 13.27 (with 8 components), while the RMSE drops from 13.49 to 3.64, reflecting reduced error in the model's predictions. The R^2 score improves from 0.18 to 0.36, indicating that the model explains more variance as components are added. The MAE also decreases from 5.31 to 1.38, suggesting that the average prediction error is significantly reduced. On the test set, similar trends are observed.These improvements in both training and testing performance show that the model generalizes well. The MAE on the test set also reduces from 5.50 to 1.28, indicating more accurate predictions as more components are used.
In conclusion, the best performance is observed when 8 components are used, as it achieves the lowest error metrics and the highest R^2 scores, suggesting this is the optimal number of components for the ICA model in this case.

g) Training model on ElasticNet regularization with different values of mixing parameters specifically: 0.001,0.01,0.1,1,2,20,0.000001; gave the below output:-

```
---------- For Alpha= 0.001 ----------
----------Training Metrics----------
Mean Squared Error (MSE):  23936256.837671585
Root Mean Squared Error (RMSE):  4892.469400790524
R2 Score:  0.013571180759088919
Adjusted R2 score:  -0.001465843924461696
Mean Absolute Error (MAE):  3953.2195573313647

----------Testing Metrics----------
Mean Squared Error (MSE):  26662996.625467606
Root Mean Squared Error (RMSE):  5163.622432504879
R2 Score:  -0.005172595960708071
Adjusted R2 score:  -0.06960673672742024
Mean Absolute Error (MAE):  4033.77479195507
---------- For Alpha= 0.01 ----------
----------Training Metrics----------
Mean Squared Error (MSE):  23936265.61577174
Root Mean Squared Error (RMSE):  4892.470297893667
R2 Score:  0.013570819007833 08
Adjusted R2 score:  -0.001466211902181696
Mean Absolute Error (MAE):  3953.213634824112
```

```
----------Testing Metrics----------
Mean Squared Error (MSE):  26661010.50907634
Root Mean Squared Error (RMSE):  5163.43011079615
R2 Score:  -0.0050977210395979355
Adjusted R2 score:  -0.06952706213187976
Mean Absolute Error (MAE):  4033.4984404358815
---------- For Alpha= 0.1 ----------
----------Training Metrics----------
Mean Squared Error (MSE):  23937065.029752858
Root Mean Squared Error (RMSE):  4892.551995610558
R2 Score:  0.013537874634164893
Adjusted R2 score:  -0.0014996577647046205
Mean Absolute Error (MAE):  3953.179615262527

----------Testing Metrics----------
Mean Squared Error (MSE):  26642937.434662502
Root Mean Squared Error (RMSE):  5161.679710584773
R2 Score:  -0.004416381301955985
Adjusted R2 score:  -0.0688020467700301
Mean Absolute Error (MAE):  4030.8716720265934
```

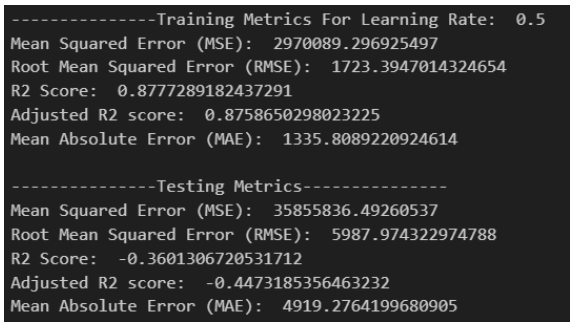
```
---------- For Alpha= 1 ----------
----------Training Metrics----------
Mean Squared Error (MSE):  23974701.962906502
Root Mean Squared Error (RMSE):  4896.39683470473
R2 Score:  0.011986831971881 55
Adjusted R2 score:  -0.003074344370010529
Mean Absolute Error (MAE):  3953.9493581634774

----------Testing Metrics----------
Mean Squared Error (MSE):  26559274.937783595
Root Mean Squared Error (RMSE):  5153.569145532404
R2 Score:  -0.0012623753830562912
Adjusted R2 score:  -0.06544586098453431
Mean Absolute Error (MAE):  4017.0727716388656
---------- For Alpha= 2 ----------
----------Training Metrics----------
Mean Squared Error (MSE):  24021557.395441048
Root Mean Squared Error (RMSE):  4901.179184180175
R2 Score:  0.010055888891326048
Adjusted R2 score:  -0.005034722536232605
Mean Absolute Error (MAE):  3955.8685156203333
```

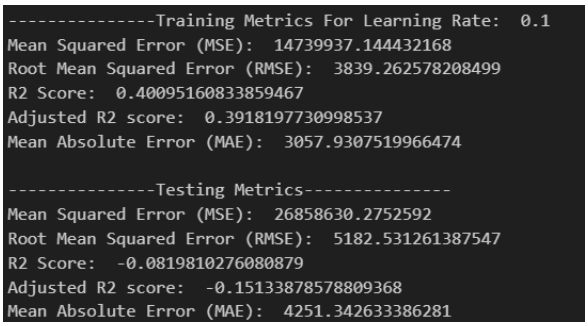
```
----------Testing Metrics----------
Mean Squared Error (MSE):  26538092.907025263
Root Mean Squared Error (RMSE):  5151.513652027456
R2 Score:  -0.0004638306004065562
Adjusted R2 score:  -0.06459612743376608
Mean Absolute Error (MAE):  4012.661752414225
---------- For Alpha= 20 ----------
----------Training Metrics----------
Mean Squared Error (MSE):  24211799.277347572
Root Mean Squared Error (RMSE):  4920.548676453427
R2 Score:  0.002215896377590565
Adjusted R2 score:  -0.012994227153238835
Mean Absolute Error (MAE):  3965.5235798092117

----------Testing Metrics----------
Mean Squared Error (MSE):  26561779.94960094
Root Mean Squared Error (RMSE):  5153.812176399228
R2 Score:  -0.001356812226248394
Adjusted R2 score:  -0.06554635147152066
Mean Absolute Error (MAE):  4007.4446117042417
```

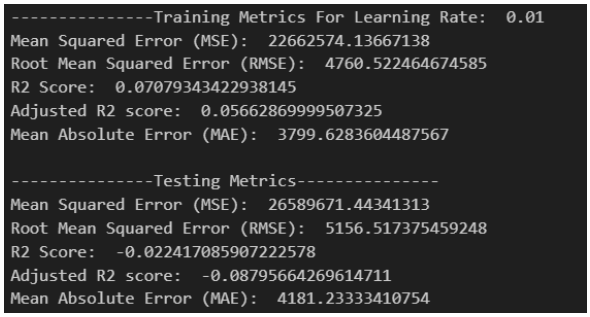
```
---------- For Alpha= 1e-06 ----------
----------Training Metrics----------
Mean Squared Error (MSE):  23936256.748147514
Root Mean Squared Error (RMSE):  4892.4693916413535
R2 Score:  0.013571184448434526
Adjusted R2 score:  -0.0014658401788758546
Mean Absolute Error (MAE):  3953.220211071582

----------Testing Metrics----------
Mean Squared Error (MSE):  26663219.25026328
Root Mean Squared Error (RMSE):  5163.643989496495
R2 Score:  -0.005180988728668012
Adjusted R2 score:  -0.06961566749332637
Mean Absolute Error (MAE):  4033.805628625759
```

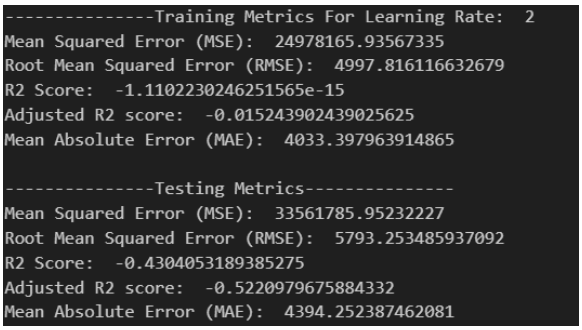
With different alpha values, the model's performance varied significantly. For alpha = 0.001, the training Mean Square Error was 23936256, and the Root Mean Square Error was 4892.47, while the R2 score was 0.0136. Testing MSE was higher at 26662219 with an RMSE of 5163.64. As alpha increased to 0.01, 0.1, and up to 20, the training MSE remained relatively stable, ranging from 23936065 to 24211799, with corresponding RMSE values between 4892.55 and 4920.55. However, testing MSE also showed a slight increase, reaching up to 26561779. The R2 scores for all alpha values were quite low, ranging from -0.005 to 0.013, indicating poor model fit. The Mean Absolute Error for training remained around 3953, while testing MAE ranged from 4007 to 4394. Overall, the model struggled with high testing errors and low R2 scores, regardless of the alpha value used.

h) I tried different values of the learning rate for the training model using Gradient Boosting Regressor, following is the output for different learning rates, specifically for 0.5,0.1,0.01,2.









With learning rate smaller like - 0.5 and 0.1 the training error was low as compared to the learning rate when it was too big or too small - 0.01 and 2. For 0.5 model had a very good R2 score on training set as compared to 0.1, but in case of training set model didn't performed well with any of the learning rates, though decreasing the learning rate from 0.1 to 0.01 seems to work as in this case the Mean absolute error decreases, but at the same time model performs badly on the training set.

Now for comparison with part(c), this model fits really good with training data as compared to model in part(c) in which MSE was 24243994 and for Gradient Boosting Regressor model it was 2970089 with learning rate as 0.5.

For comparison with part(g), the model with learning rate 0.5 fits data really good on training set in case of Gradient Boosting Regressor, having MSE as 2970089 and in part(g) the model had really large MSE compared to this.

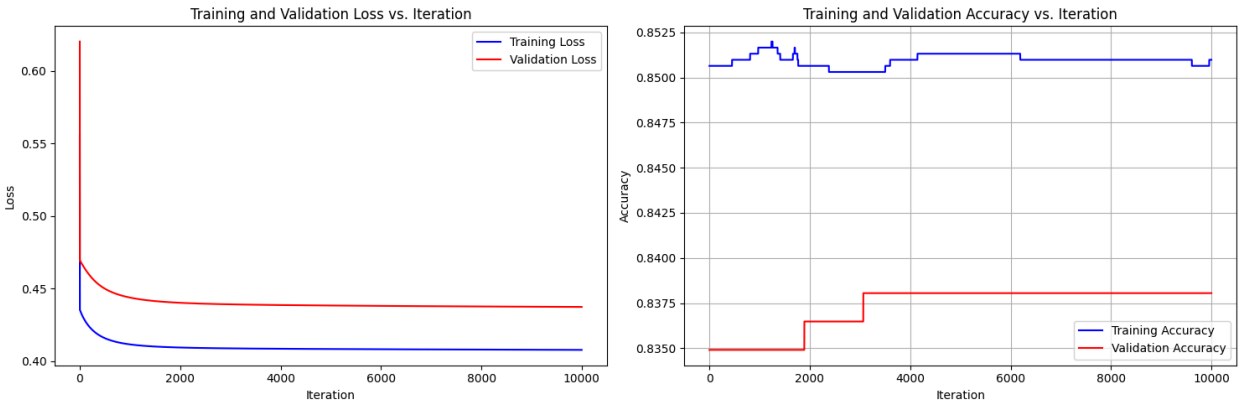
Section B

First, I handled the missing values in the dataset by filling the numeric columns with their mean and the categorical columns with their mode. Afterward, I saved the updated dataset to a new file. In the `"heart disease"` column (the target column), 84% of the actual data corresponds to `0`, indicating that 84% of the individuals in the dataset do not have heart disease.

a)When performing logistic regression, it is noticed that the model seems to converge after 5000 iterations without scaling, with training accuracy 85.09% and validation accuracy stabilizing at 83.80% when using a learning rate (alpha) of 0.0001.It is observed that with smaller alpha values, such as 0.1 and 0.01, do not yield better loss and accuracy. Instead, the best results are observed with very very small alpha values, like 0.000015 and 0.00001, which improve both loss and accuracy.
The training loss decreases gradually with each iteration, indicating that the model is learning and improving over time.

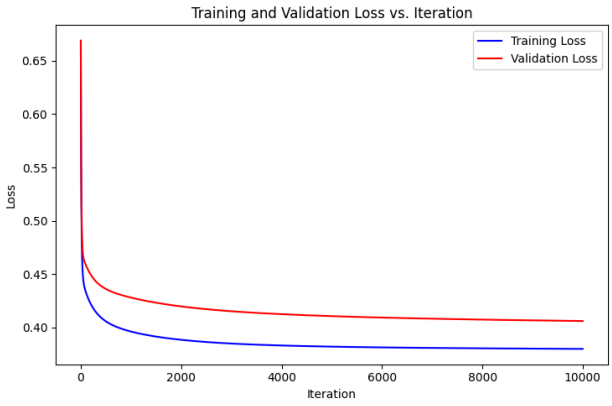
```
Iteration    0: Train Cost 0.5554, Val Cost 0.6202, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 1000: Train Cost 0.4115, Val Cost 0.4438, Train Accuracy 0.8517, Val Accuracy 0.8349
Iteration 2000: Train Cost 0.4094, Val Cost 0.4402, Train Accuracy 0.8506, Val Accuracy 0.8365
Iteration 3000: Train Cost 0.4088, Val Cost 0.4391, Train Accuracy 0.8503, Val Accuracy 0.8365
Iteration 4000: Train Cost 0.4085, Val Cost 0.4386, Train Accuracy 0.8510, Val Accuracy 0.8381
Iteration 5000: Train Cost 0.4083, Val Cost 0.4383, Train Accuracy 0.8513, Val Accuracy 0.8381
Iteration 6000: Train Cost 0.4082, Val Cost 0.4381, Train Accuracy 0.8513, Val Accuracy 0.8381
Iteration 7000: Train Cost 0.4081, Val Cost 0.4378, Train Accuracy 0.8510, Val Accuracy 0.8381
Iteration 8000: Train Cost 0.4079, Val Cost 0.4377, Train Accuracy 0.8510, Val Accuracy 0.8381
Iteration 9000: Train Cost 0.4078, Val Cost 0.4375, Train Accuracy 0.8510, Val Accuracy 0.8381
Iteration 9999: Train Cost 0.4077, Val Cost 0.4374, Train Accuracy 0.8510, Val Accuracy 0.8381
Model Accuracy On Training Data: 85.09777478084963 %
Model Accuracy On Validation Data: 83.80503144654088 %
FINAL MODEL WEIGHTS: [ 0.00923205  0.02739851 -0.02060667 -0.00419597  0.01482874  0.00367651
  0.00077823  0.02030513  0.00296728 -0.00396698  0.0241533  -0.0309826
 -0.03608884 -0.03055647  0.00361718]
FINAL BIAS: -0.007134238159601504
ALPHA: 0.0001
```

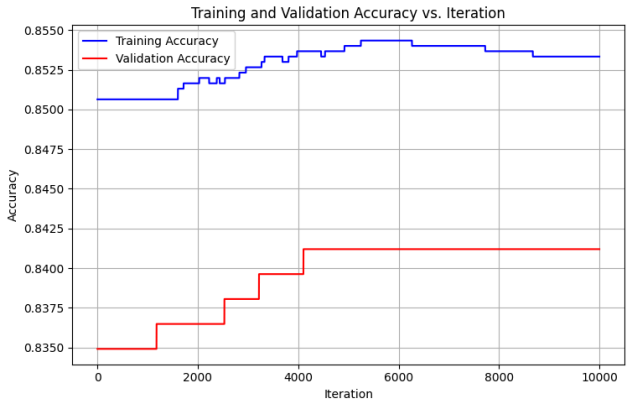
As in the below graph the training accuracy and validation accuracy are giving constant value after the iteration of 5000. A smaller learning rate (alpha = 0.0001) seems to provide stable and convergent results. Larger learning rates (e.g., 0.01 or 0.1) may cause the model to overshoot the optimal parameters or lead to divergence, as seen from the ineffective performance with higher learning rates.



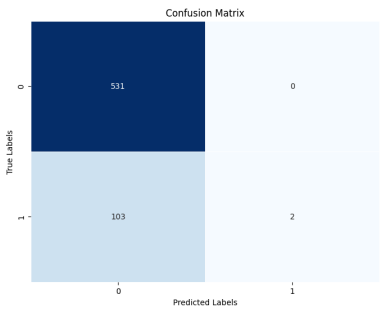
b) With scaled features with min-max scaling , the optimization landscape is more uniform, which can lead to faster convergence. This is because the algorithm does not need to take small steps in some directions while taking larger steps in others.In case of non-scaling we were taking alpha as small as possible but after scaling we do not need to take small steps.In this case when alpha is 0.0001 the model is giving training accuracy as 85.06 and validation accuracy as 83.49. But by increasing the alpha value to 0.1 the model accuracy gradually increases.It gives training accuracy as 85.33 and validation accuracy as 84.119.

```
Iteration    0: Train Cost 0.6678, Val Cost 0.6690, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 1000: Train Cost 0.3963, Val Cost 0.4281, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 2000: Train Cost 0.3885, Val Cost 0.4198, Train Accuracy 0.8517, Val Accuracy 0.8365
Iteration 3000: Train Cost 0.3850, Val Cost 0.4153, Train Accuracy 0.8527, Val Accuracy 0.8381
Iteration 4000: Train Cost 0.3832, Val Cost 0.4126, Train Accuracy 0.8537, Val Accuracy 0.8396
Iteration 5000: Train Cost 0.3821, Val Cost 0.4107, Train Accuracy 0.8540, Val Accuracy 0.8412
Iteration 6000: Train Cost 0.3814, Val Cost 0.4094, Train Accuracy 0.8543, Val Accuracy 0.8412
Iteration 7000: Train Cost 0.3809, Val Cost 0.4083, Train Accuracy 0.8540, Val Accuracy 0.8412
Iteration 8000: Train Cost 0.3805, Val Cost 0.4075, Train Accuracy 0.8537, Val Accuracy 0.8412
Iteration 9000: Train Cost 0.3803, Val Cost 0.4067, Train Accuracy 0.8533, Val Accuracy 0.8412
Iteration 9999: Train Cost 0.3800, Val Cost 0.4061, Train Accuracy 0.8533, Val Accuracy 0.8412
Model Accuracy On Training Data: 85.33378287255563 %
Model Accuracy On Validation Data: 84.11949685534591 %
FINAL MODEL WEIGHTS: [ 0.511749    2.31364771  0.01634449  0.07272869  0.86489395  0.4767781
  0.37003972  0.53835447  0.63039516  0.00848308  1.10934977 -0.31268679
 -0.06508701 -0.44560813  0.75555763]
FINAL BIAS: -3.664317345532909
ALPHA: 0.1
```

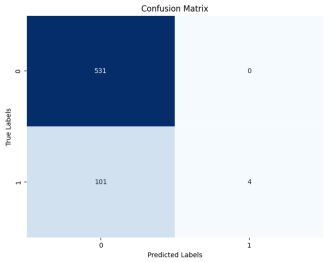




c)Without any Scaling



With Min-Max Scaling



```
Precision: 1.0000
Recall: 0.0190
F1 Score: 0.0374
ROC-AUC Score: 0.5095
```

```
Precision: 1.0000
Recall: 0.0381
F1 Score: 0.0734
ROC-AUC Score: 0.5190
[[531   0]
 [101   4]]
```

Note: Currently, 84% of the outcomes have a value of 0, meaning the data is highly imbalanced. As a result, the model is likely to learn and favor predictions for the majority class (0), potentially overlooking the minority class (1). This imbalance can lead to biased predictions, where the model performs well for the majority class but poorly for the minority class.That is why it is giving very small value for the person with heart disease (1) and it is predicting 0 for that. To reduce this we can use undersampling where we can create new dataset which has data equally divided for each class, i-e, we have same number of samples belonging to each class.
Various Metrics:-

Precision: Precision measures the accuracy of the positive predictions made by the model.

- With a precision of 1.0000, which is the same for both models, the model predicts a positive class 100% of the time.

Recall: Recall measures the ability of the model to identify all relevant positive cases.

- The model with Min-Max Scaling has a higher recall (0.0381) compared to the one without scaling (0.0190). This indicates that the model with scaling is slightly better at identifying true positive cases, although recall is still quite low for both.

F1 Score: The F1 score is the harmonic mean of precision and recall. It provides a single metric that balances the trade-off between precision and recall.

- The F1 score, which is a balance between precision and recall, is also slightly higher with Min-Max Scaling (0.0734) compared to without scaling (0.0374). This is expected because F1 considers both recall and precision, and the recall is higher in the scaled model.

ROC-AUC Score:

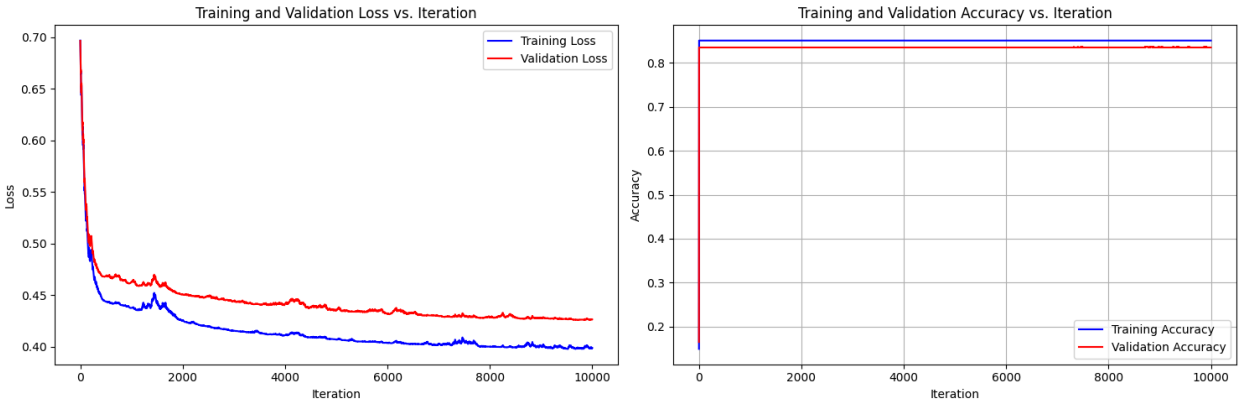
- The ROC-AUC score, which measures the ability of the model to distinguish between classes, is marginally better in the model with Min-Max Scaling (0.5190) compared to without scaling (0.5095). Both scores are quite close to 0.5, indicating that the models are not performing much better than random guessing.

The F1 score, and the ROC-AUC score shows marginally better performance for the model with Min-Max scaling than for the model without it. Although the recall is low, the precision is still perfect, which could be a sign of a dataset imbalance or a very conservative model that predicts the positive class.

d)Stochastic Gradient Descent Implementation:

```
Iteration    0: Train Cost 0.6951, Val Cost 0.6950, Train Accuracy 0.1494, Val Accuracy 0.1651
Iteration 1000: Train Cost 0.4411, Val Cost 0.4694, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 2000: Train Cost 0.4388, Val Cost 0.4684, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 3000: Train Cost 0.4356, Val Cost 0.4684, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 4000: Train Cost 0.4353, Val Cost 0.4691, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 5000: Train Cost 0.4353, Val Cost 0.4693, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 6000: Train Cost 0.4351, Val Cost 0.4690, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 7000: Train Cost 0.4356, Val Cost 0.4678, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 8000: Train Cost 0.4352, Val Cost 0.4680, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 9000: Train Cost 0.4364, Val Cost 0.4675, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 9999: Train Cost 0.4359, Val Cost 0.4675, Train Accuracy 0.8506, Val Accuracy 0.8349
Model Accuracy On Training Data: 85.06405933917735 %
Model Accuracy On Validation Data: 83.49056603773585 %
FINAL MODEL WEIGHTS: [ 1.69074373e-06 -5.13457173e-04 -6.96528843e-05 -1.16105091e-05
 -5.77311081e-05  4.78155303e-06  6.41340640e-07  2.31233601e-05
  5.04486626e-06 -3.96761304e-03 -1.66654415e-03 -1.52988294e-03
 -5.34760022e-04 -1.81770737e-03 -9.86536277e-04]
FINAL BIAS: -2.598971139384808e-05
```

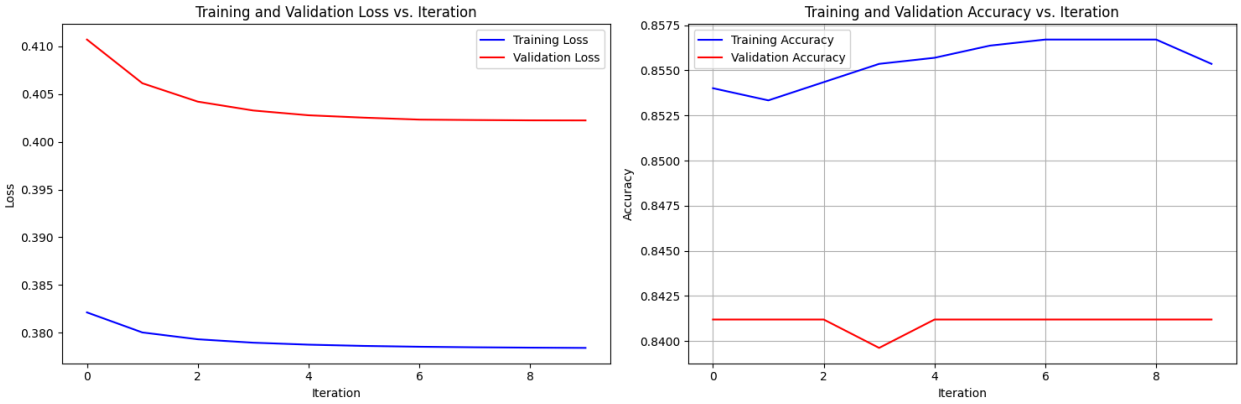
In stochastic gradient descent, setting the learning rate (α) to 0.01 with 10,000 iterations yields a training accuracy of 85.06% and a validation accuracy of 83.49%. Both accuracies improve consistently with this learning rate. However, when the learning rate decreases to 0.00001, the model's performance becomes more constant.



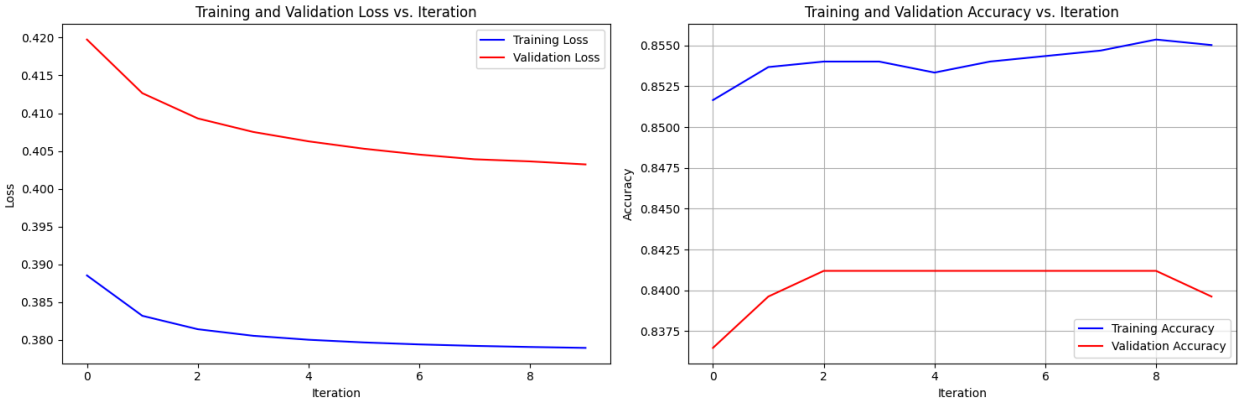
Mini-Batch Gradient Descent Implementation:

```
Iteration 1000: Train Cost 0.3821, Val Cost 0.4107, Train Accuracy 0.8540, Val Accuracy 0.8412
Iteration 2000: Train Cost 0.3800, Val Cost 0.4061, Train Accuracy 0.8533, Val Accuracy 0.8412
Iteration 3000: Train Cost 0.3793, Val Cost 0.4042, Train Accuracy 0.8543, Val Accuracy 0.8412
Iteration 4000: Train Cost 0.3789, Val Cost 0.4033, Train Accuracy 0.8554, Val Accuracy 0.8396
Iteration 5000: Train Cost 0.3787, Val Cost 0.4028, Train Accuracy 0.8557, Val Accuracy 0.8412
Iteration 6000: Train Cost 0.3786, Val Cost 0.4025, Train Accuracy 0.8564, Val Accuracy 0.8412
Iteration 7000: Train Cost 0.3785, Val Cost 0.4023, Train Accuracy 0.8567, Val Accuracy 0.8412
Iteration 8000: Train Cost 0.3785, Val Cost 0.4023, Train Accuracy 0.8567, Val Accuracy 0.8412
Iteration 9000: Train Cost 0.3784, Val Cost 0.4022, Train Accuracy 0.8567, Val Accuracy 0.8412
Iteration 10000: Train Cost 0.3784, Val Cost 0.4022, Train Accuracy 0.8554, Val Accuracy 0.8412
Model Accuracy On Training Data: 85.53607552258936 %
Model Accuracy On Validation Data: 84.11949685534591 %
FINAL MODEL WEIGHTS: [ 0.54412392  2.32628282  0.08407219  0.04093114  1.12353683  0.42794642
  0.63183632  0.3755483   0.19144298  0.58921117  2.3550087  -0.71374386
  0.39405332 -0.32856818  2.0376207 ]
FINAL BIAS: -4.26604894905234
Alpha: 0.1
```

In the implementation of mini-batch gradient descent, the model achieves an accuracy of 85.53% on the training dataset and 84.11% on the validation dataset when using a learning rate (alpha) of 0.01 and a batch size of 600. However, when the learning rate is increased to values such as 0.0001 or 0.0000015, the validation accuracy and training accuracy becomes constant.




```
Iteration 1000: Train Cost 0.3885, Val Cost 0.4197, Train Accuracy 0.8517, Val Accuracy 0.8365
Iteration 2000: Train Cost 0.3832, Val Cost 0.4126, Train Accuracy 0.8537, Val Accuracy 0.8396
Iteration 3000: Train Cost 0.3814, Val Cost 0.4093, Train Accuracy 0.8540, Val Accuracy 0.8412
Iteration 4000: Train Cost 0.3806, Val Cost 0.4075, Train Accuracy 0.8540, Val Accuracy 0.8412
Iteration 5000: Train Cost 0.3800, Val Cost 0.4063, Train Accuracy 0.8533, Val Accuracy 0.8412
Iteration 6000: Train Cost 0.3797, Val Cost 0.4053, Train Accuracy 0.8540, Val Accuracy 0.8412
Iteration 7000: Train Cost 0.3794, Val Cost 0.4045, Train Accuracy 0.8543, Val Accuracy 0.8412
Iteration 8000: Train Cost 0.3792, Val Cost 0.4039, Train Accuracy 0.8547, Val Accuracy 0.8412
Iteration 9000: Train Cost 0.3791, Val Cost 0.4036, Train Accuracy 0.8554, Val Accuracy 0.8412
Iteration 10000: Train Cost 0.3789, Val Cost 0.4032, Train Accuracy 0.8550, Val Accuracy 0.8396
Model Accuracy On Training Data: 85.50236008091706 %
Model Accuracy On Validation Data: 83.9622641509434 %
FINAL MODEL WEIGHTS: [ 0.52477758  2.37960441  0.06013184  0.04613738  1.06624057  0.44899836
  0.51812981  0.43222613  0.44664149  0.27652729  1.67816103 -0.36445294
  0.19668281 -0.32231205  1.2728637 ]
FINAL BIAS: -4.04672331077
Alpha: 0.1
```



Both models perform almost identically, with minor differences in accuracy and cost.
The model with batch size 600 achieves slightly better validation accuracy (84.12% vs. 83.96%) and lower validation cost, indicating slightly better generalization and learning stability.

With a smaller batch size (600), both training and validation costs decrease more effectively, indicating that smaller batches might better capture the nuances of the data. The larger batch size (2000) shows slower cost reduction for the validation set, which could be due to less frequent updates and less gradient noise.

Both batch sizes achieve similar final training accuracy. Validation accuracy is slightly better with the 600 batch size, suggesting that it might generalize slightly better.

e)
Below is the output for K fold validation with k=5

```
For Fold:  1
848
3390
Iteration    0: Train Cost 0.7836, Val Cost 0.7892, Train Accuracy 0.2646, Val Accuracy 0.2571
Iteration 1000: Train Cost 0.4358, Val Cost 0.4839, Train Accuracy 0.8519, Val Accuracy 0.8325
Iteration 2000: Train Cost 0.4340, Val Cost 0.4814, Train Accuracy 0.8516, Val Accuracy 0.8325
Iteration 3000: Train Cost 0.4324, Val Cost 0.4792, Train Accuracy 0.8516, Val Accuracy 0.8325
Iteration 4000: Train Cost 0.4310, Val Cost 0.4772, Train Accuracy 0.8519, Val Accuracy 0.8325
Iteration 5000: Train Cost 0.4298, Val Cost 0.4754, Train Accuracy 0.8519, Val Accuracy 0.8325
Iteration 6000: Train Cost 0.4286, Val Cost 0.4738, Train Accuracy 0.8519, Val Accuracy 0.8325
Iteration 7000: Train Cost 0.4275, Val Cost 0.4723, Train Accuracy 0.8516, Val Accuracy 0.8325
Iteration 8000: Train Cost 0.4265, Val Cost 0.4710, Train Accuracy 0.8516, Val Accuracy 0.8325
Iteration 9000: Train Cost 0.4256, Val Cost 0.4697, Train Accuracy 0.8519, Val Accuracy 0.8325
Iteration 9999: Train Cost 0.4248, Val Cost 0.4686, Train Accuracy 0.8519, Val Accuracy 0.8325
w are: [ 0.0044539  -0.00351735  0.01699569  0.01113473  0.00593023  0.00914796
 -0.00542236  0.00239459 -0.00063206 -0.00632485  0.00825866 -0.0147365
  0.00506564 -0.00605387  0.00403858]
b is -0.00018659398112021912
Accuracy: 0.8325
```

```
For Fold:  2
848
3390
Iteration    0: Train Cost 0.9924, Val Cost 0.8759, Train Accuracy 0.8445, Val Accuracy 0.8620
Iteration 1000: Train Cost 0.4625, Val Cost 0.4284, Train Accuracy 0.8445, Val Accuracy 0.8632
Iteration 2000: Train Cost 0.4589, Val Cost 0.4253, Train Accuracy 0.8445, Val Accuracy 0.8620
Iteration 3000: Train Cost 0.4559, Val Cost 0.4227, Train Accuracy 0.8445, Val Accuracy 0.8620
Iteration 4000: Train Cost 0.4533, Val Cost 0.4204, Train Accuracy 0.8445, Val Accuracy 0.8620
Iteration 5000: Train Cost 0.4511, Val Cost 0.4185, Train Accuracy 0.8445, Val Accuracy 0.8620
Iteration 6000: Train Cost 0.4493, Val Cost 0.4167, Train Accuracy 0.8445, Val Accuracy 0.8620
Iteration 7000: Train Cost 0.4476, Val Cost 0.4152, Train Accuracy 0.8445, Val Accuracy 0.8620
Iteration 8000: Train Cost 0.4461, Val Cost 0.4137, Train Accuracy 0.8445, Val Accuracy 0.8620
Iteration 9000: Train Cost 0.4448, Val Cost 0.4124, Train Accuracy 0.8445, Val Accuracy 0.8620
Iteration 9999: Train Cost 0.4436, Val Cost 0.4112, Train Accuracy 0.8445, Val Accuracy 0.8620
w are: [-0.00210622  0.00310511 -0.01093379 -0.01574776  0.00302878 -0.00746112
  0.0033852   0.01430159  0.00680513 -0.00729483  0.00616873 -0.01217197
  0.0133571  -0.00088437 -0.0018393 ]
b is -0.0001562040958065012
Accuracy: 0.8620
```

```
For Fold:  3
848
3390
Iteration    0: Train Cost 3.1900, Val Cost 3.2269, Train Accuracy 0.1519, Val Accuracy 0.1521
Iteration 1000: Train Cost 0.4587, Val Cost 0.4597, Train Accuracy 0.8496, Val Accuracy 0.8479
Iteration 2000: Train Cost 0.4522, Val Cost 0.4529, Train Accuracy 0.8496, Val Accuracy 0.8479
Iteration 3000: Train Cost 0.4471, Val Cost 0.4475, Train Accuracy 0.8493, Val Accuracy 0.8479
Iteration 4000: Train Cost 0.4430, Val Cost 0.4433, Train Accuracy 0.8487, Val Accuracy 0.8479
Iteration 5000: Train Cost 0.4397, Val Cost 0.4399, Train Accuracy 0.8487, Val Accuracy 0.8479
Iteration 6000: Train Cost 0.4370, Val Cost 0.4372, Train Accuracy 0.8484, Val Accuracy 0.8467
Iteration 7000: Train Cost 0.4348, Val Cost 0.4351, Train Accuracy 0.8484, Val Accuracy 0.8467
Iteration 8000: Train Cost 0.4330, Val Cost 0.4333, Train Accuracy 0.8484, Val Accuracy 0.8467
Iteration 9000: Train Cost 0.4315, Val Cost 0.4319, Train Accuracy 0.8481, Val Accuracy 0.8467
Iteration 9999: Train Cost 0.4303, Val Cost 0.4308, Train Accuracy 0.8484, Val Accuracy 0.8467
w are: [ 0.01061158 -0.00167243  0.00232257 -0.00172761  0.00157736 -0.00684044
 -0.01442375  0.0094857   0.00095477 -0.00158944  0.00168054 -0.00448628
 -0.00334591 -0.020494    0.00669605]
b is -0.00017554342240076474
Accuracy: 0.8467
```

```
For Fold:  4
847
3391
Iteration    0: Train Cost 0.9810, Val Cost 0.9639, Train Accuracy 0.1634, Val Accuracy 0.1688
Iteration 1000: Train Cost 0.4476, Val Cost 0.4379, Train Accuracy 0.8481, Val Accuracy 0.8477
Iteration 2000: Train Cost 0.4452, Val Cost 0.4369, Train Accuracy 0.8481, Val Accuracy 0.8477
Iteration 3000: Train Cost 0.4432, Val Cost 0.4362, Train Accuracy 0.8481, Val Accuracy 0.8477
Iteration 4000: Train Cost 0.4415, Val Cost 0.4356, Train Accuracy 0.8481, Val Accuracy 0.8477
Iteration 5000: Train Cost 0.4400, Val Cost 0.4351, Train Accuracy 0.8481, Val Accuracy 0.8477
Iteration 6000: Train Cost 0.4387, Val Cost 0.4346, Train Accuracy 0.8481, Val Accuracy 0.8477
Iteration 7000: Train Cost 0.4376, Val Cost 0.4342, Train Accuracy 0.8481, Val Accuracy 0.8477
Iteration 8000: Train Cost 0.4365, Val Cost 0.4338, Train Accuracy 0.8481, Val Accuracy 0.8477
Iteration 9000: Train Cost 0.4355, Val Cost 0.4334, Train Accuracy 0.8481, Val Accuracy 0.8477
Iteration 9999: Train Cost 0.4346, Val Cost 0.4330, Train Accuracy 0.8481, Val Accuracy 0.8477
w are: [-0.0126506  -0.00331694  0.01970029 -0.01823707  0.00663142  0.00445418
  0.01261611  0.01567156  0.00925704 -0.00541535  0.00270166  0.00130639
  0.00993238 -0.01444756  0.0007476 ]
b is -0.0002052287453423956
Accuracy: 0.8477
```

```
For Fold:  5
847
3391
Iteration    0: Train Cost 0.6155, Val Cost 0.6029, Train Accuracy 0.8475, Val Accuracy 0.8501
Iteration 1000: Train Cost 0.4446, Val Cost 0.4419, Train Accuracy 0.8472, Val Accuracy 0.8489
Iteration 2000: Train Cost 0.4412, Val Cost 0.4386, Train Accuracy 0.8475, Val Accuracy 0.8489
Iteration 3000: Train Cost 0.4385, Val Cost 0.4360, Train Accuracy 0.8478, Val Accuracy 0.8501
Iteration 4000: Train Cost 0.4365, Val Cost 0.4339, Train Accuracy 0.8475, Val Accuracy 0.8501
Iteration 5000: Train Cost 0.4348, Val Cost 0.4322, Train Accuracy 0.8475, Val Accuracy 0.8501
Iteration 6000: Train Cost 0.4335, Val Cost 0.4308, Train Accuracy 0.8475, Val Accuracy 0.8501
Iteration 7000: Train Cost 0.4323, Val Cost 0.4297, Train Accuracy 0.8475, Val Accuracy 0.8501
Iteration 8000: Train Cost 0.4314, Val Cost 0.4287, Train Accuracy 0.8475, Val Accuracy 0.8501
Iteration 9000: Train Cost 0.4306, Val Cost 0.4278, Train Accuracy 0.8475, Val Accuracy 0.8501
Iteration 9999: Train Cost 0.4299, Val Cost 0.4270, Train Accuracy 0.8475, Val Accuracy 0.8501
w are: [ 0.00465102  0.00731519 -0.00943254 -0.0034444   0.01052549  0.00816353
 -0.00327024 -0.01402814 -0.00939877 -0.00744346  0.00702153 -0.0049727
  0.01006866 -0.01535526  0.00046648]
b is -0.00017067023565920026
Accuracy: 0.8501
```

Mean of Metrics:-

```
Accuracy:  [0.8325471698113207, 0.8620283018867925, 0.847877358490566, 0.8476977567886659, 0.8500590318772137] Mean:  0.8480419237709118
```

```
Standard deviation:  0.00937998327103329
```

Accuracy shows how well the model correctly classifies data overall. It stays consistent across different tests, ranging from 83.25% to 85.01%, with an average of 84.36% and very little variation (standard deviation of 0.0060). This means the model consistently performs well at classifying correctly, making accuracy a reliable measure of its overall performance.

Precision shows how many positive predictions are actually correct. It changes a lot across different tests, ranging from 0.0 to 0.8, with an average of 0.285 and a high variability (standard deviation of 0.2936). In some cases, precision is almost perfect (0.8), meaning the model predicts positives well, but in others, it drops to 0.0, meaning the model misses all true positives. This shows the model's ability to predict positives is inconsistent.

Recall shows how well the model identifies all true positives. It varies a lot across tests, ranging from 0.0 to 0.085, with an average of 0.066 and a high variation (standard deviation of 0.0880). This means the model is inconsistent and sometimes misses all true positives in some tests.

The F1 score is a balance between precision and recall, combining them into one number. It ranges from 0.0 to 0.153, with an average value of 0.091 and a standard deviation of 0.1079. Because both precision and recall change a lot, the F1 score also shows these ups and downs.

The ROC-AUC score, which shows how well the model separates classes, has low variation and stays between 0.5 (random guessing) and 0.58, with an average of 0.5248. This means the model consistently performs close to random guessing, with weak ability to distinguish between the two classes(Heart Disease present or not).

The model shows stability in overall accuracy, it struggles with consistency in more specific metrics like precision, recall, and F1 score, especially in detecting and correctly identifying positive cases.

f)

In the case of L1 regularization, the model converged at 6,870 iterations when run for 10,000 iterations with a lambda value of 2. It was observed that as the lambda value increases, the number of iterations required for the model to converge decreases. When the lambda value was increased from 2 to 5, the model converged at 2,838 iterations, and with a lambda value of 10, it converged at 1,720 iterations.

The figure shown below is the output of L1 regularization:

```
Iteration 0: Train Cost 0.6678, Val Cost 0.6690, Train Accuracy 0.8506, Val Accuracy 0.8349
Iteration 1000: Train Cost 0.4013, Val Cost 0.4362, Train Accuracy 0.8506, Val Accuracy 0.8349
Early stopping at iteration 1722
Model Accuracy On Training Data: 85.06405933917735 %
Model Accuracy On Validation Data: 83.49056603773585 %
FINAL MODEL WEIGHTS: [ 3.40617991e-01  7.26882124e-01 -1.36892510e-01  1.37209977e-05
  5.26034279e-05  5.76686652e-04 -1.78672766e-06  7.12220862e-01
  9.03301592e-02  8.58586835e-05  4.80041160e-04  1.28111705e-04
 -1.78860417e-04 -1.24860642e-01 -7.21103355e-05]
FINAL BIAS: -2.365062408933824
ALPHA: 0.1
```

In the case of L2 regularization, it was observed that with a lambda value of 0.01, the model converged at 453 iterations using the same alpha value.

The graph shown below is the output of L2 regularization:

```
Iteration 0: Train Cost 0.6690, Val Cost 0.6702, Train Accuracy 0.8506, Val Accuracy 0.8349
Early stopping at iteration 453
Model Accuracy On Training Data: 85.06405933917735 %
Model Accuracy On Validation Data: 83.49056603773585 %
FINAL MODEL WEIGHTS: [ 0.19100066  0.08444202 -0.43000863 -0.17840089  0.0491165   0.15580543
  0.0345783   0.57938857  0.1673899  -0.22901703 -0.0341605  -0.30143712
 -0.25473664 -0.41255723 -0.0486253 ]
FINAL BIAS: -1.2858093098293157
ALPHA: 0.1
```

In both L1 and L2 regularization, the training accuracy and validation accuracy remained almost the same.
L1 regularization requires a higher lambda to converge, which might suggest more aggressive regularization or sparsity enforcement. L2 regularization converges at a lower lambda, indicating smoother regularization and possibly better balance between fitting and regularization.