AI330_ Machine Learning Projects_Fall2023

Numerical dataset:

General information about dataset:

Name	Medical Insurance Cost Prediction
Total Number of Samples in Data set	(1338, 7)
No. of samples in training\validation	(1070, 6)
Number of Features	6
No. of samples in testing	(268, 6)

Linear Regression Model:

Model Evalution:

Model Evaluation

```
[75] # Predicting on traning data
     training_data_prediction=regressor.predict(X_train)
```

```
     # R_ squared value
     r2_train=metrics.r2_score(training_data_prediction,Y_train)
     print('R Squared Value :',r2_train)
```

```
R Squared Value : 0.6693380032904606
```

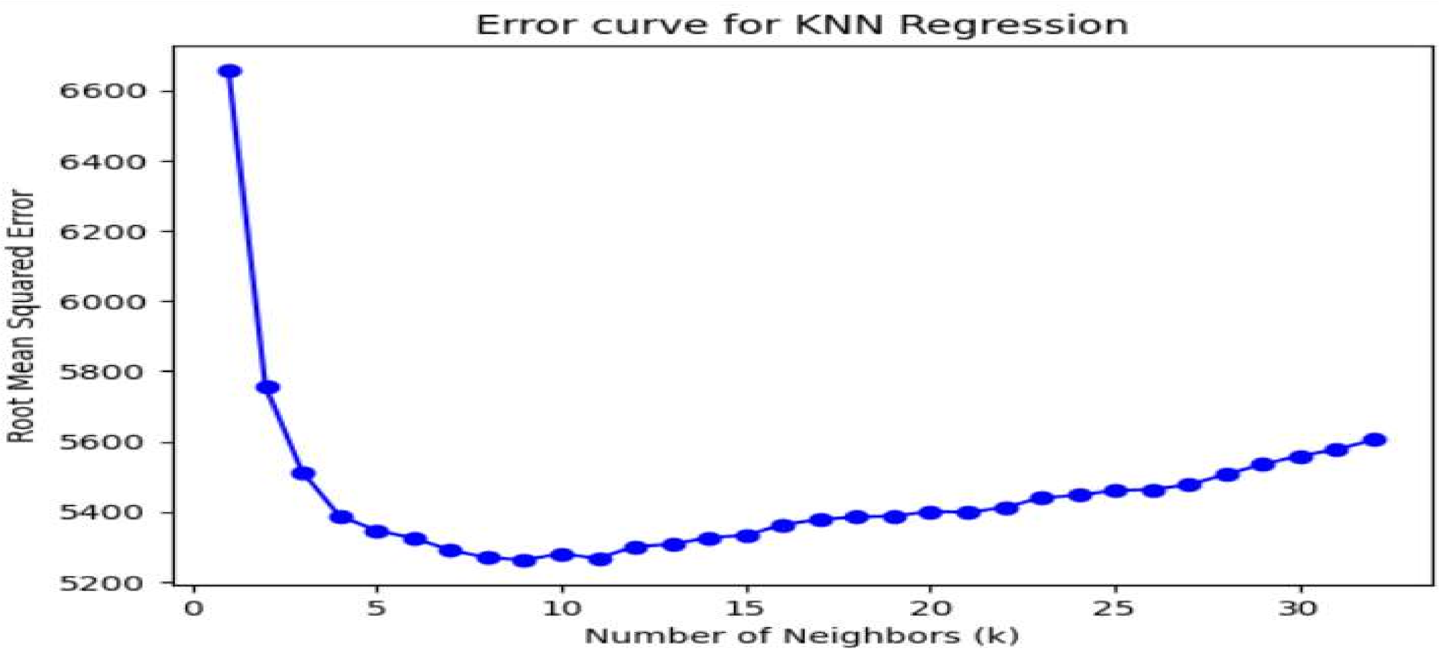
```
[77] # predicting on test data
     test_data_prediction=regressor.predict(X_test)
```

```
[78] # R Squared value
     r2_test =metrics.r2_score(test_data_prediction,Y_test)
     print('R Squared Value :',r2_test)
```

```
R Squared Value : 0.6606659986058923
```

Knn Regressor Model:

Implementation details:



Hyper Parameter :

Values for K = 32

best _K = 9

Results details:

```
# final version
# Make predictions on the Train  data
Prediction_Training_Data=final_knn.predict(X_train_scaled)

# R-squared on train data
r2_train=r2_score(Prediction_Training_Data,Y_train)
print('R Squared on Train data =',r2_train)
```

```
R Squared on Train data = 0.8187525607186243
```

```
[92] # final version
# Make predictions on the test data
Y_test_pred=final_knn.predict(X_test_scaled)

# R-squared on test data
r2_test=r2_score(Y_test_pred,Y_test)
print('R squared on Test data =',r2_test)
```

```
R squared on Test data = 0.7675351764760497
```

Mean Squared Error On test Data:

```
[93] # final version
# Mean Squared Error on test data
mse_test=mean_squared_error(Y_test_pred,Y_test)
print('Mean Squared Error on Test Data=',mse_test)
```

```
Mean Squared Error on Test Data= 27839082.124536395
```

```
[ ] rmse_values=[]
```

```
[ ] # final version
    # Try different k values and evaluate their performance
    for k in range(1, k_value + 1):  # (1,32)

        # Create a KNN model with the current k value
        knn = KNeighborsRegressor(n_neighbors=k)

        # Perform cross-validation to get a more robust estimate
        scores=cross_val_score(knn,X_train_scaled,Y_train,cv=5,scoring='neg_mean_squared_error')

         # Calculate and store RMSE
        rmse=sqrt(-scores.mean())
        rmse_values.append(rmse)
        # print the result for each k
        print('RMSE value for k=',k ,'is:',rmse)
```

```
RMSE value for k= 1 is: 6657.457023001403
RMSE value for k= 2 is: 5754.626289955937
RMSE value for k= 3 is: 5512.408864296902
RMSE value for k= 4 is: 5388.281980145013
RMSE value for k= 5 is: 5346.673664516487
RMSE value for k= 6 is: 5325.016806265275
RMSE value for k= 7 is: 5290.9998724274365
RMSE value for k= 8 is: 5268.693278561807
RMSE value for k= 9 is: 5262.253100504169
RMSE value for k= 10 is: 5280.3238951788
RMSE value for k= 11 is: 5266.968272821496
RMSE value for k= 12 is: 5300.994449859258
RMSE value for k= 13 is: 5307.495864818422
```

Connected to Python 3 Google Compute Engine backend

Cross Validation with K_folds = 5

train_val_ratio = 0.8

specifies that 80% of the data will be used for training the model, and the remaining 20% will be used for validation during cross-validation.

Image Dataset:

a. General information about dataset:

name of dataset:	**UTK Face (Age Estimation)**
number of classes and their labels:	**Classes = 2 , labels = 2**
the total number of samples in dataset:	**train images = 1839, test images = 774**
Size of Each Image:	**Images are resized to (28, 28, 3) during preprocessing.** **((32,32) with HOG)**
the number of samples used in training :	**train images = 1283**
the number of samples used in testing :	**test images = 530**

b. Implementation details:

Logistic regression :

Number of Features: Depends on HOG parameters, but not explicitly mentioned.

Feature Names: HOG features are essentially gradients in different orientations, but not explicitly named in this context.

HOG Parameters:

orientations=8: Number of orientation bins.

pixels_per_cell=(4, 4): Size (in pixels) of a cell.

cells_per_block=(1, 1): Number of cells in each block.

Dimension of Resulted Features: **(32,32)**

Hyperparameters used in your model: *(model = LogisticRegression(solver='liblinear'))*

the solver is an optimization algorithm that the model uses to find the weights that minimize the cost function. Liblinear is one of the solvers available in scikit-learn's LogisticRegression implementation.

It is specifically designed for linear models,such as logistic regression, and is well-suited for binary and multiclass classification problems.

kmeans as classifiers:

Number of Features: Depends on HOG parameters, but not explicitly mentioned.

Feature Names: HOG features are essentially gradients in different orientations, but not explicitly named in this context.

HOG Parameters:

orientations=8: Number of orientation bins.

pixels_per_cell=(4, 4): Size (in pixels) of a cell.

cells_per_block=(1, 1): Number of cells in each block.

Dimension of Resulted Features: Reduced to 50 components using PCA.

Hyperparameters:

K-Means Hyperparameters:

n_clusters=2: Number of clusters.

random_state=42: Random seed for reproducibility.

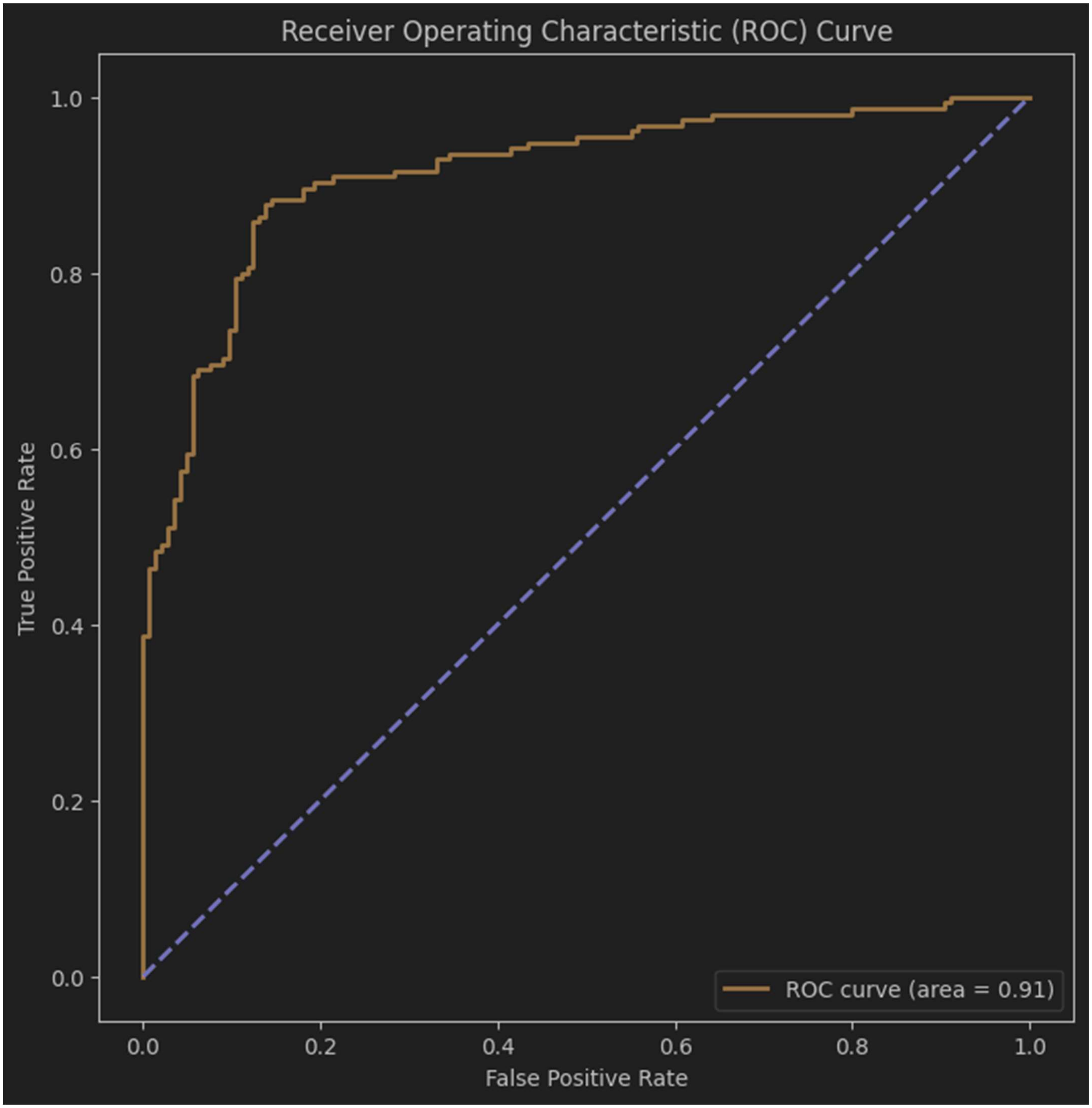
n_init=10: Number of runs with different centroid seeds.

c. **Results details:**

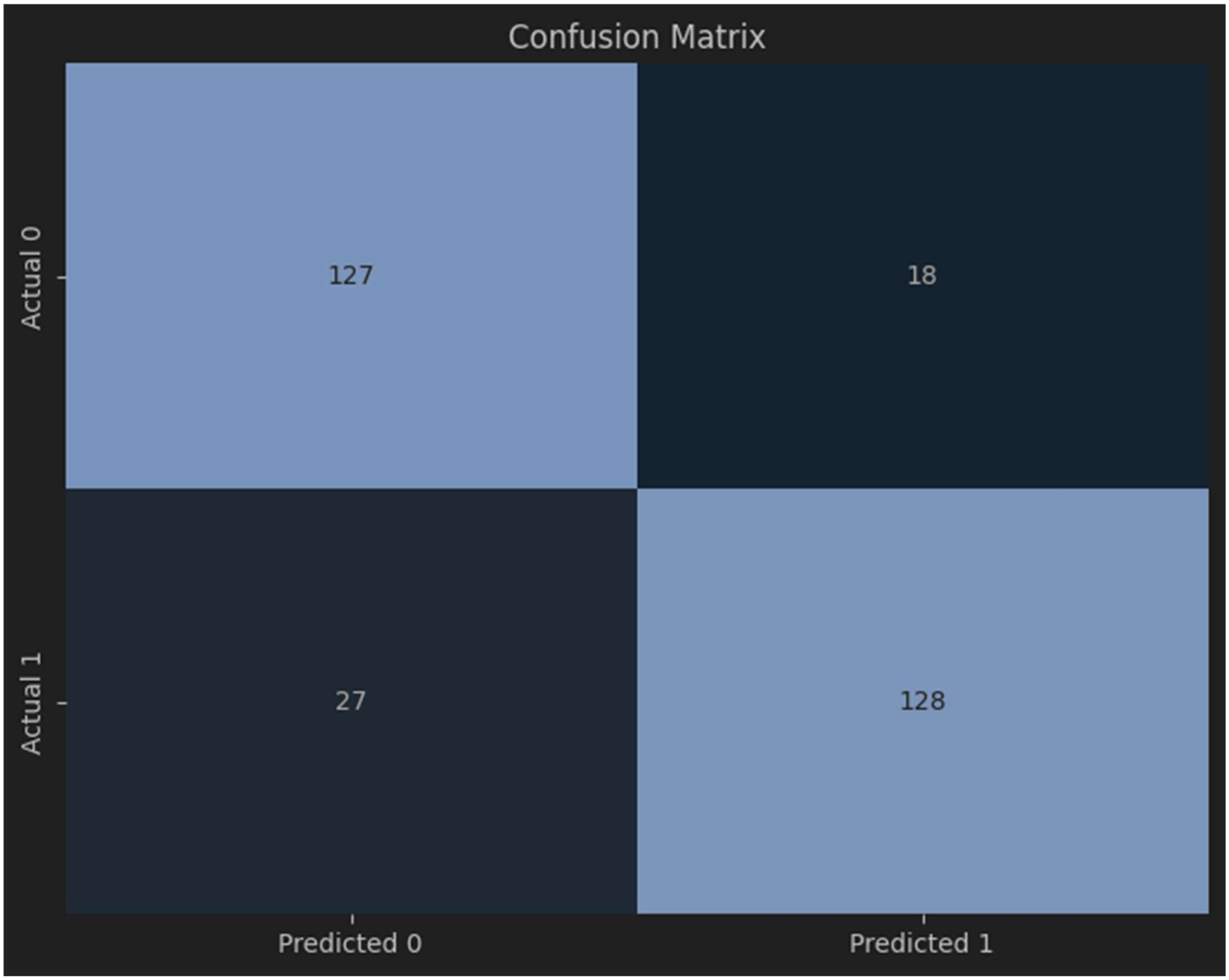
Logistic regression :

Accuracy: 0.8500

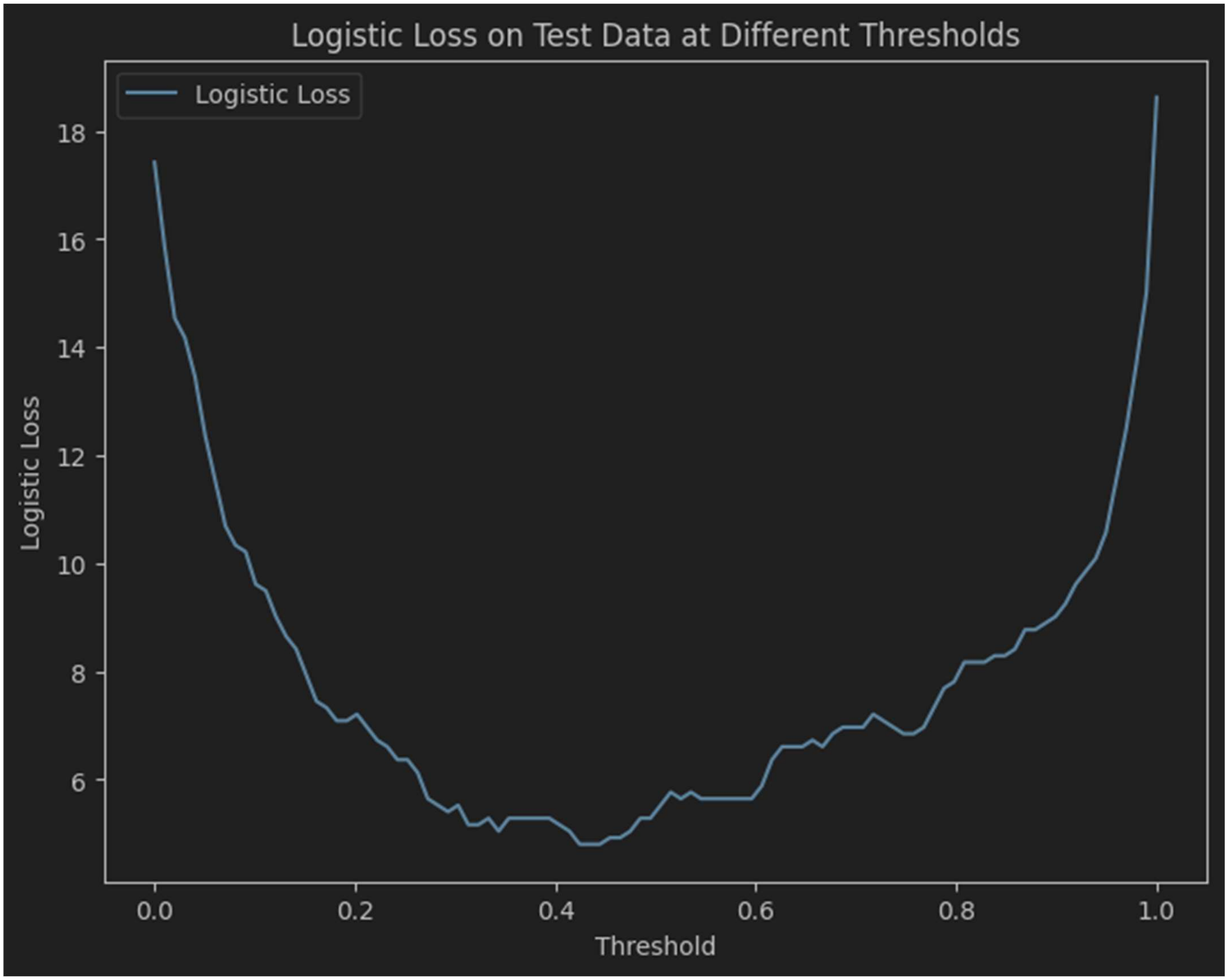
ROC curve:


Receiver Operating Characteristic (ROC) Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
ROC curve (area = 0.91)

confusion matrix:



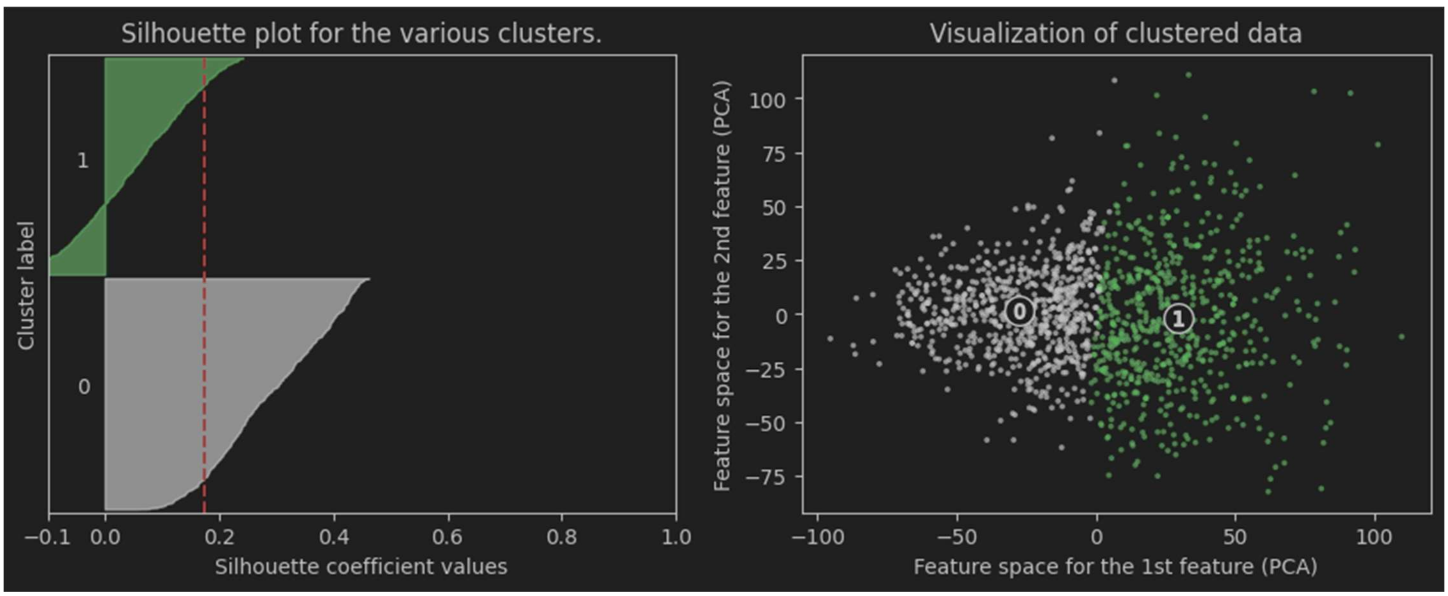
loss curve:



Logistic Loss on Test Data at Different Thresholds
Logistic Loss
18
16
14
12
10
8
6
Logistic Loss
0.0
0.2
0.4
0.6
0.8
1.0
Threshold

kmeans as classifiers:

Accuracy: 0.5697583787996883
Silhouette Coefficient: 0.17459127519351436
Calinski-Harabasz Score: 281.56693799155323
Davies-Bouldin Index: 2.0263179690082835



Evaluation Metrics:

Accuracy: Computed accuracy using scikit-learn's metrics.accuracy_score.

Additional Clustering Metrics:

- Silhouette Coefficient.

- Calinski-Harabasz Score.

- Davies-Bouldin Index.

Visualization:

- Silhouette Plot: Visualized silhouette scores for each cluster.

- Scatter Plot of Clustered Data: Visualization of clustered data in reduced feature space using PCA.