UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Data Mining Project by Yahya Momtaz

Heart Attack Analysis and Prediction

Professors:

Mod A: Professor Roberta Siciliano
Mod B: Professor Giuseppe Longo

University of Naples Federico II

Introduction

According to the world health organization, Cardiovascular diseases (CVDs) are the leading cause of death globally. In 2019 alone, around 17.9 million people died from CVDs. Of these deaths, **85%** of them were due to heart diseases. There are many factors that play a role in increasing the risk of heart disease. Identifying these factors and their impact is paramount in the field of healthcare. Identifying patients who are at greater risk enables medical professionals to respond quickly and efficiently, saving more lives.

About the Dataset:

The [Personal Key Indicators of Heart Disease] dataset contains 320K rows and 18 columns. It is a cleaned, smaller version of the [2020 annual CDC (Centers for Disease Control and Prevention) survey data of 400k adults]. For each patient (row), it contains the health status of that individual. The data was collected in the form of surveys conducted over the phone. Each year, the CDC calls around 400K U.S residents and asks them about their health status, with the vast majority of questions being yes or no questions. I made the decision to manually clean and predict the missing values after downloading the original dataset.

Data Preprocessing:

In any data-driven project, data preprocessing is an essential step that can significantly impact the quality and performance of machine learning models. The heart disease dataset requires thorough data preprocessing to ensure that the data is in a suitable format for building predictive models. In this section, we outline the various data preprocessing steps performed on the dataset.

The raw data is often in a form that is not suitable for processing (i.e., raw logs, documents, semi-structured data, other forms of heterogeneous data).

Some algorithms may work only with a specific data type, whereas the data may contain heterogeneous types (requiring portability).

In order to have comparisons afterwards, I will keep the columns as the Kaggle dataset's column.

```
# Define a dictionary for column renaming
columns_name_mapping = {
    "cvdcrhd4": "HeartDisease",
    "weight2": "Weight",
    "height3":"Height",
    "_smoker3":"Smoking",
    "drnkany5":"AlcoholDrinking",
    "cvdstrk3":"Stroke",
    "physhlth":"PhysicalHealth",
    "menthlth":"MentalHealth",
    "diffwalk":"DiffWalking",
    "_sex":"Sex",
    "_ageg5yr":"AgeCategory",
    "_race":"Race",
```

```
    "diabete4":"Diabetic",
    "exerany2":"PhysicalActivity",
    "genhlth":"GenHealth",
    "sleptim1":"SleepTime",
    "_asthms1":"Asthma",
    "chckdny2":"KidneyDisease",
    "chcscncr":"SkinCancer",
    "_bmi5":"BMI"
}
```

Data Cleaning:

Handling Missing Values: Missing data points were identified and addressed. Depending on the extent of missing data in each column, I applied appropriate strategies such as removal and advanced imputation methods. We can see number of missed values for each columns below.

```
HeartDisease            3
Weight               9852
Height              10824
Smoking                 0
AlcoholDrinking         0
Stroke                  3
PhysicalHealth          5
MentalHealth            5
DiffWalking         15280
Sex                     0
AgeCategory             0
Race                    1
Diabetic                6
PhysicalActivity        3
GenHealth               8
SleepTime               3
Asthma                  0
KidneyDisease           6
SkinCancer              3
BMI                 41357
dtype: int64
```

Duplicate Rows: Duplicate rows, if any, were removed to avoid biasing the model. Erroneous and inconsistent entries are removed, missing are estimated by imputation methods.

Handling Outliers:

Outliers in the data were identified and treated through techniques such as trimming, transformation, or removal.

Feature Engineering:

Creating New Features: New features were engineered to potentially enhance the predictive power of the model.

Feature Modification: Existing features were transformed or modified to better align with the model's assumptions.

Value Mapping:

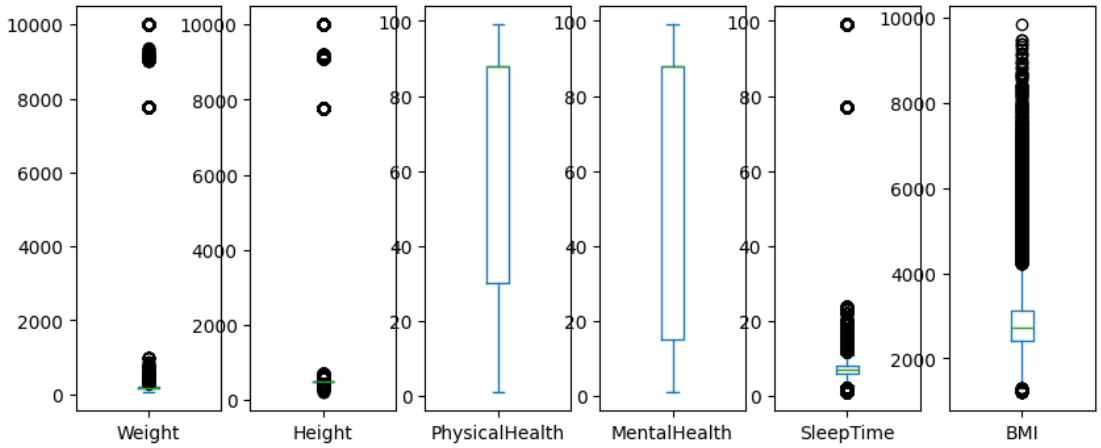
Specific values in the dataset were mapped to new values to ensure consistency and to address irregularities.

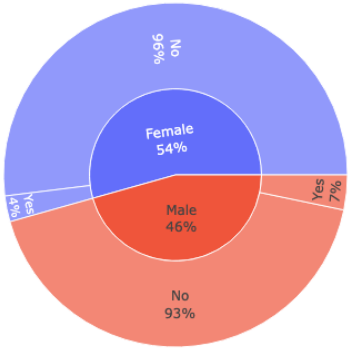
Exploratory Data Analysis (EDA):

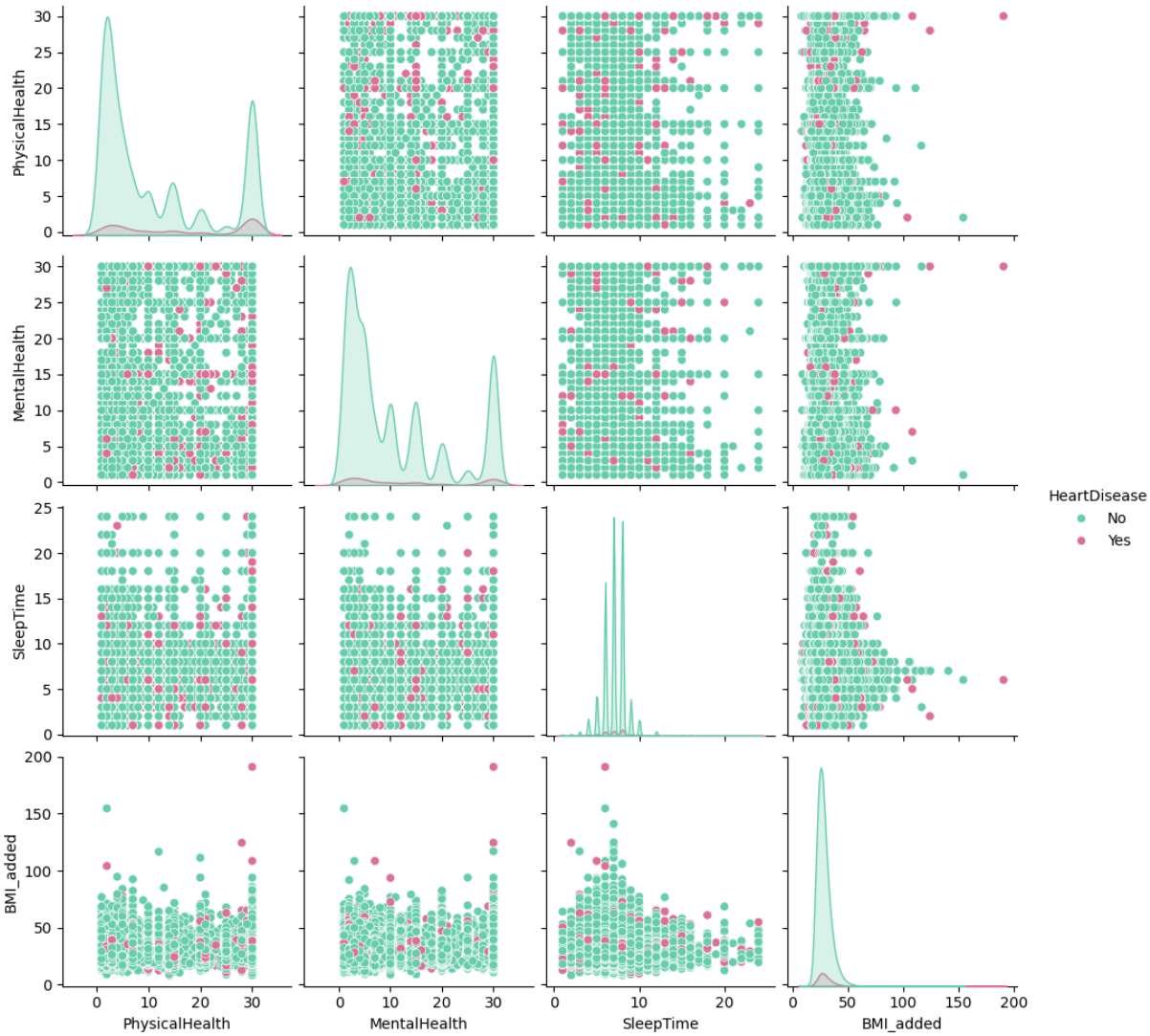
For 'Weight' and 'Height,' the data shows a wide range of values with several extreme outliers, as indicated by points that are far outside the "whiskers" of the plot. 'PhysicalHealth' and 'MentalHealth' have similar box sizes, suggesting similar variability in the number of days respondents felt their health was not good, with a few outliers indicating individuals experiencing a high number of days with poor health.

'SleepTime' appears to have a more compact interquartile range, suggesting less variability in the number of hours slept, but there are outliers on both the high and low ends, indicating some individuals get significantly more or less sleep than average.

The 'BMI' plot shows a large number of outliers above the upper whisker, suggesting a substantial number of individuals with a BMI higher than what is captured by the main spread of the data.




No
96%
Female
54%
Yes
4%
Male
46%
Yes
7%
No
93%


PhysicalHealth
MentalHealth
SleepTime
BMI_added
HeartDisease
No
Yes

The following data are observed from **POSITIVE** Heart Disease cases:

SMOKING:

A substantial proportion consists of individuals who have never smoked, followed closely by former smokers, which could reflect successful smoking cessation efforts or changes in lifestyle after diagnosis.

-Current smokers and those who are categorized as approximate smokers represent a smaller proportion of the heart disease group, suggesting either a lower prevalence of smoking among these patients or successful quitting post-diagnosis.

This smoking distribution underscores the complexity of the relationship between smoking and heart disease, where factors such as the duration and intensity of smoking, as well as quitting smoking, interact with heart disease risk.

ALCOHOL DRINKING:

The majority report not drinking alcohol.

The proportion of heart disease patients who do consume alcohol is notable and may reflect the complex relationship between alcohol use and cardiovascular health.

The distribution suggests the need for a nuanced understanding of alcohol consumption patterns among heart disease patients and their impact on health.

STROKE:

Most have not experienced a stroke, which may reflect the effectiveness of preventive measures or treatments that target both conditions.

However, the 15.8% of heart disease patients who have had a stroke underscore the importance of considering cerebrovascular health in the context of cardiovascular disease management.

This distribution highlights the need for continued research and intervention strategies to reduce the risk of stroke in patients with heart disease.

A significant majority (84.2%) of heart disease patients have not had a stroke.
A smaller portion (15.8%) of patients with heart disease have had a stroke.

DIFF WALKING:

Most individuals with heart disease do not experience difficulty walking, a substantial minority does, highlighting the impact of heart disease on physical mobility.

The 39.6% of heart disease patients reporting difficulty walking may reflect complications or severity of the disease affecting physical function.

These findings emphasize the importance of physical rehabilitation and mobility support in heart disease management.

SEX:

Males account for a larger percentage of the heart disease cases at 57.4%.
Females represent 42.6% of the heart disease cases.

DIABETIC:

The majority of heart disease cases do not coincide with diabetes, suggesting that while diabetes is a risk factor for heart disease, it is not present in all cases.

Diabetes is still prevalent in over a third of the heart disease patients, emphasizing the link between the two conditions.

The presence of borderline and gestational diabetes among heart disease patients indicates the need for monitoring and managing blood sugar levels as part of cardiovascular care.

PHYSICAL ACTIVITY:

A significant majority of individuals with heart disease engage in physical activity, which is often recommended as part of a heart-healthy lifestyle.

The substantial proportion of heart disease patients who do not engage in physical activity could be targeted for interventions to increase activity levels as part of their treatment plan.

These findings may also reflect the challenges some heart disease patients face in becoming physically active, whether due to symptoms, comorbidities, or other barriers.

GEN HEALTH:

Those who rate their general health as "Good," comprising 34% of the individuals with heart disease.

The "Fair" category is the next largest segment at 27.8%, followed by "Very Good" at 17.7% and "Poor" at 16.1%.

A smaller segment, at 4.41%, represents those who rate their health as "Excellent”.

ASTHMA:

While asthma is less common among heart disease patients, a notable fraction does contend with both conditions concurrently.

The low percentage of former asthma patients could imply successful management or resolution of asthma in the context of heart disease.

These statistics may inform healthcare providers about the comorbidity of asthma in patients with heart disease and the need to manage both conditions effectively.

KIDNEY DISEASE:

A notable percentage of individuals with heart disease also contend with kidney disease, which can complicate the management of their cardiovascular condition.

The prevalence of kidney disease in heart disease patients underscores the importance of regular kidney function monitoring in this population.

SKIN CANCER:

While the prevalence of skin cancer among individuals with heart disease is not the majority, it is still present in a significant minority.

The data may suggest the need for regular dermatological screenings for individuals with heart disease, given the non-negligible overlap between the two conditions.

The 19.4% prevalence of skin cancer in heart disease patients might be related to shared risk factors, such as age, or the effects of long-term medications.

Missing Data Imputation:

Missing data imputation is an important step in data editing process when data come from different self-updating repositories and are huge in dimensionality.

Addressing missing data is a fundamental aspect of data preprocessing. When missing data was encountered, appropriate imputation techniques were applied, such as using the mean, median, or mode, to fill in missing values.

Missing data occur when no data value is stored for one or more measurements of a given variable in a statistical survey. The goodness of data analysis does depend on the way missing data are treated in data pre-processing.

We focus the attention on Missing At Random (MAR) mechanism, where missing-ness depends only on the complete part of the data matrix and not on the missing part.

These cleaning and preprocessing steps are essential to create a clean and consistent dataset that is suitable for building machine learning models. Detailed documentation of these steps is crucial for transparency, reproducibility, and collaboration in the data analysis process.

Types of Missing Data:

In the field of data analysis and machine learning, missing data is a common challenge. Understanding the types of missing data is crucial for choosing the right imputation technique. There are three primary types of missing data:

Missing Completely at Random (MCAR):

- We mean that the probability that an observation is missing is unrelated to the value of the variable or to the value of any other variables.

- If the data are MCAR then missing values cannot be predicted any better with all information observed or not.

Missing at Random (MAR):

If data meet the requirement that missingness (i.e., the manner in which data are missing from a sample of a population) does not depend on the value of the variable after controlling for another variable.

The assumption of MissingatRandom(MAR) is satisfied when the probability of a value's being missing in one variable is unrelated to the probability of missing data in another variable, but may be related to the value of the variable itself.

It can be considered as a weaker assumption than MCAR.
MAR would be satisfied if the probability of missing income was related to marital status, but unrelated to income within a marital status.

If MAR is satisfied,the mechanism causing the missing data may be considered to be "ignorable." That is, it doesn't matter why MAR occurred, only that it occurred.

Missing Not at Random (MNAR):

The reason for observations being missing still depends on the unseen observations themselves.

Decision Tree Imputation

Imputing missing data is a crucial step in data preprocessing. It helps ensure that the dataset is complete and ready for machine learning modeling. Various imputation techniques are available, and the choice of technique depends on the nature of the data and the type of missing data.

One imputation method that we applied to the heart disease dataset is decision tree imputation. Decision tree imputation involves using a decision tree model to predict and fill in missing values based on other observed variables. This technique offers several advantages:

Non-linearity: Decision trees can capture complex, non-linear relationships in the data. They are well-suited for handling non-linear relationships between variables, which is common in real-world datasets.

Handling Categorical Data: Decision trees can naturally handle both categorical and numerical data, making them suitable for datasets with a mix of variable types.

Variable Importance: Decision tree models can provide insights into the importance of each feature for imputing missing values. This information can be valuable for understanding which variables play a key role in imputation.
Robustness to Outliers: Decision trees are robust to outliers in the data, making them a suitable choice when missing data is related to outliers.

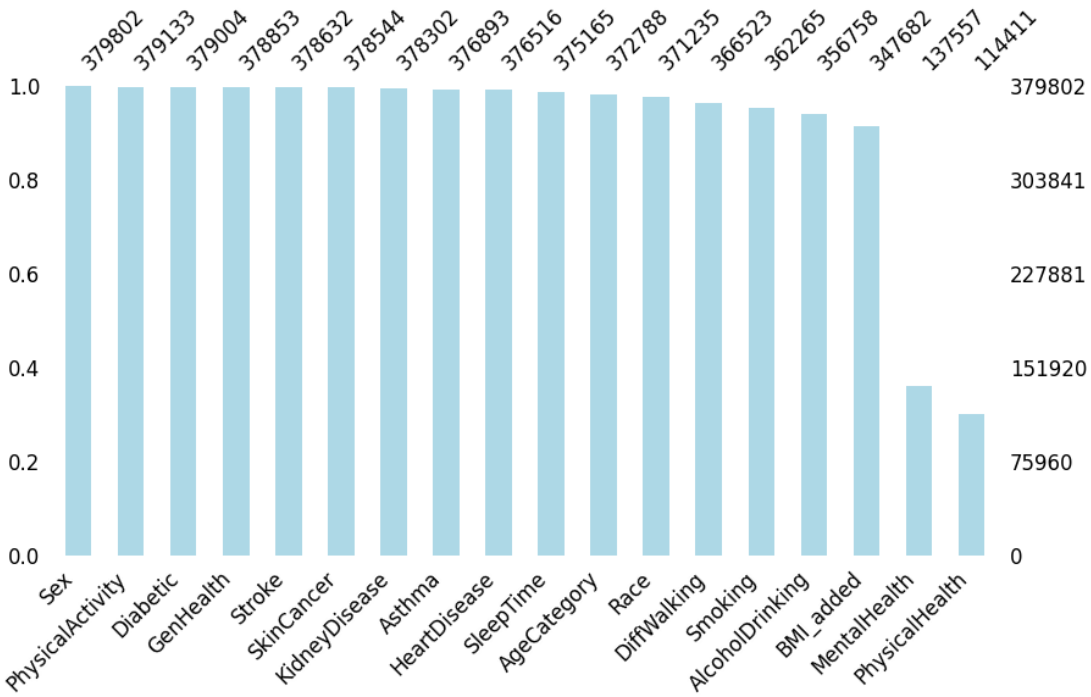
By utilizing decision tree imputation, we aim to fill in missing values in a way that preserves the underlying relationships within the data, thereby improving the quality of the dataset and enhancing the performance of machine learning models.

The choice of imputation technique should be driven by the nature of the data and the specific objectives of the analysis. In our case, decision tree imputation was selected due to its ability to capture non-linear relationships and handle both categorical and numerical data effectively.

Value Mapping for Imputation:

```
value_mapping = {
    'HeartDisease': {1:1, 2:0, 7:np.nan, 9:np.nan},
    'AlcoholDrinking': {1:1, 2:0, 7:np.nan, 9:np.nan},
    'Stroke': {1:1, 2:0, 7:np.nan, 9:np.nan},
    'PhysicalActivity': {1:1, 2:0, 7:np.nan, 9:np.nan},
    'DiffWalking': {1:1, 2:0, 7:np.nan, 9:np.nan},
    'SkinCancer': {1:1, 2:0, 7:np.nan, 9:np.nan},
    'KidneyDisease': {1:1, 2:0, 7:np.nan, 9:np.nan},
    'Asthma': {1:1, 2:2, 3:0, 9:np.nan},
    'Smoking': {1:1, 2:2, 3:3, 4:0, 9:np.nan},
    'AgeCategory': {14:np.nan},
    'Race': {9:np.nan},
    'Diabetic': {1:1, 2:2, 3:0, 4:0, 7:np.nan, 9:np.nan},
    'GenHealth': {1:1, 2:2, 3:3, 4:4, 5:5, 7:np.nan, 9:np.nan},
}

# Use the replace() method to change the values in the DataFrame
df.replace(value_mapping, inplace=True)
```

MSNO bar chart:



The data suggests that some variables have more missing data than others, which is common in real-world datasets. It's important to address these missing values appropriately as they can significantly affect the results of any analysis. Methods to handle missing data include imputation, where missing values are filled in based on other available data, or complete case analysis, where only records with no missing values are analyzed. The choice of method can depend on the type of analysis being conducted and the assumed mechanism causing the data to be missing.

Cluster Analysis

A cluster can be defined in terms of internal cohesion – homogeneity – and external isolation – separation (Gordon, 1980).
The process of dividing a homogeneous data set into different parts is referred to as dissection.

Clustering refers to a very broad set of methodsfor finding subgroups, or clusters, in a data set.

When we cluster the observations of a dataset,we seek to partition them into distinct groups so that observations within each group are quite similar to each other, while observations in different groups are quite different from each other, with respect to the measurements of a set of features.

To make this concrete,we must define what it means for two or more observations to be similar or different. This depends on the knowledge of the data to be analyzed.

Goal:

Search for the partition of the set of statistical units into a certain number of groups (i.e., clusters) such that.

with respect to the measurements of the available features (i.e., variables).

I decided to do cluster analysis for those who have heart disease with K-Means.

K-means clustering is a popular unsupervised machine learning algorithm used for clustering and partitioning data into groups or clusters based on similarity. Here are some key points about K-means clustering:

1 - Objective:

The primary goal of K-means clustering is to group similar data points together and identify natural clusters within a dataset. It does so by minimizing the variance within clusters.

2 - Algorithm:

K-means is an iterative algorithm. It starts with a predefined number of clusters, 'K', and assigns each data point to the nearest cluster centroid. Then, it recalculates the cluster centroids as the mean of the data points assigned to each cluster. This process is repeated until convergence, where the assignment of data points to clusters remains unchanged.

3 - Initialization:

The choice of initial cluster centroids can influence the results. Common methods for initialization include random selection, K-means++, and others.

4 - Number of Clusters (K):

One of the challenges in K-means is determining the optimal number of clusters (K) for a given dataset. Several techniques, like the elbow method and silhouette analysis, can help in selecting an appropriate K.

5 - Data Points and Centroids:

Each data point belongs to one cluster, and each cluster is represented by its centroid, which is the mean of all data points in that cluster.

6 - Distance Metric:

K-means typically uses Euclidean distance to measure the dissimilarity between data points and cluster centroids. However, other distance metrics can be used based on the data's nature.

7 - Scalability:

K-means is scalable and works well for large datasets. However, its performance may be affected by the choice of 'K' and the initial centroid selection.

8 - Applications:

K-means clustering is widely used in various fields, including customer segmentation, image segmentation, document categorization, anomaly detection, and more.

9 - Strengths:

K-means is simple, computationally efficient, and easy to understand. It's a good choice when you have a clear idea of the number of clusters in your data.

10 - Weaknesses:

K-means can be sensitive to the initial centroid positions and may not work well with non-spherical or unevenly sized clusters. It's also important to be cautious about outliers, as they can influence the results.

11 - Variants:

There are various variants of K-means, such as K-medoids (PAM), which uses medoids as cluster representatives, and hierarchical K-means, which builds a hierarchy of clusters.

Feature Engineering for cluster analysis:

Assuming that 0 in 'AlcoholDrinking' means not drinking and 1 means drinking

And that 1 in 'PhysicalActivity' means active and 0 means inactive

We will create a score such that:

Not drinking alcohol: +1 point

Being physically active: +1 point

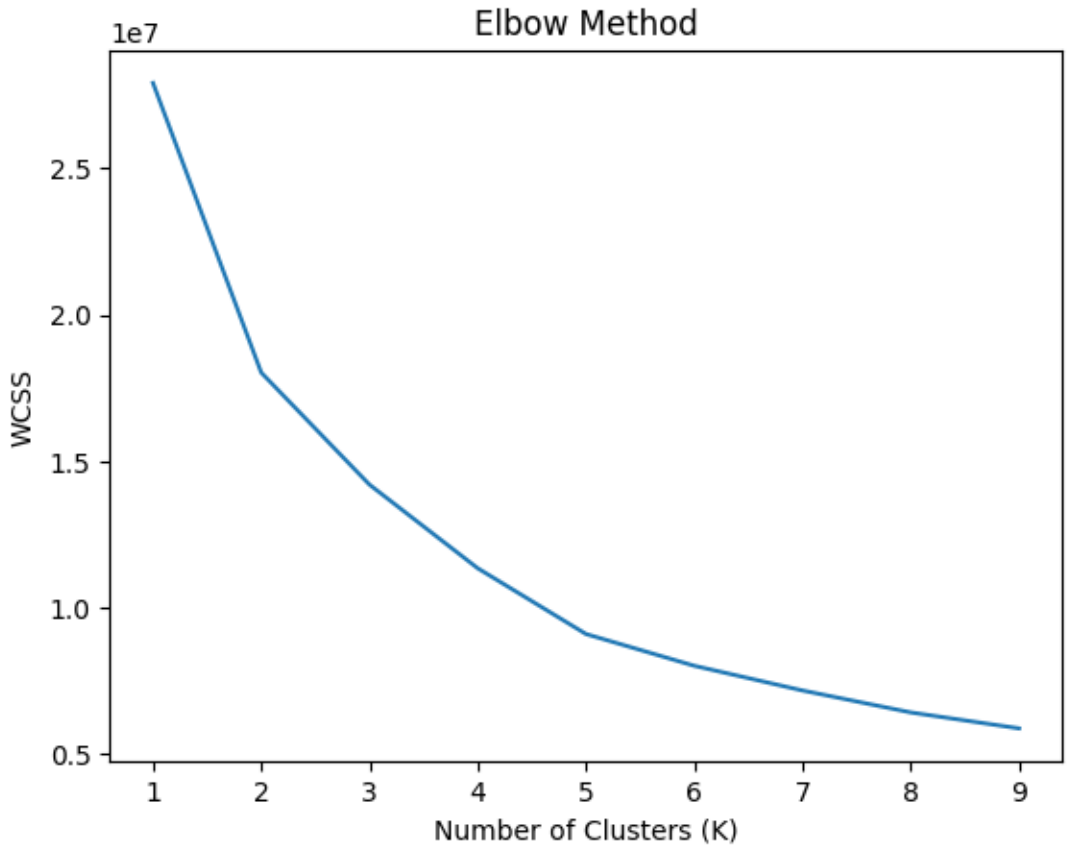
7-9 hours of sleep: +1 point, scaling down to 0 as you move away from this range

Sleep Score:

Create a normalized sleep score that peaks at 7-9 hours and decreases towards 0 or 24 hours

The simplest way could be to use a Gaussian-like function or triangular function.

Elbow method used to choose number of clusters which is 5:



KMeans Initialization:

- Here, we are creating an instance of the KMeans clustering model with specific parameters.

- Number of Clusters: n_clusters=5 specifies that we want to create 5 clusters in our data.

Initialization Method:

- init='k-means++' indicates the use of the 'k-means++' method for initializing the cluster centroids. This initialization method often leads to faster convergence and more accurate results than random initialization.

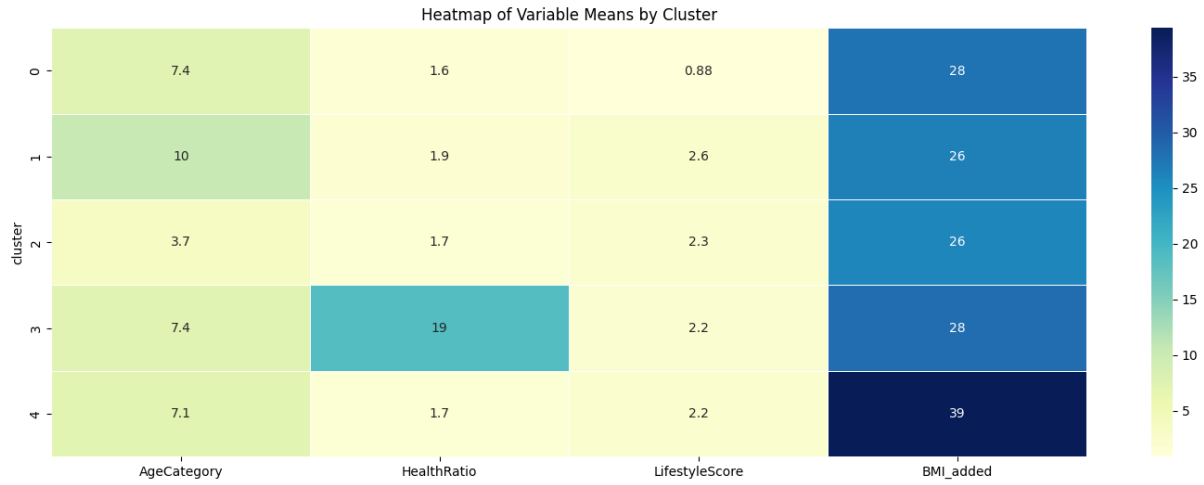
Random Seed:

- random_state=42 sets a fixed random seed to ensure reproducibility. This means that the same results will be obtained when running the code with the same seed.

Model Fitting:

- kmeans.fit_predict(features_data) fits the KMeans model to the features_data and simultaneously assigns cluster labels to each data point based on its proximity to the cluster centroids.

Here is the Heatmap of Variable Means by Cluster:



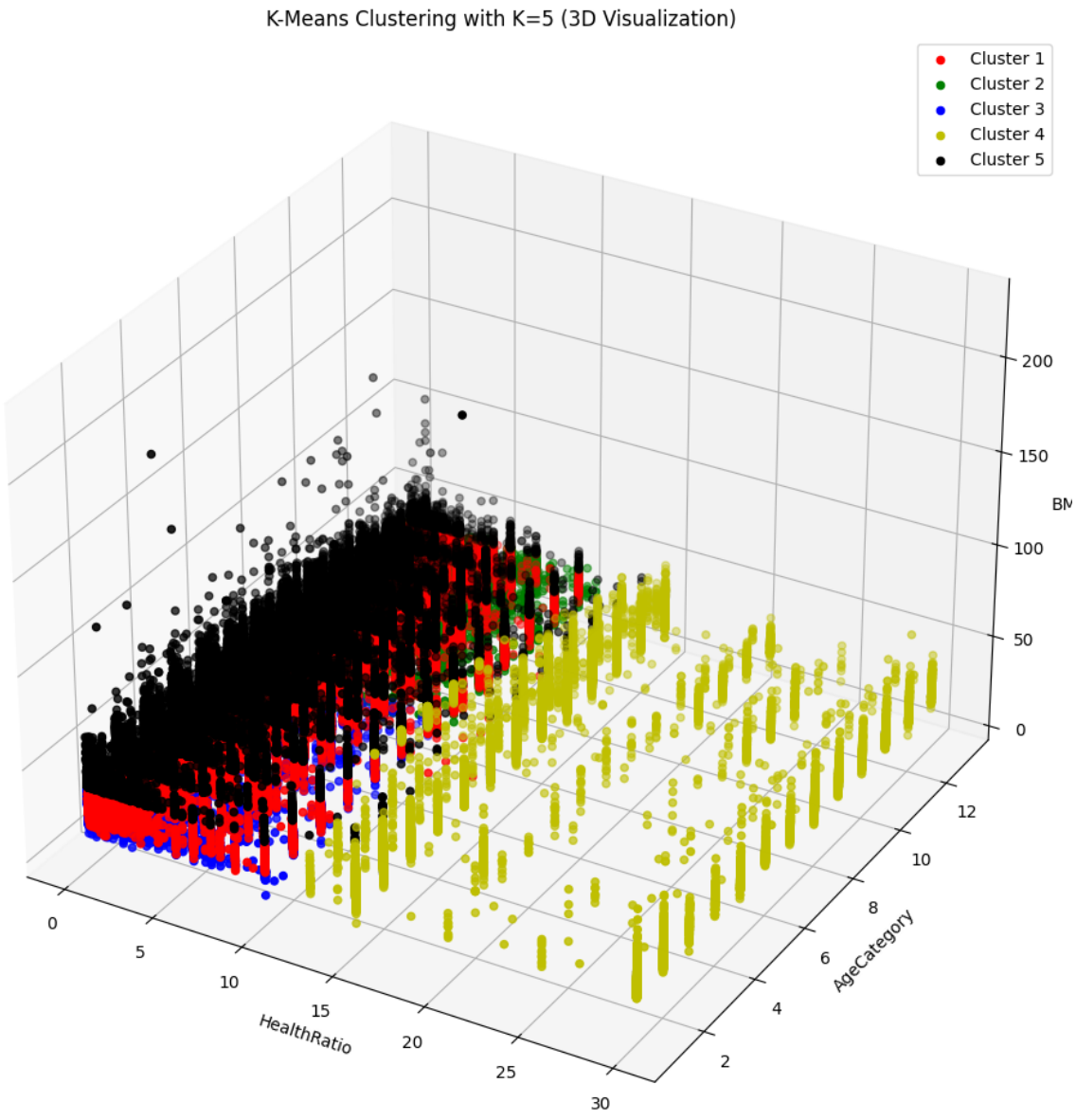
Cluster 0: exhibits moderate mean values across all the variables, suggesting a relatively balanced profile with no extreme characteristics. The BMI mean is slightly elevated, which may indicate a trend towards overweight conditions in this group.

Cluster 1: stands out with the highest mean age category, suggesting that this cluster predominantly consists of older individuals. The lifestyle score and BMI are also higher than most other clusters, potentially indicating a correlation between age and these factors.

Cluster 2: shows the lowest mean age category, which suggests it is composed of younger individuals. The health ratio and lifestyle scores are moderate, but the BMI is on the higher side, similar to Cluster 0.

-Cluster 3: is characterized by a significantly high health ratio, far exceeding the other clusters, which could imply a greater prevalence of health-related issues or concerns. The lifestyle score is relatively low, while the BMI is moderate.

Cluster 4: presents a unique profile with the highest BMI mean, indicating that individuals in this group may have the highest average body mass index, potentially placing them at a greater risk for health issues associated with obesity. Other variables like age category and health ratio are moderate, while the lifestyle score is consistent with Clusters 2 and 3.



Principal Component Analysis:

Principal Component Analysis (PCA) is a dimensionality reduction technique and a fundamental tool in the fields of data analysis, machine learning, and statistics. It's used to reduce the dimensionality of high-dimensional data while preserving as much of the variance in the data as possible. Here are some key points about PCA:

1 - Dimensionality Reduction:

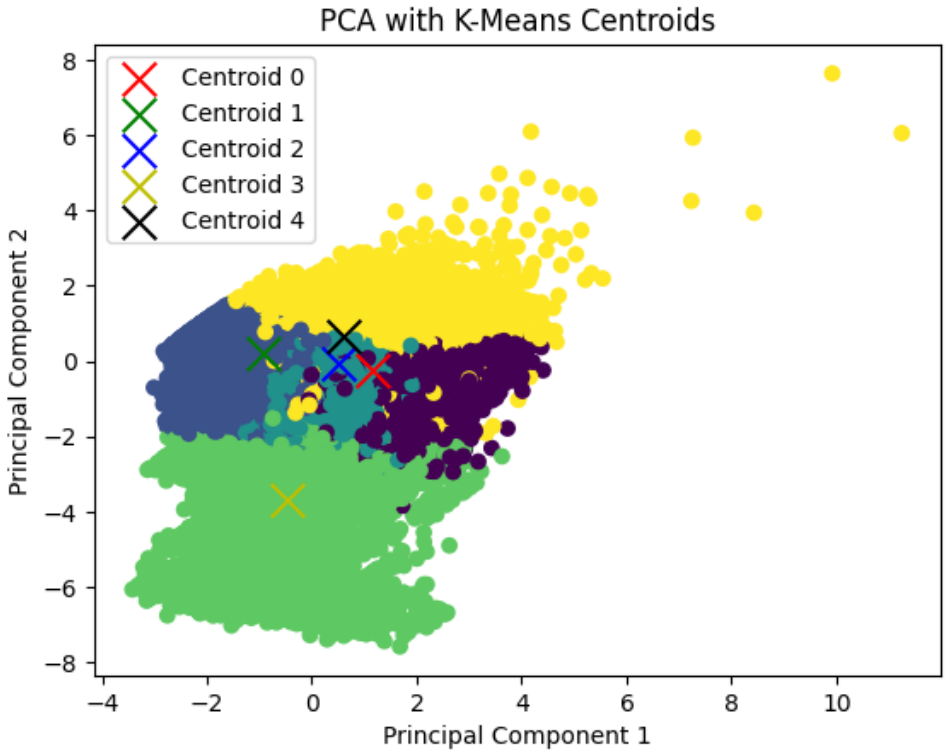
PCA is primarily used for reducing the dimensionality of datasets. It transforms a dataset with a potentially large number of features (or dimensions) into a dataset with fewer features, called principal components.

2 - Variance Maximization:

PCA identifies the axes (principal components) in the data that capture the most variance. The first principal component accounts for the most variance, the second captures the second most, and so on. This allows for dimensionality reduction while retaining as much information as possible.

Goal of PCA is to reduce the dimensionality of the data set by eliminating the redundancy of information resulting from p highly correlated variables and by replacing the latter with a smaller number h (with h < p) of new variables that are not correlated with each other and linearly linked to the starting variables.

In addition to being uncorrelated, the new variables are ordered with respect to the percentage of variability present in the original data.



Cluster Interpretation with Centroids:

Light Green Cluster:

Centroid: Slightly below average on both principal components.
Interpretation: This cluster may represent individuals who are younger (AgeCategory mean ~ 7.39) with a moderate health ratio (mean ~ 1.63), the lowest lifestyle scores (mean ~ 0.88), and slightly elevated BMI (mean ~ 27.66). These could be individuals with relatively fewer health concerns but with lifestyle habits that might need improvement.

Dark Green Cluster:

Centroid: Near the average on Principal Component 1 and slightly below average on Principal Component 2.

Interpretation: Representing a mix of young and middle-aged individuals (AgeCategory mean ~ 7.35), this cluster has the highest health ratio (mean ~ 18.74), indicating significant health concerns or frequent healthcare engagement, combined with relatively lower lifestyle scores (mean ~ 2.16) and a slightly higher BMI (mean ~ 28.25). The high health ratio is a particularly notable feature of this cluster.

Yellow Cluster:

Centroid: Above average on Principal Component 1 and moderate on Principal Component 2.
Interpretation: Likely consists of the oldest individuals (AgeCategory mean ~ 10.44) with a high health ratio (mean ~ 1.89), the highest lifestyle scores (mean ~ 2.55), and a slightly lower BMI (mean ~ 26.43). The combination of high age and lifestyle score may suggest these individuals are actively engaged in maintaining their health despite older age.

Blue Cluster:

Centroid: Average on both principal components.
Interpretation: This cluster is similar in age to the light green cluster (AgeCategory mean ~ 7.10) with a moderate health ratio (mean ~ 1.72), a fairly healthy lifestyle score (mean ~ 2.24), and significantly higher BMI (mean ~ 39.36), which is the most notable characteristic. The high BMI suggests that these individuals may be at a higher risk of obesity-related health complications.

Purple Cluster:

Centroid: Slightly above average on Principal Component 1, below average on Principal Component 2.
Interpretation: The youngest cluster (AgeCategory mean ~ 3.68) with a moderate health ratio (mean ~ 1.67) and lifestyle score (mean ~ 2.34), and the lowest BMI (mean ~ 25.93). The low BMI and younger age suggest that this cluster might be at a lower immediate risk for additional health complications.

Classification and Regression

Matrix Evaluation:

There are four numbers in the matrix:

True Negative (TN): The count of cases correctly identified as not having heart disease.

False Positive (FP): The cases incorrectly predicted as having heart disease when they don't.

False Negative (FN): The cases incorrectly predicted as not having heart disease when they do.

True Positive (TP): The cases correctly identified as having heart disease.

Model Evaluation:

Accuracy can be derived from the confusion matrix (although the exact numbers are not visible) and is calculated as (TP + TN) / (TP + TN + FP + FN).

Recall (Sensitivity):

Recall is the proportion of actual positives that were correctly identified.

It is calculated as the number of true positives divided by the total number of actual positives (true positives plus false negatives).

Mathematically, Recall = TP / TP + FN

High recall indicates that the model is good at detecting the positive cases.

Specificity, or the true negative rate, measures the model's ability to identify negative cases and is calculated as TN / (TN + FP).

Precision:

Precision refers to the proportion of positive identifications that were actually correct.

It is calculated as the number of true positives divided by the total number of positive predictions (true positives plus false positives).

Mathematically, Precision = TP / TP + FP

High precision indicates that the model has a low rate of false positives.

F1-Score:

The F1-score is the harmonic mean of precision and recall, providing a balance between them.

It is particularly useful when you want to find a balance between precision and recall, and there is an uneven class distribution (i.e., one class is rare).

Mathematically, F1-score = 2 x ((precision x recall) / (precision + recall)).

The F1-score ranges from 0 to 1, where 1 is the best possible score.

Support:

Support refers to the number of actual occurrences of the class in the specified dataset.

For each class, it shows how many instances of the class were present in the dataset that the model tried to predict.

It is important for assessing the performance of the model across classes that may not be equally represented.

In summary:

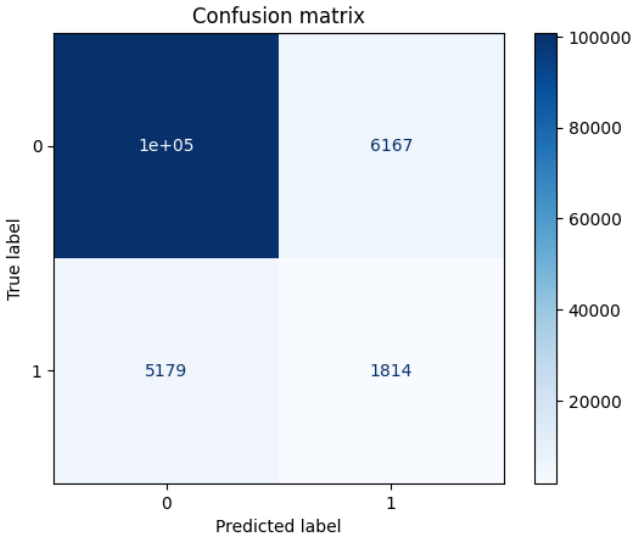
Precision is about being precise, i.e., how many of the predicted positives are truly positive.

Recall is about comprehensively capturing positives, i.e., how many positives were captured out of all actual positives.

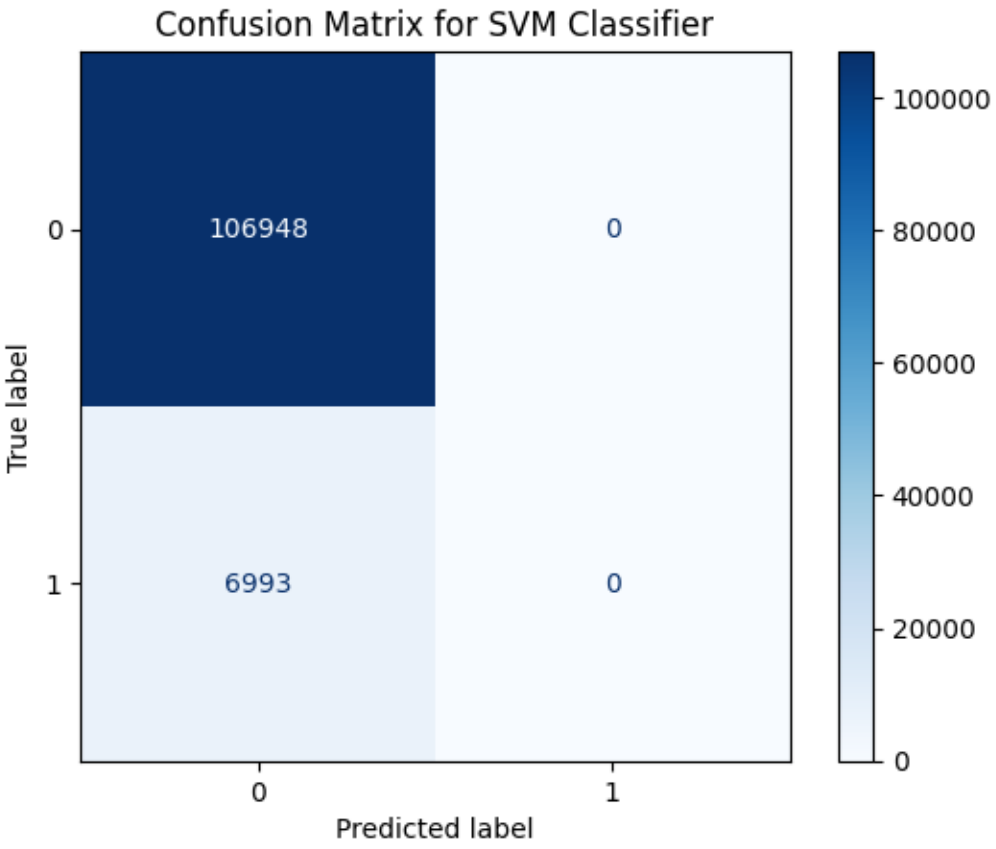
F1-Score provides a single metric that combines both precision and recall.

Support gives the actual number of occurrences of each class, which helps to understand the distribution of classes in the dataset.

Result of Decision Tree and SVM for the imputed dataset:



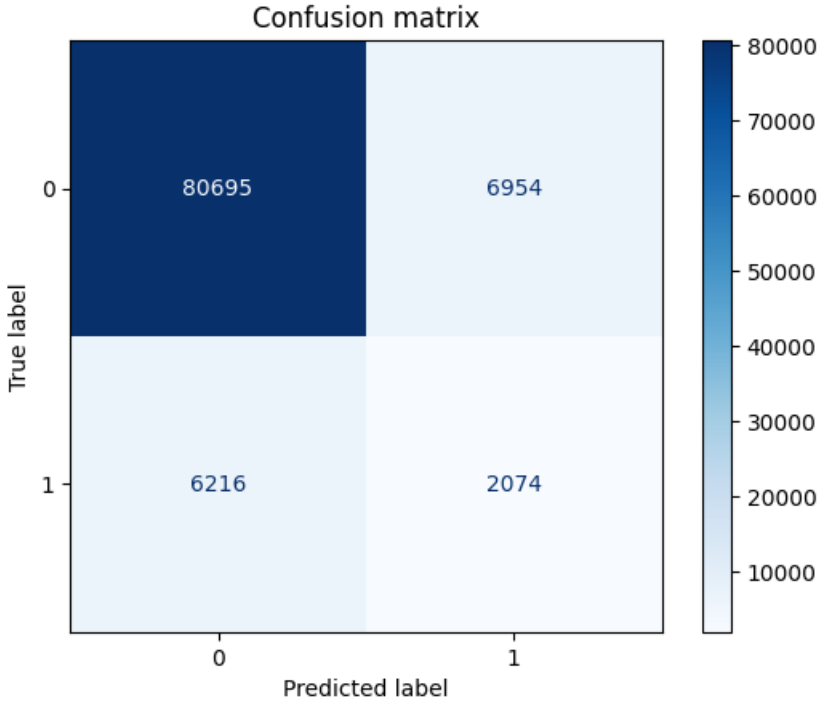
The model seems to have a high number of TNs, suggesting good TN rate. However, the FNs are quite high, which may indicate a trade-off with TP rate. This could imply that while the model is good at predicting the absence of disease, it may not be as effective at correctly identifying all the true cases of heart disease. It's essential to balance these metrics, especially in medical diagnoses, to avoid false assurances or unnecessary anxiety.



Class '0' (No Heart Disease): The model has a high precision of 0.94, meaning that when it predicts the negative class, it is correct 94% of the time. The recall is 1.00, indicating that the model identified all actual negatives correctly. The F1-score is 0.97, which is a weighted average of precision and recall and suggests excellent model performance for this class.

Class '1' (Heart Disease): The model has a precision and recall of 0.00. This means the model failed to correctly predict any of the positive cases; it never predicted '1' when it was the true label.

Result of Decision Tree and SVM for the Kaggle dataset:



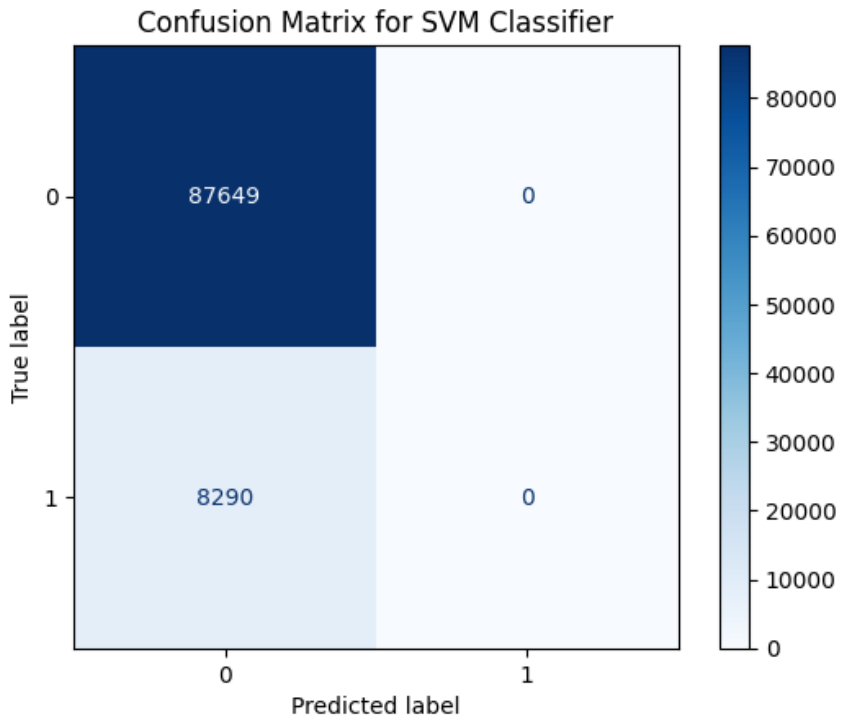
In summary, while this Decision Tree model performs reasonably well in identifying patients without heart disease, its ability to identify patients with heart disease remains a significant concern. The low precision and recall for the positive class could lead to both over-treatment for healthy patients and under-treatment for those with the disease. Addressing the class imbalance and potentially exploring different algorithms or feature selection methods could improve the model's performance in predicting heart disease more accurately. Enhancing the recall for class 1 should be a priority, as missing out on true cases of heart disease can have serious health implications for patients.

Summary:

The model trained on the imputed dataset seems to be slightly better at generalizing, as indicated by the higher accuracy and lower false negative rate.

The model trained on the Kaggle dataset, while having a lower overall accuracy, is better at identifying true cases of heart disease, as seen by the higher number of true positives.

Both models struggle with the minority class (class 1), which could be due to the class imbalance. The high false positive rates in both models also suggest that many patients without heart disease would be incorrectly identified as having it, potentially leading to unnecessary worry and medical tests.



The classifier made a total of 87,649 + 8,290 = 95,939 predictions.

87,649 were true negatives (TN): the classifier correctly predicted the negative class.

There were 8,290 false negatives (FN): the classifier incorrectly predicted the negative class when it was actually the positive class.

There were 0 true positives (TP): the classifier did’nt correctly predict any of the positive class.

There were also 0 false positives (FP): the classifier did not incorrectly predict the positive class when it was actually the negative class.

From this, we can infer that the classifier predicted every instance as the negative class. This could indicate a highly imbalanced dataset or a model that is biased towards predicting the negative class. There are no instances of the positive class being predicted, which is a critical issue if the positive class is of interest, such as in medical diagnosis or fraud detection.

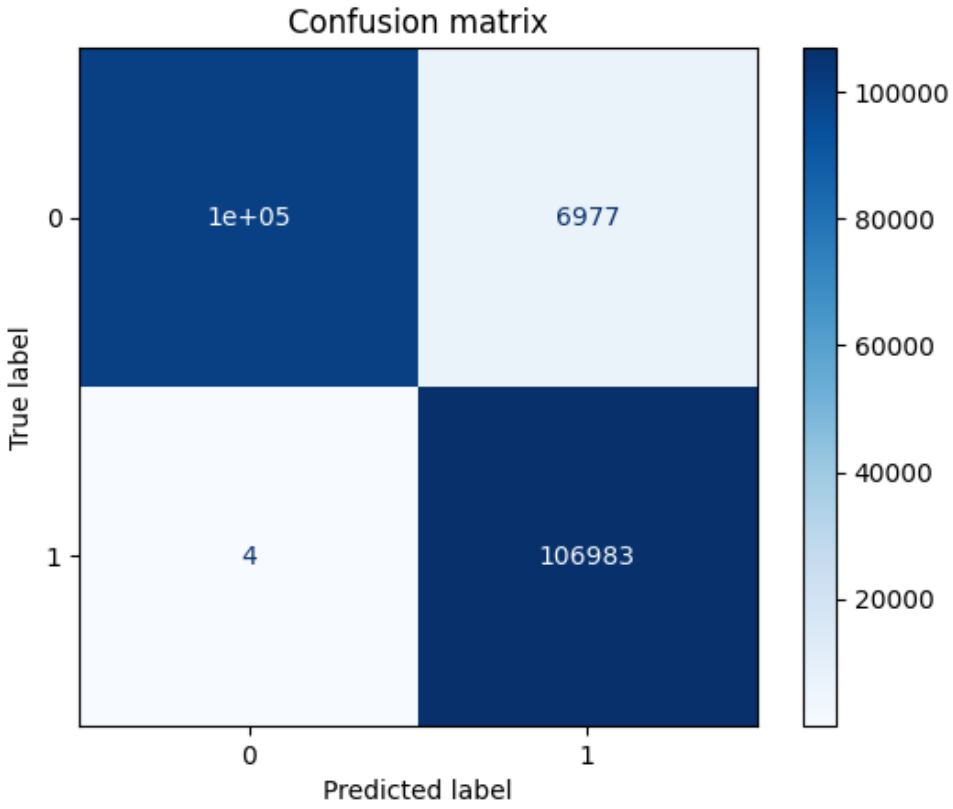
This model would have a high specificity (true negative rate) but a very low sensitivity (true positive rate), and the precision for the positive class cannot be calculated as there are no positive predictions (division by zero). The F1-score for the positive class would be 0, indicating that the model's performance on the positive class is poor.

Oversampling:

Oversampling is a technique used in the field of machine learning and data analysis to address class imbalance in a dataset. Class imbalance occurs when one or more classes or categories in a classification problem have significantly fewer instances than others. In the context of oversampling, it typically refers to the practice of increasing the number of instances in the minority class(es) to balance the class distribution.

Here is the result of Decision Tree on the oversampled imputed dataset:

```
Test Set Score after Sampling:  96.74%
Train Set Score after Sampling:  99.92%
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.93      0.97    106963
           1       0.94      1.00      0.97    106987

    accuracy                           0.97    213950
   macro avg       0.97      0.97      0.97    213950
weighted avg       0.97      0.97      0.97    213950
```



Test and Train Set Score:

The accuracy on the test set has increased to 96.74%, which is a significant improvement over the previously noted accuracy.

The accuracy on the train set is very high at 99.92%, suggesting that the model fits the training data very well.

Classification Report:

For Class '0' (No Heart Disease), the precision is perfect at 1.00, indicating no false positives. The recall is 0.93, which is also very high, indicating that the model correctly identified 93% of all actual negatives.

For Class '1' (Heart Disease), the precision is 0.94, meaning that when the model predicts the positive class, it is correct 94% of the time. The recall is perfect at 1.00, indicating the model identified all actual positives correctly.

The F1-scores for both classes are 0.97, suggesting a balanced and excellent performance in terms of precision and recall for both classes.

The macro average and weighted average scores across precision, recall, and F1 are all 0.97, which supports the model's robustness after addressing the class imbalance.

Confusion Matrix:

The confusion matrix presents a more balanced picture after oversampling. There are 106,963 true negatives and only 6,977 false positives, indicating some cases where the model incorrectly predicted heart disease when there was none.

There are 106,983 true positives and only 4 false negatives, showing an exceptional improvement in the model's ability to identify positive cases accurately.

Analysis:

Class Balance: The oversampling has corrected the class imbalance, and the Decision Tree model is now performing well on both classes, as evidenced by the high recall for the positive class.

Model Performance: The model has become much more effective at predicting both classes, with high precision and recall for both. This indicates that oversampling has had a positive impact on the model's ability to generalize.

Conclusion:

After oversampling, the Decision Tree classifier shows a vastly improved ability to correctly classify both negative and positive cases of heart disease. The precision, recall, and F1-scores are all high, and the confusion matrix indicates a significant reduction in false negatives, which

is critical for medical diagnosis applications where missing a positive case can have serious consequences.

Logistic Regression for Imputed Dataset:

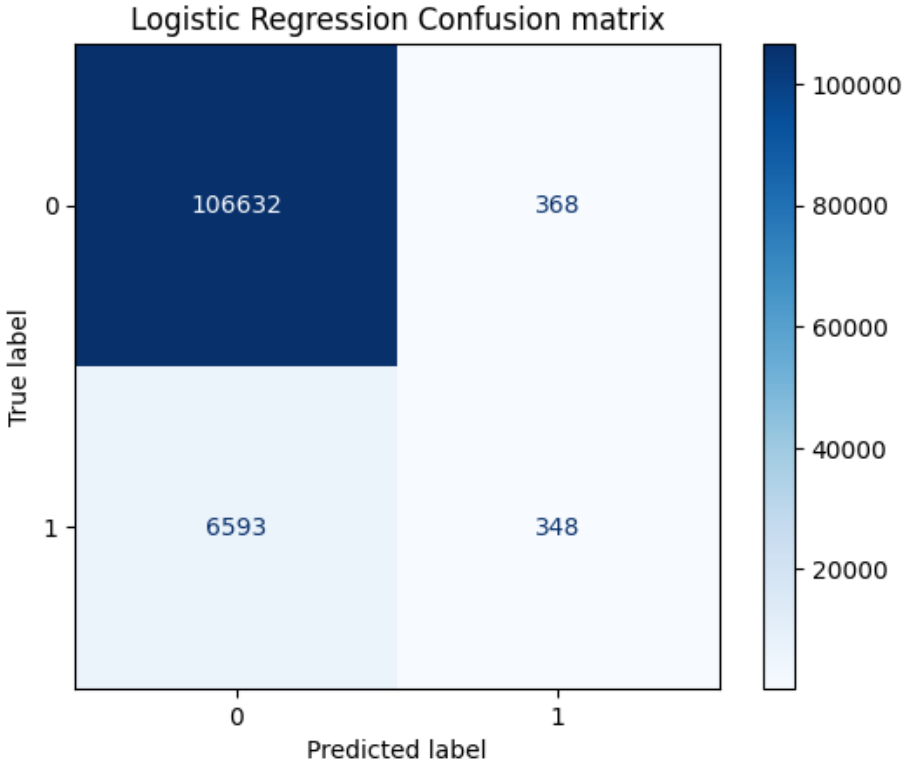
Accuracy: The model exhibits high consistency between the train set accuracy (93.86%) and test set accuracy (93.89%), which is a good indicator that the model generalizes well and is not overfitting.

F1 Score: The F1 score of 91.49% is quite high, indicating a good balance between precision and recall. This is a harmonic mean of the two, and a high score is desirable.

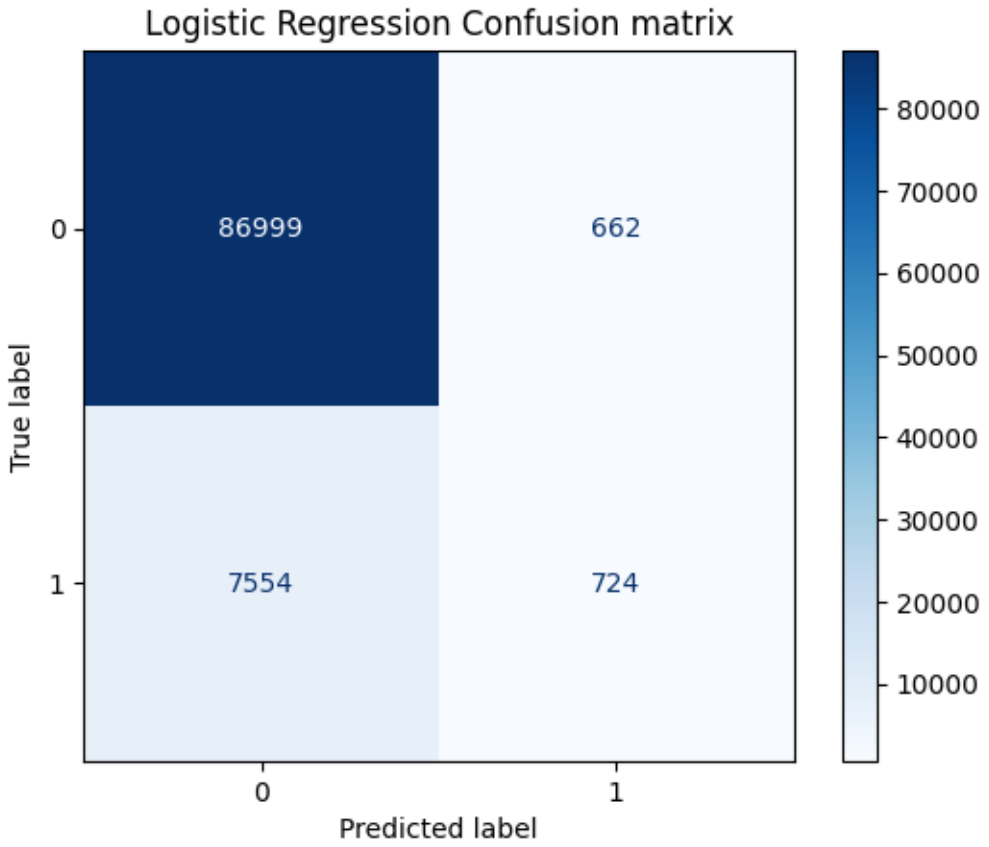
Confusion Matrix: The confusion matrix shows a large number of True Negatives (TN = 106,632), indicating that the model is proficient at identifying individuals without heart disease.

There are a relatively low number of True Positives (TP = 348), suggesting that while the model is good at identifying negatives, it struggles with detecting the positive cases of heart disease.

The False Positives (FP = 368) and False Negatives (FN = 6,593) indicate that there are still quite a few misclassifications, especially in missing out the true cases (FN), which is critical in a medical diagnosis context.



Logistic Regression for Kaggle Dataset:



Accuracy: The train set accuracy is at 91.55%, and the test set accuracy is slightly lower at 91.44%. The small difference between these two suggests that the model generalizes well.

F1 Score: The F1 score is 88.54%, which is a good score but slightly lower than what was observed for the imputed dataset.

Confusion Matrix: There are 86,999 True Negatives (TN), indicating the model's effectiveness at identifying individuals without heart disease.

True Positives (TP) are 724, which is higher than what was observed for the imputed dataset. This suggests that the model is somewhat better at identifying positive cases in the Kaggle dataset.

False Positives (FP) are relatively low at 662, but False Negatives (FN) are quite high at 7,554, indicating that the model still misses a significant number of positive cases.

In conclusion, the Logistic Regression model on the Kaggle dataset shows a robust ability to predict heart disease, with some improvement in identifying true positive cases compared to the imputed dataset. However, the high number of False Negatives in both models indicates a need for further improvement, especially considering the serious implications of failing to detect heart disease. The feature importance analysis provides insights into risk factors for heart disease, which can inform healthcare professionals and policymakers in designing interventions and preventive measures.

Multi Layer Perceptron:

Multi layer perceptron (MLP) is a supplement of feed forward neural network. It consists of three types of layers the input layer, output layer and hidden layer. The input layer receives the input signal to be processed. The required task such as prediction and classification is performed by the output layer. An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the MLP. Similar to a feed forward network in a MLP the data flows in the forward direction from input to output layer. The neurons in the MLP are trained with the back propagation learning algorithm.

Our data contains a mixture of categorical and numerical data. Let's use TensorFlow`s Feature Columns. Feature columns allow us to bridge/process the raw data in our dataset to fit our model input data requirements. Furthermore, we can separate the model building process from the data preprocessing.

```
# Define feature columns
feature_columns = []

# Numeric feature columns
for feature_name in num_var:
    feature_columns.append(tf.feature_column.numeric_column(feature_name, dtype=tf.float32))

# Categorical feature columns (use one-hot encoding)
for feature_name in cat_var:
    vocabulary_list = df[feature_name].unique()
    cat_column = tf.feature_column.categorical_column_with_vocabulary_list(feature_name, vocabulary_list)
    one_hot_column = tf.feature_column.indicator_column(cat_column)
    feature_columns.append(one_hot_column)

# Binary feature columns (use one-hot encoding)
for feature_name in bin_var:
    vocabulary_list = df[feature_name].unique()
    bin_column = tf.feature_column.categorical_column_with_vocabulary_list(feature_name, vocabulary_list)
    one_hot_column = tf.feature_column.indicator_column(bin_column)
    feature_columns.append(one_hot_column)
```

Training:

Our loss function is binary cross-entropy defined by:

$$-(y \log(p) + (1 - y) \log(1 - p))$$

where y is binary indicator if the predicted class is correct for the current observation and p is the predicted probability.

```
# Build and compile the model with the custom initializer for a specific layer
model = tf.keras.Sequential([
        tf.keras.layers.DenseFeatures(feature_columns),  # Input layer with feature columns
        tf.keras.layers.Dense(units=128, activation='relu', kernel_initializer=initializer),  # Hidden layer with custom initializer
        tf.keras.layers.Dropout(rate=0.2),  # Dropout layer for regularization
        tf.keras.layers.Dense(units=128, activation='relu'),  # Another hidden layer
        tf.keras.layers.Dense(units=1, activation='sigmoid')  # Output layer with sigmoid activation for binary classification
    ])

# Add the AUC metric to the list of metrics
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy', tf.keras.metrics.AUC()])
```

Input Layer (`dense_features`):

This layer is for inputting the data through feature columns. It has no parameters, it is used to pass the data through without learning any representations.

First Hidden Layer (`dense`):

A fully connected (dense) layer with 7,040 parameters. This indicates that it may have a number of neurons which, when combined with the size of the input, gives this number of parameters. Since it's the first trainable layer, its number of neurons can be deduced by dividing the number of parameters by the number of input features plus one (for the bias term).

Regularization Layer (`dropout`):

A dropout layer follows with a dropout rate which is responsible for helping prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time.

Second Hidden Layer (`dense_1`):

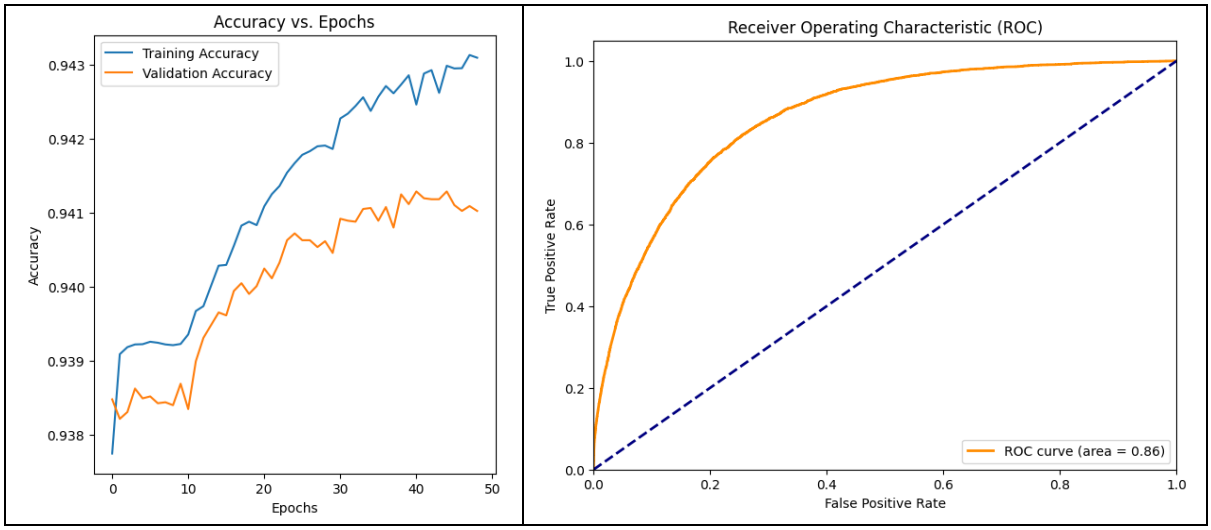
Another fully connected layer with 16,512 parameters, suggesting a substantial number of neurons that connect to the previous dense layer.

Output Layer (`dense_2`):

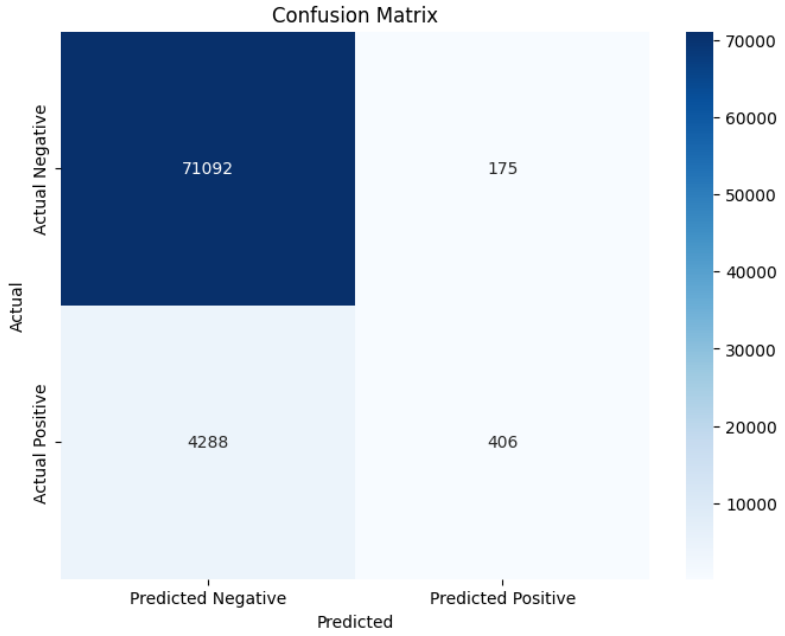
The final dense layer has 129 parameters. Since this is a binary classification model (indicated by the use of a sigmoid activation function in the provided code), it is likely that this layer has a single neuron. The 129 parameters suggest that it connects to 128 neurons from the previous layer (128 weights + 1 bias).

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_features (DenseFeatu  multiple                  0
 res)

 dense (Dense)               multiple                  7040

 dropout (Dropout)           multiple                  0

 dense_1 (Dense)             multiple                  16512

 dense_2 (Dense)             multiple                  129

=================================================================
Total params: 23681 (92.50 KB)
Trainable params: 23681 (92.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

MLP for Imputed Dataset:

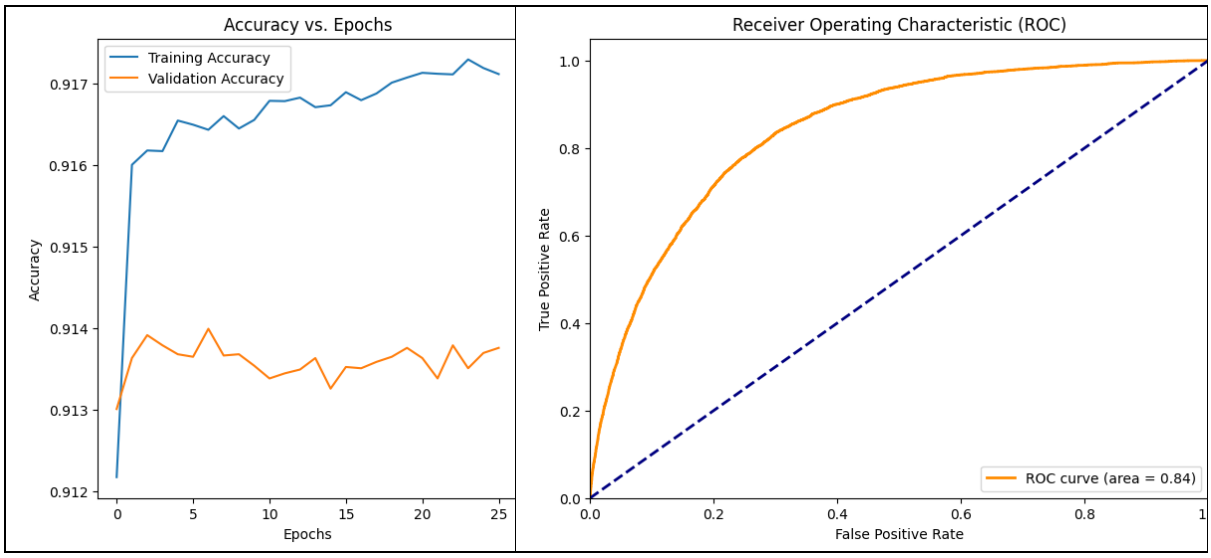


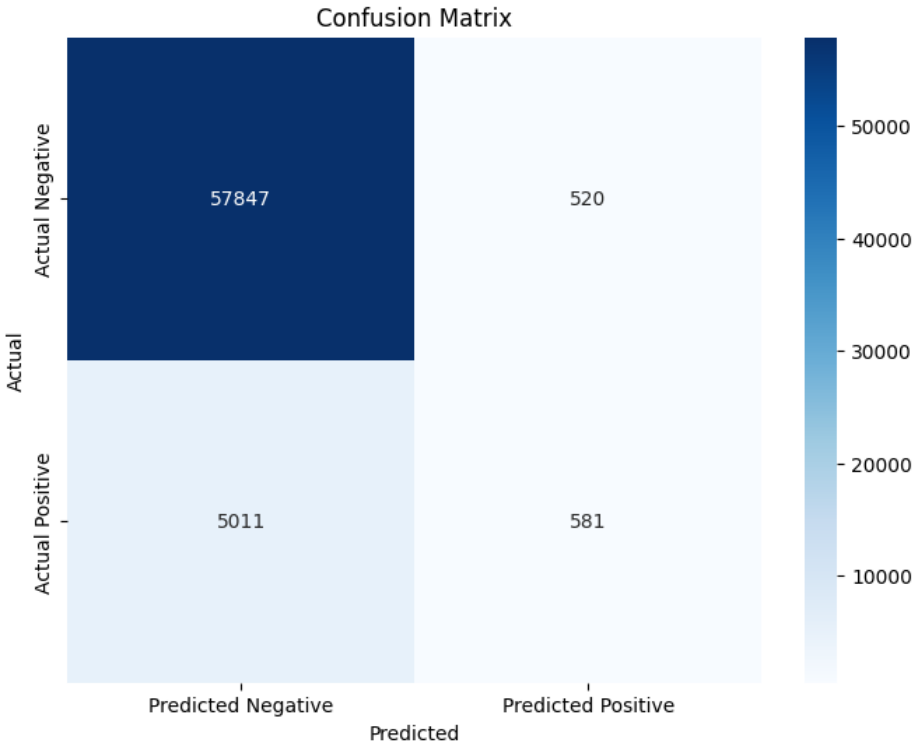
The area under the ROC curve (AUC) is a single scalar value summarizing the overall performance of the model. In this plot, the AUC is 0.86. An AUC of 0.86 indicates that the model has a good measure of separability, meaning it is able to distinguish between the positive and negative classes effectively.



Given the relatively low number of true positives (406) compared to false negatives (4,288), the model might have a low recall, indicating it struggles to identify all the positive cases. Conversely, with a small number of false positives (175), the model's precision could be quite high, indicating that when it predicts a positive result, it is likely to be correct.

MLP for Kaggle Dataset:





True Negatives (TN): 57,847 instances were correctly predicted as negative.

False Positives (FP): 520 instances were incorrectly predicted as positive when they were actually negative.

False Negatives (FN): 5,011 instances were incorrectly predicted as negative when they were actually positive.

True Positives (TP): 581 instances were correctly predicted as positive.

This confusion matrix indicates a model with a high number of true negatives and a significant number of false negatives. This suggests that the model may have a high specificity (it's good at identifying negatives), but lower sensitivity (it's not as good at identifying positives). This could imply a cautious prediction model that prefers to predict negative outcomes unless it has strong evidence for a positive one. The relatively low number of true positives compared to false negatives also suggests that the model might struggle with identifying the positive class, which could be a concern depending on the application. For instance, in a medical diagnosis context, a high number of false negatives could be very risky.

K-Fold Cross Validation:

The k-fold cross-validation method is a popular technique used in machine learning to assess a model's performance when the data sample is limited. Here is a comprehensive description based on multiple sources:

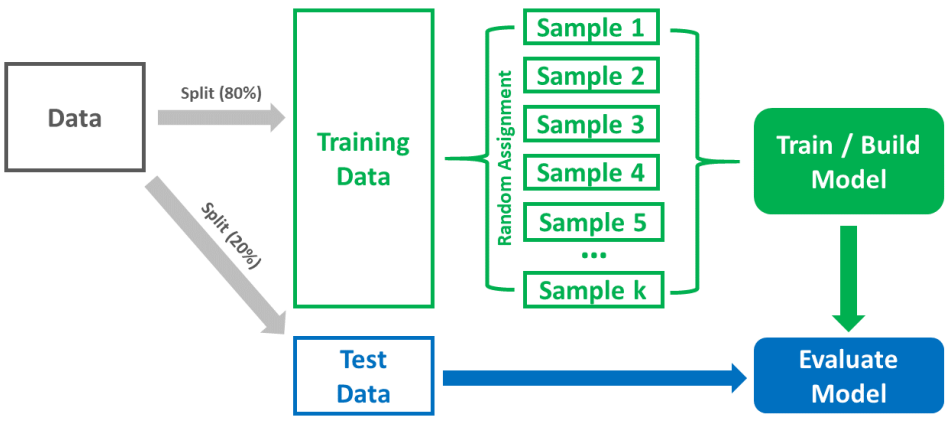
Dataset Partitioning: Initially, the dataset is randomly divided into 'k' groups, or "folds", of approximately equal size. One of these folds is chosen as the holdout set, while the rest are used for training the model.

Model Training and Evaluation: The model is trained on (k-1) folds and tested on the holdout set to calculate some performance metric (e.g., Mean Squared Error (MSE) for regression problems).

Iteration: This process is repeated 'k' times, with a different fold used as the holdout set in each iteration. As a result, every data point is part of the holdout set exactly once and part of a training fold 'k-1' times.

Performance Estimation: The performance metrics from each of the 'k' iterations are averaged to obtain a more robust estimate of model performance, which can be particularly useful when the data sample is limited3.

Configuration: The choice of 'k' is crucial as it determines the trade-off between bias and variance in the performance estimate. A common choice for 'k' is 10, although the optimal value may depend on the specific dataset and problem at hand.



Model:

```
def create_model(input_shape):
    model = Sequential([
        InputLayer(input_shape=input_shape),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

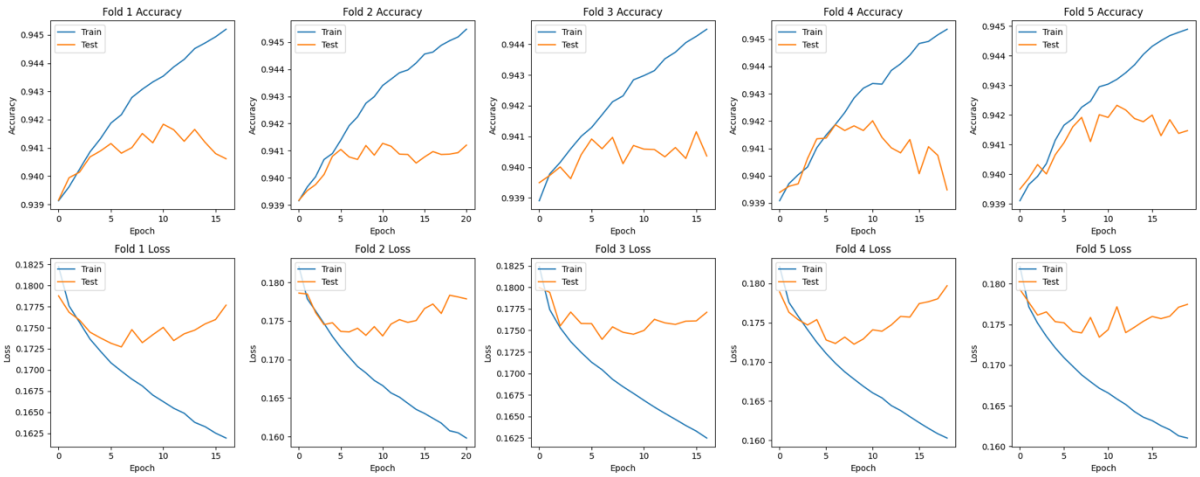
```
kfold = StratifiedKFold(n_splits=5, shuffle=True)
early_stopping = EarlyStopping(monitor='val_loss', patience=10)
```

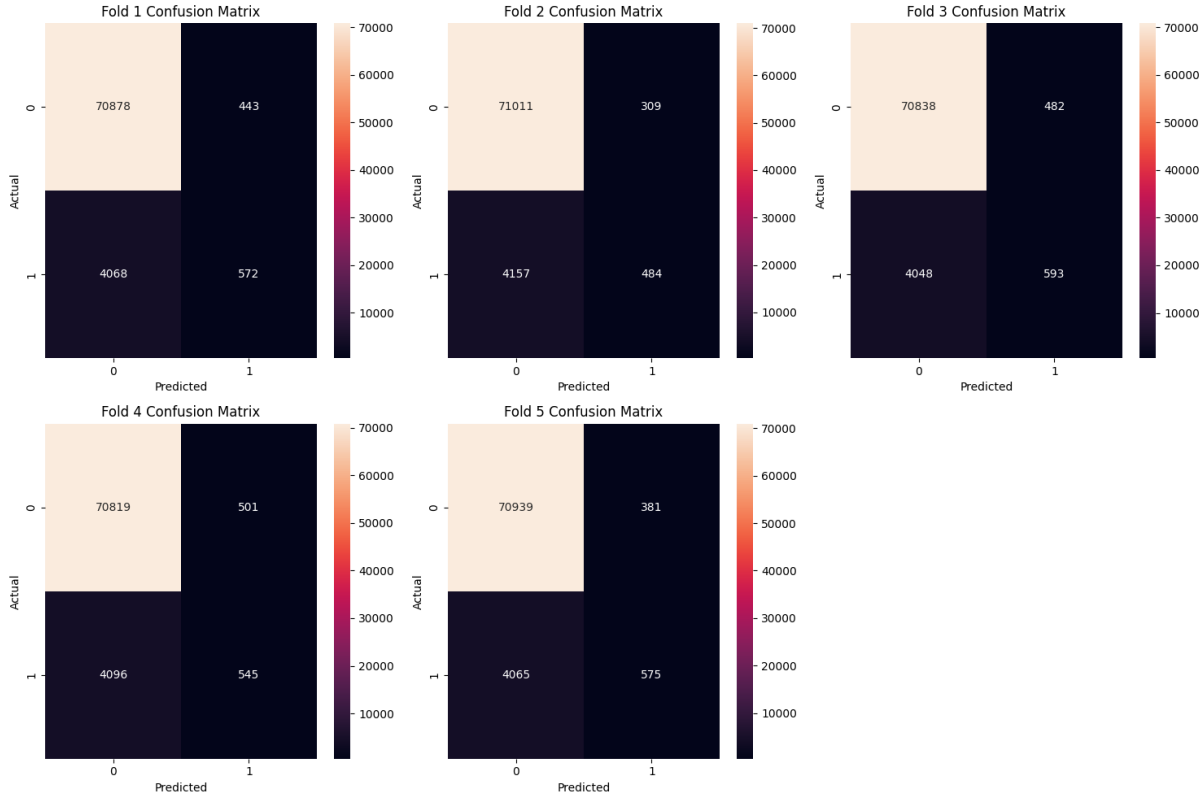
Result of K-Fold on Imputed Dataset:

The accuracy and loss graphs over the training epochs present the following observations:

Training Accuracy and Loss: Training accuracy generally increases, and training loss decreases, indicating that the MLP is learning and improving on the training data across all folds.

Testing Accuracy and Loss: Testing accuracy is less stable, showing fluctuations which may be indicative of the model's challenges in generalizing to unseen data. Testing loss also exhibits variability and does not consistently decrease, which may suggest overfitting.





True Negatives (TN): The MLP consistently predicts the negative class well across all folds, with true negatives ranging from 70,878 to 70,939.

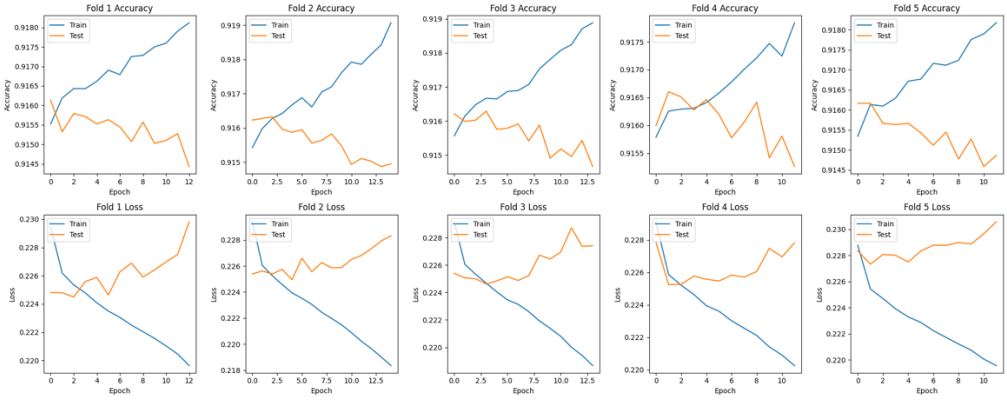
False Positives (FP): False positives remain low, between 309 to 501, indicating good specificity.

True Positives (TP): The number of true positives is relatively low, ranging from 572 to 593, suggesting that while the MLP is capable of identifying some positives, it may benefit from further tuning to improve sensitivity.

False Negatives (FN): The false negatives are significant, ranging from 4,048 to 4,157, which could be a point of concern if the positive class is critical (e.g., in disease detection).

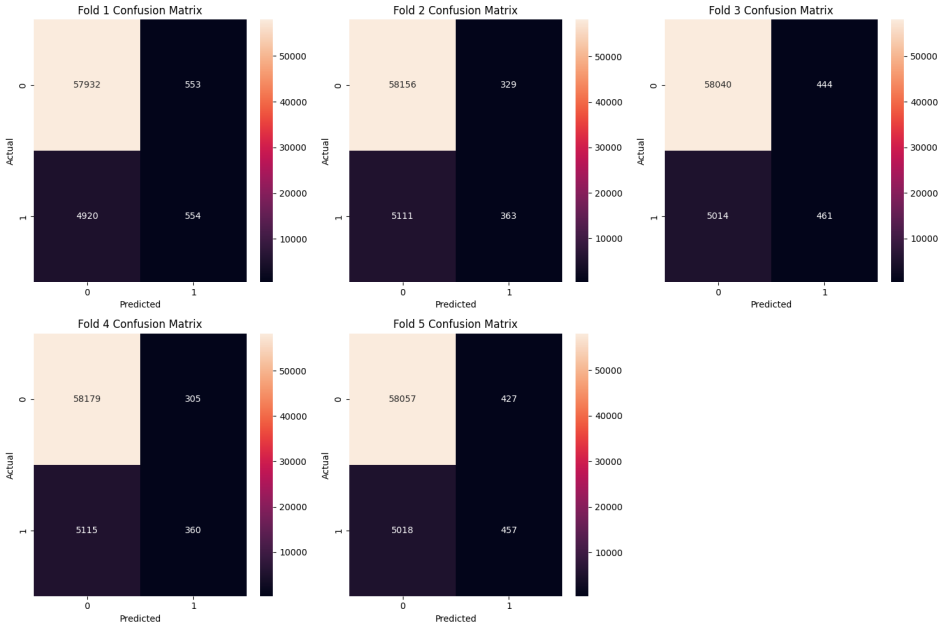
Overall Evaluation: The MLP classifier shows strong performance in predicting the negative class but has room for improvement in correctly identifying the positive class. The accuracy and loss trends suggest the model is learning from the training data; however, the fluctuation in test metrics indicates potential overfitting or instability in generalization.

K-Fold for Kaggle Dataset:



Training Accuracy: For all folds, the training accuracy improves consistently as the epochs progress, which suggests that the model is learning effectively from the training data.

Testing Accuracy: The testing accuracy exhibits some fluctuations across different folds and epochs, which is common in validation curves but may also indicate a degree of variability in how the model performs on unseen data.



True Negatives (TN): Each fold shows a high count of true negatives, indicating the model is adept at identifying the absence of heart disease.

False Positives (FP): The false positives remain relatively low across all folds, which is positive as it suggests few instances where the model incorrectly predicts the presence of heart disease.

True Positives (TP): The model captures a moderate number of true positives; however, there is room for improvement in correctly identifying all instances of heart disease.

False Negatives (FN): There is a notable count of false negatives in each fold, suggesting that the model might be prone to missing some instances of heart disease.

Overall Model Assessment:

The MLP classifier demonstrates strong performance in identifying the absence of heart disease but appears to struggle with consistently identifying its presence. The relatively high number of false negatives indicates that the model may benefit from further optimization to improve its sensitivity.

The accuracy and loss trends suggest that the model's performance is acceptable but might be improved, particularly in terms of generalization to the test data, as suggested by the variability in the testing accuracy and loss.

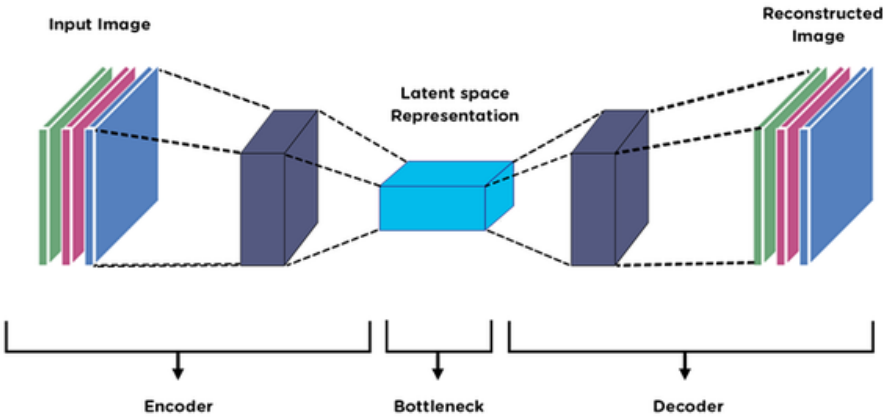
AutoEncoder:

Autoencoders are a special kind of neural network utilized for unsupervised machine learning tasks, particularly those involving dimensionality reduction and anomaly detection. Here are the core aspects of autoencoders, synthesized from multiple sources:

Basic Functionality: An autoencoder is designed to learn to copy its input to its output through a compressed representation. It first encodes the input into a lower-dimensional latent space, and then decodes this representation back to the original input format.

Architecture: The architecture comprises three main parts: the encoder, the code, and the decoder. The encoder compresses the input data to form a code, the decoder then reconstructs the original data from this code.

Learning Method: Autoencoders learn data encodings in an unsupervised manner, meaning they don't require labeled data for training. They aim to learn a lower-dimensional representation of higher-dimensional data, and are trained to capture the most significant features of the input data.



Model:

```
class Autoencoder(tf.keras.Model):
  def __init__(self, feature_count):
    self.feature_count = feature_count
    super(Autoencoder, self).__init__()
    self.encoder = tf.keras.Sequential([
      tf.keras.layers.Dense(self.feature_count, activation="relu"),
      tf.keras.layers.Dense(20, activation="relu"),
      tf.keras.layers.Dense(16, activation="relu"),
      tf.keras.layers.Dense(12, activation="relu")])

    self.decoder = tf.keras.Sequential([
      tf.keras.layers.Dense(12, activation="relu"),
      tf.keras.layers.Dense(16, activation="relu"),
      tf.keras.layers.Dense(20, activation="relu"),
      tf.keras.layers.Dense(self.feature_count, activation="linear")])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded
```

Model Inheritance: The Autoencoder class inherits from `tf.keras.Model`, indicating it is a custom model built upon TensorFlow's Keras base class.

Initialization: The `__init__` method initializes the Autoencoder with a `feature_count` argument, which sets the number of neurons in the input and output layers of the model.

Encoder: The encoder part of the model is a `Sequential` model consisting of Dense layers. It starts with a layer that has the same number of neurons as features (`feature_count`) and uses the ReLU activation function. This is followed by layers with diminishing numbers of neurons: 20, 16, and finally down to 12 neurons, each using ReLU activation functions. This represents the encoding path that compresses the input data into a lower-dimensional representation.

Decoder: The decoder is also a `Sequential` model that mirrors the encoder's architecture but in reverse, gradually increasing the number of neurons from 12 back up to `feature_count`. The layers in the decoder use ReLU activation functions, except for the final layer, which uses a linear activation function. This is typical for an output layer that needs to reconstruct the original input data from the encoded representation.

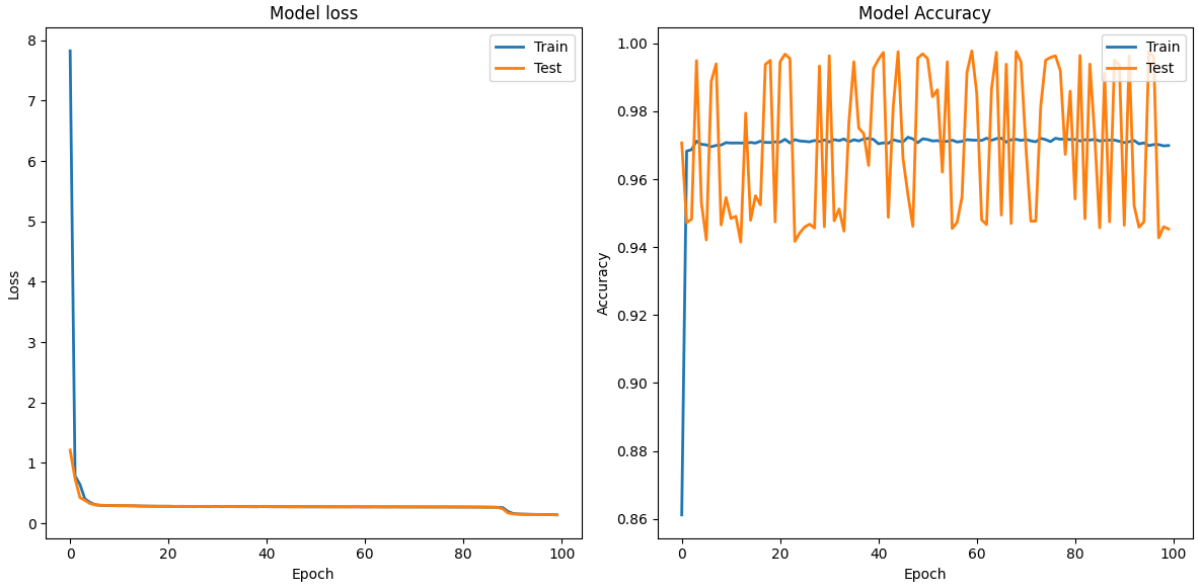
Forward Pass (`call` method): The `call` method defines the forward pass of the model. It takes an input `x`, passes it through the encoder to generate the encoded representation, and then through the decoder to reconstruct the input from the encoded representation.

Model Output: The output of the model is the reconstructed data, which ideally should be as close as possible to the original input data.

The Autoencoder model is typically used for dimensionality reduction or feature learning. In this case, the model is designed to compress data down to a 12-dimensional representation and then reconstruct the original data from this compressed form. The effectiveness of this model would depend on its ability to accurately reconstruct the input data, which can be assessed using a loss function such as mean squared error during training.

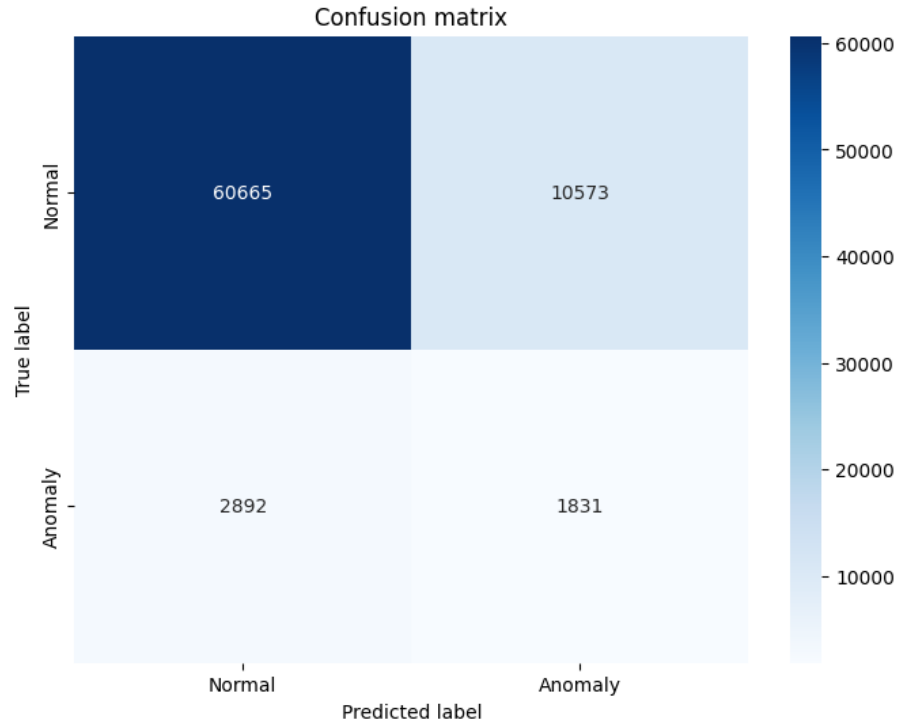
Result of AutoEncoder for Imputed Dataset:

Records in Diseased Train Data = 18496
Records in Normal Train Data = 285345
Records in Diseased Test Data = 4723
Records in Normal Test Data= 71238



Model Loss: The 'Model Loss' plot demonstrates a sharp decrease in loss for both the training (blue line) and testing (orange line) datasets as the number of epochs increases. The initial loss values start quite high but rapidly descend, indicating that the model is quickly learning from the data. The loss for both datasets seems to converge as the number of epochs increases, suggesting that the model is stabilizing and generalizing well without overfitting, as the test loss is decreasing in tandem with the training loss and both are plateauing at a similar rate after around 20 epochs.

Model Accuracy: The 'Model Accuracy' plot shows a rapid increase in accuracy during the initial epochs for both the training and testing datasets. The training accuracy (blue line) appears to plateau close to 1.0 (or 100%), which suggests that the model is nearly perfectly fitting the training data. The testing accuracy (orange line) also increases similarly and oscillates slightly above and below the training accuracy, indicating the model is performing well on the unseen data as well. Notably, the testing accuracy does not significantly diverge from the training accuracy, which would typically be a sign of overfitting.



True Positive (TP): 60665 cases were normal and correctly identified as normal.

False Negative (FN): 10573 cases were normal but incorrectly identified as anomalies.

False Positive (FP): 2892 cases were anomalies but incorrectly identified as normal.

True Negative (TN): 1831 cases were anomalies and correctly identified as such.

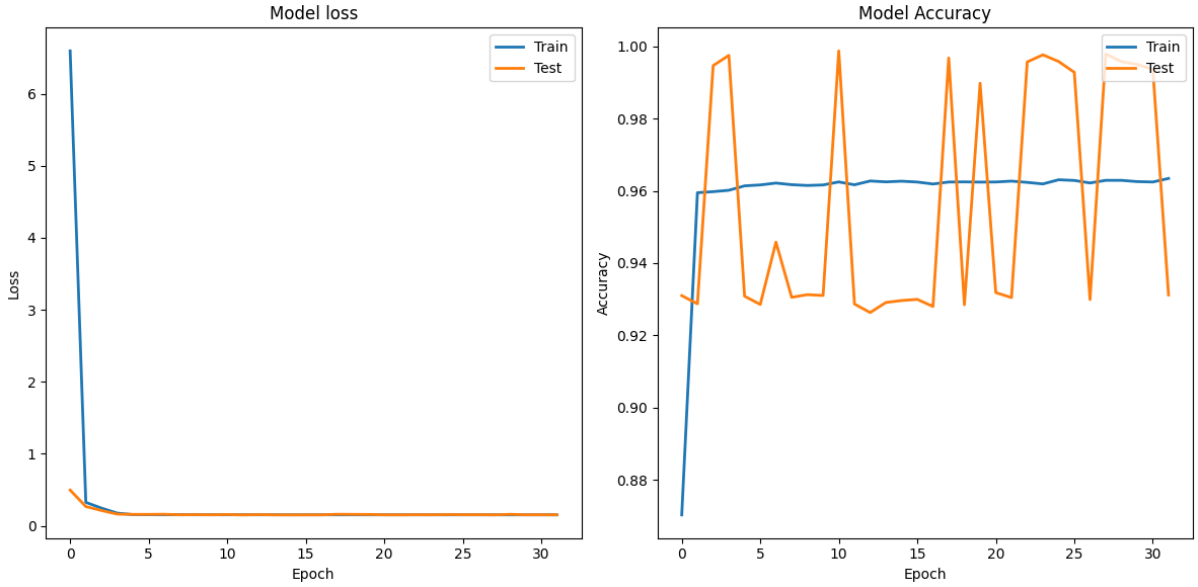
These values show the model is quite good at identifying normal cases (high TP), it struggles with a significant number of false negatives, indicating that normal cases are often misclassified as anomalies. It also has a considerable number of false positives, where anomalous cases are missed and labeled as normal.

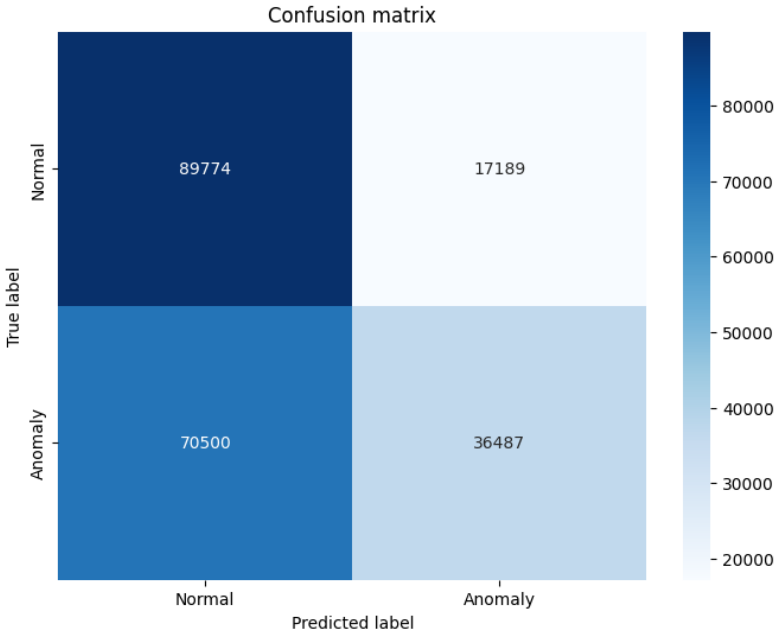
Result of AutoEncoder on Oversampled Imputed Dataset:

Model Loss: The loss on both the training and test sets decreases sharply in the initial epochs, suggesting that the model is learning and improving rapidly from its initial state.
After the initial sharp decrease, the loss curves flatten out, indicating that as epochs increase, the rate of improvement in loss reduction diminishes.
The convergence of training and test loss suggests that the model is generalizing well to unseen data. There is no sign of divergence, which would indicate overfitting.

Model Accuracy: The accuracy for both training and test sets increases sharply initially and then exhibits fluctuations throughout the remaining epochs. The training accuracy appears to be slightly higher than the test accuracy for most of the training process, which is a normal behavior as the model has direct access to the training data. The fluctuations and the close convergence of the lines suggest that the model is not overfitting significantly, which is good. It's maintaining its performance on the test set quite well throughout the training process. The

accuracy appears to be very high, generally above 95%, which is excellent. However, the effectiveness of the model should still be validated for practical use, especially considering that accuracy might not be the best metric if the original data was imbalanced.





True Negative shows the count of normal instances correctly identified as normal (92240).

False Positive shows the count of anomalies incorrectly identified as normal (14562).

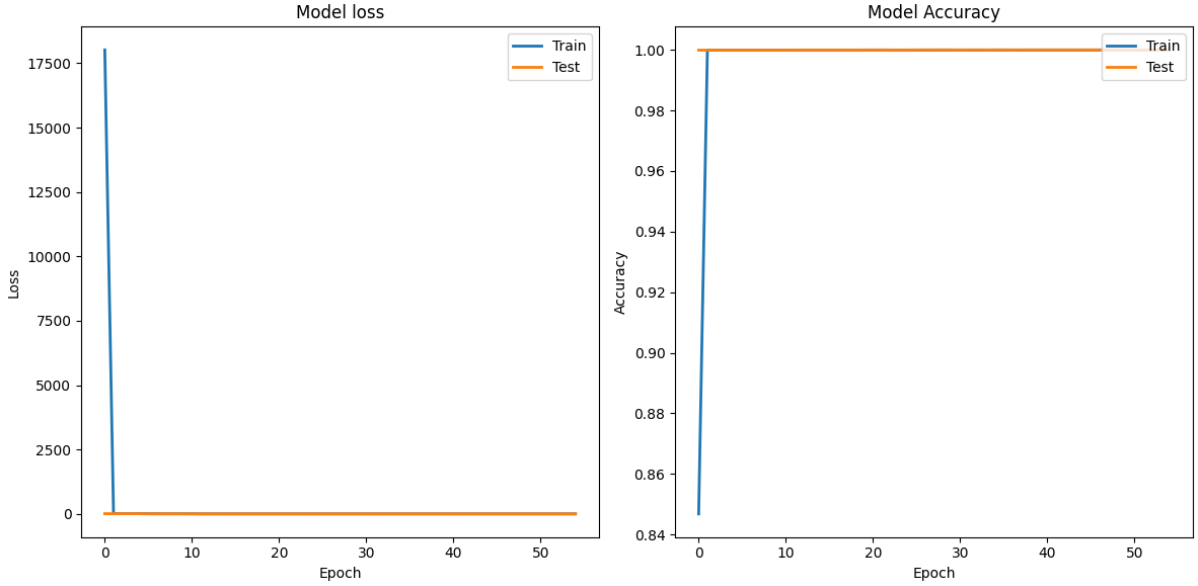
False Negative shows the count of normal instances incorrectly identified as anomalies (75285).

True Positive shows the count of anomalies correctly identified as such (31874).

The confusion matrix suggests that while the model is reasonably good at identifying normal cases (high true negative rate), it is less successful at identifying anomalies (lower true positive rate). This correlates with the lower recall for the 'True' class in the classification report. Additionally, the high number of false negatives (75285) reflects the low recall for anomalies, meaning that many actual anomalies are being overlooked by the model.

Result of AutoEncoder on Kaggle Dataset:

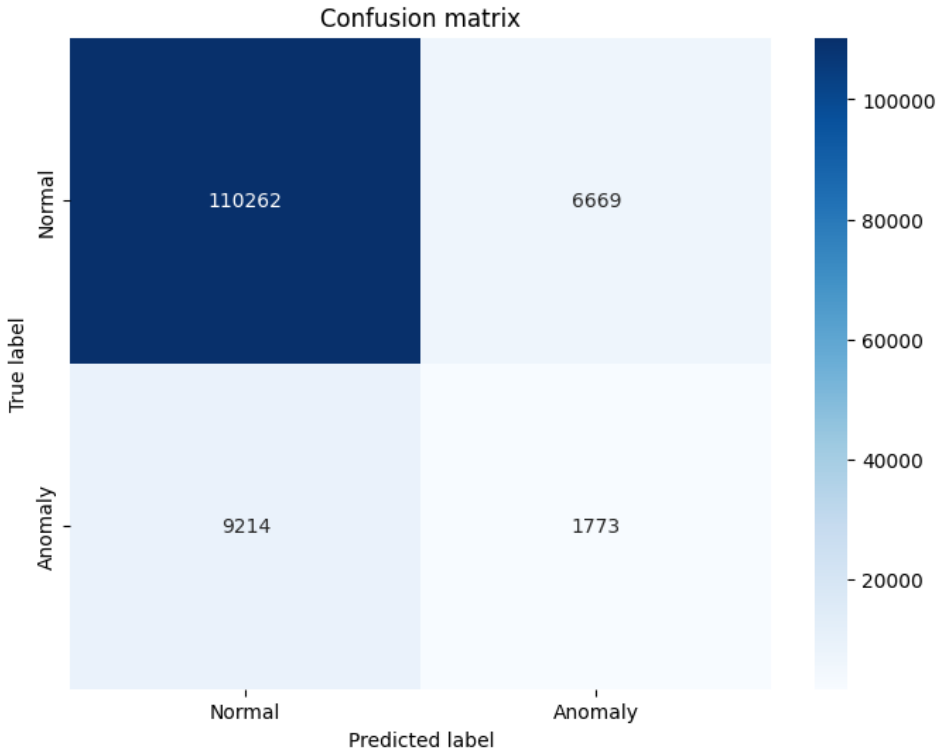
Records in Diseased Train Data = 16386
Records in Normal Train Data = 175491
Records in Diseased Test Data = 10987
Records in Normal Test Data= 116931



Model Loss: The loss on both training and test sets is decreasing, which suggests that the model is learning and improving its prediction over time.
Both training and test loss are converging to a lower point, which indicates that the model is not overfitting. Overfitting would typically be indicated by the training loss continuing to decrease while the test loss starts to increase.

Model Accuracy: The accuracy on both training and test sets starts very high and remains constant throughout the epochs, indicating that the model performs well on both the data it was trained on and unseen data. The accuracy for both is nearly perfect, which might be a sign that the task is not very complex, the model is extremely well-suited for the task, or potentially that there might be some overfitting despite what the loss indicates. However, overfitting is usually accompanied by high accuracy on the training set and lower accuracy on the test set, which doesn't seem to be the case here.

General Observations: Both charts show that the test line closely follows the training line. This is generally a good sign that the model is generalizing well. The model seems to have reached a plateau quickly, especially in terms of accuracy, which might indicate that further training would not yield significant improvements.



True Positives (TP): 98788 normal cases were correctly identified.

False Positives (FP): 18143 normal cases were incorrectly labeled as anomalies.

True Negatives (TN): 2213 anomalies were correctly identified.

False Negatives (FN): 8774 anomalies were incorrectly labeled as normal.

These results indicate that while the model is quite good at identifying normal cases (high TP), it struggles with correctly identifying anomalies (high FN, low TN). It also misclassifies a significant number of normal instances as anomalies (high FP).

Final Result and Conclusion:

	Decision Tree	SVM	Logistic Regression	MLP	K-Fold	AutoEncoder
Kaggle	86%	91%	91%	91%	91%	88%
Imputed	90%	94%	94%	94%	94%	82%
Oversampled	97%					59%

Datasets:

The models were evaluated on three datasets: the dataset from Kaggle, an original version with imputed missing values (Imputed), and one where the class distribution has been balanced through oversampling (Oversampled).

Model Performance on Kaggle Dataset:

The Support Vector Machine (SVM), Logistic Regression, and Multilayer Perceptron (MLP) models all achieved a 91% accuracy rate on the original Kaggle dataset.
The Decision Tree model performed slightly less well with an accuracy of 86%.
The Autoencoder had an accuracy of 88%, which is reasonable but lower compared to other models.

Model Performance on Imputed Dataset:

Upon the imputed dataset, all models except the Autoencoder saw an improvement in accuracy, with SVM, Logistic Regression, MLP, and the K-Fold approach all reaching 94% accuracy. The Decision Tree's performance increased to 90%, while the Autoencoder's accuracy decreased to 82%.

Model Performance on Oversampled Dataset:

With the oversampled dataset, the Decision Tree's performance saw a significant increase to 97% accuracy, suggesting that it benefited the most from the class balancing.
The SVM, Logistic Regression, MLP, and K-Fold methods maintained a high accuracy of 94%.
The Autoencoder's performance dropped notably to 59% accuracy, indicating that this model may not handle the oversampled dataset as effectively as the others.

General Observations:

The consistent accuracy across SVM, Logistic Regression, and MLP on the Kaggle and Imputed datasets suggests that these models are robust to the missing data treatment applied.
The significant improvement in the Decision Tree's performance with the Oversampled dataset indicates that class imbalance may have been adversely affecting its performance on the other datasets.

The Autoencoder's lower performance, particularly on the Oversampled dataset, might indicate that it is not as suitable for classification tasks or that it requires different preprocessing or a more complex architecture.

The performance of the models on the Oversampled dataset indicates that balancing class distribution can have a significant impact, particularly for models that are sensitive to class imbalance like Decision Trees.

In conclusion, while most models performed well across the datasets with minor variations, the Autoencoder's performance was notably poorer, especially on the oversampled data. These results highlight the importance of appropriate data preprocessing and the potential impact of class distribution on model performance.