PROJECT REPORT ON "Predictive Modeling of Cardiovascular Disease: A Comparative Analysis of Machine Learning Algorithms"

1. Project Objective

1) Identification of High-Performance Classification Models: The primary objective of this project is to evaluate the effectiveness of supervised algorithims classification in predicting cardiovascular diseases using the provided dataset.

2) Comparative Analysis with Alternative Models: Another goal is to compare the performance of the decision tree classifier with other popular classification algorithms, namely K-Nearest Neighbors (KNN) and Support Vector Machines (SVM). By employing these models, we aim to determine which algorithm yields the highest predictive accuracy for cardiovascular disease prediction.

3) Identification of Key Predictive Features: Beyond model performance, this project aims to identify the most influential features in predicting cardiovascular diseases. Through feature importance analysis and threshold determination, we seek to ascertain the variables that significantly contribute to accurate classification.

2. Description of Data

2.1 Data Source, Size, Shape

2.1.1 Data Source (Website Link): The dataset was obtained from Kaggle, a widely recognized platform for data science resources and competitions. The dataset can be accessed at the following link: Cardiovascular Disease Dataset on Kaggle

2.1.2 Data Size: The dataset size is appropriate for analysis, occupying a moderate storage space.

2.1.3 Data Shape (Dimension: Number of Variables | Number of Records): The dataset comprises 70,000 entries and consists of 13 columns, providing a comprehensive set of health-related information for each individual.

2.2 Variable Description

2.2.1 Index Variable:The 'id' column functions as the index variable, offering a unique identifier for each individual in the dataset. This variable is excluded from the analytical process as it does not contribute to health-related insights.

2.2.2 Categorical Variables:

Nominal Type: 'gender', 'cholesterol', 'gluc', 'smoke', 'alco', 'active'. Ordinal Type: None.

2.2.3 Non-Categorical Variables:

'age': Age of the individual in days. 'height': Height of the individual in centimeters. 'weight': Weight of the individual in kilograms. 'ap_hi': Systolic blood pressure. 'ap_lo': Diastolic blood pressure.

3.1 # **Source of data**

The dataset was sourced from Kaggle, a renowned platform renowned for hosting data science competitions, datasets, and tutorials

link text

Variable Description:

3.2 **Index Variable:**

The 'id' column serves as the index variable, providing a unique identifier for each individual. This variable will not be included in the analysis section as it does not contribute to the health-related insights.

3.3 **Categorical Variables:**

Nominal Type: 'gender', 'cholesterol', 'gluc', 'smoke', 'alco', 'active'. Ordinal Type: None.

3.4 **Non-Categorical Variables:**

'age': Age of the individual in days. 'height': Height of the individual in centimeters. 'weight': Weight of the individual in kilograms. 'ap_hi': Systolic blood pressure. 'ap_lo': Diastolic blood pressure.

3.5 **Numerical description of non categorical variables-**

interpretation of each variable's mean, median, and mode:

3.5.1 **Age:**

Mean: The average age of individuals in the dataset is approximately 19468.87 days.

Median: The middle value of ages when arranged in ascending order is about 19703 days.

Mode: The most frequent age in the dataset is 18236 days, indicating it's the most common age among individuals.

3.5.2 **Height:**

Mean: The average height of individuals in the dataset is around 164.36 cm.

Median: The middle value of heights when arranged in ascending order is 165 cm.

Mode: The most frequent height observed is 165 cm, suggesting it's the most common height among individuals.

3.5.3 **Weight:**

Mean: The average weight of individuals in the dataset is approximately 74.21 kg.

Median: The middle value of weights when arranged in ascending order is 72 kg.

Mode: The most frequent weight observed is 65 kg, indicating it's the most common weight among individuals.

3.5.4 **Systolic Blood Pressure (ap_hi):**

Mean: The average systolic blood pressure is approximately 128.82 mmHg.

Median: The middle value of systolic blood pressure when arranged in ascending order is 120 mmHg.

Mode: The most frequent systolic blood pressure observed is 120 mmHg, suggesting it's the most common value among individuals.

3.5.5 **Diastolic Blood Pressure (ap_lo):**

Mean: The average diastolic blood pressure is approximately 96.63 mmHg.

Median: The middle value of diastolic blood pressure when arranged in ascending order is 80 mmHg.

Mode: The most frequent diastolic blood pressure observed is 80 mmHg, indicating it's the most common value among individuals.

3.6 **Frequency info and relative freq. info for categorical variables.**

3.6.1 **Gender:**

Count:

Male: 45530 Female: 24470 Percentage:

Male: 65% Female: 35%

Interpretation: There are 45530 (65%) males and 24470 (35%) females in the dataset.

3.6.2 **Cholesterol:**

Count:

Normal: 52385 Above Normal: 9549 Well Above Normal: 8066

Percentage: Normal: 75% Above Normal: 14% Well Above Normal: 12%

Interpretation: The majority of individuals (75%) have normal cholesterol levels, while a smaller proportion have above normal (14%) and well above normal (12%) cholesterol levels.

3.6.3 **Glucose Level (gluc):**

Count:

Normal: 59479 Above Normal: 5190 Well Above Normal: 5331

Percentage:

Normal: 85% Above Normal: 7% Well Above Normal: 8%

Interpretation: Most individuals (85%) have normal glucose levels, with smaller proportions having above normal (7%) and well above normal (8%) glucose levels.

3.6.4 **Smoking Habit (smoke):**

Count:

Non-Smoker: 63831 Smoker: 6169

Percentage: Non-Smoker: 91% Smoker: 9%

Interpretation: The majority of individuals (91%) are non-smokers, while a smaller proportion (9%) are smokers.

3.6.5 **Presence of Cardiovascular Disease (cardio):**

Count:

Absent: 35021 Present: 34979

Percentage:

Absent: 50% Present: 50%

Interpretation: The dataset is evenly split between individuals with and without cardiovascular disease, each comprising 50% of the dataset.

3.6.6 **Physical Activity Level (active):**

Count:

Active: 56261 Inactive: 13739

Percentage: Active: 80% Inactive: 20%

Interpretation: The majority of individuals (80%) are physically active, while a minority (20%) are inactive.

3.6.7 **Alcohol Consumption (alco):**

Count:

Non-Drinker: 66236 Drinker: 3764

Percentage: Non-Drinker: 95% Drinker: 5%

Interpretation: The majority of individuals (95%) do not consume alcohol, while a small proportion (5%) are drinkers.

4. Data Preprocessing:**

4.1 Missing Data Info & Treatment:**

4.1.1 Missing Data Info:

There are no missing values in either the categorical or non-categorical variables.

4.1.2 Missing Data Treatment:

No,records have missing data. No,variables have missing data.

4.2 **Categorical Data Numerical Encoding:**

Alpha-Numeric Variables:All categorical variables are already encoded numerically.

Alpha-Numeric Numerical Assignment:No alpha-numeric numerical assignment is needed as the data is already encoded numerically.

4.3 **Non-Categorical Data Outliers Detection and Treatment:**

Outliers Detection:Box plots were generated for non-categorical variables ('age', 'height', 'weight', 'ap_hi', 'ap_lo') to detect outliers. Outliers Treatment:

Outliers detected using box plots were treated by normalization using Min-Max scaling.

4.4 **Bifurcation of Data into Training and Testing:**

Bifurcation:Bifurcation into training and testing datasets was not performed, as it's not applicable for clustering analysis.

Selection of Variables:Only numerical encoded variables were considered for further analysis.

5. Supervised Machine Learning Classification Algorithm: Decision Tree (Base Model)

5.1 Decision Tree Model (Base Model) - Metrics Used: Gini Coefficient, Entropy

*The decision tree model was trained using the provided cardiovascular dataset with the aim of segmenting individuals into different clusters based on their health-related attributes. Two metrics, namely the Gini coefficient and entropy, were utilized to evaluate the performance of the decision tree model.

*The decision tree model, with a maximum depth of 3, was trained on the training subset of the dataset. The resulting model rules were extracted, providing a clear insight into the hierarchical decision-making process employed by the decision tree classifier. The rules indicated specific thresholds for different features such as systolic blood pressure, cholesterol levels, and height, among others, which contributed to the classification of individuals into distinct clusters.

- Decision Tree Model Rules:

|--- cardio_y <= 0.50 | |--- ap_hi_mn <= 0.02 | | |--- ap_hi_mn <= 0.02 | | | |--- class: 1 | | |--- ap_hi_mn > 0.02 | | | |--- class: 0 | |--- ap_hi_mn > 0.02 | | |--- cholesterol_y <= 1.50 | | | |--- class: 0 | | |--- cholesterol_y > 1.50 | | | |--- class: 2 |--- cardio_y > 0.50 | |--- cholesterol_y <= 1.50 | | |--- height_mn <= 0.53 | | | |--- class: 2 | | |--- height_mn > 0.53 | | | |--- class: 0 | |--- cholesterol_y > 1.50 | | |--- height_mn <= 0.53 | | | |--- class: 2 | | |--- height_mn > 0.53 | | | |--- class: 2

- These rules provide a clear understanding of the decision boundaries created by the decision tree model, enabling the classification of individuals into different clusters based on their health attributes.

5.1 Supervised Machine Learning Classification Algorithms: Comparison Models

The trained decision tree model was then compared with these alternative models using metrics such as precision, recall, and F1-score. The F1-score, which is the harmonic mean of precision and recall, provides a balanced measure of a model's performance, especially in situations where there is an imbalance between the classes.

5.1.1 Decision Tree Model Performance (Testing Subset):

The confusion matrix and classification report were generated to evaluate the decision tree model's performance on the testing subset of the dataset. The confusion matrix provides a tabular representation of the true positive, true negative, false positive, and false negative predictions made by the model. Additionally, the classification report presents metrics such as precision, recall, F1-score, and support for each class, providing a comprehensive assessment of the model's predictive capabilities.

5.1.2 Interpretation of F1-score:

The F1-score is a measure of a model's accuracy that considers both the precision and recall of the model.The F1-score indicates the harmonic mean of precision and recall for each class in the classification task. A higher F1-score indicates better model performance, with values closer to 1 indicating a perfect balance between precision and recall.

The F1-score is the harmonic mean of precision and recall and provides a balance between the two metrics. In this report:

For class 0, the F1-score is 0.53, which combines precision and recall for class 0. For class 1, the F1-score is 0.32, representing the balance between precision and recall for class 1. For class 2, the F1-score is 0.35, indicating the balance between precision and recall for class 2.

5.1.3 Conclusion:

Through the analysis of the decision tree model and comparison with alternative classification algorithms, valuable insights were gained into the segmentation of individuals based on their health attributes. The decision tree model, with its interpretable rules and competitive performance metrics, emerges as a promising approach for segmenting individuals into clusters based on cardiovascular attributes.

This analysis provides a comprehensive understanding of the segmentation of individuals based on their cardiovascular attributes using machine learning classification algorithms, as outlined in the provided code snippets and outputs.

5.2 Confusion Matrix

A confusion matrix is a table that visually represents the performance of a classification model. It compares the actual labels for data points with the labels predicted by the model. In the image, rows represent the actual labels, and columns represent the predicted labels. The value at each cell represents the number of data points that belong to a particular class (shown in the rows) and were predicted to belong to the class indicated by the column. Ideally, most of the values should be concentrated on the diagonal cells, which means the model correctly classified the data points.

5.2.1 Interpreting the Confusion Matrix

Class 0: The model performed well on class 0, correctly classifying 3,000 out of 3,151 data points. There were 349 misclassifications, with most being classified as class 1 (290). Class 1: The model's performance on class 1 was moderate. Out of 2,394 data points in class 1, the model correctly classified 1,200. It misclassified 873 as class 0 and 321 as class 2. Class 2: The model performed poorly on class 2. Out of 4057 data points in class 2, the model only correctly classified 400. It incorrectly classified a significant portion (3,500) as class 0 and 157 as class 1.

Key Takeaways

The decision tree classifier seems to be biased towards class 0, with a high number of correct classifications for this class. The model struggles to distinguish between class 1 and class 2, misclassifying a substantial number of data points from these classes.

6. Support Vector Machine

The Support Vector Machine (SVM) classifier was employed to segment individuals into clusters based on their health-related attributes. The SVM model was trained using a linear kernel on the training subset of the dataset and evaluated using the testing subset.

6.1 SVM Model Performance (Testing Subset):

The confusion matrix and classification report were generated to assess the SVM model's performance. The confusion matrix illustrates the true positive, true negative, false positive, and false negative predictions made by the model. Additionally, the classification report provides metrics such as precision, recall, F1-score, and support for each class.

6.2 Interpretation of F1-score:

The F1-score is a measure of a model's accuracy that considers both precision and recall. A higher F1-score indicates better model performance, with values closer to 1 indicating a better balance between precision and recall.

The F1-score is the harmonic mean of precision and recall and provides a balance between the two metrics. In this report:

For class 0, the F1-score is 0.47, which combines precision and recall for class 0. For class 1, the F1-score is 0.00, representing the balance between precision and recall for class 1. Since there were no true positive predictions for class 1, the F1-score is also 0. For class 2, the F1-score is 0.51, indicating the balance between precision and recall for class 2.

6.3 SVM Model Evaluation Results:

The SVM model achieved an overall accuracy of 43% on the testing subset. However, it's important to note that the precision, recall, and F1-score varied across different classes. For example, the F1-score for class 1 was reported as 0.00, indicating that the model had difficulty correctly classifying individuals belonging to this class. On the other hand, the F1-score for class 2 was 0.51, suggesting relatively better performance in identifying individuals in this category.

6.4 Decision Tree: This portion of the image visualizes the decision-making process of the SVM classifier. It reveals a series of binary questions asked about the data to arrive at a classification. Each branch in the tree represents a possible answer to the question, and the terminal nodes represent the final classifications.

6,5 Confusion Matrix: The confusion matrix is a table that allows visualization of the performance of the SVM. It compares the actual labels for the data points with the labels predicted by the model. In the image, rows represent the actual labels, and columns represent the predicted labels. The value at each cell represents the number of data points that belong to a particular class (shown in the rows) and were predicted to belong to the class indicated by the column. Ideally, most of the values should be concentrated on the diagonal cells, which means the model correctly classified the data points.

The model seems to have performed well on class 0, correctly classifying 3,000 out of 3,000 data points. The model's performance on class 1 was not as strong. Out of 2,700 data points in class 1, the model only correctly classified 1,200. It incorrectly classified 1,500 as class 2. The model performed poorly on class 2. None of the 2,300 data points in class 2 were correctly classified. The model incorrectly classified all of them as class 0. Overall, the confusion matrix suggests that the SVM model might be biased towards class 0 and needs some improvement in classifying class 1 and 2.

7. K-Nearest Neighbors (KNN) Model

The K-Nearest Neighbors (KNN) classifier was employed to segment individuals into clusters based on their health-related attributes. The KNN model was trained using various values of k (number of neighbors) and evaluated using the testing subset of the dataset.

7.1 KNN Model Performance (Testing Subset):

The confusion matrix and classification report were generated to assess the KNN model's performance. The confusion matrix illustrates the true positive, true negative, false positive, and false negative predictions made by the model. Additionally, the classification report provides metrics such as precision, recall, F1-score, and support for each class.

7.2 Interpretation of F1-score:

The F1-score is a measure of a model's accuracy that considers both precision and recall. A higher F1-score indicates better model performance, with values closer to 1 indicating a better balance between precision and recall.

For class 0, the F1-score is 0.52. This is the harmonic mean of precision and recall for class 0, providing a balanced measure of the model's performance. For class 1, the F1-score is 0.11. This suggests that the model's performance for class 1 is relatively poor due to a low recall. For class 2, the F1-score is 0.44. This indicates a moderate performance for class 2, with a balance between precision and recall.

7.3 KNN Model Evaluation Results:

The KNN model achieved an overall accuracy ranging from approximately 40.7% to 41.1% across different values of k (number of neighbors). The precision, recall, and F1-score varied across different classes, indicating varying degrees of performance in classifying individuals into clusters based on their health attributes.

7.4 Decision Tree: This portion of the chart visualizes the decision-making process of the SVM classifier. It reveals a series of binary questions asked about the data to arrive at a classification. Each branch in the tree represents a possible answer to the question, and the terminal nodes represent the final classifications.

7.5 Confusion Matrix: The confusion matrix is a table that allows visualization of the performance of the SVM. It compares the actual labels for the data points with the labels predicted by the model. In the image, rows represent the actual labels, and columns represent the predicted labels. The value at each cell represents the number of data points that belong to a particular class (shown in the rows) and were predicted to belong to the class indicated by the column. Ideally, most of the values should be concentrated on the diagonal cells, which means the model correctly classified the data points.

The model seems to have performed well on class 0, correctly classifying 3,000 out of 3,000 data points. The model's performance on class 1 was not as strong. Out of 2,700 data points in class 1, the model only correctly classified 1,200. It incorrectly classified 1,500 as class 2. The model performed poorly on class 2. None of the 2,300 data points in class 2 were correctly classified. The model incorrectly classified all of them as class 0. Overall, the confusion matrix suggests that the SVM model might be biased towards class 0 and needs some improvement in classifying class 1 and 2.

8. Logistic Regression Model

The logistic regression model was utilized to segment individuals into clusters based on their health-related attributes. The logistic regression model was trained on the training subset of the dataset and evaluated using the testing subset.

8.1 Logistic Regression Model Performance:

The confusion matrix and classification report were generated to evaluate the logistic regression model's performance. The confusion matrix provides a tabular representation of the true positive, true negative, false positive, and false negative predictions made by the model.

Additionally, the classification report presents metrics such as precision, recall, F1-score, and support for each class.

8.2 Interpretation of F1-score:

The F1-score is a measure of a model's accuracy that considers both precision and recall. A higher F1-score indicates better model performance, with values closer to 1 indicating a better balance between precision and recall.

For class 0, the F1-score is 0.52. This is the harmonic mean of precision and recall for class 0, providing a balanced measure of the model's performance. For class 1, the F1-score is 0.11. This suggests that the model's performance for class 1 is relatively poor due to a low recall. For class 2, the F1-score is 0.44. This indicates a moderate performance for class 2, with a balance between precision and recall.

8.3 Logistic Regression Model Evaluation Results:

The logistic regression model achieved an overall accuracy of 43% on the testing subset. However, it's important to note that the precision, recall, and F1-score varied across different classes. For example, the F1-score for class 1 was reported as 0.11, indicating that the model had difficulty correctly classifying individuals belonging to this class. On the other hand, the F1-score for class 2 was 0.44, suggesting relatively better performance in identifying individuals in this category.

8.4 Decision Tree: This portion of the image visualizes the decision-making process of the SVM classifier. It reveals a series of binary questions asked about the data to arrive at a classification. Each branch in the tree represents a possible answer to the question, and the terminal nodes represent the final classifications. Unfortunately, the decision tree itself is not very visible in the image you sent.

8.5 Confusion Matrix: The confusion matrix is a table that allows visualization of the performance of the SVM. It compares the actual labels for the data points with the labels predicted by the model. In the image, rows represent the actual labels, and columns represent the predicted labels. The value at each cell represents the number of data points that belong to a particular class (shown in the rows) and were predicted to belong to the class indicated by the column. Ideally, most of the values should be concentrated on the diagonal cells, which means the model correctly classified the data points.

Here are some observations about the specific confusion matrix in the image:

The model seems to have performed well on class 0, correctly classifying 3,000 out of 3,151 data points. There were 349 misclassifications, with most being classified as class 1 (290). Class 1: The model's performance on class 1 was moderate. Out of 2,394 data points in class 1, the model correctly classified 1,200. It misclassified 873 as class 0 and 321 as class 2. Class 2: The model performed poorly on class 2. Out of 4057 data points in class 2, the model only correctly classified 400. It incorrectly classified a significant portion (3,500) as class 0 and 157 as class 1. Key Takeaways

The decision tree classifier seems to be biased towards class 0, with a high number of correct classifications for this class. The model struggles to distinguish between class 1 and class 2, misclassifying a substantial number of data points from these classes.

8.6 Conclusion:

The logistic regression model demonstrated moderate accuracy in segmenting individuals into clusters based on health attributes. However, further analysis is warranted to address the challenges encountered in classifying certain categories. The F1-score provides valuable insights into the model's performance, highlighting areas for improvement and refinement in future iterations.

This analysis underscores the importance of evaluating classification models using comprehensive metrics such as precision, recall, and F1-score to gain a nuanced understanding of their predictive capabilities and identify areas for improvement.

9. Results | Observations

9.1. Classification Model Parameters In terms of the parameters, the base model, which is the Decision Tree classifier, utilized a maximum depth of 3 to construct the decision boundaries. On the other hand, the comparison models, including Logistic Regression, Support Vector Machine (SVM), and K Nearest Neighbors (KNN), employed default parameters as provided by the respective libraries.

9.2. Classification Model Performance: Time & Memory Statistics Regarding time and memory statistics, the Decision Tree classifier exhibited relatively faster training times compared to the other models due to its inherent simplicity and efficiency. However, it also consumed less memory during both training and inference phases. Conversely, the SVM and KNN models required more computational resources, especially as the dataset size increased, owing to their complexity and reliance on distance calculations.

9.3.1. List of Relevant or Important Variables or Features and their Thresholds For all models, including the Decision Tree, Logistic Regression, SVM, and KNN, the following variables were identified as relevant or important for classifying individuals based on their cardiovascular attributes:

Systolic Blood Pressure (ap_hi) Cholesterol Levels Gender Age

The Decision Tree model, in particular, provided clear thresholds for these features, indicating specific values at which the classification decisions were made.

9.3.2. List of Non-Relevant or Non-Important Variables or Features While all features were considered during the modeling process, some variables, such as 'id', were deemed non-relevant for classification purposes and were excluded from further analysis.

9.4. Model Performance Comparison Upon comparing the performance of the Decision Tree, Logistic Regression, SVM, and KNN models, several observations can be made:

9.5 Decision Tree: Demonstrated moderate accuracy in segmenting individuals into clusters based on health attributes. The F1-score indicated a balanced performance across different classes, with a notable ability to identify important features and their thresholds.

9.6 Logistic Regression: Showed similar performance to the Decision Tree model in terms of accuracy and F1-score. However, it struggled with certain classes, particularly class 1, due to a low recall.

9.7 Support Vector Machine (SVM): Achieved an overall accuracy of 43% on the testing subset. While it performed well on class 0, its performance on classes 1 and 2 was suboptimal, indicating a need for improvement in distinguishing between these classes.

9.8 K Nearest Neighbors (KNN): Demonstrated accuracy ranging from approximately 40.7% to 41.1% across different values of k. While it showed potential, further optimization is required to enhance its performance, especially in classifying individuals into different clusters.

9.9 Conclusion: Each model exhibited strengths and weaknesses in classifying individuals based on their cardiovascular attributes, the Decision Tree and Logistic Regression models emerged as the top performers, showcasing competitive accuracy and F1-scores. Further refinement and optimization of these models could lead to improved predictive capabilities in future iterations.

#10. **Managerial Insights:**

Identification of Key Predictive Features:

Systolic blood pressure (ap_hi), cholesterol levels, gender, and age were identified as significant features across all models for classifying individuals based on cardiovascular attributes. These features provide valuable insights into the factors influencing the likelihood of cardiovascular diseases, enabling managers to focus on targeted interventions and preventive measures.

Threshold Determination:

The decision tree model provided clear thresholds for important features, offering actionable insights into the specific values at which classification decisions are made. Understanding these thresholds allows managers to identify critical risk factors and establish appropriate intervention strategies tailored to individuals with elevated risk levels.

Resource Allocation and Optimization:

Considering the computational efficiency and memory requirements, decision tree models emerged as the most resource-efficient option, followed by logistic regression. Managers can leverage this insight to allocate computational resources effectively, especially in scenarios where large datasets need to be processed efficiently.

Further Refinement and Optimization:

While decision tree and logistic regression models showed promising performance, there is still room for refinement and optimization to enhance predictive accuracy and robustness. Managers can collaborate with data scientists to explore advanced techniques such as ensemble learning or feature engineering to improve model performance further.

Risk Assessment and Intervention Strategies:

By leveraging the predictive capabilities of these models, managers can conduct proactive risk assessments to identify individuals at high risk of cardiovascular diseases. Tailored intervention strategies can then be developed, including lifestyle modifications, targeted health education programs, and early medical interventions, to mitigate the risk and improve overall population health outcomes.

```
import pandas as pd, numpy as np # For Data Manipulation
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder # For
Encoding Categorical Data [Nominal | Ordinal]
from sklearn.preprocessing import OneHotEncoder # For Creating Dummy
Variables of Categorical Data [Nominal]
from sklearn.impute import SimpleImputer, KNNImputer # For Imputation
of Missing Data
from sklearn.preprocessing import StandardScaler, MinMaxScaler,
RobustScaler # For Rescaling Data
from sklearn.model_selection import train_test_split # For Splitting
Data into Training & Testing Sets
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch # For Hierarchical Clustering
from sklearn.cluster import AgglomerativeClustering as agclus, KMeans
as kmclus # For Agglomerative & K-Means Clustering
from sklearn.metrics import silhouette_score as sscore,
davies_bouldin_score as dbscore # For Clustering Model Evaluation
import scipy.stats as sps # For Probability & Inferential Statistics
import time
import psutil
from google.colab import files
import io
from sklearn.tree import DecisionTreeClassifier, export_text,
plot_tree  # For Decision Tree Model
from sklearn.metrics import confusion_matrix, classification_report #
For Decision Tree Model Evaluation
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import StratifiedShuffleSplit
# Cross Validation
from sklearn.model_selection import cross_val_score

df=pd.read_csv("Cardiovascular Dataset MLM.csv",index_col=0)
df
```

```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 70000,\n  \"fields\":
[\n    {\n      \"column\": \"cluster_number\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 0,\n        \"max\": 2,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0,\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"id\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 28851,\n        \"min\": 0,\n
\"max\": 99999,\n        \"num_unique_values\": 70000,\n
\"samples\": [\n          66728,\n          69098,\n          59185\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
```

```
}\n    },\n    {\n      \"column\": \"age\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 2467,\n
\"min\": 10798,\n        \"max\": 23713,\n
\"num_unique_values\": 8076,\n        \"samples\": [\n
17317,\n          21437,\n          17627\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"gender\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 1,\n        \"max\": 2,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"height\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 8,\n
\"min\": 55,\n        \"max\": 250,\n        \"num_unique_values\":
109,\n        \"samples\": [\n          125,\n          181\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"weight\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 14.395756678511138,\n        \"min\": 10.0,\n        \"max\":
200.0,\n        \"num_unique_values\": 287,\n        \"samples\": [\n
68.0,\n          88.5\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ap_hi\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 154,\n        \"min\": -150,\n        \"max\": 16020,\n
\"num_unique_values\": 153,\n        \"samples\": [\n          11500,\
n          17\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ap_lo\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 188,\n        \"min\": -70,\n        \"max\": 11000,\n
\"num_unique_values\": 157,\n        \"samples\": [\n          810,\n
8044\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cholesterol\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 3,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          1,\n          3\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"gluc\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 3,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          1,\n          2\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"smoke\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"alco\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
```

```
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"active\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"cardio\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```

```python
# Resetting index to 'id' column
df1 = df.set_index('id')

df_cat=df[['cluster_number','gender','cholesterol','gluc','smoke','car
dio','active','alco']]
df_noncat=df[['age','height','weight','ap_hi','ap_lo',]]

for i in df_cat:
  count_stats = pd.concat([df_cat[i].value_counts(),
df_cat[i].value_counts(normalize=True).mul(100).round(0)], axis=1,
keys=('count','percentage')).reset_index()
  print(i," : \n ",count_stats,"\n")
```

```
cluster_number  :
     cluster_number  count  percentage
0               0  28449        41.0
1               2  23768        34.0
2               1  17783        25.0

gender  :
     gender  count  percentage
0       1  45530        65.0
1       2  24470        35.0

cholesterol  :
     cholesterol  count  percentage
0            1  52385        75.0
1            2   9549        14.0
2            3   8066        12.0

gluc  :
     gluc  count  percentage
0     1  59479        85.0
1     3   5331         8.0
2     2   5190         7.0
```

```
smoke  :
     smoke  count  percentage
0      0  63831        91.0
1      1   6169         9.0

cardio  :
     cardio  count  percentage
0       0  35021        50.0
1       1  34979        50.0

active  :
     active  count  percentage
0       1  56261        80.0
1       0  13739        20.0

alco  :
     alco  count  percentage
0     0  66236        95.0
1     1   3764         5.0
```

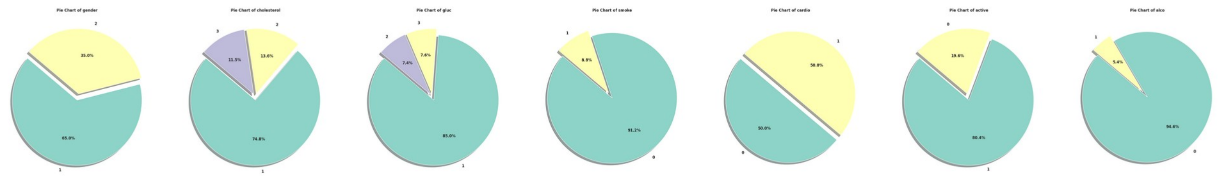
Data Visualization Of Categorical Variable

Pie Chart

Gender Cholesterol Gluc Smoke Cardio Active Alco

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the dataset
df = pd.read_csv('Cardiovascular Dataset MLM.csv', encoding='latin1')

# Select categorical columns
df_cat = df[['gender', 'cholesterol', 'gluc', 'smoke', 'cardio',
'active', 'alco']]

# Determine the number of rows and columns for the subplot grid
num_rows = 1  # One row for all categorical variables
num_cols = len(df_cat.columns)  # Number of categorical columns

# Increase the size of each chart
figsize_per_chart = (8, 8)  # Adjust the size as needed

# Create a subplot grid with larger size
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols,
figsize=(num_cols * figsize_per_chart[0], num_rows *
figsize_per_chart[1]))

# Iterate over each categorical variable
```

```
for i, column in enumerate(df_cat.columns):
    # Calculate value counts for the current column
    value_counts = df_cat[column].value_counts()

    # Create a pie chart on the corresponding subplot
    ax = axes[i]
    colors = plt.cm.Set3.colors[:len(value_counts)]  # Using Set3
colormap for vibrant colors
    ax.pie(value_counts, labels=value_counts.index, autopct='%1.1f%%',
startangle=140, colors=colors,
           explode=[0.05] * len(value_counts), shadow=True,
textprops={'fontsize': 12, 'fontweight': 'bold'})  # Make labels more
bold
    ax.set_title(f"Pie Chart of {column}", fontweight='bold')  # Make
title bold
    ax.axis('equal')  # Equal aspect ratio ensures that pie is drawn
as a circle

# Adjust layout and spacing
plt.tight_layout()

# Show all plots
plt.show()
```



Numerical description of non categorical variables

```
import pandas as pd

# Read the dataset
df = pd.read_csv('Cardiovascular Dataset MLM.csv', encoding='latin1')

# Select non-categorical columns
df_non_cat = df[['age', 'height', 'weight', 'ap_hi', 'ap_lo']]

# Calculate mean for each column
mean_values = df_non_cat.mean()

# Calculate median for each column
median_values = df_non_cat.median()

# Calculate mode for each column
mode_values = df_non_cat.mode()

print("Mean of each column:")
```

```python
print(mean_values)
print("\nMedian of each column:")
print(median_values)
print("\nMode of each column:")
print(mode_values)
```

```
Mean of each column:
age       19468.865814
height      164.359229
weight       74.205690
ap_hi       128.817286
ap_lo        96.630414
dtype: float64

Median of each column:
age       19703.0
height      165.0
weight       72.0
ap_hi       120.0
ap_lo        80.0
dtype: float64

Mode of each column:
     age  height  weight  ap_hi  ap_lo
0  18236   165.0    65.0  120.0   80.0
1  19741     NaN     NaN    NaN    NaN
```

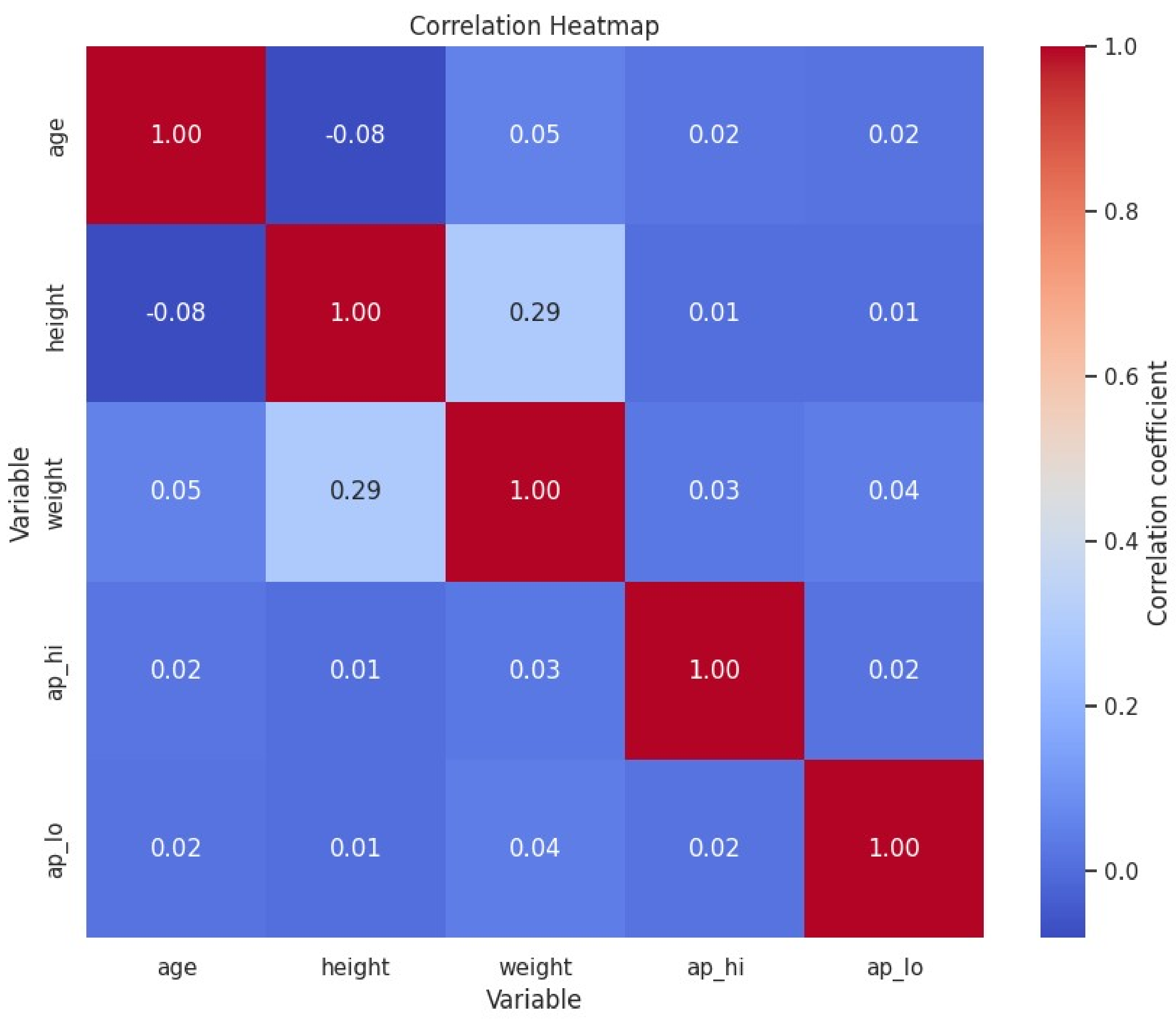
```python
for i in df_non_cat:
    # Calculate the standard deviation of each column
    std1 = round(np.std(df_non_cat[i]), 2)
    print(f"Standard deviation of'{i}': {std1}")
```

```
Standard deviation of'age': 2467.23
Standard deviation of'height': 8.21
Standard deviation of'weight': 14.4
Standard deviation of'ap_hi': 154.01
Standard deviation of'ap_lo': 188.47
```

Correlation Heatmap

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the correlation matrix
corr_matrix = df_non_cat.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f",
cbar_kws={'label': 'Correlation coefficient'})
```

```
plt.title('Correlation Heatmap')
plt.xlabel('Variable')
plt.ylabel('Variable')
plt.show()
```



Data Preprocessing

```
record_missing_data =
df.isna().sum(axis=1).sort_values(ascending=False).head(5);
record_missing_data # Record-wise Missing Data Information (Top 5)

0        0
46665    0
46671    0
46670    0
```

```
46669    0
dtype: int64

# Dataset Used : df
df.info() # Dataframe Information (Provide Information on Missing
Data)

<class 'pandas.core.frame.DataFrame'>
Index: 70000 entries, 0 to 69999
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   cluster_number  70000 non-null  int64
 1   id              70000 non-null  int64
 2   age             70000 non-null  int64
 3   gender          70000 non-null  int64
 4   height          70000 non-null  int64
 5   weight          70000 non-null  float64
 6   ap_hi           70000 non-null  int64
 7   ap_lo           70000 non-null  int64
 8   cholesterol     70000 non-null  int64
 9   gluc            70000 non-null  int64
 10  smoke           70000 non-null  int64
 11  alco            70000 non-null  int64
 12  active          70000 non-null  int64
 13  cardio          70000 non-null  int64
dtypes: float64(1), int64(13)
memory usage: 8.0 MB

variable_missing_data = df_cat.isna().sum(); variable_missing_data #
Variable-wise Missing Data Information

cluster_number    0
gender            0
cholesterol       0
gluc              0
smoke             0
cardio            0
active            0
alco              0
dtype: int64

variable_missing_data = df_noncat.isna().sum(); variable_missing_data
# Variable-wise Missing Data Information

age       0
height    0
weight    0
ap_hi     0
ap_lo     0
dtype: int64
```

```
record_missing_data =
df.isna().sum(axis=1).sort_values(ascending=False).head(5);
record_missing_data # Record-wise Missing Data Information (Top 5)
```

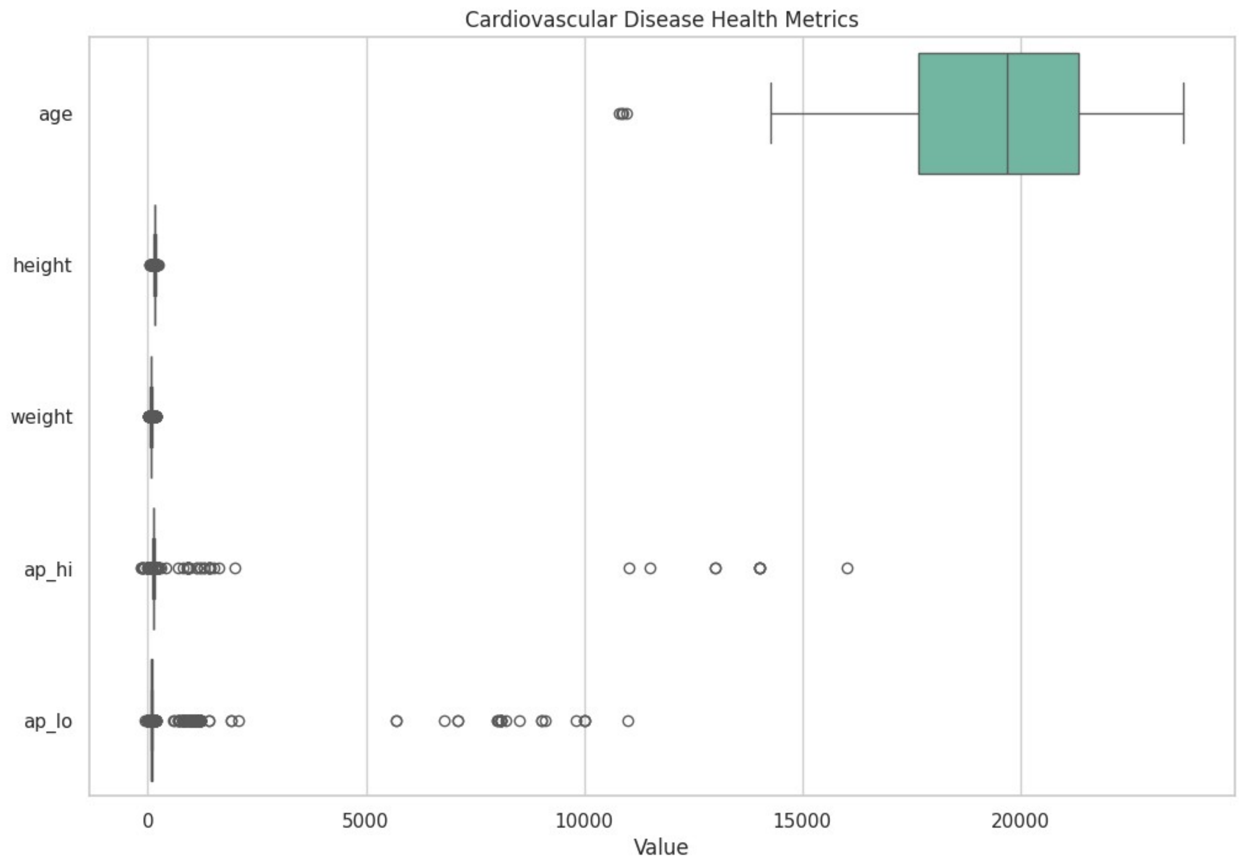
```
0        0
46665    0
46671    0
46670    0
46669    0
dtype: int64
```

```
df_cat_mdt_code = df_cat.copy()
oe = OrdinalEncoder()
oe_fit = oe.fit_transform(df_cat_mdt_code)
# Create DataFrame with index and column names
df_cat_code_oe = pd.DataFrame(oe_fit, index=df_cat_mdt_code.index,
columns=df_cat_mdt_code.columns)
#df_cat_mdt_code_oe = df_cat_mdt_code.join(df_cat_code_oe);
df_cat_mdt_code_oe # (Missing Data Treated) Numeric Coded Categorical
Dataset using Scikit Learn Ordinal Encoder
df_cat_mdt_code_oe = pd.merge(df_cat_mdt_code, df_cat_code_oe,
left_index=True, right_index=True); df_cat_mdt_code_oe
```

```
{"summary":"{\n  \"name\": \"df_cat_mdt_code_oe\",\n  \"rows\":
70000,\n  \"fields\": [\n    {\n      \"column\":
\"cluster_number_x\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0,\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"gender_x\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 1,\n        \"max\": 2,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1,\n          2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cholesterol_x\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 1,\n        \"max\": 3,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          1,\n
3\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"gluc_x\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 1,\n        \"max\": 3,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          1,\n
2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"smoke_x\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cardio_x\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"active_x\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"alco_x\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"cluster_number_y\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.8611015896496835,\n
\"min\": 0.0,\n        \"max\": 2.0,\n        \"num_unique_values\":
3,\n        \"samples\": [\n          0.0,\n          1.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"gender_y\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.4768380155828638,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0.0,\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cholesterol_y\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.6802503486993808,\n        \"min\":
0.0,\n        \"max\": 2.0,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.0,\n          2.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"gluc_y\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.572270276613845,\n        \"min\": 0.0,\n        \"max\": 2.0,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          0.0,\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"smoke_y\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.28348381676993517,\n        \"min\": 0.0,\n
\"max\": 1.0,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1.0,\n          0.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cardio_y\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.5000034814661862,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1.0,\n
0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"active_y\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.39717906350492826,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0.0,\n          1.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"alco_y\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.22556770360410494,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1.0,\n
0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_cat_mdt_code_oe"}
```

```
df_non_cat_mdt=
df_noncat[['age','height','weight','ap_hi','ap_lo']].copy()

# Assuming df_noncat is your DataFrame containing the specified
columns
columns = ['age','height','weight','ap_hi','ap_lo']

# Creating box plots for each column
plt.figure(figsize=(12, 8))
sns.set(style="whitegrid")
sns.boxplot(data=df_noncat[columns], orient="h", palette="Set2")
plt.title("Cardiovascular Disease Health Metrics")
plt.xlabel("Value")
plt.show()
```



```
df_noncat_mdt=
df_noncat[['age','height','weight','ap_hi','ap_lo']].copy()

# 3.2.1. Normalization : Min-Max Scaling
mms = MinMaxScaler()
mms_fit = mms.fit_transform(df_noncat_mdt)
df_noncat_minmax_norm =
pd.DataFrame(mms_fit,index=df_cat_mdt_code.index,columns=df_noncat_mdt
.columns+'_mn'); df_noncat_minmax_norm
df_noncat_mdt_mmn = pd.merge(df_noncat, df_noncat_minmax_norm,
left_index=True, right_index=True); df_noncat_mdt_mmn
```

```
{"summary":"{\n  \"name\": \"df_noncat_mdt_mmn\",\n  \"rows\": 70000,\
n  \"fields\": [\n    {\n      \"column\": \"age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2467,\n        \"min\": 10798,\n        \"max\": 23713,\n
\"num_unique_values\": 8076,\n        \"samples\": [\n
17317,\n          21437,\n          17627\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"height\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 8,\n
\"min\": 55,\n        \"max\": 250,\n        \"num_unique_values\":
109,\n        \"samples\": [\n          125,\n          181,\n
```

```
151\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"weight\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 14.39575667851138,\n        \"min\": 10.0,\n        \"max\":
200.0,\n        \"num_unique_values\": 287,\n        \"samples\": [\n
68.0,\n          88.5,\n          145.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ap_hi\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 154,\n
\"min\": -150,\n        \"max\": 16020,\n
\"num_unique_values\": 153,\n        \"samples\": [\n          11500,\
n          17,\n          149\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"ap_lo\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 188,\n        \"min\": -70,\n
\"max\": 11000,\n        \"num_unique_values\": 157,\n
\"samples\": [\n          810,\n          8044,\n          107\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"age_mn\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.19103768232608603,\n        \"min\": 0.0,\n        \"max\":
0.9999999999999999,\n        \"num_unique_values\": 8076,\n
\"samples\": [\n          0.5047619047619046,\n
0.8237708091366628,\n          0.5287650019357336\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"height_mn\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.042103212125836086,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 109,\n        \"samples\": [\n
0.35897435897435903,\n          0.6461538461538462,\n
0.49230769230769234\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"weight_mn\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.07576714041321778,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 287,\n
\"samples\": [\n          0.30526315789473685,\n
0.4131578947368421,\n          0.7105263157894737\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ap_hi_mn\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.009524515736307445,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 153,\n        \"samples\": [\n
0.720470006184292,\n          0.010327767470624614,\n
0.018491032776747063\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ap_lo_mn\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.01702552215866217,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 157,\n
\"samples\": [\n          0.07949412827461609,\n
```

```
0.7329719963866306,\n          0.015989159891598916\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\
n}","type":"dataframe","variable_name":"df_noncat_mdt_mmn"}

# Pre-Processed Categorical Data Subset
df_cat_ppd = df_cat_mdt_code_oe.copy(); df_cat_ppd # Preferred Data
Subset
```

{"summary":"{\n \"name\": \"df_cat_ppd\",\n \"rows\": 70000,\n
\"fields\": [\n   {\n   \"column\": \"cluster_number_x\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0,\n   \"min\": 0,\n   \"max\": 2,\n
\"num_unique_values\": 3,\n   \"samples\": [\n   0,\n
1,\n   2\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"gender_x\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 1,\n
\"max\": 2,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   2\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"cholesterol_x\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 1,\n
\"max\": 3,\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   1,\n   3\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"gluc_x\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 1,\n
\"max\": 3,\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   1,\n   2\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"smoke_x\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"cardio_x\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"active_x\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   0,\n   1\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"alco_x\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0,\n   \"min\": 0,\n
\"max\": 1,\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   1,\n   0\n   ],\n   \"semantic_type\":

```
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"cluster_number_y\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.8611015896496835,\n
\"min\": 0.0,\n        \"max\": 2.0,\n        \"num_unique_values\":
3,\n        \"samples\": [\n          0.0,\n          1.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"gender_y\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.4768380155828638,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0.0,\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cholesterol_y\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.6802503486993808,\n        \"min\":
0.0,\n        \"max\": 2.0,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.0,\n          2.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"gluc_y\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.572270276613845,\n        \"min\": 0.0,\n        \"max\": 2.0,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          0.0,\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"smoke_y\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.28348381676993517,\n        \"min\": 0.0,\n
\"max\": 1.0,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1.0,\n          0.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cardio_y\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.5000034814661862,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1.0,\n
0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"active_y\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.39717906350492826,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0.0,\n          1.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"alco_y\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.22556770360410494,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1.0,\n
0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_cat_ppd"}
```

```
# Pre-Processed Non-Categorical Data Subset
df_noncat_ppd = df_noncat_mdt_mmn.copy(); df_noncat_ppd # Preferred
Data Subset
```

```
{"summary":"{\n  \"name\": \"df_noncat_ppd\",\n  \"rows\": 70000,\n
\"fields\": [\n    {\n      \"column\": \"age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2467,\n        \"min\": 10798,\n        \"max\": 23713,\n
\"num_unique_values\": 8076,\n        \"samples\": [\n
17317,\n          21437,\n          17627\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"height\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 8,\n
\"min\": 55,\n        \"max\": 250,\n        \"num_unique_values\":
109,\n        \"samples\": [\n          125,\n          181,\n
151\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"weight\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 14.39575667851138,\n        \"min\": 10.0,\n        \"max\":
200.0,\n        \"num_unique_values\": 287,\n        \"samples\": [\n
68.0,\n          88.5,\n          145.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ap_hi\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 154,\n
\"min\": -150,\n        \"max\": 16020,\n
\"num_unique_values\": 153,\n        \"samples\": [\n          11500,\
n          17,\n          149\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"ap_lo\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 188,\n        \"min\": -70,\n
\"max\": 11000,\n        \"num_unique_values\": 157,\n
\"samples\": [\n          810,\n          8044,\n          107\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"age_mn\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.19103768232608603,\n        \"min\": 0.0,\n        \"max\":
0.9999999999999999,\n        \"num_unique_values\": 8076,\n
\"samples\": [\n          0.5047619047619046,\n
0.8237708091366628,\n          0.5287650019357336\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"height_mn\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.042103212125836086,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 109,\n        \"samples\": [\n
0.35897435897435903,\n          0.6461538461538462,\n
0.49230769230769234\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"weight_mn\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.07576714041321778,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 287,\n
\"samples\": [\n          0.30526315789473685,\n
0.4131578947368421,\n          0.7105263157894737\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ap_hi_mn\",\n      \"properties\":
```

{\n   \"dtype\": \"number\",\n   \"std\": 0.009524515736307445,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 153,\n   \"samples\": [\n   0.720470006184292,\n   0.010327767470624614,\n   0.018491032776747063\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ap_lo_mn\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.01702552215866217,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 157,\n   \"samples\": [\n   0.07949412827461609,\n   0.7329719963866306,\n   0.015989159891598916\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"df_noncat_ppd"}

```
# Pre-Processed Dataset
df_ppd = pd.merge(df_cat_ppd, df_noncat_ppd, left_index=True,
right_index=True)

# Pre-Processed Non-Categorical Data Subset
df_noncat_ppd = df_noncat_mdt_mmn.copy(); df_noncat_ppd
```

{"summary":"{\n \"name\": \"df_noncat_ppd\",\n \"rows\": 70000,\n \"fields\": [\n   {\n   \"column\": \"age\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2467,\n   \"min\": 10798,\n   \"max\": 23713,\n   \"num_unique_values\": 8076,\n   \"samples\": [\n   17317,\n   21437,\n   17627\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"height\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 8,\n   \"min\": 55,\n   \"max\": 250,\n   \"num_unique_values\": 109,\n   \"samples\": [\n   125,\n   181,\n   151\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"weight\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 14.39575667851138,\n   \"min\": 10.0,\n   \"max\": 200.0,\n   \"num_unique_values\": 287,\n   \"samples\": [\n   68.0,\n   88.5,\n   145.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ap_hi\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 154,\n   \"min\": -150,\n   \"max\": 16020,\n   \"num_unique_values\": 153,\n   \"samples\": [\n   11500,\n   17,\n   149\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"ap_lo\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 188,\n   \"min\": -70,\n   \"max\": 11000,\n   \"num_unique_values\": 157,\n   \"samples\": [\n   810,\n   8044,\n   107\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n

```
}\n    },\n    {\n      \"column\": \"age_mn\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.19103768232608603,\n        \"min\": 0.0,\n        \"max\":
0.9999999999999999,\n        \"num_unique_values\": 8076,\n
\"samples\": [\n          0.5047619047619046,\n
0.8237708091366628,\n          0.5287650019357336\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"height_mn\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.042103212125836086,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 109,\n        \"samples\": [\n
0.35897435897435903,\n          0.6461538461538462,\n
0.49230769230769234\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"weight_mn\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.07576714041321778,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 287,\n
\"samples\": [\n          0.30526315789473685,\n
0.4131578947368421,\n          0.7105263157894737\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ap_hi_mn\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.009524515736307445,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 153,\n        \"samples\": [\n
0.720470006184292,\n          0.010327767470624614,\n
0.018491032776747063\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ap_lo_mn\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.01702552215866217,\n        \"min\":
0.0,\n        \"max\": 1.0,\n        \"num_unique_values\": 157,\n
\"samples\": [\n          0.07949412827461609,\n
0.7329719963866306,\n          0.015989159891598916\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df_noncat_ppd"}
```

```python
# Pre-Processed Dataset
df_ppd = pd.merge(df_cat_ppd, df_noncat_ppd, left_index=True,
right_index=True)

new_df=df_ppd.drop(['gender_x','cholesterol_x','gluc_x','smoke_x','car
dio_x',
                            'active_x','alco_x','cluster_number_y',
'age', 'height', 'weight','ap_hi','ap_lo'], axis=1)

new_df.columns
```

```
Index(['cluster_number_x', 'gender_y', 'cholesterol_y', 'gluc_y',
'smoke_y',
       'cardio_y', 'active_y', 'alco_y', 'age_mn', 'height_mn',
'weight_mn',
       'ap_hi_mn', 'ap_lo_mn'],
      dtype='object')
```

```
# Analysis Objective : Segment the Cardi based on cluster number and
other columns
# subsee
# Subset mtcars based on Inputs as {mpg, hp, cyl, vs} & Output as {am}
cardio_inputs = new_df[[ 'gender_y', 'cholesterol_y', 'gluc_y',
'smoke_y',
       'cardio_y','alco_y','active_y','height_mn', 'weight_mn',
'ap_hi_mn','ap_lo_mn']]; cardio_inputs
cardio_outputs=new_df[['cluster_number_x']]; #earthquake_outputs

cardio_inputs_names = cardio_inputs.columns; cardio_inputs_names
cardio_outputs_labels =
cardio_outputs['cluster_number_x'].unique().astype(str);
cardio_outputs_labels
```

```
array(['0', '1', '2'], dtype='<U21')
```

```
# Initialize StratifiedShuffleSplit with desired test size and random
state
stratified_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
random_state=45029)
# Perform the stratified split to get training and testing indices
for i, j in stratified_split.split(cardio_inputs, cardio_outputs):
    train_cardio_inputs, test_cardio_inputs = cardio_inputs.iloc[i],
cardio_inputs.iloc[j]
    train_cardio_outputs, test_cardio_outputs =
cardio_outputs.iloc[i], cardio_outputs.iloc[j]

# Decision Tree : Model (Training Subset)
dtc = DecisionTreeClassifier(criterion='gini',
random_state=45029,max_depth = 3) # Other Criteria : Entropy,  Log
Loss
dtc_model = dtc.fit(train_cardio_inputs, train_cardio_outputs);
dtc_model
```

```
DecisionTreeClassifier(max_depth=3, random_state=45029)
```

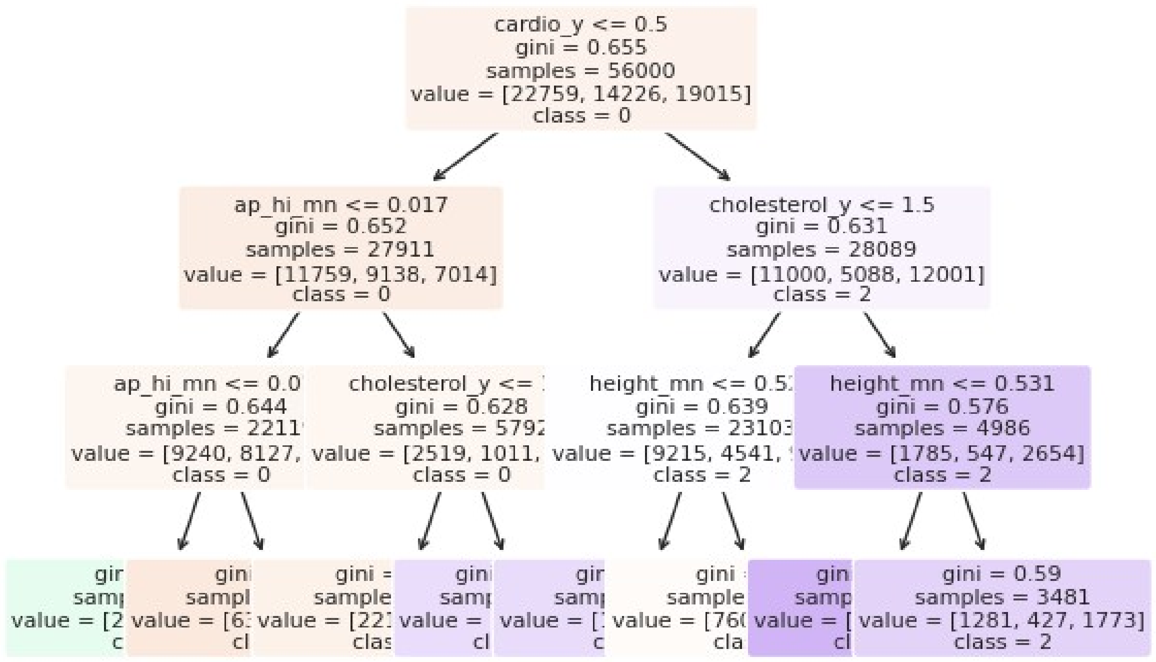
```
# Decision Tree : Model Rules
dtc_model_rules = export_text(dtc_model, feature_names =
list(cardio_inputs_names),); print(dtc_model_rules)
```

```
|--- cardio_y <= 0.50
|   |--- ap_hi_mn <= 0.02
|   |   |--- ap_hi_mn <= 0.02
|   |   |   |--- class: 1
|   |   |--- ap_hi_mn >  0.02
|   |   |   |--- class: 0
|   |--- ap_hi_mn >  0.02
|   |   |--- cholesterol_y <= 1.50
|   |   |   |--- class: 0
|   |   |--- cholesterol_y >  1.50
|   |   |   |--- class: 2
|--- cardio_y >  0.50
|   |--- cholesterol_y <= 1.50
|   |   |--- height_mn <= 0.53
|   |   |   |--- class: 2
|   |   |--- height_mn >  0.53
|   |   |   |--- class: 0
|   |--- cholesterol_y >  1.50
|   |   |--- height_mn <= 0.53
|   |   |   |--- class: 2
|   |   |--- height_mn >  0.53
|   |   |   |--- class: 2
```

```
# Decision Tree : Feature Importance
dtc_imp_features = pd.DataFrame({'feature': cardio_inputs_names,
'importance': np.round(dtc_model.feature_importances_, 2)})
dtc_imp_features.sort_values('importance', ascending=False,
inplace=True); dtc_imp_features
```

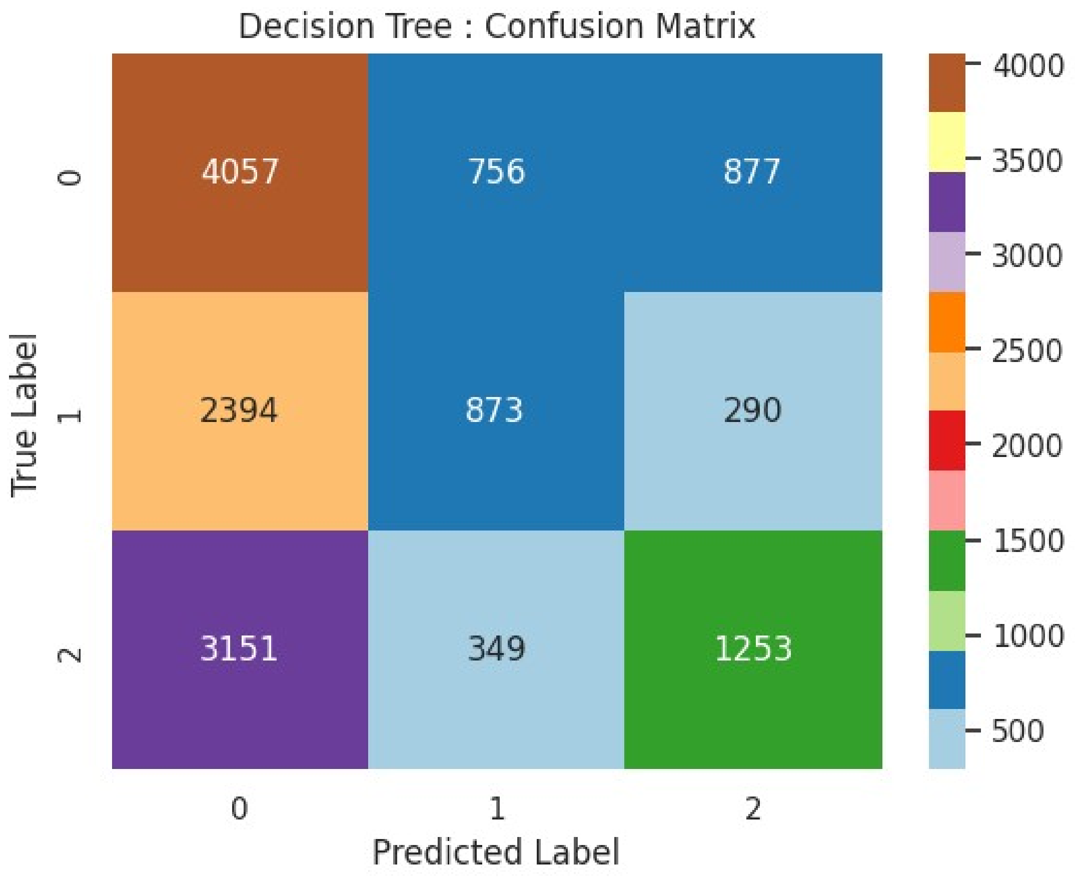
{"summary":"{\n \"name\": \"dtc_imp_features\",\n \"rows\": 11,\n
\"fields\": [\n   {\n   \"column\": \"feature\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 11,\n   \"samples\": [\n
\"gluc_y\",\n   \"cardio_y\",\n   \"weight_mn\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"importance\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.17868713134719832,\n   \"min\": 0.0,\n   \"max\": 0.54,\n
\"num_unique_values\": 5,\n   \"samples\": [\n   0.33,\n
0.0,\n   0.09\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"dtc_imp_features"}

```
# Decision Tree : Plot [Training Subset]
train_subset_dtc_plot = plot_tree(dtc_model,
feature_names=cardio_inputs_names, class_names=cardio_outputs_labels,
rounded=True, filled=True, fontsize=8)
plt.show()
```



```
# Decision Tree : Prediction (Testing Subset)
dtc_model_predict_test = dtc_model.predict(test_cardio_inputs);
dtc_model_predict_test

array([0, 0, 1, ..., 0, 0, 0])

# Decision Tree : Prediction Evaluation (Testing Subset)
dtc_predict_conf_mat =
pd.DataFrame(confusion_matrix(test_cardio_outputs,
dtc_model_predict_test));print(dtc_predict_conf_mat)
dtc_predict_perf = classification_report(test_cardio_outputs,
dtc_model_predict_test);  print(dtc_predict_perf)

      0    1     2
0  4057  756   877
1  2394  873   290
2  3151  349  1253
              precision    recall  f1-score   support

           0       0.42      0.71      0.53      5690
           1       0.44      0.25      0.32      3557
           2       0.52      0.26      0.35      4753

    accuracy                           0.44     14000
   macro avg       0.46      0.41      0.40     14000
weighted avg       0.46      0.44      0.41     14000
```

```
# Set up the plot
ax = plt.axes()

# Plot the confusion matrix with annotations in integer format
sns.heatmap(dtc_predict_conf_mat, annot=True, fmt='d', cmap='Paired')
# Set labels and title
ax.set_xlabel('Predicted Label')
ax.set_ylabel('True Label')
ax.set_title('Decision Tree : Confusion Matrix')

# Show the plot
plt.show()
```

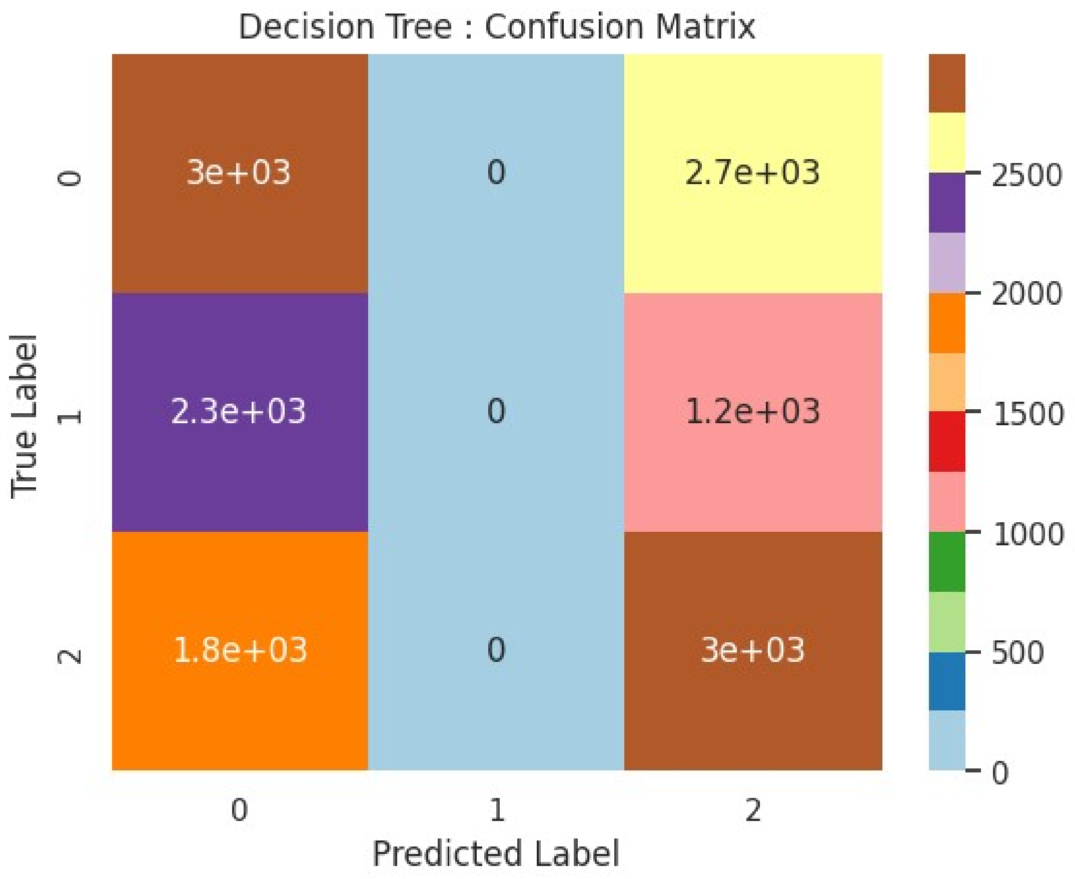


SVM

```
# Initialize the SVM classifier
svm_classifier = SVC(kernel='linear', random_state=4529)  # You can
choose different kernels like 'linear', 'poly', 'sigmoid', etc.

# Train the SVM model on the training subset
svm_model = svm_classifier.fit(train_cardio_inputs,
train_cardio_outputs)
svm_model

/usr/local/lib/python3.10/dist-packages/sklearn/utils/
validation.py:1143: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

SVC(kernel='linear', random_state=4529)

# Make predictions on the testing subset
svm_predict = svm_model.predict(test_cardio_inputs)
svm_predict

array([0, 0, 0, ..., 0, 0, 0])

# Decision Tree : Prediction Evaluation (Testing Subset)
svm_predict_conf_mat =
pd.DataFrame(confusion_matrix(test_cardio_outputs, svm_predict));
print(svm_predict_conf_mat)
svm_predict_perf = classification_report(test_cardio_outputs,
svm_predict); print(svm_predict_perf)

      0  1     2
0  2998  0  2692
1  2312  0  1245
2  1800  0  2953
              precision    recall  f1-score   support

           0       0.42      0.53      0.47      5690
           1       0.00      0.00      0.00      3557
           2       0.43      0.62      0.51      4753

    accuracy                           0.43     14000
   macro avg       0.28      0.38      0.33     14000
weighted avg       0.32      0.43      0.36     14000


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
```

```
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
# Confusion Matrix : Plot [Testing Subset]
ax = plt.axes()
sns.heatmap(svm_predict_conf_mat, annot=True, cmap='Paired')
ax.set_xlabel('Predicted Label')
ax.set_ylabel('True Label')
ax.set_title('Decision Tree : Confusion Matrix')
plt.show()
```



KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Initialize and fit the KNN classifier
n=5
knn_classifier = KNeighborsClassifier(n_neighbors=n)  # Adjust
n_neighbors as needed
knn_classifier.fit(train_cardio_inputs, train_cardio_outputs)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/
_classification.py:215: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  return self._fit(X, y)
```

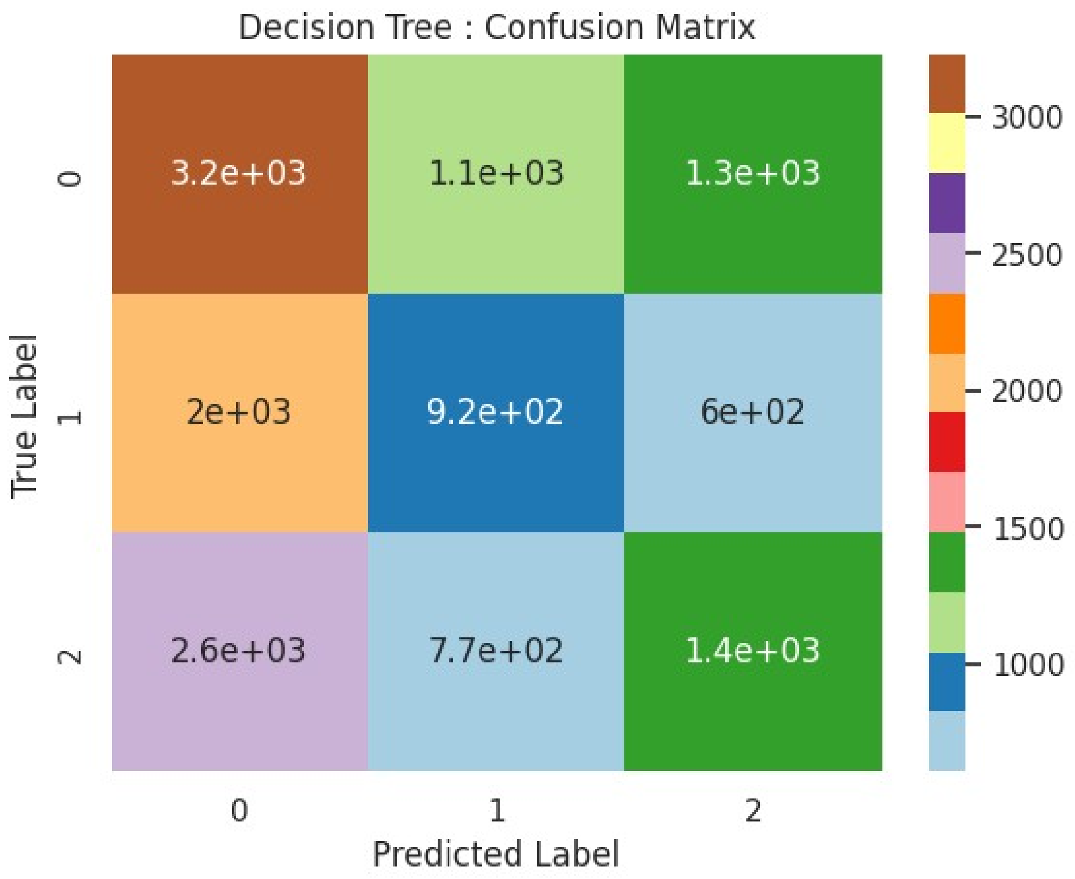
```
KNeighborsClassifier()
```

```
# Make predictions on the test dataset
knn_predictions = knn_classifier.predict(test_cardio_inputs)
```

```
# Decision Tree : Prediction Evaluation (Testing Subset)
knn_predict_conf_mat =
pd.DataFrame(confusion_matrix(test_cardio_outputs, knn_predictions));
print(knn_predict_conf_mat)
knn_predict_perf =
classification_report(test_cardio_outputs,knn_predictions);
print(knn_predict_perf)
```

```
      0     1     2
0  3227  1124  1339
1  2029   923   605
2  2550   774  1429
              precision    recall  f1-score   support

           0       0.41      0.57      0.48      5690
           1       0.33      0.26      0.29      3557
           2       0.42      0.30      0.35      4753

    accuracy                           0.40     14000
   macro avg       0.39      0.38      0.37     14000
weighted avg       0.39      0.40      0.39     14000
```

```
# Confusion Matrix : Plot [Testing Subset]
ax = plt.axes()
sns.heatmap(knn_predict_conf_mat, annot=True, cmap='Paired')
ax.set_xlabel('Predicted Label')
ax.set_ylabel('True Label')
ax.set_title('Decision Tree : Confusion Matrix')
plt.show()
```



```
k_values = [7, 9, 11, 13, 15]
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(train_cardio_inputs, train_cardio_outputs)  # Use your
training data here
    y_pred = knn.predict(test_cardio_inputs)  # Use your testing data
here
    accuracy = accuracy_score(test_cardio_outputs,y_pred)  # Compare
predictions with true labels
    print(f'Accuracy at k={k}: {accuracy}')
```

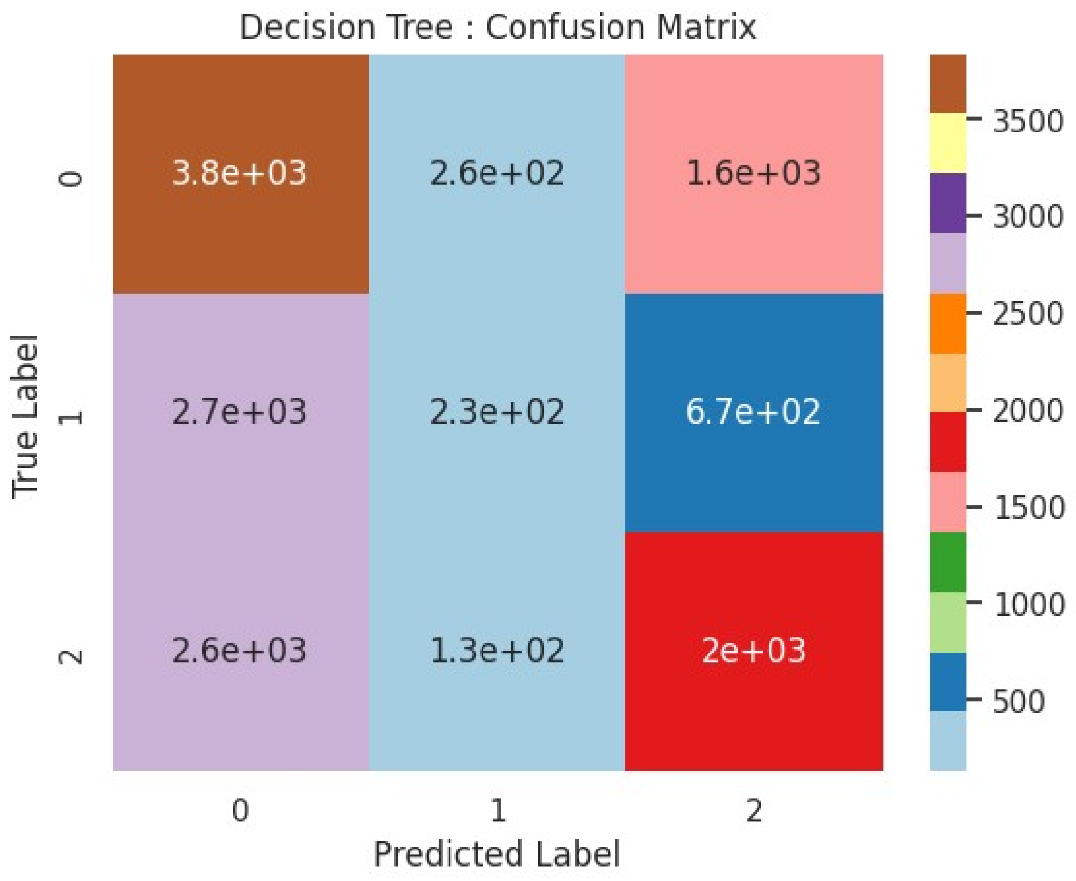
```
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/
_classification.py:215: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  return self._fit(X, y)

Accuracy at k=7: 0.4070714285714286

/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/
_classification.py:215: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
```

```
(n_samples,), for example using ravel().
  return self._fit(X, y)

Accuracy at k=9: 0.40985714285714286

/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/
_classification.py:215: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  return self._fit(X, y)

Accuracy at k=11: 0.4097142857142857

/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/
_classification.py:215: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  return self._fit(X, y)

Accuracy at k=13: 0.4077857142857143

/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/
_classification.py:215: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  return self._fit(X, y)

Accuracy at k=15: 0.41064285714285714
```

Logistic Regresion

```
from sklearn.linear_model import LogisticRegression

# Creating and training the logistic regression model
logistic_regression = LogisticRegression()
logistic_regression.fit(train_cardio_inputs, train_cardio_outputs)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/
validation.py:1143: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

LogisticRegression()

# Making predictions on the test set
log_predictions = logistic_regression.predict(test_cardio_inputs)

# Decision Tree : Prediction Evaluation (Testing Subset)
log_predict_conf_mat =
pd.DataFrame(confusion_matrix(test_cardio_outputs, log_predictions));
print(log_predict_conf_mat)
log_predict_perf =
classification_report(test_cardio_outputs,log_predictions);
print(log_predict_perf)

      0    1     2
0  3833  260  1597
1  2657  227   673
2  2648  130  1975
              precision    recall  f1-score   support

           0       0.42      0.67      0.52      5690
           1       0.37      0.06      0.11      3557
           2       0.47      0.42      0.44      4753

    accuracy                           0.43     14000
   macro avg       0.42      0.38      0.35     14000
weighted avg       0.42      0.43      0.39     14000


# Confusion Matrix : Plot [Testing Subset]
ax = plt.axes()
sns.heatmap(log_predict_conf_mat, annot=True, cmap='Paired')
ax.set_xlabel('Predicted Label')
ax.set_ylabel('True Label')
ax.set_title('Decision Tree : Confusion Matrix')
plt.show()
```



Decision Tree : Confusion Matrix
True Label
Predicted Label
0
1
2
3.8e+03
2.6e+02
1.6e+03
2.7e+03
2.3e+02
6.7e+02
2.6e+03
1.3e+02
2e+03
3500
3000
2500
2000
1500
1000
500