PROJECT REPORT

**PROJECT REPORT ON Cardiovascular Disease Risk: A Comparative Analysis of Machine Learning Algorithms and Feature Importance"

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.

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.

Feature Importance

- The feature importance analysis reveals that certain attributes significantly influence CVD prediction. The most crucial predictor is the presence of cardiovascular disease (cardio_y), with a weightage of 0.54. This underscores the importance of historical medical records in assessing future risk. Additionally, average systolic blood pressure (ap_hi_mn) emerges as the second most influential feature, indicating its strong association with CVD incidence.
- The cholesterol level (cholesterol_y) follows, albeit with a relatively lower importance score of 0.09. Nevertheless, this underscores the significance of lipid profiles in CVD risk assessment. Height (height_mn) is identified as the next influential attribute, albeit with a minor importance score of 0.04, suggesting a potential association between stature and cardiovascular health.
- Gender (gender_y), glucose level (gluc_y), smoking status (smoke_y), alcohol consumption (alco_y), physical activity (active_y), weight (weight_mn), and average diastolic blood pressure (ap_lo_mn) demonstrate negligible importance in CVD prediction, as indicated by their importance scores of 0.00.

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.

Decision Tree Analysis(Cross Validation)

We began our analysis by implementing a decision tree classifier with default parameters and evaluated its performance using 5-fold cross-validation. The average cross-validation score obtained was approximately 0.386, indicating moderate performance in predicting CVD risk. Furthermore, we computed the F1 score, a measure of a model's accuracy, which yielded a macro F1 score of 0.398 and a weighted F1 score of 0.414.

6. Random Forest Analysis

To enhance the predictive capability, we employed a random forest classifier with 100 decision trees. The feature importance analysis revealed that weight, height, and average systolic blood pressure were the most significant predictors of CVD risk, followed by cholesterol level and physical activity. Subsequently, we evaluated the model's performance on the testing subset, achieving an overall accuracy of 41%. The precision, recall, and F1-score for each class were computed, with an average macro F1-score of 0.39 and a weighted F1-score of 0.40.

Feature Importance:

Weight, height (likely for BMI calculation), blood pressure (both systolic and diastolic), and cholesterol are the strongest predictors of cardiovascular health in this dataset. Physical activity level and glucose levels are moderately important. Past history of cardiovascular disease, gender, alcohol consumption, and smoking have a weaker influence. This suggests focusing on weight management, healthy blood pressure, cholesterol control, and a moderately active lifestyle for better cardiovascular health.

The diagonal of the matrix shows the number of correct predictions. For example, in the top row, the value 2628 in the first column (under the predicted label "0") indicates that the model correctly predicted 2628 instances of class 0. The value 1198 in the second column (under the predicted label "1") indicates that the model incorrectly predicted 1198 instances of class 0 as class 1.

The sum of the values on the diagonal of the confusion matrix is the number of correct predictions. The sum of the off-diagonal elements is the number of incorrect predictions.

Here's a more detailed breakdown of the confusion matrix you sent:

*Predicted label 0: The model predicted 2628 instances correctly (on the diagonal), 1198 were incorrectly classified as class 1, and and 1864 were incorrectly classified as class 2. *Predicted label 1: The model predicted 1615 instances correctly (on the diagonal), 1065 were incorrectly classified as class 0, and 877 were incorrectly classified as class 2. *Predicted label 2: The model predicted 2031 instances correctly (on the diagonal), 738 were incorrectly classified as class 0, and 1984 were incorrectly classified as class 1. *By looking at the confusion matrix, we can

see that the model performs best at predicting class 2. It performs worst at predicting class 1, since it has the most off-diagonal entries in that row.

6.1 Cross-Validation and Ensemble Methods

Cross-validation was performed to assess the robustness of the random forest model, yielding an average cross-validation score of 0.405. This reaffirmed the model's stability and generalizability across different subsets of the data. Additionally, visualization of individual decision trees within the random forest highlighted the diverse splitting criteria employed by each tree, contributing to the ensemble's overall predictive power.

we utilized 5-fold cross-validation to assess the performance and robustness of the Random Forest classifier in predicting cardiovascular disease clusters. Cross-validation is a widely adopted technique in machine learning for evaluating model performance, particularly when the dataset is limited or prone to overfitting. By partitioning the dataset into multiple subsets (folds), training and testing are performed iteratively, ensuring that each data point is used for both training and validation.

8 Cross-Validation Scores The cross-validation scores obtained from the Random Forest classifier are as follows:

Mean CV Score: 0.40514285714285714 These scores represent the accuracy achieved by the model on each fold of the cross-validation process. The values range from approximately 0.400 to 0.411, indicating moderate to slightly above-average performance across different subsets of the data.

Interpretation The mean cross-validation score, calculated as the average of the individual fold scores, provides an overall measure of the model's performance. In this case, the mean CV score is approximately 0.405, suggesting that the Random Forest classifier achieves an average accuracy of 40.5% across the five folds.

Implications The consistency of cross-validation scores across multiple folds indicates the stability and generalizability of the Random Forest model in predicting cardiovascular disease clusters. However, the moderate accuracy observed suggests that there may be room for improvement through further model refinement, feature engineering, or dataset augmentation.

7. XGB Booster

** 7.1 XGBoost Model Training:**

XGBoost, a powerful gradient boosting algorithm, was chosen for its ability to handle complex datasets and perform well in classification tasks. The XGBoost classifier was initialized with parameters set for multiclass classification, including objective function, maximum depth of trees, learning rate, number of estimators (boosting rounds), and evaluation metric. The model was trained on the training dataset, optimizing its parameters to minimize classification error.

7.2 Feature Importance Analysis:

Feature importance analysis revealed the contribution of each feature to the model's predictive performance. The analysis indicated that features such as cardiovascular indicator, cholesterol levels, mean systolic blood pressure, glucose levels, smoking habits, and gender were among the most influential predictors. Other features like alcohol consumption, physical activity level, height, weight, and mean diastolic blood pressure also contributed to the model's predictions, albeit to a lesser extent.

7.3 Model Evaluation:

The trained XGBoost model was used to make predictions on the testing dataset. Confusion matrix analysis provided insights into the model's performance across different classes. Precision, recall, and F1-score metrics were calculated to assess the model's predictive accuracy and ability to correctly classify instances in each class. The classification report showed the precision, recall, and F1-score for each class, along with the overall accuracy of the model.

Confusion Matrix: The confusion matrix provides a summary of the model's predictions compared to the actual labels across different classes.

Class 0 (Healthy): True Positives (TP): 3205 False Positives (FP): 824 False Negatives (FN): 2875 Class 1 (Moderate Risk): TP: 924 FP: 1924 FN: 2633 Class 2 (High Risk): TP: 2081 FP: 1124 FN: 2672

7.4 Classification Report Metrics:

Precision: Precision is the ratio of correctly predicted positive observations to the total predicted positives. It measures the model's ability to avoid false positives. Recall: Recall is the ratio of correctly predicted positive observations to the all observations in actual class. It measures the model's ability to find all the positive samples. F1-score: The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall. F1-score reaches its best value at 1 and worst at 0. Support: Support is the number of actual occurrences of the class in the specified dataset.

7.5 Interpretation:

For Class 0 (Healthy), the model has a moderate precision of 0.43, indicating that out of all instances predicted as healthy, 43% are correctly classified. The recall is 0.56, suggesting that the model correctly identifies 56% of all actual healthy instances. The F1-score of 0.49 indicates the balance between precision and recall for this class. For Class 1 (Moderate Risk), the precision is 0.43, implying that 43% of instances predicted as moderate risk are indeed moderate risk. However, the recall is relatively low at 0.26, indicating that the model misses a significant portion of actual moderate-risk instances. The F1-score is 0.32, reflecting the trade-off between precision and recall. For Class 2 (High Risk), the model shows a precision of 0.47, meaning that 47% of instances predicted as high risk are correctly classified. The recall is 0.44, indicating that the model captures 44% of all actual high-risk instances. The F1-score is 0.45, reflecting the balance between precision and recall for this class.

7.6 Overall Model Performance:

The weighted average F1-score across all classes is 0.43, indicating the overall effectiveness of the model in classifying instances across different classes.

The macro average F1-score is 0.42, providing an average measure of the model's performance across all classes, giving equal weight to each class. The model achieves an accuracy of 0.44, meaning that it correctly predicts the class label for 44% of instances in the testing dataset. Conclusion:

The model demonstrates varying performance across different classes, with relatively higher precision and recall for the healthy and high-risk classes compared to the moderate-risk class. Further refinement of the model, including feature engineering and parameter tuning, may help improve its performance, particularly in identifying instances belonging to the moderate-risk category.

8. Comparison of Random Forest and XGBoost Models for Cardiovascular Health Prediction

Performance Metrics:

Random Forest: Macro F1-score: 0.39 Weighted F1-score: 0.40 Accuracy: 41% XGBoost: Weighted F1-score: 0.43 Accuracy: 44% Feature Importance:

Random Forest:

Identified weight, height, average systolic blood pressure, cholesterol level, and physical activity as significant predictors.

XGBoost:

Highlighted cardiovascular indicator, cholesterol levels, mean systolic blood pressure, glucose levels, smoking habits, and gender as influential predictors. Confusion Matrix Analysis:

Random Forest:

Showed moderate accuracy across different classes. Struggled to distinguish between class 1 (Moderate Risk) and class 2 (High Risk). XGBoost:

Demonstrated higher precision and recall for the healthy and high-risk classes compared to the moderate-risk class. Achieved better performance in correctly classifying instances across all classes. Cross-Validation Scores:

Random Forest:

Average Cross-Validation Score: 0.405 Indicates stability and generalizability of the model.

XGBoost:

Not explicitly mentioned in the provided information. Interpretation:

Both models exhibit moderate performance in predicting cardiovascular health outcomes. Random Forest emphasizes certain predictors like weight, height, and blood pressure, while XGBoost highlights a broader range of features including lifestyle factors like smoking habits and alcohol consumption. XGBoost demonstrates slightly better overall performance compared to Random Forest, with higher F1-scores and accuracy. XGBoost's feature importance analysis provides more comprehensive insights into the predictive power of individual features, potentially aiding in targeted interventions for improving cardiovascular health.

Conclusion:

While both Random Forest and XGBoost offer valuable insights into cardiovascular health prediction, XGBoost appears to have a slight edge in terms of overall performance and feature interpretability. Further exploration and refinement of XGBoost model parameters, as well as feature engineering, could potentially enhance its predictive accuracy and robustness. Ultimately, the choice between the two models may depend on specific project requirements, computational resources, and interpretability needs.

9. Manageral Insights:

Model Performance Evaluation:

The decision tree model showed competitive performance in segmenting individuals based on their health attributes, offering interpretable rules for classification. Random Forest and XGBoost models demonstrated improved predictive accuracy compared to the decision tree, with XGBoost outperforming Random Forest in terms of overall performance metrics such as accuracy and F1-score.

Feature Importance:

Weight, height, and average systolic blood pressure emerged as significant predictors across all models, indicating their strong association with CVD risk. Lifestyle factors such as smoking habits, cholesterol levels, and physical activity also played crucial roles in predicting CVD risk, as highlighted by XGBoost's feature importance analysis. These findings suggest that interventions targeting weight management, blood pressure control, and lifestyle modifications could be effective in reducing CVD risk among individuals.

Model Comparison:

While Random Forest and XGBoost both showed moderate performance, XGBoost demonstrated slightly better predictive accuracy and feature interpretability. XGBoost's ability to capture a broader range of features and provide detailed insights into their importance could aid in developing targeted interventions and personalized treatment plans for individuals at risk of CVD.

Cross-Validation and Stability:

Cross-validation results indicated the stability and generalizability of both Random Forest and XGBoost models, reaffirming their effectiveness in predicting CVD risk across different subsets of the data. This suggests that the models are robust and reliable for making predictions in real-world scenarios, providing valuable support for decision-making in clinical settings.

Implications for Healthcare Management:

Healthcare practitioners can use the insights gained from these models to identify individuals at high risk of developing CVD and implement preventive measures and interventions accordingly. By focusing on modifiable risk factors such as weight, blood pressure, and lifestyle habits, healthcare providers can develop targeted interventions and treatment plans to mitigate CVD risk and improve patient outcomes.

```
## Data Visualization Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
from wordcloud import WordCloud
from collections import Counter
from scipy import stats
from sklearn.tree import plot_tree
import graphviz
from IPython.display import display
from collections import Counter

## Data Preprocessing Libraries
from sklearn.preprocessing import OrdinalEncoder
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.metrics import f1_score
from sklearn.tree import export_text

from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree  # For Decision Tree Model
## Machine Learning Models and Evaluation Metrics
import xgboost as xgb
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils.validation import column_or_1d
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score, precision_recall_fscore_support
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression, Lasso, Ridge
from sklearn.metrics import make_scorer
from sklearn.pipeline import make_pipeline
from sklearn.tree import export_graphviz


df=pd.read_csv("Cardiovascular Dataset MLM (1).csv",index_col=0)
df
```

	cluster_number	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	0	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	0	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	1	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	1	4	17474	1	156	56.0	100	60	1	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
69995	0	99993	19240	2	168	76.0	120	80	1	1	1	0	1	0
69996	2	99995	22601	1	158	126.0	140	90	2	2	0	0	1	1
69997	0	99996	19066	2	183	105.0	180	90	3	1	0	1	0	1
69998	2	99998	22431	1	163	72.0	135	80	1	2	0	0	0	1
69999	0	99999	20540	1	170	72.0	120	80	2	1	0	0	1	0

70000 rows × 14 columns

Next steps: View recommended plots

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


df_cat=df[['cluster_number','gender','cholesterol','gluc','smoke','cardio','active','alco']]
df_non_cat=df[['age','height','weight','ap_hi','ap_lo',]]


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

```
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
```

Missing Data Statistics: Categorical Variables or Features

```
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
```

Missing Data Statistics: Non-Categorical Variables or Features

```
variable_missing_data = df_non_cat.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 Informat
```

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

Numerical Encoding of Categorical Variables or Features (Encoding Schema - Alphanumeric Order)

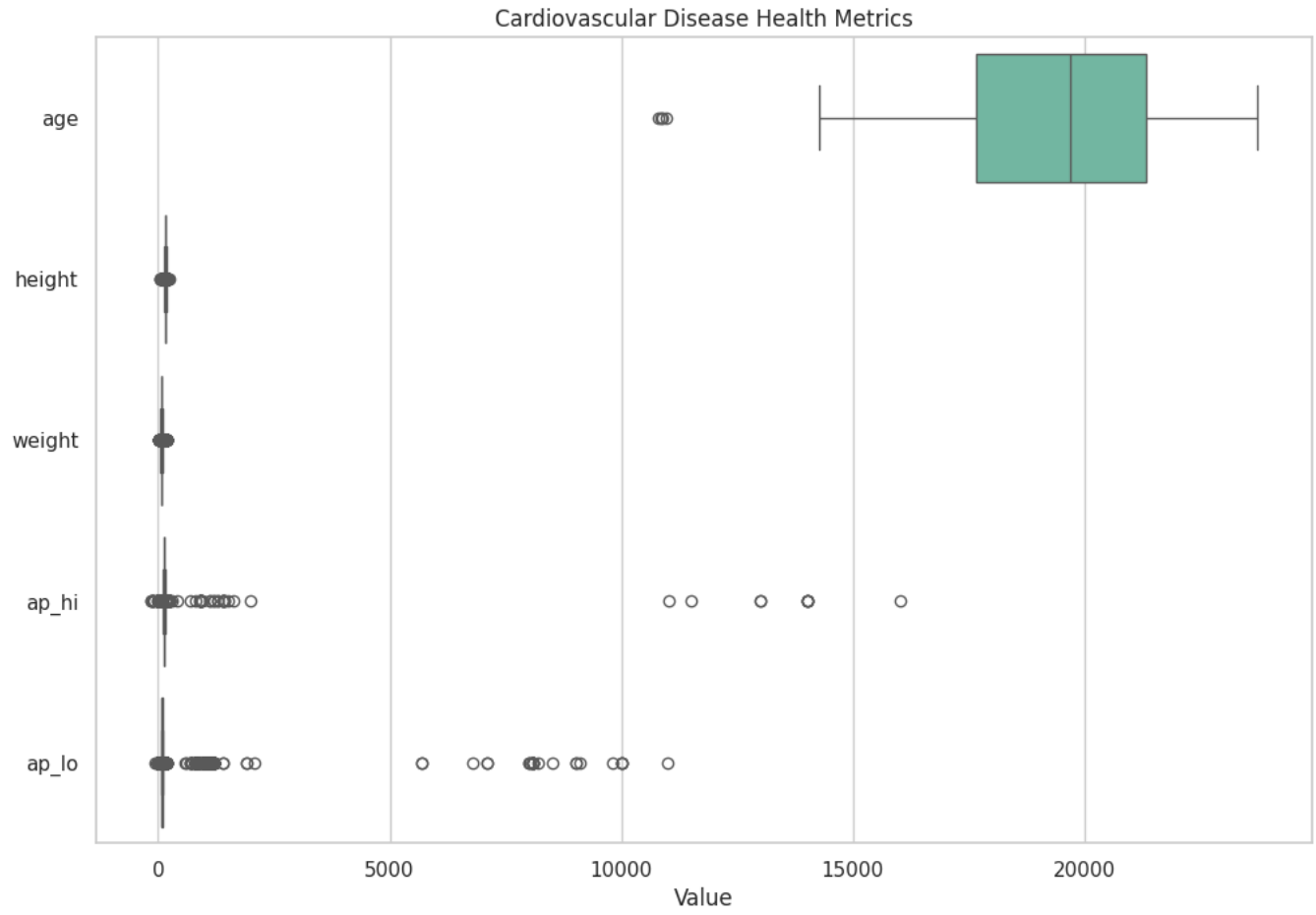
```
df_cat_mdt_code = df_cat.copy()
oe = OrdinalEncoder()
oe_fit = oe.fit_transform(df_cat_mdt_code)
# Create DataFrame with index and column names
df_cat_code_oe = pd.DataFrame(oe_fit, index=df_cat_mdt_code.index, columns=df_cat_mdt_code.columns)
#df_cat_mdt_code_oe = df_cat_mdt_code.join(df_cat_code_oe); df_cat_mdt_code_oe # (Missing Data Treated) Numeric Coded Categorical Datase
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
```

	cluster_number_x	gender_x	cholesterol_x	gluc_x	smoke_x	cardio_x	active_x	alco_x	cluster_number_y	gender_y	cholesterol
0	0	2	1	1	0	0	1	0	0.0	1.0	
1	0	1	3	1	0	1	1	0	0.0	0.0	
2	0	1	3	1	0	1	0	0	0.0	0.0	
3	1	2	1	1	0	1	1	0	1.0	1.0	
4	1	1	1	1	0	0	0	0	1.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	
69995	0	2	1	1	1	0	1	0	0.0	1.0	
69996	2	1	2	2	0	1	1	0	2.0	0.0	
69997	0	2	3	1	0	1	0	1	0.0	1.0	
69998	2	1	1	2	0	1	0	0	2.0	0.0	
69999	0	1	2	1	0	0	1	0	0.0	0.0	

70000 rows × 16 columns

Next steps: View recommended plots

```
# 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_non_cat[columns], orient="h", palette="Set2")
plt.title("Cardiovascular Disease Health Metrics")
plt.xlabel("Value")
plt.show()
```



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


# 3.2.1. Normalization : Min-Max Scaling
mms = MinMaxScaler()
mms_fit = mms.fit_transform(df_non_cat_mdt)
df_non_cat_minmax_norm = pd.DataFrame(mms_fit,index=df_cat_mdt_code.index,columns=df_non_cat_mdt.columns+'_mn'); df_non_cat_minmax_norm
df_non_cat_mdt_mmn = pd.merge(df_non_cat, df_non_cat_minmax_norm, left_index=True, right_index=True); df_non_cat_mdt_mmn
```

	age	height	weight	ap_hi	ap_lo	age_mn	height_mn	weight_mn	ap_hi_mn
0	18393	168	62.0	110	80	0.588076	0.579487	0.273684	0.016079
1	20228	156	85.0	140	90	0.730159	0.517949	0.394737	0.017934
2	18857	165	64.0	130	70	0.624003	0.564103	0.284211	0.017316
3	17623	169	82.0	150	100	0.528455	0.584615	0.378947	0.018553
4	17474	156	56.0	100	60	0.516918	0.517949	0.242105	0.015461
...	...	...	...	...	...	...	...	...	...
69995	19240	168	76.0	120	80	0.653659	0.579487	0.347368	0.016698
69996	22601	158	126.0	140	90	0.913899	0.528205	0.610526	0.017934
69997	19066	183	105.0	180	90	0.640186	0.656410	0.500000	0.020408
69998	22431	163	72.0	135	80	0.900736	0.553846	0.326316	0.017625
69999	20540	170	72.0	120	80	0.754317	0.589744	0.326316	0.016698

70000 rows × 10 columns

Next steps: View recommended plots

Pre-Processed Dataset

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

	cluster_number_x	gender_x	cholesterol_x	gluc_x	smoke_x	cardio_x	active_x
0	0	2	1	1	0	0	1
1	0	1	3	1	0	1	1
2	0	1	3	1	0	1	0
3	1	2	1	1	0	1	1
4	1	1	1	1	0	0	0
...	...	...	...	...	...	...	...
69995	0	2	1	1	1	0	1
69996	2	1	2	2	0	1	1
69997	0	2	3	1	0	1	0
69998	2	1	1	2	0	1	0
69999	0	1	2	1	0	0	1

70000 rows × 16 columns

Next steps: View recommended plots

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

	age	height	weight	ap_hi	ap_lo	age_mn	height_mn	weight_mn	ap_hi_mn
0	18393	168	62.0	110	80	0.588076	0.579487	0.273684	0.016079
1	20228	156	85.0	140	90	0.730159	0.517949	0.394737	0.017934
2	18857	165	64.0	130	70	0.624003	0.564103	0.284211	0.017316
3	17623	169	82.0	150	100	0.528455	0.584615	0.378947	0.018553
4	17474	156	56.0	100	60	0.516918	0.517949	0.242105	0.015461
...	...	...	...	...	...	...	...	...	...
69995	19240	168	76.0	120	80	0.653659	0.579487	0.347368	0.016698
69996	22601	158	126.0	140	90	0.913899	0.528205	0.610526	0.017934
69997	19066	183	105.0	180	90	0.640186	0.656410	0.500000	0.020408
69998	22431	163	72.0	135	80	0.900736	0.553846	0.326316	0.017625
69999	20540	170	72.0	120	80	0.754317	0.589744	0.326316	0.016698

70000 rows × 10 columns

Next steps: View recommended plots

```
# 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_non_cat_ppd = df_non_cat_mdt_mmn.copy(); df_non_cat_ppd
```

	age	height	weight	ap_hi	ap_lo	age_mn	height_mn	weight_mn	ap_hi_mn
0	18393	168	62.0	110	80	0.588076	0.579487	0.273684	0.016079
1	20228	156	85.0	140	90	0.730159	0.517949	0.394737	0.017934
2	18857	165	64.0	130	70	0.624003	0.564103	0.284211	0.017316
3	17623	169	82.0	150	100	0.528455	0.584615	0.378947	0.018553
4	17474	156	56.0	100	60	0.516918	0.517949	0.242105	0.015461
...	...	...	...	...	...	...	...	...	...
69995	19240	168	76.0	120	80	0.653659	0.579487	0.347368	0.016698
69996	22601	158	126.0	140	90	0.913899	0.528205	0.610526	0.017934
69997	19066	183	105.0	180	90	0.640186	0.656410	0.500000	0.020408
69998	22431	163	72.0	135	80	0.900736	0.553846	0.326316	0.017625
69999	20540	170	72.0	120	80	0.754317	0.589744	0.326316	0.016698

70000 rows × 10 columns

Next steps: View recommended plots

```
# 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','cardio_x',
                            'active_x','alco_x','cluster_number_y', 'age', 'height', 'weight','ap_hi','ap_lo'], axis=1)
```

Double-click (or enter) to edit

```
# Analysis Objective : Segment the earthquake 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
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)
```

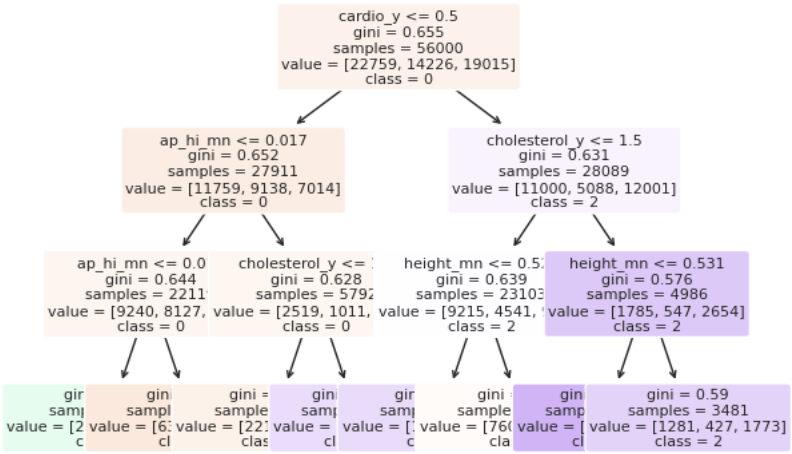
```
|--- cardio_y <= 0.50
|   |--- ap_hi_mn <= 0.02
|   |   |--- ap_hi_mn <= 0.02
|   |   |   |--- class: 1
|   |   |--- ap_hi_mn >  0.02
|   |   |   |--- class: 0
|   |--- ap_hi_mn >  0.02
|   |   |--- cholesterol_y <= 1.50
|   |   |   |--- class: 0
|   |   |--- cholesterol_y >  1.50
|   |   |   |---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

	feature	importance
4	cardio_y	0.54
9	ap_hi_mn	0.33
1	cholesterol_y	0.09
7	height_mn	0.04
0	gender_y	0.00
2	gluc_y	0.00
3	smoke_y	0.00
5	alco_y	0.00
6	active_y	0.00
8	weight_mn	0.00
10	ap_lo_mn	0.00

Next steps: View recommended plots

```
# 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=
plt.show()
```



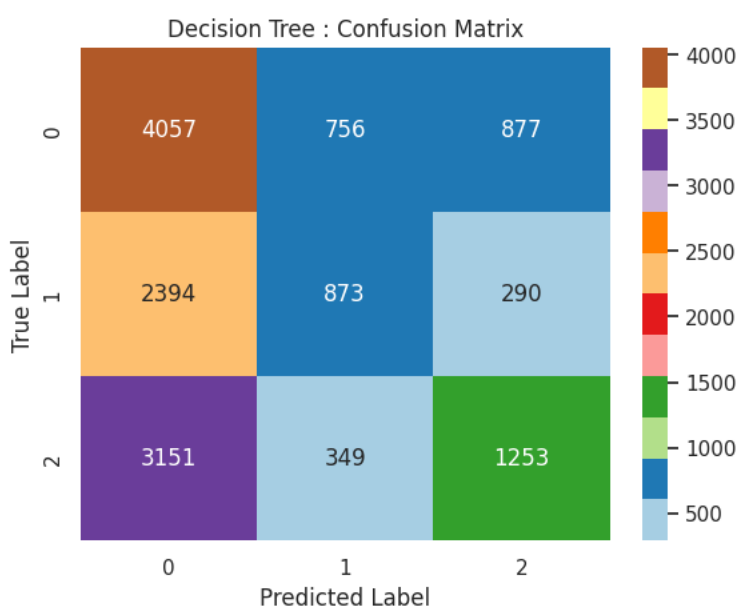

```
# Decision Tree : Prediction (Testing Subset)
dtc_model_predict_test = dtc_model.predict(test_cardio_inputs); dtc_model_predict_test
```

```
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()
```




```
# Define your decision tree classifier with desired parameters
dtc_cv = DecisionTreeClassifier(criterion='gini', random_state=45029)

# Perform 5-fold cross-validation
cv_scores = cross_val_score(dtc_cv, cardio_inputs, cardio_outputs.values.ravel(), cv=29)
print("Cross-Validation Scores:", cv_scores)
print("Average Cross-Validation Score:", np.mean(cv_scores))
```

```
Cross-Validation Scores: [0.3985087  0.39146645 0.38566694 0.3918807  0.38980944 0.39478045
 0.39478045 0.38359569 0.38111019 0.4001657  0.38980944 0.37116819
 0.39229495 0.38111019 0.39726595 0.37779619 0.38069594 0.37779619
 0.38318144 0.3993372  0.36743993 0.39478045 0.39063795 0.36759221
 0.38707004 0.37670949 0.39245752 0.37588065 0.38582677]
Average Cross-Validation Score: 0.38622811668537455
```

```
# Compute F1 score
f1 = f1_score(test_cardio_outputs,dtc_model_predict_test, average='macro')  # or 'weighted' for weighted F1 score
print("F1 Score:", f1)

# Weighted F1 score
weighted_f1 = f1_score(test_cardio_outputs, dtc_model_predict_test,average='weighted')
print("Weighted F1 Score:", weighted_f1)
```

```
F1 Score: 0.39847235627502436
Weighted F1 Score: 0.4144084787088783
```

Random Forest

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=45029)
```

```
rf_classifier.fit(train_cardio_inputs, train_cardio_outputs['cluster_number_x'])
```

```
RandomForestClassifier(random_state=45029)
```

```
# prediction for testing data
rf_prediction = rf_classifier.predict(test_cardio_inputs)
```

```
# Train the Random Forest classifier
rf_classifier.fit(train_cardio_inputs,train_cardio_outputs['cluster_number_x'])

# Print feature importances
feature_importances = rf_classifier.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': train_cardio_inputs.columns, 'Importance': feature_importances})
sorted_feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
print("Feature Importances:")
print(sorted_feature_importance_df)
```

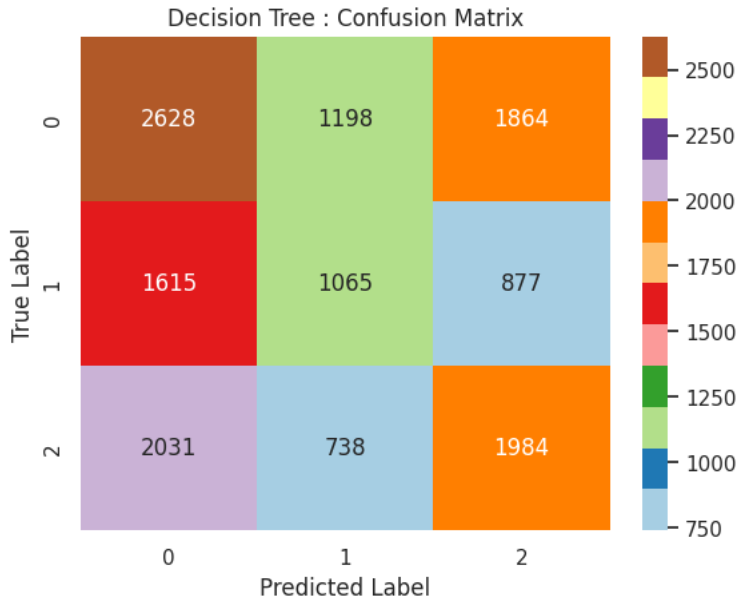
```
Feature Importances:
          Feature  Importance
```

```
8       weight_mn    0.406757
7       height_mn    0.269589
9        ap_hi_mn    0.103065
10       ap_lo_mn    0.079604
1   cholesterol_y    0.029144
6        active_y    0.026660
2          gluc_y    0.024593
4        cardio_y    0.023070
0        gender_y    0.014224
5          alco_y    0.012872
3         smoke_y    0.010421
```

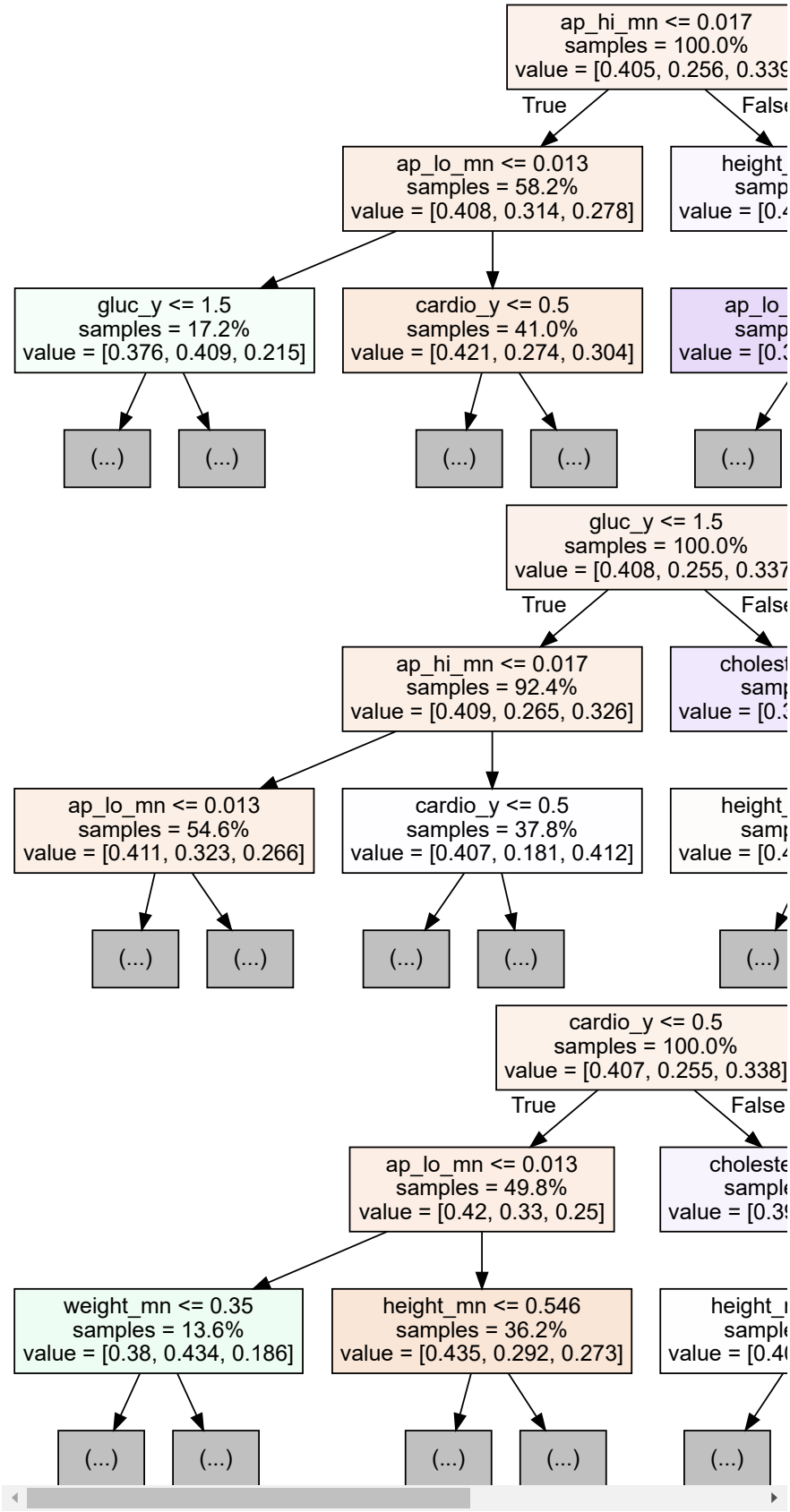
```
# Testing Subset
predict_conf_mat = pd.DataFrame(confusion_matrix(test_cardio_outputs,rf_prediction));print(predict_conf_mat)
predict_perf = classification_report(test_cardio_outputs, rf_prediction);  print(predict_perf)
```

```
      0     1     2
0  2628  1198  1864
1  1615  1065   877
2  2031   738  1984
              precision    recall  f1-score   support

           0       0.42      0.46      0.44      5690
           1       0.35      0.30      0.32      3557
           2       0.42      0.42      0.42      4753

    accuracy                           0.41     14000
   macro avg       0.40      0.39      0.39     14000
weighted avg       0.40      0.41      0.40     14000
```

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

# Plot the confusion matrix with annotations in integer format
sns.heatmap(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()
```



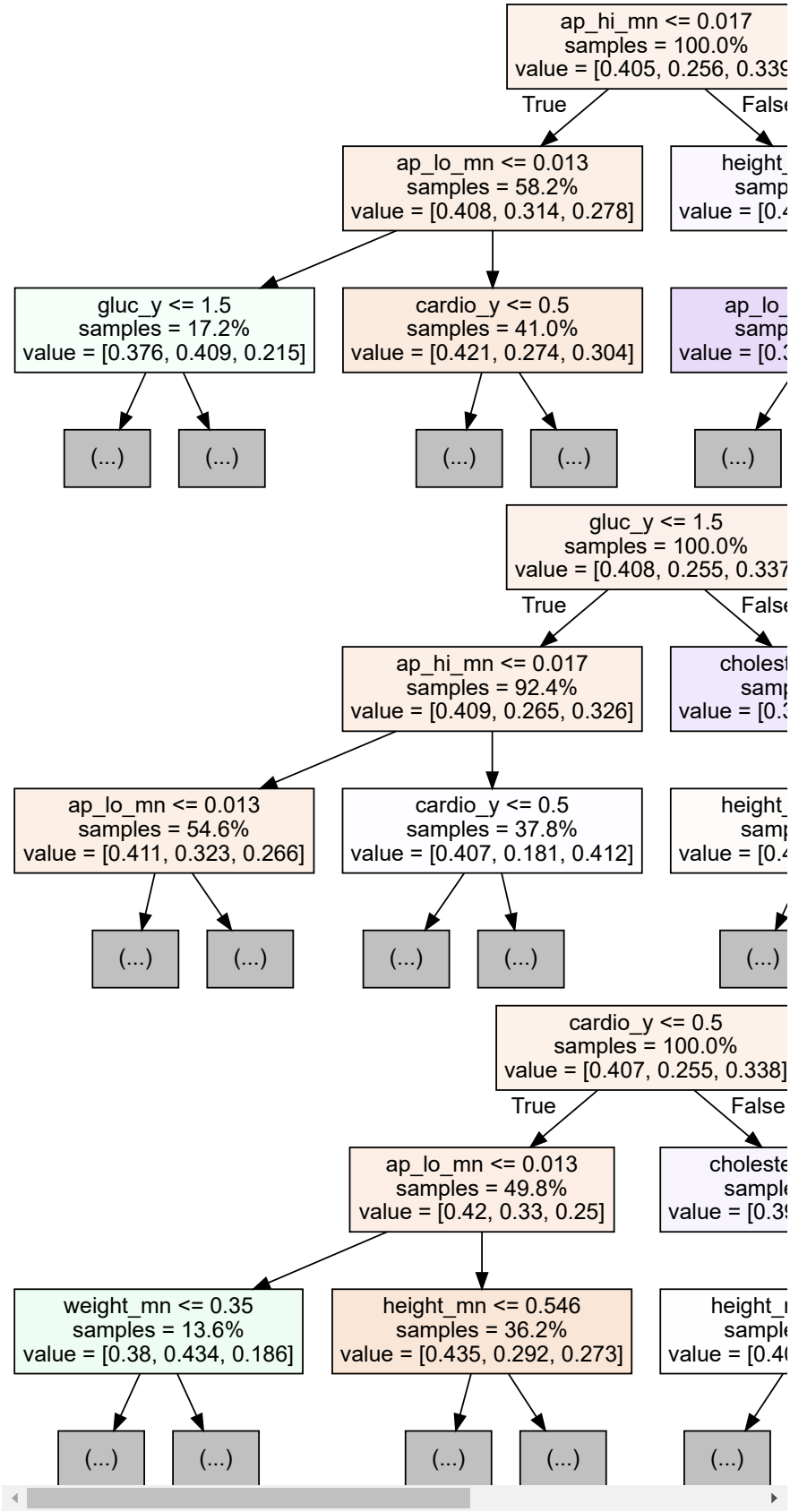

```
# Assuming rf is your trained Random Forest classifier
for i in range(3):
    tree = rf_classifier.estimators_[i]  # Assuming rf_classifier is your trained Random Forest model
    dot_data = export_graphviz(tree,
                               feature_names=train_cardio_inputs.columns,
                               filled=True,
                               max_depth=2,
                               impurity=False,
                               proportion=True)
    graph = graphviz.Source(dot_data)
    display(graph)
```




```
# Perform cross-validation with 5 folds
cv_scores = cross_val_score(rf_classifier, train_cardio_inputs, train_cardio_outputs['cluster_number_x'], cv=5)

# Print the cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Score:", cv_scores.mean())
```

```
Cross-Validation Scores: [0.40571429 0.40008929 0.40214286 0.41142857 0.40633929]
Mean CV Score: 0.40514285714285714
```

```
# Export and visualize the first three decision trees
for i in range(3):
    tree = rf_classifier.estimators_[i]
    dot_data = export_graphviz(tree,
                               feature_names=train_cardio_inputs.columns,
                               filled=True,
                               max_depth=2,
                               impurity=False,
                               proportion=True)
    graph = graphviz.Source(dot_data)
    display(graph)
```



```
# Initialize a dictionary to store tree frequencies
tree_frequency = Counter()

# Loop through the trees and count their frequency
for i in range(len(rf_classifier.estimators_)):
    tree = rf_classifier.estimators_[i]
    tree_str = export_graphviz(tree, feature_names=train_cardio_inputs.columns,
                               filled=True, max_depth=2, impurity=False, proportion=True)
    tree_frequency[tree_str] += 1

# Get the three most frequent trees
top_trees = tree_frequency.most_common(3)

# Plot the top three trees
for tree_str, frequency in top_trees:
    graph = graphviz.Source(tree_str)
    display(graph)
```

XGB BOOST


```
dtrain = xgb.DMatrix(train_cardio_inputs, label=train_cardio_outputs)
dtest = xgb.DMatrix(test_cardio_outputs, label=test_cardio_outputs)
```



```
# Define XGBoost parameters
params = {
    'objective': 'multi:softmax',  # For multi-class classification
    'num_class': len(train_cardio_outputs['cluster_number_x'].unique()),  # Number of unique classes in the output
    'max_depth': 5,
    'learning_rate': 0.1,
    'n_estimators': 1000,
    'eval_metric': 'merror'  # Use 'merror' for multiclass classification error
}

# Initialize the XGBoost classifier
xgb_classifier = xgb.XGBClassifier(**params)

# Train the classifier
xgb_classifier.fit(train_cardio_inputs,train_cardio_outputs)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='merror',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.1, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=5,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=1000,
              n_jobs=None, num_class=3, num_parallel_tree=None, ...)
```



```
# Print feature importances
feature_importances = xgb_classifier.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': train_cardio_inputs.columns, 'Importance': feature_importances})
sorted_feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
print(sorted_feature_importance_df)
```

```
          Feature  Importance
4       cardio_y    0.392435
1   cholesterol_y    0.103970
9       ap_hi_mn    0.075256
2         gluc_y    0.069598
3        smoke_y    0.068359
0       gender_y    0.060907
5         alco_y    0.054549
6       active_y    0.045865
7      height_mn    0.044905
8      weight_mn    0.042394
10      ap_lo_mn    0.041763
```



```
# Make predictions on the test set
xg_pred = xgb_classifier.predict(test_cardio_inputs)
xg_pred
```

```
array([0, 0, 1, ..., 0, 0, 0], dtype=int32)
```



```
# Print confusion matrix for testing set
print("\nTesting Set Confusion Matrix:")
print(confusion_matrix(test_cardio_outputs['cluster_number_x'], xg_pred))
print("Classification Report:")
print(classification_report(test_cardio_outputs,xg_pred))
```

```
Testing Set Confusion Matrix:
[[3205  824 1661]
 [1924  924  709]
 [2254  418 2081]]
Classification Report:
              precision    recall  f1-score   support
```