FINAL PROJECT REPORT

MSA 8150 Machine learning

HEART DISEASE PREDICTION MODEL

TEAM MEMBERS	PANTHER ID
AGASH SEKAR	002721712
RIDA FATHIMA	002695685
HARI PRIYA AVARAMPALAYAM MANOHARAN	002711275

PROBLEM STATEMENT:

Heart disease is a leading cause of death worldwide, and early detection and diagnosis of heart conditions can save lives. However, the manual analysis of patients' medical records and diagnostic tests is time-consuming and prone to error. It can be challenging for healthcare professionals to identify patterns and correlations that indicate potential heart conditions, especially in cases where there are many variables to consider.

Manual analysis of patients' medical records and diagnostic tests is prone to errors, which can lead to incorrect diagnoses or missed opportunities for early intervention. For example, healthcare professionals may miss important patterns or correlations in patients' data, leading to inaccurate diagnoses or delayed treatments. Additionally, manual analysis is time-consuming and can take healthcare professionals away from other important tasks, such as patient care and research.

This report aims to provide an overview of the dataset used for analyzing the public health burden of cardiovascular diseases (CVDs) in the United States. The dataset is compiled by the Centers for Medicare and Medicaid Services (CMS) and contains various categories of claims data for Medicare and Medicaid patients across several years.

GOAL & MOTIVATION:

Machine learning algorithms can provide a more accurate and efficient way to predict heart conditions in patients. By analyzing copious amounts of patient data, machine learning algorithms can identify patterns and correlations that healthcare professionals may miss during manual analysis. This can lead to more accurate diagnoses and earlier interventions, which can save lives.

Using machine learning algorithms for heart condition prediction can also save time for healthcare professionals, allowing them to focus on other critical tasks. Machine learning algorithms can process substantial amounts of data quickly and efficiently, freeing up healthcare professionals to focus on patient care and other important activities.

The primary objective of this project is to use the CMS dataset to identify the public health burden of CVDs and related risk factors in the United States. This includes examining trends and patterns in the data and identifying any disparities in CVD prevalence across different demographic groups, such as location, sex, and race/ethnicity. Our analysis will cover several years to provide a comprehensive understanding of the public health burden of CVDs in the United States and inform public health policies and interventions to reduce this burden.

SCOPE AND FUTURE USAGE:

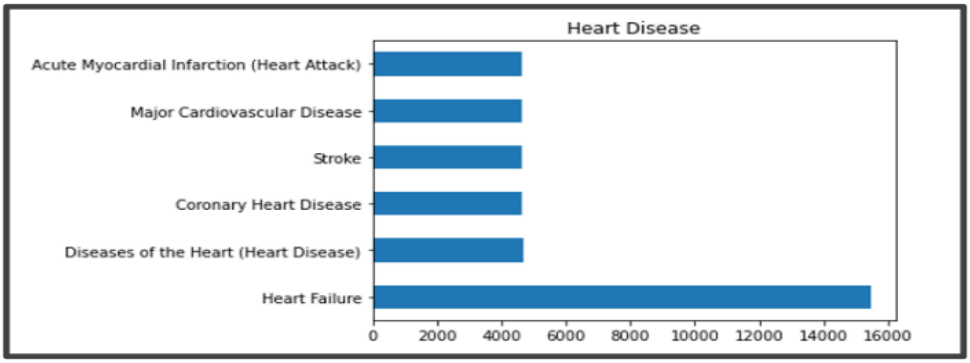
Heart Disease Prediction Models have the potential to significantly improve the accuracy of heart disease risk assessment and guide healthcare providers in developing effective preventive strategies. The scope of these models includes identifying the risk factors for heart disease, using various data sources such as medical history, lifestyle factors, and clinical tests to develop a model that can assess an individual's risk level. They can also help in implementing preventive measures such as lifestyle changes or medical interventions. Additionally, heart disease prediction models could be used to monitor patients over time and adjust their treatment plans accordingly, as well as to identify new risk factors for heart disease and further refine our understanding of the disease.
Heart disease prediction models could be used at a population level to identify areas or groups of people who are at higher risk of developing heart disease, and to develop targeted interventions to reduce their risk.

Overall, such models have the potential to play a significant role in the future of healthcare by improving heart disease prevention and management, ultimately leading to better patient outcomes.

DATASET DESCRIPTION:

The dataset used in this study was obtained from Kaggle, and it consisted of various files, such as Inpatient and Outpatient claims and Master Beneficiary Summary Files. The dataset spanned from 2004 to 2012, and it contained 39,712 rows and 36 columns. The dataset included **Multiple Labels**, such as heart failure, coronary heart disease, stroke, major cardiovascular disease, heart attack, and other types of heart diseases, with heart failure having the highest number of rows (16,467), followed by coronary heart disease (4,669), stroke (4,658), major cardiovascular disease (4,657), heart attack (4,617), and other types of heart diseases (4,644). The dataset's class imbalance was observed from the bar graph, where heart failure had the highest frequency, and the other heart disease labels had lower frequencies.

Dataset Link: https://www.kaggle.com/datasets/themrityunjaypathak/heart-disease-and-stroke-prevention



Target Feature Distribution

DATA CLEANING & PRE-PROCESSING:

Data cleaning and preprocessing is a crucial step in any data analysis or machine learning project, as it ensures that the data is accurate, consistent, and reliable. The process involves identifying and handling missing or null values, dealing with outliers and errors, and transforming the data to a usable format. The quality of the data directly affects the quality of the insights and predictions derived from it, making data cleaning, and processing a vital step in any data-driven decision-making process.

By cleaning and processing the data, we can ensure that the insights and predictions we derive from it are accurate, meaningful, and trustworthy. It also helps in avoiding biases, improving the accuracy of the models, and enabling better decision-making based on the data. Thus, documenting and reporting the steps taken for data cleaning and processing is essential to ensure transparency, reproducibility, and reliability of the analysis.

The data cleaning and preprocessing stage of this study involved multiple steps, including the identification and handling of null values, outlier detection, and normalization.

Features that have more than 60% of the values as Null, are removed.

```
df.isnull().sum()

Year                          0
LocationAbbr                  0
LocationDesc                  0
DataSource                    0
PriorityArea1                 0
PriorityArea2                 0
PriorityArea3                 0
PriorityArea4                 0
Category                      0
Topic                         0
Indicator                     0
Data_Value_Type               0
Data_Value_Unit               0
Data_Value                  529
Data_Value_Alt                0
Data_Value_Footnote_Symbol  42111
Data_Value_Footnote         42111
LowConfidenceLimit            529
HighConfidenceLimit           529
Break_Out_Category            0
Break_Out                     0
CategoryId                    0
TopicId                       0
IndicatorID                   0
Data_Value_TypeID             0
BreakOutCategoryId            0
BreakOutId                    0
LocationID                    0
GeoLocation                 820
dtype: int64
```

```
df.isnull().sum()

Year                          0
LocationAbbr                  0
LocationDesc                  0
DataSource                    0
PriorityArea1                 0
PriorityArea2                 0
PriorityArea3                 0
PriorityArea4                 0
Category                      0
Topic                         0
Indicator                     0
Data_Value_Type               0
Data_Value_Unit               0
Data_Value                  529
Data_Value_Alt                0
LowConfidenceLimit          529
HighConfidenceLimit         529
Break_Out_Category            0
Break_Out                     0
CategoryId                    0
TopicId                       0
IndicatorID                   0
Data_Value_TypeID             0
BreakOutCategoryId            0
BreakOutId                    0
LocationID                    0
dtype: int64
```

Normalization*:*

The features which have less than 60% of values as NULL are first transformed into ***Normal Distribution*** using Square Root Transformation, and then their Null values are substituted with Mean

```
df1[["Data_Value"]].describe()
```

	Data_Value
count	42111.000000
mean	3.512488
std	1.563133
min	0.632456
25%	2.121320
50%	3.271085
75%	4.647580
max	18.256506

```
df1[["HighConfidenceLimit"]].describe()
```

	HighConfidenceLimit
count	42111.000000
mean	1.717526
std	0.495398
min	0.000000
25%	1.396194
50%	1.692207
75%	2.104089
max	3.115392

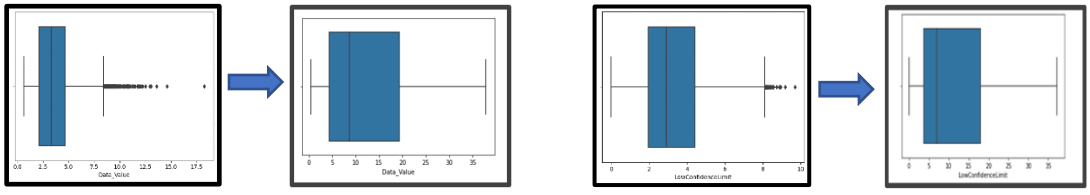
```
df1[["LowConfidenceLimit"]].describe()
```

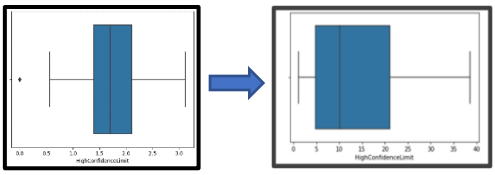
	LowConfidenceLimit
count	42111.000000
mean	3.195310
std	1.565745
min	0.000000
25%	1.949359
50%	2.863564
75%	4.427189
max	9.705668

Outlier Removal:

To Treat the outliers in ***Data_Value, LowConfidenceLimit and HighConfidenceLimit*** we perform outlier removal.Outlier analysis was performed to identify extreme values, and any data points outside the lower or upper bounds were removed.

The lower bound was calculated using LB=Q1-1.5*IQR, while the upper bound was calculated using UB=Q1+1.5*IQR, where IQR (Inter Quantile Range) was calculated as Q3-Q1.





Label Encoding:

In Label encoding is a technique where each category in a categorical feature is assigned a unique numerical value. This transformation enables machine learning algorithms to process categorical data as numerical data. We used LabelEncoder() function from scikit-learn library on the Categorical Features to perform this function.

```
lab=LabelEncoder()
df1['Topic']=lab.fit_transform(df1['Topic'])
df1['Break_Out_Category']=lab.fit_transform(df1['Break_Out_Category'])
df1['BreakOutId']=lab.fit_transform(df1['BreakOutId'])
df1['Data_Value_Unit']=lab.fit_transform(df1['Data_Value_Unit'])
df1['PriorityArea1']=lab.fit_transform(df1['PriorityArea1'])
df1['PriorityArea3']=lab.fit_transform(df1['PriorityArea3'])
```

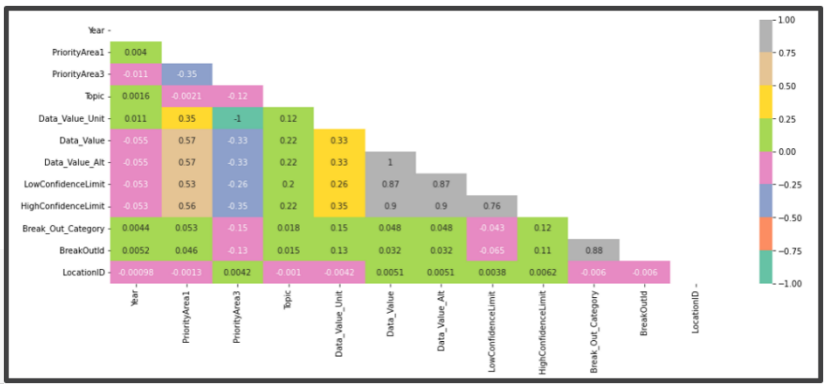
The Target Feature is also converted into Numerical from Categorical Values.

- **0** for Heart Attack
- **1** for Coronary Heart Disease
- **2** for Other Heart Diseases
- **3** for Heart Failure
- **4** for Major Cardiovascular Disease
- **5** for Stroke

Correlation:

The correlation between the columns was visualized using a correlation triangle based on the Kendall correlation coefficient formula. The grey area denoted a high correlation between the variables, dark green represented a highly negative correlation, and light green indicated no high correlation between the variables.

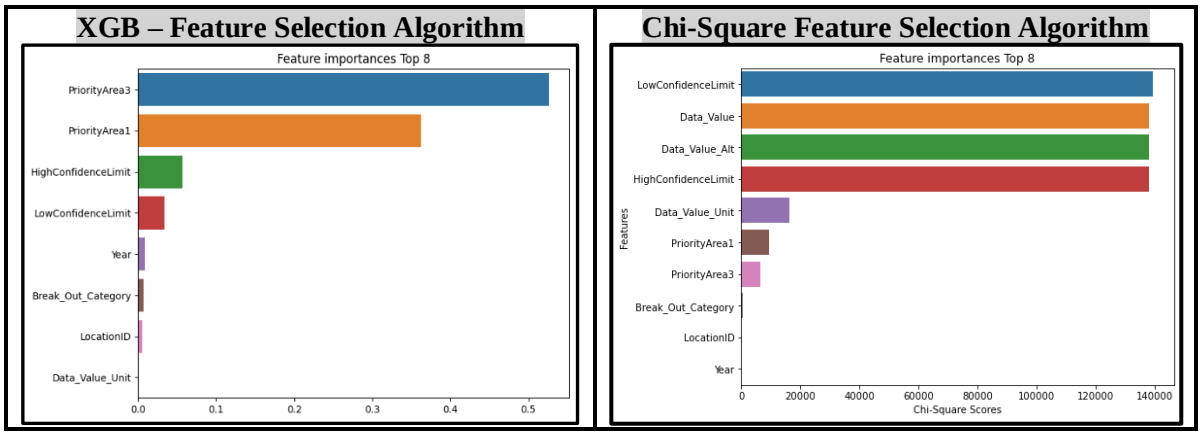
```
import seaborn as sn


plt.figure(figsize=(17,6))
corr = df1.corr(method='kendall')
my_m=np.triu(corr)
sn.heatmap(corr,mask=my_m, annot=True, cmap="Set2")
plt.show()
```



Feature Selection:

We see that our few of our features are highly correlated. This might indue the problem of Multicollinearity in our Model, due to which our Model might not give its best Performance. Hence, we performed Feature Selection, to retrieve the Top features for Analysis. Feature selection techniques, such as **XGBoost** and **chi-square**, were used to identify the top features that are most informative for the model while minimizing the effects of multicollinearity.

XGBoost algorithm ranks features based on their predictive power by assessing their frequency in splitting data across all trees, while chi-squared test selects features with high independence by comparing their observed and expected frequency distributions.

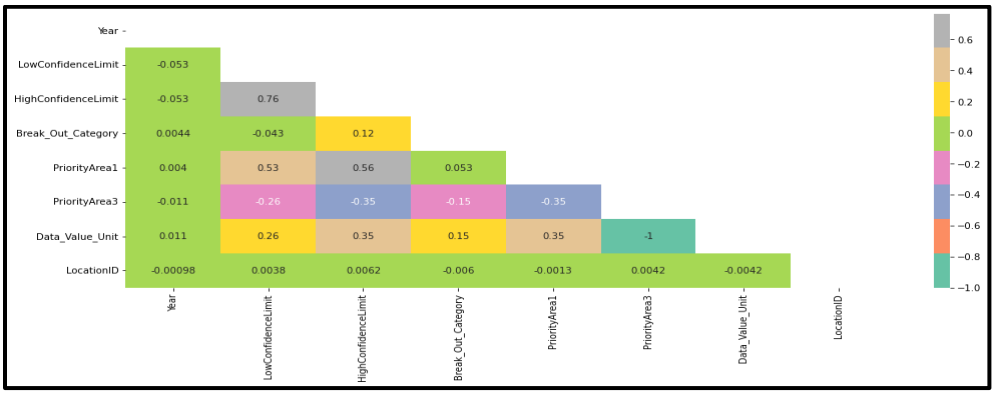


The output of the feature selection algorithms provides a visualization of the feature importance, which can be used to inform further analysis and model selection.

The Top features after Chi-Square and XGB Classification are - **'Year', 'LowConfidenceLimit', 'HighConfidenceLimit', 'Break_Out_Category', 'PriorityArea1', 'PriorityArea3', 'Data_Value_Unit', 'LocationID'.**

These features can be used for further analysis and model selection to improve the performance of the machine learning model.

Correlation After Feature Selection:



Here, we see that None of the Features are highly Correlated with each other, making the dataset suitable for analysis.

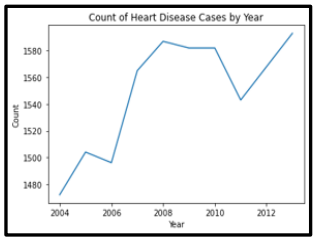
Exploratory Data Analysis:

- **Trend Analysis:**

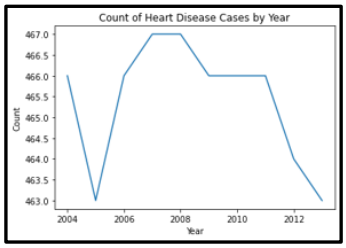
Trend analysis can be a valuable tool for studying heart disease, as it allows researchers to identify patterns and changes in disease incidence and prevalence over time. By analyzing data on the number of heart disease cases over several years, researchers can identify trends in the prevalence of different types of heart disease, such as coronary heart disease, heart failure, or stroke.

Trend analysis can also be used to identify risk factors or other factors that may be contributing to changes in the incidence of heart disease. For example, if the prevalence of heart disease is increasing over time, trend analysis can help researchers identify potential risk factors, such as changes in lifestyle behaviors or exposure to environmental factors. The below graphs show the increasing trend for each type of heart disease.

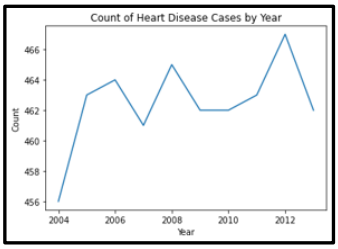
Heart Failure



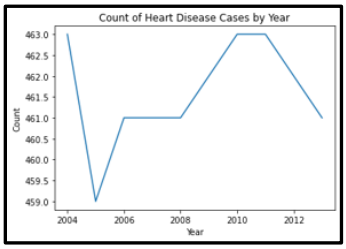
Major CV



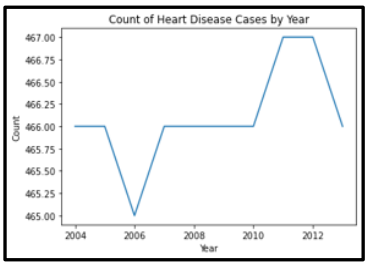
Coronary HD



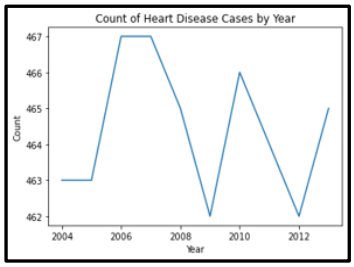
Other



Stroke



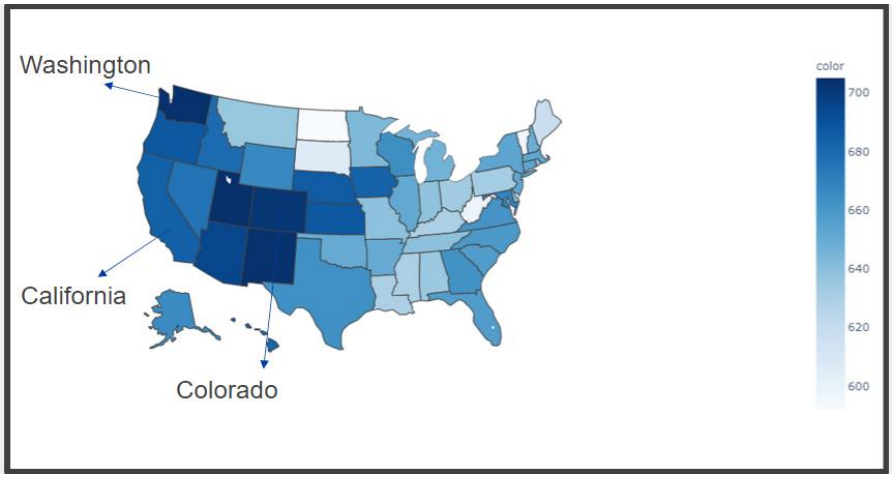
Heart Attack



- **State - Wise Analysis of Heart Disease:**

The choropleth map of the United States provides a visual representation of the regional distribution of heart disease cases, highlighting areas with a higher prevalence of the disease. From the map, it is apparent that the western region of the United States has a higher number of reported cases of heart disease compared to the eastern region. Additionally, specific states such as Washington, Colorado, California, and Arizona have a relatively higher number of reported cases.

By analyzing the regional distribution of heart disease cases, researchers and policymakers can identify areas where targeted interventions and prevention efforts may be necessary. For instance, if a particular region has a higher incidence of heart disease cases, interventions could be focused on addressing specific risk factors prevalent in that region. Similarly, the map can help policymakers allocate resources more effectively to regions with a higher burden of heart disease cases. Overall, the choropleth map provides valuable insights into the regional distribution of heart disease cases, facilitating the development of targeted interventions and prevention strategies to reduce the burden of this disease.



Hyperparameter Tuning:

We utilized the Grid Search technique to perform Hyperparameter tuning on Decision Tree and Random Forest algorithms. This involved defining a grid of hyperparameter combinations and training, evaluating the models for each combination using scikit-learn's GridSearchCV function.
The combination of hyperparameters that yielded the highest performance metric was selected as the best set of hyperparameters to be used in the models. This approach allowed us to optimize the performance of the models by identifying the most appropriate hyperparameters for each algorithm.

Modeling:

Classification modeling is a type of machine learning in which an algorithm is trained to predict the class or category of a given input. It is a supervised learning technique, which means that it learns to make predictions based on labeled data.

This project involves training and testing multiple machine-learning models on a provided dataset. The ***Logistic Regression, Decision Tree, Random Forest, and XGBoost classifiers*** are created and trained using the sci-kit-learn and XGBoost libraries. Furthermore, the models used in this project are all supervised learning models, where the target variable is known, and the models are trained to predict it based on the input features. These models are commonly used in various real-world applications such as fraud detection, image recognition, and sentiment analysis.

Model Selection Criteria:

- **Decision trees** are suitable for heart disease multi-label prediction because they can handle both categorical and continuous variables, are easy to interpret, and can handle complex interactions between variables. They can also handle imbalanced datasets and are relatively computationally efficient.
- **Random forests** reduce overfitting by combining multiple decision trees, and can handle interactions between variables. They are also robust to noise and missing data, and can handle high-dimensional datasets.
- **XGBoost** (Extreme Gradient Boosting) is suitable for heart disease multi-label prediction because it is a powerful ensemble learning method that combines multiple decision trees to improve prediction accuracy. It can handle both categorical and continuous variables, is computationally efficient, and has become a popular choice for many machine learning tasks due to its high performance and flexibility.

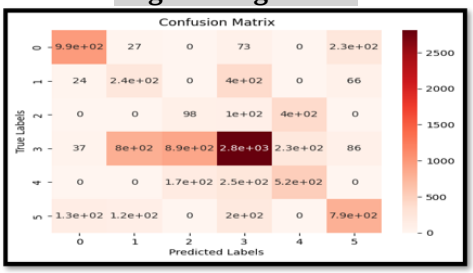
Hence, for the above reasons we have used Decision Trees, Random Forest and XGBoost Models for our Project

1. Without Sampling:

The data is divided into training and testing sets using train_test_split from sklearn – learn library. The model is trained using the training data and then used to predict the target variable for the test set (y_pred). The performance of each model is evaluated by generating a classification report and recall score, using the "micro" parameter.

The confusion matrix is computed and plotted for each model, using sklearn's confusion_matrix functions. This matrix is a useful tool for visualizing the performance of a classification model by displaying the number of true positives, false positives, true negatives, and false negatives.

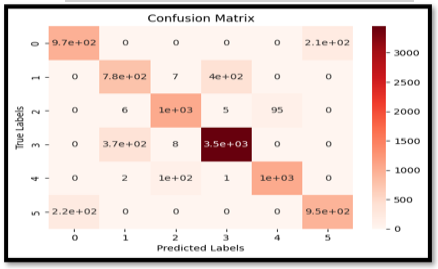
Logistic Regression



	precision	recall	f1-score	support
0	0.84	0.75	0.79	1320
1	0.20	0.33	0.25	722
2	0.08	0.16	0.11	601
3	0.73	0.58	0.65	4851
4	0.45	0.56	0.50	939
5	0.67	0.64	0.66	1240
accuracy			0.56	9673
macro avg	0.50	0.50	0.49	9673
weighted avg	0.63	0.56	0.59	9673

Recall Score is: 0.563010441434922

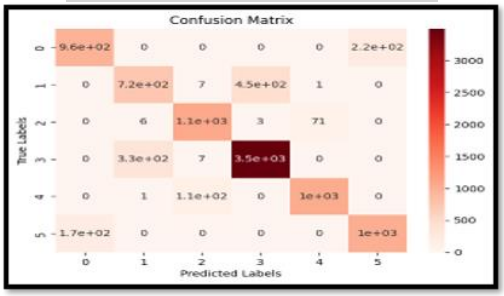
Decision Tree Classification



	precision	recall	f1-score	support
0	0.82	0.81	0.82	1199
1	0.65	0.67	0.66	1151
2	0.91	0.90	0.90	1167
3	0.90	0.89	0.90	3859
4	0.91	0.92	0.91	1140
5	0.81	0.82	0.81	1157
accuracy			0.85	9673
macro avg	0.83	0.84	0.84	9673
weighted avg	0.85	0.85	0.85	9673

Recall Score is: 0.8524759640235707

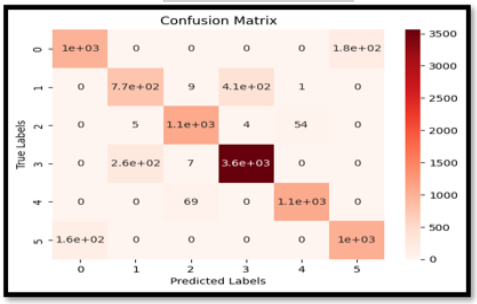
Random Forest Classification



	precision	recall	f1-score	support
0	0.82	0.85	0.83	1137
1	0.61	0.68	0.64	1060
2	0.93	0.89	0.91	1204
3	0.91	0.88	0.90	3946
4	0.90	0.93	0.92	1107
5	0.85	0.82	0.84	1219
accuracy			0.86	9673
macro avg	0.84	0.84	0.84	9673
weighted avg	0.86	0.86	0.86	9673

Recall Score is: 0.8569213274061822

XGB Classifier



	precision	recall	f1-score	support
0	0.84	0.87	0.85	1151
1	0.65	0.75	0.69	1031
2	0.95	0.93	0.94	1178
3	0.93	0.90	0.91	3972
4	0.94	0.95	0.95	1136
5	0.87	0.85	0.86	1205
accuracy			0.88	9673
macro avg	0.86	0.87	0.87	9673
weighted avg	0.89	0.88	0.88	9673

Recall Score is: 0.8809056135635274

2. UNDER SAMPLING:

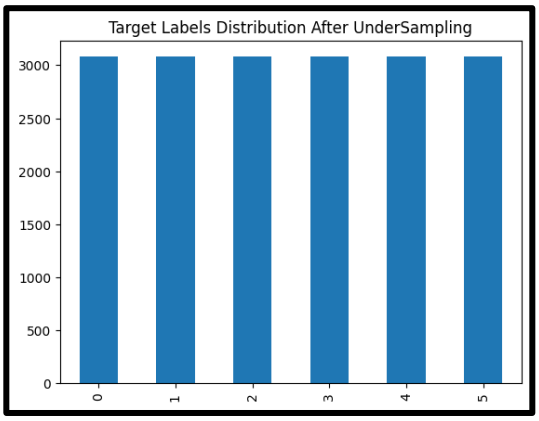
This project deals with an imbalanced dataset, where one class has significantly fewer samples than the other. To address this class imbalance, we performed Under Sampling with the help of RandomUnderSampler from the "imblearn" library. It resampled the data by randomly selecting examples from the majority class until the desired balance is achieved. The resulting resampled data is stored in X_resampled_under and y_resampled_under.

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)

# Instantiate RandomUnderSampler
rus = RandomUnderSampler(random_state=42)

# Apply RandomUnderSampler to the dataset
X_resampled_under, y_resampled_under = rus.fit_resample(X_train, y_train)



# Verify the shape of the resampled data
print("Original shape:", x.shape, y.shape)
print("Resampled shape:", X_resampled_under.shape, y_resampled_under.shape)

Original shape: (38692, 8) (38692,)
Resampled shape: (18474, 8) (18474,)
```

The shapes of the original and resampled data are:

Original Dataset Shape: (38692, 8)

Dataset Shape: (18474, 8)



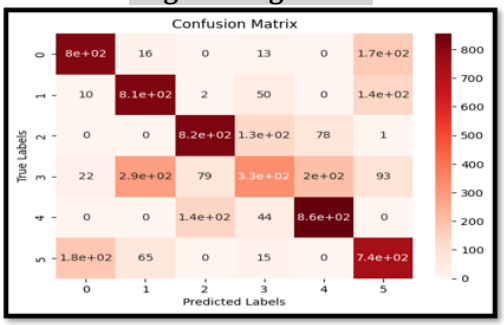
The target variable has 6 classes:

- **0** for Heart Attack
- **1** for Coronary Heart Disease
- **2** for Other Heart Diseases
- **3** for Heart Failure
- **4** for Major Cardiovascular Disease
- **5** for Stroke

Using the under-sampled data, we implemented four distinct classification models - ***Logistic Regression, Decision Tree, Random Forest, and XGBoost*** - by leveraging the sci-kit-learn and XGBoost libraries.

Training and testing sets were created for each model, and the model was trained using the training data and then used to predict the target variable for the test set. The performance of each model was evaluated using a classification report and recall score, along with a confusion matrix as shown below:

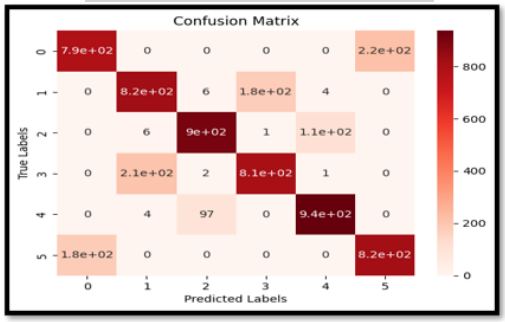
Logistic Regression



	precision	recall	f1-score	support
0	0.80	0.79	0.80	1016
1	0.80	0.69	0.74	1183
2	0.80	0.79	0.79	1036
3	0.33	0.57	0.42	583
4	0.82	0.75	0.79	1134
5	0.74	0.65	0.69	1145
accuracy			0.72	6097
macro avg	0.72	0.71	0.70	6097
weighted avg	0.75	0.72	0.73	6097

Recall Score is: 0.7160898802689848

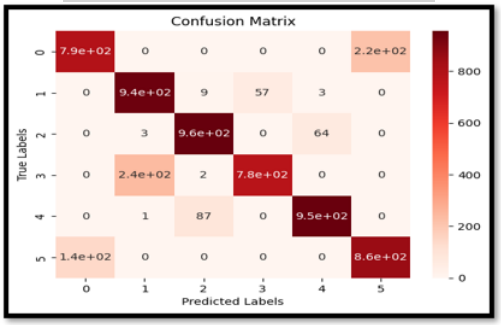
Decision Tree Classification



	precision	recall	f1-score	support
0	0.78	0.81	0.80	970
1	0.82	0.79	0.80	1041
2	0.88	0.90	0.89	1009
3	0.79	0.82	0.81	986
4	0.90	0.89	0.90	1053
5	0.82	0.79	0.80	1038
accuracy			0.83	6097
macro avg	0.83	0.83	0.83	6097
weighted avg	0.83	0.83	0.83	6097

Recall Score is: 0.8333606691815647

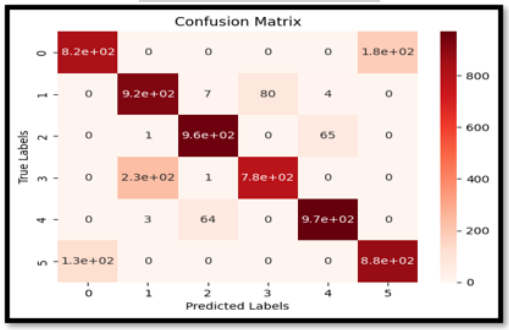
Random Forest Classification



	precision	recall	f1-score	support
0	0.80	0.85	0.82	941
1	0.93	0.79	0.86	1187
2	0.93	0.91	0.92	1050
3	0.76	0.93	0.84	830
4	0.92	0.93	0.92	1022
5	0.86	0.81	0.83	1067
accuracy			0.87	6097
macro avg	0.87	0.87	0.87	6097
weighted avg	0.87	0.87	0.87	6097

Recall Score is: 0.8661636870592094

XGB Classification



	precision	recall	f1-score	support
0	0.82	0.87	0.84	950
1	0.91	0.79	0.85	1158
2	0.94	0.93	0.93	1029
3	0.77	0.91	0.83	863
4	0.94	0.93	0.93	1039
5	0.87	0.83	0.85	1058
accuracy			0.87	6097
macro avg	0.87	0.88	0.87	6097
weighted avg	0.88	0.87	0.87	6097

Recall Score is: 0.8740364113498442

Hyperparameter Tuning: Hyperparameter tuning is performed on *Decision Tree* and *Random Forest Classification* algorithms. The grid Search technique is implemented to find the optimal combination of hyperparameters that maximizes the model's metrics.

Decision Tree Classification:

```
param_grid = {
    'max_depth': [2, 4, 6, 8, 10],
    'min_samples_leaf': [4, 8],
    'min_samples_split': [2, 4, 8, 16]
}

# Grid search for best hyperparameters
grid_search_dtc = GridSearchCV(tre, param_grid, cv=5, n_jobs=-1)
grid_search_dtc.fit(X_train1, y_train1)

# Print the best hyperparameters and accuracy score
print("Best hyperparameters for Decision Tree: ", grid_search_dtc.best_params_)
```

```
Best hyperparameters for Decision Tree:  {'max_depth': 6, 'min_samples_leaf': 8, 'min_samples_split': 2}
```

We specified a range of values for three hyperparameters in the "param_grid" dictionary:

- **max_depth** = [2,4,6,8,10]
- **min_samples_leaf**= [4,8]
- **min_samples_split** = [2,4,8,16]

The **max_depth** hyperparameter controls the maximum depth of the decision tree, while **min_samples_leaf** and **min_samples_split** specify the minimum number of samples required to be considered as a leaf node or to split an internal node, respectively.

The Best Parameters obtained are: 'max_depth': 6, 'min_samples_leaf': 8, 'min_samples_split': 2

Random Forest Classification:

```
#create a Random Forest model
rf_under = RandomForestClassifier(random_state=42)

#define the hyperparameters and their ranges for GridSearchCV
param_grid = {
   'n_estimators': [100, 200, 500],
   'max_depth': [None, 10, 20],
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [2, 4],
   'max_features': ['sqrt', 'log2', 0.5]
}

#create a GridSearchCV object with the Random Forest model and hyperparameters
grid_search = GridSearchCV(rf_under, param_grid=param_grid, cv=5)

#fit the GridSearchCV object to the data
grid_search.fit(X_train1, y_train1)

#print the best hyperparameters and the corresponding score
print("Best hyperparameters: ", grid_search.best_params_)
#print("Best score: ", grid_search.best_score_)
```

```
Best hyperparameters:  {'max_depth': None, 'max_features': 0.5, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100}
```

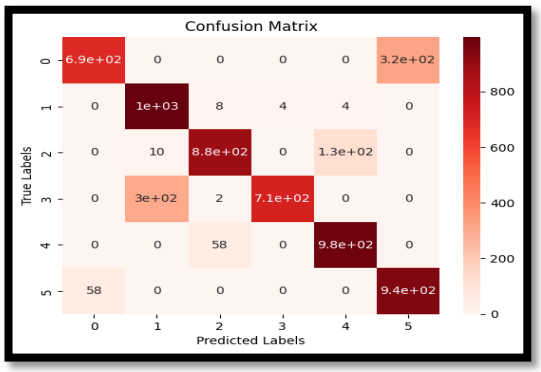
We specified a range of values for following hyperparameters in the "param_grid" dictionary:

- **max_features** = ['sqrt','log2',0.5]
- **min_samples_leaf** = [2,4]
- **n_estimators** = [100,200,500]
- **min_samples_split** = [2,5,10]

Here, **n_estimators** refers to the number of trees that are trained and used in the ensembling technique.

The Best Parameters obtained are: 'max_features: 0.5, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100

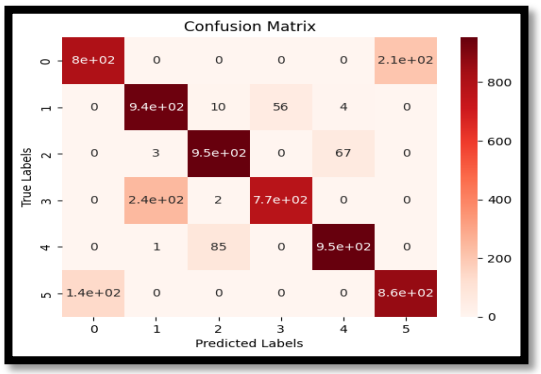
Decision Tree Classification



	precision	recall	f1-score	support
0	0.68	0.92	0.78	745
1	0.98	0.76	0.86	1310
2	0.86	0.93	0.90	952
3	0.70	0.99	0.82	715
4	0.94	0.88	0.91	1112
5	0.94	0.75	0.83	1263
accuracy			0.85	6097
macro avg	0.85	0.87	0.85	6097
weighted avg	0.88	0.85	0.85	6097

Recall Score is: 0.8527144497293752

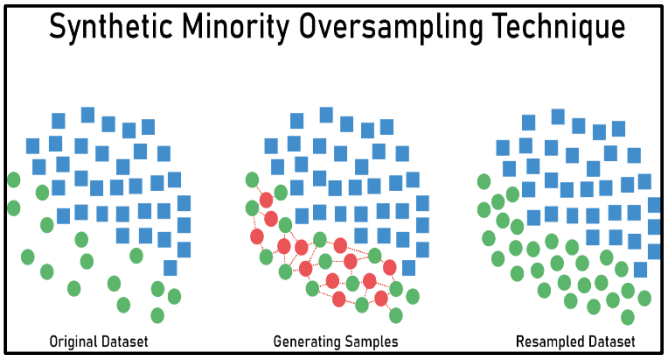
Random Forest Classification



	precision	recall	f1-score	support
0	0.81	0.86	0.83	945
1	0.94	0.79	0.86	1200
2	0.94	0.91	0.93	1048
3	0.76	0.94	0.84	819
4	0.92	0.94	0.93	1022
5	0.87	0.82	0.84	1063
accuracy			0.87	6097
macro avg	0.87	0.88	0.87	6097
weighted avg	0.88	0.87	0.87	6097

The Recall Score is : 0.8725602755453502

3. SMOTE - (Synthetic Minority Over-sampling Technique):



The SMOTE algorithm works by creating synthetic examples of the minority class by interpolating between existing examples of that class. Synthetic examples are created by selecting pairs of similar examples from the minority class and generating new examples along the line connecting them. This helps to balance the classes in the dataset and can improve the performance of machine learning models trained on imbalanced data.

We performed SMOTE with the help of SMOTE from “imblearn” library.

```
# Apply SMOTE sampling
smote = SMOTE(random_state=42)
X_resampled_smote, y_resampled_smote = smote.fit_resample(x, y)

# Verify the shape of the resampled data
print("Original shape:", x.shape, y.shape)
print("Resampled shape:", X_resampled_smote.shape, y_resampled_smote.shape)
```

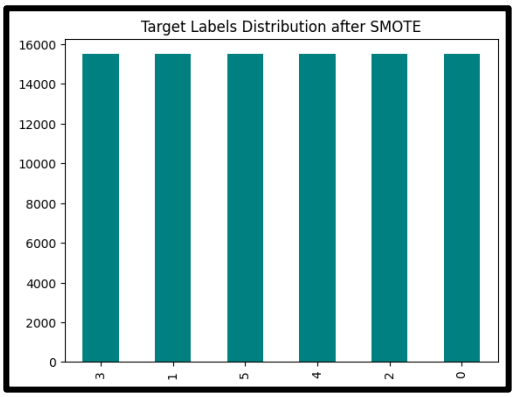
```
Original shape: (38692, 8) (38692,)
Resampled shape: (92952, 8) (92952,)
```

The shapes of the original and resampled data are:

Original Dataset Shape: (38692, 8)

Dataset Shape: (92952, 8)

To visualize the distribution of the target variable, a bar plot is generated using the Matplotlib library. This plot shows the count of samples in each class after SMOTE.

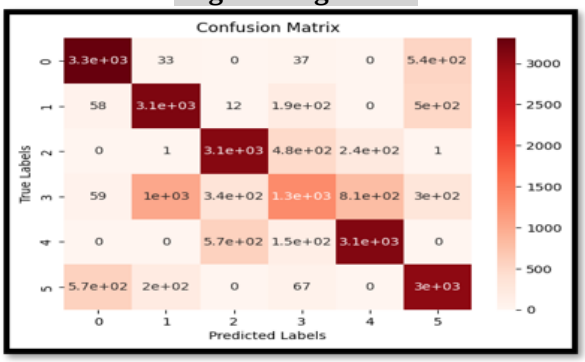


The target variable has 6 classes:

- **0** for Heart Attack
- **1** for Coronary Heart Disease
- **2** for Other Heart Diseases
- **3** for Heart Failure
- **4** for Major Cardiovascular Disease
- **5** for Stroke

Training and testing sets were created for each model, and the model was trained using the training data and then used to predict the target variable for the test set. The performance of each model was evaluated using a classification report and recall score, along with a confusion matrix as shown below:

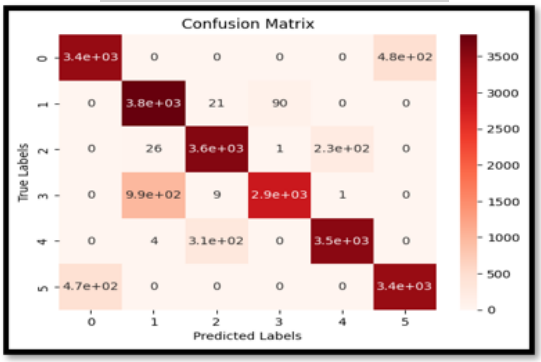
Logistic Regression



	precision	recall	f1-score	support
0	0.84	0.83	0.84	3999
1	0.80	0.71	0.76	4417
2	0.81	0.77	0.79	4020
3	0.34	0.58	0.43	2238
4	0.81	0.75	0.78	4174
5	0.78	0.69	0.74	4390
accuracy			0.73	23238
macro avg	0.73	0.72	0.72	23238
weighted avg	0.76	0.73	0.74	23238

The Recall Score is: 0.7226110527178307

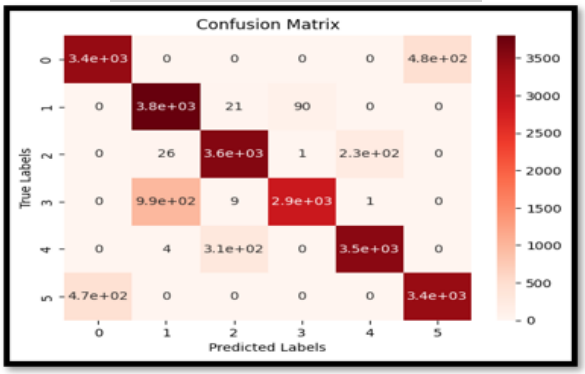
Decision Tree Classification



	precision	recall	f1-score	support
0	0.88	0.88	0.88	3906
1	0.97	0.79	0.87	4823
2	0.93	0.91	0.92	3902
3	0.74	0.97	0.84	2947
4	0.92	0.94	0.93	3761
5	0.88	0.88	0.88	3899
accuracy			0.89	23238
macro avg	0.89	0.89	0.89	23238
weighted avg	0.90	0.89	0.89	23238

The Recall Score is: 0.8867372407263964

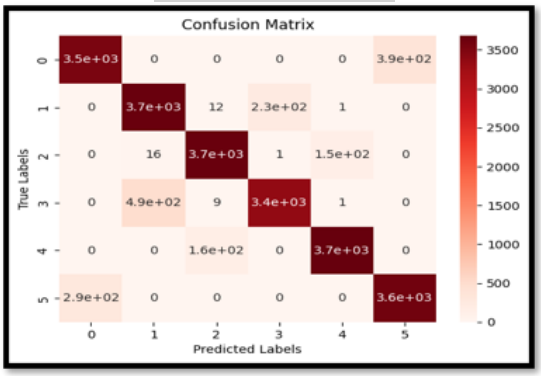
Random Forest Classification



	precision	recall	f1-score	support
0	0.89	0.89	0.89	3837
1	0.88	0.87	0.87	3927
2	0.94	0.95	0.95	3918
3	0.87	0.88	0.88	3818
4	0.96	0.95	0.95	3884
5	0.89	0.89	0.89	3854
accuracy			0.91	23238
macro avg	0.91	0.91	0.91	23238
weighted avg	0.91	0.91	0.91	23238

The Recall Score is: 0.9058729559175104

XGB Classification



	precision	recall	f1-score	support
0	0.90	0.92	0.91	3821
1	0.94	0.88	0.91	4172
2	0.96	0.95	0.95	3838
3	0.87	0.93	0.90	3592
4	0.96	0.96	0.96	3831
5	0.92	0.90	0.91	3984
accuracy			0.92	23238
macro avg	0.92	0.93	0.92	23238
weighted avg	0.93	0.92	0.92	23238

Recall Score is: 0.9253177941104802

Hyper Parameter Tuning: Hyper parameter tuning is performed on *Decision Tree and Random Forest Classification* algorithms. Grid Search technique is implemented to find the optimal combination of hyperparameters that maximizes the model's metrics.

Decision Tree Classification:

```
param_grid = {
    'max_depth': [2, 4, 6, 8, 10],
    'min_samples_leaf': [2, 4, 8],
    'min_samples_split': [2, 4, 8, 16]
}

# Grid search for best hyperparameters
grid_search_dtc = GridSearchCV(dt_over, param_grid, cv=3, n_jobs=-1)
grid_search_dtc.fit(X_train_over, y_train_over)

# Print the best hyperparameters and accuracy score
print("Best hyperparameters for Decision Tree: ", grid_search_dtc.best_params_)
```

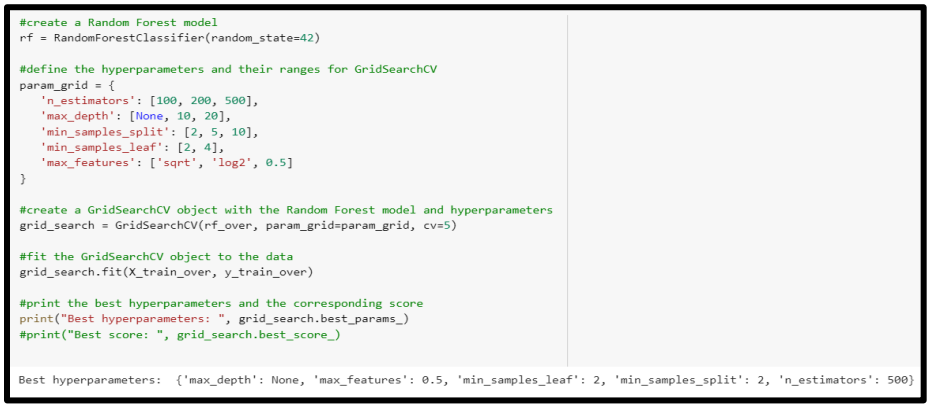
```
Best hyperparameters for Decision Tree:  {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2}
```

We specified a range of values for three hyperparameters in the "param_grid" dictionary:

- **max_depth** = [2,4,6,8,10]
- **min_samples_leaf**= [2,4,8]
- **min_samples_split** = [2,4,8,16]

The Best Parameters obtained are: 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2

Random Forest Classification:



```
#create a Random Forest model
rf = RandomForestClassifier(random_state=42)

#define the hyperparameters and their ranges for GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [2, 4],
    'max_features': ['sqrt', 'log2', 0.5]
}

#create a GridSearchCV object with the Random Forest model and hyperparameters
grid_search = GridSearchCV(rf_over, param_grid=param_grid, cv=5)

#fit the GridSearchCV object to the data
grid_search.fit(X_train_over, y_train_over)

#print the best hyperparameters and the corresponding score
print("Best hyperparameters: ", grid_search.best_params_)
#print("Best score: ", grid_search.best_score_)
```

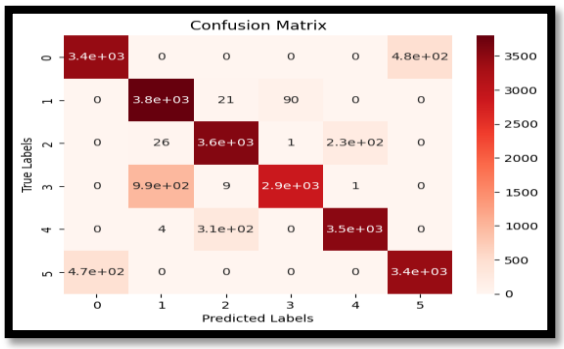
```
Best hyperparameters:  {'max_depth': None, 'max_features': 0.5, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 500}
```

We specified a range of values for three hyperparameters in the "param_grid" dictionary:

- **max_features** = ['sqrt','log2',0.5]
- **min_samples_leaf**= [2, 4]
- **n_estimators** = [100, 200, 500]
- **min_samples_split** = [2,5,10]

The Best Parameter obtained are: 'max_features: 0.5, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 500

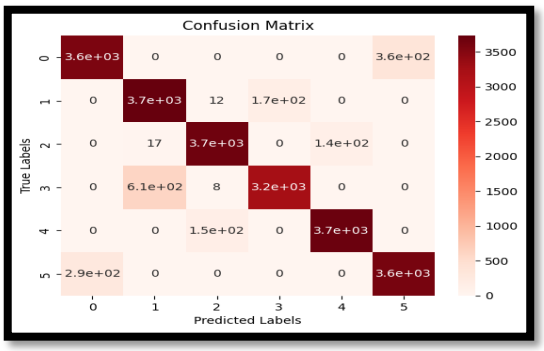
Decision Tree Classification



	precision	recall	f1-score	support
0	0.88	0.88	0.88	3906
1	0.97	0.79	0.87	4823
2	0.93	0.91	0.92	3902
3	0.74	0.97	0.84	2947
4	0.92	0.94	0.93	3761
5	0.88	0.88	0.88	3899
accuracy			0.89	23238
macro avg	0.89	0.89	0.89	23238
weighted avg	0.90	0.89	0.89	23238

The Recall Score is: 0.8943016197038078

Random Forest Classification



	precision	recall	f1-score	support
0	0.91	0.93	0.92	3847
1	0.95	0.86	0.90	4354
2	0.96	0.95	0.96	3833
3	0.84	0.95	0.89	3413
4	0.96	0.96	0.96	3833
5	0.93	0.91	0.92	3958
accuracy			0.92	23238
macro avg	0.92	0.93	0.92	23238
weighted avg	0.93	0.92	0.92	23238

The Recall Score: 0.9247783802392633

Results:

MODELS	NO SAMPLING	UNDER SAMPLING	SMOTE
LOGISTIC REGRESSION RECALL	56%	71%	72.26%
DECISION TREE RECALL	85.24%	83.33%	88.67%
DECISION TREE RECALL (Tuning)	-	85.27%	89.43%
RANDOM FOREST RECALL	85.69%	86.61%	90.58%
RANDOM FOREST RECALL (Tuning)	-	87.25%	92.47%
XGBOOST RECALL	88.09%	87.40%	**92.53%**

Conclusion:

In the case of health projects, it is crucial to reduce the number of False Negatives to ensure that patients receive the appropriate treatment for their condition. Therefore, among all the performance metrics, we computed the **Recall** score to select the model that gives the least number of false negatives. Having a higher Recall score allows us to identify more true positives, which is important for accurate diagnosis and treatment.

Recall = TP/(TP+FN)
TP = True Positive
FN = False Negatives

Therefore, by looking at the results, **XGB Classification Algorithm** performed the best after **SMOTE Sampling Technique**, giving a **Recall Score** of **92.53%**

References:

- https://www.kaggle.com/datasets/themrityunjaypathak/heart-disease-and-stroke-*prevention*
- https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/
- https://medium.com/analytics-vidhya/bank-data-smote-b5cb01a5e0a2