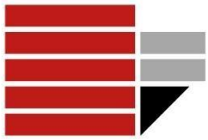
University of Calabria

Department of Mathematics and Computer Science



Machine Learning project

Author

Teame Gidena Hiben [251851]

Respective Faculty

Prof.P. Rullo

Angelica Liguori

June 05, 2024

Table of Contents

LIST OF FIGURES

List of Tables

1 Business Understanding

CMS compiles claims data for Medicare and Medicaid patients across a variety of categories and years. This includes Inpatient and Outpatient claims, Master Beneficiary Summary Files, and many other files. Indicators from this data source have been computed by personnel in CDC's Division for Heart Disease and Stroke Prevention (DHDSP). The system is designed to integrate multiple indicators from many data sources to provide a comprehensive picture of the public health burden of CVDs and associated risk factors in the United States. The data are organized by location (national and state) and indicator. The data can be plotted as trends and stratified by sex and race/ethnicity.

1.1 Background

Kaggle is a data science competition platform and online community of data scientists and machine learning practitioners under Google LLC. Kaggle enables users to find and publish datasets, explore, and build models in a web-based data science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

1.2 Business Objectives

The objective is to develop an automated classification model that accurately categorizes various cardiovascular diseases by analysing data related to the public health burden in United States.

1.3 Data Mining Objectives

The objective is to train a model capable of categorizing cardiovascular diseases in to one of the 6 categories (Heart Failure, Coronary Heart Disease, Stroke, Major Cardiovascular Disease, Diseases of the Heart (heart disease), Acute Myocardial Infarction (Heart Attack)).

1.4 Resource and Requirement

This task involves the resources and requirements analysis for the project.

1.4.1 Resources

List of resources available to the project:

Personnel: three students of machine learning.

• Data: fixed extracts.

• Computing resources: laptops; 6

1.4.2 Requirements

Programming Language

- Python: is an interpreted, high-level language with dynamic semantics

  – Modules (packages)

- NumPy: is the fundamental package for scientific computing.
- SciPy library provides user-friendly and efficient numerical routines such as routines for numerical integration and optimization.
- Pandas: is a set of structures and data analysis tools.
- Scikit-Learn library is a repository rich in simple and efficient tools for data mining and data analysis.
- Matplotlib: is a plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. – Sharing Server
- Jupiter notebook is an open-source web application that allows **us** to create and share documents that contain live code, equations, visualization, and narrative text.

2 Data Understanding

The first step of this project is to understand the data. In this section we addressed the activities of data understanding. We started with data collection, selection up to balance.

2.1 Data Collection

The first phase of our project focuses on thoroughly understanding the dataset. This involves several key activities, starting with data collection and selection, and moving on to the critical task of balancing the dataset. These data understanding activities are essential for setting the groundwork for the later stages of our data mining project. With a deep understanding of the dataset, we can effectively move forward to the next steps, including data preprocessing, feature engineering, and finally, model development.

As per the instructions outlined in our project guidelines, we meticulously obtained our dataset from the specified source detailed in the project outline. Our data was sourced directly from the reputable Heart Disease and Stroke Prevention dataset in Kaggle, https://www.kaggle.com/datasets/themrityunjaypathak/heart-disease-andstroke-prevention.

2.2 Data Description

The dataset has the following characteristics:

- Multivariate ○ 42640 rows representing the number of records. ○ 29 columns representing the number of attributes in the dataset.
- 1236560 total data

Out of the 29 attributes, we initially have the following division of datatypes:

- 22 categorical attributes (42640 rows × 22 columns)
- 6 numeric attributes (42640 rows × 6 columns)
- ordinal attribute(year)

2.3 Attribute Description

As we mentioned earlier, we have 29 attributes, and each attribute holds either numerical or categorical information. Table 2.1 describes all attributes and their description.

Attribute	Non-Null Count	Dtype	Description
Year	42640	category	The year the data was collected.
LocationAbbr	42640	category	Abbreviation of the location.
LocationDesc	42640	category	Full description of the location.
DataSource	42640	category	The source of the data (e.g., Survey, Registry).
PriorityArea1	9360	category	Priority areas related to the data.
PriorityArea2	0	category	Priority areas related to the data.
PriorityArea3	14560	category	Priority areas related to the data.
PriorityArea4	0	category	Priority areas related to the data.
Category	42640	category	The general category of the data (e.g., Health).
Topic	42640	category	Specific topic within the category (Target class).
Indicator	42640	category	The specific indicator being measured
Data_Value_Type	42640	category	The type of data value (e.g., Rate, Percentage).
Data_Value_Unit	42640	category	The unit of the data value (e.g., Per 100,000).
Data_Value	42111	float64	The value of the data.
Data_Value_Alt	42111	float64	An alternative value for the data.
Data_Value_Footnote_Symbol	529	category	Footnote symbol for additional information.
Data_Value_Footnote	529	category	Footnote text providing additional information.
LowConfidenceLimit	42111	float64	The lower bound of the confidence interval.
HighConfidenceLimit	42111	float64	The upper bound of the confidence interval.
Break_Out_Category	42640	category	The category for breaking out data (e.g., Gender).
Break_Out	42640	category	The specific break out within the category (e.g M, F).
CategoryId	42640	category	Identifier for the category.
TopicId	42640	category	Identifier for the topic.
IndicatorID	42640	category	Identifier for the indicator.
Data_Value_TypeID	42640	category	Identifier for the data value type.
BreakOutCategoryId	42640	category	Identifier for the breakout category.
BreakoutID	42640	category	Identifier for the breakout.
LocationID	42640	category	Identifier for the location.
GeoLocation	41820	category	Geographic location in POINT (longitude latitude)

Table 2.1 Attribute description

Upon examining the attributes, we identified attributes with incorrect data types. Subsequently, we proceed to convert these attributes to the appropriate data types. Specifically, attributes with an object data type are converted to categorical attributes to facilitate more efficient data processing and analysis. Additionally, the attribute representing the year is recognized as ordinal data, and appropriate conversion is applied to reflect this characteristic accurately. This ensures that the data is structured in a manner conducive to meaningful analysis and model training.

2.4 Data Exploration

2.4.1 Get General statistics.

The dataset consists of various statistical measures for different variables, providing insights into the distribution and spread of the data. *Figure 2.1* below summarizes the statistics for each numeric feature:

	count	mean	std	min	25%	50%	75%	max
PriorityArea2	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
PriorityArea4	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Data_Value	42111.0	14.780896	13.286487	0.4	4.5	10.7	21.6	333.3
Data_Value_Alt	42640.0	14.572709	13.333826	-2.0	4.4	10.4	21.5	333.3
LowConfidenceLimit	42111.0	12.661509	11.453615	0.0	3.8	8.2	19.6	94.2
HighConfidenceLimit	42111.0	17.074028	17.910274	1.1	5.0	12.7	23.4	710.5

Figure 2.1 General statistics.

We have dropped the attributes PriorityArea2 and PriorityArea4 from the dataset as they contain no values and do not serve any purpose in predicting our target variable. Throughout the dataset, these two attributes show a count of zero and all other statistical measures as NaN, this suggests that there were no observations or entries recorded for these areas.

2.4.2 Histogram & pair plots to the Numeric attributes

The histogram and pair plots provide valuable insights into the dataset's numerical attributes. *Figure 2.2* displays the frequency distribution of each numerical attribute, showing most values fall within a similar range and many distributions are right-skewed, indicating a deviation from normal distribution. *Figure 2.3* presents the pair plot or scatterplot matrix, which visualize the relationships between pairs of attributes and their frequencies in relation to the target class. Combining histograms and scatter plots, these visualizations offer a comprehensive overview of the dataset's distributions and correlations, providing crucial information for future data cleaning processes.

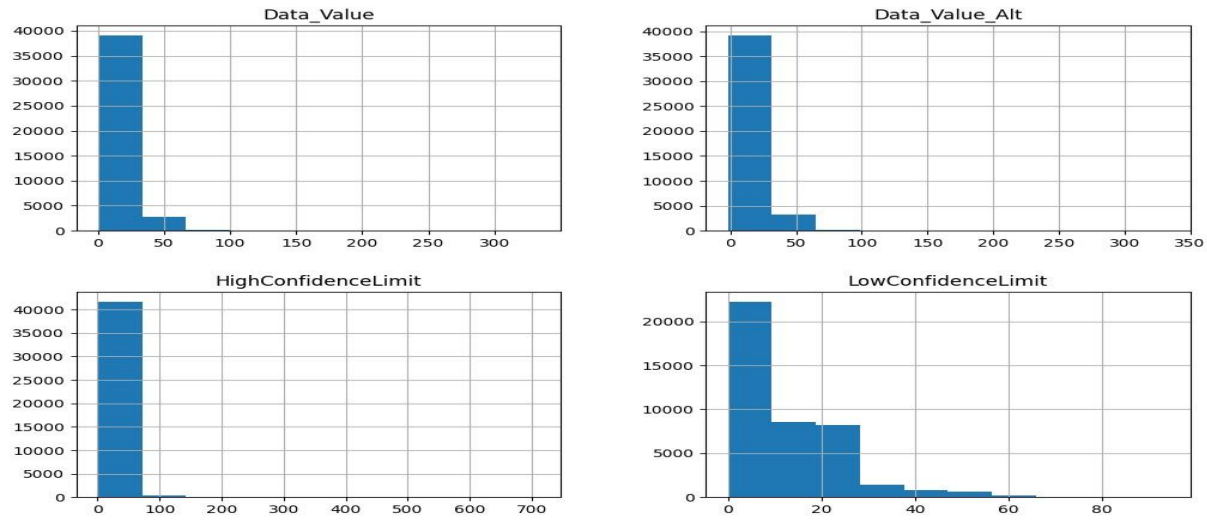


Figure 2.2 Visualizing numeric features

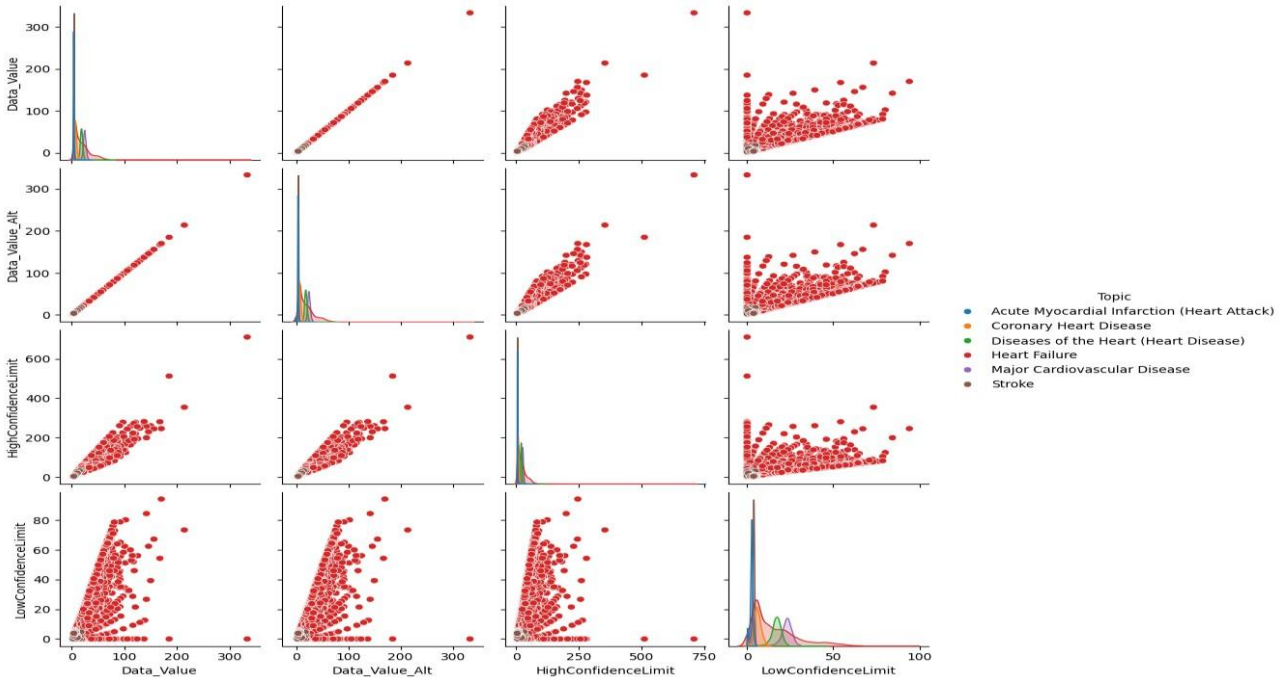


Figure 2.3 pair plot

2.4.3 bar plots to the categorical attributes

Figure 2.4 illustrate the health conditions of US residents. It is evident from the graph that the incidence of Heart Failure is significantly higher compared to other conditions. Hence, we can understand that the data is unbalanced. Heart Failure represents the majority and the other conditions forming the minority.

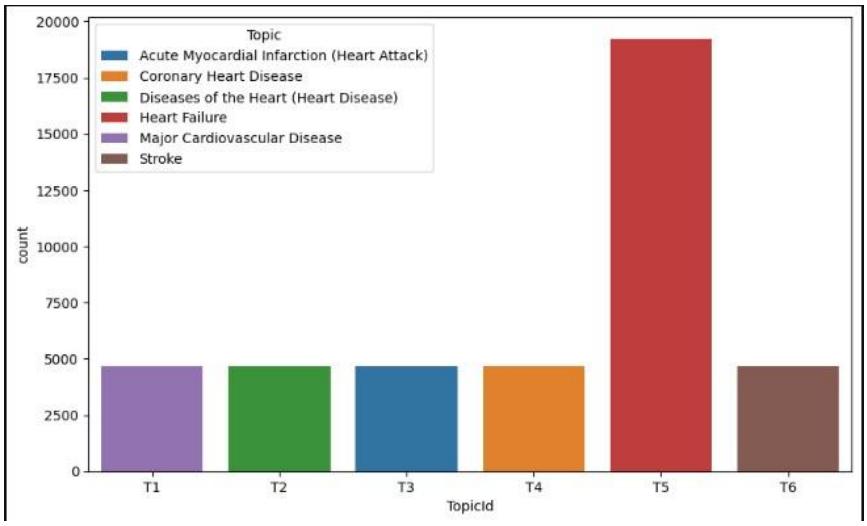


Figure 2.4 Visualizing Categorical attributes

2.4.4 Finding the outliers

The box plot shown in *Figure 2.5* visualizes numeric attributes graphically and we observe that there is an outlier present. However, since the dataset is unbalanced, it is possible that the outliers belong to the minority class. This requires careful consideration, as removing these outliers could inadvertently lead to the loss of important information from the minority class, which is already underrepresented. Therefore, it is essential to analyze these outliers in the context of the class distribution to ensure that any data cleaning steps do not negatively affect the integrity and representativeness of the dataset.

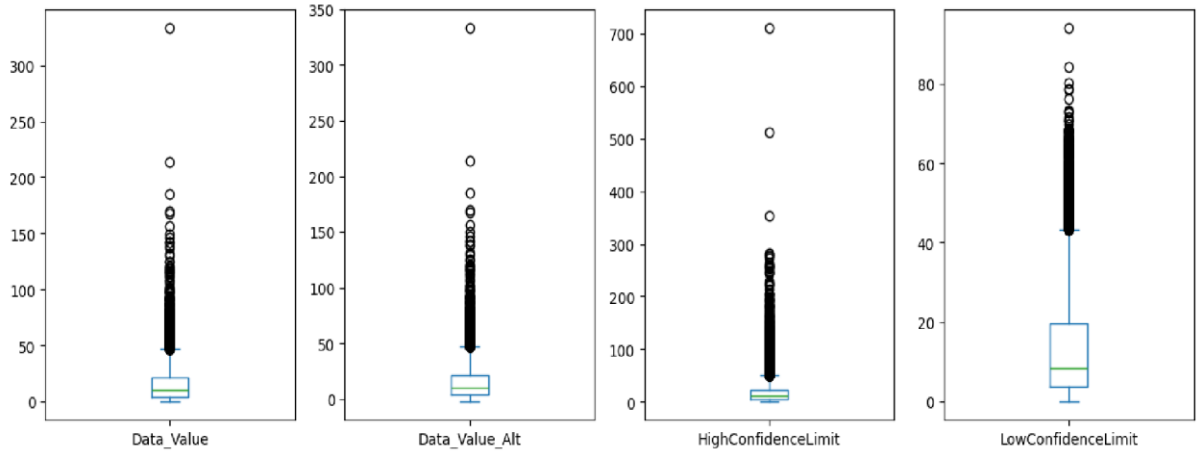


Figure 2.5 Box plot

2.4.5 Correlation between attributes

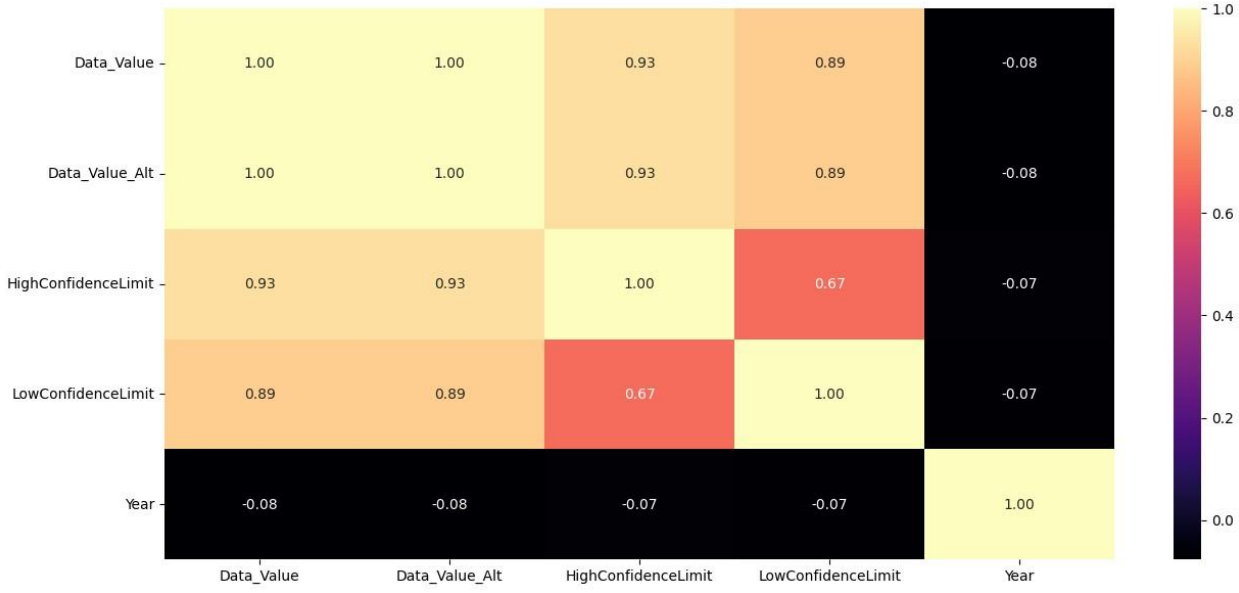


Figure 2.6 Correlation matrix

Figure 2.6 shows that the attributes are highly correlated. Specifically, Data_Value is strongly correlated with Data_Value_Alt, LowConfidenceLimit, and HighConfidenceLimit, with

correlation coefficients getting close to a threshold of 1. Therefore, it is advisable to drop the Data_Value attribute from the analysis due to its strong correlation with the other attributes. As a result, we have dropped the Data_Value attribute from the analysis.

3 Data Cleaning

After understanding the data, the next crucial stage is data cleaning. During this phase, we focused on removing missing columns, extraneous features, and duplicate values. This process includes eliminating irrelevant features, comparing and dropping location attributes with similar information, handling missing values, and avoiding contradictory examples within the dataset.
Effective data cleaning is essential for successful machine learning model training.

3.1 Drop irrelevant Features.

Dropping irrelevant Attributes is crucial for reducing noise, improving model performance, efficiency, and interpretability. Irrelevant features can introduce noise and lead to overfitting, making the model less effective. This step involves leveraging domain knowledge, performing correlation analysis, using statistical tests, applying variance thresholds, and assessing feature importance from models.

3.1.1 Attributes with low variability

Remove all the attributes with too low variability (column that exhibit the same value for all the records) and all redundant attributes. The indicator column/attribute must be eliminated because it is irrelevant to train our model, as stated in the project outline.

	count	unique	top	freq
BreakOutCategoryId	42640	4	BOC04	20800
BreakOutId	42640	9	GEN01	5200
Break_Out	42640	9	Female	5200
Break_Out_Category	42640	4	Race	20800
Category	42640	1	Cardiovascular Diseases	42640
CategoryId	42640	1	C1	42640
DataSource	42640	1	Medicare	42640
Data_Value_Footnote	529	1	Statistically unstable estimates not presented...	529
Data_Value_Footnote_Symbol	529	1	~	529
Data_Value_Type	42640	1	Crude	42640
Data_Value_TypeID	42640	1	Crude	42640
Data_Value_Unit	42640	2	Percent (%)	28080
GeoLocation	41820	51	(21.304850435000446, -157.85774940299973)	820
Indicator	42640	10	Prevalence of all heart disease hospitalizatio...	4680
IndicatorID	42640	10	MD101	4680
LocationAbbr	42640	52	AK	820
LocationDesc	42640	52	Alabama	820
LocationID	42640	52	1	820
PriorityArea1	9360	1	Million Hearts	9360
PriorityArea3	14560	1	Healthy People 2020	14560
Topic	42640	6	Heart Failure	19240
TopicId	42640	6	T5	19240
Year	42640	10	2004	4264

Figure 3.1 Attribute's unique value count

To increase data quality, it is essential to identify and remove variables with low variability. The availability of an attribute value with Low variability in a dataset indicates that attribute does not provide much information for analysis or model training. Specifically, attributes with only one unique value as shown in *Figure 3.1* have low variability and do not contribute meaningful insights or distinctions within the data. By removing these attributes, we streamline the dataset, making it more efficient and potentially improving the performance of our machine learning models. Therefore, we identified and removed the attributes with only one unique value Figure 3.1to enhance the overall quality of the data.

3.1.2 Attributes with high variability

From the dataset, we observed that there are attributes with too high variability typically IDs. Extreme variability includes attributes with excessively large IDs, which can distort analysis and model performance. Redundant qualities encompass duplicate information, increasing complexity without adding value. By systematically eliminating these less relavent attributes,*Figure 3.2* , data processing and analysis become more efficient leading to better model performance and more accurate insights.

```
[180]: ## Low variability
       df = df.drop(columns=[col for col in df.columns if df[col].nunique() < 2])

[181]: ## drop the 'Indicator' attribute and typically IDs
       df = df[df.columns.difference(
           ['Indicator', 'IndicatorID', 'BreakOutCategoryId', 'BreakOutId', 'TopicId'])]
```

Figure 3.2 Dropping ID features

3.1.3 Comparing location attributes

LocationAbbr	LocationDesc	GeoLocation	LocationID	
AK	Alaska	(64.84507995700051, -147.72205903599973)	2	820
PA	Pennsylvania	(40.79373015200048, -77.8607002939963)	42	820
ND	North Dakota	(47.47531977900047, -100.11842184899966)	38	820
NE	Nebraska	(41.6410400880005, -99.36572062299967)	31	820
NH	New Hampshire	(43.65595011300047, -71.50036091999965)	33	820
NJ	New Jersey	(40.13857004800049, -74.27369128799967)	34	820
NM	New Mexico	(34.52888095200048, -106.24058098499967)	35	820
NV	Nevada	(39.49324039000494, -117.07184856399967)	32	820
NY	New York	(42.82708183200045, -75.54397042699964)	36	820
OH	Ohio	(40.06821014100048, -82.40426005599966)	39	820
OK	Oklahoma	(35.47203135600046, -97.52187021399968)	40	820
OR	Oregon	(44.56744942400047, -120.15503132599969)	41	820
RI	Rhode Island	(41.70828019300046, -71.52247031399963)	44	820
MT	Montana	(47.06652897200047, -109.42442064499971)	30	820
SC	South Carolina	(33.99882130300454, -81.04537120699968)	45	820
SD	South Dakota	(44.35313005300484, -100.37353063699997)	46	820
TN	Tennessee	(35.68894058000048, -85.77440091399967)	47	820
TX	Texas	(31.82724007000488, -99.42677028599967)	48	820
UT	Utah	(39.36870017100474, -111.58713063499971)	49	820
VA	Virginia	(37.54268067400045, -78.45789046299967)	51	820
VT	Vermont	(43.62538123900049, -72.51764079899962)	50	820
WA	Washington	(47.52227862900048, -120.47001078999972)	53	820
WI	Wisconsin	(44.39319117400049, -89.81637074199966)	55	820
WV	West Virginia	(38.66551020200046, -80.71264013499967)	54	820
NC	North Carolina	(35.46622897500454, -79.15925046299964)	37	820
MS	Mississippi	(32.74551009900455, -89.53803082499968)	28	820
AL	Alabama	(32.84057112200048, -86.63186076199969)	1	820
IA	Iowa	(42.46948091300047, -93.81649055599968)	19	820
AR	Arkansas	(34.74865012400045, -92.27449074299966)	5	820
AZ	Arizona	(34.86597028800454, -111.76381127699972)	4	820
CA	California	(37.63864012300047, -120.99999953799971)	6	820
CO	Colorado	(38.84384075800464, -106.13361892899967)	8	820
CT	Connecticut	(41.56266182000046, -72.6498405199964)	9	820
DC	Washington, DC	(38.89037138500049, -77.03196112699965)	11	820
DE	Delaware	(39.00883066700495, -75.57774116799965)	10	820
FL	Florida	(28.93204037700476, -81.92896853899966)	12	820
GA	Georgia	(32.83968189300048, -83.62758834599966)	13	820
HI	Hawaii	(21.30485043500446, -157.85774940299973)	15	820
ID	Idaho	(43.68263800500476, -114.36373084199997)	16	820
MO	Missouri	(38.63579077600476, -92.56630005299968)	29	820
IL	Illinois	(40.48501028300046, -88.99771017799969)	17	820
IN	Indiana	(39.76691045200445, -86.14996019399968)	18	820
KS	Kansas	(38.34774030000045, -98.20078122699965)	20	820
KY	Kentucky	(37.64597027100465, -84.77497184799966)	21	820
LA	Louisiana	(31.31266064400046, -92.44568087899969)	22	820
MA	Massachusetts	(42.27687047000046, -72.08269067499964)	25	820
MD	Maryland	(39.29058096400047, -76.60926011899963)	24	820
ME	Maine	(45.25422889400585, -68.98503133599962)	23	820
MI	Michigan	(44.6613195430005, -84.71439026999968)	26	820
MN	Minnesota	(46.35564873600049, -94.79420058299967)	27	820
WY	Wyoming	(43.23554134300048, -108.10083035299967)	56	820

Name: count, dtype: int64

Figure 3.3 Comparing location related attributes

From the *Figure 3.3*, we can note that *LocationAbbr* is the abbreviation of LocationDesc, GeoLocation is the magnitude and longitude representation of place, and LocationID is the ID for
LocationDes. We can keep LocationAbbr and remove the other columns to avoid data redundcy.

3.2 Handling Missing Values

Missing values in datasets can introduce significant problems, including bias in statistical analysis, reduced data accuracy, and compromised model performance. They can distort the underlying data patterns, leading to incorrect inferences and predictions. Therefore, it is crucial to handle missing data properly to ensure robust and reliable results. In the original dataset, the information about the missing values of each attribute is depicted in the *Figure 3.4* below. The gaps with white colour represent the missing values of attributes.

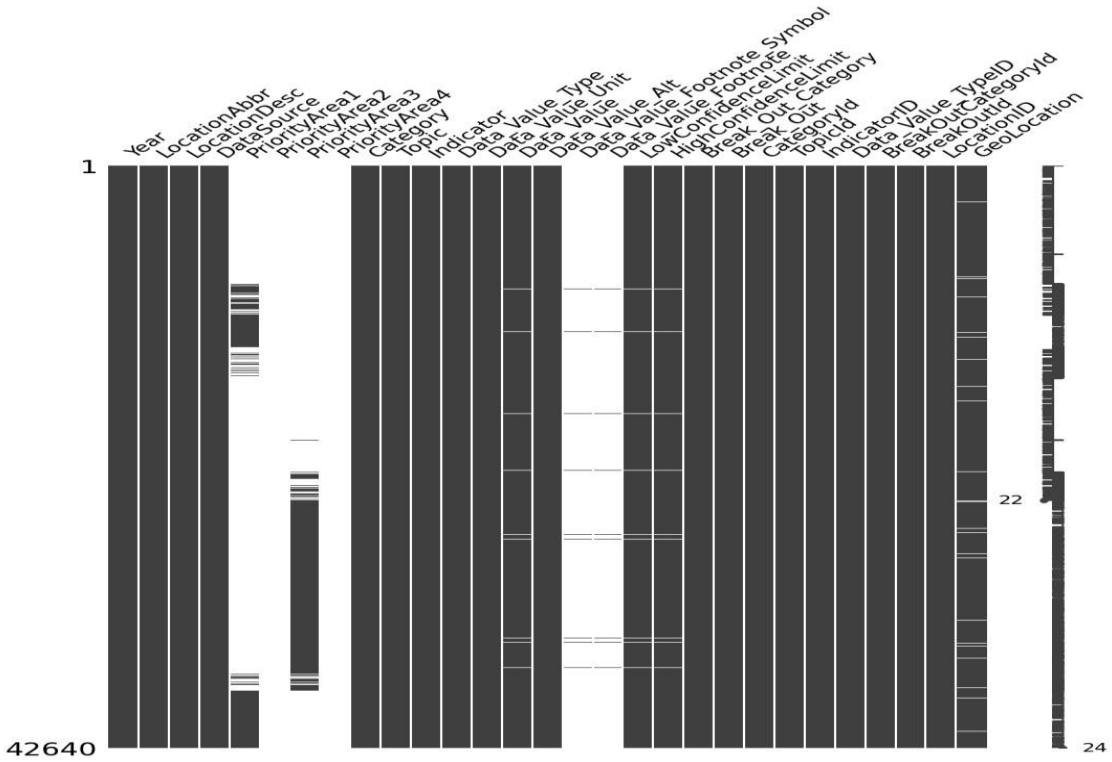


Figure 3.4 Highlighting missed values

After dropping attributes which are considered as irrelevant to our model, we have found 529 missing values for attributes, HighConfidenceLimit and LowConfidenceLimit, *Figure 3.5*.

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

Break_Out                0
Break_Out_Category       0
Data_Value_Alt           0
Data_Value_Unit          0
HighConfidenceLimit    529
LocationAbbr             0
LowConfidenceLimit     529
Topic                    0
Year                     0
dtype: int64
```

Figure 3.5 Attribute missed values count

3.2.1 Filling Missing values

We've opted to fill missing values using the median as shown in *Figure 3.6* because our data isn't normally distributed. Unlike the mean, which can be skewed by outliers in non-normally distributed data, the median is robust and provides a more accurate measure of central tendency. This ensures our imputed values are representative and reliable for analysis, maintaining the integrity of our results.

Figure 3.6 Filling missed values with median

3.2.2 Removing duplicates

3.2.2.1 Removing duplicate values

After filling in the missing data, we discovered 176 instances of duplicate rows, which we subsequently eliminated.

3.2.2.2 Avoiding contradictory examples in the data set

When encountering contradictory examples within our dataset, which occur when the same record is assigned to different topics or categories, it's recommended to address these inconsistencies. To ensure data integrity, we leverage the functionalities of pandas, a powerful data manipulation tool, to systematically identify and remove these conflicting instances from our dataset. This approach helps maintain the accuracy and reliability of our data for subsequent analyses and modeling tasks.

4 Data Preparation

4.1 Encoding Categorical Attributes

Converting non-numeric categorical features into numerical format is suitable for machine learning algorithms, as it improves the model's ability to capture patterns and make accurate predictions. Here are some of the encoding classes in scikit-learn library we have used in our project implementation:

4.1.1 One-Hot Encoding

One-hot encoding transforms categorical variables into a series of binary columns. Each unique category in the original variable becomes a separate column, and a row's entry in each of these columns is either 0 or 1, indicating the presence or absence of that category. This is typically used for nominal (unordered) categorical variables. In our project we used this to encode

nominal features such as *Break_Out*, *Break_Out_Category*, *Data_Value_Unit* and *LocationAbbr*.

4.1.2 Ordinal Encoding

Ordinal encoding converts categorical values into numeric values by assigning integers in a way that reflects the ordinal (ordered) nature of the categories. In our project '*Year*' is an ordinal attribute having inherent order, so it is encoded with this encoding mechanism.

4.1.3 Label Encoding

Label encoding converts categorical values into numeric values by assigning each unique category a different integer value. The target attribute (*Topic*) is encoded to numeric values by importing this class.

4.2 Normalization

Feature scaling is an important step in data preparation that involves standardizing the range of numerical attributes to ensure that they have similar scales. This is crucial for machine learning algorithms that rely on distance-based measures or gradient-based optimization techniques.

By scaling the features, we have ensured that our numerical attributes have similar scales leading to improved performance and convergence of our machine learning models.

In our scenario, we opted for Standard Scaler, a form of Standardization, to preprocess both the training and testing data. By removing the mean and scaling to unit variance, Standard Scaler ensures that the numeric attributes are standardized and ready for training the machine learning model. This not only improves the model's performance but also makes it more resilient to outliers and varying scales in the data.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Assuming index_numeric is a list of column indices
scaler.fit(x_train.iloc[:, index_numeric])
x_train.iloc[:, index_numeric] = scaler.transform(x_train.iloc[:, index_numeric])
x_test.iloc[:, index_numeric] = scaler.transform(x_test.iloc[:, index_numeric])
x_train
```

Figure 4.1 Normalization

4.3 Dataset Split

We split the dataset into three subsets: Train, Validation, and Test to ensure our model's performance is accurately evaluated and to generalize well to unseen data. Here's a step-by-step guide on how to use these datasets:

Training Set: This subset of the data is used to train a model. The model learns from this data, adjusting its parameters to minimize the error.

Validation Set: This subset is used to tune the hyperparameters of the model and to provide an unbiased evaluation of the model fit on the training dataset. It helps in selecting the best model.

Test Set: This final subset is used to assess the performance of the trained model on unseen data. It provides an unbiased evaluation of the final model.

The following *Figure 4.2* show how our dataset is split into training, validation, and test sets. Initially, the dataset was split into 80% training data and 20% test data using the train-test split function from Scikit-Learn's model selection module. Then, the trainng dataset is further split into 80% training set and 20% validation set. The random state parameter is set to a fixed seed value (12062024) to ensure reproducibility of the splits. Here is the general visualization of the data split in our model.

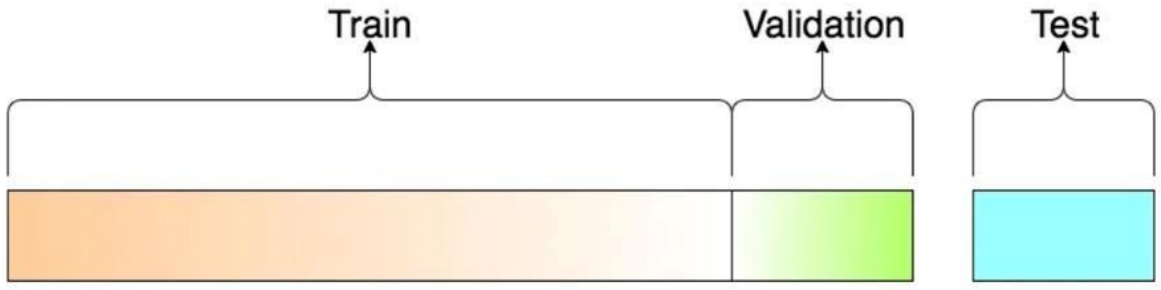


Figure 4.2 Visualization of the splits

4.4 Data Resampling

The dataset we used as a case study in our project has a huge difference between the class distributions, particularly focusing on the 'heart failure' class compared to other class labels as shown in the *Figure 4.3* below.

Heart Failure	18872
Major Cardiovascular Disease	4654
Diseases of the Heart (Heart Disease)	4642
Acute Myocardial Infarction (Heart Attack)	4517
Coronary Heart Disease	4496
Stroke	4457

Figure 4.3 Imbalanced class distribution

Imbalanced data refers to datasets where the distribution of class labels is not equal, with one class having a significantly higher number of observations than the other. This can be a problem for machine learning algorithms, as the classification can be biased towards the majority class and perform poorly on the minority class.

Excessive oversampling leads to overfitting as the same instances are repeated multiple times, while excessive under sampling leads to underfitting due to the loss of information. Therefore, a balance needs to be found between oversampling and under sampling. In our case, to handle the highly imbalanced dataset, we utilized the combination of both resampling methods to balance the class distribution.

4.4.1 Combining Random Oversampling and Under sampling.

In our approach:

By combining these two techniques, we struck a balance between oversampling and under sampling, avoiding the pitfalls of overfitting and underfitting. The oversampling helped increase the representation of minority classes, while the under sampling prevented the model from being overwhelmed by the majority class.

1. Random Oversampling of Minority Classes: We made changes on the minority classes, which had significantly fewer observations compared to the majority class. By randomly duplicating instances from these minority classes, we aimed to increase their representation in the dataset. However, we didn't match the majority class exactly, but rather bring them closer to approximately 50% of the majority class.

2. Random Under sampling of Majority Class: Concurrently, we addressed the imbalance by under sampling the majority class. This step aimed to reduce the dominance of the majority class while still maintaining a representative sample size. By randomly removing instances from the majority class, we attempted the model from being biased towards this class while ensuring that the dataset remained manageable.

This balanced approach aimed to provide a more accurate representation of the dataset, enabling machine learning algorithms to learn from all classes effectively without being biased towards any specific class.

5 Modeling

5.1 Model Selection

Certain machine learning models are better suited for multi-class classification problems due to their inherent characteristics or the availability of techniques specifically designed to handle multiple classes. The following models are typically more effective for multi-class data, leading to improved performance and better predictive outcomes.

1. Decision Tree Classifier
2. Naive Bayes Model
3. Random Forest Classifier
4. Support Vector Machine (SVM)
5. Neural Network (MLPClassifier)

5.1.1 Decision Tree Classifier

The C4.5 Decision Tree algorithm is implemented with a specific criterion called entropy, which measures the impurity of a node in the tree. This criterion is used to decide the best splits at each node during the tree construction process. *Figure 5.1* shows the results of the performance metrics for this model using **entropy** as criterion.

C45
Classification metrics:

	precision	recall	f1-score	support
0	0.94	0.89	0.91	3009
1	0.95	0.95	0.95	762
2	0.94	0.94	0.94	770
3	0.68	0.79	0.73	705
4	0.85	0.87	0.86	690
5	0.69	0.73	0.71	726
accuracy			0.87	6662
macro avg	0.84	0.86	0.85	6662
weighted avg	0.88	0.87	0.87	6662

Figure 5.1 Decision tree result using Entropy

We have also implemented the CART decision tree algorithm that utilizes Gini impurity as the criterion for node splitting. Gini impurity measures the probability of incorrectly classifying a randomly chosen element in the dataset. The performance results of the model trained on the dataset using this algorithm are are shown below in *Figure 5.2*:

CART
Classification metrics:

	precision	recall	f1-score	support
0	0.94	0.89	0.92	3009
1	0.96	0.96	0.96	762
2	0.95	0.95	0.95	770
3	0.68	0.77	0.72	705
4	0.85	0.85	0.85	690
5	0.68	0.75	0.71	726
accuracy			0.87	6662
macro avg	0.84	0.86	0.85	6662
weighted avg	0.88	0.87	0.88	6662

Figure 5.2 Decision tree result using Gini

5.1.2 Naive Bayes Model

In our project, the Naive Bayes model, leveraging the Gaussian Naive Bayes variant, is utilized as a probabilistic classifier based on Bayes' Theorem, assuming feature independence given the class label. This implementation offers simplicity and efficiency in probabilistic classification tasks, particularly effective for continuous features and we found the performance metrics shown in *Figure 5.3* below.

GaussianNaiveBayes
Classification metrics:

	precision	recall	f1-score	support
0	1.00	0.77	0.87	3009
1	0.81	0.84	0.82	762
2	0.82	0.81	0.81	770
3	0.53	0.74	0.61	705
4	0.58	0.64	0.61	690
5	0.40	0.57	0.47	726
accuracy			0.74	6662
macro avg	0.69	0.73	0.70	6662
weighted avg	0.80	0.74	0.76	6662

Figure 5.3 Naïve Bayes Result

5.1.3 Random Forest Classifier

The Random Forest algorithm is an ensemble method that combines multiple decision trees to improve classification accuracy and robustness. In this implementation, 100 decision trees are utilized within the forest. Each tree is constructed independently, and the final classification decision is made by aggregating the predictions of all trees. After implementing this model, the metrics shown in *Figure 5.4* were obtained.

RandomForest
Classification metrics:

	precision	recall	f1-score	support
0	0.97	0.87	0.92	3009
1	0.97	0.97	0.97	762
2	0.96	0.97	0.96	770
3	0.67	0.84	0.75	705
4	0.84	0.87	0.86	690
5	0.68	0.79	0.73	726
accuracy			0.88	6662
macro avg	0.85	0.88	0.86	6662
weighted avg	0.89	0.88	0.88	6662

Figure 5.4 Random Forest Classifier Result

5.1.4 Support Vector Machine (SVM)

SVM is employed in our project and it works by finding the hyperplane that best separates the classes in the feature space. In this implementation, a linear kernel is used, which means the decision boundary is a straight line. SVM aims to maximize the margin between classes, leading to better generalization performance. At the end, we observed the performance of this model as shown in *Figure 5.5*.

SVM
Classification metrics:

	precision	recall	f1-score	support
0	1.00	0.79	0.88	3009
1	0.97	0.97	0.97	762
2	0.97	0.98	0.97	770
3	0.57	0.89	0.70	705
4	0.85	0.88	0.87	690
5	0.65	0.82	0.73	726
accuracy			0.86	6662
macro avg	0.84	0.89	0.85	6662
weighted avg	0.89	0.86	0.87	6662

Figure 5.5 Support Vector Machine Result

5.1.5 Neural Network (MLPClassifier)

a Multi-Layer Perceptron (MLP) neural network is employed for classification tasks. It consists of multiple layers of nodes, including an input layer, one hidden layer with 50 neurons, and an output layer with 10 neurons. The network is trained using backpropagation with a maximum of 500 iterations. By specifying a random state, we ensure that the initialization of the neural network weights is consistent across different runs, which aids in reproducibility and comparison of results. The MLP provides flexibility in modeling complex relationships in the data, making it suitable for various classification tasks in our project. After all, we obtained the results shown in *Figure* 5.6.

Classification metrics:

	precision	recall	f1-score	support
0	0.98	0.91	0.94	3009
1	0.97	0.98	0.97	762
2	0.97	0.97	0.97	770
3	0.73	0.90	0.81	705
4	0.87	0.86	0.87	690
5	0.73	0.78	0.75	726
accuracy			0.90	6662
macro avg	0.88	0.90	0.89	6662
weighted avg	0.91	0.90	0.91	6662

Figure 5.6 Neural Networks Result

5.2 Model Comparison and Selection for Testing

We can see the classifier performance metrics for each model taking Accuracy and F1 score into account.

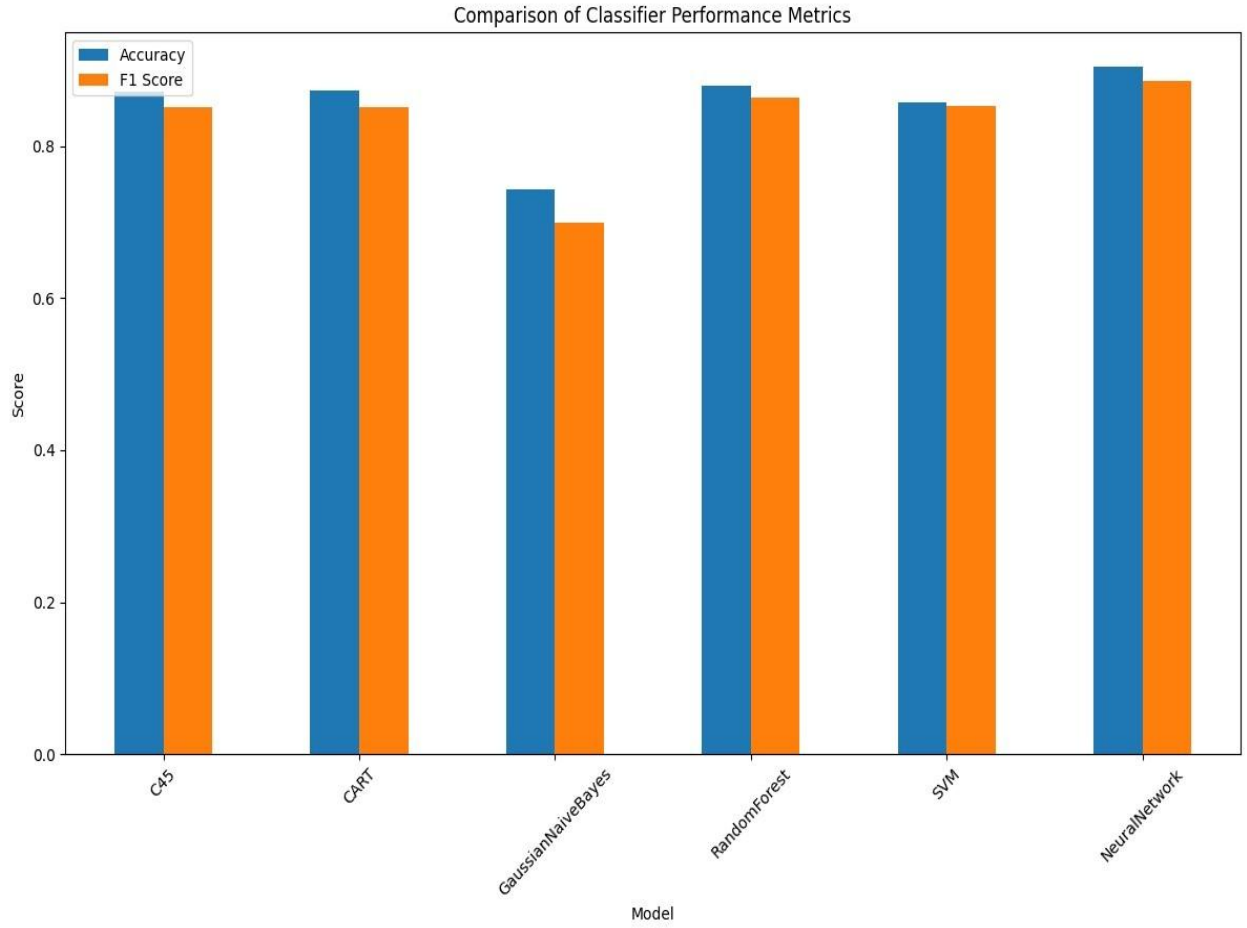


Figure 5.7 comparison of classifier performance

The *Figure 5.7* above illustrates a comparison of classifier performance metrics, specifically Accuracy and F1 Score, for six models: C4.5, CART, Gaussian Naive Bayes, Random Forest, SVM, and Neural Network. The C4.5, Random Forest, and Neural Network models exhibit the highest performance, with both Accuracy and F1 Score close to 0.9. The CART and SVM models perform moderately well, with both metrics slightly above 0.8. In contrast, the Gaussian Naive Bayes model demonstrates the lowest performance among the six models, with both metrics slightly above 0.7. Overall, the C4.5, Random Forest, and Neural Network models stand out as the top performers, while the Gaussian Naive Bayes model lags in both Accuracy and F1 Score

6 Evaluation

After evaluating all models, we selected the neural network as the best model based on its highest accuracy and F1-score on the validation set. Finally, we used the test set to provide an unbiased evaluation of our best-performing model, ensuring that it is well-tuned, and its performance is accurately assessed on unseen data.

confusion matrix is a useful tool to understand the performance of a classification model. It compares the actual target values with those predicted by the model.

6.1.1 Confusion Matrix

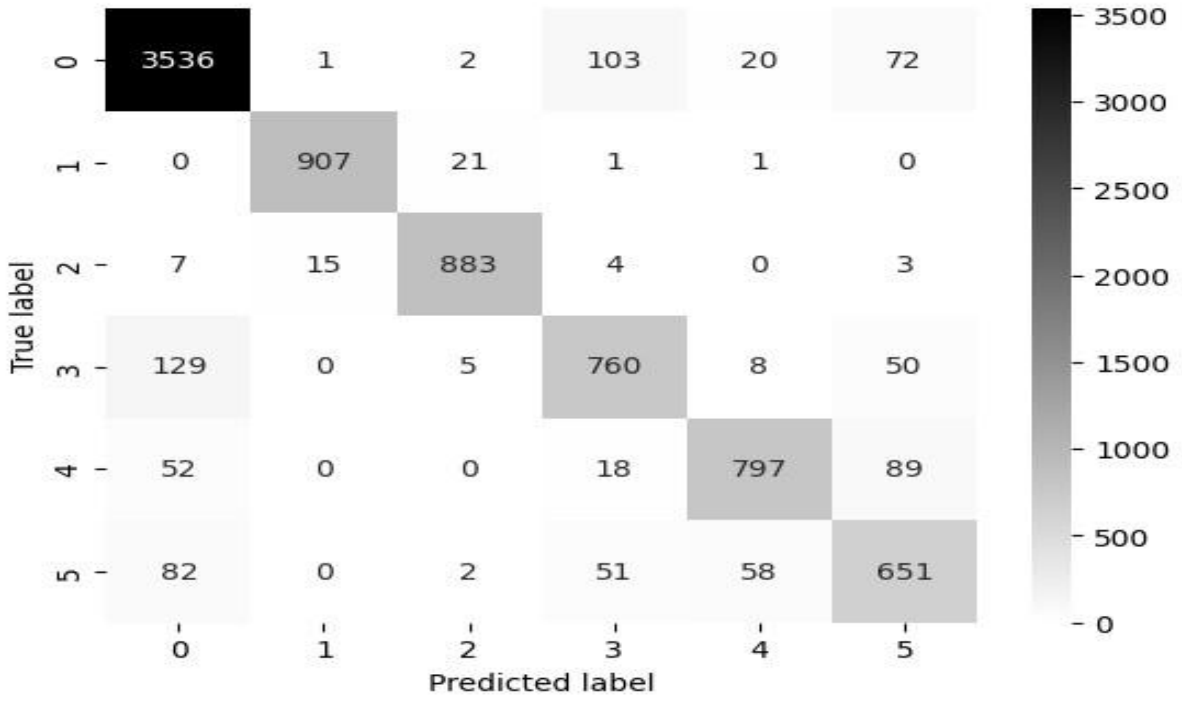


Figure 6.1 Confusion Matrix

The confusion matrix shown above in *Figure 6.1* is a visual representation of the performance of the neural network model on the test set.

Structure of the Confusion Matrix:

- Rows represent the actual classes (True Labels).
- Columns represent the predicted classes (Predicted Labels).

Each cell in the matrix indicates how many instances of a particular actual class were predicted as each class. Notably, the diagonal elements represent the correctly classified instances (True Positives), while the off-diagonal elements represent misclassifications.

6.1.1.1 Summary of the above confusion matrix.

Class label	Target class (Topic)	True Positives (correctly classified)	Total class instances
0	Heart Failure	3536	3734
1	Major Cardiovascular Disease	907	930
2	Diseases of the Heart (heart disease)	883	912
3	Acute Myocardial Infarction (Heart Attack)	760	952
4	Coronary Heart Disease	797	955
5	Stroke	651	844

Table 6.1 Confusion matrix summary

As shown in *Table 6.1* while the neural network model performs exceptionally well for Classes 0, 1, and 2, it struggles slightly with Classes 3, 4, and 5. This suggests that while the model is robust in cases with multi-class classification, there is room for improvement in the classes where the performance drops, indicating potential areas for further refinement and tuning.

7 Conclusion

In this project, we have developed an automated classification model for categorizing various cardiovascular diseases based on the provided data. We have employed different data preparation and modeling techniques to ensure that our models are accurate and reliable.

Among the five machine learning models we implemented to analyze the heart disease and stroke prevention dataset, the Neural Network outperformed the rest. Overall, the model correctly classifies 7,534 out of 8,327 data sets. This indicates that while the model is generally effective, further tuning is needed to enhance its accuracy for these specific classes. For future work, additional machine learning models should be explored to potentially improve the performance of the predictive model for this dataset.