PROGRAMMING FOR BIOINFORMATICS

(SECB3203)

FULL REPORT

ANALYZING HEART ATTACK DETECTION THROUGH VARIOUS CLASSIFICATION ALGORITHMS

Lecturer:

DR. NIES HUI WEN

Client:

DR. NOOR HIDAYAH BINTI ZAKARIA

Group Members:

No	Name	Matric No
1	JELIZA JUSTINE A/P SEBASTIN	A21EC0034
2	NUR AISYAH FATIHAH BINTI MOHAMED ROZI	A21EC0107
3	SITI NURKAMILAH BINTI SAIFUL BAHARI	A21EC0131

Table of Contents

Table of Contents	**2**
1.0 Introduction	**3**
1.1 Problem Background	**4**
1.2 Problem Statement	**4**
1.3 Objectives	**5**
1.4 Scopes	**6**
2.0 Data collection and preprocessing	**7**
2.1 Importing dataset	**8**
2.2 Data wrangling	**11**
2.3 Software and hardware requirements	**22**
3.0 Flowchart of the proposed approach	**23**
3.1 Exploratory data analysis	**25**
3.2 Model Development	**35**
3.3 Model evaluation	**43**
4.0 Testing, and Validation	**51**
4.1 Feedback collected from your client(s)	**52**
5.0 Conclusions	**53**
6.0 Appendix	**53**
7.0 References	**54**

1.0 Introduction

This research paper focuses on the domain of heart attack disease, a life-threatening medical emergency necessitating immediate treatment. Heart attacks, or myocardial infarctions, occur when blood flow carrying oxygen to a portion of the heart muscle is suddenly blocked, which results in oxygen deprivation. Without prompt restoration of blood flow, the heart muscle starts to deteriorate. In the vast majority of cases, heart attacks are caused by coronary artery disease, and a number of factors may increase the risk, such as age, lifestyle, and underlying medical conditions. Pain in the chest and upper body, dizziness, sweating profusely, and nausea are all symptoms of a heart attack (National Heart Lung and Blood Institute,2022).

Heart attacks are currently diagnosed based on whether someone suffered a ST segment elevation myocardial infarction (STEMI) or another type of acute coronary syndrome (NSTEMI or unstable angina)If someone shows heart attack symptoms confirmed by an electrocardiogram (ECG) to be a STEMI, the treatment is based on symptom onset time and treatment accessibility. Symptoms within 12 hours usually lead to coronary angiography and primary percutaneous coronary intervention (primary PCI). If immediate PCI isn't feasible within 12 hours, patients receive medication to dissolve blood clots. If symptoms are present beyond 12 hours, different treatments can be performed, which can include medication, PCI, or bypass surgery after an angiogram has been performed (NHS, 2023).

Recent research explores heart attack prediction using machine learning. The research explores six different modeling techniques: Decision Tree, K-Nearest Neighbor (KNN), Logistic Regression, Random Forest, Extreme Gradient Boost, and Support Vector Machine (SVM). Based on the experimental results, K-Nearest Neighbor (KNN) exhibited the highest precision, and accuracy among the six modeling techniques (Rizwan et al., 2022). In a study by Manjula et al. (2022) the Random Forest algorithm was found to be a powerful tool for both regression and classification procedures when compared to Logistic Regression, Naïve Bayes, SVM, KNN, Decision Tree, and XGBoost. This research paper will focus on the accuracy of Logistic Regression and Gaussian Naive Bayes methods of machine learning for predicting heart attacks.

1.1 Problem Background

According to the National Institutes of Health(2021), in 2019, 9.6 million men and 8.9 million women globally succumbed to cardiovascular disease, primarily affecting those aged 30 to 70. Myocardial infarction, a consequence of heart disease, is often caused by arterial plaques, leading to restricted blood flow and muscle tissue damage. Referring to Centers for Disease Control and Prevention (n.d), in the United States, someone experiences a heart attack every 40 seconds, emphasizing the need for early and accurate detection. A large amount of data generated by the medical industry has not been used effectively previously. Thankfully, the integration of artificial intelligence into medical diagnosis provides a promising solution.These approaches consistently demonstrate a high level of accuracy, affirming their effectiveness (Zhang. Wan & Han. J, 2017). The novel methods introduced here not only cut costs but also enhance the accuracy of heart disease prediction in a straightforward and efficient manner (Mohan et al., 2019). Our approach involves a comparison of two algorithms, logistic regression and Gaussian Naive Bayes, to estimate the likelihood of a heart attack. This approach is tailored to support healthcare professionals in delivering timely and precise assessments of heart health.

1.2 Problem Statement

The central concern lies in the provision of early and accurate heart attack identification, which is a critical matter with major implications for healthcare. The application of a diverse machine learning approach is expected to contribute to the enhancement of disease detection accuracy and timeliness, thus resulting in significant improvements in healthcare outcomes and the advancement of disease diagnosis and management.

1.3 Objectives

- To implement and train machine learning models, specifically Logistic Regression and Gaussian Naive Bayes, for heart attack detection using the collected dataset.

- To evaluate the performance of the machine learning models in terms of accuracy, precision, recall, F1-score, and ROC AUC, using appropriate metrics and techniques.

- To compare the performance between the Logistic Regression and Gaussian Naive Bayes models to determine which model exhibits superior predictive capabilities for heart attack detection.

- To provide a detailed analysis of the strengths and weaknesses of the Logistic Regression and Gaussian Naive Bayes models for heart attack detection, and their suitability for real-world clinical applications.

1.4 Scopes

The scope are :

1. Dataset Description:
   - The dataset, acquired from Kaggle, consists of records from 303 subjects (instances) and includes 14 columns. This dataset serves as a robust foundation for our analysis and model development.
   - Dataset Source: Heart Attack Analysis & Prediction Dataset (kaggle.com)

2. Software Utilized:
   - The machine learning program is created using two software platforms: R language and VS Code (Python). These software tools enable us to develop and implement the machine learning framework effectively.
   - In addition to the core software, all necessary libraries and add-ons have been installed to support the research process.
3. Limitation:
   - A notable limitation in this research is the potential for processing time delays due to the use of less powerful computers. These delays may have implications for the efficiency of data analysis and the overall research timeline.

2.0 Data collection and preprocessing

In this project aimed at heart attack detection, we initiate the process by sourcing a dataset from Kaggle, which comprises 303 instances and 13 attributes, forming the bedrock of our investigation. Subsequently, data preprocessing takes center stage, where data cleaning is employed to address issues such as missing values, duplicate records, and outliers. This step is vital for enhancing data quality and reliability. To ensure that the data is amenable to modeling, we undergo data normalization techniques, such as Z-score normalization, which standardizes the scale across all features.

In the data analysis phase, we aim to identify the most important attributes for heart attack detection. To do this, we use the Pearson correlation coefficient. We focus on two key criteria: a p-value below 0.0001 (indicating a strong statistical relationship) and a coefficient close to 1 (indicating a strong positive relationship). Attributes meeting these criteria are selected. Feature selection reduces dataset complexity, boosts model efficiency, and prevents overfitting, ensuring our models focus on the most influential attributes for accurate heart attack prediction. After the feature selection process, we proceed to train our machine learning models, specifically Gaussian Naive Bayes and Logistic Regression. We rigorously assess the performance of each machine learning model using appropriate evaluation metrics and techniques.

The key metrics we focus on include accuracy, recall, and the F1-score.on, for heart attack detection. In the end, we compare Gaussian Naive Bayes and Logistic Regression to figure out which one does a better job at heart attack detection. This comparison is like a final test to see which algorithm performs more accurately. We do this very carefully to make sure our research provides valuable insights for heart attack detection.

The dataset employed in this project, denoted as "heart.csv," is sourced from the Kaggle dataset titled "Heart Attack Analysis & Prediction Dataset." This comprehensive dataset comprises a total of 303 instances, each characterized by 14 distinct attributes.

2.1 Importing dataset

In the realm of data science, the initial step often involves acquiring and importing datasets for analysis. The dataset at the center of our exploration is "heart.csv," sourced from Kaggle's "Heart Attack Analysis & Prediction Dataset." The importation of this dataset into Python lays the foundation for subsequent analysis and model development.

```
# Import the pandas library as pd
import pandas as pd


#import data
path = 'C:\\Users\\ASUS\\Downloads\\archive\\heart.csv'
df = pd.read_csv(path)

# Display the first few rows of the DataFrame
print(df.head(20))

```

Initially, we utilized the Pandas library for data cleaning and preprocessing. We proceed to load the "heart.csv" dataset by specifying its file path and assigning it to a DataFrame, which we label as df. Finally, we print **df.head(20)** to showcase the first 20 rows of the DataFrame derived from the "heart.csv" dataset.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

wnloads/hello.py
    age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak  slp  caa  thall  output
0    63    1   3     145   233    1        0       150     0      2.3    0    0      1       1
1    37    1   2     130   250    0        1       187     0      3.5    0    0      2       1
2    41    0   1     130   204    0        0       172     0      1.4    2    0      2       1
3    56    1   1     120   236    0        1       178     0      0.8    2    0      2       1
4    57    0   0     120   354    0        1       163     1      0.6    2    0      2       1
5    57    1   0     140   192    0        1       148     0      0.4    1    0      1       1
6    56    0   1     140   294    0        0       153     0      1.3    1    0      2       1
7    44    1   1     120   263    0        1       173     0      0.0    2    0      3       1
8    52    1   2     172   199    1        1       162     0      0.5    2    0      3       1
9    57    1   2     150   168    0        1       174     0      1.6    2    0      2       1
10   54    1   0     140   239    0        1       160     0      1.2    2    0      2       1
11   48    0   2     130   275    0        1       139     0      0.2    2    0      2       1
12   49    1   1     130   266    0        1       171     0      0.6    2    0      2       1
13   64    1   3     110   211    0        0       144     1      1.8    1    0      2       1
14   58    0   3     150   283    1        0       162     0      1.0    2    0      2       1
15   50    0   2     120   219    0        1       158     0      1.6    1    0      2       1
16   58    0   2     120   340    0        1       172     0      0.0    2    0      2       1
17   66    0   3     150   226    0        1       114     0      2.6    0    0      2       1
18   43    1   0     150   247    0        1       171     0      1.5    2    0      2       1
19   69    0   3     140   239    0        1       151     0      1.8    2    2      2       1
PS C:\Users\ASUS>
```

According to the figure above, a representation is provided, illustrating 20 rows of the DataFrame extracted from the "heart.csv" file. This Data Frame encompasses 14 columns of data.

```
# Display the first few rows of the DataFrame

print(df.head())
print(df.info())
print(df.describe())
```

- **print(df.head()**): This command prints the first few rows of the DataFrame, typically the top 5 rows by default. It is useful for a quick glance at the dataset.
- **print(df.info()):** This command provides a concise summary of the DataFrame df. It includes information about the data types of each column, the number of non-null values, and the memory usage. It's helpful for understanding the overall structure of the Data Frame

- **.print(df.describe())**: This command generates descriptive statistics of the DataFrame df. It includes measures like count, mean, standard deviation, minimum, 25th percentile, median (50th percentile), 75th percentile, and maximum. It provides insights into the central tendency and spread of the numerical columns in the dataset.

```
 13  output    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
None
              age         sex          cp  ...         caa       thall      output
count  303.000000  303.000000  303.000000  ...  303.000000  303.000000  303.000000
mean    54.366337    0.683168    0.966997  ...    0.729373    2.313531    0.544554
std      9.082101    0.466011    1.032052  ...    1.022606    0.612277    0.498835
min     29.000000    0.000000    0.000000  ...    0.000000    0.000000    0.000000
25%     47.500000    0.000000    0.000000  ...    0.000000    2.000000    0.000000
50%     55.000000    1.000000    1.000000  ...    0.000000    2.000000    1.000000
75%     61.000000    1.000000    2.000000  ...    1.000000    3.000000    1.000000
max     77.000000    1.000000    3.000000  ...    4.000000    3.000000    1.000000

[8 rows x 14 columns]
PS C:\Users\ASUS>
```

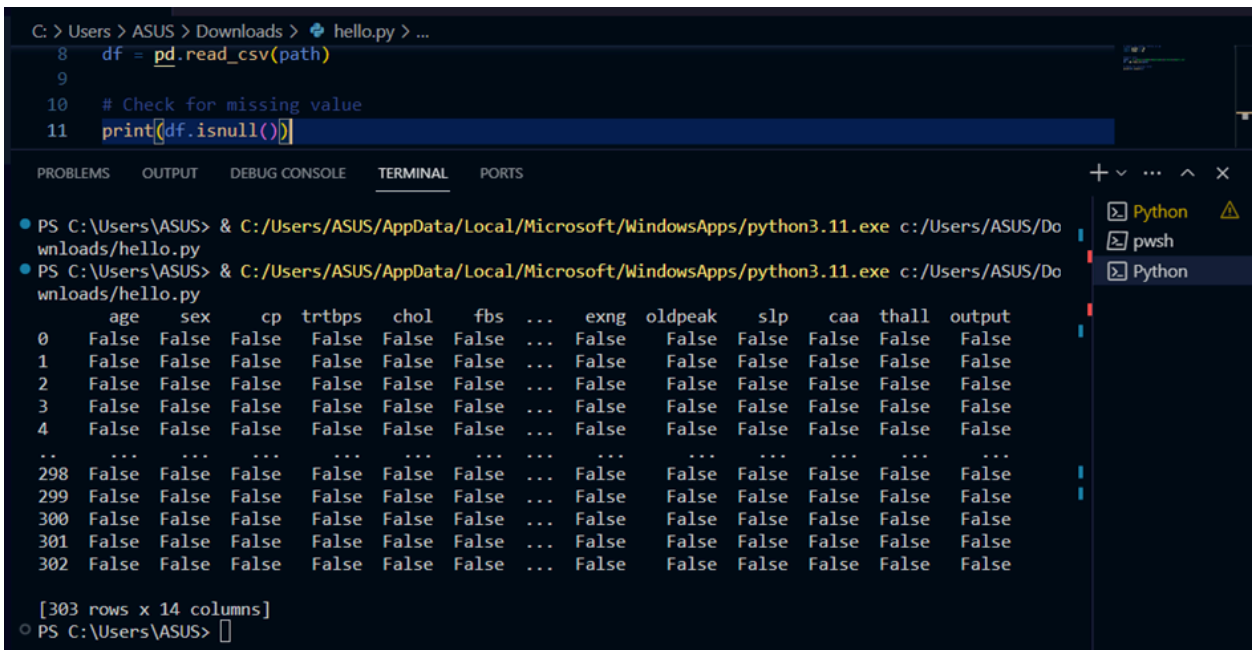
Here, we can understand the data size where it contains 8 rows and 14 columns.

2.2 Data wrangling

Identifying and handling missing values

Handling missing values is a crucial step in data analysis and machine learning because it directly impacts the accuracy and reliability of our results. When data contains gaps or missing information, it can introduce inaccuracies and biases into our analyses, leading to flawed conclusions.To check if our dataset contains null values or not, the syntax is:

print(df.isnull())



The statement " false" indicates that, upon checking each element in the dataset, none of them are identified as null or missing values. In the context of the dataset dimensions [303 rows x 14 columns], it means that across all 303 rows and 14 columns, there are no instances where data is absent or undefined. This is a positive outcome because it implies that our dataset is complete, and we have values for each attribute in every row, making it ready for further analysis without the need to address missing data. It's a reassuring sign of data integrity and completeness in this particular dataset.

```
# Check is missing values in each columns
print(df.isnull().sum())
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS> & C:/Users/ASUS/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ASUS/Do
wnloads/hello.py
PS C:\Users\ASUS> & C:/Users/ASUS/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ASUS/Do
wnloads/hello.py
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
output      0
dtype: int64
PS C:\Users\ASUS>
```

By using the syntax **print(df.isnull().sum)**, we performed a more detailed check on each column individually. The result, where each column shows a count of 0, reinforces our earlier finding that there are no null values present in the dataset. This column-wise examination is crucial because it allows us to pinpoint if there are any specific attributes or features that might have missing values.

```
# Identify duplicate rows
duplicate_rows = df[df.duplicated()]

# Print the duplicate rows
print(duplicate_rows)
```

After checking null values, we checked for duplicate values **(duplicate_rows = df[df.duplicate())** in our dataset because duplicate entries can lead to skewed results, affecting the reliability of statistical analyses and machine learning models. When there are duplicate rows, it might give undue weight to certain observations, leading to biased conclusions. Identifying and handling duplicate values is a crucial step in data cleaning and preprocessing.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ASUS> & C:/Users/ASUS/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/ASUS/Do
wnloads/hello.py
     age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak  slp  caa  thall  output
164   38    1   2     138   175    0        1       173     0      0.0    2    4      2       1
PS C:\Users\ASUS>
```

Figure above shows, there are duplicate values in row 164.

```
# Print the entire DataFrame
print(df)

# Print a subset of rows around the duplicate
print(df.loc[160:170])

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

     age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak  slp  caa  thall  output
160   56    1   1     120   240    0        1       169     0      0.0    0    0      2       1
161   55    0   1     132   342    0        1       166     0      1.2    2    0      2       1
162   41    1   1     120   157    0        1       182     0      0.0    2    0      2       1
163   38    1   2     138   175    0        1       173     0      0.0    2    4      2       1
164   38    1   2     138   175    0        1       173     0      0.0    2    4      2       1
165   67    1   0     160   286    0        0       108     1      1.5    1    3      2       0
166   67    1   0     120   229    0        0       129     1      2.6    1    2      3       0
167   62    0   0     140   268    0        0       160     0      3.6    0    2      2       0
168   63    1   0     130   254    0        0       147     0      1.4    1    1      3       0
169   53    1   0     140   203    1        0       155     1      3.1    0    0      3       0
170   56    1   2     130   256    1        0       142     1      0.6    1    1      1       0
PS C:\Users\ASUS>
```

To spot duplicate rows in our dataset,we printed the entire dataset with **print(df).** Then, to focus on rows 160 to 170 **(df_loc[160:170])**, we used slicing, like narrowing down our view to a

specific group of items. Just as you might notice we found that rows 163 and 164 had exactly the same information.

```
# Remove duplicate rows
df_no_duplicates = df.drop_duplicates()
print(df_no_duplicates)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
wnloads/hello.py
     age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak  slp  caa  thall  output
0     63    1   3     145   233    1        0       150     0      2.3    0    0      1       1
1     37    1   2     130   250    0        1       187     0      3.5    0    0      2       1
2     41    0   1     130   204    0        0       172     0      1.4    2    0      2       1
3     56    1   1     120   236    0        1       178     0      0.8    2    0      2       1
4     57    0   0     120   354    0        1       163     1      0.6    2    0      2       1
..   ...  ...  ..     ...   ...  ...      ...       ...   ...      ...  ...  ...    ...     ...
298   57    0   0     140   241    0        1       123     1      0.2    1    0      3       0
299   45    1   3     110   264    0        1       132     0      1.2    1    0      3       0
300   68    1   0     144   193    1        1       141     0      3.4    1    2      3       0
301   57    1   0     130   131    0        1       115     1      1.2    1    1      3       0
302   57    0   1     130   236    0        0       174     0      0.0    1    1      2       0

[302 rows x 14 columns]
```

To enhance the integrity of our dataset, we employed a useful tool called drop_duplicates(). It's like cleaning up duplicates in our collection to keep it neat. In our case, after executing df_no_duplicates = df.drop_duplicates(), the brown (our dataset) now has 302 rows instead of the original 303. It is because those identical rows we identified earlier were successfully removed. This process ensures that our data is more reliable and ready for insightful analysis.

Data formatting

Data formatting ensures that information is presented consistently and correctly. It involves tasks like converting data types, adjusting decimals, or representing dates uniformly.

```
# Display data types of each column
print(df.dtypes)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    POR

sex           int64
cp            int64
trtbps        int64
chol          int64
fbs           int64
restecg       int64
thalachh      int64
exng          int64
oldpeak     float64
slp           int64
caa           int64
thall         int64
output        int64
dtype: object
PS C:\Users\ASUS>
```

print(df.dtypes) is for displaying each data types in each column

- int64 represents integer data type.
- object represents string (or mixed) data type.
- float64 represents floating-point data type.

Our decision is to retain the current data types without making any changes. This choice is based on several factors:

- No NaN Values:

Since there are no missing values (NaN) in the integer columns,we don't need to convert them to float just to accommodate missing values. This avoids unnecessary conversions.

- Compatibility with Algorithms:

Logistic Regression and Gaussian Naive Bayes are flexible algorithms that can handle both integer and float data types.

Data Normalization

Before Normalization

```
df.describe()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000

```
df.describe()
```

l	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
6	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
1	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
0	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
0	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
0	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
0	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

- Count represents the number of non-null (non-missing) values in each column. It indicates the size of the dataset or the number of observations available for analysis in each specific column.
- Mean represents the average (mean) value of each column. It is calculated by summing up all the values in a column and then dividing by the total number of observations. The mean provides a measure of central tendency, giving an idea of the typical value in the dataset.
- Std represents the standard deviation, which measures the amount of variation or dispersion in each column. It quantifies how much individual data points differ from the

mean. A higher standard deviation indicates greater variability, while a lower standard deviation suggests that the values are closer to the mean.

- Min represents the minimum value in each column. It is the smallest observed value and provides insights into the lower bound of the data distribution. For example, the minimum age is 29.
- 25% (percentile) represents the 25th percentile, also known as the first quartile, is the value below which 25% of the data falls. It is a measure of the data's spread, helping to identify the lower range of the dataset.
- 50% (percentile) represents the 50th percentile, also known as the median, is the middle point of the data. It separates the lower 50% from the upper 50% of the dataset. The median is less sensitive to extreme values than the mean.
- 75% (percentile) represents the 75th percentile, also known as the third quartile, is the value below which 75% of the data falls. It represents the upper range of the dataset.
- Max represents the maximum value in each column. It is the largest observed value and provides insights into the upper bound of the data distribution. For example, the maximum age is 77.

After Normalization

```
df_normalized = (df - df.min()) / (df.max() - df.min())
print(df_normalized)
```

```
          age  sex        cp    trtbps      chol  fbs  restecg  thalachh  \
0    0.708333  1.0  1.000000  0.481132  0.244292  1.0      0.0  0.603053
1    0.166667  1.0  0.666667  0.339623  0.283105  0.0      0.5  0.885496
2    0.250000  0.0  0.333333  0.339623  0.178082  0.0      0.0  0.770992
3    0.562500  1.0  0.333333  0.245283  0.251142  0.0      0.5  0.816794
4    0.583333  0.0  0.000000  0.245283  0.520548  0.0      0.5  0.702290
..        ...  ...       ...       ...       ...  ...      ...       ...
298  0.583333  0.0  0.000000  0.433962  0.262557  0.0      0.5  0.396947
299  0.333333  1.0  1.000000  0.150943  0.315068  0.0      0.5  0.465649
300  0.812500  1.0  0.000000  0.471698  0.152968  1.0      0.5  0.534351
301  0.583333  1.0  0.000000  0.339623  0.011416  0.0      0.5  0.335878
302  0.583333  0.0  0.333333  0.339623  0.251142  0.0      0.0  0.786260

     exng   oldpeak  slp   caa     thall  output  normalized_column_minmax  \
0     0.0  0.370968  0.0  0.00  0.333333     1.0                  0.708333
1     0.0  0.564516  0.0  0.00  0.666667     1.0                  0.166667
2     0.0  0.225806  1.0  0.00  0.666667     1.0                  0.250000
3     0.0  0.129032  1.0  0.00  0.666667     1.0                  0.562500
4     1.0  0.096774  1.0  0.00  0.666667     1.0                  0.583333
..    ...       ...  ...   ...       ...     ...                       ...
298   1.0  0.032258  0.5  0.00  1.000000     0.0                  0.583333
299   0.0  0.193548  0.5  0.00  1.000000     0.0                  0.333333
300   0.0  0.548387  0.5  0.50  1.000000     0.0                  0.812500
301   1.0  0.193548  0.5  0.25  1.000000     0.0                  0.583333
302   0.0  0.000000  0.5  0.25  0.666667     0.0                  0.583333
```

```
     exng   oldpeak  slp   caa     thall  output  normalized_column_minmax  \
0     0.0  0.370968  0.0  0.00  0.333333     1.0                  0.708333
1     0.0  0.564516  0.0  0.00  0.666667     1.0                  0.166667
2     0.0  0.225806  1.0  0.00  0.666667     1.0                  0.250000
3     0.0  0.129032  1.0  0.00  0.666667     1.0                  0.562500
4     1.0  0.096774  1.0  0.00  0.666667     1.0                  0.583333
..    ...       ...  ...   ...       ...     ...                       ...
298   1.0  0.032258  0.5  0.00  1.000000     0.0                  0.583333
299   0.0  0.193548  0.5  0.00  1.000000     0.0                  0.333333
300   0.0  0.548387  0.5  0.50  1.000000     0.0                  0.812500
301   1.0  0.193548  0.5  0.25  1.000000     0.0                  0.583333
302   0.0  0.000000  0.5  0.25  0.666667     0.0                  0.583333

     normalized_column_zscore
0                    0.708333
1                    0.166667
2                    0.250000
3                    0.562500
4                    0.583333
..                        ...
298                  0.583333
299                  0.333333
300                  0.812500
301                  0.583333
302                  0.583333

[303 rows x 16 columns]
```

Data normalization is a preprocessing technique used in data analysis and machine learning to scale and transform features within a dataset to a common scale. The goal is to ensure that no single feature dominates the learning algorithm due to differences in the scale of the data. Normalization is particularly important for algorithms that rely on distance measures, such as k-nearest neighbors or gradient descent-based optimization algorithms. The data above is normalized by using Min-Max Scaling.

Min-Max scaling, also known as feature scaling, transforms the values of a numeric column to a specific range, usually [0, 1]. The formula for Min-Max scaling is as follows:

$X\text{normalized} = X - \min(X) \,/\, \max(X) - \min(X)$

Where X is the original value, $\min(X)$ is the minimum value in the column, and $\max(X)$ is the maximum value in the column. This transformation ensures that all values fall within the specified range as per in the result above.

Indicator variable

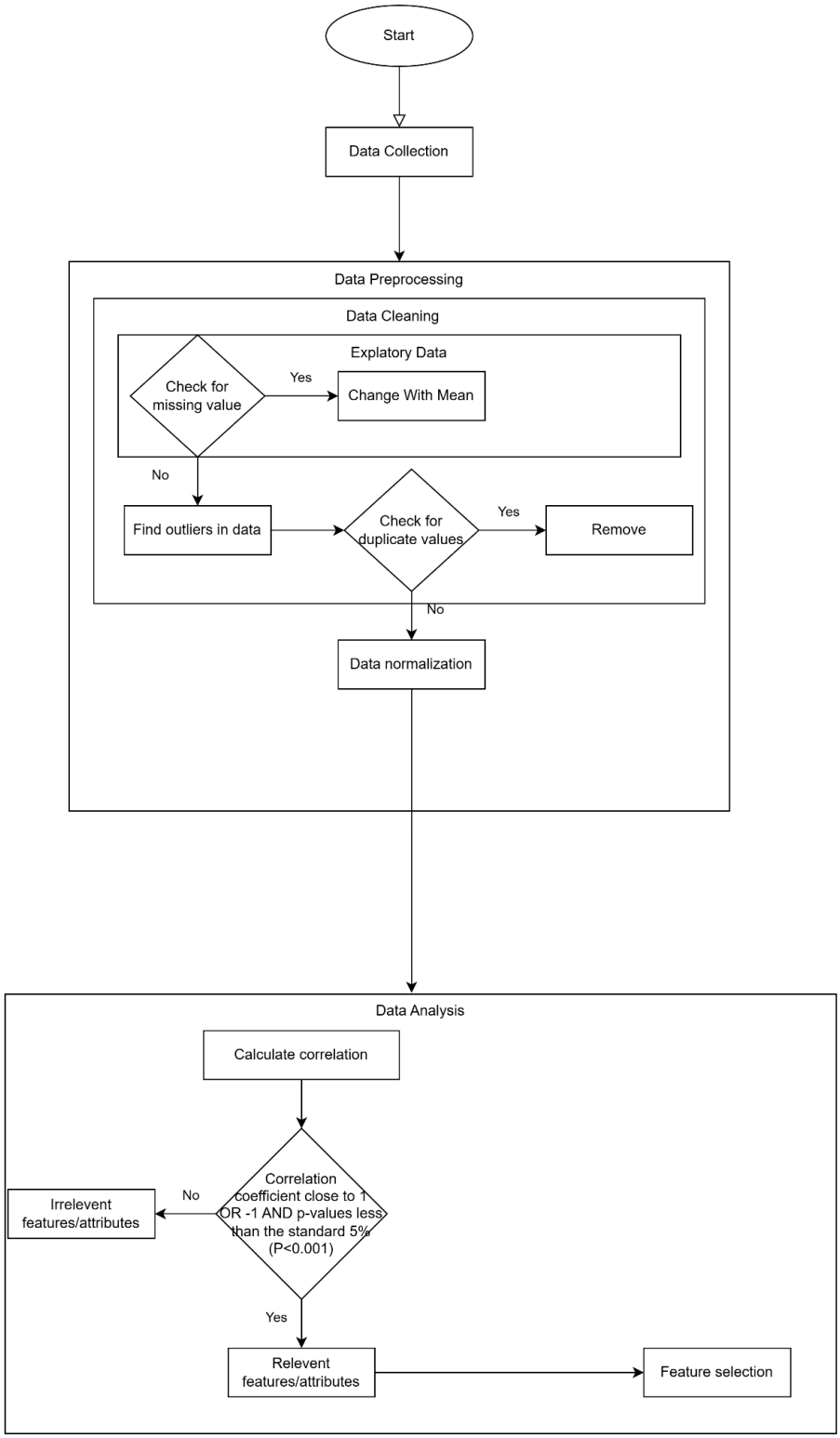
We did not perform the indicator variable steps during our data preprocessing because our dataset did not have data with categorical values. We find out that the indicator variables are typically done for categorical data types such as nominal and ordinal that then can be classified into 1 or 0.

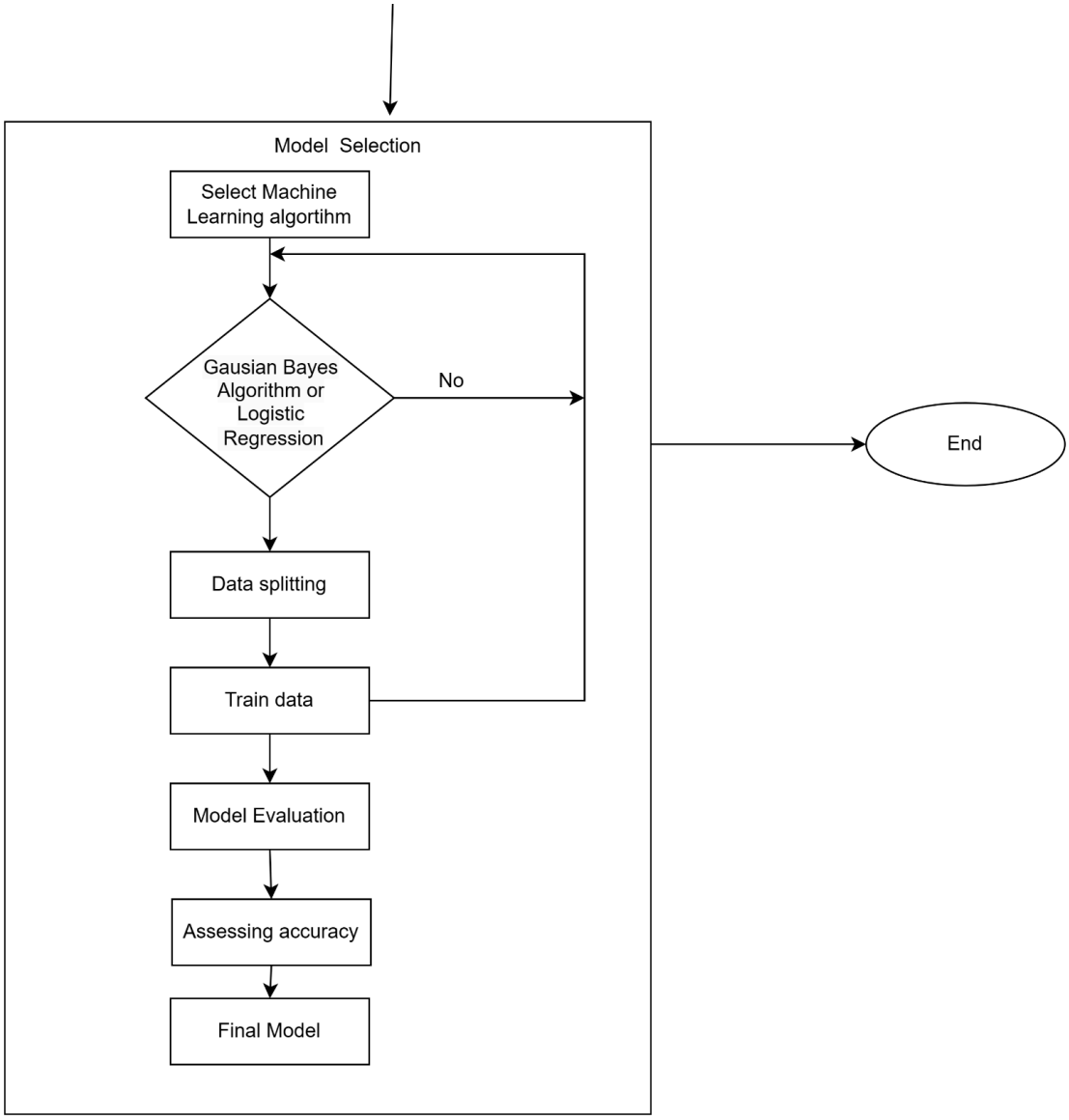
2.3 Software and hardware requirements

For this project focusing on analyzing heart attack detection through various classification algorithms, several software and hardware are needed to handle data analysis, machine learning model development, and experimentation. Software requirements for this research project are Python, R, Visual Studio Code (VS Code), Git, Microsoft Word and Excel versions 2309, Matplotlib and Seaborn for Python. Python is used for data analysis and preprocessing due to its rich ecosystem of data science libraries. For machine learning and model development R language is used because it has libraries such as “caret” for clarification tasks. Next, VS Code is used for supporting multiple programming languages. Besides, Git is required to track changes and collaborate with team members efficiently. Matplotlib and Seaborn libraries are used for data visualization in Python. Other than that, Microsoft Word and Excel versions 2309 are required for data preparation and analyzing.

Besides, the hardware required for this research include a computer with an Intel Core i3 processor, the Windows 11 operating system, 8GB of RAM, and a 222 GB SSD. A modern multi-core CPU is essential in this project for running machine learning algorithms efficiently. Next, having a substantial amount of RAM is crucial when working with large datasets. High-capacity and fast storage, such as an SSD, is essential for storing datasets, code, and model checkpoints.

3.0 Flowchart of the proposed approach




Model Selection
Select Machine Learning algortihm
Gausian Bayes Algorithm or Logistic Regression
No
Data splitting
Train data
Model Evaluation
Assessing accuracy
Final Model
End

3.1 Exploratory data analysis

Descriptive statistic

Descriptive statistics typically involve calculating various measures to describe the data's central tendency, variability, and distribution. By transposing the DataFrame using transpose(), you'll be able to see all columns with their respective statistics.

 # Display basic statistics of the dataset

summary_statistics = df.describe().transpose()

print(summary_statistics)

```
[303 rows x 14 columns]
          count        mean        std  ...    50%    75%    max
age       303.0   54.366337   9.082101  ...   55.0   61.0   77.0
sex       303.0    0.683168   0.466011  ...    1.0    1.0    1.0
cp        303.0    0.966997   1.032052  ...    1.0    2.0    3.0
trtbps    303.0  131.623762  17.538143  ...  130.0  140.0  200.0
chol      303.0  246.264026  51.830751  ...  240.0  274.5  564.0
fbs       303.0    0.148515   0.356198  ...    0.0    0.0    1.0
restecg   303.0    0.528053   0.525860  ...    1.0    1.0    2.0
thalachh  303.0  149.646865  22.905161  ...  153.0  166.0  202.0
exng      303.0    0.326733   0.469794  ...    0.0    1.0    1.0
oldpeak   303.0    1.039604   1.161075  ...    0.8    1.6    6.2
slp       303.0    1.399340   0.616226  ...    1.0    2.0    2.0
caa       303.0    0.729373   1.022606  ...    0.0    1.0    4.0
thall     303.0    2.313531   0.612277  ...    2.0    3.0    3.0
output    303.0    0.544554   0.498835  ...    1.0    1.0    1.0
```

Basic Grouping

The basics of grouping in data analysis involve dividing a dataset into groups based on one or more categorical variables and performing operations independently on each group. So, since the objective is to compare the performance of the machine learning models (Gaussian Naïve Bayes and Logistic Regression) in terms of accuracy, precision, recall, and F1-score for heart attack detection we are grouping by the output(0= less chance of heart attack while 1= more chance of heart attack). This could help analyze the model's performance in distinguishing what factors led to getting a heart attack.

grouped_data = df.groupby(output).mean()

print(grouped_data)

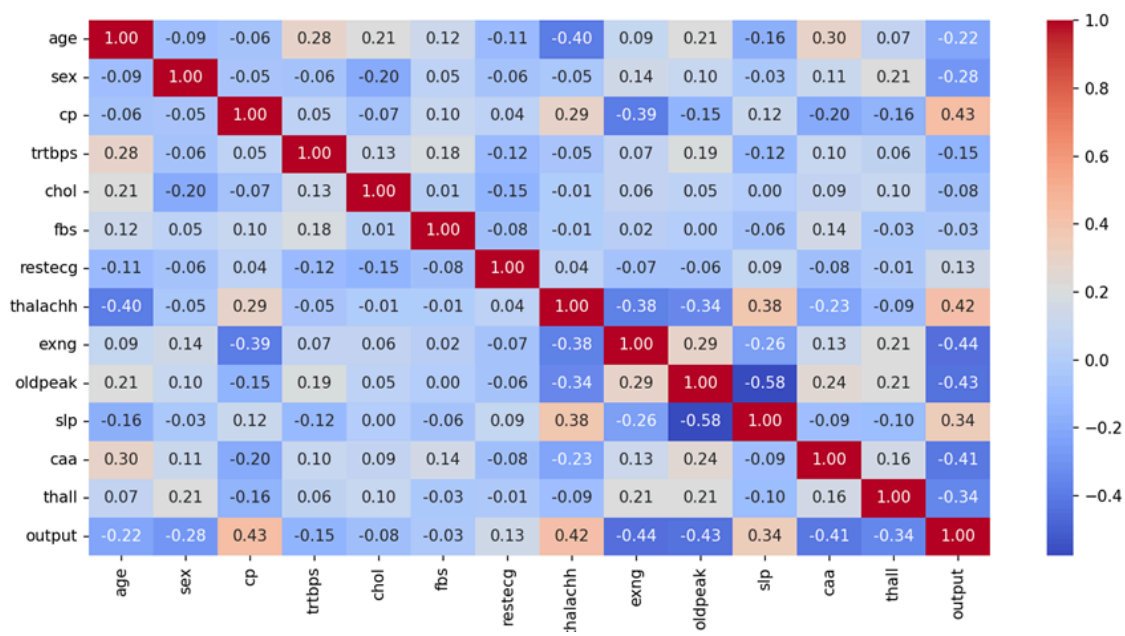
```
            age       sex       cp     trtbps       chol      fbs  ...  thalachh     exng  oldpeak      slp     caa    thall
output                                                              ...
0     56.601449  0.826087  0.478261  134.398551  251.086957  0.159420  ...  139.101449  0.550725  1.585507  1.166667  1.166667  2.543478
1     52.496970  0.563636  1.375758  129.303030  242.230303  0.139394  ...  158.466667  0.139394  0.583030  1.593939  0.363636  2.121212
```

So for further explanation, we can see the average age for a chance of getting a heart attack is 52.49 while the average age for not getting a heart attack is 56.60.

As our data is numerical and our objective is to compare Linear Regression and Gaussian Naïve Bayes, we will focus on using Pearson collection instead of the ANOVA test.

Correlation Matrix with Numbers Heatmap

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f",
linewidths=0.5)
plt.show()
```



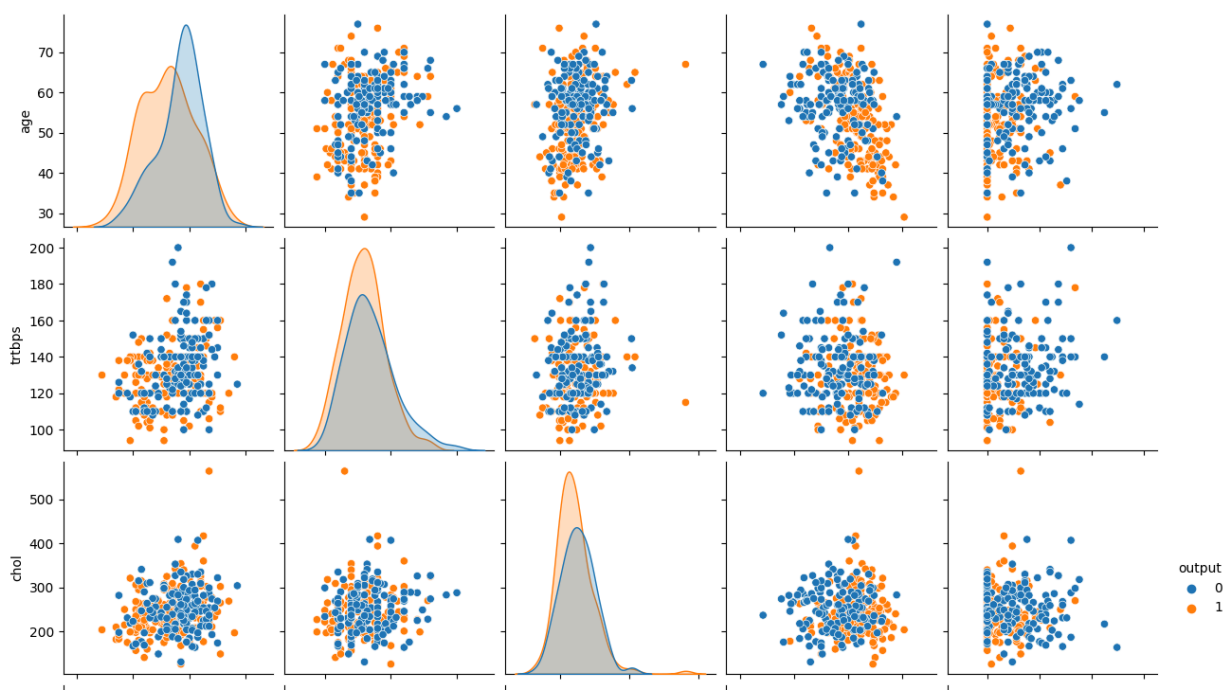
Numeric Feature Analysis

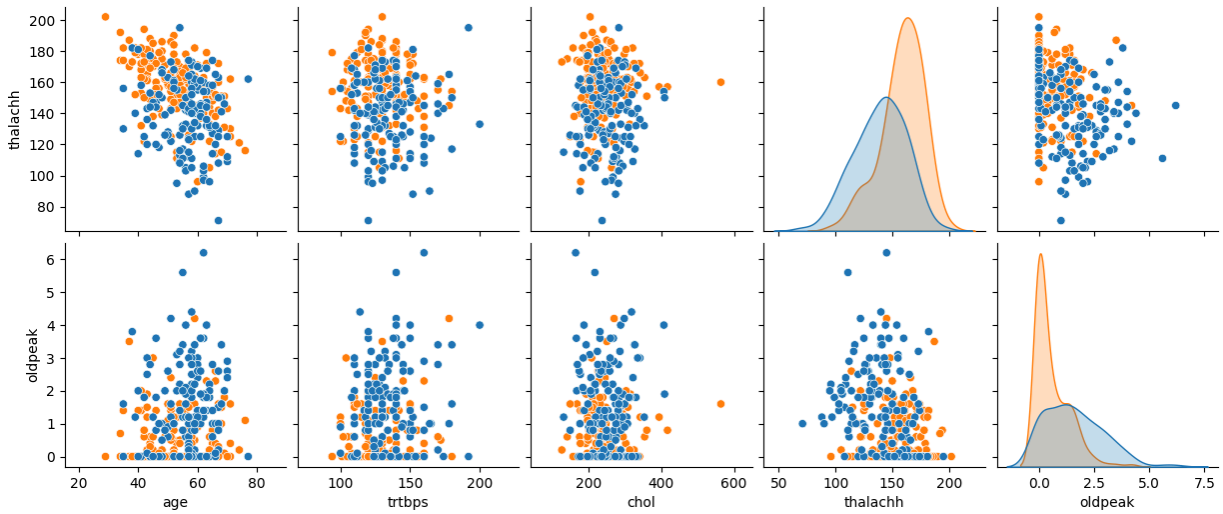
➔ Bivariate data analysis with scatter plot

Pairplot allows us to plot pairwise relationships between variables within a dataset. This creates a nice visualization and helps us understand the data by summarizing a large amount of data in a single figure.

```
numeric_list = ["age", "trtbps","chol","thalachh","oldpeak","output"]
```

```
df_numeric = df.loc[:, numeric_list]
sns.pairplot(df_numeric, hue = "output", diag_kind = "kde")
plt.show()
```





The dataset shows the relationships between pairs of six numeric variables: age, trtbps (resting blood pressure), chol (cholestoral), thalachh (maximum heart rate achieved), oldpeak (ST segment depression), and output (the target variable, which is 0 for less chance of heart attack and 1 for more chance of heart attack).

Each row and column of the plot corresponds to one of the numeric variables. The diagonal plots in the middle of the chart are density plots (kde plots) that show the distribution of each variable. The off-diagonal plots are scatter plots that show the relationship between each pair of variables. The color of the points in the scatter plots corresponds to the target variable (output), with red indicating a higher chance of a heart attack and blue indicating a lower chance.

The observations from the pairplot are as follows:

- Age vs. trtbps: There is a weak positive correlation, suggesting that as age increases, resting blood pressure tends to slightly increase as well.
- Age vs. thalachh: There is a weak negative correlation, indicating that as age increases, maximum heart rate tends to slightly decrease.
- trtbps vs. thalachh: There is a weak negative correlation, suggesting that individuals with higher resting blood pressure tend to have lower maximum heart rates.
- thalachh vs. oldpeak: There is a weak positive correlation, indicating that individuals with higher maximum heart rates tend to have slightly higher ST segment depression.
- chol vs. oldpeak: There is a weak positive correlation, suggesting that individuals with higher cholesterol tend to have slightly higher ST segment depression.

Pearson Correlation

The Pearson Correlation measures the linear dependence between two variables X and Y. The resulting coefficient is a value between -1 and 1 inclusive, where:

- 1: Total positive linear correlation.
- 0: No linear correlation, the two variables most likely do not affect each other.
- -1: Total negative linear correlation.

Pearson Correlation is the default method of the function "corr"

numeric_columns = df.select_dtypes(include=['float64', 'int64'])

correlation_matrix = numeric_colums.corr()

print(correlation_matrix)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

Correlation matrix
               age       sex        cp   trtbps      chol       fbs   restecg  thalachh      exng   oldpeak       slp       caa     thall    output
age       1.000000 -0.094962 -0.063107  0.283121  0.207216  0.119492 -0.111590 -0.395235  0.093216  0.206040 -0.164124  0.302261  0.065317 -0.221476
sex      -0.094962  1.000000 -0.051740 -0.057647 -0.195571  0.046022 -0.060351 -0.046439  0.143460  0.098322 -0.032990  0.113060  0.211452 -0.283609
cp       -0.063107 -0.051740  1.000000  0.046486 -0.072682  0.096018  0.041561  0.293367 -0.392937 -0.146692  0.116854 -0.195356 -0.160370  0.432080
trtbps    0.283121 -0.057647  0.046486  1.000000  0.125256  0.178125 -0.115367 -0.048023  0.068526  0.194600 -0.122873  0.099248  0.062870 -0.146269
chol      0.207216 -0.195571 -0.072682  0.125256  1.000000  0.011428 -0.147602 -0.005308  0.064099  0.050086  0.000417  0.086878  0.096810 -0.081437
fbs       0.119492  0.046022  0.096018  0.178125  0.011428  1.000000 -0.083081 -0.007169  0.024729  0.004514 -0.058654  0.144935 -0.032752 -0.026826
restecg  -0.111590 -0.060351  0.041561 -0.115367 -0.147602 -0.083081  1.000000  0.041210 -0.068807 -0.056251  0.090402 -0.083112 -0.010473  0.134874
thalachh -0.395235 -0.046439  0.293367 -0.048023 -0.005308 -0.007169  0.041210  1.000000 -0.377411 -0.342201  0.384754 -0.228311 -0.094910  0.419955
exng      0.093216  0.143460 -0.392937  0.068526  0.064099  0.024729 -0.068807 -0.377411  1.000000  0.286766 -0.256106  0.125377  0.205826 -0.435601
oldpeak   0.206040  0.098322 -0.146692  0.194600  0.050086  0.004514 -0.056251 -0.342201  0.286766  1.000000 -0.576314  0.236560  0.209090 -0.429146
slp      -0.164124 -0.032990  0.116854 -0.122873  0.000417 -0.058654  0.090402  0.384754 -0.256106 -0.576314  1.000000 -0.092236 -0.103314  0.343940
caa       0.302261  0.113060 -0.195356  0.099248  0.086878  0.144935 -0.083112 -0.228311  0.125377  0.236560 -0.092236  1.000000  0.160085 -0.408992
thall     0.065317  0.211452 -0.160370  0.062870  0.096810 -0.032752 -0.010473 -0.094910  0.205826  0.209090 -0.103314  0.160085  1.000000 -0.343101
output   -0.221476 -0.283609  0.432080 -0.146269 -0.081437 -0.026826  0.134874  0.419955 -0.435601 -0.429146  0.343940 -0.408992 -0.343101  1.000000
PS C:\Users\kamil>
```

P-value:

The P-value is the probability value that the correlation between these two variables is statistically significant. Normally, we choose a significance level of 0.05, which means that we are 95% confident that the correlation between the variables is significant.

By convention, when

- the p-value is <0.001: we say there is strong evidence that the correlation is significant.
- the p-value is < 0.05: there is moderate evidence that the correlation is significant.
- the 0.05 < p-value is < 0.1: there is weak evidence that the correlation is significant.
- the p-value is > 0.1: there is no evidence that the correlation is significant.

Here we compare the p-value of the output (classification of getting a heart attack) with other features

```python
from scipy import stats

pearson_coef, p_value = stats.pearsonr(df['age'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'age' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
pearson_coef, p_value = stats.pearsonr(df['sex'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'sex' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
pearson_coef, p_value = stats.pearsonr(df['cp'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'cp' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
pearson_coef, p_value = stats.pearsonr(df['trtbps'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'trtbs' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
pearson_coef, p_value = stats.pearsonr(df['chol'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'chol' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
pearson_coef, p_value = stats.pearsonr(df['fbs'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'fbs' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
pearson_coef, p_value = stats.pearsonr(df['restecg'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'restecg' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
pearson_coef, p_value = stats.pearsonr(df['thalachh'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'thalachh' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
```

```python
pearson_coef, p_value = stats.pearsonr(df['exng'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'exng' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
pearson_coef, p_value = stats.pearsonr(df['oldpeak'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'oldpeak' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
pearson_coef, p_value = stats.pearsonr(df['slp'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'slp' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
pearson_coef, p_value = stats.pearsonr(df['caa'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'caa' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
print()
pearson_coef, p_value = stats.pearsonr(df['thall'], df['output'])
print("The Pearson Correlation Coefficient and P-value of 'thall' and 'output'")
print( "The Pearson Correlation Coefficient is", pearson_coef, " with a P-value of P = ", p_value)
```

Result:

```
The Pearson Correlation Coefficient and P-value of 'age' and 'output'
The Pearson Correlation Coefficient is -0.221475827766562  with a P-value of P =  0.00010394837285416845

The Pearson Correlation Coefficient and P-value of 'sex' and 'output'
The Pearson Correlation Coefficient is -0.28360935779586227  with a P-value of P =  5.402435780432229e-07

The Pearson Correlation Coefficient and P-value of 'cp' and 'output'
The Pearson Correlation Coefficient is 0.43207959156640785  with a P-value of P =  3.6273838772544235e-15

The Pearson Correlation Coefficient and P-value of 'trtbs' and 'output'
The Pearson Correlation Coefficient is -0.1462686638415544  with a P-value of P =  0.010926538861949197

The Pearson Correlation Coefficient and P-value of 'chol' and 'output'
The Pearson Correlation Coefficient is -0.08143720051844142  with a P-value of P =  0.15803697464249777

The Pearson Correlation Coefficient and P-value of 'fbs' and 'output'
The Pearson Correlation Coefficient is -0.026825970565970136  with a P-value of P =  0.6424070490676519

The Pearson Correlation Coefficient and P-value of 'restecg' and 'output'
The Pearson Correlation Coefficient is 0.13487444702864623  with a P-value of P =  0.019033607668275408

The Pearson Correlation Coefficient and P-value of 'thalachh' and 'output'
The Pearson Correlation Coefficient is 0.4199550436638698  with a P-value of P =  2.4761460479236786e-14

The Pearson Correlation Coefficient and P-value of 'exng' and 'output'
The Pearson Correlation Coefficient is -0.4356007617136187  with a P-value of P =  2.04646758906984e-15

The Pearson Correlation Coefficient and P-value of 'oldpeak' and 'output'
The Pearson Correlation Coefficient is -0.42914583288673835  with a P-value of P =  5.814566948031603e-15

The Pearson Correlation Coefficient and P-value of 'slp' and 'output'
The Pearson Correlation Coefficient is 0.3439395324893869  with a P-value of P =  8.221388831030162e-10

The Pearson Correlation Coefficient and P-value of 'caa' and 'output'
The Pearson Correlation Coefficient is -0.40899197975692636  with a P-value of P =  1.3173455415475193e-13

The Pearson Correlation Coefficient and P-value of 'thall' and 'output'
The Pearson Correlation Coefficient is -0.34310071238956447  with a P-value of P =  9.089044024818302e-10
PS C:\Users\kamil>
```

'age' and 'output':

- Correlation: -0.2215 (moderate negative correlation)
- P-value: 0.0001 (highly significant)

'sex' and 'output':

- Correlation: -0.2836 (moderate negative correlation)
- P-value: 5.4024e-07 (highly significant)

'cp' and 'output':

- Correlation: 0.4321 (strong positive correlation)
- P-value: 3.6274e-15 (highly significant)

'trtbs' and 'output':

- Correlation: -0.1463 (weak negative correlation)
- P-value: 0.0109 (significant)

'chol' and 'output':

- Correlation: -0.0814 (weak negative correlation)
- P-value: 0.1580 (not significant)

'fbs' and 'output':

- Correlation: -0.0268 (very weak negative correlation)
- P-value: 0.6424 (not significant)

'restecg' and 'output':

- Correlation: 0.1349 (weak positive correlation)
- P-value: 0.0190 (not significant)

'thalachh' and 'output':

- Correlation: 0.4200 (strong positive correlation)
- P-value: 2.4761e-14 (highly significant)

'exng' and 'output':

- Correlation: -0.4356 (strong negative correlation)
- P-value: 2.0465e-15 (highly significant)

'oldpeak' and 'output':

- Correlation: -0.4291 (strong negative correlation)
- P-value: 5.8146e-15 (highly significant)

'slp' and 'output':

- Correlation: 0.3439 (moderate positive correlation)
- P-value: 8.2214e-10 (highly significant)

'caa' and 'output':

- Correlation: -0.4090 (strong negative correlation)
- P-value: 1.3173e-13 (highly significant)

'thall' and 'output':

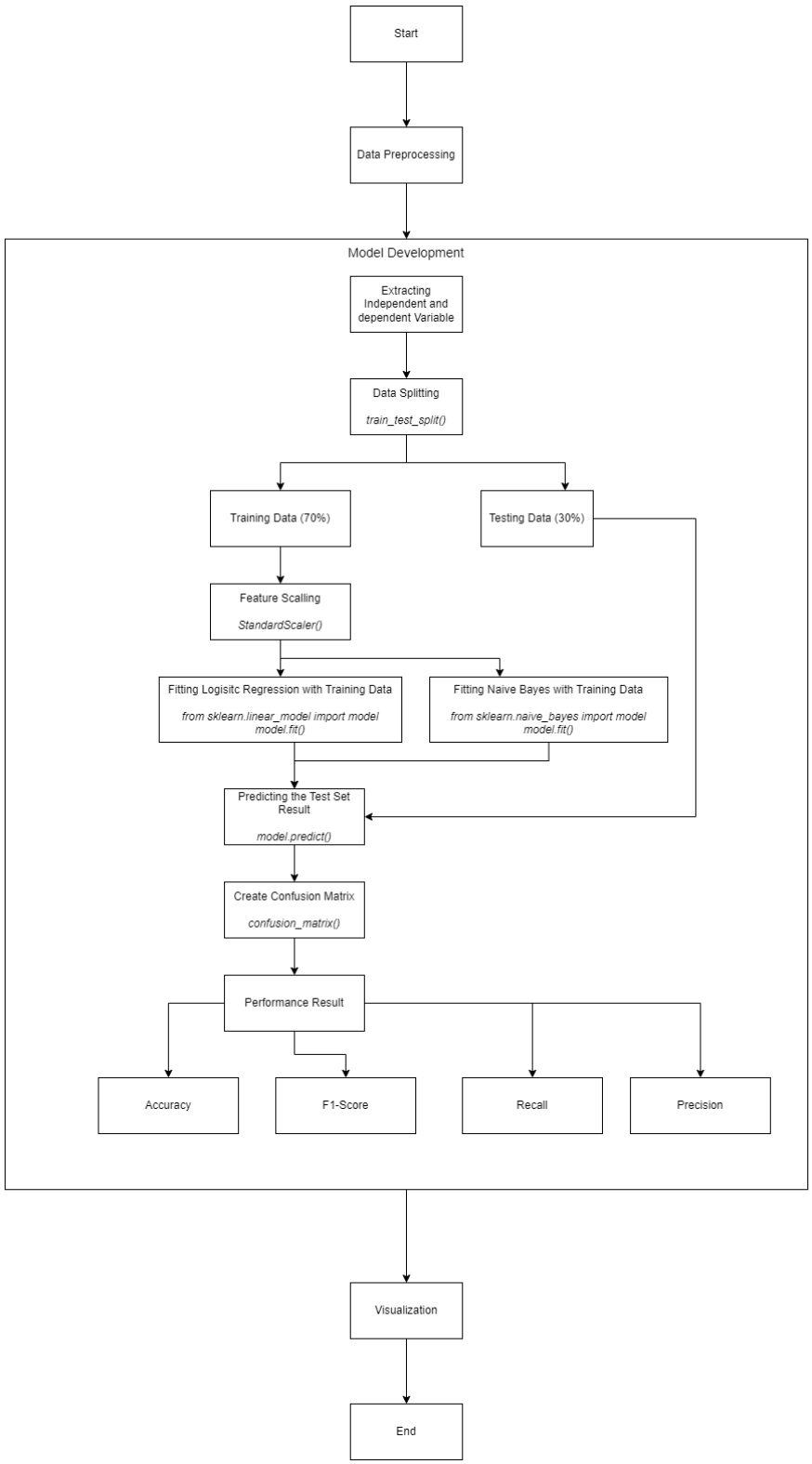
- Correlation: -0.3431 (moderate negative correlation)
- P-value: 9.0890e-10 (highly significant)

We now have a better idea of what our data looks like and which variables are important to take into account when predicting heart attack detection. We have narrowed it down to the following variables:

- age
- sex
- cp
- thalachh
- exng
- oldpeak
- slp
- caa
- thall

As we now move into building machine learning models to automate our analysis, feeding the model with variables that meaningfully affect our target variable will improve our model's prediction performance.

3.2 Model Development



```
# Assuming 'output' is your target column
output_column = 'output'

# List the features that are considered significant (excluding 'output')
significant_features = list(df.columns[:-1])  # Exclude the last column ('output')
print("Significant Features (excluding 'output'):", significant_features)

# Separate independent variables (features) and dependent variable (target)
X = df[significant_features]  # Features (excluding 'output')
y = df[output_column]  # Target variable
```

Here we imported the python libraries and update dataset which only remain columns that have p-value < 0.001(features relevant) where it means the correlation is likely significant.The code removes the insignificant columns from the dataset, keeping only the ones that are likely to be helpful for understanding or predicting the target Next,we separate the independent variable to be x-axis, and dependent variable ‘output’ column to be y-axis.

Logistic Regression:

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a pipeline with StandardScaler and Logistic Regression
model = make_pipeline(StandardScaler(), LogisticRegression())
```

Then, split the data into train (70%) and test (30%) because our data is not complex and massive. So 70/30 is the right choice to avoid overfitting. After that, scale the independent variable using Standard Scaler.This involves transforming the values of each feature so that they have a mean of 0 and a standard deviation of 1.This is often important for machine learning algorithms because it helps them work more effectively with features that have different scales

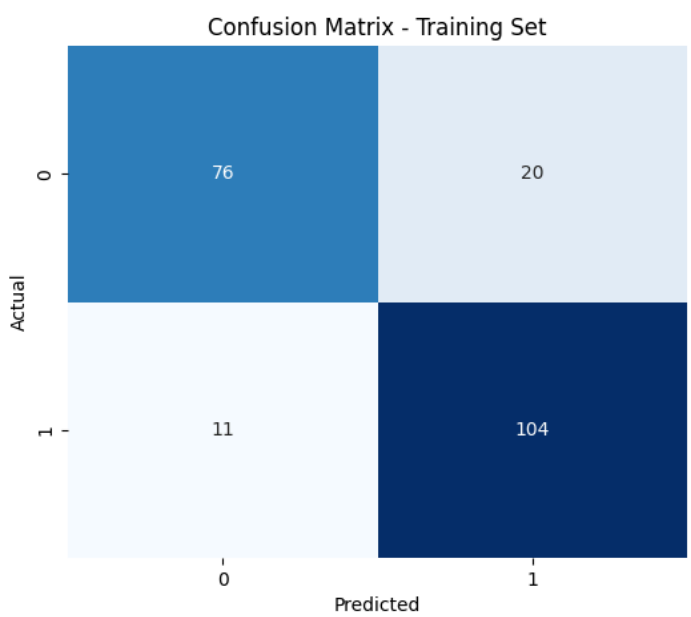
Training data for logistic regression:

```
# Fit the model on the training set
model.fit(X_train, y_train)
# Make predictions on the training set
y_train_pred = model.predict(X_train)

# Calculate accuracy score on the training set
accuracy_train = accuracy_score(y_train, y_train_pred)
print(f"Accuracy Score on Training Set: {accuracy_train:.2f}")
conf_matrix = confusion_matrix(y_train, y_train_pred)
print("Confusion Matrix:\n", conf_matrix)

# Visualize confusion matrix for training set
plt.figure(figsize=(6, 5))
sns.heatmap(cm_train, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title('Confusion Matrix - Training Set')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Accuracy Score on Training Set: 0.85
Confusion Matrix:
 [[ 76  20]
 [ 11 104]]
```

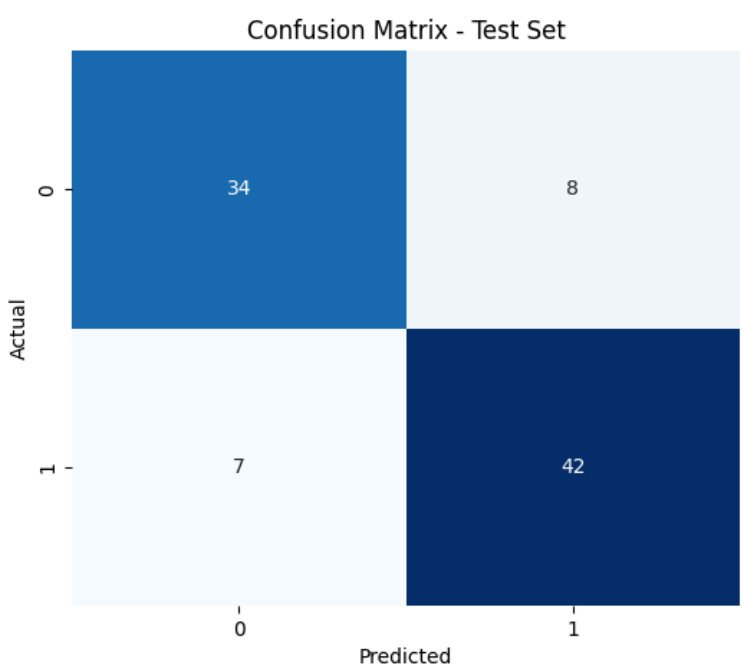


Testing data for Logistic Regression:

```
# Make predictions on the test set
y_test_pred = model.predict(X_test)

# Calculate accuracy score on the test set
accuracy_test = accuracy_score(y_test, y_test_pred)
# Generate confusion matrix for test set
cm_test = confusion_matrix(y_test, y_test_pred)
# Visualize confusion matrix for test set
plt.figure(figsize=(6, 5))
sns.heatmap(cm_test, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title('Confusion Matrix - Test Set')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Accuracy Score on Test Set: 0.84
Confusion Matrix:
 [[34  8]
 [ 7 42]]
```



Naive Bayes:

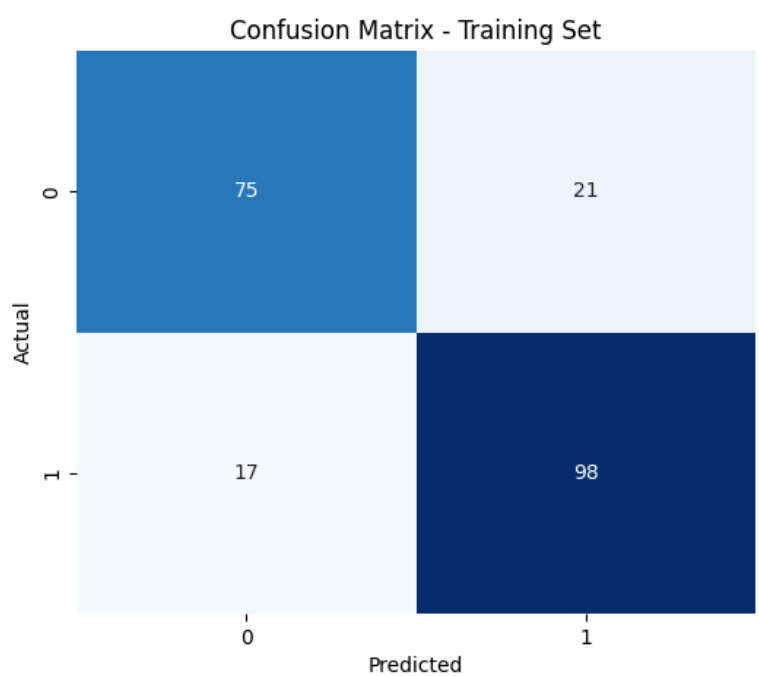
```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Create a pipeline with StandardScaler and Naive Bayes
model = make_pipeline(StandardScaler(), GaussianNB())  # Replace LogisticRegression with GaussianNB
```

Then, split the data into train (70%) and test (30%) because our data is not complex and massive. So 70/30 is the right choice to avoid overfitting. After that, scale the independent variable using Standard Scaler.This involves transforming the values of each feature so that they have a mean of 0 and a standard deviation of 1.This is often important for machine learning algorithms because it helps them work more effectively with features that have different scales

Training set for Gaussian Naive Bayes:

```
# Fit the model on the training set
model.fit(X_train, y_train)
# Make predictions on the training set
y_train_pred = model.predict(X_train)
# Calculate accuracy score on the training set
accuracy_train = accuracy_score(y_train, y_train_pred)
# Generate confusion matrix for training set
cm_train = confusion_matrix(y_train, y_train_pred)
# Visualize confusion matrix for training set
plt.figure(figsize=(6, 5))
sns.heatmap(cm_train, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title('Confusion Matrix - Training Set')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

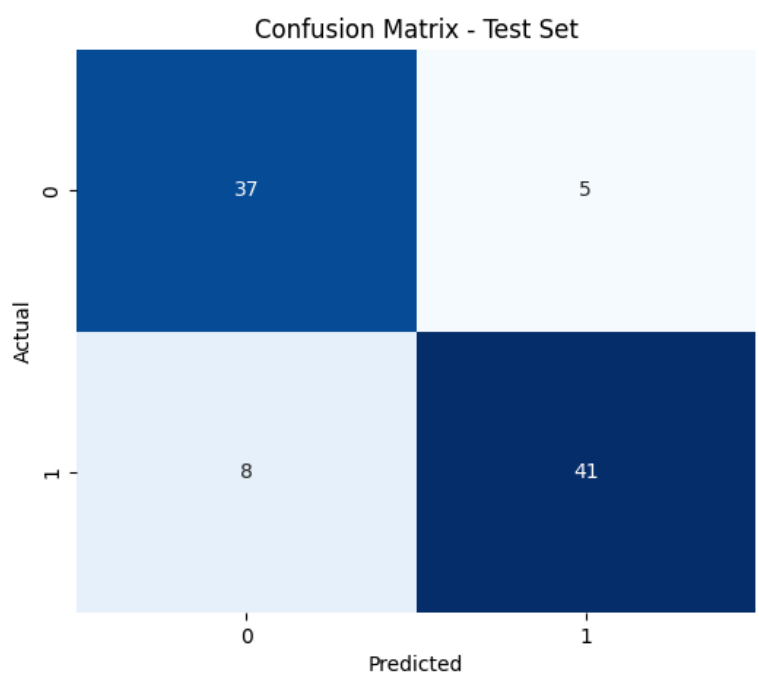
```
Accuracy Score on Training Set: 0.82
Confusion Matrix:
 [[75 21]
 [17 98]]
Classification Report on Training Set:
```



Testing Set for Gaussian Naive Bayes:

```
# Make predictions on the test set
y_test_pred = model.predict(X_test)
# Calculate accuracy score on the test set
accuracy_test = accuracy_score(y_test, y_test_pred)
# Generate confusion matrix for test set
cm_test = confusion_matrix(y_test, y_test_pred)
# Visualize confusion matrix for test set
plt.figure(figsize=(6, 5))
sns.heatmap(cm_test, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title('Confusion Matrix - Test Set')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Accuracy Score on Test Set: 0.86
Confusion Matrix:
 [[37  5]
 [ 8 41]]
```



3.3 Model evaluation

Logistic Regression:

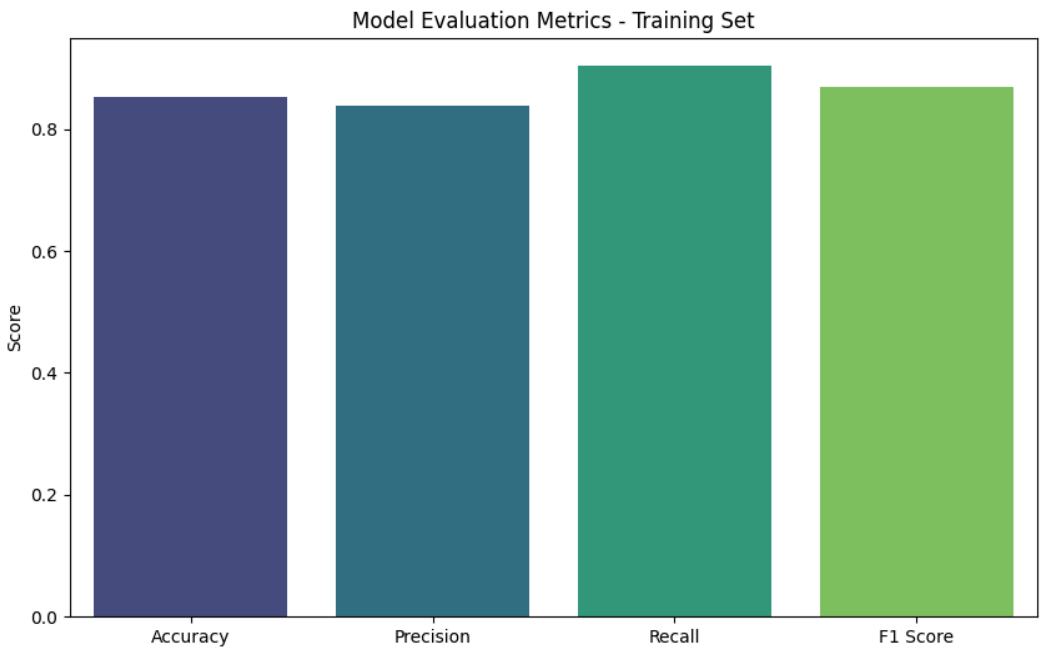
Training Set:

```
# Calculate precision, recall, and F1 score for the training set
precision_train = precision_score(y_train, y_train_pred)
recall_train = recall_score(y_train, y_train_pred)
f1_train = f1_score(y_train, y_train_pred)
# Generate classification report for training set
report_train = classification_report(y_train, y_train_pred)
print("Classification Report - Training Set:\n", report_train)
# Visualize metrics for the training set
metrics_data_train = {'Accuracy': accuracy_train, 'Precision': precision_train, 'Recall':
recall_train, 'F1 Score': f1_train}
metrics_df_train = pd.DataFrame(metrics_data_train, index=['Training Set'])
# Plot metrics for the training set
plt.figure(figsize=(10, 6))
sns.barplot(data=metrics_df_train, palette="viridis")
plt.title('Model Evaluation Metrics - Training Set')
plt.ylabel('Score')
plt.show()
```

```
Classification Report on Training Set:
              precision    recall  f1-score   support

           0       0.87      0.79      0.83        96
           1       0.84      0.90      0.87       115

    accuracy                           0.85       211
   macro avg       0.86      0.85      0.85       211
weighted avg       0.85      0.85      0.85       211
```



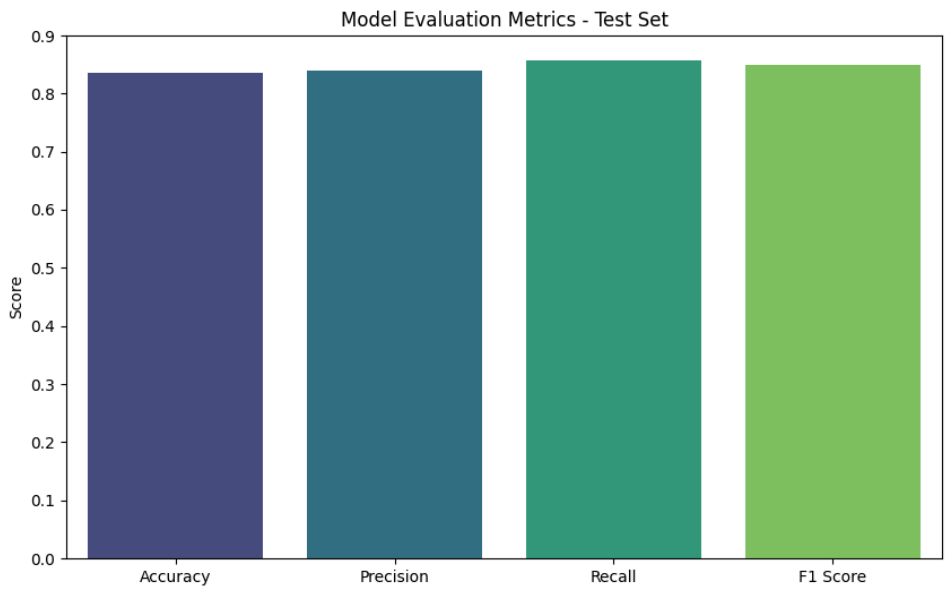
Test Set:

```
# Calculate precision, recall, and F1 score for the test set
precision_test = precision_score(y_test, y_test_pred)
recall_test = recall_score(y_test, y_test_pred)
f1_test = f1_score(y_test, y_test_pred)
# Generate classification report for test set
report_test = classification_report(y_test, y_test_pred)
print("Classification Report - Test Set:\n", report_test)
# Visualize metrics for the test set
metrics_data_test = {'Accuracy': accuracy_test, 'Precision': precision_test, 'Recall': recall_test,
'F1 Score': f1_test}
metrics_df_test = pd.DataFrame(metrics_data_test, index=['Test Set'])
# Plot metrics for the test set
plt.figure(figsize=(10, 6))
sns.barplot(data=metrics_df_test, palette="viridis")
plt.title('Model Evaluation Metrics - Test Set')
plt.ylabel('Score')
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.81      0.82        42
           1       0.84      0.86      0.85        49

    accuracy                           0.84        91
   macro avg       0.83      0.83      0.83        91
weighted avg       0.84      0.84      0.84        91
```



Gaussian Naive Bayes:

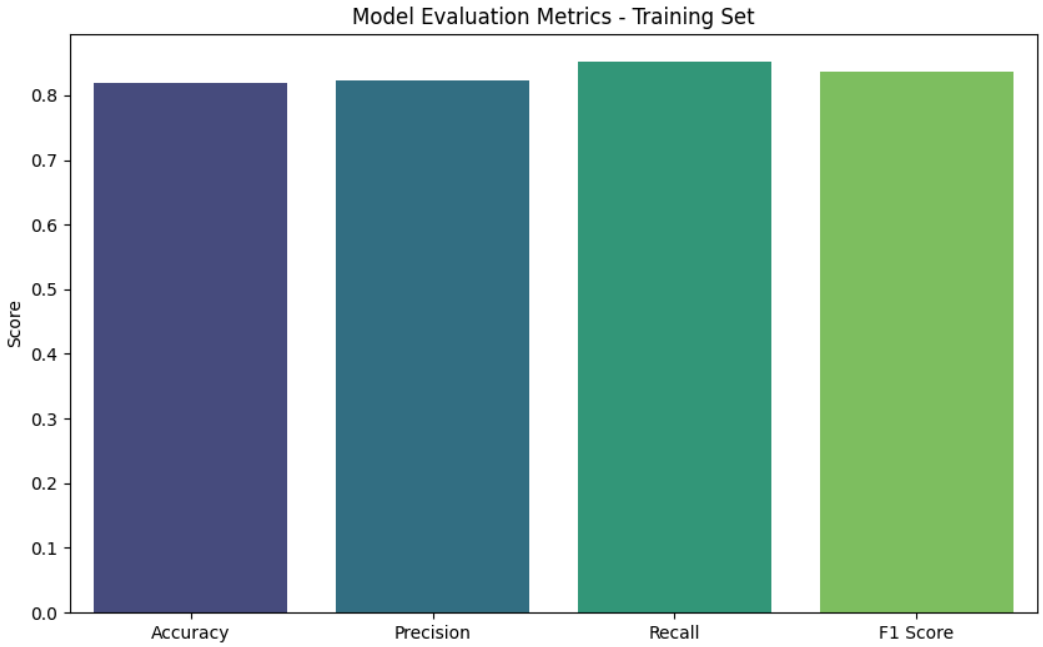
Training set:

```
# Calculate precision, recall, and F1 score for the training set
precision_train = precision_score(y_train, y_train_pred)
recall_train = recall_score(y_train, y_train_pred)
f1_train = f1_score(y_train, y_train_pred)
# Generate classification report for training set
report_train = classification_report(y_train, y_train_pred)
print("Classification Report - Training Set:\n", report_train)
# Visualize metrics for the training set
metrics_data_train = {'Accuracy': accuracy_train, 'Precision': precision_train, 'Recall':
recall_train, 'F1 Score': f1_train}
metrics_df_train = pd.DataFrame(metrics_data_train, index=['Training Set'])
# Plot metrics for the training set
plt.figure(figsize=(10, 6))
sns.barplot(data=metrics_df_train, palette="viridis")
plt.title('Model Evaluation Metrics - Training Set')
plt.ylabel('Score')
plt.show()
```

```
[17 98]]
Classification Report on Training Set:
              precision    recall  f1-score   support

           0       0.82      0.78      0.80        96
           1       0.82      0.85      0.84       115

    accuracy                           0.82       211
   macro avg       0.82      0.82      0.82       211
weighted avg       0.82      0.82      0.82       211
```



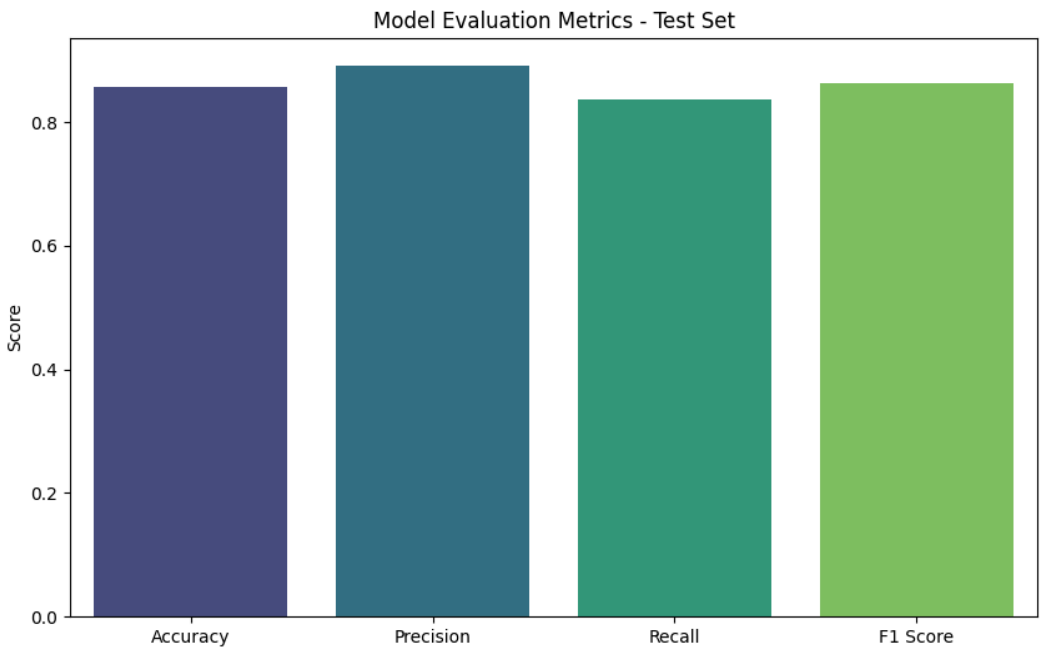
Test Set:

```
# Calculate precision, recall, and F1 score for the test set
precision_test = precision_score(y_test, y_test_pred)
recall_test = recall_score(y_test, y_test_pred)
f1_test = f1_score(y_test, y_test_pred)
# Generate classification report for test set
report_test = classification_report(y_test, y_test_pred)
print("Classification Report - Test Set:\n", report_test)
# Visualize metrics for the test set
metrics_data_test = {'Accuracy': accuracy_test, 'Precision': precision_test, 'Recall': recall_test,
'F1 Score': f1_test}
metrics_df_test = pd.DataFrame(metrics_data_test, index=['Test Set'])
# Plot metrics for the test set
plt.figure(figsize=(10, 6))
sns.barplot(data=metrics_df_test, palette="viridis")
plt.title('Model Evaluation Metrics - Test Set')
plt.ylabel('Score')
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.88      0.85        42
           1       0.89      0.84      0.86        49

    accuracy                           0.86        91
   macro avg       0.86      0.86      0.86        91
weighted avg       0.86      0.86      0.86        91
```



4.0 Testing, and Validation

From the confusion matrix, the true positive (TP) value for Gaussian Naive Bayes is higher than Logistic Regression which shows that Gaussian Naive Bayes has the ability to detect heart attack more accurately than Logistic Regression. Secondly, the false positive (FP) value which is the type I error for Logistic Regression is higher than Gaussian Naive Bayes. It means that we predicted that the person will have a heart attack but the algorithm shows that the person doesn't have a heart attack. So, it shows the poor performance of the algorithm in detecting heart attack. Next, the false negative (FN) value which is the type II error for Gaussian Naive Bayes is higher than Logistic Regression. It means that we predicted that the person didn’t have a heart attack but the algorithm says the person had a heart attack. In the real world, FP value has high impact compared to FN because for FP, the algorithm shows the person doesn't have a heart attack but the person actually has heart attack and for FN, it's vice versa.

The accuracy of Gaussian Naive Bayes is 0.86 whereas for Logistic Regression is 0.84. So, it shows that the accuracy of Gaussian Naive Bayes to detect heart attack is higher than Logistic Regression. The precision of detecting heart attack for a person who has heart attack is higher, which is 0.89 in Gaussian Naive Bayes compared to Logistic Regression which is 0.84. In terms of sensitivity, Logistic Regression has higher sensitivity which is 0.86 compared to Gaussian Naive Bayes which is 0.84 towards predicting the heart attack. In conclusion, the Gaussian Naive Bayes is more accurate in detecting heart attack compared to Logistic Regression.

4.1 Feedback collected from your client(s)

Feedback from Dr Hidayah:

- Overall group 2 shows understanding of steps taken in completing machine learning pipeline, also shows ability to complete the project using python.
- However, they still need to brush up on discussing findings by providing justifications, following the research methodology style.
- Personally, I think this class project would be helpful for them to navigate through the PSM1 project next semester.

5.0 Conclusions

In summary, when comparing Gaussian Naive Bayes and Logistic Regression for detecting heart attacks, Gaussian Naive Bayes proves to be more accurate. It excels in correctly identifying those at risk, while Logistic Regression tends to make more mistakes by falsely predicting heart attacks in individuals without the condition. Although Gaussian Naive Bayes has a slightly higher false negative rate, its overall accuracy and precision are better. In practical terms, this means that Gaussian Naive Bayes is a more reliable choice for accurately spotting potential heart attacks in this project.

6.0 Appendix

Link GitHub Group 2

https://github.com/NiesHW/SECB3203_P4B/tree/main/Group_Project/Group_2

7.0 References

Centers for Disease Control and Prevention. (n.d.). *Heart Disease Facts*. Retrieved From: https://www.cdc.gov/heartdisease/facts.htm

National Heart, Lung, and Blood Institute. (2021). *Cardiovascular Disease Is on the Rise: We Know How to Curb It Because We've Done It.* Retrieved from: https://www.nhlbi.nih.gov/news/2021/cardiovascular-disease-rise-we-know-how-curb-it-weve-done-it

NHS. (2023, July 13). *Treatments for heart attack*. Retrieved From:https://www.nhs.uk/conditions/heart-attack/treatment/

Manjula P, Aravind U R, Darshan M V, Halaswamy M H, Hemanth E, 2022, Heart Attack Prediction Using Machine Learning Algorithms, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) ICEI – 2022 (Volume 10 – Issue 11)

Mohan, S., Thirumalai, C., & Srivastava, G. (2019). *Effective heart disease prediction using hybrid machine learning techniques.* IEEE Access, 7, 81542-81554. Retrieved From: Effective Heart Disease Prediction Using Hybrid Machine Learning Techniques | IEEE Journals & Magazine | IEEE Xplore

M. Rizwan, S. Arshad, H. Aijaz, R. A. Khan and M. Z. U. Haque, "*Heart Attack Prediction using Machine Learning Approach*," 2022 Third International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT), Karachi, Pakistan, 2022, pp. 1-8, doi: 10.1109/INTELLECT55495.2022.9969395.

U.S. Department of Health and Human Services. (2022, March 24). *What is a heart attack?*. National Heart Lung and Blood Institute. Retrieved From:https://www.nhlbi.nih.gov/health/heart-attack

Zhang, W., & Han, J. (2017). *Towards heart sound classification without segmentation using convolutional neural networks. In 2017 Computing in Cardiology (CinC)* (pp. 1-4). Rennes, France. doi: 10.22489/CinC.2017.254-164. Retrieved From: https://ieeexplore.ieee.org/document/8331655