A Prediction task using the Heart failure records using Neural Network

Name: **Sanchayan Vivekananthan**

Date: **21/12/2023**

Abstract

This comprehensive report explores the development of a neural network model for predicting heart disease using the Cleveland Heart Disease dataset. Initial exploratory data analysis, preprocessing, and factor analysis lay the foundation for constructing the neural network. The report addresses key questions regarding data characteristics, handling missing values, outliers, and categorical variables. Factor analysis reveals underlying patterns in the dataset, enhancing interpretability. The subsequent prediction task involves dataset splitting, transformation, and neural network building. Evaluation metrics demonstrate the model's accuracy in heart disease classification. An experimental model introduces architectural variations, and comparative analysis highlights the original model's superior performance. The literature review surveys supervised learning algorithms and emphasized the significance of hybrid models and neural networks in heart disease prediction. Notable studies utilizing diverse neural network architectures for medical predictions are discussed, showcasing their effectiveness. The abstract encapsulates the report's emphasis on robust data analysis, model development, and the exploration of advanced neural network architectures for heart disease prediction.

1 Introduction

This report encompasses a rigorous analysis of the Cleveland Heart Disease dataset for predicting heart disease, with the primary objective of developing a predictive model for assessing the risk of mortality using a neural network (ANN). The dataset, comprising 13 columns, was systematically analysed using exploratory data analysis (EDA) factor analysis to reveal its underlying properties. Initial exploratory data analysis (EDA) identifies patterns, followed by thorough pre-processing for missing values and outliers. After standardisation and factor analysis, a neural network model was constructed using Keras, featuring three hidden layers. The initial two layers comprised five neurons each, while the third layer featured three neurons, collectively aimed at predicting the binary outcome of mortality or survival. Then systematic experiments were conducted to refine model performance by altering key parameters like hidden layer number, size, and activation functions.

Neural network models are computer architectures modelled after the human brain's architecture and operations. Their structure comprises interconnected nodes arranged in layers, with each layer adding to the network's overall design. Layers can be classified into three primary categories: input, hidden, and output. During training, the network can learn from data by adjusting the weights, or connections between nodes.

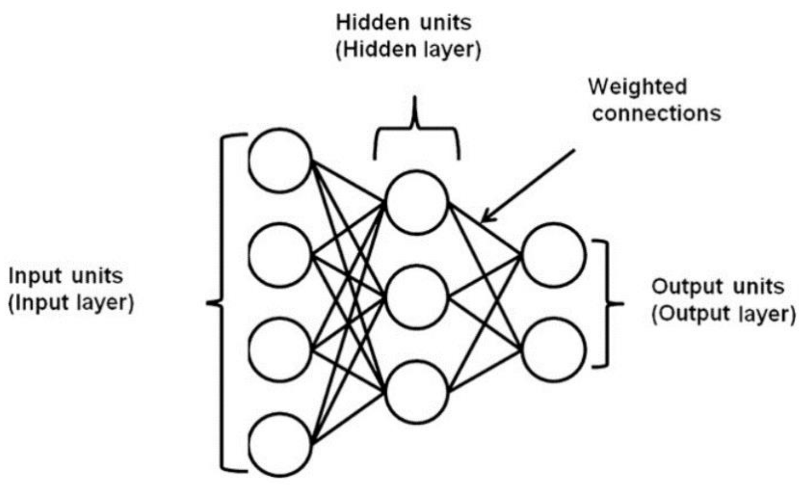


Figure 1: Simple representation of Neural Network with one hidden layer
Source: Stahl, Frederic, and Ivan Jordanov. 'An Overview of the Use of Neural Networks for Data Mining Tasks'. WIREs Data Mining and Knowledge Discovery 2, no. 3 (May 2012): 193–208. https://doi.org/10.1002/widm.1052.

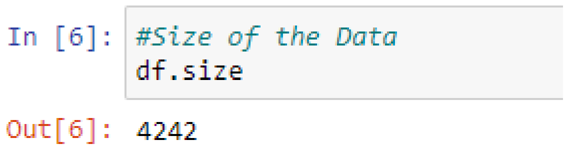
The structure of predictive neural networks involves a complex network of connected nodes that is learned from historical data. In the initial phase, the analysis of historical data is undertaken to determine how known output values can be predicted using given predictor variables. Following this training phase, a testing phase is entered using new data to ensure that adequate predictive power is achieved when confronted with previously unseen information. Once a sufficiently small prediction error is attained, the network is considered ready to accurately predict the future based on what has been learned (Guzman, 2023).

In the realm of healthcare, neural networks are essential for improving diagnostics and patient care by using various data sets. They are crucial for tasks such as diagnosing diseases, predicting risks, customizing drug responses, and selecting participants for clinical trials. Additionally, they play a key role in detecting fraud, monitoring epidemics, and extracting valuable information from electronic health records (EHR). Over the past decade, electronic healthcare records have become widely adopted, resulting in a wealth of patient data. This abundant data is increasingly being utilized, with neural networks applying deep learning techniques to tasks like assessing clinical risks, predicting outcomes, estimating treatment effects, and making recommendations (Chen et al., 2020). The potential of neural networks to offer accurate, timely, and personalized insights for enhancing decision-making in healthcare is underscored by these applications.

2 Questions

2.1 Question 1: Exploratory Data Analysis (EDA)

2.1.1 Size of the data

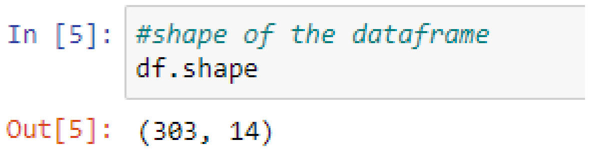


```
In [6]:  #Size of the Data
          df.size

Out[6]:  4242
```

Figure 2: Python code snippet representing the Data Size

Figure 2 indicates that the size of the data is 4242.

2.1.2 Number of rows and columns in the data:

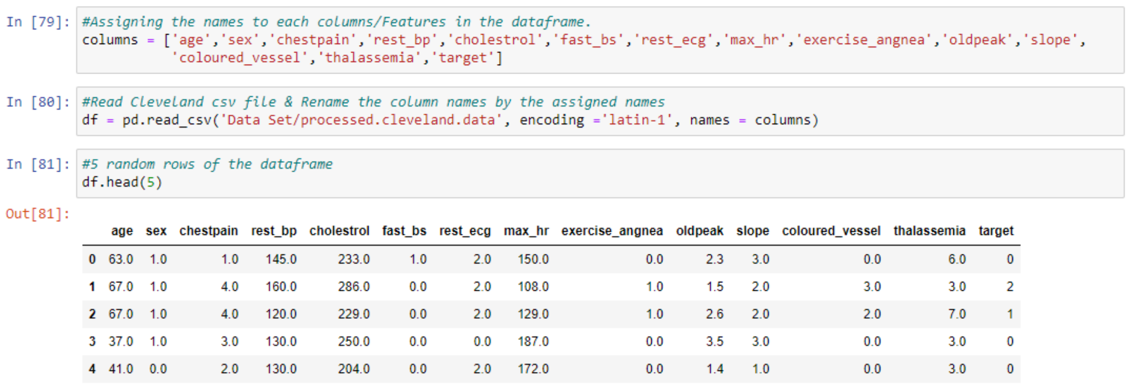


```
In [5]:  #shape of the dataframe
          df.shape

Out[5]:  (303, 14)
```

Figure 3: A Python code snippet showing the shape of the data frame

The shape of (303, 14) suggests that the Data Frame contains 303 observations/rows with data on 14 different variables/columns.

2.1.3 Assigning column names/ Rename.



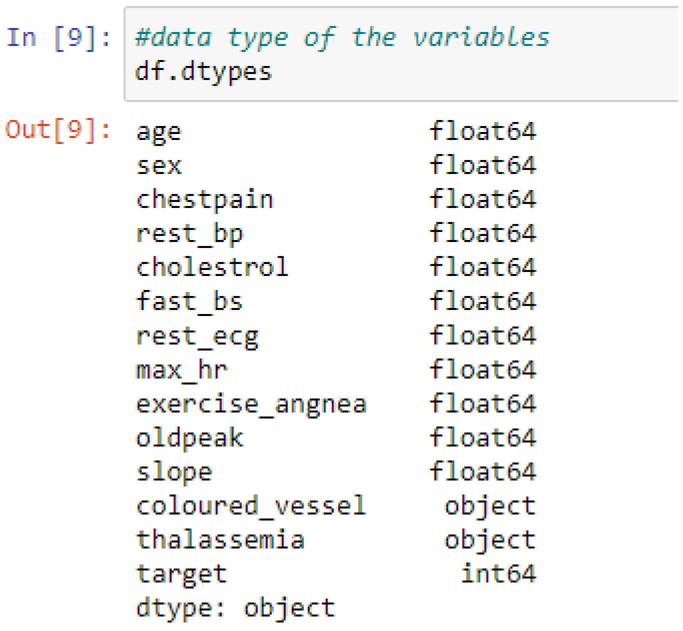
```
In [79]: #Assigning the names to each columns/Features in the dataframe.
         columns = ['age','sex','chestpain','rest_bp','cholestrol','fast_bs','rest_ecg','max_hr','exercise_angnea','oldpeak','slope',
                    'coloured_vessel','thalassemia','target']

In [80]: #Read Cleveland csv file & Rename the column names by the assigned names
         df = pd.read_csv('Data Set/processed.cleveland.data', encoding ='latin-1', names = columns)

In [81]: #5 random rows of the dataframe
         df.head(5)

Out[81]:
```

	age	sex	chestpain	rest_bp	cholestrol	fast_bs	rest_ecg	max_hr	exercise_angnea	oldpeak	slope	coloured_vessel	thalassemia	target
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0

Figure 4:A Python code snippet showing the Renaming of column names

Column names in a Data Frame are assigned using a predefined list (columns). The names, such as 'age' and 'cholesterol,' are specified to enhance clarity and prevent confusion in data analysis tasks.

2.1.4 Data type of variables/ columns



```
In [9]: #data type of the variables
        df.dtypes

Out[9]: age                 float64
        sex                 float64
        chestpain           float64
        rest_bp             float64
        cholestrol          float64
        fast_bs             float64
        rest_ecg            float64
        max_hr              float64
        exercise_angnea     float64
        oldpeak             float64
        slope               float64
        coloured_vessel      object
        thalassemia          object
        target                int64
        dtype: object
```

Figure 5: Python snippet depicting the Data types of each Variable.

The Data Frame comprises mainly numeric variables (float64) representing features like age and physiological measures, along with categorical variables (object) such as coloured_vessel and thalassemia. The target variable is of integer type (int64).

2.1.5 Interpretation of Different Attributes

Table 1 provides a comprehensive overview of every attributes present in the data set.

Variable name	Attribute Description	Measure
age	Age of patients	years

sex	Gender type	Male = 1;
		Female = 0

chestpain	Chest pain type	value 1: typical type 1 angina;
		value 2: typical type angina;
		value 3: non-angina pain;
		value 4 : asymptomatic

rest_bp	Resting blood pressure	mm Hg
cholestrol	Serum Cholesterol	mg/dl
fast_bs	Fasting blood sugar	

rest_ecg	Resting ElectroCardioGraphic results	values 0:normal;
		value1: 1 having ST-T wave abnormality;
		value 2:showing probable or definite left ventricular hypertrophy;

max_hr	Maximum heart rate achieved	value 1:yes; value 0 : no
exercise_angnea	Exercise induced angina	1 = yes; 0 = no
oldpeak	ST depression induced by exercise relative to rest	

slope	The slope of the peak exercise ST segment	value 1: unsloping;
		value 2 : flat;
		value 3 :down sloping

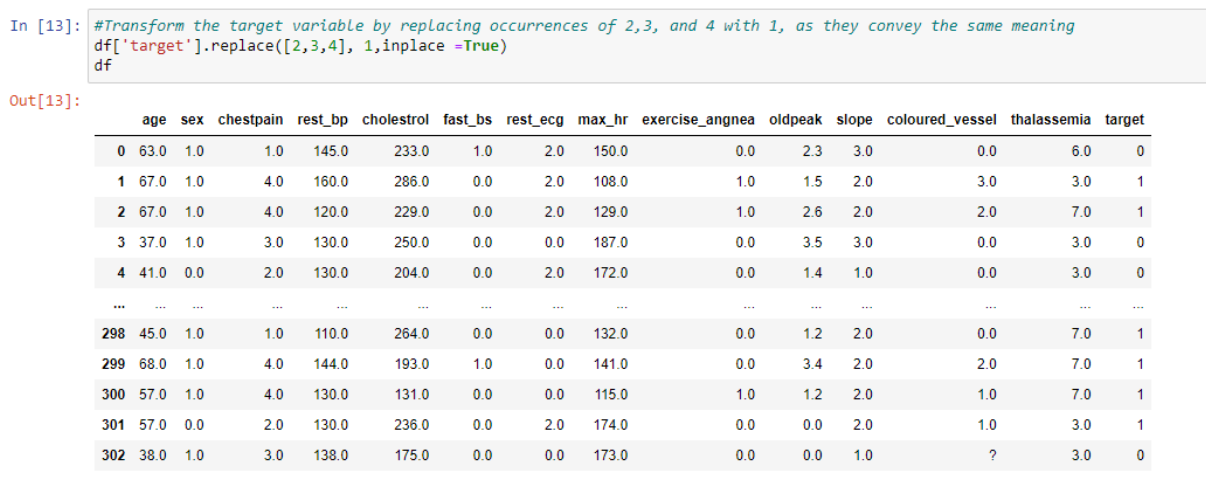
coloured_vessel	Number of major vessels colored by fluoroscopy	value 0-3

thalassemia	no explanation was provided, but probably thalassemia	value3 = normal;
		value 6 = fixed defect;
		value 7 = reversible defect

target	diagnosis of heart disease	value 0 = No heart disease value 2 = heart disease

Table 1: Data set Variable Description

2.1.6 Convert the target variable as binary output.



```
In [13]: #Transform the target variable by replacing occurrences of 2,3, and 4 with 1, as they convey the same meaning
         df['target'].replace([2,3,4], 1,inplace =True)
         df
```

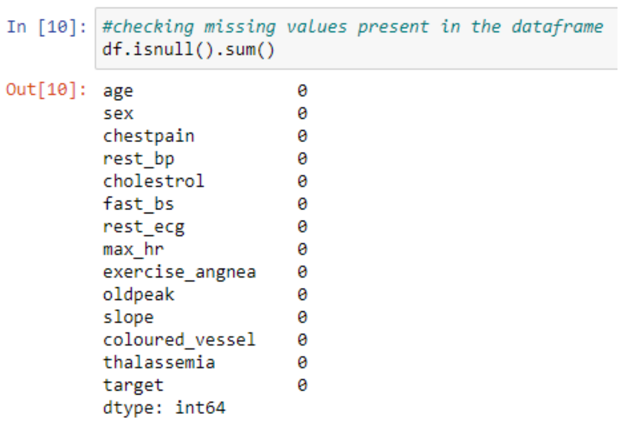
```
Out[13]:
         age  sex  chestpain  rest_bp  cholestrol  fast_bs  rest_ecg  max_hr  exercise_angnea  oldpeak  slope  coloured_vessel  thalassemia  target
    0   63.0  1.0        1.0    145.0       233.0      1.0       2.0   150.0              0.0      2.3    3.0              0.0          6.0       0
    1   67.0  1.0        4.0    160.0       286.0      0.0       2.0   108.0              1.0      1.5    2.0              3.0          3.0       1
    2   67.0  1.0        4.0    120.0       229.0      0.0       2.0   129.0              1.0      2.6    2.0              2.0          7.0       1
    3   37.0  1.0        3.0    130.0       250.0      0.0       0.0   187.0              0.0      3.5    3.0              0.0          3.0       0
    4   41.0  0.0        2.0    130.0       204.0      0.0       2.0   172.0              0.0      1.4    1.0              0.0          3.0       0
  ...    ...  ...        ...      ...         ...      ...       ...     ...              ...      ...    ...              ...          ...     ...
  298   45.0  1.0        1.0    110.0       264.0      0.0       0.0   132.0              0.0      1.2    2.0              0.0          7.0       1
  299   68.0  1.0        4.0    144.0       193.0      1.0       0.0   141.0              0.0      3.4    2.0              2.0          7.0       1
  300   57.0  1.0        4.0    130.0       131.0      0.0       0.0   115.0              1.0      1.2    2.0              1.0          7.0       1
  301   57.0  0.0        2.0    130.0       236.0      0.0       2.0   174.0              0.0      0.0    2.0              1.0          3.0       1
  302   38.0  1.0        3.0    138.0       175.0      0.0       0.0   173.0              0.0      0.0    1.0                ?          3.0       0
```

Figure 6: Python snippet showing the transformation of target variable into a binary output

The code snippet provided transforms the target variable in a Data Frame by replacing occurrences of 2, 3, and 4 with 1. This conversion essentially binarizes the target variable, condensing multiple classes into a binary outcome. This binarization is a strategic step frequently employed in binary classification tasks, wherein the positive class is denoted by 1, while all other values collectively represent the negative class.

2.1.6 Handling Missing Values & Symbols

I. Finding Null Values

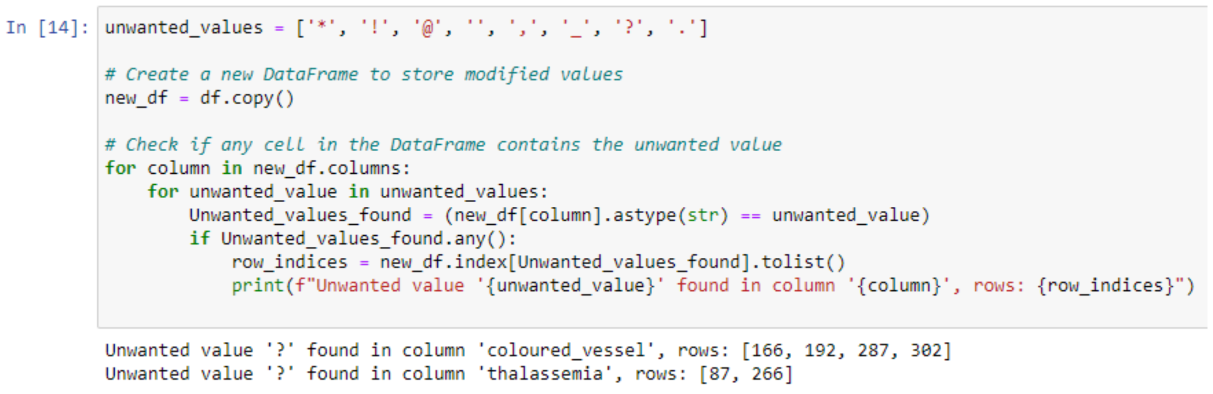


```
In [10]: #checking missing values present in the dataframe
         df.isnull().sum()

Out[10]: age                0
         sex                0
         chestpain          0
         rest_bp            0
         cholestrol         0
         fast_bs            0
         rest_ecg           0
         max_hr             0
         exercise_angnea    0
         oldpeak            0
         slope              0
         coloured_vessel    0
         thalassemia        0
         target             0
         dtype: int64
```

Figure 7: Python snippet showing the Null values in the data set

According to Figure 6 we can assure there are no null values presented in the data set.

II. Finding Symbols

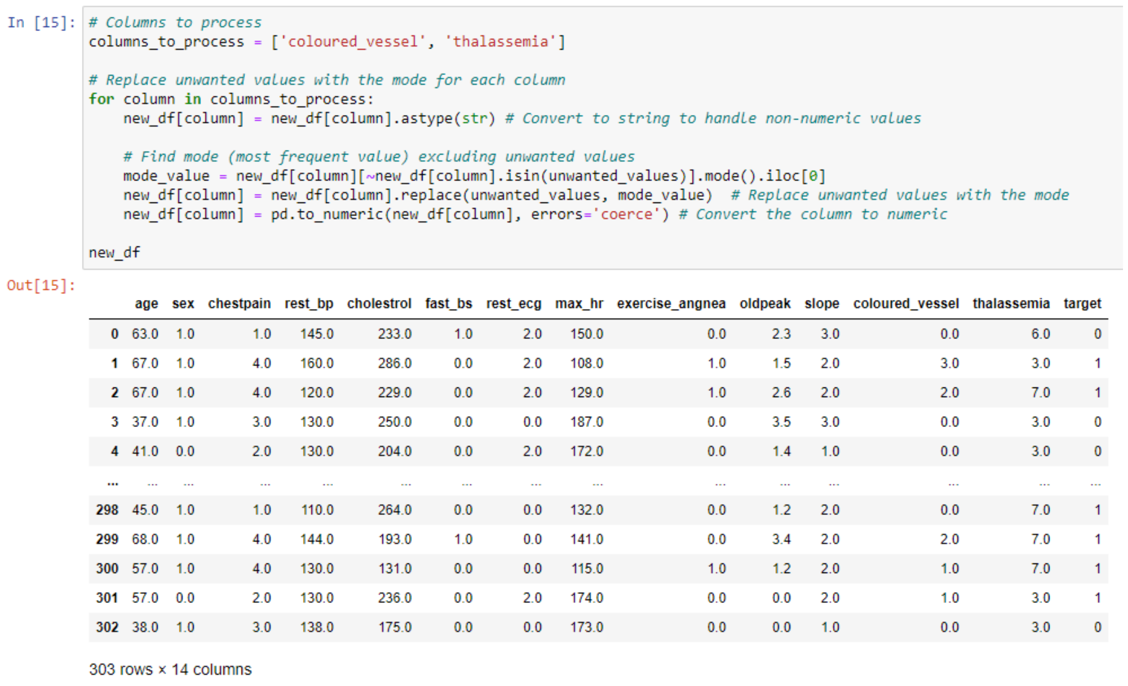


```
In [14]: unwanted_values = ['*', '!', '@', '', ',', '_', '?', '.']

         # Create a new DataFrame to store modified values
         new_df = df.copy()

         # Check if any cell in the DataFrame contains the unwanted value
         for column in new_df.columns:
             for unwanted_value in unwanted_values:
                 Unwanted_values_found = (new_df[column].astype(str) == unwanted_value)
                 if Unwanted_values_found.any():
                     row_indices = new_df.index[Unwanted_values_found].tolist()
                     print(f"Unwanted value '{unwanted_value}' found in column '{column}', rows: {row_indices}")

         Unwanted value '?' found in column 'coloured_vessel', rows: [166, 192, 287, 302]
         Unwanted value '?' found in column 'thalassemia', rows: [87, 266]
```

Figure 8: A Python code snippet of finding the symbols in the Data frame

According to figure 8 unwanted values ('?') were detected in the 'coloured_vessel' column (rows: 166, 192, 287, 302) and 'thalassemia' column (rows: 87, 266).

2.1.7 Replace **the unwanted values and convert the categorical column into numeric.**



```
In [15]: # Columns to process
         columns_to_process = ['coloured_vessel', 'thalassemia']

         # Replace unwanted values with the mode for each column
         for column in columns_to_process:
             new_df[column] = new_df[column].astype(str) # Convert to string to handle non-numeric values

             # Find mode (most frequent value) excluding unwanted values
             mode_value = new_df[column][~new_df[column].isin(unwanted_values)].mode().iloc[0]
             new_df[column] = new_df[column].replace(unwanted_values, mode_value)  # Replace unwanted values with the mode
             new_df[column] = pd.to_numeric(new_df[column], errors='coerce') # Convert the column to numeric

         new_df
```

Out[15]:

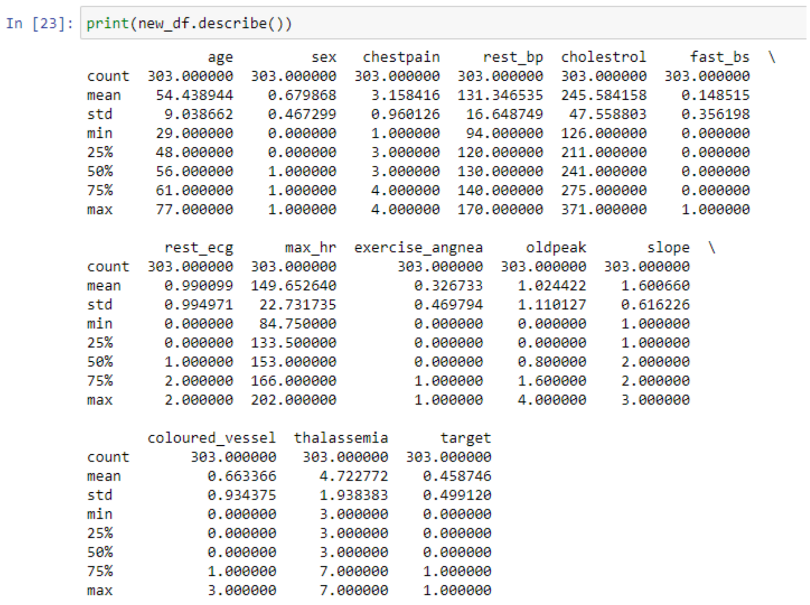
	age	sex	chestpain	rest_bp	cholestrol	fast_bs	rest_ecg	max_hr	exercise_angnea	oldpeak	slope	coloured_vessel	thalassemia	target
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	1
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	2.0	0.0	7.0	1
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	2.0	2.0	7.0	1
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	2.0	1.0	7.0	1
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	2.0	1.0	3.0	1
302	38.0	1.0	3.0	138.0	175.0	0.0	0.0	173.0	0.0	0.0	1.0	0.0	3.0	0

303 rows × 14 columns

Figure 9: A Python snippet showing the replacement of unwanted values and the conversion of categorical columns into numerical ones.

The code provided in figure 9 processes specific columns ('coloured_vessel' and 'thalassemia') in the Data Frame (new_df) by converting them to strings, replacing unwanted values with the mode (most frequently occurred value), and then converting the columns back to numeric type. This preprocessing ensures uniform data types, handles unwanted values, and facilitates numerical analysis or modelling.

2.1.8 Describing the data set.



```
In [23]: print(new_df.describe())

                age         sex   chestpain     rest_bp  cholestrol     fast_bs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean    54.438944    0.679868    3.158416  131.346535  245.584158    0.148515
std      9.038662    0.467299    0.960126   16.648749   47.558803    0.356198
min     29.000000    0.000000    1.000000   94.000000  126.000000    0.000000
25%     48.000000    0.000000    3.000000  120.000000  211.000000    0.000000
50%     56.000000    1.000000    3.000000  130.000000  241.000000    0.000000
75%     61.000000    1.000000    4.000000  140.000000  275.000000    0.000000
max     77.000000    1.000000    4.000000  170.000000  371.000000    1.000000

          rest_ecg      max_hr  exercise_angnea     oldpeak       slope  \
count  303.000000  303.000000       303.000000  303.000000  303.000000
mean     0.990099  149.652640         0.326733    1.024422    1.600660
std      0.994971   22.731735         0.469794    1.110127    0.616226
min      0.000000   84.750000         0.000000    0.000000    1.000000
25%      0.000000  133.500000         0.000000    0.000000    1.000000
50%      1.000000  153.000000         0.000000    0.800000    2.000000
75%      2.000000  166.000000         1.000000    1.600000    2.000000
max      2.000000  202.000000         1.000000    4.000000    3.000000

       coloured_vessel  thalassemia      target
count       303.000000   303.000000  303.000000
mean          0.663366     4.722772    0.458746
std           0.934375     1.938383    0.499120
min           0.000000     3.000000    0.000000
25%           0.000000     3.000000    0.000000
50%           0.000000     3.000000    0.000000
75%           1.000000     7.000000    1.000000
max           3.000000     7.000000    1.000000
```

Figure 10: Python snippet showing the description of the dataset.

The dataset consists of 303 records, each containing important features relevant to cardiovascular health. Some notable attributes include age, sex, chest pain, blood pressure, cholesterol levels, and various cardiovascular indicators. The target variable shows the presence (1) or absence (0) of heart disease, with a prevalence of around 45.9%. Key trends observed from statistical summaries include an average age of 54 years and a significant presence of male subjects. Due to the dataset's diversity across its features, it provides a solid foundation for comprehensive exploratory analysis and predictive modelling in the field of cardiovascular health. These findings highlight the dataset's importance in providing potential insights into heart-related conditions, contributing to informed decision-making in clinical or research settings.

2.1.9 Finding Outliers

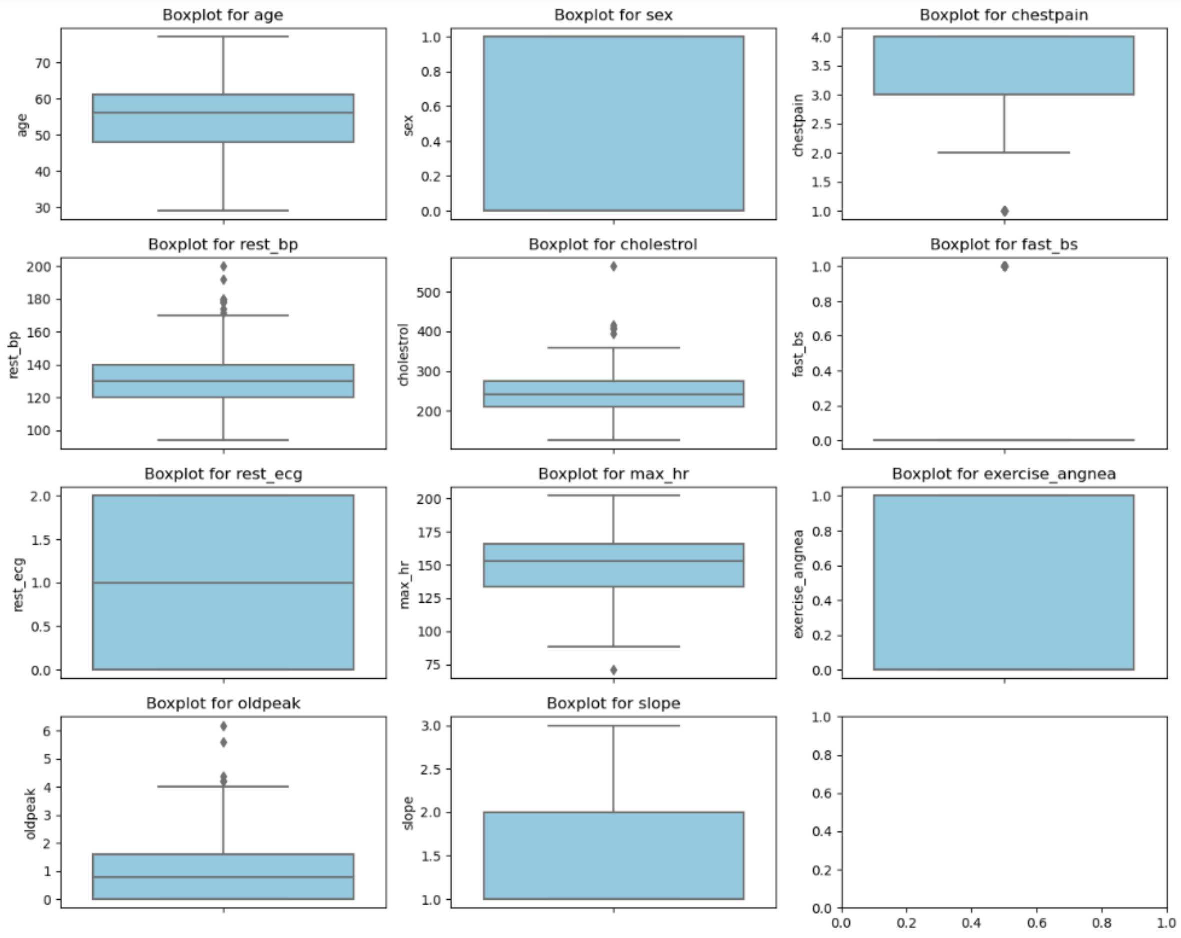
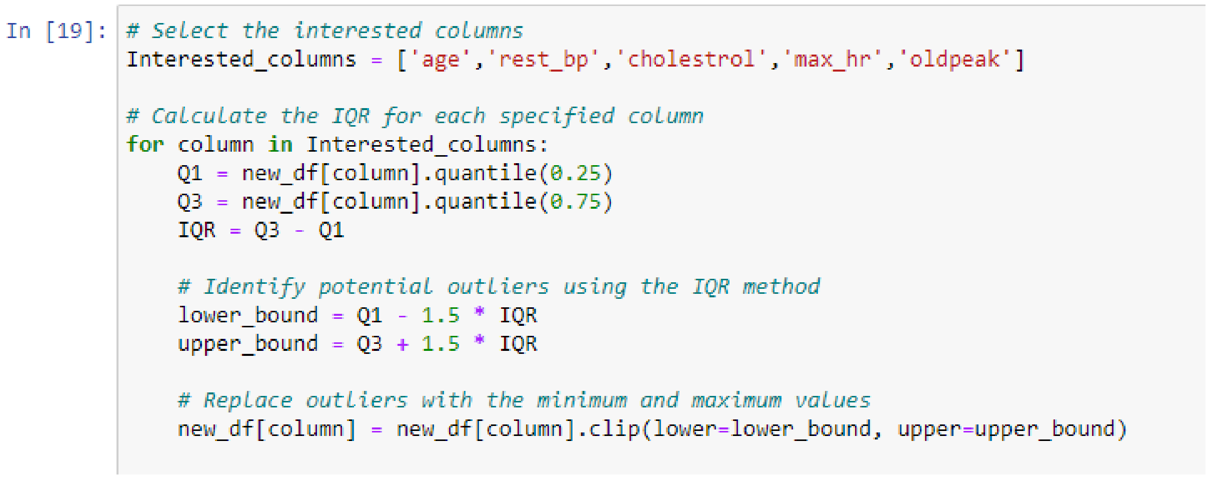


Figure 11: Box plots of all variables

The identification of outliers through box plots has revealed notable instances in specific variables within the dataset. Notably, the features rest_bp (resting blood pressure), cholesterol, max_hr (maximum heart rate), and oldpeak exhibit significant outlier observations.

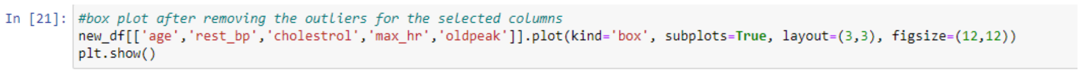
2.1.10 Removing Outliers.



```
In [19]: # Select the interested columns
         Interested_columns = ['age','rest_bp','cholestrol','max_hr','oldpeak']

         # Calculate the IQR for each specified column
         for column in Interested_columns:
             Q1 = new_df[column].quantile(0.25)
             Q3 = new_df[column].quantile(0.75)
             IQR = Q3 - Q1

             # Identify potential outliers using the IQR method
             lower_bound = Q1 - 1.5 * IQR
             upper_bound = Q3 + 1.5 * IQR

             # Replace outliers with the minimum and maximum values
             new_df[column] = new_df[column].clip(lower=lower_bound, upper=upper_bound)
```

Figure 12: Python snippet showing the removal of outliers in the observed columns

The code provided in Figure 12 focuses on a targeted subset of columns, namely 'age,' 'rest_bp,' 'cholestrol,' 'max_hr,' and 'oldpeak,' within the dataset. Employing the Interquartile Range (IQR) method, the code identifies potential outliers for each selected column. To maintain data integrity, outliers are then replaced with values confined within an acceptable range—specifically, the minimum and maximum values determined by 1.5 times the IQR. This helps ensure that extreme values don't mess up our analysis or predictions.



```
In [21]: #box plot after removing the outliers for the selected columns
         new_df[['age','rest_bp','cholestrol','max_hr','oldpeak']].plot(kind='box', subplots=True, layout=(3,3), figsize=(12,12))
         plt.show()
```

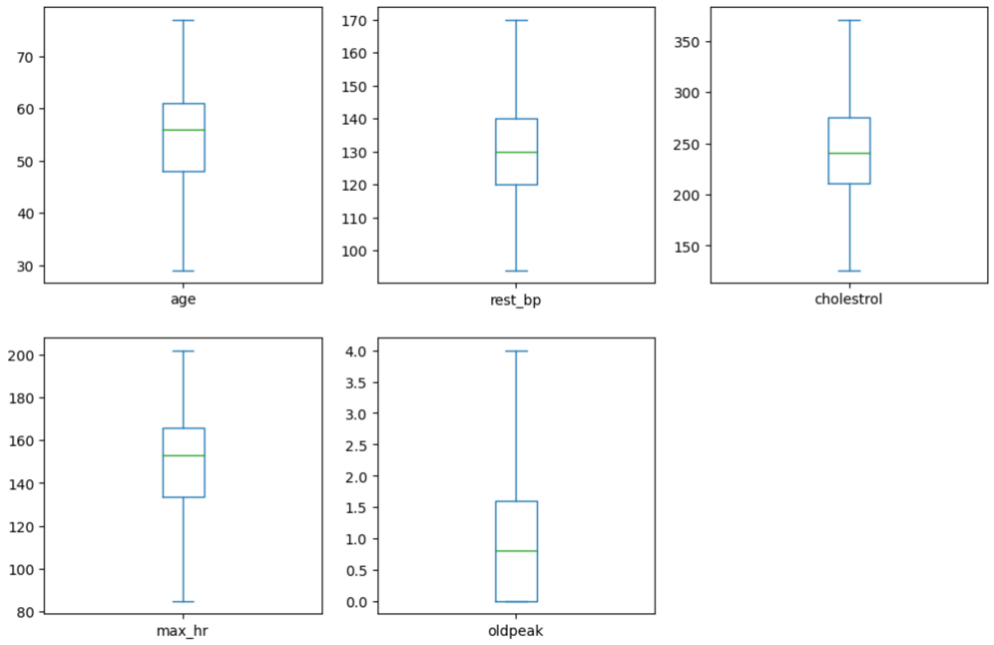
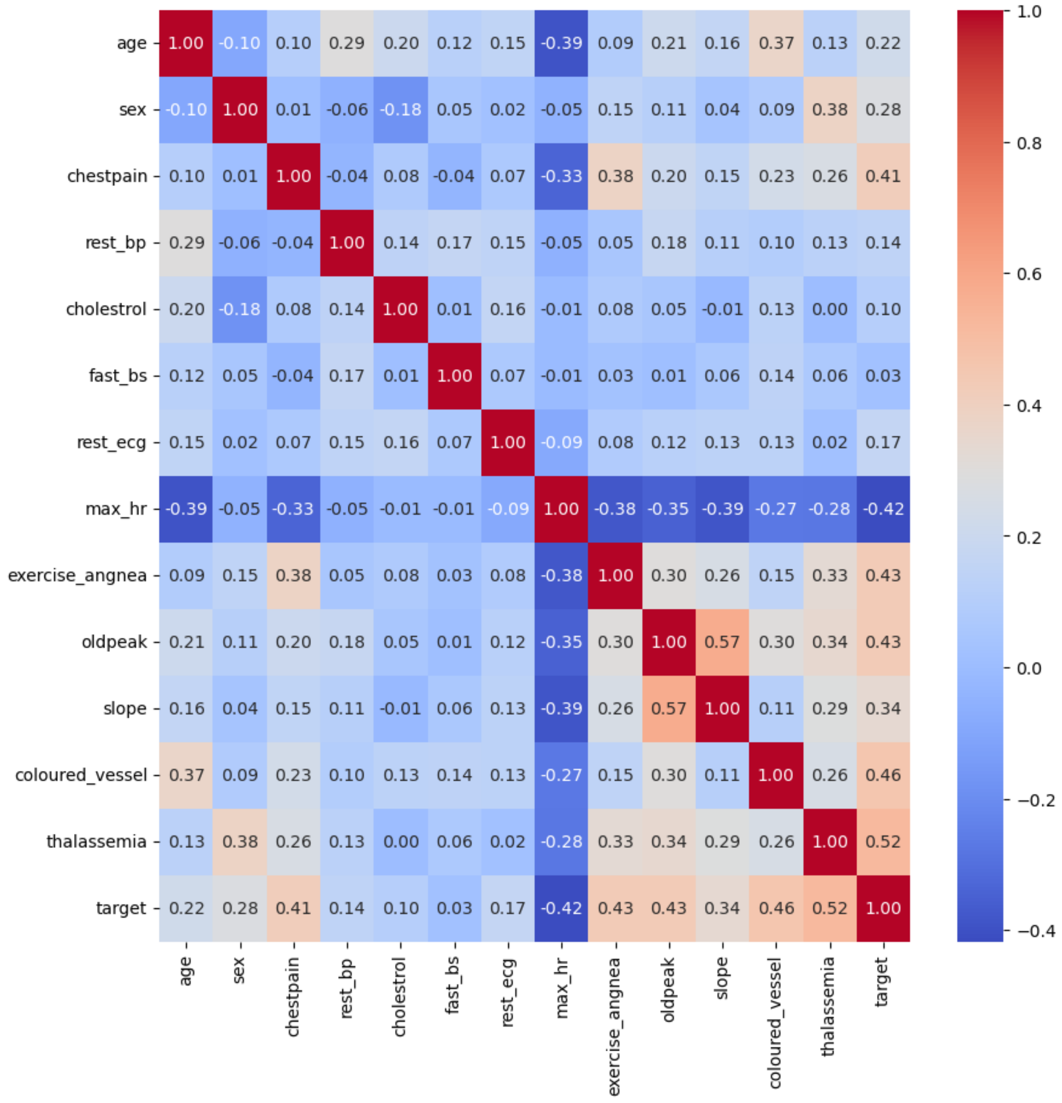


Figure 13: Python snippet showing the box plot after the removal of the outliers for the selected features

2.1.11 Correlation among the attributes

I. Correlation heatmap



	age	sex	chestpain	rest_bp	cholestrol	fast_bs	rest_ecg	max_hr	exercise_angnea	oldpeak	slope	coloured_vessel	thalassemia	target
age	1.00	-0.10	0.10	0.29	0.20	0.12	0.15	-0.39	0.09	0.21	0.16	0.37	0.13	0.22
sex	-0.10	1.00	0.01	-0.06	-0.18	0.05	0.02	-0.05	0.15	0.11	0.04	0.09	0.38	0.28
chestpain	0.10	0.01	1.00	-0.04	0.08	-0.04	0.07	-0.33	0.38	0.20	0.15	0.23	0.26	0.41
rest_bp	0.29	-0.06	-0.04	1.00	0.14	0.17	0.15	-0.05	0.05	0.18	0.11	0.10	0.13	0.14
cholestrol	0.20	-0.18	0.08	0.14	1.00	0.01	0.16	-0.01	0.08	0.05	-0.01	0.13	0.00	0.10
fast_bs	0.12	0.05	-0.04	0.17	0.01	1.00	0.07	-0.01	0.03	0.01	0.06	0.14	0.06	0.03
rest_ecg	0.15	0.02	0.07	0.15	0.16	0.07	1.00	-0.09	0.08	0.12	0.13	0.13	0.02	0.17
max_hr	-0.39	-0.05	-0.33	-0.05	-0.01	-0.01	-0.09	1.00	-0.38	-0.35	-0.39	-0.27	-0.28	-0.42
exercise_angnea	0.09	0.15	0.38	0.05	0.08	0.03	0.08	-0.38	1.00	0.30	0.26	0.15	0.33	0.43
oldpeak	0.21	0.11	0.20	0.18	0.05	0.01	0.12	-0.35	0.30	1.00	0.57	0.30	0.34	0.43
slope	0.16	0.04	0.15	0.11	-0.01	0.06	0.13	-0.39	0.26	0.57	1.00	0.11	0.29	0.34
coloured_vessel	0.37	0.09	0.23	0.10	0.13	0.14	0.13	-0.27	0.15	0.30	0.11	1.00	0.26	0.46
thalassemia	0.13	0.38	0.26	0.13	0.00	0.06	0.02	-0.28	0.33	0.34	0.29	0.26	1.00	0.52
target	0.22	0.28	0.41	0.14	0.10	0.03	0.17	-0.42	0.43	0.43	0.34	0.46	0.52	1.00

Figure 14: Correlation heatmap

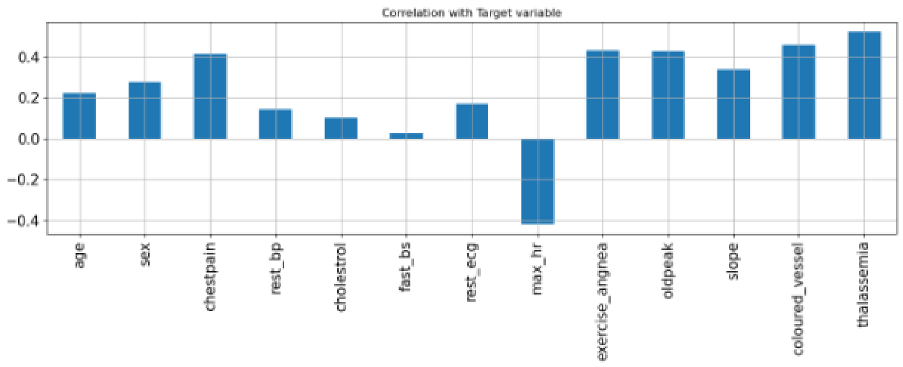


Figure 15: Correlation with the target variable

A heatmap demonstrating the relationships and correlation coefficients between characteristics is shown in Figure 14. The correlation between two features is represented by each coloured cell, and the strength of the correlation is indicated by the colour intensity of the cell. There is no correlation between the corresponding properties when the correlation value is zero, but a value less than zero implies a negative correlation.

As per figure 15 the dataset is divided into features (X) and the target variable (y). A bar graph illustrates correlations, highlighting 'max_hr' with a significant negative correlation (-0.42), indicating lower heart disease likelihood. Conversely, 'rest_bp' shows a positive correlation. 'Fast_bs' has minimal impact (0.03), while 'thalassemia' stands out with the highest positive correlation (0.52).

II. Pair plot

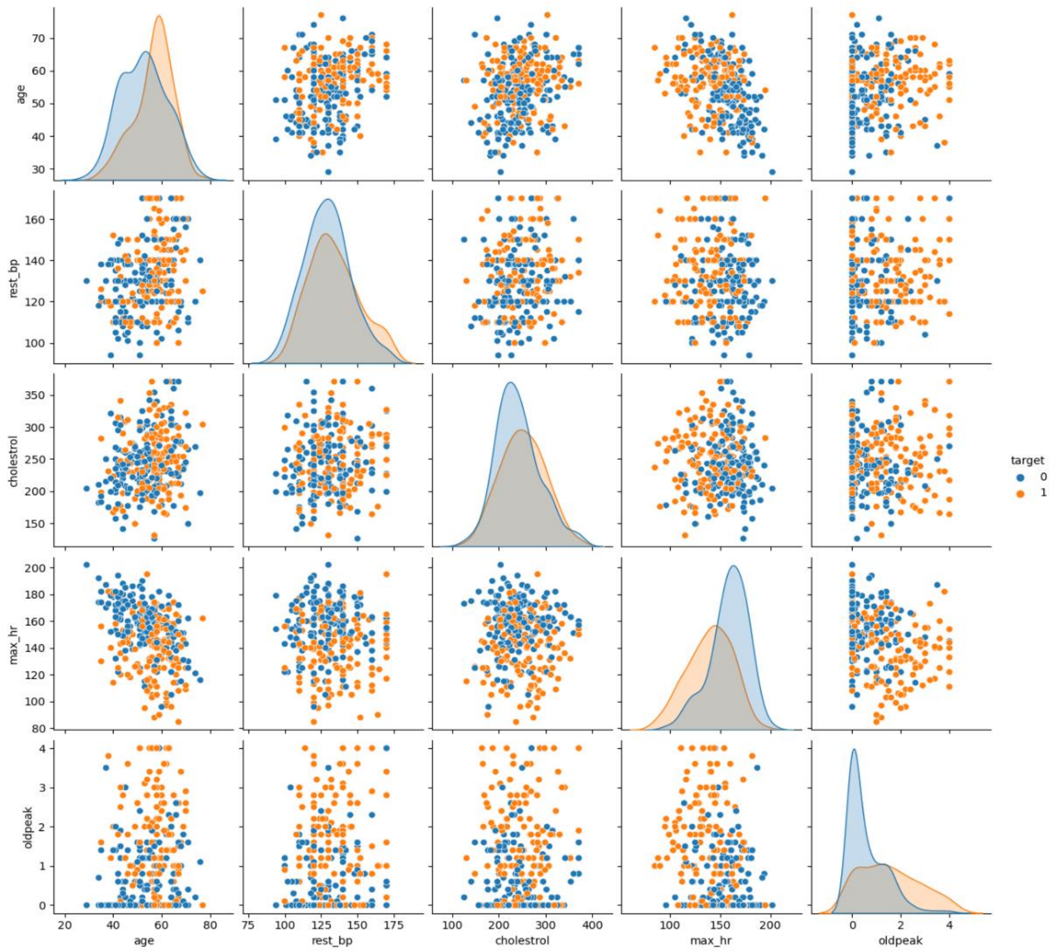


Figure 16: Pair plot of numerical features

Upon analysing the pair plot for the variables 'age,' 'rest_bp,' 'cholestrol,' 'max_hr,' and 'oldpeak,' several observations have been noted. No apparent linear trends were observed among the variables, suggesting a lack of straightforward correlations. However, intriguing clusters were identified in the scatterplots of 'max_hr' against 'age,' 'rest_bp,' and 'cholestrol,' hinting at potential patterns in specific areas. Importantly, after the removal of outliers, there were no notable extreme values, reinforcing the validity of the outlier removal process. The diagonal elements, particularly in the 'oldpeak' variable, revealed distinct trends between patients with and without heart disease.

Specifically, patients without heart disease exhibited a steeper graph on the left, contrasting with the pattern observed for diseased individuals.

2.1.12 Pie Charts for Categorical Variable

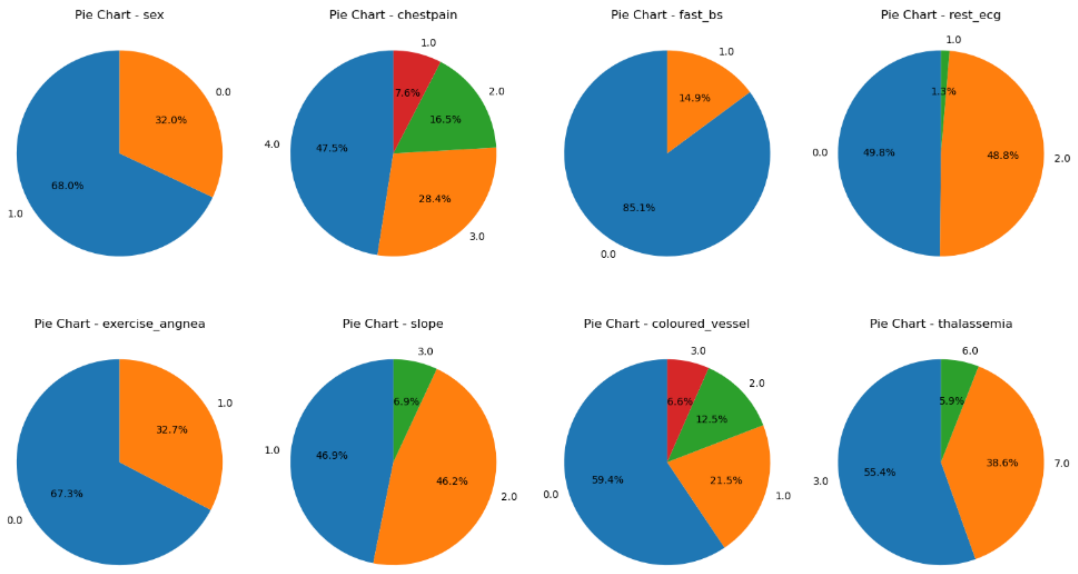


Figure 17:Pie Chart for Categorical Features

2.1.13 Histograms for Continuous Variable

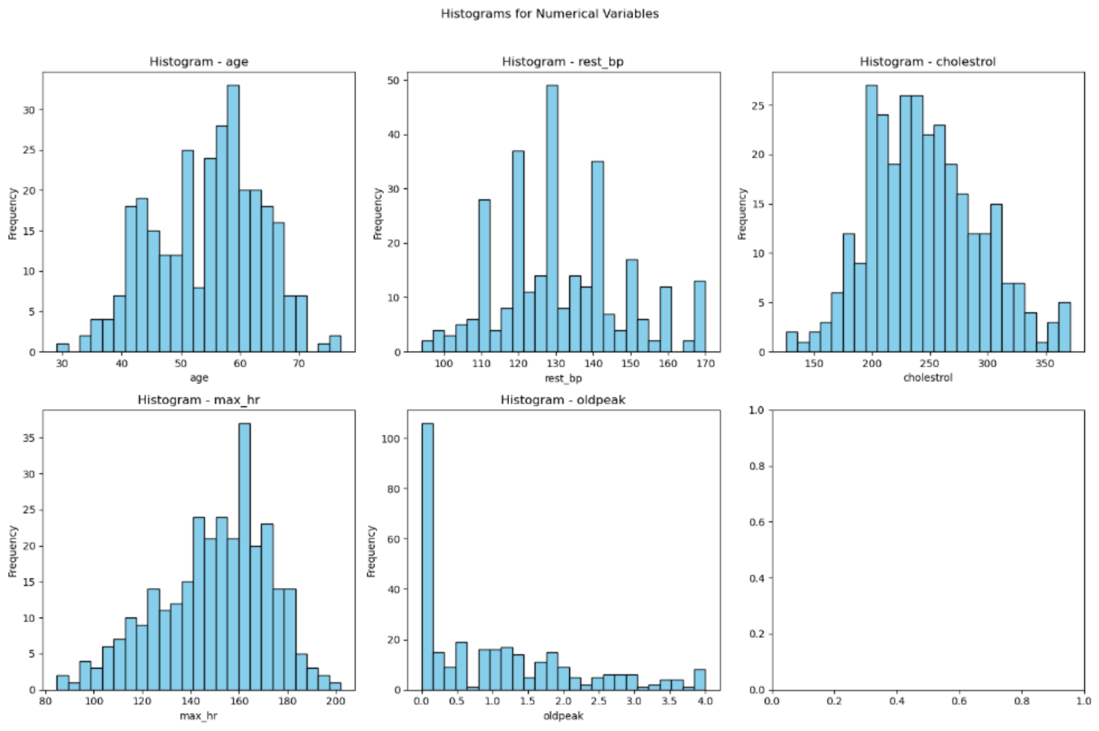


Figure 18:Histogram for Continuous Features

2.1.14 Distribution of Heart Disease

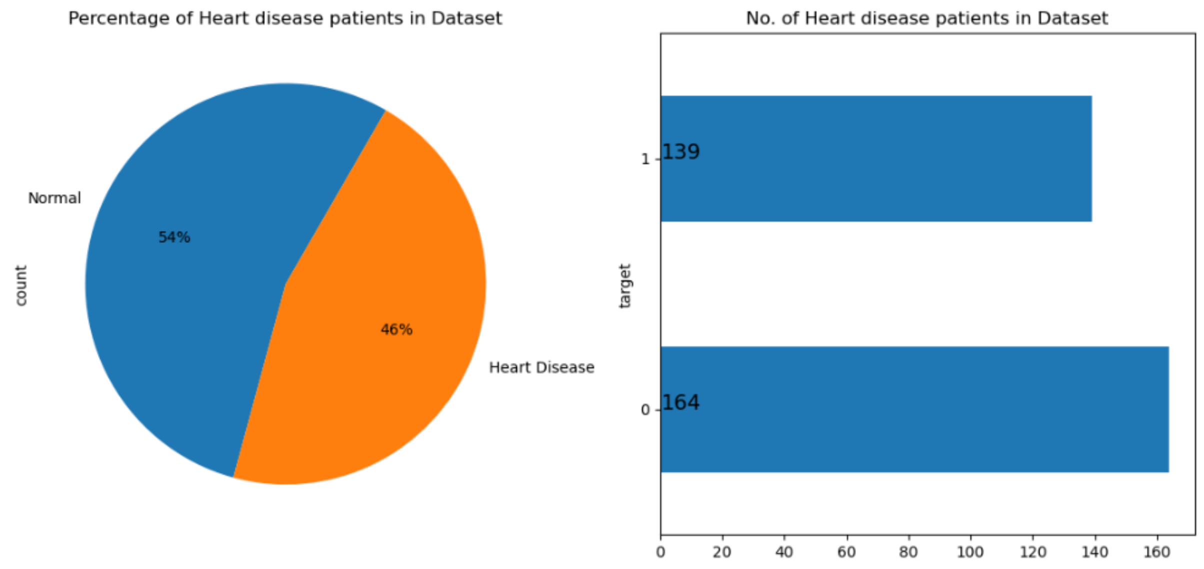


Figure 19: Pie chart and Bar chart of heart disease

A figure with two subplots was created to visualize heart disease prevalence. The pie chart depicts the percentage distribution, revealing that 54% (164 individuals) in the dataset are considered normal, while 46% (139 individuals) exhibit heart disease. The horizontal bar chart further illustrates the count of patients, emphasizing the proportion of heart disease cases within the dataset.

2.1.15 Gender-wise Distribution of Heart Disease

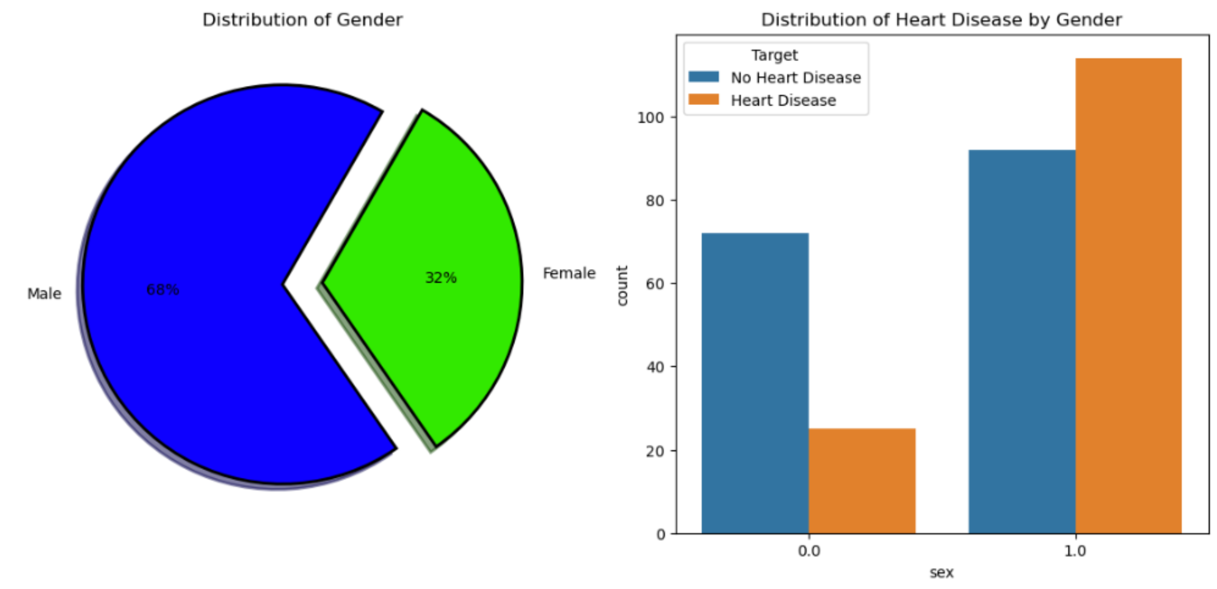


Figure 20: Gender-wise distribution (Pie chart & Bar chart)

The figure comprises two subplots visualizing gender distribution and its correlation with heart disease. The pie chart indicates a dataset with 68% male and 32% female individuals. The bar chart illustrates a higher count of heart disease cases in males (above 100) compared to females (below 30), offering a succinct overview of gender-specific heart disease prevalence.

2.1.16 Age-wise distribution of heart disease

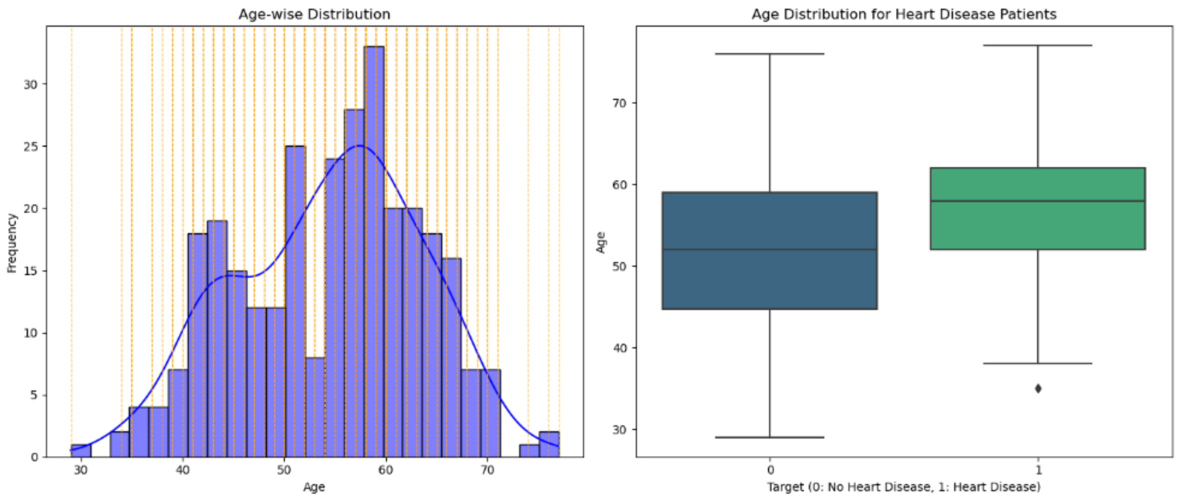


Figure 21:Representation of Age wise distribution of heart disease using a distribution plot with a rug plot and a box plot

The distribution plot with a rug plot and a box plot in Figure 19, offers insights into the relationship between age and heart disease. The left subplot's distribution plot with rug lines indicates a concentration of patients in the 50-60 age range. The right subplot's box plot reinforces that these individuals are more susceptible to heart disease, emphasizing the significance of age as a contributing factor.

2.1.17 Distribution of Chest pain type

i) Distribution of Chest Pain against the Target

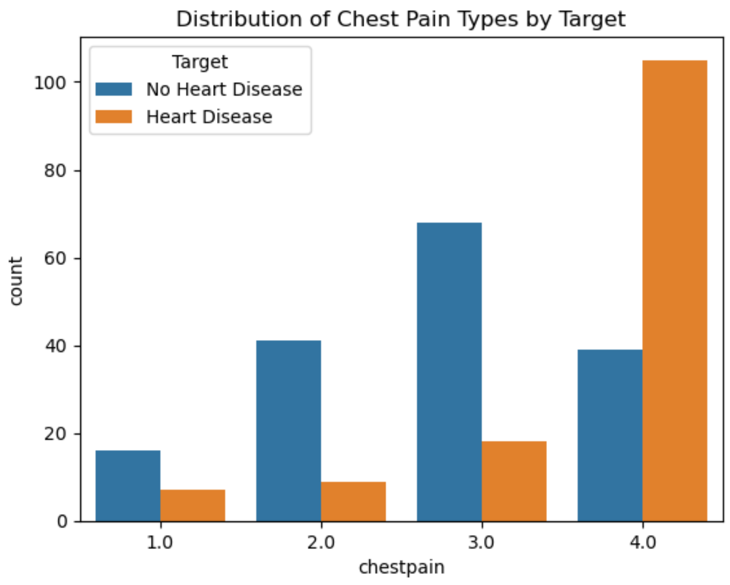


Figure 22: Distribution of Chest Pain Type by Target using Bar chart.

This bar chart illustrates the distribution of chest pain types based on the target variable. Notably, pain type 4 exhibits the highest contribution to heart disease cases, while pain type 1 registers the lowest frequency. The chart effectively captures the varying impact of chest pain types on the likelihood of heart disease, providing a clear visualization of their respective associations.

ii) Distribution of Chest Pain against Gender

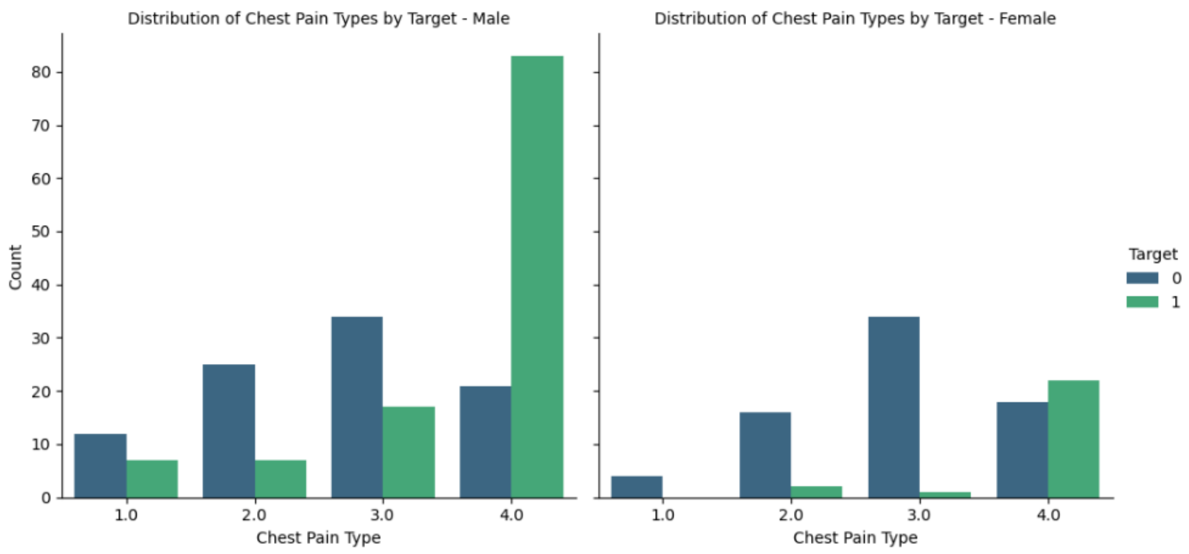
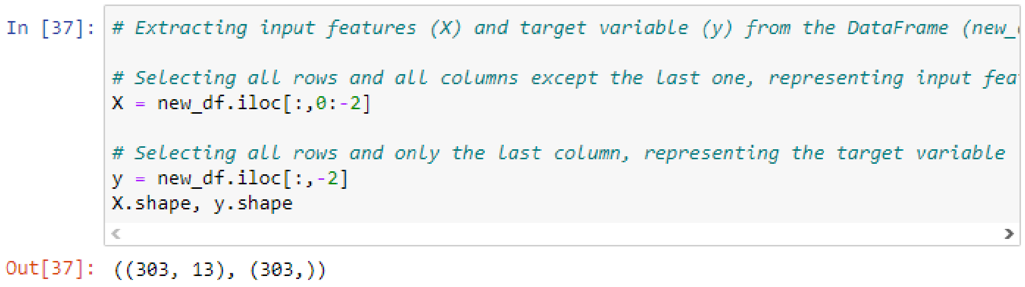


Figure 23: Distribution of Chest Pain types against Gender

The bar charts in Figure 21 suggests that males predominantly experience chest pain type 4, contributing to a higher count of heart disease cases compared to females. In contrast, for females, the occurrence of chest pain types 1, 2, and 3 is considerably lower compared to chest pain type 4. This observation highlights a potential gender-specific pattern in the distribution of chest pain types and their association with heart disease.

2.1.18 Advanced Question: e.g., Factor Analysis

2.1.18.1 Setting the input and output features.

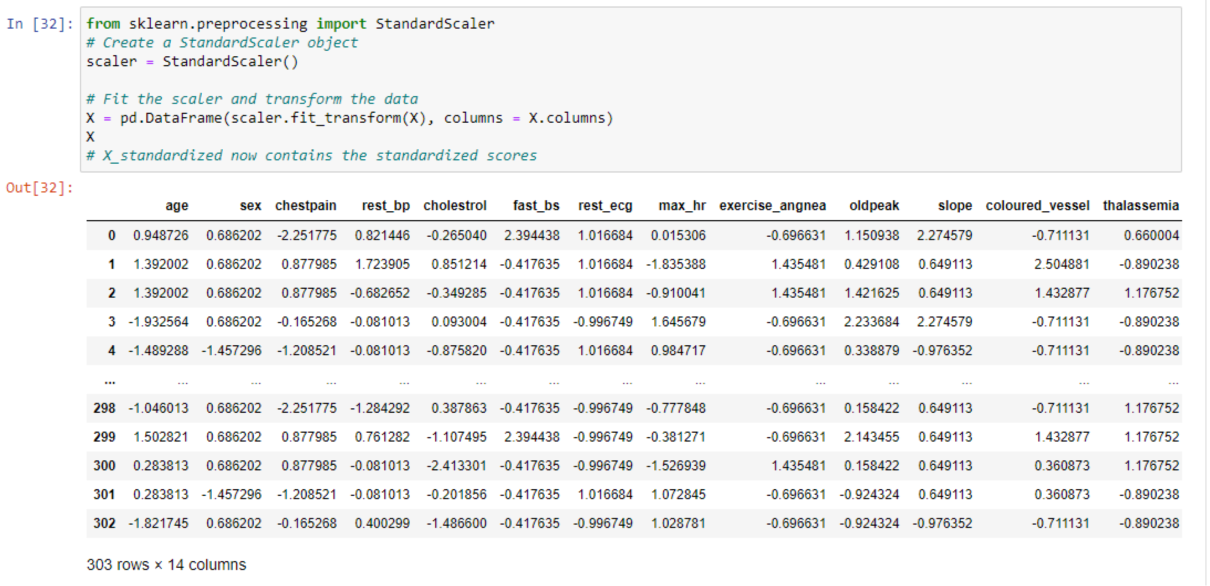


```
In [37]: # Extracting input features (X) and target variable (y) from the DataFrame (new_

         # Selecting all rows and all columns except the last one, representing input fea
         X = new_df.iloc[:,0:-2]

         # Selecting all rows and only the last column, representing the target variable
         y = new_df.iloc[:,-2]
         X.shape, y.shape

Out[37]: ((303, 13), (303,))
```

Figure 24:Python snippet showing the setting the input and output features.

The dataset (new_df) is pre-processed by segregating input features (X) and the target variable (y). Features include all rows and all columns except the last, while the target variable includes all rows and only the last column. This step, crucial for subsequent modelling, simplifies the dataset for analysis.

2.1.18.2 Standardizing data.



```
In [32]: from sklearn.preprocessing import StandardScaler
         # Create a StandardScaler object
         scaler = StandardScaler()

         # Fit the scaler and transform the data
         X = pd.DataFrame(scaler.fit_transform(X), columns = X.columns)
         X
         # X_standardized now contains the standardized scores
Out[32]:
```

	age	sex	chestpain	rest_bp	cholestrol	fast_bs	rest_ecg	max_hr	exercise_angnea	oldpeak	slope	coloured_vessel	thalassemia
0	0.948726	0.686202	-2.251775	0.821446	-0.265040	2.394438	1.016684	0.015306	-0.696631	1.150938	2.274579	-0.711131	0.660004
1	1.392002	0.686202	0.877985	1.723905	0.851214	-0.417635	1.016684	-1.835388	1.435481	0.429108	0.649113	2.504881	-0.890238
2	1.392002	0.686202	0.877985	-0.682652	-0.349285	-0.417635	1.016684	-0.910041	1.435481	1.421625	0.649113	1.432877	1.176752
3	-1.932564	0.686202	-0.165268	-0.081013	0.093004	-0.417635	-0.996749	1.645679	-0.696631	2.233684	2.274579	-0.711131	-0.890238
4	-1.489288	-1.457296	-1.208521	-0.081013	-0.875820	-0.417635	1.016684	0.984717	-0.696631	0.338879	-0.976352	-0.711131	-0.890238
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	-1.046013	0.686202	-2.251775	-1.284292	0.387863	-0.417635	-0.996749	-0.777848	-0.696631	0.158422	0.649113	-0.711131	1.176752
299	1.502821	0.686202	0.877985	0.761282	-1.107495	2.394438	-0.996749	-0.381271	-0.696631	2.143455	0.649113	1.432877	1.176752
300	0.283813	0.686202	0.877985	-0.081013	-2.413301	-0.417635	-0.996749	-1.526939	1.435481	0.158422	0.649113	0.360873	1.176752
301	0.283813	-1.457296	-1.208521	-0.081013	-0.201856	-0.417635	1.016684	1.072845	-0.696631	-0.924324	0.649113	0.360873	-0.890238
302	-1.821745	0.686202	-0.165268	0.400299	-1.486600	-0.417635	-0.996749	1.028781	-0.696631	-0.924324	-0.976352	-0.711131	-0.890238

303 rows × 14 columns

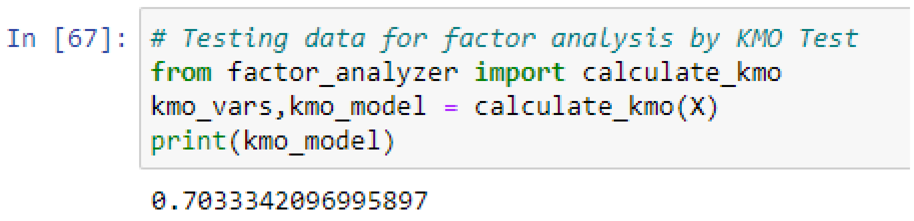
Figure 25: Python snippet showing the standardization of data.

In the data preprocessing phase, the input features (X) are standardized using the StandardScaler from the sci-kit-learn library. Standardization involves transforming the features to have a mean of 0 and a standard deviation of 1. This process is crucial for ensuring uniform scales across features, contributing to the optimal performance of machine learning models. The resulting standardized feature set, denoted as 'X_standardized', is now well-prepared for subsequent model training and evaluation.

2.1.18.3 Data Suitability Assessment for Factor Analysis

Factor analysis is a statistical method used to explore underlying relationships among observed variables and identify latent factors that contribute to their patterns. Before conducting factor analysis, it is common to assess the suitability of the data through tests such as the Kaiser-Meyer-Olkin (KMO) measure, Bartlett's Test of Sphericity, and the chi-square test. These tests help ensure that the dataset is appropriate for factor analysis by examining factors like sampling adequacy, the presence of patterns, and the correlation structure.

1) KMO Test

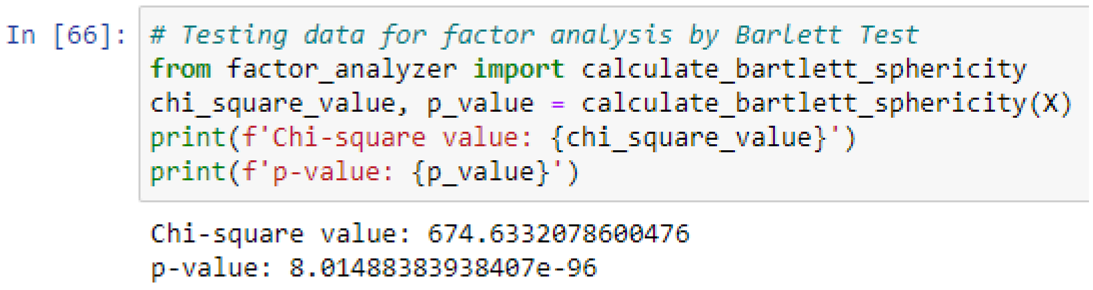


```
In [67]: # Testing data for factor analysis by KMO Test
         from factor_analyzer import calculate_kmo
         kmo_vars,kmo_model = calculate_kmo(X)
         print(kmo_model)

         0.7033342096995897
```

Figure 26: Python Snippet showing the KMO test

A KMO value close to 1 indicates favourable sampling adequacy for factor analysis. in this analysis, yielding a value of approximately 0.7033 indicates moderate to good sampling adequacy for factor analysis. This suggests that the dataset is suitable, with observed variables sharing sufficient common variance for meaningful extraction of latent factors.

2) Bartlett's Test



```
In [66]: # Testing data for factor analysis by Barlett Test
         from factor_analyzer import calculate_bartlett_sphericity
         chi_square_value, p_value = calculate_bartlett_sphericity(X)
         print(f'Chi-square value: {chi_square_value}')
         print(f'p-value: {p_value}')

         Chi-square value: 674.6332078600476
         p-value: 8.01488383938407e-96
```

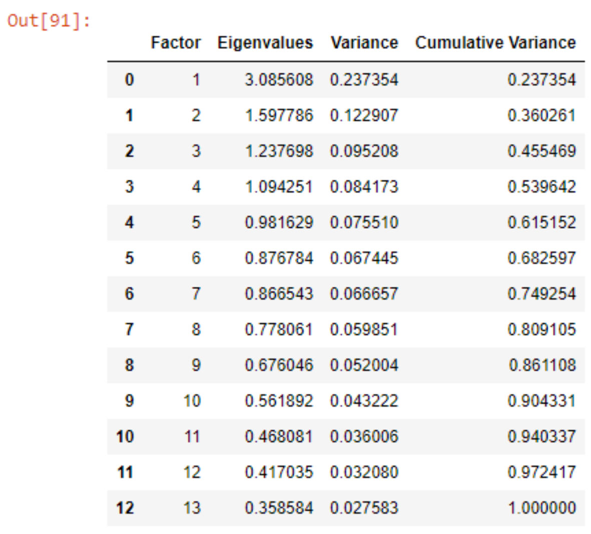
Figure 27: Python snippet showing Bartlett's Test

Bartlett's Test of Sphericity assesses whether the correlation matrix of observed variables is significantly different from an identity matrix, indicating the presence of patterns suitable for factor analysis. In this analysis, the chi-square value is 674.63, and the associated p-value is extremely low ($p < 0.001$). This result signifies a rejection of the null hypothesis, suggesting that the dataset contains meaningful patterns, reinforcing the appropriateness of the data for factor analysis.

2.1.18.4 Determining the number of factors.

Selecting the appropriate number of factors in factor analysis involves evaluating eigenvalues, with a common criterion being to retain factors with values exceeding 1. Additionally, the scree plot visually identifies the optimal factor count by observing the point where the curve levels off. These methods collectively guide the determination of a meaningful and concise factor structure for the dataset.

I. Eigen Value



Out[91]:

	Factor	Eigenvalues	Variance	Cumulative Variance
0	1	3.085608	0.237354	0.237354
1	2	1.597786	0.122907	0.360261
2	3	1.237698	0.095208	0.455469
3	4	1.094251	0.084173	0.539642
4	5	0.981629	0.075510	0.615152
5	6	0.876784	0.067445	0.682597
6	7	0.866543	0.066657	0.749254
7	8	0.778061	0.059851	0.809105
8	9	0.676046	0.052004	0.861108
9	10	0.561892	0.043222	0.904331
10	11	0.468081	0.036006	0.940337
11	12	0.417035	0.032080	0.972417
12	13	0.358584	0.027583	1.000000

Table 2: Factor Analysis Summary

The table summarizes factor analysis results, including eigenvalues, variances, and cumulative variances for each factor. Eigenvalues exceeding 1 indicate significance, while variance columns detail factor-specific contributions. Cumulative variance offers an overall perspective. This aids in selecting an optimal number of factors for a meaningful dataset factor structure.

II. Scree plot

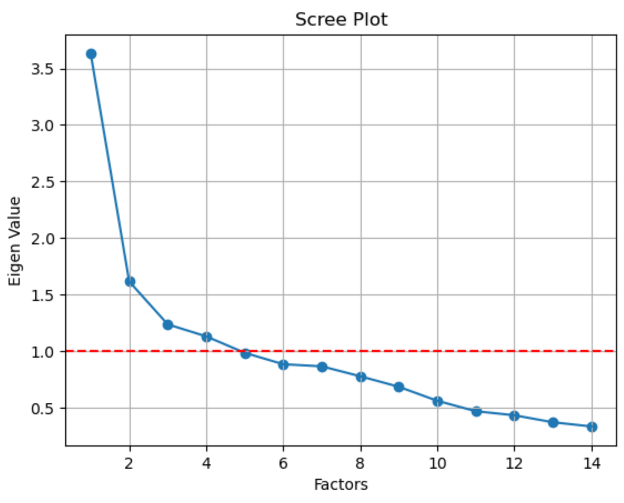


Figure 28: Scree plot

In Figure 28 the scree plot has four distinct points that lie above the red dashed horizontal line at the eigenvalue 1. These points correspond to factors with eigenvalues exceeding 1, indicating their significance in explaining variance within the dataset.

2.1.18.5 Checking Factor Loading

```
In [44]: # Obtain and display the shape of the factor loadings matrix from the Factor Analyzer results
         fa.loadings_.shape

Out[44]: (13, 3)
```

Figure 30: Python snippet of factor loading shape

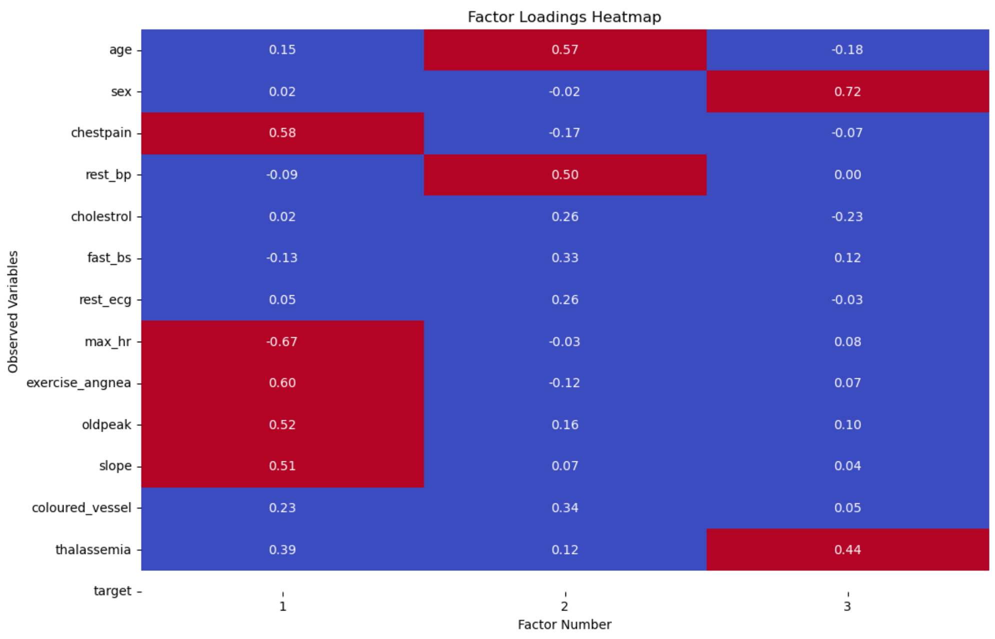


Figure 29:.Factor loading matrix (heat map)

The factor analysis model, initially targeted for four factors based on eigenvalues and the scree plot, unexpectedly selected three factors. Considering the suboptimal cumulative variance coverage of approximately 45.5% with the current three-factor model, there is a strategic inclination to explore factor rotation. The intention is to execute the rotation method and assess its impact on variance coverage. This proactive step aligns with the iterative nature of factor analysis, aiming for an improved model that better captures the underlying patterns in the dataset and achieves enhanced explanatory power.

2.1.18.6 Rotation

```
In [46]: fa = FactorAnalyzer(n_factors=4,rotation='varimax')
         fa.fit(X)
         fa_load = pd.DataFrame(fa.loadings_,index=X.columns)
```

Figure 31: a Python snippet of Factor rotation

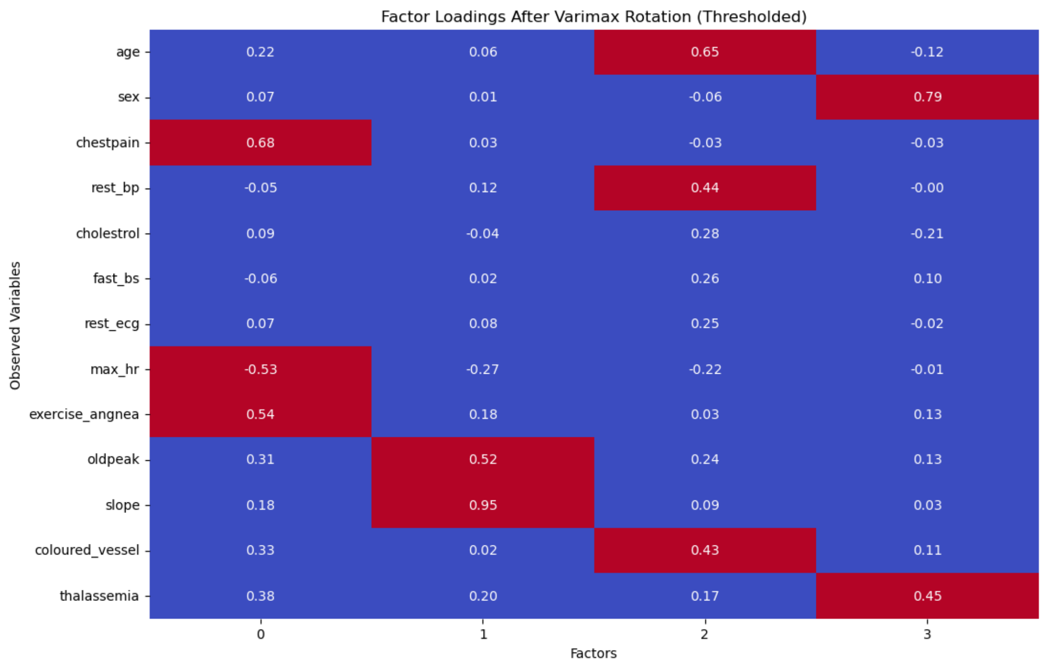


Figure 32:Factor loading matrix (heat map) after rotation.

After applying the Varimax rotation method, the Factor Loadings display a threshold outcome, revealing four factors. This refined model achieves a higher cumulative variance coverage of 54%, showcasing the effectiveness of the rotation in optimizing the factor structure. The decision to employ Varimax rotation has proven fruitful, contributing to a more comprehensive understanding of the underlying patterns in the dataset and enhancing the explanatory power of the factor analysis model.

2.1.18.7 Interpretation

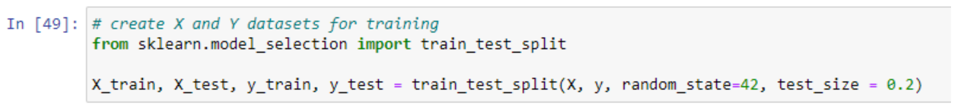
The interpretation of the factor analysis results reveals distinct and meaningful patterns in the dataset. The factors have been identified, named, and associated with specific variables, providing valuable insights into underlying constructs. These factors are mentioned in the table given below.

Factor number	Variables	Meaning full Factor name
1	'chestpain' 'max_hr' 'exercise_angnea'	Cardiovascular Symptoms Factor
2	oldpeak' 'slope' 'coloured_vessel'	Stress Response Factor
3	age' 'rest_bp'	Vascular Health Factor
4	sex' 'thalassemia'	Gender and Thalassemia Factor

Figure 33:Factor analysis Interpretation table

2.2 Question 2: e.g., Prediction Task

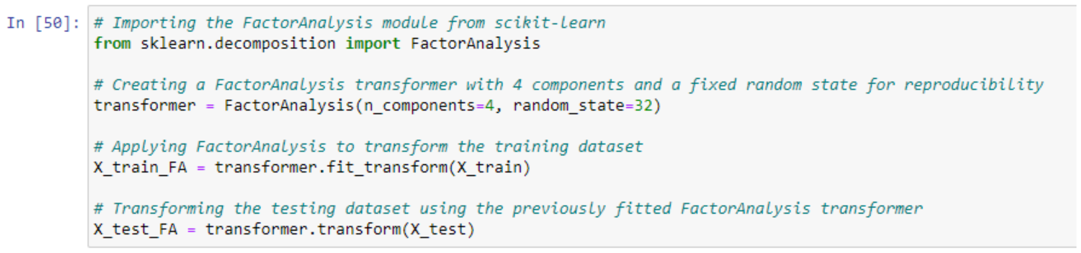
2.2.1 Splitting the Dataset.



```
In [49]: # create X and Y datasets for training
         from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size = 0.2)
```

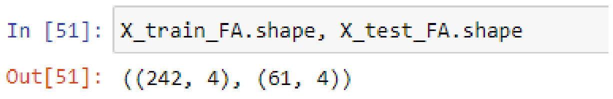
Figure 34: Python snippet showing the splitting of the data set

2.2.2 Transformation of Training and Testing Data



```
In [50]: # Importing the FactorAnalysis module from scikit-learn
         from sklearn.decomposition import FactorAnalysis

         # Creating a FactorAnalysis transformer with 4 components and a fixed random state for reproducibility
         transformer = FactorAnalysis(n_components=4, random_state=32)

         # Applying FactorAnalysis to transform the training dataset
         X_train_FA = transformer.fit_transform(X_train)

         # Transforming the testing dataset using the previously fitted FactorAnalysis transformer
         X_test_FA = transformer.transform(X_test)
```

Figure 36:Python snippet of Transforming Training and Testing data

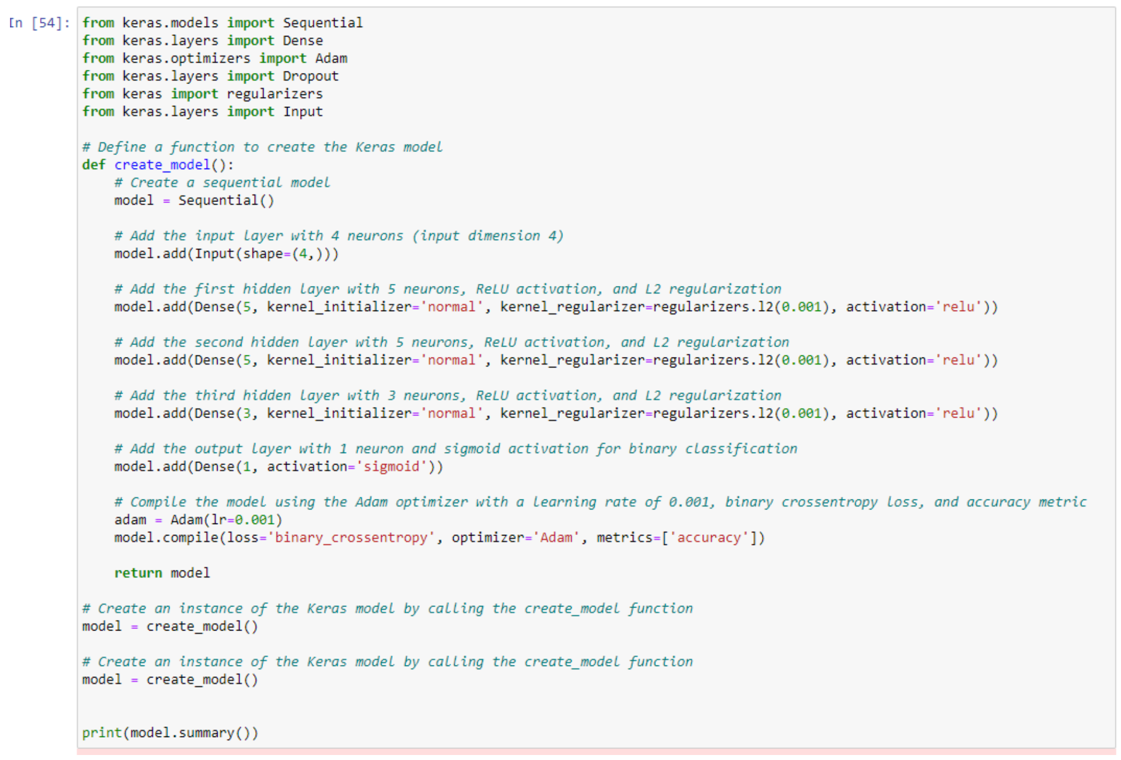


```
In [51]: X_train_FA.shape, X_test_FA.shape

Out[51]: ((242, 4), (61, 4))
```

Figure 35:DImension of transformed data after factor Analysis.

The dataset is initially split into training and testing sets using the train_test_split function, with 20% of the data reserved for testing and a fixed random state of 42 for reproducibility. Following the split, Factor Analysis is applied to the training dataset using the factor analysis transformer. This process extracts underlying factors and reduces the dimensionality of the feature space to 4 components. The same transformer is then used to transform the testing dataset, ensuring consistency in the factor analysis applied to both sets. The resulting transformed datasets, X_train_FA and X_test_FA, exhibit shapes indicating the reduced feature space dimensions for training and testing, respectively.

2.2.3 **Neural Network Building**

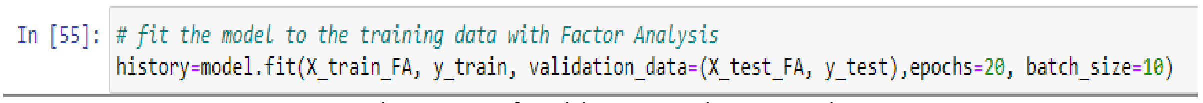


```
In [54]: from keras.models import Sequential
         from keras.layers import Dense
         from keras.optimizers import Adam
         from keras.layers import Dropout
         from keras import regularizers
         from keras.layers import Input

         # Define a function to create the Keras model
         def create_model():
             # Create a sequential model
             model = Sequential()

             # Add the input layer with 4 neurons (input dimension 4)
             model.add(Input(shape=(4,)))

             # Add the first hidden layer with 5 neurons, ReLU activation, and L2 regularization
             model.add(Dense(5, kernel_initializer='normal', kernel_regularizer=regularizers.l2(0.001), activation='relu'))

             # Add the second hidden layer with 5 neurons, ReLU activation, and L2 regularization
             model.add(Dense(5, kernel_initializer='normal', kernel_regularizer=regularizers.l2(0.001), activation='relu'))

             # Add the third hidden layer with 3 neurons, ReLU activation, and L2 regularization
             model.add(Dense(3, kernel_initializer='normal', kernel_regularizer=regularizers.l2(0.001), activation='relu'))

             # Add the output layer with 1 neuron and sigmoid activation for binary classification
             model.add(Dense(1, activation='sigmoid'))

             # Compile the model using the Adam optimizer with a learning rate of 0.001, binary crossentropy loss, and accuracy metric
             adam = Adam(lr=0.001)
             model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])

             return model

         # Create an instance of the Keras model by calling the create_model function
         model = create_model()

         # Create an instance of the Keras model by calling the create_model function
         model = create_model()


         print(model.summary())
```

Figure 37: Python snippet showing the Neural network model.

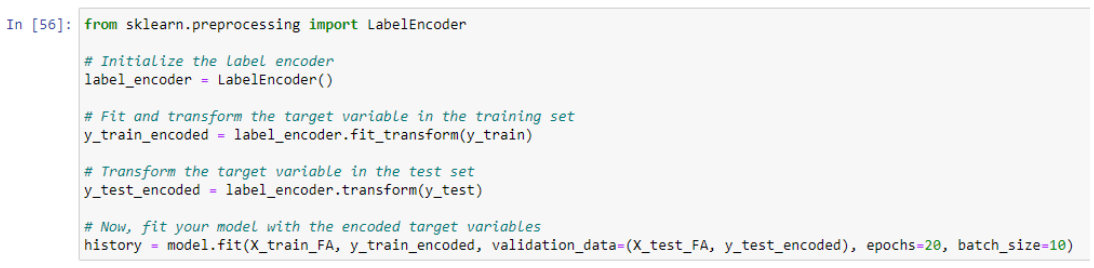
The neural network model is constructed using the Keras library, implementing a sequential architecture with multiple hidden layers for feature abstraction. The neural network comprises an input layer with 4 neurons, followed by three hidden layers with ReLU activation, L2 regularization (0.001 strength), and varying neurons (5, 5, 3). The output layer features a single neuron with sigmoid activation for binary classification. The model is compiled with Adam optimizer (lr=0.001), binary cross-entropy loss, and accuracy metric.

2.2.4 Model Training



```
In [55]: # fit the model to the training data with Factor Analysis
         history=model.fit(X_train_FA, y_train, validation_data=(X_test_FA, y_test),epochs=20, batch_size=10)
```

Figure 38: Python snippet of Model training with Factor Analysis Features

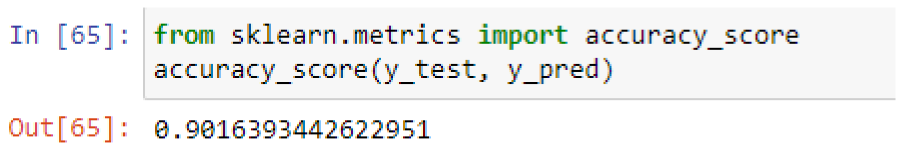


```
In [56]: from sklearn.preprocessing import LabelEncoder

         # Initialize the label encoder
         label_encoder = LabelEncoder()

         # Fit and transform the target variable in the training set
         y_train_encoded = label_encoder.fit_transform(y_train)

         # Transform the target variable in the test set
         y_test_encoded = label_encoder.transform(y_test)

         # Now, fit your model with the encoded target variables
         history = model.fit(X_train_FA, y_train_encoded, validation_data=(X_test_FA, y_test_encoded), epochs=20, batch_size=10)
```

Figure 39::Label Encoding for Target Variable and Model Training

The model is trained using factor analysis-transformed features on the training data. The training process spans 20 epochs with a batch size of 10, and the model's performance is assessed on the validation set. Additionally, a label encoder is employed to transform the target variable (binary classification) into a numerical format for compatibility with the model. The training history is recorded for evaluation and further analysis.

2.2.5 Model Accuracy Evaluation



```
In [65]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test, y_pred)

Out[65]: 0.9016393442622951
```

Figure 40: Python snippet showing the evaluation of the model(accuracy)

According to figure 40 the created neural network model exhibits an accuracy of 90.16%.

2.2.6 Training and Validation Performance

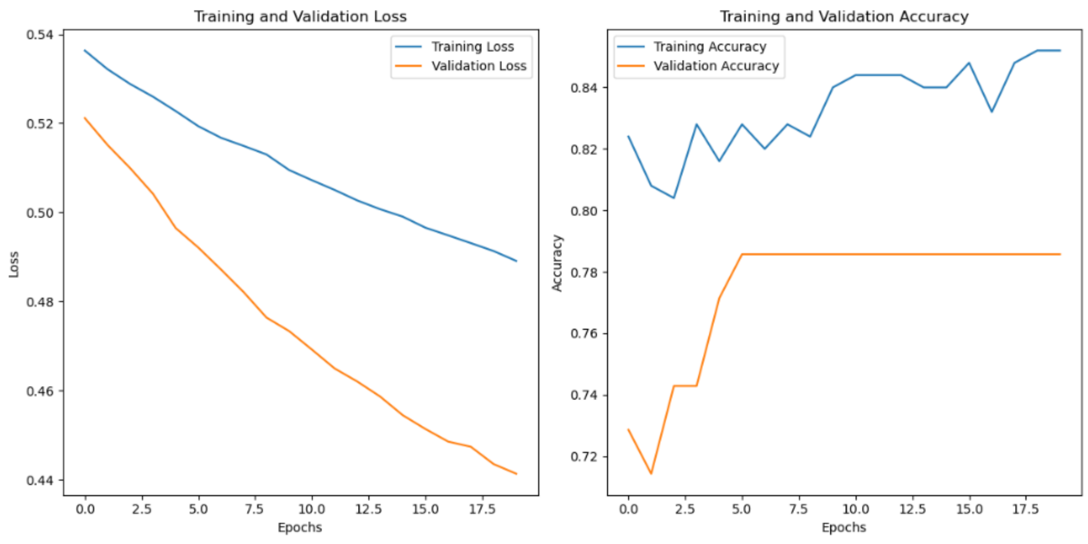
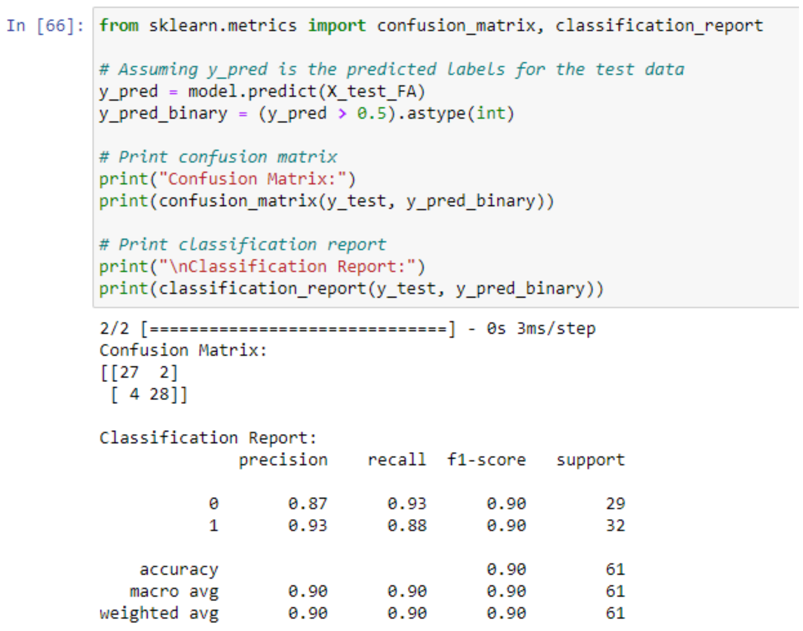


Figure 41: Training and validation loss and accuracy

2.2.7 Model Evaluation Matrix



```
In [66]: from sklearn.metrics import confusion_matrix, classification_report

         # Assuming y_pred is the predicted labels for the test data
         y_pred = model.predict(X_test_FA)
         y_pred_binary = (y_pred > 0.5).astype(int)

         # Print confusion matrix
         print("Confusion Matrix:")
         print(confusion_matrix(y_test, y_pred_binary))

         # Print classification report
         print("\nClassification Report:")
         print(classification_report(y_test, y_pred_binary))
```

```
2/2 [==============================] - 0s 3ms/step
Confusion Matrix:
[[27  2]
 [ 4 28]]

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.93      0.90        29
           1       0.93      0.88      0.90        32

    accuracy                           0.90        61
   macro avg       0.90      0.90      0.90        61
weighted avg       0.90      0.90      0.90        61
```

Figure 42:Python snippet showing the model evaluation matrix

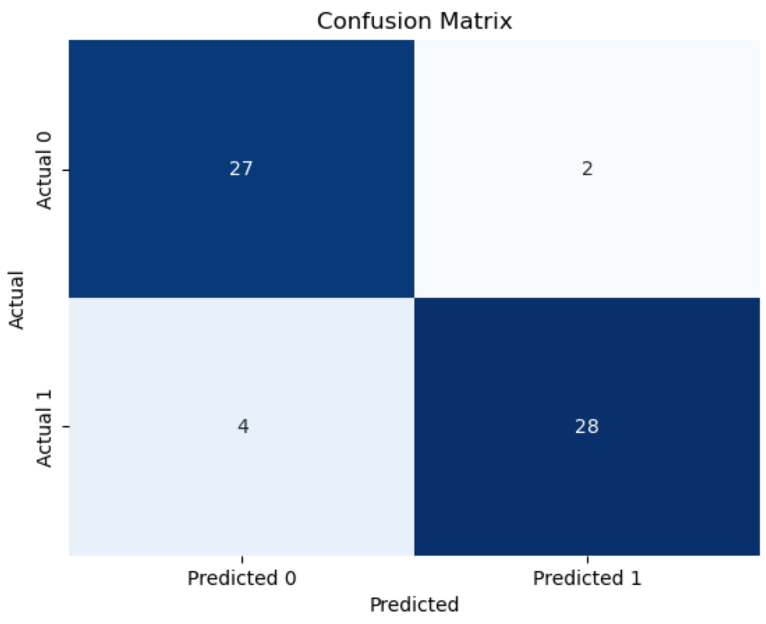


Figure 43: Visual representation of the confusion matrix

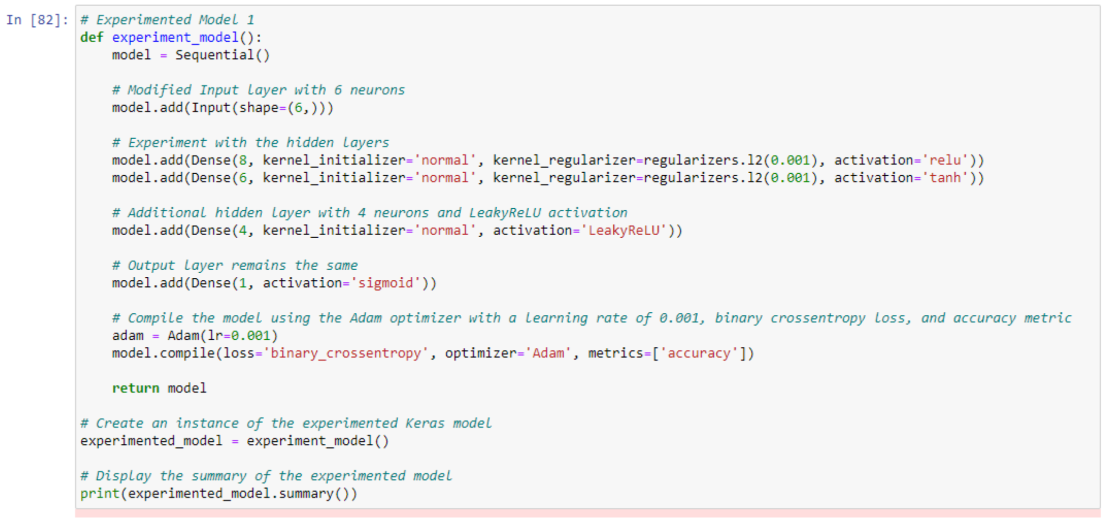
The confusion matrix illustrates the model's accuracy in distinguishing heart disease patients and non-patients. It correctly predicted 27 non-patients and 28 patients, with 2 false positives and 4 false negatives. The overall accuracy stands at 90%, demonstrating effective heart disease classification. Precision, recall, and F1-score metrics indicate a balanced performance across classes, emphasizing the model's reliability in predicting heart disease.

2.2.8 Future Optimization

The model exhibits commendable accuracy, achieving 90% correctness in predicting heart disease cases. However, there's room for improvement in reducing false positives and negatives. Fine-tuning hyperparameters, exploring different neural network architectures, and increasing the dataset size could enhance model performance. Additionally, feature engineering and incorporating domain-specific knowledge may contribute to better predictive capabilities.

2.2.10 Advanced Question 2.2

2.2.10.1 Building the Experimental Model.

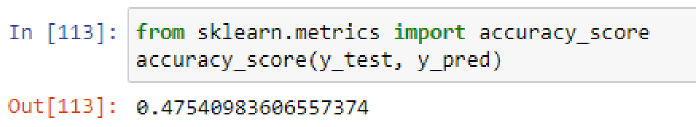


```
In [82]: # Experimented Model 1
         def experiment_model():
             model = Sequential()

             # Modified Input Layer with 6 neurons
             model.add(Input(shape=(6,)))

             # Experiment with the hidden layers
             model.add(Dense(8, kernel_initializer='normal', kernel_regularizer=regularizers.l2(0.001), activation='relu'))
             model.add(Dense(6, kernel_initializer='normal', kernel_regularizer=regularizers.l2(0.001), activation='tanh'))

             # Additional hidden layer with 4 neurons and LeakyReLU activation
             model.add(Dense(4, kernel_initializer='normal', activation='LeakyReLU'))

             # Output layer remains the same
             model.add(Dense(1, activation='sigmoid'))

             # Compile the model using the Adam optimizer with a learning rate of 0.001, binary crossentropy loss, and accuracy metric
             adam = Adam(lr=0.001)
             model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])

             return model

         # Create an instance of the experimented Keras model
         experimented_model = experiment_model()

         # Display the summary of the experimented model
         print(experimented_model.summary())
```

Figure 44:Python snippet showing the new experimental model.

The experimented model represents a departure from the original model in several key aspects. Firstly, the input layer of the experimented model comprises six neurons, as opposed to the original model's four neurons. The hidden layers in the experimented model exhibit variations in both structure and activation functions. Specifically, the experimented model incorporates three hidden layers with eight, six, and four neurons respectively. The activation functions employed in these layers include ReLU, tanh, and LeakyReLU, diverging from the original model's consistent use of ReLU activation. Additionally, while both models utilize normal weight initialization, the experimented model introduces LeakyReLU activation in one of its hidden layers. Despite these architectural differences, both models share the commonality of using the Adam optimizer with a learning rate of 0.001 and feature an output layer with one neuron employing a sigmoid activation function. These distinctions suggest that the experimented model introduces nuanced changes in network structure and activation functions, potentially influencing its learning capacity and generalization performance.

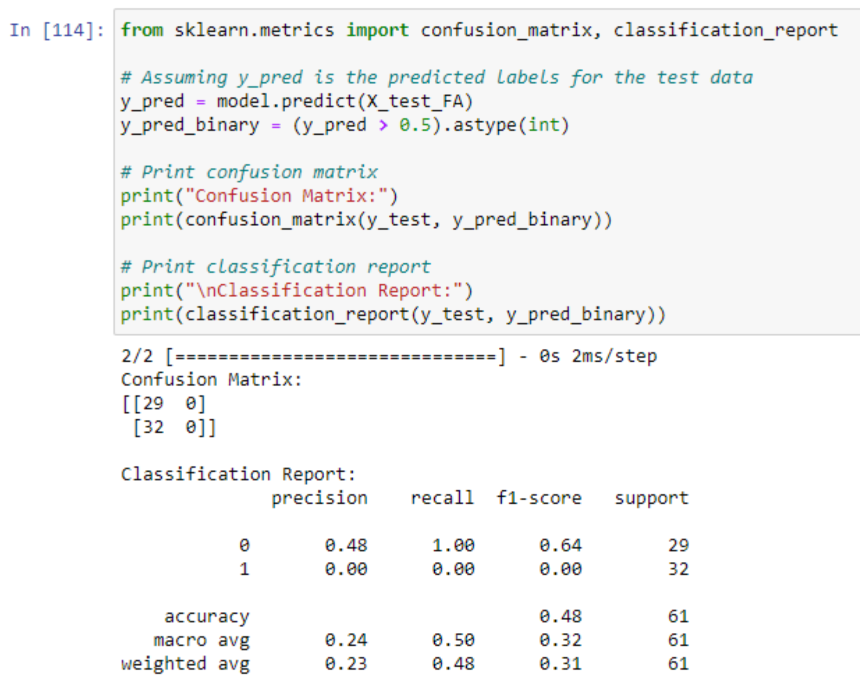
2.2.10.2 Accuracy of the experimental model



```
In [113]: from sklearn.metrics import accuracy_score
          accuracy_score(y_test, y_pred)

Out[113]: 0.47540983606557374
```

Figure 45: Python snippet showing the accuracy of the experimental model.

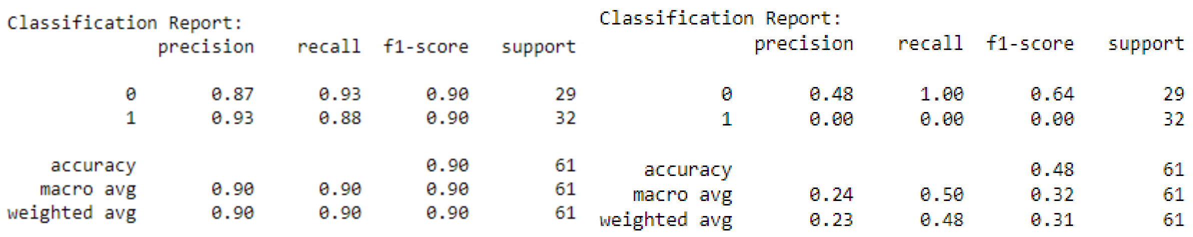
2.2.10.3 Evaluation Matrix of Experimental Model



```
In [114]: from sklearn.metrics import confusion_matrix, classification_report

          # Assuming y_pred is the predicted labels for the test data
          y_pred = model.predict(X_test_FA)
          y_pred_binary = (y_pred > 0.5).astype(int)

          # Print confusion matrix
          print("Confusion Matrix:")
          print(confusion_matrix(y_test, y_pred_binary))

          # Print classification report
          print("\nClassification Report:")
          print(classification_report(y_test, y_pred_binary))

          2/2 [==============================] - 0s 2ms/step
          Confusion Matrix:
          [[29  0]
           [32  0]]

          Classification Report:
                        precision    recall  f1-score   support

                     0       0.48      1.00      0.64        29
                     1       0.00      0.00      0.00        32

              accuracy                           0.48        61
             macro avg       0.24      0.50      0.32        61
          weighted avg       0.23      0.48      0.31        61
```

Figure 46: Python snippet showing the evaluation matrix of the Experimental model

2.2.10.4 Original Model VS Experimental Model



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

           0       0.87      0.93      0.90        29
           1       0.93      0.88      0.90        32

    accuracy                           0.90        61
   macro avg       0.90      0.90      0.90        61
weighted avg       0.90      0.90      0.90        61
```

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

           0       0.48      1.00      0.64        29
           1       0.00      0.00      0.00        32

    accuracy                           0.48        61
   macro avg       0.24      0.50      0.32        61
weighted avg       0.23      0.48      0.31        61
```

Figure 47: Comparison Matrix(left: Original, right: Experimental)

The original model outperforms the experimental model across all key metrics. The experimental model, with its altered architecture and activation functions, struggles to classify instances, especially for class 1. The precision-recall trade-off in the experimental model appears skewed, resulting in a significantly lower F1 score and overall accuracy compared to the original model. These findings suggest that the changes made in the experimental model may have negatively impacted its ability to generalize and make accurate predictions on the given dataset.