Heart Disease Prediction using Deep Learning Neural Network Model

Abstract

Accurate heart disease prediction is critical for preventing life-threatening situations, while inaccurate predictions can have fatal consequences. In this report, a neural network prediction model is employed to analyse the heart disease dataset, which comprises 14 main attributes for the analysis. To enhance the dataset, a dimension reduction technique known as factor analysis is applied. Employing a deep learning approach, the model achieves an accuracy of 90.16%.

Introduction

In this report, we utilized the Cleveland Heart Disease dataset to predict heart disease, which comprises 13 columns representing dataset features. Our approach involved initial Exploratory Data Analysis (EDA) to uncover hidden patterns and correlations among variables. We conducted thorough pre-processing to address missing values, outliers, and unwanted data. Following standardization, we applied factor analysis to reduce dataset dimensions. Subsequently, a deep learning mechanism, specifically the Artificial Neural Network (ANN), was employed to achieve high accuracy in predicting heart attacks. The choice of ANN stems from its effectiveness in handling the dataset's complexity through manipulation and classification. The proposed method utilizes two hidden layers to extract quality features from the dataset. Finally, we have evaluated the performance of the model and optimize it with using different hidden layers or activation function and compare the result with the original model.

Figure 1: Flow diagram

A neural network constitutes an interconnected system of basic processing elements or nodes, drawing inspiration from the neural networks functions in the human brain. The network's

processing capability resides in the connection strengths, known as weights, established through adaptation or learning from a specific set of training patterns. Unlike traditional programming paradigms, where instructions are explicitly provided to the computer, neural networks operate differently. In this approach, the computer learns from observed data, autonomously devising solutions to the presented problem. ANN is used in various domains such as speech recognition, computer vision and natural language processing **(Nielsen, 2015)**.

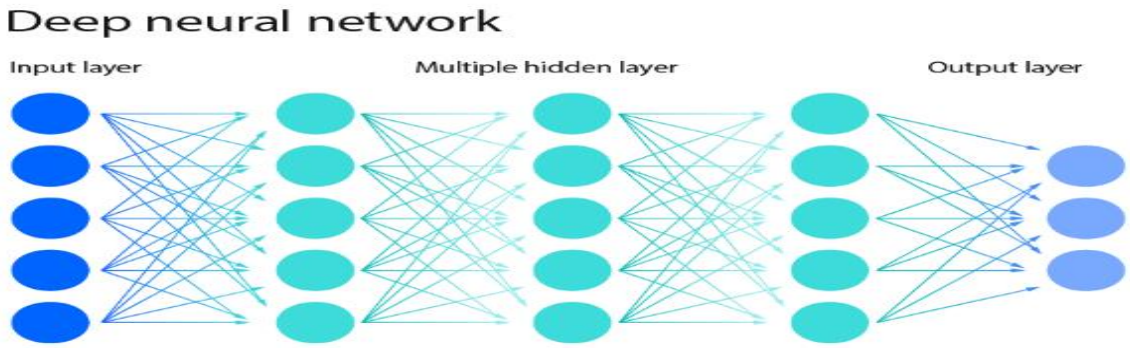

Figure 2: Deep Neural Network
Source: https://www.ibm.com/topics/neural-networks

Deep Neural Network (DNN) models have gained increased attention within the healthcare sector due to the growing complexities of healthcare data. DNN algorithms offer robust data analysis models capable of revealing concealed patterns and valuable insights from vast health datasets, a task that traditional analytics struggle to accomplish within a reasonable timeframe. Notably, Deep Learning (DL) techniques emerge as promising tools for pattern recognition within healthcare systems. In the medical and clinical domains, these advanced models prove beneficial for predictive tasks, leveraging their ability to discern intricate patterns and make accurate predictions, ultimately enhancing decision-making processes **(Shamshirband et al., 2021)**.

Exploratory Data Analysis

1. **Shape**

```
In [104]: #shape of the dataframe
          df.shape

Out[104]: (303, 14)
```

Figure 3: Shape of the dataframe

According to the figure 3, data frame has 303 rows and 14 columns.

2. Size

```
In [105]: #size of the dataframe
          df.size

Out[105]: 4242
```

Figure 4: Size of the dataframe

According to the figure 4, there are 4,242 values in the dataframe.

3. Description of attributes in the dataframe

There are 14 attributes in the dataframe.

Attributes	Description
1. age	Patient Age in Years
2. sex	Gender of Patient (0 = male; 1 = female)
3. cp_type	Chest pain type (0 = typical angina, 1 = atypical angina, 2 = non-anginal pain, 3 = asymptotic)
4. rest_bp	Resting blood pressure (in mm Hg), anything above 130-140 is typically cause for concern.
5. chol	Serum cholesterol (in mg/dl)
6. fbs	Fasting blood sugar > 120 mg/dl (1 = true; 0 = false), '>126' mg/dL signals diabetes.
7. rest_ecg	Resting electrocardiographic (ECG) results (0 = normal, 1 = having ST-T wave abnormality, 2 = left ventricular hyperthrophy)
8. thalach	Maximum heart rate achieved
9. exang	Exercise induced angina (1 = yes; 0 = no)
10. oldpeak	ST depression induced by exercise relative to rest
11. slope	The slope of the peak exercise ST segment (0 = upsloping : better heart rate with exercise, 1 = flat : minimal change, 2 = downsloping : signs of unhealthy heart)
12. ca	number of major vessels (0-3) colored by flourosopy
13. thal	(3 = normal; 6 = fixed defect; 7 = reversable defect)
14. target	Target column (1 = Yes; 0 = No)

Table 1: Description of attributes in the dataframe

According to the table 1, there are 14 attributes in the data frame. Attributes such as age, rest_bp, chol, thalach and old peak are continuous variables. However, attributes such as sex, fbs, cp_type, rest_ecg, exang, ca, slope, thal and target are categorical variables which have discrete or binary values.

4. Data type of the variables

```
In [79]: #data type of the variables
         df.dtypes

Out[79]: age         float64
         sex         float64
         cp_type     float64
         rest_bp     float64
         chol        float64
         fbs         float64
         rest_ecg    float64
         thalach     float64
         exang       float64
         oldpeak     float64
         slope       float64
         ca           object
         thal         object
         target        int64
         dtype: object
```

Figure 5: Data types of the variables

According to figure 5, all the variables in our data set are float or integers except 'ca' and 'thal'

which are object or string. It is required that all the values of the data set should be either float or integer. So, we will change the data type of the 'ca' and 'thal' attributes to integer.

5. Sample of the dataframe

```
In [106]: #10 random rows of the dataframe
          df.sample(10)
```

Out[106]:

	age	sex	cp_type	rest_bp	chol	fbs	rest_ecg	thalach	exang	oldpeak	slope	ca	thal	target
213	66.0	0.0	4.0	178.0	228.0	1.0	0.0	165.0	1.0	1.0	2.0	2.0	7.0	3
289	56.0	1.0	2.0	120.0	240.0	0.0	0.0	169.0	0.0	0.0	3.0	0.0	3.0	0
50	41.0	0.0	2.0	105.0	198.0	0.0	0.0	168.0	0.0	0.0	1.0	1.0	3.0	0
200	50.0	0.0	4.0	110.0	254.0	0.0	2.0	159.0	0.0	0.0	1.0	0.0	3.0	0
95	52.0	1.0	4.0	128.0	255.0	0.0	0.0	161.0	1.0	0.0	1.0	1.0	7.0	1
10	57.0	1.0	4.0	140.0	192.0	0.0	0.0	148.0	0.0	0.4	2.0	0.0	6.0	0
269	42.0	1.0	3.0	130.0	180.0	0.0	0.0	150.0	0.0	0.0	1.0	0.0	3.0	0
14	52.0	1.0	3.0	172.0	199.0	1.0	0.0	162.0	0.0	0.5	1.0	0.0	7.0	0
228	54.0	1.0	4.0	110.0	206.0	0.0	2.0	108.0	1.0	0.0	2.0	1.0	3.0	3
202	57.0	1.0	3.0	150.0	126.0	1.0	0.0	173.0	0.0	0.2	1.0	1.0	7.0	0

Figure 6: 10 random rows of the dataframe

6. Checking for missing values

```
In [80]: #checking missing values
         df.isnull().sum()
```

Out[80]:

```
age         0
sex         0
cp_type     0
rest_bp     0
chol        0
fbs         0
rest_ecg    0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Figure 7: Sum of missing values in all the columns

According to the figure 7, there are 0 missing values in our data set.

7. Checking for unwanted values

```
In [84]: unwanted_values = ['@', '', ',', '_', '?', '.']

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

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

Figure 8: Checking unwanted values in the data set

According to the figure 8, there is '?' unwanted value in 'ca' and 'thal' columns

8. Replacing the unwanted values

```
In [110]: #replace the '?' value in the columns 'ca' and 'thal' to their mode values because both are categorical columns.

          # Step 1: Convert '?' to NaN in the 'ca' and 'thal' columns
          new_df = df.copy()  # Create a new DataFrame to store modified values
          new_df['ca'] = new_df['ca'].replace('?',np.nan)
          new_df['thal'] = new_df['thal'].replace('?',np.nan)

          #Step 2: Convert columns to numeric (if not already)
          new_df['ca'] = pd.to_numeric(new_df['ca'])
          new_df['thal'] = pd.to_numeric(new_df['thal'])

          #Step 3: Calculate the mode values
          ca_mode = new_df['ca'].mode()[0]
          thal_mode = new_df['thal'].mode()[0]

          #Step 4: Replace NaN values with the mode values
          new_df['ca'] = new_df['ca'].fillna(ca_mode)
          new_df['thal'] = new_df['thal'].fillna(thal_mode)

          new_df
```

Out[110]:

	age	sex	cp_type	rest_bp	chol	fbs	rest_ecg	thalach	exang	oldpeak	slope	ca	thal	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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	2
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	3
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: Replacement process of unwanted values

According to the figure 9, since both columns are categorical columns, so we replaced the '?' values with the mode value.

9. Checking for outliers

```
new_df[['age','rest_bp','chol','thalach','oldpeak']].plot(kind='box', subplots=True, layout=(2,3), figsize=(8,8))
plt.show()
```

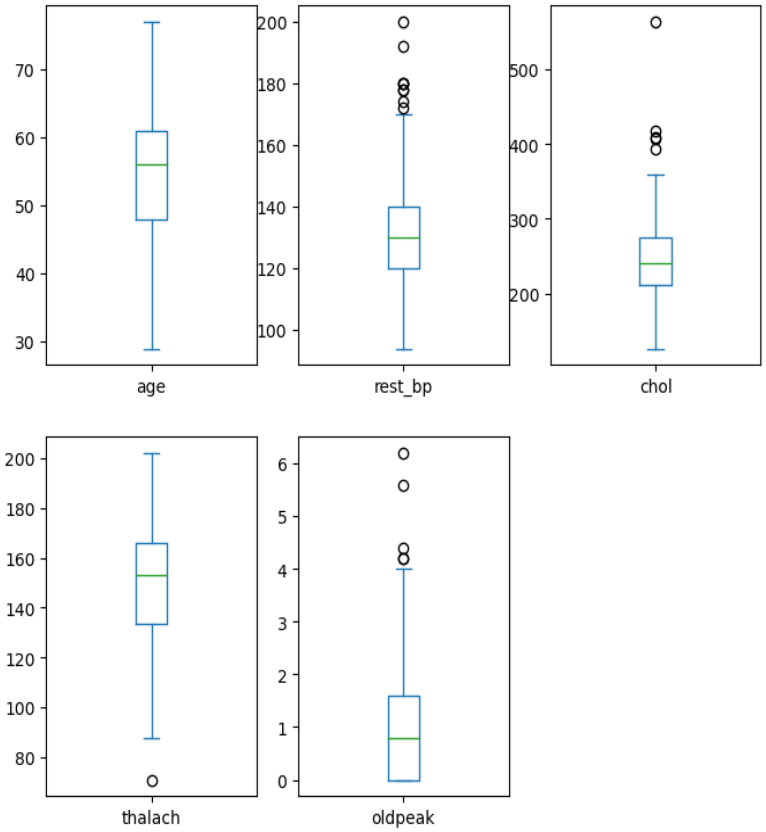

Figure 10: The box plot of continuous variables

According to figure 10, we can see with the help of box plot that rest_bp, chol, thalach and oldpeak columns have outliers.

10. Replacing outliers with the help of IQR method

```
# Select the columns of interest
columns_of_interest = ['age', 'rest_bp', 'chol', 'thalach', 'oldpeak']

# Calculate the IQR for each specified column
for column in columns_of_interest:
    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)

# Print DataFrame after replacing outliers
new_df
```

	age	sex	cp_type	rest_bp	chol	fbs	rest_ecg	thalach	exang	oldpeak	slope	ca	thal	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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	2
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	3
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 11: Replacement of outliers

According to figure 11, we applied the Interquartile Range (IQR) method to identify and replace potential outliers in each column. In IQR method we calculate the first and third quartiles (Q1 and Q3), determining the IQR as the difference between Q3 and Q1, and then identifying values outside the lower and upper bounds defined by 1.5 times the IQR. Outliers are replaced with the respective minimum and maximum values within the bounds.

11. Replacing and changing the data types to better outcome

```
#Replace the 2,3,4 values in target variable as 1 because all have same meaning.
df['target'].replace([2,3,4], 1,inplace =True)
df
```

	age	sex	cp_type	rest_bp	chol	fbs	rest_ecg	thalach	exang	oldpeak	slope	ca	thal	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

303 rows × 14 columns

Figure 12: Replacement of values in target column

```
In [57]: #changing datatypes of the variables
         # Assuming your DataFrame is named 'df'
         columns_to_convert = ['age', 'sex', 'cp_type', 'fbs', 'rest_ecg', 'exang', 'slope', 'ca', 'thal']

         # Clean non-integer values
         new_df[columns_to_convert] = new_df[columns_to_convert].apply(pd.to_numeric, errors='coerce')

         # Convert specified columns to integer
         new_df[columns_to_convert] = new_df[columns_to_convert].astype('Int64')

         new_df.dtypes

Out[57]: age          Int64
         sex          Int64
         cp_type      Int64
         rest_bp    float64
         chol       float64
         fbs          Int64
         rest_ecg     Int64
         thalach    float64
         exang        Int64
         oldpeak    float64
         slope        Int64
         ca           Int64
         thal         Int64
         target       int64
         dtype: object
```

Figure 13: Changing the datatypes of the variables

According to figure 12, there were 0,1,2,3 and 4 values. 1,2,3 and 4 are representing the same meaning so we replaced all the values to 1.
According to figure 13, we changed the data types of the variables to integer which had whole number values.

12. Descriptive Analysis

```
In [140]: new_df.describe()
```

Out[140]:

	age	sex	cp_type	rest_bp	chol	fbs	rest_ecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.0	303.0	303.0	303.000000	303.000000	303.0	303.0	303.000000	303.0	303.000000	303.0	303.0	303.0	303.000000
mean	54.438944	0.679868	3.158416	131.346535	245.584158	0.148515	0.990099	149.652640	0.326733	1.024422	1.60066	0.663366	4.722772	0.458746
std	9.038662	0.467299	0.960126	16.648749	47.558803	0.356198	0.994971	22.731735	0.469794	1.110127	0.616226	0.934375	1.938383	0.499120
min	29.0	0.0	1.0	94.000000	126.000000	0.0	0.0	84.750000	0.0	0.000000	1.0	0.0	3.0	0.000000
25%	48.0	0.0	3.0	120.000000	211.000000	0.0	0.0	133.500000	0.0	0.000000	1.0	0.0	3.0	0.000000
50%	56.0	1.0	3.0	130.000000	241.000000	0.0	1.0	153.000000	0.0	0.800000	2.0	0.0	3.0	0.000000
75%	61.0	1.0	4.0	140.000000	275.000000	0.0	2.0	166.000000	1.0	1.600000	2.0	1.0	7.0	1.000000
max	77.0	1.0	4.0	170.000000	371.000000	1.0	2.0	202.000000	1.0	4.000000	3.0	3.0	7.0	1.000000

Figure 14: Descriptive analysis of dataset

```
In [141]: plt.figure(figsize=(7,7))
          sns.heatmap(new_df.corr(),annot=True,fmt='.2f', cmap = 'coolwarm')
          plt.show()
```

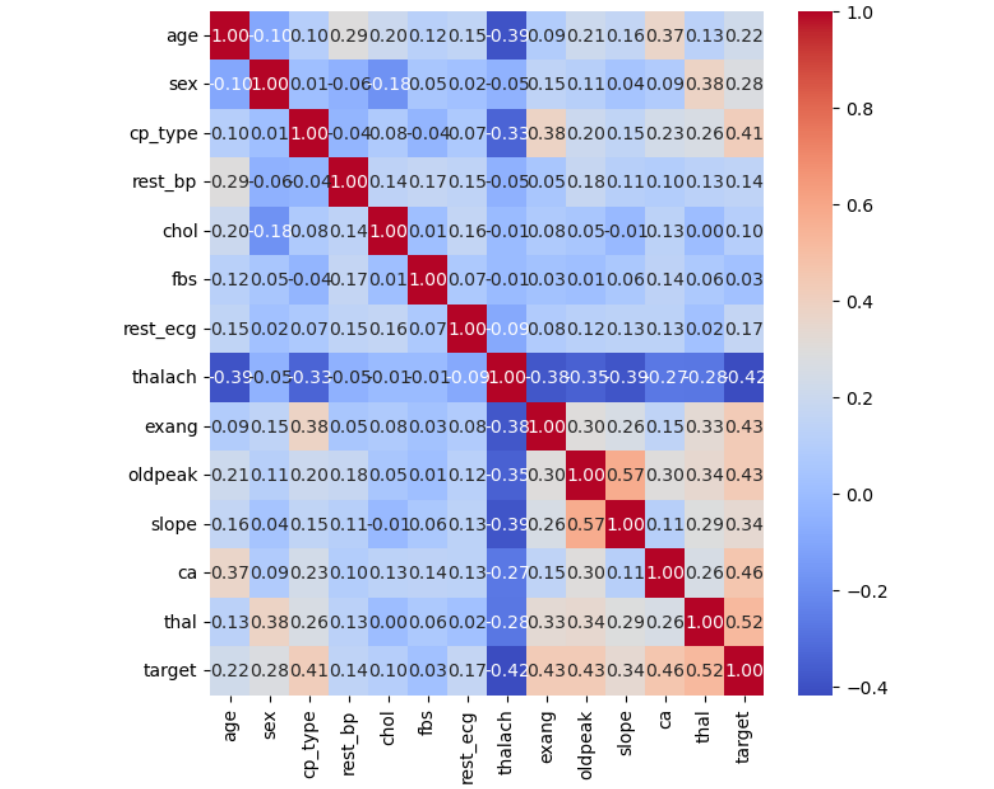

Figure 15: Correlation heatmap

According to figure 14, we can check the min, max, count, mean, std values of all the attributes. According to figure 15, we can see the correlation between the variables. Age, rest_bp, sex, fbs, chol and rest ecg have very less correlation with the heart disease. Thal has the highest positive correlation with heart disease and thalach has the highest negative correlation with the heart disease.

```
In [92]: new_df[['age','rest_bp','chol','thalach','oldpeak']].hist(figsize = (9,9), grid = True)
         plt.show()
```

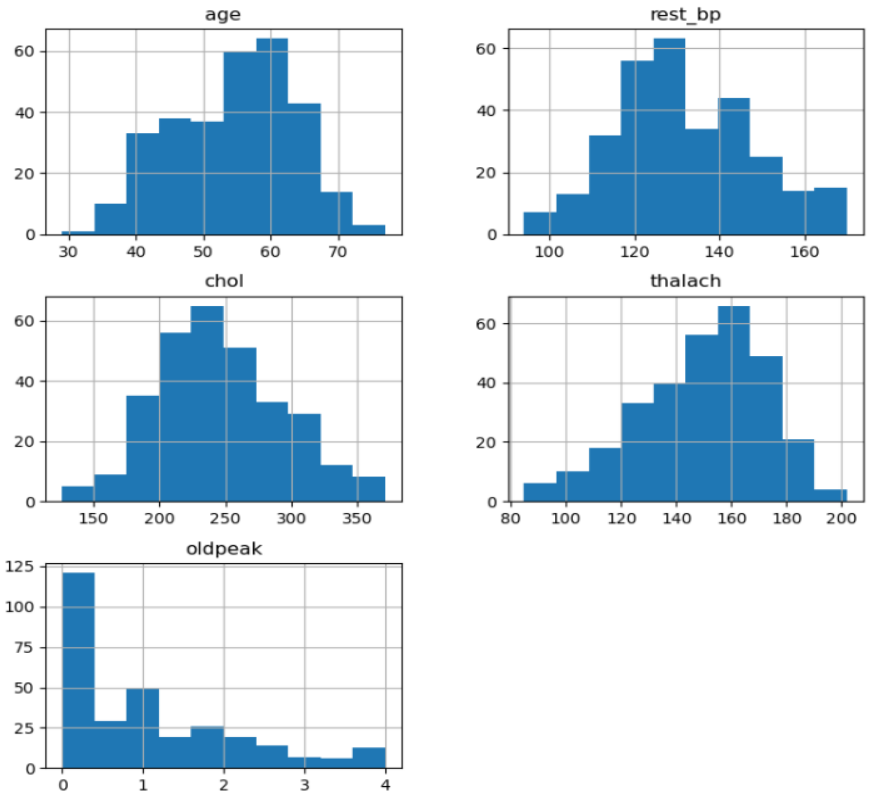

Figure 16: Histogram for continuous variables

In figure 16, we can see the distribution of the continuous variables in the dataset. Distribution of age group 55-65, resting bp 120-130, cholesterol 200-250, thalach 150-170 and oldpeak 0-0.5 have highest number of counts.

```
In [60]: # Pie chart for 'target'
target_counts = new_df['target'].value_counts()
plt.pie(target_counts, labels=target_counts.index, autopct='%1.1f%%')
plt.title('Distribution of Heart Disease')
plt.show()
```

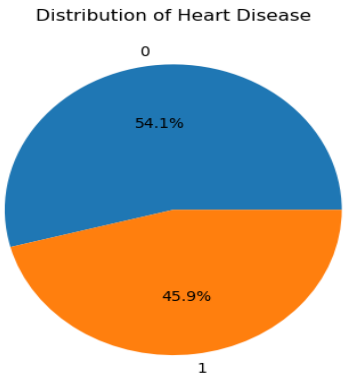

Figure 17: Distribution of heart disease

According to figure 17, 54.1% of the people don't have heart disease and 45.9% of the people have heart disease.

```
In [144]: print(pd.crosstab(new_df.sex, new_df.target))
sns.countplot(x = 'sex', data = new_df, hue = 'target' )
plt.title('Distribution of Heart Disease by Gender')
plt.legend(title='Target', labels=['No Heart Disease', 'Heart Disease'])
```

```
target   0    1
sex
0       72   25
1       92  114
```

```
Out[144]: <matplotlib.legend.Legend at 0x1d45edf36d0>
```

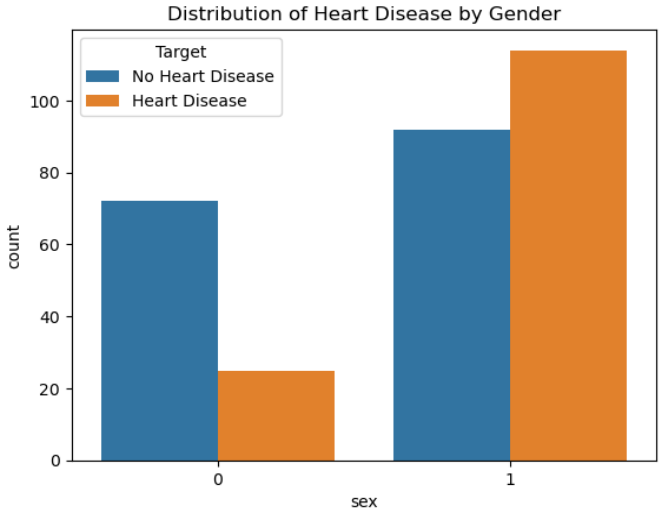

Figure 18: Distribution of heart disease by gender

According to figure 18, 72 females and 92 males do not have heart disease and 25 females and 114 males have heart disease.

```
In [148]: print(pd.crosstab(new_df.thal, new_df.target))
          sns.countplot(x = 'thal', data = new_df, hue = 'target' )
          plt.title('Distribution of Heart Disease by Thalassemia')
          plt.legend(title='Target', labels=['No Heart Disease', 'Heart Disease'])

target    0   1
thal
3       130  38
6         6  12
7        28  89

Out[148]: <matplotlib.legend.Legend at 0x1d45ee443d0>
```

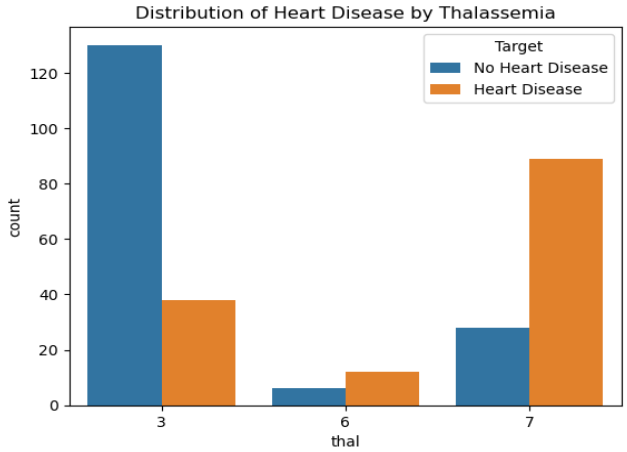

Figure 19: Distribution of heart disease by thalassemia

According to figure 19, 130 people have normal thalassemia, 6 people have fixed defect thalassemia, 28 people have reversable defect thalassemia don't have heart disease. 38 people have normal thalassemia, 12 people have fixed defect thalassemia, 89 people have reversable defect have heart disease. We can say that normal thalassemia has highest number of no heart disease people and reversable defect thalassemia has highest number of heart disease.

```
In [154]: print(pd.crosstab(new_df.slope, new_df.target))
          sns.countplot(x = 'slope', data = new_df, hue = 'target' )
          plt.title('Distribution of Heart Disease by slope of the peak exercise ST segment')
          plt.legend(title='Target', labels=['No Heart Disease', 'Heart Disease'])

target    0   1
slope
1       106  36
2        49  91
3         9  12

Out[154]: <matplotlib.legend.Legend at 0x1d4607e8ed0>
```

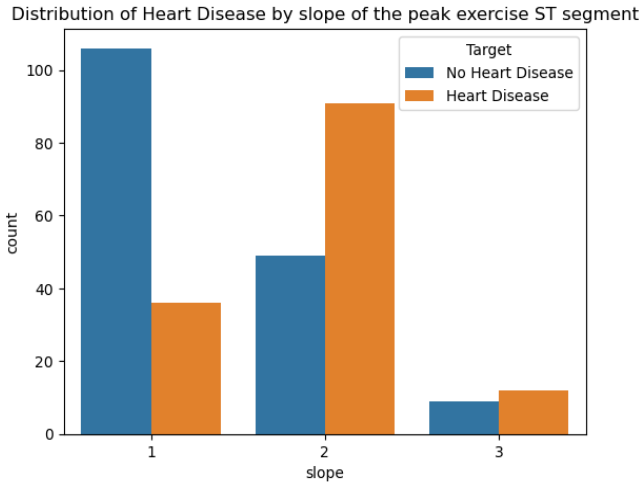

Figure 20: Distribution of heart disease by slope

As per Figure 20, individuals with a slope of 1 (indicating upsloping) exhibit the highest count of individuals without heart disease, totaling 106. Conversely, those with a slope of 2 (indicating a flat slope) demonstrate the highest count of individuals with heart disease, reaching a total of 91.

```
In [159]: print(pd.crosstab(new_df.exang, new_df.target))
          sns.countplot(x = 'exang', data = new_df, hue = 'target' )
          plt.title('Distribution of Heart Disease by exercise induced angina')
          plt.legend(title='Target', labels=['No Heart Disease', 'Heart Disease'])

target    0   1
exang
0       141  63
1        23  76

Out[159]: <matplotlib.legend.Legend at 0x1d45f53c050>
```

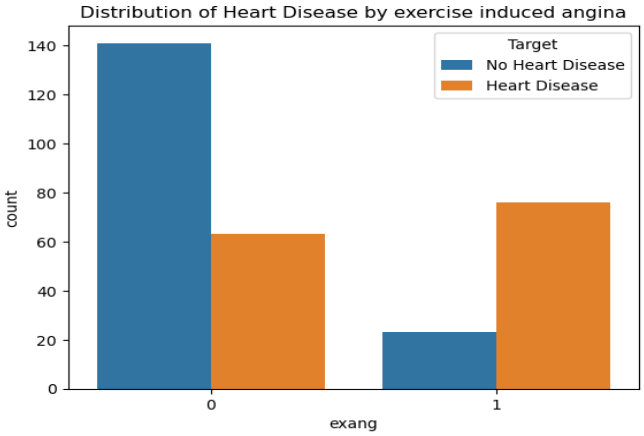

Figure 21: Distribution of heart disease by exang

According to Figure 21, individuals without exertional angina (exang) exhibit the highest count of individuals without heart disease, totaling 141. Conversely, individuals with exertional angina demonstrates the highest count of individuals with heart disease, reaching a total of 76.

13. Factor Analysis (Advance Question)

Factor Analysis serves as an unsupervised probabilistic machine learning algorithm focused on dimensionality reduction. Its primary objective is to consolidate correlated variables into fewer latent variables termed factors, sharing a mutual variance. The primary goal of Factor Analysis is to identify intercorrelations among a set of n variables using a limited number of common factors, creating meaningful categories for grouping the variables **(Agarwal, 2022)**.

```
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
```

	age	sex	cp_type	rest_bp	chol	fbs	rest_ecg	thalach	exang	oldpeak	slope	ca	thal
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 × 13 columns

Figure 22: Standard scaling of the data

In Figure 22, the dataset is undergoing standardization to ensure that all values are on the same scale.

```
In [66]: # Testing data for factor analysis by Barlett Test
         from factor_analyzer.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: 674.6332078600487
         p-value: 8.014883839379953e-96

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

         0.7033342096995896
```

Figure 23: Barlett test and KMO test

In figure 23, we applied Bartlett's and KMO tests to determine the appropriateness of the observed variables for factor analysis. The chi-square value helps evaluate whether the correlation matrix of the dataset is significantly different from an identity matrix. The p-value provides the probability of obtaining the observed chi-square
value by chance. A low p-value suggests that the variables are suitable for factor analysis. The KMO values range from 0 to 1, and a higher KMO value indicates better suitability for factor analysis. A KMO value greater than 0.6 suggests that the variables in the dataset have a substantial amount of common variance and are suitable for factor analysis.

```
In [68]: #Plotting graph to check how many number of factors are required to do factor analysis
from factor_analyzer import FactorAnalyzer
fa = FactorAnalyzer(rotation = None,impute = "drop",n_factors=X.shape[1])
fa.fit(X)
ev,_ = fa.get_eigenvalues()
plt.scatter(range(1,X.shape[1]+1),ev)
plt.plot(range(1,X.shape[1]+1),ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigen Value')
plt.grid()
```

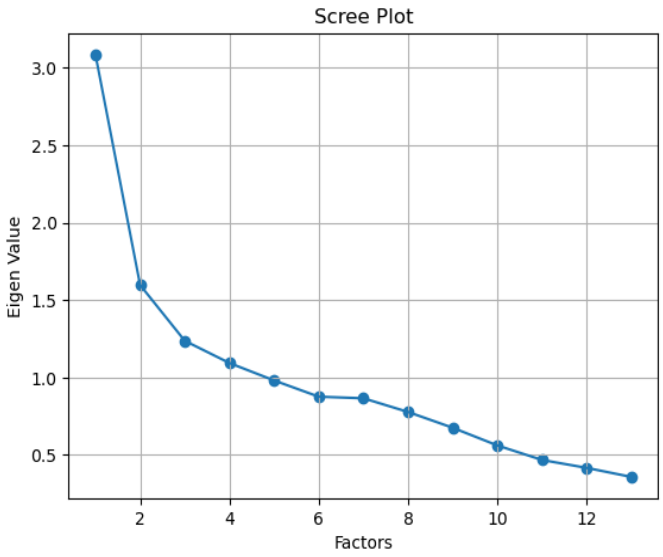

Figure 24: Graph for factor analysis

In Figure 24, we can see the 4 factors have eigen value more than 1.

```
In [69]: # instantiate the Factor Analyzer
         fa = FactorAnalyzer()

         # Fit the dataframe using Factor Analyzer
         fa.fit(X)

         # Identify the eigenvalues
         ev, v = fa.get_eigenvalues() #eigenvalues

         # display the eigenvalues
         ev

Out[69]: array([3.08560829, 1.59778581, 1.23769766, 1.09425116, 0.98162945,
                0.87678397, 0.86654342, 0.77806117, 0.67604621, 0.56189235,
                0.4680814 , 0.41703546, 0.35858364])
```

Figure 25: Eigen values of the factors

In Figure 25, we can see the 4 factors have eigen value more than 1.

```
In [170]: fa = FactorAnalyzer(n_factors=4,rotation='varimax')
          fa.fit(X)
          fa_load = pd.DataFrame(fa.loadings_,index=X.columns)

In [171]: print(pd.DataFrame(fa.get_factor_variance(),index=['Variance','Proportional Var','Cumulative Var']))

                               0         1         2         3
          Variance          1.497686  1.370756  1.177023  0.926035
          Proportional Var  0.115207  0.105443  0.090540  0.071233
          Cumulative Var    0.115207  0.220649  0.311190  0.382423

In [172]: def highlight_max(s):
              # Get 5 largest values of the column
              is_large = s.nlargest(4).values
              # Apply style is the current value is among the 5 biggest values
              return ['background-color: yellow' if v in is_large else '' for v in s]

          fa_load.sort_values(by=0,ascending=False).style.apply(highlight_max)
```

Out[172]:

	0	1	2	3
cp_type	0.672351	0.026899	-0.024521	-0.025807
exang	0.550600	0.172100	0.034116	0.139583
thal	0.381000	0.192074	0.166728	0.481089
ca	0.325101	0.021490	0.430060	0.114755
oldpeak	0.320528	0.499767	0.245430	0.153192
age	0.216180	0.065902	0.660164	-0.120492
slope	0.177059	0.977002	0.090504	0.031582
chol	0.100911	-0.052272	0.278162	-0.200927
rest_ecg	0.068873	0.082982	0.248114	-0.017688
sex	0.066606	0.005867	-0.057158	0.755782
rest_bp	-0.050152	0.100766	0.445785	0.004208
fbs	-0.063482	0.022008	0.253097	0.100444
thalach	-0.533751	-0.272259	-0.230826	-0.017019

Figure 26: Correlation of each variable in factors

In Figure 26, the variables with the highest correlation with each factor are highlighted, showcasing the top four influential variables within each factor.

```
In [174]: #Applying factor analysis
          from sklearn.decomposition import FactorAnalysis
          transformer = FactorAnalysis(n_components=4, random_state=32)
          X_train_FA = transformer.fit_transform(X_train)
          X_test_FA = transformer.transform(X_test)
```

Figure 27: Applying factor analysis method

In Figure 27, factor analysis is applied on the input data, resulting in a reduction of columns from 13 to 4 in our dataset.

Model Building and Training

1. Train and Test Dataset

```
In [173]: # 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 28: Train-test split of the dataset

In figure 28, the dataset split into training and testing sets is facilitated by the train_test_split function from the sklearn library. The dataset comprises features (X) and corresponding labels (y). By adjusting the test_size parameter, the proportion allocated for testing is specified, set to 20% (0.2) in this instance. The random_state parameter maintains reproducibility by fixing the random seed, ensuring consistent splits across executions. The outcome variables (X_train, X_test, y_train, and y_test) store the segregated sets for both features and labels.

```
In [177]: X_train_FA.shape, X_test_FA.shape
Out[177]: ((242, 4), (61, 4))
```

Figure 29: X train and X test shape

In figure 29, our dataset has been partitioned into X_train and X_test, with an 80% and 20% distribution, respectively.

2. Build a Neural Network Prediction Model

In [190]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam


# define a function to build the keras model
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(4, input_dim=4,activation='relu'))
    model.add(Dense(5, activation='relu'))
    model.add(Dense(5, activation='relu'))
    model.add(Dense(3, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # compile model
    adam = Adam(learning_rate=0.001)
    model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])
    return model

model = create_model()

print(model.summary())
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_70 (Dense)            (None, 4)                 20

 dense_71 (Dense)            (None, 5)                 25

 dense_72 (Dense)            (None, 5)                 30

 dense_73 (Dense)            (None, 3)                 18

 dense_74 (Dense)            (None, 1)                 4

=================================================================
Total params: 97 (388.00 Byte)
Trainable params: 97 (388.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Figure 30: Building a neural network model

In figure 30, we applied Keras library to construct a neural network tailored for binary classification. The model architecture encompasses multiple layers: the input layer incorporates 4 neurons utilizing the ReLU activation function, followed by three hidden layers containing 5, 5, and 3 neurons, all employing ReLU activation. The output layer, housing 1 neuron, utilizes the sigmoid activation function for binary classification. The model is configured with the Adam optimizer, binary cross-entropy loss function (suitable for binary classification), and accuracy as the evaluation metric. The model comprises a total of 97 parameters, each representing weights and biases. All parameters are trainable, indicating they will be updated during the training process.

3. Fitting the Prediction Model on the Training Dataset

```
In [58]: # fit the model on the training data
         history=model.fit(X_train_FA, y_train, validation_data=(X_test_FA, y_test),epochs=40, batch_size=10)
016
Epoch 35/40
25/25 [==============================] - 0s 3ms/step - loss: 0.4293 - accuracy: 0.8223 - val_loss: 0.3778 - val_accuracy: 0.9
016
Epoch 36/40
25/25 [==============================] - 0s 3ms/step - loss: 0.4281 - accuracy: 0.8264 - val_loss: 0.3776 - val_accuracy: 0.9
016
Epoch 37/40
25/25 [==============================] - 0s 3ms/step - loss: 0.4278 - accuracy: 0.8306 - val_loss: 0.3755 - val_accuracy: 0.9
016
Epoch 38/40
25/25 [==============================] - 0s 3ms/step - loss: 0.4261 - accuracy: 0.8264 - val_loss: 0.3738 - val_accuracy: 0.9
016
Epoch 39/40
25/25 [==============================] - 0s 3ms/step - loss: 0.4256 - accuracy: 0.8306 - val_loss: 0.3720 - val_accuracy: 0.9
016
Epoch 40/40
25/25 [==============================] - 0s 3ms/step - loss: 0.4240 - accuracy: 0.8264 - val_loss: 0.3718 - val_accuracy: 0.9
016
```

Figure 31: Fit prediction model on the training dataset

In figure 31, the model undergoes a fitting or training process on the provided training dataset. The fit method is employed to systematically refine the model's weights based on the training data. The epochs parameter specifies how many times the entire training dataset is processed through the neural network, and batch size dictates the number of samples utilized in each iteration for updating weights. The training progress is tracked, and the history object retains diverse metrics and loss values across epochs, facilitating subsequent visualization and analysis.

4. Evaluating the Performance of the Model

```
In [74]: # summarizing historical accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

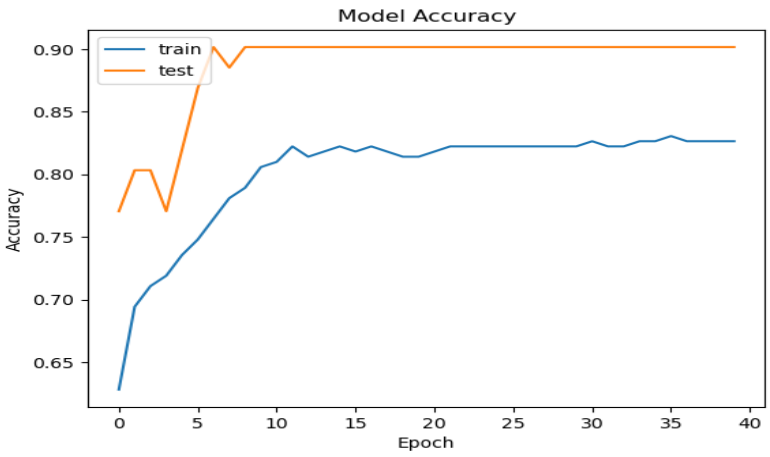

Figure 32: Prediction model accuracy visualization

In Figure 32, the graphical representation illustrates the evolution of model accuracy as the number of epochs increases. Both training and test dataset accuracies stabilize after 10 epochs.

```
In [75]: # summarize history for loss
         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.title('model loss')
         plt.ylabel('loss')
         plt.xlabel('epoch')
         plt.legend(['train', 'test'], loc='upper left')
         plt.show()
```

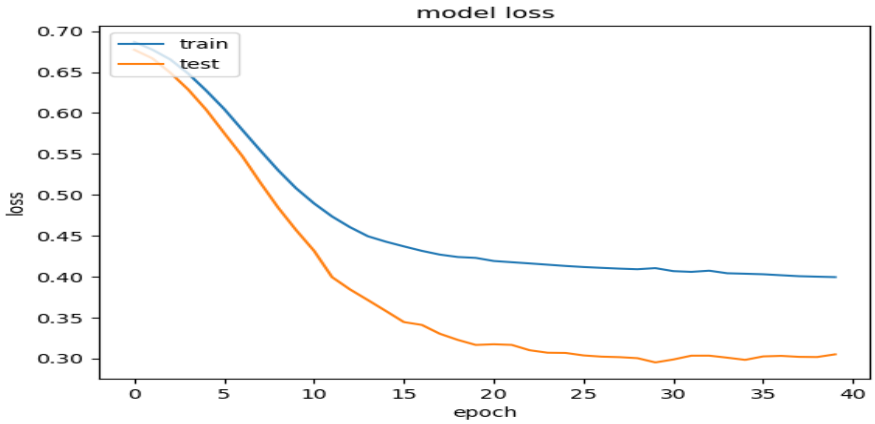

Figure 33: Prediction model loss visualization

In Figure 33, the graph visually depicts the progression of model loss as the number of epochs increase. Both training and test dataset loss values stabilize after 25 epochs

```
In [76]: y_log = model.predict(X_test_FA)

         2/2 [==============================] - 0s 3ms/step

In [77]: y_pred = np.where(y_log >0.5, 1, 0)

In [78]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test, y_pred)

Out[78]: 0.9016393442622951

In [117]: from sklearn import metrics
          print('Precision : ', np.round(metrics.precision_score(y_test, y_pred)*100,2))
          print('Accuracy : ', np.round(metrics.accuracy_score(y_test, y_pred)*100,2))
          print('Recall : ', np.round(metrics.recall_score(y_test, y_pred)*100,2))
          print('F1 score : ', np.round(metrics.f1_score(y_test, y_pred)*100,2))

          Precision :  93.33
          Accuracy :  90.16
          Recall :  87.5
          F1 score :  90.32
```

Figure 34: Performance of the prediction model

In Figure 34, we evaluated the performance of the model. Here's an explanation of each metric:

Precision (Positive Predictive Value): Precision represents the proportion of accurately predicted positive observations among the total predicted positives. In our model, it is 93.33%, indicating a high precision for positive predictions.

Accuracy: Accuracy is the ratio of correctly predicted observations to the total observations. The accuracy is 90.16%, suggesting a high overall correctness of predictions.

Recall (Sensitivity, True Positive Rate): Recall signifies the ratio of correctly predicted positive observations to the all observations in actual class. In our model, it is 87.5%, indicating a good ability to capture positive instances.

F1 Score: F1 score is the weighted average of precision and recall. The F1 score is 90.32%, combining precision and recall into a single metric.

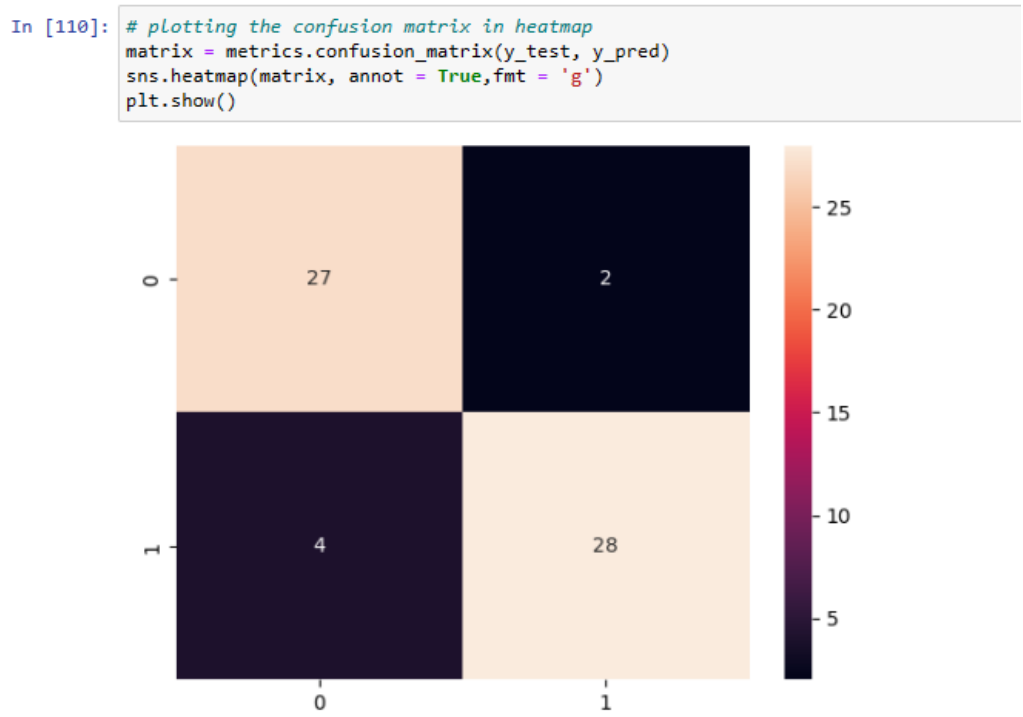

Figure 35: Confusion matrix in heatmap

In Figure 35, we can see there are 27 True negative values and 28 True positive values.

5. Experiment with relevant parameters of the Neural Network prediction model. (Advanced Question)

```
In [104]: # building an experimental neural network model
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam


# define a function to build the keras model
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(4, input_dim=4,activation='tanh'))
    model.add(Dense(3, activation='tanh'))
    model.add(Dense(2, activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))

    # compile model
    adam = Adam(learning_rate=0.001)
    model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
    return model

experiment_model = create_model()

print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 4)                 20

 dense_11 (Dense)            (None, 5)                 25

 dense_12 (Dense)            (None, 5)                 30

 dense_13 (Dense)            (None, 3)                 18

 dense_14 (Dense)            (None, 1)                 4

=================================================================
Total params: 97 (388.00 Byte)
Trainable params: 97 (388.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Figure 36: New experimental neural network model

In figure 36, we construct an experimental neural network model using the Keras library. The model is structured sequentially with various layers. The initial layer contains 4 neurons, employing the hyperbolic tangent (tanh) activation function. Subsequently, two hidden layers follow with 3 and 2 neurons, respectively, utilizing the tanh activation function. The last layer, comprising 1 neuron, incorporates the sigmoid activation function for binary classification. The model is compiled using the binary cross-entropy loss function, RMSprop optimizer and accuracy as the evaluation metric.

```
In [105]: # fit the experimental model on the training data
history_ex=experiment_model.fit(X_train_FA, y_train, validation_data=(X_test_FA, y_test),epochs=25, batch_size=10)

Epoch 1/25
25/25 [==============================] - 1s 8ms/step - loss: 0.7652 - accuracy: 0.2645 - val_loss: 0.7630 - val_accuracy: 0.229
5
Epoch 2/25
25/25 [==============================] - 0s 3ms/step - loss: 0.7394 - accuracy: 0.2975 - val_loss: 0.7365 - val_accuracy: 0.262
3
Epoch 3/25
25/25 [==============================] - 0s 2ms/step - loss: 0.7166 - accuracy: 0.3678 - val_loss: 0.7105 - val_accuracy: 0.426
2
Epoch 4/25
25/25 [==============================] - 0s 2ms/step - loss: 0.6956 - accuracy: 0.4380 - val_loss: 0.6861 - val_accuracy: 0.491
8
Epoch 5/25
25/25 [==============================] - 0s 2ms/step - loss: 0.6752 - accuracy: 0.5455 - val_loss: 0.6615 - val_accuracy: 0.639
3
```

Figure 37: Fitting of experiment model on the training dataset

In Figure 37, we fit the experimental model on the training dataset.

```
In [78]: y_pred_ex = np.where(y_log_ex >0.5, 1, 0)

In [79]: from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred_ex)

Out[79]: 0.8524590163934426

In [80]: from sklearn import metrics
print('Precision : ', np.round(metrics.precision_score(y_test, y_pred_ex)*100,2))
print('Accuracy : ', np.round(metrics.accuracy_score(y_test, y_pred_ex)*100,2))
print('Recall : ', np.round(metrics.recall_score(y_test, y_pred_ex)*100,2))
print('F1 score : ', np.round(metrics.f1_score(y_test, y_pred_ex)*100,2))
print('AUC : ', np.round(metrics.roc_auc_score(y_test, y_pred_ex)*100,2))

Precision :  89.66
Accuracy :  85.25
Recall :  81.25
F1 score :  85.25
AUC :  85.45
```

Figure 38: Performance of the experimental prediction model

According to the figure 38, we have achieved following result with the experimental model.

Precision: 89.66

Accuracy: 85.25

Recall: 81.25

F1 score: 85.25

```
In [83]: # plotting the experimental model confusion matrix in heatmap
matrix = metrics.confusion_matrix(y_test, y_pred_ex)
sns.heatmap(matrix, annot = True,fmt = 'g')
plt.show()
```

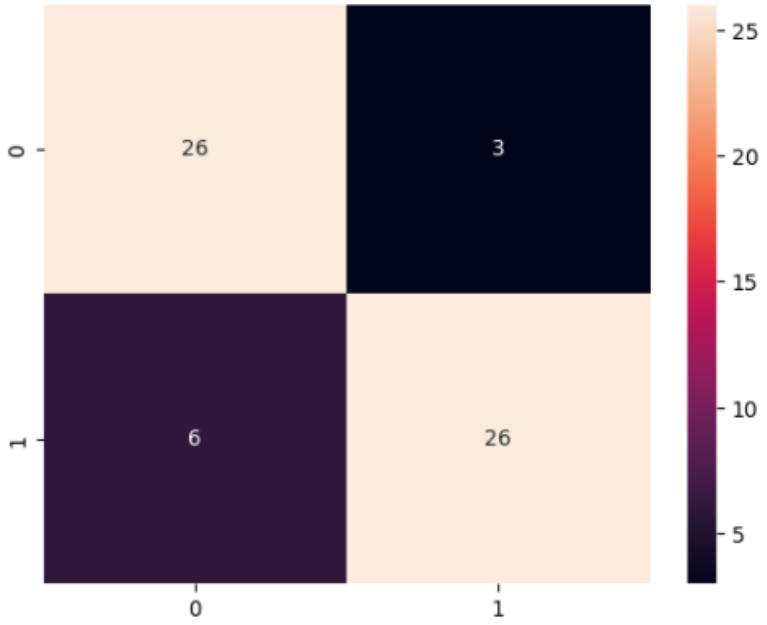

Figure 39: Confusion matrix heatmap of experimental model

According to figure 39, we can see that in our experimental model according to confusion matrix heatmap, 26 values are True negative and 26 values are True positive

6. Comparison of the Results of Both the Models

Parameters	Original Model	Experimental Model
Hidden Layers	3	2
Activation Function	Relu, Sigmoid	Tanh, Sigmoid
Optimizer	Adam	rmsprop
Precision	93.33	89.66
Accuracy	90.16	85.25
Recall	87.5	81.25
F1 Score	90.32	85.25

Table 2: Comparison between Original and Experimental Models

As indicated in Table 2, the original model outperformed the experimental model. The original model employed three hidden layers, whereas the experimental model utilized two hidden layers. Additionally, different activation functions and compile optimizers were applied in each model. To enhance model performance, optimization can be explored through varying the number of hidden layers, adjusting the number of neurons, and experimenting with different activation functions. This approach aims to identify the optimal combination of parameters that yields the highest accuracy.

7. Professional, Legal, Ethical and Social Issues

Category	Issues
Professional	Rigorous validation: Ensuring the proposed models and techniques undergo thorough validation to ensure reliability and accuracy. Feature selection: Proper consideration of feature selection techniques to enhance model performance and interpretability. Model comparison: Careful comparison of different models and methodologies for robust analysis.
Legal	Data privacy: Adhering to data protection laws and ensuring the ethical use of sensitive health-related information. Compliance: Ensuring compliance with regulations such as HIPAA in healthcare data analysis.
Ethical	Informed consent: Obtaining informed consent from individuals whose data is used, especially in medical research. Fairness: Ensuring fairness in model predictions and avoiding biases that may disproportionately impact certain groups. Transparency: Providing clear explanations of how models make predictions to build trust with users.
Social	Accessibility: Considering the accessibility of healthcare models to diverse populations, ensuring inclusivity. Health disparities: Addressing and minimizing potential biases that could contribute to health disparities.

Table 3: Professional, Legal, Ethical and Social Issues

Literature Review

1. Supervised Learning

(Khan et al., 2020) enhanced the diagnosis of heart diseases by leveraging smart healthcare technologies, particularly cloud computing and machine learning. The research utilizes electronic health records (EHRs) managed through cloud computing, reducing costs and ensuring data availability. They utilized Support Vector Machine (SVM) machine learning algorithm. The proposed intelligent cloud-based heart disease prediction system achieves a notable accuracy of 93.33%, outperforming previous approaches such as Neural Network, Logistic Regression and Framingham risk score. The conclusion emphasizes the superior performance of the proposed system during both training and validation phases, with accuracy rates of 95.31% and 93.33%, respectively. Future research recommendations could involve exploring the integration of additional advanced machine learning models and expanding the dataset for further validation.

(Patro et al., 2021) developed a predictive framework for heart disease, leveraging primary risk factors with various classifiers such as Bayesian Optimized Support Vector Machine (BO-SVM), Naïve Bayes, K-Nearest Neighbors (KNN). Utilizing the UCI Machine Repository's heart disease dataset, the research optimized algorithms to improve predictive accuracy. The Bayesian Optimized SVM demonstrates the highest performance with 93.3% accuracy, surpassing other methods. The research highlights the significance of optimization techniques in healthcare monitoring for early heart disease prediction. Future directions include collaborating with knowledge institutions, expanding the application to support medical decision-making, detecting various chronic diseases, and utilizing deep learning algorithms for increased accuracy.

2. Deep Learning Neural Network in Heart Disease Prediction

(Andari et al., 2023) introduced machine learning algorithms and the backpropagation technique of ANN to predict heart failure, a life-threatening condition caused by cardiovascular diseases. The study utilizes a real dataset with 12 features for 299 samples. Feature analysis is conducted, and three machine learning algorithms (Logistic Regression, K-Nearest Neighbors - k-NN, and Naïve Bayes) are implemented alongside the Multiple-Layer Perceptron (MLP) ANN with backpropagation. Hyperparameter tuning is performed for each algorithm to study accuracy, resulting in accuracies of 83%, 77%, 76%, and 84% for Logistic Regression, k-NN, Naïve Bayes, and ANN, respectively. The research emphasizes the importance of early detection in reducing risks and suggests that the proposed algorithm can serve as a supporting tool for physicians in predicting heart failure. Future research recommendations include feature extraction to optimize algorithm performance and efficiency. introduced machine learning algorithms and the backpropagation technique of ANN to predict heart failure, a life-threatening condition caused by cardiovascular diseases. The study employs a real dataset with 12 features for 299 samples, conducting feature analysis and implementing three machine learning algorithms (Logistic Regression, K-Nearest Neighbors - k-NN, and Naïve Bayes) alongside the Multiple-Layer Perceptron (MLP) ANN with backpropagation. Hyperparameter tuning is executed for each algorithm to investigate accuracy, resulting in accuracies of 83%, 77%, 76%, and 84% for Logistic Regression, k-NN, Naïve Bayes, and ANN, respectively. The research underscores the significance of early detection in risk reduction and suggests

that the proposed algorithm can serve as a supportive tool for physicians in predicting heart failure. Future research recommendations include exploring feature extraction techniques to enhance algorithm performance and efficiency.

(Xiao et al., 2020) predict heart disease using a novel deep neural network model called Deep Residual Neural Network (DRNN), which incorporates a deep residual structure. The research applied DRNN to the Heart Disease UCI dataset and achieved an accuracy of 95%, surpassing traditional machine learning methods such as Random Forest (83%), Decision Tree (68%), Logistic Regression (87%), KNN (60%), and Native Bayes (80%). The conclusion emphasizes the effectiveness of the deep residual structure in heart disease prediction and suggests future research directions, including exploring different residual structures and applying the depth residual structure to other classification problems.

(Ali et al., 2020) proposed a smart healthcare system that combines ensemble deep learning and feature fusion methods to enhance the accuracy of heart disease prediction by proposing. They collected heart patient data through wearable sensors and electronic medical tests. Information gain and conditional probability methods are used for feature selection and weighting, and an ensemble deep learning model is trained for heart disease prediction. The proposed system achieves an accuracy of 98.5%, outperforming traditional classifiers. The study suggests future research should focus on refining feature fusion using data mining techniques, developing methods for feature reduction, and addressing missing values and noise in large healthcare datasets.

(Dutta et al., 2020) proposed an efficient neural network with convolutional layers to classify highly imbalanced clinical data from the National Health and Nutritional Examination Survey (NHANES), aiming to predict coronary heart disease (CHD). The proposed CNN architecture achieves a classification accuracy of 77% for CHD presence and 81.8% for absence, demonstrating resilience to imbalanced data. Future research recommendations include exploring the inclusion of nutrition and dietary data for improved CHD prediction and addressing the debate between data augmentation and subsampling strategies in the context of imbalanced clinical datasets.

(Zhang et al., 2021) predicted heart disease accurately and efficiently using various body indicators. The proposed heart disease prediction model combines a feature selection method based on the LinearSVC algorithm with deep neural networks (DNN). Tested on a Kaggle heart disease dataset, the model achieves impressive results, with accuracy, recall, precision, and F1-score at 98.56%, 99.35%, 97.84%, and 0.983, respectively. Future research recommendations include adjusting DNN depth and parameters to enhance stability, exploring other deep learning optimization techniques, and considering additional optimization for improved performance.

(Li, 2023) introduced using a deep learning system to improve the prediction of all-cause mortality in heart failure (HF) patients. The proposed system employs an indicator vector to handle missing values, a convolutional neural network with various kernel sizes for feature extraction, and a multi-head self-attention mechanism to capture comprehensive channel information. The system is evaluated on a dataset from the MIMIC-III public database, comprising data from 10,311 patients. The deep learning model effectively predicts four death categories: within 30 days, within 180 days, within 365 days, and after 365 days. Future research recommendations could involve further validation on diverse datasets and exploration of additional interpretability techniques.

(Mehmood et al., 2021) proposed a method named CardioHelp that predicts the probability of cardiovascular disease using convolutional neural networks (CNN) for temporal data modeling to address the global issue of heart disease, particularly prevalent in developing countries in Africa and Asia. The study utilizes a dataset compiled by medical practitioners in African countries, incorporating 14 attributes to predict the presence of coronary heart disease (CHD). The proposed CNN model achieves an accuracy of 97%, outperforming state-of-the-art methods. The study discusses precision, recall, F1 score, and accuracy as performance metrics and emphasizes the importance of data preparation, classification using

CNN, and correlation analysis among independent variables. Future research is suggested to extend the prediction of major diseases beyond heart disease, such as cancer and brain-related diseases.

References

Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25, pp. 15-24). San Francisco, CA, USA: Determination press. https://www.ise.ncsu.edu/fuzzy-neural/wp-content

Shamshirband, S., Fathi, M., Dehzangi, A., Chronopoulos, A. T., & Alinejad-Rokny, H. (2021). A review on deep learning approaches in healthcare systems: Taxonomies, challenges, and open issues. *Journal of Biomedical Informatics, 113*, 103627-103627. https://doi.org/10.1016/j.jbi.2020.103627

Adnan Khan, M., Abbas, S., Atta, A., Ditta, A., Alquhayz, H., Farhan Khan, M., Atta-ur-Rahman, & Ali Naqvi, R. (2020). Intelligent cloud based heart disease prediction system empowered with supervised machine learning. *Computers, Materials & Continua, 65*(1), 139-151. https://doi.org/10.32604/cmc.2020.011416

Agarwal, (2022). A deep dive into factor analysis. Medium. https://medium.com/mlearning-ai/a-deep-dive-into-factor-analysis-d64e550c358f

Patro, S. P., Nayak, G. S., & Padhy, N. (2021). Heart disease prediction by using novel optimization algorithm: A supervised learning prospective. *Informatics in Medicine Unlocked, 26*, 100696. https://doi.org/10.1016/j.imu.2021.100696

Andari, B., Owayjan, M., Abou Haidar, G., & Achkar, R. (2023, October). Heart Failure Prediction Using Machine Learning and Artificial Neural Networks. In *2023 Seventh International Conference on Advances in Biomedical Engineering (ICABME)* (pp. 257-261). IEEE. https://ieeexplore.ieee.org/document/10293102

Mehmood, A., Iqbal, M., Mehmood, Z., Irtaza, A., Nawaz, M., Nazir, T., & Masood, M. (2021). Prediction of heart disease using deep convolutional neural networks. Arabian Journal for Science and Engineering, 46(4), 3409-3422. https://link.springer.com/article/10.1007/s13369-020-05105-1

Dutta, A., Batabyal, T., Basu, M., & Acton, S. T. (2020). An efficient convolutional neural network for coronary heart disease prediction. Expert Systems with Applications, 159, 113408. https://www.sciencedirect.com/science/article/abs/pii/S0957417420302323

Ali, F., El-Sappagh, S., Islam, S. R., Kwak, D., Ali, A., Imran, M., & Kwak, K. S. (2020). A smart healthcare monitoring system for heart disease prediction based on ensemble deep learning and feature fusion. Information Fusion, 63, 208-222. https://www.sciencedirect.com/science/article/pii/

Li, D., Fu, J., Zhao, J., Qin, J., & Zhang, L. (2023). A deep learning system for heart failure mortality prediction. *PloS One, 18*(2), e0276835-e0276835. https://doi.org/10.1371/journal.pone.0276835

Xiao, N., Zou, Y., Yin, Y., Liu, P., & Tang, R. (2020). DRNN: Deep residual neural network for heart disease prediction. *Journal of Physics. Conference Series, 1682*(1), 12065. https://doi.org/10.1088/1742-6596/1682/1/012065

Zhang, D., Chen, Y., Chen, Y., Ye, S., Cai, W., Jiang, J., ... & Chen, M. (2021). Heart disease prediction based on the embedded feature selection method and deep neural network. *Journal of Healthcare Engineering*, *2021*, 1-9. https://www.hindawi.com/journals/jhe/2021/6260022/

Appendix

1. Exploratory Data Analysis

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

#Assigning the column names to the dataframe.
columns =
['age','sex','cp_type','rest_bp','chol','fbs','rest_ecg','thalach','exang','oldpeak','slope','ca','thal','target'
]

#Read Cleveland csv file
df = pd.read_csv('processed.cleveland.data', encoding ='latin-1', names = columns)

#shape of the dataframe
df.shape

#size of the dataframe
df.size

#10 random rows of the dataframe
df.sample(10)

#info of the dataframe
df.info()

#data type of the variables
df.dtypes

#checking missing values
df.isnull().sum()

#Checking descriptive statistics of dataframe
```

```
df.describe()

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


#replace the '?' value in the columns 'ca' and 'thal' to their mode values because both are categorical
columns.
# Step 1: Convert '?' to NaN in the 'ca' and 'thal' columns
new_df = df.copy()  # Create a new DataFrame to store modified values
new_df['ca'] = new_df['ca'].replace('?',np.nan)
new_df['thal'] = new_df['thal'].replace('?',np.nan)
#Step 2: Convert columns to numeric (if not already)
new_df['ca'] = pd.to_numeric(new_df['ca'])
new_df['thal'] = pd.to_numeric(new_df['thal'])
#Step 3: Calculate the mode values
ca_mode = new_df['ca'].mode()[0]
thal_mode = new_df['thal'].mode()[0]
#Step 4: Replace NaN values with the mode values
new_df['ca'] = new_df['ca'].fillna(ca_mode)
new_df['thal'] = new_df['thal'].fillna(thal_mode)
new_df

# Check for outliers
new_df[['age','rest_bp','chol','thalach','oldpeak']].plot(kind='box', subplots=True, layout=(2,3),
figsize=(8,8))
plt.show()

# Replace the outliers
columns_of_interest = ['age', 'rest_bp', 'chol', 'thalach', 'oldpeak']
# Calculate the IQR for each specified column
for column in columns_of_interest:
    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)

# Print DataFrame after replacing outliers
new_df

#changing datatypes of the variables
# Assuming your DataFrame is named 'df'
```

```
columns_to_convert = ['age', 'sex', 'cp_type', 'fbs', 'rest_ecg', 'exang', 'slope', 'ca', 'thal']

# Clean non-integer values
new_df[columns_to_convert] = new_df[columns_to_convert].apply(pd.to_numeric, errors='coerce')

# Convert specified columns to integer
new_df[columns_to_convert] = new_df[columns_to_convert].astype('Int64')
new_df.dtypes

#Replace the 2,3,4 values in target variable as 1 because all have same meaning.
new_df['target'].replace([2,3,4], 1,inplace =True)
new_df


# histogram graph of continuous variables
new_df[['age','rest_bp','chol','thalach','oldpeak']].hist(figsize = (9,9), grid = True)
plt.show()

# Correlation heatmap
plt.figure(figsize=(7,7))
sns.heatmap(new_df.corr(),annot=True,fmt='.2f', cmap = 'coolwarm')
plt.show()

# Pie chart for 'target'
target_counts = new_df['target'].value_counts()
plt.pie(target_counts, labels=target_counts.index, autopct='%1.1f%%')
plt.title('Distribution of Heart Disease')
plt.show()

# count plot for distribution of heart disease by gender
print(pd.crosstab(new_df.sex, new_df.target))
sns.countplot(x = 'sex', data = new_df, hue = 'target' )
plt.title('Distribution of Heart Disease by Gender')
plt.legend(title='Target', labels=['No Heart Disease', 'Heart Disease'])

print(pd.crosstab(new_df.thal, new_df.target))
sns.countplot(x = 'thal', data = new_df, hue = 'target' )
plt.title('Distribution of Heart Disease by Thalassemia')
plt.legend(title='Target', labels=['No Heart Disease', 'Heart Disease'])

print(pd.crosstab(new_df.slope, new_df.target))
sns.countplot(x = 'slope', data = new_df, hue = 'target' )
plt.title('Distribution of Heart Disease by slope of the peak exercise ST segment')
plt.legend(title='Target', labels=['No Heart Disease', 'Heart Disease'])

print(pd.crosstab(new_df.exang, new_df.target))
sns.countplot(x = 'exang', data = new_df, hue = 'target' )
plt.title('Distribution of Heart Disease by exercise induced angina')
plt.legend(title='Target', labels=['No Heart Disease', 'Heart Disease'])
```

2. Factor Analysis

```
# splitting data in X and y
X = new_df.iloc[:,0:-1]
y = new_df.iloc[:,-1]
X.shape, y.shape

# Scaling the data
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



# Testing data for factor analysis by Barlett Test
from factor_analyzer.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}')

# Testing data for factor analysis by KMO Test
from factor_analyzer.factor_analyzer import calculate_kmo
kmo_vars,kmo_model = calculate_kmo(X)
print(kmo_model)

#Plotting graph to check how many number of factors are required to do factor analysis
from factor_analyzer import FactorAnalyzer
fa = FactorAnalyzer(rotation = None,impute = "drop",n_factors=X.shape[1])
fa.fit(X)
ev,_ = fa.get_eigenvalues()
plt.scatter(range(1,X.shape[1]+1),ev)
plt.plot(range(1,X.shape[1]+1),ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigen Value')
plt.grid()

# instantiate the Factor Analyzer
fa = FactorAnalyzer()
# Fit the dataframe using Factor Analyzer
fa.fit(X)
# Identify the eigenvalues
ev, v = fa.get_eigenvalues() #eigenvalues
# display the eigenvalues
Ev

fa = FactorAnalyzer(n_factors=4,rotation='varimax')
fa.fit(X)
fa_load = pd.DataFrame(fa.loadings_,index=X.columns)

#Correlation shows by each variable in terms of factor analysis
```

```
def highlight_max(s):
   # Get 5 largest values of the column
   is_large = s.nlargest(4).values
   # Apply style is the current value is among the 5 biggest values
   return ['background-color: yellow' if v in is_large else '' for v in s]
fa_load.sort_values(by=0,ascending=False).style.apply(highlight_max)

# 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)

#Applying factor analysis
from sklearn.decomposition import FactorAnalysis
transformer = FactorAnalysis(n_components=4, random_state=32)
X_train_FA = transformer.fit_transform(X_train)
X_test_FA = transformer.transform(X_test)
```

3. Neural Network Model Building

```
# builing a neural network model
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam


# define a function to build the keras model
def create_model():
   # create model
   model = Sequential()
   model.add(Dense(4, input_dim=4,activation='relu'))
   model.add(Dense(5, activation='relu'))
   model.add(Dense(5, activation='relu'))
   model.add(Dense(3, activation='relu'))
   model.add(Dense(1, activation='sigmoid'))

   # compile model
   adam = Adam(learning_rate=0.001)
   model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=['accuracy'])
   return model

model = create_model()

print(model.summary())

# fit the model on the training data
history=model.fit(X_train_FA, y_train, validation_data=(X_test_FA, y_test),epochs=40, batch_size=10)

# summarizing historical accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
```

```
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

#Model prediction
y_log = model.predict(X_test_FA)
y_pred = np.where(y_log >0.5, 1, 0)

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)

#Performance of the model
from sklearn import metrics
print('Precision : ', np.round(metrics.precision_score(y_test, y_pred)*100,2))
print('Accuracy : ', np.round(metrics.accuracy_score(y_test, y_pred)*100,2))
print('Recall : ', np.round(metrics.recall_score(y_test, y_pred)*100,2))
print('F1 score : ', np.round(metrics.f1_score(y_test, y_pred)*100,2))

# plotting the confusion matrix in heatmap
matrix = metrics.confusion_matrix(y_test, y_pred)
sns.heatmap(matrix, annot = True,fmt = 'g')
plt.show()
```

4. Experiment with Neural Network Model

```
# builing an experimental neural network model
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam


# define a function to build the keras model
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(4, input_dim=4,activation='tanh'))
    model.add(Dense(3, activation='tanh'))
    model.add(Dense(2, activation='tanh'))
    model.add(Dense(1, activation='sigmoid'))

    # compile model
    adam = Adam(learning_rate=0.001)
```

```
    model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
    return model

experiment_model = create_model()

print(model.summary())

# fit the experimental model on the training data
history_ex=experiment_model.fit(X_train_FA, y_train, validation_data=(X_test_FA, y_test),epochs=25,
batch_size=10)

y_log_ex = experiment_model.predict(X_test_FA)
y_pred_ex = np.where(y_log_ex >0.5, 1, 0)
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred_ex)

from sklearn import metrics
print('Precision : ', np.round(metrics.precision_score(y_test, y_pred_ex)*100,2))
print('Accuracy : ', np.round(metrics.accuracy_score(y_test, y_pred_ex)*100,2))
print('Recall : ', np.round(metrics.recall_score(y_test, y_pred_ex)*100,2))
print('F1 score : ', np.round(metrics.f1_score(y_test, y_pred_ex)*100,2))

# plotting the experimental model confusion matrix in heatmap
matrix = metrics.confusion_matrix(y_test, y_pred_ex)
sns.heatmap(matrix, annot = True,fmt = 'g')
plt.show()
```