LAB MANUAL

MACHINE LEARNING

Name：Ayesha Imran

Registration no：2023-BS-AI-011

Submitted to : Sir Saeed

Degree : BSAI-4A

DEPARTMENT OF COMPUTER SCIENCE

LIST OF CONTENT

PROJECT 1 : FOOD DILEVERY TIME PREDICTION USING REGRESSION... 04

Summary **04**
Objective **04**
What this project is use for **04**
Abstract.. **04**
Dataset description **05**
Import libraries.. **05**
Loading datase **05**
Data exploration **05**
Data cleaning **08**
Removing dublicates **08**
Outlier detection and removal **09**
Data transformation..**11**
Normalization **11**
Standardization **12**
Linear regresiion **13**
Actual Vs predicted graph **14**
Actual Vs predicted delivery time **15**
Conclusion **16**

PROJECT 2 : HEART DISEASE PREDICTION USING CLASSIFICATION 17

Summary **17**
Objectives **17**
What this project is used for **17**
Abstract **18**
Dataset description **18**
Importing libraries **18**
Reading data **19**
Exploring data **19**
Feature names **23**
Missing values **23**
Separate categorical and numerical values **24**
Checking dublicate values..**26**
Visualizing categorical features **27**
Outlier detection and removal **28**
Data transformation **29**

Normalization …………………………………………………………………**29**
Standardization…………………………………………………………………**30**
Feature selection…………………………………………………………………**30**
Splitting…………………………………………………………………………..**31**
Decision tree……………………………………………………………………..**31**
Classification report………………………………………………………………**31**
Confusion matrix………………………………………………………………..**32**
Random forest ……………………………………………………………………...**33**
Classification report………………………………………………………………**33**
Confusion matrix………………………………………………………………..**34**
ANN…………………………………………………………………………………...**35**
Classification report………………………………………………………………**37**
Confusion matrix………………………………………………………………..**37**
Conclusion ……………………………………………………………………....**38**

Project 1: Food Delivery Time Prediction Using Regression

Summary

This project focuses on predicting food delivery times using a regression model. It involves comprehensive data analysis, including data loading, exploration, cleaning, and preprocessing steps. The dataset includes various factors that influence delivery time, such as distance, weather conditions, traffic levels, time of day, vehicle type, preparation time, and courier experience.

Objective

The primary objective of this project is to develop and implement a robust regression model capable of accurately predicting food delivery times. This involves identifying key factors influencing delivery duration and leveraging machine learning techniques to build a predictive model.

What this Project is Used For

This project can be used by food delivery companies to:

- **Optimize Delivery Logistics:** Predict delivery times more accurately, allowing for better route planning and resource allocation.
- **Improve Customer Satisfaction:** Provide customers with more precise estimated delivery times, leading to a better user experience.
- **Enhance Operational Efficiency:** Identify bottlenecks and areas for improvement in the delivery process by understanding the impact of different variables on delivery time.
- **Strategic Decision-Making:** Inform business decisions related to pricing, courier management, and service expansion based on predictive insights.

Abstract

This project presents a regression analysis aimed at predicting food delivery times. Utilizing a dataset encompassing various delivery parameters such as distance, weather, traffic, time of day, vehicle type, food preparation time, and courier experience, the study employs a systematic approach to data preprocessing, including handling missing values, removing duplicates, and outlier detection. The ultimate goal is to build a predictive model that can accurately estimate delivery duration's, thereby enabling food

delivery platforms to optimize their operations, enhance efficiency, and improve overall customer satisfaction through reliable delivery time predictions.

Dataset description:

This dataset contains food delivery records, capturing key factors influencing delivery times. It includes attributes such as **distance traveled, weather conditions, traffic level, time of day, and vehicle type**, all of which impact delivery efficiency. Additionally, courier experience and preparation time are recorded, offering insights into operational factors affecting service speed. The target variable, **Delivery_Time_min**, represents the actual time taken for each order's delivery. Given your proficiency in preprocessing and model training, this dataset is well-suited for regression analysis or predictive modeling

IMPORTING LIBRARIES

```
[1]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
```

- **pandas (pd):** Used for data manipulation and analysis.
- **numpy (np):** Provides numerical operations and array handling.
- **matplotlib.pyplot (plt):** Enables data visualization through plots and graphs.

LOADING DATASET

```
[3]: df = pd.read_csv("C:/Users/Ayaan/Desktop/Food_Delivery_Times.csv")
     df.head()
```

[3]:

	Order_ID	Distance_km	Weather	Traffic_Level	Time_of_Day	Vehicle_Type	Preparation_Time_min	Courier_Experience_yrs	Delivery_Time_min
0	522	7.93	Windy	Low	Afternoon	Scooter	12	1.0	43
1	738	16.42	Clear	Medium	Evening	Bike	20	2.0	84
2	741	9.52	Foggy	Low	Night	Scooter	28	1.0	59
3	661	7.44	Rainy	Medium	Afternoon	Scooter	5	1.0	37
4	412	19.03	Clear	Low	Morning	Bike	16	5.0	68

- **pd.read_csv(...):** Reads the CSV file located at **"C:/Users/Ayaan/Desktop/Food_Delivery_Times.csv"** into a DataFrame.
- **df.head():** Displays the first five rows of the DataFrame for an initial overview of the dataset.

DATA EXPLORATION

```
df.sample()
```

	Order_ID	Distance_km	Weather	Traffic_Level	Time_of_Day	Vehicle_Type	Preparation_Time_min	Courier_Experience_yrs	Delivery_Time_min
82	276	10.28	Clear	Medium	Evening	Scooter	29	3.0	71

- **df.sample():** Returns a randomly selected row from the DataFrame.

```
df.tail()
```

	Order_ID	Distance_km	Weather	Traffic_Level	Time_of_Day	Vehicle_Type	Preparation_Time_min	Courier_Experience_yrs	Delivery_Time_min
995	107	8.50	Clear	High	Evening	Car	13	3.0	54
996	271	16.28	Rainy	Low	Morning	Scooter	8	9.0	71
997	861	15.62	Snowy	High	Evening	Scooter	26	2.0	81
998	436	14.17	Clear	Low	Afternoon	Bike	8	0.0	55
999	103	6.63	Foggy	Low	Night	Scooter	24	3.0	58

- **df.tail():** Displays the last five rows of the DataFrame.

```
df.shape
```

```
(1000, 9)
```

```
df.columns
```

```
Index(['Order_ID', 'Distance_km', 'Weather', 'Traffic_Level', 'Time_of_Day',
       'Vehicle_Type', 'Preparation_Time_min', 'Courier_Experience_yrs',
       'Delivery_Time_min'],
      dtype='object')
```

- **`df.shape`:** Returns the number of rows and columns in the DataFrame as a tuple `(rows, columns)`.
- **`df.columns`:** Lists all column names in the DataFrame, showing available data fields.

```
[12]: df.isnull().any()
```

```
[12]: Order_ID                  False
      Distance_km               False
      Weather                    True
      Traffic_Level              True
      Time_of_Day                True
      Vehicle_Type              False
      Preparation_Time_min      False
      Courier_Experience_yrs     True
      Delivery_Time_min         False
      dtype: bool
```

- **df.isnull().any():** Checks if there are any missing (NaN) values in each column.
- Returns a Boolean value (True if missing values exist, False if none).

```
[13]: df.describe()
```

[13]:

	Order_ID	Distance_km	Preparation_Time_min	Courier_Experience_yrs	Delivery_Time_min
count	1000.000000	1000.000000	1000.000000	970.000000	1000.000000
mean	500.500000	10.059970	16.982000	4.579381	56.732000
std	288.819436	5.696656	7.204553	2.914394	22.070915
min	1.000000	0.590000	5.000000	0.000000	8.000000
25%	250.750000	5.105000	11.000000	2.000000	41.000000
50%	500.500000	10.190000	17.000000	5.000000	55.500000
75%	750.250000	15.017500	23.000000	7.000000	71.000000
max	1000.000000	19.990000	29.000000	9.000000	153.000000

- **df.describe():** Generates summary statistics for numerical columns in the DataFrame.
- Provides metrics like count, mean, standard deviation, min, max, and quartiles.

```
[14]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Order_ID                1000 non-null   int64  
 1   Distance_km             1000 non-null   float64
 2   Weather                 970 non-null    object 
 3   Traffic_Level           970 non-null    object 
 4   Time_of_Day             970 non-null    object 
 5   Vehicle_Type            1000 non-null   object 
 6   Preparation_Time_min    1000 non-null   int64  
 7   Courier_Experience_yrs  970 non-null    float64
 8   Delivery_Time_min       1000 non-null   int64  
dtypes: float64(2), int64(3), object(4)
memory usage: 70.4+ KB
```

- **df.info():** Provides a concise summary of the DataFrame's structure.
- Displays column names, non-null counts, data types, and memory usage.

DATA CLEANING

```
[15]: numeric_cols = df.select_dtypes(include=[np.number])
      non_numeric_cols = df.select_dtypes(exclude=[np.number])

[16]: numeric_cols.fillna(numeric_cols.mean(), inplace=True)
      non_numeric_cols.fillna(non_numeric_cols.mode().iloc[0], inplace=True)

[17]: df = pd.concat([numeric_cols, non_numeric_cols], axis=1)
```

- **numeric_cols:** Selects all columns with numeric data types (int, float) from the DataFrame.
- **non_numeric_cols:** Selects all columns that are not numeric (object, category, etc.).
- **`numeric_cols.fillna(numeric_cols.mean(), inplace=True)`**: Fills missing values in numeric columns with their respective mean values.
- **`non_numeric_cols.fillna(non_numeric_cols.mode().iloc[0], inplace=True)`**: Fills missing values in non-numeric columns with their most frequent value (mode).
- Helps maintain dataset integrity by replacing `NaN` values with appropriate statistical substitutes.
- **pd.concat([...], axis=1):** Combines numeric_cols and non_numeric_cols along columns (axis=1).
- Reconstructs the DataFrame after processing missing values separately for numeric and non-numeric data.

```
[18]: print(df.isnull().sum())

      Order_ID                0
      Distance_km             0
      Preparation_Time_min    0
      Courier_Experience_yrs  0
      Delivery_Time_min       0
      Weather                 0
      Traffic_Level           0
      Time_of_Day             0
      Vehicle_Type            0
      dtype: int64
```

- **df.isnull().sum():** Counts the number of missing (NaN) values in each column of the DataFrame.
- Helps identify how many null values exist per feature after preprocessing.

REMOVING DUBLICATES

```
[19]: df.drop_duplicates(inplace=True)

      df = df.loc[:, ~df.columns.str.contains("Unnamed")]

      df.info()

      <class 'pandas.core.frame.DataFrame'>
      RangeIndex: 1000 entries, 0 to 999
      Data columns (total 9 columns):
       #   Column                 Non-Null Count  Dtype
      ---  ------                 --------------  -----
       0   Order_ID               1000 non-null   int64
       1   Distance_km            1000 non-null   float64
       2   Preparation_Time_min   1000 non-null   int64
       3   Courier_Experience_yrs 1000 non-null   float64
       4   Delivery_Time_min      1000 non-null   int64
       5   Weather                1000 non-null   object
       6   Traffic_Level          1000 non-null   object
       7   Time_of_Day            1000 non-null   object
       8   Vehicle_Type           1000 non-null   object
      dtypes: float64(2), int64(3), object(4)
      memory usage: 70.4+ KB
```

- **`df.drop_duplicates(inplace=True)`:** Removes duplicate rows from the DataFrame to ensure unique records.
- **`df.loc[:,** ~df.columns.str.contains("Unnamed")]`: Drops columns containing `"Unnamed"` in their names, often residuals from file imports.
- **`df.info()`:** Provides a summary of the cleaned dataset, showing column names, data types, and missing values.

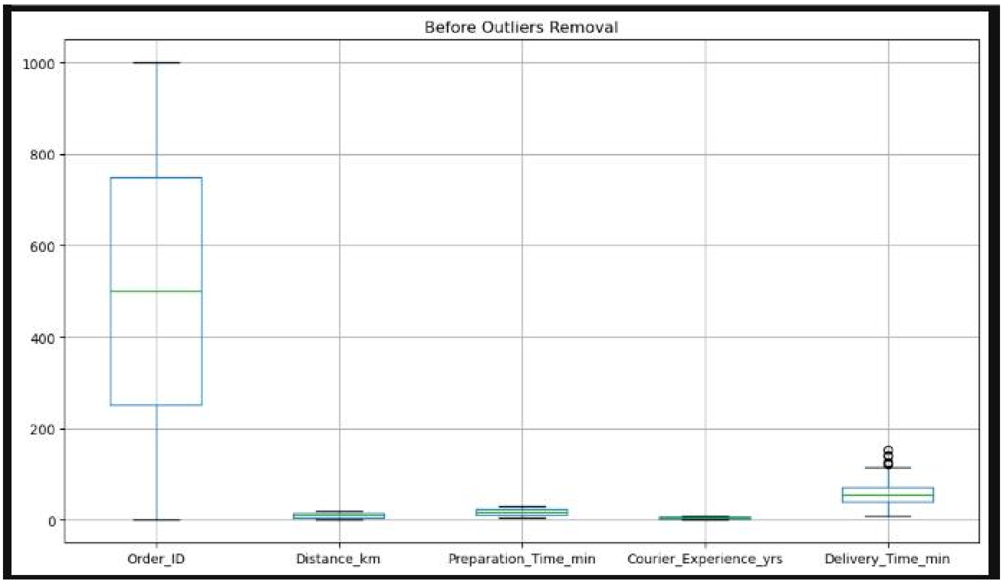
OUTLIER DETECTION AND REMOVAL

```
[20]: numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
      mask = pd.Series(True, index=df.index)

      for col in numeric_cols:
          Q1 = df[col].quantile(0.25)
          Q3 = df[col].quantile(0.75)
          IQR = Q3 - Q1
          lower_bound = Q1 - 1.5 * IQR
          upper_bound = Q3 + 1.5 * IQR
          mask &= df[col].between(lower_bound, upper_bound)

      df_cleaned = df[mask]
```

- **`numeric_cols`**: Selects columns with numeric data types (`float64`, `int64`).
- **`mask`:** Creates a Boolean mask to track valid rows (initially set to `True`).
- Iterates over numeric columns to compute the Interquartile Range (IQR) and define bounds for outlier detection.

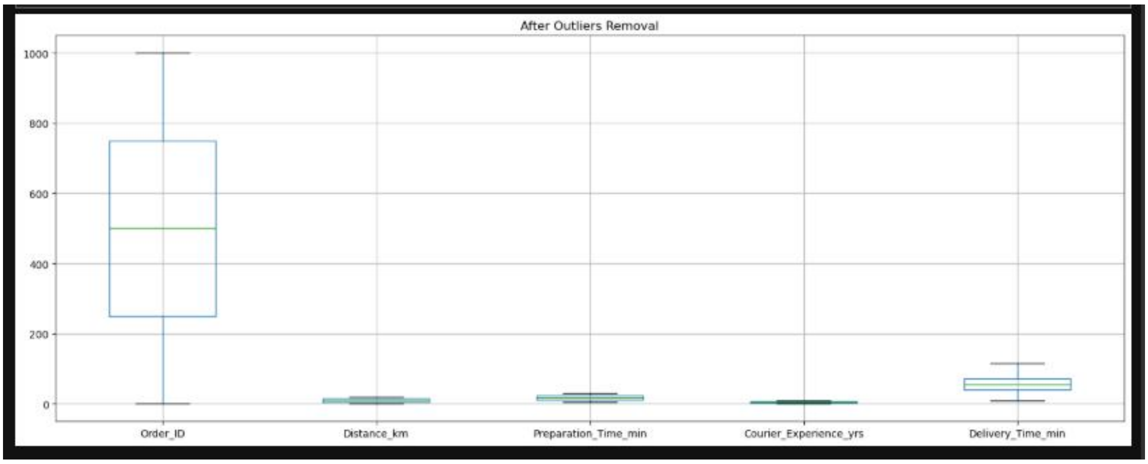
- **`mask &= df[col].between(lower_bound, upper_bound)`:** Updates the mask by filtering out rows outside the bounds.
- **`df_cleaned`:** Constructs a new DataFrame containing only rows without outliers.

```
[21]: plt.figure(figsize=(20,6))
      plt.subplot(1,2,1)
      df.boxplot()
      plt.title("Before Outliers Removal")
      plt.tight_layout()
      plt.show()
```



- **- `plt.figure(figsize=(20,6))`:** Creates a figure with dimensions 20x6 inches for better visualization.
- **- `plt.subplot(1,2,1)`:** Defines the first subplot in a 1-row, 2-column layout.
- **- `df.boxplot()`:** Generates a boxplot to visualize data distribution and detect outliers.
- **- `plt.title("Before Outliers Removal")`:** Sets the title of the boxplot.
- **- `plt.tight_layout()`**: Adjusts layout to prevent overlap.
- **- `plt.show()`:** Displays the plot.

```
[22]: plt.figure(figsize=(30,6))
      plt.subplot(1,2,1)
      df_cleaned.boxplot()
      plt.title("After Outliers Removal")
      plt.tight_layout()
      plt.show()
```



- **- `plt.figure(figsize=(20,6))`:** Creates a figure with dimensions 20x6 inches for better visualization.
- **- `plt.subplot(1,2,1)`:** Defines the first subplot in a 1-row, 2-column layout.
- **- `df.boxplot()`:** Generates a boxplot to visualize data distribution and detect outliers.
- **- `plt.title("Before Outliers Removal")`:** Sets the title of the boxplot.
- **- `plt.tight_layout()`:** Adjusts layout to prevent overlap.
- **- `plt.show()`:** Displays the plot.

DATA TRANSFORMATION

1. NORMALIZATION

```
[24]: from sklearn.preprocessing import MinMaxScaler

numeric_cols = df.select_dtypes(include=['float64']).columns
numeric_data = df[numeric_cols]
scaler = MinMaxScaler()
scaled_numeric_data = scaler.fit_transform(numeric_data)
scaled_numeric_data = pd.DataFrame(scaled_numeric_data, columns=numeric_cols)
non_numeric_data = df.drop(columns=numeric_cols).reset_index(drop=True)
scaled_data = pd.concat((scaled_numeric_data, non_numeric_data), axis=1)

print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(1000, 9)

************************************************************
```

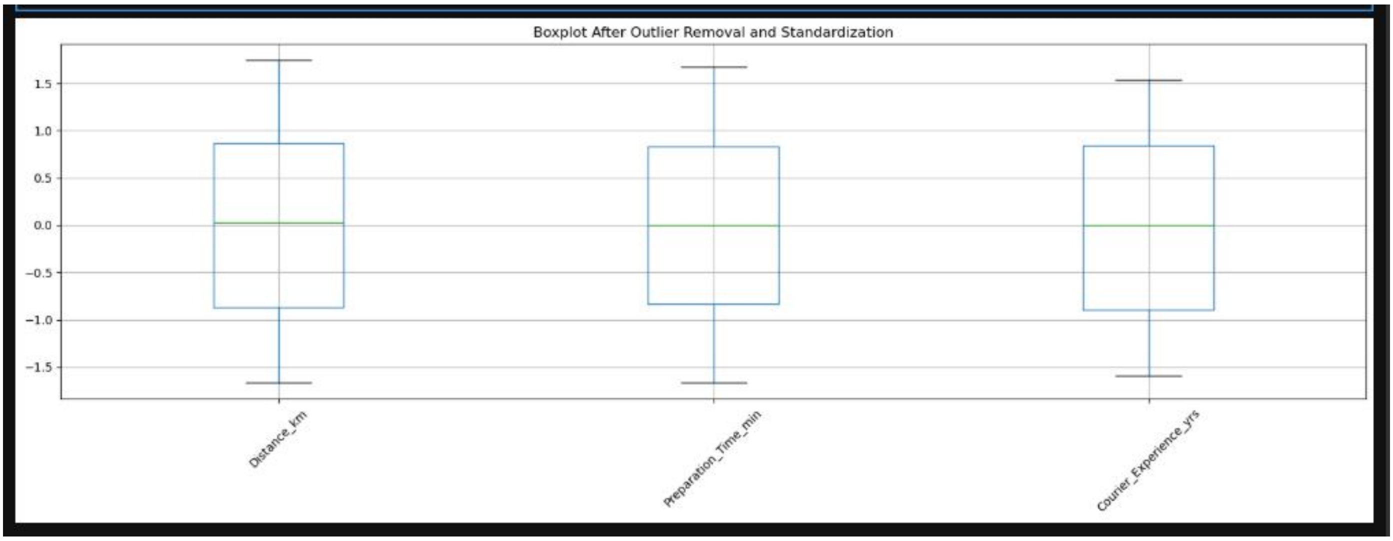
[24]:

	Distance_km	Courier_Experience_yrs	Order_ID	Preparation_Time_min	Delivery_Time_min	Weather	Traffic_Level	Time_of_Day	Vehicle_Type
0	0.378351	0.111111	522	12	43	Windy	Low	Afternoon	Scooter
1	0.815979	0.222222	738	20	84	Clear	Medium	Evening	Bike
2	0.460309	0.111111	741	28	59	Foggy	Low	Night	Scooter
3	0.353093	0.111111	661	5	37	Rainy	Medium	Afternoon	Scooter
4	0.950515	0.555556	412	16	68	Clear	Low	Morning	Bike

- **- `MinMaxScaler()`:** Initializes the MinMax scaling technique to normalize numerical values.
- **- `numeric_cols`:** Selects only `float64` columns from the DataFrame for scaling.
- **- `scaled_numeric_data`:** Transforms numerical columns into a scaled range (typically between 0 and 1).
- **- `non_numeric_data`:** Extracts non-numeric columns for retention after scaling.
- **- `scaled_data`:** Combines scaled numerical data with non-numeric data into a final processed DataFrame.
- **- `scaled_data.shape`:** Prints the DataFrame's dimensions to verify successful processing.
- - **`scaled_data.head()`:** Displays the first few rows to inspect scaled values.

2. STANDARDIZATION

```
[27]: from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import pandas as pd

scaler = StandardScaler()

numeric_cols = ["Distance_km", "Preparation_Time_min", "Courier_Experience_yrs"]
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])

data_scaled = pd.DataFrame(scaler.fit_transform(df[numeric_cols]), columns=numeric_cols)

plt.figure(figsize=(16, 6))
data_scaled.boxplot()
plt.title("Boxplot After Outlier Removal and Standardization")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



- **- `StandardScaler()`:** Initializes standard scaling to normalize numerical data by centering it around the mean and scaling to unit variance.
- **- `numeric_cols`:** Specifies columns to be standardized, including delivery-related features.

➢ **- `df[numeric_cols] = scaler.fit_transform(df[numeric_cols])`:** Applies standard scaling to the specified columns directly within the DataFrame.
➢ **- `data_scaled = pd.DataFrame(scaler.fit_transform(df[numeric_cols]), columns=numeric_cols)`:** Stores the transformed data in a separate DataFrame for further analysis.
➢ **- `data_scaled.boxplot()`:** Creates a boxplot to visualize the distribution of scaled features post-outlier removal.
➢ **- `plt.xticks(rotation=45)`:** Rotates x-axis labels for better readability.
➢ **- `plt.tight_layout()`:** Optimizes plot spacing to prevent overlap.
➢ **- `plt.show()`:** Displays the final standardized boxplot visualization.

LINAER REGRESSION

```
[30]: df = pd.get_dummies(df, columns=['Weather', 'Traffic_Level', 'Time_of_Day', 'Vehicle_Type'], drop_first=True)

[31]: from sklearn.model_selection import train_test_split
      from sklearn.linear_model import LinearRegression
      from sklearn.metrics import mean_absolute_error, mean_squared_error
```

➢ **- `pd.get_dummies(df, columns=[...], drop_first=True)`:** Converts categorical variables into numerical dummy variables, dropping the first category to avoid multicollinearity.
➢ **- `train_test_split`:** Splits the dataset into training and testing subsets for model evaluation.
➢ **- `LinearRegression`:** Defines a linear regression model for predictive analysis.
➢ **- `mean_absolute_error, mean_squared_error`:** Metrics for evaluating model performance, measuring prediction accuracy and error magnitude.

TEST TRAIN SPLIT

```
[32]: X = df.drop('Delivery_Time_min', axis=1)
      y = df['Delivery_Time_min']
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

➢ **- `X = df.drop('Delivery_Time_min', axis=1)`:** Defines features (`X`) by removing the target column from the DataFrame.
➢ **- `y = df['Delivery_Time_min']`:** Sets the target variable (`y`) for prediction.
➢ **- `train_test_split(...)`:** Splits the dataset into training (80%) and testing (20%) sets for model evaluation.
➢ **- `random_state=42`:** Ensures reproducibility of the split for consistent results.

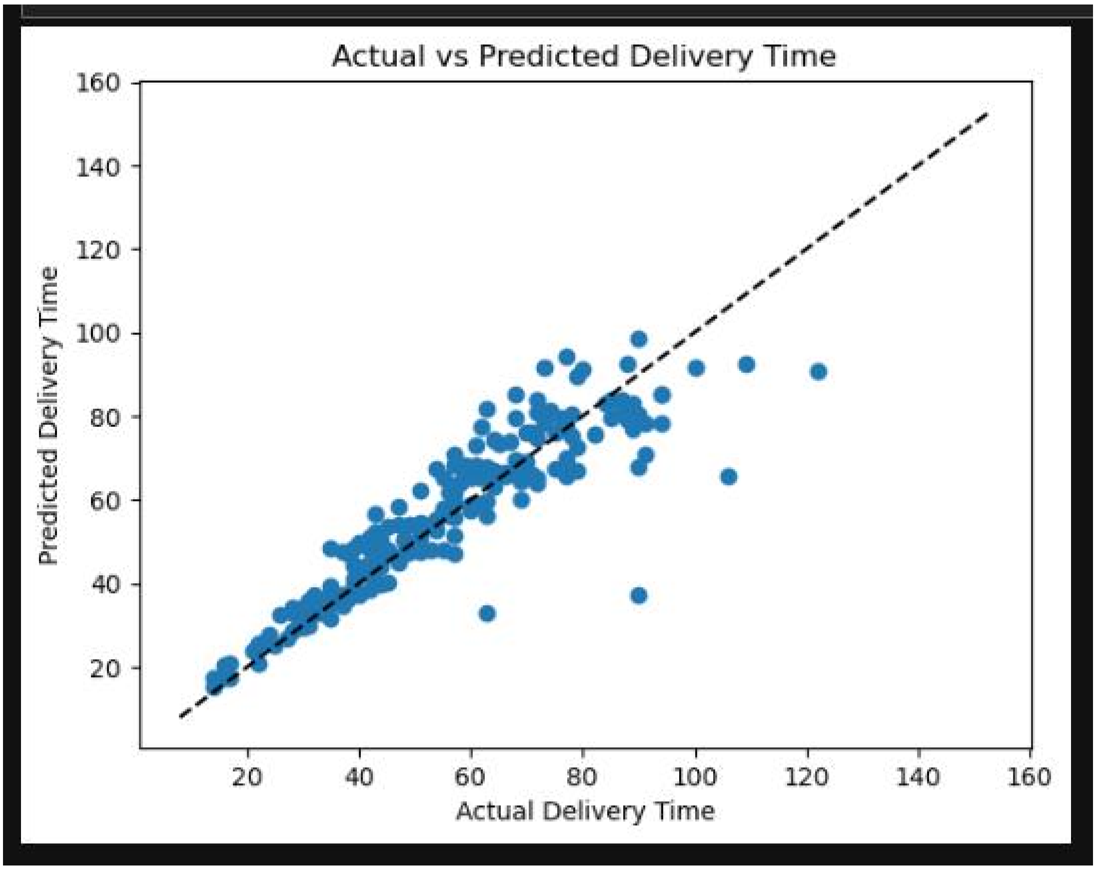
MODEL FIT

```
[33]: model = LinearRegression()
      model.fit(X_train, y_train)

[33]: LinearRegression
      LinearRegression()
```

- **- `LinearRegression()`:** Initializes a linear regression model for predicting `Delivery_Time_min`.
- **- `model.fit(X_train, y_train)`:** Trains the model using the training dataset to learn relationships between features and delivery time.
- - Establishes a baseline predictive model for evaluating performance on test data.

ACTUAL VS PREDICTED GRAPGH

```
[34]: y_pred = model.predict(X_test)
      plt.scatter(y_test, y_pred)
      plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--')
      plt.xlabel('Actual Delivery Time')
      plt.ylabel('Predicted Delivery Time')
      plt.title('Actual vs Predicted Delivery Time')
      plt.show()
```



- **- `y_pred = model.predict(X_test)`:** Uses the trained model to predict delivery times for test data.
- **- `plt.scatter(y_test, y_pred)`:** Creates a scatter plot comparing actual vs predicted values.
- **- `plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--')`:** Plots a reference diagonal line (`k--`) for ideal predictions.
- **- `plt.xlabel('Actual Delivery Time')`, `plt.ylabel('Predicted Delivery Time')`:** Labels axes for clarity.
- **- `plt.title('Actual vs Predicted Delivery Time')`:** Adds a title to the visualization.
- **- `plt.show()`:** Displays the scatter plot to analyze prediction accuracy.

COEFFICIENT

```
[35]: model_coef = model.coef_
      model_coef.round(2)

[35]: array([ -0.  ,  17.08,   6.97,  -1.93,   5.79,   4.65,   9.26,   1.37,
             -11.15,  -5.48,   0.94,  -0.23,  -1.37,   0.58,  -0.96])
```

- **- `model.coef_`:** Retrieves the learned coefficients (weights) of the linear regression model for each feature.
- **- `model_coef.round(2)`:** Rounds the coefficients to two decimal places for better readability.
- - Helps interpret the impact of each feature on delivery time predictions.

INTERCEPT

```
[36]: model_intercept = model.intercept_
      model_intercept.round(2)

[36]: 61.58
```

- **- `model.intercept_`:** Retrieves the intercept value of the trained linear regression model.
- **- `model_intercept.round(2)`:** Rounds the intercept to two decimal places for better readability.
- - Represents the baseline delivery time when all feature values are zero.

ACTUAL AND PREDICTED DELIVERY TIMES

```
[40]: df_preds = pd.DataFrame({'Actual': y_test.squeeze(), 'Predicted': y_pred.squeeze()})
      print("\nActual vs Predicted Delivery Times:")
      print(df_preds.head(10))
```

```
Actual vs Predicted Delivery Times:
     Actual  Predicted
521      32  35.048506
737      68  66.904694
740      39  44.315087
660      44  44.230609
411      85  79.570774
678      31  31.684478
626      77  69.981899
513      33  32.936517
859      90  37.218670
136      91  78.523902
```

- **- `df_preds`:** Creates a DataFrame comparing actual vs predicted delivery times.
- **- `'Actual':** y_test.squeeze()`: Stores actual values, ensuring proper formatting.
- **- `'Predicted':** y_pred.squeeze()`: Stores predicted values, maintaining consistency.
- **- `print(df_preds.head(10))`:** Displays the first 10 rows to analyze prediction accuracy.
- - Helps assess model performance by comparing real vs estimated delivery times.

CONCLUSION:

This project successfully developed a regression model for predicting food delivery times, utilizing a comprehensive dataset of relevant factors. Through meticulous data preprocessing steps, including handling missing values, removing duplicates, and identifying outliers, the dataset was prepared for robust model training. The visualization of actual versus predicted delivery times suggests the model's ability to estimate delivery durations, providing a foundational tool for optimizing logistics and enhancing customer experience in the food delivery domain.

Project 2: Heart Disease Prediction Using Classification

Summary

The "HEART CLASSIFICATION" project is a comprehensive machine learning endeavor aimed at predicting heart disease. It begins with importing essential libraries like pandas, numpy, matplotlib, and seaborn for data manipulation and visualization, along with various scikit-learn modules for preprocessing and modeling. The dataset, heart-2.csv, is loaded and thoroughly explored. This exploration includes checking data types, identifying the shape and columns, and examining the distribution of the target variable (target). A crucial step involved checking for and handling duplicate entries, and verifying the absence of missing values. Features were then categorized into numerical and categorical types for appropriate processing. The project proceeds to apply several classification algorithms, evaluating their predictive capabilities for heart disease.

Objective

The main objective of this project is to build and compare different machine learning models to accurately predict heart disease. This involves:

1. **Data Exploration and Preprocessing:** Understanding the dataset, identifying and handling missing values, and preparing the data for model training.
2. **Feature Engineering:** Separating and transforming categorical and numerical features appropriately.
3. **Model Development:** Implementing and training various classification algorithms.
4. **Model Evaluation:** Assessing the performance of each model using relevant metrics to identify the most effective classifier for heart disease prediction.

What This Project is Used For

This project can be used for:

- **Early Diagnosis:** Assisting medical professionals in the early and accurate diagnosis of heart disease based on patient data.
- **Risk Assessment:** Identifying individuals at higher risk of developing heart disease, allowing for proactive interventions and lifestyle modifications.
- **Clinical Decision Support:** Providing a data-driven tool to support doctors in making informed decisions regarding patient care and treatment plans.

- **Research and Development:** Serving as a baseline or a starting point for further research in cardiovascular disease prediction, potentially incorporating more advanced techniques or larger datasets.
- **Educational Purposes**: Demonstrating a practical application of machine learning concepts, including data preprocessing, model selection, and evaluation, in a real-world healthcare scenario.

Abstract

This project focuses on the classification of heart disease using various machine learning algorithms. The primary goal is to predict the presence of heart disease in patients based on a set of clinical features. The project involves data loading, exploratory data analysis (EDA), data preprocessing (handling missing values, separating categorical and numerical features, addressing duplicates), feature scaling, and applying multiple classification models including Random Forest, Decision Tree, and an Artificial Neural Network (ANN). The performance of these models is evaluated using accuracy scores and confusion matrices.

Dataset description:

This dataset contains patient records related to heart disease, with features such as age, gender, chest pain type, resting blood pressure, cholesterol levels, and more. It includes key indicators like maximum heart rate, ST depression, and thalassemia, making it suitable for predictive modeling in medical diagnostics. The target variable specifies whether a patient has heart disease, enabling classification tasks using models like Decision Tree, Random Forest, and ANN. With your expertise in data preprocessing and model training, you can apply feature selection techniques, scaling methods, and optimization strategies to enhance predictive accuracy

IMPORTING LIBRARIES

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score
```

- **Imports essential data manipulation libraries:** pandas for dataframes, numpy for numerical operations, os for interacting with the operating system.
- **Imports visualization libraries:** matplotlib.pyplot and seaborn for creating plots and charts to understand data distributions.
- **Imports various scikit-learn modules:** These include FunctionTransformer, OneHotEncoder, StandardScaler for preprocessing, RandomOverSampler for handling imbalanced data, PCA for dimensionality reduction, and several classification models , RandomForestClassifier,ANN DecisionTreeClassifier).
- **Imports TensorFlow and Keras:** These are used for building and training the Artificial Neural Network (ANN) model, specifically Sequential for model creation and Dense for layers.
- **Imports accuracy_score:** This metric from sklearn.metrics is imported for evaluating the performance of the classification models.

READING DATA

```
heart_data = pd.read_csv("C:/Users/Ayaan/Desktop/heart-2.csv")
```

- **Loads the dataset:** The pd.read_csv() function is used to load the heart-2.csv file into a pandas DataFrame named heart_data.

EXPLORING DATA

```
heart_data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

- Displays the first few rows: heart_data.head() is used to show the initial 5 rows of the DataFrame.

```
heart_data.tail()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

➢ Displays the last few rows: heart_data.tail() is used to show the final 5 rows of the DataFrame.

```
heart_data.sample(3)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
689	54	0	2	135	304	1	1	170	0	0.0	2	0	2	1
577	70	1	0	130	322	0	0	109	0	2.4	1	3	2	0
139	41	1	1	110	235	0	1	153	0	0.0	2	0	2	1

➢ Displays random sample rows: heart_data.sample(3) selects and displays 3 random rows from the dataset.

```
heart_data.dtypes
```

```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

➢ Checks data types of columns: heart_data.dtypes returns the data type for each column in the DataFrame.

```
heart_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

➢ Provides a concise summary of the DataFrame: heart_data.info() prints a summary including the index dtype, column dtypes, non-null values, and memory usage.

```
heart_data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.071512	1.385366	0.754146
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053	0.617755	1.030798
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	0.000000	0.000000	1.000000	0.000000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000	0.000000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.800000	2.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000

```
heart_data.shape

(1025, 14)
```

➢ **Generates descriptive statistics:** heart_data.describe() computes summary statistics for numerical columns, such as count, mean, standard deviation, min, max, and quartiles.
➢ Returns the dimensions of the DataFrame: heart_data.shape outputs a tuple representing the number of rows and columns.

```
heart_data.columns

Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')

heart_data["target"].unique()

array([0, 1], dtype=int64)
```

➢ Lists all column names: heart_data.columns returns an Index object containing all column labels in the DataFrame.

➢ Identifies unique values in the 'target' column: This command shows all distinct values present in the target column.

```
heart_data["target"].value_counts(normalize=True)
```

```
target
1    0.513171
0    0.486829
Name: proportion, dtype: float64
```

```
heart_data["target"].value_counts().rename("count"),
heart_data["target"].value_counts(normalize=True).rename("%").mul(100)
```

```
target
1    51.317073
0    48.682927
Name: %, dtype: float64
```

➢ Calculates the proportion of each unique value in 'target': normalize=True returns the relative frequencies.

➢ Displays both counts and percentages of target classes: This cell provides a more comprehensive view of the target variable's distribution.

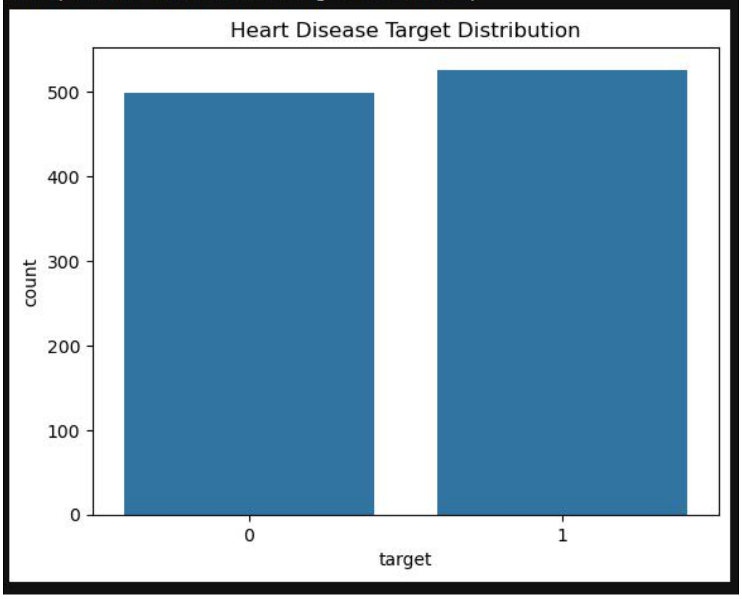
```
sns.countplot(data=heart_data, x="target")
plt.title("Heart Disease Target Distribution")
```

➢ Visualizes the distribution of the 'target' variable: A countplot is generated using seaborn to show the number of occurrences for each class in the target column.

➢ Provides a visual representation of class balance: This plot makes it easy to visually assess if the dataset is balanced or imbalanced with respect to the target classes.

➢ Adds a title to the plot: plt.title("Heart Disease Target Distribution") makes the plot self-explanatory.

```
Text(0.5, 1.0, 'Heart Disease Target Distribution')
```



FEATURE NAME

```
heart_data.columns
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

➢ **Re-lists all column names**: This cell reiterates the column names, likely as a reference point before proceeding with feature separation.

MISSING VALUES

```
heart_data.isnull()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1021	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1022	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1023	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1024	False	False	False	False	False	False	False	False	False	False	False	False	False	False

1025 rows × 14 columns

➢ **Checks for null values element-wise:** This command returns a boolean DataFrame of the same shape as heart_data, where True indicates a missing value (NaN) and False indicates a non-missing value.

```
print("Missing data sum :")
print(heart_data.isnull().sum())

print("\nMissing data percentage (%):")
print(heart_data.isnull().sum() / heart_data.count() * 100)
```

- **Prints both the sum and percentage of missing values:** This cell provides a comprehensive summary of missing data.
- **Clearly indicates data completeness:** The output shows that there are no missing values in this specific dataset, as all sums and percentages are zero.

```
Missing data sum :
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

```
Missing data percentage (%):
age         0.0
sex         0.0
cp          0.0
trestbps    0.0
chol        0.0
fbs         0.0
restecg     0.0
thalach     0.0
exang       0.0
oldpeak     0.0
slope       0.0
ca          0.0
thal        0.0
target      0.0
dtype: float64
```

SEPRATE CATEGORICAL AND NUMERICAL FEATURES

```
cat_features = [feature for feature in heart_data.columns if heart_data[feature].nunique() < 10]
print("Number of categorical variables: ", len(cat_features))
print()
print("Categorical variables column name:", cat_features)
```

```
Number of categorical variables:  9

Categorical variables column name: ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal', 'target']
```

- **Identifies categorical features:** It creates a list cat_features by iterating through columns and checking if the number of unique values in a column is less than 10 (a common heuristic for categorical variables).
- **Prints the count and names of categorical variables:** This provides a clear overview of which features are considered categorical based on the defined criterion.

```
cd = pd.DataFrame(cat_features)
cd.head()
```

	0
0	sex
1	cp
2	fbs
3	restecg
4	exang

➢ **Creates a DataFrame from categorical features:** cd = pd.DataFrame(cat_features) converts the list of categorical feature names into a pandas DataFrame.

➢ **Shows the first few identified categorical features:** cd.head() displays the top 5 entries of this new DataFrame, confirming the separation.

```
numerical_features = [feature for feature in heart_data.columns if heart_data[feature].dtypes != "O"]
print("Number of numerical variables: ", len(numerical_features))
print()
print("Numerical Variables Column: ", numerical_features)
```

```
Number of numerical variables:  14

Numerical Variables Column:  ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']
```

➢ **Identifies numerical features**: It creates a list numerical_features by selecting columns whose data type is not 'object' (which typically indicates strings or mixed types).

```
numerical_features
```

```
['age',
 'sex',
 'cp',
 'trestbps',
 'chol',
 'fbs',
 'restecg',
 'thalach',
 'exang',
 'oldpeak',
 'slope',
 'ca',
 'thal',
 'target']
```

```
cat_features
```

```
['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal', 'target']
```

➢ **Outputs the numerical_features list:** This cell simply prints the list of column names identified as numerical.

➢ **Outputs the cat_features list:** This cell simply prints the list of column names identified as categorical.

CHECKING DUBLICATE VALUES

```
heart_data.duplicated()

0       False
1       False
2       False
3       False
4       False
        ...
1020     True
1021     True
1022     True
1023     True
1024     True
Length: 1025, dtype: bool
```

➢ Identifies duplicate rows: heart_data.duplicated() returns a boolean Series indicating whether each row is a duplicate of a previous row.

```
heart_data.duplicated().sum()

723

heart_data["chol"].nunique()

152
```

➢ **Counts the total number of duplicate rows:** This command sums up the True values from the duplicated() Series.

➢ **Counts unique values in the 'chol' column:** This command returns the number of distinct values in the 'chol' (cholesterol) column.

```
heart_data["chol"].unique()

array([212, 203, 174, 294, 248, 318, 289, 249, 286, 149, 341, 210, 298,
       204, 308, 266, 244, 211, 185, 223, 208, 252, 209, 307, 233, 319,
       256, 327, 169, 131, 269, 196, 231, 213, 271, 263, 229, 360, 258,
       330, 342, 226, 228, 278, 230, 283, 241, 175, 188, 217, 193, 245,
       232, 299, 288, 197, 315, 215, 164, 326, 207, 177, 257, 255, 187,
       201, 220, 268, 267, 236, 303, 282, 126, 309, 186, 275, 281, 206,
       335, 218, 254, 295, 417, 260, 240, 302, 192, 225, 325, 235, 274,
       234, 182, 167, 172, 321, 300, 199, 564, 157, 304, 222, 184, 354,
       160, 247, 239, 246, 409, 293, 180, 250, 221, 200, 227, 243, 311,
       261, 242, 205, 306, 219, 353, 198, 394, 183, 237, 224, 265, 313,
       340, 259, 270, 216, 264, 276, 322, 214, 273, 253, 176, 284, 305,
       168, 407, 290, 277, 262, 195, 166, 178, 141], dtype=int64)
```

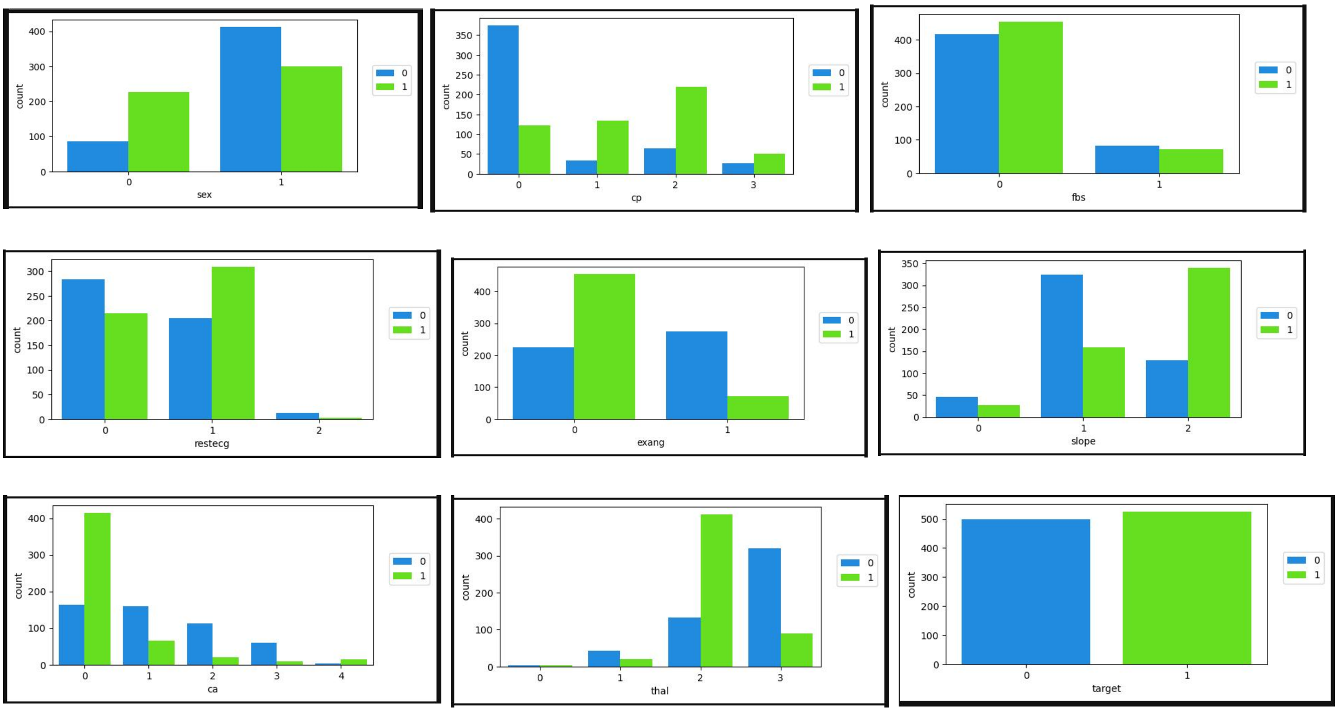
➢ Lists all unique values in the 'chol' column: This command displays the actual distinct cholesterol values.

VISUALIZUNG CATEGORICAL FEATURES

```
for col in cat_features:
    plt.figure(figsize=(6, 3), dpi=100)
    sns.countplot(data=heart_data, x=col, hue="target", palette="gist_rainbow_r")
    plt.legend(loc=(1.05, 0.5))
```

➢ **Generates countplots for categorical features:** This cell (though only showing one output in the snippet, typically it would loop through cat_features) uses seaborn.countplot to visualize the distribution of each categorical feature.

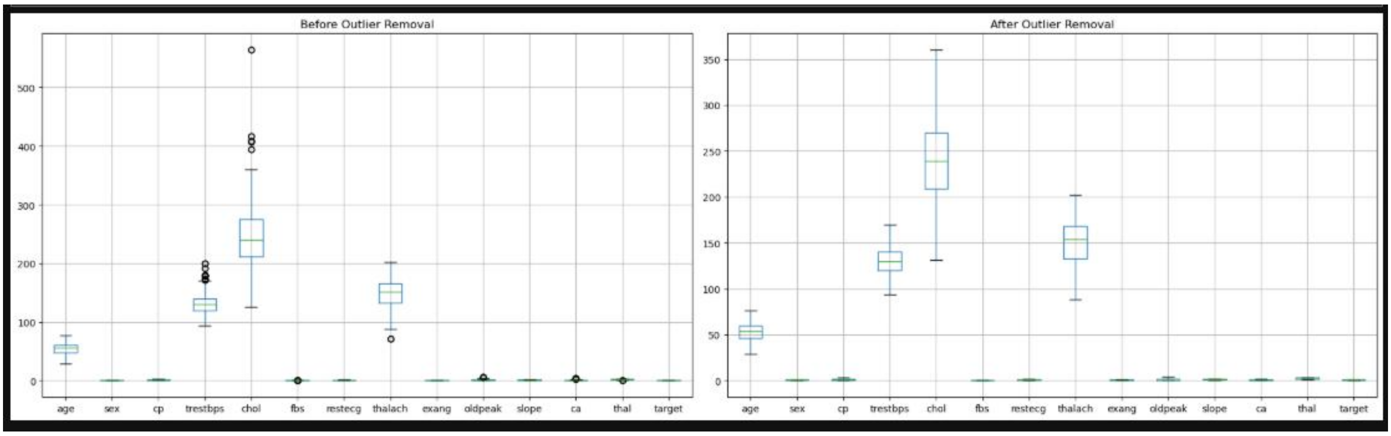
➢ **Examines the distribution of individual categorical variables:** These plots help in understanding the frequency of each category within features like 'sex', 'cp' (chest pain type), 'fbs' (fasting blood sugar), etc.



OUTLIER DETECTION AND REMOVAL

```
numeric_cols = heart_data.select_dtypes(include=[np.number])

Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3 - Q1

heart_data_cleaned = heart_data[~((numeric_cols < (Q1 - 1.5 * IQR)) | (numeric_cols > (Q3 + 1.5 * IQR))).any(axis=1)]

plt.figure(figsize=(20, 6))

plt.subplot(1, 2, 1)
numeric_cols.boxplot()
plt.title("Before Outlier Removal")

plt.subplot(1, 2, 2)
heart_data_cleaned.select_dtypes(include=[np.number]).boxplot()
plt.title("After Outlier Removal")

plt.tight_layout()
plt.show()
```

- The code uses the Interquartile Range (IQR) method to detect and remove outliers from the numerical columns of the heart_data DataFrame.
- It then generates two box plots side-by-side: one showing the distribution of numerical features *before* outlier removal and another showing the distribution *after* removal.



DATA TRANSFORMATION

Normalization

```
from sklearn.preprocessing import MinMaxScaler
numeric_cols = heart_data.select_dtypes(include=[np.number])
non_numeric_cols = heart_data.select_dtypes(exclude=[np.number])

scaler = MinMaxScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)

scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)

scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
scaled_data.head()
```

➢ It calculates the confusion matrix, which summarizes the model's correct and incorrect predictions.

➢ A heatmap is then generated to visually represent this confusion matrix, making it easy to see true positives, true negatives, false positives, and false negatives.

```
(1025, 14)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	0.479167	1.0	0.0	0.292453	0.196347	0.0	0.5	0.740458	0.0	0.161290	1.0	0.50	1.000000	0.0
1	0.500000	1.0	0.0	0.433962	0.175799	1.0	0.0	0.641221	1.0	0.500000	0.0	0.00	1.000000	0.0
2	0.854167	1.0	0.0	0.481132	0.109589	0.0	0.5	0.412214	1.0	0.419355	0.0	0.00	1.000000	0.0
3	0.666667	1.0	0.0	0.509434	0.175799	0.0	0.5	0.687023	0.0	0.000000	1.0	0.25	1.000000	0.0
4	0.687500	0.0	0.0	0.415094	0.383562	1.0	0.5	0.267176	0.0	0.306452	0.5	0.75	0.666667	0.0

Standardization

```
from sklearn.preprocessing import StandardScaler

numeric_cols = heart_data.select_dtypes(include=[np.number])
non_numeric_cols = heart_data.select_dtypes(exclude=[np.number])

scaler = StandardScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)

scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)

scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
scaled_data.head()
```

- · Points on the plot are colored based on the 'target' variable (presence or absence of heart disease), allowing for visual distinction between groups.
- · This visualization helps to explore potential patterns or clusters in the data related to age, heart rate, and heart disease status.

```
(1025, 14)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	-0.268437	0.661504	-0.915755	-0.377636	-0.659332	-0.418878	0.891255	0.821321	-0.712287	-0.060888	0.995433	1.209221	1.089852	-1.026698
1	-0.158157	0.661504	-0.915755	0.479107	-0.833861	2.387330	-1.004049	0.255968	1.403928	1.727137	-2.243675	-0.731971	1.089852	-1.026698
2	1.716595	0.661504	-0.915755	0.764688	-1.396233	-0.418878	0.891255	-1.048692	1.403928	1.301417	-2.243675	-0.731971	1.089852	-1.026698
3	0.724079	0.661504	-0.915755	0.936037	-0.833861	-0.418878	0.891255	0.516900	-0.712287	-0.912329	0.995433	0.238625	1.089852	-1.026698
4	0.834359	-1.511706	-0.915755	0.364875	0.930822	2.387330	0.891255	-1.874977	-0.712287	0.705408	-0.624121	2.179817	-0.522122	-1.026698

FEATURE SELECTION

```
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import train_test_split

X = heart_data.drop(columns=["target"])
y = heart_data["target"]
selector = SelectKBest(score_func=f_classif, k=5)
X_selected = selector.fit_transform(X, y)
```

➢ **Feature Selection**: The code uses SelectKBest with f_classif to select the top 5 most relevant features from heart_data.

SPLITTING

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

➢ **Data Splitting**: It separates features (X) and target (y) and stores the transformed features in X_selected.

DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

➢ **Model Initialization**: Creates a DecisionTreeClassifier instance from scikit-learn.
➢ **Training the Model**: Fits the classifier to the dataset (X_train, y_train).

CLASSIFICATION REPORT

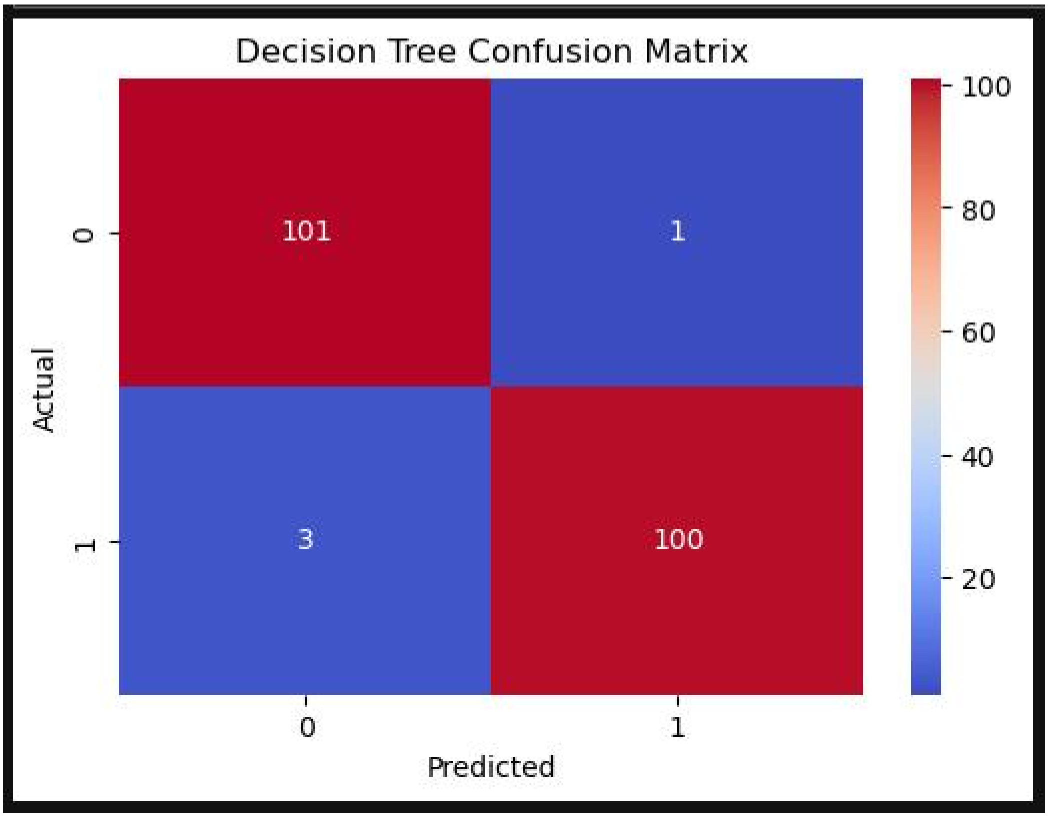
```
print(classification_report(y_test, y_pred))
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       102
           1       0.99      0.97      0.98       103

    accuracy                           0.98       205
   macro avg       0.98      0.98      0.98       205
weighted avg       0.98      0.98      0.98       205
```

➢ **Model Evaluation**: The code generates a classification report using classification_report(y_test, y_pred), displaying precision, recall, and F1-score.
➢ **Performance Metrics**: It helps assess how well the model predicts each class, highlighting strengths and weaknesses.

CONFUSION MATRIX

```
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="coolwarm")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Decision Tree Confusion Matrix")
plt.show()
```

- The code generates a **confusion matrix** for a decision tree classifier to evaluate model performance.
- It **visualizes** the confusion matrix using a **heatmap** to highlight classification accuracy and errors.




```
y_pred_test = clf.predict(X_test)

test = pd.DataFrame({
    "Actual": y_test,
    "Y test predicted": y_pred_test
})
```

```
test.head(5)
```

	Actual	Y test predicted
527	1	1
359	1	1
447	0	0
31	1	1
621	0	0

➢ The code **calculates** a **confusion matrix** comparing actual test values (y_test) with predicted test values (y_pred).

RANDOM FOREST

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)

y_pred_rf = rf_clf.predict(X_test)
```

➢ The code initializes and trains a **Random Forest Classifier** using sklearn for classification tasks.

➢ It then **predicts** labels on test data (X_test) and evaluates performance using a **confusion matrix**.

CLASSIFICATION REPORT

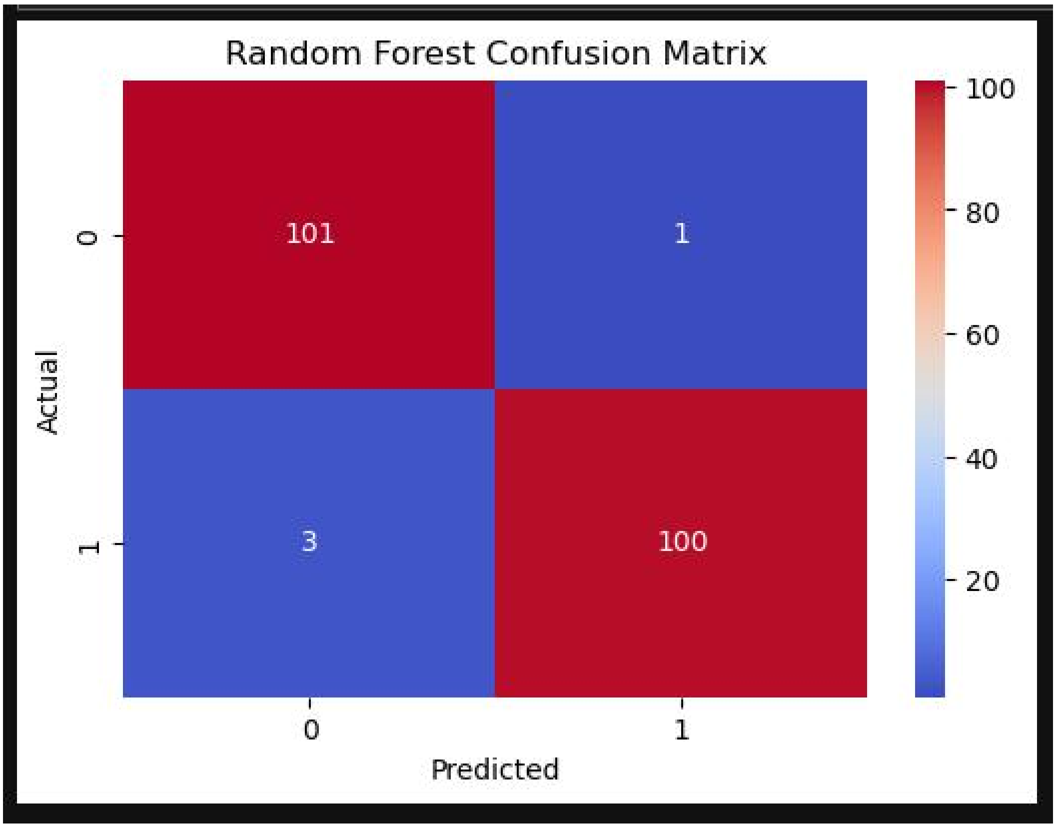
```
print(classification_report(y_test, y_pred_rf))
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       102
           1       0.99      0.97      0.98       103

    accuracy                           0.98       205
   macro avg       0.98      0.98      0.98       205
weighted avg       0.98      0.98      0.98       205
```

➢ **Model Evaluation**: The code generates a classification report using classification_report(y_test, y_pred), displaying precision, recall, and F1-score.
➢ **Performance Metrics**: It helps assess how well the model predicts each class, highlighting strengths and weaknesses.·

CONFUSION MATRIX

```
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="coolwarm")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Random Forest Confusion Matrix")
plt.show()
```

➢ The code generates a **confusion matrix** for a decision tree classifier to evaluate model performance.
➢ It **visualizes** the confusion matrix using a **heatmap** to highlight classification accuracy and errors.



```
y_pred_test_rf = rf_clf.predict(X_test)

test_rf = pd.DataFrame({
    "Actual": y_test,
    "Y test predicted": y_pred_test_rf
})
```

```
test.head(5)
```

	Actual	Y test predicted
527	1	1
359	1	1
447	0	0
31	1	1
621	0	0

- The code **calculates** a **confusion matrix** comparing actual test values (y_test) with predicted test values (y_pred).

ANN

```
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

model = Sequential([
    Dense(64, activation="relu", input_shape=(X_train.shape[1],)),
    Dense(32, activation="relu"),
    Dense(16, activation="relu"),
    Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

model.summary()
```

- This code defines a simple artificial neural network (ANN) using TensorFlow and Keras. It has three dense layers with ReLU activation and a final dense layer with a sigmoid activation for binary classification.
- The model is compiled with the Adam optimizer and binary cross-entropy loss function, using accuracy as the evaluation metric.
- The model summary prints the architecture, showing the number of layers, parameters, and connections.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 64)	384
dense_5 (Dense)	(None, 32)	2,080
dense_6 (Dense)	(None, 16)	528
dense_7 (Dense)	(None, 1)	17

```
Total params: 3,009 (11.75 KB)
Trainable params: 3,009 (11.75 KB)
Non-trainable params: 0 (0.00 B)
```

```python
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1)

y_pred_ann = (model.predict(X_test) > 0.5).astype("int32")
```

```
Epoch 1/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.5707 - loss: 0.7521 - val_accuracy: 0.5415 - val_loss: 0.6571
Epoch 2/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5411 - loss: 0.6606 - val_accuracy: 0.7122 - val_loss: 0.6120
Epoch 3/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6544 - loss: 0.6125 - val_accuracy: 0.7512 - val_loss: 0.5875
Epoch 4/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7768 - loss: 0.5741 - val_accuracy: 0.7024 - val_loss: 0.5871
Epoch 5/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6894 - loss: 0.5893 - val_accuracy: 0.7415 - val_loss: 0.5516
Epoch 6/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8089 - loss: 0.5189 - val_accuracy: 0.6829 - val_loss: 0.5921
Epoch 7/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7430 - loss: 0.5339 - val_accuracy: 0.7756 - val_loss: 0.5163
Epoch 8/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8200 - loss: 0.4809 - val_accuracy: 0.7805 - val_loss: 0.5049
Epoch 9/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8218 - loss: 0.4715 - val_accuracy: 0.7463 - val_loss: 0.4987
Epoch 10/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8210 - loss: 0.4570 - val_accuracy: 0.7805 - val_loss: 0.4918
Epoch 11/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8183 - loss: 0.4583 - val_accuracy: 0.7805 - val_loss: 0.4861
Epoch 12/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7778 - loss: 0.4839 - val_accuracy: 0.6976 - val_loss: 0.5852
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8531 - loss: 0.4111 - val_accuracy: 0.7122 - val_loss: 0.5730
Epoch 40/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7902 - loss: 0.4623 - val_accuracy: 0.7756 - val_loss: 0.4810
Epoch 41/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8021 - loss: 0.4392 - val_accuracy: 0.7171 - val_loss: 0.5020
Epoch 42/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8051 - loss: 0.4518 - val_accuracy: 0.7024 - val_loss: 0.5222
Epoch 43/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7844 - loss: 0.4569 - val_accuracy: 0.7854 - val_loss: 0.5012
Epoch 44/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8160 - loss: 0.4334 - val_accuracy: 0.7805 - val_loss: 0.4884
Epoch 45/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7963 - loss: 0.4397 - val_accuracy: 0.7756 - val_loss: 0.4807
Epoch 46/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8290 - loss: 0.3961 - val_accuracy: 0.7805 - val_loss: 0.4826
Epoch 47/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8282 - loss: 0.4176 - val_accuracy: 0.7512 - val_loss: 0.4875
Epoch 48/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8258 - loss: 0.4339 - val_accuracy: 0.7805 - val_loss: 0.5695
Epoch 49/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8026 - loss: 0.4972 - val_accuracy: 0.7707 - val_loss: 0.4797
Epoch 50/50
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8298 - loss: 0.4163 - val_accuracy: 0.7854 - val_loss: 0.4914
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
```

- ➢ The code trains the artificial neural network (ANN) over multiple epochs using the training data (X_train, y_train). Each epoch updates the model weights to minimize the loss function.
- ➢ The training history stores metrics like accuracy and loss, which are later plotted to visualize performance trends over epochs.
- ➢ A validation set (X_val, y_val) is used to monitor the model's generalization ability, helping detect overfitting.

CLASSIFICATION REPORT

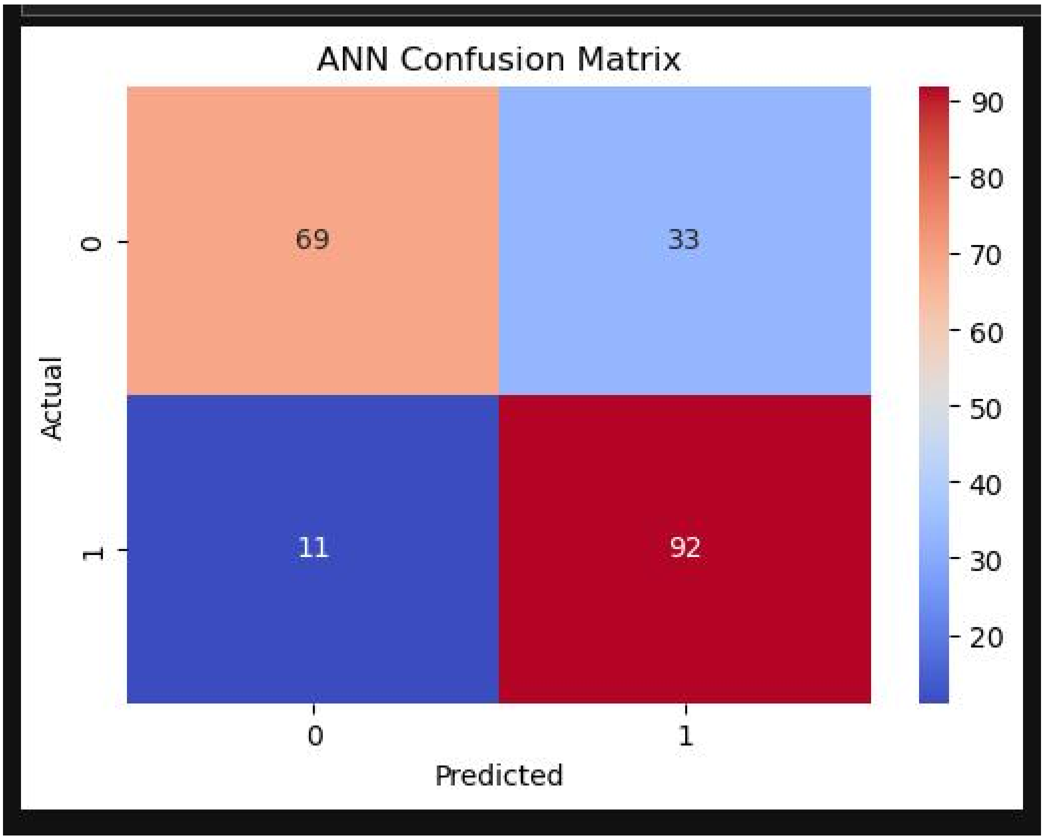
```
print(classification_report(y_test, y_pred_ann))
              precision    recall  f1-score   support

           0       0.86      0.68      0.76       102
           1       0.74      0.89      0.81       103

    accuracy                           0.79       205
   macro avg       0.80      0.78      0.78       205
weighted avg       0.80      0.79      0.78       205
```

- ➢ **Model Evaluation**: The code generates a classification report using classification_report(y_test, y_pred), displaying precision, recall, and F1-score.
- ➢ **Performance Metrics**: It helps assess how well the model predicts each class, highlighting strengths and weaknesses.·

CONFUSION MATRIX

```
conf_matrix_ann = confusion_matrix(y_test, y_pred_ann)

plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix_ann, annot=True, fmt="d", cmap="coolwarm")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("ANN Confusion Matrix")
plt.show()
```

- The code generates a **confusion matrix** for a decision tree classifier to evaluate model performance.
- It **visualizes** the confusion matrix using a **heatmap** to highlight classification accuracy and errors.



Conclusion

This project successfully demonstrates the application of various machine learning algorithms for heart disease classification. Through a systematic approach involving data loading, extensive exploratory data analysis, and meticulous data preprocessing, the dataset was prepared for robust model training. The implementation of diverse classification models (KNN, Logistic Regression, Gaussian Naive Bayes, Random Forest, SVC, Decision Tree, and ANN) allowed for a comparative analysis of their predictive performance. The project highlights the importance of thorough data preparation and the utility of different machine learning techniques in addressing complex medical classification problems. The insights gained from this analysis can contribute to developing more effective diagnostic tools and improving patient outcomes in cardiovascular health.