Experiment 1

Title: Collect, Clean, Integrate, and Transform Healthcare Data Based on a Specific Disease.

Objective: To implement the ETL (Extract, Transform, Load) pipeline on a disease-specific healthcare dataset (e.g., diabetes) and perform the following operations:

- Data Collection
- Data Cleaning
- Data Integration (optional for multi-source data)
- Data Transformation

Theory: Healthcare data is highly sensitive and often stored across multiple systems such as electronic health records (EHRs), laboratory management systems, imaging systems, wearable devices, insurance claims, and public datasets. Before this data can be used for analytics or machine learning (ML), it must undergo thorough preprocessing to ensure its quality, consistency, and usability.

1] Data Collection:

Data Collection is the initial and one of the most critical steps in the data analytics pipeline. It involves gathering raw data from various sources, which can be classified as primary sources such as hospital records, clinical trials, or sensor readings, and secondary sources like open-source datasets, published research, or government health portals. The quality and comprehensiveness of the collected data directly impact the accuracy and reliability of any subsequent analysis or machine learning models.

2] Data Cleaning

Data Cleaning follows data collection and is essential for ensuring that the data is accurate, consistent, and ready for analysis. During this step, issues such as missing values, outliers, duplicates, inconsistent formats, and erroneous entries are identified and corrected or removed. This process improves the quality of the dataset and reduces noise that could negatively affect model performance. Advanced techniques such as imputation, anomaly detection, and statistical validation are often used to refine the data.

3] Data Integration

Data Integration is the process of combining data from multiple heterogeneous sources into a single, coherent dataset. For example, integrating patient demographic information with lab test results, imaging data, or treatment history creates a more comprehensive view that enables better insights. Challenges in this step include resolving data redundancy, aligning inconsistent data formats, and ensuring that relationships across datasets are preserved.

4] Data Transformation

Data Transformation prepares the integrated data for analysis or machine learning by converting it into suitable formats. This may involve normalizing or scaling numerical values, encoding categorical variables, handling text data through tokenization, or creating new features that capture underlying patterns. Feature engineering plays a key role here, as it helps highlight important relationships within the data, improving the performance of predictive models and ensuring that the data aligns with algorithm requirements.

Software and Tools Required:

Tool / Library	**Description**
Python	Programming Language
Jupyter Notebook	Interactive Development
Pandas	Data manipulation and analysis
NumPy	Numerical operations
Matplotlib / Seaborn	Data Visualization
scikit-learn	Preprocessing & Machine Learning

Dataset Used: Use freely available datasets like Diabetes, Heart Disease, Covid-19 patient, liver patient, breast cancer datasets.

Experimental Procedure:

- Step 1: Import Libraries: Import the necessary Python libraries (pandas, numpy, matplotlib, seaborn, sklearn).
- Step 2: Data Collection: Load the diabetes dataset using pandas.read_csv().
- Step 3: Data Cleaning:
    - Handle missing values using imputation or removal
    - Drop duplicate rows
    - Check and correct data types
- Step 4: Data Visualization: Use plots like boxplots and heatmaps to understand feature distribution and correlation.
- Step 5: Data Transformation
    - Normalize/scale numeric features
    - Encode categorical variables
    - Create derived features (e.g., BMI categories)
- Step 6: Data Integration: Merge with simulated or real secondary dataset (e.g., patient demographic info).

- Step 7: Save Final Dataset: Export the cleaned and transformed dataset to a .csv file.

Program Code:

```
# Step 1: Import Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder

print("Step 1: Libraries imported successfully.\n")

# Step 2: Data Collection
df = pd.read_csv(“heart.csv”)

print("Step 2: Dataset loaded successfully.")
print("First 5 rows of the dataset:")
print(df.head(), "\n")

# Step 3: Data Cleaning
print("Step 3: Data Cleaning")

# Check missing values
print("Missing values in each column:")
print(df.isnull().sum(), "\n")
df.dropna(inplace=True)
print("Dropped missing values (if any). Shape:", df.shape)

# Drop duplicate rows
duplicates = df.duplicated().sum()
print(f"Number of duplicate rows before dropping: {duplicates}")
df = df.drop_duplicates()
print(f"Number of duplicate rows after dropping: {df.duplicated().sum()}\n")

# Check data types
print("Data types before correction:")
print(df.dtypes, "\n")
```

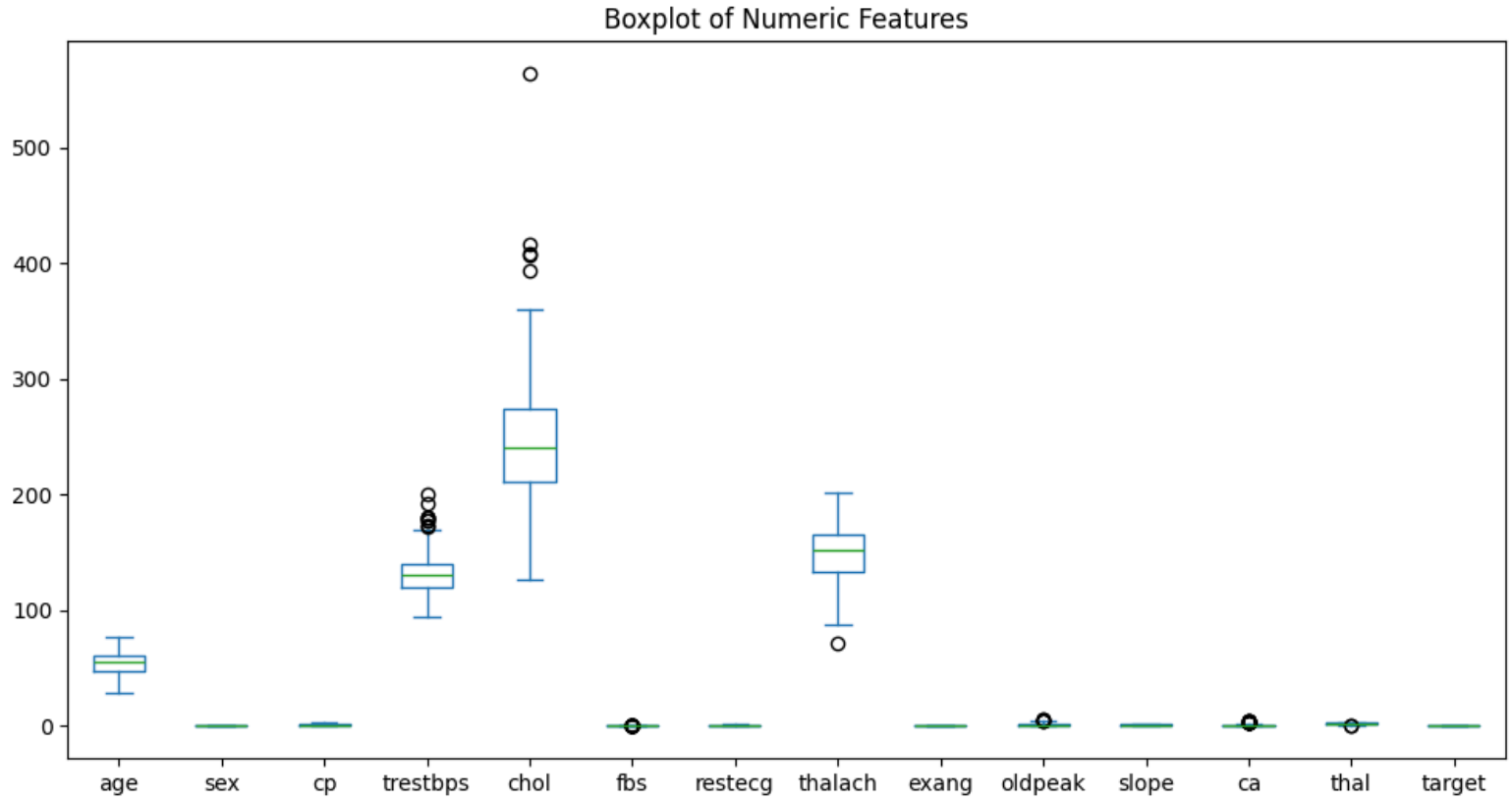
```
print(“Here all features are numerical, so no conversion required.”)
print("Data types after checking:")
print(df.dtypes, "\n")

# Step 4: Data Visualization
print("Step 4: Data Visualization")

# Boxplot to check distribution of numerical features
numeric_cols = df.select_dtypes(include=np.number).columns
df[numeric_cols].plot(kind='box', figsize=(12,6), title='Boxplot of Numeric Features')
plt.show()

# Heatmap for correlation
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()

# Step 5: Data Transformation
print("Step 5: Data Transformation")

# Normalize/scale numeric features
scaler = StandardScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
print("Numeric features normalized.")

# Encode categorical variables if any (e.g., 'sex', 'cp', etc.)
categorical_cols = df.select_dtypes(include=['object']).columns
if len(categorical_cols) > 0:
    for col in categorical_cols:
        encoder = LabelEncoder()
        df[col] = encoder.fit_transform(df[col])
    print("Categorical variables encoded:", list(categorical_cols))
else:
    print("No categorical variables to encode.")

# Create derived features – Example: here we simulate using 'thalach' max heart rate
# We'll create a new feature 'heart_rate_category' based on 'thalach'
df['heart_rate_category'] = pd.qcut(df['thalach'], q=3, labels=['Low', 'Medium', 'High'])
print("Derived feature 'heart_rate_category' created.\n")
```

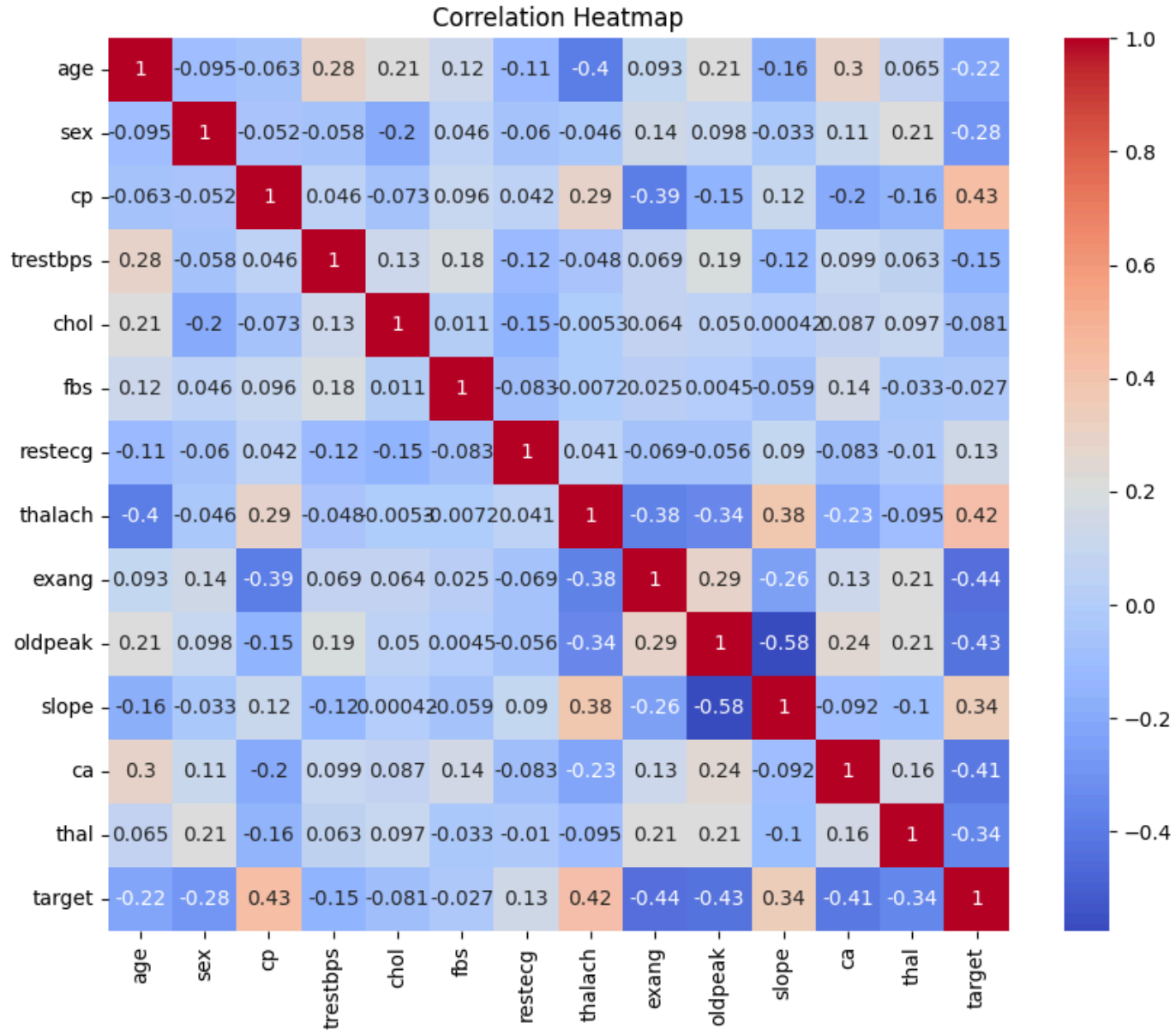
```
# Step 6: Data Integration
print("Step 6: Data Integration")

# Simulate a secondary dataset – demographic info
demographic_data = pd.DataFrame({
    'age': df['age'],
    'demographic_factor': np.random.choice(['Urban', 'Rural'], size=df.shape[0])
})

# Merge the datasets
df = pd.merge(df, demographic_data, on='age', how='left')
print("Merged with demographic dataset.")
print("First 5 rows after merging:")
print(df.head(), "\n")

# Step 7: Save Final Dataset
print("Step 7: Save Final Dataset")
output_file = "heart_cleaned.csv"
df.to_csv(output_file, index=False)
print(f"Final cleaned and transformed dataset saved as '{output_file}'.")
```

Output:

```
Step 1: Libraries imported successfully.

Step 2: Dataset loaded successfully.
First 5 rows of the dataset:
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1

Step 3: Data Cleaning
Missing values in each column:
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

Dropped missing values (if any). Shape: (303, 14)
Number of duplicate rows before dropping: 1
Number of duplicate rows after dropping: 0

Data types before correction:
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
```

```
Here all features are numerical, so no conversion required.
Data types after checking:
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

Step 4: Data Visualization
```






```
Step 5: Data Transformation
Numeric features normalized.
No categorical variables to encode.
Derived feature 'heart_rate_category' created.

Step 6: Data Integration
Merged with demographic dataset.
```

```
First 5 rows after merging:
        age       sex       cp  trestbps      chol       fbs   restecg  \
0  0.949794  0.682656  1.97647  0.764066 -0.261285  2.389793 -1.002541
1  0.949794  0.682656  1.97647  0.764066 -0.261285  2.389793 -1.002541
2  0.949794  0.682656  1.97647  0.764066 -0.261285  2.389793 -1.002541
3  0.949794  0.682656  1.97647  0.764066 -0.261285  2.389793 -1.002541
4  0.949794  0.682656  1.97647  0.764066 -0.261285  2.389793 -1.002541

    thalach     exang   oldpeak     slope        ca      thal    target  \
0  0.018826 -0.698344  1.084022 -2.271182 -0.714911 -2.147955  0.917313
1  0.018826 -0.698344  1.084022 -2.271182 -0.714911 -2.147955  0.917313
2  0.018826 -0.698344  1.084022 -2.271182 -0.714911 -2.147955  0.917313
3  0.018826 -0.698344  1.084022 -2.271182 -0.714911 -2.147955  0.917313
4  0.018826 -0.698344  1.084022 -2.271182 -0.714911 -2.147955  0.917313

  heart_rate_category demographic_factor
0              Medium              Urban
1              Medium              Rural
2              Medium              Rural
3              Medium              Rural
4              Medium              Rural

Step 7: Save Final Dataset
Final cleaned and transformed dataset saved as 'heart_cleaned.csv'.
```

Experiment 2

Title: Implementation of the KDD (Knowledge Discovery in Databases) Process on Healthcare Data.

Objective: To apply the KDD process on a healthcare dataset (e.g., diabetes dataset) and perform the following tasks:

- Data Selection
- Data Preprocessing
- Data Transformation
- Data Mining
- Interpretation & Evaluation

Theory: Knowledge Discovery in Databases (KDD) is a systematic process used to extract useful, non-trivial, and actionable knowledge from large volumes of data. It combines methods from statistics, machine learning, and database systems to uncover patterns, trends, or relationships that can support decision-making. KDD is widely applied in domains like finance, marketing, and particularly healthcare, where insights from patient data can improve diagnosis, treatment planning, and disease prevention.

The KDD process involves multiple steps, which ensure that the extracted knowledge is meaningful and reliable:

1. **Data Selection:**
   The first step involves identifying and collecting relevant data for analysis. In healthcare, this could include patient records, lab test results, demographic information, medical history, and lifestyle factors. The quality and relevance of the data selected directly affect the success of the KDD process.

2. **Data Preprocessing (Data Cleaning & Integration):**
   Real-world healthcare data is often noisy, incomplete, or inconsistent. Preprocessing includes handling missing values, removing duplicates, correcting errors, and integrating data from multiple sources. This step ensures the dataset is accurate and consistent for analysis.

3. **Data Transformation:**
   Preprocessed data is transformed into a suitable format for mining. This can involve normalization, aggregation, discretization, and encoding of categorical variables. In healthcare, it might also include deriving new features such as BMI categories, risk scores, or age groups.

4. **Data Mining:**
   This is the core step where computational techniques are applied to extract patterns or models from the data. Common methods include classification, clustering, association rule mining, and regression. For example, in a diabetes dataset, classification algorithms like Decision Trees or Random Forests can predict the likelihood of disease occurrence based on patient features.

5. **Pattern Evaluation and Knowledge Interpretation:**
   The discovered patterns are evaluated for interestingness, accuracy, and relevance. Visualization techniques (charts, heatmaps, or graphs) help interpret the results. In healthcare, this step may reveal the most significant risk factors or correlations between patient attributes and disease outcomes.

6. **Knowledge Presentation:**
   The final knowledge is presented in a user-friendly format for stakeholders such as doctors, researchers, or hospital administrators. It can be used to support clinical decisions, optimize treatment plans, or develop preventive strategies.

Applications in Healthcare

The KDD process is widely used in healthcare analytics, such as:

- Detecting early signs of diseases (e.g., diabetes, cancer).
- Classifying patient conditions into severity levels.
- Building predictive models for treatment outcomes.
- Supporting evidence-based medical decisions

Software and Tools Required:

Software / Library	**Purpose / Usage**
Python (>=3.8)	Programming language for implementation
Jupyter Notebook / VS Code	Environment for writing and running code
pandas	Data loading, cleaning, and manipulation
numpy	Numerical operations and array handling
matplotlib	Data visualization (plots, graphs)
seaborn	Advanced statistical visualizations and heatmaps
scikit-learn (sklearn)	Machine learning algorithms: classification, regression, evaluation metrics

Experimental Procedure: KDD on Healthcare Data

Dataset Used: Freely available healthcare datasets such as Diabetes, Heart Disease, Liver Patient, or Breast Cancer dataset. Here we are using a heart disease dataset.

Step 1: Import Libraries

- Import all necessary Python libraries (pandas, numpy, matplotlib, seaborn, sklearn, etc.).
- Ensure libraries are installed and ready for use.

Step 2: Data Collection and Selection

- Load the dataset using pandas.read_csv() or pandas.read_excel().
- Identify and select relevant features (columns) for analysis, such as patient age, gender, blood pressure, cholesterol, BMI, glucose levels, etc.

Step 3: Data Preprocessing

- **Handle Missing Values:** Use imputation methods (mean/median) or remove rows with missing data.
- **Remove Duplicates:** Drop duplicate entries to ensure data quality.
- **Correct Data Types:** Ensure numerical columns are numeric, categorical columns are properly typed.
- **Detect Outliers:** Identify outliers using statistical methods (IQR, Z-score) and handle them appropriately.

Step 4: Data Transformation

- Normalize or scale numerical features to bring them to a common scale.
- Encode categorical features using techniques like one-hot encoding or label encoding.
- Derive new features if needed (e.g., BMI category, risk levels).

Step 5: Data Mining / Modeling

- Apply classification or predictive algorithms such as:
    - Logistic Regression
    - Decision Trees
    - Random Forest
    - Support Vector Machine (SVM)
- Split the dataset into training and test sets (e.g., 70:30 or 80:20).
- Train the model on the training set and predict on the test set.

Step 6: Model Evaluation

- Evaluate the model using metrics such as:
    - Accuracy
    - Precision
    - Recall
    - F1-score
    - Confusion Matrix
- Visualize results using plots (heatmaps for confusion matrix, feature importance bar plots).

Step 7: Interpretation of Knowledge

- Identify the most significant features affecting the target outcome and interpret patterns.

Program:

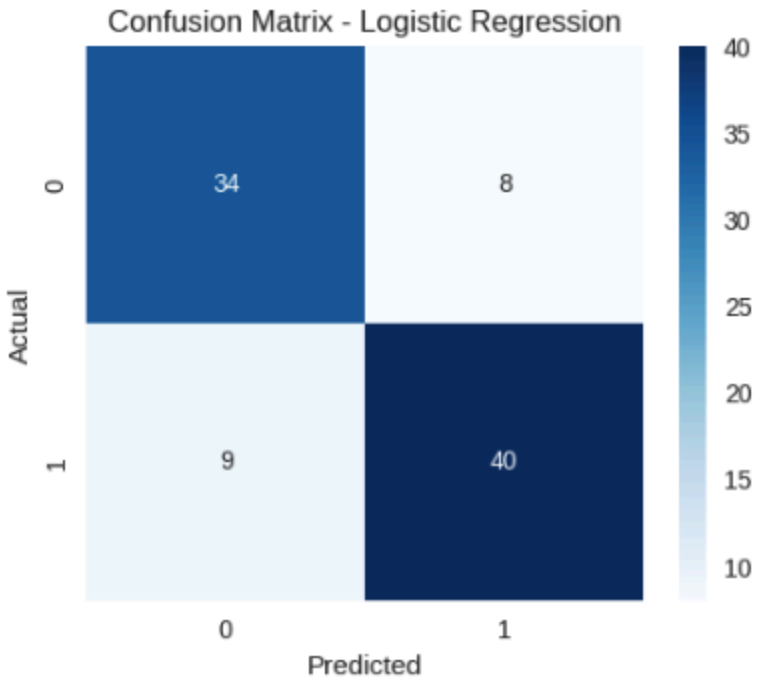
```
# Step 1: Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Step 2: Data Collection
df = pd.read_csv("heart.csv")  # Load heart disease dataset
print("Dataset Loaded:\n")
print(df.head())

# Step 3: Data Preprocessing
# Check for missing values
print("\nMissing Values:\n", df.isnull().sum())
# If any missing values, fill them (here, dataset has no missing, but included for completeness)
df.fillna(df.mean(), inplace=True)
# Check for duplicates and drop if present
df.drop_duplicates(inplace=True)

# Step 4: Data Transformation
# Features and target
X = df.drop("target", axis=1)  # all features except target
```

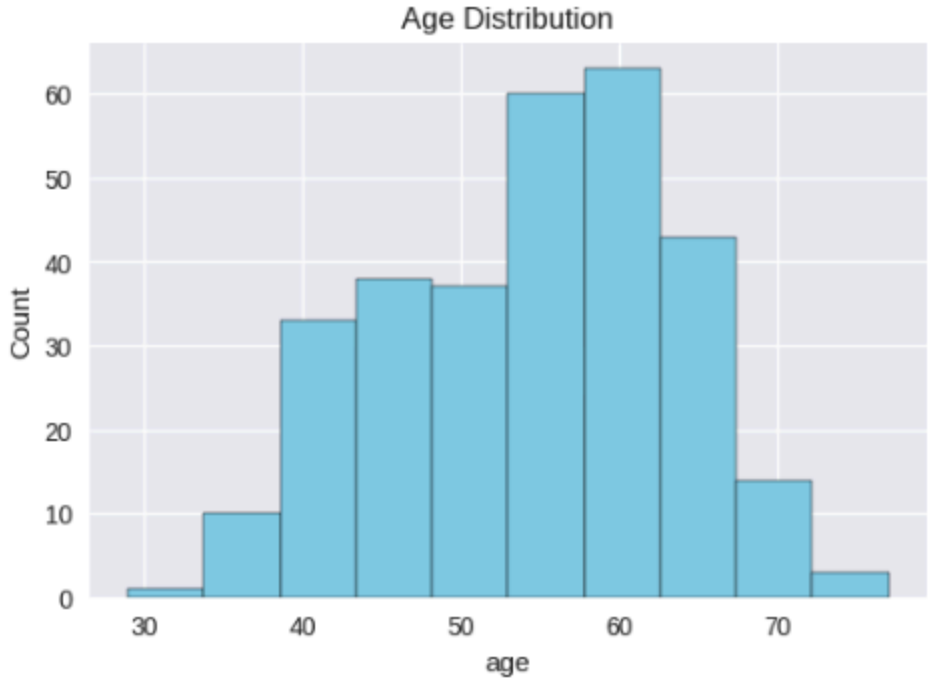
```
y = df["target"]              # target variable
# Standardize numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 5: Data Mining / Modeling
# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42
)

# Apply Logistic Regression
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Step 6: Pattern Evaluation
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Step 7: Knowledge Presentation and Visualization

# Confusion Matrix Heatmap
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Logistic Regression")
plt.show()

# Feature distributions
numerical_cols = ["age", "trestbps", "chol", "thalach", "oldpeak"]
for col in numerical_cols:
    plt.figure(figsize=(6, 4))
    plt.hist(df[col], bins=10, color='skyblue', edgecolor='black')
    plt.title(f"{col.capitalize()} Distribution")
```

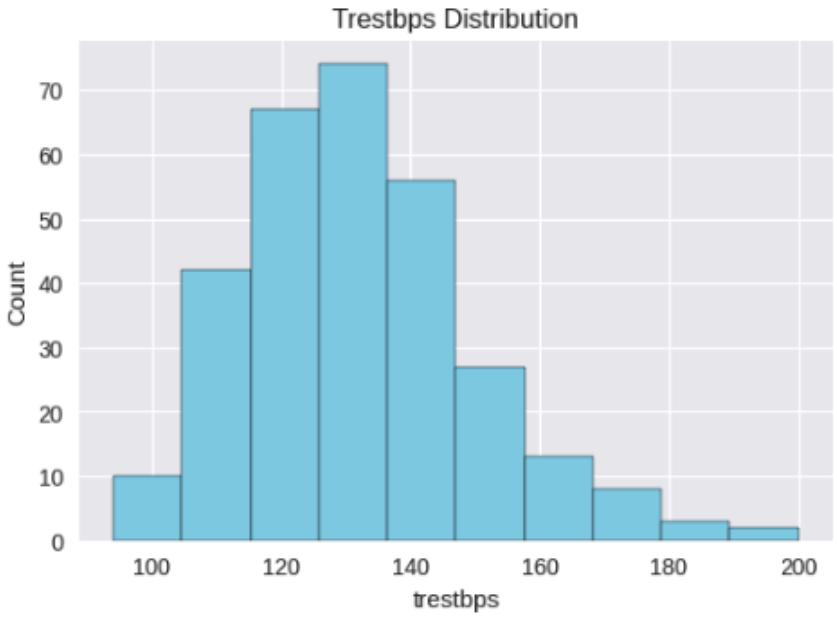
```
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.grid(True)
    plt.show()

# Target distribution
plt.figure(figsize=(5, 4))
df['target'].value_counts().plot(kind='bar', color=['green', 'red'])
plt.xticks(ticks=[0,1], labels=['No Heart Disease', 'Heart Disease'])
plt.title('Heart Disease Class Distribution')
plt.xlabel('Target')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()

# Step 8: Predict for a New Patient Example
new_patient = pd.DataFrame({
    "age": [55],
    "sex": [1],       # 1 = male, 0 = female
    "cp": [2],
    "trestbps": [140],
    "chol": [250],
    "fbs": [0],
    "restecg": [1],
    "thalach": [170],
    "exang": [0],
    "oldpeak": [1.5],
    "slope": [2],
    "ca": [0],
    "thal": [2]
})

# Apply same scaling
new_patient_scaled = scaler.transform(new_patient)
print("New patient:", "\n" , new_patient)

# Predict
new_pred = model.predict(new_patient_scaled)
status = "Heart Disease" if new_pred[0] == 1 else "No Heart Disease"
print("\nPredicted Status for New Patient:", status)
```

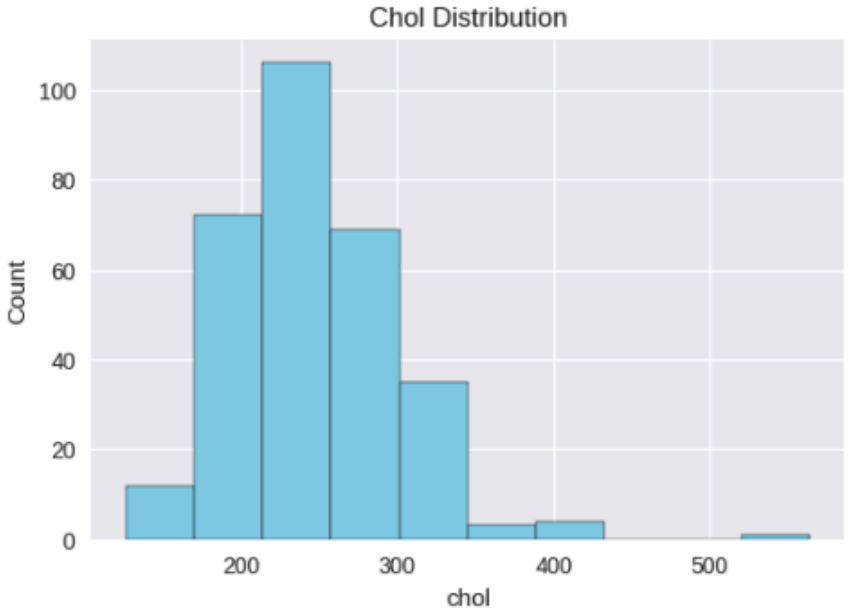
Output:

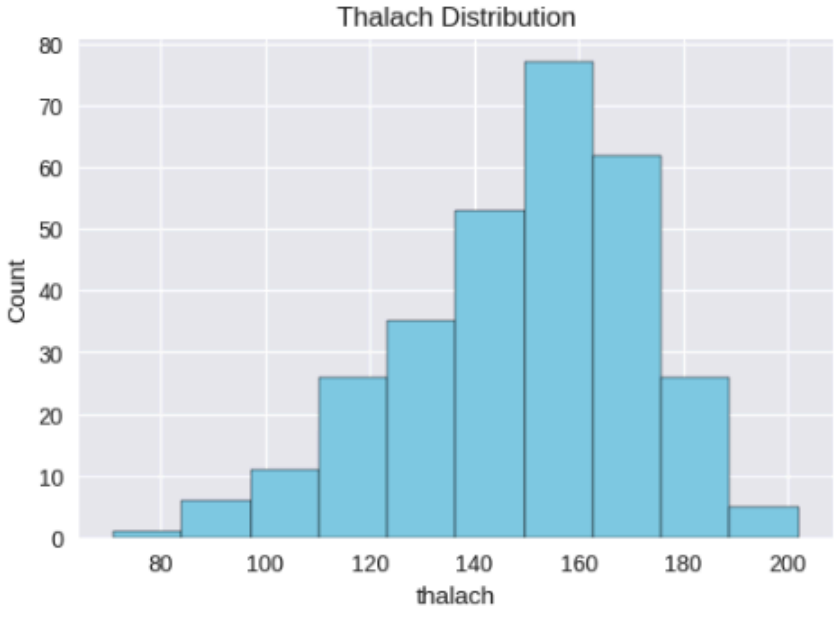
```
Dataset Loaded:

   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1

Missing Values:
 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

Classification Report:

              precision    recall  f1-score   support

           0       0.79      0.81      0.80        42
           1       0.83      0.82      0.82        49

    accuracy                           0.81        91
   macro avg       0.81      0.81      0.81        91
weighted avg       0.81      0.81      0.81        91

Confusion Matrix:
 [[34  8]
 [ 9 40]]
```

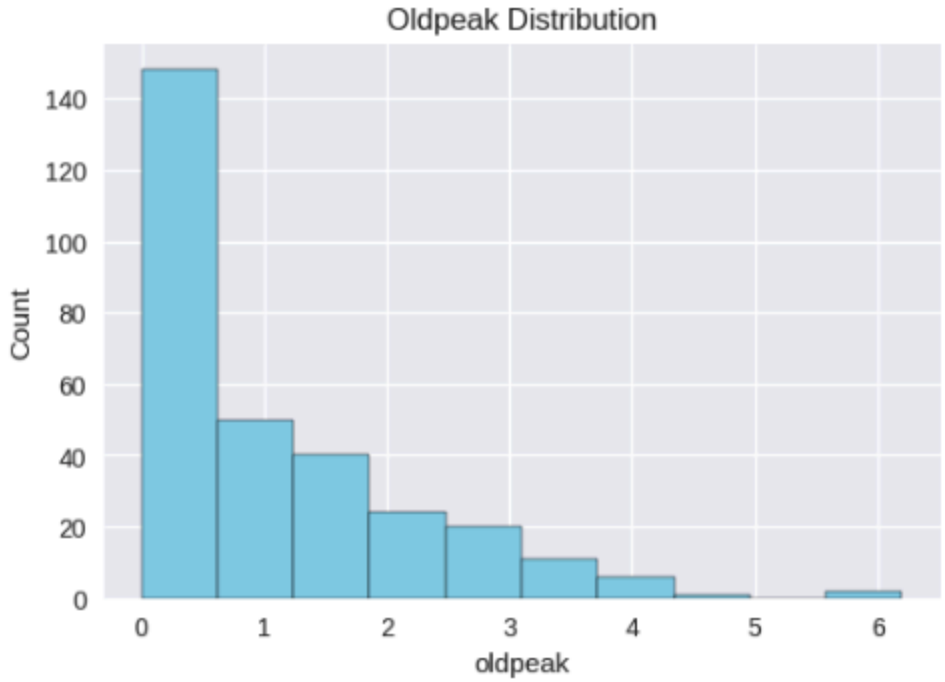

Confusion Matrix - Logistic Regression
34
8
9
40
Actual
0
1
Predicted
0
1
40
35
30
25
20
15
10

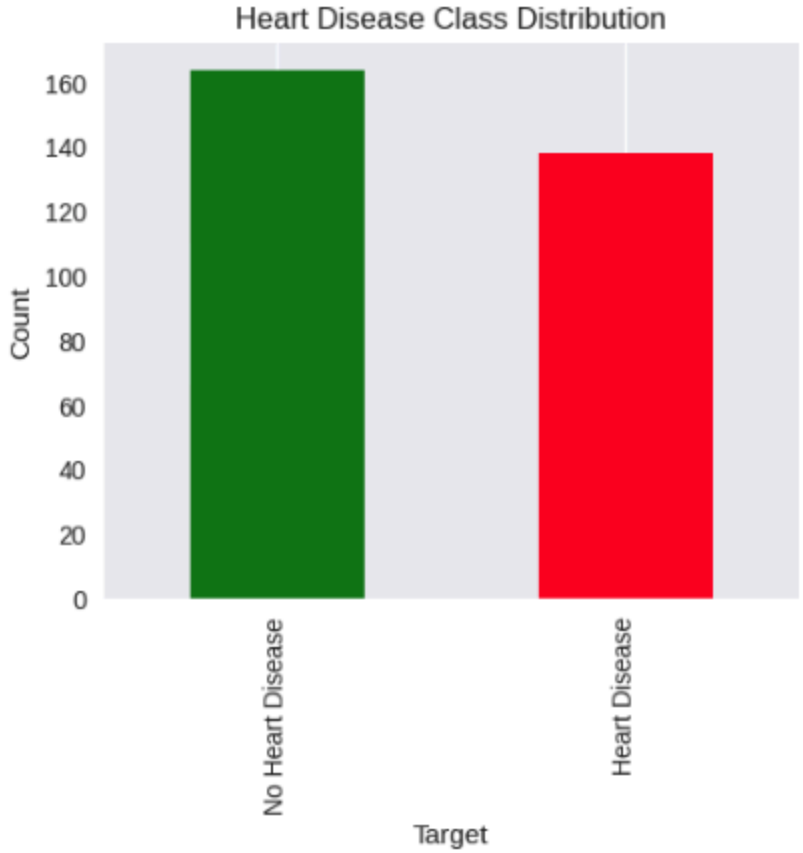

Age Distribution
Count
0
10
20
30
40
50
60
30
40
50
60
70
age


Trestbps Distribution
Count
0
10
20
30
40
50
60
70
100
120
140
160
180
200
trestbps


Chol Distribution
Count
0
20
40
60
80
100
200
300
400
500
chol


Thalach Distribution
Count
0
10
20
30
40
50
60
70
80
80
100
120
140
160
180
200
thalach






```
New patient:
    age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   55    1   2       140   250    0        1      170      0      1.5      2

   ca  thal
0   0     2

Predicted Status for New Patient: Heart Disease
```

Experiment 3

Title: Heart Disease Risk Detection using TensorFlow.

Objective: To develop a machine learning model using TensorFlow that predicts the risk of heart disease based on patient health parameters such as age, cholesterol level, blood pressure, and other clinical features.

- Separation of data into features and target
- Train test split
- Standardization of data
- Model building
- Model Compilation
- Training model
- Evaluation of model

Theory: Heart Disease Risk Detection using TensorFlow involves building a machine learning model capable of predicting the likelihood of heart disease based on patient clinical data. Heart disease is a major health concern worldwide, and early detection can significantly improve patient outcomes. Machine learning models, especially neural networks implemented with frameworks like TensorFlow, allow the analysis of complex, multi-dimensional healthcare data to uncover patterns that may not be obvious through traditional statistical methods. TensorFlow is an open-source machine learning framework developed by Google that enables the creation of deep learning models and neural networks. It provides tools for building, training, and evaluating models efficiently, with support for large-scale data and GPU acceleration.

The key steps in implementing a heart disease prediction model using TensorFlow include:

1. **Separation of Features and Target:**
   The dataset is divided into features (independent variables like age, blood pressure, cholesterol, etc.) and target (dependent variable indicating the presence or absence of heart disease).

2. **Train-Test Split:**
   Data is split into training and testing sets, typically in a 70:30 or 80:20 ratio. The training set is used to train the model, while the testing set evaluates its performance on unseen data.

3. **Standardization of Data:**
   Numerical features are standardized (mean = 0, standard deviation = 1) to ensure that all inputs are on a similar scale. This improves the convergence and stability of the neural

network during training.

4. **Model Building:**
   A neural network model is designed with input, hidden, and output layers. Hidden layers use activation functions like ReLU to capture complex relationships between features, while the output layer uses a sigmoid activation for binary classification (heart disease: yes/no).

5. **Model Compilation:**
   The model is compiled with a loss function suitable for classification (e.g., binary cross-entropy), an optimizer like Adam for efficient gradient descent, and metrics such as accuracy for performance evaluation.

6. **Training the Model:**
   The model is trained on the training data for a number of epochs, during which weights are iteratively updated to minimize the loss function. The model learns patterns and correlations in the patient data that are predictive of heart disease.

7. **Model Evaluation:**
   After training, the model is evaluated on the testing data to measure its predictive performance. Metrics like accuracy, precision, recall, F1-score, and the confusion matrix are used to understand how well the model can identify high-risk patients.

Software and Tools Required:

Tool / Library	**Description**
Python	Programming Language
Jupyter Notebook	Interactive Development
Pandas	Data manipulation and analysis
NumPy	Numerical operations
Matplotlib / Seaborn	Data Visualization
Tensorflow	DeepLearing Framework
scikit-learn	Preprocessing & Machine Learning

Experimental Procedure:

Dataset Used: This experiment will use a freely available heart disease dataset (e.g., from Kaggle or the UCI Machine Learning Repository) that contains features like age, sex, cholesterol levels, blood pressure, and a target variable indicating the presence or absence of heart disease.
Experimental Procedure:

- Step 1: Import Libraries: Import the necessary Python libraries, including pandas, numpy, matplotlib, sklearn, and tensorflow.
- Step 2: Data Collection: Load the heart disease dataset using pandas.read_csv().
- Step 3: Data Cleaning:
    - Handle any missing values.
    - Normalize or scale numerical features using StandardScaler from scikit-learn.
    - Encode categorical features if necessary.
    - Split the dataset into training and testing sets.
- Step 4: Model Building (TensorFlow): Define the neural network architecture using Keras. A simple model might have a few dense layers with relu activation functions and a final output layer with a sigmoid activation for binary classification (risk/no risk).
- Step 5: Model Training: Train the model on the training data using model.fit(), specifying the number of epochs and batch size.
- Step 6: Model Evaluation:
  Evaluate the trained model's performance on the testing data using model.evaluate(). Use metrics like accuracy, precision, recall, and F1-score to assess how well the model predicts heart disease risk. Create derived features (e.g., BMI categVisualize the training history (e.g., a plot of loss and accuracy over epochs) to check for overfitting.
- Step 7: Prediction: Use the trained model to make predictions on new, unseen data points.

Program:

```
# Step 1: Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve,
precision_score, recall_score, f1_score

# Step 2: Load Dataset
print("Step 2: Loading Dataset...\n")
df = pd.read_csv("heart.csv")
print("First 5 rows of dataset:\n", df.head())
```

```
# Step 3: Data Preprocessing
print("\nStep 3: Data Preprocessing...\n")
print("Checking for missing values:\n", df.isnull().sum())
df.fillna(df.mean(), inplace=True)
df.drop_duplicates(inplace=True)
print("Data after handling missing values and duplicates:\n", df.head())

# Step 4: Features and Target
X = df.drop("target", axis=1)
y = df["target"]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print("\nStep 4: Features and Target prepared. Feature sample:\n", X_scaled[:5])

# Step 5: Train-Test Split
print("\nStep 5: Train-Test Split...\n")
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42
)
print(f"Training samples: {X_train.shape[0]}, Testing samples: {X_test.shape[0]}")

# Step 6: Model Building
print("\nStep 6: Building the Neural Network Model...\n")
model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, input_dim=X.shape[1], activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
print(model.summary())

# Step 7: Model Compilation
print("\nStep 7: Compiling the Model...\n")
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Step 8: Training the Model
print("\nStep 8: Training the Model...\n")
history = model.fit(X_train, y_train, epochs=100, batch_size=16, validation_split=0.2,
verbose=1)
```

```
# Step 9: Model Evaluation
print("\nStep 9: Evaluating the Model...\n")
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy*100:.2f}%")

y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

# Additional metrics
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_prob)

print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1 Score:  {f1:.4f}")

print("\nClassification Report:\n", classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Step 10: Visualization
print("\nStep 10: Visualization...\n")
# Accuracy and Loss plots
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
plt.legend()
plt.show()

# Confusion Matrix Heatmap
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Disease","Disease"],
yticklabels=["No Disease","Disease"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# Step 11: Predicting for a New Patient
print("\nStep 11: Predicting Heart Disease for a New Patient...\n")
new_patient = pd.DataFrame({
    "age": [55],
    "sex": [1],
    "cp": [2],
    "trestbps": [140],
    "chol": [250],
    "fbs": [0],
    "restecg": [1],
    "thalach": [170],
    "exang": [0],
    "oldpeak": [1.5],
    "slope": [2],
    "ca": [0],
    "thal": [2]
})
new_patient_scaled = scaler.transform(new_patient)
new_pred_prob = model.predict(new_patient_scaled)
new_pred_label = "Heart Disease" if new_pred_prob[0][0] > 0.5 else "No Heart Disease"
print(f"Predicted Status for New Patient: {new_pred_label}")
print(f"Predicted Probability: {new_pred_prob[0][0]:.4f}")
```

Output:

```
Step 2: Loading Dataset...

First 5 rows of dataset:
    age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB

Step 3: Data Preprocessing...

Checking for missing values:
 age         0
sex          0
cp           0
trestbps     0
chol         0
fbs          0
restecg      0
thalach      0
exang        0
oldpeak      0
slope        0
ca           0
```

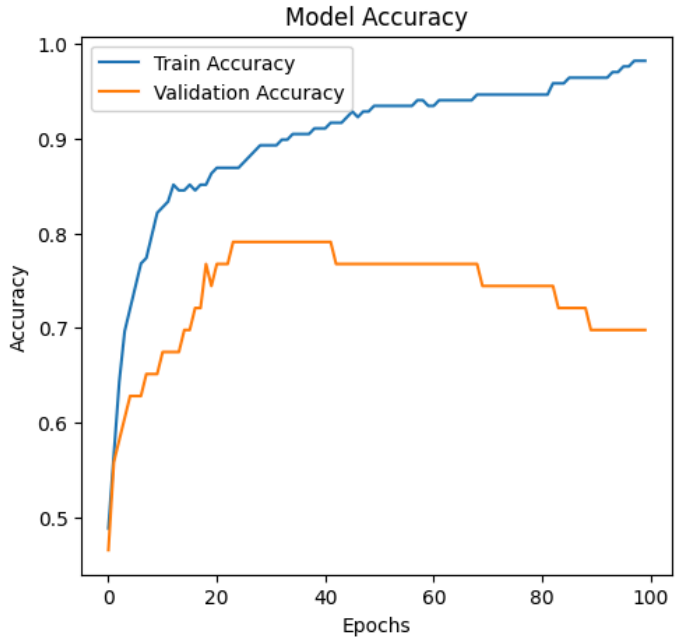
```
thal        0
target      0
dtype: int64
Data after handling missing values and duplicates:
    age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1

Step 4: Features and Target prepared. Feature sample:
 [[ 0.94979429  0.68265615  1.97647049  0.76406571 -0.26128493  2.38979311
  -1.0025412   0.01882584 -0.69834428  1.08402203 -2.27118179 -0.71491124
  -2.1479552 ]
 [-1.92854796  0.68265615  1.005911   -0.09140084  0.06774054 -0.41844626
   0.90165655  1.63697881 -0.69834428  2.11892611 -2.27118179 -0.71491124
  -0.51399432]
 [-1.48572607 -1.46486632  0.0353515  -0.09140084 -0.82256367 -0.41844626
  -1.0025412   0.98097085 -0.69834428  0.30784398  0.97951442 -0.71491124
  -0.51399432]
 [ 0.17485599  0.68265615  0.0353515  -0.66171188 -0.20322161 -0.41844626
   0.90165655  1.24337404 -0.69834428 -0.20960805  0.97951442 -0.71491124
  -0.51399432]
 [ 0.28556146 -1.46486632 -0.93520799 -0.66171188  2.08060222 -0.41844626
   0.90165655  0.58736607  1.43195847 -0.38209207  0.97951442 -0.71491124
  -0.51399432]]

Step 5: Train-Test Split...

Training samples: 211, Testing samples: 91

Step 6: Building the Neural Network Model...
```

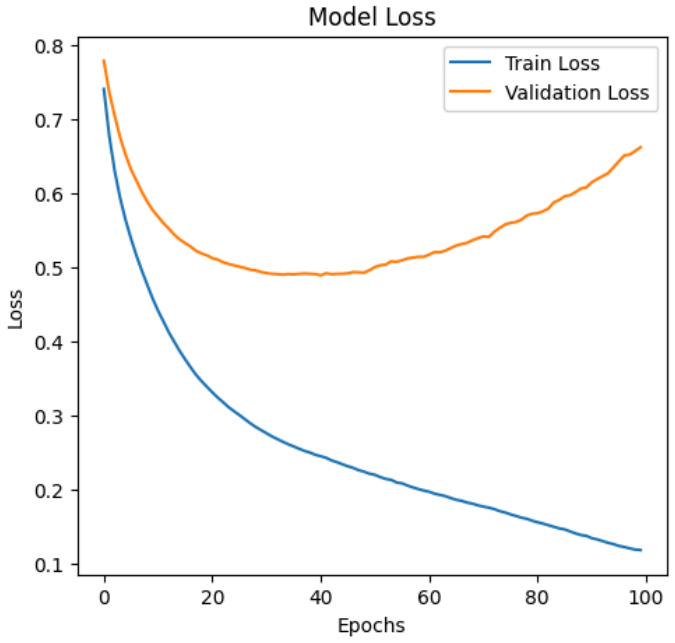
Model: "sequential"

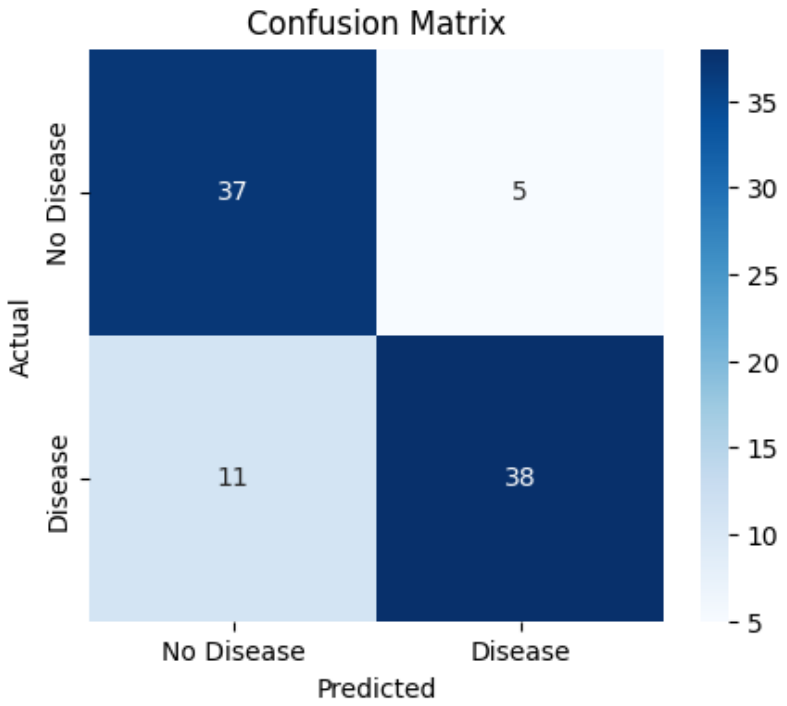
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 16)	224
dense_1 (Dense)	(None, 8)	136
dense_2 (Dense)	(None, 1)	9

Total params: 369 (1.44 KB)
Trainable params: 369 (1.44 KB)
Non-trainable params: 0 (0.00 B)

```
Step 7: Compiling the Model...


Step 8: Training the Model...

Epoch 1/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 3s 41ms/step - accuracy: 0.4350 - loss: 0.7935 - val_accuracy: 0.4651 - val_loss: 0.7781
Epoch 2/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.5397 - loss: 0.6965 - val_accuracy: 0.5581 - val_loss: 0.7353
Epoch 3/100

Step 9: Evaluating the Model...

3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8262 - loss: 0.7102
Test Loss: 0.6702
Test Accuracy: 82.42%
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
Precision: 0.8837
Recall:    0.7755
F1 Score:  0.8261
ROC-AUC:   0.8882

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.88      0.82        42
           1       0.88      0.78      0.83        49

    accuracy                           0.82        91
   macro avg       0.83      0.83      0.82        91
weighted avg       0.83      0.82      0.82        91

Confusion Matrix:
 [[37  5]
 [11 38]]

Step 10: Visualization...
```








```
Step 11: Predicting Heart Disease for a New Patient...

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
Predicted Status for New Patient: Heart Disease
Predicted Probability: 0.9884
```

Experiment 4

Title: Sentiment Analysis of Social Media tweets.

Objective: To develop a machine learning model using TensorFlow that can automatically classify social media tweets as positive, negative, or neutral, helping understand public opinion and sentiment trends on various topics.

Theory: Sentiment analysis is a natural language processing (NLP) technique that identifies and extracts subjective information from textual data, such as opinions, emotions, or attitudes. Social media platforms like Twitter generate vast amounts of user-generated content that reflect public sentiment toward events, products, or personalities.

The process involves several key steps: text preprocessing (cleaning, tokenization, and encoding), feature extraction, and model training using machine learning or deep learning algorithms. Deep learning models, particularly those implemented using TensorFlow, can capture complex patterns in text through embedding layers, recurrent neural networks (RNNs), long short-term memory networks (LSTMs), or transformer-based architectures.

Sentiment analysis has widespread applications, including brand monitoring, customer feedback analysis, political trend prediction, and public health surveillance. By classifying tweets into positive, negative, or neutral sentiment, organizations can gain actionable insights from real-time social media data.

Key Advantages of Deep Learning Models:

1. **Superior Pattern Recognition**
    a. Automatic feature learning: No manual feature engineering
    b. Complex pattern detection: Identify subtle linguistic patterns
    c. Context understanding: Capture semantic relationships
    d. Scalability: Handle large datasets efficiently
2. **Contextual Understanding**
    a. Sequential processing: Understand word order and dependencies
    b. Semantic meaning: Go beyond simple keyword matching
    c. Nuanced sentiment: Detect sarcasm, irony, and complex emotions
    d. Multi-layered analysis: Extract hierarchical features

Applications in Healthcare and Public Health

- Public health surveillance: Monitor health-related discussions
- Drug safety monitoring: Detect adverse reactions from social media
- Mental health insights: Identify depression and anxiety indicators
- Health campaign effectiveness: Measure public response to health initiatives

Software / Library	Version / Notes
Python	3.8 or above
TensorFlow	2.x
Keras	Integrated with TensorFlow
pandas	1.x
numpy	1.x
matplotlib	3.x
seaborn	0.11 or above
scikit-learn	0.24 or above
nltk	Latest, for tokenization & stopwords
re (Regular Expressions)	Built-in Python library

Experimental Procedure:

Dataset: Using freely available twitter sentiment dataset from kaggle.

Step 1: Import Libraries

- Import Python libraries such as TensorFlow, Keras, pandas, numpy, matplotlib, seaborn, scikit-learn, and NLTK.

Step 2: Data Collection

- Load a dataset of tweets (CSV or JSON format) containing the tweet text and sentiment label (positive, negative, neutral).
- Inspect the data using head(), info(), and describe() functions.

Step 3: Data Preprocessing

- Clean text by removing URLs, mentions, hashtags, emojis, punctuation, and special characters.
- Convert text to lowercase.
- Tokenize text into words or sequences.
- Remove stopwords and optionally perform stemming or lemmatization.

Step 4: Text Vectorization

- Convert text into numerical format using techniques like Tokenizer and padding sequences.
- Use embeddings such as Word2Vec, GloVe, or TensorFlow embedding layers for deep learning input.

Step 5: Train-Test Split

- Split the dataset into training and testing sets (e.g., 70% train, 30% test).

Step 6: Model Building

- Build a TensorFlow deep learning model for classification, such as LSTM, BiLSTM, or simple dense layers with embedding input.

Step 7: Model Compilation

- Compile the model using appropriate optimizer (e.g., Adam), loss function (categorical crossentropy for multi-class), and metrics (accuracy, precision, recall).

Step 8: Model Training

- Train the model on the training set and validate on a validation split.
- Monitor training loss and accuracy for each epoch.

Step 9: Model Evaluation

- Evaluate model performance on the test set using metrics such as accuracy, precision, recall, F1-score, and confusion matrix.
- Optionally, visualize performance metrics and training history.

Program:

```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.utils import to_categorical
import nltk
from nltk.corpus import stopwords

# Download NLTK data
nltk.download('stopwords')

# Load dataset
df = pd.read_csv('twitter.csv')
df.columns = ['ID', 'Platform', 'Sentiment', 'Tweet Text']

print("Dataset Head:")
print(df.head())

# Data Cleaning
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = text.lower()
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'\@\w+|\#','', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    tokens = text.split()
    tokens = [word for word in tokens if word not in stop_words]
    return ' '.join(tokens)

print("\nCleaning Tweets...")
df['clean_text'] = df['Tweet Text'].apply(clean_text)

# Encoding Sentiment Labels
sentiment_mapping = {'Positive': 2, 'Neutral': 1, 'Negative': 0}

# --- Fix: Filter for valid sentiment values before mapping ---
valid_sentiments = list(sentiment_mapping.keys())
df = df[df['Sentiment'].isin(valid_sentiments)].copy() # Use .copy() to avoid SettingWithCopyWarning
# --- End Fix ---
```

```
df['sentiment_label'] = df['Sentiment'].map(sentiment_mapping)


# Prepare text data
texts = df['clean_text'].values
labels = df['sentiment_label'].values

# Tokenization
max_features = 10000
max_len = 150

tokenizer = Tokenizer(num_words=max_features, oov_token="<OOV>")
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='post')

# Split data
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, labels, test_size=0.3, random_state=42)

# Convert labels to categorical
y_train_cat = to_categorical(y_train, num_classes=3)
y_test_cat = to_categorical(y_test, num_classes=3)

# Build the model
model = Sequential([
    Embedding(input_dim=max_features, output_dim=128, input_length=max_len),
    Bidirectional(LSTM(64, dropout=0.3, recurrent_dropout=0.3)),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')
])

# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

print("\nModel Summary:")
model.summary()
```

```
# Train the model
history = model.fit(X_train, y_train_cat,
                    batch_size=64,
                    epochs=10,
                    validation_split=0.2,
                    verbose=1)

# Evaluate the model
loss, acc = model.evaluate(X_test, y_test_cat, verbose=0)
print(f"\n Test Accuracy: {acc:.4f}")

# Classification report
y_pred_probs = model.predict(X_test)
y_pred = np.argmax(y_pred_probs, axis=1)
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Negative', 'Neutral', 'Positive']))

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues', xticklabels=['Negative', 'Neutral',
'Positive'], yticklabels=['Negative', 'Neutral', 'Positive'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy over epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
```

```
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Predict on a new tweet
new_tweet = "I love the new product!"
clean_new_tweet = clean_text(new_tweet)
seq = tokenizer.texts_to_sequences([clean_new_tweet])
padded_seq = pad_sequences(seq, maxlen=max_len, padding='post')
pred_prob = model.predict(padded_seq)
pred_class = np.argmax(pred_prob, axis=1)[0]
reverse_mapping = {v:k for k,v in sentiment_mapping.items()}
predicted_sentiment = reverse_mapping[pred_class]

print("\nNew Tweet:")
print(new_tweet)
print("Predicted Sentiment:", predicted_sentiment)
```

Output:

```
Dataset Head:
     ID   Platform Sentiment  \
0   352     Amazon   Neutral
1  8312  Microsoft  Negative
2  4371      CS-GO  Negative
3  4433     Google   Neutral
4  6273       FIFA  Negative

                                          Tweet Text
0  BBC News - Amazon boss Jeff Bezos rejects clai...
1  @Microsoft Why do I pay for WORD when it funct...
2  CSGO matchmaking is so full of closet hacking,...
3  Now the President is slapping Americans in the...
4  Hi @EAHelp I've had Madeleine McCann in my cel...

Cleaning Tweets...

Model Summary:
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Model: "sequential_8"
```

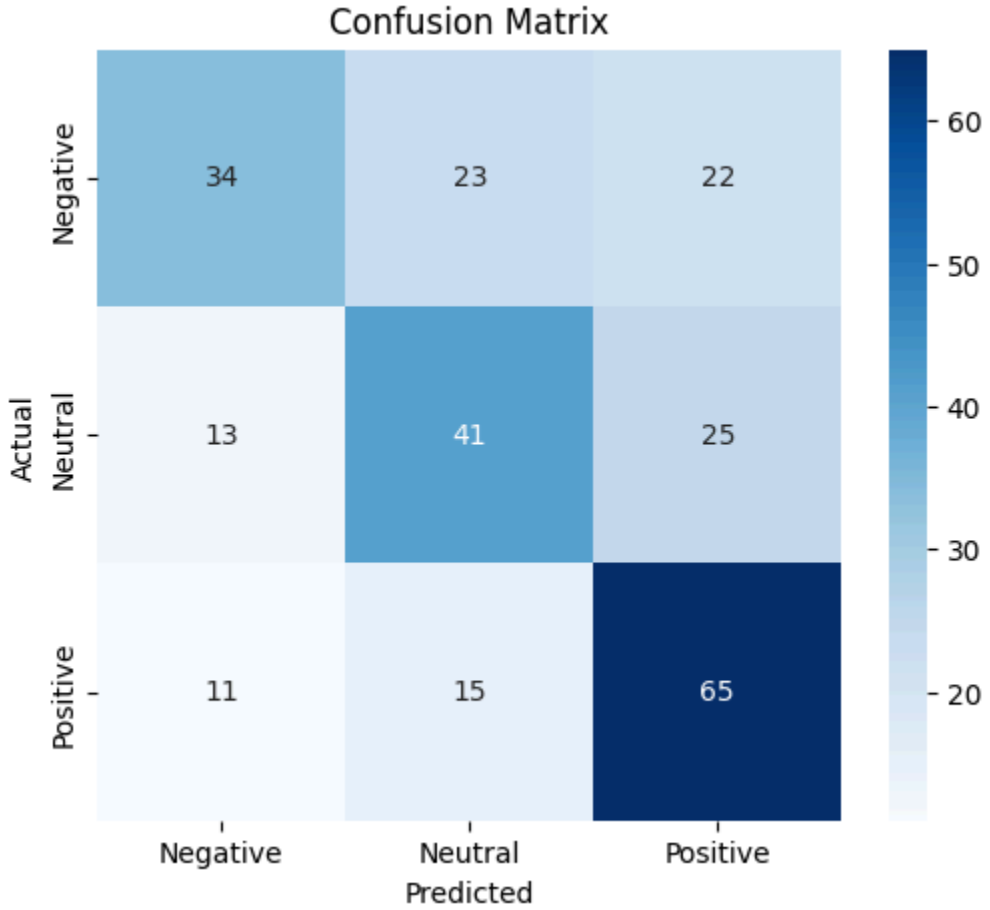
Layer (type)	Output Shape	Param #
embedding_8 (Embedding)	?	0 (unbuilt)
bidirectional_2 (Bidirectional)	?	0 (unbuilt)
dense_14 (Dense)	?	0 (unbuilt)
dropout_12 (Dropout)	?	0
dense_15 (Dense)	?	0 (unbuilt)

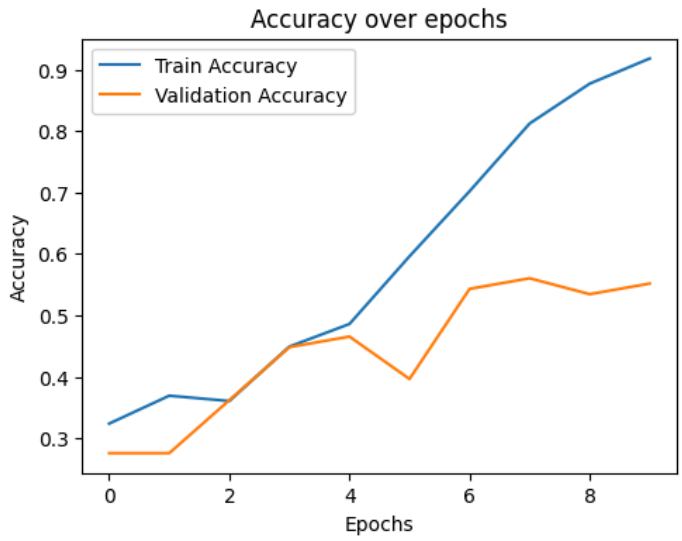
```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

Epoch 1/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 14s 569ms/step - accuracy: 0.3179 - loss: 1.1012 - val_accuracy: 0.2759 - val_loss: 1.0982
Epoch 2/10
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 620ms/step - accuracy: 0.3808 - loss: 1.0926 - val_accuracy: 0.2759 - val_loss: 1.0983
Epoch 3/10
```

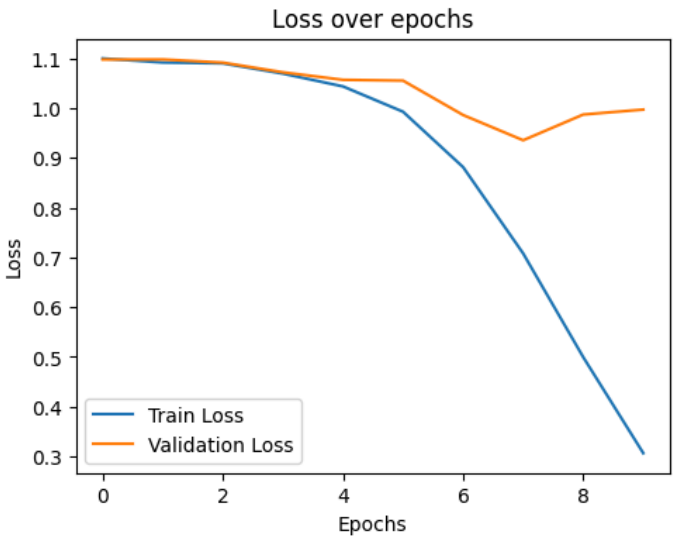
✅ Test Accuracy: 0.5622
8/8 ━━━━━━━━━━━━━━━━━━━━ **2s** 173ms/step

Classification Report:

	precision	recall	f1-score	support
Negative	0.59	0.43	0.50	79
Neutral	0.52	0.52	0.52	79
Positive	0.58	0.71	0.64	91
accuracy			0.56	249
macro avg	0.56	0.55	0.55	249
weighted avg	0.56	0.56	0.56	249








```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 90ms/step

New Tweet:
I love the new product!
Predicted Sentiment: Positive
```

Experiment 5

Title: To apply all standard Natural Language Processing (NLP) preprocessing steps on a given text in order to prepare it for further analysis or model training.

Objective: To systematically preprocess raw textual data by applying standard NLP techniques - such as cleaning, tokenization, stopword removal, stemming, lemmatization, and vectorization - so that the text is transformed into a structured, consistent, and meaningful format suitable for analysis or input into machine learning models.

Theory: Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that focuses on enabling computers to understand, interpret, manipulate, and generate human language in a way that is meaningful and useful. It bridges the gap between human communication and machine comprehension by applying algorithms, linguistic rules, and statistical models to process textual data. NLP is widely used in applications like chatbots, sentiment analysis, machine translation, speech recognition, and text summarization.

However, real-world text data is often noisy, unstructured, and inconsistent. Before performing any analysis or training a machine learning model, it is essential to preprocess the text to clean and standardize it. Preprocessing helps improve the performance and accuracy of NLP models by transforming raw text into a structured format that can be easily analyzed or fed into algorithms.

Standard NLP Preprocessing Steps

1. **Text Cleaning**
   The first step is to remove unwanted characters such as punctuation marks, special symbols, extra spaces, HTML tags, or numeric digits that don't add meaning to the text. Cleaning helps focus on the core content and reduces noise.
2. **Lowercasing**
   All text is converted to lowercase so that words like "Apple" and "apple" are treated as the same word, helping the model avoid unnecessary duplication in vocabulary.
3. **Tokenization**
   Tokenization is the process of splitting text into smaller units called tokens, such as words or sentences. This allows algorithms to process each token separately and understand the structure of the language.
4. **Stopword Removal**
   Stopwords are commonly used words like "is," "the," "and," "of," which do not carry significant meaning and may add unnecessary clutter to the data. Removing stopwords helps focus on important words.
5. **Stemming**
   Stemming reduces words to their root form by removing suffixes. For example,

"running," "runner," and "runs" would all be reduced to "run." This helps group similar words together.

6. **Lemmatization**
   Lemmatization also reduces words to their base or dictionary form, but it is more sophisticated than stemming. For example, "better" becomes "good," and "running" becomes "run." This process uses vocabulary and morphological analysis to understand the correct base form.
7. **Part of Speech (POS) Tagging**
   POS tagging identifies the grammatical category (noun, verb, adjective, etc.) of each token. This helps in understanding the role of words in sentences and assists in more complex tasks like parsing or named entity recognition.
8. **Vectorization / Encoding**
   After preprocessing, the text is converted into numerical representations such as Bag of Words (BoW), TF-IDF (Term Frequency-Inverse Document Frequency), or word embeddings like Word2Vec or GloVe. These representations allow machine learning models to process text mathematically.

Software / Library	**Description**
Python	Programming language used for implementing NLP
Jupyter Notebook	Interactive development environment for Python
NLTK (Natural Language Toolkit)	Library for text processing, tokenization, stemming, etc.
spaCy	Industrial-strength NLP library for tokenization, lemmatization, and POS tagging
re (Regular Expressions)	Library to handle text cleaning and pattern matching
pandas	Used for data handling and manipulation
scikit-learn	Provides tools for text vectorization like TF-IDF and CountVectorizer
matplotlib / seaborn	For visualizing text data and patterns (optional)

Experimental Procedure:

Dataset Used:
Freely available text datasets, such as news articles, product reviews, tweets, or medical text data.

Step 1: Import Libraries

- Import Python libraries required for NLP and text analysis, such as nltk, re, pandas, sklearn, contractions, and string.

Step 2: Data Selection

- Load the dataset using pandas.read_csv() or any text source.
- Select relevant text columns for preprocessing and analysis.

Step 3: Text Cleaning

- Expand contractions to standardize text.
- Remove special characters, numbers, punctuation, URLs, HTML tags, and extra spaces.
- Convert all text to lowercase.

Step 4: Tokenization

- Split text into individual words and sentences for further analysis.

Step 5: Stopword Removal

- Remove common stopwords (e.g., “the”, “is”, “and”) to focus on meaningful words.

Step 6: POS Tagging

- Assign part-of-speech tags (noun, verb, adjective, etc.) to each token.

Step 7: Stemming and Lemmatization

- Apply stemming to reduce words to their root form.
- Apply lemmatization with POS awareness to obtain dictionary forms of words.

Step 8: Final Token Filtering

- Remove very short or irrelevant tokens.
- Combine processed tokens back into clean text for analysis or modeling.

Step 9: Vectorization

- Convert text into numerical representation for machine learning:
    - Bag of Words (BoW)
    - TF-IDF (Term Frequency-Inverse Document Frequency)

Program:

```
import re
import nltk
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
```

```
from nltk.tag import pos_tag
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import contractions

# Download necessary NLTK resources
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('averaged_perceptron_tagger', quiet=True)
nltk.download('omw-1.4', quiet=True)

# Helper function: convert POS tags for lemmatization
def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

# NLP preprocessing pipeline with stepwise outputs
def preprocess_text_verbose(text, vectorize=True):
    print("STEP 0: Original Text")
    print(text, "\n")

    # Step 1: Expand contractions
    text = contractions.fix(text)
    print("STEP 1: Expand Contractions")
    print(text, "\n")

    # Step 2: Clean text (remove HTML, URLs, emails, numbers, punctuation, extra spaces)
    text_clean = re.sub(r'<[^>]+>', '', text)
    text_clean = re.sub(r'http\S+|www.\S+', '', text_clean)
    text_clean = re.sub(r'\S+@\S+', '', text_clean)
    text_clean = re.sub(r'[^A-Za-z\s]', '', text_clean)
    text_clean = re.sub(r'\s+', ' ', text_clean).strip()
    print("STEP 2: Cleaned Text")
    print(text_clean, "\n")

    # Step 3: Lowercasing
    text_lower = text_clean.lower()
    print("STEP 3: Lowercased Text")
```

```
print(text_lower, "\n")

# Step 4: Tokenization
tokens = word_tokenize(text_lower)
print("STEP 4: Tokens")
print(tokens, "\n")

# Step 5: Stopword removal
stop_words = set(stopwords.words('english'))
tokens_no_stop = [t for t in tokens if t not in stop_words and len(t) > 1]
print("STEP 5: Tokens after Stopword Removal")
print(tokens_no_stop, "\n")

# Step 6: POS tagging
pos_tags = pos_tag(tokens_no_stop)
print("STEP 6: POS Tags")
print(pos_tags, "\n")

# Step 7: Stemming
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(t) for t in tokens_no_stop]
print("STEP 7: Stemmed Tokens")
print(stemmed_tokens, "\n")

# Step 8: Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(t, get_wordnet_pos(p)) for t, p in pos_tags]
print("STEP 8: Lemmatized Tokens")
print(lemmatized_tokens, "\n")

# Step 9: Filter very short words
final_tokens = [t for t in lemmatized_tokens if len(t) >= 3]
final_text = ' '.join(final_tokens)
print("STEP 9: Final Tokens")
print(final_tokens, "\n")
print("Final Processed Text")
print(final_text, "\n")


# Bag of Words
cv = CountVectorizer()
bow_matrix = cv.fit_transform([final_text])
print("STEP 10a: Bag of Words Feature Names")
print(cv.get_feature_names_out())
print("BoW Matrix:\n", bow_matrix.toarray(), "\n")
```

```
    # TF-IDF
    tfidf = TfidfVectorizer()
    tfidf_matrix = tfidf.fit_transform([final_text])
    print("STEP 10b: TF-IDF Feature Names")
    print(tfidf.get_feature_names_out())
    print("TF-IDF Matrix:\n", tfidf_matrix.toarray(), "\n")

    # Return results if needed
    return {
        'final_tokens': final_tokens,
        'final_text': final_text,
        'pos_tags': pos_tags,
        'stemmed_tokens': stemmed_tokens,
        'lemmatized_tokens': lemmatized_tokens
    }

# Example usage
if __name__ == "__main__":
    sample_text = """
    Natural Language Processing (NLP) is a subfield of AI that focuses on enabling computers to
understand human language.
    It is widely used in chatbots, sentiment analysis, machine translation, and text summarization.
    """
    processed = preprocess_text_verbose(sample_text)
    print("Final Processed Text:\n", processed['final_text'])
    print("\nTokens:\n", processed['final_tokens'])
    print("\nPOS Tags:\n", processed['pos_tags'])
```

Output:

```
STEP 0: Original Text

    Natural Language Processing (NLP) is a subfield of AI that focuses on enabling computers to understand human language.
    It is widely used in chatbots, sentiment analysis, machine translation, and text summarization.


STEP 1: Expand Contractions

    Natural Language Processing (NLP) is a subfield of AI that focuses on enabling computers to understand human language.
    It is widely used in chatbots, sentiment analysis, machine translation, and text summarization.
```

```
STEP 2: Cleaned Text
('Natural Language Processing NLP is a subfield of AI that focuses on enabling '
 'computers to understand human language It is widely used in chatbots '
 'sentiment analysis machine translation and text summarization')


STEP 3: Lowercased Text
('natural language processing nlp is a subfield of ai that focuses on enabling '
 'computers to understand human language it is widely used in chatbots '
 'sentiment analysis machine translation and text summarization')


STEP 4: Tokens
['natural',
 'language',
 'processing',
 'nlp',
 'is',
 'a',
 'subfield',
 'of',
 'ai',
 'that',
 'focuses',
 'on',
 'enabling',
 'computers',
 'to',
 'understand',
 'human',
 'language',
 'it',
 'is',
 'widely',
 'used',
 'in',
 'chatbots',
 'sentiment',
 'analysis',
 'machine',
 'translation',
 'and',
 'text',
 'summarization']
```

```
STEP 5: Tokens after Stopword Removal
['natural',
 'language',
 'processing',
 'nlp',
 'subfield',
 'ai',
 'focuses',
 'enabling',
 'computers',
 'understand',
 'human',
 'language',
 'widely',
 'used',
 'chatbots',
 'sentiment',
 'analysis',
 'machine',
 'translation',
 'text',
 'summarization']


STEP 6: POS Tags
[('natural', 'JJ'),
 ('language', 'NN'),
 ('processing', 'NN'),
 ('nlp', 'JJ'),
 ('subfield', 'NN'),
 ('ai', 'NN'),
 ('focuses', 'VBZ'),
 ('enabling', 'VBG'),
 ('computers', 'NNS'),
 ('understand', 'VBP'),
 ('human', 'JJ'),
 ('language', 'NN'),
 ('widely', 'RB'),
 ('used', 'VBN'),
 ('chatbots', 'NNS'),
 ('sentiment', 'JJ'),
 ('analysis', 'NN'),
 ('machine', 'NN'),
 ('translation', 'NN'),
 ('text', 'NN'),
 ('summarization', 'NN')]
```

```
STEP 7: Stemmed Tokens
['natur',
 'languag',
 'process',
 'nlp',
 'subfield',
 'ai',
 'focus',
 'enabl',
 'comput',
 'understand',
 'human',
 'languag',
 'wide',
 'use',
 'chatbot',
 'sentiment',
 'analysi',
 'machin',
 'translat',
 'text',
 'summar']


STEP 8: Lemmatized Tokens
['natural',
 'language',
 'processing',
 'nlp',
 'subfield',
 'ai',
 'focus',
 'enable',
 'computer',
 'understand',
 'human',
 'language',
 'widely',
 'use',
 'chatbots',
 'sentiment',
 'analysis',
 'machine',
 'translation',
 'text',
 'summarization']
```

```
STEP 9: Final Tokens
['natural',
 'language',
 'processing',
 'nlp',
 'subfield',
 'focus',
 'enable',
 'computer',
 'understand',
 'human',
 'language',
 'widely',
 'use',
 'chatbots',
 'sentiment',
 'analysis',
 'machine',
 'translation',
 'text',
 'summarization']


Final Processed Text
('natural language processing nlp subfield focus enable computer understand '
 'human language widely use chatbots sentiment analysis machine translation '
 'text summarization')


STEP 10a: Bag of Words Feature Names
array(['analysis', 'chatbots', 'computer', 'enable', 'focus', 'human',
       'language', 'machine', 'natural', 'nlp', 'processing', 'sentiment',
       'subfield', 'summarization', 'text', 'translation', 'understand',
       'use', 'widely'], dtype=object)
BoW Matrix:

array([[1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])


STEP 10b: TF-IDF Feature Names
array(['analysis', 'chatbots', 'computer', 'enable', 'focus', 'human',
       'language', 'machine', 'natural', 'nlp', 'processing', 'sentiment',
       'subfield', 'summarization', 'text', 'translation', 'understand',
       'use', 'widely'], dtype=object)
TF-IDF Matrix:

array([[0.21320072, 0.21320072, 0.21320072, 0.21320072, 0.21320072,
        0.21320072, 0.42640143, 0.21320072, 0.21320072, 0.21320072,
        0.21320072, 0.21320072, 0.21320072, 0.21320072, 0.21320072,
        0.21320072, 0.21320072, 0.21320072, 0.21320072]])
```

```
Final Processed Text:

('natural language processing nlp subfield focus enable computer understand '
 'human language widely use chatbots sentiment analysis machine translation '
 'text summarization')

Tokens:

['natural',
 'language',
 'processing',
 'nlp',
 'subfield',
 'focus',
 'enable',
 'computer',
 'understand',
 'human',
 'language',
 'widely',
 'use',
 'chatbots',
 'sentiment',
 'analysis',
 'machine',
 'translation',
 'text',
 'summarization']

POS Tags:

[('natural', 'JJ'),
 ('language', 'NN'),
 ('processing', 'NN'),
 ('nlp', 'JJ'),
 ('subfield', 'NN'),
 ('ai', 'NN'),
 ('focuses', 'VBZ'),
 ('enabling', 'VBG'),
 ('computers', 'NNS'),
 ('understand', 'VBP'),
 ('human', 'JJ'),
 ('language', 'NN'),
 ('widely', 'RB'),
 ('used', 'VBN'),
 ('chatbots', 'NNS'),
 ('sentiment', 'JJ'),
 ('analysis', 'NN'),
 ('machine', 'NN'),
 ('translation', 'NN'),
 ('text', 'NN'),
 ('summarization', 'NN')]
```

Experiment 6

Title: To develop a Named Entity Recognition (NER) system for identifying and classifying key medical entities such as diseases, medications, symptoms, procedures, and anatomical terms from clinical or biomedical texts.

Objective: To design and implement a Named Entity Recognition (NER) system capable of identifying and classifying key medical entities such as diseases, medications, symptoms, procedures, and anatomical terms from clinical or biomedical texts, thereby facilitating structured information extraction for healthcare and research applications.

Theory: Natural Language Processing (NLP) plays a significant role in analyzing biomedical and clinical texts, which are rich in domain-specific terminology. One important task is Named Entity Recognition (NER), which aims to identify and classify entities such as diseases, symptoms, medications, procedures, and anatomical terms. Medical NER is essential in many real-world applications, including clinical decision support, drug discovery, biomedical research, electronic health records (EHRs), and healthcare chatbots

What is NER?
NER is a sub-task of Information Extraction (IE) that automatically locates and classifies entities into predefined categories. In the medical domain, these categories can include:

- Diseases (e.g., Diabetes, Cancer, Malaria)
- Medications/Drugs (e.g., Paracetamol, Aspirin, Ibuprofen)
- Symptoms (e.g., fever, cough, headache)
- Procedures (e.g., MRI scan, chemotherapy, surgery)
- Anatomical Terms (e.g., liver, heart, lungs)

For example:
“The patient was diagnosed with diabetes and prescribed metformin.”
Here, diabetes → Disease and metformin → Medication.

Challenges in Medical NER

- Ambiguity of Terms – Some words can mean different things depending on context. Example: “cold” (symptom vs. temperature).
- Abbreviations & Acronyms – E.g., “HTN” for hypertension, “MRI” for magnetic resonance imaging.
- Domain-Specific Vocabulary – General-purpose NER models trained on news or generic texts often perform poorly on medical corpora.

- Complex Structure of Clinical Texts – Reports often contain shorthand, unstructured notes, or misspellings.

Approaches for NER

- Rule-Based Systems – Early systems used dictionaries and handcrafted rules, but they lacked flexibility and scalability.
- Machine Learning-Based Systems – Algorithms like Conditional Random Fields (CRFs) or Support Vector Machines (SVMs) were used with linguistic features.
- Deep Learning-Based Systems – Neural architectures such as BiLSTM-CRF and Transformers improved accuracy by learning contextual representations.
- Transformer-Based Pretrained Models – State-of-the-art systems like BioBERT, SciSpacy, PubMedBERT leverage biomedical corpora to achieve high performance.

Workflow of a Medical NER System

- Text Preprocessing – Cleaning text (removing stopwords, special characters).
- NER Model – Applying a pre-trained biomedical model (e.g., SciSpacy or Med7).
- Entity Extraction – Identifying words/phrases and tagging them with categories.
- Output Representation – Storing extracted entities in structured form (JSON, database, etc.).

Applications

- Clinical Decision Support – Extracting diseases and symptoms from patient reports.
- Pharmacovigilance – Monitoring adverse drug reactions.
- Medical Research – Summarizing biomedical literature.
- EHR Systems – Automating entity extraction for structured record-keeping.

Software requirements and tools:

Software / Library	Version / Notes	Purpose
Python	3.8 or above	Programming language
transformers	Latest	Pre-trained models for NER
numpy	Latest	Numerical computations (confidence averaging)
collections	—	Counting entity types
re	—	Regular expressions if needed (optional)
torch or tensorflow	Installed with transformers	Backend for running models

Experiment procedure:

Step 1 – Environment Setup

- Install Python and required libraries: transformers, numpy, optionally matplotlib.
- Ensure GPU support if available for faster computation.

Step 2 – Load Pre-trained Model

- Import the pipeline class from the transformers library.
- Load a biomedical NER pipeline using the model d4data/biomedical-ner-all with aggregation strategy “simple”.

Step 3 – Prepare Input Data

- Write or obtain medical text containing various medical entities like diseases, medications, symptoms, and procedures.

Step 4 – Run NER Pipeline

- Apply the pipeline to the medical text to extract entities and their types along with confidence scores.

Step 5 – Display Extracted Entities

- Print each detected entity with its label (e.g., aspirin --> Medication).

Step 6 – Analyze Results

- Calculate the total number of entities detected.
- Identify unique entity types present in the text.
- Compute the average confidence score of the predictions.
- Count occurrences of each entity type using Counter.

Step 7 – Summarize Findings

- Display a structured results summary showing total entities, unique types, average confidence, and count per entity type.

Program:

```
from transformers import pipeline
import re
```

```
import numpy as np
from collections import Counter

# Load pretrained biomedical NER pipeline
ner_pipeline = pipeline("ner", model="d4data/biomedical-ner-all",
aggregation_strategy="simple")
# Medical text
text = (
    "John, a 54-year-old male, was admitted to the hospital with chest pain, shortness of breath,
and dizziness. "
    "He has a history of type 2 diabetes and hypertension. His doctor prescribed metformin and
atenolol to manage his conditions. "
    "An MRI scan of the heart and lungs was scheduled to assess the extent of damage. "
    "He also takes aspirin daily to reduce the risk of blood clots and uses inhalers containing
albuterol for asthma. "
    "Blood tests revealed low vitamin D and calcium levels, and supplements were
recommended."
)

# Run NER
results = ner_pipeline(text)

# Display extracted entities
print("Drug & Disease Entities:")
for r in results:
    print(r['word'], "-->", r['entity_group'])

total_entities = len(results)
unique_entity_types = len(entity_counter)
average_confidence = np.mean([r['score'] for r in results])
entity_counter = Counter([r['entity_group'] for r in results])

# Display formatted results summary
print("\n" + "="*40)
print("RESULTS SUMMARY")
print("="*40)
print(f"Total entities detected: {total_entities}")
print(f"Unique entity types: {unique_entity_types}")
print(f"Average confidence: {average_confidence:.3f}\n")
```

```
print("Entity Type Counts:")
for entity, count in entity_counter.items():
    print(f"{entity}: {count}")

print("\nMedical NER analysis complete!")
```

Output:

```
Device set to use cpu
Drug & Disease Entities:
54 - year - old --> Age
male --> Sex
admitted --> Clinical_event
chest --> Biological_structure
shortness of breath --> Sign_symptom
type 2 diabetes --> History
hyper --> History
metformin --> Medication
atenolol --> Medication
mri --> Diagnostic_procedure
heart --> Biological_structure
lungs --> Biological_structure
as --> Medication
##pirin --> Medication
daily --> Frequency
blood clots --> Sign_symptom
al --> Medication
blood tests --> Diagnostic_procedure
low --> Lab_value
vitamin d --> Diagnostic_procedure
calcium --> Diagnostic_procedure
supplements --> Diagnostic_procedure

========================================
RESULTS SUMMARY
========================================
Total entities detected: 22
Unique entity types: 10
Average confidence: 0.928

Entity Type Counts:
Age: 1
Sex: 1
Clinical_event: 1
Biological_structure: 3
Sign_symptom: 2
History: 2
Medication: 5
Diagnostic_procedure: 5
Frequency: 1
Lab_value: 1

Medical NER analysis complete!
```