Experiment No.8	
Explainable AI in healthcare for model interpretation	
Date of Performance:	
Date of Submission:	

Aim: To use Explainable AI in healthcare for model interpretation.
Objective: Given example statements or sentences with logo, use Explainable AI to display prediction probabilities using Lime TextExplainer
Theory:
Explainable AI (XAI) is a field of artificial intelligence that focuses on developing techniques and methods to make machine learning models more understandable and interpretable for humans. Interpretability is crucial for a variety of reasons, including trust, accountability, and making informed decisions based on AI predictions. Here are some key concepts and techniques in explainable AI for model interpretation:

Feature Importance: Understanding which features or input variables contribute the most to a model's predictions is a fundamental aspect of model interpretability. Common methods for feature importance include permutation feature importance, SHAP (SHapley Additive exPlanations), and LIME (Local Interpretable Model-agnostic Explanations).

Model-Agnostic Interpretability: Many XAI techniques are model-agnostic, meaning they can be applied to various machine learning models. Techniques like LIME and SHAP are not tied to any specific model and can provide explanations for any black-box model.

Decision Trees and Rule-Based Models: Decision trees and rule-based models are inherently interpretable. They partition the data into branches based on feature values, allowing users to understand the decision-making process. However, they might not be as accurate as complex models for some tasks.

Local Interpretability: Local interpretability focuses on explaining the predictions of a model for a specific instance or data point. This can be useful for understanding why a model made a particular prediction in a given context.

Global Interpretability: Global interpretability aims to provide a holistic view of the model's behavior across the entire dataset. This can include insights into the relationships between features and the overall decision boundaries.

Visualization: Visualization techniques can help users understand model behavior by representing data and model predictions graphically. Tools like partial dependence plots and feature importance plots are commonly used for this purpose.

SHAP Values: SHAP (SHapley Additive exPlanations) is a widely used technique that provides a unified measure of feature importance and can explain the output of any machine learning model. It is based on game theory and provides a coherent and consistent way to allocate contributions of each feature to a prediction.

Counterfactual Explanations: Counterfactual explanations provide a different perspective by showing what changes in input features would lead to a different model prediction. This is particularly useful for understanding how to achieve a desired outcome or why a specific prediction was made.

Anchors: Anchors are simple, human-friendly rules that describe the behavior of a model for a given instance. They define a minimal set of conditions that, when met, guarantee a certain prediction.

Ethical Considerations: Explainable AI is crucial in ethical AI practices. It helps identify and mitigate biases in models and ensures that AI systems do not make discriminatory or harmful decisions.

Overall, explainable AI is an evolving field that offers a range of methods and techniques to enhance the transparency and interpretability of machine learning models. These techniques help make AI more trustworthy and accessible to users, including non-experts, regulators, and stakeholders who need to understand and trust AI decisions.

Code: -

```
!pip install lime pandas scikit-learn
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from lime.lime_tabular import LimeTabularExplainer
from google.colab import drive
from google.colab import files
import io

# Mount Google Drive and upload dataset
drive.mount('/content/drive')
uploaded = files.upload()

# Load dataset
dataset =
pd.read_csv(io.BytesIO(uploaded['heart_cleveland_upload.csv']))
print(dataset.head())

# Check column names and data types
print(dataset.info())

# Define features and target
X = dataset.drop(columns=['condition'])  # Features
```

```
y = dataset['condition']  # Target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train Logistic Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Make predictions and evaluate accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")

# Initialize the LIME Tabular Explainer
explainer = LimeTabularExplainer(
    X_train.values,
    feature_names=X.columns,
    class_names=['No Condition', 'Condition'],
    mode='classification'
)
```

```
# Select an instance from the test set
instance_idx = 0  # First instance in the test set
instance = X_test.iloc[instance_idx].values.reshape(1, -1)

# Generate explanation for the selected instance
exp = explainer.explain_instance(instance[0], model.predict_proba, num_features=10)

# Display explanation
exp.show_in_notebook(show_all=False)
```

Output:

```
[1] !pip install lime pandas scikit-learn

Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 6.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from lime) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from lime) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from lime) (1.13.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from lime) (4.66.5)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.10/dist-packages (from lime) (0.24.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: networkx>=2.8 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (3.3)
Requirement already satisfied: pillow>=9.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (10.4.0)
Requirement already satisfied: imageio>=2.33 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (2.35.1)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (2024.9.20)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (24.1)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.12->lime) (0.4)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (1.4.7)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->lime) (3.1.4)
Building wheels for collected packages: lime
```

```
Successfully installed lime-0.2.0.1
```

```
[2] import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score
    from lime.lime_text import LimeTextExplainer
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.pipeline import make_pipeline
```

```
[3] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[12] from google.colab import files
     uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving heart_cleveland_upload.csv to heart_cleveland_upload.csv
```

```
[13] import io
     dataset = pd.read_csv(io.BytesIO(uploaded['heart_cleveland_upload.csv']))
```

```
[16] print(dataset.head())
```

```
[16] print(dataset.head())

     # Check column names and data types
     print(dataset.info())
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   69    1   0       160   234    1        2      131      0      0.1      1   
1   69    0   0       140   239    0        0      151      0      1.8      0   
2   66    0   0       150   226    0        0      114      0      2.6      2   
3   65    1   0       138   282    1        2      174      0      1.4      1   
4   64    1   0       110   211    0        2      144      1      1.8      1   

   ca  thal  condition  
0   1     0          0  
1   2     0          0  
2   0     0          0  
3   1     0          1  
4   0     0          0  
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 297 entries, 0 to 296
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   age        297 non-null    int64  
 1   sex        297 non-null    int64  
 2   cp         297 non-null    int64  
 3   trestbps   297 non-null    int64  
 4   chol       297 non-null    int64  
 5   fbs        297 non-null    int64  
 6   restecg    297 non-null    int64  
 7   thalach    297 non-null    int64  
 8   exang      297 non-null    int64  
 9   oldpeak    297 non-null    float64
```

```
 11  ca         297 non-null    int64  
 12  thal       297 non-null    int64  
 13  condition  297 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 32.6 KB
None
```

```
[18] dataset.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	condition
0	69	1	0	160	234	1	2	131	0	0.1	1	1	0	0
1	69	0	0	140	239	0	0	151	0	1.8	0	2	0	0
2	66	0	0	150	226	0	0	114	0	2.6	2	0	0	0
3	65	1	0	138	282	1	2	174	0	1.4	1	1	0	1
4	64	1	0	110	211	0	2	144	1	1.8	1	0	0	0

```
[19] X = dataset.drop(columns=['condition'])  # Features (all except 'condition')
     y = dataset['condition']  # Target variable
```

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

     # Display the dimensions of training and testing sets
     print("Training set:", X_train.shape)
     print("Testing set:", X_test.shape)
```

```
Training set: (237, 13)
```

```
[19] X = dataset.drop(columns=['condition'])  # Features (all except 'condition')
     y = dataset['condition']  # Target variable
```

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

     # Display the dimensions of training and testing sets
     print("Training set:", X_train.shape)
     print("Testing set:", X_test.shape)
```

```
Training set: (237, 13)
Testing set: (60, 13)
```

```
[21] # Train a Logistic Regression model
     model = LogisticRegression(max_iter=1000)
     model.fit(X_train, y_train)

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

     # Evaluate the model's accuracy
     accuracy = accuracy_score(y_test, y_pred)
     print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

```
Model Accuracy: 73.33%
```

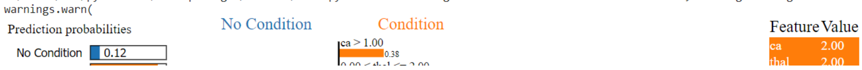
```
# Initialize the LIME Tabular Explainer
from lime.lime_tabular import LimeTabularExplainer
```

```
[21] print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

```
Model Accuracy: 73.33%
```

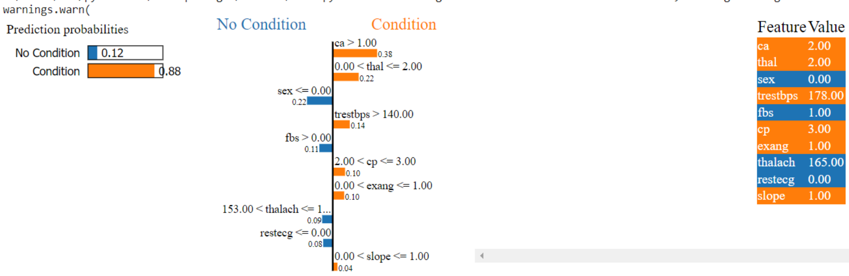
```
# Initialize the LIME Tabular Explainer
from lime.lime_tabular import LimeTabularExplainer

explainer = LimeTabularExplainer(
    X_train.values,
    feature_names=X.columns,
    class_names=['No Condition', 'Condition'],
    mode='classification'
)

# Select a specific instance from the test set to explain
instance_idx = 0  # Choose the first test instance for demonstration
instance = X_test.iloc[instance_idx].values.reshape(1, -1)

# Generate explanation for the selected instance
exp = explainer.explain_instance(instance[0], model.predict_proba, num_features=10)

# Display explanation in notebook
exp.show_in_notebook(show_all=False)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```




```
exp.show_in_notebook(show_all=False)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```



Google Collaboratory Link: -

https://colab.research.google.com/drive/1bHTJ08FC1ESjIC2TdB-Z-o3ScE0HUAfy#scrollTo=1DeON4C-AADN

Conclusion: -

Named Entity Recognition (NER) is a vital technique in Natural Language Processing (NLP) that identifies and classifies key entities in unstructured text, such as names, organizations, locations, dates, and more. By extracting these entities, NER enhances the understanding of text data, enabling better information retrieval, sentiment analysis, and summarization. It helps transform unstructured data into structured information, facilitating tasks like knowledge graph creation and improving search engine performance. Ultimately, NER is crucial for making sense of vast amounts of textual data and enabling more intelligent interactions with that data.