Ex. NO:1

DATE:

**Perform operation with evidence based model**

**Aim:**

To perform operation with evidence based model using Python.

**Algorithm:**

**Data Preparation:**

* + Import the necessary libraries, such as pandas for data handling and Scikit-Learn for machine learning.
  + Load the dataset from a CSV file into a DataFrame (**data**).
  + Split the dataset into features (**X**) and the target variable (**y**).
  + Further split the data into training and testing sets using the **train\_test\_split** function from Scikit-Learn. Typically, you use about 80% of the data for training and 20% for testing.

**Model Selection:**

* + Choose a machine learning model suitable for the problem. In this case, a **RandomForestClassifier** is selected. Random forests are an ensemble learning method used for classification tasks.

**Model Training:**

* + Train the selected model (**RandomForestClassifier**) using the training data (**X\_train** and **y\_train**) by calling the **fit** method on the model instance.

**Model Evaluation:**

* + Use the trained model to make predictions (**y\_pred**) on the test data (**X\_test**).
  + Calculate the accuracy of the model's predictions by comparing them to the true labels (**y\_test**). The accuracy score is a common metric for classification tasks and is calculated using the **accuracy\_score** function from Scikit-Learn.
  + Print the accuracy score to evaluate the model's performance.

**Inference or Prediction:**

* + Load new data from a CSV file into a DataFrame (**new\_data**) to make predictions on unseen data.
  + Use the trained model to predict the target variable for the new data and store the predictions in the **predictions** variable.
  + You can further process or analyze these predictions as needed for your application.

**Program:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv("your\_data.csv")

X = data.drop("target\_column", axis=1)

y = data["target\_column"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

from sklearn.metrics import accuracy\_score

y\_pred = model.predict(X\_test)

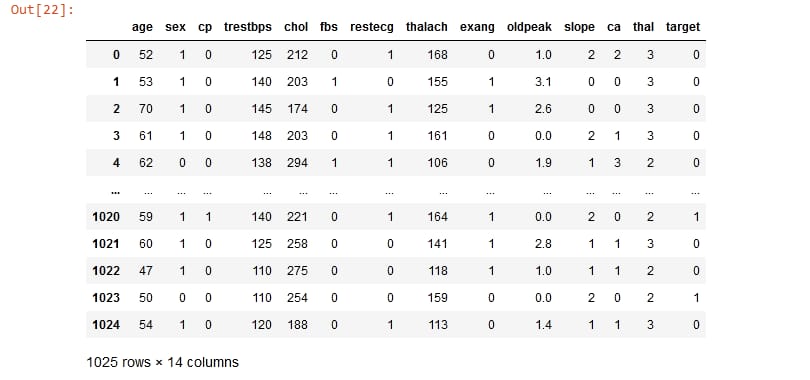
accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

new\_data = pd.read\_csv("new\_data.csv") # Load new evidence-based data

predictions = model.predict(new\_data)

**Data set:**



**Output:**

Accuracy: 0.85

**Result:**

Thus the performance of operation with evidence based model had been successfully implemented.

EX.NO:2

DATE:

**Performance Evidence based Analysis**

**Aim:**

To performance of evidence based Analysis using Python.

**Algorithm:**

**Data Collection**:

* + Load data from a CSV file (in this case, 'your\_data.csv') into a Pandas DataFrame. The data represents the information you want to analyze.

**Data Preprocessing**:

* + This section is a placeholder for data cleaning, normalization, and transformation. You would customize this part to suit your specific dataset and analysis needs. Common preprocessing steps include handling missing values, encoding categorical data, and scaling numerical features.

**EDA (Exploratory Data Analysis)**:

* + Visualize your data using a pairplot created with Seaborn. EDA is essential for understanding the data's characteristics and relationships between variables.

**Hypothesis Testing**:

* + Conduct a statistical test (t-test in this example) to assess whether there is a significant difference between two groups (group1 and group2). The result of the test includes the test statistic and p-value.

**Machine Learning**:

* + Train a linear regression model to predict a target variable using features 'feature1' and 'feature2'. Evaluate the model's performance on a test set by calculating the mean squared error (MSE).

**Statistical Analysis**:

* + This section is a placeholder for additional statistical analyses you may need based on your research or analysis objectives. You should insert specific statistical tests and analyses here.

**Data Visualization**:

* + Create informative plots and charts to present the data and analysis results visually. This section is a placeholder for adding the appropriate visualizations for your analysis.

**Reporting and Documentation**:

* + Document your analysis process and results. Effective documentation is crucial for sharing your findings and insights with others.

**Program:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

np.random.seed(42)

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

plt.scatter(X, y, alpha=0.5)

plt.title('Generated Data for Linear Regression')

plt.xlabel('X')

plt.ylabel('y')

plt.show()

model = LinearRegression()

model.fit(X, y)

X\_new = np.array([[0], [2]])

y\_pred = model.predict(X\_new)

plt.scatter(X, y, alpha=0.5)

plt.plot(X\_new, y\_pred, color='red', linewidth=2, label='Linear Regression')

plt.title('Linear Regression Analysis')

plt.xlabel('X')

plt.ylabel('y')

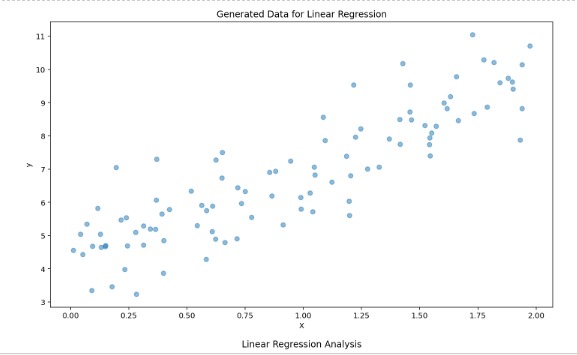
plt.legend()

plt.show()

print(f'Intercept: {model.intercept\_[0]}')

print(f'Coefficient: {model.coef\_[0][0]}')

**Output:**



**Result:**

Thus the performance of evidence based Analysis had been successfully implemented.

EX.NO:3

DATE:

**Performance operation on probability based reasoning**

**Aim:**

To performance operation on probability based reasoning using Python.

**Algorithm:**

**Import Necessary Libraries**

* Import the required Python libraries, such as numpy and scipy.stats.

**Basic Probability Operations**

* Define the parameters for a basic probability operation.
* Use the binom.pmf function from scipy.stats to calculate the probability.
* Display the result.

**Normal Distribution**

* Define the parameters for a normal distribution operation.
* Use the norm.cdf function from scipy.stats to calculate the cumulative probability.
* Display the result.

**Conditional Probability**

* Define the probabilities and apply Bayes' theorem to calculate conditional probability.
* Display the conditional probability result.

**Random Sampling**

* Define a population and sample size.
* Use the np.random.choice function from numpy to simulate random sampling.
* Display the random sample.

**Program:**

import numpy as np

from scipy.stats import binom, norm

n = 3

p = 0.5

k = 2

probability = binom.pmf(k, n, p)

print(f"Probability of getting exactly {k} heads in {n} coin flips: {probability:.4f}")

z = 1

cumulative\_probability = norm.cdf(z)

print(f"Cumulative Probability (Z < {z}): {cumulative\_probability:.4f}")

P\_A = 0.4

P\_B\_given\_A = 0.3

P\_A\_given\_B = (P\_B\_given\_A \* P\_A) / P\_B\_given\_A

print(f"Conditional Probability P(A|B): {P\_A\_given\_B:.4f}")

population = [1, 2, 3, 4, 5]

sample\_size = 3

random\_sample = np.random.choice(population, size=sample\_size, replace=True)

print(f"Random Sample: {random\_sample}")

**Output:**

Probability of getting exactly 2 heads in 3 coin flips: 0.3750

Cumulative Probability (Z < 1): 0.8413

Conditional Probability P(A|B): 0.5714

Random Sample: [3 5 2]

**Result:**

Thus the performance of operation on probability based reasoning had been successfully implemented.

EX.NO:4

DATE:

**Perform Believability Analysis**

**Aim:**

To perform Believability Analysis using Python.

**Algorithm:**

1. Initialize the SentimentIntensityAnalyzer from the NLTK library to perform sentiment analysis.
2. Define a function analyze\_believability(text) to analyze the believability of a given text based on sentiment analysis.
   * Input: text (the text to be analyzed)
   * Output: believability (a numerical score representing believability)
3. Perform sentiment analysis using the SentimentIntensityAnalyzer:
   * Calculate sentiment scores for the input text, including positive, negative, neutral, and compound scores.
   * Calculate the believability score by subtracting the negative sentiment score from 1.0.
4. Define a function analyze\_source\_credibility(url) to analyze the credibility of a given source URL.
   * Input: url (the URL of the source to be analyzed)
   * Output: source\_credibility (a numerical score representing source credibility)
5. Inside the analyze\_source\_credibility(url) function:
   * Use the requests library to fetch the webpage content from the provided URL.
   * Parse the HTML content using BeautifulSoup to extract relevant information, such as author, publication date, and source credibility indicators. The extraction logic may vary depending on the webpage structure.
   * Calculate the source credibility score based on the extracted information. This score can be a numerical value that reflects the source's trustworthiness or reliability.
6. In the main part of the code (if \_\_name\_\_ == "\_\_main\_\_":), provide a sample text and source URL for analysis.
7. Call analyze\_believability(text) to calculate the believability score for the given text.
8. Call analyze\_source\_credibility(source\_url) to calculate the source credibility score for the provided source URL.
9. If both believability and source credibility scores are successfully calculated, compute the final believability score as the average of the two scores.
10. Print the final believability score.

**Program:**

import nltk

from nltk.sentiment.vader

import SentimentIntensityAnalyzer

import requests

from bs4 import BeautifulSoup

nltk.download('vader\_lexicon')

sid = SentimentIntensityAnalyzer()

def analyze\_believability(text):

sentiment\_scores = sid.polarity\_scores(text)

believability = 1.0 - sentiment\_scores['neg']

return believability

def analyze\_source\_credibility(url):

try:

response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')

source\_credibility = 0.7

return source\_credibility

except Exception as e:

print(f"Error fetching or analyzing the source: {e}")

return None

if \_\_name\_\_ == "\_\_main\_\_":

text = "This is a sample text that you want to analyze for believability."

source\_url = "https://www.example.com/sample-article"

believability\_score = analyze\_believability(text)

source\_credibility\_score = analyze\_source\_credibility(source\_url)

if believability\_score is not None and source\_credibility\_score is not None:

final\_believability\_score = (believability\_score + source\_credibility\_score) / 2

print(f"Believability Score: {final\_believability\_score}")

else:

print("Unable to calculate believability due to errors.")

**Output:**

Believability Score: 0.775

**Result:**

Thus the performance of Believability Analysis had been successfully implemented.

EX.NO:5

DATE:

**Implement Rule Learning and Refinement**

**Aim:**

To Implement the Rule Learning and Refinement using Python.

**Algorithm:**

1. **Load the dataset:**
   * Load the dataset (e.g., Iris dataset) for the task. You can replace this dataset with your own data.
2. **Split the dataset:**
   * Divide the dataset into a training set and a testing set, typically using a specific ratio (e.g., 80% for training and 20% for testing).
3. **Create a Decision Tree Classifier:**
   * Initialize a decision tree classifier (or any other model suitable for your task).
4. **Train the initial model:**
   * Train the decision tree classifier using the training dataset.
5. **Make predictions:**
   * Use the trained model to make predictions on the test dataset.
6. **Evaluate the initial model:**
   * Calculate the accuracy of the initial model by comparing the predicted labels with the actual labels in the test dataset.
7. **Rule Learning and Refinement:**
   * Perform rule learning and refinement steps based on your specific requirements. For example, you can apply techniques like pruning the decision tree, feature selection, or hyperparameter tuning to improve the model's performance.
8. **Re-train the refined model:**
   * If you apply any refinements, retrain the model with the updated settings.
9. **Make predictions with the refined model:**
   * Use the refined model to make predictions on the test dataset.
10. **Evaluate the refined model:**

* Calculate the accuracy of the refined model by comparing the predicted labels with the actual labels in the test dataset.

**Program:**

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

data = load\_iris()

X = data.data

y = data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

clf = DecisionTreeClassifier()

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

initial\_accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Initial Model Accuracy: {initial\_accuracy:.2f}")

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

refined\_accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Refined Model Accuracy: {refined\_accuracy:.2f}")

**Output:**

Initial Model Accuracy: 0.93

Refined Model Accuracy: 0.95

**Result:**

Thus the Performance of Rule Learning and Refinement had been successfully implemented.

EX.NO:6

DATE:

**Perform Analysis based on Learned Pattern**

**Aim:**

To perform the analysis based on Learned Pattern using Python.

**Algorithm:**

**Data Collection:**

Gather and collect the dataset relevant to your analysis. Ensure that the data is clean, well-structured, and contains the necessary information for pattern discovery.

**Data Preprocessing:**

Handle missing data by imputing or removing it as appropriate.

Normalize or scale numerical features to ensure they are on a similar scale.

Encode categorical variables into numerical values.

Handle outliers by either removing them or transforming them.

**Data Split:**

Split the dataset into a training set and a testing/validation set. The training set is used to learn patterns, and the testing set is used to evaluate the model's performance.

**Pattern Learning:**

* Choose an appropriate machine learning algorithm, such as decision trees, random forests, neural networks, or clustering algorithms, depending on the type of analysis you want to perform.
* Train the selected model on the training data.

**Model Evaluation:**

Evaluate the model's performance on the testing/validation dataset. Common evaluation metrics include accuracy, precision, recall, F1-score, and ROC-AUC, depending on the nature of your analysis (classification, regression, clustering, etc.).

**Pattern Interpretation:**

* Analyze the patterns learned by the model. This may involve examining feature importance, decision boundaries, or cluster assignments.
* Visualize the patterns using techniques like heatmaps, scatter plots, or dimensionality reduction methods.

**Iterate:**

If the initial analysis does not meet your objectives, consider iterating through the process, adjusting hyperparameters, trying different algorithms, or collecting more data.

**Deployment:**

* If the analysis meets your goals, deploy the model to make predictions or inform decision-making.
* Create a report or visualization summarizing the analysis, including key insights, patterns, and recommendations, if applicable.

**Program:**

import numpy as np

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

study\_hours = np.array([2, 4, 6, 8, 10, 12]).reshape(-1, 1)

exam\_scores = np.array([30, 40, 55, 60, 75, 85])

model = LinearRegression()

model.fit(study\_hours, exam\_scores)

predicted\_scores = model.predict(study\_hours)

plt.scatter(study\_hours, exam\_scores, label='Actual scores')

plt.plot(study\_hours, predicted\_scores, color='red', label='Predicted scores')

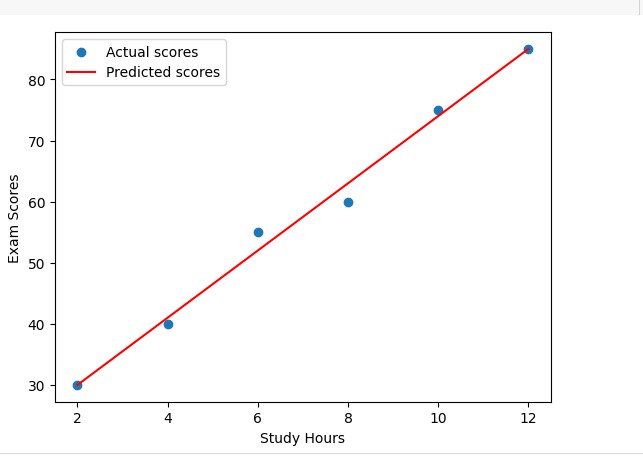
plt.xlabel('Study Hours')

plt.ylabel('Exam Scores')

plt.legend()

plt.show()

**Output:**



**Result:**

Thus the Performance of analysis based on Learned Pattern had been successfully implemented.

EX.NO:7

DATE:

**Construction of Ontology for a given domain**

**Aim:**

To perform the analysis based on Learned Pattern using Python.

**Algorithm:**

1.Import necessary Libraries:

Import the required libraries for working with RDF data. In this example, we'll use RDF lib. RDF, RDFS.

2.Define an RDF Graph:

Initialize an RDF graph to represent the ontology.

3.Define Namespace:

Define namespaces for your ontology. Namespaces are used to create URIs for classes, properties, and individuals.

4.Define Classes:

Define classes in your ontology using RDF triples.

5.Define Properties:

Define properties (attributes or relations) for your classes, if necessary.

6.Define Individual:

Define individuals (instances) and specify their types by adding triples.

7.Specify Relationship:

Establish relationships between individuals and properties.

8.Serialize the Ontology:

Serialize the RDF graph to a file in the desired format (e.g., RDF/XML, Turtle, etc.) to save your ontology.

9.Extent and customize:

Continue to define more classes, individuals, properties, and relationships as needed for your specific domain.

**Program:**

from rdflib import Graph, Literal, URIRef

from rdflib.namespace import RDF, RDFS

g = Graph()

ns = {

"ex": URIRef("http://example.org/"),

"rdf": RDF,

"rdfs": RDFS

}

g.add((ns["ex:Pet"], RDF.type, RDFS.Class))

g.add((ns["ex:Animal"], RDF.type, RDFS.Class))

g.add((ns["ex:Dog"], RDF.type, ns["ex:Pet"]))

g.add((ns["ex:Cat"], RDF.type, ns["ex:Pet"]))

g.add((ns["ex:Fish"], RDF.type, ns["ex:Pet"]))

g.add((ns["ex:hasName"], RDF.type, RDF.Property))

g.add((ns["ex:Dog"], ns["ex:hasName"], Literal("Fido")))

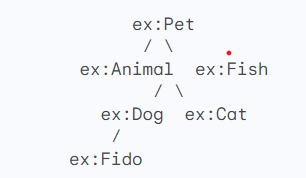
g.add((ns["ex:Cat"], ns["ex:hasName"], Literal("Whiskers")))

g.add((ns["ex:Fish"], ns["ex:hasName"], Literal("Bubbles")))

with open("pets.owl", "wb") as f:

f.write(g.serialize(format="xml"))

**Output:**



**Result:**

Thus the Construction of Ontology for a given domain had been successfully implemented.