Experiment 1: Implementation of different operation on a dataset by Bayesian Network in Python.

Objective of the Experiment

The goal of this experiment was to **model and analyze the Monty Hall problem** using a **Bayesian Network** to understand **conditional probability reasoning** — particularly how evidence changes beliefs about uncertain events.

🧠 Background: The Monty Hall Problem

It's a famous probability puzzle based on a game show scenario:

1. There are **3 doors** — behind one is a **prize**, behind the other two are **goats**.
2. You pick one door (say, Door 1).
3. The **host**, who knows where the prize is, opens one of the remaining doors that **does not have the prize**.
4. You are then given a choice to **stick** with your door or **switch** to the remaining unopened door.

Question: Should you switch or stay?

Intuitively, people often think it's 50–50, but mathematically, if you **switch**, you win **2/3 of the time**.

Your Bayesian Network experiment demonstrates **why** this happens.

🧩 1. Model Construction

📘 a) Defining the Nodes

You defined three key variables:

1. **Prize** — which door hides the prize (`1, 2, or 3`)
2. **Choice** — which door the player initially picks (`1, 2, or 3`)
3. **Host** — which door the host opens (depends on both Prize and Choice)

📘 b) Defining the Structure

```
model = DiscreteBayesianNetwork([('Prize', 'Host'), ('Choice', 'Host')])
```

This structure means:

- The **Host** depends on **Prize** and **Choice**.
- **Prize** and **Choice** are independent of each other.

Diagrammatically:

```
Prize ——▶ Host ◀—— Choice
```

🧱 2. Defining Probability Distributions

a) Prize CPD

Each door is equally likely to hide the prize:

P(Prize=1)=P(Prize=2)=P(Prize=3)=13P(Prize=1) = P(Prize=2) = P(Prize=3) = \frac{1}{3}P(Prize=1)=P(Prize=2)=P(Prize=3)=31

b) Choice CPD

The contestant chooses randomly:

P(Choice=1)=P(Choice=2)=P(Choice=3)=13P(Choice=1) = P(Choice=2) = P(Choice=3) = \frac{1}{3}P(Choice=1)=P(Choice=2)=P(Choice=3)=31

c) Host CPD

The host's behavior depends on the prize and the player's choice:

- The host never opens the door with the prize.
- The host never opens the contestant's chosen door.
- If there are two valid doors, he picks randomly between them.

This was built programmatically using:

```
host_values[v-1][3*i + j] = 1 / len(valid)
```

This logic ensures the correct probability for each situation.

🧩 3. Model Validation

You combined all the CPDs:

```
model.add_cpds(cpd_prize, cpd_choice, cpd_host)
model.check_model()
```

✔️ The model check ensures that:

- Every variable has a valid CPD.
- Probabilities are normalized (sum to 1).

🔍 4. Performing Inference

You used **Variable Elimination** to perform reasoning:

```
inference = VariableElimination(model)
query_result = inference.query(variables=['Prize'],
                               evidence={'Choice': 1, 'Host': 3})
```

This calculates:

$P(Prize \mid Choice=1, Host=3)$

Meaning:
"Given that I picked door 1, and the host opened door 3, what's the probability that the prize is behind each door?"

📊 5. Output Interpretation

The result will typically look like:

Prize	Probability
1	0.3333
2	0.6667
3	0.0000

Interpretation:

- The probability that your **initial choice (door 1)** has the prize is **1/3**.
- The probability that the **other unopened door (door 2)** has the prize is **2/3**.
- So, **switching doors** gives a **better chance of winning (2/3)**.

🧾 6. Conclusion

This Bayesian Network experiment **confirms** the counterintuitive result of the Monty Hall problem:

> If you **switch** after the host opens a door, your chance of winning **doubles** from **1/3 to 2/3**.

Experiment 2:

Experiment Explanation – Chatbot using NLP

In this experiment, we created a **rule-based chatbot** using Natural Language Processing (NLP). The purpose of the experiment was to understand how computers can process human language and respond intelligently without using machine learning models.

The chatbot works by taking text input from the user, analyzing it, and then returning the most appropriate response based on stored information in a text file. It does not “learn” like AI models; instead, it uses predefined knowledge and mathematical text-matching techniques to find the best reply.

The core idea of this experiment is to convert human language into a format a computer can understand. The text data is first broken into sentences and words. Then, all text is standardized by converting words to their base form, so the computer can easily compare meanings. When the user types a message, the system compares it to all stored sentences and identifies which sentence is most similar in meaning. That sentence is then given back as the bot’s reply.

The experiment also handles basic conversation features like greeting recognition (e.g., “hello”, “hi”) and polite endings like “thank you” and “bye”. This makes the chatbot more natural and user-friendly. If the bot cannot find a relevant response, it replies with a default message indicating that it does not understand.

🎯 Outcome of the Experiment

- Demonstrated how NLP techniques are used to process text.
- Successfully built a chatbot that interacts with users in natural language.
- Understood concepts like tokenization, lemmatization, text similarity, and conversational flow.
- Practiced converting raw text into a structured form and comparing similarity.

📌 Conclusion

The experiment shows that even without advanced AI, we can build a functional chatbot using traditional NLP techniques. It highlights how computers interpret human text, find context, and

generate meaningful responses. This forms the foundation for more advanced conversational AI systems.

Experiment no 3

This experiment demonstrates how to perform data visualization in Python using **Matplotlib** and **Seaborn**. The goal is to load a dataset, explore it, and represent key information through graphical plots. Visualization helps identify patterns, trends, and data distribution more clearly than raw numbers.

We begin by importing the dataset using **Pandas** and inspecting its structure. Then, unnecessary columns are removed to focus only on useful features. After understanding the dataset, various plots such as bar graphs, scatter plots, and histograms are created using Matplotlib and Seaborn. These visualizations help analyze relationships between variables and observe how data is spread across values.

Through this experiment, we learn how to convert data into meaningful graphics, enabling better interpretation and decision-making. It strengthens the use of Python's visualization libraries for Exploratory Data Analysis (EDA).

Exp 4:

1. Aim

The aim of this experiment is **to understand how to extract, organize, modify, and clean data efficiently**, and to apply **fuzzy logic** to handle uncertain or imprecise data during the cleaning process.

🔹 2. Introduction

In real-world datasets, data is often **large, incomplete, and uncertain**.
Before performing any analysis or building a model, we must properly **slice, index, manipulate, and clean** the data.

At the same time, **fuzzy logic** helps handle vague or approximate information — instead of saying something is "true" or "false," it allows reasoning in **degrees of truth** (values between 0 and 1).

Combining these two — data handling with fuzzy logic — gives us a more **realistic and intelligent data-processing approach.**

🔹 3. Theory

A. Data Slicing

- **Meaning:** Selecting specific portions of data (rows and columns) from a dataset.
- **Purpose:** Helps focus only on the relevant subset of data for analysis.
- **Example:** Extracting records of students aged 18–25 from a full database.

B. Data Indexing

- **Meaning:** Accessing data efficiently using labels, positions, or conditions.
- **Purpose:** Makes it easy to retrieve, modify, or update specific data points.
- **Example:** Selecting all rows where salary > 50000.

C. Data Manipulation

- **Meaning:** Modifying or transforming data to make it more meaningful or ready for analysis.
- **Purpose:** Allows creation of new features, conversions, or aggregations.
- **Example:** Calculating total marks, normalizing data, or converting text to numerical categories.

D. Data Cleaning

- **Meaning:** Detecting and correcting errors, inconsistencies, or missing values in the dataset.
- **Purpose:** Ensures that data is **accurate, complete, and reliable**.
- **Common tasks:**
    - Removing duplicates
    - Handling missing values
    - Fixing data types
    - Removing outliers

E. Fuzzy Logic

- **Concept:** Introduced by *Lotfi Zadeh*, fuzzy logic is a form of reasoning that deals with **imprecision and uncertainty**.
- Unlike classical logic (0 or 1), fuzzy logic allows **partial truth** — any value between 0 and 1.
- **Application in Data Cleaning:**
    - Real data often contains vague terms like “high”, “medium”, or “low.”
    - Fuzzy logic assigns **degrees of membership** to such data instead of fixed labels.
    - Example: For a salary of ₹60,000 —
        - Low = 0.2
        - Medium = 0.7
        - High = 0.3

This helps handle uncertain or overlapping categories intelligently.

Exp 5:

1. Aim

To understand and implement **Python libraries** and **data structures** in developing a **Recurrent Neural Network (RNN)** or **Long Short-Term Memory (LSTM)** model for sequence-based data processing and prediction tasks.

🔹 2. Introduction

This experiment focuses on how **Python libraries** (like NumPy, Pandas, TensorFlow, and Keras) and **data structures** (like lists, arrays, and tensors) are used to build and train **RNN/LSTM models** for analyzing **sequential data** such as text, time series, or speech.

In deep learning, RNN and LSTM are powerful models for understanding patterns in **data that changes over time**, unlike traditional neural networks that treat each input as independent.

🔹 3. Theory

A. Python Libraries Used

1. **NumPy**
    - Provides support for numerical computations and multidimensional arrays.
    - Used to store and manipulate input data efficiently.
    - Example: converting datasets into NumPy arrays before feeding them into neural networks.
2. **Pandas**
    - Used for data cleaning, loading, and preprocessing.
    - Helps organize data into DataFrames for easy handling.
3. **Matplotlib / Seaborn**
    - Used for data visualization such as plotting loss curves and accuracy graphs.

4. **TensorFlow / Keras**
    - Libraries for building, training, and testing deep learning models.
    - Provide pre-built layers like `LSTM`, `RNN`, and `Dense` for sequence learning.

B. Data Structures Used

1. **Lists**
    - To store sequences of data like input sentences or time steps.
    - Example: `[1, 2, 3, 4, 5]` representing a time sequence.
2. **Arrays (NumPy arrays)**
    - More efficient and faster than lists for mathematical computations.
3. **Tensors**
    - Multidimensional arrays used internally in deep learning frameworks (TensorFlow, PyTorch).
    - Example: A 3D tensor for time series → shape `[samples, time_steps, features]`.

C. Recurrent Neural Network (RNN)

- **Definition:**
  An RNN is a type of neural network designed for processing **sequential or time-dependent data**.
  Unlike traditional neural networks, RNNs have **feedback connections**, allowing information to persist from one step to the next.

Working Principle:
Each output from a neuron is fed back as input to the next step, creating a memory effect.

```
Input(t) → Hidden(t) → Output(t)
              ↑
              |
          Hidden(t-1)
```

-
- **Applications:**
    - Stock price prediction
    - Sentiment analysis
    - Speech recognition
    - Time-series forecasting
- **Limitation:**
    - RNNs struggle with **long-term dependencies** due to the *vanishing gradient problem* (old information fades over time).

D. Long Short-Term Memory (LSTM)

- **Definition:**
  LSTM is a special kind of RNN capable of learning **long-term dependencies**.
  It solves the vanishing gradient problem using **gates** that control the flow of information.
- **Main Components:**
    - **Forget Gate:** Decides which information should be discarded.
    - **Input Gate:** Decides which new information should be added to memory.
    - **Output Gate:** Controls what part of the memory is output to the next layer.
- **Advantage:**
    - Retains information over long sequences.

    - More stable during training.
    - Performs better on sequential data than basic RNN.

- **Applications:**
    - Text generation
    - Music composition
    - Weather forecasting
    - Language translation

E. Integration of Data Structures and Libraries with LSTM

1. **Data Loading (Pandas):**
   The dataset is loaded into a DataFrame for preprocessing.

2. **Conversion to Arrays (NumPy):**
   Data is transformed into numeric arrays for neural network compatibility.

3. **Sequence Creation:**
   Data is structured into time steps using lists or arrays to form input-output pairs.

4. **Model Building (Keras):**
   Using `Sequential()`, `LSTM()`, and `Dense()` layers to define the model structure.

5. **Training & Evaluation:**
   The model is compiled, trained, and evaluated using TensorFlow functions.

Exp6:

Aim:

To analyze and visualize data using Python-based EDA tools and apply **Artificial Neural Networks (ANN)** to identify and learn patterns from the processed data.

Theory:

1. Exploratory Data Analysis (EDA):
EDA helps us understand the dataset before modeling.
It includes checking missing values, data distribution, correlations, and trends.
Python libraries like **Pandas**, **NumPy**, **Matplotlib**, and **Seaborn** are used for this purpose.

2. Data Visualization:
It represents data graphically through charts such as histograms, boxplots, scatter plots, and heatmaps.
Visualization makes it easier to detect patterns, outliers, and relationships between variables.

3. Artificial Neural Network (ANN):
ANN is a computational model inspired by the human brain.
It consists of **input, hidden, and output layers** that learn complex relationships in data.
After performing EDA and cleaning, ANN is trained to make predictions or classifications based on the analyzed data.

Conclusion:

This experiment demonstrates how **EDA and visualization** help in understanding and preparing data, and how an **ANN** can then be applied to learn from it — combining both **data analysis** and **intelligent prediction** for better decision-making.

Exp7:

Aim:

To understand the use of **Arrays**, **Pandas**, and **CSV files** in real-world applications such as **handwritten digit recognition**, **image classification**, or **image caption generation** using Python.

Theory:

1. Arrays:

- Arrays are used to store multiple values in a single variable.

- In Python, **NumPy arrays** are efficient for storing and processing numerical or image data.
- For example, an image is represented as a 2D or 3D array of pixel values.

2. Pandas:

- Pandas is used for **data handling and analysis**.
- It provides **DataFrames** to store structured data and allows easy operations like filtering, sorting, and merging.
- Useful for managing datasets used in training machine learning models.

3. CSV Files:

- CSV (Comma-Separated Values) files store tabular data and are often used for importing and exporting datasets.
- Pandas can easily read and write CSV files using functions like `read_csv()` and `to_csv()`.

Application Example – Handwritten Digit Recognition System:

- The **MNIST dataset** is commonly used, containing thousands of handwritten digit images (0–9).
- Each image is stored as a **NumPy array**.
- The data is loaded and analyzed using **Pandas**, and stored or retrieved through **CSV files**.
- A machine learning or neural network model is trained to recognize digits from these array representations.

Conclusion:

This experiment demonstrates how **arrays**, **Pandas**, and **CSV files** form the foundation of **data storage, processing, and manipulation** in real-world AI and ML applications like **digit recognition**, **image classification**, and **caption generation**.

Exp8:

Experiment Title:

Implementation of Predictive Model using Python, Ada-Boosting, Random Forests and Evaluation of Classification Algorithms

---

Aim:

To build and evaluate a **predictive classification model** using Python, applying **ensemble learning techniques** like **AdaBoost** and **Random Forest**, and compare their performance using suitable evaluation metrics.

Theory:

1. Predictive Modeling:
Predictive modeling involves using historical data to **predict future outcomes**.
It's a core part of machine learning where models learn patterns from labeled datasets and make predictions on new data.

2. Random Forest:

- It is an **ensemble learning algorithm** that combines multiple **decision trees**.
- Each tree gives a prediction, and the final output is based on **majority voting**.
- It reduces overfitting and improves accuracy.
- Works well for both classification and regression problems.

3. AdaBoost (Adaptive Boosting):

- AdaBoost is a **boosting technique** that combines several **weak classifiers** (usually decision trees) into a **strong classifier**.

- It assigns **higher weights to misclassified samples**, forcing the model to focus on difficult cases.

- It improves model accuracy and robustness.

4. Evaluation of Classification Algorithms:
After training, models are evaluated using metrics such as:

- **Accuracy:** Overall correctness of predictions.

- **Precision & Recall:** Measure of how well the model identifies true positives.

- **F1-Score:** Harmonic mean of precision and recall.

- **Confusion Matrix:** Summarizes correct and incorrect predictions.

Working Principle:

1. Load and preprocess the dataset using **Pandas and NumPy**.

2. Split the data into training and testing sets.

3. Train models using **Random Forest** and **AdaBoost** algorithms.

4. Make predictions and evaluate using accuracy, precision, recall, and confusion matrix.

5. Compare model performances to find the best classifier.

Conclusion:

This experiment demonstrates how **ensemble learning methods** like **AdaBoost** and **Random Forest** enhance the accuracy and reliability of predictive models.
It also highlights how **evaluation metrics** help in comparing the effectiveness of different classification algorithms in real-world data analytics.