Lab-06

Theory:

Linear Regression is a statistical method that models the relationship between a dependent variable (often called the target or response variable) and one or more independent variables (called features). In simple linear regression, we use only one feature to predict the target variable. The relationship is modeled by fitting a line, called the regression line, that best describes the data's trend.

The equation for a simple linear regression line is: y=mx+c where:

- y is the predicted output,
- mmm is the slope of the line (determined by how much y changes as x changes),
- x is the input feature (independent variable),
- c is the y-intercept of the line.

The linear regression algorithm minimizes the difference between actual and predicted values by adjusting the slope mmm and the intercept ccc using techniques like Ordinary Least Squares (OLS).

Lab-05

Theory

In data analysis, descriptive statistics provide a way to summarize and understand data characteristics. Common metrics include:

1. Mean: The average value of a dataset, calculated by summing all values and dividing by the number of data points.

   $$Mean = N\sum X$$

   where ∑X is the sum of all values and N is the number of values.

2. Median: The middle value of a sorted dataset, which divides the data into two equal halves. If the number of values is even, the median is the average of the two middle values.

3. Mode: The most frequently occurring value in a dataset. A dataset may have multiple modes or none if all values are unique.

4. Variance: A measure of data spread around the mean. Calculated by averaging the squared differences of each value from the mean, variance shows how much individual data points deviate from the average.

$$Variance = N\sum(X - Mean)2$$

where X represents each individual value, and N is the number of values.

5. Standard Deviation: The square root of variance, providing a measure of data dispersion in the same units as the original dataset.

$$Standard\ Deviation = \sqrt{Variance}$$

Python provides libraries like pandas and statistics for handling data and computing these metrics. In this experiment, we'll use both to load datasets, compute these descriptive statistics, and understand the dataset's central tendency and variability.

Lab-04

Theory:

The **Travelling Salesman Problem (TSP)** is a well-known combinatorial optimization problem. It involves a salesman who needs to visit a set of cities exactly once and return to the starting city while minimizing the total travel distance or cost. The TSP is an NP-hard problem, meaning that no efficient algorithm exists to solve it for large inputs in polynomial time.

Brute-force Approach:

1. **Generate Permutations**: List all possible orders (permutations) in which the cities can be visited.
2. **Calculate Distance**: For each permutation, calculate the total travel distance.
3. **Find Minimum Distance**: Out of all possible routes, choose the one with the minimum travel distance.

In this lab, we use the brute-force method, which has a time complexity of O(n!), where n is the number of cities. While inefficient for large numbers of cities, this approach demonstrates the basics of solving the TSP.

Lab-03:

Theory:

Depth-First Search (DFS) is a fundamental graph traversal algorithm that explores as far down a branch as possible before backtracking. Unlike Breadth-First Search (BFS), which explores neighbors at the current depth level before moving deeper, DFS goes deep along a path until it reaches a node with no unvisited neighbors.

DFS can be implemented either recursively or using an explicit stack for iteration. It is commonly used in applications such as pathfinding, detecting cycles in graphs, and solving puzzles that require backtracking.

Working of DFS:

1. **Initialization**: Start from a selected source node and mark it as visited.
2. **Traversal Steps**:
    - Visit a node, mark it as visited, and push it onto the stack.
    - Move to an adjacent unvisited node, marking it as visited and pushing it onto the stack.
    - If a node has no unvisited neighbors, backtrack by popping the stack and return to the previous node.
3. **Termination**: The algorithm completes once all reachable nodes from the source node are visited.

The **time complexity** of DFS is O(b^d), where V is the number of vertices and E is the number of edges. This complexity arises because DFS processes each vertex and edge once.

Lab-02:

T**heory:**

Breadth-First Search (BFS) is a fundamental algorithm used for traversing or searching data structures like graphs and tree structures. BFS explores nodes level-by-level, starting from a selected source node, and it moves outwards layer by layer. Unlike Depth-First Search (DFS), which explores as deep as possible along a branch before backtracking, BFS systematically explores all neighbors at the present depth before moving to nodes at the next level.

Working of BFS:

1. **Initialization**: Start by selecting a source node and marking it as visited to avoid revisiting. Use a queue to manage the nodes to explore next, as BFS needs to follow a First In, First Out (FIFO) approach.
2. **Traversal Steps**:
    - **Step 1**: Enqueue the starting node.
    - **Step 2**: While the queue is not empty:
        - Dequeue the front node (i.e., the current node).
        - For each neighbor of this node:
            - If it has not been visited, mark it as visited and enqueue it.
        - Repeat until the queue is empty.
3. **Termination**: The algorithm finishes when all reachable nodes from the starting node are visited and the queue becomes empty.

he **time complexity** of BFS is O(V+E), where V is the number of vertices and E is the number of edges. This is because each vertex and edge in the graph is processed once.

Lab-01:

Theory:

The Tower of Hanoi is a classic recursive problem that involves moving a set of disks from one peg to another, with the help of an intermediate peg. The rules are:

1. Only one disk can be moved at a time.
2. A larger disk cannot be placed on top of a smaller disk.
3. The goal is to move all disks from the source peg to the destination peg using the helper peg.

The problem can be solved using recursion. For n disks, the steps are:

- Move the top n-1 disks from the source peg to the helper peg.
- Move the nth disk from the source peg to the destination peg.
- Move the n-1 disks from the helper peg to the destination peg.

The minimum number of moves required to solve the Tower of Hanoi problem is 2^n−1, where n is the number of disks.

Lab-Tic-tac-Toe:

Theory:

Tic-Tac-Toe is a simple, two-player game where players take turns marking spaces on a 3x3 grid, aiming to align three of their marks in a row, column, or diagonal. The game can end in a win (if a player forms a straight line) or a draw (if the board fills with no winner).

This implementation uses a list to represent the 3x3 board, with functions to handle key game operations:

1. **Display**: Shows the current state of the board after each move.
2. **Win Check**: Evaluates if a player has won by checking all possible winning combinations (rows, columns, diagonals).
3. **Draw Check**: Determines if the board is full without a winner, resulting in a draw.
4. **Input Validation**: Ensures players enter valid moves and handles errors if they select an occupied cell.

Through simple loops and conditionals, this game demonstrates core programming concepts like control flow, data handling, and user interaction, making it an excellent beginner project.