**Experiment No -1**

**Aim :- Solve the Tic-Tac-Toe problem using the Depth First Search technique.**

**Reduced code:-**

def print\_board(board):

    for row in board:

        print(" | ".join(row))

        print("-" \* 9)

def is\_winner(board, player):

    return any(all(cell == player for cell in row) for row in board) or \

           any(all(board[r][c] == player for r in range(3)) for c in range(3)) or \

           all(board[i][i] == player for i in range(3)) or \

           all(board[i][2 - i] == player for i in range(3))

def is\_draw(board):

    return all(cell != " " for row in board for cell in row)

def is\_valid\_move(board, row, col):

    return 0 <= row < 3 and 0 <= col < 3 and board[row][col] == " "

def dfs\_ai\_move(board, player):

    stack = [board]

    while stack:

        current = stack.pop()

        for r in range(3):

            for c in range(3):

                if current[r][c] == " ":

                    new\_board = [row[:] for row in current]

                    new\_board[r][c] = player

                    if is\_winner(new\_board, player):

                        return r, c

                    stack.append(new\_board)

    return None

def get\_player\_input():

    while True:

        try:

            user\_input = input("Enter row and column (e.g., 0 2): ")

            row, col = map(int, user\_input.strip().split())

            if 0 <= row <= 2 and 0 <= col <= 2:

                return row, col

            else:

                print("Values must be between 0 and 2.")

        except ValueError:

            print("Invalid input. Enter two numbers separated by space.")

def play\_game():

    board = [[" "] \* 3 for \_ in range(3)]

    players = ["X", "O"]

    current = 0  # 0 for human, 1 for AI

    while True:

        print\_board(board)

        if current == 0:

            row, col = get\_player\_input()

        else:

            print("AI (O) is thinking...")

            move = dfs\_ai\_move(board, players[1])

            row, col = move if move else (0, 0)

            print(f"AI chose position: ({row}, {col})")

        if is\_valid\_move(board, row, col):

            board[row][col] = players[current]

            if is\_winner(board, players[current]):

                print\_board(board)

                print(f"Player {players[current]} wins!")

                break

            if is\_draw(board):

                print\_board(board)

                print("It's a draw!")

                break

            current = 1 - current

        else:

            print("Invalid move. Try again.")

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

    play\_game()

**Experiment No – 2**

**Aim :- Show that the 8-puzzle states are divided into two disjoint sets, such that any state is reachable from any other state in the same set, while no state is reachable from any state in the other set.**

**Reduced code:-**

def get\_inv\_count(arr):

    arr = [x for x in arr if x != -1]

    return sum(1 for i in range(len(arr)) for j in range(i+1, len(arr)) if arr[i] > arr[j])

def is\_solvable(puzzle):

    inv = get\_inv\_count([num for row in puzzle for num in row])

    print("Inversions:", inv)

    return inv % 2 == 0

test1 = [[8, 1, 2], [-1, 4, 3], [7, 6, 5]]

test2 = [[5, 2, 8], [4, 1, 7], [-1, 3, 6]]

print("Solvable" if is\_solvable(test2) else "Not Solvable")

**Experiment No – 3**

**Aim :- To examine different types of machine learning approaches (Supervised, Unsupervised, Semi-supervised, and Reinforcement Learning) by setting up a basic classification problem and exploring how each type applies differently**

**Reduced code:-**

import numpy as np, random

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.cluster import KMeans

from sklearn.semi\_supervised import SelfTrainingClassifier

from sklearn.tree import DecisionTreeClassifier

X = np.array([[150, 7], [160, 7], [170, 8], [180, 8], [300, 15], [320, 16], [310, 15], [330, 16]])

y = np.array([1]\*4 + [0]\*4)  # 1 = Fruit, 0 = Vegetable

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

# Supervised

sup = KNeighborsClassifier(n\_neighbors=3)

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

print("Supervised Accuracy:", sup.score(X\_test, y\_test))

# Unsupervised

unsup = KMeans(n\_clusters=2, random\_state=42)

unsup.fit(X)

print("Unsupervised Clusters:", np.bincount(unsup.labels\_))

# Semi-supervised

y\_train\_semi = y\_train.copy()

for i in random.sample(range(len(y\_train)), 2):

    y\_train\_semi[i] = -1

semi = SelfTrainingClassifier(base\_estimator=DecisionTreeClassifier())

semi.fit(X\_train, y\_train\_semi)

print("Semi-Supervised Accuracy:", semi.score(X\_test, y\_test))

# Reinforcement Learning (Q-Learning)

class QLearning:

    def \_\_init\_\_(self, n=2): self.q = np.zeros((2, n))

    def act(self, s): return np.random.choice(2) if np.random.rand() < 0.1 else np.argmax(self.q[s])

    def update(self, s, a, r, s2): self.q[s, a] += 0.1 \* (r + 0.9 \* np.max(self.q[s2]) - self.q[s, a])

agent = QLearning()

for \_ in range(50):

    s = np.random.choice(2)

    a = agent.act(s)

    agent.update(s, a, [1, -1][s], (s+1)%2)

print("Q-Table:\n", agent.q)

**ML ALGORITHMS**

**1. Principal Component Analysis (PCA)**

**Explanation:**  
Principal Component Analysis is a statistical technique that transforms high-dimensional data into a lower-dimensional form while preserving as much variance as possible. It identifies the axes (principal components) along which the variation in the data is maximum. PCA is commonly used for visualization, noise reduction, and improving computational efficiency in machine learning tasks. The first principal component explains the maximum variance, and each subsequent component is orthogonal to the previous and explains the remaining variance.

**Python Code:**

from sklearn.datasets import load\_iris

from sklearn.decomposition import PCA

# Load dataset

iris = load\_iris()

X = iris.data

# Apply PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

print("Transformed shape:", X\_pca.shape)

**2. k-Nearest Neighbour (k-NN)**

**Explanation:**  
k-NN is a supervised learning algorithm used for classification and regression. In classification, it assigns a class to a data point based on the majority class among its 'k' nearest neighbors in the training set. The distance between data points is usually measured using the Euclidean distance. It is a lazy learning algorithm as it does not learn a model but rather makes predictions based on the entire training dataset.

**Python Code:**

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

# Load dataset

iris = load\_iris()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.2)

# Train k-NN

knn = KNeighborsClassifier(n\_neighbors=3)

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

print("Accuracy:", knn.score(X\_test, y\_test))

**3. Locally Weighted Regression (LWR)**

**Explanation:**  
Locally Weighted Regression (LWR) is an instance-based learning algorithm that fits a regression model to a local subset of the data around a query point. It uses a weighting function (commonly Gaussian) to assign higher weights to nearby points and lower weights to distant points. This makes LWR flexible and effective for modeling complex, non-linear relationships. It is computationally expensive but offers high accuracy for small datasets.

**Python Code:**

import numpy as np

import matplotlib.pyplot as plt

def gaussian\_kernel(x, x0, tau):

return np.exp(-np.sum((x - x0)\*\*2, axis=1) / (2 \* tau\*\*2))

def locally\_weighted\_regression(X, y, x0, tau=0.5):

W = np.diag(gaussian\_kernel(X, x0, tau))

theta = np.linalg.pinv(X.T @ W @ X) @ X.T @ W @ y

return x0 @ theta

X = np.linspace(0, 10, 100)

y = np.sin(X) + 0.1 \* np.random.randn(100)

X\_ = np.c\_[np.ones(100), X]

y\_pred = np.array([locally\_weighted\_regression(X\_, y, np.array([1, xi])) for xi in X])

plt.plot(X, y, label='Original')

plt.plot(X, y\_pred, label='LWR')

plt.legend()

plt.show()

**4. Linear Regression**

**Explanation:**  
Linear regression is a basic and widely used algorithm in regression tasks. It assumes a linear relationship between the input features and the target variable. The model tries to fit a straight line (or hyperplane in multiple dimensions) that minimizes the residual sum of squares between the observed and predicted values. It is efficient and interpretable but may not perform well with non-linear data.

**Python Code:**

from sklearn.datasets import fetch\_california\_housing

from sklearn.linear\_model import LinearRegression

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

# Load dataset

data = fetch\_california\_housing()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, test\_size=0.2)

# Train model

model = LinearRegression()

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

print("Score:", model.score(X\_test, y\_test))

**5. Polynomial Regression**

**Explanation:**  
Polynomial regression extends linear regression by fitting a polynomial equation to the dataset. It introduces non-linearity by adding polynomial features of the original features (e.g., x^2, x^3). This allows the model to fit curved data, making it more flexible. However, it may overfit if the degree of the polynomial is too high.

**Python Code:**

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

import numpy as np

import matplotlib.pyplot as plt

X = np.linspace(0, 10, 100).reshape(-1, 1)

y = np.sin(X).ravel() + np.random.randn(100) \* 0.1

poly = PolynomialFeatures(degree=3)

X\_poly = poly.fit\_transform(X)

model = LinearRegression()

model.fit(X\_poly, y)

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

plt.plot(X, y, label='Original')

plt.plot(X, y\_pred, label='Polynomial Fit')

plt.legend()

plt.show()

**6. ID3 (Decision Tree using Entropy)**

**Explanation:**  
ID3 (Iterative Dichotomiser 3) is a decision tree algorithm used for classification. It builds the tree by selecting the attribute that yields the highest information gain (reduction in entropy) for the target variable. The process continues recursively for each branch until all samples are classified or attributes are exhausted. ID3 is intuitive, interpretable, and performs well on categorical data.

**Python Code:**

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier

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

# Load dataset

iris = load\_iris()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.2)

# Train Decision Tree with entropy

clf = DecisionTreeClassifier(criterion='entropy')

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

print("Accuracy:", clf.score(X\_test, y\_test))

**7. Naive Bayes**

**Explanation:**  
Naive Bayes is a probabilistic classifier based on Bayes' Theorem, assuming independence among predictors. Despite the 'naive' assumption, it performs well in many real-world situations, especially with text data. It calculates the posterior probability for each class and predicts the class with the highest probability. Variants include Gaussian, Multinomial, and Bernoulli Naive Bayes for different types of data.

**Python Code:**

from sklearn.datasets import load\_breast\_cancer

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

from sklearn.naive\_bayes import GaussianNB

# Load dataset

cancer = load\_breast\_cancer()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(cancer.data, cancer.target, test\_size=0.2)

# Train Naive Bayes

model = GaussianNB()

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

print("Accuracy:", model.score(X\_test, y\_test))

**Python viva questions**

**✅ Section 1: Basic Python Viva Questions**

**Q1: What is Python?**  
**A:** Python is an interpreted, high-level, general-purpose programming language with simple syntax and vast libraries, suitable for scripting, automation, web development, data science, AI, etc.

**Q2: What are the key features of Python?**  
**A:**

* Easy syntax
* Interpreted and dynamically typed
* Supports multiple paradigms (OOP, functional)
* Huge standard library
* Cross-platform support

**Q3: Difference between list and tuple?**  
**A:**

* List: mutable, []
* Tuple: immutable, ()
* Lists are generally slower than tuples

**Q4: What is a dictionary in Python?**  
**A:** A key-value data structure:

python

CopyEdit

person = {'name': 'John', 'age': 30}

**Q5: What is a set in Python?**  
**A:** Unordered, mutable, no duplicate elements:

python

CopyEdit

s = {1, 2, 3}

**Q6: Explain list comprehension.**  
**A:** Compact way to create lists:

python

CopyEdit

squares = [x\*x for x in range(5)]

**Q7: What is the difference between is and ==?**  
**A:**

* is: checks identity (same memory location)
* ==: checks value equality

**Q8: What are Python's logical operators?**  
**A:** and, or, not

**Q9: What is the difference between append() and extend()?**  
**A:**

* append(x) adds a single element
* extend(iterable) adds multiple elements

**Q10: What is slicing in Python?**  
**A:** Extracting a portion of a list/string:

python

CopyEdit

s = "Python"

print(s[1:4]) # "yth"

**✅ Section 2: ML Algorithm-Based Viva Questions**

**📌 Principal Component Analysis (PCA)**

**Q11: What is PCA used for?**  
**A:** Dimensionality reduction while retaining maximum variance.

**Q12: How does PCA work?**  
**A:** It computes eigenvectors and eigenvalues of the covariance matrix to project data onto a lower-dimensional space.

**📌 k-Nearest Neighbors (KNN)**

**Q13: What is KNN?**  
**A:** A lazy learning algorithm that classifies a sample based on majority voting of its k nearest neighbors.

**Q14: Which distance metric does KNN use?**  
**A:** Euclidean distance (commonly), but can use others.

**Q15: Is KNN supervised or unsupervised?**  
**A:** Supervised learning.

**📌 Locally Weighted Regression (LWR)**

**Q16: What is LWR?**  
**A:** A non-parametric algorithm that gives more weight to nearby training data when fitting a regression line.

**Q17: What does the parameter tau represent?**  
**A:** Bandwidth controlling how local the regression is.

**📌 Linear Regression**

**Q18: What is linear regression?**  
**A:** A technique to model the linear relationship between dependent and independent variables.

**Q19: What is the cost function in linear regression?**  
**A:** Mean Squared Error (MSE)

**📌 Polynomial Regression**

**Q20: Why use polynomial regression?**  
**A:** When the relationship is non-linear, but can be modeled with polynomial terms.

**Q21: What are the risks of using high-degree polynomials?**  
**A:** Overfitting and increased complexity.

**📌 ID3 Algorithm**

**Q22: What is the ID3 algorithm?**  
**A:** A decision tree algorithm that uses information gain and entropy to split nodes.

**Q23: What is entropy in ID3?**  
**A:** A measure of impurity/uncertainty in the dataset.

**📌 Naive Bayes**

**Q24: What is the naive assumption in Naive Bayes?**  
**A:** Features are conditionally independent given the class.

**Q25: Is Naive Bayes good with large datasets?**  
**A:** Yes, it is very fast and effective.

**✅ Section 3: Dataset-Based Viva Questions**

**Q26: What is the Iris dataset used for?**  
**A:** Classification into 3 flower species based on petal and sepal features.

**Q27: What are the features of the Breast Cancer dataset?**  
**A:** Attributes like mean radius, texture, perimeter, etc., to classify as benign or malignant.

**Q28: What is the Boston Housing dataset?**  
**A:** Predicting house prices based on features like crime rate, number of rooms, etc.

**Q29: Why is Boston dataset deprecated?**  
**A:** Due to ethical concerns around feature content.

**Q30: What is the Digits dataset?**  
**A:** Handwritten digit images used for classification.

**Q31: What is a synthetic dataset?**  
**A:** Artificially generated data using functions like make\_classification() or make\_regression().

**✅ Section 4: Special Problem Questions**

**📌 Tic-Tac-Toe using DFS**

**Q32: How is DFS used in Tic-Tac-Toe?**  
**A:** It explores all possible game states recursively to determine a winning move.

**Q33: What data structure is used in DFS?**  
**A:** Stack

**Q34: How is the board represented?**  
**A:** A 3x3 list of lists.

**Q35: Can DFS always find the best move?**  
**A:** Not always optimal — best used in small state spaces.

**📌 8-Puzzle Disjoint Sets**

**Q36: What is the 8-puzzle problem?**  
**A:** A sliding puzzle with 8 numbered tiles and one empty space, goal is to reach a target configuration.

**Q37: What are the disjoint sets of states in 8-puzzle?**  
**A:** Based on inversion count; even inversions and odd inversions form separate sets.

**Q38: How do you check if a puzzle is solvable?**  
**A:** By counting the number of inversions — if even, it’s solvable.

**📌 Predicate Logic and Knowledge Rules**

**Q39: What is predicate logic?**  
**A:** A symbolic logic using predicates, variables, and quantifiers to represent knowledge.

**Q40: Give an example of predicate logic.**  
**A:**

* Human(Socrates)
* ∀x Human(x) → Mortal(x)

**Q41: How is predicate logic used in AI?**  
**A:** To represent knowledge and perform reasoning (e.g., in expert systems).