22IT149 – shrinath p
22IT150 – vasudevan c

MACHINE LEARNING ASSIGNMENT 2

Cluster algorithm:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Sample data in a dictionary format (replace this with your actual data source)
data = {
    'VIN': ['1N4AZ0CP5D', '1N4AZ1CP8K', '5YJXCAE28L', 'SADHC2S1XK', 'JN1AZ0CP9B',
            '1G1RB6S58J', '5YJ3E1EB7K', '3FA6P0SU5E', '5YJ3E1EB3K', '1C4JJXP6XN',
            '5YJSA1E29L', '5YJYGDEE3L', 'JHMZC5F1XJ', '1N4AZ0CP1D', '1N4AZ1BP4L',
            'KMHC75LH5K', '5YJ3E1EBXJ', '5YJ3E1EA5K', 'WA1F2AFY8N'],
    'Electric Range': [75, 150, 293, 234, 73, 53, 220, 19, 220, 21,
                       330, 291, 47, 75, 149, 29, 215, 220, 23],
    'Base MSRP': [0] * 19  # Assuming Base MSRP is not required for clustering
}

# Create a DataFrame
df = pd.DataFrame(data)

# Select relevant features for clustering
X = df[['Electric Range']]

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# K-Means Scatter plot with Centroids
plt.figure(figsize=(10, 6))
plt.scatter(df['Electric Range'], [0] * len(df), c=df['Cluster'], cmap='viridis', marker='o')
plt.scatter(kmeans.cluster_centers_[:, 0], [0] * kmeans.n_clusters, c='red', marker='X',
s=200, label='Centroids')
plt.title('K-Means Clustering of Electric Vehicles')
plt.xlabel('Electric Range (miles)')
plt.yticks([])  # Hide y-axis
plt.legend()
plt.grid()
plt.show()

# 1. Bar plot of electric range for each vehicle with cluster color coding
```

22IT149 – shrinath p
22IT150 – vasudevan c

```
plt.figure(figsize=(10, 6))
plt.bar(df['VIN'], df['Electric Range'], color=plt.cm.viridis(df['Cluster'] / 2))
plt.xticks(rotation=90)
plt.title('Electric Range of Vehicles by VIN (Colored by Cluster)')
plt.xlabel('VIN')
plt.ylabel('Electric Range (miles)')
plt.grid()
plt.show()

# 2. Histogram showing the distribution of Electric Range
plt.figure(figsize=(10, 6))
plt.hist(df['Electric Range'], bins=10, color='skyblue', edgecolor='black')
plt.title('Electric Range Distribution')
plt.xlabel('Electric Range (miles)')
plt.ylabel('Frequency')
plt.grid()
plt.show()

# 3. Bar plot showing the number of vehicles per cluster
cluster_counts = df['Cluster'].value_counts().sort_index()

plt.figure(figsize=(8, 6))
plt.bar(cluster_counts.index, cluster_counts.values, color='lightgreen', edgecolor='black')
plt.xticks(ticks=[0, 1, 2], labels=['Cluster 0', 'Cluster 1', 'Cluster 2'])
plt.title('Number of Vehicles per Cluster')
plt.xlabel('Cluster')
plt.ylabel('Number of Vehicles')
plt.grid()
plt.show()

# Print initial and final clusters
print("Initial Clusters (before fitting):")
print(X)

print("\nFinal Clusters (after fitting):")
print(df[['VIN', 'Electric Range', 'Cluster']])
```

!! 2 cluster

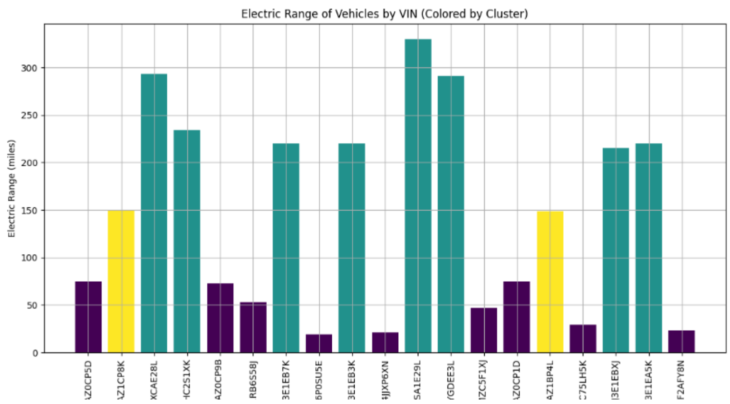
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler
```

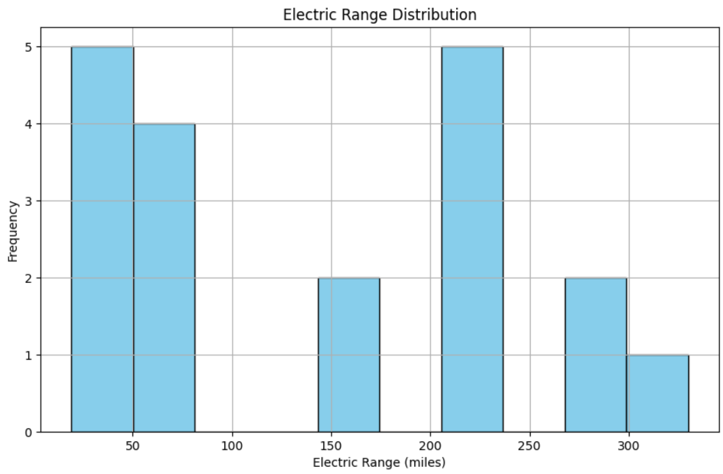
22IT149 – shrinath p
22IT150 – vasudevan c

```
# Sample data in a dictionary format (replace this with your actual data source)
data = {
    'VIN': ['1N4AZ0CP5D', '1N4AZ1CP8K', '5YJXCAE28L', 'SADHC2S1XK', 'JN1AZ0CP9B',
            '1G1RB6S58J', '5YJ3E1EB7K', '3FA6P0SU5E', '5YJ3E1EB3K', '1C4JJXP6XN',
            '5YJSA1E29L', '5YJYGDEE3L', 'JHMZC5F1XJ', '1N4AZ0CP1D', '1N4AZ1BP4L',
            'KMHC75LH5K', '5YJ3E1EBXJ', '5YJ3E1EA5K', 'WA1F2AFY8N'],
    'Electric Range': [75, 150, 293, 234, 73, 53, 220, 19, 220, 21,
                       330, 291, 47, 75, 149, 29, 215, 220, 23],
    'Base MSRP': [0] * 19  # Assuming Base MSRP is not required for clustering
}

# Create a DataFrame
df = pd.DataFrame(data)

# Select relevant features for clustering
X = df[['Electric Range']]

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Visualization
plt.figure(figsize=(10, 6))
plt.scatter(df['Electric Range'], [0] * len(df), c=df['Cluster'], cmap='viridis', marker='o')
plt.scatter(kmeans.cluster_centers_[:, 0], [0] * kmeans.n_clusters, c='red', marker='X', s=200,
label='Centroids')
```

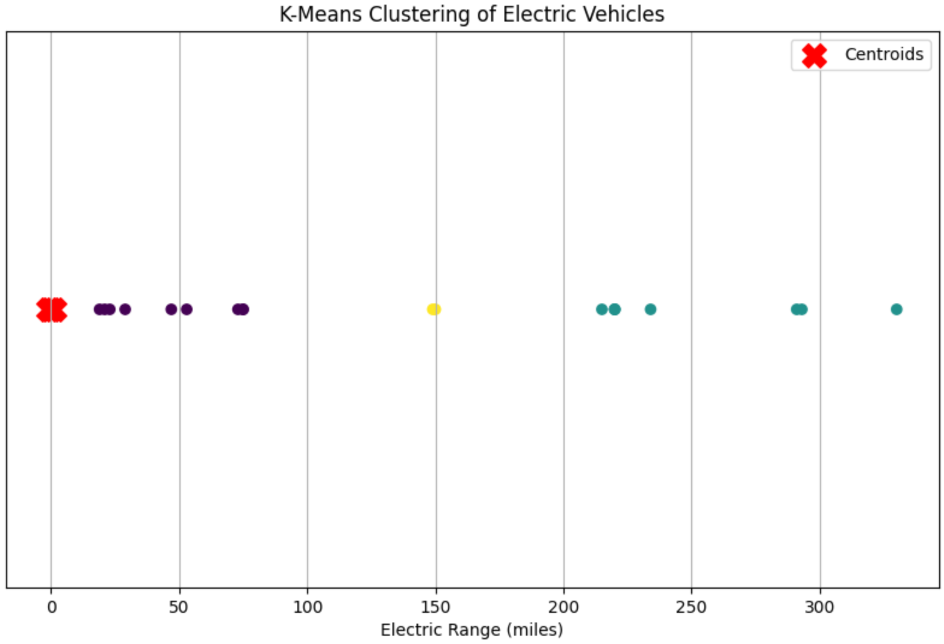
22IT149 – shrinath p
22IT150 – vasudevan c

```
plt.title('K-Means Clustering of Electric Vehicles')

plt.xlabel('Electric Range (miles)')

plt.yticks([])  # Hide y-axis

plt.legend()

plt.grid()

plt.show()


# Print initial and final clusters

print("Initial Clusters (before fitting):")

print(X)


print("\nFinal Clusters (after fitting):")

print(df[['VIN', 'Electric Range', 'Cluster']])
```





22IT149 – shrinath p
22IT150 – vasudevan c




```
D:\3 year\ml\asignment 2>python main.py
Initial Clusters (before fitting):
    Electric Range
0               75
1              150
2              293
3              234
4               73
5               53
6              220
7               19
8              220
9               21
10             330
11             291
12              47
13              75
14             149
15              29
16             215
17             220
18              23
```

22IT149 – shrinath p
22IT150 – vasudevan c

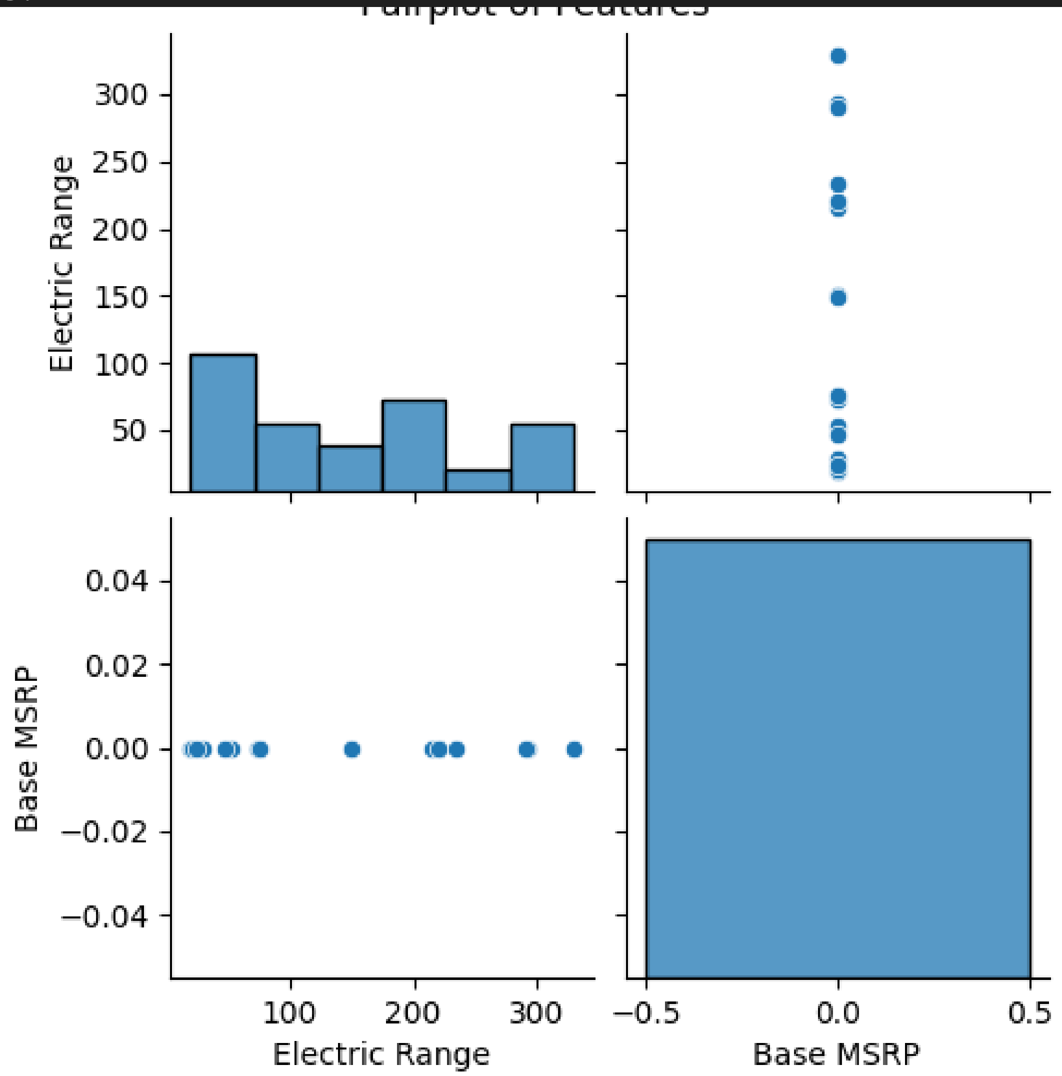
```
Final Clusters (after fitting):
           VIN  Electric Range  Cluster
0   1N4AZ0CP5D              75        0
1   1N4AZ1CP8K             150        2
2   5YJXCAE28L             293        1
3   SADHC2S1XK             234        1
4   JN1AZ0CP9B              73        0
5   1G1RB6S58J              53        0
6   5YJ3E1EB7K             220        1
7   3FA6P0SU5E              19        0
8   5YJ3E1EB3K             220        1
9   1C4JJXP6XN              21        0
10  5YJSA1E29L             330        1
11  5YJYGDEE3L             291        1
12  JHMZC5F1XJ              47        0
13  1N4AZ0CP1D              75        0
14  1N4AZ1BP4L             149        2
15  KMHC75LH5K              29        0
16  5YJ3E1EBXJ             215        1
17  5YJ3E1EA5K             220        1
18  WA1F2AFY8N              23        0
```

1. K means

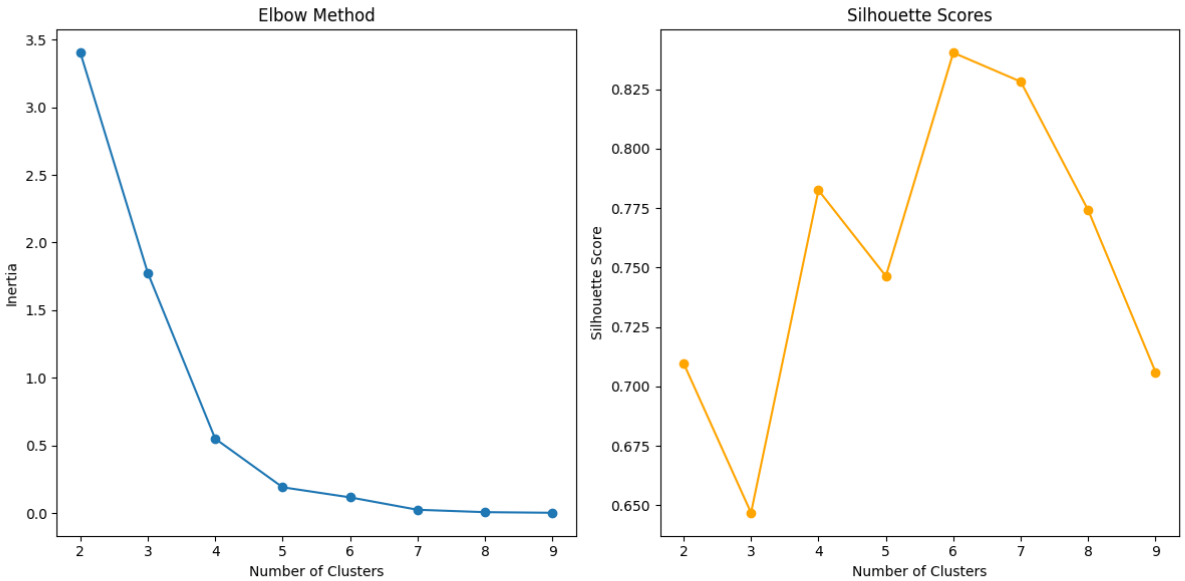
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import seaborn as sns

# Load the data
data = {
    "VIN": ["1N4AZ0CP5D", "1N4AZ1CP8K", "5YJXCAE28L", "SADHC2S1XK", "JN1AZ0CP9B", "1G1RB6S58J",
            "5YJ3E1EB7K", "3FA6P0SU5E", "5YJ3E1EB3K", "1C4JJXP6XN", "5YJSA1E29L", "5YJYGDEE3L",
            "JHMZC5F1XJ", "1N4AZ0CP1D", "1N4AZ1BP4L", "KMHC75LH5K", "5YJ3E1EBXJ", "5YJ3E1EA5K",
            "WA1F2AFY8N"],
    "Electric Range": [75, 150, 293, 234, 73, 53, 220, 19, 220, 21, 330, 291, 47, 75, 149, 29, 215, 220, 23],
    "Base MSRP": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
}

df = pd.DataFrame(data)

# Preprocess the data
X = df[['Electric Range', 'Base MSRP']]

# Normalize the data
scaler = StandardScaler()
```

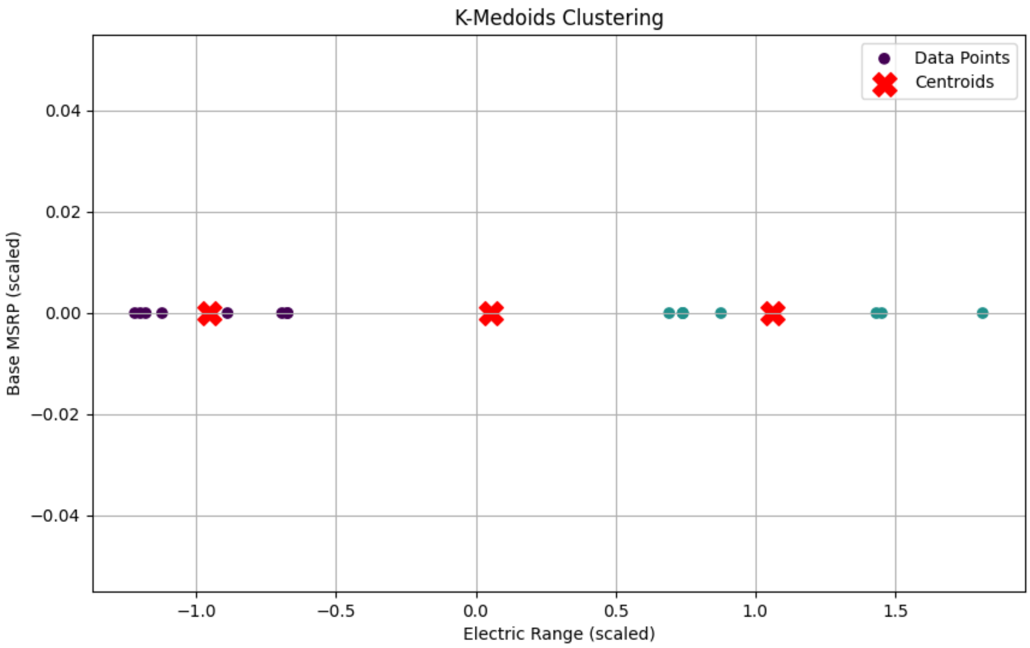
```python
X_scaled = scaler.fit_transform(X)

# Elbow Method to find optimal number of clusters
inertia = []
silhouette_scores = []
K = range(2, 10)

for k in K:
    kmedoids = KMeans(n_clusters=k, random_state=42)
    kmedoids.fit(X_scaled)
    inertia.append(kmedoids.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, kmedoids.labels_))

# Plot the Elbow Method
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(K, inertia, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')

# Plot Silhouette Scores
plt.subplot(1, 2, 2)
plt.plot(K, silhouette_scores, marker='o', color='orange')
plt.title('Silhouette Scores')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.tight_layout()
plt.show()

# Run K-Medoids with the optimal k (let's use k=3 for this example)
k = 3  # Number of clusters
kmedoids = KMeans(n_clusters=k, random_state=42)
y_kmedoids = kmedoids.fit_predict(X_scaled)

# Visualization of Clusters
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y_kmedoids, cmap='viridis', marker='o', label='Data Points')
plt.scatter(kmedoids.cluster_centers_[:, 0], kmedoids.cluster_centers_[:, 1], c='red', marker='X', s=200, label='Centroids')
plt.title('K-Medoids Clustering')
plt.xlabel('Electric Range (scaled)')
plt.ylabel('Base MSRP (scaled)')
plt.legend()
plt.grid()
```

```python
plt.show()

# Pairplot to visualize relationships between features
sns.pairplot(df[['Electric Range', 'Base MSRP']])
plt.suptitle('Pairplot of Features', y=1.02)
plt.show()

# Cluster Distribution
df['Cluster'] = y_kmedoids
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='Cluster', palette='viridis')
plt.title('Cluster Distribution')
plt.xlabel('Cluster')
plt.ylabel('Number of Vehicles')
plt.show()

```



22IT149 – shrinath p
22IT150 – vasudevan c





2 RL implementation

Grid navigation:

```
import numpy as np

import matplotlib.pyplot as plt


# Parameters

grid_size = 5
```

22IT149 – shrinath p
22IT150 – vasudevan c

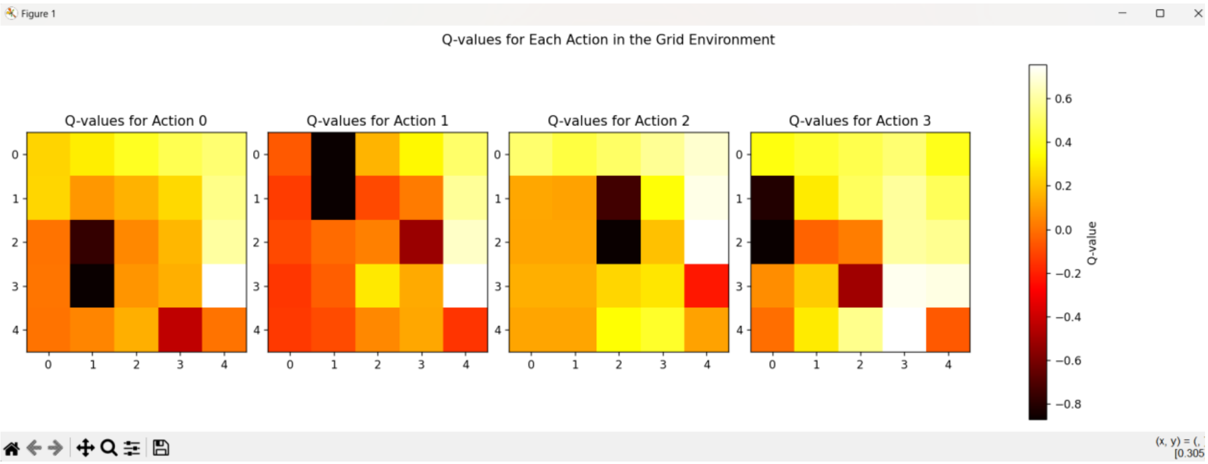
```
num_episodes = 500

learning_rate = 0.1

discount_factor = 0.9

epsilon = 1.0

epsilon_decay = 0.99

epsilon_min = 0.01


# Action space: 0 - Up, 1 - Down, 2 - Left, 3 - Right

actions = [0, 1, 2, 3]

q_table = np.zeros((grid_size, grid_size, len(actions)))


# Rewards setup

goal = (4, 4)

obstacles = [(1, 1), (2, 1), (3, 3)]

rewards = np.zeros((grid_size, grid_size))

rewards[goal] = 1  # Reward for reaching the goal

for obs in obstacles:

    rewards[obs] = -1  # Penalty for hitting obstacles


def get_state(position):

    return position[0], position[1]


# Training the agent

for episode in range(num_episodes):

    position = (0, 0)  # Start position

    total_reward = 0


    while position != goal:

        if np.random.rand() < epsilon:

            action = np.random.choice(actions)  # Explore

        else:
```

22IT149 – shrinath p
22IT150 – vasudevan c

```
            action = np.argmax(q_table[position[0], position[1]])  # Exploit

        # Move based on the action
        if action == 0 and position[0] > 0:  # Up
            new_position = (position[0] - 1, position[1])
        elif action == 1 and position[0] < grid_size - 1:  # Down
            new_position = (position[0] + 1, position[1])
        elif action == 2 and position[1] > 0:  # Left
            new_position = (position[0], position[1] - 1)
        elif action == 3 and position[1] < grid_size - 1:  # Right
            new_position = (position[0], position[1] + 1)
        else:
            new_position = position  # Stay in place if out of bounds

        # Update reward and Q-value
        reward = rewards[new_position]
        total_reward += reward
        q_table[position[0], position[1], action] += learning_rate * (
            reward + discount_factor * np.max(q_table[new_position[0], new_position[1]]) -
            q_table[position[0], position[1], action]
        )
        position = new_position  # Move to new position

    # Decay epsilon
    epsilon = max(epsilon_min, epsilon * epsilon_decay)

print("Training completed.")

# Visualization of learned Q-values
fig, axs = plt.subplots(1, len(actions), figsize=(15, 5))
```

22IT149 – shrinath p
22IT150 – vasudevan c

```
for i in range(len(actions)):

    axs[i].imshow(q_table[:, :, i], cmap='hot', interpolation='nearest')

    axs[i].set_title(f'Q-values for Action {i}')

    axs[i].set_xticks(np.arange(grid_size))

    axs[i].set_yticks(np.arange(grid_size))

    axs[i].set_xticklabels(np.arange(grid_size))

    axs[i].set_yticklabels(np.arange(grid_size))

    axs[i].grid(False)


plt.suptitle('Q-values for Each Action in the Grid Environment')

plt.tight_layout()

plt.colorbar(axs[0].imshow(q_table[:, :, 0], cmap='hot', interpolation='nearest'), ax=axs,
orientation='vertical', label='Q-value')

plt.show()
```



2 TIC TAC Toa:

```
import numpy as np

import matplotlib.pyplot as plt


# Parameters

num_episodes = 20000  # Increased number of episodes
```

22IT149 – shrinath p
22IT150 – vasudevan c

```
learning_rate = 0.1
discount_factor = 0.9
epsilon = 1.0
epsilon_decay = 0.9995  # Slower decay to encourage exploration
epsilon_min = 0.01

# Action space: 0-8 (positions on the board)
q_table = np.random.rand(3**9, 9) * 0.01  # Small random values to start

def state_to_index(state):
    """Convert board state to a unique index."""
    index = 0
    for i in range(9):
        index += (3**i) * state[i]
    return index

def check_winner(state):
    """Check if there is a winner."""
    winning_positions = [(0, 1, 2), (3, 4, 5), (6, 7, 8),
                         (0, 3, 6), (1, 4, 7), (2, 5, 8),
                         (0, 4, 8), (2, 4, 6)]
    for pos in winning_positions:
        if state[pos[0]] == state[pos[1]] == state[pos[2]] != 0:
            return state[pos[0]]
    return 0 if 0 in state else -1  # Return 0 for ongoing, -1 for draw

# Performance tracking
win_counts = np.zeros(num_episodes)
draw_counts = np.zeros(num_episodes)
loss_counts = np.zeros(num_episodes)
```

22IT149 – shrinath p
22IT150 – vasudevan c

```
def smart_opponent(state):
    """Return the opponent's action (blocking strategy)."""
    available_actions = np.where(state == 0)[0]
    for action in available_actions:
        state[action] = 2
        if check_winner(state) == 2:  # If the opponent can win
            state[action] = 0  # Reset
            return action
        state[action] = 0  # Reset

    # If no blocking move, return random
    return np.random.choice(available_actions)

# Training the agent
for episode in range(num_episodes):
    state = np.zeros(9, dtype=int)  # Empty board

    while True:
        state_index = state_to_index(state)

        if np.random.rand() < epsilon:
            available_actions = np.where(state == 0)[0]  # Available moves
            if available_actions.size == 0:  # No available actions
                draw_counts[episode] += 1
                break  # End the game if board is full
            action = np.random.choice(available_actions)  # Explore
        else:
            action = np.argmax(q_table[state_index])  # Exploit

        # Make the move
        state[action] = 1  # Agent's move
```

22IT149 – shrinath p
22IT150 – vasudevan c

```
        # Check for winner
        winner = check_winner(state)
        if winner == 1:  # Agent wins
            win_counts[episode] += 1
            break
        elif winner == -1:  # Draw
            draw_counts[episode] += 1
            break

        # Opponent's smart move
        opponent_action = smart_opponent(state)
        state[opponent_action] = 2  # Opponent's move

        # Check for opponent's win
        winner = check_winner(state)
        if winner == 2:  # Opponent wins
            loss_counts[episode] += 1
            break

        # Update Q-value
        next_state_index = state_to_index(state)
        reward = 0  # Default reward
        if winner == 1:
            reward = 1  # Win
        elif winner == -1:
            reward = 0  # Draw
        elif winner == 2:
            reward = -1  # Loss

        q_table[state_index, action] += learning_rate * (
```

```
            reward + discount_factor * np.max(q_table[next_state_index]) -
            q_table[state_index, action]
        )

    # Decay epsilon
    epsilon = max(epsilon_min, epsilon * epsilon_decay)

# Plotting the results
plt.figure(figsize=(12, 6))
episodes = np.arange(num_episodes)

plt.plot(episodes, np.cumsum(win_counts), label='Wins', color='green')
plt.plot(episodes, np.cumsum(loss_counts), label='Losses', color='red')
plt.plot(episodes, np.cumsum(draw_counts), label='Draws', color='blue')

plt.xlabel('Episodes')
plt.ylabel('Total Outcomes')
plt.title('Tic-Tac-Toe Agent Performance Over Episodes')
plt.legend()
plt.grid()
plt.show()
```

22IT149 – shrinath p
22IT150 – vasudevan c