**Approach:**

I implemented a Q-learning approach to solve the three problems using sampled transitions from provided datasets.

1. Importing Libraries:
   * numpy (as np) for numerical operations.
   * pandas (as pd) for data manipulation.
   * time for measuring the execution time.
2. Q-learning Functions:
   * initialize\_q\_table: Initializes a Q-table with zeros.
   * epsilon\_greedy\_action: Chooses an action based on epsilon-greedy strategy for exploration and exploitation.
   * update\_q\_table: Updates the Q-table based on the Q-learning update rule.
   * q\_learning: Applies Q-learning to learn a policy from the given dataset.
3. Policy Extraction and Saving Functions:
   * get\_policy: Extracts the policy from the final Q-table.
   * save\_policy: Saves the policy to a file in the required format.
4. Example Usage:
   * Reads three different datasets: 'small.csv', 'medium.csv', and 'large.csv'.
   * Applies Q-learning for each dataset, extracting the policy and saving it to a file.
   * Prints the length of each policy and the time taken for each Q-learning process.
5. Example Parameters:
   * Small Problem: 100 states, 4 actions, discount factor (gamma) of 0.95.
   * Medium Problem: 50,000 states, 7 actions, undiscounted (gamma = 1).
   * Large Problem: 312,020 states, 9 actions, discount factor of 0.95.
6. Timing Execution:
   * Measures and prints the time taken for each Q-learning process.
7. Output Files:
   * Outputs policies to 'small.policy', 'medium.policy', and 'large.policy' respectively.

**Results:**

Outputting policy of length 100 as small.policy

Time taken: 1.799976110458374 seconds

Outputting policy of length 50000 as medium.policy

Time taken: 3.7789571285247803 seconds

Outputting policy of length 312020 as large.policy

Time taken: 3.716930627822876 seconds

**Code:**

import numpy as np

import pandas as pd

import time

def initialize\_q\_table(num\_states, num\_actions):

    return np.zeros((num\_states, num\_actions))

def epsilon\_greedy\_action(q\_table, state, epsilon):

    if np.random.rand() < epsilon:

        return np.random.randint(q\_table.shape[1])  # Explore

    else:

        return np.argmax(q\_table[state])  # Exploit

def update\_q\_table(q\_table, state, action, reward, next\_state, alpha, gamma):

    current\_value = q\_table[state-1, action-1]

    max\_future\_value = np.max(q\_table[next\_state-1])

    new\_value = (1 - alpha) \* current\_value + alpha \* (reward + gamma \* max\_future\_value)

    q\_table[state-1, action-1] = new\_value

def q\_learning(dataset, num\_states, num\_actions, alpha, gamma, epsilon, num\_episodes):

    q\_table = initialize\_q\_table(num\_states, num\_actions)

    for episode in range(num\_episodes):

        state = dataset.at[episode, 's']

        action = dataset.at[episode, 'a']

        reward = dataset.at[episode, 'r']

        next\_state = dataset.at[episode, 'sp']

        update\_q\_table(q\_table, state, action, reward, next\_state, alpha, gamma)

    return q\_table

def get\_policy(q\_table):

    return np.argmax(q\_table, axis=1) + 1  # Adding 1 to convert zero-based indexing to action space (1, 2, 3, ...)

def save\_policy(policy, filename):

    with open(filename, 'w') as file:

        for action in policy:

            file.write(str(action) + '\n')

small\_dataset = pd.read\_csv('project2/data/small.csv')

medium\_dataset = pd.read\_csv('project2/data/medium.csv')

large\_dataset = pd.read\_csv('project2/data/large.csv')

# Small

start\_time = time.time()

final\_q\_table = q\_learning(dataset=small\_dataset, num\_states=100, num\_actions=4, alpha=0.1, gamma=0.95, epsilon=0.1, num\_episodes=len(small\_dataset))

policy = get\_policy(final\_q\_table)

print("Outputting policy of length " + str(len(policy)) + " as small.policy")

save\_policy(policy, 'project2/small.policy')

end\_time = time.time()

elapsed\_time = end\_time - start\_time

print(f"Time taken: {elapsed\_time} seconds")

# Medium

start\_time = time.time()

final\_q\_table = q\_learning(dataset=medium\_dataset, num\_states=50000, num\_actions=7, alpha=0.1, gamma=1, epsilon=0.1, num\_episodes=len(medium\_dataset))

policy = get\_policy(final\_q\_table)

print("Outputting policy of length " + str(len(policy)) + " as medium.policy")

save\_policy(policy, 'project2/medium.policy')

end\_time = time.time()

elapsed\_time = end\_time - start\_time

print(f"Time taken: {elapsed\_time} seconds")

# Large

start\_time = time.time()

final\_q\_table = q\_learning(dataset=large\_dataset, num\_states=312020, num\_actions=9, alpha=0.1, gamma=0.95, epsilon=0.1, num\_episodes=len(large\_dataset))

policy = get\_policy(final\_q\_table)

print("Outputting policy of length " + str(len(policy)) + " as large.policy")

save\_policy(policy, 'project2/large.policy')

end\_time = time.time()

elapsed\_time = end\_time - start\_time

print(f"Time taken: {elapsed\_time} seconds")