Ex5 : Implement MC Prediction in Blackjack game

```
import gymnasium as gym

from collections import defaultdict

env=gym.make('Blackjack-v1',render_mode='human')
```

---

1. **A state is represented as a tuple of three values:**

    1. The value of the sum of our cards
    2. The face value of one of the dealer's card
    3. Boolean value—`True` if we have a useable ace and `False` if we don't have a useable ace

```
print(env.reset())

Output : (15, 9, True)
```

2. **Action Space :**

    - The action **stand** is represented by 0
    - The action **hit** is represented by 1

```
print(env.action_space)

Output : Discrete(2)
```

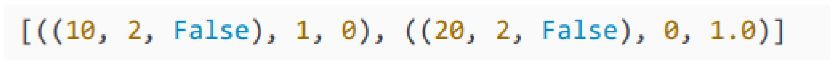
3. **Reward :**

    - **+1.0** reward if we win the game
    - **-1.0** reward if we lose the game
    - **0** reward if the game is a draw

```
# STEP 1 : DEFINE  A POLICY TO BE EVALUATED
def policy(state):
   return 0 if state[0]>19 else 1
#generate an initial state
state = env.reset()
state=state[0]
print(state)
#print the policy of this state
print(policy(state))
```

```
# STEP 2 : GENERATE AN EPISODE USING THIS POLICY
num_timesteps = 100
def generate_episode(policy):
    episode = []
    state = env.reset()
    state=state[0]
    for t in range(num_timesteps):
        action = policy(state)
        next_state, reward, done, info,trans_prob = env.step(action)
        episode.append((state, action, reward))
        if done:
            break
        state=next_state
    return episode
```

print(generate_episode(policy))



```
[((10, 2, False), 1, 0), ((20, 2, False), 0, 1.0)]
```

There are 2 states in this episode. We performed action 1 (hit) in state (10,2,False) and got a reward of 0. We performed action 0 (stand) in state (20,2, False) and got a reward of 1.0.

# STEP 3: COMPUTE THE VALUE FUNCTION OF THE STATES

To predict the value function, we generate several episodes using the given policy and compute the value of the state as an average return across several episodes

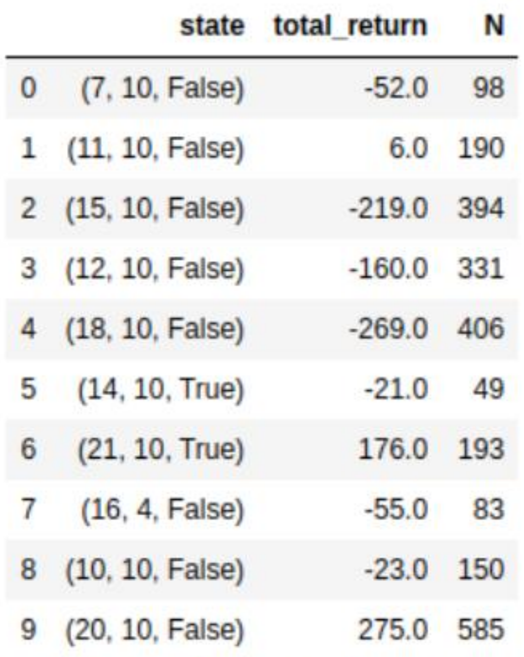
```
total_return = defaultdict(float)
N = defaultdict(int)

num_iterations = 50000
for i in range(num_iterations):
    episode = generate_episode(policy)
    states, actions, rewards = zip(*episode)
    for t, state in enumerate(states):
        if state not in states[0:t]:
         R = (sum(rewards[t:]))
         total_return[state] = total_return[state] + R
         N[state] = N[state] + 1

print(total_return[state])
print(N[state])

# convert both the dictionaries to a Data frame, Merge the frames based on
#‘state’ column

total_return = pd.DataFrame(total_return.items(),columns=['state',
'total_return'])
N = pd.DataFrame(N.items(),columns=['state', 'N'])
df = pd.merge(total_return, N, on="state")
print(df.head(10))
```



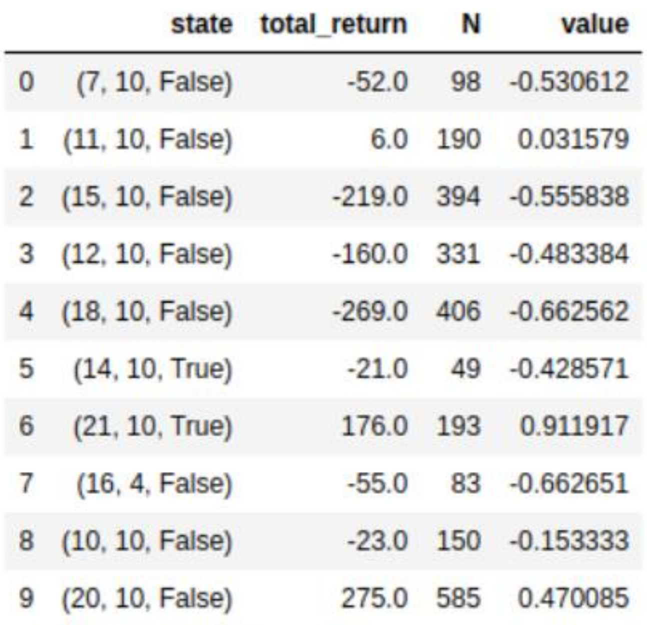
	state	total_return	N
0	(7, 10, False)	-52.0	98
1	(11, 10, False)	6.0	190
2	(15, 10, False)	-219.0	394
3	(12, 10, False)	-160.0	331
4	(18, 10, False)	-269.0	406
5	(14, 10, True)	-21.0	49
6	(21, 10, True)	176.0	193
7	(16, 4, False)	-55.0	83
8	(10, 10, False)	-23.0	150
9	(20, 10, False)	275.0	585

Figure 4.19: The total return and the number of times a state has been visited

Next, we can compute the value of the state as the average return

$$V(s) = \frac{\text{total\_return}(s)}{N(s)}$$

```
df['value'] = df['total_return']/df['N']
print(df.head(10))
print(df.shape)
```



	state	total_return	N	value
0	(7, 10, False)	-52.0	98	-0.530612
1	(11, 10, False)	6.0	190	0.031579
2	(15, 10, False)	-219.0	394	-0.555838
3	(12, 10, False)	-160.0	331	-0.483384
4	(18, 10, False)	-269.0	406	-0.662562
5	(14, 10, True)	-21.0	49	-0.428571
6	(21, 10, True)	176.0	193	0.911917
7	(16, 4, False)	-55.0	83	-0.662651
8	(10, 10, False)	-23.0	150	-0.153333
9	(20, 10, False)	275.0	585	0.470085

Figure 4.20: The value is calculated as the average of the return of each state

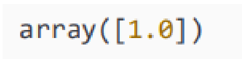
We have successfully predicted the value function of the given policy using the every-visit MC method.

Check the value of some states using the policy we set in the beginning (which was an optimal policy), which selects the action 0 (stand) when the sum value is greater than 19 and the action 1 (hit) when the sum value is lower than 19.

Question 1 : Evaluate the value of the state (21,9,False). This is a good state and hence should have a high value.

df[df['state']==(21,9,False)]['value'].values

Output :

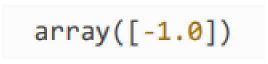


```
array([1.0])
```

Question 2 : Evaluate the value of the state (5,8,False). This is not a good state and hence should have a low value.

df[df['state']==(5,8,False)]['value'].values

Output :



```
array([-1.0])
```

Thus, we learnt how to predict the value function of the given policy using the every-visit MC prediction method.