Question 1:
Please find it attached below

Question 2:
State value function was found by iterating over all the states and updating each of them using the equation : V(s) <- summation{ p(r,s_next| s, a)* r + gamma*(V(s_next))} until convergence
And not by matrix multiplication.

Question 3:
Please find the answers attached below

Question 4:

Final V_star:

```
[[21.97714059 24.41920194 21.97728175 19.4190451  17.47714059]
 [19.77942653 21.97728175 19.77955357 17.80148388 16.02133549]
 [17.80148388 19.77955357 17.80159821 16.02133549 14.41920194]
 [16.02133549 17.80159821 16.02143839 14.41920194 12.97728175]
 [14.41920194 16.02143839 14.41920194 12.97728175 11.67955357]]
```

Final Q_star, where optimal action is argmax(Q_star)

```
[[18.77921138 18.77921138 17.80129024 21.97714059]
 [24.41920194 24.41920194 24.41920194 24.41920194]
 [18.77921138 21.97714059 17.80129024 17.47690153]
 [19.4190451  19.4190451  19.4190451  19.4190451 ]
 [14.72921138 17.47714059 14.4190451  14.72942653]
 [19.77942653 16.80148388 16.02116122 19.77942653]
 [21.97728175 17.80148388 17.80148388 17.80148388]
 [19.77942653 19.77942653 16.02116122 16.02116122]
 [17.47714059 17.80148388 14.4190451  14.4190451 ]
 [15.72942653 16.02133549 12.97714059 13.41920194]
 [17.80148388 15.02133549 14.4190451  17.80148388]
 [19.77955357 16.02133549 16.02133549 16.02133549]
 [17.80148388 17.80148388 14.4190451  14.4190451 ]
 [16.02133549 16.02133549 12.97714059 12.97714059]
 [14.41920194 14.41920194 11.67942653 11.97728175]
 [16.02133549 13.41920194 12.97714059 16.02133549]
 [17.80159821 14.41920194 14.41920194 14.41920194]
 [16.02133549 16.02133549 12.97714059 12.97714059]
 [14.41920194 14.41920194 11.67942653 11.67942653]
 [12.97728175 12.97728175 10.51148388 10.67955357]]
```

[14.41920194 11.97728175 11.97728175 14.41920194]
[16.02143839 12.97728175 13.41920194 12.97714059]
[14.41920194 14.41920194 11.97728175 11.67942653]
[12.97728175 12.97728175 10.67955357 10.51148388]
[11.67955357 11.67955357 9.51159821 9.51159821]]

Question 5:
Please find it attached below

Question 6:
Actions are as follows:
0 - move up
1 - move left
2 - move down
3 - move right

Actions at first and last state are taken as empty because they are terminal states

('STEP: ', 0)
VI
[[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]]

Policy Iteration:
[[], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], []]

('STEP: ', 1)
V
[[0. -1. -1. -1.]
[-1. -1. -1. -1.]
[-1. -1. -1. -1.]
[-1. -1. -1. 0.]]

Policy:
[[], [1], [0, 1, 2, 3], [0, 1, 2, 3], [0], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [0, 1, 2, 3], [2], [0, 1, 2, 3], [0, 1, 2, 3], [3], []]

('STEP: ', 2)
V
[[0.   -1.675 -1.9   -1.9]

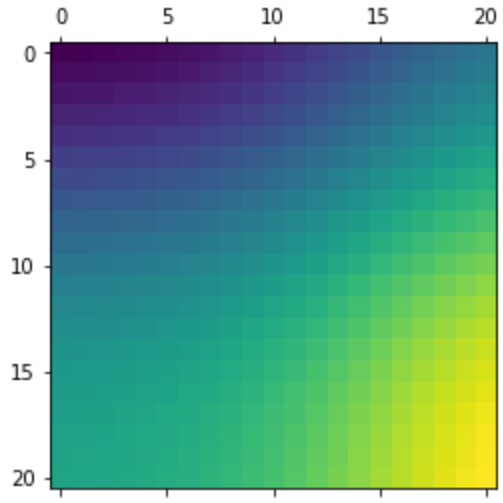
```
 [-1.675 -1.9   -1.9   -1.9  ]
 [-1.9   -1.9   -1.9   -1.675]
 [-1.9   -1.9   -1.675  0.   ]]

Policy:
[[], [1], [1], [0, 1, 2, 3], [0], [0, 1], [0, 1, 2, 3], [2], [0], [0, 1, 2, 3], [2, 3], [2], [0, 1, 2, 3], [3], [3], []]

('STEP: ', 3)
V
[[ 0.       -2.231875 -2.659375 -2.71    ]
 [-2.231875 -2.60875  -2.71     -2.659375]
 [-2.659375 -2.71     -2.60875  -2.231875]
 [-2.71     -2.659375 -2.231875  0.      ]]

Policy:
[[], [1], [1], [1, 2], [0], [0, 1], [1], [2], [0], [0], [2, 3], [2], [0, 3], [3], [3], []]

('STEP: ', 4)
V
[[ 0.         -2.6875     -3.32003125 -3.41621875]
 [-2.6875     -3.22384375 -3.37065625 -3.32003125]
 [-3.32003125 -3.37065625 -3.22384375 -2.6875    ]
 [-3.41621875 -3.32003125 -2.6875      0.        ]]

Policy:
[[], [1], [1], [1], [0], [0, 1], [1], [2], [0], [0], [2, 3], [2], [0], [3], [3], []]

('STEP: ', 5)
V
[[ 0.         -3.07705938 -3.87874141 -4.0313125 ]
 [-3.07705938 -3.72617031 -3.94474375 -3.87874141]
 [-3.87874141 -3.94474375 -3.72617031 -3.07705938]
 [-4.0313125  -3.87874141 -3.07705938  0.        ]]

Policy:
[[], [1], [1], [2], [0], [0, 1], [2], [2], [0], [3], [2, 3], [2], [0, 3], [3], [3], []]

FINAL Value function
[[ 0.         -3.4034435  -4.35966783 -4.55952426]
 [-3.4034435  -4.15981141 -4.42221027 -4.35966783]
 [-4.35966783 -4.42221027 -4.15981141 -3.4034435 ]
 [-4.55952426 -4.35966783 -3.4034435   0.        ]]
Policy
[[], [1], [1], [1, 2], [0], [0, 1], [1, 2], [2], [0], [0, 3], [2, 3], [2], [0, 3], [3], [3], []]
```

DONE

The change needed is in Policy Updation, here we need to check all the actions present in the new policy with all the old actions. Otherwise, when more than one optimal action exist, the boolean variable policy-updation will always be False, due to a different action (having exactly the same value) being selected.
The code associated with the question has the change mentioned to resolve this issue

Question 7:

The brighter the image is the more weight it has: this is the value function for the second case



Both the parts have been done,

The reward function was made as follows:
The state tells us how many cars are left by the end of day before shifting(actions), so that means these cars are 20 - cars_requested + car_returned, i assumed that initially both the location had 20 cars each, the maximum amount.
Now reward was found on the basis of cars requested times the probability associated with it. For eg: s = (10, 5), then r = double summation : 1. From 10 to 20 and 2. From 15 to 20, and inside this double summation would be pr1(request)*pr1(return)*pr2(request)*pr2(return)*(10*(i + j)) where i is from 10 to 20 and j from 15 to 20

For actions: penalty: negative reward was given

For the special case:
1 additional function was added for penalty if more than 10 cars
I assumed that if each location has more than 10 ars then both are penalised with 4$.

Furthermore, the second case took a took a lot more iterations than the first case to reach to hte best policy via policy iteration

The code explains the rest!

Q1

s	a	s'	r	p(s', r | s, a)
high	search	high	r_{search}	α
high	search	low	r_{search}	$1-\alpha$
low	search	high	-3	$1-\beta$
low	search	low	r_{search}	β
high	wait	high	r_{wait}	1
high	wait	low	-	0
low	wait	low	r_{wait}	1
low	wait	high	-	0
low	recharge	high	0	1
low	recharge	low	-	0

~~Also~~ ⇒ Zero probability also mentioned pls ignore if needed

Q3 Adding a const c to all the rewards

Now $G_t \doteq (R_{t+1}+c) + \gamma(R_{t+2}+c) + \gamma^2(R_{t+3}+c) \cdots$

3.15

$\doteq \underbrace{R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}}_{G_{t\_old}} + c + \gamma c + \gamma^2 c \cdots$

$\doteq G_{t\_old} + c(1+\gamma+\gamma^2 \cdots)$

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} + \underbrace{\frac{c}{(1-\gamma)}}_{V_c}$$

$$\boxed{V_c = c/1-\gamma}$$

As V_c added to all the terms, relatively no change in value of the terms i.e $[G_{t+1}(old) - G_{t+2}(old) = G_{t+1} - G_{t+2}]$ (Hint)

3.16 Let's suppose the episodic task goes till T

Then, $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + (\gamma)^{T} R_{t+T+1}$

Let's add a C

$$\Rightarrow G_t = (R_{t+1} + C) + \gamma(R_{t+2} + C) \cdots (\gamma)^{T}(R_{t+T+1} + C)$$

$$G_t = \sum_{k=0}^{T} (\gamma)^k R_{t+k+1} + C(1 + \gamma + \gamma^2 \cdots \gamma^T)$$

$$\Rightarrow G_t = G_{t\_old} + \underbrace{\frac{C(1-\gamma^{T+1})}{1-\gamma}}_{V_C(t)}$$

Now the closer t is with T, the smaller will be V_C & vice versa.

Yes, clearly the difference b/w subsequent state's return changes in episodic

$$G_{t+2} = G_{t+2\_old} + \underbrace{\frac{C(1-\gamma^{T-1})}{(1-\gamma)}}_{V_C(t+2)}$$

$$V_C(t) > V_C(t+2); \quad 0 < \gamma < 1 \ \& \ T > 0$$

$$\Rightarrow G_{t+2}(old) - G_{t+1}(old) > G_{t+2} - G_{t+1}$$

5) V^* in terms of Q^* ⇒

$$V^*(s) = \max_{a \in A(s)} Q^{**}(s,a)$$

i.e Optimal value of a state ($V^*(S=s)$) = the action ($a \in A(s)$) for which the max value is obtain from $Q^*(s,a)$