RL Assignment 1 Report

Environment:

Given a racetrack in which red grids represent boundaries and blue, green, orange are valid positions that a car can take in which green, orange are starting and ending positions of a car.

The given diagram can be encoded:-

1. valid grids as “1”
2. red grids as “0”

State is defined with the help of 4 components those are:

1. Car in which row (r)
2. Car in which column (c)
3. Car’s current velocity along horizontal direction (v_x)
4. Car’s current velocity along vertical direction (v_y)

Finally, the State is represented with the tuple of 4 elements (r, c, v_x, v_y).
Where,

- r range is [0,20]
- c range is [0,7]

  Based on actions
    - v_x range is [0,5]
    - v_y range is [0,5]

Assumption considered in this is whenever the car is at blue grid with (v_x =0 , v_y = 0) then assigning v_y = 1 i.e the car moves to the immediate vertical grid and as soon as the car hits the red grid episode ends.

Action is defined with the help of 2 components those are:

1. Change in Velocity in horizontal direction (dv_x) can be [-1, 0, 1]
2. Change in Velocity in vertical direction (dv_y) can be [-1, 0, 1]

Finally, the Action is represented with the tuple of 2 elements (dv_x, dv_y).
Where,

- dv_x can be [-1, 0, 1]
- dv_y can be [-1, 0, 1]

Total of 9 possible actions those are [(-1,-1), (-1,0), (-1,1), (0,-1), (0,0), (0,1), (1,-1), (1,0), (1,1)]

Assigning rewards:

- If car hits boundary “-100”
- If car is on the blue grid “-1”

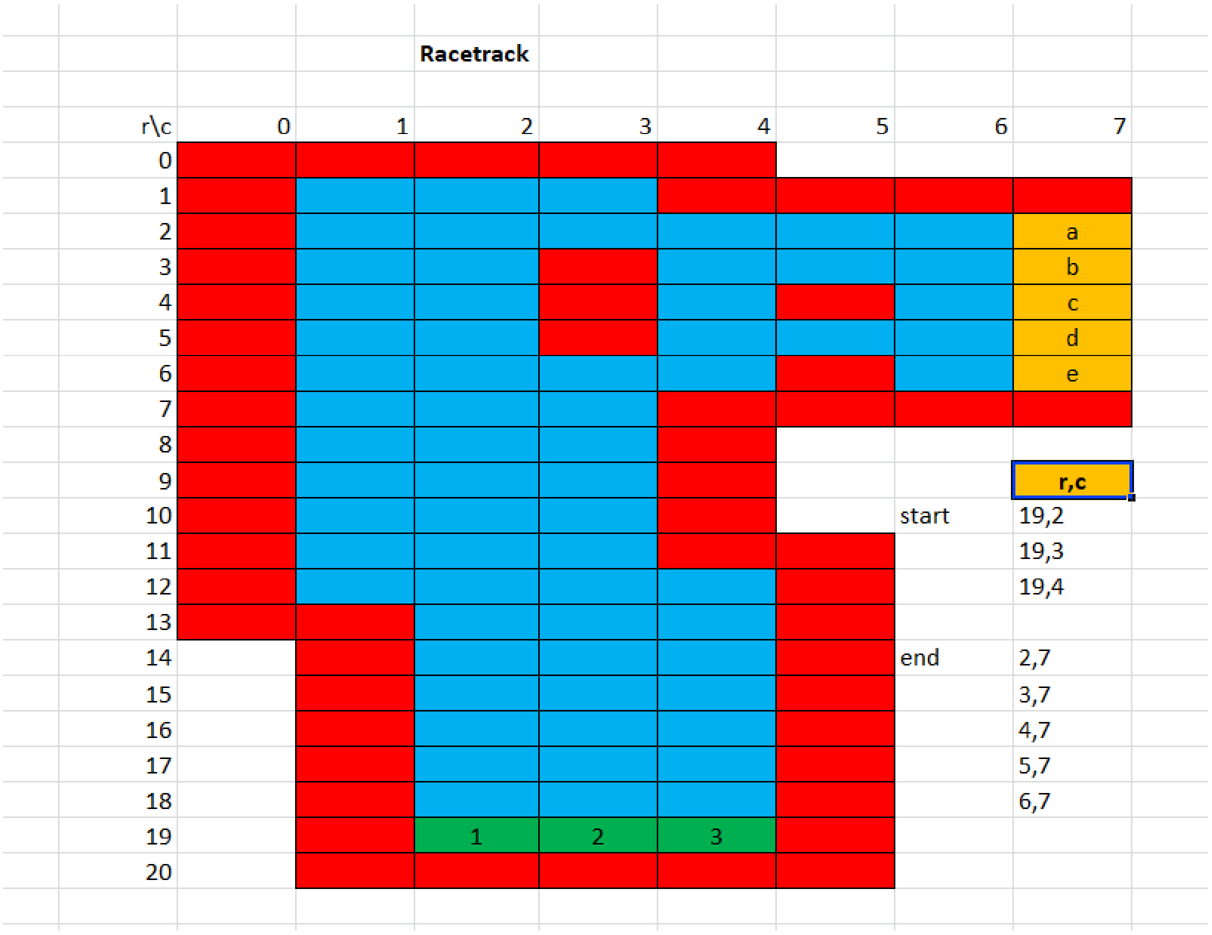
- If car hits orange line “100”

Algorithm to check whether it hit boundary while in the race:

Wrote a function **check_within_track** this detects whether the car in the race track hits the red grid or not.

- Given the coordinates of the current state and a next state.
- If the current (x_1,y_1) and next state (x_2,y_2) are in the different row (x1 does not equals x_2) then using the coordinates of the current and next state find the line equation.
- We are having the set of x values with the difference of ~0.1 from x_2 to x_1 i.e; [x_2, x_2 + 0.1, x_2 + 0.1+0.1,.............., x_1]. From these values we are calculating the corresponding y-values with the use of a line equation found before.
- Finally, checking the points whether they belong to the blue grid or red grid.
- If we find the point belongs to red grid then returning the value “1” and giving reward “-100”
- If we find the point belongs blue grid then returning the value “0” and giving reward “-1”
- If we find the point belongs orange grid then returning the value “2” and giving reward “100”

Race Track:



Racetrack

r\c	0	1	2	3	4	5	6	7
0	🟥	🟥	🟥	🟥	🟥			
1	🟥	🟦	🟦	🟦	🟥	🟥	🟥	🟥
2	🟥	🟦	🟦	🟦	🟦	🟦	🟦	🟧 a
3	🟥	🟦	🟦	🟥	🟦	🟦	🟦	🟧 b
4	🟥	🟦	🟦	🟥	🟦	🟥	🟦	🟧 c
5	🟥	🟦	🟦	🟥	🟦	🟦	🟦	🟧 d
6	🟥	🟦	🟦	🟦	🟦	🟥	🟦	🟧 e
7	🟥	🟦	🟦	🟦	🟥	🟥	🟥	🟥
8	🟥	🟦	🟦	🟦	🟥			
9	🟥	🟦	🟦	🟦	🟥			**r,c**
10	🟥	🟦	🟦	🟦	🟥		start	19,2
11	🟥	🟦	🟦	🟦	🟥	🟥		19,3
12	🟥	🟦	🟦	🟦	🟦	🟥		19,4
13	🟥	🟥	🟦	🟦	🟦	🟥		
14		🟥	🟦	🟦	🟦	🟥	end	2,7
15		🟥	🟦	🟦	🟦	🟥		3,7
16		🟥	🟦	🟦	🟦	🟥		4,7
17		🟥	🟦	🟦	🟦	🟥		5,7
18		🟥	🟦	🟦	🟦	🟥		6,7
19		🟥	🟩 1	🟩 2	🟩 3	🟥		
20		🟥	🟥	🟥	🟥	🟥		

Monte Carlo:

- On-policy:

  Monte Carlo O-Policy is a reinforcement learning technique where an agent interacts with its environment by taking actions based on the policy it is currently following. The agent then gets a reward for what it did, and the pairs of states and actions that led to the reward are saved in memory. This process is repeated over many episodes, which lets the agent build a database of state-action pairs and the rewards that come with them.

  Once the agent has enough information, it can use Monte Carlo simulation to estimate the value of each state-action pair. This is done by taking the average of the rewards for each state-action pair over all the episodes in which that state-action pair happened. Then, these estimates of value can be used to change the policy, making it more likely that the agent will choose actions that lead to higher rewards in the future.

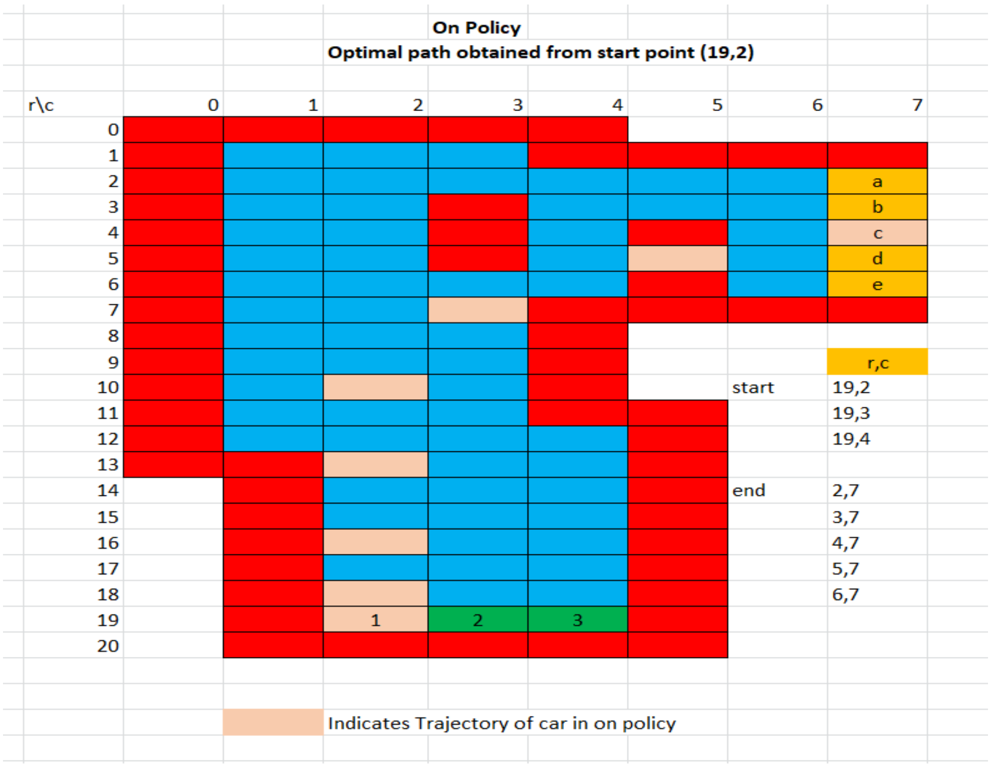
- Off-policy

  Monte Carlo Off-Policy is a reinforcement learning method in which the agent learns from experiences caused by a behaviour policy that is different from the target policy that the agent is trying to optimise. In this method, the agent chooses what to do by following a behaviour policy and gathering a series of experiences. Then, the experiences are used to evaluate the policy and make it better.

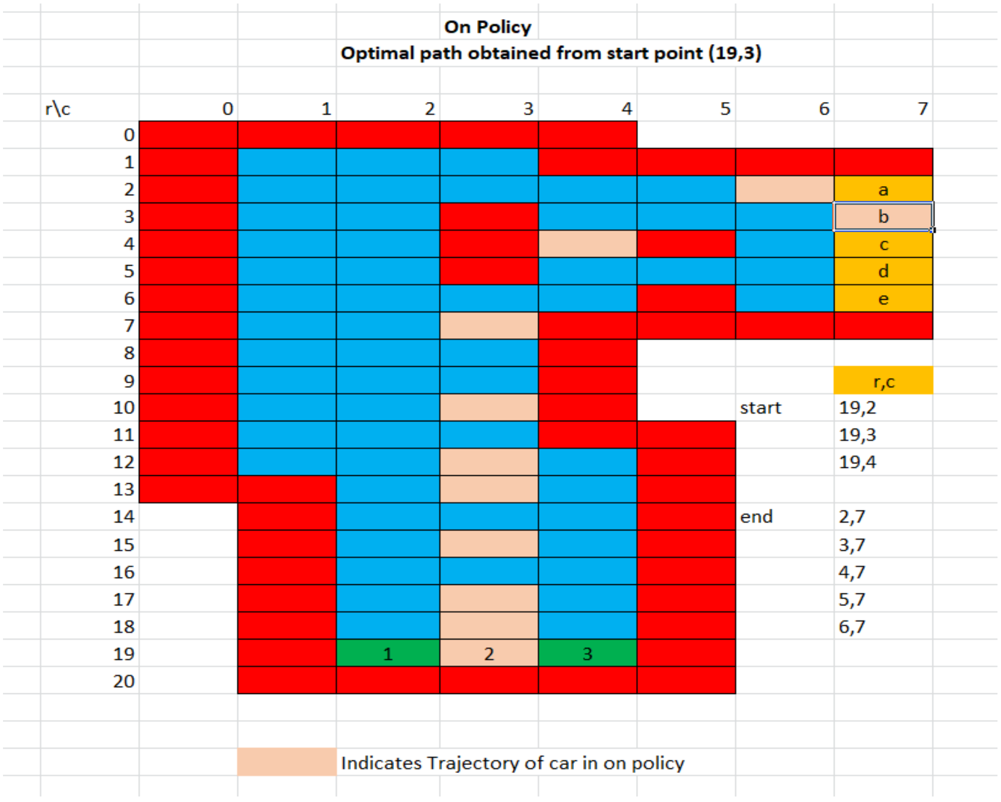
  Monte Carlo Off-Policy is different from Monte Carlo On-Policy in that the agent does not have to follow the target policy. In Monte Carlo On-Policy, the agent learns from the experiences it has when it follows the target policy. Instead, the agent can learn from any policy that looks at the space between states and actions.

On-policy Monte Carlo learns from experience by interacting with the environment using the same policy that is being evaluated, while Off-policy Monte Carlo learns from experience by interacting with the environment using a different policy than the one being evaluated.

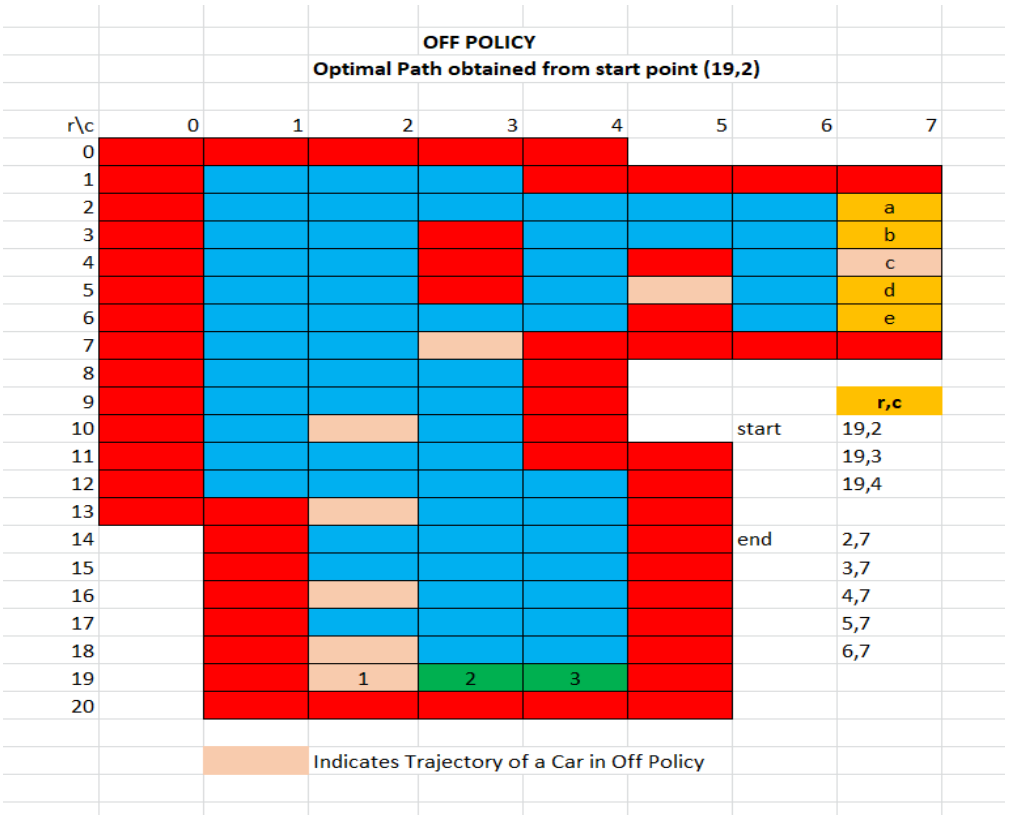
Results:



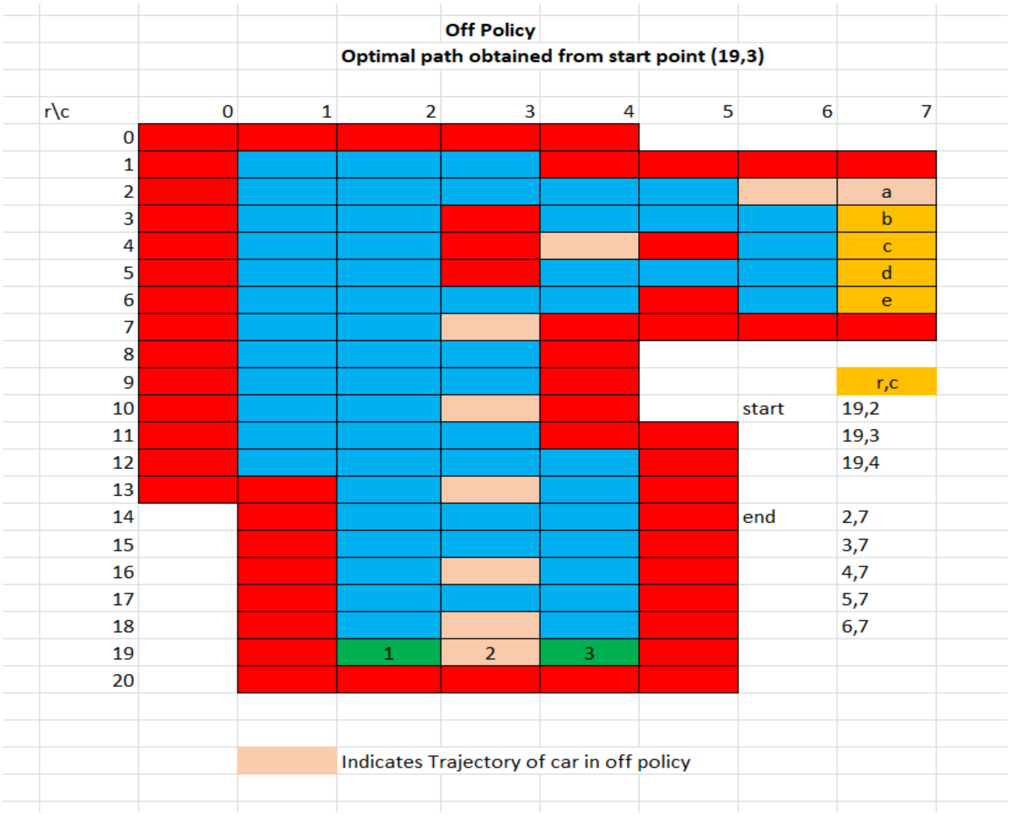
In the given figure the blocks with “biscuit” colour form the optimal path from the starting point 1 to end point c.



In the given figure the blocks with “biscuit” colour form the optimal path from the starting point 2 to end point b.



In the given figure the blocks with “biscuit” colour form the optimal path from the starting point 1 to end point c.



In the given figure the blocks with “biscuit” colour form the optimal path from the starting point 2 to end point b.

Note: With the given conditions from start state (19,4) it never goes to final state in both the methods.

We used two Monte Carlo control algorithms to find the best way to act in this situation. We have used both on-policy and off-policy Monte Carlo on our environment and compared the results.

The on-policy Monte Carlo simulation has shown that the policy keeps getting better over time. It takes about 5,00,000 episodes for the policy to converge on the best one.

The off-policy Monte Carlo simulation has shown that the policy keeps getting better over time. It takes about 3,00,000 episodes for the policy to converge on the best one.

The performance of the algorithm of both methods has been evaluated with a plot of the average return per episode over all the episodes.

Both on-policy and off-policy Monte Carlo methods were able to find a near-optimal policy in the given environment. But it took longer for the on-policy Monte Carlo method to find a solution than the off-policy Monte Carlo method because as I considered, as soon as a car hits the red grid the episode ends.

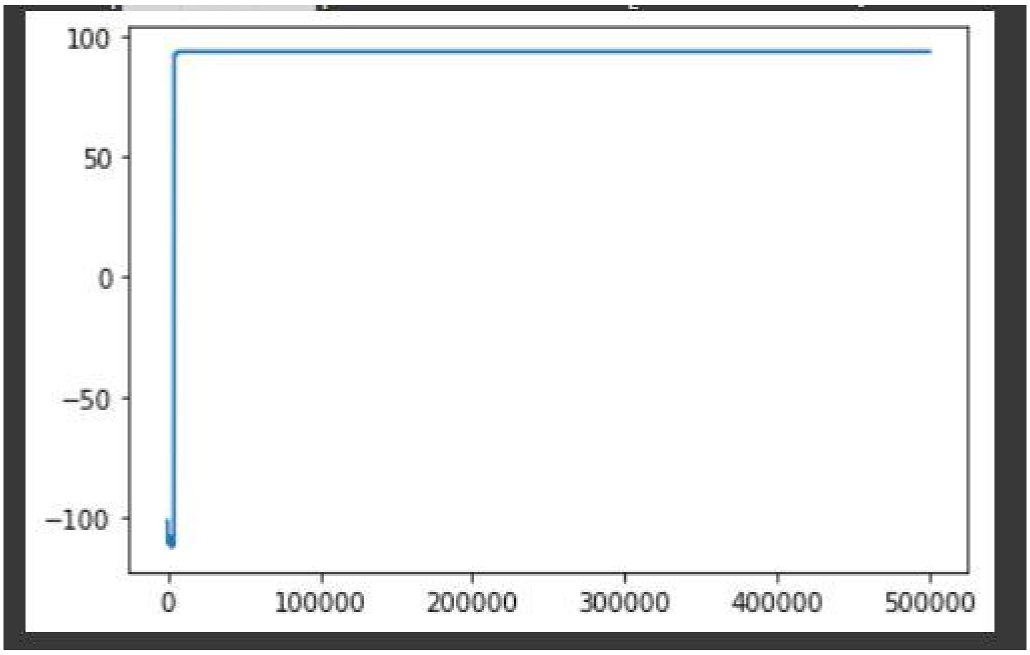


Figure 1: On Policy Convergence plot from (19,2)

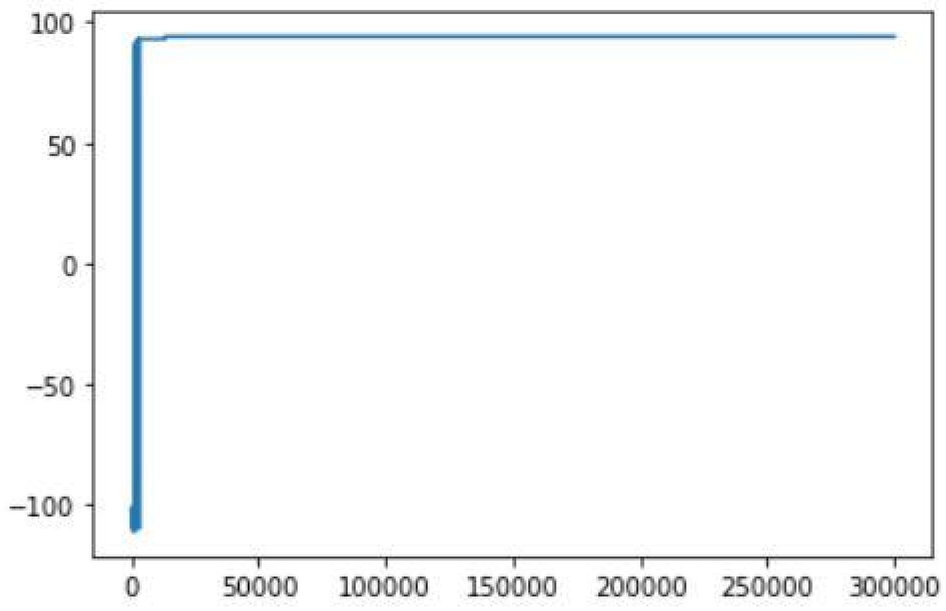


Figure 2: Off Policy Convergence plot from (19,2)

Got optimal path from the starting point (19,2) for the both policies.

How to run?

1. Open my files using Jupyter Notebooks or VScode.
2. Run the all cells