DAILY ONLINE ACTIVITIES SUMMARY

Date:	**21 May 2020**	**Name:**	**Asha Rudrappa Totagi**
Sem& Sec	**6thsem& A sec**	**USN:**	**4AL17CS015**

Online Test Summary

Subject	**Operating System**		
Max. Marks	**30**	**Score**	**30**

Certification Course Summary

Course	**Machine Learning with python**		
Certificate Provider	**Congnitive Class**	**Duration**	**6 hours**

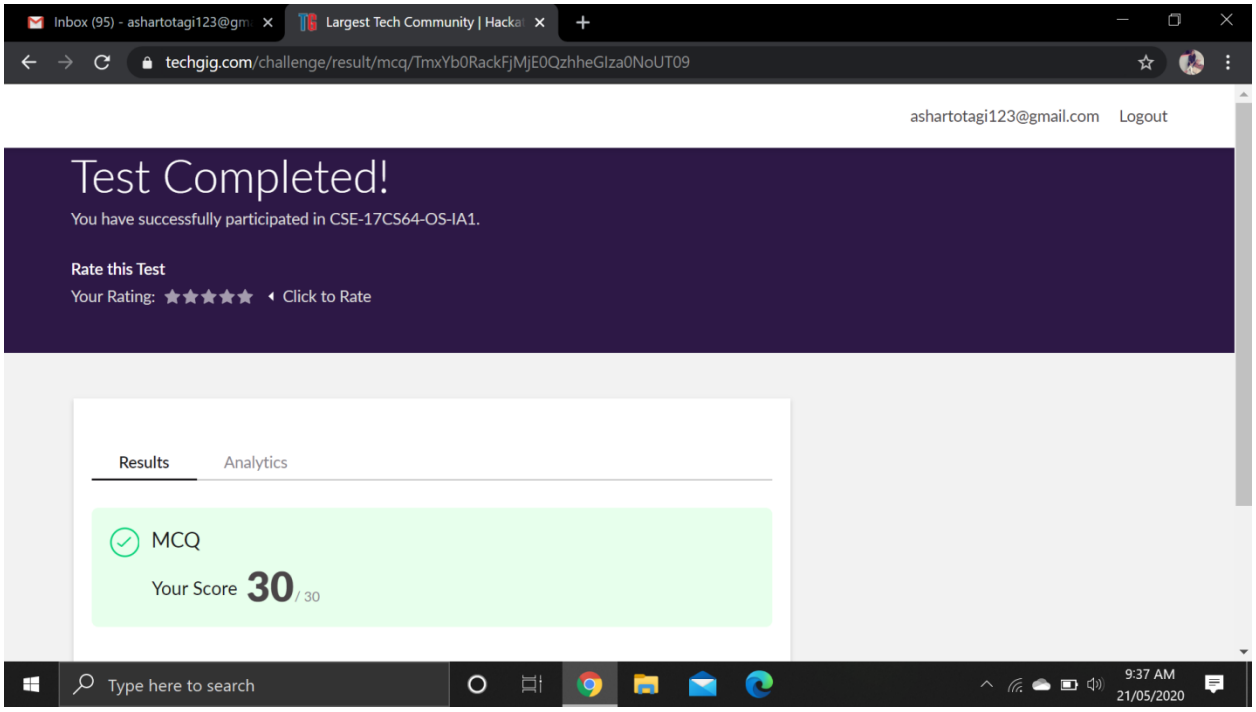
Coding Challenges

Problem Statement
Program 1:
Write a java program to implement round robin scheduling algorithm.Calculate AVG WT AND TAT.
INPUT:NO OF PROCESSES,BURST TIME AND TIME QUANTUM.

Status: DONE

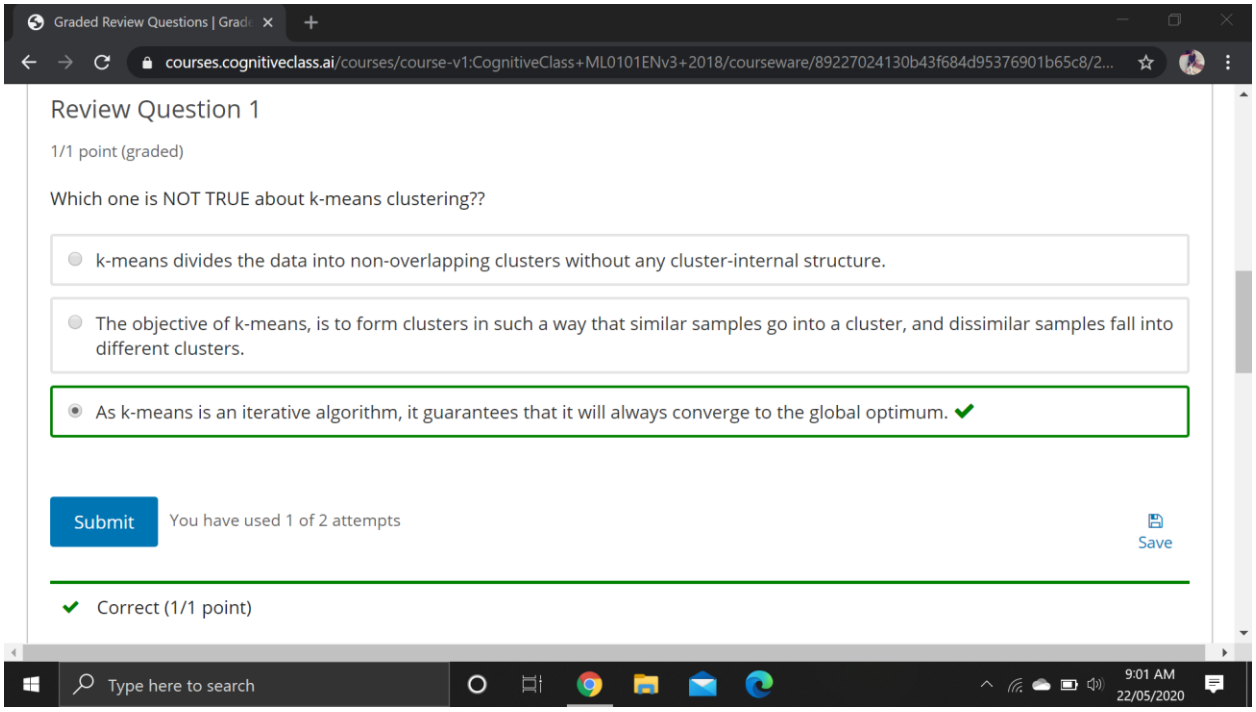
Uploaded the report in Github	**YES**
If yes Repository name	**Daily Status**
Uploaded the report in slack	**YES**

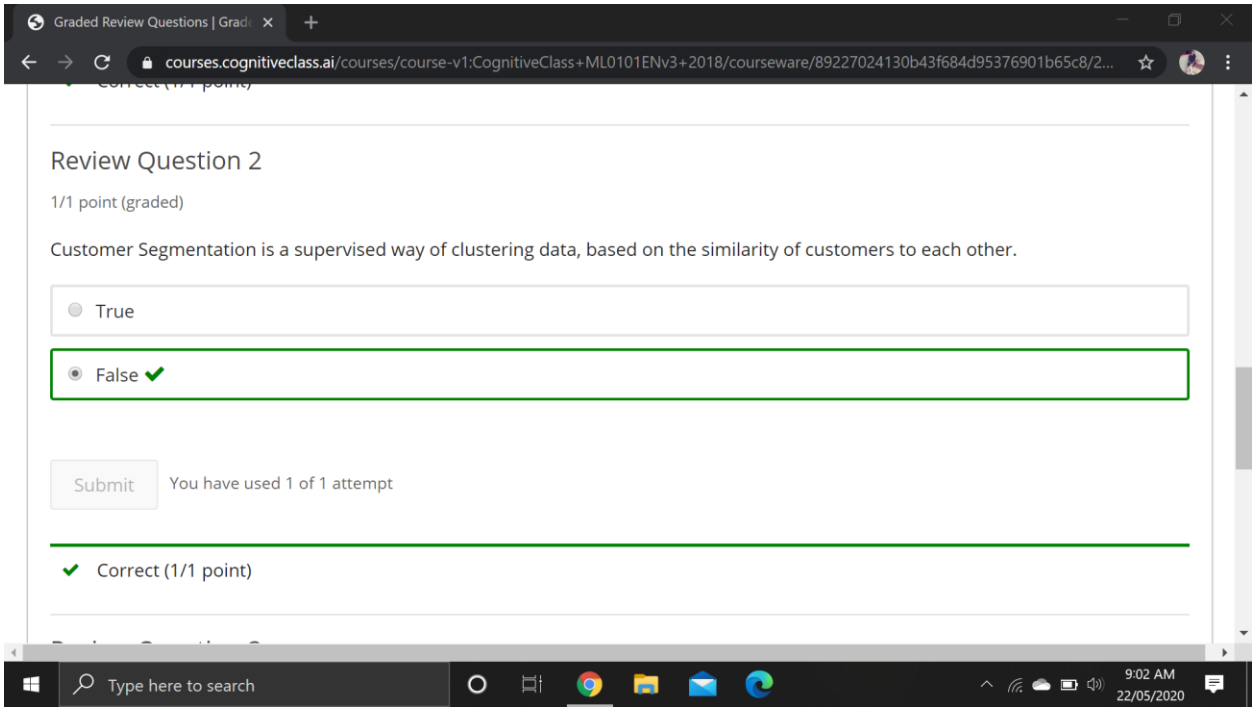
Online Test Details: (Attach the snapshot and briefly write the report for the same)

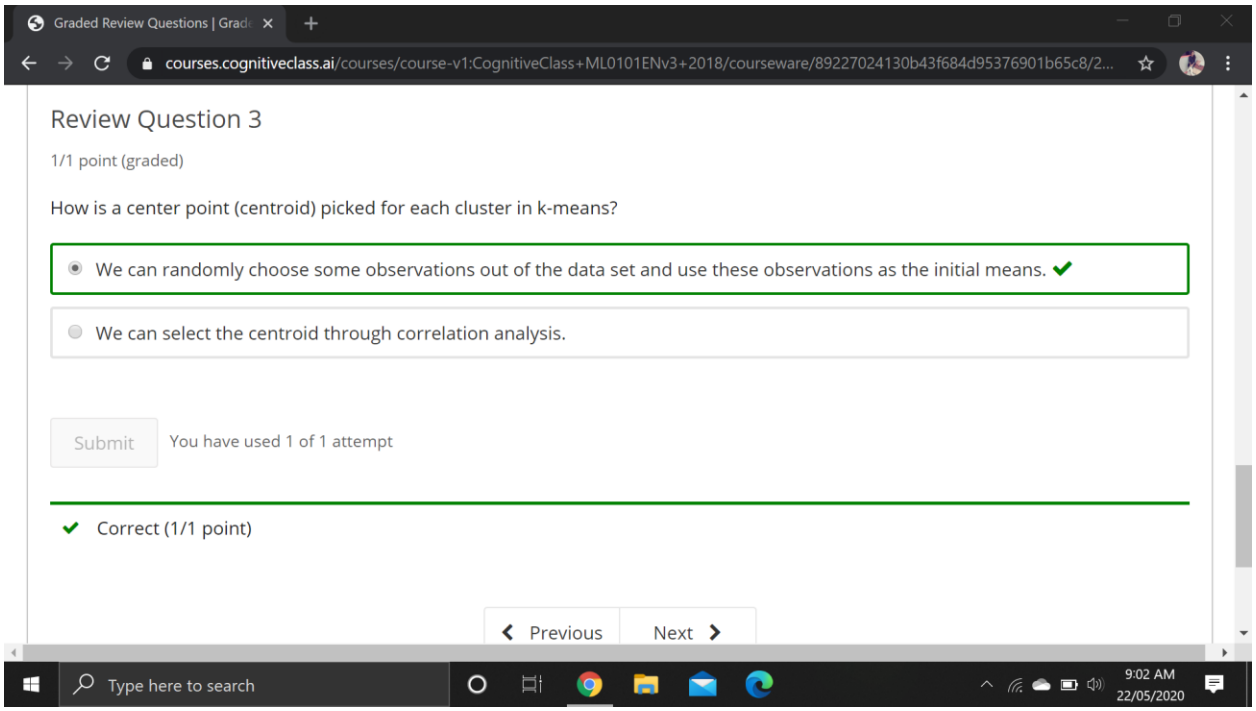


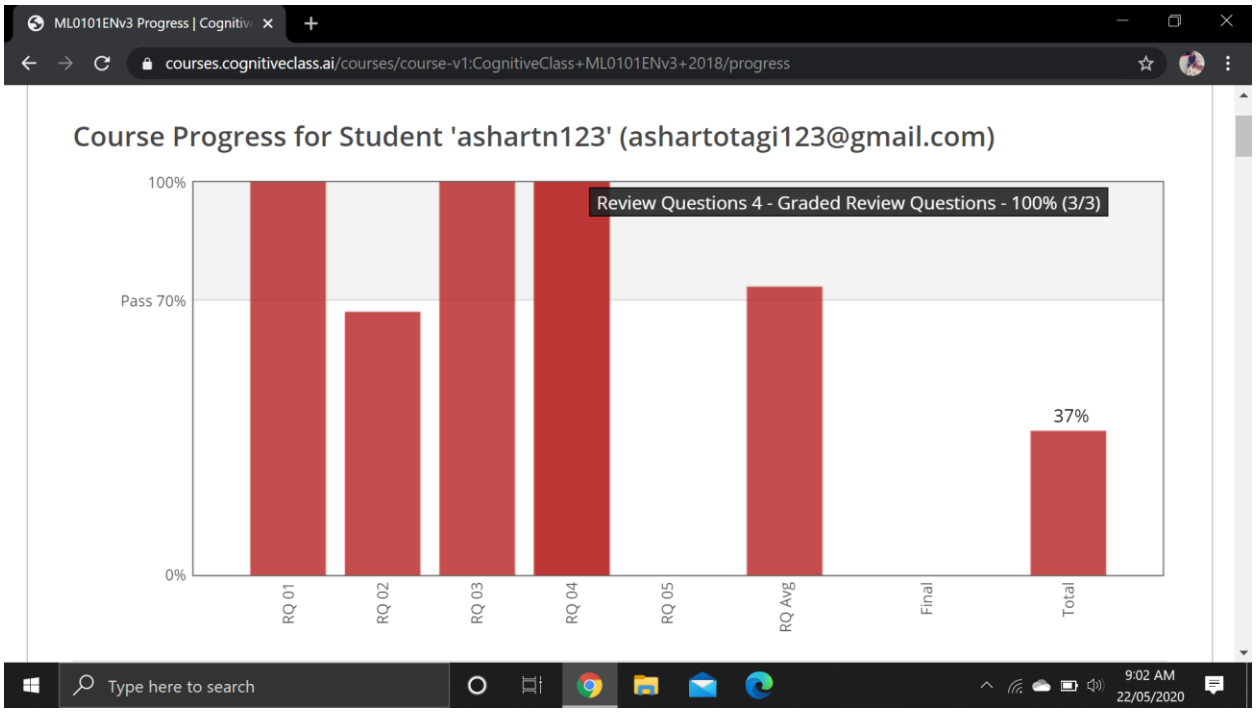
OS IA test was held today i.e, 20 May 2020. There were two rounds where each round carried 21, 9 marks respectively. Out of 30 marks I scored 30

Certification Course Details: (Attach the snapshot and briefly write the report for the same









DAY 1 (20-05-2020)- Introduction to Clustering, K-means Clustering, Hierarchical Clustering, DBSCAN Clustering.

Coding Challenges Details: (Attach the snapshot and briefly write the report for the same)

Program 1:

```
public class GFG
{
   // Method to find the waiting time for all
   // processes
   static void findWaitingTime(int processes[], int n,
             int bt[], int wt[], int quantum)
   {
      // Make a copy of burst times bt[] to store remaining
      // burst times.
      int rem_bt[] = new int[n];
      for (int i = 0 ; i < n ; i++)
         rem_bt[i] =  bt[i];

      int t = 0; // Current time

      // Keep traversing processes in round robin manner
      // until all of them are not done.
      while(true)
      {
         boolean done = true;

         // Traverse all processes one by one repeatedly
         for (int i = 0 ; i < n; i++)
         {
            // If burst time of a process is greater than 0
            // then only need to process further
            if (rem_bt[i] > 0)
            {
               done = false; // There is a pending process

               if (rem_bt[i] > quantum)
               {
                  // Increase the value of t i.e. shows
                  // how much time a process has been processed
                  t += quantum;

                  // Decrease the burst_time of current process
                  // by quantum
                  rem_bt[i] -= quantum;
```

```
                }

                // If burst time is smaller than or equal to
                // quantum. Last cycle for this process
                else
                {
                    // Increase the value of t i.e. shows
                    // how much time a process has been processed
                    t = t + rem_bt[i];

                    // Waiting time is current time minus time
                    // used by this process
                    wt[i] = t - bt[i];

                    // As the process gets fully executed
                    // make its remaining burst time = 0
                    rem_bt[i] = 0;
                }
            }
        }

        // If all processes are done
        if (done == true)
          break;
      }
    }

    // Method to calculate turn around time
    static void findTurnAroundTime(int processes[], int n,
                        int bt[], int wt[], int tat[])
    {
        // calculating turnaround time by adding
        // bt[i] + wt[i]
        for (int i = 0; i < n ; i++)
            tat[i] = bt[i] + wt[i];
}
```