**Shell.ai Hackathon**

There are 3 parts to the problem:

1. Demand Forecast (Time Series)
2. Constraint Satisfaction
3. Optimization
4. **Demand Forecast**: For Forecasting demand, I have used **Autoregression** on the log of the historical demand data. This is because in many demand points, the growth in demand was still exponential in 2018, while in other demand points, it was already tapering. Taking a log flattens the exponential and also handles the situations where the demand is tapering. I did not find other approaches for Time Series forecasting such as ARIMA suitable for this exercise because Moving Averages tend to lag and will only introduce additional error for a exponential curve
5. **Constraint Satisfaction**: Constraints provided in the problem are fairly simple, and hence constraint satisfaction has been built in my code. At any point during execution of my code, constraints are not violated.
6. **Optimization:** For optimization, I have used **stochastic hill climb** algorithm.   
     
   **Step1:** Basically, I maintained a table of all the supply points and added a number of fast changing stations across all the points. The number of **fast charging points** initially added is a ‘*tuning parameter*’ of this algorithm defined by the ‘**NETWORK\_SLACK**’factor. Basically, the idea is that the total charging capacity should be 10-20% more than the total demand to arrive at an optimal overall cost. Each time I executed the overall program, I changed the **NETWORK\_SLACK** to check if I was getting a better solution.   
     
   **Step 2:** Then **DSij** was for each Demand point connecting it to a supply point was computed using a greedy algorithm – basically every Demand Point was to be connected to the nearest Supply Point available, if the nearest Supply Point has been utilized to full capacity, then the Demand Point was connected to the next nearest Supply Point and so on.   
     
   **Step 3:** Then I randomly swapped ‘n’ number of points across (number of points to be swapped provided as a ‘*tuning parameter*’) the Supply Points and recomputed all the **DSi**j to check if I got a more optimal solution. If a more optimal solution was available, then my solution updated to the more optimal choice. This was run multiple times (number of iterations pre-decided as a ‘*tuning parameter*’ ).  
     
   **Step4:** Then I randomly swapped ‘n’ **fast charging points** with **slow charging points** (number of points to be swapped also pre-decided as a ‘*tuning parameter*’) while maintaining the overall supply capacity of the network constant. This would increase the cost of the infrastructure but reduce the cost of customer unsatisfaction as a greater number of charging points will be available now for the same overall network capacity. Hence the overall sum for the cost function is expected to be reduced.  
     
   **Step 2** to **Step 4** was executed several times in an iterative loop until the drop in the sum of the cost function tapered off.