Project Title: “REDUCING SURGE PRICES FOR RIDE HAILING SERVICES”

By

Muktanidhi S Dhotrad, Rohit Khurana and Sandeep Girada

msdhotra@ncsu.edu, rkhuran@ncsu.edu, sigirada@ncsu.edu

Under the supervision of

DR. YUNAN LIU

YINING HUANG

Submitted in Partial fulfillment of the

Requirements of the Course:

MA 548: Monte Carlo Methods for Financial Math

Term: Spring 2019 (Full Semester)

26th April 2019

Table of Contents

1. Introduction

Most of us have used Cab Services like Uber, Lyft and others. Sometimes we have also experienced surge in the prices during rush hours. The reason being, demand of the users requesting the service is more than the supply of drivers/partners providing the service at a time for the specific geographical location.

We obtained data for a specific business problem and were asked to analyze the effect of communication of rush hour locations to drivers beforehand. Our project focuses on quantifying the effect of the business decision and to reduce the overall percentage of surge pricing during rush hours to an acceptable/optimal level.

The data we obtained had driver and rider information for 28 days/4 weeks, total of 58k trips. This data does not reflect the actual logistics of a Ride Hailing Service, in fact it is quite skewed and reflects the challenge the Ride Hailing Service was facing.

We will be using several **Open Models with NHPP (Non-homogeneous Poisson Process) and Simulate the request and driver interaction using Monte Carlo Techniques** to first emulate the logistics of the given data and then to do a sensitivity analysis to show how the surge pricing and Driver Idle Time varies with the model parameters.

1.1 Problem Statement

How to bring down the surge pricing during rush hours to an acceptable level?

1.2 Hypothesis

By communicating to the drivers about the Rush hours timing and geographical location, surge pricing can be brought down. The communication here is passive, i.e. we inform the drivers at the start of the day/week and not during the rush hours/real-time.

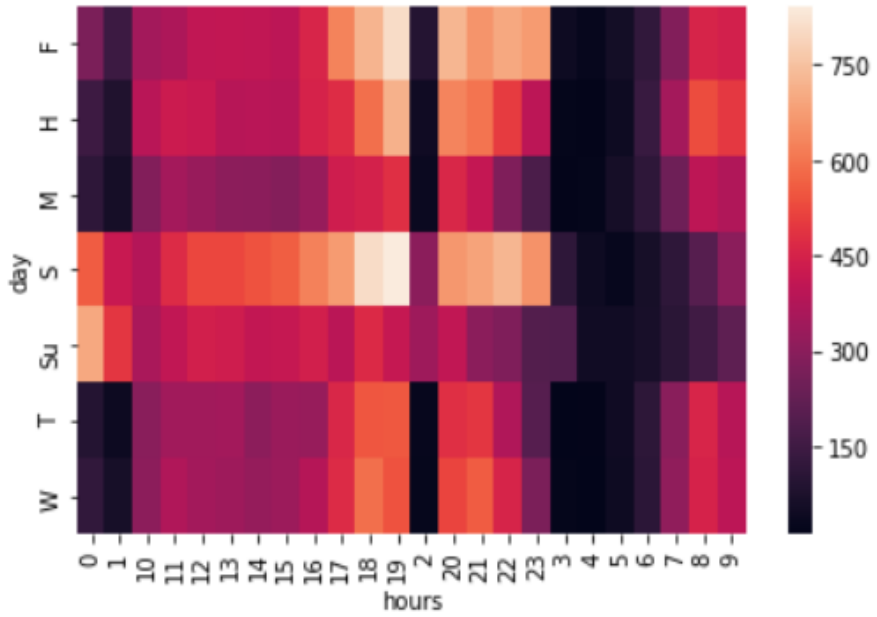
2. Data Analysis

Our dataset has records of 58000 trips spanning over 28 days. The variables in our data are Trip ID, Driver ID, Trip Status, Actual time to arrival, Estimated time to arrival, request time, surge multiplier, driver payout, start geographical location, end geographical location, trip price pre discount and rider payment. The start and end geographical location in our data includes AA, BB, CC and DD.

2.1 Data Exploration, Aggregation & Cleaning

We have classified our data of 58k trips into 168 Categories (24 hours * 7days). According to the following heatmap we can infer that the weekday rush hour comprises of 8:00 am, hours between 5:00 pm and 10:00 pm, and the weekend rush hour comprises of 4:00 pm and 10:00 pm and 12:00 am. Notations used in our heatmap are: M- Monday, T- Tuesday, W- Wednesday, H- Thursday, F- Friday, S- Saturday and Su- Sunday.

The black color indicates supply in able meet the demand, and red color indicates that supply in unable to meet the demand (i.e. shortage of supply).



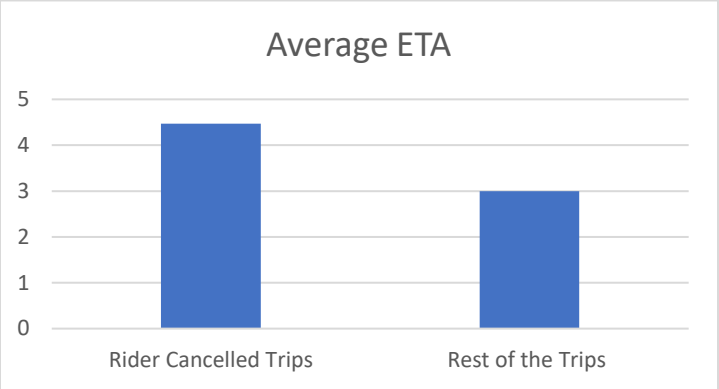
In our project we have chosen the trips starting at “CC” and ending at “AA”, “BB”, “CC” and “DD”. We notice that 73.01% of the trips start and end at “CC” itself.

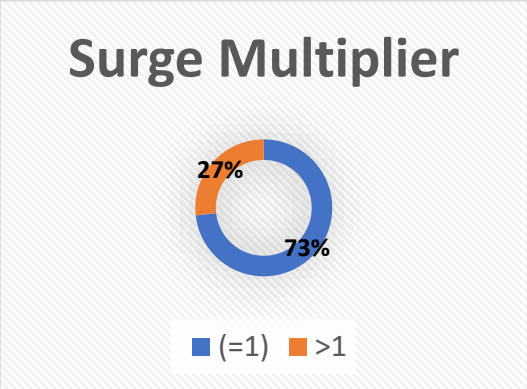
	AA	BB	CC	DD
CC	15.14%	6.12%	**73.01%**	5.74%

2.2 Key Patterns

In our project we have only considered weekday trips covering rush hours from 6pm to 10pm. We observed that 90% of the drivers leave after the first trip and the rest 10% join back the system for the next trip. We also noticed that 99.94% of the times trips are matched and 10% of the trips are cancelled. The trips that were cancelled by riders had an average ETA (estimated time of arrival) more than 4.25 minutes, while the average ETA for the rest of the trips was 3 minutes. In our dataset

we have 27% of the trips having surge multiplier greater than 1 and the rest 73% trips having surge multiplier equal to 1.





2.3 Derived Variables

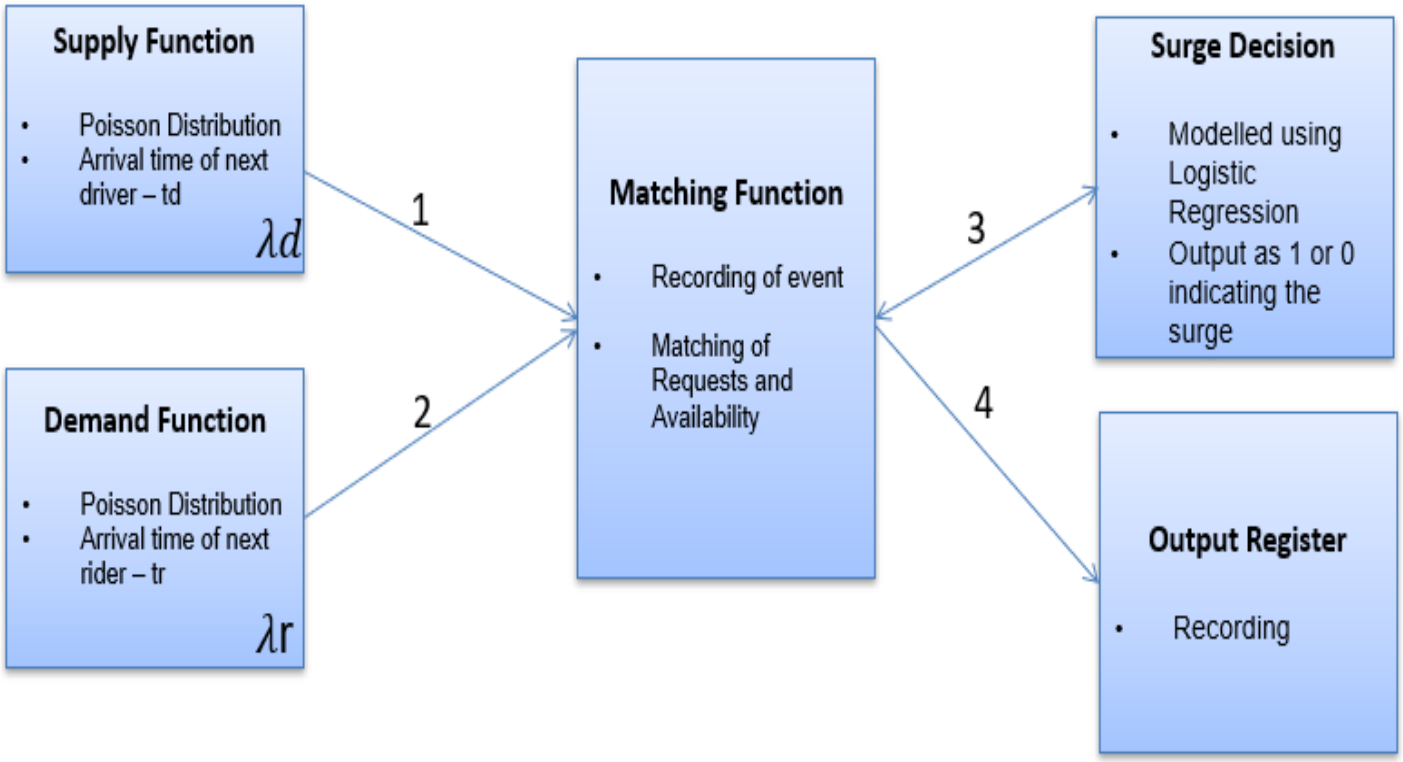
The Supply function generates the arrival time of the driver according to Poisson distribution (λd - Supply rate). Demand function generates the arrival time of the customer/user according to Poisson distribution (λr- Demand rate). We have calculated four supply rates and demand rates for different rush hours (A simplified assumption for the available data).

Rates/ per hour (pm)	**6 to 7**	**7 to 8**	**8 to 9**	**9 to 10**
Supply rate	2.75	2.57	2.12	2.29
Demand rate	3.2	3	2.75	2.32

3. Project Flow

In our project we have defined four functions: Supply Function, Demand Function, Matching Function and Surge Decision.

3.1 Overall Flow Diagram



3.2 Supply Function

The Supply function calls the Supply hours available for that specific hour and extrapolates the rate of arrival for drivers based on the probability of drivers going out of the network.

3.3 Demand Function

The demand function simply generates a rate of arrival of requests based on the requests generated during that specific hour.

3.4 Matching Function

We are using the open network model with p =0.9, i.e. probability of 90 % drivers leaving the system after completion of 1 trip in a day and probability of 10% drivers returning to system.

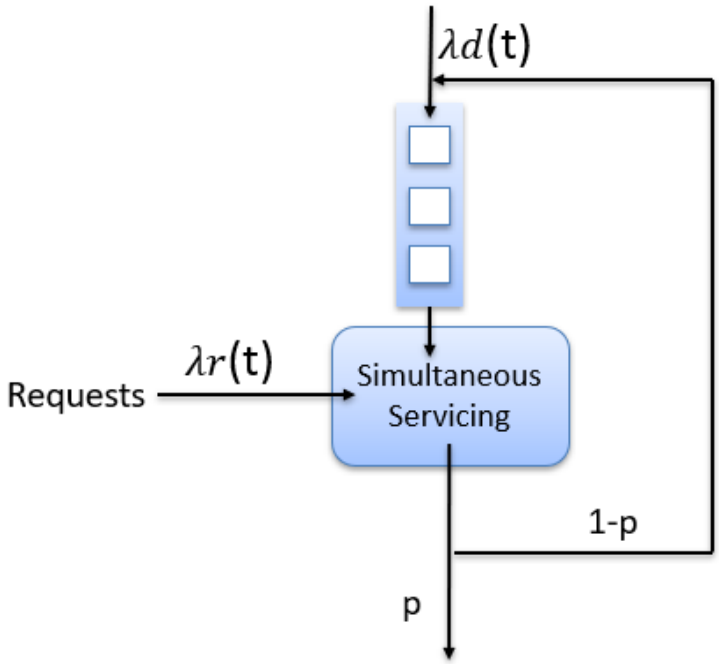


Figure:1 Open Model showing the arrival or requests, drivers and consequent matching.

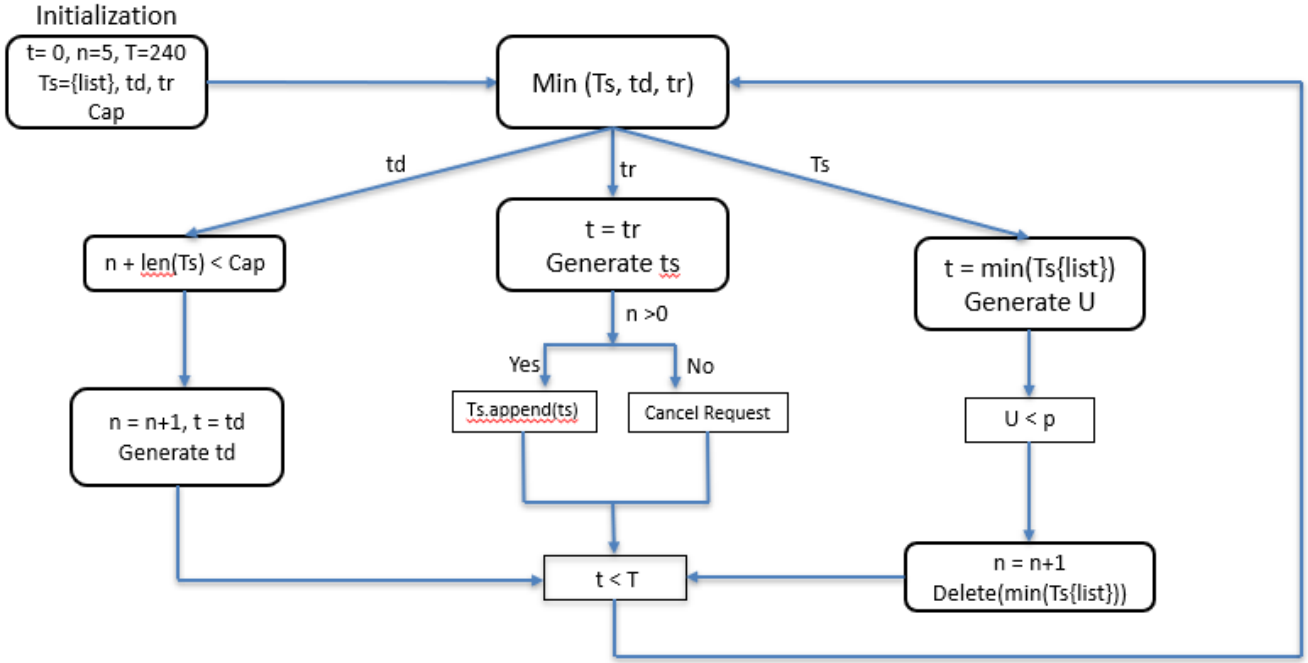


Figure 2: Flow Chart of the Discrete events comprising the Open Model.

Framework of Discrete Event Simulation:

- Variables:
  - t – amount of time that has elapsed, T – End time of simulations, Ts – List of Service times.
  - n – number of idle drivers in the system, Cap – total number of drivers simultaneously servicing the request.
  - p – probability of the driver leaving the system after servicing
- Events:
  - td – arrival time of the next driver, tr – arrival time of the next request, min(Ts) – completion time of the nearest trip request.
- Updating Procedure:
  - Minimum (Ts, td, tr)
- Output Data:
  - Percentage of surge prices
  - Idle Time of drivers in hours
  - Total number of requests
  - Total number of cancellation requests

3.5 Surge Decision

For evaluating the surge, we have first created a logistic regression model using variables - demand (requests) and supply hours (count of drivers) with parameters as below.

```
Optimization terminated successfully.
         Current function value: 0.395191
         Iterations 8
                         Results: Logit
================================================================
Model:              Logit            Pseudo R-squared: -0.205
Dependent Variable: surge            AIC:              534.3463
Date:               2019-04-24 00:35 BIC:              543.3639
No. Observations:   671              Log-Likelihood:   -265.17
Df Model:           1                LL-Null:          -220.10
Df Residuals:       669              LLR p-value:      1.0000
Converged:          1.0000           Scale:            1.0000
No. Iterations:     8.0000
----------------------------------------------------------------
                 Coef.  Std.Err.    z    P>|z|   [0.025  0.975]
----------------------------------------------------------------
supply_hours    -0.1251   0.0248 -5.0453 0.0000 -0.1737 -0.0765
requests         0.0714   0.0115  6.1931 0.0000  0.0488  0.0940
================================================================

Accuracy of logistic regression classifier on test set: 0.90
              precision    recall  f1-score   support

           0       1.00      0.09      0.17        22
           1       0.90      1.00      0.95       180
```

Further, based upon the supply and demand, surge decision function responds with following-

- 1- Surge,
- 0- No Surge

4. Experiment Results

4.1 Case A

If the rate of arrivals of drivers is constant, we have simulated the conditions with various combination of capacity and probability values (of driving staying back in the system) and our results are as below.

Fac=1				
Cap	p = 0.9	p = 0.8	p = 0.75	p = 0.7
15	**0.63**	0.6	0.587	0.566
16	0.61	0.573	0.552	0.523
17	0.588	0.535	**0.449**	**0.46**
18	0.566	**0.5**	**0.46**	**0.423**

From above, initially **63% of trips during rush hours, had surge and with increase in supply under control conditions, surge can be brought down at the expense of increased Idle Time for the Drivers.** Further assuming out target level for surge is 50%, highlighted blocks in table showcases that this can be achieved with some of the combinations of conditions.

4.2 Case B

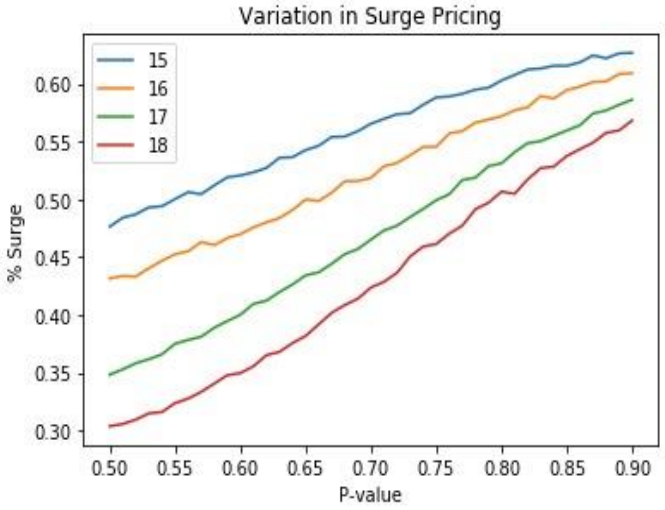
If the rate of arrival of drivers is to be increased by 10%, results of the simulation are as below.

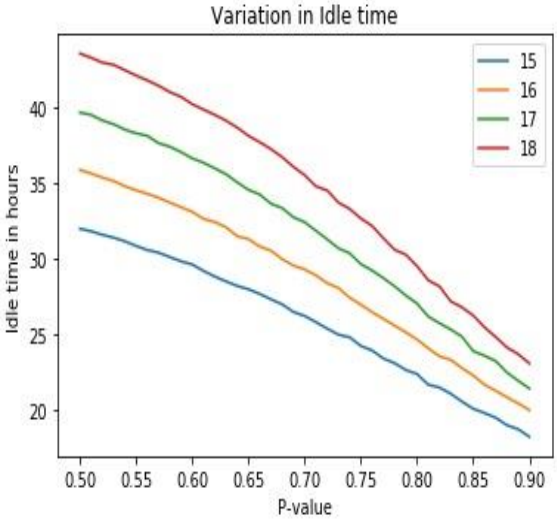
Fac=1.1				
Cap	p = 0.9	p = 0.8	p = 0.75	p = 0.7
15	**0.61**	0.578	0.545	0.52
16	0.57	0.54	**0.48**	**0.44**
17	0.56	0.51	**0.45**	**0.41**
18	0.53	**0.49**	**0.46**	**0.374**

From above, initially **61% of trips during rush hours, had surge and with increase in supply under control condition, surge can be brought down at the expense of Increased Idle Time.** Further assuming out target level for surge is 50%, highlighted blocks in table showcases that this can be achieved with some of the combinations of conditions.

4.3 Reduction of Surge Pricing at Cost of Idle Time

We have further evaluated that with **decrease in surge pricing results in increase of idle time**, as evident from below. For the same simulation, with more drivers motivated to stay in the system surge is going down but at the same time from graph on the right, drivers are spending more time idling.





4.4 Recommendations

We plan to passively communicate with drivers about the rush hours and geographical location to bring down the count of surge pricing.

5. Future Enhancements

- Model uses lot of assumption, that needs to be validated.
- The sensitivity analysis for these parameters were done by keeping the others constant but in reality, the parameters change simultaneously and this behavior needs to be understood mathematically.
- Psychological behavior of the partner/ drive does play a role and that requires attention.
- Possibility of breach of Demand & Supply Equilibrium to be studied.

6. References

Prof. Yunan Liu's lecture notes – Topic 5: Discrete Event Simulations

Author – Sheldon M. Ross, Article Title – Simulation, Fourth Edition
https://dl.acm.org/citation.cfm?id=1197255

7. Appendix – Code

```
# We are not providing actual functions for our data or the data analysis.
import numpy as np
import matplotlib.pyplot as plt
# notice the use of a simple step function for the nonhomgeneous arrival rates.
def supply_func():
    #Dummy function, include your own Supply function here.

    return([2.75,2.57,2.12,2.29])

def demand_func():
    #Dummy function, include your own Demand function here.
    return([3.2,3,2.75,2.32])

def surge_func(Supply,Demand):
    #Dummy function for a simple Surge Pricing situation.
    if Supply <= Demand :
        x=1
    else:
        x=0
    return(x)

# Specific to the Simulation and results shown in the report.
# fac is the multiplication factor that alters the rate of arrival of drivers.
# p is the probability of the driver leaving the system after trip completion
# cap is the maximum number of drivers simulatneously serving the location.
def Third_Sim(fac=1,p=0.9,cap=15):
    # Initialization of arrival rates of requests and drivers per minute.
    lam_rider_arriv=np.array(demand_func())
    lam_driver_arriv=fac*np.array(supply_func())
    # n is the number of requests actively being served
    n=0
    # k is the number of drivers available to be matched.
    k=5
    t=0
    Time_line=[]
    # 4 hour simulation
    T=240
    # 10 customers actively being served before start of simulation.
    Dummy_Cust=np.sort(-np.log(np.random.uniform(size=10))/0.5)
    Time_Serv=np.append([1000],Dummy_Cust)
    #Generate arrival time of driver based on the hour.
    t_d_arr=-np.log(np.random.uniform())/lam_driver_arriv[0]
    #Generate arrival time of requests based on the hour
    t_r_arr=-np.log(np.random.uniform())/lam_rider_arriv[0]
    # Cancelled requests due to Supply Shortage
    Nc=0
    # Count of Requests serviced.
    Ns=0
    # Count of Drivers who actively joined the system.
    Nb=k
```

```
    # Requests Count
    Nr=0
    #Simultaneous Customers being served for plotting purposes
    Serv_cust=[]
    #Cancellations list for plotting purposes
    Cancel_number=[]
    #Simultaneously active drivers for plotting purposes
    Driver_count=[]
    #Surge storage for plotting purposes
    Surge=[]
    #Simultaneously idle drivers storage for plotting purposes
    Idle_Driver=[]
    # Start of Simulation
    while(t<T):
        #To move in the smallest time step for all generated events times.
        if(t_d_arr<min(t_r_arr,Time_Serv.min())):
            #Check of supply cap to accept or reject driver arrivals
            if(k+len(Time_Serv)-1<cap):
                Nb=Nb+1
                k=k+1
            #Moving the time clock forward.
            t=t_d_arr
            i=np.int(np.floor(t/60))
            if i>3:
                i=3
            #Generating next driver arrival.
            t_d_arr=t-np.log(np.random.uniform())/lam_driver_arriv[i]

        elif(t_r_arr<Time_Serv.min()):
            Nr=Nr+1
            i=np.int(np.floor(t/60))
            if i>3:
                i=3
            #Moving the time clock forward.
            t=t_r_arr
            #Generating next request arrival.
            t_r_arr=t-np.log(np.random.uniform())/lam_rider_arriv[i]

            if(k>0):
                k=k-1
                n=n+1
                #Generating Service time
                ts=t-np.log(np.random.uniform())/0.5
                Time_Serv=np.append(Time_Serv,ts)
                #Checking for Surge based on the Surge function.
                Surge.append(surge_func(k,len(Time_Serv)-1))
            else:
              #Cancelled request due to driver shortage
#We had other models but this specific assumption mimics the dummy case.
                Nc=Nc+1
```

```
        else:
            #Moving the time clock forward
            t=Time_Serv.min()
            Ns=Ns+1
            n=n-1
            Time_Serv=np.delete(Time_Serv,np.argmin(Time_Serv))
            y=np.random.uniform()
            #Checking if the driver is going to remain or go out.
            if(y>p):
                k=k+1
        Time_line.append(t)
        Serv_cust.append(len(Time_Serv)-1)
        Cancel_number.append(Nc)
        Driver_count.append(k+len(Time_Serv)-1)
        Idle_Driver.append(k)

    Time_line=[0]+Time_line
    Time_line=np.diff(np.array(Time_line))
    Idle_Driver=np.array(Idle_Driver)
    return(sum(Surge)/Nr,sum(Time_line*Idle_Driver)/60,Nr,Nc)

# Simulations for sensitivity analysis
Cancellations=[]
Estimated_Requests=[]
Estimated_Idle_time=[]
Surge_Cap=[]

#1000 Simulations for checking is the Expected Value mimics data
for i in range(0,1000):
    Su,Idle_Time,Rec,Canc=Third_Sim(fac=1,p=0.9,cap=15)
    Surge_Cap.append(Su)
    Estimated_Idle_time.append(Idle_Time)
    Estimated_Requests.append(Rec)
    Cancellations.append(Canc)
#Expected Value
S_C_m=np.mean(Surge_Cap)
#62.95%
E_T_m=np.mean(Estimated_Idle_time)
#18.21 hours
E_R_m=np.mean(Estimated_Requests)
#678.4
C_m=np.mean(Cancellations)
#84 or 12.38% of the requests

#Standard error
S_C_se=np.std(Surge_Cap,ddof=1)
#0.04611
E_T_se=np.std(Estimated_Idle_time,ddof=1)
#2.5581
E_R_se=np.std(Estimated_Requests,ddof=1)
#26.547
C_se=np.std(Surge_Cap,ddof=1)
```

```
#0.04611

#Empirical Quantiles
np.quantile(Surge_Cap,q=[0.025,0.975])
#[0.53613, 0.71467]
np.quantile(Estimated_Idle_time,q=[0.025,0.975])
#[13.4937, 23.2405]
np.quantile(Estimated_Requests,q=[0.025,0.975])
#[629. , 731.025]
np.quantile(Cancellations,q=[0.025,0.975])
#[ 41.  , 136.05]

# Variance reduction can be used but we know that the sample variance is quite
#less and Expected Value estimates are very accurate.

# Sensitivity Analysis by varying one parameter at a time
pi=np.linspace(start=0.5,stop=0.9,num=41)
Surge_Cap=[]
Idle_time_Cap=[]
for k in [15,16,17,18]:
    Idle_time_pi=[]
    Surge_pi=[]
    for j in pi:
        Estimated_Idle_time=[]
        Surge_ratio=[]
        for i in range(0,1000):
            Su,ID,X,Y=Third_Sim(fac=1,p=j,cap=k)
            Estimated_Idle_time.append(ID)
            Surge_ratio.append(Su)

        Surge_pi.append(np.mean(Surge_ratio))
        Idle_time_pi.append(np.mean(Estimated_Idle_time))

    Surge_Cap.append(Surge_pi)
    Idle_time_Cap.append(Idle_time_pi)


# Plots for Idle Time with varing P-value and Suppy Cap
plt.plot(pi,Idle_time_Cap[0],label="15")
plt.plot(pi,Idle_time_Cap[1],label="16")
plt.plot(pi,Idle_time_Cap[2],label="17")
plt.plot(pi,Idle_time_Cap[3],label="18")
plt.ylabel("Idle time in hours")
plt.xlabel("P-value")
plt.title("Variation in Idle time")
plt.legend()
plt.show()

# Plots for Surge ratio with varing P-value and Suppy Cap
plt.plot(pi,Surge_Cap[0],label="15")
plt.plot(pi,Surge_Cap[1],label="16")
plt.plot(pi,Surge_Cap[2],label="17")
```

```
plt.plot(pi,Surge_Cap[3],label="18")
plt.ylabel("% Surge")
plt.xlabel("P-value")
plt.title("Variation in Surge Pricing")
plt.legend()
```