시뮬레이션 최종 보고서

고려대학교 과학도서관의 엘리베이터 운영 정책 제안

5조

2020170845 유혜원

2020170830 조혜윤

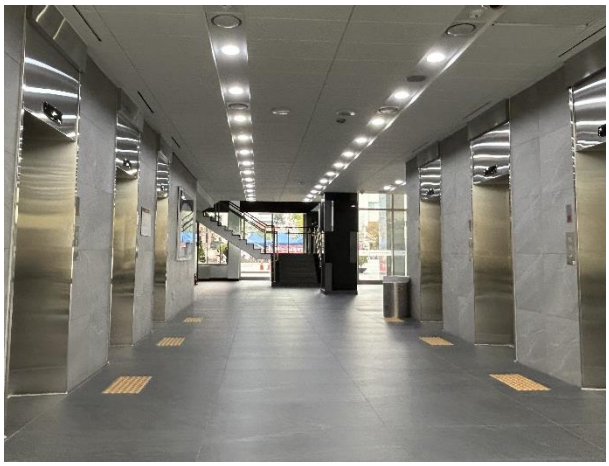
2021170843 최아정

[Table of Contents]

1. Introduction

1.1. Target System Description

고려대학교 과학도서관 건물은 B1층부터 6층까지로 이루어져 있으며, 많은 교직원과 학생들이 방문한다. 방문객 수가 많은 만큼 엘리베이터는 총 6대가 존재하며 각 엘리베이터는 B1층부터 6층까지 전층 운행을 한다. 6대의 많은 수의 엘리베이터가 존재함에도 불구하고 수업시간 전 후로 사람이 몰려 엘리베이터를 타기 위해 줄을 서며, 엘리베이터를 기다리는 시간이 증가하는 경우가 빈번히 발생하고 있는 상황이다.



[그림1] 고려대학교 과학도서관 1층

1.2. Background

여러 대의 엘리베이터를 운용하는 많은 건물들은 건물의 특징과 이용자 특성에 따라 다양한 운행 정책을 채택하고 있다. 이는 이용자의 대기 시간과 편의성, 그리고 효율성에 크게 영향을 미치고 이에 따라 전력 소모량이 달라진다.

이때 대표적으로 선택 가능한 엘리베이터 운행 정책으로는 전층 운행/홀수, 짝수층 운행/저층, 고층 운행 등이 있으며 운용하고 있는 엘리베이터 수에 따라 일부는 전층 운행을 하고 일부는 조건부 운행을 하는 등으로 설정할 수 있다. 각 옵션에 따른 효과는 다음과 같이 예상할 수 있다.

먼저 모든 엘리베이터가 전층 운행을 하는 경우, 사용자는 가장 빨리 내려오는 엘리베이터에 탑승하면 되기에 웨이팅 타임이 줄어들지만 엘리베이터 탑승 후 목적 층에 도달하기 전까지 정차하는 횟수가 많기 때문에 서비스 타임이 길어지게 된다.

반대로, 일부 층만 운행하는 방식을 채택한다면, 사용자가 엘리베이터를 탑승하기 전까지 기다리는 웨이팅 타임은 증가할 수 있지만 정차 횟수의 감소로 서비스 타임의 감소가 일어난다.

따라서 본 프로젝트를 통해 최적의 엘리베이터 운행 정책을 도출함으로써 이용자의 대기 시간 및 서비스 시간을 최소화하고, 전력 낭비를 줄일 수 있는 솔루션을 제시하고자 한다.

1.3. Study Objectives

본 프로젝트에서 집중하고자 하는 대상은 고려대학교 과학도서관에 위치한 6대의 엘리베이터이다. 과학도서관 이용자들의 이동 패턴을 파악하고 이에 따라 6대의 엘리베이터 각각을 어떠한 정책으로 운영하는 것이 최적인지 시뮬레이션을 통해 검증하고자 한다. 사용자가 엘리베이터의 도착을 기다리는 시간을 웨이팅 타임으로 정의하고, 엘리베이터를 타고 목적층까지 이동하는 시간을 서비스 타임으로 정의하였으며 최종 목표는 웨이팅 타임과 서비스 타임을 합친 총 이용 시간의 평균을 최소화하는 것이다.

2. Data Analysis

2.1. Data 수집 방법

측정 시간대	13:15~13:30, 14:45~15:00	
측정 단위	초	
측정 값	Interarrival time	고객 도착 시간
	Waiting time	버튼을 누른 시간
		버튼을 누른 엘리베이터의 1층 도착 시간
	Service time	1층 출발 시간
		각 층별 도착 시간
		엘리베이터가 한 층을 올라가는 시간
		엘리베이터가 한 층을 지나가는 시간
	Demand	각 층별 하차 인원

	A	B	C	D	E	F	G	H	I
		날짜	1. Inter-arrival time of customers			2. Waiting time (버튼~엘리베이터 1층 도착까지 시간)			
	index		수집 시작 시간	interarrival time		버튼 누른 시간	엘리베이터 도착 시간	소요시간	
	<Example>	11월 1일	1:30:00	0:05:00		1:35:00	1:38	0:03	
	<Example>		1:35:00	0:00:00		1:40:00	1:42	0:02	
	<Example>		1:35:00			1:45:00	1:50	0:05	
1	1	11-14 13:15	1:23:29	0:00:00	11-14 14:45	49:30:00	49:40:00	0:10:00	11-14 13:15
2	2	11-14 13:15	1:23:29	0:00:03	11-14 14:45	49:50:00	49:55:00	0:05:00	
3	3	11-14 13:15	1:23:32	0:00:15	11-14 14:45	49:20:00	51:08:00	1:48:00	
4	4	11-14 13:15	1:23:47	0:00:01	11-14 14:45	51:08:00	51:26:00	0:18:00	
5	5	11-14 13:15	1:23:48	0:00:05	11-14 14:45	51:32:00	51:41:00	0:09:00	
6	6	11-14 13:15	1:23:53	0:00:09	11-14 14:45	51:49:00	52:28:00	0:39:00	
7	7	11-14 13:15	1:24:02	0:00:01	11-14 14:45	52:45:00	53:13:00	0:28:00	
8	8	11-14 13:15	1:24:03	0:00:13	11-14 14:45	53:30:00	53:30:00	0:00:00	
9	9	11-14 13:15	1:24:16	0:00:01	11-14 14:45	53:39:00	54:23:00	0:44:00	
10	10	11-14 13:15	1:24:17	0:00:15	11-14 14:45	54:23:00	54:57:00	0:34:00	
11	11	11-14 14:45	1:24:32	0:00:04	11-14 14:45	54:57:00	55:30:00	0:33:00	
12	12	11-14 14:45	1:24:36	0:00:01	11-14 14:45	55:42:00	55:57:00	0:15:00	
13	13	11-14 14:45	1:24:37	0:00:04	11-14 14:45	56:03:00	56:14:00	0:11:00	11-14 14:45
14	14	11-14 14:45	1:24:41	0:00:06	11-16 13:15	18:30:00	19:12:00	0:42:00	
15	15	11-14 14:45	1:24:47	0:00:12	11-16 13:15	19:47:00	19:53:00	0:06:00	
16	16	11-14 14:45	1:24:59	0:00:11	11-16 13:15	20:16:00	20:29:00	0:13:00	
17	17	11-14 14:45	1:25:10	0:00:01	11-16 13:15	21:23:00	21:27:00	0:04:00	
18	18	11-14 14:45	1:25:11	0:00:10	11-16 13:15	21:32:00	21:46:00	0:14:00	
19	19	11-14 14:45	1:25:21	0:00:03	11-16 13:15	21:58:00	22:11:00	0:13:00	
20	20	11-14 14:45	1:25:24	0:00:13	11-16 13:15	22:24:00	23:07:00	0:43:00	
21	21	11-14 14:45	1:25:37	0:00:13	11-16 13:15	22:26	23:08:00	0:42:00	
22	22	11-14 14:45	1:25:50	0:00:03	11-16 13:15	23:59:00	24:15:00	0:16:00	
23	23	11-14 14:45	1:25:53	0:00:11	11-16 13:15	24:30:00	25:09:00	0:39:00	
24	24	11-14 14:45	1:26:04	0:00:24	11-16 13:15	25:25:00	25:38:00	0:13:00	

	I	J	K	L	M	N	O	P	Q	R	S	T	U
		3. Service Time						4. Demand Count					
		2F 도착시간	3F 도착시간	4F 도착시간	5F 도착시간	6F 도착시간		2F 하차인원	3F 하차인원	4F 하차인원	5F 하차인원	6F 하차인원	
			1:36		1:40			3		4			
			1:41	1:42				2	1				
						1:52					2		
	11-14 13:15			17.42						3			3
				15.72	31.25	54.71				2	1	1	4
				15.72	35.27	57.15				3	2	1	6
		7.12						1					1
		9.41						1					1
				16.54	32.65					1	1		2
				25.08	40.39					3	1		4
				20.35		39.91				2		1	3
				20.23		41.03				1		1	2
				26.9	47.85					7	1		8
						30.75						1	1
				18.98	36.84					1	2		3
	11-14 14:45	12.96			36.49						6		6
			23.96	44.06					1	2			3
			23.41	42.86	66.76				1	4	3		8
				23.38	41.71					1	2		3
				25.1	41.61					2	2		4
				26.88		53.21				1		1	2
		25.23		42.56	64.64	84.3				4	3	1	8
				29.32	48.59					2	4		6
		38.82	60.43	90.98				1	2	3			6
			23.28	39.23	57.99	75.1			1	1	1	1	4
				39.28						3			3

2.2. Input Data

시뮬레이션에서 사용하는 데이터의 종류로는 Interarrival time, Waiting time, Service time (Stopping time, Moving time, Passing time), Demand가 있다.

Interarrival time의 경우 고객들이 도착한 시간 간격을 계산하여 도출하고, Waiting time은 고객이 버튼을 누른 시간과 해당 엘리베이터가 1층에 도착하는 시간의 차이로 계산하여 도출하였다. Service Time의 경우 1층 출발 시간과 각 층별 도착 시간 사이로 계산할 수 있으나 이는 엘리베이터에 탑승한 고객들의 수요 층에 매우 dependent하기 때문에 고객에 따른 변동성이 크게 나타

났다. 이에 Service time을 Stopping time, Moving time, Passing time으로 분리하였으며 이 중에 Stopping time만을 Random variable로 설정하고 Moving time과 Passing time은 상수로 설정하였다. 이에 대한 자세한 설명은 3.1. Model Assumption에서 확인할 수 있다.

Demand의 경우 고객들의 수요 층을 의미하며 각 층별 하차 인원을 수집한 뒤 빈도에 따라 확률로 치환하였다.

2.3. Statistical Distribution Estimation

통계적 분포 추정을 위해 다음 5 단계를 거쳤다.

step 1	Homogeneity Check	Kruskal-Wallis test, Mann-Whitney U test
step 2	Independence Check	Correlation plot, Scatter diagram
step 3	Hypothesizing Families of Distributions	Summary statistics, Histograms and Box plots
step 4	Estimation of Parameters	Easyfit
step 5	Fitted Distibution Validation	DFD plot, P-P plot, Q-Q plot, GOF test

2.3.1. Collecting Data

수집된 데이터는 13:15에 수집된 데이터와 14:45에 수집된 데이터 두 그룹으로 나누어진다. 서로 다른 시간대에 수집된 두 데이터를 결합 가능한지 확인하기 위해 동질성 검증을 진행했다.

두 그룹 이상의 샘플을 비교하는 동질성 검증에서 주로 사용되는 Kruskal-Wallis test와 두 그룹만 비교하는 Mann-Whitney U test를 병행했다.

H0	13:15에 수집된 데이터와 14:45에 수집된 데이터의 분포가 동일하다.
H1	13:15에 수집된 데이터와 14:45에 수집된 데이터의 분포가 동일하지 않다.

Interarrival time

Kruskal-Wallis test	Kruskal-Wallis Statistic	1.4937340590356598
	Chi-square-value	3.841458820694124

Mann-Whitney U test	Mann-Whitney U Test Statistic	37963.0
	P-value	0.22174375917090194

Kruskal-Wallis statistic이 chi-square-value보다 작고 Mann-Whitney U test의 p-value가 0.05보다 크기 때문에 두 test에서 모두 귀무가설을 기각하지 못하며, 그룹간에 통계적으로 유의미한 차이가 없다고 볼 수 있다.

Service time

Kruskal-Wallis test	Kruskal-Wallis Statistic	0.8566963040448172
	Chi-square-value	3.841458820694124
Mann-Whitney U test	Mann-Whitney U Test Statistic	30082.0
	P-value	0.35483649950519147

Kruskal-Wallis statistic이 chi-square-value보다 작고 Mann-Whitney U test의 p-value가 0.05보다 크기 때문에 두 test에서 모두 귀무가설을 기각하지 못하며, 그룹간에 통계적으로 유의미한 차이가 없다고 볼 수 있다.

Waiting time

Kruskal-Wallis test	Kruskal-Wallis Statistic	0.8566963040448172
	Chi-square-value	3.841458820694124
Mann-Whitney U test	Mann-Whitney U Test Statistic	30082.0
	P-value	0.35483649950519147

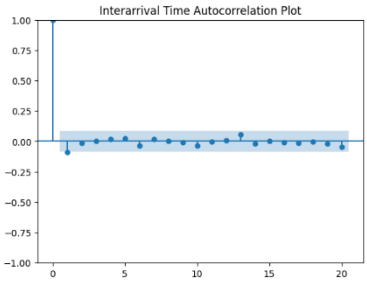
Kruskal-Wallis statistic이 chi-square-value보다 작고 Mann-Whitney U test의 p-value가 0.05보다 크기 때문에 두 test에서 모두 귀무가설을 기각하지 못하며, 그룹간에 통계적으로 유의미한 차이가 없다고 볼 수 있다.

Interarrival, service, waiting time 모두 13:15에 수집된 데이터와 14:45에 수집된 데이터의 분포에 유의미한 차이가 없었다. 이에 따라 두 그룹의 데이터를 하나의 데이터로 병합하여 이후 분석에 사용해도 된다는 결론을 도출했다.

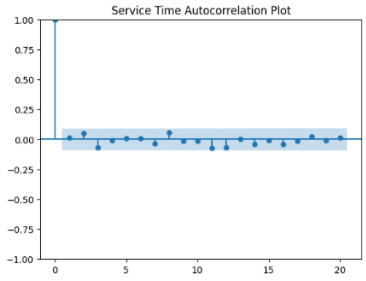
2.3.2. Independence Check

통계적 기법을 적용하기 위해서는 데이터가 서로 독립이어야 한다. 데이터가 독립인지 평가하기 위해 correlation plot과 scatter diagram을 시각화 했다.

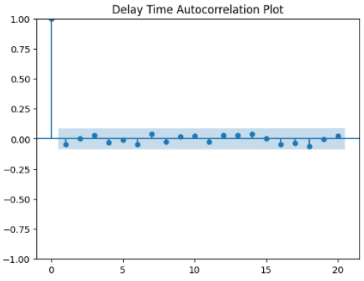
2.3.2.1. Correlation Plot



[그림1] Correlation Plot of Interarrival Time



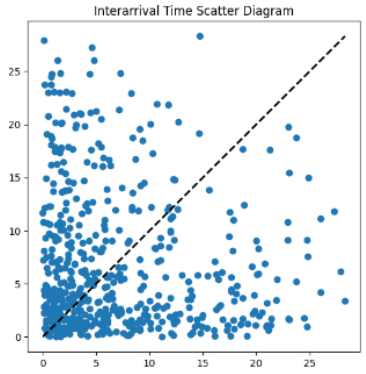
[그림2] Correlation Plot of Service Time



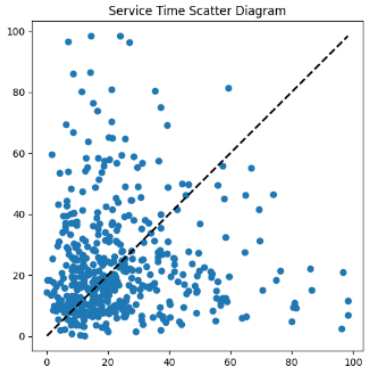
[그림3] Correlation Plot of Delay Time

Interarrival, service, waiting time data 각각의 autocorrelation이 거의 0에 근사하므로 heuristic하게 세 데이터 모두 independent하다고 판단할 수 있다.

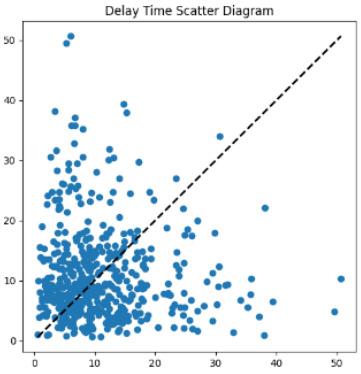
2.3.2.2. Scatter Diagram



[그림2] Scatter Diagram of Interarrival Time



[그림3] Scatter Diagram of Service Time



[그림4] Scatter Diagram of Delay Time

Interarrival, service, waiting time data의 scatter diagram이 기울기가 있는 직선 모양보다는 랜덤하게 흩어져 있는 모양에 가까우므로 세 데이터 모두 independent하다고 판단할 수 있다.

2.3.3. Hypothesizing Families of Distributions

2.3.3.1. Summary Statistics

Interarrival time

Statistic	Value	Percentile	Value
Sample Size	535	Min	0.01454
Range	28.284	25% (Q1)	1.6927
Mean	6.8777	50% (Median)	4.4286

Variance	42.947	75% (Q3)	10.508
Std. Deviation	6.5534	Max	17.701
Skewness	1.1735		

cv = Std. Deviation / Mean = 0.95284761 < 1 이므로 cv가 1인 exponential 분포와 거의 유사하지만 비교적 퍼진 정도가 작은 분포일 것이다. 왜도는 약 1.17으로 우편향되어 있음을 알 수 있다.

Service time

Statistic	**Value**	**Percentile**	**Value**
Sample Size	479	Min	0.06709
Range	98.232	25% (Q1)	10.951
Mean	23.394	50% (Median)	18.38
Variance	317.62	75% (Q3)	29.664
Std. Deviation	17.822	Max	98.299
Skewness	1.6496		

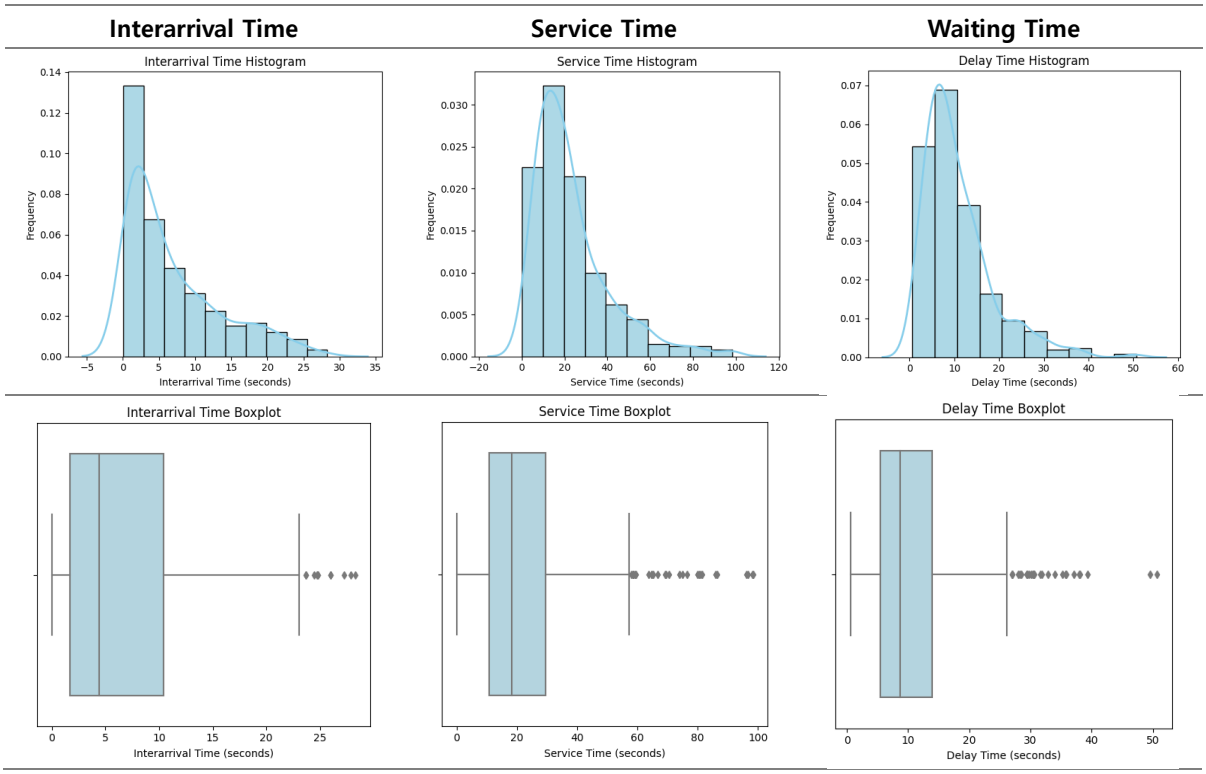
cv = Std. Deviation / Mean = 0.76181927 < 1 이므로 exponential 분포보다 퍼진 정도가 작은 분포일 것이다. 왜도는 약 1.65로 우편향되어 있는 분포임을 알 수 있다

Waiting time

Statistic	**Value**	**Percentile**	**Value**
Sample Size	511	Min	0.62618
Range	50.044	25% (Q1)	5.3998
Mean	10.758	50% (Median)	8.7201
Variance	60.123	75% (Q3)	13.969
Std. Deviation	7.7539	Max	50.67
Skewness	1.6566		

cv = Std. Deviation / Mean = 0.72075665 < 1 이므로 exponential 분포보다 퍼진 정도가 작은 분포일 것이다. 왜도는 약 1.66으로 우편향되어 있는 분포임을 알 수 있다.

2.3.3.2. Histograms and Box Plots



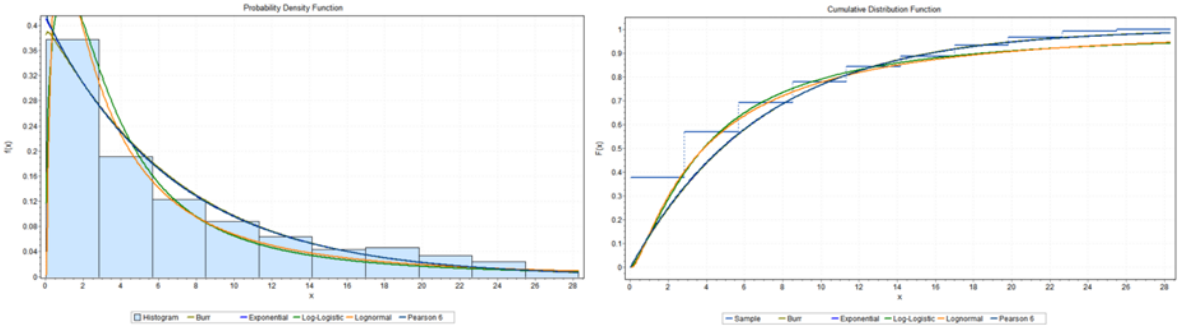
전체적으로 우편향 되어있는 분포임을 확인할 수 있다.

2.3.4. Estimation of Parameters & Goodness-of-Fit Test

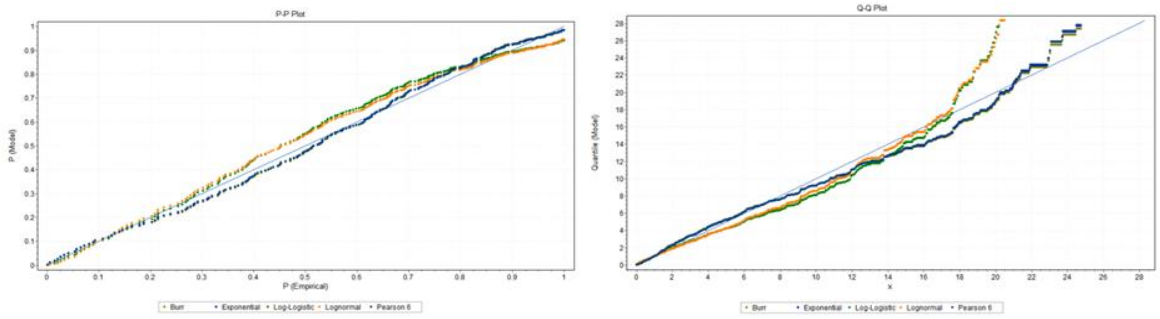
Interarrival time - Exponential

a. Choice of candidate distributions

P-value값과 분포의 형태를 통하여 추린 Interarrival time의 분포 후보는 Burr, Exponential, Log-Logistic, Lognormal, Pearson 6으로 5개이다. 후보 5개 각각의 PDF, CDF, PP-Plot, QQ-Plot, Probability Difference는 아래와 같다.



[그림5] PDF, CDF of Interarrival time



[그림6] Q-Q plot, P-P plot of Interarrival time

[그림7] Probability Difference of Interarrival time

대조한 결과, P-P plot과 Q-Q plot에서 더 큰 차이를 보이는 Log-Logistic, Lognormal을 후보군에서 제외하여 후보군을 Burr, Exponential, Pearson 6 3개로 추릴 수 있었다. 3개의 분포에 대해서 Goodness-of-Fit test를 진행하였다.

b. Goodness of fit test

		Burr	**Exponential**	Pearson 6
Kolmogorov-Smirnov	Statistic	0.04353	**0.03991**	0.0391
	P-Value	0.25519	**0.35234**	0.37688
Anderson-Darling	Statistic	1.9405	**1.7129**	1.6717
Chi-Squared	Statistic	19.018	**16.554**	16.6
	P-Value	0.02504	**0.05618**	0.05535

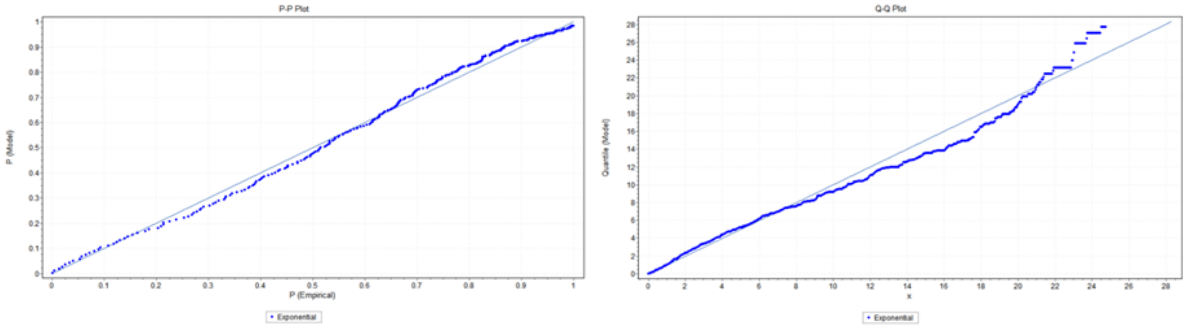
Goodness-of-Fit test 결과, Exponential과 Pearson 6가 Chi-Squared 검정에서 유의수준 0.05까지 만족시킬 수 있었으며, 다른 통계량 값에서도 Burr 보다 우수한 값을 가지기에 둘 중에 분포를 정하고자 한다. 통계적 수치로는 큰 차이가 없었기에 위의 Plot 결과 조금 더 우수한 모습을 보였던 Exponential 분포로 선정하고자 한다.

c. Parameter Estimation

Exponential Distribution의 parameter는 다음과 같다.

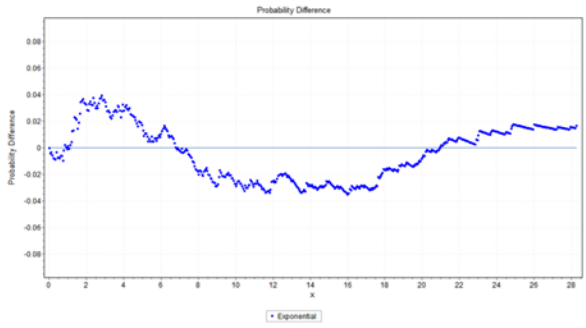
Parameter	Exponential
λ	0.1454

d. Heuristic Procedure



[그림8] Q-Q plot, P-P plot of Exponential Distribution(Interarrival time)

분포 Fitting의 양 끝, 중간 부분의 차이를 확인했으나 차이가 거의 없었다.



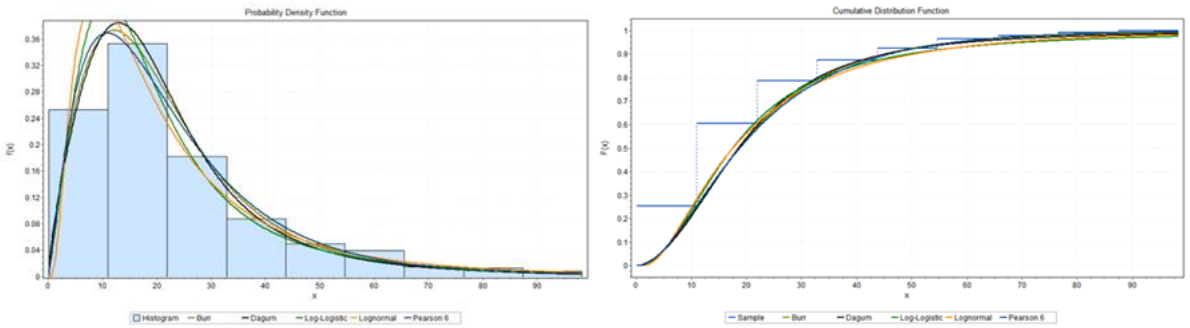
[그림9] Probability Difference of Exponential Distribution(Interarrival time)

fitted distribution의 CDF와 input data를 기반으로 작성한 empirical distribution의 CDF의 차이가 0.04이상 크기 차이나는 지점이 없다. Heuristic Procedure를 통하여 모든 분포가 적절하게 추정되었음을 확인하였다.

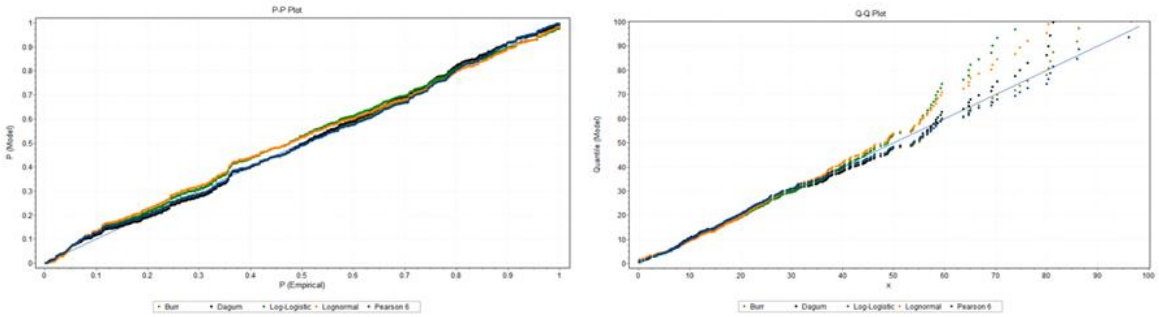
Service time – Burr

a. Choice of candidate distributions

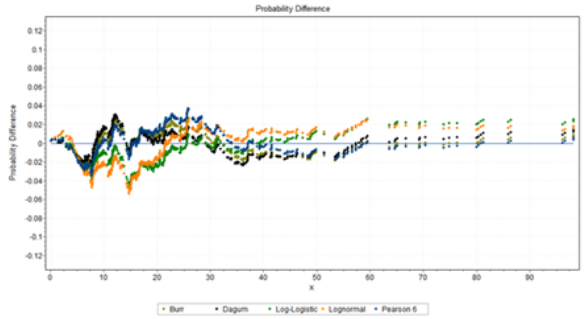
P-value값과 분포의 형태를 통하여 추린 Service time의 분포 후보는 Burr, Dagum, Log-Logistic, Pearson 6, Lognormal로 5개이다. 후보 5개 각각의 PDF, CDF, PP-Plot, QQ-Plot, Probability Difference는 아래와 같다.



[그림10] PDF, CDF of Service time



[그림11] Q-Q plot, P-P plot of Service time



[그림12] Probability Difference of Service time

대조한 결과, P-P plot과 Q-Q plot에서 더 큰 차이를 보이는 Log-Logistic, Lognormal을 후보군에서 제외하여 후보군을 Burr, Dagum, Pearson 6 3개로 추릴 수 있었다. 3개의 분포에 대해서 Goodness-of-Fit test를 진행하였다.

b. Goodness of fit test

		Burr	Dagum	Pearson 6
Kolmogorov-Smirnov	Statistic	**0.03069**	0.03057	0.03709

	P-Value	**0.74574**	0.74986	0.51333
Anderson-Darling	Statistic	**0.65852**	0.60219	0.79208
Chi-Squared	Statistic	**7.8255**	9.0238	12.172
	P-Value	**0.4507**	0.3403	0.1437

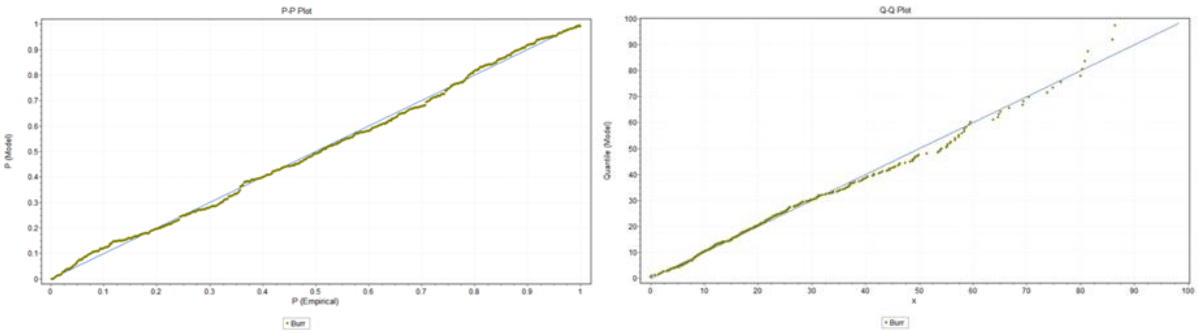
Goodness of Fitness test 결과 Burr와 Dagum이 Pearson 6보다 더 fit함을 확인할 수 있었으며, 둘 중 Chi-Squared의 P-value가 더 크며, Q-Q plot과 Probability Difference에서 더 작은 차이를 보였던 Burr분포로 선정하고자 한다.

c. Parameter Estimation

Burr Distribution의 parameter는 다음과 같다.

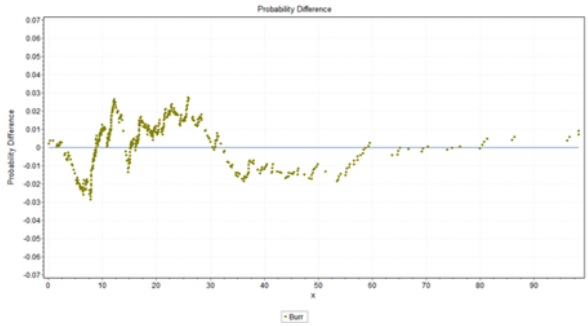
Parameter	Burr
κ	2.0536
α	1.8629
β	30.482

d. Heuristic Procedure



[그림13] Q-Q plot, P-P plot of Burr Distribution(Service time)

분포 Fitting의 양 끝, 중간 부분의 차이를 확인했으나 차이가 거의 없었다.



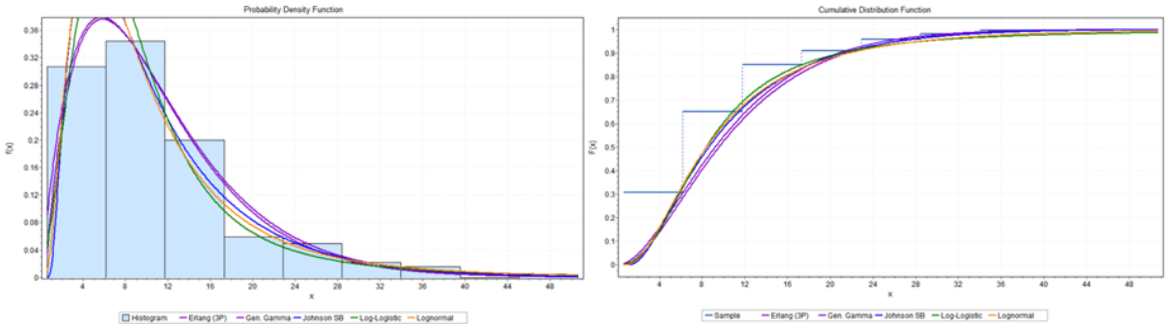
[그림14] Q-Q plot, P-P plot of Burr Distribution(Service time)

fitted distribution의 CDF와 input data를 기반으로 작성한 empirical distribution의 CDF의 차이가 0.03이상 크기 차이나는 지점이 없다. Heuristic Procedure를 통하여 모든 분포가 적절하게 추정되었음을 확인하였다.

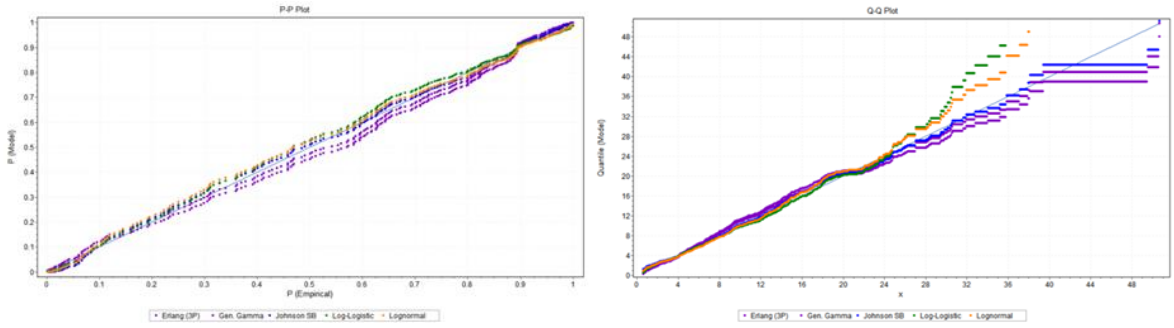
Waiting time – Lognormal

a. Choice of candidate distributions

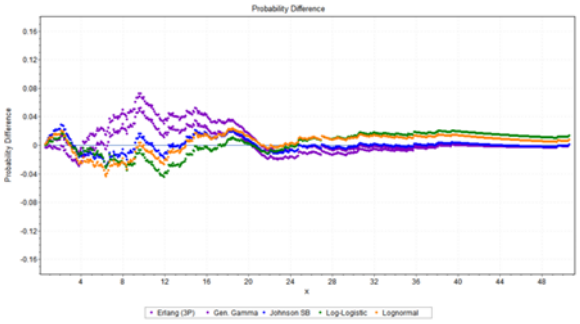
P-value값과 분포의 형태를 통하여 추린 Waiting time의 분포 후보는 Erlang(3P), Gen Gamma, Johnson SB, Log-Logistic, Lognormal으로 5개이다. 후보 5개 각각의 PDF, CDF, PP-Plot, QQ-Plot, Probability Difference는 아래와 같다.



[그림15] PDF, CDF of Waiting time



[그림16] Q-Q plot, P-P plot of Waiting time



[그림17] Probability Difference of Waiting time

Plot해본 결과, Q-Q plot에서 Erlang(3P), Gen Gamma, Johnson SB 은 가로선이 나타나기에 분포가 데이터를 제대로 설명하지 못한다고 판단하였고, 특히나 Erlang(3P)와 Gen Gamma는 Probability Difference에서 큰 차이를 보였기에 Erlang(3P), Gen Gamma, Johnson SB은 후보군에서 제외하여 후보군을 Log-Logistic, Lognormal 2개로 추릴 수 있었다. 2개의 분포에 대해서 Goodness-of-Fit test를 진행하였다.

b. Goodness of fit test

		Log-Logistic	**Lognormal**
Kolmogorov-Smirnov	Statistic	0.04553	**0.04498**
	P-Value	0.23274	**0.24499**
Anderson-Darling	Statistic	1.1568	**1.4758**
Chi-Squared	Statistic	7.0189	**7.0857**
	P-Value	0.53459	**0.52742**

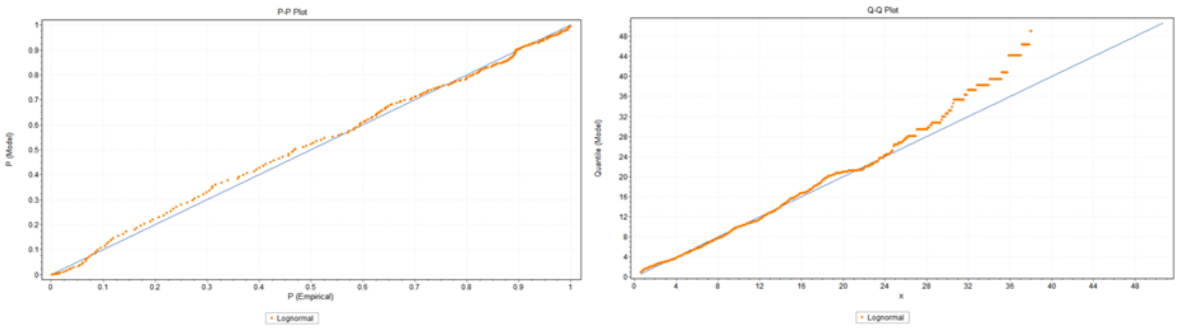
Goodness-of-Fit test를 진행한 결과, Log-Logistic, Lognormal 분포 모두 데이터의 분포를 설명할 수 있다는 결과를 확인할 수 있었다. 통계적 수치가 큰 차이가 없었기에, Heuristic한 방식으로 각각의 PDF, CDF, PP-Plot, QQ-Plot, Probability Difference을 plot해본 결과에 따라서 Lognormal 분포가 더 차이가 적었기에 Lognormal 분포로 선정하고자 한다.

c. Parameter Estimation

Lognormal Distribution의 parameter는 다음과 같다.

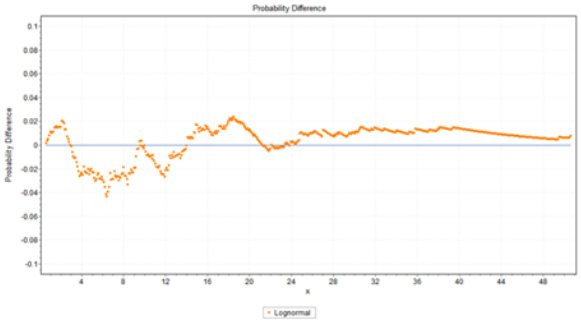
Parameter	Lognormal
σ	0.74334
μ	2.1267

d. Heuristic Procedure



[그림18] Q-Q plot, P-P plot of Lognormal Distribution(Waiting time)

분포 Fitting의 양 끝, 중간 부분의 차이를 확인했으나 차이가 거의 없었다.



[그림19] Probability Difference of Lognormal Distribution(Waiting time)

fitted distribution의 CDF와 input data를 기반으로 작성한 empirical distribution의 CDF의 차이가 0.05이상 크기 차이나는 지점이 없다. Heuristic Procedure를 통하여 모든 분포가 적절하게 추정되었음을 확인하였다.

3. Solution Approach

3.1. Model Assumption

시뮬레이션 모델의 가정은 다음과 같다.

1) 총 6대의 엘리베이터는 모두 같은 고객 이용 확률 및 서비스 타임 분포를 갖는다.

2) 현재 과학도서관은 모든 엘리베이터가 동일한 정책(전층운행)으로 운영되고 있기 때문에 6대 엘리베이터에 대한 고객의 선호 또는 서비스 타임의 차이는 모두 동일하다고 가정한다.

3) 시뮬레이션에 포함되는 고객은 1층에서 엘리베이터를 탑승해서 윗층으로 올라가는 사람들로 한정한다.

4) 고객이 시스템에 머무르는 시간은 waiting time, moving time, passing time, stopping time으로 구분한다. Service time은 moving time, passing time, stopping time의 합산으로 계산한다.

   A. Waiting time: 고객이 1층에서 엘리베이터를 기다리다 엘리베이터를 탈 때까지의 시간을 의미한다.

   B. Moving time: 엘리베이터가 1층에서 운행을 시작하여 한 층을 오르는데 걸리는 시간으로 service time을 구성하는 기본 요소이다.

   C. Passing time: 엘리베이터가 1개 층 이상을 지나쳐갈 때 각 층을 통과하는데 걸리는 시간은 Moving time보다 작다. 이를 고려하여 passing time을 별개로 측정하였다.

   D. Stopping time: 엘리베이터가 각 층에 멈춰 있는 시간으로 해당 층에서 내리는 고객 수, 문이 닫히는 시간 등에 영향을 받는다.

5) Moving time, Passing time의 경우 데이터 수집 결과 0.00초 이하의 차이를 보이는 것으로 확인하여 상수값으로 가정하였다.

6) Initial condition은 쉬는 시간 이전 과학도서관 엘리베이터의 Queue는 거의 존재하지 않음을 고려하여 시뮬레이션 시작 시점의 Queue를 0으로 설정하며, 모든 엘리베이터의 운행 상태, 버튼 상태 역시 각각 idle, false로 설정한다.

7) 엘리베이터를 기다리는 고객들은 하나의 Queue에서 대기하고, 엘리베이터가 도착했을 때 해당 엘리베이터를 탈 수 있는 고객 중 FIFO 방식으로 엘리베이터에 탑승한다.

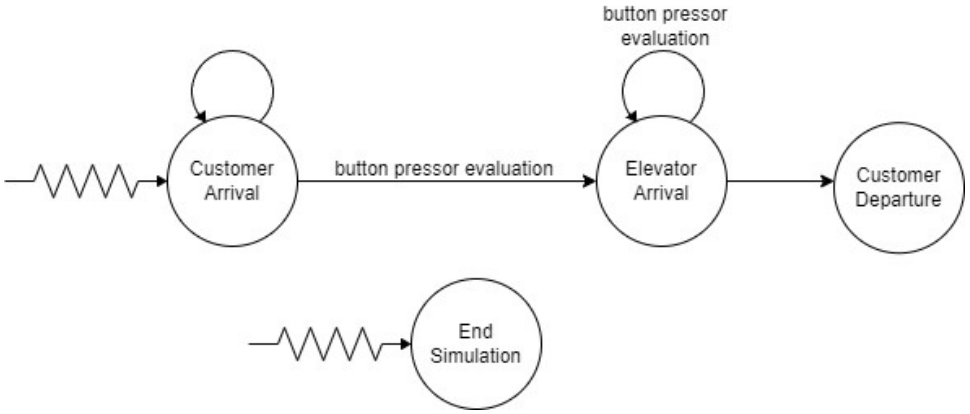
8) 고객은 엘리베이터 앞에 도착했을 때 Waiting Queue의 상황에 따라 버튼을 누를 지 누르지 않을지 결정한다. 엘리베이터를 기다리고 있는 고객이 일정 숫자 이상이라면 고객은 버튼을 누른다.

9) 본 시뮬레이션의 Stopping rule은 시뮬레이션 시간이 900초가 되는 시점이다. 이는 본 시뮬레이션의 대상 시간이 쉬는 시간으로 한정되기 때문으로, 15분에 해당한다.

3.2. Model Description

본 프로젝트의 시뮬레이션 모델은 과학도서관의 6대의 엘리베이터 각각이 어떤 policy를 채택하는 것이 customer가 system에 머무르는 시간을 최소화할 수 있는지 확인하여 가장 적절한 policy를 결정하는 것에 목적을 둔다. 본 모델의 구조 및 구성요소는 다음과 같다.

3.2.1. Event Graph

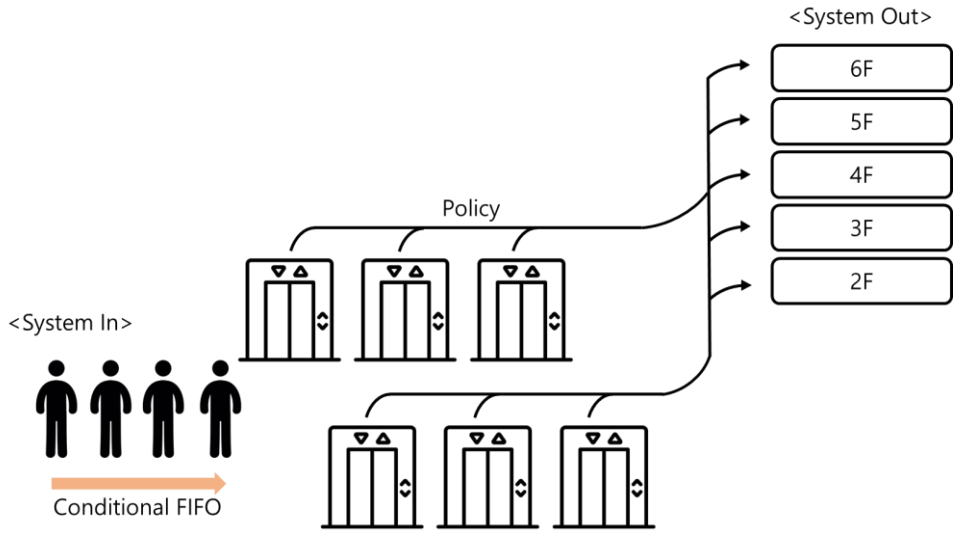


본 시뮬레이션은 첫번째 고객이 도착하면서 시작된다. 고객 도착과 동시에 inter-arrival time에 의해 다음 고객의 도착 이벤트가 생성되며, 고객이 엘리베이터 버튼을 눌러야하는지 판단하는 function이 수행된다. 만약 고객이 엘리베이터 버튼을 눌러야하는 상황이라면 버튼을 누르게 되고, 그 결과로 해당 엘리베이터가 1층에 도착하는 이벤트가 생성된다. 엘리베이터가 1층에 도착할 시 해당 엘리베이터에 탑승 가능한 고객들은 엘리베이터에 탑승한다. 엘리베이터에 탑승한 고객들은 가고자 희망했던 층에서 내리므로 각 수요 층에 따라 고객이 시스템을 떠나는 이벤트가 생성된다. 엘리베이터가 1층에 도착한 시점에 엘리베이터 버튼이 해제되므로 이때 다시 한 번 버튼을 눌러야하는지 판단하는 function이 실행된다. 본 시뮬레이션을 통해 관찰하고자 하는 시간은 쉬는 시간으로 한정되어 있기 때문에 End simulation event는 시뮬레이션 시간이 900초 (15분)이 되었을 때 발생한다.

3.2.2. Simulation model elements

Entities (Attributes)	Customer (time of arrival at ground floor, customer's demand floor)
	Elevator (elevator number)
Events	Customer Arrival, Elevator Arrival, Customer Departure, End Simulation
Queue	Waiting Queue (in ground floor)
State Variables	Status of elevators (busy, idle)
	Status of elevator buttons (pressed, not pressed)
	Number of customers in waiting queue
	Number of customers in each elevator
	Time of arrival of each each customer

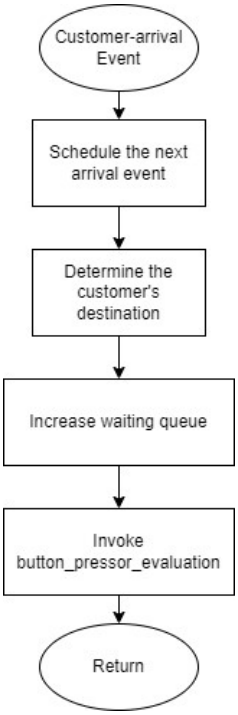
3.2.3. Simulation Scheme



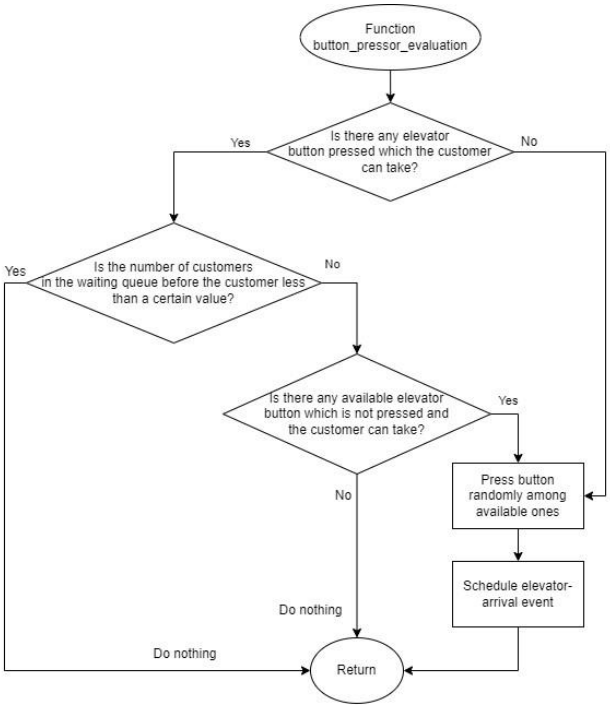
Customer arrival event의 발생으로 새로운 고객이 waiting queue에 추가될 때 해당 고객의 system in이 발생하고, 각 층에 도달하여 엘리베이터에서 내릴 때 해당 고객의 system out이 발생한다.

고객들은 FIFO 방식으로 waiting queue에서 대기하다 엘리베이터에 탑승하나, 이때 각 고객의 가기를 희망하는 층과 엘리베이터의 policy가 부합해야 한다. 즉, 엘리베이터가 도착하면 해당 엘리베이터를 탈 수 있는 고객 중에서 FIFO 방식으로 엘리베이터에 탑승한다. 각 엘리베이터는 탑승한 고객들의 수요 층만을 방문한다.

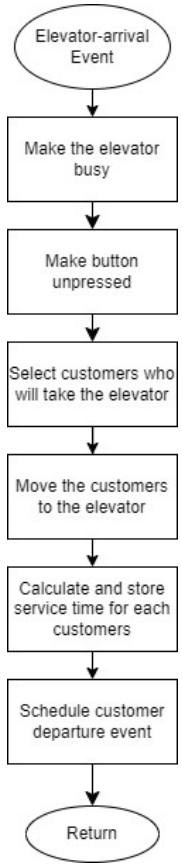
3.2.4. Model Structure



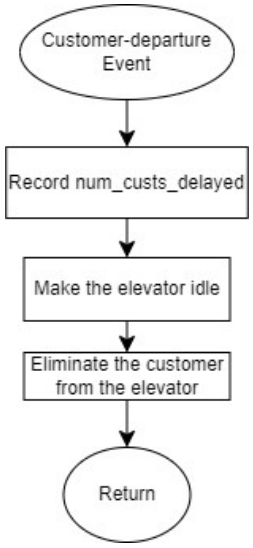
Customer Arrival



button_pressor_evaluation function (non-event function)



Elevator Arrival



Customer Departure

4. Simulation

본 시뮬레이션의 목적은 각 엘리베이터의 policy를 고객들이 시스템에 머무르는 시간을 최소화하는 방향으로 최적화하는 것에 있다. 따라서 실험을 수행할 Policy를 다음과 같이 결정하였고, 각 policy별로 동일한 input을 넣어주며 실험을 수행하였다.

- 각 엘리베이터가 가질 수 있는 Policy
  - All: 전층 운행 (2F, 3F, 4F, 5F, 6F)
  - Even: 짝수층 운행 (2F, 4F, 6F)
  - Odd: 홀수층 운행 (3F, 5F)
  - High: 고층 운행 (5F, 6F)
  - Low: 저층 운행 (2F, 3F, 4F)
- 6대 엘리베이터의 Policy 배치

- Policy0: All, All, All, All, All, All
- Policy1: Even, Odd, Even, Odd, Even, Odd
- Policy2: High, Low, High, Low, High, Low
- Policy3: All, All, Even, Odd, Even, Odd
- Policy4: All, All, High, Low, High, Low
- Policy5: All, All, All, All, Even, Odd
- Policy6: All, All, All, All, High, Low
- Policy7: All, All, Even, Odd, High, Low

4.1. Experimental results

300번의 반복실험을 수행한 결과로 얻어진 각 policy별 sample의 평균과 분산을 계산한 결과이다.

	Waiting Time		Service Time		Total Time	
policy	mean	var	mean	var	mean	var
0	9.7397	1.0872	77.1358	29.4984	86.8756	35.8247
1	10.5832	1.3419	69.4207	30.9848	80.0039	39.8696
2	10.6792	1.3173	68.5130	26.4154	79.1922	33.8622
3	9.8565	1.1248	73.5755	29.0914	83.4320	36.2274
4	9.9842	1.1404	73.7144	31.4461	83.6986	38.5519
5	9.7435	1.2521	75.6194	34.7127	85.3629	42.6622
6	9.7734	1.1090	75.8070	31.6610	85.5804	39.3709
7	9.9331	1.3053	73.3277	32.4137	83.2608	41.6684

	Elevator Utilization		Number of Customers in Queue	
policy	mean	var	mean	var
0	0.2243	0.0023	1.2387	0.0197
1	0.3277	0.0016	1.3749	0.0235
2	0.3208	0.0017	1.3813	0.0219
3	0.2801	0.0029	1.2567	0.0190
4	0.2825	0.0033	1.2647	0.0199

5	0.2501	0.0025	1.2348	0.0200
6	0.2555	0.0030	1.2368	0.0189
7	0.2744	0.0027	1.2553	0.0201

4.2. Output Analysis

총 8개의 Policy를 Paired-t test로 비교하기 위해서는 28개의 쌍에 대해 분석을 수행하여야 한다. 이는 분석의 효율성이 낮을 뿐만 아니라 분석 결과에 있어 모순이 발생할 확률이 높기 때문에 Koenig and Law의 Procedure를 따라 총 3개의 best 후보를 먼저 추리고자 하였다. 이때 기준이 되는 값은 본 프로젝트의 목적에 따라 Total time in system이다. 이후 추려진 후보군에 대해서 Waiting time, Service time, Average elevator utilization을 paired-t test를 통해 분석하였다.

4.2.1. Select 3-best candidates

총 8개의 policy가 존재하므로 k=8이며 3개의 best 후보를 추리기 위해 m=3으로 설정하였다. 신뢰수준은 95%로 설정하였으며, stage1에서 수행하는 실험 수인 $n_0 = 40$으로 설정하였으며 $d^* = 1$로 설정하였다. 위와 같은 조건에서 $h_2 = 2.267$이다.

policy	mean1	var1	N_i	N_i_minus_n_0	mean2	var2	W_i1	W_i2	mean_final
0	86.3212	36.4647	188	148	86.3983	33.9263	0.2359	0.7641	86.3801
1	82.0890	47.0804	242	202	79.3990	35.6133	0.1701	0.8299	79.8565
2	79.0822	46.8370	241	201	79.2680	31.6006	0.1789	0.8211	79.2348
3	82.1391	38.1856	197	157	83.3595	39.5974	0.2280	0.7720	83.0813
4	81.7070	40.7301	210	170	83.9368	41.4124	0.2128	0.7872	83.4623
5	83.7614	52.9665	273	233	85.6136	41.4973	0.1656	0.8344	85.3070
6	84.3788	25.8405	133	93	86.0839	39.1878	0.3185	0.6815	85.5409
7	83.0578	48.6947	251	211	83.4757	40.2695	0.1793	0.8207	83.4007

위 표에서 Mean_final이 작은 3개의 Policy는 Policy1, Policy2, Policy3이다. 따라서 Best 후보군 역시 Policy1, Policy2, Policy3으로 결정할 수 있다.

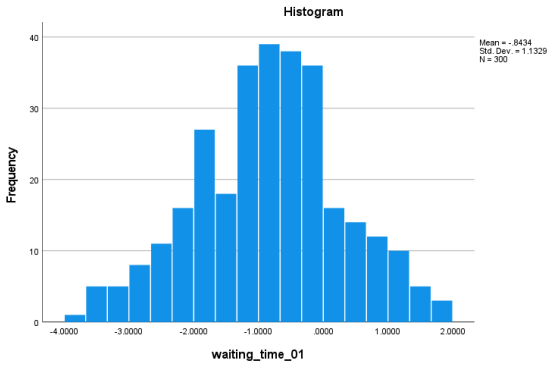
4.2.2. Paired-t test

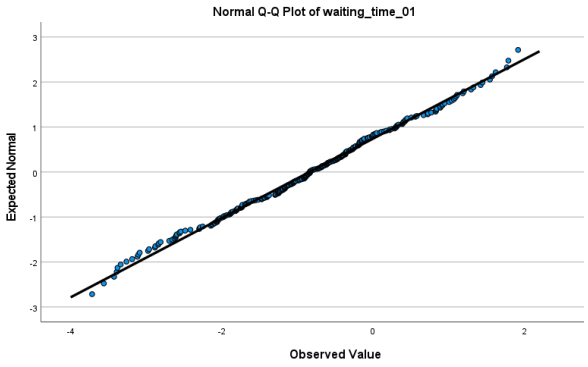
4.2.1에서 선정된 best 3 policy인 policy 1, 2, 3과 현재 과학도서관의 운영 정책인 policy 0에 대해 총 300번의 반복실험을 수행한 뒤 그 결과를 이용해 수행하였다. 반복수가 300회로 충분히 많기 때문에 Paired t test의 중요한 가정 사항인 Normality assumption을 만족시킬 것이라 예상할 수 있으나 정확한 분석을 위해 Normal Q-Q plot을 통해 각 pair의 차이 값이 Normal distribution

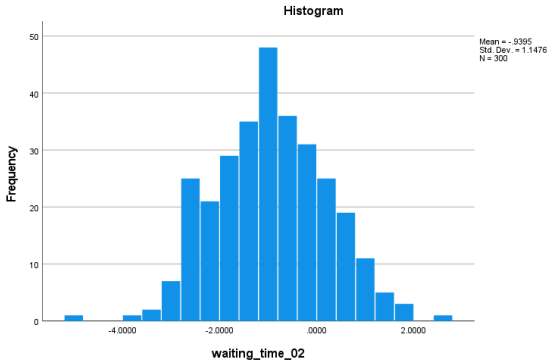
을 따르는 지 확인한 뒤 Paired-t test를 수행하였다.

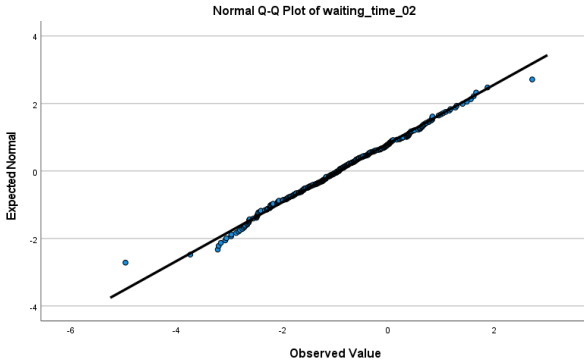
Waiting Time

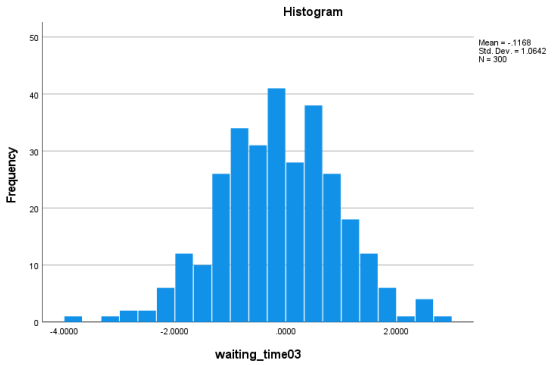
Waiting Time의 각 Policy 별 차이에 대한 히스토그램 및 Normal Q-Q Plot은 다음과 같다.

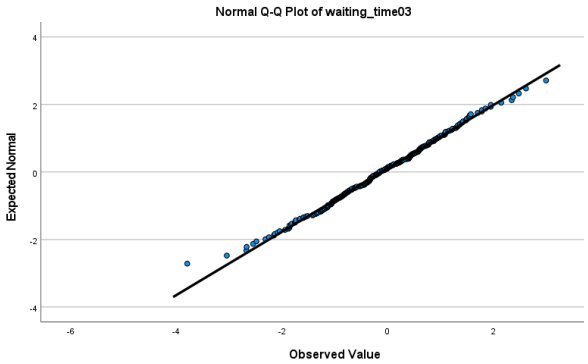


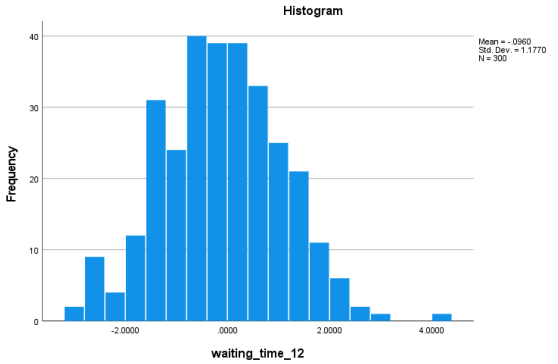


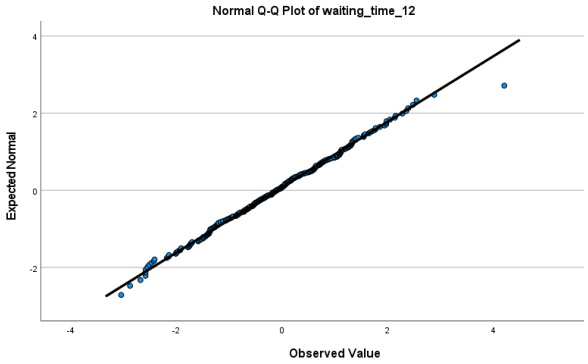


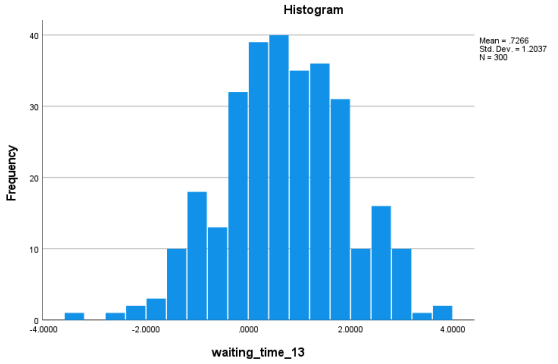


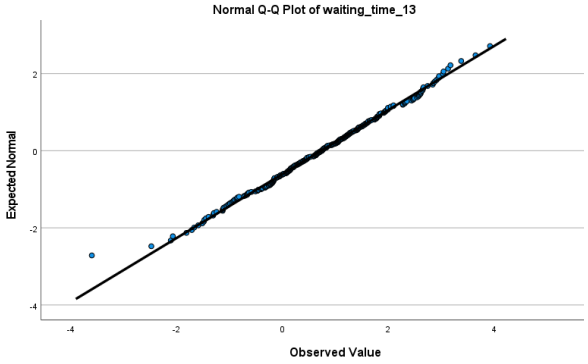


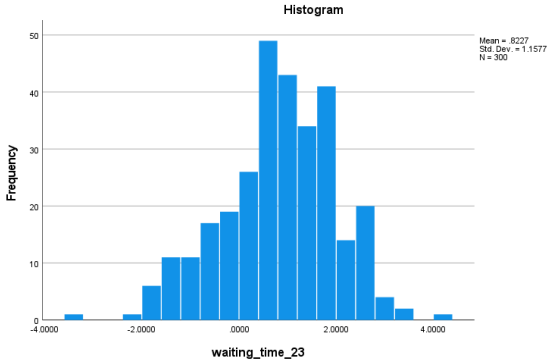


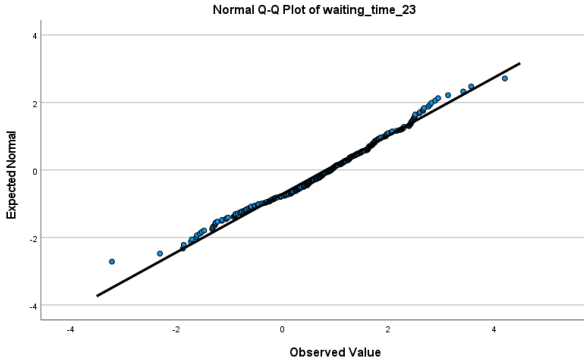












위 결과를 통해 Normality assumption을 잘 만족하고 있음을 확인할 수 있었으며 paired-t test의 수행 결과는 아래와 같다.

Paired Samples Test

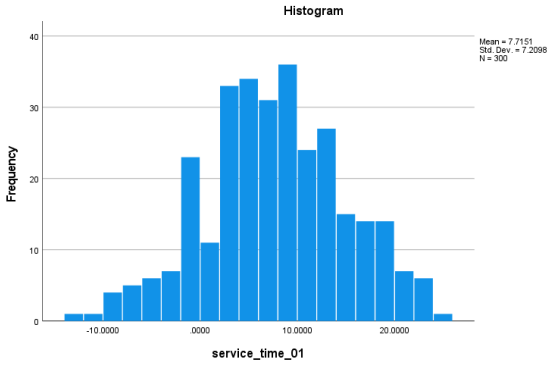
		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	Avg_waiting_time_0 - Avg_waiting_time_1	-.8434251	1.1328984	.0654079	-.9721433	-.7147069	-12.895	299	.000
Pair 2	Avg_waiting_time_0 - Avg_waiting_time_2	-.9394569	1.1475561	.0662542	-1.0698405	-.8090734	-14.180	299	.000
Pair 3	Avg_waiting_time_0 - Avg_waiting_time_3	-.1167836	1.0642385	.0614438	-.2377008	.0041336	-1.901	299	.058
Pair 4	Avg_waiting_time_1 - Avg_waiting_time_2	-.0960319	1.1769572	.0679517	-.2297559	.0376922	-1.413	299	.159
Pair 5	Avg_waiting_time_1 - Avg_waiting_time_3	.7266415	1.2036907	.0694951	.5898800	.8634030	10.456	299	.000
Pair 6	Avg_waiting_time_2 - Avg_waiting_time_3	.8226733	1.1577462	.0668425	.6911320	.9542147	12.308	299	.000

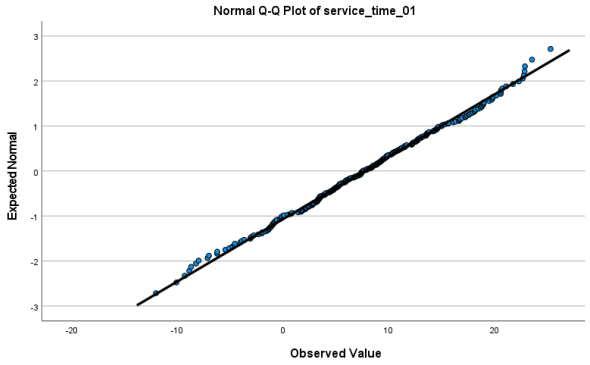
이는 95% 신뢰도를 기반으로 신뢰구간을 구한 결과로, Waiting Time의 검정 결과는 다음과 같다.

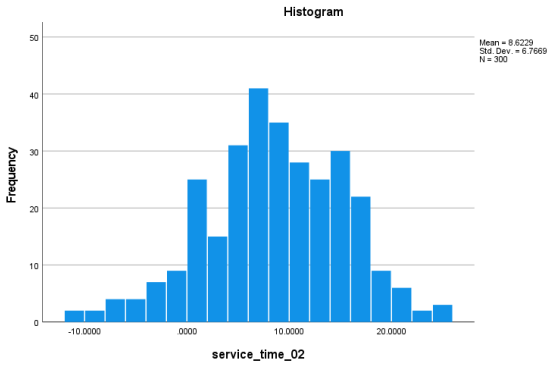
검정 결과: Policy 3 = Policy 0 < Policy 1 = Policy 2

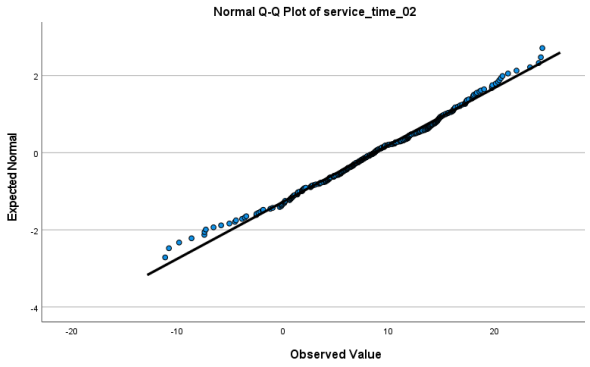
Service Time

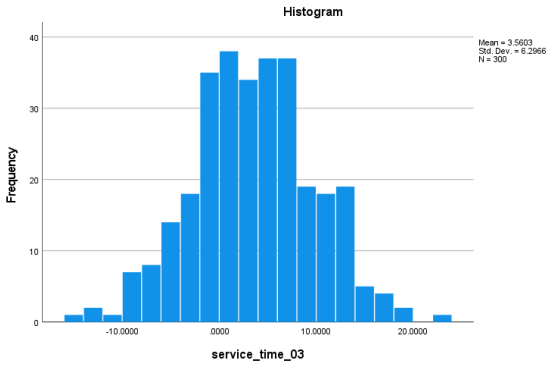
Service Time의 각 Policy 별 차이에 대한 히스토그램 및 Normal Q-Q Plot은 다음과 같다.

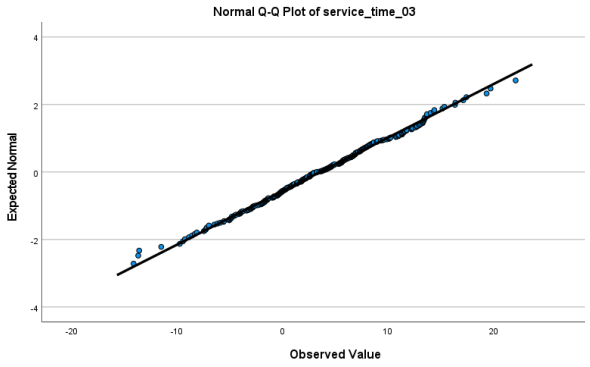


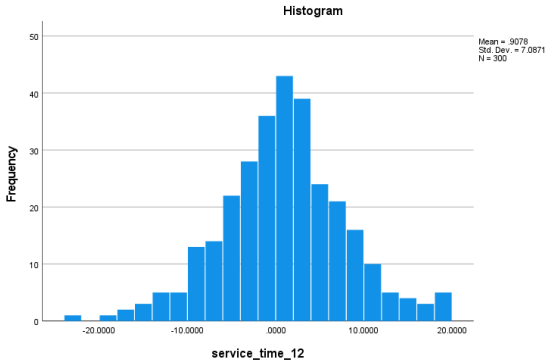


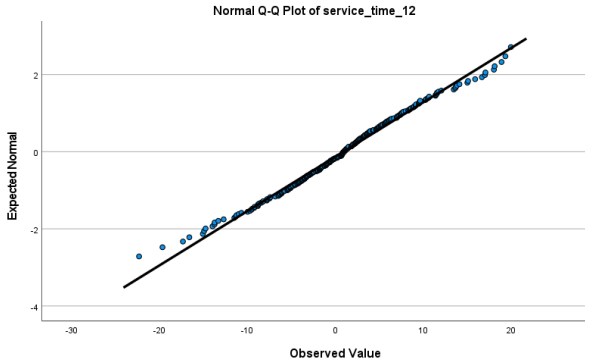


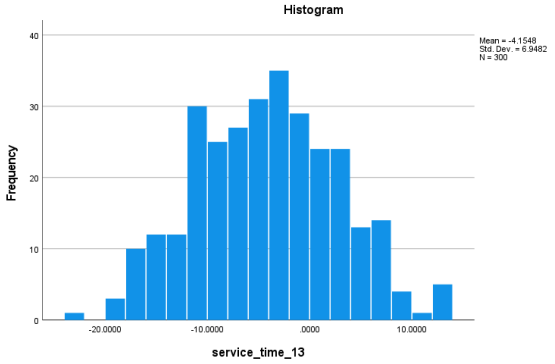


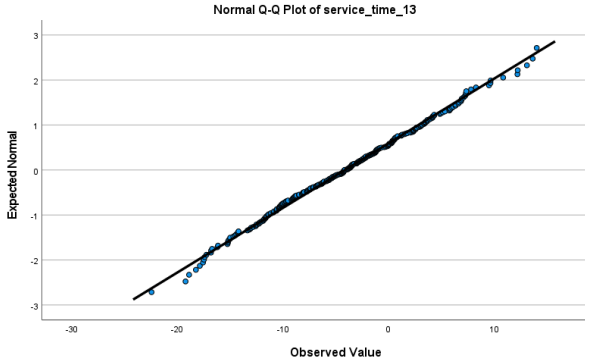


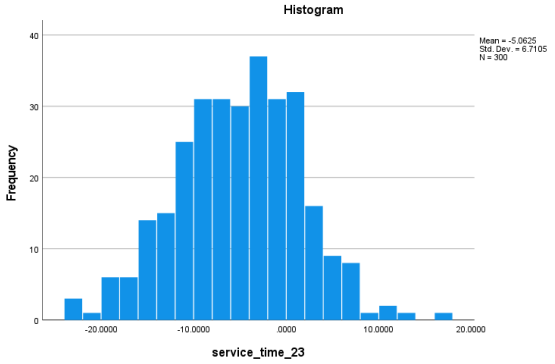


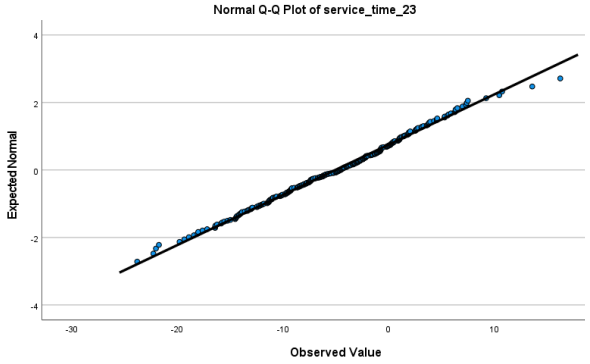












위 결과를 통해 Normality assumption을 잘 만족하고 있음을 확인할 수 있었으며 paired-t test의 수행 결과는 아래와 같다.

Paired Samples Test

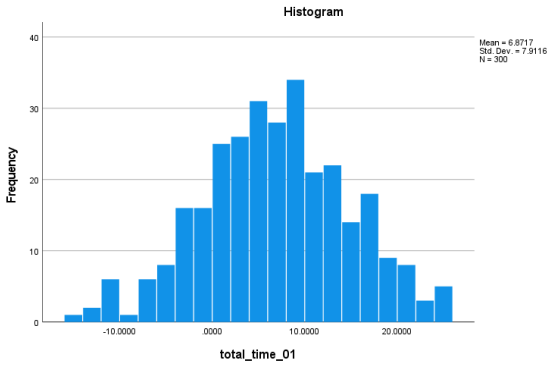
		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	Avg_service_time_0 - Avg_service_time_1	7.7150998	7.2098133	.4162588	6.8959319	8.5342678	18.534	299	.000
Pair 2	Avg_service_time_0 - Avg_service_time_2	8.6228848	6.7669365	.3906893	7.8540359	9.3917338	22.071	299	.000
Pair 3	Avg_service_time_0 - Avg_service_time_3	3.5603472	6.2966307	.3635361	2.8449336	4.2757608	9.794	299	.000
Pair 4	Avg_service_time_1 - Avg_service_time_2	.9077850	7.0871460	.4091766	.1025543	1.7130157	2.219	299	.027
Pair 5	Avg_service_time_1 - Avg_service_time_3	-4.1547526	6.9482002	.4011545	-4.9441965	-3.3653088	-10.357	299	.000
Pair 6	Avg_service_time_2 - Avg_service_time_3	-5.0625376	6.7105004	.3874309	-5.8249744	-4.3001008	-13.067	299	.000

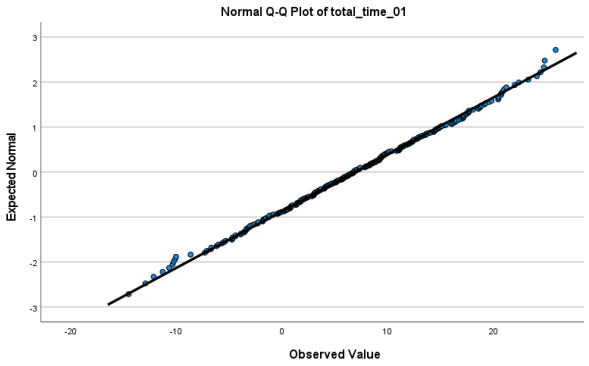
이는 95% 신뢰도를 기반으로 신뢰구간을 구한 결과로, Service Time의 검정 결과는 다음과 같다.

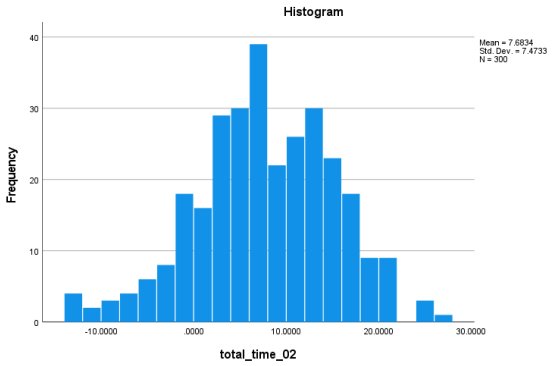
검정 결과: Policy 2 < Policy 1 < Policy 3 = Policy 0

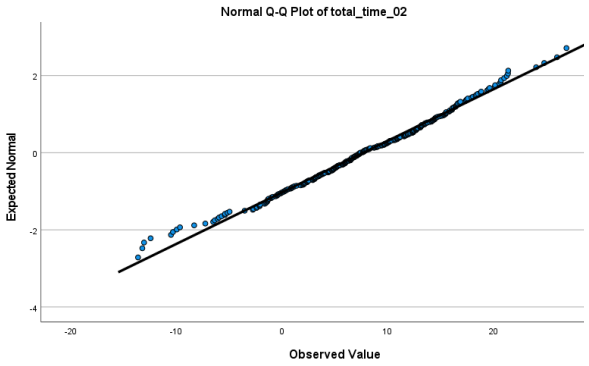
Total Time in System

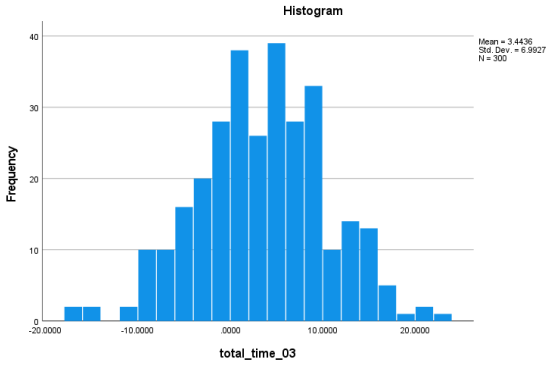
Total time in System의 각 Policy 별 차이에 대한 히스토그램 및 Normal Q-Q Plot은 다음과 같다.

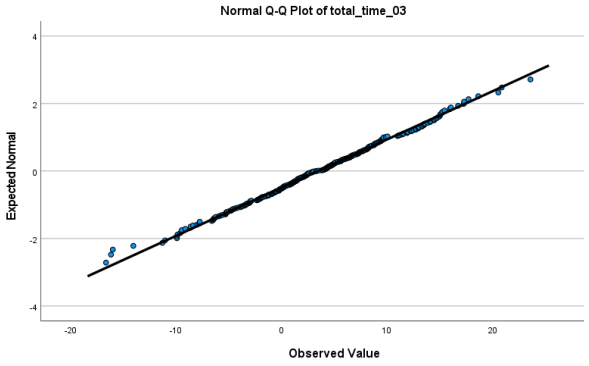


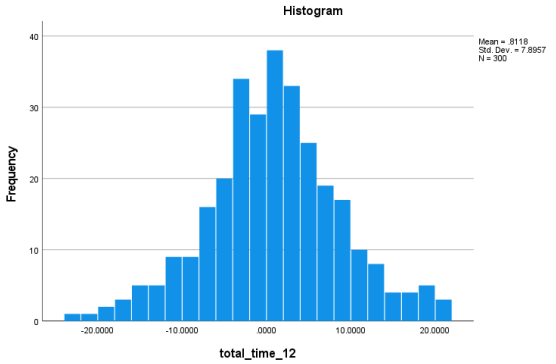


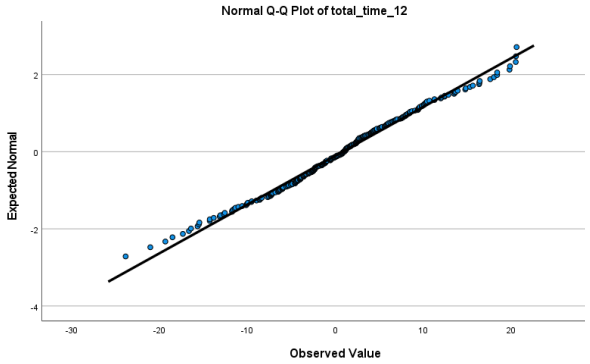


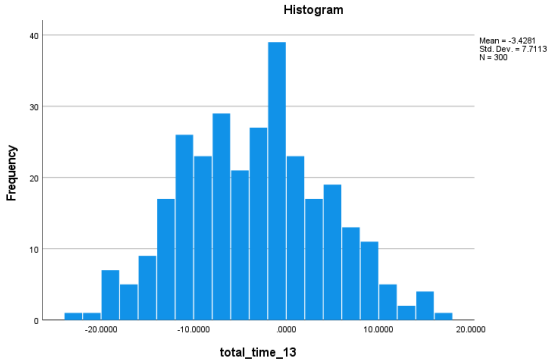


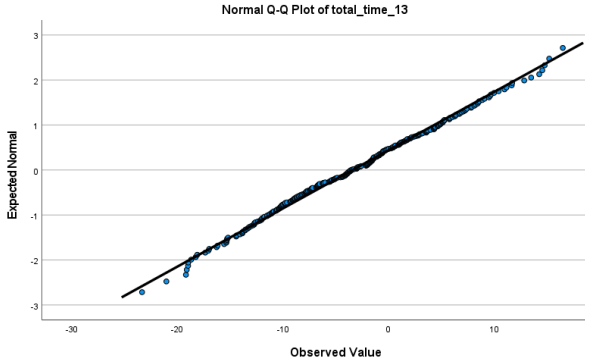


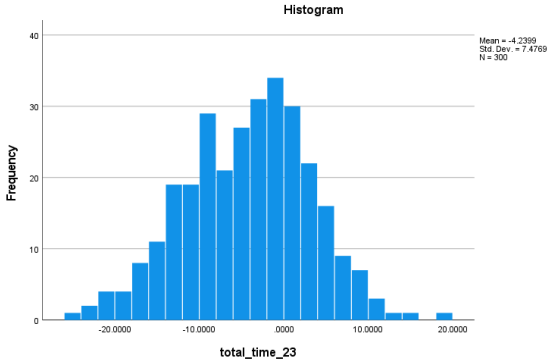


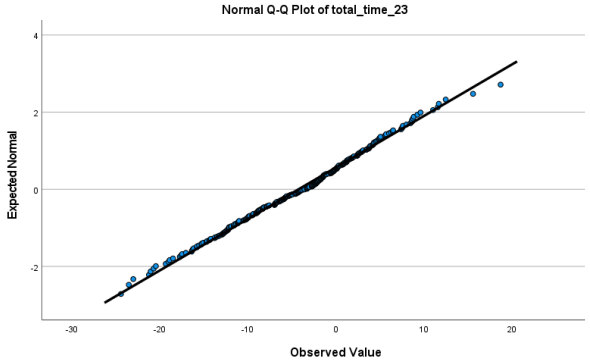












위 결과를 통해 Normality assumption을 잘 만족하고 있음을 확인할 수 있었으며 paired-t test의 수행 결과는 아래와 같다.

Paired Samples Test

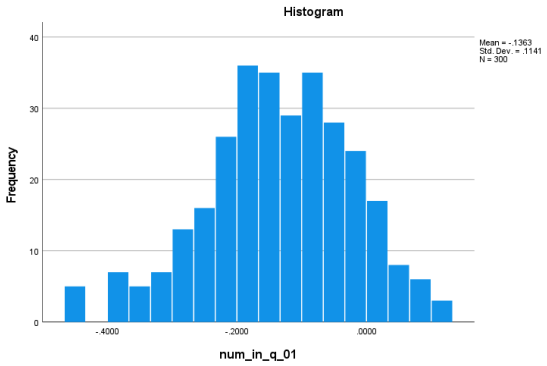
		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	Avg_total_time_0 - Avg_total_time_1	6.8716748	7.9115550	.4567738	5.9727760	7.7705736	15.044	299	.000
Pair 2	Avg_total_time_0 - Avg_total_time_2	7.6834279	7.4732552	.4314686	6.8343281	8.5325278	17.808	299	.000
Pair 3	Avg_total_time_0 - Avg_total_time_3	3.4435636	6.9926658	.4037217	2.6490676	4.2380596	8.530	299	.000
Pair 4	Avg_total_time_1 - Avg_total_time_2	.8117531	7.8957005	.4558585	-.0853443	1.7088506	1.781	299	.076
Pair 5	Avg_total_time_1 - Avg_total_time_3	-3.4281112	7.7112935	.4452117	-4.3042566	-2.5519658	-7.700	299	.000
Pair 6	Avg_total_time_2 - Avg_total_time_3	-4.2398643	7.4769233	.4316804	-5.0893809	-3.3903477	-9.822	299	.000

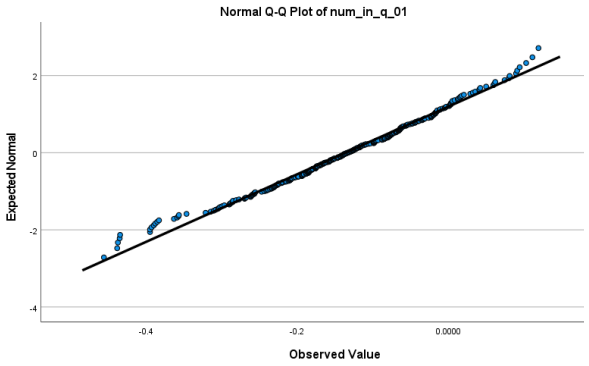
이는 95% 신뢰도를 기반으로 신뢰구간을 구한 결과로, Total Time in System의 검정 결과는 다음과 같다.

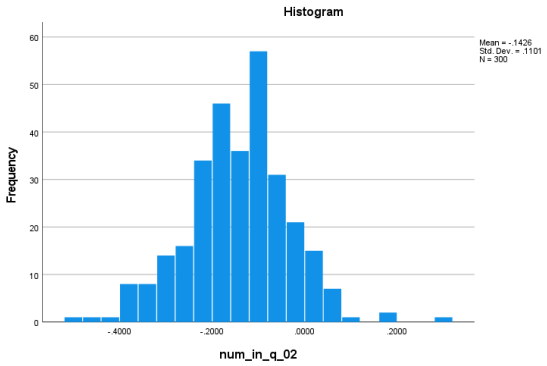
검정 결과: Policy 1 = Policy 2 < Policy 3 < Policy 0

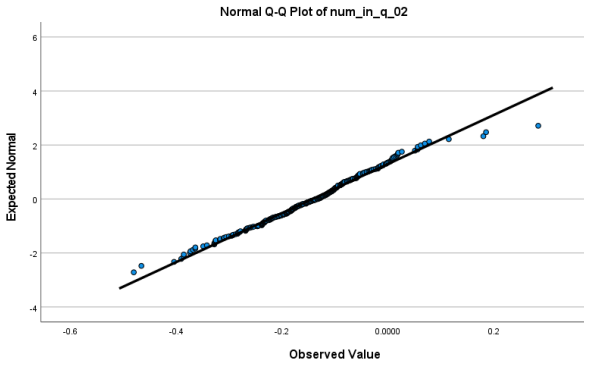
Number of Customers in Queue

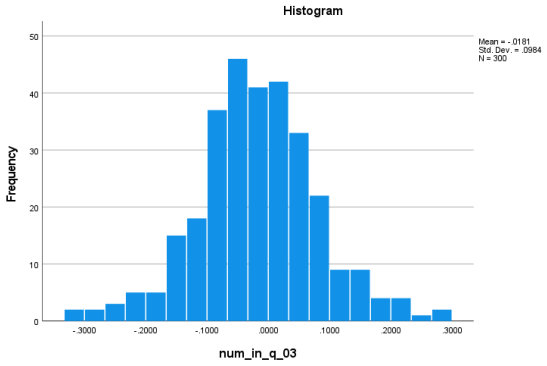
Number of Customers in Queue의 각 Policy 별 차이에 대한 히스토그램 및 Normal Q-Q Plot은 다음과 같다.

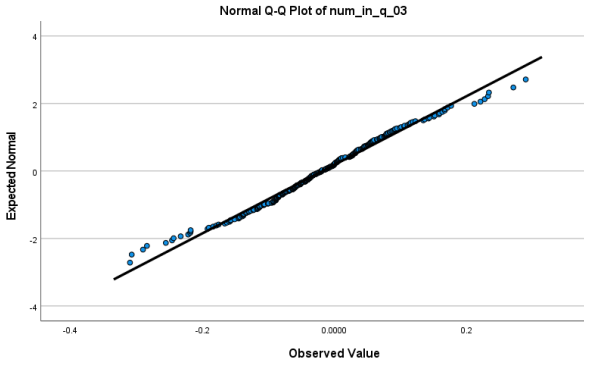


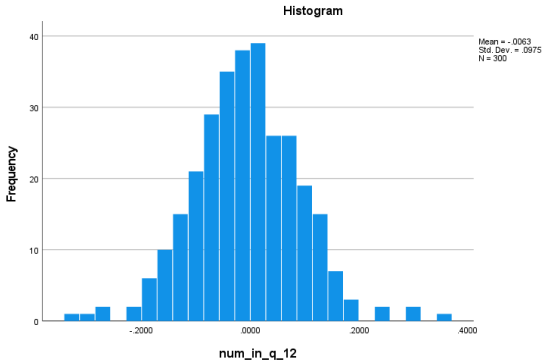


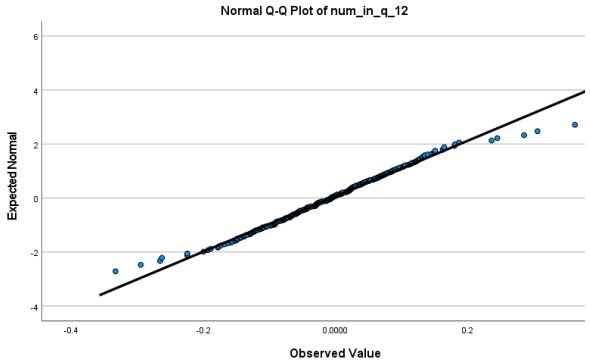


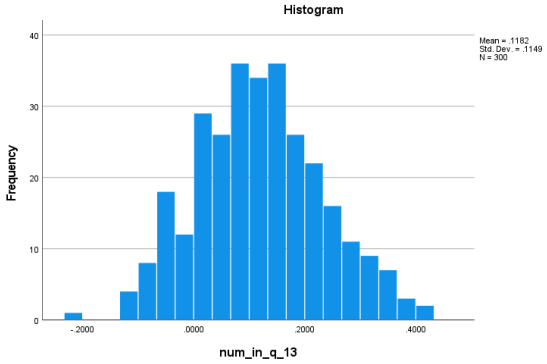


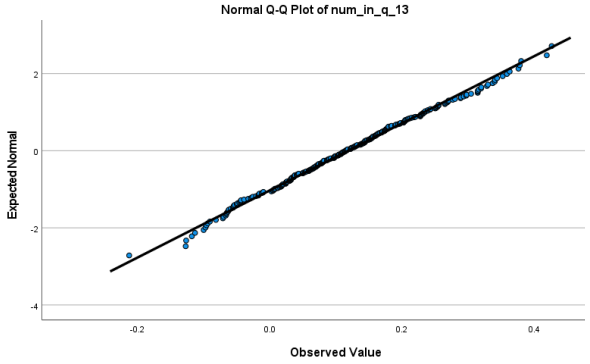


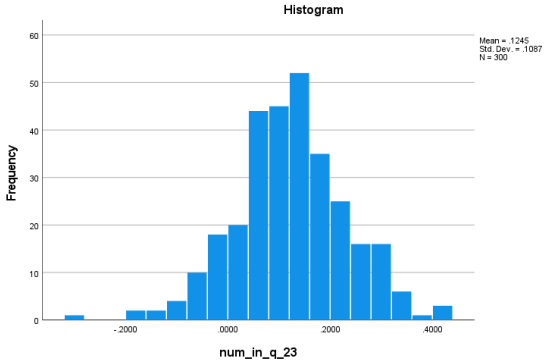


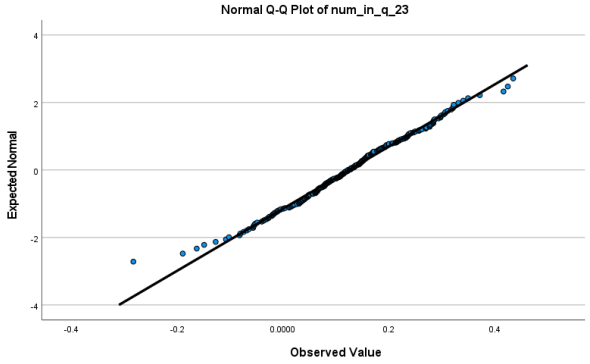












위 결과를 통해 Normality assumption을 잘 만족하고 있음을 확인할 수 있었으며 paired-t test의 수행 결과는 아래와 같다.

Paired Samples Test

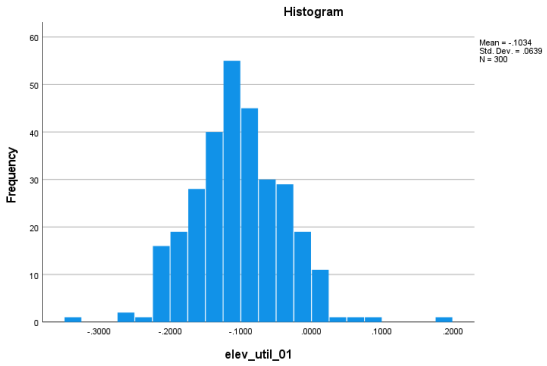
		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	Avg_num_in_q_0 - Avg_num_in_q_1	-.1362512	.1140670	.0065857	-.1492113	-.1232911	-20.689	299	.000
Pair 2	Avg_num_in_q_0 - Avg_num_in_q_2	-.1425944	.1101446	.0063592	-.1551089	-.1300799	-22.423	299	.000
Pair 3	Avg_num_in_q_0 - Avg_num_in_q_3	-.0180573	.0984046	.0056814	-.0292378	-.0068767	-3.178	299	.002
Pair 4	Avg_num_in_q_1 - Avg_num_in_q_2	-.0063432	.0974739	.0056277	-.0174180	.0047316	-1.127	299	.261
Pair 5	Avg_num_in_q_1 - Avg_num_in_q_3	.1181939	.1148803	.0066326	.1051414	.1312465	17.820	299	.000
Pair 6	Avg_num_in_q_2 - Avg_num_in_q_3	.1245372	.1087181	.0062768	.1121848	.1368895	19.841	299	.000

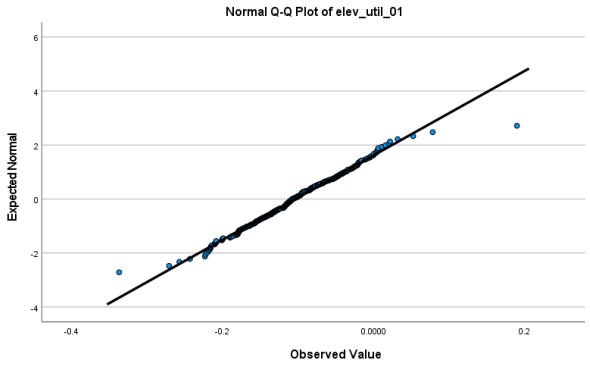
이는 95% 신뢰도를 기반으로 신뢰구간을 구한 결과로, Number of Customers in Queue의 검정 결과는 다음과 같다.

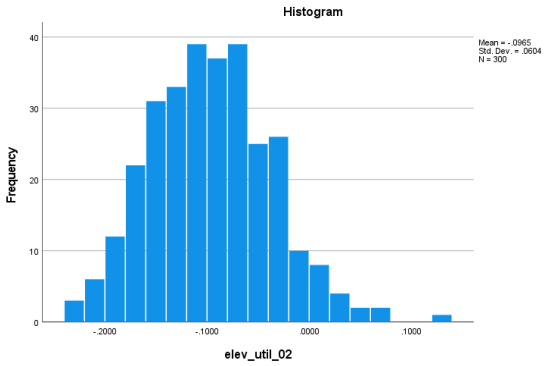
검정 결과: Policy 0 < Policy 3 < Policy 1 = Policy 2

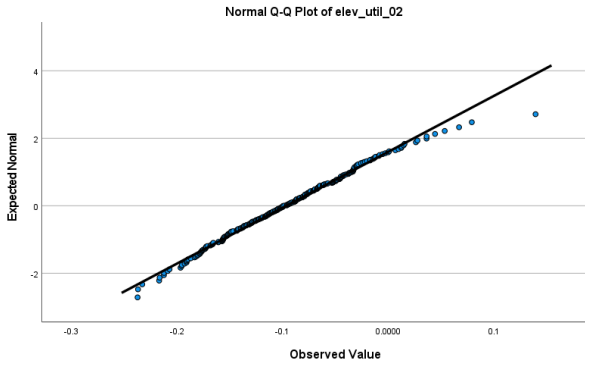
Average Elevator Utilization

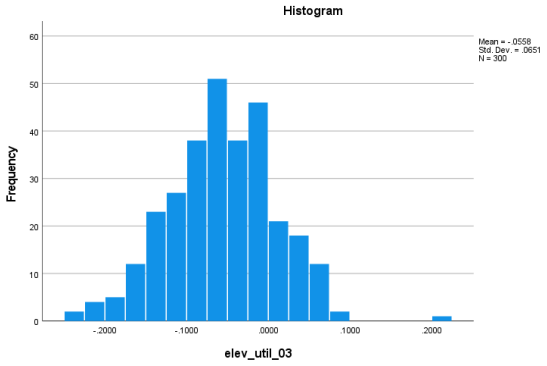
Average Elevator Utilization의 각 Policy 별 차이에 대한 히스토그램 및 Normal Q-Q Plot은 다음과 같다.

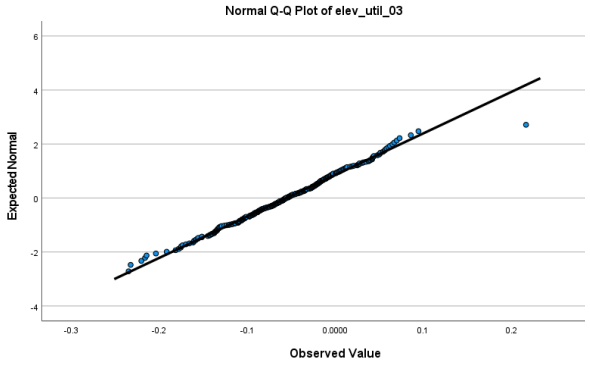


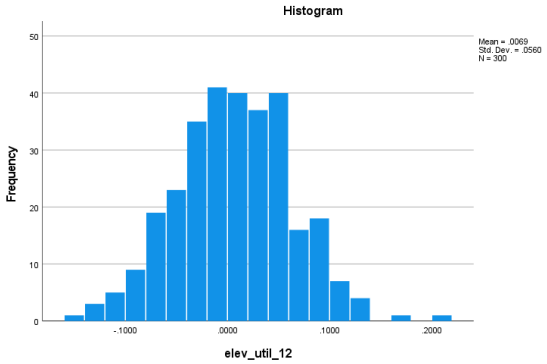


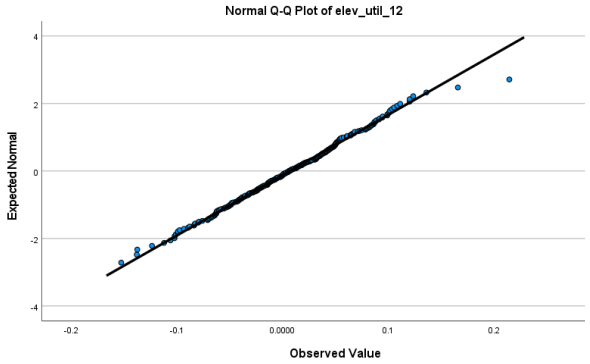


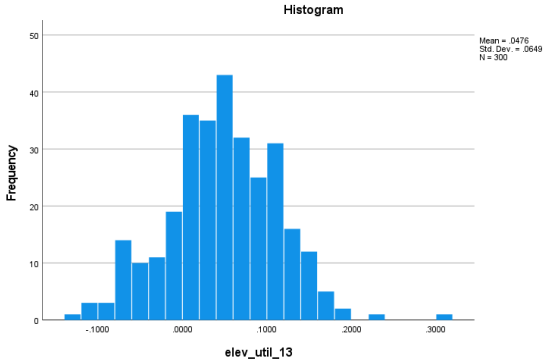


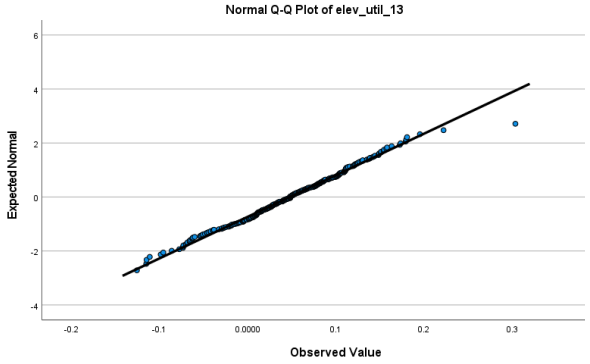


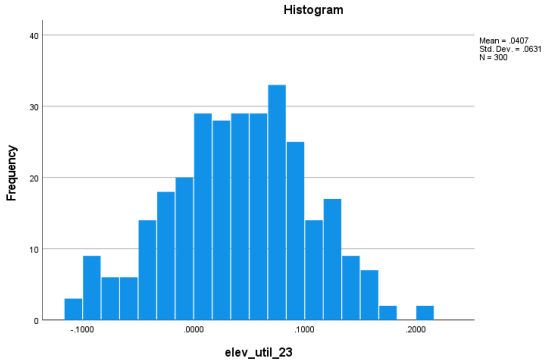


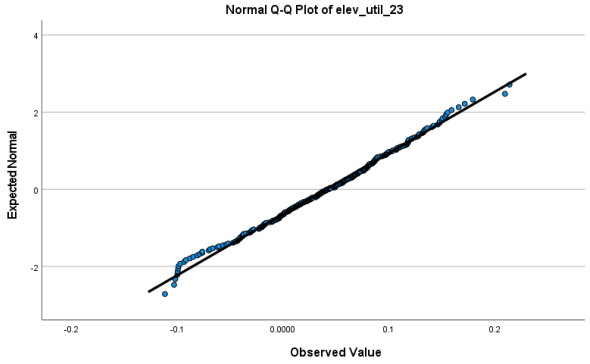












위 결과를 통해 Normality assumption을 대체로 잘 만족하고 있음을 확인할 수 있었으며 paired-t test의 수행 결과는 아래와 같다.

Paired Samples Test

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	Avg_elev_util_0 - Avg_elev_util_1	-.1033737	.0638912	.0036888	-.1106329	-.0961145	-28.024	299	.000
Pair 2	Avg_elev_util_0 - Avg_elev_util_2	-.0964940	.0603709	.0034855	-.1033532	-.0896347	-27.684	299	.000
Pair 3	Avg_elev_util_0 - Avg_elev_util_3	-.0558194	.0650503	.0037557	-.0632103	-.0484284	-14.863	299	.000
Pair 4	Avg_elev_util_1 - Avg_elev_util_2	.0068797	.0559618	.0032310	.0005214	.0132380	2.129	299	.034
Pair 5	Avg_elev_util_1 - Avg_elev_util_3	.0475543	.0649364	.0037491	.0401764	.0549323	12.684	299	.000
Pair 6	Avg_elev_util_2 - Avg_elev_util_3	.0406746	.0630829	.0036421	.0335072	.0478420	11.168	299	.000

이는 95% 신뢰도를 기반으로 신뢰구간을 구한 결과로, Average Elevator Utilization의 검정 결과는 다음과 같다.

검정 결과: Policy 0 < Policy 3 < Policy 2 < Policy 1

4.2.3. Output Summary

1) Select 3-best candidates: Policy 1, Policy2, Policy 3

2) Paired-t Test 대상: Policy 0(현재), Policy1, Policy2, Policy3

- Policy0: All, All, All, All, All, All
- Policy1: Even, Odd, Even, Odd, Even, Odd
- Policy2: High, Low, High, Low, High, Low
- Policy3: All, All, Even, Odd, Even, Odd

3) Paired-t test 결과

Waiting Time	Policy 3 = Policy 0 < Policy 1 = Policy 2
Service Time	Policy 2 < Policy 1 < Policy 3 = Policy 0
Total Time in System	Policy 1 = Policy 2 < Policy 3 < Policy 0
Number of Customers in Queue	Policy 0 < Policy 3 < Policy 1 = Policy 2
Average Elevator Utilization	Policy 0 < Policy 3 < Policy 2 < Policy 1

위 결과를 통해서 Waiting Time과 Service Time은 Trade-off 관계가 있음을 알 수 있다. 이를 Policy와 연관시켜 해석한 결과는 다음과 같다. 모든 엘리베이터가 전층 운행을 하는 경우(Policy 0) 1층에서 엘리베이터를 기다리고 있는 고객이 탈 수 있는 엘리베이터의 선택지가 더 많기 때문에 Waiting Time이 상대적으로 작다. 그러나 엘리베이터에 탑승한 고객들의 수요 층이 다양해짐에 따라 엘리베이터의 정차 횟수가 늘어나고 그 결과로 Service Time의 증가가 일어난다. 반대로, 엘리베이터가 일부 층만 운행하는 경우 엘리베이터의 선택지는 줄어들게 되어 Waiting Time은 상대적으로 길지만 정차 횟수의 감소로 Service Time이 줄어든다.

위와 같은 Trade-off가 존재하는 상황에서 결론을 내리기 위해 가장 중요한 statistic variable은 각 고객이 엘리베이터를 이용하기 위해 기다리는 시간과 엘리베이터를 타고 이동하는 시간을 합한 값을 의미하는 total time in system이다. 따라서 Paired-t test 결과를 Total time in System 측면에서 우선적으로 살펴보면 해당 값이 작은 Policy 1과 Policy 2가 실험한 Policy 중에서 가장 Best임을 알 수 있다.

Policy 1과 Policy 2는 total time in system 측면에서 통계적으로 다르다고 말할 수 없기에 두 Policy의 비교는 total time in system이 아닌 다른 static variable을 참고하였다. Average elevator utilization은 각 엘리베이터의 운행 횟수에 dependent한 variable로, 엘리베이터가 고객을 태우고

운행하는 횟수가 많을수록 높은 값을 갖는다. 시스템에 머무르는 시간이 동일하다면, 전력 소비를 줄이기 위해 운행 횟수가 적은, 즉 utilization이 적은 Policy가 더 좋은 Policy라고 말할 수 있다.

따라서 average elevator utilization 측면에서 Policy 2가 Policy 1보다 더 좋은 Policy이다. 즉, 최종 Best Policy는 Policy 2로, 고층운행과 저층운행으로 각각 3대씩 운용하는 것이다.

5. Model Validation

	avg_waiting_time	avg_service_time	avg_total_time	avg_num_in_q	elev1_utilization
mean	10.406358	85.603113	96.009471	1.190622	0.321287
std	1.260454	10.415538	11.268256	0.127206	0.126086
min	7.411698	61.680886	71.372727	0.879148	0.073071
25%	9.489104	78.661338	88.829844	1.097995	0.225806
50%	10.166653	84.620604	94.791133	1.185901	0.301869
75%	11.169883	90.806502	101.075866	1.272866	0.393496
max	16.418968	134.170758	150.589726	1.768861	0.755728

	elev2_utilization	elev3_utilization	elev4_utilization	elev5_utilization	elev6_utilization	avg_elev_utilization
mean	0.326188	0.331768	0.323403	0.323116	0.327829	0.325599

std	0.118746	0.128627	0.125830	0.118872	0.126267	0.051830
min	0.071877	0.090411	0.098651	0.063070	0.123654	0.189322
25%	0.246281	0.228733	0.222654	0.238474	0.234038	0.285651
50%	0.313617	0.312010	0.305209	0.307029	0.308953	0.323468
75%	0.392950	0.417609	0.396237	0.398852	0.413205	0.360258
max	0.770181	0.773189	0.862826	0.685976	0.837566	0.479809

Model의 validation을 검증하기 위해 실제 데이터로 시뮬레이션을 실행한 결과를 실제 값과 비교하는 trace-driven 시뮬레이션을 실행하고자 했다. 하지만 실제 system에서의 total time을 알 수 없기 때문에 model validation을 empirical하게 진행했다.

위 표는 각 policy에 대해 300번 반복 실험한 결과에 대해 통계적으로 요약한 값이다. Waiting time의 평균 값은 10초, 최솟값은 7.41초, 최댓값은 16.41초이고 service time의 평균 값은 85.6초, 최솟값은 61.7초, 최댓값은 134.2초이었다. Total time의 평균 값은 96.0초, 최솟값은 71.4초, 최댓값은 150.6초이고 queue length의 평균 값은 1.2명, 최솟값은 0.9명, 최댓값은 1.8명이었다. Waiting time, service time, total time, queue length의 mean, min, max 값이 충분히 현실성 있기에 본 모델이 validate 하다고 결론 내릴 수 있다.

6. Conclusion

엘리베이터 운영 정책을 결정하는 시뮬레이션을 통해 최종 Best Policy로 Policy 2를 도출하였다. 초기에 세운 "Service Time과 Waiting Time에 Trade off가 존재할 것"이라는 가정이 시뮬레이션 결과를 통해 성립함을 확인할 수 있었다. 이는 엘리베이터의 서비스 시간을 최적화하면서 대기 시간을 최소화하는 것 사이에 상충 관계가 존재한다는 것을 시사한다.

서비스 시간을 늘리면 대기 시간이 줄어들지만, 반대로 서비스 시간을 줄이면 대기 시간이

증가하는 Trade off가 있는 상황에서 최적의 정책을 찾기는 어려운 과제이다. 이에 따라 여러 Policy 조합을 다양한 static variable을 기준으로 비교하는 시뮬레이션 방법을 선택하였다. 시뮬레이션은 다양한 정책을 실제 운영 환경에서 테스트하고 결과를 비교함으로써, 엘리베이터 운영에서의 효과를 정량적으로 확인하는 데 효과적이었다.

시뮬레이션 결과 중에서도 최종적으로 선정된 Best Policy는 Policy 2로, 고층운행과 저층운행을 각각 3대씩 운용하는 것이었다. 이 결론은 다양한 정책의 특성을 고려하여 서비스 시간과 대기 시간 간의 Trade off를 효과적으로 균형을 맞춘 결과로 해석된다.

이러한 결론을 토대로, 엘리베이터 운영에서의 결과를 활용하는 방안과 기대효과를 살펴보면 다음과 같다. 먼저, Policy 2의 적용으로 서비스 품질이 최적화되어 이용자들의 만족도가 상승할 것이며, 건물 내에서의 효율적인 이동이 이루어져 생활의 편의성이 향상될 것으로 예측된다. 서비스 측면 뿐만 아니라 전력소비 절약 효과 또한 기대된다. 사람이 타고 다니는 균형추를 장착한 로프식 엘리베이터에서는 운행거리보다는 운행횟수가 전력 소비에 가장 큰 영향을 미친다. 엘리베이터는 큰 관성을 정지 상태에서 움직이기 시작하는 것에 많은 힘이 필요하다. 그리고 그 이후의 움직임에 대해서는 그다지 많은 전기에너지가 소요되지 않는다. 따라서 엘리베이터의 전력소비를 줄이는 가장 좋은 방법은 가능한 출발하는 횟수를 줄이는 것이다. Policy 2의 적용을 통해 기존에 비해서 최대 약 절반 가까이 엘리베이터 호출 횟수가 감소하기 때문에 전력 소비 감소의 효과를 기대한다.

다만 한계점 또한 존재한다. 데이터 수집 과정에서의 한계로 1층에서 출발하는 고객만을 대상으로 데이터를 수집하고 시뮬레이션 했다는 점에서 실제와 차이가 발생할 수 있을 것 같으며, 1층에서의 기준 만으로 elevator operating policy를 결정하기엔 어려움이 있어 보인다. 또한 시간대별 데이터를 합쳤을 때의 문제는 발생하지 않는지 호모지니어티 테스트를 실시하여 합칠 수 있음을 확인하였지만, 각각의 요일과 날짜에 대해서는 데이터 수가 너무 적기에 합쳐도 되는지 정량적으로 분석하지 못하고, 비슷할 것이라고 경험을 통해 결론을 내렸다. 마지막으로 Validation 과정에서 원래 데이터를 활용하여 시뮬레이션 돌린 결과와 원래 데이터 그 자체를 비교해야 하는데 원래 데이터는 엘리베이터 정차시간이랑 엘리베이터를 기다리는 시간만을 가지고 있어 이를 가지고 service time이나 waiting time으로 계산을 하지 못해 원래 데이터로 시뮬레이션 돌린 결과와 경험적인 수치가 일치하는지 확인하였다.

그러나 본 프로젝트는, 실제 상황을 그대로 modeling 하기 어렵다는 시뮬레이션 자체의 한계점 내에서 최대한 근거없는 가정을 배제하며 전반적인 시뮬레이션의 과정을 충실히 이행하였다. 3

명의 데이터 수집 방식을 최대한 동일하게 해 측정한 사람에 따라 데이터가 다르게 수집되지 않도록 하였으며, 다른 시각에 수집된 데이터에 대해서 Kruskal-Wallis test와 Mann-Whitney U test를 통해 합쳐도 되는지의 여부를 점검하였으며, 본 프로젝트에 적합한 형태로 모델을 변형하여 300번의 replication을 통해 결과를 얻었으며, Paired t-test 를 통해 최적의 시나리오를 도출해내었다. 추가적인 사람을 고용하여 1층에서 출발하는 사람들에 대한 데이터만을 수집하는 것이 아니라 각 층에서 모두 데이터를 수집하며, 더 많은 양의 데이터를 수집하여 시뮬레이션을 진행한다면 더욱 유의미한 결과를 얻을 수 있을 것으로 기대한다.

7. How we built code

- Simulator
- Run experiments
- Main

Simulator

```
class ElevatorSimulation:
  def __init__(self, policy, seed):
    # 1. Parameters
    self.elev_capacity = 10
    self.moving_time = 5.76
    self.passing_time = 2.67
    self.Q_LIMIT = 20
    self.time_end = 15*60 # 15min
    self.elev_policy = policy # All, Odd, Even, High, Low

    # Random number generation
    self.custs_arrival_gen = expon.rvs(loc=0.014535693, scale=6.863142713788785, size=1000, random_state=seed).tolist()
    self.custs_arrival_gen = [i for i in self.custs_arrival_gen if i >= 1 and i <60]
    self.elev_arrival_gen = lognorm.rvs(s=0.5982911680689725, loc=-1.474276076031476,
                                        scale=10.248234690476108, size=1000, random_state=seed).tolist()
    self.elev_arrival_gen = [i for i in self.elev_arrival_gen if i > 0 and i <60]
    self.stopping_time_gen = burr.rvs(c=2.5874763167784525, d=0.8045643981906201,
                                      loc=-0.6344383268940723, scale=21.480276631262164, size=1000,random_state=seed).tolist()
    self.stopping_time_gen = [i for i in self.stopping_time_gen if i >= 5 and i <90]

    self.floor = [2,3,4,5,6]
    self.floor_p = [0.02631578947,0.05789473684, 0.6210526316, 0.2315789474, 0.06315789474]

    # 2. Initialization
    self.initialize()
```

```
def initialize(self):
  # Simulation clock
  self.sim_time = 0.0

  # State variables
  self.elevator_status = [0, 0, 0, 0, 0, 0]
  self.num_in_q = 0
  self.time_last_event = 0.0
  self.button_pressed = [False, False, False, False, False, False]
  self.custs_in_elev = [0, 0, 0, 0, 0, 0]

  # Statistical counters
  self.num_custs_delayed = 0
  self.total_in_waiting = 0.0
  self.total_in_service = 0.0
  self.area_num_in_waiting = 0.0
  self.area_elev_status = [0.0] * 6

  # Event list (customer_arrival, elevator_arrival, customer_departure, end_simulation)
  self.event_df = pd.DataFrame(columns=['event_time', 'event_type', 'demand_floor', 'elevator_number'])
  self.event_df.loc[0] = [self.sim_time + self.custs_arrival_gen.pop(0), 'customer_arrival', int(np.random.choice(self.floor,1,p=self.floor_p)), None]
  self.event_df.loc[1] = [1.0e+30, 'elevator_arrival', None, None]
  self.event_df.loc[2] = [1.0e+30, 'customer_departure', None, None]
  self.event_df.loc[3] = [self.time_end, 'end_simulation', None, None]

  # Queue
  self.waiting_q_df = pd.DataFrame(columns=['time_arrival', 'destination'])
```

```
def timing(self):
  min_time_next_event = 1.0e+29
  next_event_type = ''

  # sort event_df
  self.event_df = self.event_df.sort_values(by='event_time', ascending=True)
  self.event_df.reset_index(drop=True, inplace=True)

  # Determine the event type of the next event to occur
  if self.event_df.iloc[0,0] < min_time_next_event:
    min_time_next_event = self.event_df.iloc[0, 0]
    next_event_type = self.event_df.iloc[0, 1]

  # check to see whether the event list is empty
  if next_event_type == '':
    print(f'Event list empty at time {self.sim_time}')
  else:
    pass

  self.next_event_type = next_event_type
  self.sim_time = min_time_next_event
```

```
def customer_arrival(self):
  current_custs_demand = self.event_df.iloc[0, 2]

  # Schedule next arrival & determine the customer's destination
  self.event_df.loc[len(self.event_df)] = [self.sim_time + self.custs_arrival_gen.pop(0), 'customer_arrival', int(np.random.choice(self.floor,1,p=self.floor_p)), None]

  # Increase waiting queue
  if self.num_in_q < self.Q_LIMIT:
    self.num_in_q += 1
  else:
    print(f'Overflow of the customer_arrival at time {self.sim_time}')
    return


  self.waiting_q_df.loc[len(self.waiting_q_df)] = [self.sim_time, current_custs_demand]
  self.waiting_q_df = self.waiting_q_df.sort_values('time_arrival', ascending=True)
  self.waiting_q_df.reset_index(drop=True, inplace=True)
  # Invoke button_pressor_evaluation
  self.button_pressor_evaluation()
```

```
def find_available_n_button_pressed(self, destination):
  All = [2, 3, 4, 5, 6]
  Low = [2, 3, 4]
  High = [5, 6]
  Odd = [3, 5]
  Even = [2, 4, 6]

  if destination in Low:
    available_idx = [idx for idx, value in enumerate(self.elev_policy) if value in ['All', 'Low']]
    button_pressed_idx = [idx for idx, value in enumerate(self.button_pressed) if value]
  elif destination in High:
    available_idx = [idx for idx, value in enumerate(self.elev_policy) if value in ['All', 'High']]
    button_pressed_idx = [idx for idx, value in enumerate(self.button_pressed) if value]
  if destination in Odd:
    available_idx += [idx for idx, value in enumerate(self.elev_policy) if value == 'Odd']
  elif destination in Even:
    available_idx += [idx for idx, value in enumerate(self.elev_policy) if value == 'Even']

  available_n_button_pressed = list(set(available_idx) & set(button_pressed_idx))
  return available_idx, button_pressed_idx, available_n_button_pressed
```

```
def button_pressor_evaluation(self):
  for idx, i in enumerate(self.waiting_q_df.iloc[:,1]):
    available_idx, button_pressed_idx, available_n_button_pressed = self.find_available_n_button_pressed(i)
    available_n_button_not_pressed = list(set(available_idx) - set(button_pressed_idx))
    # Checkpoint1
    # No
    if len(available_n_button_pressed) == 0:
      # Press button
      choosed_elev = random.choice(available_n_button_not_pressed)
      self.button_pressed[choosed_elev] = True
      # Schedule elevator arrival event
      self.event_df.loc[len(self.event_df)] = [self.sim_time + self.elev_arrival_gen.pop(0), 'elevator_arrival', None, choosed_elev]
      self.event_df = self.event_df.sort_values('event_time', ascending=True)
      self.event_df.reset_index(drop=True, inplace = True)
    # Yes
    else:
      # check how many customers before the customer
      cnt = 0
      for index, customer in self.waiting_q_df.iterrows():
        if idx == index:
          break
        cnt +=1
      # Checkpoint2
      # No
      if cnt >= 5:
        # Checkpoint3
        # Yes
        if len(available_n_button_not_pressed) > 0:
          # Press button
          choosed_elev = random.choice(available_n_button_not_pressed)
          self.button_pressed[choosed_elev] = True
          # Schedule elevator arrival event
          self.event_df.loc[len(self.event_df)] = [self.sim_time + self.elev_arrival_gen.pop(0), 'elevator_arrival', None, choosed_elev]
          self.event_df = self.event_df.sort_values('event_time', ascending=True)
          self.event_df.reset_index(drop=True, inplace = True)
```

```
def elevator_arrival(self):
  elev_no = self.event_df.iloc[0, 3]
  customer_in_elev = 0
  stop_floor = []

  # Make the elevator busy
  self.elevator_status[elev_no] = 1
  # Make the button unpressed
  self.button_pressed[elev_no] = False

  # Select customers who will take the elevator
  available_floor = self.get_available_floors(elev_no)
  for index, customer in self.waiting_q_df.iterrows():
    if customer_in_elev == self.elev_capacity:
      break
    destination = customer['destination']
    if destination in available_floor:
      self.num_in_q -= 1
      customer_in_elev += 1
      stop_floor.append(destination)
      time_custs_arrival = customer['time_arrival']
      waiting = self.sim_time - time_custs_arrival
      self.total_in_waiting += waiting

      # Drop customers in the queue
      self.waiting_q_df = self.waiting_q_df.iloc[1:,:]
      self.waiting_q_df.reset_index(drop=True, inplace=True)
```

```
        # Calculate service time
        service_time = []
        stop_floor_unique = sorted(list(set(stop_floor)))
        service_time_unique = {}
        last_floor = 1
        for floor in stop_floor_unique:
          if last_floor == 1:
            passing_num = floor - 2
          else:
            passing_num = floor - last_floor
          service_time_unique[floor] = self.stopping_time_gen.pop(0) + self.passing_time*passing_num
          last_floor = floor

        for floor in stop_floor:
            service_time.append(service_time_unique[floor])
        maximum_service_time = max(service_time)

        service_time.append(self.moving_time)
        self.total_in_service += sum(service_time)

        # Schedule customer departure event
        self.event_df.loc[len(self.event_df)] = [self.sim_time + maximum_service_time, 'customer_departure', None, elev_no]
        self.event_df = self.event_df.sort_values('event_time', ascending=True)
        self.event_df.reset_index(drop=True, inplace = True)

    # Invoke button pressor evaluation
    self.button_pressor_evaluation()

    # Update custs_in_elev
    self.custs_in_elev[elev_no] = customer_in_elev
```

```
def get_available_floors(self, elev_no):
  available_floor = []
  elev_policy = self.elev_policy[elev_no]
  if elev_policy == 'All':
    available_floor = [2, 3, 4, 5, 6]
  elif elev_policy == 'High':
    available_floor = [5, 6]
  elif elev_policy == 'Low':
    available_floor = [2,3,4]
  elif elev_policy == 'Odd':
    available_floor = [3, 5]
  elif elev_policy == 'Even':
    available_floor = [2, 4, 6]
  return available_floor
```

```
def customer_departure(self):
  elev_no = self.event_df.iloc[0, 3]
  # record num_custs_delayed
  self.num_custs_delayed += self.custs_in_elev[elev_no]

  # Make the elevator idle
  self.elevator_status[elev_no] = 0

  # Make custs_in_elev to zero
  self.custs_in_elev[elev_no] = 0


def update_time_avg_stats(self):
  time_since_last_event = self.sim_time - self.time_last_event
  self.time_last_event = self.sim_time

  # Update area under the number-in-waiting-line function
  self.area_num_in_waiting += self.num_in_q * time_since_last_event

  # Update area under elevator-busy indicator function
  for i in range(6):
    self.area_elev_status[i] += self.elevator_status[i] * time_since_last_event
```

```
def report(self):
  '''
  print('------------------------------------------------------------------------')
  print(f'Average waiting time in GF: {self.total_in_waiting / self.num_custs_delayed}')
  print(f'Average service time in elevator: {self.total_in_service / self.num_custs_delayed}')
  print(f'Average total time in elevator: {(self.total_in_waiting + self.total_in_service) / self.num_custs_delayed}')

  print(f'Average number in waiting queue: {self.area_num_in_waiting / self.sim_time}')

  elev_utilization = [self.area_elev_status[i] / self.sim_time for i in range(6)]
  for i in range(6):
    print(f'Elevator {i + 1} utilization: {elev_utilization[i]}')

  avg_elev_utilization = sum(elev_utilization) / 6
  print(f'Average elevator utilization: {avg_elev_utilization}')
  print('------------------------------------------------------------------------')
  '''
```

```
def save_results(self):
  df = pd.DataFrame(columns=['avg_waiting_time', 'avg_service_time', 'avg_total_time', 'avg_num_in_q',
                             'elev1_utilization', 'elev2_utilization', 'elev3_utilization', 'elev4_utilization', 'elev5_utilization', 'elev6_utilization',
                             'avg_elev_utilization'])
  avg_waiting_time = self.total_in_waiting / self.num_custs_delayed
  avg_service_time = self.total_in_service / self.num_custs_delayed
  avg_total_time = (self.total_in_waiting + self.total_in_service) / self.num_custs_delayed
  avg_num_in_q = self.area_num_in_waiting / self.sim_time
  elev_utilization = [self.area_elev_status[i] / self.sim_time for i in range(6)]
  avg_elev_utilization = sum(elev_utilization) / 6
  df.loc[0] = [avg_waiting_time, avg_service_time, avg_total_time, avg_num_in_q,
               elev_utilization[0], elev_utilization[1], elev_utilization[2], elev_utilization[3], elev_utilization[4], elev_utilization[5],
               avg_elev_utilization]
  return df
```

```
def run_simulation(self):
  # 3. Run the simulation while more delays are still needed
  while 1:
    # Determine next event
    self.timing()
    # Update time-average statistical accumulators
    self.update_time_avg_stats()
    # Invoke the appropriate event function
    if self.next_event_type == 'customer_arrival':
      self.customer_arrival()
      self.event_df = self.event_df.iloc[1:,:] # delete first row
      self.event_df = self.event_df.sort_values('event_time', ascending=True)
      self.event_df.reset_index(inplace = True, drop=True)
    elif self.next_event_type == 'elevator_arrival':
      self.elevator_arrival()
      self.event_df = self.event_df.iloc[1:,:] # delete first row
      self.event_df = self.event_df.sort_values('event_time', ascending=True)
      self.event_df.reset_index(inplace = True, drop=True)
    elif self.next_event_type == 'customer_departure':
      self.customer_departure()
      self.event_df = self.event_df.iloc[1:,:] # delete first row
      self.event_df = self.event_df.sort_values('event_time', ascending=True)
      self.event_df.reset_index(inplace = True, drop=True)
    elif self.next_event_type == 'end_simulation':
      # 4. Invoke the report generator and end the simulation
      self.report()
      self.result = self.save_results()
      break
```

Run experiments

```
class experiment:
  def __init__(self, policy_list, repeat_num):
    self.policy_list = policy_list
    self.repeat_num = repeat_num
    self.seeds = np.random.randint(10000, size=repeat_num)
    self.results = None

  def run(self):
    for idx, policy in enumerate(tqdm(self.policy_list)):
      print('₩n')
      for j in range(self.repeat_num):
        if j%10 == 0:
          print(f'{j+1} / {self.repeat_num} is processing')
        self.policy = policy
        self.seed = self.seeds[j]
        elevator_sim = ElevatorSimulation(policy=self.policy, seed = self.seed)
        elevator_sim.run_simulation()
        self.result_tmp = elevator_sim.result
        self.result_tmp['policy'] = idx
        self.result_tmp['seed'] = self.seed

        if self.results is None:
          self.results = self.result_tmp
        else:
          self.results = pd.concat([self.results, self.result_tmp], axis=0)
```

Main

```
policy_list = [['All', 'All', 'All', 'All', 'All', 'All'],
               ['Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even'],
               ['High', 'Low', 'High', 'Low', 'High', 'Low'],
               ['All', 'All', 'Odd', 'Even', 'Odd', 'Even'],
               ['All', 'All', 'High', 'Low', 'High', 'Low'],
               ['All', 'All', 'All', 'All', 'Odd', 'Even'],
               ['All', 'All', 'All', 'All', 'High', 'Low'],
               ['All', 'All', 'High', 'Low', 'Odd', 'Even']]
repeat_num = 100

exp = experiment(policy_list, repeat_num)
exp.run()
```