Assignment 3

Group 9. Rinus van Grunsven (10755373), Florens Douwes (11254483), Imad Lotfi (14651610)

23 October 2022

Exercise 3.1

A)

To solve this problem, first the Pulp package is imported. The parameters (jobs, duration, release times etc.) are created and set to the values as described in the assignment. Then, the decision variables x (starting times) and y (binary variable that indicates wheter a certain job is being processed on a specific time or not) are created. Following that, the objective (sum of all costs (z)) is set. This can be summarized as the data part (as asked in the assignment) and can be easily adjusted to the context of a new problem.

From here, several constraints of the model are added. These constraints include facts as that a job can only start after it is released, that the machine can only work on 1 job at a time and that a job has to be finished before its due date. After all the constraints are added, the build-in solver is called to solve the problem.

In [1]:
```
#import package + create problem (minimization of total costs)
import pulp
SMS_problem = pulp.LpProblem(name="SMS_problem",sense=pulp.LpMinimize)
```

In [2]:
```
#set parameters + objective
jobs = ['job1', 'job2', 'job3','job4','job5','job6','job7','job8','job9','job10']
release_time = [3,4,7,11,10,0,0,10,0,15]
duration = [4,5,3,5,7,1,0,3,2,10]
date_of_due = [11,12,20,25,20,10,30,30,10,20]
n = len(jobs)
M = 1000
costs = 1
x = [pulp.LpVariable(name = f'x_{i}', lowBound = 0, cat = 'Integer') for i in range(n)]
y = [[pulp.LpVariable(name = f'y_{i},{j}', cat = 'Binary') for j in range(n)] for i in range(n)]
i = 1
j = 2
z = [pulp.LpVariable(name = f'z_{i}', lowBound = 0, cat = 'Integer') for i in range(n)]
SMS_problem += sum([z[i] for i in range(n)]), 'sum total costs'
print(SMS_problem)
```

In [3]:
```
#add constraints
for i in range(n):
    SMS_problem += x[i] >= release_time[i], f'job_{i}_start_after_release_{i}'
for i in range (n):
    for j in range(n):
        if i == j:
            continue
        SMS_problem += y[i][j] + y[j][i] ==1, f'fix_job_{i}_before_job_{j}'
        SMS_problem += x[i] + duration [i] <= x[j] + M*(1-y[i][j])
for i in range(n):
    SMS_problem += z[i] >= x[i] + duration[i] - date_of_due[i]
for i in range(n):
    SMS_problem += z[i] >= 0
```

In [ ]:
```
#solve the ILP
SMS_problem.solve()
```

```
In [5]: #print results
        print("Optimization status:", pulp.LpStatus[SMS_problem.status])
        print("The starting times: ", [x[i].value() for i in range(n)])
        print ("Objective value: ", SMS_problem.objective.value())

        Optimization status: Optimal
        The starting times:  [3.0, 7.0, 19.0, 22.0, 12.0, 2.0, 30.0, 27.0, 0.0, 30.0]
        Objective value:  24.0
```

```
In [ ]: #print values of all variables
        for a in SMS_problem.variables():
            print(a.name, "=", a.varValue)
```

The above output and the values of all the variables can be interpreted as follows:

- The gained solution is the optimal solution. The optimal schedule is [9,6,1,2,5,3,4,8,7,10] with total costs (objective value) 24.0.
- The starting times per job (in the order of the jobs in the optimal schedule above) are: [0.0, 2.0, 3.0, 7.0, 12.0, 19.0, 22.0, 27.0, 30.0, 30.0]
- The finishing times (in the order of the jobs in the optimal schedule above) are: [2.0, 3.0, 7.0, 12.0, 19.0, 22.0, 27.0, 30.0, 30.0, 40.0]

- The tardiness per job (in the order of the jobs in the optimal schedule above) is: [0.0, 0.0, 0.0, 0.0, 0.0, 2.0, 2.0, 0.0, 0.0, 20.0]

B)

The additional requirement is added to the model as an extra constraint in the form of an if-else statement, as can be seen below:

```
In [9]: #part b
        #add constraints
        for i in range(n):
            SMS_problem += x[i] >= release_time[i], f'job_{i}_start_after_release_{i}'
        for i in range (n):
            for j in range(n):
                if i == j:
                    continue
                SMS_problem += y[i][j] + y[j][i] ==1, f'fix_job_{i}_before_job_{j}'
                if (i in [0,2,3,5,9] and j in [1,4,6,7,8]) or (i in [1,4,6,7,8] and j in [0,2,3,5,9]):
                    SMS_problem += x[i] + duration[i] + 1 <= x[j] + M*(1-y[i][j])
                else:
                    SMS_problem += x[i] + duration[i] <= x[j] + M*(1-y[i][j])
        for i in range(n):
            SMS_problem += z[i] >= x[i] + duration[i] - date_of_due[i]
        for i in range(n):
            SMS_problem += z[i] >= 0
```

Except for this small addition to the constraints part, the rest of the model has remained the same as in part a. This model resulted in the following solution:

```
In [11]: #print results
         print("Optimization status:", pulp.LpStatus[SMS_problem.status])
         print("The starting times: ", [x[i].value() for i in range(n)])
         print ("Objective value: ", SMS_problem.objective.value())

         Optimization status: Optimal
         The starting times:  [10.0, 4.0, 14.0, 17.0, 26.0, 0.0, 26.0, 23.0, 2.0, 34.0]
         Objective value:  40.0
```

As before, this can be interpreted as follows:

- The gained solution is the optimal solution. The optimal schedule is [6,9,2,1,3,4,8,7,5,10] with total costs (objective value) of 40.0.
- The starting times per job (in the order of the jobs in the optimal schedule above) are: [0.0, 2.0,

4.0, 10.0, 14.0, 17.0, 23.0, 26.0, 26.0, 34.0]
- The finishing times (in the order of the jobs in the optimal schedule above) are: [1.0, 4.0, 9.0, 14.0, 17.0, 22.0, 26.0, 26.0.0, 33.0, 44.0]

- The tardiness per job (in the order of the jobs in the optimal schedule above) is: [0.0, 0.0, 0.0, 3.0, 0.0, 0.0, 0.0, 0.0, 13.0, 24.0]
- The corresponding sieve types (in the order of the jobs in the optimal schedule above) are: [1, 2, 2, 1, 1, 1, 2, 2, 2, 1].

Exercise 3.2

The project is calculated in Excel, as shown in Figure 1 of the appendix. The expected durations for each activity are displayed in columns B16 through H16.The inverse transformation method must be used as these are expected durations.All activity durations are independent and distributed exponentially. In addition, the duration follows an exponential distribution with a parameter of 1/4 for the rate. The inverse transformation method for exponential distribution is thus required. The calculation in Excel is: '-ln(rand()) / "lambda" '. Lambda is the same the rate parameter so this will be 1/4. This function is used to calculate the realizations of activity durations for the seven activities in de cells B24 until H10023. So for example, the calculation made in cell B20 is '=-(LN(RAND())/(1/B16))'. Realizations of finish time activities are calculated in cells I24 untill O10023. This is done by adding the duration of the activity with the duration of the former one. So for example, in cell J20 it is '=I20+C20'. However, in cell M20 it is '=MAX(J20;K20)+F20'. This is because the activity five has to wait for both activity two as well as three to finish. The max makes sure that the longest of those two activities gets chosen as activity five can only start once both previous activities are finished.

a)

We can assume that the central limit theorem applies since the project has to be simulated 10000 times. The expected project finish time will be the average project finish time of those 10000 simulations. That calculation is shown in appendix Table 1 and the result is 12,089 (see appendix Figure 2).

b)

The confidence interval is calculated using the t-test. The use of the t-test is possible as the we can assume a normal distribution (as described above) and we don't know anything about the population standard deviation, but do know the sample standard deviation. The confidence interval looks as follows:

$$[\overline{X} - t_{a/2,n-1}\frac{s}{\sqrt{n}}, \overline{X} + t_{a/2,n-1}\frac{s}{\sqrt{n}}]$$

Where $\overline{X}$ = expected project finish time (which is the average expected project finish time), n = number of sample values, s = sample standard deviation, and t is the t-distribution.

The calulations are shown in appendix Table 1. These give a confidence interval of [13,859; 14,058] (see appendix Figure 2). This means that we are 95% confident that the mean lies within this interval. So, if we would run this project several times and calculate the project finishing time, the mean would lie in this interval for 95% of the cases.

c)

The probability that the expected project time takes longer is calculated in Excel with the function '=(COUNTIF(P24:P10023;">12")/10000)'. P24:P10023 are the 10000 expected project durations. This results in a probability score of 0,5815 (is 58,15%). The confidence interval is then calculated in the same way as described in exercise 3.3 b. The results are shown in appendix Figure 3. The 95% confidence interval is thus: [14,037;14,244]

Exercise 3.3

a) We can simulate the hotel room rent in excel by using multiple rows for random, binomial formulas. We arranged the data in a table (see appendix 3.3a table). With a simulation budget of 100000, we will use 100000 / 9 = 11111 rows.

When graphed (appendix 3.3b), we can see that the 140 mark is the cost with the highest estimate revenue.

We use a t-test for these calculations. The t-test is done in excel by estimating the mean, std. dev, and t value. From this, a lower- and upperbound confidence interval is calculated. These are also graphed in 3.3b.

b) To execute the steps in "ranking and selection, option 2" we can reuse the sheet created in 3.3a. With a starting budget of 900, and 9 prices, we first simulate 900 / 9 = 100 rows. These are shown in appendix 3.3c.

Next, we perform pairwise t-test for each price. We can do this in Excel by creating a 9 by 9 table containing the test: $y(\pi) \leq y(\pi') - \beta_{1-\alpha}\frac{\sqrt{s^2(\pi)+s^2(\pi')}}{\sqrt{(N)}}$. The results of these tests are in appendix 3.3d. With this, we can reject the prices 100, 100, 160, 170, 180.

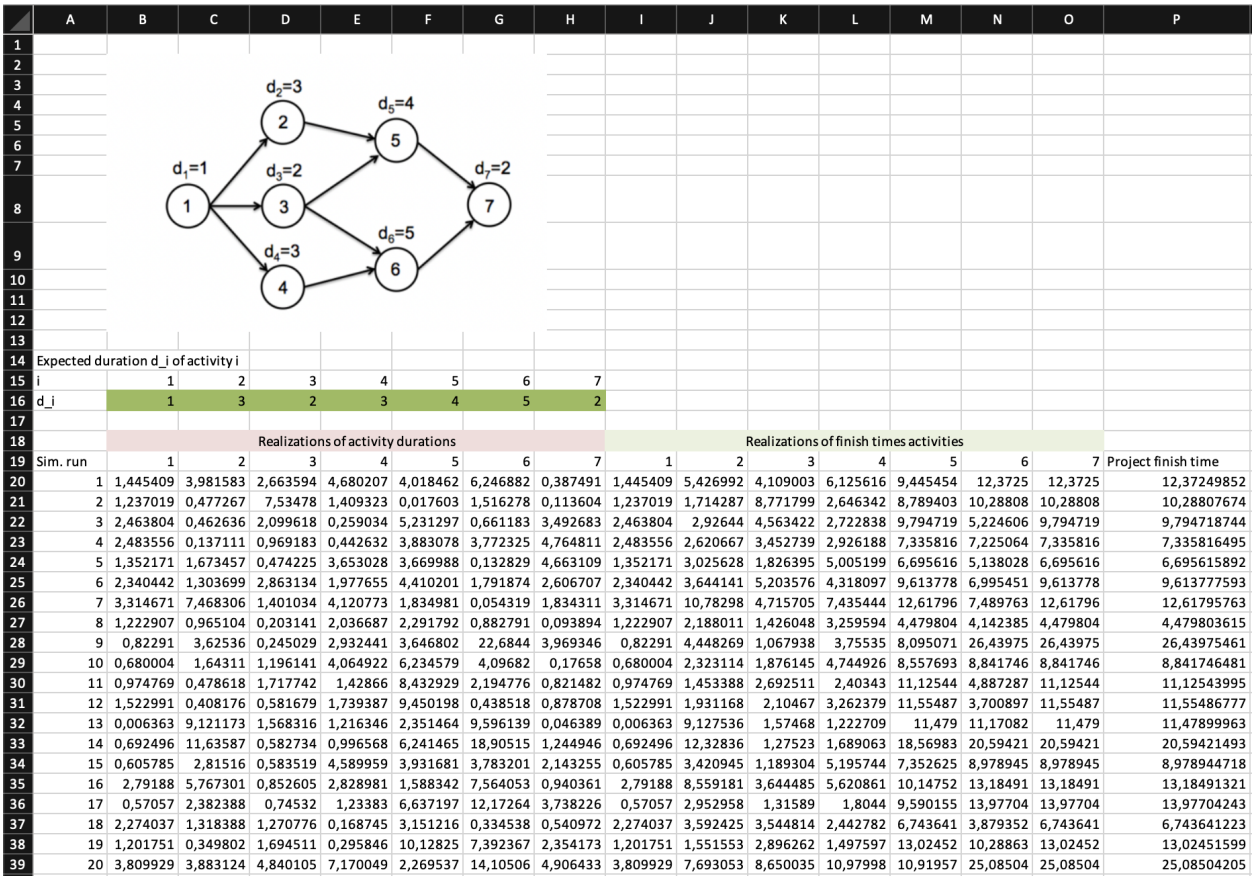
To continue, we execute step 4, which is to perform the rest of the testing budget. This is 9100 / 9 = 1011 rows. The end result is shown in appendix 3.3e and 3.3f. Now, the maximum revenue is estimated to be with 130.

An important note is that these values are generated by a random number generated, which resets between each simulation. Therefore, these test may show different results. Nonetheless, if simulated enough, the result should weight to a single price point eventually.

Appendix

Table 1: Formulas used

Name	Calclulation
Sample average	=AVERAGE(Q23:Q10022)
N	=COUNT(Q23:Q10022)
95 percentile of t-dist with n-1	=T.INV(0,95;U28-1)
Sample standard deviation (s)	=STDEV.S(Q23:Q1122)
Left CI bound	=U25-U29*U30/SQRT(U28)
Right CI bound	=U25+U29*U30/SQRT(U28)
Test statistic T	=(12-T39)/(T44/SQRT(T42))
P-value	=TDIST(T48;(T42-1);1)



Expected duration d_i of activity i																
i	1	2	3	4	5	6	7									
d_i	1	3	2	3	4	5	2									
	Realizations of activity durations							Realizations of finish times activities								
Sim. run	1	2	3	4	5	6	7	1	2	3	4	5	6	7	Project finish time	
1	1,445409	3,981583	2,663594	4,680207	4,018462	6,246882	0,387491	1,445409	5,426992	4,109003	6,125616	9,445454	12,3725	12,3725	12,37249852	
2	1,237019	0,477267	7,53478	1,409323	0,017603	1,516278	0,113604	1,237019	1,714287	8,771799	2,646342	8,789403	10,28808	10,28808	10,28807674	
3	2,463804	0,462636	2,099618	0,259034	5,231297	0,661183	3,492683	2,463804	2,92644	4,563422	2,722838	9,794719	5,224606	9,794719	9,794718744	
4	2,483556	0,137111	0,969183	0,442632	3,883078	3,772325	4,764811	2,483556	2,620667	3,452739	2,926188	7,335816	7,225064	7,335816	7,335816495	
5	1,352171	1,673457	0,474225	3,653028	3,669988	0,132829	4,663109	1,352171	3,025628	1,826395	5,005199	6,695616	5,138028	6,695616	6,695615892	
6	2,340442	1,303699	2,863134	1,977655	4,410201	1,791874	2,606707	2,340442	3,644141	5,203576	4,318097	9,613778	6,995451	9,613778	9,613777593	
7	3,314671	7,468306	1,401034	4,120773	1,834981	0,054319	1,834311	3,314671	10,78298	4,715705	7,435444	12,61796	7,489763	12,61796	12,61795763	
8	1,222907	0,965104	0,203141	2,036687	2,291792	0,882791	0,093894	1,222907	2,188011	1,426048	3,259594	4,479804	4,142385	4,479804	4,479803615	
9	0,82291	3,62536	0,245029	2,932441	3,646802	22,6844	3,969346	0,82291	4,448269	1,067938	3,75535	8,095071	26,43975	26,43975	26,43975461	
10	0,680004	1,64311	1,196141	4,064922	6,234579	4,09682	0,17658	0,680004	2,323114	1,876145	4,744926	8,557693	8,841746	8,841746	8,841746481	
11	0,974769	0,478618	1,717742	1,42866	8,432929	2,194776	0,821482	0,974769	1,453388	2,692511	2,40343	11,12544	4,887287	11,12544	11,12543995	
12	1,522991	0,408176	0,581679	1,739387	9,450198	0,438518	0,878708	1,522991	1,931168	2,10467	3,262379	11,55487	3,700897	11,55487	11,55486777	
13	0,006363	9,121173	1,568316	1,216346	2,351464	9,596139	0,046389	0,006363	9,127536	1,57468	1,222709	11,479	11,17082	11,479	11,47899963	
14	0,692496	11,63587	0,582734	0,996568	6,241465	18,90515	1,244946	0,692496	12,32836	1,27523	1,689063	18,56983	20,59421	20,59421	20,59421493	
15	0,605785	2,81516	0,583519	4,589959	3,931681	3,783201	2,143255	0,605785	3,420945	1,189304	5,195744	7,352625	8,978945	8,978945	8,978944718	
16	2,79188	5,767301	0,852605	2,828981	1,588342	7,564053	0,940361	2,79188	8,559181	3,644485	5,620861	10,14752	13,18491	13,18491	13,18491321	
17	0,57057	2,382388	0,74532	1,23383	6,637197	12,17264	3,738226	0,57057	2,952958	1,31589	1,8044	9,590155	13,97704	13,97704	13,97704243	
18	2,274037	1,318388	1,270776	0,168745	3,151216	0,334538	0,540972	2,274037	3,592425	3,544814	2,442782	6,743641	3,879352	6,743641	6,743641223	
19	1,201751	0,349802	1,694511	0,295846	10,12825	7,392367	2,354173	1,201751	1,551553	2,896262	1,497597	13,02452	10,28863	13,02452	13,02451599	
20	3,809929	3,883124	4,840105	7,170049	2,269537	14,10506	4,906433	3,809929	7,693053	8,650035	10,97998	10,91957	25,08504	25,08504	25,08504205	

Figure 1: Screenshot 3.2a

Sample average		12,08881
Confidence interval (CI) calculation		
n		10000
95 percentile of t-dist with n-1		1,645006
Sample standard deviation (s)		5,733531
Left CI bound		11,99449
Right CI bound		12,18313

Figure 2: Screenshot 3.2a2

Sample average		14,14021503
Confidence interval (CI) calculation		
n		10000
95 percentile of t-dist with n-1		1,645006033
Sample standard deviation (s)		6,291866011
Left CI bound		14,03671345
Right CI bound		14,2437166

Figure 3: Screenshot 3.2c

	A	B	C	D	E	F	G	H	I	J
1	**Optimal hotel prices: 3.3a**									
2										
3	Prices	100	110	120	130	140	150	160	170	180
4										
5	Mean	1135,82936	1172,13212	1199,568	1210,86851	1213,82054	1203,48303	1172,17172	1135,28665	1086,01026
6	Stddev	222,344583	246,604774	265,545568	291,38632	308,611378	327,973513	344,333622	359,678607	367,260498
7	N	11111	11111	11111	11111	11111	11111	11111	11111	11111
8	t	1,96017753	1,96017753	1,96017753	1,96017753	1,96017753	1,96017753	1,96017753	1,96017753	1,96017753
9	Lowerbound 95% CI	1131,69465	1167,54627	1194,62992	1205,4499	1208,08161	1197,38405	1165,7685	1128,59808	1079,18069
10	Upperbound 95% CI	1139,96407	1176,71798	1204,50607	1216,28712	1219,55947	1209,58202	1178,57494	1141,97523	1092,83983
11										
12		Price								
13	Simulation run	100	110	120	130	140	150	160	170	180
14	1	700	1320	1440	1690	1820	1350	1280	1020	1620
15	2	1000	1210	1560	1170	840	1050	1120	1700	1620
16	3	900	1210	960	910	840	1050	960	1530	900
17	4	1200	880	1200	1430	1260	1950	800	1360	900
18	5	1200	1100	1320	1690	1260	900	1440	680	1440
19	6	1000	990	960	910	1120	1650	960	850	360
20	7	1300	1540	1080	910	1400	1800	1760	1360	900
21	8	900	1210	840	780	980	1350	1280	1020	1620
22	9	1100	1320	1320	910	1680	1500	800	510	1440
23	10	1500	990	1200	1170	1540	1050	1280	850	720
24	11	700	770	1920	1170	1120	1350	800	1530	1260
25	12	1100	1100	1440	1690	1400	1500	1120	340	540
26	13	1100	1100	1200	1560	1540	1500	960	1190	540
27	14	1500	880	1200	1430	980	1350	1280	1700	1260
28	15	1100	1320	1320	1170	560	1200	1760	1360	900
29	16	1000	1210	1440	1170	420	1350	1280	1190	1260
30	17	1200	1210	1200	1300	840	1500	1440	850	1260

Figure 4: Screenshot 3.3a

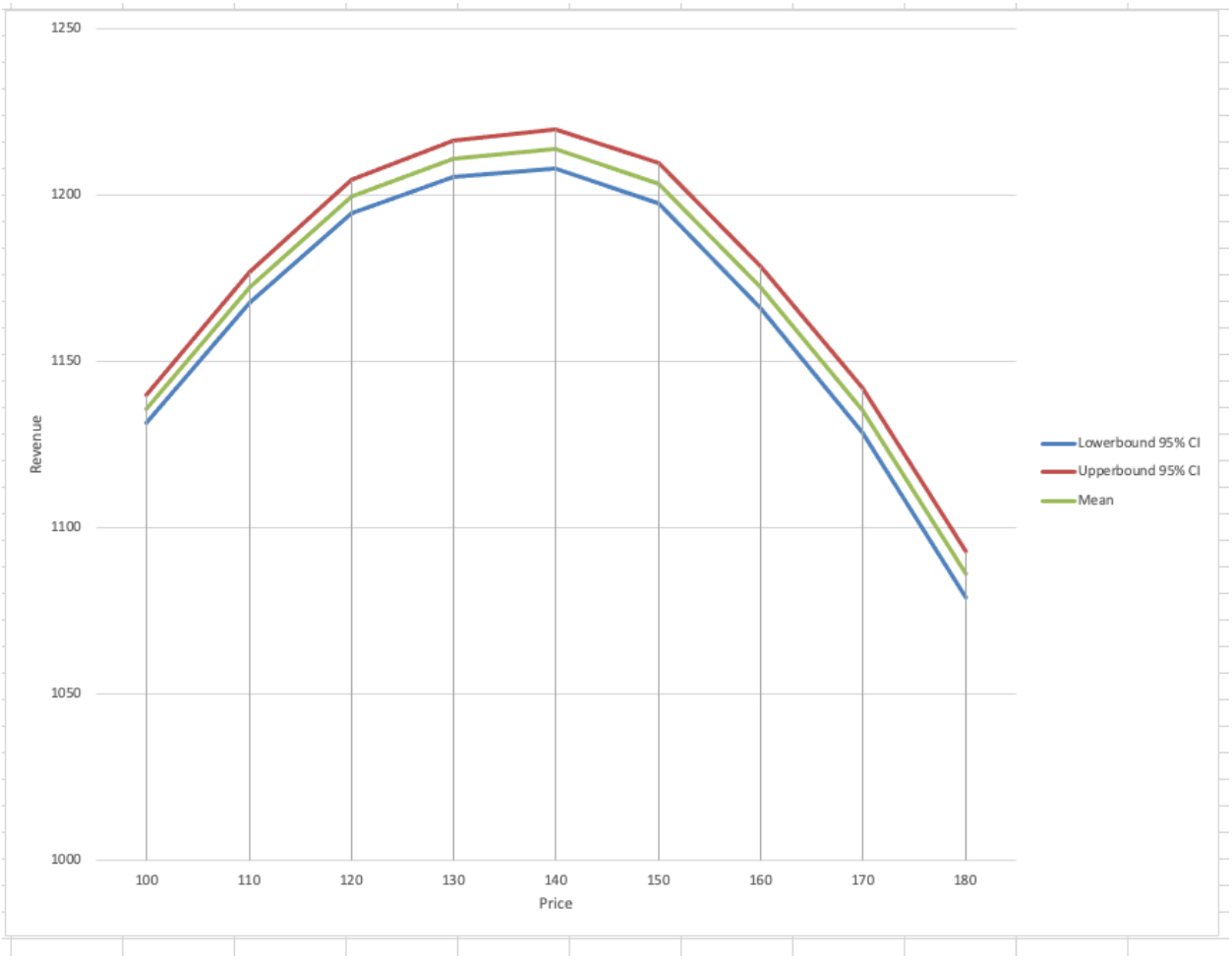


Figure 5: Screenshot 3.3b

	A	B	C	D	E	F	G	H	I	J
1	**Optimal hotel prices: 3.3b**									
2										
3	Prices	100	110	120	130	140	150	160	170	180
4										
5	Mean	1119	1171,5	1183,2	1280,5	1257,2	1198,5	1166,4	1098,2	1058,4
6	Stddev	235,999914	220,762567	260,918657	297,580733	300,451109	358,97265	315,356204	376,98533	384,851329
7	N	100	100	100	100	100	100	100	100	100
8	t	1,98421695	1,98421695	1,98421695	1,98421695	1,98421695	1,98421695	1,98421695	1,98421695	1,98421695
9	Lowerbound 95% CI	1072,1725	1127,69592	1131,42808	1221,45353	1197,58398	1127,27204	1103,82649	1023,39793	982,037147
10	Upperbound 95% CI	1165,8275	1215,30408	1234,97192	1339,54647	1316,81602	1269,72796	1228,97351	1173,00207	1134,76285
11										
12		Price								
13	Simulation run	100	110	120	130	140	150	160	170	180
14	1	1000	1210	1080	1170	1120	1500	960	1530	1260
15	2	900	1210	960	910	840	1200	960	680	1440
16	3	1200	1210	1320	1820	1120	1050	1120	1360	900
17	4	1500	1100	720	1170	840	750	1120	850	900
18	5	1100	1320	1320	1300	980	750	1280	1020	720
19	6	1600	1210	1320	1170	1260	1050	1120	1190	540
20	7	900	1210	1680	1430	1400	1500	1440	1020	1260
21	8	1400	990	1320	1430	1120	1350	960	850	1800
22	9	800	1320	1560	1430	980	0	800	1360	1800
23	10	1100	880	1200	910	1400	1500	800	340	1260
24	11	1000	1430	960	1170	1400	1650	1760	1020	720
25	12	1000	880	840	1300	1260	1500	1120	1870	1260
26	13	1300	1100	1440	1430	980	900	1120	1870	1260
27	14	1500	1320	960	1040	1120	900	320	850	900
28	15	1000	990	1200	1170	1120	1350	1920	1190	1440
29	16	900	1100	720	1560	1400	1500	1440	680	1440
30	17	1100	1430	1200	1040	1400	1200	1600	1700	360
31	18	1100	1100	1080	780	1260	1200	1120	850	1620

Figure 6: Screenshot 3.3c

t-test reject if TRUE									
	100	110	120	130	140	150	160	170	180
100	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
110	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
120	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
130	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
140	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
150	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
160	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
170	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
180	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE

a	0,05
a*	0,00639115
b	2,50152958

	D	E	F	G
0	120	130	140	150
6	1200,70737	1222,10947	1215,14105	1204,35789
3	271,115835	287,2448	308,923401	327,330215
0	2375	2375	2375	2375
5	1,96096376	1,96096376	1,96096376	1,96096376
4	1189,79817	1210,55128	1202,71055	1191,18674
6	1211,61656	1233,66767	1227,57155	1217,52905

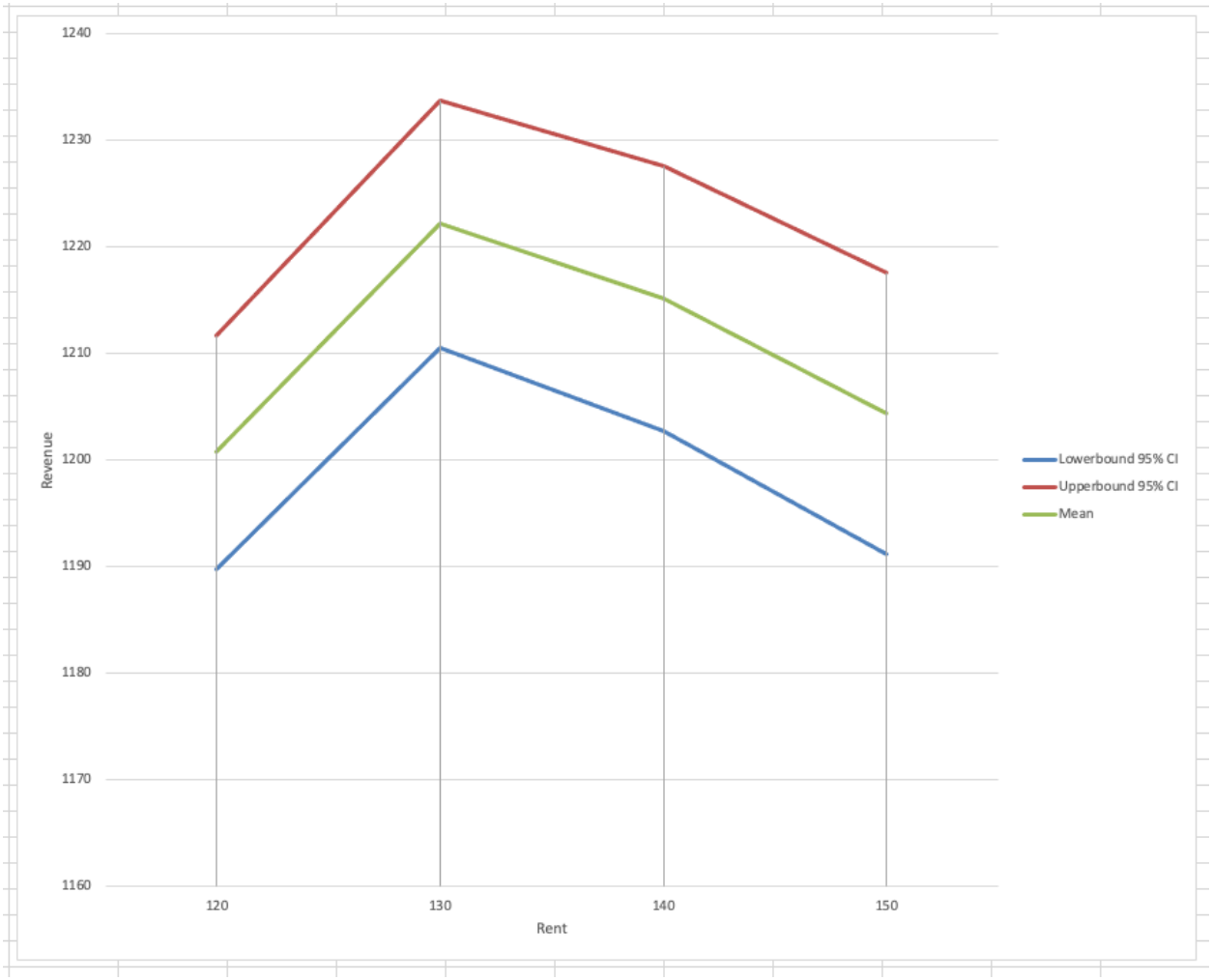


Figure 7: Screenshot 3.3f