Mathematics Analysis and Approaches

Internal Assessment

Optimising the Number of U-Bikes at a U-Bike Station

Page Count: 24

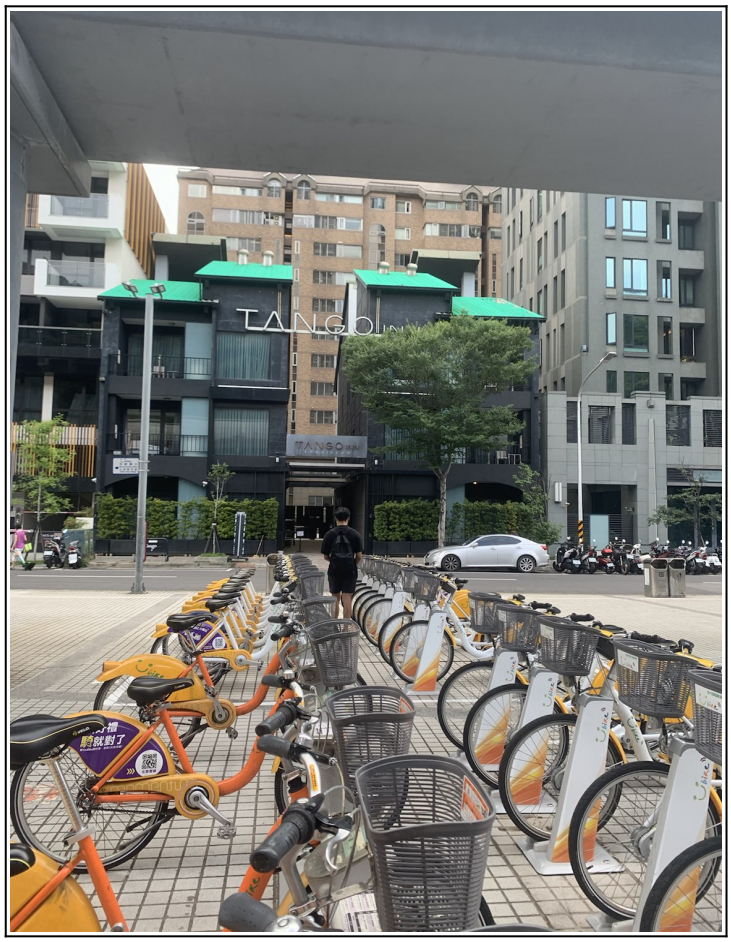

Image 1: U-Bike Station at Jiantan Station. *Source: Original photograph taken with Pixel phone.*

Introduction & Rationale

In Taiwan, specifically Taipei, U-Bike is a public city bicycle sharing service released in 2016 that offers a convenient way of travelling and is incredibly easy to use with the EasyCard. To me, it is the most cost-efficient, healthy, and enjoyable way to travel around the city without the hassle of carrying your bike around: simply park it and move on. As a declaration of its clear success, the government is introducing U-Bike 2.0, which inspired me to start to think about the numbers regarding how many people arrive or leave a U-Bike Station in a certain area. From this, I pondered how the government decides the number of U-Bikes to place at a certain area and the possible optimisation of the number of U-Bikes at U-Bike Stations. Apart from my interest in optimising, the rationale behind this is to attempt to reduce the amount of unnecessary spendings by the

government and help direct resources to sectors that require it – an attempt to steer away from the concept of squandering. In order to execute this, I will be focusing on two discrete variables, how many of the U-Bikes leave and how many of the U-Bikes arrive. Through these discrete variables, I intend to then predict the probability discrete random variables occurring in a set time period through Poisson distribution and from there, attempt to optimise the initial number of U-Bikes and the number of available U-Bike slots that should be present at a U-Bike Station.

What is the Poisson Distribution?

Poisson distribution, invented by Denis Poisson, is a method of finding the probability of a number of events occurring in a set time, distance, area, or volume. Essentially, the model allows us to express the probability of a discrete random variable occurring in a fixed period of time or space. To calculate this probability, we need the discrete, average number of times an event occurs in the same fixed period of time or space, λ . The formula is as follows:

$$P(X=x)=\frac{\lambda^{x}\cdot \mathrm{e}^{-\lambda}}{x!}$$

where λ is the mean number of occurrences in an interval, X is the discrete random variable of the number of events in an interval, x is the number of occurrences, e is Euler's Number and ! is the factorial function. Once we have a value for λ , we can substitute in any discrete value for *x*, and obtain the probability of *x* number of events occurring in the same time, distance, area or volume period that was used to find λ . There are, however, a number of conditions that must be met in order to use Poisson distribution: 1) the events, *A* and *L*, must occur independently; 2) the occurrence rate is constant; 3) the probability of the events occurring is proportional to the time period; 4) an event can occur any number of times during a time period, including zero.

Example of Poisson Distribution

A completely theoretical example to the use of Poisson distribution is as follows: Volcanoes in Hawaii erupt at a rate of approximately 11 every year. Therefore, what is the probability in a randomly selected year that there are no eruptions?

To tackle this question, we have to first assume that all the conditions are met. Then, we can substitute the values $\lambda = 11$ and $x = 0$ into the Poisson distribution formula. $\frac{11^0 \cdot e^{-11}}{0!} = 0.0000167017 \approx (1.67 \times 10^{-3})\%$ chance of Hawaiian volcanos erupting 0 times in a randomly selected year. Once again, this is theoretical and intended to introduce Poisson Distribution as it is not in the syllabus.

Aim & Approach

In this investigation, I will be using Poisson distribution to estimate the probability of two discrete random variables the number of U-Bikes that arrive is *A*, while the number of U-Bikes that leave is *L*. The type of statistics will fall under discrete distribution as all of the data have discrete values. As mentioned in the previous section, we must address also address the conditions required to use Poisson distribution. Condition 2 may be assumed to be constant, but this will be addressed later in the *Evaluation* section. Condition 3 is met as the longer one spends recording the number of bikes that leave and arrive, the higher both of these numbers will be. Conditions 1 and 4 are different as the number of times both of these events can actually occur is limited by the number of bikes that are currently in the U-Bike Station, meaning that this investigation may not be independent. For example, let's assume that a U-Bike Station has 10 slots and all of them are filled up with U-Bikes. The discrete variable, *A*, is limited and cannot increase as you cannot override the slots that are

available in the U-Bike station. Therefore, I have decided that for the data collection (where I set out to a U-Bike station to record the number of bikes that leave and the number of bikes that arrive), even if the U-Bike station is full or empty, if someone is believed to be looking to park their U-Bike or is in need of one, I will record it down. This is because the question is surrounding the optimal number of U-Bikes and U-Bike slots at a U-Bike station and should therefore be able to extend beyond the number that is currently there. The type of sampling that will be used is Convenience Sampling. The reason for this is due to COVID-19, which prevents me from gaining a sample of U-Bike stations that covers a variety of spaces in Taipei. Furthermore, I realise that U-Bike does have an online U-Bike tracker, but as of when I recorded this data and performed my investigation, the online tracker was not accurate, which motivated me to collect the data on my own. Apart from the variables, A and L, we will also need two other discrete variables that will be optimised throughout this investigation: S = The number of available slots in the U-Bike station, which defines the maximum number of U-Bikes that can be placed in the station; B = The number of U-Bikes initially at the U-Bike station.

Data Collection and Results

Case Scenario: Jiantan Station (S = 73)

Over the course of 2 weeks, I collected 5 sets of data in 5 random days of the week, all at the same time, 14:00, and the test periods lasted for 30 minutes. This is represented in the table below:

Jiantan Station S = 73	**Day**				
	1	**2**	**3**	**4**	**5**
B (Initial Number of U-Bikes)	55	46	39	69	47
A (U-Bikes Arrived)	16	20	14	26	17

L (U-Bikes Left)	17	32	29	19	23

Table 1: Discrete variable table of U-Bikes in Jiantan Station

If we calculate the average of all 3 variables through the formula:

$$\frac{(x_1+x_2+x_3+x_4+x_5)}{5}$$

where x represents either *B, A* or *L* and the subscripts denote the day, round all of the values to the nearest whole number, we obtain the values: $\bar{B}=51, \bar{A}=19, \bar{L}=24$. From this data, we now have the two equations since $\bar{A}$ and $\bar{L}$ can be substituted in for λ : $f(x)$ for the U-Bikes arriving and $g(x)$ for the U-Bikes leaving. These are the equations that we will be using for the rest of the paper:

$$f(x)=\frac{19^x \cdot e^{-19}}{x!} \qquad\qquad g(x)=\frac{24^x \cdot e^{-24}}{x!}$$

Equation 1 and Equation 2: f(x) for the probability of U-Bikes arriving and g(x) for the probability of U-Bikes leaving

Properties of the Equations

Expected Value E(X):

Using the formula for Expected Value of a Discrete Random Variable X to find the Expected Value:

$$E(X)=\sum x P(X=x)$$

If we then substitute P(X = x) with the Poisson Distribution Formula where $\lambda=19 \text{ for } f(x), \lambda=24 \text{ for } g(x)$, we can find the Expected Value of the random variable X for both equations:

$$E(X)=\sum_{x=0}^{\infty} x \cdot \left(\frac{\lambda^x \cdot e^{-\lambda}}{x!}\right)$$

$$E(X)=\lambda\cdot e^{-\lambda}\sum_{x=1}^{\infty}\frac{\lambda^{(x-1)}}{(x-1)!}$$

$$\textit{substitute } z=x-1, E(X)=\lambda\cdot e^{-\lambda}\sum_{z=0}^{\infty}\left(\frac{\lambda^{z}}{z!}\right)$$

$$\textit{Using the Maclaurin or Taylor Series rule where } \quad e^{x}=\sum_{k=0}^{\infty}\frac{x^{k}}{k!} \quad , \quad E(X)=\lambda\cdot e^{-\lambda}\cdot e^{\lambda}$$

$$\therefore E(X)=\lambda$$

$$\therefore E(f(x))=19, E(g(x))=24$$

Through this, we confirm the expected discrete random value of both equations, enforcing the idea that the theoretical mean value is the same as the mean value of the numerical experiment performed earlier. Now if I plot both of these functions on a graph, it would look like this:

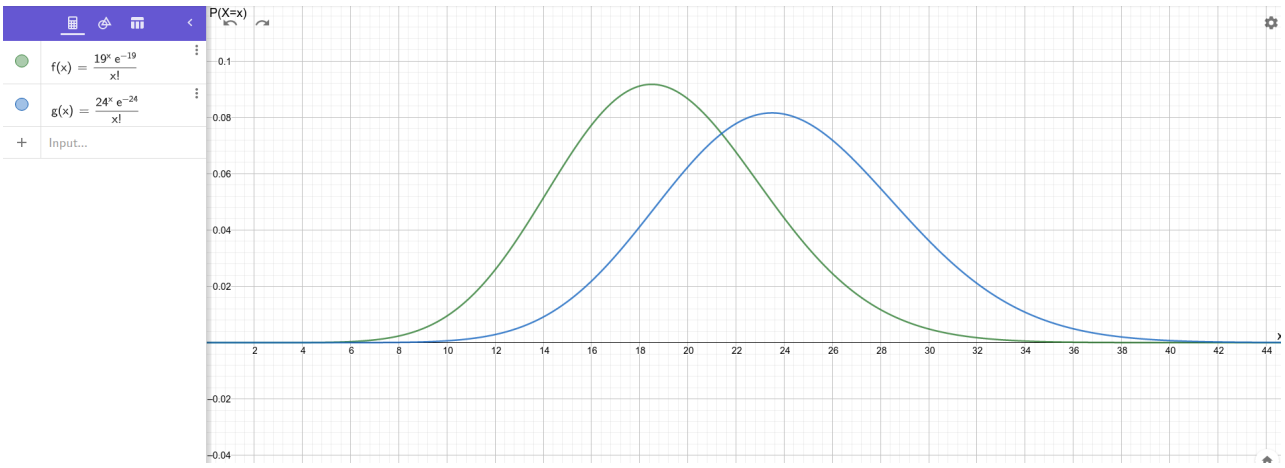

Figure 1: Functions of U-Bikes Arriving and U-Bikes Leaving plotted on a Cartesian Plane

Figure 1, as shown above gives us the probability of each number of occurrences occurring at the same time period of 1.5 hours. However, this graph utilises continuous values of x, which is not possible as the number of U-Bikes are solely discrete. Therefore we must plot it on a bar chart:

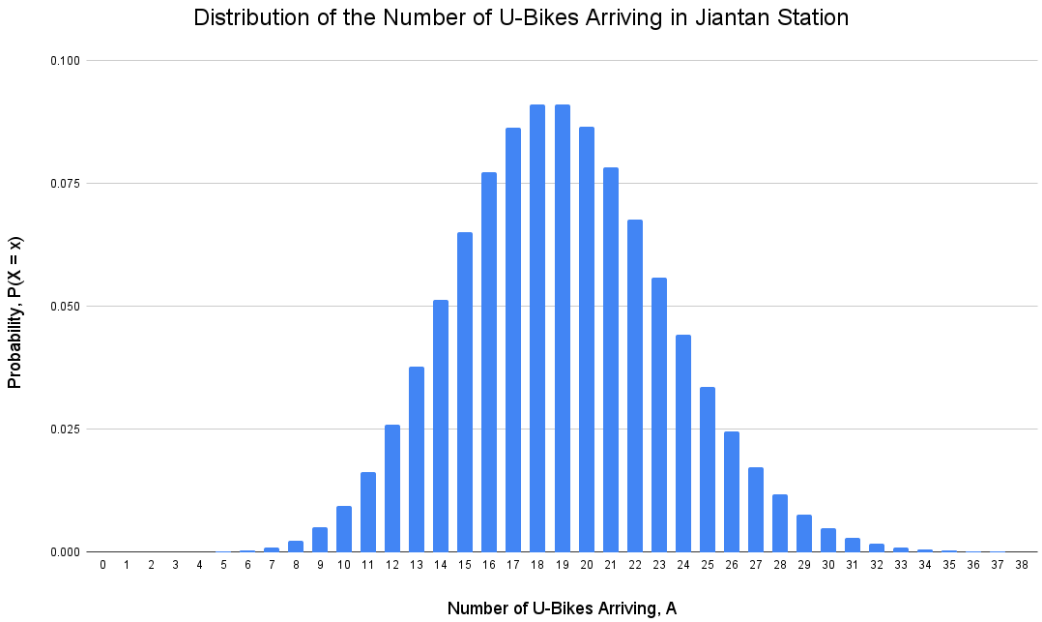

Figure 2: Bar Chart of $f(x)$

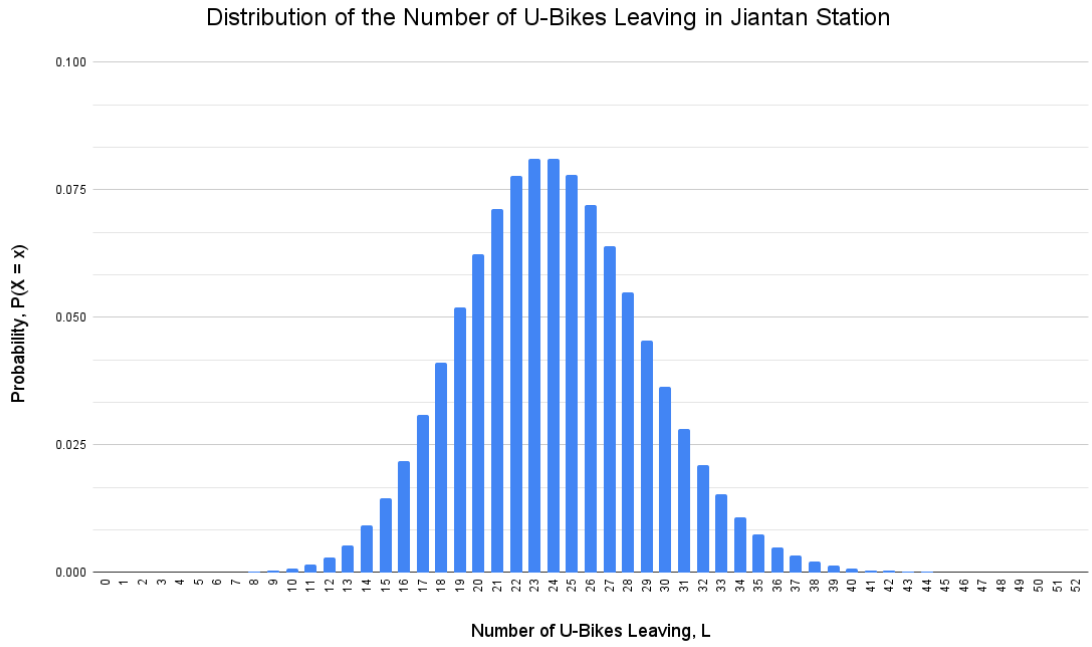

Figure 3. Bar Chart of $g(x)$

Both Figure 2 and 3 allow us to visualise the probability of any discrete number ≥ 0 of U-Bikes arriving or leaving at the time period of 1.5 hours. The total probability is 1 for both figures. As we can also visualise, the figures are slightly positively skewed, where the discrete variable along the

x-axis may increase to infinity however as the x values increases, the probability decreases significantly and is so minute, it will become insignificant (decimal places beyond 30 places). Following this, in the *Optimisation & Analysis* section, Part 1 will focus on optimising the variable, *B*, the initial number of U-Bikes, while Part 2 will focus on the variables, *S*, the number of slots available at the U-Bike station and *B*.

Optimisation & Analysis

In order to tackle the goal of optimisation, I will focus on the probability of issues arising with the U-Bikes in correlation with the variables *B* and *S*, and attempt to find the values of *B* and *S* where this probability of issues with the U-Bikes are the lowest. For the rest of this paper, I will coin the issue related with U-Bikes as "failures". These are the condition cases of failure with the U-Bike Station:

1) Number of U-Bikes arriving is greater than the free spaces available: $A - L > S - B$

2) Number of U-Bikes leaving is greater than the number of bikes available: $L - A > B$

Part 1: Optimising the Initial Number of U-Bikes (S = 73)

In order to optimise the initial number of U-Bikes within a station, I will have to consider each probability of *B* while *S* remains constant as this is Part 1. For Case 1, the value of *B* will increment from 0 to the constant value of *S* by 1, and for each incrementation, I will look for the discrete values of *A* subtracted by *L,* such that they are greater than $S - B$. For Case 2, the number of *B*, will again, increment from 0 to the constant value of S by 1, and for each incrementation, I will look for the discrete values of *L* subtracted by *A* that are greater than *B* itself.

The Process Every Incrementation

Each incrementation will yield a number of pairs of results of the numbers *A* and *L*. In order to explain this further, consider the scenario for Case 1: $B = 32$ (it is on its 33rd incrementation, as it starts from 0). A possible pair of *(A, L)* may be *(53, 4)* since *53 – 4 = 49,* which is greater than *73 – 32 = 41*. Pairs *(49, 7), (60, 19), (72, 21)* are all pairs of numbers that will fit in the condition that there are too many U-Bikes arriving at the station and there is an overflow of this number. For Case 2, it works the same way except $L - A > B$. Once I obtain a value pair of *A* and *L*, I can again substitute these values into the Poisson Distribution Formula, where $f(A)$ and $g(B)$, yielding the results which are shown in Figures 2 and 3. Then, I will use the formula: $P(A \cap B) = P(A)P(B)$ (since the events are independent as mentioned prior), multiply the two probabilities and sum it with all other pairs of the same incrementation. For each value of *B,* the sum of the probability will be denoted as the variable *P.* For each incrementation, I looked for pairs of *A* and *L* from 0 to $\bar{A}$ + *100* and $\bar{L}$ + 100. Despite the probability from the Poisson Distribution Equations reducing to an incredibly small value as *A* and *L* increases as shown in Figures 2 and 3, it is still vital to take these values into account to gain an accurate value of *P* at the end. I did not extend beyond $\bar{A}$ + *100* and $\bar{L}$ + 100 as it caused a Buffer Overflow (too many numbers for my computer to handle) in the program as the numbers were too large.

Here is the overall process every incrementation:

1. Find all pairs of *A* and *L* which fit Case 1's conditions or Case 2's conditions.
2. For every pair:
    a. Substitute the values of *A* and *L* into the Poisson Distribution Formula.
    b. Multiply the two probabilities together.

5. Sum all of the products together.

6. Record the value of B down and the sum of the probability

7. Increment to the next value of B and repeat this process.

If we were to translate this process every incrementation into a Mathematical Notation with the use of the *Iverson Bracket Notation* — a notation where if the conditions within the square brackets are met would equal a value of 1, but if the conditions are not met would equal a value of 0 — this is what we would get:

$$D(S,B)=\sum_{A=0}^{\bar{A}+100}\sum_{L=0}^{\bar{L}+100} f(A)\cdot g(L)\cdot[A-L>S-B]+f(A)\cdot g(L)\cdot[L-A>B]$$

Equation 3: where *S = 73 for all cases in Part 1 and* $\bar{A}$ *= 19,* $\bar{L}$ *= 24 for the Jiantan Station*

From this equation, we can look can look at an example: *S = 73* and imagine that *B* = 14 and while it is a summation of 0 $\bar{A}$ *+ 100* and $\bar{L}$ + 100, we can focus on a certain part of the summation to see how *Iverson Bracket Notation works: (A = 50, L = 29)*

$$f(50)\cdot g(29)\cdot[50-29>73-14]+f(50)\cdot g(29)\cdot[29-50>14]$$

Since 50-29 is not > 73-14 (21<59), this bracket would then become 0, thus reducing the former appends to 0. Moreover, 29-50 is not > 14 (-21<14) and reduces the latter append to 0 as well. Therefore, for this summation where *A = 50* and *L = 29* for *S = 73* and *B = 14,* it does not meet the conditions for Failure and thus does not add more probability to the summation.

I did not manually implement this method and instead used Python to program this process (Program is in the *Appendix* Section) dubbed with the program function name, “part_1(73, 19, 24”) . From the program, I put all of the numbers inside a table as shown below:

B: Initial Number of U-Bikes	Probability, P(X = x), of Failure
0	0.7539667657594555
1	0.7031688702589577
2	0.6480701232420623
3	0.5897042286116327
4	0.5293214789003586
5	0.46830829614850694
6	0.4080914498689646
7	0.350038012586479
8	0.29536273628481524
9	0.24505324763395542
10	0.1998205511401555
11	0.16007840573580612
12	0.12595097855668233
13	0.09730453836958229
14	0.07379640521870479
15	0.054933210461961535
16	0.04013072234249242
17	0.028768782118791147
18	0.02023684890487883
19	0.013967808509172891
20	0.009459670108357987
21	0.006286291083060956
22	0.00409921939420207
23	0.0026231394230319693
24	0.001647356202205911
25	0.0010154024122986055
26	0.000614352277978177
27	0.00036490030612134125
28	0.00021279504385442706
29	0.00012185293966048215
30	6.852634656748341e-05
31	3.785199411584485e-05
32	2.053971637743747e-05
33	1.0950690780407824e-05
34	5.7371858326533895e-06
35	2.954193778127836e-06
36	1.4953594045700612e-06
37	7.443147236064418e-07

38	3.6465283214483313e-07
39	1.7655340564027514e-07
40	8.612703889358078e-08
41	4.611094676025428e-08
42	3.499391990867967e-08
43	4.723270927935829e-08
44	9.101163390449258e-08
45	1.9288715117525592e-07
46	4.114521015666465e-07
47	8.656693120940732e-07
48	1.788713711880917e-06
49	3.626216038960927e-06
50	7.21020343310495e-06
51	1.4058513283991728e-05
52	2.687576049977586e-05
53	5.036729876628269e-05
54	9.252184489888639e-05
55	0.00016656857625336503
56	0.00029386315646444275
57	0.0005079907356395309
58	0.0008603717237473592
59	0.001427590550644129
60	0.0023205077147727877
61	0.0036949336170606587
62	0.0057632256522649585
63	0.008805632287916445
64	0.013179605016183614
65	0.019324735886620956
66	0.027760605534639327
67	0.03907482122277851
68	0.05389905182116878
69	0.07287202866883592
70	0.09659026122671506
71	0.125549438835954
72	0.16008181326841817
73	0.2002968138729791

Table 2: Part 1 - Probability of each value of B occurring.

If we were to graph this on a bar chart, this is what we would get:

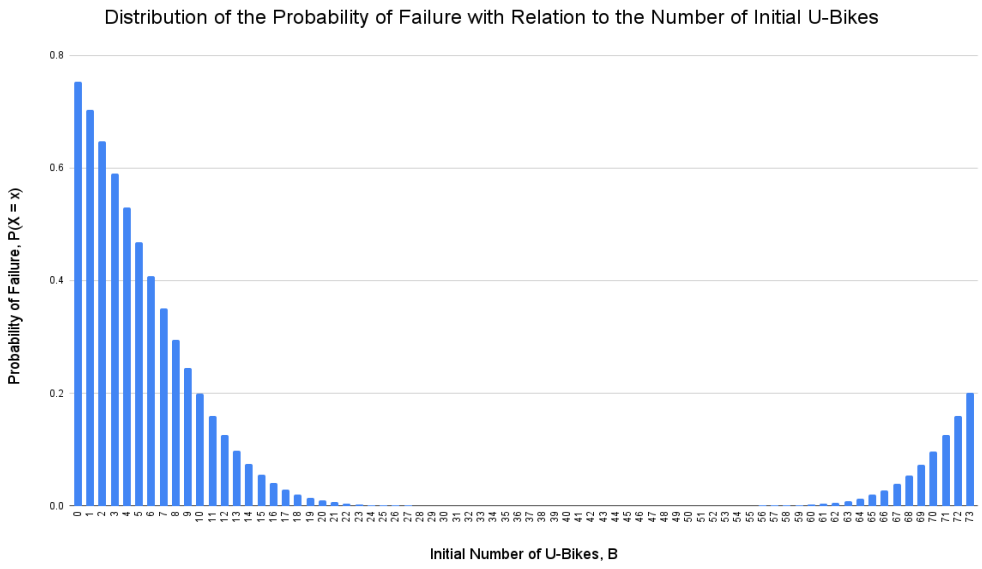

Figure 4. Probability of Failure at Jiantan Station

From here, I will find the value of B that yields the lowest probability. The reason for this is to calculate the probability of running into Issue Cases 1 and 2 and to minimise this probability to find the optimal number of U-Bikes to initially place at a U-Bike station. If I were to only address one of the condition cases, the lowest probability would always be $B = 0$ for Case 1 and $B = S$ for Case 2 since they are the ones that are least likely to yield errors for only one case, but the other case has to be taken into account as the U-Bikes are constantly arriving and leaving, not simply one of them.

Figure 4 represents the distribution of the probability of U-Bike failure in relation to the variable, B. As we can observe, having $B = 0$ will yield us almost 80% chance of failure, while having $20 \leq B \leq 63$, will return a less than 1% chance of failure. However, if we analyse the program and Figure 4 and Table 2 more carefully, we can gauge the optimal value of B, where the value of P is the lowest:

$$B = 42.$$

There is a difference of 9 U-Bikes from the data collected at the start, $\bar{B}=51$, denoting that the number of U-Bikes initially present at Jiantan Station was too high by 9 U-Bikes. When the value of *B* is 42, it gives us a less than $(3.50\times10^{-6})\%$ chance of failure while *B = 51* gives us less than $(1.41\times10^{-3})\%$ chance of failure: An improvement of more than 2 magnitudes in reducing the chance of failure.

Part 2: Optimising the Number of Slots & the Initial Number of U-Bikes

The second part of the optimisation is where both *S* and *B* are optimised to values that will yield the lowest chance of failure. It is similar to Part 1 as for every value of *S*, a range of values of *B* (from values 0 to *S*) will be procured using the method from Part 1 and again, the value of a certain *S* and *B* with the lowest probability is the optimal solution. However the constraints of S will always be greater than 0, instead of greater or equal to 0 as equal to 0 would mean there is no U-Bike Station.

It is salient to consider the lowest value of S, but, in the case that we did not have the goal of lowering the value of *S,* the program will pick the highest possible value of *S* every single time. The reason for this is because the highest possible value of *S* with the optimal value of *B* calculated from Part 1, will have the highest likelihood of avoiding Failure Condition Case 1 and since the value of *B* can change, Failure Condition Case 2 can always be avoided as well. If we consider this logically, this idea is clear and while it fits into the theoretical idea if we only consider avoiding the failures, we also need to factor in the practicality of implementing this. In order to illustrate this idea, we can focus solely on *S* first to show that the program will pick the highest value of *S*. We can use the formula as shown below because within each value of *S* there will be probabilities ranging from $0<B\leqslant S$ so we will need to average the value:

$$U(S)=\frac{\sum_{B=0}^{S} D(S,B)}{S}$$

Equation 4: U(S), a function to calculate average failure.

The function *U(S)* will not produce probability values but will again follow the idea that the lower the value is, the better. The example model only considers values of $1<S\leqslant 150$ and if we run the program function "part_2_no_normalise (19, 24)" and graph this on a bar chart, this is what we would get:

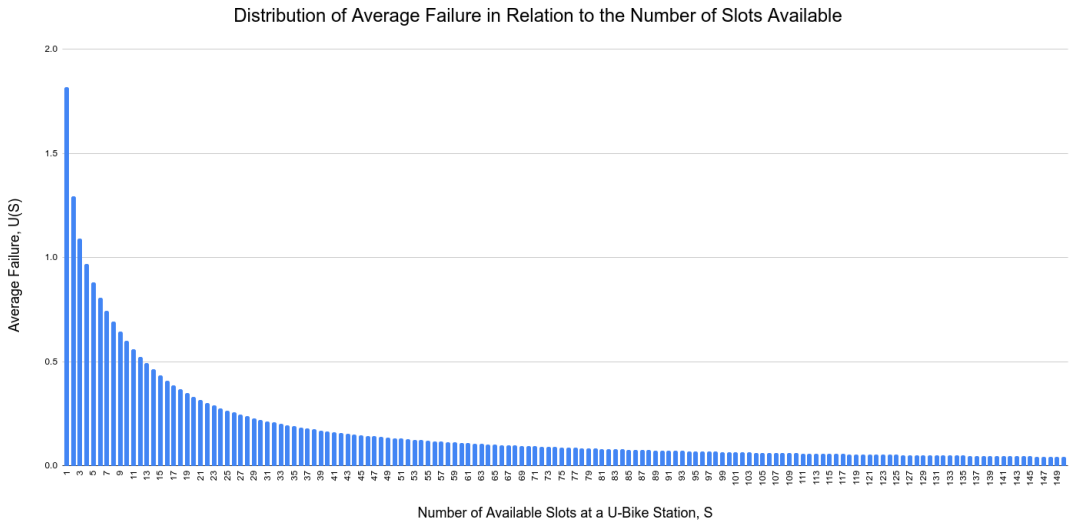

Figure 5: Average Failure in Relation to the Number of Slots Available $1<S\leqslant 150$ (Lower U(S) value is better)

Figure 5 does NOT use probability values and the figure is used merely to convey my point of a bias towards the highest value of *S*. The aim of of minimising *S* is vital as stated in the *Introduction & Rationale* section which is to reduce costs and improve the effective use of resources. In order to avoid this bias towards the highest possible value of *S*, we can consider adding a constant value (constant only for case of U-Bike station) to normalise the value of *S* which is then added to the variable, *P*, to discourage the bias towards higher values of *S*. From this the question is now: What is the constant value that should be used to normalise the value, *S?* First, let's coin this constant term as *C*. *C* has to have an inversely proportional relationship with *S* to discourage the bias for

higher values of *S*. *C* can therefore represent the budget that is given by the government to create a U-Bike station. This makes logical sense as the more budget that is given by the government, the less important the number of slots is in determining the lowest value of *P* allowing us to therefore afford to implement more U-Bike slots. I would also like to clarify that *C* does not represent the actual cost of building a U-Bike station, it is merely a theoretical study that has more of a practical aspect to it. From this idea, we can now create an example: The government has granted 300,000 for the building of U-Bike Stations in Jiantan Station and we consider the constraints:

$$C=300000 \quad , \quad 1<S\leqslant 200 \quad , \quad 0<B\leqslant S \quad , \quad 0<A\leqslant 119 \quad , \quad 0<L\leqslant 124$$

The program function, "part_2 (300000, 19, 24)", produces a different layout but if we organised the information, it would look like this:

S: Number of Available Slots	**B: Initial Number of U-Bikes**								
	0	**1**	**2**	**3**	**...**	**197**	**198**	**199**	**200**
1	0.9140519	0.9034690	--	--	--	--	--	--	--
2	0.8795228	0.8632573	0.8483736	--	--	--	--	--	--
3	0.8505670	0.8287283	0.8081619	0.7900110	--	--	--	--	--
4	0.8268521	0.7997724	0.7736328	0.7497993	--	--	--	--	--
5	0.8078824	0.7760575	0.7446770	0.7152703	--	--	--	--	--
6	0.7930615	0.7570879	0.7209621	0.6863144	--	--	--	--	--
7	0.7817507	0.7422670	0.7422670	0.6625995	--	--	--	--	--
...	--	--	--	--	--	--	--	--	--
194	0.7546134	0.7038155	0.6487167	0.5903508	--	--	--	--	--
195	0.7546167	0.7038188	0.6487201	0.5903542	--	--	--	--	--
196	0.7546200	0.7038222	0.6487234	0.5903575	--	--	--	--	--
197	0.7546234	0.7038255	0.6487267	0.5903608	--	0.2009534	0.2466899	0.2974877	0.3525865
198	0.7546267	0.7038288	0.6487301	0.5903642	--	0.1607418	0.2009568	0.2466932	0.2974911
199	0.7546300	0.7038322	0.6487334	0.5903675	--	0.1262127	0.1607451	0.2009601	0.2466965
200	0.7546334	0.7038355	0.6487367	0.5903708	--	0.0972569	0.1262161	0.1607484	0.2009634

Table 3: Sample dataset from a budget of 300,000 where the probabilities are the result of a pair of *S and B.* -- indicates that the value is invalid.

I have reduced the size in Table 3 as a 200 by 200 Table is unable to fit on the document. The table differs slightly from Table 2 as it is an index table where matchings of *S* and *B* produce a value, NOT probability of failure. If I use Python to look through the entirety of the results, the values of *S* and *B* that yield the lowest value of failure is 58, 34, respectively, with a value of around 0.0002027. This value is not the probability as it includes the normalisation factor; in order to calculate the actual probability for failure we will need to subtract the normalisation factor from the value:

$$0.0002027-\frac{58}{300000}=0.0000093667\approx(9.37\times10^{-4})\%$$

The probability for failure is higher than *S = 73 and B = 42* with a probability of around $(3.5\times10^{-6})\%$, but is still preferred over the original collected dataset with *S = 73, B = 51* with a probability of around $(1.41\times10^{-3})\%$. Referring back to the case with *S = 73 and B = 42*, the probability is evidently lower with the higher *S* value (because of a higher value of C) and the optimised *B* value. If the value of C was higher than 300,000, we would be able to obtain more slots a lower probability of failure So then how does the budget affect the optimal value of values? In order to investigate this, I found a variety of the optimal values of *S* for contrasting values of *C* and graphed *S* against *C* as *C* is the independent variable while *S* is the dependent variable here:

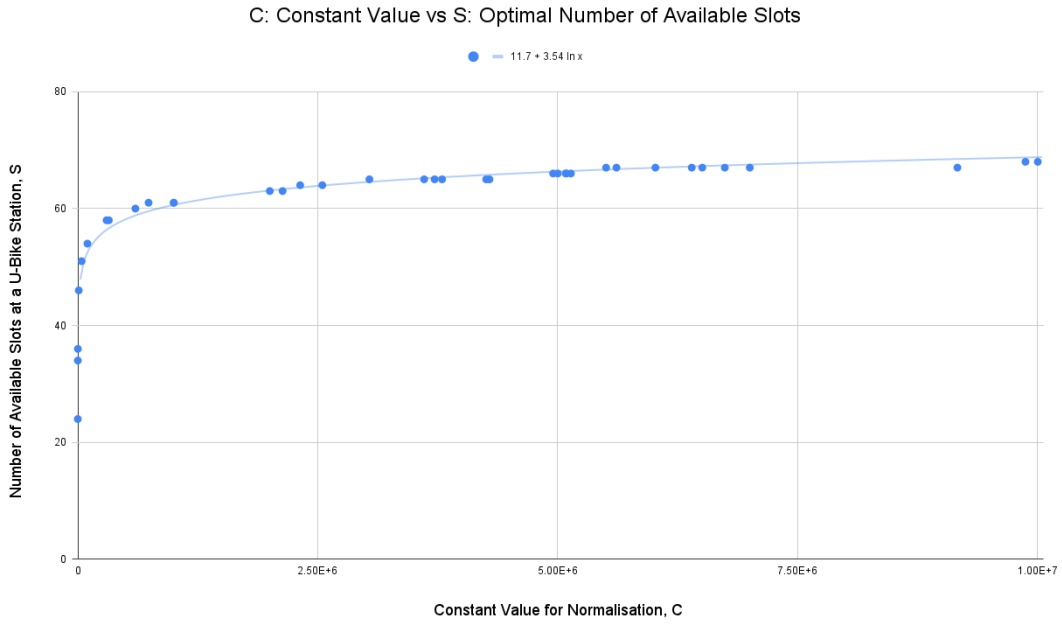

Figure 6: Effect of the *C* on the optimal value of *S*

As we can visualise, the amount of budget has a logarithmic relationship with the optimal value of *S* where the function is:

$$V(C) = 3.54 \cdot lnC + 11.7$$

Equation 5: V(C), a function to calculate the budget vs the cost of the U-Bike station

This equation indicates that there is a huge difference in the value of *C* as *S* increases above 60. If we substitute the original value of S = 73 in Jiantan Station:

$$3.54 \cdot lnC + 11.7 = 73$$

$$C = e^{\frac{(73-11.7)}{3.54}} = 33144395.01 \approx 33{,}000{,}000$$

The number obtained is around 33 million which is incredibly high however this idea of a cost factor is purely theoretical. Although in practical senses this may sound absurd as the price should increase with a linear relationship (if other factors such as purchasing in bulk are not taken into account), however it may be the case that building U-Bike stations above the size of 60 requires

significantly more land, space and maintenance, therefore discouraging the government to build such sizes $S > 60$ for Jiantan Station. But again, this is hypothesised and does not represent the realistic situation, leaving this for further evaluation in the *Evaluation* section.

Conclusion

The original aim of the investigation is to predict the optimal values of number of slots available at a U-Bike Station and the initial number of U-Bikes at the Station. Thus, through Part 1 and 2, we have successfully done this using the average number of U-Bikes arriving and leaving and a supposed budget/cost factor - all in order to avoid failure, a term coined by me throughout this paper, referring to a surplus or shortage of U-Bikes. As we have discovered for Jiantan Station with the initial number of slots being 73, the optimal number of initial bikes is 42, however by factoring in a budget/cost idea, the number of slots can fluctuate depending on the amount of money given. Although the higher the budget, the greater the number of slots used to avoid failure, a suggestion could be made to the government regarding efficient use of resources if the government accepts the increase in chance of failure. With many of the optimised values of number of slots and initial number of U-Bikes yielding probabilities below $10^{-3}\%$, resources can be intelligently allocated and redirected while still maintaining a low chance of failure. In order to apply these functions and calculations as we have done so for Jiantan Station to other U-Bike stops, we would only need 3 variables: $C, \bar{A}, \bar{L}$, but if we were to also optimise the number of initial U-Bikes we would also need S.

Evaluation

In the *Aim & Approaches* section, my solution towards *Conditions 1 & 4* is flawed as it is impossible to tell with 100% accuracy that someone wants to use or return a U-Bike, even if it is

full or empty. However, I would like to clarify that in the recording of my data, I did not meet this encounter, meaning that *Conditions 1 & 4* are still met throughout my investigation. *Condition 2,* although assumed to be true throughout the investigation, highlights a problem as the occurrence rate may change depending on the time, however this may be addressed by extending the duration of the collection of data per day to cover a wider range of times. The data would have been largely more accurate and precise if the data were collected for longer allowing the variables to be derived from the lengthened period.

In addition to this, the variable *C,* was completely theoretical and simply linked to this idea of averting from the larger values of *S* as this was not practical. In order to extend this idea, an extension could potentially look into actual cost of a U-Bike station and investigate all the components of it, which may result in a more accurate relationship between *S* and *C*. Another extension could be to investigate how many U-Bikes the number of slots necessary and initial number of U-Bikes in a completely new area with no previous U-Bike Station. Through this idea, the focus may be on the number of people in the area, the medium of transport people take, and the general likelihood that people are going to take the U-Bike.

Bibliography

- Poisson Distribution | Brilliant Math & Science Wiki. (2016). Brilliant. https://brilliant.org/wiki/poisson-distribution/
- Wikipedia contributors. (2021, May 19). *Poisson distribution*. Wikipedia. https://en.wikipedia.org/wiki/Poisson_distribution

- jbstatistics. (2012, May 26). *The Poisson Distribution: Introduction (fast version)*. YouTube. https://www.youtube.com/watch?v=8x3pnyYCBto
- Wikipedia contributors. (2021a, February 26). *Iverson bracket*. Wikipedia. https://en.wikipedia.org/wiki/Iverson_bracket
- Wikipedia contributors. (2021b, May 1). *Taylor series*. Wikipedia. https://en.wikipedia.org/wiki/Taylor_series

Appendix

Note that all loops within the programs below start from (Start to Finish+1) since in Python loops, Finish +1 will loop only to Finish. The bold text defines the functions that were used as mentioned above throughout my investigation.

```
from tabulate import tabulate
import math
def poisson_formula(mean_number, whole_number):
    return ((pow(mean_number, whole_number)*math.exp(-mean_number))/math.factorial(whole_number))
def part_1(S, A_AVG, L_AVG):
    table = [["Probability", "B: Initial Number of U-Bikes"]]
    for B in range(0, S+1):
        probability = 0
        for A in range(0, A_AVG+101):
            for L in range(0, L_AVG+101):
                if ((A-L) > S - B):
                    probability += poisson_formula(A_AVG, A) * poisson_formula(L_AVG, L)
                if ((L-A) > B):
```

```
                probability += poisson_formula(A_AVG, A) * poisson_formula(L_AVG, L)
        if probability != 0:
            table.append([probability, B])
    print (tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
    del table[0]
    print ("Optimal Value of B is {} with a probability of {}".format(min(table)[1], min(table)[0]))
def part_2_no_normalise(A_AVG, L_AVG):
    table = [["Probability", "S: Number of Available Slots"]]
    for S in range(1, 151):
        avg_probability = 0
        for B in range(0, S+1):
            probability = 0
            for A in range(0, A_AVG+101):
                for L in range(0, L_AVG+101):
                    if ((A-L) > S - B):
                        probability += poisson_formula(A_AVG, A) * poisson_formula(L_AVG, L)
                    if ((L-A) > B):
                        probability += poisson_formula(A_AVG, A) * poisson_formula(L_AVG, L)
            if probability != 0:
                avg_probability += probability
        avg_probability = avg_probability / (S)
        table.append([avg_probability, S])
```

```
        print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
    print (tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
    del table[0]
    print ("Optimal Value of S is {} with a probability of {}".format(min(table)[1], min(table)[0]))
def part_2(C, A_AVG, L_AVG):
    table = [["Probability", "S: Number of Available Slots", "B: Initial Number of U-Bikes"]]
    for S in range(1, 201):
        for B in range(0, S+1):
            probability = 0
            for A in range(0, A_AVG+101):
                for L in range(0, L_AVG+101):
                    if ((A-L) > S - B):
                        probability += poisson_formula(A_AVG, A) * poisson_formula(L_AVG, L)
                    if ((L-A) > B):
                        probability += poisson_formula(A_AVG, A) * poisson_formula(L_AVG, L)
            if probability != 0:
                probability += (S/C)
                table.append([probability, S, B])
        print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
    print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
    del table[0]
    print ("Optimal Pair is S = {} and B = {}, with a probability of {}".format(min(table)[1], min(table)[2], min(table)
[0]))
```