Simulation and Optimization

Seth Berry

2021-09-06

Contents

Chapter 1

Preface

You will find some form of this course in undergraduate and graduate business programs across the country. Instead of telling you what you should be looking for in such a class, it is easier to tell you what you shouldn't see:

1. Excel
2. Excel Add-ins
3. Palasaides

While the world runs on Excel, doing Simulation and Optimization in Excel will only ensure that you know Excel (can you use a VLookUp and sumproduct). Instead, the focus will be on understanding and implementing. You *will* be able to break problems down into the smallest parts and then roll those parts into objects. Throughout our time, we will get our hands dirty with some theory. These dives into theory will only serve to ease into greater understanding. In the end, however, the goal is application.

These techniques also serve as a nice introduction to programming in general, as they will allow us to scale from simple objects to very complex pipelines. Throughout our time here, we will mostly focus on using R. However, we don't live in a monolingual world anymore. To that end, we will see how some of these techniques translate to other languages (namely Python and Julia) and why we might want to consider one program over the other (speed, feature complete, ease, etc.).

Chapter 2

Introduction

In a world of exciting methods, simulation and optimization sit alone. Nobody touts how these things will change humanity. Nobody discusses how these methods can solve all of the world's problems. No... those conversations are reserved for things like statistical methods, machine learning, and the almighty AI! The secret, though, is that all of these fancy techniques do not exist without simulation and optimization.

Optimization is found in nearly every science: from nuclear medicine and biology, to electrical engineering and statistics, and beyond. While these fields are interesting on their own, our goal is to explore optimization in the context of business problem solving, with an occasional dive into how these techniques are used in techniques you learn in other courses.

Simulation is just as fundamental to the sciences as optimization. Nearly every event that happens is bound by some type of distribution and knowing that distribution allows us to test that event. What makes simulation so much fun is that you can program a version of the real world, run that program a few thousand times, and then generate a distribution of potential outcomes. This distribution will show us how common an event might be.

2.1 The Story

Meet Ali. Ali graduated from an Business Analytics program on the Atlantic coast in 2019 and made a smart career choice – accepting an offer to work as an analyst in the cannabis industry. The global cannabis industry has seen explosive growth during the last several years (topping 9 billion in 2020 and has a project compound annual growth rate of ~26% in America alone). While some of Ali's classmates (and family) questioned the decision, it was clear that it was an industry in need of some real analytics and Ali saw a real path towards making a difference for a business (after all, most people can't make a real difference in FAANGM). The Canadian cannabis industry has nearly a decade of maturity over the American cannabis industry, and American companies are looking to cover some of that lost ground. To that end, American companies are hiring people from all of the world to create the strongest possible teams. With diverse backgrounds and experiences, the general hope is that teams will function at the highest possible levels.

Ali belonged to a team with 3 other analysts: Alex, Jun, and Shashi. Ali was the youngest and least experienced of the entire group. What Ali lacked in experience, was more than made up by technical prowess. What Ali didn't know is that the analytics world has a dark secret. Throughout Ali's education, Python and R were touted as the most important languages in the world – they are, after all, where all of the exciting work happens. What Ali found, though, was that business analytics really runs on Excel and various add-ins.

Ali had a goal: to become the most valuable member of the team. Ali decided to take on anything the organization needed. It seemed like a good idea at first, but Ali found out that the Business Analytics program didn't really offer the proper preparation for what was to come.

Chapter 3

Linear Optimization

Ali's first task was to determine a marketing strategy. Both the Canadian and American cannabis industries are trying to normalize cannabis use (mainly through edibles and drinks) to women between the ages of 30 and 55. The working theory is that making cannabis use acceptable to this group will "allow" married men to also enjoy recreational cannabis use.

Ali's manager, Tolu, has asked to create a semi-automated system for determining advertisement spends. Thankfully, Tolu noted that Ali's coworker, Jun, has already been working in this space. Ali should be able to jump on Jun's work and make this system automated without much hassle.

3.1 Continuous Optimization

3.1.1 The Problem

What should have been an easy task became a nightmare. Ali didn't get a csv file with neatly defined columns and a clear outcome variable. No... Ali received this email (in which Jun was copied):

> Hi Ali,
>
> Here is what Rayan from Marketing needs:
>
> Instagram ads cost $50 dollars per hundred clicks
>
> TikTok ads cost $20 dollars per hundred clicks
>
> Over the last few weeks, we averaged about 1 female view for Instagram and 4 for TikTok. We need at least 80 female views in total for the coming week.
>
> We don't really do as well with men; we saw just about 1 average male view for both Instagram (.9) and TikTok (.8). We are really hoping to get at least 40 for the coming week.
>
> Where should we buy ads for the coming week?
>
> All my best,
>
> Tolu

3.1.2 From Words To Formulas

In typical analyst fashion, Ali responded with, "No problem!", and started digging through old course notes. Unfortunately, nothing looked like this problem. Ali decided that a cup of coffee with Jun was the way to go. Before a coffee invite even went out, Jun sent Ali a copy of the legacy Excel sheet... completely full of Excel equations and Solver boxes.

Ali had no idea what was going on in the sheet – there were *sumproduct* formulas, *vlookups*, and other strange things. Ali asked Jun to go over the sheet and the problem together, and Jun most graciously agreed.

Jun helped Ali break the problem down into small pieces. “The first question”, Jun said, “is what are the variables and what are their values?”

Ali thought for a minute and decided that there are two variables to this problem: Instagram and TikTok. “Correct!”, said Jun, “and what values do Instagram and TikTok have?” Ali went back to the email and saw:

> Instagram ads cost \$50 dollars per hundred clicks
>
> TikTok ads cost \$20 dollars per hundred clicks

“Awesome! The value for Instagram is 50 and the value for TikTok is 20.”, Ali said. “And what specifically are those?”, Jun asked. Ali wasn’t sure, but said, “ad costs”. “Now, let me show you something.”, and Jun wrote this on a piece of paper:

$$\text{ad cost} = 50_{instagram} + 20_{tiktok}$$

“What else do we need?”, asked Jun. Ali thought for a minute and said, “We need to get the rest of the information into the problem!”

> Over the last few weeks, we averaged about 1 female view for Instagram and 4 for TikTok.
>
> We don’t really do as well with men; we saw just about 1 average male view for both Instagram (.9) and TikTok (.8)

“Let’s put that into our problem”, and Jun was back to writing:

$$\begin{aligned} \text{ad cost} = 50_{instagram} + 20_{tiktok} \\ 1_{instagram} + 4_{tiktok} \\ .9_{instagram} + .8_{tiktok} \end{aligned}$$

“Did Rayan have an specific needs for those men and women?”, asked Jun. Again, Ali looked at the email and saw:

> We need at least 80 female views in total for the coming week.
>
> We are really hoping to get at least 40 for the coming week.

“So,”, Ali began, “We need at least 80 views for women and 40 views for men. We could have more for both, though. . . it is just the baseline.”

“Excellent! Check this out”, and Jun added the following:

$$\begin{aligned} \text{ad cost} = 50_{x1} + 20_{x2} \\ \text{women} = 1_{instagram} + 4_{tiktok} \geq 80 \\ \text{men} = .9_{instagram} + .8_{tiktok} \geq 40 \end{aligned}$$

“And we are almost there!”, Jun smiled and then asked, “What would we want to do with cost: spend as much as possible or as little as possible?” “Oh”, Ali said, “that’s easy: we definitely want to minimize our cost.”

$$\begin{aligned} \text{Minimize:} \\ \text{ad cost} = 50_{instagram} + 20_{tiktok} \\ \text{Subject to:} \\ \text{women} = 1_{instagram} + 4_{tiktok} \geq 80 \\ \text{men} = .9_{instagram} + .8_{tiktok} \geq 40 \end{aligned}$$

“Here’s the last question”, Jun said, “Could we buy a negative number of ads?”. “Absolutely not”, Ali said.

“We have this now”, and Jun showed Ali his paper

$$\begin{aligned}\text{Minimize:}\\ \text{ad cost} = 50_{instagram} + 20_{tiktok}\\ \text{Subject to:}\\ \text{women} = 1_{instagram} + 4_{tiktok} \geq 80\\ \text{men} = .9_{instagram} + .8_{tiktok} \geq 40\\ \text{instagram, tiktok} \geq 0\end{aligned}$$

“Now that we have this put completely together, we need to break it down”, Jun laughed.

“We know that this is a **minimization** problem”, Jun said and continued, “and we know that we have two **variables**: Instagram and TikTok. You may hear people call these **objective values**.”

Jun carried on, “We also know that we have some rules to follow for our problem. These rules are called **constraints**. Think of these constraints as rules that reflect reality”.

“Got it!”, exclaimed Ali.

“Let’s see them again”, Jun said:

$$\begin{aligned}\text{Subject to:}\\ \text{women} = 1_{instagram} + 4_{tiktok} \geq 80\\ \text{men} = .9_{instagram} + .8_{tiktok} \geq 40\end{aligned}$$

“This whole thing is the **constraint matrix** and we can break it down into its component parts!”, beamed Jun.

"Let’s start with the **left-hand side** of the constraint matrix, which you might hear referred to as the **A matrix**.

$$\begin{aligned}1_{instagram} + 4_{tiktok}\\ .9_{instagram} + .8_{tiktok}\end{aligned}$$

“We have 4 values, spread across 2 columns and 2 rows.”, Jun said, “Just like a normal table”. Jun continued, “Next we come to the **directions**. . . those things look like inequalities, but we will also probably encounter some equalities too”.

“Here, we just have a simple **vector** of those signs.”, Jun wrote:

$$\begin{gathered}\geq\\ \geq\end{gathered}$$

“Last thing, I promise.”, Jun said: “The **right hand side**, those values that we need to achieve, are referred to as the **marginal values**.”

$$\begin{gathered}80\\ 40\end{gathered}$$

“Tolu said that you were going to program these in Q or Boa, or something like that. I can’t help you there, but let me know if I can do anything else for you”, Jun said and walked back to the office.

Ali felt better, but getting all of that information into R was going to be a little bit tricky.

3.1.3 Application

Ali was feeling pretty good after all of this! As soon as the computer was unlocked, StackOverlow came to the rescue – a user called Not_Prof_Berry had answered a few questions about linear programming with R.

It seemed like Ali was going to need a package called `linprog`:

```
# install.packages('linprog')

library(linprog)
```

The specific function is `solveLP`, but Ali saw that it needed some objects to be created first: `cvec`, `bvec`, `Amat`, and `const.dir`. Ali remembered a common mantra among professors – "Read the flipping manual!". After reading the helpfile, Ali determined that the cvec object needed to contain the objective values:

```
objective_values <- c(50, 20)
```

Ali then figured out that bvec came from the right-hand side of the constraint matrix (the values out in the margin of the constraint matrix):

```
constraint_values <- c(80, 40)
```

The Amatrix felt a little bit tricky. It definitely needed to be the constraint matrix, but it was somewhat tough to get into the right shape. Ali tried a few things:

```
constraint_matrix <- rbind(c(1, 4),
                           c(.9, .8))

constraint_matrix2 <- matrix(c(1, 4, .9, .8),
                             ncol = 2, nrow = 2,
                             byrow = TRUE)
```

Both returned a matrix:

```
str(constraint_matrix)
```

```
 num [1:2, 1:2] 1 0.9 4 0.8
```

```
str(constraint_matrix2)
```

```
 num [1:2, 1:2] 1 0.9 4 0.8
```

Which Ali knew was needed for function to work properly.

Finally, Ali made a character vector of const.dir (i.e., the constraint directions):

```
constraint_directions <- c(">=", ">=")
```

With those 4 objects, Ali was ready to solve the problem!

```
solved_model <- solveLP(cvec = objective_values,
                        bvec = constraint_values,
                        Amat = constraint_matrix,
                        maximum = FALSE,
                        const.dir = constraint_directions)
```

With the model solved, Ali needed to grab some information: the recommended values for Instagram and TikTok, and how much money it was going to cost.

First, how much money was this going to cost:

```
solved_model$opt
```

```
[1] 1000
```

Got it. $1000 is the total spend.

Second, what is the marketing mix:

```
solved_model$solution
```

```
 1  2
 0 50
```

That gives 0 ads for Instagram and 50 ads for TikTok! There is no way that is correct. Everyone knows that a 0 for an answer means that there has to be a problem. How could Ali take this solution to Tolu? Ali figured the best course of action would be to check the solution with Jun.

3.1.4 Theory

Like all early-career analysts, Ali was feeling beaten – going back to Jun so quickly felt like a failure. Like all experienced analysts, Jun was only too happy to help and explain what was happening.

Jun reminded Ali of the complete notation they had created together.

$$\begin{aligned}
\text{Minimize:} \\
\text{ad cost} = 50_{instagram} + 20_{tiktok} \\
\text{Subject to:} \\
\text{women} = 1_{instagram} + 4_{tiktok} \geq 80 \\
\text{men} = .9_{instagram} + .8_{tiktok} \geq 40 \\
\text{instagram}, \text{tiktok} \geq 0
\end{aligned}$$

“Let’s break this down a little bit”, and turning to the whiteboard, Jun wrote:

$$1_{instagram} + 4_{tiktok} = 80$$

Can be solved with:

$$(instagram = 0, tiktok = 20) \text{ or } (instagram = 80, tiktok = 0)$$

And:

$$.9_{instagram} + .8_{tiktok} = 40$$

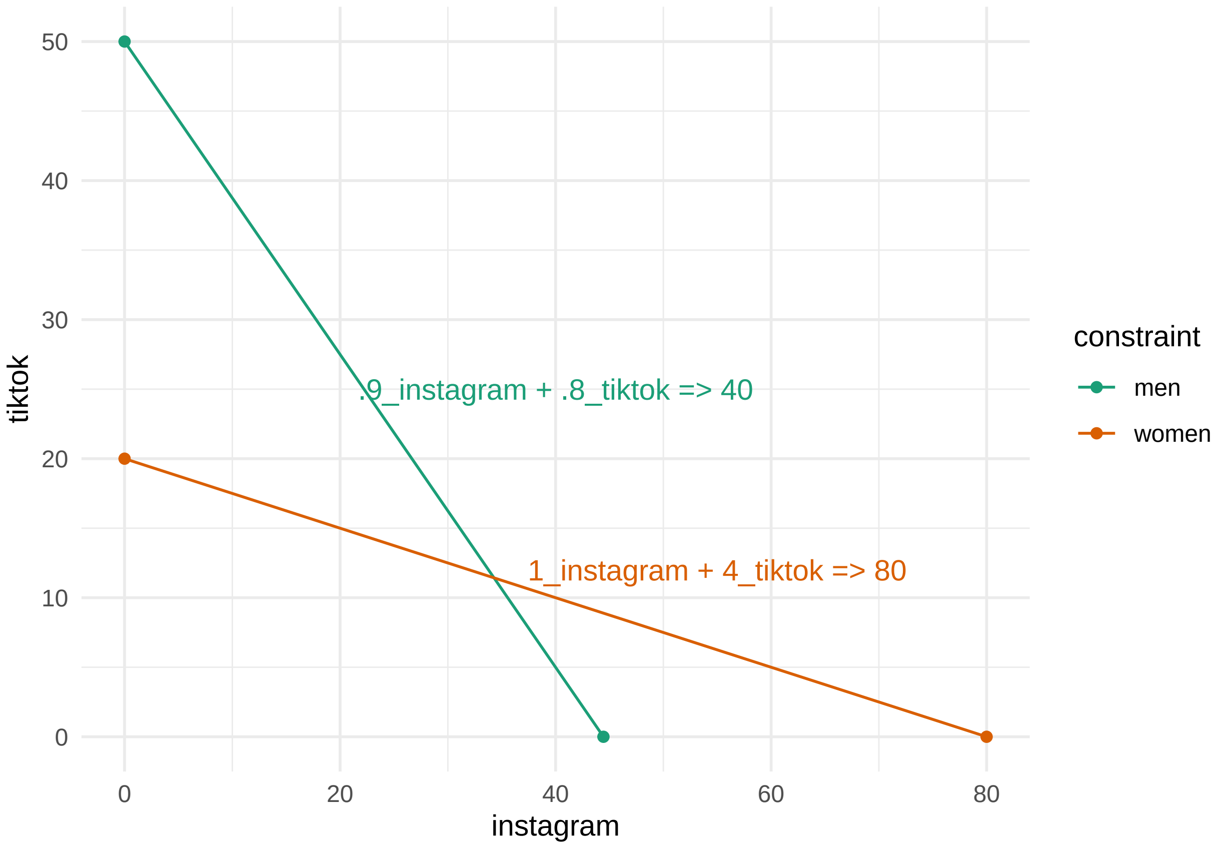
Can be solved with:

$$(instagram = 44.44, tiktok = 0) \text{ or } (instagram = 0, tiktok = 50)$$

“It’s okay if you don’t remember or didn’t take linear algebra”, Jun noted, “just know that we are solving these equations to obtain a set of points.”

“Okay”, Ali nodded, “but what do we do with those points?”

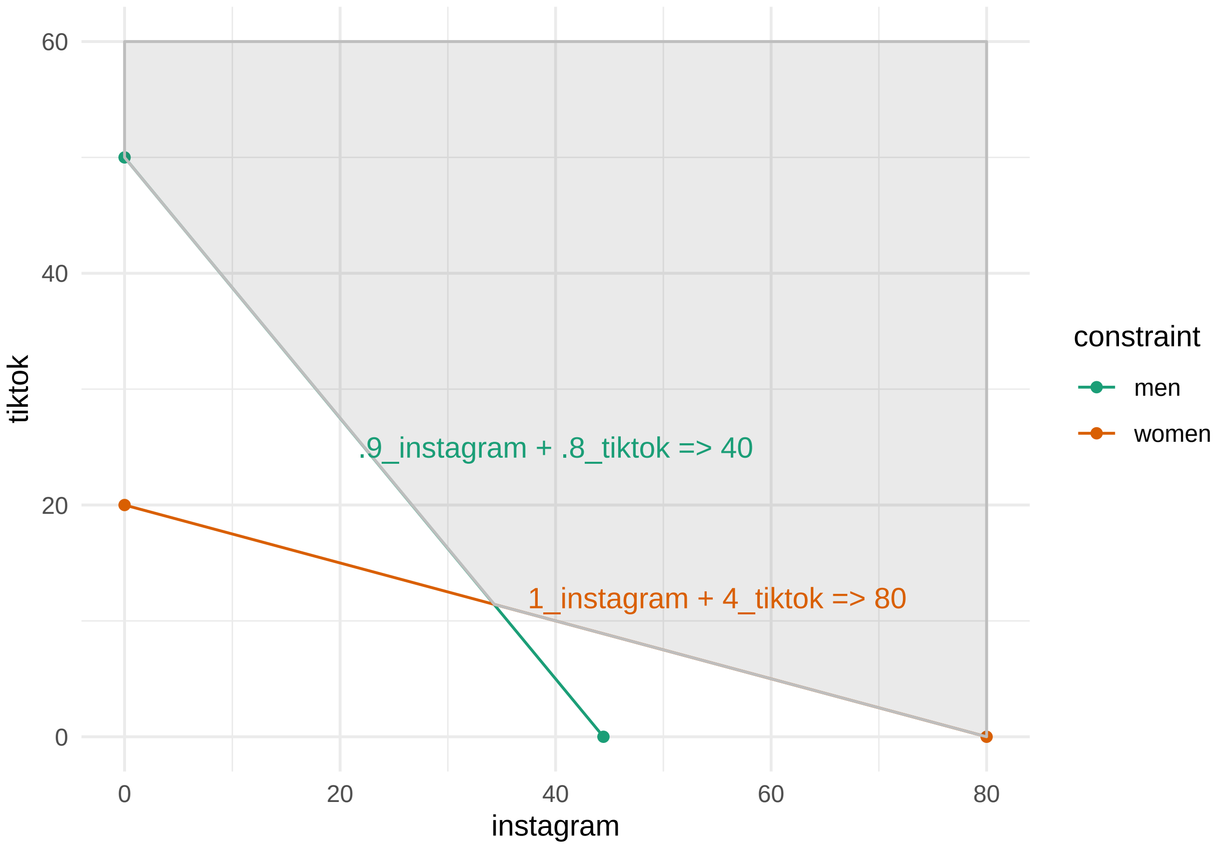
“Plot them”, and Jun went back to writing:



"Now that we have those lines plotted, we can clearly see where we might find our answer!", beamed Jun.

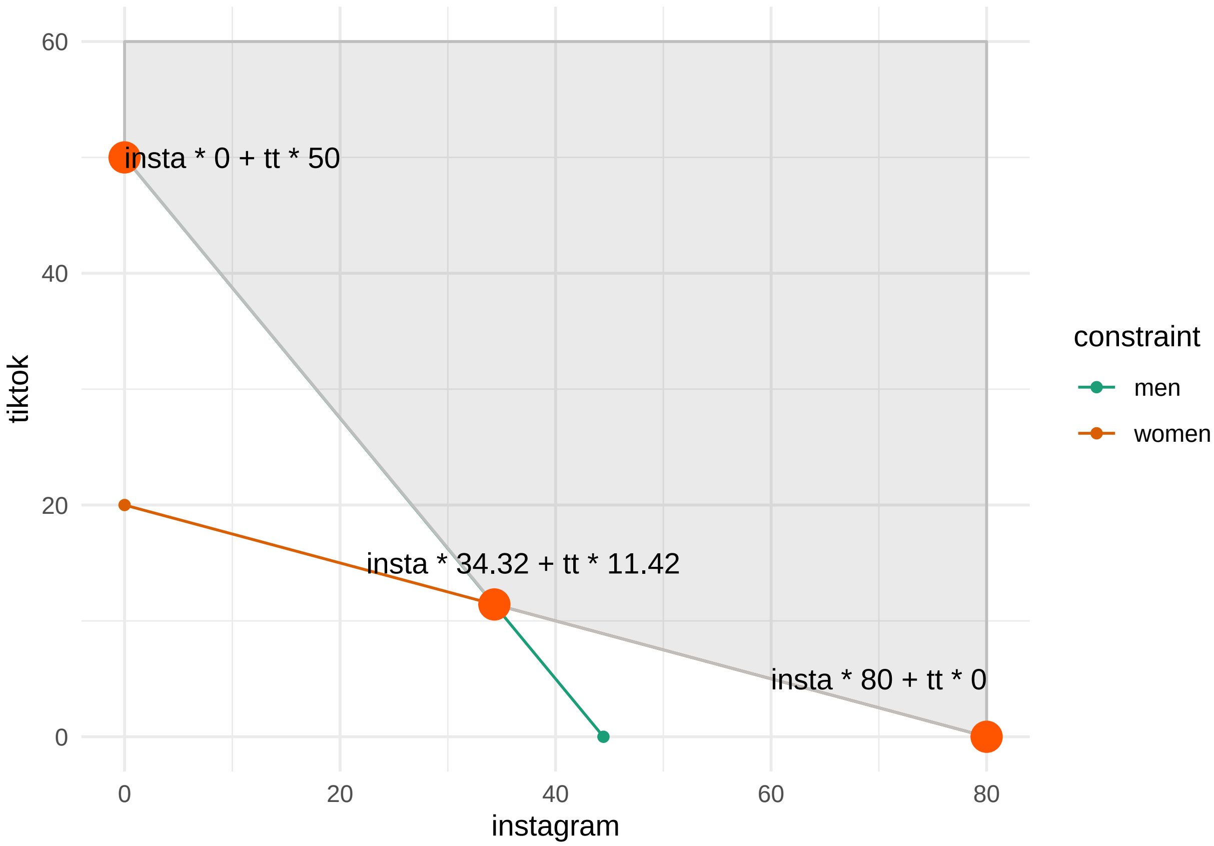
"Umm... it's a little fuzzy", admitted a puzzled Ali.

"Completely to be expected", Jun smiled. "Let's find the **feasible region** – this is the place where our answer lives!"



"We could go through and try every single set of points in this shaded area, but we would never finish and we would be wasting our time", Jun chuckled and continued, "We already know that we are looking for the values that minimize our solution, so we can completely ignore every point that doesn't sit on our lines."

Jun made a few circles on the plot and said, "These points will give us our answer!"



"This is called the **extreme point theorem** and it basically says that our **optimal** solution has to rest somewhere in the extreme points of the feasible region", said Jun, "and it makes life so much easier for finding our answer."

"We can just do the simple math now", and Jun wrote"

```
insta_cost = 50

tt_cost = 20

insta_cost * 0 + tt_cost * 50
```

```
[1] 1000
```

```
insta_cost * 34.32 + tt_cost * 11.42
```

```
[1] 1944.4
```

```
insta_cost * 80 + tt_cost * 0
```

```
[1] 4000
```

"Which of those is the smallest value?", Jun asked.

The conference room glowed with Ali's excitement! "I got it now!", Ali exclaimed. "We can get our optimal value of 1000 by purchasing 50 ads on TikTok and 0 on Instagram! The solution was correct!"

"Ahhh!", Jun started laughing, "it is definitely the optimal solution, but do you *really* think that it's the correct solution?" Jun shook his head and continued laughing, "How do you think Tolu is going to take the advice to not put anything at all on Instagram?"

via GIPHY

Ali's mind was sufficiently wrecked. How could an optimal answer not be the correct answer? Stupid analytics – nothing can ever be easy.

"What's the best path forward, then?", Ali asked Jun. "Simple", Jun replied, "ask how much they want to put on Instagram and that becomes a constraint!"

This was to be an important lesson for Ali: analytics tasks are never a one-shot deal. Clarity needs to be sought before most work can actually happen.

After a quick email exchange with Rayan from Marketing, Ali found out that at least 10 ads were needed for Instagram.

3.1.5 ClassOverflow

Let's spend some time helping Ali. We need to do two things: 1) specify an appropriate model and 2) solve it.

We will do this two different ways; both are good to know, but I'd imagine that you will find one to be more valuable than the other.

3.1.6 Using Python

A great chunk of Ali's coursework was in R, with just some excursions into Python. For statistics, R reigns supreme (but statsmodels in Python is really pretty solid). For machine learning, take your pick (only fanboys speak in absolutes about one being better than the other – both have their pros and cons). Linear programming is a bit different. R has some clear advantages in terms of flexibility, but Google has put effort towards implementing their GLOP solver in Python (among other languages).

The `pulp` package is going to look different than what we saw in R, but there are some definite improvements in expressing our model:

```python
from pulp import *

model = LpProblem(name = "test-model",
  sense = LpMinimize)

x = LpVariable(name = "instagram", lowBound = 0)
y = LpVariable(name = "tiktok", lowBound = 0)

model += (1 * x + 4 * y >= 80, "women")
model += (.9 * x + .8 * y >= 40, "men")

obj_func = 50 * x + 20 * y
model += obj_func

model

status = model.solve()

model.objective.value()

x.value()
y.value()

for var in model.variables():
  print(f"{var.name}: {var.value()}")
```

We really aren't breaking our problem down into small objects here. Instead, all we really need to do is to take our math form and pop that into our `model` – pretty easy stuff.

Finally, here is Google's OR tools. It is the current SoTA for optimization (how do you think Google gets people navigated). You'll notice that we aren't really doing anything too different than what we saw with `pulp`:

```
from ortools.linear_solver import pywraplp

solver = pywraplp.Solver.CreateSolver('GLOP')
x = solver.NumVar(0, solver.infinity(), 'instagram')
y = solver.NumVar(0, solver.infinity(), 'tiktok')

solver.NumVariables()
```

```
2
```

```
solver.Add(1 * x + 4 * y >= 80)
```

```
<ortools.linear_solver.pywraplp.Constraint; proxy of <Swig Object of type 'operations_research::MPConst
```

```
solver.Add(.9 * x + .8 * y >= 40)
```

```
<ortools.linear_solver.pywraplp.Constraint; proxy of <Swig Object of type 'operations_research::MPConst
```

```
solver.NumConstraints()
```

```
2
```

```
solver.Minimize(50 * x + 20 * y)

status = solver.Solve()

solver.Objective().Value()
```

```
999.9999999999999
```

```
x.solution_value()
```

```
0.0
```

```
y.solution_value()
```

```
49.99999999999999
```

Chapter 4

Process Simulation

Ali was absolutely smoked (no pun intended) after handling all of that optimization – hopefully some more standard modeling would come through. High hopes are always short lived, though, and Ali was thrown right back into the dark arts of Operations-based research.

During an "all-hands" meeting, Ali had the chance to listen to Rene, the Director of Retail Analytics, talk about store efficiency. Rene explained the process that typically happens.

"Our average store front is pretty small and we need to be careful about how many people are inside at any one time", Rene started. "We can't have more than 8 people waiting in the lobby before their ID's are checked", and Rene continued, "we don't want to turn people away, so we need to get more efficient in ID checks and bud-tending".

Ali was feeling good after some success and wasn't afraid to ask some questions – "Can you explain the process to me?", Ali asked.

"Sure", Rene said. "It is pretty easy, people walk in the door and wait for their ID to be checked."

"Once their ID has been checked they can meet with one of our bud-tenders – they pick their poison and then pay."

Ali was drawing the process out and made sure that it was correct:

"Seem about right?", Ali asked.

```r
library(DiagrammeR)

grViz("
digraph {
  graph [overlap = true, fontsize = 10, rankdir = LR]

  node [shape = box, style = filled, color = black, fillcolor = aliceblue]
  A [label = 'ID Check Line']
  B [label = 'ID Check']
  C [label = 'Bud Tender Line']
  D [label = 'Bud Tender']
  E [label = 'Pay']

  A->B B->C C->D D->E
}
")
```

"That's right", Rene said.

"How many people are checking IDs and how many bud tenders do you have?", Ali asked.

"It kinda depends", Rene said, "but it is usually just one person checking IDs and usually two bud tenders."

Ali had one more question: "How long do each of those steps take?"

"Uhhhhh... I'll have to ask around and get back with you", Rene noted.

"Most excellent", Ali thought, "that will give me some time to chat with Alex." Alex was the resident expert of simulations of all kind, so Ali knew an ally was there.

4.1 Discrete Event Simulation

When Ali finally caught up with Alex, Alex was only too happy to share some of the finer points on process simulation. First, Alex made note that the particular type of simulation under conversation wasn't just a process simulation, but was something called *discrete event simulation* (DES) – events are individual processes and some items goes through a series of those individual processes.

"It has roots in manufacturing, but some many things in life can be modeled through DES", Alex said.

"Let's start with something horribly boring... lines."

4.2 Queueing Theory

"There is a whole field of study regarding lines", Alex said, "and it is called *queueing theory*."

"We don't have to get crazy, but there is this thing called Kendall's Notation... it just describes the particular parts of how lines form and the distributions that guide them."

Ali thought, "The theory of lines... how do some people ever find love."

Alex could almost feel Ali's thought and said, "It really is more interesting than it sounds."

"Check this out!", and Alex was off to the races.

$A/B/C/D$

Where:

$A = arrival\, process$

$B = service\, process$

$C = server\, number$

$D = que\, capacity$

$M/D/k$

$M/M/k$

M generally stands for Markov or Exponential

D is deterministic: all jobs require a fixed amount of time.

k is the number of servers/workers/etc.

"Both of these are generally assumed to have an **infinite queue**... that is important to remember."

"If a queue is $M/D/k$, we can easily compute some helpful statistics."

$\lambda =$ arrival rate

$\mu =$ service rate

$\rho = \frac{\lambda}{\mu} =$ utilization

Average number of entities in the system is:

$$L = \rho + \frac{1}{2}\left(\frac{\rho^2}{1-\rho}\right)$$

Average number in queue:

$$L_Q = \frac{1}{2}\left(\frac{\rho^2}{1-\rho}\right)$$

Average system waiting time:

$$\omega = \frac{1}{\mu} + \frac{\rho}{2\mu(1-\rho)}$$

Average waiting time in queue:

$$\omega_Q = \frac{\rho}{2\mu(1-\rho)}$$

"The equations are not the important part here", Alex said, "but the idea that the equation captures is critical."

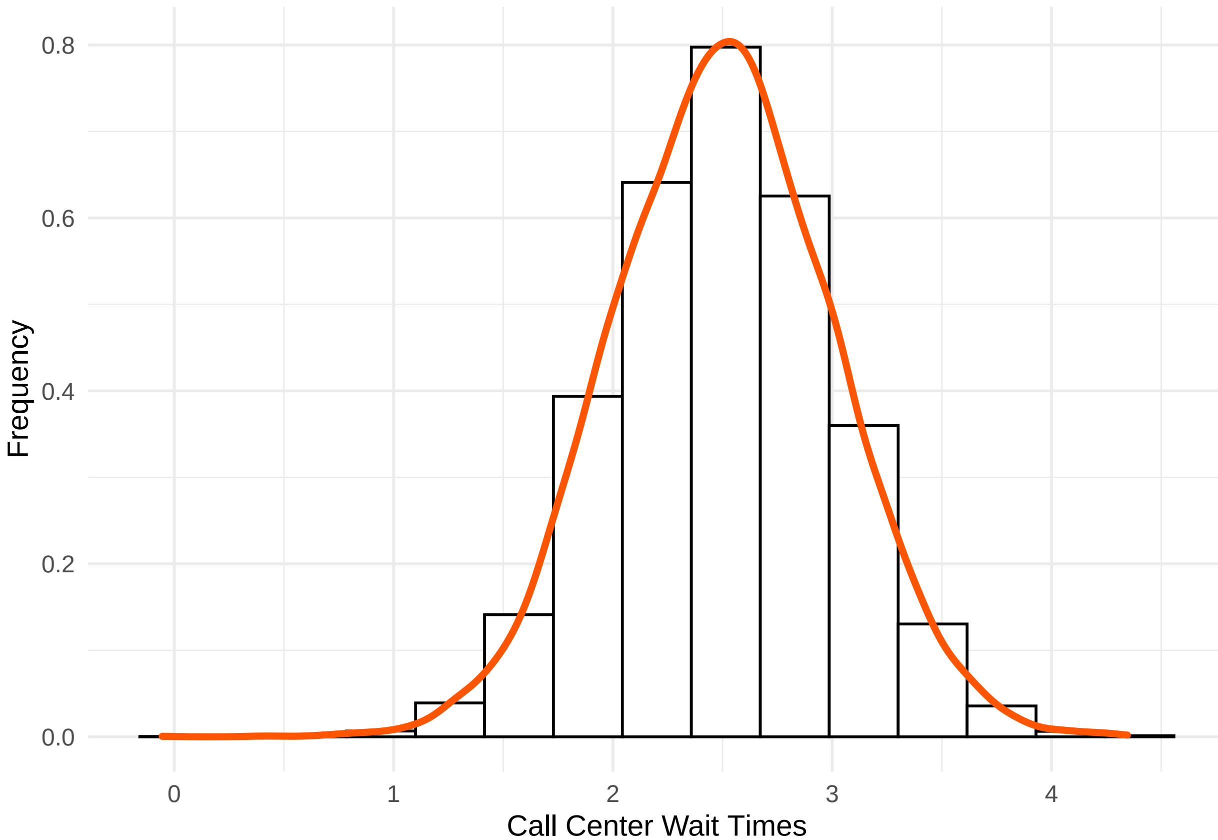
4.3 Distributions

"Distributions drive every single part of DES – every event that you can ever imagine comes from some type of distribution."

4.3.1 Normal Distribution

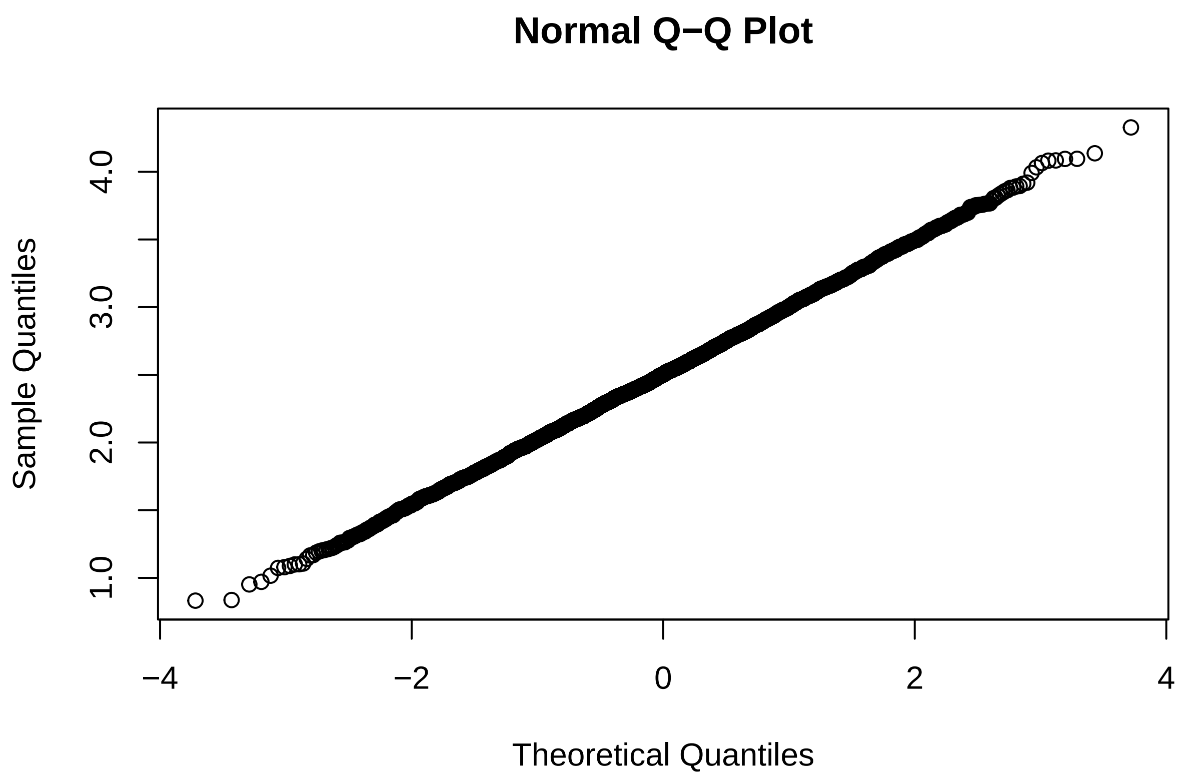
"For our normal distribution, we know the μ and σ."

"It is definitely the most common and you'll find that a lot of processes are normally distributed."

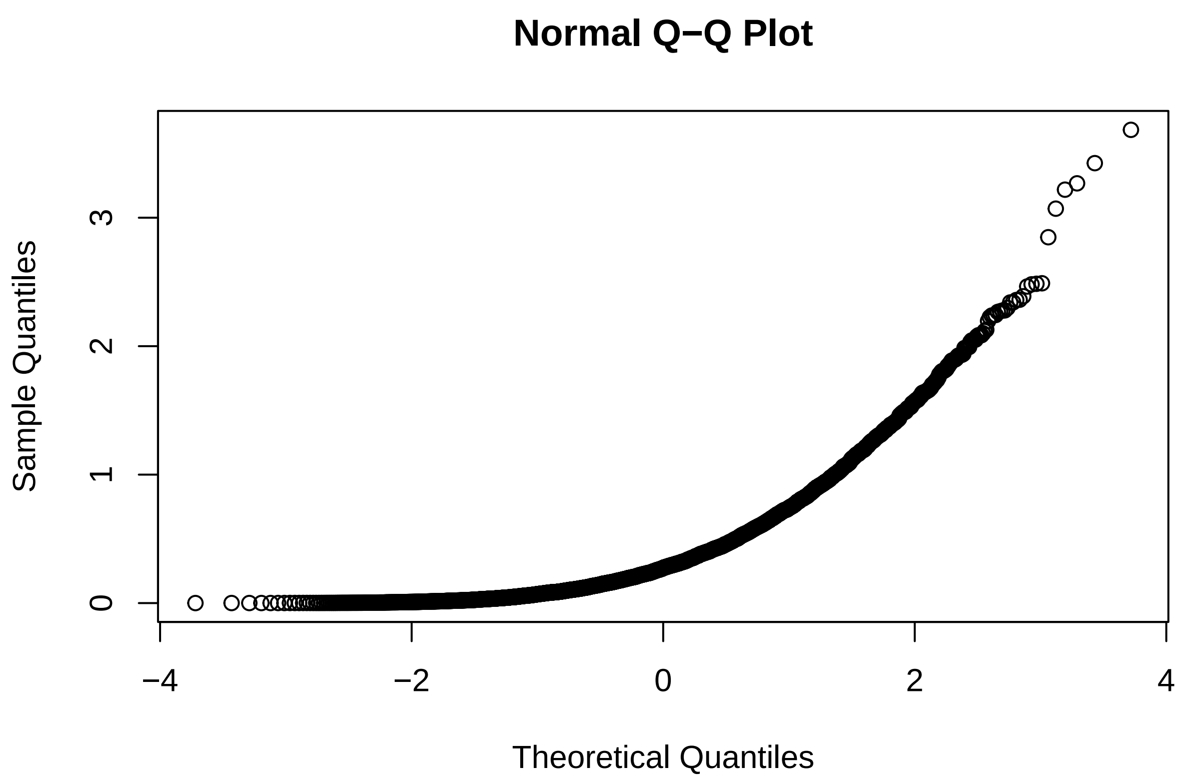


"You might get process data and want to test if a variable is normally distributed."

```
# Just creating data from a normal distribution:

normal_variable <- rnorm(n = 5000, mean = 2.5, sd = .5)

# And an exponential distribution:

exponential_variable <- rexp(n = 5000, rate = 2.5)

# We can check both with a qq plot for normality:
qqnorm(normal_variable)
```




```
qqnorm(exponential_variable)
```




```
# You'll notice the normal distribution
# just plots a straight, diagonal line.
```

```r
# The shapiro test will give a test statistic:
shapiro.test(normal_variable)
```

```
    Shapiro-Wilk normality test

data:  normal_variable
W = 0.99974, p-value = 0.8188
```

```r
shapiro.test(exponential_variable)
```

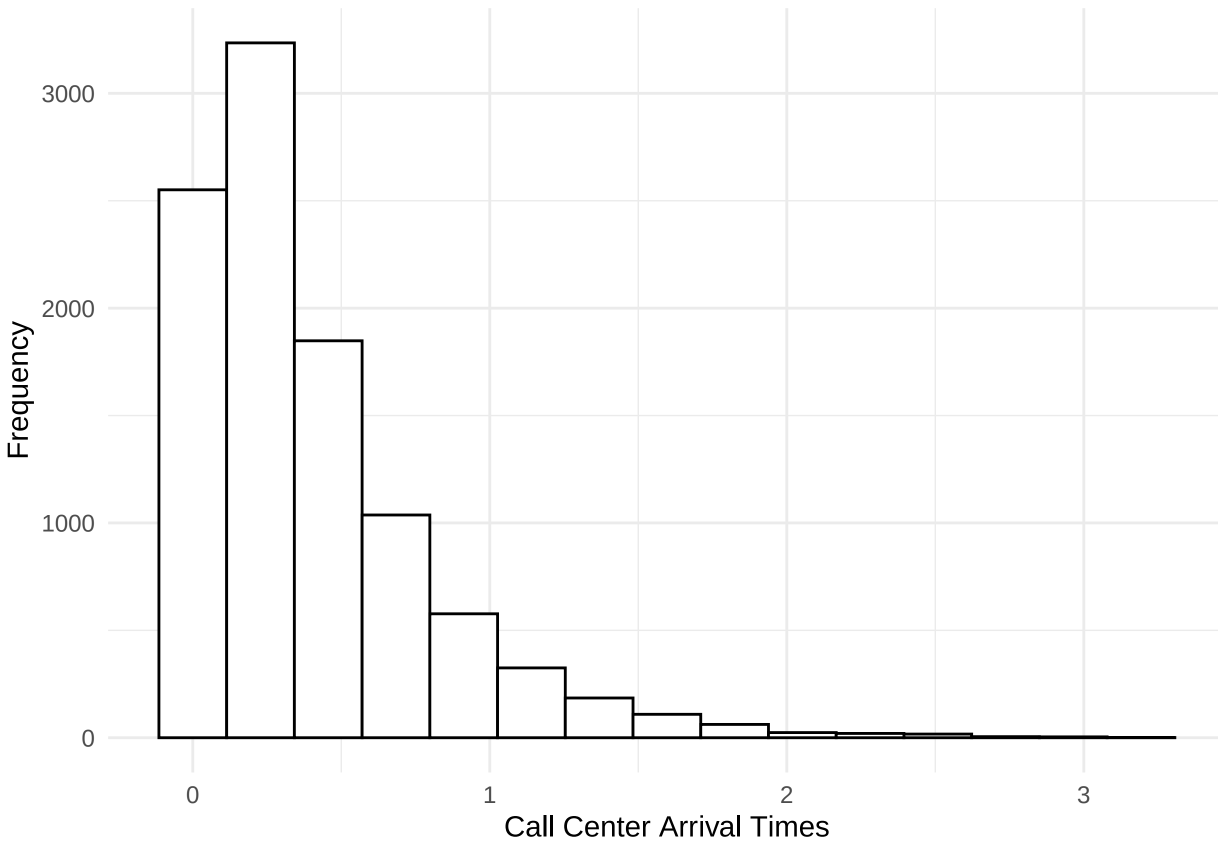
```
    Shapiro-Wilk normality test

data:  exponential_variable
W = 0.80629, p-value < 2.2e-16
```

```r
# An insignificant p-value would indicate
# no difference from normality.
```

4.3.2 Exponential Distribution

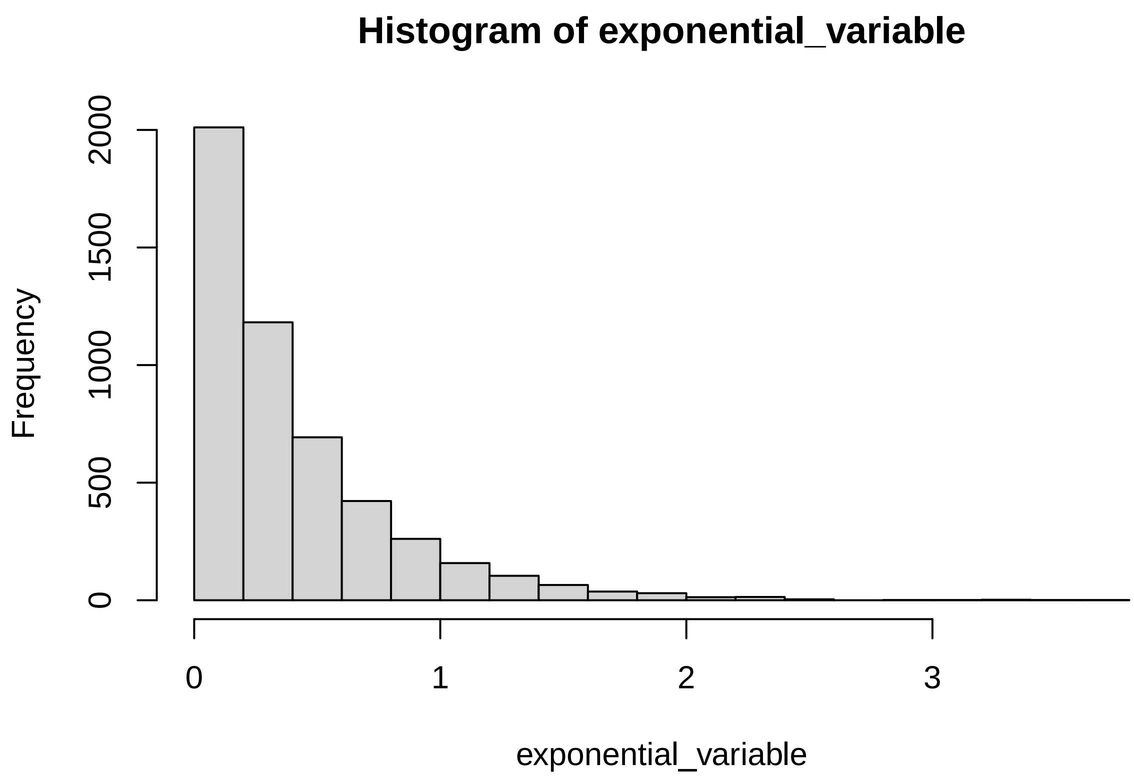
"We can only know one thing about the exponential distribution: μ (also expressed as a rate)."

"Just about any arrival process can be approximated by an exponential distribution."



"Wanna test it?"

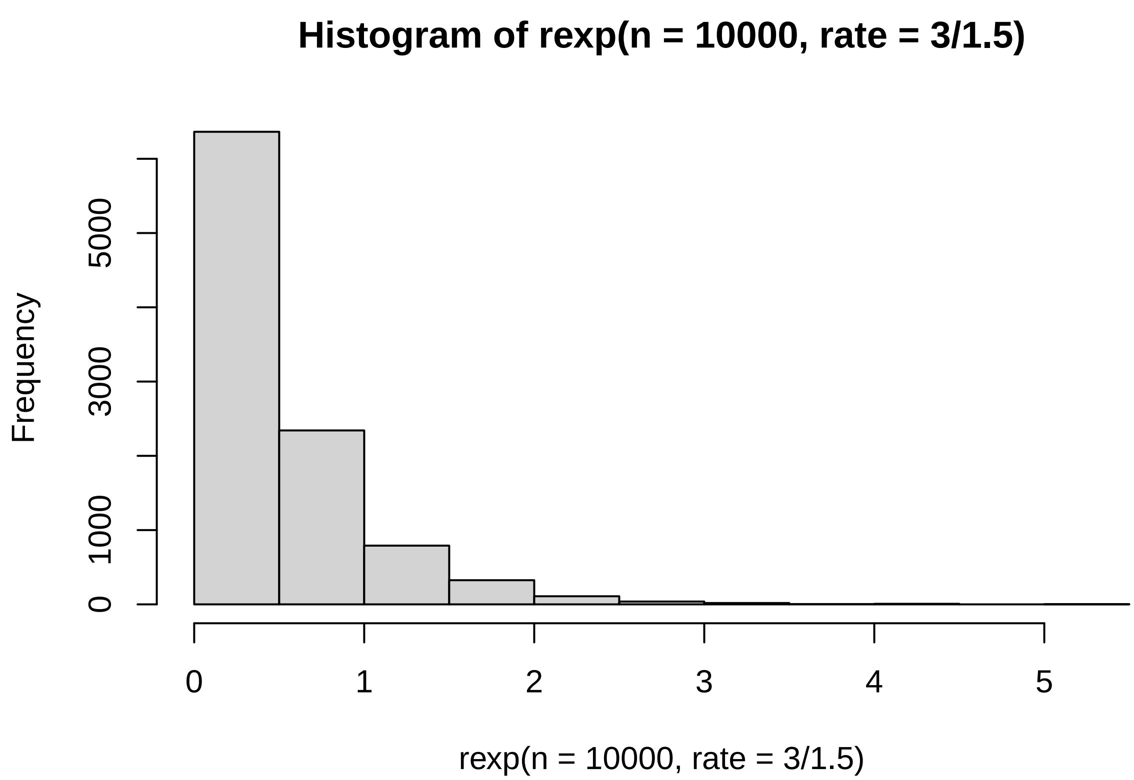
```r
hist(exponential_variable)
```



"That rate parameter is a bit confusion...the easiest way to express it is as a ratio."

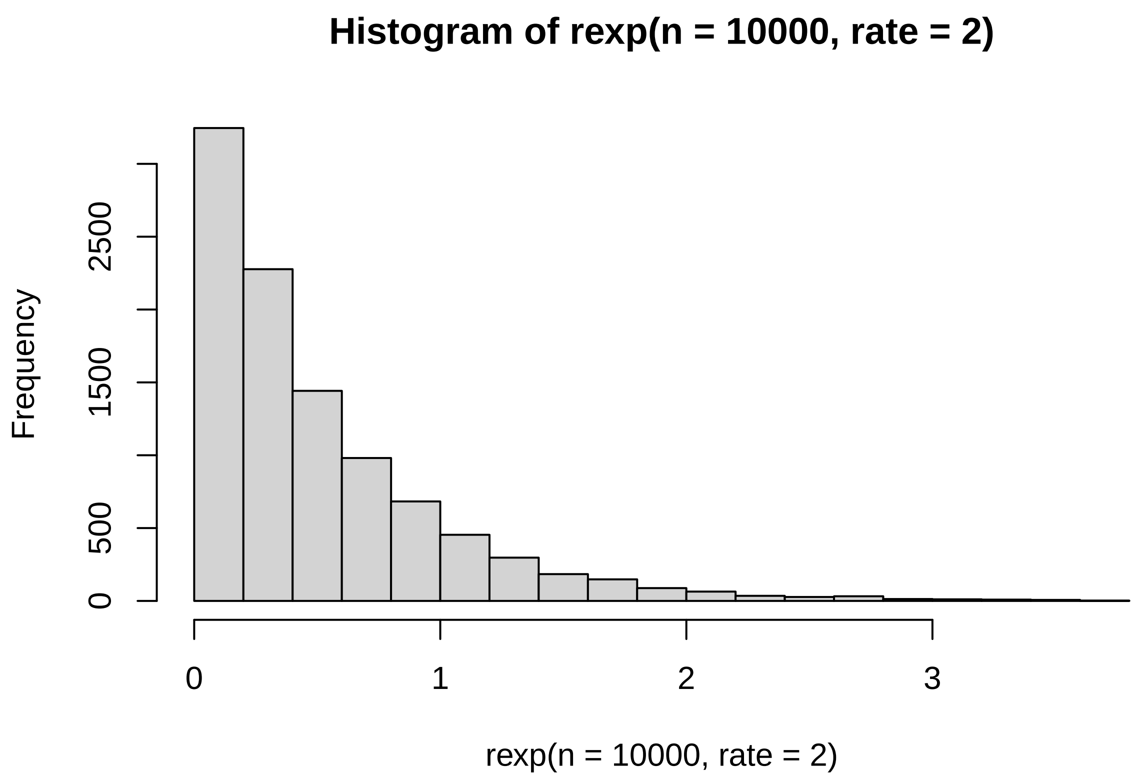
"If 3 people enter the line every minute and a half, than it would be a rate of 3/1.5".

```
hist(rexp(n = 10000, rate = 3 / 1.5))
```



"You could also just put the average rate in."

```
hist(rexp(n = 10000, rate = 2))
```

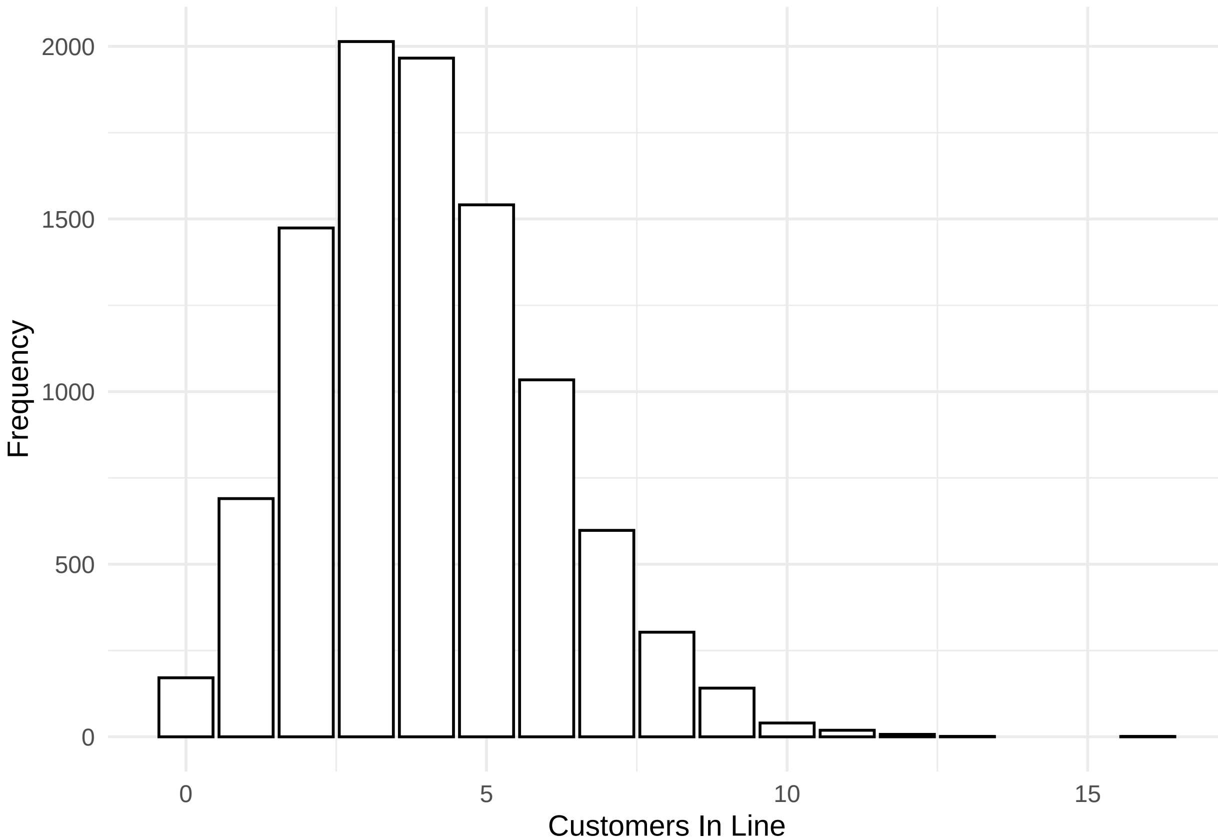


"They are really the same thing."

4.3.3 Poisson Distribution

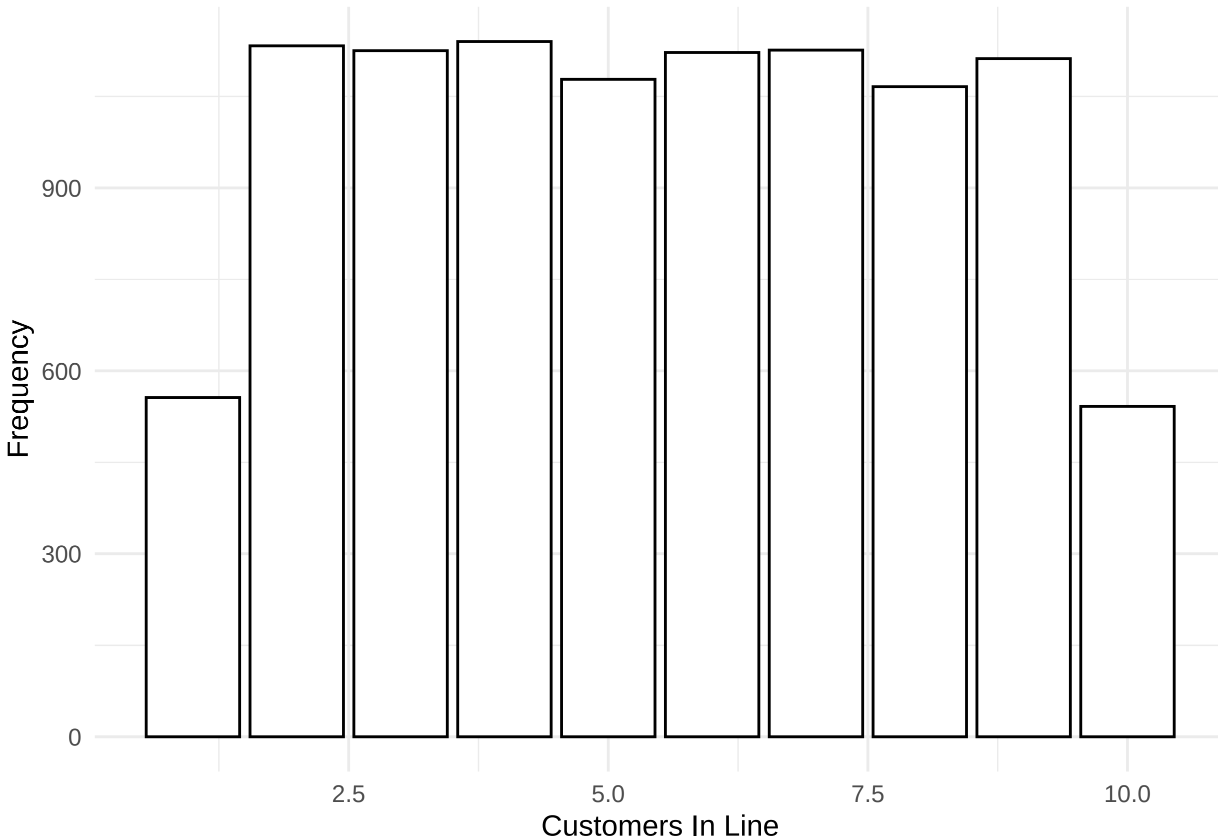
"The poisson is an interesting distribution – it tends to deal with count-related variables. It tells us the probability of a count occurring. We know its λ (again, a rate)."

"The λ is just a fancy way of saying the average number of events, or the incidence rate."



4.3.4 Uniform Distribution

"While people tend to think about the Gaussian distribution as the most vanilla of all distributions, it really is not – I would say that distinction belongs to the uniform distribution. We don't even get any fancy Greek letters, just a minimum and a maximum. Why? Because knowing the min and max will tell us that there is an equal probability of drawing a value anywhere within that range."



4.4 Performance

The *service level* for each simulation is the fraction of the demand that is satisfied.

$$Entrance\ Service\ Level = \frac{Objects\ Entering}{Objects\ Entering + Objects\ Unable\ To\ Enter}$$

Here, we are looking at the number of people who wanted to join the process, but could not. If we have a service level of 1, then 100% of objects were able to get into the process. A service level of .5 would indicate that only 50% of objects were able to enter.

The *overall mean service level* of the process is the mean of the service levels calculated from each simulation.

The *mean cycle time* at a buffer is the mean amount of time an object takes to move through the buffer during a simulation.

The *overall mean cycle time* at a buffer is the mean of the mean cycle time of the buffer for each simulation.

You will see different words for lines: buffers and queues. Just know that they are used interchangeably.

4.5 The Dispensary

The interarrival times for customers follows an exponential distribution with a rate of 1 person every 1.5 minutes.

The dispensary cannot hold any more than 8 people, for safety reasons. If a person arrives when the line is full, that person will not get in line.

The ID check is approximately normal with $\mu = 15\,seconds$ and $\sigma = 3\,seconds$. Once a person has their ID checked, they can sit in the lobby and there are 10 seats in the lobby.

The bud tender's service time is $\mu = 2.4\,minutes$ and $\sigma = .5\,minutes$.

Paying generally follows a uniform distribution, with a minimum of 5 seconds and a maximum of 15 seconds.

```
grViz("
digraph {
  graph [overlap = true, fontsize = 10, rankdir = LR]

  node [shape = box, style = filled, color = black, fillcolor = aliceblue]
  A [label = 'ID Check Line']
  B [label = 'ID Check']
  C [label = 'Bud Tender Line']
  D [label = 'Bud Tender']
  E [label = 'Pay']

  A->B B->C C->D D->E
}
")
```

4.5.1 ClassOverflow

We will need the `simmer` package for our simulation:

```
install.packages("simmer")
```

Once we have `simmer` installed, we need to load it:

```
library(simmer)
```

Let's start by defining a customer's trajectory. First, we will provide a name for `trajectory()`.

```
customer <- trajectory("Customer path")
```

Next, we need to initiate a start time with `set_attribute()` – we will use `now()` to specify our not-yet-created dispensary object.

```
customer <- trajectory("Customer path") |>
  set_attribute("start_time", function() {now(dispensary)})
```

After establishing our time, the next step for a customer is to `seize()` the "teller" (which we will define later).

```
customer <- trajectory("Customer path") |>
  set_attribute("start_time", function() {now(dispensary)}) |>
  seize("id_check")
```

Now things start to get tricky. We need to use `timeout()` to specify how long a customer is using the id check – this is the check's average working time.

We can specify how long an id check is seized (i.e., how long the check is working) – we provide a distribution with the appropriate values.

```
customer <- trajectory("Customer path") %>%
  set_attribute("start_time", function() {now(dispensary)}) |>
  seize("id_check") |>
  timeout(function() {rnorm(n = 1, mean = 15/60, sd = 3/60)})
```

After a customer spends time with the teller, the customer releases the counter.

```
customer <- trajectory("Customer path") %>%
  set_attribute("start_time", function() {now(dispensary)}) |>
  seize("id_check") |>
  timeout(function() {rnorm(n = 1, mean = 15/60, sd = 3/60)}) |>
  release("id_check")
```

From there, we can add the additional resources to our model:

```
customer <- trajectory("Customer path") %>%
  set_attribute("start_time", function() {now(dispensary)}) |>
  seize("id_check") |>
  timeout(function() {rnorm(n = 1, mean = 15/60, sd = 3/60)}) |>
  release("id_check") |>
  seize("bud_tender") |>
  timeout(function() {rnorm(n = 1, mean = 2.4, sd = .5)}) |>
  release("bud_tender") |>
  seize("payment") |>
  timeout(function() {runif(n = 1, min = 5/60, max = 15/60)}) |>
  release("payment")
```

This is all we need to do for a customer, so now we can turn our attention to the dispensary.

Our dispensary is going to provide the environment that houses our trajectory. So, we can start by creating an environment with `simmer()`:

```
dispensary <- simmer("dispensary")
```

Once we have our simulation environment defined, we can add resources to it with the aptly-named `add_resources()` function. This is where we will specify what is being seized by our customer. We need to provide some additional information to our resource: `capacity` and `queue_size`.

```
dispensary <- simmer("dispensary") |>
  add_resource("id_check", capacity = 1, queue_size = 8) |>
  add_resource("bud_tender", capacity = 2, queue_size = 10) |>
  add_resource("payment", capacity = 2)
```

To this point, we have our customer behavior (how they move through our process) and information about our work stations. The last detail is the inter-arrival time, which we can specify with `add_generator()`. It works in very much the same way that `timeout()`, in that we are specifying a distribution. The `rexp` function in R takes a rate. If we remember that, on average, one person comes into the dispensary every two minutes, we can define our rate as $\frac{1}{2}$.

Try this: mean(rexp(n = 10000, rate = 1/2))

```
dispensary <- simmer("dispensary") |>
  add_resource("id_check", capacity = 1, queue_size = 8) |>
  add_resource("bud_tender", capacity = 2, queue_size = 10) |>
  add_resource("payment", capacity = 2) |>
  add_generator("Customer", customer, function() {
    c(0, rexp(n = 100, rate = 1/1.5), -1)
  })
```

Now we can run our simulation; we just need to provide a time value for the `until` argument. Let's say we want to run this simulation for 2 hours.

```
run(dispensary, until = 120)
```

If we put it together, here is what we have:

```
customer <- trajectory("Customer path") %>%
  set_attribute("start_time", function() {now(dispensary)}) |>
  seize("id_check") |>
  timeout(function() {rnorm(n = 1, mean = 15/60, sd = 3/60)}) |>
  release("id_check") |>
  seize("bud_tender") |>
  timeout(function() {rnorm(n = 1, mean = 2.4, sd = .5)}) |>
  release("bud_tender") |>
  seize("payment") |>
  timeout(function() {runif(n = 1, min = 5/60, max = 15/60)}) |>
  release("payment")

dispensary <- simmer("dispensary") |>
  add_resource("id_check", capacity = 1, queue_size = 8) |>
  add_resource("bud_tender", capacity = 2, queue_size = 10) |>
  add_resource("payment", capacity = 2) |>
  add_generator("Customer", customer, function() {
    c(0, rexp(n = 100, rate = 1/1.5), -1)
  })

run(dispensary, until = 120)
```

```
simmer environment: dispensary | now: 120 | next: 121.437156388327
{ Monitor: in memory }
{ Resource: id_check | monitored: TRUE | server status: 0(1) | queue status: 0(8) }
{ Resource: bud_tender | monitored: TRUE | server status: 0(2) | queue status: 0(10) }
{ Resource: payment | monitored: TRUE | server status: 0(2) | queue status: 0(Inf) }
{ Source: Customer | monitored: 1 | n_generated: 101 }
```

Finally, we can start to look at our data:

```
result <- get_mon_arrivals(dispensary)

head(result)
```

```
       name start_time  end_time activity_time finished
1 Customer0   0.000000  3.147665      3.147665     TRUE
2 Customer1   3.596104  6.693559      3.097455     TRUE
3 Customer2   5.084789  8.013115      2.928327     TRUE
4 Customer3   5.801226  8.798140      2.522346     TRUE
5 Customer4   5.892428 10.449801      2.856625     TRUE
6 Customer5   6.180949 11.585012      3.209664     TRUE
  replication
1           1
2           1
3           1
4           1
5           1
6           1
```

Let's calculate a few things. First, let's how many people made it through:

```
nrow(result[result$finished == TRUE, ])
```

```
[1] 73
```

Now we can check our service level:

```
nrow(result[result$finished == TRUE, ]) / nrow(result)
```

```
[1] 1
```

The `nrow` function will tell us how many rows are in the data. In the numerator, we filtered those rows were finished was equal to TRUE (giving us the number of people who made it into the system).

Now we need to calculate how long each person was in line.

```
result$wait_time <- result$end_time - result$start_time - result$activity_time
```

Now, we can find the average wait time. We only want to do it for those who actually made it into the system though!

```
completeOnly <- result[result$finished == TRUE, ]

mean(completeOnly$wait_time)
```

```
[1] 1.463238
```

That gives us all of the information that we need for this bank configuration.

But…that is just one simulation. We really need to run this many times to get an idea about the distribution of outcomes.

We have a few choices. One choice is that we just replicate our procedure a certain number of times:

```
sim50Runs <- replicate(50, expr = {
  customer <- trajectory("Customer path") %>%
    set_attribute("start_time", function() {now(dispensary)}) |>
    seize("id_check") |>
    timeout(function() {rnorm(n = 1, mean = 15/60, sd = 3/60)}) |>
    release("id_check") |>
    seize("bud_tender") |>
    timeout(function() {rnorm(n = 1, mean = 2.4, sd = .5)}) |>
    release("bud_tender") |>
    seize("payment") |>
    timeout(function() {runif(n = 1, min = 5/60, max = 15/60)}) |>
    release("payment")

  dispensary <- simmer("dispensary") |>
    add_resource("id_check", capacity = 1, queue_size = 8) |>
    add_resource("bud_tender", capacity = 2, queue_size = 10) |>
    add_resource("payment", capacity = 2) |>
    add_generator("Customer", customer, function() {
      c(0, rexp(n = 100, rate = 1/1.5), -1)
    })

  run(dispensary, until = 120)

  result <- get_mon_arrivals(bank)

}, simplify = FALSE)
```

Chapter 5

Integer Programming

5.1 Troubling Solutions

Ali had figured out the whole linear programming thing and all felt good – until a troubling solution came back.

As per usual, trouble began with an email:

> What's good Ali? I'm Bao, from our retail operations group! We heard that you are a wizard with this stuff, so we are hoping that you can help us out.
>
> We have an issue in many of our stores: we often find that we are overstocked on certain edibles, but we can't keep others on the shelf. We'd love to balance that out, but aren't really sure if saying, "Just change production", is the answer. No matter what we make for batches, we would obviously like to do it for as cheap as possible.
>
> Here is the basics of what we have to deal with:
>
> 1. We have two primary classes of edibles: gummies and candy bars.
> 2. We aren't really worried about ingredients other than what is grown in our greenhouses. Food supplies are easy, but green supplies aren't.
> 3. To make a batch of candy, it requires 4 grams of raw flower, 1 gram of distillates, and 1 gram of pressed trichromes.
> 4. To make a batch of gummies, it requires 3 grams of raw flower and 1 gram of distillates.
> 5. Every month, we start with 200 grams of raw flower, 500 grams of distallate, and 100 grams of pressed trichromes.
> 6. If a store needs more, they can purchase a standard bag of raw flower for 80 dollars per bag.
> 7. A bag of raw flower can produce 10 grams of distallate, 20 grams of raw flower, and 2 grams of pressed trichromes
> 8. Producing candy costs 30 dollars per batch; producing gummies costs 40 dollars a batch
> 9. We need at least 1000 selling units per month.
>
> We owe you one!
>
> Bao

"Oh yeah", though Ali, "this is going to be a walk in the park."

```r
library(linprog)

cvec <- c(candy = 30,
          gummies = 40,
          bag = 80)

bvec <- c(distillate = 500,
          flower = 200,
          trichromes = 100,
          batch_need = 1000)

constDirs <- c("<=", "<=", "<=", ">=")

aMat <- rbind(dis_const = c(1, 1, -10),
              flow_const = c(4, 3, -20),
              trich_const = c(1, 0, -2),
              batch_const = c(1, 1, 0))

solveLP(cvec, bvec, aMat, maximum = FALSE,
        const.dir = constDirs)
```

```
Results of Linear Programming / Linear Optimization

Objective function (Minimum): 48666.7

Iterations in phase 1: 2
Iterations in phase 2: 1
Solution
            opt
candy   422.222
gummies 577.778
bag     161.111

Basic Variables
                  opt
candy         422.222
gummies       577.778
bag           161.111
S distillate 1111.111

Constraints
              actual dir bvec    free     dual dual.reg
distillate -611.111  <=   500 1111.11  0.00000  1111.11
flower      200.000  <=   200    0.00  3.33333  5200.00
trichromes  100.000  <=   100    0.00  6.66667   380.00
batch_need 1000.000  >= 1000    0.00 50.00000      Inf

All Variables (including slack variables)
                  opt cvec      min.c max.c    marg marg.reg
candy         422.222   30  -60.00000    36      NA       NA
gummies       577.778   40  -46.00000    70      NA       NA
bag           161.111   80 -140.00000   200      NA       NA
S distillate 1111.111    0   -6.00000    12 0.00000       NA
```

```
S flower        0.000    0   -3.33333   Inf  3.33333     5200
S trichromes    0.000    0   -6.66667   Inf  6.66667      380
S batch_need    0.000    0  -50.00000   Inf 50.00000      Inf
```

That is great! Make 422.222 batches of candy, 577.778 batches of gummies, and buy 161.111 bags…

The first thing Ali though was, "What in the actual…those numbers cannot be correct. How do you make .222 of something? Is that acceptable? On second thought, could you ever buy a tenth of a bag?"

Just to be sure, Ali checked every value and everything matched just fine. So what could be causing the problem and how can it be fixed?

Looks like another visit to Jun…who just so happens to be on vacation.

5.2 ClassOverflow

Let's spend some time exploring some package functionality:

```
library(lpSolve)

test <- lpSolve::lp(direction = "min", objective.in = cvec,
            const.mat = aMat, const.dir = constDirs,
            const.rhs = bvec, all.int = TRUE, compute.sens = 1)
```

Remember ROI from last time? It has some pretty handy functionality.

```
library(ROI)
library(ROI.plugin.glpk)
```

We can use all of the objects that we have already created

```
model_constraints <- L_constraint(L = aMat,
                                  dir = constDirs,
                                  rhs = bvec)

model_creation <- OP(objective = cvec,
                     constraints = model_constraints,
                     types = rep("I", length(cvec)),
                     maximum = FALSE)

model_solved <- ROI_solve(model_creation)

solution(model_solved, "primal")
```

```
  candy gummies     bag
    420     580     161
```

```
solution(model_solved, "objval")
```

```
[1] 48680
```

Now, Ali knows to make 420 units of candy (coincidence?), 580 units of gummies, and 161 bags.

All we have done in either case is to constrain the possible solution set to only include integer values. The "how" of this is substantially more complicated.

What follows is purely a cursory glance – if you are interested in knowing more, you can check out the resources!

1. The most naive approach is to solve as a linear programming problem (i.e., LP relaxation) and then round the results
2. Some matrices are *totally unimodular* (no big concern for us) and will always return an integer value.
3. Cutting planes solve the LP relaxation, test if the optimal value is integer. A non-integer solution will get reworked as a constraint and this process continues until an optimal integer solution is found.
4. Branch and bound algorithms produces possible solution sets and transverses subsets of those sets to find an answer.
5. Branch and cut algorithms combine the previous 2 approaches.

5.3 Transportation Problems

After sending an updated solution to Bao, Ali felt some sense of relief – learning was happening and solutions were getting easier to come by. Unfortunately for Ali, stories of success spread wildly and it wasn't long until more people came knocking. The first request was from Castel, the director of grow operations; the second request was from Blaise, the director of Human Resources.

Castel's problem seemed pretty simple: there are greenhouses that contain raw product and that raw product needs to be shipped to different processing facilities. Each processing facility has a capacity need and there is a cost for moving products between the different facilities:

Castel drew a map for the cost to move product between greenhouses and processing facilities:

And then added the following notes:

Greenhouse 1 can only contribute 1200 pounds

Greenhouse 2 can only contribute 1000 pounds

Greenhouse 3 can only contribute 800 pounds

Processing facility 1 needs at least 1100 pounds

Processing facility 2 needs at least 400 pounds

Processing facility 3 needs at least 750 pounds

Processing facility 4 needs at least 750 pounds

How much should each greenhouse send to each processing facility and do it as cheaply as possible?

Ali had all of the necessary information, so took Castel's words and translated them into an expression:

$$
\begin{aligned}
M = 35_{x_{11}} + 30_{x_{12}} + 40_{x_{13}} + 32_{x_{14}} + 37_{x_{21}} + 40_{x_{22}} + \\
42_{x_{23}} + 25_{x_{24}} + 40_{x_{31}} + 15_{x_{32}} + 20_{x_{33}} + 28_{x_{34}} \\
subject\, to \\
X_{11} + X_{12} + X_{13} + X_{14} \leq 1200 \\
X_{21} + X_{22} + X_{23} + X_{24} \leq 1000 \\
X_{31} + X_{32} + X_{33} + X_{34} \leq 800 \\
X_{11} + X_{21} + X_{31} \geq 1100 \\
X_{12} + X_{22} + X_{23} \geq 400 \\
X_{13} + X_{23} + X_{33} \geq 750 \\
X_{14} + X_{24} + X_{34} \geq 750 \\
X_{ij} \geq 0
\end{aligned}
$$

```
cMat <- c(35, 30, 40, 32, 37, 40,
     42, 25, 40, 15, 20, 28)

b <- c(1200, 1000, 800, 1100, 400, 750, 750)

A <- rbind(c(1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0),
           c(0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0),
           c(0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1),
           c(1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0),
           c(0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0),
           c(0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0),
           c(0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1))

constraints <- L_constraint(A,
                            c(rep("<=", 3), rep(">=", 4)),
                            b)

model <- OP(objective = cMat,
            constraints = constraints,
            types = rep.int("I", length(cMat)),
            maximum = FALSE)

result <- ROI::ROI_solve(model, "glpk", verbose = TRUE)
```

```
<SOLVER MSG>  ----
GLPK Simplex Optimizer, v4.65
7 rows, 12 columns, 24 non-zeros
      0: obj =   0.000000000e+00 inf =   3.000e+03 (4)
      6: obj =   1.049000000e+05 inf =   0.000e+00 (0)
*    10: obj =   8.400000000e+04 inf =   0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
GLPK Integer Optimizer, v4.65
7 rows, 12 columns, 24 non-zeros
12 integer variables, none of which are binary
Integer optimization begins...
Long-step dual simplex will be used
+    10: mip =     not found yet >=              -inf        (1; 0)
+    10: >>>>>   8.400000000e+04 >=   8.400000000e+04   0.0% (1; 0)
+    10: mip =   8.400000000e+04 >=     tree is empty   0.0% (0; 1)
INTEGER OPTIMAL SOLUTION FOUND
<!SOLVER MSG> ----
```

```
result$objval
```

```
[1] 84000
```

```
solution(result)
```

```
 [1] 850 350   0   0 250   0   0 750   0  50 750   0
```

5.4 Binary Integer Programming

The *transportation problem* proved to be an easy one, once it was broken down into it's mathematical expression. Blaise's request, though, proved to be a bit more challenging.

> How's life, my unmet friend?
>
> We've got a situation in our *connoisseur's cabinet* – it is where we keep the expensive stuff.
>
> We only have room for a single feature cabinet, so we can't put everything that we have into it.
>
> I'd like to put things in there that won't take up a ton of space, but will also bring in the cash.
>
> The list of products, prices, and space is attached. I can't use any more than 10 spaces.
>
> Any ideas?
>
> Blaise

Turns out, that Blaise was just asking for some version of a *knapsack* problem:

```
special_items <- data.frame(item = c("cannabis_caviar", "oracle", "fruity_pebbles",
                                     "loud_dream", "white_fire", "j1",
                                     "hammerhead", "sista", "goblin",
                                     "fishermen", "cloud", "paradise"),
                            space = c(1.5, 1, 1,
                                      1.25, 1, 1,
                                      6, 5, 6,
                                      7, 3, 2),
                            value = c(800, 450, 400,
                                      400, 500, 350,
                                      1600, 2325, 1005,
                                      750, 250, 875))

constraints <- L_constraint(special_items$space, "<=", 10)

model <- OP(objective = special_items$value,
            constraints = constraints,
            types = rep.int("I", 12),
            maximum = TRUE)

solved_model <- ROI_solve(model)

solution(solved_model) |>
  setNames(special_items$item)
```

```
cannabis_caviar          oracle  fruity_pebbles
              6               0               0
     loud_dream      white_fire              j1
              0               1               0
     hammerhead           sista          goblin
              0               0               0
      fishermen           cloud        paradise
              0               0               0
```

```
solution(solved_model, "objval")
```

```
[1] 5300
```

Ali wondered, "What would happen if I changed that constraint to only be a 0 or a 1?"

```
model <- OP(objective = special_items$value,
            constraints = constraints,
            types = rep.int("B", 12),
            maximum = TRUE)
```

```r
solved_model <- ROI_solve(model)

solution(solved_model) |>
  setNames(special_items$item)
```

```
cannabis_caviar          oracle  fruity_pebbles
              0               1               1
     loud_dream      white_fire              j1
              0               1               0
     hammerhead           sista          goblin
              0               1               0
      fishermen           cloud        paradise
              0               0               1
```

```r
solution(solved_model, "objval")
```

```
[1] 4550
```

What should Ali do?

Chapter 6

Simulation

Why are you looking this far ahead?

Focus on the present instead.