Simulation and Optimization

Seth Berry

2021-09-22

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

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

“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:

$$\begin{aligned}\geq\\ \geq\end{aligned}$$

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

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

“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:

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

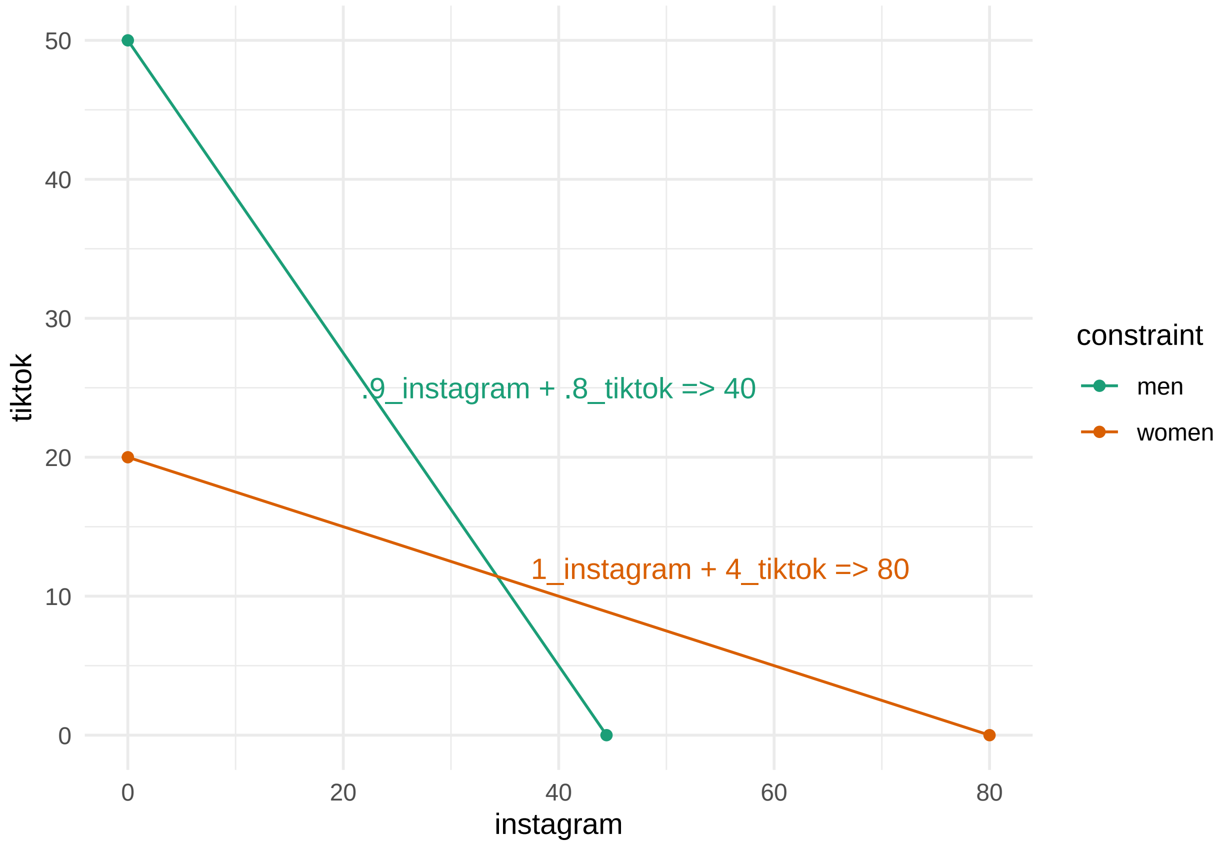
Can be solved with:

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

“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?”

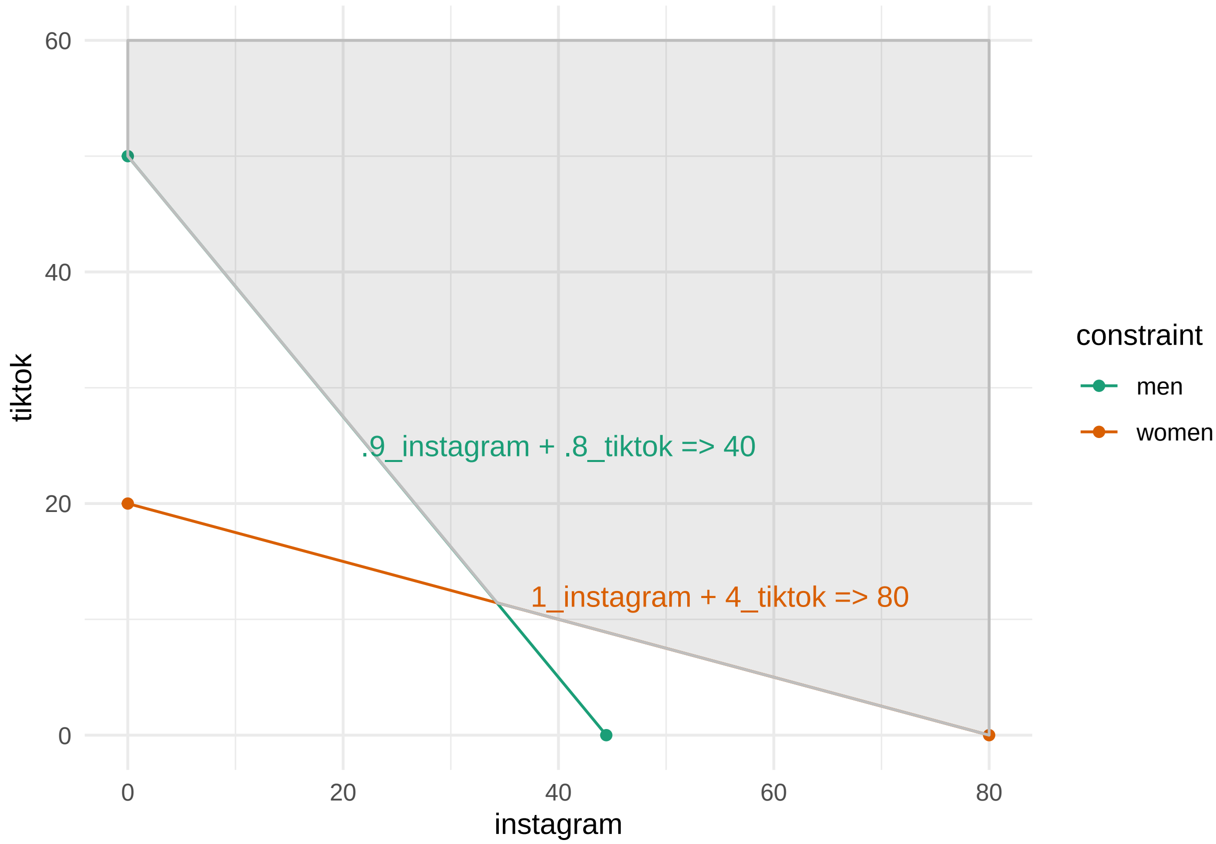
“Plot them”, and Jun went back to writing:



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

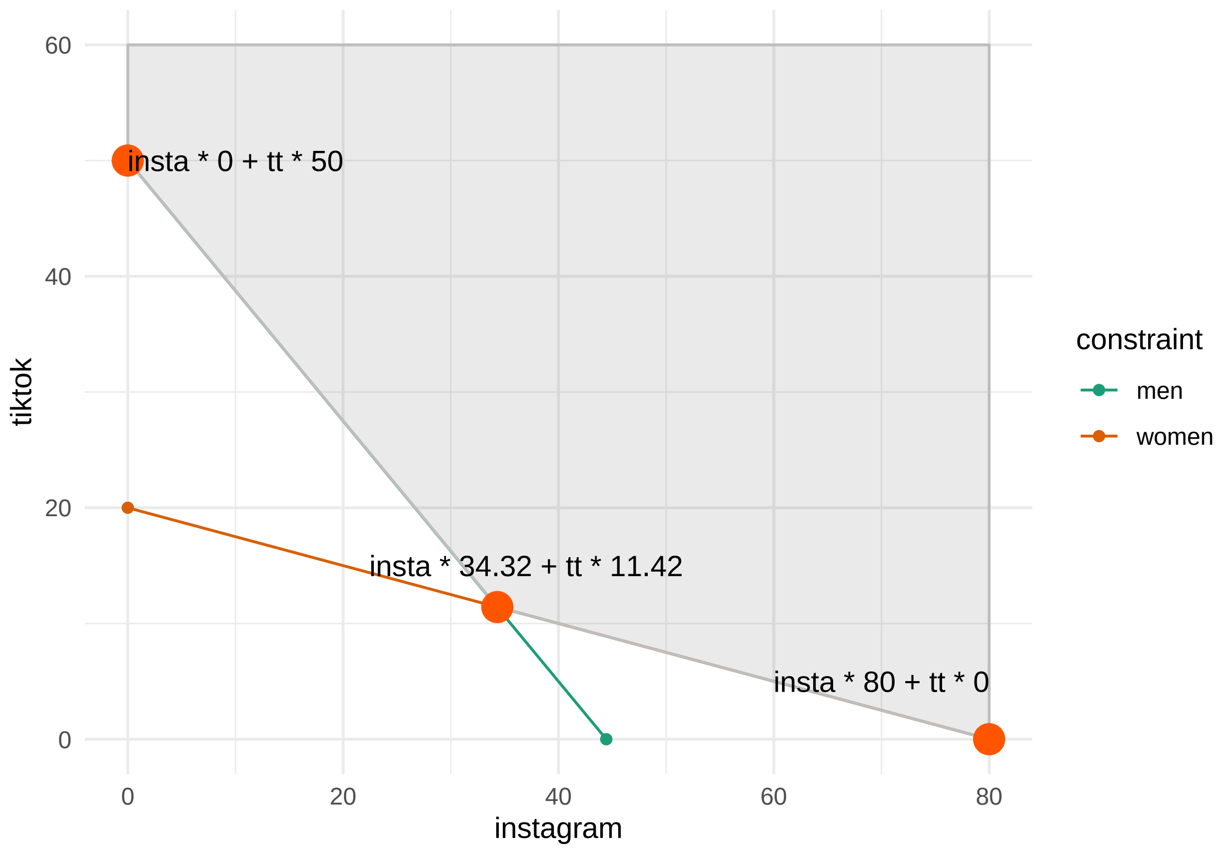
"Umm. . . it's a little fuzzy", admitted a puzzled Ali.

"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

Integer Programming

4.1 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)

# Remember...the c vector is the top part of the whole problem:
# 30candy + 40gummies + 80bag

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

# The b vector is the margins of the problem:
# What comes on the right hand side of the
# equality sign.

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

# These are the directions -- hopefully not
# much of a mystery.

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

# The a matrix comprises all of the rows of
# the constraint matrix (just not the margins
# or the directions).

# 1candy + 1gummies - 10bag
# 4candy + 3gummies - 20bag
# 1candy + 0gummies - 2bag
# 1candy + 1gummies + 0bag

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))

# All together, we have:
# 30candy + 40gummies + 80bag
# 1candy + 1gummies - 10bag <= 500
# 4candy + 3gummies - 20bag <= 200
# 1candy + 0gummies - 2bag <= 100
# 1candy + 1gummies + 0bag >= 1000

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.

4.2 ClassOverflow

Let's spend some time exploring some package functionality:

```r
library(lpSolve)

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

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

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

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

```
# When working within ROI, we string together the entire constraint
# matrix (A matrix, directions, and the b vector) using the L_constraint
# function

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

# After we string those together, we can throw them OP;
# this just creates the model, but doesn't actually
# solve the problem.
# The big difference below is in the types column;
# this changes it from continuous ("C") to
# integer ("I").

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.

4.3 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:

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

```r
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))

# Remember that the rep function below only serves to replicate whatever
# you put there. So this:
# rep("<=", 12)
# Is the same as:
# c("<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=", "<=")
# One is clearly an easier thing to code.
```

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

# The length function returns how many items are in the vector.

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, v4.47
7 rows, 12 columns, 24 non-zeros
      0: obj =  0.000000000e+000  infeas = 3.000e+003 (0)
*     6: obj =  1.049000000e+005  infeas = 0.000e+000 (0)
*    10: obj =  8.400000000e+004  infeas = 0.000e+000 (0)
OPTIMAL SOLUTION FOUND
GLPK Integer Optimizer, v4.47
7 rows, 12 columns, 24 non-zeros
12 integer variables, none of which are binary
Integer optimization begins...
+    10: mip =     not found yet >=              -inf        (1; 0)
+    10: >>>>>  8.400000000e+004 >=  8.400000000e+004   0.0% (1; 0)
+    10: mip =  8.400000000e+004 >=     tree is empty   0.0% (0; 1)
INTEGER OPTIMAL SOLUTION FOUND
<!SOLVER MSG> ----
```

```r
result$objval
```

```
[1] 84000
```

```r
transportation_solution <- solution(result)

# Just setting names to make life easier.

names(transportation_solution) <- c("g1_p1", "g1_p2", "g1_p3", "g1_p4",
                                    "g2_p1", "g2_p2", "g2_p3", "g2_p4",
                                    "g3_p1", "g3_p2", "g3_p3", "g3_p4")
```

“That’s a nice solution!”, Ali remarked and then emailed Castel the results:

> Hey Castel,
>
> From greenhouse 1, send 850 pounds to processing 1 and 350 pounds to processing 2.
>
> From greenhouse 2, send 250 pounds to processing 1 and 750 pounds to processing 4.
>
> From greenhouse 3, send 50 pounds to processing 50 and 750 to processing 3.
>
> All those moves will cost 84000 dollars.
>
> Let me know if I can help you with anything else,
>
> Ali

4.4 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:

```r
# Nothing tricky below -- all we are doing is creating a data frame.
# The data frame contains 3 columns: item, space, and value.

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))

# Below is just a little bit different from what we have seen:
# instead of specifying a whole vector on the RHS, we just
# have a single number. We are really only rocking with
# a single constraint.

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

# Going to set those variables to be integers.

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

solved_model <- ROI_solve(model)

# The line below looks weird, but we are literally saying
# to take our solution and then set the names of that
# data frame to the item names from our special_items
# data frame. The |> is R's native pipe operator.
# The only reason we are doing this is to make the solution
# easier to see (i.e., which itmes should we select).
```

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

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

solved_model <- ROI_solve(model)

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

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

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

```
[1] 4550
```

What should Ali do?

Chapter 5

Simulation

Ali was hearing a lot of people talk about “what if” scenarios and frankly, the requests for new work was coming in too fast. After dealing with marketing and retail problems, they would keep coming back to ask more questions. “What if we would purchase this many items?”, Boa and Blaise both asked. Even Castel was asking for information when the greenhouse situations would change. It seemed like Ali really needed to turn all of the code into functions… but a thought started chewing away at Ali brain.

“What if all of those things are just variables from different distributions”.

“If I know the past, I can figure out the distribution, and spin up some tables”.

The final straw was this email from Chaman, the head of grow operations:

> Good afternoon Ali,
> We have an interesting problem in our grow operations right now. Our plants are very successful, but a new strain isn't really behaving how we would expect, given the expected probability of full maturation. Nearly 99.9% of these plants live and the maturation cycle runs about 65 days. We started with a 1000 plants on our last run, but we only ended up with 750 plants at the end of the growing cycle. We also had some weird pump failures; we'd think it would only happen about 1/100 times. Realistically, how many times would something like this happen? Better said, is 750 the best we can get with 1000 plants? Thanks for the help,
> Chaman

Ali knew that two other analysts might be able to offer some insight: Shashi knew everything statistical and Alex was the simulation wizard.

Starting at the beginning is always important, so Ali swung by Shashi's cubicle.

5.1 Distributions

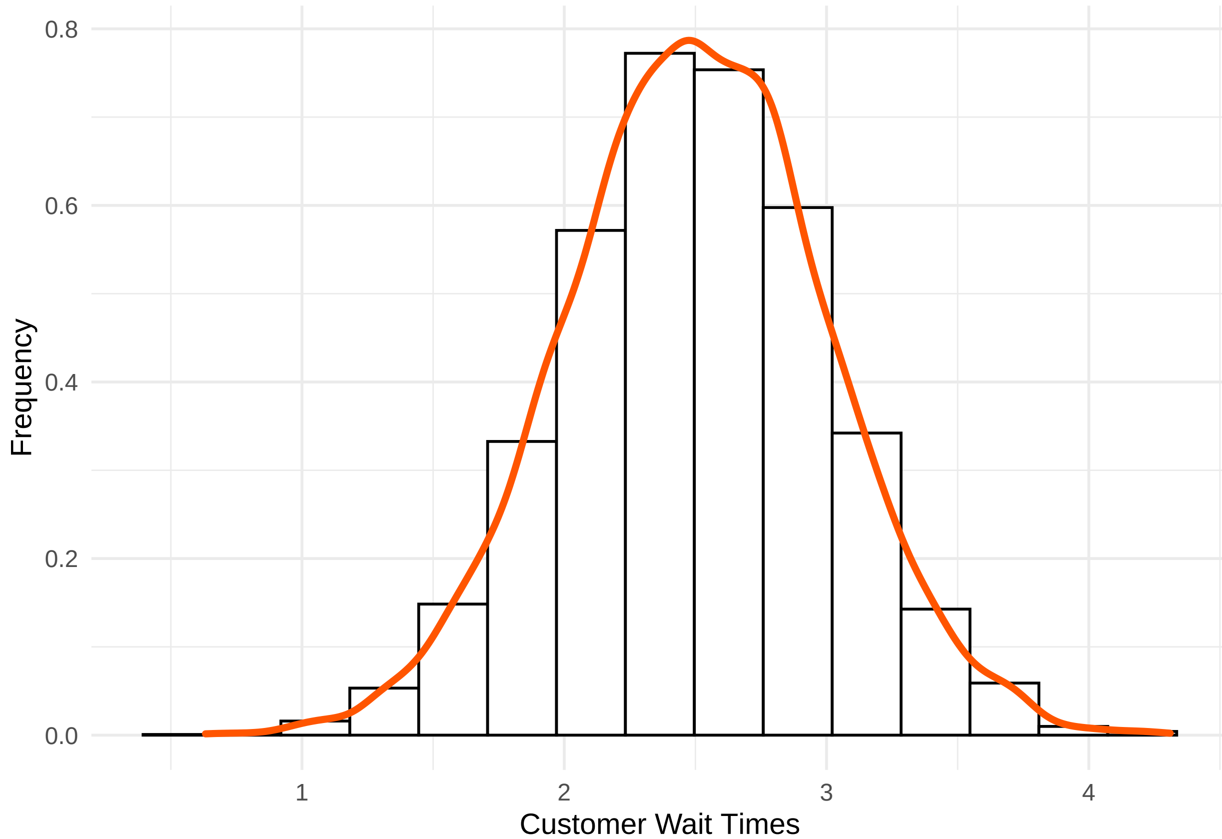
“You were right to come to me first”, Shashi said, “I'll fill you in on a few distributions that might be helpful to you.”

“Distributions drive every single part of simulation – every event that you can ever imagine comes from some type of distribution.”, Shashi said.

5.1.1 Normal Distribution

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

“It is likely the most common and you'll find that a lot of processes are normally distributed.”



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

"Let's start by creating a normal distribution."

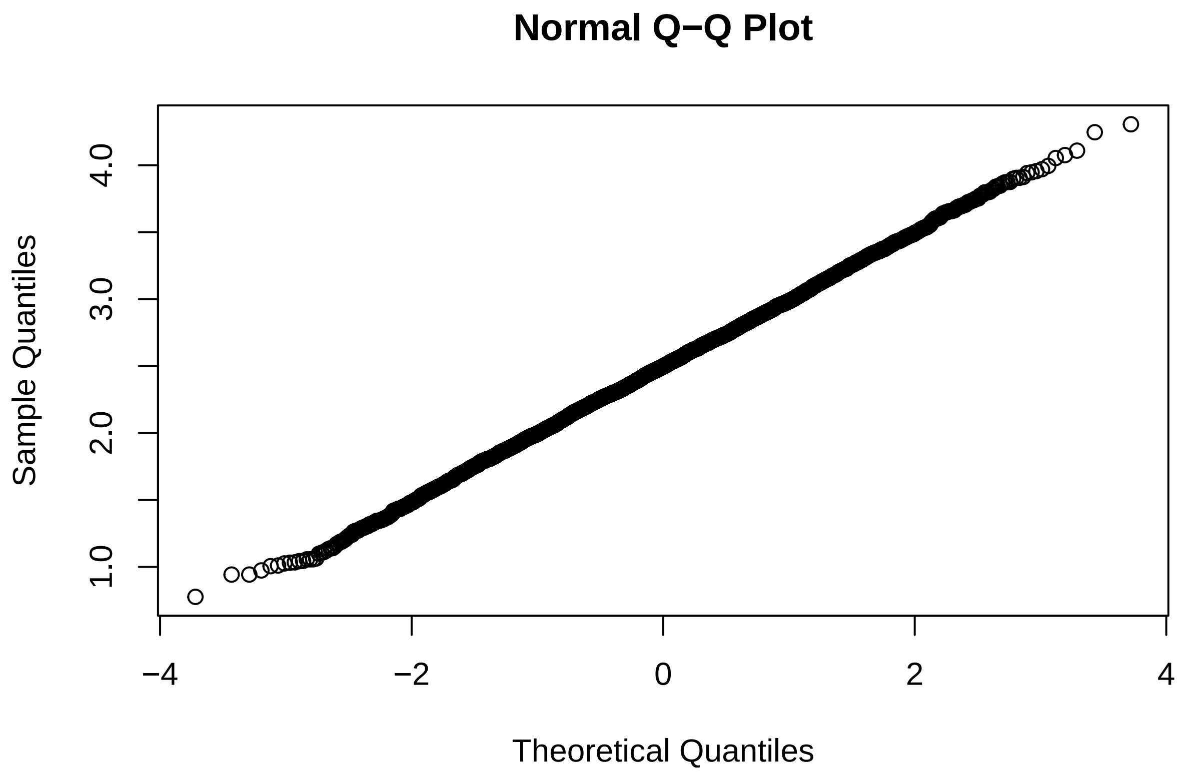
```
normal_variable <- rnorm(n = 5000, mean = 2.5, sd = .5)
```

"And an exponential distribution. . . I'll tell you more about that in a few minutes."

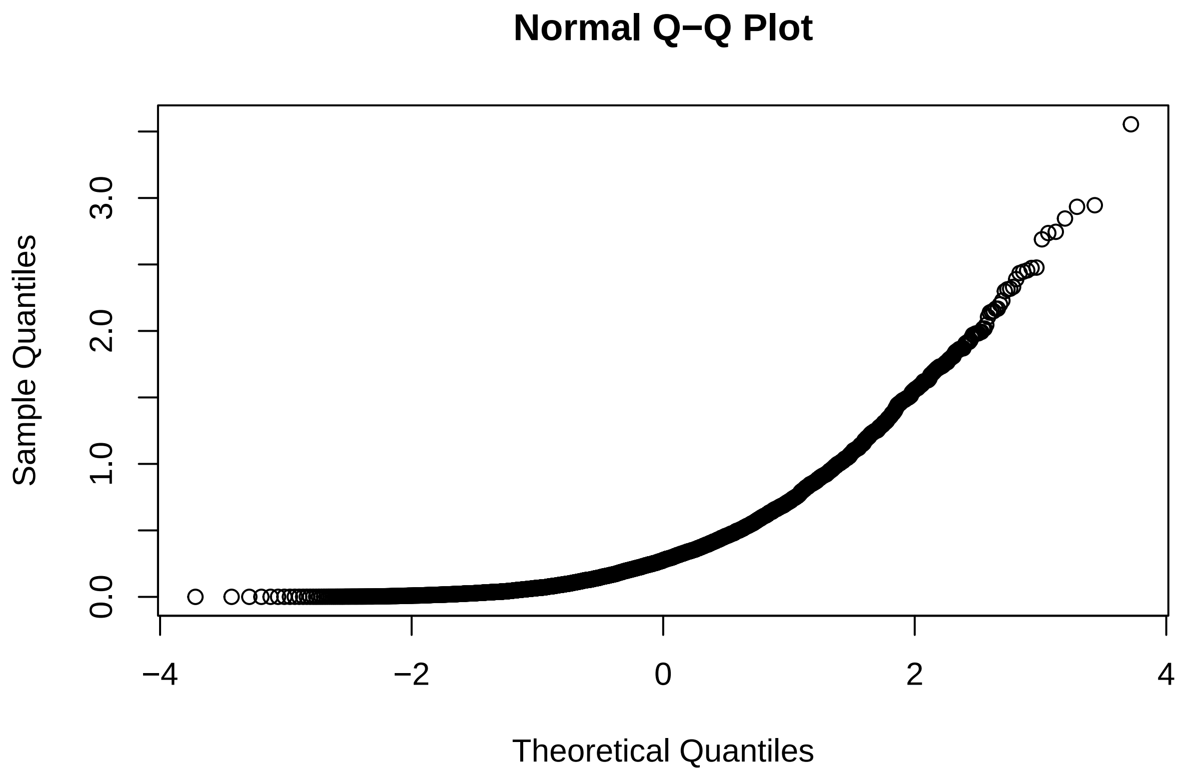
```
exponential_variable <- rexp(n = 5000, rate = 2.5)
```

"We can check both with a qq plot for normality."

```
qqnorm(normal_variable)
```




```
qqnorm(exponential_variable)
```



"The normal distribution will follow the diagonal perfectly, but the exponential looks like it curves down".

"If you ever want to test for the normal distribution, you can just use the Shapiro test."

```
shapiro.test(normal_variable)
```

```

    Shapiro-Wilk normality test

data:  normal_variable
W = 0.99973, p-value = 0.7966
```

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

```

    Shapiro-Wilk normality test

data:  exponential_variable
W = 0.81126, p-value < 2.2e-16
```

"Think about it like this", Shashi said, "We are trying to test if our observed distribution is significantly different than a normal distribution."

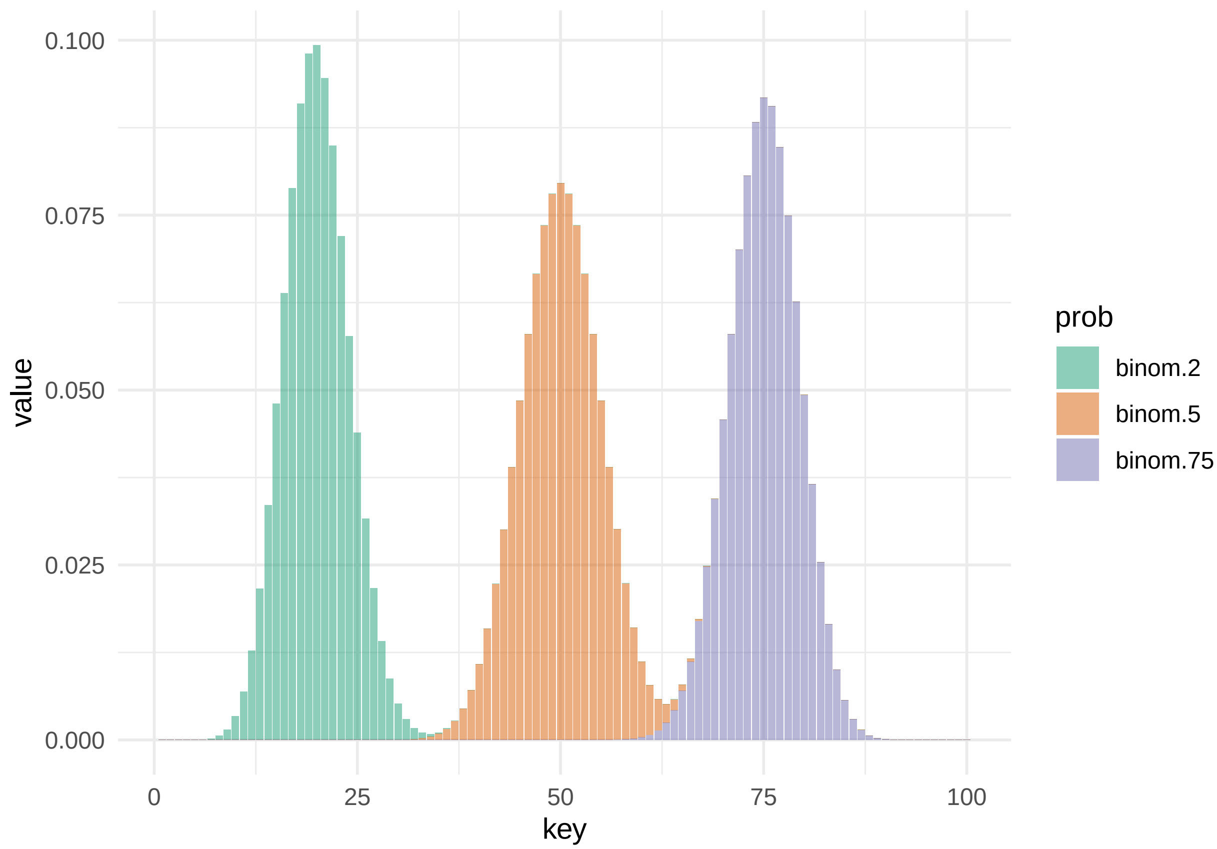
"We would want to see a p-value above .05 to suggest that we might be dealing with a normally-distributed variable."

"Clearly, the exponential distribution is significantly different."

"That is awesome!", Ali said.

5.1.2 Binomial Distribution

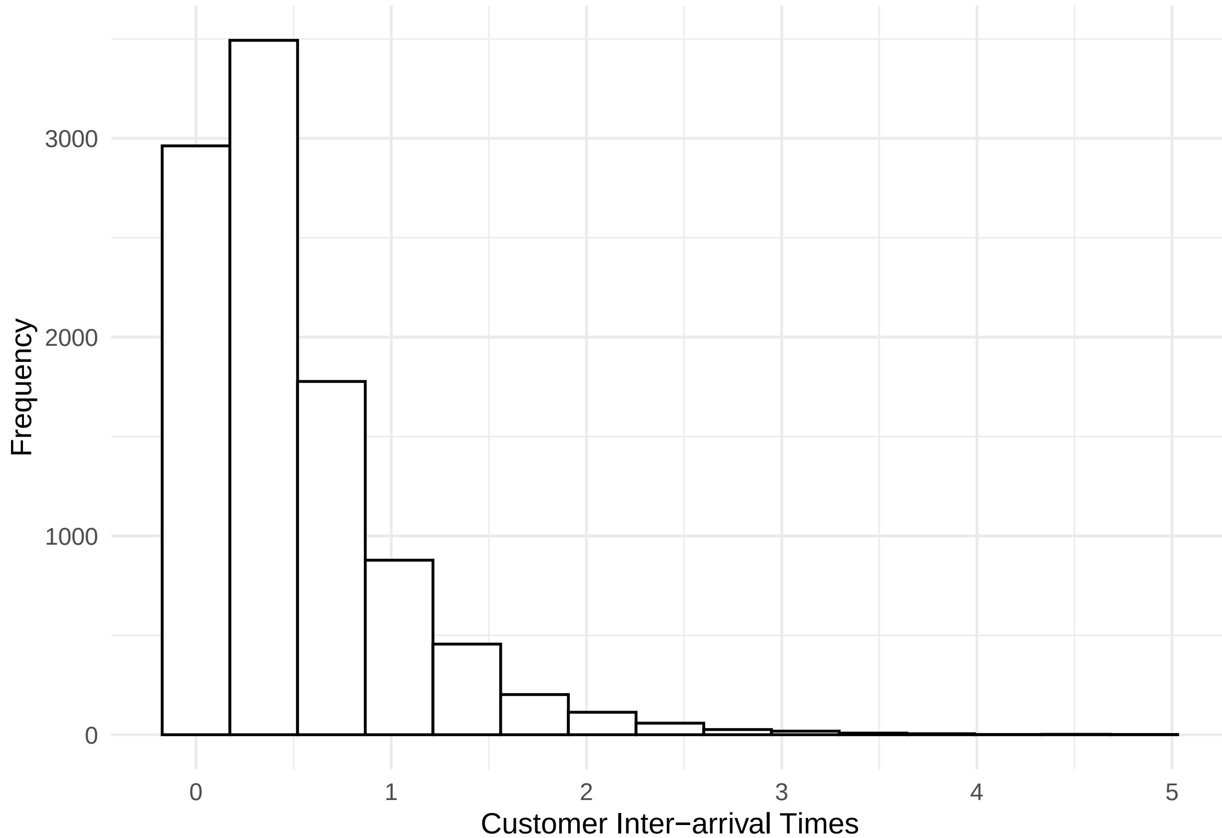
“Anything that has two outcomes – dead/alive, failure/success, missed/made – can be modeled with a binomial distribution.”



5.1.3 Exponential Distribution

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

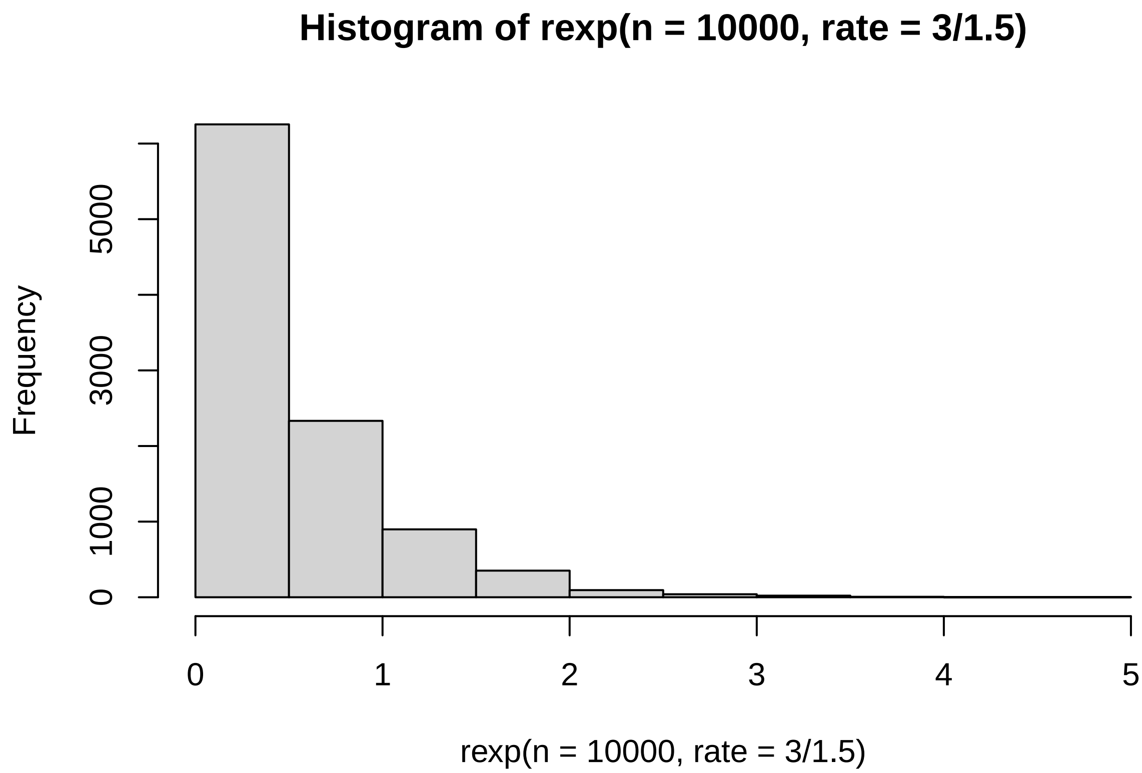
“Just about any arrival process can be approximated by an exponential distribution.”



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

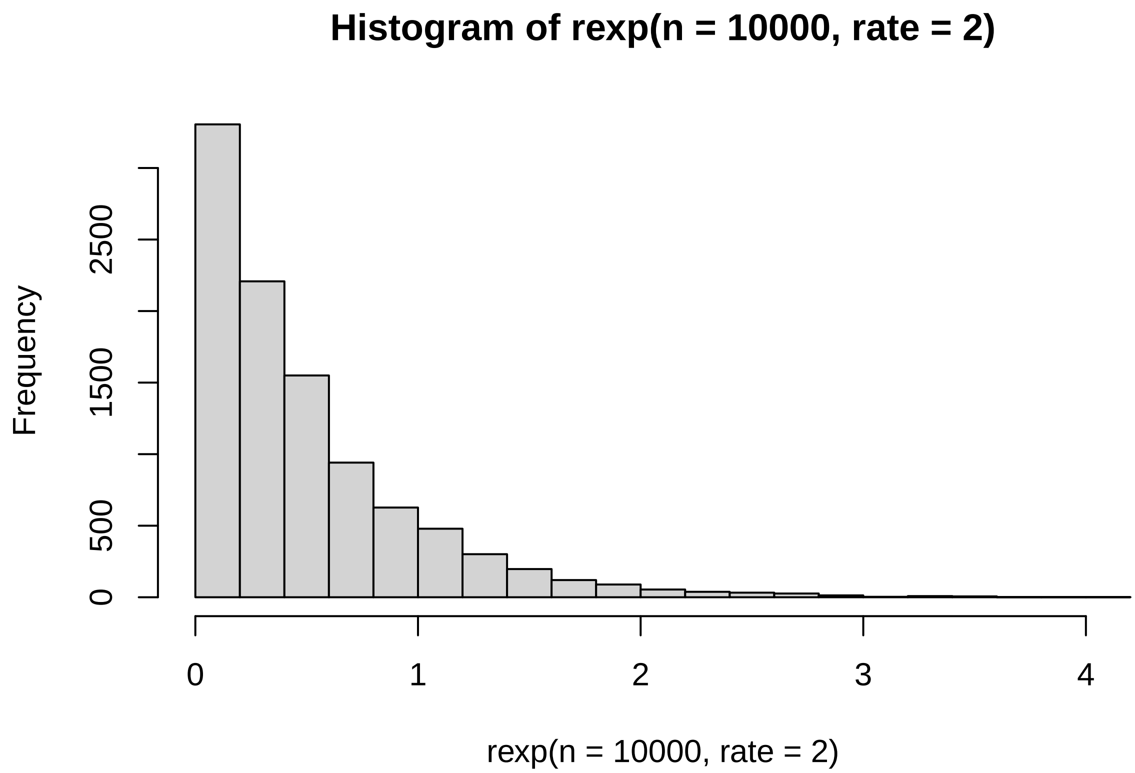
"If 3 people enter the store every minute and a half, than it would be a rate of 3/1.5".

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



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

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

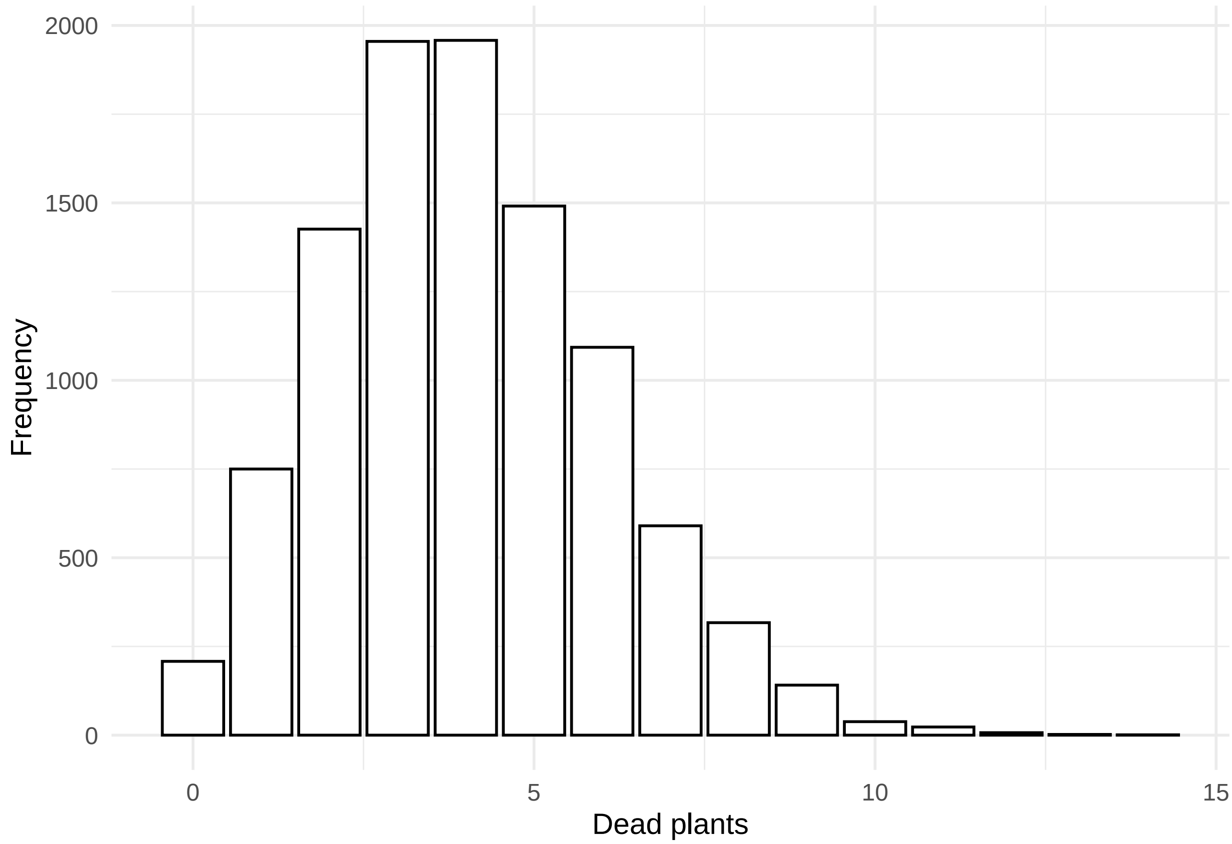


"They are really the same thing."

5.1.4 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."

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



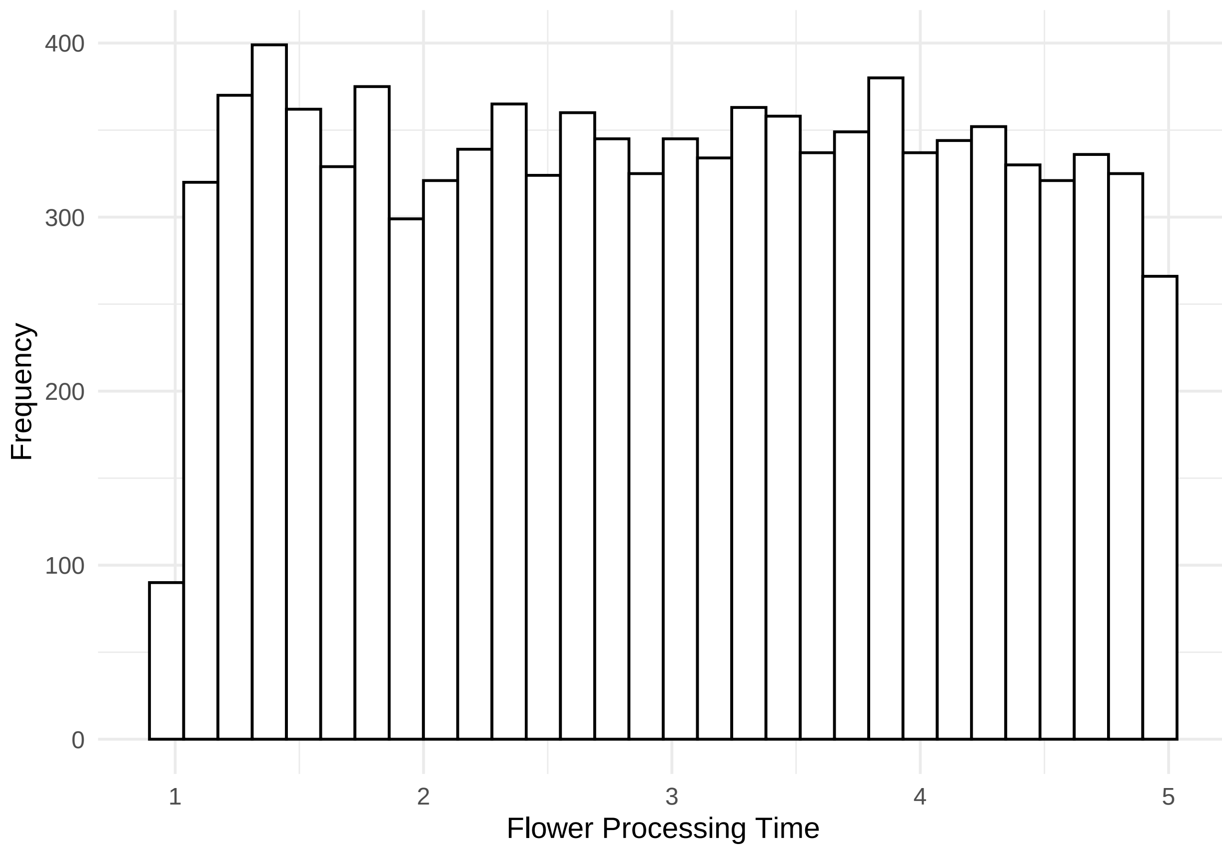
"Interestingly, the Poisson distribution has a relationship with the Exponential distribution: Poisson deals with the number of occurrences and the Exponential deals with the time between occurrences."

"Whoa... I've got a lot to learn here.", Ali mumbled.

"All with time!", Alex reassured.

5.1.5 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."



"Sound cool?", Shashi asked.

"For sure!", Ali replied, and went off to find Alex.

5.2 An Important Distinction

As soon as Ali started talking to Alex, a tangent started. "First and foremost, I want to set you straight on something: simulation is not what-if analysis."

"What's the different?", Ali asked.

"The what-if analysis isn't really guided by distributions, but more along the lines of low, medium, and high values."

"I think I need an example.", Ali said.

"I thought you'd never ask!", Alex exclaimed.

"Let's look at a really simple process: the number of people who come into a retail store."

"Let's say that every person who buys something in the store, shops for 20 minutes on average, with a standard deviation of 2.5 minutes."

"We might just be interested to know how much total time those people are in the store."

```
what_if_times <- c(low_value = 50,
                   mid_value = 100,
                   high_value = 150)

what_if_sums <- sapply(what_if_times, function(x) {
```

```
  sum(rnorm(x, mean = 20, sd = 2.5))
})

what_if_sums
```

```
 low_value  mid_value high_value
  1025.933   2000.163   3005.593
```

```
mean(what_if_sums)
```

```
[1] 2010.563
```

"That seems simple enough.", remarked Ali. "Those answers really seem pretty obvious."

Alex nodded. "They definitely do, but we have a potential problem here."

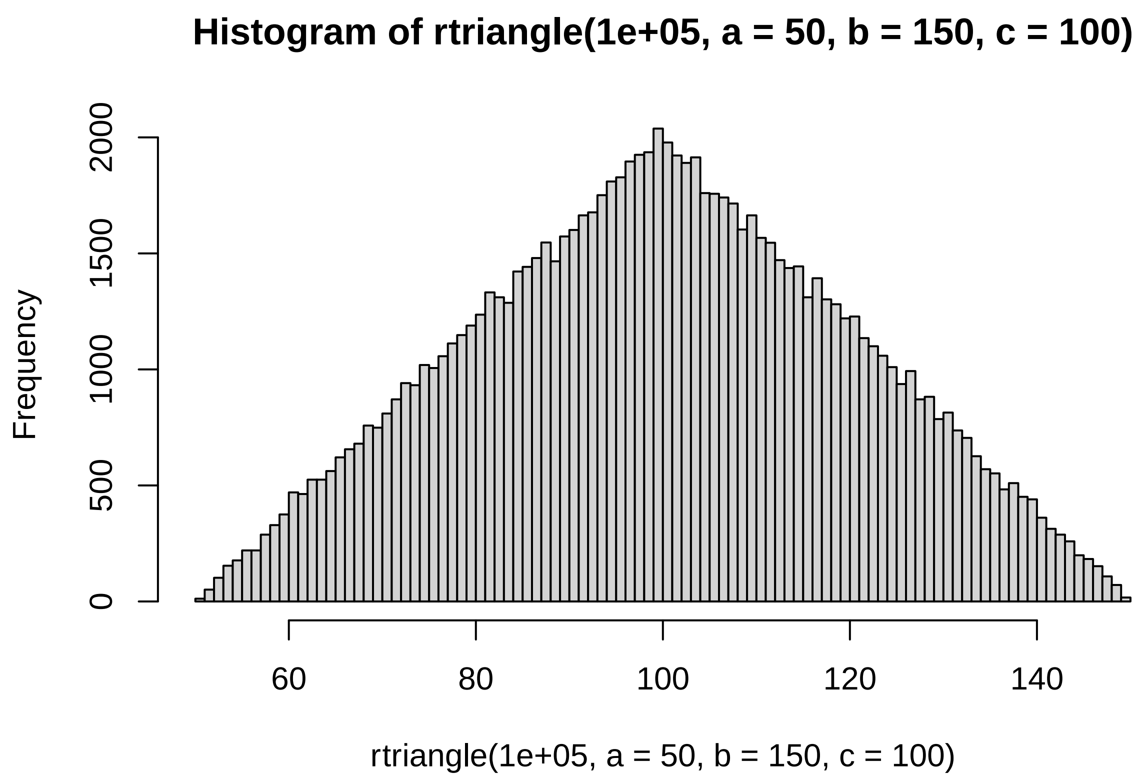
Ali had absolutely zero idea where Alex was going.

"Let me give you some more info; we are giving all of those an equal probability of occurring." Alex asked, "Does that seem reasonable?"

"Probably not.", Ali admitted.

"The most common number of people that come into the score is 100, but the highs and lows rarely happen."

"Check this distribution out:"

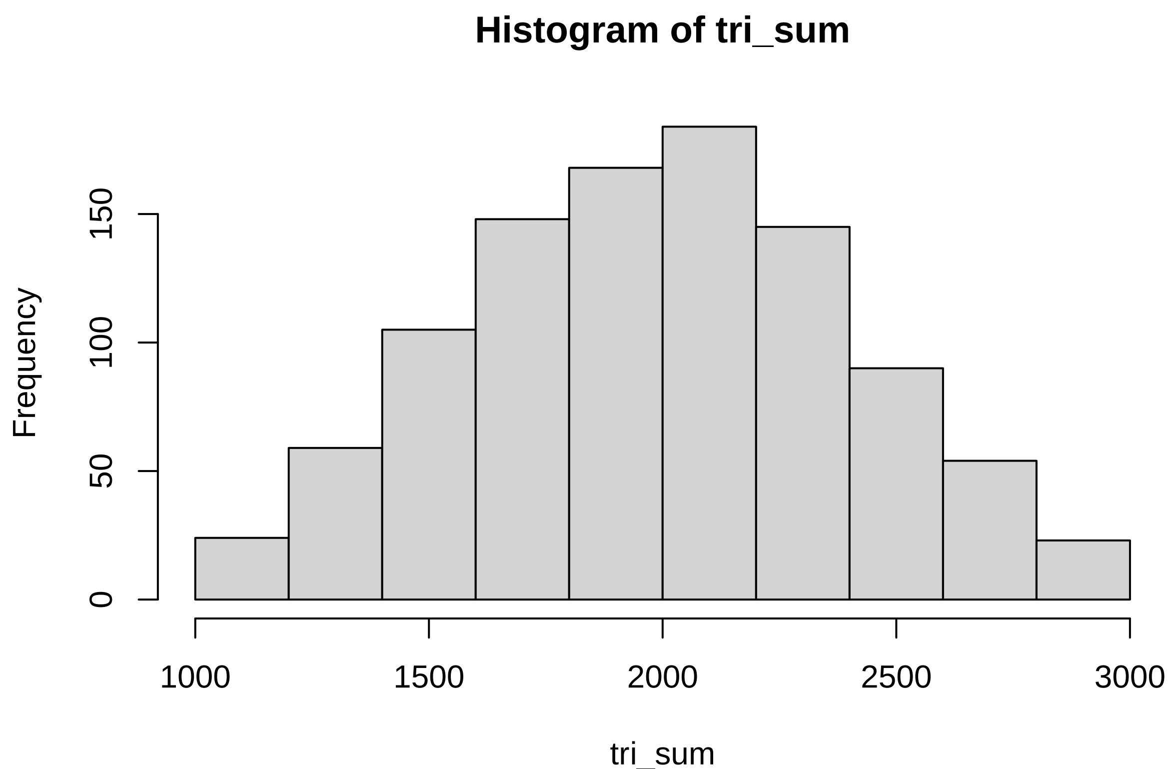


"In the triangular distribution, we have a few parameters: the lower limit, the upper limit, and the mode."

"That looks wild!", Ali smiled.

"Now, let's see how we can incorporate this into a really small simulation."

```r
library(triangle)

triangle_draws <- rtriangle(n = 1000, a = 50, b = 150, c = 100)

tri_sum <- sapply(triangle_draws, function(x) {
  sum(rnorm(x, mean = 20, sd = 2.5))
})

hist(tri_sum)
```



"Is the average wildly different?", Alex asked.

"Nope!", Ali said, "but that is a much better representation of what we would expect!"

"Using distributions is always the way to go.", Alex said.

"Shashi said something like that.", Ali replied.

"Things become even more interesting as we start to incorporate more distributions into our problems.", Alex smiled as he began typing.

"I'm sure Shashi showed you the exponential distribution", Alex continued, "so let me show you what we can do with it."

"Let's say that our stores get raw product shipments at a rate of 1 every 3 days, on average."

```r
interarrival_time <- rexp(100, rate = 1/3)
```

"Now, let's say that the average shipment follows a normal distribution, with a mean of 2 pounds and a standard deviation of .5 pounds."

```
shipment_pounds <- rnorm(100, mean = 2, sd = .5)
```

"I think I'm following you.", Ali nodded along.

Alex continued, "Let's just program something small."

"I'd be curious to know how many days it would take to get 100 shipments and how much total weight we would get."

"We will start by creating day 1 in the simulation:"

```
day <- 1
```

"And our starting pounds."

```
total_pounds_delivered <- 0
```

"Next we can specify how many shipments we would like to test this over."

```
n_shipments <- 100
```

"Finally, we need to use a `for` loop to iterate over those 100 shipments."

```
for(i in 1:n_shipments) {
  # For every shipment, we want to generate 1 draw from
  # the normal distribution.
  shipment_pounds <- rnorm(1, mean = 2, sd = .5)

  # We also want to generate the rate at which
  # the shipments will arrive.
  interarrival_time <- rexp(1, rate = 1/3)

  # Now we can add the shipped pounds to our total_pounds_delivered.
  # This will update total_pounds_delivered at every iteration.
  total_pounds_delivered <- total_pounds_delivered + shipment_pounds

  # And the same idea is used for keeping track of the days.
  day <- day + interarrival_time
}
```

"Let's see those results!"

```
day
```

```
[1] 317.1205
```

```
total_pounds_delivered
```

```
[1] 198.6589
```

"So it would take us about 317 days to get 100 shipments and we would get about 199.", Alex noted.

"That is absolutely awesome!", Ali gasped. "It is like we are programming the real world."

"That's right.", Alex nodded. "You are only limited by what you can program."

"I've got one more thing to show you…how to repeat the process."

"All we need to do is to take our complete code:"

```
 day <- 1

  total_pounds_delivered <- 0
```

```
  n_shipments <- 100

  for(i in 1:n_shipments) {

    shipment_pounds <- rnorm(1, mean = 2, sd = .5)

    interarrival_time <- rexp(1, rate = 1/3)

    total_pounds_delivered <- total_pounds_delivered + shipment_pounds

    day <- day + interarrival_time
  }
```

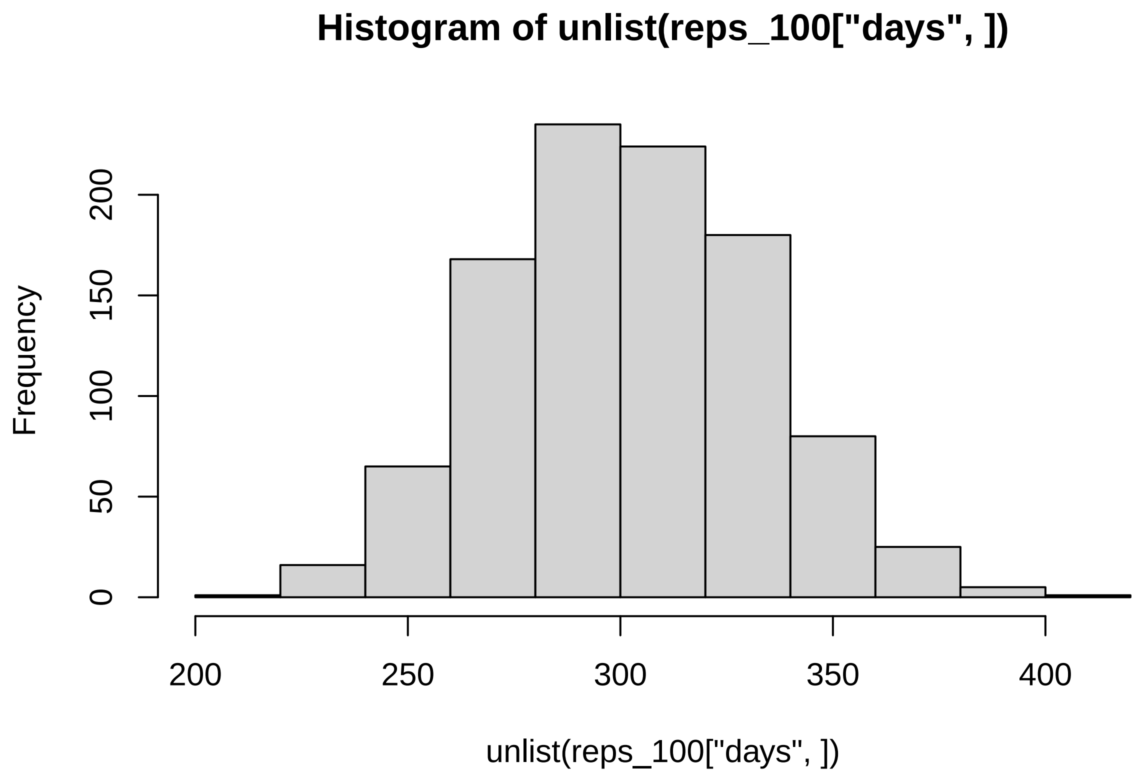
"And then throw that into the replicate function!"

```
reps_100 <- replicate(n = 1000, expr = {
  day <- 1

  total_pounds_delivered <- 0

  n_shipments <- 100

  for(i in 1:n_shipments) {

    shipment_pounds <- rnorm(1, mean = 2, sd = .5)

    interarrival_time <- rexp(1, rate = 1/3)

    total_pounds_delivered <- total_pounds_delivered + shipment_pounds

    day <- day + interarrival_time
  }

  # This is the only different part; I am just taking my results
  # and throwing them into a data frame. That way, I am always
  # returning a data frame!
  results <- data.frame(days = day,
                        total_pounds = total_pounds_delivered)

  results
}, simplify = TRUE)
```

"After that runs, `reps_100` will be a weird-looking object, but we can still use it for some histograms."

"We need to refer to the row names of `days` and `total_pounds` to extract the rows out and then `unlist` those rows into a vector:

```
hist(unlist(reps_100["days",]))
```




```
hist(unlist(reps_100["total_pounds",]))
```

Ali was in awe: "That is coolest thing I've seen."

5.3 ClassOverflow

Let's spend some time together working through the email from earlier.

Chapter 6

Nonlinear Optimization

While you have clearly taken a linear line to get to this point, you should probably go backwards across your linear line.

Chapter 7

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.

```
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."

7.1 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."

7.2 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."

λ = arrival rate

μ = 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."

7.3 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.

7.4 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
}
")
```

7.4.1 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 simmer::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.

```
simmer::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)
  })

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

```
simmer environment: dispensary | now: 120 | next: 120.118797062361
{ Monitor: in memory }
{ Resource: id_check | monitored: TRUE | server status: 0(1) | queue status: 0(8) }
{ Resource: bud_tender | monitored: TRUE | server status: 2(2) | queue status: 0(10) }
{ Resource: payment | monitored: TRUE | server status: 1(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 replication
1 Customer0   0.000000  2.172501      2.172501     TRUE           1
2 Customer1   1.153201  4.598131      3.444930     TRUE           1
3 Customer2   7.843659 10.256743      2.413084     TRUE           1
4 Customer3   8.425594 11.787949      3.362355     TRUE           1
5 Customer4   9.348817 12.425742      2.726908     TRUE           1
6 Customer5   9.876889 13.353479      2.048718     TRUE           1
```

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

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

```
[1] 57
```

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] 0.3230758
```

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)
    })

  simmer::run(dispensary, until = 120)

  result <- get_mon_arrivals(bank)

}, simplify = FALSE)
```

We can extend this idea into something a bit more complex:

```
purrr::map_df(1:100, ~{
  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)
    })

  simmer::run(dispensary, until = 120)

  result <- get_mon_arrivals(dispensary)

  result$run <- .x

  result
})
```

```
         name start_time   end_time activity_time finished replication run
1   Customer0   0.000000   2.008038      2.008038     TRUE           1   1
2   Customer1   4.651134   6.567857      1.916723     TRUE           1   1
3   Customer2   6.743405   9.920926      3.177520     TRUE           1   1
4   Customer3  10.059104  12.771622      2.712518     TRUE           1   1
5   Customer4  11.308526  14.278406      2.969880     TRUE           1   1
6   Customer5  11.981833  14.872611      2.432374     TRUE           1   1
7   Customer6  13.284047  16.680390      2.838124     TRUE           1   1
8   Customer7  15.197964  17.747368      2.549404     TRUE           1   1
9   Customer8  15.466407  18.096354      1.815371     TRUE           1   1
10  Customer9  15.652584  20.612290      3.388249     TRUE           1   1
11 Customer10  16.217795  21.046046      3.420150     TRUE           1   1
12 Customer11  18.588826  23.641303      3.302752     TRUE           1   1
13 Customer12  19.025585  23.941834      3.258955     TRUE           1   1
14 Customer13  19.350829  25.475262      2.233096     TRUE           1   1
15 Customer14  22.126344  26.216148      2.848605     TRUE           1   1
16 Customer15  23.398688  28.517364      3.391125     TRUE           1   1
17 Customer16  24.579553  29.068037      3.345005     TRUE           1   1
18 Customer17  27.040605  31.129249      3.058353     TRUE           1   1
19 Customer18  30.462466  32.751059      2.288593     TRUE           1   1
20 Customer19  34.693811  36.844043      2.150232     TRUE           1   1
21 Customer21  36.230937  39.080874      2.457354     TRUE           1   1
22 Customer20  35.834450  39.422976      3.588527     TRUE           1   1
23 Customer22  36.727095  40.859212      2.142332     TRUE           1   1
24 Customer23  37.008501  42.352183      3.418354     TRUE           1   1
25 Customer24  41.505838  43.366218      1.860380     TRUE           1   1
26 Customer25  42.078456  45.498058      3.419602     TRUE           1   1
27 Customer26  43.921780  47.071150      3.149370     TRUE           1   1
28 Customer27  44.204840  48.419763      3.343995     TRUE           1   1
29 Customer28  45.738639  48.975973      2.371365     TRUE           1   1
30 Customer29  46.278853  50.612638      2.562160     TRUE           1   1
31 Customer30  47.475571  51.099193      2.592091     TRUE           1   1
32 Customer31  49.455542  53.038677      2.814155     TRUE           1   1
33 Customer32  50.498969  53.835438      3.070833     TRUE           1   1
34 Customer33  60.327944  63.482543      3.154599     TRUE           1   1
35 Customer34  60.637957  63.579601      2.941645     TRUE           1   1
```

36	Customer36	61.704568	65.985994	2.851850	TRUE	1	1
37	Customer35	60.826323	66.581454	3.459717	TRUE	1	1
38	Customer37	64.202483	68.012305	2.509854	TRUE	1	1
39	Customer39	64.428260	69.428866	1.961994	TRUE	1	1
40	Customer38	64.341537	69.693206	3.483615	TRUE	1	1
41	Customer41	65.359181	71.304182	2.074383	TRUE	1	1
42	Customer40	64.595591	72.332210	3.333265	TRUE	1	1
43	Customer42	66.452806	73.637055	2.798810	TRUE	1	1
44	Customer43	66.702380	75.560119	3.719854	TRUE	1	1
45	Customer44	68.884953	75.934992	2.656369	TRUE	1	1
46	Customer46	74.722318	77.643779	2.167840	TRUE	1	1
47	Customer45	70.399696	77.690578	2.494751	TRUE	1	1
48	Customer48	76.533152	80.029068	2.875843	TRUE	1	1
49	Customer47	74.834607	80.164243	2.904363	TRUE	1	1
50	Customer50	77.424765	81.995549	2.372680	TRUE	1	1
51	Customer49	77.032489	83.259296	3.515830	TRUE	1	1
52	Customer51	79.354964	84.174353	2.569455	TRUE	1	1
53	Customer52	80.838093	85.416957	2.603704	TRUE	1	1
54	Customer54	84.146275	86.501615	1.516386	TRUE	1	1
55	Customer53	82.806658	86.528025	2.781471	TRUE	1	1
56	Customer55	85.521158	89.077599	2.866203	TRUE	1	1
57	Customer56	86.966727	89.175195	2.208467	TRUE	1	1
58	Customer57	88.224667	90.881362	2.096376	TRUE	1	1
59	Customer58	88.575516	91.331100	2.488491	TRUE	1	1
60	Customer60	91.701074	94.204036	2.502962	TRUE	1	1
61	Customer59	91.047475	94.557685	3.510211	TRUE	1	1
62	Customer62	92.475747	96.329544	2.237493	TRUE	1	1
63	Customer61	91.820451	97.590459	3.711783	TRUE	1	1
64	Customer63	92.809743	97.967439	2.132292	TRUE	1	1
65	Customer64	93.222429	99.889205	2.763457	TRUE	1	1
66	Customer65	93.512514	101.361356	3.720009	TRUE	1	1
67	Customer66	97.385861	102.581808	3.114449	TRUE	1	1
68	Customer67	97.662696	103.864676	2.922711	TRUE	1	1
69	Customer68	98.583535	104.223682	2.085331	TRUE	1	1
70	Customer70	99.810354	106.247230	2.358870	TRUE	1	1
71	Customer69	99.087992	106.451030	3.005744	TRUE	1	1
72	Customer71	99.850202	108.632612	2.858831	TRUE	1	1
73	Customer72	102.547239	109.570910	3.569231	TRUE	1	1
74	Customer73	105.470457	110.521803	2.163010	TRUE	1	1
75	Customer74	106.148087	111.492280	2.347971	TRUE	1	1
76	Customer75	110.840249	113.373850	2.533601	TRUE	1	1
77	Customer76	114.115075	116.377014	2.261939	TRUE	1	1
78	Customer77	114.652320	116.992588	2.340269	TRUE	1	1
79	Customer78	116.494435	118.636567	2.142133	TRUE	1	1
80	Customer0	0.000000	3.512173	3.512173	TRUE	1	2
81	Customer1	2.539255	5.021651	2.482396	TRUE	1	2
82	Customer2	3.208745	6.227910	3.019165	TRUE	1	2
83	Customer3	3.390915	7.050477	2.558789	TRUE	1	2
84	Customer4	3.422179	8.741242	3.029269	TRUE	1	2
85	Customer5	3.504727	9.849055	3.317677	TRUE	1	2
86	Customer6	4.564979	11.074624	2.706693	TRUE	1	2
87	Customer7	5.393038	12.502126	3.066615	TRUE	1	2
88	Customer8	6.890971	14.048064	3.294201	TRUE	1	2
89	Customer9	6.903770	14.538564	2.458944	TRUE	1	2

```
90  Customer11   9.971274  16.545743      2.399827     TRUE           1   2
91  Customer10   8.171401  16.617804      3.136202     TRUE           1   2
92  Customer13  14.766684  19.315892      3.120081     TRUE           1   2
93  Customer12  11.219620  19.527343      3.358654     TRUE           1   2
94  Customer15  15.767123  21.736445      2.620181     TRUE           1   2
95  Customer14  14.773180  22.617512      3.731901     TRUE           1   2
96  Customer16  18.834913  23.685021      2.268470     TRUE           1   2
97  Customer17  20.673439  25.004628      2.866642     TRUE           1   2
98  Customer18  21.550515  26.058081      2.950335     TRUE           1   2
99  Customer19  21.688129  27.504027      2.801940     TRUE           1   2
100 Customer20  26.982570  29.388128      2.405558     TRUE           1   2
101 Customer21  28.998222  31.200763      2.202541     TRUE           1   2
102 Customer23  30.596082  33.135208      2.270354     TRUE           1   2
103 Customer22  30.521650  33.393115      2.871466     TRUE           1   2
104 Customer24  31.108744  35.092458      2.246538     TRUE           1   2
105 Customer25  31.622256  35.231329      2.333972     TRUE           1   2
106 Customer26  32.271363  37.375312      2.628736     TRUE           1   2
107 Customer27  34.684920  38.252850      3.483582     TRUE           1   2
108 Customer28  36.543993  39.163010      2.213237     TRUE           1   2
109 Customer29  37.789094  40.532056      2.742962     TRUE           1   2
110 Customer31  41.227140  43.964283      2.737143     TRUE           1   2
111 Customer30  40.699818  44.062114      3.362296     TRUE           1   2
112 Customer32  42.287373  46.568926      3.065249     TRUE           1   2
113 Customer33  45.441288  47.455652      2.014364     TRUE           1   2
114 Customer34  47.833290  49.929863      2.096573     TRUE           1   2
115 Customer35  49.652266  52.178479      2.526213     TRUE           1   2
116 Customer36  56.787661  59.425273      2.637612     TRUE           1   2
117 Customer37  57.951590  61.035149      3.083559     TRUE           1   2
118 Customer38  59.486050  62.339598      2.853547     TRUE           1   2
119 Customer39  59.959647  63.371930      2.774294     TRUE           1   2
120 Customer40  62.446550  65.312475      2.865925     TRUE           1   2
121 Customer41  63.424222  66.574737      3.150515     TRUE           1   2
122 Customer42  63.642393  66.592153      1.755643     TRUE           1   2
123 Customer44  65.884140  68.288452      2.085575     TRUE           1   2
124 Customer43  65.160341  70.211197      4.067353     TRUE           1   2
125 Customer45  71.319882  73.980625      2.660744     TRUE           1   2
126 Customer46  72.981094  76.539708      3.558613     TRUE           1   2
127 Customer47  74.983119  77.384854      2.401734     TRUE           1   2
128 Customer48  75.105315  78.425609      2.284587     TRUE           1   2
129 Customer49  75.270361  79.487116      2.463941     TRUE           1   2
130 Customer50  76.674275  80.797363      2.737065     TRUE           1   2
131 Customer51  78.708423  83.003575      3.861842     TRUE           1   2
132 Customer52  78.991951  83.126154      2.881901     TRUE           1   2
133 Customer53  78.999873  85.639960      3.054289     TRUE           1   2
134 Customer54  80.420695  86.060447      3.374577     TRUE           1   2
135 Customer55  81.045504  87.532338      2.219415     TRUE           1   2
136 Customer56  81.998299  87.871123      2.182853     TRUE           1   2
137 Customer57  83.946920  90.381967      3.209284     TRUE           1   2
138 Customer58  84.510684  90.962062      3.384100     TRUE           1   2
139 Customer60  88.933308  93.023604      2.534604     TRUE           1   2
140 Customer59  84.608039  93.568265      3.617610     TRUE           1   2
141 Customer62  91.399932  95.363358      2.240013     TRUE           1   2
142 Customer61  90.947648  96.407665      3.831358     TRUE           1   2
 [ reached 'max' / getOption("max.print") -- omitted 7454 rows ]
```

Next, we can see what things might look like if we change parts of the process:

```r
purrr::map2_df(.x = 1:100, .y = runif(100, min = 1, max = 2), ~{
    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/.y), -1)
    })

  simmer::run(dispensary, until = 120)

  result <- get_mon_arrivals(dispensary)

  result$run <- .x

  result$arrival <- .y

  result
})
```

```
          name  start_time   end_time activity_time finished replication run  arrival
1     Customer0  0.00000000  3.202141     3.2021410     TRUE           1   1 1.228431
2     Customer1  0.08534413  3.271919     3.0916456     TRUE           1   1 1.228431
3     Customer2  0.09068545  5.993304     3.2112875     TRUE           1   1 1.228431
4     Customer3  0.70206098  6.164998     3.2712333     TRUE           1   1 1.228431
5     Customer4  3.99568112  8.143570     2.5927947     TRUE           1   1 1.228431
6     Customer5  4.10713292  8.402484     2.6137418     TRUE           1   1 1.228431
7     Customer6  4.60112901 10.905308     3.1622137     TRUE           1   1 1.228431
8     Customer7  4.62679987 10.970833     2.8433541     TRUE           1   1 1.228431
9     Customer9 10.22199669 12.586337     1.9514108     TRUE           1   1 1.228431
10    Customer8  7.49818900 13.855648     3.3007256     TRUE           1   1 1.228431
11   Customer10 10.50394339 15.350535     3.1826765     TRUE           1   1 1.228431
12   Customer11 11.16459705 15.878927     2.4381701     TRUE           1   1 1.228431
13   Customer12 13.70320714 18.128175     3.1422143     TRUE           1   1 1.228431
14   Customer13 14.47836859 18.614357     3.0956096     TRUE           1   1 1.228431
15   Customer14 16.24739548 20.873874     3.1754996     TRUE           1   1 1.228431
16   Customer15 18.29817804 21.689645     3.3914666     TRUE           1   1 1.228431
17   Customer16 19.42913455 23.151528     2.7761561     TRUE           1   1 1.228431
18   Customer17 19.54133095 24.644475     3.2506856     TRUE           1   1 1.228431
19   Customer18 21.12055355 25.991373     3.2992913     TRUE           1   1 1.228431
20   Customer19 21.99101128 27.575959     3.3249850     TRUE           1   1 1.228431
```

21	Customer20	22.98436861	28.869866	3.3345882	TRUE	1	1	1.228431
22	Customer22	27.27020275	30.108088	1.6926195	TRUE	1	1	1.228431
23	Customer21	24.44513696	30.398535	3.3281876	TRUE	1	1	1.228431
24	Customer23	27.99630497	33.353919	3.7114688	TRUE	1	1	1.228431
25	Customer24	30.99574371	33.960744	2.9650007	TRUE	1	1	1.228431
26	Customer25	31.39140095	35.597209	2.6618430	TRUE	1	1	1.228431
27	Customer26	32.29573888	35.991993	2.2610171	TRUE	1	1	1.228431
28	Customer27	32.35869716	38.753514	3.5083296	TRUE	1	1	1.228431
29	Customer28	33.69149304	38.948177	3.3900488	TRUE	1	1	1.228431
30	Customer29	34.21185404	40.654303	2.3941489	TRUE	1	1	1.228431
31	Customer30	35.13854969	40.984985	2.5099010	TRUE	1	1	1.228431
32	Customer31	35.99836545	43.338219	3.0324015	TRUE	1	1	1.228431
33	Customer32	37.06185215	44.031798	3.3828332	TRUE	1	1	1.228431
34	Customer34	37.76771673	45.729038	2.1968058	TRUE	1	1	1.228431
35	Customer33	37.25992213	45.813040	2.9096354	TRUE	1	1	1.228431
36	Customer35	38.11534518	47.502256	2.0755019	TRUE	1	1	1.228431
37	Customer36	40.70645296	47.876232	2.4122722	TRUE	1	1	1.228431
38	Customer37	42.62097694	49.245845	2.1238746	TRUE	1	1	1.228431
39	Customer38	43.33520581	50.259514	2.8379526	TRUE	1	1	1.228431
40	Customer40	45.36238367	51.300300	1.4916368	TRUE	1	1	1.228431
41	Customer39	43.44566391	51.641224	2.7028301	TRUE	1	1	1.228431
42	Customer42	48.61591678	52.842665	1.5758444	TRUE	1	1	1.228431
43	Customer41	47.22571248	54.241893	3.3797856	TRUE	1	1	1.228431
44	Customer43	49.42253087	55.345757	2.9473110	TRUE	1	1	1.228431
45	Customer45	49.88276039	57.282087	2.4137272	TRUE	1	1	1.228431
46	Customer44	49.67193448	57.306118	3.4339668	TRUE	1	1	1.228431
47	Customer47	51.99312816	58.974408	2.0573671	TRUE	1	1	1.228431
48	Customer46	51.54802212	59.937538	3.1285603	TRUE	1	1	1.228431
49	Customer48	53.36656521	62.212654	3.7367406	TRUE	1	1	1.228431
50	Customer49	53.72864963	62.254744	2.8300135	TRUE	1	1	1.228431
51	Customer51	54.49951138	64.697870	2.8036612	TRUE	1	1	1.228431
52	Customer50	54.46307502	64.887915	3.0818383	TRUE	1	1	1.228431
53	Customer52	54.71500566	66.621660	2.3921754	TRUE	1	1	1.228431
54	Customer53	57.45354044	67.599795	3.1041139	TRUE	1	1	1.228431
55	Customer54	59.40245001	68.933193	2.6114249	TRUE	1	1	1.228431
56	Customer55	59.59085442	70.074294	2.8450543	TRUE	1	1	1.228431
57	Customer57	60.96458292	72.190491	2.5762528	TRUE	1	1	1.228431
58	Customer56	60.65675096	72.357510	3.7465359	TRUE	1	1	1.228431
59	Customer59	65.42315224	74.594704	2.6284690	TRUE	1	1	1.228431
60	Customer58	64.05478501	74.673840	2.8381136	TRUE	1	1	1.228431
61	Customer72	75.99374357	76.450623	0.3332246	FALSE	1	1	1.228431
62	Customer60	65.58605549	76.854565	2.7444819	TRUE	1	1	1.228431
63	Customer61	66.01904641	77.604370	3.4251506	TRUE	1	1	1.228431
64	Customer75	78.22544539	78.624403	0.3021656	FALSE	1	1	1.228431
65	Customer76	78.42183831	78.785478	0.1610744	FALSE	1	1	1.228431
66	Customer62	67.46383097	79.153534	2.7055317	TRUE	1	1	1.228431
67	Customer78	78.85164670	79.300206	0.1752801	FALSE	1	1	1.228431
68	Customer79	79.51301351	79.754191	0.2411778	FALSE	1	1	1.228431
69	Customer63	68.31373763	80.080360	2.7826806	TRUE	1	1	1.228431
70	Customer81	80.06370587	80.259356	0.1956498	FALSE	1	1	1.228431
71	Customer64	70.31581618	81.110840	2.4620808	TRUE	1	1	1.228431
72	Customer65	70.56653225	81.712771	2.0224449	TRUE	1	1	1.228431
73	Customer84	82.94552003	83.221735	0.2762147	FALSE	1	1	1.228431
74	Customer66	73.82496853	83.401280	2.8090803	TRUE	1	1	1.228431

```
75   Customer67  74.33011788  83.654418     2.3611967     TRUE           1    1 1.228431
76   Customer68  75.26219396  85.870160     2.8492953     TRUE           1    1 1.228431
77   Customer69  75.36758014  86.487292     3.1309009     TRUE           1    1 1.228431
78   Customer70  75.39500355  87.360143     1.7802593     TRUE           1    1 1.228431
79   Customer71  75.67796011  89.223915     3.1052411     TRUE           1    1 1.228431
80   Customer73  76.78757803  90.246394     3.1383712     TRUE           1    1 1.228431
81   Customer74  78.09470646  91.807907     3.0522403     TRUE           1    1 1.228431
82   Customer77  78.43946208  93.194226     3.5156519     TRUE           1    1 1.228431
83   Customer80  79.89343807  93.602239     2.1700391     TRUE           1    1 1.228431
84   Customer83  82.24740125  95.273490     2.1038312     TRUE           1    1 1.228431
85   Customer82  81.40792286  95.800888     2.9738470     TRUE           1    1 1.228431
86   Customer85  83.91058837  96.998431     2.1106161     TRUE           1    1 1.228431
87   Customer86  84.67359736  97.639644     2.3519510     TRUE           1    1 1.228431
88   Customer87  85.55781097  98.855296     2.3428368     TRUE           1    1 1.228431
89   Customer88  86.70821969  99.322769     2.0625376     TRUE           1    1 1.228431
90   Customer89  88.94657055 101.041230     2.5326628     TRUE           1    1 1.228431
91   Customer90  95.50944627 101.733218     2.7927692     TRUE           1    1 1.228431
92   Customer91  96.22260186 102.876911     2.2737282     TRUE           1    1 1.228431
93   Customer92  97.67205775 104.361808     2.9608997     TRUE           1    1 1.228431
94   Customer94 100.21592948 105.921530     1.8798234     TRUE           1    1 1.228431
95   Customer93 100.15120144 106.256499     3.8513762     TRUE           1    1 1.228431
96   Customer95 100.40781629 108.111639     2.5291172     TRUE           1    1 1.228431
97   Customer96 100.86317522 109.108468     3.3325494     TRUE           1    1 1.228431
98   Customer97 101.73424117 110.626072     2.9377398     TRUE           1    1 1.228431
99   Customer98 104.12763894 110.868071     2.2378369     TRUE           1    1 1.228431
100 Customer100 104.39718803 113.372341     2.7858914     TRUE           1    1 1.228431
101  Customer99 104.17616161 113.539465     3.3431634     TRUE           1    1 1.228431
102   Customer0   0.00000000   2.929397     2.9293970     TRUE           1    2 1.043467
103   Customer1   0.77129205   3.001764     2.2304717     TRUE           1    2 1.043467
104   Customer3   2.27217929   5.098978     2.4272789     TRUE           1    2 1.043467
105   Customer2   1.86886446   5.473980     2.8157589     TRUE           1    2 1.043467
106   Customer4   3.34927986   6.691788     1.9621159     TRUE           1    2 1.043467
107   Customer5   4.05103562   9.026998     3.9570398     TRUE           1    2 1.043467
108   Customer6   6.35522172   9.706738     3.3515168     TRUE           1    2 1.043467
109   Customer7   6.91568188  11.481514     2.9618420     TRUE           1    2 1.043467
110   Customer8   7.32679004  12.886730     3.5730658     TRUE           1    2 1.043467
111   Customer9   9.61432152  14.132507     2.9828489     TRUE           1    2 1.043467
112  Customer10  10.57967306  15.381365     2.9114441     TRUE           1    2 1.043467
113  Customer11  19.86695602  22.652759     2.7858034     TRUE           1    2 1.043467
114  Customer12  19.96535182  23.400245     3.1606854     TRUE           1    2 1.043467
115  Customer13  20.52705464  24.632917     2.4041289     TRUE           1    2 1.043467
116  Customer14  21.43503472  25.239705     2.2501958     TRUE           1    2 1.043467
117  Customer15  22.53677119  26.832171     2.7556468     TRUE           1    2 1.043467
118  Customer16  22.92607622  27.292770     2.4571060     TRUE           1    2 1.043467
119  Customer17  23.31272399  29.070349     2.6713440     TRUE           1    2 1.043467
120  Customer18  23.92244416  30.296620     3.5500147     TRUE           1    2 1.043467
121  Customer19  26.11371596  32.149055     3.3805567     TRUE           1    2 1.043467
122  Customer20  26.73781420  32.731128     2.9316002     TRUE           1    2 1.043467
123  Customer21  27.17368960  34.009660     2.3370648     TRUE           1    2 1.043467
124  Customer22  28.70492264  35.392190     3.0700286     TRUE           1    2 1.043467
125  Customer23  30.49089747  35.935403     2.3179698     TRUE           1    2 1.043467
 [ reached 'max' / getOption("max.print") -- omitted 7793 rows ]
```

And a function to show:

```
employee_change_mod <- purrr::map2_df(.x = 1:100, .y = sample(1:3, 100, TRUE, c(.1, .7, .2)), ~{
  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") |>
    branch(function() {sample(1:2, 1, prob = c(.8, .2))},
           # The "continue" indicates if the branches should continue through
           # the trajectory once the branch trajectory is completed.
           continue = c(TRUE, TRUE),
           trajectory() %>%
             seize("bud_tender") |>
             timeout(function() {rnorm(n = 1, mean = 2.4, sd = .5)}) |>
             release("bud_tender"),
           trajectory() |>
             seize("devices") |>
             timeout(function() {rnorm(1, 5, 1)}) |>
             release("devices")) |>
    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 = .y, queue_size = 10) |>
    add_resource("devices", capacity = 1, queue_size = 10) |>
    add_resource("payment", capacity = 2) |>
    add_generator("Customer", customer, function() {
      c(0, rexp(n = 100, rate = 1/.y), -1)
    })

  simmer::run(dispensary, until = 120)

  result <- get_mon_arrivals(dispensary)

  result$run <- .x

  result$employees <- .y

  result
})

sapply(split(employee_change_mod, employee_change_mod$employees), function(x) {
  finished <- x[x$finished == TRUE, ]
  nrow(finished) / nrow(x)
})
```

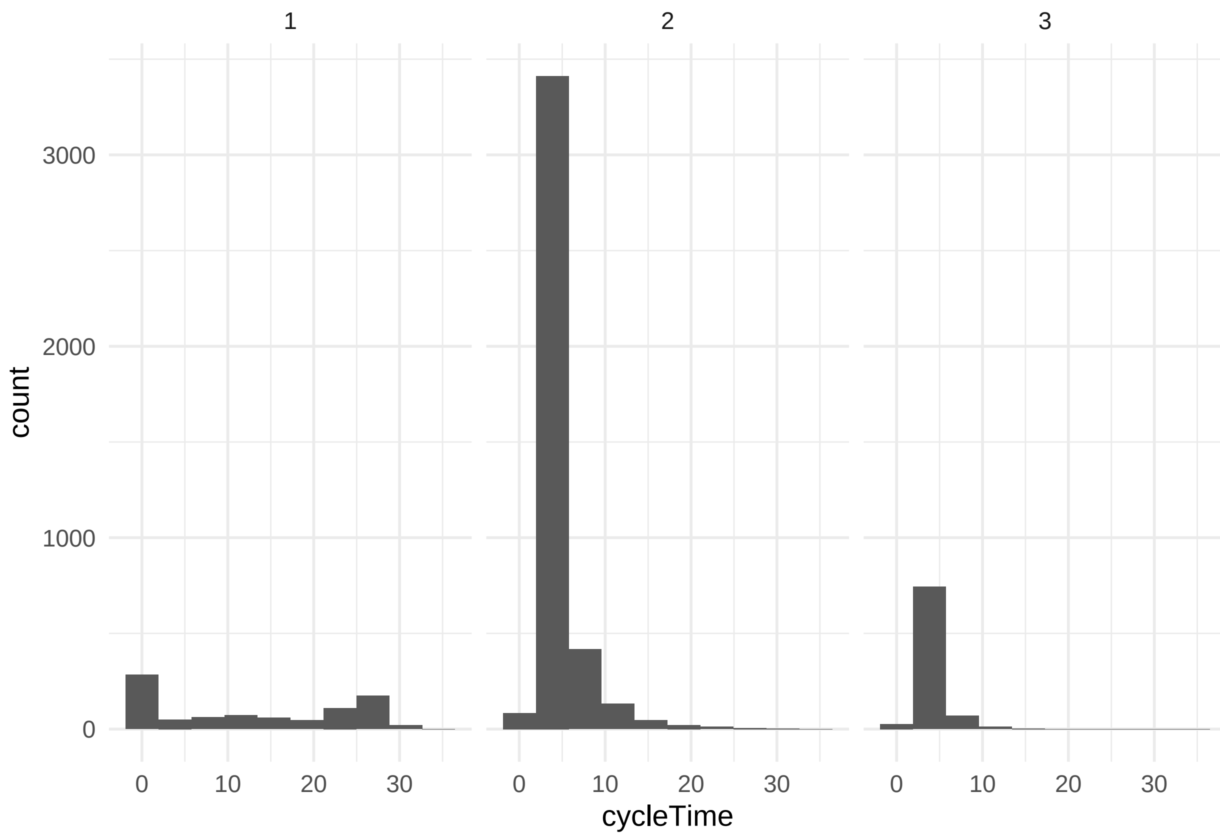
```
        1         2         3
0.6787741 1.0000000 1.0000000
```

```
employee_change_mod$cycleTime <- employee_change_mod$end_time - employee_change_mod$start_time

aggregate(employee_change_mod$cycleTime,
          by = list(employee_change_mod$employees),
```

```
          mean)
```

```
  Group.1         x
1       1 12.843056
2       2  4.232715
3       3  3.510384
```

```
library(ggplot2)

ggplot(employee_change_mod, aes(cycleTime)) +
  geom_histogram(bins = 10) +
  facet_wrap(vars(employees)) +
  theme_minimal()
```



7.5 Manufacturing

In manufacturing, one of the more important metrics we deal with is *throughput* (the total number of units produced).

Variability, while not something we can always measure/predict, can come from a few sources.

- Machine/worker processing times
- Output quality
- Demand
- Supply

Controlling variability can help to increase throughput.

7.5.1 An Example

- In a factory, we have five work stations arranged in a line (WS1 through WS5).
- Each work station is a machine with one operator.
  - WS1 (bandsaw): $\mu = 10; \sigma = 1$ (normal)
  - WS2 (contouring): $\mu = 5; \sigma = 2$ (normal)
  - WS3 (rough sanding): $min = 5; max = 15$ (uniform)
  - WS4 (finish sanding): $min = 10; max = 15$ (uniform)
  - WS5 (finish): $\mu = 10; \sigma = 2.5$ (normal)
- The work stations process one product that must move sequentially.
- Each work station has its own processing time.
- Product cannot be *stacked*
  - If WS3 has finished a unit, it cannot be passed onto WS4 if WS4 is still working on a product.
- We are looking at an 8 hour shift.

7.5.2 Improvements

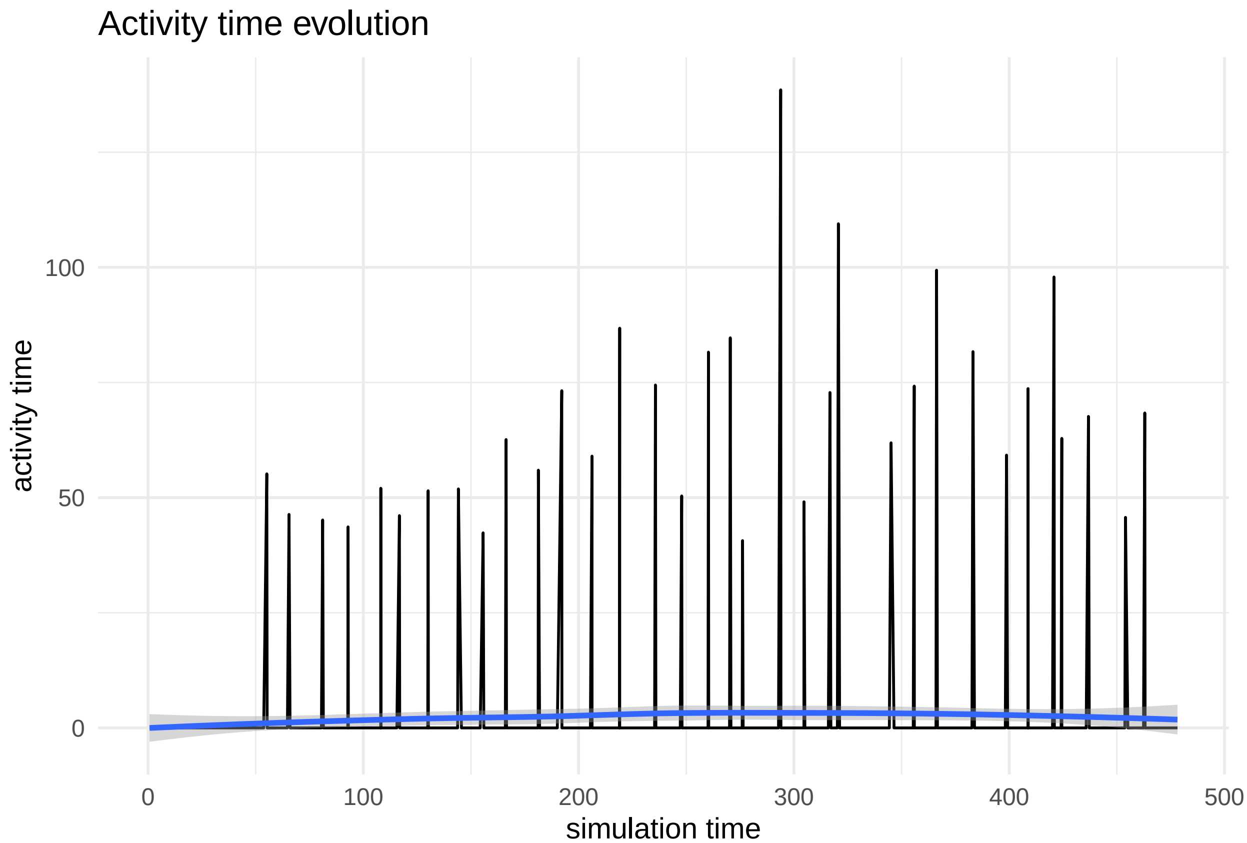
What can we do to improve our throughput?

```
library(simmer.plot)
make_parts <- trajectory("parts") %>%
  set_attribute("start_time", function() {now(machineShop)}) %>%
  seize("machine1") %>%
  timeout(function() {rnorm(n = 1, mean = 10, sd = 1)}) %>%
  release("machine1") %>%
  seize("machine2", continue = FALSE,
        reject = trajectory() %>%
          timeout(1) %>%
          rollback(amount = 2, times = Inf)) %>%
  timeout(function() {rnorm(n = 1, mean = 5, sd = 2)}) %>%
  release("machine2") %>%
  seize("machine3", continue = FALSE,
        reject = trajectory() %>%
          timeout(1) %>%
          rollback(amount = 2, times = Inf)) %>%
  timeout(function() {runif(n = 1, min = 5, max = 15)}) %>%
  release("machine3") %>%
  seize("machine4", continue = FALSE,
        reject = trajectory() %>%
          timeout(1) %>%
          rollback(amount = 2, times = Inf)) %>%
  timeout(function() {runif(n = 1, min = 10, max = 15)}) %>%
  release("machine4") %>%
  seize("machine5", continue = FALSE,
        reject = trajectory() %>%
          timeout(1) %>%
          rollback(amount = 2, times = Inf)) %>%
```

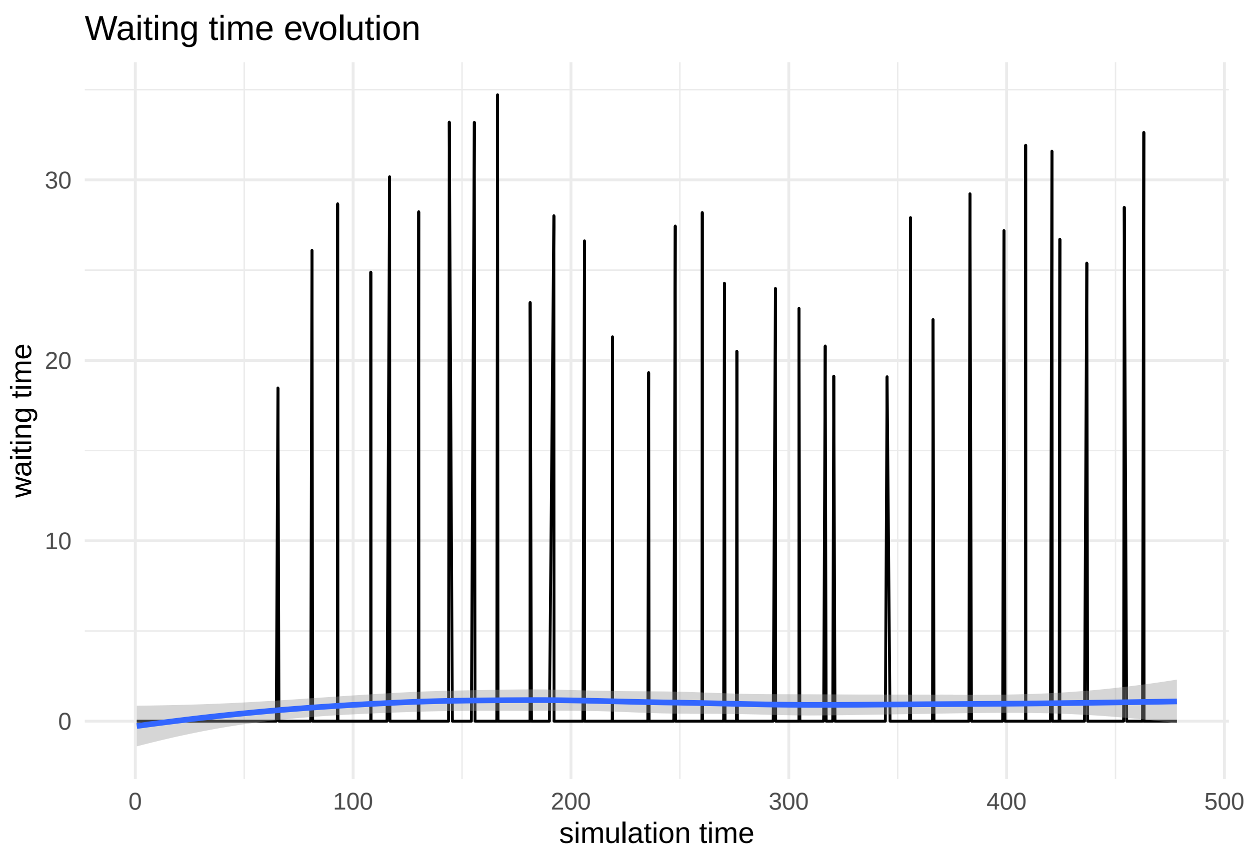
```r
  timeout(function() {rnorm(n = 1, mean = 10, sd = 2.5)}) %>%
  release("machine5")

machineShop <- simmer("machineShop") %>%
  add_resource("machine1", capacity = 1, queue_size = 1) %>%
  add_resource("machine2", capacity = 1, queue_size = 1) %>%
  add_resource("machine3", capacity = 1, queue_size = 1) %>%
  add_resource("machine4", capacity = 1, queue_size = 1) %>%
  add_resource("machine5", capacity = 1, queue_size = 1) %>%
  add_generator("part", make_parts, mon = 1, function() {c(0, rexp(1000, 1/.5), -1)})

simmer::run(machineShop, 480)
```

```
simmer environment: machineShop | now: 480 | next: 480.229766788663
{ Monitor: in memory }
{ Resource: machine1 | monitored: TRUE | server status: 1(1) | queue status: 1(1) }
{ Resource: machine2 | monitored: TRUE | server status: 1(1) | queue status: 0(1) }
{ Resource: machine3 | monitored: TRUE | server status: 0(1) | queue status: 0(1) }
{ Resource: machine4 | monitored: TRUE | server status: 1(1) | queue status: 1(1) }
{ Resource: machine5 | monitored: TRUE | server status: 1(1) | queue status: 1(1) }
{ Source: part | monitored: 1 | n_generated: 1001 }
```

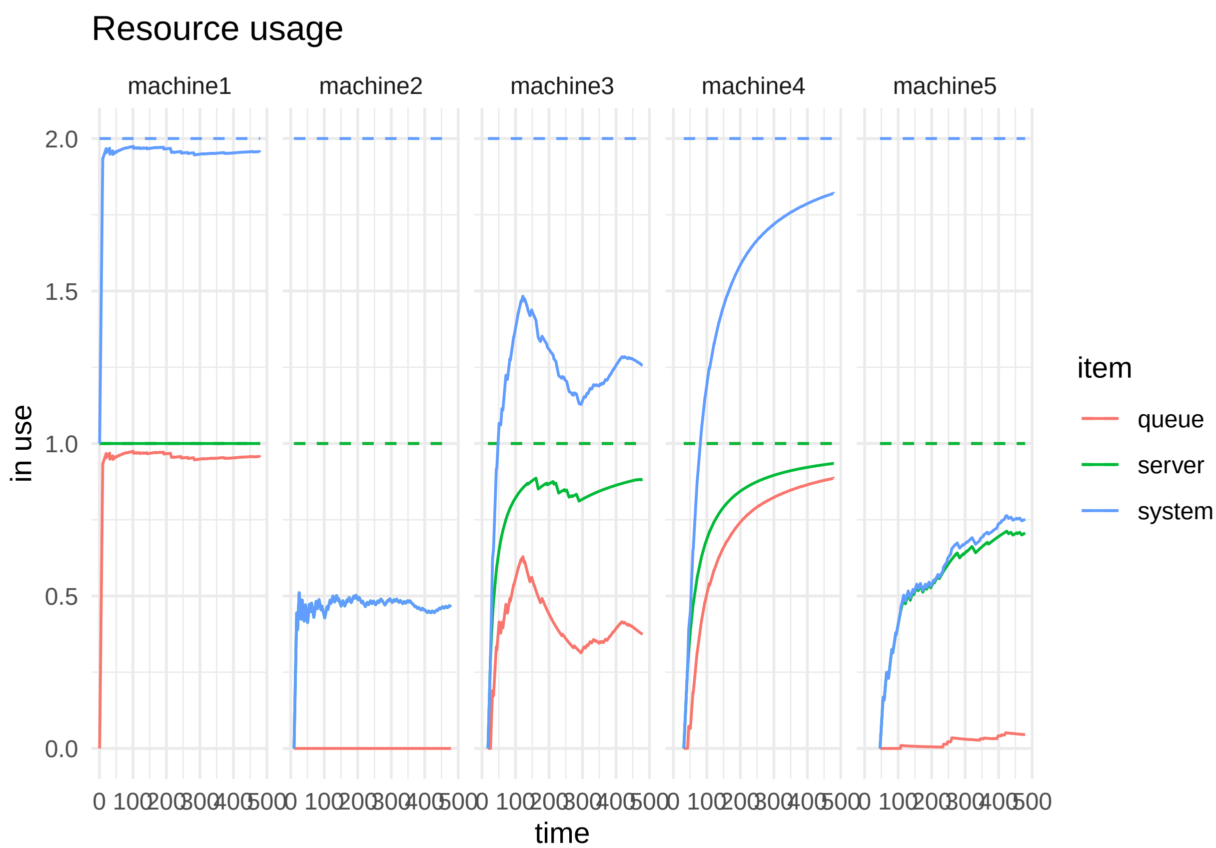
```r
result <- get_mon_arrivals(machineShop)

resourceResult <- get_mon_resources(machineShop)

plot(result, metric = "activity_time") +
  theme_minimal()
```



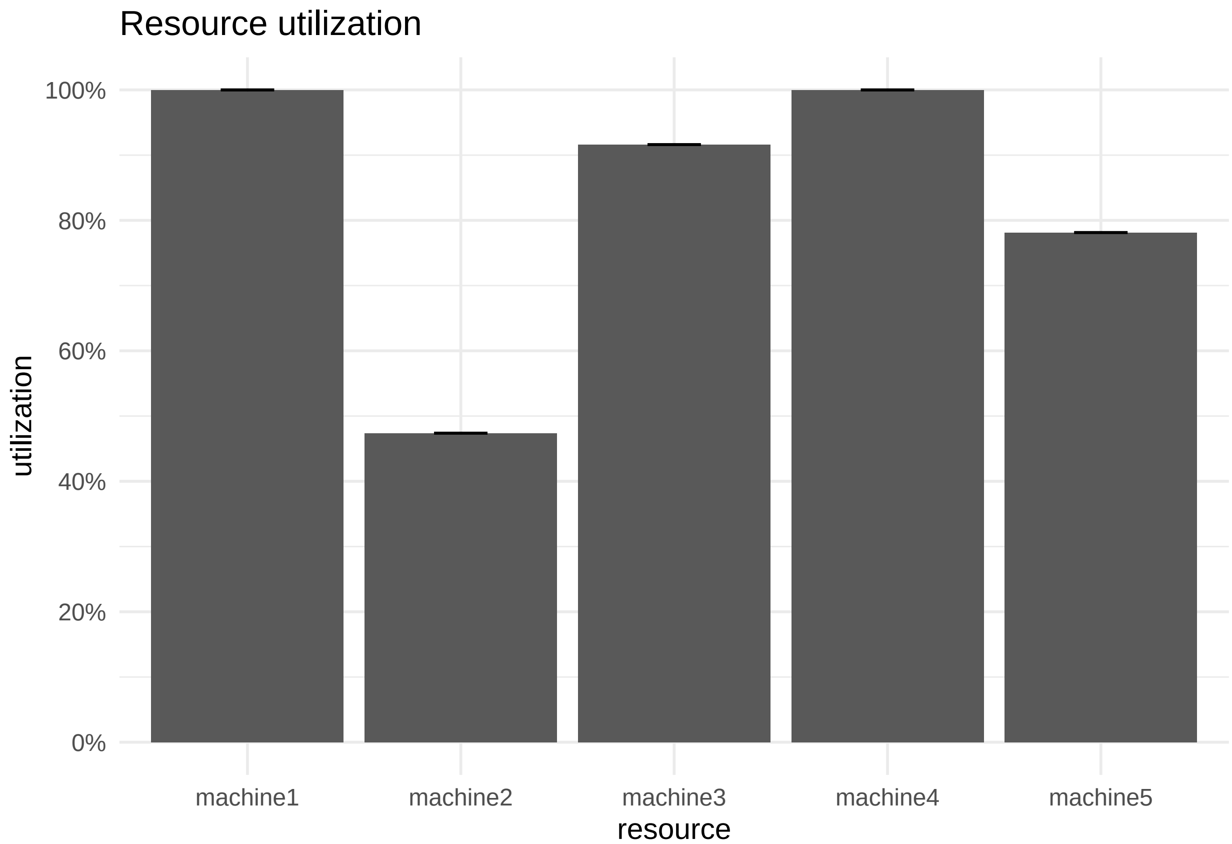

```
plot(result, metric = "waiting_time") +
  theme_minimal()
```




```
plot(resourceResult, metric="usage") +
  theme_minimal()
```




```
plot(resourceResult, metric="utilization") +
  theme_minimal()
```




```
sum(result$finished[result$finished == TRUE])
```

```
[1] 34
```

```
make_parts <- trajectory("parts") %>%
  set_attribute("start_time", function() {now(machineShop)}) %>%
  seize("machine1") %>%
  timeout(function() {rnorm(n = 1, mean = 10, sd = 1)}) %>%
  release("machine1") %>%
  seize("machine2", continue = FALSE,
        reject = trajectory() %>%
          timeout(1) %>%
          rollback(amount = 2, times = Inf)) %>%
  timeout(function() {rnorm(n = 1, mean = 5, sd = 2)}) %>%
  release("machine2") %>%
  seize("machine3", continue = FALSE,
        reject = trajectory() %>%
          timeout(1) %>%
          rollback(amount = 2, times = Inf)) %>%
  timeout(function() {runif(n = 1, min = 5, max = 15)}) %>%
  release("machine3") %>%
  seize("machine4", continue = FALSE,
        reject = trajectory() %>%
          timeout(1) %>%
          rollback(amount = 2, times = Inf)) %>%
  timeout(function() {runif(n = 1, min = 10, max = 15)}) %>%
  release("machine4") %>%
```

```
  seize("machine5", continue = FALSE,
        reject = trajectory() %>%
          timeout(1) %>%
          rollback(amount = 2, times = Inf)) %>%
  timeout(function() {rnorm(n = 1, mean = 10, sd = 2.5)}) %>%
  release("machine5")

machineShop <- simmer("machineShop") %>%
  add_resource("machine1", capacity = 1, queue_size = 1) %>%
  add_resource("machine2", capacity = 1, queue_size = 1) %>%
  add_resource("machine3", capacity = 1, queue_size = 1) %>%
  add_resource("machine4", capacity = 1, queue_size = 1) %>%
  add_resource("machine5", capacity = 1, queue_size = 1) %>%
  add_generator("part", make_parts, mon = 1, function() {c(0, rexp(1000, 1/.5), -1)})

simmer::run(machineShop, 480)
```

```
simmer environment: machineShop | now: 480 | next: 480.119363503813
{ Monitor: in memory }
{ Resource: machine1 | monitored: TRUE | server status: 1(1) | queue status: 1(1) }
{ Resource: machine2 | monitored: TRUE | server status: 0(1) | queue status: 0(1) }
{ Resource: machine3 | monitored: TRUE | server status: 1(1) | queue status: 1(1) }
{ Resource: machine4 | monitored: TRUE | server status: 1(1) | queue status: 1(1) }
{ Resource: machine5 | monitored: TRUE | server status: 1(1) | queue status: 0(1) }
{ Source: part | monitored: 1 | n_generated: 1001 }
```

```
result <- get_mon_arrivals(machineShop)
```