**WEEK 6 HOMEWORK – SAMPLE SOLUTIONS**

# *IMPORTANT NOTE*

These homework solutions show multiple approaches and some optional extensions for most of the questions in the assignment. You don't need to submit all this in your assignments; they're included here just to help you learn more – because remember, the main goal of the homework assignments, and of the entire course, is to help you learn as much as you can, and develop your analytics skills as much as possible!

**Question 13.2**

*In this problem, you can simulate a simplified airport security system at a busy airport. Passengers arrive according to a Poisson distribution with $\lambda_1 = 5$ per minute (i.e., mean interarrival rate $\mu_1 = 0.2$ minutes) to the ID/boarding-pass check queue, where there are several servers who each have exponential service time with mean rate $\mu_2 = 0.75$ minutes. [Hint: model them as one block that has more than one resource.] After that, the passengers are assigned to the shortest of the several personal-check queues, where they go through the personal scanner (time is uniformly distributed between 0.5 minutes and 1 minute).*

*Use the Arena software (PC users) or Python with SimPy (PC or Mac users) to build a simulation of the system, and then vary the number of ID/boarding-pass checkers and personal-check queues to determine how many are needed to keep average wait times below 15 minutes. [If you're using SimPy, or if you have access to a non-student version of Arena, you can use $\lambda_1 = 50$ to simulate a busier airport.]*

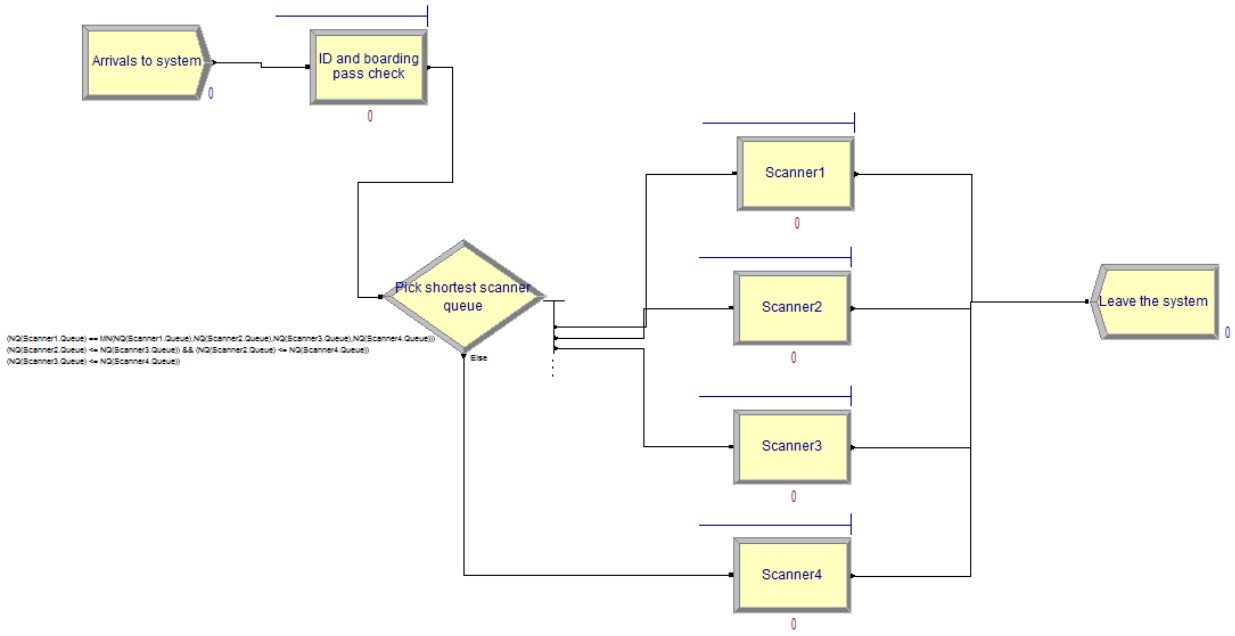
Here's one possible solution. Please note that a good solution doesn't have to try all of the possibilities in the code; they're shown to help you learn, but they're not necessary.

These solutions show both an ARENA model and a SimPy model; I suggest you look at both, just to get familiar with the two of them.

*ARENA VERSION*

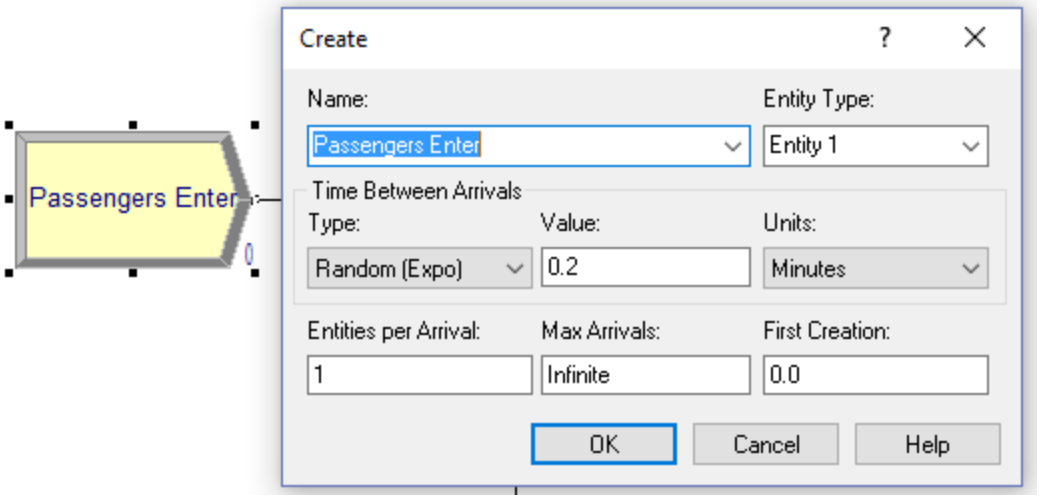
The file solution 13.2.doe contains the ARENA model for this problem. In the picture below:

(1) A "Create" block is used to model passenger arrivals to the system; they're created as entities that will go through the security system.
(2) From the passenger arrival block, they go to the ID and boarding pass check – because it's a queue, it's a "Process" block. The line above the block is the queue where passengers wait until it's their turn if it's animated.
(3) After the ID and boarding pass check, passengers go to the rhombus-shaped "Decide" block, where they identify the scanner with the shortest queue, and go there.
(4) The four scanner "Process" blocks are queues, like the ID and boarding pass check.
(5) After finishing at a scanner, passengers leave the system. That's a "Dispose" block; it tells the system that it can dispose of the entity so it doesn't need to track it anymore. (That's especially important if you have the student or trial version of Arena that can only handle 150 entities at a time.)



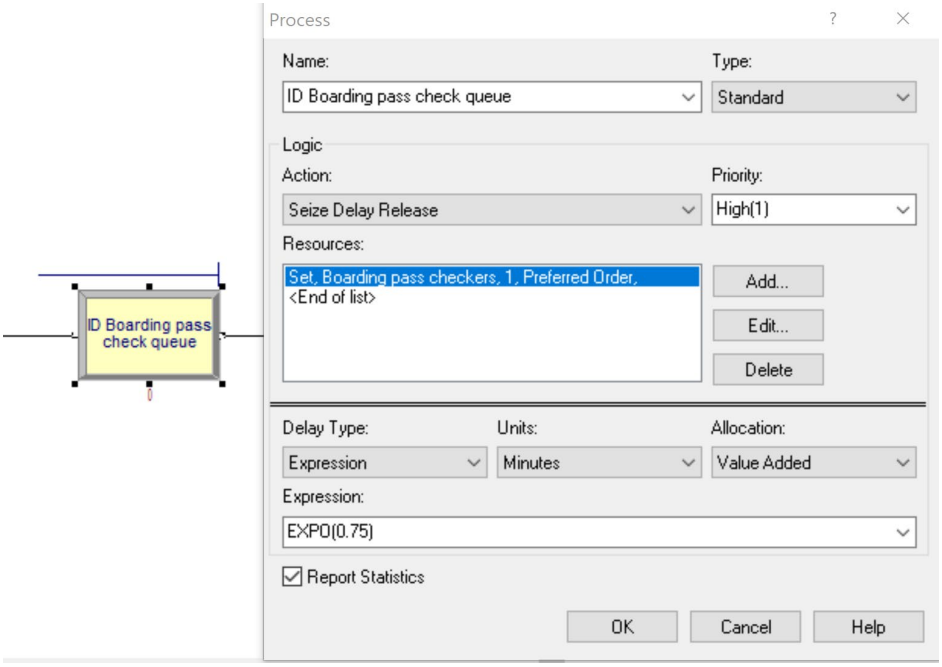

Below, we'll go through each step in more detail.

(1) Passengers arrive at the airport according to a Poisson distribution with rate 5 persons/minute. The interarrival time is thus 0.2 minutes. To simulate these arrivals, we can use a Create block in Arena (found under Basic Process) to create entities. In this model, the entities will be simulated people; for other models they can be other things – for example, phone calls if simulating a call center. Here is the Create block dialog box:



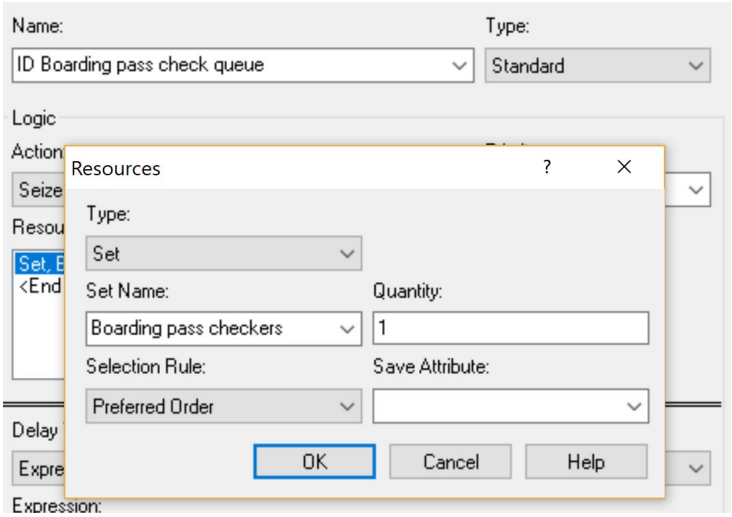

In the box, we can set the name of the block, the type of arrivals (Exponential, with interarrival time 0.2 minutes), and that there's just 1 entity created at a time – if we had a more-complex simulation, we could model families arriving together, but we won't do that here.

(2) The first step entities undergo after arriving at the airport is to get in line and get their boarding passes checked. There are multiple personnel (resources) doing the checking. For each passenger in line, 1 person takes exponential time with mean rate of 0.75 minutes. We use a Process block (again found under Basic Process) in Arena.




The action is "Seize Delay Release" – a queue where, when an entity is being served, a resource is "seized" (it's being used), there's a "delay" equal to the service time, and then it's "released" (made available to serve another entity).

The resource is actually a number of resources that can work in parallel on multiple entities at the same time. NOTE: Changing the value in the "Quantity" box changes the number of boarding-pass checkers required to serve *each entity*. It's *not* the number of boarding-pass checkers available overall. To change that, we need to use the "Resources" table (from the Basic Processes area).



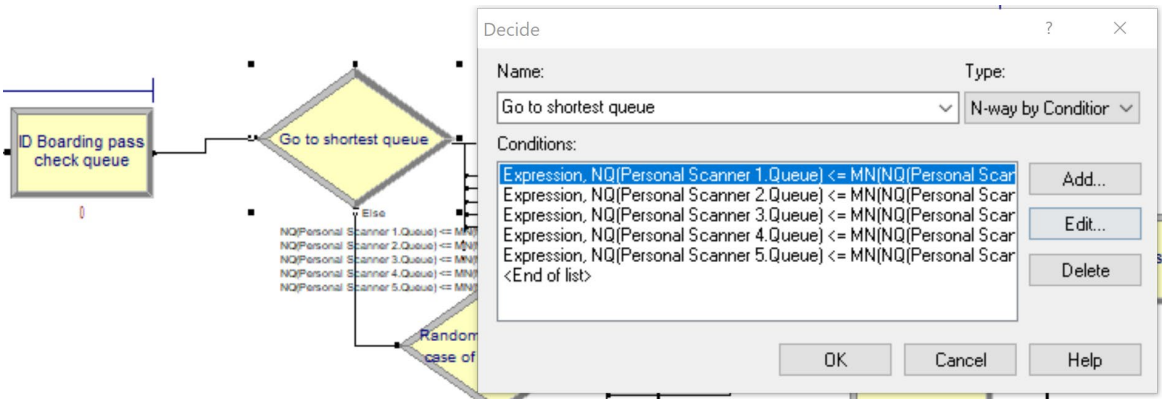

- To specify how many resources there are going to be (e.g., the number of boarding-pass checkers) we used the Resource spreadsheet in the Basic Processes area. Capacity is the number of resources of that type.

Resource - Basic Process

| | Name | Type | Capacity | Busy / Hour | Idle / Hour | Per Use | StateSet Name | Failures | Report Statistics |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Scanner 1 | Fixed Capacity | 1 | 0.0 | 0.0 | 0.0 | | 0 rows | ☑ |
| 2 | Resource 1 | Fixed Capacity | 4 | 0.0 | 0.0 | 0.0 | | 0 rows | ☑ |
| 3 | Scanner 2 | Fixed Capacity | 1 | 0.0 | 0.0 | 0.0 | | 0 rows | ☑ |
| 4 | Scanner 3 | Fixed Capacity | 1 | 0.0 | 0.0 | 0.0 | | 0 rows | ☑ |
| 5 | Scanner 4 | Fixed Capacity | 1 | 0.0 | 0.0 | 0.0 | | 0 rows | ☑ |

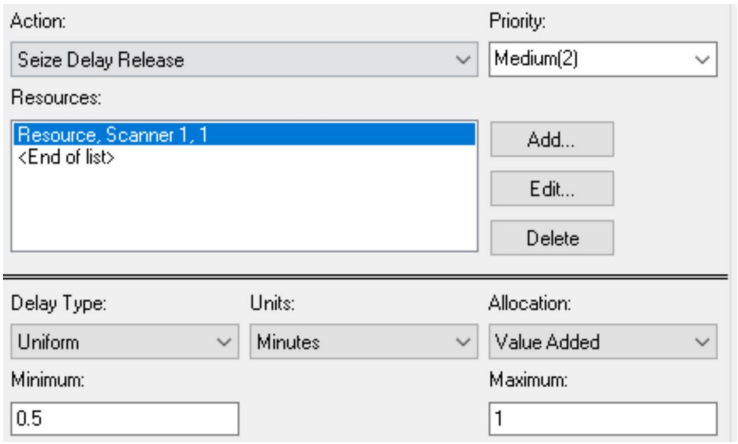
Double-click here to add a new row.

(3) After the boarding pass checking station, the passengers proceed to the scanners. There are several scanners for the passengers to choose from. Each passenger will prefer to go to the scanner with the shortest queue. To represent this decision in Arena, we use the Decide block in Arena and add IF THEN conditions in it. One example condition is – If the length of queue 2 is less than the minimum length of the queues 1,3,4 and 5, then the passenger goes to queue 2.



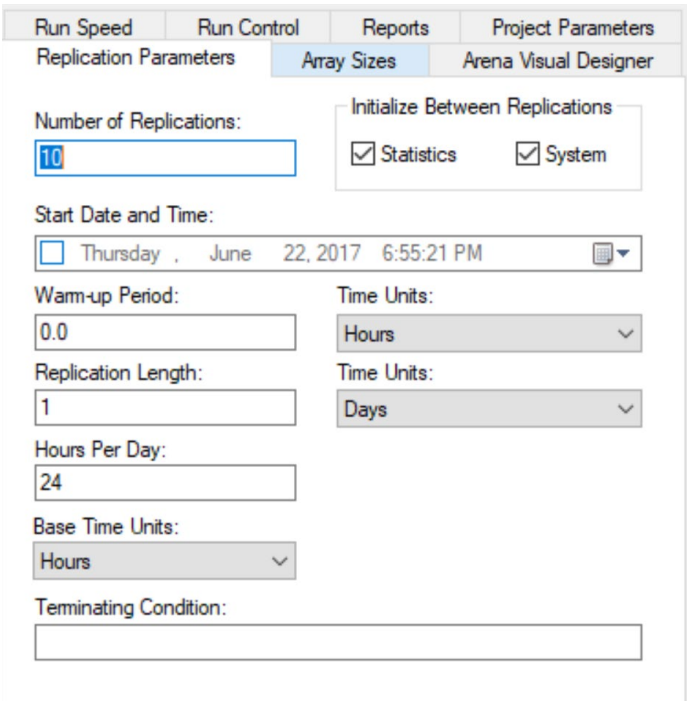

As we are making a decision with multiple outcomes, this is an "N-way by Condition".

(4) Once the decision is made, the passenger goes to one of the scanners. Scanning time is uniformly distributed with a minimum of 0.5 minutes and maximum of 1 minute. In Arena, this is again defined in a Process block with a single resource (there is only 1 scanner in each line) with a Uniform delay type. Once again, the action is Seize Delay Release as only one passenger can use a scanner at a time.




(5) Once these processes are carried out, passengers leave for their respective flights. In the simulation, this means passengers exit the system. Thus, for this we use a Dispose block in Arena that removes the entities from the system and also keeps a count of the entities that exited.

After the process flow is set up, we specify for how long we want to run our simulation and how many times we want to run the simulation (we should always run simulations multiple times to account for randomness). In Arena, this can be done by selecting Setup.. in the Run dropdown menu at the top.




In our model, we chose 100 replications.

Next, we decide how long each replication is. In this case, we run the model for 12 hours each replication.

If you're watching it run, you'll notice that it takes a long time. That's because the animation slows it down a lot. Watching the animation is very useful to make sure the model is doing what you want it to, but for running 100 replications of 12 hours each, even for a small simulation like this, it can really be slow. So, you can turn the animation off: go to Run…Run Control…Batch Run (No Animation) to run it quickly, without watching the animation.

Once all replications have run, we can look at the result reports. These can be found under the Reports section on the left, below basic processes. For our current problem, we only want to evaluate our wait times. To look at the wait times for all the replications, go to the Category Overview Report. To look at the wait times per replication, we can look at the Entities Report or the Queues Report.

Entities:

7:10:31PM **Entities** June 22, 2017

**Unnamed Project** Replications: 10

**Replication 1** Start Time: 0.00 Stop Time: 24.00 Time Units: Hours

Entity 1

| Time | Average | Half Width | Minimum | Maximum |
|---|---|---|---|---|
| Transfer Time | 0.00 | 0.000000000 | 0.00 | 0.00 |
| VA Time | 0.02503637 | 0.000265328 | 0.00841244 | 0.1228 |
| Total Time | 0.05927547 | 0.008436487 | 0.00871862 | 0.2653 |
| Wait Time | 0.03423910 | 0.008352633 | 0.00 | 0.1976 |
| NVA Time | 0.00 | 0.000000000 | 0.00 | 0.00 |
| Other Time | 0.00 | 0.000000000 | 0.00 | 0.00 |

7:11:32PM **Queues** June 22, 2017

**Unnamed Project** Replications: 10

**Replication 1** Start Time: 0.00 Stop Time: 24.00 Time Units: Hours

**Queue Detail Summary**

**Time**

| | Waiting Time |
|---|---|
| ID Boarding pass check queue.Queue | 0.02 |
| Personal Scanner 1.Queue | 0.01 |
| Personal Scanner 2.Queue | 0.01 |
| Personal Scanner 3.Queue | 0.01 |
| Personal Scanner 4.Queue | 0.01 |
| Personal Scanner 5.Queue | 0.01 |

Queues:

7:12:10PM **Category Overview** June 22, 2017

*Values Across All Replications*

**Unnamed Project**

Replications: 10 Time Units: Hours

**Entity**

**Time**

| VA Time | Average | Half Width | Minimum Average | Maximum Average | Minimum Value | Maximum Value |
|---|---|---|---|---|---|---|
| Entity 1 | 0.02492960 | 0.00 | 0.02485557 | 0.02503637 | 0.00841244 | 0.1509 |

| NVA Time | Average | Half Width | Minimum Average | Maximum Average | Minimum Value | Maximum Value |
|---|---|---|---|---|---|---|
| Entity 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

| Wait Time | Average | Half Width | Minimum Average | Maximum Average | Minimum Value | Maximum Value |
|---|---|---|---|---|---|---|
| Entity 1 | 0.03550886 | 0.00 | 0.02772016 | 0.04827202 | 0.00 | 0.2587 |

<u>But so what?</u>

Once the model is built, we need to use it to determine how many scanners and how many boarding-pass checkers are required. To do that, we can vary the numbers in the simulation, run each, and see what the results are.

There's also a shortcut, in this case. With an average of 5 arrivals per minute, we need enough capacity to check at least 5 boarding passes per minute, and to scan at least 5 passengers per minute. If 1 boarding-pass checker takes 0.75 minutes/person (or 4 people every 3 minutes), having 3 isn't enough – they could only process 4 passengers/minute. [You can verify this by running with a resource capacity of 3; you'll find that the queue at the boarding pass check gets bigger and bigger, and if you have a student/trial version the simulation will eventually stop and tell you that it has more than 150 entities in

the system.] So we need at least 4 boarding-pass checkers, which is 16/3 people per minute, slightly more than 5.

The calculation is the same for scanners: each scanner takes an average of 0.75 minutes/person, so again we need at least 4.

So, a first attempt could be 4 boarding-pass checkers and 4 scanners. Here are the results, with 100 replications:

Average wait time in queue: 3.76 minutes
Average total time through system: 5.25 minutes

But… do we have enough replications to be sure the average is really less than 15 minutes? We'll see how to test whether the median is less than 15 minutes later in the course, but for now we can verify in a different way. Out of 100 replications, the highest average wait time was 13.5 minutes, less than 15. So it's very likely that the average wait time is less than 15 minutes (think about this binomial trials, with 0 "successes" [average wait time over 15 minutes] in 100 trials). For total time in the system (wait time and processing time), just 1 of the 100 replications had an average over 15 minutes.

So, the solution seems to be that we need 4 boarding-pass checkers and 4 scanners.

By the way, you can try adding more checkers and/or more scanners, but in this case it doesn't have much effect. Because this is such a simple system, the basic calculation works well.

*SimPy VERSION*

The file solution 13.2.py shows a SimPy model for this system. As you can see, there's much more coding rather than Arena's drag-and-drop[1].

The analysis is pretty much the same, though. Once the model is built, we need to use it to determine how many scanners and how many boarding-pass checkers are required. To do that, we can vary the numbers in the simulation, run each, and see what the results are.

There's also a shortcut, in this case. With an average of 5 arrivals per minute, we need enough capacity to check at least 5 boarding passes per minute, and to scan at least 5 passengers per minute. If 1 boarding-pass checker takes 0.75 minutes/person (or 4 people every 3 minutes), having 3 isn't enough – they could only process 4 passengers/minute. [You can verify this by running with a resource capacity of 3; you'll find that the wait times get very large, because the queues get longer and longer.] So we need at least 4 boarding-pass checkers, which is 16/3 people per minute, slightly more than 5.

[1] Historical note: back when I first learned simulation in college, we didn't have the nice drag-and-drop Arena interface; instead, we used a simulation programming language called SIMAN. SIMAN is still around, in the background in Arena; if you want to see what the SIMAN code looks like, you can open `solution 13.2.doe` in Arena and go to Run…SIMAN…View.

Here are the average wait times for different numbers of boarding pass checkers and scanners:

| Boarding-pass checkers | Scanners | Avg wait time | # replications over 15 min avg wait time | Avg system time | # replications over 15 min avg system time |
|---|---|---|---|---|---|
| 3 | 3 | 78.79 | 100 | 80.30 | 100 |
| 3 | 4 | 73.30 | 100 | 74.80 | 100 |
| 4 | 3 | 72.30 | 100 | 73.80 | 100 |
| 4 | 4 | 3.92 | 0 | 5.42 | 0 |
| 4 | 5 | 2.92 | 0 | 4.42 | 0 |
| 5 | 4 | 3.38 | 0 | 1.88 | 0 |
| 5 | 5 | 0.79 | 0 | 2.29 | 0 |

As we saw above, it looks like we need 4 boarding-pass checkers and 4 scanners; passengers will need to wait on average about 4 minutes. If we add a 5th of each, passengers will wait on average less than 1 minute.

This question used a passenger arrival rate of 5 passengers/minute. But what if the arrival rate was an order of magnitude bigger, like the busy times at a very busy airport? The student/trial versions of Arena can't handle that – the model might have more than 150 entities at a time, beyond the student/trial version limits (the full version of Arena would have no problem). But SimPy can do it. Here's the average wait time (across 100 replications) for each combination of boarding-pass checkers and scanners between 35 and 40:

| **AVERAGE WAIT TIME (MINUTES)** | | Number of scanners | | | | | |
|---|---|---|---|---|---|---|---|
| | | 35 | 36 | 37 | 38 | 39 | 40 |
| Number of boarding-pass checkers | 35 | 26 | 24 | 24 | 25 | 25 | 25 |
| | 36 | 25 | 17 | 16 | 15 | 15 | 15 |
| | 37 | 25 | 15 | 8 | 7 | 6 | 7 |
| | 38 | 25 | 15 | 6 | 2 | 2 | 2 |
| | 39 | 25 | 15 | 6 | 2 | 1 | 1 |
| | 40 | 24 | 15 | 6 | 1 | 1 | 1 |

Based on these runs, it looks like 37 boarding-pass checkers and 37 scanners is sufficient to get average waiting time below 15 minutes (all the way down to 8 minutes). The code for this is in the file solution 13.2-50.py.

**Question 14.1**

*The breast cancer data set breast-cancer-wisconsin.data.txt from [http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/](http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/) (description at [http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29](http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29) ) has missing values.*

1. *Use the mean/mode imputation method to impute values for the missing data.*
2. *Use regression to impute values for the missing data.*

3. *Use regression with perturbation to impute values for the missing data.*
4. *(Optional) Compare the results and quality of classification models (e.g., SVM, KNN) build using*
   *(1) the data sets from questions 1,2,3;*
   *(2) the data that remains after data points with missing values are removed; and*
   *(3) the data set when a binary variable is introduced to indicate missing values.*

Here's one possible solution. Please note that a good solution doesn't have to try all of the possibilities in the code; they're shown to help you learn, but they're not necessary.

The file solution 14.1.R shows one possible solution. In it, missing data is identified (only variable V7 has any, and it is only a small amount). Five different data sets are created to deal with the missing data:

(1) Replacing missing values with the mode. This could have gone either way (mode or mean). The data is categorical, but it takes integer values from 1 to 10, and as we'll see later the values seem to have some relative meaning, so they're also somewhat continuous.
(2) Using regression to estimate missing values. Here too could have gone either way (see above)… but since we didn't cover multinomial logistic regression in this course, the solutions treat the data as continuous for this part. Once the missing values are estimated, the estimates are rounded (because the original values are all integer) and values larger or smaller than the extremes are shrunk to the extremes.
(3) Using regression plus perturbation.
(4) Removing rows with missing data.
(5) Adding a binary variables to indicate when data is missing, and adding the necessary interaction variables also.

Once the data sets have been created, we use KNN (for k=1,2,3,4,5) and SVM (C=0.0001,0.001,0.01,0.1,1,10) to create classification models, and measure their quality. The table below shows the results.

| | Method for dealing with missing data | | | | |
|---|---|---|---|---|---|
| Model | Impute using mode | Impute using regression | Impute using regression, then perturb | Remove rows with missing data | Add binary variable for missing data |
| KNN (k=1) | 0.952 | 0.948 | 0.948 | 0.952 | 0.952 |
| KNN (k=2) | 0.952 | 0.948 | 0.948 | 0.952 | 0.952 |
| KNN (k=3) | 0.924 | 0.919 | 0.919 | 0.923 | 0.924 |
| KNN (k=4) | 0.924 | 0.919 | 0.919 | 0.923 | 0.924 |
| KNN (k=5) | 0.919 | 0.914 | 0.914 | 0.913 | 0.919 |
| | | | | | |
| SVM (C=0.0001) | 0.662 | 0.662 | 0.662 | 0.659 | 0.662 |
| SVM (C=0.001) | 0.943 | 0.943 | 0.943 | 0.942 | 0.943 |
| SVM (C=0.01) | 0.957 | 0.957 | 0.957 | 0.957 | 0.957 |
| SVM (C=0.1) | 0.962 | 0.962 | 0.962 | 0.962 | 0.962 |
| SVM (C=1) | 0.967 | 0.962 | 0.967 | 0.966 | 0.967 |
| SVM (C=10) | 0.967 | 0.962 | 0.967 | 0.966 | 0.967 |

It turns out that there isn't much difference in model performance across the five ways of dealing with missing data. The best SVM models are a little better than the best KNN models, but SVM is harder to calibrate; the worst SVM models tested are much worse than the worst KNN models.

**Question 15.1**

*Describe a situation or problem from your job, everyday life, current events, etc., for which optimization would be appropriate. What data would you need?*

Some (perhaps overzealous!) baseball fans have tried to drive around the country to see a baseball game at each of the 30 Major League stadiums, and then return home, in the shortest number of days. This can be modeled using optimization: minimize the number of days it takes, subject to the constraints that a game is seen at each stadium, and the planned sequence of games is possible given the driving times and game schedules. The necessary data would include the Major League schedule (which stadiums have games scheduled on each day, and what time they're scheduled for), and how long it takes to drive between each pair of stadiums.