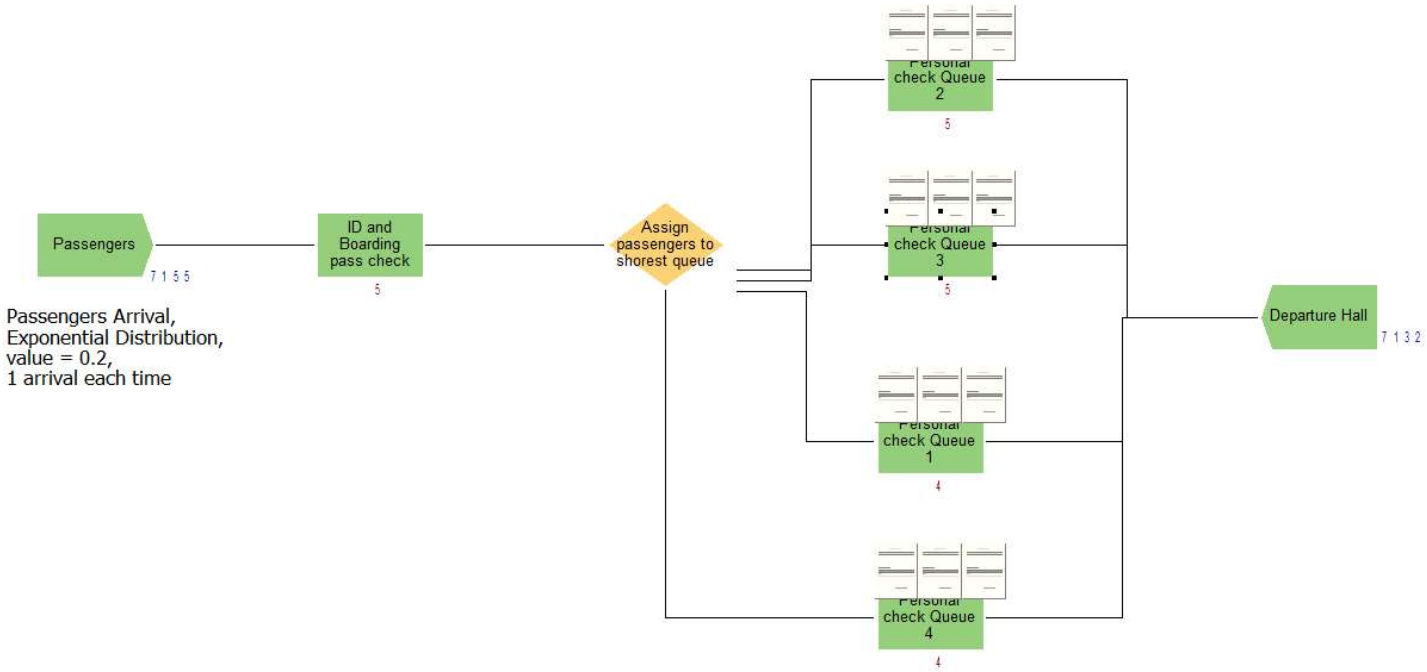
HW week 6

2022-06-25

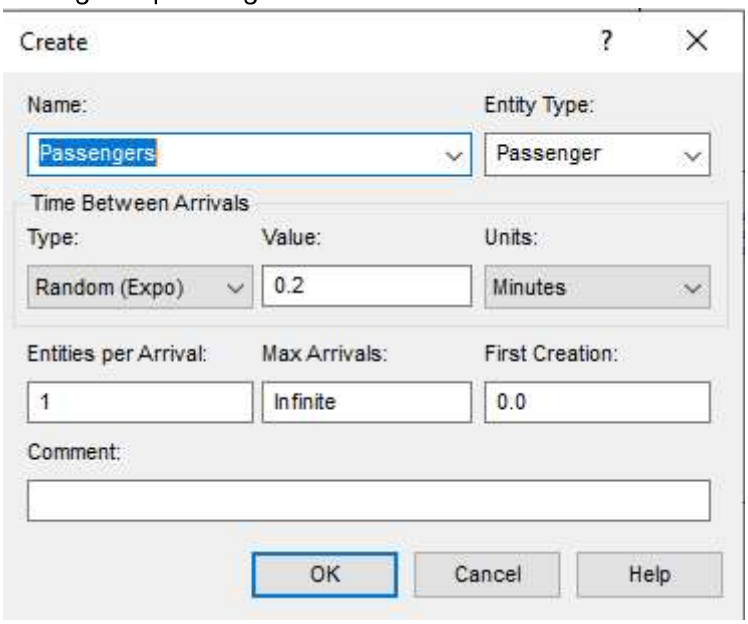
Question 13.2
Answers:

The following screenshot shows the flowchart we set up in Arena. The settings of some key blocks are also displayed in separate screenshots:

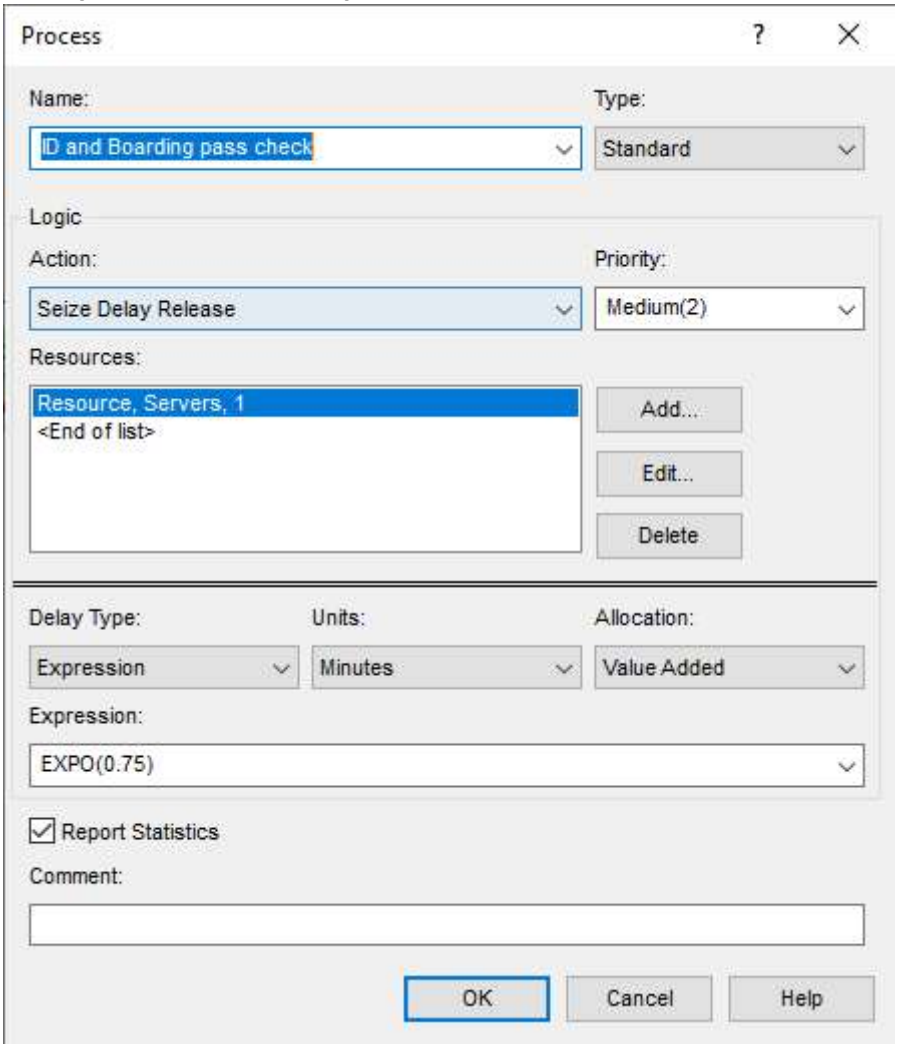
The flow chart:



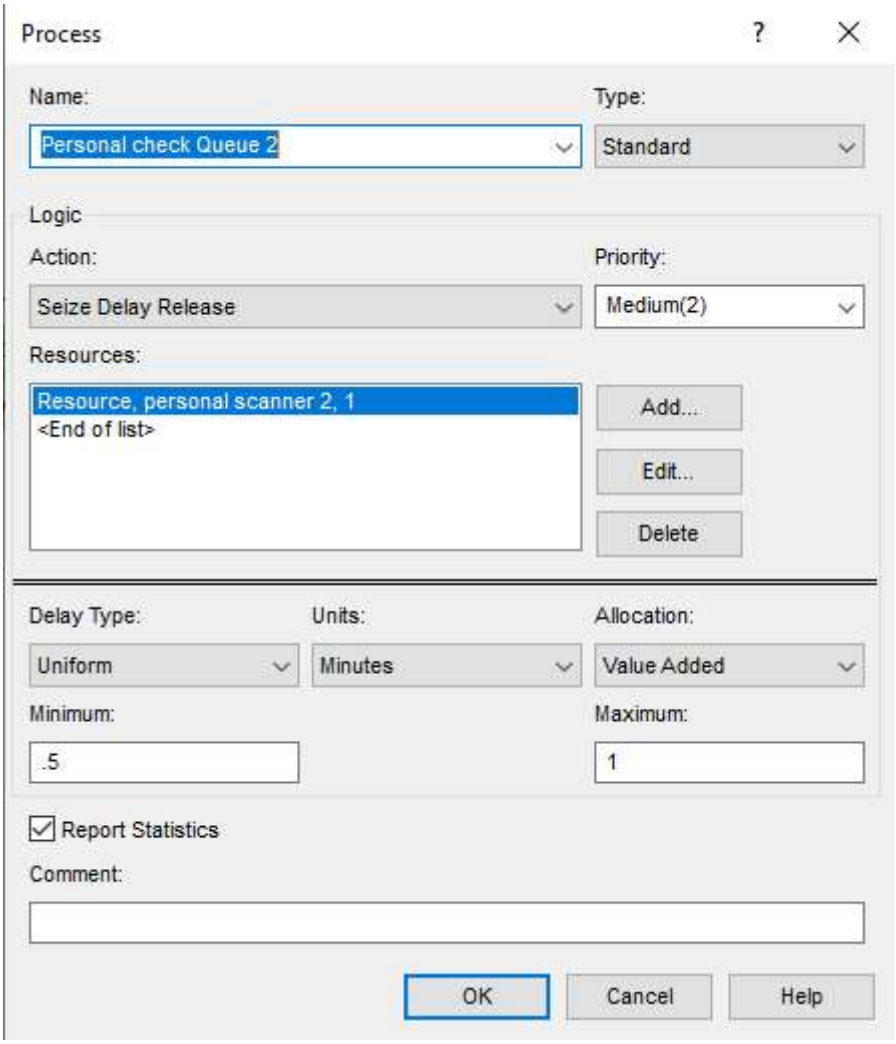
Settings of “passenger” block:



Settings of “ID and Boarding Pass Check” block:



Settings of “Personal Check Queue” block (using one out of four as example):



To change the number of personal check queues. We need to add more branches to the flowchart and change the conditions in the n-way decision activity (Assign passengers to shorest queue), so that we make sure the passengers always go to the shortest queue.
To change the number of servers at “ID and boarding pass check”, we simply need to change the capacity of “Servers” in the resources table:

	Name	Type	Capacity	Busy / Hour	Idle / Hour	Per Use	StateSet Name	Failures	Report Statistics	Comment
1	Servers	Fixed Capacity	4	0.0	0.0	0.0		0 rows	☑	
2	personal scanner 1	Fixed Capacity	1	0.0	0.0	0.0		0 rows	☑	
3	personal scanner 2	Fixed Capacity	1	0.0	0.0	0.0		0 rows	☑	
4	personal scanner 3	Fixed Capacity	1	0.0	0.0	0.0		0 rows	☑	
5	personal scanner 4	Fixed Capacity	1	0.0	0.0	0.0		0 rows	☑	

After trying different numbers of servers and personal check queues, we finally decide to keep 4 servers and 4 personal check queues in the flowchart, because this setting:

1. Avoids the error of exceeding 150 maximum passengers waiting in queues
2. Keeps the average waiting time below 15 minutes

Note: In this simulation, we let the software to run 10 replications. Each replication will last 24 hours to make sure the results are not biased. Here’s one of the 10 replication that has the longest average waiting time:

ARENA Simulation Results
suiwenyu@hotmail.com - License: STUDENT

Summary for Replication 5 of 10

Project: Unnamed Project
Analyst: suiwenyu@hotmail.com

Run execution date : 6/27/2022
Model revision date: 6/27/2022

Replication ended at time : 1440.0 Minutes
Base Time Units: Minutes

TALLY VARIABLES

Identifier	Average	Half Width	Minimum	Maximum	Observations
Passenger.VATime	1.4965	.02274	.51274	7.1201	7227
Passenger.NVATime	.00000	.00000	.00000	.00000	7227
Passenger.WaitTime	4.9610	(Corr)	.00000	18.675	7227
Passenger.TranTime	.00000	.00000	.00000	.00000	7227
Passenger.OtherTime	.00000	.00000	.00000	.00000	7227
Passenger.TotalTime	6.4575	(Corr)	.69317	22.129	7227
Personal check Queue 2.Queue.WaitingTime	2.4410	(Corr)	.00000	9.8191	1913
Personal check Queue 1.Queue.WaitingTime	2.2077	(Corr)	.00000	10.288	1806
Personal check Queue 3.Queue.WaitingTime	2.2850	(Corr)	.00000	9.7478	1894
Personal check Queue 4.Queue.WaitingTime	2.2536	(Corr)	.00000	9.4243	1618
ID and Boarding pass check.Queue.WaitingTi	2.6553	(Corr)	.00000	14.969	7246

Question 14.1
Question 14.1.1
Answer:

In this part, we will use mean/mode to impute missing values. First read the data.
From the summary info we can see that only column V7 has 16 missing values. (about 2.29% of total number of rows)

```
    rm(list = ls())
    set.seed(19)
    data <- read.table("C:\\Data\\week 6 data-summer\\data 14.1\\breast-cancer-wisconsin.data.tx
t",
                       stringsAsFactors = FALSE, header = FALSE, sep = ",",
                       na.strings = "?")
    summary(data)
```

```
##        V1                 V2               V3               V4
##  Min.   :   61634   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.:  870688   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000
##  Median : 1171710   Median : 4.000   Median : 1.000   Median : 1.000
##  Mean   : 1071704   Mean   : 4.418   Mean   : 3.134   Mean   : 3.207
##  3rd Qu.: 1238298   3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000
##  Max.   :13454352   Max.   :10.000   Max.   :10.000   Max.   :10.000
##
##        V5               V6               V7               V8
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.: 1.000   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 2.000
##  Median : 1.000   Median : 2.000   Median : 1.000   Median : 3.000
##  Mean   : 2.807   Mean   : 3.216   Mean   : 3.545   Mean   : 3.438
##  3rd Qu.: 4.000   3rd Qu.: 4.000   3rd Qu.: 6.000   3rd Qu.: 5.000
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000
##                                    NA's   :16
##        V9              V10              V11
##  Min.   : 1.000   Min.   : 1.000   Min.   :2.00
##  1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:2.00
##  Median : 1.000   Median : 1.000   Median :2.00
##  Mean   : 2.867   Mean   : 1.589   Mean   :2.69
##  3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:4.00
##  Max.   :10.000   Max.   :10.000   Max.   :4.00
##
```

```
    #Display all rows with missing values
    missing_rows <- is.na(data$V7)
    data[missing_rows,]
```

```
##          V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11
## 24  1057013  8  4  5  1  2 NA  7  3   1   4
## 41  1096800  6  6  6  9  6 NA  7  8   1   2
## 140 1183246  1  1  1  1  1 NA  2  1   1   2
## 146 1184840  1  1  3  1  2 NA  2  1   1   2
## 159 1193683  1  1  2  1  3 NA  1  1   1   2
## 165 1197510  5  1  1  1  2 NA  3  1   1   2
## 236 1241232  3  1  4  1  2 NA  3  1   1   2
## 250  169356  3  1  1  1  2 NA  3  1   1   2
## 276  432809  3  1  3  1  2 NA  2  1   1   2
## 293  563649  8  8  8  1  2 NA  6 10   1   4
## 295  606140  1  1  1  1  2 NA  2  1   1   2
## 298   61634  5  4  3  1  2 NA  2  3   1   2
## 316  704168  4  6  5  6  7 NA  4  9   1   2
## 322  733639  3  1  1  1  2 NA  3  1   1   2
## 412 1238464  1  1  1  1  1 NA  2  1   1   2
## 618 1057067  1  1  1  1  1 NA  1  1   1   2
```

Then we are going to impute the missing values with mean and mode. We are going to use mfv() function in “modeest” package to find the mode. Here’s the approach:

```
library(modeest)
```

```
# Copy the data
data_impute_mean <- data
#impute missing values with mean
data_impute_mean$V7[missing_rows] <- mean(data$V7, na.rm = TRUE)

#Display all rows with missing values in original data
#(filled with mean value)
data_impute_mean[missing_rows,]
```

```
##          V1 V2 V3 V4 V5 V6       V7 V8 V9 V10 V11
## 24  1057013  8  4  5  1  2 3.544656  7  3   1   4
## 41  1096800  6  6  6  9  6 3.544656  7  8   1   2
## 140 1183246  1  1  1  1  1 3.544656  2  1   1   2
## 146 1184840  1  1  3  1  2 3.544656  2  1   1   2
## 159 1193683  1  1  2  1  3 3.544656  1  1   1   2
## 165 1197510  5  1  1  1  2 3.544656  3  1   1   2
## 236 1241232  3  1  4  1  2 3.544656  3  1   1   2
## 250  169356  3  1  1  1  2 3.544656  3  1   1   2
## 276  432809  3  1  3  1  2 3.544656  2  1   1   2
## 293  563649  8  8  8  1  2 3.544656  6 10   1   4
## 295  606140  1  1  1  1  2 3.544656  2  1   1   2
## 298   61634  5  4  3  1  2 3.544656  2  3   1   2
## 316  704168  4  6  5  6  7 3.544656  4  9   1   2
## 322  733639  3  1  1  1  2 3.544656  3  1   1   2
## 412 1238464  1  1  1  1  1 3.544656  2  1   1   2
## 618 1057067  1  1  1  1  1 3.544656  1  1   1   2
```

```
# Copy the data
data_impute_mode <- data
#impute missing values with mode
mode <- mfv(data$V7)
data_impute_mode$V7[missing_rows] <- mode

#Display all rows with missing values in original data
#(filled with mode value)
data_impute_mode[missing_rows,]
```

```
##          V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11
## 24  1057013  8  4  5  1  2  1  7  3   1   4
## 41  1096800  6  6  6  9  6  1  7  8   1   2
## 140 1183246  1  1  1  1  1  1  2  1   1   2
## 146 1184840  1  1  3  1  2  1  2  1   1   2
## 159 1193683  1  1  2  1  3  1  1  1   1   2
## 165 1197510  5  1  1  1  2  1  3  1   1   2
## 236 1241232  3  1  4  1  2  1  3  1   1   2
## 250  169356  3  1  1  1  2  1  3  1   1   2
## 276  432809  3  1  3  1  2  1  2  1   1   2
## 293  563649  8  8  8  1  2  1  6 10   1   4
## 295  606140  1  1  1  1  2  1  2  1   1   2
## 298   61634  5  4  3  1  2  1  2  3   1   2
## 316  704168  4  6  5  6  7  1  4  9   1   2
## 322  733639  3  1  1  1  2  1  3  1   1   2
## 412 1238464  1  1  1  1  1  1  2  1   1   2
## 618 1057067  1  1  1  1  1  1  1  1   1   2
```

Question 14.1.2
Answer:

In this part, we are going to impute the missing values with regressions.
One simple way is to use mice() function in “mice” packages, which contains built-in imputation functionalities.
Hereare the documentations of this function and some tutorials that i have referred to :

https://www.rdocumentation.org/packages/mice/versions/3.14.0/topics/mice
(https://www.rdocumentation.org/packages/mice/versions/3.14.0/topics/mice)
https://www.analyticsvidhya.com/blog/2016/03/tutorial-powerful-packages-imputing-missing-values/
(https://www.analyticsvidhya.com/blog/2016/03/tutorial-powerful-packages-imputing-missing-values/)
https://rpubs.com/McCloud77/300268 (https://rpubs.com/McCloud77/300268)

To run an imputation with linear regression, we simply need to set the “method” parameter in mice() function to be “norm.predict”. Here’s the approach:

```
    library(mice)
```

```
##
## Attaching package: 'mice'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
    impute_reg <- mice(data, method = "norm.predict")
```

```
##
##  iter imp variable
##   1   1  V7
##   1   2  V7
##   1   3  V7
##   1   4  V7
##   1   5  V7
##   2   1  V7
##   2   2  V7
##   2   3  V7
##   2   4  V7
##   2   5  V7
##   3   1  V7
##   3   2  V7
##   3   3  V7
##   3   4  V7
##   3   5  V7
##   4   1  V7
##   4   2  V7
##   4   3  V7
##   4   4  V7
##   4   5  V7
##   5   1  V7
##   5   2  V7
##   5   3  V7
##   5   4  V7
##   5   5  V7
```

```
    data_impute_reg <- complete(impute_reg)

    #Display all rows with missing values in original data
    #(filled with predictions from linear regression)
    data_impute_reg[missing_rows,]
```

```
##          V1 V2 V3 V4 V5 V6       V7 V8 V9 V10 V11
## 24  1057013  8  4  5  1  2 7.191237  7  3   1   4
## 41  1096800  6  6  6  9  6 3.419208  7  8   1   2
## 140 1183246  1  1  1  1  1 1.188951  2  1   1   2
## 146 1184840  1  1  3  1  2 1.579936  2  1   1   2
## 159 1193683  1  1  2  1  3 1.260453  1  1   1   2
## 165 1197510  5  1  1  1  2 1.428797  3  1   1   2
## 236 1241232  3  1  4  1  2 1.943842  3  1   1   2
## 250  169356  3  1  1  1  2 1.562574  3  1   1   2
## 276  432809  3  1  3  1  2 1.740990  2  1   1   2
## 293  563649  8  8  8  1  2 6.432884  6 10   1   4
## 295  606140  1  1  1  1  2 1.303253  2  1   1   2
## 298   61634  5  4  3  1  2 1.186205  2  3   1   2
## 316  704168  4  6  5  6  7 2.083334  4  9   1   2
## 322  733639  3  1  1  1  2 1.469119  3  1   1   2
## 412 1238464  1  1  1  1  1 1.179806  2  1   1   2
## 618 1057067  1  1  1  1  1 1.059370  1  1   1   2
```

Question 14.1.3
Answer:

To run an imputation using linear regression with perturbations, we simply need to set the “method” parameter in mice() function to be “norm.nob”. Here’s the approach:

```
    set.seed(100)

    impute_reg_per <- mice(data, method = "norm.nob")
```

```
##
##  iter imp variable
##   1   1  V7
##   1   2  V7
##   1   3  V7
##   1   4  V7
##   1   5  V7
##   2   1  V7
##   2   2  V7
##   2   3  V7
##   2   4  V7
##   2   5  V7
##   3   1  V7
##   3   2  V7
##   3   3  V7
##   3   4  V7
##   3   5  V7
##   4   1  V7
##   4   2  V7
##   4   3  V7
##   4   4  V7
##   4   5  V7
##   5   1  V7
##   5   2  V7
##   5   3  V7
##   5   4  V7
##   5   5  V7
```

```
    data_impute_reg_per <- complete(impute_reg_per)

    #Display all rows with missing values in original data
    #(filled with predictions from linear regression with perturbations)
    data_impute_reg_per[missing_rows,]
```

```
##          V1 V2 V3 V4 V5 V6         V7 V8 V9 V10 V11
## 24  1057013  8  4  5  1  2  6.2341409  7  3   1   4
## 41  1096800  6  6  6  9  6  2.0999387  7  8   1   2
## 140 1183246  1  1  1  1  1  2.5470044  2  1   1   2
## 146 1184840  1  1  3  1  2  1.3691798  2  1   1   2
## 159 1193683  1  1  2  1  3  0.3528212  1  1   1   2
## 165 1197510  5  1  1  1  2  4.1122555  3  1   1   2
## 236 1241232  3  1  4  1  2  3.9161831  3  1   1   2
## 250  169356  3  1  1  1  2 -0.8288733  3  1   1   2
## 276  432809  3  1  3  1  2  0.5910234  2  1   1   2
## 293  563649  8  8  8  1  2  8.3972921  6 10   1   4
## 295  606140  1  1  1  1  2 -2.0122315  2  1   1   2
## 298   61634  5  4  3  1  2 -1.0535735  2  3   1   2
## 316  704168  4  6  5  6  7 -0.2588221  4  9   1   2
## 322  733639  3  1  1  1  2 -0.8094779  3  1   1   2
## 412 1238464  1  1  1  1  1 -1.3904081  2  1   1   2
## 618 1057067  1  1  1  1  1  0.7757454  1  1   1   2
```

Question 15.1
Answer:

A plant that produces multiple types of products can use optimization to determine the quantity of each type of products to be produced each year. The best production plan should maximize the gross margin ratio of the plant. Here are some data that may be needed:

1. Sales price and production cost for each type of product
2. The quantity of raw materials and labor that each type of product will consume in production (may be needed for setting constraints)
3. Maximum quantity of each product that the plant is able to produce each year