Step 1. After collecting data, our group went to fit the data and find the best distribution for each parameters

```
library(fitdistrplus)
```

```
## Loading required package: MASS
```

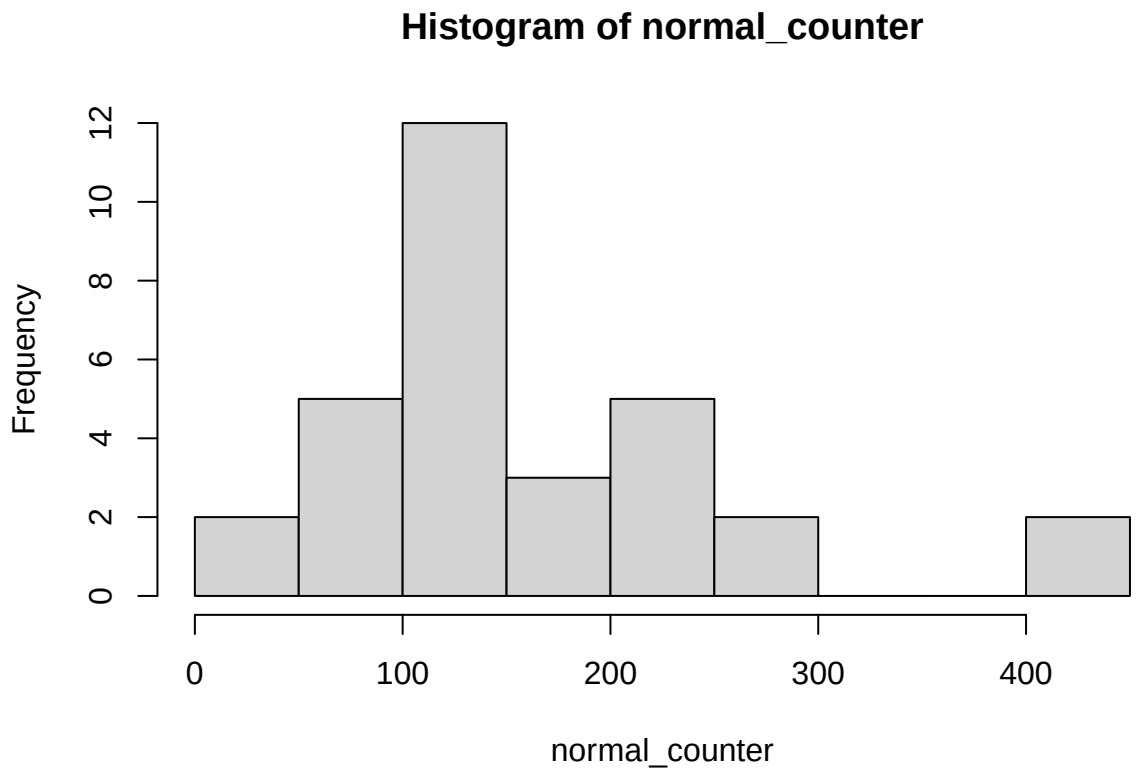
```
## Loading required package: survival
```

```
library(tidyverse)
```

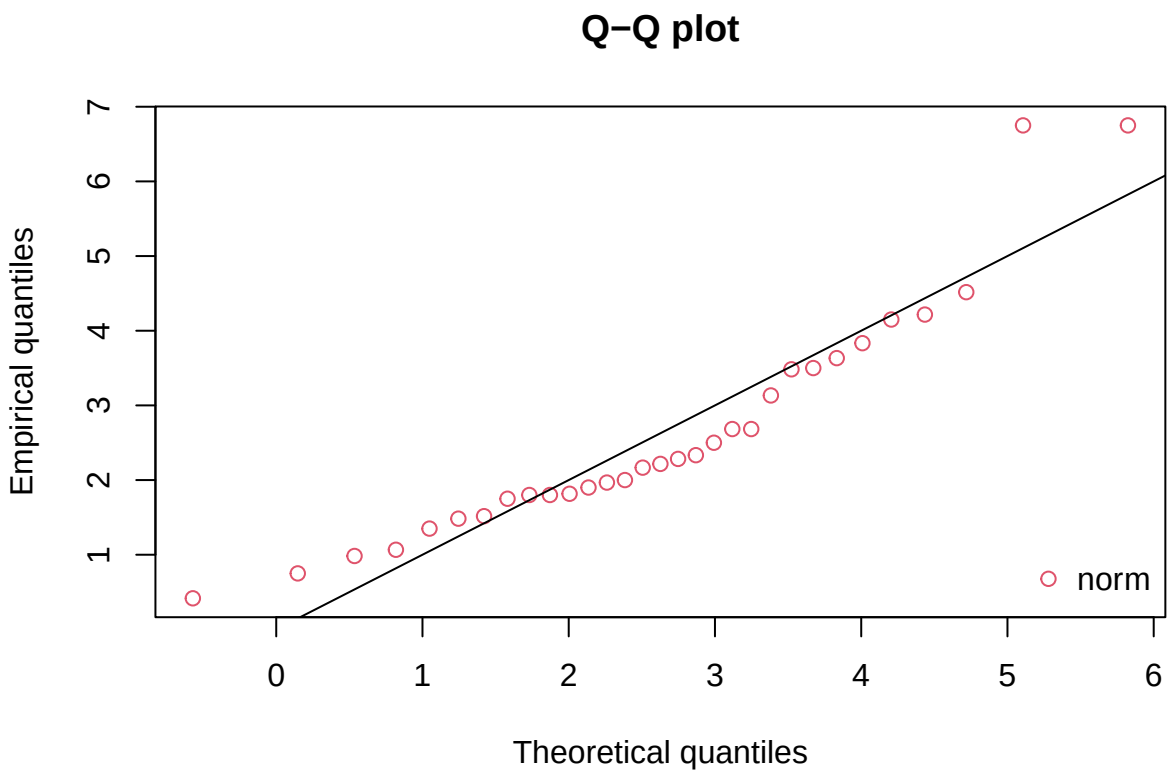
```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
```

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.8     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.1
## v readr   2.1.2     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x dplyr::select() masks MASS::select()
```

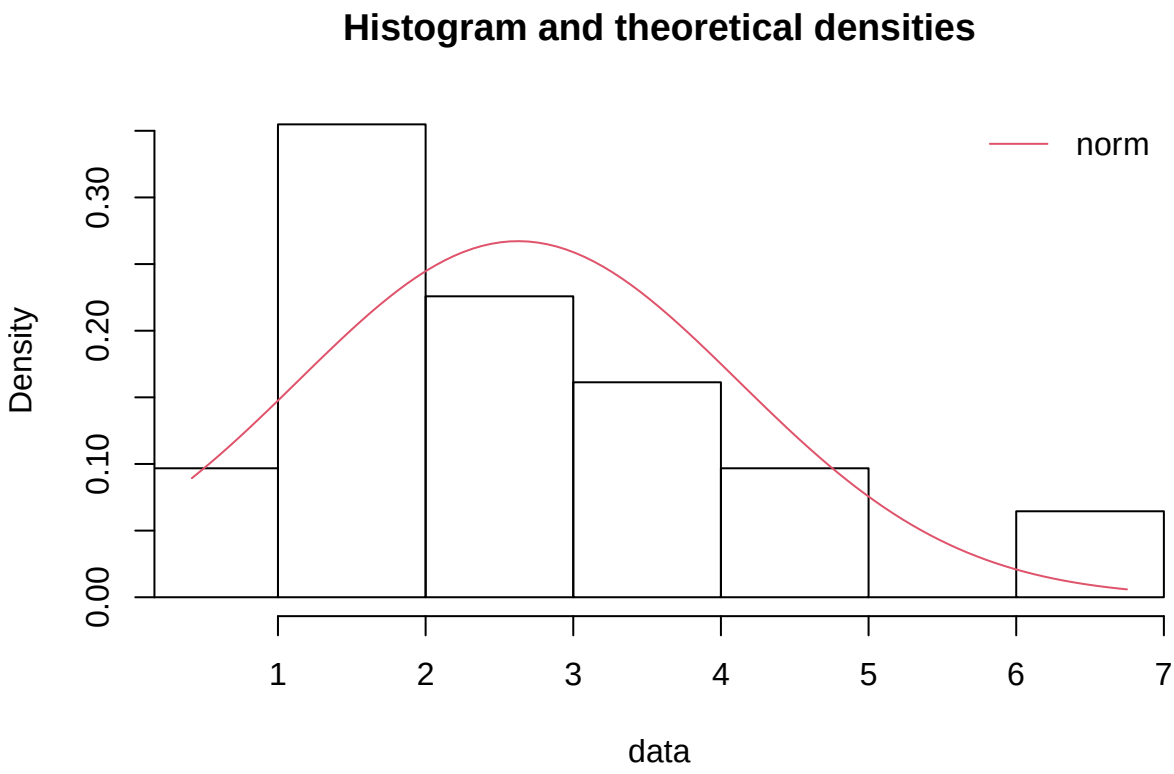
```
normal_counter<-c(89,130,108,108,405,91,118,209,210,120,150,45,161,271,59,218,188,109,140,249,253,81,64
hist(normal_counter)
```



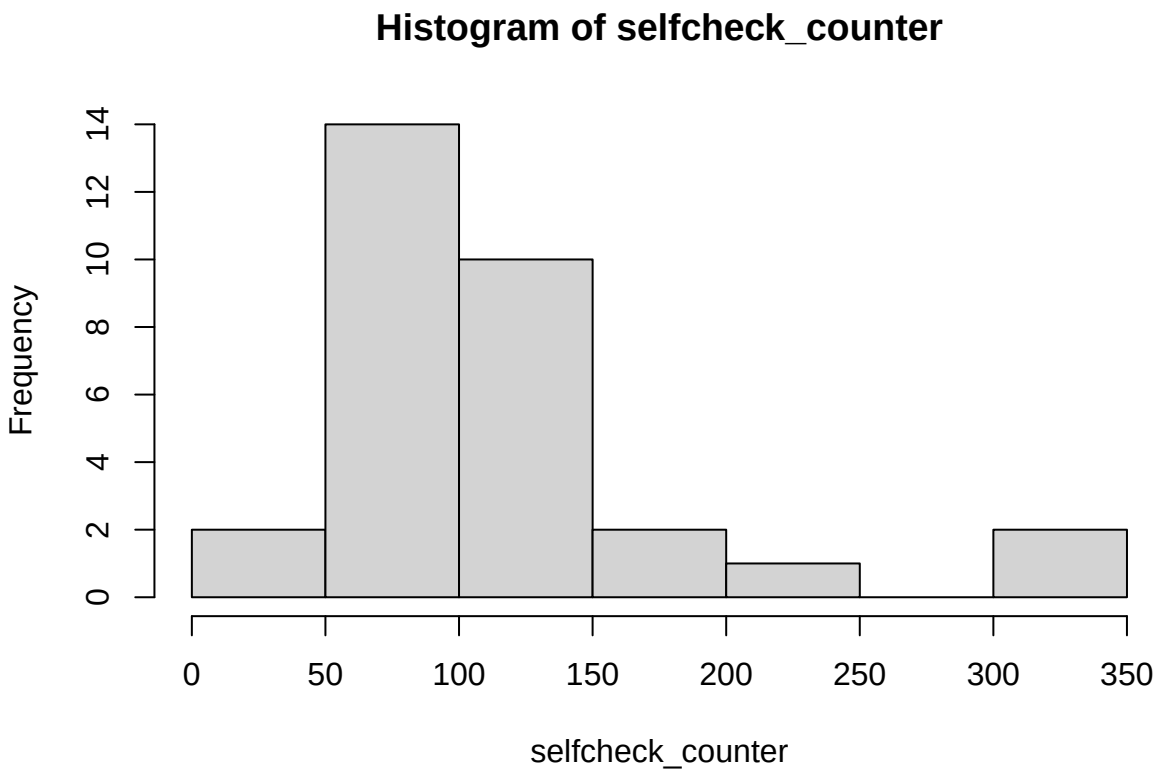
```r
normal_counter<-normal_counter/60 #convert to minute

norm_normalcounter <- fitdist(data=normal_counter,distr="norm")
mean<-norm_normalcounter$estimate[1]
sd<-norm_normalcounter$estimate[2]

qqcomp(norm_normalcounter)
```



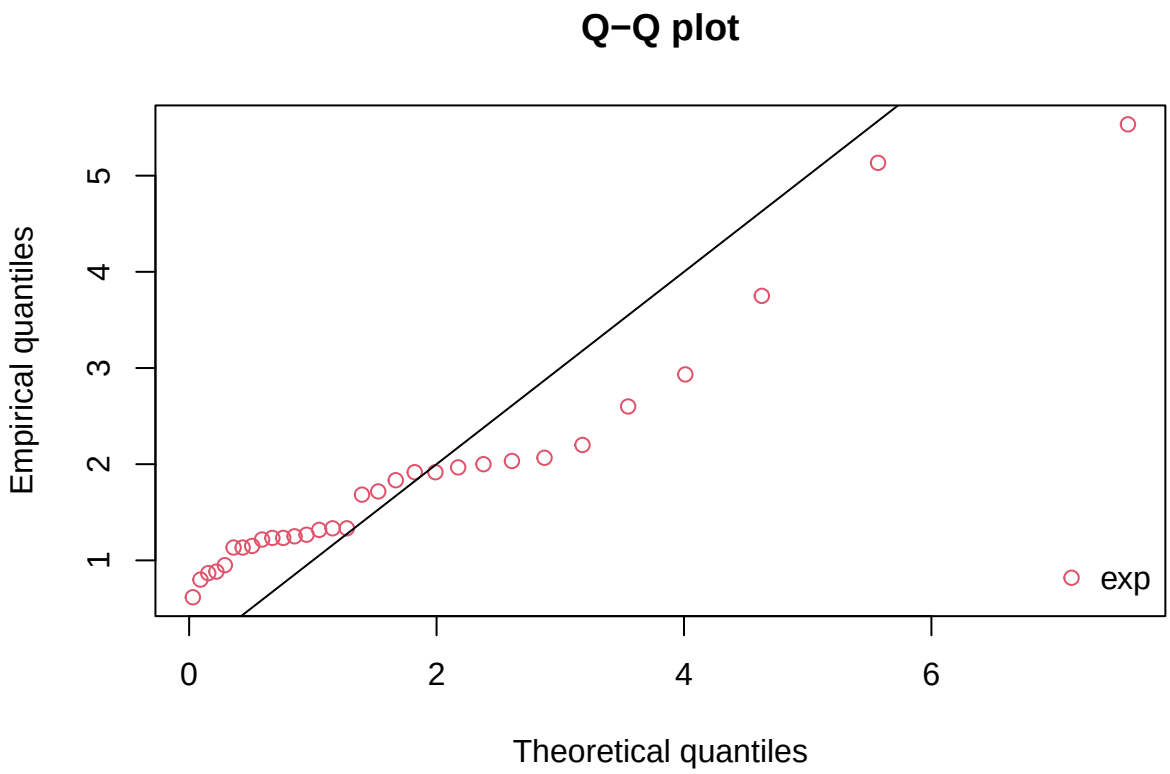

```r
denscomp(norm_normalcounter)
```



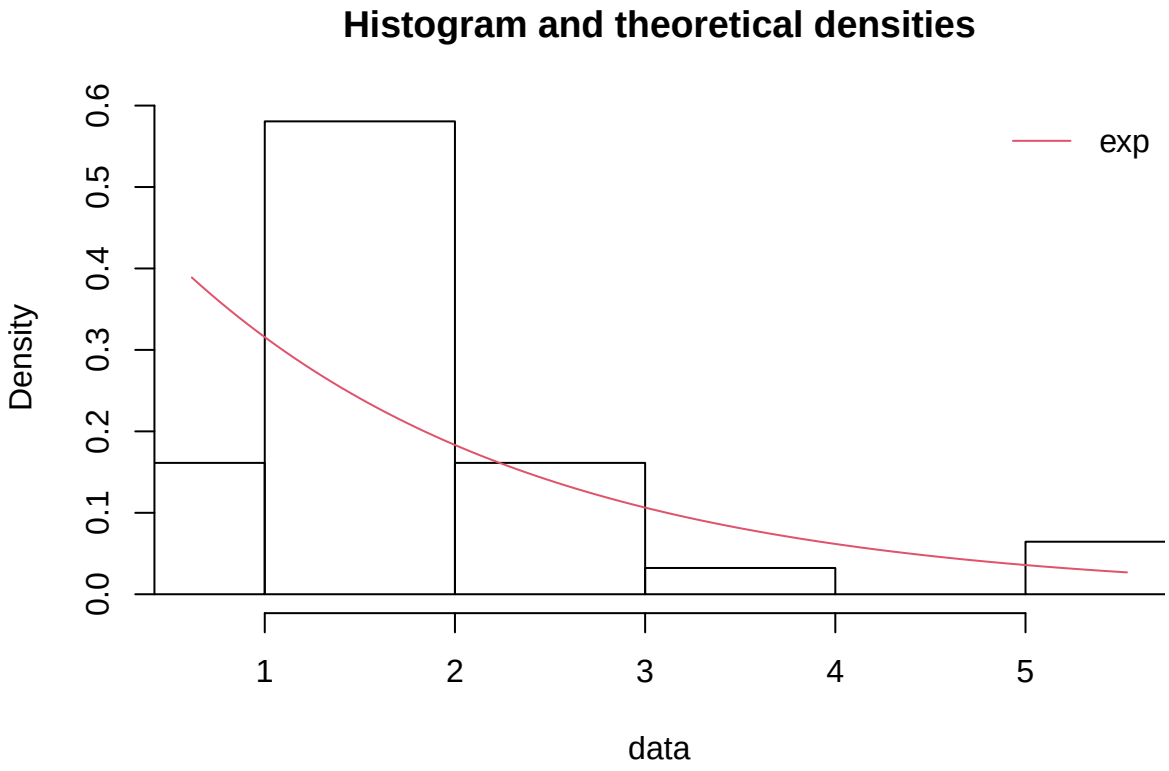

```
selfcheck_counter<-c(115,332,225,308,122,80,118,53,120,74,75,68,52,124,74,101,69,115,73,110,156,48,57,3

hist(selfcheck_counter)
```



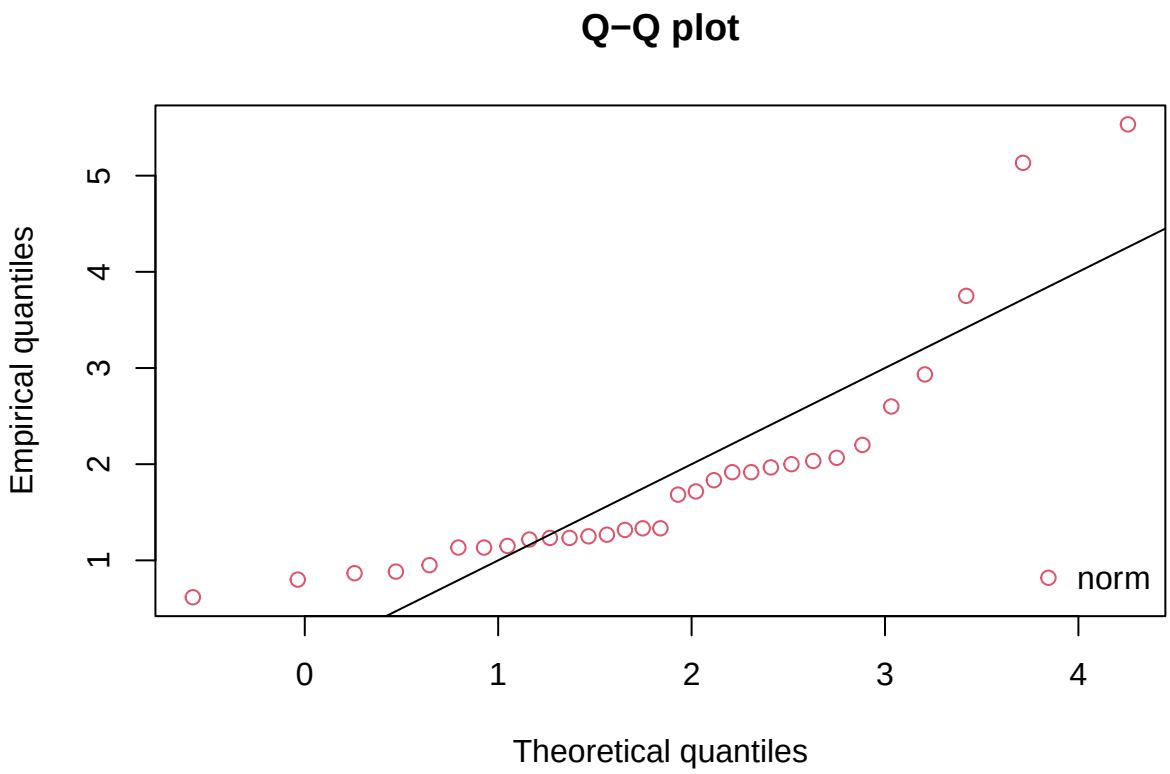

```
selfcheck_counter<-selfcheck_counter/60

#compare
exp_selfcheck <- fitdist(data=selfcheck_counter,distr="exp")
rate<-exp_selfcheck$estimate[1]

qqcomp(exp_selfcheck)
```



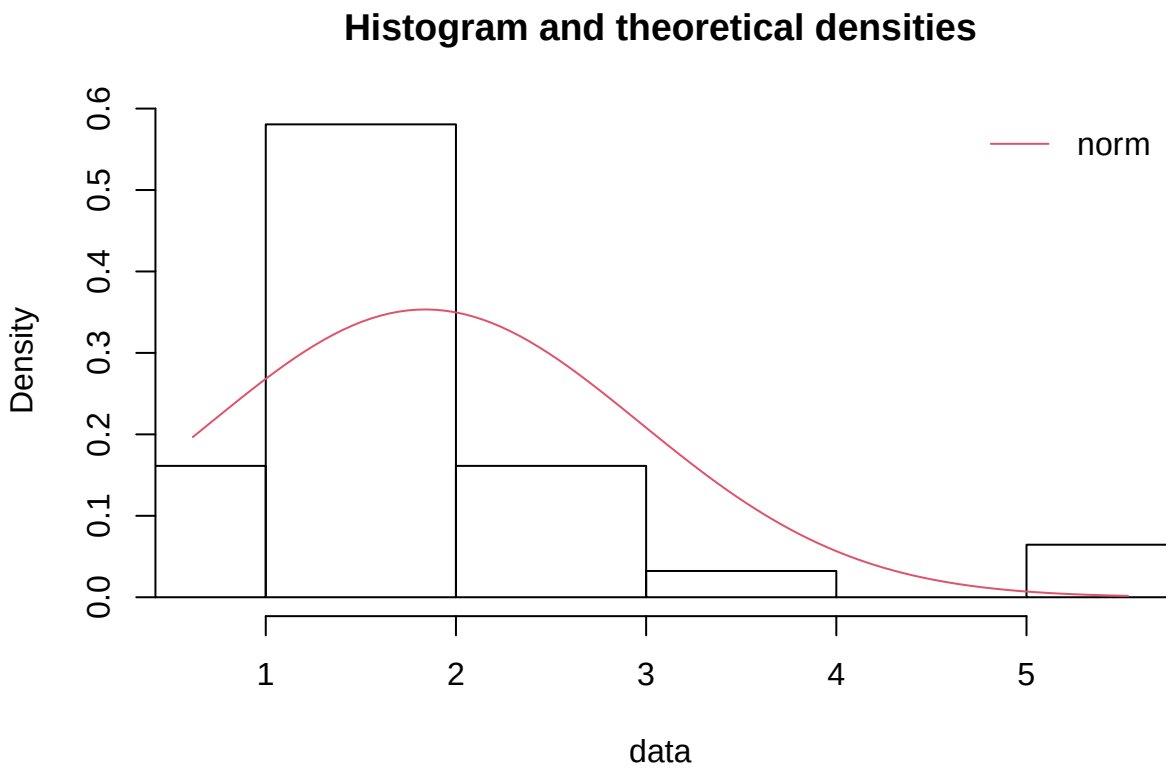

```
denscomp(exp_selfcheck)
```



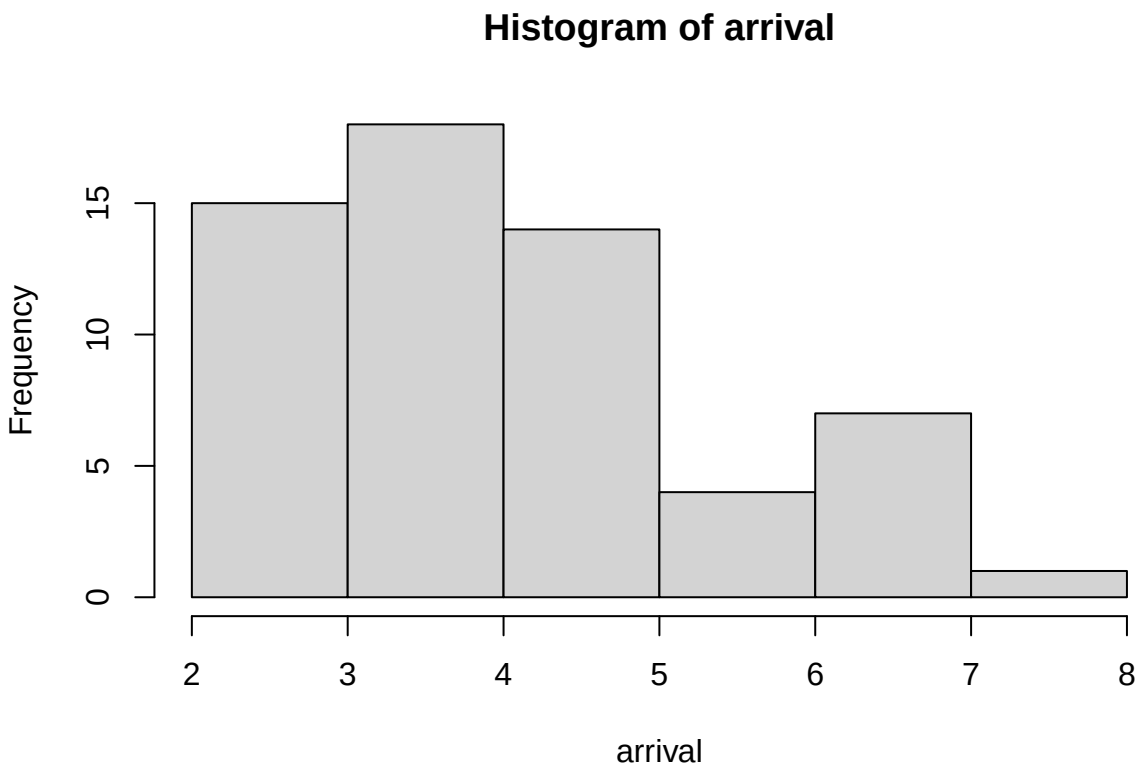

```
norm_selfcounter <- fitdist(data=selfcheck_counter,distr="norm")
mean1<-norm_selfcounter$estimate[1]
sd1<-norm_selfcounter$estimate[2]
qqcomp(norm_selfcounter)
```




```
denscomp(norm_selfcounter)
```




```
arrival<-c(3,4,7,3,4,3,4,5,3,3,5,7,4,3,5,4,3,4,5,4,5,4,3,2,5,4,3,4,5,3,5,7,4,4,5,4,5,5,7,3,2,5,4,7,3,5,

hist(arrival)
```




```
set.seed(1)
observed <- tabulate(arrival) #table when have x=0,
table(arrival)
```

```
## arrival
##  2  3  4  5  6  7  8
##  2 13 18 14  4  7  1
```

```
names(observed) <- 1:max(arrival)
observed
```

```
##  1  2  3  4  5  6  7  8
##  0  2 13 18 14  4  7  1
```

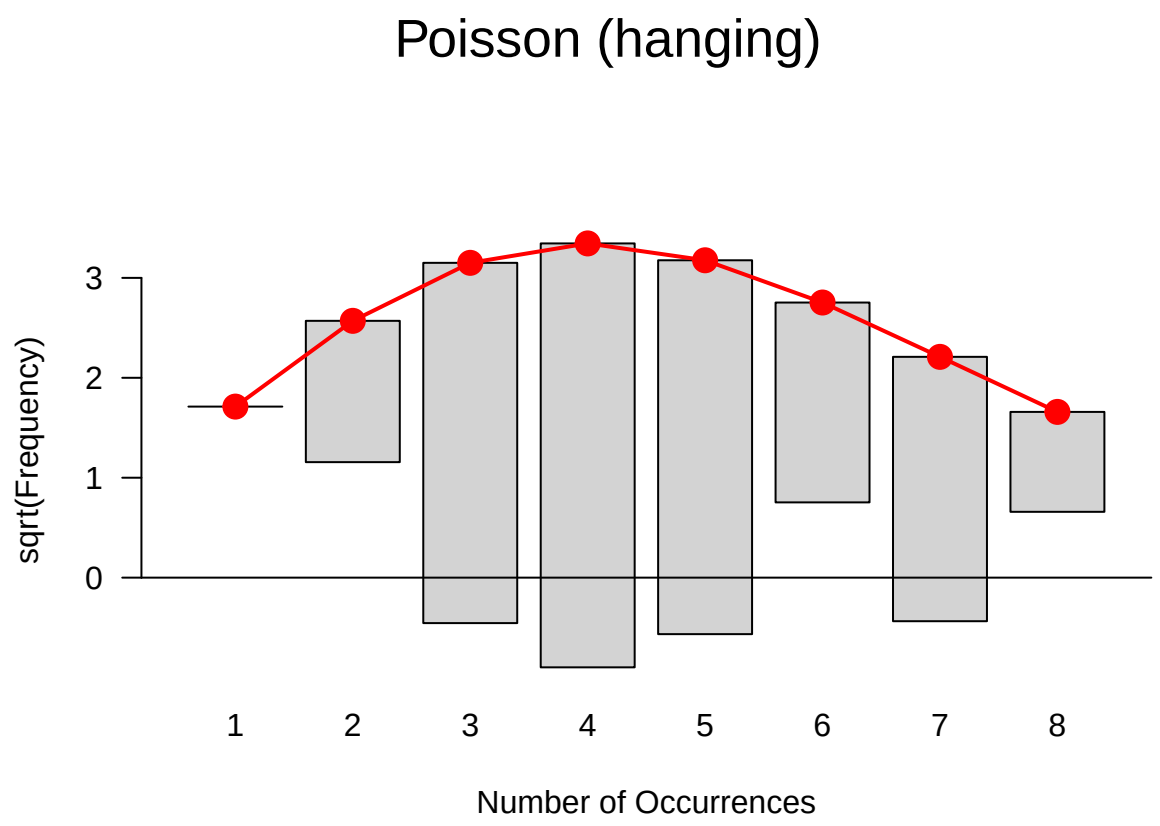
```
pois_par <- fitdist(data = arrival, # Use the dummy dataset
                                distr = "pois") # Select Poisson
(lambda <- pois_par$estimate[1]) # Store value of lambda as lambda
```

```
##   lambda
## 4.508475
```

```
fitted_pois <- dpois(x = 1:max(arrival),
                     lambda = lambda) * sum(observed)
library(vcd)
```

```
## Loading required package: grid
```

```
rootogram(x = observed,
          fitted = fitted_pois,
          type = "hanging",
          main = "Poisson (hanging)")
```



Step 2. We do a p-value test on the hypothesis h0 = The population mean of the two treatment groups are the same

```
set.seed(1)
df1 <- read.csv('../data/project.csv')
original <- mean(df1[1:31,2]) - mean(df1[32:62,2])
original
```

```
## [1] 0.788172
```

```
summary(df1)
```

```
##   treatment            outcome
##  Length:62          Min.   :0.4167
##  Class :character   1st Qu.:1.2542
##  Mode  :character   Median :1.9083
##                     Mean   :2.2328
##                     3rd Qu.:2.6625
##                     Max.   :6.7500
```

```r
treatment <- df1$treatment
treatment
```

```
##  [1] "normal" "normal" "normal" "normal" "normal" "normal" "normal" "normal"
##  [9] "normal" "normal" "normal" "normal" "normal" "normal" "normal" "normal"
## [17] "normal" "normal" "normal" "normal" "normal" "normal" "normal" "normal"
## [25] "normal" "normal" "normal" "normal" "normal" "normal" "normal" "self"
## [33] "self"   "self"   "self"   "self"   "self"   "self"   "self"   "self"
## [41] "self"   "self"   "self"   "self"   "self"   "self"   "self"   "self"
## [49] "self"   "self"   "self"   "self"   "self"   "self"   "self"   "self"
## [57] "self"   "self"   "self"   "self"   "self"   "self"
```

```r
outcome <- df1$outcome
outcome
```

```
##  [1] 1.4833333 2.1666667 1.8000000 1.8000000 6.7500000 1.5166667 1.9666667
##  [8] 3.4833333 3.5000000 2.0000000 2.5000000 0.7500000 2.6833333 4.5166667
## [15] 0.9833333 3.6333333 3.1333333 1.8166667 2.3333333 4.1500000 4.2166667
## [22] 1.3500000 1.0666667 2.2833333 0.4166667 2.6833333 1.7500000 1.9000000
## [29] 6.7500000 2.2166667 3.8333333 1.9166667 5.5333333 2.0333333 1.3333333
## [36] 3.7500000 1.9666667 0.8833333 2.0000000 5.1333333 1.2333333 1.2500000
## [43] 1.1333333 0.8666667 2.0666667 1.2333333 1.6833333 1.1500000 1.9166667
## [50] 1.2166667 1.8333333 2.6000000 0.8000000 0.9500000 0.6166667 1.3333333
## [57] 1.1333333 2.9333333 1.3166667 2.2000000 1.2666667 1.7166667
```

```r
set.seed(1)
permutation.test <- function(treatment, outcome){
    # Generate a permutation sample
    treatment_p <- sample(treatment, size= length(treatment), replace=FALSE)
    # Calculate the test statistic for the permutation sample
   mean(outcome[treatment_p == "normal"]) -
                          mean(outcome[treatment_p == "self"])
  }

# Slide 22: Use the "replicate()" Function to Run Multiple Simulations
test <- replicate(10000, permutation.test(treatment, outcome))
p_value<-mean(abs(test) >= abs(original))
p_value
```

```
## [1] 0.0244
```

```r
#since p-value is less than 0.05, we conclude that the population mean of the two treatment groups are
```

Step 3. Our group also went to use Confidence Interval to see whether the population mean of the two treatment groups are the same

```r
# mean, SD and sample size for normal group
x1 <- mean(df1[df1$treatment == "normal",2])
s1 <- sd(df1[df1$treatment == "normal",2])
n1 <- 31
```

```r
# mean, SD and sample size for self group
x2 <- mean(df1[df1$treatment == "self",2])
s2 <- sd(df1[df1$treatment == "self",2])
n2 <- 31

# point estimate = normal mean score - self mean score
x1 - x2
```

```
## [1] 0.788172
```

```r
# Slide 16: Standard Error

sp <- sqrt(((n1-1)*s1^2+(n2-1)*s2^2)/(n1+n2-2))
SE <- sp * sqrt(1/n1 + 1/n2)
SE
```

```
## [1] 0.3417635
```

```r
# Margin of error,qt is critical value of t-dist
moe <- qt(0.975,df=n1+n2-2) * SE
moe
```

```
## [1] 0.6836288
```

```r
# 95% Confidence interval = [point estimate - moe, point estimate + moe]
lowerCI <- (x1 - x2) - moe
upperCI <- (x1 - x2) + moe
c(lowerCI, upperCI)
```

```
## [1] 0.1045432 1.4718009
```

```r
t.test(outcome~treatment, alternative = "two.sided", paired=FALSE,
       var.equal=TRUE, data = df1)
```

```
##
##  Two Sample t-test
##
## data:  outcome by treatment
## t = 2.3062, df = 60, p-value = 0.02457
## alternative hypothesis: true difference in means between group normal and group self is not equal to
## 95 percent confidence interval:
##  0.1045432 1.4718009
## sample estimates:
## mean in group normal   mean in group self
##             2.626882             1.838710
```

```r
result1 <- t.test(outcome~treatment, alternative = "two.sided",
                  paired=FALSE, var.equal=TRUE, data = df1)
result1$conf.int
```

```
## [1] 0.1045432 1.4718009
## attr(,"conf.level")
## [1] 0.95
```

```
#since Confidence interval does not consist 0, we conclude that the population mean of the two treatmen
```

Step 4. After getting all the parameters, our group then use these parameters to build a simulation model.Our group has build 5 models 1.Base model,one queue to 10 normal counters 2.Base model,10 queue to 10 normal counters 3.10 queue to 10 self-checkout model 4.one queue to 10 mix model (trying to find the optimal number of normal and self counters) 5.10 queues to 2 normal and 8 self-checkout model.

```
library(simmer)
```

```
##
## Attaching package: 'simmer'

## The following object is masked from 'package:dplyr':
##
##     select

## The following object is masked from 'package:tidyr':
##
##     separate

## The following object is masked from 'package:MASS':
##
##     select
```

```
library(simmer.plot)
```

```
##
## Attaching package: 'simmer.plot'

## The following objects are masked from 'package:simmer':
##
##     get_mon_arrivals, get_mon_attributes, get_mon_resources
```

```
library(triangle)
library(scales)
```

```
##
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor
```

```
set.seed(1)
Type1_wanderer <- function() rtriangle(1, 5, 10,8 ) #rdy to eat meal
Type2_wanderer <- function() rtriangle(1, 10, 15, 13) #small purchase,immediate use
Type3_wanderer <- function() rtriangle(1, 15, 25, 20) #for trips/gathering
Type4_wanderer <- function() rtriangle(1, 25, 45, 30) #Stocking groceries
Type5_wanderer <- function() rtriangle(1, 5, 45, 20) #others
Task_duration_normal <- function() rnorm(n = 1,mean=mean,sd=sd)
rep<- function() {
Customer <-
  trajectory("Customer") %>%
          branch(function() sample(c(1,2,3,4,5), size=1, prob=c(0.15,0.37,0.11,0.35,0.02)),
              continue =FALSE,
              trajectory() %>%
                set_attribute("type",1) %>%
                timeout(Type1_wanderer()) %>%
                seize("Normal") %>%
                timeout(Task_duration_normal()) %>%
                release("Normal"),

              trajectory() %>%
                set_attribute("type",2) %>%
                timeout(Type2_wanderer()) %>%
                seize("Normal") %>%
                timeout(Task_duration_normal()) %>%
                release("Normal"),

              trajectory() %>%
                set_attribute("type",3) %>%
                timeout(Type3_wanderer()) %>%
                seize("Normal") %>%
                timeout(Task_duration_normal()) %>%
                release("Normal"),

              trajectory() %>%
                set_attribute("type",4) %>%
                timeout(Type4_wanderer()) %>%
                seize("Normal") %>%
                timeout(Task_duration_normal()) %>%
                release("Normal"),

              trajectory() %>%
                set_attribute("type",5) %>%
                timeout(Type5_wanderer()) %>%
                seize("Normal") %>%
                timeout(Task_duration_normal()) %>%
                release("Normal")

           )

plot(Customer)
NTUC <-
  simmer("NTUC") %>%
  add_resource("Normal", capacity = 10) %>%
```

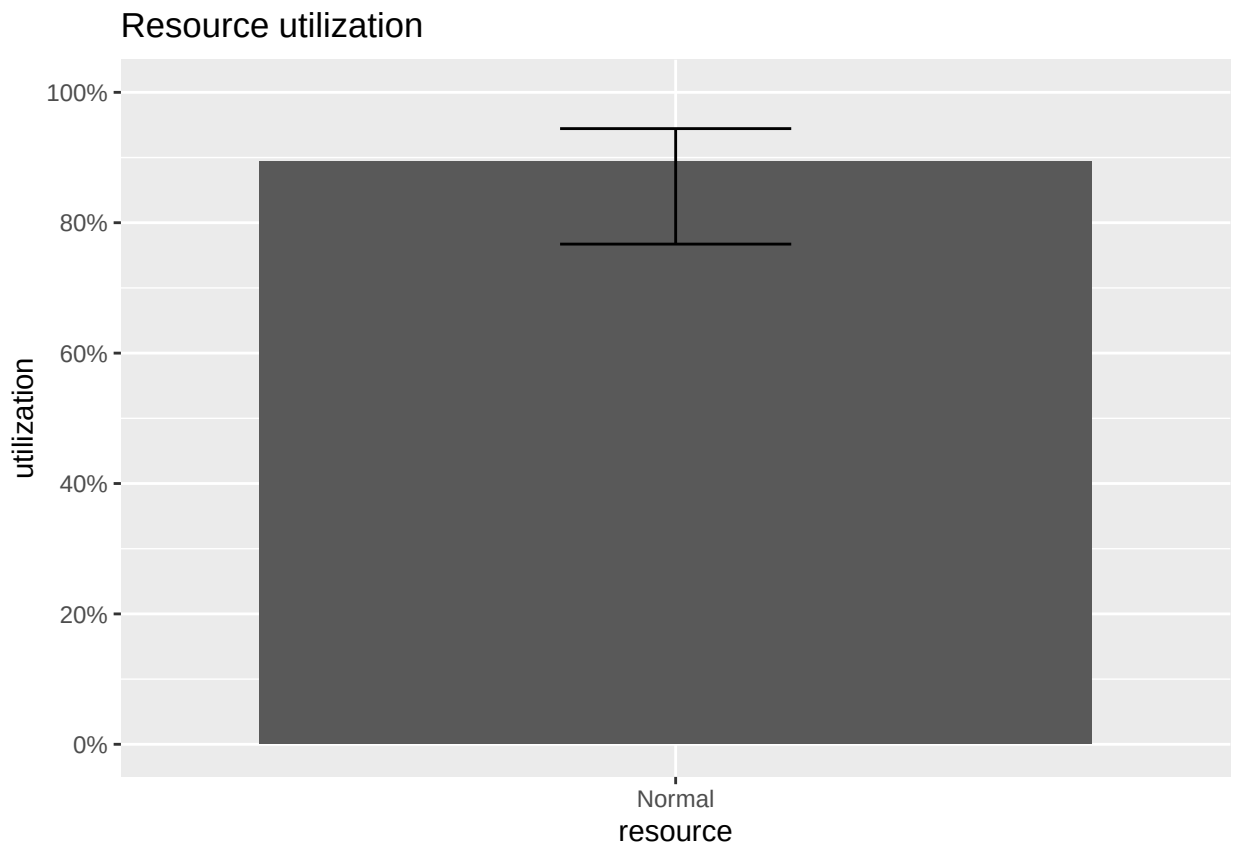
```
  add_generator("Customer", Customer,to(120,function() rexp(n = 1,rate=lambda)),
                mon = 2)%>%
  run %>% wrap() #120 minutes
}

set.seed(1)
env <- replicate(20, rep())

mon_arrivals <- get_mon_arrivals(env)
mon_resources<- get_mon_resources(env)

mon_arrivals %>%
  mutate(wait_time = end_time - start_time - activity_time)  %>%
  group_by(replication) %>%
  summarise(mean_wait_time = mean(wait_time)) %>%
  dplyr::select(mean_wait_time) %>%
  unlist() %>% as.vector() %>% mean
```

```
## [1] 10.43635
```

```
mon_resources %>%
  group_by(replication) %>%
  summarise(mean_queue_length = mean(queue)) %>%
  dplyr::select(mean_queue_length) %>%
  unlist() %>% as.vector() %>% mean
```

```
## [1] 32.43322
```

```
plot(mon_resources, metric="utilization")
```



Next we have Base model 10 queues to 10 normal counter

```
library(simmer)
library(simmer.plot)
library(triangle)
library(scales)
library(dplyr)
set.seed(1)
Type1_wanderer <- function() rtriangle(1, 5, 10,8 ) #rdy to eat meal
Type2_wanderer <- function() rtriangle(1, 10, 15, 13) #small purchase,immediate use
Type3_wanderer <- function() rtriangle(1, 15, 25, 20) #for trips/gathering
Type4_wanderer <- function() rtriangle(1, 25, 45, 30) #Stocking groceries
Type5_wanderer <- function() rtriangle(1, 5, 45, 20) #others
Task_duration_normal <- function() rnorm(n = 1,mean=mean,sd=sd)


rep<- function() {
Customer <-
  trajectory("Customer") %>%
          branch(function() sample(c(1,2,3,4,5), size=1, prob=c(0.15,0.37,0.11,0.35,0.02)),
              continue =FALSE,
              trajectory() %>%
                set_attribute("type",1) %>%
                timeout(Type1_wanderer()) %>%

                #seize("Normal") %>%
```

```
                simmer::select(c("Normal 1","Normal 2","Normal 3","Normal 4","Normal 5","Normal 6","Nor
                policy = "shortest-queue") %>%
                seize_selected() %>%
                timeout(Task_duration_normal()) %>%
                #release("Normal")%>%
                release_selected(),

              trajectory() %>%
                set_attribute("type",2) %>%
                timeout(Type2_wanderer()) %>%
                simmer::select(c("Normal 1","Normal 2","Normal 3","Normal 4","Normal 5","Normal 6","Nor
                policy = "shortest-queue") %>%
                seize_selected() %>%
                timeout(Task_duration_normal()) %>%
                #release("Normal")%>%
                release_selected(),

              trajectory() %>%
                set_attribute("type",3) %>%
                timeout(Type3_wanderer()) %>%
                simmer::select(c("Normal 1","Normal 2","Normal 3","Normal 4","Normal 5","Normal 6","Nor
                policy = "shortest-queue") %>%
                seize_selected() %>%
                timeout(Task_duration_normal()) %>%
                #release("Normal")%>%
                release_selected(),

              trajectory() %>%
                set_attribute("type",4) %>%
                timeout(Type4_wanderer()) %>%
                simmer::select(c("Normal 1","Normal 2","Normal 3","Normal 4","Normal 5","Normal 6","Nor
                policy = "shortest-queue") %>%
                seize_selected() %>%
                timeout(Task_duration_normal()) %>%
                #release("Normal")%>%
                release_selected(),

              trajectory() %>%
                set_attribute("type",5) %>%
                timeout(Type5_wanderer()) %>%
                simmer::select(c("Normal 1","Normal 2","Normal 3","Normal 4","Normal 5","Normal 6","Nor
                policy = "shortest-queue") %>%
                seize_selected() %>%
                timeout(Task_duration_normal()) %>%
                #release("Normal")%>%
                release_selected()

            )

plot(Customer)
NTUC <-
  simmer("NTUC") %>%
  add_resource("Normal 1", capacity = 1) %>%
```

```
  add_resource("Normal 2", capacity = 1) %>%
  add_resource("Normal 3", capacity = 1) %>%
  add_resource("Normal 4", capacity = 1) %>%
  add_resource("Normal 5", capacity = 1) %>%
  add_resource("Normal 6", capacity = 1) %>%
  add_resource("Normal 7", capacity = 1) %>%
  add_resource("Normal 8", capacity = 1) %>%
  add_resource("Normal 9", capacity = 1) %>%
  add_resource("Normal 10", capacity = 1) %>%
  #add_resource("Normal", capacity = 10) %>%
  #add_generator("Customer", Customer,function() rpois(n = 1,lambda = poi_arrival$estimate[1]),
                #mon = 2)%>%
  add_generator("Customer", Customer,to(120,function() rexp(n = 1,rate=lambda)),
                mon = 2)%>%
  run %>% wrap() #120 minutes
}

set.seed(1)
env <- replicate(20, rep())
mon_arrivals <- get_mon_arrivals(env)
mon_resources<- get_mon_resources(env)

mon_arrivals %>%
  mutate(wait_time = end_time - start_time - activity_time)  %>%
  group_by(replication) %>%
  summarise(mean_wait_time = mean(wait_time)) %>%
  dplyr::select(mean_wait_time) %>%
  unlist() %>% as.vector() %>% mean
```

```
## [1] 10.71679
```

```
mon_resources %>%
  group_by(replication) %>%
  summarise(mean_queue_length = mean(queue)) %>%
  dplyr::select(mean_queue_length) %>%
  unlist() %>% as.vector() %>% mean
```

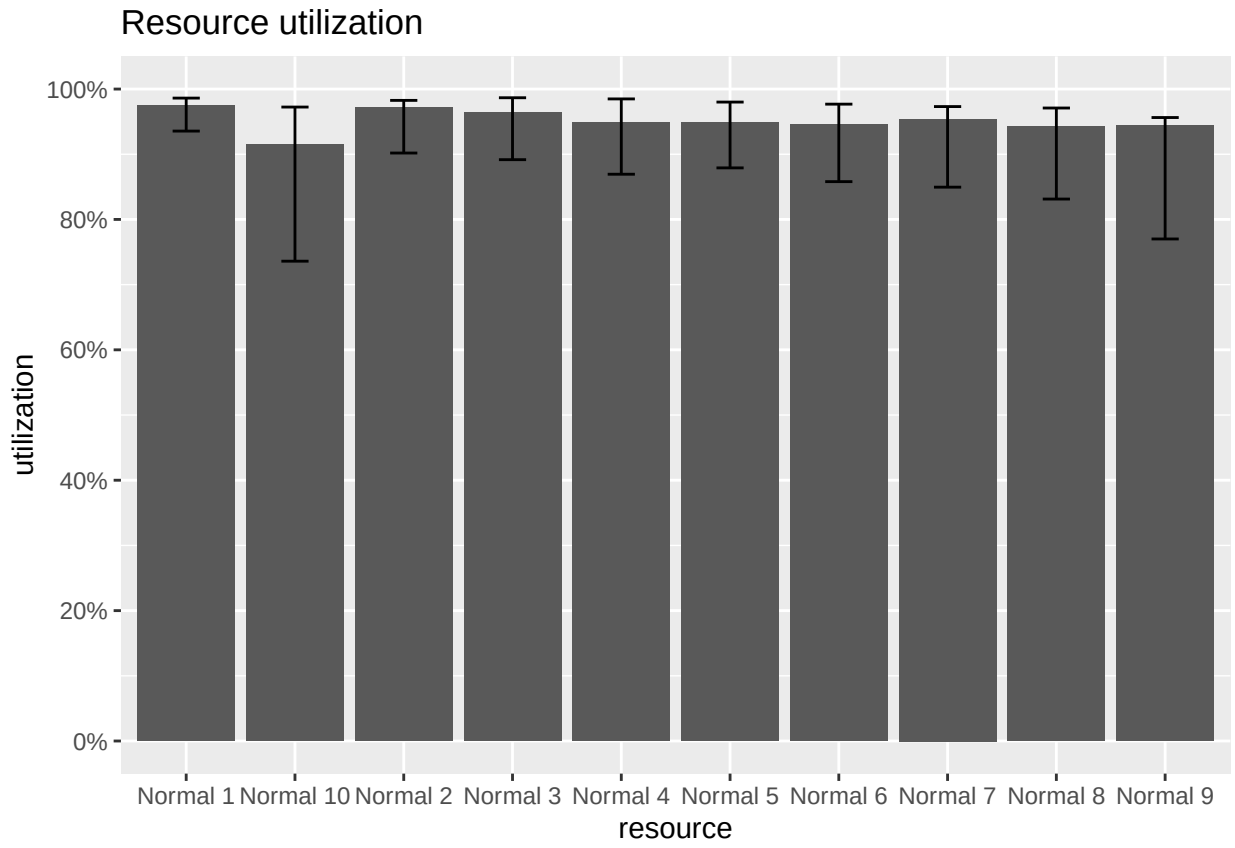
```
## [1] 3.175041
```

```
plot1 <- plot(mon_resources, items="queue")
df_resource <- plot1$data
df_resource %>% group_by(resource, replication) %>%
                summarise(mean_queue = tail(mean, n=1)) %>%
                group_by(resource) %>%
                summarise(mean_queue = mean(mean_queue))
```

```
## `summarise()` has grouped output by 'resource'. You can override using the
## `.groups` argument.

## # A tibble: 10 x 2
##    resource  mean_queue
```

```
##    <chr>          <dbl>
##  1 Normal 1        3.33
##  2 Normal 10       2.76
##  3 Normal 2        3.21
##  4 Normal 3        3.13
##  5 Normal 4        3.10
##  6 Normal 5        3.01
##  7 Normal 6        2.97
##  8 Normal 7        2.92
##  9 Normal 8        2.86
## 10 Normal 9        2.79
```

```
plot(mon_resources, metric="utilization")
```




```
#conclude model 1 and model 2 roughly the same as the total mean waiting time and total mean queue leng
```

Next we have the 1 queue to 10 self-check out model, similar to real life where self-checkout counters usually have one queue only.

```
library(simmer)
library(simmer.plot)
library(triangle)
library(scales)
set.seed(1)
Type1_wanderer <- function() rtriangle(1, 5, 10,8 ) #rdy to eat meal
```

```
Type2_wanderer <- function() rtriangle(1, 10, 15, 13) #small purchase,immediate use
Type3_wanderer <- function() rtriangle(1, 15, 25, 20) #for trips/gathering
Type4_wanderer <- function() rtriangle(1, 25, 45, 30) #Stocking groceries
Type5_wanderer <- function() rtriangle(1, 5, 45, 20) #others
Task_duration_normal <- function() rnorm(n = 1,mean=mean,sd=sd)
Task_duration_self <- function() rnorm(n = 1,mean=mean1,sd=sd1)

rep2<-function(){
Customer <-
  trajectory("Customer") %>%
          branch(function() sample(c(1,2,3,4,5), size=1, prob=c(0.15,0.37,0.11,0.35,0.02)),
              continue =FALSE,
              trajectory() %>%
                set_attribute("type",1) %>%
                timeout(Type1_wanderer()) %>%
                seize("Self") %>%
                timeout(Task_duration_self()) %>%
                release("Self"),

              trajectory() %>%
                set_attribute("type",2) %>%
                timeout(Type2_wanderer()) %>%
                seize("Self") %>%
                timeout(Task_duration_self()) %>%
                release("Self"),

              trajectory() %>%
                set_attribute("type",3) %>%
                timeout(Type3_wanderer()) %>%
                seize("Self") %>%
                timeout(Task_duration_self()) %>%
                release("Self"),

              trajectory() %>%
                set_attribute("type",4) %>%
                timeout(Type4_wanderer()) %>%
                seize("Self") %>%
                timeout(Task_duration_self()) %>%
                release("Self"),

              trajectory() %>%
                set_attribute("type",5) %>%
                timeout(Type5_wanderer()) %>%
                seize("Self") %>%
                timeout(Task_duration_self()) %>%
                release("Self")

           )

plot(Customer)
NTUC <-
  simmer("NTUC") %>%
  add_resource("Self", capacity = 10) %>%
```

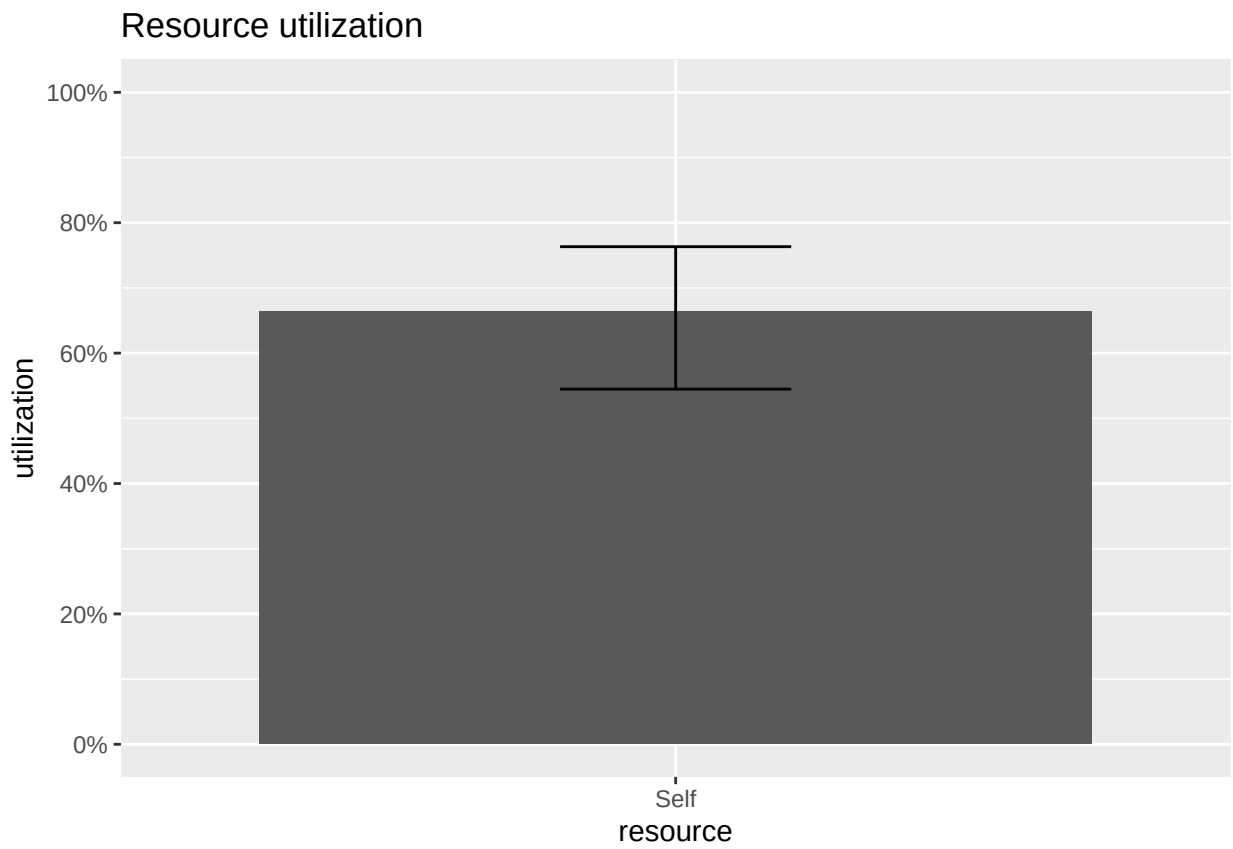
```
    add_generator("Customer", Customer,to(120,function() rexp(n = 1,rate=lambda)),
                  mon = 2)%>%
  run %>% wrap() #120 minutes
}

set.seed(1)
env1 <- replicate(20, rep2())

mon_arrivals <- get_mon_arrivals(env1)
mon_resources<- get_mon_resources(env1)

mon_arrivals %>%
  mutate(wait_time = end_time - start_time - activity_time)  %>%
  group_by(replication) %>%
  summarise(mean_wait_time = mean(wait_time)) %>%
  dplyr::select(mean_wait_time) %>%
  unlist() %>% as.vector() %>% mean
```

```
## [1] 2.02372
```

```
mon_resources %>%
  group_by(replication) %>%
  summarise(mean_queue_length = mean(queue)) %>%
  dplyr::select(mean_queue_length) %>%
  unlist() %>% as.vector() %>% mean
```

```
## [1] 7.427711
```

```
plot(mon_resources, metric="utilization")
```



Next, we have mix model trying to find out optimal number of normal and self-checkout counters, using sapply function. Where this is one queue to many normal counters and one queue to many self-check out counters.

```
library(simmer)
library(simmer.plot)
library(triangle)
library(scales)
library(tidyverse)
set.seed(1)
Type1_wanderer <- function() rtriangle(1, 5, 10,8 ) #rdy to eat meal
Type2_wanderer <- function() rtriangle(1, 10, 15, 13) #small purchase,immediate use
Type3_wanderer <- function() rtriangle(1, 15, 25, 20) #for trips/gathering
Type4_wanderer <- function() rtriangle(1, 25, 45, 30) #Stocking groceries
Type5_wanderer <- function() rtriangle(1, 5, 45, 20) #others
Task_duration_normal <- function() rnorm(n = 1,mean=mean,sd=sd)
Task_duration_self <- function() rnorm(n = 1,mean=mean1,sd=sd1)
rep3<- function(cap) {
Customer <-
  trajectory("Customer") %>%
          branch(function() sample(c(1,2,3,4,5), size=1, prob=c(0.15,0.37,0.11,0.35,0.02)),
              continue =FALSE,
              trajectory() %>%
                set_attribute("type",1) %>%
                timeout(Type1_wanderer()) %>%
                branch(function() sample(c(1,2), size=1, prob=c(cap/10,(10-cap)/10)),
```

```
            continue = FALSE,
            trajectory() %>%
              seize("Normal") %>%
              timeout(Task_duration_normal()) %>%
              release("Normal"),
            trajectory() %>%
            seize("Self") %>%
              timeout(Task_duration_self()) %>%
              release("Self")),

            trajectory() %>%
              set_attribute("type",2) %>%
              timeout(Type2_wanderer()) %>%
              branch(function() sample(c(1,2), size=1, prob=c(cap/10,(10-cap)/10)),
            continue = FALSE,
            trajectory() %>%
              seize("Normal") %>%
              timeout(Task_duration_normal()) %>%
              release("Normal"),
            trajectory() %>%
            seize("Self") %>%
              timeout(Task_duration_self()) %>%
              release("Self")),

            trajectory() %>%
              set_attribute("type",3) %>%
              timeout(Type3_wanderer()) %>%
              branch(function() sample(c(1,2), size=1, prob=c(cap/10,(10-cap)/10)),
            continue = FALSE,
            trajectory() %>%
              seize("Normal") %>%
              timeout(Task_duration_normal()) %>%
              release("Normal"),
            trajectory() %>%
            seize("Self") %>%
              timeout(Task_duration_self()) %>%
              release("Self")),

            trajectory() %>%
              set_attribute("type",4) %>%
              timeout(Type4_wanderer()) %>%
              branch(function() sample(c(1,2), size=1, prob=c(cap/10,(10-cap)/10)),
            continue = FALSE,
            trajectory() %>%
              seize("Normal") %>%
              timeout(Task_duration_normal()) %>%
              release("Normal"),
            trajectory() %>%
            seize("Self") %>%
              timeout(Task_duration_self()) %>%
              release("Self")),

            trajectory() %>%
```

```
                set_attribute("type",5) %>%
                timeout(Type5_wanderer()) %>%
                branch(function() sample(c(1,2), size=1, prob=c(cap/10,(10-cap)/10)),
              continue = FALSE,
              trajectory() %>%
                seize("Normal") %>%
                timeout(Task_duration_normal()) %>%
                release("Normal"),
              trajectory() %>%
              seize("Self") %>%
                timeout(Task_duration_self()) %>%
                release("Self"))

            )

plot(Customer)
NTUC <-
  simmer("NTUC") %>%
  add_resource("Normal", capacity = cap) %>%
  add_resource("Self", capacity = 10-cap) %>%
  add_generator("Customer", Customer,to(120,function() rexp(n = 1,rate=lambda)),
                mon = 2)%>%
  run %>% wrap() #120 minutes
}

set.seed(1)
sapply(1:9, function(cap) {
  env3 <- replicate(20, rep3(cap))
  mon_arrivals <- get_mon_arrivals(env3)
  mon_resources<- get_mon_resources(env3)

  mon_arrivals %>%
    mutate(wait_time = end_time - start_time - activity_time)  %>%
    group_by(replication) %>%
    summarise(mean_wait_time = mean(wait_time)) %>%
    dplyr::select(mean_wait_time) %>%
    unlist() %>% as.vector() %>% mean  -> mean_wait_time

  mon_resources %>%
  group_by(replication) %>%
  summarise(mean_queue_length = mean(queue)) %>%
  dplyr::select(mean_queue_length) %>%
  unlist() %>% as.vector() %>% mean -> mean_queue_length


  paste("The mean waiting time and mean queue length with ", cap, " normal counter and ",10-cap,"self-ch
})
```

```
## [1] "The mean waiting time and mean queue length with  1  normal counter and  9 self-check out counter
##  is  3.45323412716645 4.25225435901879"
## [2] "The mean waiting time and mean queue length with  2  normal counter and  8 self-check out counter
##  is  3.12291693271152 4.89757932319908"
## [3] "The mean waiting time and mean queue length with  3  normal counter and  7 self-check out counter
##  is  6.17778476092814 9.57616915485401"
## [4] "The mean waiting time and mean queue length with  4  normal counter and  6 self-check out counter
##  is  9.80895584645454 14.2246539363581"
## [5] "The mean waiting time and mean queue length with  5  normal counter and  5 self-check out counter
##  is  5.91799749842319 10.4969193897989"
## [6] "The mean waiting time and mean queue length with  6  normal counter and  4 self-check out counter
##  is  6.7060304073214 13.3114906635442"
## [7] "The mean waiting time and mean queue length with  7  normal counter and  3 self-check out counter
##  is  12.5141979217209 23.4569787293919"
## [8] "The mean waiting time and mean queue length with  8  normal counter and  2 self-check out counter
##  is  10.6611341681874 25.916435577032"
## [9] "The mean waiting time and mean queue length with  9  normal counter and  1 self-check out counter
##  is  8.22385938937766 22.2069874822834"
```

```
## [6] "The mean waiting time and mean queue length with  6  normal counter and  4 self-check out counte
## [7] "The mean waiting time and mean queue length with  7  normal counter and  3 self-check out counte
## [8] "The mean waiting time and mean queue length with  8  normal counter and  2 self-check out counte
## [9] "The mean waiting time and mean queue length with  9  normal counter and  1 self-check out counte
```

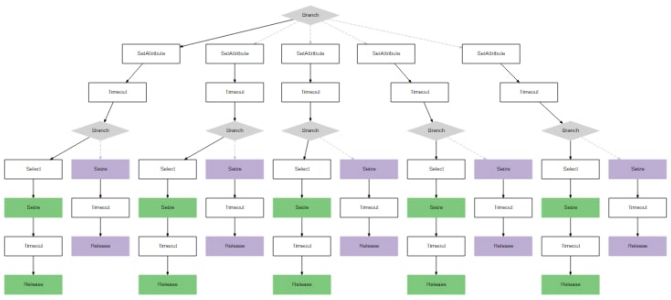
```
#conclude, under mix model, 2 normal and 8 self-checkout would be appropriate, as it has one of the low
```

Lastly,from the previous model we conclude that the optimal solution would be 1 queue to 8 self-checkout counters and 2 queues to 2 normal counters.As per real life, where usually have only 1 queue for self-checkout counters.

```
library(simmer)
library(simmer.plot)
library(triangle)
library(scales)
set.seed(1)
Type1_wanderer <- function() rtriangle(1, 5, 10,8 ) #rdy to eat meal
Type2_wanderer <- function() rtriangle(1, 10, 15, 13) #small purchase,immediate use
Type3_wanderer <- function() rtriangle(1, 15, 25, 20) #for trips/gathering
Type4_wanderer <- function() rtriangle(1, 25, 45, 30) #Stocking groceries
Type5_wanderer <- function() rtriangle(1, 5, 45, 20) #others
Task_duration_normal <- function() rnorm(n = 1,mean=mean,sd=sd)
Task_duration_self <- function() rnorm(n = 1,mean=mean1,sd=sd1)

rep3<- function() {
Customer <-
  trajectory("Customer") %>%
          branch(function() sample(c(1,2,3,4,5), size=1, prob=c(0.15,0.37,0.11,0.35,0.02)),
              continue =FALSE,
              trajectory() %>%
                set_attribute("type",1) %>%
                timeout(Type1_wanderer()) %>%
                branch(function() sample(c(1,2), size=1, prob=c(0.2,0.8)),
              continue = FALSE,
              trajectory() %>%
               simmer::select(c("Normal 1","Normal 2"),
                policy = "shortest-queue") %>%
                seize_selected() %>%
                timeout(Task_duration_normal()) %>%
                release_selected(),
              trajectory() %>%
              seize("Self") %>%
                timeout(Task_duration_self()) %>%
                release("Self")),

              trajectory() %>%
                set_attribute("type",2) %>%
                timeout(Type2_wanderer()) %>%
                branch(function() sample(c(1,2), size=1, prob=c(0.2,0.8)),
              continue = FALSE,
              trajectory() %>%
                simmer::select(c("Normal 1","Normal 2"),
                policy = "shortest-queue") %>%
```

```
    seize_selected() %>%
    timeout(Task_duration_normal()) %>%
    release_selected(),
  trajectory() %>%
  seize("Self") %>%
    timeout(Task_duration_self()) %>%
    release("Self")),

  trajectory() %>%
    set_attribute("type",3) %>%
    timeout(Type3_wanderer()) %>%
    branch(function() sample(c(1,2), size=1, prob=c(0.2,0.8)),
  continue = FALSE,
  trajectory() %>%
    simmer::select(c("Normal 1","Normal 2"),
    policy = "shortest-queue") %>%
    seize_selected() %>%
    timeout(Task_duration_normal()) %>%
    release_selected(),
  trajectory() %>%
  seize("Self") %>%
    timeout(Task_duration_self()) %>%
    release("Self")),

  trajectory() %>%
    set_attribute("type",4) %>%
    timeout(Type4_wanderer()) %>%
    branch(function() sample(c(1,2), size=1, prob=c(0.2,0.8)),
  continue = FALSE,
  trajectory() %>%
    simmer::select(c("Normal 1","Normal 2"),
    policy = "shortest-queue") %>%
    seize_selected() %>%
    timeout(Task_duration_normal()) %>%
    release_selected(),
  trajectory() %>%
  seize("Self") %>%
    timeout(Task_duration_self()) %>%
    release("Self")),

  trajectory() %>%
    set_attribute("type",5) %>%
    timeout(Type5_wanderer()) %>%
    branch(function() sample(c(1,2), size=1, prob=c(0.2,0.8)),
  continue = FALSE,
  trajectory() %>%
    simmer::select(c("Normal 1","Normal 2"),
    policy = "shortest-queue") %>%
    seize_selected() %>%
    timeout(Task_duration_normal()) %>%
    release_selected(),
  trajectory() %>%
  seize("Self") %>%
```

```
                timeout(Task_duration_self()) %>%
                release("Self"))

            )

plot(Customer)
NTUC <-
  simmer("NTUC") %>%
  add_resource("Self", capacity = 8) %>%
  add_resource("Normal 1", capacity = 1) %>%
  add_resource("Normal 2", capacity = 1) %>%

  add_generator("Customer", Customer,to(120,function() rexp(n = 1,rate=lambda)),
                mon = 2)%>%
  run %>% wrap() #120 minutes
}

#base on the results above, we conclude that 5 normal and 5 mix would be the best for the NTUC as it ha
set.seed(1)
env3 <- replicate(20, rep3())
  mon_arrivals <- get_mon_arrivals(env3)
  mon_resources<- get_mon_resources(env3)

  mon_arrivals %>%
    mutate(wait_time = end_time - start_time - activity_time)  %>%
    group_by(replication) %>%
    summarise(mean_wait_time = mean(wait_time)) %>%
    dplyr::select(mean_wait_time) %>%
    unlist() %>% as.vector() %>% mean  -> mean_wait_time

  mon_resources %>%
  group_by(replication) %>%
  summarise(mean_queue_length = mean(queue)) %>%
  dplyr::select(mean_queue_length) %>%
  unlist() %>% as.vector() %>% mean -> mean_queue_length


mean_wait_time
```

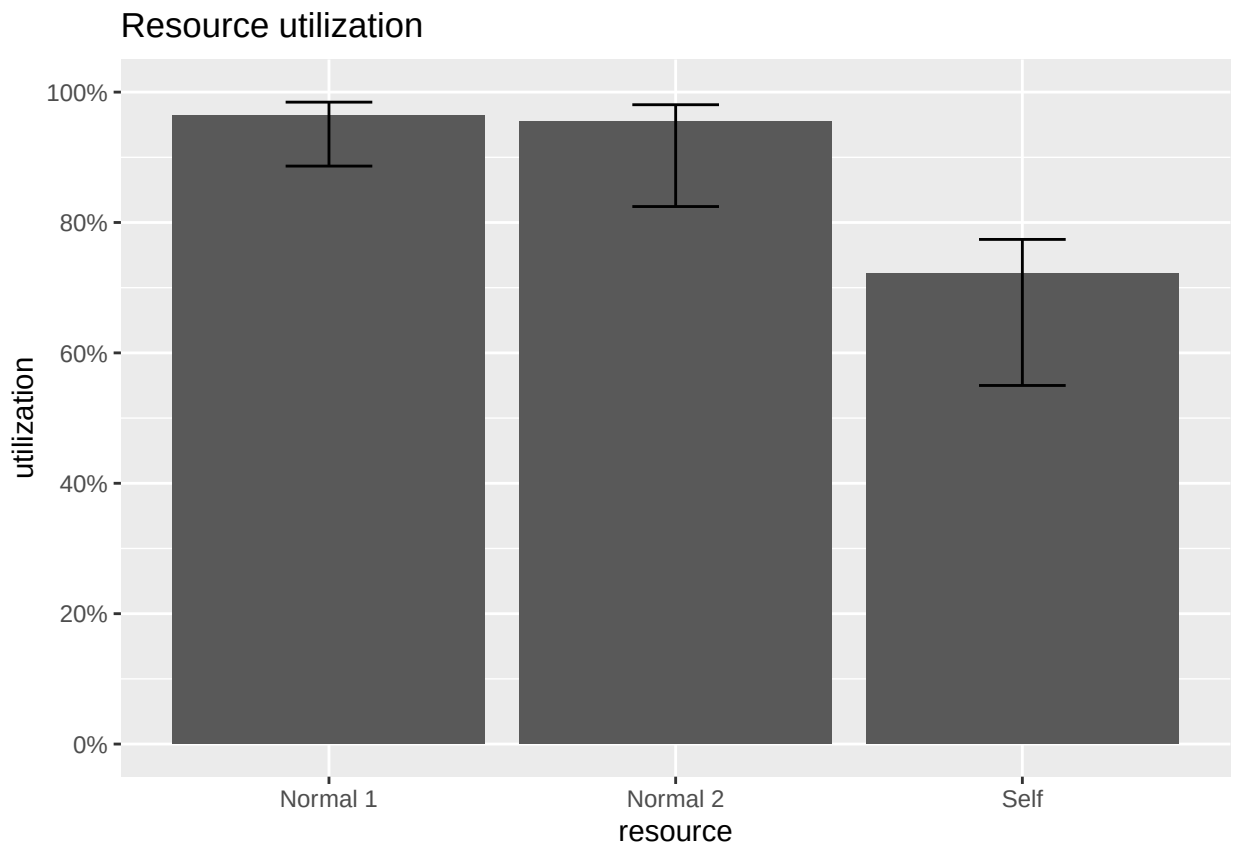


```
## [1] 5.181698
```

```
mean_queue_length
```

```
## [1] 4.119101
```

```
plot(mon_resources, metric="utilization")
```



Conclusion:

Even though 10 self-check out counter has the lowest mean waiting time and mean queue length.Some elderly who are not familiar with self-check out system would still prefer having normal counters.Moreover, some people that buys a lot of food would maybe prefer someone to help them pack.Therefore, still prefer normal counter.Therefore, our group do not recommend fully self-check out system.Thus,our group concluded that 8 self-checkout counters and 2 normal counters would be optimal as it has one of the lowest mean waiting time and lowest mean queue length. While also satisfy those customers that still prefer the normal counters.

model 5(2 queues to 2 normal counters and 1 queue to 8 self-checkout): mean waiting time = 5.18, mean queue length = 4.12, total queue in system=4.12X3=12.36 (multiple 3 because 3 queues)

Base model 2 (10 queues to 10 normal counters) : mean waiting time = 10.72 min, mean queue length = 3.18, total queue in system= 3.18X10=31.8(multiple 10 because 10 queues)

Therefore, mean waiting time has improved around 50% and around 60% for the total queue length in system. Which is a good improvement. :)