Intro to Analytics Modeling HW 6

2024-06-26

Question 13.2

Please note that this will be available at the end of the document. I used SimPy in Python and needed to add it after I've completed the other questions in R.

Importing Libraries

```
library(DAAG)
```

Read Data

```
breast_cancer_data <- read.table("~/Desktop/ISYE-6501/week 6 Homework-Summer24/breast-cancer-wisconsin.
breast_cancer_df <- data.frame(breast_cancer_data)
# convert "?" to NA
breast_cancer_df[breast_cancer_df == "?"] <- NA
head(breast_cancer_df)
```

```
##   X1000025 X5 X1 X1.1 X1.2 X2 X1.3 X3 X1.4 X1.5 X2.1
## 1  1002945  5  4    4    5  7   10  3    2    1    2
## 2  1015425  3  1    1    1  2    2  3    1    1    2
## 3  1016277  6  8    8    1  3    4  3    7    1    2
## 4  1017023  4  1    1    3  2    1  3    1    1    2
## 5  1017122  8 10   10    8  7   10  9    7    1    4
## 6  1018099  1  1    1    1  2   10  3    1    1    2
```

Data Cleansing

```
breast_cancer_df$X1.3 <- as.numeric(breast_cancer_df$X1.3)
```

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/ (description at http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29) has missing values.

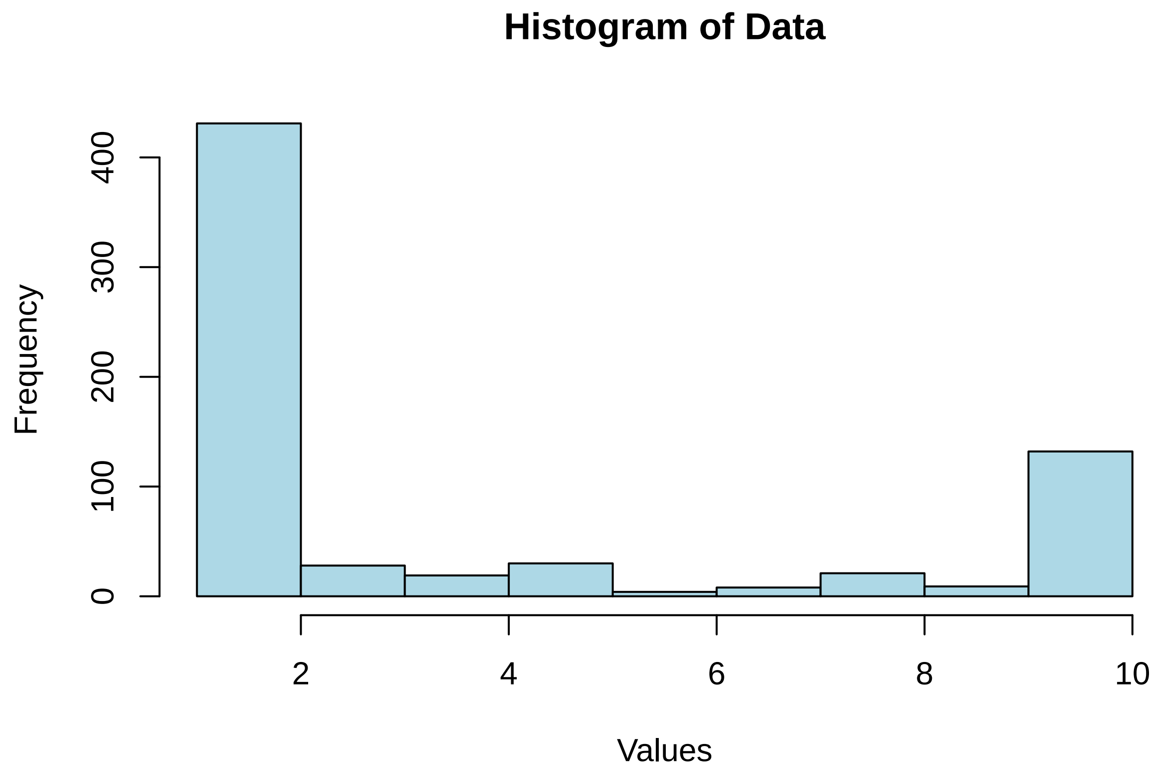
check each column if there's any missing values

```
cols <- colnames(breast_cancer_df)[2:11]
for (col in cols) {
  if (any(is.na(breast_cancer_df[[col]]))) {
    print(paste(col, "has at least one missing value"))
  }
}
```

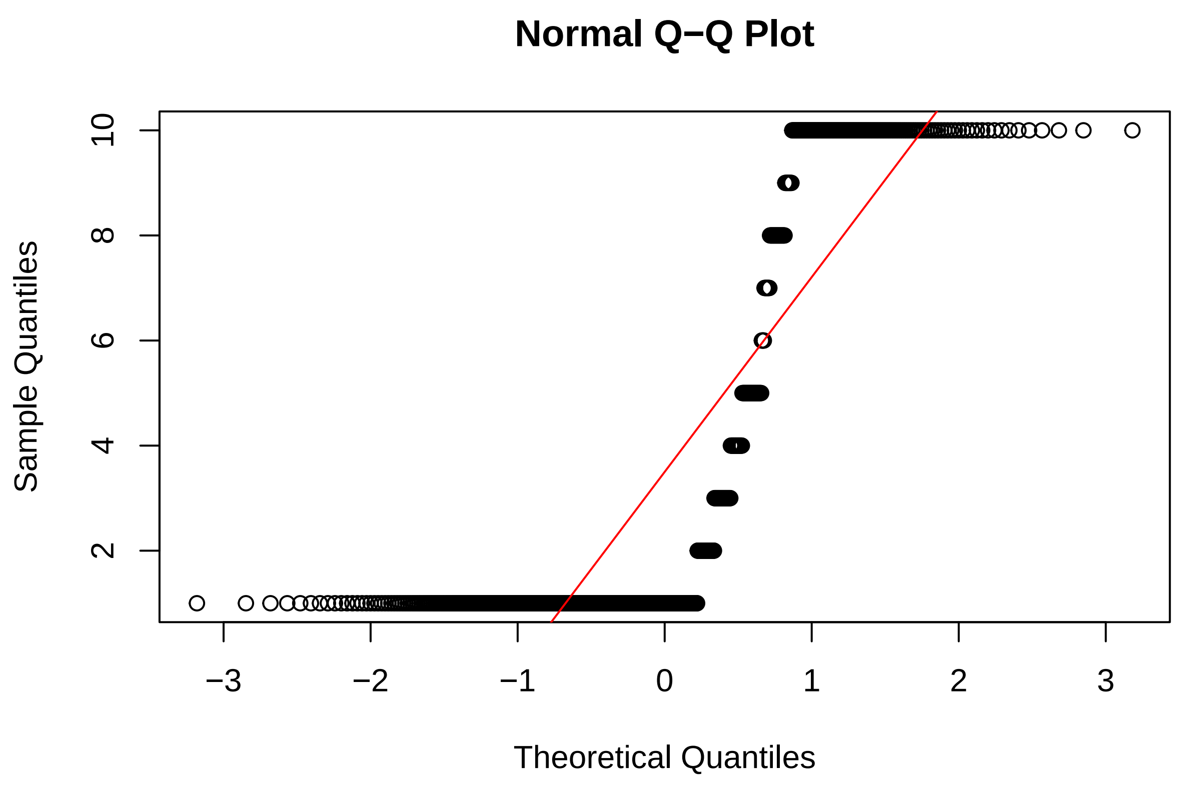
```
## [1] "X1.3 has at least one missing value"
```

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

```
# Create a histogram
hist(breast_cancer_df$X1.3, main = "Histogram of Data", xlab = "Values", ylab = "Frequency", col = "lig
```



```
qqnorm(breast_cancer_df$X1.3)
qqline(breast_cancer_df$X1.3, col = "red")
```



```r
unique(breast_cancer_df$X1.3)
```

```
##  [1] 10  2  4  1  3  9  7 NA  5  8  6
```

Observation: To determine which statistic would be better between mean and mode, I plotted a histogram, a Q-Q plot, and checked the data values to determine its type (numerical vs. categorical) and assess its normality in case it's numerical data. Based on the visualization outcomes and checking unique values, it appears that the column represents categorical data, making mode a more appropriate choice.

```r
# Get the mode and replace NA with the mode
get_mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

breast_cancer_df_mode <- breast_cancer_df
breast_cancer_df_mode$X1.3 <- as.integer(breast_cancer_df_mode$X1.3)

x1.3_mode <- get_mode(breast_cancer_df_mode$X1.3)
print(x1.3_mode)
```

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

```r
# replace NA with the mode (=1)
breast_cancer_df_mode$X1.3[is.na(breast_cancer_df_mode$X1.3)] <- x1.3_mode
```

```r
# Check if imputation using mode (=1) works fine.
cols <- colnames(breast_cancer_df_mode)[2:11]
for (col in cols) {
  if (any(is.na(breast_cancer_df_mode[[col]]))) {
    print(paste(col, "has at least one missing value"))
  } else {
    print(paste(col, "has no missing values"))
  }
}
```

```
## [1] "X5 has no missing values"
## [1] "X1 has no missing values"
## [1] "X1.1 has no missing values"
## [1] "X1.2 has no missing values"
## [1] "X2 has no missing values"
## [1] "X1.3 has no missing values"
## [1] "X3 has no missing values"
## [1] "X1.4 has no missing values"
## [1] "X1.5 has no missing values"
## [1] "X2.1 has no missing values"
```

```r
# (Optional) Get the mean and replace NA with the mode if desired.
breast_cancer_df_mean <- breast_cancer_df
breast_cancer_df_mean$X1.3 <- as.numeric(breast_cancer_df_mean$X1.3)
x1.3_mean <- mean(breast_cancer_df_mean$X1.3, na.rm = TRUE)
print(x1.3_mean)
```

```
## [1] 3.548387
```

```r
# replace NA with the mean
breast_cancer_df_mean$X1.3[is.na(breast_cancer_df_mean$X1.3)] <- x1.3_mean
head(breast_cancer_df_mean)
```

```
##   X1000025 X5 X1 X1.1 X1.2 X2 X1.3 X3 X1.4 X1.5 X2.1
## 1  1002945  5  4    4    5  7   10  3    2    1    2
## 2  1015425  3  1    1    1  2    2  3    1    1    2
## 3  1016277  6  8    8    1  3    4  3    7    1    2
## 4  1017023  4  1    1    3  2    1  3    1    1    2
## 5  1017122  8 10   10    8  7   10  9    7    1    4
## 6  1018099  1  1    1    1  2   10  3    1    1    2
```

```r
# Check if imputation using mode (=1) works fine.
cols <- colnames(breast_cancer_df_mean)[2:11]
for (col in cols) {
  if (any(is.na(breast_cancer_df_mean[[col]]))) {
    print(paste(col, "has at least one missing value"))
  } else {
    print(paste(col, "has no missing values"))
  }
}
```

```
## [1] "X5 has no missing values"
## [1] "X1 has no missing values"
## [1] "X1.1 has no missing values"
## [1] "X1.2 has no missing values"
## [1] "X2 has no missing values"
## [1] "X1.3 has no missing values"
## [1] "X3 has no missing values"
## [1] "X1.4 has no missing values"
## [1] "X1.5 has no missing values"
## [1] "X2.1 has no missing values"
```

2. Use regression to impute values for the missing data.

```
na_indices <- which(is.na(breast_cancer_df$X1.3))
breast_cancer_df_cleansed <- breast_cancer_df[-na_indices,]
lm_model <- lm(X1.3~X5+X1+X1.1+X1.2+X2+X3+X1.4+X1.5+X2.1, data=breast_cancer_df_cleansed)
summary(lm_model)
```

```
##
## Call:
## lm(formula = X1.3 ~ X5 + X1 + X1.1 + X1.2 + X2 + X3 + X1.4 +
##     X1.5 + X2.1, data = breast_cancer_df_cleansed)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6037 -0.4280 -0.2197  0.8720  8.6286
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.25090    0.31013 -13.707  < 2e-16 ***
## X5           0.01890    0.03968   0.476  0.63407
## X1          -0.16224    0.06736  -2.408  0.01629 *
## X1.1         0.18420    0.06556   2.810  0.00510 **
## X1.2         0.22088    0.04128   5.351  1.2e-07 ***
## X2           0.01924    0.05527   0.348  0.72785
## X3           0.15163    0.05336   2.842  0.00462 **
## X1.4        -0.08743    0.03972  -2.201  0.02807 *
## X1.5        -0.06300    0.05222  -1.206  0.22810
## X2.1         2.50879    0.17831  14.070  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.002 on 672 degrees of freedom
## Multiple R-squared:  0.7025, Adjusted R-squared:  0.6985
## F-statistic: 176.3 on 9 and 672 DF,  p-value: < 2.2e-16
```

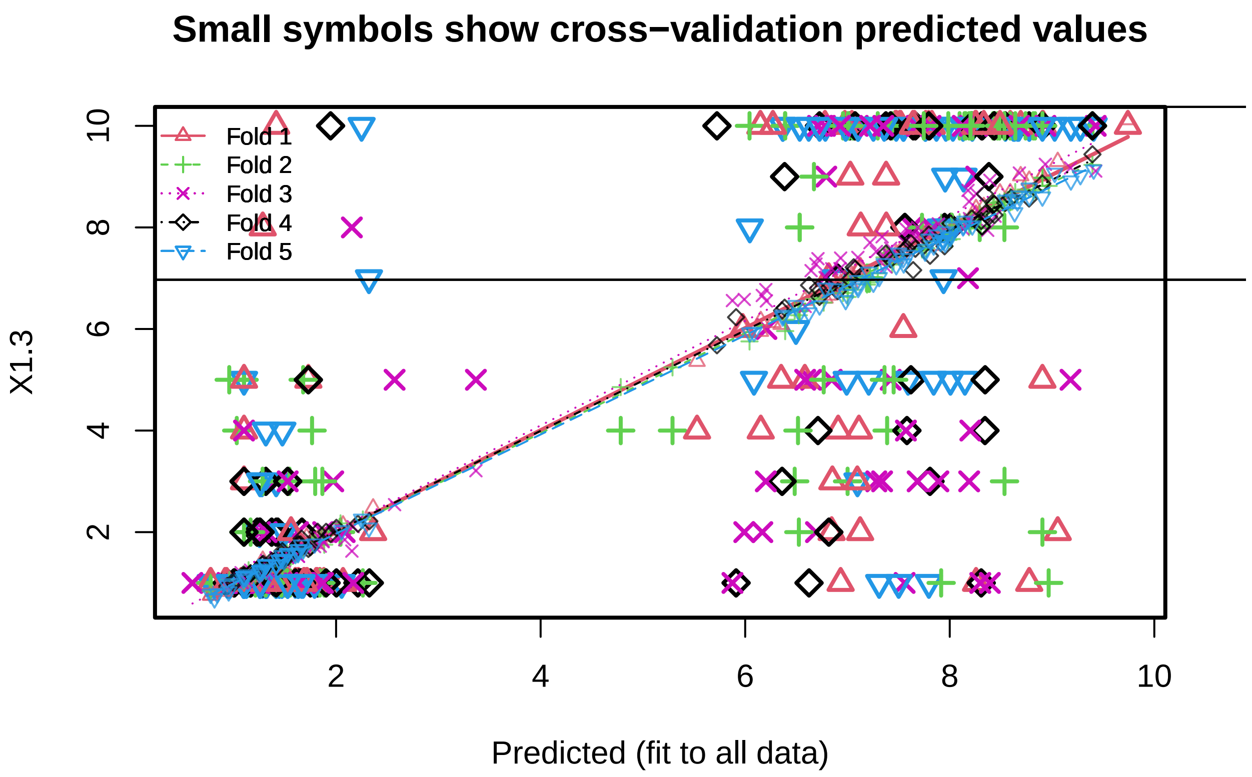
```
# Rebuild a linear regression model using significant predictors based on the p-value
lm_model <- lm(X1.3~X1+X1.1+X1.2+X3+X1.4+X2.1, data=breast_cancer_df_cleansed)
summary(lm_model)
```

```
##
```

```
## Call:
## lm(formula = X1.3 ~ X1 + X1.1 + X1.2 + X3 + X1.4 + X2.1, data = breast_cancer_df_cleansed)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.9671 -0.4143 -0.2570  0.8576  8.5857
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.28598    0.30453 -14.074  < 2e-16 ***
## X1          -0.16305    0.06504  -2.507  0.01241 *
## X1.1         0.18760    0.06502   2.885  0.00403 **
## X1.2         0.21361    0.04071   5.247 2.07e-07 ***
## X3           0.15728    0.05307   2.964  0.00315 **
## X1.4        -0.09287    0.03910  -2.375  0.01782 *
## X2.1         2.54159    0.16404  15.494  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.999 on 675 degrees of freedom
## Multiple R-squared:  0.7018, Adjusted R-squared:  0.6991
## F-statistic: 264.7 on 6 and 675 DF,  p-value: < 2.2e-16
```

```
# Use cross-validation to test how good this model really is.
model_cv <- cv.lm(breast_cancer_df_cleansed, lm_model, m=5, printit = FALSE)
```

```
## Warning in cv.lm(breast_cancer_df_cleansed, lm_model, m = 5, printit = FALSE):
##
##  As there is >1 explanatory variable, cross-validation
##  predicted values for a fold are not a linear function
##  of corresponding overall predicted values.  Lines that
##  are shown for the different folds are approximate
```



```
SST <- sum(
  (
    as.numeric(breast_cancer_df_cleansed$X1.3) - mean(as.numeric(breast_cancer_df_cleansed$X1.3)))^2
  )
R2_cv <- 1 - attr(model_cv,"ms")*nrow(breast_cancer_df)/SST
print(R2_cv)
```

```
## [1] 0.6827687
```

Observation: R squared value from the cross validation looks pretty good, which means that the linear fit would work pretty well to predict the missing values.

```
# Get predictions for missing X1.3 values
X1.3_hat <- predict(lm_model, newdata = breast_cancer_df[na_indices,])
print(X1.3_hat)
```

```
##        23        40       139       145       158       164       235       249
## 7.2021358 3.2249842 1.2570453 1.6322455 1.2873664 1.4143244 1.9771246 1.4143244
##       275       292       294       297       315       321       411       617
## 1.6322455 6.3053734 1.2570453 0.9573642 1.8318484 1.4143244 1.2570453 1.0997663
```

```
print(round(X1.3_hat))
```

```
##  23  40 139 145 158 164 235 249 275 292 294 297 315 321 411 617
##   7   3   1   2   1   1   2   1   2   6   1   1   2   1   1   1
```

```
# Impute X1.3 for missing values using predicted values from the linear regression model
breast_cancer_df_reg_imputed <- breast_cancer_df
breast_cancer_df_reg_imputed[na_indices,]$X1.3 <- X1.3_hat
breast_cancer_df_reg_imputed$X1.3 <- as.numeric(breast_cancer_df_reg_imputed$X1.3)

# Round X1.3_hat values since the original values are all integer
breast_cancer_df_reg_imputed[na_indices,]$X1.3 <- round(X1.3_hat)
breast_cancer_df_reg_imputed$X1.3 <- as.integer(breast_cancer_df_reg_imputed$X1.3)

# Make sure no X1.3 values are outside of the given range, which is between 1 and 10
breast_cancer_df_reg_imputed$X1.3[breast_cancer_df_reg_imputed$X1.3 > 10] <- 10
breast_cancer_df_reg_imputed$X1.3[breast_cancer_df_reg_imputed$X1.3 < 1] <- 1
head(breast_cancer_df_reg_imputed)
```

```
##   X1000025 X5 X1 X1.1 X1.2 X2 X1.3 X3 X1.4 X1.5 X2.1
## 1  1002945  5  4    4    5  7   10  3    2    1    2
## 2  1015425  3  1    1    1  2    2  3    1    1    2
## 3  1016277  6  8    8    1  3    4  3    7    1    2
## 4  1017023  4  1    1    3  2    1  3    1    1    2
## 5  1017122  8 10   10    8  7   10  9    7    1    4
## 6  1018099  1  1    1    1  2   10  3    1    1    2
```

3. Use regression with perturbation to impute values for the missing data.

```
# Adding a random amount up or down from the imputed estimate using linear model built above
# This can be done using a random normal distribution. We take a mean and std of the predicted values.
set.seed(1)
X1.3_hat_pert <- rnorm(nrow(breast_cancer_df_reg_imputed[na_indices,]), X1.3_hat, sd(X1.3_hat))
print(X1.3_hat_pert)
```

```
##  [1]  6.0372823  3.5664580 -0.2967562  4.5985750  1.9000664 -0.1112876
##  [7]  2.8834699  2.7871951  2.7028766  5.7375220  4.0681122  1.6822544
## [13]  0.6766886 -2.7037779  3.3487872  1.0162150
```

```
print(round(X1.3_hat_pert))
```

```
##  [1]  6  4  0  5  2  0  3  3  3  6  4  2  1 -3  3  1
```

```
# Notice that there are some negative values when we perturb the predicted values.
breast_cancer_df_pert_imputed <- breast_cancer_df
breast_cancer_df_pert_imputed[na_indices,]$X1.3 <- round(X1.3_hat_pert)
breast_cancer_df_pert_imputed$X1.3 <- as.integer(breast_cancer_df_pert_imputed$X1.3)

# Make sure no X1.3 values are outside of the given range, which is between 1 and 10
breast_cancer_df_pert_imputed$X1.3[breast_cancer_df_pert_imputed$X1.3 > 10] <- 10
breast_cancer_df_pert_imputed$X1.3[breast_cancer_df_pert_imputed$X1.3 < 1] <- 1
head(breast_cancer_df_pert_imputed)
```

```
##   X1000025 X5 X1 X1.1 X1.2 X2 X1.3 X3 X1.4 X1.5 X2.1
## 1  1002945  5  4    4    5  7   10  3    2    1    2
## 2  1015425  3  1    1    1  2    2  3    1    1    2
## 3  1016277  6  8    8    1  3    4  3    7    1    2
## 4  1017023  4  1    1    3  2    1  3    1    1    2
## 5  1017122  8 10   10    8  7   10  9    7    1    4
## 6  1018099  1  1    1    1  2   10  3    1    1    2
```

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?

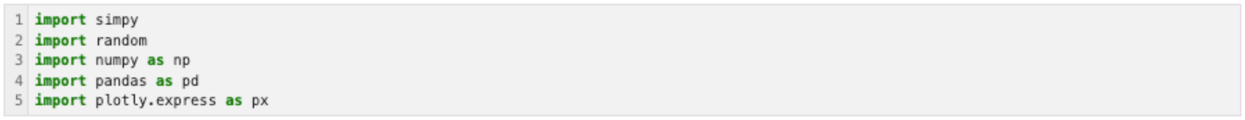
I work as a data scientist at a company specializing in lithium metal battery manufacturing. One of our primary challenges lies in electrode production, where we face issues with inconsistent quality and high production costs due to low yield. To address this, an optimization initiative is underway to refine process parameters and achieve higher quality electrodes at reduced costs.

The data that I need are as follows:

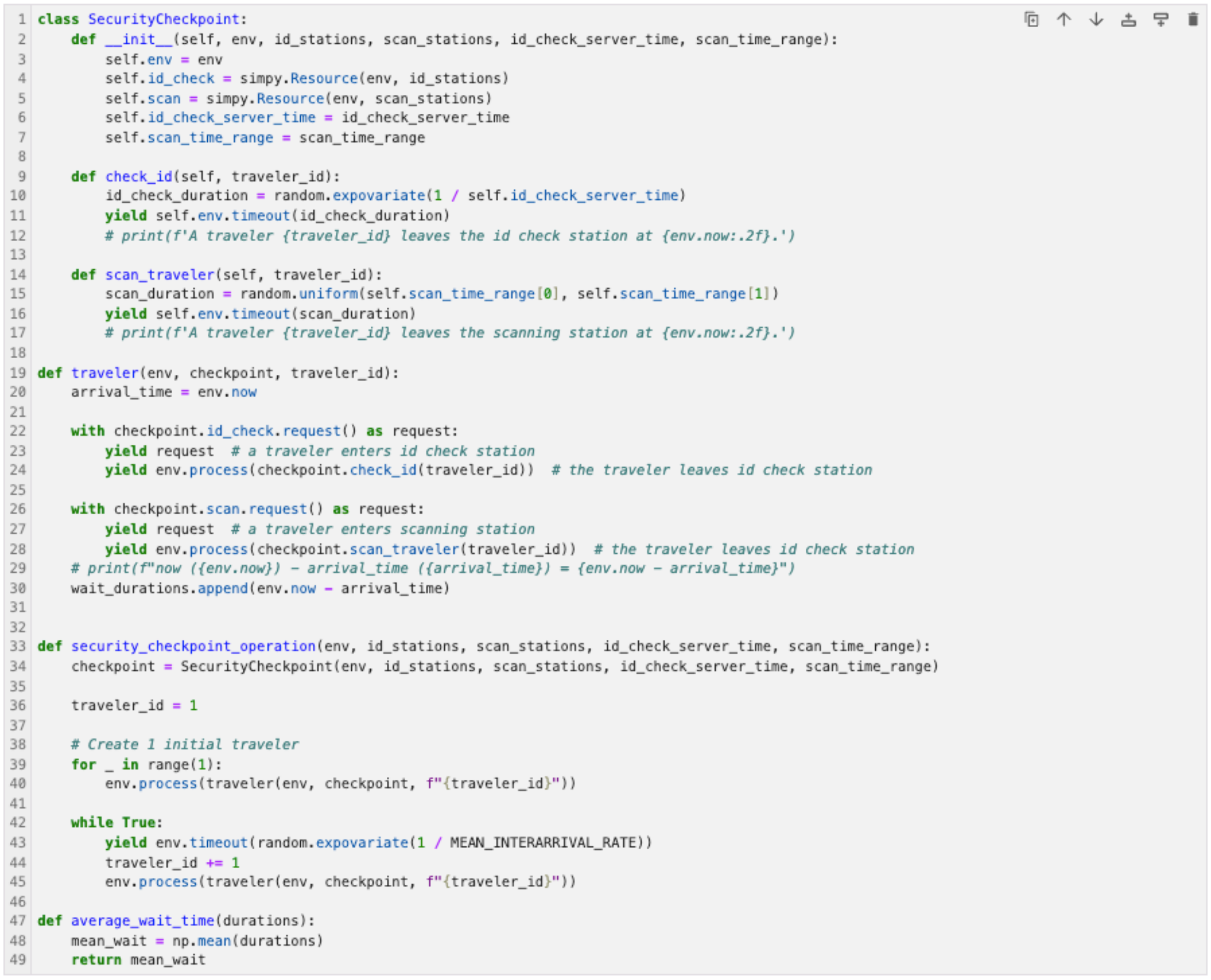
- process parameters (e.g. electrode dimensions, burr, etc.)
- quality metrics (e.g. HiPot failure rate)
- environmental conditions (e.g. ambient temperature, humidity, etc.)
- machine performance data (e.g. machine up/downtime and preventative maintenance schedule)

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.]

<My Python code for the simulation>



```python
import simpy
import random
import numpy as np
import pandas as pd
import plotly.express as px
```



```python
class SecurityCheckpoint:
    def __init__(self, env, id_stations, scan_stations, id_check_server_time, scan_time_range):
        self.env = env
        self.id_check = simpy.Resource(env, id_stations)
        self.scan = simpy.Resource(env, scan_stations)
        self.id_check_server_time = id_check_server_time
        self.scan_time_range = scan_time_range

    def check_id(self, traveler_id):
        id_check_duration = random.expovariate(1 / self.id_check_server_time)
        yield self.env.timeout(id_check_duration)
        # print(f'A traveler {traveler_id} leaves the id check station at {env.now:.2f}.')

    def scan_traveler(self, traveler_id):
        scan_duration = random.uniform(self.scan_time_range[0], self.scan_time_range[1])
        yield self.env.timeout(scan_duration)
        # print(f'A traveler {traveler_id} leaves the scanning station at {env.now:.2f}.')

def traveler(env, checkpoint, traveler_id):
    arrival_time = env.now

    with checkpoint.id_check.request() as request:
        yield request  # a traveler enters id check station
        yield env.process(checkpoint.check_id(traveler_id))  # the traveler leaves id check station

    with checkpoint.scan.request() as request:
        yield request  # a traveler enters scanning station
        yield env.process(checkpoint.scan_traveler(traveler_id))  # the traveler leaves id check station
    # print(f"now ({env.now}) - arrival_time ({arrival_time}) = {env.now - arrival_time}")
    wait_durations.append(env.now - arrival_time)


def security_checkpoint_operation(env, id_stations, scan_stations, id_check_server_time, scan_time_range):
    checkpoint = SecurityCheckpoint(env, id_stations, scan_stations, id_check_server_time, scan_time_range)

    traveler_id = 1

    # Create 1 initial traveler
    for _ in range(1):
        env.process(traveler(env, checkpoint, f"{traveler_id}"))

    while True:
        yield env.timeout(random.expovariate(1 / MEAN_INTERARRIVAL_RATE))
        traveler_id += 1
        env.process(traveler(env, checkpoint, f"{traveler_id}"))

def average_wait_time(durations):
    mean_wait = np.mean(durations)
    return mean_wait
```

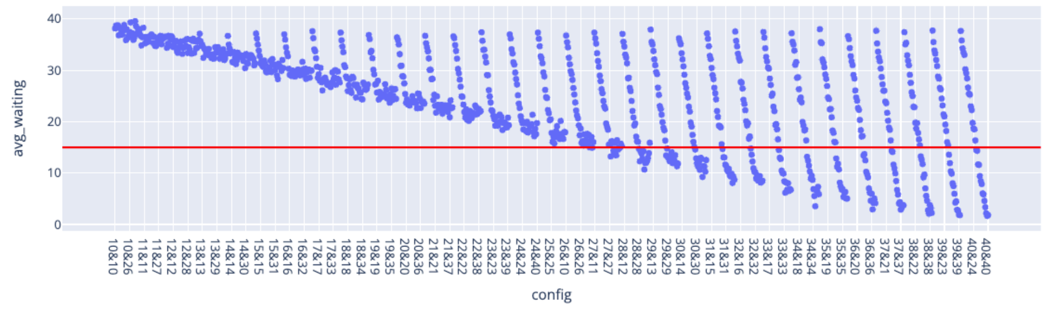
```python
SEED = 1
ID_STATIONS = 20  # will vary
ID_CHECK_SERVER_TIME = 0.75  # minutes

SCAN_STATIONS = 20  # will vary
SCAN_TIME_RANGE = [0.5, 1]  # minutes

# Passengers arrive according to a Poisson distribution with λ1 / per minute
lambda1 = 50  # λ1 = 50 to simulate a busier airport
MEAN_INTERARRIVAL_RATE = 1 / lambda1  # minutes
```

```python
# Getting unique combinations of two lists `id_check_station_time` and `scan_station_time`
import itertools

id_check_station_time = list(range(10, 41, 1))
scan_station_time = list(range(10, 41, 1))
combinations = list(itertools.product(id_check_station_time, scan_station_time))
```

```python
%%time
various_configs = list()
random.seed(SEED)
for comb in combinations:
    wait_durations = []
    ID_STATIONS = comb[0]
    SCAN_STATIONS = comb[1]
    env = simpy.Environment()
    env.process(security_checkpoint_operation(env, ID_STATIONS, SCAN_STATIONS, ID_CHECK_SERVER_TIME, SCAN_TIME_RANGE))
    env.run(until=100)
    avg_waiting = average_wait_time(wait_durations)
    various_configs.append(
        {
            "num_id_stations": ID_STATIONS,
            "num_scan_stations": SCAN_STATIONS,
            "avg_waiting": avg_waiting,
            "config": str(ID_STATIONS) + "&" + str(SCAN_STATIONS)
        }
    )
    del env

df_various_configs = pd.DataFrame(various_configs)
```

```
CPU times: user 1min 31s, sys: 15.6 ms, total: 1min 31s
Wall time: 1min 31s
```

```python
fig = px.scatter(df_various_configs.sort_values(by=["num_id_stations", "num_scan_stations"]), x="config", y="avg_waiting")
fig.add_hline(y=15, line_color="red")
fig.show()
```



Observations: To vary the number of ID/boarding-pass check stations and personal scanning stations, I iterated over combinations of different numbers of both types of stations. The range for each station type is between 10 and 40. Among 961 combinations, there are 185 combinations that result in an average travelers' waiting time of 15 minutes or less. The optimal number of each station can be determined based on external factors such as operation cost. However, if we assume the costs are the same, then the optimal setup includes 27 ID stations and 28 scanning stations, resulting in an average travelers' waiting time of approximately 13 minutes.

	num_id_stations	num_scan_stations	avg_waiting	config
522	26	36	14.988471	26&36
525	26	39	14.979172	26&39
545	27	28	13.235261	27&28
548	27	31	14.326190	27&31
549	27	32	13.821256	27&32
...	...	...	...	...
956	40	36	4.603181	40&36
957	40	37	3.373735	40&37
958	40	38	2.178493	40&38
959	40	39	1.693107	40&39
960	40	40	1.837002	40&40

185 rows × 4 columns

```python
optimals["stations_total"] = optimals["num_id_stations"] + optimals["num_scan_stations"]
```

```python
optimals.loc[optimals["avg_waiting"] <= 15].sort_values(by=["stations_total"], ascending=True).head(5)
```

	num_id_stations	num_scan_stations	avg_waiting	config	stations_total
545	27	28	13.235261	27&28	55
606	29	27	13.434151	29&27	56
576	28	28	14.727944	28&28	56
607	29	28	14.784725	29&28	57
577	28	29	12.306962	28&29	57