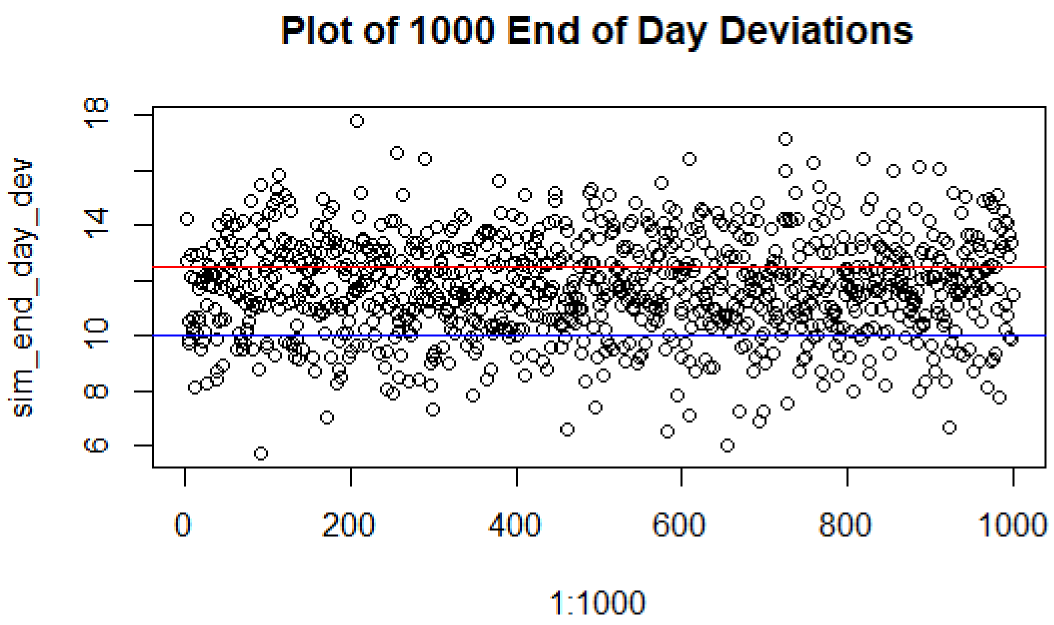
Simulation Report

John Verling

End of Day Alignment

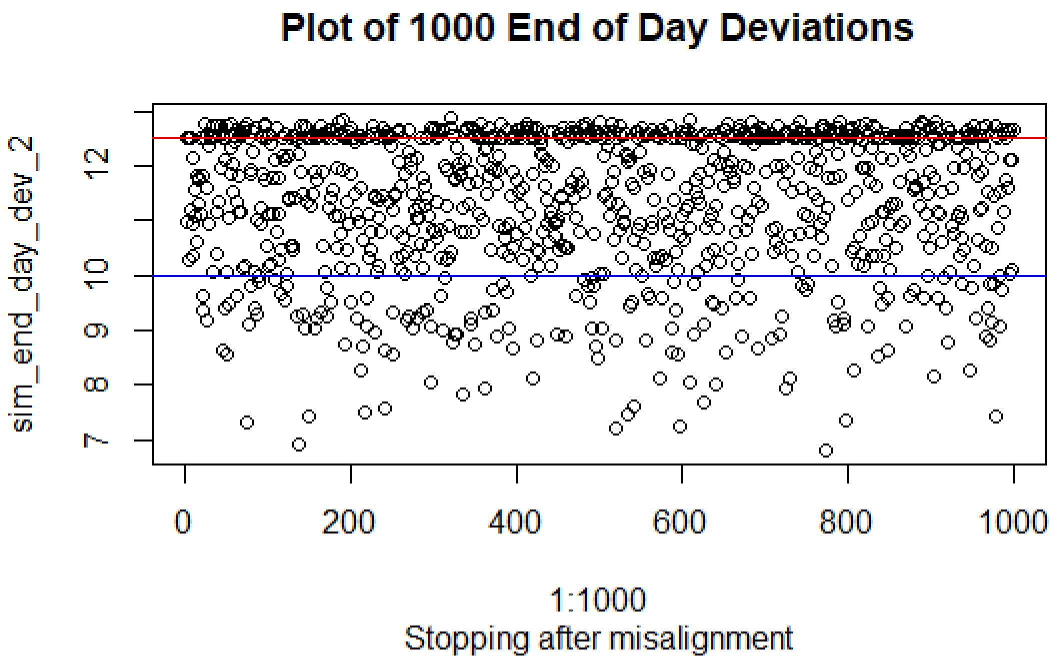
For this question I simulated 1000 days of 300 Samples per Day. Each point represents the final deviation for a day. The following graph is an example:



The red line shows the final deviations that are outside the petri-dish (12.5 mm) while the area between the red and blue lines shows the final deviations that are inaccurate (between 10 mm and 12.5 mm).

This graph is very misleading as the machine shuts down for the day after misalignment. Many of the days that ended inaccurate would have become misaligned first.

Below is a plot of the end of day deviations that takes misalignment into consideration.



In this scatter plot, if the needle becomes misaligned, the first misaligned value is taken as the last measurement of that day. We can observe now that a lot of inaccurate values are taken when the machine attempts to perform 300 samples per day.

Probability of Misalignment or Inaccuracy over day

Misalignment

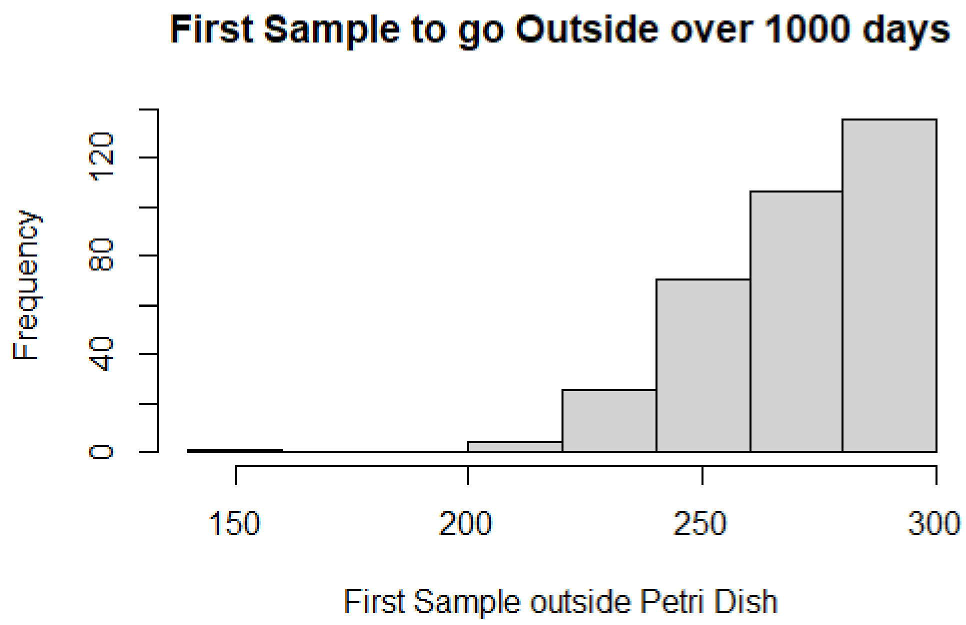
Probability

To find the probability that the machine will become misaligned at some point during a day, I simulated 1000 days of 300 samples per day. I found the number of days that had a measurement outside the 12.5 mm and divided it by the number of days.

The answer came out at 0.36 so other trials are likely to be similar to this.

Distribution

Below is the distribution of Misaligned samples.



It shows the sample numbers for the first measurement to go outside the Petri Dish over the 1000 simulated days. It is not surprising that as the sample number increases, the number of misalignments increase. We see that on approximately 120 days, misalignment happened between the 280th and 300th sample of the day. There are very few misalignments under the 200th.

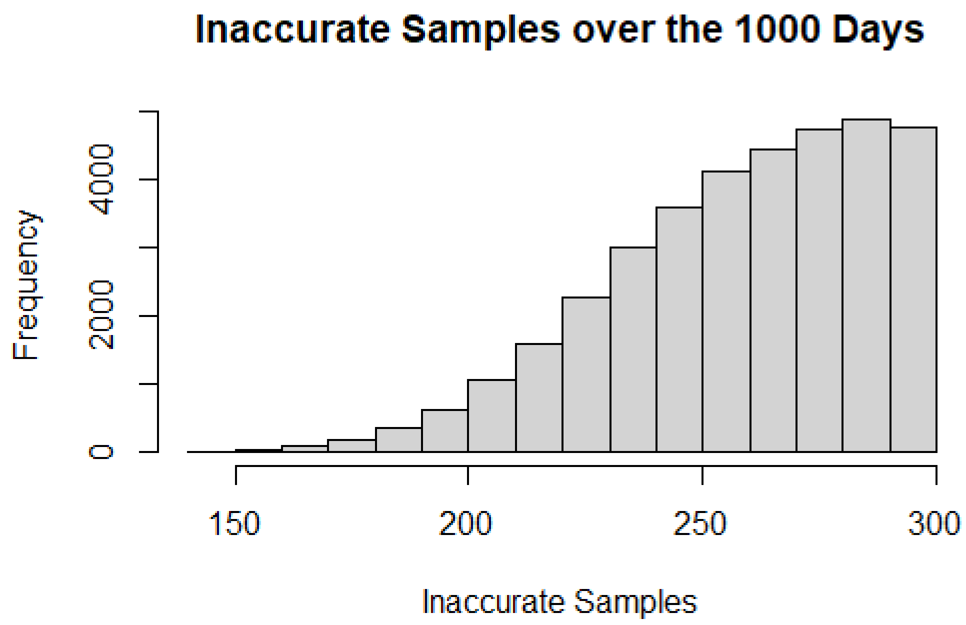
Inaccuracy

In order to find the probability of of a given measurement being inaccurate, I simulated 1000 days with an attempted 300 samples and found all the inaccurate measurements that didn't occur after misalignment. I divided the number of inaccurate measurements by the total amount of measurements (300,000).

This number came out to be around 0.12 meaning that 12% of the measurements taken over those 1000 days were inaccurate.

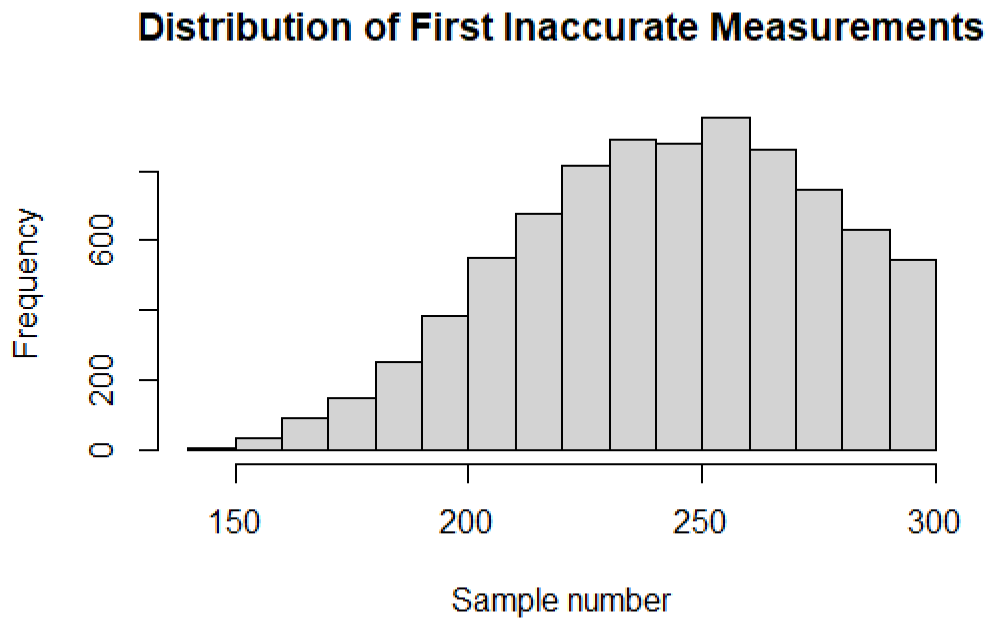
Distribution of All Inaccurate Measurements

Below is the distribution of inaccurate measurements over 1000 simulated days



We see that the inaccurate measurements begin after the 150th sample. In general, the number of inaccurate values is increasing with the sample number. However, the peak (modal class) seems to be between 280 and 290. I suspect that the 290 - 300 class is more likely to become misaligned than inaccurate.

Distribution of First Inaccurate Sample

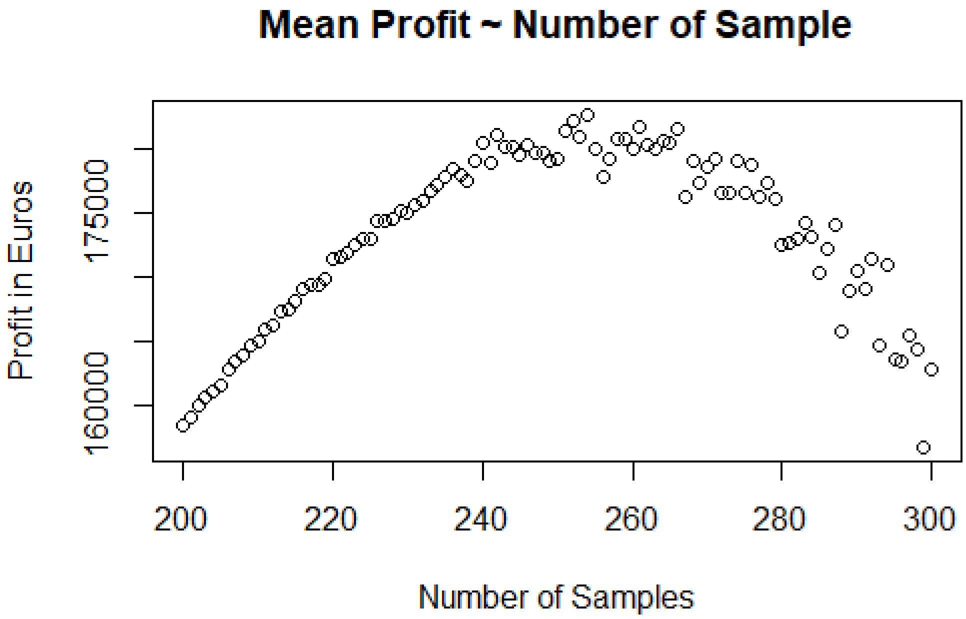


The above graph shows the first sample to be inaccurate over 10,000 days. I increased the number of days as the graph for 1000 days was not so smooth. We see the most frequent sample numbers to be between 230 and 270. This makes sense because after that, any inaccurate value is likely not to be the first. We also see the pattern that inaccuracy begins after around 150 samples.

Maximum Profit

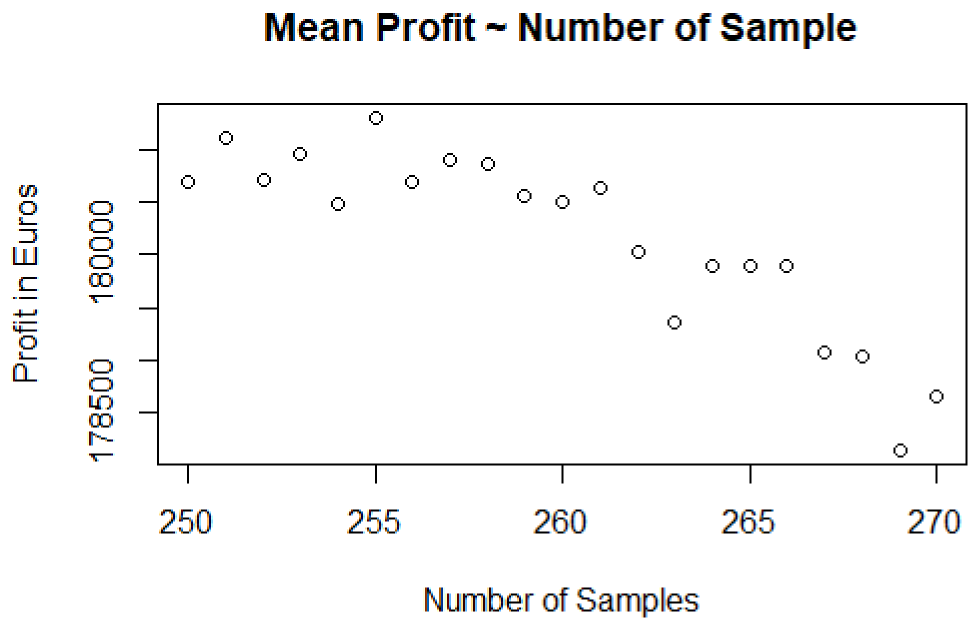
To estimate the maximum profit as a function of samples per day (N), I first simulated 1000 days for each value of N between 200 and 300 using the profit equation formed from the question. I calculated the mean profit for each N value. For example, if N = 200, I simulated 1000 days, calculated the profits for each day and found the mean of those 1000 profit values. I then repeated this process for all the N values.

I then plotted all the means for each value of N in a scatter plot, shown below:



From this graph it was clear that the maximum profit was between 250 and 270 Samples per Day and the Max Mean Profit was a little over 180,000 euros.

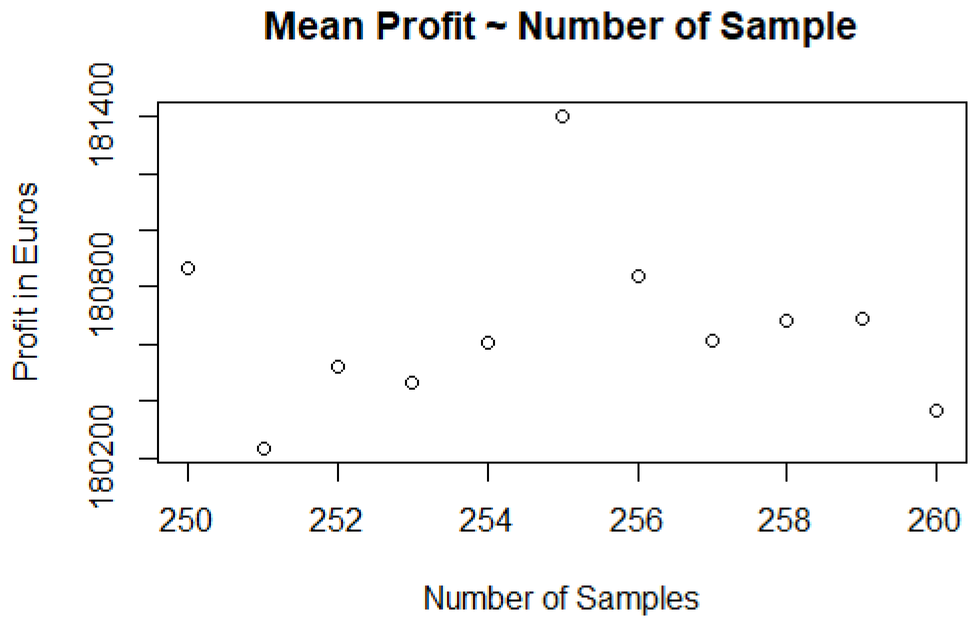
I then ran the simulation for 10,000 days for all N values between 250 and 270 Samples per Day. The plot is shown below:



This graph further narrows down the value to between 250 and 260 samples per day.

I then ran it again for 10,000 days for all N between 250 and 260.

See graph below:



From this graph (and using the max() function) we see that the maximum profit is obtained by attempting 255 samples per day, obtaining a profit of 181,403.60 euros.

Appendix

Code for Monte Carlo simulation

```
# Functions to be used later
x.dev.calculator <- function(){ # calculates deviation in x direction according to question
  x <- sample(1:3, 1) # random number between 1 and 3
  if (x == 1){
        dev.x <- runif(1,-0.2,0.4) # uniform distribution
  } else{
        dev.x <- 0
  }
  dev.x
  }

radius.calculator <- function(x,y){ # calculates the magnitude of the radius given x,y
deviations
  radius_squared = x^2 + y^2
  radius = sqrt(radius_squared)
  radius
}

# Calculating end of day deviations that would actually happen
# Excluding inaccurate values that happen after misalignment
end.day.dev.2 <- function(deviations){
  if(first.outside(deviations) <= length(deviations)){ # check if mislaignment happens
        x <- deviations[first.outside(deviations)] # if so, store the index (sample number)
  }else{
        x <- deviations[length(deviations)] # else, store final deviation
  }
  x
}



num.inaccurate <- function(deviations){ # finds the inaccurate deviations and returns the
number of them
  inaccurate.dev = c()
  for(i in 1:length(deviations)){
        if (deviations[i]>12.5){# checking to see if misalignment has already happened
        break
        }else if(deviations[i]>10 & deviations[i]<12.5){
        inaccurate.dev[i] <- deviations[i] # storing all deviations in the inaccurate zone
        }else{
        inaccurate.dev[i] <- 0
        }
  }
  length(inaccurate.dev[inaccurate.dev>0])
```

```
}

first.inaccurate <- function(deviations){ # finds the index of the first inaccurate measurement
  for (i in 1:length(deviations)){
        if(deviations[i]>10 & deviations[i]<=12.5){
        first.inaccurate <- i
        break
        }else{
        first.inaccurate <- length(deviations)+1
        }
  }
  first.inaccurate
}

# returns a vector with the indexes of inaccurate deviations
index.inaccurate <- function(deviations){
  inaccurate.indexes <- c()
  for(i in 1:length(deviations)){
        if (deviations[i]>12.5){# checking to see if misalignment has already happened
        break
        }else if(deviations[i]>10 & deviations[i]<=12.5){
        inaccurate.indexes[i] <- i  # storing all indexes for deviations
                          # in the inaccurate zone
        }else{
        inaccurate.indexes[i] <- 0  # storing a 0 if the measurement is in the accurate zone
        }
  }
  inaccurate.indexes[inaccurate.indexes>0]
}

# detects first deviation to go out of the petri-dish and returns the index
first.outside <- function(deviations){
  for (i in 1:length(deviations)){
        if(deviations[i]>12.5){
        first.outside <- i
        break
        }else{
        first.outside <- length(deviations)+1
        # return impossible index number to indicate not going outside
        }
  }
  first.outside
}

# Calculates profit for a given list of deviations with a length n
profit_day <- function(deviations, n){
  if (first.outside(deviations)>n){
        # checks if the needle goes outside at all
```

```
        p <- (n - num.inaccurate(deviations))*800 - num.inaccurate(deviations)*200
        # case where all the samples are taken
  }else if(first.inaccurate(deviations) < first.outside(deviations)){
        # checks if inaccurate values occur before misalignment
        p <- (first.outside(deviations)- 1 - num.inaccurate(deviations))*800 -
(num.inaccurate(deviations)*200) -
        (n - first.outside(deviations)+1)*400 - 75000
        # calculation takes into account inaccurate measurements and misalignment
  }else if(first.outside(deviations) < first.inaccurate(deviations)){
        # checks if the first outside is before the first inaccurate
        p <- (first.outside(deviations)-1)*800 -  # inaccurate values not applicable
        ((n - first.outside(deviations)+1)*400) - 75000
  }
  p # returns profit for that day
}

#function to calculate everything
align <- function(n){
  dev.y <- rnorm(n, -0.02, 0.1) # generating vector with n y deviations
  cum.dev.y <- cumsum(dev.y) # storing cumulative y deviations
  dev.x <- replicate(n, x.dev.calculator()) # generate vector with n x deviations
  cum.dev.x <- cumsum(dev.x) # storing cumulative y deviations
  dev.totals_over_day <- radius.calculator(cum.dev.x, cum.dev.y) # records deviation radius
  index <- 1:n
  c(end.day.disp=radius.calculator(sum(dev.x), sum(dev.y)),
        # finds theoretical end of day displacement, ignores the fact
        # that the alignment could have gone out and back in again
        end.day.disp.2 = end.day.dev.2(dev.totals_over_day),
        # finds end of day deviation, stopping after misalignment
        first.outside=first.outside(dev.totals_over_day),
        # finds first alignment that is outside the petri dish
        num.inaccurate=num.inaccurate(dev.totals_over_day),
        # computes number of misaligned deviations and checks that
        # misalignment has not happened already
        first.inaccurate=first.inaccurate(dev.totals_over_day),
        # saves the index of the first inaccurate reading
        # Question asks about "becoming inaccurate"
        profit=profit_day(dev.totals_over_day, n),
        # calculates profit for the day

        index.inaccurate=index.inaccurate(dev.totals_over_day)
        # finds all indexes for inaccurate measurements
        )
  }
```

```
# Plotting End of Day Deviations
sim_end_day_dev <- replicate(1000, align(300)["end.day.disp"])
# View(sim_end_day_dev)
plot(x=1:1000, y=sim_end_day_dev, main = "Plot of 1000 End of Day Deviations")
abline(h=12.5, col="red",)
# inaccuracy Limit is 10 mm
abline(h=10, col="blue")

# Plotting End of day deviations,
# stopping after misalignment
sim_end_day_dev_2 <- replicate(1000, align(300)["end.day.disp.2"])
plot(x=1:1000, y=sim_end_day_dev_2, main = "Plot of 1000 End of Day Deviations",
                              sub = "Stopping after misalignment")
abline(h=12.5, col="red",)
# inaccuracy Limit is 10 mm
abline(h=10, col="blue")


# Distribution of number of tests before first inaccuracy
sim_first_inaccurate <- replicate(10000, align(300)["first.inaccurate"])
sim_first_inaccurate
hist(sim_first_inaccurate[sim_first_inaccurate<=300],
          main = "Distribution of First Inaccurate Measurements",
          xlab = "Sample number")


# Probability of becoming inaccurate at some point during a day
length(sim_inaccurate[sim_inaccurate<=300])/length(sim_inaccurate) #= 0.849
# note: It is possible that the needle becomes inaccurate at some point
# and then returns to accurate. It could also become inaccyrate again

# Finding distribution of inaccurate measurement indexes
total.inaccurates <- c()  # vector to store all inaccurate measurement
# indexes for all repetitions
for(i in 1:1000){
  data <- align(300) # performs simulation for the day (assuming 300 samples)
  inaccurate.indexes <- data[7:length(data)]
  # stores the indexes of inaccurate measurements for that day
  total.inaccurates <- append(total.inaccurates, inaccurate.indexes)
  # appends the inaccurate indexes to one vector for histogram and calculation
}
#inaccurate.indexes

# Probability of any given measurement to be inaccurate,
# not including values after misalignment
length(total.inaccurates)/(300*1000) # around 0.12
# total inaccurate measurements divided by total samples over 100 days
```

```
# Probability of becoming misaligned at some point during the day
sim_first_outside <- replicate(1000, align(300)[3])
length(sim_first_outside[sim_first_outside<=300])/length(sim_first_outside) # around 0.4
# Assuming that if the needle becomes misaligned,
# the machine is shut down for the rest of the day

# Distribution of first measurements outside
hist(sim_first_outside[sim_first_outside<=300],
        xlab = "First Sample outside Petri Dish",
        main = "First Sample to go Outside over 1000 days")

# Distribution of ALL sample numbers that were inaccurate
hist(total.inaccurates[total.inaccurates<=300],
        xlab = "Inaccurate Samples",
        main = "Inaccurate Samples over the 1000 Days") # distribution of inaccurate
indexes over 1000 work days

# Finding max profit
# maximum is almost certainly between 200 and 300
average.profit.broad = replicate(300, NA)
for(N in 200:300){
  profit.sim <- replicate(1000, align(N)[6])
  average.profit.broad[N] <- mean(profit.sim) # storing average for 1000 days
                                              # with N samples
}
# View(average.profit)
# hist(average.profit[200:300], breaks = 5) #

plot(x=200:300, y=average.profit.broad[200:300],
        main = "Mean Profit ~ Number of Sample",
        xlab = "Number of Samples",
        ylab = "Profit in Euros") # plotting profit against N

max(average.profit.broad[200:300])
# Max profit is between 250 and 270 and around 183,100 euros

# rerunning between 250 to 270
average.profit = replicate(300, NA)
for(N in 250:270){
  profit.sim <- replicate(10000, align(N)[6]) # 10000 simulated days per N
  average.profit[N] <- mean(profit.sim)
}
plot(x=250:270, y=average.profit[250:270],
        main = "Mean Profit ~ Number of Sample",
        xlab = "Number of Samples",
        ylab = "Profit in Euros")

# Max seems to be between 250 and 260
```

```
average.profit = replicate(300, NA)
for(N in 250:260){
  profit.sim <- replicate(10000, align(N)[6]) # 10000 simulated days per N
  average.profit[N] <- mean(profit.sim)
}

plot(x=250:260, y=average.profit[250:260],
          main = "Mean Profit ~ Number of Sample",
          xlab = "Number of Samples",
          ylab = "Profit in Euros")
average.profit[255]
max(average.profit[250:260]) # 255 samples with a net profit 181,403..60 euros
```

Code for Shiny Dashboard

```
read.data <- function(){
  setwd("C:\\Users\\jverl\\Desktop\\Data Science MTU\\R 8010\\Assignment 2")
# setting directory
  car_data <- read.csv("STAT8010_assignment2_2022.csv") # reading in CSV
  car_data
}

collapsing.classes <- function(){
  car_data_V1 <- car_data # making a copy of data

  car_data_V1["Collapsed Class"] <- replicate(length(car_data_V1$Vehicle.Class), NA)
  #Collapsing Classes
  boolean_compact <- grepl("COMPACT", car_data_V1$Vehicle.Class)
  # making a boolean of all Vehicle classes containing "COMPACT"
  car_data_V1$`Collapsed Class`[boolean_compact] <- "COMPACT"
  # Replacing the NA values in collapsed to "COMPACT" car_data_V1$`Collapsed Class`

  # Tourism Class
  boolean_tourism_full_size <- grepl("FULL-SIZE", car_data_V1$Vehicle.Class)
  boolean_tourism_mid_size <- grepl("MID-SIZE", car_data_V1$Vehicle.Class)
  boolean_tourism_station_wagon <- grepl("STATION WAGON",
car_data_V1$Vehicle.Class)
  car_data_V1$`Collapsed Class`[boolean_tourism_full_size] <- "TOURISM"
  car_data_V1$`Collapsed Class`[boolean_tourism_mid_size] <- "TOURISM"
  car_data_V1$`Collapsed Class`[boolean_tourism_station_wagon] <- "TOURISM"
  #car_data_V1$`Collapsed Class`

  # Van Class
  boolean_van <- grepl("VAN", car_data_V1$Vehicle.Class)
  car_data_V1$`Collapsed Class`[boolean_van] <- "VAN"
```

```
  # Pick up Class
  boolean_pickup <- grepl("PICKUP", car_data_V1$Vehicle.Class)
  car_data_V1$`Collapsed Class`[boolean_pickup] <- "PICK UP"

  # Special Purpose Vehicles
  boolean_special <- grepl("SPECIAL PURPOSE VEHICLE", car_data_V1$Vehicle.Class)
  #car_data_V1$Vehicle.Class[boolean_special]
  car_data_V1$`Collapsed Class`[boolean_special] <- "SPECIAL"

  # SUV Class
  boolean_suv <- grepl("SUV", car_data_V1$Vehicle.Class)
  car_data_V1$`Collapsed Class`[boolean_suv] <- "SUV"

  # Sport Class
  boolean_sport <- grepl("TWO-SEATER", car_data_V1$Vehicle.Class)
  car_data_V1$`Collapsed Class`[boolean_sport] <- "SPORT"

  # Changing the original class in the copy
  car_data_V1$Vehicle.Class <- car_data_V1$`Collapsed Class`

  #car_data_V1$`Collapsed Class` # Checking for NA's
  #car_data_V1$Vehicle.Class[is.na(car_data_V1$`Collapsed Class`)] # length = 0, no NAs

  # Creating Column with Collapsed Transmission
  car_data_V1["Collapsed Transmission"] <- replicate(length(car_data_V1$Transmission),
NA)

  boolean_automatic <- grepl("A", car_data_V1$Transmission)
  car_data_V1$`Collapsed Transmission`[boolean_automatic] <- "Automatic"

  boolean_automatic <- grepl("M", car_data_V1$Transmission)
  car_data_V1$`Collapsed Transmission`[boolean_automatic] <- "Manual"

  # Changing original Transmission column
  car_data_V1$Transmission <- car_data_V1$`Collapsed Transmission`
  #car_data_V1$Vehicle.Class <- car_data_V1$`Collapsed Class`

  car_data_V1
}

change.names <- function(){ # part 2: changing column/variable names
  colnames(car_data_V1) <- c("Make", "Model", "Vehicle_Class", "Engine_Size_in_L",
                  "cylinders", "Transmission", "Fuel_Type",
"Fuel_Consumption_City_L_per_100km",
                  "Fuel_Consumption_HWY_L_per_100km",
"Fuel_Consumption_Combo_L_per_100km",
                  "Fuel_Consumption_Combo_mpg", "CO2_Emmisions_g_per_km")
  car_data_V1
```

```
}

car_data <- read.data() # reading data
car_data_V1 <- collapsing.classes()[1:12] # extracting relevant columns
car_data_V1 <- change.names() # changing names of variables

# Data pre-processing
# Factoring character variables
car_data_V1$Transmission <- as.factor(car_data_V1$Transmission)
car_data_V1$Vehicle_Class <- as.factor(car_data_V1$Vehicle_Class)
car_data_V1$Make <- as.factor(car_data_V1$Make)
car_data_V1$Model <- as.factor(car_data_V1$Model)
car_data_V1$Fuel_Type <- as.factor(car_data_V1$Fuel_Type)
car_data_V1$cylinders <- as.factor(car_data_V1$cylinders)

str(car_data_V1)
#View(car_data_V1)
library(shiny)
#runExample("04_mpg")
#str(car_data_v1)
#labels(car_data_v1)

library(ggplot2)
library(summarytools)

ui <- fluidPage(

  # App title ----
  titlePanel("Car CO2 Plots"),

  # Sidebar layout

  sidebarLayout(

        # Sidebar panel for inputs ----
        sidebarPanel(

        # Input: Selector Variables for PLot

        # Variable 1
        selectInput("variable_1", "Select first variable:",
                c("Make" = "Make",
                "Model" = "Model",
                "Vehicle Class" = "Vehicle_Class",
                "Engine Size/L" = "Engine_Size_in_L",
                "cylinders" = "cylinders",
                "Transmission" = "Transmission",
```

```
                    "Fuel Type" = "Fuel_Type",
                    "Fuel Consumption City L/100km" = "Fuel_Consumption_City_L_per_100km",
                    "Fuel Consumption HWY L/100km" =
"Fuel_Consumption_HWY_L_per_100km",
                    "Fuel Consumption Combo L/100km" =
"Fuel_Consumption_Combo_L_per_100km",
                    "Fuel Consumption Combo mpg" = "Fuel_Consumption_Combo_mpg",
                    "CO2 Emmisions g/km" = "CO2_Emmisions_g_per_km"
                    )),

          # Variable 2
          selectInput("variable_2", "Select second variable:",
                    c("Make" = "Make",
                    "Model" = "Model",
                    "Vehicle Class" = "Vehicle_Class",
                    "Engine Size/L" = "Engine_Size_per_L",
                    "cylinders" = "cylinders",
                    "Transmission" = "Transmission",
                    "Fuel Type" = "Fuel_Type",
                    "Fuel Consumption City L/100km" = "Fuel_Consumption_City_L_per_100km",
                    "Fuel Consumption HWY L/100km" =
"Fuel_Consumption_HWY_L_per_100km",
                    "Fuel Consumption Combo L/100km" =
"Fuel_Consumption_Combo_L_per_100km",
                    "Fuel Consumption Combo mpg" = "Fuel_Consumption_Combo_mpg",
                    "CO2 Emmisions g/km" = "CO2_Emmisions_g_per_km"
                    )),

          # Allow to user to fit a linear model or not
          checkboxInput("fit_line", "Fit line", FALSE),

          # Allow user to add factor as colour in plots
          checkboxInput("add_factor", "Choose factor"),

          # Making this condition hides the choices for factoring until factoring is chosen
          # Also, allows me to use it for conditions later on
          conditionalPanel(condition = "input.add_factor == true",
                    radioButtons("factor", "Colour Factor: ",
                              c("Transmission" = "Transmission",
                              "Vehicle_Class" = "Vehicle_Class",
                              "Make" = "Make",
                              "Model" = "Model",
                              "Fuel_Type" = "Fuel_Type",
                              "cylinders" = "cylinders"))),

          ),

          # Main panel for displaying outputs ----
```

```
        mainPanel(

        # Output: Formatted text for caption ----
        h3(textOutput("caption")),

        # Tabs to display along with their outputs
        tabsetPanel(type = "tabs",
                tabPanel("Plot", plotOutput("plot")),
                tabPanel("summary_var_1", verbatimTextOutput("summary_var_1")),
                tabPanel("Summary_var_2", verbatimTextOutput("summary_var_2")),
                tabPanel("Histogram_var_1", plotOutput("hist_var_1")),
                tabPanel("Histogram_var_2", plotOutput("hist_var_2"))
                )


        )
  )
)

server <- function(input, output) {

  output$caption <- reactive({
        paste("Chosen Variables: ", input$variable_1, " and ", input$variable_2, sep="")
# Displays Caption
  })

  # Wrote this to provide a warning about applying factoring when not applicable but could
not get it working properly
  # It prevented the unfactored plots from showing when it displayed the warning
  colour_factor_validation <- reactive({
        var_1_bool <- is.numeric(car_data_V1[[input$variable_1]])
        var_2_bool <- is.numeric(car_data_V1[[input$variable_2]])
        validate(
        need(var_1_bool | var_2_bool,
        "Colour Factor only available for Plots and Boxplots")
        )
  })
        output$plot <- renderPlot({ # Producing the plots as requested
        var_1_bool <- is.numeric(car_data_V1[[input$variable_1]])
        var_2_bool <- is.numeric(car_data_V1[[input$variable_2]])
        # if(input$add_factor){
        #   colour_factor_validation()
        # }
        #print(var_1_bool)
        #print(var_2_bool)
        if(var_1_bool & var_2_bool){ # check if both are numeric => scatter plot produced
        if(input$fit_line){ # check if "fit line" is selected
        df <- data.frame(car_data_V1[input$variable_1],
```

```
            car_data_V1[input$variable_2])
        x <- coef(lm(df)) # finding equation coefficients
        x[1] <- round(x[1],2) # intercept
        x[2] <- round(x[2],2) # slope
        correlation <- cor(df) # r
        coef_determination <- round(correlation[1,2]^2, 2)
        # rounding r and squaring to find coeff of determination
        coef_determination <- as.character(coef_determination)
# coercing to character for the paste function
        equation <- paste("Best Fit: Y = ", as.character(x[2]), "X + ",
                        as.character(x[1]),", R-Squared: ",
                        coef_determination, sep="") # creating text to print in label

        # wrapping the ggplot in ifelse to respond to factor user choice
        # In hindsight, this should have been a function to increase readability
        # and also, I copied and pasted the code several times
        ifelse(input$add_factor, g <- ggplot(data = car_data_V1,
                        aes_string(x=input$variable_1,
                            y=input$variable_2,
                            col=input$factor)),
                        g <- ggplot(data = car_data_V1,
                        aes_string(x=input$variable_1,
                        y=input$variable_2)))
        g +
        geom_point() + # create scatter plot
        geom_smooth(method="lm", se=T) + # create best fit line
        geom_text(x = median(car_data_V1[[input$variable_1]]),
                y = max(car_data_V1[input$variable_2]),
                label = equation)
            # add text with equation and r squared
        }else{ # "fit line" not selected
        # wrapping the ggplot in ifelse to respond to factor user choice
        ifelse(input$add_factor, g <- ggplot(data = car_data_V1,
                            aes_string(x=input$variable_1,
                                y=input$variable_2,
                                col=input$factor)),
                        g <- ggplot(data = car_data_V1,
                            aes_string(x=input$variable_1,
                                y=input$variable_2)))
        g +
        geom_point() # plotting without linear model
        }

        }else if(var_1_bool | var_2_bool){ # checking if one variable is numeric only =>
boxplot
        if(var_1_bool){ # if the first variable is numeric, put it on the x-axis
        # wrapping the ggplot in ifelse to respond to factor user choice
        ifelse(input$add_factor, g <- ggplot(data = car_data_V1,
```

```
                        aes_string(x=input$variable_2,
                                y=input$variable_1,
                                col=input$factor)),
                g <- ggplot(data = car_data_V1,
                        aes_string(x=input$variable_2,
                                y=input$variable_1)))

        g +
        geom_boxplot() +
        theme(axis.text.x = element_text(angle=65, vjust = 0.6))
        }else{
        if(var_2_bool){ # if second variable is numerical, put it on the x-axis
        # wrapping the ggplot in ifelse to respond to factor user choice
        ifelse(input$add_factor, g <- ggplot(data = car_data_V1,
                        aes_string(x=input$variable_1,
                                y=input$variable_2,
                                col=input$factor)),
                g <- ggplot(data = car_data_V1,
                        aes_string(x=input$variable_1,
                                y=input$variable_2)))
            g +
            geom_boxplot()+
            theme(axis.text.x = element_text(angle=65, vjust = 0.6))
        }
        }
        }else{ # both variable are categorical
        # wrapping the ggplot in ifelse to respond to factor user choice
        if(input$variable_1 != input$variable_2){
        # produces a barplot where variable 1 is the height of the bars and
        # variable 2 is indicated in the fill
        ggplot(car_data_V1, aes_string(x=input$variable_1, fill=input$variable_2)) +
        geom_bar() +
        theme(axis.text.x = element_text(angle=65, vjust = 0.6)) # angling text to avoid
overlapping labels
        }else{
        print("both the same") # handling case where the two variables are the same
        }
        }
        })


  output$summary_var_1 <- renderPrint({ # Using renderprint to print data summary
        if(is.numeric(car_data_V1[[input$variable_1]])){ # if numeric, show descriptive stats
        descr(car_data_V1[input$variable_1])
        }else{
        df <- data.frame(car_data_V1[input$variable_1],car_data_V1[input$variable_2])
        table(df) # if categorical, print a table with both variables counted
        }
```

```
  })

  output$summary_var_2 <- renderPrint({ # Using renderprint to print data summary
        if(is.numeric(car_data_V1[[input$variable_2]])){ # if numeric, show descriptive stats
        descr(car_data_V1[input$variable_2])
        }else{
        df <- data.frame(car_data_V1[input$variable_1],car_data_V1[input$variable_2])
        table(df) # if categorical, print a table with both variables counted
        }
  })

  hist_1_validation <- reactive({ # used later for the histogram for variable 1
        validate( # validation function to check for numerical data to produce histogram
        need(is.numeric(car_data_V1[[input$variable_1]]), "Non numerical data: no
histogram available")
        ) # warns user if data is not numerical
  })

  hist_2_validation <- reactive({# used later for the histogram for variable 2
        validate( # validation function to check for numerical data to produce histogram
        need(is.numeric(car_data_V1[[input$variable_2]]), "Non numerical data: no
histogram available")
        ) # warns user if data is not numerical
  })

  output$hist_var_1 <- renderPlot({ # produces histogram if validation code is satisfied
        hist_1_validation() # checks for numerical data
        if(is.numeric(car_data_V1[[input$variable_1]])){ # if true, plot histogram
        ggplot(data=car_data_V1, aes_string(input$variable_1)) +
        geom_histogram()
        }
        })
  output$hist_var_2 <- renderPlot({ # produces histogram if validation code is satisfied
        hist_2_validation() # checks for numerical data
        if(is.numeric(car_data_V1[[input$variable_2]])){ # if true, plot histogram
        ggplot(data=car_data_V1, aes_string(x=input$variable_2)) +
        geom_histogram()
        }
  })

}

# Create Shiny app ----
shinyApp(ui, server)
```