MATH501 Coursework - Report

10570155, 10696253, 10701983

26/04/2021

The following report will discuss the results of the MATH501 questions that were asked in two sections of machine learning and statistical analysis.

Machine Learning

Part (a)

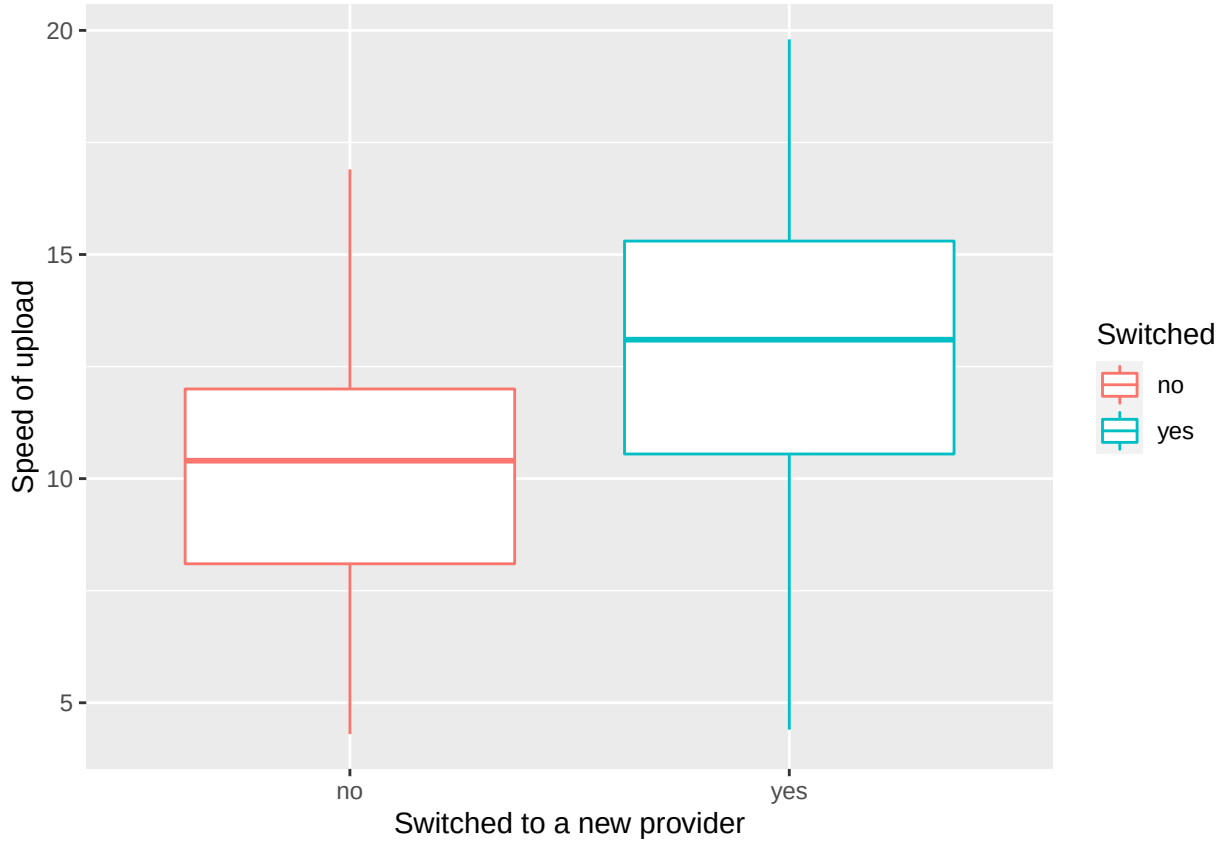
Present the data visually using box-and-whisker plots with a distinction for churn. Comment on the data in the context of the problem.

Reading our data in a dataframe:

```
data_path <- "data/churndata.txt"
churn_data <- read.csv(data_path, sep = " ")
churn_data <- na.exclude(churn_data) # removing entries with NA values
```

Boxplot with average speed of upload against an indicator whether a customer switched to a different provider:

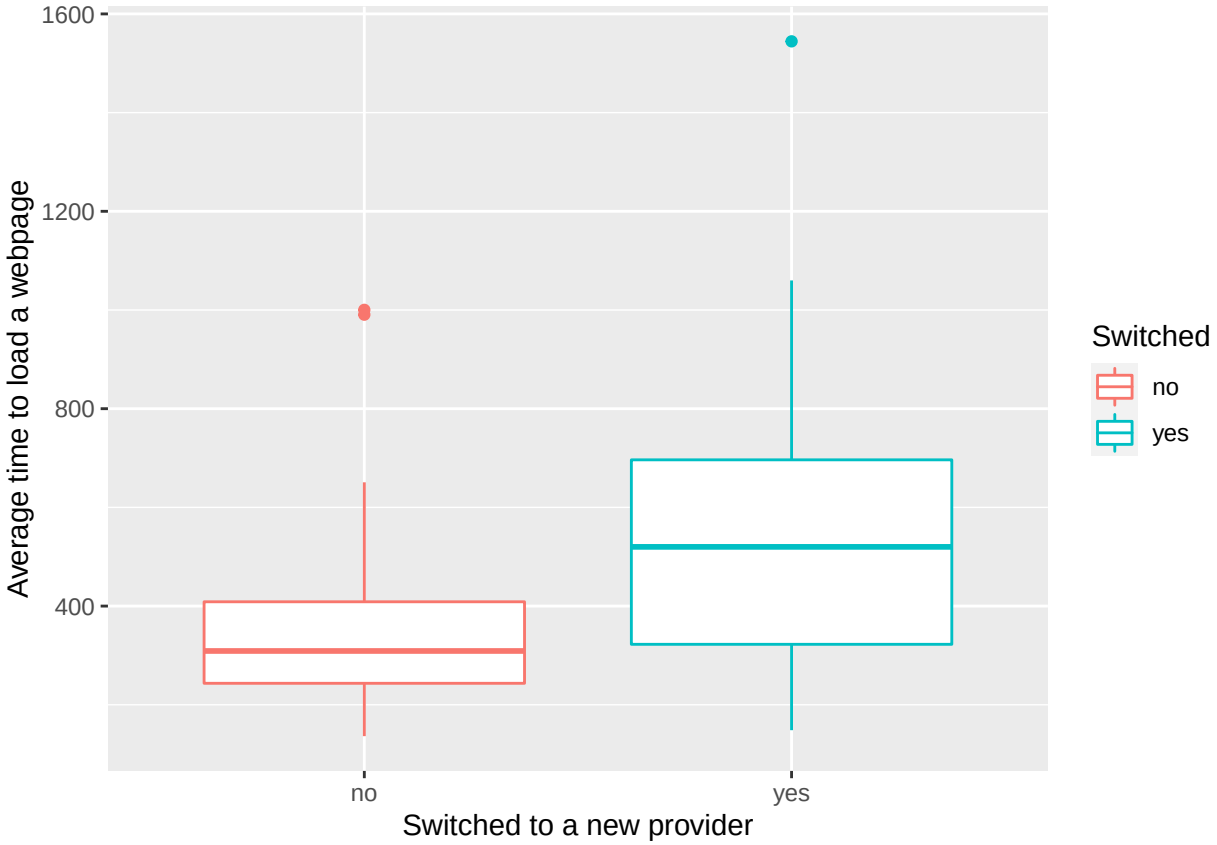
```
churn_data %>% ggplot(aes(x = churn, y = upload, color = churn)) +
 geom_boxplot() +
 labs (y = "Speed of upload",
       x = "Switched to a new provider",
       color = "Switched")
```



As we can see the customers who have not yet switched to a different operator have lower average speed of upload in the internet in comparison with the customers who have switched. In general, having higher uplink speed is a plus since it improves the Internet phone/video call experience and it does no harm to the customers Hence increasing uplink speed could not affect the customers' decision to switch to different operators.

Boxplot with the mean time to load a webpage against an indicator whether a customer switched to a different provider:

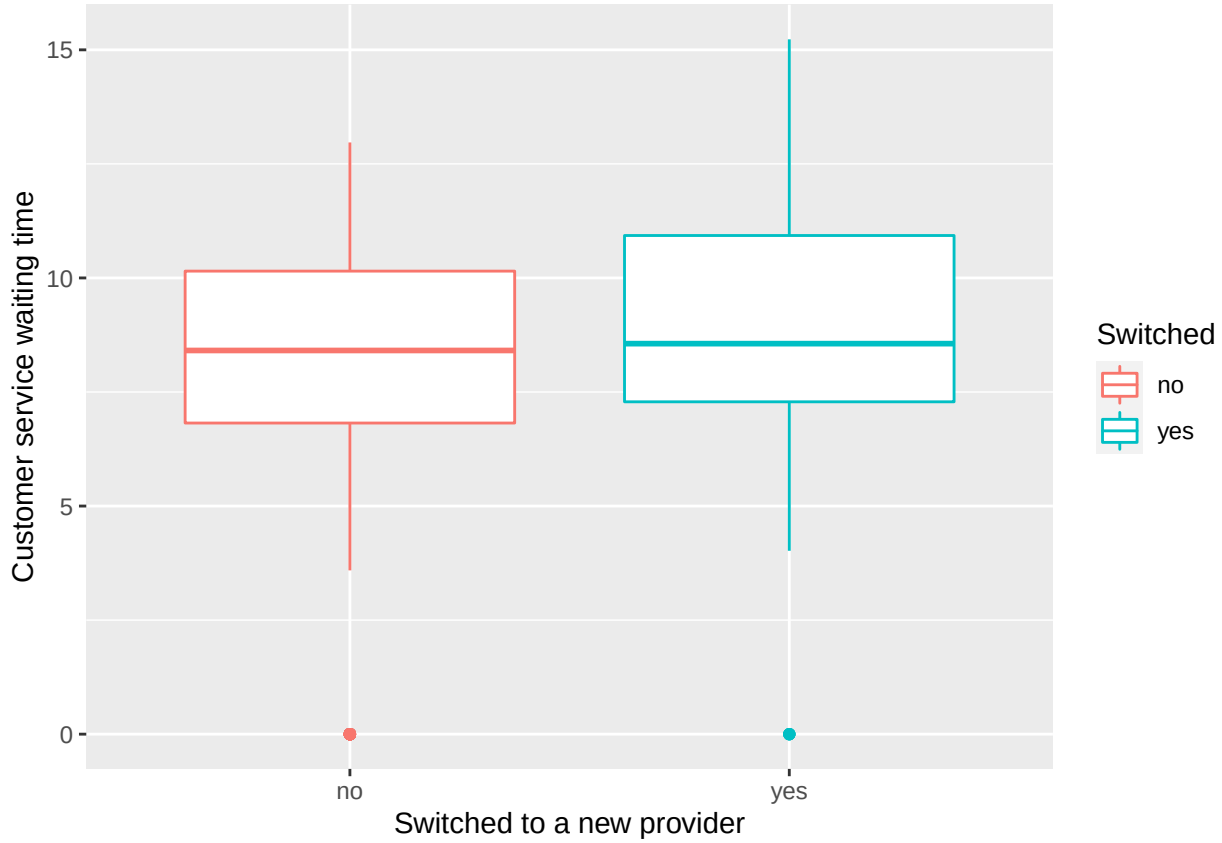
```
churn_data %>% ggplot(aes(x = churn, y = webget, color = churn)) +
 geom_boxplot() +
 labs (y = "Average time to load a webpage",
       x = "Switched to a new provider",
       color = "Switched")
```



We can observe a strong dependency between the time to load a webpage (which directly corresponds to the downlink speed) and an indicator whether customers changed their operators. The average downlink speed was significantly lower for the customers that have switched to a different provider than for those who haven't. We can conclude this as the average time to load a webpage for those clients who switched is nearly 200 units longer than of those who didn't.

Boxplot with how long a customer waited on the phone call for a customer service operator against an indicator whether a customer switched to a different provider:

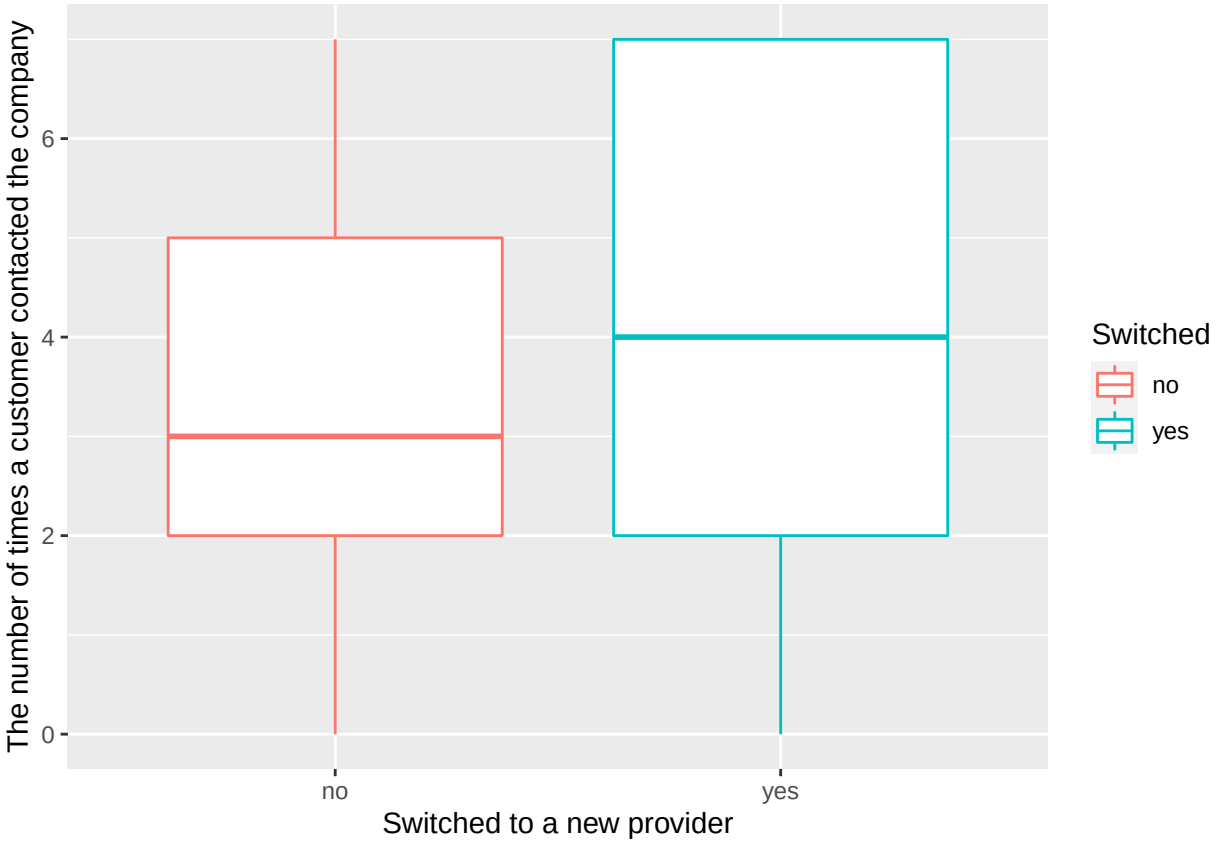
```
churn_data %>% ggplot(aes(x = churn, y = callwait, color = churn)) +
 geom_boxplot() +
 labs (y = "Customer service waiting time",
       x = "Switched to a new provider",
       color = "Switched")
```



Even though the average waiting time for a customer service operator is similar in both cases, overall the majority of of customers who switched to a different operator had to wait longer than the average and the customers who haven't changed their provider. We can assume that the time spent by a customer on a call while they're waiting for a customer service operator to attend may impact their decision to switch to another operator although the influence seems to be less significant compared to time to load a webpage.

Boxplot with the number of times a customer contacted the company via a phone call against an indicator whether a customer switched to a different provider:

```
churn_data %>% ggplot(aes(x = churn, y = enqcount, color = churn)) +
 geom_boxplot() +
 labs (y = "The number of times a customer contacted the company",
       x = "Switched to a new provider",
       color = "Switched")
```



We can observe that in average the customers that switched to a different operator contacted the company via a phone call 1 time more often than others The biggest number of contact attempts is 2 calls more than of the customers who haven't changed their providers. Needs to be mentioned that some of the customers who switched didn't contact the company even once. On the contrary minority of the customers who haven't changed their provider also have more than 5 and even 6 calls. Still it will be safe to assume that the number of calls impacts the customers' decision to choose a different operator but its importance is smaller the time to load a webpage.

Conclusion

Out of all the 4 factors that can possibly influence the 'churn' variable time to load the webpage (which subsequently leads to the downlink speed) is the most important one. Average phone call customer service waiting time doesn't differ drastically but still is higher for customers who chose different providers hence we could conclude that this aspect also plays its part in the customers' decision as well as the number of phone calls to the company. The most suspicious variable is the upload speed - for those clients who changed their providers the uplink speed was actually higher but the downlink speed was lower (comparing to the customers who didn't change their provider) while normaly the opposite should be the case (unless we're talking about 5G). Unfortunately, we don't have access to any other data hence we can only speculate that perhaps there are some issues with the provider's network.

Part (b)

Create a training set consisting of 350 randomly chosen data points and a test set consisting of the remaining 150 data points.

```
set.seed(1) # to make the results reproducible
num_subset <- sample(nrow(churn_data), 350) # randomly choose 350 numbers out of 500
```

Separating the data into predictors (X) and classifier (Y):

```
attach(churn_data)

X <- cbind(upload, webget, enqcount, callwait)
Y <- churn
Y <- as.integer(Y == 'yes')
Y <- as.factor(Y)

detach(churn_data)
```

Dividing the data into training and testing sets:

```
train.X <- X[num_subset, ] # 350 records
train.Y <- Y[num_subset]

test.X <- X[-num_subset, ] # 150 records
test.Y <- Y[-num_subset]
```

Part (c)

Using the training data set apply the K nearest neighbours method to construct a classifier to predict churn based on the four available predictors. Find the optimal K using leave-one-out cross-validation for the training data set. Calculate the test error for the classification rule obtained for the optimal K.

Before actually applying KNN we need to normalise the dataset because our data has different units and is on a different scale.

Writing a function for normalising our predictors: substracting mean value and diving the result by the standard deviation.

```
normalise <- function (inList){
  m <- mean(inList)
  s <- sd(inList)
  inList <- (inList - m)/s
  return(inList)
}
```

Applying this function to each of the columns in our test and traininig sets:

```
train.X <- apply(train.X, 2, normalise)
test.X <- apply(test.X, 2, normalise)
```

Now our predictors have values in the same range.

```
head(train.X)
```

```
##          upload     webget   enqcount   callwait
## [1,]  0.9790438  1.5490048 -0.6603278 -0.2912318
## [2,] -1.0678586 -0.7647049  1.7222982 -0.7052967
## [3,] -0.9637788 -1.0506624 -0.6603278  0.5692468
## [4,]  0.2851786 -0.6131415 -0.6603278 -0.1391924
## [5,] -1.4841777 -1.2034098 -0.1838026  0.2878121
## [6,]  0.2851786 -0.3704033  0.7692478  1.4167860
```

Writing a function for KNN with leave-one-out cross-validation. Leave-one-out is a special case of k-fold cross validation.

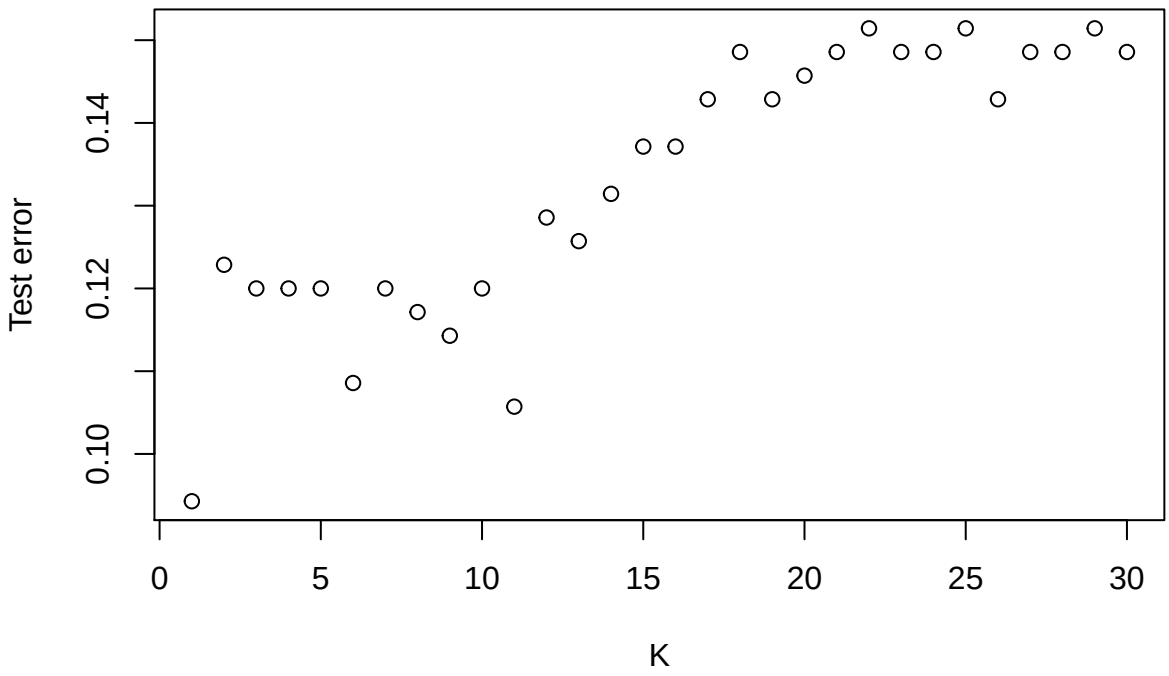
```
leave.KNN <- function(K, train.X, train.Y){
  error <- 0
  n <- nrow(train.X)
  # this function returns an error that is calculated as an average error of KNNs trained
  # using leave-one-out
        for(i in 1:n){
          # subsetting the i-th row from the train predictors and classifiers and using
          # them as temporary training sets (without an i-th row)
          temp.train.X <- train.X[-i,]
          temp.train.Y <- train.Y[-i]

          # using an i-th row as a temporary test set
          temp.test.X <- train.X[i,]
          temp.test.Y <- train.Y[i]

          # the resulting KNN is tested on only 1 entry
          temp.knn <- knn(
            train = temp.train.X,
            test = temp.test.X,
            cl = temp.train.Y, k = K)

          # the error is being calculated on whether a test entry was classified wrongly
          # or not and accumulated as we'll need to find the mean error in the end
          error <- error + mean(temp.knn != temp.test.Y)
            # 1 if the test entry was classified wrongly and 0 if correctly
        }

     return (error/n) #returning the average error
}
```

Now trying to find the optimal K in a range of values between 1 to 30. Running the *leave.KNN* function in a loop and storing the calculated in the function error in an array.

```
errors <- rep(0, 30) #trying with K from 1 to 30
for (j in 1:30) errors[j] <- leave.KNN(j, train.X, train.Y)
```

Plotting our errors

```
# plotting errors
plot(errors, xlab="K", ylab = "Test error")
```



Finding optimal K as an index of the first smallest error value

```
optim.K <- which.min(errors)
```

Now running KNN with the optimal K found from the leave-one-out cross-validation.

```
def.knn <- knn(train = train.X, test = test.X, cl = train.Y, k = optim.K) # yep that's it
tab <- table(def.knn, test.Y) # confusion table
```

Calculating the error using falsely predicted churn values

```
error.KNN <- (tab[1,2] + tab[2,1]) / sum(tab)
sprintf("Expected error: %f Test error: %f", min(errors), error.KNN)
```

```
## [1] "Expected error: 0.094286 Test error: 0.086667"
```

Part (d)

Using the training data set apply the random forest (bagging) method to construct a classifier to predict churn based on the four available predictors. Using the obtained random forest, comment on the importance of the four variables for predicting churn. Calculate the test error for the obtained random forest. Compare it to the test error found for the KNN classifier and provide an appropriate comment.

For random forest there's no need to use the created training set as it is due to the specifics of the function syntax (we just need to specify the training subset sequence, although optionally we can use the trainig subset itself). For us it is more convenient to work with dataframes so we will use the original churn_data.

```
testing.X <- churn_data[-num_subset, ] %>% subset(select = -churn)
testing.Y <- churn_data[-num_subset, ] %>% subset(select = churn)
testing.Y <- unlist(testing.Y)
```

Statistical Modelling

Load in the data and place in a data frame

```
# Data ---------------------------------------------------------------------
Patient_group <- c(1,2,3,4) #i
Dose <- c(422,744,948,2069) #di
Number_treated <- c(50,50,50,50) #ni
Number_better <- c(2,13,39,48) #yi

experiment_df <- data.frame(Patient_group, Dose, Number_treated, Number_better)
experiment_df
```

```
##   Patient_group Dose Number_treated Number_better
## 1             1  422             50             2
## 2             2  744             50            13
## 3             3  948             50            39
## 4             4 2069             50            48
```

Part (a)

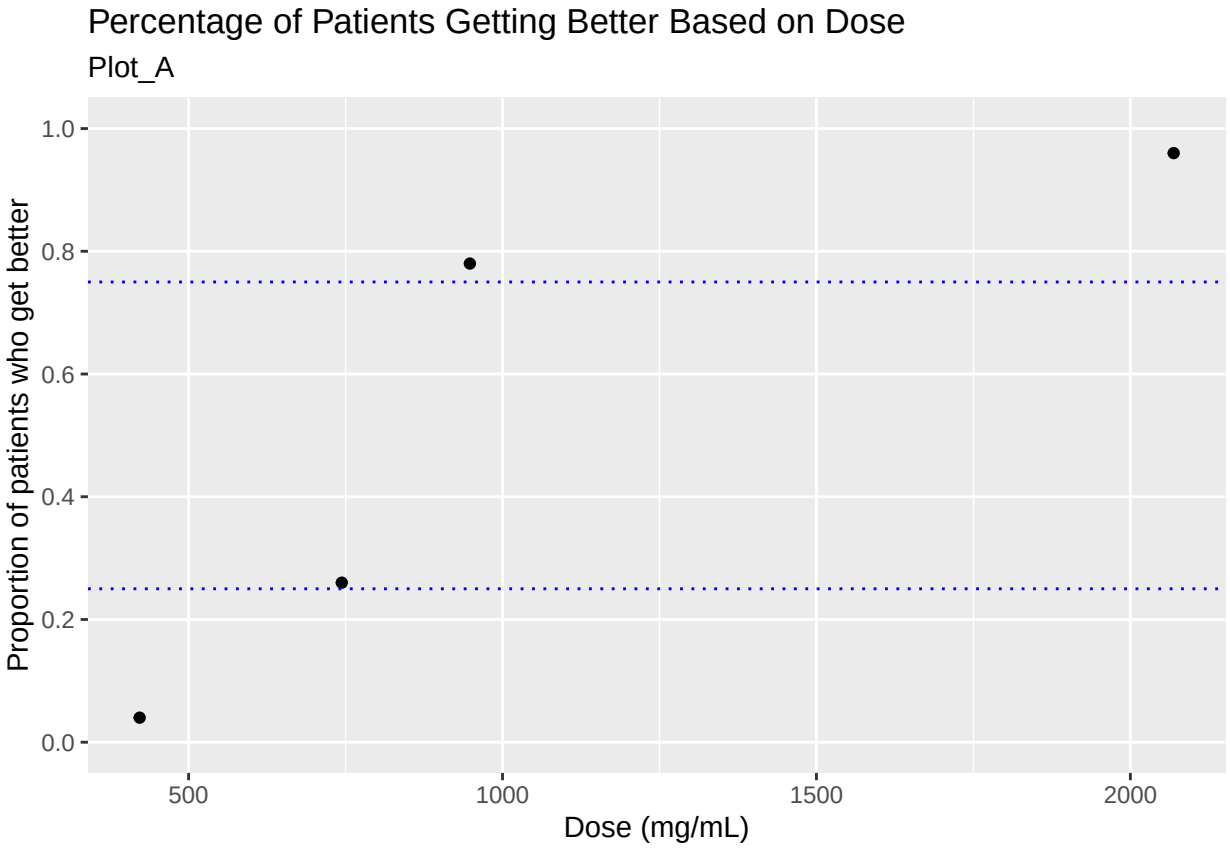
Calculate the proportion of patients who gets better with the new medicine and use ggplot2 to visualize these data.

Work out proportions with reduced blood pressure

```
experiment_df <- experiment_df %>% mutate(Proportion_reduced = Number_better / Number_treated)
```

Plot these proportions

```
Plot_A <-
  ggplot(experiment_df, aes(x = Dose, y = Proportion_reduced)) +
    geom_point() +
    geom_hline(yintercept=.25,linetype='dotted',col='blue')+
    geom_hline(yintercept=.75,linetype='dotted',col='blue')+
    labs(x = "Dose (mg/mL)",
          y = "Proportion of patients who get better",
          title = "Percentage of Patients Getting Better Based on Dose",
          subtitle = "Plot_A") +
    scale_y_continuous(breaks = c(0, 0.2, 0.4, 0.6, 0.8, 1),
                       minor_breaks = NULL,
                       limits = c(0, 1))
```



What can you conclude from the plot?

A dose of 500mg/ml or less has no real impact on the proportion of patients who get better (less than 4%). There is a reasonable increase at around the 750mg/ml as it reaches over 20% of the patients. However, the best increase is when the dose is increased to 1000mg/ml with nearly 80% of patients getting better. To increase the proportion of patients to just under 100%, it appears that the dose needs to be more than double to above 2000mg/ml. This is comparable with the current COVID vacinations which see a large increase from the first dose and almost complete immunity with a second dose.

Part (b)

Fit the model in the frequentist framework and report hat beta_0 (intercept) and hat beta_1 (Dose(x))

```
# Fitting the binary logistic regression model
m <- glm(cbind(Number_better,
               Number_treated - Number_better) ~ Dose,
         family = binomial,
         data = experiment_df)

# Maximum likelihood estimates - hat_beta_0 and hat_beta_1 of the parameters beta_0 and beta_1
beta_0_hat <- coef(m)[1]
beta_0_hat
```

```
## (Intercept)
##   -4.559752
```

```r
beta_1_hat <- coef(m)[2]
beta_1_hat
```

```
##        Dose
## 0.005271615
```

```r
# Confidence intervals for beta_0 and beta_1
confint(m)
```

```
## Waiting for profiling to be done...
```

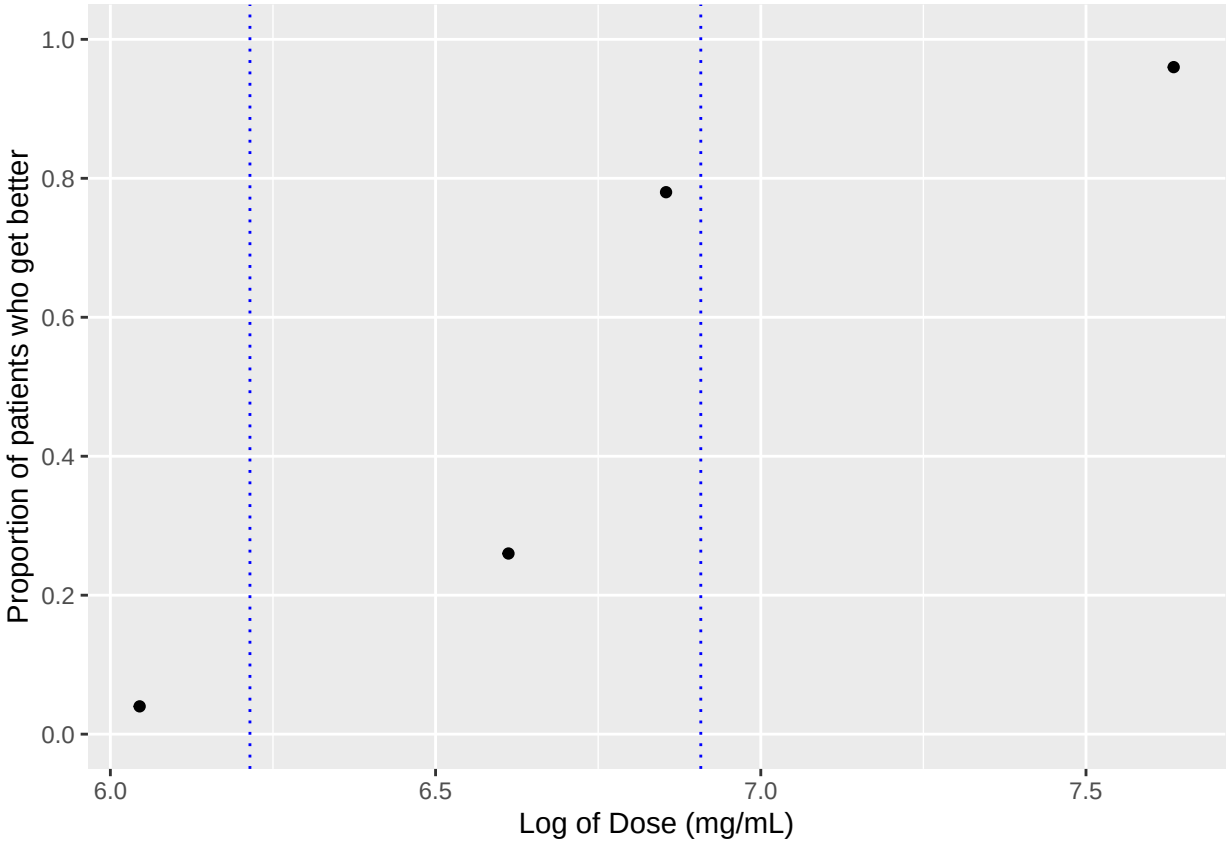
```
##                    2.5 %       97.5 %
## (Intercept) -6.336086850 -3.163180155
## Dose         0.003561412  0.007438245
```

Part (c)

Create a similar plot to that produced in part (a) to visualise log(di) against the proportion of Covid-19 patients who gets better and compare the two plots.

```r
# Work out the log of the dose and add to data frame
experiment_df <- experiment_df %>% mutate(Log_Dose = log(Dose))
```

```r
# Plot these proportions
ggplot(experiment_df, aes(x = Log_Dose, y = Proportion_reduced)) +
 geom_point() +
 geom_vline(xintercept=log(500),linetype='dotted',col='blue')+
 geom_vline(xintercept=log(1000),linetype='dotted',col='blue')+
 labs(x = "Log of Dose (mg/mL)",
      y = "Proportion of patients who get better") +
 scale_y_continuous(breaks = c(0, 0.2, 0.4, 0.6, 0.8, 1),
                    minor_breaks = NULL,
                    limits = c(0, 1))
```



What do you conclude?

INSERT ANSWER HERE ##