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
# converting classifier to a factor:
churn_data$churn <- as.factor(churn_data$churn)
```

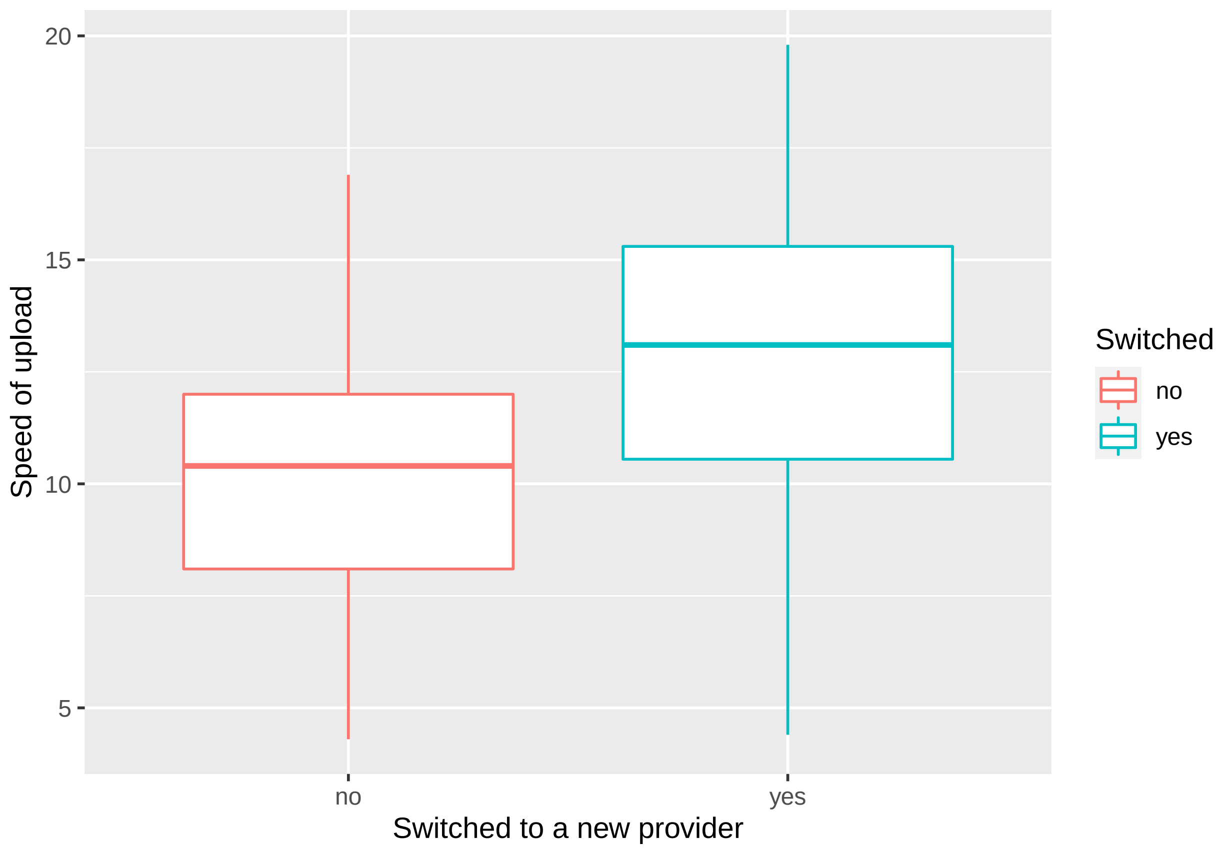
The data includes 4 predictors - 'webget', 'callwait', 'upload' and 'enqcount' and a classifier 'churn' which identifies whether a client has switched to a new operator or not.

```
head(churn_data)
```

```
##   upload webget enqcount callwait churn
## 1    9.2  283.9        5     8.14    no
## 2    7.0  298.4        6    11.59    no
## 3    6.6  163.8        5     8.25    no
## 4   15.0  566.8        2     9.50    no
## 5   11.1  210.3        5     6.96    no
## 6   15.4  857.0        2    10.80   yes
```

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

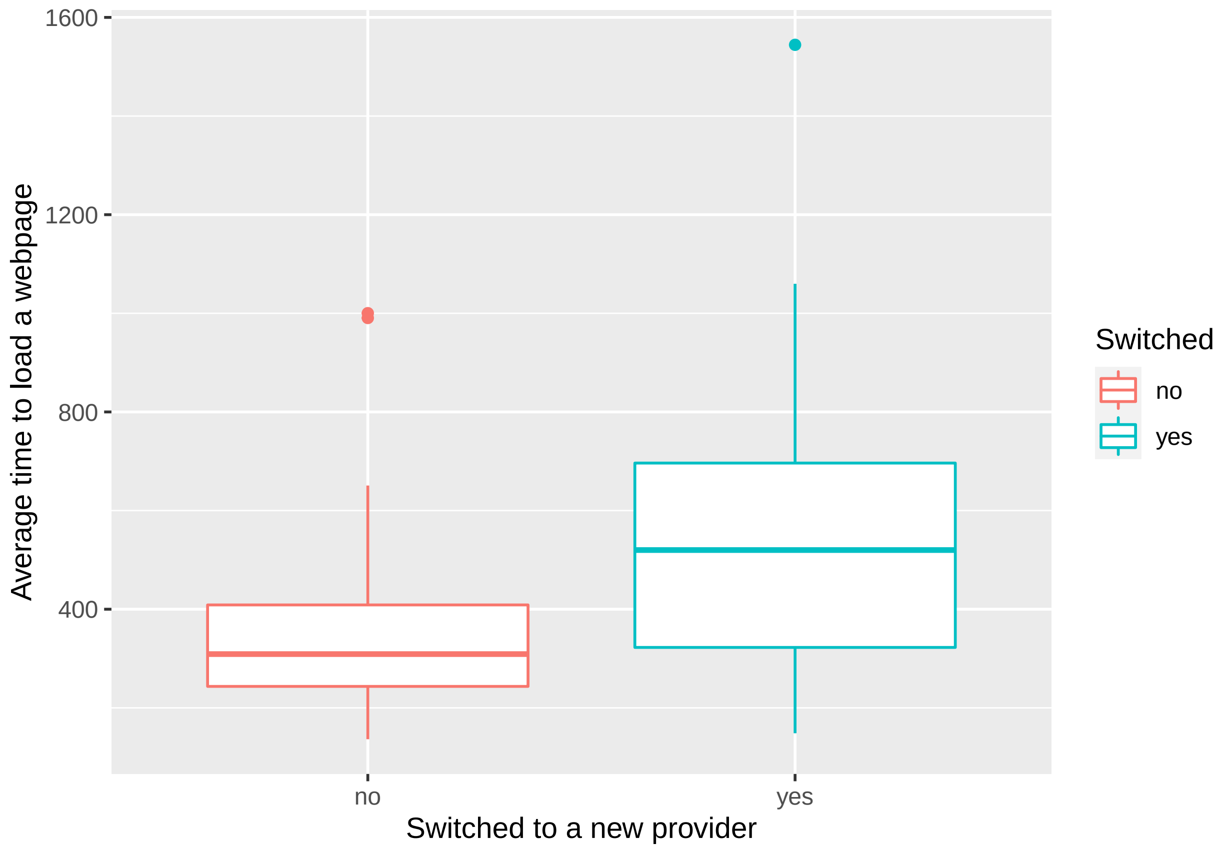
```
churn_data %>% ggplot(aes(x = churn, y = upload, color = churn)) +
 geom_boxplot() +
 labs (y = "Speed of upload",
       x = "Switched to a new provider",
       color = "Switched")
```



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:

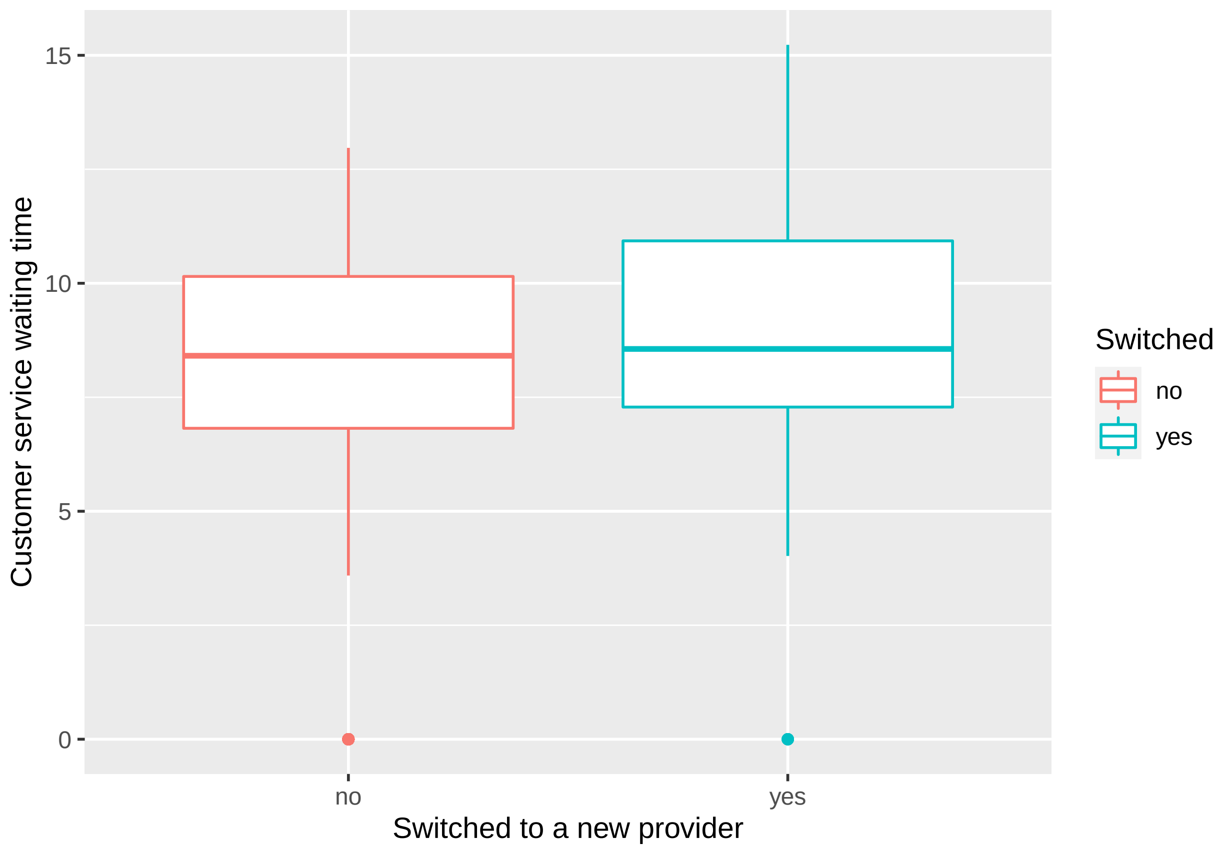
```
churn_data %>% ggplot(aes(x = churn, y = webget, color = churn)) +
 geom_boxplot() +
 labs (y = "Average time to load a webpage",
       x = "Switched to a new provider",
       color = "Switched")
```



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:

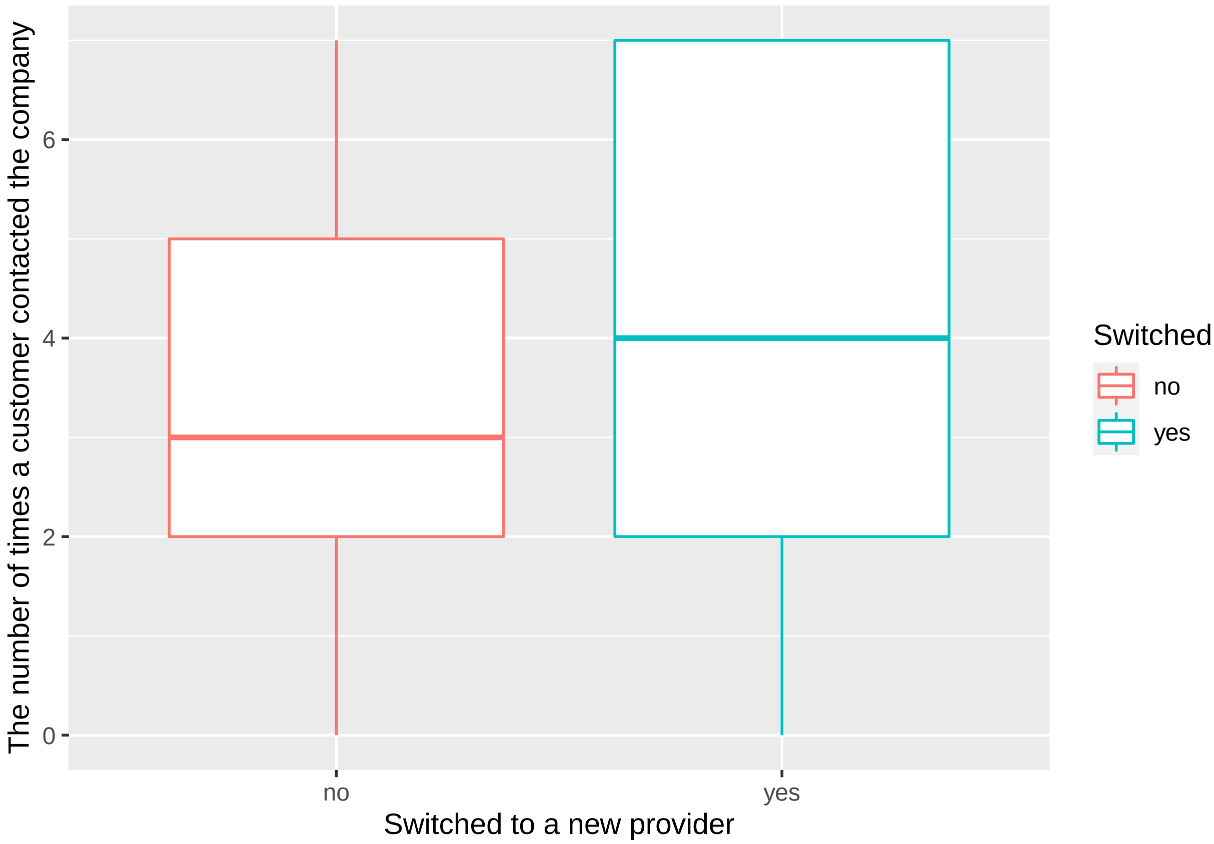
```
churn_data %>% ggplot(aes(x = churn, y = callwait, color = churn)) +
 geom_boxplot() +
 labs (y = "Customer service waiting time",
       x = "Switched to a new provider",
       color = "Switched")
```



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.

```r
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):

```r
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:

```r
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.

```r
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:

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

Now our predictors have values in the same range.

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

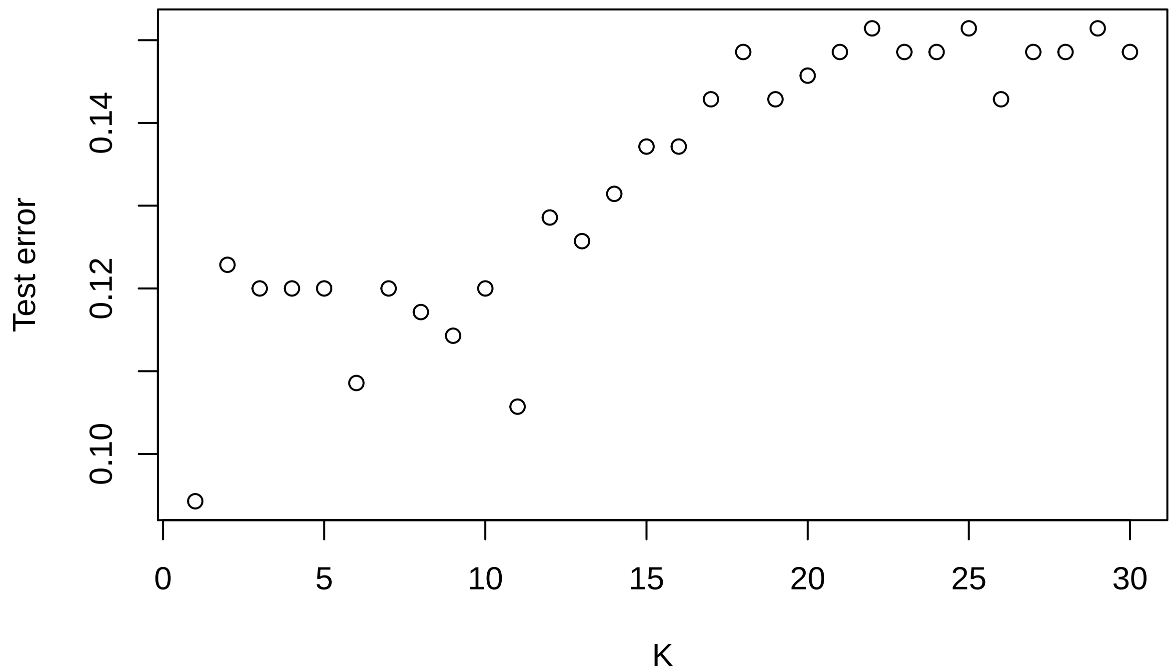
```r
leave.KNN <- function(K, train.X, train.Y){
  error <- 0
  n <- nrow(train.X)
  # this function returns an error that is calculated as an average error of KNNs trained
  # using leave-one-out
        for(i in 1:n){
          # subsetting the i-th row from the train predictors and classifiers and 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.

```r
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

```r
# 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.

```
def.knn <- knn(train = train.X, test = test.X, cl = train.Y, k = optim.K)
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"
```

Alternative solution

Instead of writing a custom take on leave-one-out approach we can use knn.cv function. The issue is that this function doesn't take a test set of predictors as an argument so we can calculate error only on the original training set.

```
alt.KNN <- function(k){
  res.knn.cv <- knn.cv(train.X, train.Y, k = 1)
  tab <- table(res.knn.cv, train.Y) # confusion matrix
```

```
  error <- (tab[1,2] + tab[2,1]) / sum(tab)
  return(error)
}
```

Finding the optimal K in the same way:

```
errors2 <- rep(0, 30) #trying with K from 1 to 30
for (j in 1:30) errors2[j] <- alt.KNN(j) # loop to find errors for K = 1:30
optim.K2 <- which.min(errors2) # finding K with the smallest error
optim.K2
```

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

We can see that in both cases optimal K=1.

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 training 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)
```

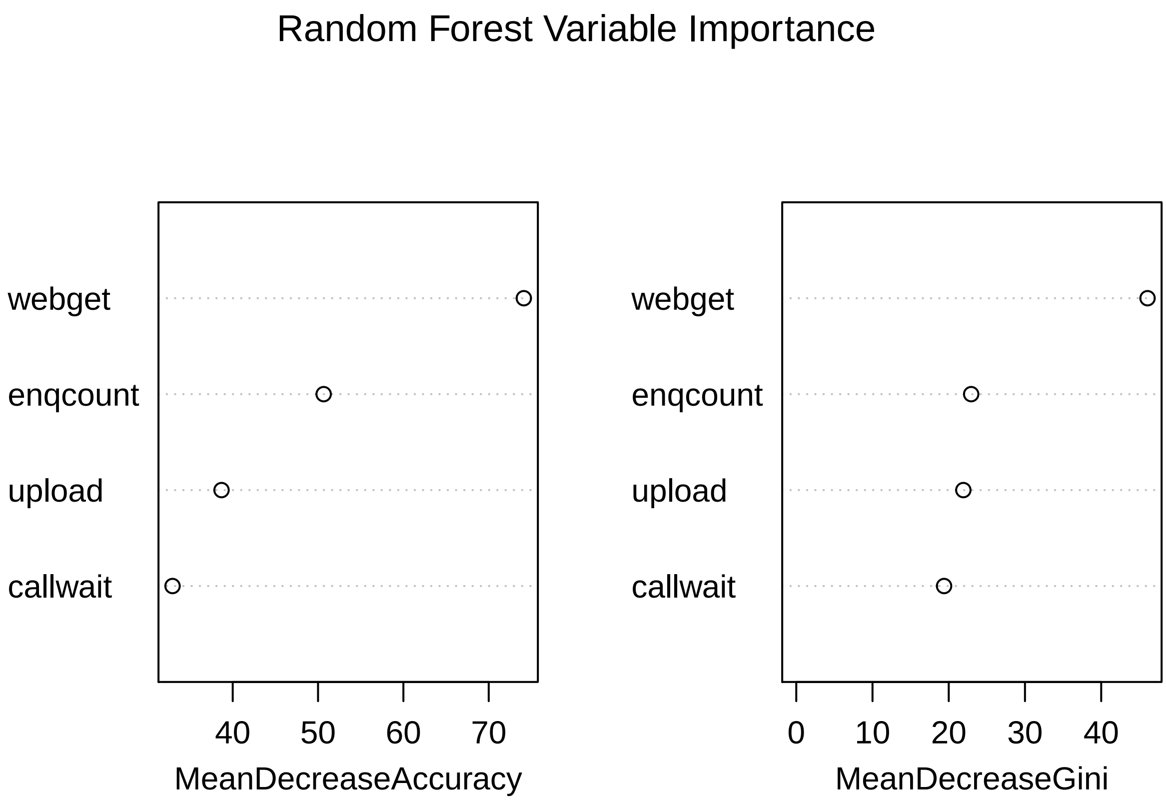
Constructing a random tree classifier based on 4 predictors. The 'mtry' variable normally can be equal to square root of the original number of predictors, hence mtry=2.

```
random.tree <- randomForest(churn ~ ., data = churn_data, subset = num_subset, mtry = 2, importance = T
```

We could use a different notation and specify the training set directly in randomForest function but using our subset sequence instead will save extra variables in global environment so we went for the option above.

Plotting mean decrease accuracy and mean decrease gini measures to identify the most important variables.

```
varImpPlot(random.tree, main = "Random Forest Variable Importance")
```



The Mean Decrease Accuracy plot expresses how much accuracy the model losses by excluding each variable. The more the accuracy suffers, the more important the variable is for the successful classification.The mean decrease in Gini coefficient is a measure of how each variable contributes to the homogeneity of the nodes and leaves in the resulting random forest. The higher the value of mean decrease accuracy or mean decrease Gini score, the higher the importance of the variable in the model.(Martinez-Taboada, F. & Redondo, J.I., 2020) Variable importance plot (mean decrease accuracy and mean decrease Gini) In our case, both MeanDecreaseAccuracy and MeanDecreaseGini measures indicate that 'webget' is ultimately the most important variable. The least important ones turned out to be 'callwait' followed by 'upload'. 'enqcount' (the number of customers enquiry calls to an operator) is the second most important predictor.

Calculating the test error for random forest classifier based on the confusion matrix produced from correctly and wrongly classified test entries:

```
rf.predict <- predict(random.tree, testing.X, type = "class")
tab <- table(rf.predict, testing.Y) # confusion matrix

error.RandomForest <- (tab[1,2] + tab[2,1]) / sum(tab)
```

Comparing the KNN and Random Forest errors:

```
## KNN Error: 0.08666667
```

```
## RandomForest Error: 0.02666667
```

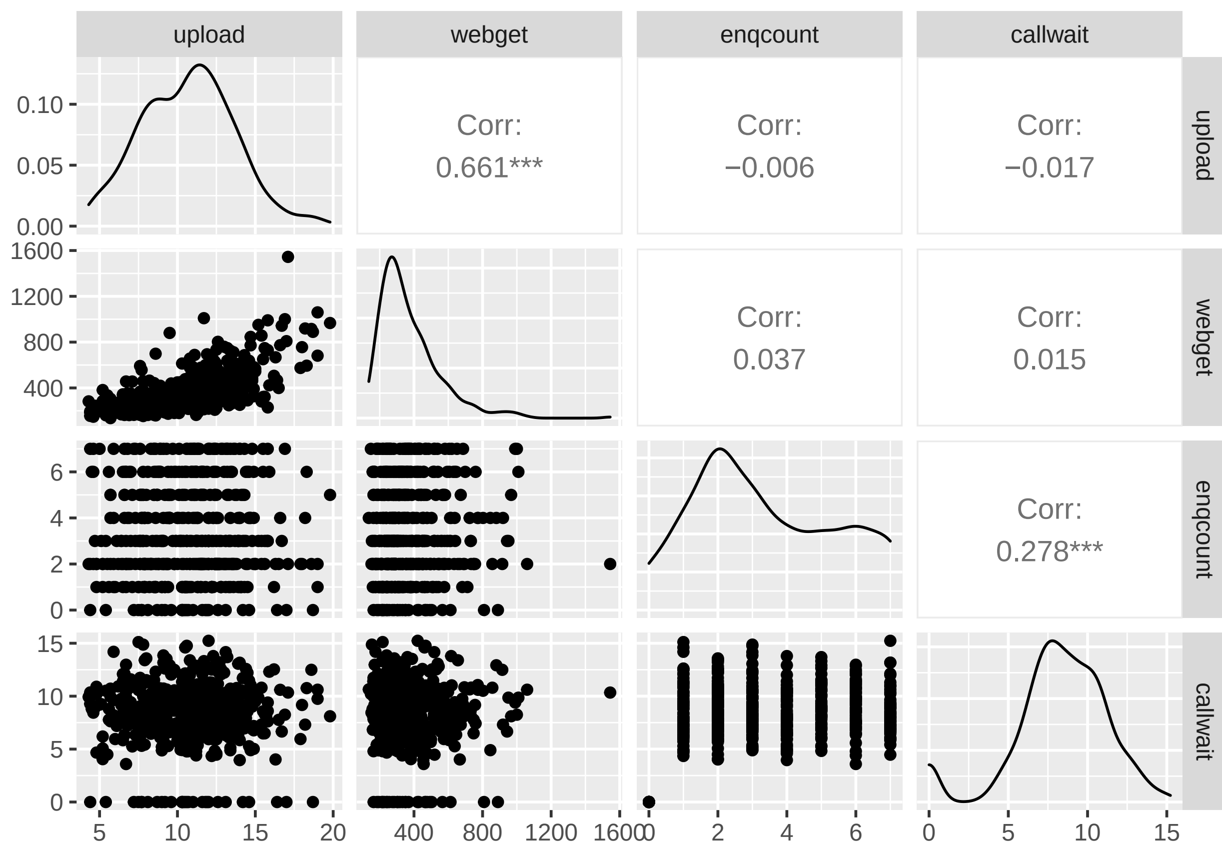
Random Forest method is more accurate than KNN with optimal K=1 with prediction error much lower than KNN. However, FUN FACT: 'keep.forest=FALSE' - this parameter removes the forest of trees from our random forest model. If this parameter is kept then it can heavily influence our prediction error.

Part(e)

Using the entire data set (training set and test set combined), perform Principal Component Analysis for the four variables: upload, webget, enqcount and callwait. Comment on the results. Using principal components, create the "best" two dimensional view of the data set. In this visualisation, use colour coding to indiciate the churn. How much of the variation or information in the data is preserved in this plot? Provide an interpretation of the first two principal components.

Separating the predictors from the dataframe to perform the Principal Component Analysis and building a graph to determine the correlation of predictors and the data spread.

```
churn_predictors <- churn_data[ , c("upload", "webget","enqcount", "callwait")]
ggpairs(churn_predictors)
```



CAN BE DISCARDED IF CRITICAL: The 'upload' and 'webget' variables have the biggest correlation coefficient 0.661. Since the number is closer to 1 we can assume that although weak there might be dependency between those 2 variables. The 'upload' data has the majority of records with speed between 7.5 and 12.5 units, while most of 'webget' entries' values rise up to 600 units. 'callwait' and 'enqcount' have second biggest but still a very weak correlation of 0.278. Most of the entries in 'enqcount' variable have at least one call and spread up until 7 calls with the highest number of entries in 2 calls while 'callwait' shows that most of the customers had to wait from 5 to 12.5 units (mins presumably) or on the contrary, we can observe that some of the customers waited for less than 2.5 units of time.

Overall the correlation between predictors is rather weak, 'upload', 'enqcount' and 'callwait' variables have high variation. Correlation between 'upload' and 'webget' predictors is on a stronger side within the current dataset since 0.661 is rather closer to 1.

Performing the PCA:

```r
churn_pca <- princomp(churn_predictors, cor = TRUE)
# the first column contains the names of cities so we exclude it.
summary(churn_pca)
```

```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3     Comp.4
## Standard deviation     1.2891552 1.1307782 0.8497143 0.58086584
## Proportion of Variance 0.4154803 0.3196648 0.1805036 0.08435128
## Cumulative Proportion  0.4154803 0.7351451 0.9156487 1.00000000
```

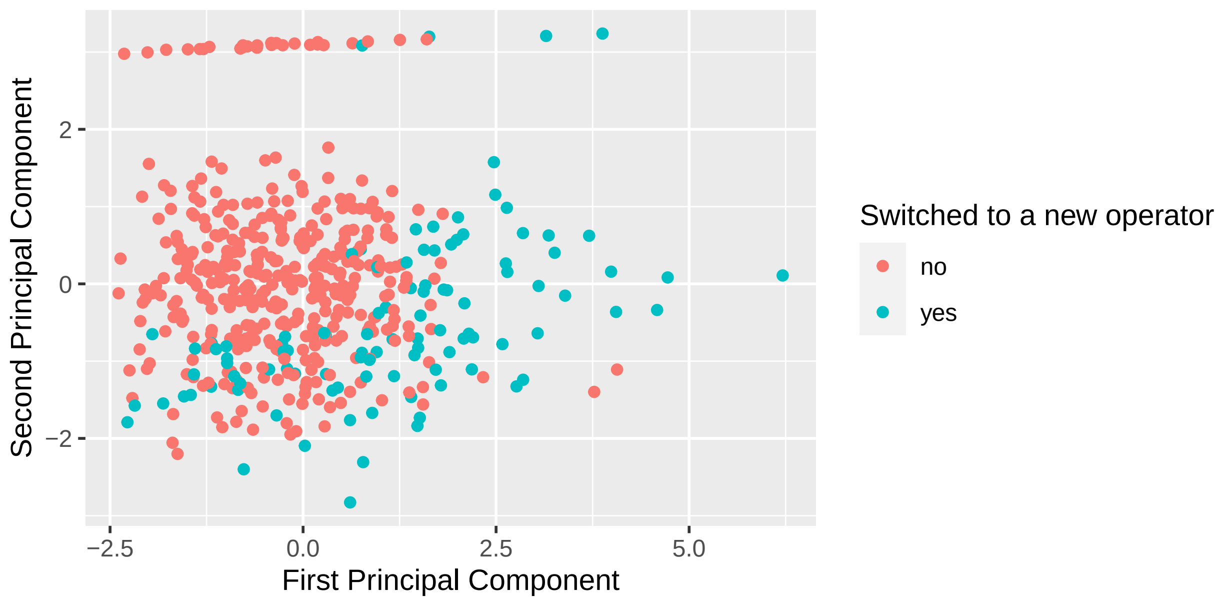
New variable Comp.1 holds 41.5% of variance in the data and has the deviation of 1.289, which is the biggest spread of the data. Comp. 2 accounts for 31.9% of the information variance giving a cumulative percentage of 73.5%. The 2 components contain different information and created by using data from variables with certain weights.

Let's create a table of new predictors, convert it to a dataframe and add a 'churn' classifier column, this way it is more convenient to plot.

```r
new_churn <- churn_pca$scores
new_churn <- data.frame(new_churn)
new_churn <- new_churn %>% mutate(churn = churn_data$churn)
```

In order to create a two-dimensional view of the data we'll use first 2 principal components and 'churn' variable to colour the points.

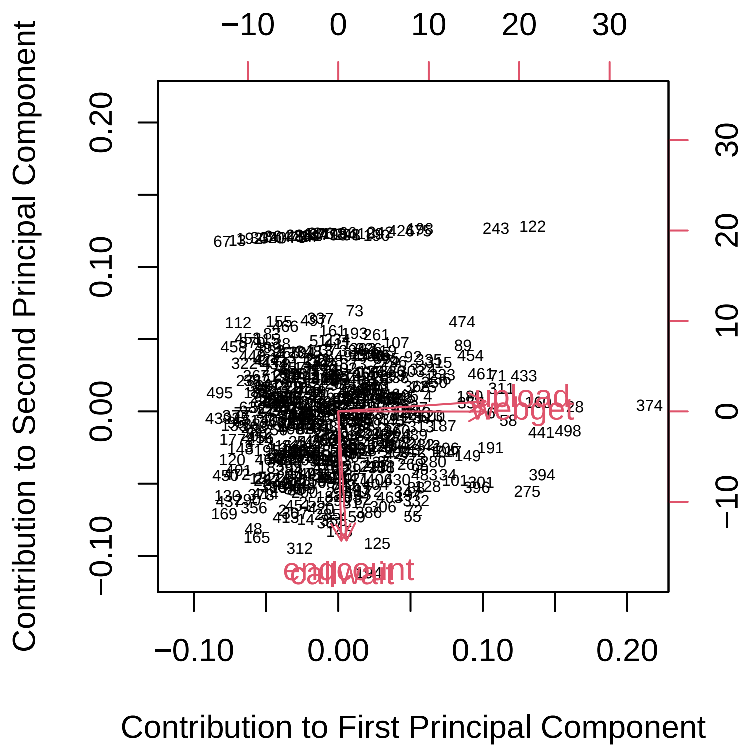
```r
new_churn %>% ggplot(aes(x = Comp.1, y = Comp.2, color = churn)) +
  geom_point() +
  labs(x = "First Principal Component",
       y = "Second Principal Component",
       color = "Switched to a new operator") +
  coord_fixed(ratio = 1)
```



Together, the first two principal components explain 73.5% of the variability.

Building a plot to demonstrate how the new components were created.

```
biplot(churn_pca, cex = c(0.5, 1),
       xlab = "Contribution to First Principal Component",
       ylab = "Contribution to Second Principal Component")
```



Upload and webget predictors have positive contribution to first principal component. Therefore, the component focuses on a customer's Internet speed. Callwait and enqcount have negative contribution to second principal component and so we can conclude that this component focuses more on the customer support experience.

##Part(f) Apply the random forest (bagging) method to construct a classifier to predict churn based on the two first principal components as predictors. In doing so, use the split of the data into a training and test set (you may use the same indices as in part (b)). Calculate the test error for the obtained random forest and comment on it. Visualise the resulting classification rule on the scatter plot of the two first principal components.

Forming a new dataset from 2 principal components with 'churn' classifier and preparing the data before using it in our random forest model. Separating a test set.

```
new_churn <- new_churn %>% subset(select = c(Comp.1, Comp.2, churn))
pca.test.X <- new_churn[-num_subset, ] %>% subset(select = -churn)
pca.test.Y <- churn_data[-num_subset, ] %>% subset(select = churn)
pca.test.Y <- unlist(pca.test.Y)
```

Training our random forest model. The mtry=1 as square root of 2 (number of predictors in our dataset) is approximately 1.4 hence closer to 1.

```
pca.random.tree <- randomForest(churn ~ ., data = new_churn, subset = num_subset, mtry = 1,
                                importance = TRUE)
```

Testing our trained model and calculating the test error.

```
pca.rf.predict <- predict(pca.random.tree, pca.test.X, type = "class")
tab <- table(pca.rf.predict, pca.test.Y) # confusion matrix
error.PCA.RandomForest <- (tab[1,2] + tab[2,1]) / sum(tab)
```

Displaying all 3 errors:

```
## KNN Error: 0.08666667
```

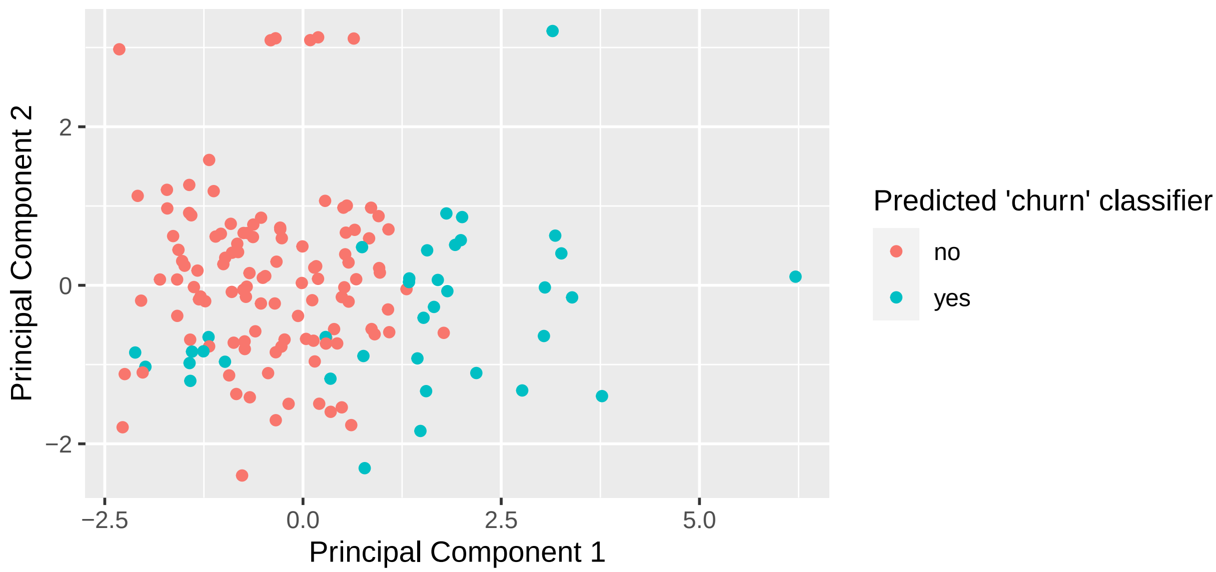
```
## RandomForest Error: 0.02666667
```

```
## RandomForest PCA Error: 0.1933333
```

As we can see, the random forest model trained on the data formed from 2 new principal components predicts the 'churn' classifier with 10% bigger error. Our assumption is that this happens because the two components altogether hold only 3/4 of the data variation. Perhaps if we could train our model on all 4 components the results would have been better but then there was no sense in doing principal component analysis in the first place. Since principal component analysis is good for reducing dimensionality by identifying the most important predictors and those that could be got rid of we can logically expect that it will influence the resulting model test error, the question is whether this change was positive or negative. Based solely on the test error we can assume that discarding 2 other principal components wasn't a good idea. One more suggestion on why the PCA didn't work well here is mentioned in the conclusions.

Visalising the resulting predicted classifiers with two principal components on the scatter plot.

```
pca.test.X %>% ggplot(aes(x = Comp.1, y = Comp.2, color = pca.rf.predict)) +
  geom_point() +
  coord_fixed(ratio = 1) +
  labs(x = "Principal Component 1", y = "Principal Component 2",
       color = "Predicted 'churn' classifier")
```



We can observe that most of the customers who have decided to switch to a different operator have positive values of principal component 1 which focuses on Internet speed factors like 'upload' and 'webget'.

CONCLUSIONS and ISSUES WITH THE DATASET

The data used in this section was flawed to begin with but not in terms of scale or missing values or even sparsity of the data. The problem lies in the meaning behind the predictors. We'd say that there's nothing wrong with Principal Component 2: both enqcount and callwait predictors (that contribute to this component) indicate the customer service quality. The bigger their values are the more negative experience a customer gets because no one wants to call to their operator with inquiries often or wait on the line for too long before a support agent gets on their call. However, let's take a closer look at the principal component 1 together with upload and webget predictors. webget indicates the time which takes for a client to load a webpage, which directly means a downlink speed: the bigger the 'webget' value is the lower is the downlink Internet speed so higher webget values actually logically carry negative meaning. Now, upload predictor is easier - we have classic 'the more the better' here since it indicates the uplink speed directly. BUT both of those predictors contribute to component 1 positively, which creates contradiction - normally you would want the 'webget' to have a negative contribution because the smaller this value is the better Internet speed a customer has (basically the same case with enqcount and callwait for component 2).

We can't blame PCA function here - we can only blame the nature of the provided data because it is as confusing as it can be. Instead, it would be better to have the 'webget' predictor converted to the same unit as 'upload' predictor. Unfortunately, the units of the predictors were not provided which hopefully didn't influence our machine learning models but is very crucial for understanding the data and any data scientists should understand the data before actually working with it. Despite that, working with this dataset was a great experience.

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)

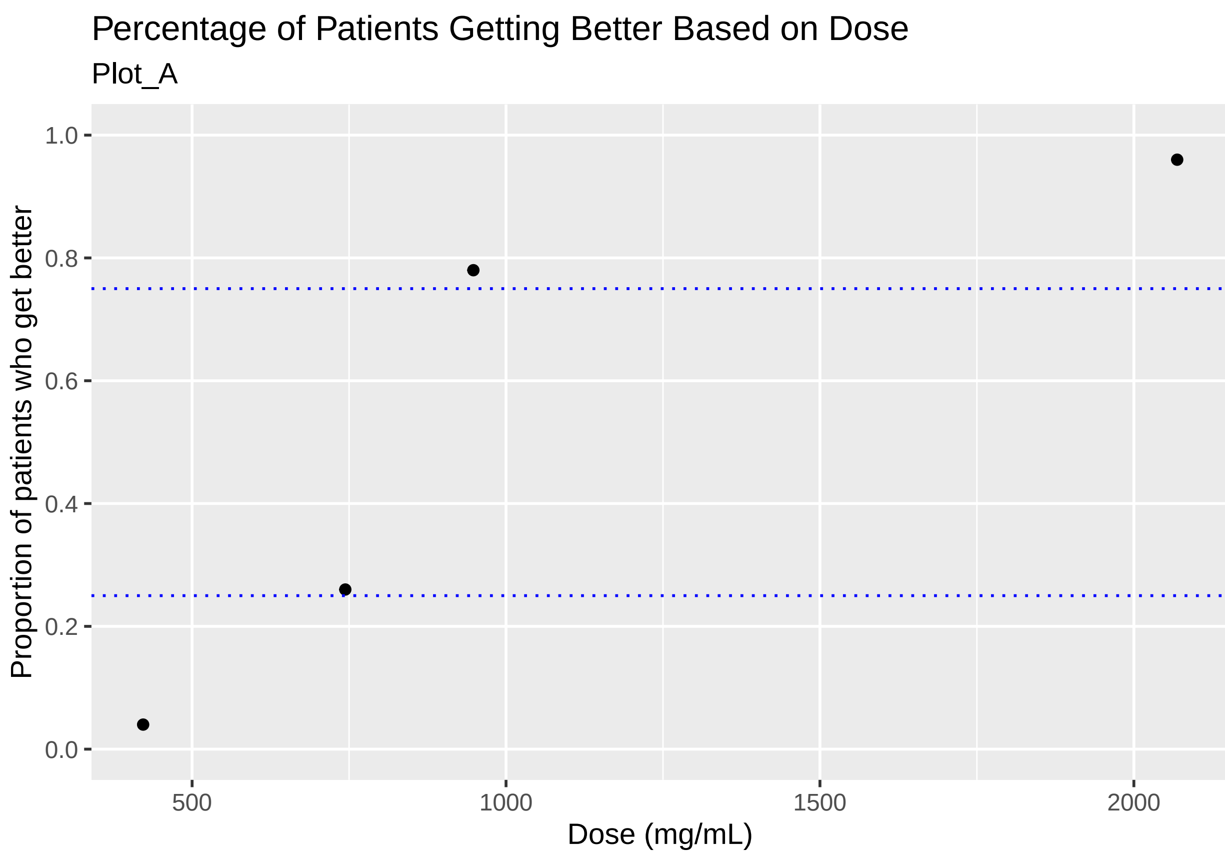
Calculate the proportion of patients who gets better with the new medicine and use ggplot2 to visualize these data.

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...
```

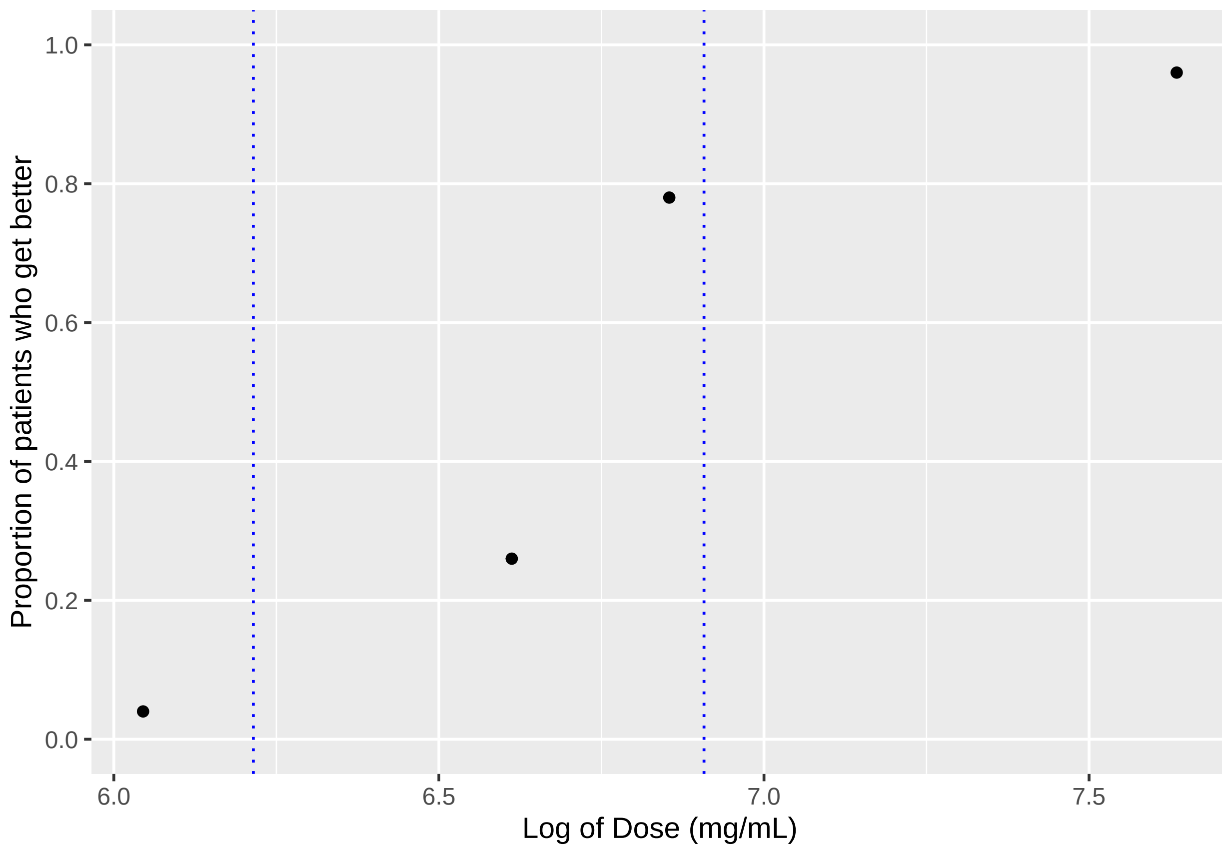
```
##                    2.5 %       97.5 %
## (Intercept) -6.336086850 -3.163180155
## Dose         0.003561412  0.007438245
```

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 ##