MATH501 Coursework - Report

10570155, 10696253, 10701983

26/04/2021

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

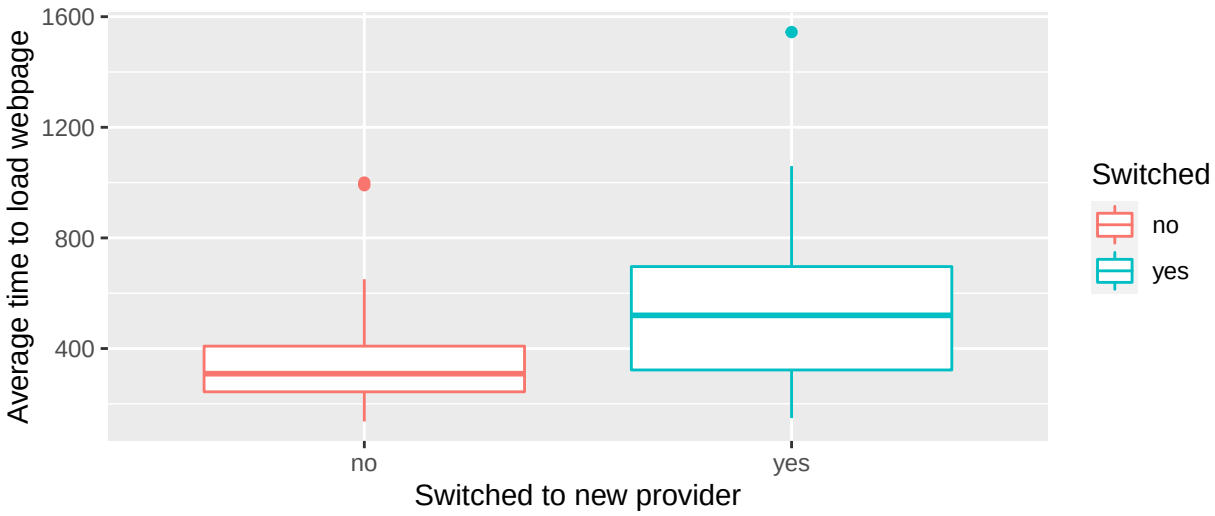
The data includes 4 predictors - 'webget', 'callwait', 'upload' and 'enqcount' and a classifier 'churn' which identifies whether a client has switched to a new operator or not.

```
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 the mean time to load a webpage against an indicator whether a customer switched to a different provider

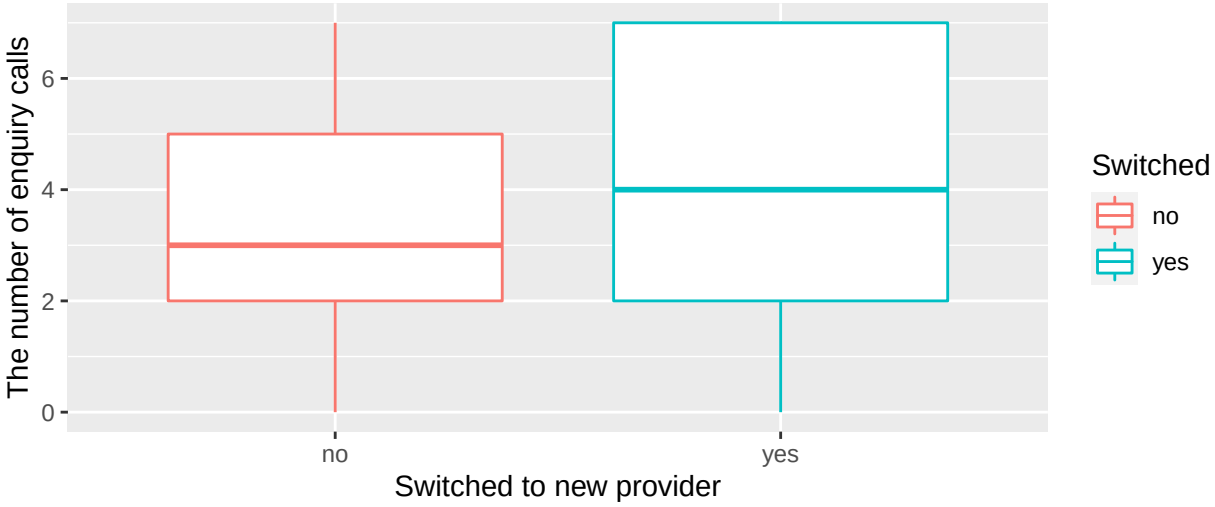
```
churn_data %>% ggplot(aes(x = churn, y = webget, color = churn)) + geom_boxplot() +
 labs(y="Average time to load webpage",x="Switched to new provider",color="Switched")
```



We can presume that there is 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 (around 550 units) than for those who haven't (around 350 units). 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. To clarify, the longer it takes to load a webpage the lower is the Internet downlink speed. Downlink is often more important as people spend more time loading media content and browsing web pages which solely depends on download speed.

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 enquiry calls",x="Switched to new provider",color="Switched")
```



On the boxplot above we can observe that on average the customers that have switched to a different operator contacted the company via a phone call at least 1 time more often than the ones who haven't with 4 and 3 calls for both types of customers respectively. The biggest number of contact attempts is 7 which is 2 calls more than of the customers who haven't changed their providers. It can be observed from the data that some of the customers who switched didn't contact the company even once. On the contrary, a minority of the customers who haven't changed their provider also have more than 5 calls to the customer

service team. It is safe to conclude that the number of calls impacts the customers' decision to choose a different operator but its importance is smaller comparing to the time to load a webpage.

Conclusion

Please refer to .rmd file to see other 2 boxplots. Out of the 4 factors, provided in the dataset, that can influence the 'churn' variable, time to load the webpage (which subsequently leads to the downlink speed) is the most important one. Average call waiting time for customer service doesn't differ drastically but still is higher for customers who chose different provider. Thus, we could conclude that this aspect also plays its part in the customers' decision along with the number of times the customer 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 normally 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 other 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 - Training Set
train.Y <- Y[num_subset]
test.X <- X[-num_subset, ] # 150 records - Test Set
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

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 by subtracting mean value and dividing 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 training 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
```

Find the optimal K using leave-one-out cross-validation for the training data set

Writing a custom 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
```

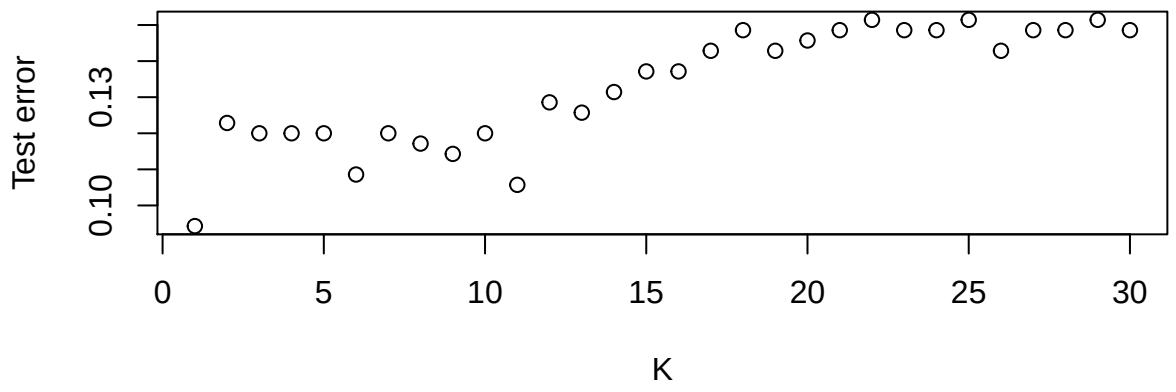
```
        }
    return (error/n) #returning the average error
}
```

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.

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



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

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

Here we find that the optimal K=1.

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
```

Calculate the test error for the classification rule obtained for the optimal K

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 only calculate the error 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
```

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

For random forest, there is no need to use the created training set, as it is due to the specifics of the function syntax. We only 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.

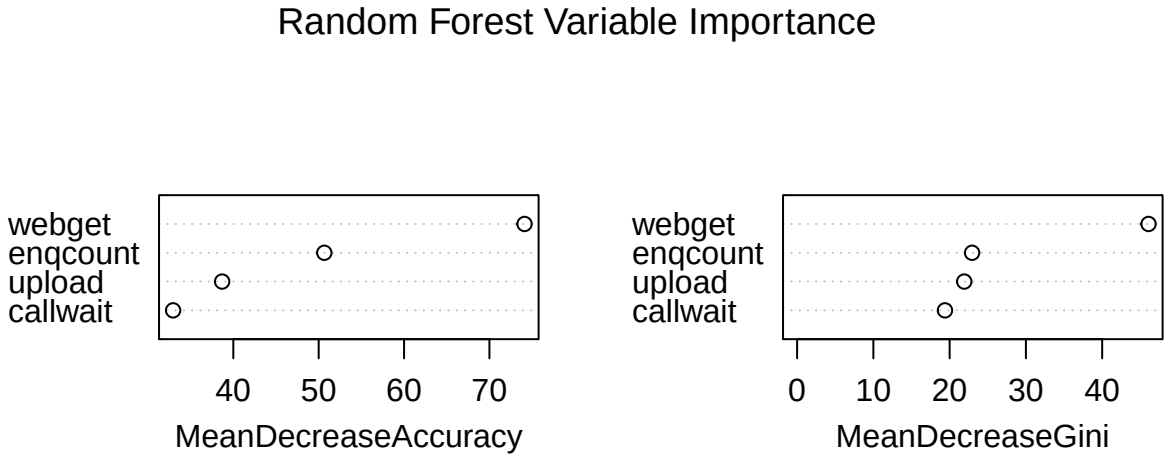
```
random.tree <-
  randomForest(churn ~ ., data=churn_data, subset=num_subset, mtry=2, importance=TRUE)
```

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 the global environment, so we went for the option above.

Using the obtained random forest, comment on the importance of the four variables for predicting churn

Plotting Mean Decrease Accuracy (MDA) and Mean Decrease Gini (MDG) measures to identify the most important variables.

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



The MDA 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 MDG 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 MDA or MDG score, the higher the importance of the variable in the model (Martinez-Taboada, F. & Redondo, J.I., 2020). In our case, both MeanDecreaseAccuracy and MDG measures indicate that 'webget' is ultimately the most important variable. Followed by 'enqcount' (the number of customers enquiry calls to an operator) as the second most important predictor with the least important ones being 'callwait' followed by 'upload'.

Calculate the test error for the obtained random forest

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

```r
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)
```

Compare it to the test error found for the KNN classifier and provide an appropriate comment

Comparing the KNN and Random Forest errors:

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

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

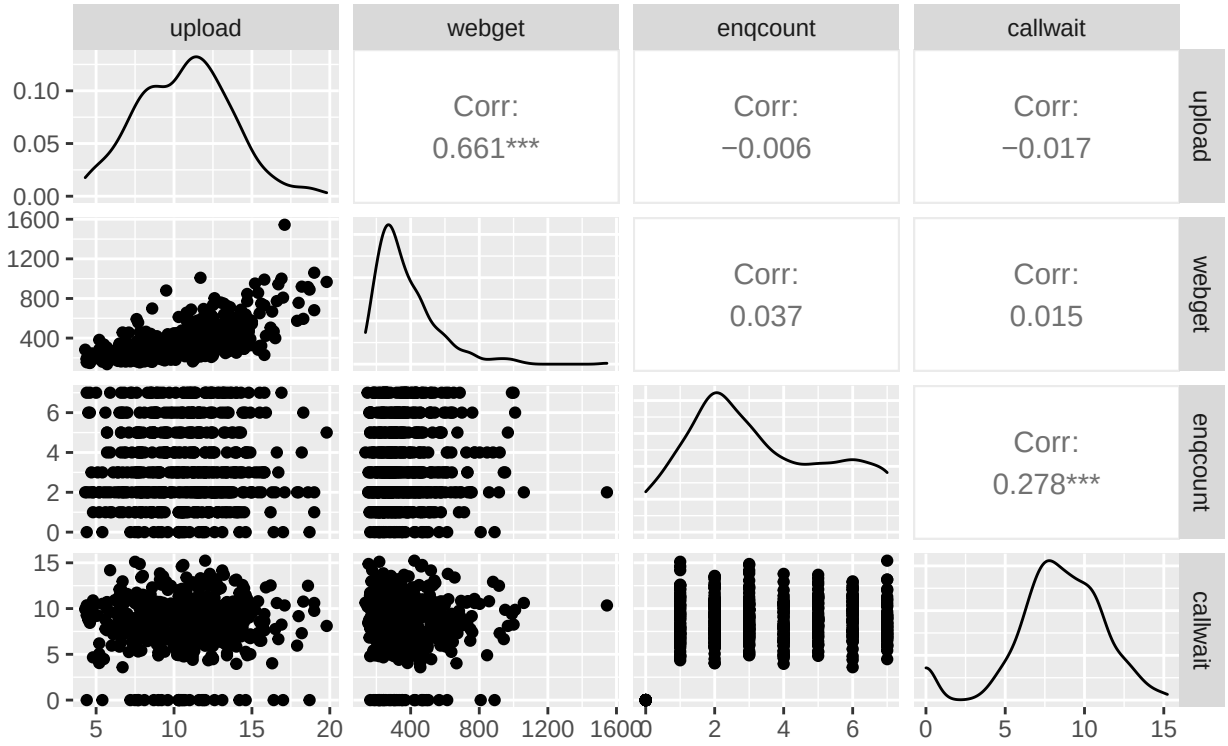
Random Forest method is more accurate than KNN with optimal K=1 with prediction error much lower than KNN. However, FUN FACT: utilising the parameter 'keep.forest=FALSE' removes the forest of trees from our random forest model. If this parameter is kept then it can heavily influence our prediction error.

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.

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)
```



Overall, the correlation between predictors is rather weak. The 'upload', 'enqcount' and 'callwait' variables have high variation, while in the case of 'webget', most of the entries' values rise up to 600 units. 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.

Performing the PCA.

```
churn_pca <- princomp(churn_predictors, cor = TRUE)
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 1.2891552, 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 are created by using data from variables with certain weights.

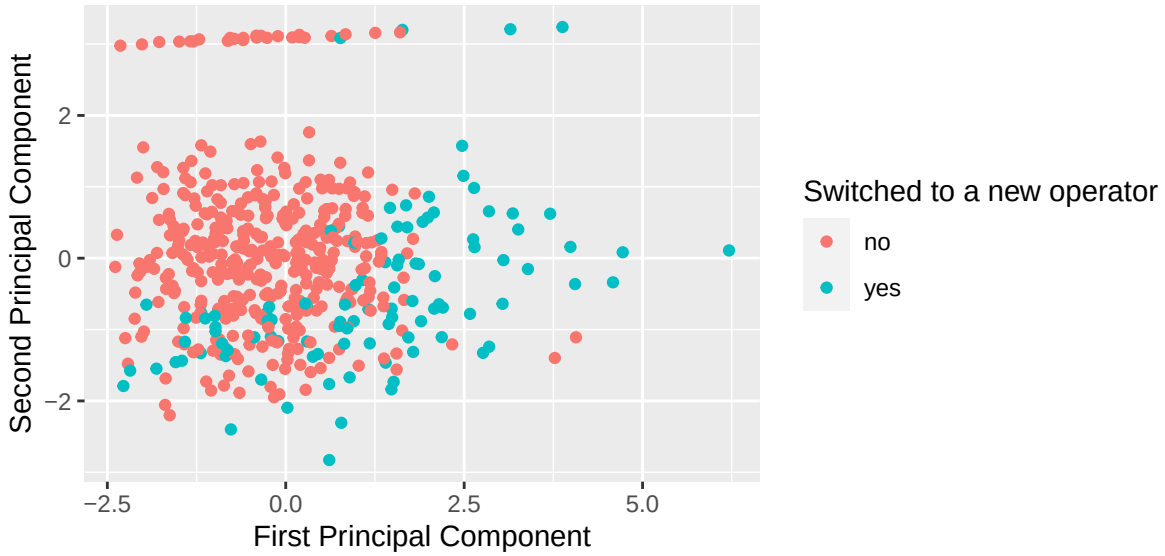
Using principal components, create the "best" two dimensional view of the data set. In this visualisation, use colour coding to indicate the churn.

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.

```
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 the first two principal components and the 'churn' variable to colour the points.

```
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)
```

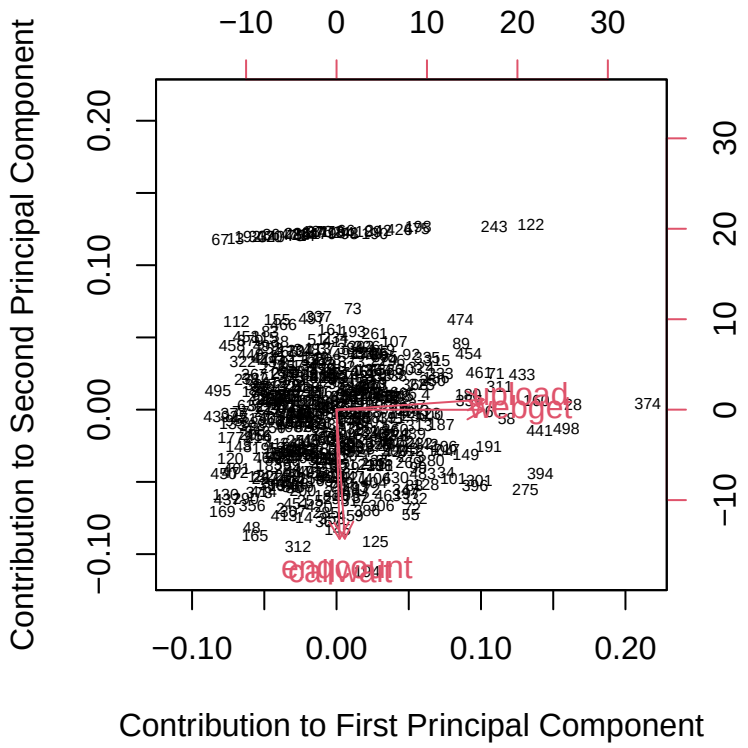


How much of the variation or information in the data is preserved in this plot? Provide an interpretation of the first two principal components.

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 the first principal component. Therefore, the component focuses on a customer's internet speed. 'Callwait' and 'enqcount' have negative contributions to the 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)).

Forming a new dataset from the two principal components with the '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)
```

Calculate the test error for the obtained random forest and comment on it.

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 the two new principal components predicts the 'churn' classifier with a 10% larger error. Our assumption is that this happens because the two components together 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.

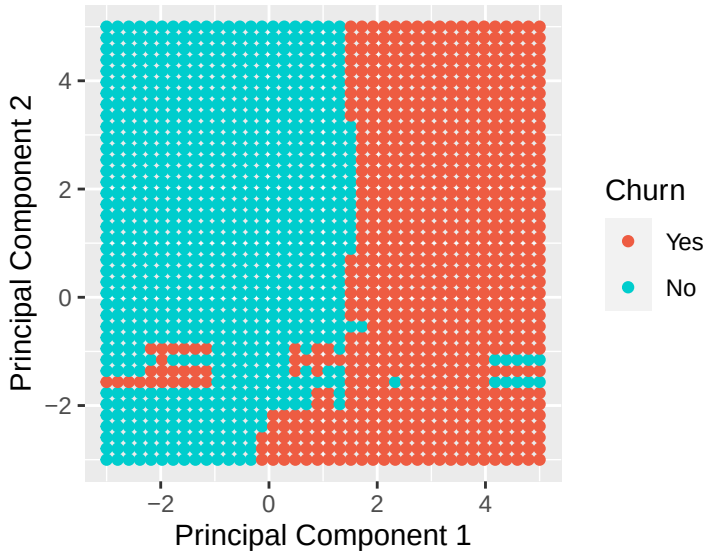
Since principal component analysis is good for reducing dimensionality by identifying the most important predictors and removing non-important predictors, we can logically expect that it will influence the resulting model test error. The question is whether this change was positive or negative. Based solely on the test error, we can assume that discarding two other principal components was not a good idea. One more suggestion on why the PCA didn't work well here is mentioned in the conclusions.

Visualise the resulting classification rule on the scatter plot of the two first principal components.

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

```
len <- length(pca.test.Y)
# creating a base grid to plot predictions
xgrid <- seq(-3,5, length.out = 40)
ygrid <- seq(-3,5, length.out = 40)
xygrid.predict <- expand.grid(Comp.1 = xgrid, Comp.2 = ygrid)
# constructing a classifier using random forest model
xygrid.class <- predict(pca.random.tree, xygrid.predict, type = "class")
col <- rep("grey", length(xygrid.class))
for (i in 1:length(xygrid.class)){
  if (xygrid.class[i] == 'yes')
  {
    col[i] = "green"
  } else
  {
    col[i] = "red"
  }
}

xygrid.predict %>% ggplot(aes(x = Comp.1, y = Comp.2, color = col)) + geom_point() +
```

```
  coord_fixed(ratio = 1) +
  labs(x = "Principal Component 1", y = "Principal Component 2",
       color = "Churn") +
  scale_color_manual(labels = c("Yes", "No"), values = c("tomato2", "cyan3"))
```



We can observe that most of the customers who have decided to switch to a different operator have values higher than around 1 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 would say that there is 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 has because no one wants to call to their operator with inquiries often or wait on the line for too long before a support agent answers them.

However, taking a closer look at the Principal Component 1 together with 'upload' and 'webget' predictors, we see that 'webget' indicates the time which it takes for a client to load a webpage, in other words downlink or download speed. The bigger the 'webget' value is, the lower the downlink speed is. Thus, higher 'webget' values should logically carry a negative meaning. Now the 'upload' predictor is easier to interpret, we have a 'the more the better' situation here, since it indicates the uplink speed directly. However, both of those predictors contribute to Principal Component 1 positively, which creates a 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 Principal Component 2).

We can't blame the 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 has not influenced our machine learning models, but are 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 learning experience.

Statistical Modelling

Load in the data and place in a data frame for use.

```
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)

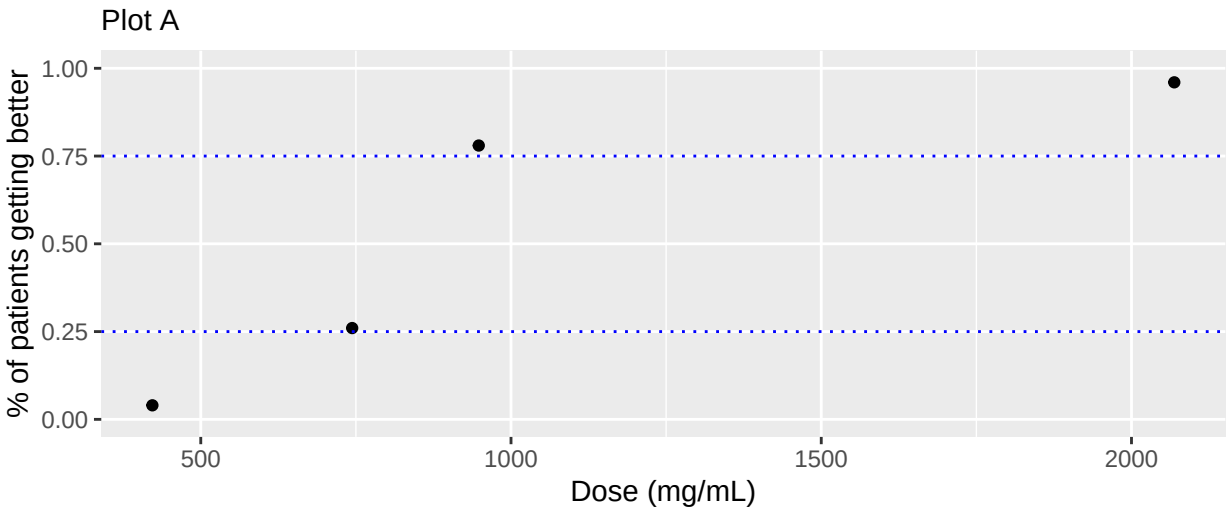
Calculate the proportion of patients who gets better with the new medicine.

Work out the proportion of patients who got better and add it to the dataframe for analysis

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

Use ggplot2 to visualize these data.

```
Plot_A <-
  ggplot(experiment_df, aes(x = Dose, y = Proportion_better))+ geom_point()+
  geom_hline(yintercept=.25,linetype='dotted',col='blue')+
  geom_hline(yintercept=.75,linetype='dotted',col='blue')+
  labs(x="Dose (mg/mL)",y="% of patients getting better",subtitle="Plot A")+
  scale_y_continuous(breaks=c(0,0.25,0.5,0.75,1),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 getting better. 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 vaccinations, which sees a large increase from the first dose and almost complete immunity with a second dose.

With this graph, we can also conclude that improving recovery from 80% to near 100% would require twice the dosage and most likely then, twice the cost. The cost analysis would therefore become important in determining the dosage to provide beyond 1000mg/ml.

Part (b)

Fit the model in the frequentist framework and report $\hat{\beta}_0$ (intercept) and $\hat{\beta}_1$ (Dose(x))

```
m <- glm(cbind(Number_better,Number_treated - Number_better) ~ Dose,
               family = binomial, data = experiment_df)
```

```
coef(m)[1] #beta_0_hat
```

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

```
coef(m)[2] #beta_1_hat
```

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

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

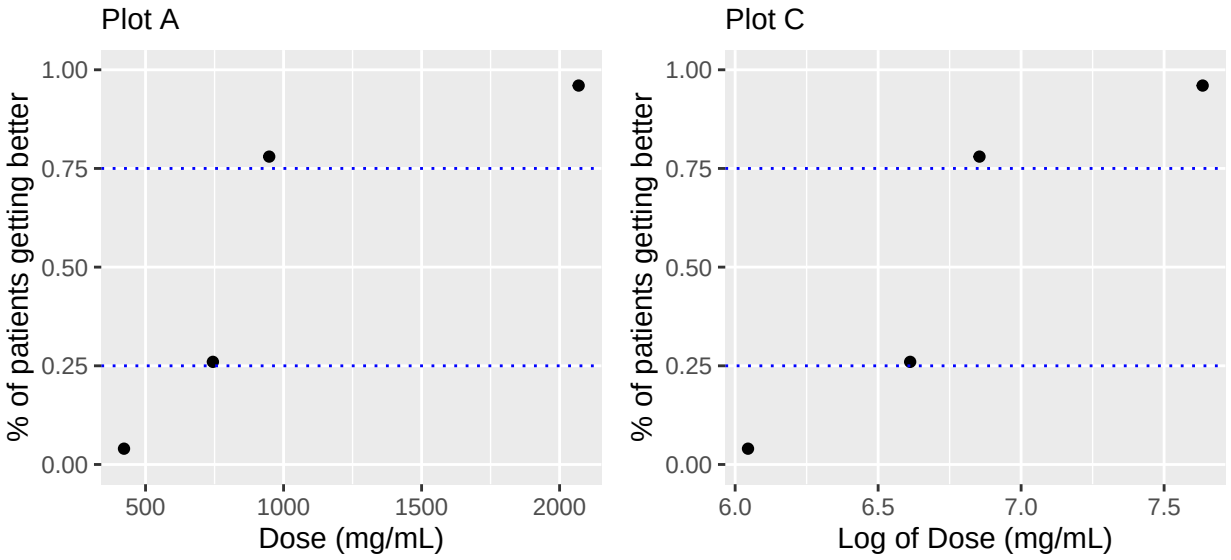
```
##                    2.5 %       97.5 %
## (Intercept) -6.336086850 -3.163180155
## Dose         0.003561412  0.007438245
```

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_C <- #Log(Dose) plot
  ggplot(experiment_df, aes(x = Log_Dose, y = Proportion_better))+  geom_point()+
  geom_hline(yintercept=.25,linetype='dotted',col='blue')+
  geom_hline(yintercept=.75,linetype='dotted',col='blue')+
  labs(x="Log of Dose (mg/mL)",y="% of patients getting better",subtitle="Plot C")+
  scale_y_continuous(breaks=c(0,0.25,0.5,0.75,1),minor_breaks=NULL,limits=c(0,1))
```



What do you conclude?

Plot A allows us to see the actual dose that relates to each point easily in terms that we understand (mg/ml). Plot C is confusing as we are unable, at a glance, to determine what dosage leads to which percentage of patients getting better.

However, Plot C does help to spread the points across the graph, which would be more useful if we had a lot of data points to show. With the current dataset, this is not really helping and so Plot A would be an easier graph to show to people, as most would understand the axes descriptions and meanings.

Overall, we can start to visualise the curve of patient improvement as dosage increases from the graphs and more so in Plot C.

Fit the logarithmic model in the frequentist framework, report beta_0_hat and beta_1_hat

```
m_log <- glm(cbind(Number_better, Number_treated - Number_better) ~ Log_Dose,
         family = binomial, data = experiment_df)
coef(m_log)[1] #beta_0_hat
```

```
## (Intercept)
##     -32.639
```

```
coef(m_log)[2] #beta_1_hat
```

```
## Log_Dose
## 4.852825
```

Calculate the 95% confidence intervals for β_0 and β_1.

```
confint(m_log) #Confidence intervals for beta_0 and beta_1
```

```
##                   2.5 %    97.5 %
## (Intercept) -44.701647 -23.66464
## Log_Dose      3.515515   6.64896
```

Which of the two frequentist models considered in parts (b) and (c) (the standard or the logarithmic model) do you prefer? Why?

Due to the size of the data set, it is difficult to see any difference in the graph as using a log in this case would help with data being compacted and needing to be scaled out in magnitude. As such there is no real benefit to this model. This model does however produce a more confusing graph as the x-axis is in units of Log of Dose which is more confusing that just the standard in graph a of dose as mg/ml. Plot A will be easier to read by medical professionals, as well as general people, due to the accessibility of this label. In a time of overwhelming data, accessibility is key to helping people understand data and provide information.

Part (d)

Use the logarithmic model implemented in part (c) to predict the probabilities that Covid-19 patients who receive doses of 600, 800 and 1500 mg/mL of the medicine get better. In addition, calculate the 95% confidence intervals for each prediction.

```
## Indirect Method
Dose_new <- c(log(600),log(800),log(1500))

# Estimated value of eta and associated standard error at Dose_new
eta_hat <- predict(m_log, newdata = data.frame(Log_Dose = Dose_new),
               se.fit = TRUE) # Extract the standard errors

# Approximate 95% confidence interval for eta
```

```r
eta_estimate_ci <- c(estimate =eta_hat$fit,
                     lower = eta_hat$fit - 2 * eta_hat$se.fit,
                     upper = eta_hat$fit + 2 * eta_hat$se.fit)

# Convert to a confidence interval for p
p_estimate_ci_indirect <- exp(eta_estimate_ci) / (1 + exp(eta_estimate_ci))

data.frame(exp(Dose_new), p_estimate_ci_indirect)
```

```
##            exp.Dose_new. p_estimate_ci_indirect
## estimate.1           600             0.16856755
## estimate.2           800             0.45022971
## estimate.3          1500             0.94535927
## lower.1              600             0.09747458
## lower.2              800             0.35451920
## lower.3             1500             0.85999239
## upper.1              600             0.27567417
## upper.2              800             0.54977155
## upper.3             1500             0.97989239
```

What can you conclude from these results?

As the dosage increases, the probability of the patient getting better increases.

Compare the predictions and confidence intervals obtained using both the indirect and direct methods.

```r
## Direct Method
p_hat_direct <- predict(m_log, newdata = data.frame(Log_Dose = Dose_new),
               type = "response", # Prediction of p, not eta
               se.fit = TRUE) # Extract the standard errors

# Confidence interval for p
p_estimate_ci_direct <- c(estimate = p_hat_direct$fit,
                          lower = p_hat_direct$fit - 2 * p_hat_direct$se.fit,
                          upper = p_hat_direct$fit + 2 * p_hat_direct$se.fit)

data.frame(exp(Dose_new), p_estimate_ci_direct)
```

```
##            exp.Dose_new. p_estimate_ci_direct
## estimate.1           600           0.16856755
## estimate.2           800           0.45022971
## estimate.3          1500           0.94535927
## lower.1              600           0.08030064
## lower.2              800           0.35134670
## lower.3             1500           0.89186736
## upper.1              600           0.25683447
## upper.2              800           0.54911272
## upper.3             1500           0.99885118
```

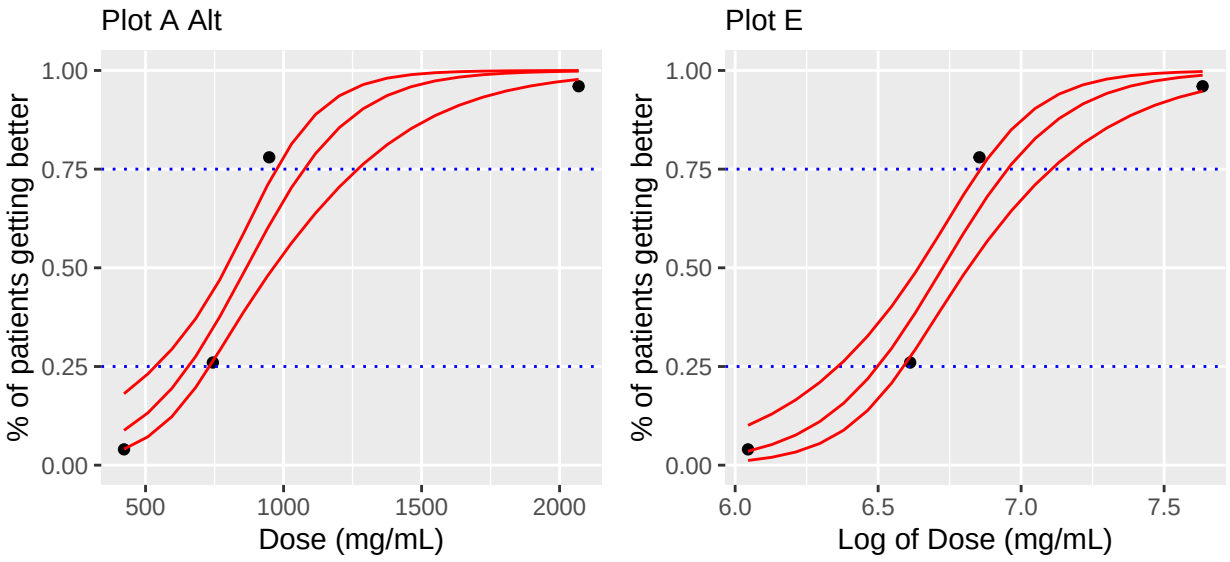
Which one of the two methods is generally recommended? Why?

It is generally recommended to use the indirect method as it provides more accurate results more regularly than the direct method. This is not evident in the current data set, but this is most likely due to its small size.

Part (e)

Use the logarithmic model implemented in part (c) to produce the plot below, with 95% confidence intervals obtained using the indirect method, and comment on it.

```
Dose_seq_log <- seq(from = min(log(Dose)), to = max(log(Dose)), length = 20)

# Obtain the estimated values of eta and standard errors at a sequence of Dose values
eta_seq_log <- predict(m_log, newdata = data.frame(Log_Dose = Dose_seq_log),
                       se.fit = TRUE) # Extract the standard errors

# Transform estimated values of eta to probabilities
p_seq_log <- exp(eta_seq_log$fit) / (1 + exp(eta_seq_log$fit))

#  Confidence intervals for eta, saved in a matrix
eta_seq_log_ci <- cbind(eta_seq_log$fit - 2 * eta_seq_log$se.fit, # Lower limits
                        eta_seq_log$fit + 2 * eta_seq_log$se.fit) # Upper limits

# Convert to confidence intervals for p
p_seq_log_ci <- exp(eta_seq_log_ci) / (1 + exp(eta_seq_log_ci))

# Put all these values into a data frame
predictions_df_log <- data.frame(Dose_seq_log,
                             p_seq_log = p_seq_log, # Estimated values of p
                             p_lower_log = p_seq_log_ci[,1], # Lower confidence limits
                             p_upper_log = p_seq_log_ci[,2]) # Upper confidence limits

# Add to the ggplot
Plot_E <-
  ggplot(experiment_df, aes(x = Log_Dose, y = Proportion_better)) + geom_point() +
  geom_hline(yintercept=.25,linetype='dotted',col='blue')+
  geom_hline(yintercept=.75,linetype='dotted',col='blue')+
  # Add the estimated values of p (predictions)
  geom_line(aes(x=Dose_seq_log,y=p_seq_log),data=predictions_df_log,colour="red")+
  # Add the lower confidence limits
  geom_line(aes(x=Dose_seq_log,y=p_lower_log),data=predictions_df_log,colour="red")+
  # Add the upper confidence limits
  geom_line(aes(x=Dose_seq_log,y=p_upper_log),data=predictions_df_log,colour="red")+
  labs(x = "Log of Dose (mg/mL)",y = "% of patients getting better", subtitle="Plot E")+
  scale_y_continuous(breaks=c(0,0.25,0.5,0.75,1),minor_breaks=NULL,limits=c(0,1))
```



In addition, perform a suitable statistical test to check whether the logarithmic model is adequate. State your hypotheses and conclusions carefully.

Null hypothesis - H_0: Model C provides an adequate fit to the data. (The model as it stands is useful)

Alternative hypothesis - H_1: Model C does not provide an adequate fit to the data. (The model as it stands is NOT useful)

```
p_value <- pchisq(deviance(m_log),df.residual(m_log),lower = FALSE)
p_value
```

```
## [1] 0.01389397
```

P-value is less than 0.05 and thus the null hypothesis is rejected, signifying that this model is not an adequate fit to the data or in other words is not useful.

Part (f)

Fit the quadratic model in the frequentist framework and report $\hat{\beta}_0$, $\hat{\beta}_1$ and $\hat{\beta}_2$

```
m_log_log2 <-
  glm(cbind(Number_better, Number_treated - Number_better) ~ Log_Dose + I(Log_Dose^2),
      family = binomial,
      data = experiment_df)

beta_0_hat <- coef(m_log_log2)[1]
  beta_0_hat
```

```
## (Intercept)
##   -96.84529
```

```
beta_1_hat <- coef(m_log_log2)[2]
  beta_1_hat
```

```
## Log_Dose
## 23.72948
```

```
beta_2_hat <- coef(m_log_log2)[3]
  beta_2_hat
```

```
## I(Log_Dose^2)
##     -1.385234
```

Perform a frequentist hypothesis test of size 0.05 of whether β_2 is statistically significant and report your conclusion with justification. State your hypotheses and conclusions carefully.

Null hypothesis - H_0: Model F provides an adequate fit to the data. (The model as it stands is useful)

Alternative hypothesis - H_1: Model F does not provide an adequate fit to the data. (The model as it stands is NOT useful)

```
p_value <- pchisq(deviance(m_log_log2),df.residual(m_log_log2),lower = FALSE)
p_value
```

```
## [1] 0.00754795
```

P-value is less than 0.05 and thus the null hypothesis is rejected, signifying that this model is not an adequate fit to the data or in other words is not useful.

In addition, report the 95% confidence interval for beta_2. Does this result confirm the conclusion of the hypothesis test?

```
tail(confint(m_log_log2),1)
```

```
##                    2.5 %    97.5 %
## I(Log_Dose^2) -4.028886 0.9388293
```

Part (g)

Use the analysis of Deviance method to compare the logarithmic model fitted in part (c) with the quadratic model fitted in part (f). State your hypotheses and conclusions carefully.

```
Deviance_c_log <- deviance(m_log)
Deviance_c_log
```

```
## [1] 8.552601
```

```r
Deviance_f_log_log2 <- deviance(m_log_log2)
Deviance_f_log_log2
```

```
## [1] 7.13771
```

```r
#  Difference in deviances
Deviance_c_log - Deviance_f_log_log2
```

```
## [1] 1.414891
```

The difference between Model C Deviance and Model F Deviance is positive, so we can conclude that it is adequate.

Compare the models

Null hypothesis - H_0: Model C is adequate compared to Model F

Alternative hypothesis - H_1: Model C is not adequate compared to Model F

```r
AnovaTest <- anova(m_log, m_log_log2, test = "Chisq")
AnovaTest[2,5]
```

```
## [1] 0.2342461
```

Here, the p-value is 0.23. As this is greater than 0.05, we do not reject the null hypothesis and so Model C is adequate compared to Model F. In other words $\beta_2 = 0$. Thus, Model C is preferred over Model F.

Part (h)

Write jags/BUGS code to perform inference about the following related Bayesian binary logistic regression model. Run your code. Include a graphical representation of the traceplots and posterior densities of β_0 and β_1 in your report and discuss your results.

```r
# Creating the model
bayesian_binary_logistic_model <- function() {
  for (i in 1:n_obs) {
    y[i] ~ dbin(p[i], n[i]) # binomial distribution
    logit(p[i]) <- eta[i] # link function
    eta[i] <- beta_0 + beta_1 * log(x[i])
  }
  # Specify prior distributions on the unknown parameters
  beta_0 ~ dnorm(0.0, 1.0E-4)
  beta_1 ~ dnorm(0.0, 1.0E-4)
  # Evaluate the linear predictor at the new value of dose (x_new)
  eta_new <- beta_0 + beta_1 * log(x_new)
  # Convert to the probability scale
  p_new <- exp(eta_new) / (1 + exp(eta_new))
}
```

```r
# Performing Bayesian inference
bayesian_binary_logistic <-
  jags(data = task_h_data,parameters.to.save = c('beta_0', 'beta_1', 'p', 'p_new'),
       n.iter = 100000,n.chains = 3,model.file = bayesian_binary_logistic_model)
```

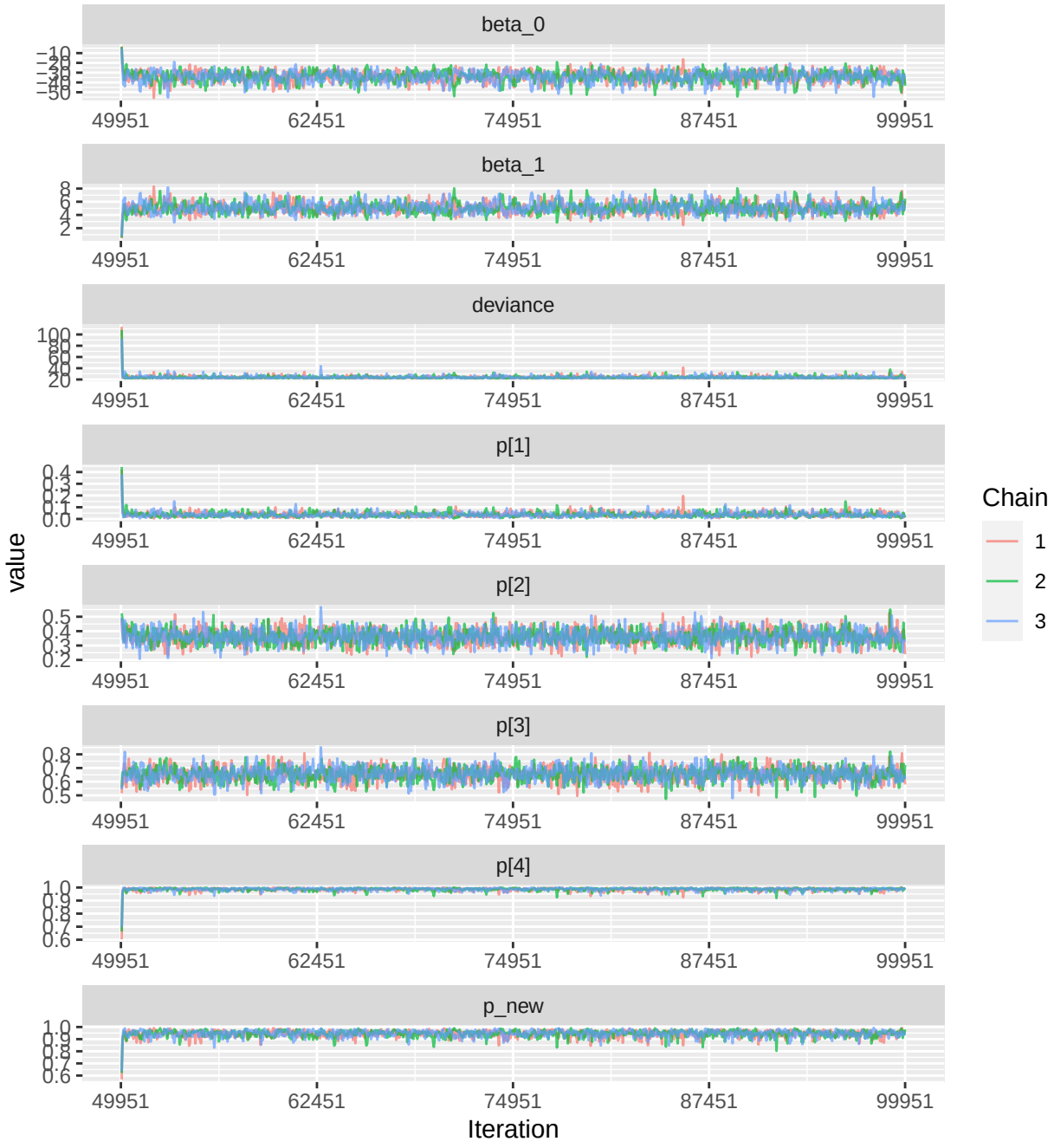
```r
# Print numerical results of the model
print(bayesian_binary_logistic, intervals = c(0.025, 0.5, 0.975))
```

```
## Inference for Bugs model at "C:/Users/tania/AppData/Local/Temp/RtmpEz6Wow/model40b43e0173b6.txt", fi
##  3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 50
##  n.sims = 3000 iterations saved
##          mu.vect sd.vect    2.5%     50%   97.5%  Rhat n.eff
## beta_0   -34.068   5.613 -46.373 -33.614 -24.249 1.002  1100
## beta_1     5.066   0.835   3.603   5.000   6.891 1.002  1300
## p[1]       0.036   0.023   0.009   0.032   0.083 1.002  1600
## p[2]       0.362   0.050   0.265   0.361   0.461 1.001  3000
## p[3]       0.656   0.052   0.553   0.656   0.756 1.002  1600
## p[4]       0.987   0.014   0.960   0.990   0.998 1.009  3000
## p_new      0.946   0.028   0.881   0.950   0.984 1.002  1700
## deviance  23.957   3.353  21.860  23.182  29.906 1.002  1800
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 5.6 and DIC = 29.6
## DIC is an estimate of expected predictive error (lower deviance is better).
```

We can see that zero does not appear within the credible intervals for β_0 and β_1, which indicates no posterior support at zero. This means that both parameters are useful in our model.

Traceplots

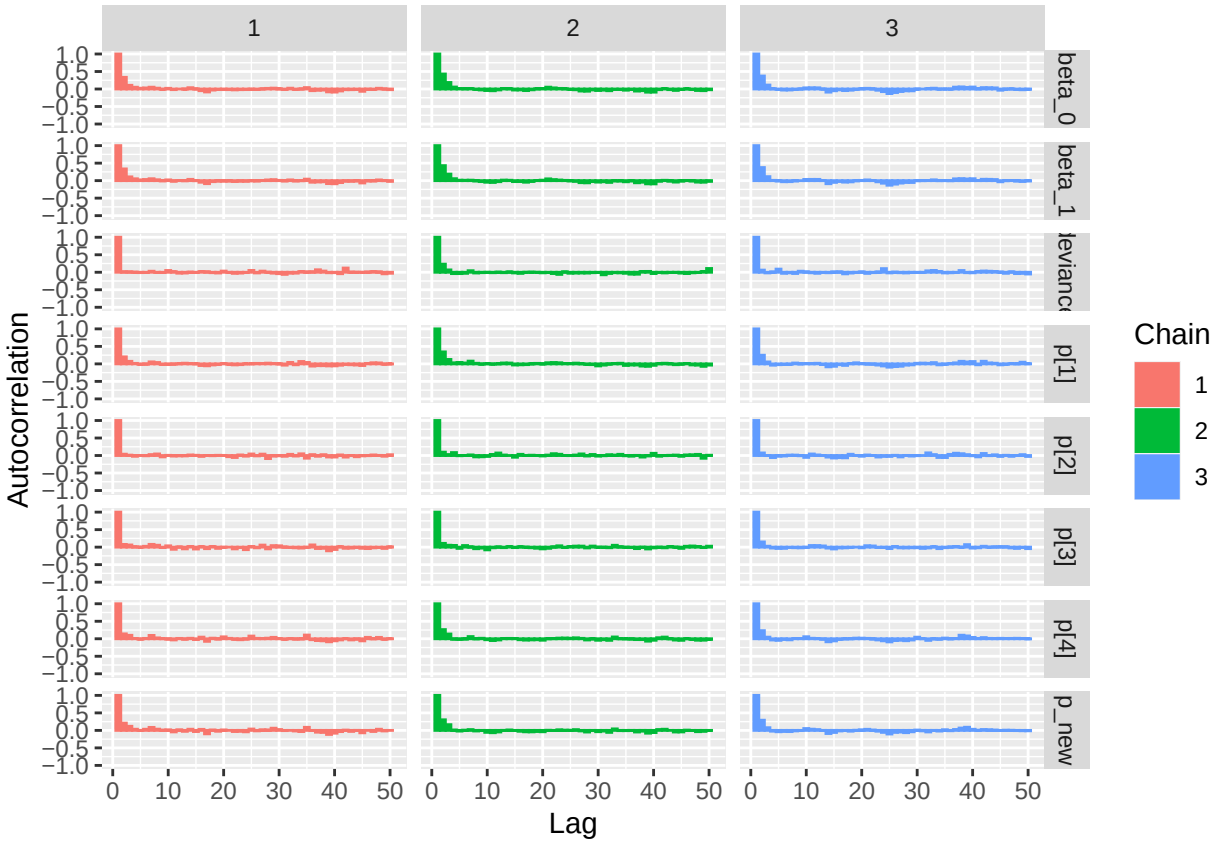
Traceplots show the history of parameters' values across iterations of the chain. The plot above shows that the average for all values appear to have converged as they follow a horizontal pattern - this suggests that they do not depend on their initial values. Those that seem more volatile are following a normal distribution.



Autocorrelation

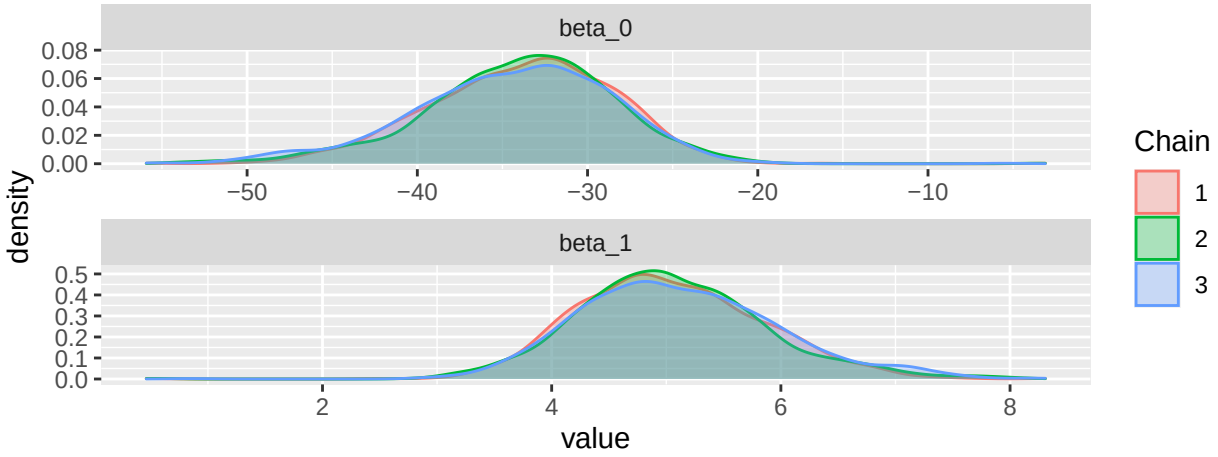
Autocorrelation is a value between -1 and 1, which measures how linearly dependent the current value of the chain is to past values, *known as lags*.

At the zeroth lag, a value in the chain has perfect autocorrelation with itself. As we progress along the chain, the values become less correlated. This is further indication that the values have converged.



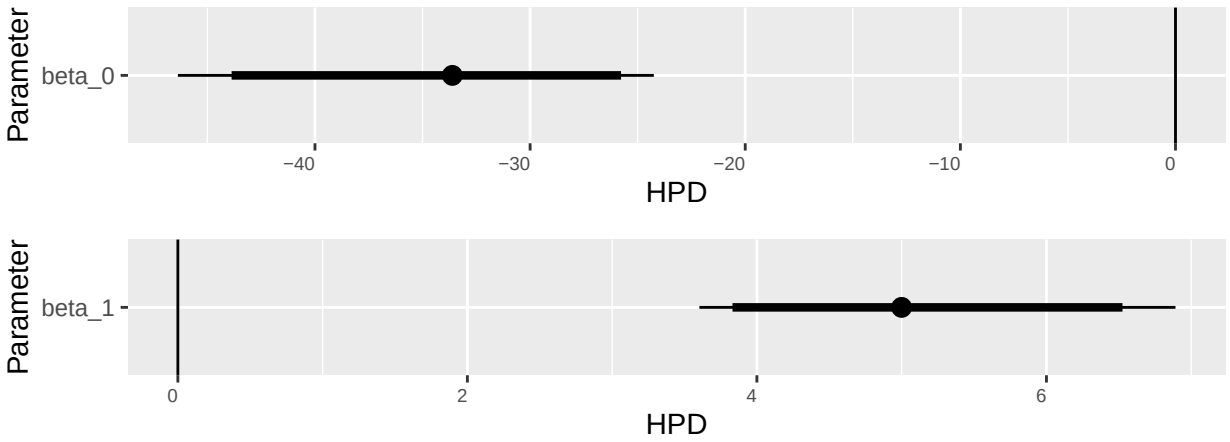
Density Plots

We can see that for both β_0 and β_1, all three chain values have converged with one another as the distributions mostly overlap one another.



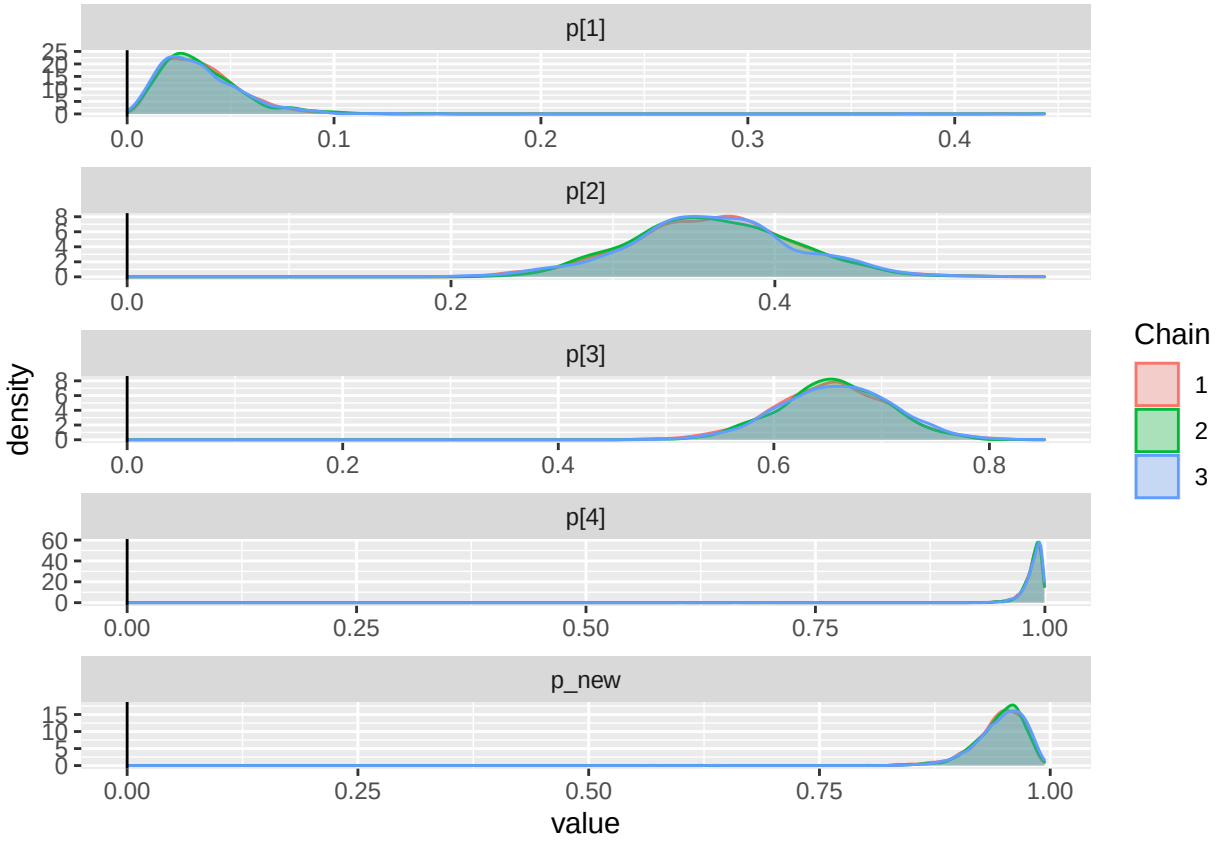
Caterpillar Plots

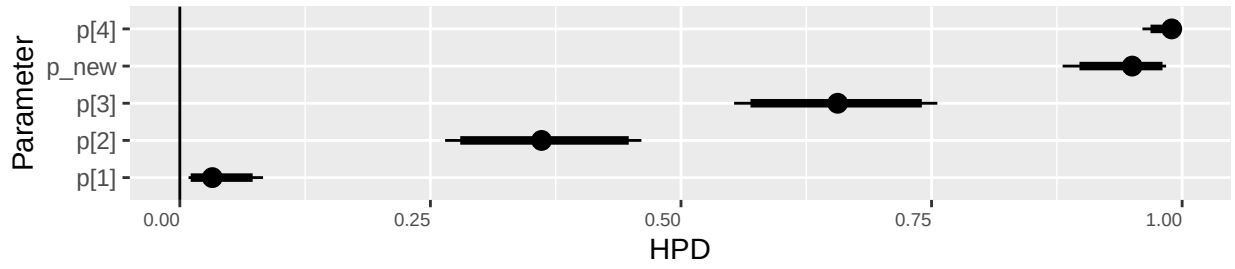
Caterpillar plots provide an insight into the distribution of parameter values. The dots represent median values, and the thick and thin lines represent the 90 and 95 of the highest posterior density regions respectively. We can see that in both plots, there is no posterior support at zero.



Plotting p Values

As we saw in the plots for β_0 and β_1, all values shown here have converged and show no indication of any posterior support at zero. Furthermore, we can see that there is a positive correlation between the dosage and the likelihood of a patient showing signs of improvement.





Part (i)

Compare the 95% confidence intervals of β_0 and β_1 obtained in part (c) using the frequentist logarithmic model, with the corresponding 95% credible intervals obtained in part (h) using the bayesian logarithmic model.

```
# Creating the frequentist model
m_log <-
  glm(cbind(n_improved, n_patients - n_improved) ~ covid_data$log_dose,
      family = binomial,data = covid_data)
```

```
# Print the 95% confidence intervals for beta_0 and beta_1, for the frequentist model
confint(m_log)
```

```
##                          2.5 %    97.5 %
## (Intercept)         -44.701647 -23.66464
## covid_data$log_dose   3.515515   6.64896
```

```
# Print the 95% credible intervals, for the bayesian logarithmic model
bayesian_binary_logistic$BUGSoutput$summary[1:2, c('mean', '50%', '2.5%', '97.5%')]
```

```
##              mean        50%       2.5%      97.5%
## beta_0 -34.068083 -33.613787 -46.373079 -24.249431
## beta_1   5.065688   4.999509   3.602621   6.891496
```

Looking at the intervals for both models, we can see that they share similar results. This is because the prior distributions for the unknown values (β_0 and β_1) were flat. Using flat or vague priors results in the Bayesian approach being based on the likelihood function, which is incredibly similar to the frequentist approach.

Part (j)

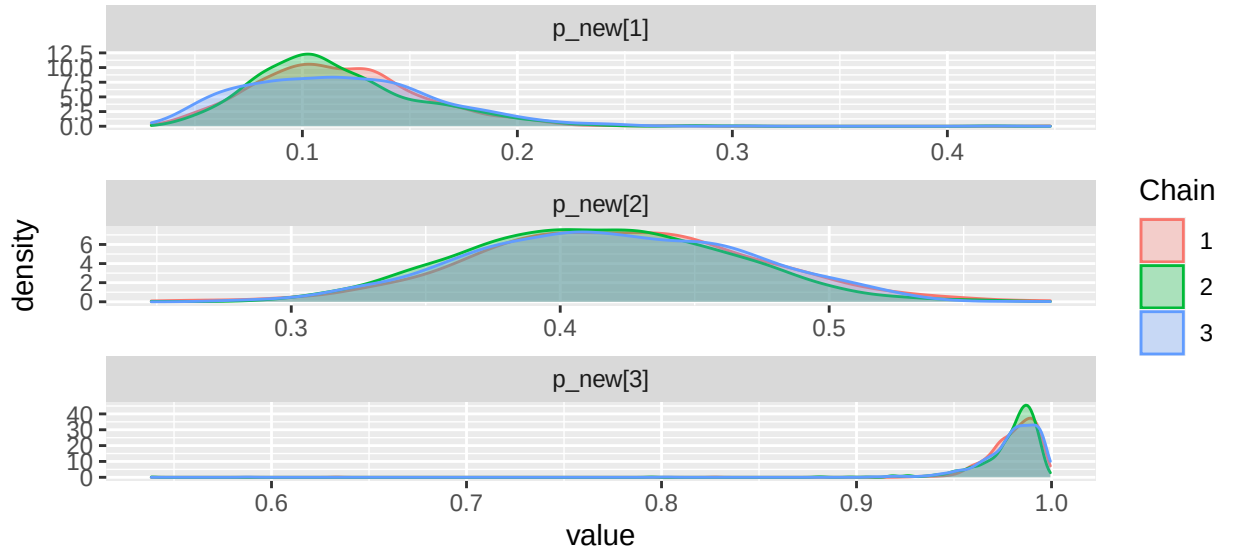
Using the Bayesian logarithmic model implemented in part (h), estimate approximate 95% credible intervals for the probability that COVID-19 patients who receive doses of 550, 780, and 1900 mg/mL of the medicine show signs of improvement.

Include a graphical representation, together with the numerical values of the 95% credible intervals, and comment on your results.

```
# Creating the model for prediction
b_binary_logistic_model_pred <- function() {
  for (i in 1:n_obs) {
    y[i] ~ dbin(p[i], n[i]) # binomial distribution

    logit(p[i]) <- eta[i] # link function

    eta[i] <- beta_0 + beta_1 * log(x[i])
  }
  # Specify prior distributions on the unknown parameters
  beta_0 ~ dnorm(0.0, 1.0E-4)
  beta_1 ~ dnorm(0.0, 1.0E-4)

  # Prediction
  for (j in 1:n_new_dose) {
    eta_new[j] <- beta_0 + beta_1 * log(x_new[j])

    # Probabilities
    p_new[j] <- exp(eta_new[j]) / (1 + exp(eta_new[j]))
  }
}

# Performing predictions
bayesian_binary_logistic_prediction <-
  jags(data = task_j_data,parameters.to.save = c('p_new'),n.iter = 10000,
      n.chains = 3,model.file = b_binary_logistic_model_pred)
```

```
# Print numerical results of the prediction
print(bayesian_binary_logistic_prediction, intervals = c(0.025, 0.5, 0.975))
```

```
## Inference for Bugs model at "C:/Users/tania/AppData/Local/Temp/RtmpEz6Wow/model40b45e472c0a.txt", fi
##  3 chains, each with 10000 iterations (first 5000 discarded), n.thin = 5
##  n.sims = 3000 iterations saved
##          mu.vect sd.vect   2.5%    50%  97.5%  Rhat n.eff
## p_new[1]   0.117   0.041  0.051  0.112  0.206 1.006  3000
## p_new[2]   0.417   0.050  0.323  0.416  0.515 1.004   650
## p_new[3]   0.979   0.019  0.944  0.983  0.997 1.011  2100
## deviance  24.018   3.576 21.871 23.290 29.366 1.005   470
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 6.4 and DIC = 30.4
## DIC is an estimate of expected predictive error (lower deviance is better).
```



The numerical and graphical results of the prediction model *(above)* follow the expected trend of: an increase in dose results in a higher likelihood of a patient showing signs of improvement. Note that the chain values in the density plot for p_new[1] have a slight variance. This may be an indication that the values have not properly converged, however, they still largely overlap one another - the variation is minimal.

Task (k)

Consider the following quadratic bayesian model for the COVID-19 data and perform inference about its parameters, writing appropriate jags/BUGS code.

Use numerical and graphical representations of the posterior probabilities to discuss whether β_2 is an appropriate term in the model.

```
# Creating the quadratic model
b_binary_logistic_quadratic_model <- function() {
  for (i in 1:n_obs) {
    y[i] ~ dbin(p[i], n[i]) # binomial distribution
    logit(p[i]) <- eta[i] # link function
    eta[i] <- beta_0 + beta_1 * log(x[i]) + beta_2 * log(x[i])^2
  }
  # Specify prior distributions on unknown parameters
  beta_0 ~ dnorm(0.0, 1.0E-4)
  beta_1 ~ dnorm(0.0, 1.0E-4)
  beta_2 ~ dnorm(0.0, 1.0E-4)
  # Evaluate the linear predictor at the new value of dose (x_new)
  eta_new <- beta_0 + beta_1 * log(x_new) + beta_2 * log(x_new)^2
  # Convert to the probability scale
  p_new <- exp(eta_new) / (1 + exp(eta_new))
}

# Performing Bayesian inference
bayesian_binary_logistic_quadratic <-
    jags(data = task_k_data,
```

```
        parameters.to.save = c('beta_0', 'beta_1', 'beta_2', 'p', 'p_new'),
        n.iter = 100000,
        n.chains = 3,
        model.file = b_binary_logistic_quadratic_model)
```

```
## Inference for Bugs model at "C:/Users/tania/AppData/Local/Temp/RtmpEz6Wow/model40b43b26246f.txt", fi
##  3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 50
##  n.sims = 3000 iterations saved
##          mu.vect sd.vect     2.5%     50%  97.5%  Rhat n.eff
## beta_0   -74.775  50.642 -174.022 -74.077 25.913 1.001  3000
## beta_1    16.971  14.819  -12.733  16.891 45.650 1.001  3000
## beta_2    -0.869   1.086   -2.948  -0.871  1.365 1.001  3000
## p[1]       0.026   0.023    0.003   0.020  0.073 1.001  3000
## p[2]       0.368   0.052    0.268   0.367  0.473 1.002  1200
## p[3]       0.672   0.052    0.570   0.674  0.769 1.002  1700
## p[4]       0.977   0.022    0.925   0.983  0.998 1.001  3000
## p_new      0.940   0.030    0.875   0.943  0.983 1.001  3000
## deviance  23.502   3.595   20.622  22.836 29.660 1.001  3000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 6.5 and DIC = 30.0
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Looking at the credible intervals for β_2, we can see that zero IS found within. This is an indication that β_2 may not be an appropriate term for our model.

Comparing the results to those from the simple logarithmic model *(below)*, we can see that while the credible intervals for β_0 and β_1 include zero in the quadratic model, they do not in the simple model.

```
## Inference for Bugs model at "C:/Users/tania/AppData/Local/Temp/RtmpEz6Wow/model40b43e0173b6.txt", fi
##  3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 50
##  n.sims = 3000 iterations saved
##          mu.vect sd.vect    2.5%     50%   97.5%  Rhat n.eff
## beta_0   -34.068   5.613 -46.373 -33.614 -24.249 1.002  1100
## beta_1     5.066   0.835   3.603   5.000   6.891 1.002  1300
## p[1]       0.036   0.023   0.009   0.032   0.083 1.002  1600
## p[2]       0.362   0.050   0.265   0.361   0.461 1.001  3000
## p[3]       0.656   0.052   0.553   0.656   0.756 1.002  1600
## p[4]       0.987   0.014   0.960   0.990   0.998 1.009  3000
## p_new      0.946   0.028   0.881   0.950   0.984 1.002  1700
## deviance  23.957   3.353  21.860  23.182  29.906 1.002  1800
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 5.6 and DIC = 29.6
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Deviance Information Criterion

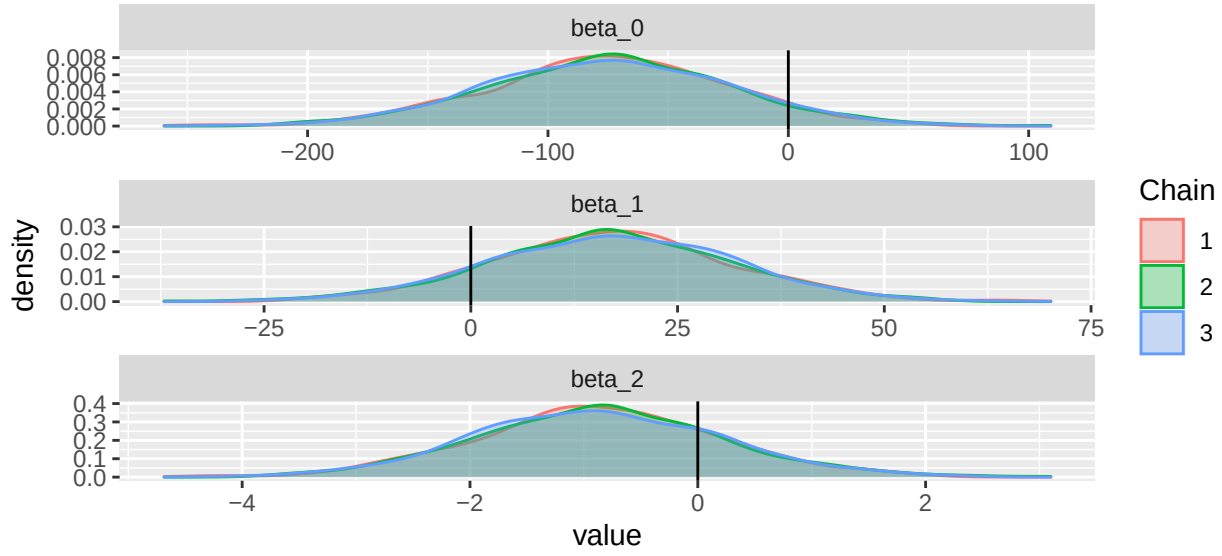
Based on the results below, we can see that the simple model produces a lower DIC value than the quadratic model, which is more favorable. Note that as the values are so similar to one another, it is possible for an execution of this code to result in the quadratic model showing a lower DIC value. Multiple executions have shown that the simple logarithmic model generally produces the lower value.

```
## [1] "Logarithmic Model DIC: 29.5777704381305"
```

```
## [1] "Quadratic Model DIC: 29.9692062382035"
```

Density Plots

The density plots illustrate the inclusion of zero within the credible intervals for all three unknown parameters in the quadratic model. Here we can more clearly see the posterior support for all unknown parameters, with significant posterior support for β_2.



Task (l)

Which of the two Bayesian models considered in part (h), the logarithmic model, and (k), the quadratic model, do you prefer? Why?

Based on the results of part (h) and (k), the simple *(logarithmic)* model appears to be the best suited for the task of predicting the effectiveness of different dosages of medicine, given the provided data.

The simple model resulted in no signs of posterior support for all unknown values while the quadratic model, with the inclusion of β_2, resulted in posterior support for all unknown values *(not just β_2)*.

Furthermore, the simple model produces a lower DIC value than the quadratic model, however the difference between the two is small. Despite this, alongside all other information gathered about the two models, the simple model is still the more favourable option of the two.

References

(Martinez-Taboada, F. & Redondo, J.I., 2020) Variable importance plot (mean decrease accuracy and mean decrease Gini) Accessed: 26 April 2021