Appendix 1: Objective 1

Kritika Dusad & Irene Alisjahbana

August 14, 2019

This appendix contains the codes that are used for Objective 1 in order to produce the outputs seen in the final report.

Data Pre-Processing

Set the working directory and load the necessary libraries.

```
#Set working directory
path1 <- "~/Google Drive/Stanford Files/Q7 Summer 2019/"
path2 <- "STATS 202 - Data Mining and Analysis/Final Project"
setwd(paste0(path1, path2))

# Load libraries
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 3.5.2
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(car)
```

```
## Warning: package 'car' was built under R version 3.5.2
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
library(ggpubr)
```

```
## Warning: package 'ggpubr' was built under R version 3.5.2
```

```
## Loading required package: ggplot2
```

```
## Loading required package: magrittr
```

Next we will load the patients data. We will use all of the studies in order to see if there is a difference between patients that received treatment and patients that did not.

```
# Load studies
Study_A = read.csv("Study_A.csv")
Study_B = read.csv("Study_B.csv")
Study_C = read.csv("Study_C.csv")
Study_D = read.csv("Study_D.csv")
Study_E = read.csv("Study_E.csv")
```

Then, we perform pre-processing on the data.

First, we want to remove the column LeadStatus from Studies A-D because Study E does not contain this column. We don't need this column for the purpose of our objective.

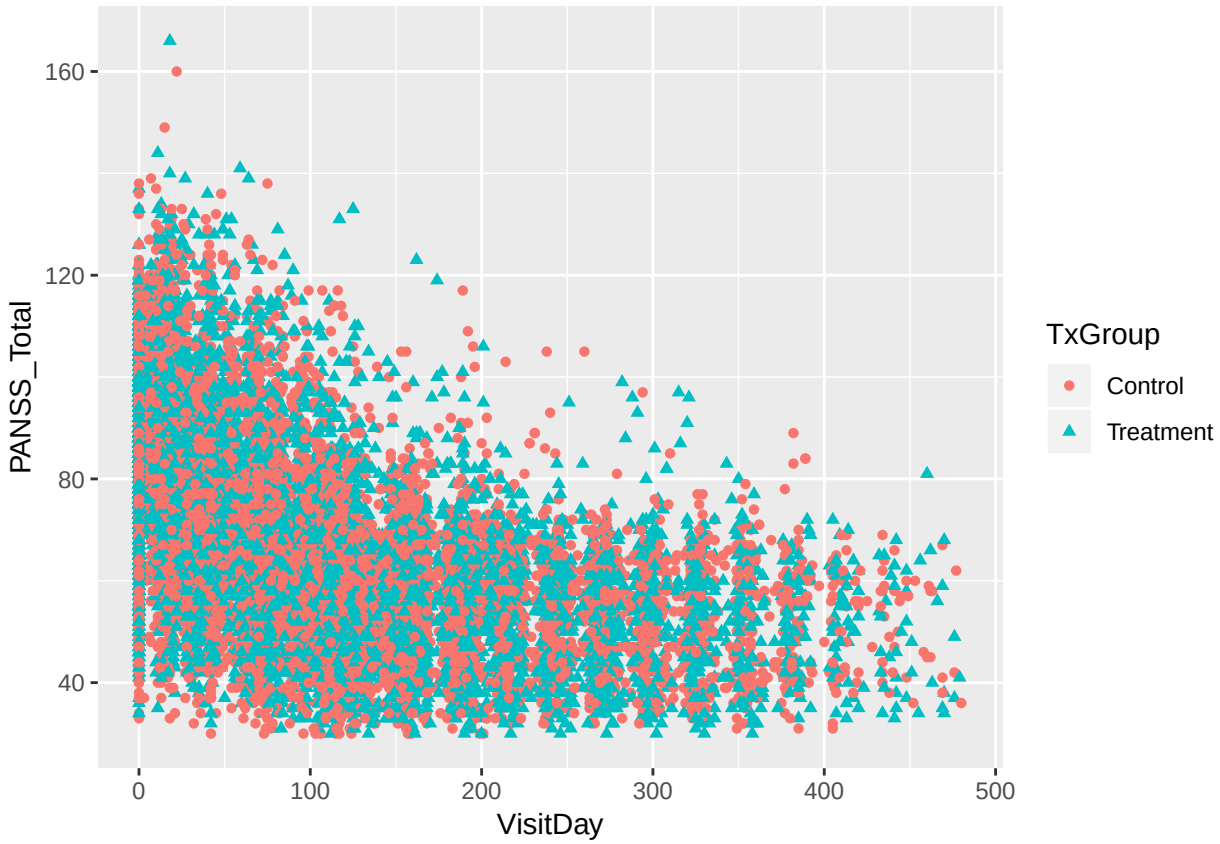
```
Study_A <- subset(Study_A, select = -c(LeadStatus))
Study_B <- subset(Study_B, select = -c(LeadStatus))
Study_C <- subset(Study_C, select = -c(LeadStatus))
Study_D <- subset(Study_D, select = -c(LeadStatus))
```

Next, we combine all the studies into one dataset.

```
# Combine all studies into one dataset
All_Studies <- rbind(Study_A, Study_B, Study_C, Study_D, Study_E)
```

Before doing any analysis, we want to visualize the data.

```
ggplot(All_Studies, aes(x=VisitDay, y=PANSS_Total, shape=TxGroup, color=TxGroup)) +
  geom_point()
```



While looking through the data, we realized that there are observations with Country as ERROR. We want to remove these before continuing.

```
# Removed observations with Country as ERROR- 18 observations removed.
All_Studies <- subset(All_Studies, Country != "ERROR")
```

We then take the treatment patients with Visit Day observations that are greater than 120. From here we extract the patient's PANSS_score for the very first and last observation. We make sure that any duplicates in the observations are removed by taking the mean values of the symptom scores.

```
# Take subset of treatment patients
treatment_obs <- subset(All_Studies, TxGroup=="Treatment",
                        select=-c(Study, Country, AssessmentiD, RaterID, SiteID))

# Take subset of patients with observations in Visit Day > 120
treatment_end <- subset(treatment_obs, VisitDay >= 120)

# Get Patient IDs with those observations
patients_treatment <- unique(treatment_end["PatientID"])
patients_treatment <- patients_treatment[,1]

# Get all observations for those patients
patient_obs_treatment <- filter(treatment_obs, PatientID %in% patients_treatment)

# Extract values of VisitDay = 0
treatment_start <- subset(patient_obs_treatment, VisitDay == 0)

# Take mean values of duplicates
```

```r
cols <- 4:34
treatment_start <- setDT(treatment_start)[, lapply(.SD,mean),
                                          by=c(names(treatment_start)[1:3]),
                                          .SDcols = cols]

# Keep the last observation for each patient
treatment_end <- treatment_end[!rev(duplicated(rev(treatment_end$PatientID))),]
```

The same is done for the control patients.

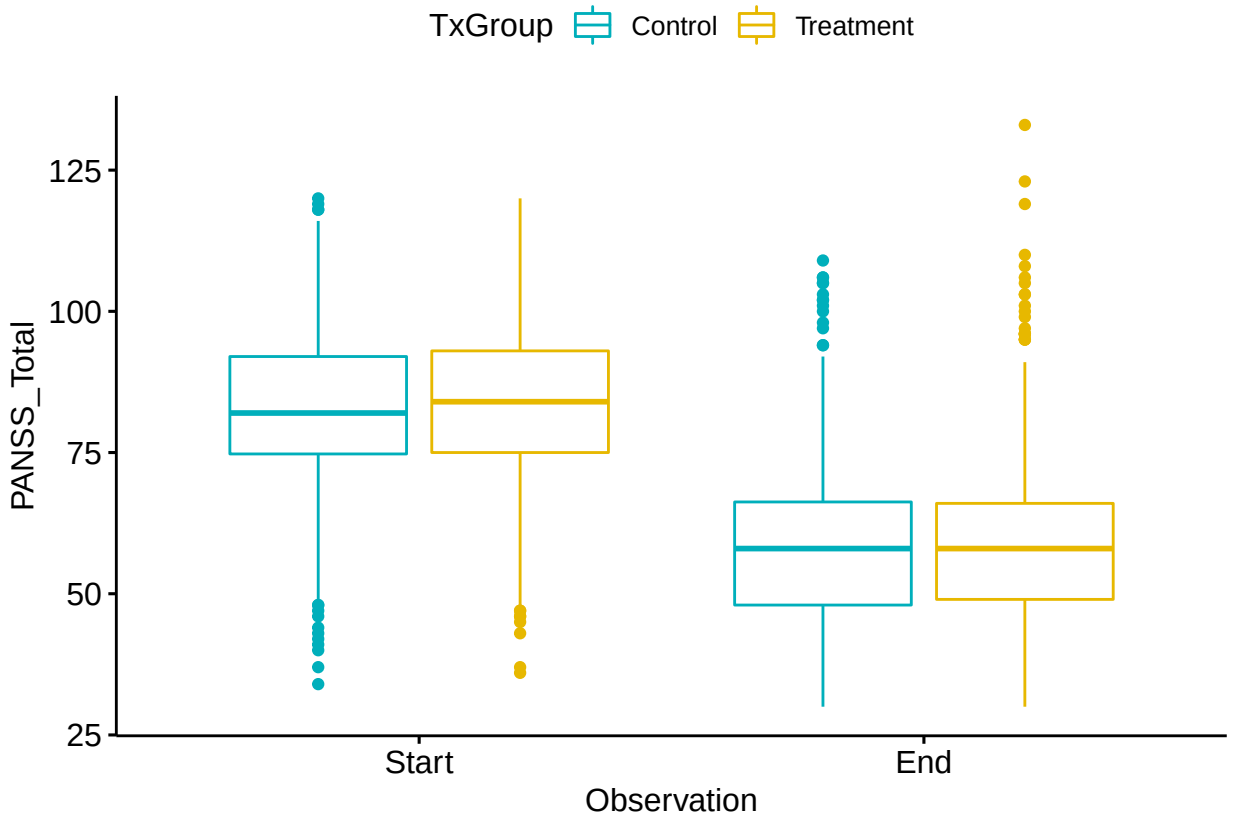
```r
# Take subset of control patients
control_obs <- subset(All_Studies, TxGroup=="Control",
                      select=-c(Study, Country, AssessmentiD, RaterID, SiteID))

# Take subset of patients with observations in Visit Day > 120
control_end <- subset(control_obs, VisitDay >= 120)

# Get Patient IDs with those observations
patients_control <- unique(control_end["PatientID"])
patients_control <- patients_control[,1]

# Get all observations for those patients
patient_obs_control <- filter(control_obs, PatientID %in% patients_control)

# Extract values of VisitDay = 0
control_start <- subset(patient_obs_control, VisitDay == 0)

# Take mean values of duplicates
control_start <- setDT(control_start)[, lapply(.SD,mean),
                                      by=c(names(control_start)[1:3]),
                                      .SDcols = cols]

# Keep the last observation for each patient
control_end <- control_end[!rev(duplicated(rev(control_end$PatientID))),]
```

We then combine both the control and treatment patients into one dataset. We name the first observation of each patient "Start" and the last observation of each patient "End". We also remove the columns PatientID and VisitDay for this objective.

```r
# Control patients
control_start$Observation <- "Start"
treatment_start$Observation <- "Start"

# Treatment patients
control_end$Observation <- "End"
treatment_end$Observation <- "End"

# Combine dataset
patients_startend = rbind(control_start,treatment_start, control_end, treatment_end)

# Drop columns
patients_startend <- subset(patients_startend, select = -c(PatientID, VisitDay))
```

This dataset can then be visualized through a box plot.

```r
# Visualize data
ggboxplot(patients_startend, x = "Observation", y = "PANSS_Total",
          color = "TxGroup", palette = c("#00AFBB", "#E7B800"))
```



Two-Way Anova Test using PANSS_Total

Before performing the 2-way ANOVA test, we have to check to see if the dataset is balanced or unbalanced.

```r
# Check to see if the data is balanced or unbalanced
table(patients_startend$TxGroup, patients_startend$Observation)
```

```
##
##             End Start
##   Control   740   740
##   Treatment 733   733
```

It can be seen from the table above that the data is unbalanced, therefore we have to perform the ANOVA test for unbalanced design. In this case, we use the Type III sum of squares.

```r
# ANOVA in unbalanced design: Type III sums of squares
anova_test <- aov(PANSS_Total ~ TxGroup * Observation, data = patients_startend)
Anova(anova_test, type = "III")
```

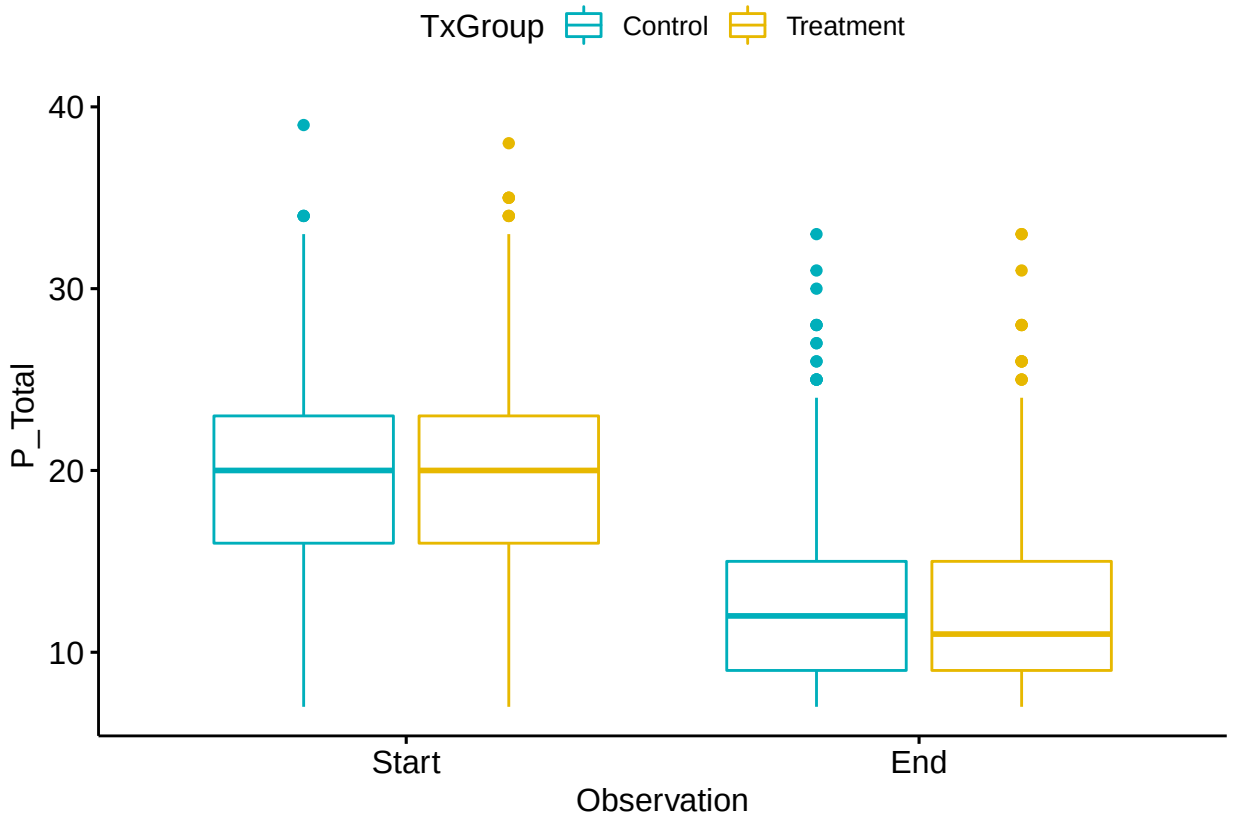
```
## Anova Table (Type III tests)
##
## Response: PANSS_Total
##                       Sum Sq   Df    F value Pr(>F)
## (Intercept)         2521829    1 12485.7989 <2e-16 ***
## TxGroup                  20    1     0.0988 0.7533
## Observation          231358    1  1145.4756 <2e-16 ***
```

```
## TxGroup:Observation       6    1     0.0313 0.8596
## Residuals              594213 2942
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

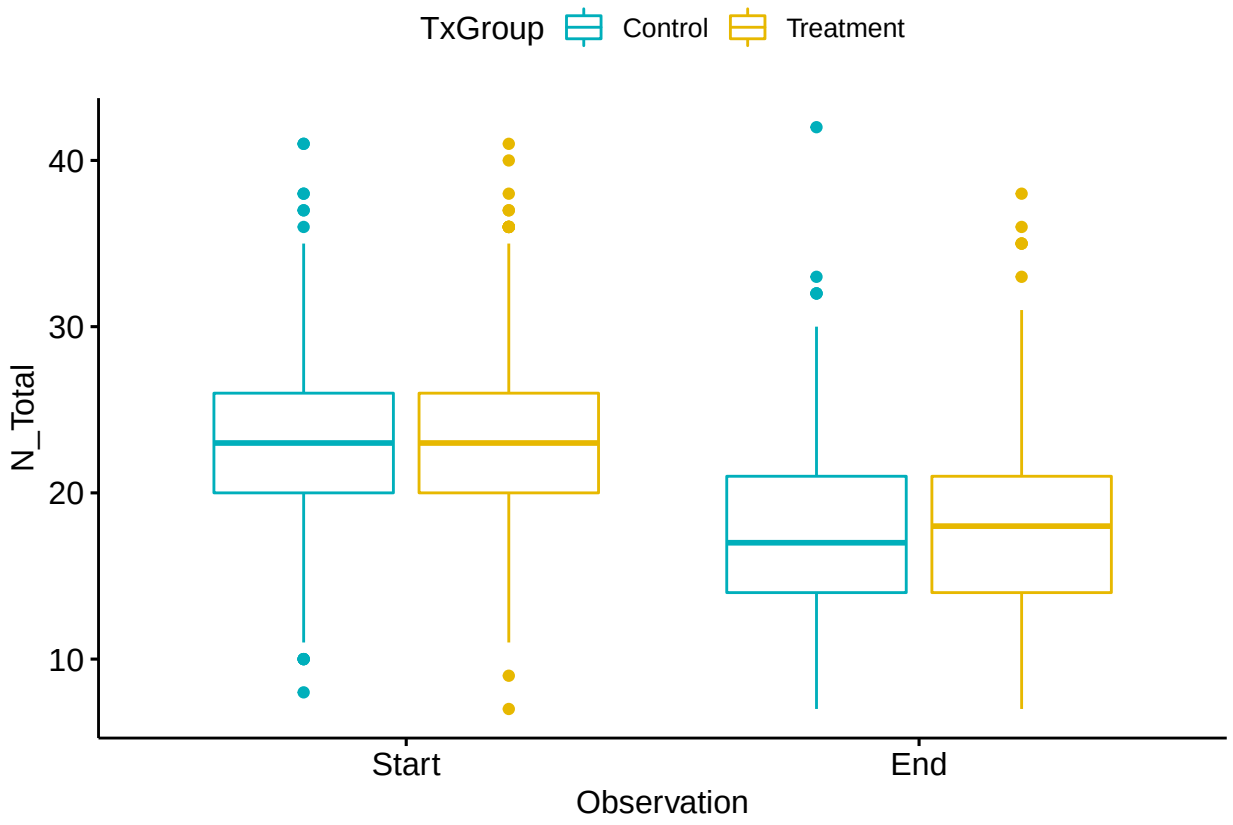
Two-Way Anova Test using Total P, N and G values

Our next experiment uses the seperate total P, N, and G values to see if the treatment has an effect on a specific type of symptom. This can be done by first summing the P, N and G scores.

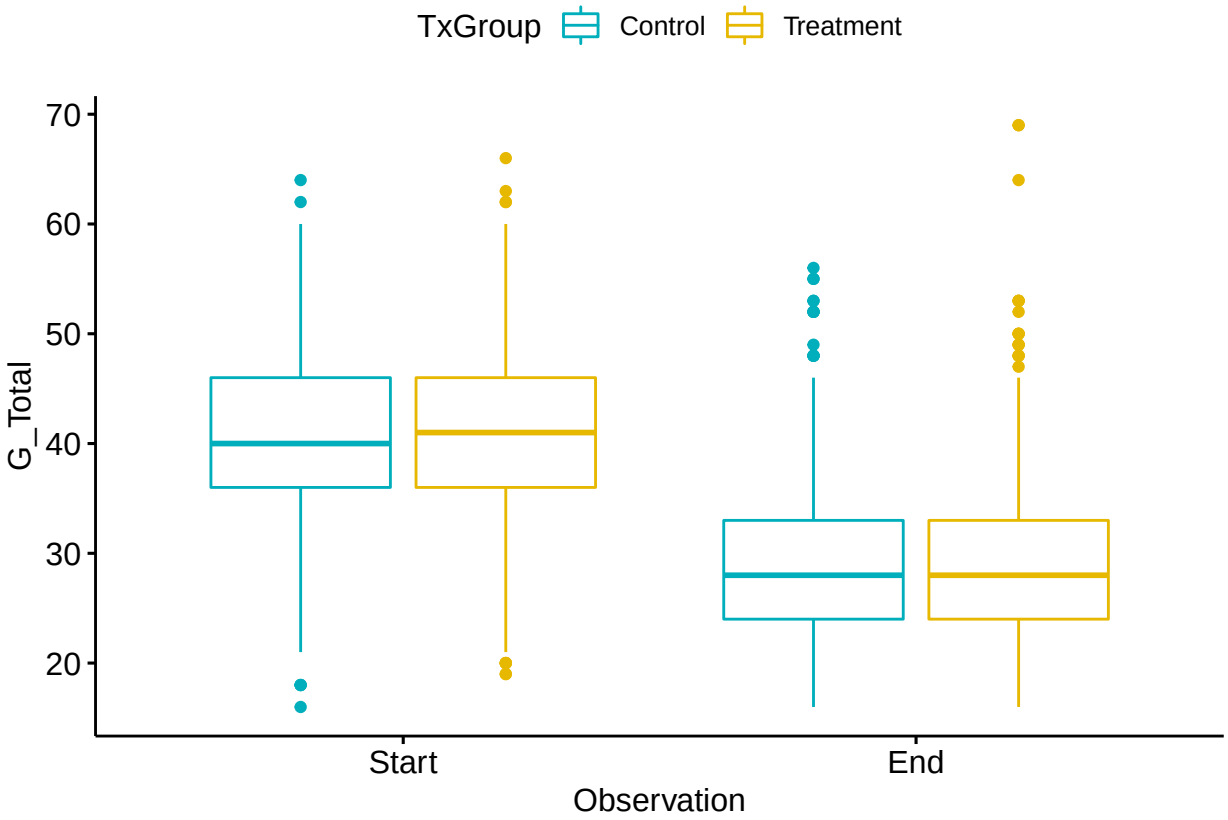
```
# Combine P, N and G values
patients_startend$P_Total <- rowSums(patients_startend[,c(2:8)])
patients_startend$N_Total <- rowSums(patients_startend[,c(9:15)])
patients_startend$G_Total <- rowSums(patients_startend[,c(16:31)])

# Visualize data
# P_Total
ggboxplot(patients_startend, x = "Observation", y = "P_Total",
          color = "TxGroup", palette = c("#00AFBB", "#E7B800"))
```




```
# N_Total
ggboxplot(patients_startend, x = "Observation", y = "N_Total",
          color = "TxGroup", palette = c("#00AFBB", "#E7B800"))
```




```
# G_Total
ggboxplot(patients_startend, x = "Observation", y = "G_Total",
          color = "TxGroup", palette = c("#00AFBB", "#E7B800"))
```



The Two-Way ANOVA test is then performed for each category.

```
# P_Total: ANOVA in unbalanced design: Type III sums of squares
anova_test <- aov(P_Total ~ TxGroup * Observation, data = patients_startend)
Anova(anova_test, type = "III")
```

```
## Anova Table (Type III tests)
##
## Response: P_Total
##                     Sum Sq   Df   F value Pr(>F)
## (Intercept)         112965    1 4693.6551 <2e-16 ***
## TxGroup                  1    1    0.0515 0.8205
## Observation          20239    1  840.9199 <2e-16 ***
## TxGroup:Observation      0    1    0.0025 0.9601
## Residuals            70807 2942
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# N_Total: ANOVA in unbalanced design: Type III sums of squares
anova_test <- aov(N_Total ~ TxGroup * Observation, data = patients_startend)
Anova(anova_test, type = "III")
```

```
## Anova Table (Type III tests)
##
## Response: N_Total
##                     Sum Sq   Df   F value Pr(>F)
## (Intercept)         221232    1 8748.5594 <2e-16 ***
## TxGroup                 21    1    0.8115 0.3678
## Observation          11132    1  440.2142 <2e-16 ***
```

```
## TxGroup:Observation      4    1    0.1540 0.6948
## Residuals            74397 2942
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# G_Total: ANOVA in unbalanced design: Type III sums of squares
anova_test <- aov(G_Total ~ TxGroup * Observation, data = patients_startend)
Anova(anova_test, type = "III")
```

```
## Anova Table (Type III tests)
##
## Response: G_Total
##                      Sum Sq   Df    F value Pr(>F)
## (Intercept)         610852    1 10425.4995 <2e-16 ***
## TxGroup                  1    1     0.0188 0.8909
## Observation          54394    1   928.3469 <2e-16 ***
## TxGroup:Observation      1    1     0.0105 0.9182
## Residuals           172378 2942
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Linear Regression

We try to employ a different model to see if we can capture the effect differences between the treatment and control patients. This time we use simple linear regression. As the dataset used is different from that of the ANOVA test, we have to preprocess the dataset differently.

```
# Subset if patient has visitday greater than 120.
studies_subset <- subset(All_Studies, VisitDay>=120)
IDs <- studies_subset["PatientID"][,1]
studies <- subset(filter(All_Studies, All_Studies$PatientID %in% IDs))

# Removing the duplicates
cols <- 9:39
processed_studies <- setDT(studies)[, lapply(.SD,mean),
                                    by=c("Country", "PatientID", "VisitDay",
                                         "TxGroup"),
                                    .SDcols = cols]

# Taking ceiling of Ps, Ns, and Gs.
processed_studies <- processed_studies %>% mutate_at(5:34, ceiling)

# Calculating PANSS_Total with new Ps, Ns, and Gs.
processed_studies$PANSS_Total <- rowSums(processed_studies[5:34])

#Combining Ns, Ps, and Gs scores. Making separate columns for them.
processed_studies$P_total <- rowSums(processed_studies[5:11])
processed_studies$N_total <- rowSums(processed_studies[12:18])
processed_studies$G_total <- rowSums(processed_studies[19:34])
```

For our linear regression model, we use PANSS_Total as the response and use TxGroup and Visit Day interaction terms as the predictors. We use the squared value of VisitDay to make sure our relationship is close to linear.

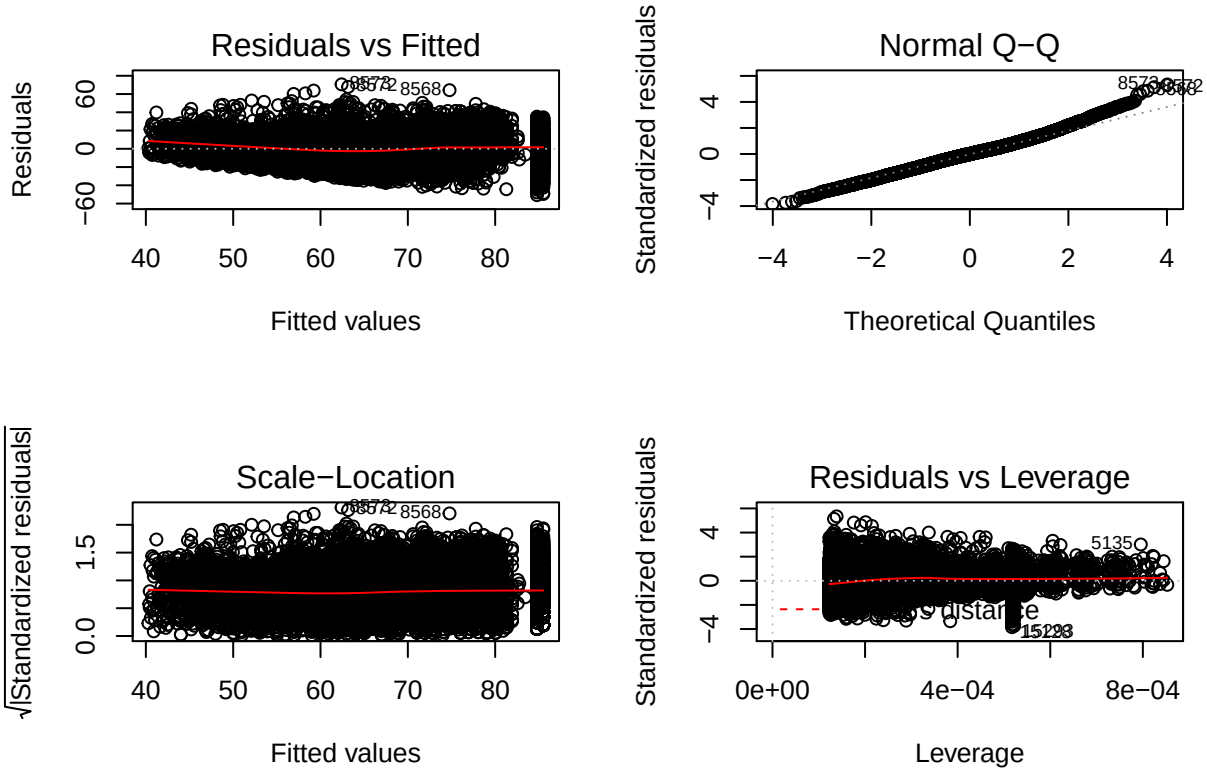
```
# For PANSS_Total
lm_PANSS <- lm(PANSS_Total ~ sqrt(VisitDay)*TxGroup, data=processed_studies)
```

```r
summary(lm_PANSS)
```

```
##
## Call:
## lm(formula = PANSS_Total ~ sqrt(VisitDay) * TxGroup, data = processed_studies)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -50.786  -8.423   0.016   7.775  70.587
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                      84.78554    0.30060 282.053   <2e-16 ***
## sqrt(VisitDay)                   -2.02615    0.02826 -71.688   <2e-16 ***
## TxGroupTreatment                  0.72127    0.42607   1.693   0.0905 .
## sqrt(VisitDay):TxGroupTreatment -0.03944    0.03999  -0.986   0.3241
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.22 on 16233 degrees of freedom
## Multiple R-squared:  0.3921, Adjusted R-squared:  0.392
## F-statistic:  3490 on 3 and 16233 DF,  p-value: < 2.2e-16
```

We can visualize our results using the `plot()` function.

```r
par(mfrow=c(2,2))
plot(lm_PANSS)
```



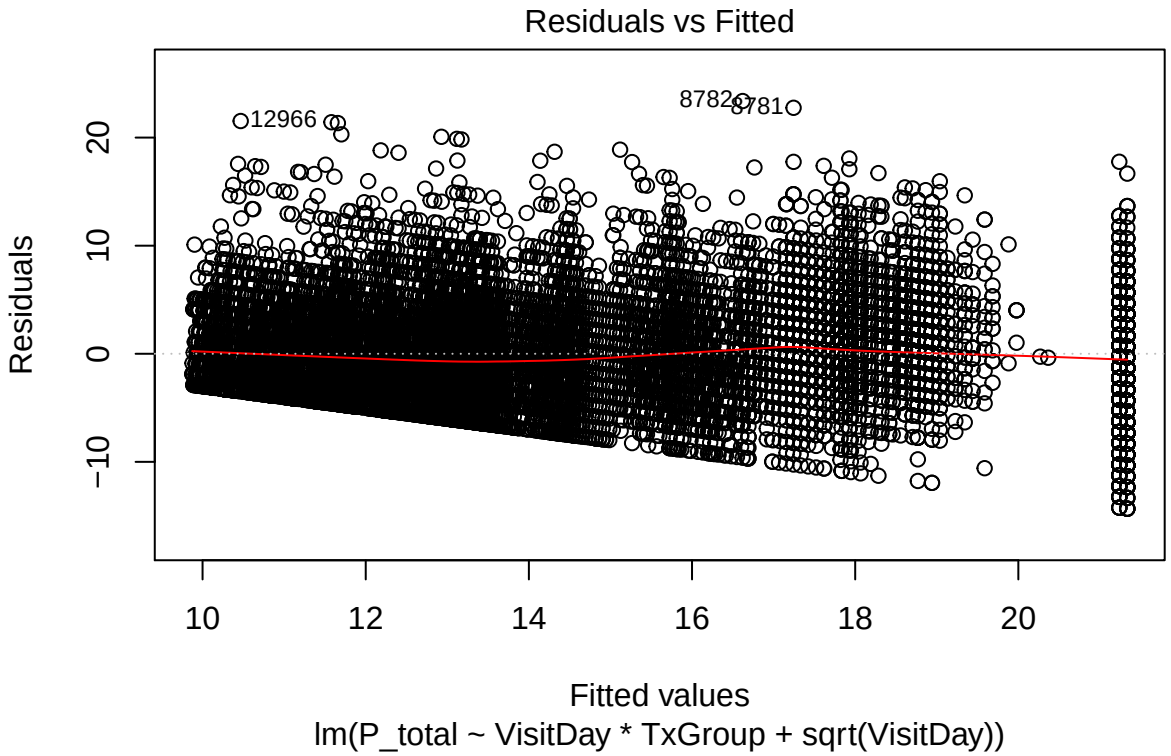
We performed the same linear regression with the total values for P, N and G totals.

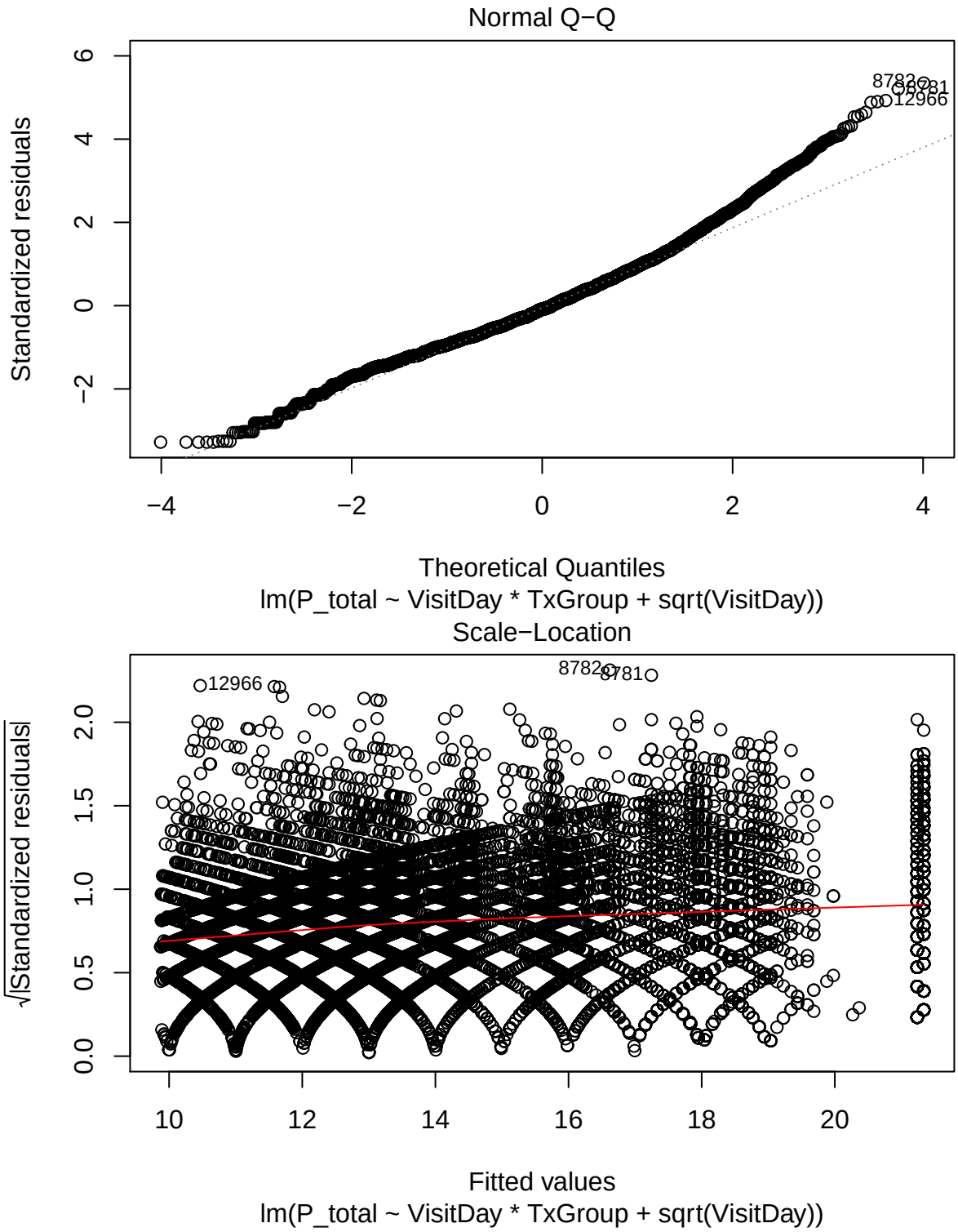
P Total:

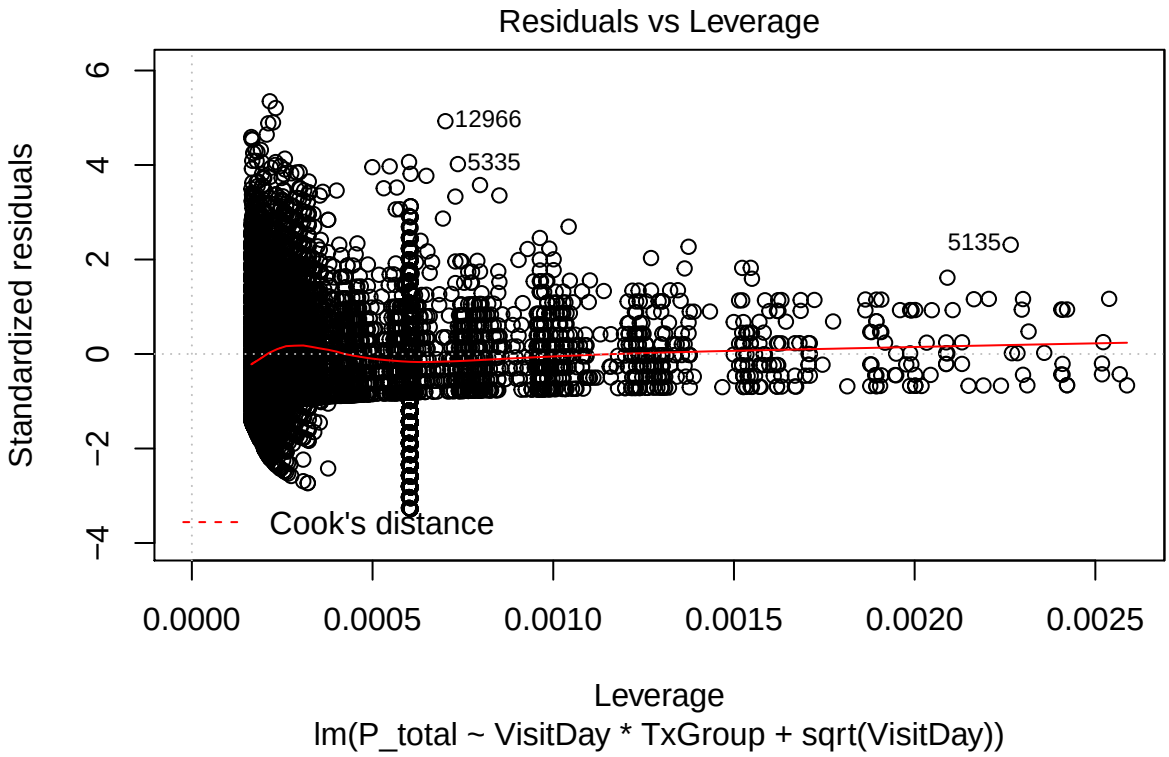
```r
lm_Ptotal <- lm(P_total ~ VisitDay*TxGroup + sqrt(VisitDay), data=processed_studies)
summary(lm_Ptotal)
```

```
##
## Call:
## lm(formula = P_total ~ VisitDay * TxGroup + sqrt(VisitDay), data = processed_studies)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.3355  -3.0688  -0.3686   2.6068  23.3796
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                21.2375575  0.1071775 198.153   <2e-16 ***
## VisitDay                    0.0215851  0.0011742  18.383   <2e-16 ***
## TxGroupTreatment            0.0979164  0.1044589   0.937    0.349
## sqrt(VisitDay)             -0.9903947  0.0213854 -46.312   <2e-16 ***
## VisitDay:TxGroupTreatment -0.0002636  0.0006938  -0.380    0.704
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.372 on 16232 degrees of freedom
## Multiple R-squared:  0.3472, Adjusted R-squared:  0.3471
## F-statistic:  2159 on 4 and 16232 DF,  p-value: < 2.2e-16
```

```r
plot(lm_Ptotal)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(P_total ~ VisitDay * TxGroup + sqrt(VisitDay))
Scale−Location
√|Standardized residuals|
12966
8781
Fitted values
lm(P_total ~ VisitDay * TxGroup + sqrt(VisitDay))

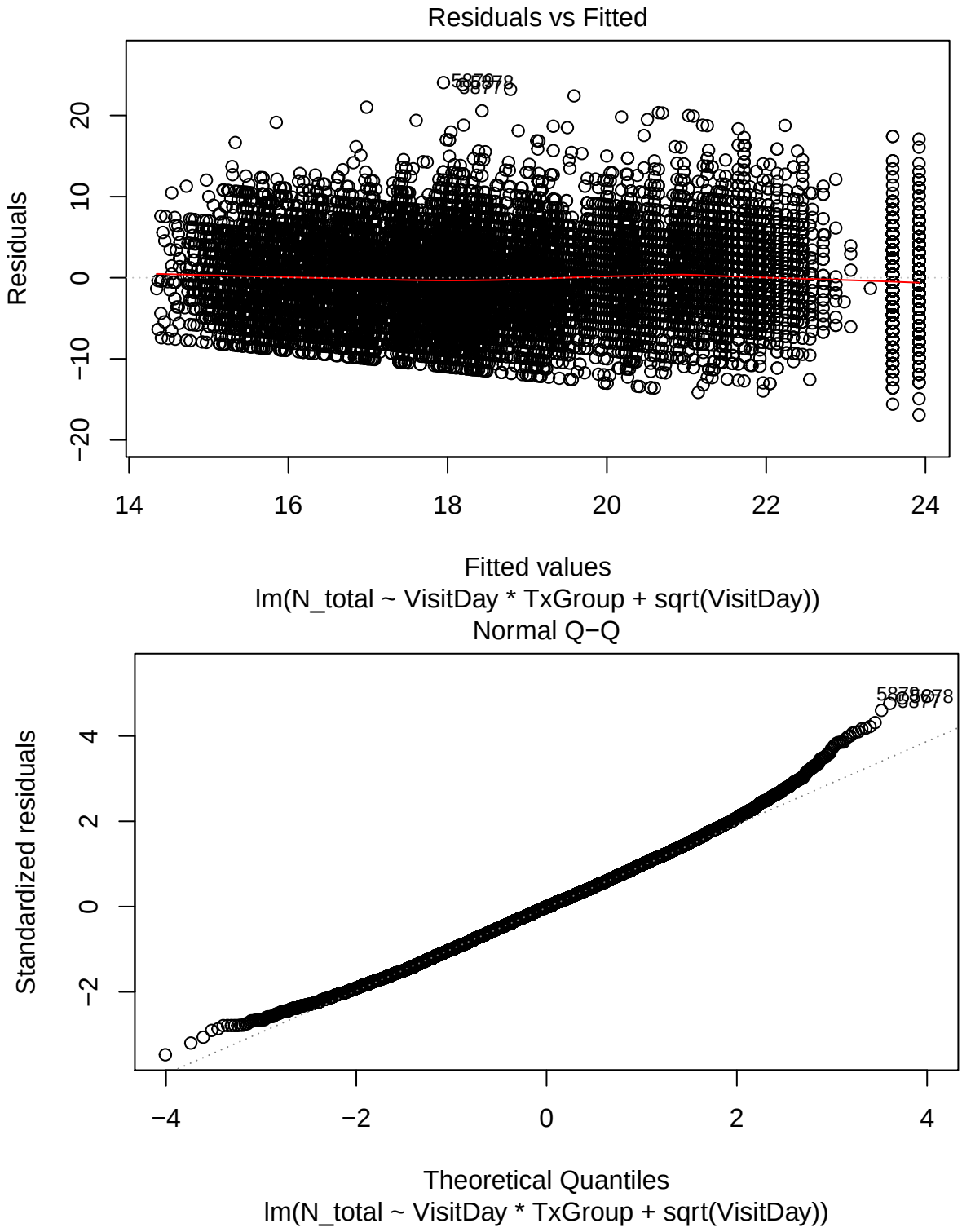


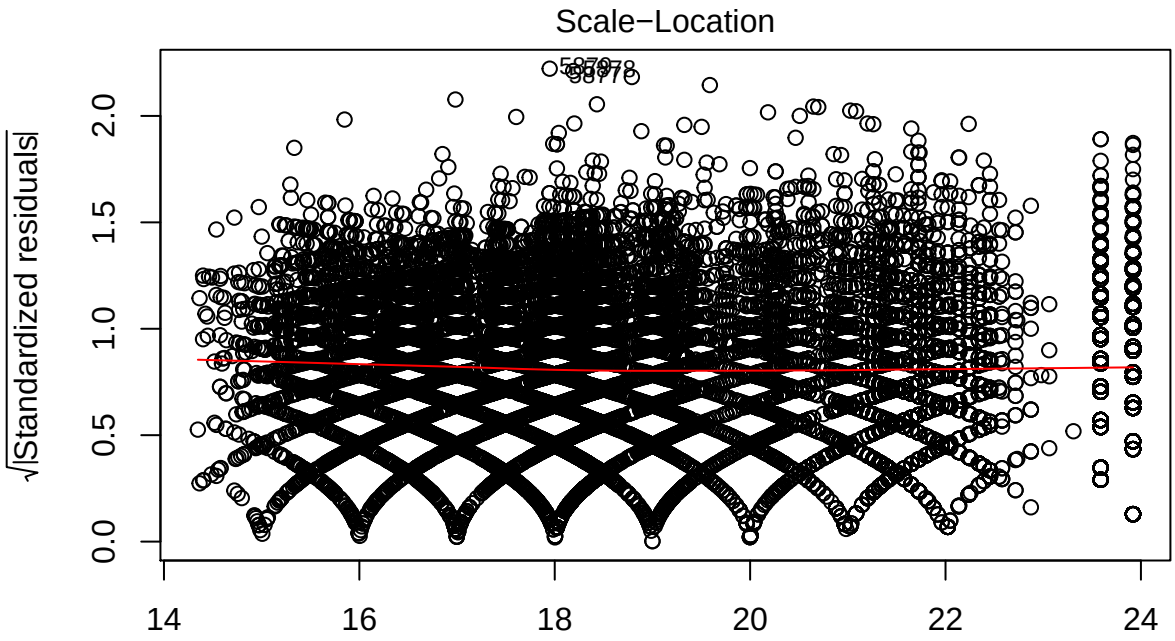
N Total:

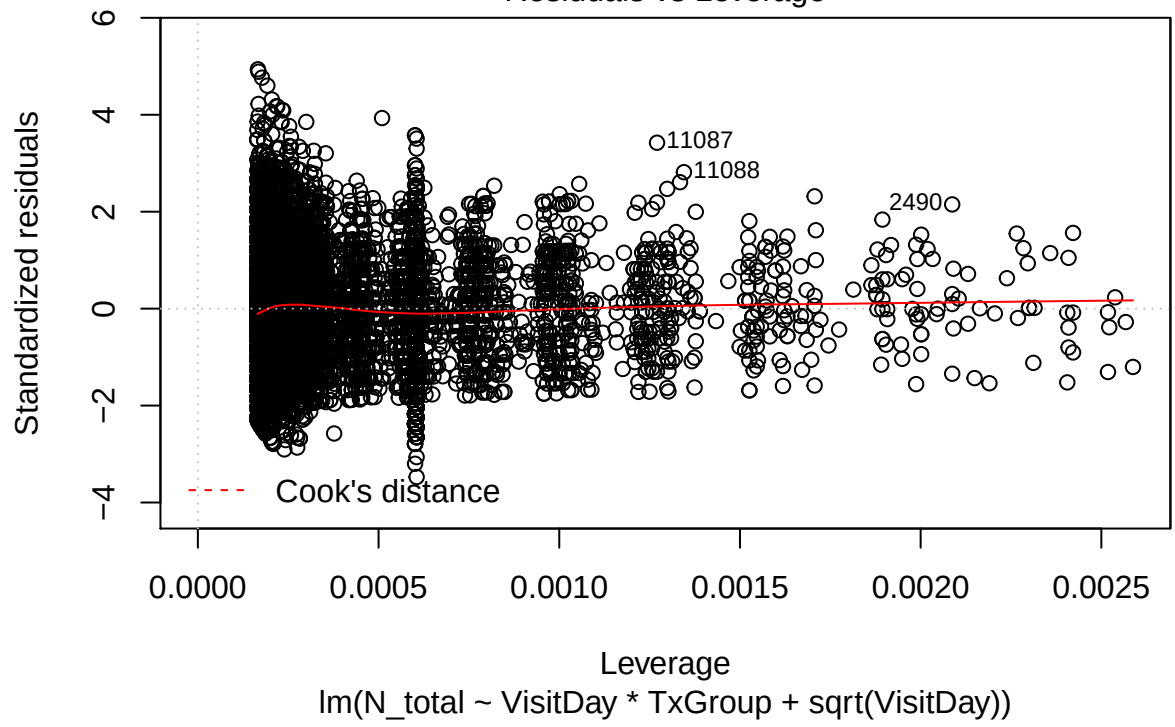
```
lm_Ntotal <- lm(N_total ~ VisitDay*TxGroup + sqrt(VisitDay), data=processed_studies)
summary(lm_Ntotal)
```

```
##
## Call:
## lm(formula = N_total ~ VisitDay * TxGroup + sqrt(VisitDay), data = processed_studies)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.9191  -3.3225  -0.0355   3.0809  24.0518
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                23.5881343  0.1194288 197.508  < 2e-16 ***
## VisitDay                    0.0100086  0.0013084   7.649 2.13e-14 ***
## TxGroupTreatment            0.3309351  0.1163994   2.843  0.00447 **
## sqrt(VisitDay)             -0.6179331  0.0238299 -25.931  < 2e-16 ***
## VisitDay:TxGroupTreatment -0.0017582  0.0007731  -2.274  0.02298 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.872 on 16232 degrees of freedom
## Multiple R-squared:  0.1908, Adjusted R-squared:  0.1906
## F-statistic: 957.1 on 4 and 16232 DF,  p-value: < 2.2e-16
```

```
plot(lm_Ntotal)
```


Residuals vs Fitted
Residuals
20
10
0
−10
−20
14
16
18
20
22
24
Fitted values
lm(N_total ~ VisitDay * TxGroup + sqrt(VisitDay))
Normal Q−Q
Standardized residuals
4
2
0
−2
−4
−2
0
2
4
Theoretical Quantiles
lm(N_total ~ VisitDay * TxGroup + sqrt(VisitDay))





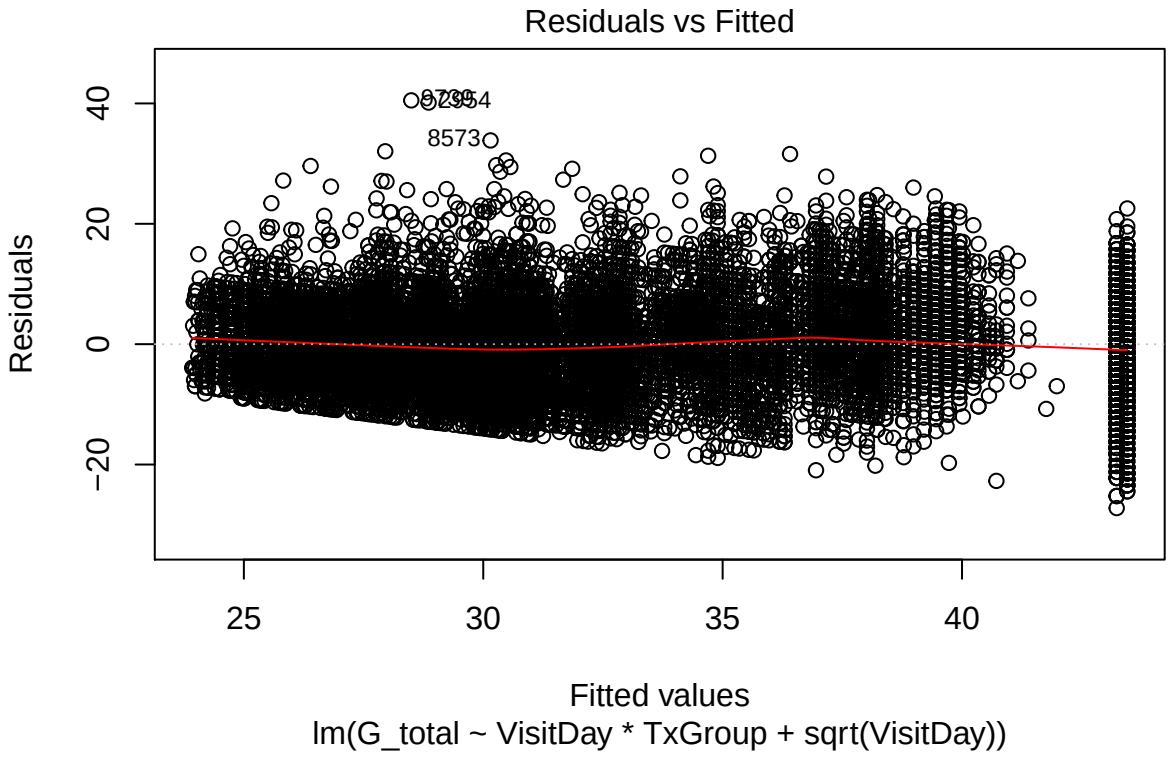
G Total:

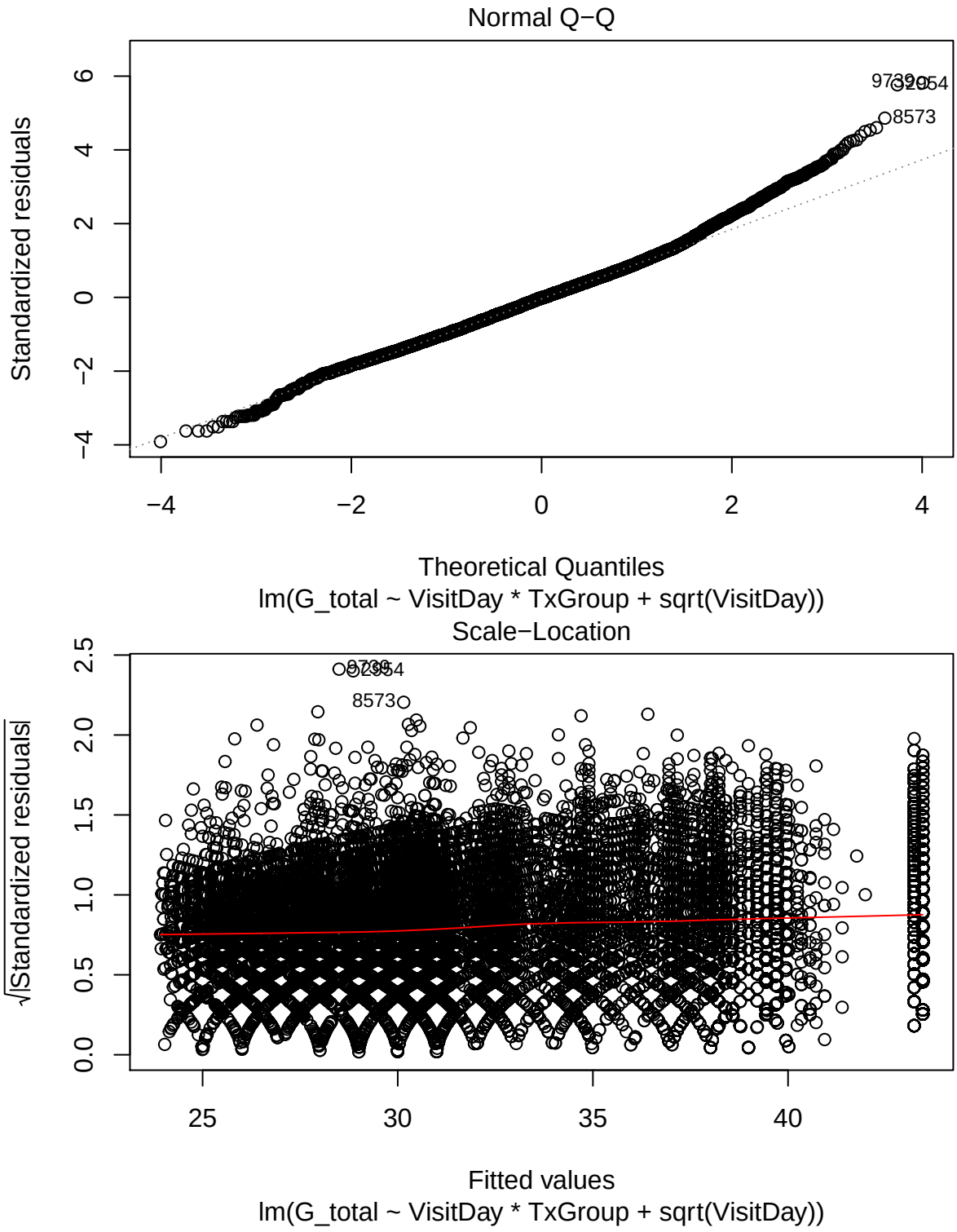
```
lm_Gtotal <- lm(G_total ~ VisitDay*TxGroup + sqrt(VisitDay), data=processed_studies)
summary(lm_Gtotal)
```

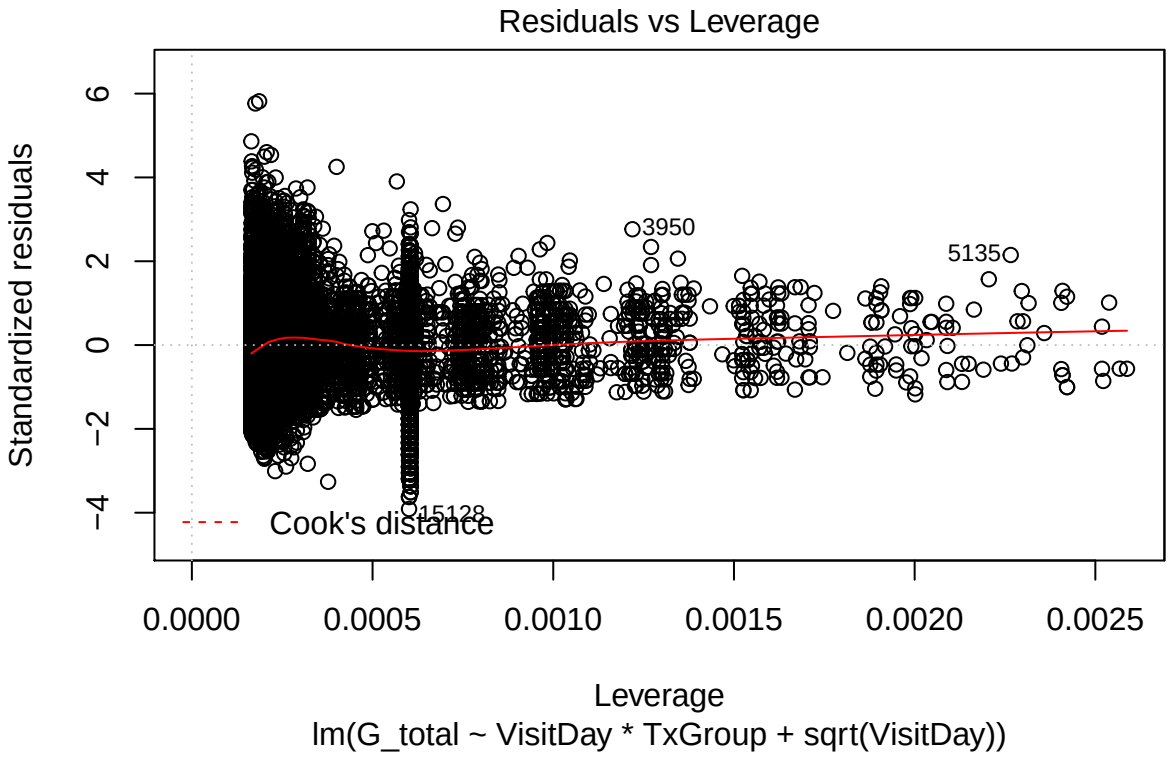
```
##
## Call:
```

```
## lm(formula = G_total ~ VisitDay * TxGroup + sqrt(VisitDay), data = processed_studies)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -27.230  -4.699  -0.105   4.159  40.503
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                43.2299993  0.1707221 253.219   <2e-16 ***
## VisitDay                    0.0282596  0.0018703  15.109   <2e-16 ***
## TxGroupTreatment            0.2215405  0.1663917   1.331    0.183
## sqrt(VisitDay)             -1.4997852  0.0340646 -44.028   <2e-16 ***
## VisitDay:TxGroupTreatment -0.0005077  0.0011052  -0.459    0.646
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.964 on 16232 degrees of freedom
## Multiple R-squared:  0.3596, Adjusted R-squared:  0.3594
## F-statistic:  2278 on 4 and 16232 DF,  p-value: < 2.2e-16
```

```
plot(lm_Gtotal)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(G_total ~ VisitDay * TxGroup + sqrt(VisitDay))
Scale−Location
√|Standardized residuals|
Fitted values
lm(G_total ~ VisitDay * TxGroup + sqrt(VisitDay))



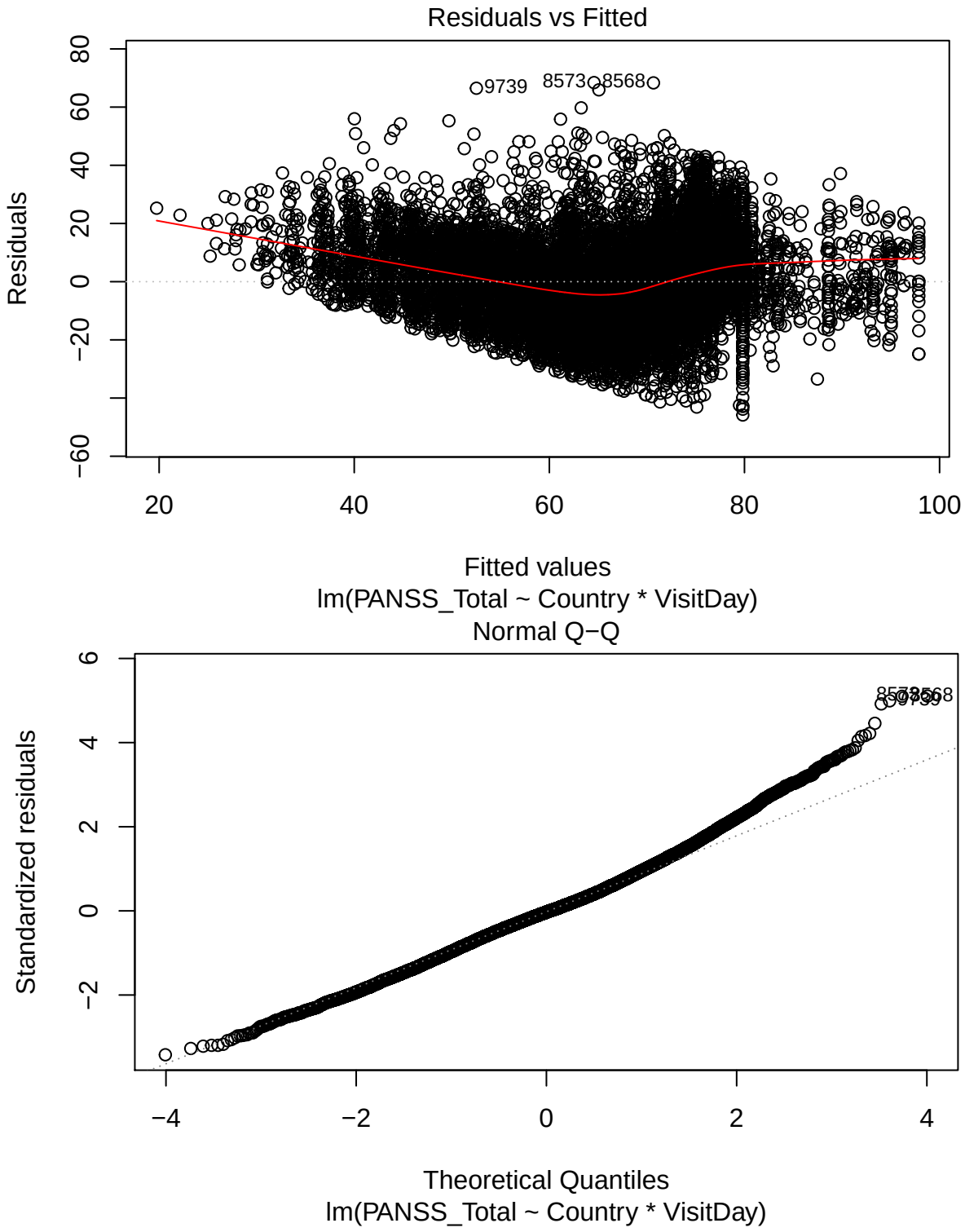
For our last model, we wanted to see if the Countries had any effect on the PANSS_Total Score.

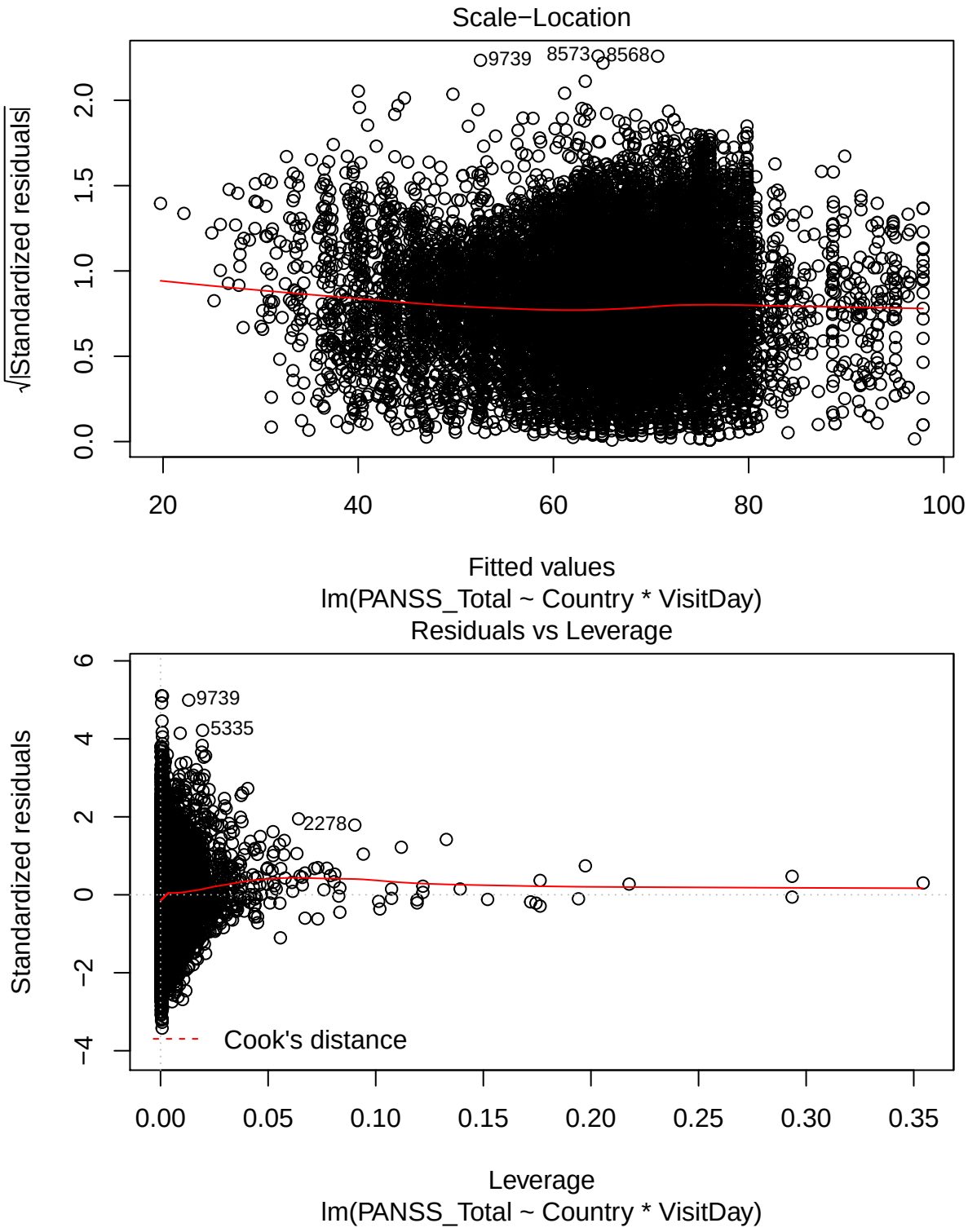
```
lm_PANSS_with_country <- lm(PANSS_Total ~ Country*VisitDay, data=processed_studies)
summary(lm_PANSS_with_country)
```

```
##
## Call:
## lm(formula = PANSS_Total ~ Country * VisitDay, data = processed_studies)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -45.828  -8.437  -0.400   7.875  68.421
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   72.350496   0.454870 159.058  < 2e-16 ***
## CountryCzech Republic          4.073117   0.918633   4.434 9.32e-06 ***
## CountryIndia                   4.170328   1.280398   3.257 0.001128 **
## CountryRomania                22.696236   1.597282  14.209  < 2e-16 ***
## CountryRussia                  7.477869   0.581304  12.864  < 2e-16 ***
## CountrySpain                   5.143375   1.322909   3.888 0.000102 ***
## CountryUkraine                 8.389526   0.810887  10.346  < 2e-16 ***
## CountryArgentina              16.254493   1.634645   9.944  < 2e-16 ***
## CountryAustralia               6.490144   2.623479   2.474 0.013376 *
## CountryBelgium                 0.774498   1.903507   0.407 0.684102
## CountryBulgaria               11.052333   1.589521   6.953 3.71e-12 ***
## CountryCanada                  5.374713   2.139756   2.512 0.012020 *
## CountryChina                   3.650544   0.544805   6.701 2.14e-11 ***
## CountryFrance                 13.173226   5.647624   2.333 0.019685 *
## CountryGermany                 7.268691   1.938956   3.749 0.000178 ***
```

```
## CountryGreece                    10.094410   2.842942   3.551 0.000385 ***
## CountryHungary                    7.723445   1.286810   6.002 1.99e-09 ***
## CountryJapan                      3.600084   0.718571   5.010 5.50e-07 ***
## CountryKorea                     -0.163997   1.828410  -0.090 0.928531
## CountryMexico                    20.822163   2.106672   9.884  < 2e-16 ***
## CountryPoland                     3.006001   1.052316   2.857 0.004288 **
## CountryPortugal                  25.521644   1.585889  16.093  < 2e-16 ***
## CountrySlovakia                   2.981237   1.350800   2.207 0.027327 *
## CountryTaiwan                     1.493068   1.871020   0.798 0.424884
## CountryUK                       -27.688605   7.275697  -3.806 0.000142 ***
## VisitDay                         -0.062175   0.002911 -21.359  < 2e-16 ***
## CountryCzech Republic:VisitDay   -0.013004   0.005280  -2.463 0.013788 *
## CountryIndia:VisitDay            -0.130832   0.012958 -10.097  < 2e-16 ***
## CountryRomania:VisitDay          -0.109861   0.014746  -7.450 9.78e-14 ***
## CountryRussia:VisitDay           -0.050243   0.003915 -12.833  < 2e-16 ***
## CountrySpain:VisitDay            -0.054585   0.008787  -6.212 5.35e-10 ***
## CountryUkraine:VisitDay          -0.031987   0.005288  -6.049 1.49e-09 ***
## CountryArgentina:VisitDay        -0.130558   0.014851  -8.791  < 2e-16 ***
## CountryAustralia:VisitDay        -0.048206   0.014605  -3.301 0.000967 ***
## CountryBelgium:VisitDay          -0.048181   0.011672  -4.128 3.68e-05 ***
## CountryBulgaria:VisitDay         -0.111201   0.015818  -7.030 2.15e-12 ***
## CountryCanada:VisitDay           -0.063217   0.014615  -4.326 1.53e-05 ***
## CountryChina:VisitDay            -0.048700   0.003577 -13.616  < 2e-16 ***
## CountryFrance:VisitDay           -0.094282   0.057515  -1.639 0.101176
## CountryGermany:VisitDay          -0.100540   0.014686  -6.846 7.87e-12 ***
## CountryGreece:VisitDay           -0.058639   0.028296  -2.072 0.038246 *
## CountryHungary:VisitDay          -0.051644   0.007735  -6.676 2.53e-11 ***
## CountryJapan:VisitDay            -0.016147   0.004238  -3.810 0.000139 ***
## CountryKorea:VisitDay            -0.018893   0.010673  -1.770 0.076703 .
## CountryMexico:VisitDay           -0.175800   0.024380  -7.211 5.80e-13 ***
## CountryPoland:VisitDay            0.004085   0.006017   0.679 0.497290
## CountryPortugal:VisitDay         -0.156673   0.012711 -12.326  < 2e-16 ***
## CountrySlovakia:VisitDay         -0.002225   0.007710  -0.289 0.772896
## CountryTaiwan:VisitDay           -0.060367   0.014111  -4.278 1.90e-05 ***
## CountryUK:VisitDay                0.026024   0.077227   0.337 0.736132
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.4 on 16187 degrees of freedom
## Multiple R-squared:  0.3766, Adjusted R-squared:  0.3747
## F-statistic: 199.5 on 49 and 16187 DF,  p-value: < 2.2e-16
```

```
plot(lm_PANSS_with_country)
```


Residuals vs Fitted
9739
8573
8568
Residuals
80
60
40
20
0
−20
−60
20
40
60
80
100
Fitted values
lm(PANSS_Total ~ Country * VisitDay)
Normal Q−Q
Standardized residuals
6
4
2
0
−2
−4
−2
0
2
4
Theoretical Quantiles
lm(PANSS_Total ~ Country * VisitDay)


Scale−Location
9739
8573
8568
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
20
40
60
80
100
Fitted values
lm(PANSS_Total ~ Country * VisitDay)
Residuals vs Leverage
9739
5335
2278
Standardized residuals
−4
−2
0
2
4
6
Cook's distance
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Leverage
lm(PANSS_Total ~ Country * VisitDay)

Appendix 2: Objective 2

Kritika Dusad & Irene Alisjahbana

August 14, 2019

This appendix contains the codes that are used for Objective 2 in order to produce the outputs seen in the final report.

Data Pre-Processing

Set the working directory and load the necessary libraries.

```
#Set working directory
path1 <- "~/Google Drive/Stanford Files/Q7 Summer 2019/"
path2 <- "STATS 202 - Data Mining and Analysis/Final Project"
setwd(paste0(path1, path2))

# Load libraries
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 3.5.2
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(car)
```

```
## Warning: package 'car' was built under R version 3.5.2
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
library(ggpubr)
```

```
## Warning: package 'ggpubr' was built under R version 3.5.2
## Loading required package: ggplot2
## Loading required package: magrittr
```

```
library(factoextra)
```

```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/13EFCZ
```

```
library(NbClust)
library(cluster)
library(scatterplot3d)
library(Hmisc)
```

```
## Warning: package 'Hmisc' was built under R version 3.5.2
## Loading required package: lattice
## Loading required package: survival
## Loading required package: Formula
##
## Attaching package: 'Hmisc'
## The following objects are masked from 'package:dplyr':
##
##     src, summarize
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

Next we will load the patients data. We will use all of the studies in order to see if there is a difference between patients that received treatment and patients that did not.

```
# Load studies
Study_A = read.csv("Study_A.csv")
Study_B = read.csv("Study_B.csv")
Study_C = read.csv("Study_C.csv")
Study_D = read.csv("Study_D.csv")
Study_E = read.csv("Study_E.csv")
```

Then, we perform pre-processing on the data.

First, we want to remove the column LeadStatus from Studies A-D because Study E does not contain this column. We don't need this column for the purpose of our objective.

```
Study_A <- subset(Study_A, select = -c(LeadStatus))
Study_B <- subset(Study_B, select = -c(LeadStatus))
Study_C <- subset(Study_C, select = -c(LeadStatus))
Study_D <- subset(Study_D, select = -c(LeadStatus))
```

Next, we combine all the studies into one dataset.

```
# Combine all studies into one dataset
All_Studies <- rbind(Study_A, Study_B, Study_C, Study_D, Study_E)
```

While looking through the data, we realized that there are observations with Country as ERROR. We want to remove these before continuing.

```
# Removed observations with Country as ERROR- 18 observations removed.
All_Studies <- subset(All_Studies, Country != "ERROR")
```

Similar to the previous objective, we remove duplicate entries and also create colmns for P, N and G total scores seperately

```
# Removing the duplicates
cols <- 9:39
processed_studies <- setDT(All_Studies)[, lapply(.SD,mean),
                                        by=c("Country", "PatientID", "VisitDay",
                                             "TxGroup"), .SDcols = cols]

# Taking ceiling of Ps, Ns, and Gs.
processed_studies <- processed_studies %>% mutate_at(5:34, ceiling)

# Calculating PANSS_Total with new Ps, Ns, and Gs.
processed_studies$PANSS_Total <- rowSums(processed_studies[5:34])

#Combining Ns, Ps, and Gs scores. Making separate columns for them.
processed_studies$P_total <- rowSums(processed_studies[5:11])
processed_studies$N_total <- rowSums(processed_studies[12:18])
processed_studies$G_total <- rowSums(processed_studies[19:34])
```

Because we are trying to segment the schizophrena patients into k groups, we are only using the baseline measurements for each patient (VisitDay = 0). Therefore we would like to segment our data to only include VisitDay = 0 observations.

```
# VisitDay = 0
base_data <- subset(processed_studies, VisitDay==0)
```

For this objective, we decide to use the total P, N and G values in addition to the country as our parameters. Because there are over 25 countries included in the dataset, we decide to group the patients based on their continent to reduce the dimensionality. We then change it into one-hot encodings. Finally, we scaled the data.

```
# Take Country and P, N, and G totals as parameters
cluster_data <- subset(base_data[,c(1,36:38)])

# Group into continents
australasia <- c("India", "Korea", "China", "Taiwan",
                 "Japan", "Russia", "Australia")
europe <- c("Ukraine", "Spain", "Romania", "Czech Republic",
            "Poland", "Portugal", "Bulgaria","Germany",
            "Greece", "Slovakia", "France", "Hungary", "Belgium",
            "Austria", "Sweden", "UK")
northamerica <- c("USA", "Canada")
southamerica <- c("Argentina", "Mexico","Brazil")

for (i in 1:nrow(cluster_data)){
  if(cluster_data$Country[i] %in% australasia){
    cluster_data$Continent[i] <- "Australasia"
  } else if (cluster_data$Country[i] %in% europe){
    cluster_data$Continent[i] <- "Europe"
  } else if (cluster_data$Country[i] %in% northamerica){
    cluster_data$Continent[i] <- "North America"
  } else if (cluster_data$Country[i] %in% southamerica){
    cluster_data$Continent[i] <- "South America"
```

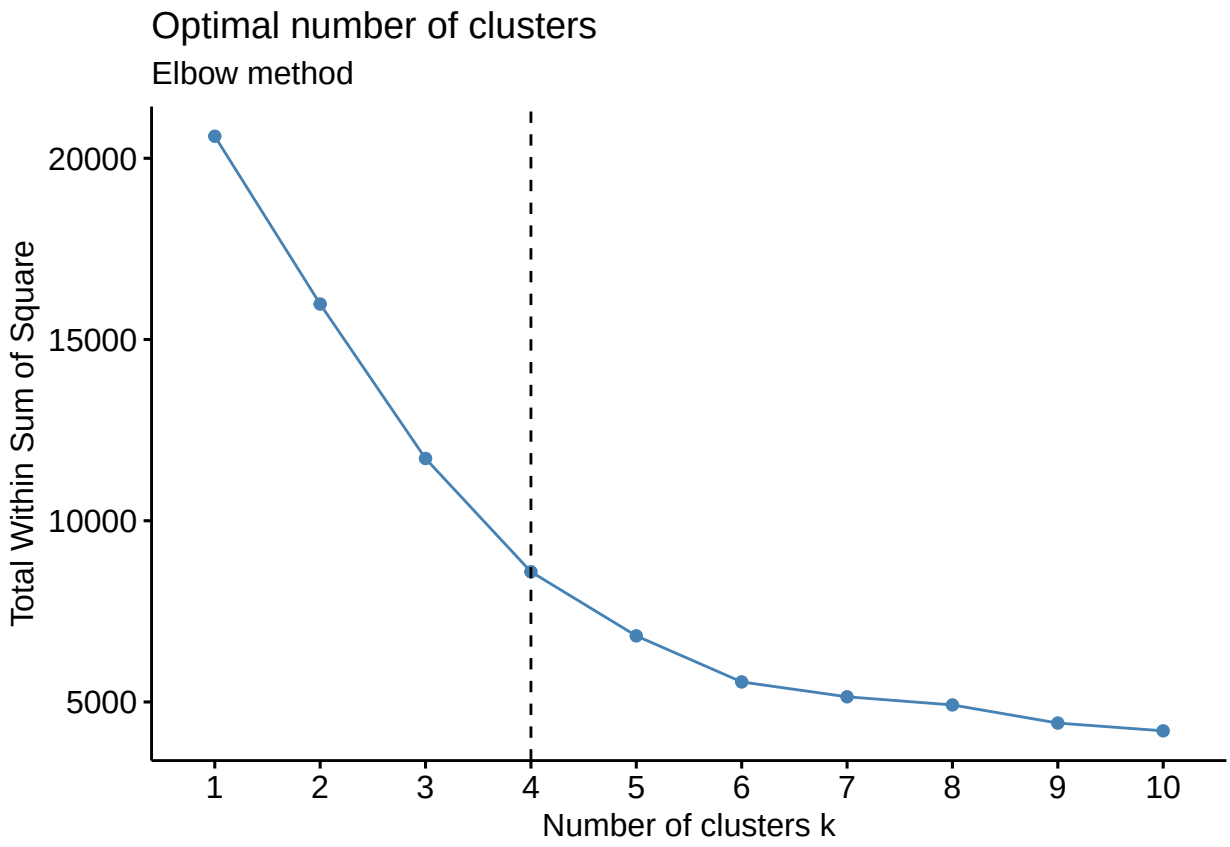
```
  }
}

# Convert as factors
cluster_data$Continent <- as.factor(cluster_data$Continent)

# Remove Countries
cluster_data <- subset(cluster_data, select = -c(Country))

# Change into one hot encoding
cluster_data.ohe <- model.matrix(~.-1, data=cluster_data)

# Scale data
cluster_data.ohe <- scale(cluster_data.ohe)
```

Determining optimal number of clusters

There are several methods to determine the optimal number of clusters. For this objective, we used three methods: Elbow Method, Average Silhouette Method, and Gap Statistics Method.

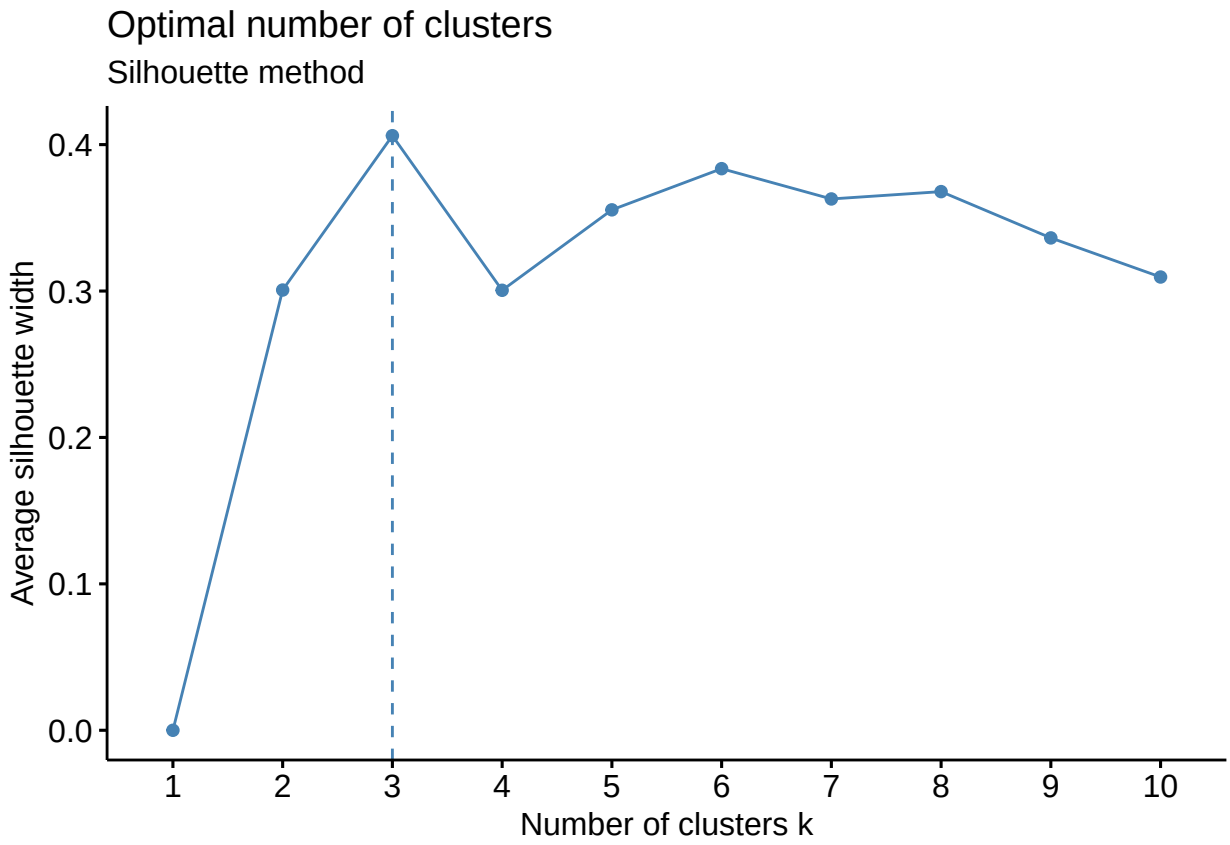
Elbow Method

```
set.seed(123) # for reproducibility
fviz_nbclust(cluster_data.ohe, kmeans, method = "wss") +
  geom_vline(xintercept = 4, linetype = 2)+
  labs(subtitle = "Elbow method")
```



Average Silhouette Method

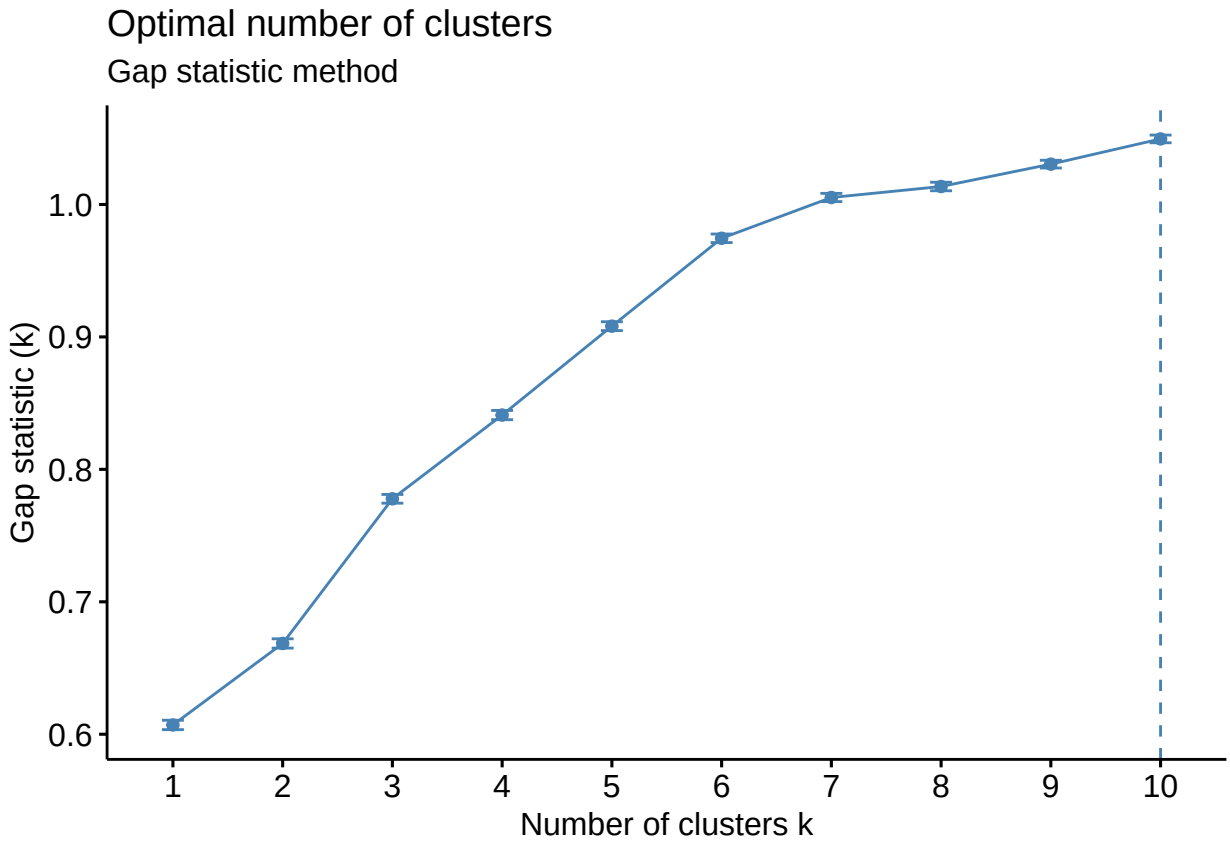
```
set.seed(123) # for reproducibility
fviz_nbclust(cluster_data.ohe, kmeans, method = "silhouette") +
  labs(subtitle = "Silhouette method")
```



Gap Statistic

```
set.seed(123)
fviz_nbclust(cluster_data.ohe, kmeans, nstart = 25,
             method = "gap_stat", nboot = 50, iter.max=50) +
  labs(subtitle = "Gap statistic method", verbose = FALSE)
```

```
## Warning: Quick-TRANSfer stage steps exceeded maximum (= 147250)

## Warning: Quick-TRANSfer stage steps exceeded maximum (= 147250)

## Warning: Quick-TRANSfer stage steps exceeded maximum (= 147250)
```



Perform K-Means Clustering

The three methods show different numbers of optimal cluster. Therefore, we pick 3 clusters. We can visualize our clusters using a 3D scatterplot:

```
set.seed(123) # for Reproducibility
clustering<- kmeans(cluster_data.ohe, 3, nstart=25)

clustering$centers # Display center
```

```
##       P_total     N_total     G_total ContinentAustralasia ContinentEurope
## 1 -0.05080931  0.09897914  0.13885511           -0.9641402       1.6895751
## 2  0.35708259 -0.20862769 -0.08721740           -0.9641402      -0.4908801
## 3 -0.19342897  0.08063603 -0.01183391            1.0368414      -0.4908801
##   ContinentNorth America ContinentSouth America
## 1             -0.6437115              0.6145003
## 2              1.5529634             -0.1785336
## 3             -0.6437115             -0.1785336
```

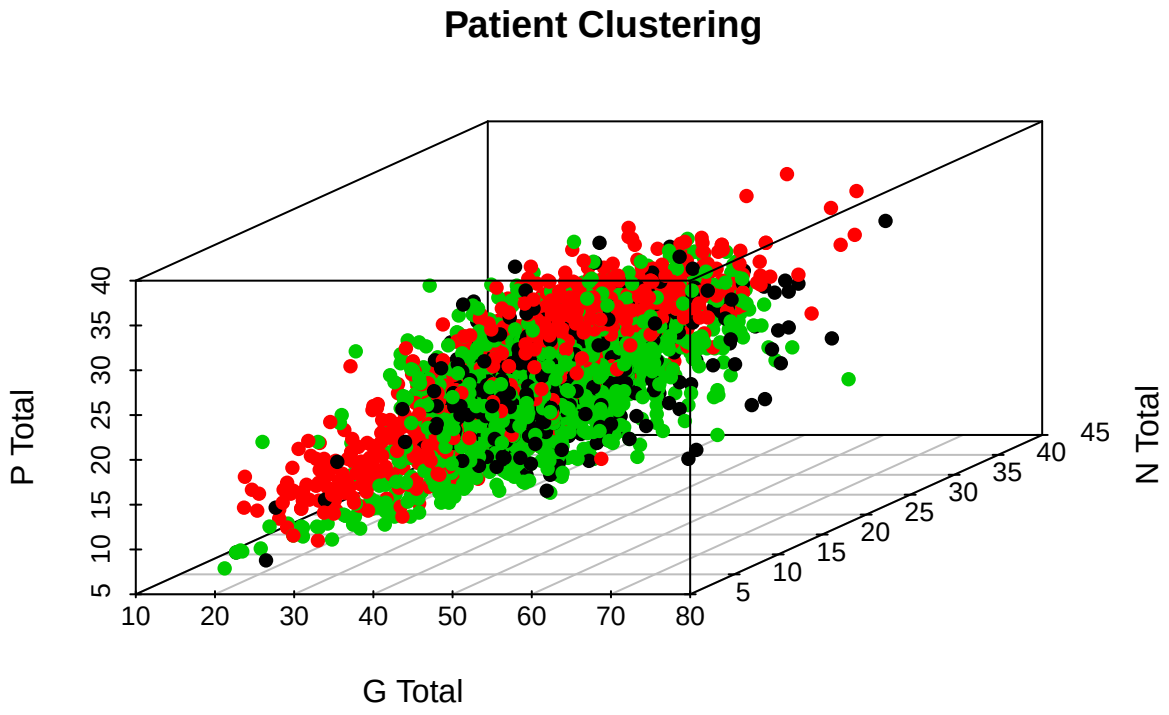
```
table(clustering$cluster)
```

```
##
##    1    2    3
##  663  863 1419
```

Finally we can visualize our data:

```
cluster_df = data.frame(cluster_data.ohe)
s3d <- scatterplot3d(cluster_data$G_total, cluster_data$N_total,
```

```
                    cluster_data$P_total, color=clustering$cluster, pch=16,
                    xlab ="G Total", ylab = "N Total", zlab = "P Total",
                    main = "Patient Clustering")
```



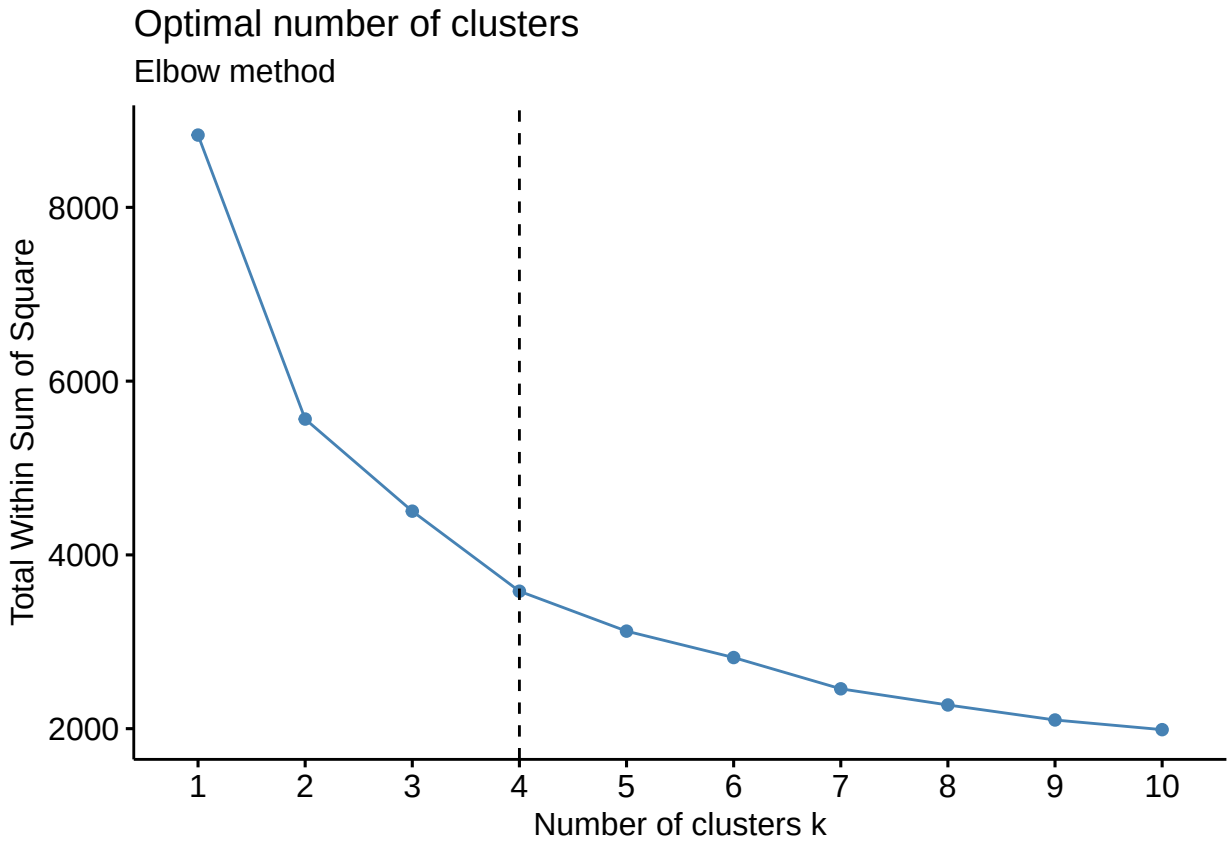
K-means Clustering without Countries

For this approach, we only use the total P, N, G and not include the country. We scale the data beforehand.

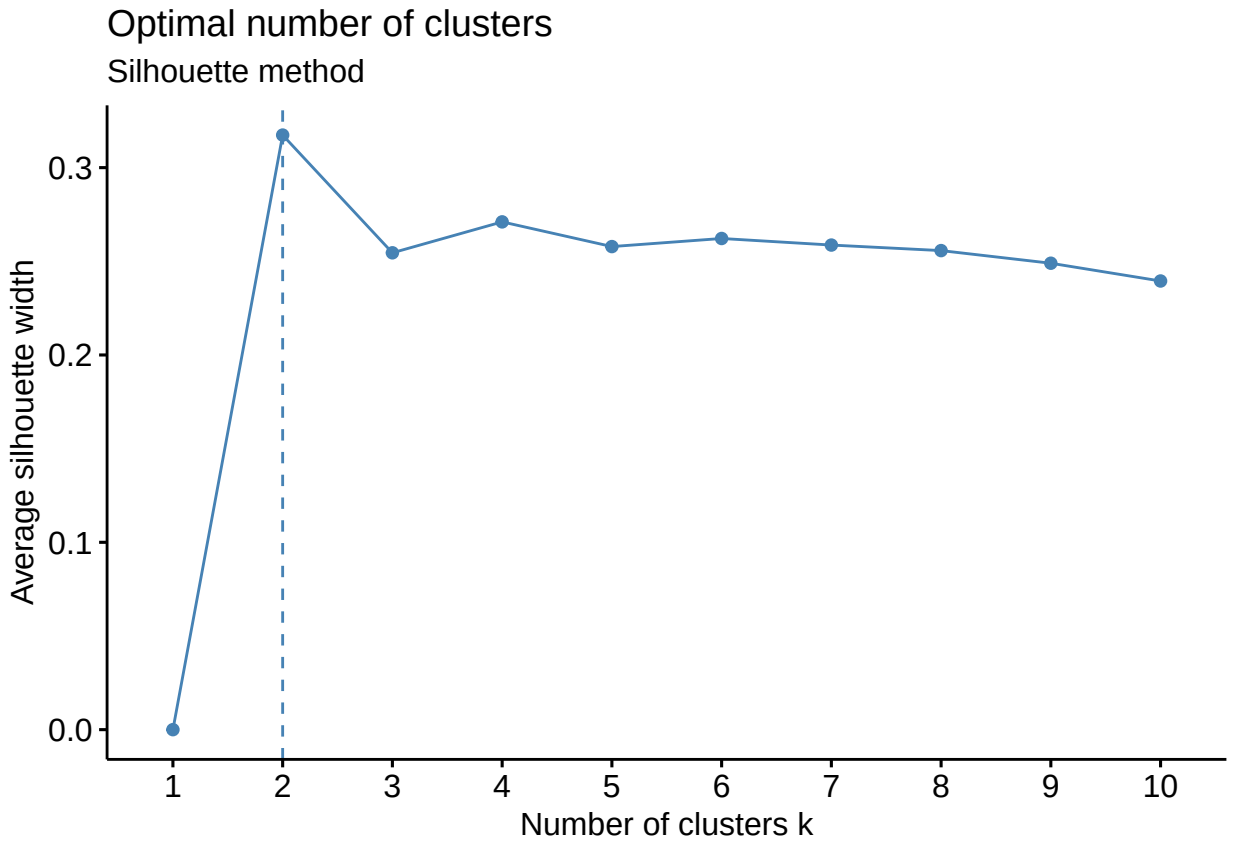
```
# Take P, N and G total
cluster_data <- subset(base_data[,c(36:38)])

# Scale data
cluster_data <- scale(cluster_data)
```

Similiary, we use three methods to determine the optimal number of clusters.

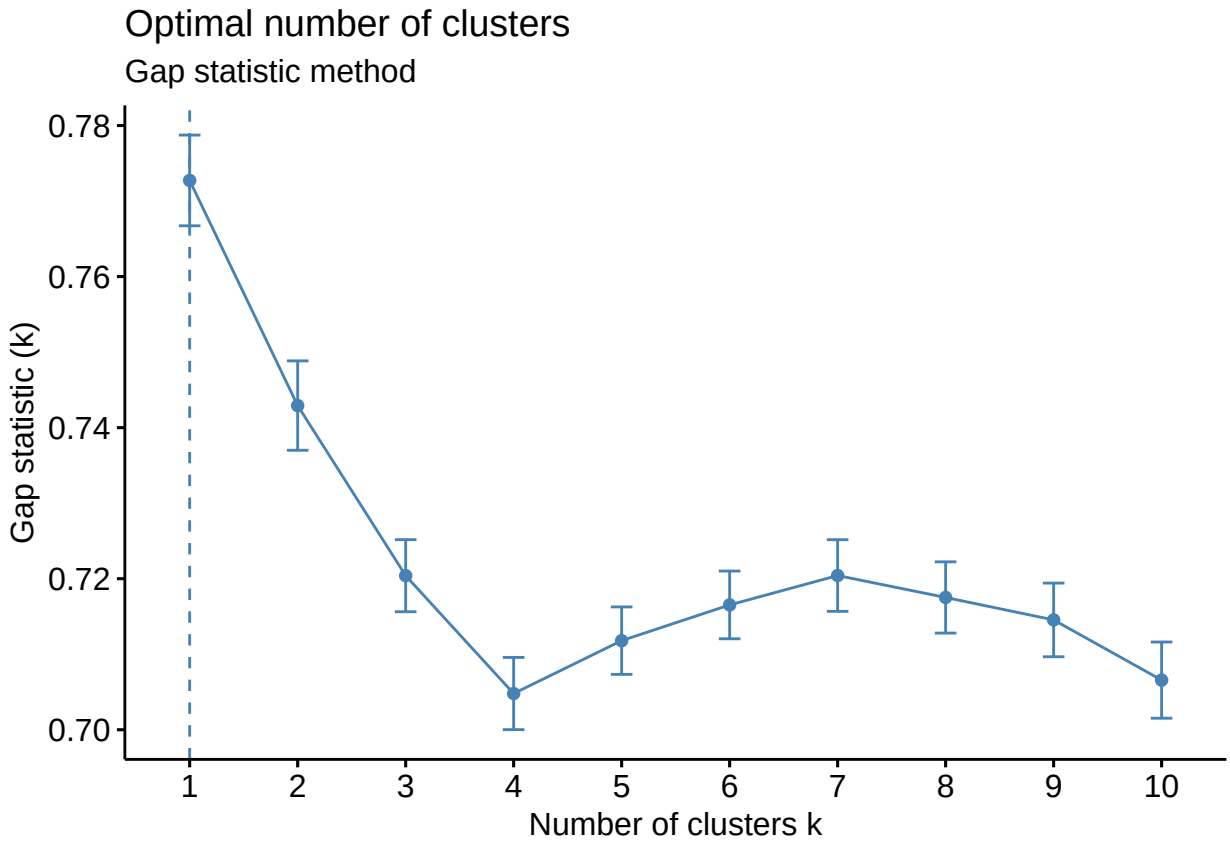
```
set.seed(123) # for reproducibility
fviz_nbclust(cluster_data, kmeans, method = "wss") +
  geom_vline(xintercept = 4, linetype = 2)+
  labs(subtitle = "Elbow method")
```




```
set.seed(123) # for reproducibility
fviz_nbclust(cluster_data, kmeans, method = "silhouette") +
  labs(subtitle = "Silhouette method")
```




```
set.seed(123)
fviz_nbclust(cluster_data, kmeans, nstart = 25,
             method = "gap_stat", nboot = 50, iter.max=50) +
  labs(subtitle = "Gap statistic method", verbose = FALSE)
```



We choose 2 clusters and perform k-means clustering.

```
set.seed(123) # for Reproducibility
clustering<- kmeans(cluster_data, 2, nstart=25)

clustering$centers # Display center
```

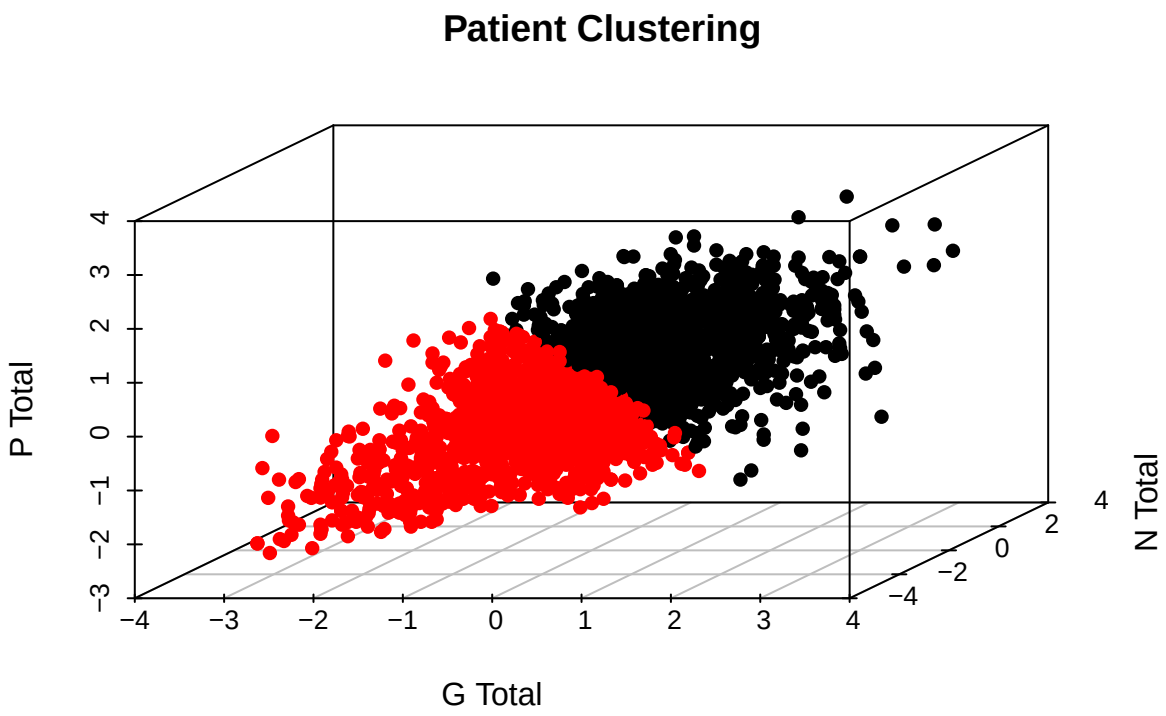
```
##      P_total    N_total    G_total
## 1  0.7041483  0.4604875  0.7525578
## 2 -0.6133338 -0.4010981 -0.6554998
```

```
table(clustering$cluster)
```

```
##
##    1    2
## 1371 1574
```

Finally we can visualize our data:

```
cluster_df = data.frame(cluster_data)
s3d <- scatterplot3d(cluster_df$G_total, cluster_df$N_total,
                     cluster_df$P_total, color=clustering$cluster, pch=16,
                     xlab ="G Total", ylab = "N Total", zlab = "P Total",
                     main = "Patient Clustering")
```


Patient Clustering
P Total
G Total
N Total
−4
−3
−2
−1
0
1
2
3
4

Appendix 3: Objective 3

Kritika Dusad & Irene Alisjahbana

August 14, 2019

This appendix contains the codes that are used for Objective 3 in order to produce the outputs seen in the final report.

Data Pre-Processing

Set the working directory and load the necessary libraries.

```
#Set working directory
path1 <- "~/Google Drive/Stanford Files/Q7 Summer 2019/"
path2 <- "STATS 202 - Data Mining and Analysis/Final Project"
setwd(paste0(path1,path2))

# Load libraries
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
library(ggpubr)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: magrittr
```

```
library(ggplot2)
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
library(FactoMineR)
library(mlr)
```

```
## Loading required package: ParamHelpers
```

```
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
library(Matrix)
library(VIM)
```

```
## Loading required package: colorspace
```

```
## Loading required package: grid
```

```
## VIM is ready to use.
##  Since version 4.0.0 the GUI is in its own package VIMGUI.
##
##           Please use the package to use the new (and old) GUI.
```

```
## Suggestions and bug-reports can be submitted at: https://github.com/alexkowa/VIM/issues
```

```
##
## Attaching package: 'VIM'
```

```
## The following object is masked from 'package:datasets':
##
##     sleep
```

```
library(Hmisc)
```

```
## Loading required package: lattice
```

```
## Loading required package: survival
```

```
## Loading required package: Formula
```

```
##
## Attaching package: 'Hmisc'
```

```
## The following object is masked from 'package:mlr':
##
##     impute
```

```
## The following objects are masked from 'package:dplyr':
##
##     src, summarize
```

```
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
library(rms)
```

```
## Loading required package: SparseM
```

```
##
## Attaching package: 'SparseM'
```

```
## The following object is masked from 'package:base':
##
##     backsolve
```

```
##
## Attaching package: 'rms'

## The following objects are masked from 'package:car':
##
##     Predict, vif
```

```
library(glmnet)
```

```
## Loading required package: foreach

## Loaded glmnet 2.0-18
```

```
library(randomForest)
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(rsample)
```

```
## Loading required package: tidyr

##
## Attaching package: 'tidyr'

## The following object is masked from 'package:Matrix':
##
##     expand

## The following object is masked from 'package:magrittr':
##
##     extract
```

```
library(caret)
```

```
##
## Attaching package: 'caret'

## The following object is masked from 'package:survival':
##
##     cluster

## The following object is masked from 'package:mlr':
##
##     train
```

```
library(vtreat)
```

```
##
## Attaching package: 'vtreat'
```

```
## The following object is masked from 'package:VIM':
##
##     prepare
```

```
library(magrittr)
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library(zoo)
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

First, we will load the patients data. We will only use Studies A-D for this objective.

```
# Load studies
Study_A = read.csv("Study_A.csv")
Study_B = read.csv("Study_B.csv")
Study_C = read.csv("Study_C.csv")
Study_D = read.csv("Study_D.csv")
```

Next, we combine all the studies into one dataset.

```
# Combine all studies into one dataset
all_studies <- rbind(Study_A, Study_B, Study_C, Study_D)
```

While looking through the data, we realized that there are observations with Country as ERROR. We want to remove these before continuing.

```
# Removed observations with Country as ERROR- 18 observations removed.
all_studies <- subset(all_studies, Country != "ERROR")
```

Because we want to predict the patient's PANSS score on the 18th week, we take the patients with observations greater than that week.

```
# Subset if patient has visitday greater than 120.
studies_subset <- subset(all_studies, VisitDay>=120)
IDs <- studies_subset["PatientID"][,1]
all_studies <- subset(filter(all_studies, all_studies$PatientID %in% IDs))
```

Similar to the previous objective, we remove duplicate entries and also create colmns for P, N and G total scores seperately

```
# Removing the duplicates
cols <- 9:39
processed_studies <- setDT(all_studies)[, lapply(.SD,mean),
                                        by=c("Country", "PatientID", "VisitDay", "TxGroup"),
                                        .SDcols = cols]

# Taking ceiling of Ps, Ns, and Gs.
processed_studies <- processed_studies %>% mutate_at(5:34, ceiling)
```

```
# Calculating PANSS_Total with new Ps, Ns, and Gs.
processed_studies$PANSS_Total <- rowSums(processed_studies[5:34])

#Combining Ns, Ps, and Gs scores. Making separate columns for them.
processed_studies$P_total <- rowSums(processed_studies[5:11])
processed_studies$N_total <- rowSums(processed_studies[12:18])
processed_studies$G_total <- rowSums(processed_studies[19:34])
```

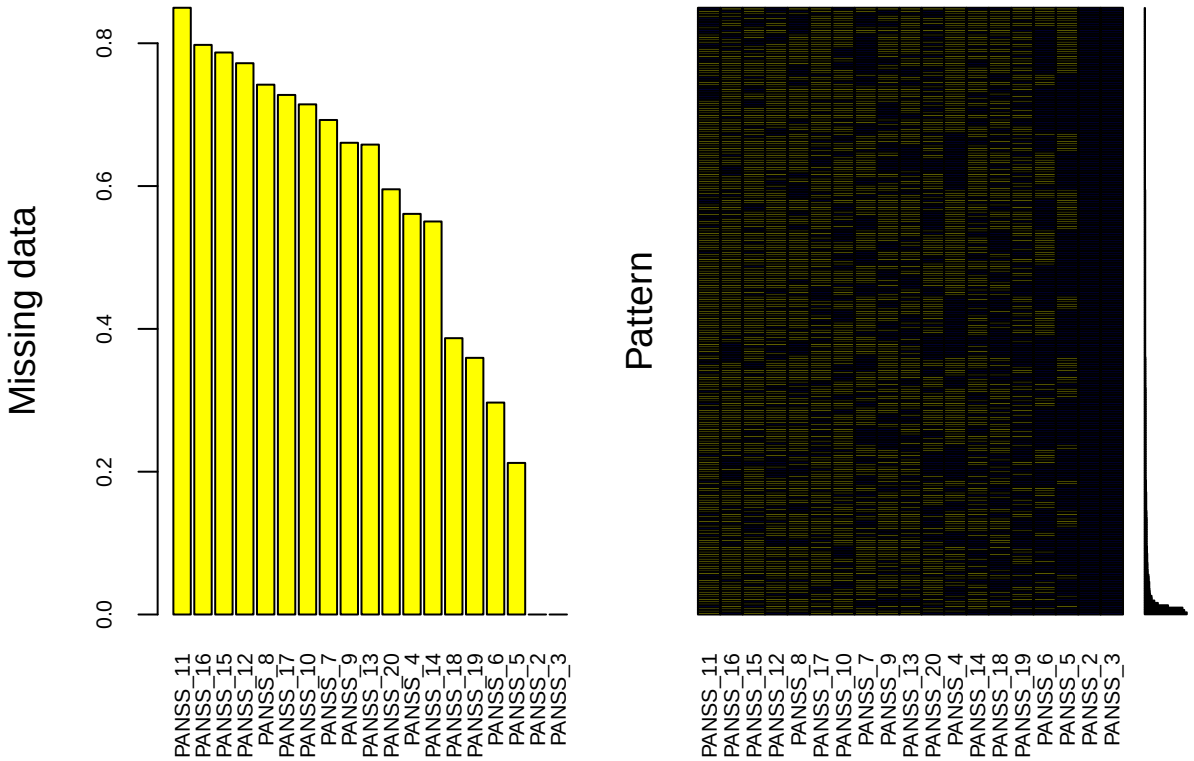
We then want to make a dataset that contains each patients observation every week leading up to the 18th week.

```
# Only take Country, PatientID, visitDay and PANSS_Total
weekly_studies <- subset(processed_studies, select = c(1, 2,3,35))

# Populate new dataframe with VisitDay = 0
visitday_0 <- subset(weekly_studies, VisitDay==0)
IDs <- visitday_0["PatientID"][,1]
weekly_PANSS <- data.frame(PatientID = IDs)
merge_weekly <- merge(weekly_PANSS, visitday_0 , by="PatientID", all= TRUE)
names(merge_weekly)[ncol(merge_weekly)] <- "PANSS_0"

# Populate dataframe for each week
for (i in 1:69){
    week <- subset(processed_studies, VisitDay<=i*7 & VisitDay > i*7-7)

  merge_weekly <- merge(merge_weekly, week[,c("PatientID", "PANSS_Total")],
                        by="PatientID", all= TRUE)
  col=ncol(merge_weekly)
  names(merge_weekly)[col] <- paste("PANSS_",i, sep = "")
}
```

We then want to visualize the missing data for Week 0 - Week 18.

```
panss_plot <- aggr(merge_weekly[,3:21], col=c('navyblue','yellow'),
                   numbers=TRUE, sortVars=TRUE,
                   labels=names(merge_weekly[,6:24]), cex.axis=.7,
                   gap=3, ylab=c("Missing data","Pattern"))
```

```
## Warning in plot.aggr(res, ...): not enough vertical space to display
## frequencies (too many combinations)
```




```
##
##  Variables sorted by number of missings:
##  Variable     Count
##  PANSS_11 0.8497182
##  PANSS_16 0.7977458
##  PANSS_15 0.7871008
##  PANSS_12 0.7720726
##   PANSS_8 0.7420163
##  PANSS_17 0.7276143
##  PANSS_10 0.7144646
##   PANSS_7 0.6925485
##   PANSS_9 0.6606137
##  PANSS_13 0.6581090
##  PANSS_20 0.5954915
##   PANSS_4 0.5610520
##  PANSS_14 0.5504070
##  PANSS_18 0.3869756
##  PANSS_19 0.3594239
##   PANSS_6 0.2968065
##   PANSS_5 0.2122730
##   PANSS_2 0.0000000
##   PANSS_3 0.0000000
```

Here we also drop other unecessary columns.

```
drop <- c("PatientID", "VisitDay")
merge_weekly <- merge_weekly[,!(names(merge_weekly) %in% drop)]
```

For missing values in Week 18, we take the next closest value.

```
# Take the next closest value
week18_idx <- grep("PANSS_18", colnames(merge_weekly))
numcol <- ncol(merge_weekly);
end_weeks <- merge_weekly[,c(week18_idx:numcol)]
end_weeks <- data.frame(t(na.locf(data.frame(t(end_weeks)), fromLast = TRUE)))
merge_weekly <- merge_weekly[, -c(week18_idx:numcol)]

# Combine back to original data
merge_weekly$PANSS_18 <- end_weeks$PANSS_18
```

Finally we split the dataset into 70% training and 30% test set.

```
set.seed(123)
split <- initial_split(merge_weekly, prop= 0.7)
train <- training(split)
test<- testing(split)
```

Linear Regression

For linear regression, we first have to impute the missing values.

```
# Impute missing value
mi <- aregImpute(formula = ~ PANSS_1 + PANSS_2 + PANSS_3 + PANSS_4 + PANSS_5+
                  PANSS_6 + PANSS_7 + PANSS_8 + PANSS_9 + PANSS_10 + PANSS_11 +
                  PANSS_12 + PANSS_13 + PANSS_14 + PANSS_15+ PANSS_16 + PANSS_17 + PANSS_18,
                data=merge_weekly, n.impute = 20, burnin=3, nk=3, tlinear = FALSE,
                type='regression', pmmtype=1)
```

```
## Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
Iteration 11
Iteration 12
Iteration 13
Iteration 14
Iteration 15
Iteration 16
Iteration 17
Iteration 18
Iteration 19
Iteration 20
Iteration 21
Iteration 22
Iteration 23
```

```
# view multiple imputation summary
mi
```

```
##
```

```
## Multiple Imputation using Bootstrap and PMM
##
## aregImpute(formula = ~PANSS_1 + PANSS_2 + PANSS_3 + PANSS_4 +
##     PANSS_5 + PANSS_6 + PANSS_7 + PANSS_8 + PANSS_9 + PANSS_10 +
##     PANSS_11 + PANSS_12 + PANSS_13 + PANSS_14 + PANSS_15 + PANSS_16 +
##     PANSS_17 + PANSS_18, data = merge_weekly, n.impute = 20,
##     nk = 3, tlinear = FALSE, type = "regression", pmmtype = 1,
##     burnin = 3)
##
## n: 1597  p: 18   Imputations: 20     nk: 3
##
## Number of NAs:
##  PANSS_1  PANSS_2  PANSS_3  PANSS_4  PANSS_5  PANSS_6  PANSS_7  PANSS_8
##      896      339      474     1106     1185     1055     1141     1357
##  PANSS_9 PANSS_10 PANSS_11 PANSS_12 PANSS_13 PANSS_14 PANSS_15 PANSS_16
##     1233     1051      879     1257     1274     1162      618      574
## PANSS_17 PANSS_18
##      951        0
##
##          type d.f.
## PANSS_1     s    2
## PANSS_2     s    2
## PANSS_3     s    2
## PANSS_4     s    2
## PANSS_5     s    2
## PANSS_6     s    2
## PANSS_7     s    2
## PANSS_8     s    2
## PANSS_9     s    2
## PANSS_10    s    2
## PANSS_11    s    2
## PANSS_12    s    2
## PANSS_13    s    2
## PANSS_14    s    2
## PANSS_15    s    2
## PANSS_16    s    2
## PANSS_17    s    2
## PANSS_18    s    2
##
## R-squares for Predicting Non-Missing Values for Each Variable
## Using Last Imputations of Predictors
##  PANSS_1  PANSS_2  PANSS_3  PANSS_4  PANSS_5  PANSS_6  PANSS_7  PANSS_8
##    0.864    0.916    0.873    0.939    0.934    0.882    0.914    0.914
##  PANSS_9 PANSS_10 PANSS_11 PANSS_12 PANSS_13 PANSS_14 PANSS_15 PANSS_16
##    0.943    0.981    0.951    0.980    0.959    0.986    0.968    0.934
## PANSS_17
##    0.925
```

Then we put the imputed values into our original dataset.

```
# Put back into original dataset
weekly_complete <- impute.transcan(mi, imputation=20, data=merge_weekly,
                                   list.out=TRUE,pr=FALSE, check=FALSE)
weekly_complete <- data.frame(weekly_complete)
merge_weekly_complete <- cbind(merge_weekly[,c(1,2)], weekly_complete)
```

Similarly, we split the training and test set.

```
#Splitting dataset into 70% training and 30% test set for complete dataset
set.seed(123)
split_complete <- initial_split(merge_weekly_complete, prop= 0.7)
train_complete <- training(split_complete)
test_complete <- testing(split_complete)
```

Finally we fit our linear model.

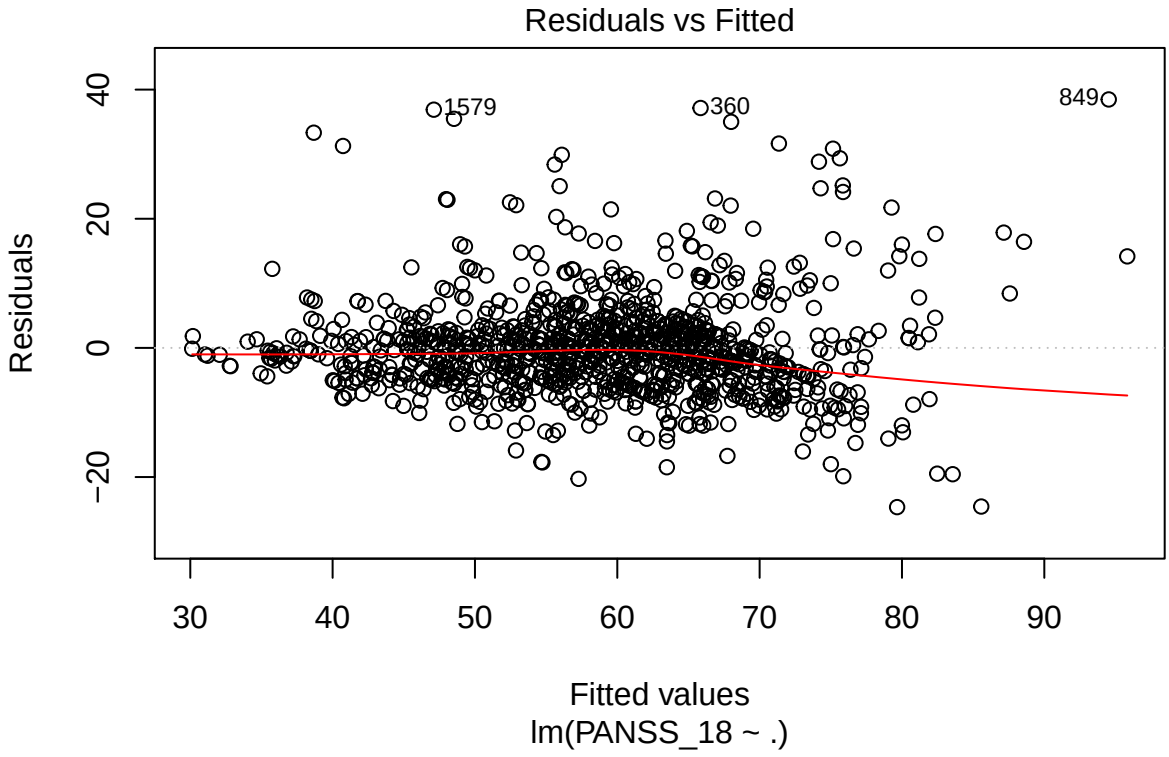
```
# Fit model
lm.fit = lm(PANSS_18 ~ ., data = train_complete)
summary(lm.fit)
```

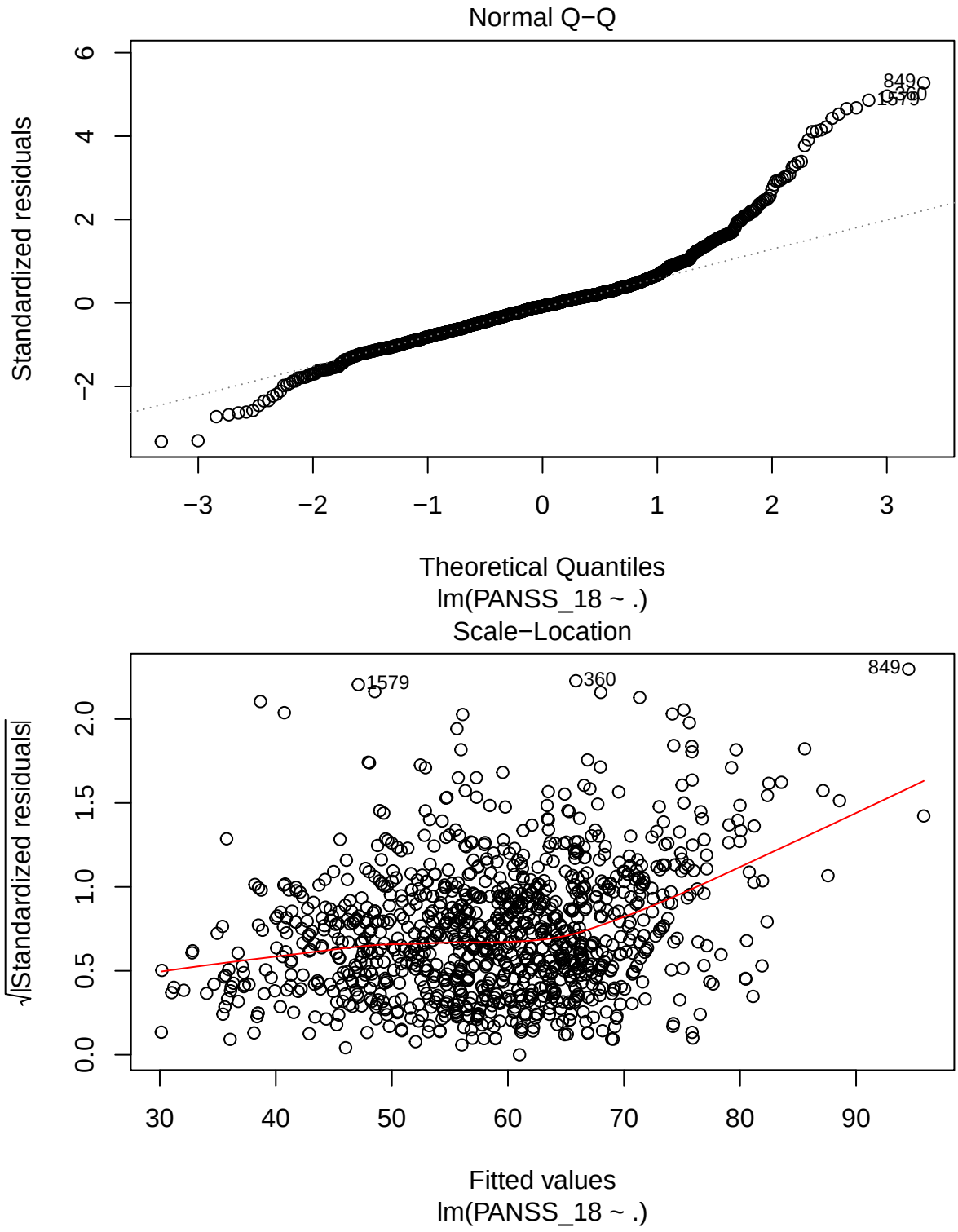
```
##
## Call:
## lm(formula = PANSS_18 ~ ., data = train_complete)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -24.659  -4.325  -0.606   2.721  38.474
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)             4.677686   2.497756   1.873 0.061373 .
## CountryCzech Republic   2.483905   1.415714   1.755 0.079625 .
## CountryIndia           -2.642457   1.874385  -1.410 0.158896
## CountryRomania          3.921967   2.093559   1.873 0.061291 .
## CountryRussia           0.671202   1.027851   0.653 0.513886
## CountrySpain            0.866859   1.952165   0.444 0.657096
## CountryUkraine          0.598096   1.309940   0.457 0.648063
## CountryArgentina       -1.677252   2.060457  -0.814 0.415814
## CountryAustralia       -1.414535   3.546620  -0.399 0.690090
## CountryBelgium         -0.947687   2.852482  -0.332 0.739778
## CountryBulgaria         3.043802   2.030184   1.499 0.134096
## CountryCanada           0.322382   3.262385   0.099 0.921301
## CountryChina            1.601152   0.931894   1.718 0.086053 .
## CountryFrance          -2.255587   7.759841  -0.291 0.771356
## CountryGermany         -2.085680   2.480598  -0.841 0.400648
## CountryGreece           8.124903   3.042124   2.671 0.007681 **
## CountryHungary         -2.080267   1.904425  -1.092 0.274931
## CountryJapan           -0.207269   1.165636  -0.178 0.858901
## CountryKorea           -1.039170   2.579002  -0.403 0.687076
## CountryMexico           2.878989   2.465507   1.168 0.243184
## CountryPoland           0.549966   1.719522   0.320 0.749154
## CountryPortugal        -1.960859   2.075114  -0.945 0.344901
## CountrySlovakia        -2.085113   2.281086  -0.914 0.360875
## CountryTaiwan          -0.362583   2.834902  -0.128 0.898252
## PANSS_0                -0.018128   0.033018  -0.549 0.583110
## PANSS_1                -0.016955   0.035494  -0.478 0.632968
## PANSS_2                 0.041997   0.041010   1.024 0.306035
## PANSS_3                -0.025623   0.028873  -0.887 0.375050
## PANSS_4                 0.013929   0.020028   0.695 0.486896
## PANSS_5                 0.033713   0.025746   1.309 0.190666
## PANSS_6                 0.089861   0.022420   4.008 6.54e-05 ***
```

```
## PANSS_7                0.045688   0.024449   1.869 0.061931 .
## PANSS_8                0.004448   0.027277   0.163 0.870495
## PANSS_9               -0.076074   0.020801  -3.657 0.000267 ***
## PANSS_10              -0.018692   0.022152  -0.844 0.398988
## PANSS_11              -0.088177   0.025971  -3.395 0.000711 ***
## PANSS_12              -0.001500   0.018348  -0.082 0.934841
## PANSS_13               0.012343   0.021579   0.572 0.567439
## PANSS_14               0.043864   0.024104   1.820 0.069063 .
## PANSS_15               0.300852   0.035910   8.378  < 2e-16 ***
## PANSS_16               0.445793   0.027989  15.928  < 2e-16 ***
## PANSS_17               0.086881   0.011205   7.754 2.06e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.607 on 1076 degrees of freedom
## Multiple R-squared:  0.6522, Adjusted R-squared:  0.639
## F-statistic: 49.22 on 41 and 1076 DF,  p-value: < 2.2e-16
```

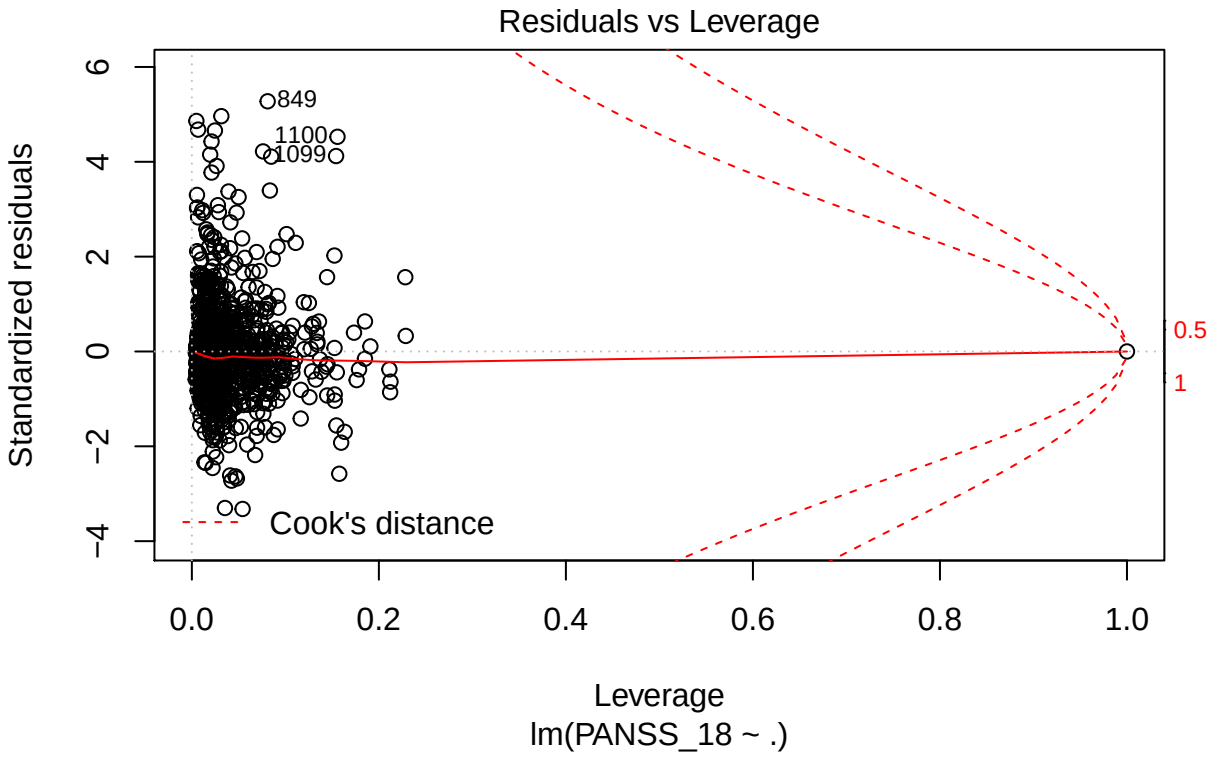
We can visualize the residuals of our fit.

```
plot(lm.fit)
```






```
## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced

## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced
```



And then predict on the test set to find the test MSE.

```
lm.pred = predict(lm.fit, test_complete)
mse_lm  = mean((test_complete$PANSS_18 - lm.pred)^2)
mse_lm
```

```
## [1] 53.03279
```

Linear Regression with Lasso

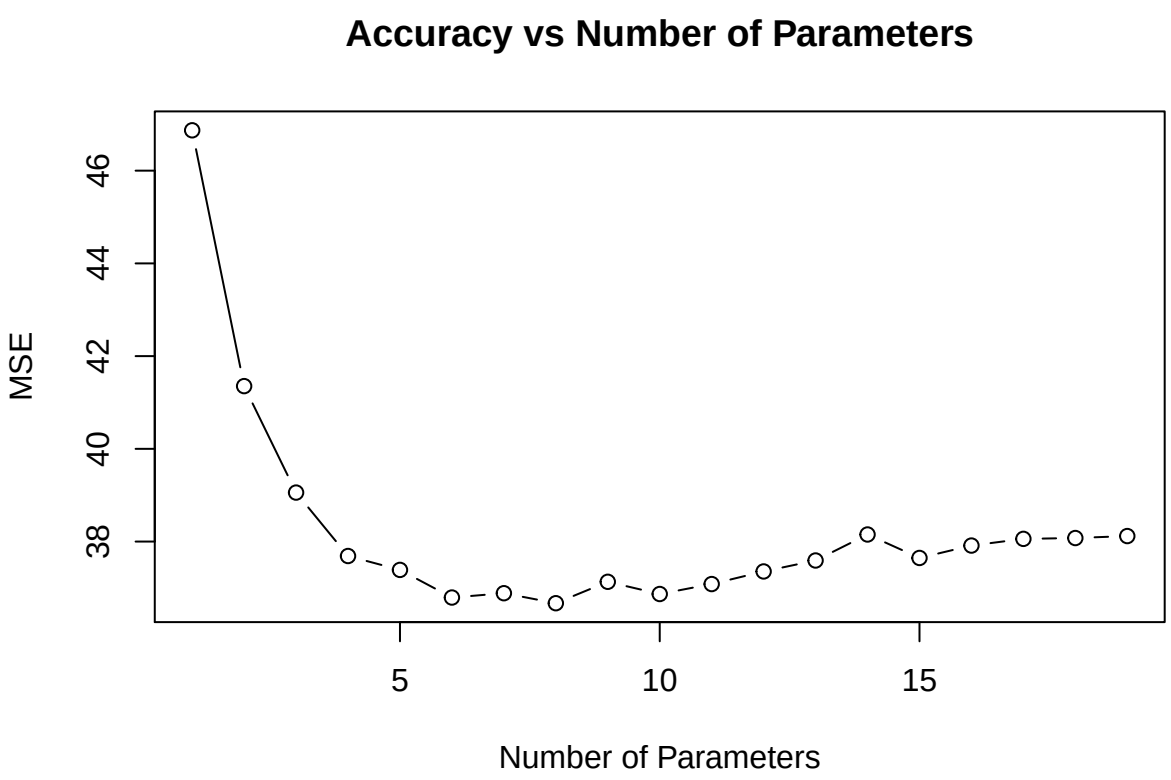
```
# Preprocess data
set.seed(123)
x = model.matrix(PANSS_18 ~ ., data = train_complete)
y = train_complete$PANSS_18
x.test = model.matrix(PANSS_18 ~ ., data = test_complete)

# Fit model
lasso.fit = glmnet(x, y, alpha = 1)
lasso.pred = predict(lasso.fit, s = 0.01, newx = x.test)

# Predict values
mse_lasso = mean((test_complete$PANSS_18 - lasso.pred)^2)
mse_lasso
```

```
## [1] 52.99754
```

Random Forest

```
set.seed(123) # For reproducibility
num_param = ncol(train_complete)-1
```

```
mse=c()

for(i in 1:num_param){
  rf.panss = randomForest(PANSS_18~.,data = train_complete,
                          mtry=i, importance=TRUE, ntree=1000)

  rf.pred = predict(rf.panss, test_complete)
  mse[i] = mean((test_complete$PANSS_18 - rf.pred)^2)

}

# Visualize accuracy
plot(1:num_param,mse,type='b', main = "Accuracy vs Number of Parameters",
     xlab = "Number of Parameters", ylab = "MSE")
```




```
# Use minimum MSE
best_mse = which.min(mse)

# Fit model
set.seed(123)
rf.panss = randomForest(PANSS_18 ~ ., data = train_complete, ntree = 10000, mtry = best_mse) # 19

# Predict values
rf.pred = predict(rf.panss, test_complete)
mse_rf = mean((test_complete$PANSS_18 - rf.pred)^2)
mse_rf
```

```
## [1] 36.90424
```

Gradient Boosting

First we want to randomize our data.

```
# randomize data
random_index <- sample(1:nrow(train), nrow(train))
random_train <- train[random_index, ]
```

Then we create a hyperparameter grid to tune our model.

```
# create hyperparameter grid
hyper_grid <- expand.grid(
  shrinkage = c(0.01, 0.05, 0.1),
  interaction.depth = c(9, 11, 13, 15),
  n.minobsinnode = c(1, 3),
  bag.fraction = c(0.8, 0.9, 1),
  optimal_trees = 0,
  min_MSE = 0
)
```

We then train our model and display the top 10 results.

```
# grid search
for(i in 1:nrow(hyper_grid)) {

  # reproducibility
  set.seed(123)

  # train model
  gbm.tune <- gbm(
    formula = PANSS_18 ~ .,
    distribution = "gaussian",
    data = random_train,
    n.trees = 2000,
    interaction.depth = hyper_grid$interaction.depth[i],
    shrinkage = hyper_grid$shrinkage[i],
    n.minobsinnode = hyper_grid$n.minobsinnode[i],
    bag.fraction = hyper_grid$bag.fraction[i],
    train.fraction = 0.75,
    n.cores = NULL,
    verbose = FALSE
  )

  # add min training error and trees to grid
  hyper_grid$optimal_trees[i] <- which.min(gbm.tune$valid.error)
  hyper_grid$min_MSE[i] <- min(gbm.tune$valid.error)
}

# Display top 10 results
hyper_grid %>% arrange(min_MSE) %>% head(10)
```

```
##    shrinkage interaction.depth n.minobsinnode bag.fraction optimal_trees
## 1       0.10                 9              1          0.8          1879
## 2       0.10                 9              1          0.9          2000
## 3       0.10                11              1          1.0           231
## 4       0.10                13              1          0.9          1995
## 5       0.05                13              1          0.8          1996
```

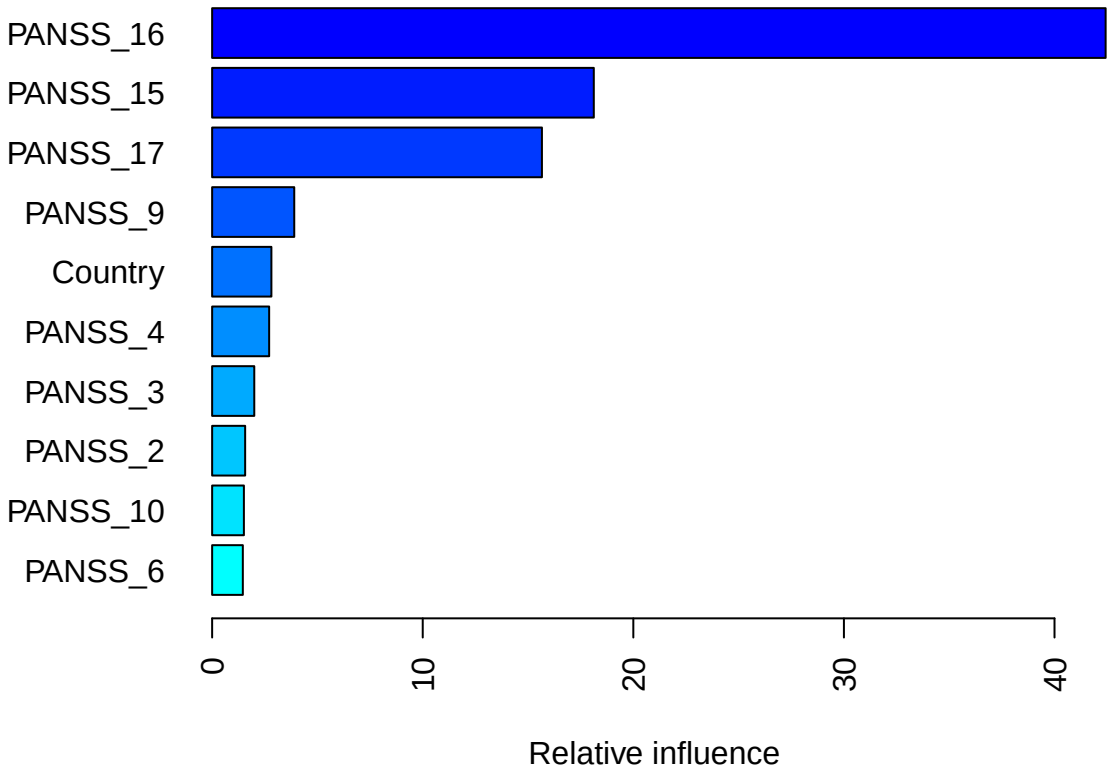
```
## 6       0.05                 9             1          0.9          1999
## 7       0.05                13             1          0.9          1995
## 8       0.10                15             1          0.9          1995
## 9       0.10                13             1          0.8          1983
## 10      0.10                13             1          1.0           419
##     min_MSE
## 1  37.90164
## 2  38.55142
## 3  38.96685
## 4  39.03589
## 5  39.09879
## 6  39.10192
## 7  39.62614
## 8  39.69176
## 9  39.80136
## 10 39.80220
```

Using the optimal parameters, we train our model.

```
# Train GBM model using optimal parameters
set.seed(123)
gbm.fit.final <- gbm(
  formula = PANSS_18 ~ .,
  distribution = "gaussian",
  data = train,
  n.trees =136,
  interaction.depth = 11,
  shrinkage = 0.1,
  n.minobsinnode = 1,
  bag.fraction = .8,
  train.fraction = 1,
  n.cores = NULL,
  verbose = FALSE
)
```

We can also visualize the relative importance of each parameter.

```
# Plot relative importance
par(mar = c(5, 8, 1, 1))
summary(
  gbm.fit.final,
  cBars = 10,
  method = relative.influence,
  las = 2
)
```




```
##                var    rel.inf
## PANSS_16 PANSS_16 42.4269509
## PANSS_15 PANSS_15 18.1230079
## PANSS_17 PANSS_17 15.6603761
## PANSS_9   PANSS_9  3.8992178
## Country   Country  2.8136464
## PANSS_4   PANSS_4  2.7069589
## PANSS_3   PANSS_3  2.0005403
## PANSS_2   PANSS_2  1.5682473
## PANSS_10 PANSS_10  1.5087493
## PANSS_6   PANSS_6  1.4569684
## PANSS_12 PANSS_12  1.2462186
## PANSS_11 PANSS_11  1.0479561
## PANSS_14 PANSS_14  0.9531162
## PANSS_5   PANSS_5  0.9135924
## PANSS_8   PANSS_8  0.8934001
## PANSS_0   PANSS_0  0.8256093
## PANSS_13 PANSS_13  0.6689297
## PANSS_7   PANSS_7  0.6518555
## PANSS_1   PANSS_1  0.6346586
```

Finally we can predict the values for the test data and see the test MSE.

```
# predict values for test data
pred <- predict(gbm.fit.final, n.trees = gbm.fit.final$n.trees, test)

# Test MSE
mse_gbm2 = mean((test$PANSS_18 - pred)^2)
mse_gbm2
```

```
## [1] 28.94244
```

XGBoost Model

Our next approach is using the XGBoost Model. Before training the model however, we have to first convert our training dataset into the correct format.

```
# variable names
features <- setdiff(names(train), "PANSS_18")

# Create the treatment plan from the training data
treatplan <- designTreatmentsZ(train, features, verbose = FALSE)

# Get the "clean" variable names from the scoreFrame
new_vars <- treatplan %>% use_series(scoreFrame) %>%
  filter(code %in% c("clean", "lev")) %>% use_series(varName)

# Prepare the training data
features_train <- prepare(treatplan, train, varRestriction = new_vars) %>% as.matrix()
response_train <- train$PANSS_18

# Prepare the test data
features_test <- prepare(treatplan, test, varRestriction = new_vars) %>% as.matrix()
response_test <- test$PANSS_18
```

Next, we create the hyperparameter grid.

```
# create hyperparameter grid
hyper_grid <- expand.grid(
  eta = c(.05, .1),
  max_depth = c(1, 3, 5),
  min_child_weight = c(1, 3, 5),
  subsample = c(.65, .9, 1),
  colsample_bytree = c(.7, .8, 0.9),
  optimal_trees = 0,
  min_RMSE = 0
)
```

Then we train and display the best model

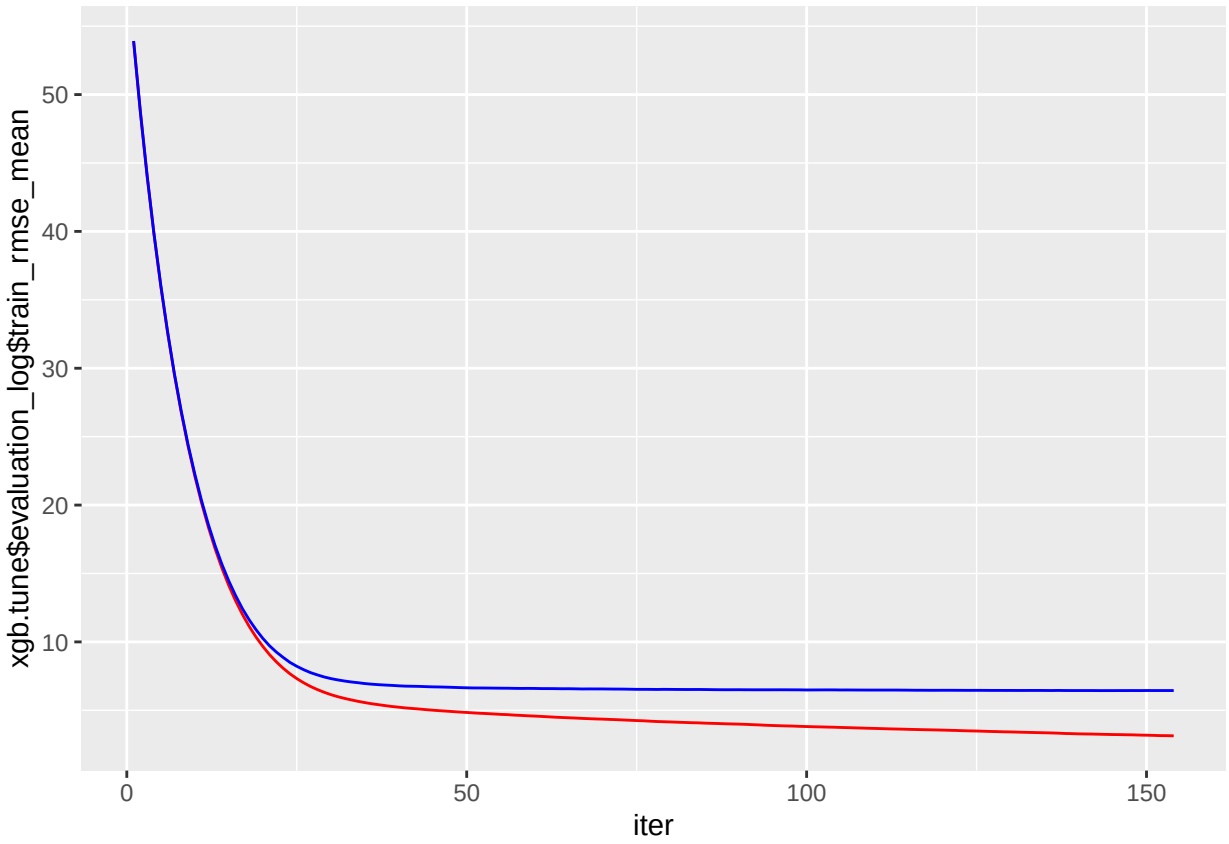
```
# grid search
for(i in 1:nrow(hyper_grid)) {

  # create parameter list
  params <- list(
    eta = hyper_grid$eta[i],
    max_depth = hyper_grid$max_depth[i],
    min_child_weight = hyper_grid$min_child_weight[i],
    subsample = hyper_grid$subsample[i],
    colsample_bytree = hyper_grid$colsample_bytree[i]
  )

  # reproducibility
  set.seed(123)

  # train model
```

```
  xgb.tune <- xgb.cv(
    params = params,
    data = features_train,
    label = response_train,
    nrounds = 1000,
    nfold = 5,
    objective = "reg:linear",
    verbose = 0,
    early_stopping_rounds = 10
  )

  # add min training error and trees to grid
  hyper_grid$optimal_trees[i] <- which.min(xgb.tune$evaluation_log$test_rmse_mean)
  hyper_grid$min_RMSE[i] <- min(xgb.tune$evaluation_log$test_rmse_mean)
}

# Display 10 best models
hyper_grid %>% dplyr::arrange(min_RMSE) %>% head(10)
```

```
##     eta max_depth min_child_weight subsample colsample_bytree
## 1  0.05         5                1      0.90              0.7
## 2  0.10         5                1      0.90              0.7
## 3  0.10         5                1      0.90              0.8
## 4  0.05         5                1      0.65              0.8
## 5  0.10         5                1      1.00              0.9
## 6  0.05         5                1      1.00              0.8
## 7  0.05         5                1      0.90              0.8
## 8  0.10         5                1      1.00              0.8
## 9  0.05         5                1      0.90              0.9
## 10 0.10         5                1      1.00              0.7
##    optimal_trees min_RMSE
## 1            354 6.090661
## 2            178 6.118413
## 3            158 6.119801
## 4            335 6.122779
## 5            191 6.149202
## 6            306 6.165481
## 7            384 6.166369
## 8            157 6.177280
## 9            296 6.179886
## 10           167 6.194122
```

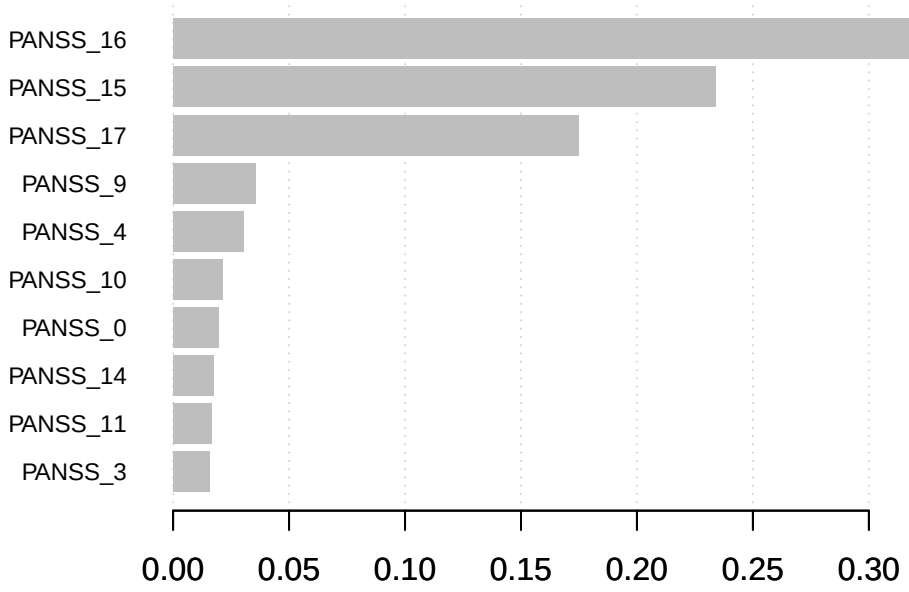
We can plot our train and test error for the best model.

```
# plot error
ggplot(xgb.tune$evaluation_log) +
  geom_line(aes(iter, xgb.tune$evaluation_log$train_rmse_mean), color = "red") +
  geom_line(aes(iter, xgb.tune$evaluation_log$test_rmse_mean), color = "blue")
```



Using our best model, we train our final model.

```
set.seed(123)

# parameter list
params <- list(
  eta = 0.05,
  max_depth = 5,
  min_child_weight = 1,
  subsample = 1,
  colsample_bytree = 0.7
)

# train final model
xgb.fit.final <- xgboost(
  params = params,
  data = features_train,
  label = response_train,
  nrounds = 376,
  objective = "reg:linear",
  verbose = 0
)

# create importance matrix
importance_matrix <- xgb.importance(model = xgb.fit.final)

# variable importance plot
xgb.plot.importance(importance_matrix, top_n = 10)
```



Finally, we predict on the test data and calculate the test MSE.

```
# predict values for test data
pred <- predict(xgb.fit.final, features_test)

# results
mse_xgb = mean((test$PANSS_18 - pred)^2)
mse_xgb
```

```
## [1] 35.07826
```

Summary of models

Below is the summary of the test MSE for all the models.

```
summary <- t(data.frame("Linear Regression" = mse_lm, "LASSO" = mse_lasso,
                        "Random Forest" = mse_rf, "Gradient Boosting" = mse_gbm2,
                        "XGBoost" = mse_xgb))
colnames(summary) <- "Test MSE"
summary
```

```
##                   Test MSE
## Linear.Regression 53.03279
## LASSO             52.99754
## Random.Forest     36.90424
## Gradient.Boosting 28.94244
## XGBoost           35.07826
```

Predict on Study E

Before predicting on Study E, we first have to preprocess the data similarly as our training set.

```
# Load study E
study_e = read.csv("Study_E.csv")
data_e = read.csv("sample_submission_PANSS.csv")
```

```r
# Get Patient IDs
patient_id <- data_e[,1]
data_obs <- subset(filter(study_e, study_e$PatientID %in% patient_id))

# Removing the duplicates
cols <- 9:39
processed_studies_e <- setDT(data_obs)[, lapply(.SD,mean),
                                       by=c("Country", "PatientID", "VisitDay", "TxGroup"),
                                       .SDcols = cols]

# Taking ceiling of Ps, Ns, and Gs.
processed_studies_e <- processed_studies_e %>% mutate_at(5:34, ceiling)

# Calculating PANSS_Total with new Ps, Ns, and Gs.
processed_studies_e$PANSS_Total <- rowSums(processed_studies_e[5:34])

#Combining Ns, Ps, and Gs scores. Making separate columns for them.
processed_studies_e$P_total <- rowSums(processed_studies_e[5:11])
processed_studies_e$N_total <- rowSums(processed_studies_e[12:18])
processed_studies_e$G_total <- rowSums(processed_studies_e[19:34])

# Only take Country, PatientID, visitDay and PANSS_Total
weekly_studies_e <- subset(processed_studies_e, select = c(1,2,3,35))

# Populate new dataframe with VisitDay = 0
visitday_0 <- subset(weekly_studies_e, VisitDay==0)
IDs <- visitday_0["PatientID"][,1]
weekly_PANSS_e <- data.frame(PatientID = IDs)
merge_weekly_e <- merge(weekly_PANSS_e, visitday_0 , by="PatientID", all= TRUE)
names(merge_weekly_e)[ncol(merge_weekly_e)] <- "PANSS_0"

# Populate dataframe for each week
for (i in 1:18){
    week <- subset(processed_studies_e, VisitDay<=i*7 & VisitDay > i*7-7)

  merge_weekly_e <- merge(merge_weekly_e, week[,c("PatientID", "PANSS_Total")],
                          by="PatientID", all= TRUE)
  col=ncol(merge_weekly_e)
  names(merge_weekly_e)[col] <- paste("PANSS_",i, sep = "")
}

#Get index of NA rows for Week 18 and predict for that only
index <- which(is.na(merge_weekly_e$PANSS_18)) #303 patients to predict

# Test Set
# test_e <- prepare(treatplan, merge_weekly_e[index,],
#                   varRestriction = new_vars) %>% as.matrix()
test_e <- merge_weekly_e[index,]
```

Finally, we predict.

```r
# Final submission: predict values for test data
# pred_val <- predict(xgb.fit.final, test_e)
pred_val <- predict(gbm.fit.final, n.trees = gbm.fit.final$n.trees, test_e)
```

```r
# Submission File
data_e[-index,2] <- merge_weekly_e$PANSS_18[-index]
data_e[index,2] <- pred_val
write.csv(data_e, "PANSS_forecasting_7.csv", row.names = FALSE)
```

Appendix 4: Objective 4

Kritika Dusad & Irene Alisjahbana

August 14, 2019

This appendix contains the codes that are used for Objective 4 in order to produce the outputs seen in the final report.

Data Pre-Processing

Set the working directory and load the necessary libraries.

```
#Set working directory
path1 <- "~/Google Drive/Stanford Files/Q7 Summer 2019/"
path2 <- "STATS 202 - Data Mining and Analysis/Final Project"
setwd(paste0(path1,path2))

# Load libraries
library(data.table)
library(tree)
library(Matrix)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following objects are masked from 'package:data.table':
##
##     dcast, melt
```

```
library(tree)
library(vtreat)
library(xgboost)
library(tidyverse)
```

```
## Registered S3 method overwritten by 'cli':
##   method     from
##   print.tree tree
```

```
## -- Attaching packages ---------------------------------------------------------- tidyverse 1.2.1 --
```

```
## v tibble  2.1.3     v purrr   0.3.2
## v tidyr   0.8.3     v dplyr   0.8.3
## v readr   1.3.1     v stringr 1.4.0
## v tibble  2.1.3     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------ tidyverse_conflicts() --
## x dplyr::between()   masks data.table::between()
## x tidyr::expand()    masks Matrix::expand()
## x dplyr::filter()    masks stats::filter()
## x dplyr::first()     masks data.table::first()
```

```
## x dplyr::lag()       masks stats::lag()
## x dplyr::last()      masks data.table::last()
## x purrr::lift()      masks caret::lift()
## x dplyr::slice()     masks xgboost::slice()
## x purrr::transpose() masks data.table::transpose()
```

```
library(caret)
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'
```

```
## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
```

```
## The following object is masked from 'package:base':
##
##     Recall
```

```
library(mlr)
```

```
## Loading required package: ParamHelpers
```

```
##
## Attaching package: 'mlr'
```

```
## The following object is masked from 'package:caret':
##
##     train
```

```
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
library(dplyr)
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(magrittr)
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

Next we will load the patients data. We will only use Study A-D to train and test our model.

```
# Load studies
Study_A = read.csv("Study_A.csv")
Study_B = read.csv("Study_B.csv")
Study_C = read.csv("Study_C.csv")
Study_D = read.csv("Study_D.csv")
```

Then, we combine all the studies into one dataset.

```
# Combine all studies into one dataset
all_studies <- rbind(Study_A, Study_B, Study_C, Study_D)
```

While looking through the data, we realized that there are observations with Country as ERROR. We want to remove these before continuing.

```
# Removed observations with Country as ERROR- 18 observations removed.
all_studies <- subset(all_studies, Country != "ERROR")
```

Then, we convert the categorical value LeadStatus into numerical data. Because in this objective, we are trying to predict if the observation will not pass, we consider Pass = 0, Flagged or Assigned to CS = 1. For tree and RandomForest, we want to rename the two classes into "Error" and "Passed"

```
# For tree, and RandomForest
Assessment <- ifelse(all_studies$LeadStatus
                                != "Passed", "Error", "Passed")
all_studies_tree <- data.frame(all_studies, Assessment)

# For GBM: Converting categorical LeadStatus to numerical data
all_studies$LeadStatus <- ifelse(all_studies$LeadStatus == "Passed", 0, 1)
```

Then we can drop all other columnst that are not needed in the analysis.

```
# For GBM and XGBoost
drop <- c("Study","SiteID","AssessmentiD", "PatientID",  "RaterID", "TxGroup")
studies_subset <- all_studies[,!(names(all_studies) %in% drop)]

# For tree and RandomForest
drop <- c("Study","SiteID","AssessmentiD", "PatientID",  "RaterID", "TxGroup", "LeadStatus")
studies_subset_tree <- all_studies_tree[,!(names(all_studies_tree) %in% drop)]
```

We then split our dataset into training and testing sets. Our split is 70:30 for training and testing.

```
# Train and Test set 70%:30%
set.seed(2)
smpl_size <- nrow(studies_subset)*0.7
train <- sample(1:nrow(studies_subset), smpl_size)
test <- studies_subset[-train, ]
leadstatus_test <- studies_subset[-train, "LeadStatus"]

# For tree and RandomForest:
test_tree <- studies_subset_tree[-train, ]
leadstatus_test_tree <- studies_subset_tree[-train, "Assessment"]
```

Classification Tree

Our first approach uses a single tree where the first step is to fit our model to the tree.

```
# Tree model on training set
tree_leadstatus <- tree(Assessment ~ ., studies_subset_tree, subset = train)
summary(tree_leadstatus)
```

```
##
## Classification tree:
## tree(formula = Assessment ~ ., data = studies_subset_tree, subset = train)
## Variables actually used in tree construction:
## [1] "Country"     "G10"         "PANSS_Total" "VisitDay"
## Number of terminal nodes:  5
## Residual mean deviance:  0.9347 = 13690 / 14640
## Misclassification error rate: 0.2172 = 3182 / 14650
```

After fitting, we predict on the test set and obtain the error rate.

```
# Tree model on test set
tree_pred <- predict(tree_leadstatus, test_tree, type="class")
table(tree_pred, leadstatus_test_tree)
```

```
##          leadstatus_test_tree
## tree_pred Error Passed
##    Error    563    399
##    Passed   953   4364
```

The classification accuracy can be obtained below:

```
# Accuracy
mean(tree_pred == leadstatus_test_tree)*100
```

```
## [1] 78.46791
```

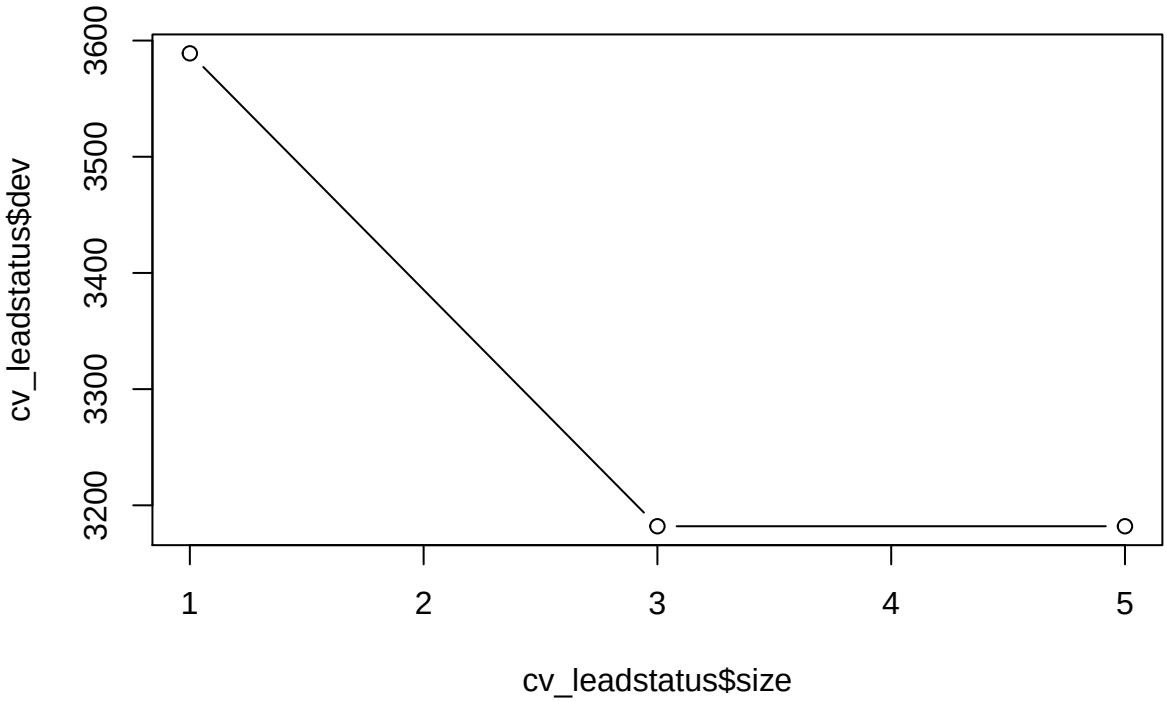
To prevent overfitting, we try to use cross validation and pruning.

```
# Cross-validation
cv_leadstatus <- cv.tree(tree_leadstatus, FUN = prune.misclass)
cv_leadstatus
```

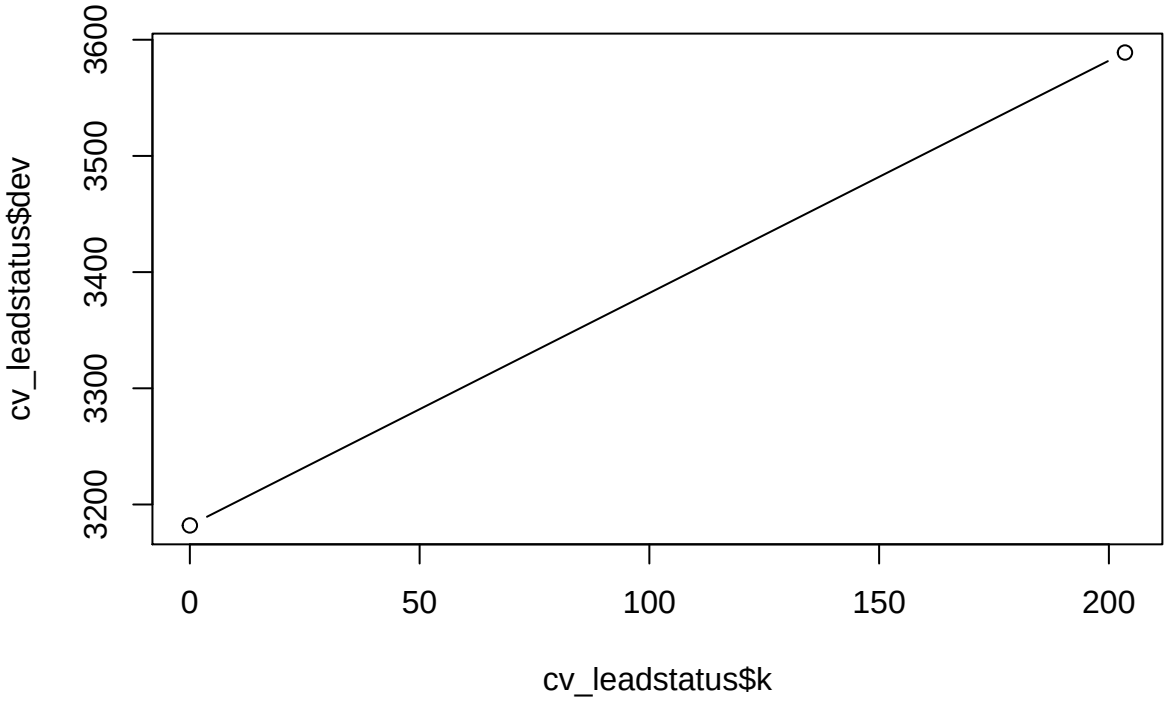
```
## $size
## [1] 5 3 1
##
## $dev
## [1] 3182 3182 3589
##
## $k
## [1]  -Inf   0.0 203.5
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

We can visualize the error rate as a function of size and k.

```
plot(cv_leadstatus$size, cv_leadstatus$dev, type = "b")
```

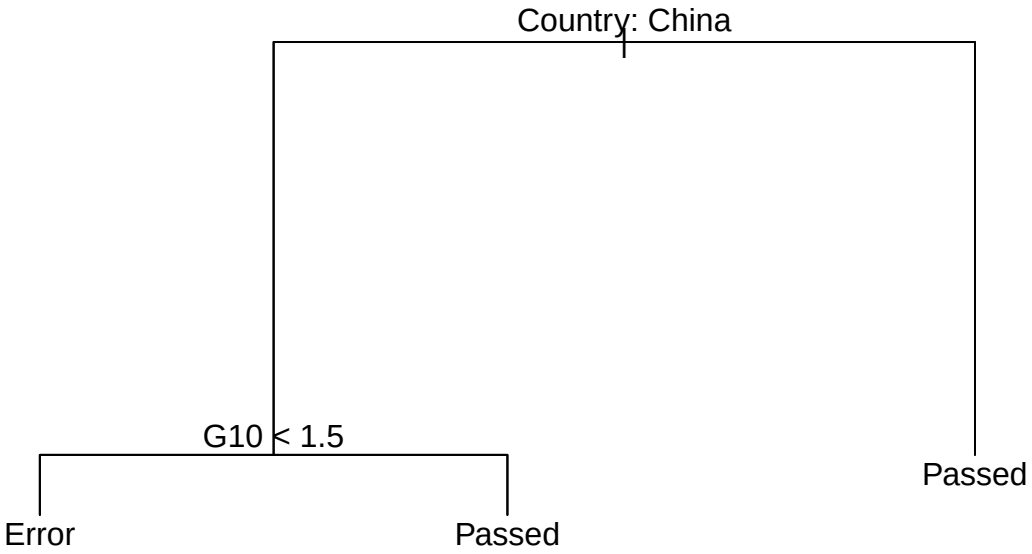



```
plot(cv_leadstatus$k, cv_leadstatus$dev, type = "b")
```



From the cross validation, it can be seen that the best complexity is 3. So we use that and visualize the tree.

```
# Prune tree
prune_leadstatus <- prune.misclass(tree_leadstatus, best=3)

# Visualize tree
plot(prune_leadstatus)
text(prune_leadstatus, pretty=0)
```



Finally we predict on the test set.

```
yhat_prune <- predict(prune_leadstatus, newdata=test_tree, type = "class")
table(yhat_prune, leadstatus_test_tree)
```

```
##           leadstatus_test_tree
## yhat_prune Error Passed
##     Error    563    399
##     Passed   953   4364
```

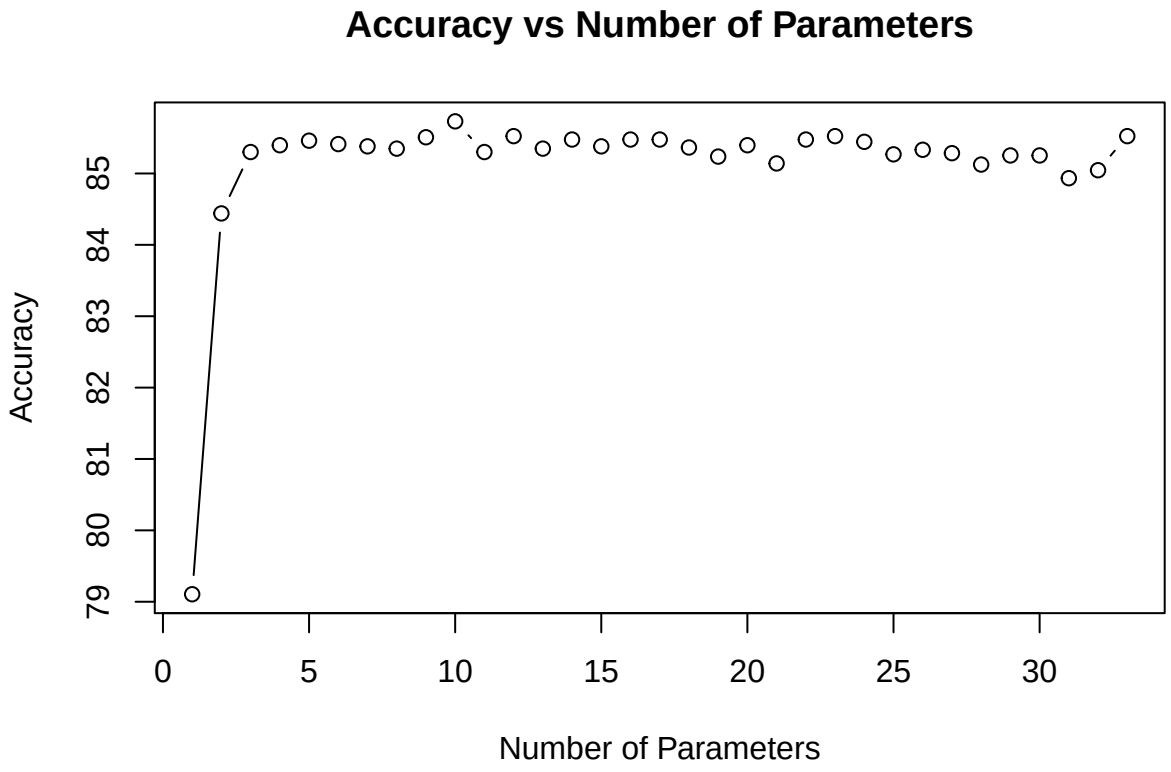
```
acc_tree <- mean(yhat_prune == leadstatus_test_tree)*100
acc_tree
```

```
## [1] 78.46791
```

Bagging and RandomForest

Here we determine the optimal number of parameters to use in our model and visualize the accuracy. Bagging is just a special case of RandomForest where m = p.

```
set.seed(123) # For reproducibility
num_param = ncol(studies_subset)-1
acc=c()

for(i in 1:num_param){
  rf_leadstatus = randomForest(Assessment~.,data = studies_subset_tree,subset = train,
                               mtry=i, importance=TRUE, ntree=100)

  leadstatus_pred = predict(rf_leadstatus,test_tree)
  acc[i] = mean(leadstatus_pred == leadstatus_test_tree)*100

}

# Visualize accuracy
plot(1:num_param,acc,type='b', main = "Accuracy vs Number of Parameters",
     xlab = "Number of Parameters", ylab = "Accuracy")
```

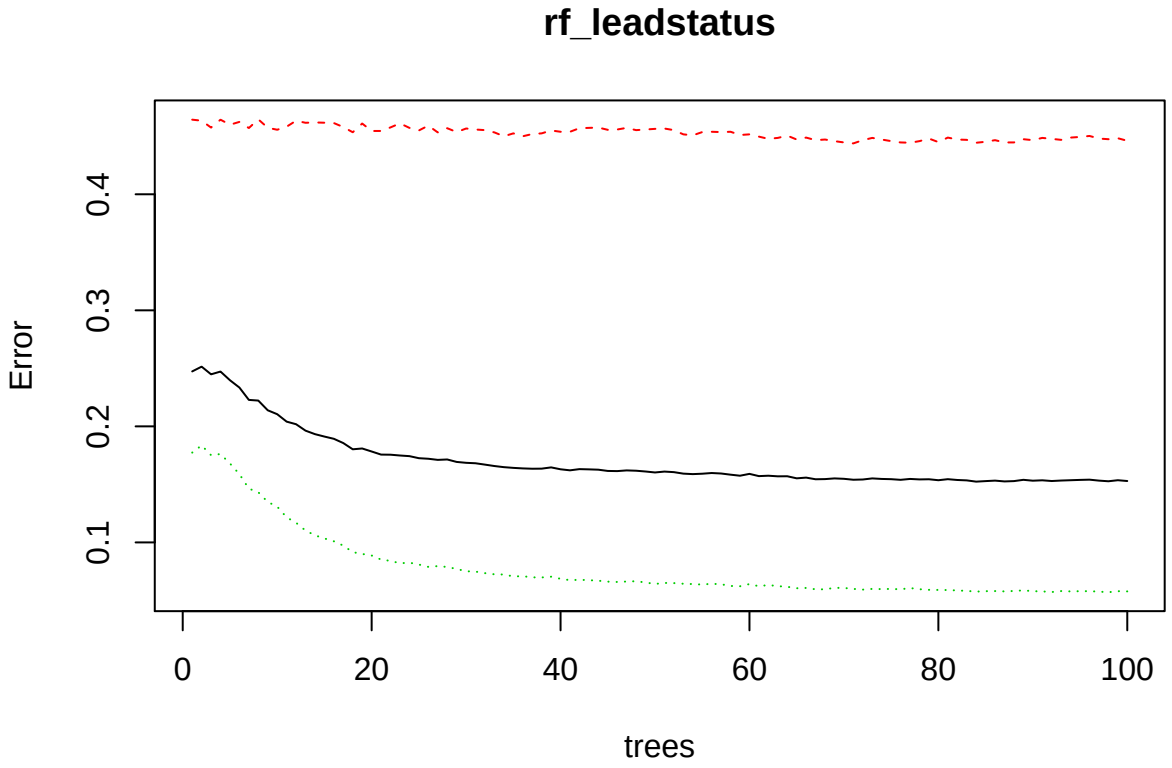


Based on our plot, the optimal number of parameters that result in the highest accuracy is chosen for our model.

```
# Use the best number of parameters
best_num <- which.max(acc)

rf_leadstatus <- randomForest(Assessment~., data = studies_subset_tree, subset=train,
                              mtry=best_num, importance=TRUE, ntree=100)

# View Summary
rf_leadstatus
```

```
##
## Call:
##  randomForest(formula = Assessment ~ ., data = studies_subset_tree,      mtry = best_num, importance
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 10
##
##         OOB estimate of  error rate: 15.29%
## Confusion matrix:
##        Error Passed class.error
## Error   1988   1601  0.44608526
## Passed   639  10422  0.05777055
```

```
plot(rf_leadstatus)
```



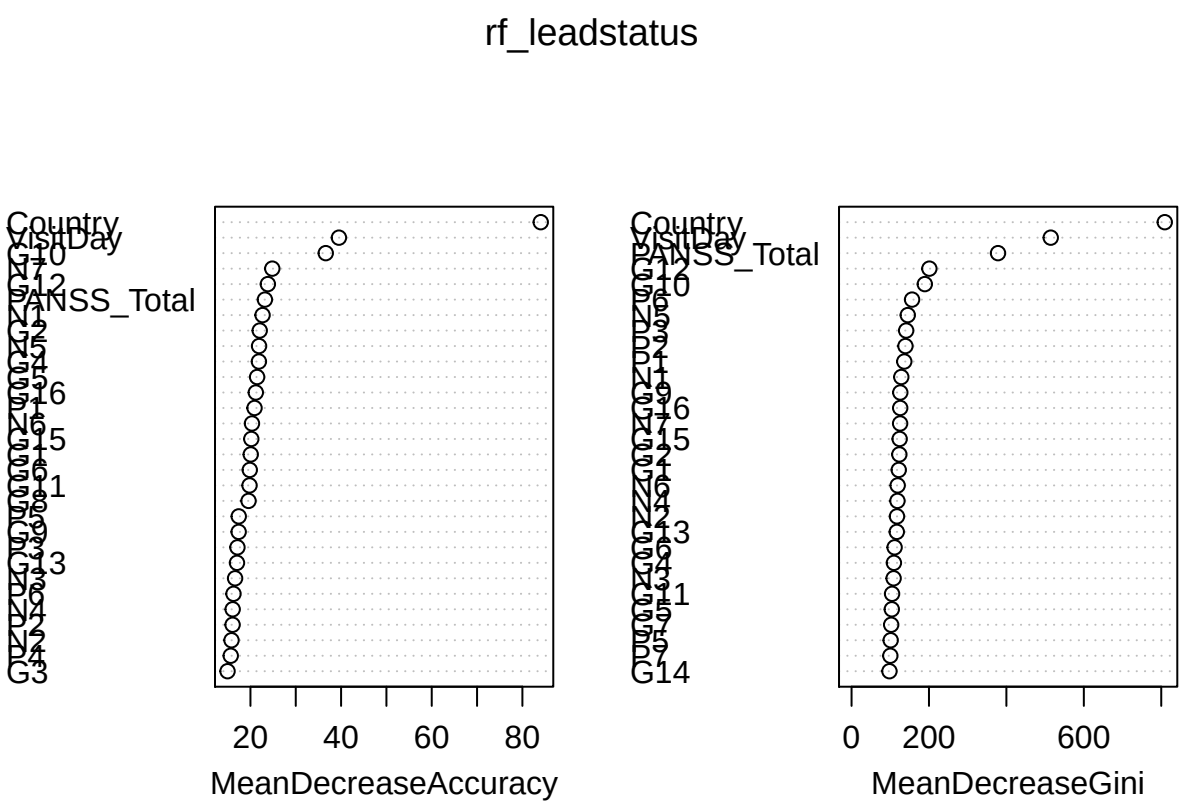
We can then view the importance of each variable and plot it.

```
# Using importance function:
importance(rf_leadstatus)
```

```
##                  Error    Passed MeanDecreaseAccuracy MeanDecreaseGini
## Country     102.954593 47.563907             84.05183        809.08211
## VisitDay     20.159962 32.573180             39.52593        514.54551
## P1            4.376233 19.996806             20.90082        136.17804
## P2           12.362337 10.915763             16.06042        139.21436
## P3            9.904023 13.900311             17.12986        141.08271
## P4            8.195376 12.231332             15.65013         95.87821
## P5           15.152057  9.365316             17.42034        101.09703
## P6           13.464851 12.230269             16.26070        156.32691
## P7           10.841314 11.567650             13.66745        100.15966
## N1           12.176020 15.747292             22.67382        128.47311
## N2           11.693248 12.786120             15.82543        117.34592
## N3           10.201492 14.647140             16.61136        108.62944
## N4           11.491584 12.246592             16.07628        118.56535
## N5           11.569368 17.468228             21.90003        145.20658
## N6            7.559584 15.356904             20.33805        119.11416
## N7           10.620416 21.800663             24.81770        125.47396
## G1           13.512007 14.946312             20.06786        121.94953
## G2            3.083372 19.671361             22.04826        123.48546
## G3           10.304156  9.685615             14.94473         87.68850
## G4            3.876683 21.159243             21.86830        109.39783
## G5           13.714210 12.934183             21.47433        103.99509
## G6           16.125257 11.416712             19.85642        111.23930
## G7            9.470934 11.739032             14.21634        102.35515
## G8           10.958389 15.150142             19.58167         95.71887
```

```
## G9           13.025453  9.276952            17.40155        125.92544
## G10          27.601799 26.852247            36.61638        189.24670
## G11          10.811873 14.568678            19.80033        104.67357
## G12          18.580409 19.017516            23.84305        201.02534
## G13           9.853455 15.477151            17.03112        116.86636
## G14          12.847352  8.214167            14.32589         97.88171
## G15          11.374730 16.654102            20.18916        124.26948
## G16          12.130228 16.452091            21.18480        125.67759
## PANSS_Total   8.870205 19.745509            23.18883        378.36295
```

```
varImpPlot(rf_leadstatus)
```



Finally, we can use our final model to predict on the test set to obtain the accuracy.

```
# Using bagged model to predict LeadStatus on the test set
yhat_rf <- predict(rf_leadstatus, newdata=test_tree)
table(yhat_rf, leadstatus_test_tree)
```

```
##         leadstatus_test_tree
## yhat_rf  Error Passed
##   Error    826    236
##   Passed   690   4527
```

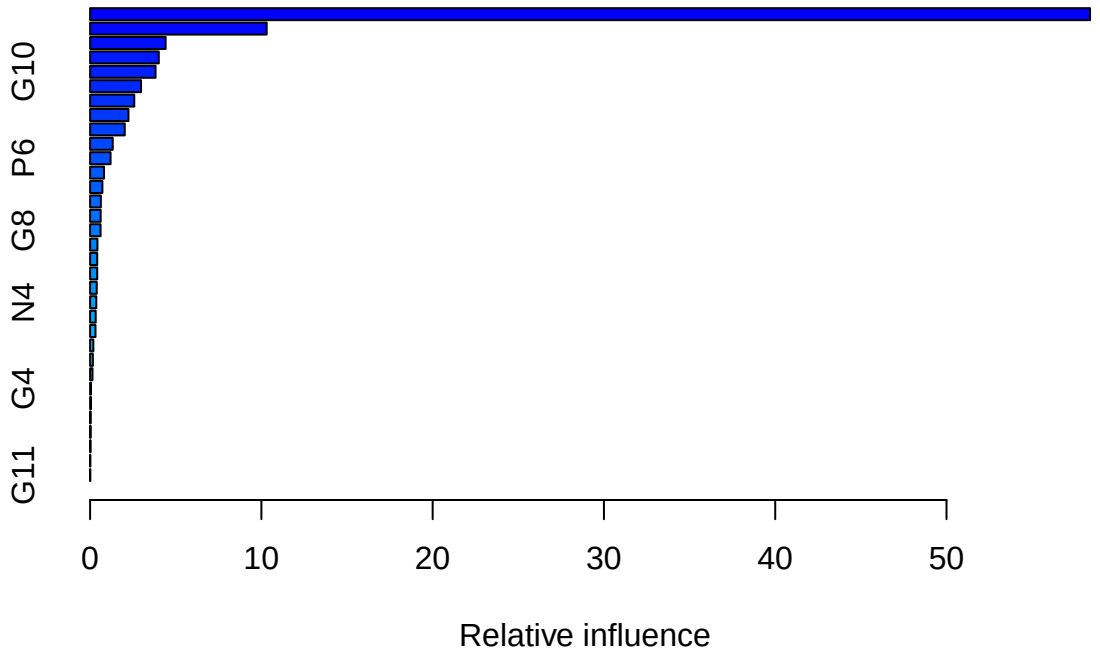
```
acc_rf <- mean(yhat_rf == leadstatus_test_tree)*100
acc_rf
```

```
## [1] 85.25243
```

Gradient Boosting

We use the gradient boosting method in order to create our model. We first train our model using our training set.

```
# Boosting on training set
boost_leadstatus <- gbm(LeadStatus~., data=studies_subset[train,],
                        distribution="bernoulli", n.trees=2000,
                        shrinkage=0.01)
summary(boost_leadstatus)
```




```
##                     var      rel.inf
## Country         Country 58.381637707
## VisitDay       VisitDay 10.309066656
## PANSS_Total PANSS_Total  4.408982135
## P2                   P2  4.009316160
## G10                 G10  3.821415161
## G12                 G12  2.973048542
## P3                   P3  2.578598603
## P4                   P4  2.239569350
## N1                   N1  2.022786426
## G16                 G16  1.322642129
## P6                   P6  1.193629616
## N3                   N3  0.810751159
## G15                 G15  0.717952656
## N5                   N5  0.627568166
## P7                   P7  0.616181922
## G8                   G8  0.612165245
## P1                   P1  0.427556034
## G9                   G9  0.416126576
## G14                 G14  0.416076748
## N7                   N7  0.385783240
## N4                   N4  0.360613529
## G13                 G13  0.327378542
```

```
## N2                   N2  0.313964633
## N6                   N6  0.188269337
## G3                   G3  0.160614188
## G5                   G5  0.140866901
## G4                   G4  0.053297921
## P5                   P5  0.053251950
## G1                   G1  0.032702625
## G6                   G6  0.032430201
## G2                   G2  0.026048413
## G7                   G7  0.011441492
## G11                 G11  0.008266034
```

Then we can test our model with our test data. Because the output of the predictions are in the form of probabilities, we need to convert them to the binary class of 1 (flagged or assigned to CS) and 0 (passed). Our threshold is 0.5.

```
# Boosting model on test data"
yhat_leadstatus <- predict(boost_leadstatus,test, n.trees=2000, type='response')
yhat_leadstatus <- ifelse(yhat_leadstatus > 0.5, 1, 0)

table(yhat_leadstatus == leadstatus_test)
```

```
##
## FALSE  TRUE
##  1298  4981
```

We can calculate the rate of true positive in our model:

```
acc_gbm <- mean(yhat_leadstatus==leadstatus_test)*100
acc_gbm
```

```
## [1] 79.32792
```

XGBoost

```
# variable names
train <- studies_subset[train,]
features <- setdiff(names(train), "LeadStatus")

# Create the treatment plan from the training data
treatplan <- designTreatmentsZ(train,  features, verbose = FALSE)

# Get the "clean" variable names from the scoreFrame
new_vars <- treatplan %>% use_series(scoreFrame) %>%
  filter(code %in% c("clean", "lev")) %>% use_series(varName)

# Prepare the training data
train_data <- prepare(treatplan, train, varRestriction = new_vars) %>% as.matrix()
train_labels <- train$LeadStatus

# Prepare the test data
test_data <- prepare(treatplan, test, varRestriction = new_vars) %>% as.matrix()
test_labels <- test$LeadStatus

#Converting dataframe to dmatrix
dtrain <- xgb.DMatrix(data=train_data, label = train_labels)
```

```
dtest <- xgb.DMatrix(data=test_data, label = test_labels)
```

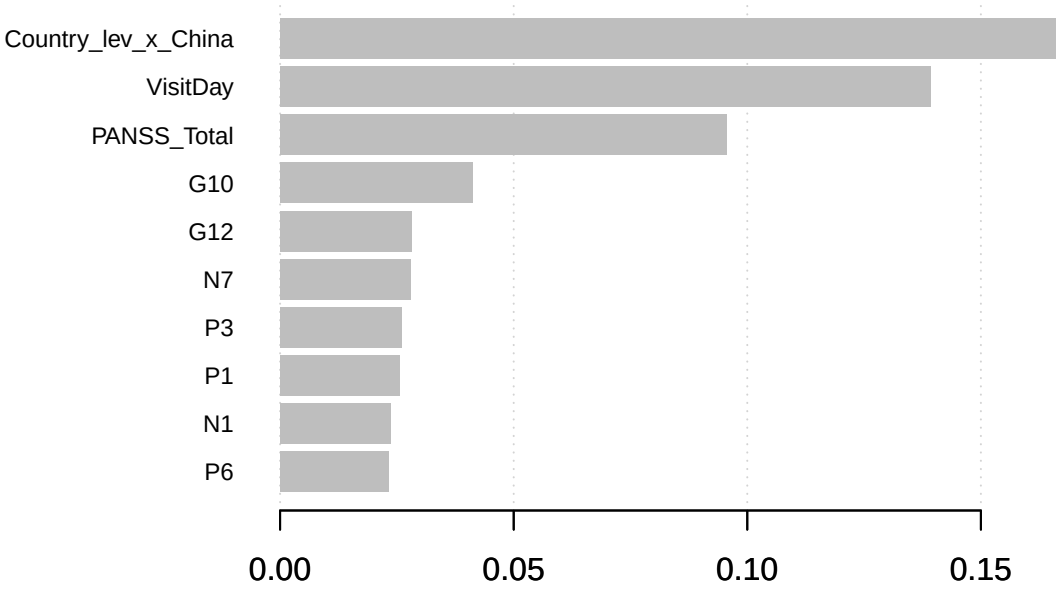
```
params <- list(booster = "gbtree", objective = "binary:logistic",
               eta=0.3, gamma=0, max_depth=6, min_child_weight=1,
               subsample=1, colsample_bytree=1)

# Determine nrounds
xgbcv <- xgb.cv( params = params, data = train_data, label= train_labels, nrounds = 200, nfold = 5,
                 showsd = T, stratified = T, print_every_n = 10,
                 early_stopping_rounds = 20, maximize = F)
```

```
## [1]  train-error:0.185017+0.004127   test-error:0.203073+0.009577
## Multiple eval metrics are present. Will use test_error for early stopping.
## Will train until test_error hasn't improved in 20 rounds.
##
## [11] train-error:0.144864+0.004409   test-error:0.177135+0.008755
## [21] train-error:0.130410+0.004402   test-error:0.173995+0.009111
## [31] train-error:0.119044+0.005355   test-error:0.171128+0.008467
## [41] train-error:0.106877+0.004318   test-error:0.170377+0.009642
## [51] train-error:0.096280+0.003269   test-error:0.169080+0.007197
## [61] train-error:0.088157+0.004831   test-error:0.166213+0.007487
## [71] train-error:0.080563+0.003530   test-error:0.166623+0.007613
## [81] train-error:0.073891+0.004741   test-error:0.166418+0.007518
## [91] train-error:0.067457+0.004706   test-error:0.167032+0.006453
## Stopping. Best iteration:
## [76] train-error:0.076911+0.004379   test-error:0.164984+0.007520
```

```
set.seed(123)

# train final model
xgb.fit.final <- xgboost (params = params, data = train_data, label = train_labels, nrounds = 112,
                  print_every_n = 10, early_stopping_round = 10,
                  maximize = F , eval_metric = "logloss")
```

```
## [1]  train-logloss:0.574571
## Will train until train_logloss hasn't improved in 10 rounds.
##
## [11] train-logloss:0.368644
## [21] train-logloss:0.342568
## [31] train-logloss:0.326421
## [41] train-logloss:0.301511
## [51] train-logloss:0.286267
## [61] train-logloss:0.268156
## [71] train-logloss:0.252736
## [81] train-logloss:0.238631
## [91] train-logloss:0.230225
## [101]    train-logloss:0.219814
## [111]    train-logloss:0.209822
## [112]    train-logloss:0.208806
```

```
# create importance matrix
importance_matrix <- xgb.importance(model = xgb.fit.final)

# variable importance plot
xgb.plot.importance(importance_matrix, top_n = 10)
```




```
# predict values for test data
xgb_pred <- predict(xgb.fit.final, test_data)
xgb_pred <- ifelse (xgb_pred > 0.5, 1, 0)

# results
table(xgb_pred, test_labels)
```

```
##         test_labels
## xgb_pred    0    1
##        0 4483  728
##        1  280  788
```

```
acc_xgb <- mean(xgb_pred==test_labels)*100
acc_xgb
```

```
## [1] 83.94649
```

Summary of Models

Below is the summary of the accuracy for all the models.

```
summary <- t(data.frame("Tree" = acc_tree, "Random Forest" = acc_rf,
                        "Gradient Boosting" = acc_gbm, "XGBoost" = acc_xgb))
colnames(summary) <- "Accuracy"
summary
```

```
##                   Accuracy
## Tree              78.46791
## Random.Forest     85.25243
## Gradient.Boosting 79.32792
## XGBoost           83.94649
```

Predict on Study E

Finally we can use our best model on Study E.

```
# Load Data
Study_E = read.csv("Study_E.csv")
```

```r
# Save AssessmentiDs
Study_E_AssessmentiD <- Study_E$AssessmentiD

# Drop unecessary columns
drop <- c("Study","SiteID","AssessmentiD", "PatientID",  "RaterID", "TxGroup")
E_subset <- Study_E[,!(names(Study_E) %in% drop)]

# Predict on patients using trained model
Study_E_pred <- predict(boost_leadstatus, E_subset, n.trees=2000, type='response')
LeadStatus <- as.data.frame(Study_E_pred)

# Create output for Kaggle
Study_E_predictions <- cbind(LeadStatus, Study_E_AssessmentiD)
LeadStatus_E<- Study_E_predictions[,c(2, 1)]
names(LeadStatus_E) <- c("AssessmentiD", "LeadStatus")
write.csv(LeadStatus_E, file="StudyE_boosted_pred_2.csv", row.names = FALSE)
```