'Post-Operative Patient' Perdiction

Helen Han

May 10th, 2018

Packages Required

Introduction/Summary This document is to show and describe each step of analysis taken of the "Post-Operative Patient Data Set". The goal of analysis of data set is to predict whether a patient after surgery will be sent to the hospital floor or sent their home through solid approaches and proper processes of analysis.

Data Exploration

Part one. Data Cleaning

0. Set Seed

I set the seed only this initial occasion instead of every time when I run each algorithm, which picks up the same random subset of data. In this way, I can improve fitting my model by comparing results when I run the same algorithms multiple times.

```
set.seed(3)
```

1. Import the Data set

While importing and viewing data set, the following steps were taken.

- Check out the types of values for each feature along with the levels of them.
- Examine rows and columns to check missing values and/or inappropriate values.

The data consisted of 9 features with their current type of factors. 8 of those features were label values and only one "V8" column contained numbers.

```
mydata <- read.csv("post-operative.csv", header = FALSE)
#View(mydata)
summary(mydata)
```

```
##     V1        V2             V3        V4            V5
##  high:13   high:17   excellent:43   high:30   stable  :45
##  low :19   low :25   good     :47   low : 3   unstable:45
##  mid :58   mid :48                  mid :57
##
##
##           V6              V7      V8      V9
##  mod-stable: 1   mod-stable:21   ? : 3   A :63
##  stable    :83   stable    :46   10:65   A : 1
##  unstable  : 6   unstable  :23   15:19   I : 2
##                                  5 : 2   S :24
##                                  7 : 1
```

```
str(mydata)
```

```
## 'data.frame':    90 obs. of  9 variables:
##  $ V1: Factor w/ 3 levels "high","low","mid": 3 3 1 3 3 1 3 1 3 3 ...
##  $ V2: Factor w/ 3 levels "high","low","mid": 2 1 2 2 3 2 2 3 1 2 ...
##  $ V3: Factor w/ 2 levels "excellent","good": 1 1 1 2 1 2 1 1 2 1 ...
##  $ V4: Factor w/ 3 levels "high","low","mid": 3 1 1 1 1 3 1 3 3 3 ...
##  $ V5: Factor w/ 2 levels "stable","unstable": 1 1 1 1 1 1 1 2 1 2 ...
```

```
##  $ V6: Factor w/ 3 levels "mod-stable","stable",..: 2 2 2 3 2 2 2 3 2 2 ...
##  $ V7: Factor w/ 3 levels "mod-stable","stable",..: 2 2 1 1 2 3 1 2 2 1 ...
##  $ V8: Factor w/ 5 levels "?","10","15",..: 3 2 2 3 2 3 4 2 2 2 ...
##  $ V9: Factor w/ 4 levels "A","A ","I","S": 1 4 1 2 1 4 4 4 4 4 ...
```

2. Rename Columns and Set Responsive Variable

I renamed columns to correspond to the data attributes, and set "Discharge_Decision" column as the responsive feature.

```
names(mydata) <- c("Internal_Temp", "Surface_Temp", "Oxygen_Saturation",
                   "Blood_Pressure", "Suf_Temp_Stability", "Internal_Temp_Stability",
                   "Blood_Pressure_Stability", "Comfort", "Discharge_Decision")
summary(mydata$Discharge_Decision)
```

```
##  A A   I  S
## 63  1  2 24
```

3. Clean Data

I took the following steps to clean the data set:

- Convert value "?" to "NA" and drop NA instances in "Comfort" feature.
- Overwrite class "A" with "A" in "Discharge_Decision" feature.
- Remove levels in features that do not occur.
- Convert type of values of "Comfort" feature from 'factor' to 'numeric'.

```
mydata[mydata == "?"] <- NA
mydata[mydata == "A "] <- "A"
mydata1 <- na.omit(mydata)
mydata1$Discharge_Decision <- factor(mydata1$Discharge_Decision)
mydata1$Comfort <- as.numeric(as.character(mydata1$Comfort))
str(mydata1)
```

```
## 'data.frame':    87 obs. of  9 variables:
##  $ Internal_Temp           : Factor w/ 3 levels "high","low","mid": 3 3 1 3 3 1 3 1 3 3 ...
##  $ Surface_Temp            : Factor w/ 3 levels "high","low","mid": 2 1 2 2 3 2 2 3 1 2 ...
##  $ Oxygen_Saturation       : Factor w/ 2 levels "excellent","good": 1 1 1 2 1 2 1 1 2 1 ...
##  $ Blood_Pressure          : Factor w/ 3 levels "high","low","mid": 3 1 1 1 1 3 1 3 3 3 ...
##  $ Suf_Temp_Stability      : Factor w/ 2 levels "stable","unstable": 1 1 1 1 1 1 1 2 1 2 ...
##  $ Internal_Temp_Stability : Factor w/ 3 levels "mod-stable","stable",..: 2 2 2 3 2 2 2 3 2 2 ...
##  $ Blood_Pressure_Stability: Factor w/ 3 levels "mod-stable","stable",..: 2 2 1 1 2 3 1 2 2 1 ...
##  $ Comfort                 : num  15 10 10 15 10 15 5 10 10 10 ...
##  $ Discharge_Decision      : Factor w/ 3 levels "A","I","S": 1 3 1 1 1 3 3 3 3 3 ...
##  - attr(*, "na.action")= 'omit' Named int  47 49 71
##   ..- attr(*, "names")= chr  "47" "49" "71"
```

```
summary(mydata1$Discharge_Decision)
```

```
##  A  I  S
## 62  1 24
```

All of the variables are categorical except for the ones in 'V8' column. While the column is 'quantitative' as it has order and scale, it also can be practically 'categorical' as it is the data collected from patients on their perceived comfort at discharge. One patients' rate of 15 may not the same with another patient's. In conclusion, the results of each step showed more comparable when the 'Comfort' was reserved as 'quantitative' and 'categorical'. So I converted the column as numeric.

4. Data Manipulation

This is the point that I had to make a decision on the data set. After cleaning data, I realized that there is only one instance of "I" which means the patient to be sent to 'Intensive Care Unit' from 87 observations out of three classes of responsible variable. When I simulated to compare two cases of with and without "I", given the number of instances of the other two classes, the imbalance can possibly distort the analysis. Also, there would be more technically applicable classification options such as 'Logistic Regression' if I keep two classes in the outcome feature. At this point, I estimated to drop the "I" instance or overwrite it with "A" which means the patient to be admitted to hospital. I chose to overwrite it as the class "I" can be in the category of "A" among those classes, and I wanted to secure one more observation for analysis from the relatively small data set. - Convert class "I" to "A" in responsive variable, "Discharge_Decision"

```
# Convert "I" to "A"
mydata1[mydata1 == "I"] <- "A"
mydata1$Discharge_Decision <- factor(mydata1$Discharge_Decision)
# View Cleaned Data
summary(mydata1)
```

```
##  Internal_Temp Surface_Temp Oxygen_Saturation Blood_Pressure
##  high:12       high:16      excellent:41      high:28
##  low :18       low :24      good     :46      low : 3
##  mid :57       mid :47                        mid :56
##
##
##
##  Suf_Temp_Stability Internal_Temp_Stability Blood_Pressure_Stability
##  stable  :44        mod-stable: 1           mod-stable:21
##  unstable:43        stable    :81           stable    :45
##                     unstable  : 5           unstable  :21
##
##
##
##     Comfort      Discharge_Decision
##  Min.   : 5.00   A:63
##  1st Qu.:10.00   S:24
##  Median :10.00
##  Mean   :10.94
##  3rd Qu.:10.00
##  Max.   :15.00
```

5. View Cleaned Data

```
summary(mydata1)
```

```
##  Internal_Temp Surface_Temp Oxygen_Saturation Blood_Pressure
##  high:12       high:16      excellent:41      high:28
##  low :18       low :24      good     :46      low : 3
##  mid :57       mid :47                        mid :56
##
##
##
##  Suf_Temp_Stability Internal_Temp_Stability Blood_Pressure_Stability
##  stable  :44        mod-stable: 1           mod-stable:21
##  unstable:43        stable    :81           stable    :45
##                     unstable  : 5           unstable  :21
##
```

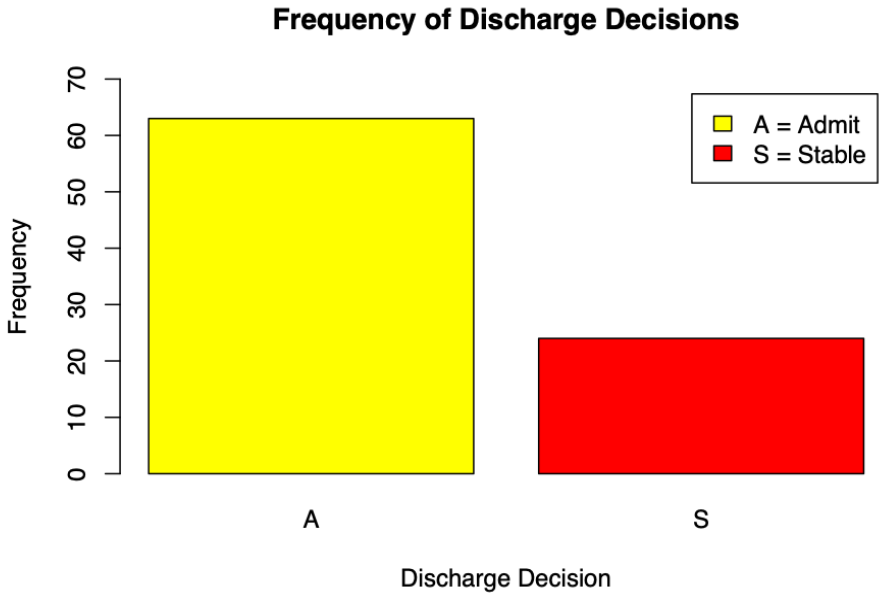
```
##
##
##     Comfort      Discharge_Decision
##  Min.   : 5.00   A:63
##  1st Qu.:10.00   S:24
##  Median :10.00
##  Mean   :10.94
##  3rd Qu.:10.00
##  Max.   :15.00
```

Part two. Plot Data

1. Bar plot

I came up with bar chart to present the current data status of outcome feature. This delivers intuitive snapshot of the observed variables in the feature. From the chart, I learned the current proportion of patients admitted to hospital after surgery and sent home in the data set.

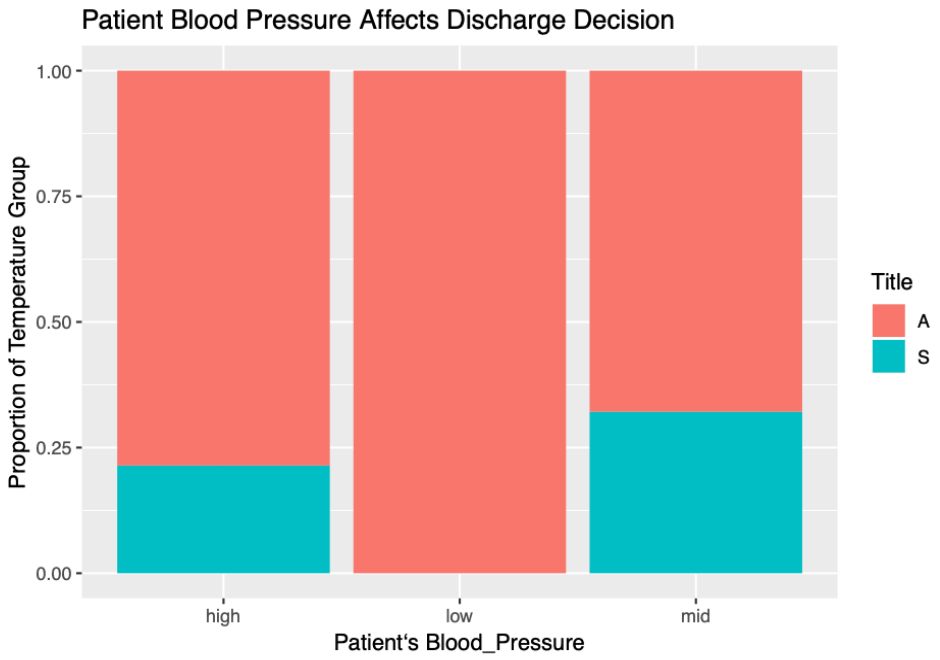
```
counts <- table(mydata1$Discharge_Decision)
barplot(counts, col = c("yellow", "red", "green"),
        main = "Frequency of Discharge Decisions",
        xlab = "Discharge Decision",
        ylab = "Frequency",
        ylim = c(0,70),
        legend.text = c("A = Admit", "S = Stable"))
```



2. Geom plot

The bar chart is good to understand how variables are spread as a category of classes in each feature. However, it doesn't show relationships between features. In this regard, geom plot is useful to understand relationships between certain features. Through the chart below, I was able to understand how patients' blood pressure level affected the discharge decision.

```
ggplot(mydata1,
       aes(x = Blood_Pressure, fill = Discharge_Decision)) +
  geom_bar(position = "fill") +
  ggtitle("Patient Blood Pressure Affects Discharge Decision") +
  xlab("Patient`s Blood_Pressure") +
  ylab("Proportion of Temperature Group") +
  scale_fill_discrete(name = "Title", labels = c("A", "S"))
```




```
#how core affects discharge decision
```

DATA ENGINEERING

Part two. Pre-Processing with Feature Extraction (Algorithm)

1. Create Dummy Variables

For machine learning, there is constraint to fit a model with categorical data with some exceptions such as 'Decision Tree'. So I needed to convert all the 'categorical' variables to numeric before I split the data set into train and test. While I created dummy variables, I singled out the outcome feature, 'Discharge_Decision'.

```
dummy <- dummyVars("~.", data = subset(mydata1,select=-c(Discharge_Decision)))
mydata2 <- data.frame(predict(dummy, newdata = subset(mydata1,select=-c(Discharge_Decision))))
mydata2$Discharge_Decision <- mydata1$Discharge_Decision
str(mydata2)
```

```
## 'data.frame':    87 obs. of  21 variables:
##  $ Internal_Temp.high                : num  0 0 1 0 0 1 0 1 0 0 ...
##  $ Internal_Temp.low                 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Internal_Temp.mid                 : num  1 1 0 1 1 0 1 0 1 1 ...
```

```
##  $ Surface_Temp.high                 : num  0 1 0 0 0 0 0 0 1 0 ...
##  $ Surface_Temp.low                  : num  1 0 1 1 0 1 1 0 0 1 ...
##  $ Surface_Temp.mid                  : num  0 0 0 0 1 0 0 1 0 0 ...
##  $ Oxygen_Saturation.excellent       : num  1 1 1 0 1 0 1 1 0 1 ...
##  $ Oxygen_Saturation.good            : num  0 0 0 1 0 1 0 0 1 0 ...
##  $ Blood_Pressure.high               : num  0 1 1 1 1 0 1 0 0 0 ...
##  $ Blood_Pressure.low                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Blood_Pressure.mid                : num  1 0 0 0 0 1 0 1 1 1 ...
##  $ Suf_Temp_Stability.stable         : num  1 1 1 1 1 1 1 0 1 0 ...
##  $ Suf_Temp_Stability.unstable       : num  0 0 0 0 0 0 0 1 0 1 ...
##  $ Internal_Temp_Stability.mod.stable : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Internal_Temp_Stability.stable    : num  1 1 1 0 1 1 1 0 1 1 ...
##  $ Internal_Temp_Stability.unstable  : num  0 0 0 1 0 0 0 1 0 0 ...
##  $ Blood_Pressure_Stability.mod.stable: num  0 0 1 1 0 0 1 0 0 1 ...
##  $ Blood_Pressure_Stability.stable   : num  1 1 0 0 1 0 0 1 1 0 ...
##  $ Blood_Pressure_Stability.unstable  : num  0 0 0 0 0 1 0 0 0 0 ...
##  $ Comfort                           : num  15 10 10 15 10 15 5 10 10 10 ...
##  $ Discharge_Decision                : Factor w/ 2 levels "A","S": 1 2 1 1 1 2 2 2 2 2 ...
```

2. Split Data

- Split the data set into train and test with 75% vs 25% ratio.
- Divided features of test data from outcome feature in advance for applying prediction.

```
trainIndex <- createDataPartition(mydata2$Discharge_Decision, p = .75, list = FALSE, times = 1)
train <- mydata2[trainIndex, ]
test <- mydata2[-trainIndex, ]
#trainIndex
dim(train); dim(test);
```

```
## [1] 66 21
```

```
## [1] 21 21
```

```
test_x <- subset(test,select=-c(Discharge_Decision))
test_y <- subset(test,select=c(Discharge_Decision))
```

3. Scale Train Data

Before applying training data set to algorithms, the range of variables should be standardized to make them comparable. As the 'Comfort' feature only has different range of variables, I scaled it in the range from 0 to 1 as the other variables.

```
train_sc <- train
train_sc$Comfort <- rescale(train$Comfort, to=c(0,1))
str(train_sc)
```

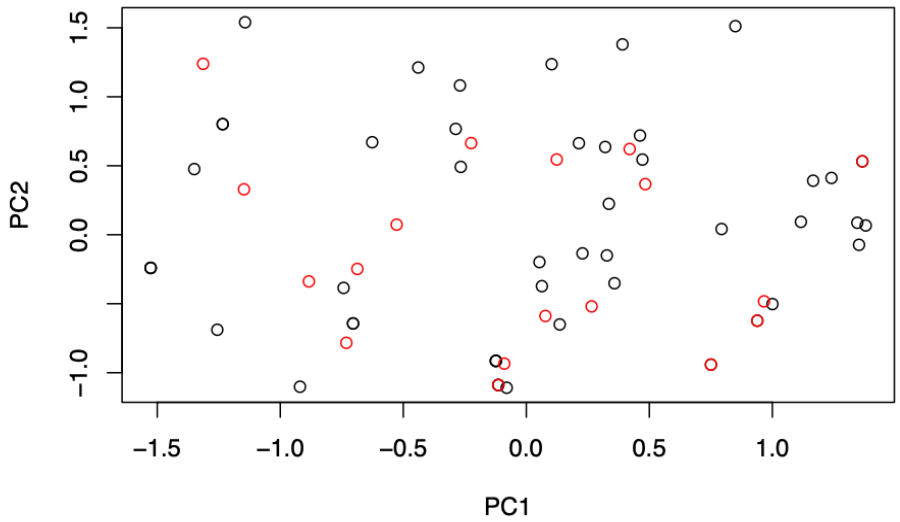
```
## 'data.frame':    66 obs. of  21 variables:
##  $ Internal_Temp.high                : num  0 0 1 0 1 0 1 0 0 0 ...
##  $ Internal_Temp.low                 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Internal_Temp.mid                 : num  1 1 0 1 0 1 0 1 1 1 ...
##  $ Surface_Temp.high                 : num  0 1 0 0 0 0 0 1 0 0 ...
##  $ Surface_Temp.low                  : num  1 0 1 0 1 1 0 0 1 0 ...
##  $ Surface_Temp.mid                  : num  0 0 0 1 0 0 1 0 0 1 ...
##  $ Oxygen_Saturation.excellent       : num  1 1 1 1 0 1 1 0 1 0 ...
##  $ Oxygen_Saturation.good            : num  0 0 0 0 1 0 0 1 0 1 ...
##  $ Blood_Pressure.high               : num  0 1 1 1 0 1 0 0 0 0 ...
##  $ Blood_Pressure.low                : num  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ Blood_Pressure.mid                : num  1 0 0 0 1 0 1 1 1 1 ...
##  $ Suf_Temp_Stability.stable         : num  1 1 1 1 1 1 0 1 0 1 ...
##  $ Suf_Temp_Stability.unstable       : num  0 0 0 0 0 0 1 0 1 0 ...
##  $ Internal_Temp_Stability.mod.stable : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Internal_Temp_Stability.stable    : num  1 1 1 1 1 1 0 1 1 1 ...
##  $ Internal_Temp_Stability.unstable  : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ Blood_Pressure_Stability.mod.stable: num  0 0 1 0 0 1 0 0 1 0 ...
##  $ Blood_Pressure_Stability.stable   : num  1 1 0 1 0 0 1 1 0 1 ...
##  $ Blood_Pressure_Stability.unstable : num  0 0 0 0 1 0 0 0 0 0 ...
##  $ Comfort                           : num  1 0.5 0.5 0.5 1 0 0.5 0.5 0.5 1 ...
##  $ Discharge_Decision                : Factor w/ 2 levels "A","S": 1 2 1 1 2 2 2 2 2 1 ...
```

4. PCA

Main reason that I ran PCA was to look at how the train data set was spread visually after completing pre-processing of data. My guess was the PCA would present less meaningful result as the binary data set, which is often quite sparse, used to perform PCA was from dummy variables. And as I expected, it was hard to find any pattern on the PCA result. However, out of the outcome, I was able to get a sense that the data set neither is linear, nor has outliers. Based on this, I estimated that Support Vector Machin (SVM) or K-Nearest Neighbors (KNN) would perform better than Logistic Regression for classification algorithm of the data set.

```
pca1 <- prcomp(subset(train_sc, select=-c(Discharge_Decision)), center=TRUE, scale. = FALSE)
plot(pca1$x[,1:2], col=train$Discharge_Decision)
```




```
pca1
```

```
## Standard deviations (1, .., p=20):
##  [1] 8.270952e-01 7.365075e-01 6.971205e-01 6.244554e-01 5.685594e-01
##  [6] 5.284298e-01 4.942518e-01 4.228344e-01 3.755670e-01 2.573555e-01
## [11] 2.212836e-01 1.871523e-01 1.288395e-01 3.768283e-16 2.485758e-16
## [16] 2.178850e-16 1.968992e-16 1.396609e-16 8.253641e-17 4.092784e-17
##
## Rotation (n x k) = (20 x 20):
##                                          PC1          PC2          PC3
```

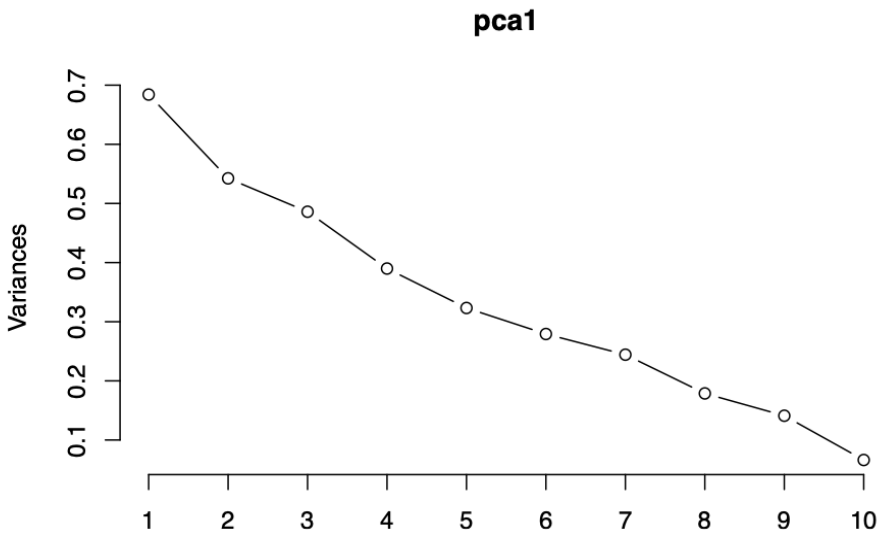
```
## Internal_Temp.high                  0.016739411  0.114157814 -0.195199075
## Internal_Temp.low                   0.102873107  0.092204919 -0.025075034
## Internal_Temp.mid                  -0.119612517 -0.206362733  0.220274109
## Surface_Temp.high                   0.193781349  0.141022639  0.068225162
## Surface_Temp.low                   -0.008290437  0.287607434  0.155008093
## Surface_Temp.mid                   -0.185490912 -0.428630073 -0.223233255
## Oxygen_Saturation.excellent        -0.420020169  0.003591122 -0.465963850
## Oxygen_Saturation.good              0.420020169 -0.003591122  0.465963850
## Blood_Pressure.high                -0.281202352  0.429821503  0.082296446
## Blood_Pressure.low                 -0.010398665 -0.018354657  0.019839917
## Blood_Pressure.mid                  0.291601017 -0.411466846 -0.102136363
## Suf_Temp_Stability.stable          -0.415105419 -0.083731348  0.356906332
## Suf_Temp_Stability.unstable         0.415105419  0.083731348 -0.356906332
## Internal_Temp_Stability.mod.stable  0.027889207  0.011676938  0.005441533
## Internal_Temp_Stability.stable     -0.046438407 -0.008352134  0.076982733
## Internal_Temp_Stability.unstable    0.018549199 -0.003324805 -0.082424266
## Blood_Pressure_Stability.mod.stable -0.036445843  0.349367393  0.023188391
## Blood_Pressure_Stability.stable    -0.106659288 -0.394045944  0.225924394
## Blood_Pressure_Stability.unstable   0.143105131  0.044678551 -0.249112784
## Comfort                             0.068322778 -0.039652725  0.035201505
##                                             PC4          PC5
## Internal_Temp.high                  0.202396140 -0.026646378
## Internal_Temp.low                   0.158606319  0.497780375
## Internal_Temp.mid                  -0.361002460 -0.471133997
## Surface_Temp.high                  -0.001269395 -0.231208425
## Surface_Temp.low                   -0.115319812 -0.098832196
## Surface_Temp.mid                    0.116589207  0.330040620
## Oxygen_Saturation.excellent        -0.108244969 -0.153399989
## Oxygen_Saturation.good              0.108244969  0.153399989
## Blood_Pressure.high                -0.163599656  0.287399290
## Blood_Pressure.low                 -0.047331532 -0.001926271
## Blood_Pressure.mid                  0.210931188 -0.285473019
## Suf_Temp_Stability.stable           0.406527706 -0.077153027
## Suf_Temp_Stability.unstable        -0.406527706  0.077153027
## Internal_Temp_Stability.mod.stable -0.012384379 -0.042539415
## Internal_Temp_Stability.stable      0.057620585  0.003996178
## Internal_Temp_Stability.unstable   -0.045236206  0.038543238
## Blood_Pressure_Stability.mod.stable  0.027518448 -0.210736218
## Blood_Pressure_Stability.stable    -0.427378334  0.282706042
## Blood_Pressure_Stability.unstable   0.399859885 -0.071969823
## Comfort                            -0.019644618  0.038691193
##                                             PC6          PC7
## Internal_Temp.high                  0.0259423331 -0.146453473
## Internal_Temp.low                   0.1115747711 -0.241513663
## Internal_Temp.mid                  -0.1375171042  0.387967136
## Surface_Temp.high                  -0.2851604301 -0.333402833
## Surface_Temp.low                    0.6528924318  0.082948980
## Surface_Temp.mid                   -0.3677320017  0.250453854
## Oxygen_Saturation.excellent         0.1238955240 -0.165315064
## Oxygen_Saturation.good             -0.1238955240  0.165315064
## Blood_Pressure.high                -0.1380230735  0.246647008
## Blood_Pressure.low                 -0.0277929477 -0.013458912
## Blood_Pressure.mid                  0.1658160212 -0.233188096
## Suf_Temp_Stability.stable           0.0463418887 -0.087725062
```

```
## Suf_Temp_Stability.unstable         -0.0463418887  0.087725062
## Internal_Temp_Stability.mod.stable  -0.0462066206 -0.026489045
## Internal_Temp_Stability.stable      -0.0009200352  0.031467603
## Internal_Temp_Stability.unstable     0.0471266558 -0.004978558
## Blood_Pressure_Stability.mod.stable -0.3986202645 -0.257023135
## Blood_Pressure_Stability.stable      0.2071949340 -0.259844208
## Blood_Pressure_Stability.unstable    0.1914253305  0.516867343
## Comfort                             -0.0037205698  0.032379098
##                                              PC8           PC9        PC10
## Internal_Temp.high                   0.410886972 -4.781233e-01  0.37455831
## Internal_Temp.low                   -0.310193755  3.670251e-01 -0.25759850
## Internal_Temp.mid                   -0.100693217  1.110982e-01 -0.11695981
## Surface_Temp.high                    0.427367796  3.827944e-01 -0.12010317
## Surface_Temp.low                    -0.240028854 -1.880668e-01  0.06066842
## Surface_Temp.mid                    -0.187338942 -1.947276e-01  0.05943475
## Oxygen_Saturation.excellent         -0.006807320  1.530056e-01 -0.03544700
## Oxygen_Saturation.good               0.006807320 -1.530056e-01  0.03544700
## Blood_Pressure.high                  0.135327142 -3.979809e-03 -0.01246872
## Blood_Pressure.low                   0.086028176  7.781228e-02  0.02351194
## Blood_Pressure.mid                  -0.221355319 -7.383247e-02 -0.01104322
## Suf_Temp_Stability.stable            0.058689753 -4.004561e-05 -0.09010372
## Suf_Temp_Stability.unstable         -0.058689753  4.004561e-05  0.09010372
## Internal_Temp_Stability.mod.stable  -0.009876893 -2.870078e-02 -0.20012161
## Internal_Temp_Stability.stable      -0.135432637  3.539040e-01  0.67228589
## Internal_Temp_Stability.unstable     0.145309530 -3.252032e-01 -0.47216427
## Blood_Pressure_Stability.mod.stable -0.446066824 -2.330641e-01  0.03113986
## Blood_Pressure_Stability.stable      0.242237223 -1.167835e-02  0.07210660
## Blood_Pressure_Stability.unstable    0.203829601  2.447425e-01 -0.10324646
## Comfort                              0.170323600 -1.486592e-03  0.08164580
##                                             PC11          PC12
## Internal_Temp.high                  -0.074319313  0.048933660
## Internal_Temp.low                    0.041693050 -0.058016143
## Internal_Temp.mid                    0.032626263  0.009082483
## Surface_Temp.high                    0.012100533  0.102404665
## Surface_Temp.low                    -0.002151723 -0.074112519
## Surface_Temp.mid                    -0.009948810 -0.028292147
## Oxygen_Saturation.excellent          0.055022522 -0.026471826
## Oxygen_Saturation.good              -0.055022522  0.026471826
## Blood_Pressure.high                  0.245395462  0.357938556
## Blood_Pressure.low                  -0.489424808 -0.638850184
## Blood_Pressure.mid                   0.244029345  0.280911628
## Suf_Temp_Stability.stable            0.017168359 -0.018637070
## Suf_Temp_Stability.unstable         -0.017168359  0.018637070
## Internal_Temp_Stability.mod.stable  -0.016971702  0.041846092
## Internal_Temp_Stability.stable       0.005462540  0.032298323
## Internal_Temp_Stability.unstable     0.011509162 -0.074144415
## Blood_Pressure_Stability.mod.stable  0.051662821 -0.116484734
## Blood_Pressure_Stability.stable     -0.023340435  0.100779677
## Blood_Pressure_Stability.unstable   -0.028322386  0.015705057
## Comfort                              0.788040067 -0.572587673
##                                             PC13          PC14
## Internal_Temp.high                   0.0762505001  6.238914e-02
## Internal_Temp.low                   -0.0290776795  6.238914e-02
## Internal_Temp.mid                   -0.0471728206  6.238914e-02
```

```
## Surface_Temp.high                   -0.0663917823  1.101061e-01
## Surface_Temp.low                     0.0384181868  1.101061e-01
## Surface_Temp.mid                     0.0279735955  1.101061e-01
## Oxygen_Saturation.excellent          0.0088532498 -3.866668e-01
## Oxygen_Saturation.good              -0.0088532498 -3.866668e-01
## Blood_Pressure.high                  0.0008203423  3.041736e-01
## Blood_Pressure.low                   0.0309165796  3.041736e-01
## Blood_Pressure.mid                  -0.0317369218  3.041736e-01
## Suf_Temp_Stability.stable           -0.0053420428  6.079680e-02
## Suf_Temp_Stability.unstable          0.0053420428  6.079680e-02
## Internal_Temp_Stability.mod.stable   0.7860751366 -3.428978e-01
## Internal_Temp_Stability.stable      -0.2399073499 -3.428978e-01
## Internal_Temp_Stability.unstable    -0.5461677868 -3.428978e-01
## Blood_Pressure_Stability.mod.stable -0.0459907013 -7.126579e-02
## Blood_Pressure_Stability.stable      0.0351680180 -7.126579e-02
## Blood_Pressure_Stability.unstable    0.0108226833 -7.126579e-02
## Comfort                              0.0709449018  6.938894e-17
##                                             PC15          PC16
## Internal_Temp.high                   1.978092e-01 -3.642310e-01
## Internal_Temp.low                    1.978092e-01 -3.642310e-01
## Internal_Temp.mid                    1.978092e-01 -3.642310e-01
## Surface_Temp.high                    4.156607e-01  3.171013e-01
## Surface_Temp.low                     4.156607e-01  3.171013e-01
## Surface_Temp.mid                     4.156607e-01  3.171013e-01
## Oxygen_Saturation.excellent          9.819514e-02  5.032227e-02
## Oxygen_Saturation.good               9.819514e-02  5.032227e-02
## Blood_Pressure.high                  3.910859e-02 -2.131103e-01
## Blood_Pressure.low                   3.910859e-02 -2.131103e-01
## Blood_Pressure.mid                   3.910859e-02 -2.131103e-01
## Suf_Temp_Stability.stable            8.263732e-02 -1.095561e-01
## Suf_Temp_Stability.unstable          8.263732e-02 -1.095561e-01
## Internal_Temp_Stability.mod.stable   1.953819e-01 -2.102175e-01
## Internal_Temp_Stability.stable       1.953819e-01 -2.102175e-01
## Internal_Temp_Stability.unstable     1.953819e-01 -2.102175e-01
## Blood_Pressure_Stability.mod.stable -2.659824e-01  2.861034e-02
## Blood_Pressure_Stability.stable     -2.659824e-01  2.861034e-02
## Blood_Pressure_Stability.unstable   -2.659824e-01  2.861034e-02
## Comfort                             -3.989864e-17 -3.295975e-16
##                                             PC17          PC18
## Internal_Temp.high                  -1.378568e-01  1.525695e-01
## Internal_Temp.low                   -1.378568e-01  1.525695e-01
## Internal_Temp.mid                   -1.378568e-01  1.525695e-01
## Surface_Temp.high                   -2.160891e-01  4.020675e-03
## Surface_Temp.low                    -2.160891e-01  4.020675e-03
## Surface_Temp.mid                    -2.160891e-01  4.020675e-03
## Oxygen_Saturation.excellent          4.171761e-02 -4.927023e-01
## Oxygen_Saturation.good               4.171761e-02 -4.927023e-01
## Blood_Pressure.high                 -9.651579e-02 -3.743616e-01
## Blood_Pressure.low                  -9.651579e-02 -3.743616e-01
## Blood_Pressure.mid                  -9.651579e-02 -3.743616e-01
## Suf_Temp_Stability.stable            1.322480e-01 -7.110529e-02
## Suf_Temp_Stability.unstable          1.322480e-01 -7.110529e-02
## Internal_Temp_Stability.mod.stable  -9.473831e-02  6.400190e-02
## Internal_Temp_Stability.stable      -9.473831e-02  6.400190e-02
```

```
## Internal_Temp_Stability.unstable    -9.473831e-02  6.400190e-02
## Blood_Pressure_Stability.mod.stable -4.863371e-01 -2.427356e-02
## Blood_Pressure_Stability.stable     -4.863371e-01 -2.427356e-02
## Blood_Pressure_Stability.unstable   -4.863371e-01 -2.427356e-02
## Comfort                              1.873501e-16  9.107298e-17
##                                             PC19          PC20
## Internal_Temp.high                   3.315605e-01  7.371514e-02
## Internal_Temp.low                    3.315605e-01  7.371514e-02
## Internal_Temp.mid                    3.315605e-01  7.371514e-02
## Surface_Temp.high                   -1.846148e-02  2.883419e-02
## Surface_Temp.low                    -1.846148e-02  2.883419e-02
## Surface_Temp.mid                    -1.846148e-02  2.883419e-02
## Oxygen_Saturation.excellent          2.960729e-01  7.848000e-02
## Oxygen_Saturation.good               2.960729e-01  7.848000e-02
## Blood_Pressure.high                 -1.495322e-01 -1.484740e-01
## Blood_Pressure.low                  -1.495322e-01 -1.484740e-01
## Blood_Pressure.mid                  -1.495322e-01 -1.484740e-01
## Suf_Temp_Stability.stable           -2.340272e-01  6.325805e-01
## Suf_Temp_Stability.unstable         -2.340272e-01  6.325805e-01
## Internal_Temp_Stability.mod.stable  -3.246714e-01 -1.220895e-01
## Internal_Temp_Stability.stable      -3.246714e-01 -1.220895e-01
## Internal_Temp_Stability.unstable    -3.246714e-01 -1.220895e-01
## Blood_Pressure_Stability.mod.stable -1.835417e-02  1.387065e-01
## Blood_Pressure_Stability.stable     -1.835417e-02  1.387065e-01
## Blood_Pressure_Stability.unstable   -1.835417e-02  1.387065e-01
## Comfort                             -2.307182e-16  7.415943e-17
```

```
#Print proportion of variance
plot(pca1, type = "l")
```




```
summary(pca1)
```

```
## Importance of components:
```

```
##                           PC1    PC2    PC3    PC4     PC5     PC6    PC7
## Standard deviation     0.8271 0.7365 0.6971 0.6245 0.56856 0.52843 0.4943
## Proportion of Variance 0.1991 0.1579 0.1414 0.1135 0.09408 0.08127 0.0711
## Cumulative Proportion  0.1991 0.3570 0.4984 0.6119 0.70599 0.78726 0.8584
##                           PC8     PC9    PC10    PC11    PC12    PC13
## Standard deviation     0.42283 0.37557 0.25736 0.22128 0.18715 0.12884
## Proportion of Variance 0.05204 0.04105 0.01928 0.01425 0.01019 0.00483
## Cumulative Proportion  0.91039 0.95145 0.97072 0.98497 0.99517 1.00000
##                            PC14      PC15      PC16      PC17      PC18
## Standard deviation     3.768e-16 2.486e-16 2.179e-16 1.969e-16 1.397e-16
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                            PC19      PC20
## Standard deviation     8.254e-17 4.093e-17
## Proportion of Variance 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00
```

```
pca_pred <- predict(pca1, train_sc)
```

1. Feature Selection

a. Recursive Feature Elimination (RFE)

I basically selected Recursive Feature Elimination with Random Forest function as the data was based on categorical. And I used k-fold Cross Validation for resampling method to keep the variance low. To get the optimal result, I changed the number of fold from 10 to 100, and the accuracy improved as the number put went up at a certain point until 50. And the Kappa value, which is regarded as rsquared value on regression, started showing zero (0.0000) from number 25. However, when I ran it again with number 100, the figure went down around the same level with 15. Therefore, I chose to keep the number to 50 for the best result.

n = 10: 0.7287

n = 15: 0.7300

n = 20: 0.7333

n = 25: 0.7533 KAPPA: 0.0000

n = 50: 0.8125 KAPPA: 0.0000

n = 100: 0. 7273 KAPPA: 0.0000

For the whole trail, 'Surface_Tem.high' had been the top variable.

```
control <- rfeControl(functions = rfFuncs, method = "cv", number = 50)
results <- rfe(train_sc[,1:20], train_sc[,21], sizes = c(1:20), rfeControl = control)
# summarize the results
print(results)
```

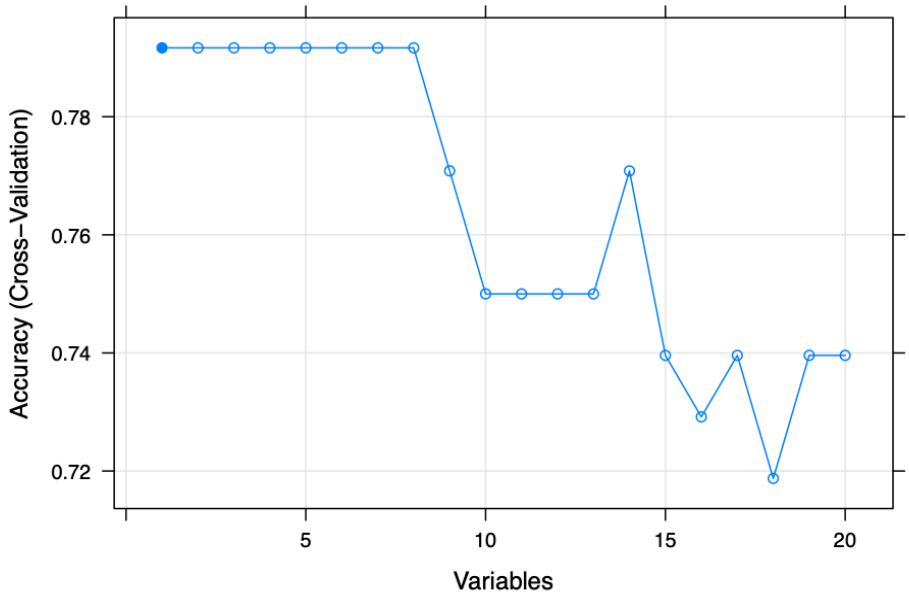
```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (50 fold)
##
## Resampling performance over subset size:
##
##  Variables Accuracy    Kappa AccuracySD KappaSD Selected
##          1   0.7917  0.00000     0.2696  0.0000        *
##          2   0.7917  0.00000     0.2696  0.0000
##          3   0.7917  0.00000     0.2696  0.0000
```

```
##          4   0.7917  0.00000     0.2696  0.0000
##          5   0.7917  0.00000     0.2696  0.0000
##          6   0.7917  0.00000     0.2696  0.0000
##          7   0.7917  0.00000     0.2696  0.0000
##          8   0.7917  0.00000     0.2696  0.0000
##          9   0.7708  0.00000     0.2910  0.0000
##         10   0.7500  0.00000     0.3094  0.0000
##         11   0.7500  0.00000     0.3094  0.0000
##         12   0.7500  0.00000     0.3094  0.0000
##         13   0.7500  0.00000     0.3094  0.0000
##         14   0.7708  0.00000     0.2910  0.0000
##         15   0.7396 -0.04762     0.3260  0.2182
##         16   0.7292 -0.09524     0.3414  0.3008
##         17   0.7396 -0.04762     0.3260  0.2182
##         18   0.7188 -0.04545     0.3406  0.2132
##         19   0.7396 -0.04762     0.3260  0.2182
##         20   0.7396 -0.04762     0.3260  0.2182
##
## The top 1 variables (out of 1):
##    Surface_Temp.high
```

```
# list the chosen features
predictors(results)
```

```
## [1] "Surface_Temp.high"
```

```
# plot the results
plot(results, type = c("g", "o"))
```



b. Random Forest

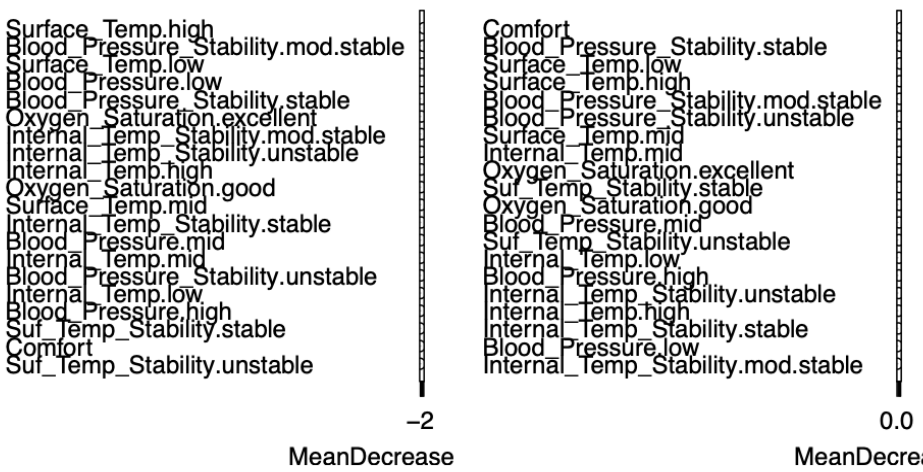
I also applied Random Forest using randomForest function to see important features in ranking of Mean

Decrease Accuracy and Mean Decrease Gini. After running the Random Forest 10 times in a row, and 'Surface_Temp', 'Comfort', 'Blood_Pressure' were ranked the highest most often.

```
train.rf <- randomForest(train_sc$Discharge_Decision ~., train_sc, importance = TRUE)
importance(train.rf)
```

```
##                                           A          S
## Internal_Temp.high                 1.77184709 -4.15453279
## Internal_Temp.low                 -0.61043724 -3.13994080
## Internal_Temp.mid                 -0.02898974 -2.96140474
## Surface_Temp.high                  7.46475307 -0.36265791
## Surface_Temp.low                   2.32576794  0.97907801
## Surface_Temp.mid                   1.46362241 -5.12496337
## Oxygen_Saturation.excellent        2.69004252 -4.73425806
## Oxygen_Saturation.good             2.14017862 -5.44963130
## Blood_Pressure.high                0.57342761 -6.55307244
## Blood_Pressure.low                 1.94121577  0.00000000
## Blood_Pressure.mid                 2.04856162 -5.36082091
## Suf_Temp_Stability.stable         -1.52741861 -4.38196947
## Suf_Temp_Stability.unstable       -0.69197833 -5.07248644
## Internal_Temp_Stability.mod.stable  0.00000000  0.00000000
## Internal_Temp_Stability.stable    -0.74289687  0.08149719
## Internal_Temp_Stability.unstable  -1.04455259  1.32682446
## Blood_Pressure_Stability.mod.stable  6.17391199 -0.90304124
## Blood_Pressure_Stability.stable    0.97291213  1.52214439
## Blood_Pressure_Stability.unstable -1.24490113 -1.18017745
## Comfort                           -2.91844593 -1.44677868
##                                   MeanDecreaseAccuracy MeanDecreaseGini
## Internal_Temp.high                          -0.3161764       0.58104095
## Internal_Temp.low                           -2.2096306       0.83127310
## Internal_Temp.mid                           -1.5373990       0.98734541
## Surface_Temp.high                            6.3703596       1.16596845
## Surface_Temp.low                             2.8424810       1.23946336
## Surface_Temp.mid                            -0.6893501       1.05302025
## Oxygen_Saturation.excellent                  0.1538777       0.92424121
## Oxygen_Saturation.good                      -0.6636958       0.90611143
## Blood_Pressure.high                         -2.4564084       0.76487095
## Blood_Pressure.low                           1.8553912       0.14656626
## Blood_Pressure.mid                          -1.0999764       0.88979923
## Suf_Temp_Stability.stable                   -3.2181151       0.91678486
## Suf_Temp_Stability.unstable                 -3.3239316       0.83984118
## Internal_Temp_Stability.mod.stable           0.0000000       0.03164501
## Internal_Temp_Stability.stable              -0.7276211       0.52167147
## Internal_Temp_Stability.unstable            -0.2160231       0.62785580
## Blood_Pressure_Stability.mod.stable          5.1591507       1.14802259
## Blood_Pressure_Stability.stable              1.5989417       1.31866086
## Blood_Pressure_Stability.unstable           -1.6515371       1.07687571
## Comfort                                     -3.3213264       1.70892219
```

```
varImpPlot(train.rf)
```



In summary, from comparing the results, I confirmed that 'Surface_Temp.high' is the most important feature, and 'Comfort' and 'Blood_Pressure' are in the most important feature category.

DATA CLUSTERING

1. K-means Clustering and Metrics

I assume that K-means clustering algorithm on the data set may not bring sensible result as the variables are binary. The algorithm computes the mean and new mean between each cluster, however, for binary or categorical data, the standard mean may not be meaningful.

The reason I initially specified k with 2 is that I predict two classes. - Performs K-means clustering with k = 2

```
train_clust <- subset(train_sc, select=-c(Discharge_Decision))
# Performs K-means clustering with k = 2
k_mean <- kmeans(train_clust, 2, iter.max = 10, nstart = 10)
# Cluster number for each of the observations
k_mean$cluster
```

```
##  1  2  3  5  6  7  8  9 10 11 12 13 14 15 17 18 19 20 21 22 23 25 26 28 30
##  2  2  2  2  1  2  2  1  2  1  1  2  1  1  1  1  1  1  1  2  1  1  1  2  1
## 32 33 34 35 36 37 38 40 41 44 46 48 50 51 53 55 57 58 59 60 61 62 63 64 65
##  1  2  1  2  1  2  1  1  1  1  1  2  1  1  2  2  1  1  1  1  1  2  1  2  2
## 66 67 68 69 70 73 74 76 78 81 84 85 86 87 88 90
##  1  2  2  2  2  2  2  1  2  2  2  2  2  2  1  1
```

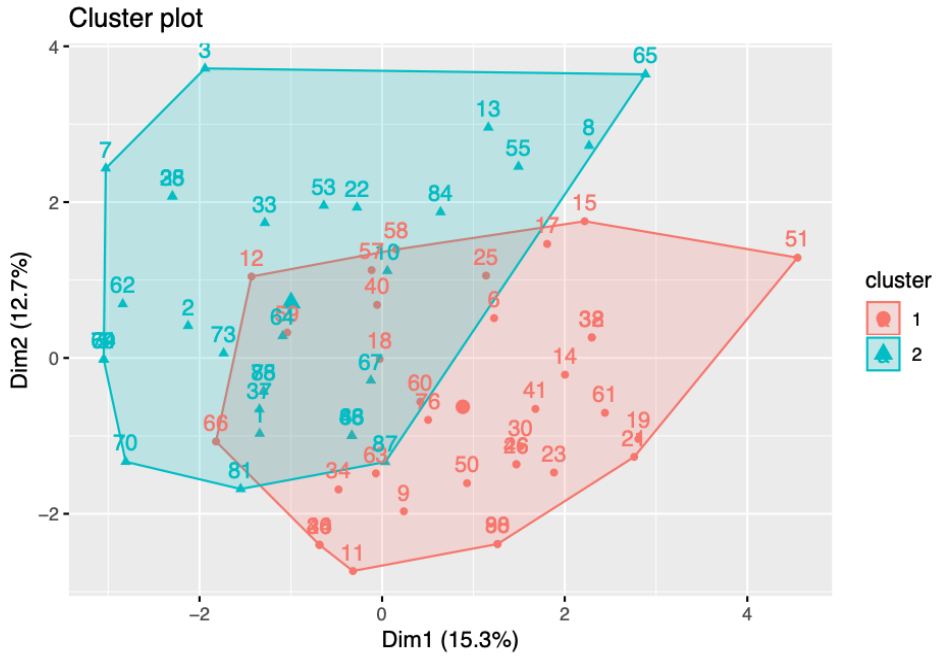
```
# Cluster size
k_mean$size
```

```
## [1] 35 31
```

```r
# Cluster means
k_mean$centers
```

```
##   Internal_Temp.high Internal_Temp.low Internal_Temp.mid Surface_Temp.high
## 1         0.08571429         0.2571429         0.6571429        0.28571429
## 2         0.19354839         0.1290323         0.6774194        0.09677419
##   Surface_Temp.low Surface_Temp.mid Oxygen_Saturation.excellent
## 1        0.2857143        0.4285714                           0
## 2        0.2580645        0.6451613                           1
##   Oxygen_Saturation.good Blood_Pressure.high Blood_Pressure.low
## 1                      1           0.2571429         0.02857143
## 2                      0           0.4193548         0.03225806
##   Blood_Pressure.mid Suf_Temp_Stability.stable Suf_Temp_Stability.unstable
## 1          0.7142857                 0.4285714                   0.5714286
## 2          0.5483871                 0.5483871                   0.4516129
##   Internal_Temp_Stability.mod.stable Internal_Temp_Stability.stable
## 1                         0.02857143                     0.9428571
## 2                         0.00000000                     0.9354839
##   Internal_Temp_Stability.unstable Blood_Pressure_Stability.mod.stable
## 1                       0.02857143                          0.2285714
## 2                       0.06451613                          0.2580645
##   Blood_Pressure_Stability.stable Blood_Pressure_Stability.unstable
## 1                       0.4857143                         0.2857143
## 2                       0.4838710                         0.2580645
##     Comfort
## 1 0.6428571
## 2 0.5322581
```

```r
# Plot K-means
fviz_cluster(k_mean, data = train_clust)
```




```
str(k_mean)
```

```
## List of 9
##  $ cluster     : Named int [1:66] 2 2 2 2 1 2 2 1 2 1 ...
##   ..- attr(*, "names")= chr [1:66] "1" "2" "3" "5" ...
##  $ centers     : num [1:2, 1:20] 0.0857 0.1935 0.2571 0.129 0.6571 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "1" "2"
##   .. ..$ : chr [1:20] "Internal_Temp.high" "Internal_Temp.low" "Internal_Temp.mid" "Surface_Temp.hig
##  $ totss       : num 223
##  $ withinss    : num [1:2] 99.8 87.2
##  $ tot.withinss: num 187
##  $ betweenss   : num 36.3
##  $ size        : int [1:2] 35 31
##  $ iter        : int 1
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

```
summary(k_mean)
```

```
##              Length Class  Mode
## cluster      66     -none- numeric
## centers      40     -none- numeric
## totss         1     -none- numeric
## withinss      2     -none- numeric
## tot.withinss  1     -none- numeric
## betweenss     1     -none- numeric
## size          2     -none- numeric
```

```
## iter          1     -none- numeric
## ifault        1     -none- numeric
```

```
k_mean
```

```
## K-means clustering with 2 clusters of sizes 35, 31
##
## Cluster means:
##   Internal_Temp.high Internal_Temp.low Internal_Temp.mid Surface_Temp.high
## 1         0.08571429         0.2571429         0.6571429        0.28571429
## 2         0.19354839         0.1290323         0.6774194        0.09677419
##   Surface_Temp.low Surface_Temp.mid Oxygen_Saturation.excellent
## 1        0.2857143        0.4285714                           0
## 2        0.2580645        0.6451613                           1
##   Oxygen_Saturation.good Blood_Pressure.high Blood_Pressure.low
## 1                      1           0.2571429         0.02857143
## 2                      0           0.4193548         0.03225806
##   Blood_Pressure.mid Suf_Temp_Stability.stable Suf_Temp_Stability.unstable
## 1          0.7142857                 0.4285714                   0.5714286
## 2          0.5483871                 0.5483871                   0.4516129
##   Internal_Temp_Stability.mod.stable Internal_Temp_Stability.stable
## 1                         0.02857143                      0.9428571
## 2                         0.00000000                      0.9354839
##   Internal_Temp_Stability.unstable Blood_Pressure_Stability.mod.stable
## 1                       0.02857143                           0.2285714
## 2                       0.06451613                           0.2580645
##   Blood_Pressure_Stability.stable Blood_Pressure_Stability.unstable
## 1                       0.4857143                         0.2857143
## 2                       0.4838710                         0.2580645
##     Comfort
## 1 0.6428571
## 2 0.5322581
##
## Clustering vector:
##  1  2  3  5  6  7  8  9 10 11 12 13 14 15 17 18 19 20 21 22 23 25 26 28 30
##  2  2  2  2  1  2  2  1  2  1  1  2  1  1  1  1  1  1  1  2  1  1  1  2  1
## 32 33 34 35 36 37 38 40 41 44 46 48 50 51 53 55 57 58 59 60 61 62 63 64 65
##  1  2  1  2  1  2  1  1  1  1  1  2  1  1  2  2  1  1  1  1  1  2  1  2  2
## 66 67 68 69 70 73 74 76 78 81 84 85 86 87 88 90
##  1  2  2  2  2  2  2  1  2  2  2  2  2  2  1  1
##
## Within cluster sum of squares by cluster:
## [1] 99.78571 87.20968
##  (between_SS / total_SS =  16.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
k_mean$withinss
```

```
## [1] 99.78571 87.20968
```

```r
k_mean$tot.withinss
```

```
## [1] 186.9954
```

```r
k_mean$betweenss
```

```
## [1] 36.33794
```

```r
k_mean$totss
```

```
## [1] 223.3333
```

```r
k_mean$betweenss/k_mean$totss
```

```
## [1] 0.1627072
```

As expected, the result showed high variance within the clusters and low variance between the clusters: The 'Within cluster sum of squares' by cluster is high with the value of 99.79 and 87.21 respectively, meaning less similarity within the groups. Also, the ratio of 'Between cluster sum of squares' over 'Total cluster sum of squares', (between_SS / total_SS), is very low as 16.3%. And this means there is homogeneity between clusters.

This could be 1. for the small size of data set, 2. for the improper number of k, or 3. for the characteristic of the data set with dummy variables. Also, data was less clearly clustered by being overlapped.

- Sum of squares within each cluster: 99.78571, 87.20968
- Total sum of squares within cluster: 186.9954
- Total sum of squares between cluster: 36.33794
- Total sum of squares: 223.3333
- Total sum of squares between cluster / Total sum of squares: 0.1627072

a. "Within sum of square (WSS)" method: Elbow method

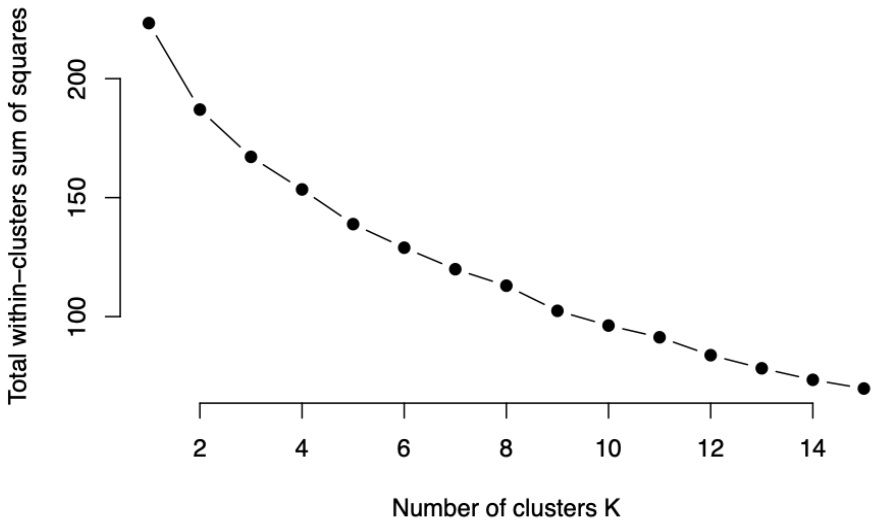
To improve the result by providing optimal values of k, I used WSS method, which gives the smallest value of k that still has a low SSE. by calculating the sum of squared errors (SSE).

- Use the "Within sum of square (WSS)" metrics.

```r
# Compute total within-cluster sum of square
wss <- function(k) {kmeans(train_clust, k, nstart = 10 )$tot.withinss}
# Set range of k values: 1 to 15
km_values <- 1:15
# Extract wss for 1 to 15 clusters
wss_values <- map_dbl(km_values, wss)
```

```
## Warning: did not converge in 10 iterations
```

```r
# Plot wss_values
plot(km_values, wss_values,
       type="b", pch = 19, frame = FALSE,
       xlab="Number of clusters K",
       ylab="Total within-clusters sum of squares")
```



The elbow method showed 2 and 9 as k value. As the result when k = 2 was not positive, I picked 9 as optimal number.

- Performs K-means clustering with k = 9

```
# Performs K-means clustering with k = 9
k_mean <- kmeans(train_clust, 9, nstart = 1)
k_mean$withinss
```

```
## [1] 21.500000  5.750000 12.208333 13.166667  9.833333 13.928571  5.071429
## [8] 12.718750 10.875000
```

```
k_mean$tot.withinss
```

```
## [1] 105.0521
```

```
k_mean$betweenss
```

```
## [1] 118.2812
```

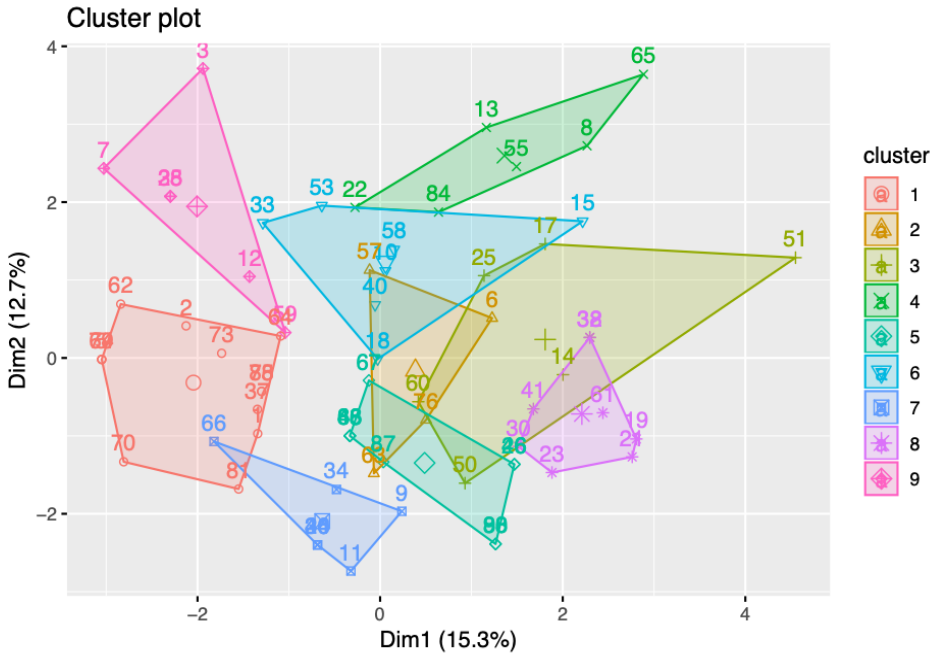
```
k_mean$totss
```

```
## [1] 223.3333
```

```
k_mean$betweenss/k_mean$totss
```

```
## [1] 0.5296175
```

```
fviz_cluster(k_mean, data = train_clust)
```



The result improved significantly: 'The sum of squares within each cluster' and 'The average distance of each cluster' decreased by more than 30%, and 'The total sum of squares between cluster' increased by 30% as well. As a result, the final ratio went up to 50.1%.

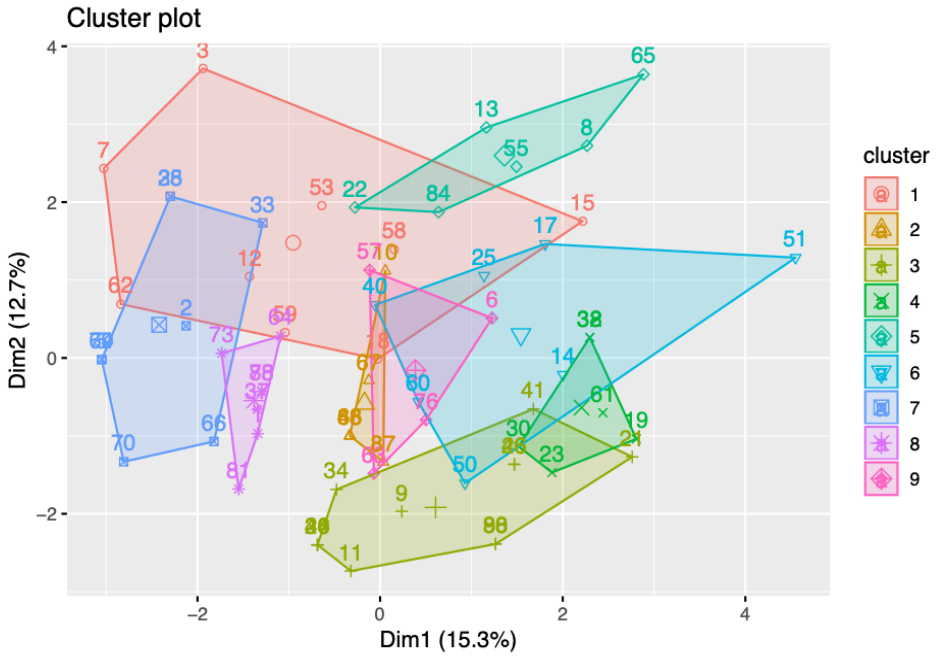
- Sum of squares within each cluster: 2.68750, 11.87500, 13.00000, 11.33333, 20.27778, 10.54167, 7.00000, 8.71875, 24.06667
- Total sum of squares within cluster: 109.5007
- Total sum of squares between cluster: 113.8326
- Total sum of squares between cluster / Total sum of squares: 0.5096984
- Compare K-means with multiple nstart trials

The final ratio was further improved from 50.96% to 54.12% when the value of nstart was large. Therefore, I computed K-means clustering with a large value of nstart such as 25 in order to get a more stable result.

```
# K-means with nstart = 1
k_mean <- kmeans(train_clust, 9, nstart = 1)
k_mean$betweenss/k_mean$totss
```

```
## [1] 0.5205135
```

```
fviz_cluster(k_mean, data = train_clust)
```




```
# K-means with nstart = 25
k_mean <- kmeans(train_clust, 9, nstart = 25)
```

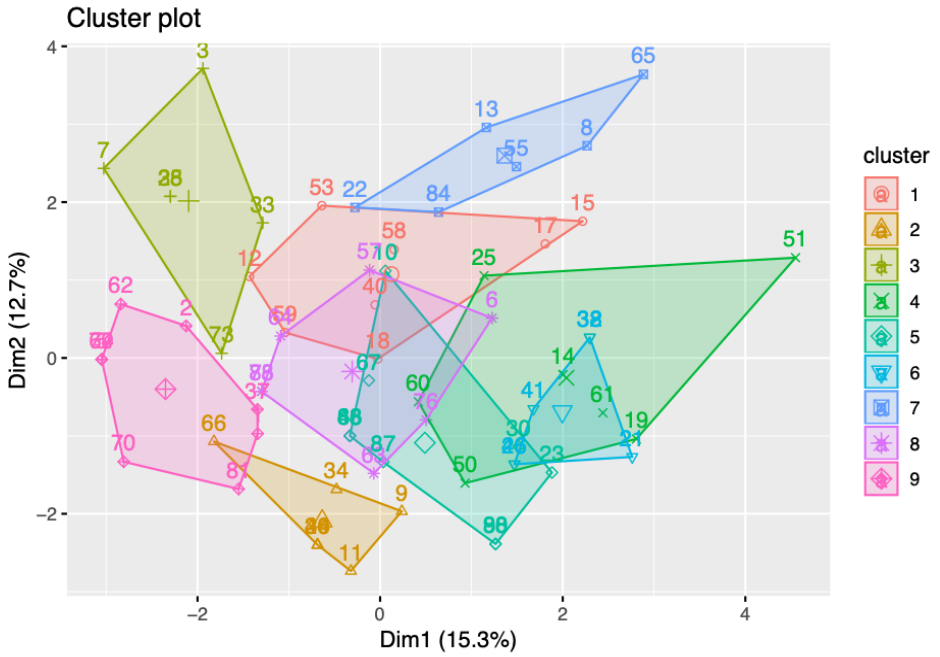
```
## Warning: did not converge in 10 iterations
```

```
## Warning: did not converge in 10 iterations
```

```
k_mean$betweenss/k_mean$totss
```

```
## [1] 0.5431787
```

```
fviz_cluster(k_mean, data = train_clust)
```



b. Silhouette method

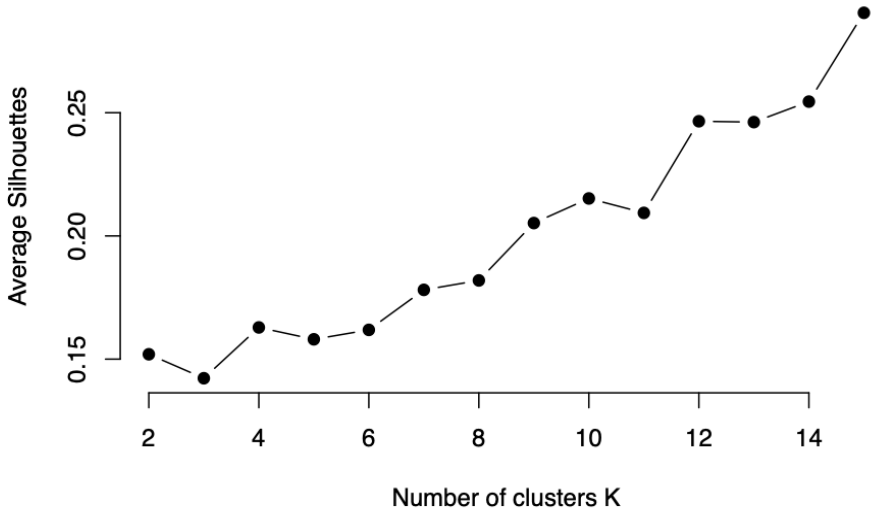
Silhouette approach measures the average silhouette of observations for different values of k. It is more qualitative clustering to see how well each object lies within its cluster. A high average silhouette width indicates a good clustering.

- Use the Silhouette metrics

```
# Compute average silhouette for k clusters
avg_sil <- function(k) {
  k_mean <- kmeans(train_clust, centers = k, nstart = 25)
  ss <- silhouette(k_mean$cluster, dist(train_clust))
  mean(ss[, 3])}
# Set range of k values: 2 to 15
ks_values <- 2:15
# Extract avg silhouette for 2-15 clusters
avg_sil_values <- map_dbl(ks_values, avg_sil)
```

```
## Warning: did not converge in 10 iterations
```

```
# Plot the result
plot(ks_values, avg_sil_values,
       type = "b", pch = 19, frame = FALSE,
       xlab = "Number of clusters K",
       ylab = "Average Silhouettes")
```



The Silhouette method showed 2 and 4 for k values.

- Performs K-means clustering with k = 4

```
# Performs K-means clustering with k = 4
k_mean <- kmeans(train_clust, 4, nstart = 25)
k_mean$withinss
```

```
## [1] 33.73214 30.50000 49.03750 36.88889
```

```
k_mean$tot.withinss
```

```
## [1] 150.1585
```

```
k_mean$betweenss
```

```
## [1] 73.1748
```

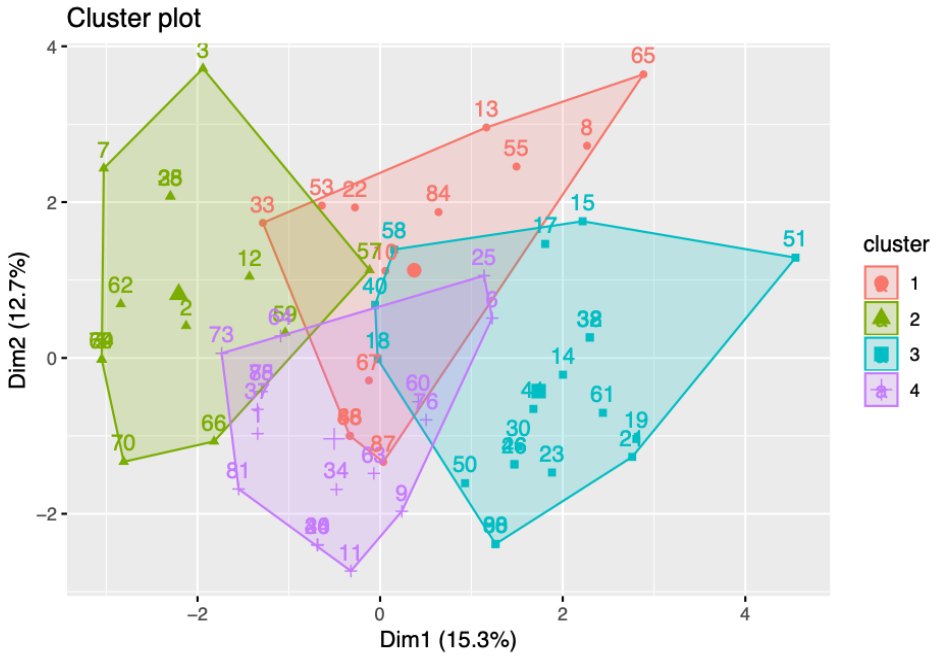
```
k_mean$totss
```

```
## [1] 223.3333
```

```
k_mean$betweenss/k_mean$totss
```

```
## [1] 0.3276484
```

```
# Plot the result
fviz_cluster(k_mean, data = train_clust)
```



The metric presented lower score with 32.86%, which means the clustering is low quality. I think it is mainly the data set itself is sparse as there is less differentiation among dummy variables, and that made the clusters overlapped.

- Sum of squares within each cluster: 49.03750, 33.73214, 32.06667, 35.08824
- Total sum of squares within cluster: 149.9245
- Total sum of squares between cluster: 73.40879
- Total sum of squares between cluster / Total sum of squares: 0. 0.3286961

c. Gap Statistic Method

The Gap Statistic compares the total intracluster variation for different values of k with their expected values under null reference distribution of the data.

```
# Perform gap stat
gap_stat <- clusGap(train_clust, FUN = kmeans, nstart = 25, K.max = 10, B = 50)
```

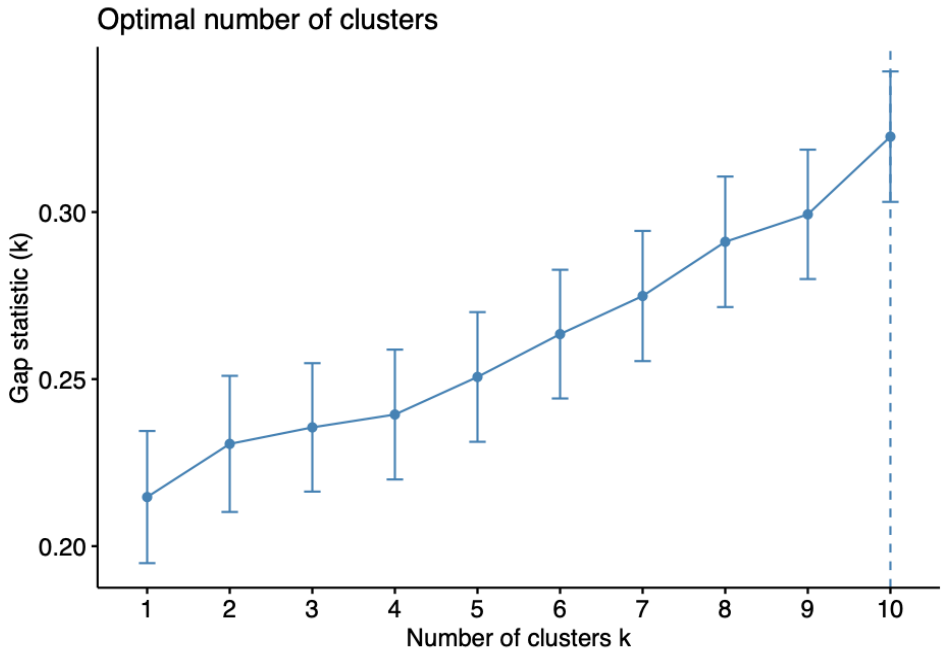
```
## Warning: did not converge in 10 iterations
```

```
# Print the result
print(gap_stat, method = "firstmax")
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = train_clust, FUNcluster = kmeans, K.max = 10, B = 50,     nstart = 25)
## B=50 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'firstmax'): 10
##          logW   E.logW       gap     SE.sim
##  [1,] 3.729434 3.944131 0.2146965 0.01977905
##  [2,] 3.627357 3.857969 0.2306116 0.02037316
```

```
##  [3,] 3.557529 3.793080 0.2355512 0.01921823
##  [4,] 3.499510 3.738914 0.2394034 0.01943075
##  [5,] 3.441101 3.691752 0.2506510 0.01940677
##  [6,] 3.386300 3.649770 0.2634702 0.01926293
##  [7,] 3.335637 3.610518 0.2748812 0.01948445
##  [8,] 3.281855 3.572960 0.2911052 0.01953630
##  [9,] 3.237362 3.536683 0.2993207 0.01936243
## [10,] 3.179800 3.502379 0.3225788 0.01952792
```

```
# Plot the result
fviz_gap_stat(gap_stat)
```



The Gap Statistic showed 9 for k values which was suggested by WSS along with 2.

In conclusion, I will confirm the optimal value for k is 9 as two of the methods out of three brought the same value with the best result of 54.12%.

- WSS: 2 and 9
- Silhouette: 2 and 4
- Gap Statistic: 9

In addition, I will choose the Gap Statistic approach as the best option for K-means clustering. Even though WSS provide the same values, it came up with two values of 2 and 9. Considered K-means clustering is sensitive to the initial input of k value and can bring different results if the order of data input is changed. This means applying the optimal value at the first trial is critical to get the best result. WSS brought 50% chance to choose the optimal value and Silhouette didn't provide the right value. Therefore, the Gap Statistic method is recommended as the best option for K-means clustering of the data set.

2. Hierarchical Clustering

Hierarchical clustering is an alternative approach to K-means clustering. Main difference is that it does not require to pre-specify the number of clusters. Also, hierarchical clustering results in a dendrogram, a tree-based representation of the observations.

a. Determining Distance Method

I used and compared 'Euclidean' and 'Manhattan' metrics to get better distance for Hierarchical clustering. Also, I applied all four linkage methods of "average", "single", "complete", and "ward" with each of matric to choose the best option.

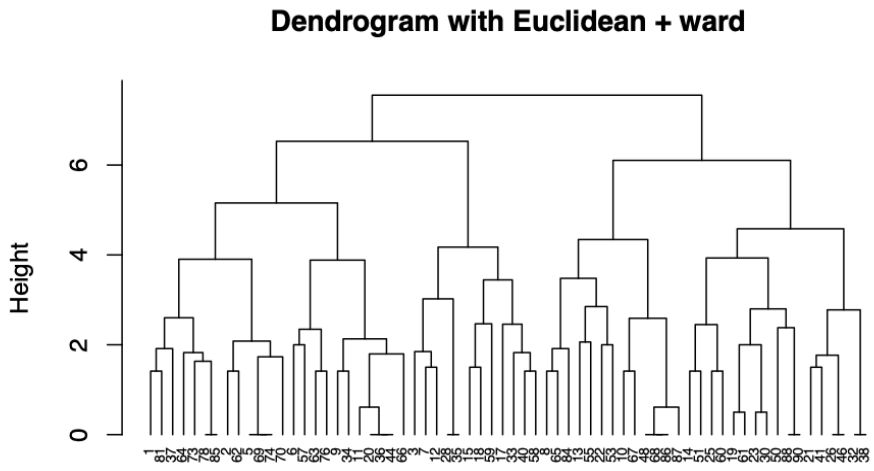
i. Euclidean Method

- Compute distance
- Calculate coefficient scores of all four linkage methods
- Plot the dendrogram

```
# Dissimilarity matrix as "Euclidean"
hc_eucld <- dist(train_clust, method = "euclidean")
# Hierarchical clustering methods to assess
m <- c( "average", "single", "complete", "ward")
names(m) <- c( "average", "single", "complete", "ward")
# Compute coefficient
ac <- function(x) {agnes(hc_eucld, method = x)$ac}
# Print method and coefficient
map_dbl(m, ac)
```

```
##   average    single  complete      ward
## 0.6369444 0.5088205 0.7124180 0.8564081
```

```
#Plot dendrogram
hc_ecld_wrd <- agnes(hc_eucld, method = "ward")
pltree(hc_ecld_wrd, cex = 0.6, hang = -1, main = "Dendrogram with Euclidean + ward")
```



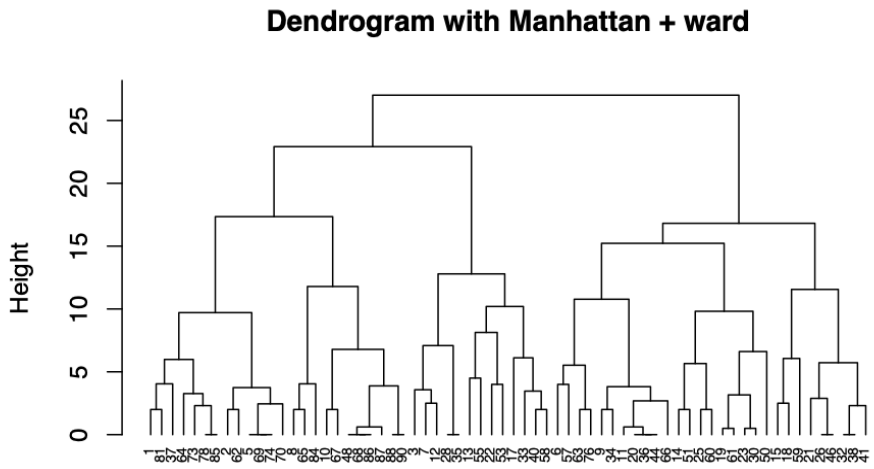
'Ward' method showed the highest ecoefficiency with 0.8564.

ii. Manhattan Method

```
# Dissimilarity matrix as "Manhattan"
hc_mht <- dist(train_clust, method = "manhattan")
# Compute coefficient
ac <- function(x) {agnes(hc_mht, method = x)$ac}
# Print method and coefficient
map_dbl(m, ac)
```

```
##   average    single  complete      ward
## 0.7943940 0.6414141 0.8674242 0.9304248
```

```
#Plot dendrogram
hc_mht_wrd <- agnes(hc_mht, method = "ward")
pltree(hc_mht_wrd, cex = 0.6, hang = -1, main = "Dendrogram with Manhattan + ward")
```

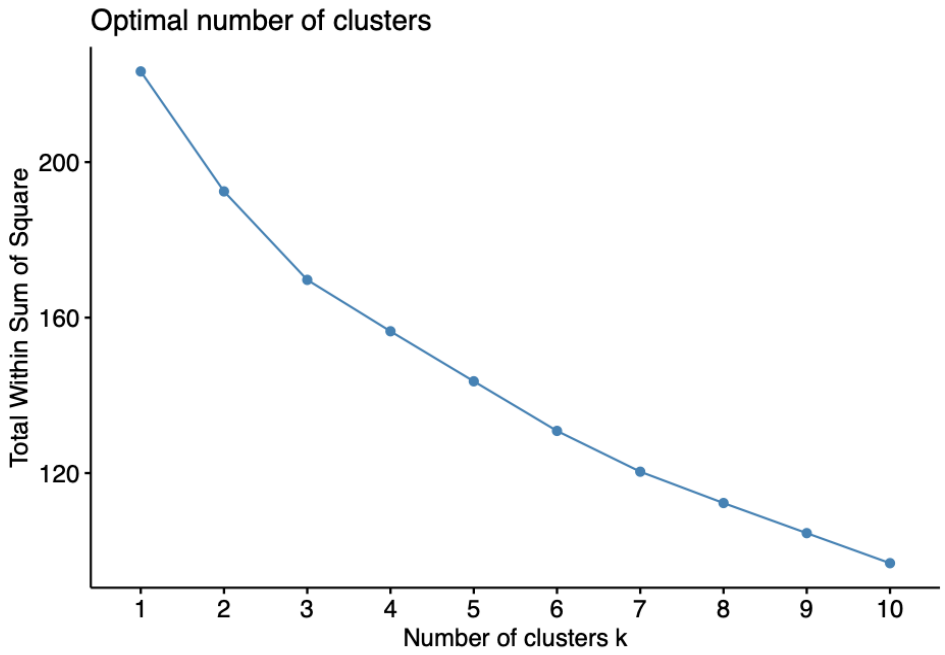


I decided Manhattan metric over Euclidean as Manhattan provided higher ecoefficiency over the four methods. And I also picked ward linkage method which presented the best result as well with 0.9304.

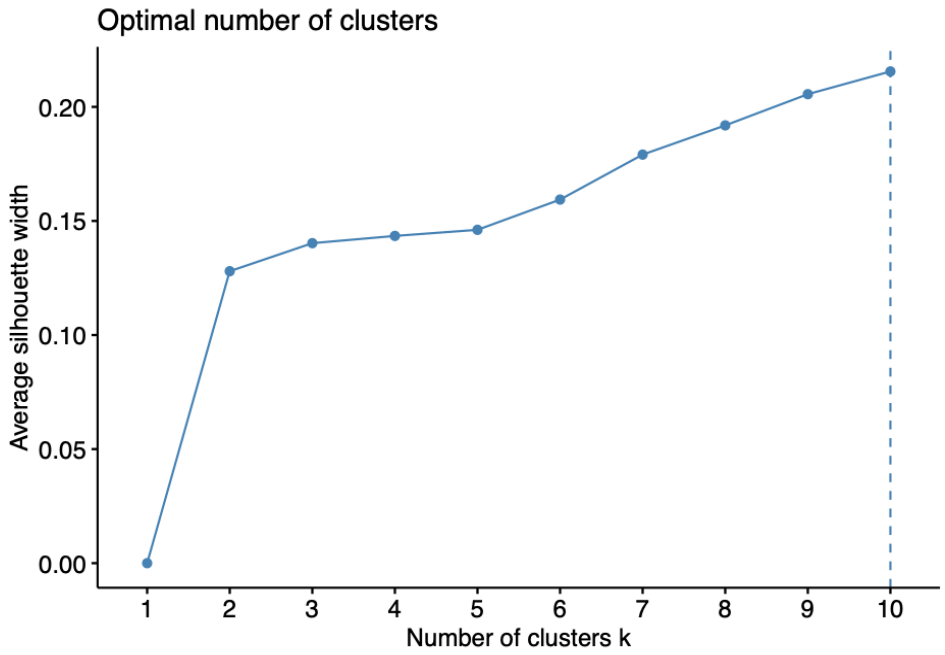
b. Determining Optimal Cluster

I executed the same approaches as I did for K-means clustering to get the optimal cluster for Hierarchical clustering.

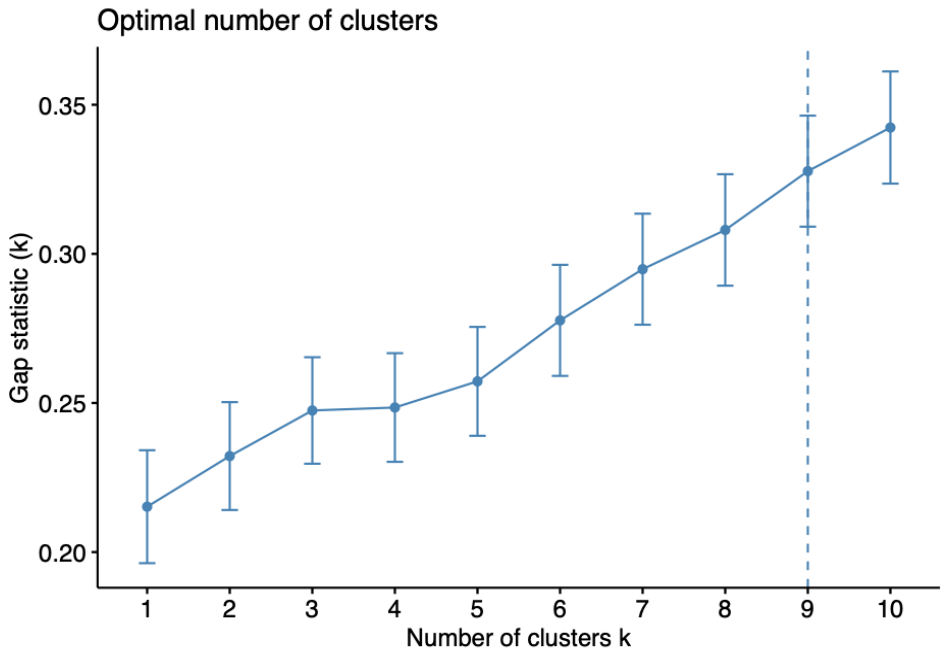
```
fviz_nbclust(train_clust, FUN = hcut, method = "wss")
```




```
fviz_nbclust(train_clust, FUN = hcut, method = "silhouette")
```




```
fviz_nbclust(train_clust, FUN = hcut, method = "gap_stat")
```

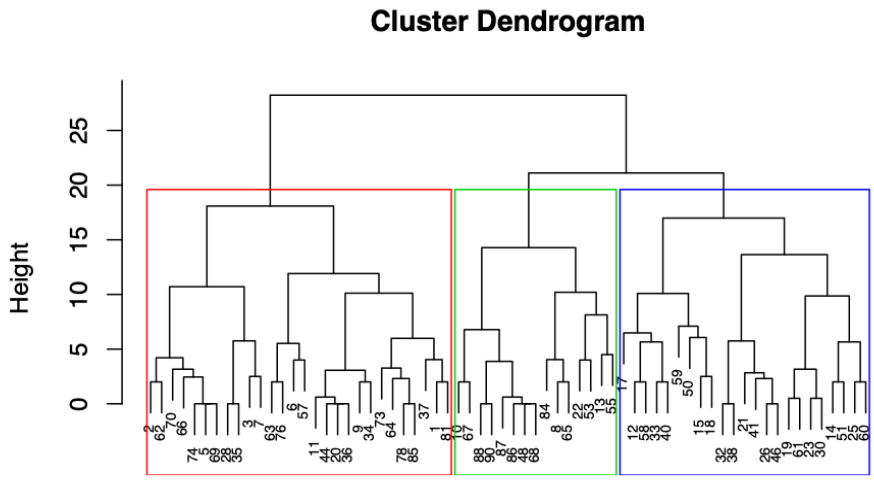


I selected optimal number of cluster as 3 for WSS method, 2 for Silhouette, and 9 for Gap Statistic.

```
# Assign for cut tree with Mantahhan + Ward's method
hc_cut <- hclust(hc_mht, method = "ward.D2" )
#Cut tree into 3 groups from optimal number of WSS
sub_grp_wss <- cutree(hc_cut, k = 3)
# View each cluster
table(sub_grp_wss)
```

```
## sub_grp_wss
##  1  2  3
## 28 15 23
```

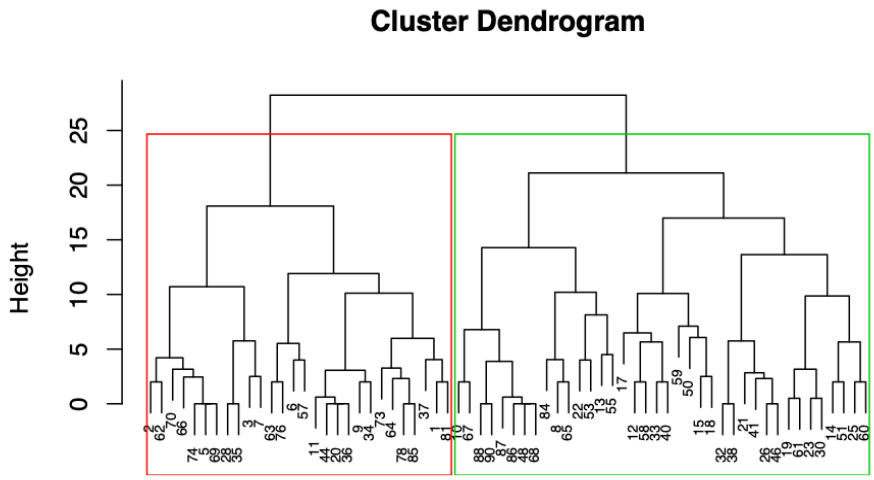
```
# Plot cluster
plot(hc_cut, cex = 0.6)
rect.hclust(hc_cut, k = 3, border = 2:5)
```




```
#Cut tree into 3 groups from optimal number of Silhouette
sub_grp_sil <- cutree(hc_cut, k = 2)
# View each cluster
table(sub_grp_sil)
```

```
## sub_grp_sil
##  1  2
## 28 38
```

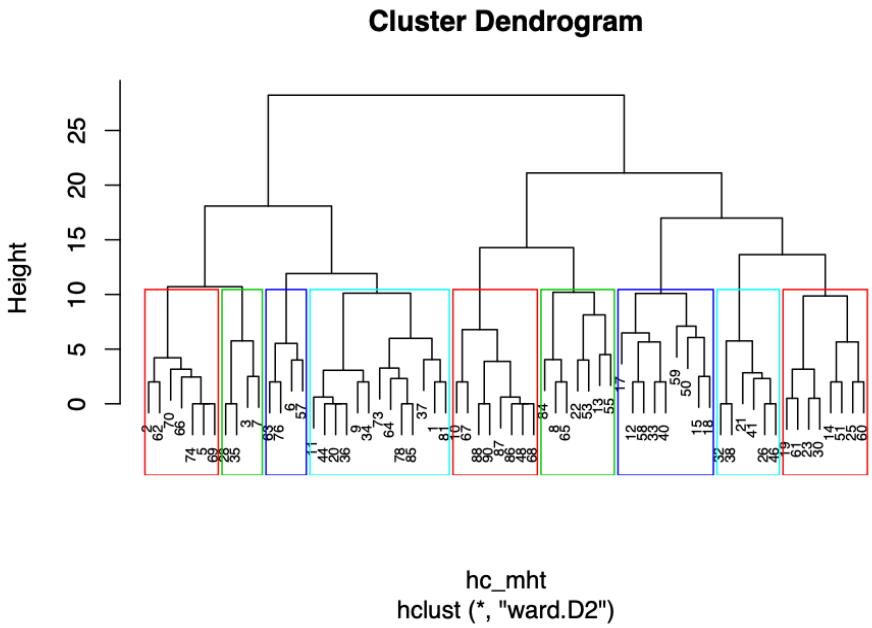
```
# Plot cluster
plot(hc_cut, cex = 0.6)
rect.hclust(hc_cut, k = 2, border = 2:5)
```




```
#Cut tree into 3 groups from optimal number of Gap Statistic
sub_grp_gap <- cutree(hc_cut, k = 9)
# View each cluster
table(sub_grp_gap)
```

```
## sub_grp_gap
##  1  2  3  4  5  6  7  8  9
## 13  7  4  4  7  8  9  8  6
```

```
# Plot cluster
plot(hc_cut, cex = 0.6)
rect.hclust(hc_cut, k = 9, border = 2:5)
```



I finally determined Gap Statistic method as the best option for Hierarchical clustering as I did for K-means clustering. Based on the different approach, the Gap Statistic method brought the same value, which makes sense as it is the same data set. Furthermore, even though all of three methods presented imbalanced result on the dendrogram comparison, Gap Statistic showed relatively balanced cluster groups than WSS and Silhouette method.

In regard with two approaches of K-means and Hierarchical clustering, I would use both at the same time. K-means clustering is useful to show how many clusters I want to look at as I indicated and to present whether the cluster is cleanly clustered or overlapped. And Hierarchical method let me analyze dendrogram and split it at a height to figure out how many clusters and what belongs to each cluster with ecoefficiency, which is good to understand the balance among the clusters.

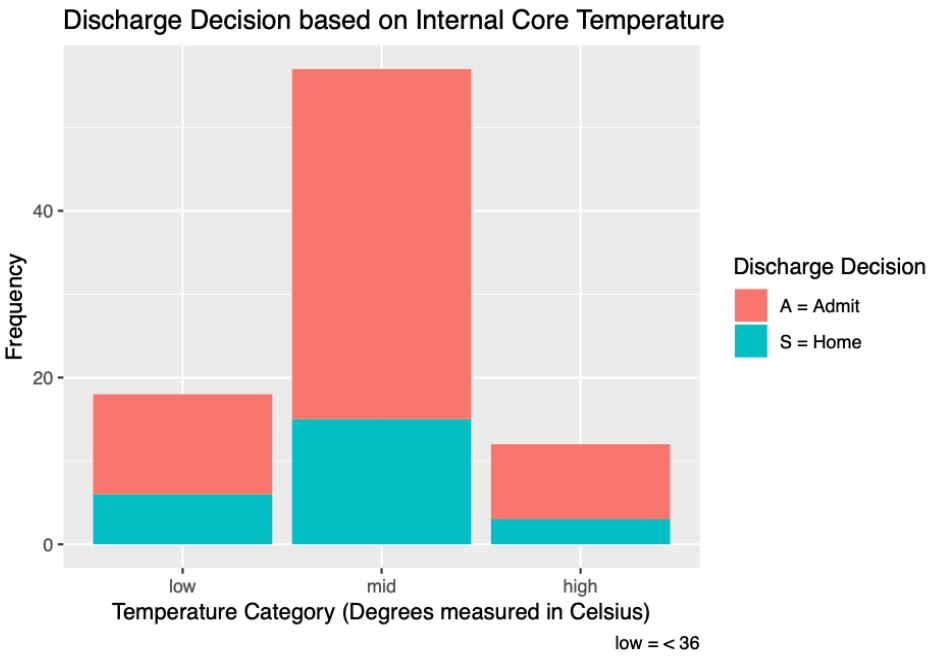
DATA CLASSIFICATION

1. Create a histogram of each feature

a. Internal Temperature

```
g <- ggplot(mydata1, aes(x = ordered(Internal_Temp,levels = c("low", "mid", "high")), fill = Discharge_I
g + geom_histogram(stat = "count", position = "stack") +
  ggtitle("Discharge Decision based on Internal Core Temperature") +
  labs(x = "Temperature Category (Degrees measured in Celsius)", y = "Frequency", caption = "low = < 36
  scale_fill_discrete(name = "Discharge Decision", labels = c("A = Admit", "S = Home"))
```

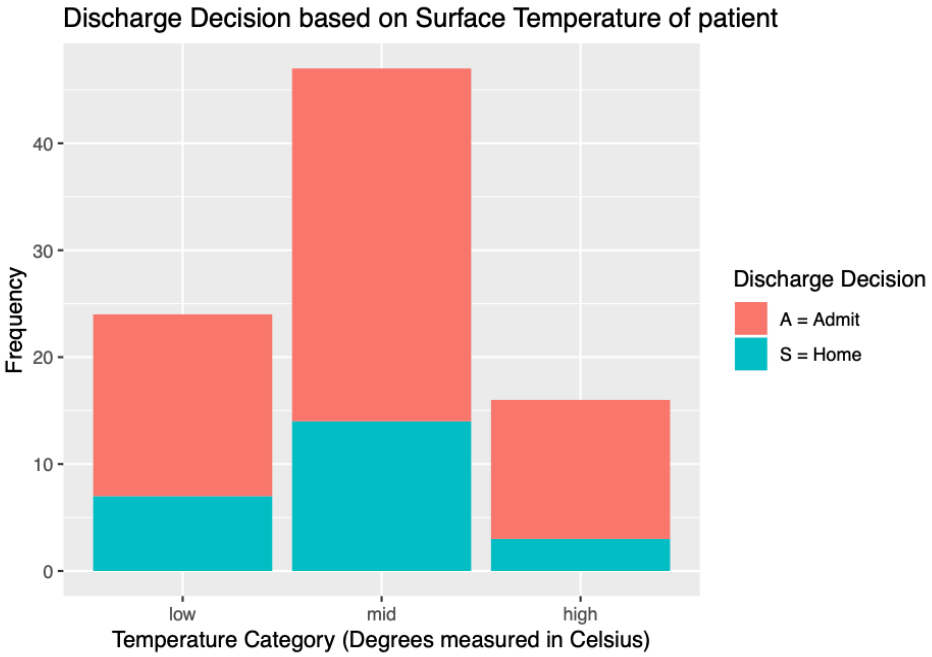
```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



b. Surface Temperature

```
g <- ggplot(mydata1, aes(x = ordered(Surface_Temp,levels = c("low", "mid", "high")), fill = Discharge_D
g + geom_histogram(stat = "count", position = "stack") +
  ggtitle("Discharge Decision based on Surface Temperature of patient") +
  labs(x = "Temperature Category (Degrees measured in Celsius)", y = "Frequency") +
  scale_fill_discrete(name = "Discharge Decision", labels = c("A = Admit", "S = Home"))
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```

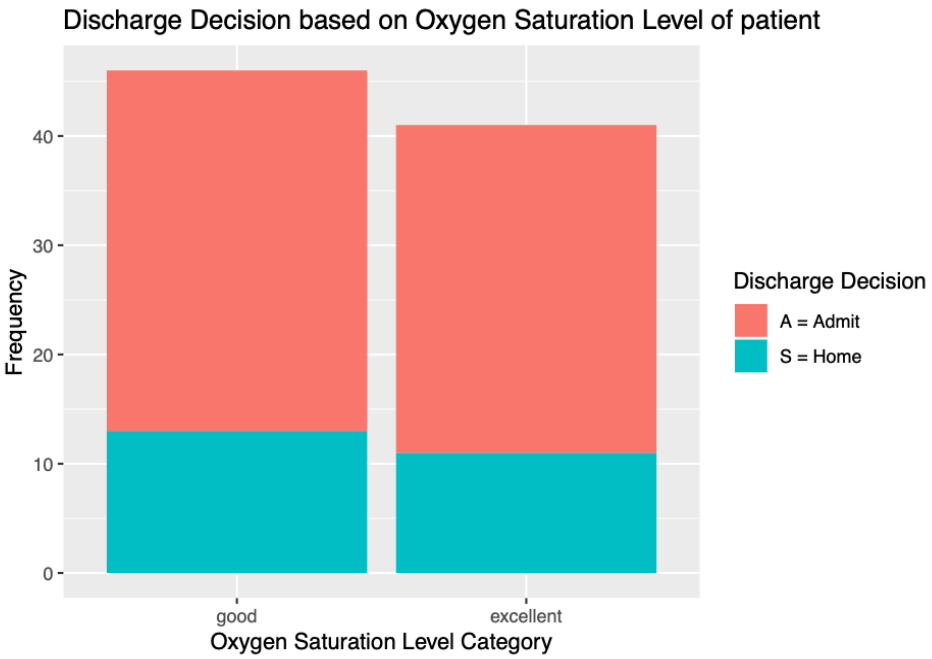


c.

Oxygen Saturation

```
g <- ggplot(mydata1, aes(x = ordered(Oxygen_Saturation,levels = c("good", "excellent")), fill = Dischar
g + geom_histogram(stat = "count", position = "stack") +
  ggtitle("Discharge Decision based on Oxygen Saturation Level of patient") +
  labs(x = "Oxygen Saturation Level Category", y = "Frequency") +
  scale_fill_discrete(name = "Discharge Decision", labels = c("A = Admit", "S = Home"))
```

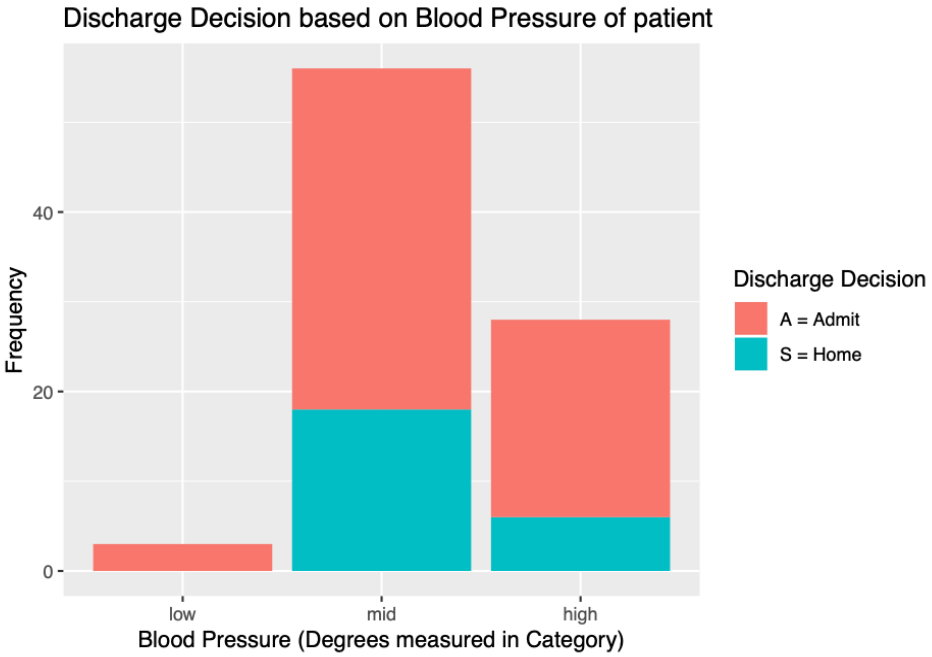
```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



d. Blood Pressure

```
g <- ggplot(mydata1, aes(x = ordered(Blood_Pressure,levels = c("low", "mid", "high")), fill = Discharge
g + geom_histogram(stat = "count", position = "stack") +
  ggtitle("Discharge Decision based on Blood Pressure of patient") +
  labs(x = "Blood Pressure (Degrees measured in Category)", y = "Frequency") +
  scale_fill_discrete(name = "Discharge Decision", labels = c("A = Admit", "S = Home"))
```

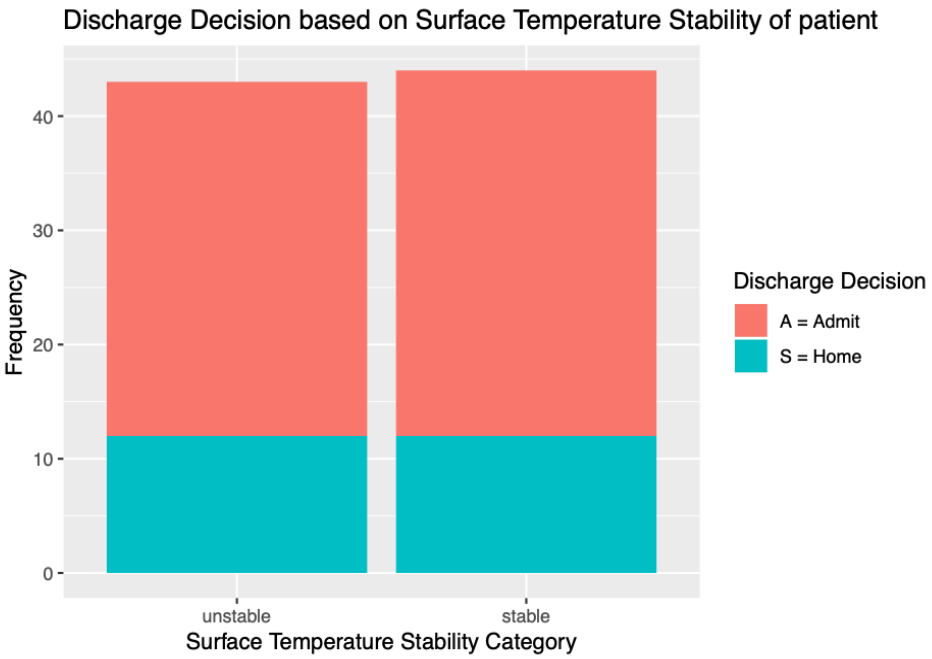
```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



e. Surface Temperature Stability

```
g <- ggplot(mydata1, aes(x = ordered(Suf_Temp_Stability,levels = c("unstable", "stable")), fill = Discha
g + geom_histogram(stat = "count", position = "stack") +
  ggtitle("Discharge Decision based on Surface Temperature Stability of patient") +
  labs(x = "Surface Temperature Stability Category", y = "Frequency") +
  scale_fill_discrete(name = "Discharge Decision", labels = c("A = Admit", "S = Home"))
```

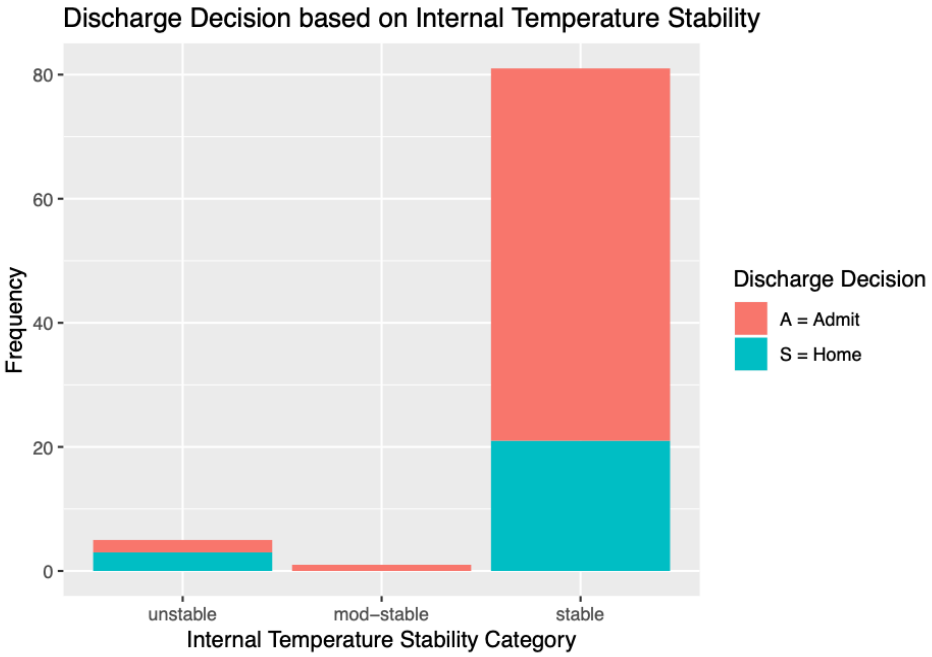
```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



f. Internal Temperature Stability

```
g <- ggplot(mydata1, aes(x = ordered(Internal_Temp_Stability,levels = c("unstable", "mod-stable", "stabl
g + geom_histogram(stat = "count", position = "stack") +
  ggtitle("Discharge Decision based on Internal Temperature Stability") +
  labs(x = "Internal Temperature Stability Category", y = "Frequency") +
  scale_fill_discrete(name = "Discharge Decision", labels = c("A = Admit", "S = Home"))
```

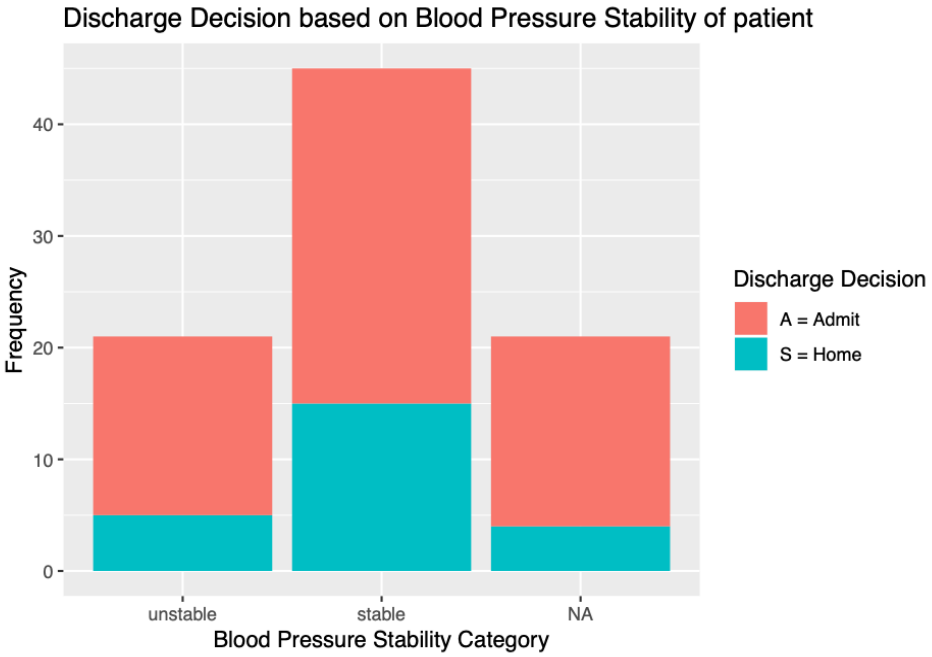
```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



g. Blood Pressure Stability

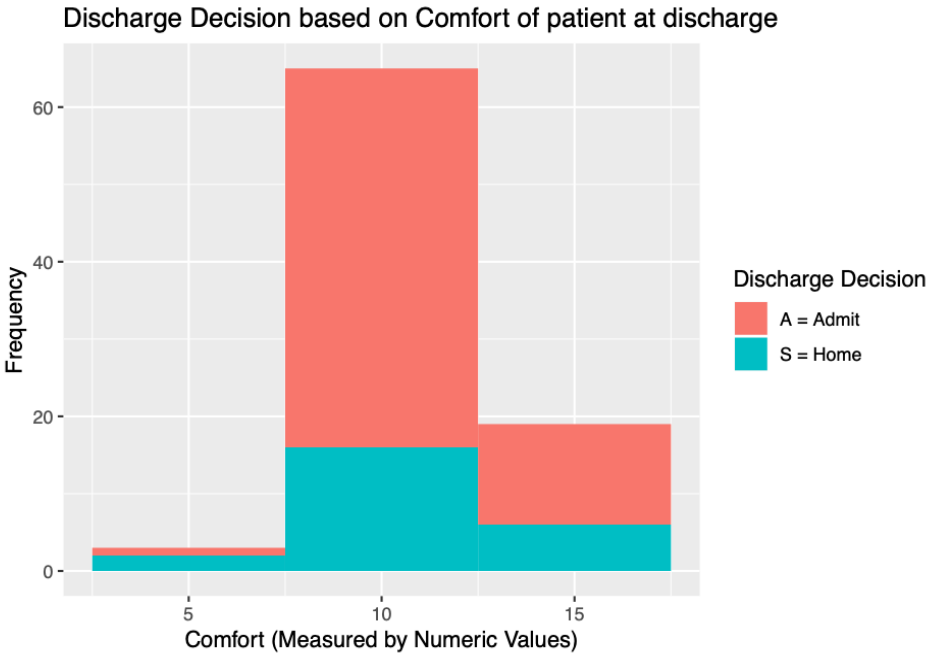
```
g <- ggplot(mydata1, aes(x = ordered(Blood_Pressure_Stability,levels = c("unstable", "stable")), fill =
g + geom_histogram(stat = "count", position = "stack") +
  ggtitle("Discharge Decision based on Blood Pressure Stability of patient") +
  labs(x = "Blood Pressure Stability Category", y = "Frequency") +
  scale_fill_discrete(name = "Discharge Decision", labels = c("A = Admit", "S = Home"))
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



h. Comfort of patients at discharge

```
g <- ggplot(mydata1, aes(x=Comfort,fill=Discharge_Decision))
g + geom_histogram(binwidth=5) +
 ggtitle("Discharge Decision based on Comfort of patient at discharge") +
 labs(x = "Comfort (Measured by Numeric Values)", y = "Frequency") +
 scale_fill_discrete(name = "Discharge Decision", labels = c("A = Admit", "S = Home"))
```



2. Classification Algorithm

a. Logistic Regression

From the result of PCA and K-means clustering, I conjecture that Logistic Regression would not be a better option for classification of the data set. However, I wanted to execute Logistic Regression and to evaluate the performance with other classification methods that I learned in the classroom. So, I fit a Logistic Regression model in order to predict the probability of a patient after surgery to get admitted to hospital or to be sent home.

- Perform simple Logistic Regression model algorithm
- Analyze parameters that I used
- Set prediction with train and test
- Compute metrics

```
# Execute simple GLM model
log_model <- glm(Discharge_Decision ~., family=binomial, data = train_sc)
# View results
summary(log_model)
```

```
##
## Call:
## glm(formula = Discharge_Decision ~ ., family = binomial, data = train_sc)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4069  -0.8658  -0.6711   0.8968   1.9377
##
```

```
## Coefficients: (7 not defined because of singularities)
##                                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)                          1.6492     1.9860   0.830    0.406
## Internal_Temp.high                  -1.0764     1.2513  -0.860    0.390
## Internal_Temp.low                    0.2383     0.7405   0.322    0.748
## Internal_Temp.mid                        NA         NA      NA       NA
## Surface_Temp.high                   -0.1322     0.9747  -0.136    0.892
## Surface_Temp.low                     0.3711     0.6802   0.546    0.585
## Surface_Temp.mid                         NA         NA      NA       NA
## Oxygen_Saturation.excellent          0.2056     0.6507   0.316    0.752
## Oxygen_Saturation.good                   NA         NA      NA       NA
## Blood_Pressure.high                 -0.6697     0.7183  -0.932    0.351
## Blood_Pressure.low                 -15.8117  1675.5700  -0.009    0.992
## Blood_Pressure.mid                       NA         NA      NA       NA
## Suf_Temp_Stability.stable            0.3420     0.6310   0.542    0.588
## Suf_Temp_Stability.unstable              NA         NA      NA       NA
## Internal_Temp_Stability.mod.stable  -17.6520  2399.5455  -0.007    0.994
## Internal_Temp_Stability.stable       -2.4905     1.5791  -1.577    0.115
## Internal_Temp_Stability.unstable         NA         NA      NA       NA
## Blood_Pressure_Stability.mod.stable  -0.1776     0.9481  -0.187    0.851
## Blood_Pressure_Stability.stable      0.1010     0.7223   0.140    0.889
## Blood_Pressure_Stability.unstable        NA         NA      NA       NA
## Comfort                             -0.5067     1.4451  -0.351    0.726
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 77.346  on 65  degrees of freedom
## Residual deviance: 70.244  on 52  degrees of freedom
## AIC: 98.244
##
## Number of Fisher Scoring iterations: 15
```

```
# Get ecoefficiency
exp(coef(log_model))
```

```
##                          (Intercept)                   Internal_Temp.high
##                         5.202691e+00                         3.408198e-01
##                    Internal_Temp.low                    Internal_Temp.mid
##                         1.269130e+00                                   NA
##                    Surface_Temp.high                     Surface_Temp.low
##                         8.761549e-01                         1.449389e+00
##                     Surface_Temp.mid          Oxygen_Saturation.excellent
##                                   NA                         1.228318e+00
##               Oxygen_Saturation.good                  Blood_Pressure.high
##                                   NA                         5.118466e-01
##                  Blood_Pressure.low                   Blood_Pressure.mid
##                         1.358468e-07                                   NA
##           Suf_Temp_Stability.stable          Suf_Temp_Stability.unstable
##                         1.407787e+00                                   NA
##  Internal_Temp_Stability.mod.stable       Internal_Temp_Stability.stable
##                         2.156815e-08                         8.287237e-02
##    Internal_Temp_Stability.unstable Blood_Pressure_Stability.mod.stable
##                                   NA                         8.372607e-01
##     Blood_Pressure_Stability.stable   Blood_Pressure_Stability.unstable
##                         1.106266e+00                                   NA
```

```
##                            Comfort
##                       6.024634e-01
```

```
# Plot logistic model
#plot(log_model)
```

I did not use any particular set of parameters to enhance the performance of Logistic Regression but applied 'glm function'. As I stated, I preferred to identify the best option for analyzing my data through various approaches first. Once I select the optimal method, I will improve it by adapting set of parameters.

The values I focused on for Logistic Regression model was 'Akaike Information Criteria (AIC)' which is a penalized likelihood value of fitting the model. The value of the model was 99.014 which is considered within the common range if it is around 100. But the p-values of each feature was not significant except for two; Blood_Pressure_Stability.stable(0.388) and Internal_Temp_Stability.stable(0.411). I also looked at coefficients values and most of them were negative. 'Null Deviance' and 'Residual Deviance' values were 77.346 and 70.244 respectively. Those values show the lack of fitting of the model, but it would be meaningful to when compared with the same values from other trials of Logistic Regression. Therefore, in this case, I will take numeric metrics scores from confusion matrix for comparison among different classification algorithms.

- Scale 'Comfort' feature of test data set before predicting
- Set test data set prediction
- Calculate Confusion Metrix
- Compute metrics scores

I had to round up and even conver the test_pred value to factor for setting the level comparable to reference value. During the process, I had to debug my note pc for version issue of Rstudio, and was able to come up with the complicated code only worked.

```
# Test Set Prediction
test_x$Comfort <- rescale(test$Comfort, to=c(0,1))
test_pred <- predict(log_model, test_x, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```
test_pred
```

```
##            4           16           24           27           29
## 7.326792e-01 6.273781e-02 2.828617e-01 1.945735e-08 3.099525e-01
##           31           39           42           43           45
## 1.679759e-01 5.526058e-01 3.392440e-01 4.207323e-02 5.564650e-01
##           52           54           56           72           75
## 3.501124e-01 2.298921e-01 2.232171e-01 5.564650e-01 2.769094e-01
##           77           79           80           82           83
## 3.404576e-01 8.539332e-01 2.213146e-01 4.054889e-01 4.211291e-01
##           89
## 3.263666e-01
```

```
# Confusion Matrix
test_pred_fact <- factor(ifelse(test_pred<0.5,"A","S"),levels=levels(test_y$Discharge_Decision))
confusionMatrix(test_pred_fact, test_y$Discharge_Decision)
```

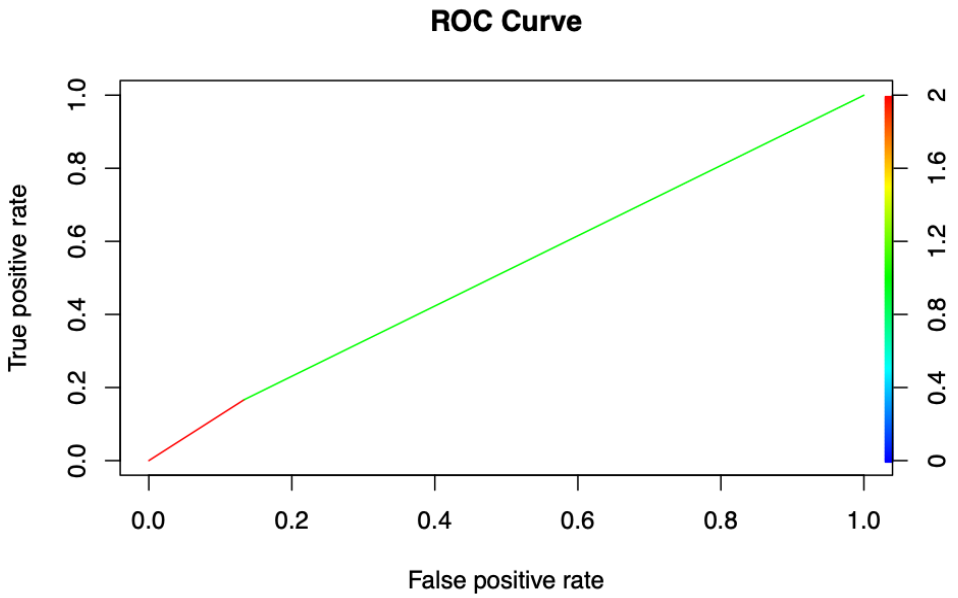
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  S
##          A 12  4
```

```
##          S  3  2
##
##                Accuracy : 0.6667
##                  95% CI : (0.4303, 0.8541)
##     No Information Rate : 0.7143
##     P-Value [Acc > NIR] : 0.7705
##
##                   Kappa : 0.1404
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.8000
##             Specificity : 0.3333
##          Pos Pred Value : 0.7500
##          Neg Pred Value : 0.4000
##              Prevalence : 0.7143
##          Detection Rate : 0.5714
##    Detection Prevalence : 0.7619
##       Balanced Accuracy : 0.5667
##
##        'Positive' Class : A
##
```

```
# Precision score
CM <- confusionMatrix(test_pred_fact, test_y$Discharge_Decision)
# Get numeric metrics scores
CM$byClass
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.8000000            0.3333333            0.7500000
##       Neg Pred Value            Precision               Recall
##            0.4000000            0.7500000            0.8000000
##                   F1           Prevalence       Detection Rate
##            0.7741935            0.7142857            0.5714286
## Detection Prevalence    Balanced Accuracy
##            0.7619048            0.5666667
```

```
# Receiver Operating Characteristic (ROC) Curve
pred_ROC <- prediction(test_x$Surface_Temp.high, test_y)
roc <- performance(pred_ROC, measure = "tpr", x.measure = "fpr")
plot(roc, main = "ROC Curve", colorize = T)
```




```
#abline(a = 0, b = 1)
# Area Under ROC curve (AUC) value
auc <- performance(pred_ROC, measure = "auc")
auc <- auc@y.values[[1]]
auc
```

```
## [1] 0.5166667
```

The accuracy of Logistic Regression model was 0.6667 with 0.7705 of p-value. This means from Logistic Regression, I can classify a patient to be admitted or to be sent to home after surgery with 66.6% of accuracy. Sensitivity is 0.8 which presents 8 patients out of 10 were properly admitted, and specificity is 0.33 describing only 3 patients out of 10 were sent home correctly.

The Receiver Operating Characteristic (ROC) Curve is calculated with sensitive (true positive) value as x-axis and specificity (false positive) value as y-axis. And for Area Under ROC curve (AUC), there are standardized categories as below:

- 0.9 ~ 1.0: Excellent
- 0.8 ~ 0.9: Good
- 0.7 ~ 0.8: Fair
- 0.6 ~ 0.7: Poor
- 0.5 ~ 0.6: Fail

As a result, the AUC value from the model was 0.5167, and can be considered not meaningful. While Logistic Regression is used to predict a binary outcome, as stated several times, it may be hard to generate meaningful result from dummy variables. Also, the number of observations of the data set was only 87, relatively small.

b. Support Vector Machine (SVM) I learned that SVM is flexible model which can handle any shape of data set including linear, radial, and polynomial. As my data set is non-linear and scattered without

patterns as well as small sized, I expect that SVM would better fit.

- Convert outcome variable as factor before train SVM
- Perform SVM using 'svmLinear' parameter of 'caret' package, took 'boot' as resampling method, and with 10 number of resampling iterations.

```
# Convert outcome variable as factor
train_svm <- train_sc
train_svm$Discharge_Decision <- as.factor(train_sc$Discharge_Decision)
typeof(train_svm$Discharge_Decision)
```

```
## [1] "integer"
```

```
# Perform SVM using 'svmLinear' parameter
trctrl <- trainControl(method = "boot", number = 10)
svm_Linear <- train(Discharge_Decision ~., data = train_svm, method = "svmLinear",
                    trControl=trctrl, scale = FALSE)
svm_Linear
```

```
## Support Vector Machines with Linear Kernel
##
## 66 samples
## 20 predictors
##  2 classes: 'A', 'S'
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 66, 66, 66, 66, 66, 66, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.6525958  -0.001051203
##
## Tuning parameter 'C' was held constant at a value of 1
```

The accuracy on the training data set ranged from 0.55 to 0.67. The tuning parameter 'C' is set as 1 as default as it is a linear model. However, the result shows the data set is not linear. Based on that, test set prediction would present less accuracy score.

- Analyze parameters that I used

I tried training data set with 'repeatedcv' parameter, and the accuracy score improved constantly showing 0.70, and kappa value went up from negative 0.1478 to negative 0.0380.

```
# Perform SVM using 'repeatedcv' parameter
trctrl_rcv <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
svm_Linear <- train(Discharge_Decision ~., data = train_svm, method = "svmLinear",
                    trControl=trctrl_rcv, scale = FALSE)
svm_Linear
```

```
## Support Vector Machines with Linear Kernel
##
## 66 samples
## 20 predictors
##  2 classes: 'A', 'S'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
```

```
## Summary of sample sizes: 59, 60, 59, 59, 59, 59, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7096825  -0.03019608
##
## Tuning parameter 'C' was held constant at a value of 1
```

As a result, I got better accuracy (0.7143) from SVM than Logistic Regression (0.6667). The p-value went down to 0.2207 from 0.7705, which means the result is more meaningful, however, the result still within not significant range. Sensitivity score went up to 0.9333 from 0.8000, and specificity went down to 0.1667 from 0.3333. This means more patients admitted and sent home properly.

```
# Test Set Prediction
test_pred_svm <- predict(svm_Linear, test_x)
test_pred_svm
```

```
##  [1] S A A A A A A A A A A A A A A A S A A A A
## Levels: A S
```

```
# Confusion Matrix
confusionMatrix(test_pred_svm, test_y$Discharge_Decision)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  S
##          A 14  5
##          S  1  1
##
##                Accuracy : 0.7143
##                  95% CI : (0.4782, 0.8872)
##     No Information Rate : 0.7143
##     P-Value [Acc > NIR] : 0.6078
##
##                   Kappa : 0.125
##  Mcnemar's Test P-Value : 0.2207
##
##             Sensitivity : 0.9333
##             Specificity : 0.1667
##          Pos Pred Value : 0.7368
##          Neg Pred Value : 0.5000
##              Prevalence : 0.7143
##          Detection Rate : 0.6667
##    Detection Prevalence : 0.9048
##       Balanced Accuracy : 0.5500
##
##        'Positive' Class : A
##
```

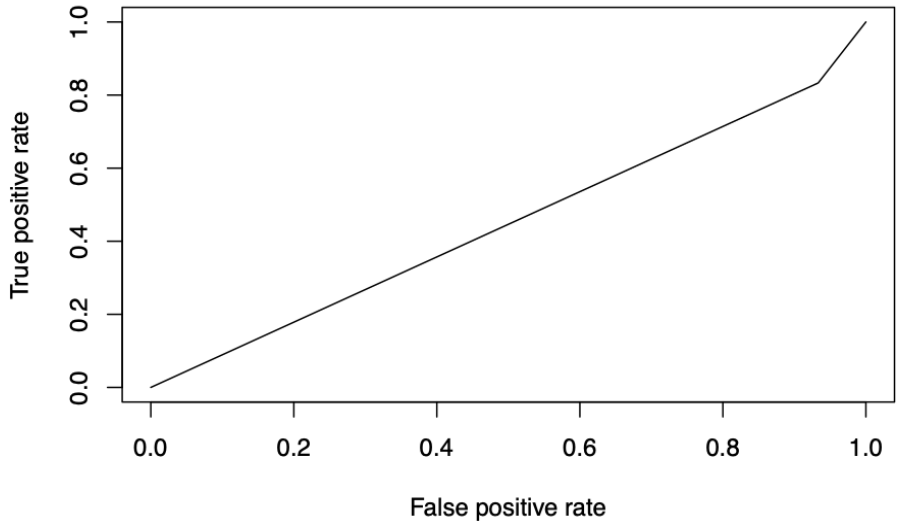
```
# Create prediction vect_y
test_pred_results <- ifelse(test_pred_svm == "A", 1, 0)
test_pred_results
```

```
##  [1] 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
```

```r
pred <- prediction(test_pred_results, test_y)

# Precision score
CM <- confusionMatrix(test_pred_svm, test_y$Discharge_Decision)
CM$byClass
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.9333333            0.1666667            0.7368421
##       Neg Pred Value            Precision               Recall
##            0.5000000            0.7368421            0.9333333
##                   F1           Prevalence       Detection Rate
##            0.8235294            0.7142857            0.6666667
## Detection Prevalence    Balanced Accuracy
##            0.9047619            0.5500000
```

```r
# ROC Curve
roc <- performance(pred, "tpr", "fpr")
plot(roc)
```




```r
# AUC
auc.tmp <- performance(pred, "auc")
auc <- as.numeric(auc.tmp@y.values)
auc
```

```
## [1] 0.45
```

- Grid search

To improve the fit of model, I kept parameter as 'repeatedcv', and conducted grid search. By using expand.grid) function, I calculated optimal C value out of range from 0.01 though 5.0, and got 0.1 for the best value for C.

```r
grid <- expand.grid(C = c(0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2,5))

svm_Linear_Grid <- train(Discharge_Decision ~., data = train_svm, method = "svmLinear",
```

```
                    trControl=trctrl_rcv,
                    preProcess = c("center"),
                    tuneGrid = grid,
                    tuneLength = 10)
```

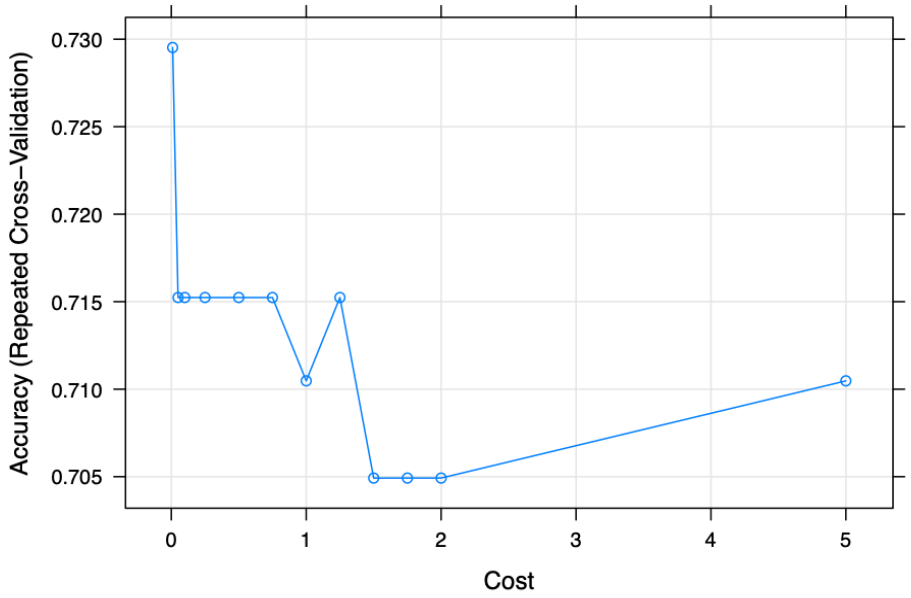
```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
svm_Linear_Grid
```

```
## Support Vector Machines with Linear Kernel
##
## 66 samples
## 20 predictors
##  2 classes: 'A', 'S'
##
## Pre-processing: centered (20)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 59, 59, 59, 60, 60, 59, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.01  0.7295238   0.00000000
##   0.05  0.7152381  -0.02352941
##   0.10  0.7152381  -0.02352941
##   0.25  0.7152381  -0.02352941
##   0.50  0.7152381  -0.02352941
##   0.75  0.7152381  -0.02352941
##   1.00  0.7104762  -0.03137255
##   1.25  0.7152381  -0.02352941
##   1.50  0.7049206  -0.03803922
##   1.75  0.7049206  -0.03803922
##   2.00  0.7049206  -0.03803922
##   5.00  0.7104762  -0.03137255
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

```r
plot(svm_Linear_Grid)
```



Updating C value as 0.1 didn't improve accuracy, but even reduced p-value further to 0.0412 from 0.2207, where the result can be considered significant. Moreover, the AUC value went up to 0.5 from 0.45.

```r
# Test Set Grid Prediction
test_pred_grid <- predict(svm_Linear_Grid, newdata = test_x)
test_pred_grid
```

```
##  [1] A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
## Levels: A S
```

```r
# Confusion Matrix
confusionMatrix(test_pred_grid, test_y$Discharge_Decision)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  S
##          A 15  6
##          S  0  0
##
##                Accuracy : 0.7143
##                  95% CI : (0.4782, 0.8872)
##     No Information Rate : 0.7143
##     P-Value [Acc > NIR] : 0.60782
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : 0.04123
##
##             Sensitivity : 1.0000
```

```
##             Specificity : 0.0000
##          Pos Pred Value : 0.7143
##          Neg Pred Value :    NaN
##              Prevalence : 0.7143
##          Detection Rate : 0.7143
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : A
##
```

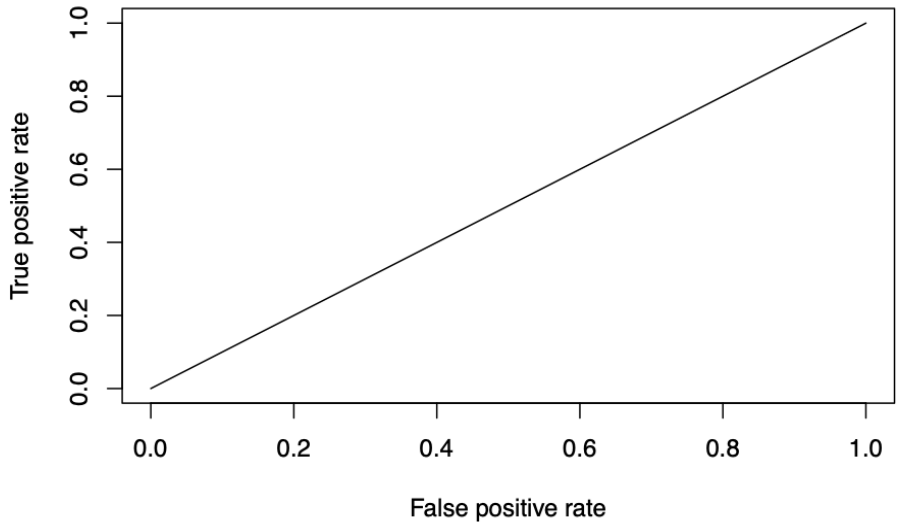
```
# Create prediction vect_y
test_pred_grid_results <- ifelse(test_pred_grid == "A", 1, 0)
test_pred_grid_results
```

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

```
pred_grid <- prediction(test_pred_grid_results, test_y)

# Precision score
CM_grid <- confusionMatrix(test_pred_grid, test_y$Discharge_Decision)
CM_grid$byClass
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            1.0000000            0.0000000            0.7142857
##       Neg Pred Value            Precision               Recall
##                  NaN            0.7142857            1.0000000
##                   F1           Prevalence       Detection Rate
##            0.8333333            0.7142857            0.7142857
## Detection Prevalence    Balanced Accuracy
##            1.0000000            0.5000000
```

```
# ROC Curve
roc <- performance(pred_grid, "tpr", "fpr")
plot(roc)
```




```
# AUC
auc.tmp <- performance(pred_grid, "auc")
auc <- as.numeric(auc.tmp@y.values)
auc
```

```
## [1] 0.5
```

I applied non-linear SVM method with parameter, 'svmRadial', and conducted the same grid search. Then I applied C = 0.25 and fit the model again. Unfortunately, the new approach didn't bring better fit results, which explains the data is not affected in applying linear or non-linear SVM algorithms.

```
svm_Radial <- train(Discharge_Decision ~., data = train_svm, method = "svmRadial",
  trControl=trctrl_rcv,
  preProcess = c("center"),
  tuneLength = 10)
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

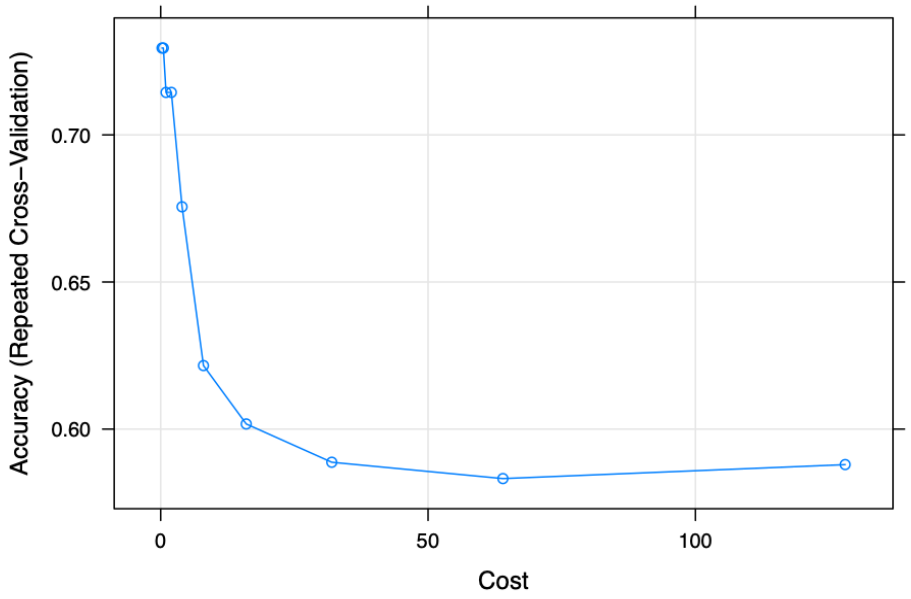
```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
svm_Radial
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 66 samples
## 20 predictors
##  2 classes: 'A', 'S'
##
## Pre-processing: centered (20)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 59, 61, 59, 59, 59, 59, ...
## Resampling results across tuning parameters:
##
```

```
##   C       Accuracy   Kappa
##     0.25  0.7295238   0.00000000
##     0.50  0.7295238   0.00000000
##     1.00  0.7144444  -0.02521008
##     2.00  0.7144444  -0.02521008
##     4.00  0.6755556  -0.08963585
##     8.00  0.6215873  -0.16512605
##    16.00  0.6017460  -0.16156863
##    32.00  0.5887302  -0.17534547
##    64.00  0.5831746  -0.18486928
##   128.00  0.5879365  -0.16526144
##
## Tuning parameter 'sigma' was held constant at a value of 0.03220109
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.03220109 and C = 0.25.
```

```
plot(svm_Radial)
```




```
grid_radial <- expand.grid(sigma = c(0.01, 0.02, 0.025, 0.03, 0.04,
 0.05, 0.06, 0.07,0.08, 0.09, 0.1, 0.25, 0.5, 0.75,0.9),
 C = c(0.01, 0.05, 0.1, 0.25, 0.5, 0.75,
 1, 1.5, 2,5))
svm_Radial_Grid <- train(Discharge_Decision ~., data = train_svm, method = "svmRadial",
                    trControl=trctrl_rcv,
                    preProcess = c("center"),
                    tuneGrid = grid_radial,
                    tuneLength = 10)
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
svm_Radial_Grid
```

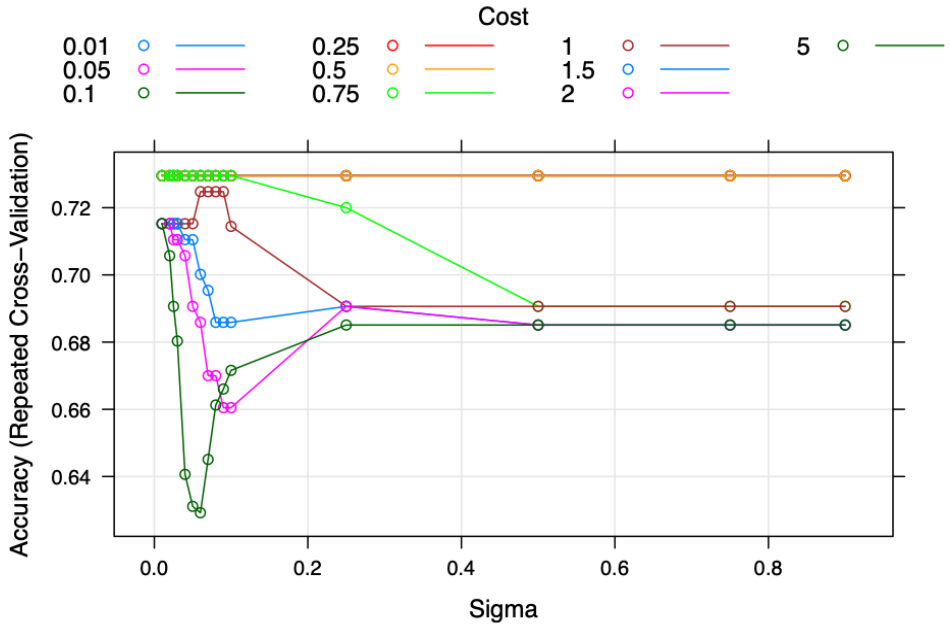
```
## Support Vector Machines with Radial Basis Function Kernel
##
## 66 samples
## 20 predictors
##  2 classes: 'A', 'S'
##
## Pre-processing: centered (20)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 59, 60, 60, 59, 59, 59, ...
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   0.010  0.01  0.7295238   0.000000000
##   0.010  0.05  0.7295238   0.000000000
##   0.010  0.10  0.7295238   0.000000000
##   0.010  0.25  0.7295238   0.000000000
##   0.010  0.50  0.7295238   0.000000000
##   0.010  0.75  0.7295238   0.000000000
##   0.010  1.00  0.7152381  -0.023529412
```

```
##   0.010  1.50  0.7152381  -0.023529412
##   0.010  2.00  0.7152381  -0.023529412
##   0.010  5.00  0.7152381  -0.023529412
##   0.020  0.01  0.7295238   0.000000000
##   0.020  0.05  0.7295238   0.000000000
##   0.020  0.10  0.7295238   0.000000000
##   0.020  0.25  0.7295238   0.000000000
##   0.020  0.50  0.7295238   0.000000000
##   0.020  0.75  0.7295238   0.000000000
##   0.020  1.00  0.7152381  -0.023529412
##   0.020  1.50  0.7152381  -0.023529412
##   0.020  2.00  0.7152381  -0.023529412
##   0.020  5.00  0.7057143  -0.036862745
##   0.025  0.01  0.7295238   0.000000000
##   0.025  0.05  0.7295238   0.000000000
##   0.025  0.10  0.7295238   0.000000000
##   0.025  0.25  0.7295238   0.000000000
##   0.025  0.50  0.7295238   0.000000000
##   0.025  0.75  0.7295238   0.000000000
##   0.025  1.00  0.7152381  -0.023529412
##   0.025  1.50  0.7152381  -0.023529412
##   0.025  2.00  0.7104762  -0.031372549
##   0.025  5.00  0.6906349  -0.062072829
##   0.030  0.01  0.7295238   0.000000000
##   0.030  0.05  0.7295238   0.000000000
##   0.030  0.10  0.7295238   0.000000000
##   0.030  0.25  0.7295238   0.000000000
##   0.030  0.50  0.7295238   0.000000000
##   0.030  0.75  0.7295238   0.000000000
##   0.030  1.00  0.7152381  -0.023529412
##   0.030  1.50  0.7152381  -0.023529412
##   0.030  2.00  0.7104762  -0.031372549
##   0.030  5.00  0.6803175  -0.076582633
##   0.040  0.01  0.7295238   0.000000000
##   0.040  0.05  0.7295238   0.000000000
##   0.040  0.10  0.7295238   0.000000000
##   0.040  0.25  0.7295238   0.000000000
##   0.040  0.50  0.7295238   0.000000000
##   0.040  0.75  0.7295238   0.000000000
##   0.040  1.00  0.7152381  -0.023529412
##   0.040  1.50  0.7104762  -0.031372549
##   0.040  2.00  0.7057143  -0.039215686
##   0.040  5.00  0.6406349  -0.130812325
##   0.050  0.01  0.7295238   0.000000000
##   0.050  0.05  0.7295238   0.000000000
##   0.050  0.10  0.7295238   0.000000000
##   0.050  0.25  0.7295238   0.000000000
##   0.050  0.50  0.7295238   0.000000000
##   0.050  0.75  0.7295238   0.000000000
##   0.050  1.00  0.7152381  -0.023529412
##   0.050  1.50  0.7104762  -0.031372549
##   0.050  2.00  0.6906349  -0.062072829
##   0.050  5.00  0.6311111  -0.148851541
##   0.060  0.01  0.7295238   0.000000000
```

```
##    0.060  0.05  0.7295238   0.000000000
##    0.060  0.10  0.7295238   0.000000000
##    0.060  0.25  0.7295238   0.000000000
##    0.060  0.50  0.7295238   0.000000000
##    0.060  0.75  0.7295238   0.000000000
##    0.060  1.00  0.7247619  -0.007843137
##    0.060  1.50  0.7001587  -0.048739496
##    0.060  2.00  0.6858730  -0.069915966
##    0.060  5.00  0.6292063  -0.151694678
##    0.070  0.01  0.7295238   0.000000000
##    0.070  0.05  0.7295238   0.000000000
##    0.070  0.10  0.7295238   0.000000000
##    0.070  0.25  0.7295238   0.000000000
##    0.070  0.50  0.7295238   0.000000000
##    0.070  0.75  0.7295238   0.000000000
##    0.070  1.00  0.7247619  -0.007843137
##    0.070  1.50  0.6953968  -0.056582633
##    0.070  2.00  0.6700000  -0.087282913
##    0.070  5.00  0.6450794  -0.134327731
##    0.080  0.01  0.7295238   0.000000000
##    0.080  0.05  0.7295238   0.000000000
##    0.080  0.10  0.7295238   0.000000000
##    0.080  0.25  0.7295238   0.000000000
##    0.080  0.50  0.7295238   0.000000000
##    0.080  0.75  0.7295238   0.000000000
##    0.080  1.00  0.7247619  -0.007843137
##    0.080  1.50  0.6858730  -0.072268908
##    0.080  2.00  0.6700000  -0.089635854
##    0.080  5.00  0.6612698  -0.110308123
##    0.090  0.01  0.7295238   0.000000000
##    0.090  0.05  0.7295238   0.000000000
##    0.090  0.10  0.7295238   0.000000000
##    0.090  0.25  0.7295238   0.000000000
##    0.090  0.50  0.7295238   0.000000000
##    0.090  0.75  0.7295238   0.000000000
##    0.090  1.00  0.7247619  -0.007843137
##    0.090  1.50  0.6858730  -0.072268908
##    0.090  2.00  0.6604762  -0.105322129
##    0.090  5.00  0.6660317  -0.102464986
##    0.100  0.01  0.7295238   0.000000000
##    0.100  0.05  0.7295238   0.000000000
##    0.100  0.10  0.7295238   0.000000000
##    0.100  0.25  0.7295238   0.000000000
##    0.100  0.50  0.7295238   0.000000000
##    0.100  0.75  0.7295238   0.000000000
##    0.100  1.00  0.7144444  -0.025210084
##    0.100  1.50  0.6858730  -0.072268908
##    0.100  2.00  0.6604762  -0.105322129
##    0.100  5.00  0.6715873  -0.095798319
##    0.250  0.01  0.7295238   0.000000000
##    0.250  0.05  0.7295238   0.000000000
##    0.250  0.10  0.7295238   0.000000000
##    0.250  0.25  0.7295238   0.000000000
##    0.250  0.50  0.7295238   0.000000000
```

```
##   0.250  0.75  0.7200000  -0.015686275
##   0.250  1.00  0.6906349  -0.064425770
##   0.250  1.50  0.6906349  -0.064425770
##   0.250  2.00  0.6906349  -0.064425770
##   0.250  5.00  0.6850794  -0.073949580
##   0.500  0.01  0.7295238   0.000000000
##   0.500  0.05  0.7295238   0.000000000
##   0.500  0.10  0.7295238   0.000000000
##   0.500  0.25  0.7295238   0.000000000
##   0.500  0.50  0.7295238   0.000000000
##   0.500  0.75  0.6906349  -0.064425770
##   0.500  1.00  0.6906349  -0.064425770
##   0.500  1.50  0.6850794  -0.073949580
##   0.500  2.00  0.6850794  -0.073949580
##   0.500  5.00  0.6850794  -0.073949580
##   0.750  0.01  0.7295238   0.000000000
##   0.750  0.05  0.7295238   0.000000000
##   0.750  0.10  0.7295238   0.000000000
##   0.750  0.25  0.7295238   0.000000000
##   0.750  0.50  0.7295238   0.000000000
##   0.750  0.75  0.6906349  -0.064425770
##   0.750  1.00  0.6906349  -0.064425770
##   0.750  1.50  0.6850794  -0.073949580
##   0.750  2.00  0.6850794  -0.073949580
##   0.750  5.00  0.6850794  -0.073949580
##   0.900  0.01  0.7295238   0.000000000
##   0.900  0.05  0.7295238   0.000000000
##   0.900  0.10  0.7295238   0.000000000
##   0.900  0.25  0.7295238   0.000000000
##   0.900  0.50  0.7295238   0.000000000
##   0.900  0.75  0.6906349  -0.064425770
##   0.900  1.00  0.6906349  -0.064425770
##   0.900  1.50  0.6850794  -0.073949580
##   0.900  2.00  0.6850794  -0.073949580
##   0.900  5.00  0.6850794  -0.073949580
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.9 and C = 0.01.
```

```
plot(svm_Radial_Grid)
```




```
# Test Set Grid Prediction
test_pred_Radial_Grid <- predict(svm_Radial_Grid, newdata = test_x)
test_pred_Radial_Grid
```

```
##  [1] A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A
## Levels: A S
```

```
# Confusion Matrix
confusionMatrix(test_pred_Radial_Grid, test_y$Discharge_Decision)
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction  A  S
##          A 15  6
##          S  0  0
##
##                Accuracy : 0.7143
##                  95% CI : (0.4782, 0.8872)
##     No Information Rate : 0.7143
##     P-Value [Acc > NIR] : 0.60782
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : 0.04123
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.7143
##          Neg Pred Value :    NaN
```

```
##              Prevalence : 0.7143
##          Detection Rate : 0.7143
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : A
##
```

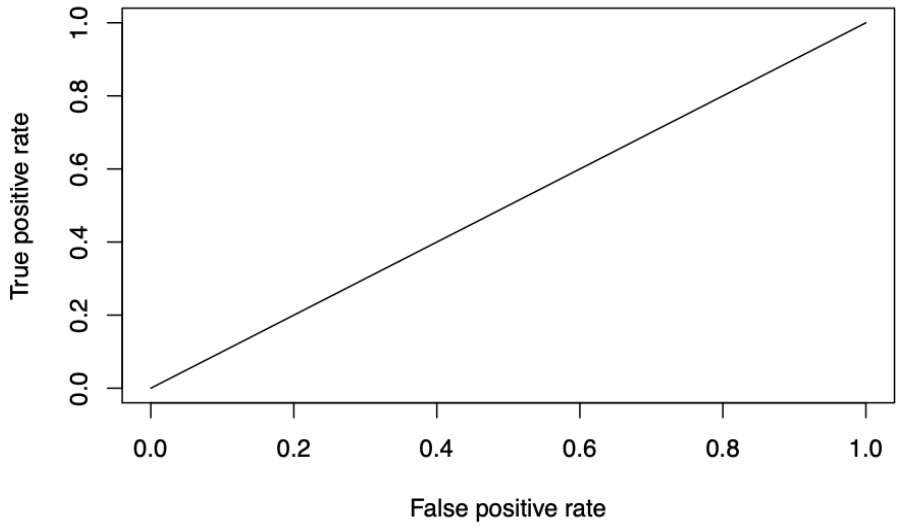
```
# Create prediction vect_y
test_pred_Radial_grid_results <- ifelse(test_pred_Radial_Grid == "A", 1, 0)
test_pred_Radial_grid_results
```

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

```
pred_Radial_grid <- prediction(test_pred_Radial_grid_results, test_y)

# Precision score
CM_Radial_grid <- confusionMatrix(test_pred_Radial_Grid, test_y$Discharge_Decision)
CM_Radial_grid$byClass
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            1.0000000            0.0000000            0.7142857
##       Neg Pred Value            Precision               Recall
##                  NaN            0.7142857            1.0000000
##                   F1           Prevalence       Detection Rate
##            0.8333333            0.7142857            0.7142857
## Detection Prevalence    Balanced Accuracy
##            1.0000000            0.5000000
```

```
# ROC Curve
roc <- performance(pred_Radial_grid, "tpr", "fpr")
plot(roc)
```




```
# AUC
auc.tmp <- performance(pred_Radial_grid, "auc")
```

```
auc <- as.numeric(auc.tmp@y.values)
auc
```

```
## [1] 0.5
```

Overall, even though there are not critical differences in terms of fitting models between Logistic Regression and SVM, SVM with parameters of 'svmLinear' with 'repeatedcv' provided the best result. Given that the accuracy, sensitivity, and specificity scores are similar, lower p-value and better AUC score adds more robust points to the model.

I also considered K Nearest Neighbor (KNN) algorithm for the data set as it doesn't require particular assumptions about data set and versatile. However, considering the data set is quite small size and the KNN can be affected noise or irrelevant features and scale of data, I speculated it may not be suitable for dominantly consisted with dummy variables from categorial data.