Breast Cancer Data: Handling Missing Values

Alena Fedash

2022-10-31

Contents

Please find below my step-by-step solution in R with explanations and comments for each step.

Question 1

Task: Use the mean/mode imputation method to impute values for the missing data.

Step 0: Load the libraries

```
library(dplyr)
library(tidyverse)
library(dslabs)
library(data.table)
library(ggplot2)
library(plotly)
library(outliers)
library(qcc)
library(mctest)
library(ppcor)
library(car)
library(psych)
library(ggthemes)
library(corrplot)
library(DAAG)
library(GGally)
library(caret)
library(psych)
library(ggpubr)
library(tree)
library(randomForest)
library(vip)
library(rsample)
library(knitr)
```

Step 1: Load the data set

```
data <- read.table("breast-cancer-wisconsin.data.txt",
                   header = FALSE,
                   stringsAsFactors = FALSE,
                   sep = ",",
                   dec = ".")
head(data)
```

```
##        V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11
## 1 1000025  5  1  1  1  2  1  3  1   1   2
## 2 1002945  5  4  4  5  7 10  3  2   1   2
## 3 1015425  3  1  1  1  2  2  3  1   1   2
## 4 1016277  6  8  8  1  3  4  3  7   1   2
## 5 1017023  4  1  1  3  2  1  3  1   1   2
## 6 1017122  8 10 10  8  7 10  9  7   1   4
```

As we can see, the data set lacks column names. Let's set them for better understanding of the data, and also set the types of data contained in each column.

Id number is an integer variable, columns 2 through 9 are values for each attribute on a scale from 1 to 10, so we can set them as integers too, and the final column is class, which means that we should set it as a factor. Since there are only 2 classes - '2' and '4', lets transform this variable to 0 (benign) and 1 (malignant).

Atrribute description for further reference: 1. Sample code number: id number 2. Clump Thickness: 1 - 10 3. Uniformity of Cell Size: 1 - 10 4. Uniformity of Cell Shape: 1 - 10 5. Marginal Adhesion: 1 - 10 6. Single Epithelial Cell Size: 1 - 10 7. Bare Nuclei: 1 - 10 8. Bland Chromatin: 1 - 10 9. Normal Nucleoli: 1 - 10 10. Mitoses: 1 - 10 11. Class: (2 for benign, 4 for malignant)

```
#transform response classes to 0 and 1:
data$V11[data$V11==2] <- 0
data$V11[data$V11==4] <- 1

#set value types
data <- transform(data,
                  #ID Number
                  V1=as.integer(data$V1),
                  V2=as.integer(data$V2),
                  V3=as.integer(data$V3),
                  V4=as.integer(data$V4),
                  V5=as.integer(data$V5),
                  V6=as.integer(data$V6),
                  V7=as.integer(data$V7),
                  V8=as.integer(data$V8),
                  V9=as.integer(data$V9),
                  V10=as.integer(data$V10),
                  V11=as.factor(data$V11))
```

```
## Warning in eval(substitute(list(...)), `_data`, parent.frame()): NAs introduced
## by coercion
```

```
#change column names
colnames(data) <- c("ID", "Clump_Thickness", "Uniformity_Size", "Uniformity_Shape",
                    "Marginal_Adhesion", "Single_Epith_Size", "Bare_Nuclei", "Bland_Chromatin",
                    "Normal_Nucleoli", "Mitoses", "Class")
summary(data)
```

```
##        ID           Clump_Thickness  Uniformity_Size  Uniformity_Shape
##  Min.   :   61634   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.:  870688   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000
##  Median : 1171710   Median : 4.000   Median : 1.000   Median : 1.000
##  Mean   : 1071704   Mean   : 4.418   Mean   : 3.134   Mean   : 3.207
##  3rd Qu.: 1238298   3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000
##  Max.   :13454352   Max.   :10.000   Max.   :10.000   Max.   :10.000
##
##  Marginal_Adhesion Single_Epith_Size  Bare_Nuclei     Bland_Chromatin
##  Min.   : 1.000    Min.   : 1.000    Min.   : 1.000   Min.   : 1.000
##  1st Qu.: 1.000    1st Qu.: 2.000    1st Qu.: 1.000   1st Qu.: 2.000
##  Median : 1.000    Median : 2.000    Median : 1.000   Median : 3.000
##  Mean   : 2.807    Mean   : 3.216    Mean   : 3.545   Mean   : 3.438
##  3rd Qu.: 4.000    3rd Qu.: 4.000    3rd Qu.: 6.000   3rd Qu.: 5.000
##  Max.   :10.000    Max.   :10.000    Max.   :10.000   Max.   :10.000
##                                      NA's   :16
##  Normal_Nucleoli     Mitoses       Class
##  Min.   : 1.000   Min.   : 1.000   0:458
##  1st Qu.: 1.000   1st Qu.: 1.000   1:241
##  Median : 1.000   Median : 1.000
##  Mean   : 2.867   Mean   : 1.589
##  3rd Qu.: 4.000   3rd Qu.: 1.000
##  Max.   :10.000   Max.   :10.000
##
```

Step 2: Locate NA Values

As we can see from the summary of our data above, **there are NA values in only one column - Bare_Nuclei**. There are 16 NAs. Let's see the rows where they are contained:

```
na<-data[is.na(data$Bare_Nuclei),]
na
```

```
##          ID Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 24  1057013               8               4                5                 1
## 41  1096800               6               6                6                 9
## 140 1183246               1               1                1                 1
## 146 1184840               1               1                3                 1
## 159 1193683               1               1                2                 1
## 165 1197510               5               1                1                 1
## 236 1241232               3               1                4                 1
## 250  169356               3               1                1                 1
## 276  432809               3               1                3                 1
## 293  563649               8               8                8                 1
## 295  606140               1               1                1                 1
## 298   61634               5               4                3                 1
## 316  704168               4               6                5                 6
## 322  733639               3               1                1                 1
## 412 1238464               1               1                1                 1
## 618 1057067               1               1                1                 1
##     Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 24                  2          NA               7               3       1     1
## 41                  6          NA               7               8       1     0
## 140                 1          NA               2               1       1     0
## 146                 2          NA               2               1       1     0
```

```
## 159                3          NA                1                1       1      0
## 165                2          NA                3                1       1      0
## 236                2          NA                3                1       1      0
## 250                2          NA                3                1       1      0
## 276                2          NA                2                1       1      0
## 293                2          NA                6               10       1      1
## 295                2          NA                2                1       1      0
## 298                2          NA                2                3       1      0
## 316                7          NA                4                9       1      0
## 322                2          NA                3                1       1      0
## 412                1          NA                2                1       1      0
## 618                1          NA                1                1       1      0
```

Step 3: Is imputation reasonable to use?

We can see that we have **16 rows with NA values in Bare_Nuclei column**. But is imputation possible here? If **NAs constitute less than 5% of the data, then imputation is reasonable to use**. Let's find out if that is the case here - check the share that NA rows constitute from all rows:

```
nrow(na)/nrow(data)*100
```

```
## [1] 2.288984
```

Rows with NA values are 2.29% of our data - this is less than 5%, so we can continue with imputation

Step 5: Mean Imputation

I will try both imputation approaches for this task. Let's start with **mean imputation**.

We duplicate our data set

```
#create a new data set
data_mean_imp <- data

#imputation
data_mean_imp$Bare_Nuclei[is.na(data_mean_imp$Bare_Nuclei)] <- round(mean(data_mean_imp$Bare_Nuclei, na
data_mean_imp[24:41,]
```

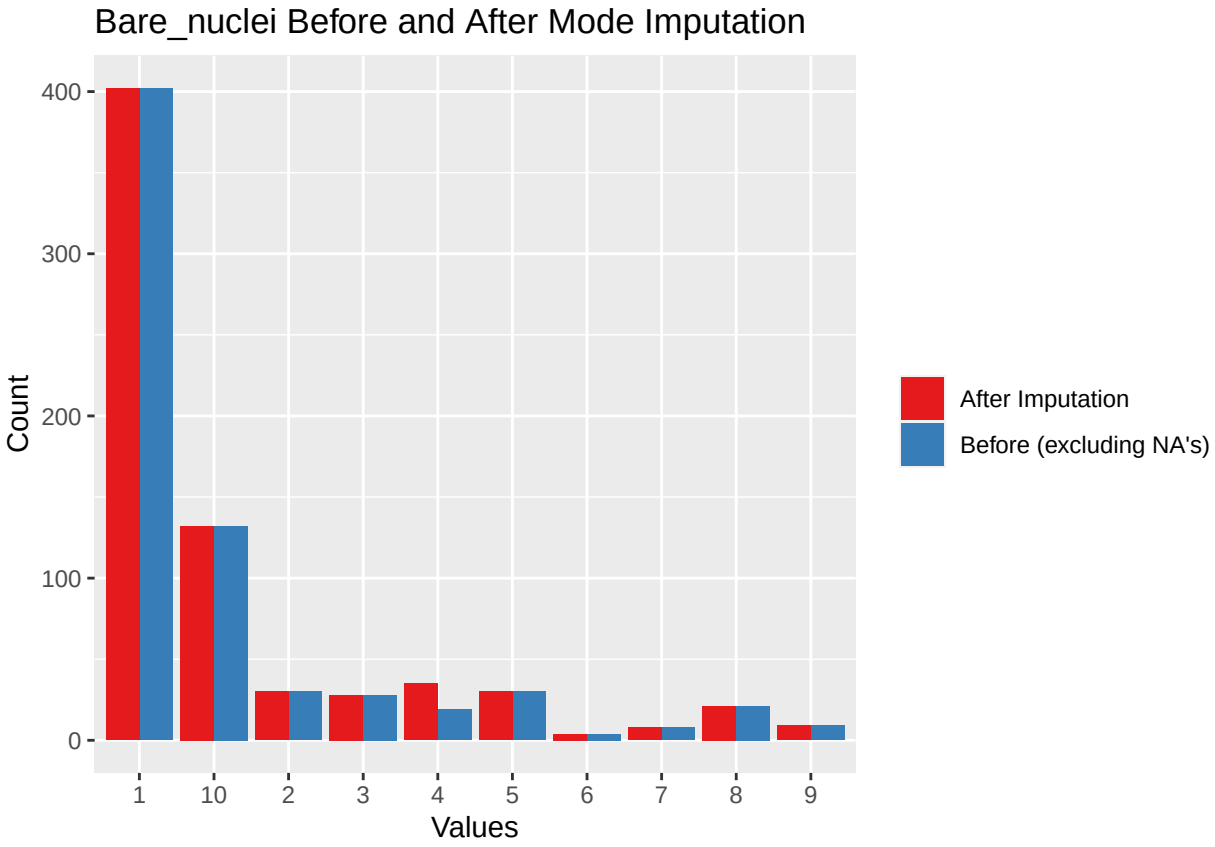
```
##         ID Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 24 1057013               8               4                5                 1
## 25 1059552               1               1                1                 1
## 26 1065726               5               2                3                 4
## 27 1066373               3               2                1                 1
## 28 1066979               5               1                1                 1
## 29 1067444               2               1                1                 1
## 30 1070935               1               1                3                 1
## 31 1070935               3               1                1                 1
## 32 1071760               2               1                1                 1
## 33 1072179              10               7                7                 3
## 34 1074610               2               1                1                 2
## 35 1075123               3               1                2                 1
## 36 1079304               2               1                1                 1
## 37 1080185              10              10               10                 8
## 38 1081791               6               2                1                 1
## 39 1084584               5               4                4                 9
## 40 1091262               2               5                3                 3
```

```
## 41 1096800              6              6               6                9
##    Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 24                 2           4               7               3       1     1
## 25                 2           1               3               1       1     0
## 26                 2           7               3               6       1     1
## 27                 1           1               2               1       1     0
## 28                 2           1               2               1       1     0
## 29                 2           1               2               1       1     0
## 30                 2           1               1               1       1     0
## 31                 1           1               2               1       1     0
## 32                 2           1               3               1       1     0
## 33                 8           5               7               4       3     1
## 34                 2           1               3               1       1     0
## 35                 2           1               2               1       1     0
## 36                 2           1               2               1       1     0
## 37                 6           1               8               9       1     1
## 38                 1           1               7               1       1     0
## 39                 2          10               5               6       1     1
## 40                 6           7               7               5       1     1
## 41                 6           4               7               8       1     0
```

As we can see, **the missing values have been imputed with a mean of 3.55, which we rounded to 4**. We did the rounding, since the 1 to 10 values for the predictors are more like factors, and having a continuous factor value would be incorrect.

Has the distribution of the responses for the Bare_Nuclei column changed much due to imputation? Let's visualize responses before and after imputation to check:

```r
#create indicators of missing or imputing vals data
Scales <- c(rep("Before (excluding NA's)", 10), rep("After Imputation", 10))

#use our 1-10 values as factors twcie to compare both sets
Values <- as.factor(rep(names(table(data$Bare_Nuclei[is.na(data$Bare_Nuclei) == FALSE])), 2))

#calculate how often those values appear
Count <- c(as.numeric(table(data$Bare_Nuclei[is.na(data$Bare_Nuclei) == FALSE])),
           as.numeric(table(data_mean_imp$Bare_Nuclei)))

#use all these values as a data frame for the plot
bar_nuclei <- data.frame(Scales, Values, Count)

#finally, build the plot
ggplot(bar_nuclei, aes(Values, Count, fill = Scales)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.title = element_blank())+
  ggtitle('Bare_nuclei Before and After Mode Imputation')
```



As we can see, due to imputation the number of responses '4' has incrfeased, and is almost double than what it was

Has the mean value of the column changed a lot? Let's see:

```
#mean before imputation
mean(data$Bare_Nuclei[is.na(data$Bare_Nuclei) == FALSE])
```

```
## [1] 3.544656
```

```
#mean after imputation
mean(data_mean_imp$Bare_Nuclei)
```

```
## [1] 3.555079
```

Probably due to the fact that NAs only contributed to 2.9% of the data, the mean after imputation has not changed significantly - it is 3.555.

But '4', which we imputed, is in the lower half of the 1-10 scale describing Bare Nuclei. What does this mean for our data - what bias are we adding?

I found out that bare nuclei is a used to describe nuclei not surrounded by cytoplasm, and that **increased bare nuclei is often associated with malignant tumors**. So, by replacing the missing values with mean ones, that are actually in the lower half of 1-10 scale, we might be making our data set slightly more positive, and possibly even risking some patients to whom those missing values refer - what if due to those low imputed values the model would consider their tumor as benign and make a mistake?

The bias introduced by mean imputation is obvious - and with this example it is clear why this is one of the main drawbacks of mean imputation, despite the fact that it was easy to perform.

Let's see how **mode imputation** would change the picture.

Step 6: Mode Imputation

Let's do mode imputation.

First, let's find the mode: we get all values present in the column other than the missing ones, get the frequency with which they appear (tabulate function), and then choose the value with the highest frequency of appearance in the column as our **mode**:

```
#values other than the missing ones in the column:
val <- unique(data$Bare_Nuclei[!is.na(data$Bare_Nuclei)])

#Mode of Bare_nuclei
mode_barenuclei <- val[which.max(tabulate(match(data$Bare_Nuclei, val)))]
mode_barenuclei
```

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

The mode for bare_nuclei is 1, which is lower than the mean of 3.545. So, in a way, by imputing values of 1 we are also making the set more positive (if lower bare nuclei are associated with lower risk of cancer).

Let's impute the values:

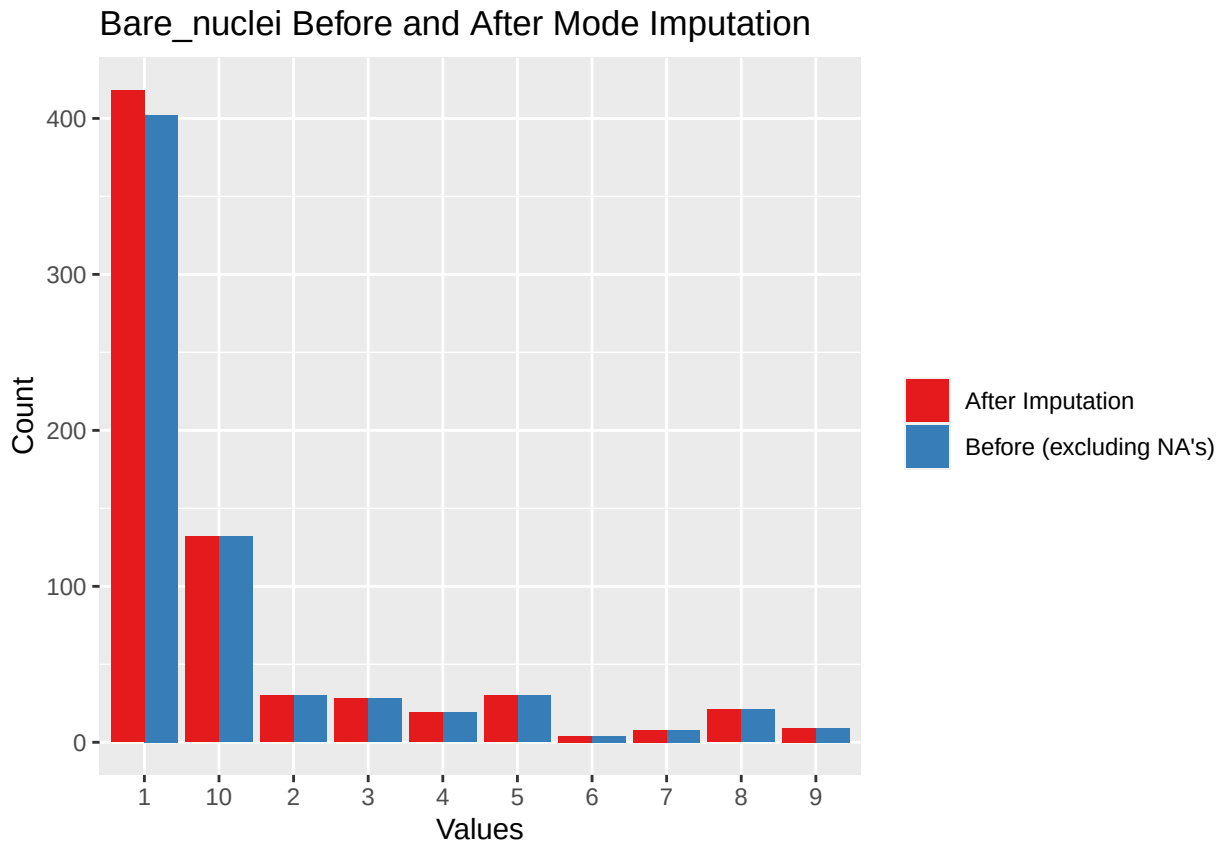
```
#duplicate the data set
data_mode_imp <- data

#imputation
data_mode_imp$Bare_Nuclei[is.na(data_mode_imp$Bare_Nuclei)] <- mode_barenuclei
data_mode_imp[24:41,]
```

```
##         ID Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 24 1057013               8               4                5                 1
## 25 1059552               1               1                1                 1
## 26 1065726               5               2                3                 4
## 27 1066373               3               2                1                 1
## 28 1066979               5               1                1                 1
## 29 1067444               2               1                1                 1
## 30 1070935               1               1                3                 1
## 31 1070935               3               1                1                 1
## 32 1071760               2               1                1                 1
## 33 1072179              10               7                7                 3
## 34 1074610               2               1                1                 2
## 35 1075123               3               1                2                 1
## 36 1079304               2               1                1                 1
## 37 1080185              10              10               10                 8
## 38 1081791               6               2                1                 1
## 39 1084584               5               4                4                 9
## 40 1091262               2               5                3                 3
## 41 1096800               6               6                6                 9
##    Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 24                 2           1               7               3       1     1
## 25                 2           1               3               1       1     0
## 26                 2           7               3               6       1     1
## 27                 1           1               2               1       1     0
## 28                 2           1               2               1       1     0
## 29                 2           1               2               1       1     0
## 30                 2           1               1               1       1     0
## 31                 1           1               2               1       1     0
## 32                 2           1               3               1       1     0
```

```
## 33                8           5                7                4       3      1
## 34                2           1                3                1       1      0
## 35                2           1                2                1       1      0
## 36                2           1                2                1       1      0
## 37                6           1                8                9       1      1
## 38                1           1                7                1       1      0
## 39                2          10                5                6       1      1
## 40                6           7                7                5       1      1
## 41                6           1                7                8       1      0
```

As we can see, NA values have been replaced with 1 now. Have we changes the data significantly? Let's check visually. Since we now have rounded values only, we can check the frequency of responses for each number on the scale:

```
#create indicators of missing or imputing vals data
Scales <- c(rep("Before (excluding NA's)", 10), rep("After Imputation", 10))

#use our 1-10 values as factors twcie to compare both sets
Values <- as.factor(rep(names(table(data$Bare_Nuclei[is.na(data$Bare_Nuclei) == FALSE])), 2))

#calculate how often those values appear
Count <- c(as.numeric(table(data$Bare_Nuclei[is.na(data$Bare_Nuclei) == FALSE])),
           as.numeric(table(data_mode_imp$Bare_Nuclei)))

#use all these values as a data frame for the plot
bar_nuclei <- data.frame(Scales, Values, Count)

#finally, build the plot
ggplot(bar_nuclei, aes(Values, Count, fill = Scales)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.title = element_blank())+
  ggtitle('Bare_nuclei Before and After Mode Imputation')
```



As we can see, with mode imputation we only affected the first number on response scale, 1.

Mode/Mean Imputation - Thoughts

Mode and mean imputations were relatively easy to perform and understand, which is probably the main benefit of such methods. However, as mentioned above, although both mode and mean imputations have not changed our data set significantly in general, they **might have changed individuals' data enough to make a more positive/negative prediction**. Our data set is about cancer diagnostics, and such imputations can lead to great risks of missing someone's malignant tumor, for example, since we are suggesting to use mean/mode values of a parameters only based on the mean values, without considering any other factors. If Bare nuclei is a significant predictor of breast cancer that affects 50%+ of our 'malignant or benign' prediction for the patient, replacing it with such an extreme value of 1, as we did with mode imputation, might be dangerous - because of our imputation, the patient might be declared as healthy when they really are not.

So, let's see what other imputation methods intail, and how they would compare to the simple methods we have just used.

Question 2

Task: Use regression to impute values for the missing data.

Step 1: Adjust the data set

First, before performing linear regression, we need to adjust the data set.

For regression imputation, we will use **column with missing values as the response, and other attributes (excluding the actual response) as predictors**. So let's remove the response and id columns

and create a new data set for this purpose:

```
#get numbers of rows with NA:
narows <- which(is.na(data$Bare_Nuclei)==T, arr.ind=T)

#create new df without response variable and without NA rows
data_mod <- data[-narows,2:10]
head(data_mod)
```

```
##   Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 1               5               1                1                 1
## 2               5               4                4                 5
## 3               3               1                1                 1
## 4               6               8                8                 1
## 5               4               1                1                 3
## 6               8              10               10                 8
##   Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses
## 1                 2           1               3                1       1
## 2                 7          10               3                2       1
## 3                 2           2               3                1       1
## 4                 3           4               3                7       1
## 5                 2           1               3                1       1
## 6                 7          10               9                7       1
```

The new data set is ready, and we can do linear regression now.

Step 2: Liner Regression

First, let's try running a basic linear regression to see how many significant variables we have. If not all are significant, we will use **stepwise regression**, since it is a simple method which would be enough for the purpose of this task - to narrow down the variables used for the model.

As always, I will use caret package so that cross-validation is automatically performed:

```
set.seed(1)
#5 fold cross validation with 10 repetitions
data_ctrl <- trainControl(method = "repeatedcv", number = 5, repeats=10)
#use bare_nuclei as the response variable
reg <- train(Bare_Nuclei~.,
                     data=data_mod,
                     trControl=data_ctrl,
                     method="lm")
summary(reg)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.7316 -0.9426 -0.3002  0.6725  8.6998
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -0.616652   0.194975  -3.163  0.00163 **
## Clump_Thickness    0.230156   0.041691   5.521 4.83e-08 ***
## Uniformity_Size   -0.067980   0.076170  -0.892  0.37246
```

```
## Uniformity_Shape   0.340442   0.073420   4.637 4.25e-06 ***
## Marginal_Adhesion  0.339705   0.045919   7.398 4.13e-13 ***
## Single_Epith_Size  0.090392   0.062541   1.445  0.14883
## Bland_Chromatin    0.320577   0.059047   5.429 7.91e-08 ***
## Normal_Nucleoli    0.007293   0.044486   0.164  0.86983
## Mitoses           -0.075230   0.059331  -1.268  0.20524
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.274 on 674 degrees of freedom
## Multiple R-squared:  0.615,  Adjusted R-squared:  0.6104
## F-statistic: 134.6 on 8 and 674 DF,  p-value: < 2.2e-16
```

As we can see, only **3 variables out of 8 are significant with simple linear regression, and R2 of the cross-validated model is 61.5%**.

Hence, **we should use stepwise regression to improve the model and help us select the variables**. I will perform backward stepwise for several reasons:

- we don't have an extreme case where the number of predictors is larger than the number of rows
- backward has an advantage of starting with a full model (not null as with forward direction), so to start it will simultaneously consider all the variables to help deal with collinearity - backward is more likely to keep more variables in conflicting case, and we don't want to miss anything important when predicting cancer diagnostics.

Step 3: Stepwise regression for variable selection

```
set.seed(1)
data_ctrl <- trainControl(method = "repeatedcv", number = 5, repeats=10)
stepwise <- train(Bare_Nuclei~.,
                  data=data_mod,
                  trControl=data_ctrl,
                  method="lmStepAIC",
                  direction = "backward", trace=F)
summary(stepwise)
```

```
##
## Call:
## lm(formula = .outcome ~ Clump_Thickness + Uniformity_Shape +
##     Marginal_Adhesion + Bland_Chromatin, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.8115 -0.9531 -0.3111  0.6678  8.6889
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -0.53601    0.17514  -3.060   0.0023 **
## Clump_Thickness    0.22617    0.04121   5.488 5.75e-08 ***
## Uniformity_Shape   0.31729    0.05086   6.239 7.76e-10 ***
## Marginal_Adhesion  0.33227    0.04431   7.499 2.03e-13 ***
## Bland_Chromatin    0.32378    0.05606   5.775 1.17e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 2.274 on 678 degrees of freedom
## Multiple R-squared:  0.6129, Adjusted R-squared:  0.6107
## F-statistic: 268.4 on 4 and 678 DF,  p-value: < 2.2e-16
```

The same 4 variables are significant: Clump_Thickness, Uniformity_Shape, MArginal_Adhesion and Bland_Chromatin - let's use them to build and cross-validate our final linear regression model that predicts Bare_nuclei values:

```
set.seed(1)

data_ctrl <- trainControl(method = "repeatedcv", number = 5, repeats=10)

model <- train(Bare_Nuclei ~ Clump_Thickness + Uniformity_Shape + Marginal_Adhesion + Bland_Chromatin,
                    data=data_mod,
                    trControl=data_ctrl,
                    method="lm")
summary(model)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.8115 -0.9531 -0.3111  0.6678  8.6889
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -0.53601    0.17514  -3.060   0.0023 **
## Clump_Thickness    0.22617    0.04121   5.488 5.75e-08 ***
## Uniformity_Shape   0.31729    0.05086   6.239 7.76e-10 ***
## Marginal_Adhesion  0.33227    0.04431   7.499 2.03e-13 ***
## Bland_Chromatin    0.32378    0.05606   5.775 1.17e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.274 on 678 degrees of freedom
## Multiple R-squared:  0.6129, Adjusted R-squared:  0.6107
## F-statistic: 268.4 on 4 and 678 DF,  p-value: < 2.2e-16
```

As we can see, our final model has **R2 of 61.3% after cross-validation and 4 significant predictors**. We can now use it to predict the missing values.

Step 4: Predicting Missing Values

Let's use the final model to predict the missing values in bare_nuclei column:

```
set.seed(1)
bare_nuclei_pred <- predict(model, newdata=data[narows,])
bare_nuclei_pred
```

```
##        24        41       140       146       159       165       236       250
## 5.4585352 7.9816106 0.9872832 1.6218560 0.9807851 2.2157441 2.7152652 1.7634059
##       276       293       295       298       316       322       412       618
## 2.0741942 6.0866099 0.9872832 2.5265324 5.2438347 1.7634059 0.9872832 0.6634986
```

We have our predictions, but they are integers, and some of them are out of range (lower than 1) - let's round

them all and impute rounded values into the data frame:

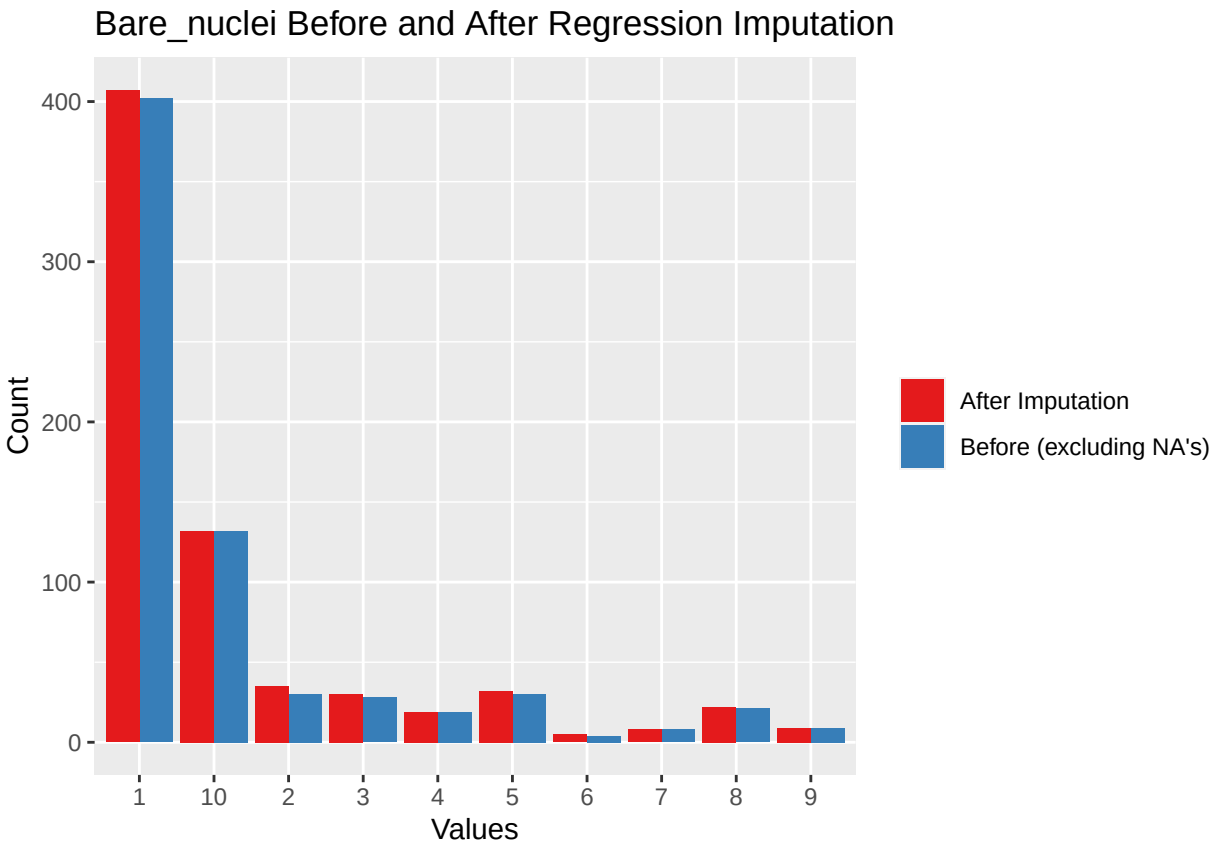
```
data_reg_imp <- data
data_reg_imp[narows,]$Bare_Nuclei <- round(bare_nuclei_pred)
summary(data_reg_imp$Bare_Nuclei)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   3.528   6.000  10.000
```

As we can see, after rounding predicted values, all the values are within the 1-10 range.

Has the data set changed a lot? Let's visualize:

```
#create indicators of missing or imputing vals data
Scales <- c(rep("Before (excluding NA's)", 10), rep("After Imputation", 10))

#use our 1-10 values as factors twcie to compare both sets
Values <- as.factor(rep(names(table(data$Bare_Nuclei[is.na(data$Bare_Nuclei) == FALSE])), 2))

#calculate how often those values appear
Count <- c(as.numeric(table(data$Bare_Nuclei[is.na(data$Bare_Nuclei) == FALSE])),
           as.numeric(table(data_reg_imp$Bare_Nuclei)))

#use all these values as a data frame for the plot
bar_nuclei <- data.frame(Scales, Values, Count)

#finally, build the plot
ggplot(bar_nuclei, aes(Values, Count, fill = Scales)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.title = element_blank())+
  ggtitle('Bare_nuclei Before and After Regression Imputation')
```



As we can see, unlike with mode imputation, the distribution of added values with imputation is much better - and probably higher in prediction quality.

Regression Imputation - Thoughts

Although this method was harder to perform than the mode/mean imputation, it was definitely worth it - as we can see on the graph above, regression has helped us not only add a number of same values to the set, but to add to each of the numbers on 1-10 scale with accordance to other predictors values.

Thanks to this, we have more or less real values, which are at least based on something else - other variables in this case, so we can be more certain that when it comes to using the set for class prediction.

Question 3

Task: Use regression with perturbation to impute values for the missing data.

Step 1: Perturb the predictions

First, let's perturb the predicted values using a random normal distribution: we will use predicted values as the mean, and the standard deviation of predicted values as the standard deviation:

```
set.seed(1)
pert <- rnorm(nrow(data[narows,]),
              mean=bare_nuclei_pred,
              sd=sd(bare_nuclei_pred))
pert
```

```
##  [1]  4.0777220  8.3863924 -0.8545876  5.1381323  1.7070775  0.4072891
```

```
##  [7]  3.7896436  3.3908019  3.3433164  5.4134808  4.3195118  3.3858147
## [13]  3.8745124 -3.1181778  3.4668265  0.5644572
```

As we can see, the values are continuous and need rounding, and there are also some negative values. Let's bound the negative values to positive and also round the values:

```
pert <- abs(round(pert))
pert
```

```
##  [1] 4 8 1 5 2 0 4 3 3 5 4 3 4 3 3 1
```

```
summary(pert)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   2.750   3.000   3.312   4.000   8.000
```

We can see that we have one 0 value, which is our of range. Let's replace it with the closest value - 1:

```
pert[which.min(pert)]=1
pert
```

```
##  [1] 4 8 1 5 2 1 4 3 3 5 4 3 4 3 3 1
```

Step 2: Impute values

Now let's impute the values that we got with perturbation:

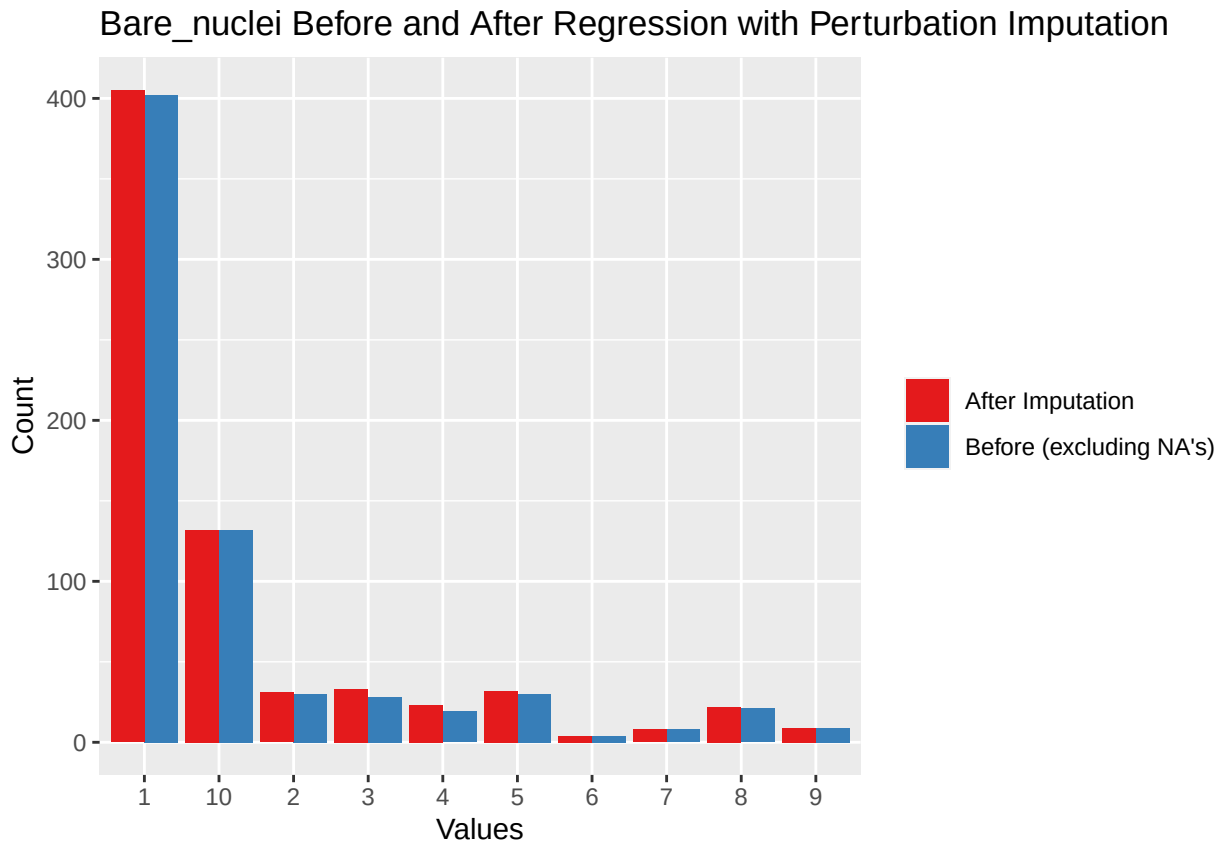
```
data_pert_imp <- data
data_pert_imp[narows,]$Bare_Nuclei <- pert
summary(data_pert_imp$Bare_Nuclei)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   3.541   5.500  10.000
```

The mean is quite close to the mean we had in the initial data.

Let's visualize the changes in the data:

```
#create indicators of missing or imputing vals data
Scales <- c(rep("Before (excluding NA's)", 10), rep("After Imputation", 10))

#use our 1-10 values as factors twcie to compare both sets
Values <- as.factor(rep(names(table(data$Bare_Nuclei[is.na(data$Bare_Nuclei) == FALSE])), 2))

#calculate how often those values appear
Count <- c(as.numeric(table(data$Bare_Nuclei[is.na(data$Bare_Nuclei) == FALSE])),
           as.numeric(table(data_pert_imp$Bare_Nuclei)))

#use all these values as a data frame for the plot
bar_nuclei <- data.frame(Scales, Values, Count)

#finally, build the plot
ggplot(bar_nuclei, aes(Values, Count, fill = Scales)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.title = element_blank())+
  ggtitle('Bare_nuclei Before and After Regression with Perturbation Imputation')
```



Looks like the most values were added to the mid-range responses like 3, 4 and 5, but in general the picture of our data has not changed significantly, so imputation with perturbation definitely has not altered the trends in the data set.

Regression with Perturbation - Thoughts

Perturbation is definitely useful to correct the drawback of regression imputation - its inability to capture all the variability of the data. The values we got with perturbation do differ from what we got with regression, and perhaps there was more variability for the mid-range values (3,4,5..) - but in general this method also helped us spread imputed values across the whole 1-10 range, not just add lots of same values, which can be beneficial when the share of data with NA values is closer to 5%.

In general, regression and regression with perturbation imputation method appear to be much better than mean/mode imputation and at first glance show better results on our set.

Question 4

Task: (Optional) Compare the results and quality of classification models (e.g., SVM, KNN) build using:

- (1) the data sets from questions 1,2,3;
- (2) the data that remains after data points with missing values are removed
- (3) the data set when a binary variable is introduced to indicate missing values.

[KNN] Step 1: Sampling

We have 4 data sets from tasks 1-3, so we will be trying 4 knn models.

First, **we need to split the data into training and test sets**. Let's split it with 75-25 ratio:

1. Data set with mean imputation:

```
set.seed(1)

#find the number of rows for test set
index <- round(nrow(data_mean_imp)*0.25,digits=0)

#randomly sample the rows which will go to the test set (20%)
test_sample <- sample(1:nrow(data_mean_imp), index)

#create the test set - exclude id column
mean_test <- data_mean_imp[test_sample,2:11]
head(mean_test)
```

```
##     Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 679               1               1                1                 1
## 129               8               3                5                 4
## 509               5               1                1                 1
## 471               3               1                1                 1
## 299               8               2                1                 1
## 270               1               1                1                 1
##     Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 679                 2           1               1               1       1     0
## 129                 5          10               1               6       2     1
## 509                 2           1               1               1       1     0
## 471                 1           1               2               1       1     0
## 299                 5           1               1               1       1     0
## 270                 2           1               3               1       1     0
```

```
#create the train set
mean_train <- data_mean_imp[-test_sample,2:11]
head(mean_train)
```

```
##   Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 2               5               4                4                 5
## 3               3               1                1                 1
## 4               6               8                8                 1
## 5               4               1                1                 3
## 6               8              10               10                 8
## 7               1               1                1                 1
##   Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 2                 7          10               3               2       1     0
## 3                 2           2               3               1       1     0
## 4                 3           4               3               7       1     0
## 5                 2           1               3               1       1     0
## 6                 7          10               9               7       1     1
## 7                 2          10               3               1       1     0
```

```
#CHECK PROPORTIONS:

#whole set
prop.table(table(data_mean_imp$Class))
```

```
##
##         0         1
## 0.6552217 0.3447783
```

```
#test set
prop.table(table(mean_test$Class))
```

```
##
##         0         1
## 0.6685714 0.3314286
```

```
#train set
prop.table(table(mean_train$Class))
```

```
##
##         0         1
## 0.6507634 0.3492366
```

The train and test sets for mean imputation data have been created, and the proportion of classes is similar throughout all the sets, so we can continue to the other 3 data set and use **the same sample for each set**, so that we can fairly compare the results:

2. Data set with mode imputation:

```
#create the test set - exclude id column
mode_test <- data_mode_imp[test_sample,2:11]
head(mode_test)
```

```
##     Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 679               1               1                1                 1
## 129               8               3                5                 4
## 509               5               1                1                 1
## 471               3               1                1                 1
## 299               8               2                1                 1
## 270               1               1                1                 1
##     Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 679                 2           1               1               1       1     0
## 129                 5          10               1               6       2     1
## 509                 2           1               1               1       1     0
## 471                 1           1               2               1       1     0
## 299                 5           1               1               1       1     0
## 270                 2           1               3               1       1     0
```

```
#create the train set
mode_train <- data_mode_imp[-test_sample,2:11]
head(mode_train)
```

```
##   Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 2               5               4                4                 5
## 3               3               1                1                 1
## 4               6               8                8                 1
## 5               4               1                1                 3
## 6               8              10               10                 8
## 7               1               1                1                 1
##   Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 2                 7          10               3               2       1     0
## 3                 2           2               3               1       1     0
## 4                 3           4               3               7       1     0
## 5                 2           1               3               1       1     0
## 6                 7          10               9               7       1     1
## 7                 2          10               3               1       1     0
```

3. Data set with regression imputation:

```
#create the test set - exclude id column
reg_test <- data_reg_imp[test_sample,2:11]
head(reg_test)
```

```
##     Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 679               1               1                1                 1
## 129               8               3                5                 4
## 509               5               1                1                 1
## 471               3               1                1                 1
## 299               8               2                1                 1
## 270               1               1                1                 1
##     Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 679                 2           1               1               1       1     0
## 129                 5          10               1               6       2     1
## 509                 2           1               1               1       1     0
## 471                 1           1               2               1       1     0
## 299                 5           1               1               1       1     0
## 270                 2           1               3               1       1     0
```

```
#create the train set
reg_train <- data_reg_imp[-test_sample,2:11]
head(reg_train)
```

```
##   Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 2               5               4                4                 5
## 3               3               1                1                 1
## 4               6               8                8                 1
## 5               4               1                1                 3
## 6               8              10               10                 8
## 7               1               1                1                 1
##   Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 2                 7          10               3               2       1     0
## 3                 2           2               3               1       1     0
## 4                 3           4               3               7       1     0
## 5                 2           1               3               1       1     0
## 6                 7          10               9               7       1     1
## 7                 2          10               3               1       1     0
```

4. Data set with regression+perturbation imputation:

```
#create the test set - exclude id column
pert_test <- data_pert_imp[test_sample,2:11]
head(pert_test)
```

```
##     Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 679               1               1                1                 1
## 129               8               3                5                 4
## 509               5               1                1                 1
## 471               3               1                1                 1
## 299               8               2                1                 1
## 270               1               1                1                 1
##     Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 679                 2           1               1               1       1     0
## 129                 5          10               1               6       2     1
## 509                 2           1               1               1       1     0
```

```
## 471                1           1               2               1       1     0
## 299                5           1               1               1       1     0
## 270                2           1               3               1       1     0
```

```
#create the train set
pert_train <- data_pert_imp[-test_sample,2:11]
head(pert_train)
```

```
##   Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 2               5               4                4                 5
## 3               3               1                1                 1
## 4               6               8                8                 1
## 5               4               1                1                 3
## 6               8              10               10                 8
## 7               1               1                1                 1
##   Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 2                 7          10               3               2       1     0
## 3                 2           2               3               1       1     0
## 4                 3           4               3               7       1     0
## 5                 2           1               3               1       1     0
## 6                 7          10               9               7       1     1
## 7                 2          10               3               1       1     0
```

[KNN] Step 2: Preprocessing + Building the Models

Now that we have the sets, we can build the models. I will use caret so as to validate the models along the way and get a more real accuracy. We will train model on the train set and cross-validate them using repeated cross-validation with 5 folds and 10 repetitions.

KNN requires normalized/scaled data, so I will also include preprocessing to standardize the values.

As for k values, I will use k from 1 to 50 and then compare the accuracy of the cross validated model to choose the best k.

1. Mean imputation KNN:

```
set.seed(1)
mean_knn <- train(as.factor(Class)~.,#Class - response values
                      mean_train,
                      method = "knn",
                      preProcess = c("center", "scale"), #standardize the data
                      tuneGrid = data.frame(k = c(1:50)), #use k values from 1 to 50
                      trControl = trainControl(
                        method = "repeatedcv", #repeated cross validation
                        number = 5, #5 k-folds
                        repeats = 10)) #repeating cross-validation 10 times, each time the folds are sp

mean_knn
```

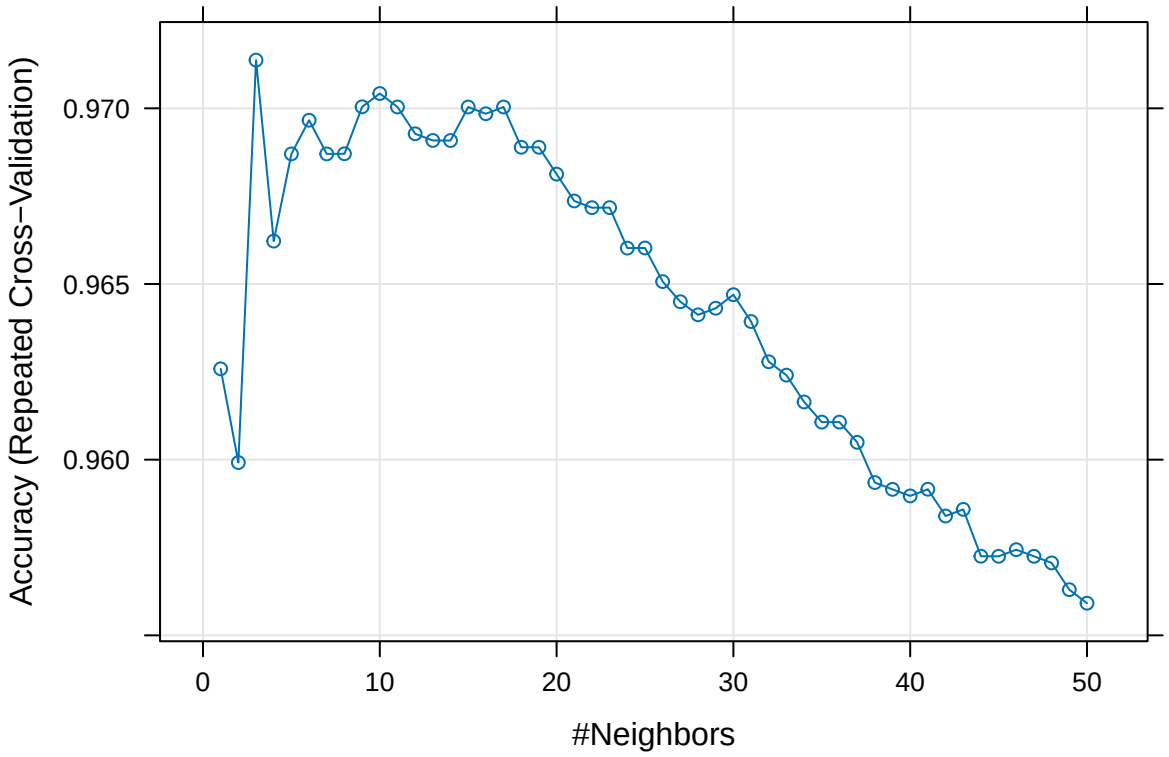
```
## k-Nearest Neighbors
##
## 524 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
```

```
## Summary of sample sizes: 419, 420, 419, 419, 419, 419, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9625853  0.9171501
##    2  0.9599186  0.9113545
##    3  0.9713731  0.9371285
##    4  0.9662209  0.9256871
##    5  0.9687026  0.9311998
##    6  0.9696605  0.9332822
##    7  0.9687027  0.9310980
##    8  0.9687063  0.9311428
##    9  0.9700433  0.9341084
##   10  0.9704207  0.9349498
##   11  0.9700378  0.9340630
##   12  0.9692759  0.9323768
##   13  0.9690854  0.9320421
##   14  0.9690836  0.9320828
##   15  0.9700379  0.9341856
##   16  0.9698456  0.9337471
##   17  0.9700379  0.9341053
##   18  0.9688913  0.9315315
##   19  0.9688913  0.9315543
##   20  0.9681275  0.9297957
##   21  0.9673638  0.9281014
##   22  0.9671715  0.9276258
##   23  0.9671733  0.9276298
##   24  0.9660214  0.9250659
##   25  0.9660250  0.9250424
##   26  0.9650689  0.9228843
##   27  0.9644957  0.9215645
##   28  0.9641220  0.9207650
##   29  0.9643107  0.9211528
##   30  0.9646953  0.9220050
##   31  0.9639315  0.9202942
##   32  0.9627868  0.9176673
##   33  0.9624059  0.9167954
##   34  0.9616421  0.9150481
##   35  0.9610688  0.9137394
##   36  0.9610688  0.9137597
##   37  0.9604937  0.9124522
##   38  0.9593491  0.9098621
##   39  0.9591549  0.9094150
##   40  0.9589680  0.9089677
##   41  0.9591585  0.9094365
##   42  0.9583966  0.9076992
##   43  0.9585853  0.9081545
##   44  0.9572519  0.9050807
##   45  0.9572501  0.9050958
##   46  0.9574388  0.9055227
##   47  0.9572501  0.9050738
##   48  0.9570614  0.9046117
##   49  0.9562995  0.9028802
##   50  0.9559149  0.9020078
```

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 3.
```

As seen in the output, the **best k is 3 for the mean imputation data set**. Let's visualize the results to see how it compares to other k's:

```
plot(mean_knn)
```



All k values give high result on the test set after cross-validation, but k=3 is indeed the best one.

We will later test this model on the test set to get true accuracy, but now let's repeat the process on other data sets with other imputation methods.

2. Mode imputation KNN:

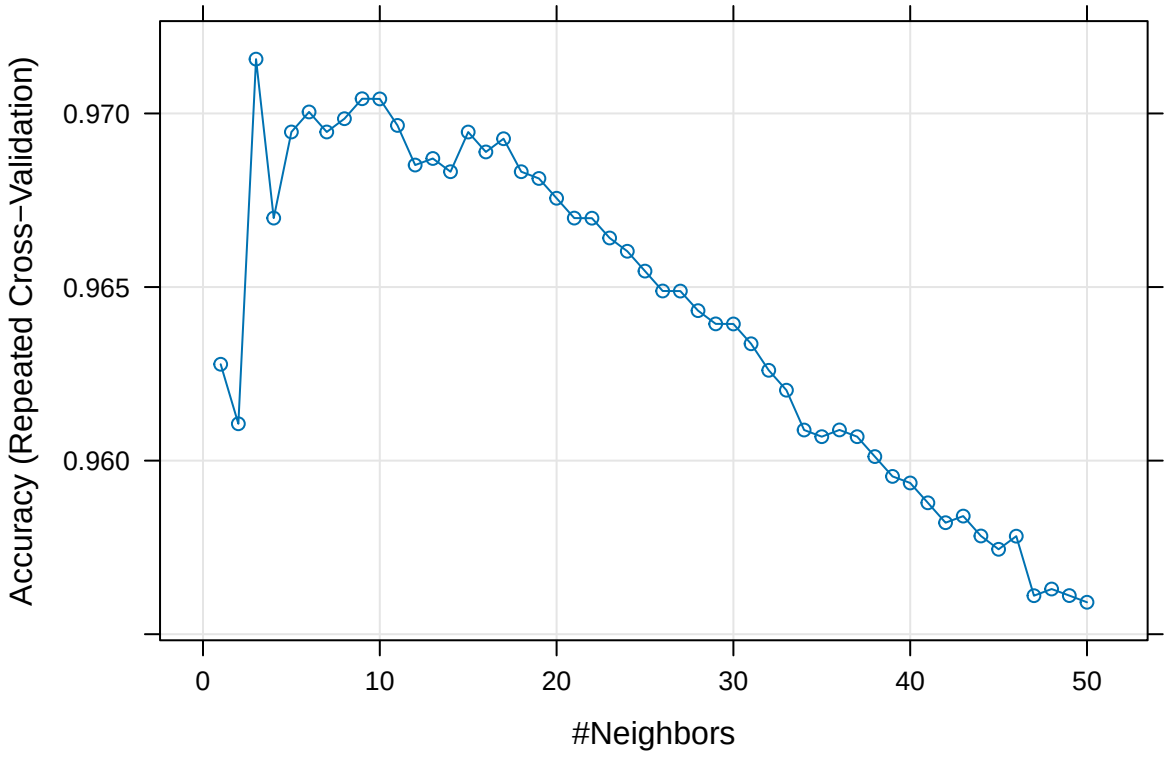
```
set.seed(1)
mode_knn <- train(as.factor(Class)~.,
                  mode_train,
                  method = "knn",
                  preProcess = c("center", "scale"),
                  tuneGrid = data.frame(k = c(1:50)),
                  trControl = trainControl(
                    method = "repeatedcv",
                    number = 5,
                    repeats = 10))

mode_knn
```

```
## k-Nearest Neighbors
##
## 524 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 419, 420, 419, 419, 419, 419, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9627758  0.9175648
##    2  0.9610614  0.9138201
##    3  0.9715654  0.9375451
##    4  0.9669864  0.9273431
##    5  0.9694664  0.9328839
##    6  0.9700433  0.9341103
##    7  0.9694664  0.9328235
##    8  0.9698492  0.9336993
##    9  0.9704243  0.9349534
##   10  0.9704188  0.9349421
##   11  0.9696551  0.9331946
##   12  0.9685140  0.9306982
##   13  0.9687026  0.9311897
##   14  0.9683217  0.9303972
##   15  0.9694646  0.9328813
##   16  0.9688914  0.9315821
##   17  0.9692723  0.9323792
##   18  0.9683217  0.9302611
##   19  0.9681276  0.9298253
##   20  0.9675580  0.9285457
##   21  0.9669865  0.9272864
##   22  0.9669829  0.9272215
##   23  0.9664133  0.9259950
##   24  0.9660305  0.9251362
##   25  0.9654591  0.9237955
##   26  0.9648858  0.9225169
##   27  0.9648840  0.9224776
##   28  0.9643198  0.9212363
##   29  0.9639388  0.9203075
##   30  0.9639370  0.9202732
##   31  0.9633656  0.9190143
##   32  0.9626018  0.9172489
##   33  0.9620286  0.9159484
##   34  0.9608820  0.9133310
##   35  0.9606897  0.9129089
##   36  0.9608839  0.9133608
##   37  0.9606897  0.9129150
##   38  0.9601165  0.9116161
##   39  0.9595450  0.9103137
##   40  0.9593563  0.9098642
##   41  0.9587849  0.9085777
##   42  0.9582117  0.9073063
```

```
##    43  0.9584021  0.9077362
##    44  0.9578289  0.9064043
##    45  0.9574461  0.9055186
##    46  0.9578234  0.9063466
##    47  0.9561091  0.9024385
##    48  0.9563014  0.9028663
##    49  0.9561127  0.9024709
##    50  0.9559204  0.9020155
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 3.
```

Once again, **the best k for mode imputation set is 3, giving 97.16% accuracy on train set after cross-validation**:

```
plot(mode_knn)
```



Looks like knn performs on our data set best with low k values from 3 to 10.

3. Regression imputation KNN:

```
set.seed(1)
reg_knn <- train(as.factor(Class)~.,
                         reg_train,
                         method = "knn",
                         preProcess = c("center", "scale"),
                         tuneGrid = data.frame(k = c(1:50)),
                         trControl = trainControl(
```

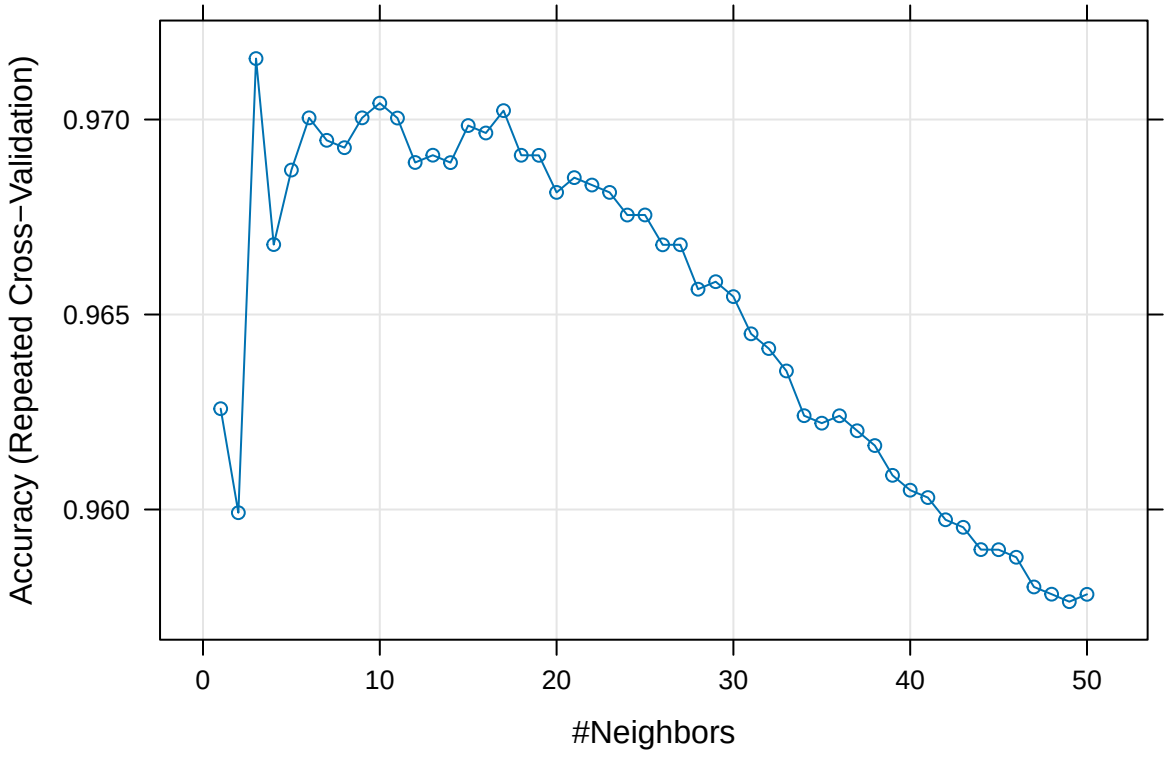
```
                        method = "repeatedcv",
                        number = 5,
                        repeats = 10))

reg_knn
```

```
## k-Nearest Neighbors
##
## 524 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 419, 420, 419, 419, 419, 419, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9625853  0.9171501
##    2  0.9599186  0.9112847
##    3  0.9715636  0.9375315
##    4  0.9667941  0.9269095
##    5  0.9687044  0.9311716
##    6  0.9700433  0.9341211
##    7  0.9694664  0.9328235
##    8  0.9692778  0.9324290
##    9  0.9700433  0.9341084
##   10  0.9704207  0.9349498
##   11  0.9700378  0.9340630
##   12  0.9688985  0.9315670
##   13  0.9690854  0.9320421
##   14  0.9688949  0.9316499
##   15  0.9698456  0.9337580
##   16  0.9696551  0.9333112
##   17  0.9702284  0.9345354
##   18  0.9690836  0.9319818
##   19  0.9690818  0.9319741
##   20  0.9681312  0.9298213
##   21  0.9685104  0.9307136
##   22  0.9683199  0.9302345
##   23  0.9681312  0.9298400
##   24  0.9675507  0.9285739
##   25  0.9675525  0.9285220
##   26  0.9667869  0.9267986
##   27  0.9667888  0.9267759
##   28  0.9656495  0.9242309
##   29  0.9658418  0.9245771
##   30  0.9654590  0.9237030
##   31  0.9645048  0.9215895
##   32  0.9641275  0.9206935
##   33  0.9635524  0.9193846
##   34  0.9624058  0.9167727
##   35  0.9622135  0.9163506
##   36  0.9624040  0.9168088
```

```
##    37  0.9620194  0.9159212
##    38  0.9616403  0.9150639
##    39  0.9608766  0.9133334
##    40  0.9604956  0.9124399
##    41  0.9603069  0.9120330
##    42  0.9597373  0.9107613
##    43  0.9595432  0.9103207
##    44  0.9589717  0.9089956
##    45  0.9589699  0.9089935
##    46  0.9587758  0.9085456
##    47  0.9580139  0.9067968
##    48  0.9578288  0.9063672
##    49  0.9576365  0.9059301
##    50  0.9578252  0.9063570
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 3.
```

The situation does not change for regression imputation set - after cross validation **the best k=3 with accuracy of 97.16%**:

```
plot(reg_knn)
```



4. Regression+perturbation imputation KNN:

```
set.seed(1)
pert_knn <- train(as.factor(Class)~.,
                  pert_train,
```

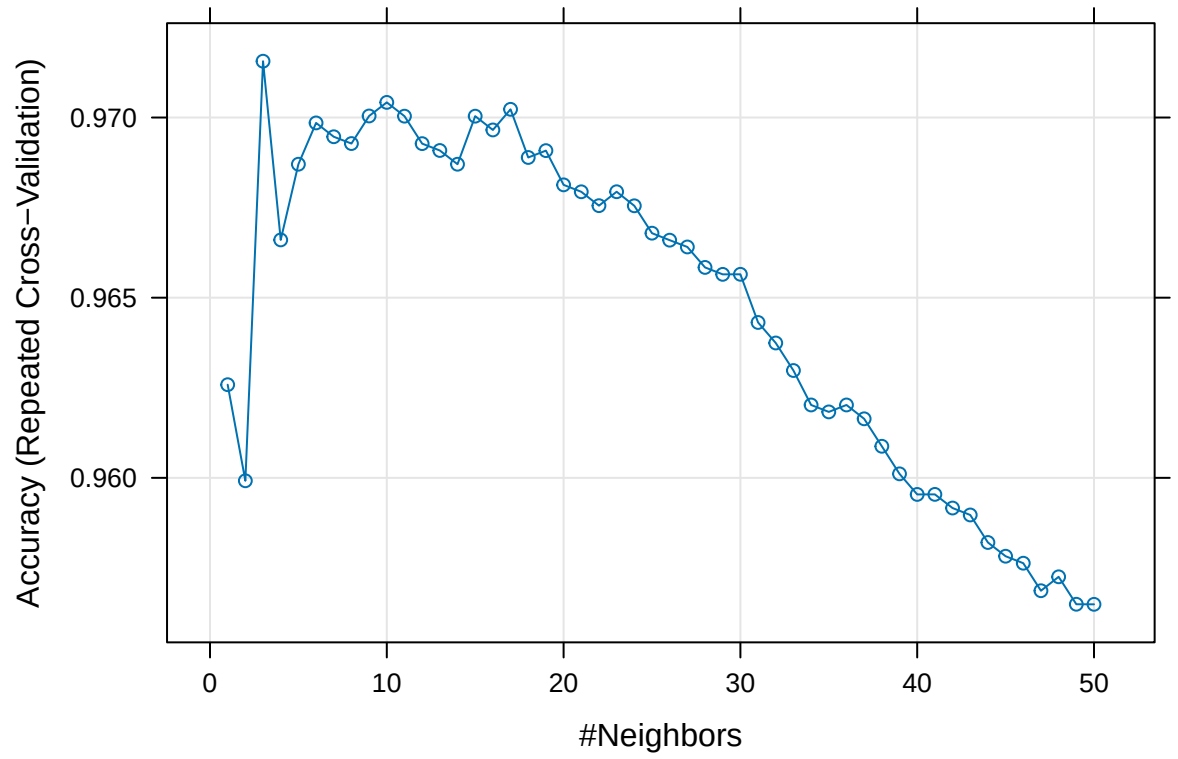
```
                      method = "knn",
                      preProcess = c("center", "scale"),
                      tuneGrid = data.frame(k = c(1:50)),
                      trControl = trainControl(
                        method = "repeatedcv",
                        number = 5,
                        repeats = 10))

pert_knn
```

```
## k-Nearest Neighbors
##
## 524 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 419, 420, 419, 419, 419, 419, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9625853  0.9171501
##    2  0.9599186  0.9112847
##    3  0.9715636  0.9375315
##    4  0.9666036  0.9264896
##    5  0.9687044  0.9311716
##    6  0.9698510  0.9336936
##    7  0.9694664  0.9328235
##    8  0.9692778  0.9324290
##    9  0.9700433  0.9341084
##   10  0.9704207  0.9349498
##   11  0.9700378  0.9340630
##   12  0.9692759  0.9323768
##   13  0.9690854  0.9320421
##   14  0.9687044  0.9312453
##   15  0.9700379  0.9341856
##   16  0.9696551  0.9333112
##   17  0.9702284  0.9345354
##   18  0.9688913  0.9315485
##   19  0.9690818  0.9319741
##   20  0.9681312  0.9298213
##   21  0.9679389  0.9294152
##   22  0.9675543  0.9285065
##   23  0.9679407  0.9294032
##   24  0.9675507  0.9285482
##   25  0.9667906  0.9267998
##   26  0.9665964  0.9263627
##   27  0.9664078  0.9259089
##   28  0.9658399  0.9246356
##   29  0.9656477  0.9241474
##   30  0.9656495  0.9241333
##   31  0.9643125  0.9211562
##   32  0.9637429  0.9198213
```

```
##    33  0.9629791  0.9180850
##    34  0.9620230  0.9159144
##    35  0.9618307  0.9154765
##    36  0.9620230  0.9159363
##    37  0.9616384  0.9150486
##    38  0.9608765  0.9133062
##    39  0.9601110  0.9116002
##    40  0.9595395  0.9102767
##    41  0.9595413  0.9102999
##    42  0.9591622  0.9094371
##    43  0.9589699  0.9090151
##    44  0.9582062  0.9072580
##    45  0.9578252  0.9063923
##    46  0.9576311  0.9059557
##    47  0.9568673  0.9041772
##    48  0.9572537  0.9050508
##    49  0.9564918  0.9033131
##    50  0.9564900  0.9033101
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 3.
```

Once again, after cross-validation of KNN model, **the best k=3 with accuracy of 97.16%**:

```
plot(pert_knn)
```



[KNN] Step 3: Predicting + Estimating the Accuracy

We have found the best KNN tunes for each of the sets and know the accuracy after cross-validation on the training set.

Now, for each data set, let's predict the response class and calculate the test accuracy.

1. Mean imputation KNN Accuracy:

```
set.seed(1)

#make the prediction
predict_mean <- predict(mean_knn,mean_test)

#confusion matrix
cm_mean <- table(Actual=mean_test$Class, Predicted=predict_mean)
cm_mean
```

```
##       Predicted
## Actual   0   1
##      0 113   4
##      1   4  54
```

```
#calculate accuracy
accuracy_mean <- sum(predict_mean == mean_test[,10])/nrow(mean_test)
accuracy_mean
```

```
## [1] 0.9542857
```

Results: With **Mean Imputation** the best **k=3** for cross-validated KNN Model, with **97.14% accuracy on the training set** and **95.43% accuracy on the test set**.

2. Mode imputation KNN Accuracy:

```
set.seed(1)

#make the prediction
predict_mode <- predict(mode_knn,mode_test)

#confusion matrix
cm_mode <- table(Actual=mode_test$Class, Predicted=predict_mode)
cm_mode
```

```
##       Predicted
## Actual   0   1
##      0 113   4
##      1   4  54
```

```
#calculate accuracy
accuracy_mode <- sum(predict_mode == mode_test[,10])/nrow(mode_test)
accuracy_mode
```

```
## [1] 0.9542857
```

Results: With **Mode Imputation** the best **k=3** for cross-validated KNN Model, with **97.16% accuracy on the training set** and **95.43% accuracy on the test set**.

The result is the same with mean imputation set.

3. Regression imputation KNN Accuracy:

```r
set.seed(1)

#make the prediction
predict_reg <- predict(reg_knn,reg_test)

#confusion matrix
cm_reg <- table(Actual=reg_test$Class, Predicted=predict_reg)
cm_reg
```

```
##       Predicted
## Actual   0   1
##      0 113   4
##      1   4  54
```

```r
#calculate accuracy
accuracy_reg <- sum(predict_reg == reg_test[,10])/nrow(reg_test)
accuracy_reg
```

```
## [1] 0.9542857
```

Results: With **Regression Imputation** the best **k=3** for cross-validated KNN Model, with **97.16% accuracy on the training set** and **95.43% accuracy on the test set**.

Once again, the same result.

4. Regression + Perturbation imputation KNN Accuracy:

```r
set.seed(1)

#make the prediction
predict_pert <- predict(pert_knn,pert_test)

#confusion matrix
cm_pert <- table(Actual=pert_test$Class, Predicted=predict_pert)
cm_pert
```

```
##       Predicted
## Actual   0   1
##      0 113   4
##      1   4  54
```

```r
#calculate accuracy
accuracy_pert <- sum(predict_pert == pert_test[,10])/nrow(pert_test)
accuracy_pert
```

```
## [1] 0.9542857
```

Results: With **Regression Imputation** the best **k=3** for cross-validated KNN Model, with **97.16% accuracy on the training set** and **95.43% accuracy on the test set**.

All imputation methods end up giving us KNN models with the same optimal k and the same accuracy on the test set.

Let's use the same 4 data sets to perform SVM and compare the results

[kSVM] Step 1: Preprocessing + Building the Models

For kSVM we will also standardize the data and use repeated cross-validation to estimate accuracy on the training sets. We will use the simple linear kernel and try different C values as a generated value grid of 25

values from 0.01 to 100. **Larger C will make our model choose smaller-margine hyperplanes to get better classification accuracy, and smaller C will lead to a larger margin separation and perhaps will lead to larger error**. Since we are building a model for medical diagnostics, it is probably a good idea to keep the margin larger, so that the model is also universal for future data points (new patients) and not just tailored for this particular data set.

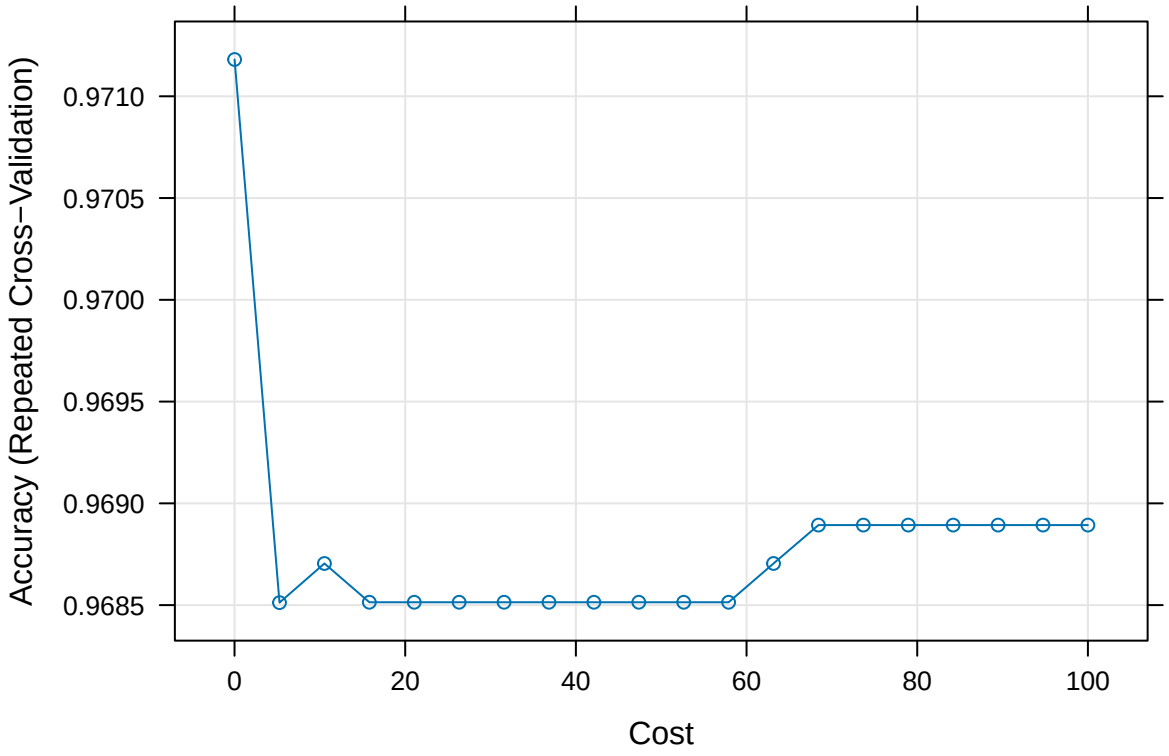
1. Mean imputation SVM:

```
set.seed(1)
mean_svm <- train(as.factor(Class)~.,
                          mean_train,
                          method = "svmLinear",
                          preProcess = c("center", "scale"),
                          tuneGrid = expand.grid(C = seq.int(0.01, 100, length = 20)),
                          trControl = trainControl(
                            method = "repeatedcv",
                            number = 5,
                            repeats = 10))
mean_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 524 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 419, 420, 419, 419, 419, 419, ...
## Resampling results across tuning parameters:
##
##   C           Accuracy   Kappa
##     0.010000  0.9711808  0.9367047
##     5.272632  0.9685122  0.9308459
##    10.535263  0.9687045  0.9312735
##    15.797895  0.9685140  0.9308484
##    21.060526  0.9685140  0.9308484
##    26.323158  0.9685140  0.9308484
##    31.585789  0.9685140  0.9308484
##    36.848421  0.9685140  0.9308484
##    42.111053  0.9685140  0.9308484
##    47.373684  0.9685140  0.9308484
##    52.636316  0.9685140  0.9308484
##    57.898947  0.9685140  0.9308484
##    63.161579  0.9687045  0.9312581
##    68.424211  0.9688932  0.9316758
##    73.686842  0.9688932  0.9316758
##    78.949474  0.9688932  0.9316758
##    84.212105  0.9688932  0.9316758
##    89.474737  0.9688932  0.9316758
##    94.737368  0.9688932  0.9316758
##   100.000000  0.9688932  0.9316758
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

The best C-value after cross-validation for mean imputation data is 0.01, with an accuracy of 97.12%:

```
plot(mean_svm)
```



2. Mode imputation SVM:

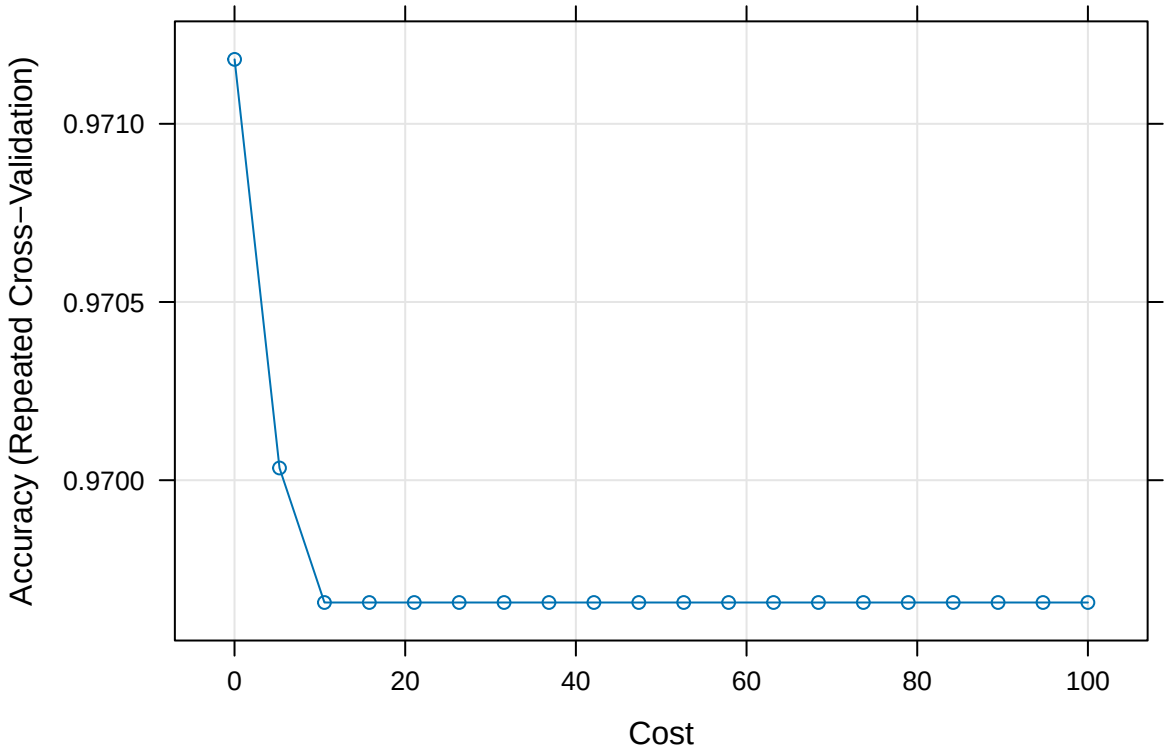
```
set.seed(1)
mode_svm <- train(as.factor(Class)~.,
                          mode_train,
                          method = "svmLinear",
                          preProcess = c("center", "scale"),
                          tuneGrid = expand.grid(C = seq.int(0.01, 100, length = 20)),
                          trControl = trainControl(
                            method = "repeatedcv",
                            number = 5,
                            repeats = 10))
mode_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 524 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 419, 420, 419, 419, 419, 419, ...
```

```
## Resampling results across tuning parameters:
##
##   C           Accuracy   Kappa
##     0.010000  0.9711808  0.9367047
##     5.272632  0.9700343  0.9342126
##    10.535263  0.9696569  0.9333614
##    15.797895  0.9696569  0.9333614
##    21.060526  0.9696569  0.9333614
##    26.323158  0.9696569  0.9333614
##    31.585789  0.9696569  0.9333614
##    36.848421  0.9696569  0.9333614
##    42.111053  0.9696569  0.9333614
##    47.373684  0.9696569  0.9333614
##    52.636316  0.9696569  0.9333614
##    57.898947  0.9696569  0.9333614
##    63.161579  0.9696569  0.9333614
##    68.424211  0.9696569  0.9333614
##    73.686842  0.9696569  0.9333614
##    78.949474  0.9696569  0.9333614
##    84.212105  0.9696569  0.9333614
##    89.474737  0.9696569  0.9333614
##    94.737368  0.9696569  0.9333614
##   100.000000  0.9696569  0.9333614
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

The best C-value after cross-validation for mode imputation data is 0.01, with an accuracy of 97.12% - the same result as with mean imputation:

```
plot(mode_svm)
```



3. Regression imputation SVM:

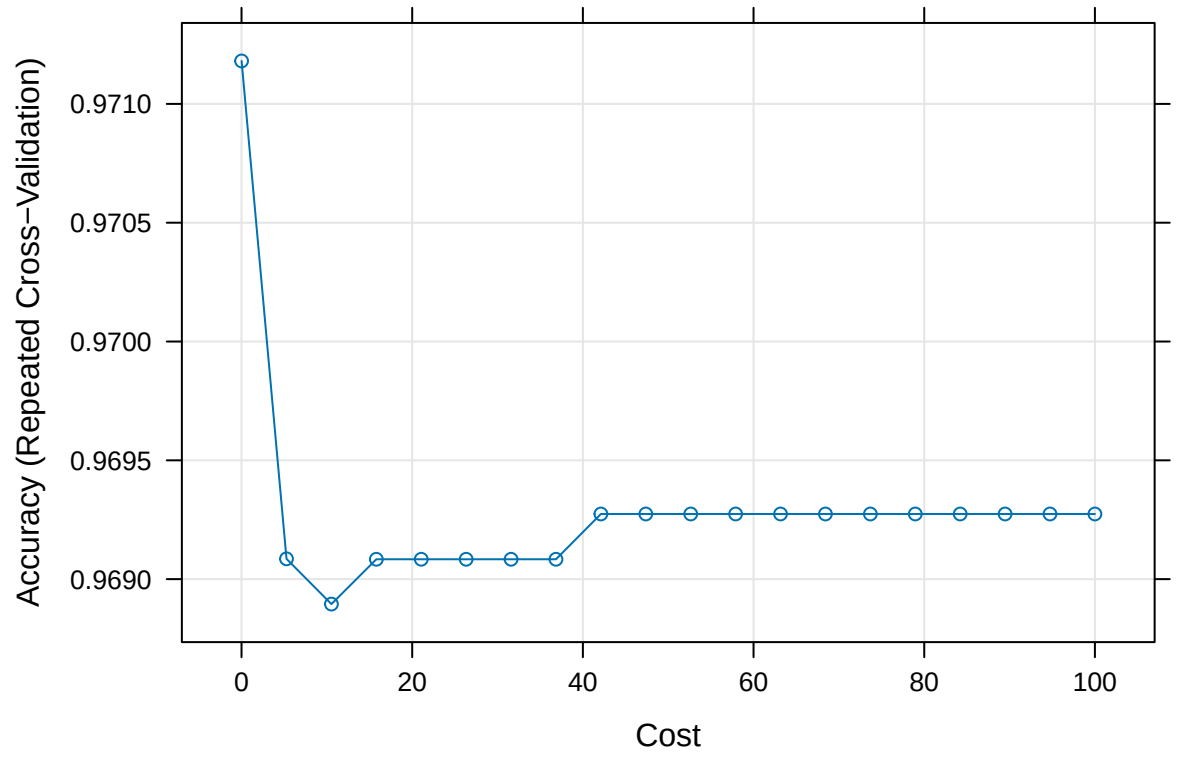
```
set.seed(1)
reg_svm <- train(as.factor(Class)~.,
                           reg_train,
                           method = "svmLinear",
                           preProcess = c("center", "scale"),
                           tuneGrid = expand.grid(C = seq.int(0.01, 100, length = 20)),
                           trControl = trainControl(
                             method = "repeatedcv",
                             number = 5,
                             repeats = 10))
reg_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 524 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 419, 420, 419, 419, 419, 419, ...
## Resampling results across tuning parameters:
##
##   C           Accuracy   Kappa
##     0.010000  0.9711808  0.9367047
```

```
##     5.272632  0.9690855  0.9321133
##    10.535263  0.9688950  0.9317036
##    15.797895  0.9690837  0.9321213
##    21.060526  0.9690837  0.9321213
##    26.323158  0.9690837  0.9321213
##    31.585789  0.9690837  0.9321213
##    36.848421  0.9690837  0.9321213
##    42.111053  0.9692741  0.9325310
##    47.373684  0.9692741  0.9325310
##    52.636316  0.9692741  0.9325310
##    57.898947  0.9692741  0.9325310
##    63.161579  0.9692741  0.9325310
##    68.424211  0.9692741  0.9325310
##    73.686842  0.9692741  0.9325310
##    78.949474  0.9692741  0.9325310
##    84.212105  0.9692741  0.9325310
##    89.474737  0.9692741  0.9325310
##    94.737368  0.9692741  0.9325310
##   100.000000  0.9692741  0.9325310
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

Same result: **the best C-value after cross-validation for regression imputation data is 0.01, with an accuracy of 97.12%**:

```
plot(reg_svm)
```



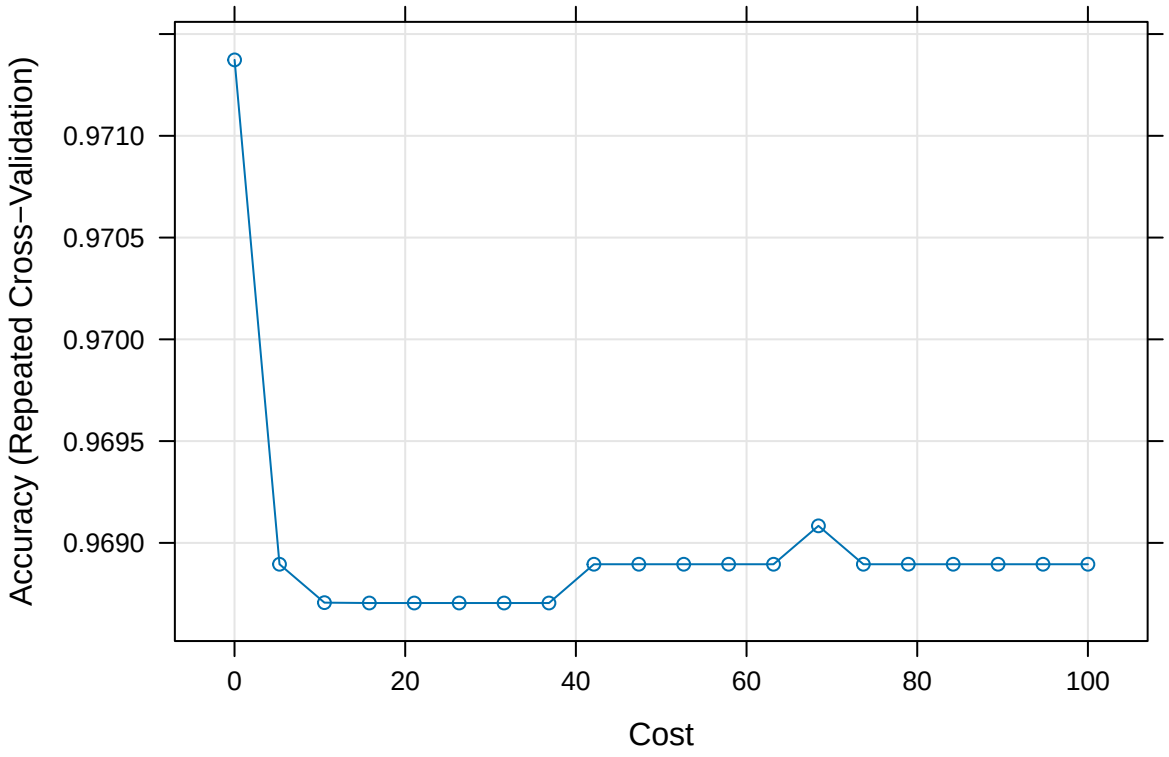
4. Regression & Perturbation imputation SVM:

```
set.seed(1)
pert_svm <- train(as.factor(Class)~.,
                          pert_train,
                          method = "svmLinear",
                          preProcess = c("center", "scale"),
                          tuneGrid = expand.grid(C = seq.int(0.01, 100, length = 20)),
                          trControl = trainControl(
                            method = "repeatedcv",
                            number = 5,
                            repeats = 10))
pert_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 524 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 419, 420, 419, 419, 419, 419, ...
## Resampling results across tuning parameters:
##
##   C           Accuracy   Kappa
##     0.010000  0.9713731  0.9371102
##     5.272632  0.9688950  0.9317036
##    10.535263  0.9687063  0.9312699
##    15.797895  0.9687045  0.9312625
##    21.060526  0.9687045  0.9312625
##    26.323158  0.9687045  0.9312625
##    31.585789  0.9687045  0.9312625
##    36.848421  0.9687045  0.9312625
##    42.111053  0.9688950  0.9316722
##    47.373684  0.9688950  0.9316722
##    52.636316  0.9688950  0.9316722
##    57.898947  0.9688950  0.9316722
##    63.161579  0.9688950  0.9316722
##    68.424211  0.9690837  0.9320897
##    73.686842  0.9688950  0.9316722
##    78.949474  0.9688950  0.9316722
##    84.212105  0.9688950  0.9316722
##    89.474737  0.9688950  0.9316722
##    94.737368  0.9688950  0.9316722
##   100.000000  0.9688950  0.9316722
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

The accuracy is a bit higher this time**the best C-value after cross-validation for regression + perturbation imputation data is 0.01, with an accuracy of 97.14%**:

```r
plot(pert_svm)
```



OVerall, **with kSVM models the cross-validated models have a very similar accuracy of 97, and KNN surpasses them by only 0.02%**. Let's see if KNN and SVM are as similar on the test data.

[kSVM] Step 2: Predicting + Estimating the Accuracy

For each data set, let's predict the response class and calculate the test accuracy.

1. Mean imputation kSVM Accuracy:

```r
set.seed(1)

#make the prediction
predict_mean_svm <- predict(mean_svm,mean_test)

#confusion matrix
cm_mean_svm <- table(Actual=mean_test$Class, Predicted=predict_mean_svm)
cm_mean_svm
```

```
##       Predicted
## Actual   0   1
##      0 112   5
##      1   2  56
```

```r
#calculate accuracy
accuracy_mean_svm <- sum(predict_mean_svm == mean_test[,10])/nrow(mean_test)
accuracy_mean_svm
```

```
## [1] 0.96
```

Results: With **Mean Imputation** the best **C=0.01** for cross-validated kSVM Model, with **97.12% accuracy on the training set** and **96% accuracy on the test set**.

2. Mode imputation kSVM Accuracy:

```
set.seed(1)

#make the prediction
predict_mode_svm <- predict(mode_svm,mode_test)

#confusion matrix
cm_mode_svm <- table(Actual=mode_test$Class, Predicted=predict_mode_svm)
cm_mode_svm
```

```
##       Predicted
## Actual   0   1
##      0 112   5
##      1   2  56
```

```
#calculate accuracy
accuracy_mode_svm <- sum(predict_mode_svm == mode_test[,10])/nrow(mode_test)
accuracy_mode_svm
```

```
## [1] 0.96
```

Results: With **Mode Imputation** the best **C=0.01** for cross-validated kSVM Model, with **97.12% accuracy on the training set** and **96% accuracy on the test set**.

The result is the same with mean imputation set.

3. Regression imputation kSVM Accuracy:

```
set.seed(1)

#make the prediction
predict_reg_svm <- predict(reg_svm,reg_test)

#confusion matrix
cm_reg_svm <- table(Actual=reg_test$Class, Predicted=predict_reg_svm)
cm_reg_svm
```

```
##       Predicted
## Actual   0   1
##      0 112   5
##      1   2  56
```

```
#calculate accuracy
accuracy_reg_svm <- sum(predict_reg_svm == reg_test[,10])/nrow(reg_test)
accuracy_reg_svm
```

```
## [1] 0.96
```

Results: With **Regression Imputation** the best **C=0.01** for cross-validated kSVM Model, with **97.12% accuracy on the training set** and **96.57% accuracy on the test set**.

With regression imputation, the results are slightly higher

4. Regression + Perturbation imputation kSVM Accuracy:

```r
set.seed(1)

#make the prediction
predict_pert_svm <- predict(pert_svm,pert_test)

#confusion matrix
cm_pert_svm <- table(Actual=pert_test$Class, Predicted=predict_pert_svm)
cm_pert_svm
```

```
##       Predicted
## Actual   0   1
##      0 112   5
##      1   2  56
```

```r
#calculate accuracy
accuracy_pert_svm <- sum(predict_pert_svm == pert_test[,10])/nrow(pert_test)
accuracy_pert_svm
```

```
## [1] 0.96
```

Results: With **Regression and Perturbation Imputation** the best **C=0.01** for cross-validated kSVM Model, with **97.14% accuracy on the training set** and **96% accuracy on the test set**.

All imputation methods provide us with good kSVM models with high accuracy on the training set of 96%, which is 0.5% higher than test accuracies with KNN - so kSVM works slightly better on our data.

Modeling on data with removed NAs

Let's build KNN and kSVM for a data set where NA values have been removed

[KNN] Step 1: Sampling

Remove NA values:

```r
rem_data <- data[-narows,]
rem_data[23:30,]
```

```
##         ID Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 23 1056784               3               1                1                 1
## 25 1059552               1               1                1                 1
## 26 1065726               5               2                3                 4
## 27 1066373               3               2                1                 1
## 28 1066979               5               1                1                 1
## 29 1067444               2               1                1                 1
## 30 1070935               1               1                3                 1
## 31 1070935               3               1                1                 1
##    Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 23                 2           1               2               1       1     0
## 25                 2           1               3               1       1     0
## 26                 2           7               3               6       1     1
## 27                 1           1               2               1       1     0
## 28                 2           1               2               1       1     0
## 29                 2           1               2               1       1     0
## 30                 2           1               1               1       1     0
## 31                 1           1               2               1       1     0
```

Sample the data (75/25):

```
set.seed(3)

#find the number of rows for test set
index <- round(nrow(rem_data)*0.25,digits=0)

#randomly sample the rows which will go to the test set (20%)
test_sample <- sample(1:nrow(rem_data), index)

#create the test set - exclude id column
rem_test <- rem_data[test_sample,-1]
head(rem_test)
```

```
##     Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 668               3               1                1                 1
## 563               1               1                1                 1
## 406               1               1                1                 1
## 344               1               1                1                 1
## 189               5               8                4                10
## 575              10               9                7                 3
##     Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 668                 2           1               3               1       1     0
## 563                 2           1               3               1       1     0
## 406                 2           1               2               1       1     0
## 344                 2           1               1               1       1     0
## 189                 5           8               9              10       1     1
## 575                 4           2               7               7       1     1
```

```
#create the train set
rem_train <- rem_data[-test_sample,-1]
head(rem_train)
```

```
##   Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 1               5               1                1                 1
## 2               5               4                4                 5
## 3               3               1                1                 1
## 5               4               1                1                 3
## 6               8              10               10                 8
## 7               1               1                1                 1
##   Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 1                 2           1               3               1       1     0
## 2                 7          10               3               2       1     0
## 3                 2           2               3               1       1     0
## 5                 2           1               3               1       1     0
## 6                 7          10               9               7       1     1
## 7                 2          10               3               1       1     0
```

```
#CHECK PROPORTIONS:

#whole set
prop.table(table(rem_data$Class))
```

```
##
##         0         1
## 0.6500732 0.3499268
```

```
#test set
prop.table(table(rem_test$Class))
```

```
##
##         0         1
## 0.6608187 0.3391813
```

```
#train set
prop.table(table(rem_train$Class))
```

```
##
##         0         1
## 0.6464844 0.3535156
```

Class proportion in the two new sets is good, we can build a model.

[KNN] Step 2: Preprocessing + Building the Model

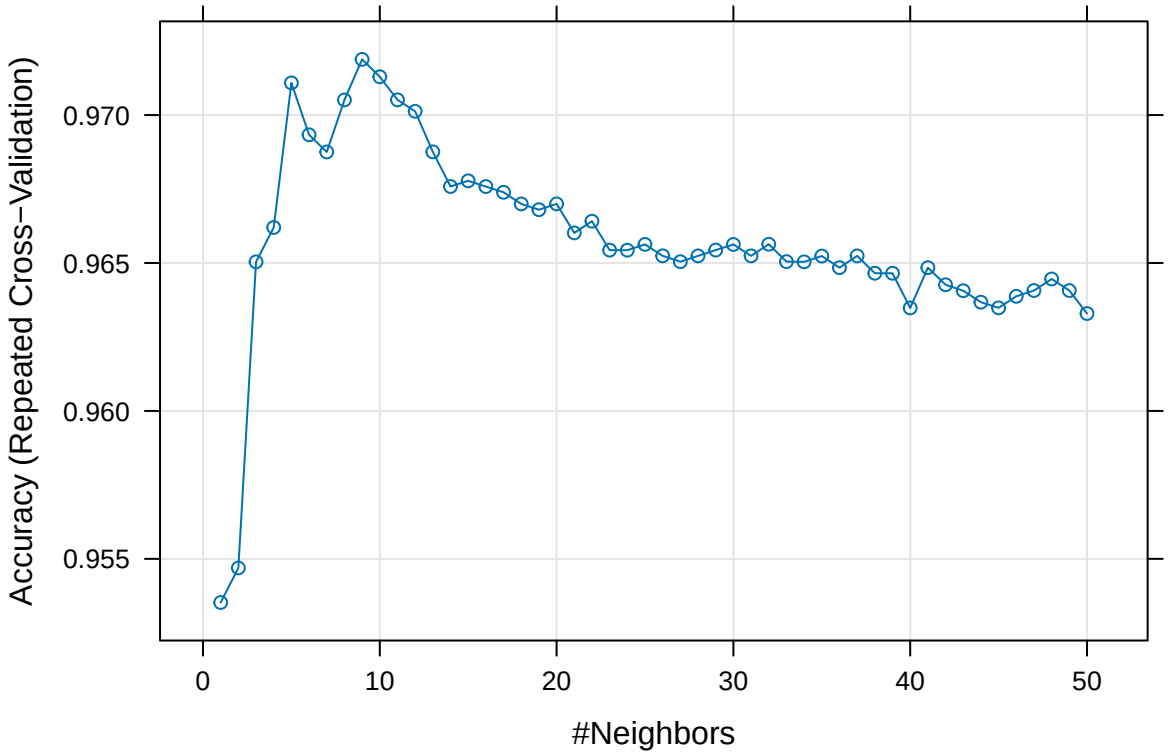
```
set.seed(1)
rem_knn <- train(as.factor(Class)~ .,
                      rem_train,
                      method = "knn",
                      preProcess = c("center", "scale"),
                      tuneGrid = data.frame(k = c(1:50)),
                      trControl = trainControl(
                        method = "repeatedcv",
                        number = 5,
                        repeats = 10))

rem_knn
```

```
## k-Nearest Neighbors
##
## 512 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 410, 409, 410, 409, 410, 409, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9535249  0.8974000
##    2  0.9546938  0.9001320
##    3  0.9650386  0.9233990
##    4  0.9661999  0.9259385
##    5  0.9710867  0.9367926
##    6  0.9693333  0.9329507
##    7  0.9687527  0.9316825
##    8  0.9705117  0.9355763
##    9  0.9718823  0.9386037
##   10  0.9712941  0.9373393
##   11  0.9705136  0.9356535
##   12  0.9701271  0.9347977
```

```
##   13  0.9687564  0.9317599
##   14  0.9675857  0.9292009
##   15  0.9677760  0.9295885
##   16  0.9675838  0.9291180
##   17  0.9673896  0.9287313
##   18  0.9669955  0.9278326
##   19  0.9668014  0.9274302
##   20  0.9669974  0.9278511
##   21  0.9660170  0.9256697
##   22  0.9664130  0.9265338
##   23  0.9654345  0.9243486
##   24  0.9654326  0.9243135
##   25  0.9656287  0.9247563
##   26  0.9652422  0.9239210
##   27  0.9650442  0.9234645
##   28  0.9652403  0.9238855
##   29  0.9654364  0.9243082
##   30  0.9656267  0.9246454
##   31  0.9652403  0.9238412
##   32  0.9656343  0.9246954
##   33  0.9650461  0.9233551
##   34  0.9650385  0.9233635
##   35  0.9652346  0.9237356
##   36  0.9648425  0.9229238
##   37  0.9652403  0.9237895
##   38  0.9646540  0.9224621
##   39  0.9646521  0.9224298
##   40  0.9634794  0.9198686
##   41  0.9648425  0.9227940
##   42  0.9642636  0.9215170
##   43  0.9640619  0.9210506
##   44  0.9636754  0.9201929
##   45  0.9634831  0.9197075
##   46  0.9638734  0.9205629
##   47  0.9640695  0.9209842
##   48  0.9644598  0.9218118
##   49  0.9640695  0.9209392
##   50  0.9632928  0.9191989
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

Unlike previously where optimal k was 3, with removed NA values the **cross-validated model with highest accuracy (97.19%) is at k=9:**

```
plot(rem_knn)
```



Let's see which accuracy our model will have on the test set.

[KNN] Step 3: Predicting + Estimating the Accuracy

Calculate model's accuracy on the test data:

```
set.seed(1)

#make the prediction
predict_rem <- predict(rem_knn,rem_test)

#confusion matrix
cm_rem <- table(Actual=rem_test$Class, Predicted=predict_rem)
cm_rem
```

```
##       Predicted
## Actual   0   1
##      0 112   1
##      1   7  51
```

```
#calculate accuracy
accuracy_rem <- sum(predict_rem == rem_test[,10])/nrow(rem_test)
accuracy_rem
```

```
## [1] 0.9532164
```

Results: Our KNN model built on the data where **rows with missing data have been removed** , has the accuracy of **95.32% on the test set** using the best k=9 found after cross-validation of models build with different k on training data.

This test accuracy is similar to what we got using imputation methods.

[kSVM] Step 1: Preprocessing + Building the Models

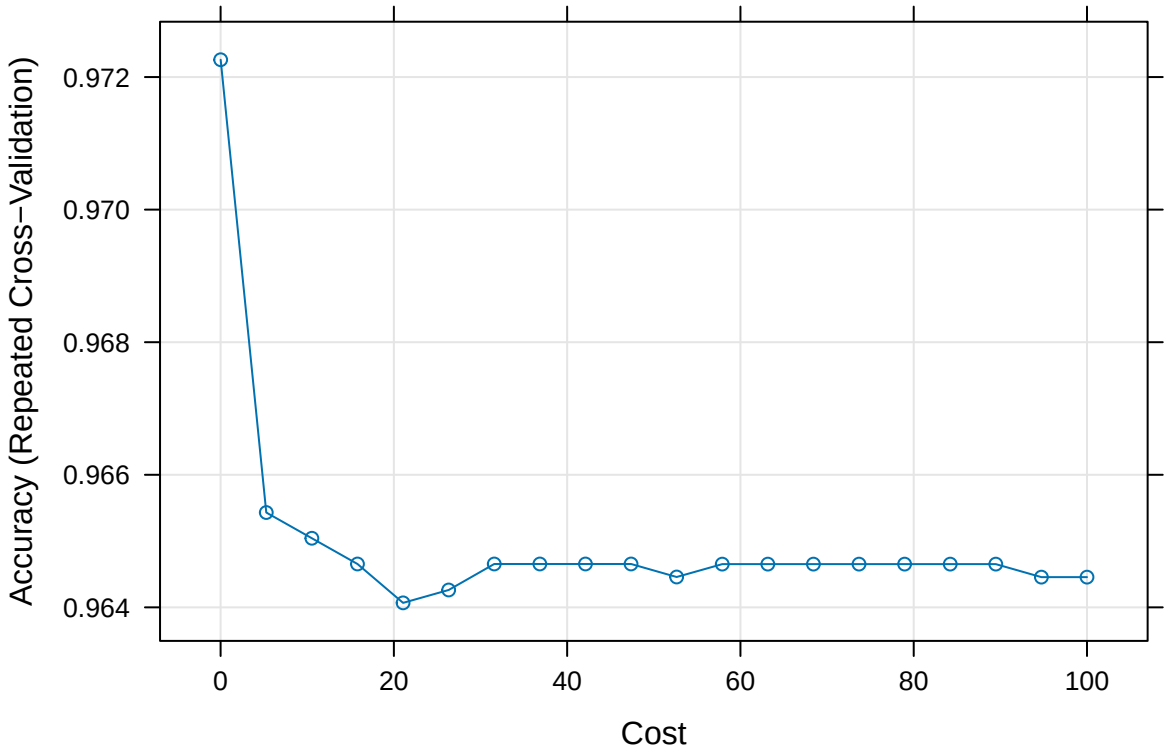
Now let's build the kSVM model for the data set with removed NAs using the same process as before:

```
set.seed(1)
rem_svm <- train(as.factor(Class)~.,
                           rem_train,
                           method = "svmLinear",
                           preProcess = c("center", "scale"),
                           tuneGrid = expand.grid(C = seq.int(0.01, 100, length = 20)),
                           trControl = trainControl(
                             method = "repeatedcv",
                             number = 5,
                             repeats = 10))
rem_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 512 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 410, 409, 410, 409, 410, 409, ...
## Resampling results across tuning parameters:
##
##   C           Accuracy   Kappa
##     0.010000  0.9722612  0.9398476
##     5.272632  0.9654308  0.9248136
##    10.535263  0.9650424  0.9239649
##    15.797895  0.9646541  0.9231505
##    21.060526  0.9640677  0.9218354
##    26.323158  0.9642619  0.9222756
##    31.585789  0.9646541  0.9231119
##    36.848421  0.9646541  0.9231119
##    42.111053  0.9646541  0.9231119
##    47.373684  0.9646541  0.9231119
##    52.636316  0.9644580  0.9226687
##    57.898947  0.9646522  0.9231032
##    63.161579  0.9646522  0.9231032
##    68.424211  0.9646522  0.9231032
##    73.686842  0.9646522  0.9231032
##    78.949474  0.9646522  0.9231032
##    84.212105  0.9646522  0.9231032
##    89.474737  0.9646522  0.9231032
##    94.737368  0.9644561  0.9226716
##   100.000000  0.9644561  0.9226716
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

The best C-value after cross-validation for data with removed NAs is 0.01, with an accuracy of 97.23%:

```
plot(rem_svm)
```



[kSVM] Step 2: Predicting + Estimating the Accuracy

Estimate model's accuracy on the test set:

```
set.seed(1)

#make the prediction
predict_rem_svm <- predict(rem_svm,rem_test)

#confusion matrix
cm_rem_svm <- table(Actual=rem_test$Class, Predicted=predict_rem_svm)
cm_rem_svm
```

```
##       Predicted
## Actual   0   1
##      0 112   1
##      1   5  53
```

```
#calculate accuracy
accuracy_rem_svm <- sum(predict_rem_svm == rem_test[,10])/nrow(rem_test)
accuracy_rem_svm
```

```
## [1] 0.9649123
```

As we can see, removal of NA values has led us to KNN and kSVM model with similar accuracies to those with imputation methods used - **with best C=0.01, accuracy on the test set is 96.49%**. So far, all

models have very similar test accuracies.

Modeling on data with binary variable

Finally, let's build KNN and kSVM for a data set where a binary variable indicates whether the data is missing

[KNN] Step 1: Sampling

Let's add the binary variable to the data (0 when data is missing):

```
binary <- data
binary$Missing[is.na(data$Bare_Nuclei)==T] <- 0
binary$Missing[is.na(data$Bare_Nuclei)==F] <- 1

binary[23:30,]
```

```
##         ID Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 23 1056784               3               1                1                 1
## 24 1057013               8               4                5                 1
## 25 1059552               1               1                1                 1
## 26 1065726               5               2                3                 4
## 27 1066373               3               2                1                 1
## 28 1066979               5               1                1                 1
## 29 1067444               2               1                1                 1
## 30 1070935               1               1                3                 1
##    Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 23                 2           1               2               1       1     0
## 24                 2          NA               7               3       1     1
## 25                 2           1               3               1       1     0
## 26                 2           7               3               6       1     1
## 27                 1           1               2               1       1     0
## 28                 2           1               2               1       1     0
## 29                 2           1               2               1       1     0
## 30                 2           1               1               1       1     0
##    Missing
## 23       1
## 24       0
## 25       1
## 26       1
## 27       1
## 28       1
## 29       1
## 30       1
```

And add the interaction factor:

```
#0 if NA
binary$Interact[is.na(data$Bare_Nuclei)==T] <- 0

#normal value if not NA
binary$Interact[is.na(data$Bare_Nuclei)==F] <- as.integer(data[-narows,]$Bare_Nuclei)

binary[23:30,]
```

```
##         ID Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 23 1056784               3               1                1                 1
```

```
## 24 1057013               8              4                5                1
## 25 1059552               1              1                1                1
## 26 1065726               5              2                3                4
## 27 1066373               3              2                1                1
## 28 1066979               5              1                1                1
## 29 1067444               2              1                1                1
## 30 1070935               1              1                3                1
##    Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 23                 2           1               2               1       1     0
## 24                 2          NA               7               3       1     1
## 25                 2           1               3               1       1     0
## 26                 2           7               3               6       1     1
## 27                 1           1               2               1       1     0
## 28                 2           1               2               1       1     0
## 29                 2           1               2               1       1     0
## 30                 2           1               1               1       1     0
##    Missing Interact
## 23       1        1
## 24       0        0
## 25       1        1
## 26       1        7
## 27       1        1
## 28       1        1
## 29       1        1
## 30       1        1
```

Now we can sample the data (75/25 as usual):

```
set.seed(1)

#find the number of rows for test set
index <- round(nrow(binary)*0.25,digits=0)

#randomly sample the rows which will go to the test set (20%)
test_sample <- sample(1:nrow(binary), index)

#create the test set - exclude id column
bin_test <- binary[test_sample,-1]
head(bin_test)
```

```
##     Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 679               1               1                1                1
## 129               8               3                5                4
## 509               5               1                1                1
## 471               3               1                1                1
## 299               8               2                1                1
## 270               1               1                1                1
##     Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 679                 2           1               1               1       1     0
## 129                 5          10               1               6       2     1
## 509                 2           1               1               1       1     0
## 471                 1           1               2               1       1     0
## 299                 5           1               1               1       1     0
## 270                 2           1               3               1       1     0
##     Missing Interact
```

```
## 679       1        1
## 129       1       10
## 509       1        1
## 471       1        1
## 299       1        1
## 270       1        1
```

```
#create the train set
bin_train <- binary[-test_sample,-1]
head(bin_train)
```

```
##   Clump_Thickness Uniformity_Size Uniformity_Shape Marginal_Adhesion
## 2               5               4                4                 5
## 3               3               1                1                 1
## 4               6               8                8                 1
## 5               4               1                1                 3
## 6               8              10               10                 8
## 7               1               1                1                 1
##   Single_Epith_Size Bare_Nuclei Bland_Chromatin Normal_Nucleoli Mitoses Class
## 2                 7          10               3               2       1     0
## 3                 2           2               3               1       1     0
## 4                 3           4               3               7       1     0
## 5                 2           1               3               1       1     0
## 6                 7          10               9               7       1     1
## 7                 2          10               3               1       1     0
##   Missing Interact
## 2       1       10
## 3       1        2
## 4       1        4
## 5       1        1
## 6       1       10
## 7       1       10
```

```
#CHECK PROPORTIONS:

#whole set
prop.table(table(binary$Class))
```

```
##
##         0         1
## 0.6552217 0.3447783
```

```
#test set
prop.table(table(bin_test$Class))
```

```
##
##         0         1
## 0.6685714 0.3314286
```

```
#train set
prop.table(table(bin_train$Class))
```

```
##
##         0         1
## 0.6507634 0.3492366
```

Class proportion in the two new sets is good, we can build a model

[KNN] Step 2: Preprocessing + Building the Model

We will use the interaction factor in our set instead of Bare_Nuclei (and not use the 'missing' column, since the 0 values are already in the interaction variable)

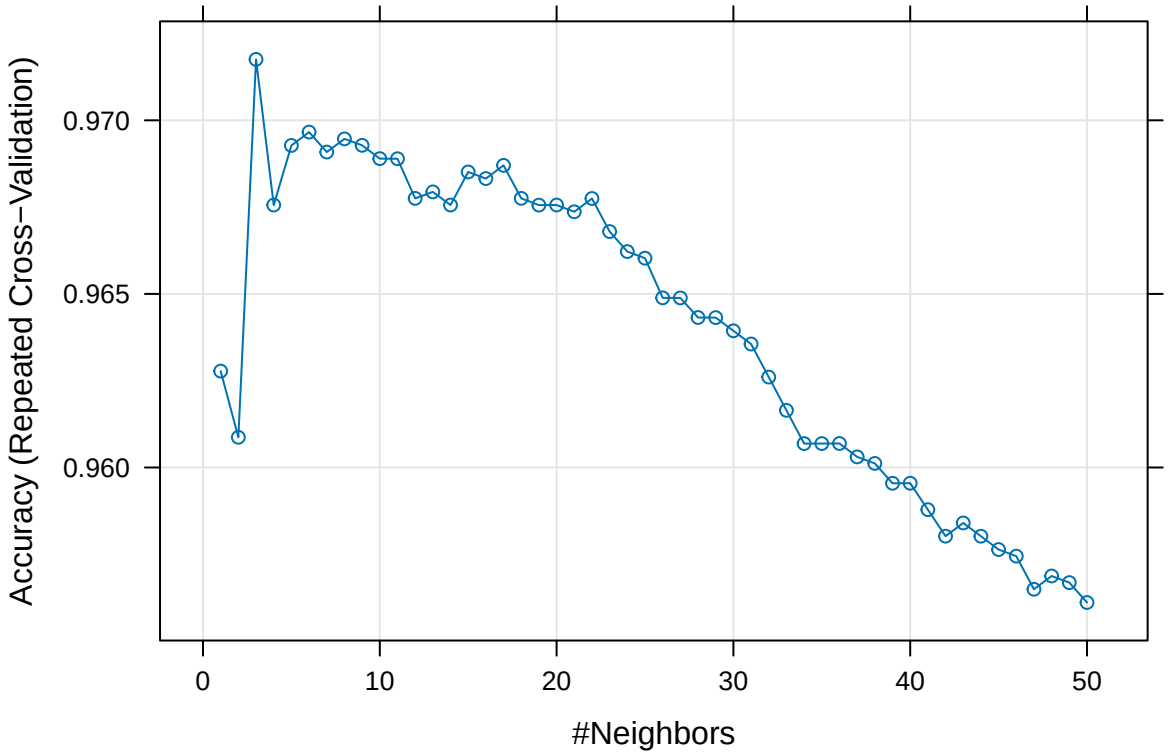
```
set.seed(1)
bin_knn <- train(as.factor(Class)~ Clump_Thickness + Uniformity_Size + Uniformity_Shape + Marginal_Adhes
                       bin_train,
                       method = "knn",
                       preProcess = c("center", "scale"),
                       tuneGrid = data.frame(k = c(1:50)),
                       trControl = trainControl(
                         method = "repeatedcv",
                         number = 5,
                         repeats = 10))

bin_knn
```

```
## k-Nearest Neighbors
##
## 524 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 419, 420, 419, 419, 419, 419, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9627758  0.9175648
##    2  0.9608691  0.9133504
##    3  0.9717559  0.9379703
##    4  0.9675597  0.9286260
##    5  0.9692759  0.9324485
##    6  0.9696606  0.9332689
##    7  0.9690855  0.9319686
##    8  0.9694646  0.9328079
##    9  0.9692796  0.9323742
##   10  0.9688968  0.9315012
##   11  0.9688932  0.9314839
##   12  0.9677521  0.9289920
##   13  0.9679389  0.9294663
##   14  0.9675597  0.9286558
##   15  0.9685122  0.9307615
##   16  0.9683217  0.9303250
##   17  0.9687027  0.9311060
##   18  0.9677521  0.9289880
##   19  0.9675580  0.9285573
##   20  0.9675597  0.9285532
##   21  0.9673657  0.9281398
##   22  0.9677485  0.9289661
##   23  0.9667960  0.9268614
##   24  0.9662192  0.9255645
##   25  0.9660287  0.9250790
```

```
##   26  0.9648858  0.9225169
##   27  0.9648840  0.9224776
##   28  0.9643198  0.9212363
##   29  0.9643180  0.9211957
##   30  0.9639370  0.9203131
##   31  0.9635561  0.9194445
##   32  0.9626036  0.9172520
##   33  0.9616458  0.9151067
##   34  0.9606879  0.9129116
##   35  0.9606897  0.9129253
##   36  0.9606915  0.9129442
##   37  0.9603051  0.9120539
##   38  0.9601165  0.9116161
##   39  0.9595450  0.9103137
##   40  0.9595468  0.9103156
##   41  0.9587849  0.9085777
##   42  0.9580212  0.9068704
##   43  0.9584021  0.9077362
##   44  0.9580194  0.9068343
##   45  0.9576366  0.9059485
##   46  0.9574461  0.9054845
##   47  0.9564918  0.9033244
##   48  0.9568746  0.9041932
##   49  0.9566841  0.9037510
##   50  0.9561109  0.9024568
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 3.
```

Using data with a binary variable indicating missing values and an intercation factor, the **cross-validated model with highest accuracy (97.18%) is at k=3:**

```
plot(bin_knn)
```



Let's see which accuracy our model will have on the test set.

[KNN] Step 3: Predicting + Estimating the Accuracy

Calculate model's accuracy on the test data:

```
set.seed(1)

#make the prediction
predict_bin <- predict(bin_knn,bin_test)

#confusion matrix
cm_bin <- table(Actual=bin_test$Class, Predicted=predict_bin)
cm_bin
```

```
##       Predicted
## Actual   0   1
##      0 113   4
##      1   4  54
```

```
#calculate accuracy
accuracy_bin <- sum(predict_bin == bin_test[,10])/nrow(bin_test)
accuracy_bin
```

```
## [1] 0.9542857
```

Results: Our KNN model built on the data where **a binary variable indicated NA values and an interaction factor replaces the attribute with missing data**, has the accuracy of **95.43% on the test set** using the best k=3 found after cross-validation of models build with different k on training data.

This test accuracy is the same with what we got using imputation methods.

[kSVM] Step 1: Preprocessing + Building the Models

Now let's build the kSVM model using the same data set:

```
set.seed(1)
bin_svm <- train(as.factor(Class)~ Clump_Thickness + Uniformity_Size + Uniformity_Shape + Marginal_Adhes
                           bin_train,
                           method = "svmLinear",
                           preProcess = c("center", "scale"),
                           tuneGrid = expand.grid(C = seq.int(0.01, 100, length = 20)),
                           trControl = trainControl(
                             method = "repeatedcv",
                             number = 5,
                             repeats = 10))
bin_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 524 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 419, 420, 419, 419, 419, 419, ...
## Resampling results across tuning parameters:
##
##   C           Accuracy   Kappa
##     0.010000  0.9696588  0.9332686
##     5.272632  0.9698456  0.9338120
##    10.535263  0.9696569  0.9333837
##    15.797895  0.9696569  0.9333837
##    21.060526  0.9696569  0.9333837
##    26.323158  0.9694664  0.9329639
##    31.585789  0.9694664  0.9329639
##    36.848421  0.9694664  0.9329639
##    42.111053  0.9694664  0.9329639
##    47.373684  0.9694664  0.9329639
##    52.636316  0.9694664  0.9329639
##    57.898947  0.9692760  0.9325389
##    63.161579  0.9692760  0.9325389
##    68.424211  0.9692760  0.9325389
##    73.686842  0.9692760  0.9325389
##    78.949474  0.9692760  0.9325389
##    84.212105  0.9692760  0.9325389
##    89.474737  0.9690837  0.9321001
##    94.737368  0.9692760  0.9325389
##   100.000000  0.9692760  0.9325389
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 5.272632.
```

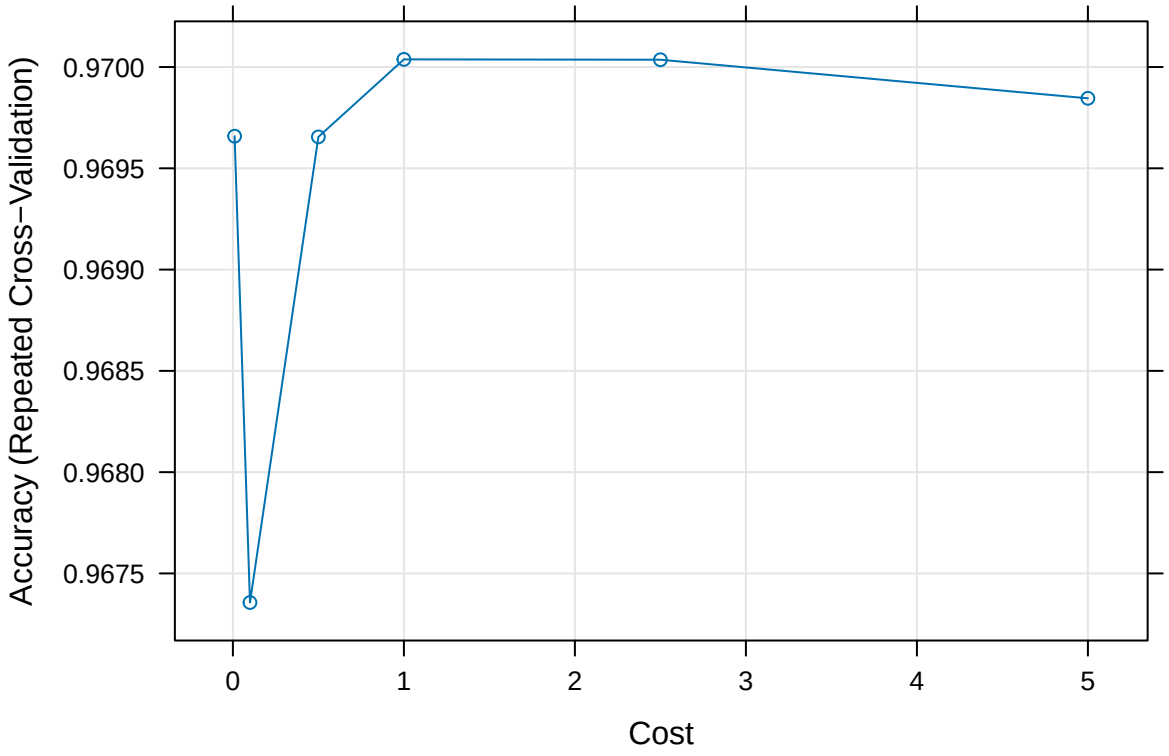
Looks like the best accuracy lies between 0.01 and 5 this time - let's test several values there to find the best C:

```r
set.seed(1)
bin_svm <- train(as.factor(Class)~ Clump_Thickness + Uniformity_Size + Uniformity_Shape + Marginal_Adhes
                           bin_train,
                           method = "svmLinear",
                           preProcess = c("center", "scale"),
                           tuneGrid = expand.grid(C = c(0.01, 0.1, 0.5, 1, 2.5, 5)),
                           trControl = trainControl(
                             method = "repeatedcv",
                             number = 5,
                             repeats = 10))
bin_svm
```

```
## Support Vector Machines with Linear Kernel
##
## 524 samples
##   9 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 419, 420, 419, 419, 419, 419, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.01  0.9696588  0.9332686
##   0.10  0.9673565  0.9283091
##   0.50  0.9696551  0.9333752
##   1.00  0.9700379  0.9342219
##   2.50  0.9700361  0.9341835
##   5.00  0.9698456  0.9338120
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.
```

The best C-value after cross-validation for data with binary variable for NAs was found at C=1, with an accuracy of 97.00%:

```r
plot(bin_svm)
```



[kSVM] Step 2: Predicting + Estimating the Accuracy

Estimate model's accuracy on the test set:

```
set.seed(1)

#make the prediction
predict_bin_svm <- predict(bin_svm,bin_test)

#confusion matrix
cm_bin_svm <- table(Actual=bin_test$Class, Predicted=predict_bin_svm)
cm_bin_svm
```

```
##       Predicted
## Actual   0   1
##      0 113   4
##      1   2  56
```

```
#calculate accuracy
accuracy_bin_svm <- sum(predict_bin_svm == bin_test[,10])/nrow(bin_test)
accuracy_bin_svm
```

```
## [1] 0.9657143
```

With **best C=1, the test accuracy is 96.57%**.

Results Table

Overall, all the data sets (with imputation, NA removal and binary variables indicating missing data) have led us to **similar models with only a slight difference in test accuracy**. In general,**test accuracies of kSVM models is a bit higher than KKN**, but all the models appear to have good classification quality. It seems like for this data set, **the key is to find the best tuning parameters like k and C instead of using a particular imputation approach** - but perhaps with more data and future classifications imputation's benefits compared to removed rows, for example, will be more obvious.

Below is the comparison of accuracies for all models we've built:

KNN Models

```
# Define the data
knn <- tibble::tribble(
    ~Type, ~Best_k, ~Accuracy_Train_CV, ~Accuracy_Test,
    "Mean Imputation",       3,           "97.14%",       "95.43%",
    "Mode Imputation",       3,           "97.16%",       "95.43%",
    "Regression Imputation",       3,           "97.16%",       "95.43%",
    "Regression with Perturbation Imputation",       3,           "97.16%",       "95.43%",
    "Removal of Missing Values",       9,           "97.19%",       "95.32%",
    "Binary Variable with Interaction Factor",       3,           "97.18%",       "95.43%"
)

# Use knitr's kable function to generate the table
knitr::kable(knn, digits = 3, row.names = FALSE, align = 'c')
```

Type	Best_k	Accuracy_Train_CV	Accuracy_Test
Mean Imputation	3	97.14%	95.43%
Mode Imputation	3	97.16%	95.43%
Regression Imputation	3	97.16%	95.43%
Regression with Perturbation Imputation	3	97.16%	95.43%
Removal of Missing Values	9	97.19%	95.32%
Binary Variable with Interaction Factor	3	97.18%	95.43%

kSVM Models

```
# svm tibble
svm <- tibble::tribble(
  ~Type, ~Best_C, ~Accuracy_Train_CV, ~Accuracy_Test,
  "Mean Imputation", 0.01,  "97.12%", "96.00%",
  "Mode Imputation", 0.01,  "97.12%", "96.00%",
  "Regression Imputation", 0.01,  "97.12%", "96.57%",
  "Regression with Perturbation Imputation", 0.01,  "97.14%", "96.00%",
  "Removal of Missing Values", 0.01,  "97.23%", "96.49%",
  "Binary Variable with Interaction Factor", 1,  "97.00%", "96.57%"
)

# Display the table
kable(svm, digits = 3, row.names = FALSE, align = "c")
```

Type	Best_C	Accuracy_Train_CV	Accuracy_Test
Mean Imputation	0.01	97.12%	96.00%
Mode Imputation	0.01	97.12%	96.00%

Type	Best_C	Accuracy_Train_CV	Accuracy_Test
Regression Imputation	0.01	97.12%	96.57%
Regression with Perturbation Imputation	0.01	97.14%	96.00%
Removal of Missing Values	0.01	97.23%	96.49%
Binary Variable with Interaction Factor	1.00	97.00%	96.57%