HW10

2022-10-31

Question 14.1

Setup

Clear the environment, set the seed, load the dplyr library, and write a function to make it simple to reload or update the dataset. Considering that I'll want to start with the original dataset for each inquiry portion, I wrote the load data function.

```
# Clear the environment
rm(list = ls())

# Comment in set.seed(33) to repeat results
set.seed(33)

# Load dplyr lib
require(dplyr)
```

```
## Loading required package: dplyr
```

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

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

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

```
# Create function to re-load data, since we'll want to start with a fresh dataset for each part
load_data <- function() {

  # Load cancer data into a data frame
  data_df <- read.table("breast-cancer-wisconsin.data.txt", header=FALSE, sep=",", stringsAsFactors = T

    # Update V11 (response) field from 2/4 to 0/1
  data_df$V11[data_df$V11 == 2] <- 0
  data_df$V11[data_df$V11 == 4] <- 1

  # Replace ? with NA in data_df
  data_df[data_df=='?'] <- ''
  # Return data_df
  return(data_df)
}
```

Identification of Fields with Missing Data

I developed a function called Find Columns with Missing() that filters the data tbl and counts the number of missing rows in order to determine which fields had missing data. I discovered that column V7 had 16 missing values and was the only column with no missing data.

```
### PART 0: Identifiy fields with missing data
# Load data into a data frame
data_df <- load_data()
```

```
## Warning in ‘[<-.factor‘(‘*tmp*‘, thisvar, value = ""): invalid factor level, NA
## generated
```

```
# Change data_df into dplyr table
data_tbl <- tbl_df(data_df)
```

```
## Warning: ‘tbl_df()‘ was deprecated in dplyr 1.0.0.
## Please use ‘tibble::as_tibble()‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was generated.
```

```
# Function to identify columns with missing data
Find_Columns_with_Missing <- function(table, column) {
  filtered_tbl <- filter(table, is.na(table[column]))
  records <- nrow(filtered_tbl)

  return(records)
}

# Create placeholder for Find_Columns_with_Missing results
missing_tbl <- tbl_df(colnames(data_tbl))

# Loop through each column in data_tbl
for (i in 1:nrow(missing_tbl)) {
  missing_tbl[i,2] <- Find_Columns_with_Missing(data_tbl, i)
}

# Filter to only show columns with missing variables
cols_w_na_data <- filter(missing_tbl, missing_tbl[2]>0)
# V7 has 16 missing values
cols_w_na_data
```

```
## # A tibble: 1 x 2
##   value  ...2
##   <chr> <int>
## 1 V7       16
```

Imputing Using Mode

I choose to utilize mode to impute values for part 1 because to the ordinal nature of the factors. With the use of a mode function, I substituted the mode for V7's missing data.

```
### PART 1: Impute using mode
# Load data into a data frame
data_df <- load_data()
```

```
## Warning in `[<-.factor`(`*tmp*`, thisvar, value = ""): invalid factor level, NA
## generated
```

```
# Function to calculate the mode
# Source: https://stackoverflow.com/users/169947/ken-williams
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

# Impute nulls with mode (due to ordinal scale of bare_nuclei)
data_df$V7[is.na(data_df[,'V7'])] <- Mode(data_df[,'V7'])
data_df <- transform(data_df, V7 = as.numeric(as.character(V7)))
summary(data_df)
```

```
##        V1                 V2               V3               V4        
##  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  
##        V5               V6               V7               V8        
##  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   : 3.486   Mean   : 3.438  
##  3rd Qu.: 4.000   3rd Qu.: 4.000   3rd Qu.: 5.000   3rd Qu.: 5.000  
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000  
##        V9              V10              V11        
##  Min.   : 1.000   Min.   : 1.000   Min.   :0.0000  
##  1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:0.0000  
##  Median : 1.000   Median : 1.000   Median :0.0000  
##  Mean   : 2.867   Mean   : 1.589   Mean   :0.3448  
##  3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:1.0000  
##  Max.   :10.000   Max.   :10.000   Max.   :1.0000  
```

Impute using Linear Regression

For Part 2, I first divided the dataset into two parts: one with all records that had complete data and one with all records missing data. Using the dataset, I created an imputation model that used all factors, with the exception of V1 (i.e. ID) and V11 (i.e. Response). Using the imputation model, I used step() to perform backward step factor selection. Using the step-recommender, I used the step-recommender to select the factors

```r
### PART 2: Impute using Regression
# Load data into a data frame
data_df <- load_data()
```

```
## Warning in ‘[<-.factor‘(‘*tmp*‘, thisvar, value = ""): invalid factor level, NA
## generated
```

```r
# Splice table into records with/without missing data
data_df_w_na <- filter(data_df, is.na(data_df$V7))
data_df_wo_na <- filter(data_df, !is.na(data_df$V7))

# Create a linear regression model
imputation_model <- lm(as.numeric(V7) ~ V2 + V3 + V4 + V5 + V6 + V8 + V9 + V10, data_df_wo_na)
summary(imputation_model)
```

```
## 
## Call:
## lm(formula = as.numeric(V7) ~ V2 + V3 + V4 + V5 + V6 + V8 + V9 + 
##     V10, data = data_df_wo_na)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -4.1137 -0.7185 -0.4731 -0.2994  7.3848 
## 
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  1.862817   0.162497  11.464  < 2e-16 ***
## V2           0.068118   0.034746   1.960  0.05035 .  
## V3           0.087939   0.063482   1.385  0.16643    
## V4           0.110046   0.061190   1.798  0.07255 .  
## V5          -0.076950   0.038270  -2.011  0.04475 *  
## V6           0.043216   0.052123   0.829  0.40733    
## V8           0.044536   0.049211   0.905  0.36579    
## V9           0.119422   0.037076   3.221  0.00134 ** 
## V10          0.001405   0.049448   0.028  0.97733    
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
## 
## Residual standard error: 1.896 on 674 degrees of freedom
## Multiple R-squared:  0.2326, Adjusted R-squared:  0.2235 
## F-statistic: 25.54 on 8 and 674 DF,  p-value: < 2.2e-16
```

```r
# Use stepwise for factor selection
step(imputation_model, direction = "backward")
```

```
## Start:  AIC=882.53
## as.numeric(V7) ~ V2 + V3 + V4 + V5 + V6 + V8 + V9 + V10
## 
##        Df Sum of Sq    RSS    AIC
## - V10   1     0.003 2421.8 880.53
## - V6    1     2.470 2424.3 881.22
## - V8    1     2.943 2424.8 881.36
## - V3    1     6.895 2428.7 882.47
## <none>              2421.8 882.53
## - V4    1    11.622 2433.4 883.80
## - V2    1    13.810 2435.6 884.41
## - V5    1    14.527 2436.3 884.61
## - V9    1    37.280 2459.1 890.96
##
## Step:  AIC=880.53
## as.numeric(V7) ~ V2 + V3 + V4 + V5 + V6 + V8 + V9
##
##        Df Sum of Sq    RSS    AIC
## - V6    1     2.603 2424.4 879.26
## - V8    1     2.953 2424.8 879.36
## - V3    1     6.925 2428.7 880.48
## <none>              2421.8 880.53
## - V4    1    11.620 2433.4 881.80
## - V2    1    13.877 2435.7 882.43
## - V5    1    14.699 2436.5 882.66
## - V9    1    37.921 2459.7 889.14
##
## Step:  AIC=879.26
## as.numeric(V7) ~ V2 + V3 + V4 + V5 + V8 + V9
##
##        Df Sum of Sq    RSS    AIC
## - V8    1     3.126 2427.6 878.14
## <none>              2424.4 879.26
## - V3    1     9.964 2434.4 880.06
## - V4    1    12.613 2437.0 880.80
## - V5    1    13.821 2438.2 881.14
## - V2    1    14.269 2438.7 881.27
## - V9    1    41.469 2465.9 888.84
##
## Step:  AIC=878.14
## as.numeric(V7) ~ V2 + V3 + V4 + V5 + V9
##
##        Df Sum of Sq    RSS    AIC
## <none>              2427.6 878.14
## - V5    1    11.502 2439.1 879.37
## - V3    1    12.764 2440.3 879.72
## - V4    1    13.847 2441.4 880.03
## - V2    1    15.460 2443.0 880.48
## - V9    1    47.920 2475.5 889.49
```

```
##
## Call:
## lm(formula = as.numeric(V7) ~ V2 + V3 + V4 + V5 + V9, data = data_df_wo_na)
##
## Coefficients:
## (Intercept)           V2           V3           V4           V5           V9
##     1.96962      0.07169      0.11320      0.11926     -0.06574      0.13053
```

```
# Re-train the linear regression using stepwise recommended factors
step_model <- lm(as.numeric(V7) ~ V2 + V3 + V4 + V5 + V9, data_df_wo_na)
summary(step_model)
```

```
## 
## Call:
## lm(formula = as.numeric(V7) ~ V2 + V3 + V4 + V5 + V9, data = data_df_wo_na)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -4.0534 -0.7407 -0.4819 -0.3385  7.3673 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  1.96962    0.13706  14.370  < 2e-16 ***
## V2           0.07169    0.03453   2.076 0.038230 *  
## V3           0.11320    0.06000   1.887 0.059628 .  
## V4           0.11926    0.06069   1.965 0.049810 *  
## V5          -0.06574    0.03671  -1.791 0.073735 .  
## V9           0.13053    0.03570   3.656 0.000276 ***
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
## 
## Residual standard error: 1.894 on 677 degrees of freedom
## Multiple R-squared:  0.2308, Adjusted R-squared:  0.2251 
## F-statistic: 40.63 on 5 and 677 DF,  p-value: < 2.2e-16
```

```
# Predict values for V7 and round to convert to integers
V7 <- data.frame(round(predict(step_model, data_df_w_na)))
colnames(V7) <- c("V7")

# Impute the predictions to data_df_w_na
data_df_w_na <- cbind(data_df_w_na[,1:6], V7, data_df_w_na[,8:11])

# Combine data_df_w_na and data_df_wo_na into imputed_data_df
imputed_data_df <- rbind(data_df_w_na[,1:11], data_df_wo_na[,1:11])
imputed_data_df <- transform(imputed_data_df, V7 = as.numeric(V7))
summary(imputed_data_df)
```

```
##        V1                 V2               V3               V4        
##  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  
##        V5               V6               V7               V8        
##  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   : 3.531   Mean   : 3.438  
##  3rd Qu.: 4.000   3rd Qu.: 4.000   3rd Qu.: 5.500   3rd Qu.: 5.000  
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000  
##        V9              V10              V11        
##  Min.   : 1.000   Min.   : 1.000   Min.   :0.0000  
##  1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:0.0000  
##  Median : 1.000   Median : 1.000   Median :0.0000  
##  Mean   : 2.867   Mean   : 1.589   Mean   :0.3448  
##  3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:1.0000
##  Max.   :10.000   Max.   :10.000   Max.   :1.0000
```

Impute with Regression & Perturbation

Part 3 was similar to that in Part 2, but it also involved establishing a normal distribution of values and adding those values to the projected V7 values to produce the perturbed V7 values. The initial range of the perturbed results was 0:10, which was outside the initial range of 1:10; hence, I modified the 0 values to 1. The following code, which is exclusive to Part 3, has been bolded:

```
### PART 3: Impute using Regression with Perturbation
# Load data into a data frame
data_df <- load_data()
```

```
## Warning in ‘[<-.factor‘(‘*tmp*‘, thisvar, value = ""): invalid factor level, NA
## generated
```

```
# Splice table into records with/without missing data
data_df_w_na <- filter(data_df, is.na(data_df$V7))
data_df_wo_na <- filter(data_df, !is.na(data_df$V7))

# Create a linear regression model
imputation_model <- lm(as.numeric(V7) ~ V2 + V3 + V4 + V5 + V6 + V8 + V9 + V10, data_df_wo_na)
summary(imputation_model)
```

```
##
## Call:
## lm(formula = as.numeric(V7) ~ V2 + V3 + V4 + V5 + V6 + V8 + V9 +
##     V10, data = data_df_wo_na)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1137 -0.7185 -0.4731 -0.2994  7.3848
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.862817   0.162497  11.464  < 2e-16 ***
## V2           0.068118   0.034746   1.960  0.05035 .
## V3           0.087939   0.063482   1.385  0.16643
## V4           0.110046   0.061190   1.798  0.07255 .
## V5          -0.076950   0.038270  -2.011  0.04475 *
## V6           0.043216   0.052123   0.829  0.40733
## V8           0.044536   0.049211   0.905  0.36579
## V9           0.119422   0.037076   3.221  0.00134 **
## V10          0.001405   0.049448   0.028  0.97733
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 1.896 on 674 degrees of freedom
## Multiple R-squared:  0.2326, Adjusted R-squared:  0.2235
## F-statistic: 25.54 on 8 and 674 DF,  p-value: < 2.2e-16
```

```r
# Use stepwise for factor selection
step(imputation_model, direction = "backward")
```

```
## Start:  AIC=882.53
## as.numeric(V7) ~ V2 + V3 + V4 + V5 + V6 + V8 + V9 + V10
##
##         Df Sum of Sq    RSS    AIC
## - V10   1     0.003 2421.8 880.53
## - V6    1     2.470 2424.3 881.22
## - V8    1     2.943 2424.8 881.36
## - V3    1     6.895 2428.7 882.47
## <none>              2421.8 882.53
## - V4    1    11.622 2433.4 883.80
## - V2    1    13.810 2435.6 884.41
## - V5    1    14.527 2436.3 884.61
## - V9    1    37.280 2459.1 890.96
##
## Step:  AIC=880.53
## as.numeric(V7) ~ V2 + V3 + V4 + V5 + V6 + V8 + V9
##
##         Df Sum of Sq    RSS    AIC
## - V6    1     2.603 2424.4 879.26
## - V8    1     2.953 2424.8 879.36
## - V3    1     6.925 2428.7 880.48
## <none>              2421.8 880.53
## - V4    1    11.620 2433.4 881.80
## - V2    1    13.877 2435.7 882.43
## - V5    1    14.699 2436.5 882.66
## - V9    1    37.921 2459.7 889.14
##
## Step:  AIC=879.26
## as.numeric(V7) ~ V2 + V3 + V4 + V5 + V8 + V9
##
##         Df Sum of Sq    RSS    AIC
## - V8    1     3.126 2427.6 878.14
## <none>              2424.4 879.26
## - V3    1     9.964 2434.4 880.06
## - V4    1    12.613 2437.0 880.80
## - V5    1    13.821 2438.2 881.14
## - V2    1    14.269 2438.7 881.27
## - V9    1    41.469 2465.9 888.84
##
## Step:  AIC=878.14
## as.numeric(V7) ~ V2 + V3 + V4 + V5 + V9
##
##         Df Sum of Sq    RSS    AIC
## <none>              2427.6 878.14
## - V5    1    11.502 2439.1 879.37
## - V3    1    12.764 2440.3 879.72
## - V4    1    13.847 2441.4 880.03
## - V2    1    15.460 2443.0 880.48
## - V9    1    47.920 2475.5 889.49

##
```

```
## Call:
## lm(formula = as.numeric(V7) ~ V2 + V3 + V4 + V5 + V9, data = data_df_wo_na)
##
## Coefficients:
## (Intercept)           V2           V3           V4           V5           V9
##     1.96962      0.07169      0.11320      0.11926     -0.06574      0.13053
```

```
# Re-train the linear regression using stepwise recommended factors
step_model <- lm(as.numeric(V7) ~ V2 + V3 + V4 + V5 + V9, data_df_wo_na)

# Predict values for V7
V7 <- data.frame(predict(step_model, data_df_w_na))

# Create a normal distribution for perturbation
normal_dist <- data.frame(rnorm(nrow(V7), mean = 0, sd = 1))

# Add perturbation to predicted V7 values and round
perturbed_V7 <- data.frame(round(V7[,1] + normal_dist[,1]))
colnames(perturbed_V7) <- c("V7")

# Impute the predictions to data_df_w_na
data_df_w_na <- cbind(data_df_w_na[,1:6], perturbed_V7, data_df_w_na[,8:11])

# Combine data_df_w_na and data_df_wo_na into imputed_data_df
imputed_data_df <- rbind(data_df_w_na[,1:11], data_df_wo_na[,1:11])
imputed_data_df <- transform(imputed_data_df, V7 = as.numeric(as.character(V7)))
summary(imputed_data_df)
```

```
##        V1                 V2               V3               V4
##  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
##        V5                V6               V7               V8
##  Min.   : 1.000   Min.   : 1.000   Min.   : 0.000   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   : 3.534   Mean   : 3.438
##  3rd Qu.: 4.000   3rd Qu.: 4.000   3rd Qu.: 5.500   3rd Qu.: 5.000
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000
##        V9              V10              V11
##  Min.   : 1.000   Min.   : 1.000   Min.   :0.0000
##  1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:0.0000
##  Median : 1.000   Median : 1.000   Median :0.0000
##  Mean   : 2.867   Mean   : 1.589   Mean   :0.3448
##  3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:1.0000
##  Max.   :10.000   Max.   :10.000   Max.   :1.0000
```

```
# Update min value of V7 to fit 1:10 scale
imputed_data_df$V7[imputed_data_df$V7 == 0] <- 1
summary(imputed_data_df)
```

```
##        V1                 V2               V3               V4        
##  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  
##        V5               V6               V7               V8        
##  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   : 3.535   Mean   : 3.438  
##  3rd Qu.: 4.000   3rd Qu.: 4.000   3rd Qu.: 5.500   3rd Qu.: 5.000  
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000  
##        V9              V10              V11        
##  Min.   : 1.000   Min.   : 1.000   Min.   :0.0000  
##  1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.:0.0000  
##  Median : 1.000   Median : 1.000   Median :0.0000  
##  Mean   : 2.867   Mean   : 1.589   Mean   :0.3448  
##  3rd Qu.: 4.000   3rd Qu.: 1.000   3rd Qu.:1.0000  
##  Max.   :10.000   Max.   :10.000   Max.   :1.0000  
```

Question 15.1

I work at a grocey store and I think that optimization will be applicable to supply chain decisions. Given an item's price, manufacturer's location, distribution centers, store location, and the cost to transport to the store. Our store would probably need a optimization model to calculate the best option for products to get to the store.