Question 14.1 - Initial analysis

1. Read table and assign column names

```
rm(list = ls())
cancer <- read.csv("breast-cancer-wisconsin.data.txt", stringsAsFactors = FALSE, header
= FALSE, sep = ",")

colnames(cancer) <- c("Sample_code_number", "Clump_Thickness", "Uniformity_of_cell_siz
e", "Uniformity_of_cell_shape", "Marginal_Adhesion", "Single_Epith_Cell_Size", "Bare_Nuc
lei", "Bland_Chromatin", "Normal_Nucleoli", "Mitoses", "Class")
```

2. Initital look: how much data is missing

```
missingvalues <- which(cancer$Bare_Nuclei == "?")
missingvalues_percent <- length(missingvalues)/nrow(cancer)

# < 5% data is missing, which is not too much
print(missingvalues_percent)
```

```
## [1] 0.02288984
```

3. Check if data is biased in some way

We can compare Class == 2's distribution between each table, cancer_missingvalues (with mssing values) and cancer_no_missingvalues (without missing values).

```
# data set without missing data
cancer_missingvalues <- cancer[missingvalues,]

# data set with missing data
cancer_no_missingvalues <- cancer[-missingvalues,]

# check how much Class = 2 represent in the table with missing values
sum(cancer_missingvalues$Class == 2) / nrow(cancer_missingvalues)
```

```
## [1] 0.875
```

```
# check how much Class = 2 represent in the table without missing values
sum(cancer_no_missingvalues$Class == 2) / nrow(cancer_no_missingvalues)
```

```
## [1] 0.6500732
```

The result shows that Class = 2 makes up 87.5% in the cancer table with missing values versus 65% in the cancer table without missing values. This might suggest that there is a potential bias associated with missing values in the "Bare_Nuclei" column, particularly towards instances classified as Class 2. We can dive deeper into

the mechanisms causing missing values and its relationship with “Class” to understand the extent of this bias and its implications for interpretation.

Question 14.1.1 - Use the mean/mode imputation method to impute values for the missing data.

1. Get mean and mode of Bare_Nuclei from the table without missing values

Mode imputation - handles missing data where missing values are replaced with the mode of the variable, the most frequently occurring value in the dataset. Mode imputation is commonly used for categorical or ordinal variables where missing values can be reasonably replaced by the most common value.

Mean imputation - handles missing data where missing values are replaced with the mean of the non-missing values in the dataset.

```
# get mean of Bare_Nuclei from table without missing values
mean_Bare_Nuclei <- round(mean(as.integer(cancer_no_missingvalues$Bare_Nuclei)))
print(mean_Bare_Nuclei)
```

```
## [1] 4
```

```
# create a function to get mode
mode_function <- function(x){
  which.max(tabulate(x))
  }

# get mode of Bare_Nuclei from table without missing values
mode_Bare_Nuclei <- mode_function(as.integer(cancer_no_missingvalues$Bare_Nuclei))
print(mode_Bare_Nuclei)
```

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

The result shows us the mode of Bare_Nuclei is 1, while the mean (rounded) is 4. 2 very distinct values. This might yield different results in later analysis.

2. Replace the NAs with mean and mode values, saved as 2 different tables for later analysis

```
# create a copied cancer table and replace ? in column "Bare_Nuclei" with mean value
cancer_copy <- cancer
cancer_copy$mean_method <- ifelse(cancer_copy$Bare_Nuclei == "?", mean_Bare_Nuclei, canc
er_copy$Bare_Nuclei)

# create a copied cancer table and replace ? in column "Bare_Nuclei" with mode value
cancer_copy$mode_method <- ifelse(cancer_copy$Bare_Nuclei == "?", mode_Bare_Nuclei, canc
er_copy$Bare_Nuclei)

# see the output
cancer_copy[cancer_copy$Bare_Nuclei == "?", c("Bare_Nuclei", "mean_method", "mode_metho
d")]
```

```
##     Bare_Nuclei mean_method mode_method
## 24            ?           4           1
## 41            ?           4           1
## 140           ?           4           1
## 146           ?           4           1
## 159           ?           4           1
## 165           ?           4           1
## 236           ?           4           1
## 250           ?           4           1
## 276           ?           4           1
## 293           ?           4           1
## 295           ?           4           1
## 298           ?           4           1
## 316           ?           4           1
## 322           ?           4           1
## 412           ?           4           1
## 618           ?           4           1
```

Question 14.1.2 - Use regression to impute values for the missing data

1.Linear regression is efficient and can handle large datasets with multiple predictor variables relatively quickly.

```
set.seed(123)

# change the Bare_Nuclei to numeric, and "?" to NA
cancer_copy$Bare_Nuclei <- as.numeric(ifelse(cancer_copy$Bare_Nuclei == "?", NA, cancer_
copy$Bare_Nuclei))

# create a linear regression model
lm_model <- lm(Bare_Nuclei ~ Clump_Thickness + Uniformity_of_cell_size + Uniformity_of_c
ell_shape + Marginal_Adhesion + Single_Epith_Cell_Size + Bland_Chromatin + Normal_Nucleo
li + Mitoses, data = cancer_copy)

# use predict to impute missing values
predicted_values <- round(predict(lm_model, newdata = cancer_copy))

# replace missing values with predicted values
cancer_copy$predicted_values <- predicted_values

missing_values <- is.na(cancer_copy$Bare_Nuclei)
cancer_copy[missing_values, c("Bare_Nuclei", "predicted_values")]
```

```
##     Bare_Nuclei predicted_values
## 24           NA                5
## 41           NA                8
## 140          NA                1
## 146          NA                2
## 159          NA                1
## 165          NA                2
## 236          NA                3
## 250          NA                2
## 276          NA                2
## 293          NA                6
## 295          NA                1
## 298          NA                2
## 316          NA                6
## 322          NA                2
## 412          NA                1
## 618          NA                1
```

2. Now I want to calculate the performance of the model

```
# extract the actual values from cancer_regression table (copy of cancer table) without
the NA values
true_values <- cancer_copy$Bare_Nuclei[!is.na(cancer_copy$Bare_Nuclei)]

# extract predicted values where true values were missing
predicted_values_outcome <- predicted_values[!is.na(true_values)]

# calculate RMSE
rmse <- sqrt(mean((true_values - predicted_values_outcome)^2))
```

```
## Warning in true_values - predicted_values_outcome: longer object length is not
## a multiple of shorter object length
```

```
print(rmse)
```

```
## [1] 4.441803
```

Since our Bare_Nuclei scale from 1 to 10 mostly, an RMSE of 4.441803 may be acceptable for this purpose.

Question 14.1.3 - Use regression with perturbation to impute values for the missing data

```
reg_imputed <- round(predict(lm_model, newdata = cancer_copy))

perturb <- rnorm(length(predicted_values_outcome), 0, 1)

# perturb the imputed values
perturbed_imputed <- round(reg_imputed + perturb)

# make sure the perturbed values are within the range of 1 and 10
perturbed_imputed[perturbed_imputed < 1] <- 1
perturbed_imputed[perturbed_imputed > 10] <- 10

# replace missing values with perturbed imputed values
cancer_copy$perturbed_values <- perturbed_imputed

cancer_copy[missing_values, c("Bare_Nuclei", "perturbed_values")]
```

```
##     Bare_Nuclei perturbed_values
## 24           NA                4
## 41           NA                7
## 140          NA                1
## 146          NA                1
## 159          NA                2
## 165          NA                2
## 236          NA                2
## 250          NA                2
## 276          NA                2
## 293          NA                7
## 295          NA                3
## 298          NA                1
## 316          NA                7
## 322          NA                3
## 412          NA                1
## 618          NA                1
```

Question 14.1 - Final put together

```
cancer_copy[missing_values, c("Bare_Nuclei", "mean_method", "mode_method", "predicted_va
lues", "perturbed_values")]
```

```
##     Bare_Nuclei mean_method mode_method predicted_values perturbed_values
## 24           NA           4           1                5                4
## 41           NA           4           1                8                7
## 140          NA           4           1                1                1
## 146          NA           4           1                2                1
## 159          NA           4           1                1                2
## 165          NA           4           1                2                2
## 236          NA           4           1                3                2
## 250          NA           4           1                2                2
## 276          NA           4           1                2                2
## 293          NA           4           1                6                7
## 295          NA           4           1                1                3
## 298          NA           4           1                2                1
## 316          NA           4           1                6                7
## 322          NA           4           1                2                3
## 412          NA           4           1                1                1
## 618          NA           4           1                1                1
```

Question 14.1.4 (Optional) - Compare the results and quality of classification models (e.g., SVM, KNN) build

Question 14.1.4 (1) Usingthe data sets from questions 1,2,3

1. Split the data into train data, validation data, and test data, following 60-20-20 splitting rule

```
set.seed(123)

cancer_copy$new_class = ifelse(cancer_copy$Class == 2, 0, 1)

# good rule of thumb to split the data 60-20-20
split <- sample(1:3, nrow(cancer_copy), replace = TRUE, prob = c(0.6, 0.2, 0.2))

train_data <- cancer_copy[split == 1, ]
validation_data <- cancer_copy[split == 2, ]
test_data <- cancer_copy[split == 3, ]
```

2. Find accuracy for using mean_menthod dataset

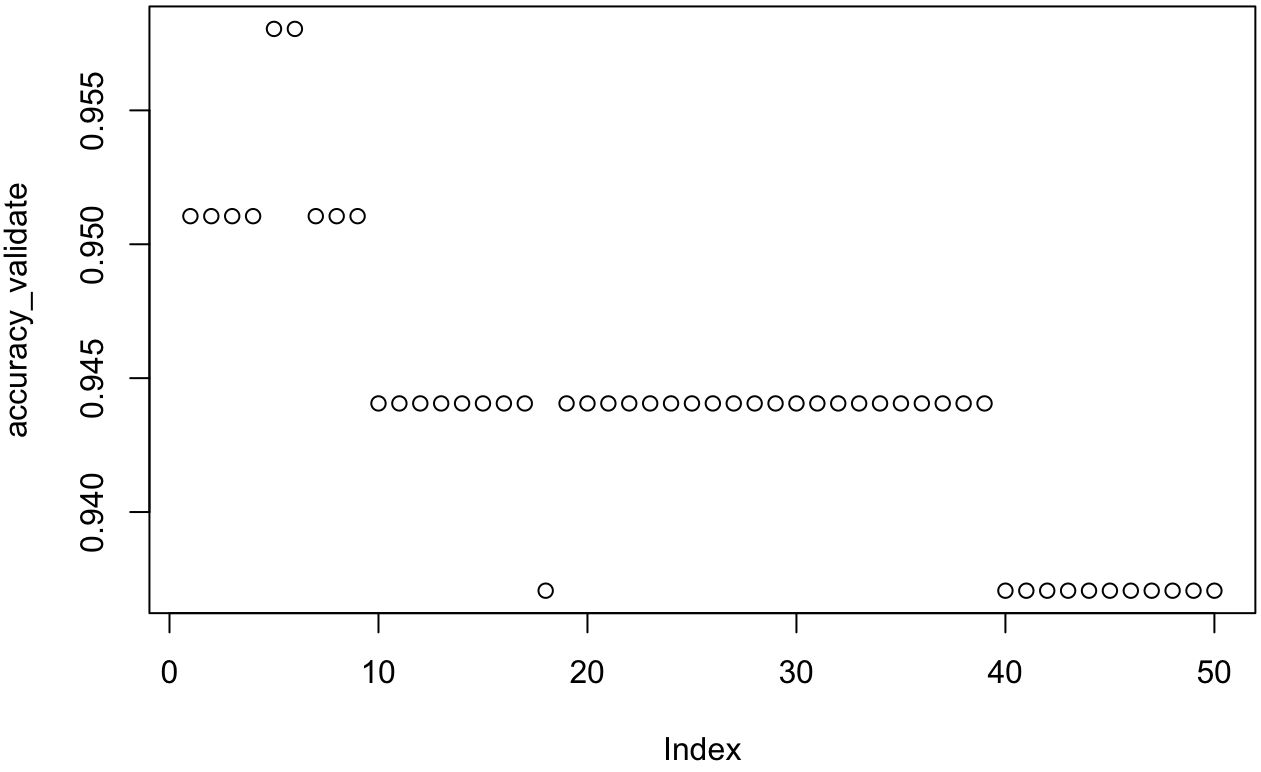
```
set.seed(123)

# create a list of k values to loop through. Each k is the number of neighbors
k_list = c(seq(from = 1, to = 50, by = 1))

# create an empty list to store the result
accuracy_validate <- numeric(length(k_list))

for (k_value in 1: max(k_list)) {
  knn_model <- kknn(new_class~
                      mean_method + # use mean_method as a factor
                      Clump_Thickness +
                      Uniformity_of_cell_size +
                      Uniformity_of_cell_shape +
                      Marginal_Adhesion +
                      Single_Epith_Cell_Size +
                      Bland_Chromatin +
                      Normal_Nucleoli +
                      Mitoses,
                    train_data,
                    validation_data,
                    k = k_value,
                    scale = TRUE)

  # round up the predicted values to 0 or 1
  predicted <- as.integer(fitted(knn_model) + 0.5)

  # check the accuracy of the prediction for each k
  accuracy_validate[k_value] <- sum(predicted == validation_data$new_class) / nrow(valid
ation_data)
}

max(accuracy_validate)
```

```
## [1] 0.958042
```

```
which.max(accuracy_validate)
```

```
## [1] 5
```

```
plot(accuracy_validate)
```



2. Find accuracy for using mode_menthod dataset

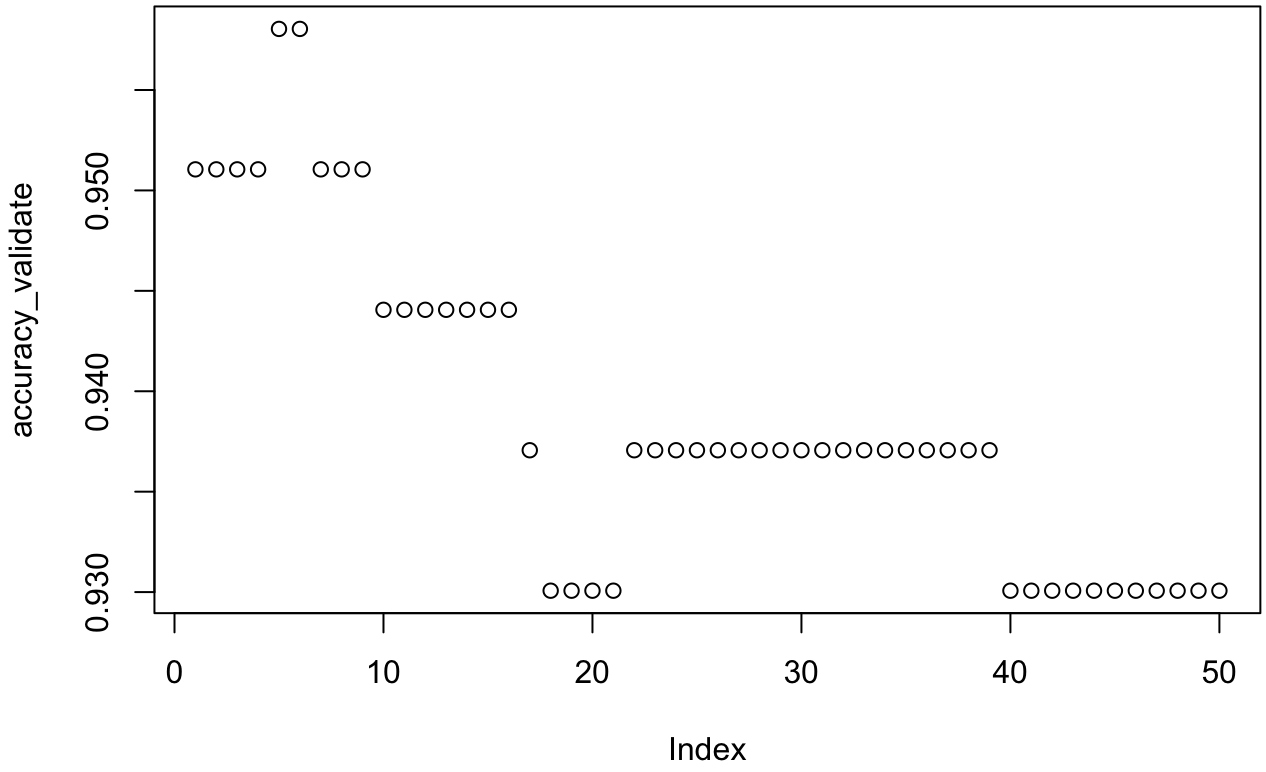
```
set.seed(123)

# create a list of k values to loop through. Each k is the number of neighbors
k_list = c(seq(from = 1, to = 50, by = 1))

# create an empty list to store the result
accuracy_validate <- numeric(length(k_list))

for (k_value in 1: max(k_list)) {
  knn_model <- kknn(new_class~
                      mode_method + # use mode_method as a factor
                      Clump_Thickness +
                      Uniformity_of_cell_size +
                      Uniformity_of_cell_shape +
                      Marginal_Adhesion +
                      Single_Epith_Cell_Size +
                      Bland_Chromatin +
                      Normal_Nucleoli +
                      Mitoses,
                    train_data,
                    validation_data,
                    k = k_value,
                    scale = TRUE)

  # round up the predicted values to 0 or 1
  predicted <- as.integer(fitted(knn_model) + 0.5)

  # check the accuracy of the prediction for each k
  accuracy_validate[k_value] <- sum(predicted == validation_data$new_class) / nrow(valid
ation_data)
}

max(accuracy_validate)
```

```
## [1] 0.958042
```

```
which.max(accuracy_validate)
```

```
## [1] 5
```

```
plot(accuracy_validate)
```



2. Find accuracy for using regression dataset

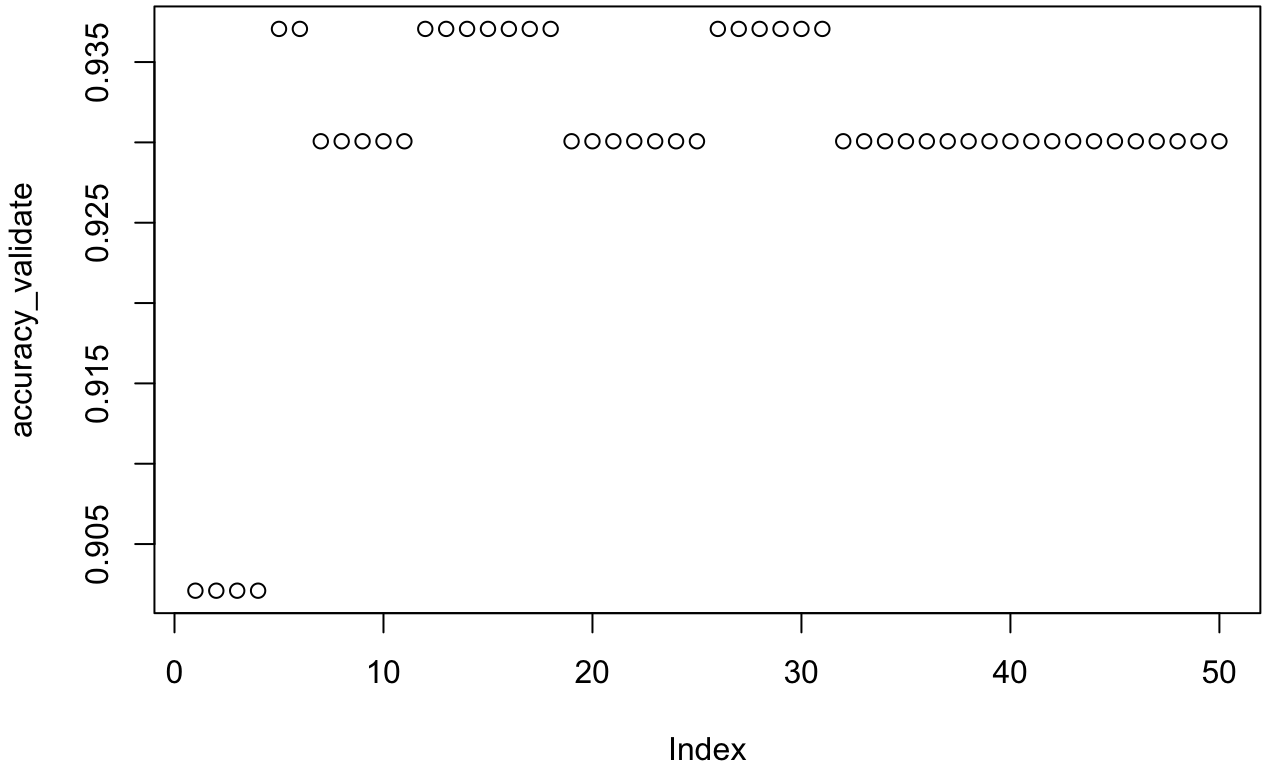
```
set.seed(123)

# create a list of k values to loop through. Each k is the number of neighbors
k_list = c(seq(from = 1, to = 50, by = 1))

# create an empty list to store the result
accuracy_validate <- numeric(length(k_list))

for (k_value in 1: max(k_list)) {
  knn_model <- kknn(new_class~
                      predicted_values + # use predicted_values as a factor
                      Clump_Thickness +
                      Uniformity_of_cell_size +
                      Uniformity_of_cell_shape +
                      Marginal_Adhesion +
                      Single_Epith_Cell_Size +
                      Bland_Chromatin +
                      Normal_Nucleoli +
                      Mitoses,
                    train_data,
                    validation_data,
                    k = k_value,
                    scale = TRUE)

  # round up the predicted values to 0 or 1
  predicted <- as.integer(fitted(knn_model) + 0.5)

  # check the accuracy of the prediction for each k
  accuracy_validate[k_value] <- sum(predicted == validation_data$new_class) / nrow(valid
ation_data)
}

max(accuracy_validate)
```

```
## [1] 0.9370629
```

```
which.max(accuracy_validate)
```

```
## [1] 5
```

```
plot(accuracy_validate)
```



4. Find accuracy for using regression with perturbation dataset

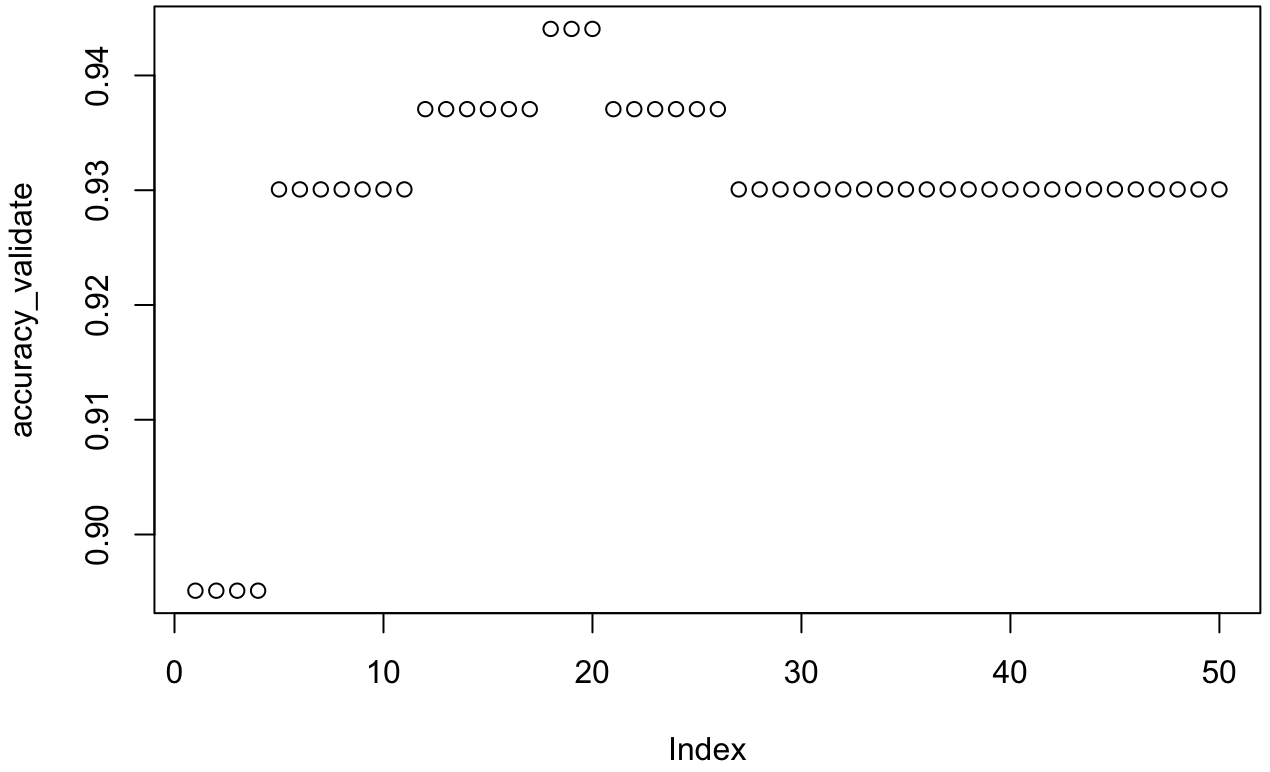
```
set.seed(123)

# create a list of k values to loop through. Each k is the number of neighbors
k_list = c(seq(from = 1, to = 50, by = 1))

# create an empty list to store the result
accuracy_validate <- numeric(length(k_list))

for (k_value in 1: max(k_list)) {
  knn_model <- kknn(new_class~
                      perturbed_values + # use perturbed_values as a factor
                      Clump_Thickness +
                      Uniformity_of_cell_size +
                      Uniformity_of_cell_shape +
                      Marginal_Adhesion +
                      Single_Epith_Cell_Size +
                      Bland_Chromatin +
                      Normal_Nucleoli +
                      Mitoses,
                    train_data,
                    validation_data,
                    k = k_value,
                    scale = TRUE)

  # round up the predicted values to 0 or 1
  predicted <- as.integer(fitted(knn_model) + 0.5)

  # check the accuracy of the prediction for each k
  accuracy_validate[k_value] <- sum(predicted == validation_data$new_class) / nrow(valid
ation_data)
}

max(accuracy_validate)
```

```
## [1] 0.9440559
```

```
which.max(accuracy_validate)
```

```
## [1] 18
```

```
plot(accuracy_validate)
```



Question 14.1.4 (2) Using the data that remains after data points with missing values are removed

1. Split the data into train data, validation data, and test data, following 60-20-20 splitting rule

```
set.seed(123)

# cancer_no_missingvalues is previously declared to no include rows without missing valu
es
cancer_no_missingvalues$new_class = ifelse(cancer_no_missingvalues$Class == 2, 0, 1)

# good rule of thumb to split the data 60-20-20
split <- sample(1:3, nrow(cancer_no_missingvalues), replace = TRUE, prob = c(0.6, 0.2,
0.2))

train_data <- cancer_no_missingvalues[split == 1, ]
validation_data <- cancer_no_missingvalues[split == 2, ]
test_data <- cancer_no_missingvalues[split == 3, ]
```

2. Find accuracy for using dataset without missing values

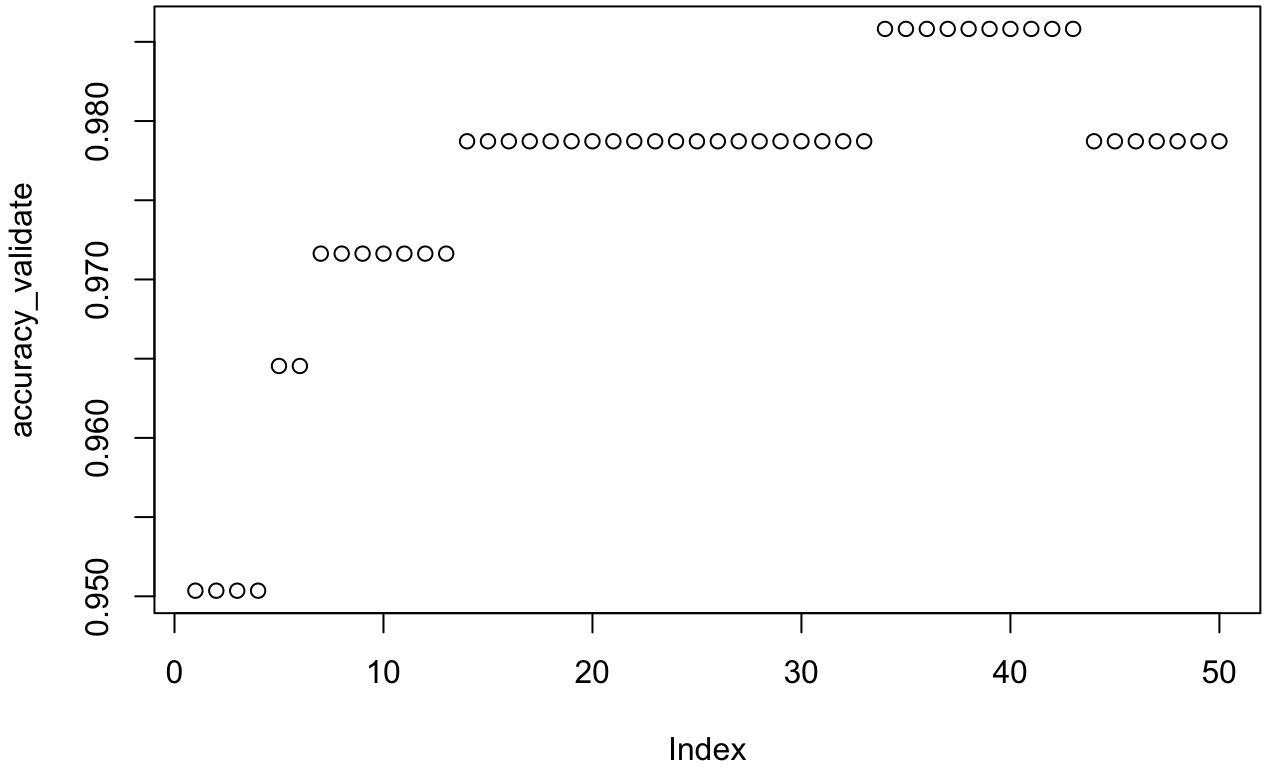
```
set.seed(123)

# create a list of k values to loop through. Each k is the number of neighbors
k_list = c(seq(from = 1, to = 50, by = 1))

# create an empty list to store the result
accuracy_validate <- numeric(length(k_list))

for (k_value in 1: max(k_list)) {
  knn_model <- kknn(new_class~
                      as.numeric(Bare_Nuclei) + # use Bare_Nuclei as a factor, this excl
udes missing data
                      Clump_Thickness +
                      Uniformity_of_cell_size +
                      Uniformity_of_cell_shape +
                      Marginal_Adhesion +
                      Single_Epith_Cell_Size +
                      Bland_Chromatin +
                      Normal_Nucleoli +
                      Mitoses,
                    train_data,
                    validation_data,
                    k = k_value,
                    scale = TRUE)

  # round up the predicted values to 0 or 1
  predicted <- as.integer(fitted(knn_model) + 0.5)

  # check the accuracy of the prediction for each k
  accuracy_validate[k_value] <- sum(predicted == validation_data$new_class) / nrow(valid
ation_data)
}

max(accuracy_validate)
```

```
## [1] 0.9858156
```

```
which.max(accuracy_validate)
```

```
## [1] 34
```

```
plot(accuracy_validate)
```



Question 14.1.4 (3) Using the data set when a binary variable is introduced to indicate missing values

1. Split the data into train data, validation data, and test data, following 60-20-20 splitting rule

```
set.seed(123)

# cancer_no_missingvalues is previously declared to no include rows without missing valu
es
cancer_copy$binary_value = ifelse(is.na(cancer_copy$Bare_Nucle), 0, 1)

# good rule of thumb to split the data 60-20-20
split <- sample(1:3, nrow(cancer_copy), replace = TRUE, prob = c(0.6, 0.2, 0.2))

train_data <- cancer_copy[split == 1, ]
validation_data <- cancer_copy[split == 2, ]
test_data <- cancer_copy[split == 3, ]
```

2. Find accuracy for using dataset without missing values

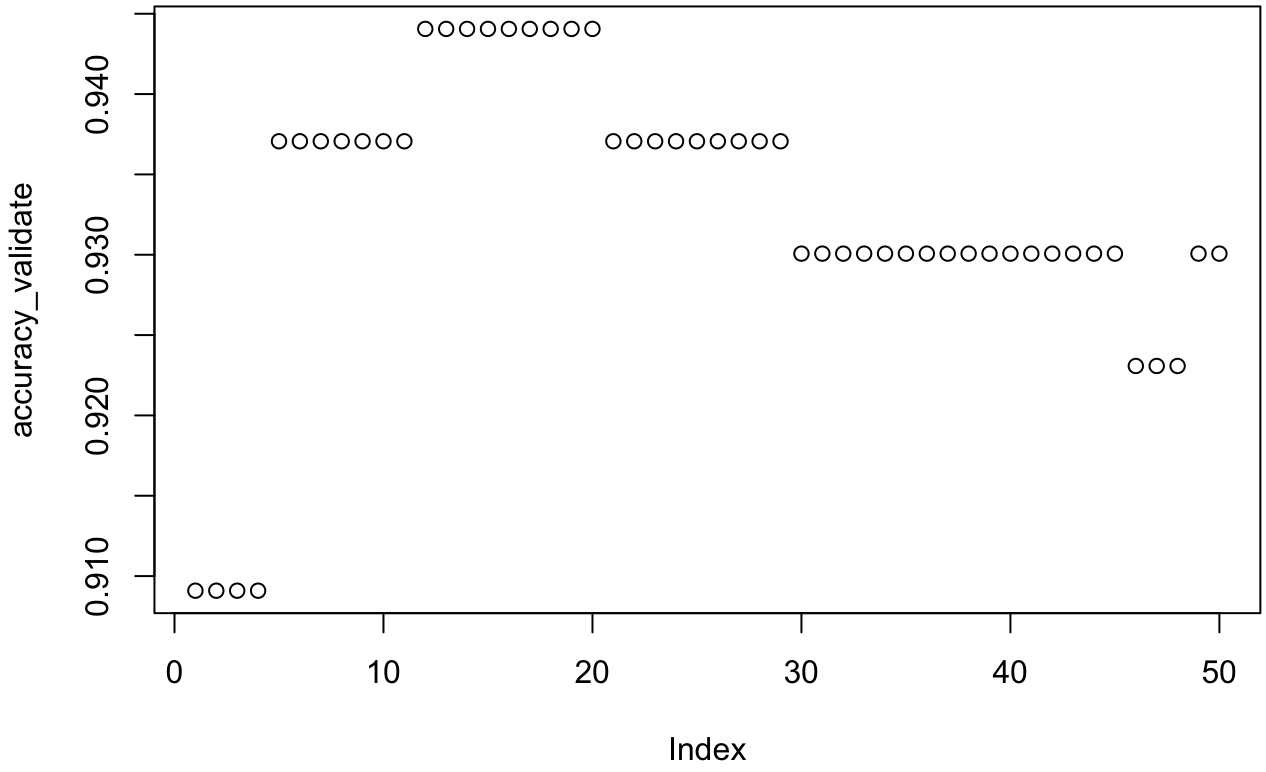
```
set.seed(123)

# create a list of k values to loop through. Each k is the number of neighbors
k_list = c(seq(from = 1, to = 50, by = 1))

# create an empty list to store the result
accuracy_validate <- numeric(length(k_list))

for (k_value in 1: max(k_list)) {
  knn_model <- kknn(new_class~
                      binary_value + # use binary value as a factor
                      Clump_Thickness +
                      Uniformity_of_cell_size +
                      Uniformity_of_cell_shape +
                      Marginal_Adhesion +
                      Single_Epith_Cell_Size +
                      Bland_Chromatin +
                      Normal_Nucleoli +
                      Mitoses,
                    train_data,
                    validation_data,
                    k = k_value,
                    scale = TRUE)

  # round up the predicted values to 0 or 1
  predicted <- as.integer(fitted(knn_model) + 0.5)

  # check the accuracy of the prediction for each k
  accuracy_validate[k_value] <- sum(predicted == validation_data$new_class) / nrow(valid
ation_data)
}

max(accuracy_validate)
```

```
## [1] 0.9440559
```

```
which.max(accuracy_validate)
```

```
## [1] 12
```

```
plot(accuracy_validate)
```



All models seem to produce a pretty high accuracy. This may indicate the models are overfitting. Removing the data points with the missing values shows a even better accuracy. In the real world, I would build 2 models, one with missing data removed, and one using with mean imputation or regression imputation. Mean imputatio is easy to implement, and it preserves the original distribution of the non-missing values. Regression model on the other hand takes into account relationships between variables, potentially leading to more accurate imputations.

Question 15.1

I work as an Analyst in the HR department. Optimization can be used to streamline the recruitment process to efficiently identify and hire the most qualified candidates while minimizing time-to-fill and cost-per-hire.

Data Needed:

1. Job Descriptions: Detailed descriptions of job roles, responsibilities, required qualifications, and desired skills.
2. Candidate Data: Information about applicants, including resumes, education, work experience, skills, and contact details.
3. Recruitment Source Data: Data on the effectiveness of various recruitment sources (e.g., job boards, social media, employee referrals) in attracting qualified candidates.
4. Interview Data: Feedback from interviewers, assessment scores, and evaluations of candidate performance during interviews.
5. Time-to-Fill Data: Metrics tracking the time it takes to fill vacant positions from the time the job posting is published to when the candidate accepts the offer.

Citation: 1. https://www.tutorialspoint.com/how-to-find-mode-for-an-r-data-frame-column (https://www.tutorialspoint.com/how-to-find-mode-for-an-r-data-frame-column)