HW3: NBC, KNN, SVM, and Ensemble Learning

Eunmi(Ellie) Jeong

2020 11 01

Section 1. Data preparation

1-1. Data preparation-training data

Data cleaning to remove data quality issues (missing data, duplicate data, biased data, outliers/noises, attributes with no or low variance)

```
data <- read.csv("Disease Prediction Training.csv")
str(data) #49000 obs. 12 vars
summary(data)
```

- Check missing data

```
sum(is.na(data)) #no missing data
```

```
[1] 0
```

- Check and remove duplicate data record

```
nrow(data[duplicated(data), ]) #1752 duplicates
data[duplicated(data)|duplicated(data, fromLast = T), ]
data <- data %>%
  distinct(.keep_all = T)
```

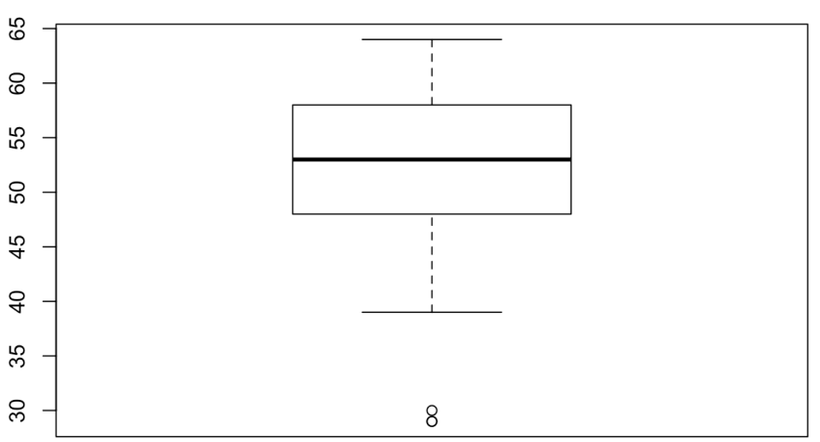
- Remove low or no variance

```
caret::nearZeroVar(data, saveMetrics = T) #none
```

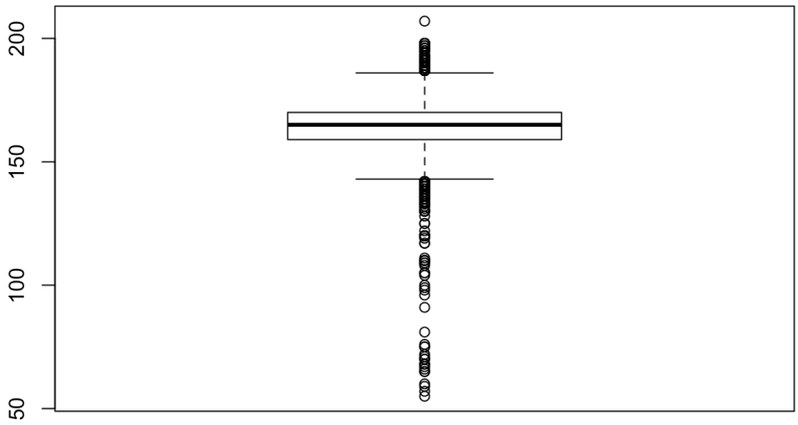
```
                    freqRatio percentUnique zeroVar   nzv
Age                  1.046990   0.059261768   FALSE FALSE
Gender               1.816908   0.004232983   FALSE FALSE
Height               1.127329   0.213765662   FALSE FALSE
Weight               1.058977   0.524889942   FALSE FALSE
High.Blood.Pressure  2.683583   0.302658314   FALSE FALSE
Low.Blood.Pressure   2.284982   0.302658314   FALSE FALSE
Cholesterol          5.220281   0.006349475   FALSE FALSE
Glucose             10.786544   0.006349475   FALSE FALSE
Smoke                9.939569   0.004232983   FALSE FALSE
Alcohol             16.782461   0.004232983   FALSE FALSE
Exercise             3.984492   0.004232983   FALSE FALSE
Disease              1.039013   0.004232983   FALSE FALSE
```

- Identify&remove noises & outlier

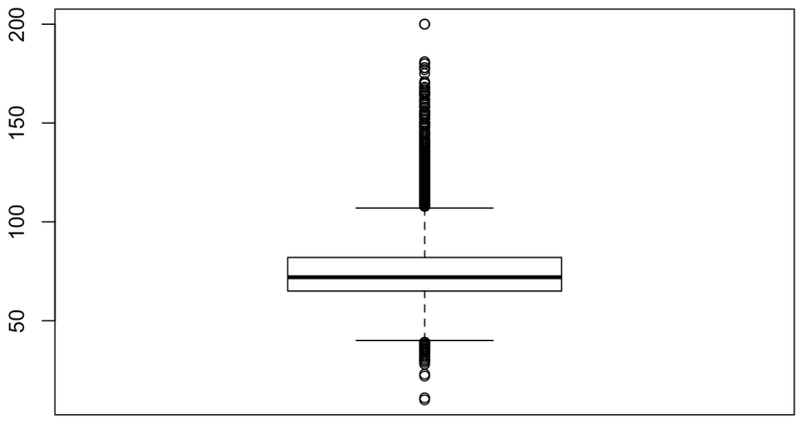
```
boxplot(data$Age)
```



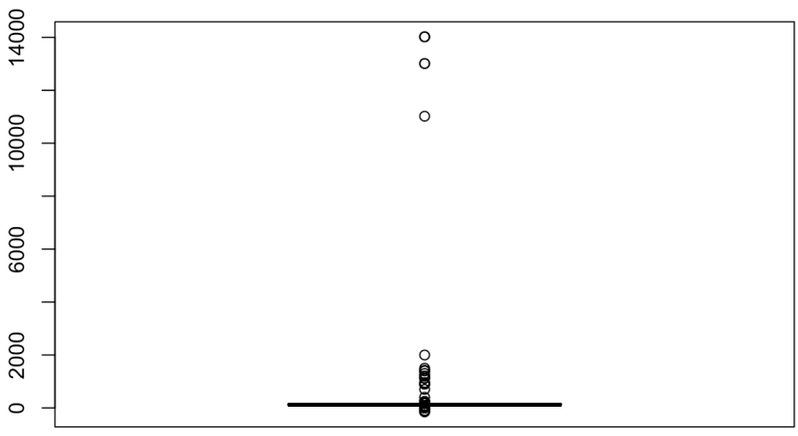

```
data$Age[data$Age %in% boxplot.stats(data$Age)$out] <- median(data$Age, na.rm = T)
boxplot(data$Height)
```



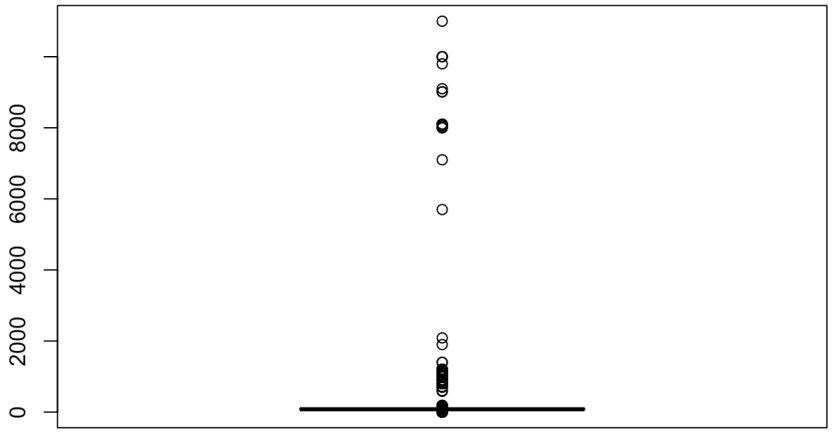

```
data$Height[data$Height %in% boxplot.stats(data$Height)$out] <- median(data$Height, na.rm = T)
boxplot(data$Weight) #twice
```




```
data$Weight[data$Weight %in% boxplot.stats(data$Weight)$out] <- median(data$Weight, na.rm = T)
data$Weight[data$Weight %in% boxplot.stats(data$Weight)$out] <- median(data$Weight, na.rm = T)
boxplot(data$High.Blood.Pressure)
```




```
data$High.Blood.Pressure[data$High.Blood.Pressure %in% boxplot.stats(data$High.Blood.Pressure)$out] <- median(dat
a$High.Blood.Pressure, na.rm = T)
boxplot(data$Low.Blood.Pressure)
```




```
data$Low.Blood.Pressure[data$Low.Blood.Pressure %in% boxplot.stats(data$Low.Blood.Pressure)$out] <- median(data$L
ow.Blood.Pressure, na.rm = T)
```

- Change attribute type

```
data$Smoke <- as.factor(data$Smoke)
data$Alcohol <- as.factor(data$Alcohol)
data$Exercise <- as.factor(data$Exercise)
data$Disease <- as.factor(data$Disease)
```

- Discretization: necessary for Naive Bayes Classifier

```
data$age_grp <- arules::discretize(data$Age, method = "interval", breaks = 3, labels = c("low", "middle", "high")
)
data %>%
  group_by(age_grp) %>%
    summarise(avg = mean(Age), count = n(), min = min(Age), max = max(Age))
```

```
# A tibble: 3 x 5
  age_grp   avg count   min   max
  <fct>   <dbl> <int> <dbl> <dbl>
1 low      43.1 10958    39    47
2 middle   51.8 17760    48    55
3 high     59.6 18530    56    64
```

```
data$height_grp <- arules::discretize(data$Height, method = "interval", breaks = 3, labels = c("low", "middle", "
high"))
data %>%
  group_by(height_grp) %>%
  summarise(avg = mean(Height), count = n(), min = min(Height), max = max(Height))
```

```
# A tibble: 3 x 5
  height_grp   avg count   min   max
  <fct>      <dbl> <int> <dbl> <dbl>
1 low         154.  8961   143   157
2 middle      165. 30163   158   171
3 high        176.  8124   172   186
```

```
data$weight_grp <- arules::discretize(data$Weight, method = "interval", breaks = 3, labels = c("low", "middle", "
high"))
data %>%
  group_by(weight_grp) %>%
  summarise(avg = mean(Weight), count = n(), min = min(Weight), max = max(Weight))
```

```
# A tibble: 3 x 5
  weight_grp   avg count   min   max
  <fct>      <dbl> <int> <dbl> <dbl>
1 low         57.0  9048  41    62.3
2 middle      72.2 28563  62.4  83.5
3 high        91.4  9637  84   105
```

```
data$h_blood_grp <- arules::discretize(data$High.Blood.Pressure, method = "interval", breaks = 3, labels = c("low
", "middle", "high"))
data %>%
  group_by(h_blood_grp) %>%
  summarise(avg = mean(High.Blood.Pressure), count = n(), min = min(High.Blood.Pressure), max = max(High.Blood.Pr
essure))
```

```
# A tibble: 3 x 5
  h_blood_grp   avg count   min   max
  <fct>       <dbl> <int> <dbl> <dbl>
1 low          106.  8743    90   116
2 middle       126. 32382   117   143
3 high         155.  6123   144   170
```

```
data$l_blood_grp <- arules::discretize(data$Low.Blood.Pressure, method = "interval", breaks = 3, labels = c("low"
, "middle", "high"))
data %>%
  group_by(l_blood_grp) %>%
  summarise(avg = mean(Low.Blood.Pressure), count = n(), min = min(Low.Blood.Pressure), max = max(Low.Blood.Press
ure))
```

```
# A tibble: 3 x 5
  l_blood_grp   avg count   min   max
  <fct>       <dbl> <int> <dbl> <dbl>
1 low          70.1  7502    65    78
2 middle       82.8 36616    79    91
3 high         99.7  3130    92   105
```

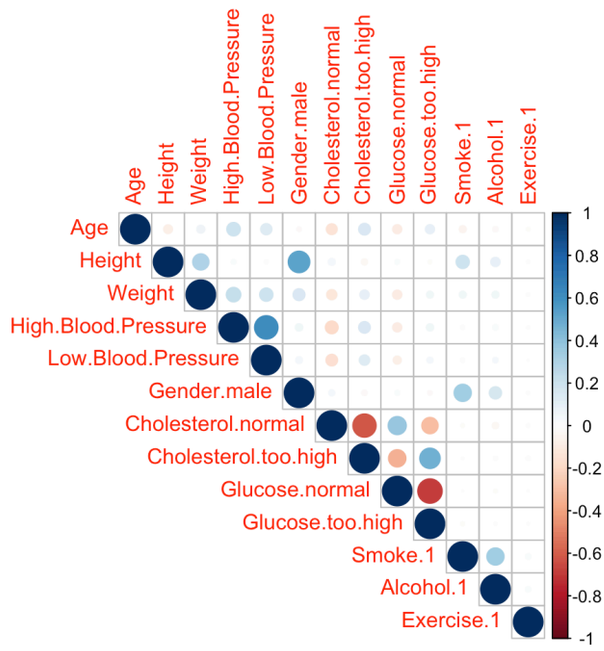
```
training_cat <- data[sapply(data, is.factor)]
```

- Numerization: necessary for KNN, RF(Random Forest), GBM(Gradient Boosting Machine), SVM(linear & non-linear)
- Standardization & Normalization: necessary for numerical values which will be used in KNN, RF(Random Forest), GBM(Gradient Boosting Machine), SVM(linear & non-linear)

```
dmy <- caret::dummyVars(~Gender+Cholesterol+Glucose+Smoke+Alcohol+Exercise,data = data, fullRank = T)
dummy <- data.frame(predict(dmy, newdata = data))
subset <- subset(data,select= c(Age, Height, Weight, High.Blood.Pressure, Low.Blood.Pressure, Disease))
training_num <- cbind(subset, dummy)
pre_process <- preProcess(training_num, method = c("scale", "center"))
training_num_s <- predict(pre_process, newdata = training_num)
```

- Check attribute's correlation
  - Positive relationship: Glucose-Cholesterol, Cholesterol-high/low blood presssure, male-smoke, male-alcohol, male-height, alcohol-smoke, height-weight, hight blood pressure-low blood pressure
  - Negative relationship: age-Cholesterol, age-height

```
cor_matrix <- cor(training_num_s [complete.cases(training_num_s ), sapply(training_num_s , is.numeric)], method =
"pearson")
corrplot(cor_matrix, type = "upper")
```



1-2. Data preparation-testing data

Data cleaning to remove data quality issues (missing data, duplicate data, biased data, outliers/noises, attributes with no or low variance)

```
data2 <- read.csv("Disease Prediction Testing.csv")
str(data2) #21000 obs. 12 vars
```

- Check missing data

```
sum(is.na(data2)) #no missing data
```

```
[1] 0
```

- Check and remove duplicate data record

```
nrow(data2[duplicated(data2), ]) #no duplicates
```

```
[1] 0
```

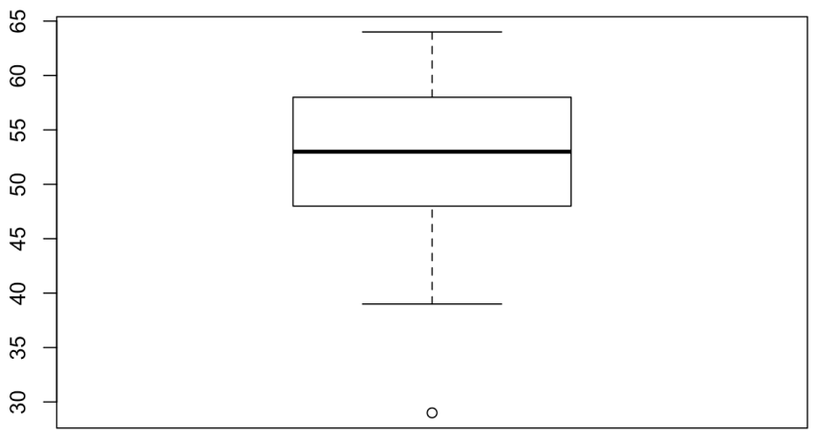
- Remove low or no variance

```
caret::nearZeroVar(data2, saveMetrics = T) #none
```

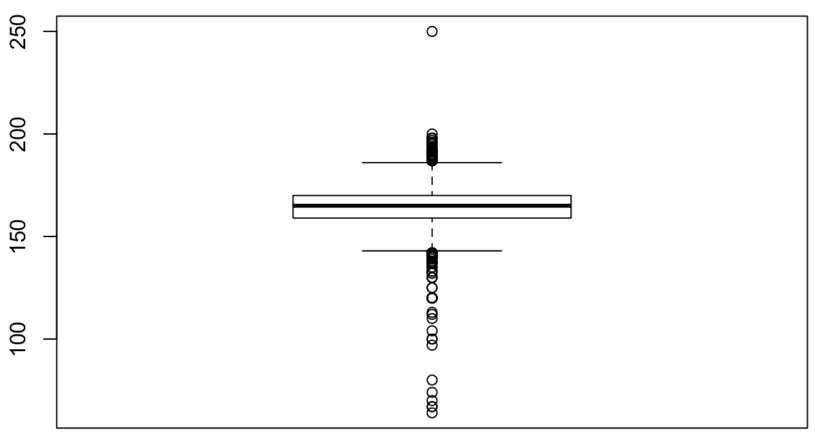
```
                    freqRatio percentUnique zeroVar   nzv
ID                   1.000000  100.00000000   FALSE FALSE
Age                  1.011354    0.12857143   FALSE FALSE
Gender               1.863767    0.00952381   FALSE FALSE
Height               1.148675    0.38571429   FALSE FALSE
Weight               1.014706    0.86666667   FALSE FALSE
High.Blood.Pressure  2.944093    0.54761905   FALSE FALSE
Low.Blood.Pressure   2.442870    0.43809524   FALSE FALSE
Cholesterol          5.523558    0.01428571   FALSE FALSE
Glucose             11.072671    0.01428571   FALSE FALSE
Smoke               10.388286    0.00952381   FALSE FALSE
Alcohol             17.987342    0.00952381   FALSE FALSE
Exercise             4.126953    0.00952381   FALSE FALSE
```

- Identify&remove noises & outlier

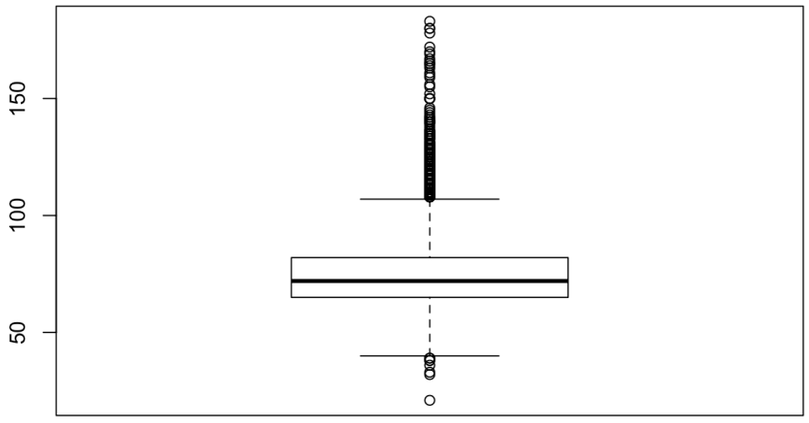
```
boxplot(data2$Age)
```



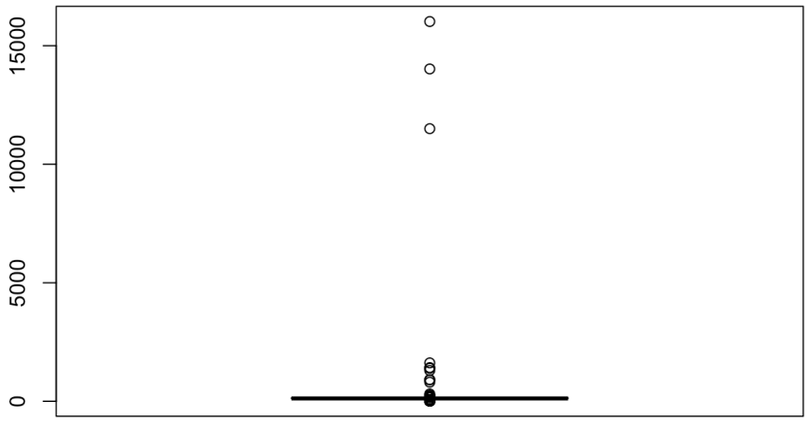

```
data2$Age[data2$Age %in% boxplot.stats(data2$Age)$out] <- median(data2$Age, na.rm = T)
boxplot(data2$Height)
```



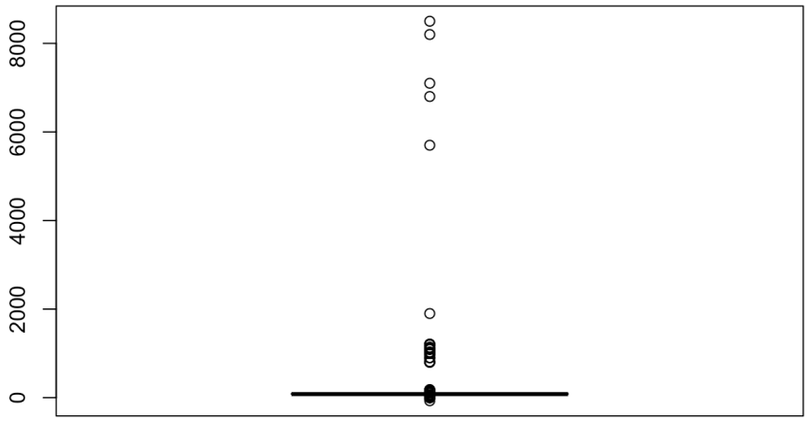

```
data2$Height[data2$Height %in% boxplot.stats(data2$Height)$out] <- median(data2$Height, na.rm = T)
boxplot(data2$Weight) #twice
```




```
data2$Weight[data2$Weight %in% boxplot.stats(data2$Weight)$out] <- median(data2$Weight, na.rm = T)
boxplot(data2$High.Blood.Pressure)
data2$Weight[data2$Weight %in% boxplot.stats(data2$Weight)$out] <- median(data2$Weight, na.rm = T)
boxplot(data2$High.Blood.Pressure)
```




```
data2$High.Blood.Pressure[data2$High.Blood.Pressure %in% boxplot.stats(data2$High.Blood.Pressure)$out] <- median(
data2$High.Blood.Pressure, na.rm = T)
boxplot(data2$Low.Blood.Pressure)
```




```
data2$Low.Blood.Pressure[data2$Low.Blood.Pressure %in% boxplot.stats(data2$Low.Blood.Pressure)$out] <- median(dat
a2$Low.Blood.Pressure, na.rm = T)
```

- Change attribute type

```
data2$ID <- as.factor(data2$ID)
data2$Smoke <- as.factor(data2$Smoke)
data2$Alcohol <- as.factor(data2$Alcohol)
data2$Exercise <- as.factor(data2$Exercise)
```

- Discretization: necessary for Naive Bayes Classifier

```
data2$age_grp <- arules::discretize(data2$Age, method = "interval", breaks = 3, labels = c("low", "middle", "high
"))
data2 %>%
  group_by(age_grp) %>%
  summarise(avg = mean(Age), count = n(), min = min(Age), max = max(Age))
```

```
# A tibble: 3 x 5
  age_grp   avg count   min   max
  <fct>   <dbl> <int> <dbl> <dbl>
1 low      43.1  4914    39    47
2 middle   51.8  7896    48    55
3 high     59.6  8190    56    64
```

```
data2$height_grp <- arules::discretize(data2$Height, method = "interval", breaks = 3, labels = c("low", "middle",
"high"))
data2 %>%
  group_by(height_grp) %>%
  summarise(avg = mean(Height), count = n(), min = min(Height), max = max(Height))
```

```
# A tibble: 3 x 5
  height_grp   avg count   min   max
  <fct>      <dbl> <int> <dbl> <dbl>
1 low         154.  3893   143   157
2 middle      164. 13575   158   171
3 high        176.  3532   172   186
```

```
data2$weight_grp <- arules::discretize(data2$Weight, method = "interval", breaks = 3, labels = c("low", "middle",
"high"))
data2 %>%
  group_by(weight_grp) %>%
  summarise(avg = mean(Weight), count = n(), min = min(Weight), max = max(Weight))
```

```
# A tibble: 3 x 5
  weight_grp   avg count   min   max
  <fct>      <dbl> <int> <dbl> <dbl>
1 low         57.2  4123  43    62.3
2 middle      71.8 12678  62.7  82
3 high        90.1  4199  82.4 102
```

```
data2$h_blood_grp <- arules::discretize(data2$High.Blood.Pressure, method = "interval", breaks = 3, labels = c("l
ow", "middle", "high"))
data2 %>%
  group_by(h_blood_grp) %>%
  summarise(avg = mean(High.Blood.Pressure), count = n(), min = min(High.Blood.Pressure), max = max(High.Blood.Pr
essure))
```

```
# A tibble: 3 x 5
  h_blood_grp   avg count   min   max
  <fct>       <dbl> <int> <dbl> <dbl>
1 low          106.  3817    90   116
2 middle       126. 14579   117   143
3 high         155.  2604   144   170
```

```
data2$l_blood_grp <- arules::discretize(data2$Low.Blood.Pressure, method = "interval", breaks = 3, labels = c("lo
w", "middle", "high"))
data2 %>%
  group_by(l_blood_grp) %>%
  summarise(avg = mean(Low.Blood.Pressure), count = n(), min = min(Low.Blood.Pressure), max = max(Low.Blood.Press
ure))
```

```
# A tibble: 3 x 5
  l_blood_grp   avg count   min   max
  <fct>       <dbl> <int> <dbl> <dbl>
1 low          70.1  3223    65    78
2 middle       82.7 16512    79    91
3 high         99.8  1265    92   105
```

```
testing_cat <- data2[sapply(data2, is.factor)]
```

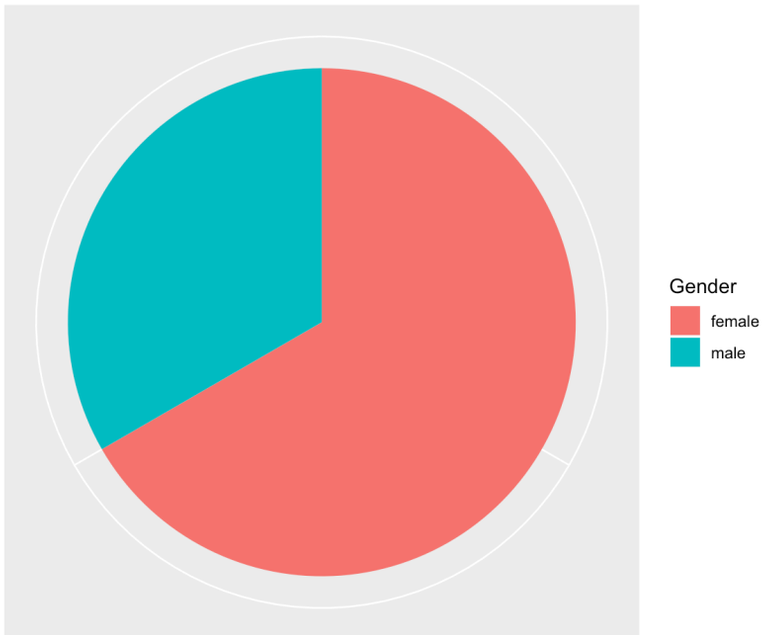
- Numerization: necessary for KNN, RF(Random Forest), GBM(Gradient Boosting Machine), SVM(linear & non-linear)
- Standardization & Normalization: necessary for numerical values which will be used in KNN, RF(Random Forest), GBM(Gradient Boosting Machine), SVM(linear & non-linear)

```
dmy2 <- caret::dummyVars(~Gender+Cholesterol+Glucose+Smoke+Alcohol+Exercise,data = data2, fullRank = T)
dummy2 <- data.frame(predict(dmy2, newdata = data2))
subset2 <- subset(data2,select= c(ID, Age, Height, Weight, High.Blood.Pressure, Low.Blood.Pressure))
testing_num <- cbind(subset2, dummy2)
pre_process2 <- preProcess(testing_num, method = c("scale", "center"))
testing_num_s <- predict(pre_process2, newdata = testing_num)
```

1-3. Exploratory Data Analysis (EDA)-traning data

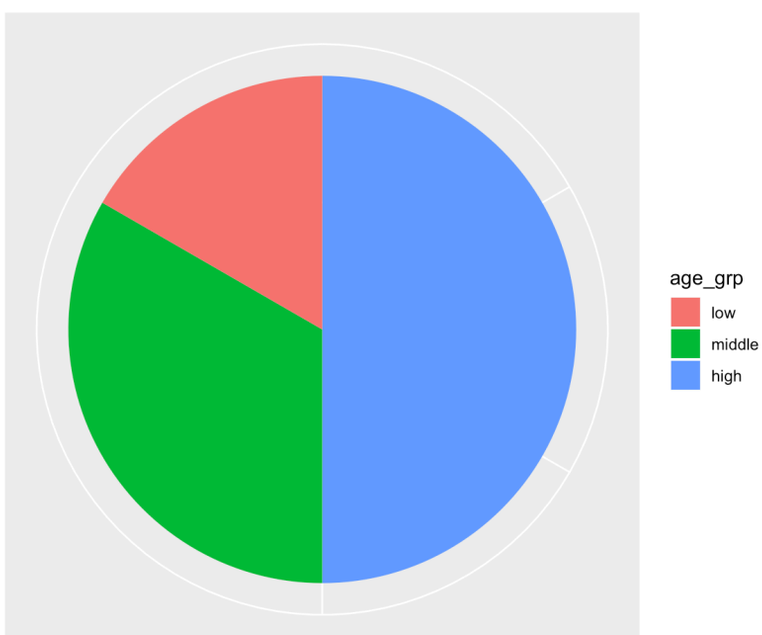
- Piechart: Gender distribution (Female>Male)

```
data %>%  group_by(Gender) %>%
  summarise(pct = percent(n()/nrow(data))) %>%
  ggplot(aes(x = factor(1), y = pct, fill = Gender)) +
  geom_bar(stat = "identity") +
  coord_polar(theta = "y") +
  theme(axis.ticks = element_blank(),
        axis.text = element_blank(),
        axis.title = element_blank())
```



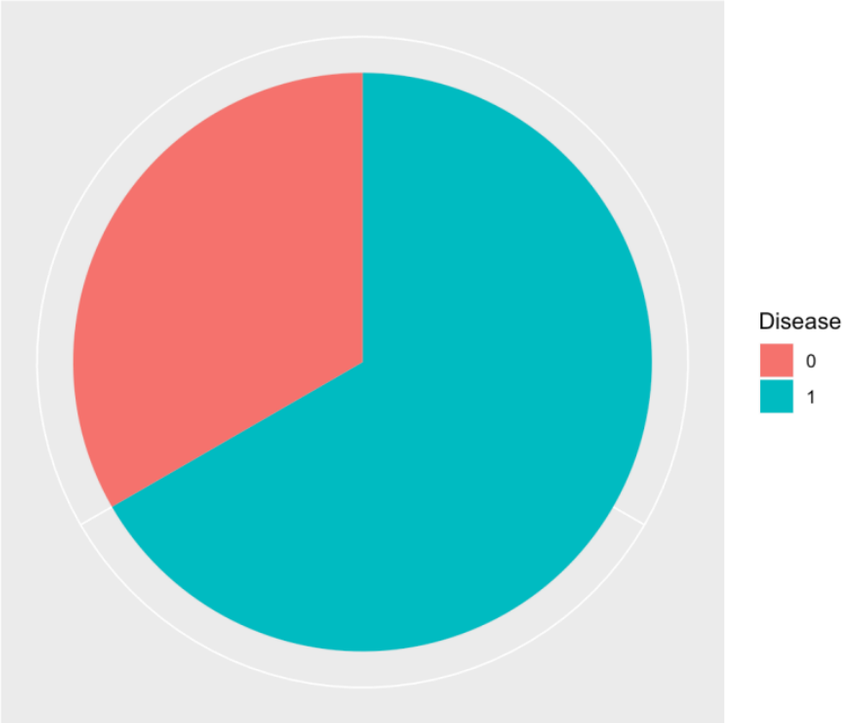
- Piechart: Age_group distribution (Hight>Middle>Low)

```
data %>%  group_by(age_grp) %>%
  summarise(pct = percent(n()/nrow(data))) %>%
  ggplot(aes(x = factor(1), y = pct, fill = age_grp)) +
  geom_bar(stat = "identity") +
  coord_polar(theta = "y") +
  theme(axis.ticks = element_blank(),
        axis.text = element_blank(),
        axis.title = element_blank())
```



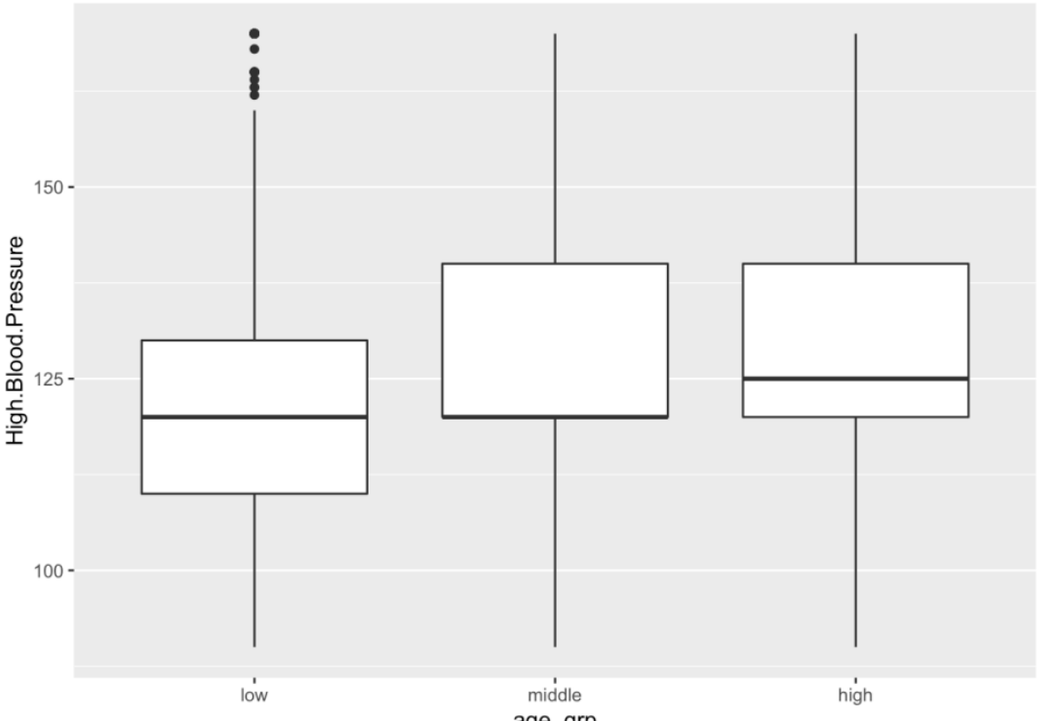
- Piechart: Disease distribution (Yes>No)

```
data %>%  group_by(Disease) %>%
  summarise(pct = percent(n()/nrow(data))) %>%
  ggplot(aes(x = factor(1), y = pct, fill = Disease)) +
  geom_bar(stat = "identity") +
  coord_polar(theta = "y") +
  theme(axis.ticks = element_blank(),
        axis.text = element_blank(),
        axis.title = element_blank())
```



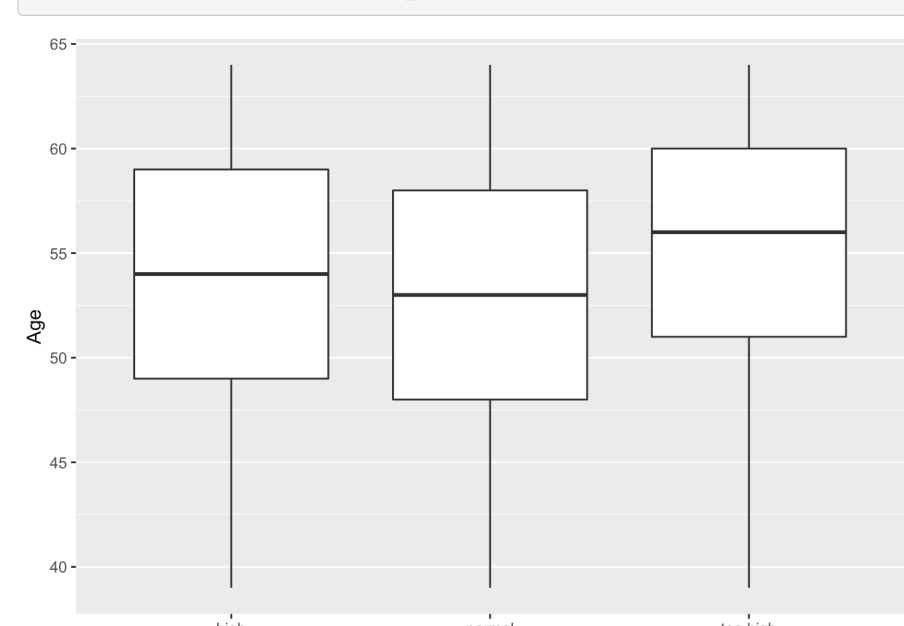
- Boxplot: age group-High.Blood.Pressure / older people tend to have higher blood pressure

```
ggplot(data, aes(x = age_grp, y = High.Blood.Pressure)) +
  geom_boxplot() +
  theme(panel.grid.major.x = element_blank())
```



- Boxplot: Glucose-Age / older people tend to have higher glucose

```
ggplot(data, aes(x = Glucose, y = Age)) +
  geom_boxplot() +
  theme(panel.grid.major.x = element_blank())
```



- Density plot: Weight-Cholesterol / High Cholesterol level is concentrated on 70kg. Heavier people are more likely to have higher Cholesterol level

```
ggplot(data, aes(Weight)) +
  geom_density(aes(fill=factor(Cholesterol))) +
  labs(title="Density plot",
       subtitle="Weight Grouped by Cholesterol level",
       caption="Source: data",
       x="Weight",
       fill="Cholesterol level")
```

- Heatmap: Alcohol-Smoke / People who drink and smoke show higher blood pressure, but smoke does not affect blood pressure by itself.

```
ggplot(data, aes(x = Alcohol, y = Smoke)) +
  geom_tile(aes(fill = High.Blood.Pressure), color = "white") +
  scale_fill_gradient(low = "green", high = "red") +
  theme(axis.ticks = element_blank()) +
  labs(x = "Alcohol", y = "Smoke")
```

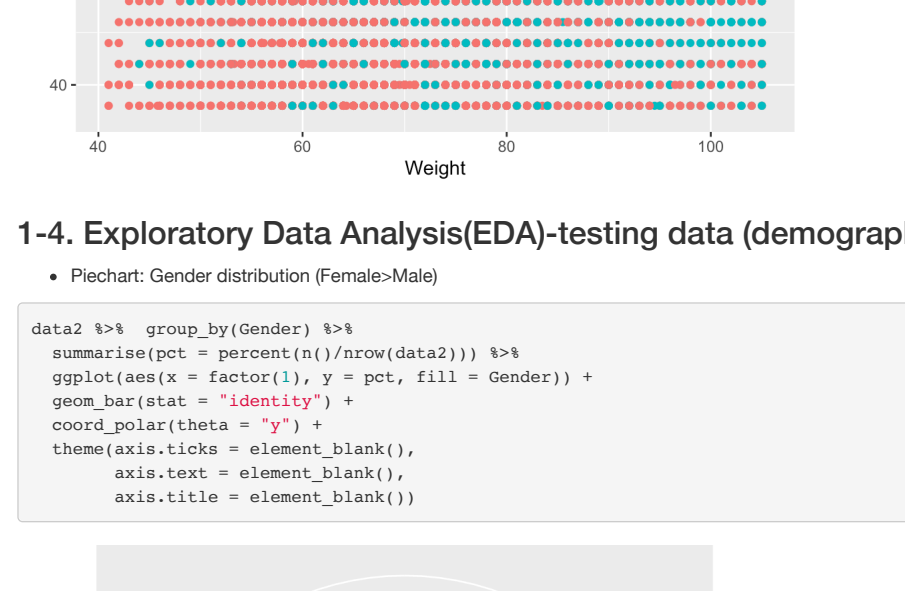
- Scatter plot: Weight-Age / Weight and Age are in positive relationship, but not very strong. The heavier, the older, people are more likely to get a disease.

```
ggplot(data, aes(x = Weight, y = Age)) +
  geom_point(aes(color = Disease)) +
  geom_smooth(method = "lm")
```

1-4. Exploratory Data Analysis(EDA)-testing data (demographic analysis)

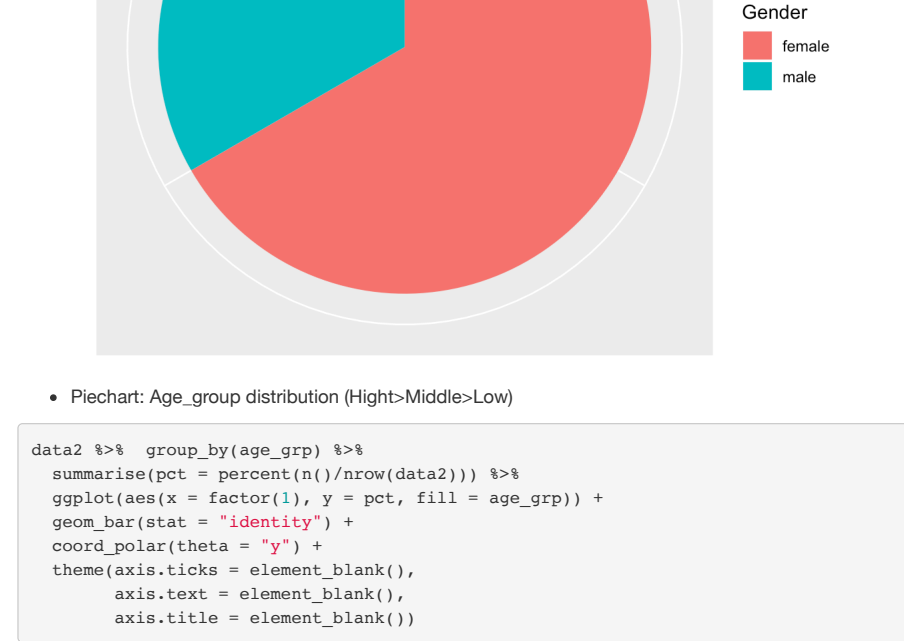
- Piechart: Gender distribution (Female>Male)

```
data2 %>%  group_by(Gender) %>%
  summarise(pct = percent(n()/nrow(data2))) %>%
  ggplot(aes(x = factor(1), y = pct, fill = Gender)) +
  geom_bar(stat = "identity") +
  coord_polar(theta = "y") +
  theme(axis.ticks = element_blank(),
        axis.text = element_blank(),
        axis.title = element_blank())
```



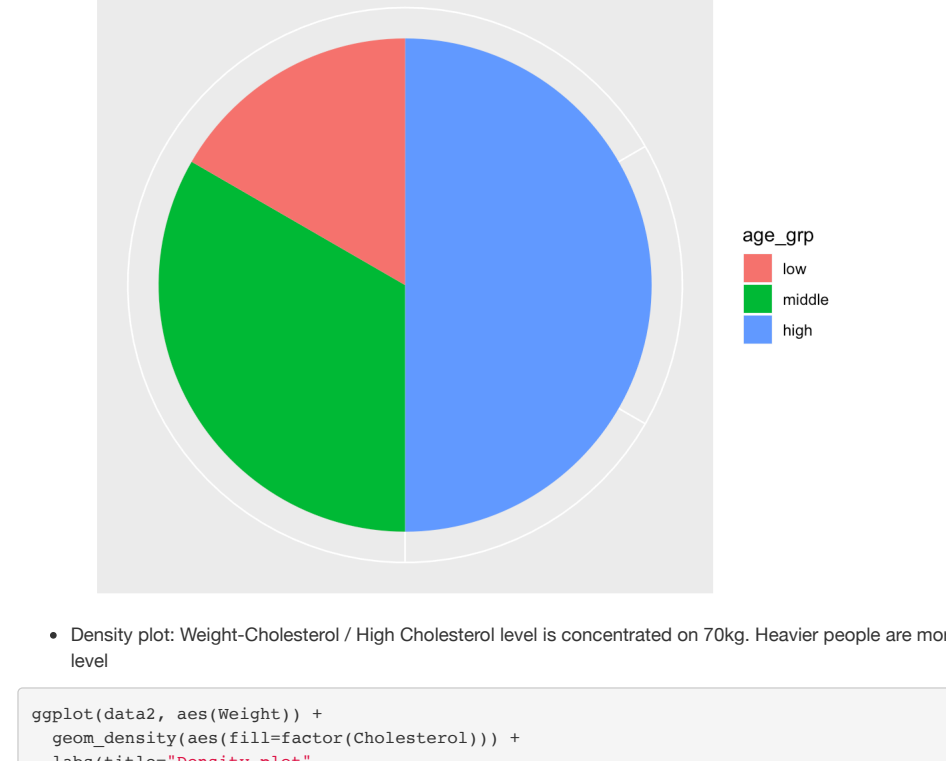
- Piechart: Age_group distribution (High>Middle>Low)

```
data2 %>%  group_by(age_grp) %>%
  summarise(pct = percent(n()/nrow(data2))) %>%
  ggplot(aes(x = factor(1), y = pct, fill = age_grp)) +
  geom_bar(stat = "identity") +
  coord_polar(theta = "y") +
  theme(axis.text = element_blank(),
        axis.title = element_blank())
```



- Density plot: Weight-Cholesterol / High Cholesterol level is concentrated on 70kg. Heavier people are more likely to have higher Cholesterol level

```
ggplot(data2, aes(Weight)) +
  geom_density(aes(fill=factor(Cholesterol))) +
  labs(title="Density plot",
       subtitle="Weight Grouped by Cholesterol level",
       caption="Source: data2",
       x="Weight",
       fill="Cholesterol level")
```



Test data shows similar demographic distribution(age, gender) and weight-Cholesterol density to those of training data

Section2. Build, tune and evaluate various machine learning algorithms

2-1. Building machine learning algorithms

2-1-1. Naive Bayes Classifier(NBC)

- Building a model with default value
- Hyper-parameters: fL=0, usekernel=False

```
nbc1 <- train(training_cat[,!names(training_cat)%in%"Disease"], training_cat$Disease, method = "nb")
nbc1
```

```
Naive Bayes

47248 samples
   11 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 47248, 47248, 47248, 47248, 47248, 47248, ...
Resampling results across tuning parameters:

  usekernel  Accuracy   Kappa
  FALSE      0.6799417  0.3600425
   TRUE      0.6799417  0.3600425

Tuning parameter 'fL' was held constant at a value of 0
Tuning
 parameter 'adjust' was held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were fL = 0, usekernel = FALSE and adjust
 = 1.
```

- Model tuning
  - fl: adding smoothing to fix zero probility issue (1~4)
  - kernel: probability density function for continuous attributes (True, False)

```
nbc2 <- train(training_cat[,!names(training_cat)%in%"Disease"], training_cat$Disease, method = "nb",
              trControl = trainControl(method = "cv", number = 3),
              tuneGrid = expand.grid(fL = 1:4, usekernel = c(TRUE, FALSE), adjust = 1:3))
nbc2
```

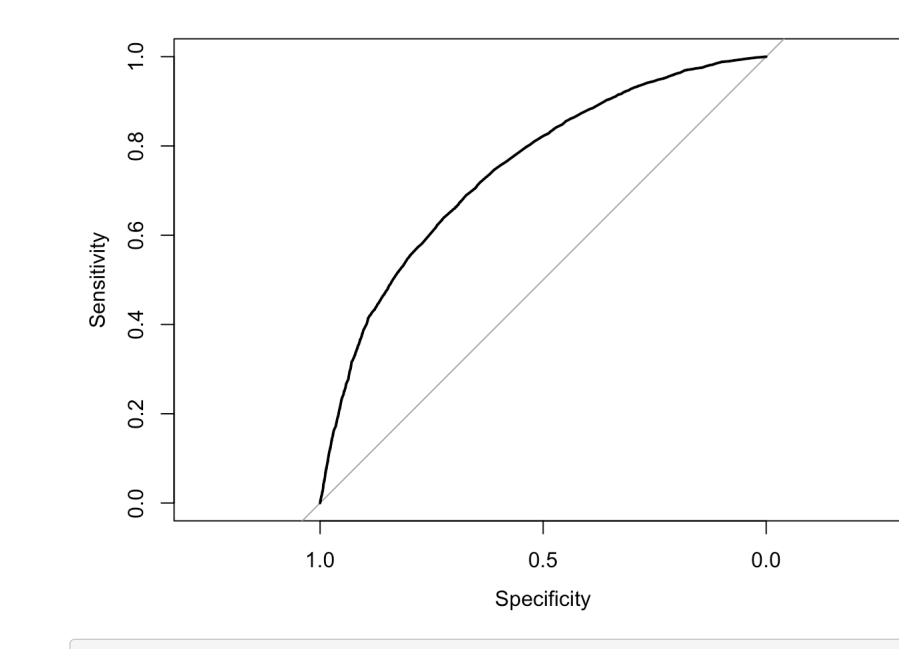
```
Naive Bayes

47248 samples
   11 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (3 fold)
Summary of sample sizes: 31499, 31499, 31498
Resampling results across tuning parameters:

  fL  usekernel  adjust  Accuracy   Kappa
  1   FALSE      1       0.6794361  0.3589925
  1   FALSE      2       0.6794361  0.3589925
  1   FALSE      3       0.6794361  0.3589925
  1    TRUE      1       0.6794361  0.3589925
  1    TRUE      2       0.6794361  0.3589925
  1    TRUE      3       0.6794361  0.3589925
  2   FALSE      1       0.6794361  0.3589925
  2   FALSE      2       0.6794361  0.3589925
  2   FALSE      3       0.6794361  0.3589925
  2    TRUE      1       0.6794361  0.3589925
  2    TRUE      2       0.6794361  0.3589925
  2    TRUE      3       0.6794361  0.3589925
  3   FALSE      1       0.6794573  0.3590364
  3   FALSE      2       0.6794573  0.3590364
  3   FALSE      3       0.6794573  0.3590364
  3    TRUE      1       0.6794573  0.3590364
  3    TRUE      2       0.6794573  0.3590364
  3    TRUE      3       0.6794573  0.3590364
  4   FALSE      1       0.6794361  0.3589946
  4   FALSE      2       0.6794361  0.3589946
  4   FALSE      3       0.6794361  0.3589946
  4    TRUE      1       0.6794361  0.3589946
  4    TRUE      2       0.6794361  0.3589946
  4    TRUE      3       0.6794361  0.3589946

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were fL = 3, usekernel = FALSE and adjust
 = 1.
```

```
nbc2$bestTune
```

```
   fL usekernel adjust
13  3     FALSE      1
```

- ROC & AUC
  - nbc1: 0.7478

```
roc_nbc1 <- predict(nbc1, newdata = training_cat, type = "prob")
roc_curve1 <- roc(training_cat$Disease,roc_nbc1$"1")
plot(roc_curve1)
```

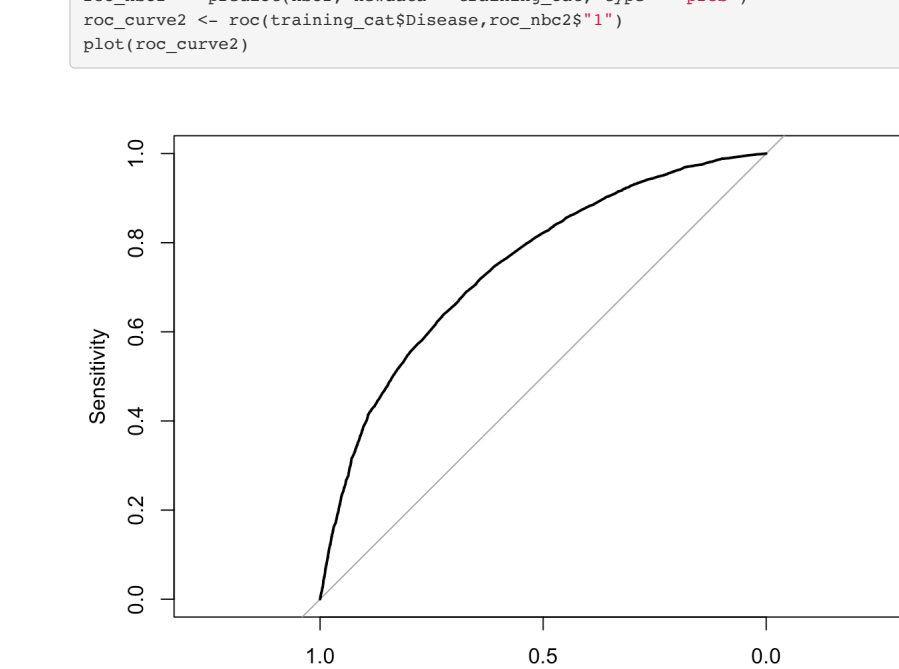


```
auc(roc_curve1)
```

```
Area under the curve: 0.7478
```

- nbc2: 0.7478

```
roc_nbc2 <- predict(nbc2, newdata = training_cat, type = "prob")
roc_curve2 <- roc(training_cat$Disease,roc_nbc2$"1")
plot(roc_curve2)
```



```
auc(roc_curve2)
```

```
Area under the curve: 0.7478
```

- Performance evalution method & metrics
  - Method: Cross validation, ROC
  - Metrics: accuracy, kappa (the higher, the better), AUC (the closer to 1, the more accurate/ no difference in nbc1 & nbc2)
- Best performing model & hyper-parameters
  - Model: nbc2
  - Hyper-parameters: fL=1, kernel=False
- Prediction

```
nbc2_p <- predict(nbc2, newdata = testing_cat, type = "raw")
list_p <- subset(data2,select=10)
list_p$NBC <- nbc2_p
```

2-1-2. KNN

- Building a model with default value
- Hyper-parameter: k=5,7,9, distance function='Euclidean' (default for numerical values)
- I used random sampling due to long execution time

```
set.seed(1)
sample_knn <- training_num_s %>%
  sample_frac(size = 0.2, replace = FALSE)
knn1 <- train(Disease ~ ., data = sample_knn, method = "knn")
print(knn1)
```

```
k-Nearest Neighbors

9450 samples
  13 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 9450, 9450, 9450, 9450, 9450, 9450, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.6527715  0.3052906
  7  0.6628355  0.3254808
  9  0.6706309  0.3411305

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 9.
```

- Model tuning
  - k: number of neighbors (10~15)

```
set.seed(1)
knn2 <- train(Disease ~ ., data = sample_knn, method = "knn",
              tuneGrid = data.frame(k = seq(10, 15)),
              trControl = trainControl(method = "repeatedcv",
                                       number = 5, repeats = 3))
print(knn2)
```

```
k-Nearest Neighbors

9450 samples
  13 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 7560, 7560, 7560, 7560, 7561, 7561, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
  10  0.6948501  0.3897670
  11  0.6986949  0.3975541
  12  0.6955201  0.3911340
  13  0.6997178  0.3996521
  14  0.6999291  0.4001070
  15  0.7007757  0.4018869

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 15.
```

```
knn2$bestTune
```

```
   k
6 15
```

- Performance evalution method & metrics
  - Method: Cross validation
  - Metrics: accuracy, kappa (the higher, the better)

*Best performing model & hyper-parameters + Model: knn2 + Hyper-parameters: k=15

- Prediction

```
knn2_p <- predict(knn2, newdata = testing_num_s)
list_p$KNN <- knn2_p
```

2-1-3. SVM (linear)

- Buidling & tuning a model (linear)
- I used random sampling due to long execution time
- Hyper-parameter
  - C: Cost of constraints (0~2)
  - Lower: soft margin, high bias, low variance -> underfitting
  - Higher: hard margin, low bias, high variance -> overfitting

```
set.seed(123)
sample <- training_num_s %>%
  sample_frac(size = 0.05, replace = FALSE)
svm_linear <- train(
  Disease ~., data = sample, method = "svmLinear",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(C = seq(0, 2, length = 20)),
)
svm_linear
```

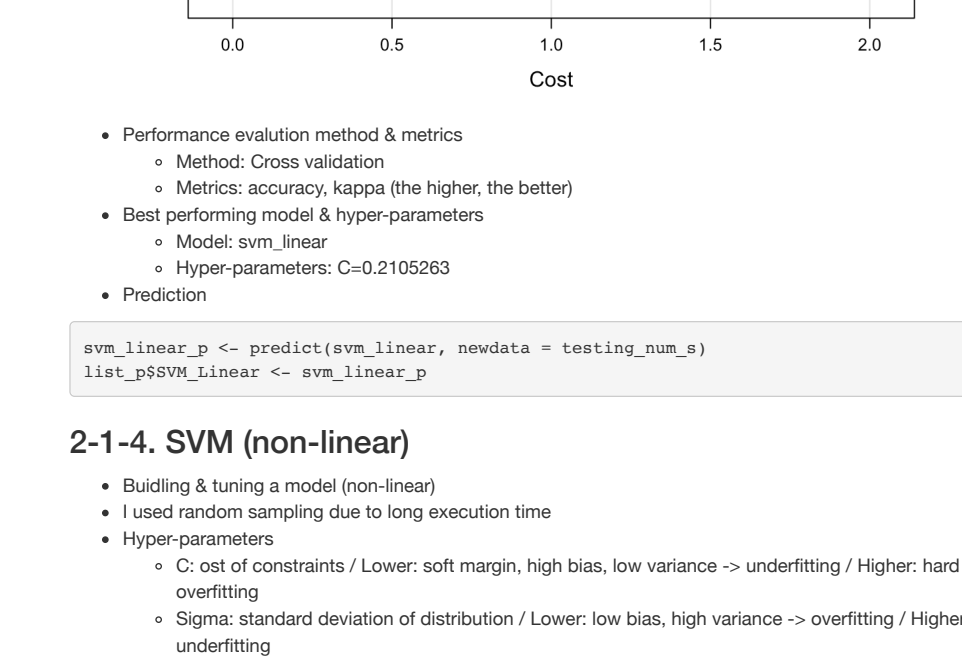
```
Support Vector Machines with Linear Kernel

2362 samples
  13 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 2127, 2127, 2125, 2125, 2126, 2126, ...
Resampling results across tuning parameters:

  C          Accuracy   Kappa
  0.0000000        NaN        NaN
  0.1052632  0.7277693  0.4559752
  0.2105263  0.7286168  0.4576637
  0.3157895  0.7286168  0.4576637
  0.4210526  0.7281913  0.4568149
  0.5263158  0.7277693  0.4559729
  0.6315789  0.7281949  0.4568217
  0.7368421  0.7281949  0.4568217
  0.8421053  0.7281949  0.4568217
  0.9473684  0.7281949  0.4568217
  1.0526316  0.7281949  0.4568217
  1.1578947  0.7281949  0.4568217
  1.2631579  0.7281949  0.4568217
  1.3684211  0.7286168  0.4576638
  1.4736842  0.7286168  0.4576638
  1.5789474  0.7281949  0.4568186
  1.6842105  0.7286168  0.4576638
  1.7894737  0.7281949  0.4568186
  1.8947368  0.7286168  0.4576638
  2.0000000  0.7281949  0.4568186

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 0.2105263.
```

```
svm_linear$bestTune
```

```
          C
3 0.2105263
```

- Accuracy-Cost plot: 6 C values which show the highest accuracy (selected the smallest one among them)

```
plot(svm_linear)
```



- Performance evalution method & metrics
  - Method: Cross validation
  - Metrics: accuracy, kappa (the higher, the better)
- Best performing model & hyper-parameters
  - Model: svm_linear
  - Hyper-parameters: C=0.2105263
- Prediction

```
svm_linear_p <- predict(svm_linear, newdata = testing_num_s)
list_p$SVM_Linear <- svm_linear_p
```

2-1-4. SVM (non-linear)

- Buidling & tuning a model (non-linear)
- I used random sampling due to long execution time
- Hyper-parameters
  - C: cost of constraints / Lower: soft margin, high bias, low variance -> underfitting / Higher: hard margin, low bias, high variance -> overfitting
  - Sigma: standard deviation of distribution / Lower: low bias, high variance -> overfitting / Higher: high bias, low variance -> underfitting

```
svm_non_linear <- train(
  Disease ~., data = sample, method = "svmRadial",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
)
svm_non_linear
```

```
Support Vector Machines with Radial Basis Function Kernel

2362 samples
  13 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 2127, 2125, 2126, 2126, 2125, 2125, 2126, ...
Resampling results across tuning parameters:

  C       Accuracy   Kappa
    0.25  0.7176294  0.4354677
    0.50  0.7193189  0.4388120
    1.00  0.7163546  0.4329268
    2.00  0.7142431  0.4286659
    4.00  0.7049210  0.4100691
    8.00  0.7002545  0.4007718
   16.00  0.6850164  0.3702938
   32.00  0.6769709  0.3543271
   64.00  0.6689307  0.3382476
  128.00  0.6634294  0.3272810

Tuning parameter 'sigma' was held constant at a value of 0.07043372
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.07043372 and C = 0.5.
```

```
svm_non_linear$bestTune
```

```
       sigma   C
2 0.07043372 0.5
```

- Performance evalution method & metrics
  - Method: Cross validation
  - Metrics: accuracy, kappa (the higher, the better)
- Best performing model & hyper-parameters
  - Model: svm_non_linear
  - Hyper-parameters: C=0.5, Sigma=0.07043372
- Prediction

```
svm_non_linear_p <- predict(svm_non_linear, newdata = testing_num_s)
list_p$SVM_RBF <- svm_non_linear_p
```

2-1-5. Random Forest

- Building a model
- I used random sampling due to long execution time
- Hyper-parameters: mtry (1~5)-the number of features to choose
  - Higher: trees look similar & higher probability of overfitting
  - Lower: trees look different & higher probability of underfitting

```
fitControl <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
customGrid <- expand.grid(mtry = 1:5)
set.seed(1)
rf <- train(Disease ~ ., data = sample, method = "rf", trControl = fitControl, tuneGrid = customGrid)
rf
```

```
Random Forest

2362 samples
  13 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 1890, 1890, 1889, 1890, 1889, 1890, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
  1     0.7068824  0.4144571
  2     0.7207105  0.4417445
  3     0.7177461  0.4357183
  4     0.7082936  0.4166895
  5     0.7097069  0.4194596

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.
```

```
rf$bestTune
```

```
  mtry
2    2
```

- Performance evalution method & metrics
  - Method: Cross validation
  - Metrics: accuracy, kappa (the higher, the better)
- Best performing model & hyper-parameters
  - Model: rf
  - Hyper-parameters: mtry=2
- Prediction

```
rf_p <- predict(rf, newdata = testing_num_s)
list_p$RF <- rf_p
```

2-1-6. Gradient Boosting Machine (GBM)

- Building a model
- I used random sampling due to long execution time
- Hyper-parameters:
  - nrounds: the number of boosting iteratoin
  - max-depth: maximum depth of a tree. the higher, the more likely to overfit
  - eta: step size shrinkage used in update to prevents overfitting.
  - gamma: minimum loss reduction required to make a further partition on a leaf node of the tree. the larger gamma is, the more conservative the algorithm will be.
  - colsample_bytree: subsample ratio of columns when constructing each tree. can be used to prevent overfitting.
  - min_child_weight: minimum sum of instance weight needed in a child. The larger, the more conservative the algorithm will be.
  - subsample: subsample ratio of the training instances. can be used to prevent overfitting

```
set.seed(1)
gbm <- train(Disease ~., data = sample,
             method = "xgbTree")
gbm
gbm$bestTune
```

- Performance evalution method & metrics
  - Method: Cross validation
  - Metrics: accuracy, kappa (the higher, the better)
- Best performing model & hyper-parameters
  - Model: gbm
  - Hyper-parameters: nrounds=50, max_depth=1, eta=0.3, gamma=0, colsample_bytree=0.6, min_child_weight=1, subsample=1
- Prediction

```
gbm_p <- predict(gbm, newdata = testing_num_s)
list_p$GBM <- gbm_p
```

2-2. Best performing models & model specification summary

Algorithm	Model ID	Specification
NBC	nbc2	fL=0, usekernel=False
KNN	knn2	k=15
SVM_Linear	svm_linear	C=0.2105263
SVM_non-linear	svm_non_linear	Sigma=0.07043372, C=0.5
RF	rf	mtry=2
GBM	gbm	nrounds=50, max_depth=1, eta=0.3, gamma=0, colsample_bytree=0.6, min_child_weight=1, subsample=1

Section 3. Prediction and interpretation

3-1. Exporting prediction in csv file

```
write.csv(list_p,"C:\\Users\\jemmint\\Desktop\\SU IM DOC\\prediction.csv")
```

3-2. Conclusion

Overall, the performance of 6 algorithms (NBC, KNN, SVM_Linear, SVM_non_linear, RF, GBM) seems to be similar in terms of accuracy and kappa. All the best performing models show around 70% accuracy and 40& kappa. However, each algorithm has different features. NBC is good for handling categorical values which are independent each other, so it is robust to outliers. Also, it's helpful to get the probability of individual classes. However, additional technique(laplace smoothing) is required to fix zero probabilities which can appear when new data is input. Except for NBC, 5 other algorithms can be used with numerical values. Among them, a distance-based algorithm, KNN, is sensitive to noises and outliers. Therefore, data cleaning and standardization should be done carefully at the data preprocessing stage. SVM is a discriminative black box model used for determining decision boundary. As it's a black-box model, interpretation is not easy. However, it can be utilized for various tasks since it can handle non-linear decision bounday. The ensemble algorithms RF and GBM are also useful for dealing with various tasks. RF is intuitive and robust to overfitting. GBM produces comparatively accurate prediction since it uses gradient descent to construct new base learners that minimize loss function. However, both RF and GBM require considerabe time for modeling and numerous hyper-parameters for tuning.