Heart Disease Prediction

Rita Han

2022-11-23

Contents

1.0 Abstract

Introduction

Modern days, people are busy with lives and satisfaction on materialistic needs, but neglect of body and mental health. This cause more and more heart suffer with various causes. Heart is the most crucial organ inside human body. Cardiovascular disease(heart disease) are #1 cause of death globally, which can take 17.9 million lives per year in estimation. This occupies 31% of all deaths worldwide.
People with heart disease need early detection due to its high risk of death. In this case, machine learning model can be a great helper. Thus, I choose the heart failure prediction dataset to predict if someone is at high risk of being diagonised with heart disease. This dataset contains 11 features that can be used to predict possible heart disease. I first got to know the dataset and visualized it and then use cross-validation for the machine to better learn the data and to tune machine learning models: Random Forest, XGBoost, KNN, Decision Tree to compare which model perform the best on the train set in order prediction on the test set.

Description of the Dataset

The Heart Failure Prediction Dataset was created by combining five different datasets with observations from different places among the world, with total 918 observations and 11 variables that can be possible predictors and our target variable 'heart_disease'.
-Age: age of the patient [years]
-Sex: sex of the patient [M: Male, F: Female]
-ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
-RestingBP: resting blood pressure [mm Hg]
-Cholesterol: serum cholesterol [mm/dl]
-FastingBS: fasting blood sugar [1: if FastingBS $>$ 120 mg/dl, 0: otherwise]
-RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of $>$ 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
-MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
-ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
-Oldpeak: oldpeak = ST [Numeric value measured in depression]
-ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
-HeartDisease: output class [1: heart disease, 0: Normal]

```
#input file and use clean_names() function to put variables into a clear format
heart = read_csv("/Users/ritahan/Desktop/pstat131/heart.csv") %>%
  clean_names()
```

```
## Rows: 918 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (5): Sex, ChestPainType, RestingECG, ExerciseAngina, ST_Slope
## dbl (7): Age, RestingBP, Cholesterol, FastingBS, MaxHR, Oldpeak, HeartDisease
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
#display the first couple lines of the dataset
heart$heart_disease=as.character(heart$heart_disease)
heart$fasting_bs=as.character(heart$fasting_bs)
heart= heart %>%
  filter(heart$cholesterol != 0)
```

```r
heart=heart %>%
  mutate(fasting_bs=factor(fasting_bs),
         heart_disease=factor(heart_disease),
         chest_pain_type=factor(chest_pain_type),
         resting_ecg=factor(resting_ecg),
         exercise_angina=factor(exercise_angina))
heart %>%
  head()
```

```
## # A tibble: 6 x 12
##     age sex   chest_pai~1 resti~2 chole~3 fasti~4 resti~5 max_hr exerc~6 oldpeak
##   <dbl> <chr> <fct>         <dbl>   <dbl> <fct>   <fct>    <dbl> <fct>     <dbl>
## 1    40 M     ATA             140     289 0       Normal     172 N           0
## 2    49 F     NAP             160     180 0       Normal     156 N           1
## 3    37 M     ATA             130     283 0       ST          98 N           0
## 4    48 F     ASY             138     214 0       Normal     108 Y           1.5
## 5    54 M     NAP             150     195 0       Normal     122 N           0
## 6    39 M     NAP             120     339 0       Normal     170 N           0
## # ... with 2 more variables: st_slope <chr>, heart_disease <fct>, and
## #   abbreviated variable names 1: chest_pain_type, 2: resting_bp,
## #   3: cholesterol, 4: fasting_bs, 5: resting_ecg, 6: exercise_angina
```

```r
heart %>%
  summary()
```

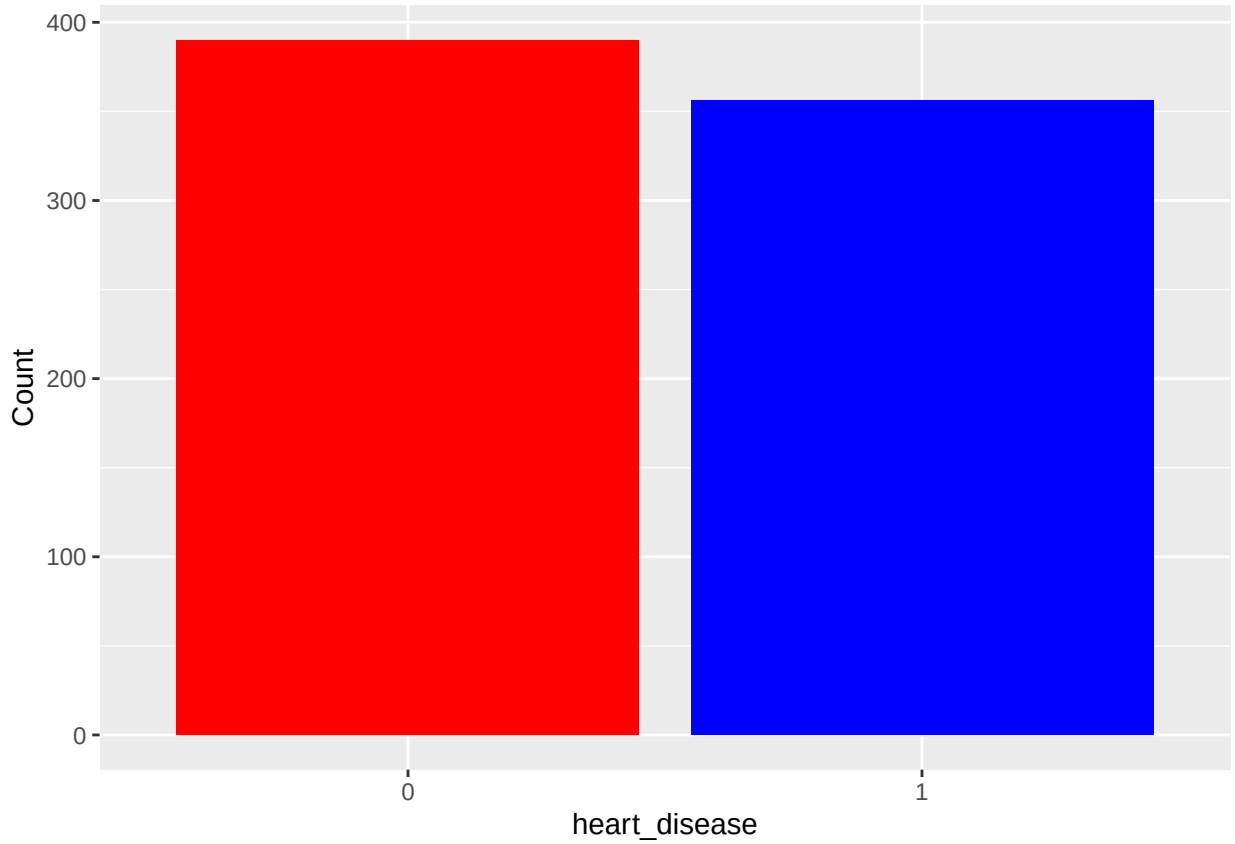
```
##       age            sex            chest_pain_type   resting_bp
##  Min.   :28.00   Length:746         ASY:370         Min.   : 92
##  1st Qu.:46.00   Class :character   ATA:166         1st Qu.:120
##  Median :54.00   Mode  :character   NAP:169         Median :130
##  Mean   :52.88                      TA : 41         Mean   :133
##  3rd Qu.:59.00                                      3rd Qu.:140
##  Max.   :77.00                                      Max.   :200
##   cholesterol    fasting_bs resting_ecg      max_hr      exercise_angina
##  Min.   : 85.0   0:621      LVH   :176   Min.   : 69.0   N:459
##  1st Qu.:207.2   1:125      Normal:445   1st Qu.:122.0   Y:287
##  Median :237.0              ST    :125   Median :140.0
##  Mean   :244.6                           Mean   :140.2
##  3rd Qu.:275.0                           3rd Qu.:160.0
##  Max.   :603.0                           Max.   :202.0
##     oldpeak          st_slope         heart_disease
##  Min.   :-0.1000   Length:746         0:390
##  1st Qu.: 0.0000   Class :character   1:356
##  Median : 0.5000   Mode  :character
##  Mean   : 0.9016
##  3rd Qu.: 1.5000
##  Max.   : 6.2000
```

I first make 'heart_disease and 'fasting_bs' to characters, then it's easy to mutate all characteristic variables to factors for further prediction. I filtered out the observations with cholesterol equal to 0. Obtain a cholesterol equal to 0 is very rare, so we can regard these as no observation. Thus, filter them out for further analysis might be a good idea.

2.0 EDA

The data is already a cleaned one! Let's get a deeper understanding of the dataset, I first visualized the distribution of the '1' and '0' from 'heart_disease'. Then, I made a correlation heatmap for all numeric variables to see if there is any interesting correlations. Next, I visualize the numeric variables with density plots fill by heart_disease, which can easily see the relationship between each numeric variable and heart disease. And, for categorical variables, I used barplot fill by 'heart_disease' and boxplot to explore some interesting relationships.
heart_disease count

```
heartdisease_counting <- heart %>%
    group_by(heart_disease) %>%
    summarise(Count = n())

ggplot(data = heartdisease_counting, aes(x = heart_disease,
                                         y = Count)) +
    geom_histogram(stat = "identity", fill = c("red", "blue"))
```



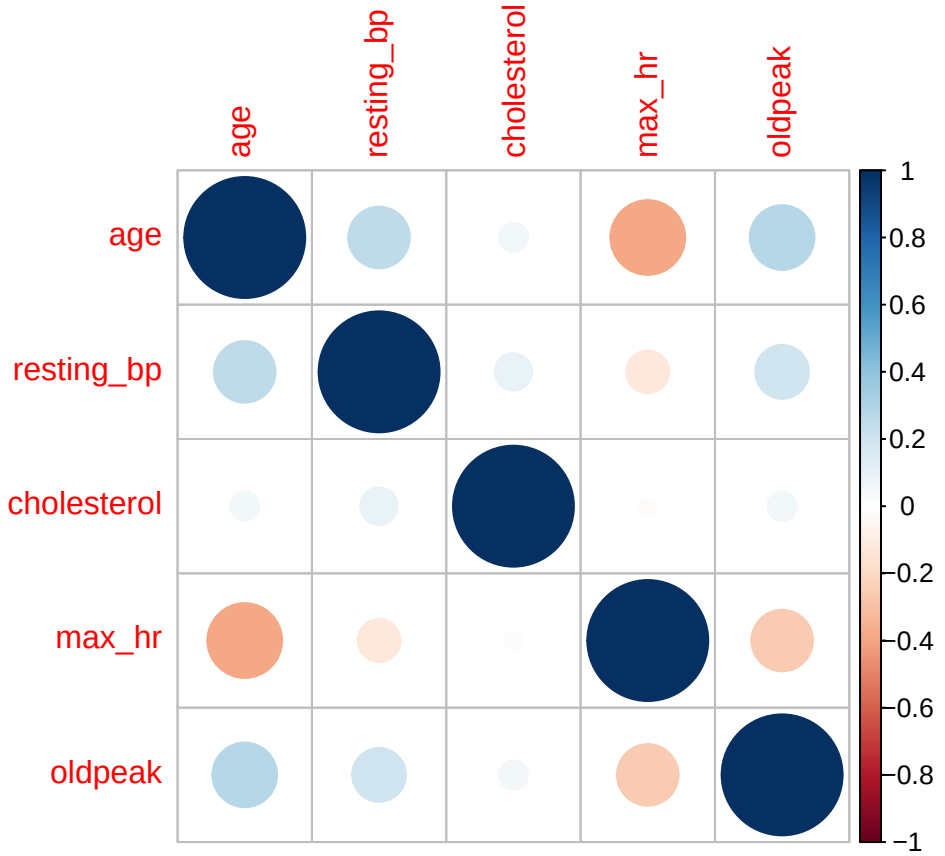
```
percent_heartdisease <- percent(heartdisease_counting$Count[1]/sum(heartdisease_counting$Count),
    accuracy = 0.01)
percent_heartdisease
```

```
## [1] "52.28%"
```

From this plot, we can see there are 52.28% of the dataset is observed with heart disease.

correlations

```
heart %>%
  select(is.numeric) %>%
  cor() %>%
  corrplot()
```

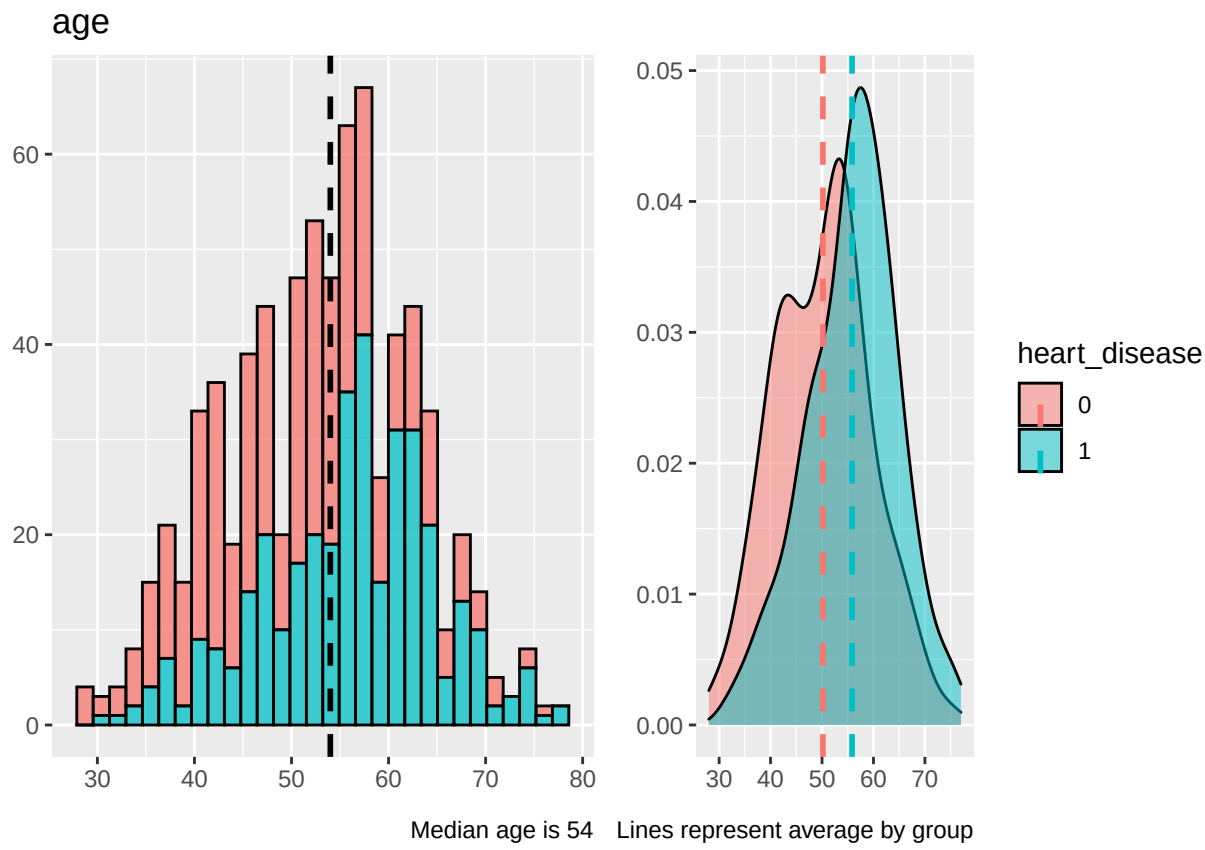


From this correlation heatmap, we can see max_hr is negatively correlated with age, this makes sense because maximum heart rate achieved often happens on younger people. Additionally, 'resting_bp','fasting_bs','oldpeak', are positively correlated with age, since older people usually have high blood pressure, sugar.

Numerical variables

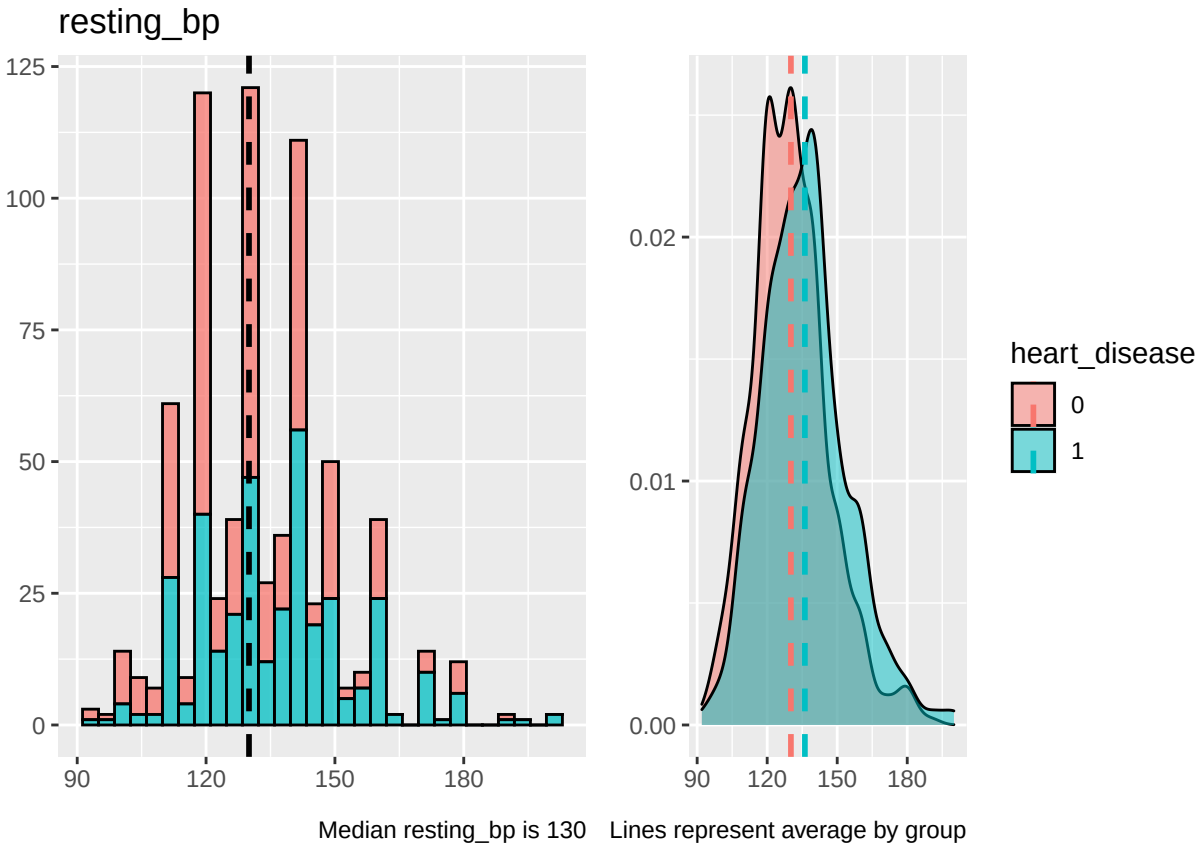
```
plot_histogram <- function(df, var1, var2) {
    # From object to string: deparse(substitute(varname))
    var1name <- as.name(var1)
    df %>%
        ggplot(aes(x = {
            {
                var1name
            }
        }, fill = {
            {
                var2
            }
        })) + geom_histogram(alpha = 0.75, position = "stack",
        color = "black", bins = 30) + geom_vline(aes(xintercept = median({
        {
            var1name
        }
    })), linetype = 2, size = 1) + labs(caption = paste0("Median ",
        {
            {
                var1
            }
        }, " is ", round(median({
            {
                df
            }
        }[[{
            {
                var1
            }
        }]]), 2)), y = element_blank(), x = element_blank(),
        title = paste0({
            {
                var1
            }
        })) + theme(legend.position = "none")
}

plot_density <- function(df, var1, var2) {
    var1name <- as.name(var1)
    df %>%
        ggplot(aes(x = {
            {
                var1name
            }
        }, fill = {
            {
                var2
            }
        })) + geom_density(alpha = 0.5, color = "black") + geom_vline(data = df %>%
        group_by({
            {
                var2
```

```
            }
        }) %>%
        summarize(mean.grp = mean({
            {
                var1name
            }
        })), aes(xintercept = mean.grp, color = heart_disease), linetype = "dashed",
        size = 1) + labs(caption = paste0("Lines represent average by group"),
        y = element_blank(), x = element_blank(), title = "")
}

plot_2plots <- function(df, var1, var2) {
    p1 <- plot_histogram({
        {
            df
        }
    }, {
        {
            var1
        }
    }, {
        {
            var2
        }
    })
    p2 <- plot_density({
        {
            df
        }
    }, {
        {
            var1
        }
    }, {
        {
            var2
        }
    })

    grid.arrange(p1, p2, ncol = 2)

}
plot_2plots(heart, "age", heart_disease)
```



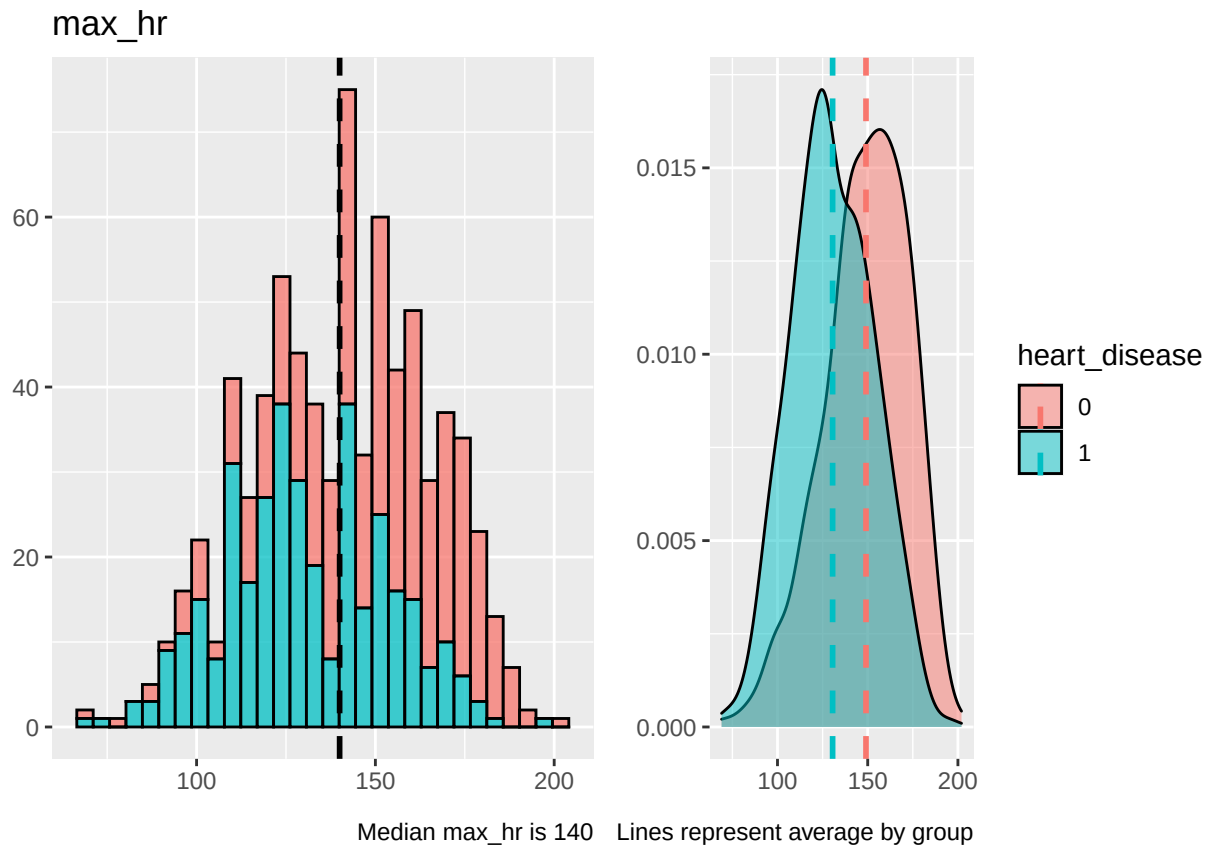
From the age plots, we can easily see that heart disease occur more on old people.

```
plot_2plots(heart, "resting_bp", heart_disease)
```



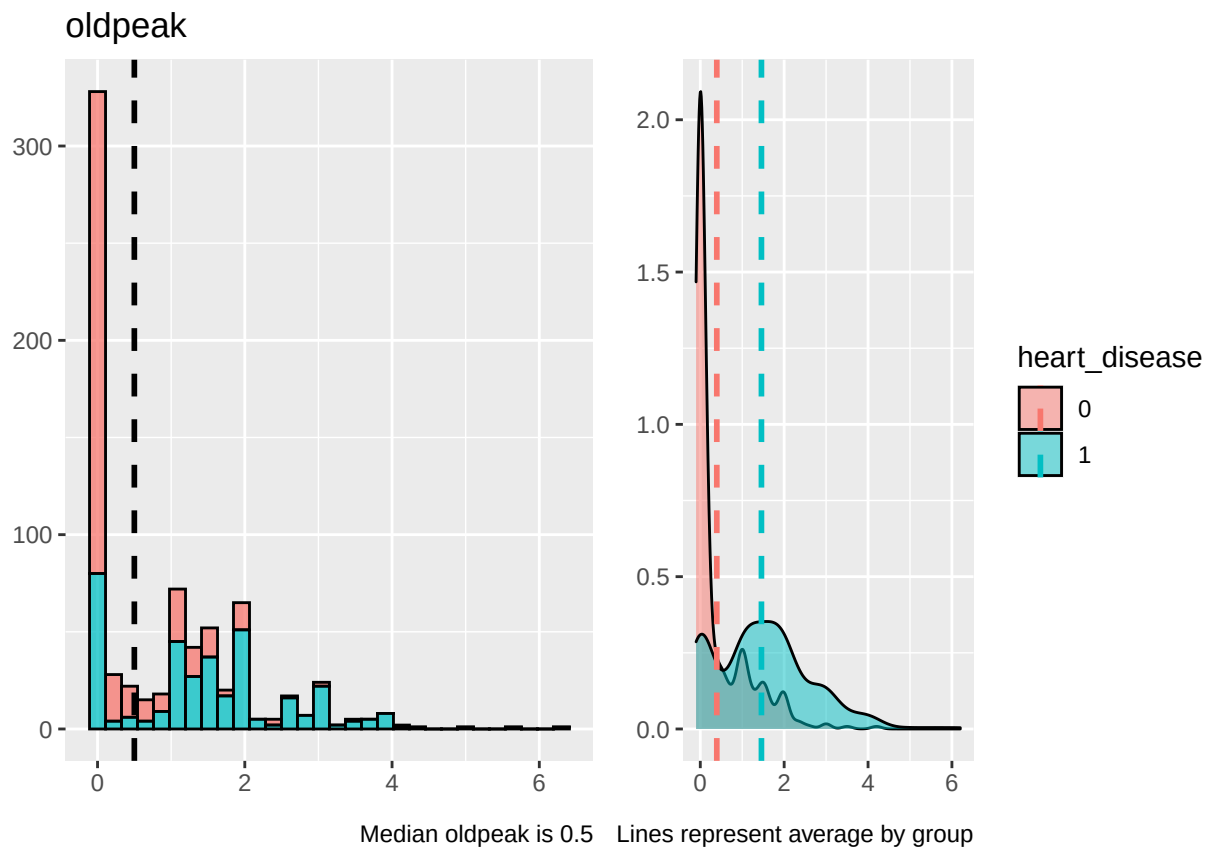
From the resting_bp plots, we can see that when resting blood pressure is between 100-150 mmhg, the occurance of heart-disdease is basically half-half; but when exceeding 150mmhg, the occurance of heart-disease increases a little.

```
plot_2plots(heart, "max_hr", heart_disease)
```



From the max_hr graphs, there is a negative correlation between max_hr and heart_disease. People who acheive higher heart rate have less probibility to get heart disease.

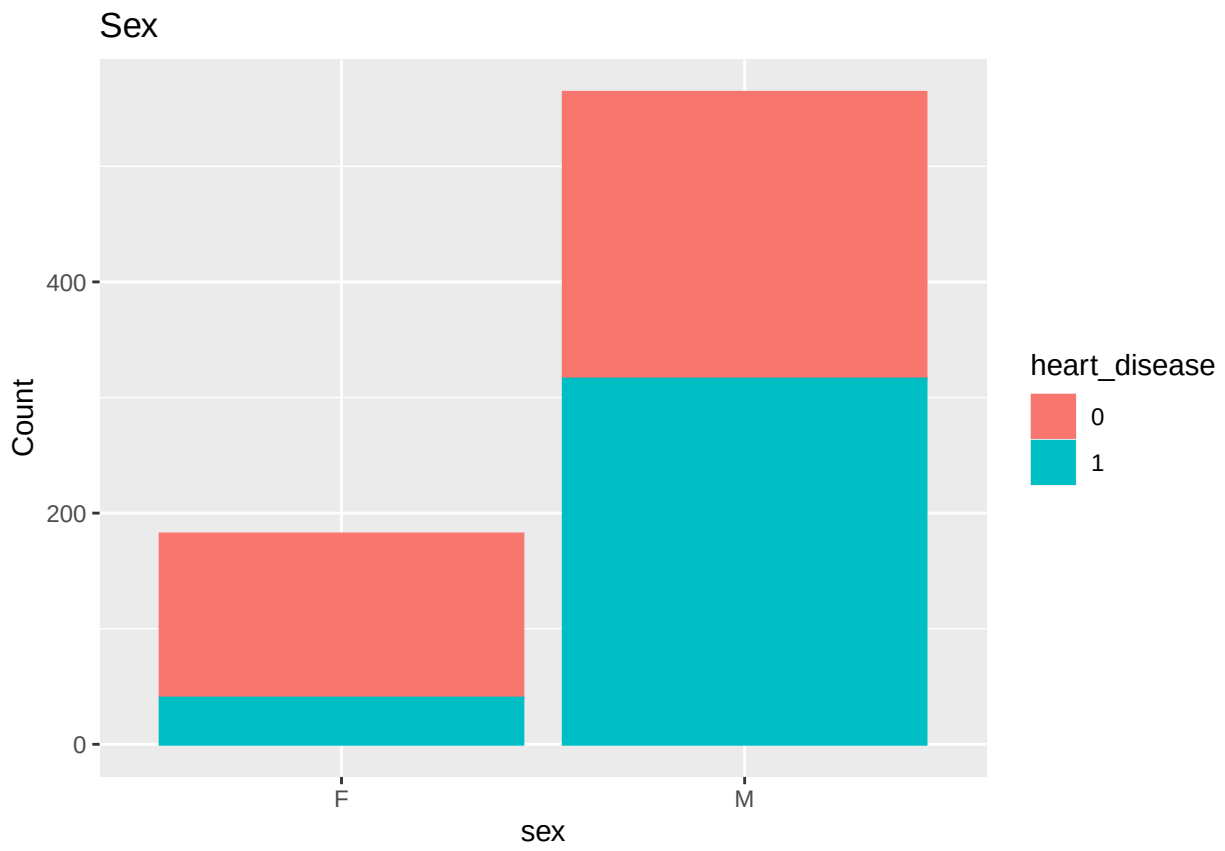
```
plot_2plots(heart, "oldpeak", heart_disease)
```



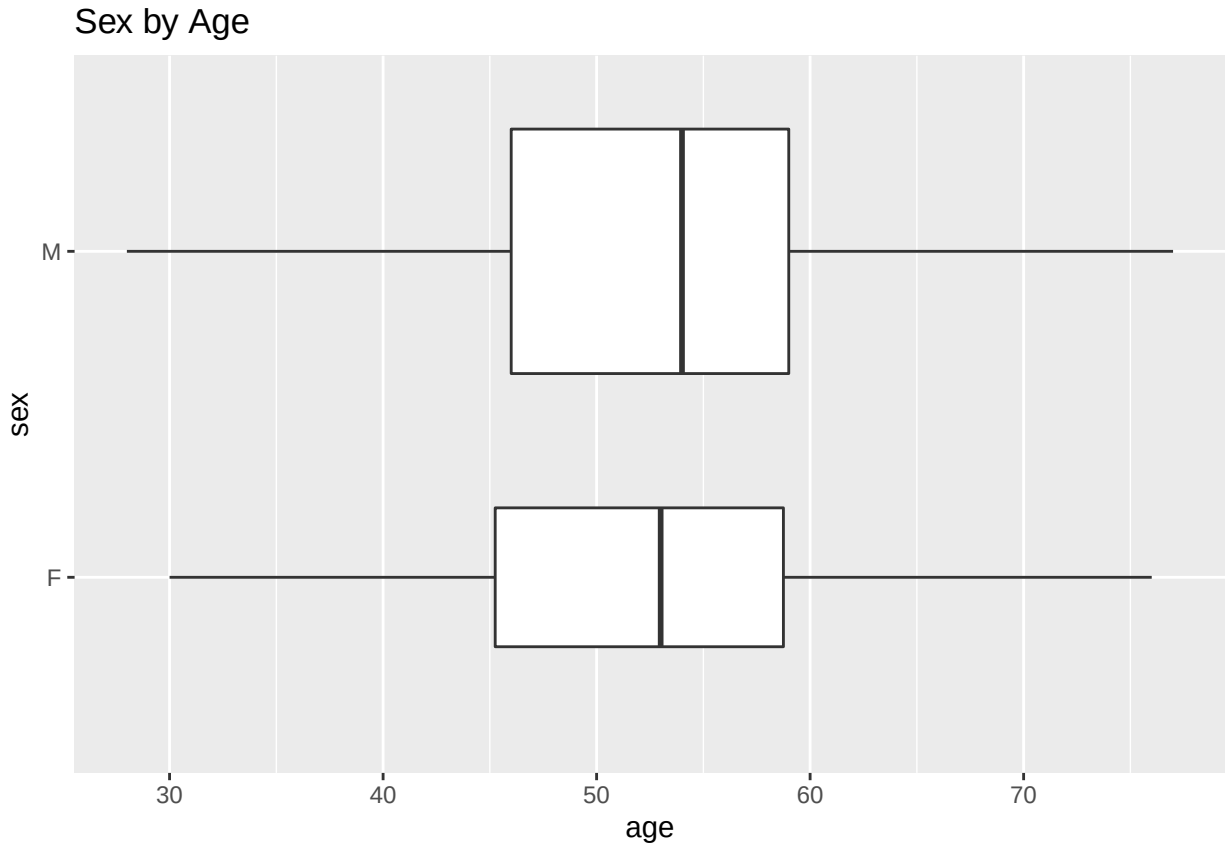
'Oldpeak' means a depressed/lower line on an ECG induced by exercise. We can easily see that people with high oldpeak get heart disease more than people with low oldpeak.

Categorical variables

```
#barplot of sex fill by heart_disease
options(repr.plot.width = 10, repr.plot.height = 10)
ggplot(heart) + geom_bar(aes(x = sex, color = heart_disease,
    fill = heart_disease)) + ggtitle("Sex") + labs(x = "sex",
    y = "Count")
```

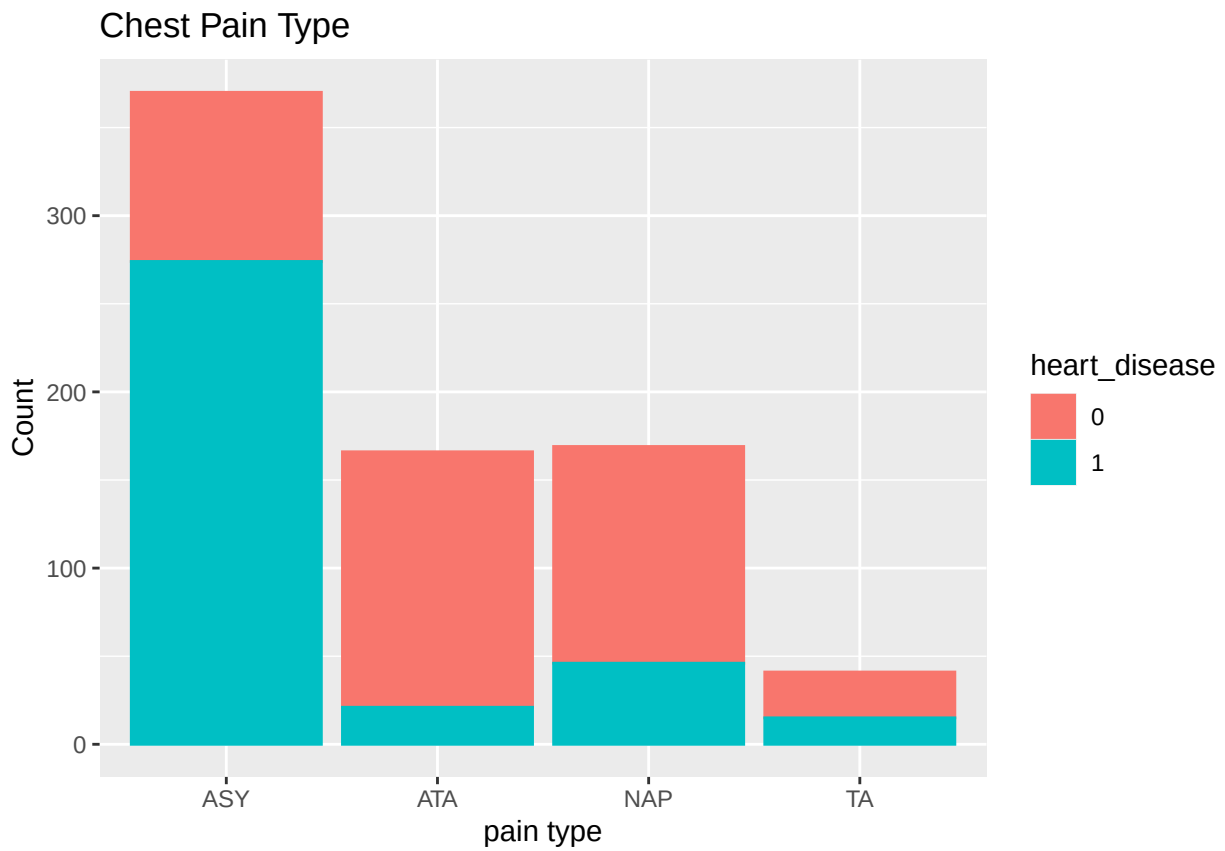



```
#boxplot of sex and age
ggplot(heart, aes(reorder(sex, age), age)) +
  geom_boxplot(varwidth = TRUE) +
  coord_flip() +
  labs(
    title = "Sex by Age",
    x = "sex"
  )
```



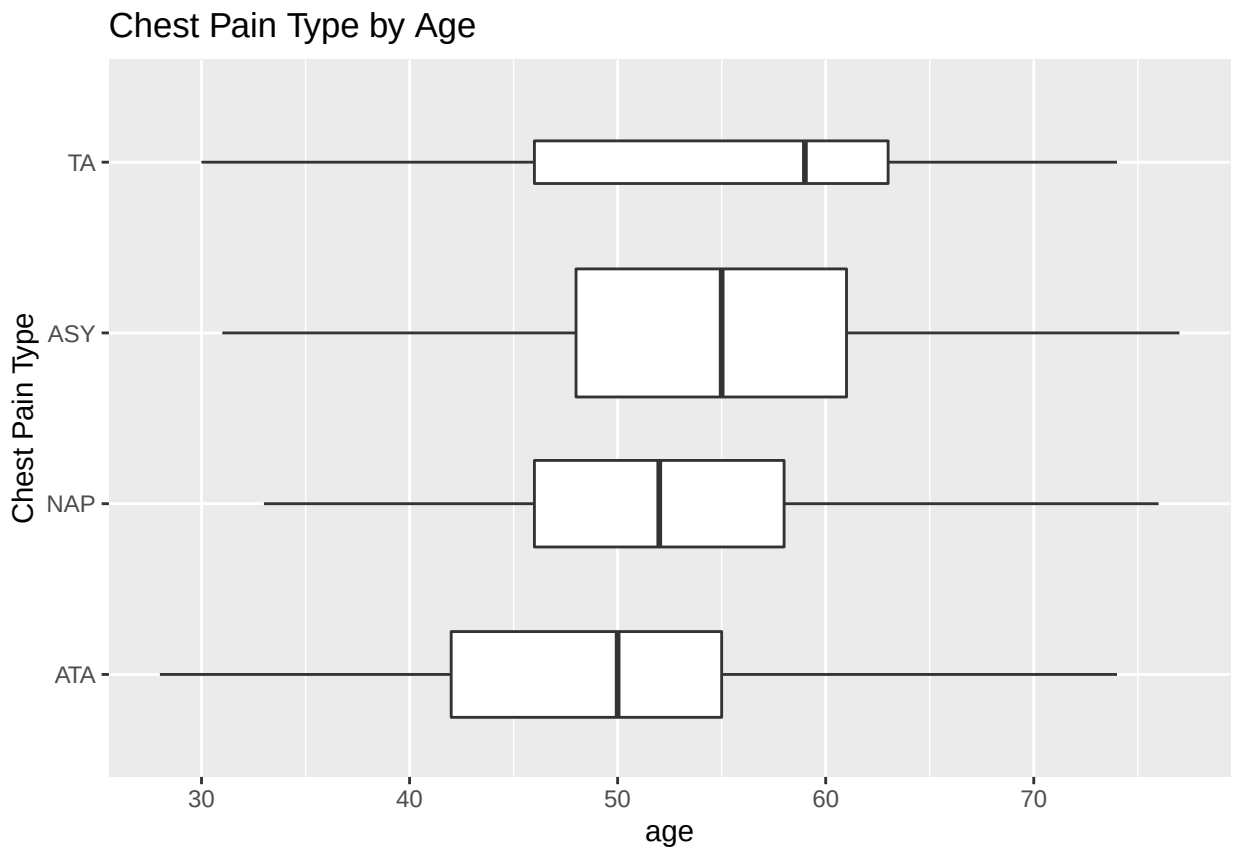
From the Sex graph, the observations of male is much higher than the one of female. Thus, we only concentrate on the heart disease proportion for each sex. Male with heart disease have slightly higher proportion among all male, but female with heart disease have a much lower proportion among all female. Also, with the help from sex by age boxplot, we see that the observations of male and female are basically selected from the same age range. Thus, we can conclude that, according to this dataset, male is much likely to get heart disease.

```
#barplot of chest pain type fill by heart_disease
options(repr.plot.width = 10, repr.plot.height = 10)
ggplot(heart) + geom_bar(aes(x = chest_pain_type, color = heart_disease,
    fill = heart_disease)) + ggtitle("Chest Pain Type") + labs(x = "pain type",
    y = "Count")
```

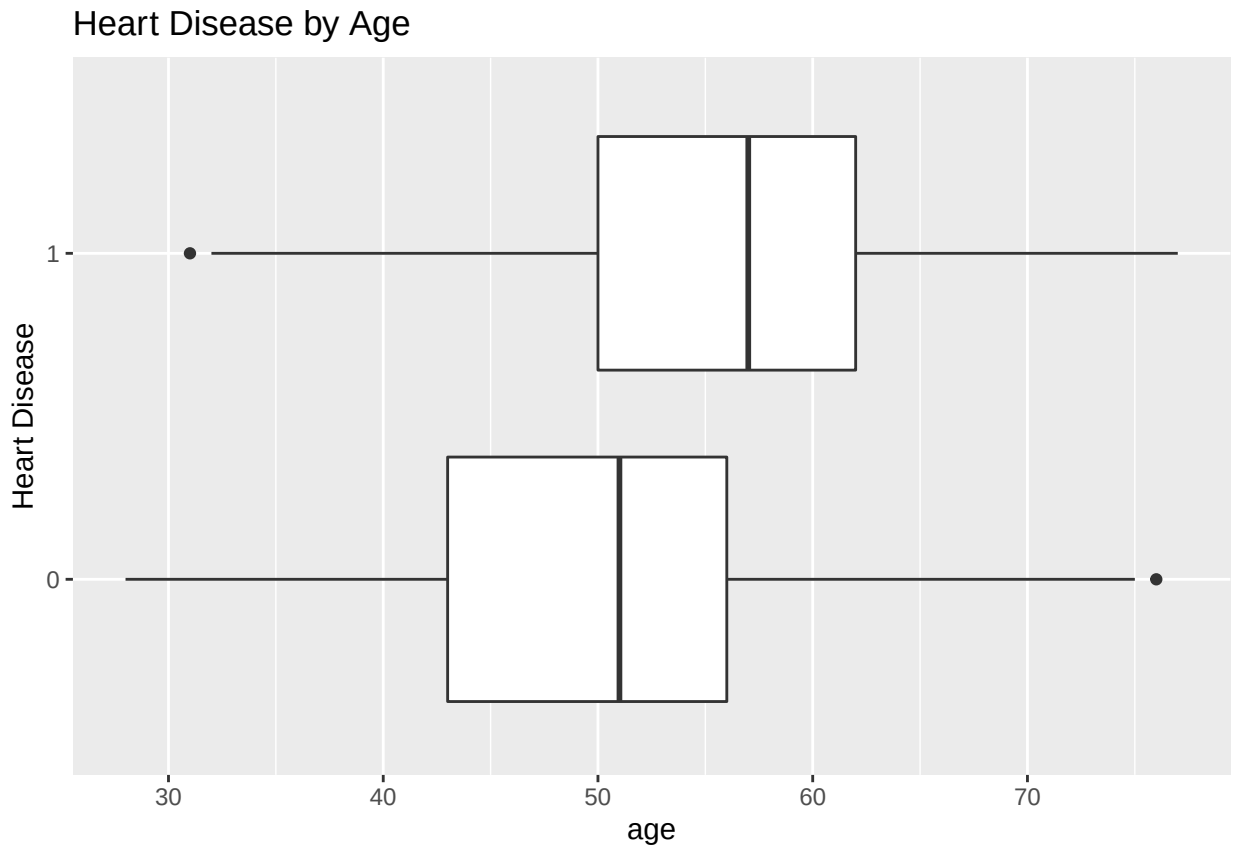


The graph chest pain type above clearly shows 'ASY' chest pain type has a high probobility to cause heart disease, but the others are not likewise; especially, 'ATA' and 'NAP' are less likely to cause heart disease. The observasions of 'TA' is fewer, but the proportion of causing heart disease or not is close to 50%.

```
#boxplot of chest pain type and age
ggplot(heart, aes(reorder(chest_pain_type, age), age)) +
  geom_boxplot(varwidth = TRUE) +
  coord_flip() +
  labs(
    title = "Chest Pain Type by Age",
    x = "Chest Pain Type"
  )
```

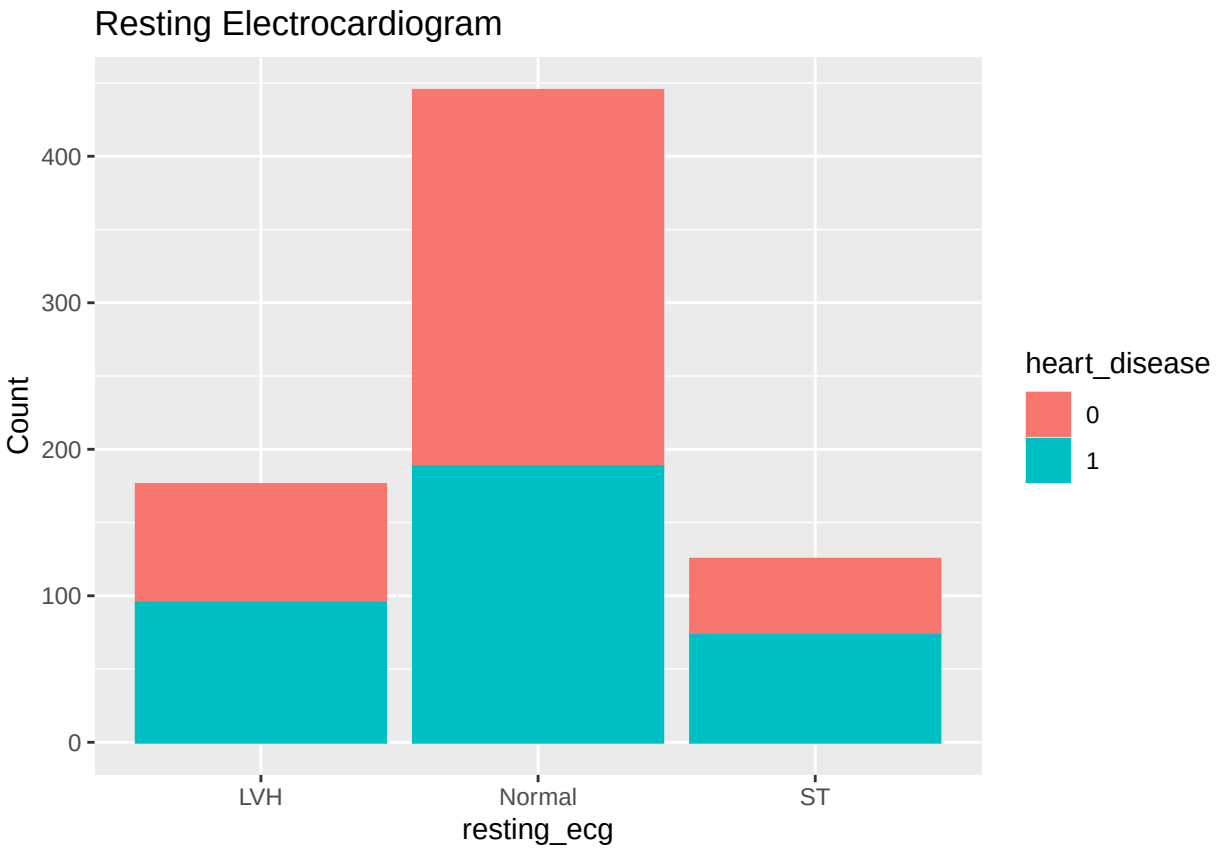



```
#boxplot of heart-disease and age
ggplot(heart, aes(reorder(heart_disease, age), age)) +
  geom_boxplot(varwidth = TRUE) +
  coord_flip() +
  labs(
    title = "Heart Disease by Age",
    x = "Heart Disease"
  )
```



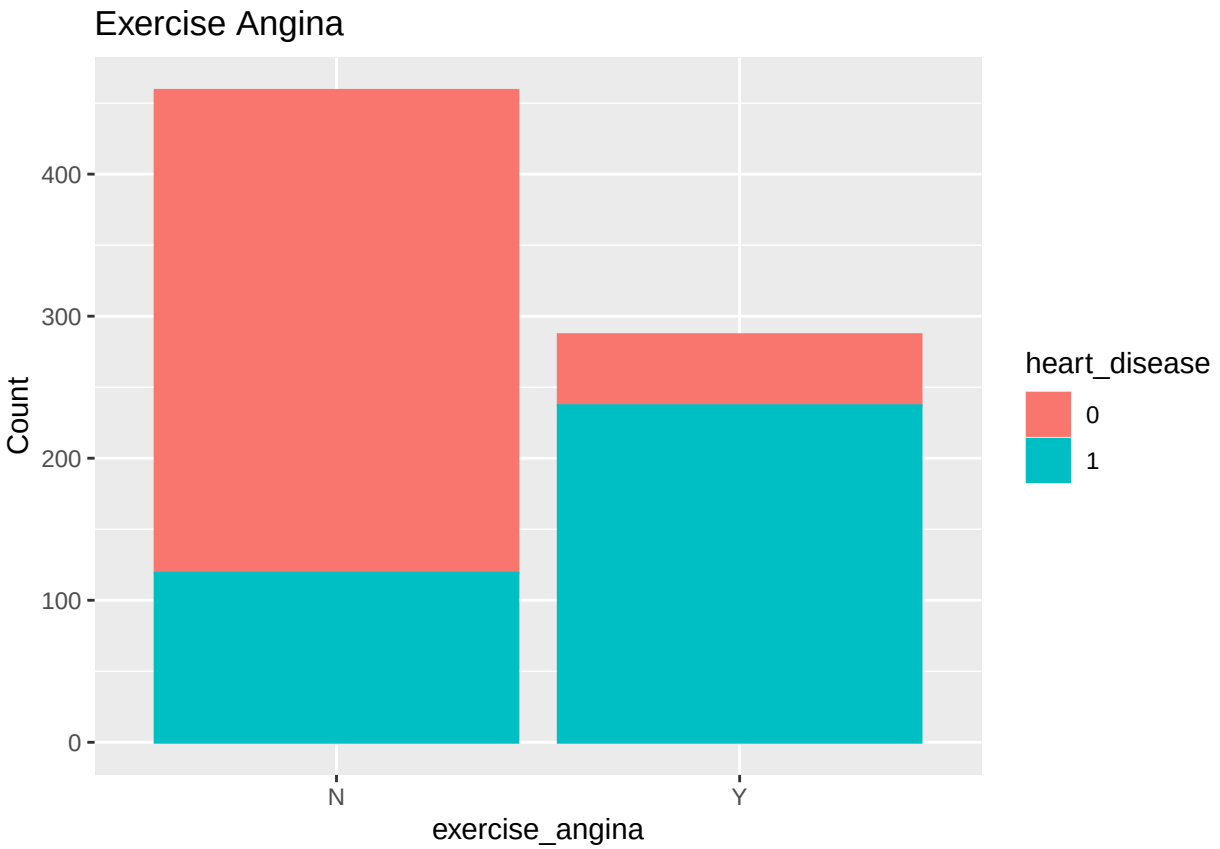
The 'chest pain type by age' graph and the 'heart disease by age' graph both supports the result from the previous chest pain type bar plot. Age can be an important cause of heart disease. Older people are more likely have heart disease than younger ones. Moreover, the 'ATA' and 'NAP' are seen more in younger people. Thus, these two types are less likely to relate to heart disease. 'TA' and 'ASY' occur more on older people. Thus, they have a higher relation to heart disease. And since, both 'TA' and 'ASY' are roughly ranged between age of 50-60, we can conclude that age of 50-60 might be a common period to get heart disease.

```
#barplot of restingECG fill by heart_disease
options(repr.plot.width = 10, repr.plot.height = 10)
ggplot(heart) + geom_bar(aes(x = resting_ecg, color = heart_disease,
    fill = heart_disease)) + ggtitle("Resting Electrocardiogram") + labs(x = "resting_ecg",
    y = "Count")
```



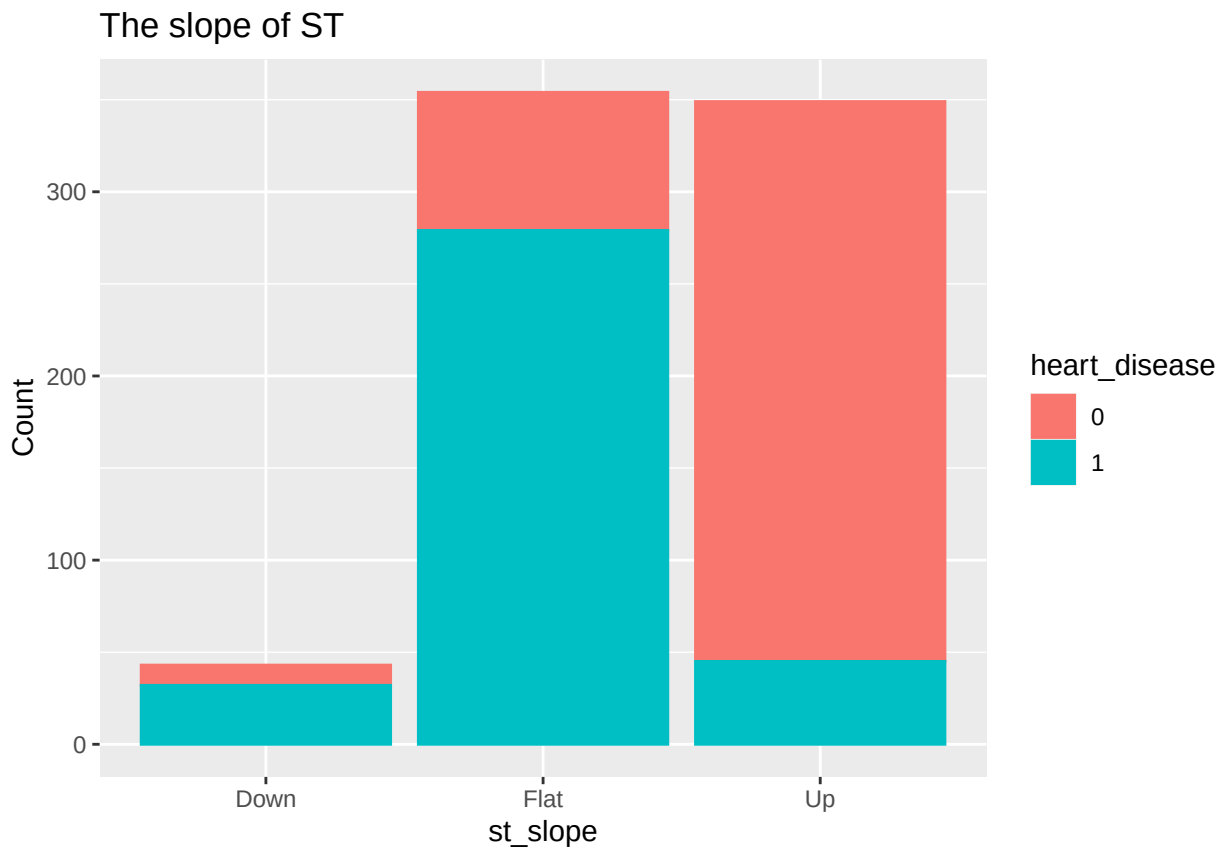
From the resting electrocardiogram barplot, base on this particular dataset, we can conclude that people with normal restingECG have a slightly lower proportion of having heart disease, people with 'LVH','ST' have a higher proportion of having heart disease, which make sense because normal resting ECG means the heart is at a relatively good condition.

```
#barplot of exerciseangina fill by heart_disease
options(repr.plot.width = 10, repr.plot.height = 10)
ggplot(heart) + geom_bar(aes(x = exercise_angina, color = heart_disease,
    fill = heart_disease)) + ggtitle("Exercise Angina") + labs(x = "exercise_angina",
    y = "Count")
```



From the Exercise angina graph, it is obvious that people with exercise induced angina are much more likely to have heart disease. Angina occurs when the heart needs more oxygen-rich blood but the demand is not met, which can be a big influencer of heart disease.

```
#barplot of st_slope fill by heart_disease
options(repr.plot.width = 10, repr.plot.height = 10)
ggplot(heart) + geom_bar(aes(x = st_slope, color = heart_disease,
    fill = heart_disease)) + ggtitle("The slope of ST") + labs(x = "st_slope",
    y = "Count")
```



From the slope of ST graph, we see that upward slope is less likely to be related to heart disease, however, flat and downward slope have a high relationship with heart disease.
Thus, we can conclude from the EDA section that Age is a important factor that influence the target variable and also other variables, except 'cholesterol' which isn't really influenced by age, however, other variables doesn't really correlate with each other.

3.0 Set Up for Model Building

With a deeper understanding of the data, we are going to prepare for the model building. I first split the data into train and test set, make a recipe and do a cross-validation for resampling.
train and testing split

```
heart_split <- heart %>%
  initial_split(prop = 0.8, strata = "heart_disease")

heart_train <- training(heart_split)
heart_test <- testing(heart_split)
```

```
dim(heart_train)
```

```
## [1] 596  12
```

```
dim(heart_test)
```

```
## [1] 150  12
```

There are 596 observations in the train set, and 150 observations in the test set. Stratified sampling was used on the response variable 'heart_disease'.

Recipe Building

We will create the recipe with dummy-coding categorical variables, and center and scale all predictors for model usage.

```
heart_recipe=recipe(heart_disease ~ . , data=heart_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors())
```

K-Fold Cross Validation

```
heart_folds <- vfold_cv(heart_train, v = 10, strata = heart_disease)  # 10-fold CV
```

3.0Model Building

We are ready for model building! The four model I used are Random Forest, XGBoost, KNN, Decision Tree. I first tried to create a regular grid to tune the random forest, and then tried to let the tidymodel tune by default. After comparing the result, they are both time-consuming, especially when I specified to a higher level, and the result is really close, so I decided to choose tuning by default.

Random Forest

```
rf_model <-
  rand_forest() %>%
  set_engine("randomForest") %>%
  set_mode("classification") %>%
  set_args(mtry = tune(), trees = tune(), min_n = tune())
```

```
rf_wf <-
  workflow() %>%
  add_model(rf_model) %>%
  add_recipe(heart_recipe)
rf_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
```

```
## * step_dummy()
## * step_normalize()
##
## -- Model ---------------------------------------------------------------------
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = tune()
##   trees = tune()
##   min_n = tune()
##
## Computational engine: randomForest
```

Create regular grid to tune with specified ranges.

```
# rf_grid=grid_regular(mtry(range = c(2,10)),
#                              trees(range = c(100, 1300)),
#                              min_n(range = c(5, 35)),
#                              levels = 4)
```

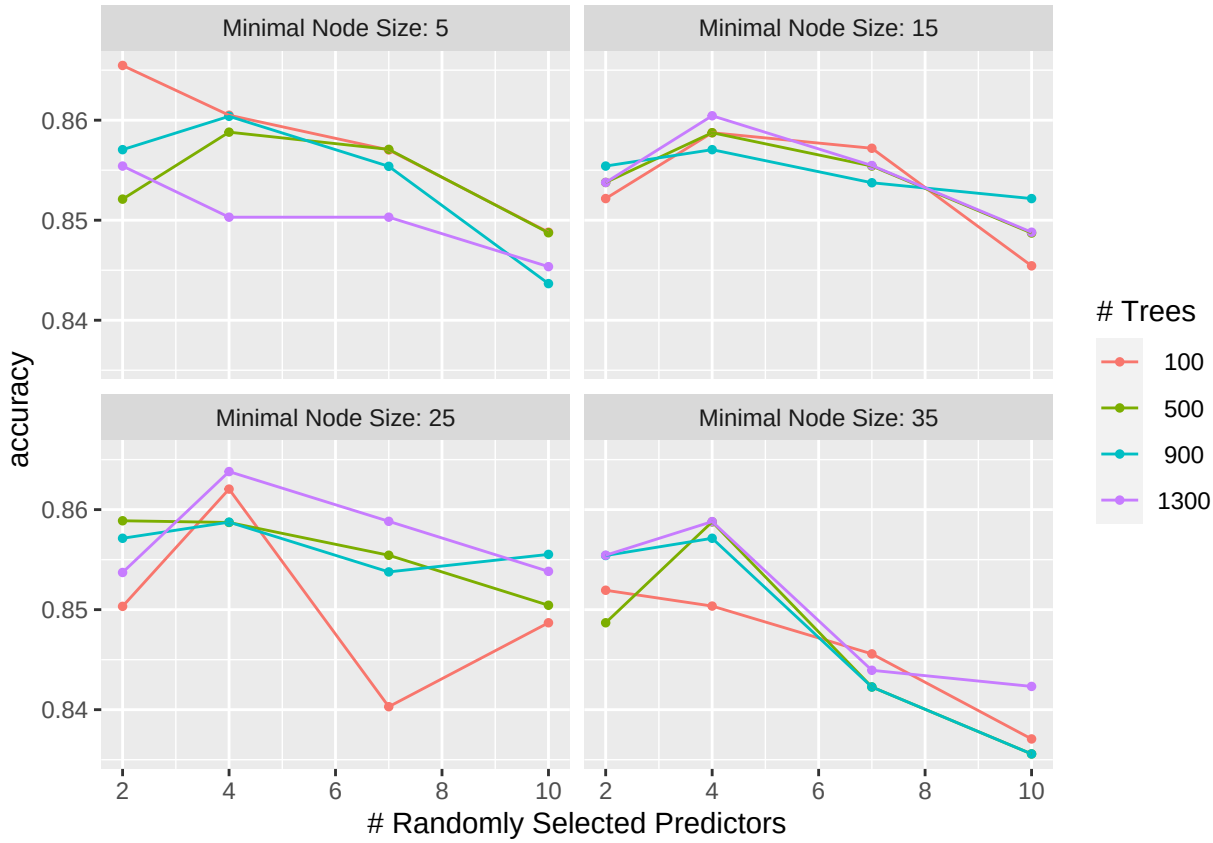
```
# rf_tune=tune_grid(rf_wf,
#                   resamples=heart_folds,
#                   grid=rf_grid,
#                   metrics=metric_set(accuracy))
```

I save the rf_grid, rf_tune into a rda for time-saving

```
#save(rf_grid, rf_tune, file=
#       "/Users/ritahan/Desktop/pstat131/rfgrid.rda")
```

```
load("/Users/ritahan/Desktop/pstat131/rfgrid.rda")
```

```
autoplot(rf_tune)
```




```
rf_tune_final <- rf_tune %>%
  select_best(metric = "accuracy")
rf_tune_final
```

```
## # A tibble: 1 x 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1     2   100     5 Preprocessor1_Model01
```

```
rf_tune_wf <- rf_wf %>%
  finalize_workflow(rf_tune_final)
rf_tune_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_normalize()
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (classification)
```

```
##
## Main Arguments:
##   mtry = 2
##   trees = 100
##   min_n = 5
##
## Computational engine: randomForest
```

```
#fitting the final workflow to the train set(create regular grid to tune)
rf_tune_fit=fit(rf_tune_wf, heart_train)
#accuracy of rf model
rf_tune_acc <- predict(rf_tune_fit, new_data = heart_train, type = "class") %>%
  bind_cols(heart_train %>%
              select(heart_disease)) %>%
  accuracy(truth = heart_disease, estimate = .pred_class)
rf_tune_acc
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.926
```

rf on test set

```
rf_tune_acc_test <- predict(rf_tune_fit, new_data = heart_test,
                            type = "class") %>%
  bind_cols(heart_test %>%
              select(heart_disease)) %>%
  accuracy(truth = heart_disease, estimate = .pred_class)
rf_tune_acc_test
```

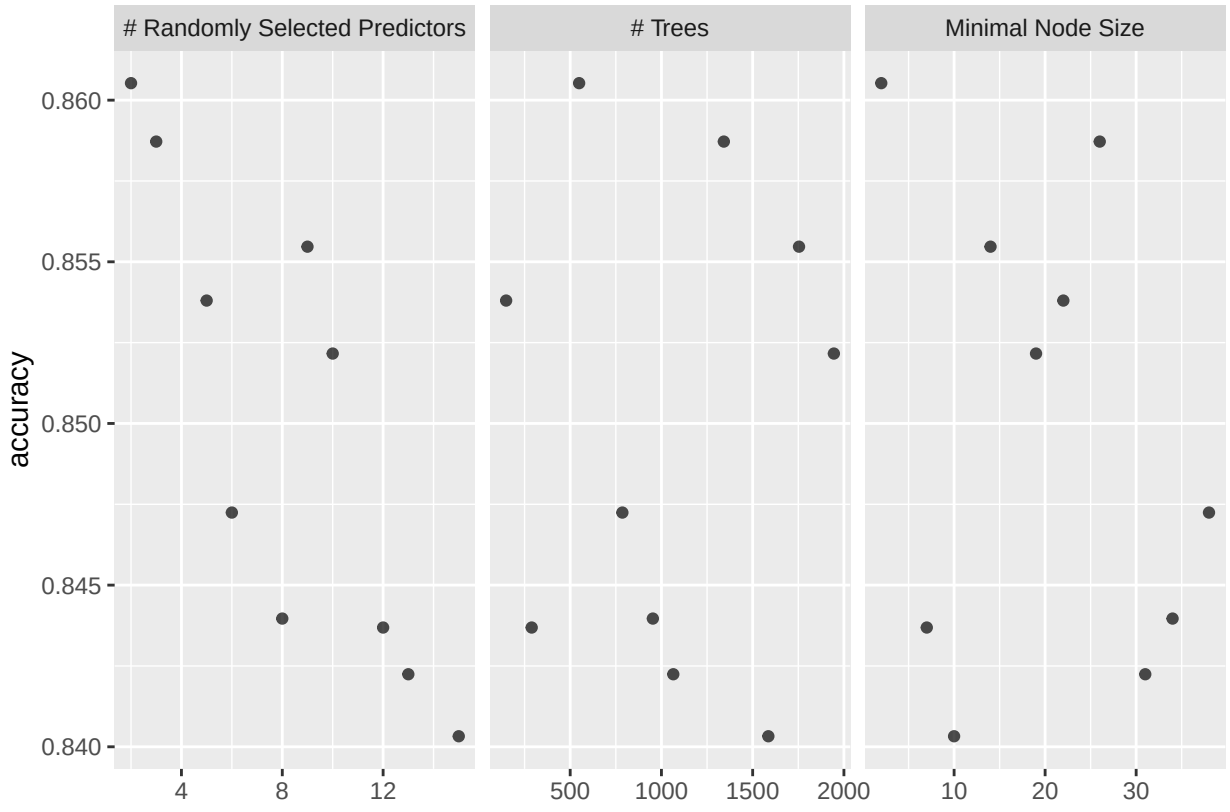
```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary          0.88
```

Next, I will try do not create grid and let 'tidymodel' to chose the range for parameters by default.

```
rf_results <-
  rf_wf %>%
  tune_grid(resamples = heart_folds,
            metrics = metric_set(accuracy)
  )
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

```
autoplot(rf_results)
```




```
rf_results %>%
  collect_metrics()
```

```
## # A tibble: 10 x 9
##     mtry trees min_n .metric  .estimator  mean     n std_err .config
##    <int> <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1     9  1754    14 accuracy binary     0.855    10 0.0128  Preprocessor1_Mode~
##  2     5   149    22 accuracy binary     0.854    10 0.0109  Preprocessor1_Mode~
##  3    10  1945    19 accuracy binary     0.852    10 0.0117  Preprocessor1_Mode~
##  4     8   953    34 accuracy binary     0.844    10 0.00752 Preprocessor1_Mode~
##  5     2   549     2 accuracy binary     0.861    10 0.0122  Preprocessor1_Mode~
##  6    15  1586    10 accuracy binary     0.840    10 0.0168  Preprocessor1_Mode~
##  7    13  1065    31 accuracy binary     0.842    10 0.00951 Preprocessor1_Mode~
##  8     3  1342    26 accuracy binary     0.859    10 0.0131  Preprocessor1_Mode~
##  9     6   786    38 accuracy binary     0.847    10 0.00742 Preprocessor1_Mode~
## 10    12   289     7 accuracy binary     0.844    10 0.0167  Preprocessor1_Mode~
```

```
rf_param_final <- rf_results %>%
  select_best(metric = "accuracy")
rf_param_final
```

```
## # A tibble: 1 x 4
##    mtry trees min_n .config
##   <int> <int> <int> <chr>
## 1     2   549     2 Preprocessor1_Model05
```

```
final_rf_wf <- rf_wf %>%
  finalize_workflow(rf_param_final)
final_rf_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_normalize()
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = 2
##   trees = 549
##   min_n = 2
##
## Computational engine: randomForest
```

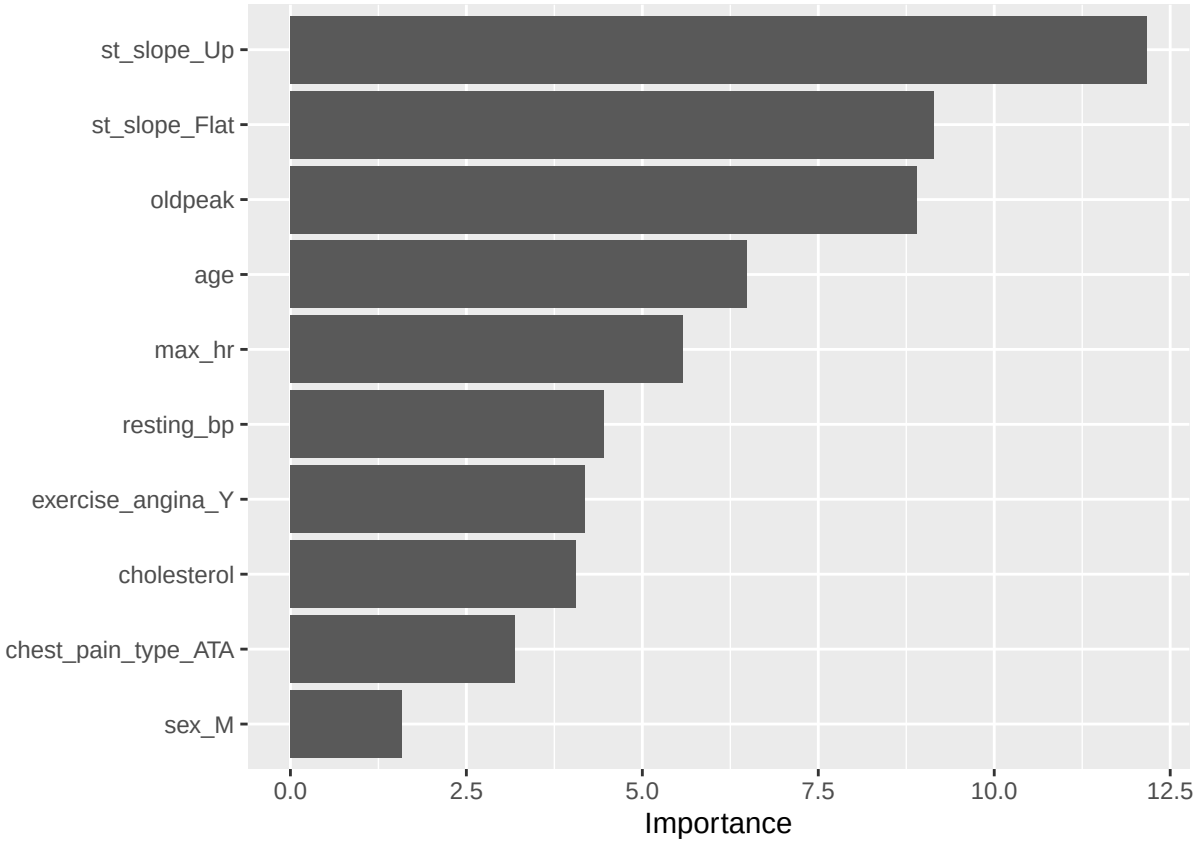
```
#fitting the final workflow to the train set
rf_fit=fit(final_rf_wf, heart_train)
#accuracy of rf model
rf_acc <- predict(rf_fit, new_data = heart_train, type = "class") %>%
  bind_cols(heart_train %>%
              select(heart_disease)) %>%
  accuracy(truth = heart_disease, estimate = .pred_class)
rf_acc
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.950
```

```
#fitting the final workflow to the test set(tuning by default)
rf_fit=fit(final_rf_wf, heart_test)
#accurary of rf model
rf_acc_test <- predict(rf_fit, new_data = heart_test, type = "class") %>%
  bind_cols(heart_test %>%
              select(heart_disease)) %>%
  accuracy(truth = heart_disease, estimate = .pred_class)
rf_acc_test
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary          0.96
```

```
rf_fit %>%
  extract_fit_engine() %>%
  vip()
```



Through comparing the results of creating a regular grid to tune and letting 'tidymodel' to chose the range for parameters and tune by default. I found that creating regular grid to tune is more time-consuming and the accuracy on test set is really close and even worse than the tuning by default(0.88<0.89). Thus, for the rest of the model, I choose to tune it by default for time-saving and relatively good accuracy.

XGBoost

```
xgb_model <-
  boost_tree() %>%
  set_engine("xgboost") %>%
  set_mode("classification") %>%
  set_args( trees = tune(),min_n = tune(),
            tree_depth = tune(),
            learn_rate = tune(),
            loss_reduction = tune(),
            sample_size = tune(),
            stop_iter = tune())
```

```r
xgb_wf <-
  workflow() %>%
  add_model(xgb_model) %>%
  add_recipe(heart_recipe)
xgb_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: boost_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_normalize()
##
## -- Model -----------------------------------------------------------------------
## Boosted Tree Model Specification (classification)
##
## Main Arguments:
##   trees = tune()
##   min_n = tune()
##   tree_depth = tune()
##   learn_rate = tune()
##   loss_reduction = tune()
##   sample_size = tune()
##   stop_iter = tune()
##
## Computational engine: xgboost
```

```r
# xgb_results <-
#   xgb_wf %>%
#   tune_grid(resamples = heart_folds,
#             metrics = metric_set(accuracy)
#   )
```

```r
#save(xgb_results, file="/Users/ritahan/Desktop/pstat131/xgbresults.rda")
```

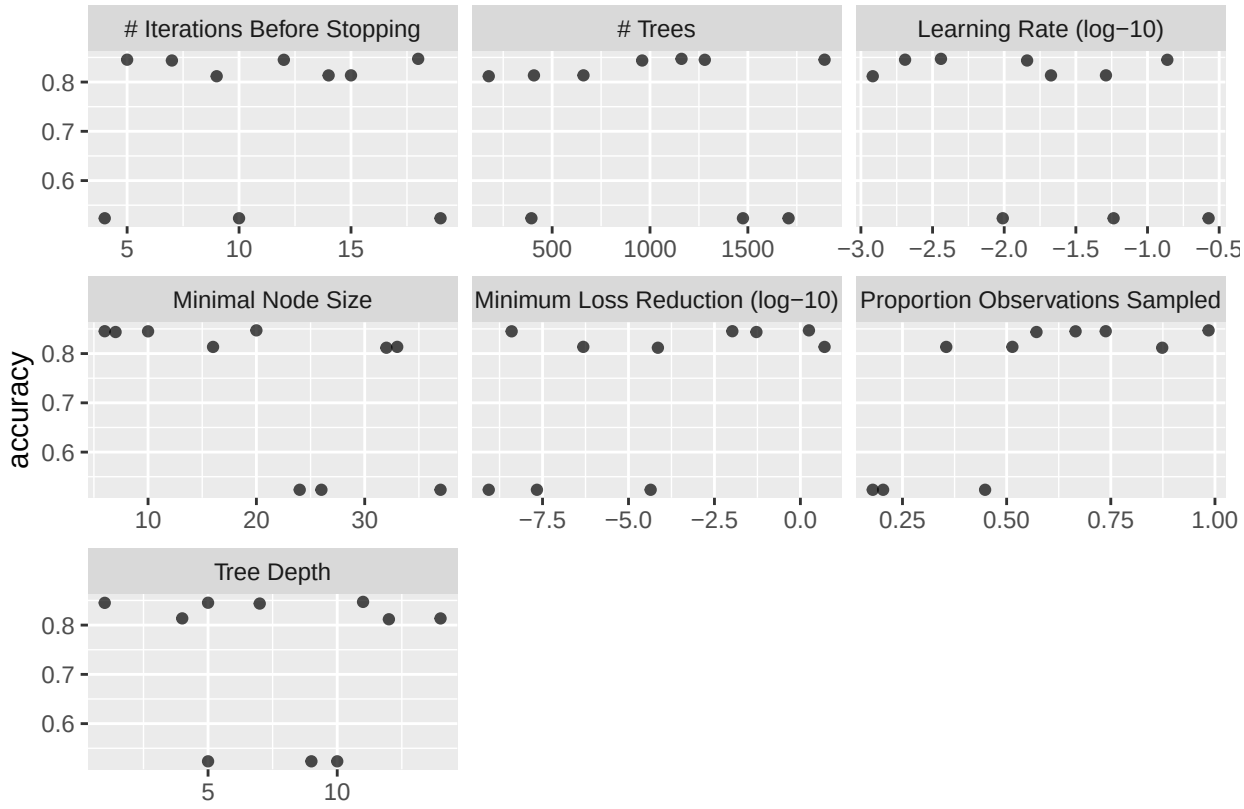
```r
load("/Users/ritahan/Desktop/pstat131/xgbresults.rda")
```

```r
xgb_results %>%
  collect_metrics()
```

```
## # A tibble: 10 x 13
##    trees min_n tree_depth learn~1 loss_r~2 sampl~3 stop_~4 .metric .esti~5  mean
##    <int> <int>      <int>   <dbl>    <dbl>   <dbl>   <int> <chr>   <chr>   <dbl>
##  1  1892     6          5 0.00203 1.05e- 2   0.738       5 accura~ binary  0.845
##  2   960     7          7 0.0144  5.25e- 2   0.572       7 accura~ binary  0.844
##  3  1280    10          1 0.137   4.00e- 9   0.665      12 accura~ binary  0.845
##  4   407    16         14 0.0212  5.05e+ 0   0.355      14 accura~ binary  0.814
##  5  1161    20         11 0.00362 1.77e+ 0   0.985      18 accura~ binary  0.847
```

```
##  6  1475    24          5 0.00977 2.18e- 8   0.179      10 accura~ binary  0.524
##  7   394    26         10 0.0580  8.63e-10   0.204      19 accura~ binary  0.524
##  8   176    32         12 0.00121 7.17e- 5   0.873       9 accura~ binary  0.812
##  9   660    33          4 0.0513  4.84e- 7   0.514      15 accura~ binary  0.814
## 10  1708    37          9 0.267   4.41e- 5   0.448       4 accura~ binary  0.524
## # ... with 3 more variables: n <int>, std_err <dbl>, .config <chr>, and
## #   abbreviated variable names 1: learn_rate, 2: loss_reduction,
## #   3: sample_size, 4: stop_iter, 5: .estimator
```

```
autoplot(xgb_results)
```




```
xgb_param_final <- xgb_results %>%
  select_best(metric = "accuracy")
xgb_param_final
```

```
## # A tibble: 1 x 8
##   trees min_n tree_depth learn_rate loss_reduction sample_size stop_iter .config
##   <int> <int>      <int>      <dbl>          <dbl>       <dbl>     <int> <chr>
## 1  1161    20         11    0.00362           1.77       0.985        18 Prepro~
```

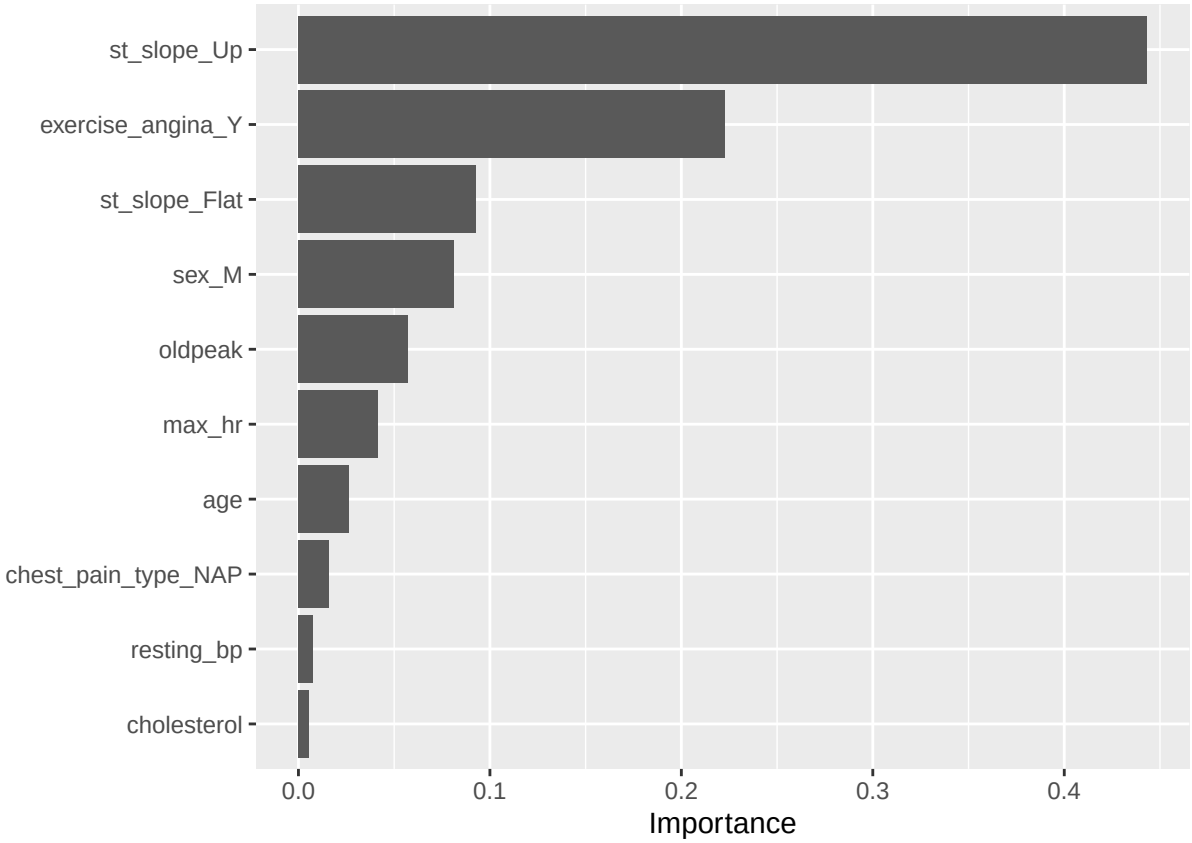
```
final_xgb_wf <- xgb_wf %>%
  finalize_workflow(xgb_param_final)
final_xgb_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: boost_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_normalize()
##
## -- Model -----------------------------------------------------------------------
## Boosted Tree Model Specification (classification)
##
## Main Arguments:
##   trees = 1161
##   min_n = 20
##   tree_depth = 11
##   learn_rate = 0.00362082579057878
##   loss_reduction = 1.7711445798218
##   sample_size = 0.984743045249488
##   stop_iter = 18
##
## Computational engine: xgboost
```

```
#fitting the final workflow to the train set
xgb_fit=fit(final_xgb_wf, heart_train)
#accurary of xgb model
xgb_acc <- predict(xgb_fit, new_data = heart_train, type = "class") %>%
  bind_cols(heart_train %>%
              select(heart_disease)) %>%
  accuracy(truth = heart_disease, estimate = .pred_class)
xgb_acc
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.859
```

```
xgb_fit %>%
  extract_fit_engine() %>%
  vip()
```



KNN

```
#install.packages("kknn")
library(kknn)
```

```
knn_model <-
  nearest_neighbor() %>%
  set_engine("kknn") %>%
  set_mode("classification") %>%
  set_args(neighbors = tune(),
           weight_func = tune(),
           dist_power = tune())
```

```
knn_wf <-
  workflow() %>%
  add_model(knn_model) %>%
  add_recipe(heart_recipe)
knn_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
```

```
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_normalize()
##
## -- Model ----------------------------------------------------------------------
## K-Nearest Neighbor Model Specification (classification)
##
## Main Arguments:
##   neighbors = tune()
##   weight_func = tune()
##   dist_power = tune()
##
## Computational engine: kknn
```

```
# knn_results <-
#   knn_wf %>%
#   tune_grid(resamples = heart_folds,
#             metrics = metric_set(accuracy)
#   )
```

```
#save(knn_results, file="/Users/ritahan/Desktop/pstat131/knnresults.rda")
```

```
load("/Users/ritahan/Desktop/pstat131/knnresults.rda")
```

```
knn_results %>%
  collect_metrics()
```

```
## # A tibble: 10 x 9
##    neighbors weight_func  dist_power .metric .esti~1  mean     n std_err .config
##        <int> <chr>             <dbl> <chr>   <chr>   <dbl> <int>   <dbl> <chr>
##  1         8 biweight          1.78  accura~ binary  0.852    10  0.0135 Prepro~
##  2        13 cos               1.13  accura~ binary  0.860    10  0.0170 Prepro~
##  3         6 epanechnikov      0.122 accura~ binary  0.844    10  0.0181 Prepro~
##  4         4 gaussian          0.446 accura~ binary  0.842    10  0.0181 Prepro~
##  5         9 inv               1.83  accura~ binary  0.855    10  0.0170 Prepro~
##  6         2 optimal           0.664 accura~ binary  0.827    10  0.0145 Prepro~
##  7        14 rank              0.750 accura~ binary  0.862    10  0.0116 Prepro~
##  8         4 rectangular       0.980 accura~ binary  0.847    10  0.0153 Prepro~
##  9        10 triangular        1.25  accura~ binary  0.864    10  0.0160 Prepro~
## 10        11 triweight         1.60  accura~ binary  0.847    10  0.0130 Prepro~
## # ... with abbreviated variable name 1: .estimator
```

```
autoplot(knn_results)
```


```
knn_param_final <- knn_results %>%
  select_best(metric = "accuracy")
knn_param_final
```

```
## # A tibble: 1 x 4
##   neighbors weight_func dist_power .config
##       <int> <chr>            <dbl> <chr>
## 1        10 triangular        1.25 Preprocessor1_Model09
```

```
final_knn_wf <- knn_wf %>%
  finalize_workflow(knn_param_final)
final_knn_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_normalize()
```

```
##
## -- Model ----------------------------------------------------------------------
## K-Nearest Neighbor Model Specification (classification)
##
## Main Arguments:
##   neighbors = 10
##   weight_func = triangular
##   dist_power = 1.24821168610826
##
## Computational engine: kknn
```

```
#fitting the final workflow to the train set
knn_fit=fit(final_knn_wf, heart_train)
#accurary of knn model
knn_acc <- predict(knn_fit, new_data = heart_train, type = "class") %>%
  bind_cols(heart_train %>%
              select(heart_disease)) %>%
  accuracy(truth = heart_disease, estimate = .pred_class)
knn_acc
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.950
```

Decision tree

```
dt_model <-
  decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification") %>%
  set_args(cost_complexity = tune(),
               tree_depth = tune(),
               min_n = tune())
```

```
dt_wf <-
  workflow() %>%
  add_model(dt_model) %>%
  add_recipe(heart_recipe)
dt_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: decision_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_normalize()
##
```

```
## -- Model ---------------------------------------------------------------------
## Decision Tree Model Specification (classification)
##
## Main Arguments:
##   cost_complexity = tune()
##   tree_depth = tune()
##   min_n = tune()
##
## Computational engine: rpart
```

```
dt_results <-
  dt_wf %>%
  tune_grid(resamples = heart_folds,
            metrics = metric_set(accuracy)
  )
```

```
#save(dt_results, file="/Users/ritahan/Desktop/pstat131/dtresults.rda")
```

```
load("/Users/ritahan/Desktop/pstat131/dtresults.rda")
```

```
dt_results %>%
  collect_metrics()
```

```
## # A tibble: 10 x 9
##    cost_complexity tree_depth min_n .metric  .esti~1  mean     n std_err .config
##              <dbl>      <int> <int> <chr>    <chr>   <dbl> <int>   <dbl> <chr>
##  1        1.64e-10          1     3 accuracy binary  0.812    10 0.0127  Prepro~
##  2        3.80e- 9          3    11 accuracy binary  0.815    10 0.0140  Prepro~
##  3        1.44e- 7         11    36 accuracy binary  0.792    10 0.00902 Prepro~
##  4        2.00e- 4          5    38 accuracy binary  0.795    10 0.0102  Prepro~
##  5        2.92e- 6         14    29 accuracy binary  0.819    10 0.0114  Prepro~
##  6        1.59e- 8         13    17 accuracy binary  0.815    10 0.0133  Prepro~
##  7        1.14e- 4          6    22 accuracy binary  0.815    10 0.0182  Prepro~
##  8        4.91e- 6         10     7 accuracy binary  0.760    10 0.0210  Prepro~
##  9        3.73e- 2          8    28 accuracy binary  0.805    10 0.0110  Prepro~
## 10        5.97e- 3          7    15 accuracy binary  0.825    10 0.0154  Prepro~
## # ... with abbreviated variable name 1: .estimator
```

```
dt_param_final <- dt_results %>%
  select_best(metric = "accuracy")
dt_param_final
```

```
## # A tibble: 1 x 4
##   cost_complexity tree_depth min_n .config
##             <dbl>      <int> <int> <chr>
## 1         0.00597          7    15 Preprocessor1_Model10
```

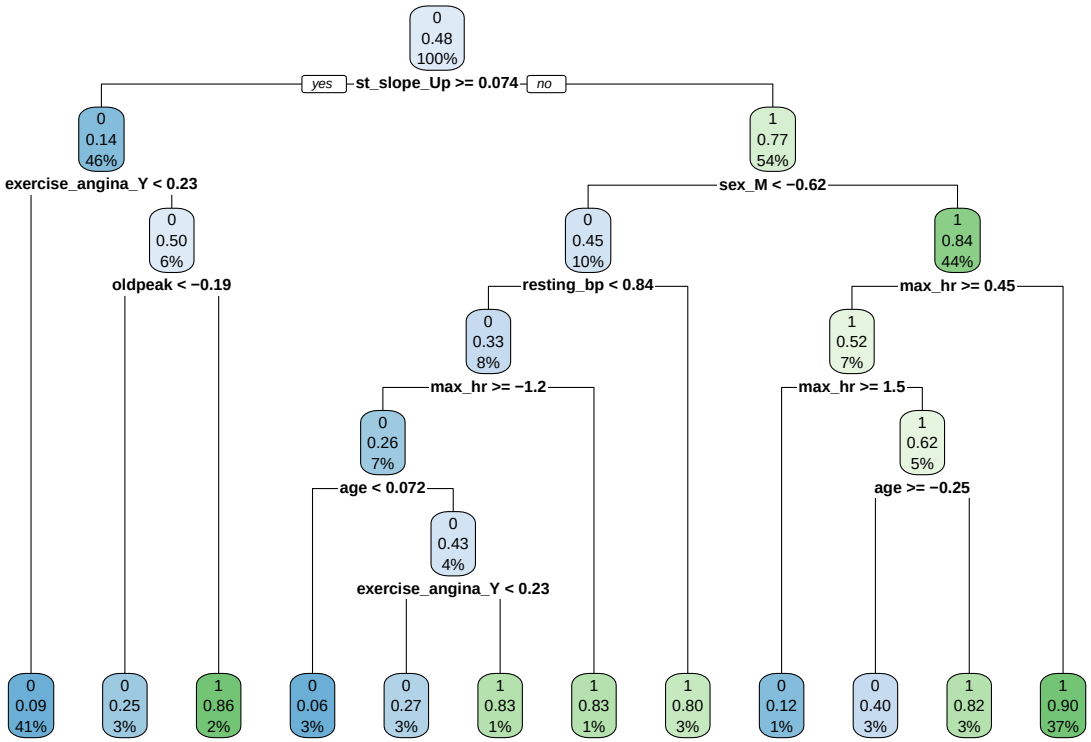
```
final_dt_wf <- dt_wf %>%
  finalize_workflow(dt_param_final)
final_dt_wf
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: decision_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_dummy()
## * step_normalize()
##
## -- Model -----------------------------------------------------------------------
## Decision Tree Model Specification (classification)
##
## Main Arguments:
##   cost_complexity = 0.00597345983576604
##   tree_depth = 7
##   min_n = 15
##
## Computational engine: rpart
```

```
#fitting the final workflow to the train set
dt_fit=fit(final_dt_wf, heart_train)
#accurary of knn model
dt_acc <- predict(dt_fit, new_data = heart_train, type = "class") %>%
  bind_cols(heart_train %>%
              select(heart_disease)) %>%
  accuracy(truth = heart_disease, estimate = .pred_class)
dt_acc
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.883
```

```
dt_fit %>%
  extract_fit_engine() %>%
  rpart.plot()
```



Accuracy of Models

```
rf_table_acc <- augment(rf_fit, new_data = heart_train) %>%
  accuracy(heart_disease, estimate = .pred_class) %>%
  select(.estimate)

xgb_table_acc <- augment(xgb_fit, new_data = heart_train) %>%
  accuracy(heart_disease, estimate = .pred_class) %>%
  select(.estimate)

knn_table_acc <- augment(knn_fit, new_data = heart_train) %>%
  accuracy(heart_disease, estimate = .pred_class) %>%
  select(.estimate)

dt_table_acc <- augment(dt_fit, new_data = heart_train) %>%
  accuracy(heart_disease, estimate = .pred_class) %>%
  select(.estimate)

heart_disease_train_acc=c(rf_table_acc$.estimate,
                          xgb_table_acc$.estimate,
                          knn_table_acc$.estimate,
                          dt_table_acc$.estimate)

model_names=c('Random Forest',
```

```
               'XGBoost',
               'K-Nearest Neighbor',
               'Decision Tree')
```

```
acc_table <- tibble(Model = model_names,
                            Accuracy = heart_disease_train_acc)

acc_table
```

```
## # A tibble: 4 x 2
##   Model               Accuracy
##   <chr>                  <dbl>
## 1 Random Forest          0.841
## 2 XGBoost                0.859
## 3 K-Nearest Neighbor     0.950
## 4 Decision Tree          0.883
```

Roc_auc of models

```
rf_table_auc <- augment(rf_fit, new_data = heart_train) %>%
  roc_auc(heart_disease, estimate = .pred_0) %>%
  select(.estimate)

xgb_table_auc <- augment(xgb_fit, new_data = heart_train) %>%
  roc_auc(heart_disease, estimate = .pred_0) %>%
  select(.estimate)

knn_table_auc <- augment(knn_fit, new_data = heart_train) %>%
  roc_auc(heart_disease, estimate = .pred_0) %>%
  select(.estimate)

dt_table_auc <- augment(dt_fit, new_data = heart_train) %>%
  roc_auc(heart_disease, estimate = .pred_0) %>%
  select(.estimate)

heart_disease_train_auc=c(rf_table_auc$.estimate,
                          xgb_table_auc$.estimate,
                          knn_table_auc$.estimate,
                          dt_table_auc$.estimate)
```

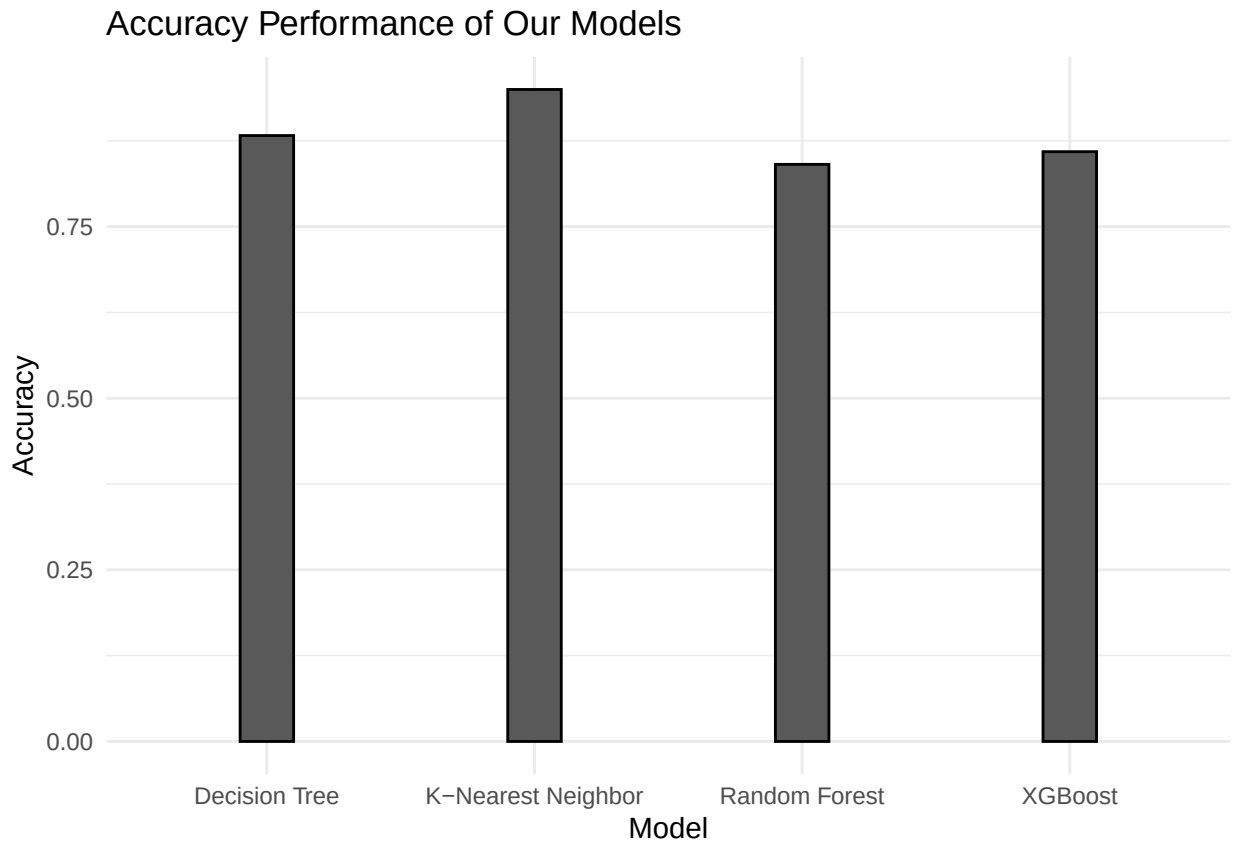
```
auc_table <- tibble(Model = model_names,
                            Roc_auc = heart_disease_train_auc)

auc_table
```

```
## # A tibble: 4 x 2
##   Model               Roc_auc
##   <chr>                 <dbl>
## 1 Random Forest         0.911
## 2 XGBoost               0.930
## 3 K-Nearest Neighbor    0.994
## 4 Decision Tree         0.905
```

Graphs for Model Evaluation

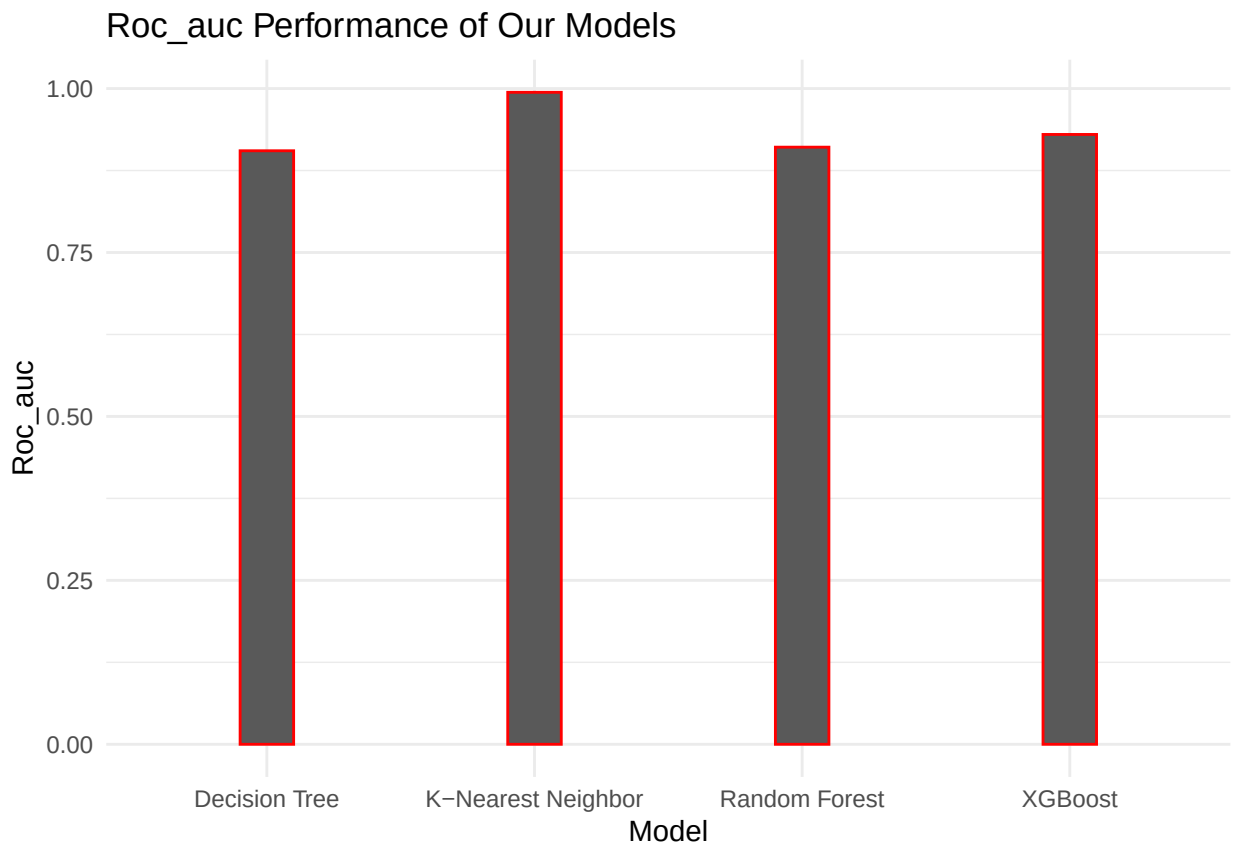
Barplot of models' accuracy

```
acc_table_bar_plot <- ggplot(acc_table,
       aes(x = Model, y = Accuracy)) +
  geom_bar(stat = "identity", width=0.2,  color = "black") +
  labs(title = "Accuracy Performance of Our Models") +
  theme_minimal()

acc_table_bar_plot
```



Barplot of Models' roc-auc

```
auc_table_bar_plot <- ggplot(auc_table,
       aes(x = Model, y = Roc_auc)) +
  geom_bar(stat = "identity", width=0.2,  color = "red") +
  labs(title = "Roc_auc Performance of Our Models") +
  theme_minimal()

auc_table_bar_plot
```



From all the graphs above, all of the models perform fairly well. Among them, it is obvious that knn is the best-performing model. Next, we are going to fit it on to the test set.

Predicting

```
knn_heart_predict <- predict(knn_fit,  # fitting our model to testing data
                             new_data = heart_test,
                             type = "class")

knn_heart_predict_with_actual <- knn_heart_predict %>%
  bind_cols(heart_test)  # adding the actual values side by side to our predicted values

knn_heart_predict_with_actual
```

```
## # A tibble: 150 x 13
##    .pred_cl~1   age sex   chest~2 resti~3 chole~4 fasti~5 resti~6 max_hr exerc~7
##    <fct>      <dbl> <chr> <fct>     <dbl>   <dbl> <fct>   <fct>    <dbl> <fct>
##  1 0             40 M     ATA         140     289 0       Normal     172 N
##  2 0             54 M     NAP         150     195 0       Normal     122 N
##  3 0             54 M     ATA         110     208 0       Normal     142 N
##  4 1             37 M     ASY         140     207 0       Normal     130 Y
##  5 0             54 F     ATA         120     273 0       Normal     150 N
##  6 0             43 F     TA          100     223 0       Normal     142 N
##  7 0             49 F     ATA         124     201 0       Normal     164 N
```

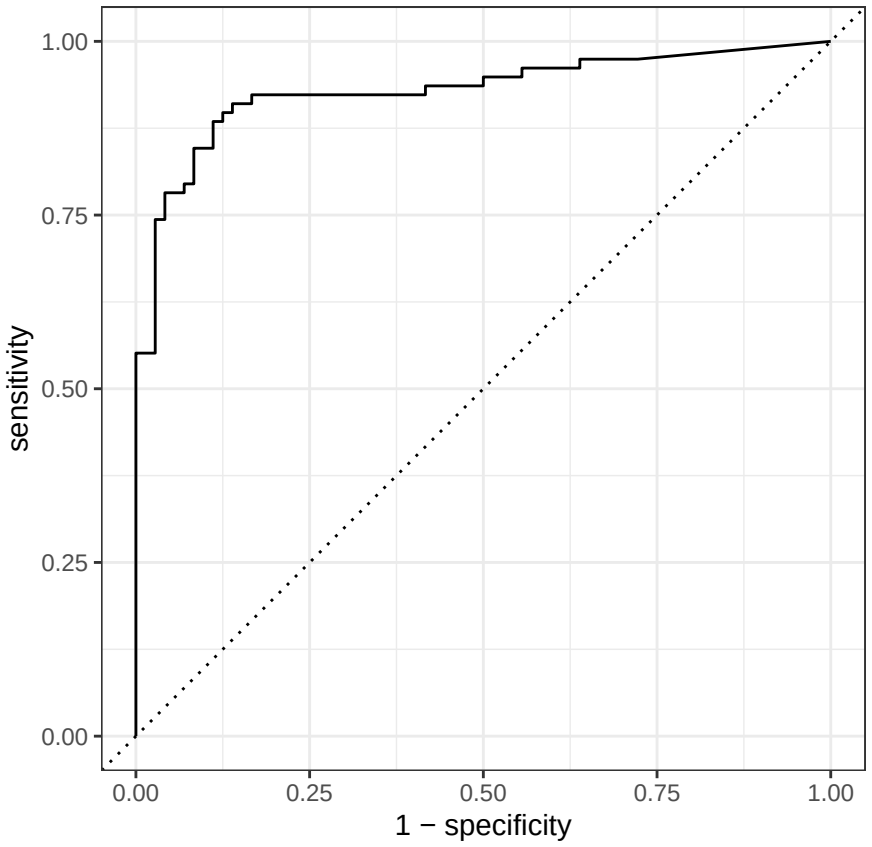
```
##  8 0             52 M     ATA          120     284 0       Normal     118 N
##  9 0             51 M     ATA          125     188 0       Normal     145 N
## 10 0             41 F     ATA          110     250 0       ST         142 N
## # ... with 140 more rows, 3 more variables: oldpeak <dbl>, st_slope <chr>,
## #   heart_disease <fct>, and abbreviated variable names 1: .pred_class,
## #   2: chest_pain_type, 3: resting_bp, 4: cholesterol, 5: fasting_bs,
## #   6: resting_ecg, 7: exercise_angina
```

```
#accuracy on test set for knn
knn_test_acc <- predict(knn_fit, new_data = heart_test, type = "class") %>%
  bind_cols(heart_test %>%
              select(heart_disease)) %>%
  accuracy(truth = heart_disease, estimate = .pred_class)
knn_test_acc
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.887
```

Roc Curve of Test Set Prediction in knn

```
knn_prediction_roc_curve=augment(knn_fit, new_data=heart_test) %>%
  roc_curve(heart_disease, estimate=.pred_0)
autoplot(knn_prediction_roc_curve)
```



```r
auc <- augment(knn_fit, new_data = heart_test) %>%
  roc_auc(heart_disease, estimate = .pred_0) %>%
  select(.estimate)
auc
```

```
## # A tibble: 1 x 1
##   .estimate
##       <dbl>
## 1     0.929
```

5.0 Conclusion:

Through the whole analysis, the best model to predict the heart disease is KNN. The accuracy for the train set is 0.95, the roc-auc is 0.99, and the accuracy for the test set is 0.89, and the roc-auc is 0.93. The performance of all models were fairly well. From the importance variable plot, the most important variables are 'st_slope', 'excercise_angina', 'oldpeak' and the least important one is 'cholestoral'. This is surprising because I assume 'Age' would be an important variable.
The places that can be improved are creating regular grid for tuning if time and my computering power on Mac was enough. Also, the dataset has 916 observations which might be a small dataset after I remove some observations during model cleaning or dealing with NAs. It might be a more professional one if I combine more datasets to one, since heart disease prediction can be a extremely helpful for human beings.