Angiographic Disease Classification

Project Report

Summer 2022

Introduction to Statistical Models and Data Mining

Prof. Miles Chen

August 2, 2022

Team: YYDS

Jingyi Lang

Zhitong Zhou

Olivia Wang

Yanjun Jiang

Introduction

This report presents a process of creating a high performing supervised classification model that uses a given training data set and a test data set on RStudio, which contains 13 predictors showing the values of the patients' physical conditions. The response variable is the diagnosis of heart disease (angiographic disease status) of the patients. Based on the data description[1] and the description in Centers for Disease Control and Prevention[2], the variable "exang," specifically the exercise induced angina (stable angina), may be associated with the diagnosis of heart disease. We aim to fit a classification model that produces the highest mean f-score for the test data set for further learning.

References

[1] *UCLA stat 101C 22 summer - second competition*. Kaggle. (n.d.). Retrieved August 1, 2022, from https://www.kaggle.com/competitions/ucla-stat-101c-22summer-second-competition/data.

[2] Kumar, N. (n.d.). *Advantages and disadvantages of Random Forest algorithm in machine learning*. Advantages and Disadvantages of Random Forest Algorithm in Machine Learning. Retrieved August 1, 2022, from http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-random.html.

[3] Centers for Disease Control and Prevention. (2022, January 20). *Other conditions related to heart disease. Centers for Disease Control and Prevention*. Retrieved August 1, 2022, from https://www.cdc.gov/heartdisease/other_conditions.htm.

[4] XGBoost R Tutorial. https://xgboost.readthedocs.io/en/stable/R-package/xgboostPresentation.html.

[1] Reference [1]
[2] Reference [3]

Exploratory Data Analysis

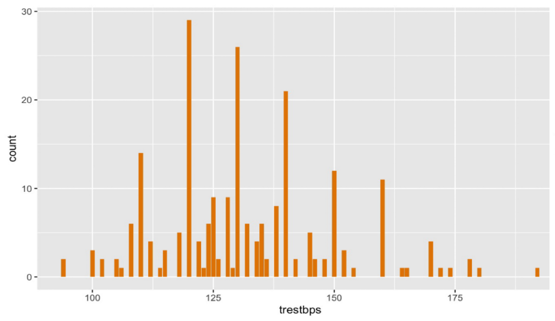
1. Data Cleaning and Factorizing Categorical Variables

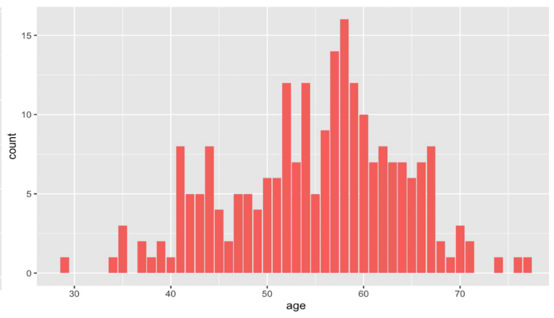
When we load the data, we noticed that some missing values "?" exist in variables *ca* and *thal*. So we first decide to remove the missing values in the data set. However, after we built an initial model and tried to predict the test data set, we found there was also some missing value "?" in variable *ca* in the test data set. Then we tried to change the missing value "?" to another factor and replace the missing value with the most frequent value in the variable. After testing both ideas, the way to set "?" as another factor gives us a higher f_meas.

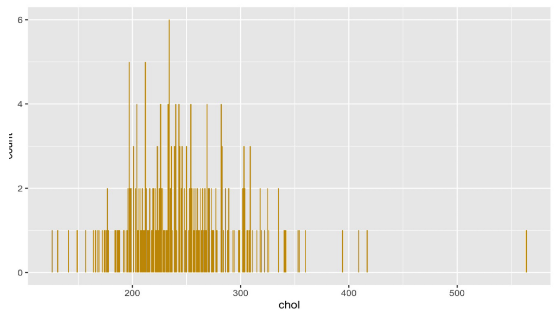
Because the dataset contains both categorical and numeric variables, our next step is to convert all categorical variables into factors in order to ensure our future steps on modeling the data will work on these variables correctly. There are 8 categorical variables and 5 numerical variables in the dataset.

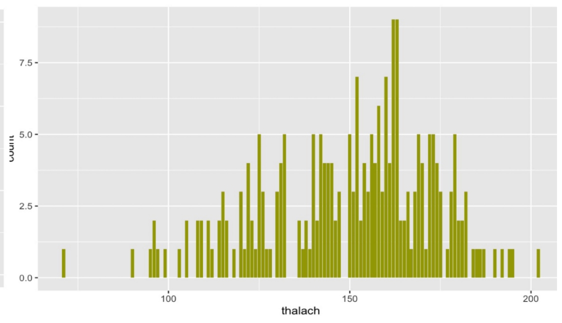
2. Histograms for Numerical Variables

Histograms are useful in showing the distribution of a variable, so we use histograms to visualize the distributions for both categorical and numerical variables. Based on the plots, we noticed that in our numerical variables, the variable *oldpeak,* have a severe skewed distribution while other numerical variables have relatively bell shaped distribution.

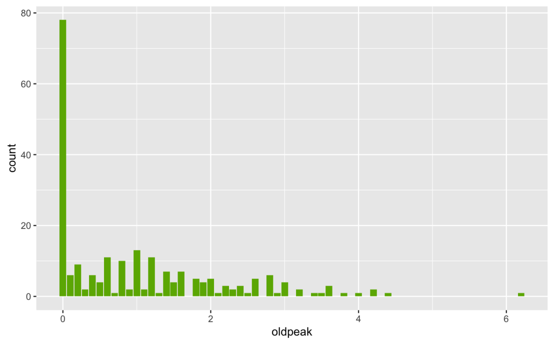


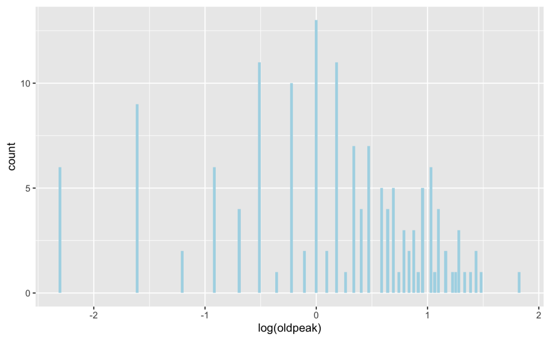






- *old peak: ST depression induced by exercise relative to rest*

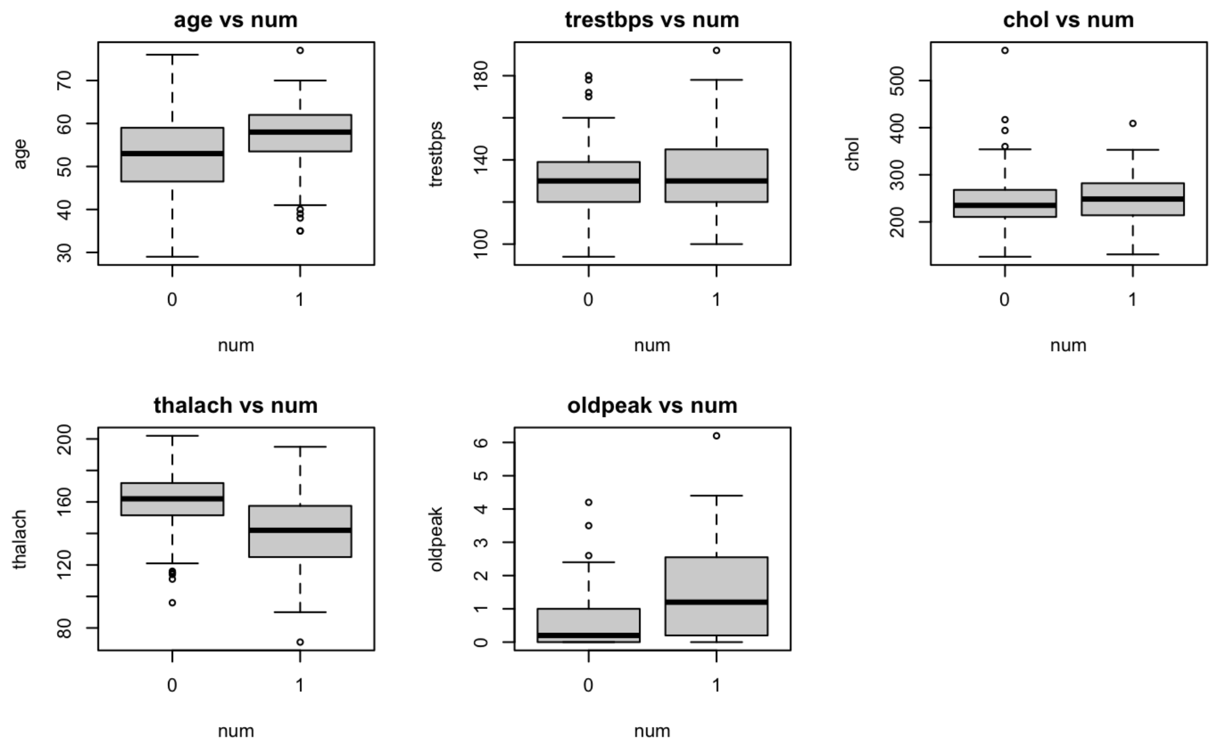




From the green plot above we can see that the distribution of *oldpeak* has a significant skewed pattern, most of the observations are between 0 and 2, only a handful of colleges get more than 2. There is also an extreme value around 6. So we perform log transformation on *oldpeak* to see if the transformation can improve the distribution, the scale after log transformation will represent an increase by one unit resulting in an increase by a *factor* of the log base. The blue plot shows the distribution after the transformation in which the log *(oldpeak)* variable has a better bell-shaped distribution. We will discuss more about log transformation in the recipe section of this report.

3. Box Plots for Numerical Variables

Because our response variable is categorical, we create a boxplot for each numerical variable against the response to examine the distribution of each predictor and the two levels of our response. If the boxplot shows the ranges of the variables are different for each level of the response (num), that indicates the variable has a significant effect on the response.

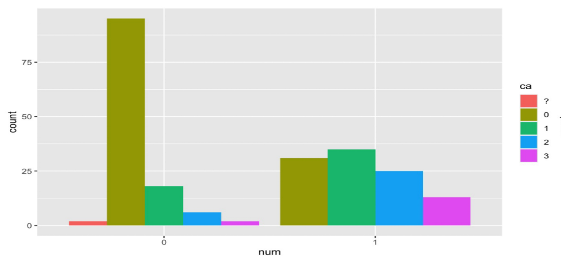


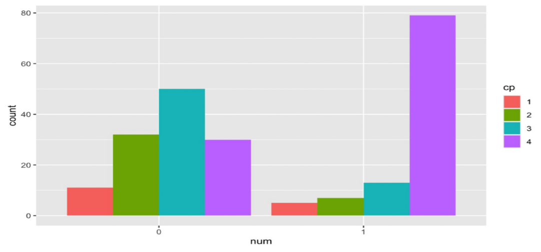
The above graphs age vs. num reveal that the median age of people with > 50% diameter narrowing is higher than the people with < 50% diameter narrowing' age. And the plot thalach vs. num shows median maximum heart rate (*thalach*) is higher for people with >

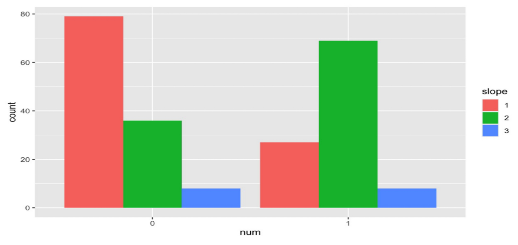
50% diameter narrowing compared to the people with < 50% diameter narrowing. Plot tresbps vs. num indicate The resting blood pressure (in mm Hg on admission to the hospital) *trestbps's* distribution is slightly higher for people with > 50% diameter narrowing. Moreover, ST depression induced by exercise relative to rest (*oldpeak*) in people with > 50% diameter narrowing is also higher. Additionally, the serum cholesterol in mg/dl (*chol*) is also slightly higher in people with > 50% diameter narrowing, but people with < 50% diameter narrowing have more outliers. Overall, the box plot indicates variables age, thalach, and oldpeak have relatively significant effect on the response, variables trestbps and chol will affect the response less.

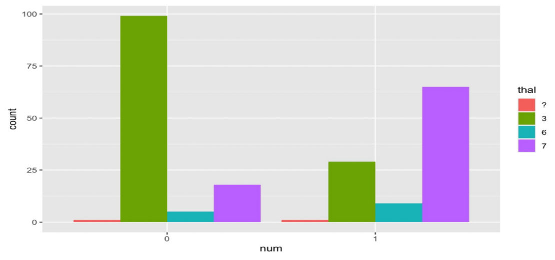
4. Bar Chart for Categorical Variables

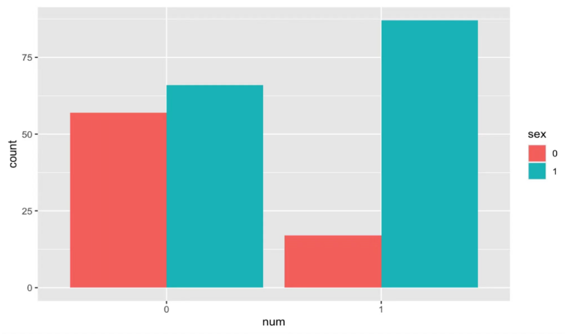
In order to visualize the distributions for our 8 categorical variables against our response variable, we plot bar charts for each categorical variable vs. the two levels in our response. Ideally, we want the frequency of each level by different levels of response to be obvious to indicate that these variables will significantly affect the response.

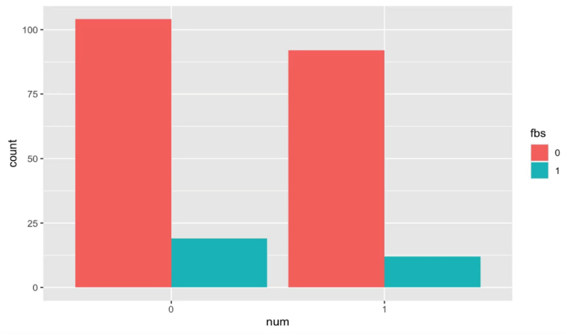


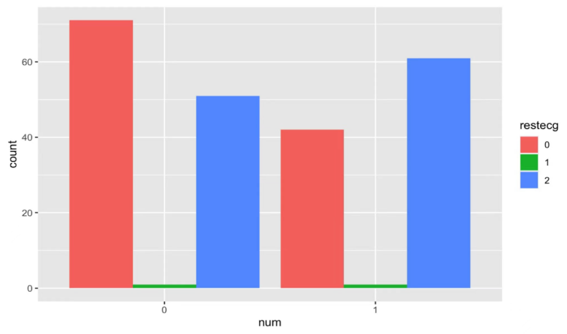


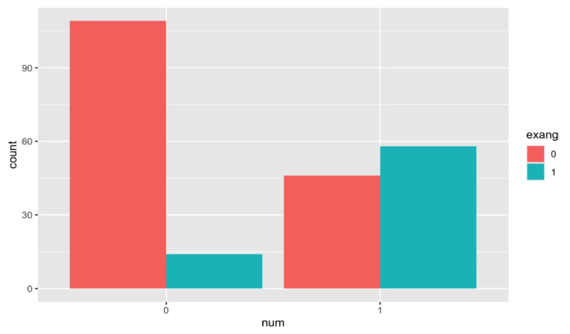












From the plot related to variable *sex* on the top left, we can see there are more males with > 50% diameter narrowing compared to the number of females. Exercise induced angina (*exang*) hardly exists in people with < 50% diameter narrowing but exists more frequently in people with > 50% diameter narrowing. For resting electrocardiographic results (*restecg*), the proportion of it being normal is higher in people with < 50% diameter narrowing and the proportion showing probable or definite left ventricular hypertrophy by Estes' criteria is high in people with > 50% diameter narrowing. Lastly, fasting blood sugar > 120 mg/dl (*fbs*) have similar occurrence rate in both response levels. Through the bar charts, the two levels of the variable *fbs* have similar frequency in both levels of response, therefore, it might have less effect on the response variable. Rest of the variables shows different frequency by levels respect to the levels of response, so we considered variables *sex, cp, restecg, exang, slope, ca,* and *thal* have significant effect on the response.

5. Interactions

After visualization of the variables, we considered finding the significant interactions among the numerical variables. We focused on the two factor interactions only and used a linear regression model to find the significance interactions based on the p-value from the F test. With significant level = 0.05, we found interaction *trestbps:oldpeak* significantly affected the response. We will consider adding the interaction *trestbps:oldpeak* in the model test section to observe whether it will optimize the accuracy of the model.

Recipes

For the recipes we used for our models, since our candidate models are classification models which require recipes, we created four recipes to check if there would be significant differences or improvements in our predicted test results, and then added these recipes to a list as the preprocessor.

1. **Basic recipe** (code *basic_rec*), which used *recipe()* function to create a basic recipe for the models we used. Set up the basic rec with *recipe()* function, *recipe(num ~., data = train_data).*

2. **Logarithmic transformation recipe** (code *log_rec*) , it is based on the basic recipe *recipe(num ~., data = train_data)* then adds *step_dummy(all_nominal_predictors())* and *step_log(oldpeak, , base = 10, signed = TRUE)*. Because there are some variables in the type of factor, use *step_dummy(all_nominal_predictors())* to change these variables to numerical value. Function *step_log(oldpeak, , base = 10, signed = TRUE)* logarithmic transform variable *oldpeak* in the basic recipe. Because based on the exploratory process, we found that the variable *oldpeak* is highly skewed, so we decided to add log transformation in the recipe to check if there would be any improvements for the test results.

3. **Normalized recipe** (code *norm_reci*), it is based on the basic recipe *recipe(num ~., data = train_data)* then adds *step_normalize(all_numeric_predictors()*) to normalize all numeric predictors because we expect to increase coherence of our data and gets rid of a variety of irregularities that can make it more difficult to interpret the data.

4. **Interaction recipe** (code *int_reci*), it is based on the basic recipe *recipe(num ~., data = train_data)* then adds *step_dummy(all_nominal_predictors())* to change the factors or the nominal variables into numeric numbers. Then add another function *step_interact(terms = ~ + trestbps:oldpeak)* to add the interaction *trestbps:oldpeak*.

Candidate Models

In order to avoid the overfitting issue, our team decides to test the performance of each model by using cross-validation on the training data set. The cross-validation is to partition the data into v equally sized blocks, then set one block as the validation set, while the remaining blocks combine to be the training data. We divided the data into 10 cross-validation folds for evaluating our models' performance. The table below shows the 6 candidate models we decided to test.

Model Identifier	**Type of Model**	**Engine**	**Recipe Used**	**Hyperparameters**
lda_model	Linear Discriminant Analysis	MASS	basic_rec log_rec norm_rec	NA
logi_model_tuning	Logistic Regression	glmnet	basic_rec log_rec norm_rec	● penalty = tune() ● mixture = 0
rf_model_tuning	Random Forest	ranger	basic_rec log_rec norm_rec	● mtry = tune() ● Trees = tune() ● min_n = tune()

Model Identifier	**Type of Model**	**Engine**	**Recipe Used**	**Hyperparameters**
knn_model_tuning	K-nearest Neighbors	kknn	basic_rec log_rec norm_rec	● neighbors = tune()
svm_model_tuning	Support Vector Machine	kernlab	basic_rec log_rec norm_rec	● trees = tune() ● tree_depth = tune() ● min_n = tune() ● loss_reduction = tune() ● sample_size = tune() ● mtry = tune() ● learn_rate = tune()
xgb_model_tuning	XGBoost	xgboost	basic_rec log_rec norm_rec	● trees = tune() ● tree_depth = tune() ● min_n = tune() ● loss_reduction = tune() ● sample_size = tune() ● mtry = tune() ● learn_rate = tune()

- **Model:** ***lda_model***

  We performed a Linear Discriminant Analysis (LDA) model with hyperparameters involved. Then, we chose "MASS" as the engine (*set_engine("MASS")*), and "classification" as mode (*set_mode("classification")*) for classification purposes. We performed this lda_model with basic recipe, logarithmic transformation recipe, and normalized recipe respectively to evaluate the model.

- **Model:** ***logi_model_tuning***

  We also attempted a Logistic Regression model to see its performance on the training data set. It applies a logistic function for predictors when modeling.
  We set up the logistic regression model with the *logistic_reg()* function with tuning parameters "penalty" and "mixture", engine as "glmnet" (*set_engine("glmnet")*), and mode as "classification" (*set_mode("classification")*) . And we chose the grid of parameters with *grid_regular(penalty(range(-2, 1)), mixture(), levels = 10)*. Then, we found the best logistic regression model with penalty as 0.3162278, and mixture as 0. After tuning, we updated the logistic regression model with the best parameters, and

called it *logi_model.*

- **Model:** ***rf_model_tuning***

  After we fitted the LDA and Logistic regression models, we also expected to see the performance of the Random Forest model. The random forest classification model is to randomize the variables used in creating trees, each tree built by a different set of variables that select randomly.
  We created a random forest model with the *rand_forest()* function with tuning parameters "*mtry", "min_n", and "trees*", engine as *"ranger"*(*set_engine("ranger")*), and mode as "classification" (*set_mode("classification")*). After adding the model and the recipes to create the workflow, we randomly selected ranges for the "mtry," "trees," and "min_n" parameters *mtry(range(1, 14)), trees(range(20, 100)), min_n(range(1, 10)), levels = 5* in the *grid_regular()*. and resampled the v-folds in the *tune_grid()* to fit the model. Then we found that the best models are the one with basic recipe in the parameters of *mtry = 13, trees = 20, and min_n = 2,* and the one with logarithmic recipe in the parameters of mtry = 10, trees = 85, min_n = 7 . We then updated two different Random Forest models with the two sets of parameters, one to fit the training data with the basic recipe, calling it *rf_model_basic*, and another one to fit the training data with the logarithmic recipe, calling it *rf_model_log*.

- **Model:** ***knn_model_tuning***

  We also attempted a knn model to see the performance of the model on this training data set. The principle of K nearest-neighbor(KNN) classification is given a training data set and looks for a new input instance, and finds the k instances closer to the instance in the training data set. If most of these k instances belong to a certain class, then the sample will be assigned to this class. We set up the knn model with *nearest_neighbor()* function by tuning the parameter "neighbors", set the engine as "kknn" (*set_engine("kknn")*), and mode as "classification" (*set_mode("classification")*). And we chose the grid of knn with *grid_regular(neighbors(range(1, 100)), levels = 10)*. We then found the best neighbor parameter is 23. Lastly, update the knn model with *"neighbors = 23",* and call it *knn_model.*

- **Model: *svm_model_tuning***

  After tuning the KNN model, we attempted the Support-vector Machine model (svm). We set up the model with the *svm_rbd()* function, with *cost = tune()*, and *rbd_sigma = tune()*. Set the engine as "kernlab" (*set_engine("kernlab")*), and set the mode as "classification" (*set_mode("classification")*). Then we chose *level = 10* in the *grid_regular()* function. After piping into a workflow, fitting the model with different recipes, we found the best parameters *cost = 3.174802*, and *rbf_sigma = 0.0004641589.* Lastly, update the *svm_model_tuning* with the best parameters, and call it *svm_model*.

- **Model: *xgb_model_tuning***

  Compared to random forest classification, XGBoost[3] is an implementation of gradient boosting trees algorithm with high predictive accuracy. We set up the model with *boost_tree()* function, with *trees = tune(). tree_depth = tune(), min_n = true(), loss_reduction = tune(), sample_size = tune(), mtry = tune(), and learn_rate = tune()*. We also set the engine as "xgboost" (*set_engine("xgboost")*), and set the mode as "classification" (*set_mode("classification")*). In the function *grid_latin_hypercube()*, we chose the size as 30. After piping into a workflow and fitting the model with different recipes, we found the best parameters *mtry = 3, trees = 1270, min_n = 8, tree_depth = 6, learn_rate = 2.391532e-08, loss_reduction = 3.521858e-09, sample_size = 0.7709096.* Lastly, we updated the *xgb_model_tuning* with the best parameters, and called it *xgb_model.*

[3] Reference [4]

Model Evaluation and Tuning

For a better and clearer comparison among the different candidate models, we used the same **10 v-folds** generated by the cross-validation to measure the performance of the candidate models.

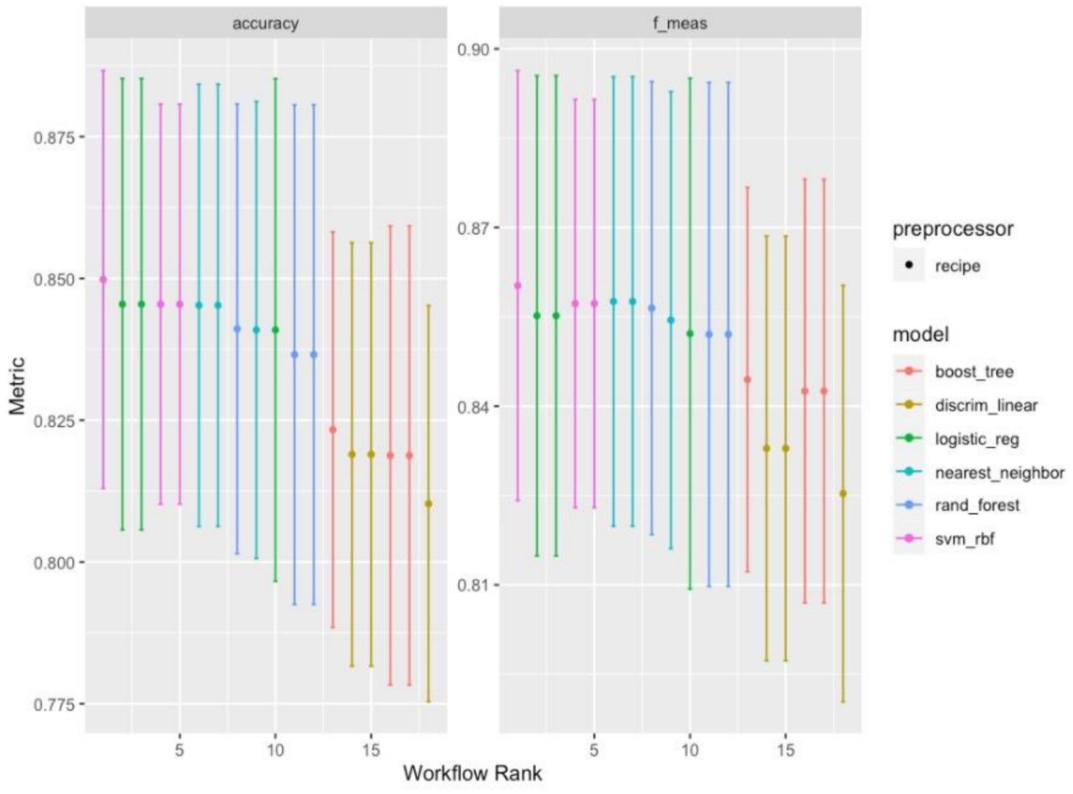
After turning the candidate models' hyperparameters respectively, we picked the parameter combination which generated the highest f score for each model. Our goal is to find the model with the highest f score (*f_meas*), because the higher the f score, the higher the precision of the prediction is. Below are the numeric and visualized performance of all classification models after turning and picking the best combination of hyperparameters.

- **Summary Table of the Performance of Each Model**

Model Identifier	**Metric Score (F score)**	**Hyperparameters**	**SE of Metric**
lda_model	0.8329163	NA	NA
logi_model	0.8571401	● penalty = 0.3162278 ● mixture = 0	● std_err = 0.02450781
rf_model_basic	0.8525979	● mtry = 13 ● trees = 20 ● min_n = 2 ● mean = 0.8613955 ● n = 10	● std_err = 0.03083329
rf_model_log	0.8564572	● mtry = 10 ● trees = 85 ● min_n = 7	● std_err = 0.02312475
knn_model	0.8575869	● neighbors = 23	● std_err = 0.02293967

svm_model_tuning	0.8602447	• cost = 3.174802 • rbf_sigma = 0.0004641589	• std_err = 0.02084176
xgb_model_tuning	0.8481258	• trees = 1270 • tree_depth = 6 • min_n = 8 • loss_reduction = 3.521858e-09 • sample_size = 0.7709096 • mtry = 3 • learn_rate = 2.391532e-08	• std_err = 0.02067101

- **Autoplot of All Models**



According to the autoplot of all model performance, *discrim_linear* shows the performance of the lda model (mean f score around 0.83). The *nearest-neighbor* represents the performance of KNN classification model (mean f score around 0.855). The *logistic_reg* shows the performance of the logistic regression model (mean f score around 0.855). The *rand_forest* represents the performance of the random forest classification model (mean f score around 0.855). The *boost_tree* represents the performance of the XGBoost clasification model (mean f score around 0.84). The *svm_rbf* represents the performance of the svm model (mean f score around 0.86). After we used the models to predict the test data and uploaded it onto Kaggle to check the mean f scores, we found that the random forest classification model with the basic recipe and log recipe produced the highest 2 mean f scores (0.85 and 0.90). Since our target is to find the model without overfitting issues and generate the highest f score, we decided to use the **random forest model** as our final model.

Final Model: Random Forest Model with 13 Predictors

- **Final Model:**
    - rf_model_basic <- rand_forest(trees = 20, mtry = 13, min_n = 2) %>% set_engine("ranger") %>% set_mode("classification")
    - rf_model_log <- rand_forest(mtry = 10, trees = 85, min_n = 7) %>% set_engine("ranger") %>% set_mode("classification")

The final model we used is a random forest classification model with all 13 variables. Random forest is one of the most popular algorithms for tackling classification problems. There are three hyperparameters that affect its performance: *trees, mtry*, and *min_n*. The parameter *trees* represent the number of trees contained in the model, *mtry* represents the number of variables that will be randomly assigned at each split, *min_n* represents the minimum number of data points in a node.

The first final model we decided to use is setting the hyperparameters as *trees = 20, mtry = 13*, and *min_n = 2*, with the basic recipe (*rf_model_basi*c). The second model we decide to use is the model with the hyperparameters *mtry = 10, trees = 85, min_n = 7* and the log recipe (*rf_model_log*). We did not choose the models that have a higher f score than these two random forest models do because of the possible overfitting issue.

- **Strength**[4]
  1. Random forest classifiers could perform an accuracy on the prediction.
  2. Random forest classifiers could handle large amounts of variables and data with a perfect balancing algorithm when some classes are more infrequent than others.
  3. Random forest could prevent the overfitting issue because the algorithm uses multiple decision trees by using random variables.
  4. Random forest classifier works well on both categorical and numerical variables.

- **Weakness**
  1. Random forest classifier models generally take more time to run.
  2. Compared with other classification models like KNN and SVM, random forest is more complex because there are a large number of trees in the model.
  3. As the Kaggle competition posts the f mean score of all testing data out, the results tell us that SVM performs better on predicting the whole testing set.

- **Possible improvements**
  1. **Try a different set of variables.** Due to the time limit, we do not have enough time to run enough combinations of variables. There should be a combination of variables that can be used to generate a better model.
  2. We can also **try more different recipes** for fitting a better model.
  3. Trying some different models that we didn't learn or having errors while fitting this set of data.

[4] Reference [2]

4. For the additional data, our team considers that the patients' eating habits, such as vegetarian or not, their sporting time per day, and their family disease history might be helpful for a more accurate prediction.

Appendix: Final Annotated Script

#model 1:

~~~
```{r}
# set library

library(readr)
library(tidymodels)
library(tidyverse)

# let train be training data set, test be test data set
train <- read_csv("heart_train.csv")
test <- read_csv("heart_test.csv")


~~~

```{r}
# drop nas if there have any na value
train <- train %>% drop_na()
# let train_y be num
train_y <- train$num
train_id <- train$id
train_x <- train[, -c(1, 15)]
```

```{r}
# set train_data
train_data <- data.frame(train_y, train_x)
```

~~~~
```



```{r}
library(dplyr)
# change catigorocal variables be factors as well as "?"
colNames <- c("train_y","sex", "cp", "fbs", "restecg", "exang", "slope", "ca",
"thal")
train_data <- train_data %>%
       mutate_each_(funs(factor(.)),colNames)

test <- test %>%
       mutate_each_(funs(factor(.)),colNames)

```



```{r}
# after tunning the parameters, I find when trees = 20, mtry = 13, min_n = 2
produced the largest f_means score

basic_rec <- recipe(train_y ~., data = train_data)

set.seed(10)
# identify the random forest classificantion model
rf_predict_model <- rand_forest( trees = 20,
                                mtry = 13,
                                min_n = 2) %>%
  set_engine("ranger") %>%
  set_mode("classification")
~~~~

~~~~
# fit model with basic recipe
rf_wflow <-
 workflow() %>%
 add_recipe(basic_rec) %>%
 add_model(rf_predict_model)

rf_res <- rf_wflow %>%
  fit(train_data)



pred_1 <- test$id %>%
  bind_cols(predict(rf_res, new_data = test))

# change column names
colnames(pred_1) <- c("Id", "Predicted")

pred_1

write_csv(pred_1, "rf_classi_prediction(20_13_2)")
```


# ```{r}
# test_2<- read.csv("rf_classi_prediction(20_13_2).csv")
# pred_1$Predicted == test_2$Predicted
# ```
~~~~

~~~
# Model 2:

```{r}
# load the training and test data
train <- read_csv("heart_train.csv")
test <- read_csv("heart_test.csv")
```

```{r}
# load needed libraries
library(tidymodels)
library(readr)
library(ranger)
```

```{r}
# Assign names to variables
num <- train$num
train_id <- train$id
train_x <- train[, -c(1, 15)]
train_data <- cbind(num, train_x)

# Switch the categorical variables into factors
cat_cols <- c("sex", "cp", "fbs", "restecg", "exang",
              "slope", "ca", "thal", "num")
train_data <- train_data %>%
  mutate_each_(funs(factor(.)), cat_cols)

test <- test %>%
~~~

~~~
  mutate_each_(funs(factor(.)), cat_cols)
```


```{r}
# Creating basic and logarithmic recipes

log_rec <- recipe(num ~., data = train_data) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_log(oldpeak, base = 10, signed = TRUE)
```


```{r}
set.seed(10)
# random forest model with log_rec and tuned hyper-parameters
rf_model_log <-
  rand_forest(mtry = 10,
              trees = 85,
              min_n = 7) %>%
  set_engine("ranger") %>%
  set_mode("classification")

rf_wflow <-
 workflow() %>%
 add_recipe(log_rec) %>%
 add_model(rf_model_log)

rf_res <- rf_wflow %>%
  fit(train_data)

pred <- test$id %>%
~~~

~~~
  bind_cols(predict(rf_res, new_data = test))
colnames(pred) <- c("Id", "Predicted")
```

```{r}
write_csv(pred, "rf_classi_log(m10_85_mn7)")
```



# run using macbook: all TRUE, however using windows there are 2 two
FALSE value.
# ```{r}
# test_1 <- read.csv("rf_classi_log(m10_85_mn7).csv")
# pred$Predicted == test_1$Predicted
# ```
~~~

Appendix: Team Member Contributions

Jingyi Lang was in charge of writing R codes, and writing and proofreading the final report.

Zhitong Zhou was responsible for writing the final report.

Olivia Wang was in charge of writing R codes, writing and proofreading the final report.

Yanjun Jiang was responsible for writing the final report.