# P8106 Data Science II Final Project Report: Predicting COVID-19 Recovery Time and Identifying Important Risk Factors

Sarah Forrest (sef2183), Yueran Zhang (yz4188) and John Cheng (jc5635)

5/10/2023

## Contents

# Data Description and Data Manipulation

From a source dataset of 10,000 observations (i.e. `recovery.RData`) collected as part of a combined cohort study on recovery time from COVID-19, two random samples of 2,000 participants each were drawn using seeds 2183 and 4188. These two datasets were then merged and duplicate observations were removed, resulting in a final dataset of 3,579 observations and 15 variables: recovery time from COVID-19, which is the outcome of this analysis, and 14 predictors that encompass pre-existing characteristics (e.g. age, gender, BMI, height, weight), medical records (e.g. diabetes history, hypertension, and vaccination status), and lifestyle (e.g. smoking status). The combined dataset was split into training (70%) and test (30%) datasets using the `createDataPartition()` function. The primary analysis uses time to recovery as a continuous outcome, while the secondary analysis uses time to recovery as a binary outcome, comparing patients whose recovery time exceeded 30 days to patients whose recovery time was 30 days or fewer.

Data cleaning steps entailed the removal of the ID variable, as it does not convey useful information, and the conversion of several numeric and integer variables to factor variables; the resulting dataset contains 8 categorical (factor) predictors and 6 continuous predictors. A review of the dataset showed that there were no null or missing values.

# Exploratory Data Analysis (EDA)

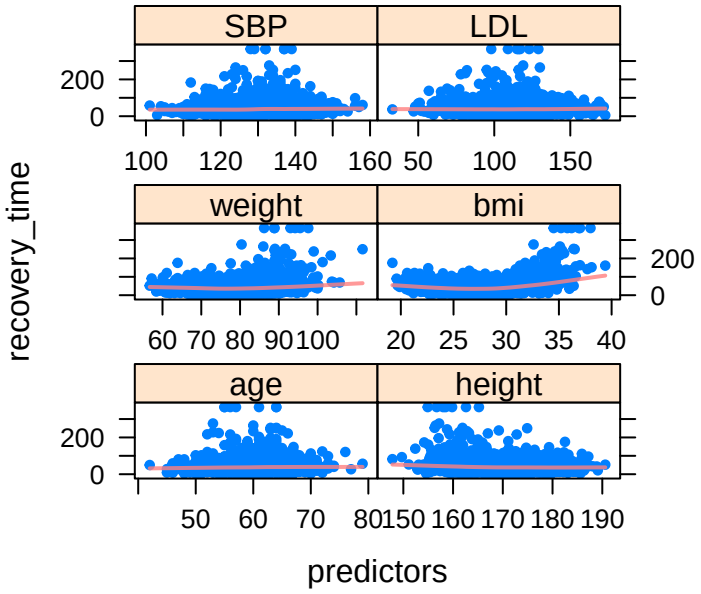
Our EDA includes violin plots to illustrate the distributions of categorical predictors and scatterplots to visualize the distributions of continuous predictors in the dataset, accompanied by a discussion of patterns identified. We additionally examine the distributions of COVID-19 recovery time, both as a continuous variable and as a categorical variable (see Figures 3 and 4 in the Appendix).

### Categorical predictors

From Figure 1 in the Appendix, patients with hypertension appear to have a longer recovery time compared to patients without hypertension. Additionally, vaccinated patients appear to have a shorter recovery time compared to unvaccinated patients.

### Continuous predictors

**Figure 2: COVID-19 recovery times by continuous predictors (`SBP`, `LDL`, `weight`, `bmi`, `age`, `height`)**




Note: `recovery_time` in days; `SBP` in mm/Hg, `LDL` in mg/dL, `weight` in kg, `bmi` in kg/m^2, `age` in years, and `height` in cm.

From Figure 2 above, the following patterns were observed:

- Patients with lower height values appear to have a slightly longer recovery time than patients with higher height values, while patients with lower weight values appear to have a slightly shorter recovery time than patients with higher weight values.
- A slight U-shaped association can be observed between patient BMI and recovery time. Patients with BMI values near the min and max in the dataset have longer recovery times than patients with BMI values in the middle.
- Patient age, SBP, and LDL cholesterol do not appear to have a clear association with COVID-19 recovery time.

**Outcome variable**

Figure 3 in the Appendix shows that when recovery time is treated as a continuous outcome, the mean recovery time from COVID-19 is 43 days and the median is 39 days. Therefore, a right skew is present. The longest recovery time in the dataset was 365 days and the minimum was only 2 days; there are several outliers. Figure 4 in the Appendix shows that when recovery time is treated as a binary outcome variable (with a cutoff of 30 days), approximately a third of participants had a recovery time of 30 days or fewer, while most participants took more than 30 days to recover.

It is important to clarify that since EDA is performed on the combined sample as a whole, we explicitly do not rely on insights from EDA to inform model training, as the models are intended to only see training data and not test data. In the following sections, we will proceed to use cross-validation (CV) to select a final model for predicting time to recovery from COVID-19 as a continuous variable, and CV again to select a final model for predicting time to recovery from COVID-19 as a binary two-level factor variable.

Note: In our EDA, we relabel numerically-coded factor levels as text values to facilitate meaningful interpretation of the visualizations below (see `dat_exp`). However in the model training section, model fitting is performed without the relabeling (see `dat`). `dat_exp` and `dat` are otherwise functionally synonymous datasets.

# 1. Primary Analysis - Time to Recovery as a Continuous Outcome

## Model Training

Several different models were fit to predict time to recovery from COVID-19 as a continuous variable. The `train()` function from the caret package was used to fit each model using either the training dataset or a matrix of all the predictors and a vector of the response variable, `recovery_time`, from the training dataset as inputs. Each model was also fit using CV with 10 folds repeated 5-times. The `trainControl()` function was used to specify this CV method, which was called on within the `train()` function using the trControl argument. Additionally, the method argument was used within the `train()` function to specify the type of model to fit. The resulting model object for each model contains the final model (finalModel) and information about the CV performance. The `predict()` function from the caret package was also used to generate predictions for the test dataset using each final model that was trained using the training dataset. The root mean squared error (RMSE) between the predicted and actual recovery times on the test dataset was calculated in order to evaluate each model's performance and support model comparison.

**1. Linear model** A linear model that assumes a linear relationship between the predictor and response variables (linearity), and assumes that the errors are normally distributed (normality) and have constant variance (homoscedasticity),and that the observations are independent of each other (independence). The linear model is the most basic and assumes a linear relationship between the variables, while the other models allow for more flexible relationships. A method = "lm" argument was used within the `train()` function to specify a linear model fit.

**2. Lasso model** A lasso model is a linear regression model that adds a penalty term to the sum of absolute values of the coefficients to prevent overfitting, which can lead to sparse models by shrinking some coefficients to zero. It has the same assumptions as the linear model. A method = "glmnet" argument was used within the `train()` function to specify a lasso model fit. Additionally, a grid of tuning parameters for the model were set using the tuneGrid argument within the `train()` function. The grid contains 100 values of the lambda parameter, ranging from exp(-1) to exp(5) with equal intervals on the log scale. The alpha parameter is set to 1, indicating that we are only considering Lasso regularization. The `expand.grid()` function is used to generate all possible combinations of alpha and lambda in the grid for tuning.

**3. Elastic net model** An elastic net model is linear regression model that combines the penalties of the lasso and ridge regression methods to prevent overfitting, which can result in better prediction accuracy than either method alone. It has the same assumptions as the linear model. A method = "glmnet" argument was used within the `train()` function to specify a elastic net model fit. Additionally, a grid of tuning parameters for the model were set using the tuneGrid argument within the `train()` function. The grid includes 21 values of the alpha parameter, ranging from 0 to 1 with equal intervals. The lambda parameter is set to a sequence of 50 values, ranging from exp(2) to exp(-2) with equal intervals on the log scale. The `expand.grid()` function generates all possible combinations of alpha and lambda in the grid for tuning.

**4. Principal components regression (PCR) model** A principal components regression (PCR) model is an unsupervised dimension reduction method that entails linear combinations representing the best predictors. PCR aims for non-overlapping information in principal components, and hence uncorrelated principal components are desired. Out of a total $p$ principal components, $m$ are selected (a tuning parameter chosen via CV) – and as m increases, so does the flexibility of the model. Here, we specify the `tuneGrid` parameter in the `train()` function for principal components, as well as use the `preProcess` parameter to standardize predictors. The optimal number of components here is `18`, as shown in the RMSE (CV) plot in Figure 5 in the Appendix.

**5. Partial least squares (PLS) model** A PLS model is a model that seeks to find a low-dimensional representation of the predictor variables that explains the maximum variance in the response variable. It has the same assumptions as the linear model as well as a latent variable assumption, the assumption that the predictor variables are linearly related to the response variable via a set of underlying latent variables. A method = "pls" argument was used within the `train()` function to specify a PLS model fit. Additionally, a tuning parameter for the model were set using the tuneGrid argument within the `train()` function. The

tuneGrid object includes a data frame with a single column ncomp that ranges from 1 to 17, representing the number of components used in the model. The preProcess argument is set to "center" and "scale", which means that the training data will be centered and scaled prior to model fitting.

**6. Generalized additive model (GAM) model** A GAM model is a model that extends the linear model by allowing for nonlinear relationships between the predictor and response variables, using flexible functions called splines. It has the same assumptions as the linear model. A method = "gam" argument was used within the `train()` function to specify a GAM fit.

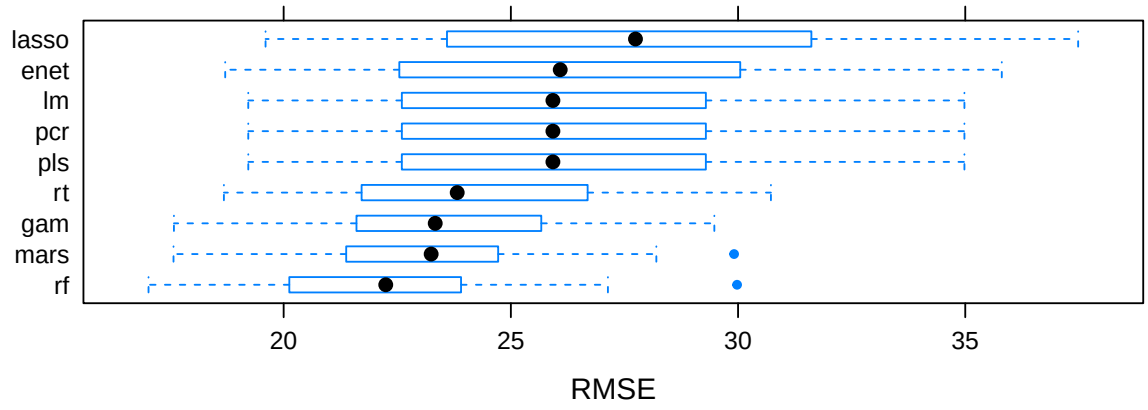
**7. Multivariate adaptive regression spline (MARS) model** A MARS model is a model that uses piecewise linear or nonlinear functions to model the relationship between the predictor and response variables, which can capture complex nonlinear relationships. It has the same assumptions as the linear model. A method = "earth" argument was used within the `train()` function to specify a MARS model fit. Additionally, the `expand.grid()` function is used to generate a grid of tuning parameters. The mars_grid object includes two arguments: degree is set to 1, 2, and 3, representing the number of possible product hinge functions in a single term, and nprune is set to integers between 2 and 17, representing the upper bound on the number of terms in the model. The tuneGrid argument in the `train()` function uses the mars_grid object to specify the parameters for model tuning. Figure 6 in the Appendix shows the tuning parameter selection.

**7. Regression tree model** A regression tree model is a decision tree-based model that partitions the predictor space into a set of rectangular regions, where each region represents a terminal node. The model then fits a simple constant value (mean or median) to the response variable within each region. In addition to the same assumptions as a linear model, regression tree models also assume that the effect of each predictor variable on the target variable is additive. A method = "rpart" argument was used within the `train()` function to specify a regression tree model fit. Figure 7 in the Appendix shows the tuning parameter selection.

**8. Random forest model** A random forest model combines multiple decision trees to improve predictive accuracy and reduce overfitting. It works by fitting multiple decision trees to bootstrap samples of the training dataset and randomly selecting a subset of predictors for each split. The final prediction is the average of the predictions from all the trees. While random forest models do not make explicit assumptions about the distribution or shape of the data, important considerations are whether the observations are independent, whether multicollinearity and/or outliers are present, whether the lasses are roughly balanced, and adequate sample size. A method = "ranger" argument was used within the `train()` function to specify a random forest model fit using the ranger method, which is a fast implementation of the random forest algorithm. The tuneGrid argument in the `train()` function uses the rf_grid object to specify the parameters for model tuning. The object sets up a grid of parameters to search over during the training process, and consists of combinations of `mtry` (the number of variables to consider at each split) from 1 to 15, `splitrule` (the splitting rule used to determine how to split a node) set to "variance", and `min.node.size` (the minimum size of the terminal nodes) from 1 to 6. The importance parameter is set to "impurity", which means that variable importance is measured by calculating the mean decrease in impurity. This is a measure of how much each variable contributes to reducing the overall impurity (or variance) in the model. Figure 8 in the Appendix shows the tuning parameter selection.

**Model comparison**

**Figure 9: Model Comparison Plot Using RMSE**

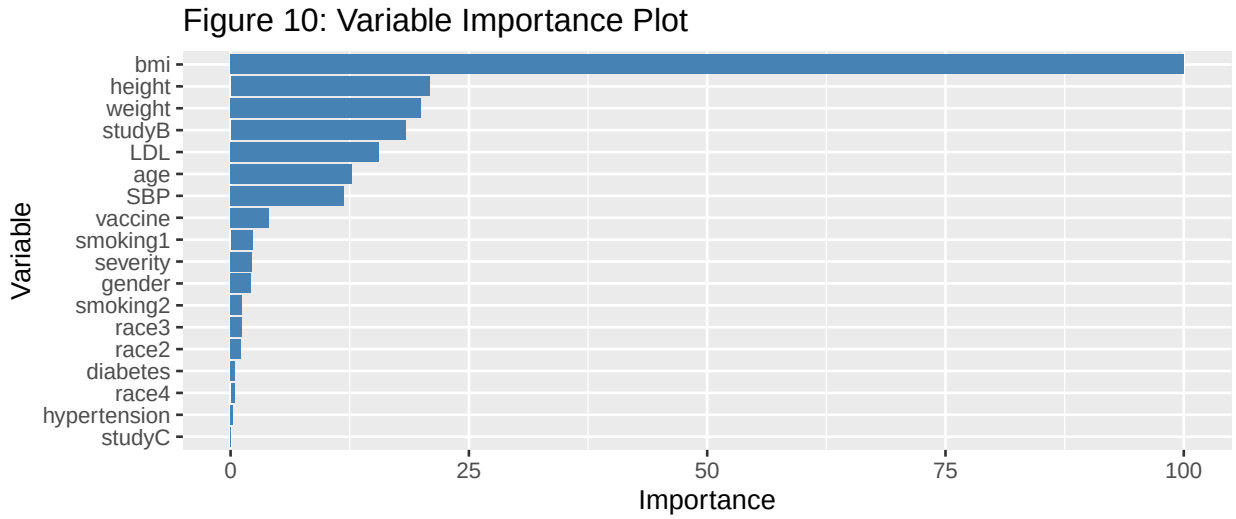



The models were assessed by comparing the RMSE for each model. Low RMSE indicates the best performing model (low prediction error), while high RMSE indicates the worst performing model (high prediction error). The final model was selected by comparing the median training RMSE for all models. The lasso model had the highest median training RMSE (i.e., worst performance), and the random forest model had the lowest median training RMSE (i.e., best performance). Therefore, the random forest model was selected as the final model in this study.

## Results

The final random forest model predicts the time to recovery from COVID-19 based on the predictor variables. The optimal model was determined using CV with 10-fold and 5 repetitions and it's performance was assessed using RMSE. The model's performance improved with a higher value of `mtry` and a lower value of `min.node.size`. The optimal model had an `mtry` value of 6 and a `min.node.size` value of 1. This resulted in a model with 11 trees, where each split randomly selected six predictor variables, and the RMSE value was 22.47, which is lower than the average RMSE value of 24.22 across all the tested models. Therefore, this model has better performance than the other models.

**Model interpretation:**




The `varImpPlot()` function was used to display the variable importance scores for each predictor in predicting the outcome of the model. The BMI variable has the highest importance score of 100, meaning it has the greatest impact on the outcome, recovery time from COVID-19. Height and weight have importance scores of around 20, indicating that they are also important predictors. Other predictors such as LDL cholesterol (LDL), age, systolic blood pressure (SBP), and study B (i.e., being in study B) also have significant importance scores (imp $>$ 10), indicating that they contribute to the predictive power of the model. Finally, the variables for Black (race3), Asian (race2), and Hispanic (race4) races, as well as diabetes status, hypertension status, and study C (i.e., being in study C) have the lowest importance scores (imp $<$ 2), indicating that they have the least impact on recovery time.

**Model performance:** The model's training error (training RMSE) of 10.04 indicates that, on average, the model's predictions on the training data deviate from the actual values by 10.04 units. Meanwhile, the test error (test RMSE) of 21.86 suggests that the model's performance on unseen data is worse than its performance on the training data.

### Conclusion

After evaluating eight different models, the random forest model was identified as the best predictor for recovery time from COVID-19 based on the RMSE metric. The model's variable importance scores revealed that BMI, height, weight, LDL cholesterol, age, and systolic blood pressure were the most critical predictors for predicting recovery time, while variables such as gender, race, diabetes status, hypertension status, and smoking status were found to be less important. The model performs well on the training data, however its performance on unseen data is relatively poorer. Overall, the random forest model provides valuable insights into the most important predictors of recovery time from COVID-19, which can be useful in developing targeted interventions for individuals at high risk of prolonged recovery.

## 2. Secondary Analysis on Time to Recovery as a Binary Outcome (>30 days vs. <= 30 days)

### Model Training

Four different models were fit to predict time to recovery from COVID-19 as a binary variable (>30 days vs. <= 30 days). The `train()` function from the caret package was used to fit each model using either the

training dataset. Each model was also trained using CV. The `trainControl()` function was used to create the `ctrl_two_class object` with the method argument set to "cv" to indicate that k-fold CV will be used. Additionally, the summaryFunction argument was set to "twoClassSummary" to calculate metrics such as sensitivity, specificity, and area under the ROC curve to evaluate the performance of the models. This object is called using the trControl argument. Additionally, the method argument was used within the `train()` function to specify the type of model to fit. The resulting model object for each model contains the final model (finalModel) and information about the CV performance. The `predict()` function from the caret package was also used to generate predictions for the test dataset using each final model that was trained using the training dataset. The area under the ROC curve (AUC) was calculated in order to evaluate each model's performance and support model comparison.

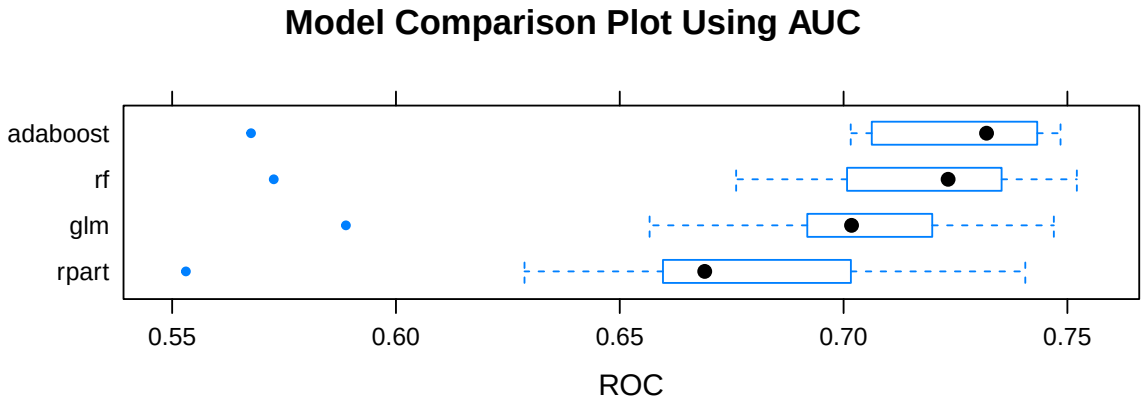
**1. Logistic regression model** Logistic regression is a classification method that applies logit transformation to predict probabilities in [0, 1] – i.e. COVID-19 recovery time as a binary two-level factor variable. Logistic regression assumes linearity of the logit, independence of observations, absence of multicollinearity, the outcome variable is binary, and linearity in the log-odds.The `method` argument was set to `glm` for logistic regression and the `metric` argument defined as `ROC` for resampling results.

**2. Classification tree** Classification trees are particularly useful where the relationship between the response variable and the predictor variables is non-linear or complex, and where the predictor variables have non-monotonic relationships with the response variable. A classification tree assumes binary splits, hierarchical structure, and and absense of multicollinearity. The `method` argument is set to `rpart`, indicating that the model should use the recursive partitioning method for classification. The `tuneGrid` argument specifies a grid of values for the `cp` parameter, which controls the complexity of the model. Figure 11 in the Appendix shows the tuning parameter selection and Figure 12 shows the classification tree using the `rpart.plot()` function.

**3. Random forest** Random forest models combine multiple decision trees to improve predictive accuracy and reduce overfitting, and can be applied for classification analysis. It works by fitting multiple decision trees to bootstrap samples of the training dataset and randomly selecting a subset of predictors for each split. The final prediction is the average of the predictions from all the trees. Random forests share assumptions with decision trees. A `method = ranger` argument was used within the `train()` function to specify fast implementation of the random forest algorithm. The tuneGrid argument in the `train()` function uses the `rf_class_grid` object to specify the parameters for model tuning: combinations of `mtry` (the number of variables to consider at each split) from 1 to 15, `splitrule` (the splitting rule used to determine how to split a node) set to "gini", and `min.node.size` (the minimum size of the terminal nodes) from 1 to 6. Figure 13 in the Appendix shows the tuning parameter selection.

**4. Adaboost** AdaBoost is a boosting method for classification that entails a continual process of weighting the training data, fitting classification trees, and re-weighting. to improve assignment of class labels. The AdaBoost classification model assumes that the training data can be iteratively weighted to improve the performance of the classification trees and the overall model and absence of multidisciplinary. The `gbmA.grid` called in the `train()` function is used to set the following arguments: `n.trees` to 2,000-5,000 (incremented by 1,000), `interaction.depth` to 1-4, `shrinkage` to 0.005-0.02, and `n.minobsinnode` to 1. The `train()` function then calls `gbmA.grid`, as well as a `method` argument specifies the type of model to be trained (`gbm`) and a `distribution` argument for the distribution of the response variable (`adaboost`). Figure 14 in the Appendix shows the tuning parameter selection.

**Model comparison** The classification models are compared on the basis of resampling results (i.e. training AUC for each model). AUC is a measure of the overall performance of a classification model, with values ranging from 0 to 1 (a value of 1 represents a perfect classifier). Therefore, a large AUC is generally considered better for model fit.




The AdaBoost model exhibits the highest mean (0.72) and median (0.73) training AUC among the four fitted classification models (i.e. best performance). Therefore, the AdaBoost model was selected as the final classification model for predicting time to recovery from COVID-19 as a binary outcome.

## Results

The final AdaBoost model was fitted on the training data, comprised of 2,507 randomly selected observations; the optimal model was determined using CV and was trained with 2,000 iterations. The performance of the selected AdaBoost model was subsequently evaluated on test data, consisting of the remaining 1,072 observations. The AdaBoost model exhibits a test AUC of 0.725, as shown in Figure 15 in the Appendix, indicating decent performance on unseen data.

**Model interpretation:**

The selected AdaBoost model included 18 predictors, with 17 predictors having a non-zero influence on the outcome (COVID-19 recovery time as a binary variable). The relative importance of the 18 predictors used in the final model was then assessed; the predictor with the highest relative importance was BMI at 31.2%, followed by study B (i.e., being in study B) at 21.8%, and being vaccinated (vaccine1) at 11.1%. The remaining 14 predictors had lower relative importance.

## Conclusion

After training four different classification models, the AdaBoost was identified as the best model for classifying the outcome variable, COVID-19 recovery time, based on a comparison of the resampling results (i.e. training AUC). The selected model's variable relative influence values revealed that BMI, study group, vaccination status, male gender, and LDL cholesterol were the most influential predictors for classifying COVID-19 recovery time. The model's mean test AUC of 0.73 suggests that it performs reasonably well on the unseen data. Overall, the AdaBoost model provides valuable insights into the most important predictors of recovery time category from COVID-19. However, a limitation of the AdaBoost model is that it's sensitive to outliers. These data points can be given high weights by the algorithm during the training process, which can lead to over fitting and reduced generalization performance.

# Appendix

**Table 1. Data summary**

Table 1: Data summary

| | |
|---|---|
| Name | dat_exp |
| Number of rows | 3579 |
| Number of columns | 15 |
| Column type frequency: | |
| factor | 8 |
| numeric | 7 |
| Group variables | None |

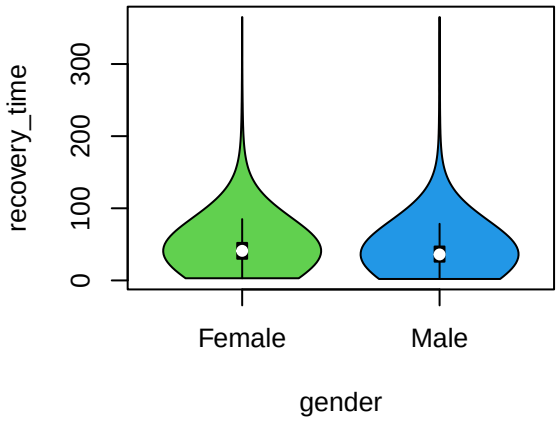
**Variable type: factor**

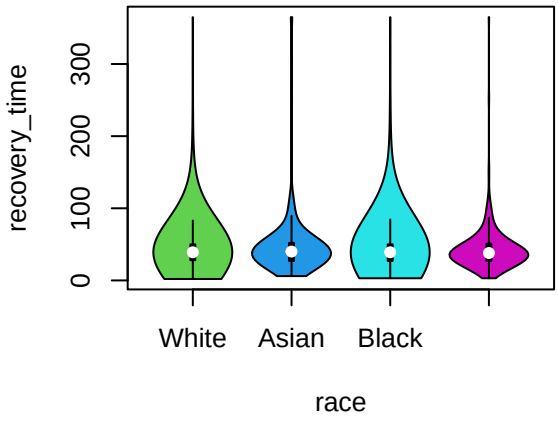
| skim_variable | n_missing | complete_rate | ordered | n_unique | top_counts |
|---|---|---|---|---|---|
| gender | 0 | 1 | FALSE | 2 | Fem: 1855, Mal: 1724 |
| race | 0 | 1 | FALSE | 4 | Whi: 2310, Bla: 734, His: 349, Asi: 186 |
| smoking | 0 | 1 | FALSE | 3 | Nev: 2164, For: 1064, Cur: 351 |
| hypertension | 0 | 1 | FALSE | 2 | No: 1836, Yes: 1743 |
| diabetes | 0 | 1 | FALSE | 2 | No: 3058, Yes: 521 |
| vaccine | 0 | 1 | FALSE | 2 | Vac: 2115, Not: 1464 |
| severity | 0 | 1 | FALSE | 2 | Not: 3248, Sev: 331 |
| study | 0 | 1 | FALSE | 3 | B: 2163, A: 711, C: 705 |

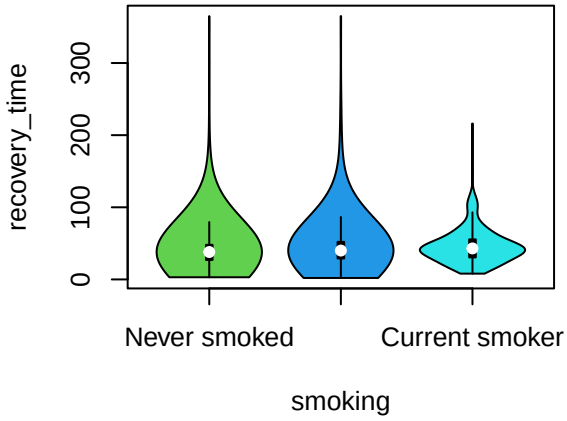
**Variable type: numeric**

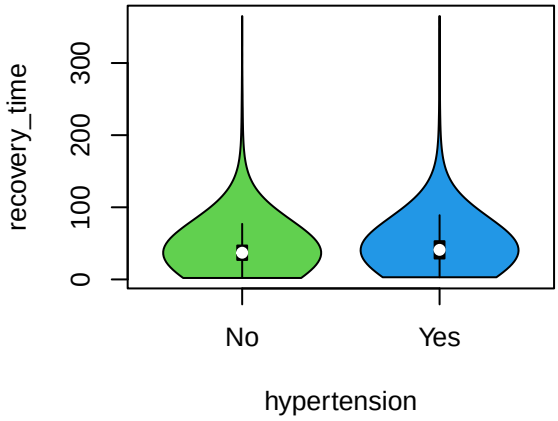
| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
|---|---|---|---|---|---|---|---|---|---|---|
| age | 0 | 1 | 60.08 | 4.45 | 42.0 | 57.0 | 60.0 | 63.0 | 79.0 | |
| height | 0 | 1 | 169.94 | 5.92 | 147.8 | 166.0 | 170.0 | 173.9 | 190.6 | |
| weight | 0 | 1 | 80.00 | 7.14 | 56.7 | 75.3 | 80.0 | 84.8 | 111.6 | |
| bmi | 0 | 1 | 27.76 | 2.77 | 19.2 | 25.8 | 27.7 | 29.5 | 39.4 | |
| SBP | 0 | 1 | 130.27 | 7.93 | 101.0 | 125.0 | 130.0 | 136.0 | 158.0 | |
| LDL | 0 | 1 | 109.91 | 19.56 | 33.0 | 97.0 | 110.0 | 123.0 | 173.0 | |
| recovery_time | 0 | 1 | 43.02 | 29.55 | 2.0 | 28.0 | 39.0 | 50.0 | 365.0 | |

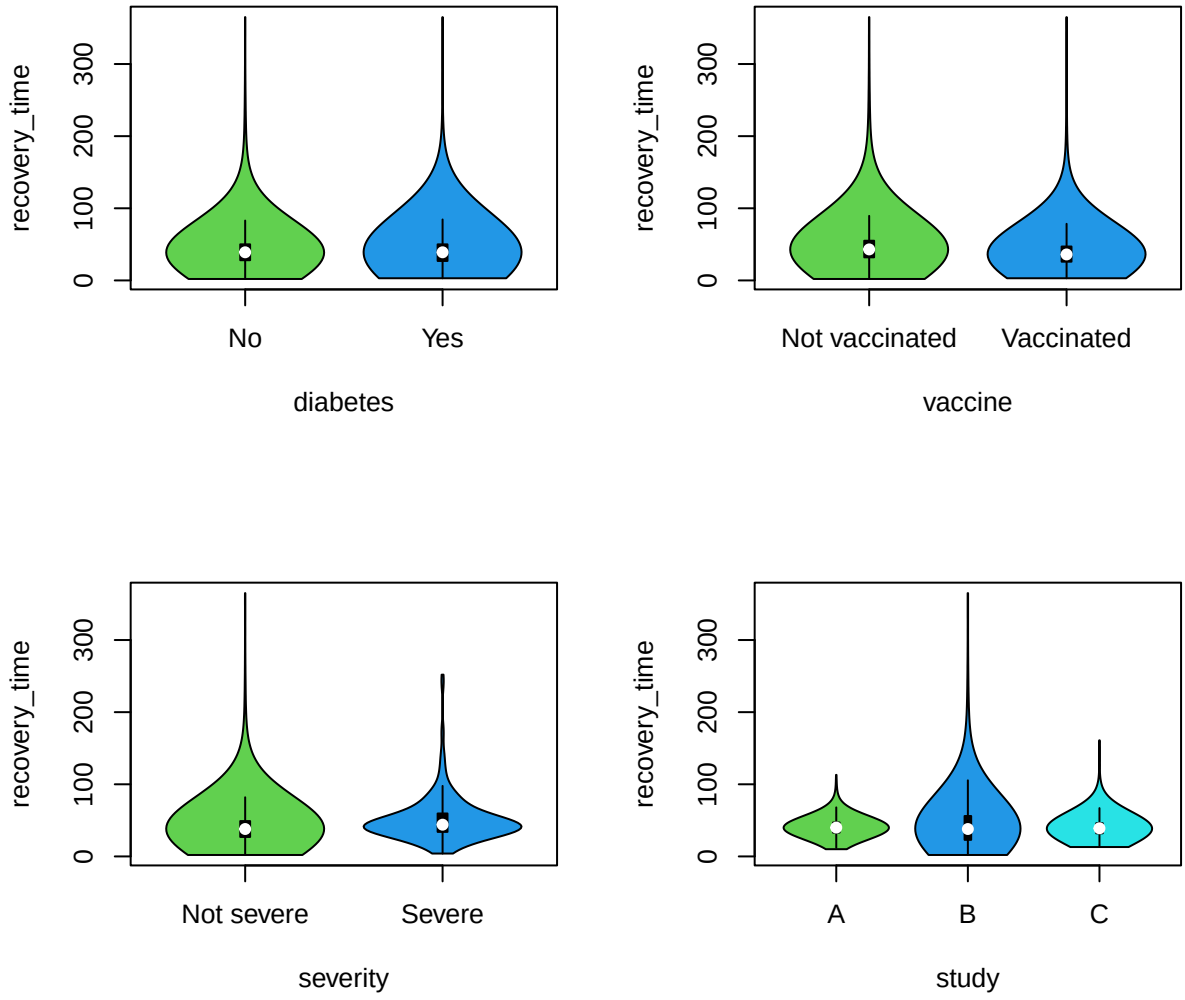
**Figure 1: COVID-19 recovery times by levels of categorical predictors (`gender`, `race`, `smoking`, `hypertension`, `diabetes`, `vaccine`, `severity`, `study`)**


recovery_time
0
100
200
300
Female
Male
gender


recovery_time
0
100
200
300
White
Asian
Black
race


recovery_time
0
100
200
300
Never smoked
Current smoker
smoking


recovery_time
0
100
200
300
No
Yes
hypertension




Note: `recovery_time` in days.

**Figure 3. Recovery time as continuous outcome**

Table 4: Summary of Recovery Time

| Mean | Median | Maximum | Minimum |
|---|---|---|---|
| 43 | 39 | 365 | 2 |

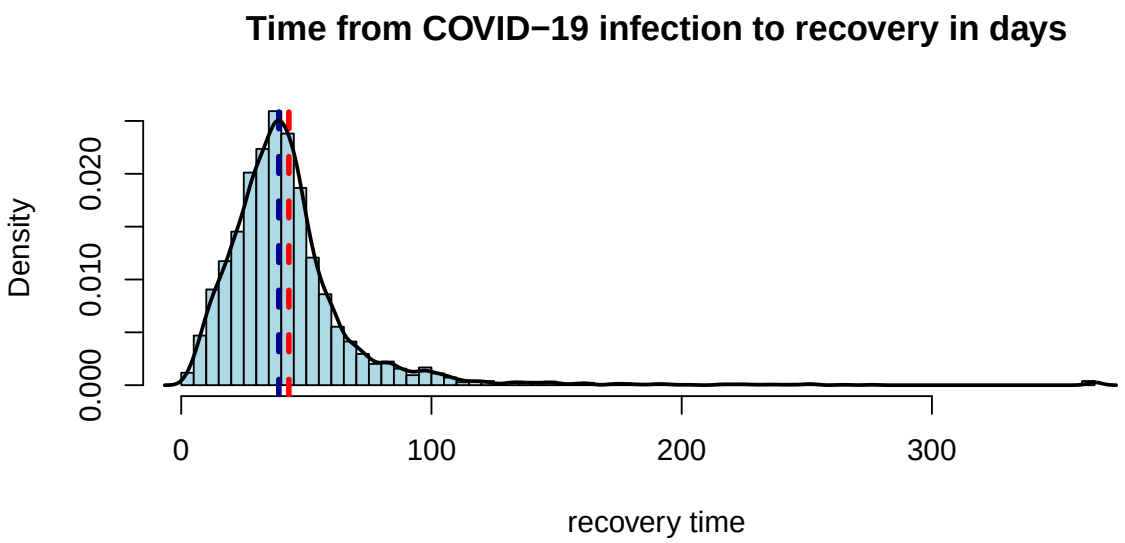
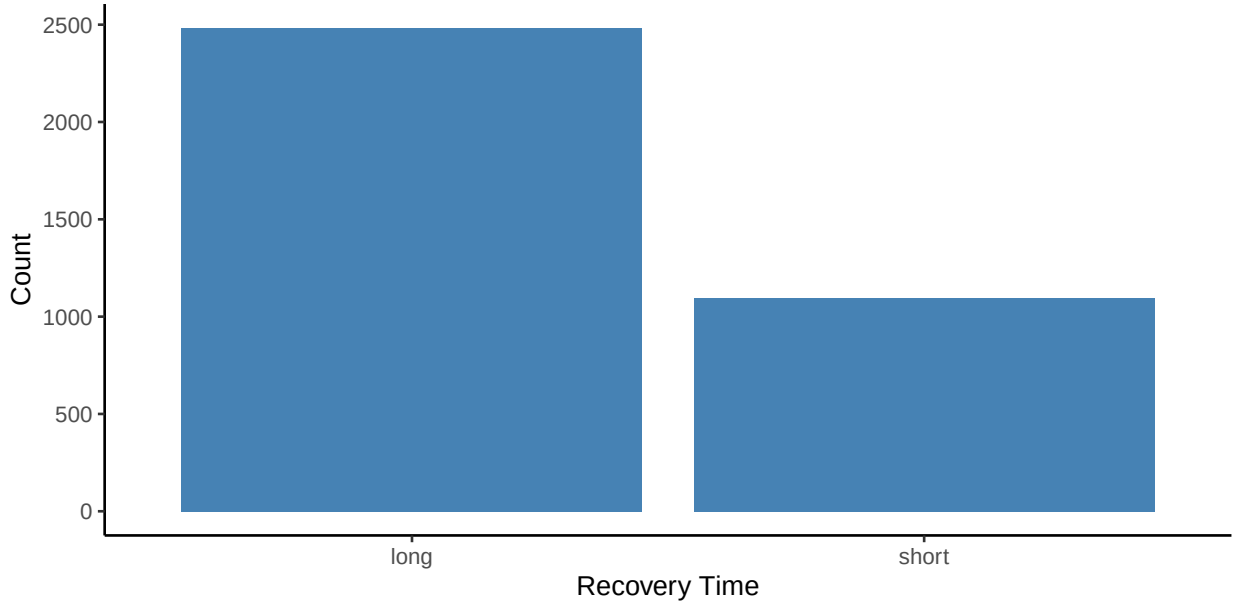



**Figure 4. Recovery time as binary outcome**




## Plots for the primary analysis

**Figure 5. CV plot for PCR model**

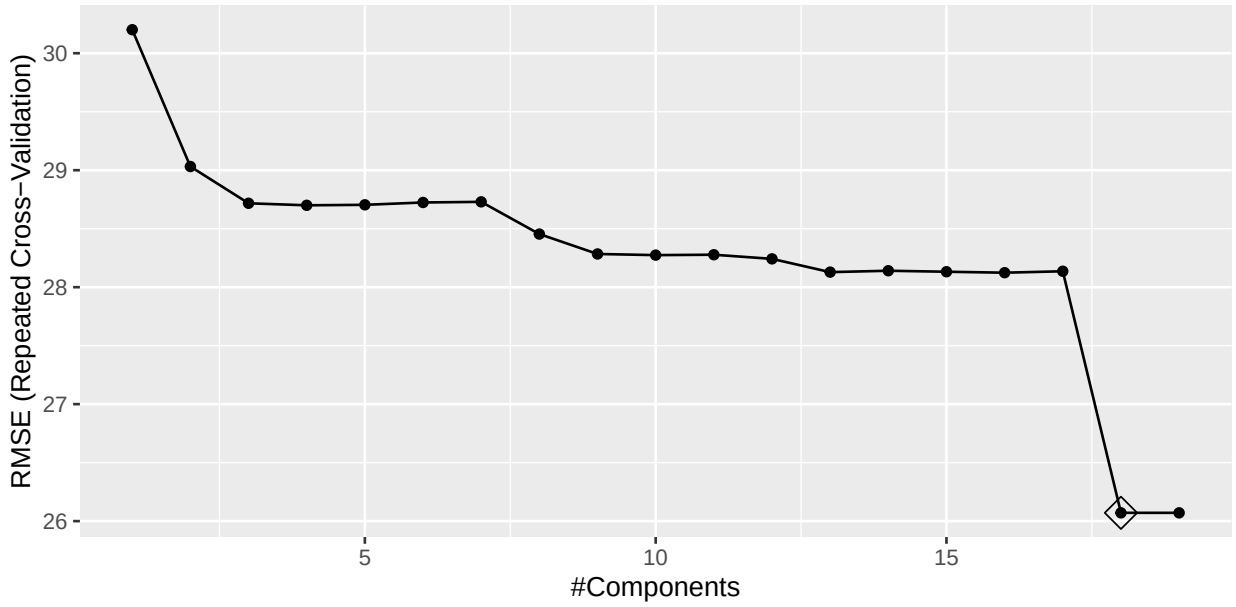



**Figure 6. CV plot for MARS model**

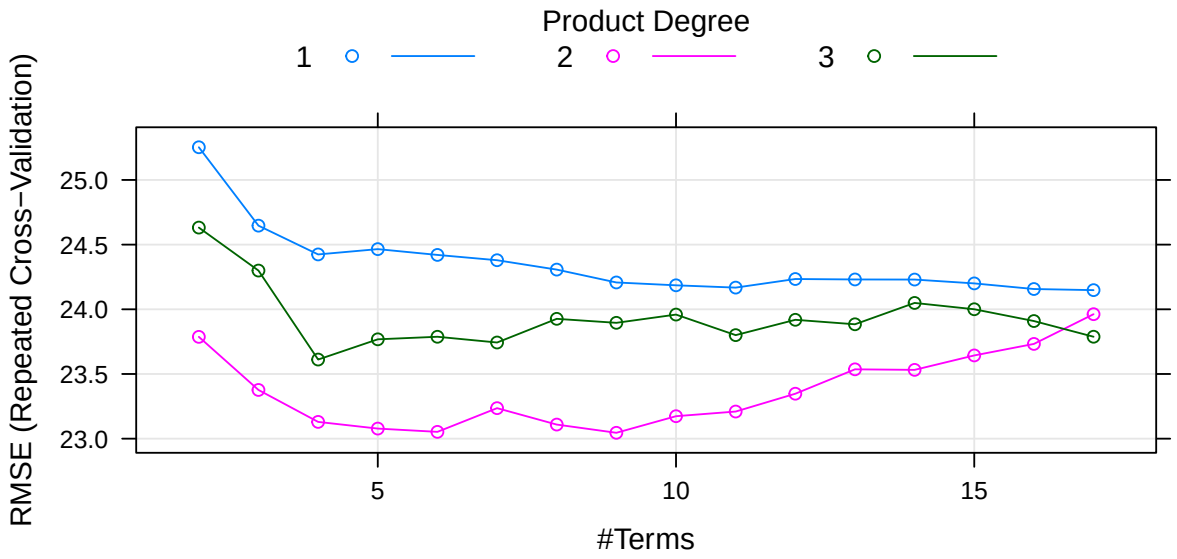



**Figure 7. CV plot for regression tree model**

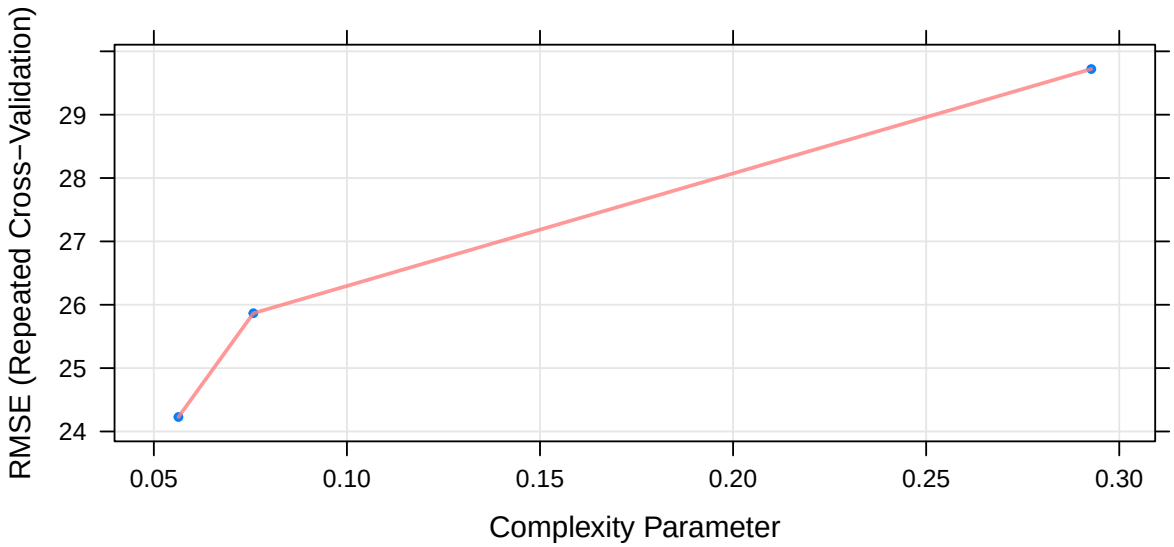
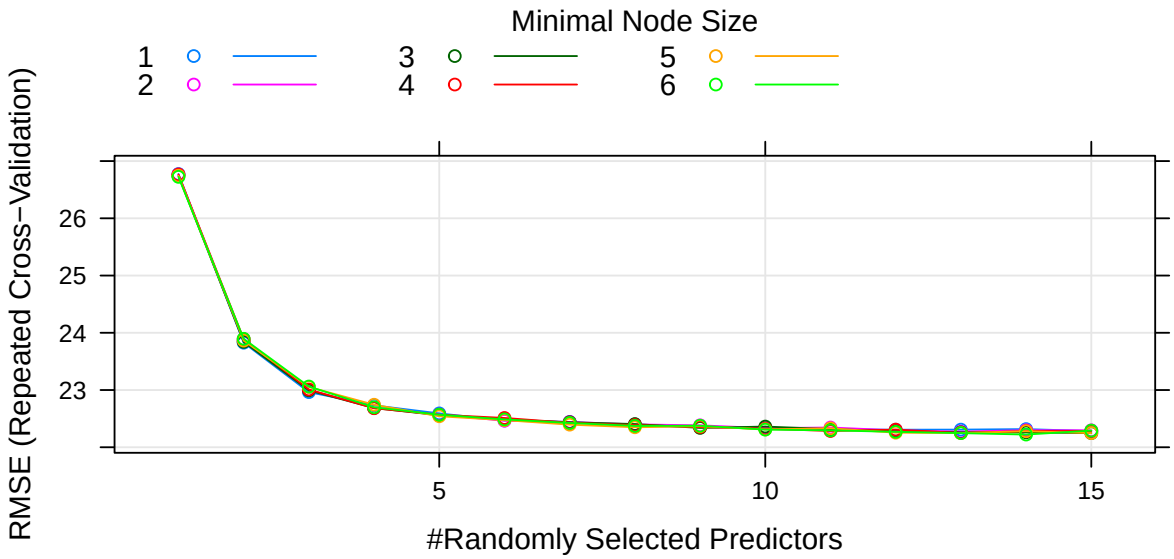



**Figure 8. CV plot for random forest model - primary analysis**




## Plots for the secondary analysis

**Figure 11. CV plot for classification tree model**

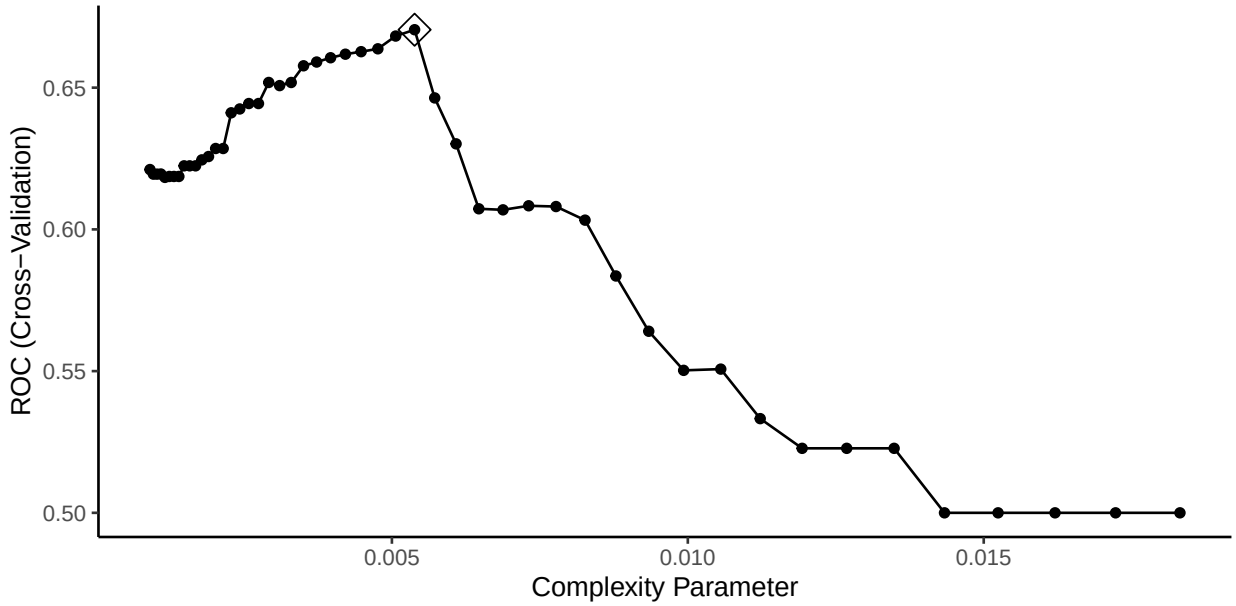



**Figure 12. Classification tree for classification tree model**

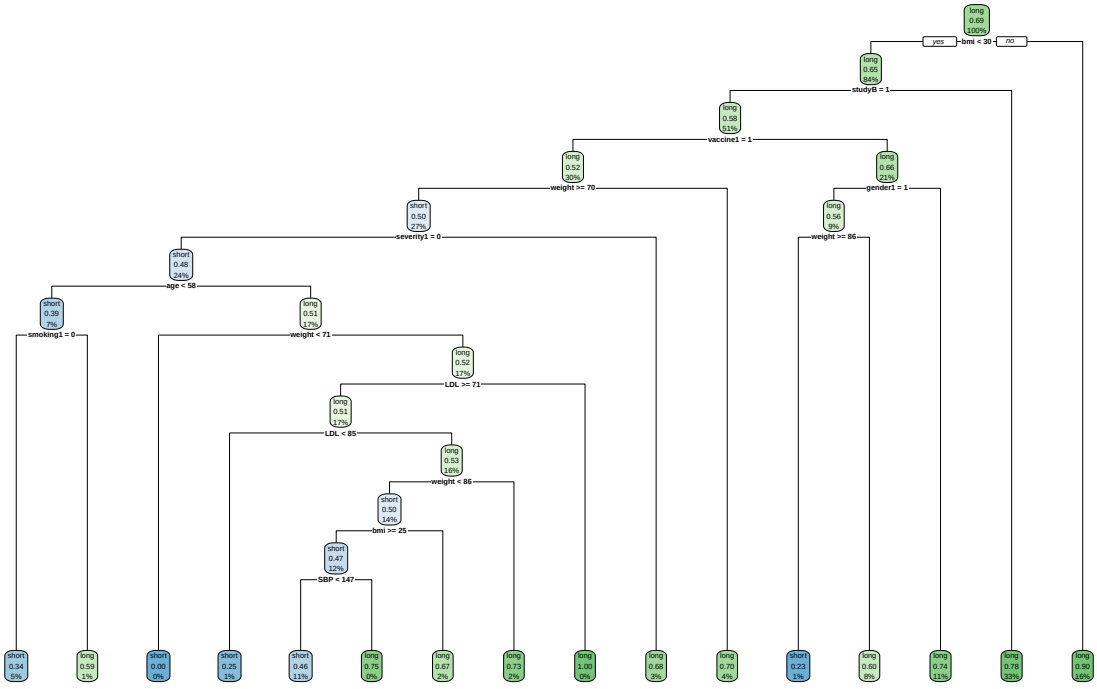


**Figure 13. CV plot for random forest model - secondary analysis**

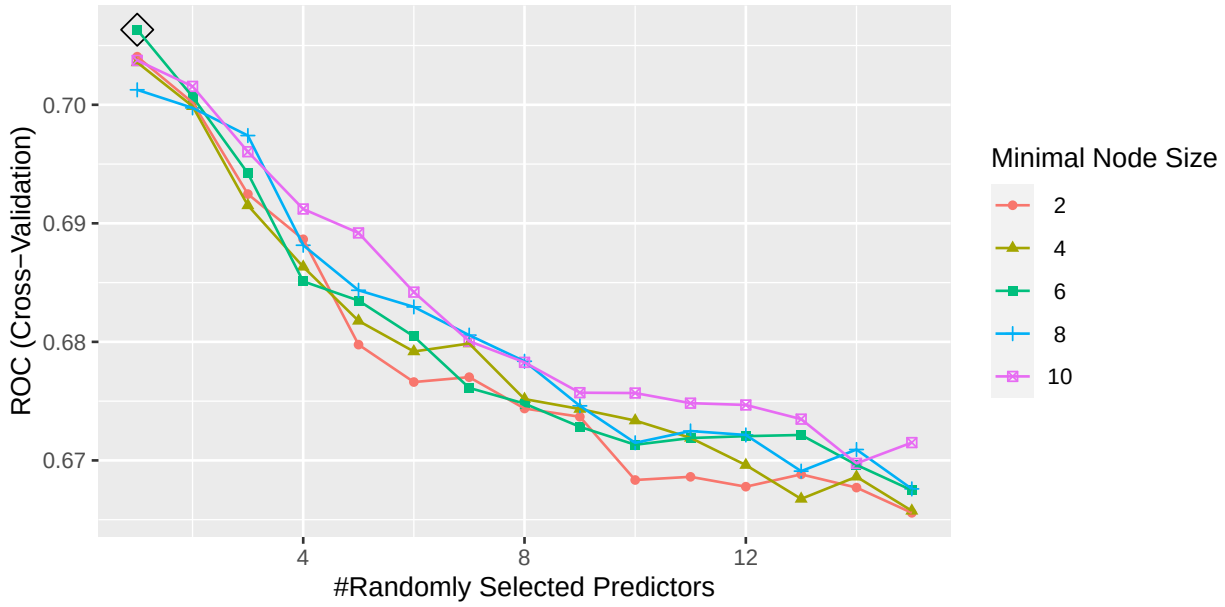



**Figure 14. CV plot for AdaBoost model**

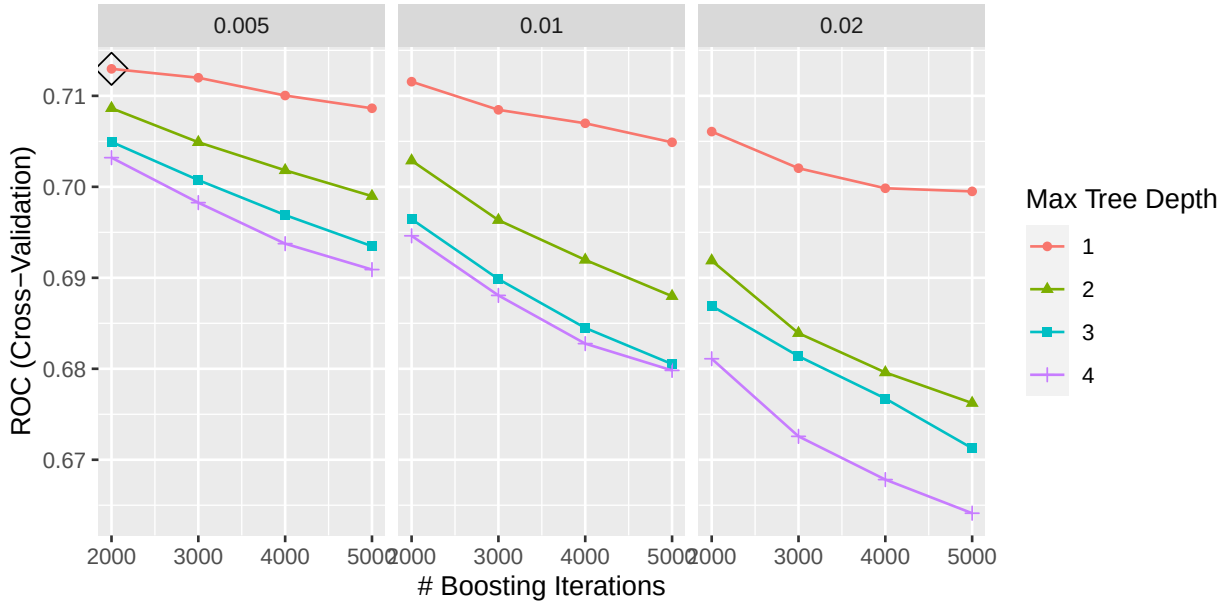
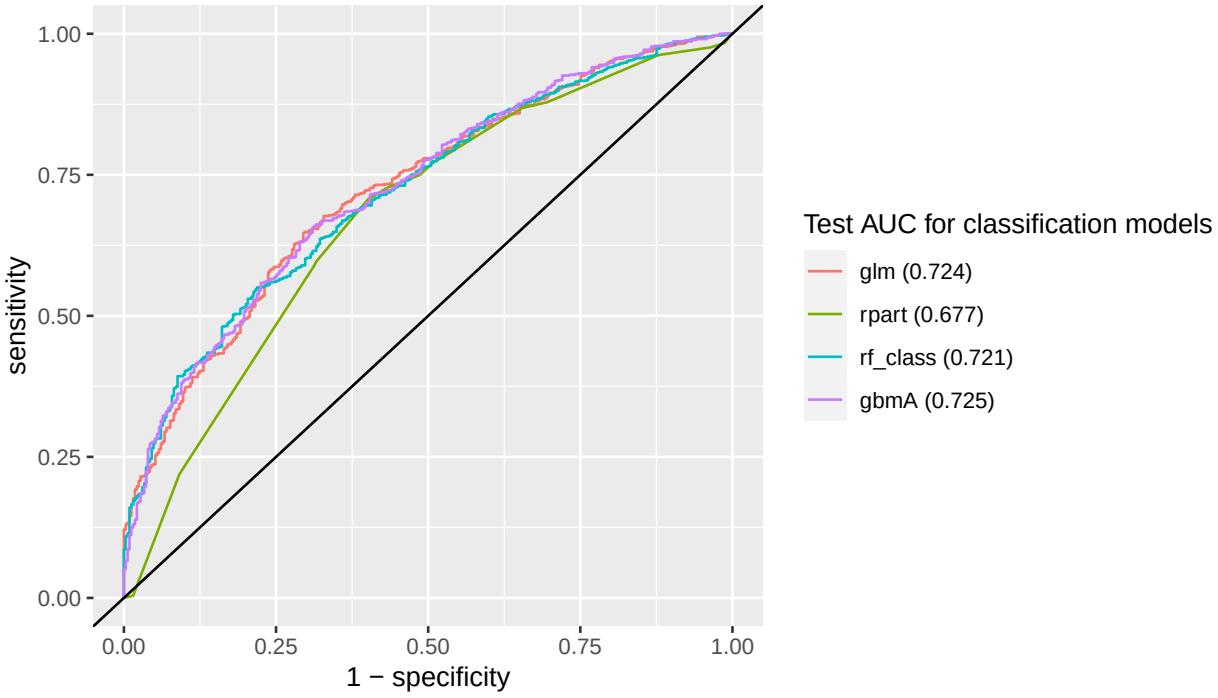



**Figure 15. Test AUC for classification models**


1.00
0.75
0.50
0.25
0.00
sensitivity
0.00
0.25
0.50
0.75
1.00
1 − specificity
Test AUC for classification models
glm (0.724)
rpart (0.677)
rf_class (0.721)
gbmA (0.725)