A Statistical Analysis of Heart Failure Data

By

Mary Clare Morley

A Senior Thesis in partial fulfillment of the requirements

of the Glynn Family Honors Program

Spring 2021

Introduction:

Heart failure is a chronic, progressive condition in which the heart cannot pump enough blood to meet the needs of the body. There are different types of heart failure corresponding to which part of the heart is failing, and whether it is failing by being unable to relax, or unable to contract. The type of heart failure that this project is concerned with is systolic dysfunction, which occurs when the heart muscle is weakened and cannot contract like it is supposed to, causing less blood to be pumped out of the ventricles (Ahmad, 2017; Chicco, 2020).

Heart failure can be a sign of underlying cardiovascular disease, and finding a way to predict it is thus becoming a priority for doctors. So far, there is evidence that doctors are less accurate at predicting heart failure patient prognosis than predictive models (Chicco, 2020; Buchan, 2019). Electronic medical records that quantify symptoms, test results and other features allow statistics to be used to highlight patterns and correlations that may not otherwise be detected by doctors. Statistical models can be used to predict a patient's prognosis from their data, and can assess which features are most important among those included in medical records (Buchan, 2019).

In my thesis, I analyzed a dataset consisting of 299 medical records from patients who were suffering from systolic heart failure. I utilized five statistical modeling methods: classical logistic regression, lasso logistic regression, single decision tree, bagged decision trees and a random forest, to create models that could predict whether or not a patient survives based on the demographic data and medical test results present in the dataset. I implemented all three of these models using the *tidyverse* package in the R programming language (R Core Team,

2020; Wickham et al., 2019). The *tidyverse* is a package consisting of several important packages for data cleaning, wrangling and manipulation that has made R one of the premier software languages for statistical analysis. This paper consists of an evaluation of these five methods and their usefulness in a case like this, as well as an evaluation of the efficacy of using the *tidyverse* package in using these methods. In addition, the models helped me to draw some conclusions about patterns and correlations between heart failure symptoms and patient data. I describe the data that will be used, then I apply the methods to the data and assess the results, and finally the methods will be compared and contrasted.

Data:

The data analyzed in this study is from the UC Irvine Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records). It contains 299 records of patients who had heart failure. Each record contains 13 features consisting of demographic information and medical measurements for each patient. These features are: age, sex, whether the patient is a smoker or not, anemia (Boolean), high blood pressure (Boolean), level of creatinine phosphokinase in the blood, diabetes (Boolean), ejection fraction or the percentage of blood leaving the heart at each contraction, amount of platelets in the blood, level of serum creatinine in the blood, level of serum sodium in the blood, and the number days in the patient's follow-up period. A Boolean variable is one that indicates whether something is true or false, for instance, whether a patient has anemia or not (Hosmer, 2013). The target event, or what we are looking to predict, is the column that indicates whether or not the patient died as a result of the heart failure.

The variables that will be focused on most closely are creatinine phosphokinase, serum creatinine, serum sodium, and ejection fraction. The presence of creatinine phosphokinase (CPK) in the blood indicates muscle tissue damage, while serum creatinine is a waste product produced when a muscle breaks down. Ejection fraction states the percentage of blood that the left ventricle of a heart pumps out at each contraction. A high presence of serum sodium is correlated with the correct functioning of muscles and nerves.

All of the patients were over 40 years old and were suffering from the same type of heart failure, systolic dysfunction (Ahmad, 2017). They also were all admitted to the same hospital in Faisalabad, Pakistan. This is important to note because conclusions drawn from this data may not necessarily be applicable to heart failure patients in other areas of the world. Previous studies have suggested that Pakistan is currently undergoing an “epidemic” of cardiovascular diseases, of which heart failure is often a concurrent condition, due to certain risk factors like hypertension and diabetes being made more likely by facets of Pakistani culture such as smoking and oily foods (Ahmad, 2017; Jafar, 2007).

In the models that will be discussed in the next section, I chose not to use the time variable as a predictive variable among the other features. This is because a patient’s follow-up time is obviously cut short when a patient dies, which skews the results. Additionally, complete follow-up time is a variable that would not be accessible to a medical professional when predicting a patient’s prognosis, so it is more realistic to not include it (Chicco, 2020).

Methods:

i. Logistic Regression

The target variable in this study, whether the patient dies or not is nominal, meaning that there exist discrete levels of the variable between which there is no natural order or distance. More specifically, the response is binary, because there are only two of these unordered levels. Because of this, logistic regression must be used instead of the more common linear regression. Logistic regression is a basic statistical model that uses a logistic function to model a binary dependent variable. Logistic regression models the probability p that Y belongs to a particular category; in our case, the probability that a patient will die. To do this it uses the logistic function:

$$\mathrm{p(X)} = \frac{\mathrm{e}^{\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_n X_n}}{1+\ \mathrm{e}^{\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_n X_n}}$$

where p is the probability that Y will occur based on particular X values $(X_1, \ldots X_n)$ corresponding to features 1 through n. (Hosmer, 2013; James, 2017).

To apply this method in R, one would usually use the `glm` function:

```
model <- glm(y ~ x1 + x2, data = data, family = "binomial")
```

where `y` is the response variable and `x1` and `x2` are predictor variables.

ii. LASSO Logistic Regression

LASSO Regression is similar to the more basic linear and logistic regressions, but it uses a penalty term to shrink the effects of less-important features in order to encourage simple models. Unlike its alternative, ridge regression, LASSO regression can perform variable selection while modeling the response variable by shrinking the coefficients of these less important

features all the way to zero, thus eliminating their influence from the model (James, 2017). The objective function for this penalized form of logistic regression is:

$$\min_{(\beta 0,\beta)\in\mathbb{R}p+1} -[1/N \sum_{i=1}^{N} y_i \cdot (\beta_0 + x_i^T\beta) - \log(1 + e^{\beta_0 + x_i^T\beta)})] + \lambda\left[\frac{(1-\alpha)\|\beta\|_2^2}{2} + \alpha\|\beta\|_1\right]$$

The above equation is for a mixture model, combining both lasso and ridge logistic regression. In the lasso case, α is equal to 1. The λ value controls how much the coefficients are shrunk. When λ is equal to zero, the lasso model gives the same fit as the logistic regression above, while a sufficiently large λ can cause all coefficient estimates to be zero (James, 2017).

Obtaining this lambda value is a very important part of the lasso process. The `cv.glmnet` function in the *glmnet* package uses cross-validation to find the optimal lambda (Friedman, 2010). The grid of potential lambda values that are iterated through in the cross-validation is selected by a complicated pathwise coordinate descent process (Friedmann, 2010).

To apply this method in R, one would usually use the `cv.glmnet` function:

```
model <- cv.glmnet(x,y, alpha = 1, family = "binomial",
                   type.measure = "class")
```

where `y` is the response variable and `x` is a matrix of the predictor variables (Hastie, 2014).

iii. Decision Tree

Decision trees are a natural model for classification. A decision tree is a flowchart-like structure that displays an algorithm in terms of a series of segmented regions in the prediction space. In order to make a prediction for a given observation, decision trees use the mode of the training observations in the region to which the given observation belongs (James, 2017). Trees decide where to split in a node by calculating the loss of diversity of all possible splits and

choose the one with the largest loss of diversity, meaning the split that most efficiently splits one category of outcomes from another.

To apply this method in R, one would usually use the `rpart` function:

```
tree <- rpart(y ~ x1 + x2, data = data, method = "class")
```

where `y` is the response variable, `x1` and `x2` are predictor variables, and `"class"` is indicating that we want to produce a classification tree (predicting a qualitative response) rather than a regression tree (predicting a quantitative response) (James, 2017).

Trees are useful because they work on all variable types and are very interpretable, though they could be less accurate than other methods due to the large error that can occur near region boundaries.

iv. Bagged Decision Tree

Bagging, also known as "bootstrap aggregating" is an ensemble method, meaning that it combines several base models to produce one optimal predictive model. They can be used to raise accuracy of unstable base prediction methods, such as decision trees. Bootstrapping is a form of resampling where sets of the same sample size of the original data are made with replacement (James, 2017).

To apply this method in R, one could use the `bagging` function:

```
bagged_tree <- bagging(y ~ x1 + x2, data = data, nbagg = 100)
```

where `y` is the response variable, `x1` and `x2` are predictor variables, and `nbagg` is the number of aggregated trees (Peters, 2019).

A bagged decision tree is produced by constructing a decision tree for each of a number of bootstrap sample sets, and choosing the splits that were in the majority of these trees (James, 2017).

v. Random Forest

Random forests are similar to bagged decision trees, with the added element of randomness in features that are chosen in each tree (James, 2017). At each node, the algorithm only considers some of the features, and chooses the best split among these features.

To apply this method in R, once could use the *randomForest* package:

```
forest <- randomForest(y ~ x1 + x2, data = data, ntree=300)
```

where `y` is the response variable, `x1` and `x2` are predictor variables and "ntree" is the number of trees in the forest.

Ensemble methods often lead to a big gain in performance compared to single decision trees by lowering the misclassification rate. The downside to this is that they are less interpretable, and thus provide less information about the context of the situation.

0. Confusion Matrices

I will assess the predictive accuracy of each method via confusion matrices, which are 2x2 matrices showing how many subjects were correctly classified for each true value.

Table 1: General format of confusion matrix

	true survived	true died
predicted survived	n_{11}	n_{12}
predicted died	n_{21}	n_{22}

An example is shown above. True survived and true died indicates the true outcomes of individual patients, in this case, whether or not they died as a result of their heart failure. Predicted survived and predicted died indicate the results of the model. Therefore, n_{11} is the number of patients in the testing set that the model predicted to have survived and that actually survived, while n_{12} is the number of patients that died and that the model incorrectly classified as having survived. A confusion matrix displaying the results of a perfectly accurate model would have n_{12} and n_{21} equal to zero.

Applications and Visualizations:

Out of the 299 patients, 203 (68%) survived while 96 (32%) died due to their heart failure. In order to be able to test the models thoroughly, the data was split up into two sets, one intended for training the models, and one on which the models will be tested. This proportion of 68:32 survivors to deaths was retained to ensure that both sets were representative of the entire population. The training set contains 225 patient's records, while the testing set contains the remaining 74 patient's records.

i. First, I applied a logistic regression model to the training data. I used the *glm* package within the *tidymodels* framework. Table 2 shows a summary of the model's results. The variables that are significant to the model, as indicated by their p-values are age, ejection_fraction, serum_creatinine and serum_sodium. A one year increase in age will raise a patient's log odds of dying from heart disease by about 0.049. A one-percentage increase in ejection fraction will lower the log odds of the patient dying from heart failure by 0.082

Table 2: Results of logistic regression model

Predictors	Estimates	Standard Error	Z Statistic	P-value
age	0.0489	0.0154	3.1700	0.0015
anaemia	0.3320	0.3550	0.9360	0.3490
creatinine_phosphokinase	0.0003	0.0002	1.9500	0.0517
diabetes	-0.3110	0.3580	-0.8680	0.3860
ejection_fraction	-0.0819	0.0179	-4.5700	0.0000
high_blood_pressure	-0.0284	0.3610	-0.0787	0.9370
platelets	0.0000	0.0000	0.0537	0.9570
serum_creatinine	0.7790	0.2250	3.4700	0.0005
serum_sodium	-0.0821	0.0401	-2.0500	0.0405
sex	-0.6480	0.4190	-1.5500	0.1220
smoking	0.3540	0.4190	0.8460	0.3980

After applying the logistic regression model to the training data, the model was used to predict the outcomes of the 74 patients represented in the testing data. Table 3 shows the comparison of the model's predictions with the actual outcomes of the test patients.

Table 3: Confusion matrix displaying accuracy of logistic regression model

	true survived	true died
predicted survived	45	15
predicted died	5	9

This confusion matrix shows that this model has fairly good predictive power, correctly classifying 54/74 of the patients, about 73%.

ii. The training data was then used to train a LASSO model. I used the *glmnet* package within the *tidymodels* framework. *Tidymodels* supports a "mixture" of regularization methods in logistic regression, so I set mixture = 1, to ensure that lasso regularization was the only type used.

The aforementioned pathwise coordinate descent process that is built into the commonly used *glmnet* package is not included in the *tidymodels* frame work. To obtain the grid of lambda values to be optimized through cross-validation I ran the same data in `cv.glmnet` so that I could see what the maximum and minimum lambda values would be. Based on this, I tested 100 values equally spaced between 10^-4 and 10^-.5. I used the `tune_grid` function in *tidymodels* to choose the optimal value based on two metrics, the area under the ROC curve, and the accuracy rate. The ROC curve is the curve formed by plotting a predictive model's true positive rate against its false positive rate. A larger area under this curve, also known as AUC, represents a better model (James, 2017). To optimize lambda, I created a cross-validation object. This split the training data into ten equal groups that would then each be tested against models trained on the other nine groups. Once again, the proportion of survivors to deaths was retained in these ten smaller groups.

Figure 1 shows the AUC as a function of penalty terms. It seems from this plot that a lambda value between 0.01 and 0.1 would be best, because this is the range on the x-axis in which the highest AUC occurred.

Figure 2 similarly shows the accuracy rate as a function of penalty. It seems from this plot that a term between .001 and .01 would be best, but between .01 and .1 would not be bad either. Since we want to maximize both metrics, a lambda value of 0.0381 was chosen (Finch). I then trained the final lasso model with this lambda value as the penalty.

Lasso regression is especially useful for determining which features are most important, so examining which features the model chose to shrink could give us more information about what variables are most important in predicting a patient's death from heart failure.

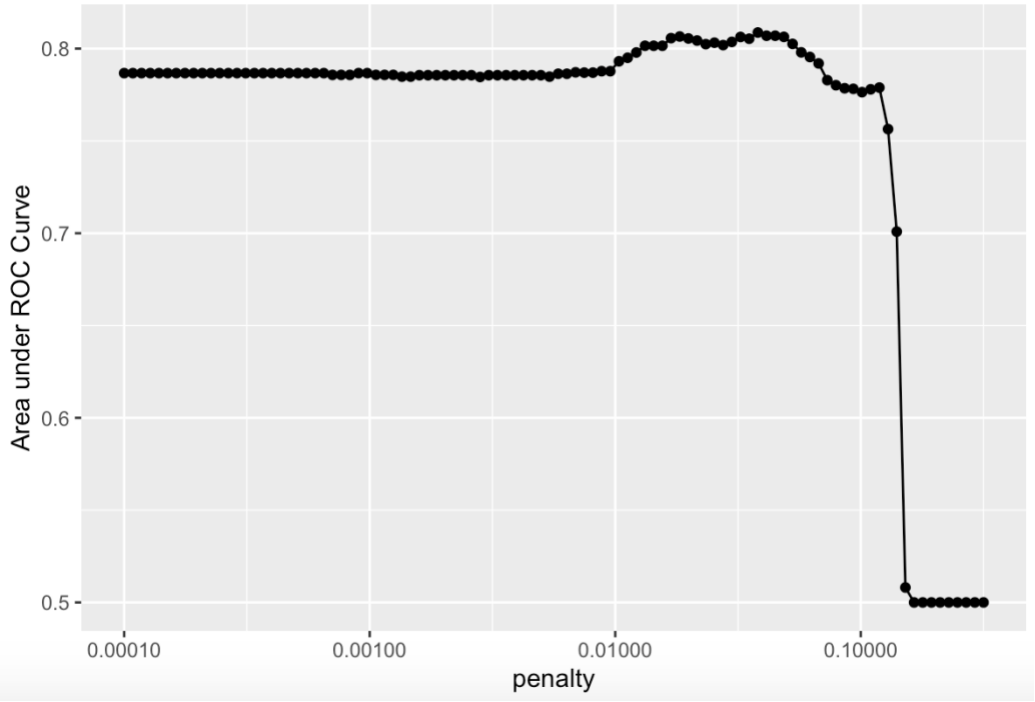

Figure 1: the AUC as a function of lasso penalty

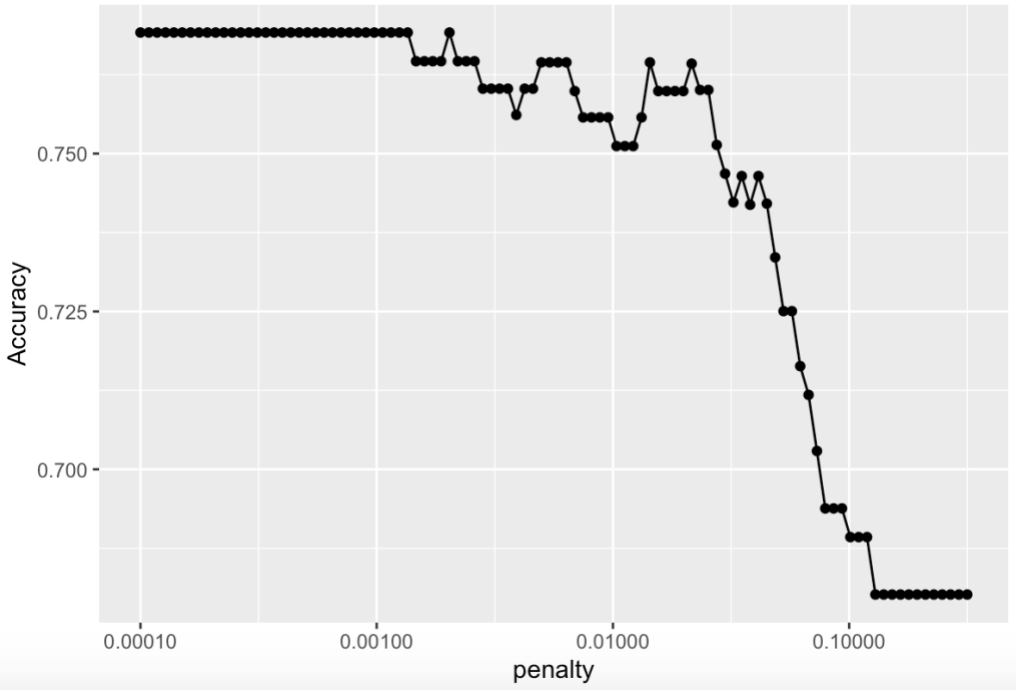

Figure 2: Accuracy rate as a function of lasso penalty

Table 4: Results of the lasso model

Predictor	Estimate
age	0.0414
anaemia	0.1530
creatinine_phosphokinase	0.0002
diabetes	-0.1660
ejection_fraction	-0.0703
high_blood_pressure	0
platelets	0
serum_creatinine	0.6630
serum_sodium	-0.0629
sex	-0.3300
smoking	0.0752

As can be seen from Table 4, two variables had their coefficients shrunk all the way to zero: high_blood_pressure and platelets. This means that the model determined that these variable were not important when predicting death due to heart failure. Additionally, creatinine_phosphokinase had a coefficient very close to zero, indicating that this variable is also not very important to prediction. It is difficult to derive log odd interpretations from these results, like was done from the logistic regression, because the penalty term in the algorithm makes the estimates biased. The lasso model is intended mostly for prediction, so these coefficients can only provide direction as to which variables are important and which ones are not.

The final lasso model was then tested against the test set to produce the confusion matrix shown in Table 5. The accuracy rate, at 73%, is the same as that of the regular logistic regression.

Table 5: Confusion matrix showing accuracy rate of lasso model

	true survived	true died
predicted survived	45	15
predicted died	5	9

iii. Then a decision tree was made using the training data and the *rpart* engine in the *tidymodels* framework. I tuned two parameters: the cost complexity value, and the maximum tree depth, or how many nodes a particular patient could go through. Similar to what I did for the LASSO model, I used the `tune_grid` function to find the optimal tree depth and cost complexity for the decision tree.

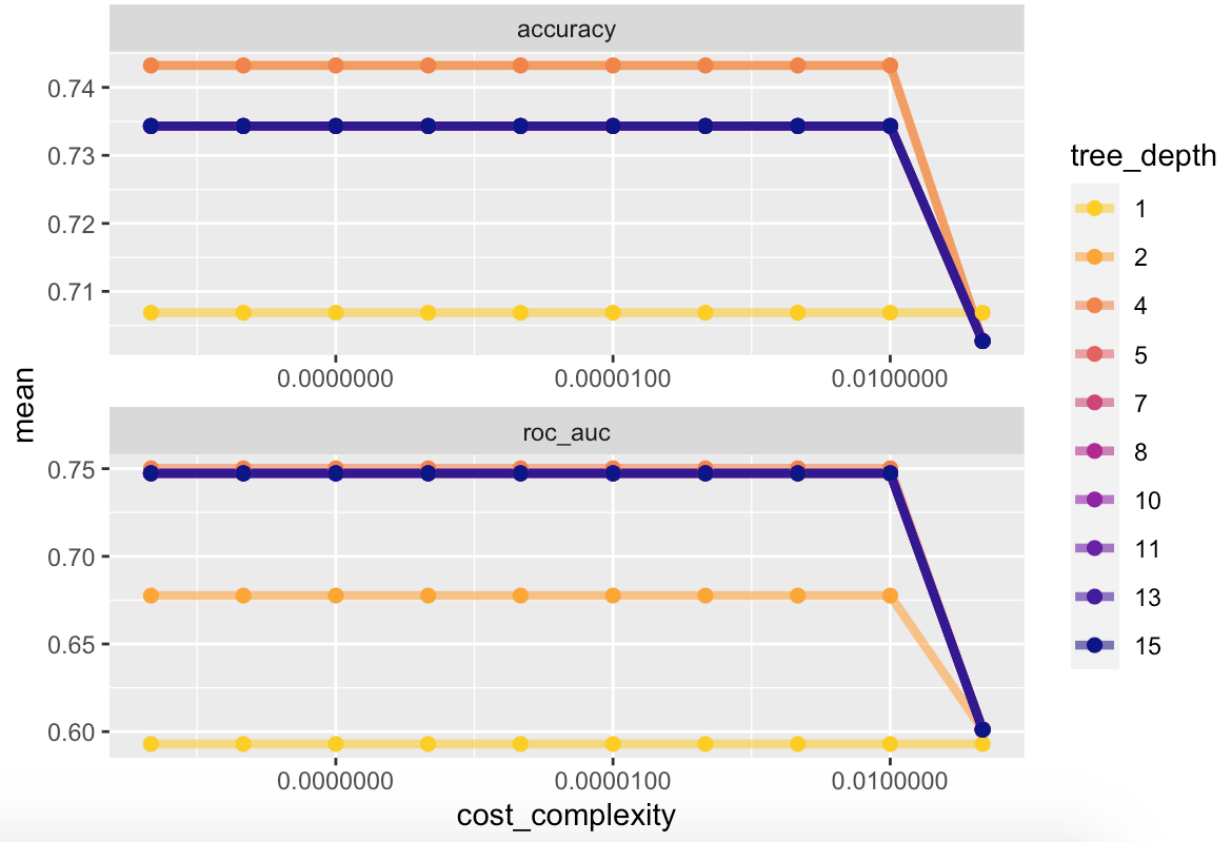

Figure 3: AUC and accuracy as a function of cost complexity at various tree depths

Figure 3 shows the accuracy and AUC as a function of cost complexity at various tree depths. Cross-validation was used to obtain these metrics, testing 100 different pairs of cost complexity and tree depth values consisting of ten values of each parameter. As you can see, differences in tree depth caused the most difference in both metrics, while differences in cost complexity less than 0.01 cause no difference. To maximize both metrics, a tree depth of four was chosen with a cost complexity value of 10^-10.

A final decision tree model was then trained on the training data with the optimized parameters, and the resulting plot can be seen in Figure 4. The root node, at the top, indicated that patients with a serum creatinine level of less than 1.8 have a 32% chance of dying. Then we can see from the following nodes that 84% of patients fall in this category, while 16% do not. If a patient has less than 1.8 serum creatinine, an ejection fraction of at least 33%, and are under age 80, they only have an 8% chance of dying as a result of their heart failure, as can be seen in the dark blue box in the bottom left corner of the plot. This is the leaf, or bottom node, that is by far the most common for patients to fall into, at 53% of patients belonging to these three categories. On the other hand, if a patient has a serum creatinine level of over 1.8 and has a serum sodium level of less than 136, they have an 85% chance of dying, as can be seem in the dark green box in the bottom right corner of the plot. This is the second most common leaf node, and the most common leaf node indicating a death outcome, with 12% of patients falling in both of the qualifying categories. The other leaf nodes are less common, and indicate patients whose prognosis is less clear.

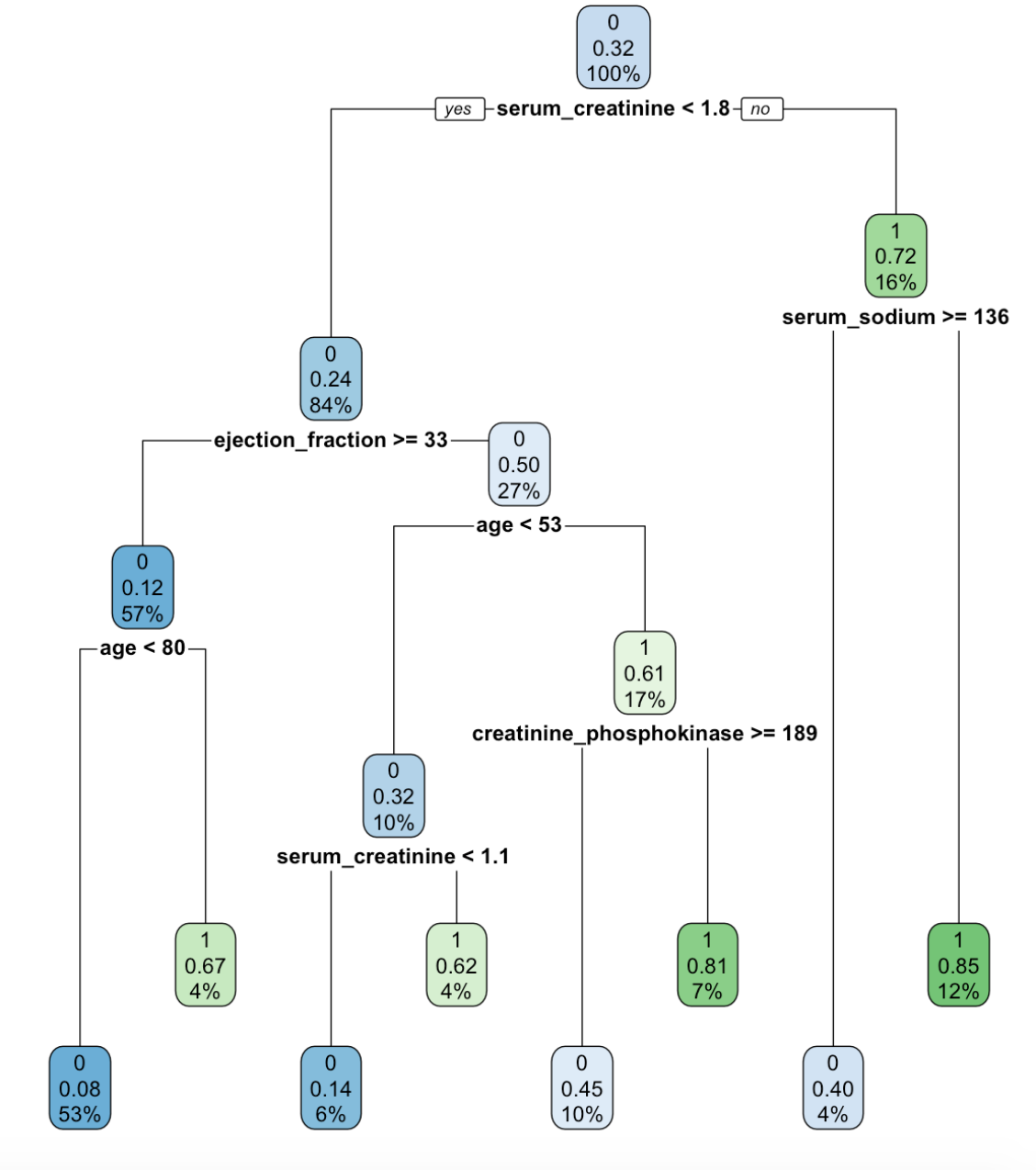

Figure 4: Resulting graphic of decision tree

The above decision tree was tested with the testing data to produce the confusion matrix shown in Table 6. This method has a higher overall accuracy rate than the prior two, at 78.4%.

Table 6: Confusion matrix showing accuracy of decision tree

	true survived	true died
predicted survived	45	11
predicted died	5	13

iv. The *rpart* engine was used again, but this time to create a bagged decision tree model. Twenty-five bootstrap resamples were used.

Here, the `fit_resamples` function was used to optimize the parameters `cost_complexity` and `min_n`, or the minimum data points in a node that are required for the node to be split further. After resampling, a cost complexity of 0, and a minimum n of 2 was chosen.

Using these parameters, a final model was made. This bagged decision tree was tested with the testing data to produce the confusion matrix shown in Table 7.

Table 7: Confusion matrix showing accuracy of bagged decision trees

	true survived	true died
predicted survived	45	13
predicted died	5	11

This method had an accuracy rate of 75.7%, which is higher than that of logistic regression and lasso regression, but lower than that of the single decision tree.

v. Finally, the *ranger* package was used within the *tidymodels* framework to produce a random forest model (Wright, 2017). All three parameters, `mtry` (the number of predictor that

are randomly sampled at each split within the trees), number of trees (`trees`), and `min_n` were tuned using cross-validation.

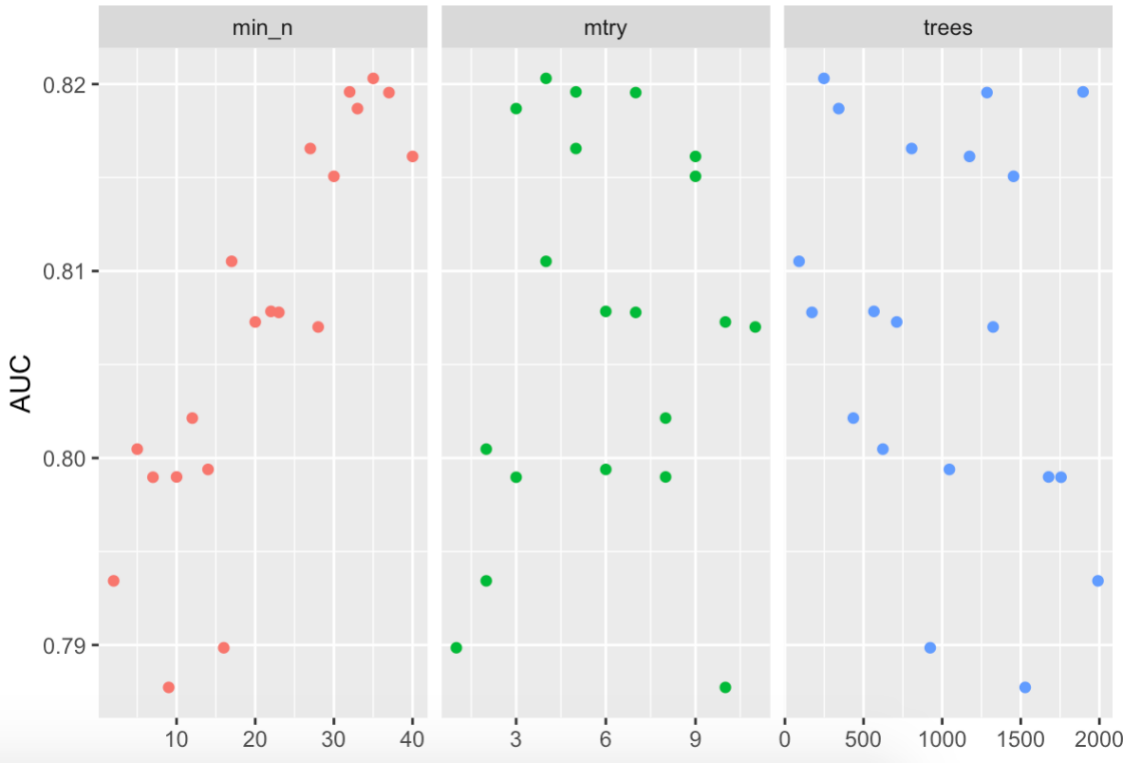

Figure 5: AUC of models with varying values of `min_n`, `mtry`, and `trees`

Figure 5 shows the AUC of various models with difference values of these parameters. Based on these results, the parameters will be tuned again within a smaller range. For instance, choosing a `min_n` between 30 and 40, a `mtry` between 3 and 9 and a number of `trees` between 200 and 2000 will allow us to retain the models shown here with the highest AUC while also optimizing for more specific parameter values.

These parameters were tuned within these ranges, and a combination of 3 predictors chosen at each split, 200 overall trees and 32 minimum subjects in order to split a node was found to maximize both AUC and accuracy.

A final random forest model was fit with these tuned parameters and was tested against the test set. The resulting confusion matrix is shown in Table 8.

Table 8: Confusion matrix showing accuracy of random forest

	true survived	true died
predicted survived	47	15
predicted died	3	9

This model had the same accuracy rate as the bagged decision trees, at 75.7%

Comparison of Methods:

Table 9: Comparison of accuracy of all methods.

Method	Accuracy	True Positive Rate	True Negative Rate
Logistic Regression	0.7297	0.9000	0.3750
Lasso Logistic Regression	0.7297	0.9000	0.3750
Decision Tree	0.7837	0.9000	0.5417
Bagged Decision Tree	0.7568	0.9000	0.4583
Random Forest	0.7568	0.9400	0.3750

As can be seen in Table 9, the single decision tree had the highest overall accuracy and the highest true negative rate (proportion of patients who were correctly predicted to have died) while the random forest had the highest true positive rate (proportion of patients who were correctly predicted to have survived).

In terms of predictive power, the single decision tree was able to predict the most individual patient's outcomes correctly, predicting the true outcome of 58 out of the 74 patients in the test set. Additionally, the single decision tree could be easily used in a clinical

setting since it can be represented in clear visualizations. One could follow the paths of the tree shown in Figure 4 to predict whether or not a patient would survive based on their symptoms. These factors combined with the easy interpretability of the decision tree makes the single decision tree the best model to use for this data.

The logistic regression and lasso logistic regression are also easily interpreted methods with a fairly high accuracy rate, making them also good models to use in this scenario. Additionally, statistical inference can be performed when using classical logistic regression. For instance, one could obtain confidence intervals for the estimated coefficients, information that would be important when using the model for explanation rather than prediction.

The ensemble methods, the bagged decision tree and random forest, did not perform well enough in predictability to outweigh the loss of interpretability that comes from aggregating less complicated models. Both of these methods usually raise performance from their simpler method, which in this case is the single decision tree. The fact that this did not occur in this study could be due to the newness of the *tidymodels* framework and thus the lack of helpful functions in optimizing these more complicated methods. It would be interesting to use the same methods in a more classical R setting, to see if the ensemble methods would raise the accuracy rate then, as they are expected to.

Conclusions:

Overall, the single decision tree model performed the best on the test set of data. Prior studies conducted on this data indicated that serum creatinine and ejection fraction were the most important variables in predicting a patient's survival, and this study does not refute that

conclusion (Chicco). Along with these two variables, the decision tree focuses on serum sodium, creatinine phosphokinase, and age.

It follows, based on the preceding definition of heart failure and the corresponding variables, that high serum creatinine and CPK and low ejection fraction and serum sodium would indicate heart failure. Extremely high or low values would indicate more extreme heart failure. Having a way to relate these symptoms to each other would be helpful in a real-life assessment of a patient, and that is what the decision tree provides. Additionally, the decision tree algorithm's search for a loss of diversity leads to the creation of thresholds for some of these variables that could also be helpful information. For instance, while it is known that more serum creatinine indicates worse heart failure, the conclusion that heart failure patients with a serum creatinine level over 1.8 have a 68% chance of dying due to their heart failure gives more specificity and context to this trend.

This work could be expanded by applying different predictive methods, such as support vector machines, a popular approach for classification problems, or mixture models, logistic regressions that combine lasso with its related ridge regression. Both of these models involve parameters that must be tuned, like the lambda was tuned for the lasso model above. This means that they would be prime candidates for modeling in *tidymodels*, as *tidymodels* includes many useful functions for tuning.

Additionally, prior research on this subject seems to be especially focused on Southeast Asia, Pakistan in particular (Chicco, 2020). It would be worthwhile to repeat a similar study with patients from a different area of the world. As mentioned before, certain aspects of the Pakistani lifestyle may cause patients there to be more prone to heart failure, and more severe

heart failure (Jafar, 2007). Similar data gathered from patients of other cultures could shed light on to what extent and in what ways heart failure risk differs around the world.

Bibliography

Ahmad, Tanvir, et al. “Survival Analysis of Heart Failure Patients: A Case Study.” *Plos One*, vol. 12, no. 7, 2017, doi:10.1371/journal.pone.0181001.

Buchan, T.A., et al. “Physician Prediction versus Model Predicted Prognosis in Ambulatory Patients with Heart Failure.” *The Journal of Heart and Lung Transplantation*, vol. 38, no. 4, 2019, doi:10.1016/j.healun.2019.01.971.

Chicco, Davide, and Giuseppe Jurman. “Machine Learning Can Predict Survival of Patients with Heart Failure from Serum Creatinine and Ejection Fraction Alone.” *BMC Medical Informatics and Decision Making*, vol. 20, no. 1, 2020, doi:10.1186/s12911-020-1023-5.

Finch, W. Holmes and Maria E. Hernandez Finch. "Regularization Methods for Fitting Linear Models with Small Sample Sizes: Fitting the Lasso Estimator using R." *Practical Assessment, Research, and Evaluation*, vol. 21 , no. 7, 2016.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22. URL http://www.jstatsoft.org/v33/i01/.

Hastie, Trevor, and Junyang Qian. “Glmnet Vignette.” *Stanford University*, 26 June 2014, web.stanford.edu/~hastie/glmnet/glmnet_alpha.html.

Hosmer, David W., et al. *Applied Logistic Regression*. Wiley, 2013.

Jafar, Tazeen H., et al. “Coronary Artery Disease Epidemic in Pakistan: More Electrocardiographic Evidence of Ischaemia in Women than in Men.” *Heart*, vol. 94, no. 4, 2007, pp. 408–413., doi:10.1136/hrt.2007.120774.

Jafar, Tazeen H., et al. “Heart Disease Epidemic in Pakistan: Women and Men at Equal Risk.” *American Heart Journal*, vol. 150, no. 2, Aug. 2005, pp. 221–226., doi:10.1016/j.ahj.2004.09.025.

James, Gareth, et al. *An Introduction to Statistical Learning with Applications in R*. Springer, 2017.

Peters, Andrea and Torsten Hothorn. ipred: Improved Predictors. R package version 0.9-9. https://CRAN.R-project.org/package=ipred, 2019.

R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Therneau, Terry and Beth Atkinson (2019). rpart: Recursive Partitioning and Regression Trees. R package version 4.1-15. https://CRAN.R-project.org/package=rpart

Wickham et al., (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686

Wright, Marvin N. and Andreas Ziegler (2017). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. Journal of Statistical Software, 77(1), 1-17. doi:10.18637/jss.v077.i01

Appendix:

```
heart_failure <- read.csv("~/Downloads/heart_failure_clinical_records_dataset.csv")

heartfail = subset(heart_failure, select = -c(time))
heartfail$DEATH_EVENT = as.factor(heartfail$DEATH_EVENT)
# should all variables be factors??
heartfail$anaemia = as.factor(heartfail$anaemia)
heartfail$diabetes = as.factor(heartfail$diabetes)
heartfail$high_blood_pressure = as.factor(heartfail$high_blood_pressure)
heartfail$sex = as.factor(heartfail$sex)
heartfail$smoking = as.factor(heartfail$smoking)

glimpse(heartfail)
heartfail %>%
  count(DEATH_EVENT) %>%
  mutate(prop = n/sum(n))
# see that in the data, 32% of the subjects died

set.seed(123)
install.packages("rsample")
library(rsample)
splits <- initial_split(heartfail, strata = DEATH_EVENT)
heart_other <- training(splits)
heart_test <- testing(splits)
heart_test %>%
  count(DEATH_EVENT) %>%
  mutate(prop = n/sum(n))
heart_other %>%
  count(DEATH_EVENT) %>%
  mutate(prop = n/sum(n))

install.packages("generics")
library(generics)
lr_mod <-
  logistic_reg(mode = "classification") %>%
  set_engine("glm") %>%
  fit(as.factor(DEATH_EVENT) ~ ., data = heart_other)

predictions_glm <- lr_mod %>%
  predict(new_data = heart_test) %>%
  bind_cols(heart_test %>% dplyr::select(DEATH_EVENT))
predictions_glm
```

```
confmat = table(predictions_glm)
confmat
accuracy = mean(predictions_glm$.pred_class == predictions_glm$DEATH_EVENT)
accuracy

ls_mod <-
  logistic_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet") #%>%
  #fit(as.factor(DEATH_EVENT) ~ ., data = heart_other)

library(recipes)
ls_recipe <-
  recipe(DEATH_EVENT ~ ., data = heart_other) #%>%
  #step_num2factor(DEATH_EVENT, levels = survived, skip = T)

ls_grid <- tibble(penalty = 10^seq(-4,-.5, length.out = 100))
set.seed(15)
heart_boot <- bootstraps(heart_other, strata = DEATH_EVENT)
heart_cv <- vfold_cv(heart_other, v = 10, repeats = 1, strata = DEATH_EVENT)

library(workflows)
ls_workflow <-
  workflow() %>%
  add_model(ls_mod) %>%
  add_recipe(ls_recipe)

library(tune)
ls_res <-
  ls_workflow %>%
  tune_grid(heart_cv,
            grid = ls_grid,
            control = control_grid(save_pred = TRUE),#)
            metrics = metric_set(roc_auc))
# gathering auc data on resampling

ls_plot <-
  ls_res %>%
  collect_metrics() %>%
  ggplot(aes(x = penalty, y = mean)) +
  geom_point() +
  geom_line() +
  ylab("Area under ROC Curve") +
  scale_x_log10(labels = scales::label_number())
```

```
ls_plot

top_models <-
  ls_res %>%
  show_best("roc_auc", n = 15) %>%
  arrange(penalty)
top_models
# showing top models based on auc

ls_res <-
  ls_workflow %>%
  tune_grid(heart_cv,
            grid = ls_grid,
            control = control_grid(save_pred = TRUE))
            #metrics = metric_set(yardstick::accuracy))
# getting accuracy data on resampling

ls_plot <-
  ls_res %>%
  collect_metrics() %>%
  ggplot(aes(x = penalty, y = mean)) +
  geom_point() +
  geom_line() +
  ylab("Accuracy") +
  scale_x_log10(labels = scales::label_number())

ls_plot

top_models <-
  ls_res %>%
  show_best("accuracy", n = 15) %>%
  arrange(penalty)
top_models
# showing top models based on accuracy

ls_res %>% collect_metrics()

ls_res %>%
  collect_metrics() %>%
  ggplot(aes(penalty, mean, color = .metric)) +
  geom_errorbar(aes(
    ymin = mean - std_err,
    ymax = mean + std_err
  ),
```

```
  alpha = 0.5
  ) +
  geom_line(size = 1.5) +
  facet_wrap(~.metric, scales = "free", nrow = 2) +
  scale_x_log10() +
  theme(legend.position = "none")

lambda <- ls_res %>%
  select_best("accuracy")

final_lasso <- finalize_workflow(
  ls_workflow,
  lambda
)

lasso_fit <- final_lasso %>%
  fit(heart_other)

tidy(lasso_fit)

library(vip)

# this isnt working rn
final_lasso %>%
  fit(heart_other) %>%
  pull_workflow_fit() %>%
  vi(lambda = lambda$penalty) %>%
  mutate(
    Importance = abs(Importance),
    Variable = fct_reorder(Variable, Importance)
  ) %>%
  ggplot(aes(x = Importance, y = Variable, fill = Sign)) +
  geom_col() +
  scale_x_continuous(expand = c(0, 0)) +
  labs(y = NULL)

predictions_lasso <- lasso_fit %>%
  predict(new_data = heart_test) %>%
  bind_cols(heart_test %>% dplyr::select(DEATH_EVENT))
predictions_lasso

confmat = table(predictions_lasso)
confmat
accuracy = mean(predictions_lasso$.pred_class == predictions_lasso$DEATH_EVENT)
```

```
accuracy


#decision tree part

dt_mod <-
  decision_tree(
    cost_complexity = tune(),
    tree_depth = tune()
  ) %>%
  set_engine("rpart") %>%
  set_mode("classification")

tree_grid <- grid_regular(cost_complexity(),
                          tree_depth(),
                          levels = 10)

dt_wf <- workflow() %>%
  add_model(dt_mod) %>%
  add_recipe(ls_recipe)

dt_res <-
  dt_wf %>%
  tune_grid(resamples = heart_cv,
            grid = tree_grid)

dt_res %>%
  collect_metrics()

dt_res %>%
  collect_metrics() %>%
  mutate(tree_depth = factor(tree_depth)) %>%
  ggplot(aes(cost_complexity, mean, color = tree_depth)) +
  geom_line(size = 1.5, alpha = 0.6) +
  geom_point(size = 2) +
  facet_wrap(~ .metric, scales = "free", nrow = 2) +
  scale_x_log10(labels = scales::label_number()) +
  scale_color_viridis_d(option = "plasma", begin = .9, end = 0)

dt_res %>%
  show_best("roc_auc")

dt_res %>%
  show_best("accuracy")
```

```
best_tree <- dt_res %>%
  select_best("roc_auc")

tree_final <-
  dt_wf %>%
  finalize_workflow(best_tree)

dt_final <-
  tree_final %>%
  fit(data = heart_other)

library(rpart.plot)
rpart.plot(dt_final$fit$fit$fit)

predictions_tree <- dt_final %>%
  predict(new_data = heart_test) %>%
  bind_cols(heart_test %>% dplyr::select(DEATH_EVENT))
predictions_tree

confmat = table(predictions_tree)
confmat
accuracy = mean(predictions_tree$.pred_class == predictions_tree$DEATH_EVENT)
accuracy

# bagged decision tree
library(baguette)
bt_mod <- bag_tree(
   # cost_complexity = tune(),
   # min_n = tune(),
  ) %>%
  set_engine("rpart", times=25) %>%
  set_mode("classification")

bt_wf <- workflow() %>%
  add_model(bt_mod) %>%
  add_recipe(ls_recipe)

set.seed(199)
fit_bag <- fit_resamples(
  bt_wf,
  heart_cv,
  #metrics = metric_set(rmse, rsq),
  control = control_resamples(verbose = TRUE,
```

```
                save_pred = TRUE,
                extract = function(x) extract_model(x)))

bt_best <- fit_bag %>%
  select_best("accuracy")

bt_final <- finalize_model(
  bt_mod,
  bt_best
)

bt_final <-
  bt_wf %>%
  finalize_workflow(bt_best)

bt_res <- bt_wf %>%
  fit(data = heart_other)

predictions_bag <- bt_res %>%
  predict(new_data = heart_test) %>%
  bind_cols(heart_test %>% dplyr::select(DEATH_EVENT))
predictions_bag

confmat = table(predictions_bag)
confmat
accuracy = mean(predictions_bag$.pred_class == predictions_bag$DEATH_EVENT)
accuracy

# random forest
library(ranger)
rf_mod <- rand_forest(
  mtry = tune(),
  trees = tune(),
  min_n = tune()
) %>%
  set_engine("ranger") %>%
  set_mode("classification")

rf_wf <- workflow() %>%
  add_model(rf_mod) %>%
  add_recipe(ls_recipe)

rf_res <- tune_grid(
  rf_wf,
```

```
  resamples = heart_cv,
  grid = 20
)

rf_res

rf_res %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  select(mean, min_n, mtry, trees) %>%
  pivot_longer(min_n:trees,
               values_to = "value",
               names_to = "parameter"
  ) %>%
  ggplot(aes(value, mean, color = parameter)) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~parameter, scales = "free_x") +
  labs(x = NULL, y = "AUC")

rf_grid <- grid_regular(
  mtry(range = c(3, 9)),
  trees(range = c(200,2000)),
  min_n(range = c(30, 40)),
  levels = 5
)

set.seed = (302)
rf_res2 <- tune_grid(
  rf_wf,
  resamples = heart_cv,
  grid = rf_grid
)

rf_best <- rf_res2 %>%
  select_best("roc_auc")

rf_final <- finalize_model(
  rf_mod,
  rf_best
)

rf_final <-
  rf_wf %>%
  finalize_workflow(rf_best)
```

```
rf_final <-
  rf_final %>%
  fit(data = heart_other)

rf_final

predictions_rf <- rf_final %>%
  predict(new_data = heart_test) %>%
  bind_cols(heart_test %>% dplyr::select(DEATH_EVENT))
predictions_rf

confmat = table(predictions_rf)
confmat
accuracy = mean(predictions_rf$.pred_class == predictions_rf$DEATH_EVENT)
accuracy
```