PSTAT 131 - Homework Assignment 3

Akshat Ataliwala (7924145)

April 17, 2022

Classification

For this assignment, we will be working with part of a Kaggle data set that was the subject of a machine learning competition and is often used for practicing ML models. The goal is classification; specifically, to predict which passengers would survive the Titanic shipwreck.

Load the data from `data/titanic.csv` into *R* and familiarize yourself with the variables it contains using the codebook (`data/titanic_codebook.txt`).

Notice that `survived` and `pclass` should be changed to factors. When changing `survived` to a factor, you may want to reorder the factor so that *"Yes"* is the first level.

Make sure you load the `tidyverse` and `tidymodels`!

Remember that you'll need to set a seed at the beginning of the document to reproduce your results.

```
library(tidymodels)
library(tidyverse)
library(ggplot2)
library(corrr)
library(klaR)
library(MASS)
library(discrim)
library(poissonreg)
tidymodels_prefer()
data <- read_csv("data/titanic.csv")
data %>% head(5)
```

```
## # A tibble: 5 x 12
##   passenger_id survived pclass name  sex     age sib_sp parch ticket  fare cabin
##          <dbl> <chr>     <dbl> <chr> <chr> <dbl>  <dbl> <dbl> <chr>  <dbl> <chr>
## 1            1 No            3 Brau~ male     22      1     0 A/5 2~  7.25 <NA>
## 2            2 Yes           1 Cumi~ fema~    38      1     0 PC 17~ 71.3  C85
## 3            3 Yes           3 Heik~ fema~    26      0     0 STON/~  7.92 <NA>
## 4            4 Yes           1 Futr~ fema~    35      1     0 113803 53.1  C123
## 5            5 No            3 Alle~ male     35      0     0 373450  8.05 <NA>
## # ... with 1 more variable: embarked <chr>
```

```
data$survived <- as.factor(data$survived)
data$survived <- relevel(data$survived, "Yes")
data$pclass <- as.factor(data$pclass)
data %>% head(5)
```

```
## # A tibble: 5 x 12
##   passenger_id survived pclass name  sex     age sib_sp parch ticket  fare cabin
##          <dbl> <fct>    <fct>  <chr> <chr> <dbl>  <dbl> <dbl> <chr>  <dbl> <chr>
## 1            1 No       3      Brau~ male     22      1     0 A/5 2~  7.25 <NA>
## 2            2 Yes      1      Cumi~ fema~    38      1     0 PC 17~ 71.3  C85
## 3            3 Yes      3      Heik~ fema~    26      0     0 STON/~  7.92 <NA>
## 4            4 Yes      1      Futr~ fema~    35      1     0 113803 53.1  C123
## 5            5 No       3      Alle~ male     35      0     0 373450  8.05 <NA>
## # ... with 1 more variable: embarked <chr>
```

1. Split the data, stratifying on the outcome variable, survived. You should choose the proportions to split the data into. Verify that the training and testing data sets have the appropriate number of observations. Take a look at the training data and note any potential issues, such as missing data.Why is it a good idea to use stratified sampling for this data?

```
set.seed(3478)

data_split <- initial_split(data, prop = 0.8, strata = survived)

train <- training(data_split)
test <- testing(data_split)
```

```
dim(data)
```

```
## [1] 891  12
```

```
0.8 * nrow(data)
```

```
## [1] 712.8
```

```
dim(train)
```

```
## [1] 712  12
```

```
dim(test)
```

```
## [1] 179  12
```

Here, we can see that our total dataset had 891 observations and 12 columns. I chose a train/test split of 80% of our data being used for training, meaning that we should expect 0.8 * 891 = 712.8 rows in the training set. This is indeed the case as the train set has 712 rows and the remaining 179 are in the test set.

```
colSums(is.na(train)) / nrow(train)
```

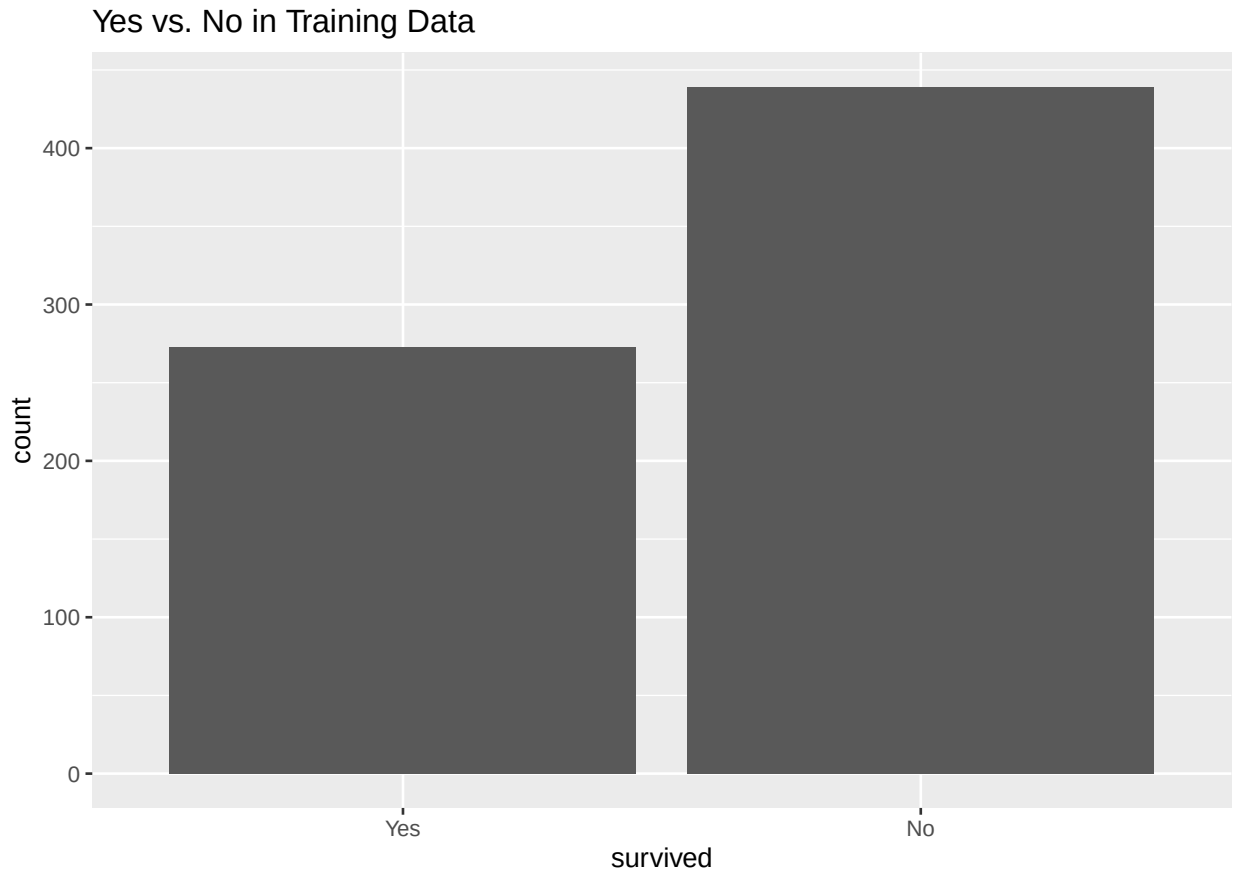
```
## passenger_id     survived       pclass         name          sex          age
##     0.000000     0.000000     0.000000     0.000000     0.000000     0.206461
##       sib_sp        parch       ticket         fare        cabin     embarked
##     0.000000     0.000000     0.000000     0.000000     0.764045     0.002809
```

Here, I am looking at the proportion of missing values for each feature in the training set. As you can see, there are 3 features with missing values: age, cabin, and embarked. Their respective missing proportions are 20.6%, 76.4%, and 0.2%.

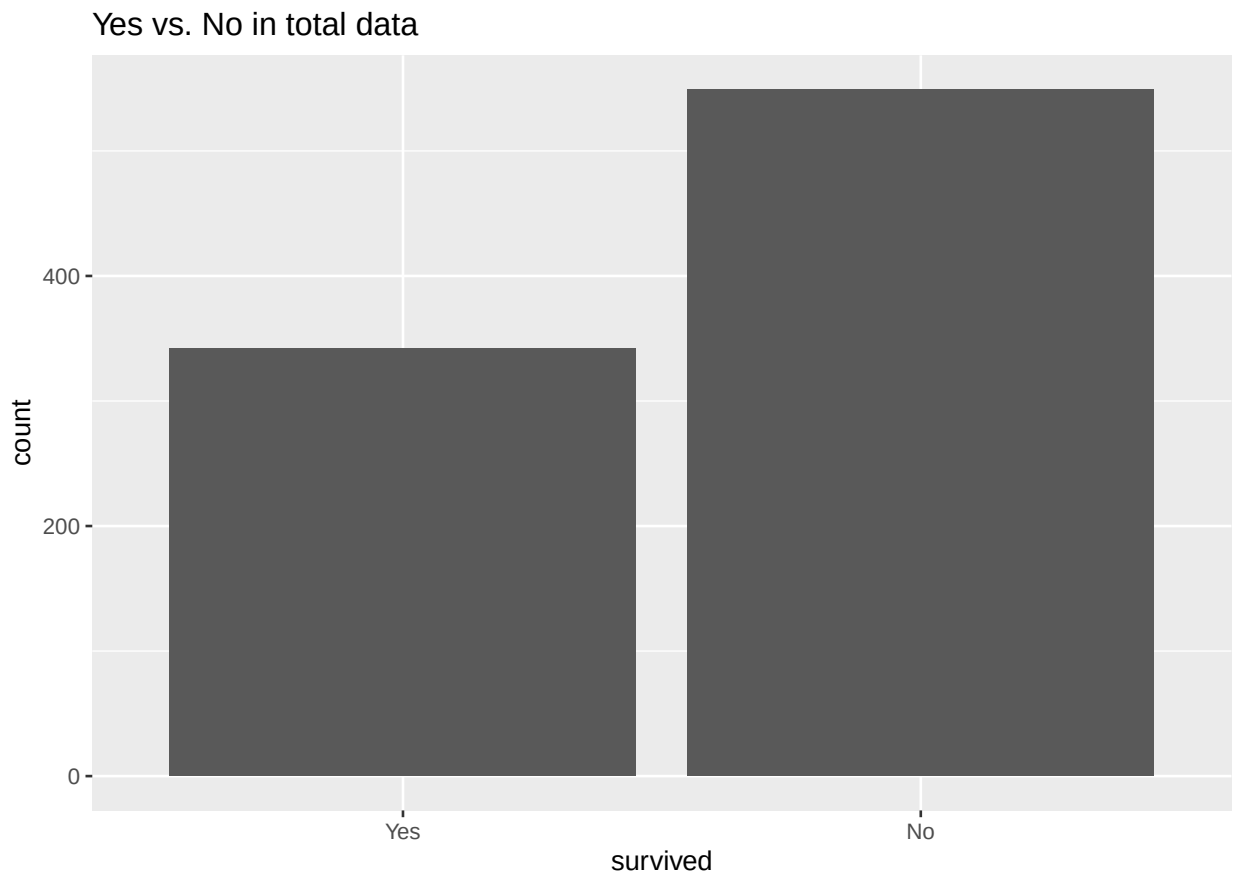
It's a good idea to use stratified sampling because it enables us to obtain a sample that best represents the entire population being sampled from. In this case, we stratify by our response variable, survived, so there appropriate representation of both survived and not that our model can use to learn on.

2. Using the training data set, explore/describe the distribution of the outcome variable `survived`.

```
sur_train_dist <- ggplot(train, aes(x = survived)) + geom_bar() +
  ggtitle("Yes vs. No in Training Data")
sur_train_dist
```




```
sur_data_dist <- ggplot(data, aes(x = survived)) + geom_bar() +
  ggtitle("Yes vs. No in total data")
sur_data_dist
```



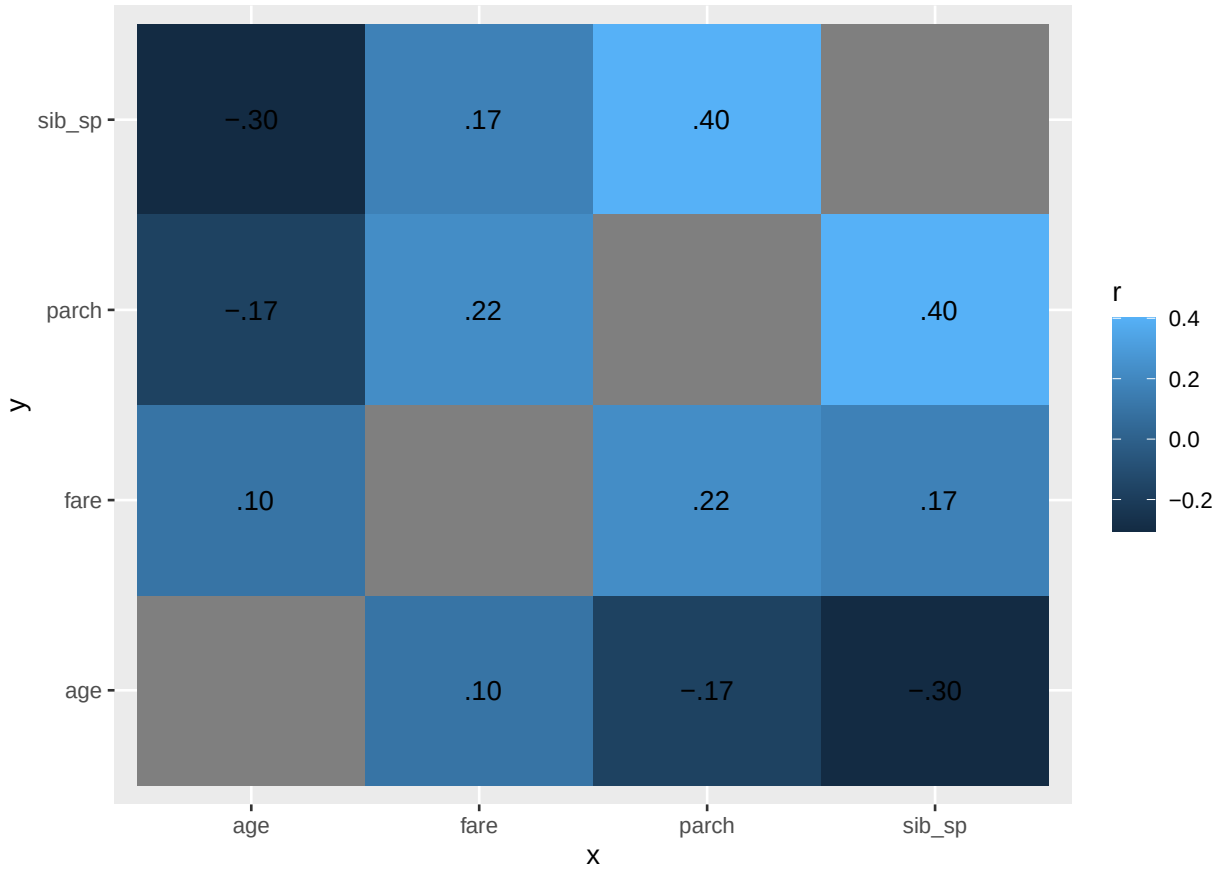
As you can see, there are 30-40% more values for No compared to Yes, meaning that there were far more people that didn't survive as opposed to those that did. This is why we stratify by Survived, to ensure that the populations of Yes and No are proportionally represented in the training data when compared to the total dataset.

3. Using the training data set, create a correlation matrix of all continuous variables. Create a visualization of the matrix, and describe any patterns you see. Are any predictors correlated with each other? Which ones, and in which direction?

```
corr_titanic <- train %>%
  select("age", "sib_sp", "parch", "fare") %>%
  correlate() %>%
  stretch() %>%
  ggplot(aes(x, y, fill = r)) + geom_tile() +
  geom_text(aes(label = as.character(fashion(r))))
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

```
corr_titanic
```



Based on our data, passenger id, pclass, name, sex, ticket, and embarked are all discrete/non-numeric variables, and survived is our response variable. Thus, our continuous predictor variables include age, sub_sp, parch, and fare. Overall, there is not much correlation between these variables, with the highest correlation being a +0.40 between sib_sp and parch, suggesting that as the # of siblings/spouses aboard the titanic increases, the # of parents/children aboard increases moderately as well.

4. Using the training data, create a recipe predicting the outcome variable `survived`. Include the following predictors: ticket class, sex, age, number of siblings or spouses aboard, number of parents or children aboard, and passenger fare.

Recall that there were missing values for `age`. To deal with this, add an imputation step using `step_impute_linear()`. Next, use `step_dummy()` to **dummy** encode categorical predictors. Finally, include interactions between:

- Sex and passenger fare, and
- Age and passenger fare.

You'll need to investigate the `tidymodels` documentation to find the appropriate step functions to use.

```
titanic_recipe <- recipe(survived ~ pclass + sex + age + sib_sp + parch + fare,
                         data = train) %>%
  step_impute_linear(age, impute_with = imp_vars(all_predictors())) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_interact(terms = ~ sex_male:fare) %>%
  step_interact(terms = ~ age:fare)
```

5. Specify a logistic regression model for classification using the `"glm"` engine. Then create a workflow. Add your model and the appropriate recipe. Finally, use `fit()` to apply your workflow to the training data.

```
log_reg <- logistic_reg() %>% set_engine("glm") %>% set_mode("classification")
log_wflow <- workflow() %>% add_model(log_reg) %>% add_recipe(titanic_recipe)
log_fit <- fit(log_wflow, train)
```

Hint: Make sure to store the results of `fit()`. You'll need them later on.

6. Repeat Question 5, but this time specify a linear discriminant analysis model for classification using the `"MASS"` engine.

```
lda_mod <- discrim_linear() %>% set_mode("classification") %>% set_engine("MASS")
lda_wflow <- workflow() %>% add_model(lda_mod) %>% add_recipe(titanic_recipe)
lda_fit <- fit(lda_wflow, train)
```

7. Repeat Question 5, but this time specify a quadratic discriminant analysis model for classification using the `"MASS"` engine.

```
qda_mod <- discrim_quad() %>% set_mode("classification") %>% set_engine("MASS")
qda_wflow <- workflow() %>% add_model(qda_mod) %>% add_recipe(titanic_recipe)
qda_fit <- fit(qda_wflow, train)
```

8. Repeat Question 5, but this time specify a naive Bayes model for classification using the `"klaR"` engine. Set the `usekernel` argument to `FALSE`.

```
nb_mod <- naive_Bayes() %>% set_mode("classification") %>% set_engine("klaR") %>%
  set_args(usekernel = FALSE)
nb_wflow <- workflow() %>% add_model(nb_mod) %>% add_recipe(titanic_recipe)
nb_fit <- fit(nb_wflow, train)
```

9. Now you've fit four different models to your training data. Use `predict()` and `bind_cols()` to generate predictions using each of these 4 models and your training data. Then use the *accuracy* metric to assess the performance of each of the four models. Which model achieved the highest accuracy on the training data?

```
log_reg_predict <- predict(log_fit, new_data = train)
lda_predict <- predict(lda_fit, new_data = train)
qda_predict <- predict(qda_fit, new_data = train)
nb_predict <- predict(nb_fit, new_data = train)
```

```
predictions <- bind_cols(train %>% select(survived), log_reg_predict, lda_predict,
                         qda_predict, nb_predict)
```

```
## New names:
## * .pred_class -> .pred_class...2
## * .pred_class -> .pred_class...3
## * .pred_class -> .pred_class...4
## * .pred_class -> .pred_class...5
```

```
names(predictions) <- c("Truth", "Log_Reg", "LDA", "QDA", "NB")
predictions
```

```
## # A tibble: 712 x 5
##    Truth Log_Reg LDA   QDA   NB
##    <fct> <fct>   <fct> <fct> <fct>
##  1 No    No      No    No    No
##  2 No    No      No    No    No
##  3 No    No      No    No    No
##  4 No    No      No    No    No
##  5 No    No      No    No    No
##  6 No    Yes     Yes   No    Yes
##  7 No    No      No    No    No
##  8 No    Yes     Yes   No    No
##  9 No    No      No    No    No
## 10 No    No      No    Yes   Yes
## # ... with 702 more rows
```

```
log_reg_acc <- augment(log_fit, new_data = train) %>%
  accuracy(truth = survived, estimate = .pred_class)

lda_acc <- augment(lda_fit, new_data = train) %>%
  accuracy(truth = survived, estimate = .pred_class)

qda_acc <- augment(qda_fit, new_data = train) %>%
  accuracy(truth = survived, estimate = .pred_class)

nb_acc <- augment(nb_fit, new_data = train) %>%
  accuracy(truth = survived, estimate = .pred_class)
```

```
accuracies <- c(log_reg_acc$.estimate, lda_acc$.estimate,
                nb_acc$.estimate, qda_acc$.estimate)
models <- c("Logistic Regression", "LDA", "Naive Bayes", "QDA")
results <- tibble(accuracies = accuracies, models = models)
results %>%
  arrange(-accuracies)
```

```
## # A tibble: 4 x 2
##   accuracies models
##        <dbl> <chr>
## 1      0.805 Logistic Regression
## 2      0.801 LDA
## 3      0.789 QDA
## 4      0.788 Naive Bayes
```

10. Fit the model with the highest training accuracy to the testing data. Report the accuracy of the model on the testing data. Again using the testing data, create a confusion matrix and visualize it. Plot an ROC curve and calculate the area under it (AUC). How did the model perform? Compare its training and testing accuracies. If the values differ, why do you think this is so?

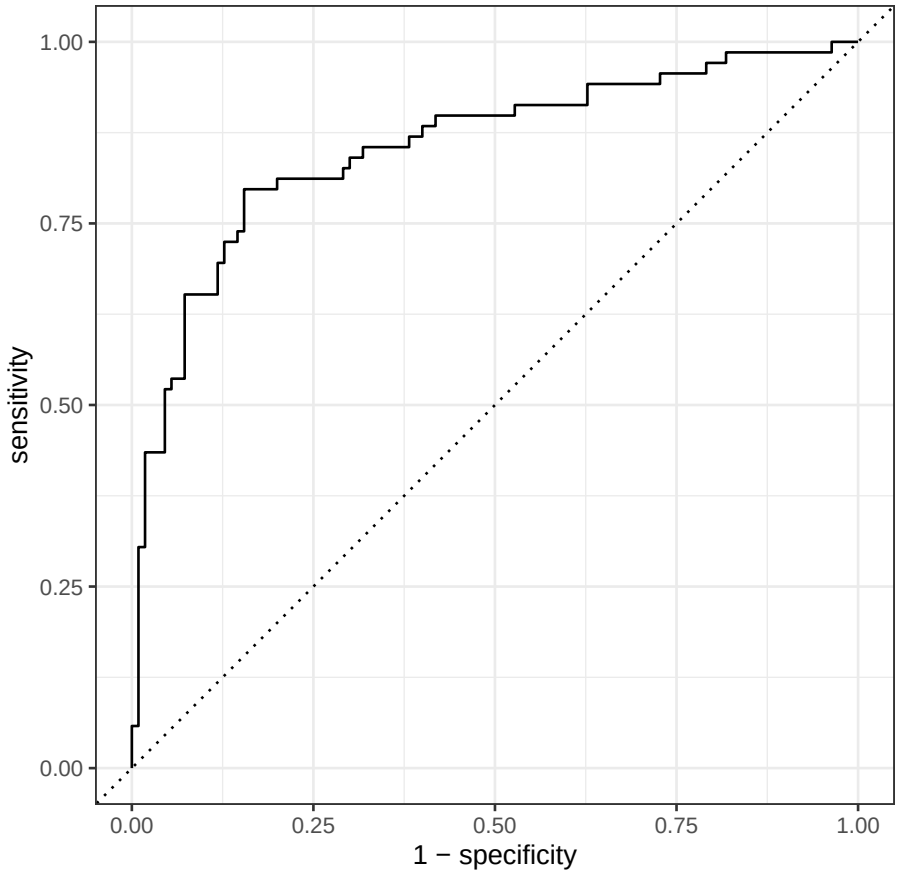
```
final_model_acc <- augment(log_fit, new_data = test) %>%
  accuracy(truth = survived, estimate = .pred_class)
final_model_acc
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.810
```

```
final_model_conf <- augment(log_fit, new_data = test) %>%
  conf_mat(truth = survived, estimate = .pred_class)
final_model_conf
```

```
##           Truth
## Prediction Yes No
##        Yes  51 16
##        No   18 94
```

```
final_model_roc <- augment(log_fit, new_data = test) %>%
  roc_curve(survived, .pred_Yes) %>%
  autoplot()
final_model_roc
```




```
final_model_predictions <- log_fit %>% predict(new_data = test, type = 'prob') %>% bind_cols(test %>% s
auc <- final_model_predictions %>% roc_auc(survived, .pred_Yes, event_level = "first")
auc
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.855
```

The model with the highest training accuracy (0.8048) was logistic regression, so I will use that as my final model. When given the the test set, the model preformed similarly but slightly better, with an accuracy of 0.8101. The confusion matrix shows that 18 False Positives and 16 False negatives. Overall, this is a relatively decent model when looking at the classification accuracy on the test set. The test set performance is slightly stronger than the train set performance, which means that we could probably benefit from more data being used to train. However, because this is a limited dataset (wont ever get bigger), we could do things like cross-validation or changing the train/test split to strengthen our training process.