Homework 3

PSTAT 131

Contents

Binary 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.

Question 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.

```
# Load the data frist
titanic_data <- read_csv("titanic.csv")

# Convert 'survived' and 'pclass' to factors
titanic_data$survived <- as.factor(titanic_data$survived)
titanic_data$pclass <- as.factor(titanic_data$pclass)

# Set a seed for reproducibility
set.seed(42)

# Split the data, stratifying on 'survived'
data_split <- initial_split(titanic_data, prop = 0.7, strata = "survived")
train_data <- training(data_split)
test_data <- testing(data_split)

# Number of observations
cat("Number of observations in training set:", nrow(train_data), "\n")
```

```
## Number of observations in training set: 623
```

```r
cat("Number of observations in testing set:", nrow(test_data), "\n")
```

```
## Number of observations in testing set: 268
```

```r
# Check for missing data in the training set
cat("Missing data in the training set:\n")
```

```
## Missing data in the training set:
```

```r
print(summarize_all(train_data, funs(sum(is.na(.)))))
```

```
## # A tibble: 1 x 12
##   passenger_id survived pclass  name   sex   age sib_sp parch ticket  fare cabin
##          <int>    <int>  <int> <int> <int> <int>  <int> <int>  <int> <int> <int>
## 1            0        0      0     0     0   122      0     0      0     0   493
## # i 1 more variable: embarked <int>
```

```r
## From the table above, in the training set, there are 122 observations' "age"
#is missing, and 493 observations' "cabin" is missing. And these missing data,
#especially those without cabin info might negatively affect the accurancy of
#the model.
```

Why is it a good idea to use stratified sampling for this data?

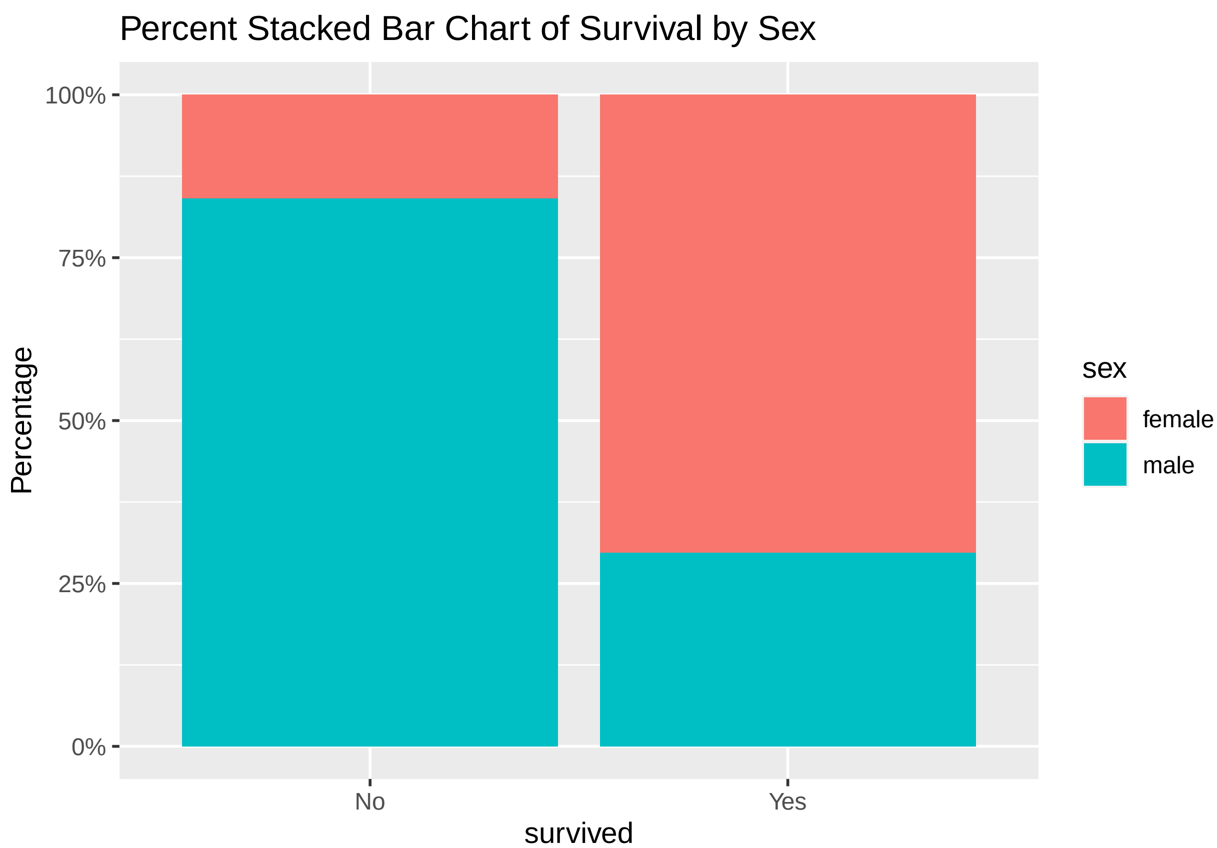
```r
## In the context of the Titanic dataset, stratified sampling is particularly
#important because the survival rate is historically low, leading to a class
#imbalance in the data. By stratifying on the 'survived' column, the model is
#exposed to a representative mix of survivors and non-survivors during both
#training and testing. This approach mitigates the risk of the model being biased
#towards predicting the majority class, thereby enhancing its predictive accuracy
#specifically for this dataset.
```

Question 2

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

Create a percent stacked bar chart (recommend using `ggplot`) with `survived` on the x-axis and `fill = sex`. Do you think `sex` will be a good predictor of the outcome?

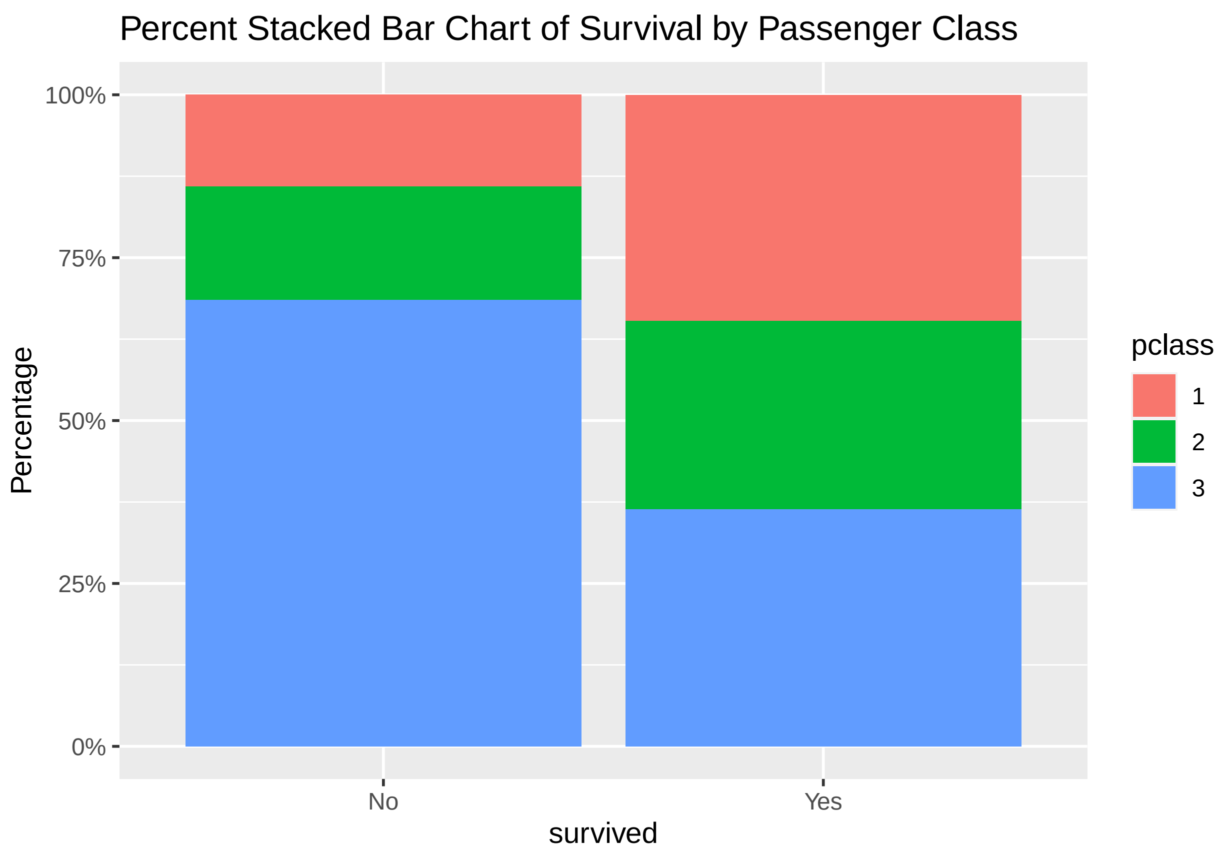
```r
# Create the percent stacked bar chart per the requirment.
ggplot(train_data, aes(x = survived, fill = sex)) +
  geom_bar(position = "fill") +
  scale_y_continuous(labels = scales::percent_format(scale = 100)) +
  labs(title = "Percent Stacked Bar Chart of Survival by Sex",
       y = "Percentage")
```




```
## From the bar chart above, among those who did not survive, a significant
#majority are male (80%). And among those who did survive, a significant majority
#are female (70%). These substantial differences in survival rates between sex
#suggest that sex is likely to be a strong predictor of survival. The model would
#benefit from including this variable. So yes, it is a good predictor.
```

Create one more percent stacked bar chart of `survived`, this time with `fill = pclass`. Do you think passenger class will be a good predictor of the outcome?

```
# Create the percent stacked bar chart for 'pclass'
ggplot(train_data, aes(x = survived, fill = pclass)) +
  geom_bar(position = "fill") +
  scale_y_continuous(labels = scales::percent_format(scale = 100)) +
  labs(title = "Percent Stacked Bar Chart of Survival by Passenger Class",
       y = "Percentage")
```




```
## From the bar chart above, Among those who did not survive, the majority were
#from the third class with 65%, followed by 20% from the second class and 15%
#from the first class. And among those who did survive, the distribution is more
#even, with 36% from the first class, 36% from the third class, and 28% from the
#second class. Specifically, passengers from the first class appear to have
#higher chances of surviving compared to those from the third class, as evidenced
#by the higher percentage of survivors in Pclass1 and the higher percentage of
#non-survivors in Pclass3. So yes, 'pclass' will be a good predictor of the
#outcome.
```

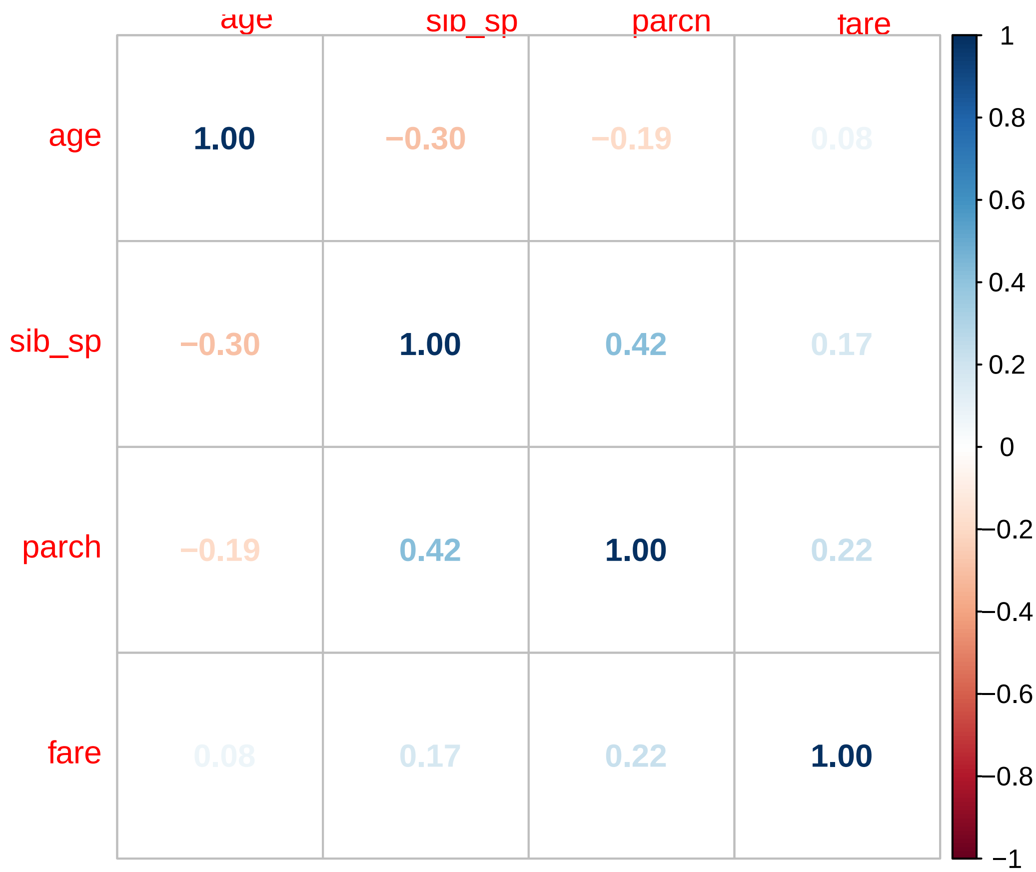
Why do you think it might be more useful to use a percent stacked bar chart as opposed to a traditional stacked bar chart?

```
## The chart normalizes the data, making it easier to see that, for example, a
#higher proportion of first-class passengers survived compared to third-class
#passengers. This format is especially useful given that the number of passengers
#varies between classes and genders, allowing for a more apples-to-apples
#comparison. In other words, percent stacked bar chart makes it easier to read
#the difference between multiple variables within one specific group.
```

Question 3

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

```r
# Select continuous variables
continuous_vars <- select(train_data, age, sib_sp, parch, fare)

# Compute the correlation matrix
cor_matrix <- cor(continuous_vars, use = "pairwise.complete.obs")

# Visualize the correlation matrix (with rotated x-axis labels to read easierly)
corrplot(cor_matrix, method = "number", tl.srt = 360)
```




```r
## Negative Correlation between sib_sp and age: In this context, it suggests that
#younger passengers are more likely to be traveling with siblings or spouses,
#which does make sense.

## Positive Correlation between sib_sp and parch: This indicates that passengers
#who are traveling with siblings or spouses are also more likely to be traveling
#with parents or children. Again, this makes sense as families are likely to
#travel together.

## Also, 'Fare' is not strongly correlated with other variables, probably because
#it is possibly more related to social or economic status, which isn't explained
#by the other continuous variables like 'parch' or 'sib_sp'.
```

Question 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.

```
# Define the recipe for Question 4
recipe_q4 <- recipe(survived ~ pclass + sex + age + sib_sp + parch + fare,
                    data = train_data) %>%
  # Impute missing values for 'age' using linear imputation
  step_impute_linear(age, impute_with= imp_vars(sib_sp)) %>%
  # Dummy encode categorical predictors
  step_dummy(all_nominal_predictors()) %>%
  # Include interaction terms
  step_interact(~ starts_with("sex") : age + age:fare)

# Print and check the recipe
recipe_q4
```

Question 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.

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

```
# Specify a logistic regression model with the 'glm' engine
logistic_spec <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

# Create a workflow combining the recipe and model specification
workflow_q5 <- workflow() %>%
  add_recipe(recipe_q4) %>%
  add_model(logistic_spec)

# Store the results of fit()
fit_workflow_q5 <- fit(workflow_q5, data = train_data)

# Display the results to check the model fit
fit_workflow_q5
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
```

```
## Model: logistic_reg()
##
## -- Preprocessor ----------------------------------------------------------------
## 3 Recipe Steps
##
## * step_impute_linear()
## * step_dummy()
## * step_interact()
##
## -- Model -----------------------------------------------------------------------
##
## Call:  stats::glm(formula = ..y ~ ., family = stats::binomial, data = data)
##
## Coefficients:
##   (Intercept)            age         sib_sp          parch           fare
##     3.1618660     -0.0153437     -0.3572281     -0.2346975     -0.0099770
##     pclass_X2      pclass_X3       sex_male  sex_male_x_age     age_x_fare
##    -0.7401441     -2.0048715     -1.0795435     -0.0641477      0.0004169
##
## Degrees of Freedom: 622 Total (i.e. Null);  613 Residual
## Null Deviance:      829.6
## Residual Deviance: 539.7     AIC: 559.7
```

Question 6

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

```r
# Specify a Linear Discriminant Analysis model with the 'MASS' engine
lda_spec <- discrim_linear() %>%
  set_engine("MASS") %>%
  set_mode("classification")

# Create a new workflow combining recipe_q4 and the new LDA model specification.
workflow_q6 <- workflow() %>%
  add_recipe(recipe_q4) %>%
  add_model(lda_spec)

# Store the results of fit() in a new variable
fit_workflow_q6 <- fit(workflow_q6, data = train_data)

# Display the results to check the model fit
fit_workflow_q6
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: discrim_linear()
##
## -- Preprocessor ----------------------------------------------------------------
## 3 Recipe Steps
##
## * step_impute_linear()
## * step_dummy()
```

```
## * step_interact()
##
## -- Model ---------------------------------------------------------------------
## Call:
## lda(..y ~ ., data = data)
##
## Prior probabilities of groups:
##        No       Yes
## 0.6163724 0.3836276
##
## Group means:
##          age    sib_sp     parch     fare pclass_X2 pclass_X3  sex_male
## No  29.68856 0.5677083 0.3567708 22.48433 0.1744792 0.6848958 0.8411458
## Yes 28.31317 0.4435146 0.4853556 46.90032 0.2887029 0.3640167 0.2970711
##     sex_male_x_age age_x_fare
## No       25.908660   665.0898
## Yes       7.923317  1439.8651
##
## Coefficients of linear discriminants:
##                          LD1
## age            -0.0045217512
## sib_sp         -0.2312120167
## parch          -0.1410511932
## fare           -0.0024362813
## pclass_X2      -0.3585050280
## pclass_X3      -1.2124231243
## sex_male       -1.1837871204
## sex_male_x_age -0.0357868566
## age_x_fare      0.0001345862
```

Question 7

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

```r
# Create a parsnip model specification for QDA
qda_spec <- discrim_quad() %>%
  set_engine('MASS') %>%
  set_mode('classification')

# Create a new workflow combining recipe_q4 and the new QDA model specification.
workflow_q7 <- workflow() %>%
  add_recipe(recipe_q4) %>%
  add_model(qda_spec)

# Store the results of fit() in a new variable
fit_workflow_q7 <- fit(workflow_q7, data = train_data)

# Display the results to check the model fit
fit_workflow_q7
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
```

```
## Model: discrim_quad()
##
## -- Preprocessor ----------------------------------------------------------------
## 3 Recipe Steps
##
## * step_impute_linear()
## * step_dummy()
## * step_interact()
##
## -- Model -----------------------------------------------------------------------
## Call:
## qda(..y ~ ., data = data)
##
## Prior probabilities of groups:
##        No       Yes
## 0.6163724 0.3836276
##
## Group means:
##          age    sib_sp     parch     fare pclass_X2 pclass_X3  sex_male
## No  29.68856 0.5677083 0.3567708 22.48433 0.1744792 0.6848958 0.8411458
## Yes 28.31317 0.4435146 0.4853556 46.90032 0.2887029 0.3640167 0.2970711
##     sex_male_x_age age_x_fare
## No       25.908660   665.0898
## Yes       7.923317  1439.8651
```

Question 8

Repeat Question 5, but this time specify a k-nearest neighbors model for classification using the "kknn" engine. Choose a value for k to try.

```r
# Specify a k-NN model with the 'kknn' engine and choose k (e.g., k = 8)
knn_spec <- nearest_neighbor(neighbors = 8) %>%
  set_engine("kknn") %>%
  set_mode("classification")

# Create a new workflow combining recipe_q4 and the new k nearest neghbors model
workflow_q8 <- workflow() %>%
  add_recipe(recipe_q4) %>%
  add_model(knn_spec)

# Store the results of fit() in a new variable
fit_workflow_q8 <- fit(workflow_q8, data = train_data)

# Display the results to check the model fit
fit_workflow_q8
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
## -- Preprocessor ----------------------------------------------------------------
## 3 Recipe Steps
##
```

```
## * step_impute_linear()
## * step_dummy()
## * step_interact()
##
## -- Model -----------------------------------------------------------------------
##
## Call:
## kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(8,     data, 5))
##
## Type of response variable: nominal
## Minimal misclassification: 0.1701445
## Best kernel: optimal
## Best k: 8
```

Question 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 metric of **area under the ROC curve** to assess the performance of each of the four models.

```r
# Generate predictions for each of the 4 models using the training data
pred_logistic <- predict(fit_workflow_q5, new_data = train_data,
                         type = "prob") %>% as_tibble()
pred_lda <- predict(fit_workflow_q6, new_data = train_data,
                    type = "prob") %>% as_tibble()
pred_qda <- predict(fit_workflow_q7, new_data = train_data,
                    type = "prob") %>% as_tibble()
pred_knn <- predict(fit_workflow_q8, new_data = train_data,
                    type = "prob") %>% as_tibble()

# Create data frames to store truth and predicted probabilities
df_logistic <- tibble(truth = as.factor(train_data$survived),
                      .pred_Yes = pred_logistic$.pred_Yes)
df_lda <- tibble(truth = as.factor(train_data$survived),
                 .pred_Yes = pred_lda$.pred_Yes)
df_qda <- tibble(truth = as.factor(train_data$survived),
                 .pred_Yes = pred_qda$.pred_Yes)
df_knn <- tibble(truth = as.factor(train_data$survived),
                 .pred_Yes = pred_knn$.pred_Yes)

# Reorder the factor levels so that "Yes" is the reference level for each
df_logistic$truth <- relevel(df_logistic$truth, ref = "Yes")
df_lda$truth <- relevel(df_lda$truth, ref = "Yes")
df_qda$truth <- relevel(df_qda$truth, ref = "Yes")
df_knn$truth <- relevel(df_knn$truth, ref = "Yes")

# Calculate AUC-ROC for each model
roc_logistic <- roc_auc(df_logistic, truth, .pred_Yes)
roc_lda <- roc_auc(df_lda, truth, .pred_Yes)
roc_qda <- roc_auc(df_qda, truth, .pred_Yes)
roc_knn <- roc_auc(df_knn, truth, .pred_Yes)
```

```r
# Display AUC-ROC for each model
print(paste("AUC-ROC for Logistic Regression: ", roc_logistic$.estimate))
```

```
## [1] "AUC-ROC for Logistic Regression:  0.861118375174337"
```

```r
print(paste("AUC-ROC for LDA: ", roc_lda$.estimate))
```

```
## [1] "AUC-ROC for LDA:  0.855604951185495"
```

```r
print(paste("AUC-ROC for QDA: ", roc_qda$.estimate))
```

```
## [1] "AUC-ROC for QDA:  0.846626569037658"
```

```r
print(paste("AUC-ROC for k-NN: ", roc_knn$.estimate))
```

```
## [1] "AUC-ROC for k-NN:  0.974301560320781"
```

Question 10

Fit all four models to your **testing** data and report the AUC of each model on the **testing** data. Which model achieved the highest AUC on the **testing** data?

```r
# Generate predictions for each of the 4 models using the testing data
pred_logistic_test <- predict(fit_workflow_q5, new_data = test_data,
                              type = "prob") %>% as_tibble()
pred_lda_test <- predict(fit_workflow_q6, new_data = test_data,
                         type = "prob") %>% as_tibble()
pred_qda_test <- predict(fit_workflow_q7, new_data = test_data,
                         type = "prob") %>% as_tibble()
pred_knn_test <- predict(fit_workflow_q8, new_data = test_data,
                         type = "prob") %>% as_tibble()

# Create data frames to store truth and predicted probabilities for testing data
df_logistic_test <- tibble(truth = as.factor(test_data$survived),
                           .pred_Yes = pred_logistic_test$.pred_Yes)
df_lda_test <- tibble(truth = as.factor(test_data$survived),
                      .pred_Yes = pred_lda_test$.pred_Yes)
df_qda_test <- tibble(truth = as.factor(test_data$survived),
                      .pred_Yes = pred_qda_test$.pred_Yes)
df_knn_test <- tibble(truth = as.factor(test_data$survived),
                      .pred_Yes = pred_knn_test$.pred_Yes)

# Reorder the factor levels so that "Yes" is the reference level for each
df_logistic_test$truth <- relevel(df_logistic_test$truth, ref = "Yes")
df_lda_test$truth <- relevel(df_lda_test$truth, ref = "Yes")
df_qda_test$truth <- relevel(df_qda_test$truth, ref = "Yes")
df_knn_test$truth <- relevel(df_knn_test$truth, ref = "Yes")

# Calculate AUC-ROC for each model on testing data
roc_logistic_test <- roc_auc(df_logistic_test, truth, .pred_Yes)
```

```r
roc_lda_test <- roc_auc(df_lda_test, truth, .pred_Yes)
roc_qda_test <- roc_auc(df_qda_test, truth, .pred_Yes)
roc_knn_test <- roc_auc(df_knn_test, truth, .pred_Yes)

# Display AUC-ROC for each model on testing data
print(paste("Test AUC-ROC for Logistic Regression: ",
            roc_logistic_test$.estimate))
```

```
## [1] "Test AUC-ROC for Logistic Regression:  0.864666078258312"
```

```r
print(paste("Test AUC-ROC for LDA: ",
            roc_lda_test$.estimate))
```

```
## [1] "Test AUC-ROC for LDA:  0.861900558987938"
```

```r
print(paste("Test AUC-ROC for QDA: ",
            roc_qda_test$.estimate))
```

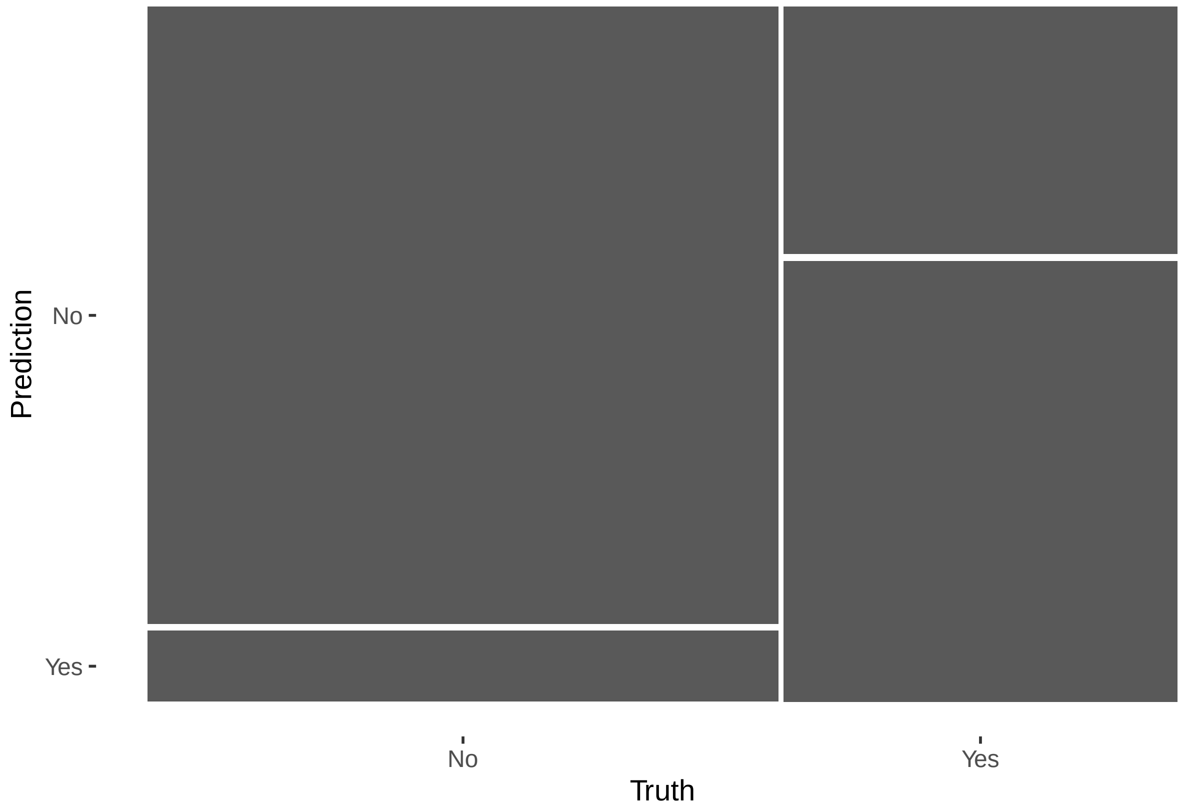
```
## [1] "Test AUC-ROC for QDA:  0.844895557516918"
```

```r
print(paste("Test AUC-ROC for k-NN: ",
            roc_knn_test$.estimate))
```

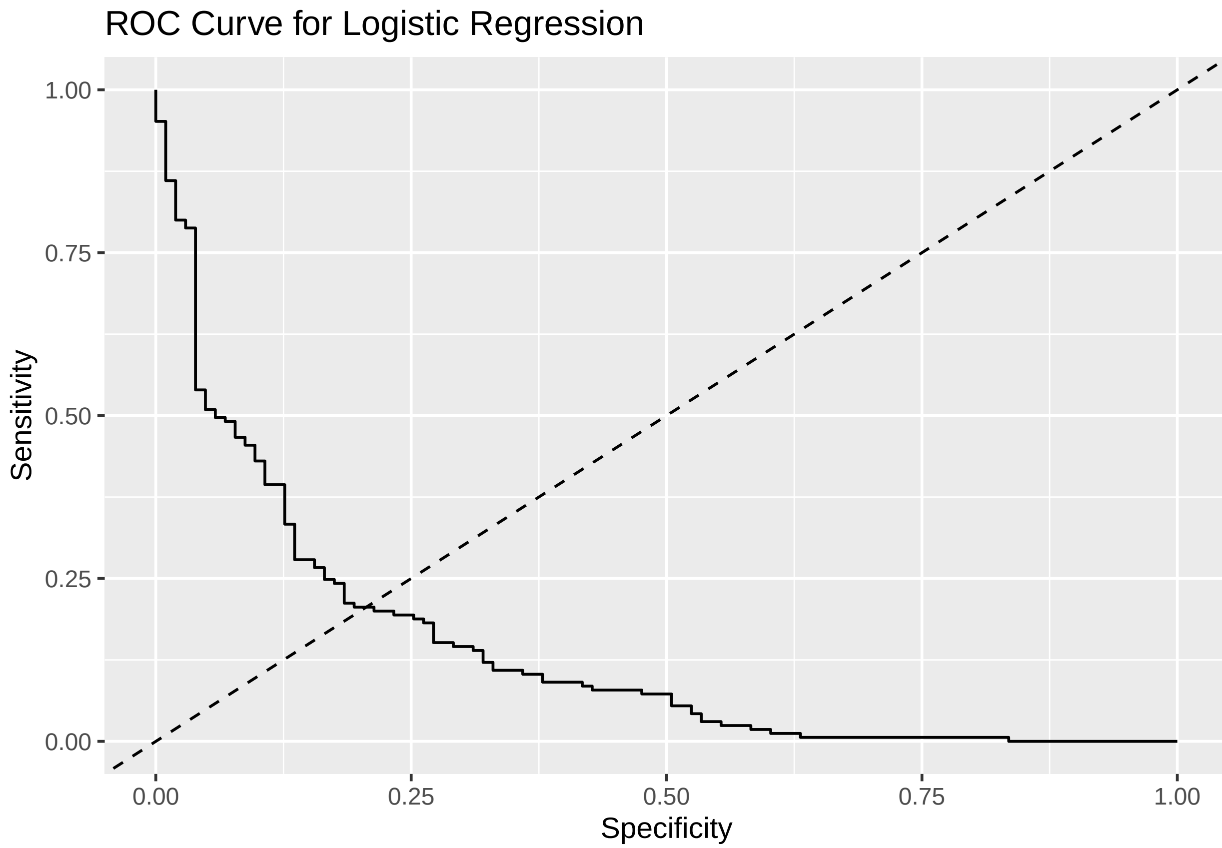
```
## [1] "Test AUC-ROC for k-NN:  0.851220947337452"
```

Using your top-performing model, create a confusion matrix and visualize it. Create a plot of its ROC curve.

```r
## Based on the output above, Logistic Regression model seems has the highest AUC-ROC performance.

# Create a new column in pred_logistic_test for predicted class labels
pred_logistic_test$.pred_class <- ifelse(pred_logistic_test$.pred_Yes > 0.5,
                                         "Yes", "No")

# Create a new tibble combining 'survived' and '.pred_class'
combined_data <- tibble(
  truth = test_data$survived,
  .pred_class = pred_logistic_test$.pred_class
)

# Convert .pred_class to a factor
combined_data$.pred_class <- as.factor(combined_data$.pred_class)

# Create the confusion matrix
confusion <- conf_mat(combined_data, truth = truth, estimate = .pred_class)

# Visualize the confusion matrix
autoplot(confusion)
```




```
# Create a new tibble combining 'survived' and predicted probabilities
roc_data <- tibble(
  truth = as.factor(test_data$survived),
  .pred_Yes = pred_logistic_test$.pred_Yes
)

# Create the ROC curve
roc_curve_data <- roc_curve(roc_data, truth, .pred_Yes)

# Plot the ROC curve using ggplot2
ggplot(roc_curve_data, aes(x =  specificity, y = sensitivity)) +
  geom_line() +
  geom_abline(linetype = "dashed") +
  labs(title = "ROC Curve for Logistic Regression",
       x = "Specificity",
       y = "Sensitivity")
```



How did your best model perform? Compare its **training** and **testing** AUC values. If the values differ, why do you think this is so?

```
## The best-performing model in the training data is k-NN, and the AUC_ROC value
#is as high as 0.974. WHile in the testing data, the best model is the Logistic
#Regression with value of approximately 0.865.

## The AUC-ROC values for both the training and testing datasets are very close
#for Logistic Regression, LDA, and QDA, suggesting good generalization. However,
#for k-NN, the AUC-ROC is significantly higher on the training data compared to
#the testing data (0.974 vs. 0.851), which could be a sign of overfitting to the
#training data.

## The Logistic Regression model, with AUC-ROC values of 0.861 on the training
#data and 0.865 on the test data, appears to be the most reliable and consistent
#model for this specific project. The small difference between the training and
#testing AUC values suggests that the model has generalized well to the unseen
#data, without signs of overfitting or underfitting.
```