Homework 4

PSTAT 131

Contents

Resampling 1

Resampling

For this assignment, we will be working with **two** of our previously used data sets – one for classification and one for regression. For the classification problem, our goal is (once again) to predict which passengers would survive the Titanic shipwreck. For the regression problem, our goal is (also once again) to predict abalone age.

Load the data from `data/titanic.csv` and `data/abalone.csv` into *R* and refresh your memory about the variables they contain using their attached codebooks.

Make sure to change `survived` and `pclass` to factors, as before, and make sure to generate the `age` variable as `rings` + 1.5!

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

Section 1: Regression (abalone age)

Question 1 Follow the instructions from Homework 2 to split the data set, stratifying on the outcome variable, `age`. You can choose the proportions to split the data into. Use k-fold cross-validation to create 5 folds from the training set.

```
# Set a seed for reproducibility
set.seed(123)

# Read the dataset
abalone_data <- read_csv("abalone.csv")

# Calculate the age (which is calculated as the number of rings + 1.5)
abalone_data <- abalone_data %>%
  # Then create a new variable for 'age'
  mutate(age = rings + 1.5)

# Perform stratified sampling to split the data into training and testing sets
data_split <- initial_split(abalone_data, prop = 0.8, strata = "age")
train_data <- training(data_split)
test_data <- testing(data_split)

# View the first few rows of both sets to confirm the split, and check 'age'
head(train_data)
```

```
## # A tibble: 6 x 10
##   type  longest_shell diameter height whole_weight shucked_weight viscera_weight
##   <chr>         <dbl>    <dbl>  <dbl>        <dbl>          <dbl>          <dbl>
## 1 M             0.35     0.265  0.09         0.226         0.0995         0.0485
## 2 I             0.33     0.255  0.08         0.205         0.0895         0.0395
## 3 I             0.355    0.28   0.085        0.290         0.095          0.0395
## 4 M             0.365    0.295  0.08         0.256         0.097          0.043
## 5 M             0.465    0.355  0.105        0.480         0.227          0.124
## 6 F             0.45     0.355  0.105        0.522         0.237          0.116
## # i 3 more variables: shell_weight <dbl>, rings <dbl>, age <dbl>
```

```
head(test_data)
```

```
## # A tibble: 6 x 10
##   type  longest_shell diameter height whole_weight shucked_weight viscera_weight
##   <chr>         <dbl>    <dbl>  <dbl>        <dbl>          <dbl>          <dbl>
## 1 F             0.53      0.42  0.135        0.677          0.256          0.142
## 2 I             0.425     0.3   0.095        0.352          0.141          0.0775
## 3 M             0.475     0.37  0.125        0.509          0.216          0.112
## 4 M             0.49      0.38  0.135        0.542          0.218          0.095
## 5 M             0.45      0.32  0.1          0.381          0.170          0.075
## 6 F             0.615     0.48  0.165        1.16           0.513          0.301
## # i 3 more variables: shell_weight <dbl>, rings <dbl>, age <dbl>
```

```
# Create 5 folds from the training set
folds <- vfold_cv(train_data, v = 5, strata = "age")

# Print the folds to check them
folds
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits             id
##   <list>             <chr>
## 1 <split [2670/670]> Fold1
## 2 <split [2672/668]> Fold2
## 3 <split [2672/668]> Fold3
## 4 <split [2673/667]> Fold4
## 5 <split [2673/667]> Fold5
```

Set up the same recipe from Homework 2.

```
# Create the recipe with manual interactions using step_mutate()
abalone_recipe <- recipe(age ~ ., data = train_data) %>%
  update_role(rings, new_role = "ID variable") %>%
  step_dummy(type, one_hot = TRUE) %>%
  step_mutate(interaction_1 = type_M * shucked_weight,
              interaction_2 = longest_shell * diameter,
              interaction_3 = shucked_weight * shell_weight) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

# Print the recipe to check it
print(abalone_recipe)
```

Question 2 In your own words, explain what we are doing when we perform k-fold cross-validation:

- What **is** k-fold cross-validation?

```
## K-fold cross-validation is a resampling technique used to assess a model's
#performance when the true error is unknown. In k-fold cross-validation, the
#original training dataset is randomly partitioned into k equally sized
#subsamples, in other words, folds.
```

- Why should we use it, rather than simply comparing our model results on the entire training set?

```
## Here are some advantages of using k-fold cross-validation:
# 1.Bias-Variance Tradeoff: It gives a more "unbiased" estimate of the model's
#performance. Training and testing on the same dataset is likely to lead to
#overfitting, where the model learns the noise in the data rather than the actual
#relationship between variables.
# 2.Better Generalization: By training the model on different subsets of the
#data, we can ensure that the model generalizes well to new data.
# 3.Resource Efficiency: It allows for efficient use of data as each observation
#is used for both training and validation, especially important if the available
#dataset is small.
```

- If we split the training set into two and used one of those two splits to evaluate/compare our models, what resampling method would we be using?

```
## "Holdout method". In this approach, the model is trained on one subset and
#validated on a separate subset that was not used during training. The holdout
#method is simpler but less robust than k-fold cross-validation because the
#evaluation may depend heavily on which observations are included in the training
#and test sets.
```

Question 3 Set up workflows for three models:

1. k-nearest neighbors with the `kknn` engine, tuning `neighbors`;
2. linear regression;
3. elastic net **linear** regression, tuning `penalty` and `mixture`.

Use `grid_regular` to set up grids of values for all of the parameters we're tuning. Use values of `neighbors` from 1 to 10, the default values of penalty, and values of mixture from 0 to 1. Set up 10 levels of each.

```
# k-NN Model Specification
knn_spec <- nearest_neighbor(neighbors = tune(), engine = "kknn") %>%
  set_mode("regression")

# k-NN Workflow
knn_wf <- workflow() %>%
  add_model(knn_spec) %>%
  add_recipe(abalone_recipe)

# k-NN Grid
knn_grid <- grid_regular(
```

```r
  neighbors(range = c(1, 10)),
  levels = 10
)

# Linear Regression Model Specification
lm_spec <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# Linear Regression Workflow
lm_wf <- workflow() %>%
  add_model(lm_spec) %>%
  add_recipe(abalone_recipe)

# Elastic Net Model Specification
enet_spec <- linear_reg(
  penalty = tune(),
  mixture = tune()
) %>%
  set_engine("glmnet") %>%
  set_mode("regression")

# Elastic Net Workflow
enet_wf <- workflow() %>%
  add_model(enet_spec) %>%
  add_recipe(abalone_recipe)

# Elastic Net Grid
enet_grid <- grid_regular(
  penalty(),
  mixture(range = c(0, 1)),
  levels = 10
)

# Check both grid
print(knn_grid)
```

```
## # A tibble: 10 x 1
##    neighbors
##        <int>
##  1         1
##  2         2
##  3         3
##  4         4
##  5         5
##  6         6
##  7         7
##  8         8
##  9         9
## 10        10
```

```
print(enet_grid)
```

```
## # A tibble: 100 x 2
##          penalty mixture
##            <dbl>   <dbl>
##  1 0.0000000001        0
##  2 0.00000000129       0
##  3 0.0000000167        0
##  4 0.000000215         0
##  5 0.00000278          0
##  6 0.0000359           0
##  7 0.000464            0
##  8 0.00599             0
##  9 0.0774              0
## 10 1                   0
## # i 90 more rows
```

How many models total, **across all folds**, will we be fitting to the **abalone data**? To answer, think about how many folds there are, how many combinations of model parameters there are, and how many models you'll fit to each fold.

```
# Number of folds
num_folds <- 5

# Number of parameter combinations for k-NN
num_knn_params <- nrow(knn_grid)

# Number of parameter combinations for Elastic Net
num_enet_params <- nrow(enet_grid)

# Linear Regression has only 1 model
num_lm_params <- 1

# Total number of models for each fold
total_models_each_fold <- num_knn_params + num_lm_params + num_enet_params

# Total number of models across all folds
total_models_all_folds <- num_folds * total_models_each_fold

# Print the total number of models across all folds
print(paste("Total number of models across all folds: ", total_models_all_folds))
```

```
## [1] "Total number of models across all folds:  555"
```

Question 4 Fit all the models you created in Question 3 to your folded data.

Suggest using ***tune_grid()****; see the documentation and examples included for help by running* ***?tune_grid****. You can also see the code in* ***Lab 4*** *for help with the tuning process.*

```
# Fit k-NN model using tune_grid()
knn_results <- tune_grid(
  knn_wf,
```

```
  resamples = folds,
  grid = knn_grid
)

# Fit Linear Regression model using fit_resamples()
lm_results <- fit_resamples(
  lm_wf,
  resamples = folds
)

# Fit Elastic Net model using tune_grid()
enet_results <- tune_grid(
  enet_wf,
  resamples = folds,
  grid = enet_grid
)

# Check the results with detailed output
collect_metrics(knn_results)
```

```
## # A tibble: 20 x 7
##    neighbors .metric .estimator  mean     n std_err .config
##        <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1         1 rmse    standard   2.98      5  0.0775 Preprocessor1_Model01
##  2         1 rsq     standard   0.313     5  0.0167 Preprocessor1_Model01
##  3         2 rmse    standard   2.76      5  0.0747 Preprocessor1_Model02
##  4         2 rsq     standard   0.360     5  0.0166 Preprocessor1_Model02
##  5         3 rmse    standard   2.60      5  0.0700 Preprocessor1_Model03
##  6         3 rsq     standard   0.401     5  0.0154 Preprocessor1_Model03
##  7         4 rmse    standard   2.50      5  0.0666 Preprocessor1_Model04
##  8         4 rsq     standard   0.428     5  0.0142 Preprocessor1_Model04
##  9         5 rmse    standard   2.43      5  0.0635 Preprocessor1_Model05
## 10         5 rsq     standard   0.447     5  0.0129 Preprocessor1_Model05
## 11         6 rmse    standard   2.39      5  0.0611 Preprocessor1_Model06
## 12         6 rsq     standard   0.461     5  0.0121 Preprocessor1_Model06
## 13         7 rmse    standard   2.36      5  0.0593 Preprocessor1_Model07
## 14         7 rsq     standard   0.472     5  0.0115 Preprocessor1_Model07
## 15         8 rmse    standard   2.33      5  0.0580 Preprocessor1_Model08
## 16         8 rsq     standard   0.480     5  0.0110 Preprocessor1_Model08
## 17         9 rmse    standard   2.32      5  0.0577 Preprocessor1_Model09
## 18         9 rsq     standard   0.487     5  0.0108 Preprocessor1_Model09
## 19        10 rmse    standard   2.30      5  0.0577 Preprocessor1_Model10
## 20        10 rsq     standard   0.493     5  0.0106 Preprocessor1_Model10
```

```
collect_metrics(lm_results)
```

```
## # A tibble: 2 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   2.18      5  0.0611 Preprocessor1_Model1
## 2 rsq     standard   0.542     5  0.0146 Preprocessor1_Model1
```

```
collect_metrics(enet_results)
```

```
## # A tibble: 200 x 8
##          penalty mixture .metric .estimator  mean     n std_err .config
##            <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.0000000001        0 rmse    standard   2.26      5  0.0659 Preprocessor1_M~
##  2 0.0000000001        0 rsq     standard   0.511     5  0.0167 Preprocessor1_M~
##  3 0.00000000129       0 rmse    standard   2.26      5  0.0659 Preprocessor1_M~
##  4 0.00000000129       0 rsq     standard   0.511     5  0.0167 Preprocessor1_M~
##  5 0.0000000167        0 rmse    standard   2.26      5  0.0659 Preprocessor1_M~
##  6 0.0000000167        0 rsq     standard   0.511     5  0.0167 Preprocessor1_M~
##  7 0.000000215         0 rmse    standard   2.26      5  0.0659 Preprocessor1_M~
##  8 0.000000215         0 rsq     standard   0.511     5  0.0167 Preprocessor1_M~
##  9 0.00000278          0 rmse    standard   2.26      5  0.0659 Preprocessor1_M~
## 10 0.00000278          0 rsq     standard   0.511     5  0.0167 Preprocessor1_M~
## # i 190 more rows
```

```
# Check for notes or warnings
show_notes(knn_results)
```

```
## Great job! No notes to show.
```

```
show_notes(lm_results)
```

```
## unique notes:
## ----------------------------------------------------
## prediction from a rank-deficient fit may be misleading
```

```
show_notes(enet_results)
```

```
## Great job! No notes to show.
```

Question 5 Use `collect_metrics()` to print the mean and standard errors of the performance metric ***root mean squared error (RMSE)*** for each model across folds.

Decide which of the models has performed the best. Explain how/why you made this decision. Note that each value of the tuning parameter(s) is considered a different model; for instance, KNN with $k = 4$ is one model, KNN with $k = 2$ another.

```
# Collect metrics for k-NN
knn_metrics <- collect_metrics(knn_results)

# Collect metrics for Linear Regression
lm_metrics <- collect_metrics(lm_results)

# Collect metrics for Elastic Net
enet_metrics <- collect_metrics(enet_results)

# For k-NN: filter for RMSE and sort the models by RMSE for each fold
best_knn <- knn_metrics %>%
```

```
  filter(.metric == "rmse") %>%
  arrange(mean) %>%
  slice_head(n = 1)

# For Elastic Net: filter for RMSE and sort the models by RMSE for each fold
best_enet <- enet_metrics %>%
  filter(.metric == "rmse") %>%
  arrange(mean) %>%
  slice_head(n = 1)

# For Linear Regression: Since there's only one model, take the RMSE as is
best_lm <- lm_metrics %>%
  filter(.metric == "rmse") %>%
  summarise(mean_rmse = mean(mean), std_err_rmse = mean(std_err))

# Print best models for each fold for k-NN and Elastic Net
print(best_knn)
```

```
## # A tibble: 1 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        10 rmse    standard    2.30     5  0.0577 Preprocessor1_Model10
```

```
print(best_enet)
```

```
## # A tibble: 1 x 8
##        penalty mixture .metric .estimator  mean     n std_err .config
##          <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.0000000001   0.111 rmse    standard    2.18     5  0.0619 Preprocessor1_Mod~
```

```
# Print RMSE for Linear Regression
print(best_lm)
```

```
## # A tibble: 1 x 2
##   mean_rmse std_err_rmse
##       <dbl>        <dbl>
## 1      2.18       0.0611
```

```
## Based on the output above, the best k-NN model has 10 neighbors and an RMSE of
#approximately 2.300986 with a standard error of 0.05772904. The best Elastic Net #model has a penalty
#2.182027 with a standard error of 0.06192046. And the Linear Regression
#model has an RMSE of 2.182331 with a standard error of 0.06114094.

## Based on the RMSE values, both the Elastic Net and Linear Regression models
#have nearly the same performance, and they are slightly better than the best
#k-NN model. However, since Linear Regression performs almost as well as Elastic
#Net but is simpler, it may be considered the best model for this specific
#problem based on Occam's razor principle, which suggests that among models with
#similar performance, the simpler one is often preferable.
```

Question 6 Use `finalize_workflow()` and `fit()` to fit your chosen model to the entire **training set**.

Lastly, use `augment()` to assess the performance of your chosen model on your **testing set**. Compare your model's **testing** RMSE to its average RMSE across folds.

```
# Fit the workflow to the entire training set
final_lm_fit <- fit(lm_wf, data = train_data)

# Assess the performance on the testing set
test_results <- augment(final_lm_fit, new_data = test_data)

# Calculate RMSE on the testing set
test_rmse <- test_results %>%
  metrics(truth = age, estimate = .pred) %>%
  filter(.metric == "rmse")

# Print the RMSE on the testing set
print(test_rmse)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        2.19
```

```
## From question 5, we have managed to obtain RMSE for linear regression model,
#which is 2.182331. And from the output above, we have the RMSE on the testing
#set is 2.186413. By observing two RMSE values, the RMSE for the testing set is
#very close to the average RMSE obtained from the cross-validation. This is a
#good sign, indicating that the model generalizes well to new, unseen data.
```

Section 2: Classification (Titanic survival)

Question 7 Follow the instructions from Homework 3 to split the data set, stratifying on the outcome variable, `survived`. You can choose the proportions to split the data into. Use k-fold cross-validation to create 5 folds from the training set.

```
# Load the Titanic data
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)

# Perform stratified sampling to split the data into training and testing sets
data_split_titanic <- initial_split(titanic_data, prop = 0.7,
                                    strata = "survived")
train_data_titanic <- training(data_split_titanic)
test_data_titanic <- testing(data_split_titanic)

# Create 5 folds from the training set
```

```r
folds_titanic <- vfold_cv(train_data_titanic, v = 5, strata = "survived")

# Print the folds to check them
print(folds_titanic)
```

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits            id
##   <list>            <chr>
## 1 <split [498/125]> Fold1
## 2 <split [498/125]> Fold2
## 3 <split [498/125]> Fold3
## 4 <split [498/125]> Fold4
## 5 <split [500/123]> Fold5
```

Question 8 Set up the same recipe from Homework 3 – but this time, add `step_upsample()` so that there are equal proportions of the `Yes` and `No` levels (you'll need to specify the appropriate function arguments). *Note: See Lab 5 for code/tips on handling imbalanced outcomes.*

```r
# Define the recipe for the Titanic dataset
recipe_titanic <- recipe(survived ~ pclass + sex + age + sib_sp + parch + fare,
                         data = train_data_titanic) %>%
  # Upsample to balance the classes in 'survived'
  step_upsample(survived, over_ratio = 1) %>%
  # 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_titanic
```

Question 9 Set up workflows for three models:

1. k-nearest neighbors with the `kknn` engine, tuning `neighbors`;
2. logistic regression;
3. elastic net **logistic** regression, tuning `penalty` and `mixture`.

Set up the grids, etc. the same way you did in Question 3. Note that you can use the same grids of parameter values without having to recreate them.

```r
# k-NN Model Specification for Titanic
knn_spec_titanic <- nearest_neighbor(neighbors = tune(),
                                     engine = "kknn") %>%
  set_mode("classification")

# k-NN Workflow for Titanic
knn_wf_titanic <- workflow() %>%
  add_model(knn_spec_titanic) %>%
```

```r
  add_recipe(recipe_titanic)

# Logistic Regression Model Specification for Titanic
logreg_spec_titanic <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

# Logistic Regression Workflow for Titanic
logreg_wf_titanic <- workflow() %>%
  add_model(logreg_spec_titanic) %>%
  add_recipe(recipe_titanic)

# Elastic Net Logistic Regression Model Specification for Titanic
enet_spec_titanic <- logistic_reg(
  penalty = tune(),
  mixture = tune()
) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

# Elastic Net Workflow for Titanic
enet_wf_titanic <- workflow() %>%
  add_model(enet_spec_titanic) %>%
  add_recipe(recipe_titanic)

# k-NN Grid for Titanic
knn_grid_titanic <- grid_regular(
  neighbors(range = c(1, 10)),
  levels = 10
)

# Elastic Net Grid for Titanic
enet_grid_titanic <- grid_regular(
  penalty(),
  mixture(range = c(0, 1)),
  levels = 10
)

# Check both grids to ensure they are correctly set up
print(knn_grid_titanic)
```

```
## # A tibble: 10 x 1
##    neighbors
##        <int>
##  1         1
##  2         2
##  3         3
##  4         4
##  5         5
##  6         6
##  7         7
##  8         8
##  9         9
```

```
## 10        10
```

```
print(enet_grid_titanic)
```

```
## # A tibble: 100 x 2
##          penalty mixture
##            <dbl>   <dbl>
##  1 0.0000000001        0
##  2 0.00000000129       0
##  3 0.0000000167        0
##  4 0.000000215         0
##  5 0.00000278          0
##  6 0.0000359           0
##  7 0.000464            0
##  8 0.00599             0
##  9 0.0774              0
## 10 1                   0
## # i 90 more rows
```

Question 10 Fit all the models you created in Question 9 to your folded data.

```
# Fit k-NN Model
knn_results_titanic <- tune_grid(
  object = knn_wf_titanic,
  resamples = folds_titanic,
  grid = knn_grid_titanic
)

# Fit Logistic Regression Model
logreg_results_titanic <- tune_grid(
  object = logreg_wf_titanic,
  resamples = folds_titanic
)

# Fit Elastic Net Model
enet_results_titanic <- tune_grid(
  object = enet_wf_titanic,
  resamples = folds_titanic,
  grid = enet_grid_titanic
)

# Check the results with detailed output
collect_metrics(knn_results_titanic)
```

```
## # A tibble: 20 x 7
##    neighbors .metric  .estimator  mean     n std_err .config
##        <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1         1 accuracy binary     0.777     5  0.0293 Preprocessor1_Model01
##  2         1 roc_auc  binary     0.758     5  0.0298 Preprocessor1_Model01
##  3         2 accuracy binary     0.777     5  0.0288 Preprocessor1_Model02
##  4         2 roc_auc  binary     0.791     5  0.0361 Preprocessor1_Model02
##  5         3 accuracy binary     0.780     5  0.0277 Preprocessor1_Model03
```

```
##  6         3 roc_auc  binary     0.809     5  0.0305 Preprocessor1_Model03
##  7         4 accuracy binary     0.779     5  0.0267 Preprocessor1_Model04
##  8         4 roc_auc  binary     0.815     5  0.0269 Preprocessor1_Model04
##  9         5 accuracy binary     0.790     5  0.0319 Preprocessor1_Model05
## 10         5 roc_auc  binary     0.824     5  0.0259 Preprocessor1_Model05
## 11         6 accuracy binary     0.790     5  0.0319 Preprocessor1_Model06
## 12         6 roc_auc  binary     0.824     5  0.0223 Preprocessor1_Model06
## 13         7 accuracy binary     0.788     5  0.0328 Preprocessor1_Model07
## 14         7 roc_auc  binary     0.828     5  0.0236 Preprocessor1_Model07
## 15         8 accuracy binary     0.795     5  0.0314 Preprocessor1_Model08
## 16         8 roc_auc  binary     0.828     5  0.0235 Preprocessor1_Model08
## 17         9 accuracy binary     0.790     5  0.0316 Preprocessor1_Model09
## 18         9 roc_auc  binary     0.829     5  0.0223 Preprocessor1_Model09
## 19        10 accuracy binary     0.798     5  0.0257 Preprocessor1_Model10
## 20        10 roc_auc  binary     0.831     5  0.0218 Preprocessor1_Model10
```

```
collect_metrics(logreg_results_titanic)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.785     5  0.0225 Preprocessor1_Model1
## 2 roc_auc  binary     0.849     5  0.0216 Preprocessor1_Model1
```

```
collect_metrics(enet_results_titanic)
```

```
## # A tibble: 200 x 8
##          penalty mixture .metric  .estimator  mean     n std_err .config
##            <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.0000000001        0 accuracy binary     0.794     5  0.0206 Preprocessor1_~
##  2 0.0000000001        0 roc_auc  binary     0.846     5  0.0241 Preprocessor1_~
##  3 0.00000000129       0 accuracy binary     0.794     5  0.0206 Preprocessor1_~
##  4 0.00000000129       0 roc_auc  binary     0.846     5  0.0241 Preprocessor1_~
##  5 0.0000000167        0 accuracy binary     0.794     5  0.0206 Preprocessor1_~
##  6 0.0000000167        0 roc_auc  binary     0.846     5  0.0241 Preprocessor1_~
##  7 0.000000215         0 accuracy binary     0.794     5  0.0206 Preprocessor1_~
##  8 0.000000215         0 roc_auc  binary     0.846     5  0.0241 Preprocessor1_~
##  9 0.00000278          0 accuracy binary     0.794     5  0.0206 Preprocessor1_~
## 10 0.00000278          0 roc_auc  binary     0.846     5  0.0241 Preprocessor1_~
## # i 190 more rows
```

```
# Check for notes or warnings
show_notes(knn_results_titanic)
```

```
## Great job! No notes to show.
```

```
show_notes(logreg_results_titanic)
```

```
## Great job! No notes to show.
```

```r
show_notes(enet_results_titanic)
```

```
## Great job! No notes to show.
```

Question 11 Use `collect_metrics()` to print the mean and standard errors of the performance metric ***area under the ROC curve*** for each model across folds.

Decide which of the models has performed the best. Explain how/why you made this decision.

```r
# Collect metrics for k-NN
knn_metrics_titanic <- collect_metrics(knn_results_titanic)

# Collect metrics for Logistic Regression
logreg_metrics_titanic <- collect_metrics(logreg_results_titanic)

# Collect metrics for Elastic Net
enet_metrics_titanic <- collect_metrics(enet_results_titanic)

# For k-NN: filter for AUC and sort the models by AUC for each fold
best_knn_titanic <- knn_metrics_titanic %>%
  filter(.metric == "roc_auc") %>%
  arrange(mean) %>%
  slice_head(n = 1)

# For Elastic Net: filter for AUC and sort the models by AUC for each fold
best_enet_titanic <- enet_metrics_titanic %>%
  filter(.metric == "roc_auc") %>%
  arrange(mean) %>%
  slice_head(n = 1)

# For Logistic Regression: Since there's only one model, take the AUC as is
best_logreg_titanic <- logreg_metrics_titanic %>%
  filter(.metric == "roc_auc") %>%
  summarise(mean_auc = mean(mean), std_err_auc = mean(std_err))

# Print best models for each fold for k-NN and Elastic Net
print(best_knn_titanic)
```

```
## # A tibble: 1 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1         1 roc_auc binary     0.758     5  0.0298 Preprocessor1_Model01
```

```r
print(best_enet_titanic)
```

```
## # A tibble: 1 x 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1       1   0.333 roc_auc binary       0.5     5       0 Preprocessor1_Model040
```

```r
# Print AUC for Logistic Regression
print(best_logreg_titanic)
```

```
## # A tibble: 1 x 2
##   mean_auc std_err_auc
##      <dbl>       <dbl>
## 1    0.849      0.0216
```

```r
## Based on the output above, we know that The k-NN model has an AUC of 0.7700835
#with a standard error of 0.02847453. The Elastic Net model has an AUC of 0.5
#with a standard error of 0, which suggests that it performed no better than
#random guessing. And the Logistic Regression model has an AUC of 0.8525593 with
#a standard error of 0.02295994.

## The model with the highest AUC is the Logistic Regression model, with an AUC
#of about 0.8526. A higher AUC indicates better performance, so in this case,
#with a relatively low standard error of approximately 0.023, the Logistic
#Regression model is the best-performing model among the three.
```

Question 12 Use `finalize_workflow()` and `fit()` to fit your chosen model to the entire **training set**.

Lastly, use `augment()` to assess the performance of your chosen model on your **testing set**. Compare your model's **testing** ROC AUC to its average ROC AUC across folds.

```r
# Extract the best parameters
best_params <- logreg_results_titanic %>%
  select_best(metric = "roc_auc")

# Finalize the workflow with the best parameters
final_logreg_wf_titanic <- finalize_workflow(logreg_wf_titanic, best_params)

# Fit the finalized workflow to the entire training set
final_logreg_fit_titanic <- fit(final_logreg_wf_titanic,
                                data = train_data_titanic)

# Get the predictions on the testing set
test_results_titanic <- augment(final_logreg_fit_titanic,
                                new_data = test_data_titanic)

# Reorder the factor levels so that "Yes" is the reference level
test_results_titanic$survived <- relevel(test_results_titanic$survived,
                                         ref = "Yes")

# Calculate ROC AUC on the testing set
test_roc_auc_titanic <- test_results_titanic %>%
  roc_auc(truth = survived, .pred_Yes)

# Print the ROC AUC on the testing set
print(test_roc_auc_titanic)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
```

```
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.858
```

```
## From the result above, we have the AUC value is 0.8610768 on the testing set.
#And in Q11, we have the AUC value for Logistic regression model is 0.8525593.
#Both of these AUC values are close to each other and relatively high, indicating
#a good predictive performance.
```