Using Animal Shelter Data to Predict Animal Survival Outcomes

Peter Loyland, Henry Gliedman, Matthew Sentieri

5/13/25

Introduction

Over 60,000 animals a year enter animal shelters in the Los Angeles Metro area, according to the LA department of Animal Services. With that many animals entering shelters each year, it becomes increasingly important to understand how to properly care for as many animals as possible with a limited budget. The Long Beach, CA animal shelter dataset allows us to begin to model problems, and inform decisions for animal shelter workers. The dataset contains information regarding the type of animals in the shelter, their physical health and characteristics, as well as the outcome of their time at the shelter. The question we want to ask is how can we model and predict the outcome of an animal's time at the shelter. More specifically, we want to predict if the animal will survive their stay.

The first step in our statistical analysis is data cleaning. We removed 7 variables from the dataset for varying reasons. Many of them were repetitive, for example 'outcome_is_dead' is giving the same information as 'was_outcome_alive' but in different wording. Others variables, such as 'animal_id' were deemed trivial for our model. We removed a number of columns which had a high proportion of NA values. In addition, we created a new dummy variable 'has_name', which returns a 1 if the animal has a name, and a 0 if not, allowing us to quantify the value of an animal getting a name. A days_at_shelter variable was also created, acting to show the length of an animals stay in Long Beach.

The explanatory variables in our analysis are shown below. A large majority of our variables are categorical. A table of proportions is available in the appendix.

Table 1: Variable Summary

Variable	Role	Type
animal_type	Explanatory	Categorical
primary_color	Explanatory	Categorical
secondary_color	Explanatory	Categorical
sex	Explanatory	Categorical
age	Explanatory	Numerical
intake_condition	Explanatory	Categorical
intake_type	Explanatory	Categorical
intake_subtype	Explanatory	Categorical
days_at_shelter	Explanatory	Numerical
jurisdiction	Explanatory	Categorical
latitude	Explanatory	Numerical
longitude	Explanatory	Numerical
was_outcome_alive	Response	Categorical
has_name	Explanatory	Categorical

Explanitory Data Analysis

The next step to our statistic analysis of the Long Beach Animal shelter is some pre modeling explanatory data analysis. This will inform our modeling decisions and model interpretations going forward. First some univariate analysis is useful to understand our important variables better. To observe our numeric explanatory variables, it is helpful to create a table of summary statistics. This table will show the range and relative distribution of both the age and days at shelter variable.

Table 2: Numerical Summaries

variable	max	min	mean	median	sd
days_at_shelter	1082	1.00e-07	22.75115	7	47.20917
age	10957	-3.63e+03	944.18889	366	1188.42178

Based on the numeric summaries, some right skew seems possible. Figure 1 displays the distributions of these variables, showing the assumed right skew. Figure 1 also displays the distribution of our response variable of survival.

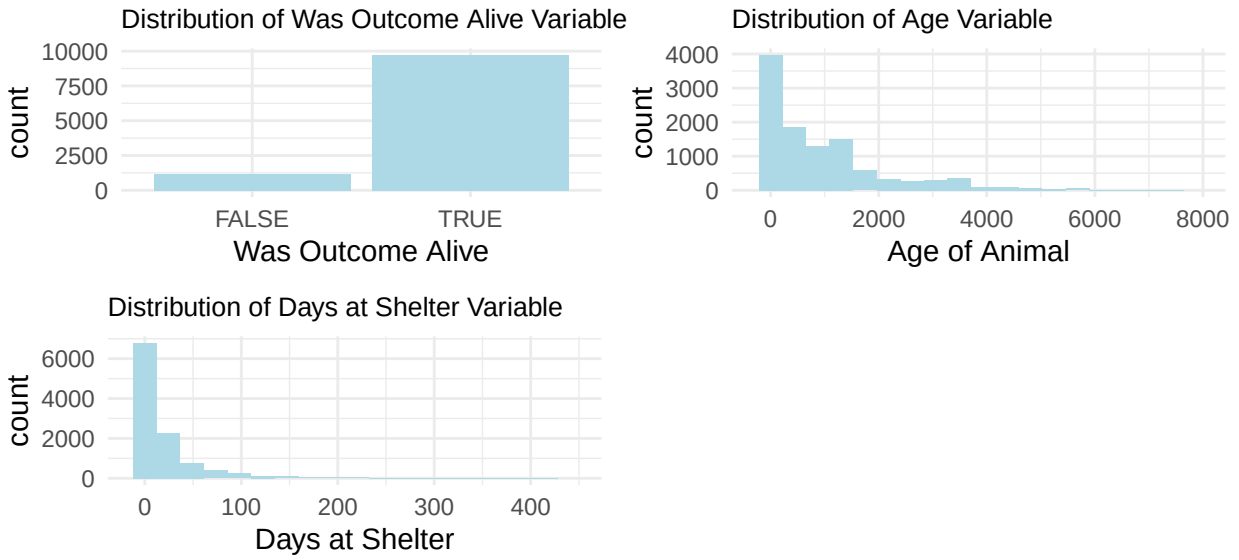

Figure 1: Figures of Univariate Analysis

Next, we would like to plot some relationships between our response variable `was_outcome_alive` and some explanatory variables in our parameter set, seeing if there is any obvious correlations. Figure 3 shows the relationship between a pet being named and it's survival, displaying that named animals are more likely to survive their time in the shelter. We also looked at the relationship between days at shelter and survival. It appears that animals in the shelter longer, have a higher likelihood of survival. This may be because if an animal is sick, injured, or seen as a lost cause by the shelter, they may opt to euthanize the animal soon after arrival.

Model Building

Now that our dataset has been explored, we continue in our analysis to the model construction process. In model construction, we are going to utilize a number of model specifications, and a series of different tuning processes to determine the best possible way to model survival rate. The first of these specifications

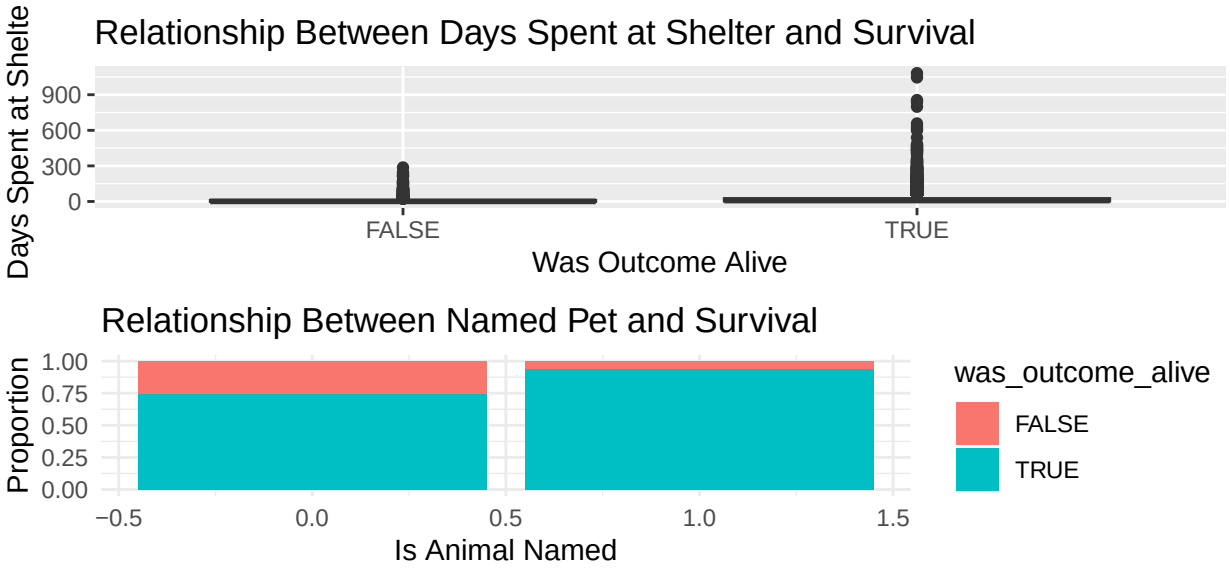

Figure 2: Multivariable EDA With Was Outcome Alive Response Variable

is LASSO. To set the recipe for this model we used a number of step functions. We used a step_novel and step_other to find and group any variable levels together with fewer than 5% of the population makeup. A step_zv() function is used to remove any parameters of zero variance, and a step log was used to log both days and age. That log will create a better variable distribution for the LASSO model. We handled our NA values using a step_impute_median for numeric variables and step_impute_mode for categorical, replacing any NA values with the variables corresponding metric. Lastly, we normalized all of our numeric variables, and dummied all categorical predictors. We used 5 fold cross validation to tune our penalty value, finding the optimal penalty value to be .000464.

LASSO MODEL

```
set.seed(12283)

shelter_split <- initial_split(shelter_clean, prop = 0.8, strata = was_outcome_alive)
train_data <- training(shelter_split)
test_data <- testing(shelter_split)

suppressWarnings({
  shelter_recipe <- recipe(was_outcome_alive ~ ., data = train_data) %>%
    step_novel(all_nominal_predictors()) %>%
    step_other(threshold = .05) %>%
    step_zv(all_predictors()) %>%
    step_log(age, days_at_shelter) %>%
    step_impute_median(all_numeric_predictors()) %>%
    step_impute_mode(all_nominal_predictors()) %>%
    step_dummy(all_nominal_predictors()) %>%
    step_normalize(all_numeric_predictors(), na_rm = TRUE)

  lasso_spec <- logistic_reg(penalty = tune(), mixture = 1) %>%
    set_engine("glmnet") %>%
    set_mode("classification")
```

```
  lasso_workflow <- workflow() %>%
    add_recipe(shelter_recipe) %>%
    add_model(lasso_spec)

  shelter_folds <- vfold_cv(train_data, v = 5, strata = was_outcome_alive)
  lasso_grid <- grid_regular(penalty(), levels = 10)

  lasso_tune <- tune_grid(
    lasso_workflow,
    resamples = shelter_folds,
    grid = lasso_grid,
    control = control_grid(save_pred = TRUE)
  )
})
```

```
## > A | warning: NaNs produced, !  The following columns have zero variance so scaling cannot be used:
##                   animal_type_new, primary_color_new, secondary_color_new, sex_new,
##                   intake_condition_new, intake_type_new, intake_subtype_new, and
##                   jurisdiction_new.
##                 i Consider using ?step_zv (‘?recipes::step_zv()‘) to remove those columns
##                   before normalizing.

## There were issues with some computations   A: x1                                                    > B
## There were issues with some computations   A: x1There were issues with some computations   A: x1   B
```

```
best_lambda <- select_best(lasso_tune)

final_lasso_workflow <- finalize_workflow(lasso_workflow, best_lambda)

final_lasso_fit <- fit(final_lasso_workflow, data = train_data)

lasso_preds <- predict(final_lasso_fit, new_data = test_data, type = "prob")

test_results <- test_data %>%
  select(was_outcome_alive) %>%
  bind_cols(lasso_preds)


lasso_roc <- roc_curve(test_results, truth = was_outcome_alive, ".pred_TRUE")
```

##Cart Model

The next model specification we used was the CART model (Classification and Regression Tree). This model required a lot of the same steps as our Lasso model. However, our recipe was much simpler as regression trees handle compex variables much better. The only step functions needed were the impute median to deal with NA values of numeric parameters, and normalization of our numeric variables. Here we used 10 fold cross validation to tune both tree depth and cost complexity.

```
set.seed(12283)

cart_recipe <- recipe(was_outcome_alive ~ ., data = train_data) %>%
  step_impute_median(all_numeric_predictors()) %>%
```

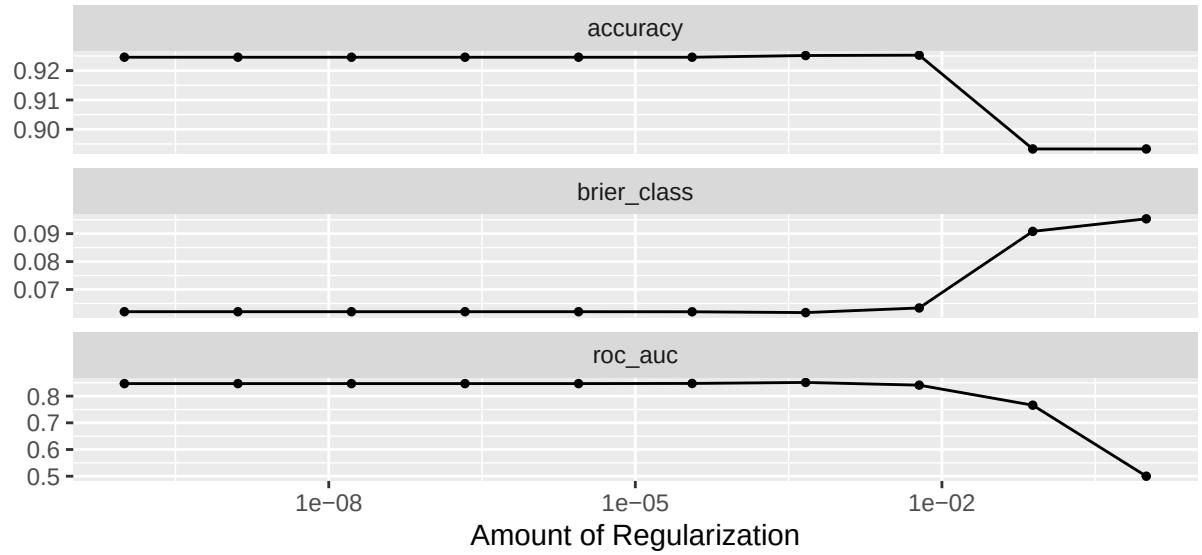

Figure 3: Figure Displaying Optimal Penalty for LASSO Model

```
  step_normalize(all_numeric_predictors())

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

cart_cv <- vfold_cv(train_data, v = 10)

cart_workflow <- workflow() %>%
  add_model(cart_spec) %>%
  add_recipe(cart_recipe)

cart_grid <- grid_regular(tree_depth(), cost_complexity(), levels = 5)

cart_tuned <- tune_grid(cart_workflow, resamples = cart_cv, grid = cart_grid, metrics = metric_set(roc_

best_cart <- select_best(cart_tuned)

final_cart_workflow <- finalize_workflow(cart_workflow, best_cart)

cart_fit <- fit(final_cart_workflow, data = train_data)

cart_probs <- predict(cart_fit, test_data, type = "prob")
cart_pred <- predict(cart_fit, test_data)

cart_roc <- roc_curve(data.frame(
  was_outcome_alive = test_data$was_outcome_alive,
  .pred_TRUE = cart_probs$.pred_TRUE
), was_outcome_alive, .pred_TRUE)
```

XGBoost Model

Our final model specification is an XGBoost model. XGBoost uses similar decision tree principles as the cart, but utilizes iterative learning to create tree models that learn from the mistakes of prior trees. Here we used the same recipe as the Cart model but with an added step of imputing categorical variables, and making them dummy variables. We 5 fold cross validated to tune our number of trees, learning rate, and tree depth.

```
xgb_recipe<- recipe(was_outcome_alive ~ ., data = train_data) %>%
  step_impute_median(all_numeric_predictors()) %>%
   step_impute_mode(all_nominal_predictors())%>%
  step_other(all_nominal_predictors(), threshold = 0.10)%>%
  step_dummy(all_nominal_predictors())%>%
  step_normalize(all_numeric_predictors())

xgb_spec <-
    boost_tree(trees= tune(), learn_rate= tune(), tree_depth= tune()) |>
    set_mode("classification") |>
    set_engine("xgboost")

xgb_cv <- vfold_cv(train_data, v = 5)

xgb_workflow <- workflow() %>%
  add_model(xgb_spec) %>%
  add_recipe(xgb_recipe)

xgb_grid <- grid_regular(trees(range = c(50, 200)), learn_rate(), tree_depth(range = c(2, 5)), levels =

xgb_tuned <- tune_grid(xgb_workflow, resamples = xgb_cv, grid = xgb_grid, metrics = metric_set(roc_auc)


best_xgb <- select_best(xgb_tuned)


final_xgb_workflow <- finalize_workflow(xgb_workflow, best_xgb)

xgb_fit <- fit(final_xgb_workflow, data = train_data)


xgb_probs <- predict(xgb_fit, test_data, type = "prob")
xgb_pred <- predict(xgb_fit, test_data)

xgb_roc <- roc_curve(data.frame(
  was_outcome_alive = test_data$was_outcome_alive,
  .pred_TRUE = xgb_probs$.pred_TRUE
), was_outcome_alive, .pred_TRUE)
```

Below is a table of model assessment metrics for each of our initial specifications. These model metrics include accuracy, sensitivity or true positive rate, and specificity or true negative rate. These metrics allow us to analyze the success of each model. All three of our model performed similarly, with just a 1.2% accuracy difference between our best and worst model. A full table of all model metrics is shown below.

Table 3: Performance Metrics for Initial Models

Model	Sensitivity	Specificity	Accuracy
Lasso Model	0.989	0.408	0.926
Cart Model	0.977	0.489	0.925
XG Boost Model	0.994	0.459	0.937

We will now select our top two models by accuracy, Lasso Model and XGBoost, to perform model refinement on.

Model Refinement

Now that initial models have been created, we can begin to refine our better models. The goal with these models refinements is to retain predictive accuracy while making the models simpler. Vip or variable importance plots can be useful in finding our most important variables in the model.

```
p1 <- vip(final_lasso_fit, num_features = 5) + labs(title= "Lasso VIP") + theme_minimal()
p3 <- vip(xgb_fit, num_features = 5)+labs(title= "XGBoost VIP") + theme_minimal()
grid.arrange(p1, p3, nrow = 1)
```

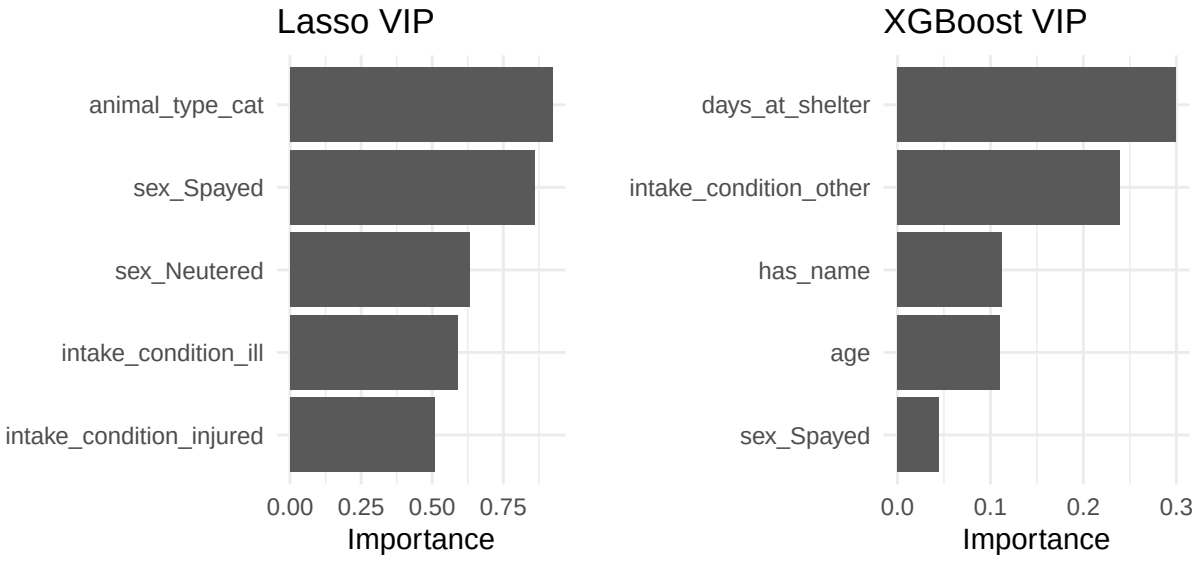

Figure 4: Variable Importance Plots

LASSO

To refine our Lasso model, we used a very similar recipe to before. However, we removed the has_name, animal_type, sex, and age variables from the recipe. This will remove any obviously correlated variables, and again create a potentially more informative model while retaining a level of predictive accuracy. We utilized the same system of cross validation as before. The full code for the refined lasso is in the appendix. Some model metrics are shown below.

Here is the model metrics set for the refined Lasso.

```
## # A tibble: 3 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.895
## 2 sens     binary         0.994
## 3 spec     binary         0.0644
```

Refined XGBoost

We refined our XGBoost model in a very similar way. However, in this recipe we only removed intake condition, and age, as we felt they would be pretty obvious predictors of an animals health anywhere. We again used the same cross validation process. The code for this refined model is also available in the appendix, with some model metrics shown below.

Here is the model metrics set for the refined XGBoost.

```
## # A tibble: 3 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.916
## 2 sens     binary         0.987
## 3 spec     binary         0.322
```

Comparison

Now that all of the model have been created, comparison should be done to determine the best one. The first step in our model comparison is creating an Roc-Auc plot with the refined and unrefined XGBoost and LASSO. The plot is shown below with code in the appendix.

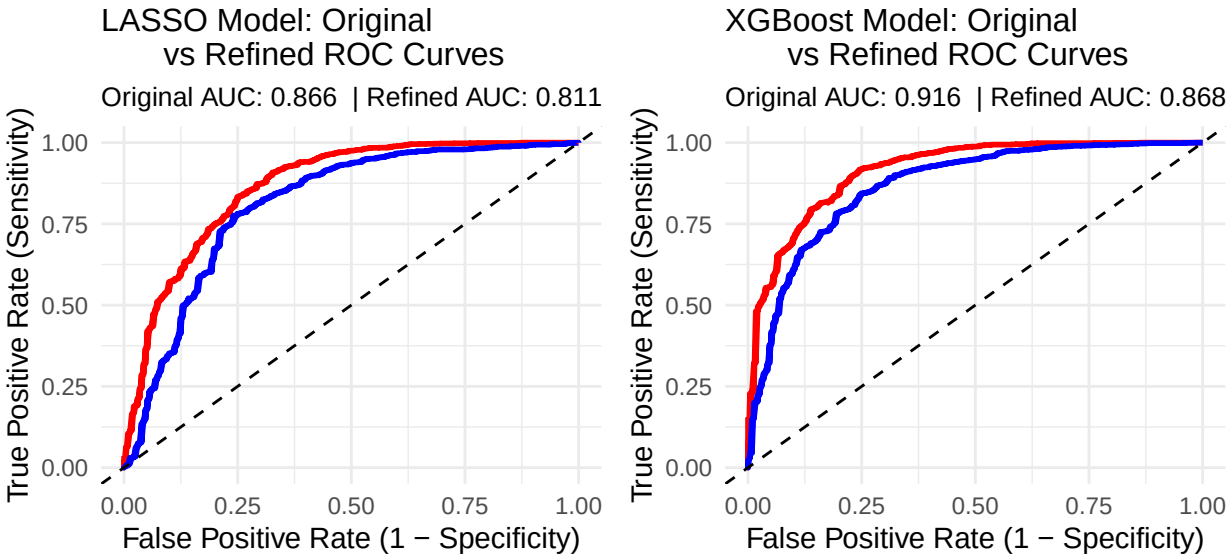

Figure 5: Refined vs Unrefined Model Comparison by ROC AUC

New Variables

We newly introduced a variable, `is_near_holiday`. It categorizes whether an animal's intake date falls within a certain proximity to major holidays, including Independence Day, Christmas, Halloween, New Year's, and Valentine's Day. The goal was to determine whether holiday-related intake patterns could help predict an animal's outcome. To evaluate its effectiveness, we built two boosting models using LASSO regularization: a baseline model with all predictors except `is_near_holiday`, and a holiday model using only `is_near_holiday`. We compared their accuracy and ROC-AUC using 3-fold cross-validation. Results showed that the holiday model performed significantly worse in terms of ROC-AUC, suggesting it is a weak predictor of survival outcomes. Accuracy also dropped slightly compared to the baseline model. This indicates that proximity to holidays does not strongly influence shelter outcomes, or that any potential effects may be overshadowed by more influential variables. Consequently, `is_near_holiday` will not be useful in the final predictive model.

```
## # A tibble: 4 x 3
##    mean metric   model
##   <dbl> <chr>    <chr>
## 1 0.936 accuracy base
## 2 0.872 roc_auc  base
## 3 0.925 accuracy holiday
## 4 0.501 roc_auc  holiday
```

Conclusion

Now that we have determined the optimal parameters for our models, we can evaluate our results using the table below.

Table 4: Performance Metrics for Final Models

Model	Sensitivity	Specificity	Accuracy
Lasso Model	0.989	0.408	0.926
Lasso Model Refined	0.994	0.064	0.895
XG Boost Model	0.994	0.459	0.937
XG Boost Model Refined	0.987	0.322	0.916

From the table, we see that the unrefined XG Boost model is the best, most robust model tested. It provides the highest accuracy, a fantastic true positive (sensitivity) rate of 99.4%, and the best true negative rate (specificity) out of any of our model at 45.9%.

Table 5: Confusion Matrix for XG Boost Model

Truth	Prediction	Count
FALSE	FALSE	107
TRUE	FALSE	126
FALSE	TRUE	12
TRUE	TRUE	1931

Looking at our true negative rate and our confusion matrix, it is apparent that it is still challenging to predict an animal that will not survive the shelter despite having comprehensive data. However, our model does significantly better than a naive model that simply predicts every animal will live. In our dataset, 89.3% of the animals entering the shelter ended up alive, meaning a naive model would be 89.3% accurate.

In contrast, our model is 93.7% accurate, demonstrating that our model has learned patterns to help predict animal outcomes.

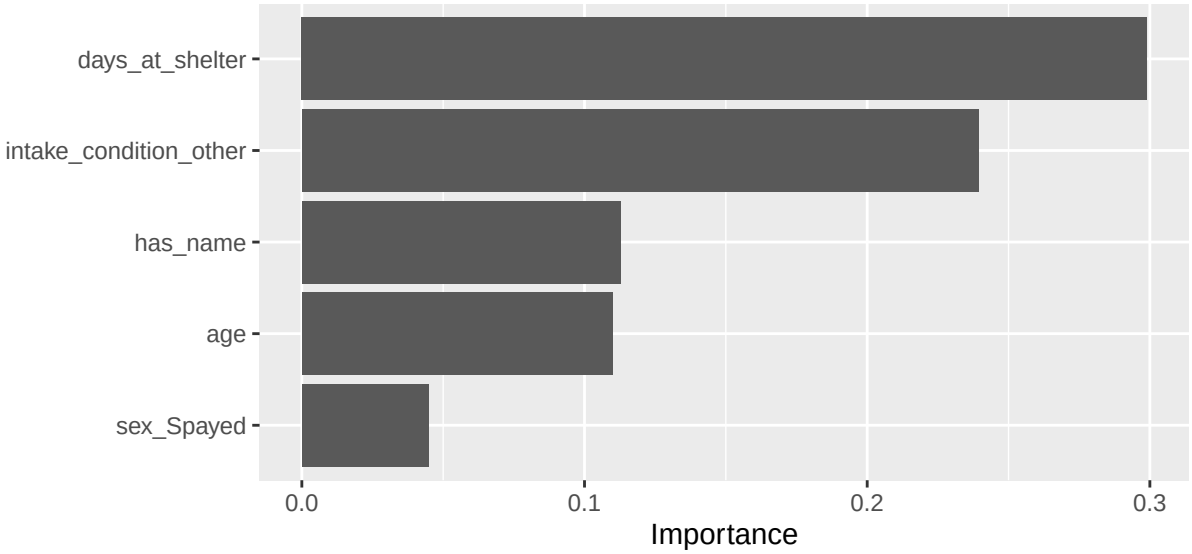

Figure 6: VIP Ranking of top 5 most Important Variables

Looking at the most important variables for the model, some variables have a more obvious correlation with animal outcome than other. The most important variables for our model are: Number of Days Spent at Shelter, if the animal has a name or not, the age of the animal, and if the animal is spayed.

Some of these important relationships would be expected. For instance, the number of days spent at the shelter can indicate one of two things. For one, if the animal spends very little time at the shelter, it may indicate that it was a lost cause from the beginning and needed to be put down immediately. However, this is not always the case for animals with short stays. Often times, owners of lost animals check the animal shelter and pick their pet up after a short stay at the shelter, resulting in a positive shelter outcome. On the other side of the spectrum, we have animals who have very long stays at the shelter. This would indicate that the animal is struggling to be adopted, and may unfortunately have to be euthanized eventually. The unfortunate reality is that many of these animal shelters are operating on very limited resources, and the longer an animal stays in the shelter, the more they cost the shelter as the shelter has to feed and take care of the animal. If an animal is at the shelter for a long period of time and hasn't been adopted, the shelter often ends up putting the animal down.

A variable that does not have as clear of a correlation is has_name. This variable came in as the third most important for our model. A possible explanation for this is that animals that already have names are animals that are either dropped off because people can no longer take care of them, or they are found with a collar on. Many times, these animals with collars on are simply runaway pets that have been picked up by animal control, so the owners shortly come and pick the pet up, resulting in a positive shelter outcome. On the contrary, if a pet does not have a name, there is a high probability it is a stray. Obviously, strays have a higher probability of sickness, injury, or malnourishment, all do to lack of proper care.

The age of the animal is a pretty obvious indicator. First off, many people looking to adopt are looking for pets to have for many years, so naturally they are looking for younger animals. In addition, older pets tend to require more care perhaps even medications, making the pet more expensive to keep at the shelter, and more expensive to adopt. While most pet owners do not see their pet as a financial obligation, but as a family member, there is still the fact the at the end of the day, money will always play a factor. This also applies for the animal shelter. It is the shelters mission to save as many pets as possible, but on a small budget. These factors often result in negative outcomes for older animals at the vet.

As expected, whether or not an animal is spayed/neutered is important to determining whether or not an animal will survive the shelter. At an animal shelter, all animals are spayed or neutered before adoption. These procedures are a large cost for the shelter, as the cost per animal for this procedure can be anywhere from $100-$200. Thus, if an animal enters the shelter already spayed or neutered, it greatly increases the chances of survival. In addition, if the animal enters the shelter already spayed or neutered, it is likely that the animal is domesticated, thus increasing the liklihood of adoption.

Lastly, a more difficult predictor to understand is 'intake_condition_other'. Since we used step_other in our recipe, it grouped all characteristics that make up less than 5% of the animal population into the other category. This includes the intake conditions aged, behavior, feral and welfare seizures. These categories would make sense as a good predictors since all of these conditions that were placed into the other category likely increase the liklihood of the animal being euthanized.

In this project, we created several models, with the XG boost model providing the best accuracy at 93.7%. While predicting negative animal outcomes remains a challenge, our model still provides meaningful insight that could potentially help shelters identify at-risk animals earlier.

Appendix

Categorical Variable Summary:

Refined Model Code

Lasso Model

```
lasso_refined_recipe <- recipe(was_outcome_alive ~ ., data = shelter_clean) %>%
  step_rm(has_name, animal_type, sex, age) %>%
  step_novel(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_impute_median(all_numeric_predictors()) %>%
  step_impute_mode(all_nominal_predictors())%>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_numeric_predictors(), na_rm = TRUE)

refined_lasso_workflow <- workflow() %>%
  add_recipe(lasso_refined_recipe) %>%
  add_model(lasso_spec)


refined_lasso_tune <- tune_grid(
  refined_lasso_workflow,
  resamples = shelter_folds,
  grid = lasso_grid
)

best_refined_lambda <- select_best(refined_lasso_tune, metric= "roc_auc")
final_refined_lasso <- finalize_workflow(refined_lasso_workflow, best_refined_lambda)
refined_lasso_fit <- fit(final_refined_lasso, data = train_data)



refined_preds <- predict(refined_lasso_fit, new_data = test_data, type = "prob")
refined_results <- test_data %>%
  select(was_outcome_alive) %>%
```

```
  bind_cols(refined_preds)
refined_roc <- roc_curve(refined_results, truth = was_outcome_alive, ".pred_TRUE")
```

XGBoost Model

```
xgb_refined_recipe <- recipe(was_outcome_alive ~ ., data = train_data) %>%
  step_rm(intake_condition, age) %>%
  step_impute_median(all_numeric_predictors()) %>%
  step_impute_mode(all_nominal_predictors())%>%
  step_other(all_nominal_predictors(), threshold = 0.10)%>%
  step_dummy(all_nominal_predictors())%>%
  step_normalize(all_numeric_predictors())


refined_xgb_workflow <- workflow() %>%
  add_model(xgb_spec) %>%
  add_recipe(xgb_refined_recipe)


refined_xgb_tuned <- tune_grid(refined_xgb_workflow, resamples = xgb_cv, grid = xgb_grid)

best_refined_xgb <- select_best(refined_xgb_tuned)

final_refined_xgb <- finalize_workflow(refined_xgb_workflow, best_refined_xgb)
refined_xgb_fit <- fit(final_refined_xgb, data = train_data)



refined_xgb_probs <- predict(refined_xgb_fit, test_data, type = "prob")$.pred_TRUE
refined_xgb_roc <- roc(test_data$was_outcome_alive, refined_xgb_probs)
refined_xgb_auc <- auc(refined_xgb_roc)
```

Model Comparison

```
lasso_roc_obj <- roc(response = test_data$was_outcome_alive,
                     predictor = test_results$.pred_TRUE)
refined_lasso_roc_obj <- roc(response = test_data$was_outcome_alive,
                             predictor = refined_results$.pred_TRUE)


xgb_roc_obj <- roc(response = test_data$was_outcome_alive,
                   predictor = xgb_probs$.pred_TRUE)
refined_xgb_roc_obj <- roc(response = test_data$was_outcome_alive,
                           predictor = refined_xgb_probs)

original_lasso_auc <- auc(lasso_roc_obj)
refined_lasso_auc <- auc(refined_lasso_roc_obj)
original_xgb_auc <- auc(xgb_roc_obj)
refined_xgb_auc <- auc(refined_xgb_roc_obj)
```

```r
create_roc_df <- function(roc_obj) {
  data.frame(
    fpr = 1 - roc_obj$specificities,
    tpr = roc_obj$sensitivities
  )
}

lasso_roc_df <- create_roc_df(lasso_roc_obj)
refined_lasso_roc_df <- create_roc_df(refined_lasso_roc_obj)
xgb_roc_df <- create_roc_df(xgb_roc_obj)
refined_xgb_roc_df <- create_roc_df(refined_xgb_roc_obj)


lasso_comparison <- ggplot() +
  geom_line(data = lasso_roc_df,
            aes(x = fpr, y = tpr),
            color = "red", linewidth = 1) +
  geom_line(data = refined_lasso_roc_df,
            aes(x = fpr, y = tpr),
            color = "blue", linewidth = 1) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  labs(title = "LASSO Model: Original vs Refined ROC Curves",
       subtitle = paste("Original AUC:", round(original_lasso_auc, 3),
                        " | Refined AUC:", round(refined_lasso_auc, 3)),
       x = "False Positive Rate (1 - Specificity)",
       y = "True Positive Rate (Sensitivity)") +
  theme_minimal()


xgb_comparison <- ggplot() +
  geom_line(data = xgb_roc_df,
            aes(x = fpr, y = tpr),
            color = "red", linewidth = 1) +
  geom_line(data = refined_xgb_roc_df,
            aes(x = fpr, y = tpr),
            color = "blue", linewidth = 1) +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  labs(title = "XGBoost Model: Original vs Refined ROC Curves",
       subtitle = paste("Original AUC:", round(original_xgb_auc, 3),
                        " | Refined AUC:", round(refined_xgb_auc, 3)),
       x = "False Positive Rate (1 - Specificity)",
       y = "True Positive Rate (Sensitivity)") +
  theme_minimal()

library(gridExtra)
grid.arrange(lasso_comparison, xgb_comparison, ncol = 2)

cat("\nModel AUC Summary:\n")
cat("-------------------\n")
cat(sprintf("Original LASSO AUC: %.3f\n", original_lasso_auc))
cat(sprintf("Refined LASSO AUC:  %.3f\n", refined_lasso_auc))
cat(sprintf("Original XGB AUC:  %.3f\n", original_xgb_auc))
cat(sprintf("Refined XGB AUC:   %.3f\n", refined_xgb_auc))
```

###New Variable Creation:

```
shelter_cleaner = shelter %>%
  mutate(intake_date = as.Date(intake_date, format = "%m/%d/%Y"))%>%
  mutate(
    year = year(intake_date),
    near_independence = abs(intake_date - as.Date(paste0(year, "-07-04"))) <= 7,
    near_christmas = abs(intake_date - as.Date(paste0(year, "-12-25"))) <= 7,
    near_halloween = abs(intake_date - as.Date(paste0(year, "-10-31"))) <= 3,
    near_newyears = abs(intake_date - as.Date(paste0(year, "-01-01"))) <= 3,
    near_valentines = abs(intake_date - as.Date(paste0(year, "-02-14"))) <= 3
  )%>%
  select(-animal_id, -reason_for_intake, -crossing, -outcome_subtype, -outcome_is_dead, -geored, -outcor
  mutate(has_name = ifelse(is.na(animal_name), 0, 1)) %>%
  select(-animal_name) %>%
  mutate(days_at_shelter = as.Date(outcome_date, format = "%m/%d/%Y") - as.Date(intake_date, format = "
  mutate(age = (as.Date(intake_date, format = "%m/%d/%Y") - as.Date(dob, format = "%m/%d/%Y"))) %>%
  select(-outcome_date, - intake_date, -dob) %>%
  mutate(days_at_shelter = as.numeric(days_at_shelter), age = as.numeric(age)) %>%
  filter(age > 0) %>%
  mutate(days_at_shelter = log(days_at_shelter), log_age= log(age))


shelter_cleaner <- shelter_cleaner %>%
  mutate(
    near_holiday = case_when(
      near_independence ~ "Independence",
      near_christmas ~ "Christmas",
      near_halloween ~ "Halloween",
      near_newyears ~ "NewYears",
      near_valentines ~ "Valentines",
      TRUE ~ "None"
    ),
    is_near_holiday = factor(near_holiday)
  ) %>%
  select(-starts_with("near_"), -year)
```

New variable model creation along with base model for comparison:

```
set.seed(13223)

shelter_cleaner <- shelter_cleaner %>%
  filter(days_at_shelter > 0) %>%
  mutate(
    days_at_shelter = log(days_at_shelter),
    was_outcome_alive = factor(was_outcome_alive)
  )

data_split2 <- initial_split(shelter_cleaner, prop = 0.8)
train_data2 <- training(data_split2)
test_data2 <- testing(data_split2)

base_recipe <- recipe(was_outcome_alive ~ ., data = train_data2) %>%
  step_rm(is_near_holiday) %>%
```

```
  step_mutate_at(all_numeric_predictors(), fn = ~ifelse(is.infinite(.), NA, .)) %>%
  step_impute_median(all_numeric_predictors()) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric_predictors())

holiday_recipe <- recipe(was_outcome_alive ~ is_near_holiday, data = train_data2) %>%
  step_mutate_at(all_numeric_predictors(), fn = ~ifelse(is.infinite(.), NA, .)) %>%
  step_impute_median(all_numeric_predictors()) %>%
  step_novel(all_nominal_predictors()) %>%
  step_unknown(all_nominal_predictors()) %>%
  step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric_predictors())

xgb_spec <- boost_tree(
  trees = tune(),
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune(),
  sample_size = tune(),
  mtry = tune()
) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

folds <- vfold_cv(train_data2, v = 3)

base_wf <- workflow() %>%
  add_model(xgb_spec) %>%
  add_recipe(base_recipe)

holiday_wf <- workflow() %>%
  add_model(xgb_spec) %>%
  add_recipe(holiday_recipe)

xgb_grid <- grid_latin_hypercube(
  trees(),
  tree_depth(),
  learn_rate(),
  loss_reduction(),
  sample_size = sample_prop(),
  finalize(mtry(), train_data2),
  size = 20
)

base_tune <- tune_grid(base_wf, resamples = folds, grid = xgb_grid, metrics = metric_set(accuracy, roc_
holiday_tune <- tune_grid(holiday_wf, resamples = folds, grid = xgb_grid, metrics = metric_set(accuracy
```

Characteristic	**N = 10,874**[1]
animal_type	
bird	46 (0.4%)
cat	4,936 (45%)
dog	5,505 (51%)
guinea pig	82 (0.8%)
livestock	6 (<0.1%)
other	39 (0.4%)
rabbit	209 (1.9%)
reptile	32 (0.3%)
wild	19 (0.2%)
primary_color	
black	3,549 (33%)
blue	166 (1.5%)
brown	2,256 (21%)
gold	28 (0.3%)
gray	1,497 (14%)
green	31 (0.3%)
orange	471 (4.3%)
pink	2 (<0.1%)
red	105 (1.0%)
silver	36 (0.3%)
tan	657 (6.0%)
tricolor	182 (1.7%)
wheat	1 (<0.1%)
white	1,885 (17%)
yellow	8 (<0.1%)
secondary_color	
black	1,243 (11%)
blue	29 (0.3%)
brown	1,274 (12%)
gold	17 (0.2%)
gray	469 (4.3%)
green	7 (<0.1%)
orange	97 (0.9%)
purple	2 (<0.1%)
red	49 (0.5%)
silver	25 (0.2%)
tan	719 (6.6%)
tricolor	74 (0.7%)
wheat	3 (<0.1%)
white	6,854 (63%)
yellow	12 (0.1%)
sex	
Female	2,700 (25%)
Male	3,329 (31%)
Neutered	2,703 (25%)
Spayed	2,142 (20%)
intake_condition	
aged	69 (0.6%)