Predicting Tsunami Risk Using Earthquake Data

Minu Pabbathi

2024-12-08

Contents

Introduction

For Californians, earthquakes are no strange occurrence. Southern California alone experiences around 100,000 earthquakes a year, although only several hundred of them are strong enough to be felt. Despite this, those on the coast harbor a nagging fear that any one of those simple earthquakes might evolve into something worse. Earthquakes in coastal regions are particularly dangerous as they carry the risk of developing into tsunamis. Thus, the aim of this project is to use earthquake data to predict whether there is risk of a tsunami.

To do this, I will first explore and clean the data by removing duplicate values, accounting for missing data, and selecting relevant features. I will then split the data into training and testing sets and create 10 folds for cross validation. The training set will be used to train 7 predictive models. Since our outcome variable, tsunami risk, is a binary categorical variable, we will need to use classification models. I chose to implement logistic regression, linear discriminant analysis, k-nearest neighbors, random forest, boosted tree, radial support vector machine, and linear support vector machine. Once we fit each model, we will select the top two best performing models and fit those to the testing data to evaluate the efficacy of our models.

The data used for this project came from Kaggle and can be found at this link. More information on each variable can be found in the codebook.

Loading Libraries and Data

Before we start, we need to load a few libraries and the data. Since the variables `alert` and `tsunami` are categorical, we convert them into factors to make them compatible with later functions that we will use.

```r
library(tidyverse)
library(tidymodels)
library(skimr) # skim()
 library(gridExtra) # arranging plots
library(corrplot) # correlation plot
library(naniar) # vis_miss()
library(ROSE) # oversampling
library(car) # VIF
library(discrim) #LDA
library(kernlab) # SVM kernels
library(vip) # variable importance plot
tidymodels_prefer()
earthquakes <- read_csv("./data/earthquakes.csv")
earthquakes <- earthquakes %>%
  mutate(alert = as.factor(alert), tsunami = as.factor(tsunami))
```

EDA

Finding and Removing Duplicate Values

In order to understand our data better, we can check the dimensions of our data set and make sure that there aren't any duplicated data. We can run the following code to see how many observations we have in total and how many are duplicates.

```r
dim(earthquakes)
```

```
## [1] 1137   43
```

```r
sum(duplicated(earthquakes))
```

```
## [1] 337
```

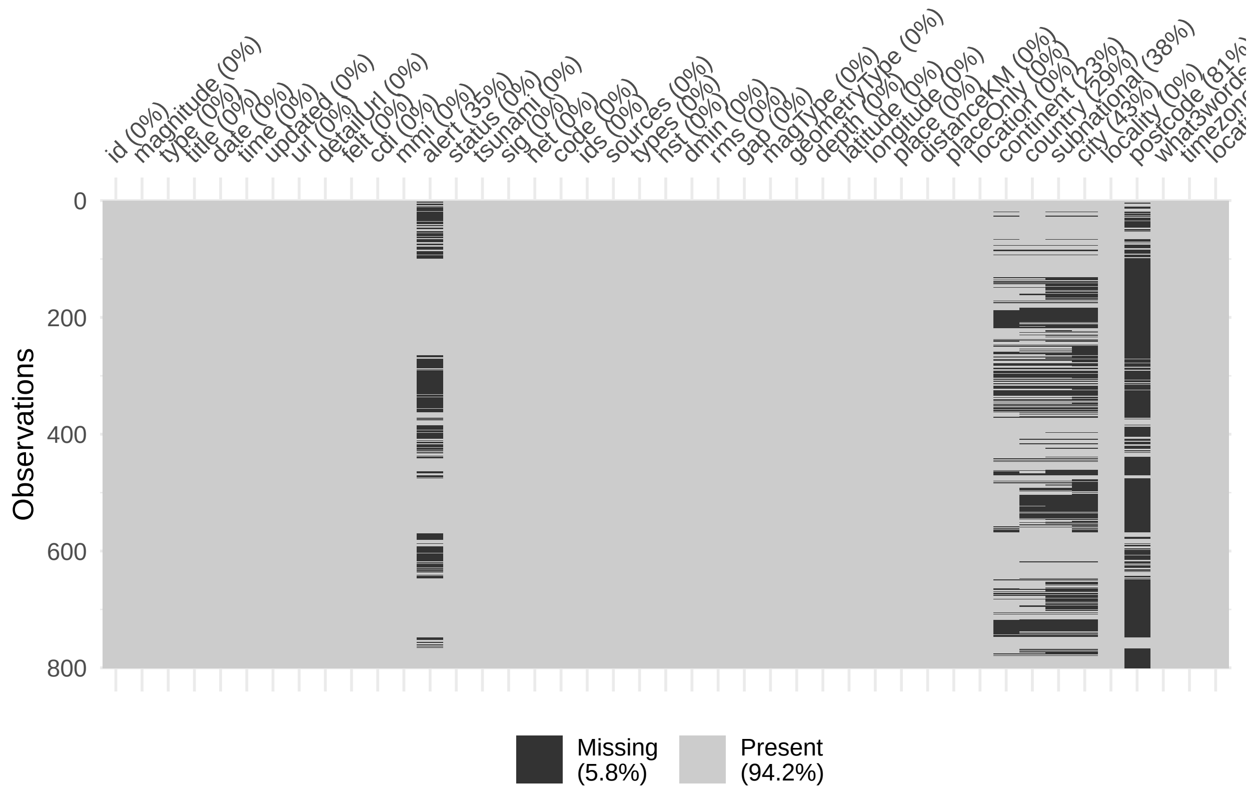
337 of our observations are repeats, so let's remove them from our dataset and only keep unique observations.

```r
earthquakes <- unique(earthquakes)
```

Missing Data

Next, we check for missing data. We can visualize which values are missing with a call to `vis_miss()`.

```r
vis_miss(earthquakes)
```



Most of the missing columns do not seem to be useful for our model, so they can be removed. However, `alert` could be informative in determining tsunami risk. When creating a recipe for our model, we can impute values, or fill in missing data, for `alert` in order to generate a functional model.

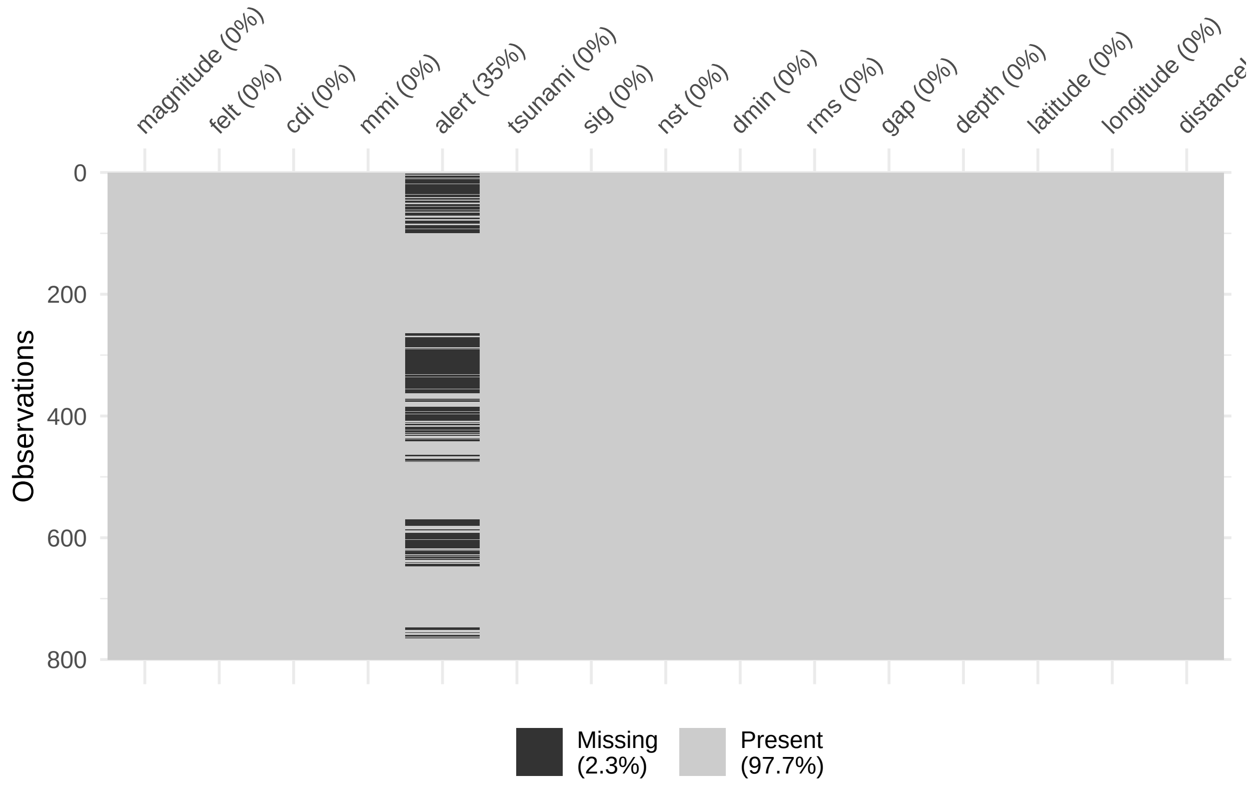
Feature Selection

Not all of the variables in our dataset will be valuable for prediction, and oftentimes a simple model will perform just as well as a more complex one. Based on the graph above we can remove certain variables that might not be helpful in our final models. Since `postcode` and `city` have a lot of missing values and don't seem to be important for prediction, it is easier to just remove them from our dataset. We can also remove `url`, `detailUrl`, `what3words`, and `locationDetails` since they just provide additional information about the observation and will not have much impact on the final analysis. `type` and `geometryType` have the same value for all observations so they won't make any difference. `date`, `time`, and `timezone` require an additional layer of analysis to our models, and so to keep things simple, we will exclude them for this project. The same goes for `location`, `country`, `subnational`, `continent`, `place`, `placeOnly`, `title`, and `locality`. Finally, we can remove `id`, `ids`, `sources`, `types`, `status`, `net`, and `code` since they are only identifying labels.

```
earthquakes <- select(earthquakes, -"postcode", -"url", -"detailUrl", -"what3words", -"locationDetails"
```

We can re-visualize the new data:

```
vis_miss(earthquakes)
```



Summary Statistics

For our initial inspection of the data, it is helpful to look at the bigger picture. We can quickly visualize all of our data with a call to `skim()`. This function provides an overall summary of our data, statistics about nominal variables, and statistics about our numeric data.

```r
skim(earthquakes)
```

Table 1: Data summary

Name	earthquakes
Number of rows	800
Number of columns	15
Column type frequency:	
factor	2
numeric	13
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
alert	278	0.65	FALSE	4	gre: 485, yel: 28, red: 6, ora: 3
tsunami	0	1.00	FALSE	2	0: 745, 1: 55

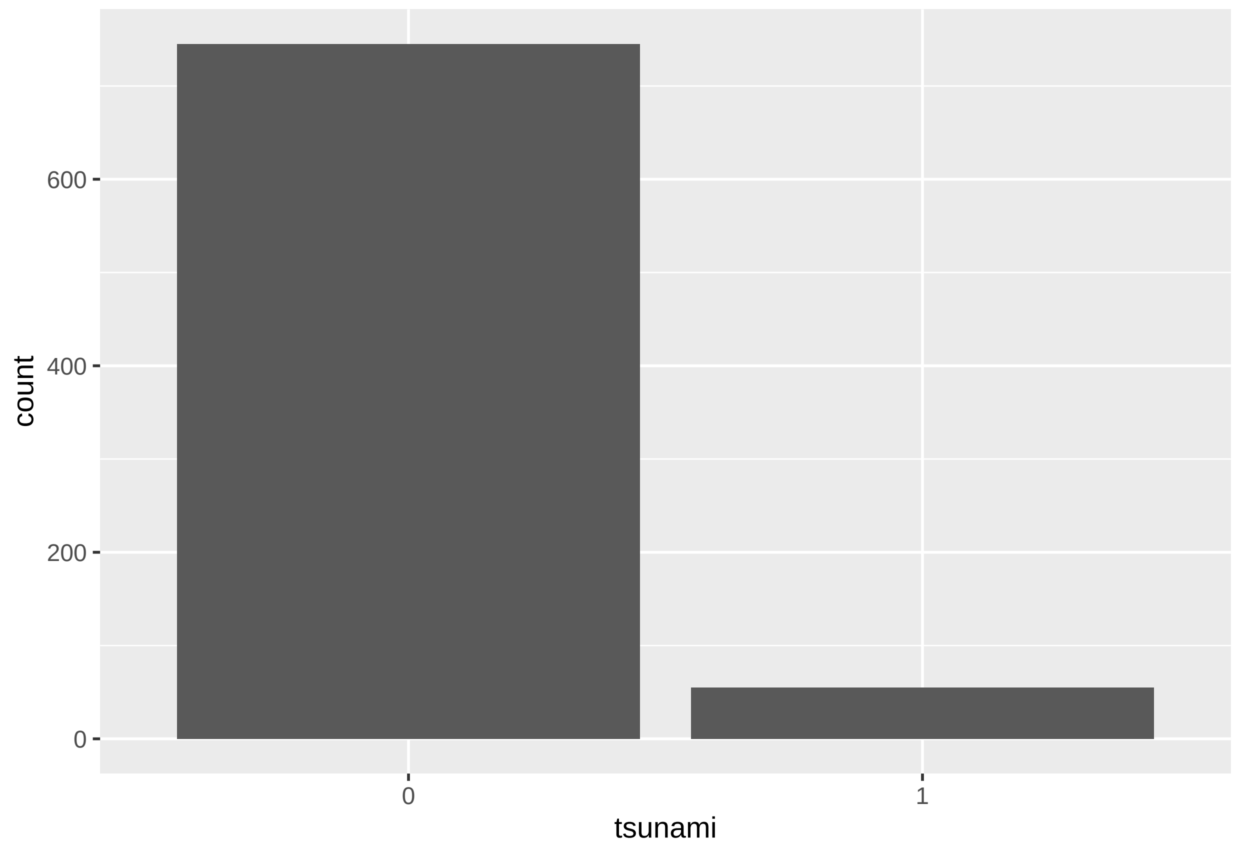
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
magnitude	0	1	4.83	1.04	3.00	3.80	5.20	5.60	7.60	
felt	0	1	474.17	6746.87	0.00	0.00	2.00	27.00	183786.00	
cdi	0	1	2.97	2.59	0.00	0.00	3.00	5.00	9.00	
mmi	0	1	4.35	1.50	1.00	4.00	4.00	5.00	9.00	
sig	0	1	428.61	255.27	138.00	234.00	449.00	504.00	2419.00	
nst	0	1	108.41	87.97	0.00	36.00	97.00	147.00	619.00	
dmin	0	1	1.27	1.64	0.00	0.10	0.62	1.90	12.46	
rms	0	1	0.60	0.30	0.00	0.36	0.63	0.78	2.52	
gap	0	1	56.15	41.03	0.00	30.00	49.00	69.00	256.00	
depth	0	1	38.38	83.01	-0.25	7.46	10.00	35.00	639.50	
latitude	0	1	28.07	19.80	-43.71	23.81	31.67	37.80	68.18	
longitude	0	1	-8.56	118.82	-179.81	-104.51	-71.56	123.75	179.97	
distanceKM	0	1	55.28	58.81	0.00	16.00	37.00	64.00	298.00	

Outcome Variable

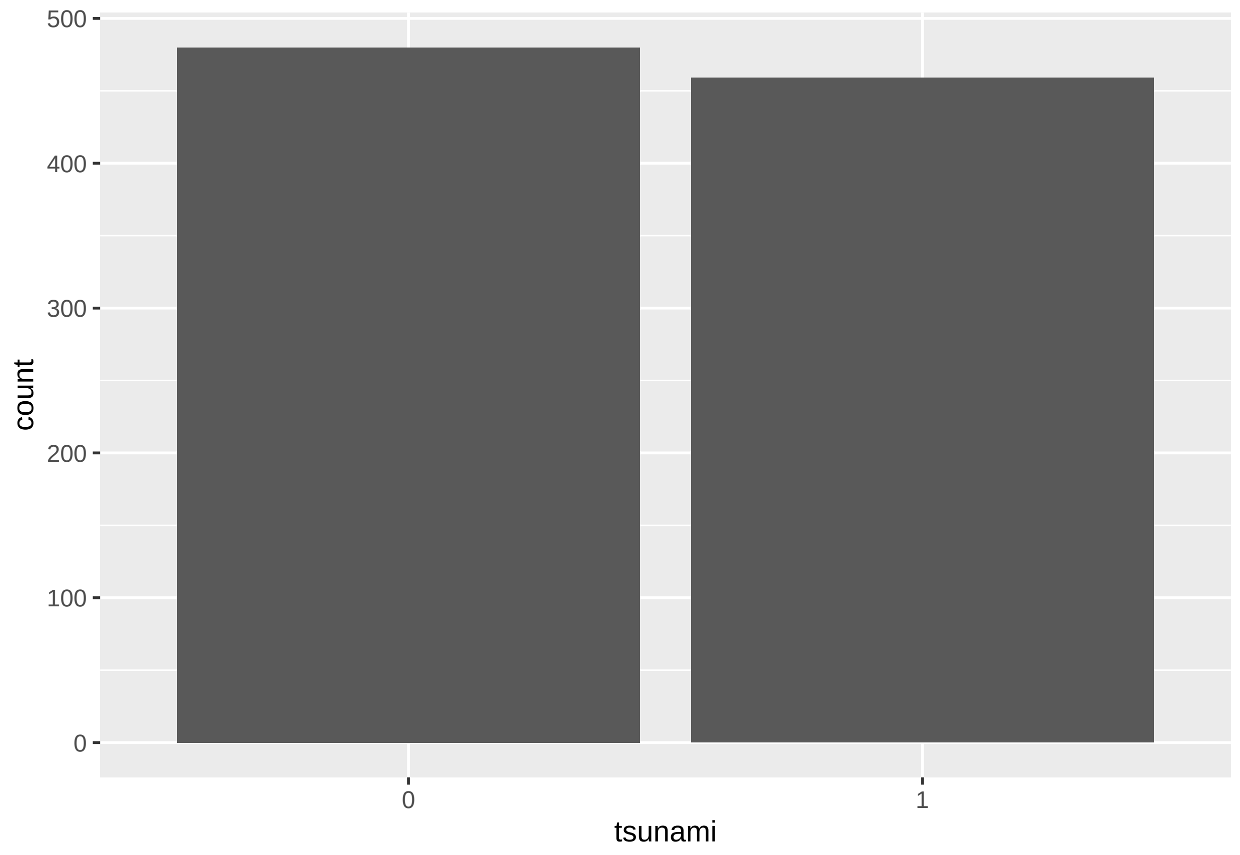
We can now look at the distributions of individual variables. We first examine our outcome variable, `tsunami`. Since it is a categorical variable, we use a barplot to visualize the distribution.

```r
ggplot(earthquakes, aes(x = tsunami)) +
  geom_bar()
```



Since there is a large difference in the number of positive and negative cases, we can use resampling techniques to balance the categories of the outcome variable. I chose to use oversampling, in which minority case samples are duplicated in order to create a more balanced class distribution.

```r
set.seed(42)
earthquakes <- ovun.sample(tsunami ~ . , data = earthquakes, method = "over")$data
ggplot(earthquakes, aes(x = tsunami)) +
  geom_bar()
```



Now that the target variable is more balanced, let's look at other relationships among the data!

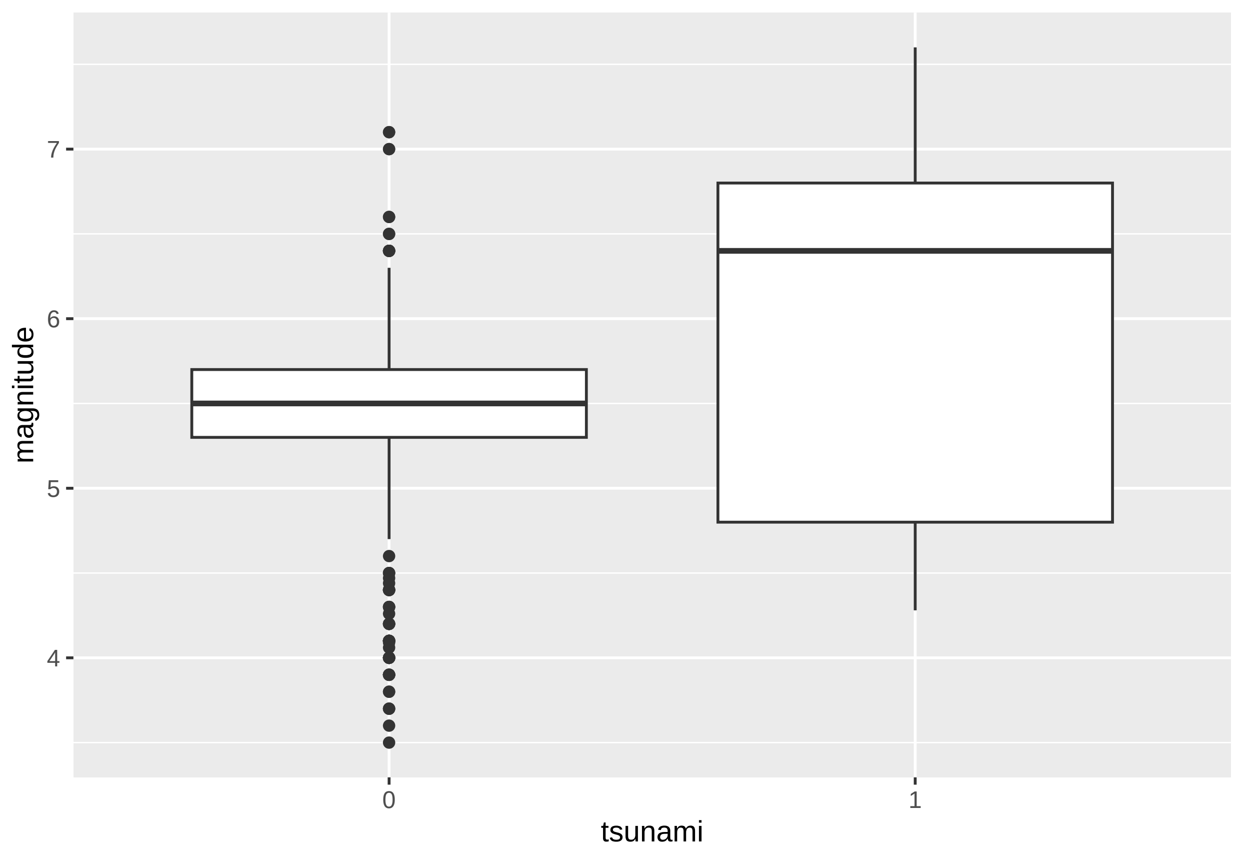
Plotting Variables Against Target

Looking at how individual variables affect the target variable can give us some insight into how useful they will be for prediction. I was particularly curious about how magnitude, longitude, and latitude would affect tsunami risk, so I used different plots to visualize their relationship.

Magnitude vs. Tsunami

It seems intuitive that magnitude would affect tsunami risk - after all the stronger an earthquake is the more destructive its effects will be. The plot below corroborates this, as the mean magnitude of earthquakes with a tsunami risk appears much higher than the magnitude of earthquakes without a tsunami risk, indicating that magnitude might be an influential variable in predicting the risk of a tsunami.

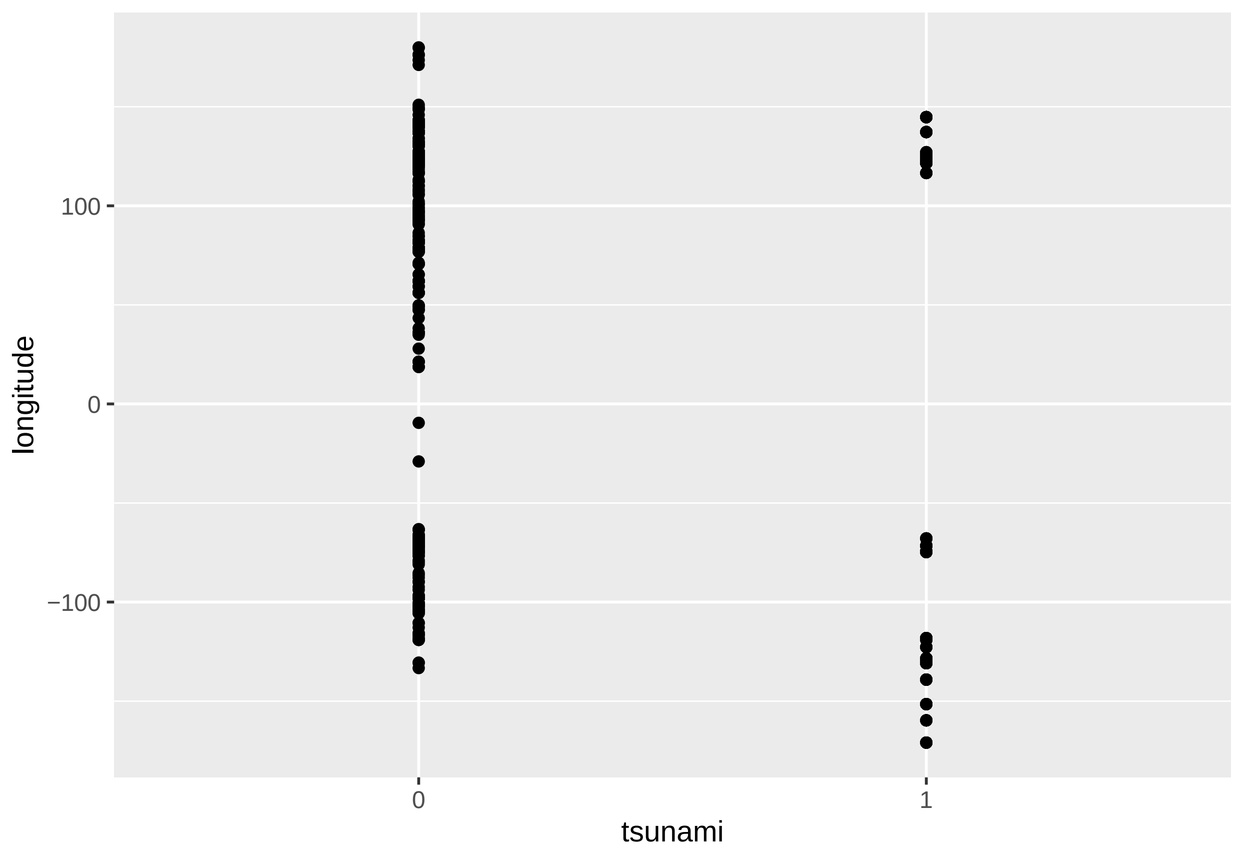
```r
ggplot(earthquakes, aes(x = tsunami, y = magnitude)) +
  geom_boxplot()
```



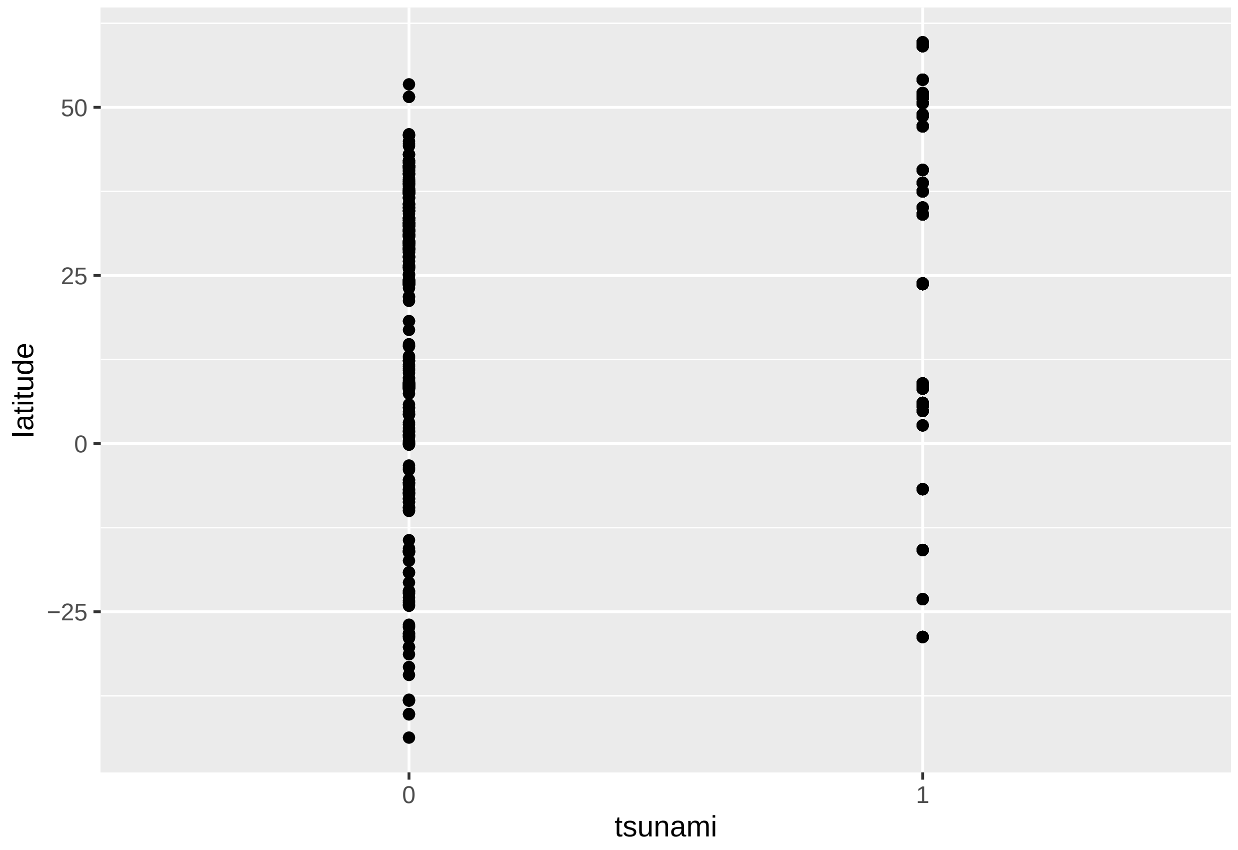
Tsunami Risk by Longitude/Latitude

I was also curious about how longitude and latitude would affect tsunami risk. Based on the plots below, more extreme longitudes tend to have higher tsunami risks and higher latitudes have a slightly higher tsunami risk. Although the relationships are not very strong, it seems worthwhile to include them in our models.

```r
ggplot(earthquakes, aes(x = tsunami, y = longitude)) +
  geom_point()
```



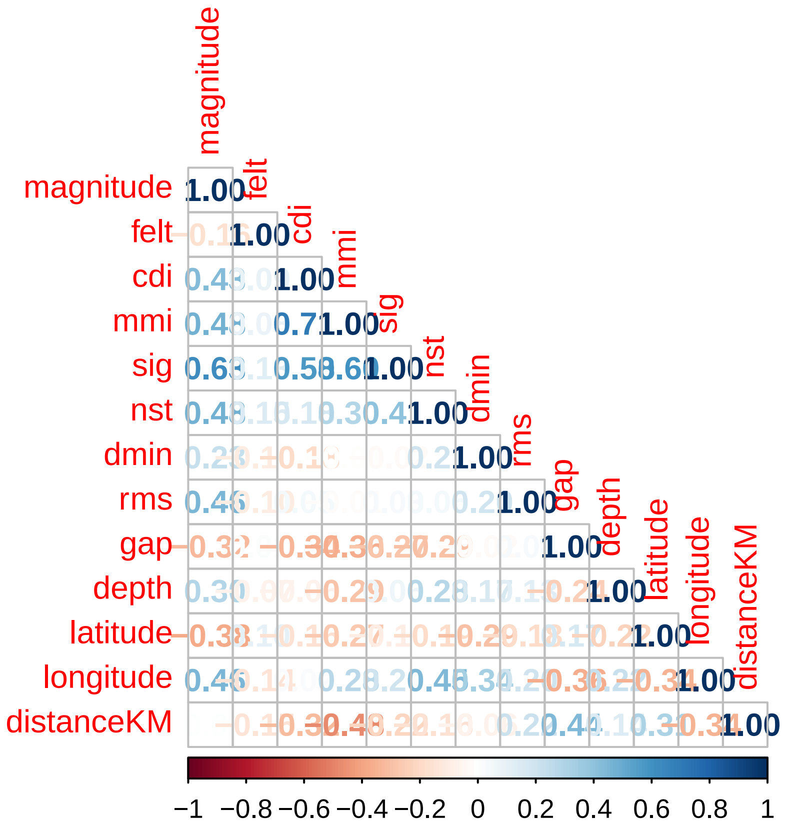
```r
ggplot(earthquakes, aes(x = tsunami, y = latitude)) +
  geom_point()
```



Correlation Plot

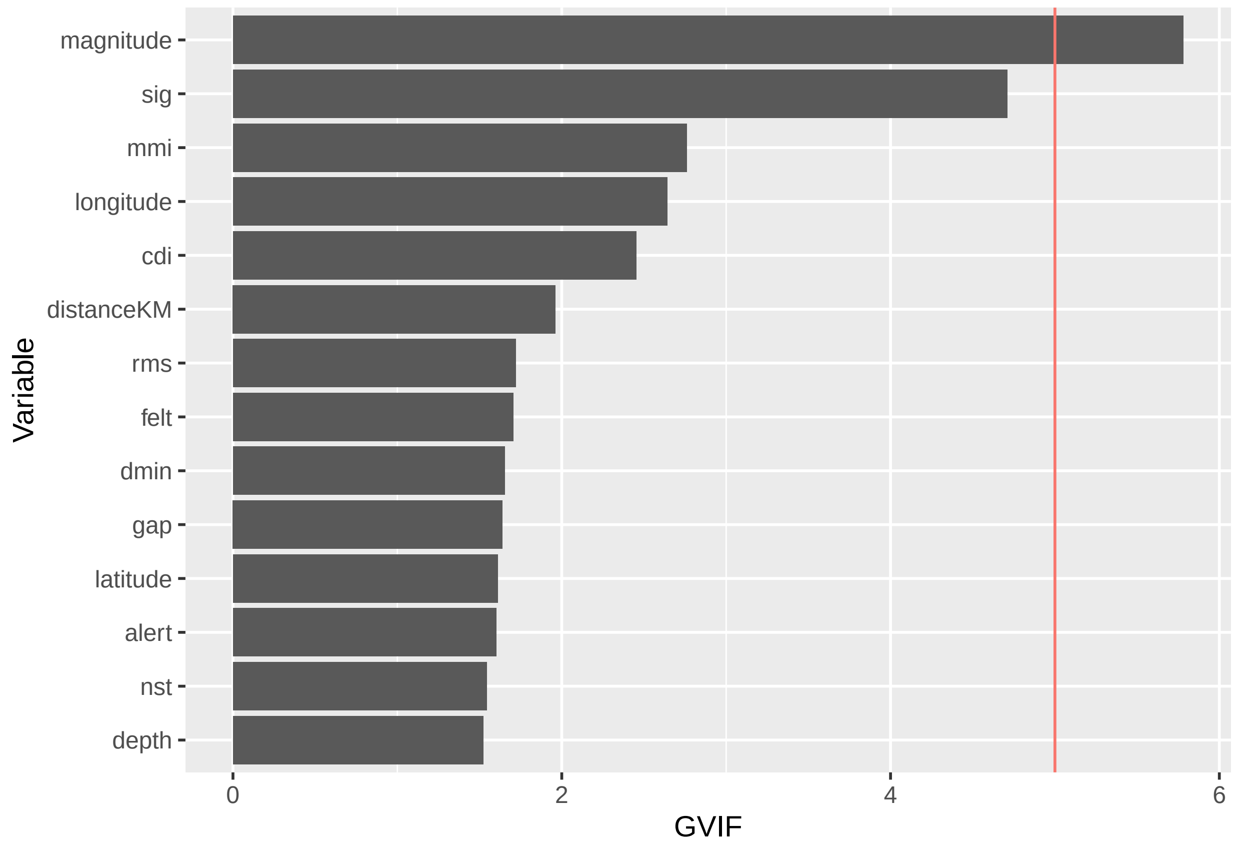
A correlation plot is a helpful tool to visualize relationships between variables. The closer the value is to 1 or -1, the more correlated two variables are. A negative correlation indicates an inverse relationship, while a positive correlation implies a direct relationship.

```
earthquakes %>%
  select(where(is.numeric)) %>%
  cor() %>%
  corrplot(method = "number", type = "lower")
```



A few variables are relatively highly correlated, which can diminish the efficacy of our models. We can check which ones are highly correlated with others by checking their variance inflation factor, or VIF. A value of 1 indicates no correlation, while a value above 5 indicates severe correlation. Since we have categorical variables, we use the GVIF instead of the VIF.

```r
earthquakes_vif <- vif(glm(tsunami ~ ., data = earthquakes, family = binomial))
data.frame(earthquakes_vif, Variable = rownames(earthquakes_vif), adj_gvif = earthquakes_vif[, 3]) %>%
  ggplot(aes(x = reorder(Variable, adj_gvif), y = adj_gvif)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  geom_hline(aes(yintercept = 5, color = "red"), show.legend = F) +
  labs(x = "Variable", y = "GVIF")
```

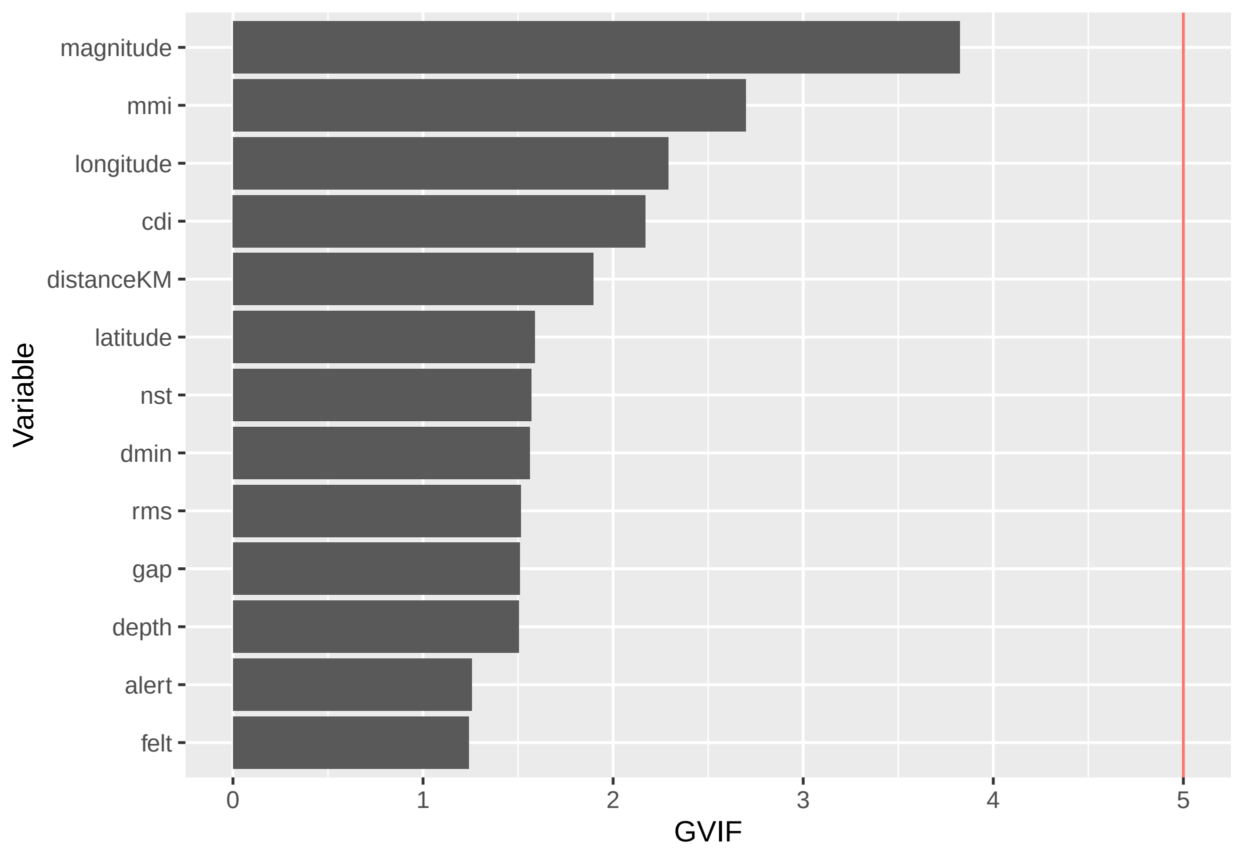


`magnitude` is highly correlated with other predictors, but as we saw above, it seems to have an effect on `tsunami`. Instead, we can drop one of the variables that are highly correlated with `magnitude`. I chose to drop `sig` (significance) since magnitude is used to determine significance, making it redundant.

```r
earthquakes <- select(earthquakes, -"sig")
```

We can re-check the VIFs:

```r
earthquakes_vif <- vif(glm(tsunami ~ ., data = earthquakes, family = binomial))
data.frame(earthquakes_vif, Variable = rownames(earthquakes_vif), adj_gvif = earthquakes_vif[, 3]) %>%
  ggplot(aes(x = reorder(Variable, adj_gvif), y = adj_gvif)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  geom_hline(aes(yintercept = 5, color = "red"), show.legend = F) +
  labs(x = "Variable", y = "GVIF")
```



The VIF of `magnitude` is now below 5!

Now that we have looked at our variables, let's build some models!

Data Splitting and Cross Validation

Training/Testing Split

In order to properly evaluate our models, we need to split our dataset into a training set and a testing set. This is so our models can be trained on a subset of the data and ultimately evaluated on the testing set. Training our models on all of the data can lead to overfitting, which is when the model becomes too closely aligned to the training set by attributing random noise in the data to variation due to predictors. I chose to designate 75% of the data as the training set so that there is sufficient data to train the models, but also enough that we can properly evaluate our models on the testing set. We stratify the split on the outcome variable to ensure an equal distribution in our training and testing sets.

```r
set.seed(42)
earthquakes_split <- initial_split(earthquakes, strata = tsunami)
earthquakes_train <- training(earthquakes_split)
earthquakes_test <- testing(earthquakes_split)
```

Now we can check the dimensions of our training and testing sets:

```
nrow(earthquakes_train)
```

```
## [1] 704
```

```
nrow(earthquakes_test)
```

```
## [1] 235
```

Our data split looks good, so we can move on to cross-validation!

Cross Validation

To tune our models, we can use 10-fold cross-validation. The data will be randomly shuffled and then separated into 10 subsets, or folds. The model will then be trained on 9 of the folds and tested on the 10th fold. This process is repeated so that each fold is used as a testing set once. This gives us a better estimate of our accuracy since we can run several different test scenarios without requiring additional data and use those test results to optimize hyperparameter values. We once again stratify the folds on the outcome variable to ensure an equal distribution.

```
set.seed(42)
earthquakes_folds <- vfold_cv(earthquakes_train, v = 10, strata = tsunami)
```

Model Fitting

Building a Recipe

Before we build our models, we can set up a recipe that preprocesses our data without having to manually do so for each model.

We will be using 14 predictors: `magnitude`, `felt`, `cdi`, `mmi`, `alert`, `tsunami`, `nst`, `dmin`, `rms`, `gap`, `depth`, `latitude`, `longitude`, and `distanceKM`.

We can also impute values here for `alert` by replacing all of the NA values with the mode. We then dummy-code all of the nominal predictors and center and scale all of the predictors.

```
earthquakes_recipe <- recipe(tsunami ~ ., data = earthquakes_train) %>%
  step_impute_mode(alert) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())
```

Model Building

For this project I chose to fit logistic regression, linear discriminant analysis, k-nearest neighbors, random forest, boosted tree, radial support vector machine, and linear support vector machine because they are commonly used in binary classification problems. Each model has a fairly similar building process that I will explain in detail below.

Setting Up Models and Workflows

First, we specify which model we want to use, which parameters to tune, and which mode and engine to use. Since we are working with a classification problem, each model has been set to “classification”. We then combine each model with our recipe to set up workflows that can process everything in one step.

```
# Logistic Model
log_spec <- logistic_reg() %>%
  set_mode("classification") %>%
  set_engine("glm")

log_wf <- workflow() %>%
  add_model(log_spec) %>%
  add_recipe(earthquakes_recipe)

# LDA
lda_spec <- discrim_linear() %>%
  set_mode("classification") %>%
  set_engine("MASS")

lda_wf <- workflow() %>%
  add_model(lda_spec) %>%
  add_recipe(earthquakes_recipe)

# KNN
knn_spec <- nearest_neighbor(neighbors = tune()) %>%
  set_mode("classification") %>%
  set_engine("kknn")

knn_wf <- workflow() %>%
  add_model(knn_spec) %>%
  add_recipe(earthquakes_recipe)

# Random Forest
rf_spec <- rand_forest(mtry = tune(),
                       trees = tune(),
                       min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

rf_wf <- workflow() %>%
  add_model(rf_spec) %>%
  add_recipe(earthquakes_recipe)

# Boosted Tree
bt_spec <- boost_tree(trees = tune(),
                      learn_rate = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

bt_wf <- workflow() %>%
  add_model(bt_spec) %>%
  add_recipe(earthquakes_recipe)
```

```r
# SVM with Radial Kernel
svm_rbf_spec <- svm_rbf(cost = tune()) %>%
  set_mode("classification") %>%
  set_engine("kernlab")

svm_rbf_wf <- workflow() %>%
  add_recipe(earthquakes_recipe) %>%
  add_model(svm_rbf_spec)

# SVM with Linear Kernel
svm_lin_spec <- svm_poly(degree = 1, cost = tune()) %>%
  set_mode("classification") %>%
  set_engine("kernlab")

svm_lin_wf <- workflow() %>%
  add_recipe(earthquakes_recipe) %>%
  add_model(svm_lin_spec)
```

Setting Up Tuning Grids

Certain models have hyperparameters that need to be tuned, or optimized. To do this, we set up a grid of potential values and use our validation folds to find which hyperparameter values perform best in each model. Note that the logistic and LDA models do not have any hyperparameters, and so do not need to be tuned. This can take a while to run, so I ran each tuning grid outside of the markdown file and loaded them. As an example of how a tuning grid is set up, I commented the tuning grid for KNN.

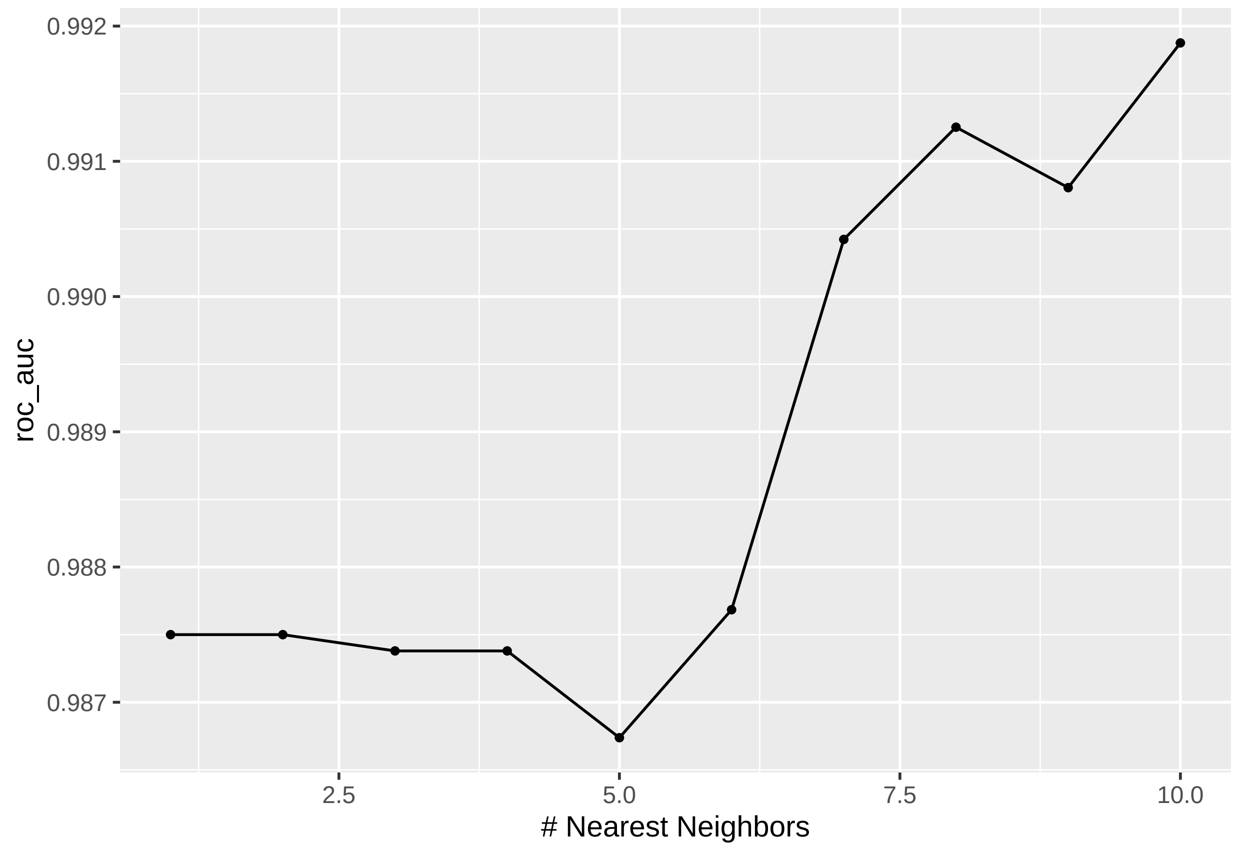
```r
# Logistic Model
# no tuning required

# LDA
# no tuning required

# KNN
knn_grid <- grid_regular(neighbors(range = c(1, 10)),
                         levels = 10)
load("tune_knn.rda")
#tune_knn <- tune_grid(knn_wf,
#                       resamples = earthquakes_folds,
#                       grid = knn_grid,
#                       metrics = metric_set(roc_auc))

# Random Forest
rf_grid <- grid_regular(mtry(range = c(1, 14)),
                        trees(range = c(100, 500)),
                        min_n(range = c(10, 20)),
                        levels = 14)
load("tune_rf.rda")

# Boosted Tree
bt_grid <- grid_regular(trees(range = c(10, 2000)),
                        learn_rate(range = c(0.1, 0.3)),
                        levels = 10)
```

```r
load("tune_bt.rda")

# SVM with Radial Kernel
svm_rbf_grid <- grid_regular(cost(), levels = 5)

load("tune_svm_rbf.rda")

# SVM with Linear Kernel
svm_lin_grid <- grid_regular(cost(), levels = 5)

load("tune_svm_lin.rda")
```

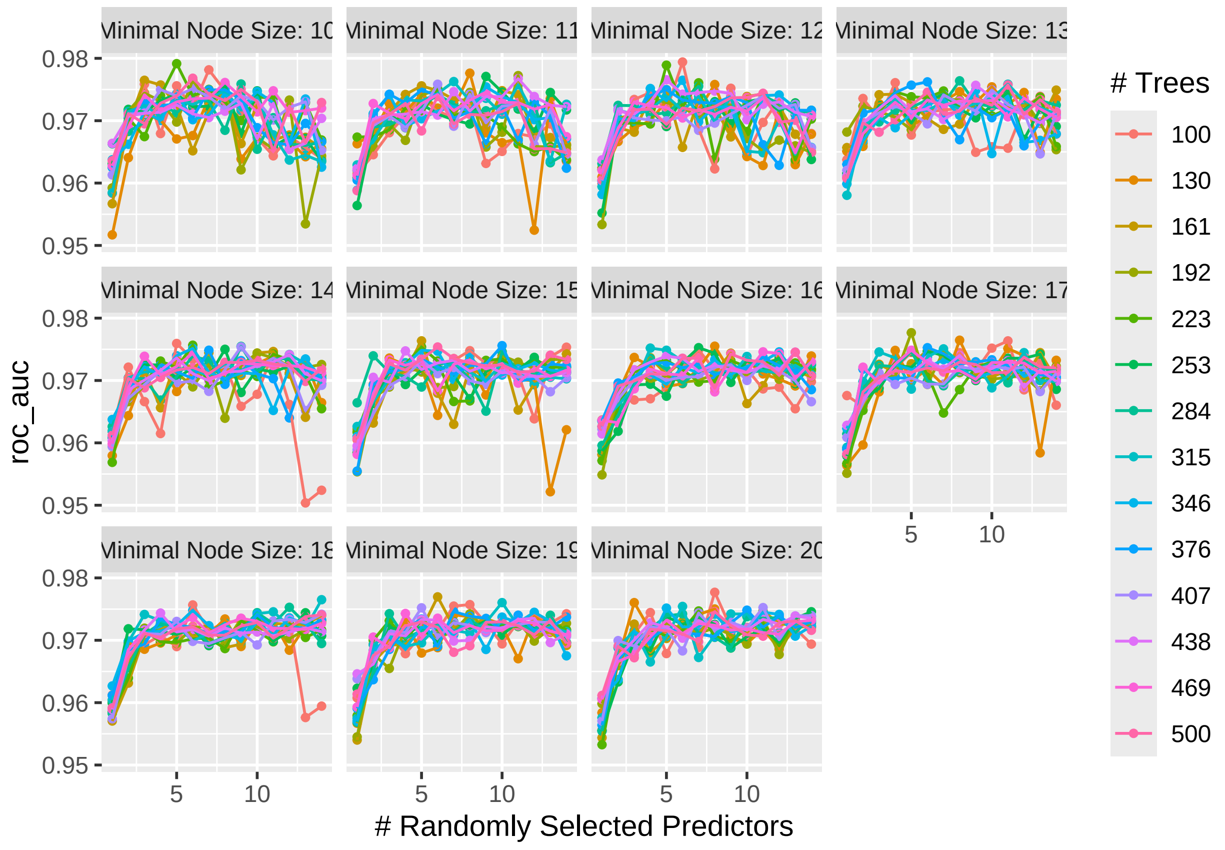
Tuning Autoplots

We can visualize the tuning process for each model using the `autoplot` function.

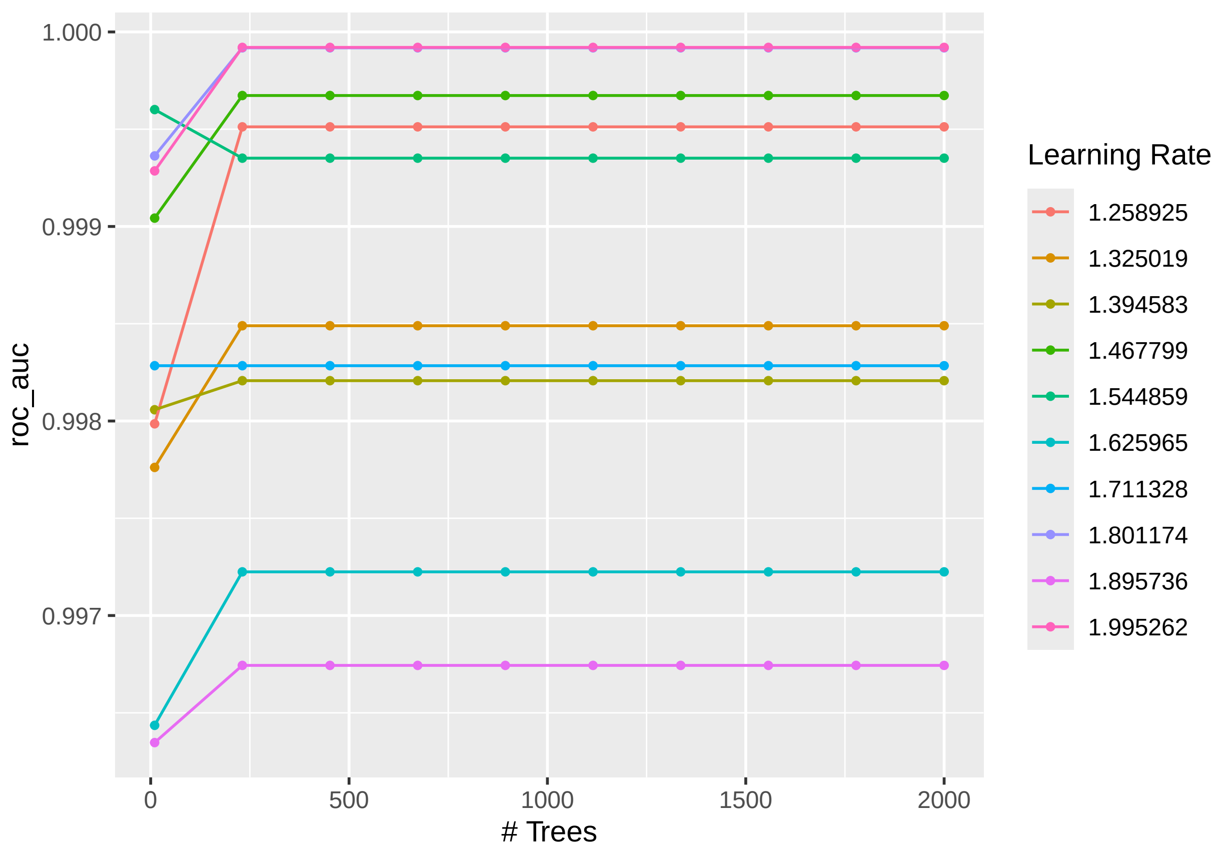
```r
# Logistic Model
# no tuning plot

# LDA
# no tuning plot

# KNN
autoplot(tune_knn)
```



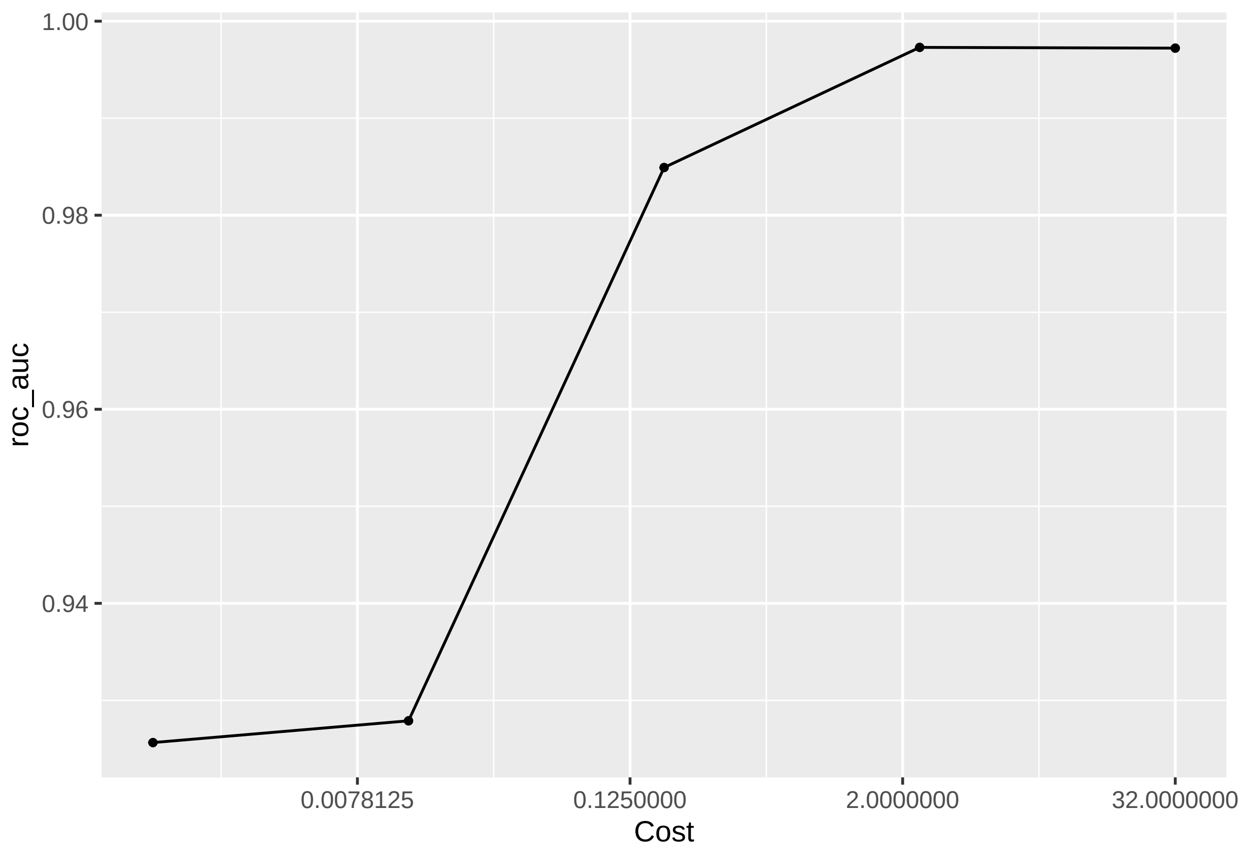
```
# Random Forest
autoplot(tune_rf)
```



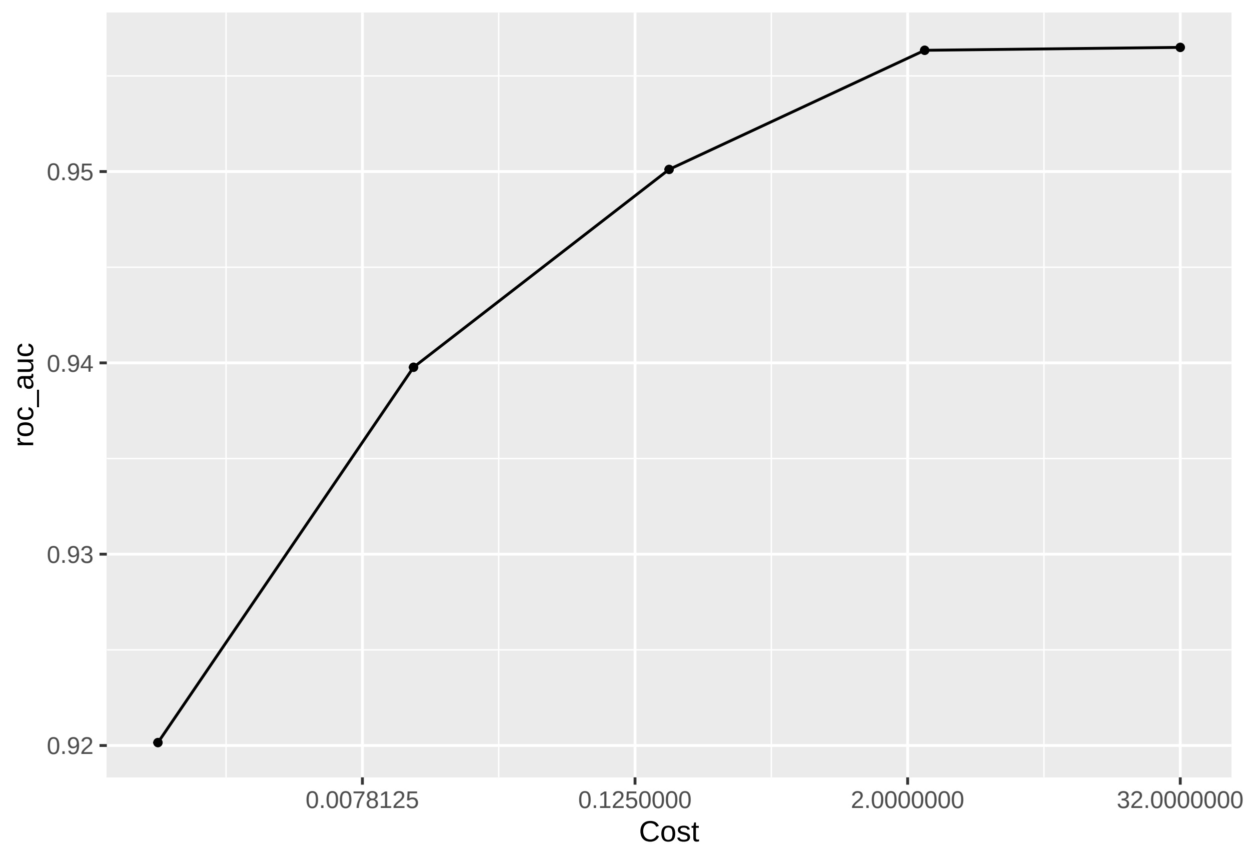
```
# Boosted Tree
autoplot(tune_bt)
```



```
# SVM with Radial Kernel
autoplot(tune_svm_rbf)
```



```r
# SVM with Linear Kernel
autoplot(tune_svm_lin)
```



Tuning Hyperparameters

Using the `select_best` function, we pick the best performing hyperparameter for each model.

```r
# Logistic Model
# no tuning

# LDA
# no tuning

# KNN
best_knn <- select_best(tune_knn, metric = "roc_auc")

# Random Forest
best_rf <- select_best(tune_rf, metric = "roc_auc")

# Boosted Tree
best_bt <- select_best(tune_bt, metric = "roc_auc")

# SVM with Radial Kernel
best_svm_rbf <- select_best(tune_svm_rbf, metric = "roc_auc")

# SVM with Linear Kernel
best_svm_lin <- select_best(tune_svm_lin, metric = "roc_auc")
```

Finalizing Workflows

We can then feed those tuned parameters back into our workflows to finalize them.

```r
# Logistic Model
# no need to finalize workflow

# LDA
# no need to finalize workflow

# KNN
knn_final_wf <- finalize_workflow(knn_wf, best_knn)

# Random Forest
rf_final_wf <- finalize_workflow(rf_wf, best_rf)

# Boosted Tree
bt_final_wf <- finalize_workflow(bt_wf, best_bt)

# SVM with Radial Kernel
svm_rbf_final_wf <- finalize_workflow(svm_rbf_wf, best_svm_rbf)

# SVM with Linear Kernel
svm_lin_final_wf <- finalize_workflow(svm_lin_wf, best_svm_lin)
```

Fitting Models

Now that our workflows are finalized, we can use them to fit each model to the training set.

```r
# Logistic Model
log_fit <- fit(log_wf, data = earthquakes_train)

# LDA
lda_fit <- fit(lda_wf, data = earthquakes_train)

# KNN
knn_final_fit <- fit(knn_final_wf, data = earthquakes_train)

# Random Forest
rf_final_fit <- fit(rf_final_wf, earthquakes_train)

# Boosted Tree
bt_final_fit <- fit(bt_final_wf, earthquakes_train)

# SVM with Radial Kernel
svm_rbf_final_fit <- fit(svm_rbf_final_wf, earthquakes_train)

# SVM with Linear Kernel
svm_lin_final_fit <- fit(svm_lin_final_wf, earthquakes_train)
```

Collecting Metrics

Our models have finally been fit! We can now evaluate them and visualize how well each model performed.

ROC Curves One way to visualize performance is a ROC (Receiver Operating Characteristic) curve, in which the true positive rate (sensitivity) is plotted against the false positive rate (1 - specificity) at different threshold values. The closer the ROC curve is to the top-left corner, the better the model is at distinguishing between positive and negative classes. A model with no discriminative power will have an ROC curve along the diagonal line, essentially equivalent to random guessing. All of our models have curves close to the top-left corner, which is a great sign!

```r
# Logistic Model
log_preds <- augment(log_fit, earthquakes_train)
r1 <- roc_curve(log_preds, tsunami, .pred_1, event_level = "second") %>%
  autoplot()

# LDA
lda_preds <- augment(lda_fit, earthquakes_train)
r2 <- roc_curve(lda_preds, truth = tsunami, .pred_1, event_level = "second") %>%
  autoplot()

# KNN
knn_preds <- augment(knn_final_fit, earthquakes_train)
r3 <- roc_curve(knn_preds, tsunami, .pred_1, event_level = "second") %>%
  autoplot()

# Random Forest
rf_preds <- augment(rf_final_fit, earthquakes_train)
r4 <- roc_curve(rf_preds, truth = tsunami, .pred_1, event_level = "second") %>%
  autoplot()

# Boosted Tree
bt_preds <- augment(bt_final_fit, earthquakes_train)
r5 <- roc_curve(bt_preds, tsunami, .pred_1, event_level = "second") %>%
  autoplot()

# SVM with Radial Kernel
svm_rbf_preds <- augment(svm_rbf_final_fit, earthquakes_train)
r6 <- roc_curve(svm_rbf_preds, tsunami, .pred_1, event_level = "second") %>%
  autoplot()

# SVM with Linear Kernel
svm_lin_preds <- augment(svm_lin_final_fit, earthquakes_train)
r7 <- roc_curve(svm_lin_preds, tsunami, .pred_1, event_level = "second") %>%
  autoplot()

grid.arrange(r1, r2, r3, r4, r5, r6, r7, nrow = 2)
```

Confusion Matrices Another way to visualize performance is using a confusion matrix. On the x-axis are the number of observations that are actually in each case, and on the y-axis are the number of observations in each case as predicted by the model. The top-left and bottom-right boxes (also in dark gray) are the number of accurately classified observations, and the bottom-left and top-right boxes (in light gray) are the number of misclassified observations. Overall, each model correctly classified a majority of the observations.

```r
# Logistic Model
c1 <- conf_mat(log_preds, truth = tsunami, .pred_class) %>%
```

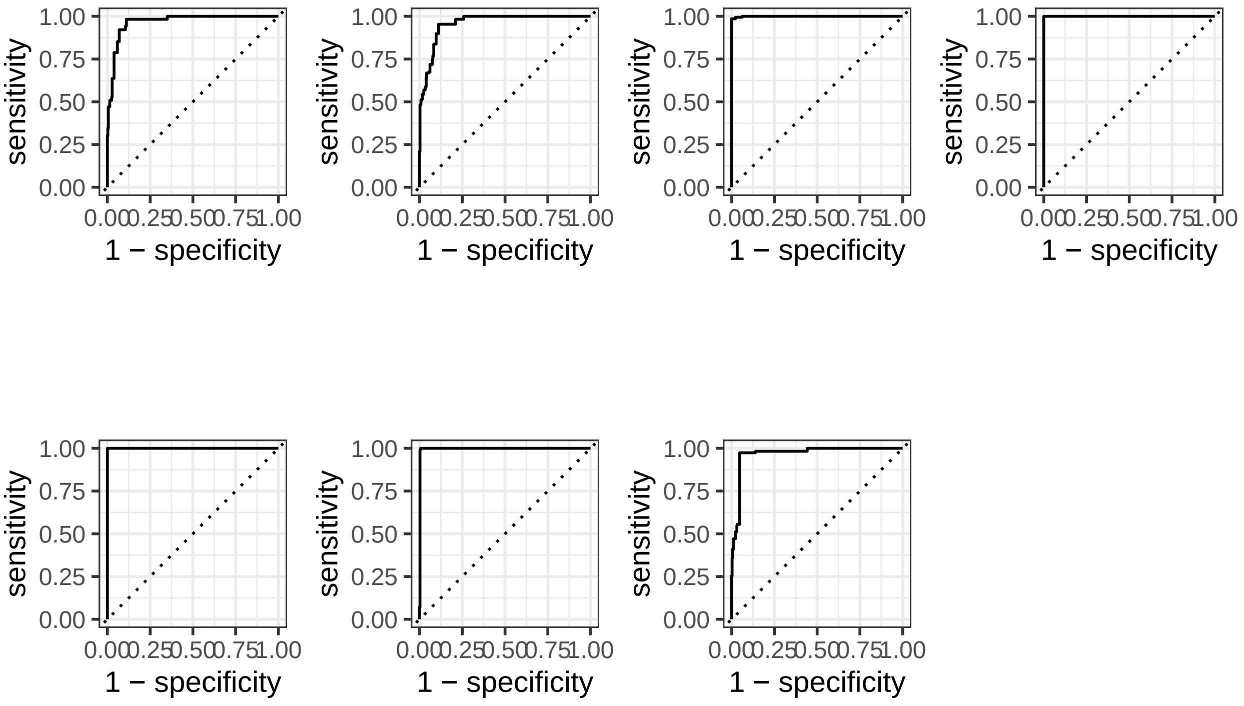


Figure 1: From top to bottom, left to right: logistic, LDA, KNN, RF, BT, Radial SVM, Linear SVM

```r
  autoplot(type = "heatmap")

# LDA
c2 <- conf_mat(lda_preds, truth = tsunami, .pred_class) %>%
  autoplot(type = "heatmap")

# KNN
c3 <- conf_mat(knn_preds, truth = tsunami, .pred_class) %>%
  autoplot(type = "heatmap")

# Random Forest
c4 <- conf_mat(rf_preds, truth = tsunami, .pred_class) %>%
  autoplot(type = "heatmap")

# Boosted Tree
c5 <- conf_mat(bt_preds, truth = tsunami, .pred_class) %>%
  autoplot(type = "heatmap")

# SVM with Radial Kernel
c6 <- conf_mat(svm_rbf_preds, truth = tsunami, .pred_class) %>%
  autoplot(type = "heatmap")

# SVM with Linear Kernel
c7 <- conf_mat(svm_lin_preds, truth = tsunami, .pred_class) %>%
  autoplot(type = "heatmap")

grid.arrange(c1, c2, c3, c4, c5, c6, c7, nrow = 2)
```

AUC The AUC, or area under the curve, is a metric for assessing how well a model can discriminate between cases. The AUC ranges from 0 to 1 with the following interpretations:

- $<$ 0.5: No discriminative power (equivalent to random guessing).
- 0.5 - 0.7: Poor discrimination.
- 0.7 - 0.8: Acceptable discrimination.
- 0.8 - 0.9: Excellent discrimination.
- $>$ 0.9: Outstanding discrimination.

We can create a table with all of the AUCs from each model and arrange them in descending order to compare them.

```r
model_aucs <- as.data.frame(rbind(
  Logistic = roc_auc(log_preds, truth = tsunami, .pred_1, event_level = "second")$.estimate,
  LDA = roc_auc(lda_preds, truth = tsunami, .pred_1, event_level = "second")$.estimate,
  KNN = roc_auc(knn_preds, truth = tsunami, .pred_1, event_level = "second")$.estimate,
  RF = roc_auc(rf_preds, truth = tsunami, .pred_1, event_level = "second")$.estimate,
  BT = roc_auc(bt_preds, truth = tsunami, .pred_1, event_level = "second")$.estimate,
  SVM_RBF = roc_auc(svm_rbf_preds, truth = tsunami, .pred_1, event_level = "second")$.estimate,
  SVM_Lin = roc_auc(svm_lin_preds, truth = tsunami, .pred_1, event_level = "second")$.estimate))
colnames(model_aucs) <- c("AUC")
model_aucs %>%
  arrange(desc(AUC))
```

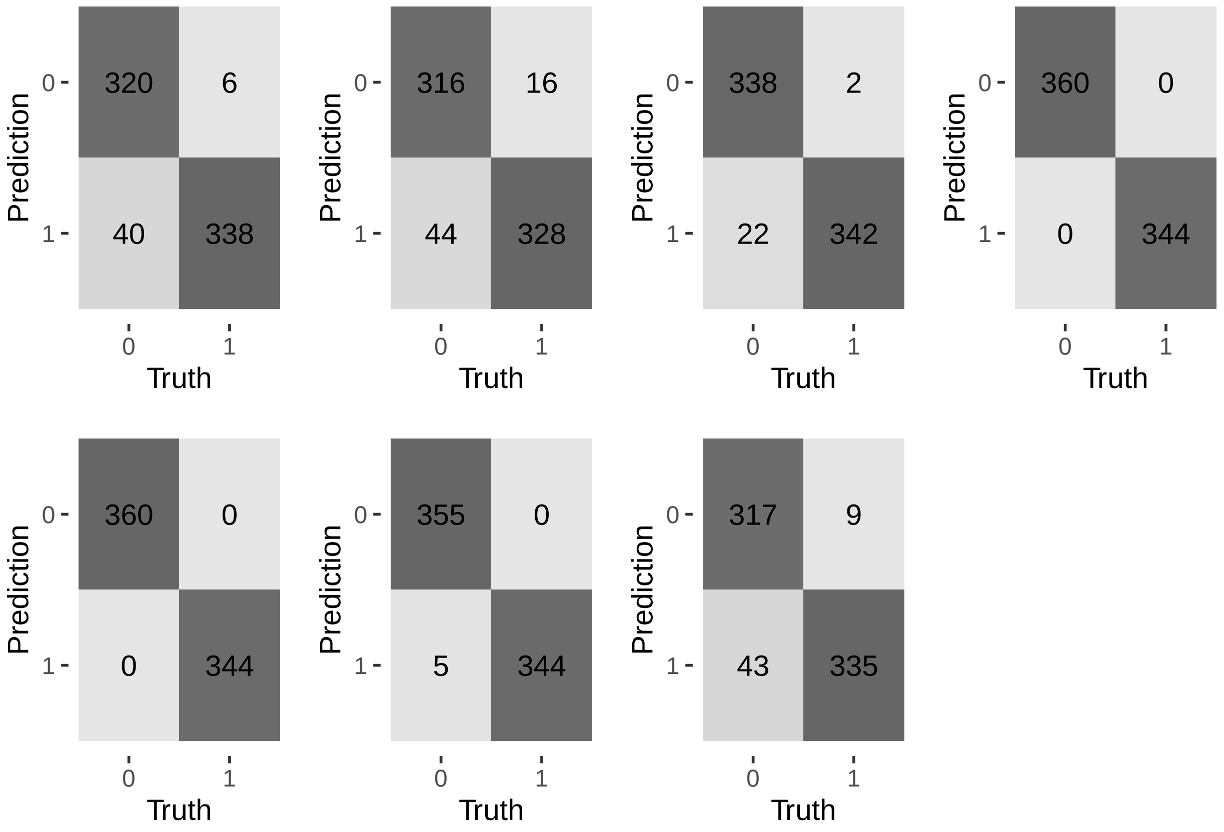


Figure 2: From top to bottom, left to right: logistic, LDA, KNN, RF, BT, Radial SVM, Linear SVM

```
##                AUC
## RF       1.0000000
## BT       1.0000000
## KNN      0.9994509
## SVM_RBF  0.9973918
## Logistic 0.9679264
## SVM_Lin  0.9678214
## LDA      0.9587048
```

All of our models performed pretty well! Based on these results, our top two performing models were boosted tree and random forest which both had perfect AUCs of 1. We can now fit those to the testing data.

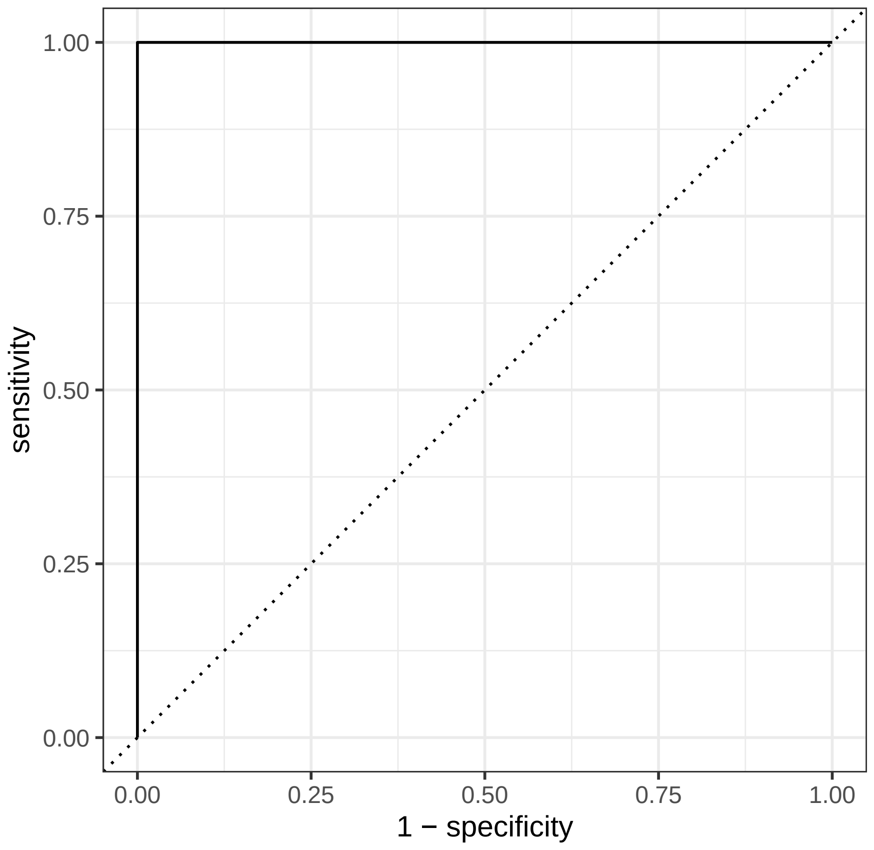
Testing Models

Now that we have our top two models, we can analyze their performance on the testing data. This is important to ensure that our models are actually accurate and not just overfitted to the training data. Since the testing data was not used for training, it provides a new set of observations for the model to predict, allowing us to better evaluate its performance.

Boosted Tree

The second-best performing model was boosted tree. This is not much of a surprise as boosted tree is a pretty powerful model. Since it is an ensemble method, it is generally good at accounting for complex patterns and is not as susceptible to noise in the data. We can use our previous metrics to gauge its performance on the test data.

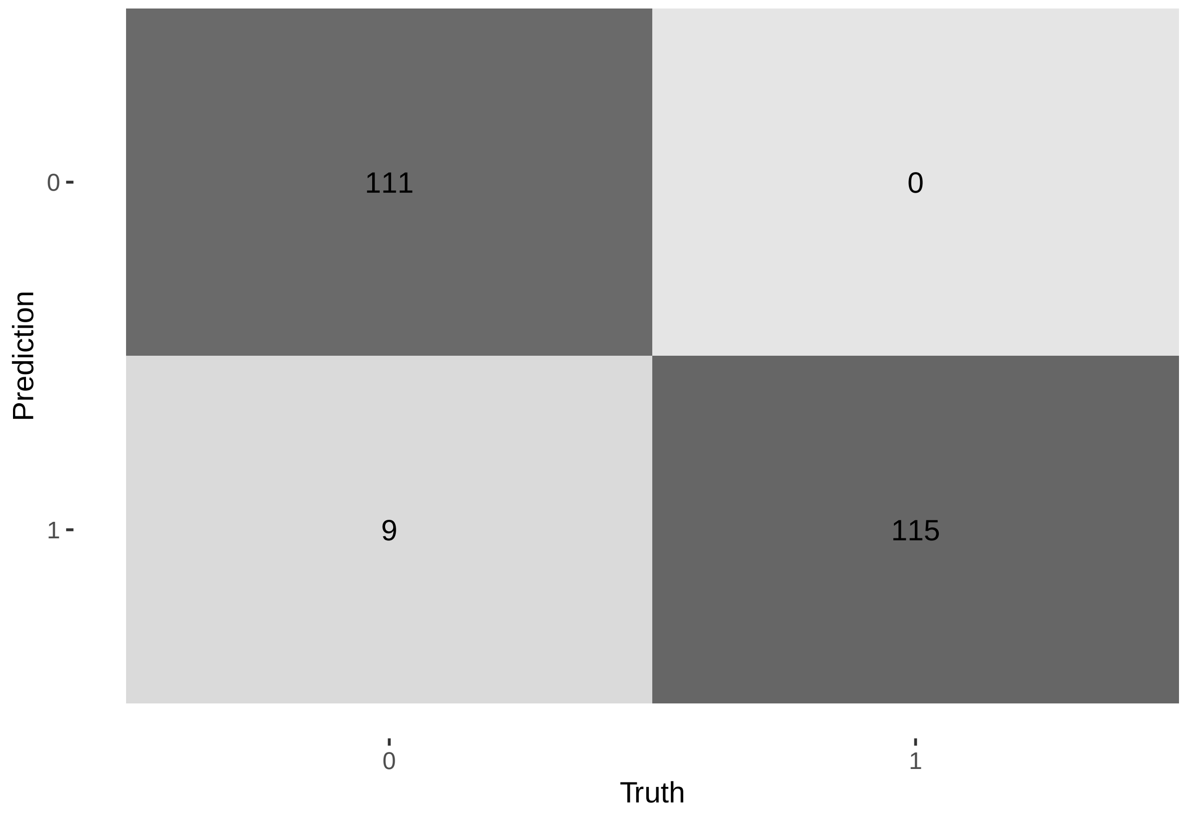
```r
bt_final_preds <- augment(bt_final_fit, earthquakes_test)

roc_curve(bt_final_preds, tsunami, .pred_1, event_level = "second") %>%
  autoplot()
```



```r
roc_auc(bt_final_preds, tsunami, .pred_1, event_level = "second")$.estimate
```

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

```r
conf_mat(bt_final_preds, truth = tsunami, .pred_class) %>%
  autoplot(type = "heatmap")
```

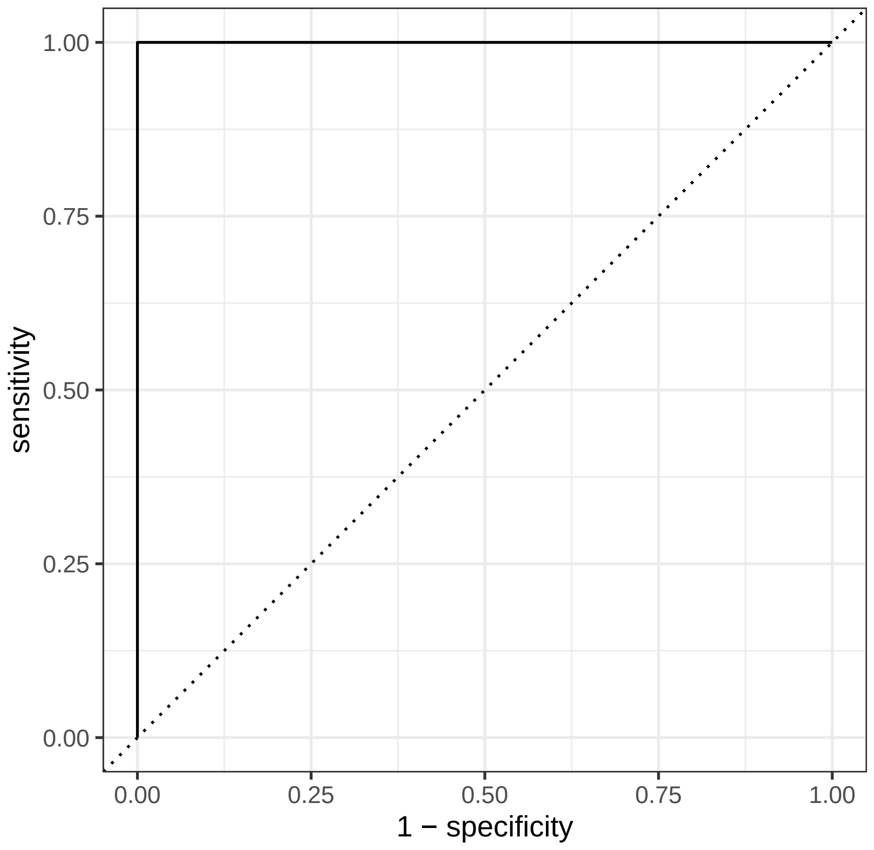


The model performed extremely well on the testing data, with an AUC of 1 and high accuracy! This is a great sign that our model is indeed useful and generalizable to new scenarios.

Random Forest

Last but most certainly not least is our top performing model: random forest. This again is not much of a surprise as random forest is also an ensemble method and is well-regarded for its flexibility and accuracy. We can once again evaluate its ROC curve, AUC, and confusion matrix.

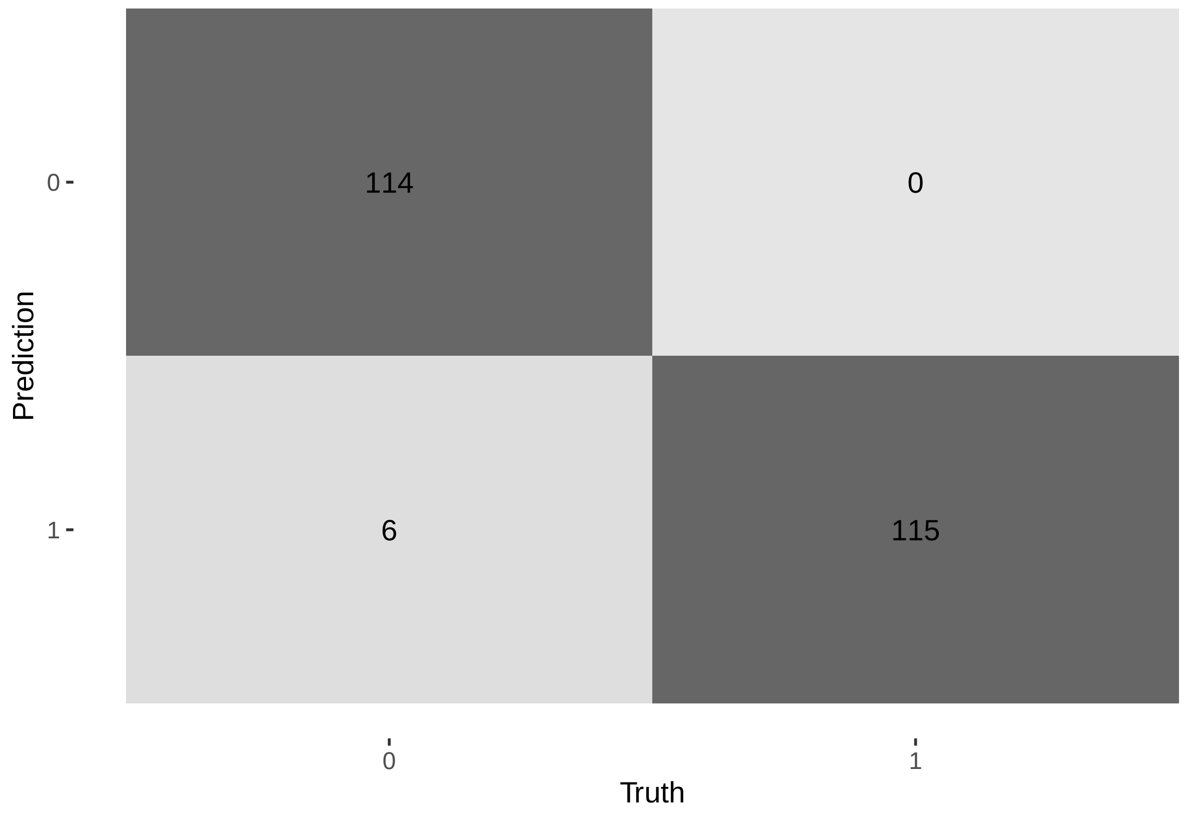
```r
rf_final_preds <- augment(rf_final_fit, earthquakes_test)

roc_curve(rf_final_preds, tsunami, .pred_1, event_level = "second") %>%
  autoplot()
```



```
roc_auc(rf_final_preds, tsunami, .pred_1, event_level = "second")$.estimate
```

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

```
conf_mat(rf_final_preds, truth = tsunami, .pred_class) %>%
  autoplot(type = "heatmap")
```

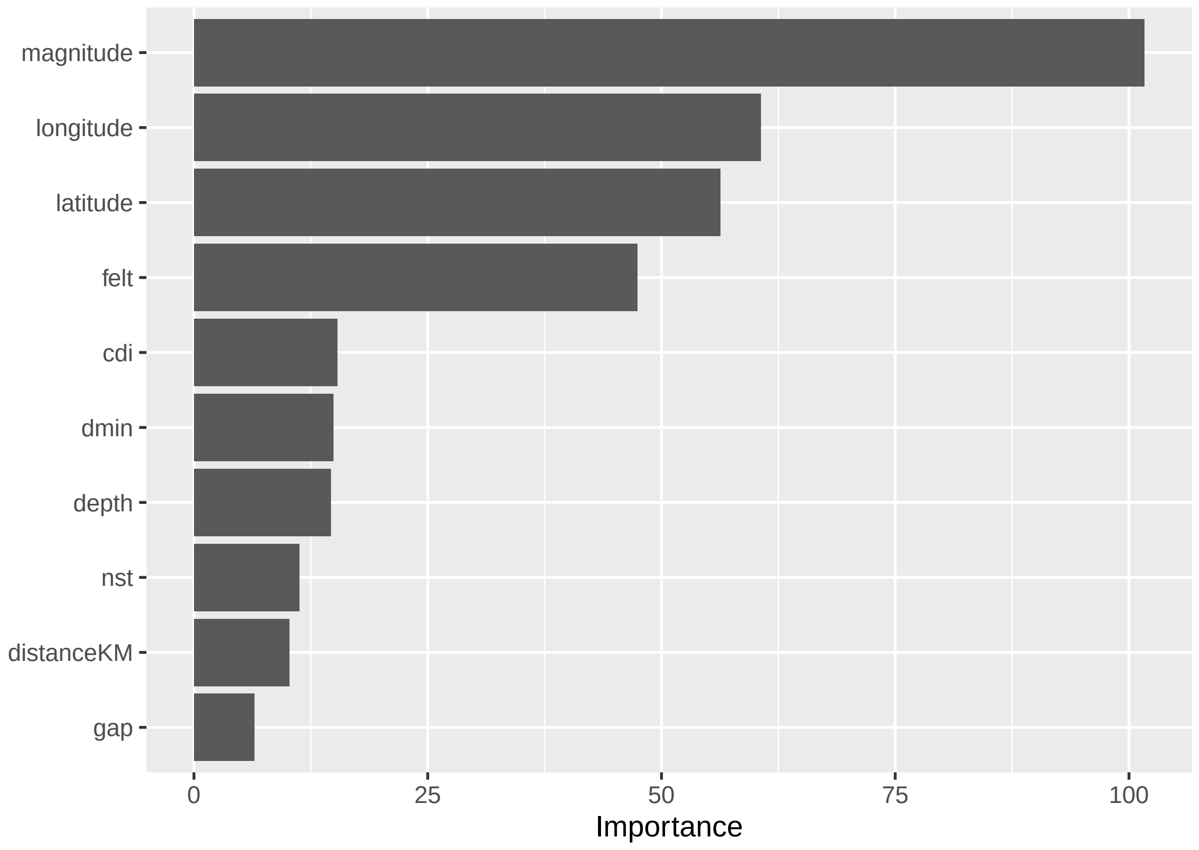


The model had a slightly lower AUC but higher accuracy than boosted tree. It still performed exceptionally well, which bodes well for generalizability and future applications.

Variable Importance Plot

Random forests have the additional benefit of determining which variables were most useful in predicting the outcome. As we suspected during EDA, `magnitude` was quite helpful in determining tsunami risk. `longitude`, `latitude`, and `felt` also proved influential in the final model. This makes sense as the NOAA states that location, magnitude, and depth are key contributors to tsunami generation.

```r
rf_final_fit %>%
  extract_fit_engine() %>%
  vip()
```



Conclusion

After exploring, tidying, and preprocessing our data, we fit 7 different classification models. Of those, random forest and boosted tree performed the best. Even among models that performed worse, the lowest AUC was 0.9584, indicating that all of the models we fit were good classifiers.

As expected, random forest and boosted tree performed the best out of all models. Both of these models are well-known for their flexibility and robustness, although they are quite computationally expensive. In our case, it worked out since our dataset was small. However, with larger datasets this might lead to complications when tuning and fitting models.

The worst performing model was LDA, although in isolation it performed relatively well. LDA relies on the predictors being normally distributed given the target and the data being linearly separable - that is, a linear decision boundary can be used to distinguish between cases. Our predictors might not have been normally distributed and our data might not have been entirely linearly separable, leading to a lower AUC.

In the future, I think it would be interesting to create a model that could quantify the risk of a tsunami. Although knowing whether or not a risk exists is still useful, a numerical probability might be more informative in fully understanding the risk and responding accordingly. However, that would require a much more detailed dataset and more domain knowledge regarding how tsunami risk can be quantified.

Overall, this project was a great opportunity to practice and build my machine learning skills, as well as gain experience with a subject of interest and relevance to me. As of writing this, a 7.0 magnitude earthquake in Northern California triggered tsunami warnings for those along the California coast, including my family in the Bay Area. My hope is that eventually tsunami prediction models will improve drastically, allowing those most vulnerable to their effects more time to prepare.

Data Source

Sur, S., & Every Earthquake API. (2024). *Global Earthquake Data* [Dataset]. Kaggle. https://www.kaggle.com/datasets/shreyasur965/recent-earthquakes/data