Introduction

Data Cleaning

Exploratory Data Analysis

Model Building

Running the Models

Finding New Models

Fitting the Testing Sett

Conclusion and Final Remarks

Code ▾

Final Project

Edward Ho

2022-06-10

Introduction

My project will outline predictor variables of interest that could potentially predict the severity of a car accident.

What's in this dataset? What is it?

To begin, the data set is a countrywide record of car accidents from 49 different states in the United States. The data set covers a wide time frame where it tracks monthly and daily data spanning from 2016,2017, and 2021. It's important to note that the data was collected from various sources—such as the US and state departments of transportation, law enforcement agencies, traffic cameras, traffic sensors, etc.; so there's a wide variety of data, with over 1.5 accidents, or observations, that have been recorded with the highest frequencies of recorded accidents being from the state of California. This model contain various variables, but most notably, one variable contained is the severity attribute which spans from 1-4, where 1 is the lowest severity and 4 is the highest. This severity is determined by it's impact on traffic, or the shortest delay on oncoming traffic as a result of the accident.

Why even create the model?

I'll list further details on the variables below, but essentially, the original data set contained nearly 47 different variables. All these variables levels in importance based on what one might be interested in predicting, but as I noted previously, I want to see what factors may be especially important in predicting severity of a certain accident. Many times we are forced to drive in various different weather conditions and usually we bat a blind eye towards the dangers that may result from these conditions. Weather conditions such as wind speed, weather conditions (rain, snow, etc.), night/day, temperature, and maybe even the presence of certain road factors (roundabouts, cross ways, etc.) may be important to know the relationship of these accidents and how they might vary in severity to potentially raise awareness to future drivers.

Loading Data and Packages

The project was found through Kaggle, which contained nearly 1.5 million observations in the data. Therefore, I wasn't able to import the raw data as a whole into R, therefore I must note that I did some light preliminary cleaning of the data set by filtering by only what I will be using as the subject of my analysis, 2021 California data. Therefore, after filtering out blank inputs, and grouping by 2021 and California exclusive cases, we now have a new data set that I can do further cleaning on. Let's clean the data a bit before displaying the full table and splitting the data for further analysis.

Hide

```
library(ggplot2)
library(readr)
library(tidymodels)
library(tidyverse)
library(ISLR)
library(ISLR2)
library(discrim)
library(janitor)
library(poissonreg)
library(pROC)
library(corrr)
library(corrplot)
library(purrr)
library(caret)
library(dplyr)
library(binom)
library(themis)
library(randomForest)
library(parsnip)
library(rpart)
library(rpart.plot)
library(ranger)
library(vip)
library(xgboost)
library(kknn)
library(rpart)
```

Hide

```
accidents_data <- read_csv("data/accidents1.csv")
```

Data Cleaning

Below, we will be removing some unneeded variables, renaming some columns, and other techniques for needed clarity in reading the data. Note, there is no direct need to remove missing/blank observations since I have already made sure they were not included through excel and before importing the csv, but for certainty, I'll rerun the process and check again below.

- Check for any remaining na's

Hide

```
accidents <- na.omit(accidents_data)
sum(is.na(accidents))
```

```
## [1] 0
```

- Remove the start_time and state column since I had already filtered that completely in excel. In most cases, wind direction seems pretty negligable as well so it wouldn't be something worth much towards severity in my opinion.

Hide

```
accidents <- accidents %>%
  select(-Start_Time,-State,-Wind_Direction)
```

- Cleaning the variable names for clarity

Hide

```
accidents <- janitor::clean_names(accidents)
```

- Now I must factor encode all the variables that are boolean values.

Hide

```
accidents$severity <- factor(accidents$severity)
accidents$side <- factor(accidents$side)
accidents$weather_condition <- factor(accidents$weather_condition)
accidents$bump <- factor(accidents$bump)
accidents$crossing <- factor(accidents$crossing)
accidents$railway <- factor(accidents$railway)
accidents$stop <- factor(accidents$stop)
accidents$traffic_signal <- factor(accidents$traffic_signal)
accidents$sunrise_sunset <- factor(accidents$sunrise_sunset)
```

Now the data is cleaned, and we can proceed with the analysis

Variables

Below are noteworthy variables that will be used in our analysis (note that there are 17 variables, but I chose only select few important ones to display here. A full, detailed explanation of each variable can be found in the codebook in the zip file.)

Table 1: Variable Details

Variable Name	Description	Type

Variable Name	Description	Type
severity	Shows the severity of the accident, a number between 1 and 4, where 1 indicates the least impact on traffic (i.e., short delay as a result of the accident) and 4 indicates a significant impact on traffic (i.e., long delay).	Factor
Distance(mi)	The length of the road extent affected by the accident	Numeric
visbility_mi	Shows visibility (in miles).	Numeric
wind_speed_mph	Shows wind speed (in miles per hour).	Numeric
precipitation_in	Shows precipitation amount in inches, if there is any.	Numeric
weather_condition	Shows the weather condition (rain, snow, thunderstorm, fog, etc.)	Factor
crossing	Presence of crossing in a nearby location.	Factor
stop	Presence of stop in a nearby location.	Factor
traffic_signal	Presence of traffic signal in a nearby location.	Factor
sunrise_sunset	hows the period of day (i.e. day or night) based on sunrise/sunset.	Factor

Data Split

I split my data using a proportion of 70/30, where the training data is 70% of my total data, and the testing data covers the other 30%. Because I had a large imbalance in the response variable, severity, then I must use stratified sampling (which you can see in the EDA portion).

Hide

```
accidents_split <- initial_split(accidents, prop = 0.70,strata = severity)
accidents_train <- training(accidents_split)
accidents_test <- testing(accidents_split)

# save(accidents,accidents_recipe,accidents_train,file = "R_Scripts/processed\\accidents.r
da")
# save(accidents_train,file = "R_Scripts/processed\\accidents_train.rda")
# save(accidents_test,file = "R_Scripts/processed\\accidents_test.rda")
load("R_Scripts/processed\\accidents.rda")
load("R_Scripts/processed\\accidents_train.rda")
load("R_Scripts/processed\\accidents_test.rda")
```

Now I run a sanity check on the dimensions of the testing and training set to see if they match. And as we can see below, they do match in dimensions so there aren’t any issues here. Now let’s proceed to the next section, exploratory data analysis.

Hide

```
dim(accidents)
```

```
## [1] 30605    17
```

Hide

```
dim(accidents_train) + dim(accidents_test)
```

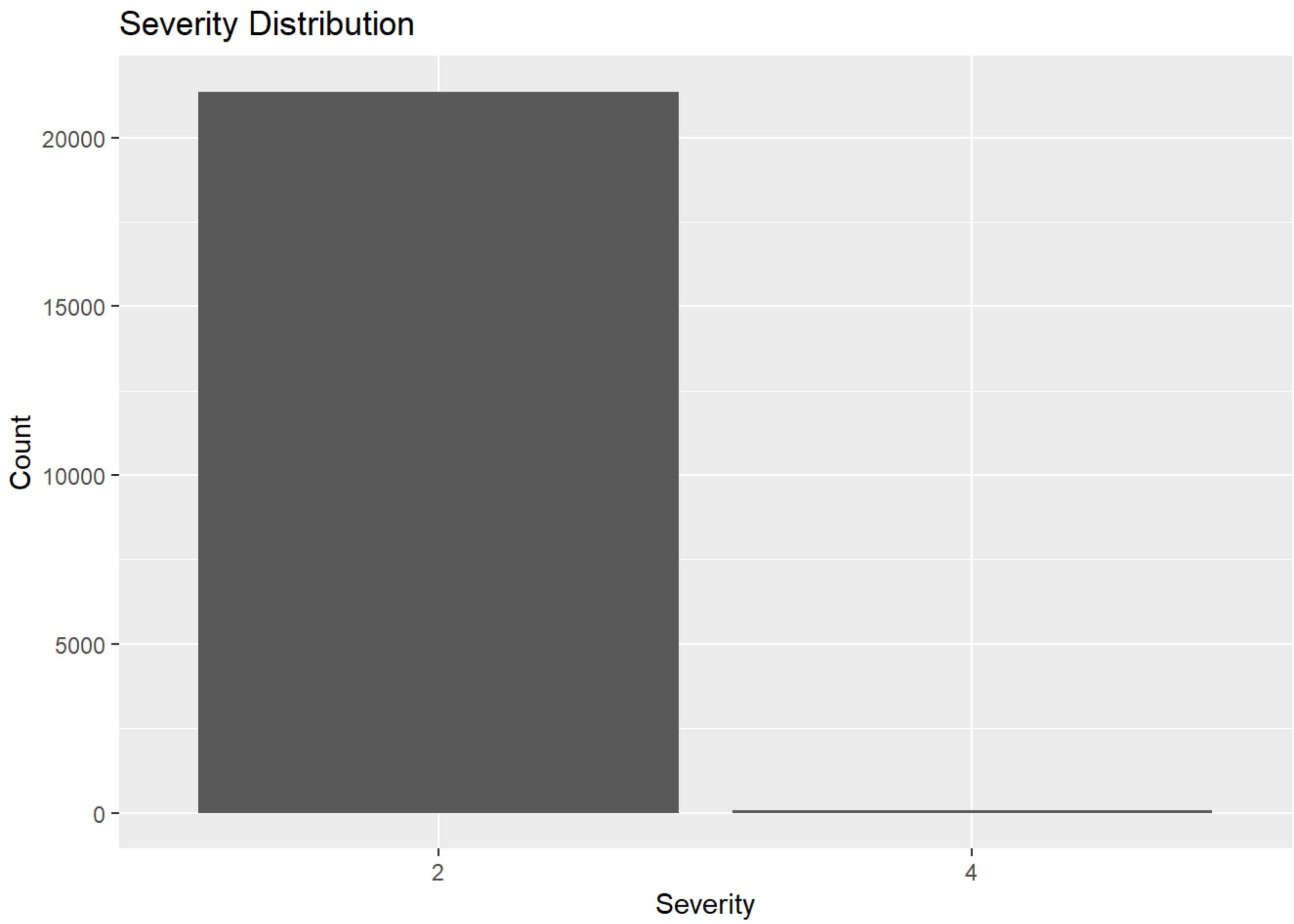
```
## [1] 30605    34
```

Exploratory Data Analysis

The proceeding section will be exploratory data analysis on the training set to find any potential patterns and expectations we might have before we proceed with fitting the models.

Severity Distribution

Code



Here is the proportion of the response variable with a severity of 4.

Hide

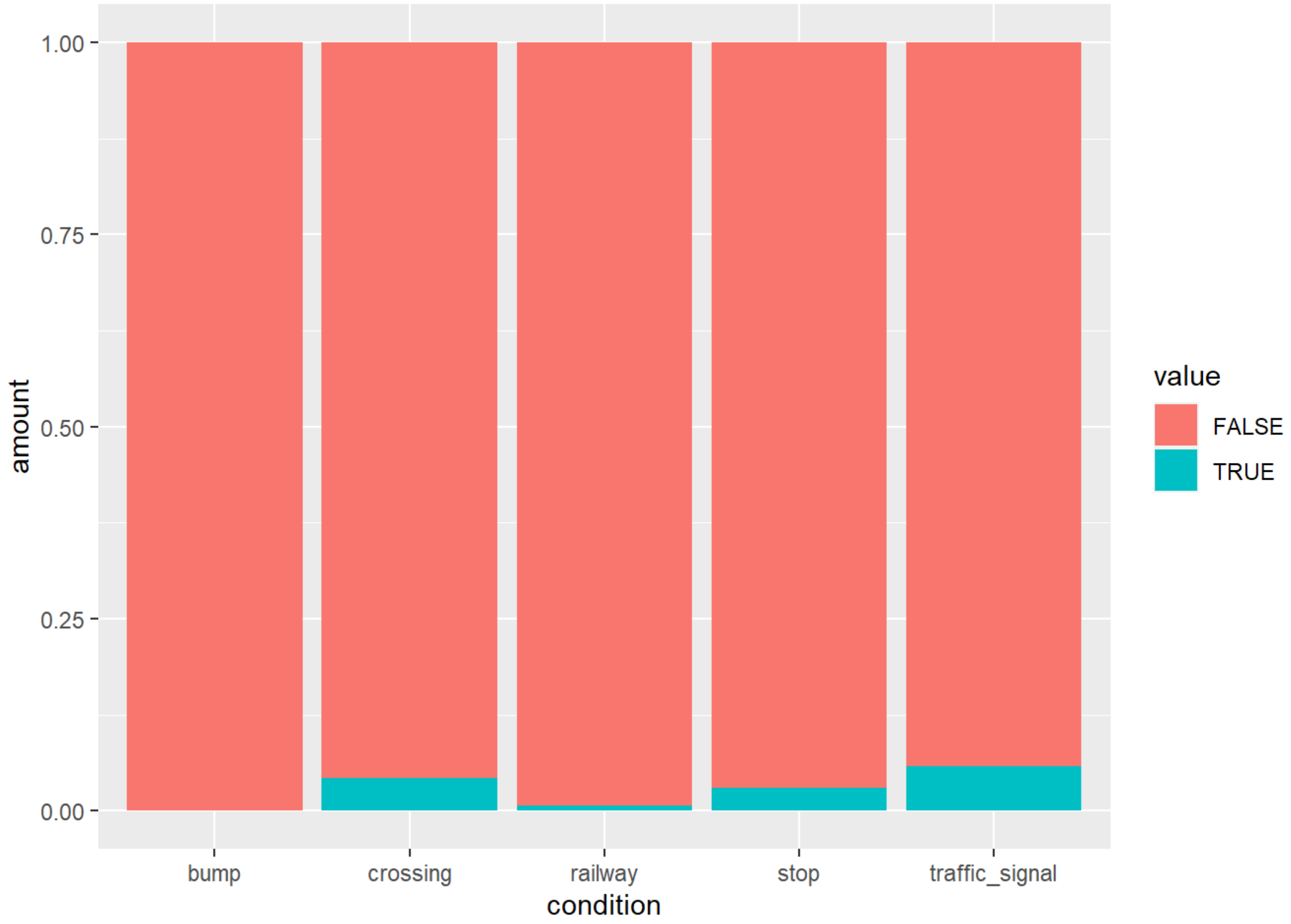
```
sum(accidents_train$severity=="4")/nrow(accidents_train)
```

```
## [1] 0.002894086
```

Notice that we find ourselves an extremely prevalent class unbalance in our response observations, therefore, something such as down sampling will be extremely valuable later on when creating the recipe. Also, notice how the response variables only contain 2 and 4 despite the range being from 1-4, therefore it appears all the accidents were either mild or extreme in terms of severity, so therefore our recipe will be a binary classification models.

Exploring the Location landmarks Factor Variables

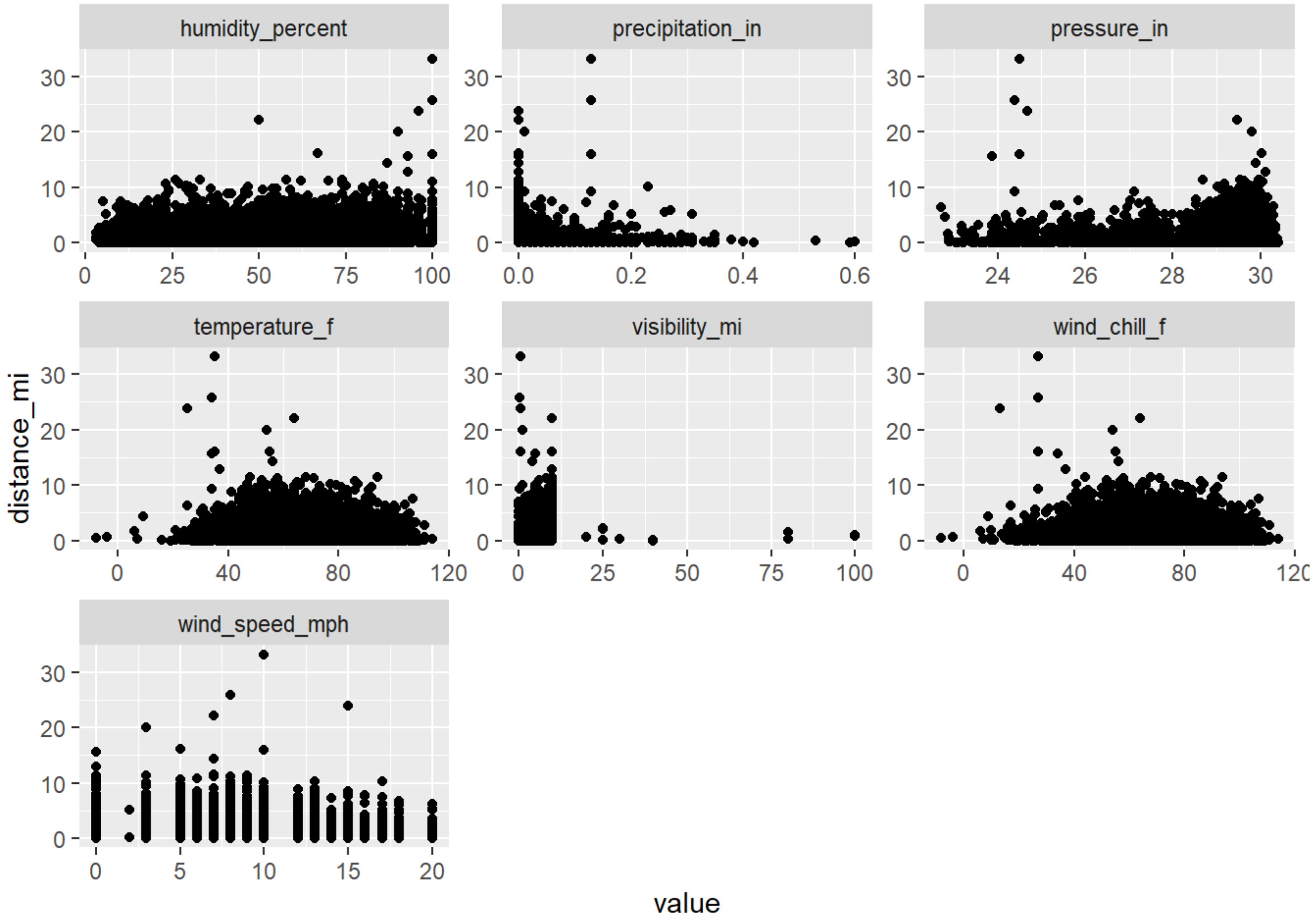
Code



The first thing we can notice already is the that crossing, traffic signals, and stop related traffic incidents are much more frequent than bumps and railway accidents. Therefore, we might be able to say that since there is a lower amount of railway and bump accidents in both categories, it might be two variables that won't probably be too important in determining the accidents. Additionally, it would make sense that these two factors are something that doesn't seem exclusive to severe accidents as well based off of experience. Being a daily driver myself, most accidents just generally tend to stray away from bumps and railways as they aren't populated parts of the road.

Distance Correlation with weather conditions?

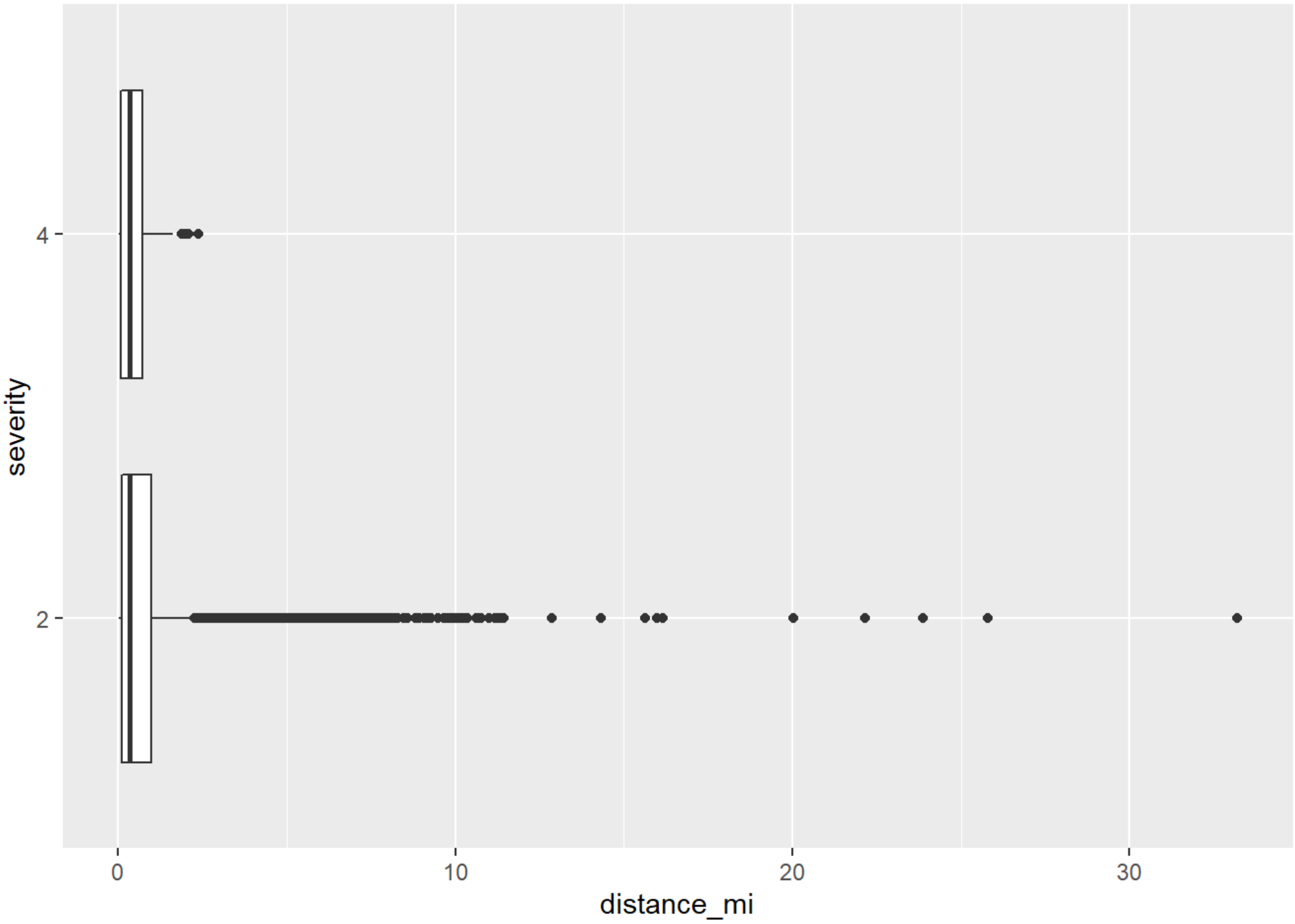
Code



As seen here, there doesn't seem to be any apparent correlation between the distance with any of the numerical weather conditions. Therefore, it doesn't seem as though weather conditions are too valuable in determining the severity of an accident IN TERMS OF STRICTLY distance, but it's important to note that distance isn't the only thing that accounts for severity, so we should keep that mind moving on.

Road Sidedness and distance_mi/visibility_mi

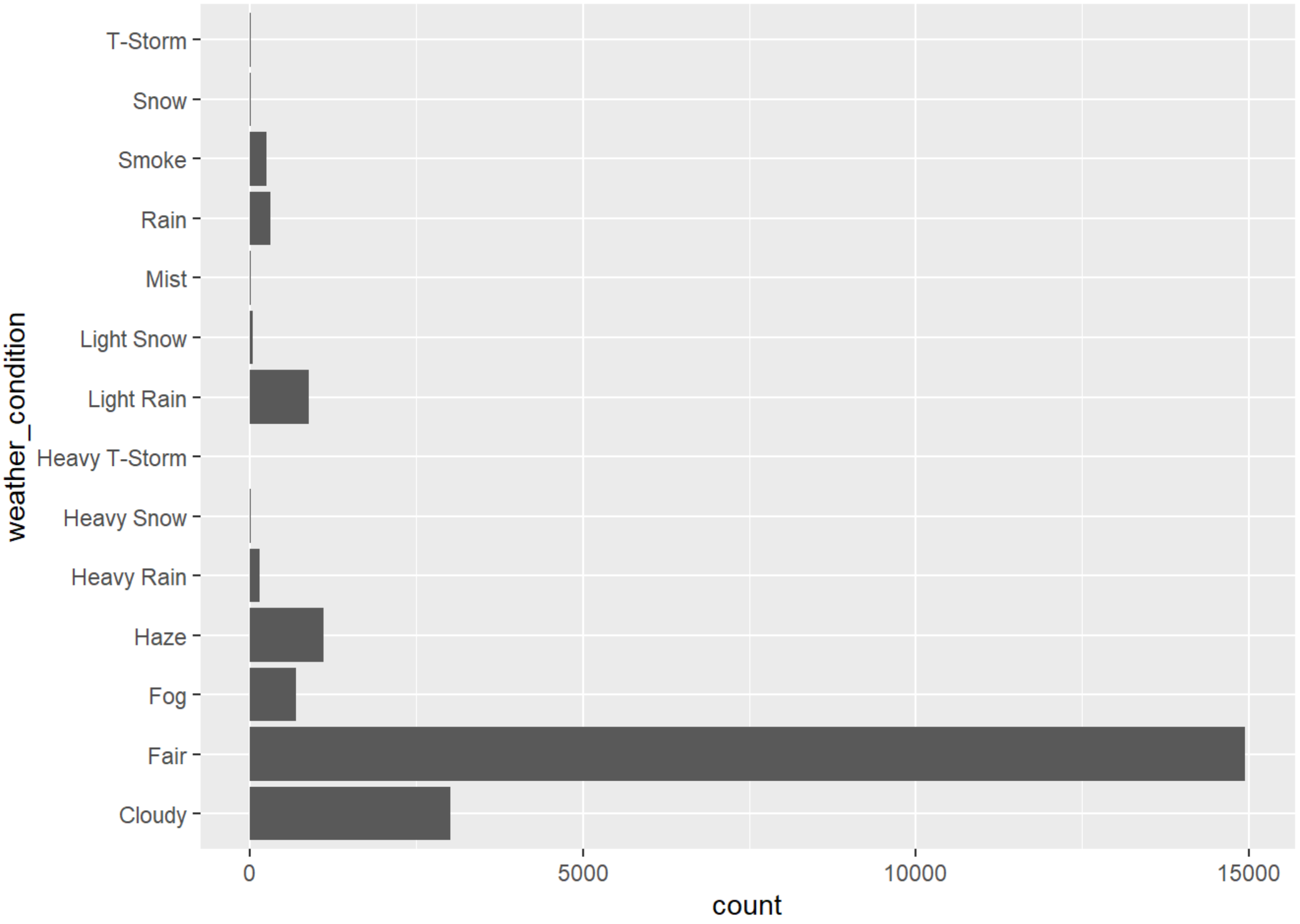
Code



As I noted previously, the severity of an accident is based largely on how much of traffic the accident ends up holding up. Therefore, we'd have to notions that we'd think that a longer distance of traffic that's held up by the accident would mean a higher severity and they would be correlated. We actually see that there appears to be no correlation but there are pretty extreme outliers in the severity 2 category, which might affect how if our models consider distance as significant or valuable, but remember, this is only a preliminary check and distance can certainly still be a significant variable.

Weather conditions

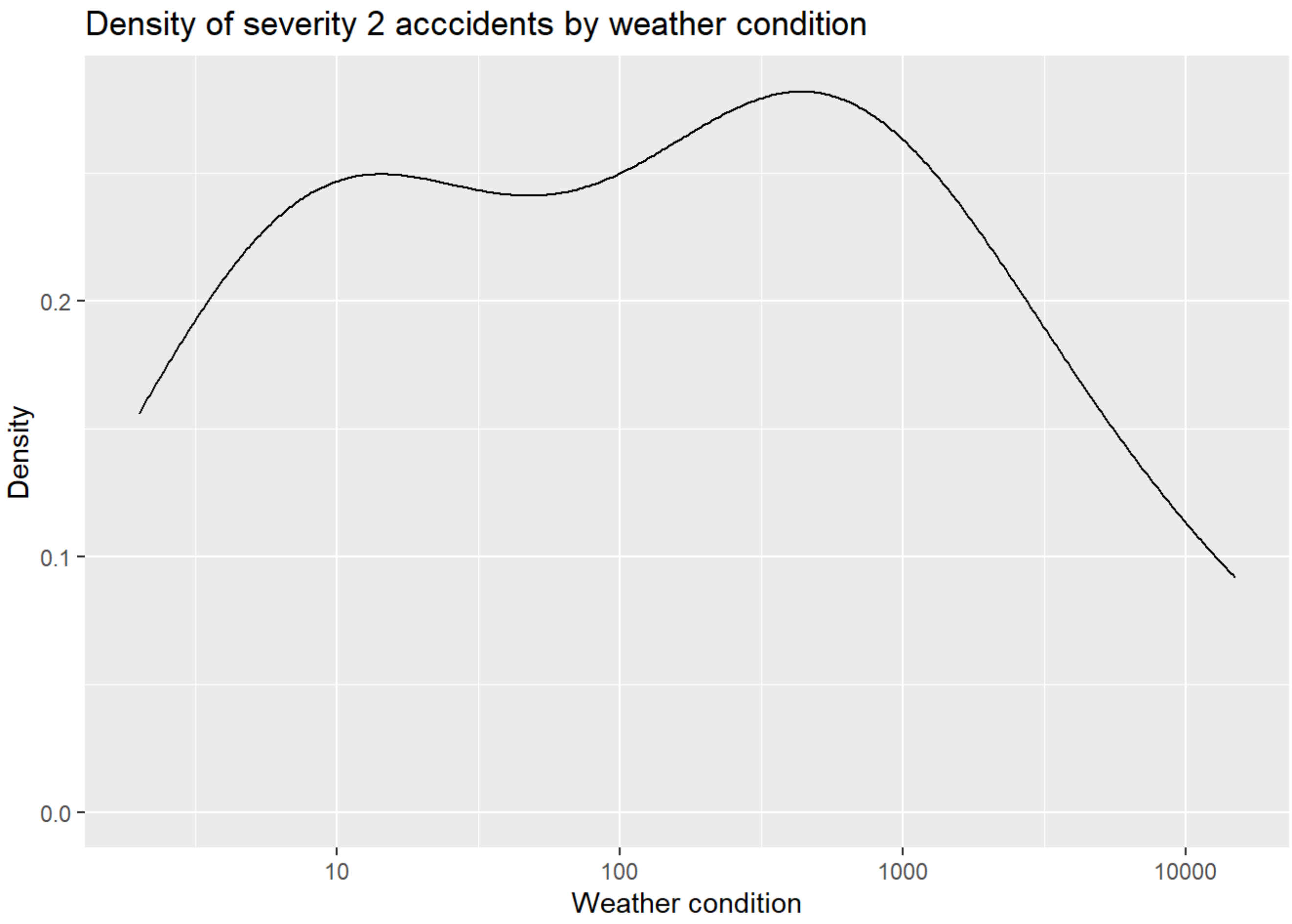
Code



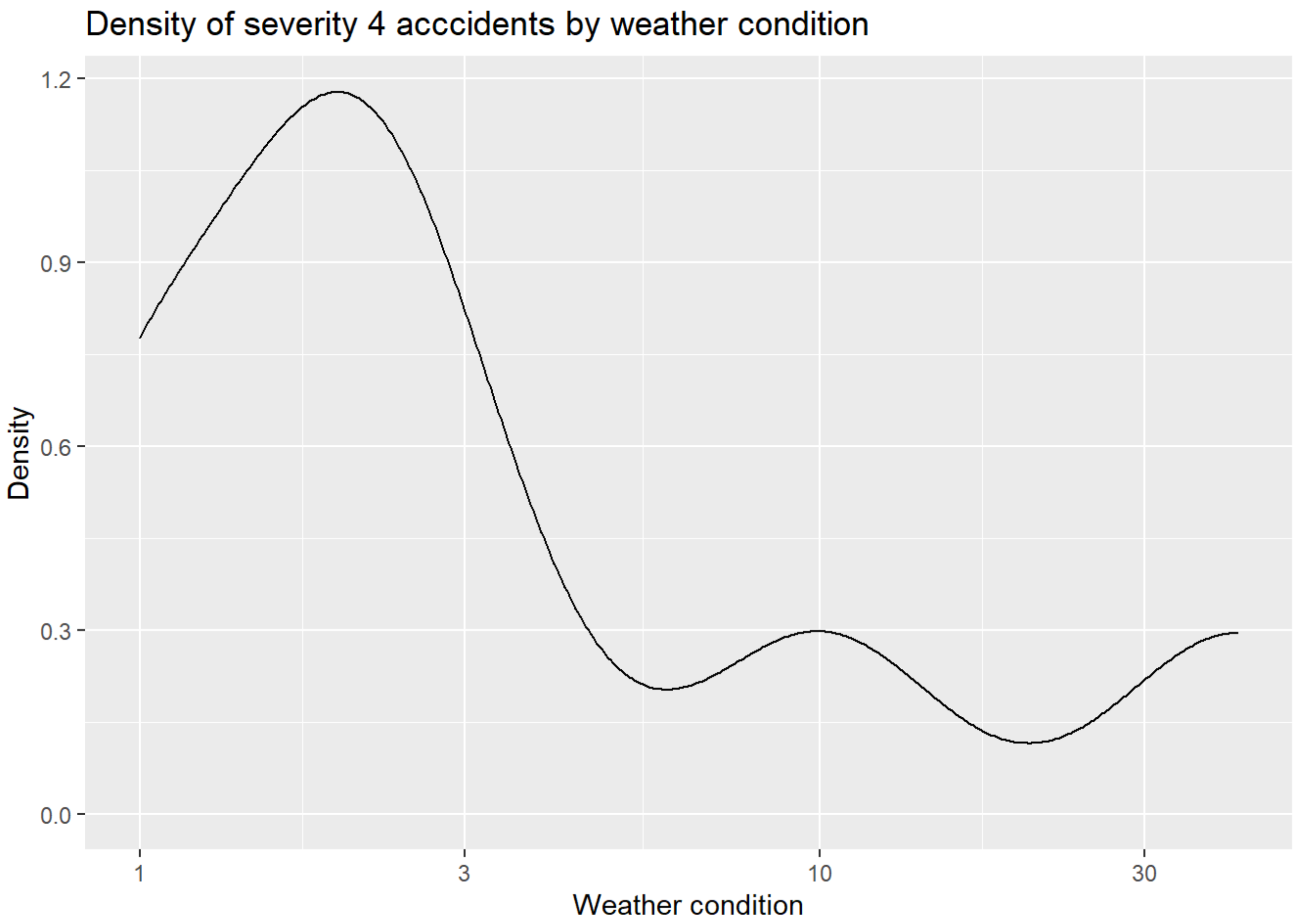
This graph helps display what’s considered to be the most abundant weather conditions at the time of the accident. This is important to display because abundances/large disparities between the appearance of certain weather conditions might highly skew the models ability to predict weather condition as valuable predictors. Additionally, large disparities in certain data values can mean it wont yield an accurate representation of what’s actually true, so therefore printing this is valuable because we should take our models conclusion with a grain of salt until we can get more data on low counts of certain conditions, such as mist, t-storm, snow, heavy t-storm, mist, and heavy snow.

Severity and Weather Condition

Code


Density of severity 2 acccidents by weather condition
Density
0.2
0.1
0.0
10
100
1000
10000
Weather condition

Code

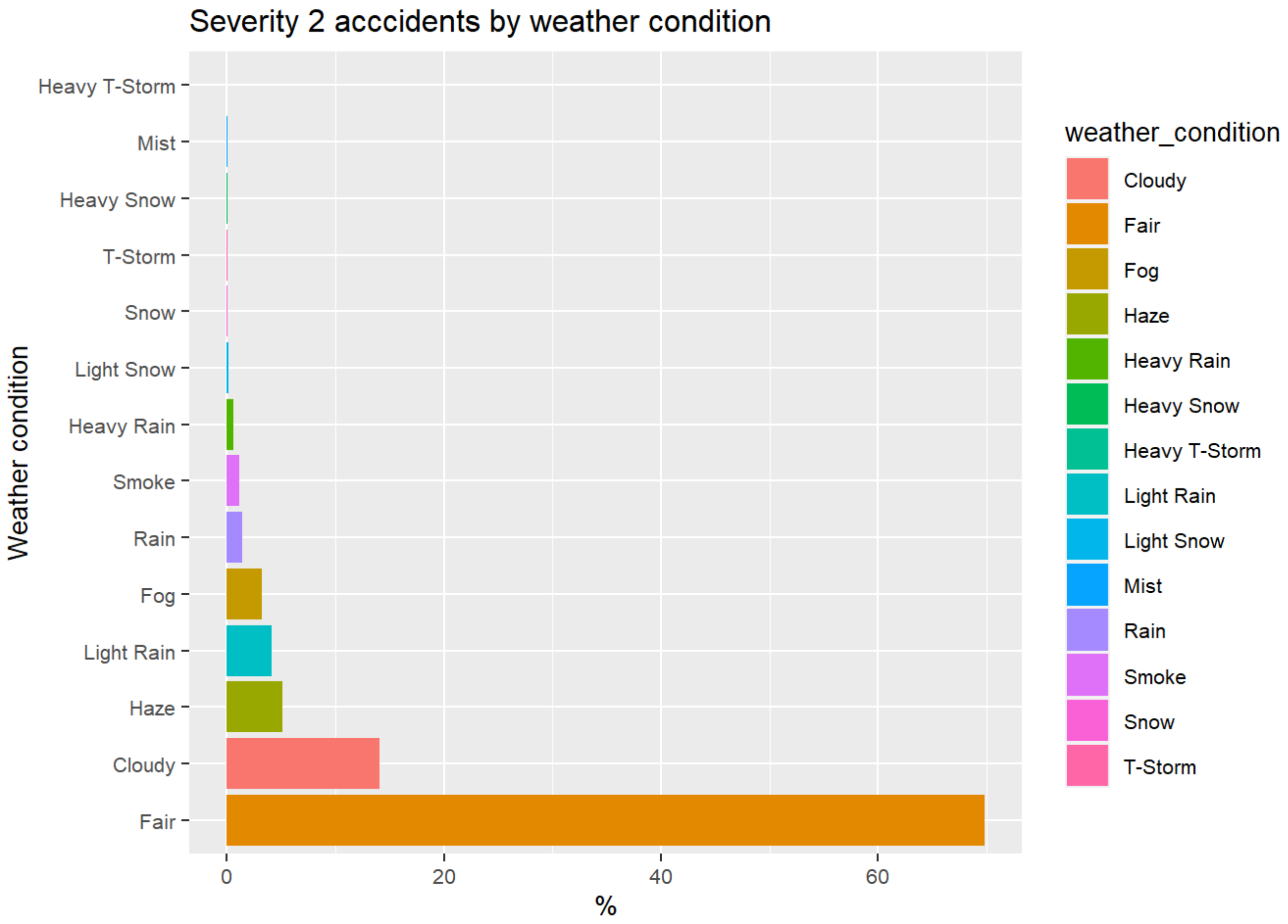

Density of severity 4 acccidents by weather condition

Essentially, what this graph is telling is that as the amount of different weather conditions increase in our data set, there are fewer and fewer severity 4 accidents. Basically, there are only a few weather conditions that actually account for severity 4 accidents. But on the other hand, there are quite a few weather conditions that account for our severity 2 accidents. Meaning that severity 4 has little diversity in weather conditions that describe the incidents, but there are a lot more weather condition diversity in severity 2 accidents which may be a good in using weather conditions to determine what the accident was. We can visual this better with the graphs below.

Code

```
## # A tibble: 14 × 2
##    weather_condition    count
##    <fct>                <dbl>
##  1 Fair              69.8    
##  2 Cloudy            14.0    
##  3 Haze               5.16   
##  4 Light Rain         4.12   
##  5 Fog                3.25   
##  6 Rain               1.41   
##  7 Smoke              1.18   
##  8 Heavy Rain         0.655  
##  9 Light Snow         0.192  
## 10 Snow               0.0796 
## 11 T-Storm            0.0421 
## 12 Heavy Snow         0.0375 
## 13 Mist               0.0281 
## 14 Heavy T-Storm      0.00936
```

Code

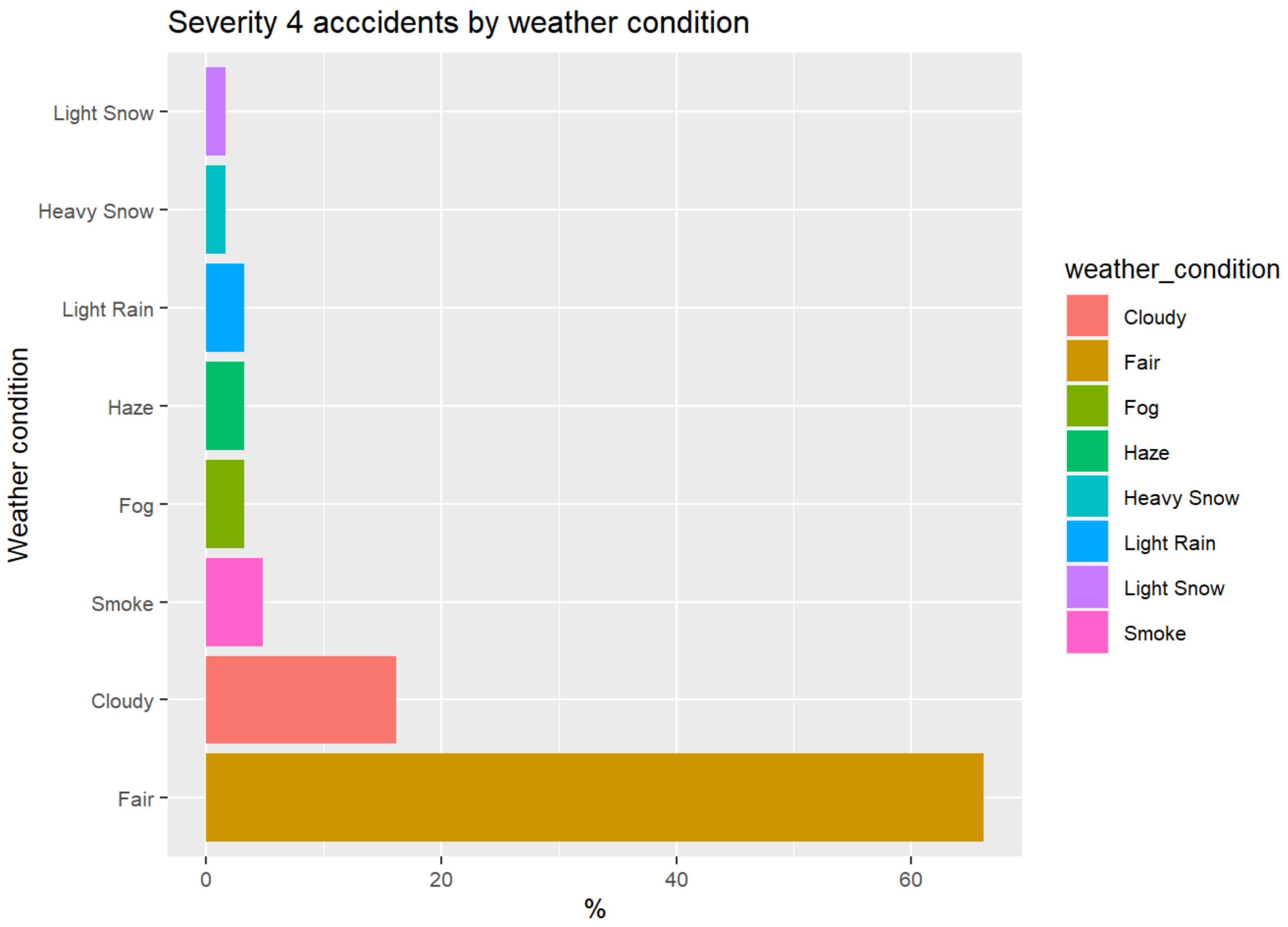


In this graphic, we numerically graph the types and amount of weather conditions in severity 2 incidents. Notice that there are 14 weather conditions that happened in severity 2 accidents, which is essentially all of the accident types.

Code

```
## # A tibble: 8 × 2
##   weather_condition count
##   <fct>             <dbl>
## 1 Fair              66.1
## 2 Cloudy            16.1
## 3 Smoke              4.84
## 4 Fog                3.23
## 5 Haze               3.23
## 6 Light Rain         3.23
## 7 Heavy Snow         1.61
## 8 Light Snow         1.61
```

Code



We build a similar model for our severity 4 accidents and you can already notice a difference. First, it’s important to note that the weather condition “fair” is most abundant in both severities which make sense since this is California only data, and California typically doesn’t experience an abundance of weather conditions. But what’s unique of the two severities is that severity 4 accidents only cover 7 different weather conditions. Again, this can be due to a lack of severity 4 representation in the data which is worth nothing, but nonetheless, it’s something that is very apparent between the two.

Model Building

Now that we have taken some idea of the distributions of our data, it is time to build our recipe's and models to hopefully achieve some predictive powers in determining "predictive" variables in predicting severity of California accidents, and to potentially find what's a good telling sign of what might be valuable in predicting what finds an accident's severity.

Recipe, Folds, and Saving the Data

Hide

```
accidents_folds <- vfold_cv(accidents_train, v=5, repeats = 4,strata=severity)

accidents_recipe <- recipe(severity ~ ., data = accidents_train) %>%
  step_dummy(side) %>%
  step_dummy(weather_condition) %>%
  step_dummy(bump) %>%
  step_dummy(crossing) %>%
  step_dummy(railway) %>%
  step_dummy(stop) %>%
  step_dummy(traffic_signal) %>%
  step_dummy(sunrise_sunset) %>%
  step_normalize(all_predictors()) %>%
  step_upsample(severity,over_ratio = 0.50)

save(accidents_folds,accidents_recipe,accidents_train,file = "R_Scripts/model_fitting/mode
l_setup.rda")
```

- Firstly, I create only 5 folds since we have such an abundance of observations, but to compensate for the lower value of folds, I will increase repeats to 4. And again, due to our class unbalance, I must make sure to stratify our data by severity.
- Next, I will dummy encode all of our factor variables
- Normalize the predictor variables
- Finally, as I noted before, a valuable way to deal with the class unbalance is to up sample. This means I will randomly sample with replacement our under-represented class until it matches about 50% (in my case) of the more abundance class. The reason for doing this is because if we don't up sample, then our model will continually predict severity 2 because that's the only data that the model is trained to predict since there's almost no appearances of severity 4 accidents, meaning that the models predictive power will be extremely low.

Preparing the Models

The models I'll be using are:

- logistic regression
- single decision tree
- boosted tree
- random forests
- k-nearest neighbors

Below, all my models were tuned to 4 levels as too many more levels would greatly increase run-time and computing power as 4 levels already took over ~4 hours to run due to my high number of observations, so 4 levels should be enough especially since I have 4 repeats.

logistic regression

Hide

```
log_reg <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

log_wkflow <- workflow() %>%
  add_model(log_reg) %>%
  add_recipe(accidents_recipe)


save(log_wkflow,file = "R_Scripts/model_fitting/log_model.rda")
```

First, it’d be nice to run logistic model using the glm engine to find any potentially signicant variables by looking at the p-values later on.

single decision tree

Hide

```
tree_spec <- decision_tree() %>%
  set_engine("rpart")

class_tree_spec <- tree_spec %>%
  set_mode("classification")

class_tree_wf <- workflow() %>%
  add_model(class_tree_spec %>%
  set_args(cost_complexity = tune())) %>%
  add_recipe(accidents_recipe)

class_tree_param_grid <- grid_regular(cost_complexity(range = c(0,1),trans = identity_tran
s()), levels = 4)

class_tree_tune_res <- tune_grid(
  class_tree_wf,
  resamples = accidents_folds,
  grid = class_tree_param_grid)

save(class_tree_wf,class_tree_tune_res,file = "R_Scripts/model_fitting/class_tree_model.rd
a")
```

Using identity_trans, we can flip the scale for better interpretation, where 0 is low cost-complexity meaning we have low amounts of pruning happening to the nodes, and as we increase to 1, means we are increasing pruning of said number of nodes.

random forests

Hide

```
spec <- rand_forest() %>%
  set_engine("ranger", importance = "impurity")

rf_spec <- spec %>%
  set_mode("classification")

rf_spec_wf <- workflow() %>%
  add_model(rf_spec %>% set_args(mtry = tune(),trees = tune(), min_n = tune())) %>%
  add_recipe(accidents_recipe)

rf_param_grid <- grid_regular(mtry(range = c(1,16)),trees(range = c(100,500)),min_n(range
 = c(1,16)),levels = c(4,4,4))

rf_tune_res <- tune_grid(
  rf_spec_wf,
  resamples = accidents_folds,
  grid = rf_param_grid)

save(rf_spec_wf,rf_tune_res,file = "R_Scripts/model_fitting/rf_model.rda")
```

As for the random forests, I tuned mtry to contain all 16 predictor variables, and made trees range from 100,500. The reason being is that trees after a certain range shouldn’t contain too many differences in roc_auc therefore, due to computing power, we shouldn’t need more than 500 trees to find a notable difference. Nextly, I have many observations but I only set my minimum node amount to 16 observations per node, this is because I don’t want too small of a tree and would much prefer a flexible tree if possible. Additionally, if the range is too large we will miss out on accuracy, therefore 1-16 is a good range for high precision. Notably though, I will be playing around with min_n ranges later to see if lowering the range might actually yield a better roc_auc.

boosted tree

Hide

```
spec <- boost_tree() %>%
  set_engine("xgboost")

boost_spec <- spec %>%
  set_mode("classification")

boost_spec_wf <- workflow() %>%
  add_model(boost_spec %>% set_args(mtry = tune(),trees = tune(),min_n = tune())) %>%
  add_recipe(accidents_recipe)

boost_param_grid <- grid_regular(mtry(range = c(1,16)),trees(range = c(100,500)),min_n(range = c(1,16)),levels = c(4,4,4))

boost_tune_res <- tune_grid(
  boost_spec_wf,
  resamples = accidents_folds,
  grid = boost_param_grid)

save(boost_spec_wf,boost_tune_res,file = "R_Scripts/model_fitting/boost_model.rda")
```

Again, I tuned both my boosted trees and random forest to similar hyperparameter ranges because I want to compare the two models performance in a relatively similar fashion.

K-nearest Neighbors

Hide

```
spec <- nearest_neighbor() %>%
  set_engine("kknn")

knn_spec <- spec %>%
  set_mode("classification")

knn_wkflow <- workflow() %>%
  add_model(knn_spec %>% set_args(neighbors = tune())) %>%
  add_recipe(accidents_recipe)

knn_param_grid <- grid_regular(neighbors(range = c(1,1000)),levels = 4)

knn_tune_res <- tune_grid(
  knn_wkflow,
  resamples = accidents_folds,
  grid = knn_param_grid)

save(knn_tune_res, knn_wkflow, file = "R_Scripts/model_fitting/knn_model.rda")
```

The only parameters I tuned were neighbors, and it’s generally safe that neighbors is tuned to the square root of the number of observations (in my case, 30k). Therefore, a range from 1-1000 is pretty safe. I set the engine to kknn to denote k-nearest neighbors and set the mode to classification. And I saved my workflow

Running the Models

Loading Data

Hide

```
load("R_Scripts/model_fitting/model_setup.rda")
load("R_Scripts/model_fitting/log_model.rda")
load("R_Scripts/model_fitting/class_tree_model.rda")
load("R_Scripts/model_fitting/rf_model.rda")
load("R_Scripts/model_fitting/boost_model.rda")
load("R_Scripts/model_fitting/knn_model.rda")
```

Logistic

Code

```
## # A tibble: 29 × 5
##    term             estimate std.error statistic  p.value
##    <chr>               <dbl>     <dbl>     <dbl>    <dbl>
##  1 (Intercept)       -1.86      2.20      -0.848 3.97e- 1
##  2 distance_mi       -0.200     0.0195   -10.2   1.70e-24
##  3 temperature_f      0.905     0.216      4.18  2.86e- 5
##  4 wind_chill_f      -1.08      0.213     -5.08  3.78e- 7
##  5 humidity_percent  -0.0784    0.0263    -2.98  2.88e- 3
##  6 pressure_in        0.0255    0.0164     1.56  1.19e- 1
##  7 visibility_mi     -0.144     0.0274    -5.27  1.40e- 7
##  8 wind_speed_mph    -0.117     0.0177    -6.62  3.60e-11
##  9 precipitation_in  -0.284     0.0522    -5.44  5.40e- 8
## 10 side_R             0.0527    0.0150     3.50  4.58e- 4
## # … with 19 more rows
```

From intial view, we see that distance_mi, temp_f, wind_chill, humidity, visibility, wind_speed, precipiation, side, haze, heavy and light snow, crossing, railway, stop, traffic_signal, and sunrise_sunset all appear to be significant predictors with a p-value < 0.05. Again, but our data is extremely skewed as we noted in our EDA, so this might leads to some problems of predictions and it might draw conclusions due to lack of data even despite upsampling. Therefore, let's find the accuracy of our model and create a confusion matrix below to better/simply visual our data.

Code

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.982
```

We see that the our logistic model is extremely accurate, but how true is this? Let's create the confusion matrix.

Code

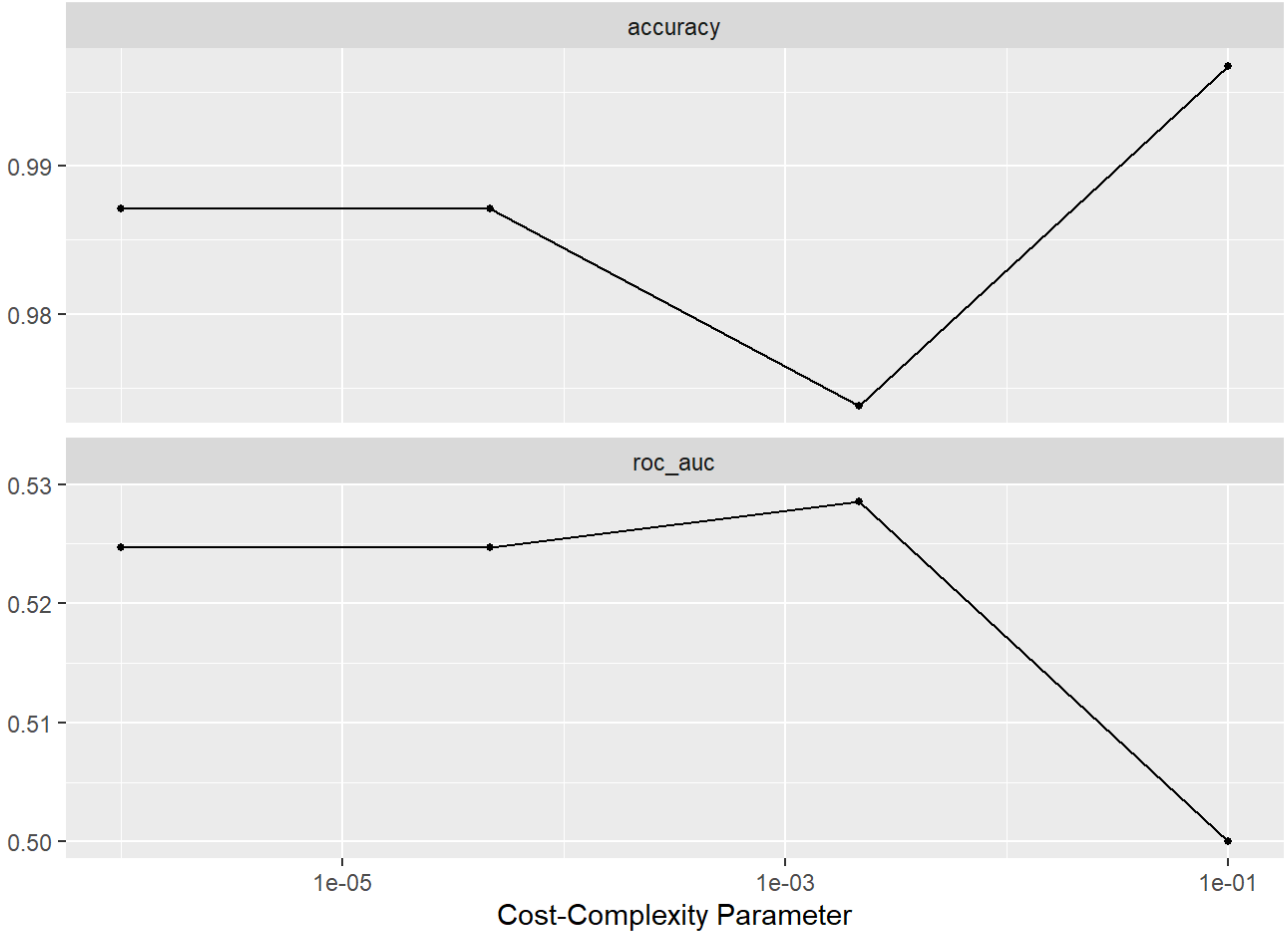
```
##           Truth
## Prediction     2     4
##          2 24047    74
##          4   358     5
```

Ahhh, now we know why our logistic model is accurate. Since logistic accuracy the model’s ability to predict the right thing it’ll obviously be extremely accurate since there are so many severity 2’s to predict. But from our confusion matrix we actually see that the model is extremely bad at predicting severity 4’s and more often than not, it’ll wrongly predict or never predict severity 4’s. This means logistic might be a poor model to run despite it’s high accuracy, and although it might be a linear model, our data is over-represented and the simplicity of logistic models might not be able to capture the differences due to lack of data. So we should probably perform different models below.

Single Decision Tree

Plot of accuracy and roc_auc

Code



The first thing we should note is that our roc_auc is extremely poor no matter what our cost-complexity is. Even though our accuracy is increasing, it might be for the same reason that we are continually predicting severity 2’s and not 4’s so therefore our accuracy increase although that’s not too helpful. Therefore, we can’t accurately predict severity 4’s and ultimately, our roc_auc decreases as a result. A hover roc_auc means that our model is as good as coin flipping if it’s severity 2 or 4, so therefore, using the model wouldn’t be quite useful.

Best Performing Model

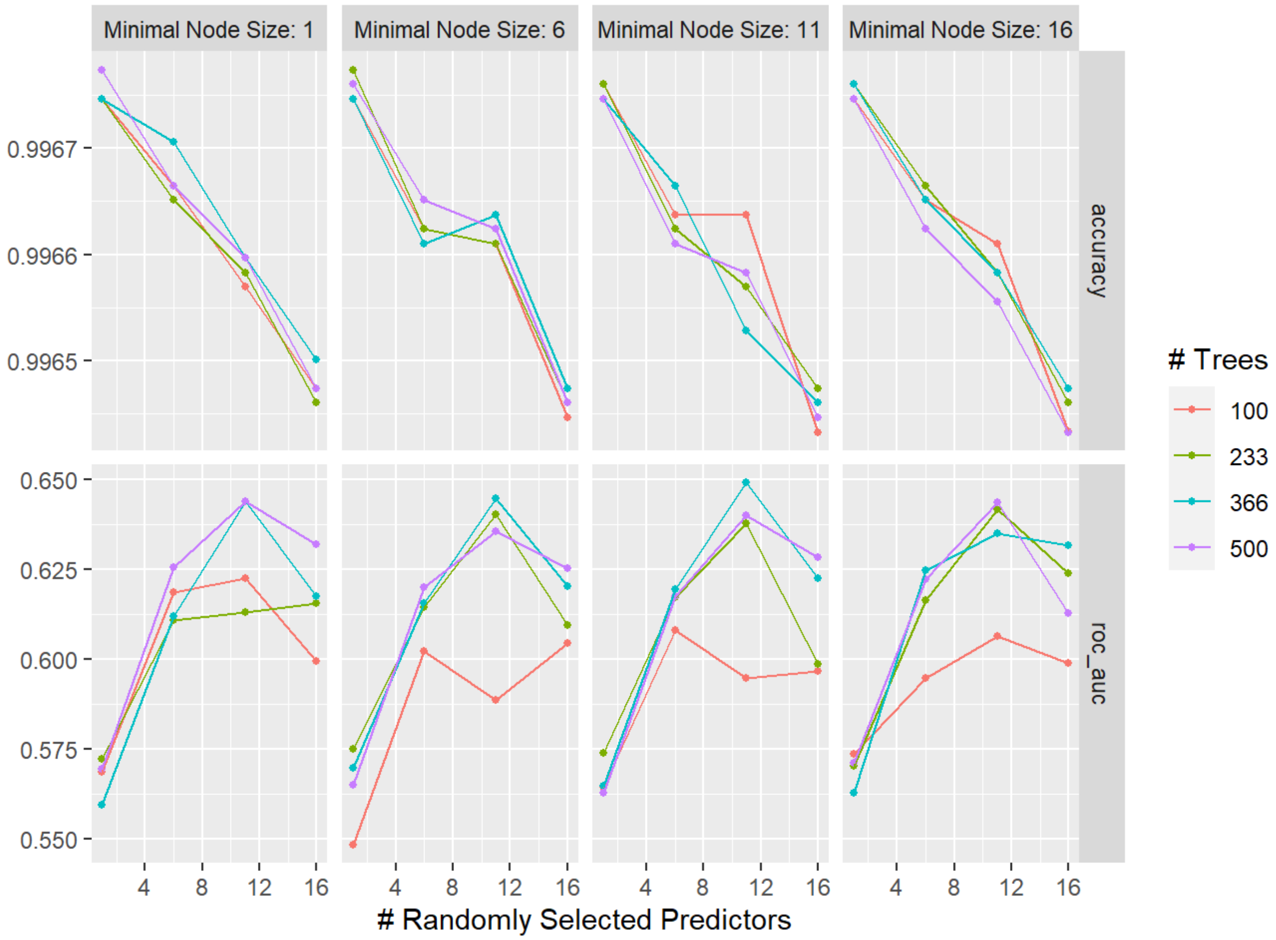
Code

```
## # A tibble: 4 × 7
##   cost_complexity .metric .estimator  mean     n std_err .config
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1       0.00215   roc_auc binary     0.529    30 0.0108  Preprocessor1_Model3
## 2       0.000001  roc_auc binary     0.525    30 0.00974 Preprocessor1_Model1
## 3       0.0000464 roc_auc binary     0.525    30 0.00974 Preprocessor1_Model2
## 4       0.1       roc_auc binary     0.5      30 0       Preprocessor1_Model4
```

Random Forest

Maybe performing more in-depth models such as random forest and boosted trees might end up being more valuable and predictive, so lets see the results of both below.

Code



We can see that depending on our hyper-parameters, accuracy is not quite an issue because although it seems to dip heavily, there really isn’t much difference in accuracy as we get a >99% accuracy every single time. Now, lets analyze the roc_auc. Already we see that our roc_auc for random forest models is doing MUCH better than our logistic and single decision tree, which might be expected due to their limitations and flexibility. Our random forest generally yields an roc_auc of around ~0.63-0.65, meaning that our models is not too bad and this might be a noteworthy model when testing our test data.

For the most part, we see that there seems to be a declining success after around 11 predictors, meaning that there seems to be 5 predictors that are pretty invaluable for our model. Additionally, having our tree’s above 233 generally don’t increase our roc_auc by all too much so any amount of trees after this point don’t seem to produce any groundbreaking changes.

But what I find interesting is that I only limited the minimum node size from 1-16, which relative to having a data set with 30k observations is quite low. That means that we create trees with 1-16 minimum observations before moving onto the next node. A larger minimum node amount means that a smaller tree is built, and we see that 1 node doesn’t seem to differ much when compared with larger minimum node values so our trees are overall pretty flexible. But we will explore this further in a later section.

Best Performing model

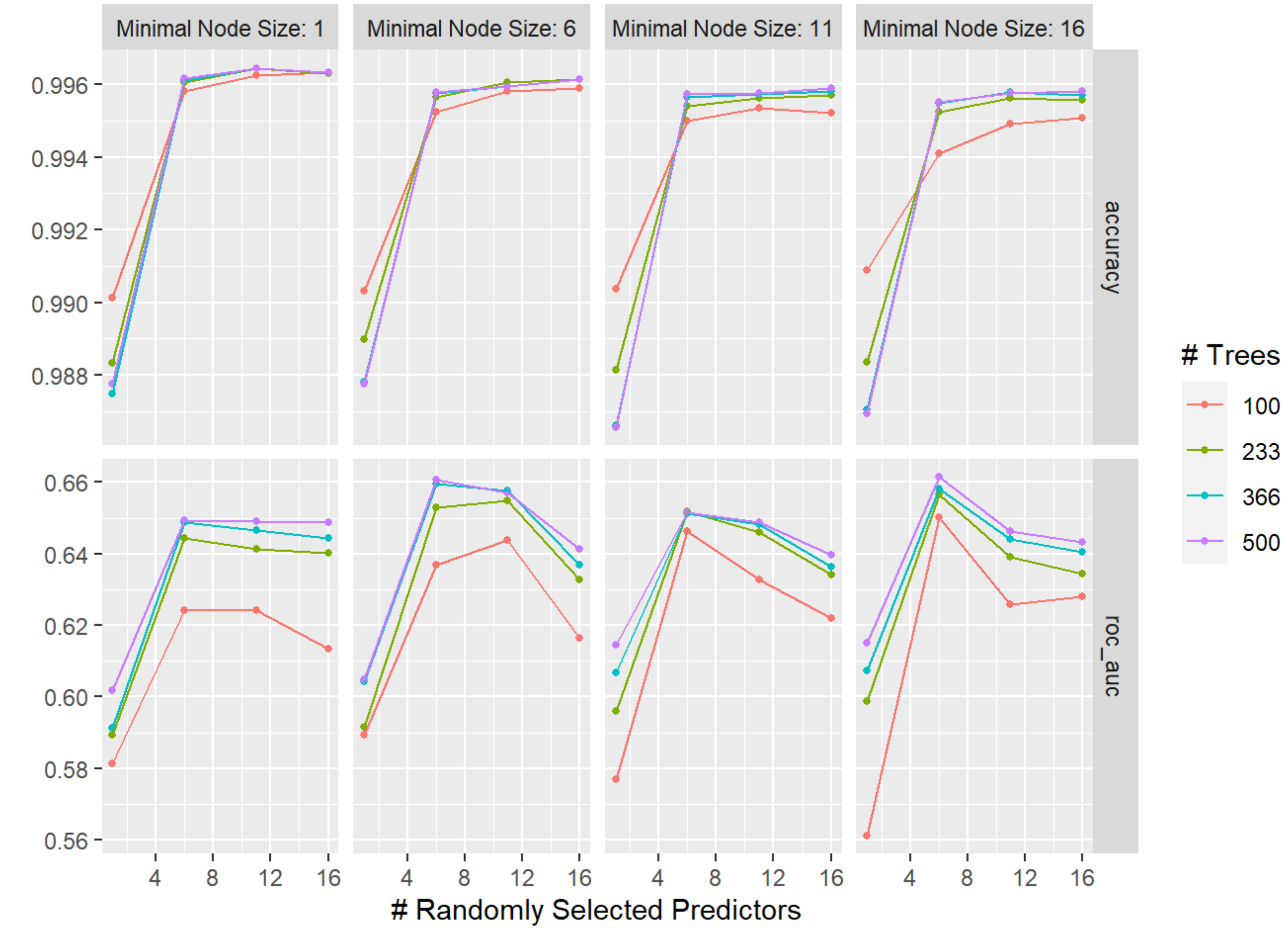
Code

```
## # A tibble: 5 × 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    11   366    11 roc_auc binary     0.649    30  0.0219 Preprocessor1_Model43
## 2    11   366     6 roc_auc binary     0.645    30  0.0173 Preprocessor1_Model27
## 3    11   366     1 roc_auc binary     0.644    30  0.0182 Preprocessor1_Model11
## 4    11   500     1 roc_auc binary     0.644    30  0.0196 Preprocessor1_Model15
## 5    11   500    16 roc_auc binary     0.643    30  0.0223 Preprocessor1_Model63
```

As I noted, since there doesn’t seem to be a big difference in increase node size, and a flexible model is possible, then perhaps we can try lowering our node range in a different mode using our best hyper-parameters in terms of trees and mtry. We see that a minimum node of 1 also produces pretty good roc_auc, therefore we can create a range that’s around 1 and maybe see if a min_n around there might find better roc_auc/accuracy.

Boosted Tree

Code



Compared to random forest, we already see that generally our model is preforming better in terms of roc_auc, even if by a little. Also compared to random forest, our sweat spots for predictors are around 6 predictors while the amount of trees seem to produce a pretty negligible difference in roc_auc. And what's even more interesting, while random forest accuracy seemed to continually decrease when adding more predictors, it seems that after 6 predictors variables, our accuracy actually increase while roc_auc decreases slightly. Again, this can maybe be attributed to over-predictions of severity 2's, but we can draw a confusion matrix later to see if that's the case.

Best Performing model

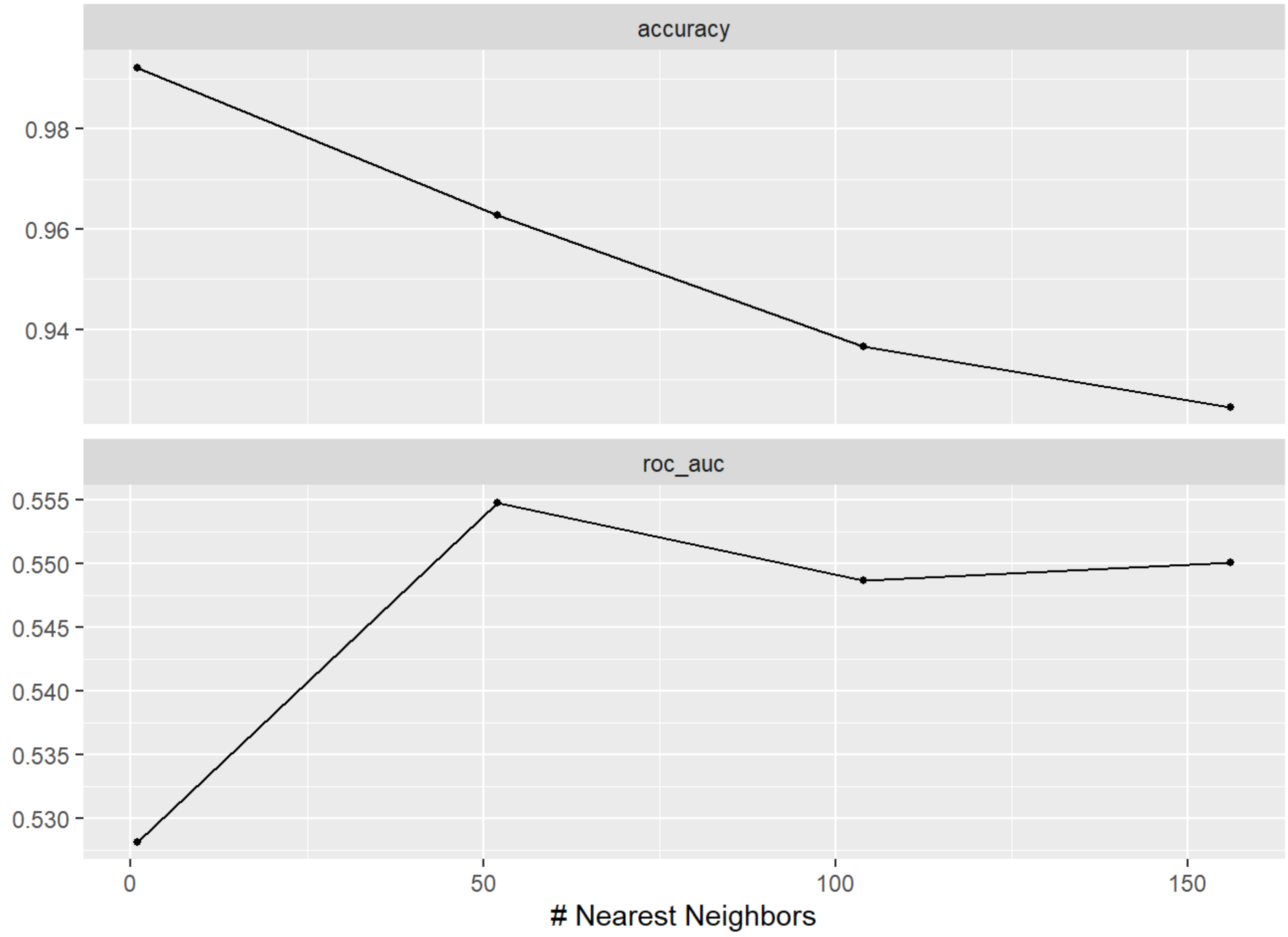
Code

```
## # A tibble: 5 × 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     6   500    16 roc_auc binary     0.661    30  0.0201 Preprocessor1_Model32
## 2     6   500     6 roc_auc binary     0.661    30  0.0165 Preprocessor1_Model24
## 3     6   366     6 roc_auc binary     0.660    30  0.0162 Preprocessor1_Model23
## 4     6   366    16 roc_auc binary     0.658    30  0.0200 Preprocessor1_Model31
## 5    11   366     6 roc_auc binary     0.658    30  0.0201 Preprocessor1_Model39
```

Since boosted trees are similar to random forest, we will do the model fitting by seeing if a change in the range of min_n around the best min_n will produce a better roc_auc. Especially because we see a min_n of 6 and 16 generally produce the best roc_auc.

K-nearest Neighbors

Code



One thing to note is that k-nearest neighbors actually perform poorly when it comes to large datasets, high number of dimensions, and many outliers, and with a class inbalanced,large data set, and from our eda, we found distance_mi had many outliers, and finally with over 17 predictors, we can already assume this model won't be too significant.

And follow our assumptions, we see that our roc_auc is quite low, with almost being similar to the coinflip probabilities of our decision tree model. It also has a pretty noteable continuous decrease in accuracy as the amount of neighbors increase.

Since this is a model that isn't particular impressive, there's no need to show the best performing tuned parameters since we won't be using it to fit our testing set.

Finding New Models

Above we ran some models with tuned parameters in the forest and boosted models. Notice we had a pretty large minimum node range, this can mean that we actually couldn't capture what's exactly happening at certain node levels since we can't see the minute difference that might happened at lower ranges.

Also, we see the models actually performed pretty well at lower minimum nodes too, meaing that our model is actually quite flexible as it can take in one observations minimum for each note and still perform well in terms of roc_auc. Therefore, maybe we can decrease the range of our min_n around those runner-up values in our best performing mtry and trees values to see how this might produce a more precise story within those ranges and if it at all will maybe show us if those rangers are better since we can see more details with a lower range. So we will change the grid and hypertune the parameters differently to see if any changes could be made.

Setting up the grid of hyperparameters

Hide

```
# Random Forest

spec <- rand_forest() %>%
  set_engine("ranger", importance = "impurity")

rf_spec <- spec %>%
  set_mode("classification")

min_n_rf_spec_wf <- workflow() %>%
  add_model(rf_spec %>% set_args(mtry = 11,trees = 366, min_n = tune())) %>%
  add_recipe(accidents_recipe)

min_n_rf_param_grid <- grid_regular(min_n(range = c(1,3)),levels = 4)

min_n_rf_tune_res <- tune_grid(
  min_n_rf_spec_wf,
  resamples = accidents_folds,
  grid = min_n_rf_param_grid)

save(rf_spec_wf,min_n_rf_tune_res,file = "R_Scripts/model_fitting/min_n_rf_model.rda")

# Boosted Tree

spec <- boost_tree() %>%
  set_engine("xgboost")

boost_spec <- spec %>%
  set_mode("classification")

min_n_boost_spec_wf <- workflow() %>%
  add_model(boost_spec %>% set_args(mtry = 6,trees = 500,min_n = tune())) %>%
  add_recipe(accidents_recipe)

min_n_boost_param_grid <- grid_regular(min_n(range = c(5,7)),levels = 4)

min_n_boost_tune_res <- tune_grid(
  min_n_boost_spec_wf,
  resamples = accidents_folds,
  grid = min_n_boost_param_grid)

save(boost_spec_wf,min_n_boost_tune_res,file = "R_Scripts/model_fitting/min_n_boost_model.
rda")
```

As I noted before, I want to fit models around a min_n that performed almost as good as our best model so we can maybe see a more detailed story of what might be happening at those closer ranges. For the random forest, this will be around 1-3, and for boosted trees, around 5-7. Let’s print the results below.

Printing out the Graphs

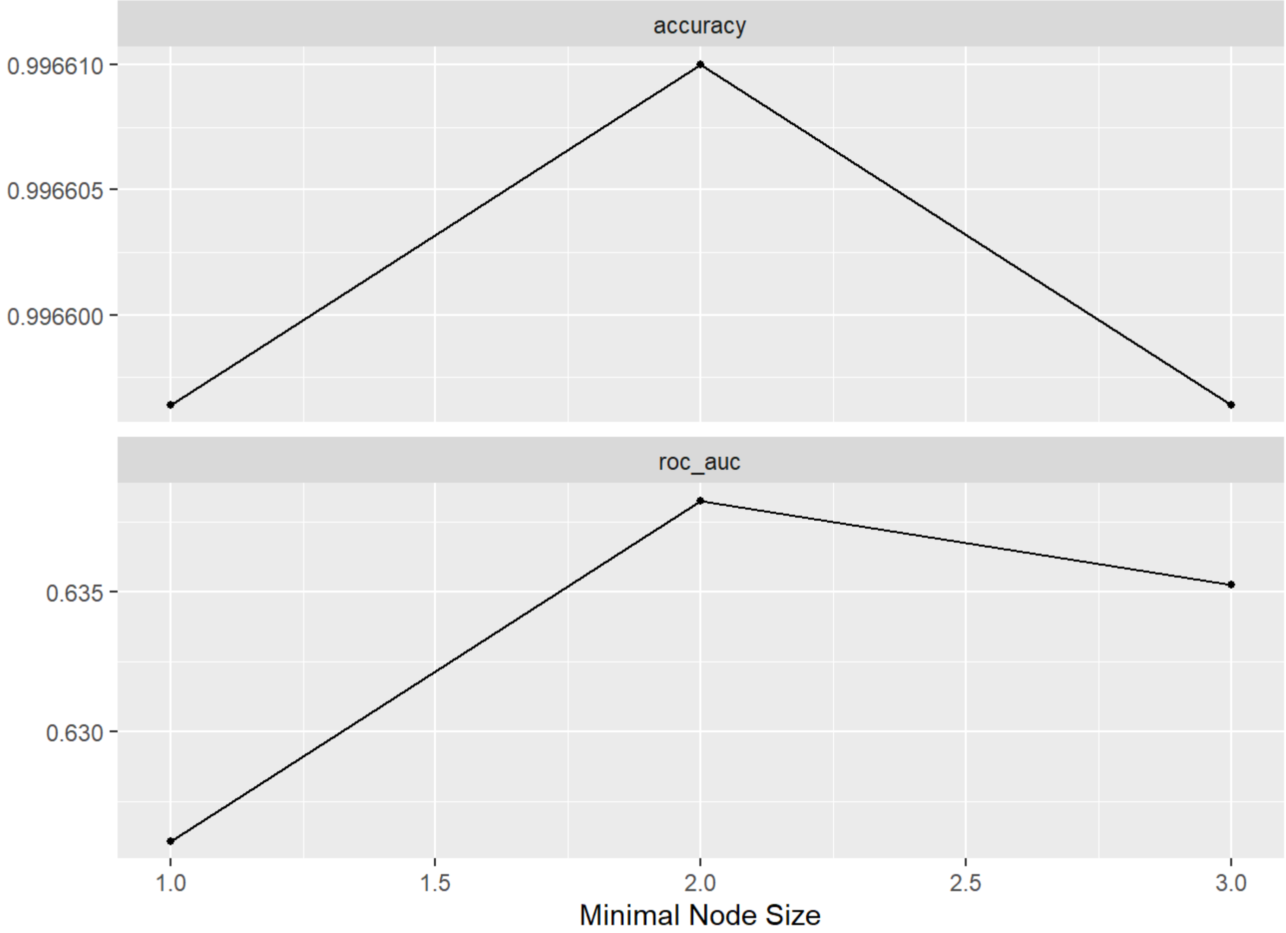
Loading data

Hide

```
# Loading the data
load("R_Scripts/model_fitting/min_n_rf_model.rda")
load("R_Scripts/model_fitting/min_n_boost_model.rda")
```

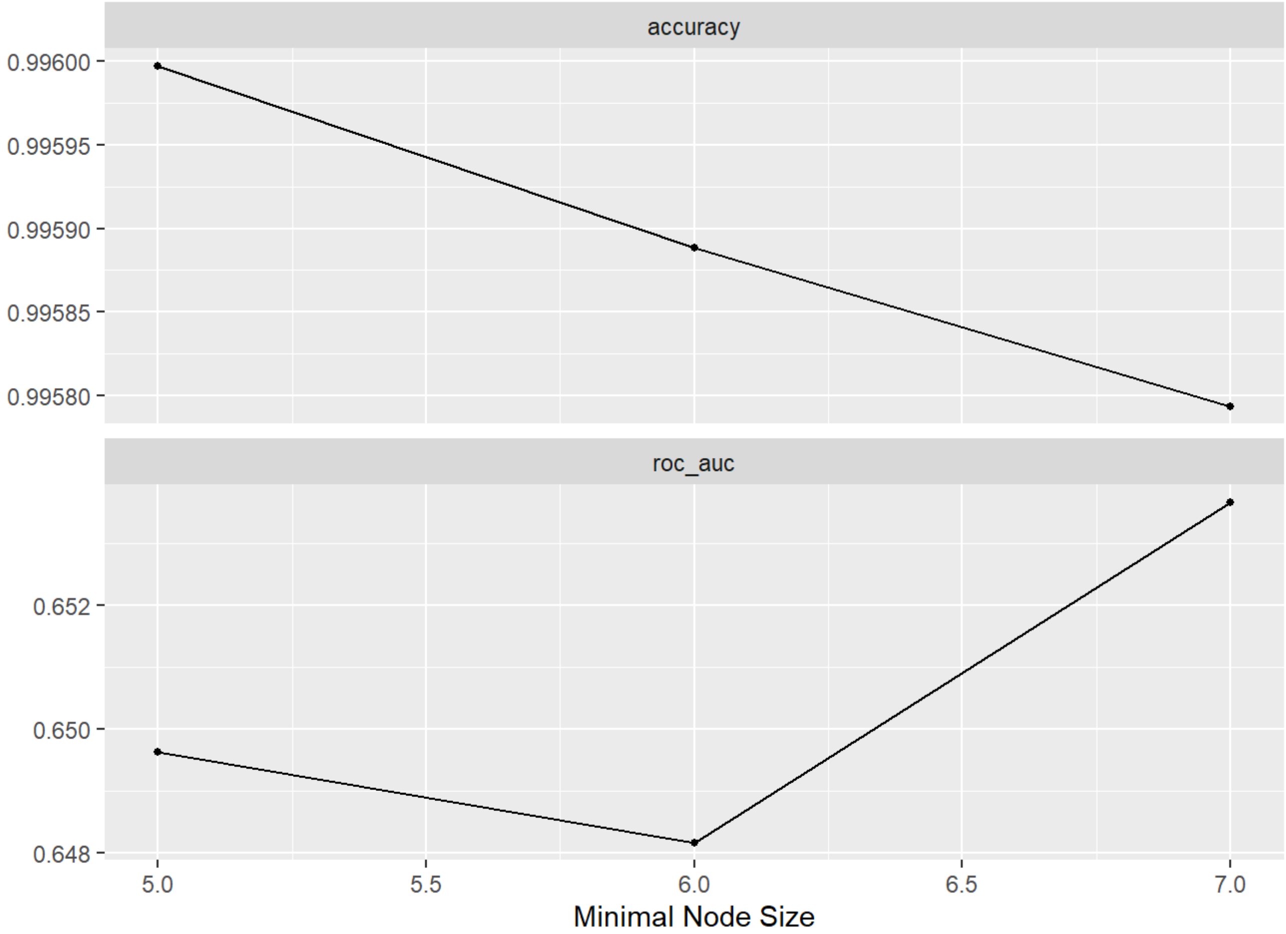
New min_n range Random Forest

Code



New min_n range Boosted Tree

Code



In conclusion, we see that although there might be some increase in roc_auc near ~7 and beyond for min_n for our boosted tree models, our accuracy continuously decreases and roc_auc is stil lower than our best performing model so it might be in our best interest to just maintain our best hyper parameters previously of mtry = 5, trees = 500, and min_n = 16 to maintain a equal balance of accuracy and roc_auc. As for the random forest model, although a min_n of two isn't too bad, notice that 11 actually yields our best results of accuracy combined with roc_auc. So for the most part, it might just be the safest option to also revert back to our original best hyper parameters as well since we have confirmed it might be the best working model within our original and tighter min_n range doesn't seem to produce much difference.

Fitting the Testing Sett

Ultimately, we determined that it's in our best interest to just use our previously best tuned hyperameters found initially, therefore we will just stick with those when fitting the testing set.

Setting up the best final workflows

Since we determined that the boosted trees overall yields the best roc_auc while maximizing accuracy, we will fit our testing data with this model.

Hide

```
boost_final <- finalize_workflow(boost_spec_wf,select_best(boost_tune_res,metric = "roc_au
c"))

boost_final_result <- fit(boost_final, data = accidents_train)

save(boost_final_result,file = "R_Scripts/model_fitting/best_model.rda")
```

I want to emphasize that I'm mainly focusing on roc_auc because accuracy can be an extremely skewed value here especially since we have such a large class inbalance and naturally the models will produce a high accuracy due to the fact.

Displaying Graphics

loading data

Hide

```
load("R_Scripts/model_fitting/best_model.rda")
```

roc_auc value of best boosted tree Model on test data

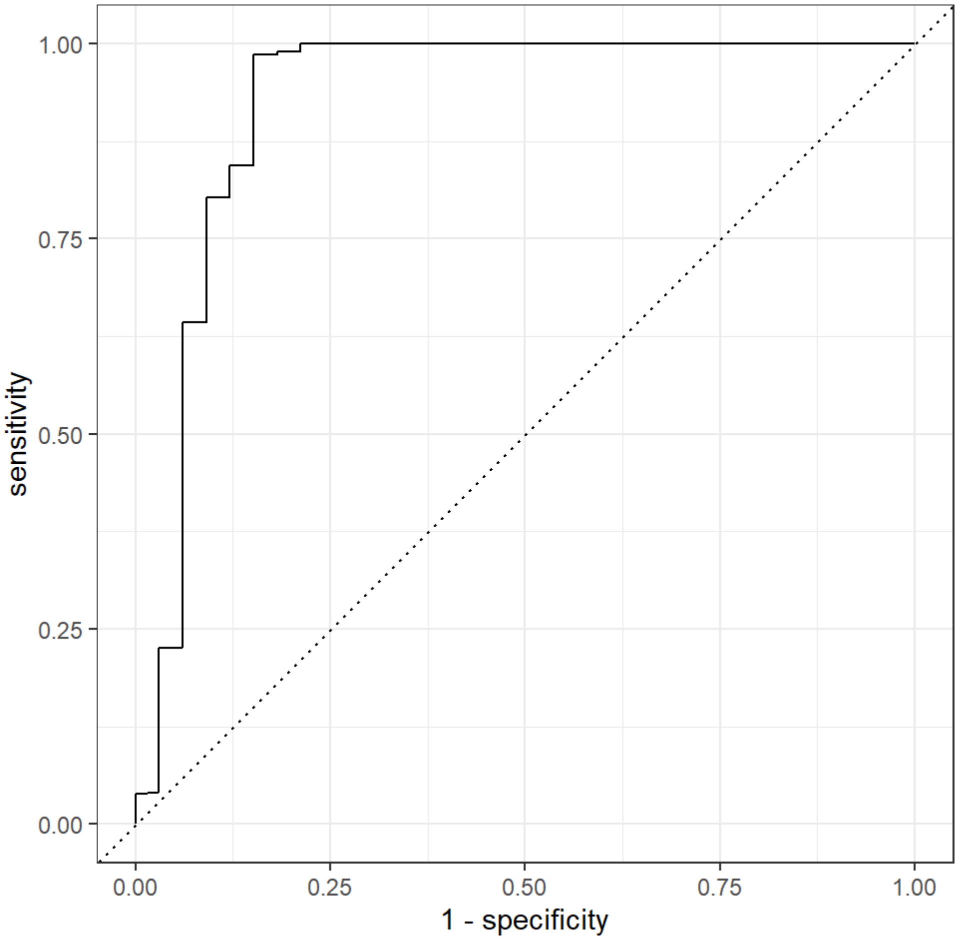
Code

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.925
```

Notice that overall, when it comes to predicting our severity 2 accidents, we actually have an extremely high roc_auc. This is predicted since we have an abundance of severity 2 values, so lets see the roc_auc for severity 4 accidents. Wow! Our roc_auc is very high and initially it may seem like our model is excellent at predicting severity 4. But let's not take the number for granted as it could've just been overfitted to predict severity 2, so it'll be in our best interest to see how well our model did based on looking at a heat matrix.

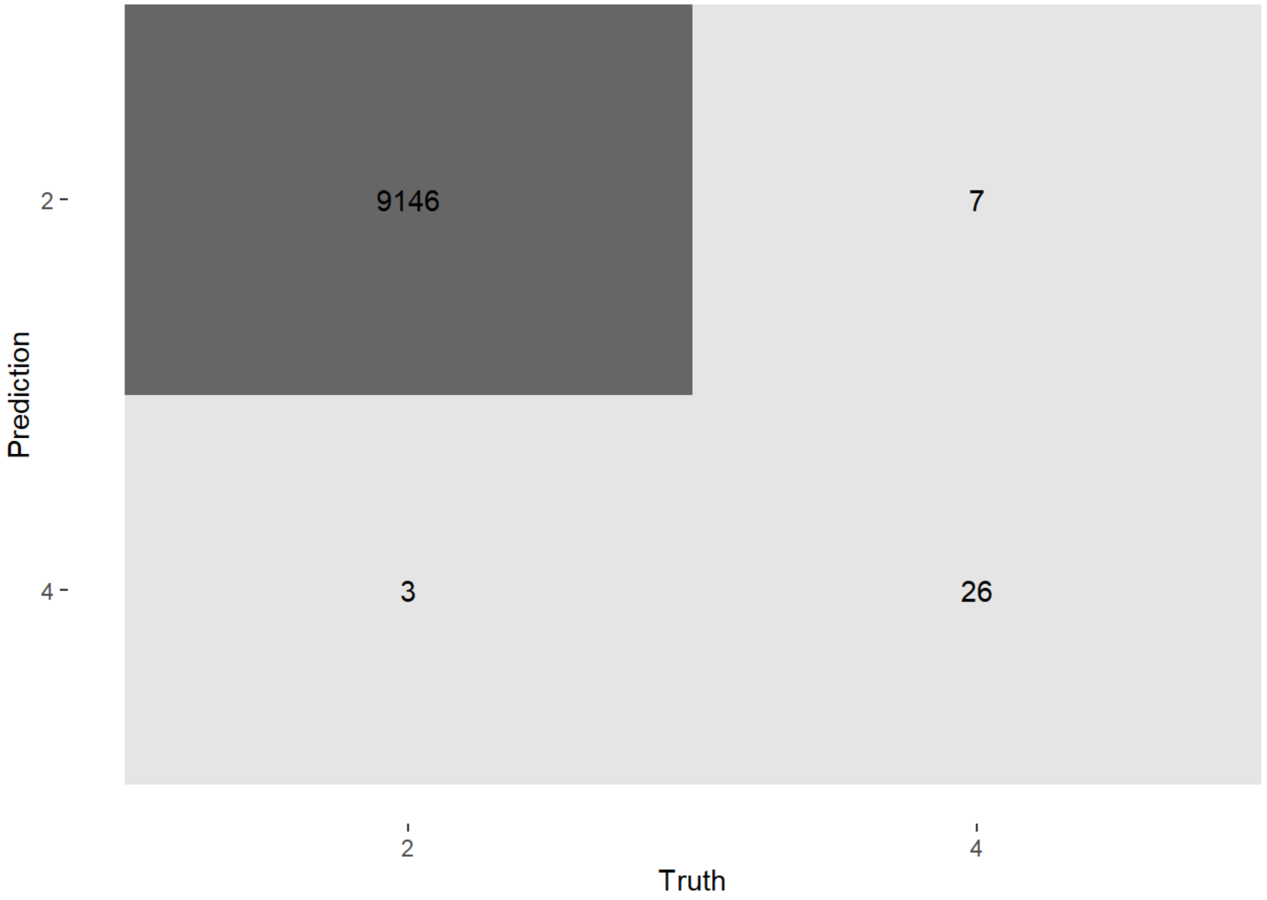
roc_auc curve of best boosted tree Model on test data

Code



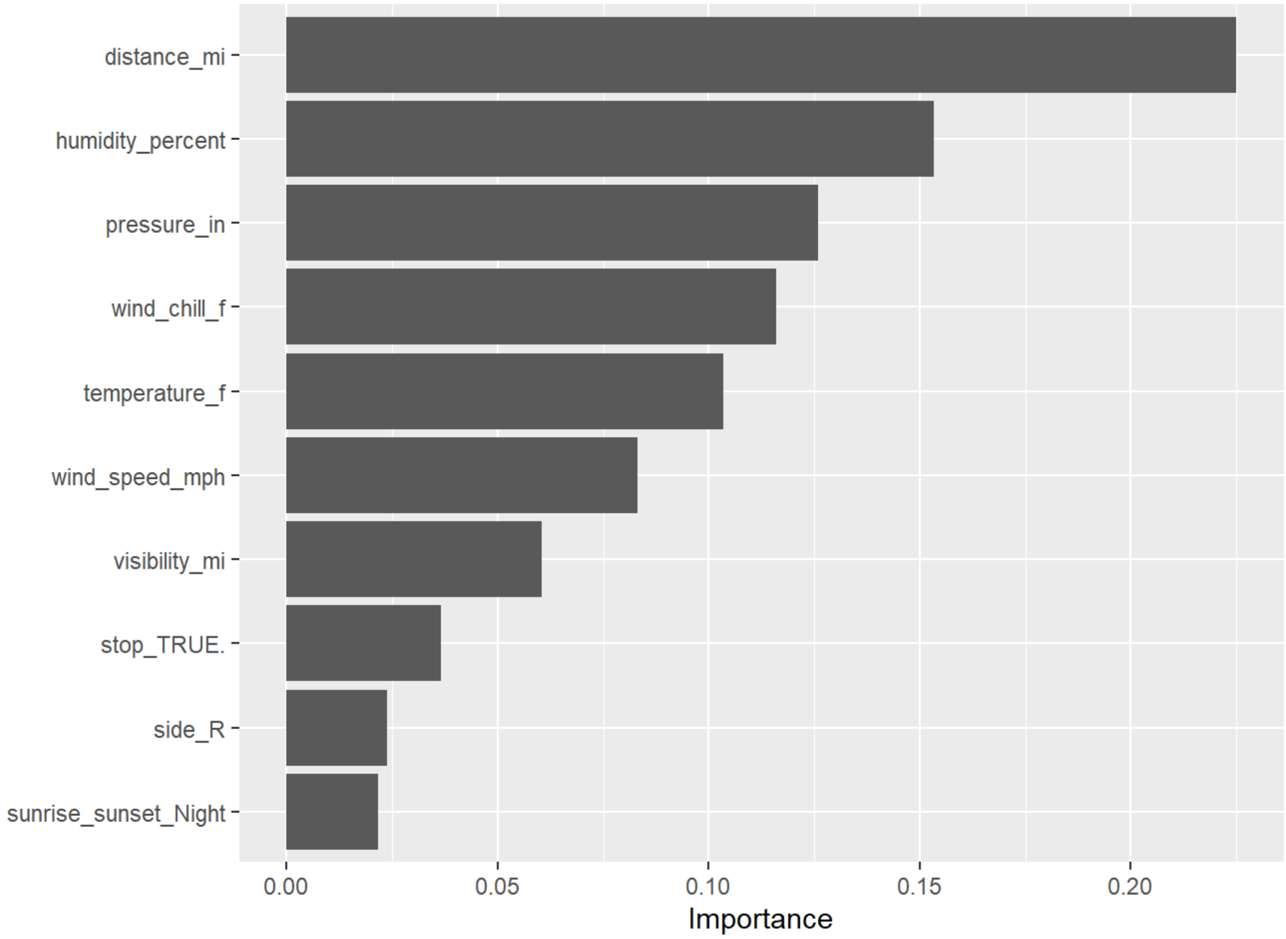
The graphic above is just the roc_auc curve for visualization of how well our model fitted the the testing data.

Code



Looking at our heat matrix, we actually see our model did quite well in predicting severity 4 despite the major disparity in severity 4 accidents. It only predicted severity 4 wrong 3 times, while accurately predicting it right 26 times. And as expected, it did a great job in predicting severity 2, which may be due to the obvious fact that there’s an abundance of severity 2 observations as I’ve continually noted.

Code



It also might be in our best interest to see what the most valuable predictors were in our boosted tree model. Our variable importance chart tells us that distance_mi was by far the best predictor of severity which was expected since we'd expect a more severe accident to block more of traffic of course. After we see that humidity_percent, pressure_in, wind_chill_f, temperature_f were all pretty significant predictors which is actually extremely surprising. All of them being important also may be due to the fact that they were all already initially significantly correlated so potentially running interactions in the recipes would be worth modeling in the future for further accuracy perhaps. Then we see that stop signs, sidedness, and sunrise_sunset had decently/low importance. Stop signs might be significant because those are typically where intersections exist so more traffic would be held up as a result of this, and potentially more severe accidents would happen in the day time since more traffic exists during those hours, but what was surprising is road sideness which I'd think wouldn't be too important.

Conclusion and Final Remarks

The entire analysis covers the possible correlation between various factors that could potentially yield varying levels of severity when it comes to traffic accidents. But, as the very famous saying goes, correlation doesn't equal causation. So it's very important that we take this entire analysis with a grain of salt, and ultimately, as the end of the day whether it rains, snows, or it's night time, we'll always have to drive to places when it's required. Although driving is unavoidable, and therefore accidents as well, there are specially precautions that this data set can make aware and raise awareness for drivers across California that we are all too familiar with. Many of us has experienced the results of terrible traffic accidents, many of us have complained about having to sit for an hour staring at the cars in front of us, and, unfortunately, some have also been at the center of these accidents.

California is the leading number of fatal car accidents in the US each year, so car accidents are nothing to take lightly. Doing analysis as such, and continually providing future analysis can be crucial in lowering these accidents. This could involve finding what are leading causes of accidents and then tackling these issues according, such as providing potentially safer means of travel when the weather condition is poor, implementing stricter restrictions when it comes to certain things such as running red lights or stop signs, or even building better functions upon cars that could prevent weather related accidents such as wheels that can handle all rain and snow. The list goes on and the importance increases as more people are taken to the road. Again, it's important to note that my data set was extremely class unbalanced, and only limited to one year, so many things could change had I expanded my horizons, but nonetheless the model produced quite an accurate prediction and provides even further evidence to continue this type of accident exploration in the future.

In regards to the models, if I had infinite computing power there would be many things that could be altered for a higher roc_auc. One such way may be playing around with the hyperparameters ranges even further, or the up sampling proportions, by either increasing or decreasing the value. My model doesn't seem to over predict severity 2 accidents so therefore there shouldn't be a reason to lower the proportions, so maybe increasing the proportion might produce higher roc_auc. Additionally, another option would be the down sample my abundant class, severity 2, and then adding weights to the lower class. Because I had already limited my extremely large, original data set to only 2021 and California, I no longer wanted to cut my data by anymore so I wouldn't leave out data that could've been crucial in analysis therefore I decided to choose up sampling instead. Nonetheless, down sampling would indeed be an option to explore. With that said, 2021 was also the year that COVID was still a major concern and ongoing pandemic, therefore there might be issues with documentation and maybe even less people on the road therefore far less severe accidents that what might normally occur in previous and future years, which is definitely something that should be considered when looking at my results.

To make some final remarks, I believe that boosted trees and random forests will still function the best in terms of future analysis because k-nearest neighbors has obvious limitations, as such with logistic and singular decisions trees. Although my roc_auc wasn't impressive by any means, with the two best models producing only ~66% roc_auc, it still fit my testing data quite well meaning my model was by no means over fitted. But, as shown in logistic regression pertaining to the model, we must take the accuracy with a grain of salt because since we have so many severity 2 observations, all our models are clearly going to mainly predict severity 2 therefore leading to a high accuracy since getting all or some of the severity 4 prediction wrongs will not hurt accuracy by all too much. Overall, I am pretty happy with the outcome of the models, and would like to make some changes in the future in terms of data set range and even explore other states and years as well.