Homework 5

PSTAT 131/231

Elastic Net Tuning

For this assignment, we will be working with the file `"pokemon.csv"`, found in `/data`. The file is from Kaggle: https://www.kaggle.com/abcsds/pokemon.

The Pokémon franchise encompasses video games, TV shows, movies, books, and a card game. This data set was drawn from the video game series and contains statistics about 721 Pokémon, or "pocket monsters." In Pokémon games, the user plays as a trainer who collects, trades, and battles Pokémon to (a) collect all the Pokémon and (b) become the champion Pokémon trainer.

Each Pokémon has a primary type (some even have secondary types). Based on their type, a Pokémon is strong against some types, and vulnerable to others. (Think rock, paper, scissors.) A Fire-type Pokémon, for example, is vulnerable to Water-type Pokémon, but strong against Grass-type.

Figure 1: Fig 1. Vulpix, a Fire-type fox Pokémon from Generation 1.

The goal of this assignment is to build a statistical learning model that can predict the **primary type** of a Pokémon based on its generation, legendary status, and six battle statistics.

Read in the file and familiarize yourself with the variables using `pokemon_codebook.txt`.

```
library(tidymodels)
library(ISLR) # For the Smarket data set
library(ISLR2) # For the Bikeshare data set
library(discrim)
library(poissonreg)
library(corrr)
library(glmnet)
library(klaR) # for naive bayes
tidymodels_prefer()
setwd("~/Desktop/PSTAT 131/homework-5")
set.seed(3435)
```

Exercise 1

Install and load the `janitor` package. Use its `clean_names()` function on the Pokémon data, and save the results to work with for the rest of the assignment. What happened to the data? Why do you think `clean_names()` is useful?

```
library(janitor)
```

```
pokemon <- read.csv("data/Pokemon.csv")
pokemon <- clean_names(pokemon)
pokemon <- tibble(pokemon)
pokemon
```

```
## # A tibble: 800 x 13
##        x name       type_1 type_2 total    hp attack defense sp_atk sp_def speed
##    <int> <chr>      <chr>  <chr>  <int> <int>  <int>   <int>  <int>  <int> <int>
##  1     1 Bulbasaur  Grass  "Pois~   318    45     49      49     65     65    45
##  2     2 Ivysaur    Grass  "Pois~   405    60     62      63     80     80    60
##  3     3 Venusaur   Grass  "Pois~   525    80     82      83    100    100    80
##  4     3 VenusaurM~ Grass  "Pois~   625    80    100     123    122    120    80
##  5     4 Charmander Fire   ""       309    39     52      43     60     50    65
##  6     5 Charmeleon Fire   ""       405    58     64      58     80     65    80
##  7     6 Charizard  Fire   "Flyi~   534    78     84      78    109     85   100
##  8     6 Charizard~ Fire   "Drag~   634    78    130     111    130     85   100
##  9     6 Charizard~ Fire   "Flyi~   634    78    104      78    159    115   100
## 10     7 Squirtle   Water  ""       314    44     48      65     50     64    43
## # ... with 790 more rows, and 2 more variables: generation <int>,
## #   legendary <chr>
```

the clean_names function has resulting names are unique and consist only of the _ character, numbers, and letters. and accented characters are transliterated to ASCII.

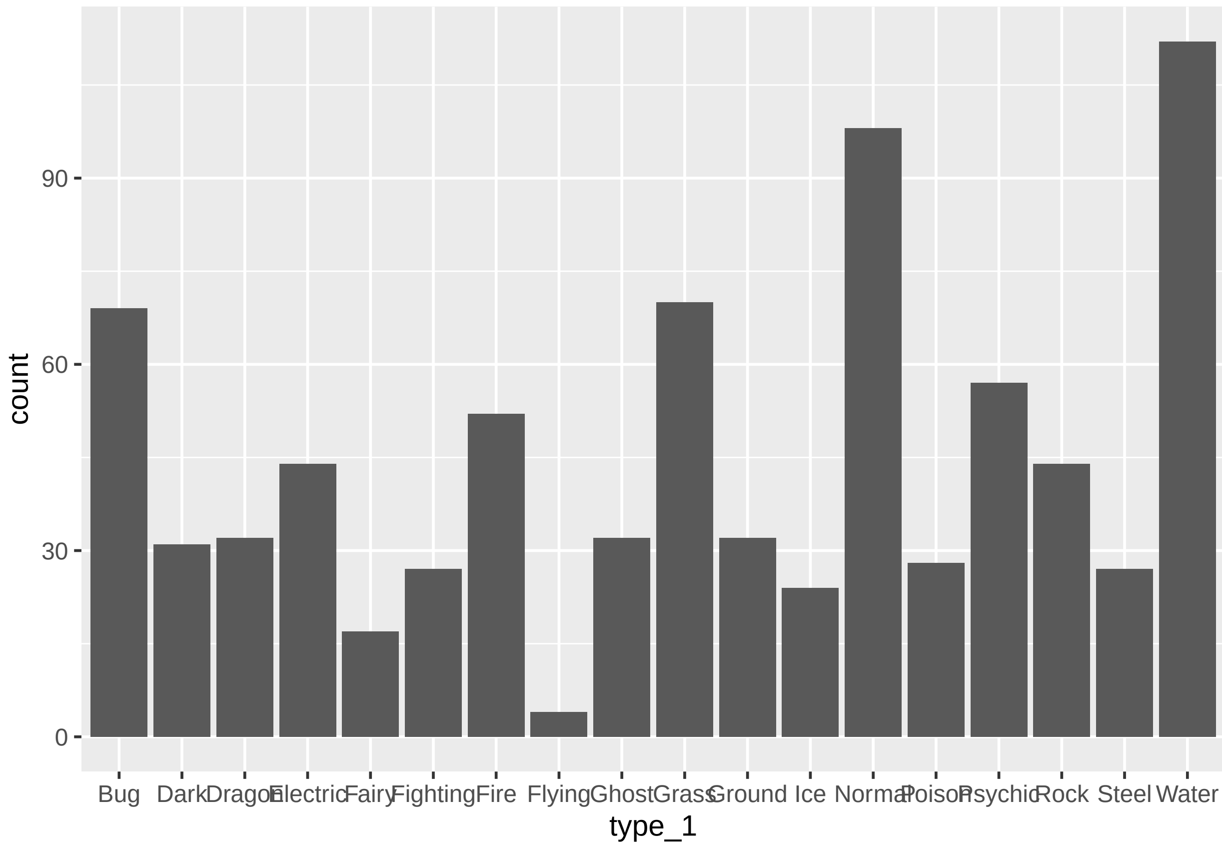
Exercise 2

Using the entire data set, create a bar chart of the outcome variable, `type_1`.

How many classes of the outcome are there? Are there any Pokémon types with very few Pokémon? If so, which ones?

```
library(ggplot2)
```

```
g <- ggplot(pokemon, aes(type_1))
g + geom_bar()
```



There are a total of 18 types and from the bar chart we see that there are few pokemons for flying and fariy.

For this assignment, we'll handle the rarer classes by simply filtering them out. Filter the entire data set to contain only Pokémon whose `type_1` is Bug, Fire, Grass, Normal, Water, or Psychic.

After filtering, convert `type_1` and `legendary` to factors.

```
pokemon <- pokemon[pokemon$type_1 %in% c("Bug","Fire", "Grass", "Normal", "Water", "Psychic"), ]

pokemon$type_1 <- as.factor(pokemon$type_1)
pokemon$legendary <- as.factor(pokemon$legendary)
```

Exercise 3

Perform an initial split of the data. Stratify by the outcome variable. You can choose a proportion to use. Verify that your training and test sets have the desired number of observations.

Next, use *v*-fold cross-validation on the training set. Use 5 folds. Stratify the folds by `type_1` as well. *Hint: Look for a `strata` argument.* Why might stratifying the folds be useful?

```
pokemon_split <- initial_split(pokemon, prop = 0.7,strata = type_1)
pokemon_train <- training(pokemon_split)
pokemon_test <- testing(pokemon_split)
```

```
dim(pokemon_train)
```

```
## [1] 318  13
```

```
dim(pokemon_test)
```

```
## [1] 140  13
```

So the dimension makes sense.

Stratifying the sample by type 1 is good because the number of samples in each cateries is different, and we want to stratify sampling by type_1 so that each fold can best represent the entire training set. this makes sure that every subgroup is represented.

```
pokemon_folds <- vfold_cv(pokemon_train, v = 5,strata = type_1)
```

Exercise 4

Set up a recipe to predict `type_1` with `legendary`, `generation`, `sp_atk`, `attack`, `speed`, `defense`, `hp`, and `sp_def`.

- Dummy-code `legendary` and `generation`;
- Center and scale all predictors.

```
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense+ hp+sp_def
  step_dummy(all_nominal_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_center(all_predictors())
```

Exercise 5

We'll be fitting and tuning an elastic net, tuning `penalty` and `mixture` (use `multinom_reg` with the `glmnet` engine).

Set up this model and workflow. Create a regular grid for `penalty` and `mixture` with 10 levels each; `mixture` should range from 0 to 1. For this assignment, we'll let `penalty` range from -5 to 5 (it's log-scaled).

How many total models will you be fitting when you fit these models to your folded data?

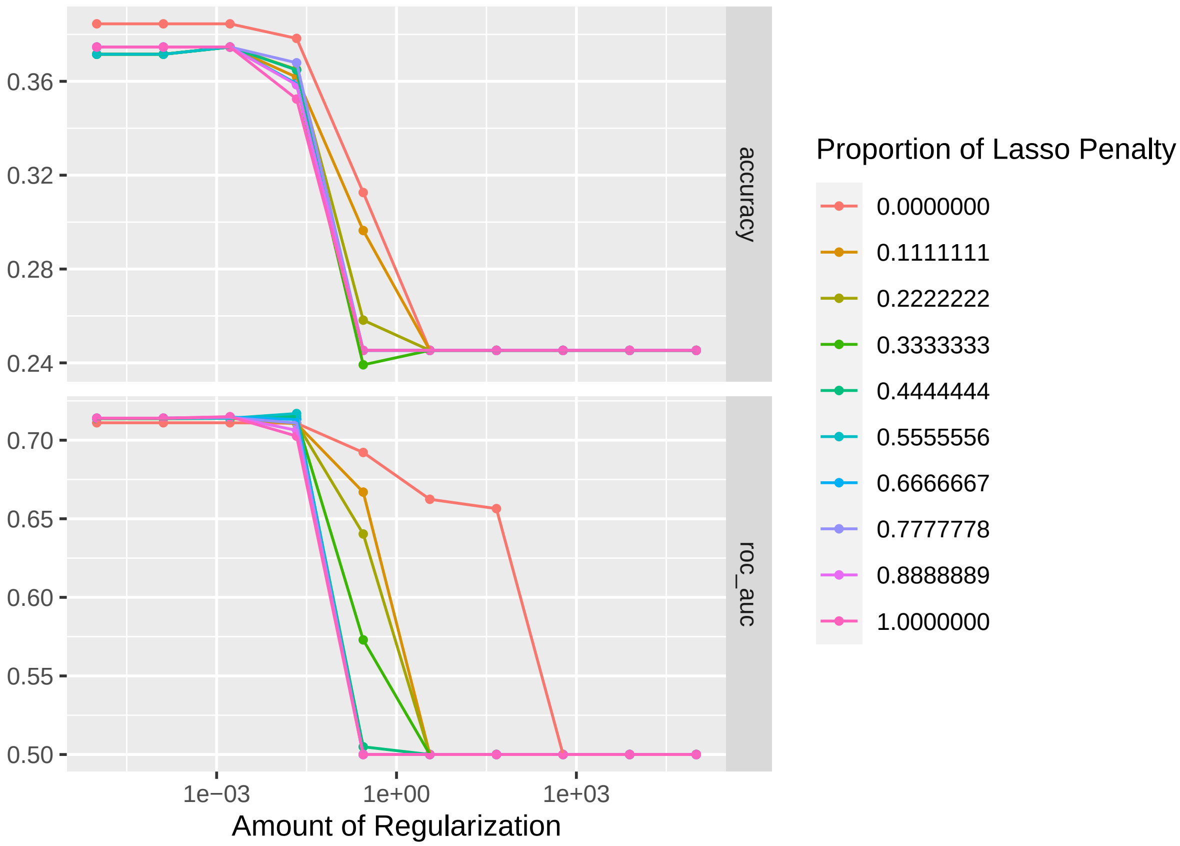
```
spec <- multinom_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet") %>%
  set_mode("classification")


grid <- grid_regular(penalty(c(-5,5)),mixture(c(0,1)),levels = 10)

wf <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(spec)
```

There are a total of 10 models fitting to 5 folds of data.

Exercise 6

Fit the models to your folded data using `tune_grid()`.

```
tune_res <- tune_grid(
  wf,
  resamples = pokemon_folds,
  grid = grid
)
autoplot(tune_res)
```



Use `autoplot()` on the results. What do you notice? Do larger or smaller values of `penalty` and `mixture` produce better accuracy and ROC AUC?

I notice that smaller lasso penalty produce better accuracy and ROC AUC. And smaller mixture produce better accuracy and ROC AUC.

Exercise 7

Use `select_best()` to choose the model that has the optimal `roc_auc`. Then use `finalize_workflow()`, `fit()`, and `augment()` to fit the model to the training set and evaluate its performance on the testing set.

```
final_model = wf %>%
finalize_workflow(select_best(tune_res, metric = "roc_auc")) %>%
fit(pokemon_train) %>% #fit data to training data
augment(pokemon_test)

accuracy(final_model, truth = type_1, estimate = .pred_class) #get accuracy
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.364
```

Exercise 8

Calculate the overall ROC AUC on the testing set.

```
roc_auc(final_model, truth = type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Water, .pr
```

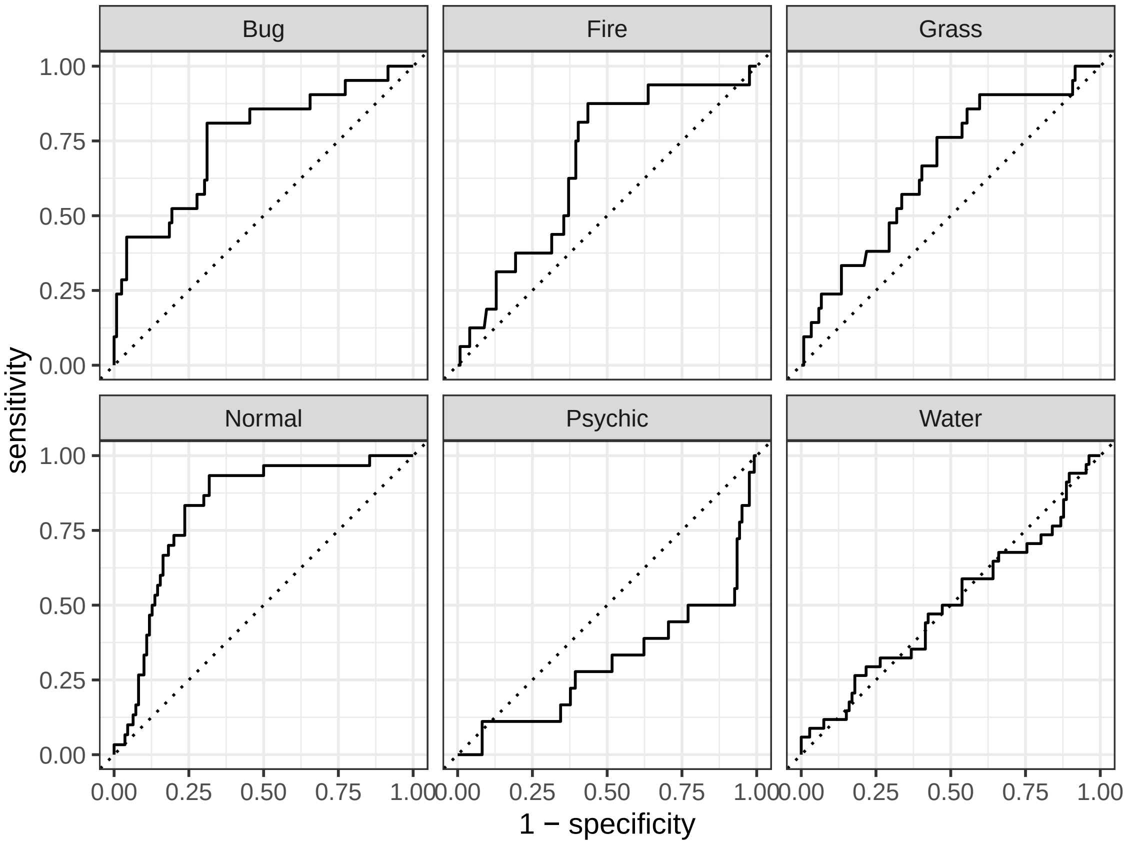
```
## # A tibble: 1 x 3
```

```
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.615
```

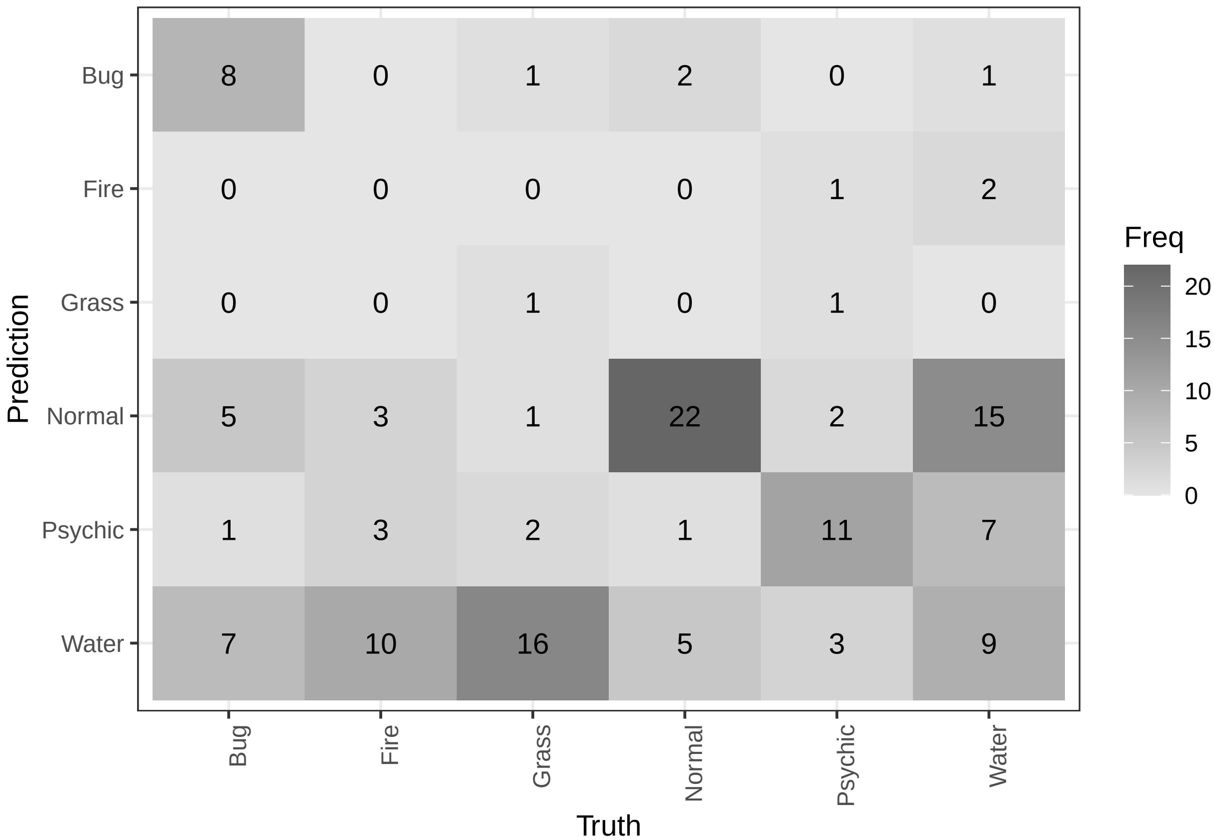
Then create plots of the different ROC curves, one per level of the outcome. Also make a heat map of the confusion matrix.

What do you notice? How did your model do? Which Pokemon types is the model best at predicting, and which is it worst at? Do you have any ideas why this might be?

```
autoplot(roc_curve(final_model, truth = type_1, .pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pre
```




```
conf_mat(final_model, truth = type_1, estimate = .pred_class) %>% #calclate confusion matri
autoplot(type = "heatmap") + #autoplot with a heatmap
theme_bw() + #change theme
theme(axis.text.x = element_text(angle = 90, hjust=1)) #rotate x axis labels
```



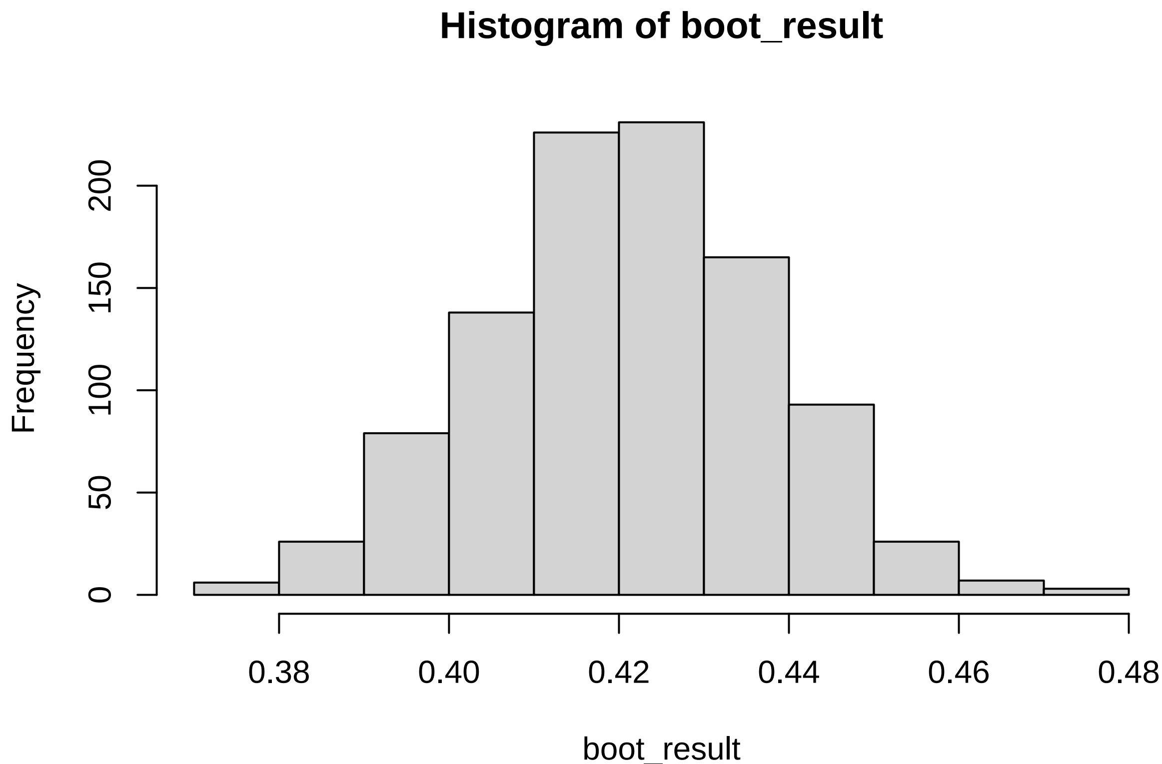
I see that the overall roc_auc is only 0.602 which is not very high. The model performed badly because the accuracy is only ~.37 and the ROC AUC is only ~0.6. The model is not very good at predicting the type of pokemon. However, from the roc curve we can see that the model did well in predicting Normal pokemons, and is worst at predicting psychic, since the roc curve is below the diagonal. This might due to the fact that the Psychic pokemons are different than other pokemons, and the Normal pokemons are similar to other pokemons. And there is more normal pokemon and less psychic pokemon.

For 231 Students

Exercise 9

In the 2020-2021 season, Stephen Curry, an NBA basketball player, made 337 out of 801 three point shot attempts (42.1%). Use bootstrap resampling on a sequence of 337 1's (makes) and 464 0's (misses). For each bootstrap sample, compute and save the sample mean (e.g. bootstrap FG% for the player). Use 1000 bootstrap samples to plot a histogram of those values. Compute the 99% bootstrap confidence interval for Stephen Curry's "true" end-of-season FG% using the quantile function in R. Print the endpoints of this interval.

```
make <- array(1, 337)
miss <- array(0, 464)
shots <- c(make, miss)
N <- length(shots)
boot_result <- numeric(1000)
for (i in 1:1000) {
  boot_samp <- sample(shots, replace = TRUE)
  boot_result[i] <- mean(boot_samp)
}
hist(boot_result)
```


Histogram of boot_result
Frequency
0
50
100
150
200
0.38
0.40
0.42
0.44
0.46
0.48
boot_result