HW5

```
library(readr)
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 0.2.0 --

## v broom        0.7.12     v rsample      0.1.1
## v dials        0.1.0      v tibble       3.1.6
## v infer        1.0.0      v tidyr        1.2.0
## v modeldata    0.1.1      v tune         0.2.0
## v parsnip      0.2.1      v workflows    0.2.6
## v purrr        0.3.4      v workflowsets 0.2.1
## v recipes      0.2.0      v yardstick    0.0.9

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x purrr::discard()  masks scales::discard()
## x dplyr::filter()   masks stats::filter()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Search for functions across packages at https://www.tidymodels.org/find/
```

```
library(ISLR)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v stringr 1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x scales::col_factor() masks readr::col_factor()
## x purrr::discard()     masks scales::discard()
## x dplyr::filter()      masks stats::filter()
## x stringr::fixed()     masks recipes::fixed()
## x dplyr::lag()         masks stats::lag()
## x yardstick::spec()    masks readr::spec()
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-4
```

```
tidymodels_prefer()
r = getOption("repos")
r["CRAN"] = "http://cran.us.r-project.org"
options(repos = r)
install.packages("weatherData")
```

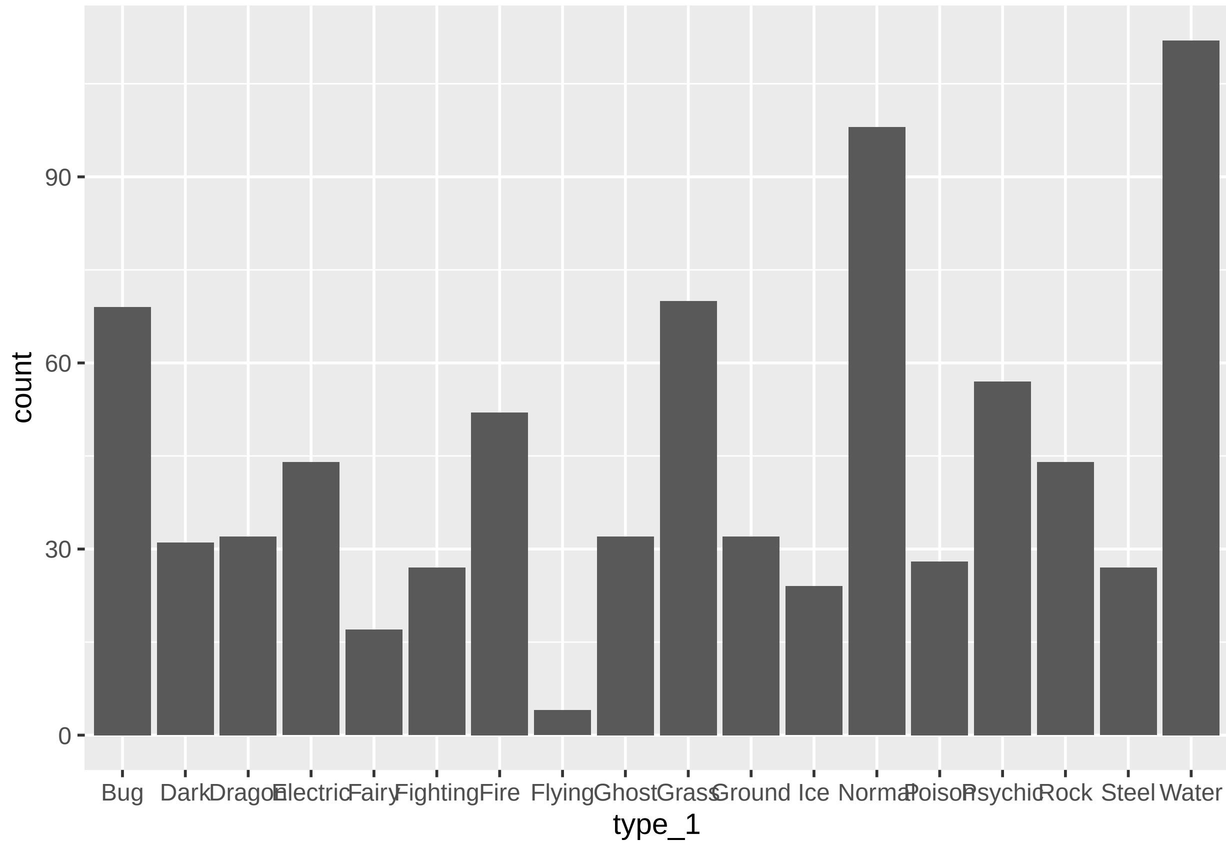
```
## Warning: package 'weatherData' is not available for this version of R
##
## A version of this package for your version of R might be available elsewhere,
## see the ideas at
## https://cran.r-project.org/doc/manuals/r-patched/R-admin.html#Installing-packages
```

Question 1

The data's object and column names are all converted to snake case (words separated by underscores like_this). Clean_names() is useful because it handles every kind of messy column name that's present in the data set and makes it easier to call when piping with "%>%".

Question 2

```
#bar chart
Pokemon_clean %>%
  ggplot(aes(x=type_1)) +
  geom_bar()
```




```r
table(Pokemon_clean$type_1)
```

```
##
##      Bug     Dark   Dragon Electric    Fairy Fighting     Fire   Flying
##       69       31       32       44       17       27       52        4
##    Ghost    Grass   Ground      Ice   Normal   Poison  Psychic     Rock
##       32       70       32       24       98       28       57       44
##    Steel    Water
##       27      112
```

```r
#filter out by specific classes
Pokemon <- Pokemon_clean %>%
  filter(type_1 %in% c("Bug", "Fire", "Grass", "Normal", "Water", "Psychic"))

#factor type_1 and legendary
Pokemon$type_1 <- as.factor(Pokemon$type_1)
Pokemon$legendary <- as.factor(Pokemon$legendary)

sapply(Pokemon,class)
```

```
##      number        name      type_1      type_2       total          hp
##   "numeric" "character"    "factor" "character"   "numeric"   "numeric"
##      attack     defense      sp_atk      sp_def       speed  generation
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##   legendary
```

```
##    "factor"
```

```
View(Pokemon)
```

There are 18 classes. The flying class appears to have very few Pokemon in comparison to the rest (4), but the fairy, fighting, flying, ice, poison, and steel are all classes that have less than 30 pokemon in it.

Question 3

```
#initial split
set.seed(458)
Pokemon_split <- initial_split(Pokemon, strata = "type_1", prop = 0.7)
Pokemon_train <- training(Pokemon_split)
Pokemon_test <- testing(Pokemon_split)

dim(Pokemon_train)
```

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

```
dim(Pokemon_test)
```

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

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

```
## #  5-fold cross-validation using stratification
## # A tibble: 5 x 2
##   splits           id
##   <list>           <chr>
## 1 <split [252/66]> Fold1
## 2 <split [253/65]> Fold2
## 3 <split [253/65]> Fold3
## 4 <split [256/62]> Fold4
## 5 <split [258/60]> Fold5
```

Stratifying the folds can be useful in ensuring that each fold of the dataset has the same proportion of observations with a given label.

Question 4

```
#set up a recipe
pokemon_recipe <- recipe(type_1 ~ legendary + generation + sp_atk + attack + speed + defense + hp + sp_
  step_dummy(c(legendary, generation)) %>%
  step_normalize(all_predictors())
```

Question 5

```
pokemon_reg <- multinom_reg(mixture = 0, penalty = 0) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

pokemon_grid <- grid_regular(penalty(range = c(-5, 5)), mixture(range = c(0,1)), levels = 10)
pokemon_grid
```

```
## # A tibble: 100 x 2
##          penalty mixture
##            <dbl>   <dbl>
##  1      0.00001        0
##  2      0.000129       0
##  3      0.00167        0
##  4      0.0215         0
##  5      0.278          0
##  6      3.59           0
##  7     46.4            0
##  8    599.             0
##  9   7743.             0
## 10 100000              0
## # ... with 90 more rows
```

10 (penalties) x 2 (mixtures) x 5 (folds) = 100 models that will be fitted when fitting to the folded data

Question 6

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

pokemon_workflow <- workflow() %>%
  add_recipe(pokemon_recipe) %>%
  add_model(pokemon_spec)

pokemon_res <- tune_grid(
  pokemon_workflow,
  resamples = pokemon_folds,
  grid = pokemon_grid
)
```

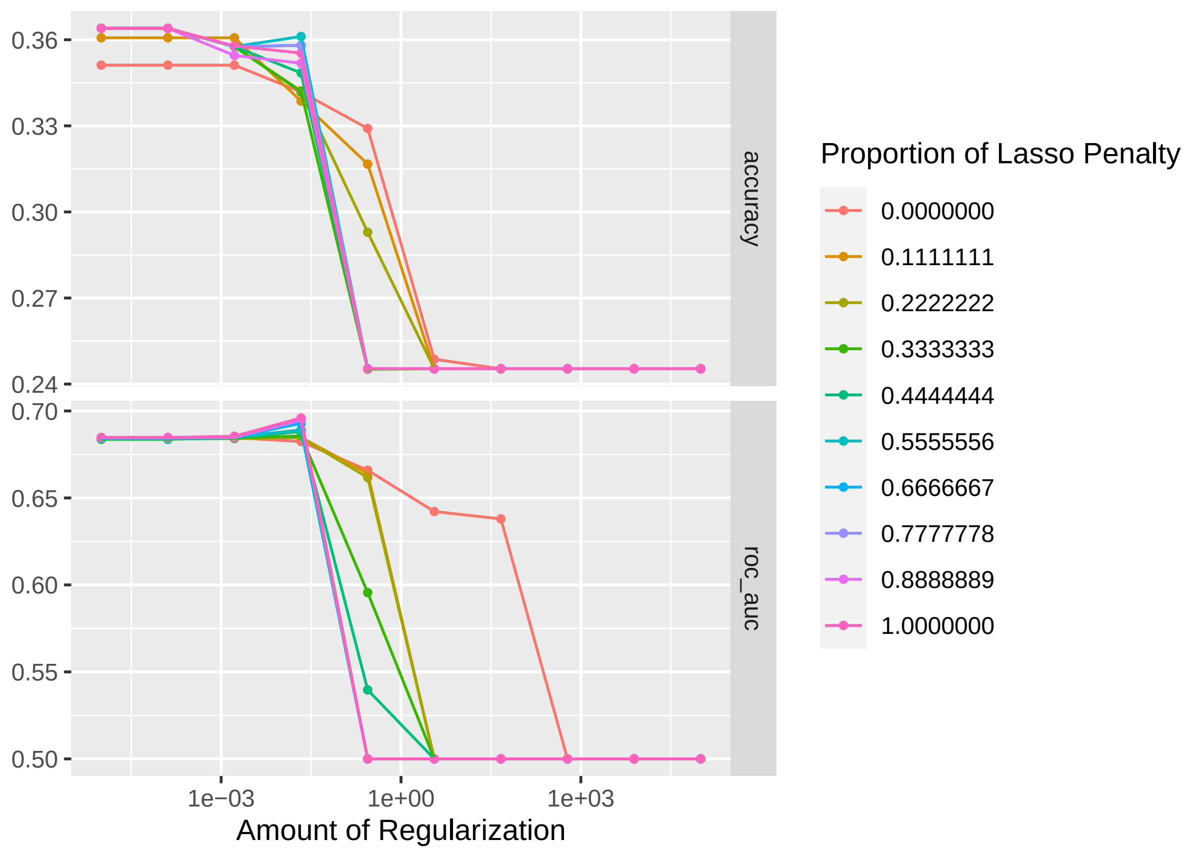
```
## ! Fold1: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold2: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold3: preprocessor 1/1: The following variables are not factor vectors and wil...

## ! Fold4: preprocessor 1/1: The following variables are not factor vectors and wil...
```

```
## ! Fold5: preprocessor 1/1: The following variables are not factor vectors and wil...
```

```
pokemon_res
```

```
## # Tuning results
## # 5-fold cross-validation using stratification
## # A tibble: 5 x 4
##   splits           id    .metrics           .notes
##   <list>           <chr> <list>             <list>
## 1 <split [252/66]> Fold1 <tibble [200 x 6]> <tibble [1 x 3]>
## 2 <split [253/65]> Fold2 <tibble [200 x 6]> <tibble [1 x 3]>
## 3 <split [253/65]> Fold3 <tibble [200 x 6]> <tibble [1 x 3]>
## 4 <split [256/62]> Fold4 <tibble [200 x 6]> <tibble [1 x 3]>
## 5 <split [258/60]> Fold5 <tibble [200 x 6]> <tibble [1 x 3]>
##
## There were issues with some computations:
##
##   - Warning(s) x5: The following variables are not factor vectors and will be ignore...
##
## Use `collect_notes(object)` for more information.
```

```
autoplot(pokemon_res)
```



```r
collect_metrics(pokemon_res)
```

```
## # A tibble: 200 x 8
##     penalty mixture .metric  .estimator  mean     n std_err .config
##       <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.00001        0 accuracy multiclass 0.351     5  0.0178 Preprocessor1_Model~
##  2 0.00001        0 roc_auc  hand_till  0.685     5  0.0196 Preprocessor1_Model~
##  3 0.000129       0 accuracy multiclass 0.351     5  0.0178 Preprocessor1_Model~
##  4 0.000129       0 roc_auc  hand_till  0.685     5  0.0196 Preprocessor1_Model~
##  5 0.00167        0 accuracy multiclass 0.351     5  0.0178 Preprocessor1_Model~
##  6 0.00167        0 roc_auc  hand_till  0.685     5  0.0196 Preprocessor1_Model~
##  7 0.0215         0 accuracy multiclass 0.342     5  0.0189 Preprocessor1_Model~
##  8 0.0215         0 roc_auc  hand_till  0.682     5  0.0201 Preprocessor1_Model~
##  9 0.278          0 accuracy multiclass 0.329     5  0.0306 Preprocessor1_Model~
## 10 0.278          0 roc_auc  hand_till  0.666     5  0.0230 Preprocessor1_Model~
## # ... with 190 more rows
```

From the plot, it appears that smaller values of penalty and mixture produce better accuracy and ROC AUC.

Question 7

```r
best_penalty <- select_best(pokemon_res, metric = "roc_auc")
best_penalty
```

```
## # A tibble: 1 x 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1  0.0215       1 Preprocessor1_Model094
```

```r
pokemon_final <- finalize_workflow(pokemon_workflow, best_penalty)

pokemon_final_fit <- fit(pokemon_final, data = Pokemon_train)
```

```
## Warning: The following variables are not factor vectors and will be ignored:
## `generation`
```

```r
augment(pokemon_final_fit, new_data = Pokemon_test)
```

```
## # A tibble: 140 x 20
##    number name      type_1 type_2 total    hp attack defense sp_atk sp_def speed
##     <dbl> <chr>     <fct>  <chr>  <dbl> <dbl>  <dbl>   <dbl>  <dbl>  <dbl> <dbl>
##  1      1 Bulbasaur Grass  Poison   318    45     49      49     65     65    45
##  2      2 Ivysaur   Grass  Poison   405    60     62      63     80     80    60
##  3      6 Charizar~ Fire   Dragon   634    78    130     111    130     85   100
##  4      6 Charizar~ Fire   Flying   634    78    104      78    159    115   100
##  5      9 Blastoise Water  <NA>     530    79     83     100     85    105    78
##  6     12 Butterfr~ Bug    Flying   395    60     45      50     90     80    70
```

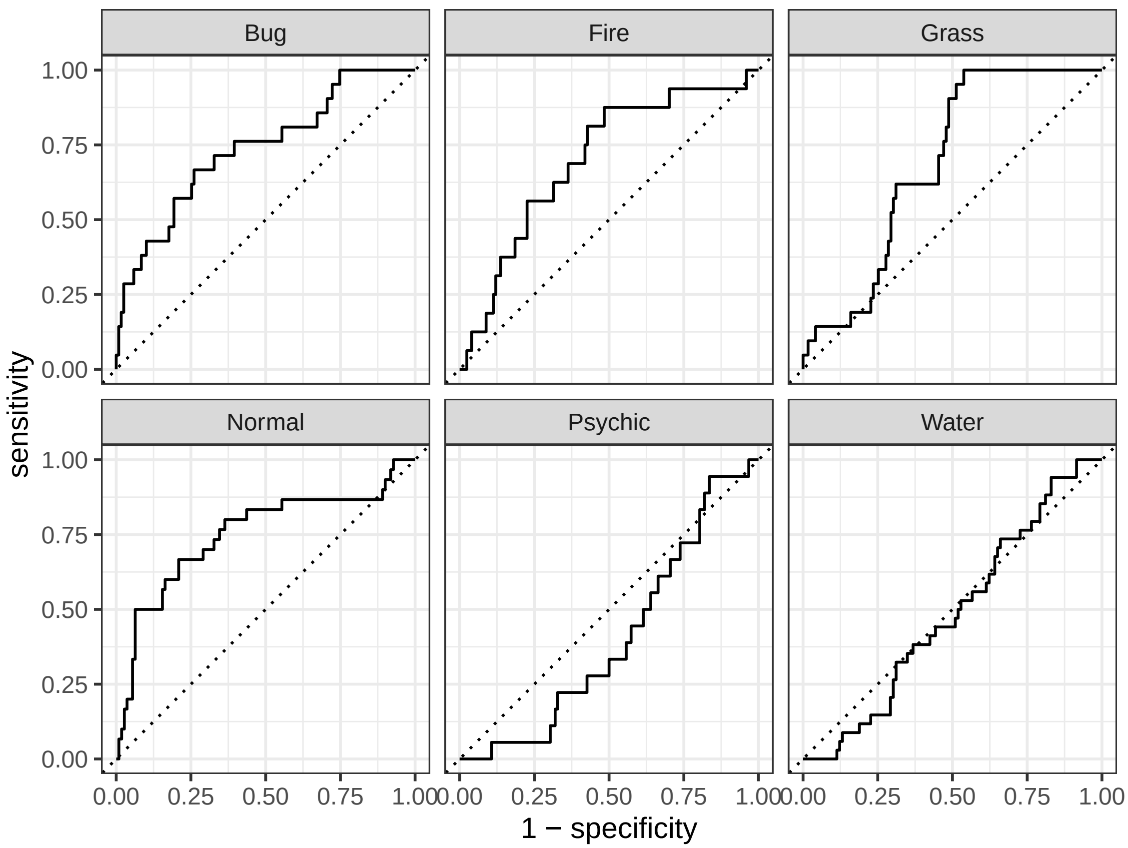
```
## 7       16 Pidgey     Normal Flying    251     40      45       40      35      35     56
## 8       17 Pidgeotto  Normal Flying    349     63      60       55      50      50     71
## 9       18 PidgeotM~  Normal Flying    579     83      80       80     135      80    121
## 10      21 Spearow    Normal Flying    262     40      60       30      31      31     70
## # ... with 130 more rows, and 9 more variables: generation <dbl>,
## #   legendary <fct>, .pred_class <fct>, .pred_Bug <dbl>, .pred_Fire <dbl>,
## #   .pred_Grass <dbl>, .pred_Normal <dbl>, .pred_Psychic <dbl>,
## #   .pred_Water <dbl>
```

Question 8

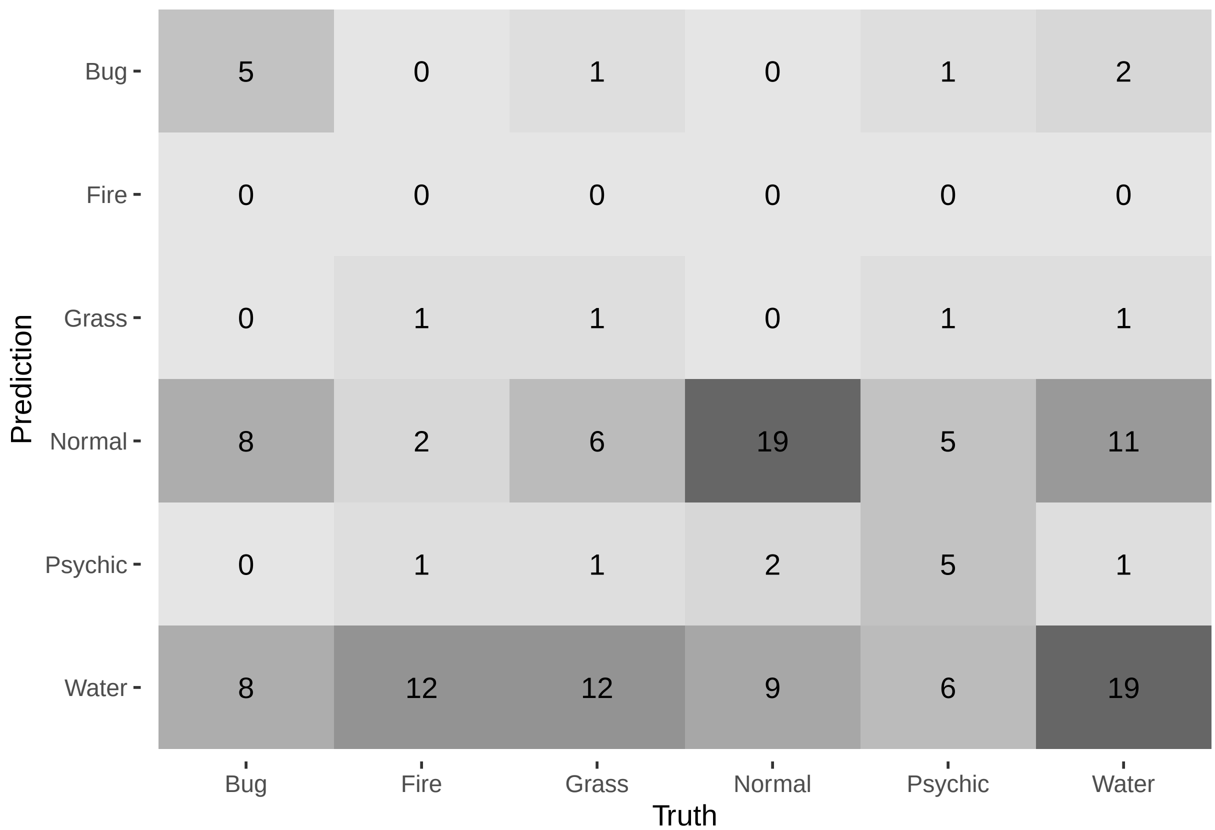
```
augment(pokemon_final_fit, new_data = Pokemon_test) %>%
  roc_auc(truth = type_1, estimate = c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Water, .p
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.619
```

```
augment(pokemon_final_fit, new_data = Pokemon_test) %>%
  roc_curve(truth = type_1, estimate = c(.pred_Bug, .pred_Fire, .pred_Grass, .pred_Normal, .pred_Water,
  autoplot()
```



```r
augment(pokemon_final_fit, new_data = Pokemon_test) %>%
  conf_mat(truth = type_1, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```



In terms of performance, bug, fire, grass, and normal are all the types that the model is best at predicting. Comparing each curve to one another and trying to average out the performance, I believe the model performed relatively accurate, since the psychic and water predictions were not as good as the other types. This may be because the psychic and water types had more missing values, more specifically, missing type 2 values that gave us less data to work with.