Lab 17: KNN Classification, Cross-validation, Decision boundary plots

Example: Forensic Glass Classification

This data set is provided as part of the MASS package for R, and the description below borrows from the data documentation provided in that package.

We have measurements of different attributes of fragments of glass, as well as the type of glass. If we could develop a reliable mechanism for classifying glass, glass shards found at crime scenes could potentially be used as evidence in criminal trials. There are 7 glass types in the originally published data, but only 6 occur in this data set: "window float glass (WinF: 70), window non-float glass (WinNF: 76), vehicle window glass (Veh: 17), containers (Con: 13), tableware (Tabl: 9) and vehicle headlamps (Head: 29)".

For each glass fragment, we have measurements of the refractive index of the glass, as well as the percentage by weight of 8 different elements in the glass. These are our explanatory/predictive variables.

Read in data, train/test split, validation splits

I've written this code for you. No need to update.

```
library(readr)
library(dplyr)
library(ggplot2)
library(GGally)
library(gridExtra)
library(caret)

set.seed(9215)

# Read in data, tqke a first look
glass <- read_csv("http://www.evanlray.com/data/mass/fgl.csv") %>%
  mutate(type = factor(type))

head(glass)
```

```
## # A tibble: 6 x 10
##       RI    Na    Mg    Al    Si     K    Ca    Ba    Fe type
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>
## 1  3.01   13.6  4.49  1.1   71.8 0.06   8.75     0  0    WinF
## 2 -0.39   13.9  3.6   1.36  72.7 0.48   7.83     0  0    WinF
## 3 -1.82   13.5  3.55  1.54  73.0 0.39   7.78     0  0    WinF
## 4 -0.340  13.2  3.69  1.29  72.6 0.570  8.22     0  0    WinF
## 5 -0.580  13.3  3.62  1.24  73.1 0.55   8.07     0  0    WinF
## 6 -2.04   12.8  3.61  1.62  73.0 0.64   8.07     0  0.26 WinF
```

```
dim(glass)
```

```
## [1] 214  10
```

```
# Train/test split
tt_inds <- caret::createDataPartition(glass$type, p = 0.7)

train_glass <- glass %>% slice(tt_inds[[1]])
test_glass <- glass %>% slice(-tt_inds[[1]])

# Cross-validation splits
crossval_val_fold_inds <- caret::createFolds(
  y = train_glass$type, # response variable as a vector
```

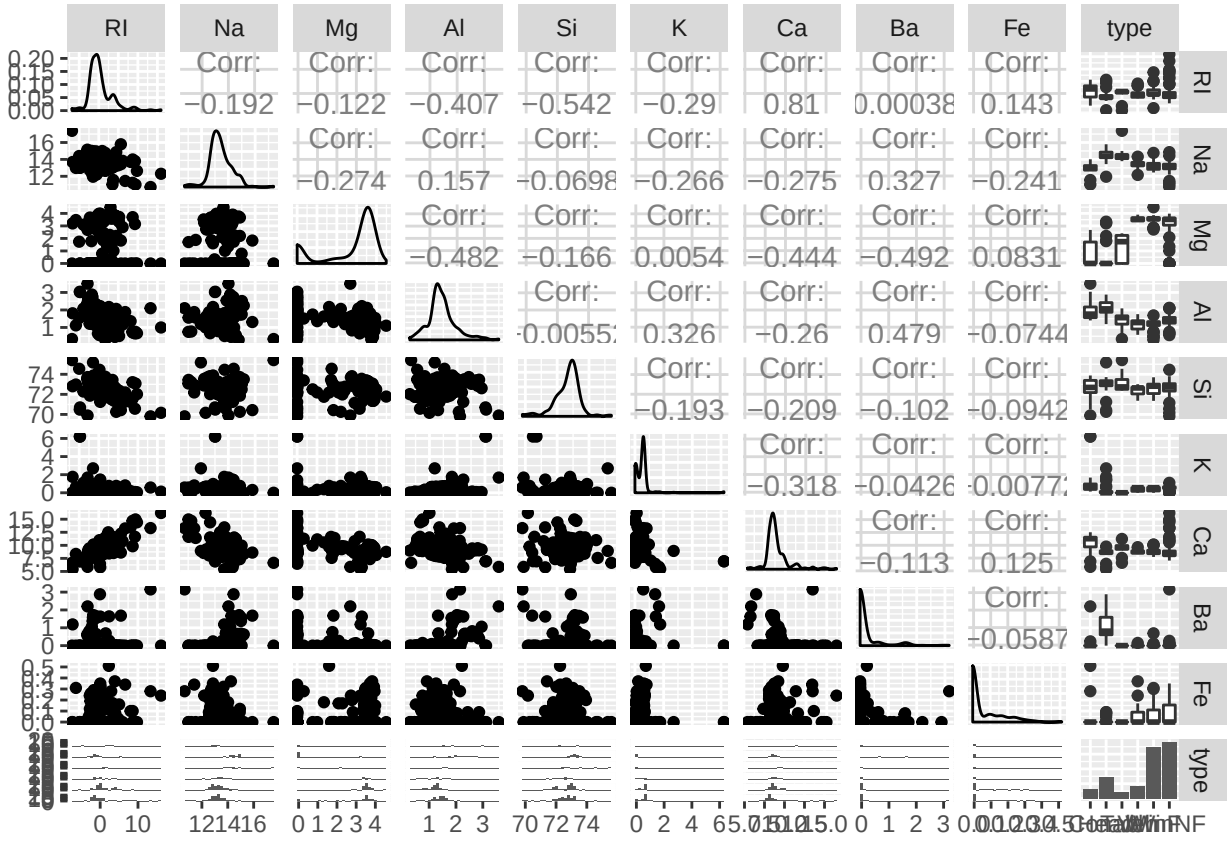
```
  k = 10 # number of folds for cross-validation
)
```

1. Make some exploratory plots of the training data.

Your goal is to understand the data well enough that you can identify two features (two of the measured attributes) that would be useful for classifying glass in part 2. Ideally, each feature you choose will individually have something to say about glass type (for example, maybe the distribution of values for the feature is different for the different glass types), and there will not be a strong relationship between the two features you choose (so they carry different information about glass type and you don't just have basically the same information from your two features). Don't spend a huge amount of time on this though. A pairs plot could be good enough.

```
vars <- c('RI', "Na", "Mg", "Al", "Si","K","Ca", "Ba", "Fe", "type")
ggpairs(glass%>%select(vars) )
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



I will use Na and Al.

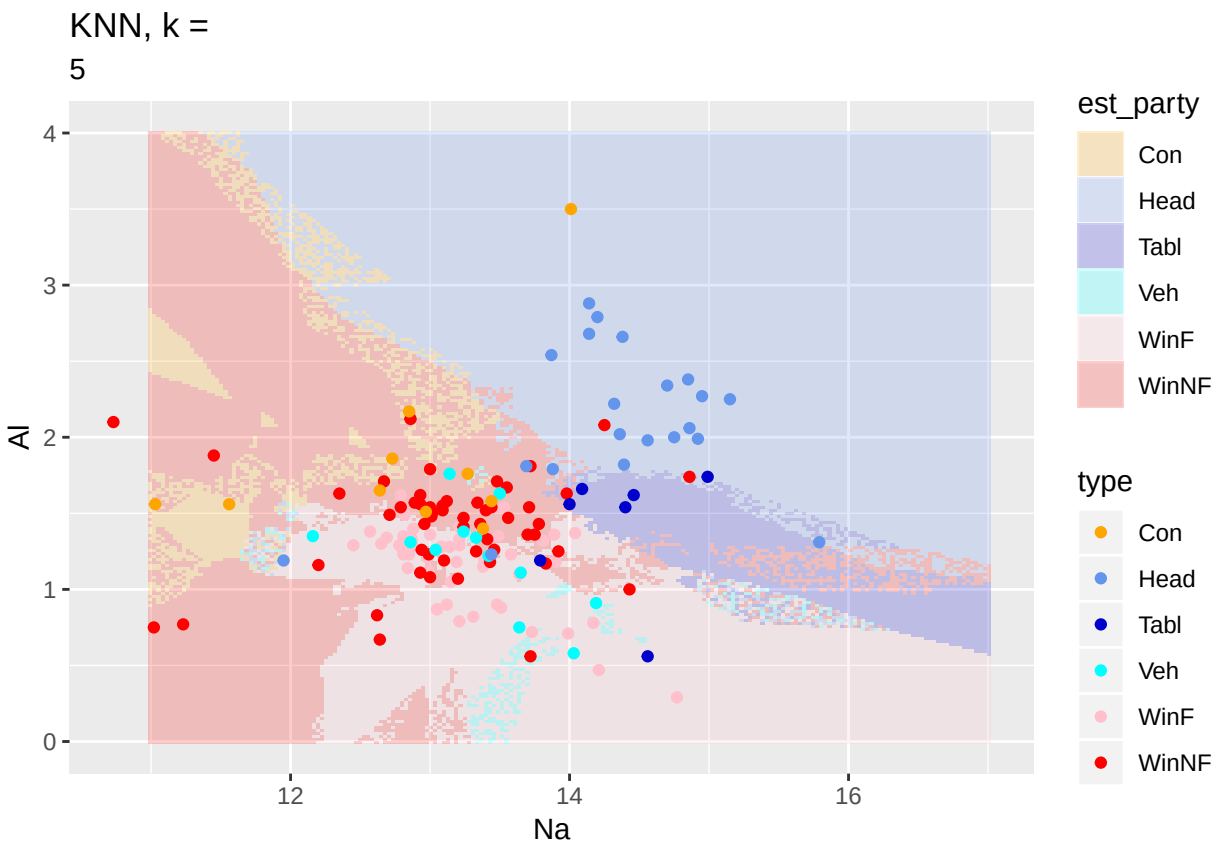
2. Using your two selected features, fit a KNN model and make a plot showing the decision boundaries in the two-dimensional feature space. Optional, if time: make a few different versions of the plot with different values of k to see how the decision boundaries change with the number of neighbors used.

To save you some time retyping all my code from the handout, I'm pasting in the code I used for the party affiliation

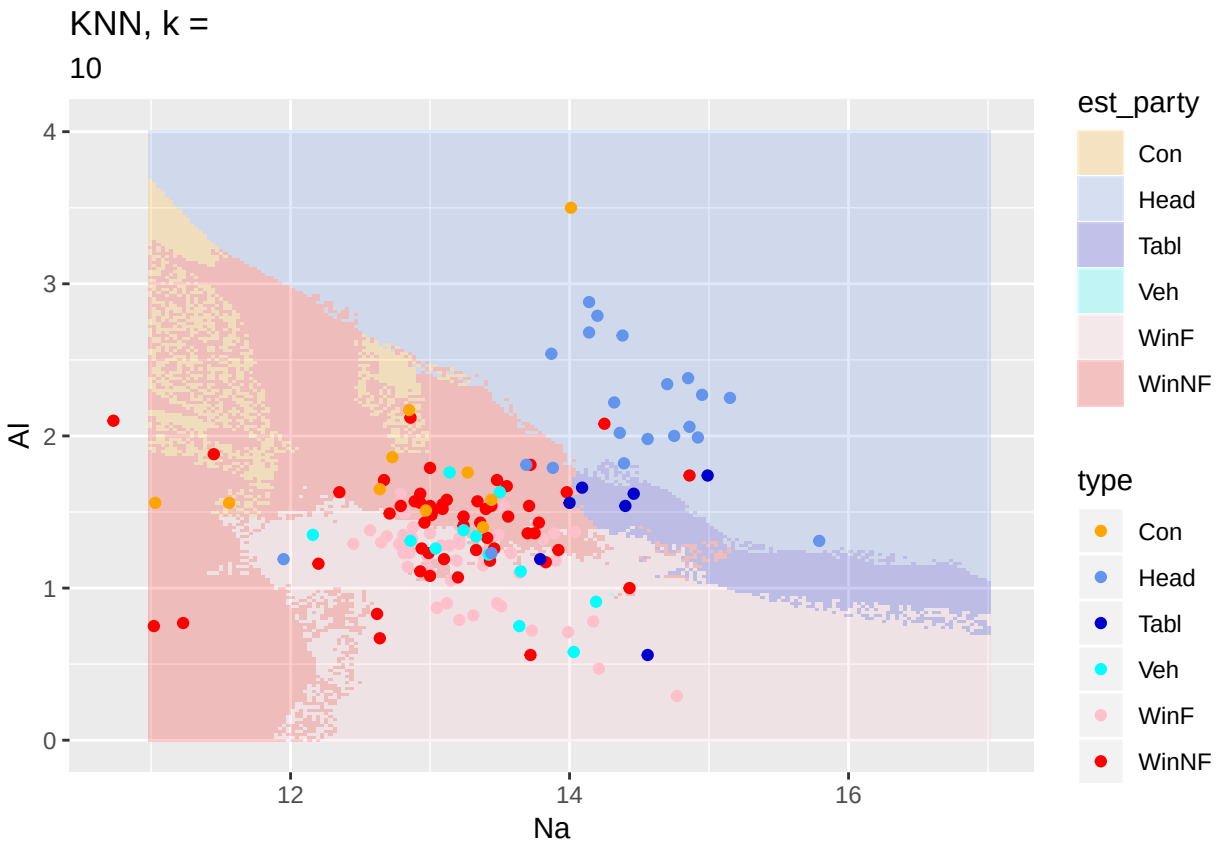
example below. You will need to make the appropriate changes to tailor the code to the data set you're working with in this lab.

```
# "train" the KNN model
train_and_plot_knn <- function(neighbour){


knn_fit <- train(
  form = type ~ Na + Al,
  data = train_glass,
  method = "knn",
  preProcess = "scale",
  trControl = trainControl(method = "none"),
  tuneGrid = data.frame(k = neighbour)
)

# a grid of values for age and popul at which to get the estimated class.
# it's not a test data set in the sense that we don't have observations of party to go with these points,
# but we will treat it as a "test set" in the sense that we will obtain predictions at these points
test_grid <- expand.grid(
  Na = seq(from = 11, to = 17, length = 201),
  Al = seq(from = 0, to = 4, length = 201)
)
head(test_grid)

# use predict to find the estimated most likely class at each point in our grid
y_hats <- predict(knn_fit, newdata = test_grid, type = "raw")

# add the estimated types into the test_grid data frame
background_knn <- test_grid %>%
  mutate(
    est_party = y_hats
  )

# make the plot.  geom_raster does the shading in the background, alpha = 0.2 makes it transparent
ggplot() +
  geom_raster(data = background_knn,
    mapping = aes(x = Na, y = Al, fill = est_party), alpha = 0.2) +
  geom_point(data = train_glass, mapping = aes(x = Na, y = Al, color = type)) +
  scale_color_manual("type", values = c("orange", "cornflowerblue", "mediumblue","cyan", "pink", "red")) +
  scale_fill_manual(values = c("orange", "cornflowerblue", "mediumblue","cyan", "pink", "red")) +
  ggtitle("KNN, k = " ,neighbour)
}
```

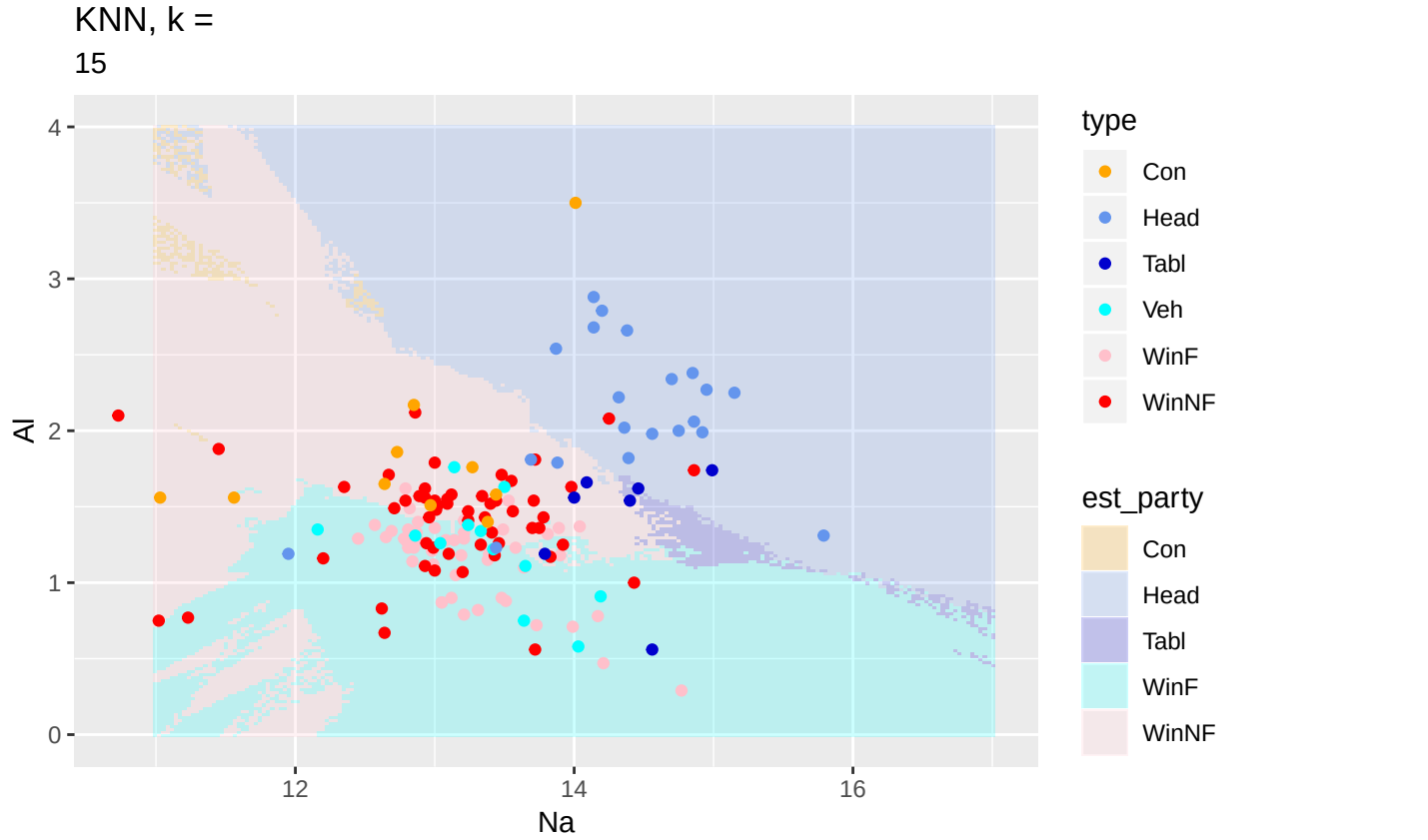
```
train_and_plot_knn(5)
```



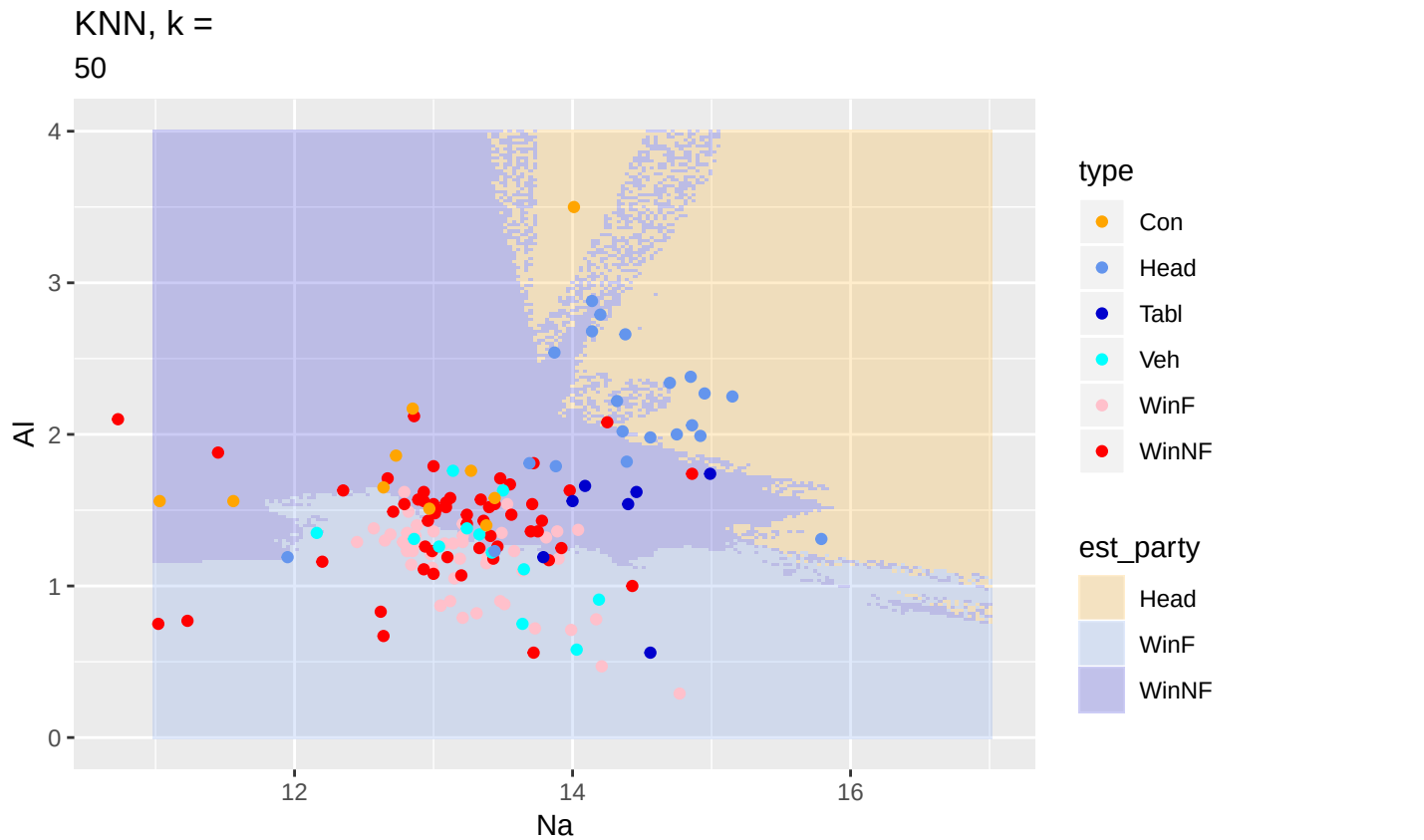

```
train_and_plot_knn(10)
```



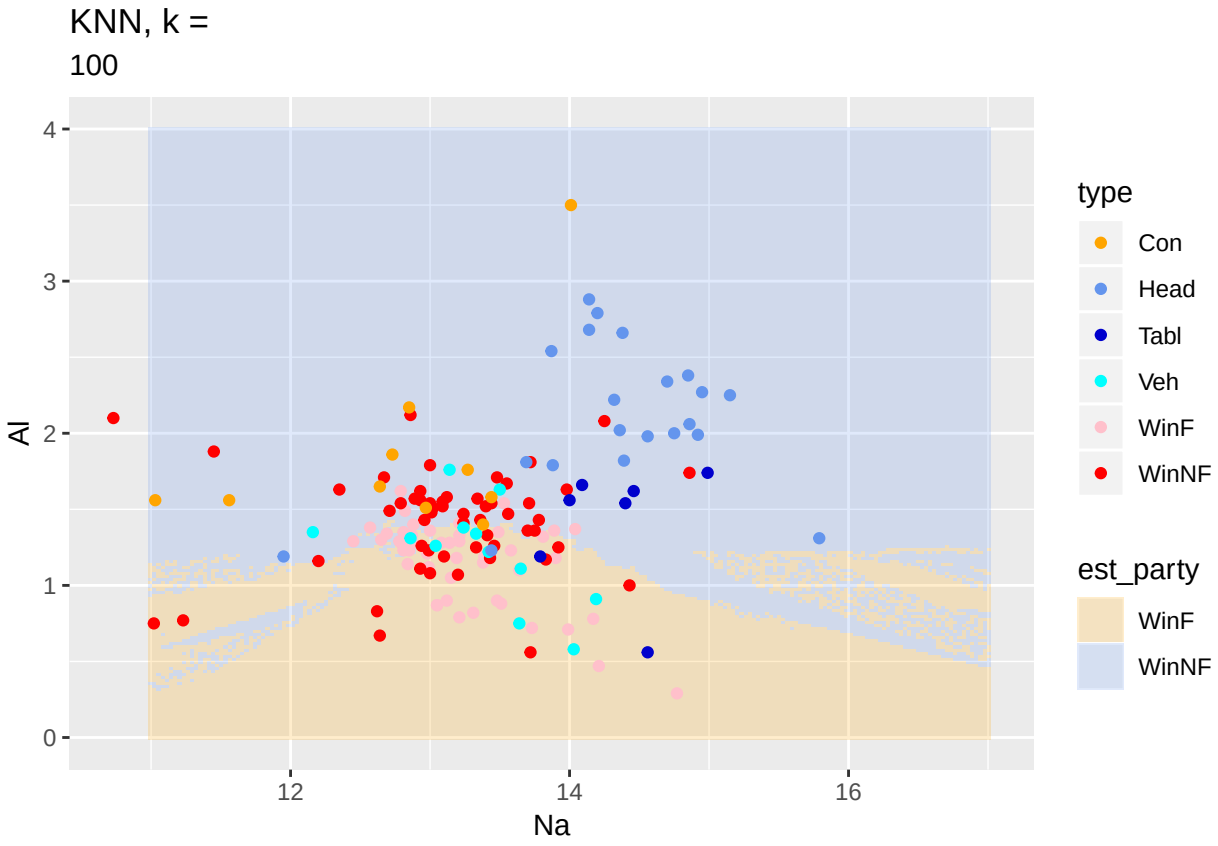

```
train_and_plot_knn(15)
```




```
train_and_plot_knn(50)
```




```
train_and_plot_knn(100)
```



3. Use cross-validation to evaluate the classification performance of a KNN classifier using all 9 features for a few values of k.

Your code should do the following things:

1. Allocate space to store your cross-validation results. At a minimum, this will be a vector of values long enough to store the validation set classification error rate for each validation fold
2. For i = 1, . . . , number of validation folds
   a. Assemble training set data and validation set data specific to the index i
   b. Fit your model to the training set data
   c. Generate predictions for the validation set data
   d. Calculate validation set classification error rate and save it in the space you allocated in step 1 above.
3. Calculate the mean classification error rate across all 10 validation folds.

You will need to do this for each value of k you investigate. For today, you can organize this in any way that makes sense to you. Options include using two nested for loops where one iterates over the values of k you're looking at and the other iterates over cross-validation folds; manually doing steps b, c, and d for each model within a single for loop; or repeating your whole block of code multiple times, once for each value of k.

```
calculate_val_mse <- function(neighbour){
val_mse <- rep(NA, 10)

for(i in seq_len(10)) {
  train <- train_glass %>% slice(-crossval_val_fold_inds[[i]])
  val <- train_glass %>% slice(crossval_val_fold_inds[[i]])
  knn_fit <- train(
  form = type ~ RI + Na + Mg + Al + Si + K + Ca + Ba + Fe,
  data = train,
  method = "knn",
  preProcess = "scale",
  trControl = trainControl(method = "none"),
  tuneGrid = data.frame(k = neighbour)
)
```

```
  y_hats <- predict(knn_fit, newdata = val, type = "raw")
  val_mse[i] <- mean(y_hats != val$type)
}
mean(val_mse)
}

calculate_val_mse(5)
```

```
## [1] 0.3679762
```

```
calculate_val_mse(10)
```

```
## [1] 0.3344643
```

```
calculate_val_mse(15)
```

```
## [1] 0.4000595
```

```
calculate_val_mse(50)
```

```
## [1] 0.6191667
```

```
calculate_val_mse(100)
```

```
## [1] 0.6717262
```

4. Using your selected value of k from cross-validation, refit your KNN model to the full training set and get the test set error rate. How does your performance compare to what you'd get if you just predicted the most common class in the training set?

Worse than predicting the most common class.

```
knn_fit <- train(
  form = type ~ RI + Na + Mg + Al + Si + K + Ca + Ba + Fe,
  data = train_glass,
  method = "knn",
  preProcess = "scale",
  trControl = trainControl(method = "none"),
  tuneGrid = data.frame(k = 10)
)
y_hats <- predict(knn_fit, newdata = test_glass, type = "raw")
mean(y_hats != test_glass$type)
```

```
## [1] 0.2786885
```

```
mean("WinNF" != test_glass$type)
```

```
## [1] 0.6393443
```