Predicting the Class of Mushrooms

Syaqira Farouk

5/18/2020

Introduction

Regression analysis can be used both for regression and classification. In this analysis we will be testing several types of models to predict the class of various mushrooms (classification). The data can be found on this website:

https://www.kaggle.com/uciml/mushroom-classification (https://www.kaggle.com/uciml/mushroom-classification)

This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family Mushroom drawn from The Audubon Society Field Guide to North American Mushrooms (1981). Each species is identified as definitely ‘edible’, definitely ‘poisonous’, or of unknown edibility and not recommended. The unknown class was combined with the ‘poisonous’ one.

The models will be compared to each other in terms of ‘accuracy’, ‘sensitivity’, ‘specificity’ and ‘computation time’. The ‘computation time’ should be regarded as a general estimate as different hardwares will affect the result. The objective is to pick the best model for this classification process. We will require these libraries:

```
if (!require("tidyverse")) {
   install.packages("tidyverse")
   library(tidyverse)
}
if (!require("readxl")) {
   install.packages("readxl")
   library(readxl)
}
if (!require("httr")) {
   install.packages("httr")
   library(httr)
}
if (!require("knitr")) {
   install.packages("knitr")
   library(knitr)
}
if (!require("kableExtra")) {
   install.packages("kableExtra")
   library(kableExtra)
}
if (!require("caret")) {
   install.packages("caret")
   library(caret)
}
if (!require("randomForest")) {
   install.packages("randomForest")
   library(randomForest)
}
if (!require("gam")) {
   install.packages("gam")
   library(gam)
}
```

Method/Analysis

We will first load the dataset and view the structure:

```
raw_dat<- read_csv("https://github.com/syaqirafarouk/harvardx-capstone-cyo/raw/master/mushrooms_
raw.csv")

str(raw_dat)
```

```
## tibble [8,124 x 23] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ class                   : chr [1:8124] "p" "e" "e" "p" ...
##  $ cap-shape               : chr [1:8124] "x" "x" "b" "x" ...
##  $ cap-surface             : chr [1:8124] "s" "s" "s" "y" ...
##  $ cap-color               : chr [1:8124] "n" "y" "w" "w" ...
##  $ bruises                 : logi [1:8124] TRUE TRUE TRUE TRUE FALSE TRUE ...
##  $ odor                    : chr [1:8124] "p" "a" "l" "p" ...
##  $ gill-attachment         : logi [1:8124] FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ gill-spacing            : chr [1:8124] "c" "c" "c" "c" ...
##  $ gill-size               : chr [1:8124] "n" "b" "b" "n" ...
##  $ gill-color              : chr [1:8124] "k" "k" "n" "n" ...
##  $ stalk-shape             : chr [1:8124] "e" "e" "e" "e" ...
##  $ stalk-root              : chr [1:8124] "e" "c" "c" "e" ...
##  $ stalk-surface-above-ring: chr [1:8124] "s" "s" "s" "s" ...
##  $ stalk-surface-below-ring: chr [1:8124] "s" "s" "s" "s" ...
##  $ stalk-color-above-ring  : chr [1:8124] "w" "w" "w" "w" ...
##  $ stalk-color-below-ring  : chr [1:8124] "w" "w" "w" "w" ...
##  $ veil-type               : chr [1:8124] "p" "p" "p" "p" ...
##  $ veil-color              : chr [1:8124] "w" "w" "w" "w" ...
##  $ ring-number             : chr [1:8124] "o" "o" "o" "o" ...
##  $ ring-type               : chr [1:8124] "p" "p" "p" "p" ...
##  $ spore-print-color       : chr [1:8124] "k" "n" "n" "k" ...
##  $ population              : chr [1:8124] "s" "n" "n" "s" ...
##  $ habitat                 : chr [1:8124] "u" "g" "m" "u" ...
##  - attr(*, "problems")= tibble [210 x 5] (S3: tbl_df/tbl/data.frame)
##   ..$ row     : int [1:210] 6039 6041 6376 6425 6435 6559 6664 6669 6764 6850 ...
##   ..$ col     : chr [1:210] "gill-attachment" "gill-attachment" "gill-attachment" "gill-attac
hment" ...
##   ..$ expected: chr [1:210] "1/0/T/F/TRUE/FALSE" "1/0/T/F/TRUE/FALSE" "1/0/T/F/TRUE/FALSE"
"1/0/T/F/TRUE/FALSE" ...
##   ..$ actual  : chr [1:210] "a" "a" "a" "a" ...
##   ..$ file    : chr [1:210] "'https://github.com/syaqirafarouk/harvardx-capstone-cyo/raw/mast
er/mushrooms_raw.csv'" "'https://github.com/syaqirafarouk/harvardx-capstone-cyo/raw/master/mushr
ooms_raw.csv'" "'https://github.com/syaqirafarouk/harvardx-capstone-cyo/raw/master/mushrooms_ra
w.csv'" "'https://github.com/syaqirafarouk/harvardx-capstone-cyo/raw/master/mushrooms_raw.csv'"
...
##  - attr(*, "spec")=
##   .. cols(
##   ..   class = col_character(),
##   ..   `cap-shape` = col_character(),
##   ..   `cap-surface` = col_character(),
##   ..   `cap-color` = col_character(),
##   ..   bruises = col_logical(),
##   ..   odor = col_character(),
##   ..   `gill-attachment` = col_logical(),
##   ..   `gill-spacing` = col_character(),
##   ..   `gill-size` = col_character(),
##   ..   `gill-color` = col_character(),
##   ..   `stalk-shape` = col_character(),
##   ..   `stalk-root` = col_character(),
##   ..   `stalk-surface-above-ring` = col_character(),
##   ..   `stalk-surface-below-ring` = col_character(),
##   ..   `stalk-color-above-ring` = col_character(),
```

```
##   ..   `stalk-color-below-ring` = col_character(),
##   ..   `veil-type` = col_character(),
##   ..   `veil-color` = col_character(),
##   ..   `ring-number` = col_character(),
##   ..   `ring-type` = col_character(),
##   ..   `spore-print-color` = col_character(),
##   ..   population = col_character(),
##   ..   habitat = col_character()
##   .. )
```

Also included on the website is the 'attribute information' for our reference:

```
GET("https://github.com/syaqirafarouk/harvardx-capstone-cyo/blob/master/capstone-cyo-attribute-information.xls?raw=true", write_disk(tf<- tempfile(fileext = ".xls")))
```

```
## Response [https://raw.githubusercontent.com/syaqirafarouk/harvardx-capstone-cyo/master/capstone-cyo-attribute-information.xls]
##   Date: 2020-05-21 06:26
##   Status: 200
##   Content-Type: application/octet-stream
##   Size: 30.7 kB
## <ON DISK>  C:\Users\mnstrchr\AppData\Local\Temp\RtmpmmcuVw\file190c604b3ba1.xls
```

```
attributes <- read_excel(tf, sheet = 1)
rm("tf")

kable(attributes, align = "c") %>%
  column_spec(column = 1, bold = TRUE) %>%
  kable_styling(font_size = 4.5 )
```

Original Attribute Information	...2	...3	...4	...5	...6	...7	...8	...9	...10	...11	...12	...13	...14	...15	...16	...17	...18	...19	...20	...21	...22	...23	...24	...25
classes	e	edible	p	poisonous	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
cap-shape	b	bell	c	conical	x	convex	f	flat	k	knobbed	s	sunken	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
cap-surface	f	fibrous	g	grooves	y	scaly	s	smooth	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
cap-color	n	brown	b	buff	c	cinnamon	g	gray	r	green	p	pink	u	purple	e	red	w	white	y	yellow	NA	NA	NA	NA
bruises	t	bruises	f	no	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
odor	a	almond	l	anise	c	creosote	y	fishy	f	foul	m	musty	n	none	p	pungent	s	spicy	NA	NA	NA	NA	NA	NA
gill-attachment	a	attached	d	descending	f	free	n	notched	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
gill-spacing	c	close	w	crowded	d	distant	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
gill-size	b	broad	n	narrow	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
gill-color	k	black	n	brown	b	buff	h	chocolate	g	gray	r	green	o	orange	p	pink	u	purple	e	red	w	white	y	yellow
stalk-shape	e	enlarging	t	tapering	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
stalk-root	b	bulbous	c	club	u	cup	e	equal	z	rhizomorphs	r	rooted	?	missing	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
stalk-surface-above-ring	f	fibrous	y	scaly	k	silky	s	smooth	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
stalk-surface-below-ring	f	fibrous	y	scaly	k	silky	s	smooth	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

Original Attribute Information	...2	...3	...4	...5	...6	...7	...8	...9	...10	...11	...12	...13	...14	...15	...16	...17	...18	...19	...20	...21	...22	...23	...24	...25
stalk-color-above-ring	n	brown	b	buff	c	cinnamon	g	gray	o	orange	p	pink	e	red	w	white	y	yellow	NA	NA	NA	NA	NA	NA
stalk-color-below-ring	n	brown	b	buff	c	cinnamon	g	gray	o	orange	p	pink	e	red	w	white	y	yellow	NA	NA	NA	NA	NA	NA
veil-type	p	partial	u	universal	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
veil-color	n	brown	o	orange	w	white	y	yellow	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
ring-number	n	none	o	one	t	two	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
ring-type	c	cobwebby	e	evanescent	f	flaring	l	large	n	none	p	pendant	s	sheathing	z	zone	NA	NA	NA	NA	NA	NA	NA	NA
spore-print-color	k	black	n	brown	b	buff	h	chocolate	r	green	o	orange	u	purple	w	white	y	yellow	NA	NA	NA	NA	NA	NA
population	a	abundant	c	clustered	n	numerous	s	scattered	v	several	y	solitary	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
habitat	g	grasses	l	leaves	m	meadows	p	paths	u	urban	w	waste	d	woods	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

We see that it is a tibble with 8124 rows and 23 columns. We also see that most columns are of class ‘character’ and we know that they are descriptive. We will now try to convert these columns into factors and put them in a data frame:

```
dat<- as.data.frame(lapply(raw_dat[,1:ncol(raw_dat)],factor))

str(dat)
```

```
## 'data.frame':    8124 obs. of  23 variables:
##  $ class                    : Factor w/ 2 levels "e","p": 2 1 1 2 1 1 1 1 2 1 ...
##  $ cap.shape                : Factor w/ 6 levels "b","c","f","k",..: 6 6 1 6 6 6 1 1 6 1 ...
##  $ cap.surface              : Factor w/ 4 levels "f","g","s","y": 3 3 3 4 3 4 3 4 4 3 ...
##  $ cap.color                : Factor w/ 10 levels "b","c","e","g",..: 5 10 9 9 4 10 9 9 9 10
...
##  $ bruises                  : Factor w/ 2 levels "FALSE","TRUE": 2 2 2 2 1 2 2 2 2 2 ...
##  $ odor                     : Factor w/ 9 levels "a","c","f","l",..: 7 1 4 7 6 1 1 4 7 1 ...
##  $ gill.attachment          : Factor w/ 1 level "FALSE": 1 1 1 1 1 1 1 1 1 1 ...
##  $ gill.spacing             : Factor w/ 2 levels "c","w": 1 1 1 1 2 1 1 1 1 1 ...
##  $ gill.size                : Factor w/ 2 levels "b","n": 2 1 1 2 1 1 1 1 2 1 ...
##  $ gill.color               : Factor w/ 12 levels "b","e","g","h",..: 5 5 6 6 5 6 3 6 8 3 ...
##  $ stalk.shape              : Factor w/ 2 levels "e","t": 1 1 1 1 2 1 1 1 1 1 ...
##  $ stalk.root               : Factor w/ 5 levels "?","b","c","e",..: 4 3 3 4 4 3 3 3 4 3 ...
##  $ stalk.surface.above.ring: Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...
##  $ stalk.surface.below.ring: Factor w/ 4 levels "f","k","s","y": 3 3 3 3 3 3 3 3 3 3 ...
##  $ stalk.color.above.ring  : Factor w/ 9 levels "b","c","e","g",..: 8 8 8 8 8 8 8 8 8 8 ...
##  $ stalk.color.below.ring  : Factor w/ 9 levels "b","c","e","g",..: 8 8 8 8 8 8 8 8 8 8 ...
##  $ veil.type                : Factor w/ 1 level "p": 1 1 1 1 1 1 1 1 1 1 ...
##  $ veil.color               : Factor w/ 4 levels "n","o","w","y": 3 3 3 3 3 3 3 3 3 3 ...
##  $ ring.number              : Factor w/ 3 levels "n","o","t": 2 2 2 2 2 2 2 2 2 2 ...
##  $ ring.type                : Factor w/ 5 levels "e","f","l","n",..: 5 5 5 5 1 5 5 5 5 5 ...
##  $ spore.print.color        : Factor w/ 9 levels "b","h","k","n",..: 3 4 4 3 4 3 3 4 3 3 ...
##  $ population               : Factor w/ 6 levels "a","c","n","s",..: 4 3 3 4 1 3 3 4 5 4 ...
##  $ habitat                  : Factor w/ 7 levels "d","g","l","m",..: 6 2 4 6 2 2 4 4 2 4 ...
```

We now have a data frame with 8124 observations and 23 variables with factor levels ranging from 1 to 12. This is somewhat problematic as 1 level is redundant and 12 levels is too many.

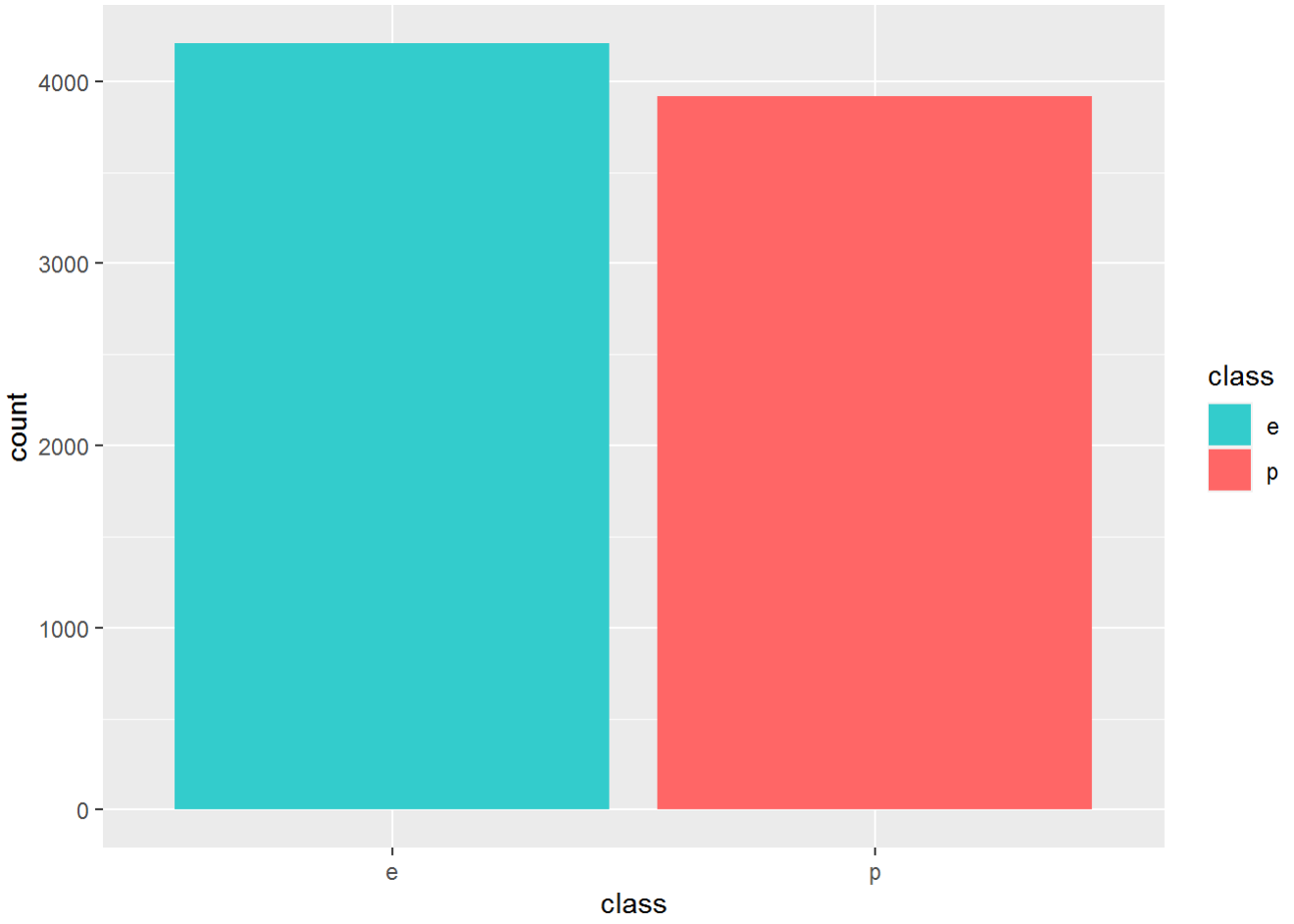
Regression analysis uses numerical data to predict an outcome. When categorical variables are included in the regression analysis, it is converted into numerical representations. This is easy to see when the variable is binarily categorized. One category will be represented by 0 and the other with 1. However, when more than 2 categories are present, we have to perform additional steps for numerical conversions via ‘dummy coding’.

The process of ‘dummy coding’ is basically to split the variable with ‘n’ categories (we use levels and categories interchangeably depending on the context) into ‘n-1’ variables; each now can be represented with either 0 or 1 depending if it occurs. For example, a variable, A, with 3 categories: 1,2,3, will be ‘dummy coded’ into 2 variables A2 and A3. If for variable A, 2 occurs, then the value of A2 will be 1 and A3 will be 0. If 3 occurs, then the value of A2 will be 0 and A3 will be 1. If 1 occurs, the value of A2 and A3 will both be 0. In R, this ‘dummy coding’ is performed automatically when training categorical variables on regression algorithms. However, large numbers of categories within categorical variables should be simplified for calculations. It is also worth noting that R creates the ‘dummy codes’ based on the first level of the variable by default. Hence, variable A is coded into A2 and A3 instead of A1 and A2 or other combinations.

Categories can be statistically insignificant, especially in datasets with many variables. For example, they could have very low relative occurences or very high relative occurences with no effect on the prediction. In both cases, the regression analysis will likely be redundant because the categories either dominate or disappear within the analysis. For example, provided that no one category is unique to the prediction; if A:1 occurs 90% of the time and A:2 and A:3 occurs 5% of the time each, A:1 dominates the data and the categorization on A is redundant towards the prediction analysis; it is all the same to include or exclude A and its categories. Thus, we should inspect the variables carefully.

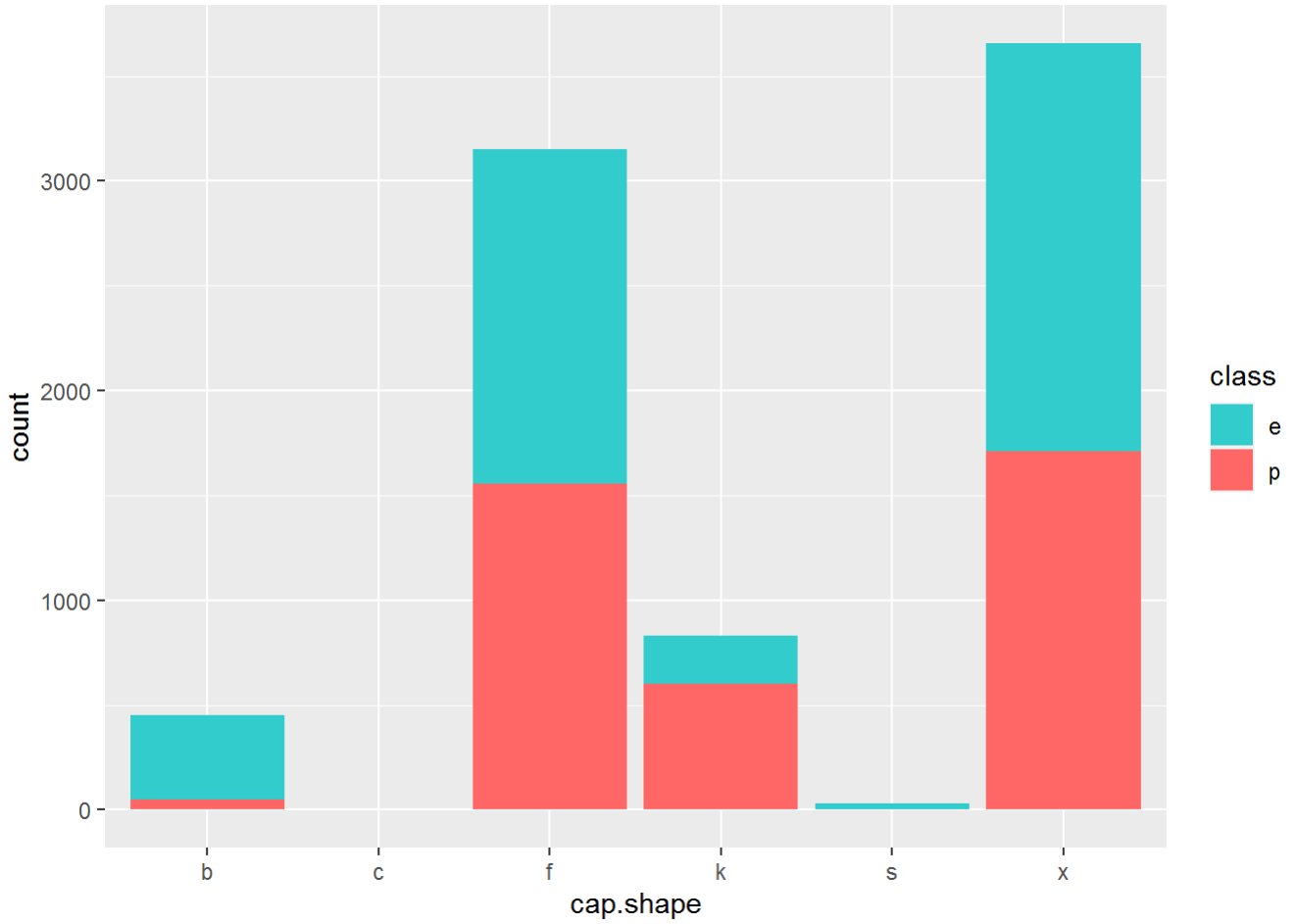
First, we will visualise the proportions of ‘edible’ and ‘poisonous’ in our datasets:

```
dat %>%
  ggplot(aes(class, fill = class)) +
  geom_bar() +
  scale_fill_manual(values = c("#33CCCC","#FF6666"))
```



We will also visualise the occurrences of each category in each variable and their proportions of ‘edible’ and ‘poisonous’:

```
dat %>%
  ggplot(aes(cap.shape, fill = class)) +
  geom_bar() +
  scale_fill_manual(values = c("#33CCCC","#FF6666"))
```

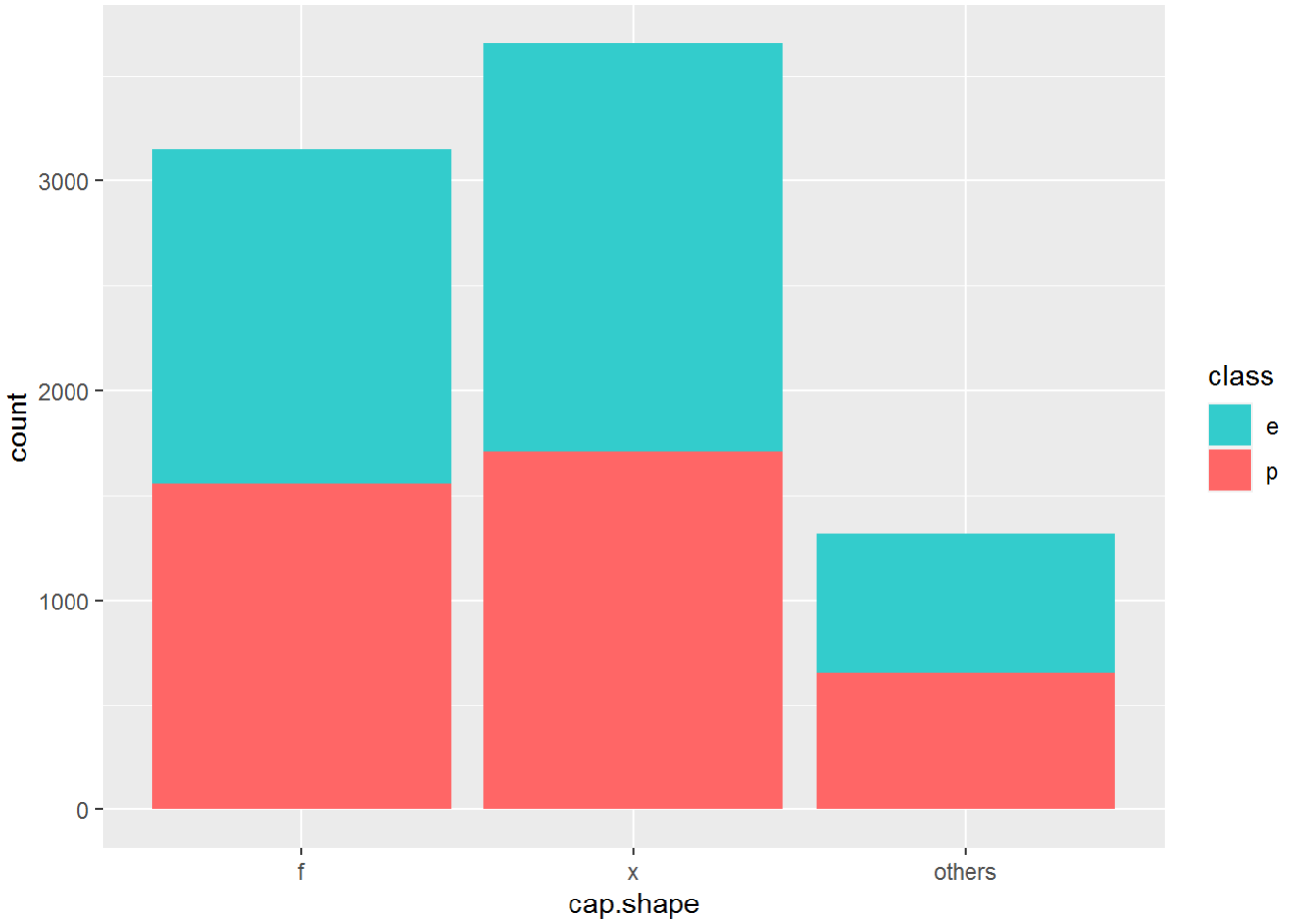


By exploring the first predictor variable, we see that some categories do not occur very frequently.

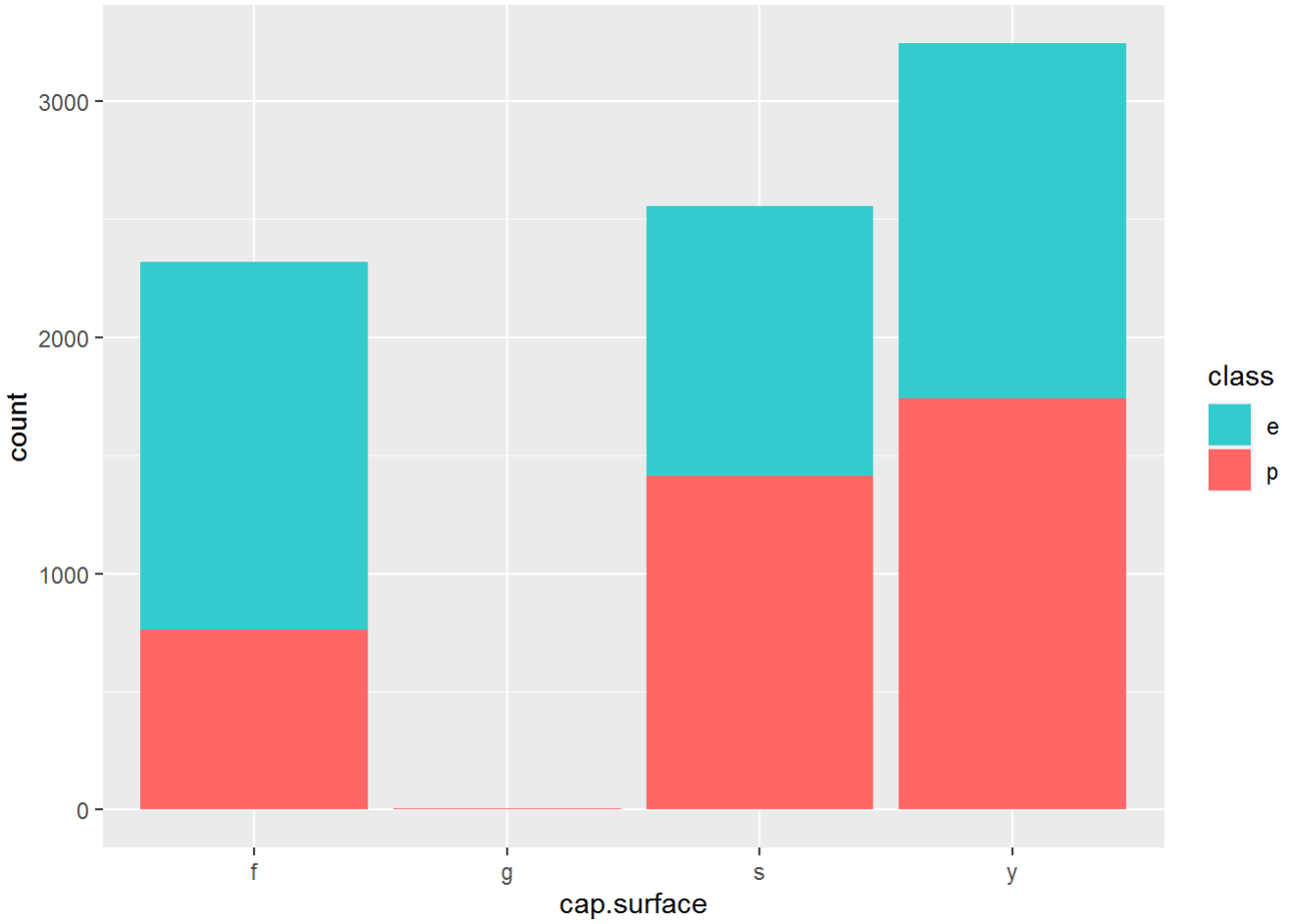
We will make a calculated decision to merge low occuring categories together. It will not be ideal to exclude them altogether as observations were made on many other variables. We have to be careful in making this decision because merging categories reduces the total number of unique combinations in our dataset. Furthermore, if some of the low occuring categories are specifically unique to either ‘edible’ or ‘poisonous’ (very strong predicting power), merging them with other low occuring categories that don’t have as strong of a predicting power could mask its potential to contribute to the regression analysis.

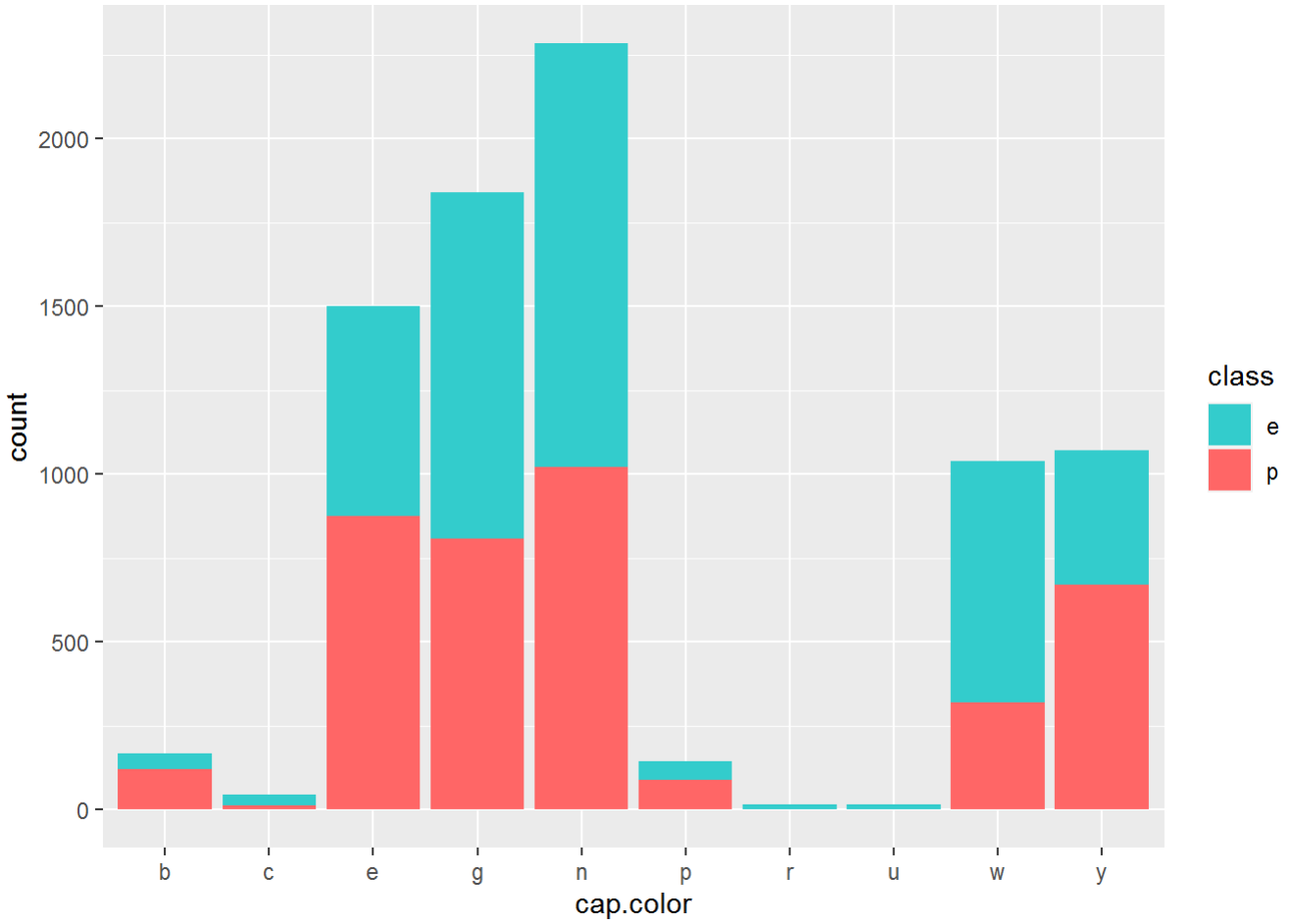
To avoid this as much as possible, we do not merge low occuring categories in variables with less than 4 categories and we do not merge low occuring categories with those that are not low occuring. We will define low occurence by relativity within each variable. We must also pay attention to the ‘class’ proportion ratio of each category; it is safer to merge categories which have the same or very similar proportion ratio. Finally, We will exclude variables we suspect have reporting errors. We will first do this for the ‘cap.shape’ variable above:

```
dat$cap.shape<- dat$cap.shape %>%
   fct_collapse(b= c("b","c","k","s")) %>%
   factor(labels = c("others","f","x")) %>%
   factor(levels = c("f","x","others"))

dat %>%
  ggplot(aes(cap.shape, fill = class)) +
  geom_bar() +
  scale_fill_manual(values = c("#33CCCC","#FF6666"))
```

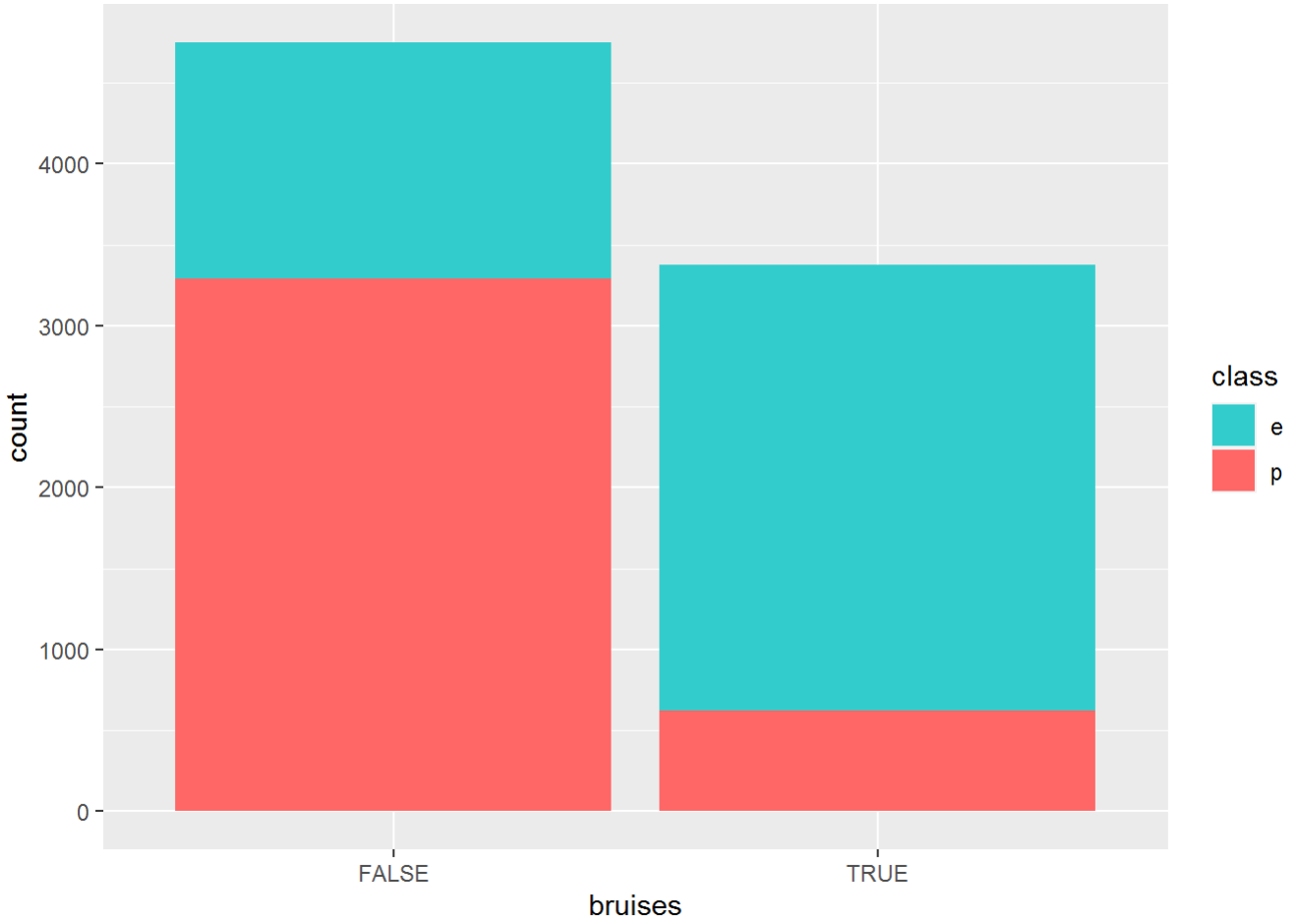


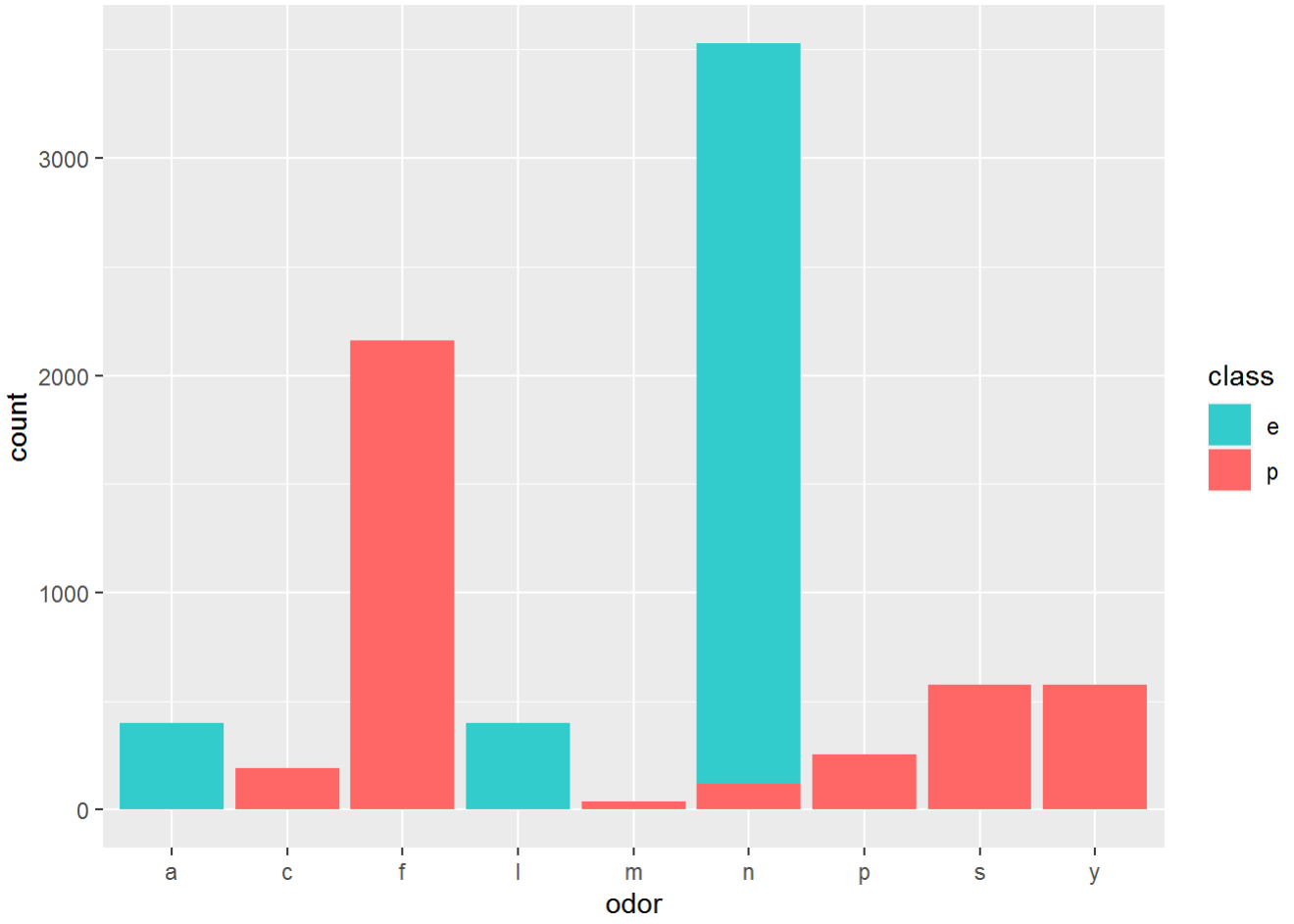
We will continue to analyse each variable and simplify our data as required:


3000
2000
1000
0
count
f
g
s
y
cap.surface
class
e
p

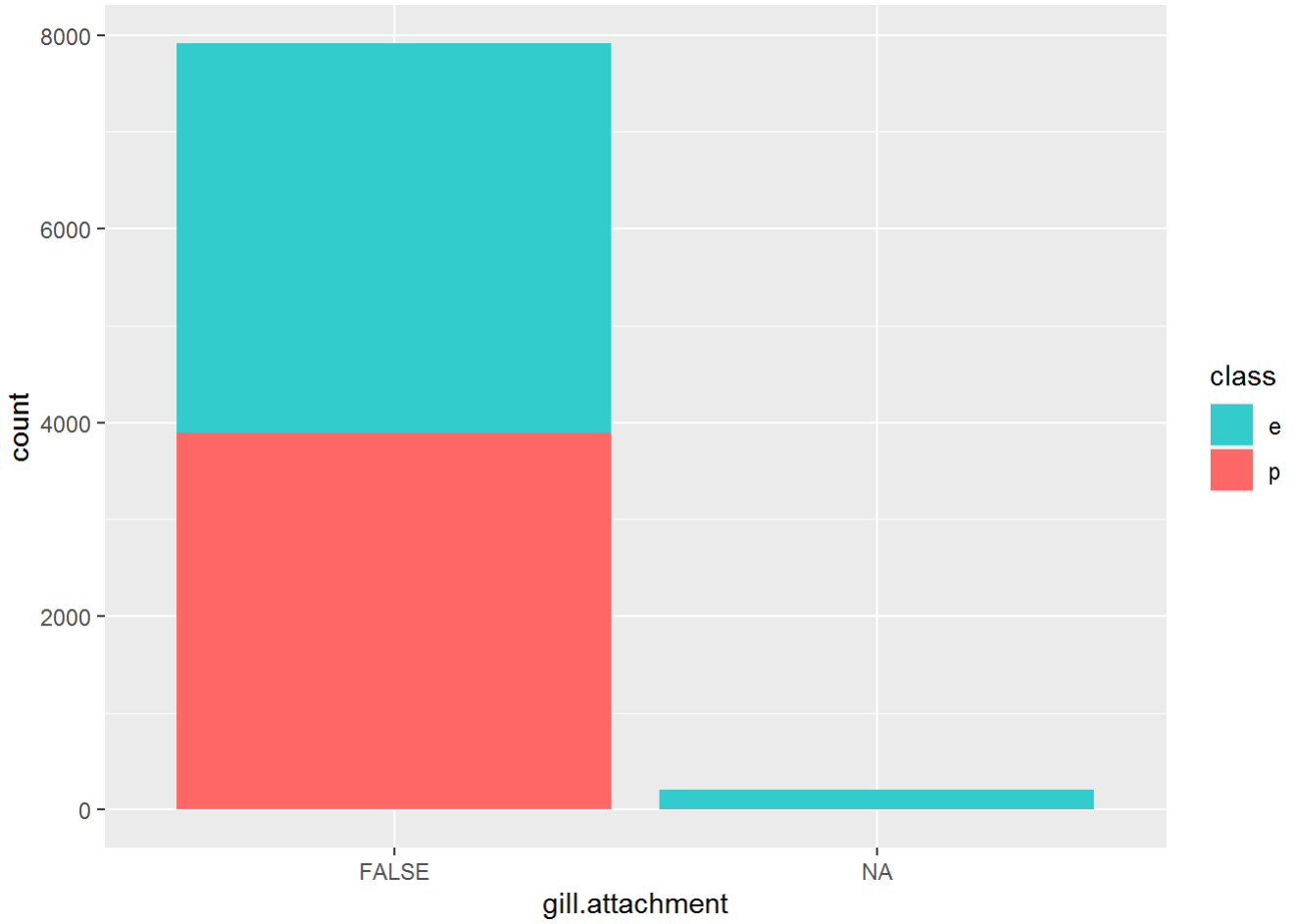

2000
1500
1000
500
0
count
b
c
e
g
n
p
r
u
w
y
cap.color
class
e
p

```
dat$cap.color<- dat$cap.color %>%
  fct_collapse(b= c("b","c","p","r","u")) %>%
  factor(labels = c("others","e","g","n","w","y")) %>%
  factor(levels = c("e","g","n","w","y","others"))
```

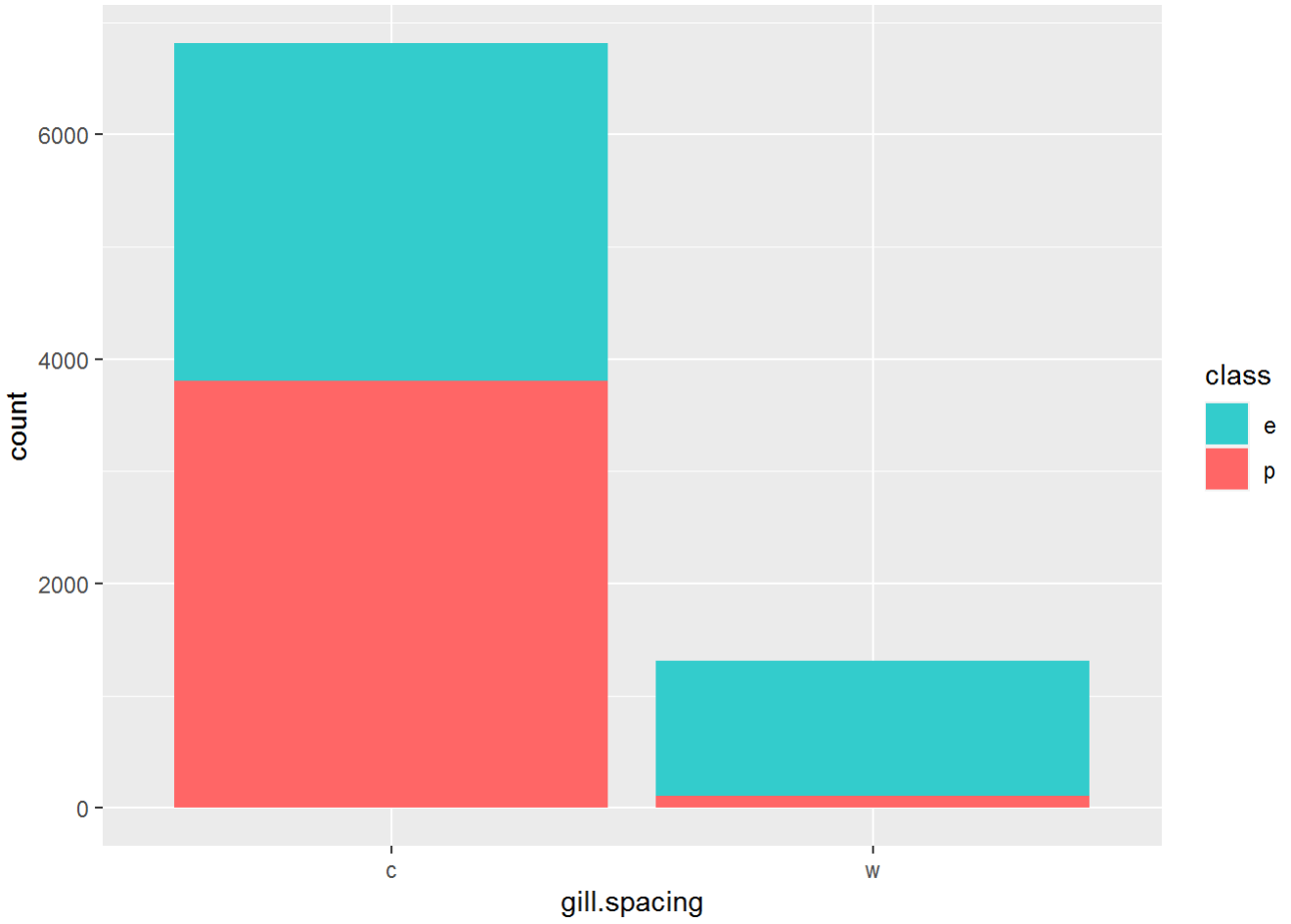


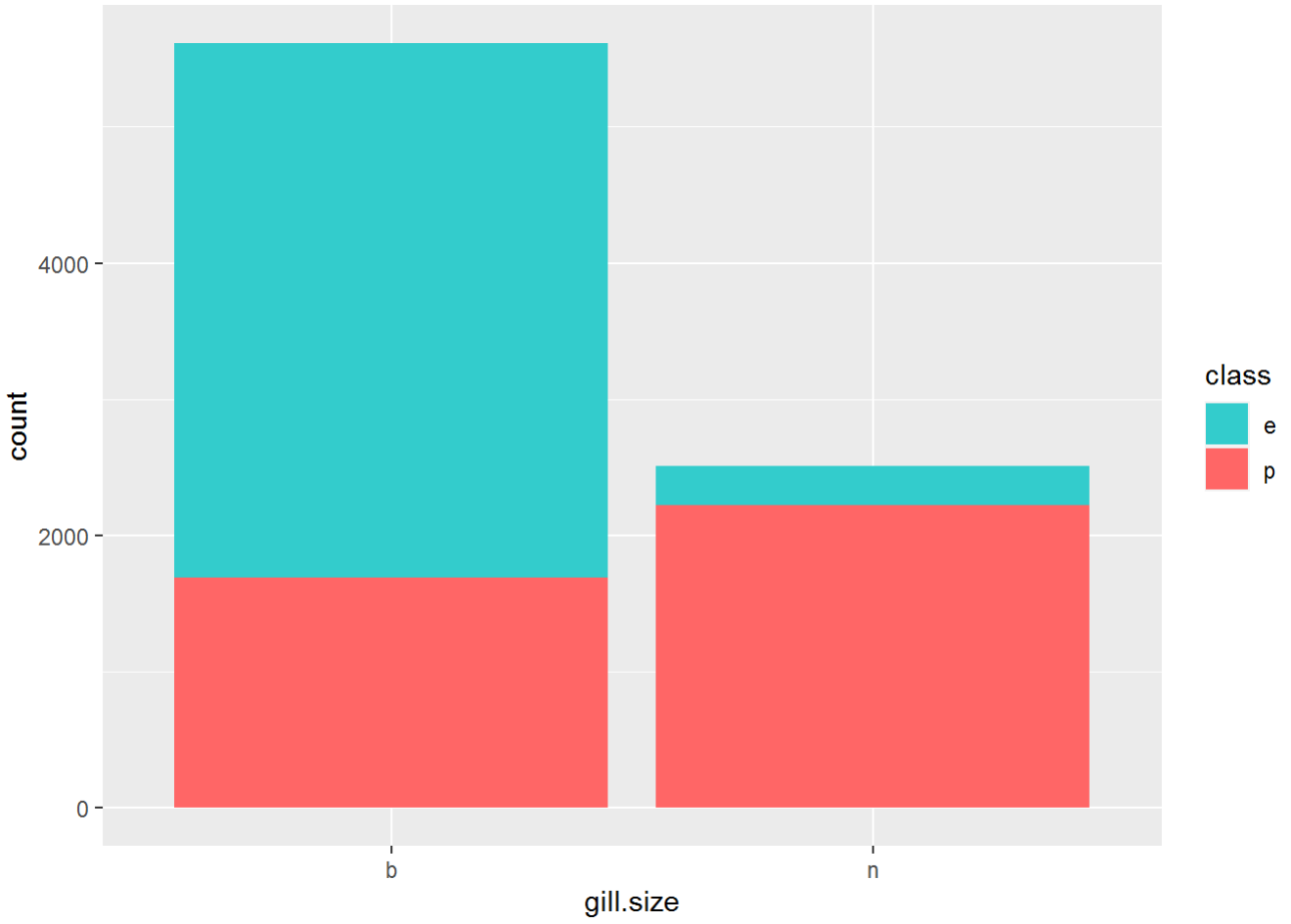


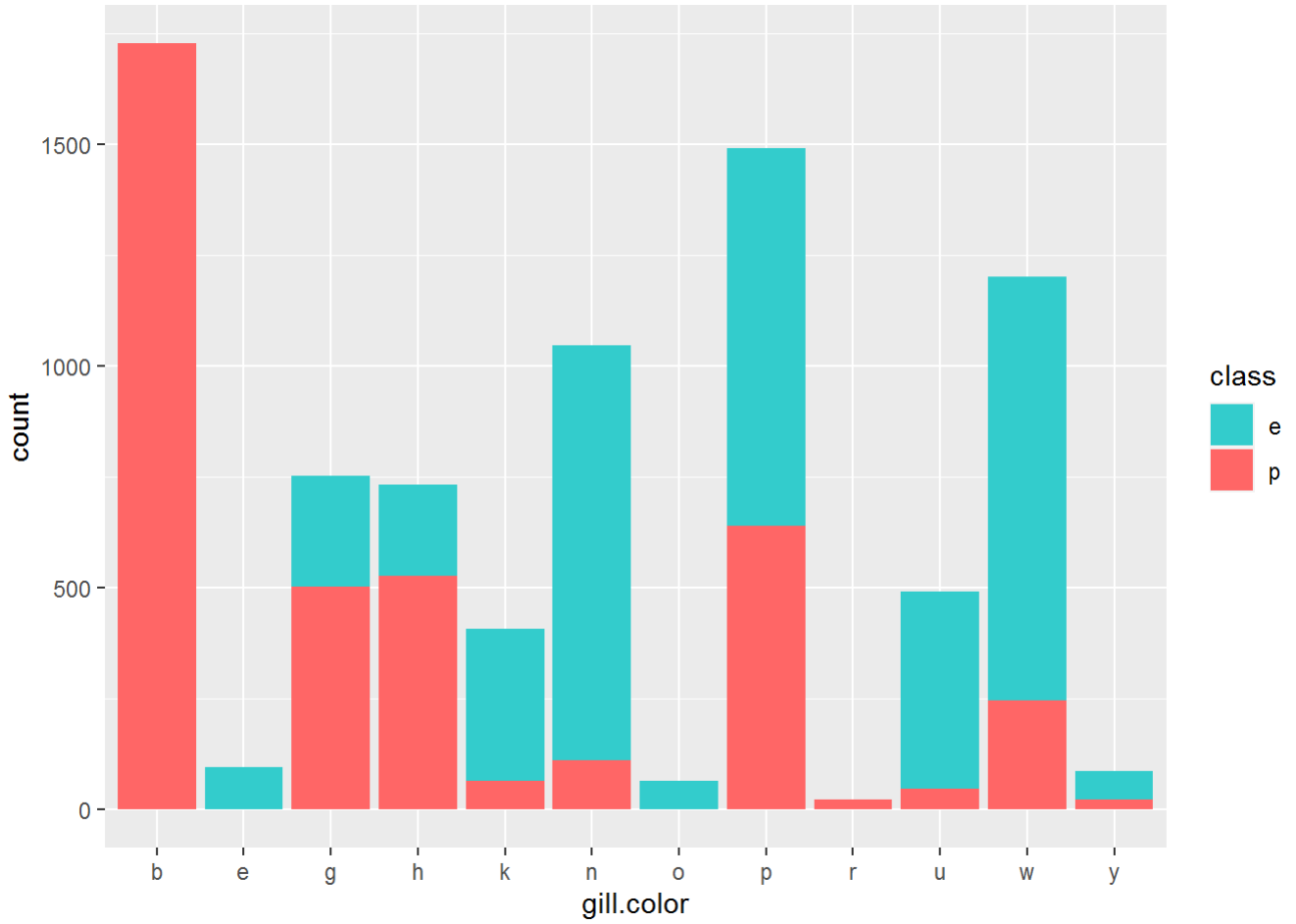

```
dat$odor<- dat$odor %>%
  fct_collapse(a= c("a","c","l","m","p","s","y")) %>%
  factor(labels = c("others","f","n")) %>%
  factor(levels = c("f","n","others"))
```



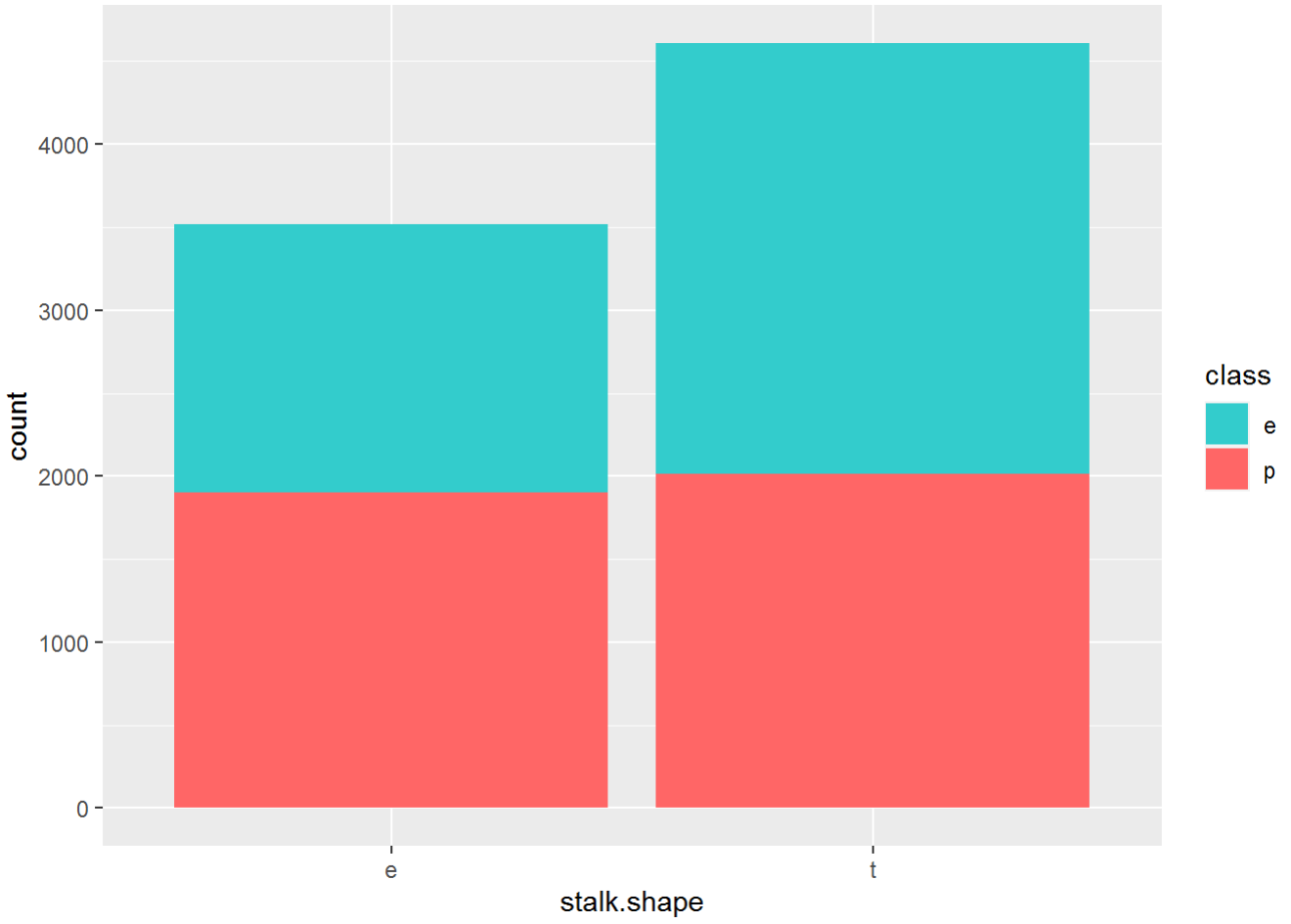

```
dat<- dat %>% select(-"gill.attachment")
```

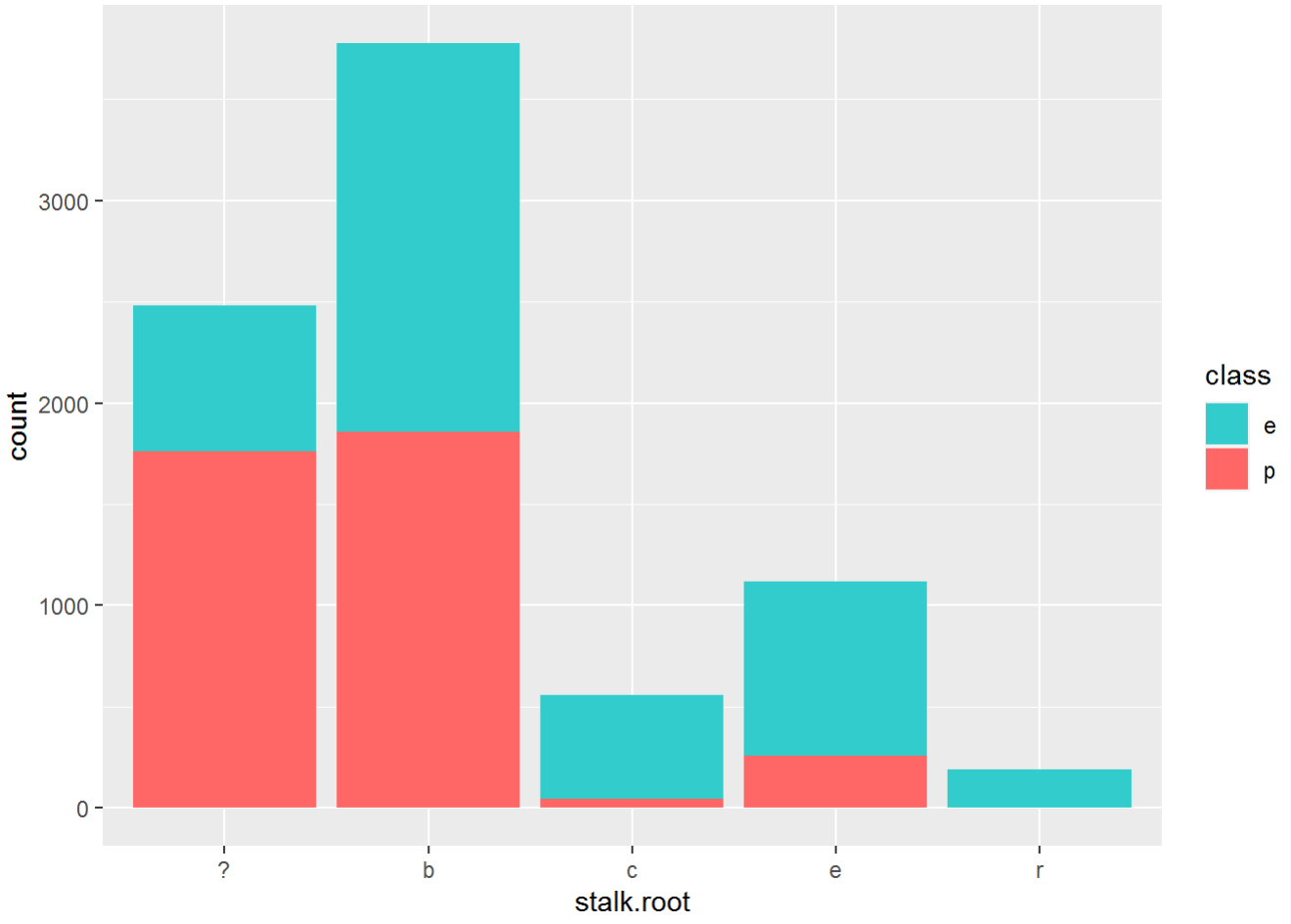

6000
4000
2000
0
count
c
w
gill.spacing
class
e
p


4000
2000
0
count
b
n
gill.size
class
e
p

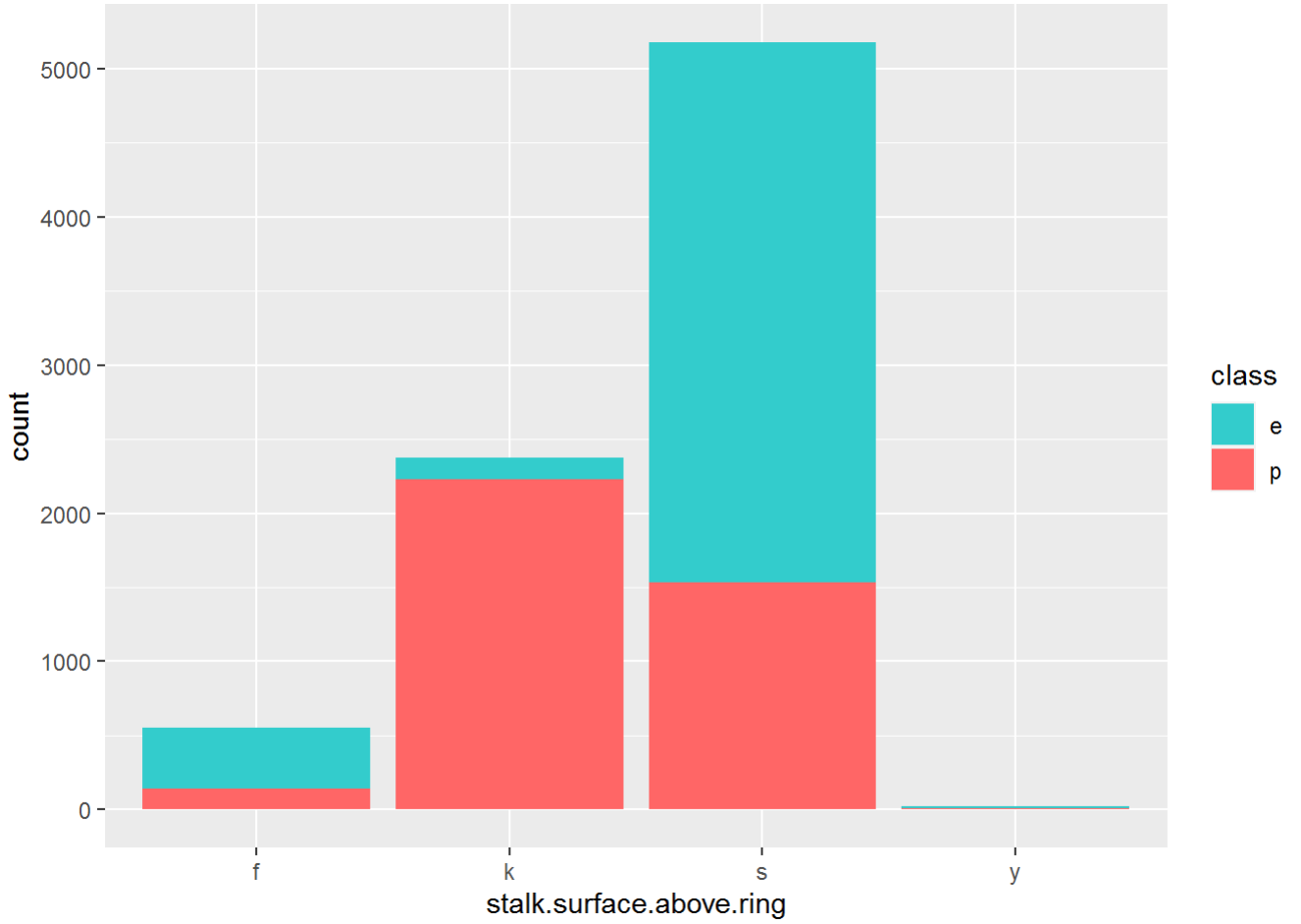



```
dat$gill.color<- dat$gill.color %>%
  fct_collapse(e= c("e","k","o","r","u","y")) %>%
  factor(labels = c("b","others","g","h","n","p","w")) %>%
  factor(levels = c("b","g","h","n","p","w","others"))
```

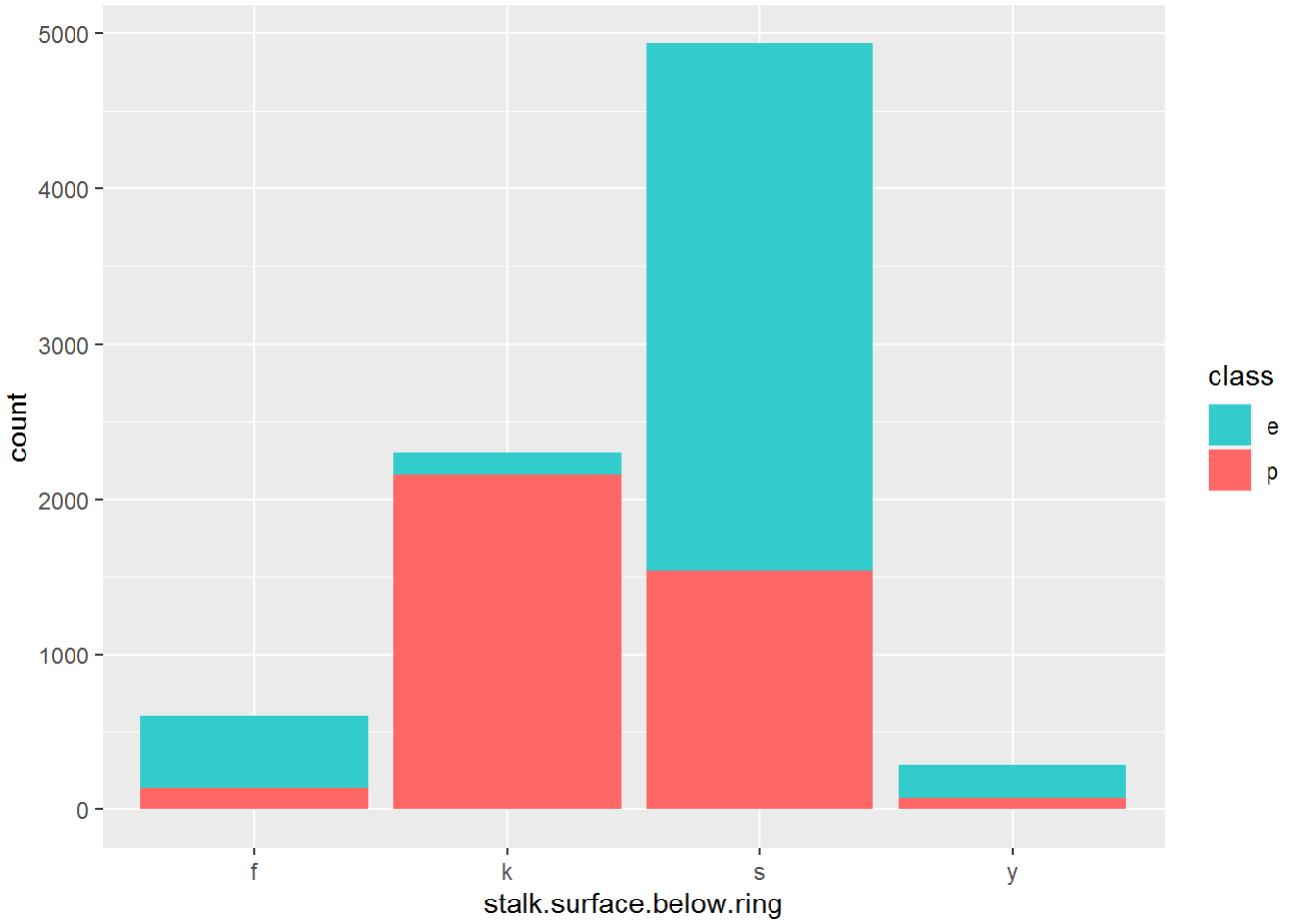

4000
3000
2000
1000
0
count
e
t
stalk.shape
class
e
p


3000
2000
1000
0
count
?
b
c
e
r
stalk.root
class
e
p

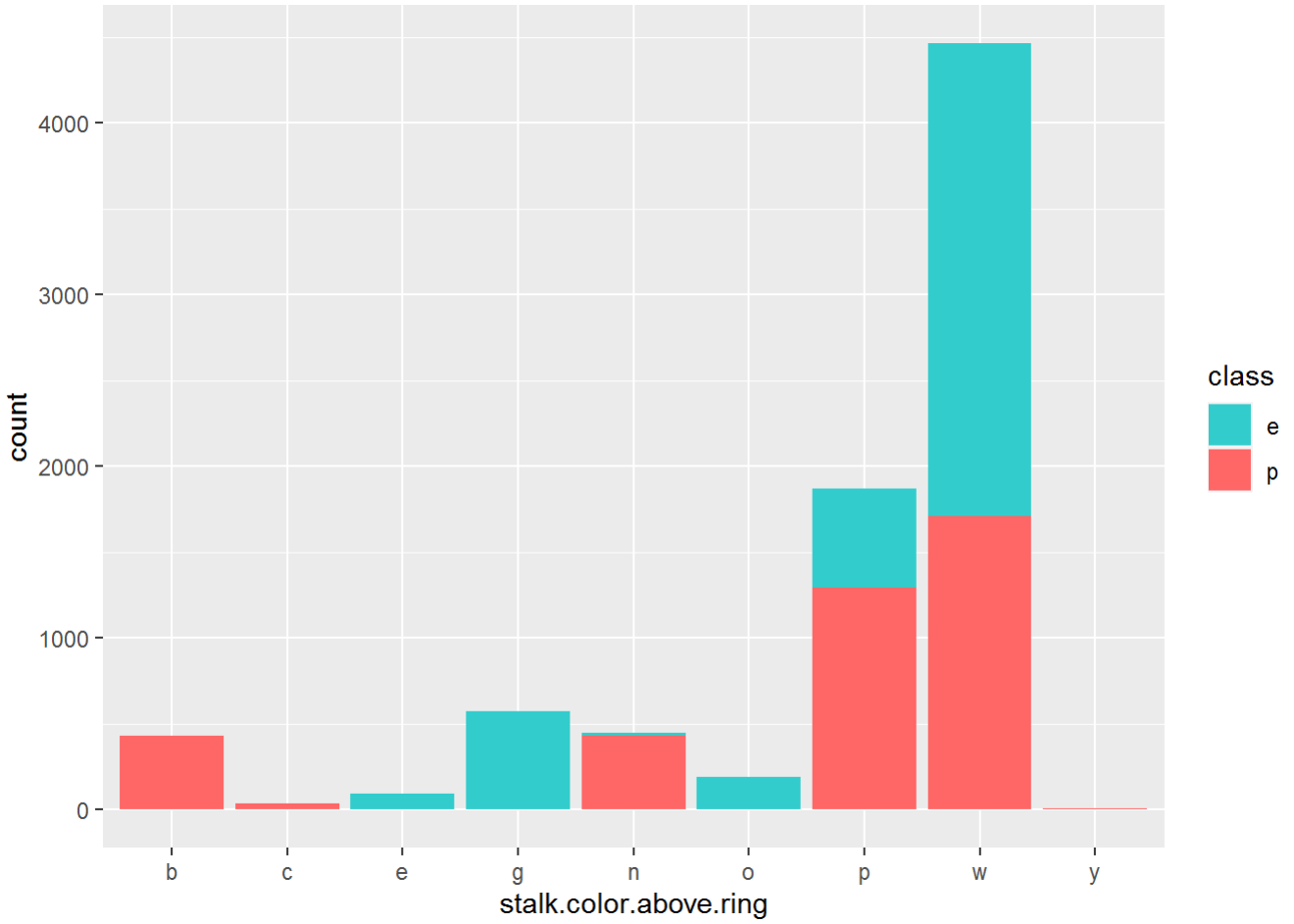
```
levels(dat$stalk.root)<- c("NA","b","c","e","r")
dat$stalk.root<- dat$stalk.root %>%
  fct_collapse(c= c("c","r")) %>%
  factor(labels = c("NA","b","others","e")) %>%
  factor(levels = c("b","e","NA","others"))
```



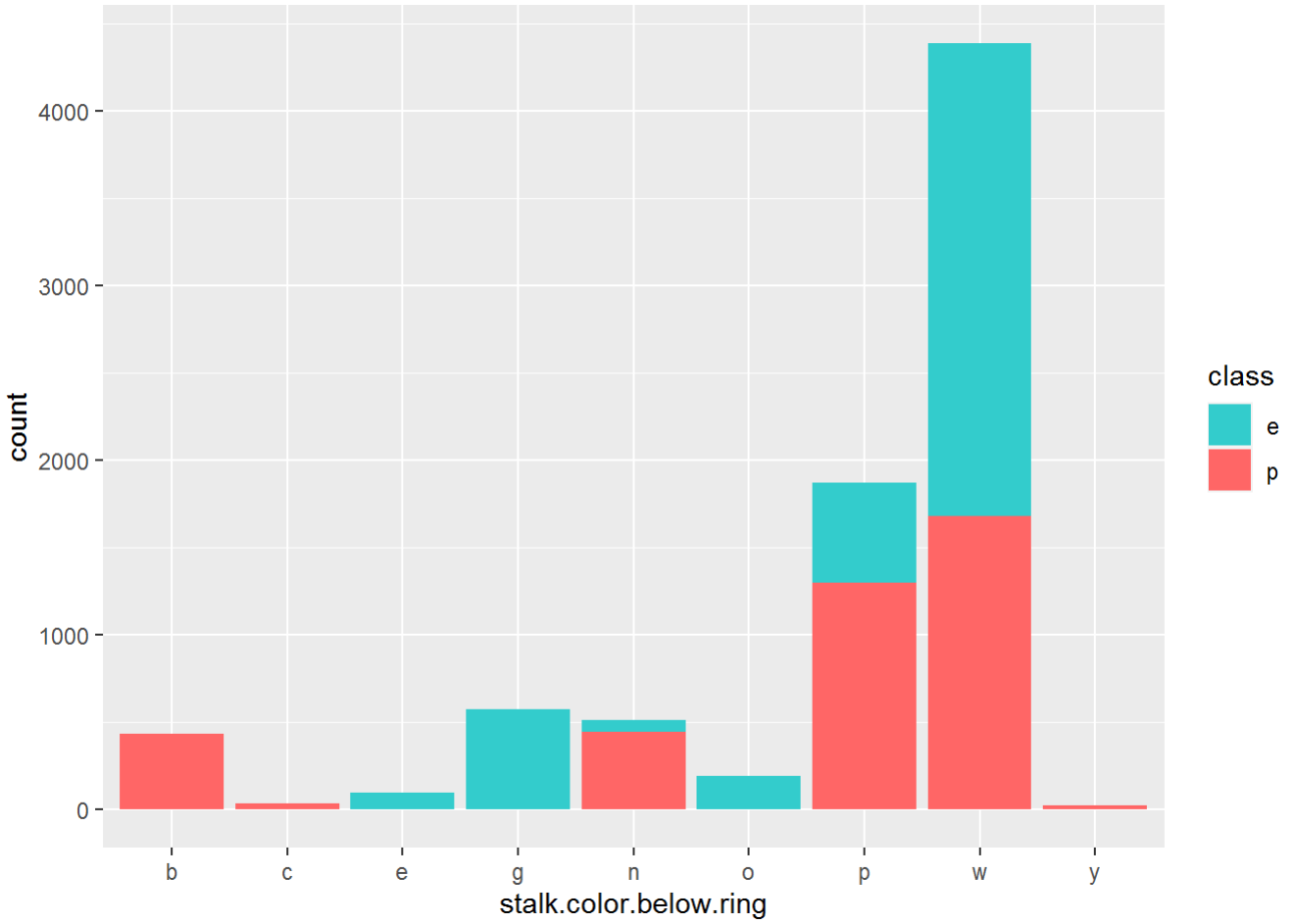

```
dat$stalk.surface.above.ring<- dat$stalk.surface.above.ring %>%
  fct_collapse(f= c("f","y")) %>%
  factor(labels = c("others","k","s")) %>%
  factor(levels = c("k","s","others"))
```



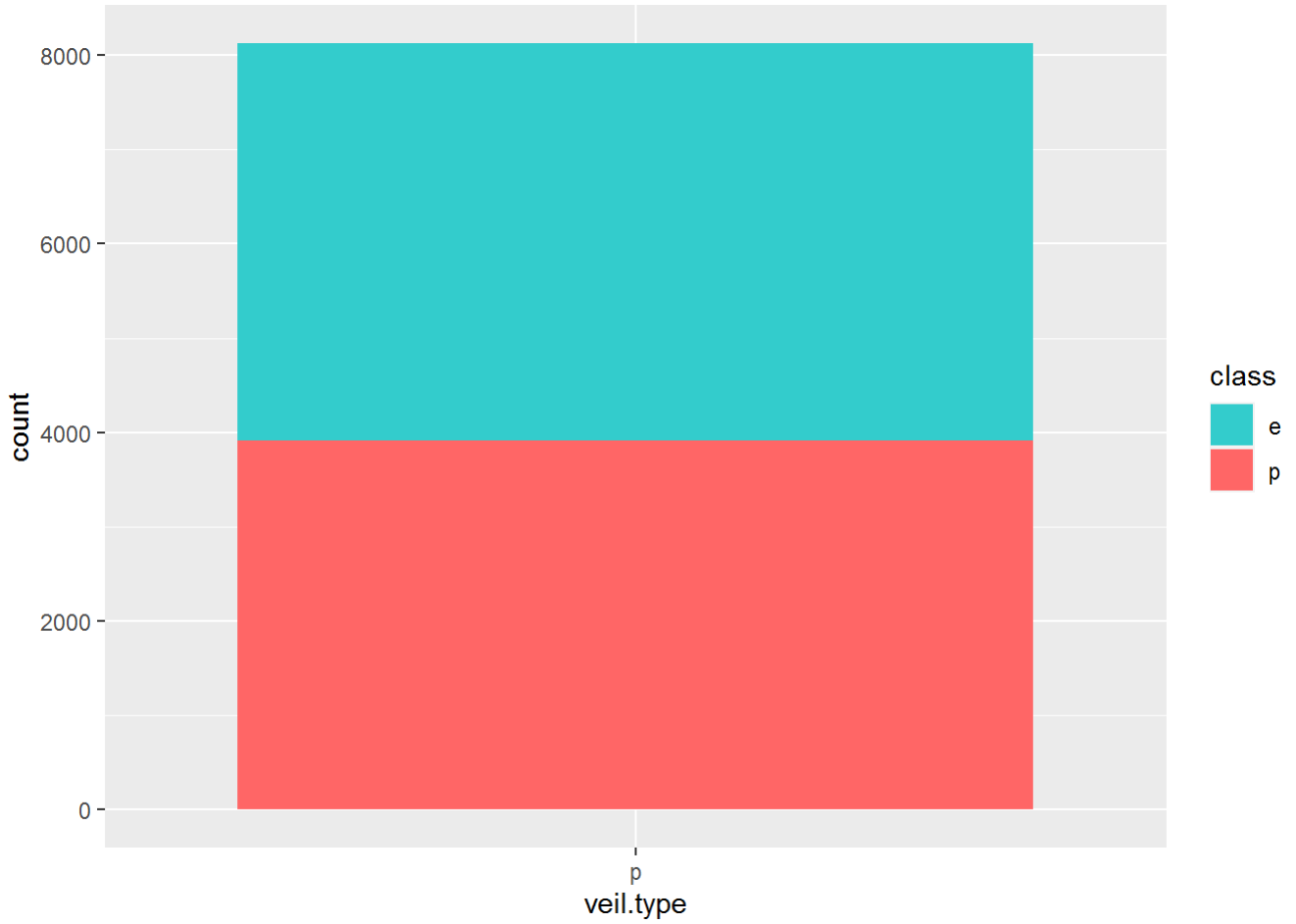

```
dat$stalk.surface.below.ring<- dat$stalk.surface.below.ring %>%
  fct_collapse(f= c("f","y")) %>%
  factor(labels = c("others","k","s")) %>%
  factor(levels = c("k","s","others"))
```



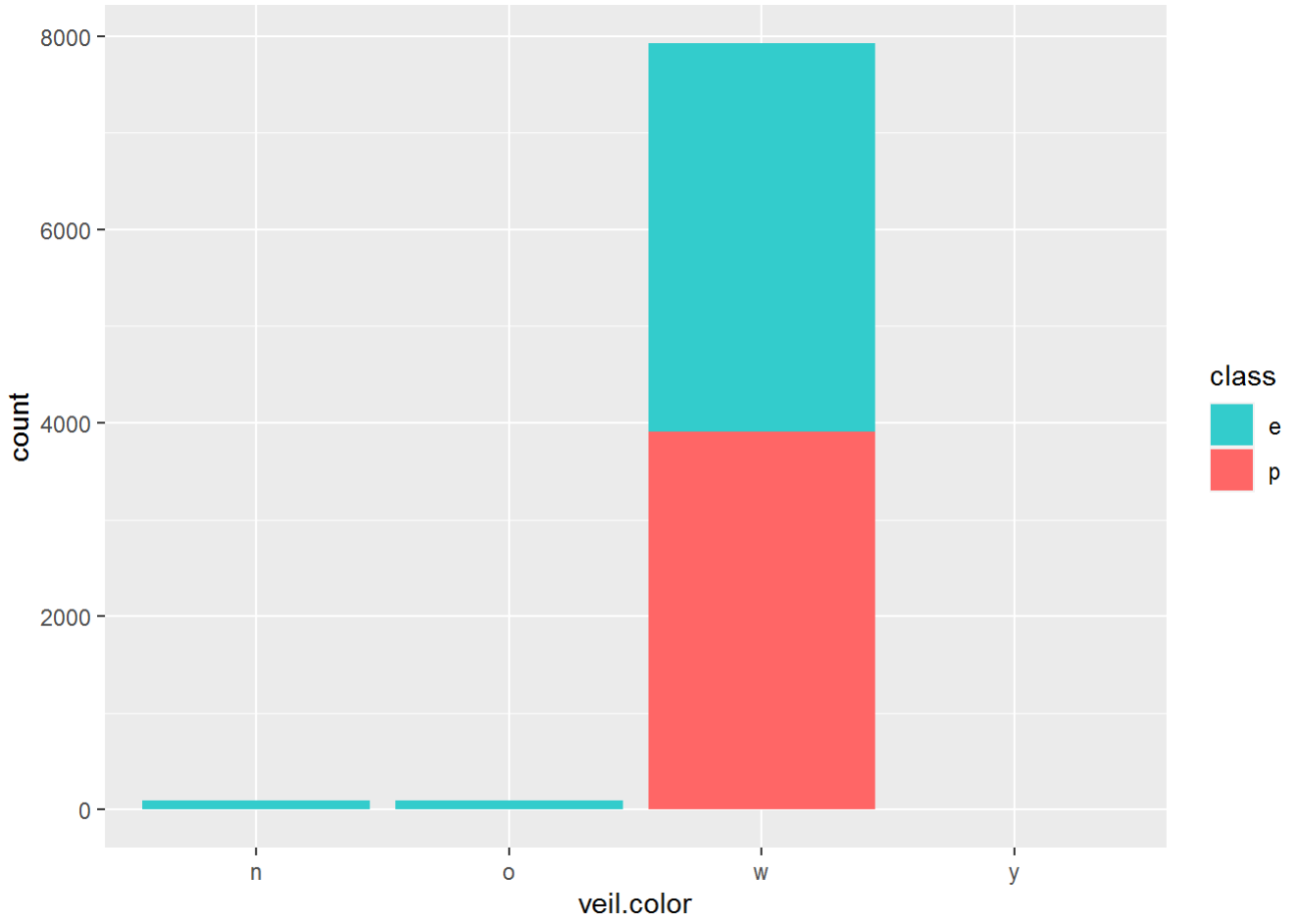

```
dat$stalk.color.above.ring<- dat$stalk.color.above.ring %>%
  fct_collapse(b= c("b","c","e","g","n","o","y")) %>%
  factor(labels = c("others","p","w")) %>%
  factor(levels = c("p","w","others"))
```



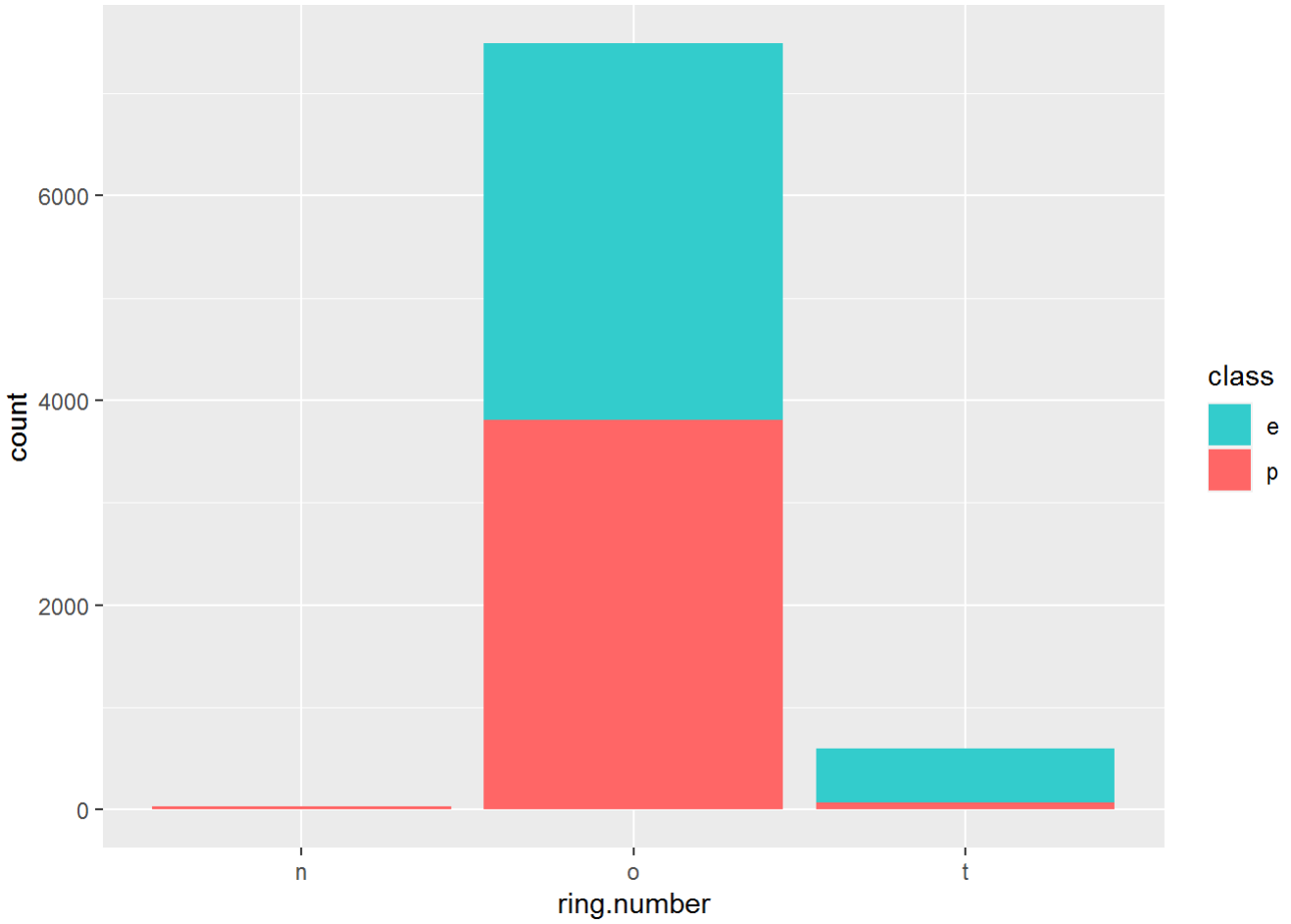

```
dat$stalk.color.below.ring<- dat$stalk.color.below.ring %>%
  fct_collapse(b= c("b","c","e","g","n","o","y")) %>%
  factor(labels = c("others","p","w")) %>%
  factor(levels = c("p","w","others"))
```

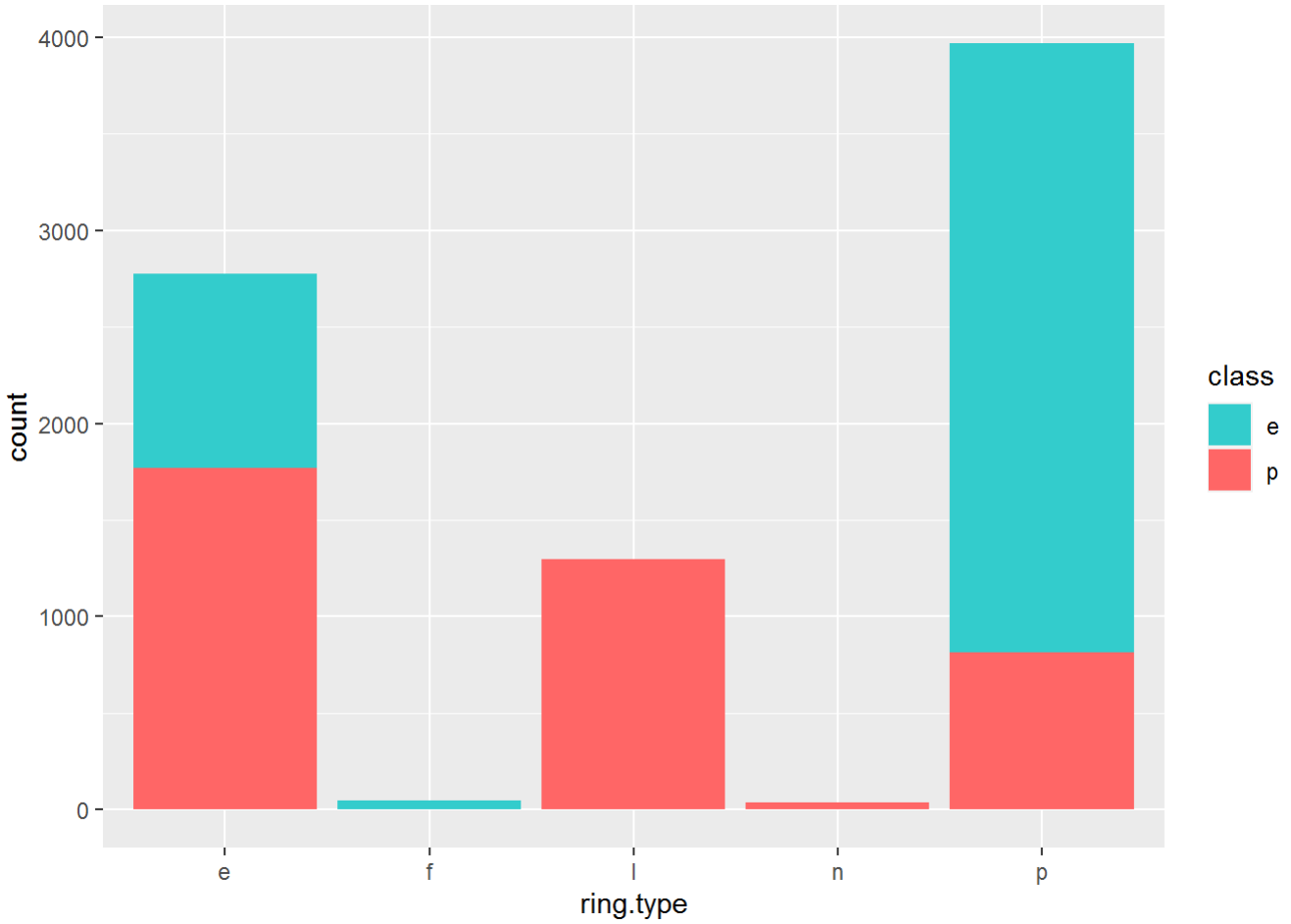



```
dat<- dat %>% select(-"veil.type")
```

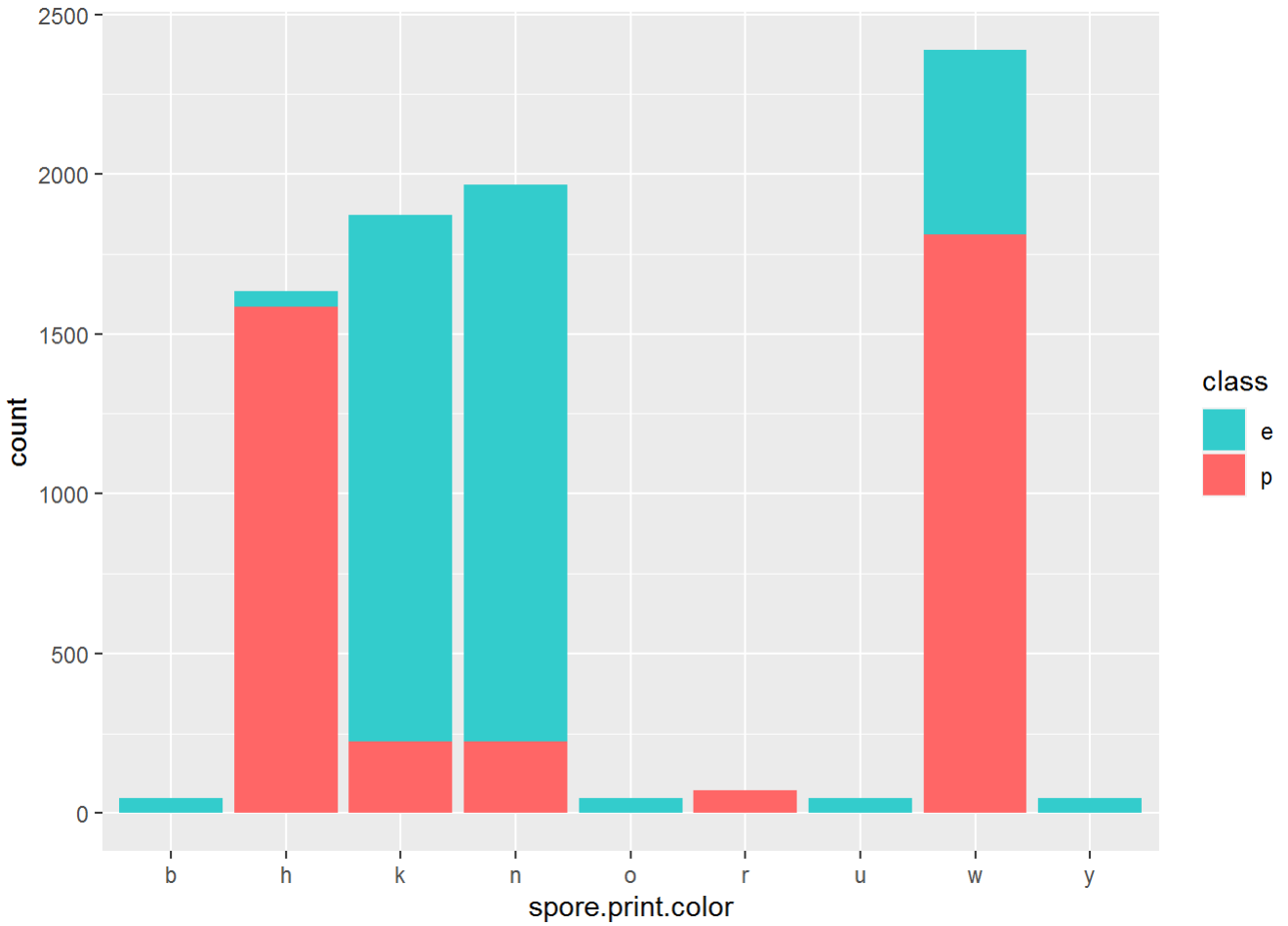



```
dat$veil.color<- dat$veil.color %>%
  fct_collapse(n= c("n","o","y")) %>%
  factor(labels = c("others","w")) %>%
  factor(levels = c("w","others"))
```

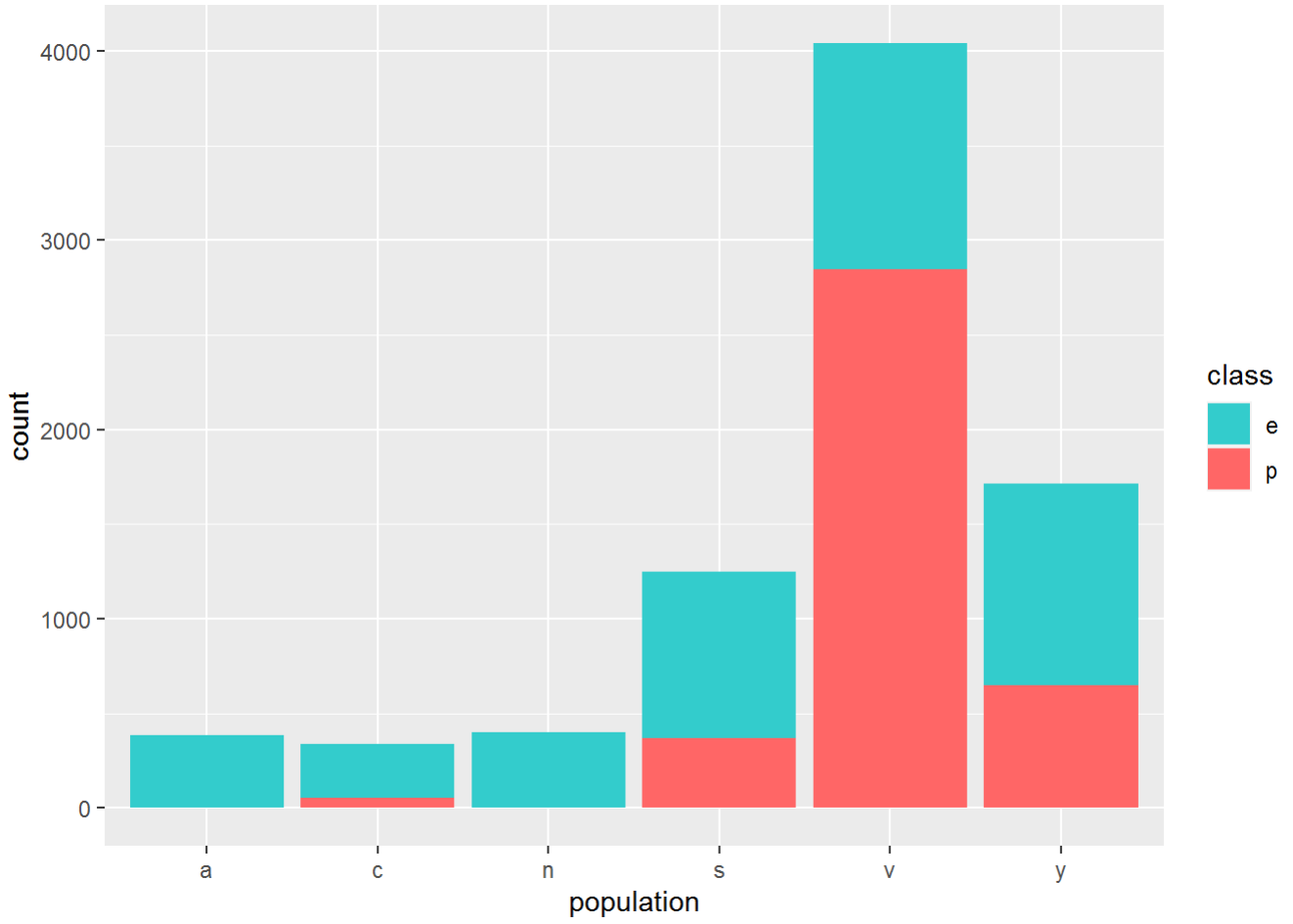

6000
4000
2000
0
count
n
o
t
ring.number
class
e
p


4000
3000
2000
1000
0
count
e
f
l
n
p
ring.type
class
e
p

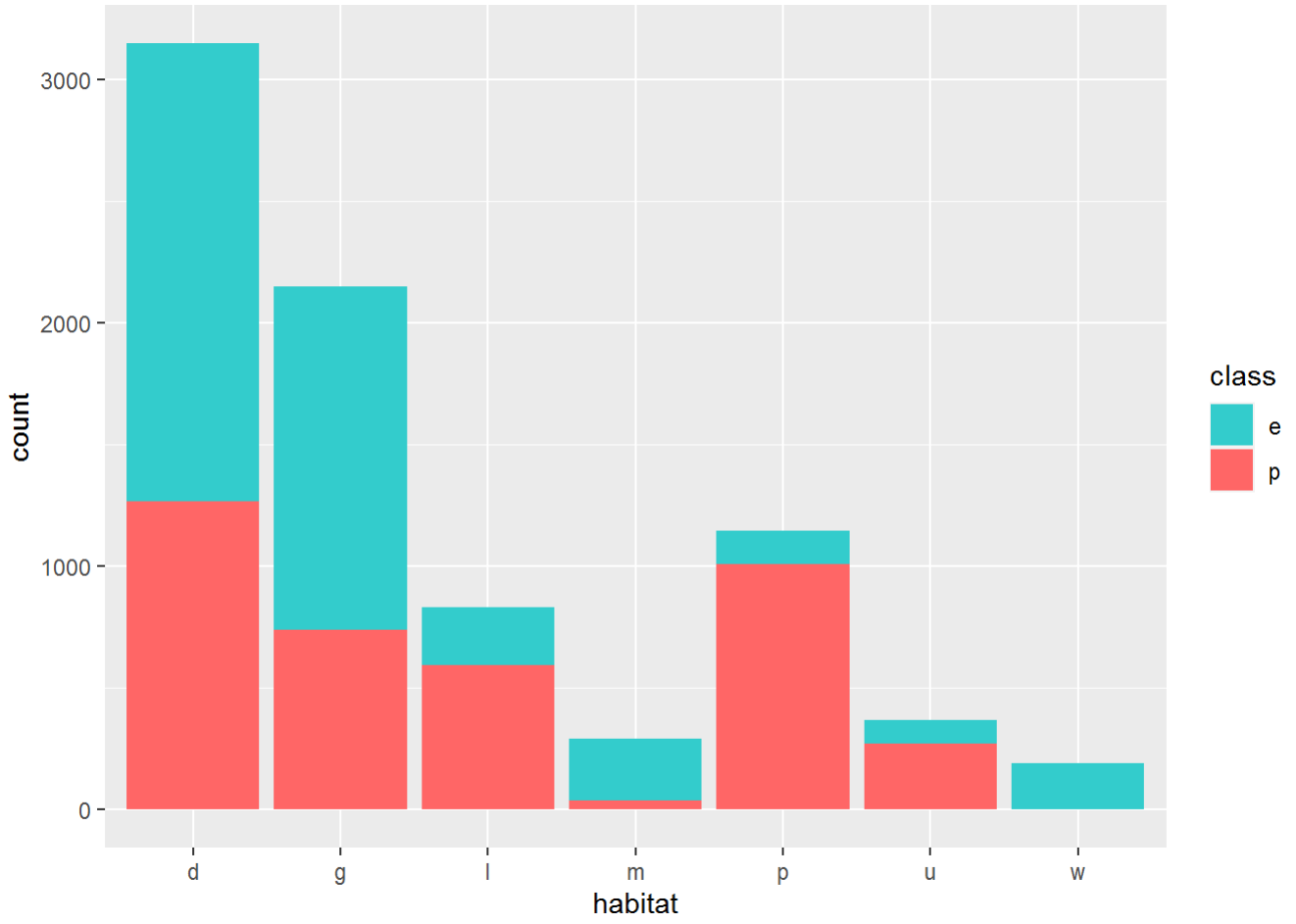
```
dat$ring.type<- dat$ring.type %>%
  fct_collapse(f= c("f","n")) %>%
  factor(labels = c("e","others","l","p")) %>%
  factor(levels = c("e","l","p","others"))
```




```
dat$spore.print.color<- dat$spore.print.color %>%
  fct_collapse(b= c("b","o","r","u","y")) %>%
  factor(labels = c("others","h","k","n","w")) %>%
  factor(levels = c("h","k","n","w","others"))
```




```
dat$population<- dat$population %>%
  fct_collapse(a= c("a","c","n")) %>%
  factor(labels = c("others","s","v","y")) %>%
  factor(levels = c("s","v","y","others"))
```




```
dat$habitat<- dat$habitat %>%
  fct_collapse(l= c("l","m","u","w")) %>%
  factor(labels = c("d","g","others","p")) %>%
  factor(levels = c("d","g","p","others"))
```

```
str(dat)
```

```
## 'data.frame':    8124 obs. of  21 variables:
##  $ class                   : Factor w/ 2 levels "e","p": 2 1 1 2 1 1 1 1 2 1 ...
##  $ cap.shape               : Factor w/ 3 levels "f","x","others": 2 2 3 2 2 2 3 3 2 3 ...
##  $ cap.surface             : Factor w/ 4 levels "f","g","s","y": 3 3 3 4 3 4 3 4 4 3 ...
##  $ cap.color               : Factor w/ 6 levels "e","g","n","w",..: 3 5 4 4 2 5 4 4 4 5 ...
##  $ bruises                 : Factor w/ 2 levels "FALSE","TRUE": 2 2 2 2 1 2 2 2 2 2 ...
##  $ odor                    : Factor w/ 3 levels "f","n","others": 3 3 3 3 2 3 3 3 3 3 ...
##  $ gill.spacing            : Factor w/ 2 levels "c","w": 1 1 1 1 2 1 1 1 1 1 ...
##  $ gill.size               : Factor w/ 2 levels "b","n": 2 1 1 2 1 1 1 1 2 1 ...
##  $ gill.color              : Factor w/ 7 levels "b","g","h","n",..: 7 7 4 4 7 4 2 4 5 2 ...
##  $ stalk.shape             : Factor w/ 2 levels "e","t": 1 1 1 1 2 1 1 1 1 1 ...
##  $ stalk.root              : Factor w/ 4 levels "b","e","NA","others": 2 4 4 2 2 4 4 4 2 4
...
##  $ stalk.surface.above.ring: Factor w/ 3 levels "k","s","others": 2 2 2 2 2 2 2 2 2 2 ...
##  $ stalk.surface.below.ring: Factor w/ 3 levels "k","s","others": 2 2 2 2 2 2 2 2 2 2 ...
##  $ stalk.color.above.ring  : Factor w/ 3 levels "p","w","others": 2 2 2 2 2 2 2 2 2 2 ...
##  $ stalk.color.below.ring  : Factor w/ 3 levels "p","w","others": 2 2 2 2 2 2 2 2 2 2 ...
##  $ veil.color              : Factor w/ 2 levels "w","others": 1 1 1 1 1 1 1 1 1 1 ...
##  $ ring.number             : Factor w/ 3 levels "n","o","t": 2 2 2 2 2 2 2 2 2 2 ...
##  $ ring.type               : Factor w/ 4 levels "e","l","p","others": 3 3 3 3 1 3 3 3 3 3 ...
##  $ spore.print.color       : Factor w/ 5 levels "h","k","n","w",..: 2 3 3 2 3 2 2 3 2 2 ...
##  $ population              : Factor w/ 4 levels "s","v","y","others": 1 4 4 1 4 4 4 1 2 1 ...
##  $ habitat                 : Factor w/ 4 levels "d","g","p","others": 4 2 4 4 2 2 4 4 2 4 ...
```

We finally have a data frame of 8124 observations and 21 variables of factor levels ranging from 2 to 7-which makes more sense and is easier to work with. If we perform regression analysis on this data, we will be working with 20 predictors with ‘class’ being our prediction result (label). This is a 20-dimensional-space data.

To start our classification analysis, We create partitions on our data; a train set for training and a test set for comparison. We choose a 80:20 train-to-test partition to allow the algorithm to train with more data input, allowing for the various combinations of predictors and observations. This will increase the accuracy of predicting unknown future cases reporting various combinations of observations.

```
set.seed(2, sample.kind = "Rounding")
test_index<- createDataPartition(dat$class, times = 1, p = 0.2, list = FALSE)
train_set<- dat[-test_index, ]
test_set<- dat[test_index, ]
```

LOGISTIC REGRESSION MODEL

The first model we will test is the ‘logistic regression’ model or ‘logit regression’. The ‘logit regression’ model is used widely in statistics to model the probability of a binary event occuring. Hence, it is an obvious choice for this analysis as we want to predict the ‘class’ of the mushrooms: ‘edible’ or ‘poisonous’:

```
set.seed(3, sample.kind = "Rounding")
fit_glm<- train(class ~ ., data = train_set, method = "glm", family = "binomial")
pred_glm<- predict(fit_glm, test_set)
cm_glm<- confusionMatrix(pred_glm,test_set$class)
```

We first examine the summary of the ‘Confusion Matrix’ on our ‘glm’ model.

```
cm_glm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   e   p
##          e 842   0
##          p   0 784
##
##                Accuracy : 1
##                  95% CI : (0.9977, 1)
##     No Information Rate : 0.5178
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5178
##          Detection Rate : 0.5178
##    Detection Prevalence : 0.5178
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : e
##
```

We see that the class ‘e’/‘edible’ is our ‘Positive’ class. This is important to know because when measuring ‘sensitivity’, we are measuring how accurate the algorithm will predict the ‘true positives’ and when measuring ‘specificity’, we measure the accuracy of ‘true negatives’.

By this definition; ‘sensitivity’ is the rate of correctly predicting ‘edible’ (the mushrooms which we predict ‘edible’ are actually ‘edible’) and ‘specificity’ is the rate of correctly identifying ‘poisonous’ (the mushrooms which we predict ‘poisonous’ are actually ‘poisonous’). Hence, we see that for this analysis, high sensitivity is more important than high specificity in terms of potential effects of ingestion. Ingesting mushrooms that were predicted ‘edible’ but was actually ‘poisonous’ could be fatal (illness or even death).

Now we tabulate our results:

```
results<- tibble(model = "glm",
                 sensitivity = signif(cm_glm$byClass[1]),
                 specificity = signif(cm_glm$byClass[2]),
                 accuracy = signif(cm_glm$byClass[11]),
                 computation_time = "< 30 seconds")

kable(results, align = 'c') %>%
  kable_styling(full_width = FALSE)
```

model	sensitivity	specificity	accuracy	computation_time
glm	1	1	1	< 30 seconds

K-NEAREST NEIGHBOR

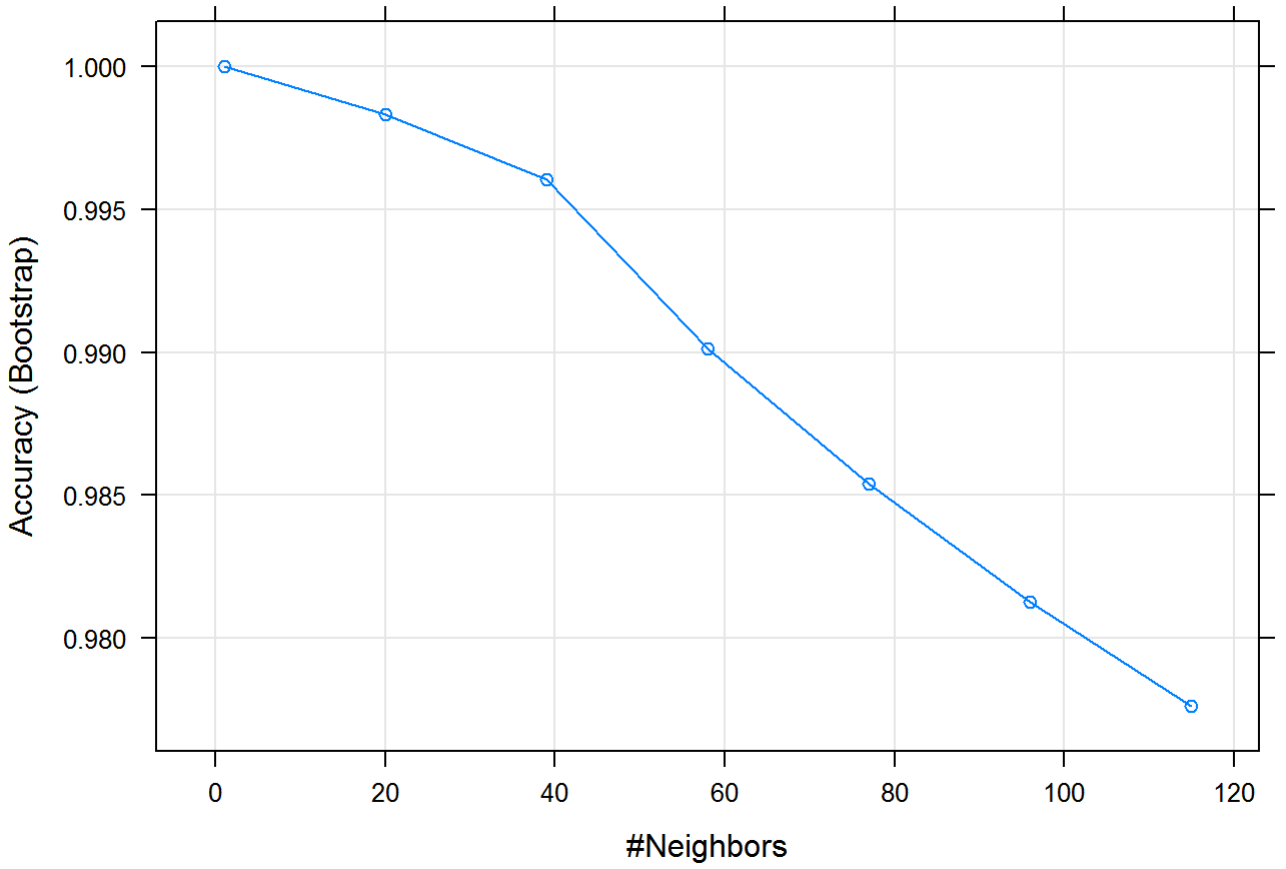
This algorithm uses the ‘k’ closest training examples in the test space. The output will depend on whether we use this algorithm for regression or for classification.

In classification types, the output is a class grouping. An object is grouped together with the majority of its ‘k’ neighbors. This is why, the value ‘k’ is usually an odd-number to ensure that there will always be a tie-breaker (the algorithm never fails). However, it would make sense to insist on ‘k’ being an even-number to demonstrate confidence in the performance of this method. If ‘k’ is even, we are allowing the possibility that the neighbors will not be helpful with the classification of the object (there is no majority). In this case, ‘k-NN’ will not be a suitable choice. Furthermore, We prefer ‘k’ to be large without sacrificing accuracy because large values of ‘k’ will make our analysis ‘smoother’. We will tune our ‘k-NN’ model by changing the values of ‘k’:

```
set.seed(5, sample.kind = "Rounding")
fit_knn<- train(class ~ ., data = train_set, method = "knn",
                tuneGrid = data.frame(k = seq(1,115,19)))
fit_knn
```

```
## k-Nearest Neighbors
##
## 6498 samples
##   20 predictor
##    2 classes: 'e', 'p'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 6498, 6498, 6498, 6498, 6498, 6498, ...
## Resampling results across tuning parameters:
##
##   k    Accuracy   Kappa
##     1  1.0000000  1.0000000
##    20  0.9983056  0.9966054
##    39  0.9960390  0.9920658
##    58  0.9901140  0.9802069
##    77  0.9854109  0.9707952
##    96  0.9812806  0.9625277
##   115  0.9776214  0.9552073
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

We see that the best tune is k=1 with an accuracy of 1. We can also see that if we consider larger ‘k’-values, the accuracy does not decrease significantly or very fast.



This is not surprising as all 20 predictors are categorical with 2 to 7 non-ordinal categories.

The mathematical distance between observations of the same ‘class’ are smaller compared to the mathematical distance between observations of different ‘classes’. Hence, we would expect neighbors to be of the same ‘class’. At k=1, we only look at the single closest neighbor to our object which will always be of the same ‘class’ as our object. It is theoretically improbable for our object to be closest to a neighbor of a different ‘class’ compared to a neighbor of the same ‘class’. However, as we include more neighbors (larger ‘k’ values), the number of neighbors of a different ‘class’ also increases because we are increasing the allowable distance to our object. This is why we see a decrease in accuracy with larger ‘k’ values.

Although k=1 will always maximize the accuracy, it produces a very ‘noisy’ analysis. If we include a large enough (but not too large) number of neighbors we should be able to get an accurate majority vote and a ‘smoother’ analysis. But, sometimes this does not work because neighbors are very scattered or the object has the same distance to neighbors of both ‘classes’.

Furthermore, because they are categorical predictors there are less possible values within each predictor compared to a numerical/continuous predictor; and it is exactly because of this, that absolute distinction can be made according to each combination of categories across all 20 predictors. Unlike continuous predictors, we cannot seperate values with intervals implying that neighbors can be unique even if they belong to the same ‘class’ or similar even if they belong to different ‘classes’ (uniqueness). So, we will always see a reduction in accuracy with higher ‘k’-values (due to uniqueness) but the decrease will be slow (due to smaller range of variability compared to continuous predictors).

We now complete our prediction:

```
pred_knn<- predict(fit_knn, test_set)
cm_knn<- confusionMatrix(pred_knn,test_set$class)

results<- rbind(results,c("knn",
                          signif(cm_knn$byClass[1]),
                          signif(cm_knn$byClass[2]),
                          signif(cm_knn$byClass[11]),
                          "< 4 minutes with tuning"))

kable(results, align = 'c') %>%
  kable_styling(full_width = FALSE)
```

model	sensitivity	specificity	accuracy	computation_time
glm	1	1	1	< 30 seconds
knn	1	1	1	< 4 minutes with tuning

RANDOM FOREST

'Random forests' can also be used for both classification and regression. In classification types, 'random forests' operate by growing a lot of decision trees at training time and reporting the class that is the mode of the classes of the individual trees. 'Random forests' are more accurate than decision trees because it does not overfit during training. 'Random forests' are also more suitable than other regression models when working with categorical variables because it uses 'branch networks'.

The argument 'ntree' sets the number of tress to grow. The larger the number of trees, the longer the computation time but the higher the accuracy. If we only want to make predictions, we can choose a lower 'ntree' setting but if we are interested in sorting the 'variable importance', we should use a higher 'ntree' setting. We will use ntree=200 because we are interested in the 'variable importance'. The parameter 'mtry' adjusts the number of variables used for splitting at each tree node. This is the parameter we will tune:

```
set.seed(10, sample.kind = "Rounding")
fit_rf<- train(class ~ ., data = train_set, method = "rf",
               tuneGrid = data.frame(mtry = seq(2,20,4)),
               ntree = 200)
fit_rf
```

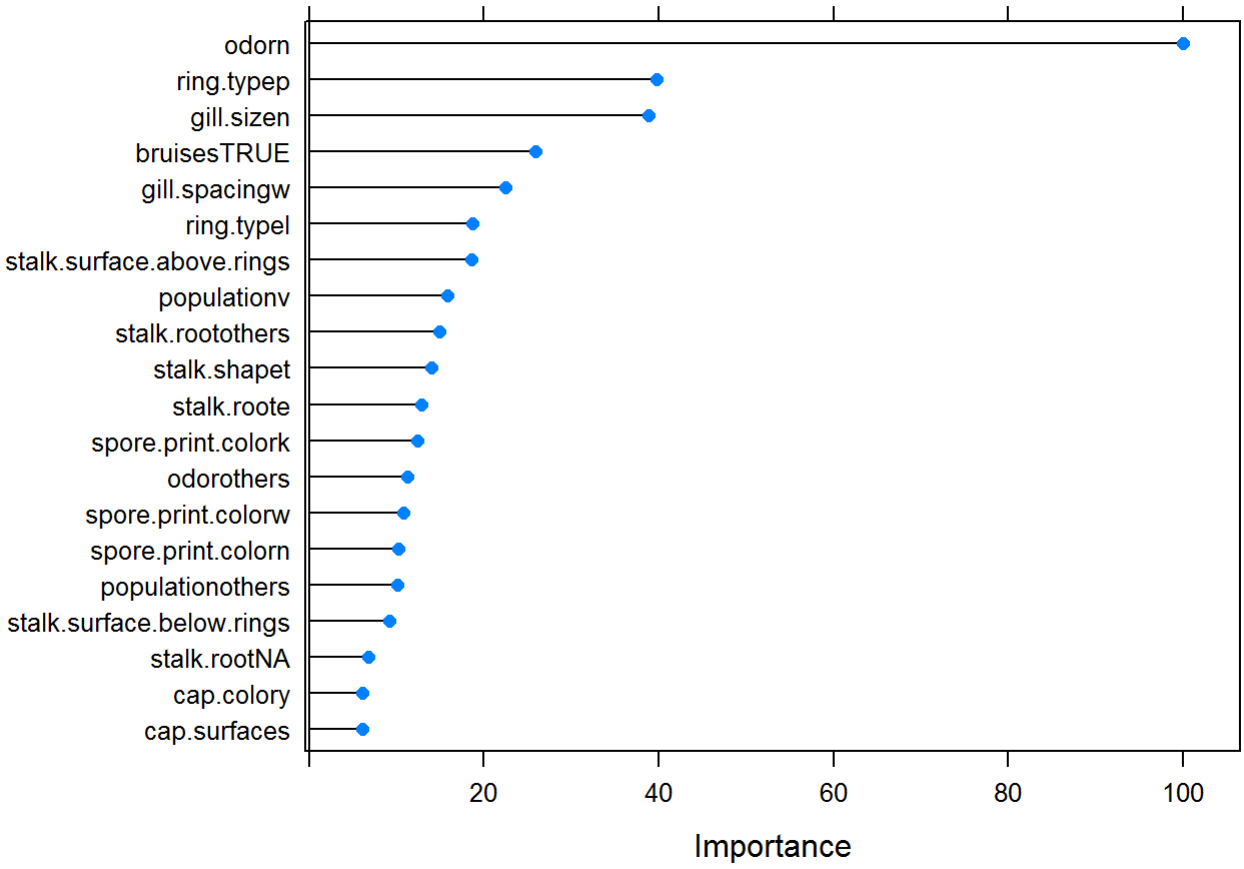
```
## Random Forest
##
## 6498 samples
##   20 predictor
##    2 classes: 'e', 'p'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 6498, 6498, 6498, 6498, 6498, 6498, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##    2    0.999129  0.9982559
##    6    1.000000  1.0000000
##   10    1.000000  1.0000000
##   14    1.000000  1.0000000
##   18    1.000000  1.0000000
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```

```
pred_rf<- predict(fit_rf, test_set)
cm_rf<- confusionMatrix(pred_rf,test_set$class)

results<- rbind(results,c("rf",
                          signif(cm_rf$byClass[1]),
                          signif(cm_rf$byClass[2]),
                          signif(cm_rf$byClass[11]),
                          "< 10 minutes with tuning"))

kable(results, align = 'c') %>%
  kable_styling(full_width = FALSE)
```

model	sensitivity	specificity	accuracy	computation_time
glm	1	1	1	< 30 seconds
knn	1	1	1	< 4 minutes with tuning
rf	1	1	1	< 10 minutes with tuning

Finally, we wil look at the ‘variable importance’ (%):

```
var_imp<- varImp(fit_rf)
plot(var_imp, top = 20)
```



We see that the variable 'odorn' (variable 'odor', category 'n'; 'dummy coded' to variable 'odorn') is absolutely important in this analysis as it was 100% used to make predictions.

LOCAL REGRESSION - LOCALLY ESTIMATED SCATTERPLOT SMOOTHING

'Local regression' generalizes moving average and polynomial regression. Its common methods which were developed for scatterplot smoothing, are LOESS (locally estimated scatterplot smoothing) and LOWESS (locally weighted scatterplot smoothing). They are non-parametric regression methods that combine multiple regression models in a k-nearest-neighbor-based model. It can also be used for classification:

```
set.seed(7, sample.kind = "Rounding")
fit_gam<- train(class ~ ., data = train_set, method = "gamLoess")
pred_gam<- predict(fit_gam, test_set)
cm_gam<- confusionMatrix(pred_gam,test_set$class)

results<- rbind(results,c("loess",
                          signif(cm_gam$byClass[1]),
                          signif(cm_gam$byClass[2]),
                          signif(cm_gam$byClass[11]),
                          "< 1 minute"))

kable(results, align = 'c') %>%
  kable_styling(full_width = FALSE)
```

model	sensitivity	specificity	accuracy	computation_time

model	sensitivity	specificity	accuracy	computation_time
glm	1	1	1	< 30 seconds
knn	1	1	1	< 4 minutes with tuning
rf	1	1	1	< 10 minutes with tuning
loess	1	1	1	< 1 minute

LINEAR DISCRIMINANT ANALYSIS (Statistically Inappropriate)

'LDA' tries to find a linear combination of features that characterizes or separates two or more events. 'LDA' uses continuous independent variables and has a categorical dependent variable; similar to the 'logit regression' model. However, 'LDA' assumes that the independent variables are normally distributed. This is a fundamental aspect of 'LDA'.

So, although we could convert our categorical variables by 'dummy coding'; our variables are most definitely not normally distributed. So, this model is not suitable for our analysis. However, we will include this method in our analysis for comparison and to demonstrate that applying fundamentally incorrect analysis could still yield results:

```
set.seed(4, sample.kind = "Rounding")
fit_lda<- train(class ~ ., data = train_set, method = "lda")
pred_lda<- predict(fit_lda, test_set)
cm_lda<- confusionMatrix(pred_lda,test_set$class)

kable(tibble(sensitivity = signif(cm_lda$byClass[1]),
       specificity = signif(cm_lda$byClass[2]),
       accuracy = signif(cm_lda$byClass[11])), align = 'c') %>%
  kable_styling(full_width = FALSE)
```

sensitivity	specificity	accuracy
1	0.991071	0.995536

As we can see, there are results to work with and if we do not understand what 'LDA' is, we would be making a technical mistake by using it.

Results

model	sensitivity	specificity	accuracy	computation_time
glm	1	1	1	< 30 seconds
knn	1	1	1	< 4 minutes with tuning
rf	1	1	1	< 10 minutes with tuning
loess	1	1	1	< 1 minute

We see that all the models tested returned an accuracy of 1 with differing ‘computation time’. To make a decision we will have to examine the technical and statistical suitability of the models on our analysis type.

The ‘k-NN’ method may not be the best choice because of the categorical nature of our predictors. Our data points are most likely very scattered in the 20-dimensional space. If we have to use the tuning parameter of k=1, we are not fully utilising the ‘k-NN’ method of majority votes. However, as we saw above, using larger ‘k’ values does not decrease the accuracy very significantly and we can choose to use this method if we want to. But, we have better alternatives.

The ‘random forest’ model is the most suitable by technicality as our variables are categorical. It also has a built-in function to measure and compare the ‘variable importance’. However, it takes the longest to compute out of all our models.

We are basically indiffirent between the ‘glm’ and the ‘loess’ method; apart from their ‘computation time’.

Our results ranking look like this:

rank	model	sensitivity	specificity	accuracy	computation_time
1	glm	1	1	1	< 30 seconds
2	rf	1	1	1	< 10 minutes with tuning
3	loess	1	1	1	< 1 minute
4	knn	1	1	1	< 4 minutes with tuning

Conclusion

The ‘glm’ or ‘logistic regression’ model is the best choice for this analysis and on this dataset.

Our analysis suggests that it is worth looking into eliminating some variables/predictors used in this analysis without sacrificing accuracy but improving calculation time. This will increase the model’s overall performance. The ‘random forest’ results gives us a bit of an idea of which variables have more predicting power but a full Categorical Principal Component Analysis (CATPCA) and Multiple Correspondence Analysis will give us more information.

In the earlier part of the analysis during data exploration, we had already simplified the data by reducing the number of categories. Our findings suggest that we can work with this reduced version for any future predictions instead of the original full categorization. But we should be careful in making this assumption because if a new species of mushroom appears and exhibits a whole different combination of categories; masked by those that we have merged, we would not be able to detect it.

The ‘attribute information’ for the simplified version is as follows:

```
## Response [https://raw.githubusercontent.com/syaqirafarouk/harvardx-capstone-cyo/master/capsto
ne-cyo-attribute-information.xls]
##   Date: 2020-05-21 06:45
##   Status: 200
##   Content-Type: application/octet-stream
##   Size: 30.7 kB
## <ON DISK>  C:\Users\mnstrchr\AppData\Local\Temp\RtmpmmcuVw\file190c78e77cb5.xls
```

Simplified Attribute Information	…2	…3	…4	…5	…6	…7	…8	…9	…10	…11	…12	…13	…14	…15
classes	p	poisonous	e	edible	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
cap-shape	x	convex	f	flat	others	b,c,k,s	NA	NA	NA	NA	NA	NA	NA	NA
cap-surface	f	fibrous	g	grooves	y	scaly	s	smooth	NA	NA	NA	NA	NA	NA
cap-color	n	brown	g	gray	w	white	y	yellow	e	red	others	b,c,r,p,u	NA	NA
bruises	t	bruises	f	no	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
odor	f	foul	n	none	others	a,l,c,y,m,p,s	NA	NA	NA	NA	NA	NA	NA	NA
gill-spacing	c	close	w	crowded	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
gill-size	b	broad	n	narrow	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
gill-color	n	brown	b	buff	h	chocolate	g	gray	p	pink	w	white	others	k,r,o,u,e,y
stalk-shape	e	enlarging	t	tapering	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
stalk-root	b	bulbous	e	equal	NA	missing	others	c,u,z,r	NA	NA	NA	NA	NA	NA
stalk-surface-above-ring	k	silky	s	smooth	others	f,y	NA	NA	NA	NA	NA	NA	NA	NA
stalk-surface-below-ring	k	silky	s	smooth	others	f,y	NA	NA	NA	NA	NA	NA	NA	NA
stalk-color-above-ring	p	pink	w	white	others	n,b,c,g,o,e,y	NA	NA	NA	NA	NA	NA	NA	NA
stalk-color-below-ring	p	pink	w	white	others	n,b,c,g,o,e,y	NA	NA	NA	NA	NA	NA	NA	NA
veil-color	w	white	others	n,o,y	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
ring-number	n	none	o	one	t	two	NA	NA	NA	NA	NA	NA	NA	NA
ring-type	e	evanescent	l	large	p	pendant	others	c,f,n,s,z	NA	NA	NA	NA	NA	NA
spore-print-color	k	black	n	brown	h	chocolate	w	white	others	b,r,o,u,y	NA	NA	NA	NA
population	s	scattered	v	several	y	solitary	others	a,c,n	NA	NA	NA	NA	NA	NA
habitat	g	grasses	p	paths	d	woods	others	l,m,u,w	NA	NA	NA	NA	NA	NA