# Data Science 'Mushrooms' Capstone Project

*András Gelencsér*

*14 December 2019*

## Contents

# Overview

This document describes the result of the capstone project for the HarvardX Data Science course based on the Mushrooms dataset avaiable in the UCI Machine Learning Repository (Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.)

The "Mushrooms" dataset was recorded down from The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf and is avaiable on the https://archive.ics.uci.edu/ml/datasets/mushroom url.

The goal of this project is to develop a model based on data science and machine learning techniques to predict if a mushroom is eatable based on the known attributes.

The dataset contains 8124 records and uses 22 attributes to describe the mushrooms. Each dataset contains additional attribute to flag if the mushroom is edible or not.

I used 80% of the avaiable data for train the model. The reamining 20% will be used as test dataset for evaluating the developed model.

For the developed model is very important to detect poisonus mushrooms to avoid possible health damages or death. Therefore the model performance will be evaluated with the F1 score instead of the overal accuracy. Because the models use the edible prediction as true value, I will try to get a high specificity with the developed models..

The following steps were done during the project for developing the algorithm:

1. Download the data and prepare for the analysis
2. Analyze the data structure
3. Define the classification model based on the result the data analysis
4. Implement and train the model based on the train set
5. Review the model based on the test set

I developed two different prediction model. The first model uses a decision tree, the second model uses a reduced set of the features with high correlation to the edibility.

Both developed models reached an F1 score of 1 with sensitivity 1 and specificity 1. That shows, that both models had a perfect prediction for the test set.

# Methods & analysis

## Preparation

For the analysis the data we will use the following R packages:

- tidyverse
- caret
- gridExtra
- ggplot2

the following R code loads and install the required packages if needed:

```
if(!require(tidyverse)) install.packages("tidyverse",
                         repos = "http://cran.us.r-project.org", dependencies = TRUE)
if(!require(caret)) install.packages("caret",
                         repos = "http://cran.us.r-project.org", dependencies = TRUE)
if(!require(gridExtra)) install.packages("gridExtra",
                         repos = "http://cran.us.r-project.org", dependencies = TRUE)
if(!require(ggplot2)) install.packages("ggplot2",
                         repos = "http://cran.us.r-project.org", dependencies = TRUE)
```

At first, we need to prepare the data set for the analysis. The mushrooms dataseet contains 8,124 records about different mushrooms.

The following R code downloads the data from the https://archive.ics.uci.edu/ml/datasets/mushroom site. To avoid unnecessary data traffic, the data will be downloaded only if not done yet.

```
#create directory for the data file if necessary
if (!dir.exists("mushrooms")){
  dir.create("mushrooms")
}
baseurl = "https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/"
files_to_download = c("agaricus-lepiota.data", "agaricus-lepiota.names")

for (f in files_to_download){
  #download the files only if not done yet
  if (!file.exists(paste("./mushrooms/",f,sep='')))
  {
    download.file(paste(baseurl,f,sep=''), paste("./mushrooms/",f,sep=''))
  }
}

fl <- file("mushrooms/agaricus-lepiota.data")
mushroom_data <- str_split_fixed(readLines(fl), ",", 23)
close(fl)
```

The dataset is stored in the “agaricus-lepiota.data” file. This file contains 23 columns separated via comma. There is no header line in the file. The description file “agaricus-lepiota.names” gives us information about the classes and the 22 attributes in the data file.

So we can extract the data by parsing the data file. The following R code reads the data and stores in a data frame:

```
fl <- file("mushrooms/agaricus-lepiota.data")
mushroom_data <- str_split_fixed(readLines(fl), ",", 23)
close(fl)
```

```r
#set the column names for the features
colnames(mushroom_data) <- c("classes", "cap-shape", "cap-surface", "cap-color",
                             "bruises?", "odor", "gill-attachment",
                             "gill-spacing", "gill-size", "gill-color", "stalk-shape",
                             "stalk-root", "stalk-surface-above-ring",
                             "stalk-surface-below-ring", "stalk-color-above-ring",
                             "stalk-color-below-ring", "veil-type", "veil-color",
                             "ring-number", "ring-type", "spore-print-color",
                             "population", "habitat")
#convert to data frame
mushroom_data <- as.data.frame(mushroom_data)

#remove temporary file variable
rm(fl)
```

For training the classification method we will use 80% of the avaiable data as training set. The remaining data will be used for evaluating the developed method. The following R code creates the train and test set:

```r
#initialize random sequenz
set.seed(1, sample.kind = "Rounding")
#create index for train and test set
#20% of the data will be used for the test set
test_idx = createDataPartition(y = mushroom_data$classes, times=1, p=0.2, list=FALSE)
train_data = mushroom_data[-test_idx,]
test_data = mushroom_data[test_idx,]
#remove temporary variables
rm(mushroom_data, test_idx)
```

## Available Data

The dataset contains 8,124 records. For the analysis we have data about 6,500 mushrooms (80% of all data). All the features are categorical with different possible values.

1. cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
2. cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
3. cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y
4. bruises?: bruises=t,no=f
5. odor: almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
6. gill-attachment: attached=a,descending=d,free=f,notched=n
7. gill-spacing: close=c,crowded=w,distant=d
8. gill-size: broad=b,narrow=n
9. gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y
10. stalk-shape: enlarging=e,tapering=t
11. stalk-root: bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?
12. stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
13. stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
14. stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
15. stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
16. veil-type: partial=p,universal=u
17. veil-color: brown=n,orange=o,white=w,yellow=y
18. ring-number: none=n,one=o,two=t
19. ring-type: cobwebby=c,evanescent=e,flaring=f,large=l, none=n,pendant=p,sheathing=s,zone=z
20. spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y
21. population: abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y

22. habitat: grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d

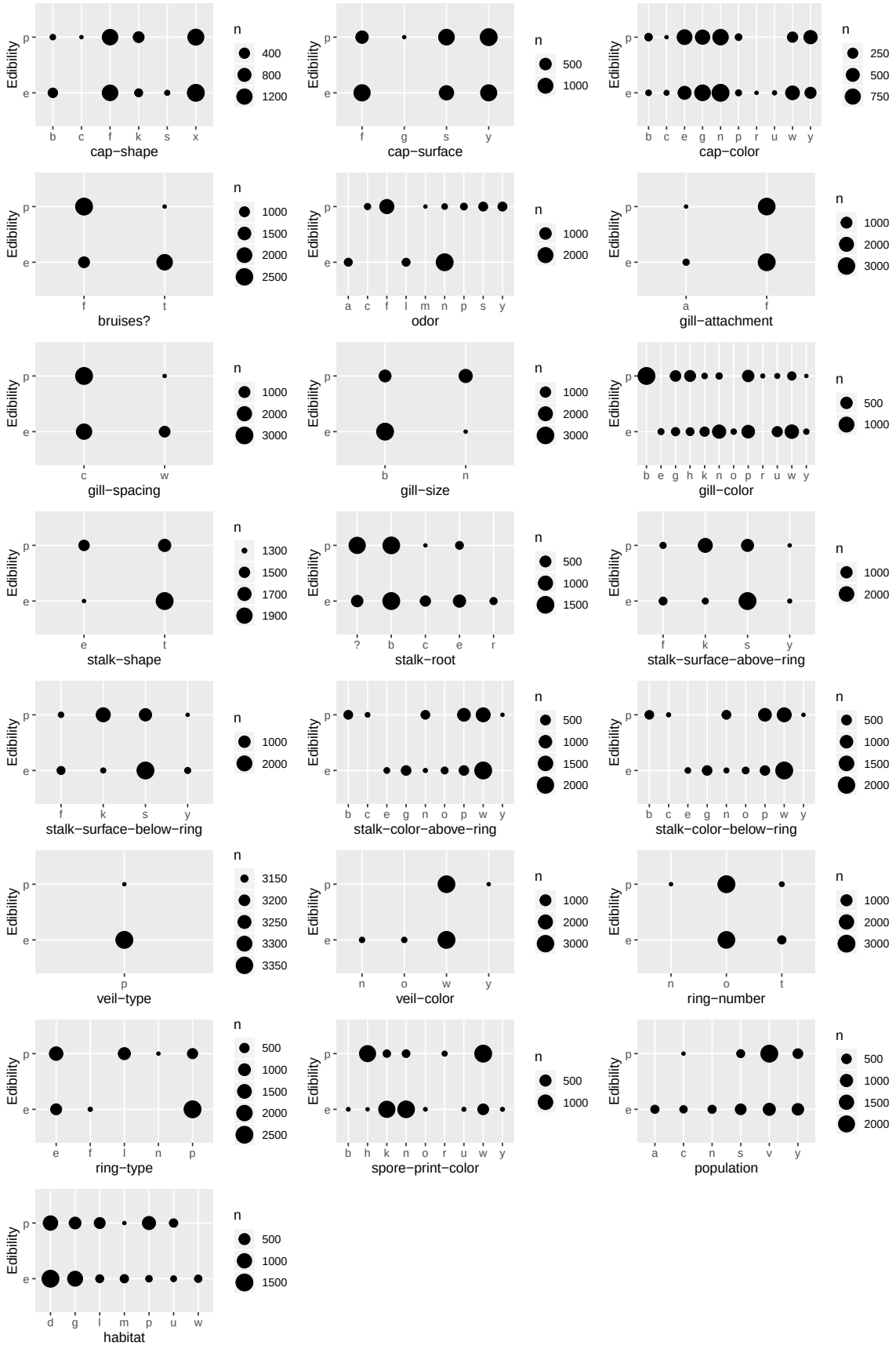
For testing the preformance the predition model we have about 1,600 mushrooms data (20% of all data) The dataset contains only the short 1 character code of the attribute values, but it's not disturbing for the model development.

## Feature analysis

As first step we can investigate the distribution of the features corresponding to the edibility classification.

The following code plots this distribution for each feature.

```
feature_ditribution_plot <- function(data){
  plots <- list()
  #Plots all the attributes for eatable <-> poisious
  for (i in 1:(ncol(data)-1))
  {
    summarized_data <- data %>% group_by(classes, .[,i+1]) %>% summarise(n = n())
    names(summarized_data)[2] <- "attr"
    plot <- summarized_data %>% ggplot(aes(attr , classes)) + geom_point(aes(size=n)) +
      xlab(names(data)[i+1]) + ylab("Edibility")
    plots[[i]] <- plot
  }
  rm(summarized_data, i, plot)
  grid.arrange(grobs=plots,ncol=3)
}

feature_ditribution_plot(train_data)
```


Edibility
p
e
b c f k s x
cap−shape
f g s y
cap−surface
b c e g n p r u w y
cap−color
f t
bruises?
a c f l m n p s y
odor
a f
gill−attachment
c w
gill−spacing
b n
gill−size
b e g h k n o p r u w y
gill−color
e t
stalk−shape
? b c e r
stalk−root
f k s y
stalk−surface−above−ring
f k s y
stalk−surface−below−ring
b c e g n o p w y
stalk−color−above−ring
b c e g n o p w y
stalk−color−below−ring
p
veil−type
n o w y
veil−color
n o t
ring−number
e f l n p
ring−type
b h k n o r u w y
spore−print−color
a c n s v y
population
d g l m p u w
habitat
n

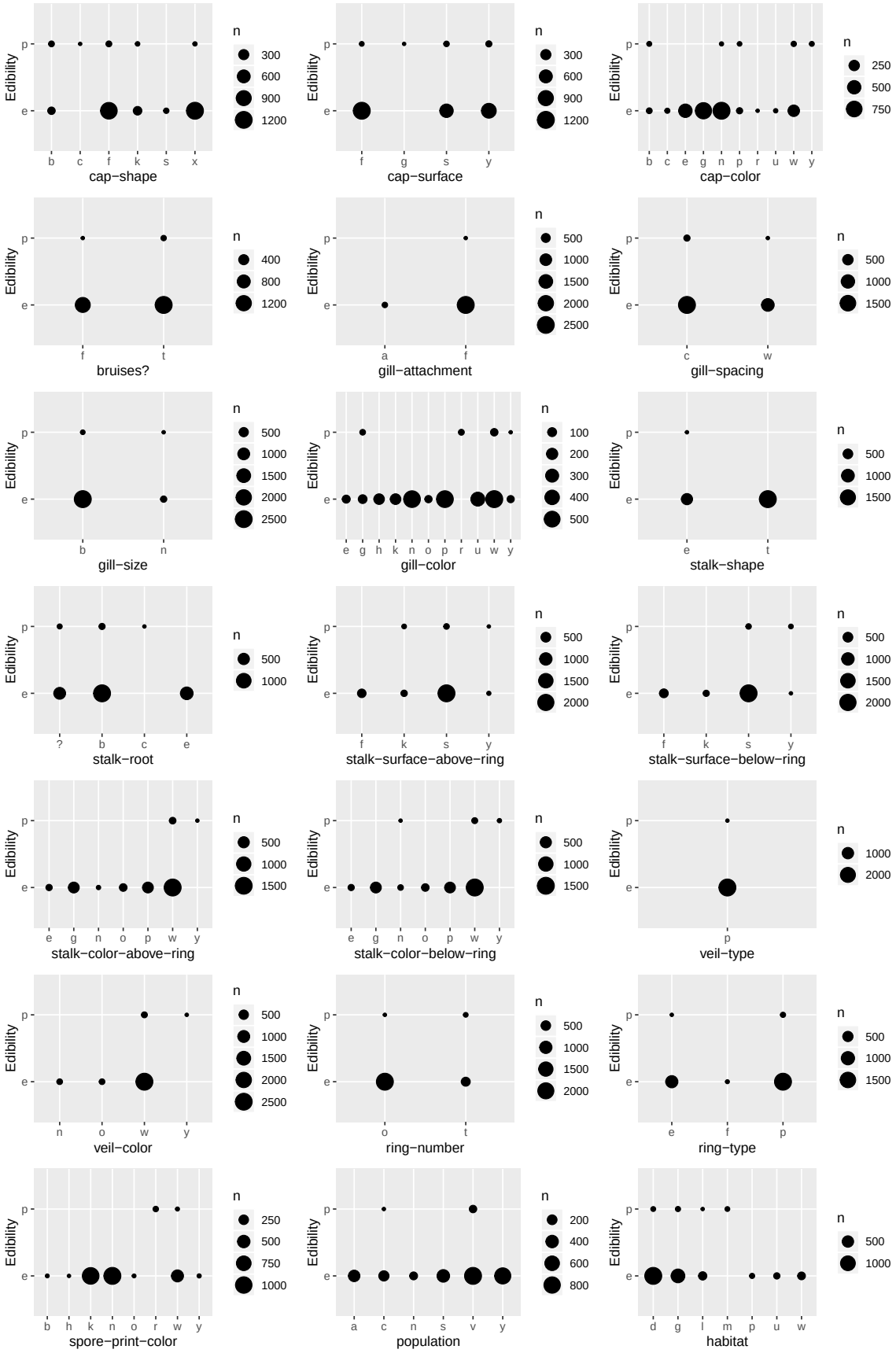
We can see, that the feature veil-type has only the value 'p' (partial), there is no data record with the veil-type value 'u' (universal). This feature can not provide decision information about the mushrooms, therefor we can remove this feature from the dataset.

On the other side, there are some features with values that can decide the classification. E.g. if we see a mushroom with the value 'f' (foul) for the feature odor, we can classify the mushroom as poisonous. The feature values 'a','c', 'f', 'l', 'm', 'p', 's', 'y' for the attribute odor can give us either 'e' or 'p' edibility classification. With this information we can classify 3,689 mushrooms. We can not classify only the mushrooms with the odor value 'n' (none). There are 2,809 mushrooms remaining in the dataset without classification.

We can try to analyse all the mushrooms, where the odor value is 'n'. For this analysis we can remove the attribute odor from the dataset, because we examine further only a specific odor value.

The following code removes the odor attribute and plots the feature distirbution for the remaining data:

```
odor_n <- train_data %>% filter(`odor` == 'n') %>% select(-`odor`)
feature_ditribution_plot(odor_n)
```


Edibility
p
e
n
cap−shape
b c f k s x
300 600 900 1200
cap−surface
f g s y
300 600 900 1200
cap−color
b c e g n p r u w y
250 500 750
bruises?
f t
400 800 1200
gill−attachment
a f
500 1000 1500 2000 2500
gill−spacing
c w
500 1000 1500
gill−size
b n
500 1000 1500 2000 2500
gill−color
e g h k n o p r u w y
100 200 300 400 500
stalk−shape
e t
500 1000 1500
stalk−root
? b c e
500 1000
stalk−surface−above−ring
f k s y
500 1000 1500 2000
stalk−surface−below−ring
f k s y
500 1000 1500 2000
stalk−color−above−ring
e g n o p w y
500 1000 1500
stalk−color−below−ring
e g n o p w y
500 1000 1500
veil−type
p
1000 2000
veil−color
n o w y
500 1000 1500 2000 2500
ring−number
o t
500 1000 1500 2000
ring−type
e f p
500 1000 1500
spore−print−color
b h k n o r w y
250 500 750 1000
population
a c n s v y
200 400 600 800
habitat
d g l m p u w
500 1000

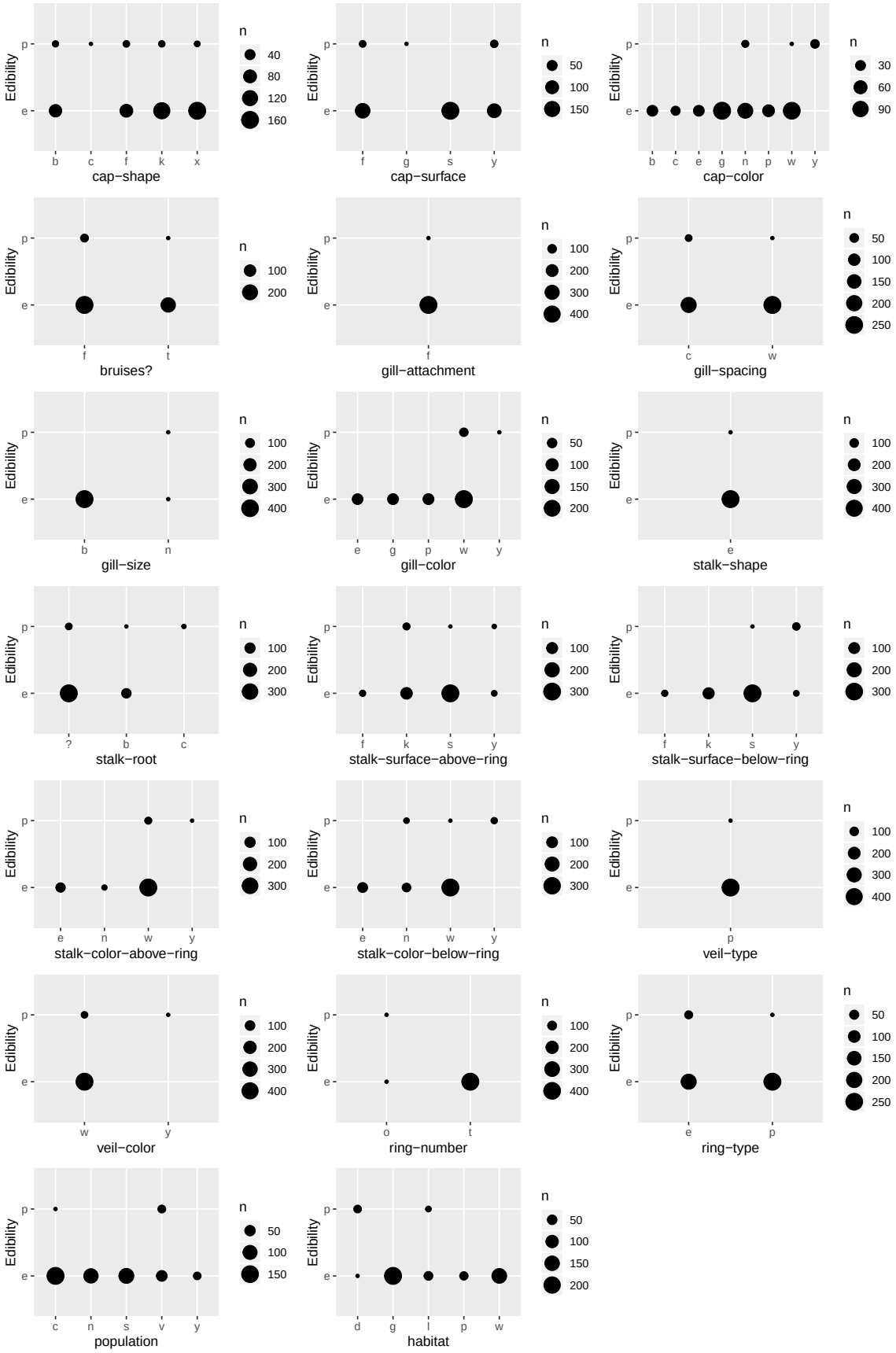
In the reduced data set we see again some features that we can use for classification the mushrooms.

E.g. if we see a mushroom with the value ‘k’ (black) for the feature spore-print-color, we can classify the mushroom as edible. The feature values ‘b’, ‘h’, ‘k’, ‘n’, ‘o’, ‘r’, ‘y’ for the feature spore-print-color can give us a classification for the remaining data set. We can not make a classification only it the spore-print-color value is ‘w’ (white).

We can reuse the same logik like for the feature odor and filter the remaing data with the spore-print-color feature value ‘w’.

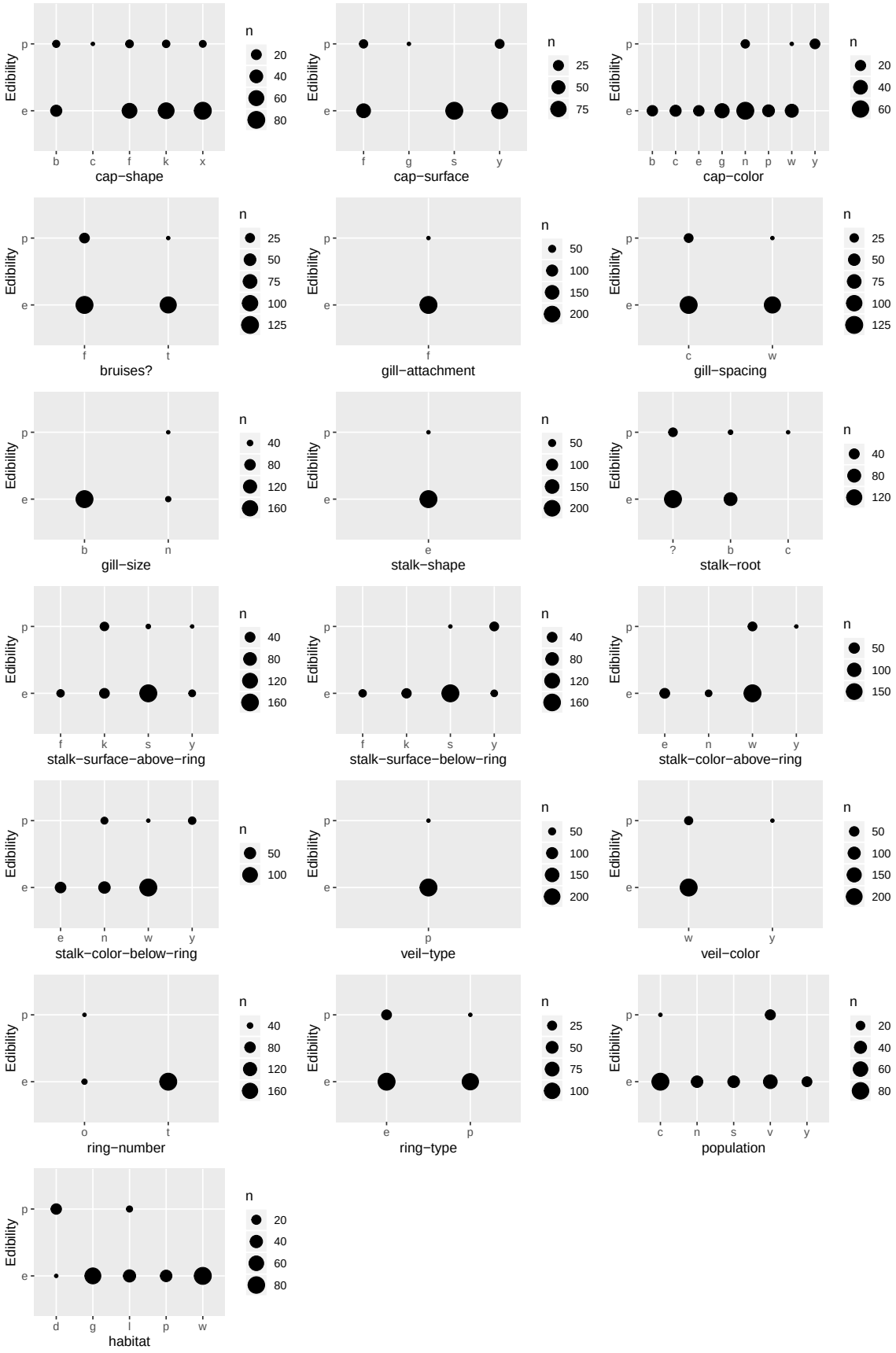
The following code removes the odor attribute and plots the feature distirbution for the remaining data:

```
spore_print_color_w <- odor_n %>% filter(`spore-print-color` == 'w') %>%
                       select(-`spore-print-color`)
feature_ditribution_plot(spore_print_color_w)
```


Edibility
p
e
n
cap−shape
b c f k x
40 80 120 160
cap−surface
f g s y
50 100 150
cap−color
b c e g n p w y
30 60 90
bruises?
f t
100 200
gill−attachment
f
100 200 300 400
gill−spacing
c w
50 100 150 200 250
gill−size
b n
100 200 300 400
gill−color
e g p w y
50 100 150 200
stalk−shape
e
100 200 300 400
stalk−root
? b c
100 200 300
stalk−surface−above−ring
f k s y
100 200 300
stalk−surface−below−ring
f k s y
100 200 300
stalk−color−above−ring
e n w y
100 200 300
stalk−color−below−ring
e n w y
100 200 300
veil−type
p
100 200 300 400
veil−color
w y
100 200 300 400
ring−number
o t
100 200 300 400
ring−type
e p
50 100 150 200 250
population
c n s v y
50 100 150
habitat
d g l p w
50 100 150 200

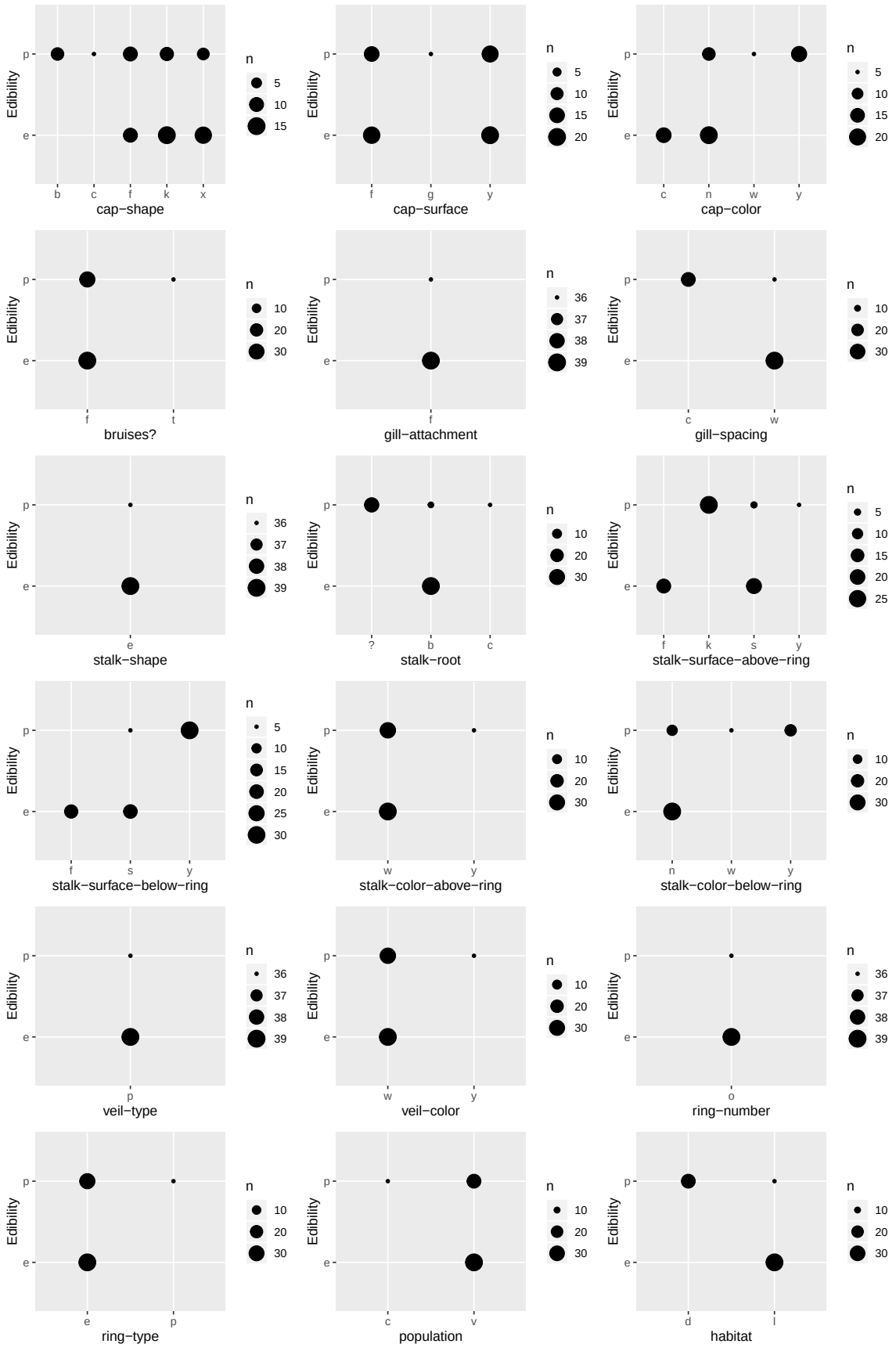
We can repeate this analysis and look for features that we can use for classifying the remaining data set. As next step we can get the gill-color feature and examine only the mushrooms with the grill color 'w' (white).

```
gill_color_w <- spore_print_color_w %>% filter(`gill-color` == 'w') %>%
                select(-`gill-color`)
feature_ditribution_plot(gill_color_w)
```


Edibility
p
e
n
cap−shape
b c f k x
20 40 60 80
cap−surface
f g s y
25 50 75
cap−color
b c e g n p w y
20 40 60
bruises?
f t
25 50 75 100 125
gill−attachment
f
50 100 150 200
gill−spacing
c w
25 50 75 100 125
gill−size
b n
40 80 120 160
stalk−shape
e
50 100 150 200
stalk−root
? b c
40 80 120
stalk−surface−above−ring
f k s y
40 80 120 160
stalk−surface−below−ring
f k s y
40 80 120 160
stalk−color−above−ring
e n w y
50 100 150
stalk−color−below−ring
e n w y
50 100
veil−type
p
50 100 150 200
veil−color
w y
50 100 150 200
ring−number
o t
40 80 120 160
ring−type
e p
25 50 75 100
population
c n s v y
20 40 60 80
habitat
d g l p w
20 40 60 80

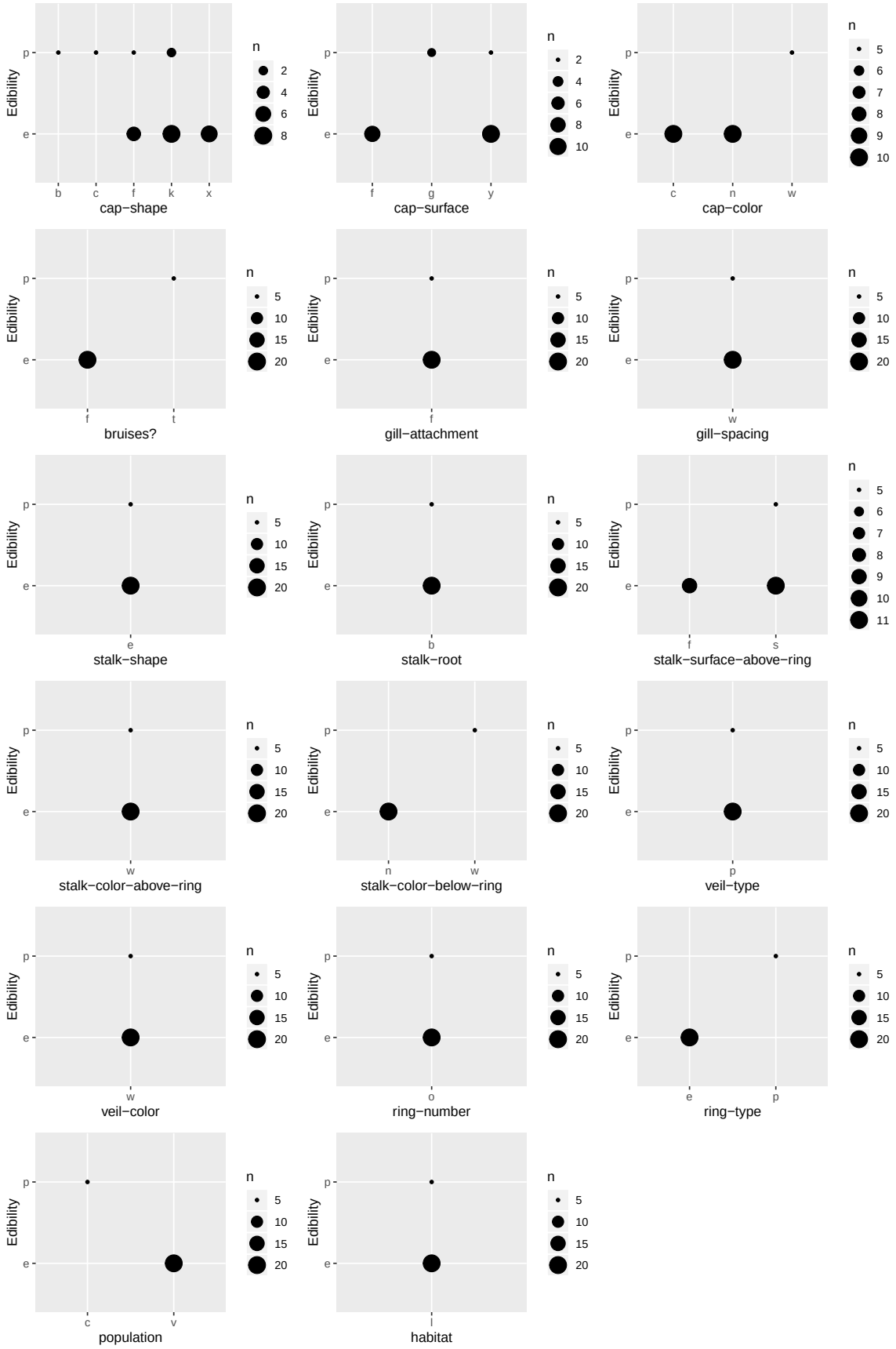
As next step we can get the gill-size feature and examine only the mushrooms with the grill size ‘n’ (narrow).

```
gill_size_n <- gill_color_w %>% filter(`gill-size` == 'n') %>% select(-`gill-size`)
feature_ditribution_plot(gill_size_n)
```


Edibility
p
e
b c f k x
cap−shape
f g y
cap−surface
c n w y
cap−color
f t
bruises?
f
gill−attachment
c w
gill−spacing
e
stalk−shape
? b c
stalk−root
f k s y
stalk−surface−above−ring
f s y
stalk−surface−below−ring
w y
stalk−color−above−ring
n w y
stalk−color−below−ring
p
veil−type
w y
veil−color
o
ring−number
e p
ring−type
c v
population
d l
habitat
n
5
10
15
20
25
30
36
37
38
39

As next step we can get the stalk-surface-below-ring feature and examine only the mushrooms with the stalk-surface-belowe-ring 's' (smooth).

```
stalk_surface_below <- gill_size_n %>% filter(`stalk-surface-below-ring` == 's') %>%
                       select(-`stalk-surface-below-ring`)
feature_ditribution_plot(stalk_surface_below)
```


Edibility
p
e
n
cap−shape
b c f k x
cap−surface
f g y
cap−color
c n w
bruises?
f t
gill−attachment
f
gill−spacing
w
stalk−shape
e
stalk−root
b
stalk−surface−above−ring
f s
stalk−color−above−ring
w
stalk−color−below−ring
n w
veil−type
p
veil−color
w
ring−number
o
ring−type
e p
population
c v
habitat
l

The remaining dataset has only 25 mushrooms. As we can see on the plot, now we can use some feature for classifying this remaining mushrooms. E.g. the feature ring-type give us a good classification: if we have a ring type of 'e' (evanescent) than the mushroom is edible. On the other case, if we have a ring-type of 'p' (pedant), the mushroom is poisonous.

With the steps about we defined a decision tree for classifying all the mushrooms in the train set. We can use this information to develop a classification model for the mushrooms.

The analyse above give us an other important information: some fatures are more relevant for the classification some features gives no additional information for the method. That means, the different features have different information weight for the classification.

We have to try to calculate a correlation between the features and the edibility. For the unordered categorical features we can use the uncertinity coefficient as correlation score. (See: https://en.wikipedia.org/wiki/Uncertainty_coefficient)

The uncertinity score builds on the conditinal entropy form the information theory.

The entropy for a single feature is defined as follows:

$$H(X) = -\sum_{x} P_X(x) log P_X(x)$$

The conditional entropy of X given Y is defined as follows:

$$H(X|Y) = -\sum_{x,y} P_{X,Y}(x,y) log P_{X|Y}(x|y)$$

The uncertinity coefficient of the two features can be calculated as follows:

$$U(X|Y) = \frac{H(X) - H(X|Y)}{H(X)}$$

We can implement an R function to calculate the entropy, conditional entropy and the uncertinity coefficient.

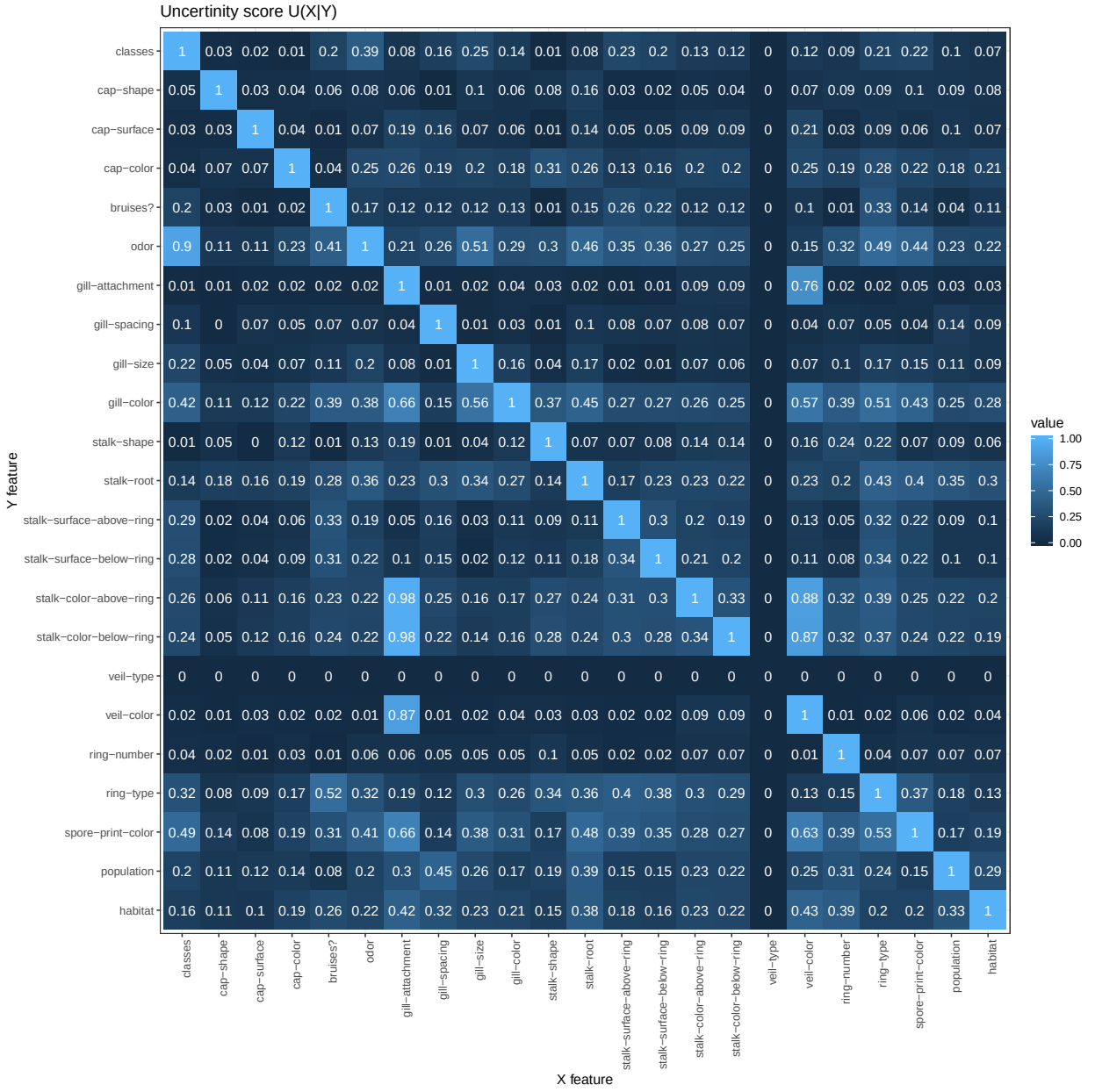
The following R code implements the calculation functions:

```
entropy <- function(dataset, colX){

  ret <- 0
  summarized_data <- dataset %>% group_by(.[,colX]) %>% summarise(n = n())
  rowCount <- nrow(dataset)
  level_items <- levels(dataset[,colX])
  for (item in level_items){
    prob <- summarized_data %>% filter(.[,1] == item) %>% pull(n) / rowCount
    ret <- ret + prob * log(prob,base = 2)
  }
  return (-ret)
}

cond_entropy <- function(dataset, colX, colY){
  ret <- 0
  rowCount <- nrow(dataset)
  summarized_y <- dataset %>% group_by(.[,colY]) %>% summarise(n = n())
  names(summarized_y)[1] <- "Y"
  level_itemsy <- levels(summarized_y$Y)

  for (itemy in level_itemsy){
    proby <- summarized_y %>% filter(Y == itemy) %>% pull(n) / rowCount
```

```
    summarized_x <- dataset %>% filter(.[,colY] == itemy) %>% group_by(.[,colX]) %>%
                    summarise(n = n())
    names(summarized_x)[1] <- "X"
    level_itemsx <- levels(summarized_x$X)
    for (itemx in level_itemsx){
      filtered <- summarized_x %>% filter(X == itemx)
      if (nrow(filtered) > 0 && proby != 0){
        probxy <-  filtered$n / rowCount
        probx_at_y <- probxy / proby
        ret <- ret + probxy * log(probx_at_y,base = 2)
      }
    }
  }
  return (-ret)
}

uncertinity <- function(dataset, colX, colY){
  entr <- entropy(dataset,colX)
  cond_entr <- cond_entropy(dataset, colX, colY)
  if (entr == 0){
    return(0)
  }
  return ( (entr - cond_entr) / entr)
}
```

We can plot the uncertinity coefficient between all the features as a matrix. The following R code creates the plot:

```
uncertinity_plot <- function(dataset){
  #calculate uncertinity for all feature pairs
  uncertinity_df <- data_frame(X=character(), idx=numeric(), Y=character(),
                               idy=numeric(), value=numeric())
  labels <- names(dataset)
  for (i in 1:ncol(dataset))
  {
    for(j in 1:ncol(dataset))
    {
      uncertinity_df <- bind_rows(uncertinity_df, data_frame(Y = labels[j], idy=j ,
                                  X=labels[i], idx=i, value=uncertinity(train_data,i,j)))
    }
  }

  #plot the uncertinity with colors
  uncertinity_df %>% ggplot(aes(x=reorder(X,idx), y=reorder(Y,-idy), fill=value)) +
    geom_tile() + geom_text(aes(label=round(value,2)), color='white') +
    theme_bw() + theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    xlab('X feature') + ylab('Y feature') + ggtitle('Uncertinity score U(X|Y)')
}

uncertinity_plot(train_data)
```




This plot shows the uncertinity coefficient between all feature pairs. The light blue values show a high uncertinity coefficient (high correlation) the dark blue values shows a low uncertinity coefficient (low correlation) The value 1 in the matrix shows a perfect correlation, therefore the matrix diagonal elements are 1. (The diagonal elements show the correlation of the features with himself, therefore there is a perfect correlation)

We see, that the feature veil-type has 0 correlation with all other features. In the first data analysis we saw, that the feature veil-type has only one value and has no prediction information, that causes the 0 correlation.

For the classification model we examine the first column of the matrix. This column shows the correlation betweem edibility and other feature. As we see, the feature odor has the highest correlation with the edibility. The other feature have only a correlation belowe 0.5.

We can use this information for developing a classification model that use only the features with high correlation with the edibility.

# The classification model

The classification model will be evaluated with the $F_1$ score. This score includes not only the overall accuracy, it considers the prediction specificity and sensitivity too.

The $F_1$ score is the harmonic average of the precision and reacall :

$$F_1 = \frac{1}{\frac{1}{2}(\frac{1}{recall} + \frac{1}{precision})}$$

where recall is the true positiv rate (sensitivity) and precision is the positiv prediction value (specificity).

The model should predict the edibility of the mushrooms, therefore it's more important to make the right prediction for the poisonous mushrooms as the edible. Therefore if two models have the same $F_1$ score, the model with the higher true negativ rate (specificity) will be declared as better model.

After the data analysis we can develop two different model: * model based on simple decision rules * model based on the feature value combinations

The following two chapter describe the two models.

## Decision tree model

As we saw before, we can develop a prediction model based on the feature distribution between the edible and poisonus classes. A very easy way to define a decision tree is to examine the generated charts like we did it before.

On the other side, this method is fixed on the current train data set. If the dataset changes, we must to make the analysis again manually.

Therefore we need to develop a method to create a decision tree automatically. For this method we can use the same logik as we did:

1. create the edibility distribution for all features for the dataset
2. search for features, that have values with only either 'e' or 'p' edibility. If we don't find a feature, than we cann't improve our decision tree anymore
3. if we found some feature in step 2, we select one of them and add to the decision tree
4. filter out all reords from the dataset that match the feature values with distinct edibility values
5. if there is no remaining record, than we reached a full decision tree for the train set
6. if there are remaining records, we start at step 1 again, but we use only the remaining records for the next loop

With this method we can find a decision tree for the train set, that we can use for predict the edibility for the test dataset.

For the prediction, we can go throug the features in the prediction tree and check if the feature in the actual record matches either the feature values with dictinct 'e' or 'p' edibility. If we find a match, we use the corresponding edibility value as prediction. If there is no match for all the feature in the prediction tree, we predict 'p' edibility. This is, because, we want to be sure not to predict edible for a possible poisonous mushroom.

## Feature based model

The uncertinity score matrix give us an overview about the correlation between the edibility classification and other features. With this information we can develop a prediction model. We can assume, that the most relevant features give use enough information to predict the edibility and we can ignore further features.

For the prediction we take the $n$ most relevant features and select the edibility to all the feature value combination to the features and examine which edibility values occures to this combination. For example, we

can use the featuers 'odor' and 'spore-print-color' and generate a summary for all existing feature combination to this two featue und the edibility classification.

The following R code lists all features value combination for the mentioned features:

```
odor_sporeprintcolor_values <- train_data %>% group_by(classes, odor, `spore-print-color`) %>% summaris
odor_sporeprintcolor_values %>% print(n = nrow(odor_sporeprintcolor_values))
```

```
## # A tibble: 24 x 3
## # Groups:   classes, odor [10]
##    classes odor  `spore-print-color`
##    <fct>   <fct> <fct>
##  1 e       a     k
##  2 e       a     n
##  3 e       a     u
##  4 e       l     k
##  5 e       l     n
##  6 e       l     u
##  7 e       n     b
##  8 e       n     h
##  9 e       n     k
## 10 e       n     n
## 11 e       n     o
## 12 e       n     w
## 13 e       n     y
## 14 p       c     k
## 15 p       c     n
## 16 p       f     h
## 17 p       f     w
## 18 p       m     w
## 19 p       n     r
## 20 p       n     w
## 21 p       p     k
## 22 p       p     n
## 23 p       s     w
## 24 p       y     w
```

There are some odor/spore-print-color value combination that have both 'p' and 'e' edibitlity classification. for this combination we don't have a definitly prediction. If we want to have a high specificity (we want to predict the poisonous mushrooms as good as possible), we have to predict 'p' in this case.

There are some odor/spore-print-color value combination with exclusive 'p' edibitlity classification. In this case we can predict 'p'.

There are some odor/spore-print-color value combination with exclusive 'e' edibitlity classification. In this case (and only in this case) we can predict, that the mushroom is edible.

Our prediciton model will search all feature value combination with exlusive 'e' edibility form the train set and predict for any record from the test set that mach to one of this value combination 'e' edibility. For all other records in the test set we predict 'p' edibility.

The number of the used feature influences the $F_1$ score of our prediction. With more used feature we can make better prediction. After a specific number of used feature we will get worse $F_1$ score again. That is the effect of the overtraining. Therefore we have to find out the nuber of the features to use for the best prediction performance.

## Implementation & result

In this chapter i will implement the definied two prediction model and analyse the result. For the feature based model i will use n-fold cross validation. For this purpose i implemented a function to calculate the n-fold cross validation for given train und test dataset calling a given function.

The following R code implements the n-fold cross validation (The function have to be called with cv_n >= 2):

```
#function for cross validation
#parameters:
#     trainset: the train set to use for the cross validation
#     cv_n:     the count of the cross validation
#     FUNC:     the function to call for the actual cross validation train and test set (calculated fro
#     ...:      additional parameter necessary for calling the provided function
#
#return:
#     dataframe with the function result for the cross validations (the data frame has cv_n items)

cross_validation <- function(trainset, cv_n, FUNC,...){

  #get the count of the data rows on the train set
  data_count = nrow(trainset)

  #initialize the data frame for the result
  values_from_cv = data_frame()

  #randomise the trainset.
  #If the train set is ordered (not randomised, like the movielens dataset) the cross validation
  #will not be independent and provide wrong result
  trainset_randomised <- trainset[sample(nrow(trainset)),]

  #create the train- and testset for the cross validation
  #we need cv_n run, therefore we use a loop
  for (i in c(1:cv_n)){
    #evaulate the size of the test set. This will be the 1/cv_n part of the data
    part_count = data_count / cv_n

    #select the data from the parameter train set
    #we get the part_count size elements from the parameter train set
    idx = c(   (trunc((i-1) * part_count) + 1) : trunc(i * part_count) )

    #tmp holds the new test set
    test = trainset_randomised[idx,]
    #train holds the new test set
    train = trainset_randomised[-idx,]

    #call the provided function to the actual train and test set.
    akt_value <- FUNC(train, test,...)

    #add the result to the data frame
    #the column 'cv' contains the idx of the cross validation run
    values_from_cv <- bind_rows(values_from_cv, akt_value %>% mutate(cv = i))
  }
```

```
  #return the results of each cross validation
  return(values_from_cv)
}
```

For calculation of the $F_1$ score, i will us the confuseMatrix R function.

## Decision tree model

As first iteration I implemented a naive decision tree model based on the analysis in the previos chapters. This implementation is not trainable, it depends on the obervation about the train dataset. The R implementation this naive decision tree is wie follow:

```
predict_dectree_naive <- function(features){
  predicted <- tibble(y = character())
  for (i in 1 : nrow(features))
  {
    y <- tryCatch({
        odor <- features[i,]$odor
        spore_print_color <- features[i,]$`spore-print-color`
        gill_color <- features[i,]$`gill-color`
        gill_size <- features[i,]$`gill-size`
        stalk_surface <- features[i,]$`stalk-surface-below-ring`
        ring_type <- features[i,]$`ring-type`

        if (odor %in% c('c','f','m','p','s','y')){
          'p'
        }
        else if (odor  %in% c('a','l')){
          'e'
        }
        else if (spore_print_color %in% c('e','g', 'n','o','p','y')){
          'e'
        }
        else if (spore_print_color %in% c('r')){
          'p'
        }
        else if (gill_color %in% c('y')){
          'p'
        }
        else if (gill_color %in% c('e','g','p')){
          'e'
        }
        else if (gill_size %in% c('b')){
          'e'
        }
        else if (stalk_surface %in% c('f')){
          'e'
        }
        else if (stalk_surface %in% c('y')){
          'p'
        }
        else if (ring_type %in% c('e')){
          'e'
        }
```

```
        else{
          'p'
        }
    }, warning = function(w){
        return('p')
    }, error = function(e) {
        return('p')
    })
    predicted <- bind_rows(predicted, data_frame(y = y))
  }

  predicted <- predicted %>% mutate(y=factor(y, levels=c('e','p'))) %>% pull(y)
  return(predicted)
}
```

With this function we can predict the edibility for the test set:

```
predicted_naive <- predict_dectree_naive(test_data %>% select(-classes))
result_naive_tree_model <- confusionMatrix( predicted_naive, test_data$classes)
result_naive_tree_model
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   e   p
##          e 831   0
##          p  11 784
##
##                Accuracy : 0.9932
##                  95% CI : (0.9879, 0.9966)
##     No Information Rate : 0.5178
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9865
##
##  Mcnemar's Test P-Value : 0.002569
##
##             Sensitivity : 0.9869
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9862
##              Prevalence : 0.5178
##          Detection Rate : 0.5111
##    Detection Prevalence : 0.5111
##       Balanced Accuracy : 0.9935
##
##        'Positive' Class : e
##
```

The $F_1$ score is 0.9934 with a specificity of 1.

The next step is to develop a method, that builds the decision tree based on the train set automatically.

The following R code implements the building of the decision tree based on a given dataset:

```
extend_decision_tree <- function(dataset, decision_rule){

  newRules <- data_frame(feature=character(),decided_proz=numeric(), e_values=character(), p_values=cha

  for (i in 1:(ncol(dataset)-1))
  {
    feature <- names(dataset)[i+1]
    summarized_data <- dataset %>% group_by(classes, .[,i+1]) %>% summarise(n = n())
    names(summarized_data)[2] <- "attr"
    values <- levels(summarized_data$attr)
    e_values <- list()
    p_values <- list()
    count = nrow(dataset)
    decided_count = 0
    e_pos <- 1
    p_pos <- 1
    for(val in values){
      e_count <- summarized_data %>% filter(classes=='e' & attr == val) %>% pull(n)
      e_count <- ifelse(is_empty(e_count),0,e_count)
      p_count <- summarized_data %>% filter(classes=='p' & attr == val) %>% pull(n)
      p_count <- ifelse(is_empty(p_count),0,p_count)


      if (e_count == 0 && p_count > 0){
        p_values[[p_pos]] <- val
        decided_count <- decided_count + p_count
        p_pos <- p_pos + 1
      }
      if (e_count > 0 && p_count == 0){
        e_values[[e_pos]] <- val
        decided_count <- decided_count + e_count
        e_pos <- e_pos + 1
      }
    }
    if (decided_count > 0){
      e_values <- paste(e_values, collapse=",")
      p_values <- paste(p_values,collapse=",")
      newRules <- bind_rows(newRules, data_frame(feature=feature, decided_proz=decided_count / count, e
                                                 p_values = p_values))

    }
  }

  if (nrow(newRules) > 0){
    idx <- which.max(newRules$decided_proz)
    bestRule <- newRules[idx,]
    e_values <- unlist(str_split(bestRule$e_values, ','))
    p_values <- unlist(str_split(bestRule$p_values, ','))

    remaining_data <- dataset %>% filter( !(.[,bestRule$feature] %in% e_values) & !(.[,bestRule$feature
    decision_rule <-bind_rows(decision_rule, bestRule)
    if (nrow(remaining_data) > 0){
      decision_rule <- extend_decision_tree(remaining_data, decision_rule)
```

```
    }
  }

  return(decision_rule)
}

train_decision_tree_model <- function(dataset){


  tree <- data_frame(feature=character(), decided_proz=numeric(), e_values=character(), p_values=charac
  tree <- extend_decision_tree(dataset, tree)
  return(tree)
}
```

Additionally we need a function to predict the edibility for the test set based on the trained decision tree. The funcion has to chekc the first decision rules existing in the decision tree and check if the actual record match either the values for the edible or the poisonous classification. If there is no match, the function have to go through the following rules until a match is found or the decision tree has no more rules. If there was no match in the entire decision tree, we predict poisonous classification. (We want to be on the safe side with the poisonous mushrooms.)

The following R code implements the predict function for the trained decision tree:

```
predict_decision_tree <- function(dataset, tree){
  ret <- list()
  for(i in 1:nrow(dataset)){
    y <- 'p'
    count_rules <- nrow(tree)
    for(j in 1:count_rules){
      feature <- tree$feature[j]
      e_values <- unlist(str_split(tree$e_values[j], ','))
      p_values <- unlist(str_split(tree$p_values[j], ','))

      if (dataset[i,feature] %in% e_values){
        y <- 'e'
        break;
      }

      if (dataset[i,feature] %in% p_values){
        y <- 'p'
        break;
      }
    }
    ret <- c(ret, y)
  }
  return(factor(ret, levels = c('e','p')))
}
```

```
predict_tree <- train_decision_tree_model(train_data)
predicted <- predict_decision_tree(test_data, predict_tree)
result_decision_tree_model <- confusionMatrix( predicted, test_data$classes)
result_decision_tree_model
```

```
## 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
##
```

With the trained decision tree we have $F_1$ score of 1 with a specificity 1. This model made a perfect prediction for the test set.

## Feature based model

# Conclusion