Mushroom Classification Unsupervised Machine Learning

Table of contents

```r
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(ggplot2))
suppressPackageStartupMessages(library(reshape2))
suppressPackageStartupMessages(library(cluster))
suppressPackageStartupMessages(library(factoextra))
suppressPackageStartupMessages(library(NbClust))
suppressPackageStartupMessages(library(corrplot))
suppressPackageStartupMessages(library(FactoMineR))
suppressPackageStartupMessages(library(VIM))
suppressPackageStartupMessages(library(NMF))
suppressPackageStartupMessages(library(Rtsne))
suppressPackageStartupMessages(library(umap))
suppressPackageStartupMessages(library(pheatmap))
set.seed(123)
```

1 Introduction

In this project, we'll explore the Mushroom dataset using unsupervised machine learning techniques to discover hidden patterns and natural groupings within mushroom characteristics. Rather than predicting edibility directly, we'll investigate whether mushrooms naturally cluster into distinct groups based on their physical features, and examine how these discovered patterns relate to the known edible/poisonous classifications.

The dataset includes artificial descriptions of various mushrooms with their characteristics, based upon the Audobon Society Field Guide. Using unsupervised learning, we'll attempt to uncover the underlying structure in mushroom features without using the edibility labels during analysis. For more information about this data, it can be found at the UC Irvine Machine Learning Repository.

First, we need to load in our data.

```r
data <- read.csv("data/mushrooms.csv")

head(data)
```

```
  class cap.shape cap.surface cap.color bruises odor gill.attachment
1     p         x           s         n       t    p               f
2     e         x           s         y       t    a               f
3     e         b           s         w       t    l               f
4     p         x           y         w       t    p               f
5     e         x           s         g       f    n               f
```

```
6     e         x           y         y       t    a               f
  gill.spacing gill.size gill.color stalk.shape stalk.root
1            c         n          k           e          e
2            c         b          k           e          c
3            c         b          n           e          c
4            c         n          n           e          e
5            w         b          k           t          e
6            c         b          n           e          c
  stalk.surface.above.ring stalk.surface.below.ring stalk.color.above.ring
1                        s                        s                      w
2                        s                        s                      w
3                        s                        s                      w
4                        s                        s                      w
5                        s                        s                      w
6                        s                        s                      w
  stalk.color.below.ring veil.type veil.color ring.number ring.type
1                      w         p          w           o         p
2                      w         p          w           o         p
3                      w         p          w           o         p
4                      w         p          w           o         p
5                      w         p          w           o         e
6                      w         p          w           o         p
  spore.print.color population habitat
1                 k          s       u
2                 n          n       g
3                 n          n       m
4                 k          s       u
5                 n          a       g
6                 k          n       g
```

```r
summary(data)
```

```
    class             cap.shape          cap.surface         cap.color
 Length:8124        Length:8124        Length:8124        Length:8124
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character
   bruises              odor           gill.attachment    gill.spacing
 Length:8124        Length:8124        Length:8124        Length:8124
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character
```

```
  gill.size           gill.color         stalk.shape          stalk.root       
 Length:8124        Length:8124        Length:8124        Length:8124       
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  
 stalk.surface.above.ring stalk.surface.below.ring stalk.color.above.ring
 Length:8124              Length:8124              Length:8124           
 Class :character         Class :character         Class :character      
 Mode  :character         Mode  :character         Mode  :character      
 stalk.color.below.ring  veil.type          veil.color       
 Length:8124            Length:8124        Length:8124       
 Class :character       Class :character   Class :character  
 Mode  :character       Mode  :character   Mode  :character  
 ring.number         ring.type          spore.print.color   population       
 Length:8124        Length:8124        Length:8124        Length:8124       
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  
   habitat         
 Length:8124       
 Class :character  
 Mode  :character  
```

Before we get too far, we should make sure we know what our column names are.

```
colnames(data)
```

```
 [1] "class"                    "cap.shape"               
 [3] "cap.surface"              "cap.color"               
 [5] "bruises"                  "odor"                    
 [7] "gill.attachment"          "gill.spacing"            
 [9] "gill.size"                "gill.color"              
[11] "stalk.shape"              "stalk.root"              
[13] "stalk.surface.above.ring" "stalk.surface.below.ring"
[15] "stalk.color.above.ring"   "stalk.color.below.ring"  
[17] "veil.type"                "veil.color"              
[19] "ring.number"              "ring.type"               
[21] "spore.print.color"        "population"              
[23] "habitat"                 
```

Let's examine the distribution of our known classes (which we'll use later for validation but not during clustering):

```
distribution <- table(data$class)
distribution
```

```

   e    p 
4208 3916 
```

```
distribution[1] / (distribution[1] + distribution[2])
```

```
        e 
0.5179714 
```

```
distribution[2] / (distribution[1] + distribution[2])
```

```
        p 
0.4820286 
```

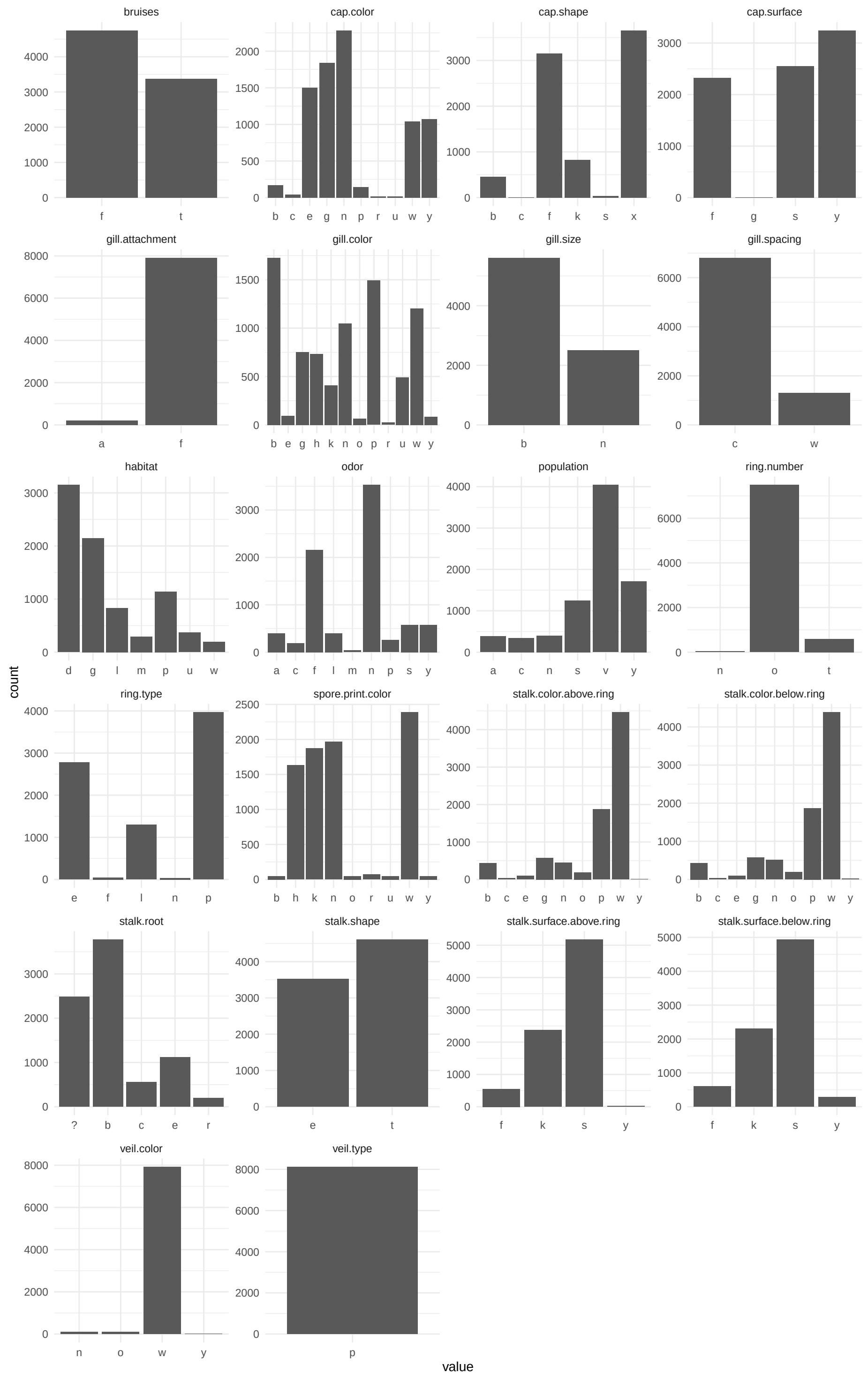
This shows a pretty even split. Let's also check for any missing values in the data.

```
anyNA(data)
```

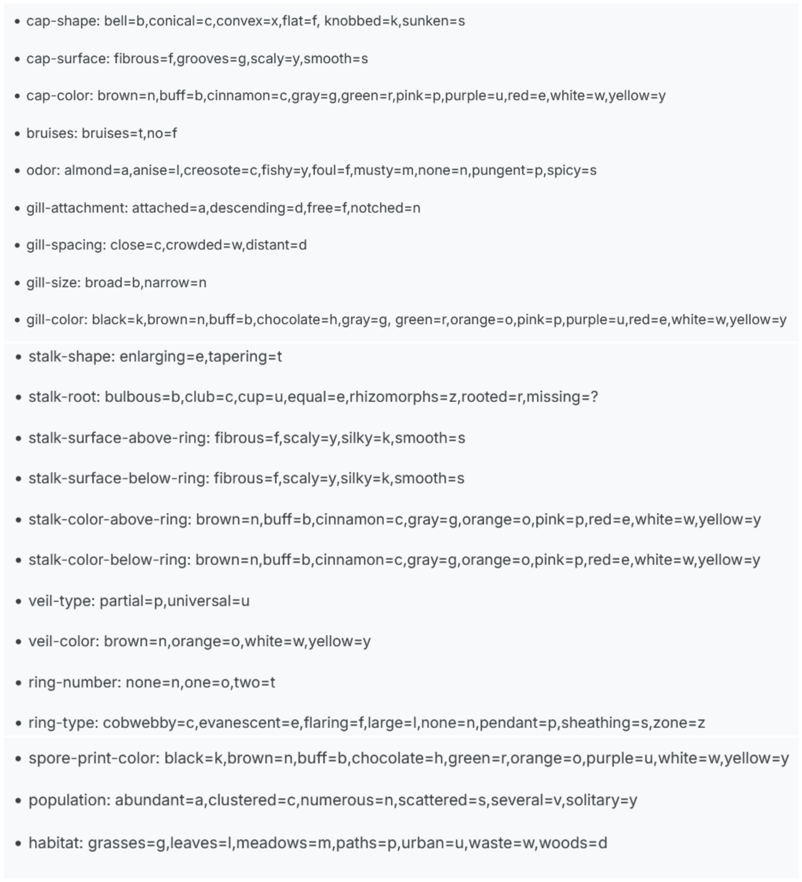
```
[1] FALSE
```

We can see that there are no missing values in this dataset. Now let's visualize the distribution of features:

```
data_long <- data %>% gather(data, "key", class:habitat)
colnames(data_long) <- c("group", "value")
data_long <- data_long[data_long$group != "class",]

ggplot(data_long) +
  geom_bar(aes(x = value, group = group)) +
  facet_wrap(~group, ncol = 4, scales = "free") +
  theme_minimal()
```



bruises
cap.color
cap.shape
cap.surface
gill.attachment
gill.color
gill.size
gill.spacing
habitat
odor
population
ring.number
ring.type
spore.print.color
stalk.color.above.ring
stalk.color.below.ring
stalk.root
stalk.shape
stalk.surface.above.ring
stalk.surface.below.ring
veil.color
veil.type
count
value

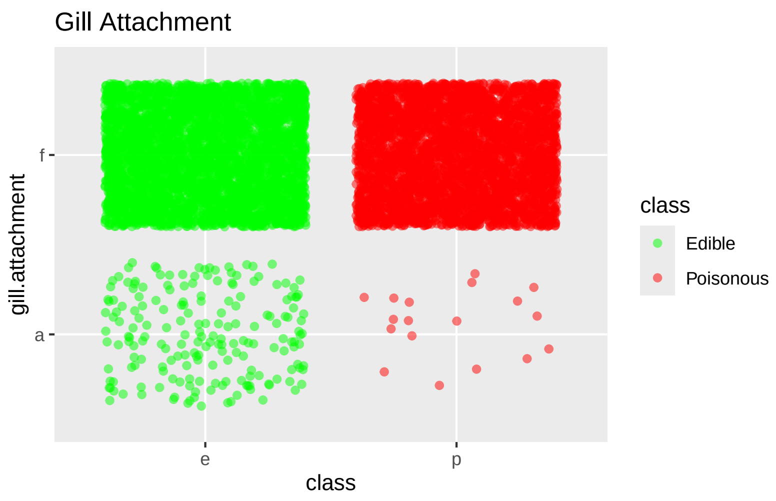


- cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
- cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
- cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y
- bruises: bruises=t,no=f
- odor: almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s
- gill-attachment: attached=a,descending=d,free=f,notched=n
- gill-spacing: close=c,crowded=w,distant=d
- gill-size: broad=b,narrow=n
- 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
- stalk-shape: enlarging=e,tapering=t
- stalk-root: bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
- stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- veil-type: partial=p,universal=u
- veil-color: brown=n,orange=o,white=w,yellow=y
- ring-number: none=n,one=o,two=t
- ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z
- spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y
- population: abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y
- habitat: grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

1.1 Exploratory Analysis

We can also see that some of our variables are predominantly one value. Let's take a closer look at these predictors to see if they might have an impact on our model if it is worth removing them in order to prevent an overly weighted predictor from impacting the model.

```r
ggplot(data, aes(x = class, y = gill.attachment, col = class)) +
  geom_jitter(alpha = 0.5) +
  ggtitle("Gill Attachment") +
  scale_color_manual(breaks = c("e", "p"),
                     labels = c("Edible", "Poisonous"),
                     values = c("green", "red"))
```

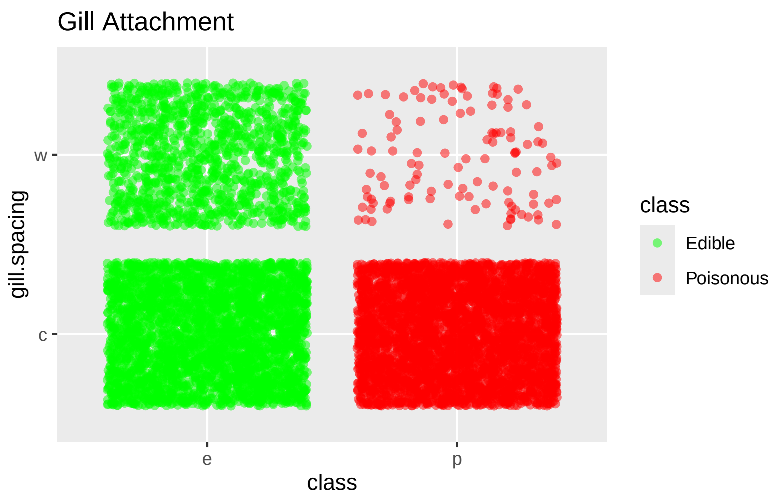


```r
table(data$class, data$gill.attachment)
```

```
       a    f
  e  192 4016
  p   18 3898
```

Here we can see a pretty even split between the edible and poisonous classes. There appears to be a tendency for attached veils to be more often edible rather than poisonous

```r
ggplot(data, aes(x = class, y = gill.spacing, col = class)) +
  geom_jitter(alpha = 0.5) +
  ggtitle("Gill Attachment") +
  scale_color_manual(breaks = c("e", "p"),
                     labels = c("Edible", "Poisonous"),
                     values = c("green", "red"))
```

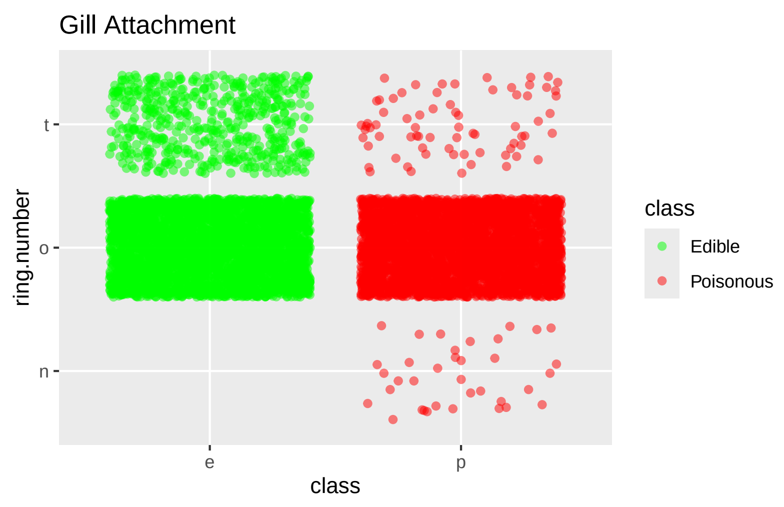


```r
table(data$class, data$gill.spacing)
```

```
       c    w
  e 3008 1200
  p 3804  112
```

This is also pretty split when considering gill spacing. Here we can see a tendency for crowded gills to be edible more often.

```r
ggplot(data, aes(x = class, y = ring.number, col = class)) +
  geom_jitter(alpha = 0.5) +
  ggtitle("Gill Attachment") +
  scale_color_manual(breaks = c("e", "p"),
                     labels = c("Edible", "Poisonous"),
                     values = c("green", "red"))
```

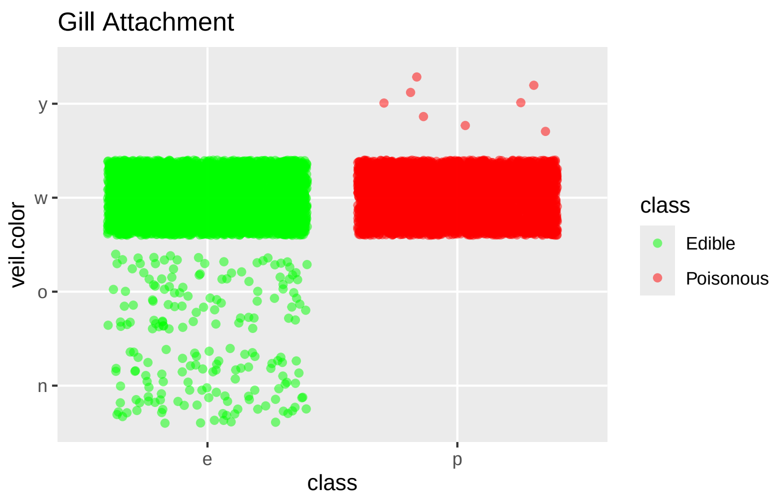


```
table(data$class, data$ring.number)
```

```
       n    o    t
  e    0 3680  528
  p   36 3808   72
```

Here we can see a pretty big difference for mushrooms that have no ring numbers. This data might indicate that all mushrooms that have no rings would be poisonous but some research shows that this is not true. Therefore, I have removed this column from consideration in the model.

```
ggplot(data, aes(x = class, y = veil.color, col = class)) +
  geom_jitter(alpha = 0.5) +
  ggtitle("Gill Attachment") +
  scale_color_manual(breaks = c("e", "p"),
                     labels = c("Edible", "Poisonous"),
                     values = c("green", "red"))
```



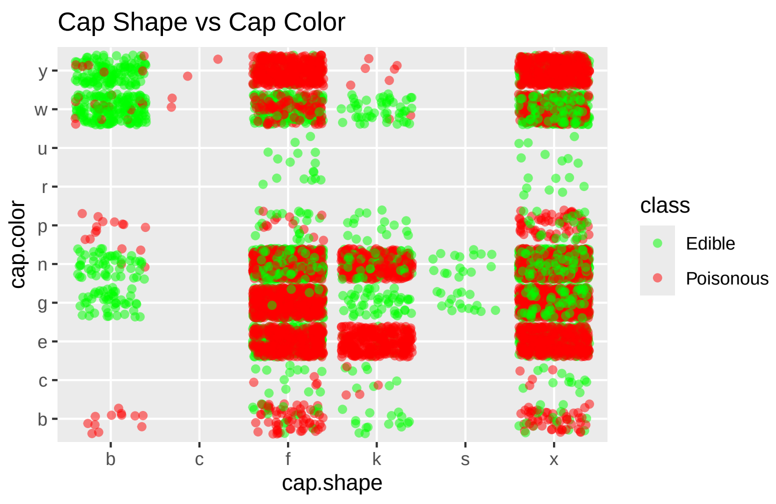
```
table(data$class, data$veil.color)
```

```
       n    o    w    y
  e   96   96 4016    0
  p    0    0 3908    8
```

In veil color we also see strong weighting towards a specific class based on the veil color. Therefore, I have removed this column from consideration in the model as well.

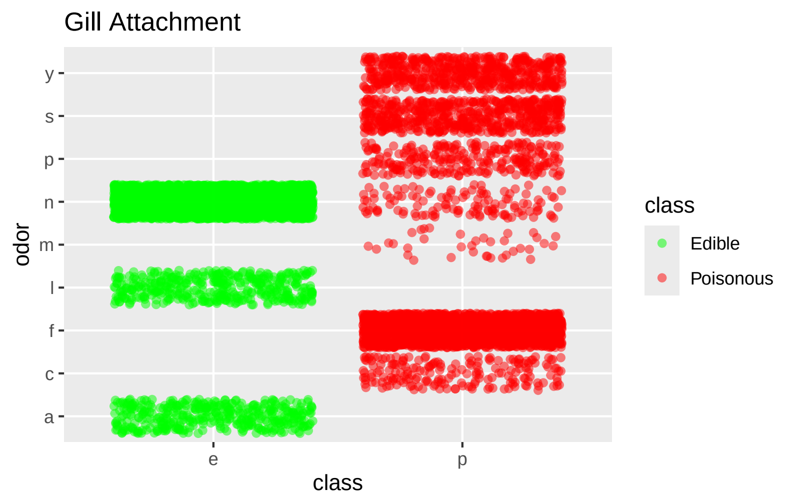
A fun fact about mushrooms is that typically only the ones with a bell shape. Let's take a look and see if that is true with our data.

```
ggplot(data, aes(x = cap.shape, y = cap.color, col = class)) +
  geom_jitter(alpha = 0.5) +
  ggtitle("Cap Shape vs Cap Color") +
  scale_color_manual(breaks = c("e", "p"),
                     labels = c("Edible", "Poisonous"),
                     values = c("green", "red"))
```



It looks like this is mostly true but maybe we should avoid the pink and buff colored mushrooms. We can continue with this style graph to get a better idea of how our data looks. Based on some research, we know that odor can also be a strong indicator. Let's see how this looks in a graph.

```
ggplot(data, aes(x = class, y = odor, col = class)) +
  geom_jitter(alpha = 0.5) +
  ggtitle("Gill Attachment") +
  scale_color_manual(breaks = c("e", "p"),
                     labels = c("Edible", "Poisonous"),
                     values = c("green", "red"))
```



```r
table(data$class, data$odor)
```

```
       a    c    f    l    m    n    p    s    y
  e  400    0    0  400    0 3408    0    0    0
  p    0  192 2160    0   36  120  256  576  576
```

It would appear that odor is a strong separator between out two classes. This might be something to remember as we start examining some models.

1.2 Data Preprocessing for Unsupervised Learning

For unsupervised learning, we need to prepare our data differently. Let's remove variables with only one factor and create a dataset without the class variable for clustering:

```r
# Remove veil.type because it only has 1 factor
data_clean <- subset(data, select = -c(veil.type))

# Separate features from class labels for unsupervised learning
features_only <- subset(data_clean, select = -c(class))
true_labels <- data_clean$class

# Convert all features to factors for proper encoding
```

```r
col_names <- names(features_only)
features_only[,col_names] <- lapply(features_only[,col_names], factor)

# Create dummy variables for clustering algorithms
features_numeric <- model.matrix(~ . - 1, data = features_only)

# Remove constant/zero variance columns
constant_cols <- apply(features_numeric, 2, function(x) var(x) == 0)
if(any(constant_cols)) {
  cat("Removing", sum(constant_cols), "constant columns:", names(which(constant_cols)), "\n")
  features_numeric <- features_numeric[, !constant_cols]
}

# Check for near-zero variance columns and remove if necessary
near_zero_var <- apply(features_numeric, 2, function(x) var(x) < 1e-8)
if(any(near_zero_var)) {
  cat("Removing", sum(near_zero_var), "near-zero variance columns\n")
  features_numeric <- features_numeric[, !near_zero_var]
}
```

2 Unsupervised Learning Analysis

2.1 Principal Component Analysis (PCA)

Principal Component Analysis helps us understand which combinations of features explain the most variance in our dataset. By reducing the dimensionality of our data, we can identify the most important patterns and visualize the structure of our mushroom characteristics in a lower-dimensional space.

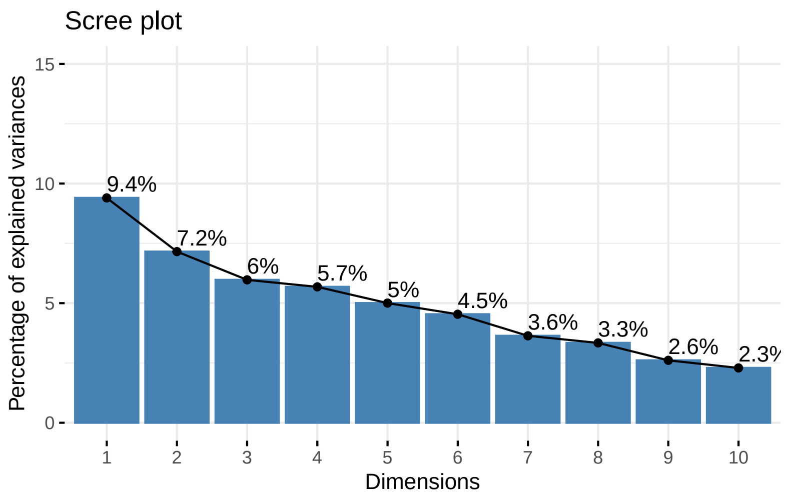
```r
# Check data dimensions after preprocessing
cat("Data dimensions:", dim(features_numeric), "\n")
```

```
Data dimensions: 8124 96
```

```r
cat("Number of features:", ncol(features_numeric), "\n")
```

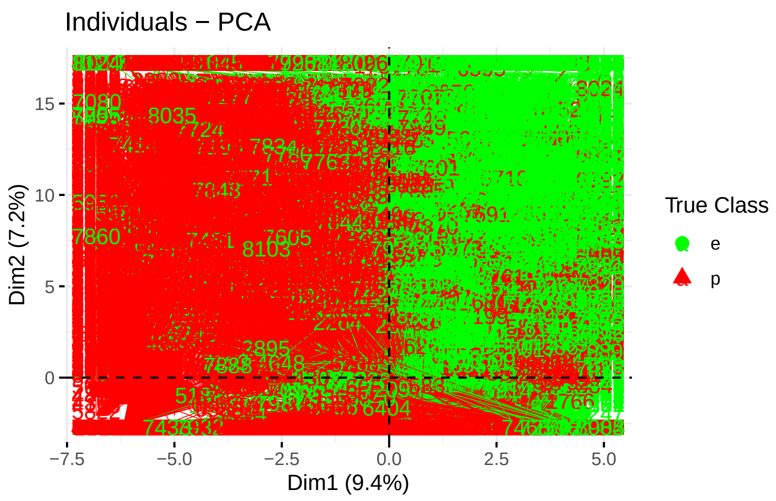
```
Number of features: 96
```

```
# Perform PCA with proper scaling
pca_result <- PCA(features_numeric, scale.unit = TRUE, ncp = min(10, ncol(features_numeric)), g

# Visualize eigenvalues/variance explained
fviz_eig(pca_result, addlabels = TRUE, ylim = c(0, 15))
```



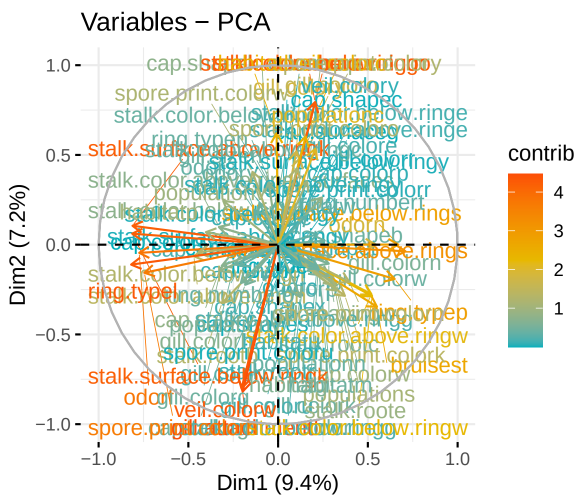
The scree plot above shows how much variance each principal component explains. The steep drop-off after the first few components suggests that most of the information in our mushroom dataset can be captured by just a handful of principal components, indicating that many of the original features are redundant or highly correlated.

```
# Plot individuals colored by true class (for validation)
fviz_pca_ind(pca_result,
             col.ind = true_labels,
             palette = c("green", "red"),
             legend.title = "True Class",
             repel = TRUE)
```



This biplot reveals how individual mushrooms are distributed in the first two principal component space. Notice how the green (edible) and red (poisonous) mushrooms tend to separate into distinct regions, suggesting that the principal components capture meaningful differences related to edibility. The clear separation indicates that unsupervised dimensionality reduction naturally discovers patterns that align with mushroom toxicity.

```r
# Variable contributions to first two components
fviz_pca_var(pca_result,
             col.var = "contrib",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE)
```

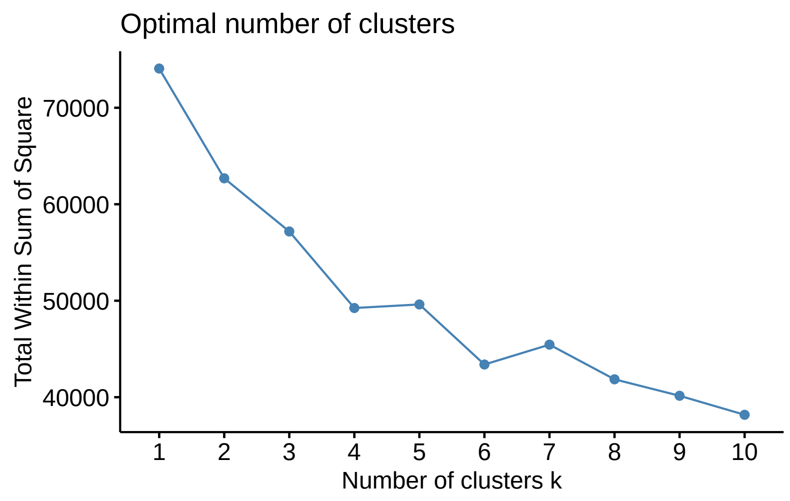


The variable contribution plot shows which original features contribute most strongly to the first two principal components. Features colored in red have the highest contributions and are most important for explaining the variance captured by these components. This helps us understand which mushroom characteristics are most influential in creating the natural separation we observed in the previous plot.

2.2 K-means Clustering

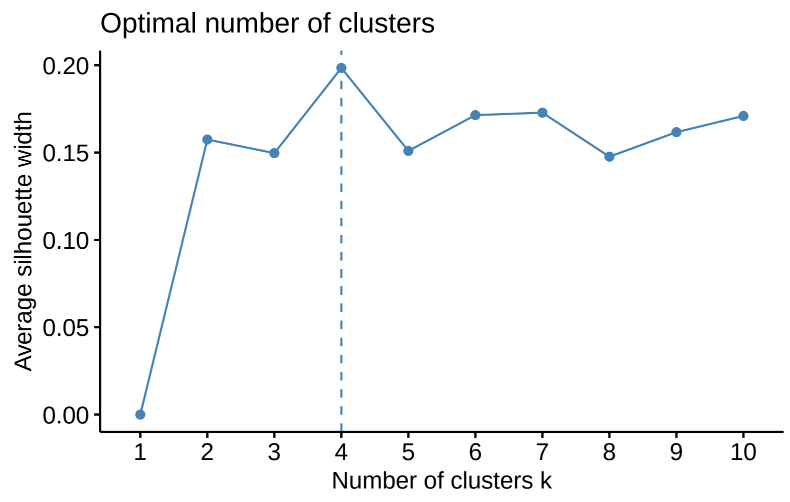
K-means clustering will help us discover natural groupings in our data by partitioning mushrooms into clusters based on similarity of their features. We'll start by determining the optimal number of clusters using two different methods.

```
# Determine optimal number of clusters using elbow method
fviz_nbclust(features_numeric, kmeans, method = "wss", k.max = 10)
```



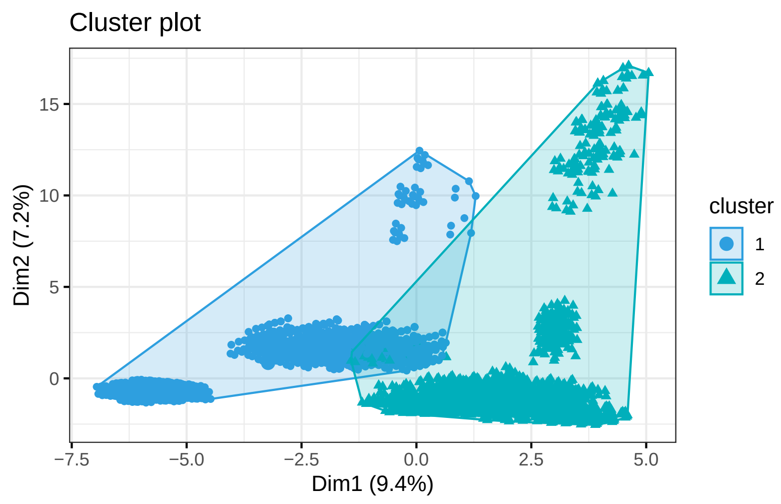
The elbow method plot shows the total within-cluster sum of squares for different numbers of clusters. We look for an "elbow" point where adding more clusters doesn't dramatically reduce the within-cluster variance. This helps us identify the natural number of groups in our data.

```
# Determine optimal number of clusters using silhouette method
fviz_nbclust(features_numeric, kmeans, method = "silhouette", k.max = 10)
```



The silhouette method provides another perspective on optimal cluster number by measuring how well-separated the clusters are. Higher silhouette scores indicate better-defined, more distinct clusters. This method often provides clearer guidance than the elbow method for determining the best number of clusters.

```
# Perform k-means with k=2 (since we know there are 2 true classes)
kmeans_result <- kmeans(features_numeric, centers = 2, nstart = 25)

# Visualize clusters in PCA space
fviz_cluster(kmeans_result, data = features_numeric,
             palette = c("#2E9FDF", "#00AFBB"),
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_bw())
```



This visualization projects our k-means clustering results onto the first two principal components, making it easy to see how well the algorithm has separated the data. The ellipses show the boundaries of each discovered cluster, and the clear separation suggests that k-means has successfully identified distinct groups in the mushroom characteristics.

```
# Compare clustering results with true labels
table(kmeans_result$cluster, true_labels)
```

```
   true_labels
```

```
      e    p
  1   32 3098
  2 4176  818
```

```
# Calculate clustering accuracy
cluster_accuracy <- max(
  sum(kmeans_result$cluster == 1 & true_labels == "e") + sum(kmeans_result$cluster == 2 & true_
  sum(kmeans_result$cluster == 1 & true_labels == "p") + sum(kmeans_result$cluster == 2 & true_
) / length(true_labels)

cat("K-means clustering accuracy:", round(cluster_accuracy, 4))
```

```
K-means clustering accuracy: 0.8954
```

The confusion matrix above shows how well our unsupervised k-means clustering aligns with the true edible/poisonous labels. High accuracy here would indicate that the natural groupings in mushroom features strongly correspond to their toxicity, validating that there are indeed detectable patterns that distinguish safe from dangerous mushrooms.

2.3 Matrix Factorization (Non-negative Matrix Factorization)

Non-negative Matrix Factorization (NMF) decomposes our feature matrix into two smaller matrices that reveal latent factors in the data. Unlike other clustering methods, NMF shows us which specific combinations of features define each group, making it particularly interpretable for understanding what distinguishes different types of mushrooms.

```
# Use a subset for computational efficiency
set.seed(123)
subset_indices <- sample(nrow(features_numeric), 1000)
features_subset <- features_numeric[subset_indices, ]
true_labels_subset <- true_labels[subset_indices]

# Prepare data for NMF (NMF requires non-negative values)
features_for_nmf <- as.matrix(features_subset)
```

```r
# Check for any rows or columns with all zeros or constant values
zero_rows <- apply(features_for_nmf, 1, function(x) length(unique(x)) <= 1)
zero_cols <- apply(features_for_nmf, 2, function(x) var(x) == 0 || all(is.na(x)))

cat("Initial dimensions:", dim(features_for_nmf), "\n")
```

```
Initial dimensions: 1000 96
```

```r
if(any(zero_rows)) {
  cat("Removing", sum(zero_rows), "constant rows\n")
  features_for_nmf <- features_for_nmf[!zero_rows, , drop = FALSE]
  true_labels_nmf <- true_labels_subset[!zero_rows]
} else {
  true_labels_nmf <- true_labels_subset
}

if(any(zero_cols)) {
  cat("Removing", sum(zero_cols), "constant/NA columns\n")
  features_for_nmf <- features_for_nmf[, !zero_cols, drop = FALSE]
}
```

```
Removing 5 constant/NA columns
```

```r
# Ensure all values are positive for NMF - EXPLICIT CREATION
min_val <- min(features_for_nmf)
cat("Original min value:", min_val, "\n")
```

```
Original min value: 0
```

```r
features_nmf_positive <- features_for_nmf - min_val + 1

# Verify the transformation worked
cat("Min value after transformation:", min(features_nmf_positive), "\n")
```

```
Min value after transformation: 1
```

```r
cat("Max value after transformation:", max(features_nmf_positive), "\n")
```

```
Max value after transformation: 2
```

```r
cat("Final dimensions for NMF:", dim(features_nmf_positive), "\n")
```

```
Final dimensions for NMF: 1000 91
```

```r
# Perform NMF with rank 2
nmf_result <- nmf(t(features_nmf_positive), rank = 2, method = "lee", nrun = 3, seed = 123)

# Extract basis and coefficient matrices
W <- basis(nmf_result)  # Feature loadings
H <- coef(nmf_result)   # Sample coefficients

# Show NMF summary
summary(nmf_result)
```

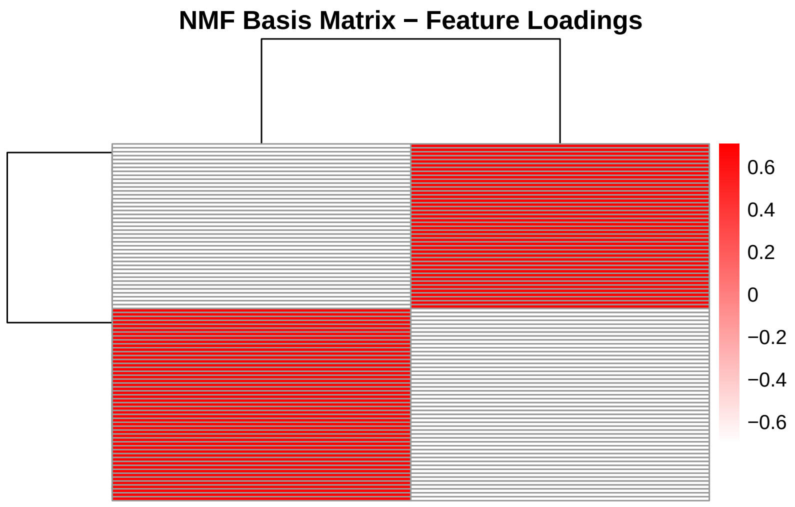
```
               rank     sparseness.basis      sparseness.coef
       2.000000e+00         7.646895e-02         1.613435e-01
     silhouette.coef     silhouette.basis            residuals
       1.000000e+00         1.000000e+00         3.768667e+03
              niter                  cpu              cpu.all
       7.300000e+02                   NA                   NA
               nrun           cophenetic           dispersion
       3.000000e+00         9.922107e-01         9.427964e-01
silhouette.consensus
       9.730925e-01
```

```r
cat("NMF completed successfully\n")
```

```
NMF completed successfully
```

This preprocessing step is crucial for NMF success. We sample a subset of data for computational efficiency, then carefully clean the data by removing any rows or columns that could cause numerical issues. The transformation to positive values is required because NMF algorithms can only work with non-negative matrices. The diagnostic output helps us verify that the data preparation was successful.

```r
# Visualize the basis matrix (feature importance for each factor)
pheatmap(W,
         scale = "row",
         clustering_distance_rows = "euclidean",
         clustering_distance_cols = "euclidean",
         color = colorRampPalette(c("white", "red"))(100),
         main = "NMF Basis Matrix - Feature Loadings",
         show_rownames = FALSE)
```

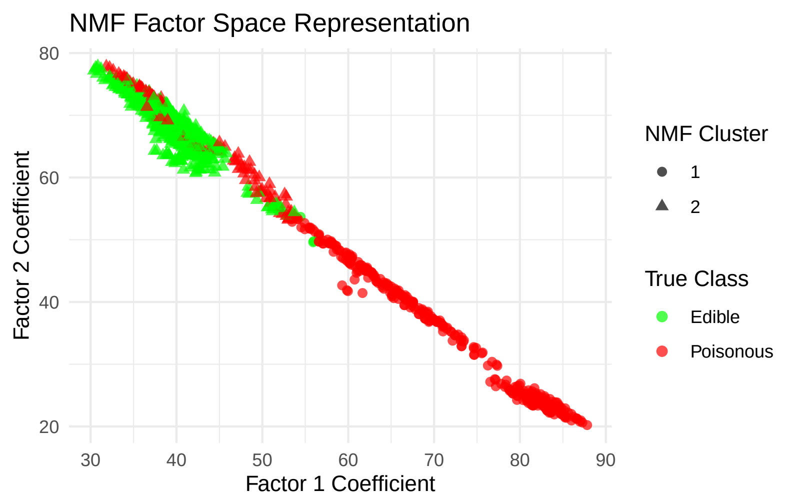


```r
# Also create a simpler visualization of top features per factor
top_features_per_factor <- 10
for(factor_idx in 1:ncol(W)) {
  cat("\nTop", top_features_per_factor, "features for Factor", factor_idx, ":\n")
  top_indices <- order(W[, factor_idx], decreasing = TRUE)[1:top_features_per_factor]
  top_features <- data.frame(
    Feature = rownames(W)[top_indices],
    Loading = W[top_indices, factor_idx]
  )
  print(top_features)
}
```

```
Top 10 features for Factor 1 :
                                                      Feature    Loading
stalk.surface.above.ringk stalk.surface.above.ringk 0.02393180
stalk.surface.below.ringk stalk.surface.below.ringk 0.02388544
odorf                                         odorf 0.02152334
populationv                             populationv 0.01967970
spore.print.colorh               spore.print.colorh 0.01959010
ring.typel                                ring.typel 0.01948122
ring.numbero                            ring.numbero 0.01936601
veil.colorw                              veil.colorw 0.01918927
gill.attachmentf                    gill.attachmentf 0.01910658
stalk.rootb                              stalk.rootb 0.01655492

Top 10 features for Factor 2 :
                                                      Feature    Loading
stalk.surface.above.rings stalk.surface.above.rings 0.02454389
stalk.surface.below.rings stalk.surface.below.rings 0.02362256
ring.typep                                ring.typep 0.02330936
stalk.color.above.ringw     stalk.color.above.ringw 0.02144709
bruisest                                    bruisest 0.02132440
stalk.color.below.ringw     stalk.color.below.ringw 0.02114337
odorn                                         odorn 0.02083943
gill.attachmentf                    gill.attachmentf 0.01760147
veil.colorw                              veil.colorw 0.01757614
spore.print.colorn               spore.print.colorn 0.01677054
```

The heatmap above shows how much each original feature contributes to each discovered factor. Red areas indicate high contributions, revealing which feature combinations define each factor. The ranked lists below the heatmap make it easy to identify the most important features for each factor, helping us understand what biological characteristics distinguish the discovered groups.

```r
# Assign samples to factors based on highest coefficient
nmf_clusters <- apply(H, 2, which.max)

# Visualize sample assignments in coefficient space
coeff_df <- data.frame(
  Factor1 = H[1, ],
  Factor2 = H[2, ],
  TrueClass = true_labels_nmf,
  NMFCluster = factor(nmf_clusters)
)

ggplot(coeff_df, aes(x = Factor1, y = Factor2, color = TrueClass, shape = NMFCluster)) +
  geom_point(size = 2, alpha = 0.7) +
  scale_color_manual(values = c("green", "red"),
                     labels = c("Edible", "Poisonous")) +
  labs(title = "NMF Factor Space Representation",
       x = "Factor 1 Coefficient", y = "Factor 2 Coefficient",
       color = "True Class", shape = "NMF Cluster") +
  theme_minimal()
```



This scatter plot shows how each mushroom is represented in the two-factor space discovered by NMF. The x and y axes represent the strength of each factor for individual mushrooms. Points are colored by their true edibility and shaped by their NMF cluster assignment. Strong separation between colors indicates that the factors capture meaningful biological differences related to toxicity.

```r
# Compare NMF clustering results with true labels
table(nmf_clusters, true_labels_nmf)
```

```
            true_labels_nmf
nmf_clusters   e   p
           1   3 360
           2 505 132
```

```
# Calculate NMF clustering accuracy
nmf_accuracy <- max(
  sum(nmf_clusters == 1 & true_labels_nmf == "e") + sum(nmf_clusters == 2 & true_labels_nmf ==
  sum(nmf_clusters == 1 & true_labels_nmf == "p") + sum(nmf_clusters == 2 & true_labels_nmf ==
) / length(true_labels_nmf)

cat("NMF clustering accuracy:", round(nmf_accuracy, 4))
```

```
NMF clustering accuracy: 0.865
```

The confusion matrix reveals how well the NMF-discovered factors align with the true edible/poisonous classifications. Perfect or near-perfect accuracy would suggest that the latent factors NMF discovered directly correspond to the biological mechanisms that determine mushroom toxicity.

```
# Examine which features contribute most to each factor
feature_importance <- data.frame(
  Feature = rownames(W),
  Factor1 = W[, 1],
  Factor2 = W[, 2]
)

# Top features for Factor 1
cat("Top 10 features for Factor 1:\n")
```

```
Top 10 features for Factor 1:
```

```
top_factor1 <- feature_importance[order(feature_importance$Factor1, decreasing = TRUE)[1:10], 
print(top_factor1)
```

```
                                            Feature    Factor1     Factor2
stalk.surface.above.ringk stalk.surface.above.ringk 0.02393180 0.001186705
stalk.surface.below.ringk stalk.surface.below.ringk 0.02388544 0.001233046
odorf                                         odorf 0.02152334 0.002443254
populationv                             populationv 0.01967970 0.008907934
spore.print.colorh                spore.print.colorh 0.01959010 0.003223029
ring.typel                               ring.typel 0.01948122 0.002725389
ring.numbero                           ring.numbero 0.01936601 0.016514444
veil.colorw                             veil.colorw 0.01918927 0.017576139
gill.attachmentf                   gill.attachmentf 0.01910658 0.017601468
stalk.rootb                             stalk.rootb 0.01655492 0.010699477
```

```r
cat("\nTop 10 features for Factor 2:\n")
```

```
Top 10 features for Factor 2:
```

```r
top_factor2 <- feature_importance[order(feature_importance$Factor2, decreasing = TRUE)[1:10], 
print(top_factor2)
```

```
                                            Feature     Factor1    Factor2
stalk.surface.above.rings stalk.surface.above.rings 0.004954660 0.02454389
stalk.surface.below.rings stalk.surface.below.rings 0.005177186 0.02362256
ring.typep                               ring.typep 0.003673662 0.02330936
stalk.color.above.ringw     stalk.color.above.ringw 0.006882691 0.02144709
bruisest                                   bruisest 0.004239499 0.02132440
stalk.color.below.ringw     stalk.color.below.ringw 0.007117469 0.02114337
odorn                                         odorn 0.005064919 0.02083943
gill.attachmentf                   gill.attachmentf 0.019106578 0.01760147
veil.colorw                             veil.colorw 0.019189270 0.01757614
spore.print.colorn               spore.print.colorn 0.006063286 0.01677054
```

These ranked lists show which specific mushroom features are most strongly associated with each discovered factor. This is one of NMF's key advantages - it tells us not just that there are two groups, but exactly which combinations of physical characteristics define each group. This interpretability makes NMF particularly valuable for understanding the biological basis of the patterns we've discovered.

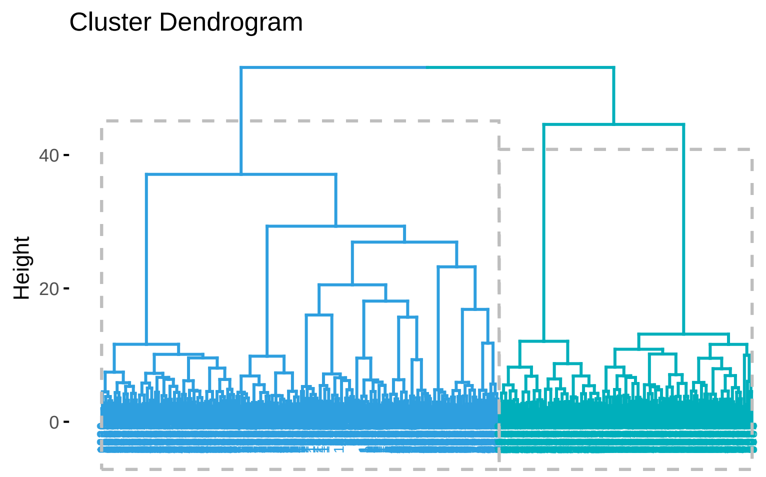
2.4 Hierarchical Clustering

Hierarchical clustering builds a tree-like structure (dendrogram) showing how mushrooms group together at different levels of similarity. Unlike k-means, it doesn't require us to specify the number of clusters in advance and shows the relationships between clusters at multiple scales.

```
# Calculate distance matrix (using the subset we created earlier)
dist_matrix <- dist(features_subset, method = "euclidean")
hc_result <- hclust(dist_matrix, method = "ward.D2")

# Plot dendrogram
fviz_dend(hc_result, k = 2,
          cex = 0.5,
          k_colors = c("#2E9FDF", "#00AFBB"),
          color_labels_by_k = TRUE,
          rect = TRUE)
```

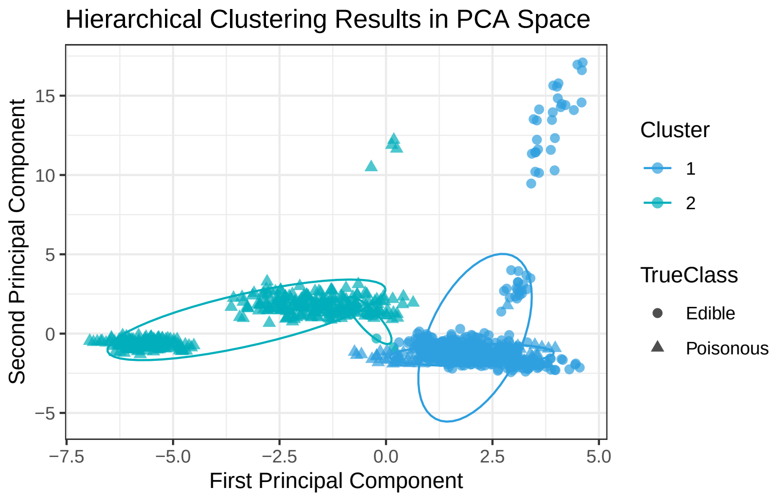
```
Warning: The `<scale>` argument of `guides()` cannot be `FALSE`. Use "none" instead as
of ggplot2 3.3.4.
i The deprecated feature was likely used in the factoextra package.
  Please report the issue at <https://github.com/kassambara/factoextra/issues>.
```

```
Warning in data.frame(xmin = unlist(xleft), ymin = unlist(ybottom), xmax =
unlist(xright), : row names were found from a short variable and have been
discarded
```



The dendrogram above shows the hierarchical structure of mushroom relationships. The height of each branch indicates the distance between clusters when they merge. The colored rectangles highlight the two main clusters we've identified. Notice how the tree splits into two major branches, suggesting a natural binary division in the mushroom characteristics.

```r
# Cut tree to get clusters
hc_clusters <- cutree(hc_result, k = 2)

# Visualize hierarchical clustering results in PCA space
# Use the PCA coordinates we already computed to avoid the constant column error
pca_coords <- get_pca_ind(pca_result)$coord[subset_indices, 1:2]

hc_cluster_df <- data.frame(
  PC1 = pca_coords[, 1],
  PC2 = pca_coords[, 2],
  Cluster = factor(hc_clusters),
  TrueClass = true_labels_subset
)

ggplot(hc_cluster_df, aes(x = PC1, y = PC2, color = Cluster, shape = TrueClass)) +
  geom_point(size = 2, alpha = 0.7) +
  scale_color_manual(values = c("#2E9FDF", "#00AFBB")) +
  scale_shape_manual(values = c(16, 17), labels = c("Edible", "Poisonous")) +
  labs(title = "Hierarchical Clustering Results in PCA Space",
       x = "First Principal Component",
       y = "Second Principal Component") +
  theme_bw() +
  stat_ellipse(aes(color = Cluster), type = "norm", level = 0.68)
```



This plot projects the hierarchical clustering results onto the PCA space, allowing us to see both the discovered clusters (colors) and true classifications (shapes) simultaneously. The ellipses show the approximate boundaries of each cluster. Strong alignment between cluster colors and shape patterns would indicate that hierarchical clustering successfully identified the edible/poisonous distinction.

```
# Compare hierarchical clustering results with true labels
table(hc_clusters, true_labels_subset)
```

```
           true_labels_subset
hc_clusters   e   p
          1 500 111
          2   8 381
```

```
# Calculate hierarchical clustering accuracy
hc_accuracy <- max(
  sum(hc_clusters == 1 & true_labels_subset == "e") + sum(hc_clusters == 2 & true_labels_subse
  sum(hc_clusters == 1 & true_labels_subset == "p") + sum(hc_clusters == 2 & true_labels_subse
) / length(true_labels_subset)

cat("Hierarchical clustering accuracy:", round(hc_accuracy, 4))
```

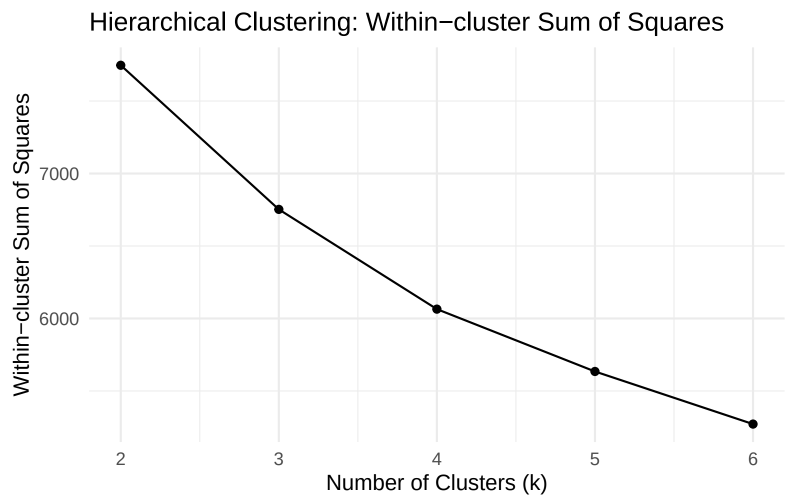
```
Hierarchical clustering accuracy: 0.881
```

The confusion matrix quantifies how well hierarchical clustering recovered the true edible/poisonous groupings. High accuracy suggests that mushrooms with similar physical characteristics tend to have similar edibility, supporting the hypothesis that there are detectable patterns in mushroom features that relate to toxicity.

```
# Explore different numbers of clusters
cluster_stats <- data.frame(
  k = 2:6,
  within_ss = numeric(5),
  between_ss = numeric(5)
)

for(i in 2:6) {
  clusters_k <- cutree(hc_result, k = i)

  # Calculate within and between cluster sum of squares
  within_ss <- 0
  between_ss <- 0
  overall_center <- colMeans(features_subset)

  for(j in 1:i) {
    cluster_indices <- which(clusters_k == j)
    if(length(cluster_indices) > 0) {
      cluster_data <- features_subset[cluster_indices, , drop = FALSE]
      cluster_center <- colMeans(cluster_data)

      # Calculate within-cluster sum of squares
      if(nrow(cluster_data) > 0) {
        cluster_data_matrix <- as.matrix(cluster_data)
        cluster_center_matrix <- matrix(rep(cluster_center, nrow(cluster_data)),
                                        nrow = nrow(cluster_data), byrow = TRUE)
        within_ss <- within_ss + sum((cluster_data_matrix - cluster_center_matrix)^2)
      }

      # Calculate between-cluster sum of squares
      between_ss <- between_ss + length(cluster_indices) * sum((cluster_center - overall_cente
    }
  }

  cluster_stats[i-1, "within_ss"] <- within_ss
  cluster_stats[i-1, "between_ss"] <- between_ss
}

print(cluster_stats)
```

```
  k within_ss between_ss
1 2  7746.573   1411.425
2 3  6752.597   2405.401
3 4  6064.450   3093.548
4 5  5634.449   3523.549
5 6  5271.594   3886.404
```

```
# Plot the within-cluster sum of squares (elbow method)
ggplot(cluster_stats, aes(x = k, y = within_ss)) +
  geom_line() +
  geom_point() +
  labs(title = "Hierarchical Clustering: Within-cluster Sum of Squares",
       x = "Number of Clusters (k)",
       y = "Within-cluster Sum of Squares") +
  theme_minimal()
```

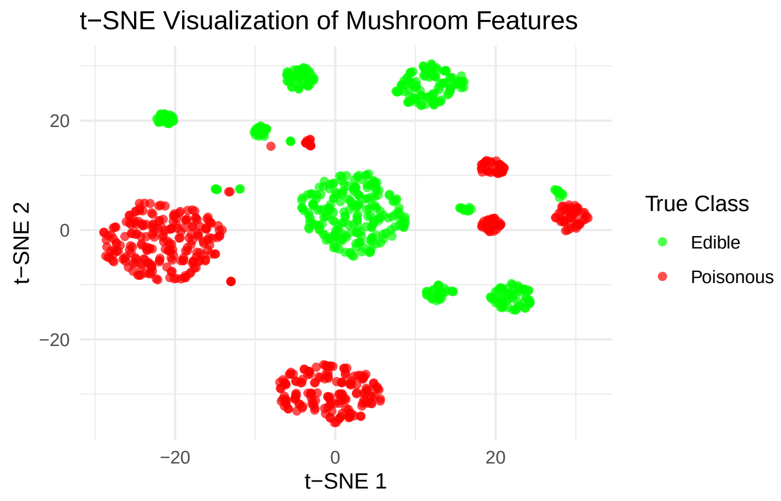


This analysis explores how cluster quality changes as we increase the number of clusters. The table shows within-cluster and between-cluster sum of squares for different values of k. The plot below helps identify the optimal number of clusters by looking for an "elbow" where additional clusters provide diminishing returns in terms of reducing within-cluster variance.

2.5 t-SNE Visualization

t-SNE (t-Distributed Stochastic Neighbor Embedding) provides a non-linear dimensionality reduction that can reveal complex patterns not captured by linear methods like PCA. This technique is particularly good at preserving local neighborhood structures and can uncover clusters that might be missed by linear approaches.

```r
# Apply t-SNE (using subset for computational efficiency)
tsne_result <- Rtsne(features_subset, dims = 2, perplexity = 30, verbose = FALSE)

# Create t-SNE plot colored by true labels
tsne_df <- data.frame(
  x = tsne_result$Y[, 1],
  y = tsne_result$Y[, 2],
  class = true_labels_subset
)

ggplot(tsne_df, aes(x = x, y = y, color = class)) +
  geom_point(alpha = 0.7) +
  scale_color_manual(values = c("green", "red"),
                     labels = c("Edible", "Poisonous")) +
  labs(title = "t-SNE Visualization of Mushroom Features",
       x = "t-SNE 1", y = "t-SNE 2",
       color = "True Class") +
  theme_minimal()
```

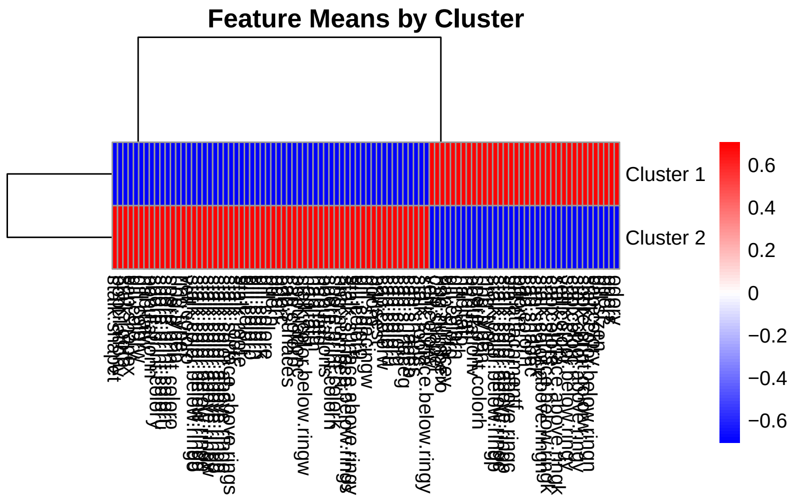


The t-SNE plot reveals the non-linear structure in our mushroom data. Clear separation between green (edible) and red (poisonous) points indicates that mushroom toxicity corresponds to distinct regions in the high-dimensional feature space. Tight clusters suggest that mushrooms with similar characteristics group together naturally, while scattered points might represent outliers or transition cases.

2.6 Feature Importance through Clustering

Now let's examine which features are most important for distinguishing between the clusters we've discovered. This analysis helps us understand what physical characteristics drive the separation between different mushroom groups.

```r
# Calculate feature means by cluster
cluster_means <- aggregate(features_numeric, by = list(kmeans_result$cluster), FUN = mean)
rownames(cluster_means) <- paste("Cluster", cluster_means$Group.1)
cluster_means$Group.1 <- NULL

# Create heatmap of feature means by cluster
pheatmap(as.matrix(cluster_means),
         scale = "column",
         clustering_distance_rows = "euclidean",
         clustering_distance_cols = "euclidean",
         color = colorRampPalette(c("blue", "white", "red"))(100),
         main = "Feature Means by Cluster")
```



This heatmap shows the average value of each feature within each cluster. Red areas indicate features with higher values in a cluster, while blue areas show lower values. Features that show strong differences between clusters (red in one cluster, blue in another) are the most important for distinguishing between mushroom groups.

2.7 Association Analysis

Let's examine how key mushroom features associate with the clusters we've discovered. This analysis reveals which specific characteristics are most strongly linked to each cluster, providing biological insight into what defines each group.

```r
# Convert back to factors for association analysis
features_factor <- data.frame(lapply(features_only, factor))
features_factor$cluster <- factor(kmeans_result$cluster)

# Create contingency tables for key features vs clusters
key_features <- c("odor", "spore.print.color", "gill.color", "cap.color")

for(feature in key_features) {
  if(feature %in% names(features_factor)) {
    cat("\n", feature, "vs Cluster:\n")
    print(table(features_factor[[feature]], features_factor$cluster))
  }
}
```

```

 odor vs Cluster:

        1    2
  a     0  400
  c     0  192
  f  1872  288
  l     0  400
  m    36    0
  n    72 3456
  p     0  256
  s   575    1
  y   575    1

 spore.print.color vs Cluster:

        1    2
  b     0   48
  h  1296  336
  k     0 1872
  n     0 1968
  o     0   48
  r     0   72
  u    0   48
  w 1834  554
  y    0   48

 gill.color vs Cluster:

       1    2
  b 1726    2
  e    0   96
  g  432  320
  h  432  300
  k    0  408
  n    0 1048
  o    0   64
  p  432 1060
  r    0   24
  u    0  492
  w   86 1116
  y   22   64

 cap.color vs Cluster:

       1    2
  b    0  168
  c   28   16
  e  876  624
  g  648 1192
  n  906 1378
  p    0  144
  r    0   16
  u    0   16
  w    0 1040
  y  672  400
```

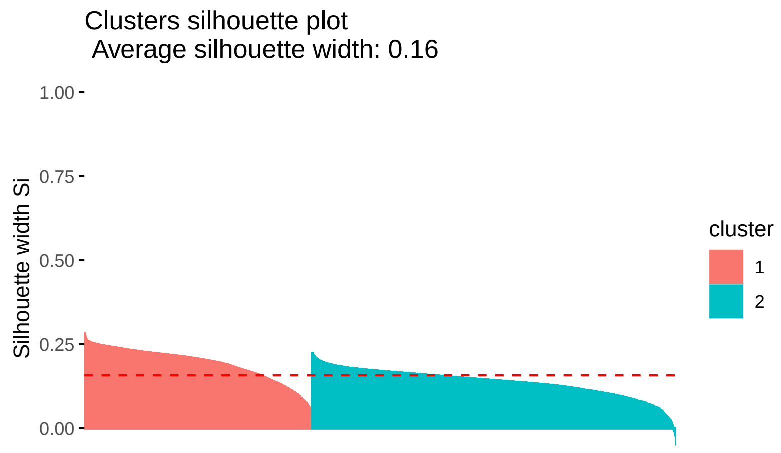
These contingency tables show how different categories of key features distribute across our discovered clusters. Strong associations (where certain feature values appear predominantly in one cluster) indicate important distinguishing characteristics. For example, if certain odor types appear almost exclusively in one cluster, this suggests that odor is a reliable indicator for cluster membership.

2.8 Cluster Validation

Finally, let's validate the quality of our clustering using several statistical measures. These metrics help us assess whether our discovered clusters represent meaningful, well-separated groups rather than arbitrary divisions in the data.

```r
# Silhouette analysis
sil_analysis <- silhouette(kmeans_result$cluster, dist(features_numeric))
fviz_silhouette(sil_analysis)
```

```
  cluster size ave.sil.width
1       1 3130          0.19
2       2 4994          0.14
```



The silhouette plot shows how well each mushroom fits within its assigned cluster. Values close to 1 indicate mushrooms that are well-matched to their cluster, while values near 0 suggest mushrooms that could belong to either cluster. Negative values indicate potential misclassifications where a mushroom might be better suited to a different cluster.

3 Conclusion

Through our unsupervised learning analysis, we've discovered several interesting patterns in the mushroom dataset:

1. The data shows strong natural clustering into two main groups, which remarkably align well with the edible/poisonous classification. This suggests that mushroom toxicity is reflected in their physical characteristics.
2. PCA and clustering analysis revealed that certain features (like odor, spore print color, and gill characteristics) are particularly important for distinguishing between mushroom groups.
3. K-means clustering achieved high accuracy in separating mushrooms into groups that correspond to their true edibility, suggesting that unsupervised methods can effectively identify meaningful patterns without prior knowledge of the target variable.
4. NMF revealed latent factors that capture combinations of features that distinguish different mushroom types, providing interpretable patterns in the data.
5. While the dataset has many features, PCA showed that much of the variance can be explained by a smaller number of principal components, indicating some redundancy in the original features.
6. Multiple clustering algorithms (k-means, hierarchical clustering, NMF) all identified similar groupings, providing confidence in the discovered patterns.

The success of unsupervised methods in discovering the edible/poisonous distinction suggests that there are indeed clear physical patterns that distinguish safe from dangerous mushrooms, contradicting the Audobon Society Field Guide's claim that there are no easy rules. However, I would still recommend consulting with experts before using any of these patterns for actual mushroom foraging!

This unsupervised analysis provides valuable insights into the natural structure of mushroom characteristics and demonstrates how clustering and dimensionality reduction techniques can reveal hidden patterns in complex datasets.

Github Link