# Compositional data exploration & preprocessing

Bram Duthoo

2025-01-03

# Preprocessing

## Inladen data & libraries

Om te beginnen worden alle nodige libraries & data ingeladen. Recipes.db is een database gevormd op basis van recepten in patenten. De data wordt omgezet naar een matrix waarin elke rij een recept voorstelt en elke kolom een ingrediënt categorie.

```
# 1.1) Libraries
library(DBI)
library(RSQLite)
library(tidyverse)
library(compositions) # acomp, clr, ilr etc.
library(MASS)        # lda()
library(ggplot2)
library(ggfortify)   # Voor evt. autoplot(pca, data=...) als je wilt
library(dplyr)
library(tidyr)

# 1.2) Database inlezen
db <- dbConnect(RSQLite::SQLite(),
                "C:/Users/bramd_finhsgu/OneDrive - UGent/Thesis/Thesis_bestanden/recipes9.d
b")

recipes            <- dbReadTable(db, "recipes")
chemical_groups    <- dbReadTable(db, "chemical_groups")
ingredients        <- dbReadTable(db, "ingredients")
recipe_ingredients <- dbReadTable(db, "recipe_ingredients")

# 1.3) Samenvoegen (pas code indien nodig)
data_grouped <- recipe_ingredients %>%
  left_join(ingredients, by = c("ingredient_id" = "id")) %>%  # Voeg ingrediënten toe
  left_join(chemical_groups, by = c("chemical_group_id" = "id")) %>%  # Voeg chemische groepe
n toe
  group_by(recipe_id, group_name) %>%
  summarise(total_quantity = sum(total_quantity, na.rm = TRUE), .groups = "drop") %>%
  pivot_wider(names_from = group_name, values_from = total_quantity, values_fill = 0) %>%
  # LET OP: we gebruiken expliciet dplyr::select(...) om conflicts te vermijden
  left_join(
    recipes %>% dplyr::select(id, recipe_name, cooking_time, temperature, flavor_label, prepa
ration),
    by = c("recipe_id" = "id")
  ) %>%
  # Ook hier expliciet dplyr::select(...):
  dplyr::select(recipe_name, cooking_time, temperature, flavor_label, preparation, everything
(), -recipe_id)
```

Op basis hiervan maken we de objecten die nodig zijn voor de preprocessing

```
# 1.4) Matrix & flavor_labels
composition_matrix <- data_grouped %>%
  dplyr::select(-recipe_name, -cooking_time, -temperature, -preparation, -flavor_label) %>%
  as.matrix()

rownames(composition_matrix) <- data_grouped$recipe_name
flavor_labels <- data_grouped$flavor_label

# DB niet meer nodig:
dbDisconnect(db)

# 1.5) Bepaal water_col & vet_col
water_col <- "Water"     # pas aan
vet_col   <- "Vetten"    # pas aan
epsilon   <- 1e-5

# 1.6) Maak A1, A2, B1 objecten

# A1: water+vet apart (geen filtering)
A1 <- composition_matrix

# A2: water+vet => carrier (geen filtering)
A2 <- composition_matrix
if(!all(c(water_col, vet_col) %in% colnames(A2))){
  stop("Kolommen water/vet niet gevonden in composition_matrix.")
}
carrier_vec <- A2[, water_col] + A2[, vet_col]
A2 <- cbind(A2, carrier = carrier_vec)
A2 <- A2[, !(colnames(A2) %in% c(water_col, vet_col)), drop=FALSE]

# B1: voorbeeld van 'carrier' met gefilterde kolommen (irrelevante)
# (We doen het filtering meteen.)
irrelevant_cols <- c("Zuren","Volatiles","extra","Smaakenhancer") # pas aan
B1 <- A2[, !(colnames(A2) %in% irrelevant_cols), drop=FALSE]
```

# Ingredient categoriën reduceren

Minder categoriën om resultaten interpreteerbaarder te maken, minder ruis. Vanuit patenten weten we toch dat enkel sommige categoriën relevant zijn voor de smaakontwikkeling, dus enkel hierop willen we analyses uitvoeren. Helpt ook bij compositionele data sinds catgorien met veel nullen analyses moeilijk maken (logtransformaties)

In de patenten worden water en vetten primair beschreven als carriers van smaken, zonder specifieke aandacht voor hun unieke chemische bijdragen aan smaakontwikkeling. Vooral vetten worden vaak veralgemeend als “vegetable oil” of “animal fat,” waardoor gedetailleerde informatie over hun chemische samenstelling verloren gaat. Het samenvoegen in een enkele categorie (“carrier”) weerspiegelt deze praktische realiteit en minimaliseert onnodige ruis in de data-analyse. Daarnaast werd in patenten vermeld dat sommige categoriën niet echt bijdragen aan het creëren van de vleessmaken. Het elimineren van deze categoriën weerspiegelt de praktische realiteit.

# Water en vet categorie samenvoegen

In compositional data, zeros are problematic because of the log-ratio transformation (e.g., ILR or CLR). To address this, we replace zeros with a small value ( `epsilon` ) to ensure valid transformations. After pseudocounting, the data is normalized by row (recipe total = 1) to reflect compositional constraints. ILR (Isometric Log-Ratio) transformations are used to map the normalized data into an orthogonal space suitable for PCA and discriminant analysis.

```
# A1: water + vet apart
A1_pseudo <- A1
A1_pseudo[A1_pseudo == 0] <- epsilon
row_sums_A1 <- rowSums(A1_pseudo)
A1_norm <- sweep(A1_pseudo, 1, row_sums_A1, "/")

acompA1 <- acomp(A1_norm)
ilrA1   <- ilr(acompA1)

# A2: carrier
A2_pseudo <- A2
A2_pseudo[A2_pseudo == 0] <- epsilon
row_sums_A2 <- rowSums(A2_pseudo)
A2_norm <- sweep(A2_pseudo, 1, row_sums_A2, "/")

acompA2 <- acomp(A2_norm)
ilrA2   <- ilr(acompA2)
```

Below, we use PCA (Principal Component Analysis) to examine how much variance is explained by the primary components in both A1 (no carrier) and A2 (with carrier). This helps us understand the impact of combining water and fat into a single variable.

```
# PCA for A1 (water + vet apart)
pcaA1 <- prcomp(ilrA1, center = FALSE, scale = FALSE)
pcaA1_summary <- summary(pcaA1)

# PCA for A2 (carrier)
pcaA2 <- prcomp(ilrA2, center = FALSE, scale = FALSE)
pcaA2_summary <- summary(pcaA2)

# PCA Results Comparison
cat("PCA A1 Summary:\n")
```

```
## PCA A1 Summary:
```

```
print(pcaA1_summary)
```

```
## Importance of components:
##                            PC1     PC2     PC3     PC4     PC5    PC6     PC7
## Standard deviation     16.0230 10.5689 6.28056 5.67647 4.49907 4.3963 3.28852
## Proportion of Variance  0.5114  0.2225 0.07858 0.06419 0.04032 0.0385 0.02154
## Cumulative Proportion   0.5114  0.7339 0.81252 0.87670 0.91703 0.9555 0.97707
##                            PC8    PC9
## Standard deviation     2.92413 1.7204
## Proportion of Variance 0.01703 0.0059
## Cumulative Proportion  0.99410 1.0000
```

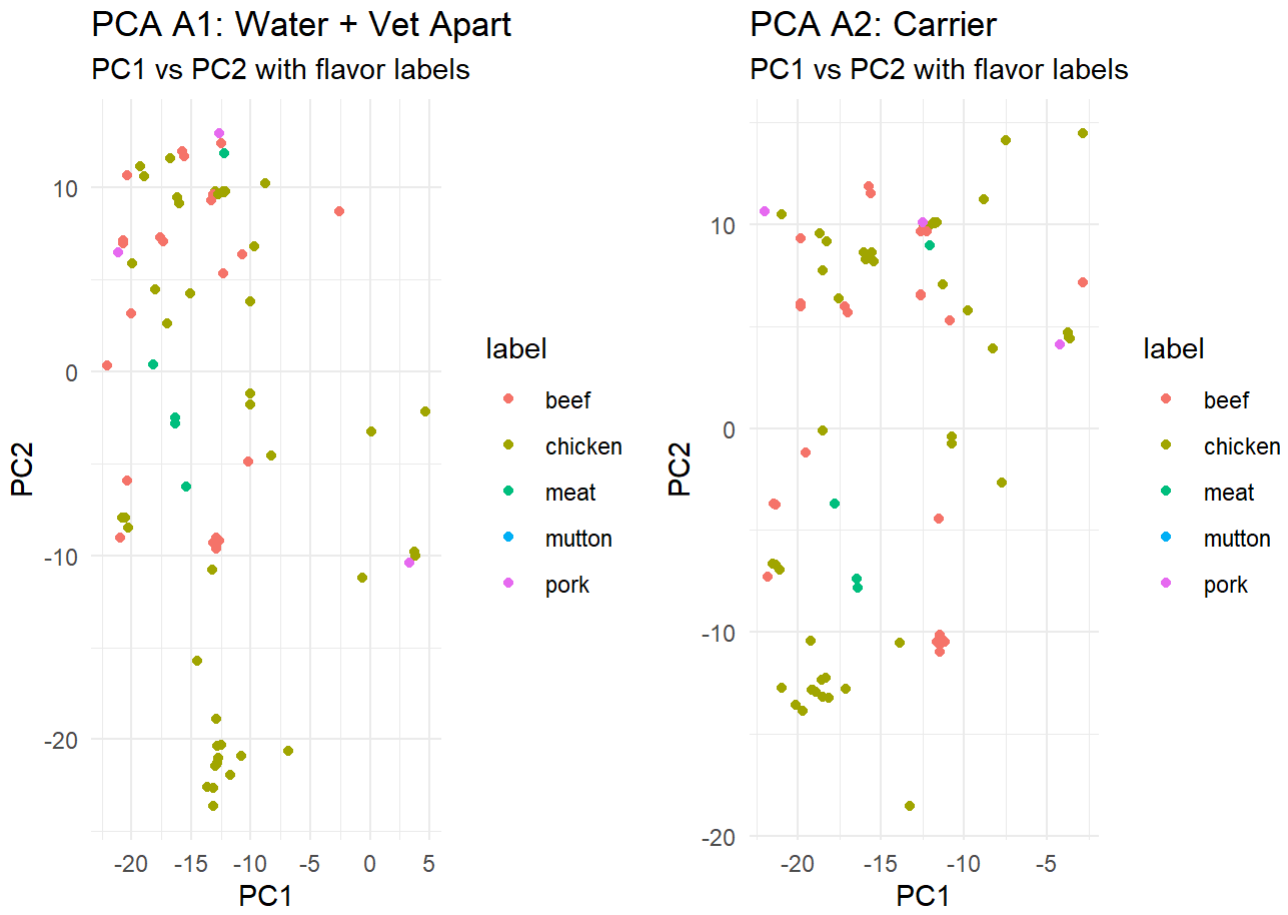
```
cat("\nPCA A2 Summary:\n")
```

```
##
## PCA A2 Summary:
```

```
print(pcaA2_summary)
```

```
## Importance of components:
##                            PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     16.5905 8.7422 5.91038 5.30253 4.50672 3.23868 2.81907
## Proportion of Variance  0.6021 0.1672 0.07642 0.06151 0.04443 0.02295 0.01739
## Cumulative Proportion   0.6021 0.7693 0.84575 0.90726 0.95169 0.97464 0.99202
##                            PC8
## Standard deviation     1.90950
## Proportion of Variance 0.00798
## Cumulative Proportion  1.00000
```

```
# Plots for PCA Comparison
dfA1pca <- data.frame(PC1 = pcaA1$x[, 1],
                      PC2 = pcaA1$x[, 2],
                      label = flavor_labels)
p1 <- ggplot(dfA1pca, aes(x = PC1, y = PC2, color = label)) +
  geom_point() +
  labs(title = "PCA A1: Water + Vet Apart",
       subtitle = "PC1 vs PC2 with flavor labels",
       x = "PC1", y = "PC2") +
  theme_minimal()

dfA2pca <- data.frame(PC1 = pcaA2$x[, 1],
                      PC2 = pcaA2$x[, 2],
                      label = flavor_labels)
p2 <- ggplot(dfA2pca, aes(x = PC1, y = PC2, color = label)) +
  geom_point() +
  labs(title = "PCA A2: Carrier",
       subtitle = "PC1 vs PC2 with flavor labels",
       x = "PC1", y = "PC2") +
  theme_minimal()

library(patchwork) # For side-by-side plot
p1 + p2
```




Discriminant Analysis (DA) tests how well the matrices distinguish flavor_labels based on chemical composition. A1 and A2 are compared based on classification accuracy.

```
# DA for A1
dfA1 <- as.data.frame(ilrA1)
dfA1$label <- flavor_labels

ldaA1 <- lda(label ~ ., data = dfA1)
predA1 <- predict(ldaA1, dfA1)$class
accA1 <- mean(predA1 == dfA1$label)
cat("Discriminant Accuracy A1 (water + vet apart):", accA1, "\n")
```

```
## Discriminant Accuracy A1 (water + vet apart): 0.8536585
```

```
# DA for A2
dfA2 <- as.data.frame(ilrA2)
dfA2$label <- flavor_labels

ldaA2 <- lda(label ~ ., data = dfA2)
predA2 <- predict(ldaA2, dfA2)$class
accA2 <- mean(predA2 == dfA2$label)
cat("Discriminant Accuracy A2 (carrier):", accA2, "\n")
```

```
## Discriminant Accuracy A2 (carrier): 0.7560976
```

```
# Comparison
cat("\nComparison:\n")
```

```
##
## Comparison:
```

```
cat("A1 Accuracy:", accA1, "| A2 Accuracy:", accA2, "\n")
```

```
## A1 Accuracy: 0.8536585 | A2 Accuracy: 0.7560976
```

Hoewel het samenvoegen van water en vetten tot “carrier” een lichte afname in discriminant vermogen en nuance veroorzaakt, blijven de belangrijkste patronen in de data behouden. Deze stap is daarom statistisch en praktisch verantwoord, gezien de aard van de patenten en het doel om de dataset overzichtelijker en robuuster te maken.

# Ongewenste categoriën Filteren

Columns with many zeros or low variance may not contribute significantly to flavor classification or PCA. Here, we test the impact of removing “irrelevant” columns ( `Zuren` , `Volatiles` , etc.). Steps: - Examine correlations and variance to identify redundancy. - Apply compositional transformations (pseudocount + normalization + ILR). - Evaluate the impact on PCA and Discriminant Analysis (DA).

```
#checken of speudo en norm niet moeten gebeuren
# === Inspect Pairwise Correlations and Variance ===
cat("\n=== Pairwise Correlations in B1 ===\n")
```

```
##
## === Pairwise Correlations in B1 ===
```

```
cor_B1 <- cor(B1)
print(cor_B1)
```

```
##                              Aminozuren Ribonucleotides    Suiker
## Aminozuren                   1.00000000      0.43634262 0.8018575
## Ribonucleotides              0.43634262      1.00000000 0.2981982
## Suiker                       0.80185746      0.29819818 1.0000000
## Zwavel bevattende aminozuren 0.04164819      0.10860201 0.0137317
## carrier                      0.18883338      0.09866948 0.1567181
##                              Zwavel bevattende aminozuren    carrier
## Aminozuren                                     0.04164819 0.18883338
## Ribonucleotides                                0.10860201 0.09866948
## Suiker                                         0.01373170 0.15671815
## Zwavel bevattende aminozuren                   1.00000000 0.66108349
## carrier                                        0.66108349 1.00000000
```

```
cat("\n=== Column Variance in B1 ===\n")
```

```
##
## === Column Variance in B1 ===
```

```
var_B1 <- apply(B1, 2, var)
print(var_B1)
```

```
##                   Aminozuren              Ribonucleotides
##                  78011.76165                     52.84184
##                       Suiker Zwavel bevattende aminozuren
##                   8710.74813                     80.40697
##                      carrier
##                  91905.53050
```

```
cat("\n=== Pairwise Correlations in A2 ===\n")
```

```
##
## === Pairwise Correlations in A2 ===
```

```
cor_A2 <- cor(A2)
print(cor_A2)
```

```
##                              Aminozuren Ribonucleotides Smaakenhancer
## Aminozuren                    1.00000000     0.436342617    0.71569910
## Ribonucleotides               0.43634262     1.000000000    0.18751949
## Smaakenhancer                 0.71569910     0.187519490    1.00000000
## Suiker                        0.80185746     0.298198184    0.82532839
## Zuren                        -0.10181242    -0.064527816   -0.08087102
## Zwavel bevattende aminozuren  0.04164819     0.108602005   -0.06409068
## extra                        -0.19693819    -0.013459938   -0.13735568
## Volatiles                    -0.14763120     0.001692042   -0.10205719
## carrier                       0.18883338     0.098669480    0.09199575
##                                  Suiker       Zuren
## Aminozuren                    0.8018575 -0.10181242
## Ribonucleotides               0.2981982 -0.06452782
## Smaakenhancer                 0.8253284 -0.08087102
## Suiker                        1.0000000 -0.10034100
## Zuren                        -0.1003410  1.00000000
## Zwavel bevattende aminozuren  0.0137317 -0.12466588
## extra                        -0.1771277 -0.04281252
## Volatiles                    -0.1385368 -0.03709792
## carrier                       0.1567181 -0.05469376
##                              Zwavel bevattende aminozuren       extra
## Aminozuren                                     0.04164819 -0.19693819
## Ribonucleotides                                0.10860201 -0.01345994
## Smaakenhancer                                 -0.06409068 -0.13735568
## Suiker                                         0.01373170 -0.17712772
## Zuren                                         -0.12466588 -0.04281252
## Zwavel bevattende aminozuren                   1.00000000  0.46559184
## extra                                          0.46559184  1.00000000
## Volatiles                                      0.41443268  0.82245843
## carrier                                        0.66108349  0.39593425
##                                Volatiles     carrier
## Aminozuren                   -0.147631202  0.18883338
## Ribonucleotides               0.001692042  0.09866948
## Smaakenhancer                -0.102057186  0.09199575
## Suiker                       -0.138536770  0.15671815
## Zuren                        -0.037097917 -0.05469376
## Zwavel bevattende aminozuren  0.414432679  0.66108349
## extra                         0.822458433  0.39593425
## Volatiles                     1.000000000  0.32394887
## carrier                       0.323948869  1.00000000
```

```
cat("\n=== Column Variance in A2 ===\n")
```

```
##
## === Column Variance in A2 ===
```

```
var_A2 <- apply(A2, 2, var)
print(var_A2)
```

```
##                   Aminozuren              Ribonucleotides
##                 7.801176e+04                 5.284184e+01
##                Smaakenhancer                       Suiker
##                 1.029353e+05                 8.710748e+03
##                       Zuren Zwavel bevattende aminozuren
##                 4.467146e-03                 8.040697e+01
##                       extra                    Volatiles
##                 6.329094e+03                 1.943276e-03
##                     carrier
##                 9.190553e+04
```

Here we prepare B1 for compositional analysis. Zero values are replaced with epsilon, and data is normalized row-wise (composition sums = 1). ILR transformation maps the data to an orthogonal space.

```
# === Pseudocount and Normalization ===
B1_pseudo <- B1
B1_pseudo[B1_pseudo == 0] <- epsilon
row_sums_B1 <- rowSums(B1_pseudo)
B1_norm <- sweep(B1_pseudo, 1, row_sums_B1, "/")

# ILR Transformation
acompB1 <- acomp(B1_norm)
ilrB1 <- ilr(acompB1)

cat("\nILR-transformed B1 dimensions:", dim(ilrB1), "\n")
```

```
##
## ILR-transformed B1 dimensions: 123 4
```

PCA is applied to ILR-transformed B1 to assess how much variance is explained by principal components. We compare these results to A2 (carrier without filtering).

```
# PCA on B1 (carrier + filtering)
pcaB1 <- prcomp(ilrB1, center = FALSE, scale = FALSE)
pcaB1_summary <- summary(pcaB1)

# Comparison to A2
cat("PCA B1 Summary:\n")
```

```
## PCA B1 Summary:
```

```
print(pcaB1_summary)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4
## Standard deviation     7.7047 5.9522 4.9041 3.39963
## Proportion of Variance 0.4552 0.2717 0.1844 0.08863
## Cumulative Proportion  0.4552 0.7269 0.9114 1.00000
```

```
cat("\nComparing PCA Results:\n")
```

```
##
## Comparing PCA Results:
```

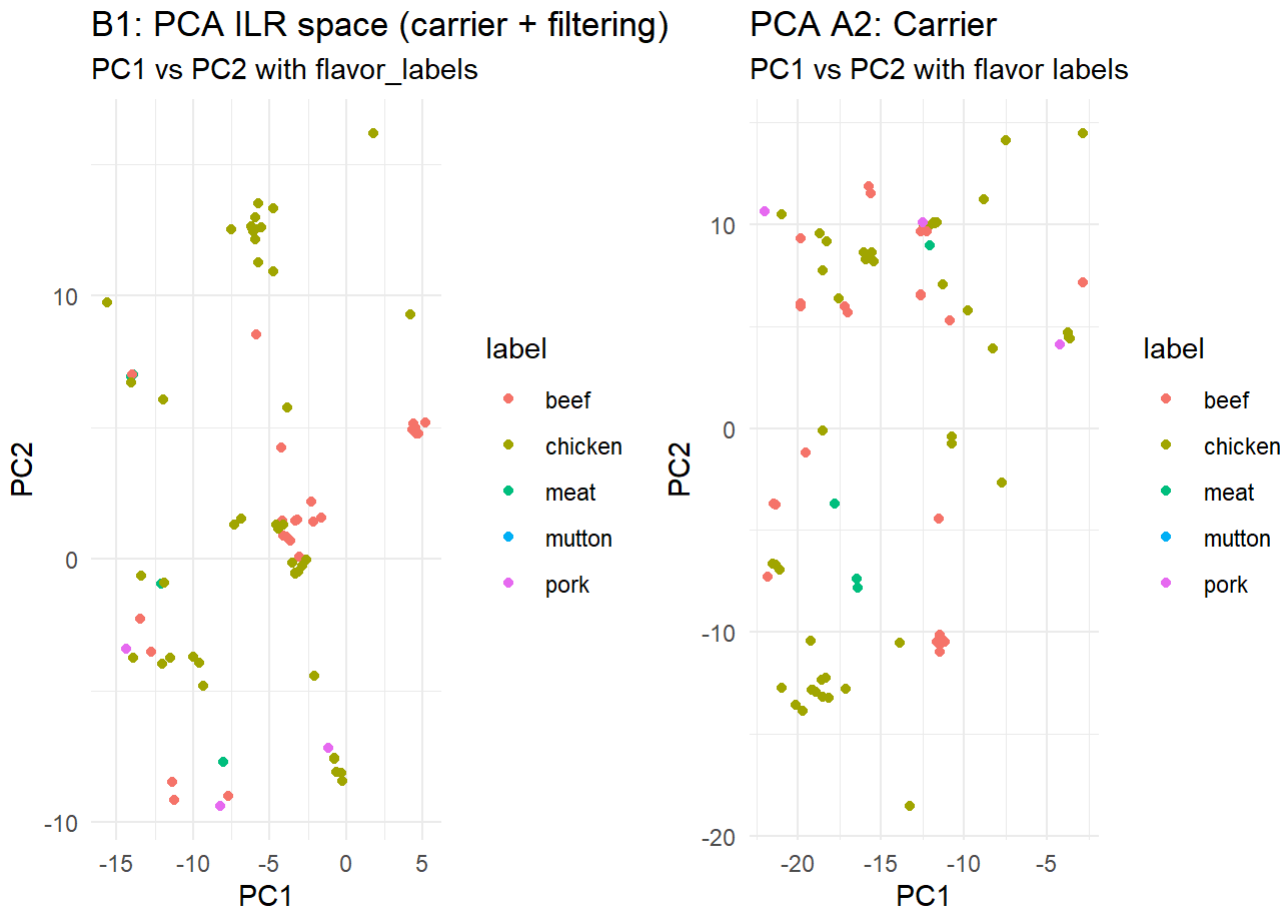
```
cat("B1 Variance Explained (PC1):", pcaB1_summary$importance[2, 1], "\n")
```

```
## B1 Variance Explained (PC1): 0.45524
```

```
cat("A2 Variance Explained (PC1):", pcaA2_summary$importance[2, 1], "\n")
```

```
## A2 Variance Explained (PC1): 0.60214
```

```
# Visualization
dfB1pca <- data.frame(PC1 = pcaB1$x[, 1],
                      PC2 = pcaB1$x[, 2],
                      label = flavor_labels)
p_b1 <- ggplot(dfB1pca, aes(x = PC1, y = PC2, color = label)) +
  geom_point() +
  labs(title = "B1: PCA ILR space (carrier + filtering)",
       subtitle = "PC1 vs PC2 with flavor_labels",
       x = "PC1", y = "PC2") +
  theme_minimal()

# Combine PCA plots for side-by-side comparison
library(patchwork)
p_b1 + p2
```



```
# DA on B1
dfB1 <- as.data.frame(ilrB1)
dfB1$label <- flavor_labels

ldaB1 <- lda(label ~ ., data = dfB1)
predB1 <- predict(ldaB1, dfB1)$class
accB1 <- mean(predB1 == dfB1$label)

# Compare with A2
cat("\nDiscriminant Accuracy B1 (carrier + filtering):", accB1, "\n")
```

```
##
## Discriminant Accuracy B1 (carrier + filtering): 0.6585366
```

```
cat("Discriminant Accuracy A2 (carrier):", accA2, "\n")
```

```
## Discriminant Accuracy A2 (carrier): 0.7560976
```

```
# Interpretation:
# - DA accuracy shows whether filtering affects classification.
# - Compare to `A2` to assess if filtering improves or reduces accuracy.
```

De filtering van kolommen zoals extra, Zuren, en Volatiles is statistisch verantwoord. Deze groepen hebben een lage correlatie met belangrijke smaakgerelateerde componenten, minimale variantie, en dragen weinig bij aan de classificatie en clustering van recepten. Het verwijderen van deze categorieën helpt om ruis te verminderen en de analyses te verfijnen zonder betekenisvolle informatie te verliezen.

# Epsilon gevoeligheid analyse

In compositional data analysis is het gebruik van een pseudotelling ( `epsilon` ) cruciaal om nullen te vervangen, aangezien log-ratio-transformaties zoals ILR en CLR niet gedefinieerd zijn voor nullen. In deze analyse evalueren we de gevoeligheid van de resultaten voor verschillende epsilon-waarden en hoe deze de PCA beïnvloeden.

**Doel:**
Het vervangen van nullen met een kleine epsilon-waarde ( `1e-6` , `1e-5` , `1e-4` ) en het normaliseren van de rijen om de proporties correct te behouden.

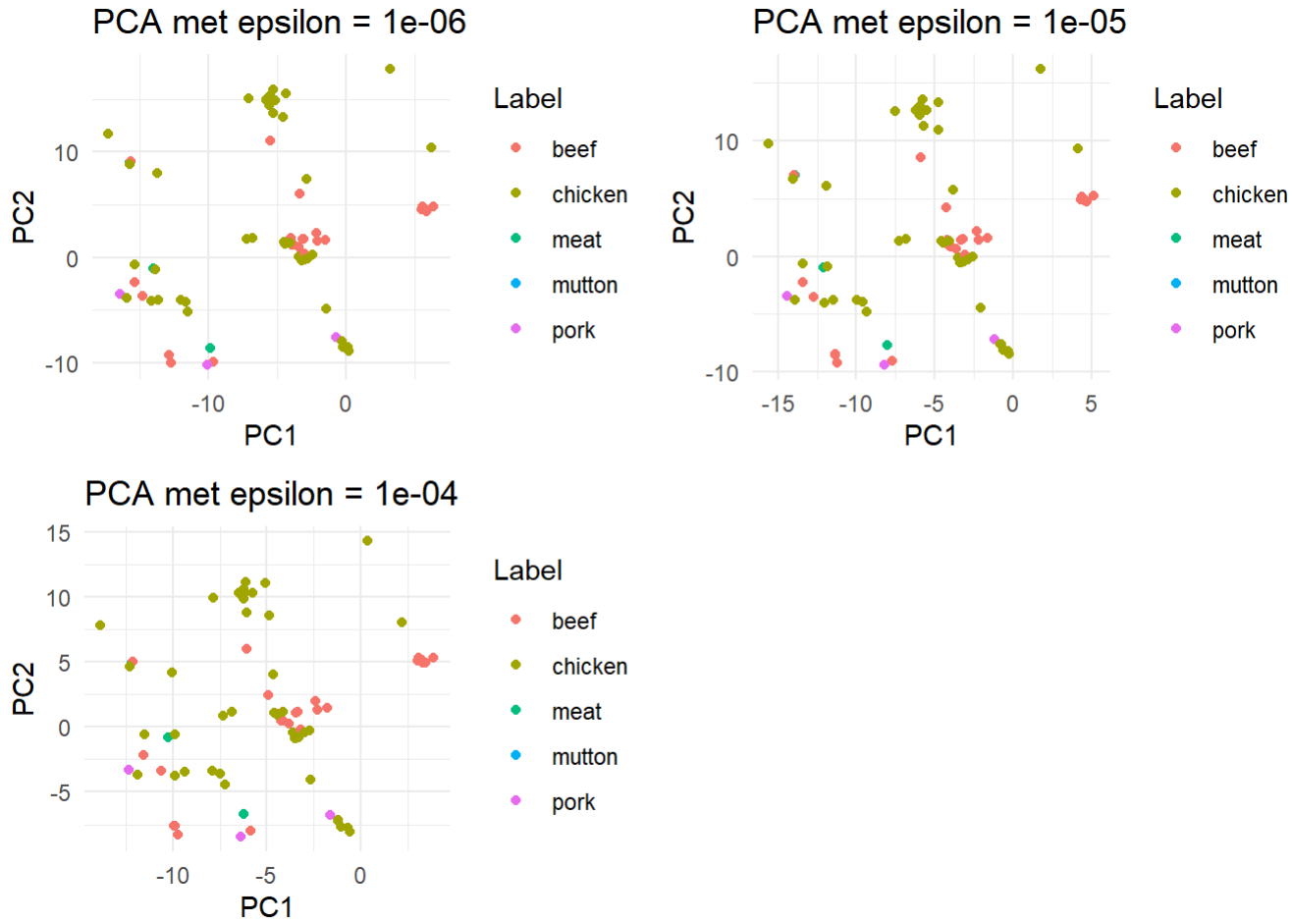
```
# Definieer epsilon-waarden
epsilon_values <- c(1e-6, 1e-5, 1e-4)

# Opslag voor resultaten
results <- list()

# Test verschillende epsilon-waarden
for (eps in epsilon_values) {
  # Pseudotelling toepassen
  B1_pseudo <- B1
  B1_pseudo[B1_pseudo == 0] <- eps

  # Normaliseren
  row_sums_B1 <- rowSums(B1_pseudo)
  B1_norm <- sweep(B1_pseudo, 1, row_sums_B1, "/")

  # PCA uitvoeren
  acomp_B1 <- acomp(B1_norm)
  ilr_B1 <- ilr(acomp_B1)
  pca_B1 <- prcomp(ilr_B1, center = FALSE, scale = FALSE)

  # Resultaten opslaan
  results[[as.character(eps)]] <- list(
    epsilon = eps,
    norm_matrix = B1_norm,
    pca = pca_B1
  )
}
cat("Epsilon gevoeligheidstest voltooid voor waarden:", epsilon_values, "\n")
```

```
## Epsilon gevoeligheidstest voltooid voor waarden: 1e-06 1e-05 1e-04
```

```
# Vergelijk PCA-resultaten
for (eps in names(results)) {
  cat("\n=== PCA Resultaten voor Epsilon =", eps, "===\n")
  print(summary(results[[eps]]$pca))
}
```

```
##
## === PCA Resultaten voor Epsilon = 1e-06 ===
## Importance of components:
##                           PC1    PC2    PC3     PC4
## Standard deviation     8.7455 6.8016 5.6801 3.94275
## Proportion of Variance 0.4484 0.2712 0.1892 0.09115
## Cumulative Proportion  0.4484 0.7197 0.9089 1.00000
##
## === PCA Resultaten voor Epsilon = 1e-05 ===
## Importance of components:
##                           PC1    PC2    PC3     PC4
## Standard deviation     7.7047 5.9522 4.9041 3.39963
## Proportion of Variance 0.4552 0.2717 0.1844 0.08863
## Cumulative Proportion  0.4552 0.7269 0.9114 1.00000
##
## === PCA Resultaten voor Epsilon = 1e-04 ===
## Importance of components:
##                           PC1    PC2    PC3    PC4
## Standard deviation     6.7067 5.1118 4.1248 2.8519
## Proportion of Variance 0.4673 0.2715 0.1768 0.0845
## Cumulative Proportion  0.4673 0.7388 0.9155 1.0000
```

```
library(ggplot2)

# Combineer resultaten voor visualisatie
pca_plots <- list()
for (eps in names(results)) {
  pca_obj <- results[[eps]]$pca
  pca_data <- data.frame(
    PC1 = pca_obj$x[, 1],
    PC2 = pca_obj$x[, 2],
    label = flavor_labels
  )
  pca_plots[[eps]] <- ggplot(pca_data, aes(x = PC1, y = PC2, color = label)) +
    geom_point() +
    labs(
      title = paste("PCA met epsilon =", eps),
      x = "PC1",
      y = "PC2",
      color = "Label"
    ) +
    theme_minimal()
}

# Print alle PCA-plots
library(gridExtra)
do.call(grid.arrange, c(pca_plots, ncol = 2))
```




```
# Samenvatting van belangrijkste bevindingen
cat("\n=== Samenvatting van Epsilon Gevoeligheidstest ===\n")
```

```
##
## === Samenvatting van Epsilon Gevoeligheidstest ===
```

```
for (eps in names(results)) {
  pca_obj <- results[[eps]]$pca
  variance_explained <- summary(pca_obj)$importance[2, 1]
  cat(
    "Epsilon:", eps,
    "| Variance Explained (PC1):", variance_explained, "\n"
  )
}
```

```
## Epsilon: 1e-06 | Variance Explained (PC1): 0.44844
## Epsilon: 1e-05 | Variance Explained (PC1): 0.45524
## Epsilon: 1e-04 | Variance Explained (PC1): 0.46728
```

De gevoeligheidstest bevestigt dat de keuze van epsilon een beperkte invloed heeft op de resultaten van de PCA. De verklaarde variantie van de eerste component en de clustering in de PCA-plots blijven grotendeels stabiel. Dit suggereert dat de analyse robuust is tegen kleine veranderingen in de epsilon-waarde binnen het bereik van 1e-06 tot 1e-04.

Het gebruik van een epsilon-waarde van 1e-05 lijkt een goede balans te bieden tussen stabiliteit en rekenkundige consistentie. Deze waarde zal worden gehanteerd in verdere analyses.

# Outlier detection

Outliers kunnen een grote invloed hebben op compositional data analyses (CoDa). Het is belangrijk om deze te identificeren en te evalueren, aangezien ze mogelijk unieke recepten vertegenwoordigen of datainvoerfouten kunnen zijn. Hier gebruiken we de Isometric Log-Ratio (ILR)-transformatie om de composities te analyseren en outliers te detecteren met behulp van Mahalanobis-afstanden. De recipe_id wordt gebruikt om outliers direct te koppelen aan de originele recepten in de database.

```
# Voeg een unieke rijnummerkolom toe aan data_grouped
data_grouped_with_index <- data_grouped %>%
  mutate(row_number = row_number())  # Rij-index toevoegen

# Voeg pseudocount toe en normaliseer de data
B1_pseudo <- B1
B1_pseudo[B1_pseudo == 0] <- epsilon
row_sums_B1 <- rowSums(B1_pseudo)
B1_norm <- sweep(B1_pseudo, 1, row_sums_B1, "/")

# ILR-transformatie
acompB1 <- acomp(B1_norm)
ilrB1 <- ilr(acompB1)

# Bereken Mahalanobis-afstanden
dist_ilr <- mahalanobis(ilrB1, center = colMeans(ilrB1), cov = cov(ilrB1))

# Vrijheidsgraden en cutoff waarde berekenen
df_ilr <- ncol(ilrB1)  # Vrijheidsgraad
cutoff <- qchisq(0.975, df = df_ilr)  # 97.5% cutoff

# Identificeer outliers
outliers_idx <- which(dist_ilr > cutoff)

# Maak een tabel met de rij-indexen en recipe_id's van outliers
outlier_table <- data.frame(
  Row_Index = outliers_idx,  # Rij-indexen in B1
  Recipe_ID = rownames(B1)[outliers_idx]  # Recipe ID's
)

# Toon het aantal outliers en hun details
cat("Aantal Outliers:", nrow(outlier_table), "\n")
```

```
## Aantal Outliers: 8
```

```
cat("Outliers (Details):\n")
```

```
## Outliers (Details):
```

```
print(outlier_table)
```

```
##                 Row_Index       Recipe_ID
## Example VI              15      Example VI
## Example VII             16     Example VII
## Example VIII            17    Example VIII
## Example VII - K        102 Example VII - K
## Example VII - O        106 Example VII - O
## Example 2              120       Example 2
## Example 3              121       Example 3
## Example 5              123       Example 5
```

```
# Haal de originele recepten op uit de data_grouped_with_index tabel via rijnummers
outlier_recipes <- data_grouped_with_index %>%
  filter(row_number %in% outlier_table$Row_Index)

# Toon de originele recepten
cat("\nOriginele Outlier Recepten:\n")
```

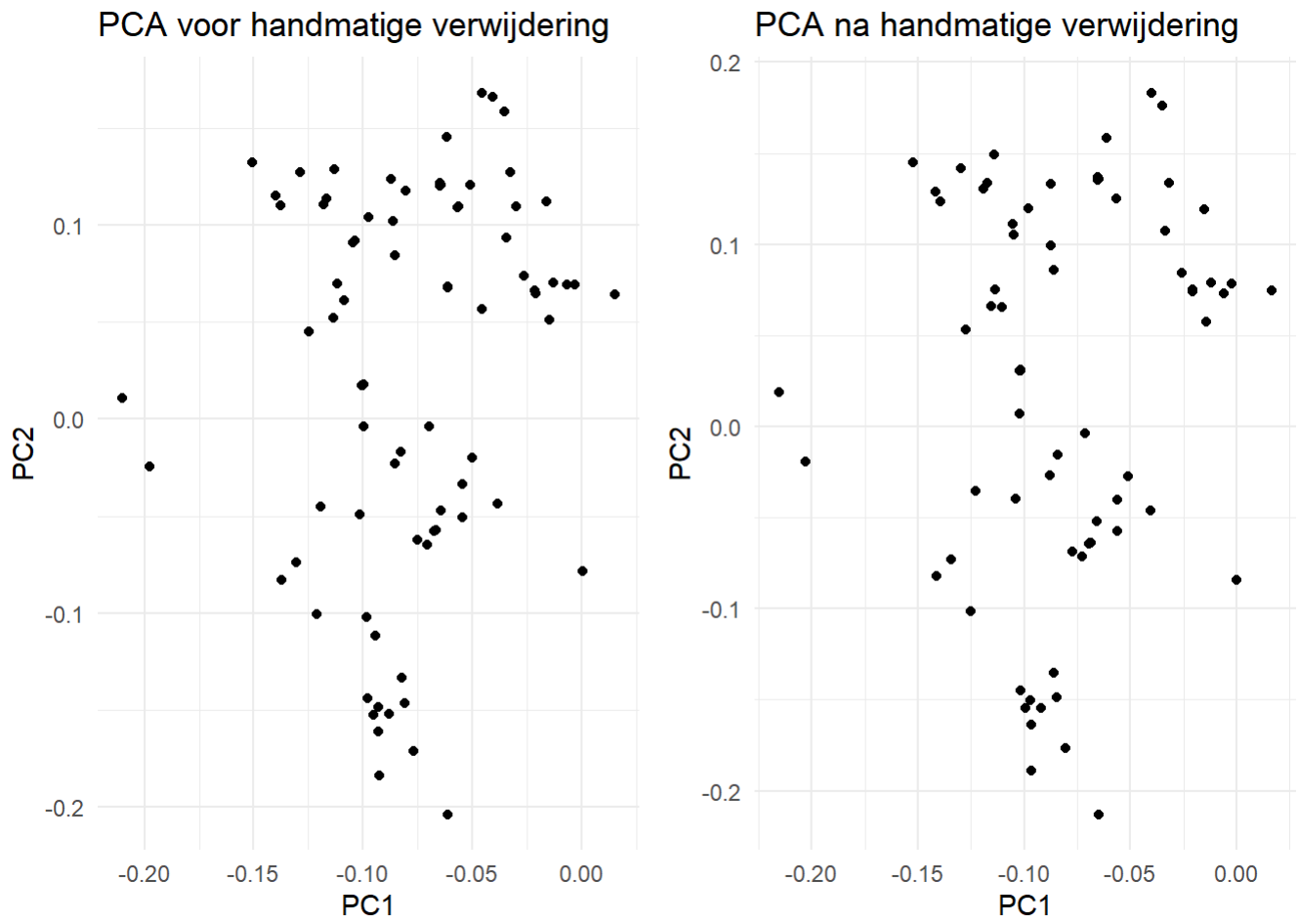
```
##
## Originele Outlier Recepten:
```

```
print(outlier_recipes)
```

```
## # A tibble: 8 × 16
##   recipe_name     cooking_time temperature flavor_label preparation Aminozuren
##   <chr>                  <dbl>       <int> <chr>        <chr>            <dbl>
## 1 Example VI               240         100 beef         <NA>              155.
## 2 Example VII              0.5         163 beef         <NA>              155.
## 3 Example VIII             0.5         163 beef         <NA>              155.
## 4 Example VII - K          240          72 chicken      <NA>              616
## 5 Example VII - O          240          72 chicken      <NA>              616
## 6 Example 2                120         130 beef         cooked              0
## 7 Example 3                120         130 pork         cooked              0
## 8 Example 5                180         100 meat         cooked              0
## # ℹ 10 more variables: Ribonucleotides <dbl>, Smaakenhancer <dbl>,
## #   Suiker <dbl>, Vetten <dbl>, Water <dbl>, Zuren <dbl>,
## #   `Zwavel bevattende aminozuren` <dbl>, extra <dbl>, Volatiles <dbl>,
## #   row_number <int>
```

schrijf motivatie outliers op basis patent

```
# === Handmatig geselecteerde rijen voor verwijdering ===
# Geef hier de rijnummers in die je wilt verwijderen
manual_outliers_idx <- c(8,15,16,17,52,53,102,106,120,121,123)  # Vervang door jouw rijnummer
s

# Controleer dat de rijnummers geldig zijn
if (any(manual_outliers_idx > nrow(B1))) {
  stop("Een of meer opgegeven rijnummers bestaan niet in B1!")
}

# Data zonder de geselecteerde rijen
B1_no_manual_outliers <- B1[-manual_outliers_idx, ]

# === PCA Analyse ===
# Data vóór verwijdering
pca_before <- prcomp(ilr(acomp(sweep(B1, 1, rowSums(B1), "/"))), center=FALSE, scale=FALSE)

# Data na verwijdering
B1_no_manual_norm <- sweep(B1_no_manual_outliers, 1, rowSums(B1_no_manual_outliers), "/")
pca_after <- prcomp(ilr(acomp(B1_no_manual_norm)), center=FALSE, scale=FALSE)

# PCA Plots
library(gridExtra)
plot_before <- autoplot(pca_before, data = NULL) +
  labs(title = "PCA voor handmatige verwijdering", x = "PC1", y = "PC2") +
  theme_minimal()

plot_after <- autoplot(pca_after, data = NULL) +
  labs(title = "PCA na handmatige verwijdering", x = "PC1", y = "PC2") +
  theme_minimal()

grid.arrange(plot_before, plot_after, ncol = 2)
```




```
# === Discriminant Analyse (DA) ===
# Data opzetten voor DA
flavor_labels_before <- flavor_labels  # Labels vóór verwijdering
flavor_labels_after <- flavor_labels[-manual_outliers_idx]  # Labels na verwijdering

# DA vóór verwijdering
ilr_before <- ilr(acomp(sweep(B1, 1, rowSums(B1), "/")))
df_before <- data.frame(ilr_before, label = flavor_labels_before)
lda_before <- lda(label ~ ., data = df_before)
pred_before <- predict(lda_before)$class
acc_before <- mean(pred_before == df_before$label)

# DA na verwijdering
ilr_after <- ilr(acomp(B1_no_manual_norm))
df_after <- data.frame(ilr_after, label = flavor_labels_after)
lda_after <- lda(label ~ ., data = df_after)
pred_after <- predict(lda_after)$class
acc_after <- mean(pred_after == df_after$label)

# Resultaten DA
cat("Discriminant Analyse Resultaten:\n")
```

```
## Discriminant Analyse Resultaten:
```

```
cat("Accuraatheid vóór verwijdering:", acc_before, "\n")
```

```
## Accuraatheid vóór verwijdering: 0.8130081
```

```
cat("Accuraatheid na verwijdering:", acc_after, "\n")
```

```
## Accuraatheid na verwijdering: 0.8482143
```

De verwijdering van outliers blijkt gerechtvaardigd te zijn, aangezien deze recepten duidelijk afwijken van de algemene structuur van de dataset, zoals geïllustreerd door de verbeterde clustering in de PCA en de verhoogde discriminant analyse (DA) accuraatheid van 81.3% naar 84.8%. Deze verbeteringen suggereren dat de outliers mogelijk ruis of uitzonderlijke gevallen vertegenwoordigen die niet bijdragen aan de identificatie van algemene patronen in de data. Door deze outliers te verwijderen, wordt de dataset consistenter, wat belangrijk is voor compositional data analyses (CoDa), waarbij extreme waarden de balans tussen variabelen kunnen verstoren.