Predicting Pokémon Types Using Clustering and Classification

Code Submission

Justin Zhang, Isaac Baguisa, Alex Faassen

2025-04-04

Note: Commented out unnecessary "test" outputs.

```
## Universal libraries
library(png)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.4.3
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(knitr)
```

Data

Pre-processing

```
setwd("Data")

## Load Stats
stats = read.csv("pokemon_stats.csv", header = TRUE)
# Check
# str(stats)
# Image path
stats$image_path = paste0("images/", tolower(stats$name.simple), ".png")
# Check
# head(stats$image_path)

## Load Image Registry
img_reg = read.csv("pokemon_img.csv", header = TRUE)
# Has Image
stats$has_img = tolower(stats$name.simple) %in% img_reg$Name
```

```r
# No Img
stats[!stats$has_img, "name"]
```

```
## character(0)
```

```r
## Load Images
# Test: Abomasnow
abomasnow = readPNG("images/abomasnow.png")
ggplot()+
  annotation_raster(abomasnow,xmin=-Inf,xmax=+Inf,ymin=-Inf,ymax= +Inf)
```

```r
# Pixels
# str(abomasnow)
pix = prod(dim(abomasnow)[1:2])

# Image List
image_list = list.files("images", pattern = "*.png") %>% paste("images/", ., sep = "")

# Helper fn
flatten_img = function(img_path) {
  img = readPNG(img_path) # Read image as raster array (dim: [120, 120, 4])
  return(as.vector(img[,,-4])) # Flatten to vector + Remove Alpha
}

# Image Dataset
images = sapply(image_list, flatten_img) %>% t() %>% as.data.frame() %>%
  mutate(image_path = image_list) %>% dplyr::select(image_path, everything())
```

```
# colnames(images) = c("image_path", rep())
# Check
images[1:5, 1:5]
```

```
##                                          image_path V1 V2 V3 V4
## images/abomasnow.png             images/abomasnow.png  0  0  0  0
## images/abra.png                       images/abra.png  0  0  0  0
## images/absol.png                     images/absol.png  0  0  0  0
## images/accelgor.png               images/accelgor.png  0  0  0  0
## images/aegislash-blade.png images/aegislash-blade.png  0  0  0  0
```

```
# str(images)

## Match Datasets
# Match the rows to represent the same pokemon, in the same order.
images = images %>%
  semi_join(stats, by = "image_path") %>%
  arrange(match(image_path, stats$image_path))
# Check
mean(stats$image_path == images$image_path)
```

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

```
## Save Datasets
# save(stats, images, file = "pokemon.RData")

## Reset
global_vars = c("global_vars", "images", "stats")
rm(list = setdiff(ls(), global_vars)) # Remove all vars except for global_vars
setwd("..")
```

EDA Visuals

```
library(png)
library(ggplot2)
library(patchwork)
```

```
## Warning: package 'patchwork' was built under R version 4.4.2
```

```
library(forcats)
```

```
## Warning: package 'forcats' was built under R version 4.4.2
```

```
library(tidyr)

## Example Pokemon
ex1 = readPNG("Data/images/pikachu.png")
ex2 = readPNG("Data/images/charizard.png")
p1 = ggplot()+
  annotation_raster(ex1,xmin=-Inf,xmax=+Inf,ymin=-Inf,ymax= +Inf) +
  labs(title = "Pikachu (Primary Type: Electric)")
p2 = ggplot()+
  annotation_raster(ex2,xmin=-Inf,xmax=+Inf,ymin=-Inf,ymax= +Inf) +
  labs(title = "Charizard (Primary Type: Fire)")
p1 + p2
```

Pikachu (Primary Type: Electric)

Charizard (Primary Type: Fire)

```
## Type Colours
type_cols = c(
  "bug" = "#A8B820", "dark" = "#705848", "dragon" = "#7038F8", "electric" = "#F8D030",
  "fairy" = "#EE99AC", "fighting" = "#C03028", "fire" = "#F08030", "flying" = "#A890F0",
  "ghost" = "#705898", "grass" = "#78C850", "ground" = "#E0C068", "ice" = "#98D8D8",
  "normal" = "#A8A878", "poison" = "#A040A0", "psychic" = "#F85888", "rock" = "#B8A038",
  "steel" = "#B8B8D0", "water" = "#6890F0"
)



## Top Pokemon Types: Bar Chart + Attack Boxplot
# Top N primary types (by freq)
top5_types <- stats %>%
  mutate(type1 = tolower(type1)) %>%
  count(type1, sort = TRUE) %>%
  top_n(5, n)

filtered_data <- stats %>%
  mutate(type1 = tolower(type1)) %>%
  filter(type1 %in% top5_types$type1) %>%
  mutate(type1 = fct_infreq(type1))

# Bar chart of pokemon by primary type
type_freq = ggplot(filtered_data, aes(x = fct_infreq(type1), fill = type1)) +
  geom_bar() +
```

```
  scale_fill_manual(values = type_cols) +
  labs(
    title = "Top 5 Pokémon Types",
    x = "Primary Type",
    y = "Count"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "left")



# Boxplot of Attack by Type
type_attack = ggplot(filtered_data, aes(x = type1, y = attack, fill = type1)) +
  geom_boxplot() +
  scale_fill_manual(values = type_cols) +
  labs(
    title = "Attack Stat Distributions",
    x = "Primary Type",
    y = "Attack Stat"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none")

# Boxplot of Sp. Attack by Type
type_sp_attack = ggplot(filtered_data, aes(x = type1, y = sp_attack, fill = type1)) +
  geom_boxplot() +
  scale_fill_manual(values = type_cols) +
  labs(
    title = "Special Attack Stat Distributions",
    x = "Primary Type",
    y = "Special Attack Stat"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none")



## Table of summary stats for key features
key_features <- c("hp", "attack", "defense", "sp_attack", "sp_defense", "speed", "height_m", "weight_kg

# Summary stats
summary_table <- stats %>%
  dplyr::select(all_of(key_features)) %>%
  summarise(across(everything(), list(
    mean = ~round(mean(.x, na.rm = TRUE), 2),
    sd = ~round(sd(.x, na.rm = TRUE), 2),
    min = ~min(.x, na.rm = TRUE),
    q1 = ~quantile(.x, 0.25, na.rm = TRUE),
    median = ~median(.x, na.rm = TRUE),
    q3 = ~quantile(.x, 0.75, na.rm = TRUE),
    max = ~max(.x, na.rm = TRUE)
  ), .names = "{.col}_{.fn}")) %>%
  pivot_longer(cols = everything(),
```

```
                names_to = c("Feature", "Statistic"),
                names_pattern = "^(.*)_(mean|sd|min|q1|median|q3|max)$",
                values_to = "Value") %>%
  pivot_wider(names_from = Statistic, values_from = Value)



## Primary type distribution
type_counts = as.data.frame(table(stats$type1))
colnames(type_counts) = c("Type", "Count")

type_dist = ggplot(type_counts, aes(x = reorder(Type, -Count), y = Count, fill = Type)) +
  geom_bar(stat = "identity", color = "black") +
  geom_text(aes(label = Count), vjust = -0.3, size = 3.5) +  # Data labels
  scale_fill_manual(values = type_cols) +
  labs(title = "Primary Type Distribution",
       x = "Pokémon Type", y = "Number of Pokémon") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none") +
  ylim(0, max(type_counts$Count) * 1.1)  # Add space for the labels
# type_dist



# Save
# save(p1, p2, type_freq, type_attack, type_sp_attack, summary_table, type_dist, file = "Figures/data_d

## Reset
rm(list = setdiff(ls(), global_vars)) # Remove all vars except for global_vars
```

Image Dimension Reduction

```
library(patchwork)
library(ggfortify)
```

```
## Warning: package 'ggfortify' was built under R version 4.4.2
```

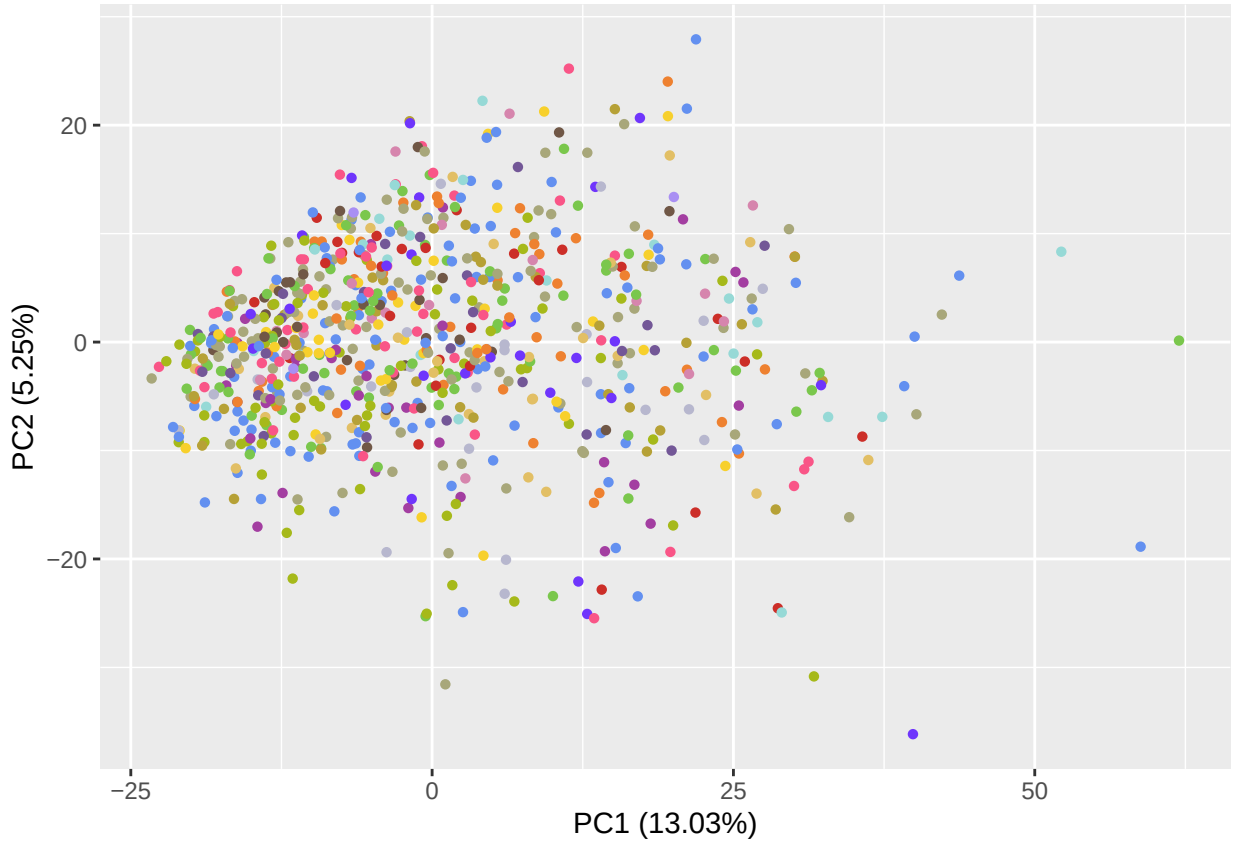
```
## Helper functions
barplot = function(values){
  n = length(values)
  df = data.frame(value = values,  index = 1:n)
  ggplot(df, aes(index, value, fill = value)) +
    geom_bar(color = "black", stat = "identity")+
    scale_fill_gradient2(low="#619CFF", mid="white", high="#F8766D")+
    theme_bw()
}

getImg = function(flat_img){
  # matrix(unlist(gsvalues), 120, 120, 4, byrow = T)
  # rasterGrob(array(flat_img, dim = c(120, 120, 4)))
  array(as.numeric(flat_img), dim = c(120, 120, 3)) # (120, 120, 4)
}
```

```r
plotImg <- function(img_raster) {
  # Plot using ggplot2
  ggplot() +
    annotation_raster(img_raster, xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = Inf) +
    theme_void()  # Remove axes for clean visualization
}
```

PCA

```r
## PCA
# Defaults: center = TRUE, scale. = FALSE
PCA = prcomp(images[,-1], center = TRUE, scale. = FALSE)

sum_PCA = summary(PCA)
# sum_PCA

# Note: Too big to save to GitHub.
```

```r
library(ggfortify)

## Biplot
# Colours
stats$type1 = factor(stats$type1)
# unique(stats$type1)
type_colours = c("#7AC74C", "#EE8130", "#6390F0", "#A6B91A", "#A8A77A", "#A33EA1",
            "#F7D02C", "#E2BF65", "#D685AD", "#CC2E28", "#F95587", "#B6A136", "#735797",
            "#96D9D6", "#6F35FC", "#705746", "#B7B7CE", "#A98FF3")
names(type_colours) = unique(stats$type1)

# autoplot(PCA, data = cbind(stats$name, stats$type1, images[,-1]), shape = FALSE, color = "stats$type1
#   scale_colour_manual(values = type_colours) +
#   theme(legend.position = "none")

# Without labels, just types.
pca_biplot = autoplot(PCA, data = cbind(stats$name, stats$type1, images[,-1]), label = FALSE, shape = 1
  scale_colour_manual(values = type_colours) +
  theme(legend.position = "none")
pca_biplot
```




```
# Note: Too big to save to GitHub.
```

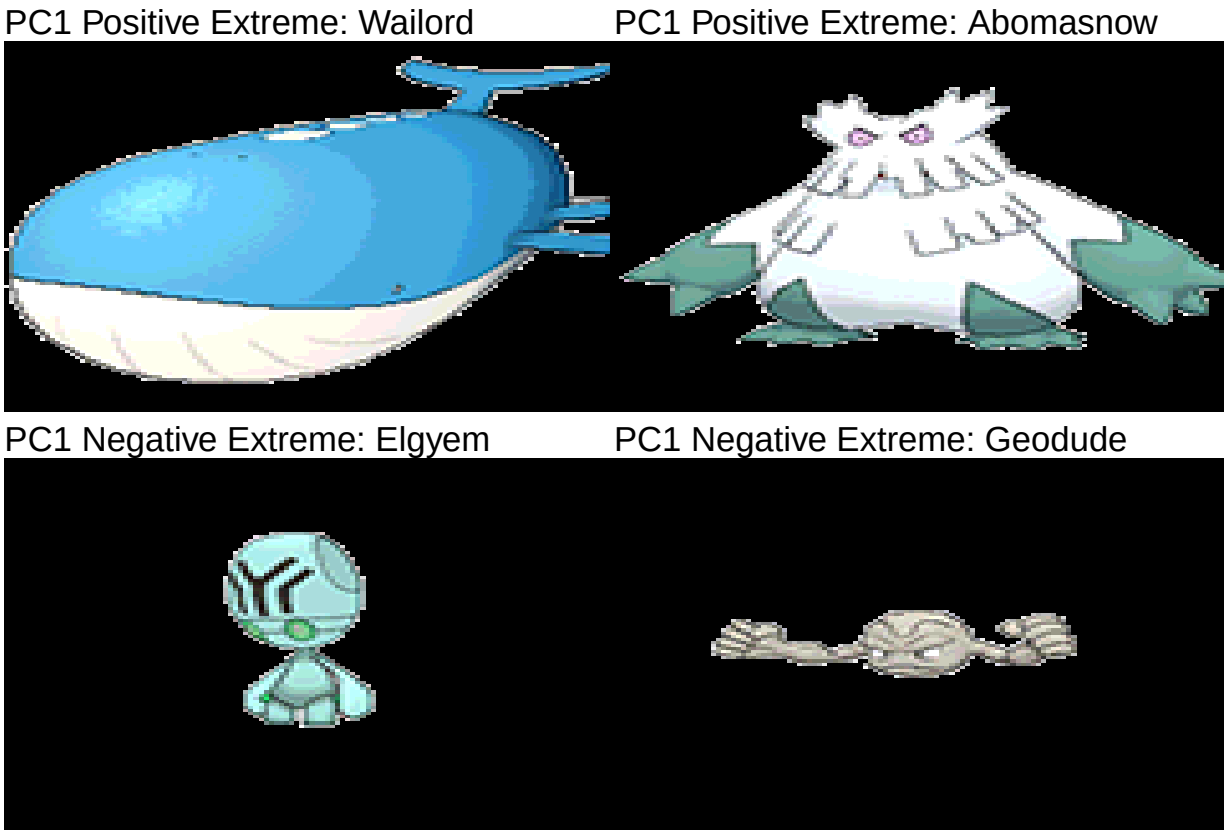
Loading Vectors

```
## Smallest PC1 Pokemon
# Extract PCA-transformed data
pca_data <- as.data.frame(PCA$x)  # Convert PCA results to a data frame

# Add Pokémon names and types for reference
pca_data$name <- stats$name  # Ensure stats$name contains Pokémon names
pca_data$type1 <- stats$type1  # Primary type for reference

# Sort by PC1 in ascending order and select the two smallest
smallest_pc1_pokemon <- pca_data[order(pca_data$PC1), ][1:2, ]

# Print result
print(rownames(smallest_pc1_pokemon))
```

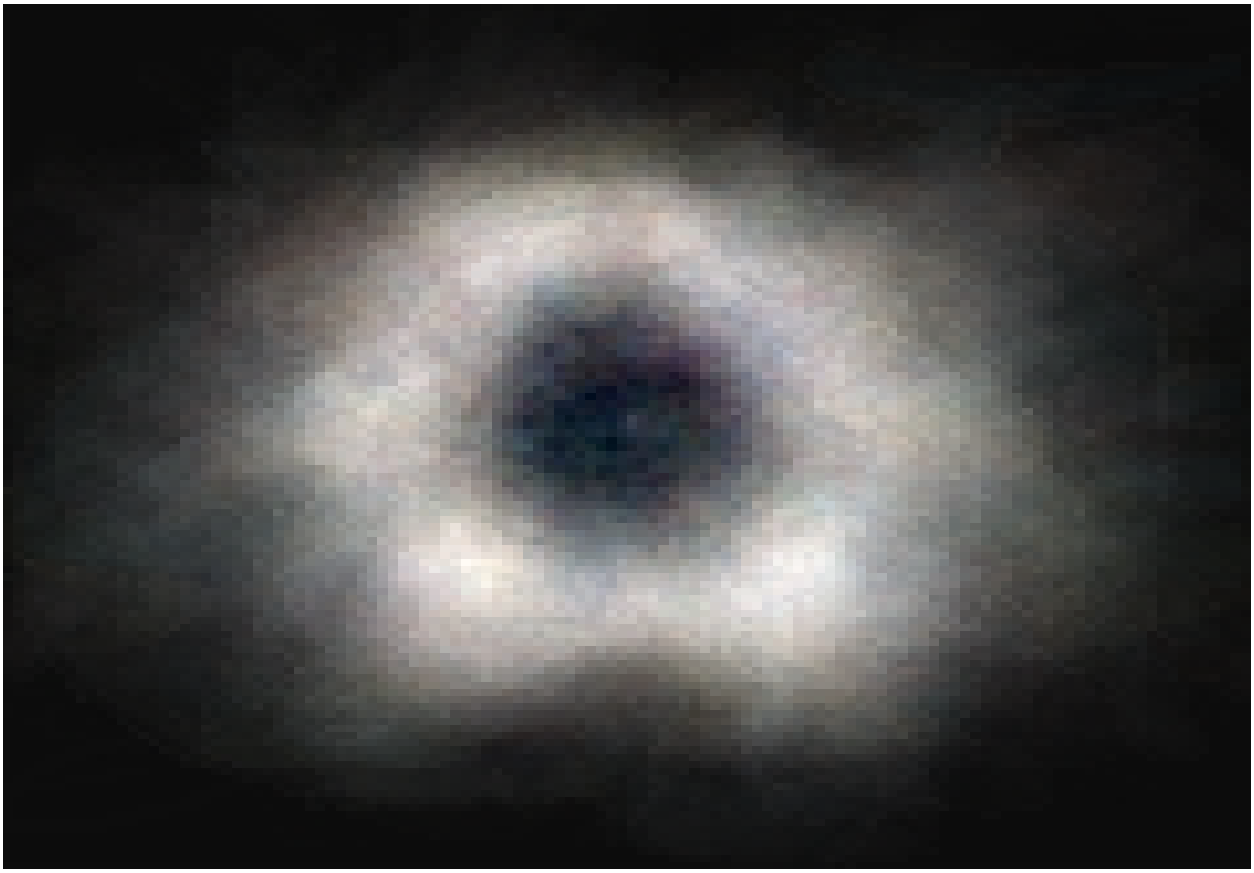
```
## [1] "images/pikipek.png" "images/unown.png"
```

```
## PC1
# Extremes
pc1_pos1 = plotImg(getImg(images[stats$name == "Wailord", -1]))
pc1_pos2 = plotImg(getImg(images[stats$name == "Abomasnow", -1]))
pc1_neg1 = plotImg(getImg(images[stats$name == "Elgyem", -1])) # Pikipek
pc1_neg2 = plotImg(getImg(images[stats$name == "Geodude", -1])) # Unown
```

```
# exPlots = sapply(c(pos1, pos2, neg1, neg2), function(rgba) plotImg(getImg(rgba)))
# (pos1 + pos2) / (neg1 + neg2)
pc1_outliers = (pc1_pos1 + labs(title = "PC1 Positive Extreme: Wailord") +
                pc1_pos2 + labs(title = "PC1 Positive Extreme: Abomasnow")) /
               (pc1_neg1 + labs(title = "PC1 Negative Extreme: Elgyem") +
                pc1_neg2 + labs(title = "PC1 Negative Extreme: Geodude"))
print(pc1_outliers)
```




```
# Loading Vector
pca = PCA$rotation[,1]
pca_norm = (pca - min(pca)) / (max(pca) - min(pca))
# pca_norm = (pca - mean(pca))/(2 * sd(pca))
# pca_clip = pmax(pmin(pca, 1), 0)
pca_v1 = plotImg(getImg(pca_norm))
pca_v1
```



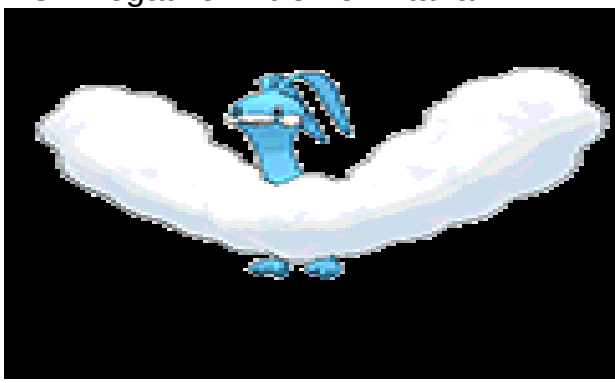
```
## Notes:
# - Likely corresponds to Pokemon image Area.
```

```
## PC2
# Extremes
pc2_pos1 = plotImg(getImg(images[stats$name == "Slowking", -1]))
pc2_pos2 = plotImg(getImg(images[stats$name == "Gardevoir", -1]))
pc2_neg1 = plotImg(getImg(images[stats$name == "Altaria", -1]))
pc2_neg2 = plotImg(getImg(images[stats$name == "Swablu", -1]))
# exPlots = sapply(c(pos1, pos2, neg1, neg2), function(rgba) plotImg(getImg(rgba)))
# (pos1 + pos2) / (neg1 + neg2)
pc2_outliers = (pc2_pos1 + labs(title = "PC1 Positive Extreme: Slowking") +
              pc2_pos2 + labs(title = "PC1 Positive Extreme: Gardevoir")) /
             (pc2_neg1 + labs(title = "PC1 Negative Extreme: Altaria") +
              pc2_neg2 + labs(title = "PC1 Negative Extreme: Swablu"))
print(pc2_outliers)
```

PC1 Positive Extreme: Slowking

PC1 Positive Extreme: Gardevoir

PC1 Negative Extreme: Altaria



PC1 Negative Extreme: Swablu

```
# Loading Vector
pca = PCA$rotation[,2]
pca_norm = (pca - min(pca)) / (max(pca) - min(pca))
pca_v2 = plotImg(getImg(pca_norm))
pca_v2
```

```
## Notes:
# - Likely corresponds to a Pokemon's relative width/height ratio.
# - Notably, neither PC1 nor PC2 correspond to colour, yet.

## Save
# save(pc1_pos1, pc1_pos2, pc1_neg1, pc1_neg2, pca_v1, pc2_pos1, pc2_pos2,
#      pc2_neg1, pc2_neg2, pca_v2, file = "Figures/appendix_A.RData")
```

Variance Explained

```
## Variance Explained
ve = sum_PCA$importance[3,]

## Retain PCs
var_retain = 0.8 # % of VE
num_pcs = min(which(ve >= var_retain)); num_pcs
```

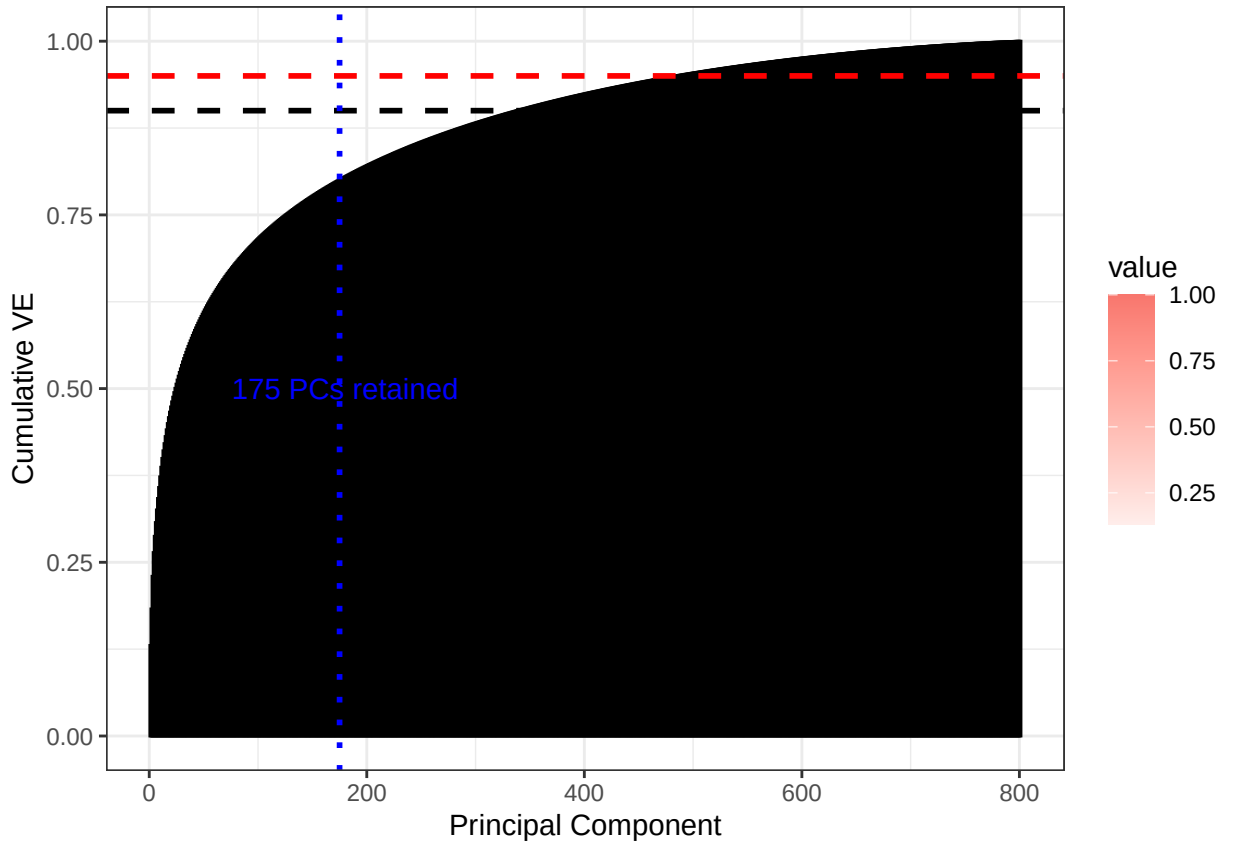
```
## [1] 175
```

```
dr_images = data.frame(
  image_path = images$image_path,
  PCA$x[,1:num_pcs]
)

# Plot
pca_cumVE = barplot(ve)+
```

```
  xlab("Principal Component")+
  ylab("Cumulative VE")+
  ylim(0, 1)+
  geom_hline(aes(yintercept = 0.9), linewidth = 1, linetype = "dashed") +
  geom_hline(aes(yintercept = 0.95), linewidth = 1, linetype = "dashed", color = "red") +
  geom_vline(xintercept = num_pcs, linetype = "dotted", color = "blue", size = 1) +
  annotate("text", x = num_pcs + 5, y = 0.5, label = paste0(num_pcs, " PCs retained"), color = "blue")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
pca_cumVE
```



Visualize Compressed Images

```
## Abomasnow
pokemon = which(stats$name == "Abomasnow")
# Original image
abomasnow_og = plotImg(getImg(images[pokemon, -1])) + labs(title = "Original: Abomasnow")
# Compressed image
scores = (as.numeric(images[pokemon, -1]) - PCA$center) %*% PCA$rotation
scores_95VE = scores[, 1:num_pcs, drop = FALSE] %*% t(PCA$rotation[, 1:num_pcs]) + PCA$center
# scores_95VE_norm = (scores_95VE - min(scores_95VE)) / (max(scores_95VE) - min(scores_95VE))
```

```
scores_95VE_clip = pmax(pmin(scores_95VE, 1), 0)
abomasnow_compressed = plotImg(getImg(scores_95VE_clip)) +
  labs(title = paste("PCA Reconstructed (", num_pcs, " PCs)", sep=""))
# Side by side
abomasnow_og + abomasnow_compressed
```

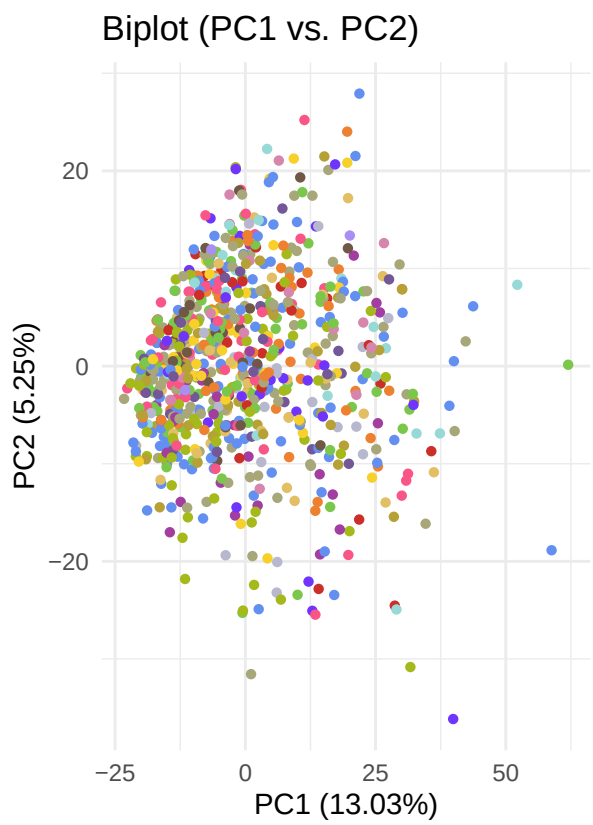

```
## Gardevoir
pokemon = which(stats$name == "Gardevoir")
# Original image
gardevoir_og = plotImg(getImg(images[pokemon, -1])) + labs(title = "Original: Gardevoir")
# Compressed image
scores = (as.numeric(images[pokemon, -1]) - PCA$center) %*% PCA$rotation
scores_95VE = scores[, 1:num_pcs, drop = FALSE] %*% t(PCA$rotation[, 1:num_pcs]) + PCA$center
# scores_95VE_norm = (scores_95VE - min(scores_95VE)) / (max(scores_95VE) - min(scores_95VE))
scores_95VE_clip = pmax(pmin(scores_95VE, 1), 0)
gardevoir_compressed = plotImg(getImg(scores_95VE_clip)) +
  labs(title = paste("PCA Reconstructed (", num_pcs, " PCs)", sep=""))
# Side by side
gardevoir_og + gardevoir_compressed
```

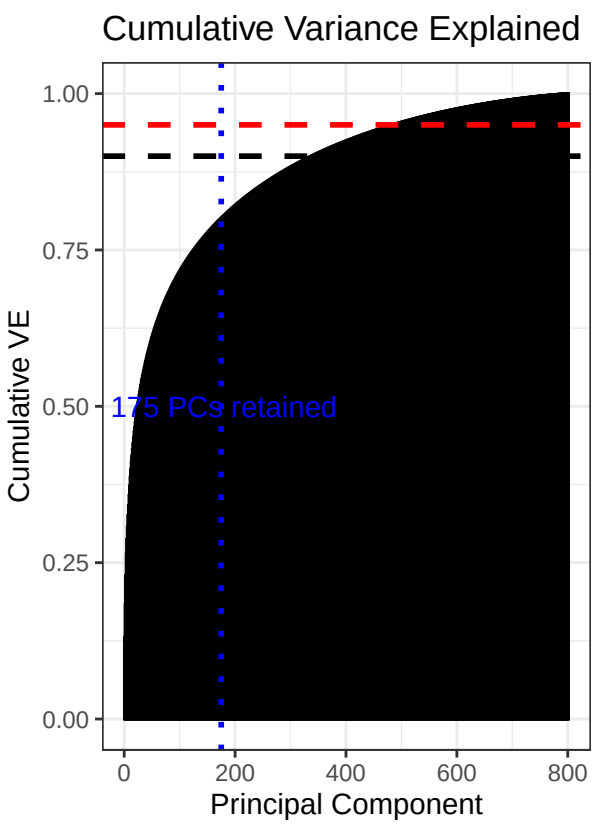
Original: Gardevoir PCA Reconstructed (175 PCs)

```
## Save
# save(gardevoir_og, gardevoir_compressed, file = "Figures/appendix_B.RData")
```

Save Report Images

```
## Cumulative VE + Biplot
mod_cumVE = pca_cumVE + theme(legend.position = "none") + labs(title = "Cumulative Variance Explained")
mod_biplot = pca_biplot + labs(title = "Biplot (PC1 vs. PC2)") + theme_minimal() + theme(legend.positio
cumVE_biplot = mod_biplot + mod_cumVE
  # mod_cumVE + mod_biplot
cumVE_biplot
```



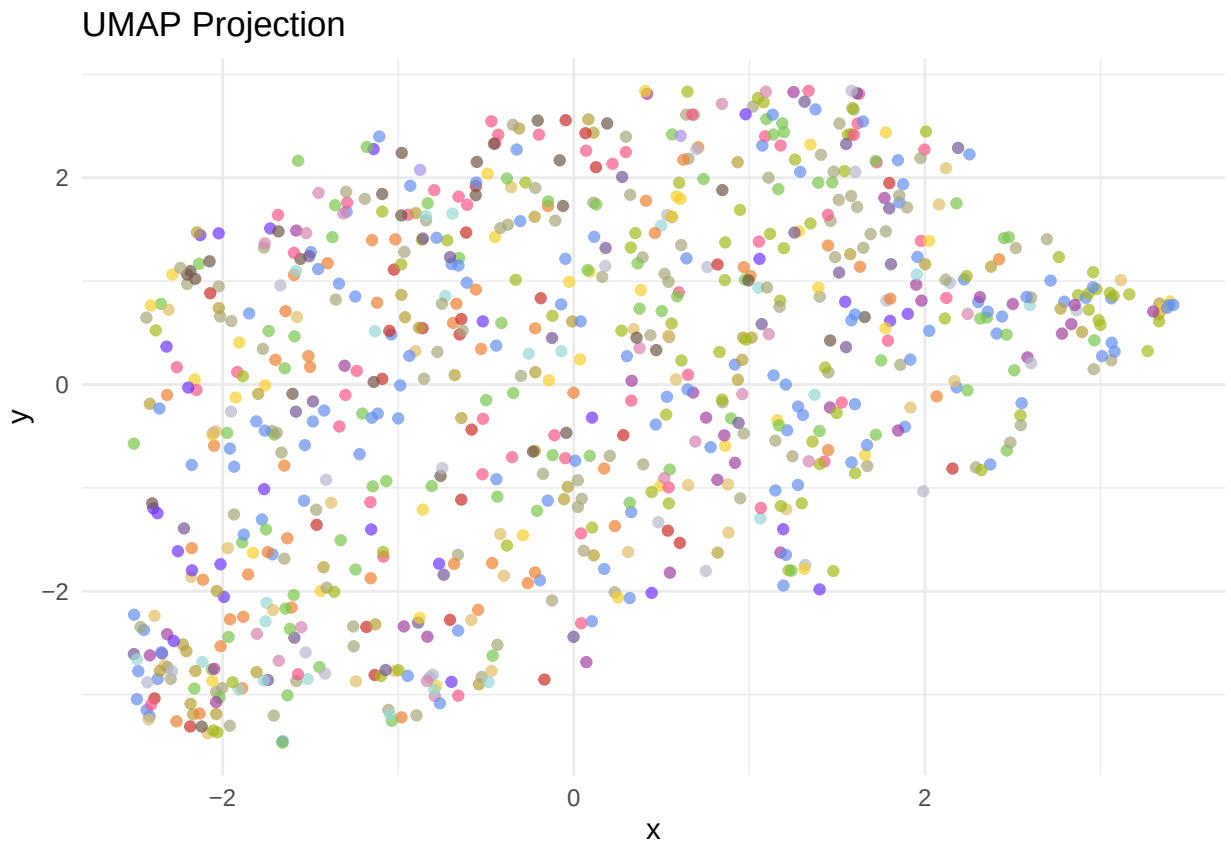



```
ggsave("Figures/scree_biplot.png", plot = cumVE_biplot, width = 7, height = 4, units = "in")

## Test
library(png)
library(grid)
img = readPNG("Figures/scree_biplot.png")
g = rasterGrob(img, interpolate = TRUE)

cumVE_biplot_img = ggplot() +
  annotation_custom(g, xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = Inf) +
  theme_void()

## Save
save(cumVE_biplot_img, abomasnow_og, abomasnow_compressed, file = "Figures/img_dim_red.RData")

## Reset
global_vars = c(global_vars, "dr_images", "type_colours")
rm(list = setdiff(ls(), global_vars)) # Remove all vars except for global_vars
```

UMAP

```
## UMAP
# library(umap)
#
# UMAP_imgs = umap(images[,-1])
```

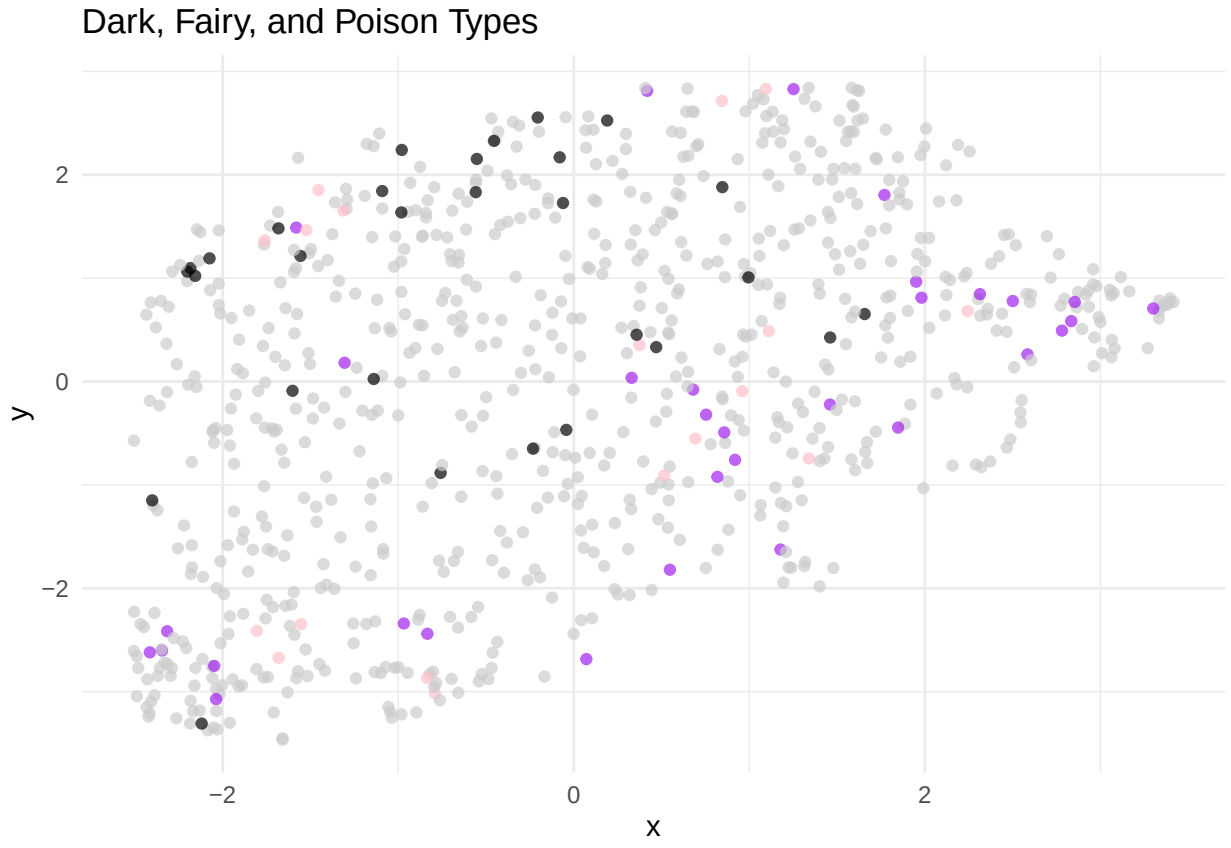
```r
## Load object
load("DimensionReduction/umap_pokemon.rds")

# Plot
type_colours = type_colours[match(levels(stats$type1), names(type_colours))] # Colours

umap_df = data.frame(x = UMAP_imgs$layout[,1], y = UMAP_imgs$layout[,2], type = stats$type1)
umap_plt = ggplot(umap_df, aes(x, y, color = type)) +
  geom_point(alpha = 0.7) +
  theme_minimal() +
  scale_colour_manual(values = type_colours) +
  ggtitle("UMAP Projection") +
  theme(legend.position = "none")#"left")
umap_plt
```




```r
## Save object
# save(UMAP_imgs, file = "umap_pokemon.rds")

# Note: Too large to save on GitHub.
```

```r
# Define key types to highlight
highlight_types = c("dark", "fairy", "poison")
# Clustered types: Dark, Fairy, Poison
# Less-well clustered: Flying

# Reassign colors: keep highlights, grey out others
```

```
highlight_colours = type_colours
highlight_colours["dark"] = "black"
highlight_colours["fairy"] = "pink"
highlight_colours["poison"] = "purple"
highlight_colours[!(names(highlight_colours) %in% highlight_types)] = "grey80"

# Create updated UMAP dataframe
umap_df = data.frame(
  x = UMAP_imgs$layout[,1],
  y = UMAP_imgs$layout[,2],
  type = stats$type1
)

# Plot with custom color mapping
umap_plt_col = ggplot(umap_df, aes(x, y, color = type)) +
  geom_point(alpha = 0.7, size = 1.5) +
  scale_colour_manual(values = highlight_colours) +
  theme_minimal() +
  ggtitle("Dark, Fairy, and Poison Types") +
  theme(legend.position = "none")

umap_plt_col
```




```
## Save
save(umap_plt, umap_plt_col, file = "Figures/umap_plt.RData")
```

```r
## Reset
global_vars = c(global_vars, "UMAP_imgs")
rm(list = setdiff(ls(), global_vars)) # Remove all vars except for global_vars
```

Clustering

Setup

```r
set.seed(2201)
# Load libraries and setup
library(cluster)
library(factoextra)
```

```
## Warning: package 'factoextra' was built under R version 4.4.3
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.4.3
```

```
## Loading required package: lattice
```

```r
library(ggplot2)
library(ggfortify)
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.4.2
```

```
## Warning: package 'readr' was built under R version 4.4.2
```

```
## Warning: package 'lubridate' was built under R version 4.4.2
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v lubridate 1.9.4     v stringr   1.5.1
## v purrr     1.0.2     v tibble    3.2.1
## v readr     2.1.5
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(VIM)
```

```
## Warning: package 'VIM' was built under R version 4.4.3
```

```
## Loading required package: colorspace
## VIM is ready to use.
##
## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues
##
## Attaching package: 'VIM'
##
## The following object is masked from 'package:datasets':
##
##     sleep
```

```r
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 4.4.2
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
load("Data/pokemon.RData")
load("Data/dr_pokemon2.RData")
load("DimensionReduction/umap_pokemon.rds")

stats_numeric <- stats %>%
  dplyr::select(-c(abilities, capture_rate, classfication, japanese_name, name,
           name.simple, type1, type2, image_path, has_img))

#Imputation on missing data using KNN
stats_numeric_impKNN <- kNN(stats_numeric)
#Remove cols with variance = 0
# Check the variance of each column
variances <- apply(stats_numeric_impKNN, 2, var)
zero_var_col <- which(variances == 0)
stats_numeric_impKNN <- stats_numeric_impKNN[, -zero_var_col]
```

Helper functions

```r
# Helper function lecture 7
scatterplot = function(X, M, cluster, label = FALSE){
  X_df <- data.frame(X, cluster = as.factor(cluster))
  M_df <- data.frame(M)

  if (length(unique(cluster)) == 1) {
    plt <- ggplot(X_df, aes(x = PC1, y = PC2)) +
      geom_point() +
      geom_point(data = M_df, aes(x = PC1, y = PC2), shape = 4, size = 4, color = "red") +
      labs(title = "Scatterplot of Pokemon Clusters")

    if (label) {
      plt <- plt + geom_text(aes(label = stats$name), nudge_x = 0.1, size = 3)
    }
    return(plt)
  }
  else {
    ggplot(X_df, aes(x = PC1, y = PC2, color = cluster)) +
      geom_point(alpha = 0.7) +
      geom_point(data = M_df, aes(x = PC1, y = PC2), shape = 4, size = 4, color = "black") +
      scale_color_manual(values = rainbow(length(unique(cluster)))) +
      theme_minimal() +
      labs(title = "K-Means Clustering of Pokemon (PC1 vs PC2)", x = "PC1", y = "PC2") +
      theme(legend.position = "right")
  }
```

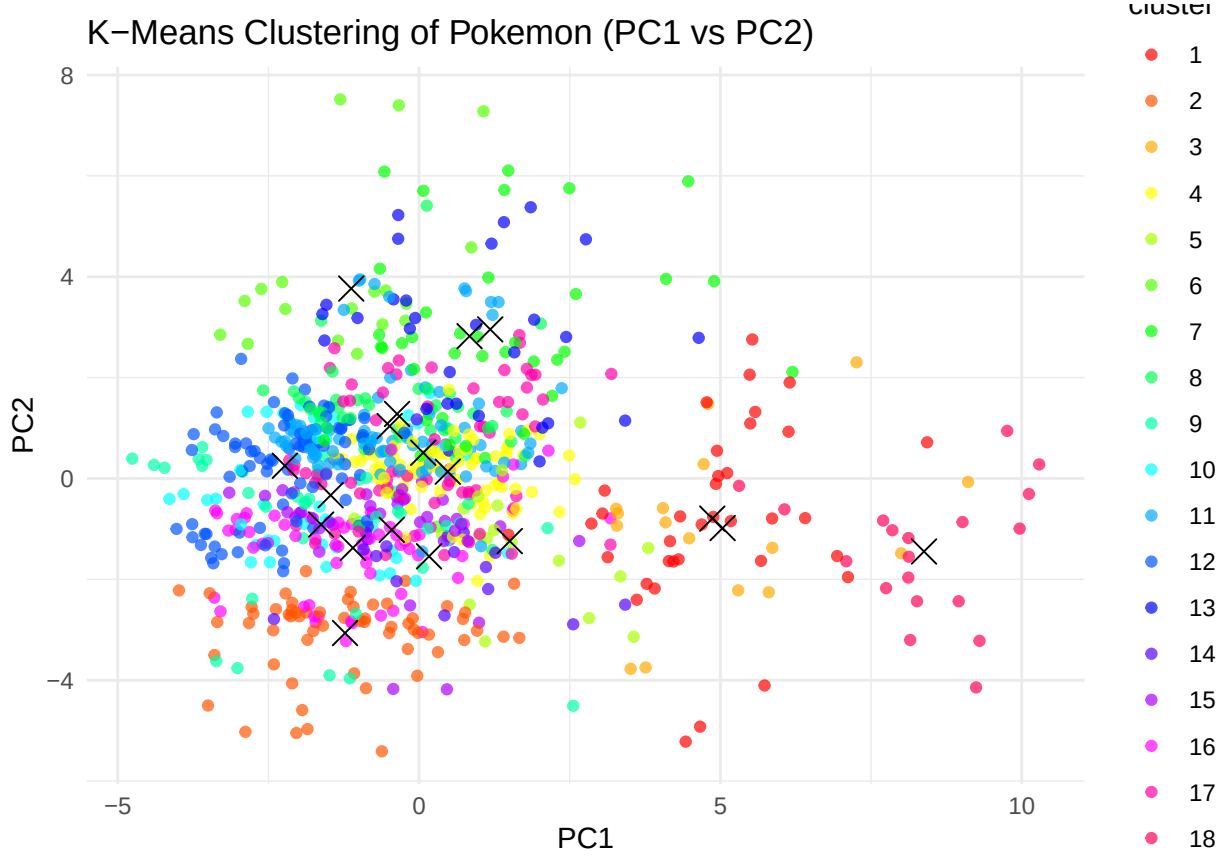
```
}
# Reference lecture 7 slide 10, and modifications from hw 3
weighted_kmeans <- function(X, K) {
  X <- as.matrix(X)
  n <- nrow(X)

  # Step 1: Initialize first centroid
  set.seed(2201)
  centroids <- X[sample(1:n, 1), , drop = FALSE]

  # Step 2: Compute the initial weights distance
  distances <- as.matrix(dist(rbind(centroids, X)))[2:(n+1), 1]
  weights <- distances^2 / sum(distances^2)

  # Step 3: Initialize the remaining K-1 centroids
  for (i in 2:K) {
    P <- weights / sum(weights)
    pick <- sample(1:n, 1, prob = P)
    new_centroid <- X[pick, , drop = FALSE]
    centroids <- rbind(centroids, new_centroid)

    # Compute new weights based on the distance from the new centroid
    new_distances <- as.matrix(dist(rbind(new_centroid, X)))[2:(n+1), 1]
    weights <- new_distances^2 / sum(new_distances^2)
  }

  converged <- FALSE
  clusters <- rep(0, n)
  while (!converged) {
    # Assign points to the nearest centroid based on weighted distances
    distances <- as.matrix(dist(rbind(centroids, X)))
    distances <- distances[1:K, (K+1):(n+K)]

    # Weight the distances using the computed weights (soft clustering)
    weighted_distances <- distances * weights

    # Assign each point to the nearest centroid based on weighted distances
    new_clusters <- apply(weighted_distances, 2, which.min)

    # Update centroids using weighted averages
    new_centroids <- matrix(NA, ncol = ncol(X), nrow = K)
    for (k in 1:K) {
      cluster_points <- X[new_clusters == k, ]
      cluster_weights <- weights[new_clusters == k]
      if (length(cluster_points) > 0) {
        new_centroids[k, ] <- colSums(cluster_points * cluster_weights) / sum(cluster_weights)
      }
    }

    # Check for convergence (if centroids don't change)
    if (all(centroids == new_centroids)) {
      converged <- TRUE
    } else {
```

```
      centroids <- new_centroids
      clusters <- new_clusters
    }
  }


  return(list(centroids = centroids, clusters = clusters))
}

Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
```

Clustering on Stats Dataset

```
# PCA on stats
pca_stats <- prcomp(stats_numeric_impKNN, center = TRUE, scale. = TRUE)
pca_stats_df <- as.data.frame(pca_stats$x)
colnames(pca_stats_df) <- paste0("PC", 1:ncol(pca_stats_df))
var_explained <- summary(pca_stats)$importance[2, ]  # Proportion of variance explained
cumulative_var <- cumsum(var_explained)
# Keep 20 PCs (90% VE)
pca_stats_df <- pca_stats_df[,1:20]
```

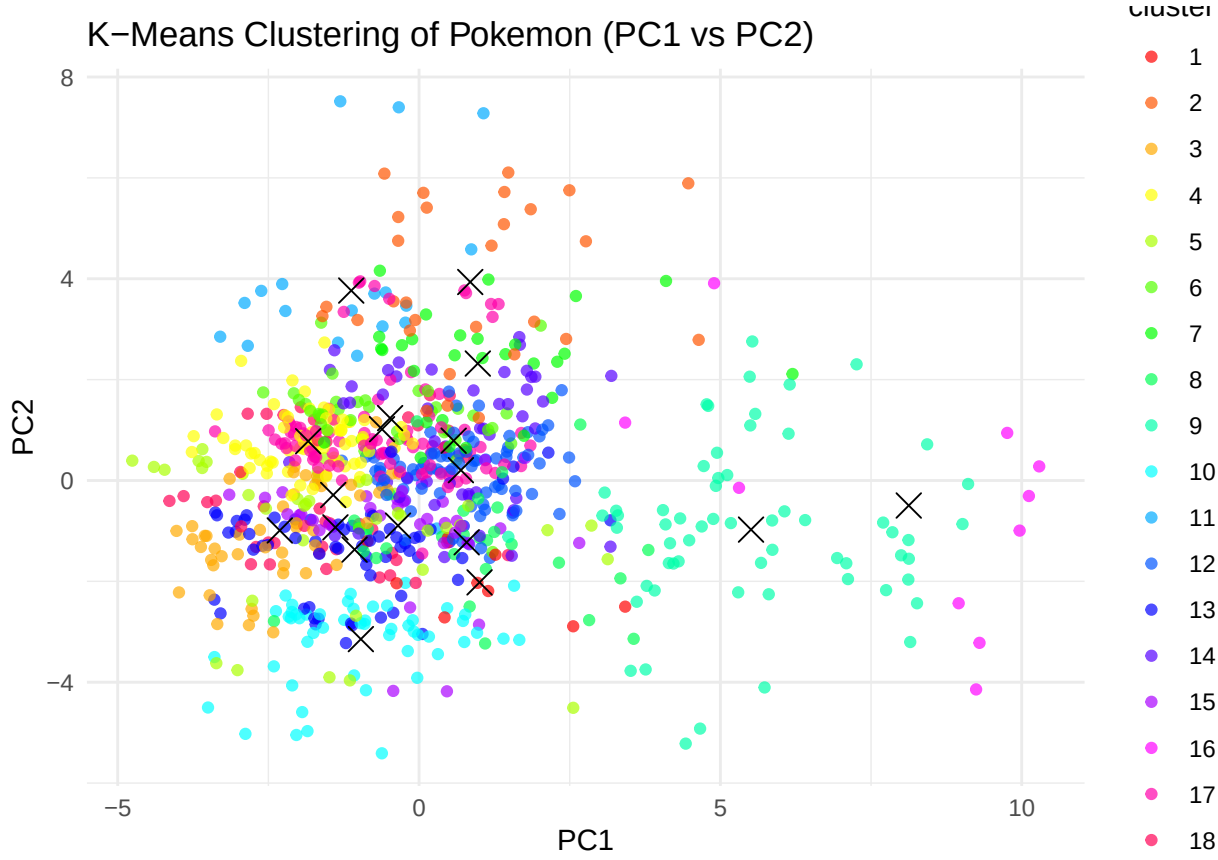
```
## Kmeans on stats k = 18
k_types <- length(unique(stats$type1))
kmeans_stats <- kmeans(pca_stats_df, centers = k_types)
scatterplot(pca_stats_df, kmeans_stats$centers, kmeans_stats$cluster)
```



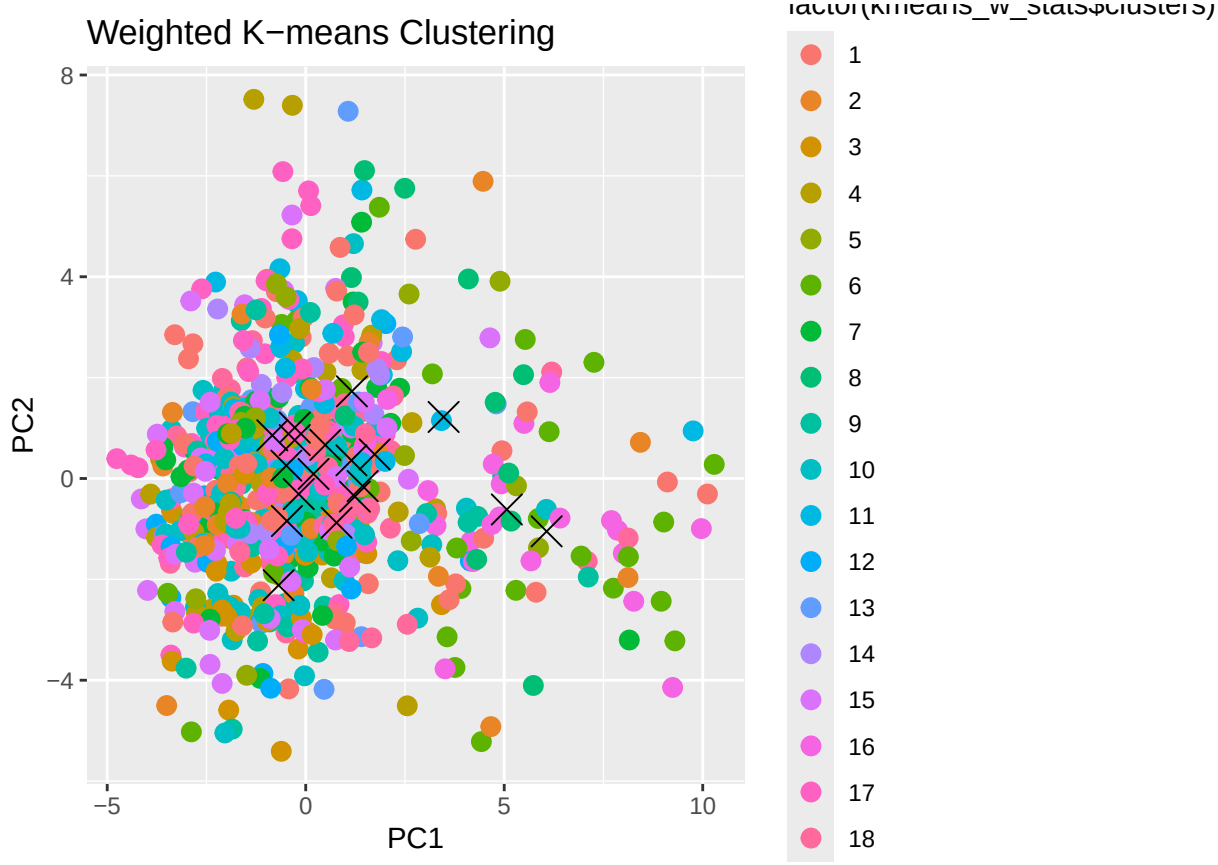

```
## Comparing standard kmeans with kmeans++ and weighted kmeans

#Ref: lecture 7
# Randomly select the first centroid
n <- nrow(pca_stats_df)
M <- pca_stats_df[sample(1:n, 1), , drop = FALSE]

for (i in 2:k_types) {
  # Dist from each point to the nearest centroid
  D <- as.matrix(dist(rbind(M, pca_stats_df)))[2:(n+1), 1]

  # Probability for each point to be chosen as the next centroid
  P <- D^2 / sum(D^2)

  # Select the next centroid based on P
  pick <- sample(1:n, 1, prob = P)
  M <- rbind(M, pca_stats_df[pick, , drop = FALSE])
}

kmeans_stats_pp <- kmeans(pca_stats_df, centers = M, algorithm = "Lloyd")
```

```
## Warning: did not converge in 10 iterations
```

```
scatterplot(pca_stats_df, kmeans_stats_pp$centers, kmeans_stats_pp$cluster)
```

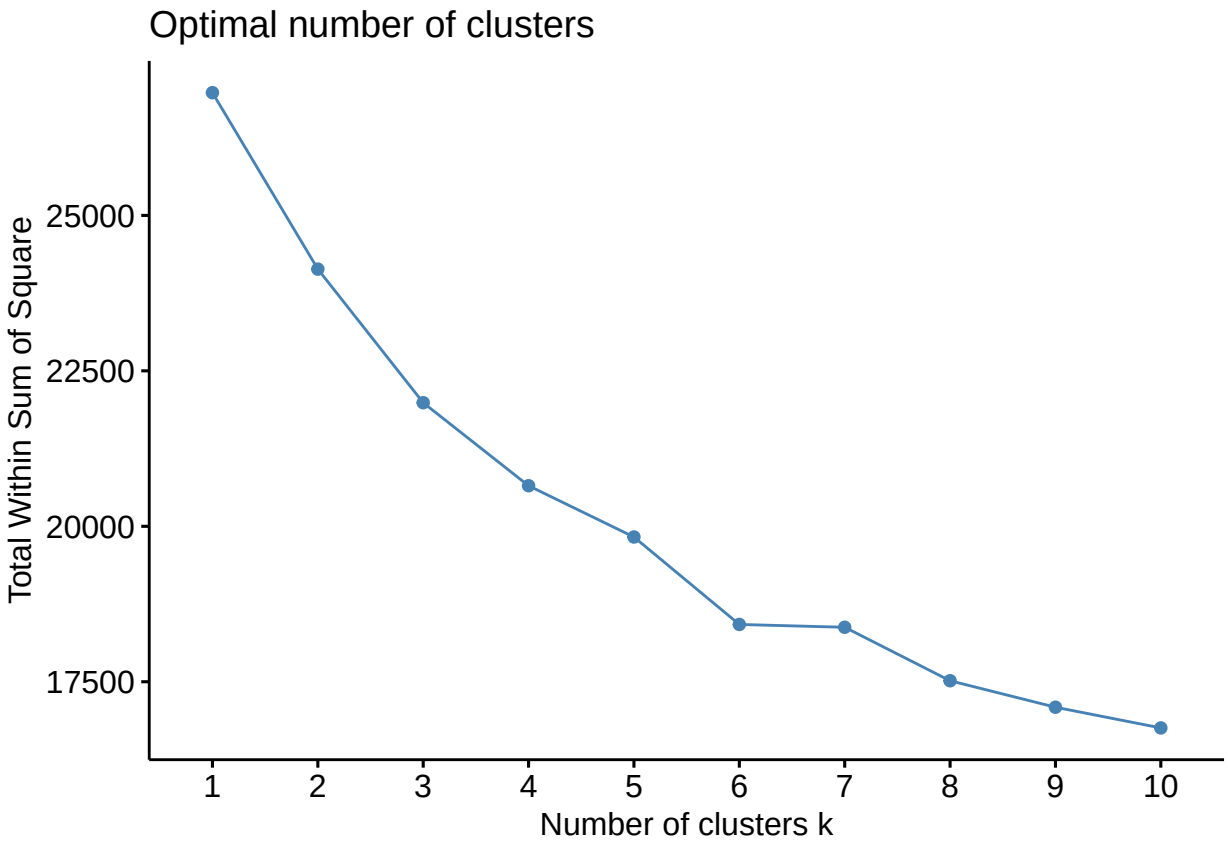



```r
# Weighted kmeans
K <- 18
kmeans_w_stats <- weighted_kmeans(pca_stats_df, K)
colnames(kmeans_w_stats$centroids) <- colnames(pca_stats_df)

ggplot(data.frame(pca_stats_df), aes(PC1, PC2, color = factor(kmeans_w_stats$clusters))) +
  geom_point(size = 3) +
  geom_point(data = data.frame(kmeans_w_stats$centroids), aes(PC1, PC2),
             color = "black", shape = 4, size = 5) +
  labs(title = "Weighted K-means Clustering", x = "PC1", y = "PC2")
```

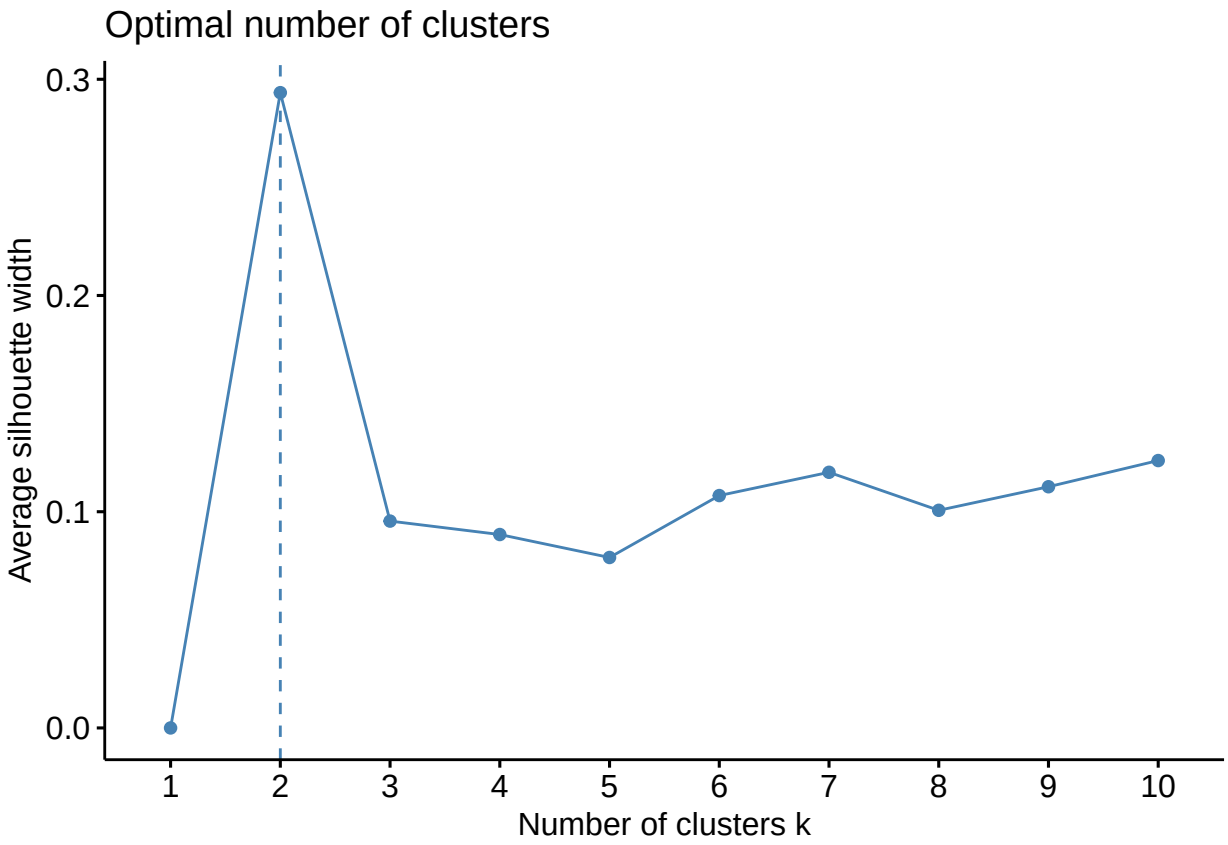



```
## Tune for optimal K

# Elbow method
fviz_nbclust(pca_stats_df, kmeans, method = "wss", algorithm = "Lloyd")
```

```
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
```

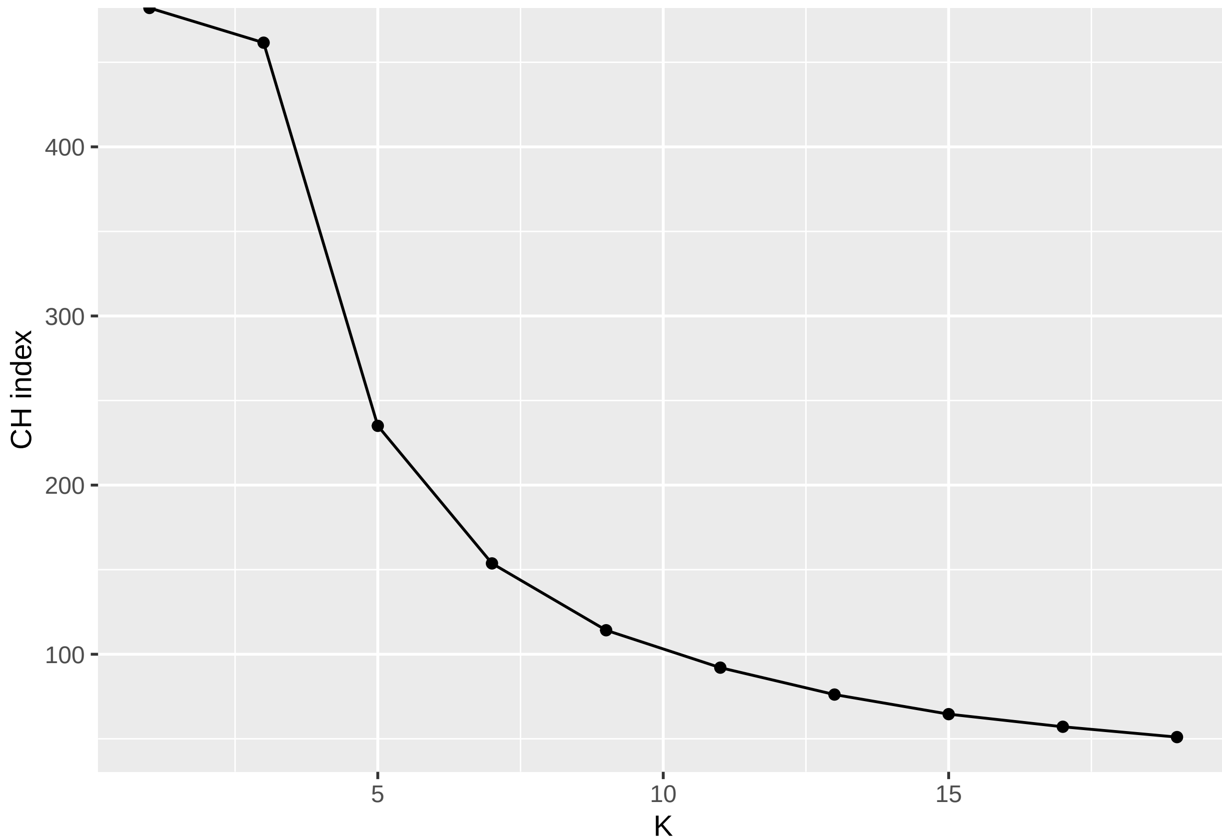



```
# Silhouette method
fviz_nbclust(pca_stats_df, kmeans, method = "silhouette")
```




```
#CH index
CHs = c()
Ks = seq(1, 20, 2)

for(K in Ks){
  KM = kmeans(pca_stats_df, centers = k_types, nstart = 25)

  # Between-cluster sum of squares
  B = KM$betweenss

  # Within-cluster sum of squares
  W = KM$tot.withinss

  # Number of data points
  n = nrow(pca_stats_df)

  # Calculate the Calinski-Harabasz index
  CH = (B / (K - 1)) / (W / (n - K))

  # Append the CH index for the current K to the list
  CHs = c(CHs, CH)
}
df = data.frame(K = Ks, CH = CHs)
ggplot(df, aes(K, CH)) +
  geom_point() +
  geom_line() +
```

```r
  ylab("CH index")
```

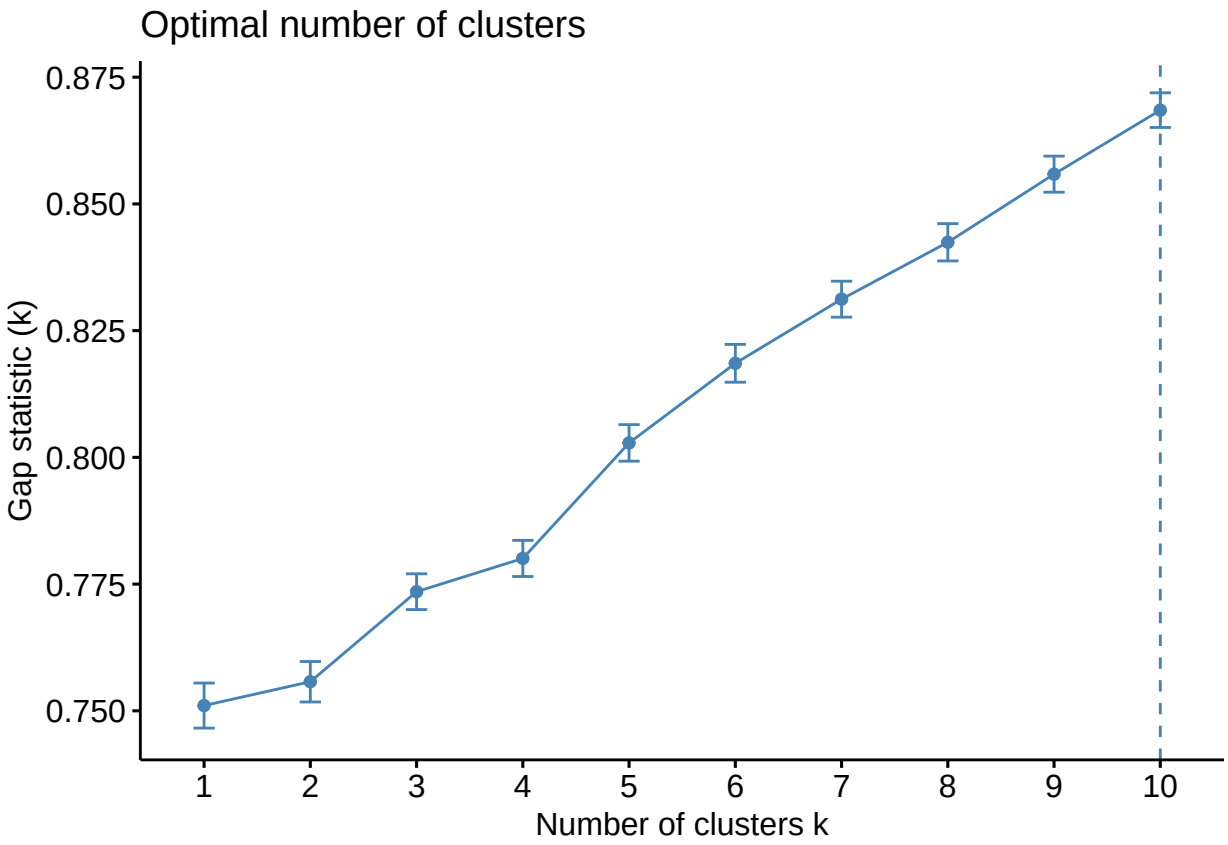



```r
# Gap statistics
gapstat_stats = clusGap(pca_stats_df, FUN = kmeans, nstart = 50, K.max = 10, B = 50)
```

```
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
```

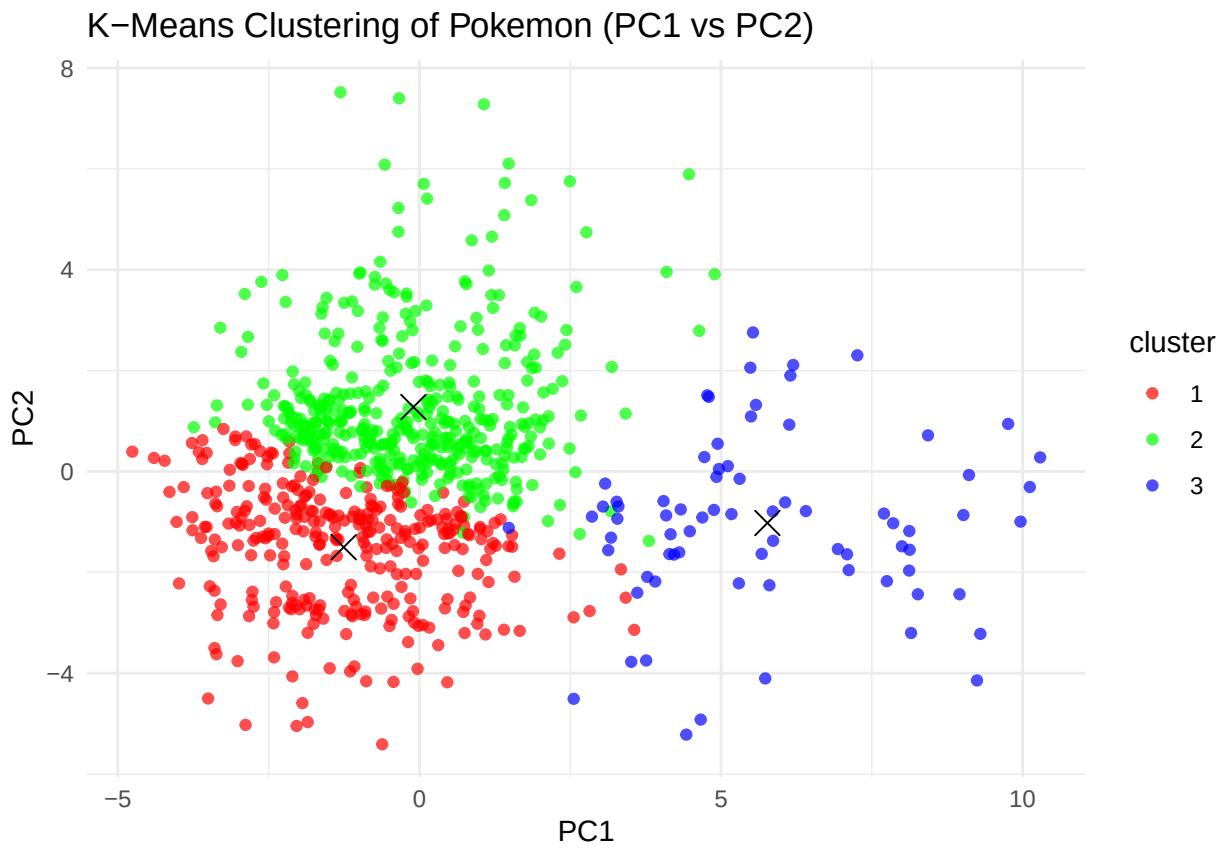
```
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
```

```
fviz_gap_stat(gapstat_stats, maxSE = list(method = "Tibs2001SEmax", SE.factor = 1))
```




```
# RESULTS

# Elbow method: Optimal K = 3

# Silhouette method: Optimal K = 2

# CH Index: Optimal K = 1

# Gap stats: Optimal K = 10
```

```
# Kmeans on optimal K = 3
kmeans_stats_optimal <- kmeans(pca_stats_df, centers = 3)
scatterplot(pca_stats_df, kmeans_stats_optimal$centers, kmeans_stats_optimal$cluster)
```




```
# Comparing with true labels

comp_labels <- data.frame(type1 = stats$type1)
comp_labels$cluster <- as.factor(kmeans_stats$cluster)

# Map clusters to true labels
cluster_label_mapping <- comp_labels %>%
  mutate(cluster = factor(kmeans_stats$cluster)) %>%
  group_by(cluster) %>%
  summarise(mode_type = Mode(type1), .groups = 'drop')


# Map cluster labels to the most common actual type
labelled_clusters <- comp_labels %>%
  left_join(cluster_label_mapping, by = "cluster") %>%
  select(type1, cluster, mode_type)

# Calculate accuracy
#mean(labelled_clusters$type1 == labelled_clusters$mode_type)
```

```
comp_labels <- data.frame(type1 = stats$type1)
comp_labels$cluster <- as.factor(kmeans_stats_pp$cluster)

# Map clusters to true labels
cluster_label_mapping <- comp_labels %>%
  mutate(cluster = factor(kmeans_stats_pp$cluster)) %>%
```

```
  group_by(cluster) %>%
  summarise(mode_type = Mode(type1), .groups = 'drop')


# Map cluster labels to the most common actual type
labelled_clusters <- comp_labels %>%
  left_join(cluster_label_mapping, by = "cluster") %>%
  select(type1, cluster, mode_type)

# Calculate accuracy
#mean(labelled_clusters$type1 == labelled_clusters$mode_type)
```

```
comp_labels <- data.frame(type1 = stats$type1)
comp_labels$cluster <- as.factor(kmeans_w_stats$cluster)

# Map clusters to true labels
cluster_label_mapping <- comp_labels %>%
  mutate(cluster = factor(kmeans_w_stats$cluster)) %>%
  group_by(cluster) %>%
  summarise(mode_type = Mode(type1), .groups = 'drop')


# Map cluster labels to the most common actual type
labelled_clusters <- comp_labels %>%
  left_join(cluster_label_mapping, by = "cluster") %>%
  select(type1, cluster, mode_type)

# Calculate accuracy  (13 types)
#mean(labelled_clusters$type1 == labelled_clusters$mode_type)
```

```
comp_labels$cluster <- as.factor(kmeans_stats_optimal$cluster)

# Map clusters to true labels
cluster_label_mapping <- comp_labels %>%
  mutate(cluster = factor(kmeans_stats_optimal$cluster)) %>%
  group_by(cluster) %>%
  summarise(mode_type = Mode(type1), .groups = 'drop')


# Map cluster labels to the most common actual type
labelled_clusters <- comp_labels %>%
  left_join(cluster_label_mapping, by = "cluster") %>%
  select(type1, cluster, mode_type)

# Calculate accuracy
# mean(labelled_clusters$type1 == labelled_clusters$mode_type)
```

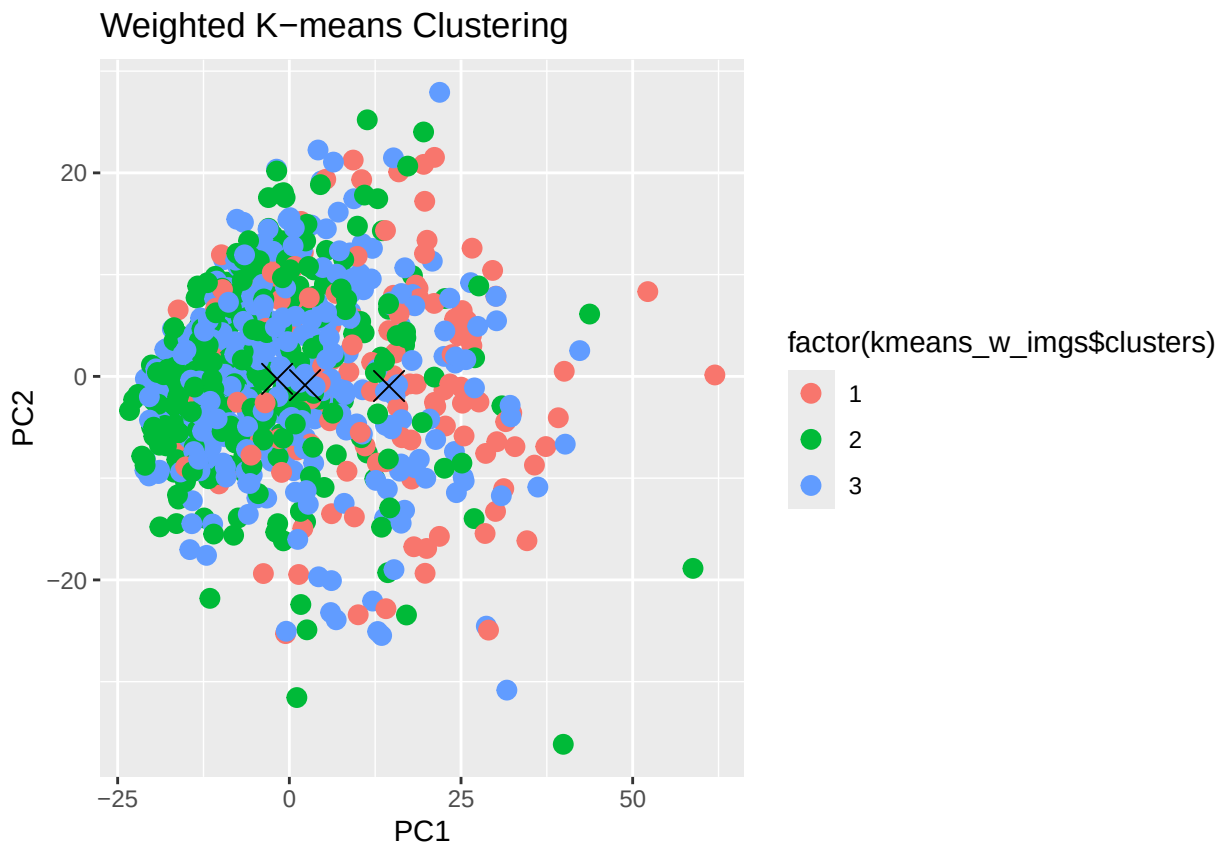
Clustering on Image Dataset

```
## Kmeans on images k = 18
pca_imgs_df<- dr_images[,-1]  # Remove image_path column
k_types <- length(unique(stats$type1))
kmeans_imgs <- kmeans(pca_imgs_df, centers = k_types)
```

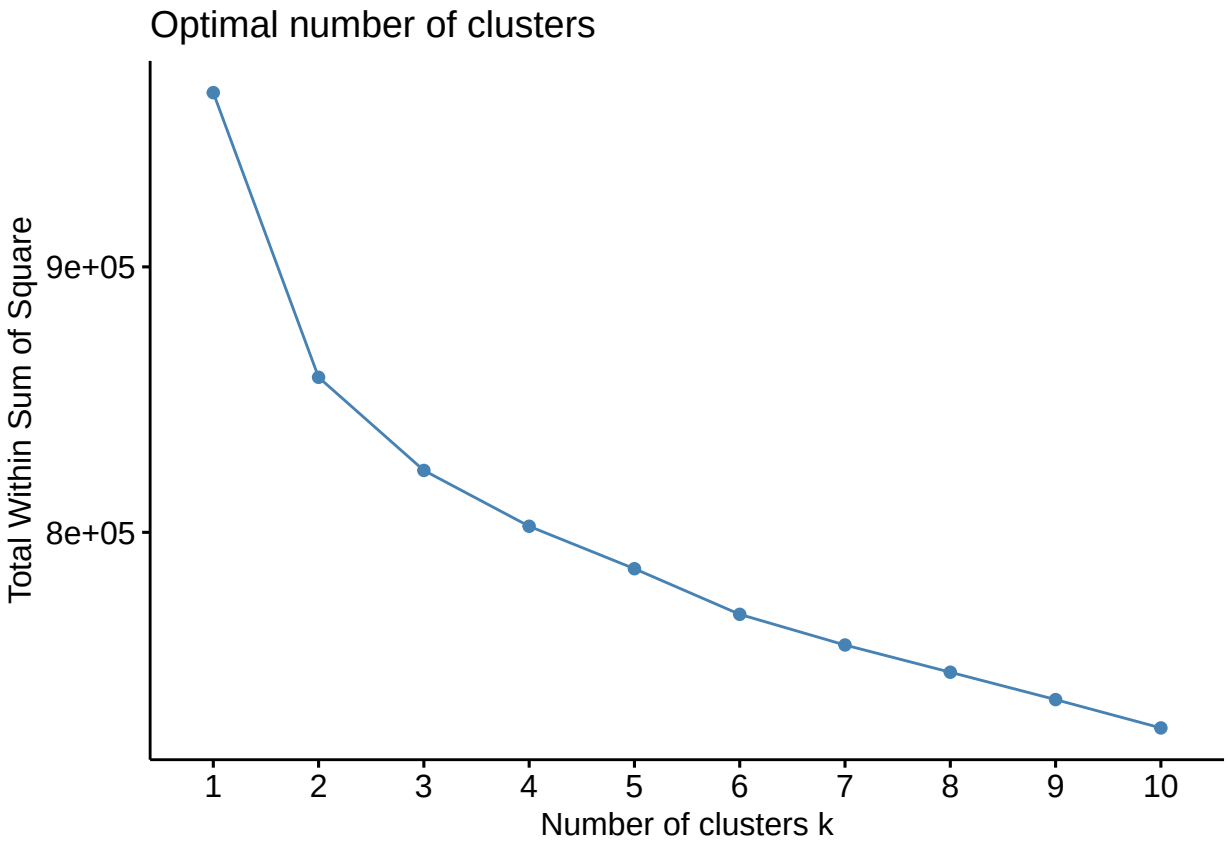
```r
## Comparing standard kmeans with kmeans++ and weighted kmeans

#Ref: lecture 7
# Randomly select the first centroid
n <- nrow(pca_imgs_df)
M <- pca_imgs_df[sample(1:n, 1), , drop = FALSE]

# Select remaining k-1 centroids using weighted probability
for (i in 2:k_types) {
  # Compute distance from each point to the nearest centroid
  D <- as.matrix(dist(rbind(M, pca_imgs_df)))[2:(n+1), 1]

  # Probability for each point to be chosen as the next centroid
  P <- D^2 / sum(D^2)

  # Select the next centroid based on P
  pick <- sample(1:n, 1, prob = P)
  M <- rbind(M, pca_imgs_df[pick, , drop = FALSE])
}

kmeans_imgs_pp <- kmeans(pca_imgs_df, centers = M, algorithm = "Lloyd")
```

```
## Warning: did not converge in 10 iterations
```

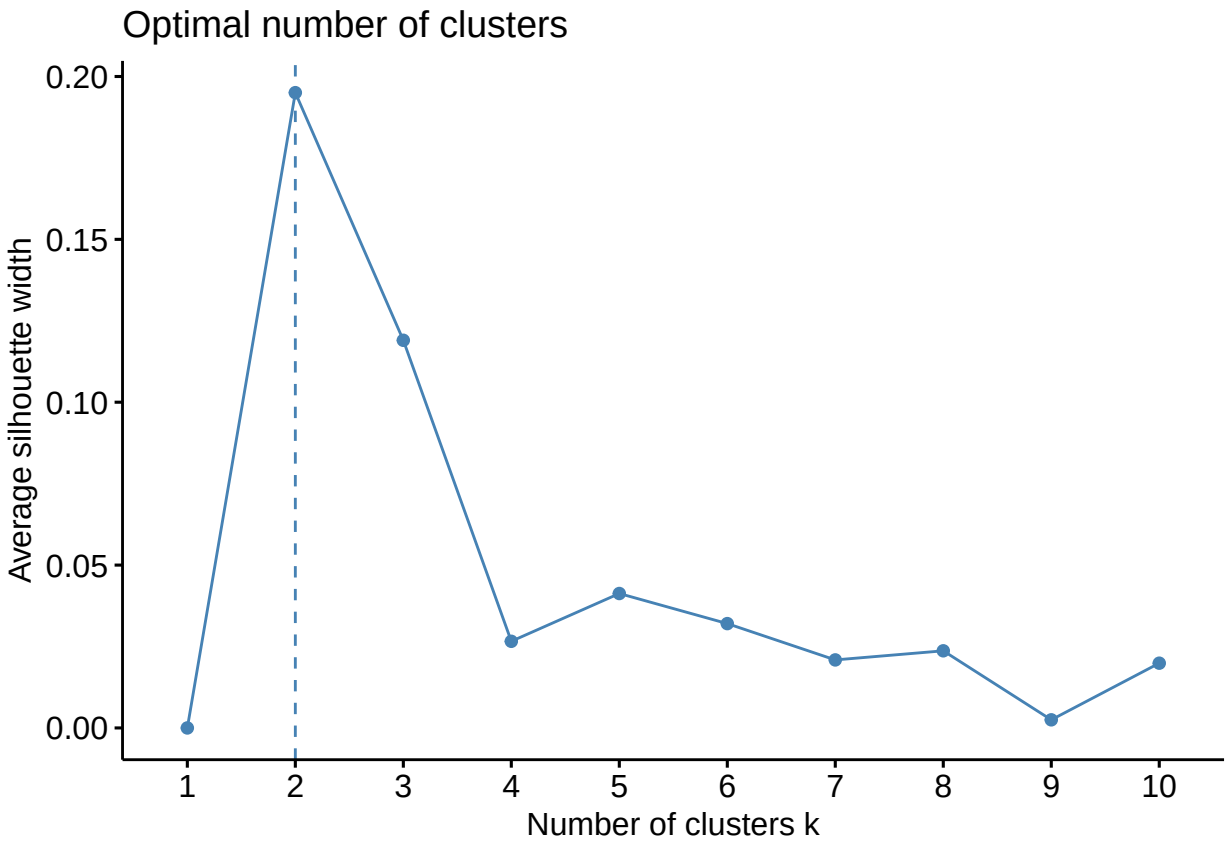
```r
#Weighted K-means

K <- 3
kmeans_w_imgs <- weighted_kmeans(pca_imgs_df, K)
colnames(kmeans_w_imgs$centroids) <- colnames(pca_imgs_df)

ggplot(data.frame(pca_imgs_df), aes(PC1, PC2, color = factor(kmeans_w_imgs$clusters))) +
  geom_point(size = 3) +
  geom_point(data = data.frame(kmeans_w_imgs$centroids), aes(PC1, PC2),
             color = "black", shape = 4, size = 5) +
  labs(title = "Weighted K-means Clustering", x = "PC1", y = "PC2")
```




```
## Tune for optimal k

# Elbow method
fviz_nbclust(pca_imgs_df, kmeans, method = "wss")
```



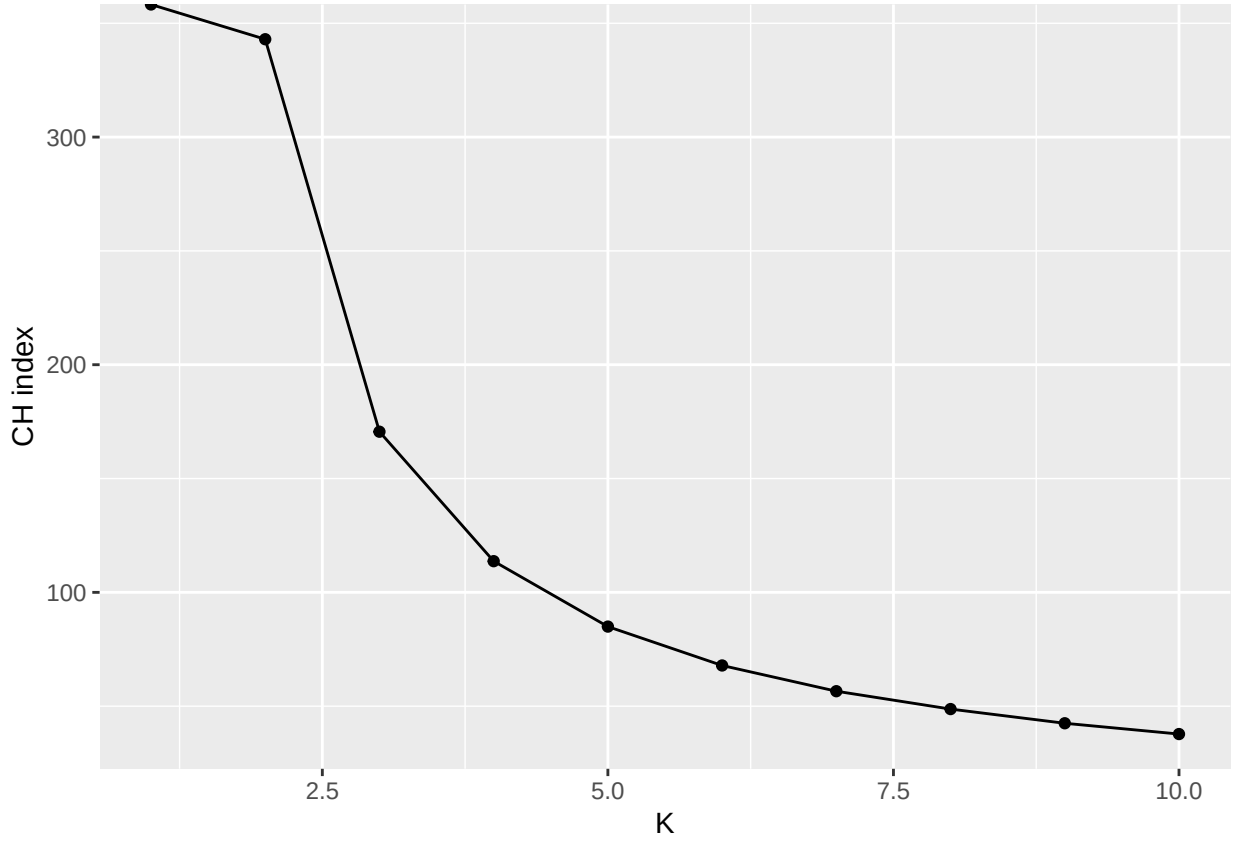

```
# Silhouette method
fviz_nbclust(pca_imgs_df, kmeans, method = "silhouette")
```




```
# CH Index
CHs = c()
Ks = seq(1, 10, 1)

for(K in Ks){
  KM = kmeans(pca_imgs_df, centers = k_types, nstart = 25)

  # Between-cluster sum of squares
  B = KM$betweenss

  # Within-cluster sum of squares
  W = KM$tot.withinss

  # Number of data points
  n = nrow(pca_imgs_df)

  # Calculate the Calinski-Harabasz index
  CH = (B / (K - 1)) / (W / (n - K))

  # Append the CH index for the current K to the list
  CHs = c(CHs, CH)
}
```

```
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
```

```
df = data.frame(K = Ks, CH = CHs)
ggplot(df, aes(K, CH)) +
  geom_point() +
  geom_line() +
  ylab("CH index")
```

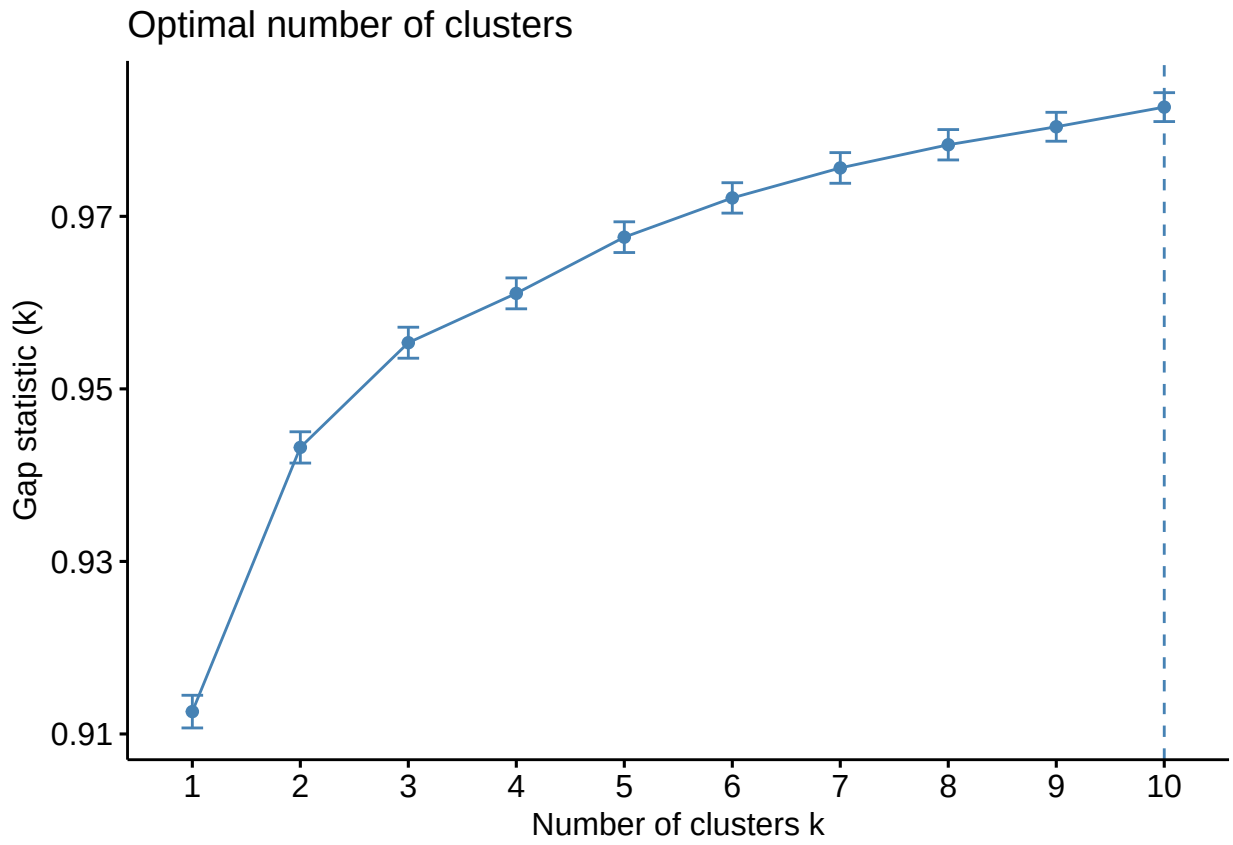



```
#Gap statistics
gapstat_img = clusGap(pca_imgs_df, FUN = kmeans, nstart = 50, K.max = 10, B = 50)
```

```
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
```

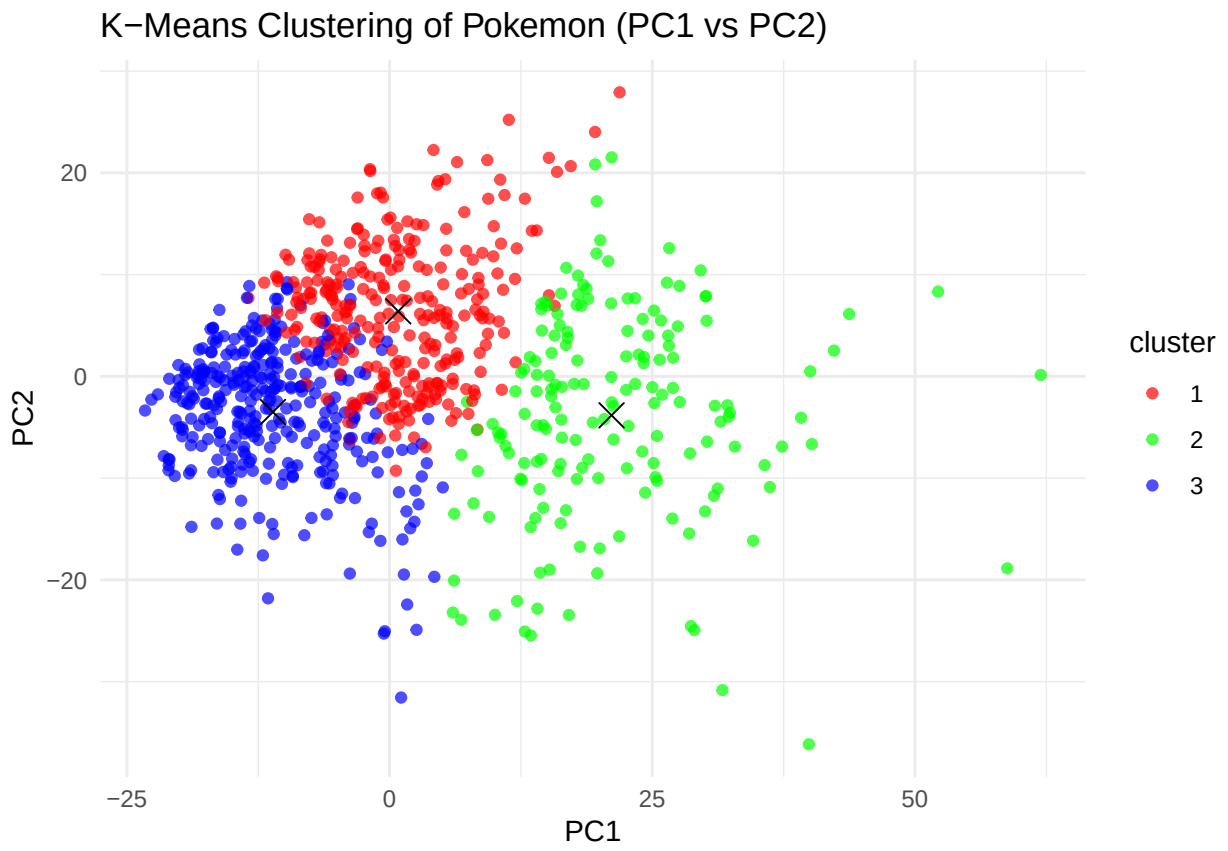
```
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
## Warning: did not converge in 10 iterations
```

```r
fviz_gap_stat(gapstat_img, maxSE = list(method = "Tibs2001SEmax", SE.factor = 1))
```




```
# RESULTS

# Elbow method: Optimal K = 3

# Silhouette method: Optimal K = 2

# CH Index: Optimal K = 1

# Gap statistics: Optimal K = 10
```

```
#Kmeans on optimal K = 3
kmeans_imgs_optimal <- kmeans(pca_imgs_df, centers = 3, nstart = 25)
scatterplot(pca_imgs_df, kmeans_imgs_optimal$centers, kmeans_imgs_optimal$cluster)
```




```
## Comparing with true labels

comp_labels$cluster <- as.factor(kmeans_imgs$cluster)

# Map clusters to true labels
cluster_label_mapping <- comp_labels %>%
  mutate(cluster = factor(kmeans_imgs$cluster)) %>%
  group_by(cluster) %>%
  summarise(mode_type = Mode(type1), .groups = 'drop')

# Map cluster labels to the most common actual type
labelled_clusters <- comp_labels %>%
  left_join(cluster_label_mapping, by = "cluster") %>%
  select(type1, cluster, mode_type)

# Calculate accuracy
#mean(labelled_clusters$type1 == labelled_clusters$mode_type)
```

```
comp_labels$cluster <- as.factor(kmeans_imgs_pp$cluster)

# Map clusters to true labels
cluster_label_mapping <- comp_labels %>%
  mutate(cluster = factor(kmeans_imgs_pp$cluster)) %>%
  group_by(cluster) %>%
  summarise(mode_type = Mode(type1), .groups = 'drop')
```

```r
# Map cluster labels to the most common actual type
labelled_clusters <- comp_labels %>%
  left_join(cluster_label_mapping, by = "cluster") %>%
  select(type1, cluster, mode_type)

# Calculate accuracy
#mean(labelled_clusters$type1 == labelled_clusters$mode_type)
```

```r
comp_labels <- data.frame(type1 = stats$type1)
comp_labels$cluster <- as.factor(kmeans_w_imgs$cluster)

# Map clusters to true labels
cluster_label_mapping <- comp_labels %>%
  mutate(cluster = factor(kmeans_w_imgs$cluster)) %>%
  group_by(cluster) %>%
  summarise(mode_type = Mode(type1), .groups = 'drop')

# Map cluster labels to the most common actual type
labelled_clusters <- comp_labels %>%
  left_join(cluster_label_mapping, by = "cluster") %>%
  select(type1, cluster, mode_type)

# Calculate accuracy
#mean(labelled_clusters$type1 == labelled_clusters$mode_type)
```

```r
comp_labels$cluster <- as.factor(kmeans_imgs_optimal$cluster)

# Map clusters to true labels
cluster_label_mapping <- comp_labels %>%
  mutate(cluster = factor(kmeans_imgs_optimal$cluster)) %>%
  group_by(cluster) %>%
  summarise(mode_type = Mode(type1), .groups = 'drop')


# Map cluster labels to the most common actual type
labelled_clusters <- comp_labels %>%
  left_join(cluster_label_mapping, by = "cluster") %>%
  select(type1, cluster, mode_type)

# Calculate accuracy
#mean(labelled_clusters$type1 == labelled_clusters$mode_type)
```

```r
# Clustering with UMAP

umap_data <- UMAP_imgs$layout
colnames(umap_data) <- c("PC1","PC2") #Change col name later
k_types <- length(unique(stats$type1))
kmeans_imgs_umap <- kmeans(umap_data, centers = k_types, nstart = 25)
```

```r
comp_labels$cluster <- as.factor(kmeans_imgs_umap$cluster)

# Map clusters to true labels
cluster_label_mapping <- comp_labels %>%
  mutate(cluster = factor(kmeans_imgs_umap$cluster)) %>%
```

```
  group_by(cluster) %>%
  summarise(mode_type = Mode(type1), .groups = 'drop')

# Map cluster labels to the most common actual type
labelled_clusters <- comp_labels %>%
  left_join(cluster_label_mapping, by = "cluster") %>%
  select(type1, cluster, mode_type)

# Calculate accuracy
#mean(labelled_clusters$type1 == labelled_clusters$mode_type)
```

```
#Kmeans++ with UMAP data

set.seed(1234)
n <- nrow(umap_data)
M <- umap_data[sample(1:n, 1), , drop = FALSE]

# Select remaining k-1 centroids using weighted probability
for (i in 2:k_types) {
  # Compute distance from each point to the nearest centroid
  D <- as.matrix(dist(rbind(M, umap_data)))[2:(n+1), 1]

  # Probability for each point to be chosen as the next centroid
  P <- D^2 / sum(D^2)

  # Select the next centroid based on P
  pick <- sample(1:n, 1, prob = P)
  M <- rbind(M, umap_data[pick, , drop = FALSE])
}

kmeans_umap_pp <- kmeans(umap_data, centers = M, algorithm = "Lloyd")
```

```
## Warning: did not converge in 10 iterations
```

```
comp_labels$cluster <- as.factor(kmeans_umap_pp$cluster)

# Map clusters to true labels
cluster_label_mapping <- comp_labels %>%
  mutate(cluster = factor(kmeans_umap_pp$cluster)) %>%
  group_by(cluster) %>%
  summarise(mode_type = Mode(type1), .groups = 'drop')

# Map cluster labels to the most common actual type
labelled_clusters <- comp_labels %>%
  left_join(cluster_label_mapping, by = "cluster") %>%
  select(type1, cluster, mode_type)

# Calculate accuracy
# mean(labelled_clusters$type1 == labelled_clusters$mode_type)
```

```
# Summary of stats for standard kmeans clusters
stats_with_cluster <- stats %>%
  mutate(km_cluster = kmeans_stats$cluster)

km_cluster_summary_stats <- stats %>%
```

```
  mutate(km_cluster = kmeans_stats$cluster) %>%
  group_by(km_cluster) %>%
  summarise(
    attack_mean = mean(attack),
    hp_mean = mean(hp),
    defense_mean = mean(defense),
    speed_mean = mean(speed),
    .groups = 'drop'
  )
```

```
# Summarize kmeans cluster assignments
comp_labels$cluster <- as.factor(kmeans_stats$cluster)
cluster_label_mapping <- comp_labels %>%
  mutate(cluster = factor(kmeans_stats$cluster)) %>%
  group_by(cluster) %>%
  summarise(mode_type = Mode(type1), .groups = 'drop')
labelled_clusters <- comp_labels %>%
  left_join(cluster_label_mapping, by = "cluster") %>%
  select(type1, cluster, mode_type)
km_cluster_summary_stats_comp <- km_cluster_summary_stats
km_cluster_summary_stats_comp$mode_type <- cluster_label_mapping$mode_type

kable(table(comp_labels), caption = "Summary of Cluster Assignments from Standard K-means (Stats Data)")
```

Table 1: Summary of Cluster Assignments from Standard K-means (Stats Data)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
bug	3	6	0	3	0	0	4	2	2	12	1	19	2	0	1	16	1	0
dark	1	0	0	10	0	0	2	1	0	0	0	4	0	5	0	3	1	2
dragon	0	0	2	0	9	0	0	0	0	0	0	0	0	11	0	0	0	5
electric	5	0	0	9	0	1	4	2	1	0	0	15	0	0	0	1	1	0
fairy	1	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0
fighting	0	0	0	0	0	0	1	0	0	0	0	1	0	1	25	0	0	0
fire	3	0	0	2	1	2	1	38	0	0	0	0	0	0	0	4	0	1
flying	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
ghost	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26	1
grass	2	50	1	0	0	1	2	0	5	14	0	0	0	0	2	0	1	0
ground	1	0	0	2	4	5	1	0	0	0	0	8	6	0	0	0	4	1
ice	2	0	0	7	0	0	0	0	0	0	1	6	3	0	0	1	3	0
normal	2	2	1	38	1	4	0	0	4	0	1	22	0	0	2	26	0	2
poison	0	0	0	2	1	2	0	3	0	17	3	1	0	0	1	2	0	0
psychic	4	0	11	0	0	0	0	0	5	0	0	0	0	0	1	0	30	2
rock	4	0	0	0	1	4	3	0	0	0	6	2	18	1	0	4	2	0
steel	3	0	1	0	0	0	17	0	0	0	0	0	0	0	0	0	1	2
water	4	2	0	4	1	0	1	0	3	2	76	7	0	0	1	7	3	3

Classification

```
# load packages
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:patchwork':
##
##     area

## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.4.3

## Loaded gbm 2.2.2

## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

```
#load data
load("./Data/pokemon.RData")
load("./Data/dr_pokemon2.RData")
stats2 = read.csv('pokedex.csv')
```

```
do_LDA = function(data, train, attributes, to_plot, title){
  x_train = as.matrix(data[train,attributes])
  y_train = data[train,'type1']
  x_test = as.matrix(data[-train,attributes])
  y_test = data[-train,'type1']

  LDA = lda(x_train, grouping = y_train)

  pred = predict(LDA)
  PX = pred$x
  df = data.frame(PX, types = pred$class)
  predm = predict(LDA, LDA$means)
  PM = predm$x
  dfm = data.frame(PM, types = predm$class)
  plt = ggplot(mapping = aes(LD1, LD2, color = types))+
    geom_point(data = df)+
    geom_point(data = dfm, color = "black", shape = 4, size = 3)

  df2 = data.frame(PX, types = c(pred$class, y_train), type = c(rep("predicted", length(train)), rep("t
  #print(df2)
  plt2 = ggplot(df2, aes(LD1, LD2, color = types))+
    geom_point()+
    facet_wrap(~type)+
    labs(title = title)

  pred_test = predict(LDA, x_test)
  PXtest = pred_test$x
  accstest = c()
  accs = c()
  for(L in 1:10){
    LDAreg_train = lda(PX[,1:L,drop = F], grouping = y_train)
    labelsreg_train = predict(LDAreg_train)$class
    #labelsreg = classify(PX[,1:L,drop = F], PM[,1:L, drop = F], pi)
```

```
    accs = c(accs, mean(y_train == labelsreg_train))

    LDAreg = lda(PX[,1:L,drop = F], grouping = y_train)
    labelsreg = predict(LDAreg, PXtest[,1:L,drop = F])$class
    #labelsreg = classify(PXtest[,1:L,drop = F], PM[,1:L, drop = F], pi)
    accstest = c(accstest, mean(y_test == labelsreg))
  }

  return(list(data.frame(L = 1:10, accuracy_train = accs, accuracy_test = accstest), plt, plt2))

  }
```

```
# set missing values from other Pokemon dataset and remove non-numerical attributes
names_comp = c(tolower(stats[,'name.simple']), tolower(stats[,'name']))
stats$weight_kg = stats2[stats2[,'name'] %in% names_comp,]$weight
stats$height_m = stats2[stats2[,'name'] %in% names_comp,]$height

stats_num = stats[,!(colnames(stats) %in% c('abilities', 'capture_rate', 'classfication', 'japanese_nam
```

```
pca_stats <- prcomp(stats_num, center = TRUE, scale. = TRUE)
pca_stats_df <- as.data.frame(pca_stats$x)
colnames(pca_stats_df) <- paste0("PC", 1:ncol(pca_stats_df))
loadings <- pca_stats$rotation
```

```
var_explained <- summary(pca_stats)$importance[2, ]  # Proportion of variance explained
cumulative_var <- cumsum(var_explained)
# Keep 20 PCs (90% VE)
pca_stats_df <- pca_stats_df[,1:20]
```

```
#combined dataframe for PDA, GB
stats_comb = cbind(pca_stats_df, stats[,c('name', 'type1', 'generation')])
```

```
#get train set and run LDA on full dataset
n = dim(stats_comb)[1]
stats_comb$ind = seq(1, n, by=1)
train = c()
#use stratified sampling
for(t in unique(stats_comb$type1)){
  stat_t = stats_comb[stats_comb$type1 == t,]
  train = c(train, sample(stat_t$ind, 0.8*dim(stat_t)[1]))
}

lda  = do_LDA(stats_comb, train, colnames(stats_comb)[1:20], T, 'LDA Classification All Pokemon (Stats)
```

```
#Save plot/table
lda_full = lda[[3]]
save(lda_full,  file="./Figures/Lda_full.RData")

lda_full_res = lda[[1]][1:8,]
save(lda_full_res, file="./Figures/Lda_full_res.RData")
```

```
#Run LDA on each generation of pokemon
test_error = data.frame(row.names = seq(1, 10, by=1))
train_error = data.frame(row.names = seq(1, 10, by=1))
for(i in unique(stats_comb$generation)){
```

```
  gen_i = stats_comb[stats_comb$generation == i,]
  n_i = dim(gen_i)[1]
  gen_i$ind = seq(1, n_i, by=1)
  train_i = c()
  for(t in unique(stats$type1)){
    gen_i_t = gen_i[gen_i$type1 == t,]
    train_i = c(train_i, sample(gen_i_t$ind, 0.8*dim(gen_i_t)[1]))
  }

    lda = do_LDA(gen_i, train_i, colnames(stats_comb)[1:20], F, 'LDA Classification Gen 1 Pokemon (Stats
  train_error[,i] =  lda[[1]][,'accuracy_train']
  test_error[,i] =  lda[[1]][,'accuracy_test']

  #only want to save plot for gen1
  if(i == 1){
    lda_gen1 = lda[[3]]
    save(lda_gen1,  file="./Figures/Lda_gen1.RData")
  }
}
#train_error
#test_error
```

```
#save table
comb_error = cbind(train_error[3:6,1:4], test_error[3:6,1:4])
colnames(comb_error) = c('gen1-train', 'gen2-train', 'gen3-train', 'gen4-train', 'gen1-test', 'gen2-tes
save(comb_error,  file="./Figures/Lda_gen_res.RData")
```

```
do_GB = function(data, train, attributes, to_plot, s){
  train_data = data[train,attributes]
  test_data = data[-train,attributes]
  rownames(test_data) = NULL

  gbm_model = gbm(
    formula = type1 ~ .,
    data = train_data,
    distribution = "multinomial",
    shrinkage = s,
    cv.folds = 1
  )
  pred_gbm_train = predict(gbm_model, newdata = train_data, n.trees = gbm_model$n.trees, type = 'respons
  pred_gbm = predict(gbm_model, newdata = test_data, n.trees = gbm_model$n.trees, type = 'response')
  #print(colnames(pred_gbm))

  train_test = ifelse(seq(1, dim(data)[1], by=1) %in% train, 'train', 'test - correct')
  df= data.frame(data[,c('PC1', 'PC2')], test = train_test)

  correct_train = 0
  for(i in 1:dim(pred_gbm_train)[1]){
    pred_val = as.numeric(which.max(pred_gbm_train[i,,1]))
    if(colnames(pred_gbm_train)[pred_val] == paste(train_data[i,'type1'])){
      correct_train = correct_train + 1
    }
  }
  train_accuracy = correct_train/dim(pred_gbm_train)[1]
```

```
  correct_test = 0
  for(i in 1:dim(pred_gbm)[1]){
    pred_val = as.numeric(which.max(pred_gbm[i,,1]))
    if(colnames(pred_gbm)[pred_val] == paste(test_data[i,'type1'])){
      correct_test = correct_test + 1
    }
    else{
      df[!(df$test == 'test - correct'),][i,'test'] = 'test - incorrect'
    }
  }
  test_accuracy = correct_test/dim(pred_gbm)[1]

  plt = ggplot(data = df, aes(PC1, PC2, col = factor(test))) + geom_point() + labs(title = 'Plot of Grad
  return(list(train_accuracy, test_accuracy, plt))
}
```

```
# run GB on with learn rate = 0.1 and print plot
gb = do_GB(stats_comb, train, c(colnames(stats_comb)[1:20], 'type1'), T, 0.1)
gbm_plot = gb[[3]]
save(gbm_plot,  file="./Figures/GBM_plot.RData")
```

```
#Show GB accuracy for different learn rates
accuracy_gb = data.frame(shrinkage = c(0.01, 0.1, 0.25), train_accuracy = rep(0, 3), test_accuracy = rep
shrinkage = c(0.01, 0.1, 0.25)
#train_gb = sample(1:n, 0.8*n)
for(s in 1:3){
  gb = do_GB(stats_comb, train, c(colnames(stats_comb)[1:20], 'type1'), T, shrinkage[s])
  accuracy_gb[s,'train_accuracy'] = gb[[1]]
  accuracy_gb[s,'test_accuracy'] = gb[[2]]
}

save(accuracy_gb,  file="./Figures/GBM_res.RData")
```

```
accuracy_df = data.frame(row.names = c('train', 'test'))
for(i in unique(stats_comb$generation)){
  gen_i = stats_comb[stats_comb$generation == i,]
  n_i = dim(gen_i)[1]
  gen_i$ind = seq(1, n_i, by=1)
  train_i = c()
  for(t in unique(stats_comb$type1)){
    gen_i_t = gen_i[gen_i$type1 == t,]
    train_i = c(train_i, sample(gen_i_t$ind, 0.8*dim(gen_i_t)[1]))
  }
  #train_i = sample(1:n_i, 0.8*n_i)
  gb = do_GB(gen_i, train_i, c(colnames(stats_comb)[1:20], 'type1'), F, 0.1)
  accuracy_df[,i] =  c(gb[[1]], gb[[2]])
}
```