Clustering and PCA

Yifei Sun

Contents

```
library(ISLR)
library(factoextra)  #visualizing cluster and PCA
library(gridExtra)   #arrange multiple plots
library(corrplot)
library(RColorBrewer) #to generate colors
library(gplots)
library(jpeg)
library(tidyverse)
```

The dataset we use contains data on 166 first generation Pokemon, including their names and basic stats: HP, Attack, Defense, Special Attack, Special Defense, and Speed. The data is from Kaggle (https://www.kaggle.com/abcsds/pokemon). We will apply unsupervised learning methods on this data. The list of Pokemon can be found at (https://pokemondb.net/pokedex/national).

```
dat <- read.csv("data/Pokemon.csv")
head(dat)
```

```
##                    Name HitPoints Attack Defense SpecialAttack SpecialDefense
## 1             Bulbasaur        45     49      49            65             65
## 2               Ivysaur        60     62      63            80             80
## 3              Venusaur        80     82      83           100            100
## 4 VenusaurMega Venusaur        80    100     123           122            120
## 5            Charmander        39     52      43            60             50
## 6            Charmeleon        58     64      58            80             65
##   Speed Legendary
## 1    45     FALSE
## 2    60     FALSE
## 3    80     FALSE
## 4    80     FALSE
## 5    65     FALSE
## 6    80     FALSE
```

```
dim(dat)
```

```
## [1] 166   8
```

```
dat1 <- dat[,2:7]
head(dat1)
```

```
##   HitPoints Attack Defense SpecialAttack SpecialDefense Speed
## 1        45     49      49            65             65    45
## 2        60     62      63            80             80    60
## 3        80     82      83           100            100    80
## 4        80    100     123           122            120    80
## 5        39     52      43            60             50    65
## 6        58     64      58            80             65    80
```

```
dat1 <- scale(dat1)
head(dat1)
```

```
##       HitPoints     Attack    Defense SpecialAttack SpecialDefense      Speed
## [1,] -0.7394793 -0.8989689 -0.7632830    -0.1980097     -0.1603732 -0.9295212
## [2,] -0.2066947 -0.4761322 -0.2744790     0.2375417      0.4277405 -0.4240598
## [3,]  0.5036847  0.1743859  0.4238124     0.8182769      1.2118920  0.2498887
## [4,]  0.5036847  0.7598521  1.8203953     1.4570855      1.9960435  0.2498887
## [5,] -0.9525931 -0.8013912 -0.9727705    -0.3431935     -0.7484868 -0.2555726
## [6,] -0.2777327 -0.4110804 -0.4490519     0.2375417     -0.1603732  0.2498887
```

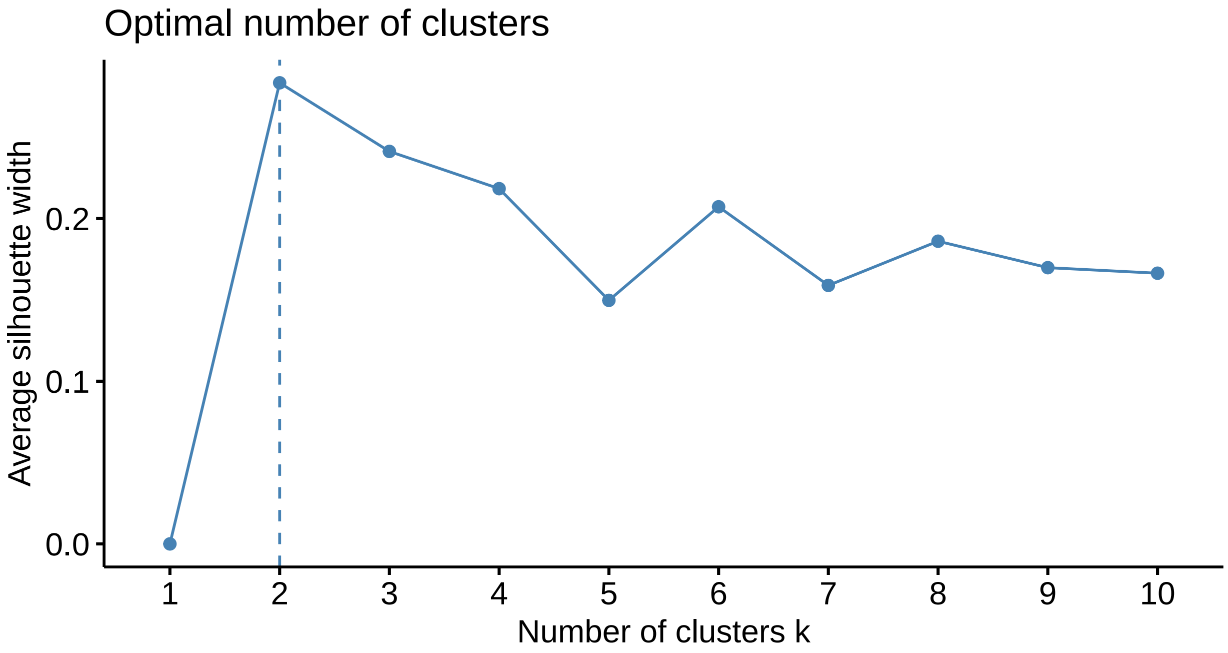
```
rownames(dat1) <- dat[,1]
head(dat1)
```

```
##                          HitPoints     Attack    Defense SpecialAttack
## Bulbasaur               -0.7394793 -0.8989689 -0.7632830    -0.1980097
## Ivysaur                 -0.2066947 -0.4761322 -0.2744790     0.2375417
## Venusaur                 0.5036847  0.1743859  0.4238124     0.8182769
## VenusaurMega Venusaur    0.5036847  0.7598521  1.8203953     1.4570855
## Charmander              -0.9525931 -0.8013912 -0.9727705    -0.3431935
## Charmeleon              -0.2777327 -0.4110804 -0.4490519     0.2375417
##                         SpecialDefense      Speed
## Bulbasaur                   -0.1603732 -0.9295212
## Ivysaur                      0.4277405 -0.4240598
## Venusaur                     1.2118920  0.2498887
## VenusaurMega Venusaur        1.9960435  0.2498887
## Charmander                  -0.7484868 -0.2555726
## Charmeleon                  -0.1603732  0.2498887
```

K means clustering

Partitioning methods such as k-means clustering require the users to specify the number of clusters to be generated. The function `fviz_nbclust()` determines and visualizes the optimal number of clusters using different methods: within cluster sums of squares, average silhouette and gap statistics. We use average silhouette, and the greater the silhouette value the better.

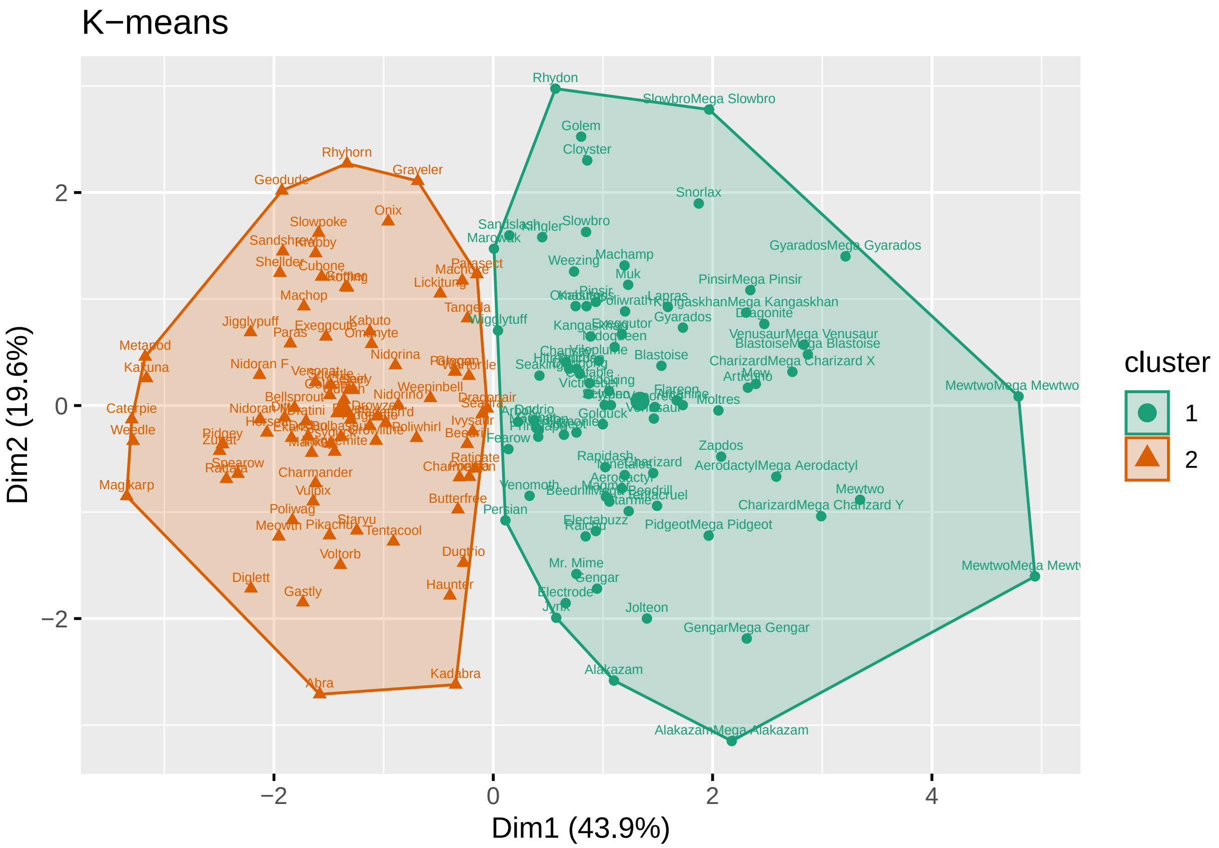
```
fviz_nbclust(dat1,
             FUNcluster = kmeans,
             method = "silhouette") #optimal number = 2
```



```
set.seed(1)
km <- kmeans(dat1, centers = 2, nstart = 20)  #baseR function
```

The function `fviz_cluster()` provides ggplot2-based visualization of partitioning methods including K means. Observations are represented by points in the plot, using principal components if $p > 2$. An ellipse is drawn around each cluster.

```
km_vis <- fviz_cluster(list(data = dat1, cluster = km$cluster),
                       ellipse.type = "convex",
                       geom = c("point","text"),
                       labelsize = 5,
                       palette = "Dark2") + labs(title = "K-means")

km_vis
```



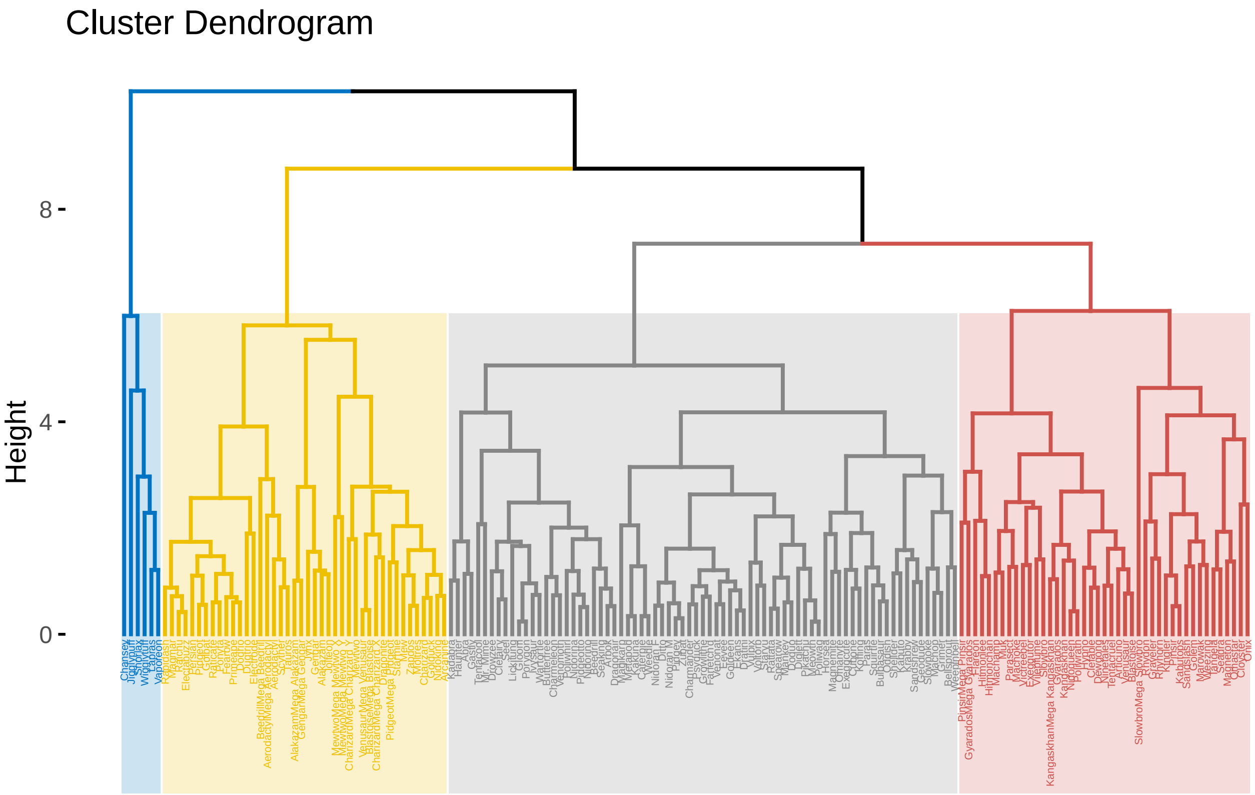
Hierarchical clustering

We can also apply hierarchical clustering on this data. Here we use the Euclidean distance and different types of linkage.

```
hc.complete <- hclust(dist(dat1), method = "complete") #default setting
hc.average <- hclust(dist(dat1), method = "average")
hc.single <- hclust(dist(dat1), method = "single")
hc.centroid <- hclust(dist(dat1), method = "centroid")

# average and centroid are usually between single and complete linkage
```

The function `fviz_dend()` can be applied to visualize the dendrogram.

```
fviz_dend(hc.complete, k = 4,
          cex = 0.3,
          palette = "jco",
          color_labels_by_k = TRUE,
          rect = TRUE, rect_fill = TRUE, rect_border = "jco",
          labels_track_height = 2.5)
```

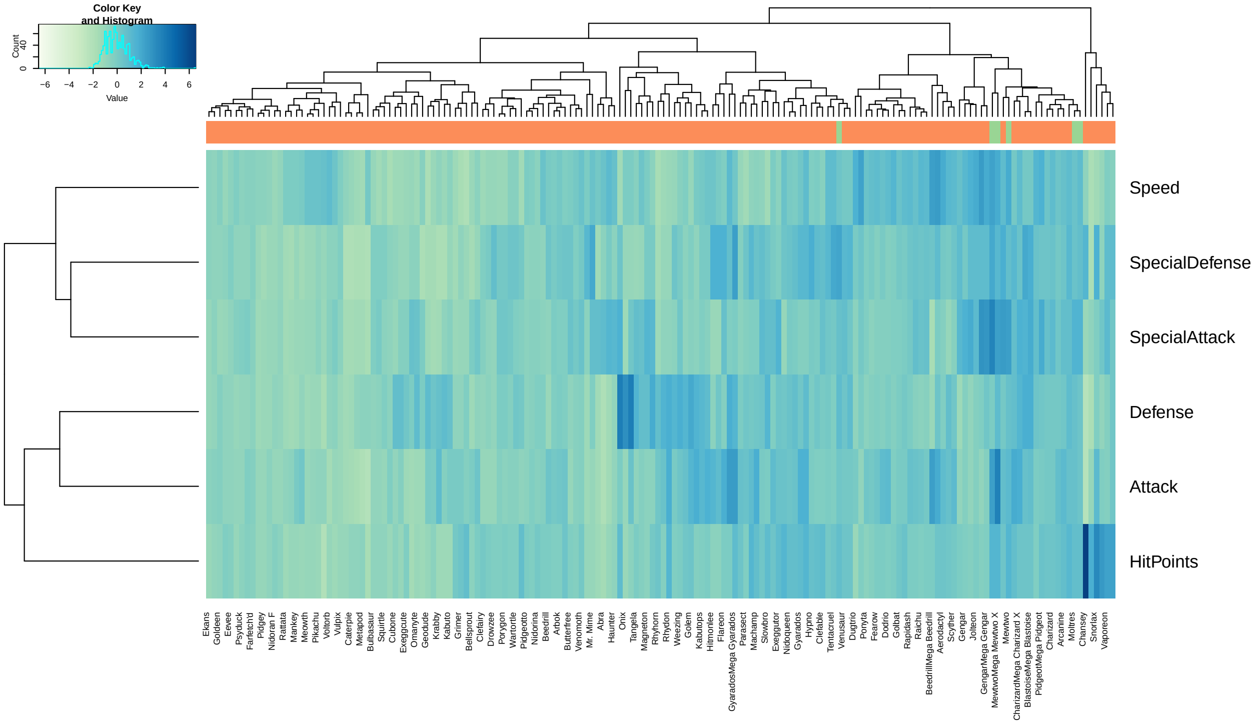


```
ind4.complete <- cutree(hc.complete, 4)

# Who are in the fourth cluster?
dim(dat[ind4.complete == 4,])
```

```
## [1] 6 8
```

To display more details, we show the heatmap of the data.

```
#display.brewer.all(n=NULL, type="all", select=NULL, exact.n=TRUE)
col1 <- colorRampPalette(brewer.pal(9, "GnBu"))(100)
col2 <- colorRampPalette(brewer.pal(3, "Spectral"))(2)

heatmap.2(t(dat1),
          col = col1, keysize=.8, key.par = list(cex=.5),
          trace = "none", key = TRUE, cexCol = 0.75,
          labCol = as.character(dat[,1]),
          ColSideColors = col2[as.numeric(dat[,"Legendary"])+1],
          margins = c(10, 10))
```



PCA

The function `prcomp()` can be used to perform PCA. PCA finds low-dimensional features to represent data (p < n)

```
pca <- prcomp(dat1)
pca$rotation
```

```
##                     PC1        PC2         PC3          PC4         PC5
## HitPoints      0.3638022  0.2862972 -0.72425610  0.078135517 -0.42663445
## Attack         0.4363956  0.3149994  0.27779372  0.573757403 -0.12333177
## Defense        0.3031184  0.5812622  0.48929801 -0.361032760  0.03453292
## SpecialAttack  0.4378985 -0.3119077  0.03743076 -0.654754892 -0.29724064
## SpecialDefense 0.5204254 -0.1331800 -0.25037584  0.006718302  0.80453025
## Speed          0.3503261 -0.6049135  0.30786700  0.324966961 -0.25682461
##                        PC6
## HitPoints      -0.27020899
## Attack          0.53736079
## Defense        -0.44643458
## SpecialAttack   0.43874986
## SpecialDefense -0.03766086
## Speed          -0.49498168
```

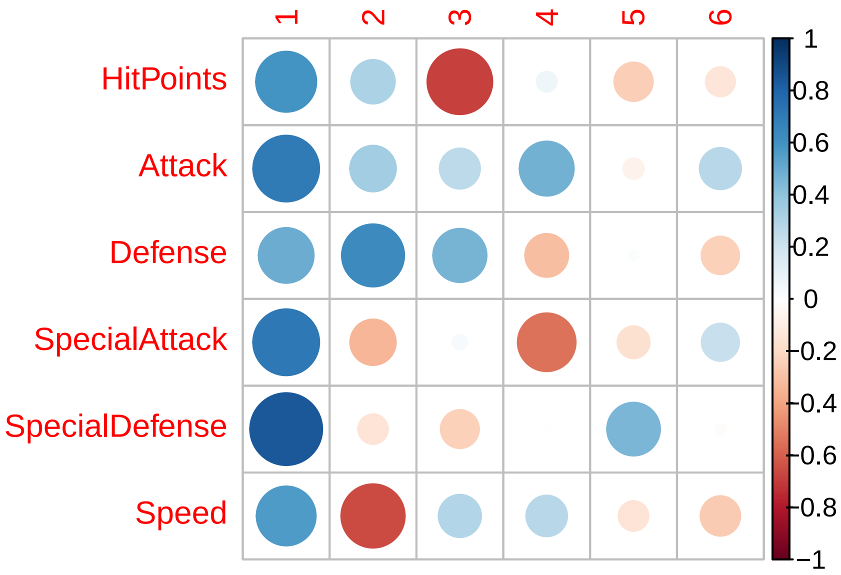
```
# pca$x returns z scores

pca$sdev #sd(Zk) = d_k/sqrt(n-1), in decreasing order
```

```
## [1] 1.6238460 1.0848056 0.9487926 0.8345883 0.5670204 0.5177487
```

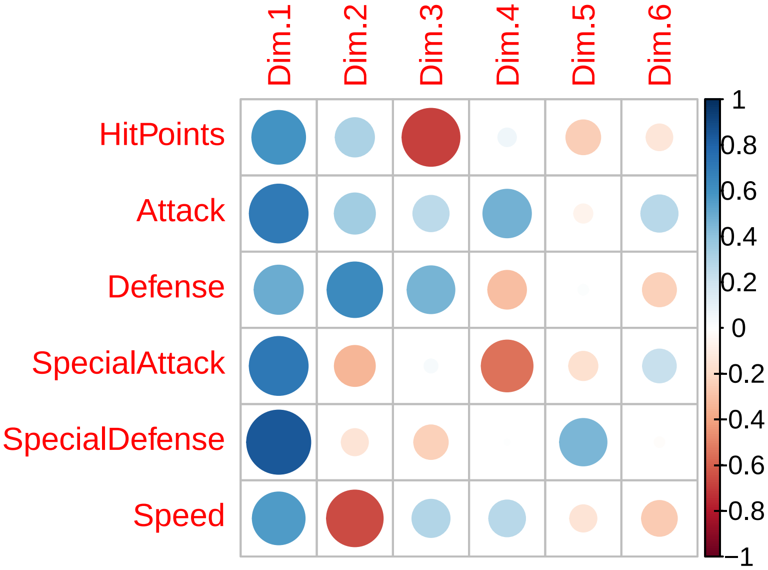
```
pca$rotation %*% diag(pca$sdev)  #correlation loading: between X (original p features) and Z (transform
```

```
##                      [,1]       [,2]        [,3]         [,4]        [,5]
## HitPoints      0.5907587  0.3105768 -0.68716880  0.065210990 -0.24191044
## Attack         0.7086393  0.3417131  0.26356862  0.478851234 -0.06993163
## Defense        0.4922176  0.6305564  0.46424231 -0.301313729  0.01958087
## SpecialAttack  0.7110798 -0.3383592  0.03551403 -0.546450792 -0.16854151
## SpecialDefense 0.8450908 -0.1444744 -0.23755473  0.005607016  0.45618508
## Speed          0.5688756 -0.6562135  0.29210193  0.271213634 -0.14562479
##                       [,6]
## HitPoints      -0.13990035
## Attack          0.27821784
## Defense        -0.23114091
## SpecialAttack   0.22716216
## SpecialDefense -0.01949886
## Speed          -0.25627611
```

```
corrplot(pca$rotation %*% diag(pca$sdev))
```

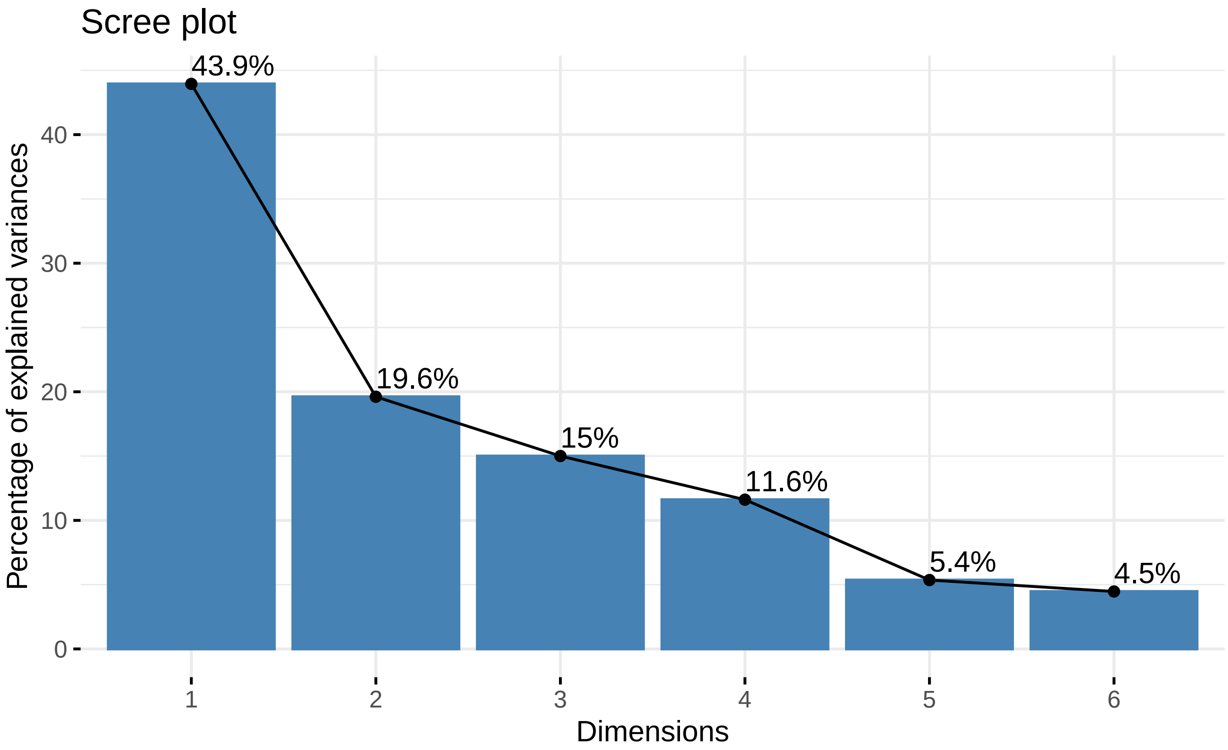


```
var <- get_pca_var(pca)

corrplot(var$cor)
```



The function `fviz_eig()` plots the eigenvalues/variances against the number of dimensions.

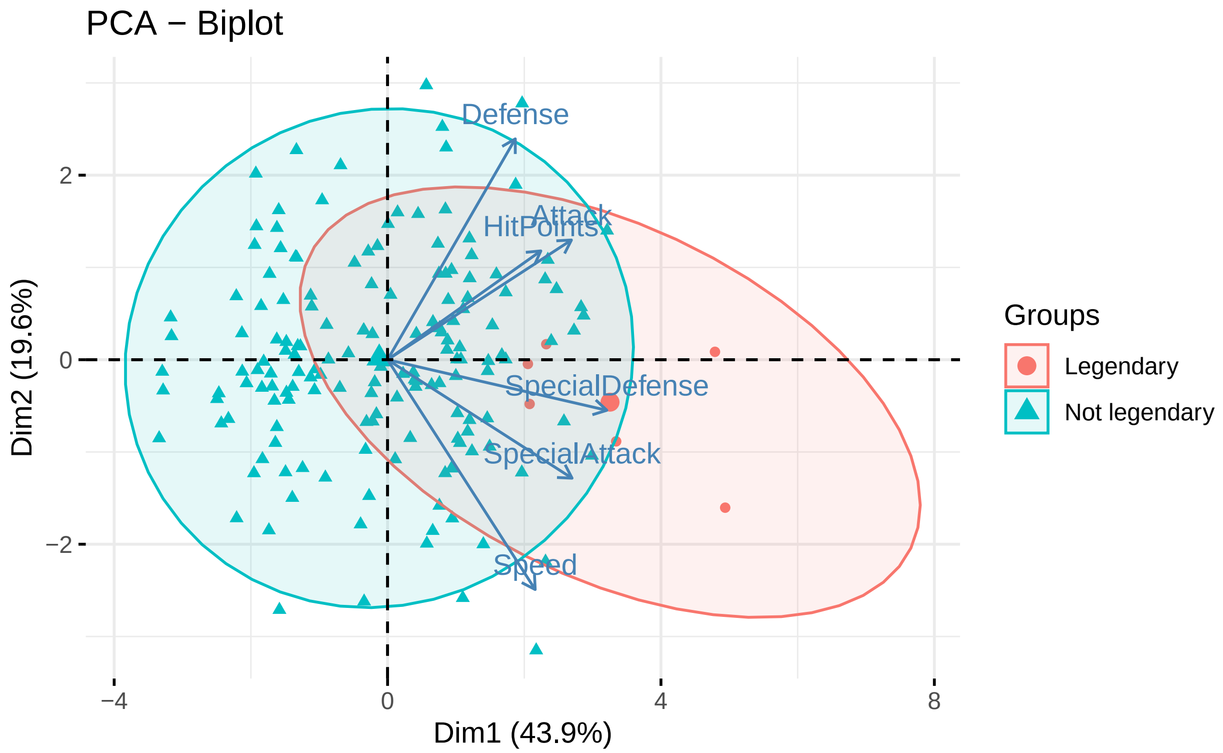
```
# Scree plot: p = 6. Want to find "elbow": where after this pc the line flatten out
fviz_eig(pca, addlabels = TRUE)
```



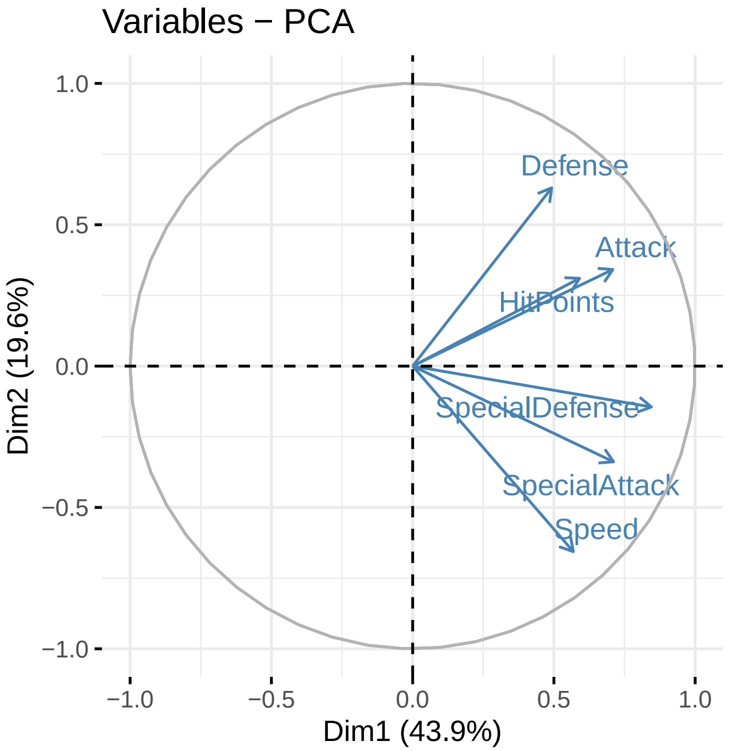
PCA - Biplot

The function `fviz_pca_biplot()` can be used to obtain the biplot of individuals and variables.

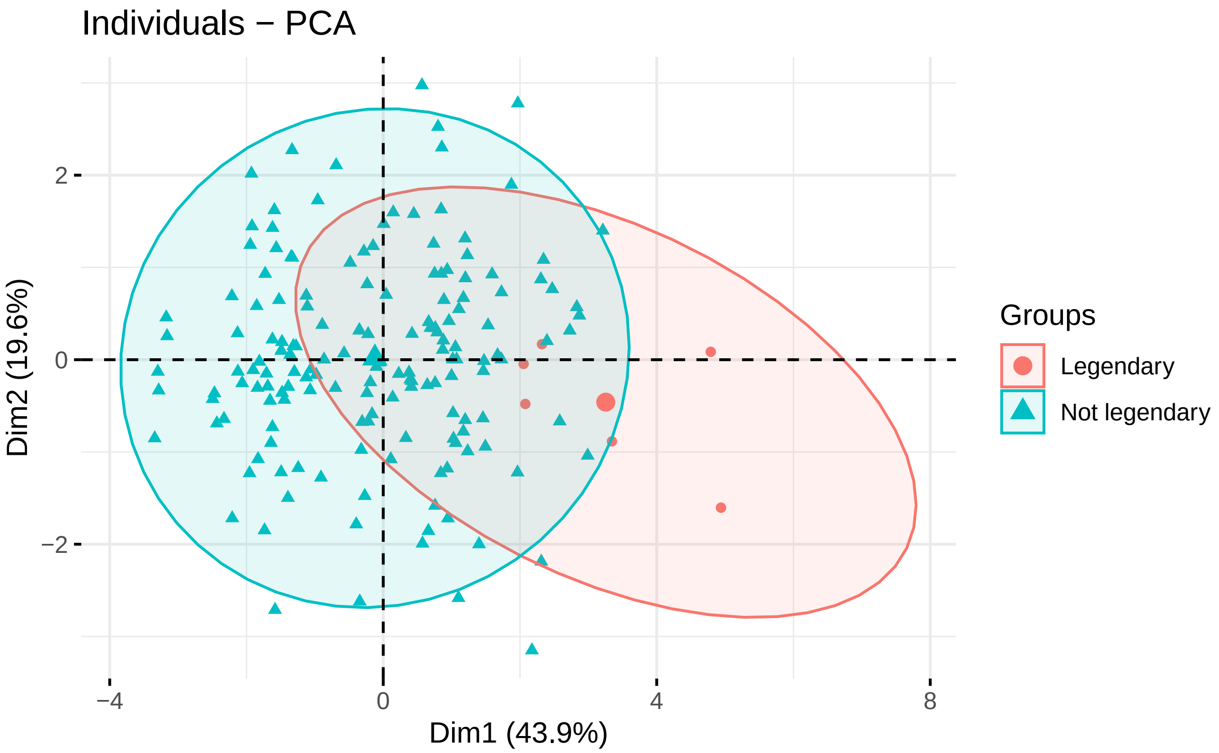
```
fviz_pca_biplot(pca, axes = c(1,2),
                habillage = ifelse(dat$Legendary==TRUE, "Legendary","Not legendary"),
                label = c("var"),
                addEllipses = TRUE)
```



```r
fviz_pca_var(pca, col.var = "steelblue", repel = TRUE)
```



```r
fviz_pca_ind(pca,
             habillage = ifelse(dat$Legendary==TRUE,"Legendary","Not legendary"),
             label = "none",
             addEllipses = TRUE)
```

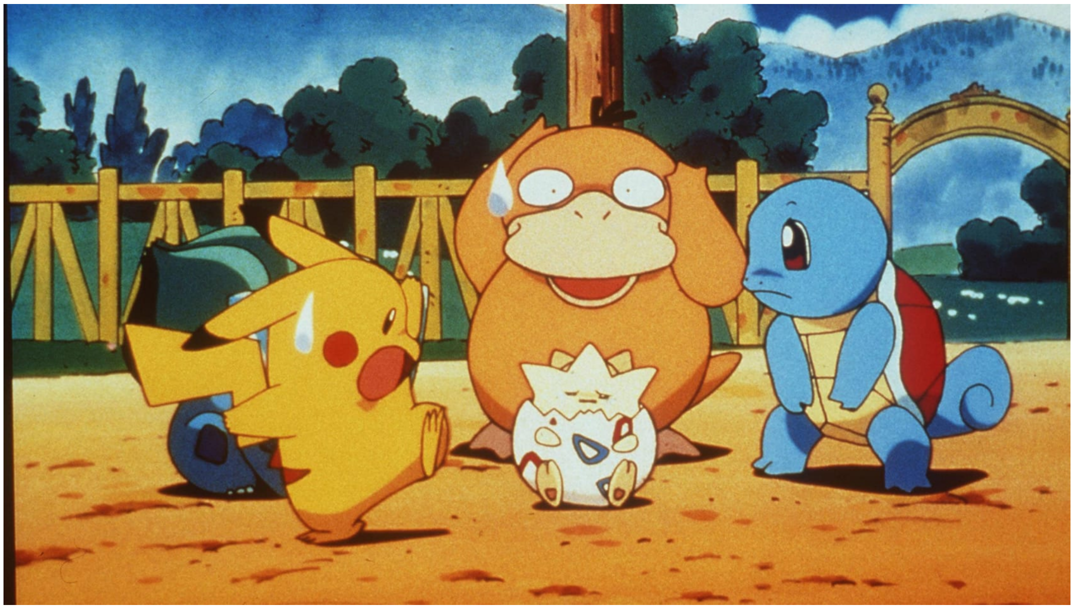


PCA for image compression

```
img <- readJPEG("image.jpeg")
dim(img)
```

```
## [1]  948 1685    3
```

```
knitr::include_graphics("image.jpeg")
```



```
r <- img[,,1]
g <- img[,,2]
b <- img[,,3]

img.r.pca <- prcomp(r, center = FALSE)
img.g.pca <- prcomp(g, center = FALSE)
img.b.pca <- prcomp(b, center = FALSE)

rgb.pca <- list(img.r.pca, img.g.pca, img.b.pca)

# Approximate X with XV_kV_k^T
compress <- function(pr, k)
{
  compressed.img <- pr$x[,1:k] %*% t(pr$rotation[,1:k])
  compressed.img
}

# Using first 20 PCs
pca20 <- sapply(rgb.pca,
                compress,
                k = 20,
                simplify = "array") #can use map function

writeJPEG(pca20, "pca20.jpeg")
knitr::include_graphics("pca20.jpeg")
```