

Clustering and PCA

```
library(factoextra) # provides viz tools for clustering and pca. The fuctions are in base R.
library(gridExtra)
library(corrplot)
library(RColorBrewer)
library(gplots) # For heatmaps
```

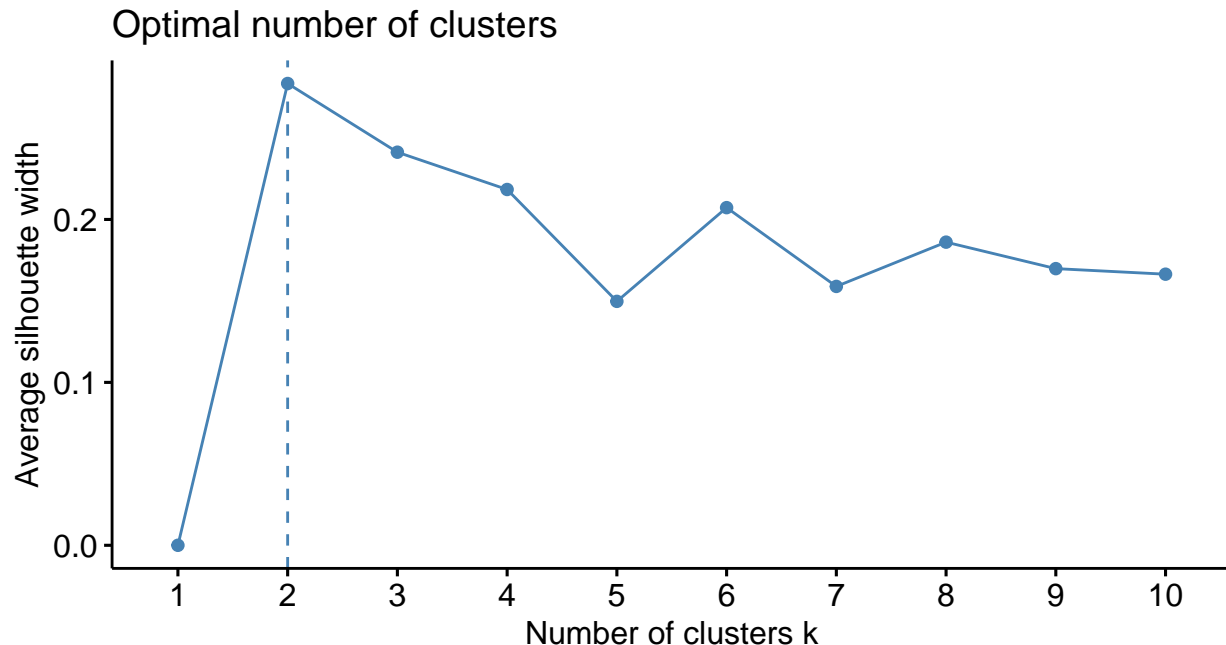
The dataset we use contains data on 166 first generation Pokemon, including their names and basic stats: HP, Attack, Defense, Special Attack, Special Defense, and Speed. The data is from Kaggle (<https://www.kaggle.com/abcsds/pokemon>). We will apply unsupervised learning methods on this data. The list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

```
dat <- read.csv("Pokemon.csv")
dat1 <- dat[,2:7]
dat1 <- scale(dat1)
rownames(dat1) <- dat[,1]
```

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")
```

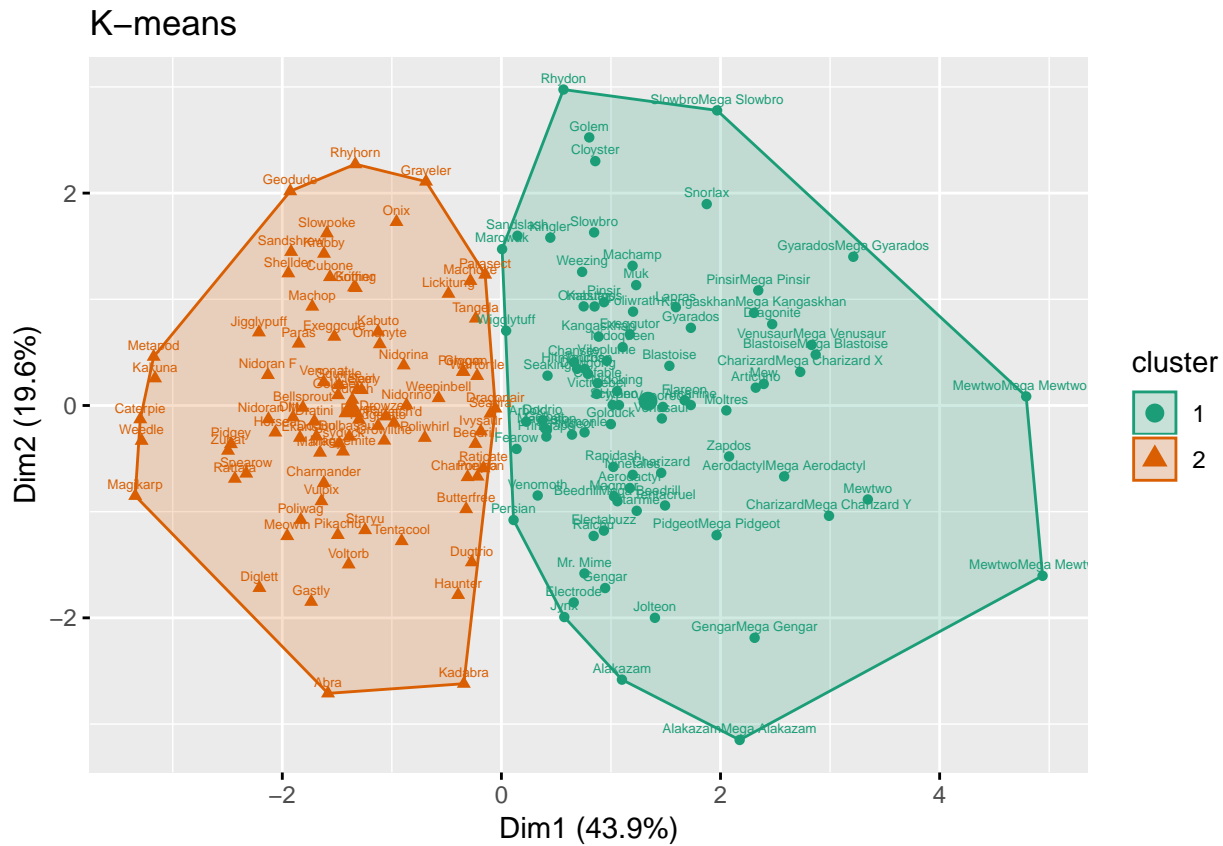


```
set.seed(1)
km <- kmeans(dat1, centers = 2, nstart = 20) # centers = 2 means k = 2. we are starting from 20.
```

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"), labels = 5,
                        palette = "Dark2") + labs(title = "K-means")
```

km_vis



The method above plots just the 2 principal components. We calculate the 1st and 2nd PC of each variable.

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")
# dist is the distance between the observations. Euclidean distance is the default. We can also use manhattan distance.
hc.average <- hclust(dist(dat1), method = "average")
hc.single <- hclust(dist(dat1), method = "single")
hc.centroid <- hclust(dist(dat1), method = "centroid")
```

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

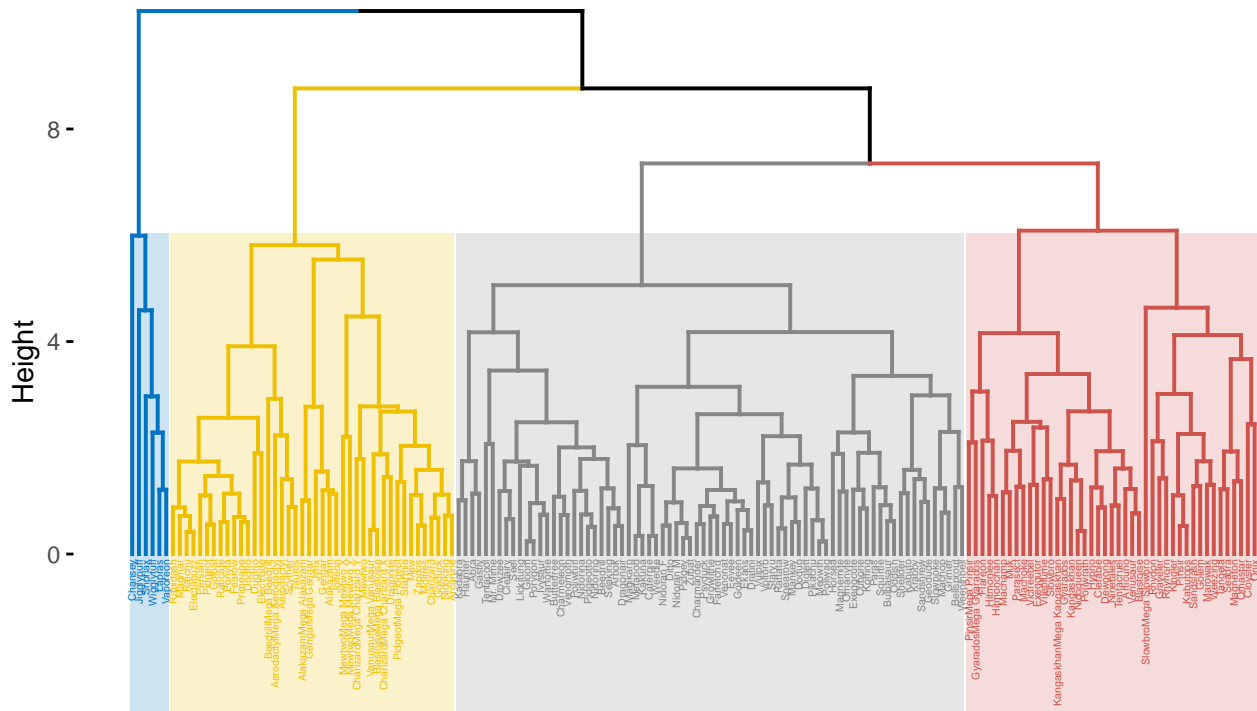
```
fviz_dend(hc.complete, k = 4, # you will get 4 clusters or subgroups
          cex = 0.3, # This is for the font size of the labels.)
```

```

palette = "jco", # used to specify the theme or color
color_labels_by_k = TRUE, # specify whether to color labels or not
rect = TRUE, # draws the rectangles
rect_fill = TRUE, # fills rectangle by its color
rect_border = "jco",
labels_track_height = 2.5 # change height so we can fully display names of pokemon
)

```

Cluster Dendrogram



```

ind4.complete <- cutree(hc.complete, 4) # return the index of the cluster

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

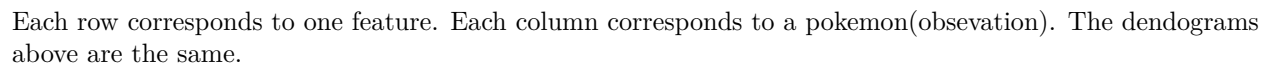
```

##	Name	HitPoints	Attack	Defense	SpecialAttack	SpecialDefense	Speed
## 45	Jigglypuff	115	45	20	45	25	20
## 46	Wigglytuff	140	70	45	85	50	45
## 122	Chansey	250	5	5	35	105	50
## 143	Lapras	130	85	80	85	95	60
## 146	Vaporeon	130	65	60	110	95	65
## 156	Snorlax	160	110	65	65	110	30
##	Legendary						
## 45	FALSE						
## 46	FALSE						
## 122	FALSE						
## 143	FALSE						
## 146	FALSE						
## 156	FALSE						

You can inspect the clusters to understand why the pokemon fall in their respective clusters.

```
#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), # We need to use the transpose of the data, so we have a diagram simila
  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))
```



The function `prcomp()` can be used to perform PCA.

##	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
```

gives the matrix $V = (\phi_1, \phi_2 \dots \phi_p)$. In our case, $p = 6$. You call the eigenvalues or factor loadings.

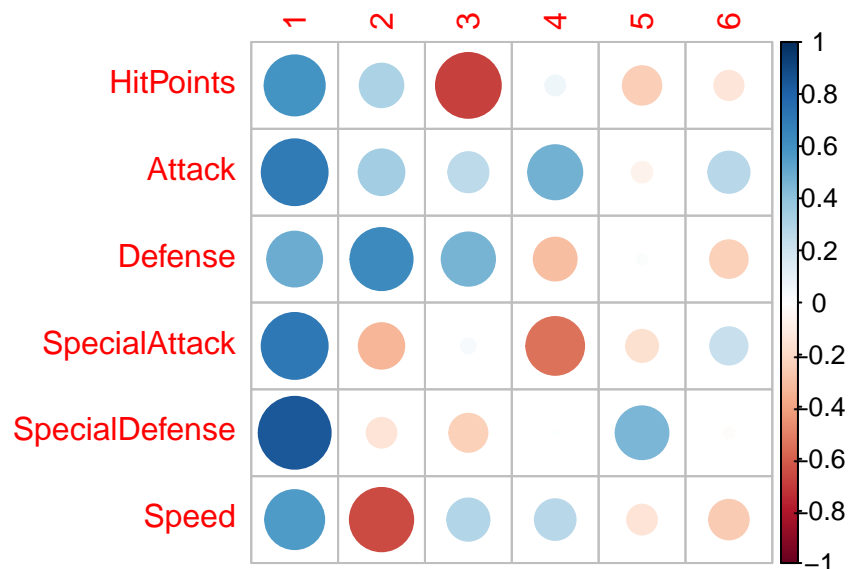
pca\$sdev # stand dev of each principal component.

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

```
pca$rotation %>% diag(pca$sdev)
```

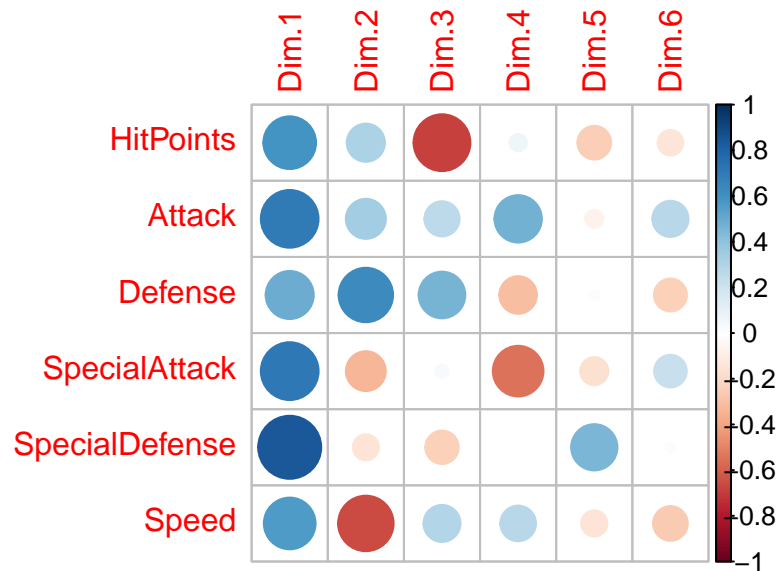
```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## HitPoints  0.5907587  0.3105768 -0.68716880  0.065210990 -0.24191044
## Attack     0.7086393  0.3417131  0.26356862  0.478851234 -0.06993163
## Defense    0.4922176  0.6305564  0.46424231 -0.301313729  0.01958087
## SpecialAttack 0.7110798 -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)) # each element is the correlation between the variable and the principal component.



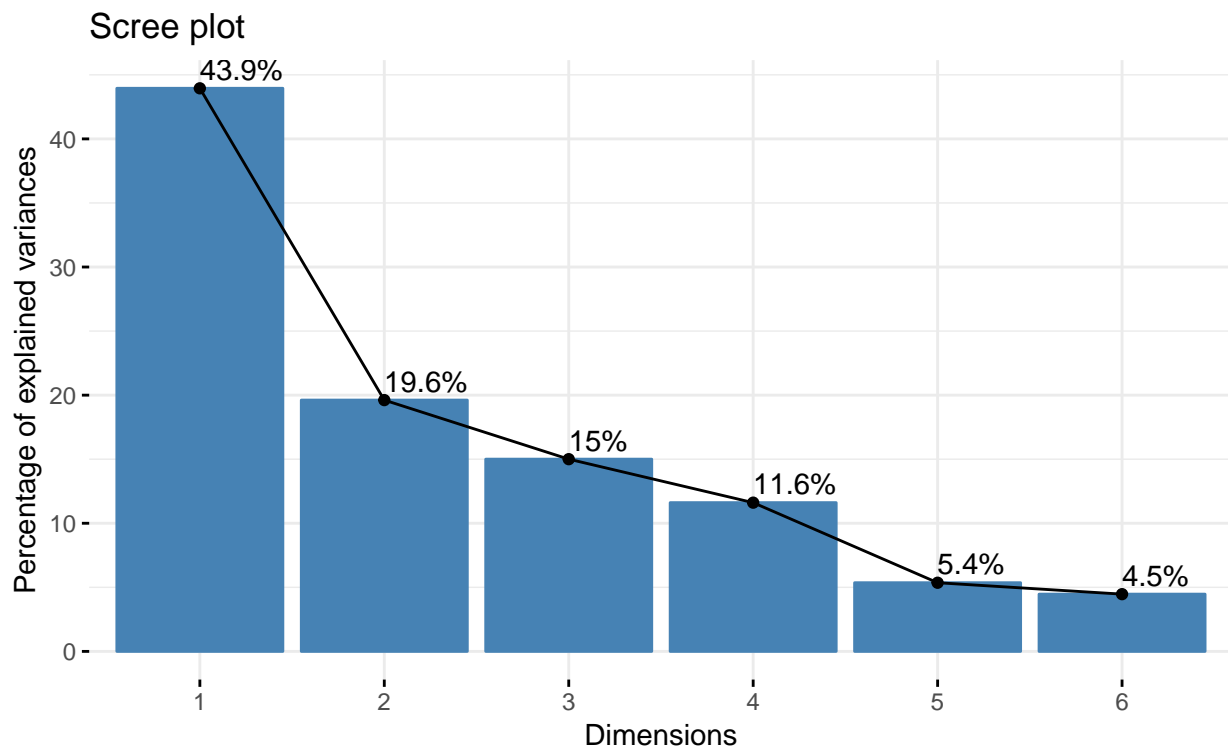
```
var <- get_pca_var(pca)
```

corrplot(var\$cor) # This is exactly the same as the previous plot. You can use any of them.



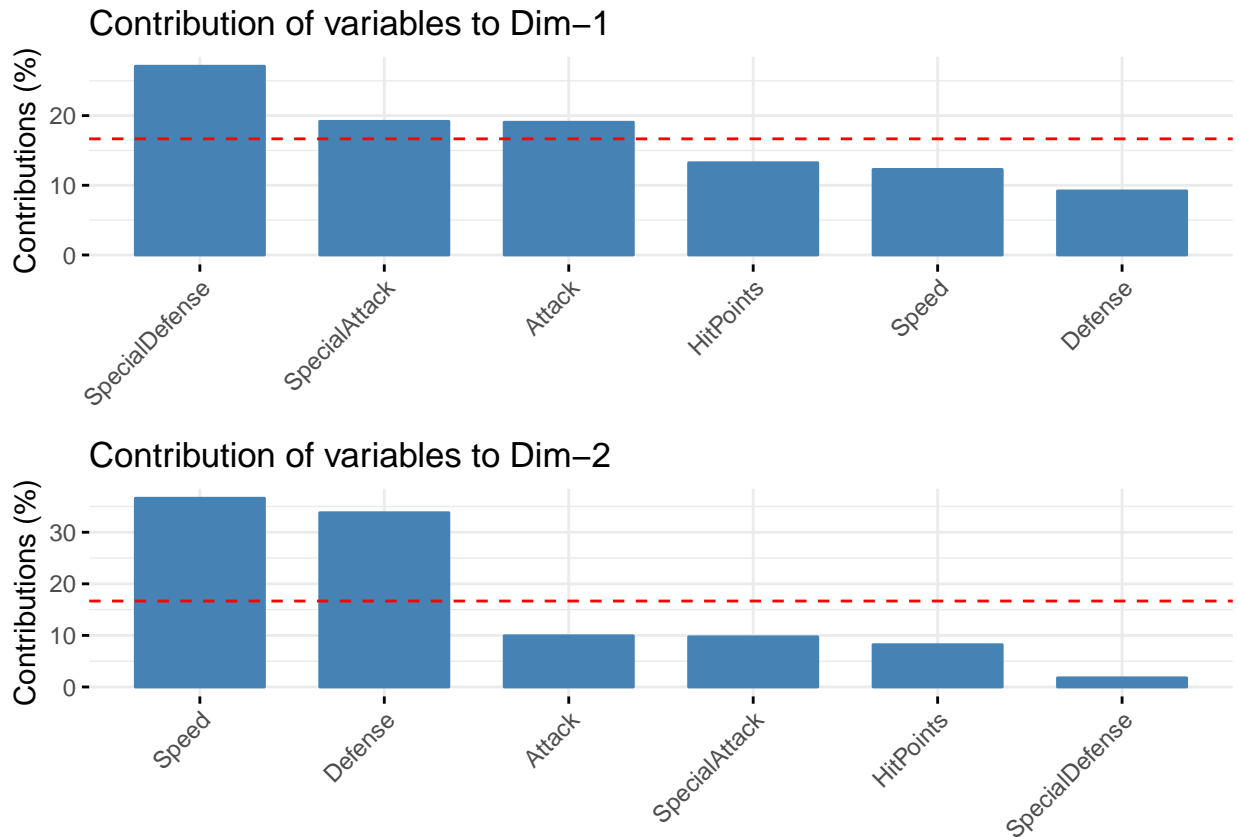
The function `fviz_eig()` plots the eigenvalues/variances against the number of dimensions.

```
fviz_eig(pca, addlabels = TRUE)
```



The function `fviz_contrib()` can be used to visualize the contribution of variables from the results of PCA.

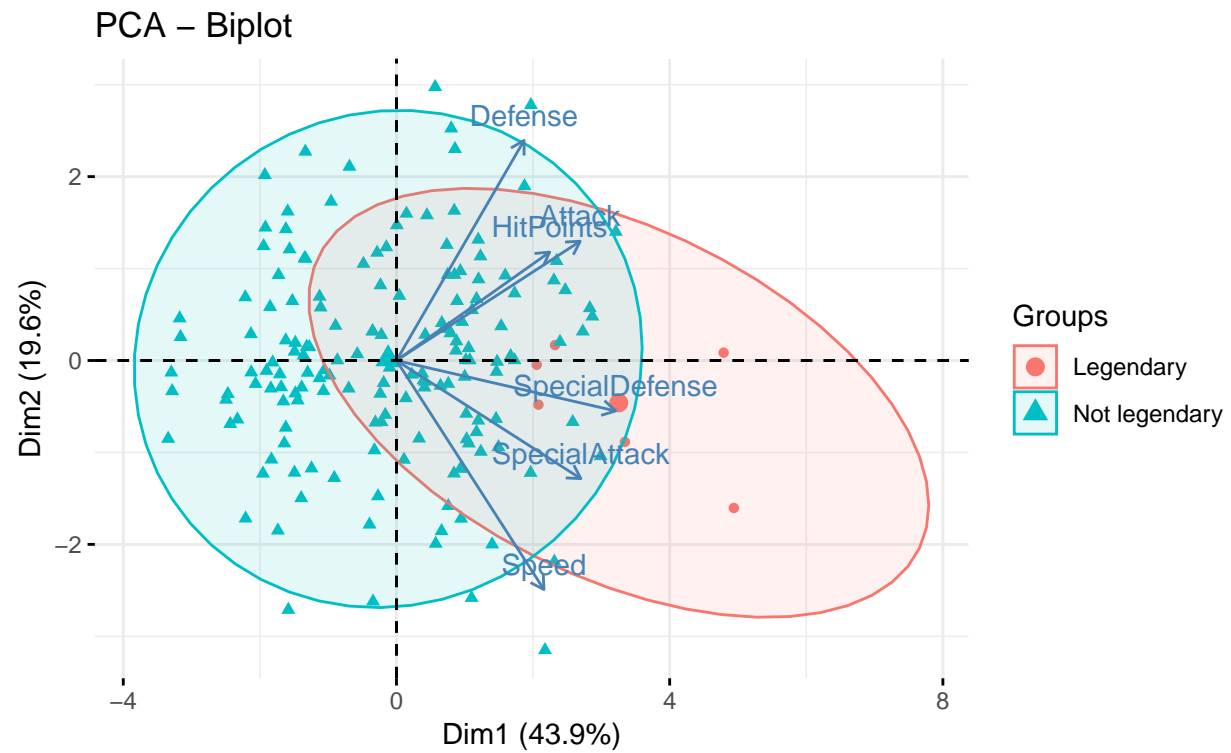
```
a <- fviz_contrib(pca, choice = "var", axes = 1)
b <- fviz_contrib(pca, choice = "var", axes = 2)
grid.arrange(a, b, nrow = 2)
```



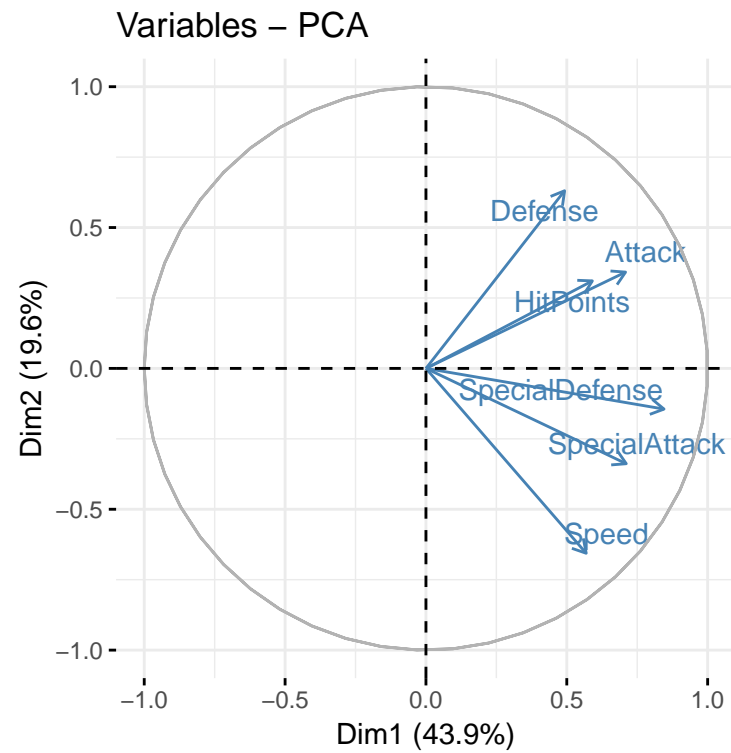
Here we are visualizing the contribution of each variable. We are plotting the rotation matrix which is just the square of the rotation matrix and then plot. The red line shows the average. In our case, it's 100/6 (we have 6 variables). The dim are the principal components.

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

```
fviz_pca_biplot(pca, axes = c(1,2), # specifies which PCs plot
  habillage = ifelse(dat$Legendary == TRUE, "Legendary", "Not legendary"), # this is for g
  label = c("var"),
  # We only variable the variable names without the observation names. Use individual for o
  addEllipses = TRUE)
```



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



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