PCA I: Up and running

2024-09-02

```
stopifnot(
  require(broom),
  require(FactoInvestigate),
  require(FactoMineR),
  require(ggfortify),
  require(ggrepel),
  require(glue),
  require(httr),
  require(patchwork),
  require(tidyverse)
)

old_theme <- theme_set(theme_minimal())
```

- **M1 MIDS/MFA**
- **Université Paris Cité**
- Année 2024-2025
- Course Homepage

- Moodle

Objectives

```
opts <- options()  # save old options

options(ggplot2.discrete.colour="viridis")
options(ggplot2.discrete.fill="viridis")
options(ggplot2.continuous.fill="viridis")
options(ggplot2.continuous.colour="viridis")
```

Computing SVD

PCA and SVD

Visualizing PCA

Definition : Thin SVD

Let X be a $n \times p$ real matrix with rank r.
Let U, D, V be a singular value decomposition of X such that the diagonal entries of D are non-increasing.
Let U_r (resp. V_r) be the $n \times r$ (resp. $p \times r$) matrix formed by the first r columns of U (resp. V)
Then

$$X = U_r \times D_r \times V_r^T$$

is a *thin Singular Value Decomposition* of X.

```r
A <- matrix(rnorm(12, 1), nrow = 4)

foo <- svd(A)

U <- foo$u

D <- foo$d

V <- foo$v

# xtable::xtableMatharray(A)
# xtable::xtableMatharray(U)
# xtable::xtableMatharray(diag(D))
# xtable::xtableMatharray(t(V))
A
```

```
           [,1]          [,2]       [,3]
[1,] -0.1354518  2.2926756615  1.3412783
[2,]  0.5964804 -0.6137002181  1.2093249
[3,] -0.3736019  0.0002272261  0.7723767
[4,]  0.2312187  1.8705878898 -0.7018153
```

```r
U
```

```
            [,1]       [,2]       [,3]
[1,] -0.82329213 -0.4189300 -0.1398984
[2,]  0.12259593 -0.6525113  0.7202804
[3,] -0.05358695 -0.3644787 -0.5335104
[4,] -0.55162374  0.5156374  0.4207032
```

```r
diag(D)
```

```
         [,1]    [,2]      [,3]
[1,] 3.057172 0.00000 0.0000000
[2,] 0.000000 2.03645 0.0000000
[3,] 0.000000 0.00000 0.7459495
```

```r
t(V)
```

```
            [,1]        [,2]        [,3]
[1,]  0.02522492 -0.97954970 -0.19961484
[2,] -0.03784548  0.19859952 -0.97934976
[3,]  0.99896518  0.03225854 -0.03206187
```

Example

In R, the SVD of matrix A is obtained by `svd_ <- svd(A)` with `U <- svd_$u`, `V <- svd_$V` and `D <- diag(svd_$d)`

Two perspectives on PCA (and SVD)

Faithful projection

Let X be a $n \times p$ matrix, a reasonable task consists in finding a $k \lll p$-dimensional subspace E of $\mathbb{R}^p$ such that

$$X \times \Pi_E$$

is as close as possible from X. By as close as possible, we mean that the sum of squared Eudlidean distances between the rows of X and the rows of $X \times \Pi_E$ is as small as possible Picking E as the subspace generated by the first k right-singular vectors of X solves the problem

Maximizing the projected variance

Let X be a $n \times p$ matrix, a reasonable task consists in finding a $k \lll p$-dimensional subspace E of $\mathbb{R}^p$ such that

$$X \times \Pi_E$$

retains as much variance as possible.
This means maximizing

$$\mathrm{var}\left(X \times \Pi_E\right)$$

PCA up and running

We will walk through Principal Component Analysis using historical datasets

- `USArrests`: crime data across states (individuals) in USA
- `iris`: morphological data of 150 flowers (individuals) from genus *iris*
- `decathlon`: scores of athletes at the ten events making a decathlon
- Life tables for a collection of Western European countries and the USA

> ⚠ Some datasets have non-numerical columns.
> Such columns are not used when computing the SVD that underpins PCA
> But non-numerical columns may be incorporated into PCA visualization

When PCA is used as a feature engineering device to prepare the dataset for regression, supervised classification, clustering or any other statistical/machine learning treatment, visualizing the interplay between non-numerical columns and the numerical columns constructed by PCA is crucial

Iris flower dataset

A showcase for Principal Component Analysis

> The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by Ronald Fisher in 1936. Edgar Anderson collected the data to quantify the morphologic variation of Iris flowers of three related species. Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus"
>
> The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters

Wikipedia

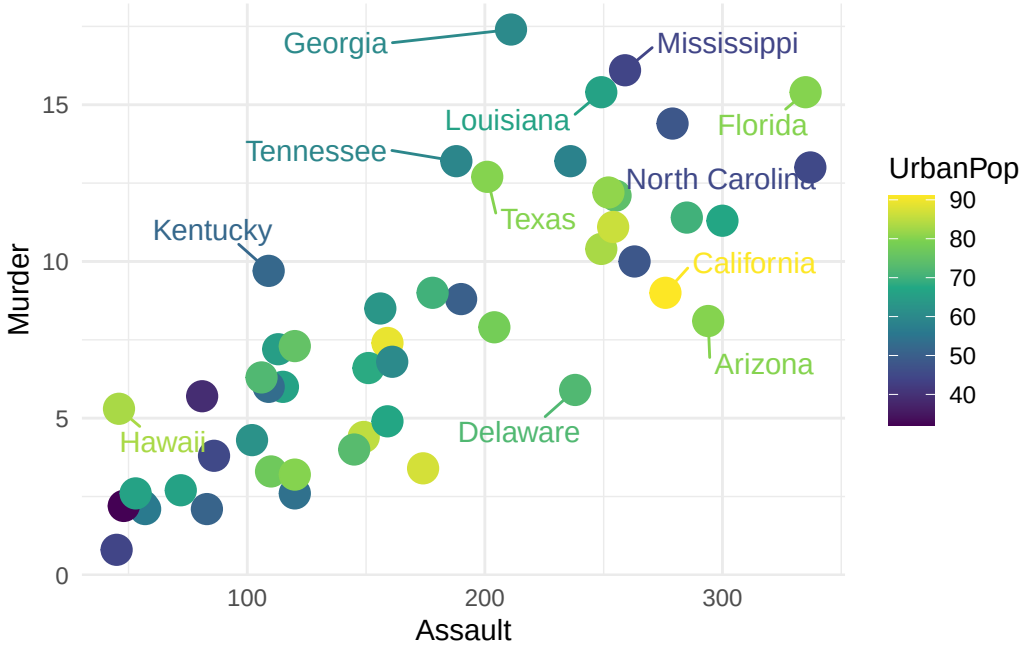
`USArrests`

```
data(USArrests)

USArrests %>%
  sample_n(10) %>%
  knitr::kable()
```

	Murder	Assault	UrbanPop	Rape
Maine	2.1	83	51	7.8
Alaska	10.0	263	48	44.5
New York	11.1	254	86	26.1
Idaho	2.6	120	54	14.2
Illinois	10.4	249	83	24.0
Louisiana	15.4	249	66	22.2
South Carolina	14.4	279	48	22.5
Pennsylvania	6.3	106	72	14.9
Missouri	9.0	178	70	28.2
Kentucky	9.7	109	52	16.3

```
USArrests %>%
  rownames_to_column() %>%
  ggplot() +
  aes(x = Assault, y = Murder) +
  aes(colour = UrbanPop, label=rowname) +
  geom_point(size = 5) +
  geom_text_repel(box.padding = unit(0.75, "lines"))
```

```
Warning: ggrepel: 38 unlabeled data points (too many overlaps). Consider
increasing max.overlaps
```



Decathlon

```
data(decathlon)
```

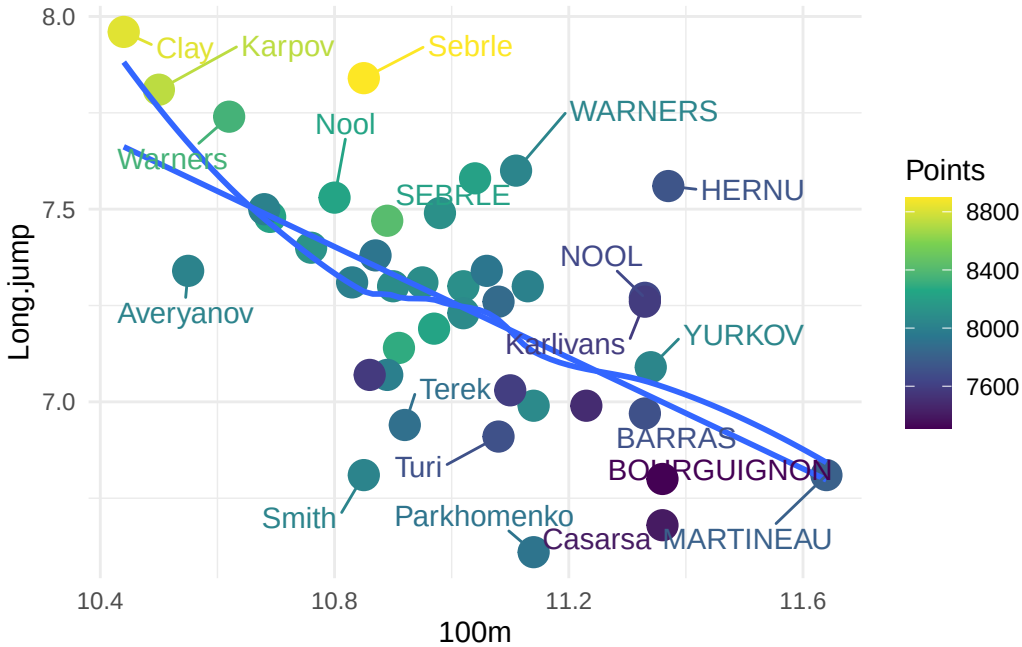
```r
decathlon %>%
  rownames_to_column() %>%
  dplyr::select(-c(`400m`, `110m.hurdle`, `Shot.put`,`Javeline`, `1500m`, `Rank`, `Competition
  sample_n(10) %>%
  knitr::kable()
```

rowname	100m	Long.jump	High.jump	Discus	Pole.vault
Drews	10.87	7.38	1.88	40.11	5.00
Lorenzo	11.10	7.03	1.85	40.22	4.50
Uldal	11.23	6.99	1.85	43.01	4.50
Qi	11.06	7.34	1.97	45.13	4.50
Smirnov	10.89	7.07	1.94	42.47	4.70
Averyanov	10.55	7.34	1.94	39.88	4.80
WARNERS	11.11	7.60	1.98	41.10	4.92
SEBRLE	11.04	7.58	2.07	43.75	5.02
NOOL	11.33	7.27	1.98	37.92	4.62
Macey	10.89	7.47	2.15	48.34	4.40

```r
decathlon %>%
  rownames_to_column() %>%
  ggplot() +
  aes(x = `100m`, y = Long.jump) +
  aes(colour = Points, label=rowname) +
  geom_point(size = 5) +
  geom_smooth(se = FALSE) +
  geom_smooth(method="lm", se=FALSE) +
  geom_text_repel(box.padding = unit(0.75, "lines"))
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'

Warning: The following aesthetics were dropped during statistical transformation: colour
and label.
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?

`geom_smooth()` using formula = 'y ~ x'

Warning: The following aesthetics were dropped during statistical transformation: colour
and label.
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?

Warning: ggrepel: 21 unlabeled data points (too many overlaps). Consider
```

```
increasing max.overlaps
```



Using `prcomp` from base R

> The calculation is done by a singular value decomposition of the (centered and possibly scaled) data matrix, not by using `eigen` on the covariance matrix.
>
> This is generally the preferred method for numerical accuracy.
>
> The `print` method for these objects prints the results in a nice format and the plot method produces a *scree plot*

At first glance, performing Principal Component Analysis consists in applying SVD to the $n \times p$ data matrix.

Raw PCA on `iris` dataset

```
pca_iris <- iris %>%
  select(- Species) %>%
  prcomp(x = .,          #<<
         center = FALSE, #<<
         scale. = FALSE) #<<

class(pca_iris) ; is.list(pca_iris)
```

```
[1] "prcomp"
```

```
[1] TRUE
```

```
names(pca_iris)
```

```
[1] "sdev"     "rotation" "center"   "scale"    "x"
```

Result has 5 components:

- `sdev`: singular values
- `rotation`: orthogonal matrix V in SVD
- `center`: `FALSE` or centering row vector
- `scale`: FALSE of scaling row vector
- `x`: matrix $U \times D$ from SVD

From data to PCA and back

Let

- Z be the $n \times p$ matrix fed to `prcomp`
- X be the $n \times p$ matrix forming component `x` of the result
- V be the $p \times p$ matrix forming component `rotation` of the result
- μ be the centering vector, $\mathbf{0}$ if `center=FALSE`
- σ be the scaling vector, $\mathbf{1}$ is `scale.=FALSE`

then

$$\mathrm{diag}(\sigma) \times (Z - \mathbf{1} \times \mu^T) = X \times V^T$$

or

$$Z = \mathrm{diag}(\sigma)^{-1} \times X \times V^T + \mathbf{1} \times \mu^T$$

PCA can be pictured as follows

- optional centering and scaling
- performing SVD
- gathering the byproducts of SVD so as to please statisticians
- possibly truncating the result so as to obtain a truncated SVD

Glossary

The columns of components `x` are called the *principal components* of the dataset

⚠ Note that the precise connection between the principal components and the dataset depend on centering and scaling

Conventions

PCA vocabulary	SVD vocabulary
Principal axis	Left singular vector
Principal component	Scaled left singular vector

PCA vocabulary	SVD vocabulary
Factorial axis	Right singular vector
Component of inertia	Squared singular value

Modus operandi

1. Read the data.
2. Choose the active individuals and variables (cherrypick amongs numeric variables)
3. Choose if the variables need to be standardised
4. Choose the number of dimensions you want to keep (the *rank*)
5. Analyse the results
6. Automatically describe the dimensions of variability
7. Back to raw data.

Impact of standardization and centering

When performing PCA, once the *active variables* have been chosen, we wonder whether we should centre and/or standardize the columns

> PCA investigates the spread of the data not their location
> Centering is performed by default when using `prcomp`

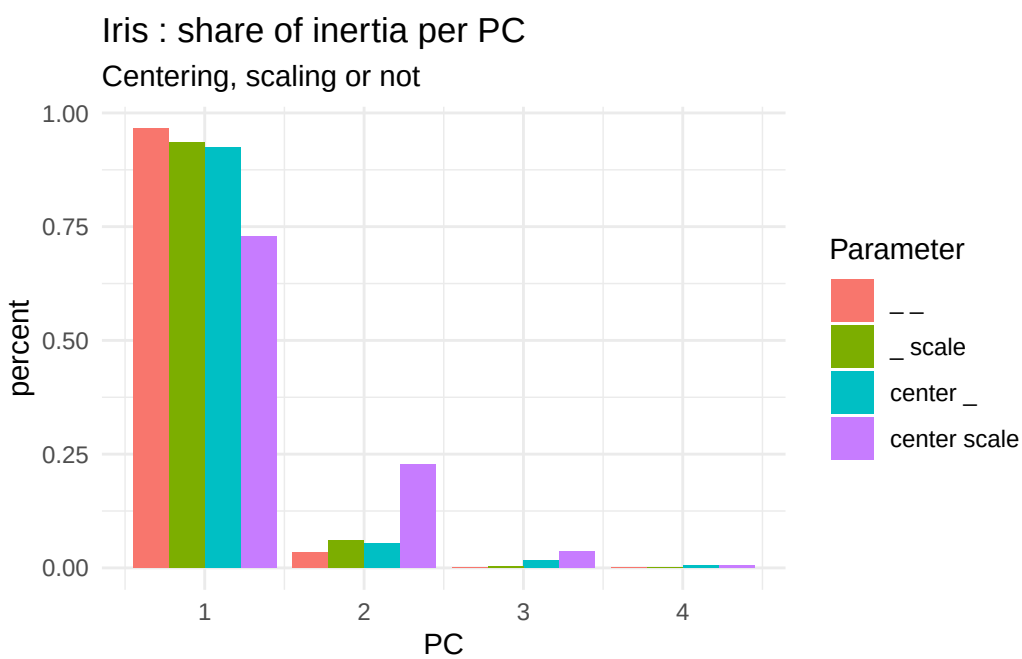
Centering/Scaling: iris dataset

On iris dataset Centering and standardizing the columns slightly spreads out the eigenvalues

```
iris <- datasets::iris
iris_num <- dplyr::select(iris, -Species)

list_pca_iris <-
  map2(.x=rep(c(FALSE, TRUE), 2),
       .y=rep(c(FALSE, TRUE), c(2,2)),
       ~ prcomp(iris_num, center= .x, scale.=.y)
)

names(list_pca_iris) <- stringr::str_c("pca_iris",
    c("", "c", "s", "c_s"),
    sep="_")
```

```
config_param <- as.vector(outer(c("_", "center"),
                                c("_", "scale"), FUN=paste)) %>%
                rep( c(4,4,4,4))
```

```
list_pca_iris %>%
  purrr::map_dfr(~ broom::tidy(., matrix="pcs")) %>%
  mutate(Parameter=config_param) -> df_pca_iris_c_s

p_pca_c_s <- df_pca_iris_c_s %>%
  ggplot() +
  aes(x=PC, y=percent, fill=Parameter) +
  geom_col(position="dodge")

p_pca_c_s +
  labs(title="Iris : share of inertia per PC",
       subtitle = "Centering, scaling or not")
```



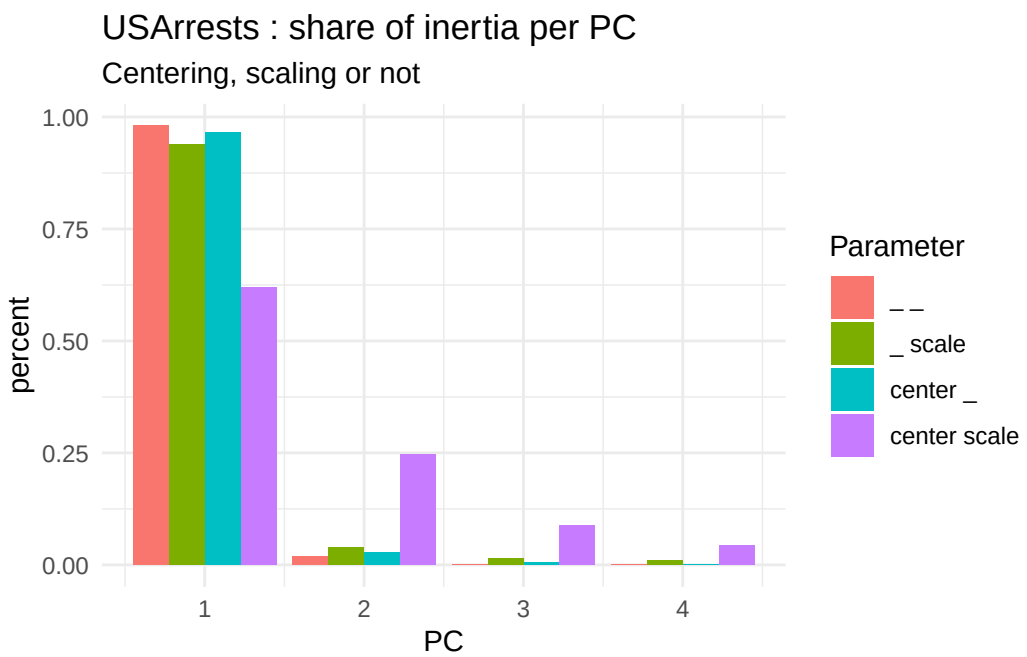
More on the centering/scaling dilemma

```
usarrests_num <- USArrests
data(decathlon)

list_pca_usarrests <-
  map2(.x=rep(c(FALSE, TRUE), 2),
       .y=rep(c(FALSE, TRUE), c(2,2)),
       ~ prcomp(USArrests, center= .x, scale.=.y)
)

list_pca_decathlon <-
  map2(.x=rep(c(FALSE, TRUE), 2),
       .y=rep(c(FALSE, TRUE), c(2,2)),
```

```
        ~ prcomp(decathlon[, 1:10], center= .x, scale.=.y)
)

names(list_pca_usarrests) <- c("pca_usarrests", "pca_usarrests_c", "pca_usarrests_s", "pca_usar

names(list_pca_decathlon) <- c("pca_decathlon", "pca_decathlon_c", "pca_decathlon_s", "pca_deca
```

> For `USArrests`, scaling is mandatory

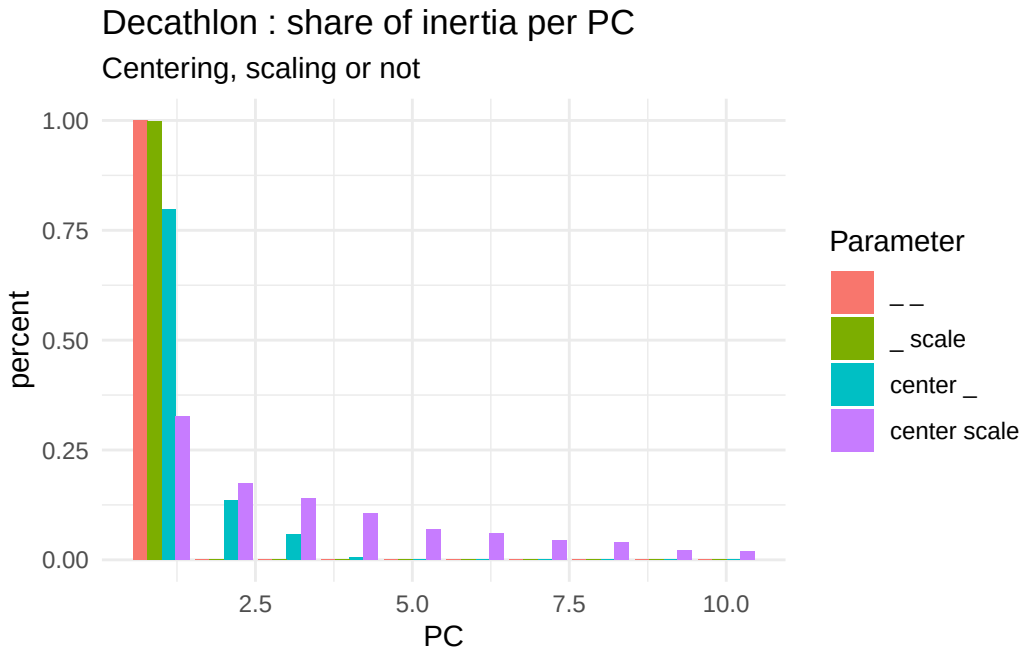
```
list_pca_usarrests %>%
  purrr::map_dfr(~ broom::tidy(., matrix="pcs")) %>%
  mutate(Parameter=config_param) -> df_pca_usarrests_c_s

p_pca_c_s %+%
  df_pca_usarrests_c_s  +
  labs(title="USArrests : share of inertia per PC",
       subtitle = "Centering, scaling or not")
```



For `decathlon`, centering is mandatory, scaling is recommended

```
config_param_deca <- as.vector(outer(c("_", "center"),
                           c("_", "scale"), FUN=paste)) %>%
                 rep(rep(10,4))

list_pca_decathlon %>%
  purrr::map_dfr(~ broom::tidy(., matrix="pcs")) %>%
  mutate(Parameter=config_param_deca) -> df_pca_decathlon_c_s
```

```r
p_pca_c_s %+%
  df_pca_decathlon_c_s  +
  labs(title="Decathlon : share of inertia per PC",
       subtitle = "Centering, scaling or not")
```



In order to investigate the columns means, there is no need to use PCA. Centering is almost always relevant.

The goal of PCA is to understand the fluctuations of the data and, when possible, to show that most fluctuations can be found in a few priviledged directions

Scaling is often convenient: it does not alter correlations. Second, high variance columns artificially capture an excessive amount of inertia, without telling us much about the correlations between variables

Visualizing PCA

Recall Joliffe's big picture

- Visualize the correlation plot
- Visualize the share of inertia per principal component (screeplot)
- Visualize the individuals in the factorial planes generated by the leading principal axes
- Visualize the (scaled) orginal variables in the basis defined by the factorial axes (correlation circle)
- Visualize simultaneously individuals and variables (biplot)

Inspecting the correlation matrix

Using `corrr::...`

```
iris %>%
  dplyr::select(where(is.numeric)) %>%
  corrr::correlate() -> tb #<<
```

```
Correlation computed with
* Method: 'pearson'
* Missing treated using: 'pairwise.complete.obs'
```

```
tb %>%
  corrr::shave() %>%       #<<
  corrr::fashion() %>%     #<<
  knitr::kable(format="markdown")
```

term	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length				
Sepal.Width	-.12			
Petal.Length	.87	-.43		
Petal.Width	.82	-.37	.96	

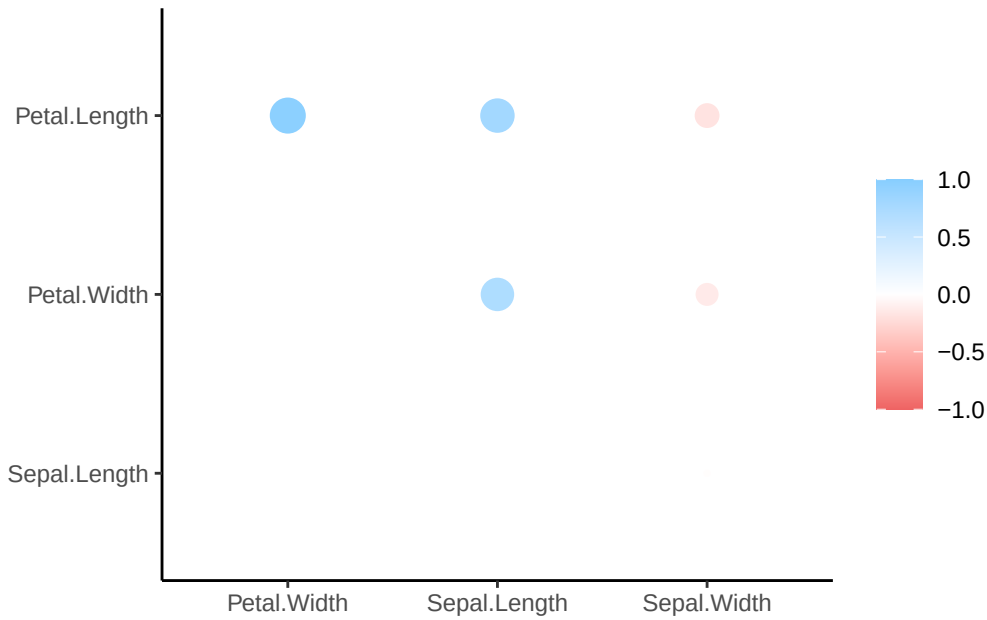
About `corrr::...`

Package `corrr`

- `correlate` explores pairwise correlations just as `cor`
- It outputs a dataframe (a `tibble`)
- The `corrr` API is designed with data pipelines in mind
- `corrr` offers functions for manipulating correlation matrices

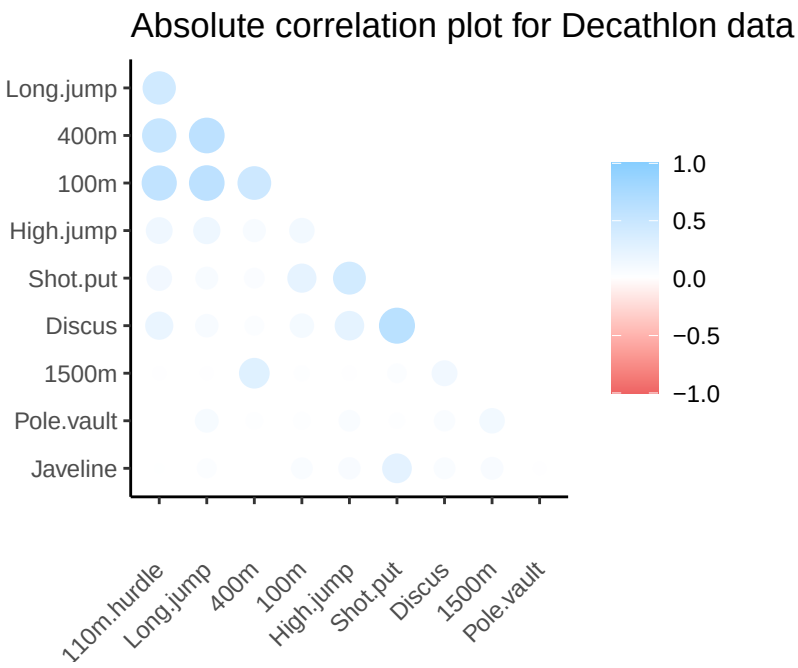
Other plots

```
tb %>%
  corrr::rearrange(absolute = FALSE) %>%
  corrr::shave(upper = FALSE) %>%
  corrr::rplot()  #<<
```



Correlation plot for `decathlon`

```
decathlon  %>%
  rownames_to_column("Name") %>%     #<< tidyverse does not like rownames
  mutate(across(-c(Name, Rank, Points, Competition),
                ~ (.x -mean(.x))/sd(.x))) -> decathlonR2

decathlonR2 %>%
  dplyr::select(-c(Name, Rank, Points, Competition)) %>%
  corrr::correlate(quiet = TRUE, method="pearson") %>%
  mutate(across(where(is.numeric), abs)) %>%
  corrr::rearrange() %>%
  corrr::shave() -> df_corr_decathlon

df_corr_decathlon %>%
  corrr::rplot() +
  coord_fixed() +
  ggtitle("Absolute correlation plot for Decathlon data") +
  theme(axis.text.x = element_text(angle = 45, vjust = 0.5, hjust=1))
```



If our goal is to partitition the set of columns into tightly connected groups, this correlation plot already tells us a story:

- short running races `100m`, ..., `400m` along `long jump` form a cluster,
- `shot.put`, `discus`, and to a lesser extent `high jump` form another cluster
- `pole.vault`, `1500m` and `javeline` look pretty isolated

PCA Visualization : first goals

- Look at the data in PC coordinates (Factorial plane)
- Look at the rotation matrix (Factorial axes)
- Look at the variance explained by each PC (Screeplot)

Inspect eigenvalues

```
iris_pca <- list_pca_iris[[4]]

iris_pca %>%
  broom::tidy(matrix="pcs") %>%
  knitr::kable(format="markdown", digits=2)
```

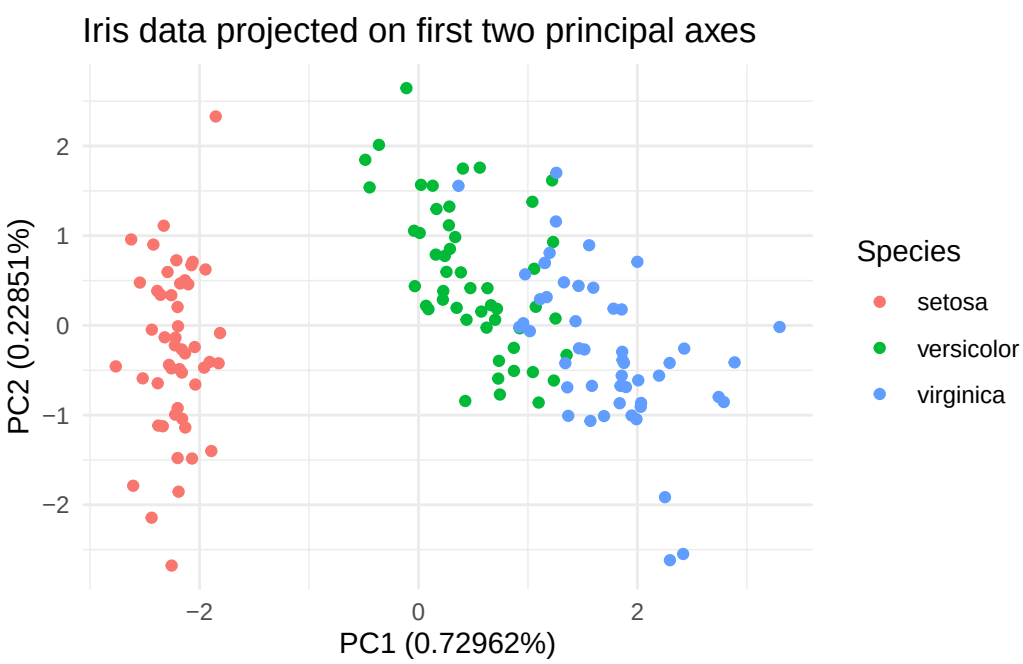
PC	std.dev	percent	cumulative
1	1.71	0.73	0.73
2	0.96	0.23	0.96
3	0.38	0.04	0.99
4	0.14	0.01	1.00

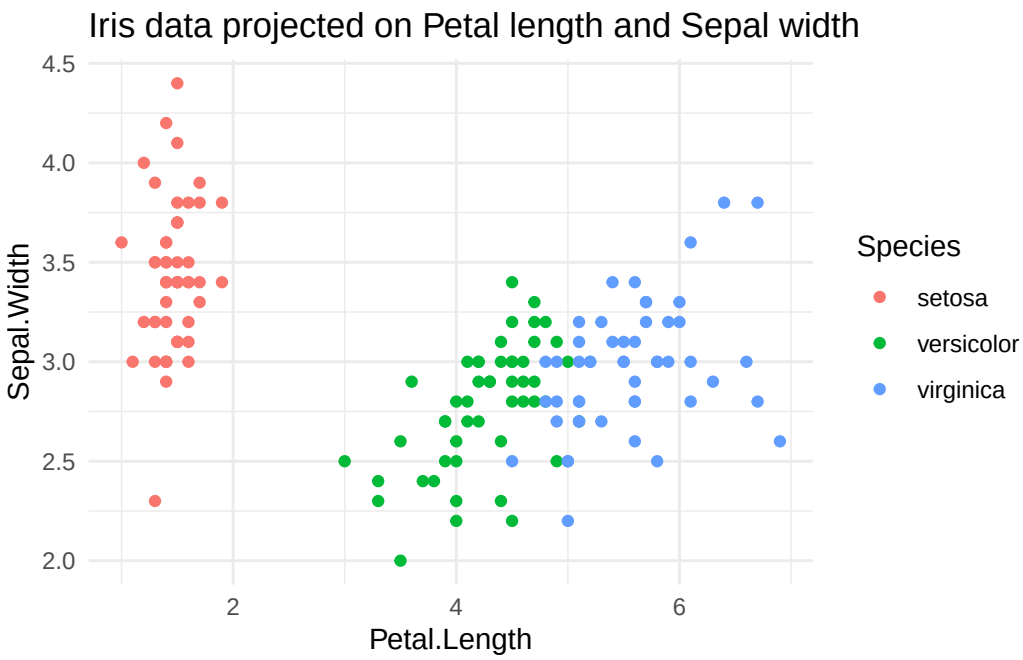
```
iris_plus_pca <- broom::augment(iris_pca, iris)
```

Scatterplots

```
share_variance <- broom::tidy(iris_pca, "pcs")[["percent"]]

iris_plus_pca %>%
  ggplot() +
  aes(x=.fittedPC1, y=.fittedPC2) +
  aes(colour=Species) +
  geom_point() +
  ggtitle("Iris data projected on first two principal axes") +
  xlab(paste("PC1 (", share_variance[1], "%)", sep="")) +
  ylab(paste("PC2 (", share_variance[2], "%)", sep="")) -> p

iris_plus_pca %>%
  ggplot() +
  aes(y=Sepal.Width, x=Petal.Length) +
  aes(colour=Species) +
  geom_point() +
  ggtitle("Iris data projected on Petal length and Sepal width") -> q
```

```
p ; q
```

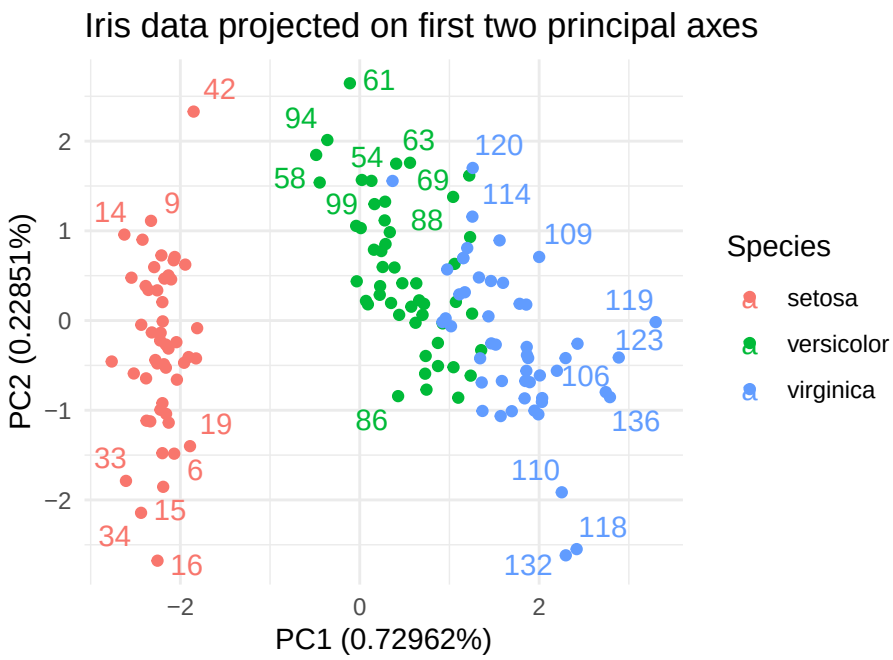




Biplot : overplotting the original variables in the factorial plane

```
total_inertia <- sum(share_variance)

p +
  aes(label=.rownames) +
  geom_text_repel(verbose = FALSE) +
  coord_fixed()
```

```
Warning: ggrepel: 122 unlabeled data points (too many overlaps). Consider
increasing max.overlaps
```

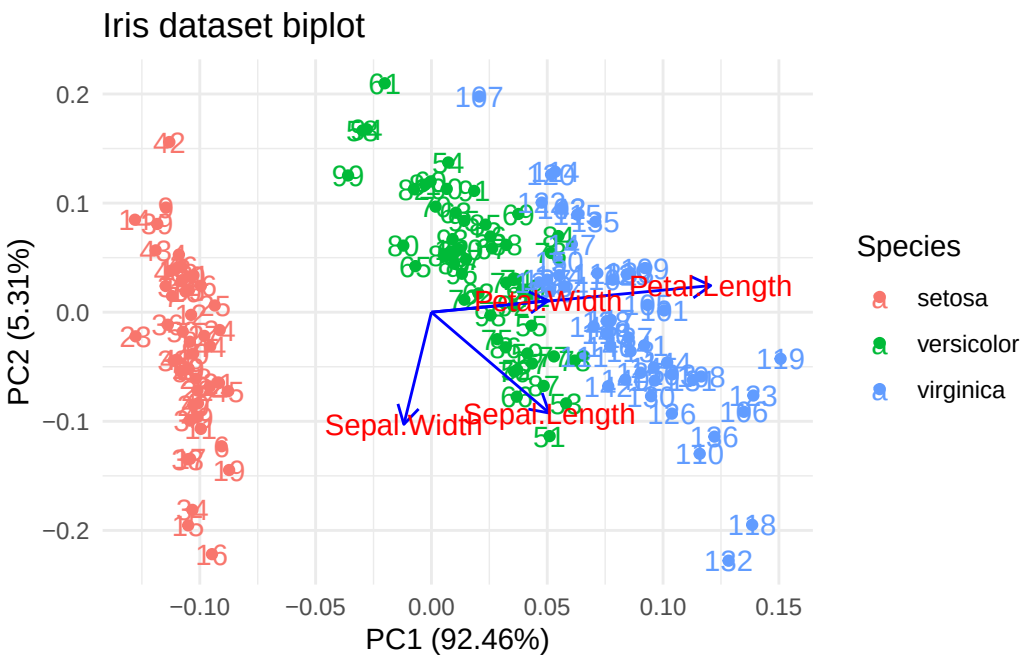


See plot_pca

This is a scatterplot on the plane spanned by the two leading principal components

```
iris %>%
  select(where(is.numeric)) %>%
  prcomp(.scale=TRUE) %>%
  autoplot(                    #<<
   data = iris,
   colour = 'Species',
   label = TRUE,               #<<
   loadings = TRUE,            #<<
   loadings.colour = 'blue',
   loadings.label = TRUE
  ) +
  ggtitle("Iris dataset biplot")
```

```
Warning: In prcomp.default(., .scale = TRUE) :
 extra argument '.scale' will be disregarded
```



Scatterplot of the factorial plane is overplotted with *loadings* corresponding to native variables

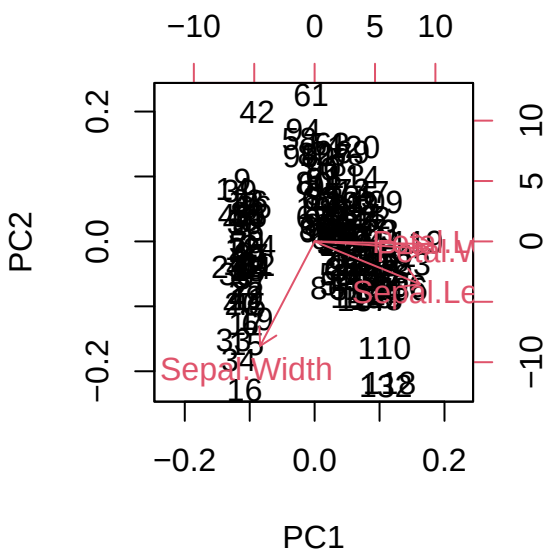
!

```
require(plotly)

(
autoplot(prcomp(iris[,-5]),
         data = iris,
         colour = 'Species',
         label = TRUE,
         loadings = TRUE,
         loadings.label = TRUE) +
  ggtitle("Iris dataset ")
) %>% ggplotly()
```

Official plots for PCA on iris dataset

```
plot(list_pca_iris[[4]])
```

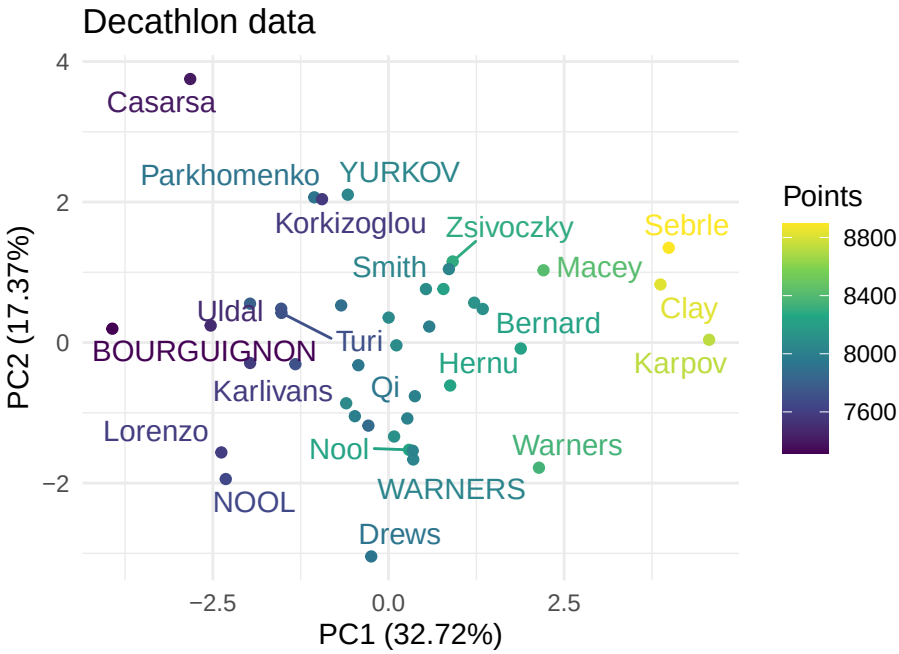

```
biplot(list_pca_iris[[4]])
```



Plotting PCA on decathlon data

```
decathlon_pca <- list_pca_decathlon[[4]]
inertias <- decathlon_pca$sdev^2
share_variance <- round((100*(inertias)/
                         sum(inertias)), 2)

p %+% broom::augment(decathlon_pca, decathlon) +
  aes(colour=Points) +
  ggtitle("Decathlon data") +
  xlab(paste("PC1 (", share_variance[1], "%)", sep="")) +
  ylab(paste("PC2 (", share_variance[2], "%)", sep="")) +
  aes(label=.rownames) +
  geom_text_repel(verbose = FALSE) +
  coord_fixed()
```

```
Warning: ggrepel: 18 unlabeled data points (too many overlaps). Consider
increasing max.overlaps
```

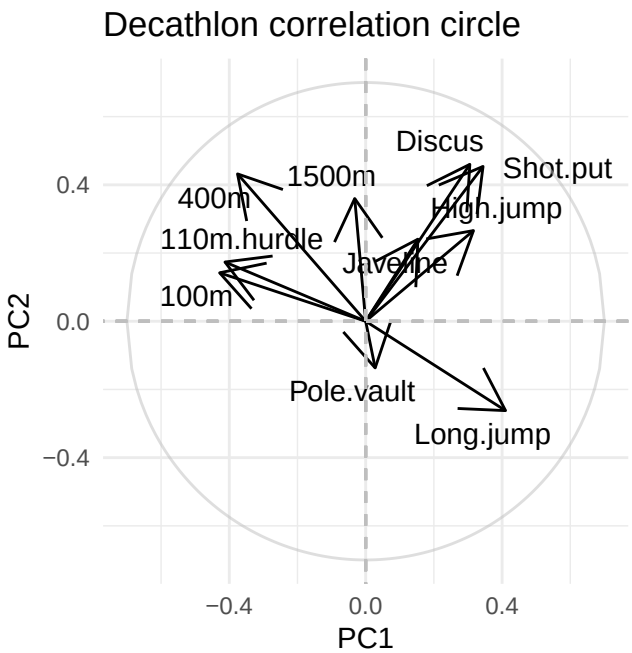


Decathlon correlation circle

```
decathlon_pca$rotation %>%
  as_tibble(rownames="Name") %>%
  ggplot() +
  aes(x=PC1, y=PC2, label=Name) +
  geom_segment(xend=0, yend=0, arrow = grid::arrow(ends = "first")) +
  geom_text_repel() +
  coord_fixed() +
  xlim(-.7, .7) +
  ylim(-.7, .7) +
  stat_function(fun = ~  sqrt(.7^2 - .^2), alpha=.5, color="grey") +
  stat_function(fun = ~ - sqrt(.7^2 - .^2), alpha=.5, color="grey") +
  geom_segment(x = 0, y = -1, xend = 0, yend = 1, linetype = "dashed", color = "grey") +
  geom_segment(x = -1, y = 0, xend = 1, yend = 0, linetype = "dashed", color = "grey") +
  ggtitle("Decathlon correlation circle")
```

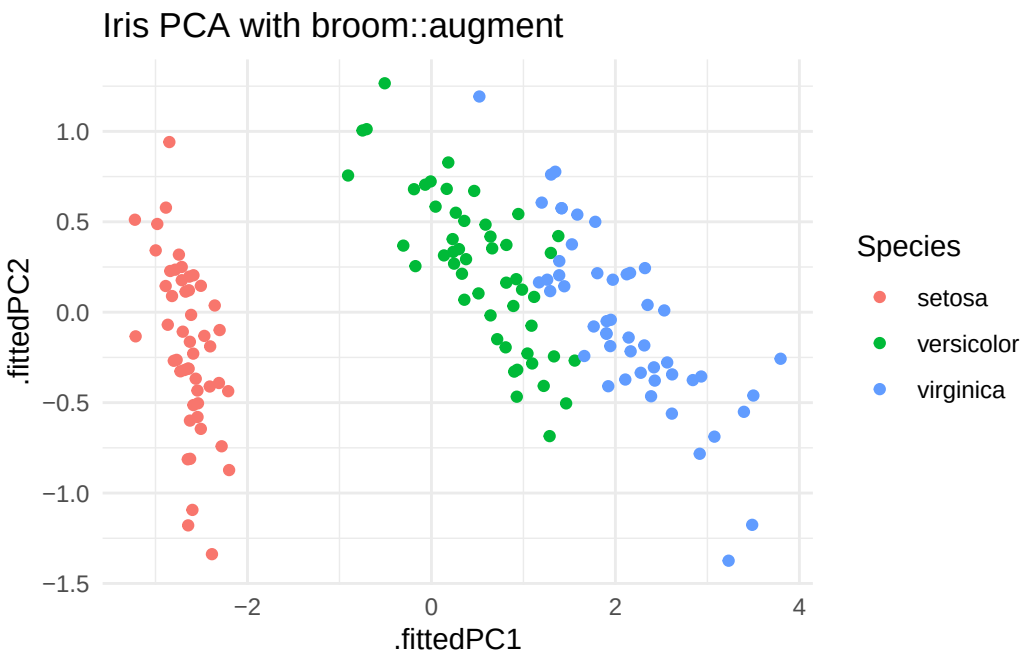
```
Warning: The following aesthetics were dropped during statistical transformation: label.
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?
The following aesthetics were dropped during statistical transformation: label.
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
```

```
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?
```



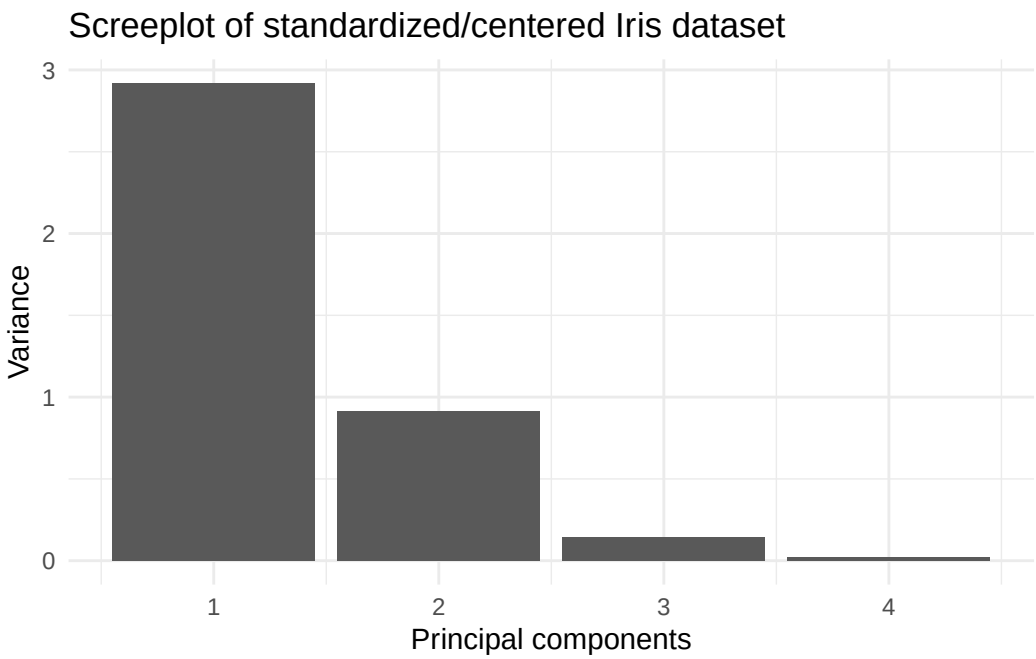
Using `broom::augment`

```
pca_fit <- prcomp(iris[,-5])

pca_fit %>%
  broom::augment(iris) %>% #<<  add original dataset back in
  ggplot(aes(x= .fittedPC1, y=.fittedPC2, color = Species)) +
  geom_point(size = 1.5) +
  ggtitle("Iris PCA with broom::augment") -> p

p
```



Screeplot

```
zpca <- list_pca_iris[[4]]

tibble(x=1:4, y=zpca$sdev^2) |>
  ggplot() +
  aes(x=x, y=y) +
  geom_col() +
  xlab("Principal components") +
  ylab("Variance") +
  ggtitle("Screeplot of standardized/centered Iris dataset")
```

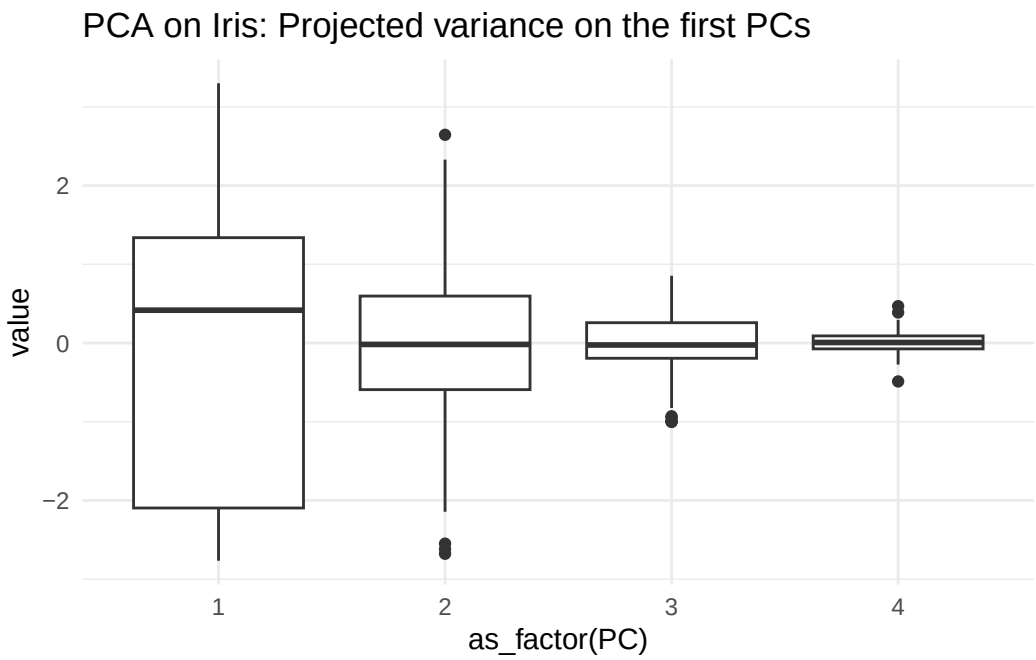


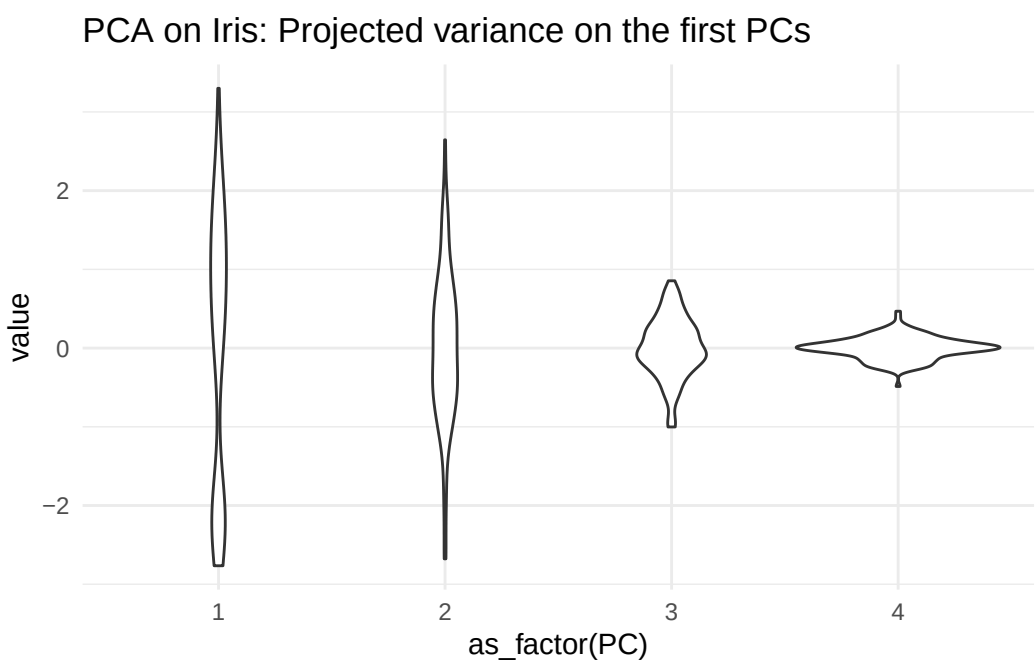
Variations on screeplot

```
zetitle <-  "PCA on Iris: Projected variance on the first PCs"

p <-list_pca_iris[[4]] %>%
  broom::tidy() %>%
  ggplot() +
  aes(x= as_factor(PC), y= value) +
  labs(xlab="PC", ylab="Coord") +
  ggtitle(zetitle)
```

> We center and standardize columns, and then perform a change of basis

```
p + geom_boxplot()  ; p + geom_violin()
```





Two packages

Several packages handle the field of factorial analysis

`ADE4`

- 2002-...
- Duality diagrams: `dudi.pca`
- Ecology oriented
- ADE4 Homepage

`FactoMineR`

- 2003-...
- `Factoshiny`
- `FactoExtra`
- `FactoInvestigate`
- A nice book
- A MOOC

FactoMineR homepage

http://factominer.free.fr

FactoMineR on PCA

http://factominer.free.fr/factomethods/principal-components-analysis.html

FactoMineR : scatterplot and correlation circle

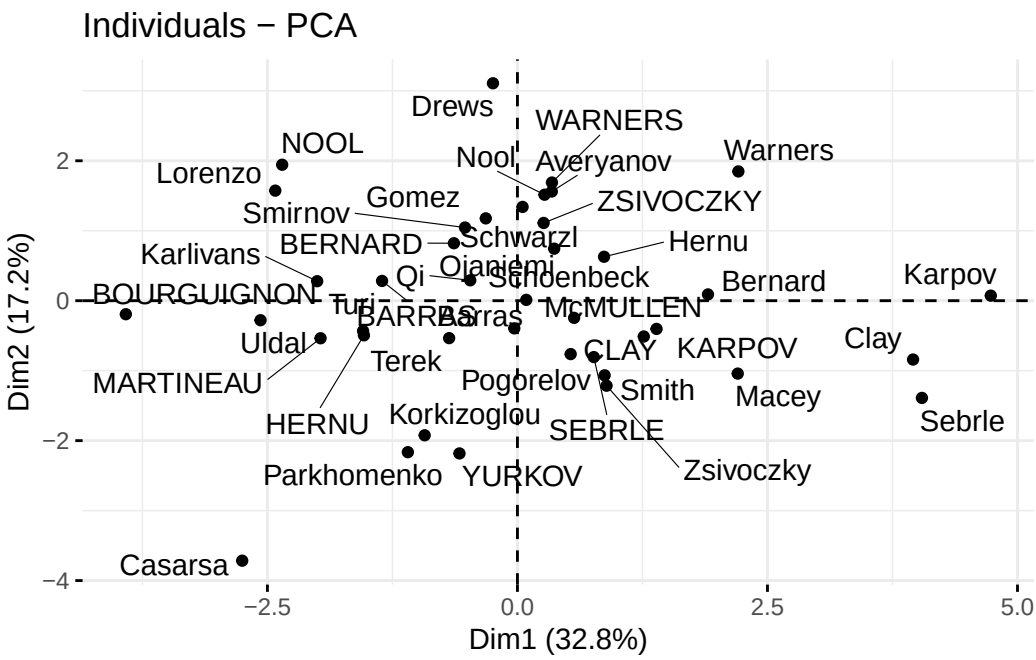
```
iris %>%
  dplyr::select(where(is.numeric)) %>%
  FactoMineR::PCA() %>%
  plot(graph.type="ggplot",
       title="Iris data with FactoMineR")
```

Revisiting Decathlon data

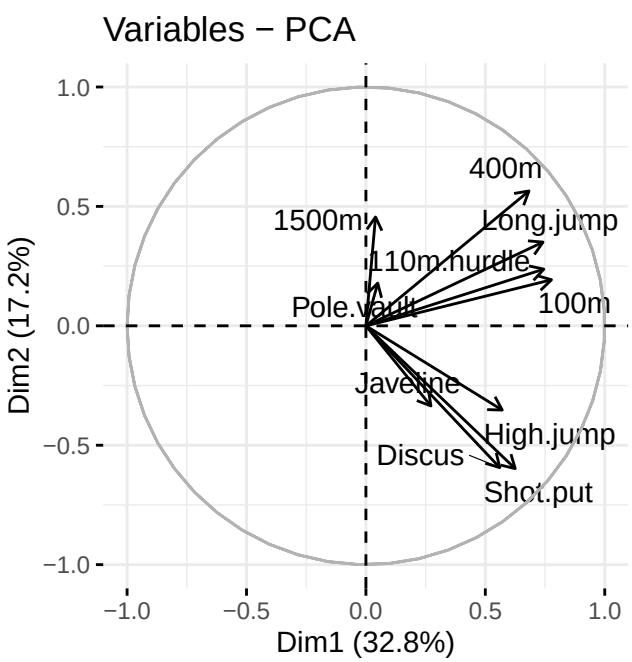
Ad hoc transformation of running scores: average speed matters

```
decathlonTime <- mutate(decathlon,
                        `100m` = 100/`100m`,
                        `400m` = 400/`400m`,
                        `110m.hurdle` = 110/`110m.hurdle`,
                        `1500m` = 1500/`1500m`)

# rownames(decathlonTime)  <- rownames(decathlon)

PCAdecathlontime <- PCA(decathlonTime[,1:10],  graph = FALSE)
```

```
factoextra::fviz_pca_ind(PCAdecathlontime, repel = TRUE)
```




```
factoextra::fviz_pca_var(PCAdecathlontime, repel = TRUE)
```



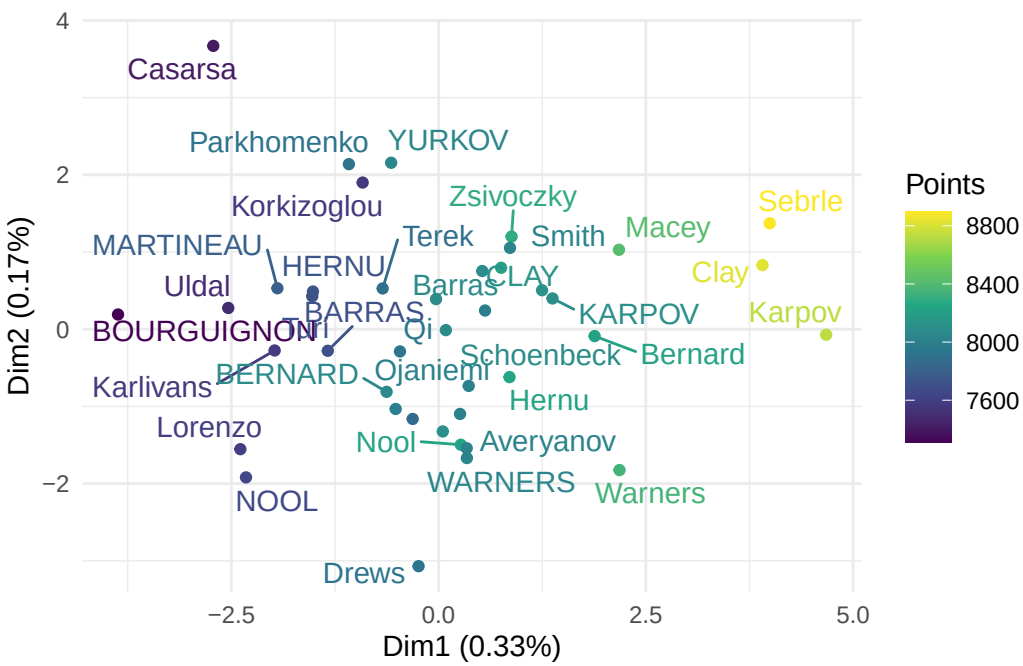
When inspecting `decathlon` some columns are positively correlated with column `Points`, others are not. The negatively correlated columns contain running races times. In order to handle all scores in an homogenous way, running races scores are converted into average speeds. The larger, the better.

The correlation circle becomes more transparent: columns can be clustered into three subsets: - running races and long jump, p - puts, javeline and high jump - 1500m and pole vault which are poorly correlated with first two principal components

We can do it with `tidyverse`

```
decathlonTime[, 1:10] %>%
  prcomp(scale.=TRUE) -> pc_decathlonTime

share_variance <-  round(broom::tidy(pc_decathlonTime, matrix="pcs")[["percent"]], 2)

pc_decathlonTime  %>%
  broom::augment(data=decathlonTime) %>%
  ggplot() +
  aes(x=.fittedPC1, y=.fittedPC2) +
  aes(color=Points, label=.rownames) +
  geom_point() +
  geom_text_repel() +
  xlab(paste("Dim1 (", share_variance[1]  ,"%)", sep="")) +
  ylab(paste("Dim2 (", share_variance[2]  ,"%)", sep="")) 
```

```
Warning: ggrepel: 7 unlabeled data points (too many overlaps). Consider
increasing max.overlaps
```

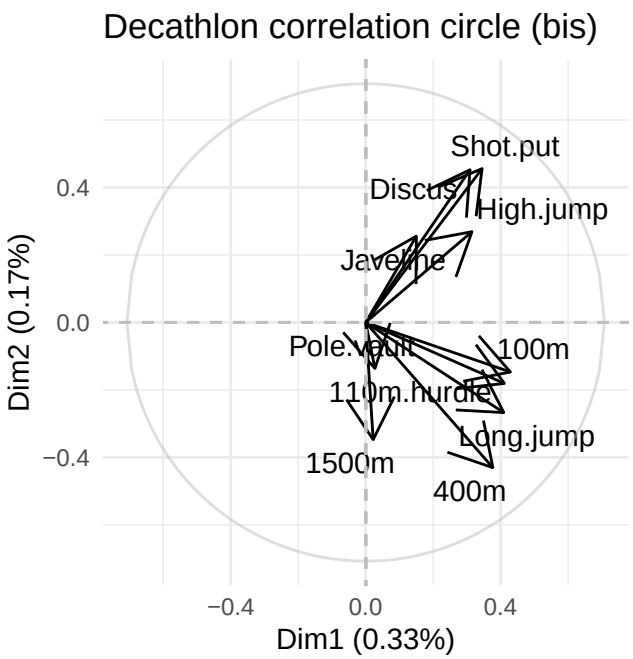



```
radius <-  sqrt(broom::tidy(pc_decathlonTime, matrix="pcs")[["cumulative"]][2])

pc_decathlonTime[["rotation"]] %>%
  as_tibble(rownames="Name") %>%
  ggplot() +
  aes(x=PC1, y=PC2, label=Name) +
  geom_segment(xend=0, yend=0, arrow = grid::arrow(ends = "first") ) +
  geom_text_repel() +
  coord_fixed() +
```

```
  xlim(-radius, radius) +
  ylim(-radius, radius) +
  stat_function(fun = ~  sqrt(radius^2 - .^2), alpha=.5, color="grey") +
  stat_function(fun = ~ - sqrt(radius^2 - .^2), alpha=.5, color="grey") +
  geom_segment(x = 0, y = -1, xend = 0, yend = 1, linetype = "dashed", color = "grey") +
  geom_segment(x = -1, y = 0, xend = 1, yend = 0, linetype = "dashed", color = "grey") +
  ggtitle("Decathlon correlation circle (bis)") +
  xlab(paste("Dim1 (", share_variance[1]  ,"%)", sep="")) +
  ylab(paste("Dim2 (", share_variance[2]  ,"%)", sep=""))
```

```
Warning: The following aesthetics were dropped during statistical transformation: label.
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?
```

```
Warning: The following aesthetics were dropped during statistical transformation: label.
i This can happen when ggplot fails to infer the correct grouping structure in
  the data.
i Did you forget to specify a `group` aesthetic or to convert a numerical
  variable into a factor?
```



PCA and the tidy approach

The output of `prcomp` (or of any contender) is complex object represented by a list with five named elements: `x`, `sdev`, `rotation`, `center` `scale`

In R, the list is assigned a dedicated `S3` class: `prcomp`

The `prcomp` class is not `ggplot2` friendly

In the `tidyverse` framework, visualization abides to the *grammar of graphics* framework: plots are built on dataframes (usually a single dataframe)

The different plots that serve to understand the output of PCA (screeplot, correlation circle, biplot, …) have to be built on dataframes that themselves have to be built from the output of PCA and possibly also from the original dataframe

Package `broom` ( ) offers off-the-shelf components for building PCA graphical pipelines (and many other things)

> `broom` is an attempt to bridge the gap from untidy outputs of predictions and estimations to the tidy data we want to work with. It centers around three `S3` methods, each of which take common objects produced by R statistical functions (`lm`, `t.test`, `nls`, `prcomp`, etc) and convert them into a `tibble`
>
> `broom` is particularly designed to work with Hadley's `dplyr` package (see the broom+dplyr vignette for more)

Three (generic) functions

`tidy` constructs a tibble that summarizes the model's statistical findings. This includes coefficients and p-values for each term in a regression, per-cluster information in clustering applications, or per-test information for multtest functions

`augment` add columns to the original data that was modeled. This includes predictions, residuals, and cluster assignments

`glance` construct a concise one-row summary of the model. There is no such method for class `prcomp`

`tidy` for class `prcomp`

```r
require(broom)
```

```r
pc <- prcomp(USArrests, scale = TRUE)

# information about samples  (U x D matrix)
# Default behaviour

broom::tidy(pc, matrix="samples") %>%
  head(5) %>%
  knitr::kable()
```

row	PC	value
Alabama	1	-0.9756604
Alabama	2	-1.1220012
Alabama	3	0.4398037
Alabama	4	0.1546966
Alaska	1	-1.9305379

The output of `tidy.prcomp` is always a `tibble`

Depending on argument `matrix`, the output gathers information on different components of the SVD factorization

`tidy(pc, "samples")` provide a long format tibble version of $U \times D$

The tibble has three columns

- `row`: rownames in the dataframe that served as input to `prcomp`
- `PC`: index of principal components
- `value`: score of individual indexed by `row` on principal components `PC`, $(U \times D)[\text{row}, \text{PC}]$

The `x` component of `pc` is a *wide format* version of the output of `tidy(pc, "samples")`

Documentation

```
as_tibble(pc$x) %>%
  add_column(row=rownames(pc$x)) %>%
  pivot_longer(starts_with("PC"),
               names_to="PC",
               names_prefix="PC") %>%
  head(5)
```

```
# A tibble: 5 x 3
  row     PC     value
  <chr>   <chr>  <dbl>
1 Alabama 1     -0.976
2 Alabama 2     -1.12
3 Alabama 3      0.440
4 Alabama 4      0.155
5 Alaska  1     -1.93
```

`tidy` for class `prcomp`

```
pc <- prcomp(USArrests, scale = TRUE)

# information about rotation (V matrix)
broom::tidy(pc, "rotation") %>%
  head(5)
```

```
# A tibble: 5 x 3
  column     PC  value
  <chr>   <dbl>  <dbl>
1 Murder      1 -0.536
2 Murder      2 -0.418
3 Murder      3  0.341
4 Murder      4  0.649
5 Assault     1 -0.583
```

With `matrix="rotation"`, the result is again a long format tibble version of the orthogonal matrix V from the SVD factorization

This tibble is convenient when plotting the correlation circles associated with different sets of components

```
as_tibble(pc$rotation) %>%
  add_column(column=rownames(pc$rotation)) %>%
  pivot_longer(starts_with("PC"),
               names_to="PC",
               names_prefix="PC") %>%
  head()
```

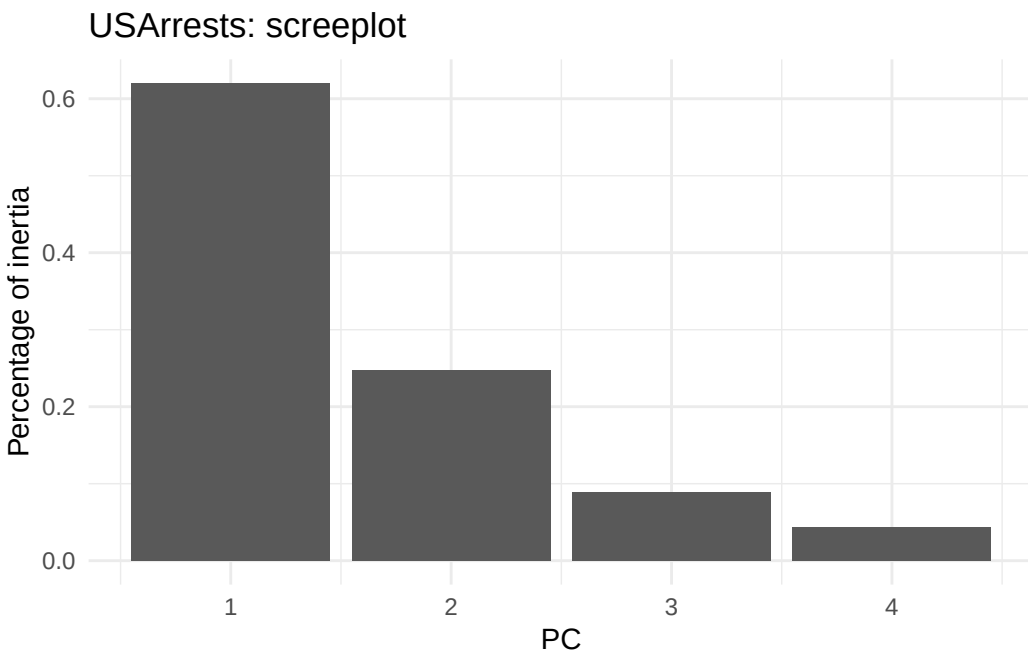
```
# A tibble: 6 x 3
  column  PC     value
  <chr>   <chr>  <dbl>
1 Murder  1     -0.536
2 Murder  2     -0.418
3 Murder  3      0.341
4 Murder  4      0.649
5 Assault 1     -0.583
6 Assault 2     -0.188
```

🧹 `tidy` for class `prcomp`

```
# information about singular values
broom::tidy(pc, "pcs") %>%
  knitr::kable()
```

PC	std.dev	percent	cumulative
1	1.5748783	0.62006	0.62006
2	0.9948694	0.24744	0.86750
3	0.5971291	0.08914	0.95664
4	0.4164494	0.04336	1.00000

```
broom::tidy(pc, "pcs") %>%
  ggplot() +
  aes(x=as.integer(PC), y=percent) +
  geom_col() +
  xlab("PC") +ylab("Percentage of inertia") +
  ggtitle("USArrests: screeplot")
```



With `matrix="pcs"` or `matrix="eigen"`, we obtain a tibble that is convenient for generating a `screeplot`

It contains the information returned by `summary(pc)`

Indeed, column `percent` is obtained by squaring column `std.dev` and normalizing the column

See `sloop::s3_get_method(tidy.prcomp)`

or

```
as_tibble(pc$sdev) %>%
  rename(std.dev=value) %>%
  rownames_to_column("PC") %>%
  mutate(PC=as.integer(PC),
         percent=(std.dev^2/sum(std.dev^2)),
         cumulative=cumsum(percent))
```

```
# A tibble: 4 x 4
     PC std.dev percent cumulative
  <int>   <dbl>   <dbl>      <dbl>
1     1   1.57   0.620       0.620
2     2   0.995  0.247       0.868
3     3   0.597  0.0891      0.957
4     4   0.416  0.0434      1
```

```
# or
t(summary(pc)$importance) %>%
  data.frame() %>%
  rownames_to_column()
```

```
  rowname Standard.deviation Proportion.of.Variance Cumulative.Proportion
```

```
1     PC1          1.5748783                0.62006                0.62006
2     PC2          0.9948694                0.24744                0.86750
3     PC3          0.5971291                0.08914                0.95664
4     PC4          0.4164494                0.04336                1.00000
```

Package

Vignette

`broom`'s ambitions go far beyond tidying the output of `prcomp(...)`

The vignette describes the goals of the package and the way they fit in the `tidverse`

`broom::tidy()`: a generic `S3` function

Dispatching to ad hoc methods

`broom::tidy()` is an `S3` generic function.

When called, it invokes a dispatcher that calls a method tailored to the class of the object passed as argument

```
> broom::tidy
function (x, ...)
{
    UseMethod("tidy")
}
```

> When a function calling `UseMethod("fun")` is applied to an object with class attribute `c("first", "second")`, the system searches for a function called `fun.first` and, if it finds it, applies it to the object…

The hard work is performed by the `tidy.prcomp()` registered for class `prcomp`

```r
class(pc)
```

```
[1] "prcomp"
```

```r
# sloop::s3_get_method(tidy.prcomp)
```

See S3 section in Advanced R Programmming

`broom::augment()`

An excerpt of the body of `augment.prcomp`

```
function (x, data = NULL, newdata, ...)
{
    ret <- if (!missing(newdata)) {
        # ...
    }
    else {
        pred <- as.data.frame(predict(x))
```

```
        names(pred) <- paste0(".fitted", names(pred))
        if (!missing(data) && !is.null(data)) {
            cbind(.rownames = rownames(as.data.frame(data)),
                data, pred)
        }
        else {
          # ...
        }
    as_tibble(ret)
}
```

Function `augment` is also an `S3` generic function.

The method relies on another generic function `predict`

For `prcomp` `predict` returns the `x` component of the list

The output combines the columns of the original dataframe and (`data`) with their names and the predictions

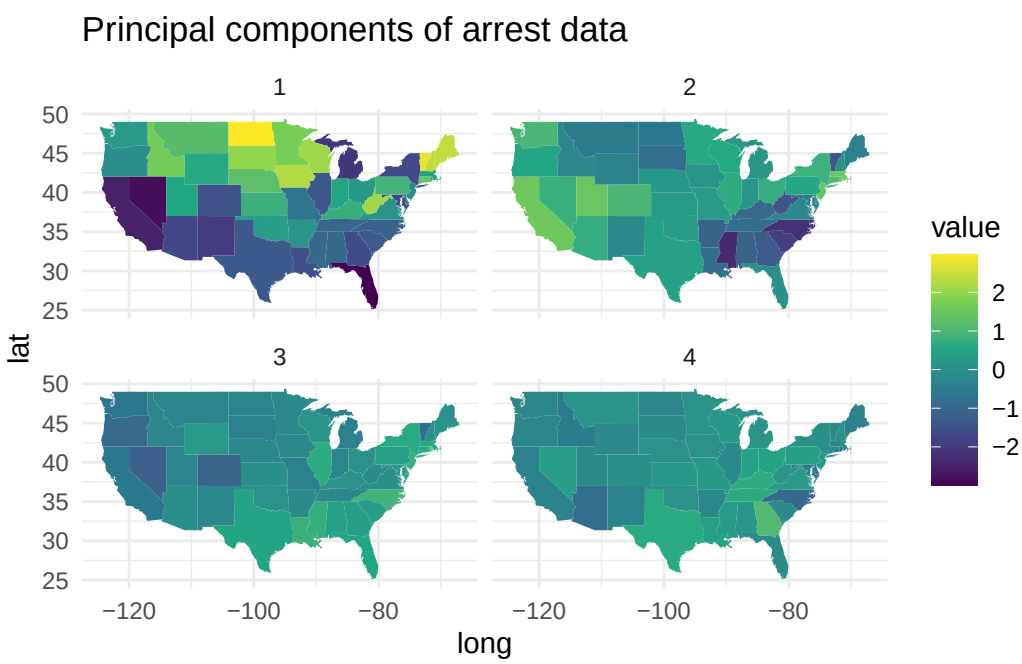
🎨 Mixing PCA and geographical information

```r
require("maps")
```

```
Loading required package: maps


Attaching package: 'maps'

The following object is masked from 'package:purrr':

    map
```

```r
pc %>%
  broom::tidy(matrix = "samples") %>%
  mutate(region = tolower(row)) %>%
  inner_join(map_data("state"), by = "region") %>%
  ggplot() +
  aes(long, lat) +
  aes(group = group, fill = value) +
  geom_polygon() +
  facet_wrap(~PC) +
  scale_fill_viridis_c() +
  ggtitle("Principal components of arrest data")
```

```
Warning in inner_join(., map_data("state"), by = "region"): Detected an unexpected many-to-many
i Row 1 of `x` matches multiple rows in `y`.
i Row 1 of `y` matches multiple rows in `x`.
i If a many-to-many relationship is expected, set `relationship =
  "many-to-many"` to silence this warning.
```

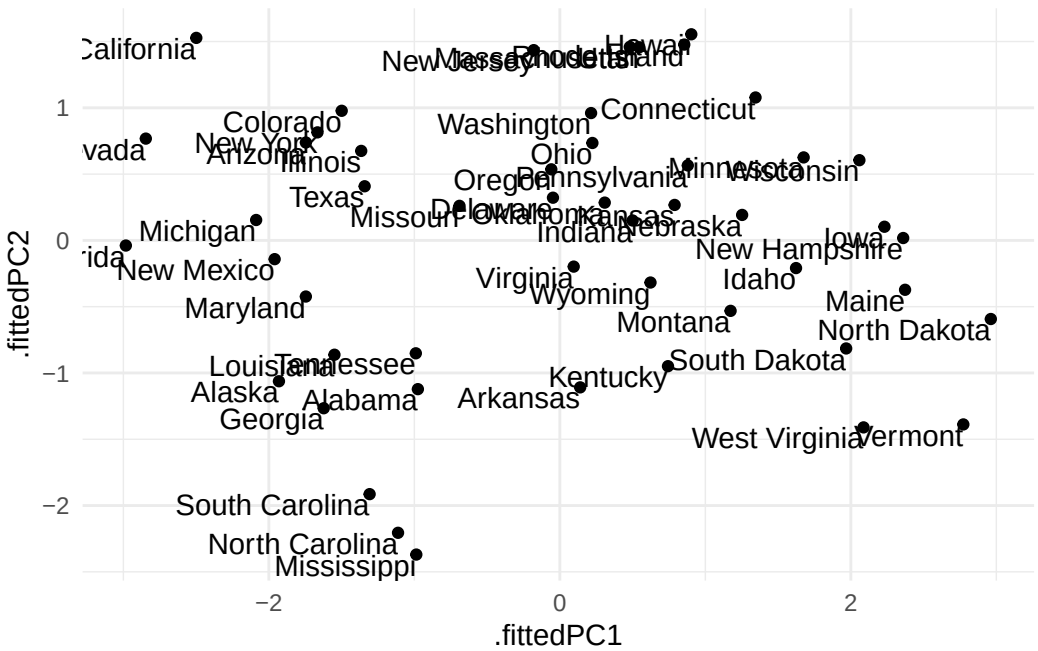


augment for prcomp

```
au <- augment(pc, data = USArrests)

au %>% head(3)
```

```
# A tibble: 3 x 9
  .rownames Murder Assault UrbanPop  Rape .fittedPC1 .fittedPC2 .fittedPC3
  <chr>      <dbl>   <int>    <int> <dbl>      <dbl>      <dbl>      <dbl>
1 Alabama     13.2     236       58  21.2     -0.976     -1.12      0.440
2 Alaska      10       263       48  44.5     -1.93      -1.06     -2.02
3 Arizona      8.1     294       80  31       -1.75       0.738    -0.0542
# i 1 more variable: .fittedPC4 <dbl>
```

```
ggplot(au) +
  aes(.fittedPC1, .fittedPC2) +
  geom_point() +
  geom_text(aes(label = .rownames), vjust = 1, hjust = 1)
```


California
New Jersey
Massachusetts
Rhode Island
Hawaii
Connecticut
Washington
Colorado
New York
Arizona
Illinois
Ohio
Oregon
Pennsylvania
Minnesota
Wisconsin
Texas
Missouri
Delaware
Oklahoma
Kansas
Indiana
Nebraska
Michigan
New Hampshire
Iowa
New Mexico
Virginia
Wyoming
Idaho
Maine
Maryland
Montana
North Dakota
South Dakota
Louisiana
Tennessee
Kentucky
Alaska
Alabama
Arkansas
Georgia
West Virginia
Vermont
South Carolina
North Carolina
Mississippi
.fittedPC2
.fittedPC1
1
0
−1
−2
−2
0
2