Momocs: Using R to analyze shape

We will go through the Momocs package. This is one of my favorite R packages and I have used it extensively to analyze leaf shape when I was a PhD student at Davis. For example in Martinez et al., 2015. My collegue at the time, Dan Chitwood introduced me to the package where he used it to create some stunning work analyzing modulations in shape in Grape Leaves and violins!

The package has changed a bit since I last visited. It appears as though Vincent Bonhomme has really embraced Tidyverse and has updated the `Momocs` package to reflect this. AND it appears that he is busily working on the Momocs 2.0 release (so stay tuned!).

Getting enviroment ready

```r
library(Momocs)
library(tidyverse)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2

## Warning: package 'readr' was built under R version 3.3.2

## Warning: package 'dplyr' was built under R version 3.3.2
```

Examining Objects

The Momocs package come with some great example datasets that we can use for examples, so we will just use these to make things easier.

First: Coo Objects

A shape is described in euclidian space and the Shapes can be organized into a collection of coordinates.

a Coo object that carries:

A component named `$coo`, a list of shapes (as matrix.ces); most of the time, a component named $fac, a data.frame to store covariates, either factors or numerics;possibly, other components of interest.

```r
## Check out one of the data sets:
shapes
```

```
## Out (outlines)
##   - 30 outlines, 836 +/- 255 coords (in $coo)
##   - 0 classifiers (in $fac):
## # A tibble: 0 x 0
##   - also: $ldk
```

```r
str(shapes)
```

```
## coo : List of 30
##  $ arrow   : num [1:888, 1:2] 200 199 198 197 197 196 195 194 193 192 ...
##  $ bone    : num [1:810, 1:2] 200 199 198 197 196 195 194 193 192 191 ...
##  $ buttefly: num [1:1077, 1:2] 200 200 200 199 199 199 198 198 197 197 ...
##  $ cat     : num [1:710, 1:2] 200 200 199 198 197 197 196 195 196 197 ...
##  $ check   : num [1:494, 1:2] 200 199 199 198 197 197 196 195 194 194 ...
##  $ cross   : num [1:806, 1:2] 200 200 199 198 198 197 197 196 195 195 ...
```

```
##  $ dog     : num [1:768, 1:2] 200 199 198 197 196 195 194 193 192 191 ...
##  $ fish    : num [1:943, 1:2] 200 199 198 197 196 195 194 193 192 191 ...
##  $ hand    : num [1:995, 1:2] 200 199 198 197 196 195 194 193 192 191 ...
##  $ hands   : num [1:1660, 1:2] 200 199 198 197 197 198 197 196 195 194 ...
##  $ heart   : num [1:554, 1:2] 200 199 198 197 197 198 199 198 198 197 ...
##  $ info    : num [1:514, 1:2] 200 199 198 197 196 195 194 193 192 191 ...
##  $ lady    : num [1:853, 1:2] 200 199 198 197 197 198 199 198 198 197 ...
##  $ leaf    : num [1:680, 1:2] 200 199 198 197 197 196 196 196 196 196 ...
##  $ leaf2   : num [1:792, 1:2] 200 200 201 200 200 200 200 200 200 199 ...
##  $ leaf3   : num [1:620, 1:2] 200 201 200 200 200 200 200 200 200 199 ...
##  $ leaf4   : num [1:1300, 1:2] 200 199 198 198 197 196 195 194 193 194 ...
##  $ moon    : num [1:571, 1:2] 200 199 198 197 196 195 194 193 192 191 ...
##  $ morph   : num [1:719, 1:2] 200 199 198 197 197 198 197 196 195 194 ...
##  $ parrot  : num [1:534, 1:2] 200 199 198 197 196 196 195 194 193 193 ...
##  $ penta   : num [1:957, 1:2] 200 199 198 197 197 196 196 197 196 196 ...
##  $ pigeon  : num [1:1240, 1:2] 200 199 198 197 196 195 194 193 192 191 ...
##  $ plane   : num [1:853, 1:2] 200 199 198 197 197 198 197 196 195 194 ...
##  $ puzzle  : num [1:778, 1:2] 200 199 198 197 197 198 199 198 197 196 ...
##  $ rabbit  : num [1:662, 1:2] 200 199 198 197 196 195 194 193 192 191 ...
##  $ sherrif : num [1:735, 1:2] 200 199 198 197 197 198 199 198 197 197 ...
##  $ snail   : num [1:930, 1:2] 200 199 198 197 196 195 194 193 192 191 ...
##  $ star    : num [1:1107, 1:2] 200 199 198 197 196 196 196 195 194 194 ...
##  $ tetra   : num [1:793, 1:2] 200 199 198 197 196 195 194 193 192 191 ...
##  $ umbrella: num [1:724, 1:2] 200 200 199 199 198 197 197 196 195 194 ...
## fac : Classes 'tbl_df', 'tbl' and 'data.frame':  0 obs. of  0 variables
## ldk :  list()
```

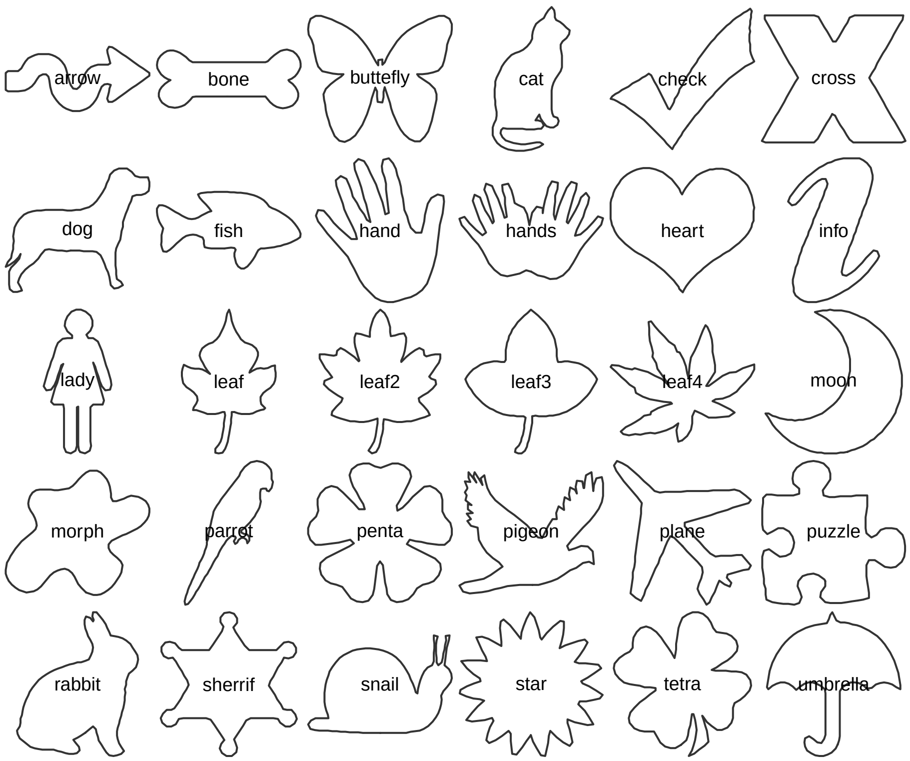
```
## Check out the coordinates of a single shape.
shapes[18] %>% head()
```

```
##      [,1] [,2]
## [1,]  200   50
## [2,]  199   49
## [3,]  198   49
## [4,]  197   50
## [5,]  196   50
## [6,]  195   49
```

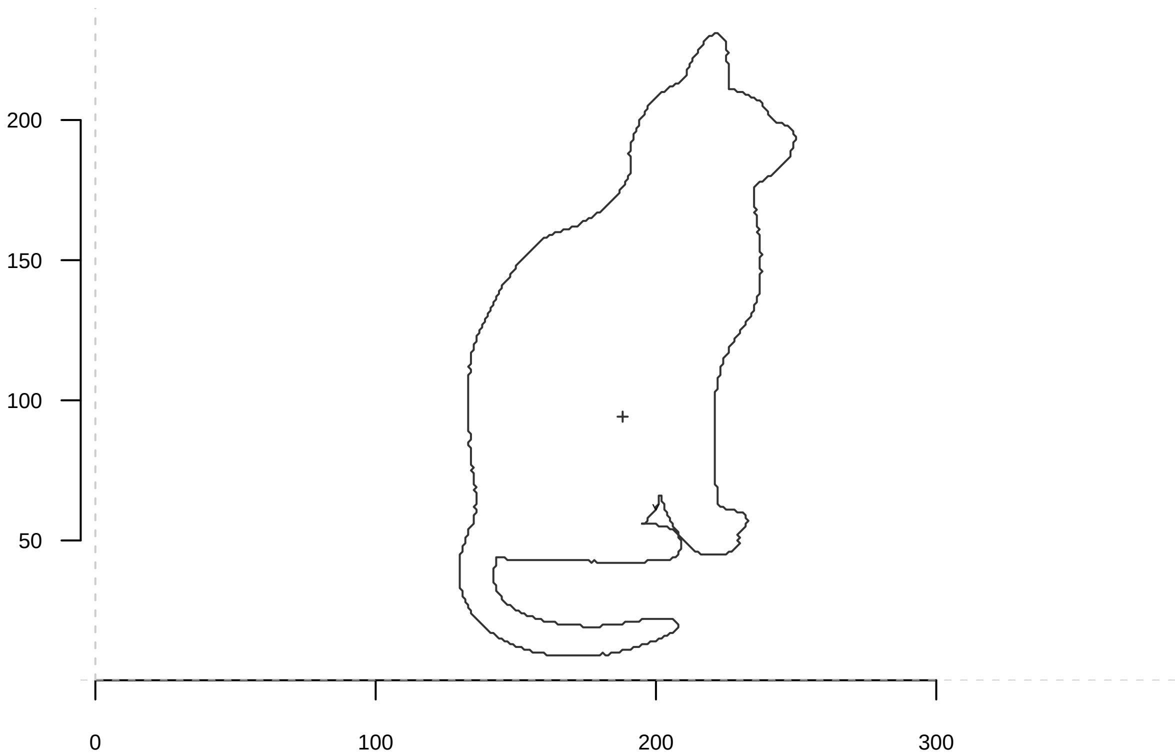
Visualizations

Since we are literally exploring shape, visualization of our data is extremely important. Momocs comes with a few different ways to visualize.

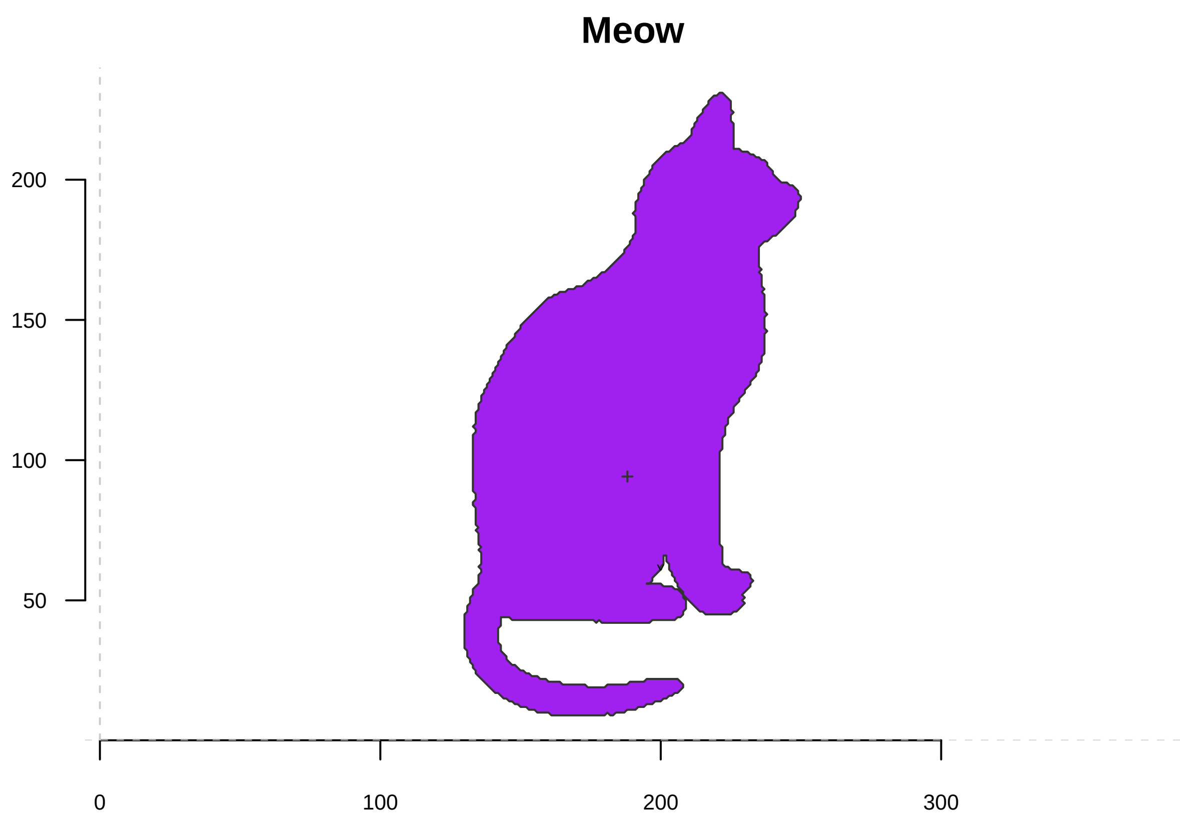
```
panel(shapes, names = TRUE) ## base R
```




```r
cat <- shapes[4]
coo_plot(cat)
```




```r
## #Rcatladies
coo_plot(cat, col="purple", main="Meow")
```



More detail into the data structure of Momocs

As discussed previously, a dataset (group of shapes) in Momocs is described as a Coo. Once you apply a method to that you get a Coe.

Coo + Morphometric method = Coe (x; y) coordinates + appropriate method = quantitative variables

Break for Discussion on S3 objects

- [ ] How many of you are coming from Python? Or use Python?
- [ ] How many of you use S3 and know what it is in R?
- [ ] How is S3 different from classes in Python?
- [ ] What about S4 and S5 then?!

** What is the difference between S3 and 4?** - S3 can only dispatch on it's first argument, whereas S4 can dispatch on multiple arguments. If you want to be able to write methods for function foo that should do different things if given an object of class "bar" or given objects of class "bar" and "foobar", or given objects of class "barfoo" and "foobar", then S4 provides a far better way to handle such complexities. (ref)

More on S3: * Read Advanced R The S3 object system Chapter

So, let's delve a bit more into our data.

```
class(shapes)
```

```
## [1] "Out" "Coo"
```

Does that make sense?

Another example of a more popular class - the tibble

```r
class(iris)
```

```
## [1] "data.frame"
```

```r
iris_tibble <- as_data_frame(iris)

## Tibbles have 3 classes
class(iris_tibble)
```

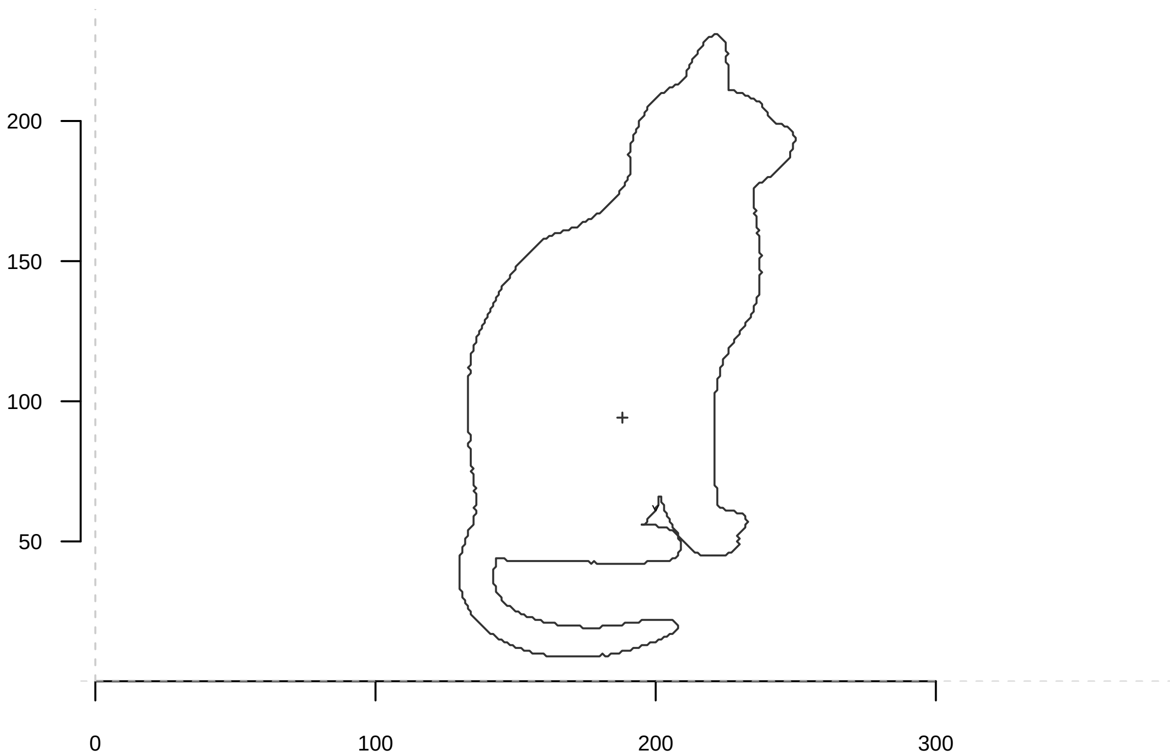
```
## [1] "tbl_df"     "tbl"        "data.frame"
```

Now back to Momocs. . . .

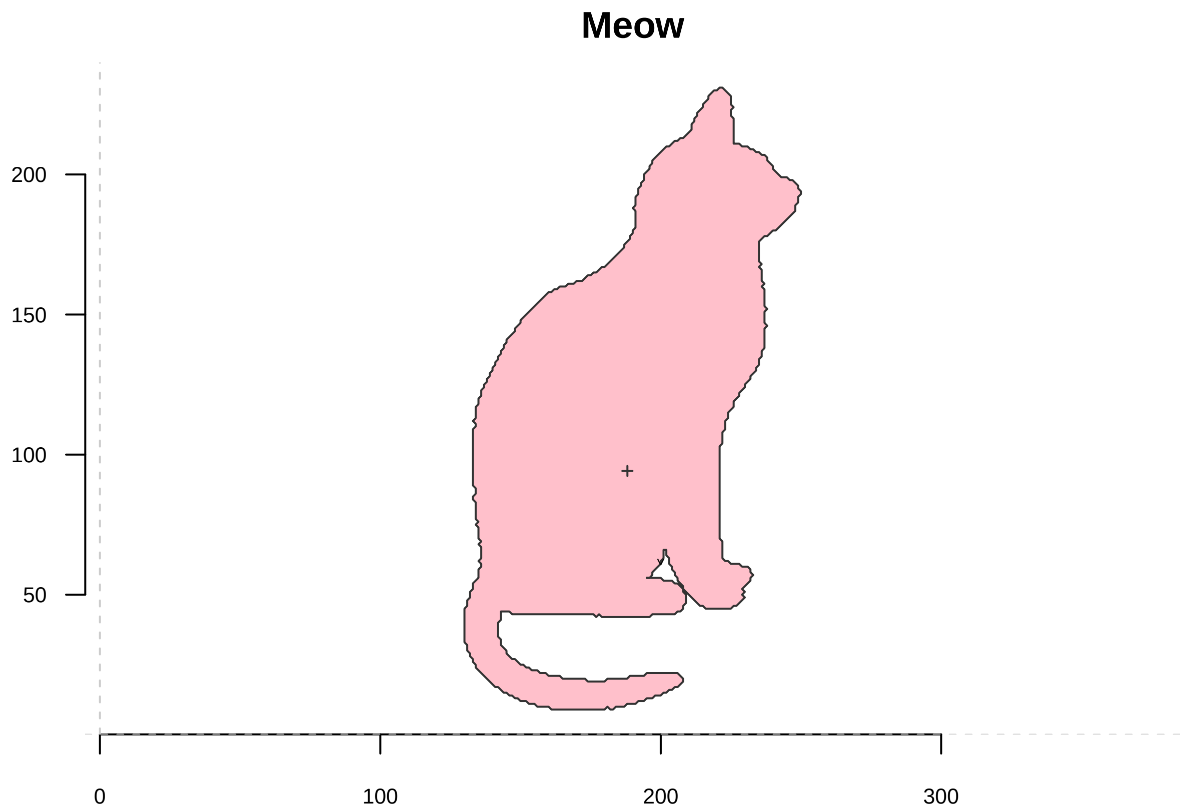
```r
## Let's check out the Cat Shape
cat <- shapes[4]
class(cat) # just a matrix
```

```
## [1] "matrix"
```

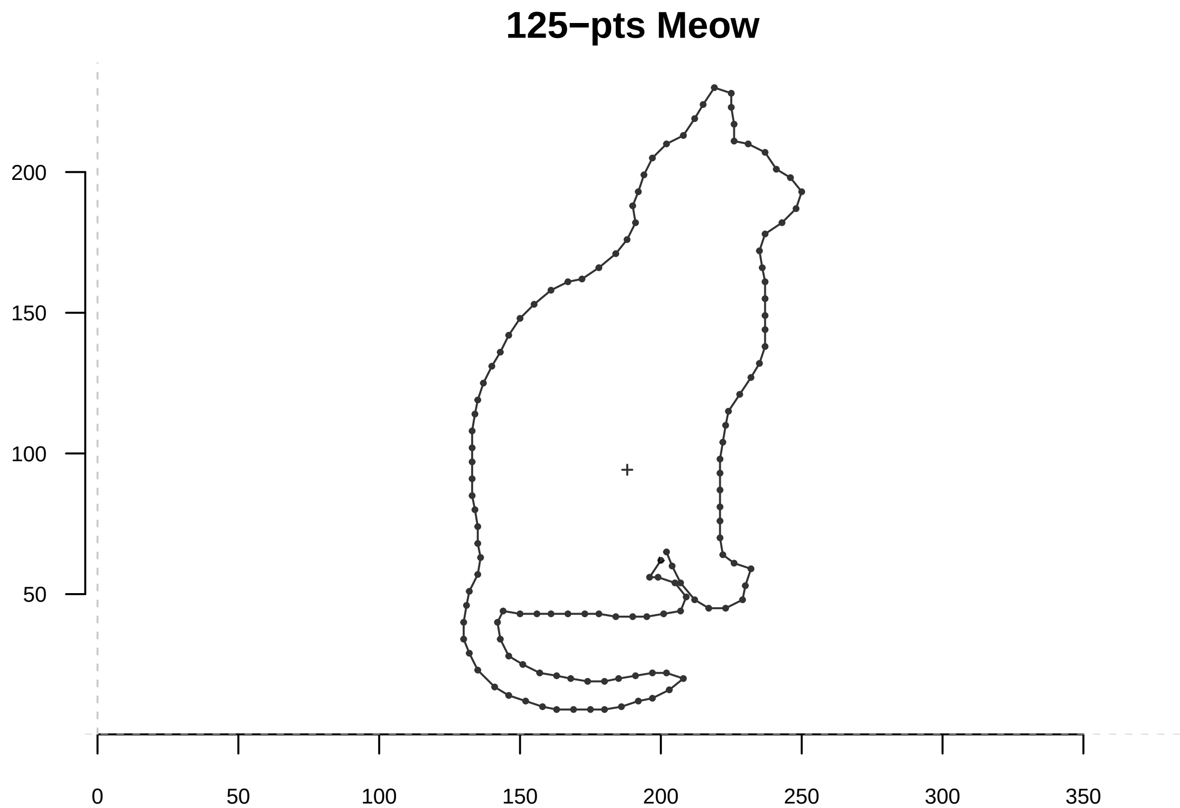
```r
coo_plot(cat)
```



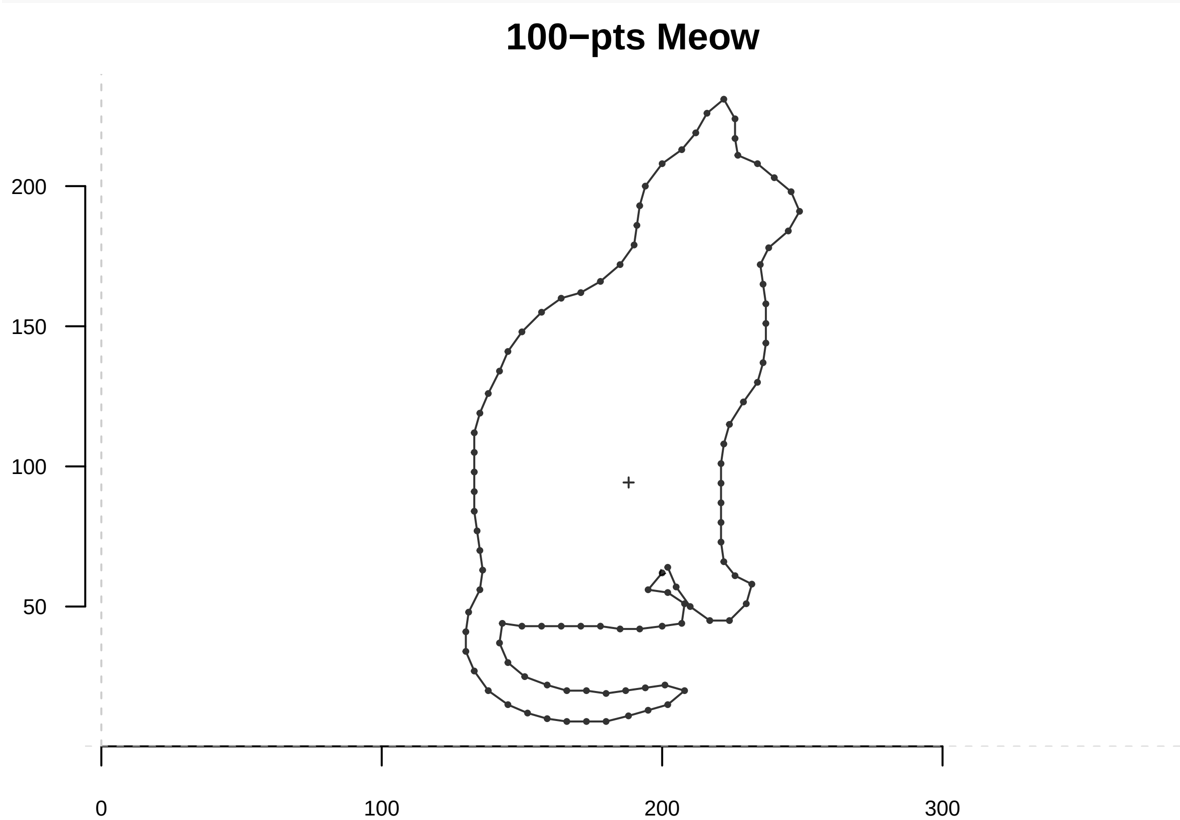

```r
## Play with plot attributes
coo_plot(cat, col="pink", main="Meow")
```



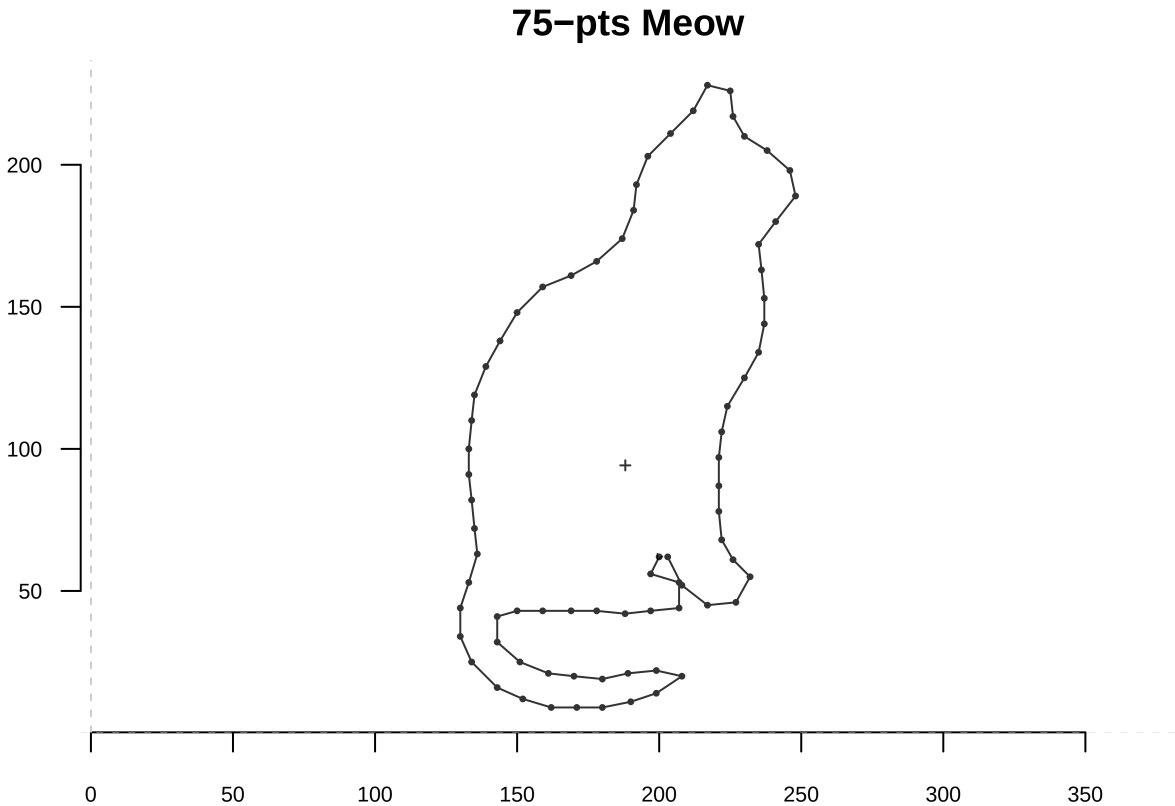

```
## More attributes
?coo_plot

## Meeeeeow
coo_plot(coo_sample(cat, 125), points=TRUE, pch=20, main="125-pts Meow")
```



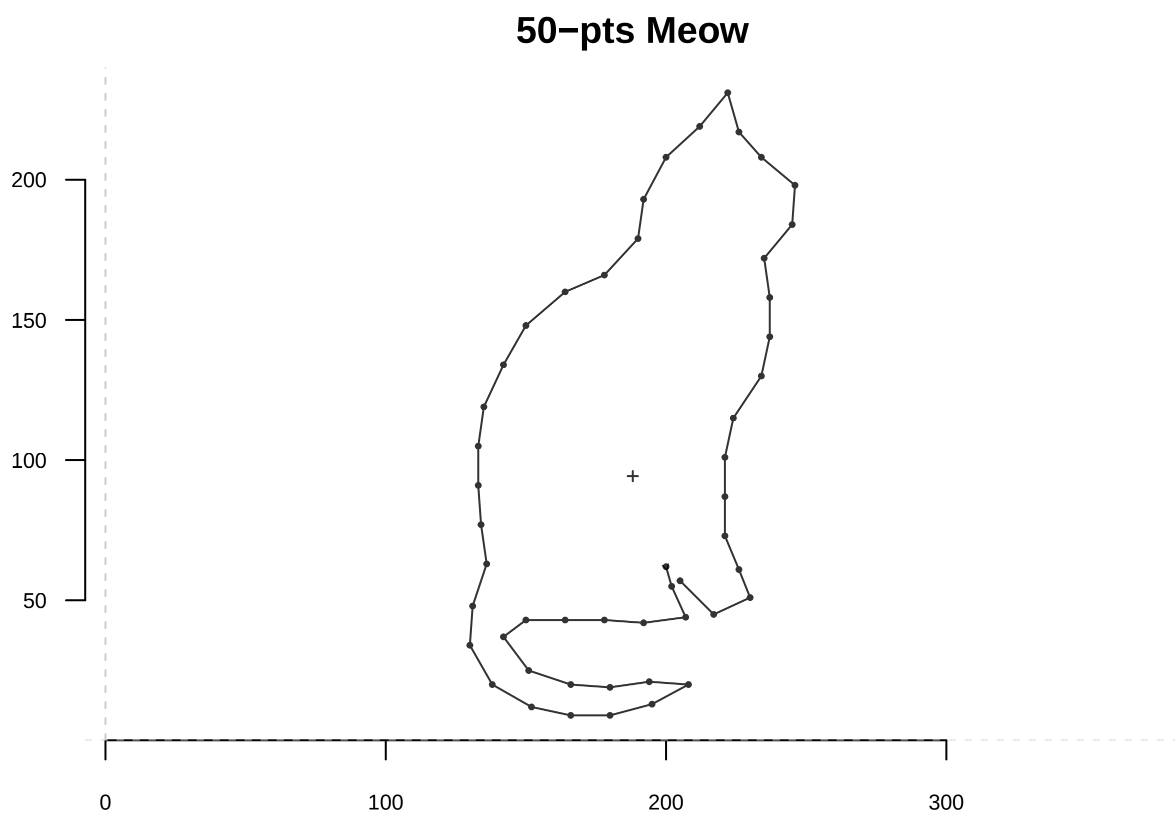
```
coo_plot(coo_sample(cat, 100), points=TRUE, pch=20, main="100-pts Meow")
```



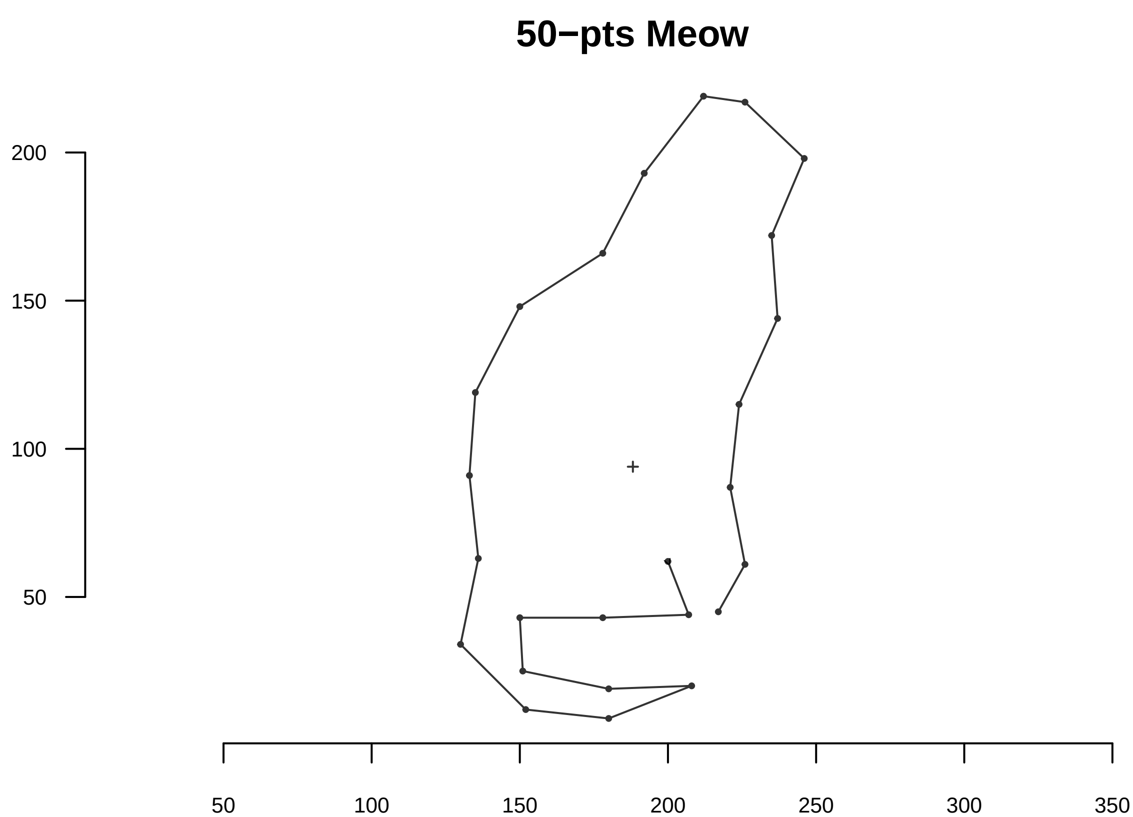
```r
coo_plot(coo_sample(cat, 75), points=TRUE, pch=20, main="75-pts Meow")
```



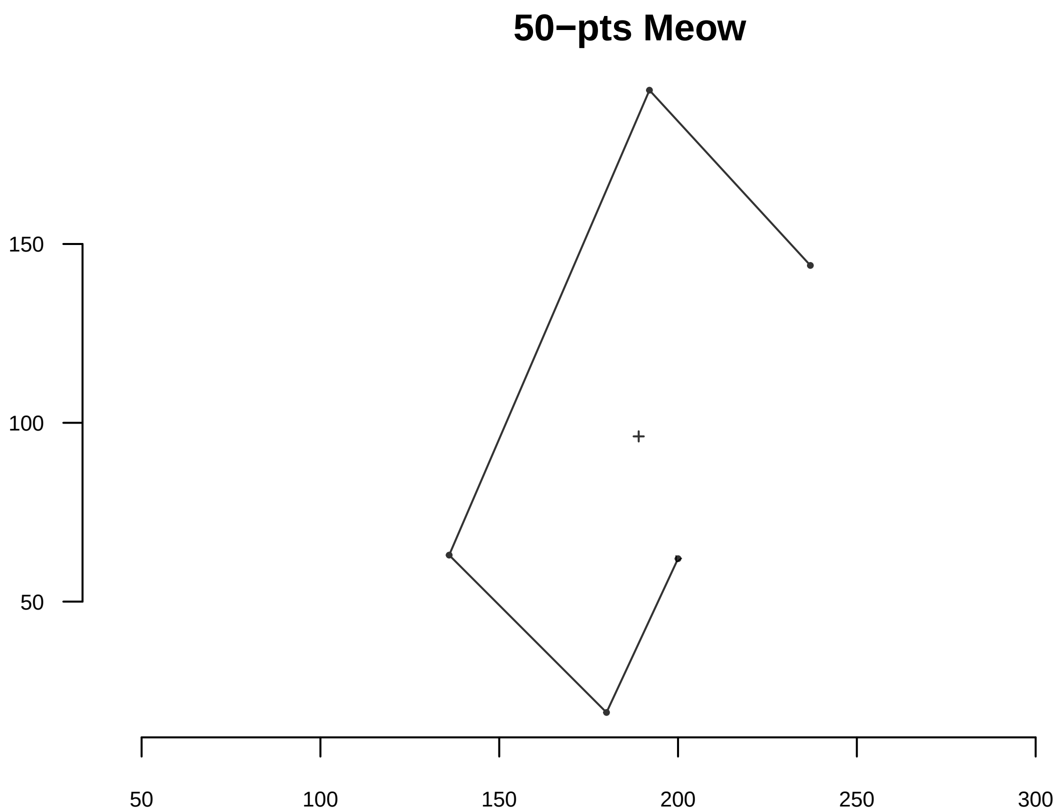

```r
coo_plot(coo_sample(cat, 50), points=TRUE, pch=20, main="50-pts Meow")
```




```
coo_plot(coo_sample(cat, 25), points=TRUE, pch=20, main="50-pts Meow")
```



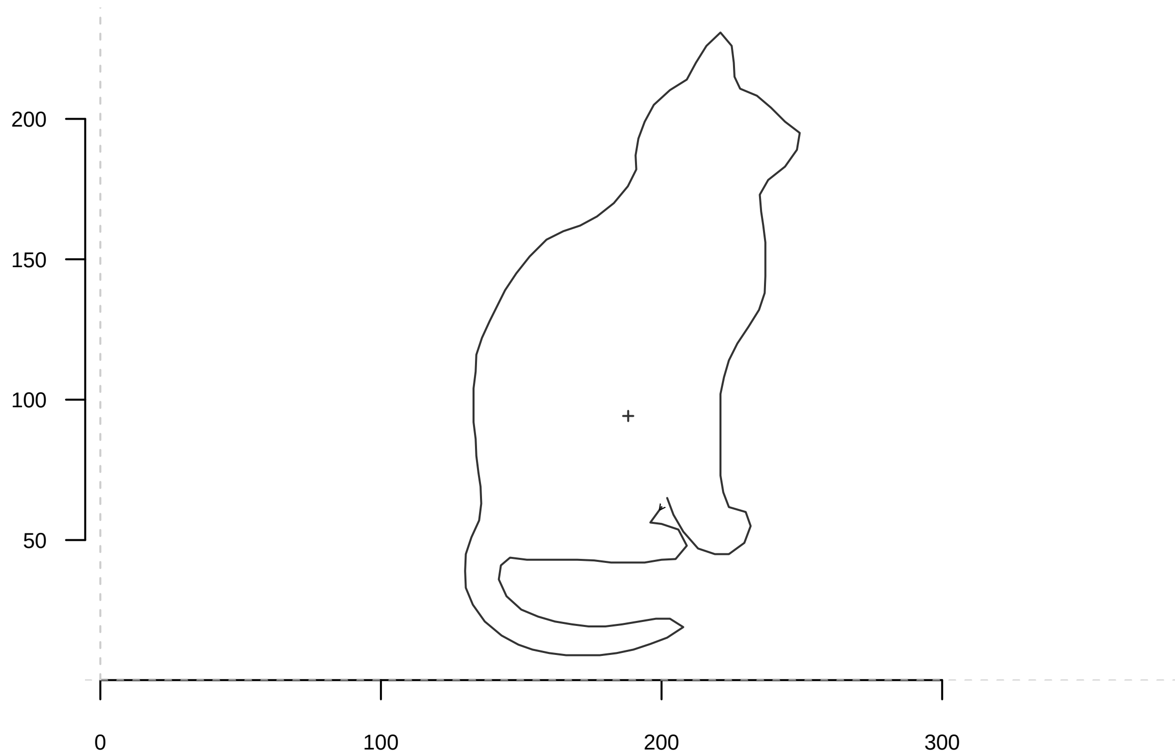
```r
coo_plot(coo_sample(cat, 5), points=TRUE, pch=20, main="50-pts Meow")
```



More Functions useful for visualizing

`coo_smooth()`: Smoothes coordinates using a simple moving average. May be useful to remove digitization noise, mainly on outlines and open outlines.

```r
cat %>% coo_smooth(1) %>% coo_sample(120) %>% coo_plot()
```

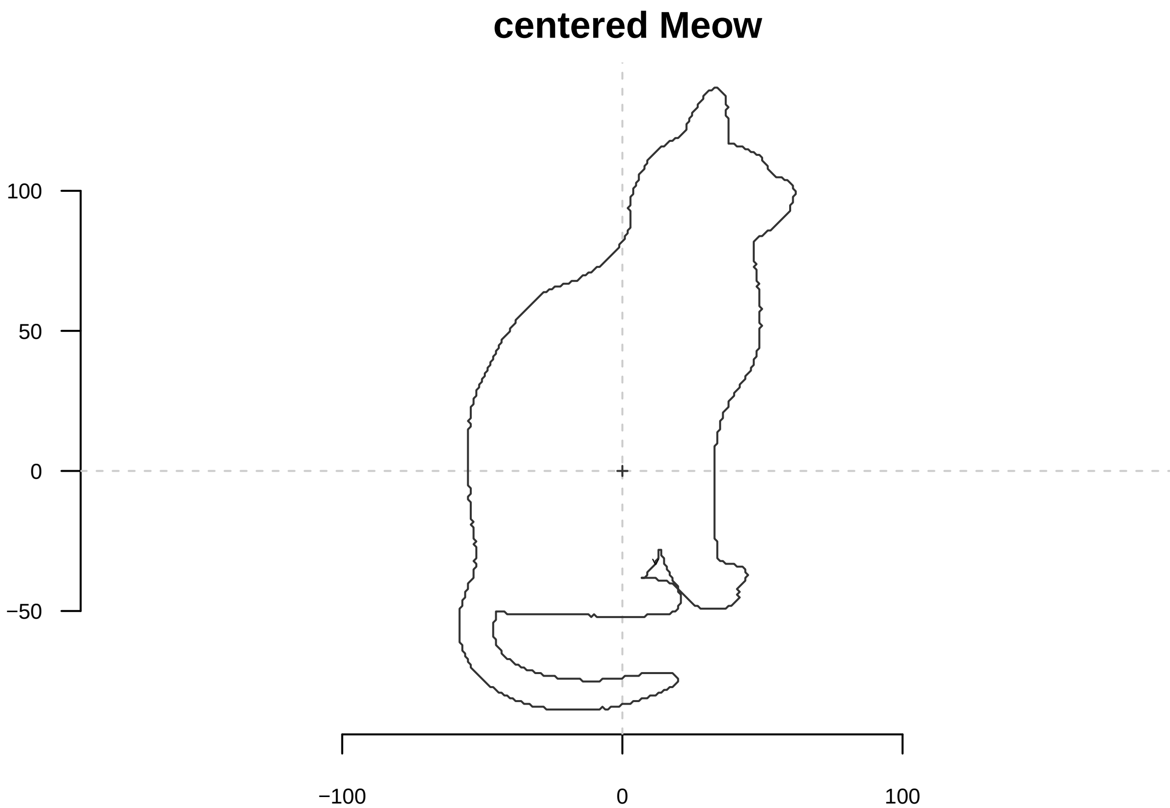


Morphometrics: Comparing across samples

Morphometrics: refers to the quantitative analysis of form, a concept that encompasses size and shape. Morphometric analyses are commonly performed on organisms, and are useful in analyzing their fossil record, the impact of mutations on shape, developmental changes in form, covariances between ecological factors and shape, as well for estimating quantitative-genetic parameters of shape ref wikipedia.

In morphometric analysis it is important to find the center so that you can compare across samples. This is important so that the program knows how to align.

```
coo_plot(coo_center(cat), main="centered Meow")
```



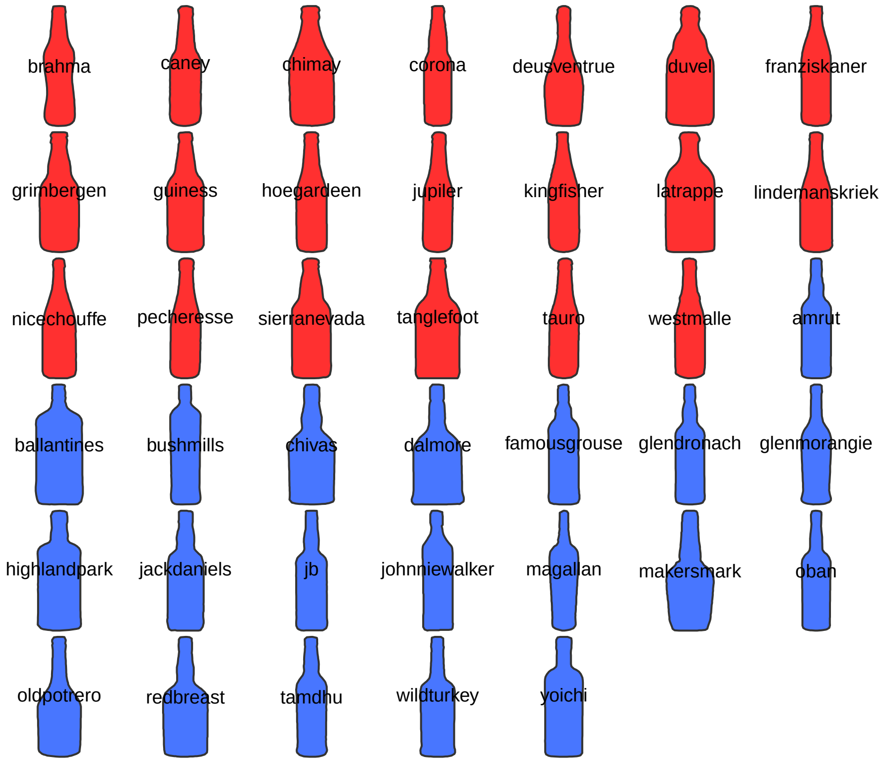
The exciting aspect of momocs is that you can analyze and compare a bunch of shapes!

```
## let's start using another dataset - bot
bot
```

```
## Out (outlines)
##   - 40 outlines, 162 +/- 21 coords (in $coo)
##   - 2 classifiers (in $fac):
## # A tibble: 40 x 2
##   type   fake
##   <fct>  <fct>
## 1 whisky a
## 2 whisky a
## 3 whisky a
## 4 whisky a
## 5 whisky a
## 6 whisky a
## # ... with 34 more rows
##   - also: $ldk
```

The bot dataset is composed of 40 bottles of whiskey and beer.

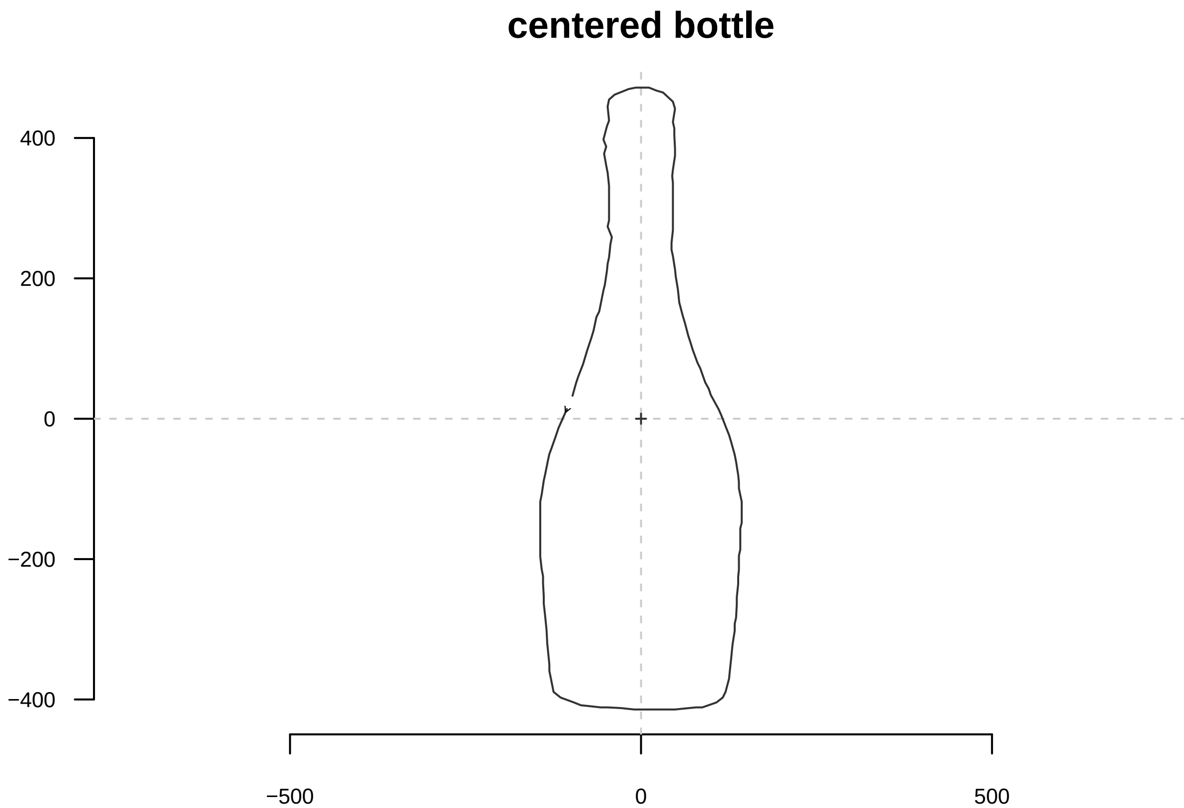
```
## Let's check them all out
panel(bot, fac="type", names=TRUE)
```




```
## Individual bottle
head(bot[2])
```

```
##      [,1] [,2]
## [1,]   53  535
## [2,]   53  525
## [3,]   54  505
## [4,]   53  495
## [5,]   54  485
## [6,]   54  464
```

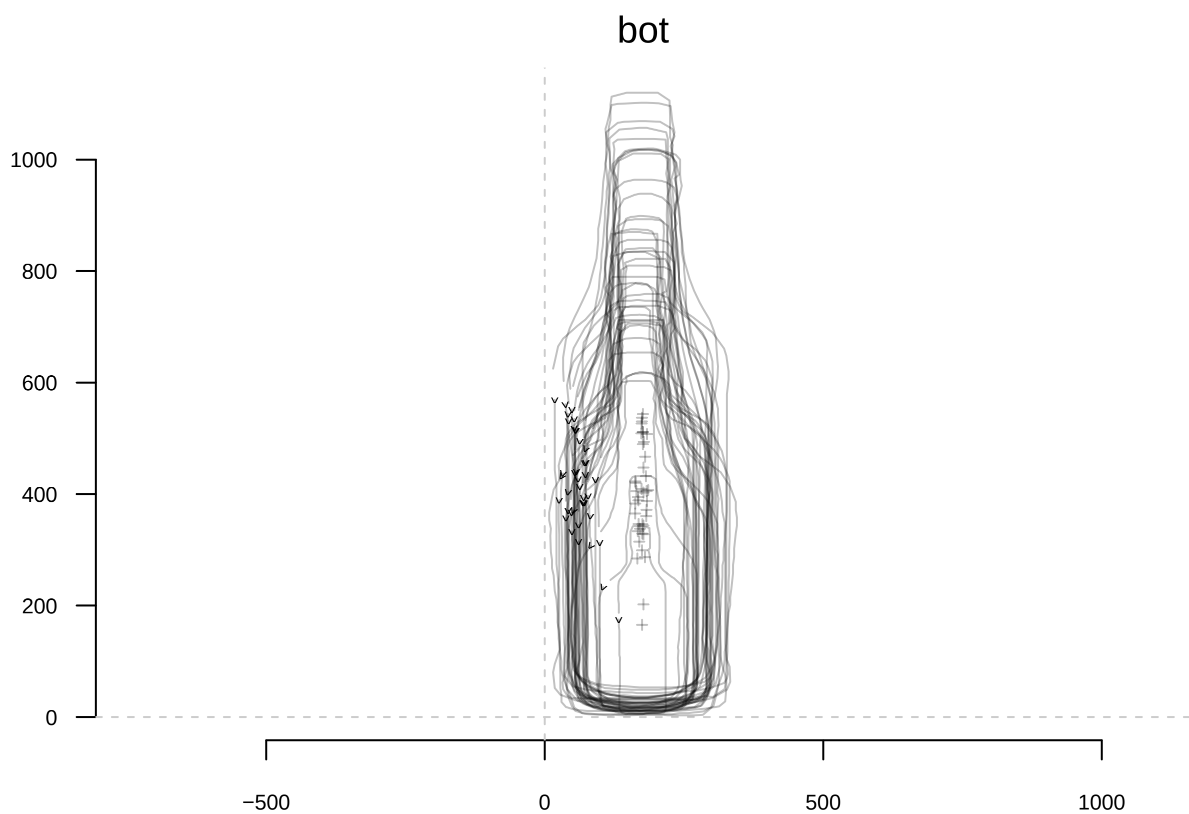
```
## - [ ] try other bottles.
coo_plot(coo_center(bot[5]), main="centered bottle")
```



Now let's look at all of them.

```
## Stack
stack(bot)
```

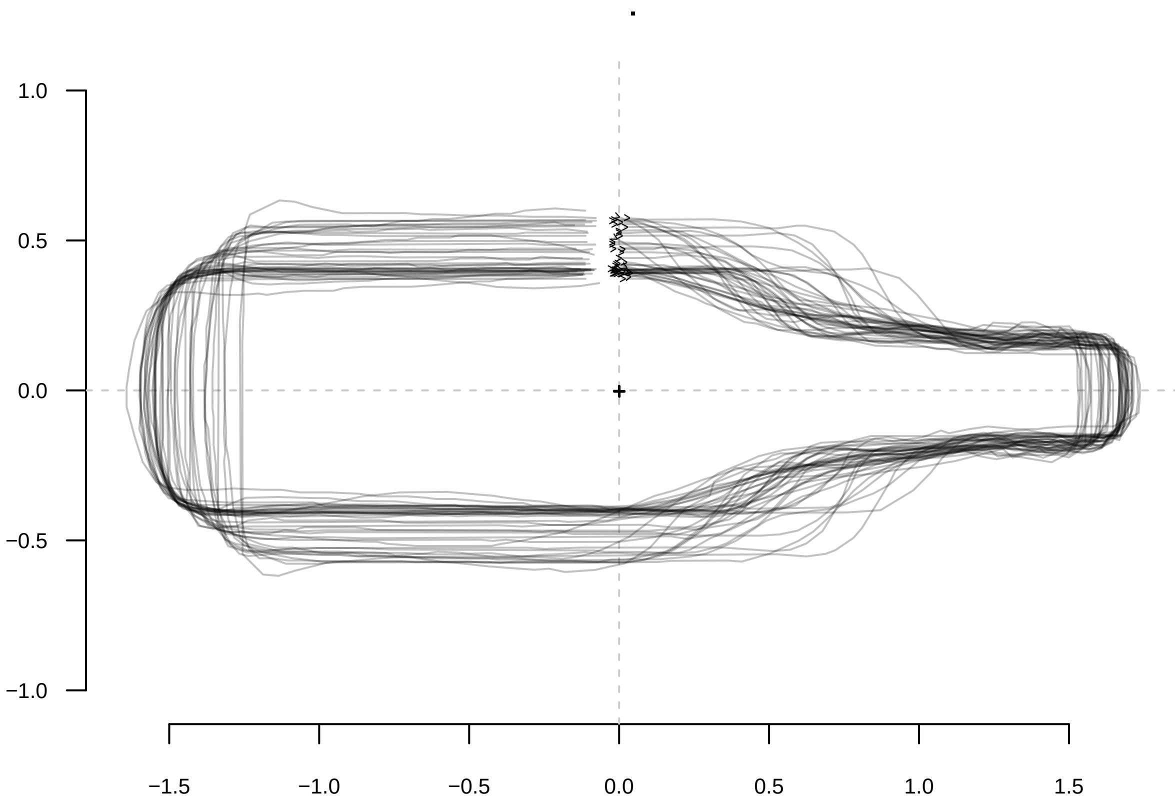
```
## will soon be deprecated, see ?pile
```




```
## Stack and visualize
## - [ ] Take away other functions to see what they do.

bot %>%
  coo_center() %>%
  coo_alignxax() %>%
  coo_scale %>%
  coo_slidedirection("up") %T>%
  print() %>%
  stack()
```

```
## Out (outlines)
##   - 40 outlines, 162 +/- 21 coords (in $coo)
##   - 2 classifiers (in $fac):
## # A tibble: 40 x 2
##   type   fake
##   <fct>  <fct>
## 1 whisky a
## 2 whisky a
## 3 whisky a
## 4 whisky a
## 5 whisky a
## 6 whisky a
## # ... with 34 more rows
##   - also: $ldk
```

```
## will soon be deprecated, see ?pile
```



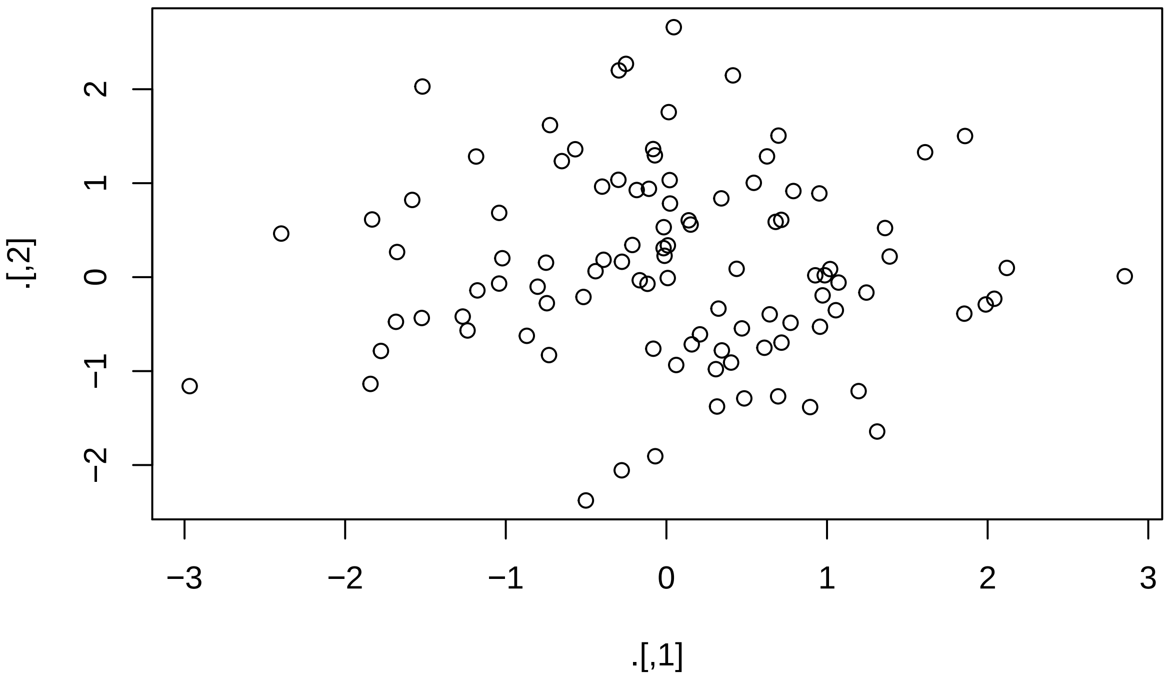
Break for Discussion on Pipes

- [ ] How many of you all use pipes in R?
- [ ] What are the benefits of using pipes?
- [ ] What make pipes hard to use?
- [ ] Who uses the more complex pipes `%T>%`?

Note: We used the T pipe `%T>%`.

The "tee" operator, %T>% works like %>%, except it returns the left-hand side value, and not the result of the right-hand side operation. This is useful when a step in a pipeline is used for its side-effect (printing, plotting, logging, etc.).

```
## Tee operator example
rnorm(200) %>%
matrix(ncol = 2) %T>%
plot %>% # plot usually does not return anything.
colSums
```



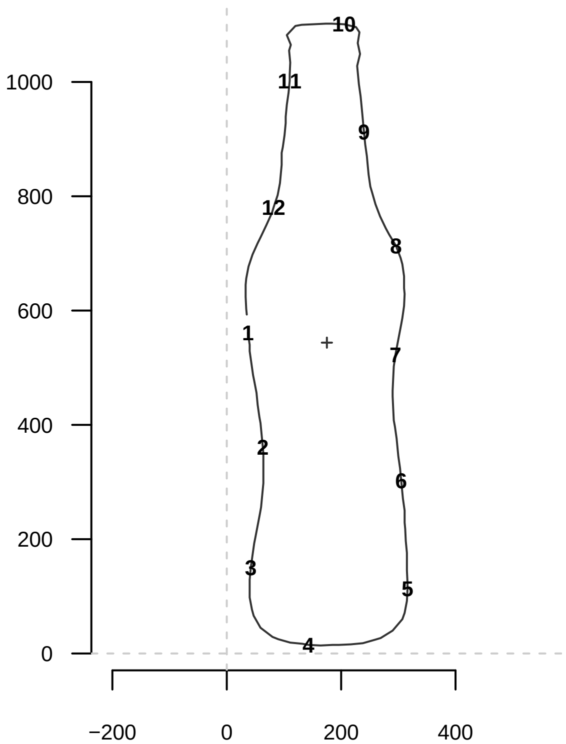

```
## [1]  4.438353 11.984870
```

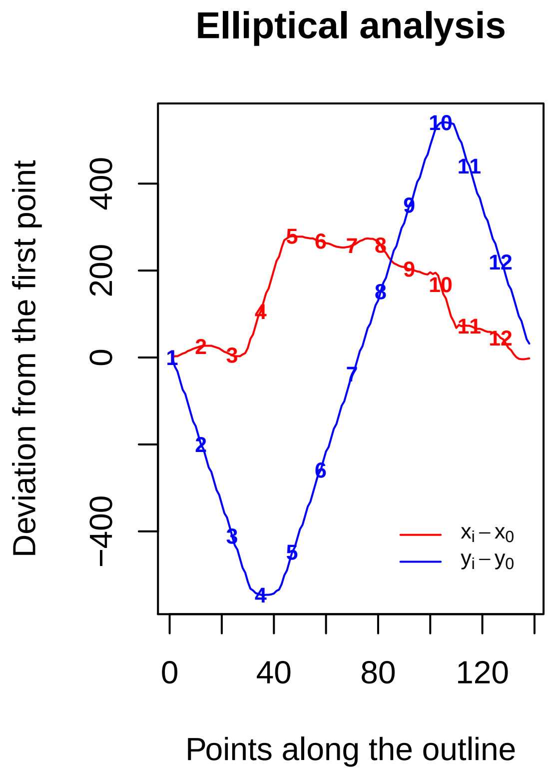
Elliptical fourier analysis

- [ ] Head back to power point for a sec.

Now that we have a very basic understanding of Elliptical Fourier analysis, let's get our hands dirty exploring the datasets.

```
coo_oscillo(bot[1], "efourier")
```






```
## # A tibble: 138 x 2
##        dx    dy
##     <dbl> <dbl>
##  1     0.    0.
##  2     3.  -21.
##  3     3.  -32.
##  4     6.  -53.
##  5     9.  -74.
##  6    11.  -84.
##  7    15. -105.
##  8    17. -126.
##  9    20. -147.
## 10    22. -158.
## # ... with 128 more rows
```

Remember:

A dataset (group of shapes) in Momocs is described as a Coo. Once you apply a method to that you get a Coe.

Coo + Morphometric method = Coe (x; y) coordinates + appropriate method = quantitative variables

```
## actual efourier transformation
bot.f <- efourier(bot)
```

```
## you selected `norm=TRUE`, which is not recommended. See ?efourier
```

```
## 99%
##  10
```

```
## 'nb.h' not provided and set to 10 (99% harmonic power)
```

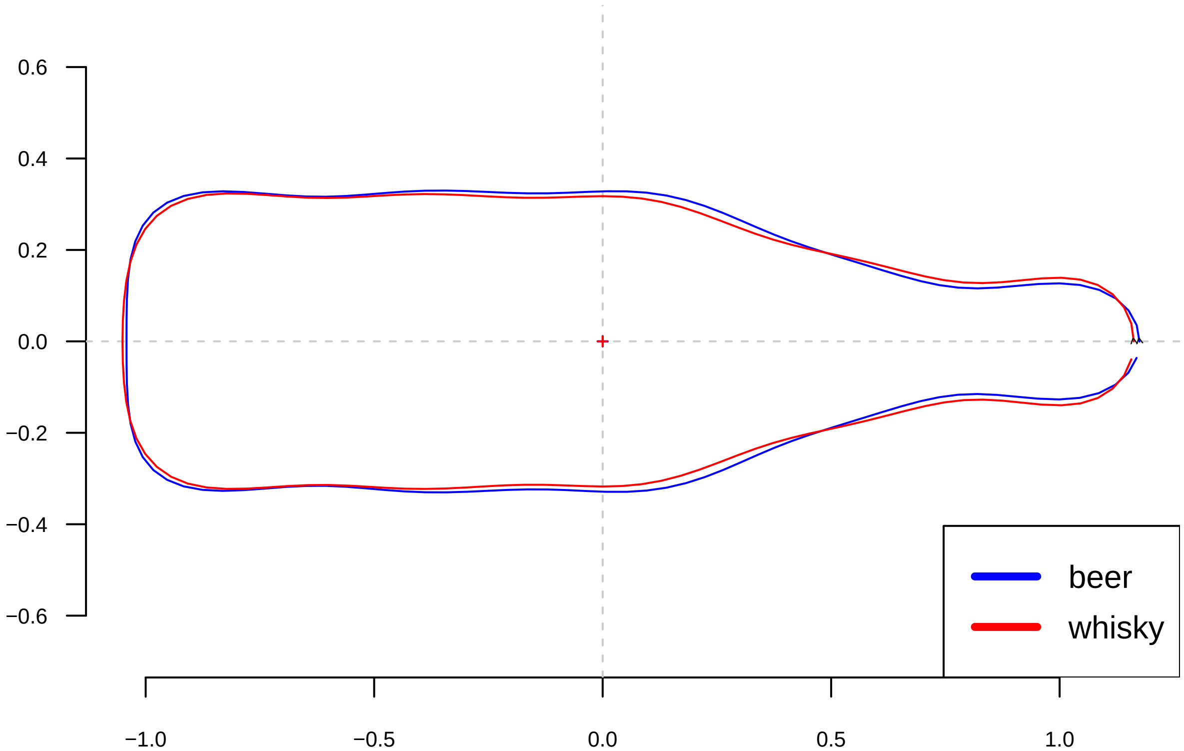
```r
## Now we have a Coe object
bot.f
```

```
## An OutCoe object [ elliptical Fourier analysis ]
## --------------------
##  - $coe: 40 outlines described, 10 harmonics
## # A tibble: 40 x 2
##   type   fake
##   <fct>  <fct>
## 1 whisky a
## 2 whisky a
## 3 whisky a
## 4 whisky a
## 5 whisky a
## 6 whisky a
## # ... with 34 more rows
```

```r
class(bot.f)
```

```
## [1] "OutCoe" "Coe"
```

```r
# mean shape, per group
bot.ms <- mshapes(bot.f, 1)
beer   <- bot.ms$shp$beer    %T>% coo_plot(border="blue")
whisky <- bot.ms$shp$whisky  %T>% coo_draw(border="red")
legend("bottomright", lwd=4,
       col=c("blue", "red"), legend=c("beer", "whisky"))
```



PCA

Break for Discussion on PCA

Principle Component Analysis (PCA): A way to break down the variance between samples. Its operation can be thought of as revealing the internal structure of the data in a way that best explains the variance in the data.

- [ ] How many people use PCA regularly?
- [ ] What kinds of questions can you ask with PCA?
- [ ] How is PCA different from clustering?
- We will go into a bit more detail on how PCA and clustering are related in next half of tutorial.

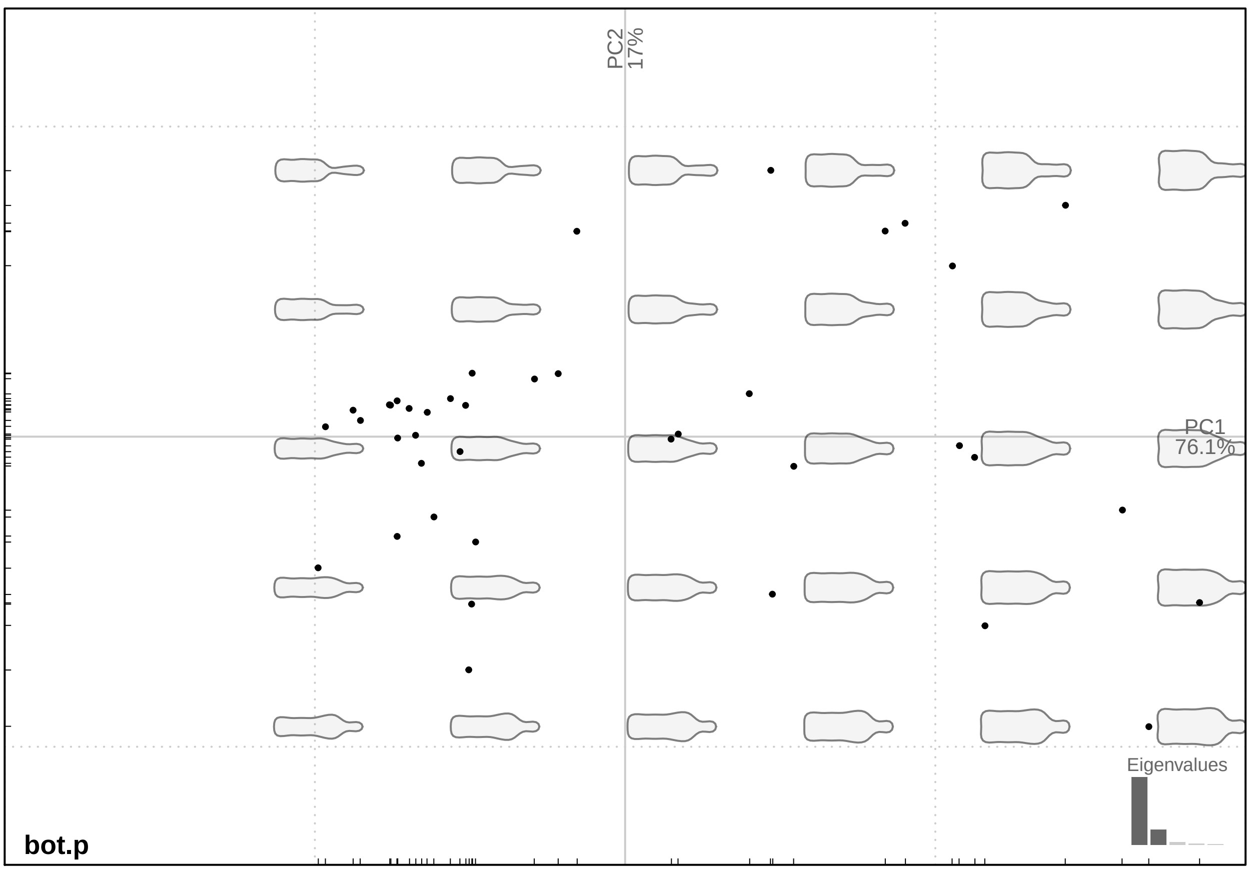
Now back to Momocs...

Let's perform PCA on our Elliptical Fourier bottles

```
bot.p <- PCA(bot.f)
class(bot.p)         # a PCA object, let's plot it
```

```
## [1] "PCA"    "prcomp"
```

```
plot(bot.p)
```

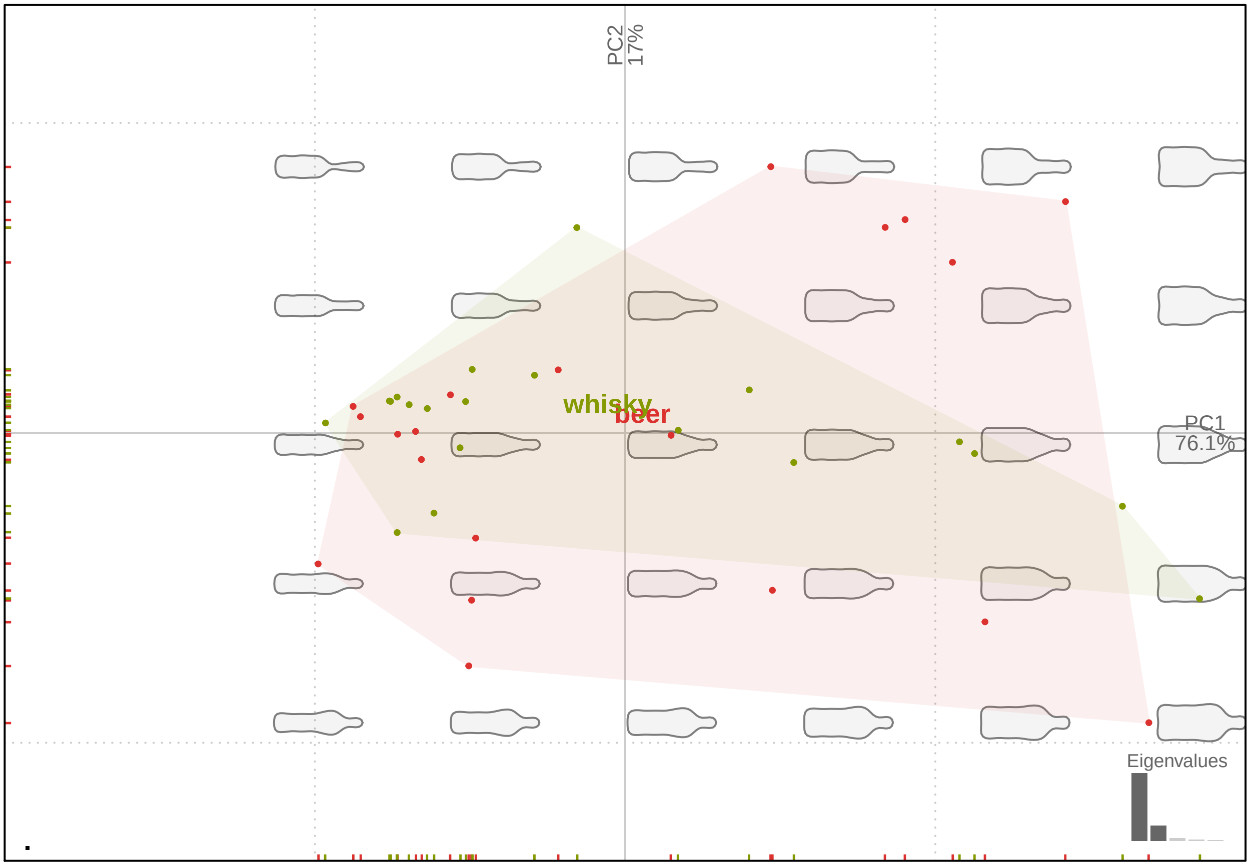



```
## Visualize
bot %>% efourier(norm=TRUE) %>% PCA() %>% plot("type")
```

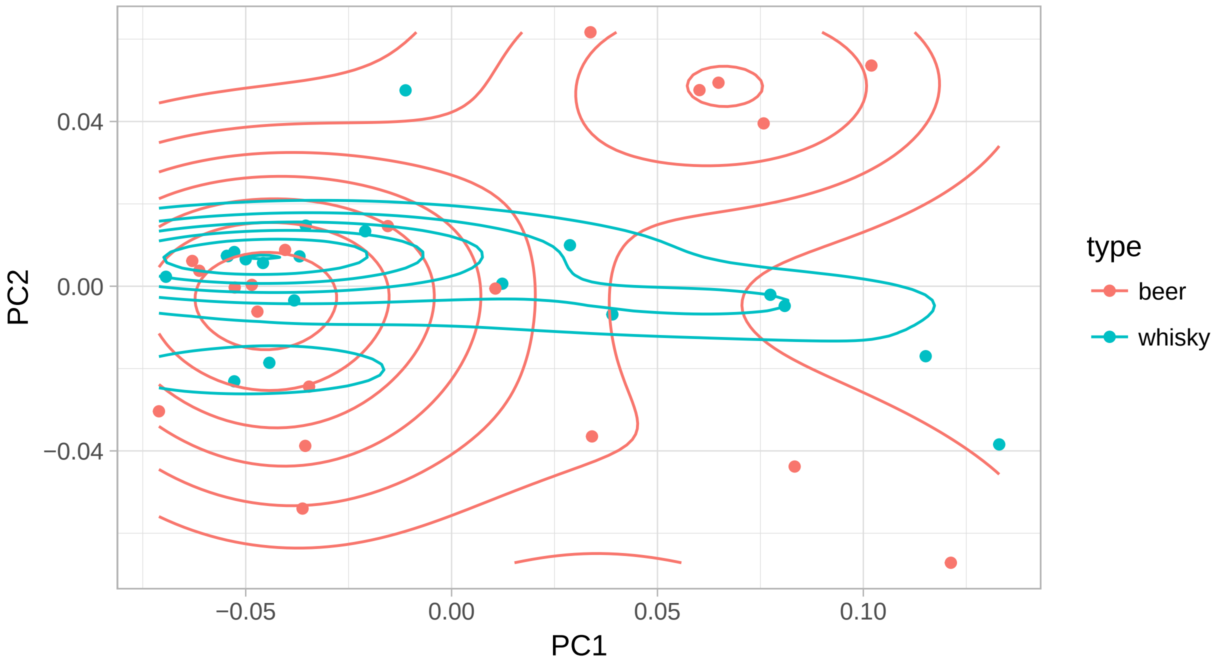
```
## you selected `norm=TRUE`, which is not recommended. See ?efourier
```

```
## 99%
##  10
```

```
## 'nb.h' not provided and set to 10 (99% harmonic power)
```



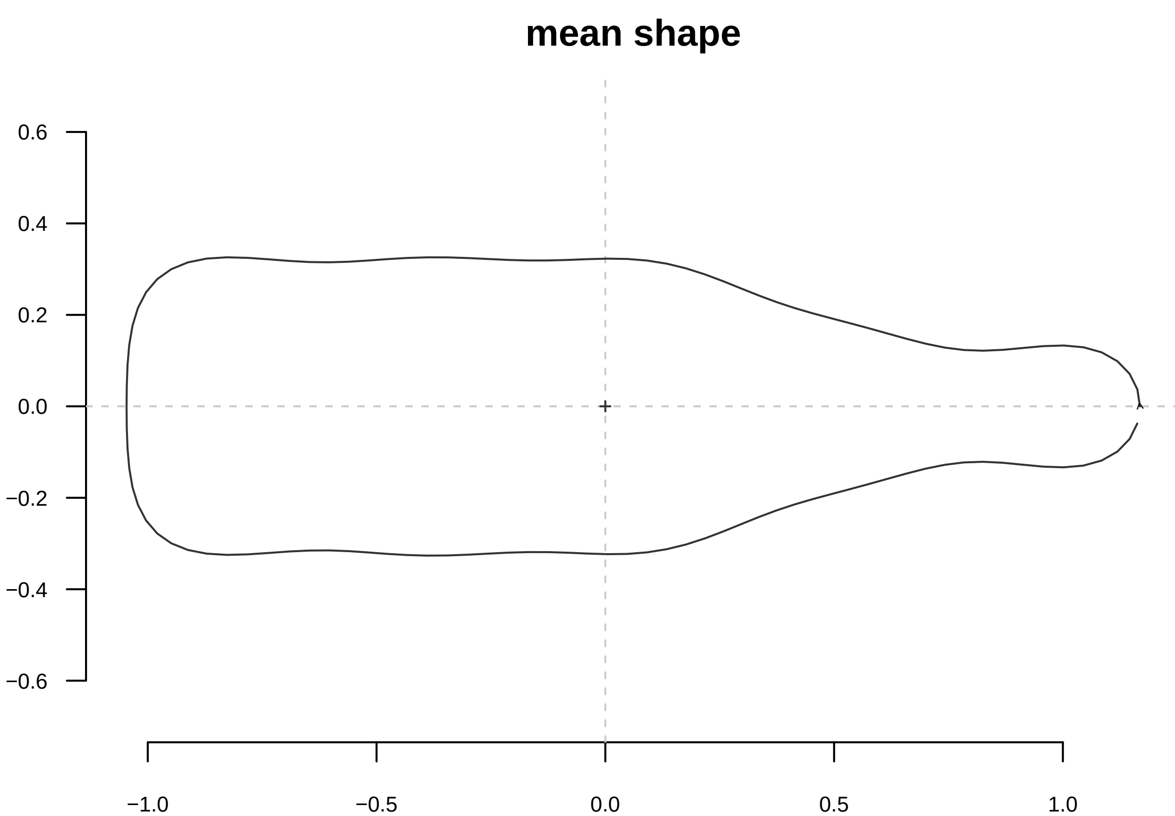

```
bot.p %>% as_df() %>% ggplot() +
  aes(x=PC1, y=PC2, col=type) + coord_equal() +
  geom_point() + geom_density2d() + theme_light()
```



You can also get the mean and standard deviation of shapes

```
# mean shape
bot.f %>% mshapes() %>% coo_plot(main = "mean shape")
```

```
## no 'fac' provided, returns meanshape
```

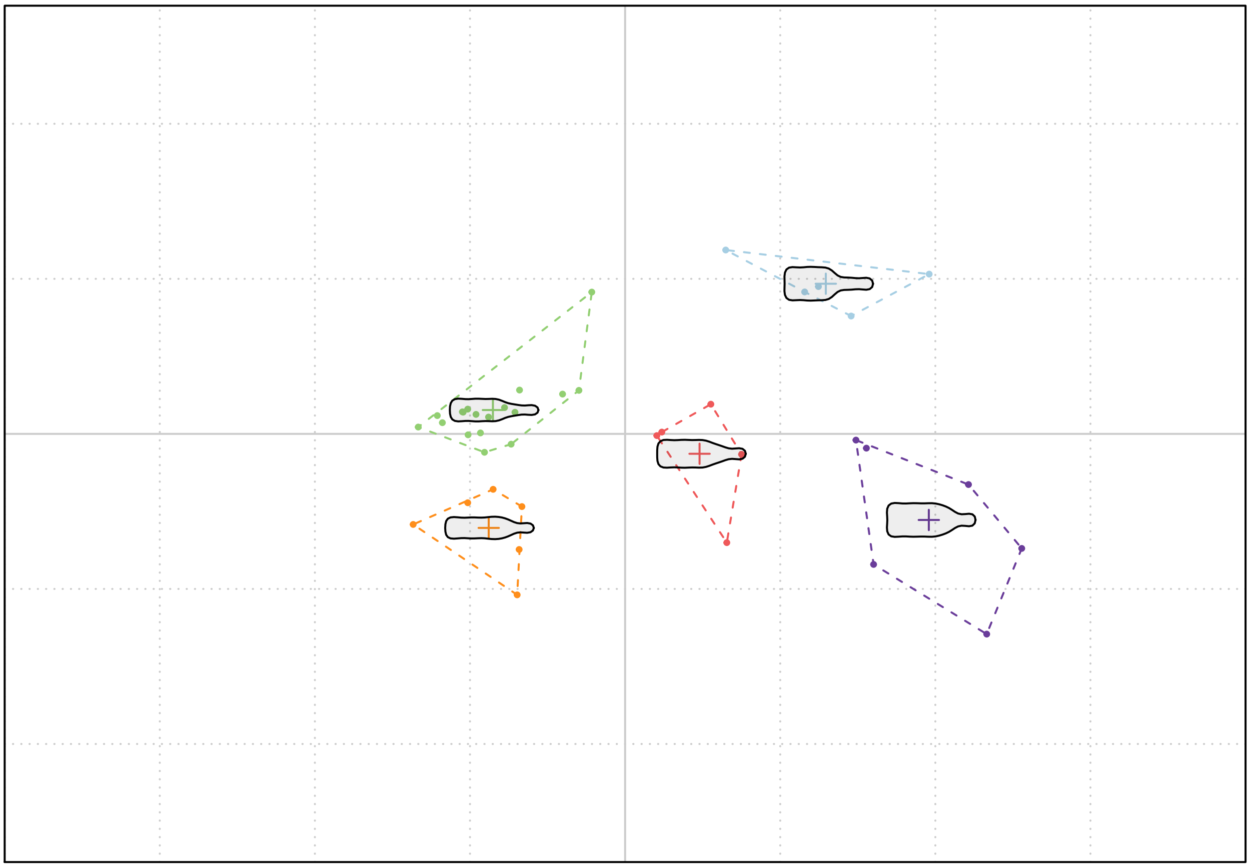


You can even get the standard deviation of shapes, but you have to use another package, so I won't go into too much detail here.

Kmeans clustering on shapes

What sort of questions do we ask of these objects?

```
## This is k-means implements in Momocs
KMEANS(bot.p, centers = 5)
```



```
## K-means clustering with 5 clusters of sizes 5, 18, 5, 6, 6
##
## Cluster means:
##           PC1          PC2
## 1  0.06730453  0.050371247
## 2 -0.04433327  0.008000365
## 3  0.02497106 -0.006671549
## 4 -0.04573875 -0.031541607
## 5  0.10184224 -0.028875902
##
## Clustering vector:
##        brahma          caney         chimay         corona    deusventrue
##             4              2              5              2              2
##         duvel   franziskaner     grimbergen        guiness     hoegardeen
##             5              4              3              3              2
##       jupiler     kingfisher       latrappe lindemanskriek    nicechouffe
##             2              2              5              2              2
```

```
##    pecheresse   sierranevada     tanglefoot          tauro      westmalle
##             2              3              5              2              2
##         amrut    ballantines      bushmills         chivas        dalmore
##             2              5              4              1              1
##   famousgrouse    glendronach   glenmorangie   highlandpark    jackdaniels
##             4              2              2              5              3
##            jb   johnniewalker      magallan     makersmark           oban
##             2              4              4              1              2
##    oldpotrero      redbreast         tamdhu     wildturkey         yoichi
##             1              1              2              2              3
##
## Within cluster sum of squares by cluster:
## [1] 0.002720829 0.006537571 0.001916822 0.001860776 0.006121728
##  (between_SS / total_SS =  89.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

There is a lot of other things you can do with this package:

```
apropos("coo_")
```

```
##  [1] "coo_align"                    "coo_aligncalliper"
##  [3] "coo_alignminradius"           "coo_alignxax"
##  [5] "coo_angle_edge1"              "coo_angle_edges"
##  [7] "coo_angle_tangent"            "coo_area"
##  [9] "coo_arrows"                   "coo_baseline"
## [11] "coo_bookstein"                "coo_boundingbox"
## [13] "coo_calliper"                 "coo_centdist"
## [15] "coo_center"                   "coo_centpos"
## [17] "coo_centre"                   "coo_centsize"
## [19] "coo_check"                    "coo_chull"
## [21] "coo_chull_onion"              "coo_circularity"
## [23] "coo_circularityharalick"      "coo_circularitynorm"
## [25] "coo_close"                    "coo_convexity"
## [27] "coo_diffrange"                "coo_down"
## [29] "coo_draw"                     "coo_draw_rads"
## [31] "coo_dxy"                      "coo_eccentricityboundingbox"
## [33] "coo_eccentricityeigen"        "coo_elongation"
## [35] "coo_extract"                  "coo_flipx"
## [37] "coo_flipy"                    "coo_force2close"
## [39] "coo_interpolate"              "coo_intersect_angle"
## [41] "coo_intersect_direction"      "coo_intersect_segment"
## [43] "coo_is_closed"                "coo_jitter"
## [45] "coo_ldk"                      "coo_left"
## [47] "coo_length"                   "coo_likely_anticlockwise"
## [49] "coo_likely_clockwise"         "coo_listpanel"
## [51] "coo_lolli"                    "coo_lw"
## [53] "coo_nb"                       "coo_oscillo"
## [55] "coo_perim"                    "coo_perimcum"
## [57] "coo_perimpts"                 "coo_plot"
## [59] "coo_range"                    "coo_range_enlarge"
```

```
## [61] "coo_rectangularity"          "coo_rectilinearity"
## [63] "coo_rev"                     "coo_right"
## [65] "coo_rotate"                  "coo_rotatecenter"
## [67] "coo_ruban"                   "coo_sample"
## [69] "coo_sample_prop"             "coo_samplerr"
## [71] "coo_scale"                   "coo_scalex"
## [73] "coo_scaley"                  "coo_shearx"
## [75] "coo_sheary"                  "coo_slice"
## [77] "coo_slide"                   "coo_slidedirection"
## [79] "coo_slidegap"                "coo_smooth"
## [81] "coo_smoothcurve"             "coo_solidity"
## [83] "coo_tangle"                  "coo_template"
## [85] "coo_template_relatively"     "coo_theta3"
## [87] "coo_thetapts"                "coo_trans"
## [89] "coo_trim"                    "coo_trimbottom"
## [91] "coo_trimtop"                 "coo_truss"
## [93] "coo_unclose"                 "coo_up"
## [95] "coo_width"
```

Over 95 different functions acting on Coo objects! I hope that you explore more on your own time and find a nice data set to explore. This tutorial was largely based off this tutorial, which goes into a lot more detail about the package. It also explain how to input your own data! Basically what you need is binary jpeg images, then they can be imported. So keep an eye out for cool datasets where you can employ this package! I am working on illustrations of butterflies and cattipilars at the moment!

References:

Other package that allows more explicit view of shapes

shapes