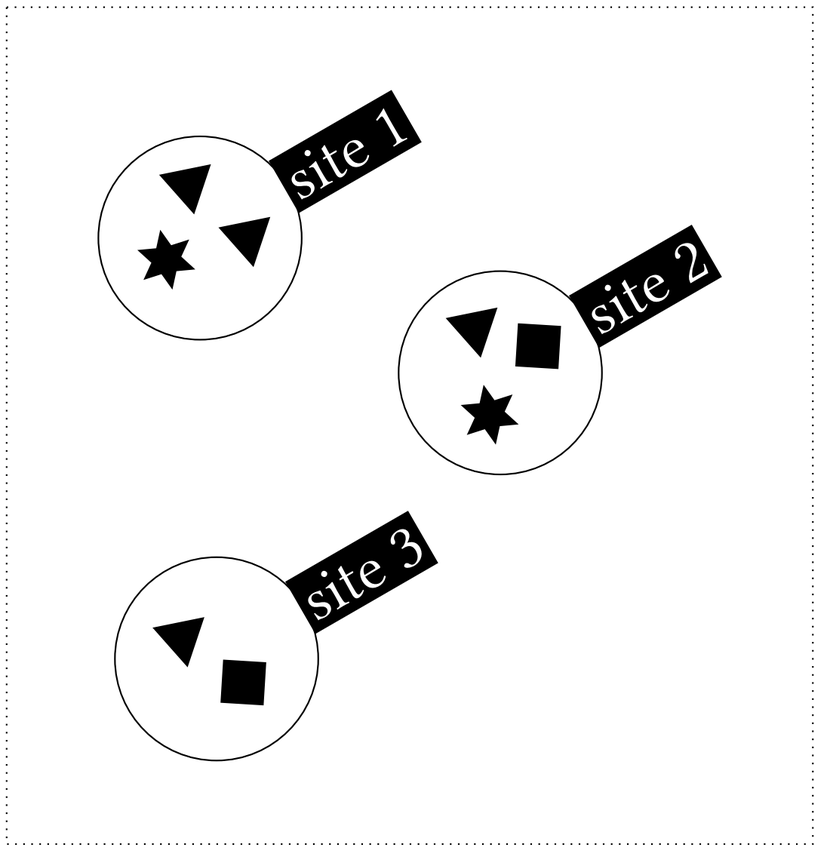
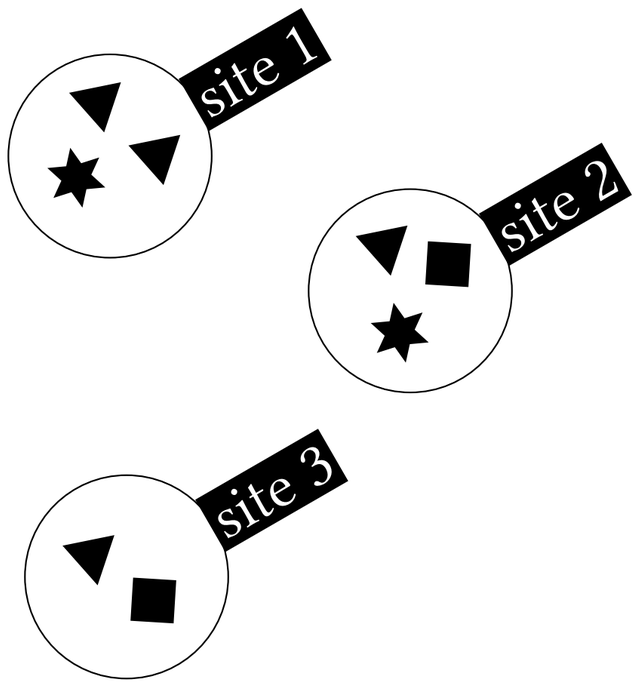
plyr

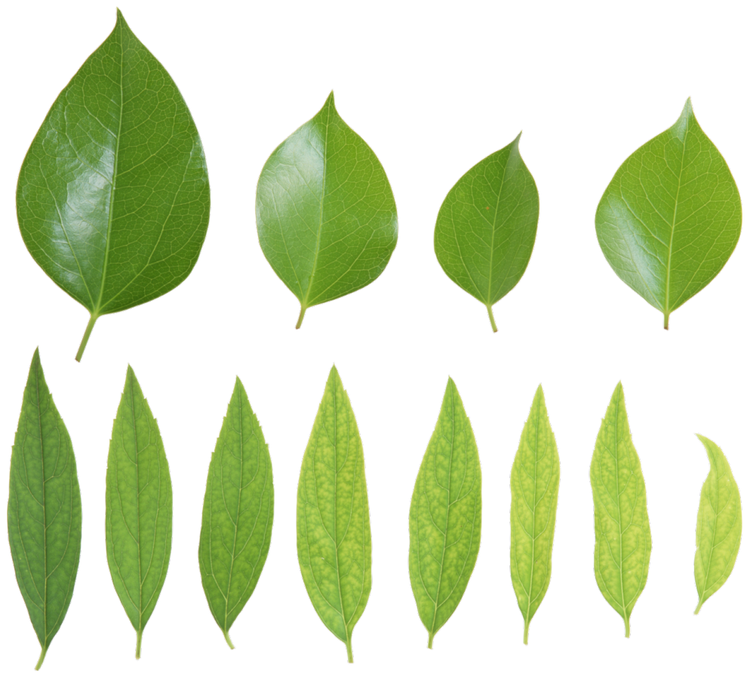
the split-apply-combine strategy

the split-apply-combine strategy ?

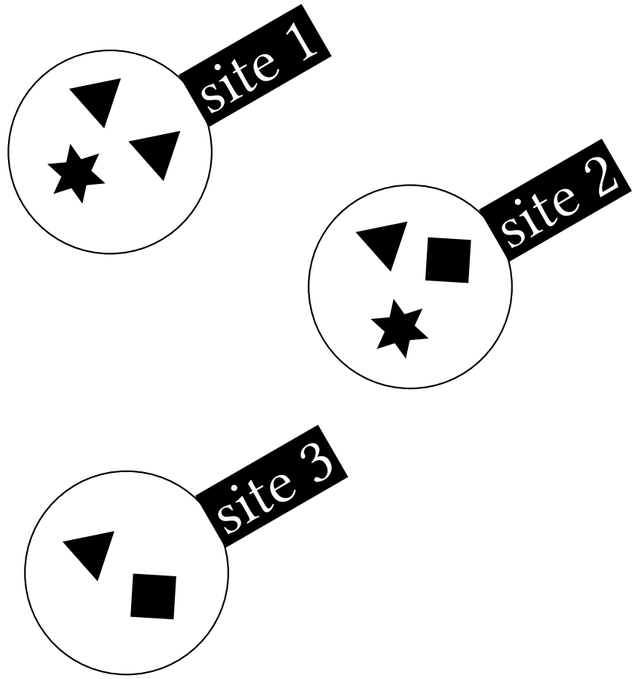


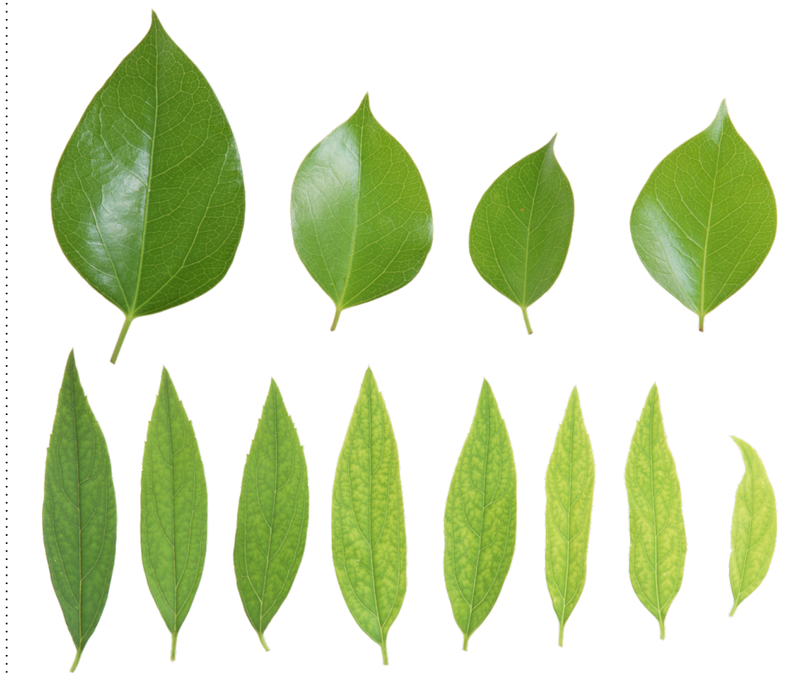
the split-apply-combine strategy ?

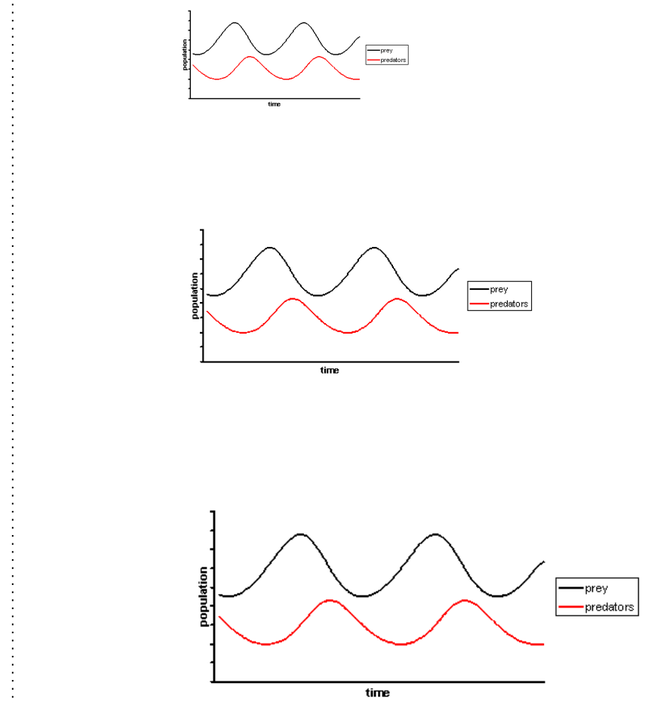




the split-apply-combine strategy ?







the split-apply-combine strategy ?

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

a **data.frame** example: compute the mean Sepal.Length (*iris* dataset)

```
   Species Sepal.Width Sepal.Length
    setosa         3.1          4.9
    setosa         3.7          5.4
versicolor         2.3          5.0
versicolor         2.7          5.6
 virginica         3.0          7.2
 virginica         2.8          7.4
```

a **data.frame** example: compute the mean Sepal.Length (*iris* dataset)

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

split

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4

Species	Sepal.Width	Sepal.Length
versicolor	2.3	5.0
versicolor	2.7	5.6

Species	Sepal.Width	Sepal.Length
virginica	3.0	7.2
virginica	2.8	7.4

a **data.frame** example: compute the mean Sepal.Length (*iris* dataset)

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

split

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4

Species	Sepal.Width	Sepal.Length
versicolor	2.3	5.0
versicolor	2.7	5.6

Species	Sepal.Width	Sepal.Length
virginica	3.0	7.2
virginica	2.8	7.4

apply

`mean(Sepal.Length)`

5.006

5.936

6.588

a **data.frame** example: compute the mean Sepal.Length (*iris* dataset)

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

split

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4

Species	Sepal.Width	Sepal.Length
versicolor	2.3	5.0
versicolor	2.7	5.6

Species	Sepal.Width	Sepal.Length
virginica	3.0	7.2
virginica	2.8	7.4

apply

`mean(Sepal.Length)`

5.006

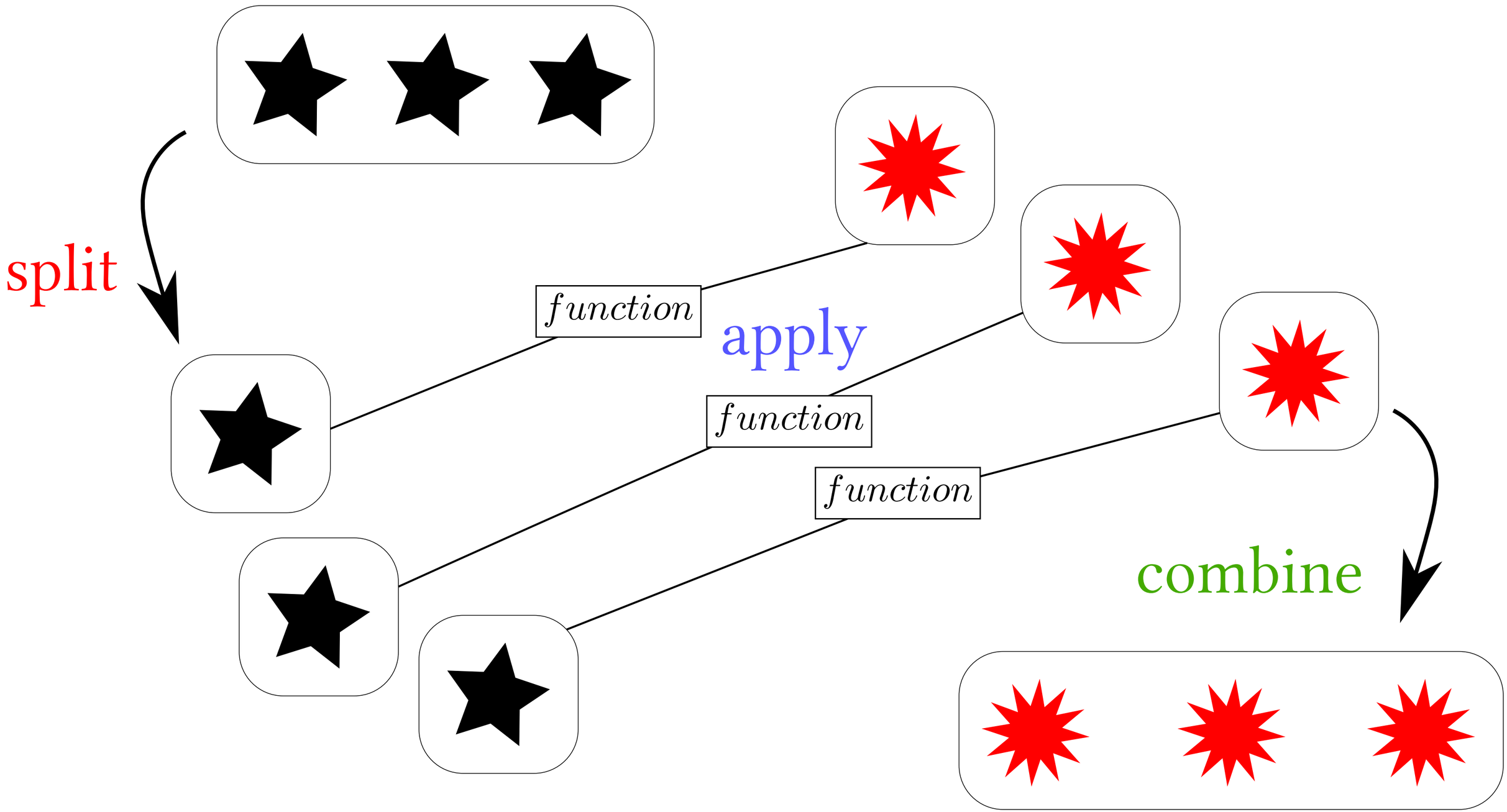
5.936

6.588

combine

Species	Length.mean
setosa	5.006
versicolor	5.936
virginica	6.588

a **collection** of things



a **collection** of things

all rows of a **matrix**

all columns of a **data.frame**

all elements of a **list**

all combinations of factors in a **data.frame**

a **collection** of things

all rows of a **matrix**	apply()
all columns of a **data.frame**	lapply() / apply()
all elements of a **list**	lapply()
all combinations of factors in a **data.frame**	aggregate()

a **collection** of things

each row of a **matrix** — apply()

each column of a **data.frame** — lapply() / apply()

each element of a **list** — lapply()

data belonging to each category in a **data.frame** — aggregate()

**ply()

**ply()

ddply()

data.frame

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

data.frame

Species	Length.mean
setosa	5.006
versicolor	5.936
virginica	6.588

**ply()

data.frame

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

ddply()

data.frame

Species	Length.mean
setosa	5.006
versicolor	5.936
virginica	6.588

array (e.g. matrix)

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	6	11	16	21
[2,]	2	7	12	17	22
[3,]	3	8	13	18	23
[4,]	4	9	14	19	24
[5,]	5	10	15	20	25

aaply()

array (e.g. matrix)

	[,1]
[1,]	1
[2,]	2
[3,]	3
[4,]	4
[5,]	5

**ply()

data.frame

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

ddply()

data.frame

Species	Length.mean
setosa	5.006
versicolor	5.936
virginica	6.588

aaply()

array (e.g. matrix)

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	6	11	16	21
[2,]	2	7	12	17	22
[3,]	3	8	13	18	23
[4,]	4	9	14	19	24
[5,]	5	10	15	20	25

array (e.g. matrix)

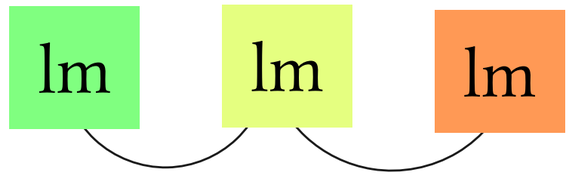
	[,1]
[1,]	1
[2,]	2
[3,]	3
[4,]	4
[5,]	5

dlply()

data.frame

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

list



[1] ddply()

task: get the mean sepal length for each species

```
# Define the function to apply
mean_sep_length <- function(df) {
  mean(df[ ,'Sepal.Length'])
}
```

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

[1] ddply()

task: get the mean sepal length for each species

```
# Define the function to apply
mean_sep_length <- function(df) {
  mean(df[ ,'Sepal.Length'])
}
```

```
> mean_sep_length(iris)
[1] 5.843333
```

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

[1] ddply()

task: get the mean sepal length for each species

```
# Define the function to apply
mean_sep_length <- function(df) {
  mean(df[ ,'Sepal.Length'])
}
```

```
> mean_sep_length(iris)
```

```
[1] 5.843333
```

```
library(plyr)
> ddply(iris, ~ Species, mean_sep_length)
```

Species	V1
1 setosa	5.006
2 versicolor	5.936
3 virginica	6.588

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

a **data.frame** example: compute the mean Sepal.Length (*iris* dataset)

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4
versicolor	2.3	5.0
versicolor	2.7	5.6
virginica	3.0	7.2
virginica	2.8	7.4

split

Species	Sepal.Width	Sepal.Length
setosa	3.1	4.9
setosa	3.7	5.4

Species	Sepal.Width	Sepal.Length
versicolor	2.3	5.0
versicolor	2.7	5.6

Species	Sepal.Width	Sepal.Length
virginica	3.0	7.2
virginica	2.8	7.4

apply

```
mean_sep_length
```

5.006

5.936

6.588

combine

Species	V1
setosa	5.006
versicolor	5.936
virginica	6.588

[1]ddply()

Your turn

task: get the mean **petal** length for each species

[1]ddply()

task: get the mean petal length for each species

```
# Define the function to apply
mean_pet_length <- function(df) {
  mean(df[ ,'Petal.Length'])
}
```

```
> mean_pet_length(iris)
```

```
[1] 3.758
```

```
library(plyr)
> ddply(iris, ~ Species, mean_pet_length)
```

```
     Species    V1
1     setosa 1.462
2 versicolor 4.260
3  virginica 5.552
```

[1]ddply()

task: get the mean petal length for each species

```
# Define the function to apply
mean_pet_length <- function(df) {
  mean(df[ ,'Petal.Length'])
}
```

NO NAME HERE...

```
> mean_pet_length(iris)
```

```
[1] 3.758
```

...MEANS NO NAME THERE

```
library(plyr)
> ddply(iris, ~ Species, mean_pet_length)
```

```
     Species    V1
1     setosa 1.462
2 versicolor 4.260
3  virginica 5.552
```

[1]ddply()

task: get the mean petal length for each species

```
# Define the function to apply
mean_pet_length <- function(df) {
  data.frame(length = mean(df[ ,'Petal.Length']))
}
```

[1] ddply()

task: get the mean petal length for each species

```
# Define the function to apply
mean_pet_length <- function(df) {
  data.frame(length = mean(df[ ,'Petal.Length']))
}
```

```
> mean_pet_length(iris)
```

```
  length
1  3.758
```

[1] ddply()

task: get the mean petal length for each species

```
# Define the function to apply
mean_pet_length <- function(df) {
  data.frame(length = mean(df[ ,'Petal.Length']))
}
```

```
> mean_pet_length(iris)
```

```
  length
1  3.758
```

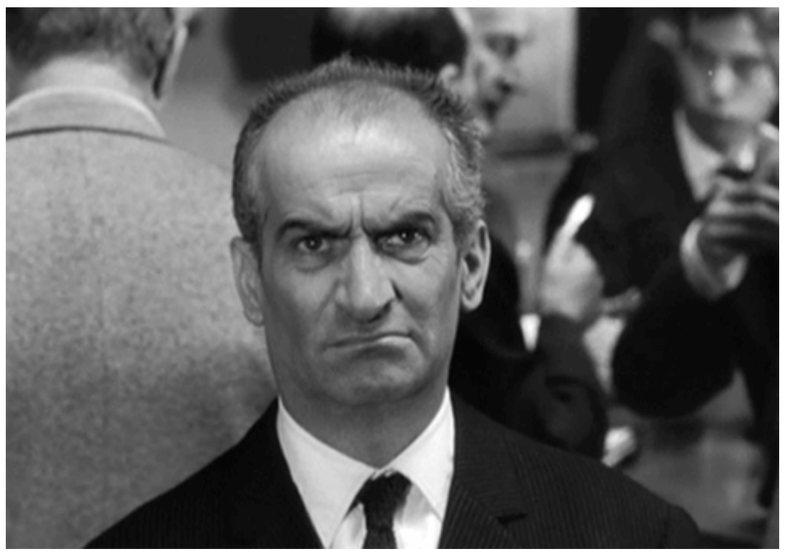
```
library(plyr)
> ddply(iris, ~ Species, mean_pet_length)
```

```
     Species  length
1     setosa   1.462
2 versicolor   4.260
3  virginica   5.552
```

[2]ddply()

(a bit) more complicated

task: get the mean **sepal length** and **width** for each species



[2] ddply()

task: get the mean **sepal length** and **width** for each species

```
# Define the function to apply
mean_sep_dims <- function(df) {
  data.frame(length = mean(df[ ,'Sepal.Length']))
}
```

[2] ddply()

task: get the mean **sepal length** and **width** for each species

```
# Define the function to apply
mean_sep_dims <- function(df) {
  data.frame(length = mean(df[ ,'Sepal.Length']),
             width  = mean(df[ ,'Sepal.Width']))
}
```

[2] ddply()

task: get the mean **sepal length** and **width** for each species

```
# Define the function to apply
mean_sep_dims <- function(df) {
  data.frame(length = mean(df[ ,'Sepal.Length']),
             width  = mean(df[ ,'Sepal.Width']))
}
```

```
> mean_sep_dims(iris)
```

```
     length    width
1 5.843333 3.057333
```

[2] ddply()

task: get the mean **sepal length** and **width** for each species

```
# Define the function to apply
mean_sep_dims <- function(df) {
  data.frame(length = mean(df[ ,'Sepal.Length']),
             width  = mean(df[ ,'Sepal.Width']))
}
> mean_sep_dims(iris)
     length    width
1 5.843333 3.057333

library(plyr)
> ddply(iris, ~ Species, mean_sep_dims)
     Species length width
1     setosa  5.006 3.428
2 versicolor  5.936 2.770
3  virginica  6.588 2.974
```

Your turn

task: get the mean **petal length and width** for each species

[3]ddply()

```
# Define the function to apply
mean_sep_dims <- function(df) {
  c(mean(df[ ,'Sepal.Length']), mean(df[ ,'Sepal.Width']))
}
```

[3]ddply()

```
# Define the function to apply
mean_sep_dims <- function(df) {
  c(mean(df[ ,'Sepal.Length']), mean(df[ ,'Sepal.Width']))
}

> mean_sep_dims(iris)
                                    [1] 5.843333 3.057333
```

[3]ddply()

```
# Define the function to apply
mean_sep_dims <- function(df) {
  c(mean(df[ ,'Sepal.Length']), mean(df[ ,'Sepal.Width']))
}

> mean_sep_dims(iris)
[1] 5.843333 3.057333
```

```
> library(plyr)
> ddply(iris, ~ Species, mean_sep_dims)
```

	Species	V1	V2
1	setosa	5.006	3.428
2	versicolor	5.936	2.770
3	virginica	6.588	2.974

[3]ddply()

```
# Define the function to apply
mean_sep_dims <- function(df) {
  c(mean(df[ ,'Sepal.Length']), mean(df[ ,'Sepal.Width']))
}

> mean_sep_dims(iris)
                                    [1] 5.843333 3.057333
```

```
> library(plyr)
> ddply(iris, ~ Species, mean_sep_dims)
```

Species	V1	V2
setosa	5.006	3.428
versicolor	5.936	2.770
virginica	6.588	2.974

[3]ddply()

The most unambiguous behaviour is achieved when [the function to apply] returns a data frame - in that case pieces will be combined...

(from ?ddply)

-> Always return a **data.frame** [with **names**, ndlr]

... do not program by coincidence

[https://pragprog.com/the-pragmatic-programmer/extracts/coincidence]

[3bis]ddply()

Make it short: use `summarise()`


```
> ddply(iris, ~ Species, summarise,
        length = mean(Sepal.Length),
        width  = mean(Sepal.Width))

     Species length width
1     setosa  5.006 3.428
2 versicolor  5.936 2.770
3  virginica  6.588 2.974
```

-> Do the same with Petals

[4]ddply()

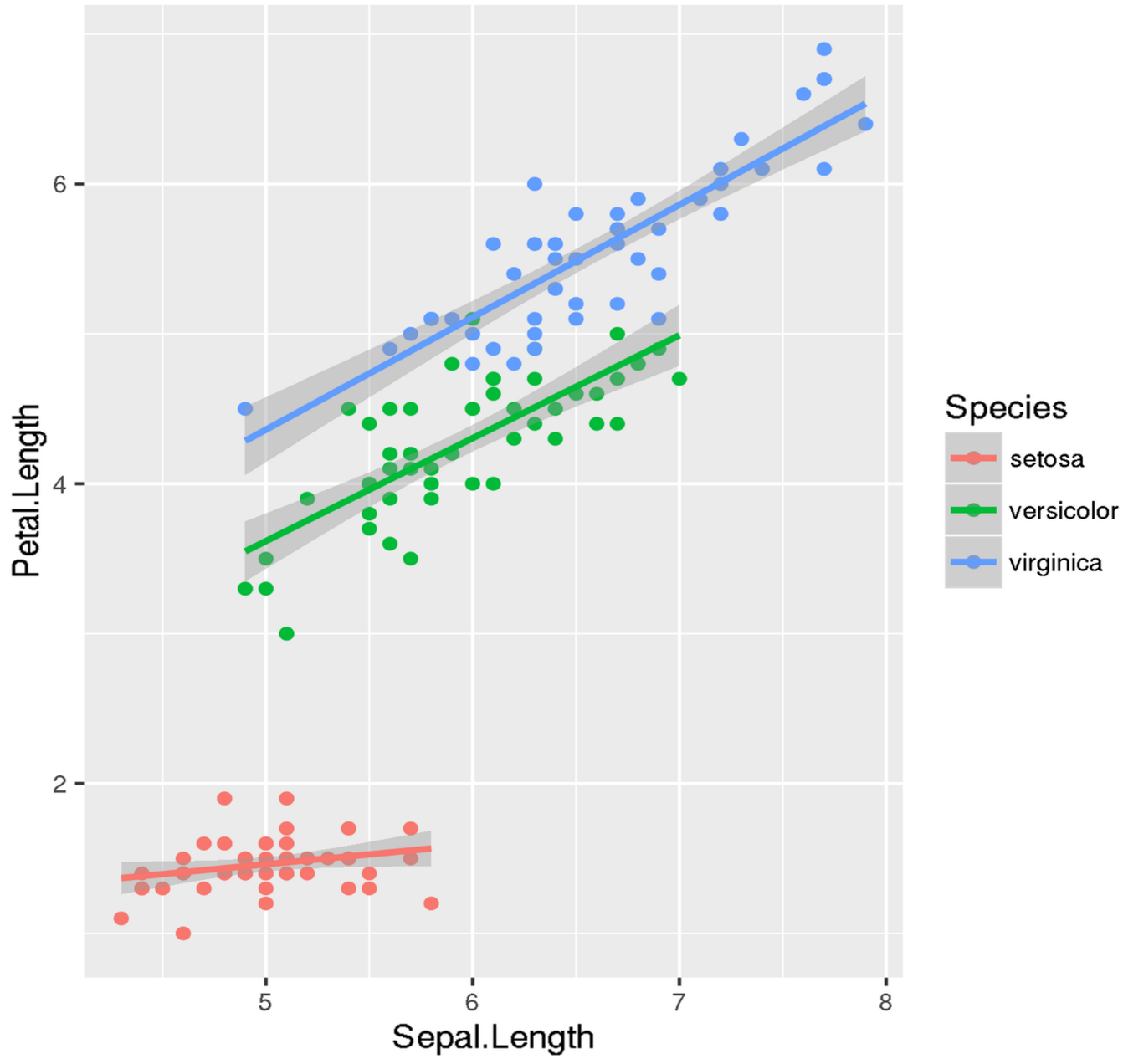
(a bit) more complicated

task: explore the **Petal length ~ Sepal length** relationship

[4] ddply()

task: explore the **Petal length ~ Sepal length** relationship

```
ggplot(iris, aes(Sepal.Length, Petal.Length, color = Species)) +
  geom_point() +
  geom_smooth(method = 'lm')
```



[4] ddply()

task: explore the **Petal length ~ Sepal length** relationship

get the coefficients of a linear regression (example for all species altogether)

```
iris_model <- lm(Petal.Length ~ Sepal.Length,
   data = iris)
coeffs <- coefficients(iris_model)
```

write this within a function that returns a data.frame !

[4] ddply()

task: explore the **Petal length ~ Sepal length** relationship

get the coefficients of a linear regression (example for all species altogether)

```
iris_model <- lm(Petal.Length ~ Sepal.Length,
   data = iris)
coeffs <- coefficients(iris_model)
```

write this within a function that returns a data.frame !

```
get_coeffs <- function(df) {
  model <- lm(Petal.Length ~ Sepal.Length, data = df)
  coeffs <- coefficients(model)

  # coeffs is a vector -> convert to data.frame
  data.frame(slope     = coeffs["Sepal.Length"],
             intercept = coeffs["(Intercept)"])
}
```

[4] ddply()

task: explore the **Petal length ~ Sepal length** relationship

```
get_coeffs <- function(df) {
  model <- lm(Petal.Length ~ Sepal.Length, data = df)
  coeffs <- coefficients(model)

  # coeffs is a vector -> convert to data.frame
  data.frame(slope     = coeffs["Sepal.Length"],
             intercept = coeffs["(Intercept)"])
}
```

```
> ddply(iris, ~ Species, get_coeffs)
```

	Species	slope	intercept
1	setosa	0.1316317	0.8030518
2	versicolor	0.6864698	0.1851155
3	virginica	0.7500808	0.6104680

Your turn

task: get the coefficients of the **Petal.Width ~ Sepal.Width** relationship for each species

[5] ddply()

What if my task takes a lot of time ?

e.g. get bootstrapped estimates of slope coefficients (Petal.Length ~ Sepal.Length)

```
> source('http://alex.lecairn.org/boot_iris.R')

> ddply(iris, ~ Species, get_boot_coeffs)
```

data.frame with 3000 rows and 3 columns

[5]ddply()

What if my task takes a lot of time ?

Solution 1: report about the time it takes and go get a coffee

```
ddply(iris, ~ Species, get_boot_coeffs,
      .progress = 'time')
```

(try also .progress = 'tk' or 'win')

[5]ddply()

What if my task takes a lot of time ?

Solution 1: report about the time it takes and go get a coffee

```
ddply(iris, ~ Species, get_boot_coeffs,
      .progress = 'time')
```

(try also .progress = 'tk' or 'win')

Solution 2: use parallel computing

```
library(doParallel)
registerDoParallel(cl = makeCluster(2))

ddply(iris, ~ Species, get_boot_coeffs,
      .parallel = TRUE)
```

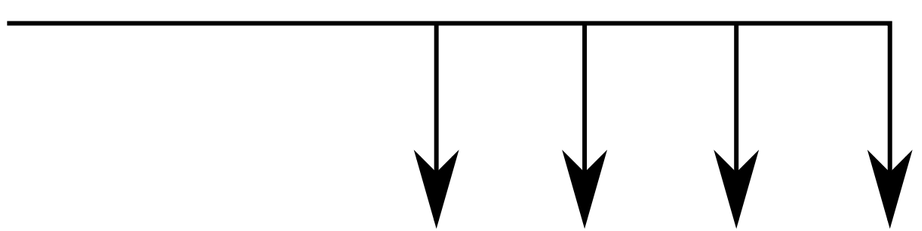
WHAT IF I DDPLY DOES NOT DO IT ?

WHAT IF I DDPLY DOES NOT DO IT ?

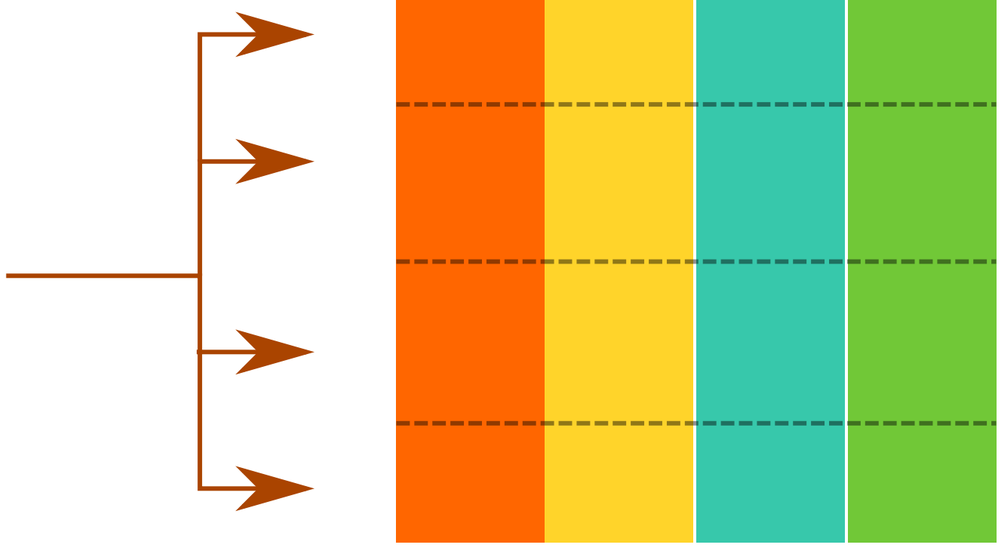
YOU PROBABLY WANT TO REFORMAT YOUR DATA, BUT....

a*ply: apply on all **rows of a data.frame**
rows or columns of a matrix

```
a*ply(matrix, 2, mean)
```



```
a*ply(matrix, 1, mean)
```



e.g. Compute abundances of species in a community matrix

```
> library(vegan)
> data(dune)
> dune <- as.matrix(dune)
```

species ->

<- sites

	Achimill	Agrostol	Airaprae	Alopgeni	Anthodor	Bellpere	Bromhord	Chenalbu	Cirsarve	Comapalu	Eleopalu	Elymr
1	1	0	0	0	0	0	0	0	0	0	0	
2	3	0	0	2	0	3	4	0	0	0	0	
3	0	4	0	7	0	2	0	0	0	0	0	
4	0	8	0	2	0	2	3	0	2	0	0	
5	2	0	0	0	4	2	2	0	0	0	0	
6	2	0	0	0	3	0	0	0	0	0	0	

"a community matrix"

e.g. Compute abundances of species in a community matrix

species ->

<- sites

	Achimill	Agrostol	Airaprae	Alopgeni	Anthodor	Bellpere	Bromhord	Chenalbu	Cirsarve	Comapalu	Eleopalu	Elymr
1	1	0	0	0	0	0	0	0	0	0	0	
2	3	0	0	2	0	3	4	0	0	0	0	
3	0	4	0	7	0	2	0	0	0	0	0	
4	0	8	0	2	0	2	3	0	2	0	0	
5	2	0	0	0	4	2	2	0	0	0	0	
6	2	0	0	0	3	0	0	0	0	0	0	

```
> aaply(dune, 2, mean) # average species abundance (column sums)
```

```
> aaply(dune, 1, mean) # average site abundance (row sums)
```

e.g. Compute abundances of species in a community matrix

species ->

<- sites

	Achimill	Agrostol	Airaprae	Alopgeni	Anthodor	Bellpere	Bromhord	Chenalbu	Cirsarve	Comapalu	Eleopalu	Elymr
1	1	0	0	0	0	0	0	0	0	0	0	
2	3	0	0	2	0	3	4	0	0	0	0	
3	0	4	0	7	0	2	0	0	0	0	0	
4	0	8	0	2	0	2	3	0	2	0	0	
5	2	0	0	0	4	2	2	0	0	0	0	
6	2	0	0	0	3	0	0	0	0	0	0	

```
> aaply(dune, 2, mean) # average species abundance (column sums)

> aaply(dune, 1, mean) # average site abundance (row sums)
```

Your turn: do the same with **adply**

-> what changes ?

e.g. Compute abundances of species in a community matrix

species ->

<- sites

	Achimill	Agrostol	Airaprae	Alopgeni	Anthodor	Bellpere	Bromhord	Chenalbu	Cirsarve	Comapalu	Eleopalu	Elymr
1	1	0	0	0	0	0	0	0	0	0	0	
2	3	0	0	2	0	3	4	0	0	0	0	
3	0	4	0	7	0	2	0	0	0	0	0	
4	0	8	0	2	0	2	3	0	2	0	0	
5	2	0	0	0	4	2	2	0	0	0	0	
6	2	0	0	0	3	0	0	0	0	0	0	

```
> aaply(dune, 2, mean) # average species abundance (column sums)

> aaply(dune, 1, mean) # average site abundance (row sums)
```

e.g. Compute abundances of species in a community matrix

species ->

<- sites

	Achimill	Agrostol	Airaprae	Alopgeni	Anthodor	Bellpere	Bromhord	Chenalbu	Cirsarve	Comapalu	Eleopalu	Elymr
1	1	0	0	0	0	0	0	0	0	0	0	
2	3	0	0	2	0	3	4	0	0	0	0	
3	0	4	0	7	0	2	0	0	0	0	0	
4	0	8	0	2	0	2	3	0	2	0	0	
5	2	0	0	0	4	2	2	0	0	0	0	
6	2	0	0	0	3	0	0	0	0	0	0	

Multi-argument example: `diversity(data, index = 'Shannon')`

```
> aaply(dune, 1, diversity, index = 'Simpson')
```

Your turn: do the same with **adply**

-> what changes ?

But really, use (long) data.frames !

- native to R
- reliable combine behavior
- integrates well with ggplot
- everything works with d.f.
- more expressive

...

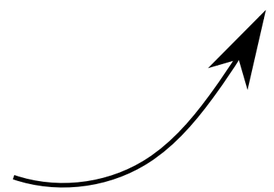
But really, use (long) data.frames !

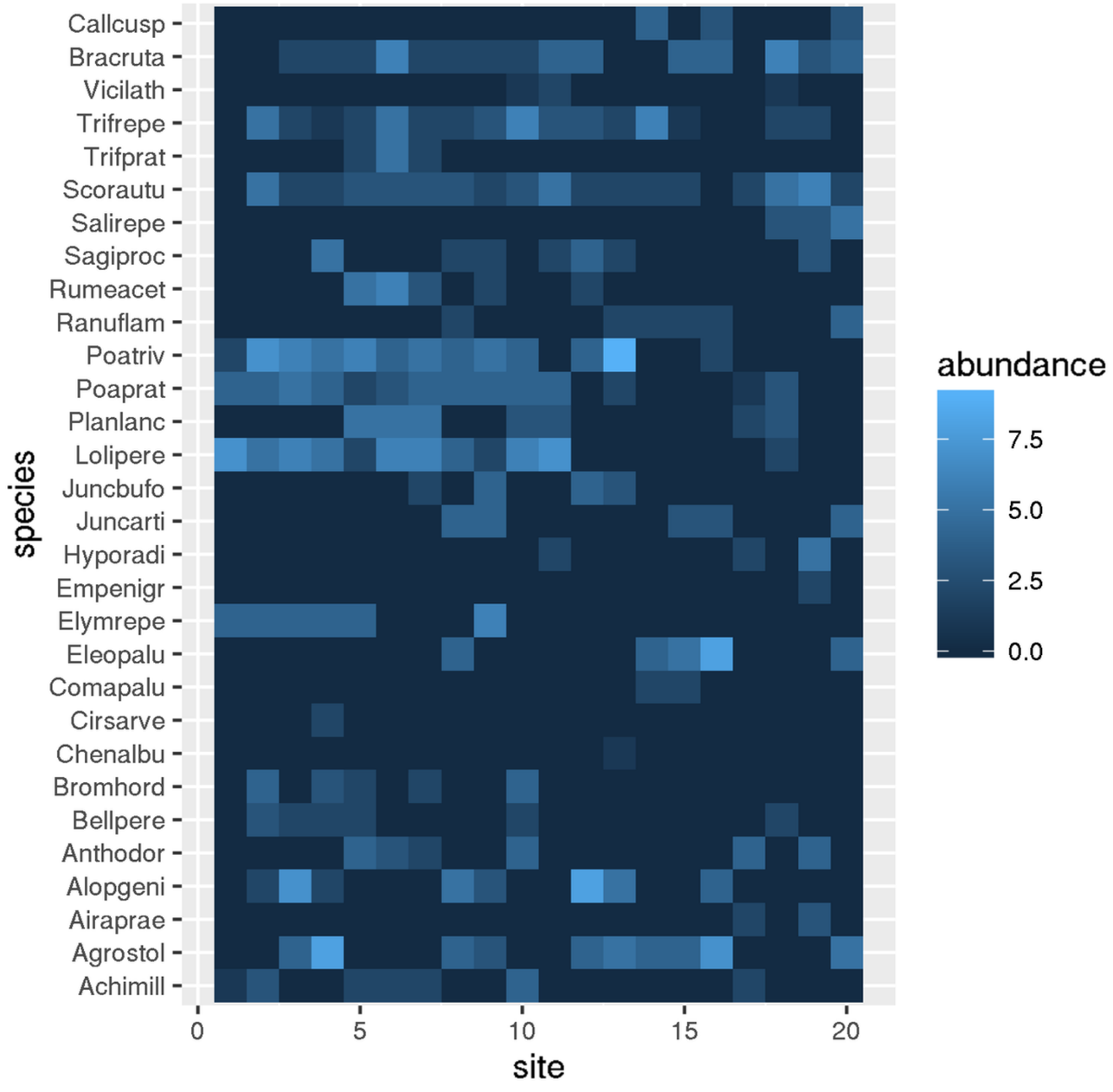
- native to R
- reliable combine behavior
- integrates well with ggplot
- everything works with d.f.
- more expressive

...

```
> # transform into long data.frame
> library(reshape2)
> dune <- melt(dune)
> names(dune) <- c('site', 'species', 'abundance')
```

	site	species	abundance
1	1	Achimill	1
2	1	Agrostol	0
3	1	Airaprae	0
4	1	Alopgeni	0
5	1	Anthodor	0
6	1	Bellpere	0
7	2	Achimill	3
8	2	Agrostol	0
9	2	Airaprae	0
10	2	Alopgeni	2
11	2	Anthodor	0
12	2	Bellpere	3
13	3	Achimill	0
14	3	Agrostol	4
15	3	Airaprae	0
16	3	Alopgeni	7
17	3	Anthodor	0






```
> # transform into long data.frame
> library(reshape2)
> dune <- melt(dune)
> names(dune) <- c('site', 'species'

> ggplot(dune) +
>   geom_tile(aes(x = site, y = species, fill = abundance))
```

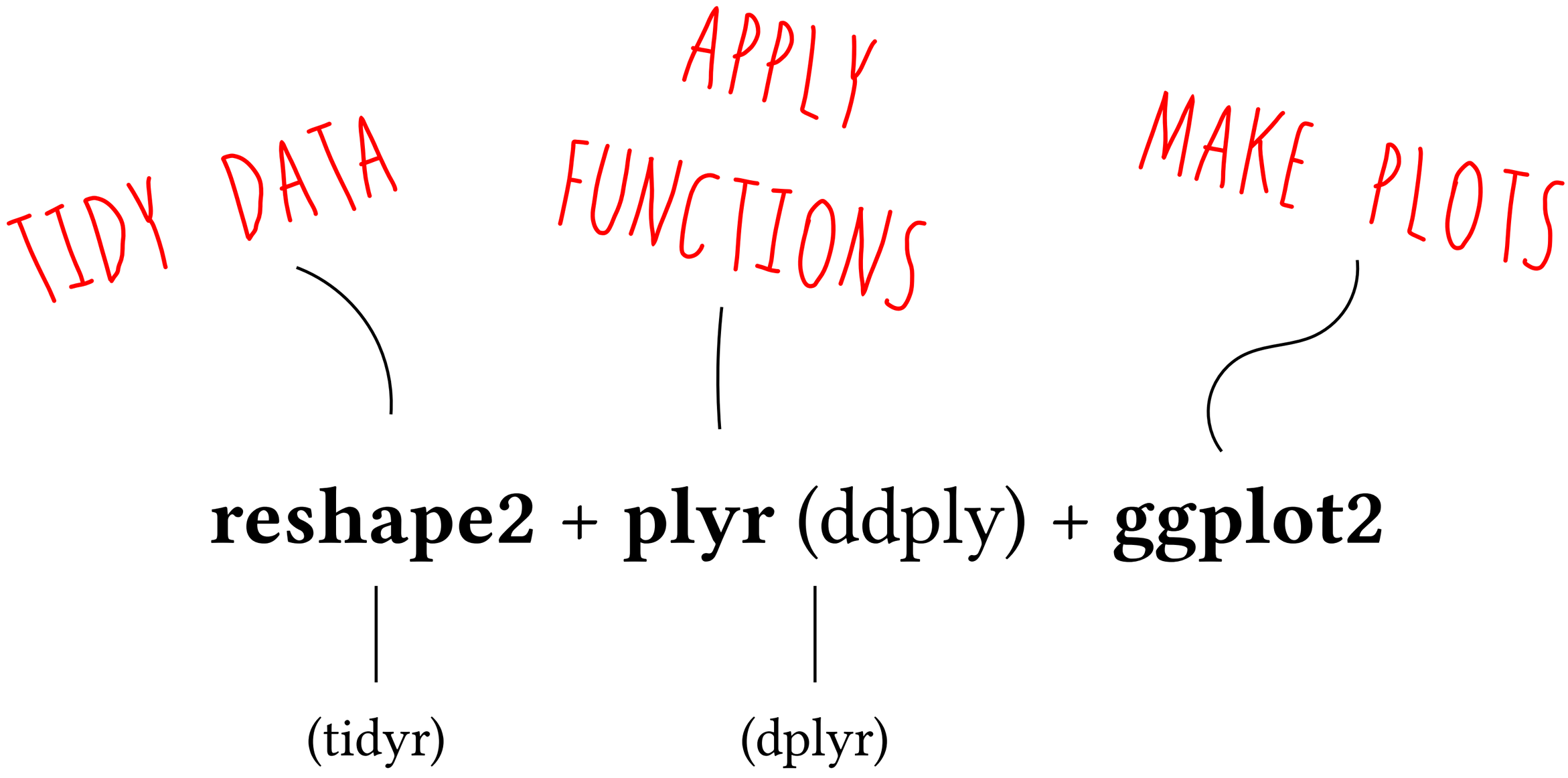
```
> # transform into long data.frame
> library(reshape2)
> dune <- melt(dune)
> names(dune) <- c('site', 'species', 'abundance')

> ddply(dune, ~ species, summarise, ab = mean(abundance)) # mean by species

> ddply(dune, ~ site, summarise, ab = mean(abundance)) # mean by site
```

```
    site  species abundance
1      1 Achimill         1
2      1 Agrostol         0
3      1 Airaprae         0
4      1 Alopgeni         0
5      1 Anthodor         0
6      1 Bellpere         0
7      2 Achimill         3
8      2 Agrostol         0
9      2 Airaprae         0
10     2 Alopgeni         2
11     2 Anthodor         0
12     2 Bellpere         3
13     3 Achimill         0
14     3 Agrostol         4
15     3 Airaprae         0
16     3 Alopgeni         7
17     3 Anthodor         0
```

The modern R workflow



(thank you)