An introduction to 

Rearranging and manipulating data

Noémie Becker & Benedikt Holtmann

Winter Semester 16/17

Course outline

- Review – Checking and cleaning data
- Rearranging and manipulating data
  - Reshaping data
  - Combining data sets
  - Making new variables
  - Subsetting data
  - Summarising data

Import data

Set working directory using **setwd()**

```
setwd("~/Desktop")
```

Import data using **read.table()** and **read.csv()** functions

```
myData<- read.csv(file = "datafile.csv”,
                  header = TRUE,
                  sep = ", ",
                  strip.white = TRUE,
                  na.strings = " ")
```

Get an overview

```
str(datafile)
```

```
'data.frame':     769 obs. of  12 variables:
$ Snail.ID: int  1 1 1 1 1 1 1 1 1 1 ...
$ Sex      : Factor w/ 4 levels "female","male",..: 2 2 4 2 2 2 2 2 2 2 ...
$ Size     : Factor w/ 2 levels "large","small": 2 2 2 2 2 2 2 2 2 2 ...
$ Feeding : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
$ Distance: num  0.17 0.87 0.22 0.13 0.36 0.84 0.69 0.6 0.85 0.59 ...
$ Depth   : num  1.66 1.26 1.43 1.46 1.21 1.56 1.62 162 1.96 1.93 ...
$ Temp    : int  21 21 18 19 21 21 20 20 19 19 …
```

Get an overview

summary(datafile**)**

```
    Snail.ID        Sex          Size          Feeding          Distance
Min.   : 1.00   female:384   large:383   Mode :logical   Min.   :0.0000
1st Qu.: 4.00   male  :385   small:385   FALSE:503       1st Qu.:0.2800
Median : 8.50                NA's :  1   TRUE :266       Median :0.5100
Mean   : 8.49                            NA's :0         Mean   :0.5125
3rd Qu.:12.00                                            3rd Qu.:0.7500
Max.   :16.00                                            Max.   :1.0000 ...
...
...
Continues
```

Get an overview

head(x)
tail(x)

	Snail.ID	Sex	Size	Feeding	Distance	Depth	Temp
1	1	male	small	FALSE	0.17	1.66	21
2	1	male	small	FALSE	0.87	1.26	21
3	1	male	small	FALSE	0.22	1.43	18
4	1	male	small	FALSE	0.13	1.46	19
5	1	male	small	FALSE	0.36	1.21	21
6	1	male	small	TRUE	0.84	1.56	21

Get an overview

str()	provides an overview of an object
summary()	returns basic statistical summary for variables
head()	returns the first records of an object
tail()	returns the last records of an object
sort()	sorts a vector or factor into ascending or descending order
order()	takes a set of vectors as arguments and sorts recursively by each vector, breaking ties by looking at successive vectors in the argument list

Built-in data

- Many packages come with built-in data sets
- To save memory, data sets are not loaded until they are referenced the first time
- The function **data()** will list all loaded packages and their built-in data sets
- Built-in data is usually used for examples that you can find in the help file

Course outline

- Review – Checking and cleaning data
- **Rearranging and manipulating data**
  - Reshaping data
  - Combining data sets
  - Making new variables
  - Subsetting data
  - Summarising data

Reshaping data

We will use data on fish abundance

```
Fish_survey<- read.csv("Fish_survey.csv", header = TRUE)
```

	Site	Month	Transect	Trout	Perch	Stickleback
1	River1	January	1	10	5	28
2	River1	January	2	0	13	42
3	River1	January	3	8	19	9
4	River2	January	1	3	5	72
5	River2	January	2	2	9	33
6	River2	January	3	15	24	65

...

...

Reshaping data

We will use data on fish abundance

```
Fish_survey<- read.csv("Fish_survey.csv", header = TRUE)
```

	Site	Month	Transect	Trout	Perch	Stickleback
1	River1	January	1	10	5	28
2	River1	January	2	0	13	42
3	River1	January	3	8	19	9
4	River2	January	1	3	5	72
5	River2	January	2	2	9	33
6	River2	January	3	15	24	65

...

...

Reshaping data

Reshaping data using the package **tidyr**

library(tidyr)

To make one single column including all three species you can use the function **gather()**

Reshaping data

Example gather()

```
Fish_survey_long <- gather(Fish_survey, Species, Abundance, 4:6)
```

	Site	Month	Transect	Species	Abundance
1	River1	January	1	Trout	10
2	River1	January	2	Trout	0
3	River1	January	3	Trout	8
4	River2	January	1	Trout	3
5	River2	January	2	Trout	2
6	River2	January	3	Trout	15

…

…

Reshaping data

To convert the data back into a format with separate columns for each use the function **spread()**

Example spread()

Fish_survey_wide <- spread(Fish_survey_long, Species, Abundance)

Reshaping data

Reshaping data using the package **reshape2**

library(reshape2)

Instead of **gather()** the reshape2 package uses the function **melt()**

Caution: Do not confuse the reshape2 library with the reshape function!!

Reshaping data

Reshaping data using the package **reshape2**

Example melt()

```
Fish_survey_long <- melt(Fish_survey,
                         id.vars = c("Site","Month","Transect"),
                         measure.vars = c("Trout","Perch","Stickleback"),
                         variable.name = "Species", value.name = "Abundance")
```

Reshaping data

Reshaping data using the package **reshape2**

Similarly, instead of **spread()** the reshape2 package uses the function **dcast()**

Example dcast()

```
Fish_survey_wide <- dcast(Fish_survey_long,
                          Site + Month + Transect ~ Species,
                          value.var = "Abundance")
```

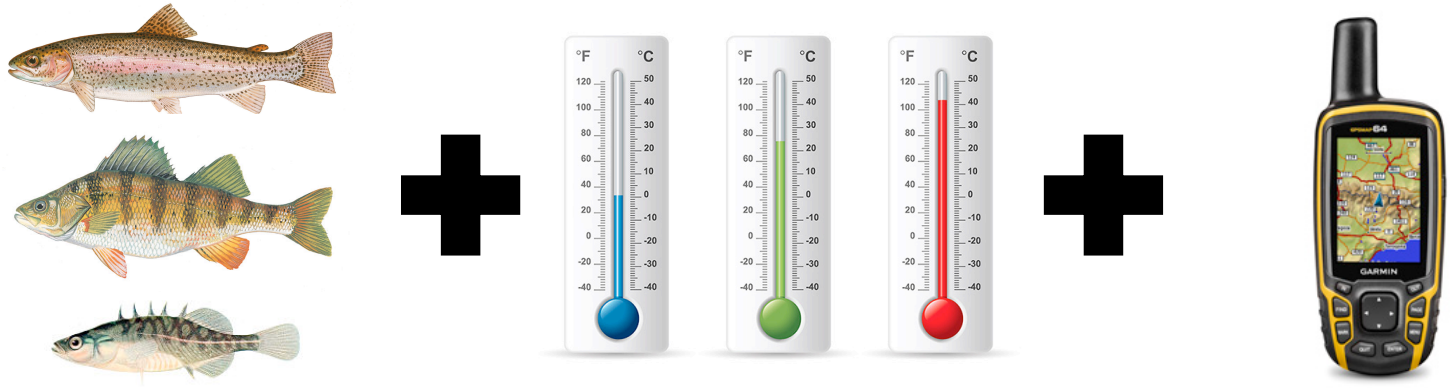
Combining data sets

To combine data sets we will use the package **dplyr**

```
install.packages(dplyr)
library(dplyr)
```

Combining data sets

To combine data sets we will use the package **dplyr**



Combining data sets

To combine data sets we will use the package **dplyr**

Import data sets

```
Fish_survey_long<- read.csv("Fish_survey_long.csv", header = TRUE,
stringsAsFactors=FALSE)

Water_data<- read.csv("Water_data.csv", header = TRUE,
stringsAsFactors=FALSE)

GPS_location<- read.csv("GPS_data.csv", header = TRUE,
stringsAsFactors=FALSE)
```

Combining data sets

Why not just use **cbind()**?

- Data sets need to have the **same number** of rows
- Rows need to be in the **same order** because rows are matched by position

X1	X2
A	1
B	1
A	2
B	2

+

X1	X3
A	T
A	F
B	F
B	T

+

X1	X4
A	1
A	2
A	3

Combining data sets

We can join data sets by using the columns they share:

Fish survey
Site
Month
Transect
Species

+

Water characteristics
Site
Month
Water temp.
O_2 - content

+

GPS
Site
Transect
Latitude
Longitude

Combining data sets

Functions to combine data sets in dplyr	
left_join(a, b, by = "x1")	Joins matching rows from b to a
right_join(a, b, by = "x1")	Joins matching rows from a to b
inner_join(a, b, by = "x1")	Returns all rows from a where there are matching values in b
full_join(a, b, by = "x1")	Joins data and returns all rows and columns
semi_join(a, b, by = "x1")	All rows in a that have a match in b, keeping just columns from a.
anti_join(a, b, by = "x1")	All rows in a that do not have a match in b

Combining data sets

1. Join water characteristics to fish abundance data using **inner_join()**

```
Fish_and_Water <- inner_join(Fish_survey_long, Water_data,
                             by = c("Site", "Month"))
```

Combining data sets

Check the new data frame

```
str(Fish_and_Water)
head(Fish_and_Water)
```

```
'data.frame': 72 obs. of  7 variables:
 $ Site          : Factor w/ 2 levels "River1","River2": 1 1 1 2 2 2 1 1 1 2 ...
 $ Month         : chr  "January" "January" "January" "January" ...
 $ Transect      : int  1 2 3 1 2 3 1 2 3 1 ...
 $ Species       : Factor w/ 3 levels "Perch","Stickleback",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ Abundance     : int  10 0 8 3 2 15 2 7 0 11 ...
 $ Mean_water_temp: num  3.6 3.6 3.6 6.2 6.2 6.2 2.3 2.3 2.3 8 ...
 $ Mean_O2_content: num  12.6 12.6 12.6 12 12 12 9.8 9.8 9.8 12.3 ...
```

Combining data sets

2. Add GPS loacations to new Fish_and_Water data set using **inner_join()**

```
Fish_survey_combined <- inner_join(Fish_and_Water, GPS_location,
                                   by = c("Site", "Transect"))
```

Check if it worked:

```
str(Fish_survey_combined)
head(Fish_survey_combined)
```

Adding new variables

We will use data on bird behaviour

```
Bird_Behaviour<- read.csv("Bird_Behaviour.csv", header = TRUE,
                          stringsAsFactors=FALSE)
# Get an overview
str(Bird_Behaviour)
```

X1	X2
A	1
B	1
A	2
B	2

X1	X2	X3
A	1	T
B	1	F
A	2	T
B	2	F

Adding new variables

Three ways adding a new variable (log of FID)

Using **$**

```
Bird_Behaviour$log_FID <- log(Bird_Behaviour$FID)
```

Using **[ ]** - operator

```
Bird_Behaviour[ , "log_FID"] <- log(Bird_Behaviour$FID)
```

Using **mutate()** from **dplyr** package

```
Bird_Behaviour <- mutate(Bird_Behaviour, log_FID = log(FID))
```

Adding new variables

Adding a new variable

head(Bird_Behaviour)

	Ind	Species	Sex	Year	FID	Disturbance	Fledglings	**log_FID**
1	PD1	Passer_domesticus	male	2013	5	8	1	1.6094379
2	PD1	Passer_domesticus	male	2014	2	40	4	0.6931472
3	PD1	Passer_domesticus	male	2015	8	30	4	2.0794415
4	PD2	Passer_domesticus	female	2013	10	35	3	2.3025851
5	PD2	Passer_domesticus	female	2014	10	15	0	2.3025851
6	PD2	Passer_domesticus	female	2015	6	6	2	1.7917595

Adding new variables

Split one column into two using **separate()** from **dplyr** package

```
Bird_Behaviour <- separate(Bird_Behaviour, Species,
                           c("Genus","Species"), sep="_", remove=TRUE)
```

X1	X2
A	1_1
B	1_2
A	2_1
B	2_2

X1	X2.1	X2.2
A	1	1
B	1	2
A	2	1
B	2	2

Adding new variables

Split one column into two using **separate()** from **dplyr** package

head(Bird_Behaviour)

	Ind	**Genus**	**Species**	Sex	Year	FID	...
1	PD1	Passer	domesticus	male	2013	5	...
2	PD1	Passer	domesticus	male	2014	2	...
3	PD1	Passer	domesticus	male	2015	8	...
4	PD2	Passer	domesticus	female	2013	10	...
5	PD2	Passer	domesticus	female	2014	10	...
6	PD2	Passer	domesticus	female	2015	6	...

Adding new variables

Combine two columns using **unite()** from **tidyr** package

```
Bird_Behaviour <- unite(Bird_Behaviour, "Genus_Species",
                        c(Genus, Species),  sep="_", remove=TRUE)
```

X1	X2.1	X2.2
A	1	1
B	1	2
A	2	1
B	2	2

X1	X2
A	1_1
B	1_2
A	2_1
B	2_2

Adding new variables

Combine two columns using **unite()** from **tidyr** package

head(Bird_Behaviour)

	Ind	**Genus_Species**	Sex	Year	FID	...
1	PD1	Passer_domesticus	male	2013	5	...
2	PD1	Passer_domesticus	male	2014	2	...
3	PD1	Passer_domesticus	male	2015	8	...
4	PD2	Passer_domesticus	female	2013	10	...
5	PD2	Passer_domesticus	female	2014	10	...
6	PD2	Passer_domesticus	female	2015	6	...

Subsetting data

Subsetting data

- Using [ ] – operator
- Using **subset()**
- Subsetting with functions from **dplyr** package
  - slice()
  - filter()
  - sample_frac()
  - sample_n()
  - select()

Subsetting data

Subsetting using [ ] - operator

Examples:

```
Bird_Behaviour[ , 1:4]       # selects the first 4 columns

Bird_Behaviour[c(2,3), ]    # selects rows 2 and 3

Bird_Behaviour[1:3, 1:4]  # selects the rows 1 to 3 and columns 1 to 4

Bird_Behaviour[c(1:3, 6), c(1:4, 8]  # selects the rows 1 to 3 and 6,
and the columns 1 to 4 and 8
```

Subsetting data

Subsetting using [ ] and $ operator

Examples:

```
BirdBird_Behaviour[Bird_Behaviour$Sex == "male",  ]

# selects all rows with males
```

Subsetting data

Subsetting using **subset()**

subset(x, subset, select, ...)

Argument	Description
x	The object from which to extract subset
subset	A logical expression that describes the set of rows to return
select	An expression indicating which columns to return

Subsetting data

Examples subset():

subset(Bird_Behaviour, FID < 10)
selects all rows with FID smaller than 10m

subset(Bird_Behaviour, FID < 10 & Sex == "male")
selects all rows for males with FID smaller than 10m

subset(Bird_Behaviour, FID > 10 | FID < 15, select = c(Ind, Sex, Year))
selects all rows that have a value of FID greater than 10 or less than 15. We keep only the IND, Sex and Year column

Subsetting data

Review of logical operators

Operator	Description
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to
x & y	x and y
x | y	x or y

Check out **?base::Logic** and **?Comparison** to learn more

Subsetting rows in dplyr

Subsetting by rows using **slice()** and **filter()**

Examples slice() and filter():

Bird_Behaviour.sclice<- slice(Bird_Behaviour, 3:5) # selects rows 3-5

Bird_Behaviour.filter<- filter(Bird_Behaviour, FID < 5) # selects rows that meet certain criteria

Subsetting rows in dplyr

Taking a random sample of rows using **sample_frac()** and **sample_n()**

Examples sample_frac() and **sample_n():**

```
Bird_Behaviour.50 <- sample_frac(Bird_Behaviour, size = 0.5,
                                 replace=FALSE)
# takes randomly 50% of the rows


Bird_Behaviour_50Rows<- sample_n(Bird_Behaviour, 50,
                                 replace=FALSE)
# takes randomly 50 rows
```

Subsetting columns in dplyr

Subsetting by columns using **select()**

Examples select():

Bird_Behaviour_col <- select(Bird_Behaviour, Ind, Sex, Fledglings)
selects the columns Ind, Sex, and Fledglings

Bird_Behaviour_reduced <- select(Bird_Behaviour, -Disturbance)
excludes the variable disturbance

Summarising data

Summarizing data with **dplyr**

Get the overall mean for FID using **summarise()** and **mean()**

```
summarise(Bird_Behaviour, mean.FID=mean(FID))
```

```
    mean.FID
1   11.82639
```

Summarising data

Summarizing data with dplyr

We can add other measurements to this:

```
summarise(Bird_Behaviour,
                mean.FID=mean(FID),      # mean
                min.FID=min(FID),        # minimum
                max.FID=max(FID),        # maximum
                med.FID=median(FID),     # median
                sd.FID=sd(FID),          # standard deviation
                var.FID=var(FID),        # variance
                n.FID=n())               # sample size
```

Summarising data

Summarizing data with dplyr

We can add other measurements to this:

	mean.FID	min.FID	max.FID	med.FID	sd.FID	var.FID	n.FID
1	11.82639	1	30	10	8.082036	65.3193	144

Summarising data

Summarizing data with dplyr

Get summaries for each species

Before you calculate summaries, you have to apply the **group_by()** function

Bird_Behaviour_by_Species <- group_by(Bird_Behaviour, Species)

Summarising data

Summarizing data with dplyr

After we applied the **group_by()** function, we can get summaries for each species

```
Summary.species<- summarise(Bird_Behaviour_by_Species,
                            mean.FID=mean(FID),      # mean
                            min.FID=min(FID),        # minimum
                            max.FID=max(FID),        # maximum
                            med.FID=median(FID),     # median
                            sd.FID=sd(FID),          # standard deviation
                            var.FID=var(FID),        # variance
                            n.FID=n())               # sample size
```

Summarising data

Summarizing data with dplyr

as.data.frame(Summary.species)

	Species	mean.FID	min.FID	max.FID	med.FID	sd.FID	var.FID	n.FID
1	Fringilla_coelebs	20.44	5	30	21	6.31	39.83	48
2	Passer_domesticus	6.10	1	10	7	3.12	9.71	48
3	Passer_montanus	8.94	1	20	8	5.61	31.51	48

Which R functions did we learn?

gather()	takes multiple columns and collapses them into key-value pairs
spread()	spreads a key-value pair across multiple columns
melt()	reshapes wide format to long format
dcast()	reshapes long format to wide format
inner_join()	Joins data and returns all rows from x where there are matching values in y, and all columns from x and y
separate()	separates single column into multiple columns
unite()	pastes multiple columns into one
subset()	returns subsets which meet certain conditions
slice()	selects rows by position
filter()	extracts rows that meet logical criteria
sample_frac()	randomly selects a fraction of rows

Which R functions did we learn?

sample_n()	randomly selects n rows
select()	selects columns by name or helper function
summarise()	summarises multiple values to a single value
mean()	computes the arithmetic mean
min()	returns the minimum of the input values
max()	returns the maximum of the input values
median()	computes the median
sd()	computes the standard deviation
var()	computes the variance
n()	returns the number of rows
group_by()	takes an existing table and converts it into a grouped table where operations are performed "by group"
View()	invokes a spreadsheet-style data viewer on a matrix-like object