Basic R functions

Getting help

`help(topic)` or `?topic`	Documentation on *topic*.
`help.search("topic")`	Search the help system for *topic*.
`apropos("topic")`	Find objects by (partial) name in the search list matching the regular expression *topic*.
`help.start()`	Start the HTML version of help.

Working directory

`getwd()`	Get current working directory.
`setwd("path/to/files")`	Set working directory.
`dir()`	Show files in the current directory (same as `list.files()`).
`dir(recursive=TRUE)`	Show all files, including subdirectories.

Working environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

Packages

`install.packages()`	Install packages from repositories or local files.
`library(dplyr)`	Load and attach add-on package 'dplyr'.
`require(readr)`	Load and attach add-on package 'readr'.
`search()`	Get list of attached packages.
`attach(name)`	Attach set of R objects to search path.
`detach(name)`	Detach objects or libraries from search path.
`vignette(dplyr")`	View a specified package vignette (here from 'dplyr').
`browseVignettes("dplyr")`	List available vignettes for package 'dplyr' in HTML browser.
`dplyr::filter()`	Use a particular function from a package (here filter from 'dplyr').

Operators

Assignment

`<-`	Assign right value to left name.
`=`	Leftward, only allowed at top level.
`<<-`	Leftward, used inside functions.
`->`	Assign left value to right name.
`->>`	Rightward, used inside functions.

Arithmetic

`+`	Addition.
`-`	Subtraction.
`*`	Multiplication.
`/`	Division.
`^`	Exponentiation.
x%%y	Modulus (x mod y) 5%%2 is 1.
x%/%y	Integer division 5%%2 is 2.

Relational

`<`	Less than.
`<=`	Less than or equal to.
`>`	Greater than.
`>=`	Greater than or equal to.
`==`	Exactly equal to.
`!=`	Not equal to.

Logical

`&`	Element-wise AND operation.
`|`	Element-wise OR operation.
!	Element-wise NOT operation.
`&&`	Operand-wise AND operation.
`||`	Operand-wise OR operation.

Others

`:`	Create regular sequence.
%in%	Value matching.
%*%	Matrix multiplication.
`|>`	Base R forward pipe operator.
%>%	*magrittr* forward-pipe operator.

Data types and conversion

Overview

Data type	Example	Check	Convert
logical	`x <- c(TRUE, FALSE)`	`is.logical(x)`	`as.logical(x)`
integer	`x <- c(5L, 9L)`	`is.integer(x)`	`as.integer(x)`
double	`x <- c(1.5, 4.7)`	`is.double(x)`	`as.double(x)`
numeric	`x <- c(5L, 4.7)`	`is.numeric(x)`	`as.numeric(x)`
complex	`x <- c(5i, 9i)`	`is.complex(x)`	`as.complex(x)`
character	`x <- c("A", "B")`	`is.character(x)`	`as.character(x)`
NULL	`x <- NULL`	`is.null(x)`	`as.null(x)`
NA	`x <- c(NA, NA)`	`is.na(x)`	`as.na(x)`
factor	`x <- factor(c("A","B"))`	`is.factor(x)`	`as.factor(x)`
date	`x <- as.Date("2020-06-01")`		`as.Date(x)`

For a complete list on checking and converting functions, use `methods(is)` and `methods(as)`.

`typeof(x)` Determine the (R internal) type or storage mode of any object.

Functions for logical data types

`isTRUE(x)` Check if *x* represents single TRUE value.
`isFALSE(x)` Check if *x* represents single FALSE value.
`any(x)` Check if at least one value in *x* is TRUE.
`all(x)` Check if all values in *x* are TRUE.

Data objects in R

Homogeneous data types

	Vector	Matrix	Array
Create	`c()`,`:`,`rep(x,..)`, `seq(from,to,by)`	`matrix(data,nrow,ncol)`, `cbind()`,`rbind()`	`array(data, dim)`
Subset	`[..]`	`[..,..]`	`[..,..,..]`
Check	`is.vector(x)`	`is.matrix(x)`	`is.array(x)`
Convert	`as.vector(x)`	`as.matrix(x)`	`as.array(x)`
Dimension `dim(x)`	1	2 (rows,cols)	3 (rows,cols,layers)
Length `length(x)`	items in vector	rows*cols	rows*cols *layers
Names	`names(x)`	`colnames(x)`, `rownames(x)`	

Heterogeneous data types

	List	Data frame	Tibble
Create	`list()`	`data.frame()`	`tibble::tibble()`
Subset	`$` or `[[..]]`	`$` or `[[..]]` or `[..,..]`	`$` or `[[..]]` or `[..,..]`
Check	`is.list(x)`	`is.data.frame(x)`	`tibble::is_tibble(x)`
Convert	`as.list(x)`	`as.data.frame(x)`	`tibble::as_tibble(x)`
Dimension `dim(x)`	1	2 (rows,cols)	2 (rows,cols)
Length `length(x)`	items in list	nr columns	nr columns
Names	`names(x)`	`names(x)`	`names(x)`

Object information

`class(x)`	Get or set the class of *x* (`class(x) <- "myclass"`).
`unclass(x)`	Remove the class attribute of *x*.
`attr(x,which)`	Get or set the attribute which of *x*.
`attributes(x)`	Get or set the list of attributes of *x*.
`str(x)`	Display the internal *structure* of an R object.
`nrow(x)`,`ncol(x)`)	Get number of rows and columns in 2- 3-dim. objects
`ls()`	Show objects in the search path; specify *pat="pat"* to search on a pattern.
`ls.str()`	`str()` for each object in the search path.
`methods(a)`	Show S3 methods of *a*.
`methods(class=class(a))`	List all the methods to handle objects of class *a*.
`print()`	Print its argument in the console; generic function.
`format(x)`	Format an R object for pretty printing.

Subsetting and extracting data

Base approach

Subscript	Explanation
Blank	Returns all elements
Positive	Vector of positive integers refers to element positions that should be returned.
Negative	Vector of negative integers refers to element positions that should be OMITTED.
Character	Vector of characters refers to element names that should be returned.
Logical	Vector of logical values returns only elements where corresponding value is TRUE.

Indexing vectors

Index	Returns
`x[2]`	2nd element.
`x[c(1,4)]`	1st and 4th element.
`x[-2]`	All EXCEPT 2nd element.
`x[-c(1,4]`	All EXCEPT 1st and 4th element.
`x["a"]`	Element named *a*.
`x[x>5]`	All elements greater than 5.
`x[x %in% c(1,4,13)]`	All elements equal to given set.
!	All elements equal to given set.
!	All elements equal to given set.

Indexing matrices

Index	Returns
`x[1:3, ]`	Row 1-3, all columns.
`x[ ,c(2,4)]`	All rows, column 2 and 4.
`x[1,2:4]`	Row 1, column 2-4.
`x["a", c("x", "y")]`	Row named *a*, columns named *x* or *y*.

Indexing lists

Index	Returns
`x[2]`	List with 2nd object n.
`x[[2]]`	2nd object.
`x[["a"]]`	Object named *a*.
`x$a`	Object named *a*.

Indexing data frames/tibbles

Same as indexing matrices plus:

Index	Returns
`x[["a"]]`	Column named *a*.
`x$a`	Column named *a*.

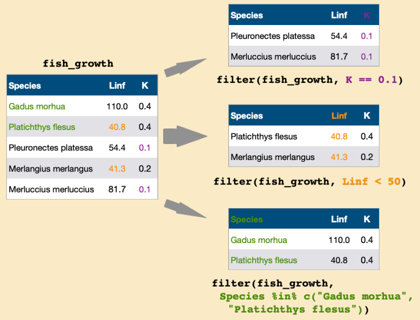
Additional function:
`subset(x,...)` Return subsets of vectors, matrices or data frames which meet conditions (...).

Tidyverse approach for dataframes & tibbles

`load(dplyr)` or `load(tidyverse)`

Extract rows

`filter(.data, a < 5)`	Extract rows that meet logical criteria (e.g. elements in col *a* less than 5).
`slice(.data, 5:9)`	Extract rows by position (e.g. row 5-9).



fish_growth

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4
Pleuronectes platessa	54.4	0.1
Merlangius merlangus	41.3	0.2
Merluccius merluccius	81.7	0.1

Species	Linf	K
Pleuronectes platessa	54.4	0.1
Merluccius merluccius	81.7	0.1

```
filter(fish_growth, K == 0.1)
```

Species	Linf	K
Platichthys flesus	40.8	0.4
Merlangius merlangus	41.3	0.2

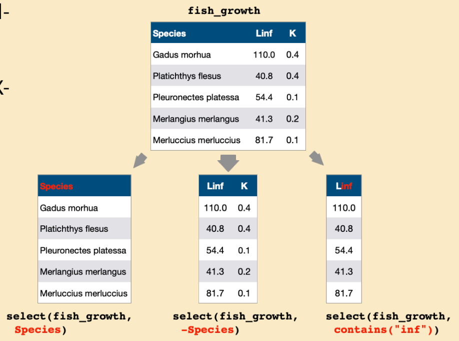
```
filter(fish_growth, Linf < 50)
```

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4

```
filter(fish_growth,
  Species %in% c("Gadus morhua",
   "Platichthys flesus"))
```

Extract columns

`select(.data, a, b)`	Extract columns (e.g. column *a* and *b*).
`select(.data, a:d)`	Extract columns *a* - *d*.
`select(.data, -a)`	Extract all columns EXCEPT *a*.



fish_growth

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4
Pleuronectes platessa	54.4	0.1
Merlangius merlangus	41.3	0.2
Merluccius merluccius	81.7	0.1

Species
Gadus morhua
Platichthys flesus
Pleuronectes platessa
Merlangius merlangus
Merluccius merluccius

```
select(fish_growth,
 Species)
```

Linf	K
110.0	0.4
40.8	0.4
54.4	0.1
41.3	0.2
81.7	0.1

```
select(fish_growth,
 -Species)
```

Linf
110.0
40.8
54.4
41.3
81.7

```
select(fish_growth,
 contains("inf"))
```

Input and output

Base functions

Input

`data(x)`	Load specified R data sets.
`load(file)`	Reload datasets written with `save()` function (file format: *.R*, *.Rdata*).
`read.table(file, sep="", dec=".")`	Read a file in table format (whitespace-separated) into data frame.
`read.csv(file, sep=",", dec=".")`	Read CSV file (comma-separated) into data frame.
`read.csv2(file, sep=";", dec=",")`	Read CSV file (semicolon-separated) into data frame.
`read.delim(file, sep="\t", dec=".")`	Read tab-delimited files into data frame.
`read.delim2(file, sep="\t", dec=",")`	Read tab-delimited files (with comma as decimal separator).
`read.fwf(file, width)`	Read fixed-width format files into data frame; *widths* is an integer vector, giving the widths of the fixed-width fields.

Output

`save(x,y, file)`	Write external representation of R objects (e.g. *x,y*) to specified file (use *.R* or *.Rdata* file extension).
`save.image(file)`	Same as `save()` but saves all objects in current workspace.
`write.table(x,file,sep=" ")`	Print matrix, data frame or tibble to a file; output format depends in the extension provided.
`write.csv(x,file)`	Print matrix, data frame or tibble to a CSV file (comma-separated).
`write.csv2(x,file)`	Print matrix, data frame or tibble to a CSV file (semicolon-separated).
`cat(..., file, sep=" ")`	Print the objects after coercing to character; useful for producing output in user-defined functions.
`sink(file)`	Save R output to file, until `sink()` is called again (with no argument).

Other file types

Additional packages to import the following file types:

- **readODS** - Calc files (.ods)
- **readxl** - Excel files (.xls and .xlsx)
- **haven** - SPSS, Stata, and SAS files
- **xml2** - XML
- **jsonlite** - json
- **DBI** - Databases
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Tidyverse approach: the readr package

`load(readr)` or `load(tidyverse)`

Input

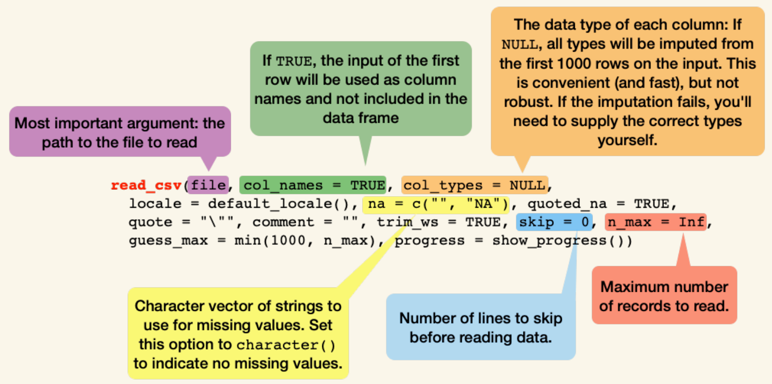
Read tabular or flat files into tibbles:

`read_table(file)`	Whitespace-separated files.
`read_csv(file)`	Comma-separated CSV files.
`read_csv2(file)`	Semicolon-separated CSV files.
`read_tsv(file)`	Tab-delimited files.
`read_delim(file, delim="|")`	Files with any delimiter/separator.
`read_fwf(file)`	Fixed-width files.

Read non-tabular files:

`read_file(file)`	Read file into single string.
`read_lines(file)`	Read each line into separate strings.
`read_log(file)`	Read Apache style log files.

All the read functions have the same or similar syntax:


The data type of each column: If NULL, all types will be imputed from the first 1000 rows on the input. This is convenient (and fast), but not robust. If the imputation fails, you'll need to supply the correct types yourself.
If TRUE, the input of the first row will be used as column names and not included in the data frame
Most important argument: the path to the file to read
Maximum number of records to read.
Character vector of strings to use for missing values. Set this option to character() to indicate no missing values.
Number of lines to skip before reading data.

```
read_csv(file, col_names = TRUE, col_types = NULL,
  locale = default_locale(), na = c("", "NA"), quoted_na = TRUE,
  quote = "\"", comment = "", trim_ws = TRUE, skip = 0, n_max = Inf,
  guess_max = min(1000, n_max), progress = show_progress())
```

Output

`write_csv(x, path)`	Write to CSV file.
`write_excel_csv(x, path)`	Write to CSV for Excel.
`write_tsv(x, path)`	Write to tab-delimited file.
`write_delim(x, path)`	Write to file with arbitrary delimiter.
`write_file(x, path)`	Write string to file.
`write_lines(x, path)`	Write string vector to file, one element per line.
`write_rds(x, path)`	Write to RDS file.

Data manipulation

Base functions

The base functions are mostly vector-specific but some can be applied to other objects:

Arrange elements

`rev(x)`	Reverse the elements of *x*.
`sort(x)`	Sort the elements of *x* in increasing order; to sort in decreasing order: `rev(sort(x))`
`order(x)`	Return the position of each element if sorted in ascending (descending) order.

NA handling

`na.omit(x)`	Suppress the observations with missing data (NA); if *x* is a matrix or data frame entire row is suppressed.
`na.fail(x)`	Returns an error message if *x* contains at least one NA.

Get position

`which(x)`	Return a vector of the indices of all TRUE elements in the logical vector *x*; *x* can be a comparison operation (e.g `which(1:5 > 2)`).
`which.max(x)`	Return the index of the greatest element of *x*.
`which.min(x)`	Return the index of the smallest element of *x*.
`match(x, y)`	Return a vector of the same length than *x* with the positions of (first) matches of *x* in *y* (NA otherwise).

Modify objects

`unique(x)`	Return a vector, data frame or array like *x* but with duplicate elements/rows removed.
`sample(x, size)`	Take a sample of the specified *size* from the elements of *x* using either with or without replacement.
`expand.grid()`	Create a data frame from all combinations of the supplied vectors or factors.
`split(x, f)`	Divide the data in the vector *x* into the groups defined by *f*.
`unsplit(value,f)`	Reverse the effect of `split()` and put elements or rows in a list of vectors or data frames back in the positions given by *f*.

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Summarize & Aggregate

`table(x)`	Return a contingency table of the counts at each combination of factor levels or integer values.
`prop.table(x,margin)`	Return table entries (vector, matrix or array) as fraction of marginal table; if *x* is a vector, `x/sum(x)` is returned, if *x* is matrix *margin* can be specified (1=row sums, 2= column sums).
`aggregate(x,by, FUN)`	Split the data into subsets, computes summary statistics for each (specified in *FUN*), and returns the result in a convenient form.

Matrix algebra

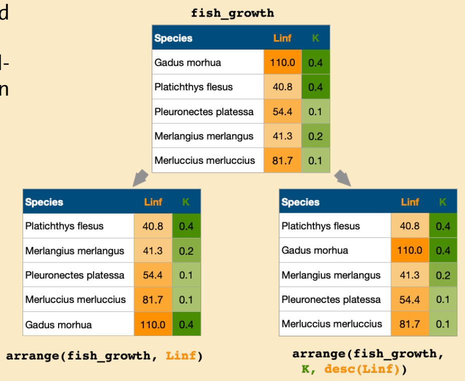
`t(x)`	Transpose *x*
`diag(x)`	Construct a diagonal matrix.
`solve(a,b)`	Solve `a %*% x = b` for *x*.
! for *x*.	
! for *x*.	
`solve(a)`	Return the matrix inverse of *a*.

Tidyverse functions

`load(tibble)` and `load(tidyr)` and `load(dplyr)` or `load(tidyverse)`
The tidyverse functions take data frames or tibbles as table input and generally return tibbles as output.

Arrange tables

`arrange(.data,..)`	Order rows by the specified variables.
`desc(x)`	Sort a vector in descending order; useful within `arrange()`.



fish_growth

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4
Pleuronectes platessa	54.4	0.1
Merlangius merlangus	41.3	0.2
Merluccius merluccius	81.7	0.1

Species	Linf	K
Platichthys flesus	40.8	0.4
Merlangius merlangus	41.3	0.2
Pleuronectes platessa	54.4	0.1
Merluccius merluccius	81.7	0.1
Gadus morhua	110.0	0.4

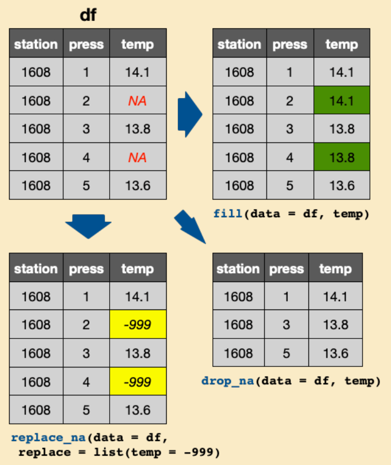
`arrange(fish_growth, Linf)`

Species	Linf	K
Platichthys flesus	40.8	0.4
Gadus morhua	110.0	0.4
Merlangius merlangus	41.3	0.2
Pleuronectes platessa	54.4	0.1
Merluccius merluccius	81.7	0.1

`arrange(fish_growth, K, desc(Linf))`

NA handling

`drop_na(data, ...)`	Drop entire rows containing missing values.
`fill(data, ..., .direction)`	Fill missing values in using e.g. the previous (direction="down") or following (direction="up") entry. Useful if only values that change are recorded.
`replace_na(data, replace = list(),..)`	Replace missing values with a value specified for each column.



df

station	press	temp
1608	1	14.1
1608	2	*NA*
1608	3	13.8
1608	4	*NA*
1608	5	13.6

station	press	temp
1608	1	14.1
1608	2	14.1
1608	3	13.8
1608	4	13.8
1608	5	13.6

`fill(data = df, temp)`

station	press	temp
1608	1	14.1
1608	2	*-999*
1608	3	13.8
1608	4	*-999*
1608	5	13.6

```
replace_na(data = df,
 replace = list(temp = -999)
```

station	press	temp
1608	1	14.1
1608	3	13.8
1608	5	13.6

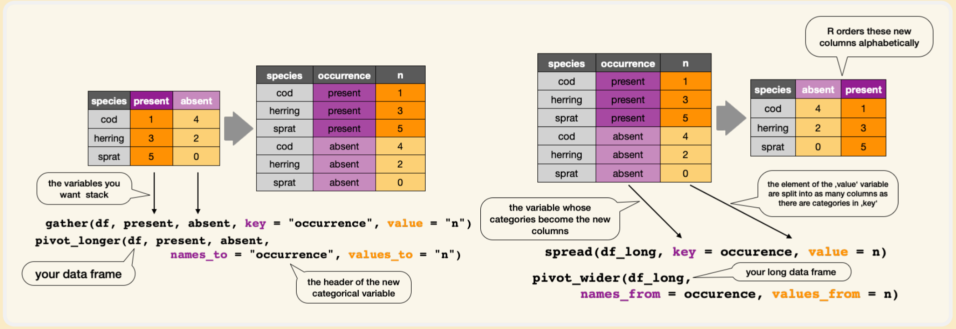
`drop_na(data = df, temp)`

Modify tables

Switch between long and wide format

(*gather* and *spread* got replaced with the more flexible *pivot_longer* and *pivot_wider* functions.)

`pivot_longer(.data, cols, names_to, values_to)`	Pivot data from wide to long → replaces the older function *gather()*.
`pivot_wider(.data, names_from, values_from)`	Pivot data from long to wide → replaces the older function *spread()*.



R orders these new columns alphabetically
the variables you want stack
your data frame
the header of the new categorical variable
the variable whose categories become the new columns
the element of the ‚value‘ variable are split into as many columns as there are categories in ‚key‘
your long data frame

species	present	absent
cod	1	4
herring	3	2
sprat	5	0

species	occurrence	n
cod	present	1
herring	present	3
sprat	present	5
cod	absent	4
herring	absent	2
sprat	absent	0

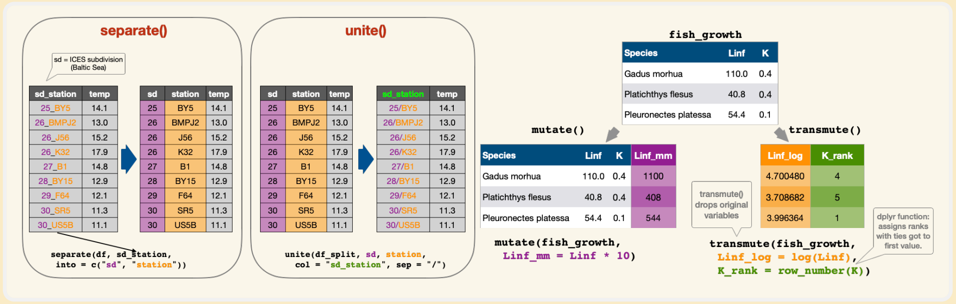
```
gather(df, present, absent, key = "occurrence", value = "n")
pivot_longer(df, present, absent,
             names_to = "occurrence", values_to = "n")
```

species	absent	present
cod	4	1
herring	2	3
sprat	0	5

```
spread(df_long, key = occurence, value = n)
pivot_wider(df_long,
      names_from = occurence, values_from = n)
```

Modify columns and rows

`separate(data, col, into, sep)`	Separate a character column into multiple columns using a regular expression separator.
`separate_row(data,.., sep)`	Separate a collapsed column into multiple rows.
`unite(data,  col,.., sep)`	Unite multiple columns into one by pasting strings together.
`add_column(data,..)`	Add new column(s) (tibble).
`add_row(data,..)`	Add one or more rows to a table.
`rename(.data, newname=oldname)`	Rename columns.
`mutate(.data,..)`	Add new variables while preserving existing ones (dplyr).
`transmute(.data,..)`	Add new variables while dropping existing ones.



separate()
unite()
sd = ICES subdivision (Baltic Sea)
mutate()
transmute()
transmute() drops original variables
dplyr function: assigns ranks with ties got to first value.

sd_station	temp
25_BY5	14.1
26_BMPJ2	13.0
26_J56	15.2
26_K32	17.9
27_B1	14.8
28_BY15	12.9
29_F64	12.1
30_SR5	11.3
30_US5B	11.1

sd	station	temp
25	BY5	14.1
26	BMPJ2	13.0
26	J56	15.2
26	K32	17.9
27	B1	14.8
28	BY15	12.9
29	F64	12.1
30	SR5	11.3
30	US5B	11.1

```
separate(df, sd_station,
 into = c("sd", "station"))
```

sd	station	temp
25	BY5	14.1
26	BMPJ2	13.0
26	J56	15.2
26	K32	17.9
27	B1	14.8
28	BY15	12.9
29	F64	12.1
30	SR5	11.3
30	US5B	11.1

sd_station	temp
25/BY5	14.1
26/BMPJ2	13.0
26/J56	15.2
26/K32	17.9
27/B1	14.8
28/BY15	12.9
29/F64	12.1
30/SR5	11.3
30/US5B	11.1

```
unite(df_split, sd, station,
 col = "sd_station", sep = "/")
```

fish_growth

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4
Pleuronectes platessa	54.4	0.1

Species	Linf	K	Linf_mm
Gadus morhua	110.0	0.4	1100
Platichthys flesus	40.8	0.4	408
Pleuronectes platessa	54.4	0.1	544

```
mutate(fish_growth,
 Linf_mm = Linf * 10)
```

Linf_log	K_rank
4.700480	4
3.708682	5
3.996364	1

```
transmute(fish_growth,
 Linf_log = log(Linf),
 K_rank = row_number(K))
```

Expand and combine tables

Expand tables

`expand_grid()`	Create a tibble from all combinations of the supplied vectors, matrices or data frames (similar to the base function *expand.grid()*.
`complete(data, fill=list())`	Complete a data frame with missing combinations of data (keeps all variables).
`expand(data,..)`	Expand a data frame to include all possible combinations of values present in the specified variables (removing all other variables).

Combine rows

`bind_rows(..)`	Bind multiple tables below each other as they are (by row).
`intersect(x,y,..)`	Bind rows that appear in both *x* AND *y*.
`union(x,y,..)`	Bind rows that appear in *x* OR *y* (duplicates removed).
`setdiff(x,y,..)`	Bind rows that appear in *x* but NOT in *y*.

Combine columns

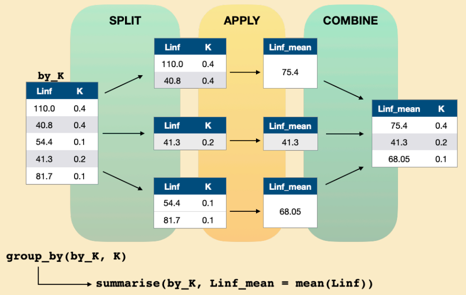
`bind_cols(..)`	Bind multiple tables beside each other as they are (by column).
`left_join(x,y,by)`	Merge table *y* into table *x* by common column values (specified in `by` argument). Rows in *x* with no match in *y* will have NA values in the new columns. If there are multiple matches between *x* and *y*, all combinations of the matches are returned.
`right_join(x,y,by)`	Similar to *left_join()* but merges table *x* into table *y*.
`full_join(x,y,by)`	Join table *x* and *y* while retaining all rows and all columns from both tables. When values do not match in specified variable NAs returned for new columns.
`inner_join(x,y,by)`	Join table *x* and *y* but only those rows where values match in both tables.

Reduce and expand to list-columns

`nest(.data,..)`	Nesting creates a list-column of data frames.
`unnest(data, cols,..)`	Flatten the nested list-column back out into regular columns.

Summarise & Aggregate

`group_by(.data,..)`	Group by one or more variables; takes an existing tbl and converts it into a grouped tbl where operations are performed "by group".
`ungroup(x,..)`	Remove grouping.
`summarise(.data,..)`	Create one or more scalar variables summarizing the variables of an existing tbl based on the specified functions (e.g. *mean*, *min*, *max*).
`count(x,..)`	Count the unique values of one or more variables (if grouped). A shorthand notation for `summarise(.., n=n())`
`distinct(.data,..)`	Select only unique or distinct rows from a tbl. While `unique(x)` works for vectors, `distinct()` works for an entire tbl.



fish_growth

Species	Linf	K
Gadus morhua	110.0	0.4
Platichthys flesus	40.8	0.4
Pleuronectes platessa	54.4	0.1
Merlangius merlangus	41.3	0.2
Merluccius merluccius	81.7	0.1

Linf_mean	Linf_min	Linf_max	K_mean
65.64	40.8	110	0.24

```
summarise(fish_growth,
  Linf_mean = mean(Linf),
  Linf_min = min(Linf),
  Linf_max = max(Linf),
  K_mean = mean(K)
)
```

Useful summarise functions

The following base and dplyr functions return single elements and can be used with the base function `aggregate()` or the dplyr function `summarise()`.

Center:	`mean()`, `median()`	Position:	`first()`, `last()`, `nth()` (all dplyr)
Spread:	`var()`, `sd()`, `IQR()`, `mad()`	Count:	`n()`, `n_distinct()` (all dplyr)
Range:	`min()`, `max()`, `quantile()`	Logical:	`any()`, `all()`