

Erweitertes Datenmanagement in R im tidyverse

Prof. Dr. Rainer Stollhoff

Vgl.
R for Data Science, Grolemund & Wickham,
<http://r4ds.had.co.nz/exploratory-data-analysis.html>

Agenda

- **Daten einlesen**
- Daten transformieren

- Die folgenden Übersichten und Dokumentation stammen aus:
 - Data import with the tidyverse : : **CHEAT SHEET**, Posit Software, PBC, <https://posit.co/wp-content/uploads/2022/10/data-import.pdf>
 - Data tidying with tidyr : : **CHEAT SHEET**, Posit Software, PBC, <https://posit.co/wp-content/uploads/2022/10/tidyr.pdf>
 - Data transformation with dplyr : : **CHEAT SHEET**, Posit Software, PBC, <https://posit.co/wp-content/uploads/2022/10/data-transformation-1.pdf>
- Es wurden u.A. die folgenden Packages genutzt:
 - tidyr: <https://CRAN.R-project.org/package=tidyr>
 - dplyr: <https://CRAN.R-project.org/package=dplyr>

read_* tibble



Daten einlesen

Environment **Files** **Plots** **Help**

Import Dataset

Global Data

mpg_d

set_1

From Text (base)...

From Text (readr)...

From Excel...

4

61

Import Dataset

Name: btw17_erststimmen

Input File

Encoding: Automatic

Heading: ☐ Yes ☒ No

Row names: Automatic

Separator: Comma

Decimal: Period

Quote: Double quote (")

Comment: None

na.strings: NA

☒ Strings as factors

Data Frame

V1 V2

Nr Gebiet

1 Flensburg - Schleswig

2 Nordfriesland - Dithmarschen Nord

3 Steinburg - Dithmarschen Süd

4 Rendsburg-Eckernförde

5 Kiel

6 Plön - Neumünster

7 Pinneberg

8 Segeberg - Stormarn-Mitte

9 Ostholstein - Stormarn-Nord

10 Herzogtum Lauenburg - Stormarn-Süd

11 Lübeck

12 Schwerin - Ludwigslust-Parchim I - Nordwestmecklenburg I

13 Ludwigslust-Parchim II - Nordwestmecklenburg II - Landkr

14 Rostock - Landkreis Rostock II

15 Vorpommern-Rügen - Vorpommern-Greifswald I

16 Mecklenburgische Seenplatte I - Vorpommern-Greifswald II

17 Mecklenburgische Seenplatte II - Landkreis Rostock III

Import Cancel

Import Text Data

File/Url: ~/Documents/Sync/THW/18-19WS/MaschinellesLernen/R_Programmierung/Credit.csv

Browse...

Data Preview:

Income (double)	Limit (integer)	Rating (integer)	Cards (integer)	Age (integer)	Education (integer)	Gender (character)	Student (character)	Married (character)	Ethnicity (character)	Balance (integer)
14.891	3606	283	2	34	11	Male	No	Yes	Caucasian	333
106.025	6645	483	3	82	15	Female	Yes	Yes	Asian	903
104.593	7075	514	4	71	11	Male	No	No	Asian	580
148.924	9504	681	3	36	11	Female	No	No	Asian	964
55.882	4897	357	2	68	16	Male	No	Yes	Caucasian	331
80.180	8047	569	4	77	10	Male	No	No	Caucasian	1151
20.996	3388	259	2	37	12	Female	No	No	African American	203
71.408	7114	512	2	87	9	Male	No	No	Asian	872
15.125	3300	266	5	66	13	Female	No	No	Caucasian	279

Previewing first 50 entries.

Import Options:

Name: Credit

Skip: 0

☒ First Row as Names

☒ Trim Spaces

☒ Open Data Viewer

Delimiter: Comma

Quotes: Default

Locale: Configure...

Escape: None

Comment: Default

NA: Default

Code Preview:

```
library(readr)
Credit <- read_csv("R_Programmierung/Credit.csv")
View(Credit)
```

Import Cancel

Reading rectangular data using readr

Daten aufräumen



table1

```
#> # A tibble: 6 x 4
```

```
#>   country      year  cases population
```

```
#>   <chr>      <int> <int>      <int>
```

```
#> 1 Afghanistan 1999    745    19987071
```

```
#> 2 Afghanistan 2000   2666    20595360
```

```
#> 3 Brazil      1999   37737   172006362
```

```
#> 4 Brazil      2000   80488   174504898
```

```
#> 5 China       1999  212258  1272915272
```

```
#> 6 China       2000  213766  1280428583
```

country, year, cases, population



```
table2
```

```
#> # A tibble: 12 x 4
```

```
#>   country      year type      count
```

```
#>   <chr>      <int> <chr>    <int>
```

```
#> 1 Afghanistan 1999 cases      745
```

```
#> 2 Afghanistan 1999 population 19987071
```

```
#> 3 Afghanistan 2000 cases      2666
```

```
#> 4 Afghanistan 2000 population 20595360
```

```
#> 5 Brazil      1999 cases      37737
```

```
#> 6 Brazil      1999 population 172006362
```

```
#> # ... with 6 more rows
```

country, year, cases, population



```
table3
```

```
#> # A tibble: 6 x 3
```

```
#>   country      year rate
```

```
#> * <chr>      <int> <chr>
```

```
#> 1 Afghanistan 1999 745/19987071
```

```
#> 2 Afghanistan 2000 2666/20595360
```

```
#> 3 Brazil       1999 37737/172006362
```

```
#> 4 Brazil       2000 80488/174504898
```


```
#> 5 China        1999 212258/1272915272
```

```
#> 6 China        2000 213766/1280428583
```

rate = cases/population

country, year, cases, population

In a tidy
data set:



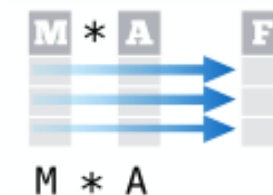
Each **variable** is saved
in its own **column**

&



Each **observation** is
saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



```
table4a # cases
#> # A tibble: 3 x 3
#>   country    `1999` `2000`
#> * <chr>      <int>  <int>
#> 1 Afghanistan    745    2666
#> 2 Brazil        37737   80488
#> 3 China         212258  213766

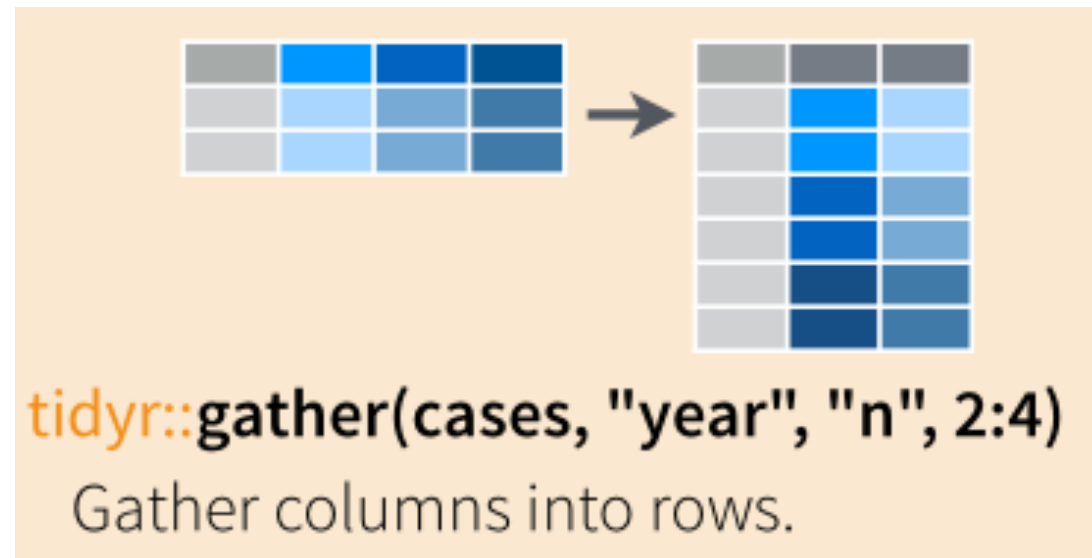
table4b # population
#> # A tibble: 3 x 3
#>   country    `1999`    `2000`
#> * <chr>      <int>      <int>
#> 1 Afghanistan 19987071  20595360
#> 2 Brazil      172006362  174504898
#> 3 China       1272915272 1280428583
```

Tidy up data

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4



Tidy up data

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583



tidyr::spread(pollution, size, amount)

Spread rows into columns.

Tidy up data



```
tidyr::separate(storms, date, c("y", "m", "d"))
```

Separate one column into several.

country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

table3

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Tidy up data



tidyr::unite(data, col, ..., sep)

Unite several columns into one.

country	year	rate
Afghanistan	1999	745 / 19987071
Afghanistan	2000	2666 / 20595360
Brazil	1999	37737 / 172006362
Brazil	2000	80488 / 174504898
China	1999	212258 / 1272915272
China	2000	213766 / 1280428583

country	century	year	rate
Afghanistan	19	99	745 / 19987071
Afghanistan	20	0	2666 / 20595360
Brazil	19	99	37737 / 172006362
Brazil	20	0	80488 / 174504898
China	19	99	212258 / 1272915272
China	20	0	213766 / 1280428583

table6

Combine Data Sets

a			b		
x1	x2		x1	x3	
A	1	+	A	T	=
B	2		B	F	
C	3		D	T	

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::left_join(a, b, by = "x1")
Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::right_join(a, b, by = "x1")
Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::inner_join(a, b, by = "x1")
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::full_join(a, b, by = "x1")
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

dplyr::semi_join(a, b, by = "x1")
All rows in a that have a match in b.

x1	x2
C	3

dplyr::anti_join(a, b, by = "x1")
All rows in a that do not have a match in b.

y			z		
x1	x2		x1	x2	
A	1	+	B	2	=
B	2		C	3	
C	3		D	4	

Set Operations

x1	x2
B	2
C	3

dplyr::intersect(y, z)
Rows that appear in both y and z.

x1	x2
A	1
B	2
C	3
D	4

dplyr::union(y, z)
Rows that appear in either or both y and z.

x1	x2
A	1

dplyr::setdiff(y, z)
Rows that appear in y but not z.

Binding

x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

dplyr::bind_rows(y, z)
Append z to y as new rows.

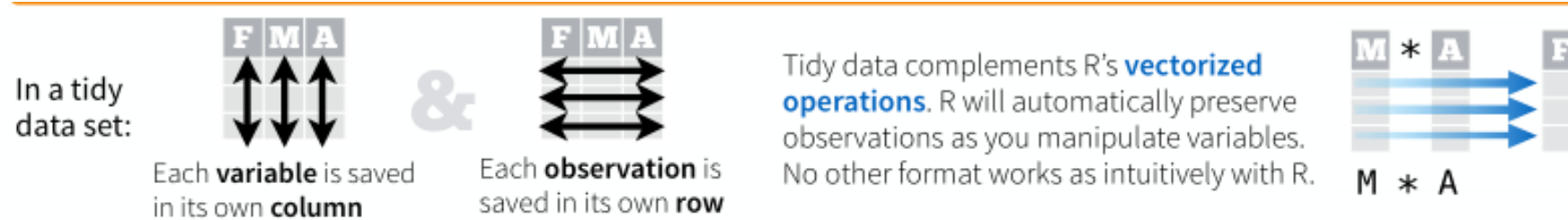
x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

dplyr::bind_cols(y, z)
Append z to y as new columns.
Caution: matches rows by position.

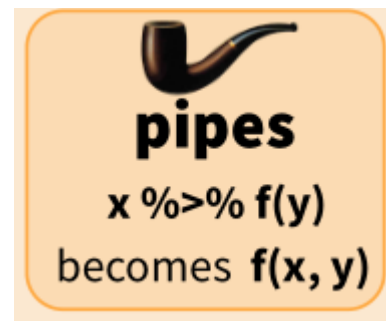
Agenda

- Daten einlesen
- **Daten transformieren**

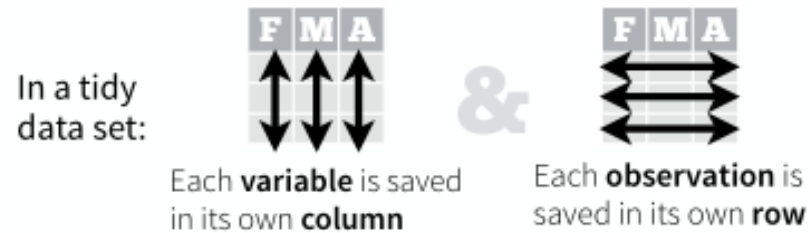
tidyr Data



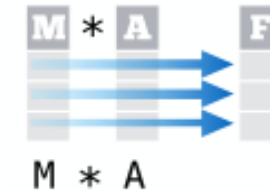
and piping operations



Tibble Datensätze in R



Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



R-exploDA-tidyr.R* x mpg x View (mpg)

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
1	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
2	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
3	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
4	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
5	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact
6	audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact
7	audi	a4	3.1	2008	6	auto(av)	f	18	27	p	compact
8	audi	a4 quattro	1.8	1999	4	manual(m5)	4	18	26	p	compact
9	audi	a4 quattro	1.8	1999	4	auto(l5)	4	16	25	p	compact
10	audi	a4 quattro	2.0	2008	4	manual(m6)	4	20	28	p	compact
11	audi	a4 quattro	2.0	2008	4	auto(s6)	4	19	27	p	compact
12	audi	a4 quattro	2.8	1999	6	auto(l5)	4	15	25	p	compact
13	audi	a4 quattro	2.8	1999	6	manual(m5)	4	17	25	p	compact
14	audi	a4 quattro	3.1	2008	6	auto(s6)	4	17	25	p	compact
15	audi	a4 quattro	3.1	2008	6	manual(m6)	4	15	25	p	compact
16	audi	a6 quattro	2.8	1999	6	auto(l5)	4	15	24	p	midsize
17	audi	a6 quattro	3.1	2008	6	auto(s6)	4	17	25	p	midsize

str (mpg)

```
> str(mpg)
Classes 'tbl_df', 'tbl' and 'data.frame':    234 obs. of  11 variables:
 $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
 $ model       : chr  "a4" "a4" "a4" "a4" ...
 $ displ      : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
 $ year       : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
 $ cyl        : int   4 4 4 4 6 6 6 4 4 4 ...
 $ trans      : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
 $ drv        : chr  "f" "f" "f" "f" ...
 $ cty        : int  18 21 20 21 16 18 18 18 16 20 ...
 $ hwy        : int  29 29 31 30 26 26 27 26 25 28 ...
 $ fl         : chr  "p" "p" "p" "p" ...
 $ class      : chr  "compact" "compact" "compact" "compact" ...
```

Subset Observations (Rows)



```
dplyr::filter(iris. Sepal.Length > 7)
```

```
> ## Alle Hondas
```

```
> filter(.data=mpg,manufacturer=="honda")
```

```
dplyr::dis # A tibble: 9 x 11
```

```
Remove manufacturer model displ year cyl trans drv cty hwy fl class
      <chr>      <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
```

```
dplyr::san 1 honda civic 1.6 1999 4 manual(m5) f 28 33 r subcompact
```

```
Random 2 honda civic 1.6 1999 4 auto(14) f 24 32 r subcompact
```

```
dplyr::san 3 honda civic 1.6 1999 4 manual(m5) f 25 32 r subcompact
```

```
Random 4 honda civic 1.6 1999 4 manual(m5) f 23 29 p subcompact
```

```
Random 5 honda civic 1.6 1999 4 auto(14) f 24 32 r subcompact
```

```
dplyr::slic 6 honda civic 1.8 2008 4 manual(m5) f 26 34 r subcompact
```

```
Select ro 7 honda civic 1.8 2008 4 auto(15) f 25 36 r subcompact
```

```
dplyr::top 8 honda civic 1.8 2008 4 auto(15) f 24 36 c subcompact
```

```
Select ar 9 honda civic 2 2008 4 manual(m6) f 21 29 p subcompact
```

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, , !, xor, any, all	Boolean operators

Subset Variables (Columns)



dplyr::select(iris, Sepal.Width, Petal.Length, Species)

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, contains("."))

Select columns whose name contains a character string.

select(iris, ends_with("Length"))

Select columns whose name ends with a character string.

select(iris, everything())

Select every column.

select(iris, matches(".t."))

Select columns whose name matches a regular expression.

select(iris, num_range("x", 1:5))

Select columns named x1, x2, x3, x4, x5.

select(iris, one_of(c("Species", "Genus")))

Select columns whose names are in a group of names.

select(iris, starts_with("Sepal"))

Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)

Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)


Select all columns except Species.

```
> ## Verbrauchsvariablen
> select(.data = mpg, cty, hwy)
# A tibble: 234 x 2
   cty    hwy
  <int> <int>
1    18    29
2    21    29
3    20    31
4    21    30
5    16    26
6    18    26
7    18    27
8    18    26
9    16    25
10   20    28
# ... with 224 more rows
> |
```

```
> ## Die ersten 3
> select(.data = mpg, 1:3)
# A tibble: 234 x 3
  manufacturer model      displ
  <chr>         <chr>    <dbl>
1 audi         a4         1.8
2 audi         a4         1.8
3 audi         a4         2
4 audi         a4         2
5 audi         a4         2.8
6 audi         a4         2.8
7 audi         a4         3.1
8 audi         a4 quattro 1.8
9 audi         a4 quattro 1.8
10 audi         a4 quattro 2
# ... with 224 more rows
> |
```

Sortieren mit arrange ()

Arrange Cases



arrange(.data, ...)

Order rows by values of a column (low to high),
use with **desc()** to order from high to low.

arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

```
> ## Wer hat die niedrigste Reichweite?
```

```
> arrange(mpg, cty)
```

```
# A tibble: 234 x 11
```

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
	<chr>	<chr>	<dbl>	<int>	<int>	<chr>	<chr>	<int>	<int>	<chr>	<chr>
1	dodge	dakota pickup 4wd	4.7	2008	8	auto(15)	4	9	12	e	pick~
2	dodge	durango 4wd	4.7	2008	8	auto(15)	4	9	12	e	suv
3	dodge	ram 1500 pickup 4wd	4.7	2008	8	auto(15)	4	9	12	e	pick~
4	dodge	ram 1500 pickup 4wd	4.7	2008	8	manual(m6)	4	9	12	e	pick~
5	jeep	grand cherokee 4wd	4.7	2008	8	auto(15)	4	9	12	e	suv
6	chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(14)	r	11	15	e	suv
7	chevrolet	k1500 tahoe 4wd	5.3	2008	8	auto(14)	4	11	14	e	suv
8	chevrolet	k1500 tahoe 4wd	5.7	1999	8	auto(14)	4	11	15	r	suv
9	dodge	caravan 2wd	3.3	2008	6	auto(14)	f	11	17	e	mini~
10	dodge	dakota pickup 4wd	5.2	1999	8	manual(m5)	4	11	17	r	pick~

```
# ... with 224 more rows
```

```
> ## Und wer die höchste?
```

```
> arrange(mpg, desc(cty))
```

```
# A tibble: 234 x 11
```

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
	<chr>	<chr>	<dbl>	<int>	<int>	<chr>	<chr>	<int>	<int>	<chr>	<chr>
1	volkswagen	new beetle	1.9	1999	4	manual(m5)	f	35	44	d	subcompact
2	volkswagen	jetta	1.9	1999	4	manual(m5)	f	33	44	d	compact
3	volkswagen	new beetle	1.9	1999	4	auto(14)	f	29	41	d	subcompact
4	honda	civic	1.6	1999	4	manual(m5)	f	28	33	r	subcompact
5	toyota	corolla	1.8	2008	4	manual(m5)	f	28	37	r	compact
6	honda	civic	1.8	2008	4	manual(m5)	f	26	34	r	subcompact
7	toyota	corolla	1.8	1999	4	manual(m5)	f	26	35	r	compact
8	toyota	corolla	1.8	2008	4	auto(14)	f	26	35	r	compact
9	honda	civic	1.6	1999	4	manual(m5)	f	25	32	r	subcompact
10	honda	civic	1.8	2008	4	auto(15)	f	25	36	r	subcompact

```
# ... with 224 more rows
```

Statistiken mit summarise()

Summarise Data



dplyr::summarise(iris, avg = mean(Sepal.Length))

Summarise data into single row of values.

dplyr::summarise_each(iris, funs(mean))

Apply summary function to each column.

dplyr::count(iris, Species, wt = Sepal.Length)

Count number of rows with each unique value of variable (with or without weights).



Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

dplyr::first

First value of a vector.

dplyr::last

Last value of a vector.

dplyr::nth

Nth value of a vector.

dplyr::n

of values in a vector.

dplyr::n_distinct

of distinct values in a vector.

IQR

IQR of a vector.

min

Minimum value in a vector.

max

Maximum value in a vector.

mean

Mean value of a vector.

median

Median value of a vector.

var

Variance of a vector.

sd

Standard deviation of a vector.

```
> summarise(mpg, mean(cty))
# A tibble: 1 x 1
  `mean(cty)`
    <dbl>
1         16.9
```

> ## Wieviele Autos gibt es pro Hersteller?

> count(mpg, manufacturer)

A tibble: 15 x 2

	manufacturer	n
	<chr>	<int>
1	audi	18
2	chevrolet	19
3	dodge	37
4	ford	25
5	honda	9
6	hyundai	14
7	jeep	8
8	land rover	4
9	lincoln	3
10	mercury	4
11	nissan	13
12	pontiac	5
13	subaru	14
14	toyota	34
15	volkswagen	27

Statistiken mit summarise()

Summarise Data



dplyr::summarise(iris, avg = mean(Sepal.Length))

Summarise data into single row of values.

dplyr::summarise_each(iris, funs(mean))

Apply summary function to each column.

dplyr::count(iris, Species, wt = Sepal.Length)

Count number of rows with each unique value of variable (with or without weights).



Summarise uses **summarise()** to take a vector of values and return a single value.

dplyr::first

First value of a vector.

dplyr::last

Last value of a vector.

dplyr::nth

Nth value of a vector.

dplyr::n

of values in a vector.

dplyr::n_distinct

of distinct values in a vector.

IQR

IQR of a vector.

> summarise_each(mpg, funs(mean))
`summarise_each()` is deprecated.
Use `summarise_all()`, `summarise_at()`, or `summarise_if()`.
To map `funs` over all variables, use `summarise_all()`. To map over a subset of variables, use `summarise_at()` or `summarise_if()`.

```
# A tibble: 1 x 11
  manufacturer model displ year
  <dbl> <dbl> <dbl> <dbl>
1      NA      NA  3.47 2004.
```

Warning messages:

1: In mean.default(manufacturer) : argument is not numeric or logical

2: In mean.default(model) : argument is not numeric or logical

Mean value of a vector.

median

Median value of a vector.

var

Variance of a vector.

sd

Standard deviation of a vector.

```
> summarise(mpg, mean(cty))
# A tibble: 1 x 1
  `mean(cty)`
  <dbl>
1      16.9
```

> ## Wieviele Autos gibt es pro Hersteller?

> count(mpg, manufacturer)

```
# A tibble: 15 x 2
  manufacturer n
  <chr> <int>
1 audi 18
2 chevrolet 19
3 dodge 37
4 ford 25
5 honda 9
6 hyundai 14
7 jeep 8
8 land rover 4
9 lincoln 3
10 mercury 4
11 nissan 13
12 pontiac 5
13 subaru 14
14 toyota 34
15 volkswagen 27
```

Gruppenweise Betrachtung

Group Data

`dplyr::group_by(iris, Species)`

Group data into rows with the same value of Species.

`dplyr::ungroup(iris)`

Remove grouping information from data frame.

`iris %>% group_by(Species) %>% summarise(...)`

Compute separate summary row for each group.



```
> ## Minimale, Maximale und durchschnittliche Reichweite je Hersteller  
> summarise(group_by(mpg,manufacturer),min(cty),max(cty),mean(cty))  
# A tibble: 15 x 4
```

	manufacturer	`min(cty)`	`max(cty)`	`mean(cty)`
	<chr>	<dbl>	<dbl>	<dbl>
1	audi	15	21	17.6
2	chevrolet	11	22	15
3	dodge	9	18	13.1
4	ford	11	18	14
5	honda	21	28	24.4
6	hyundai	16	21	18.6
7	jeep	9	17	13.5
8	land rover	11	12	11.5
9	lincoln	11	12	11.3
10	mercury	13	14	13.2
11	nissan	12	23	18.1
12	pontiac	16	18	17
13	subaru	18	21	19.3
14	toyota	11	28	18.5
15	volkswagen	16	35	20.9

Piping Operationen

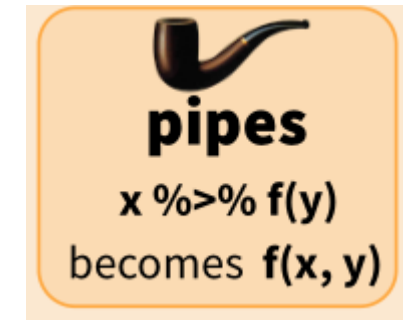
- Mit dem pipe Operator %>% werden die Ergebnisse eines Funktionsaufrufs zum Argument des nächsten Funktionsaufrufs

```
### so nicht - Gruppierungen gehen verloren
group_by(mpg,manufacturer)
summarise(mpg,mean(cty))

### aber mit pipe
group_by(mpg,manufacturer) %>%
summarise(mean(cty))

### entspricht das dem verschachtelten Aufruf
summarise(group_by(mpg,manufacturer),mean(cty))

### und ist besser lesbar
```



x |> f(y) ersetzt f(x,y)


```
mpg %>%
  group_by(manufacturer) %>%
  summarise(hwy_l100km = mean(1/hwy)*4.55/1.61*100) %>%
  arrange(hwy_l100km)
```

```
> ## Mutatis mutandis
> mpg %>%
+   group_by(manufacturer) %>%
+   summarise(hwy_l100km = mean(1/hwy*4.55/1.61*100)) %>%
+   arrange(hwy_l100km)
# A tibble: 15 x 2
  manufacturer hwy_l100km
  <chr>        <dbl>
1 honda        8.73
2 volkswagen   9.91
3 hyundai     10.6
4 pontiac      10.7
5 audi         10.8
6 subaru       11.1
7 nissan       12.0
8 toyota       12.1
9 chevrolet    13.7
10 ford        15.0
11 mercury     15.7
12 dodge       16.4
13 jeep        16.6
14 lincoln     16.7
15 land rover  17.3

> ## ist leider schon weg
> select(mpg,hwy_l100km)
Error in .f(x[[i]], ...) : object 'hwy_l100km' not found
```

Neue Variablen in tibbles

Make New Variables



dplyr::mutate(iris, sepal = Sepal.Length + Sepal.Width)

Compute and append one or more new columns.

dplyr::mutate_each(iris, funs(min_rank))

Apply window function to each column.

dplyr::transmute(iris, sepal = Sepal.Length + Sepal.Width)

Compute one or more new columns. Drop original columns.

```
## so bleibst bestehen|
mpg2 <- mutate(mpg, hwy_l1p100km = 1/hwy*4.55/1.61*100)
mpg2 %>% group_by(manufacturer) %>% summarise(MW=mean(hwy_l1p100km))

> ## so verschwindet der Rest
> mpg2 <- transmute(mpg, hwy_l1p100km = 1/hwy*4.55/1.61*100)
> mpg2
# A tibble: 234 x 1
#   hwy_l1p100km
#   <dbl>
1         9.75
2         9.75
3         9.12
4         9.42
5        10.9
6        10.9
7        10.5
8        10.9
9        11.3
10       10.1
# ...with 224 more rows
```



Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

Make New Variables



`dplyr::mutate(iris, sepal = Sepal.Length + Sepal.Width)`

Compute and append one or more new columns.

`dplyr::mutate_each(iris, funs(min_rank))`

Apply window function to each column.

`dplyr::transmute(iris, sepal = Sepal.Length + Sepal.Width)`

Compute one or more new columns. Drop original columns.

`dplyr::lead`

Copy with values shifted by 1.

`dplyr::lag`

Copy with values lagged by 1.

`dplyr::dense_rank`

Ranks with no gaps.

`dplyr::min_rank`

Ranks. Ties get min rank.

`dplyr::percent_rank`

Ranks rescaled to [0, 1].

`dplyr::row_number`

Ranks. Ties got to first value.

`dplyr::ntile`

Bin vector into n buckets.

`dplyr::between`

Are values between a and b?

`dplyr::cume_dist`

Cumulative distribution.

`dplyr::cumall`

Cumulative **all**

`dplyr::cumany`

Cumulative **any**

`dplyr::cummean`

Cumulative **mean**

`cumsum`

Cumulative **sum**

`cummax`

Cumulative **max**

`cummin`

Cumulative **min**

`cumprod`

Cumulative **prod**

`pmax`

Element-wise **max**

`pmin`

Element-wise **min**