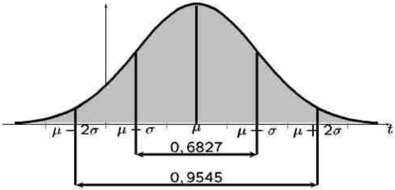
Statistics

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}$$



Bachelor Studiengang Informatik

Prof. Dr. Egbert Falkenberg

Fachbereich Informatik & Ingenieurwissenschaften

Wintersemester 23/24

Some useful R commands

Bachelor Studiengang Informatik

Dr. Egbert Falkenberg

Fachbereich Informatik & Ingenieurwissenschaften

September 2021

Generate Sequences

Data: Conversion, Information, Manipulation

Some Statistical Functions
- Univariate Data
- Bivariate Data
- Descriptive statistics by groups

Making Graphs
- Graphics with the Base Package
- Graphics with ggplot

Random Samples and Permutations

Functions for Probability Distributions

Inferential Statistics

Literature

Generate Sequences

from:to: generates a sequence

```
> 3:10
[1]  3  4  5  6  7  8  9 10
```

c(): generic function which combines its arguments to form a vector. All arguments are coerced to a common type which is the type of the returned value.

```
> c(1,7:9)
[1] 1 7 8 9
> c(1:5, 10.5, "next")
[1] "1"    "2"    "3"    "4"    "5"    "10.5" "next"
```

Generate Sequences

Data: Conversion, Information, Manipulation

Some Statistical Functions
Univariate Data
Frequencies
Bivariate Data
two categorial variables
two continous variables
Descriptive statistics by groups

Making Graphs
Graphics with the Base Package
Graphics with ggplot

Random Samples and Permutations

Functions for Probability Distributions

Inferential Statistics

Literature

Generate Sequences

seq(from,to) seq(from,to,by=), seq(from,to,length.out=):
from, to : starting and (maximal) end values
by number : increment of the sequence
length.out : desired length of the sequence

```
> seq(0, 1, length.out = 11)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(1, 9, by = 2)     # matches 'end'
[1] 1 3 5 7 9           # stay below end
> seq(1, 6, by = 3)
[1] 1 4
> seq(1.5, 2.1, by = 0.1)
[1] 1.5 1.6 1.7 1.8 1.9 2.0 2.1
```

Generate Sequences

rep(x, ...): replicates the values in x
x : a vector or a factor
... : further arguments like
times : an integer-valued vector giving the number of times to repeat each element if of length length(x), or to repeat the whole vector if of length 1.
length.out : desired length of the output vector

```
> rep("abc", times = 3)
[1] "abc" "abc" "abc"
> rep(1:4, times = 2)
[1] 1 2 3 4 1 2 3 4
> rep(1:4,  times = c(2,3,1,2))
[1] 1 1 2 2 2 3 4 4
> rep(c("a","b","c"), length.out = 5)
[1] "a" "b" "c" "a" "b"
> rep(1:4, length.out = 6)
[1] 1 2 3 4 1 2
```

Data: Conversion, Information, Manipulation

as.integer(), as.logical(), as.character(), ...: convert type

```
> x <- pi * c(-1:1, 10)
> x
[1] -3.141593  0.000000  3.141593 31.415927
> x %>% as.integer()
[1] -3  0  3 31
c("-.123","2.7","312.3") %>% as.numeric()
[1]  -0.123   2.700 312.300
> as.numeric(c("-.123","2.7","312.3")) %>% as.character()
[1] "-0.123" "2.7"    "312.3"
```

is.na(): indicates which elements are missing

```
> c("-2","45","AB") %>% as.integer()
[1] -2 45 NA
> c("-2","45","AB") %>% as.integer() %>% is.na()
[1] FALSE FALSE  TRUE
```

Data: Conversion, Information, Manipulation

duplicated(): determines which elements are duplicates of elements with smaller subscripts, and returns a logical vector indicating which elements are duplicates.

```
> x <- c(1:4,seq(1,8,by=2))
> x
[1] 1 2 3 4 1 3 5 7
> x %>% duplicated()
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
```

unique(x): returns x with duplicate elements removed.

```
> c(1:4,seq(1,8,by=2)) %>% unique()
[1] 1 2 3 4 5 7
```

Data: Conversion, Information, Manipulation

rev(x): provides a reversed version of x.

```
x <- c(1:5, 5:3)
> x
[1] 1 2 3 4 5 5 4 3
> rev(x)
[1] 3 4 5 5 4 3 2 1
```

sort(x, decreasing = FALSE): sort (or order) a vector or factor (partially) into ascending or descending order.

```
> x <- c(1:5, 5:3)
> sort(x)
[1] 1 2 3 3 4 4 5 5
> sort(x, decreasing = TRUE)
[1] 5 5 4 4 3 3 2 1
```

Data: Conversion, Information, Manipulation

cut(x, breaks, labels = NULL): divides the range of x into intervals and codes the values in x according to which interval they fall. If labels = NULL the leftmost interval corresponds to level one, the next leftmost to level two and so on.

- breaks : either a numeric vector of two or more unique cut points or a single number giving the number of intervals into which x is to be cut.
- labels : labels for the levels of the resulting category. By default, labels are constructed using “(a,b]” interval notation. If labels = FALSE, simple integer codes are returned instead of a factor.

Data: Conversion, Information, Manipulation

```
x <- c(1,2,3,4,5,2,3,4,5,6,7)
> cut(x, breaks = 2)
 [1] (0.994,4] (0.994,4] (0.994,4] (0.994,4] (4,7.01]
 [6] (0.994,4] (0.994,4] (0.994,4] (4,7.01]  (4,7.01]
[11] (4,7.01]
Levels: (0.994,4] (4,7.01]
> cut(x, breaks = c(0,2,5,8))
 [1] (0,2] (0,2] (2,5] (2,5] (2,5] (0,2] (2,5] (2,5] (2,5]
[10] (5,8] (5,8]
Levels: (0,2] (2,5] (5,8]
> cut(x, breaks = c(0,2,5,8),
+     labels = c("class 1","class 2","class 3"))
 [1] class 1 class 1 class 2 class 2 class 2 class 1 class 2
 [8] class 2 class 2 class 3 class 3
Levels: class 1 class 2 class 3
```

Some Statistical Functions- Univariate Data

In the follwoing we use the following vector to demonstrate the commands:

```
> x <- c(sample(x = 1:5, size = 10, replace = TRUE),
+        NA,NA,sample(x = 3:8, size = 5, replace = TRUE))
> x
 [1]  3  5  4  4  5  5  3  5  1  4 NA NA  6  4  7  8  4
```

`na.rm = TRUE` means that missing values are excluded.

- **max(x, na.rm = TRUE):** find the maximum value in x

```
> max(x)
[1] NA
> max(x, na.rm = TRUE)
[1] 8
```

- **min(x, na.rm=TRUE):** find the minimum value in x

```
> min(x, na.rm = TRUE)
[1] 1
```

- **mean(x, na.rm=TRUE):** find the mean of the values in x

```
> mean(x, na.rm = TRUE)
[1] 4.533333
```

Some Statistical Functions- Univariate Data

- **median(x, na.rm=TRUE):** find the median of the values in x

```
> median(x, na.rm = TRUE)
[1] 4
```

- **sum(x, na.rm=TRUE):** sum all values of x

```
> sum(x, na.rm = TRUE)
[1] 68
```

- **var(x, na.rm=TRUE):** find the variance of the values in x

```
> var(x, na.rm = TRUE)
[1] 2.838095
```

- **sd(x, na.rm=TRUE):** find the stand deviation of the values in x

```
> sd(x, na.rm = TRUE)
[1] 1.684665
```

- **cumsum(x):** cumulative sum of the values in x

```
> x[!is.na(x)]
 [1] 3 5 4 4 5 5 3 5 1 4 6 4 7 8 4
> x[!is.na(x)] %>% cumsum()
 [1]  3  8 12 16 21 26 29 34 35 39 45 49 56 64 68
```

Some Statistical Functions- Univariate Data

- **rank(x):** find the ranks of the values in x

```
> x[!is.na(x)] %>% rank()
 [1]  2.5 10.5  6.0  6.0 10.5 10.5  2.5 10.5  1.0  6.0 13.0
[12]  6.0 14.0 15.0  6.0
```

- **summary(x):** in case of vector of numbers some characteristic number are calculated

```
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA’s
  1.000   4.000   4.000   4.533   5.000   8.000       2
```

- **quantiles(x, probs = seq(0,1,0.25)):** produces sample quantiles corresponding to the given probabilities

```
> x[!is.na(x)] %>% quantile()
  0%  25%  50%  75% 100%
   1    4    4    5    8
> x[!is.na(x)] %>% quantile(probs = c(0.2,0.4,0.6,0.8))
20% 40% 60% 80%
3.8 4.0 5.0 5.2
```

Some Statistical Functions- Univariate Data

In the following we use the following data set:

```
> sample_values
# A tibble: 10 x 2
   X         Y
   <chr> <dbl>
 1 c         1
 2 b         1
 3 c         5
 4 a         0
 5 c         2
....
```

table(x): frequency counts of entries

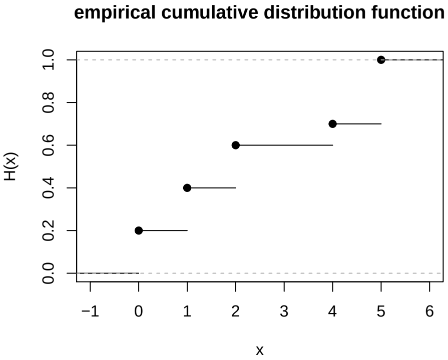
```
> table(sample_values$X)

a b c
3 2 5
> table(sample_values$Y)

0 1 2 4 5
2 2 2 1 3
```

Some Statistical Functions- Univariate Data

ecdf(): compute an empirical cumulative distribution function, with several methods for plotting, printing and computing with such an “ecdf” object.

```
> H <- ecdf(sample_values$Y)
> # values of H at all sample values
> H(sample_values$Y)
 [1] 0.4 0.4 1.0 0.2 0.6 1.0 0.6 1.0 0.2 0.7
> # calculate values of H
> H(c(-1,0.345,4,4.8,6))
[1] 0.0 0.2 0.7 0.7 1.0
> # values of H at the sample values
> H_tab <- sample_values %>% select (Y) %>% rename(x = Y) %>%
+   mutate(H_x = H(x)) %>% unique() %>% arrange(x)
> H_tab
# A tibble: 5 x 2
      x   H_x
  <dbl> <dbl>
1     0   0.2
2     1   0.4
3     2   0.6
4     4   0.7
5     5   1
> # plot.ecdf implements the plot method for ecdf objects,
> plot.ecdf(sample_values$Y, ylab = "H(x)",
+           main = "empirical cumulative distribution function")
```

Some Statistical Functions- Univariate Data



Some Statistical Functions- Bivariate Data

The examples are applied to the data set:

```
> cat_data
# A tibble: 20 x 2
   hair  sex
   <chr> <chr>
 1 blond female
 2 blond female
 3 black male
 4 blond female
 5 blond male
 6 red   male
 7 blond male
.....
```

Some Statistical Functions- Bivariate Data

table(x,y): build a contingency table of the counts at each combination of values.

```
> table(cat_data$hair, cat_data$sex)

        female male
  black      1    5
  blond      5    4
  brown      2    1
  red        0    2
```

addmargins() : puts margins on tables

```
> table(cat_data) %>% addmargins()
       sex
hair    female male Sum
  black      1    5   6
  blond      5    4   9
  brown      2    1   3
  red        0    2   2
  Sum        8   12  20
```

Some Statistical Functions- Bivariate Data

Calculate χ^2 and the indifference table using the chisq.test() function:

- chisq.test performs chi-squared contingency table tests and goodness-of-fit tests.
- The variable statistic resp. expected contains the χ^2-value resp. the indifference table.

```
cat_data %>% table() %>% chisq.test() -> chi.results
chi.results$statistic
> chi.results$statistic
X-squared
 1.760462
chi.results$expected
> chi.results$expected
       sex
hair     female male
  black    2.20 1.80
  blond    3.85 3.15
  brown    2.75 2.25
  red      2.20 1.80
```

Some Statistical Functions- Bivariate Data

The examples are applied to the data set:

```
> cont_data
# A tibble: 20 x 2
       x     y
   <dbl> <dbl>
 1     1     1
 2     8     6
 3     9     9
 4     2     7
 5     4     7
 6     6     8
 7     9     8
 8     6     1
....
```

Some Statistical Functions- Bivariate Data

summary(): gives some important characteristic numbers

```
> summary(cont_data)
       x               y       
 Min.   : 1.00   Min.   :1.00  
 1st Qu.: 3.75   1st Qu.:3.00  
 Median : 6.00   Median :7.00  
 Mean   : 5.95   Mean   :5.55  
 3rd Qu.: 8.25   3rd Qu.:8.00  
 Max.   :10.00   Max.   :9.00  
```

cov(x,y): find the covariance of the variables x, y

```
> cov(cont_data$x,cont_data$y)
[1] 0.45
```

cor(x,y): find the coefficient of correlation of the variables x, y

```
> cor(cont_data$x,cont_data$y)
[1] 0.05379476
```

Some useful R commands

Dr. Egbert Falkenberg

Generate Sequences

Data: Conversion, Information, Manipulation

Some Statistical Functions

Univariate Data

Frequencies

Bivariate Data

two categorial variables

two continous variables

Descriptive statistics by groups

Making Graphs

Graphics with the Base Package

Graphics with ggplot

Random Samples and Permutations

Functions for Probability Distributions

Inferential Statistics

Literature

Some Statistical Functions- Bivariate Data

lm($Y \sim X$): carries out linear regression (Y depends on X), lm returns an object of clas “lm” which contains several components. Some importants are

- coefficients: a named vector of coefficients
- residuals: the residuals, that is response minus fitted values.
- fitted.values: the fitted mean values.

Some Statistical Functions- Bivariate Data

```
> X <- cont_data$x
> Y <- cont_data$y
> reg <- lm(Y ~ X)
> reg

Call:
lm(formula = Y ~ X)

Coefficients:
(Intercept)            X
    5.23392      0.05312

> # coefficients of the regression line
> reg$coefficients
(Intercept)           X
 5.23392358  0.05312209
```

Some Statistical Functions- Bivariate Data

```
> # table with x, y, fitted value and residual
> cont_data %>%
+   mutate(pred = reg$fitted.values,
+          res = reg$residuals)
# A tibble: 20 x 4
       x     y  pred    res
   <dbl> <dbl> <dbl>  <dbl>
 1     1     1  5.29 -4.29
 2     8     6  5.66  0.341
 3     9     9  5.71  3.29
 4     2     7  5.34  1.66
 5     4     7  5.45  1.55
 6     6     8  5.55  2.45
 7     9     8  5.71  2.29
 8     6     1  5.55 -4.55
 9     6     1  5.55 -4.55
10     8     2  5.66 -3.66
11     9     4  5.71 -1.71
12     2     7  5.34  1.66
13     6     7  5.55  1.45
14     3     9  5.39  3.61
15    10     3  5.77 -2.77
16     5     8  5.50  2.50
17     1     3  5.29 -2.29
18     7     7  5.61  1.39
19     9     4  5.71 -1.71
20     8     9  5.66  3.34
```

Descriptive statistics by groups

To compute summary statistics by groups, the functions group_by() and summarise() (in dplyr package) can be used.
Example:: We want to group the built-in R data set named iris by Species and then compute the number of element in each group, the mean and the standard deviation.

```
> iris %>%
+   group_by(Species) %>%
+   summarise(
+     count = n(),
+     mean = mean(Sepal.Length, na.rm = TRUE),
+     sd = sd(Sepal.Length, na.rm = TRUE)
+   )
# A tibble: 3 x 4
  Species    count  mean    sd
  <fct>      <int> <dbl> <dbl>
1 setosa        50  5.01 0.352
2 versicolor    50  5.94 0.516
3 virginica     50  6.59 0.636
```

Making Graphs

There are many ways to create graphics with R. Due to the power of the graphic commands available in R, it is often difficult to find the appropriate commands and options.
With the R standard graphics can be created simply and very quickly. The functions are very powerful and flexible, but the syntax is difficult for beginners to customize graphics themselves. Furthermore the syntax depends on the functions.
In contrast ggplot2 is based on an intuitive syntax called the Grammar of graphics. Once you get used to it you can create very complex graphics with an elegant and consistent “grammar”. ggplot2 is designed to be used with tidy data. It is part of the package tidyverse.
Here a few graphic commands with a few options from the base package and the corrersponding commands applying gglot2 are presented.
Regarding ggplot the R grahics cookbook [4] is very useful. A good overview can be found in chapter 3 of [1]. Especially helpful when creating more complex graphics with the base package is “Cookbook for R Graphs” from Winston Chung [5].

Graphics with the Base Package

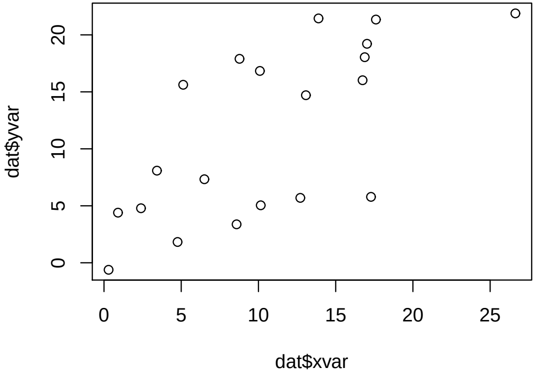
Scatterplots: We want to show a scatterplot of two continous variables.

```
> # Make some noisily increasing data
> dat <-
+   tibble(
+     xvar = 1:20 + rnorm(20,sd=3),
+     yvar = 1:20 + rnorm(20,sd=3)
+   )
> dat
# A tibble: 20 x 2
     xvar    yvar
    <dbl>   <dbl>
 1  4.85   -0.113
 2  2.96    9.18
 3  0.389  -6.32
 4  4.24    0.904
 5  5.10    4.41
...
```

Graphics with the Base Package

plot(x,y) shows a diagramm with all points (x,y):

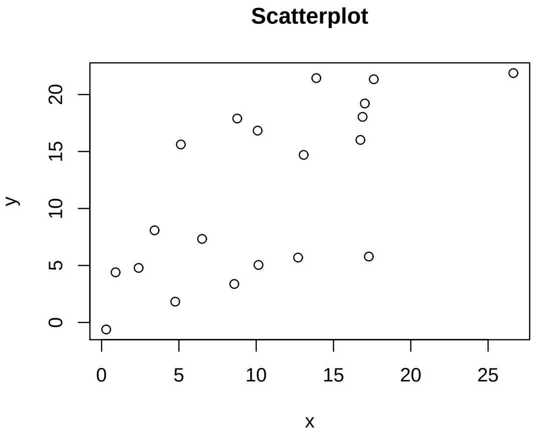
```
> # simple plot
> plot(x = dat$xvar, y = dat$yvar)
```



Graphics with the Base Package

labels of the axis and a title additionally:

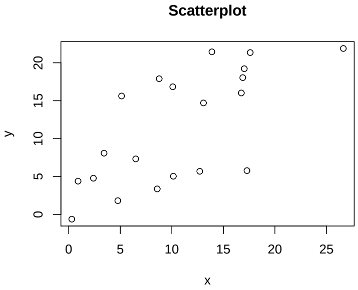
```
> # title and labels of the axis
> plot(x = dat$xvar, y = dat$yvar,
+      xlab = "x", ylab = "y",
+      main = "Scatterplot")
```



Graphics with the Base Package

adding lines using the abline()-command:
abline(a,b) adds a line with intercept a and slope b and abline(lm($y \sim x$)) adds a regression line.

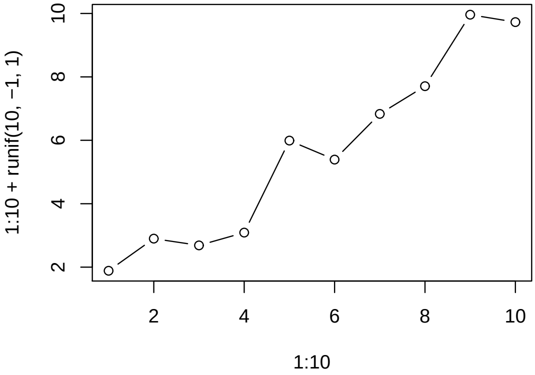
```
> # additional lines
> plot(x = dat$xvar, y = dat$yvar,
+      xlab = "x", ylab = "y",
+      main = "Scatterplot",
+      sub = "red: regression line, blue: y=a+bx")
> abline(lm(dat$yvar ~ dat$xvar), col = "red")
> abline(a = 5, b = 0.6, col = "blue")
```



Graphics with the Base Package

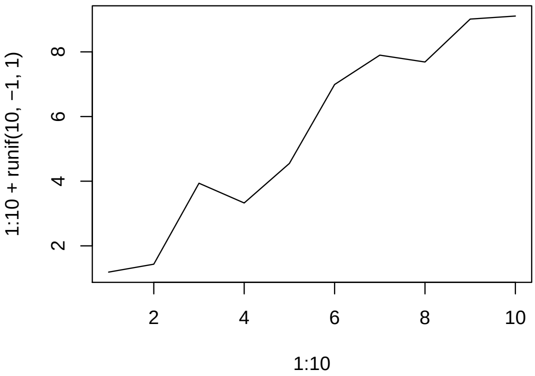
Line Charts: type = “b” and type =l” option of plot():

```
plot(x = 1:10, y = 1:10 + runif(10,-1,1), type = "b")
```



Graphics with the Base Package

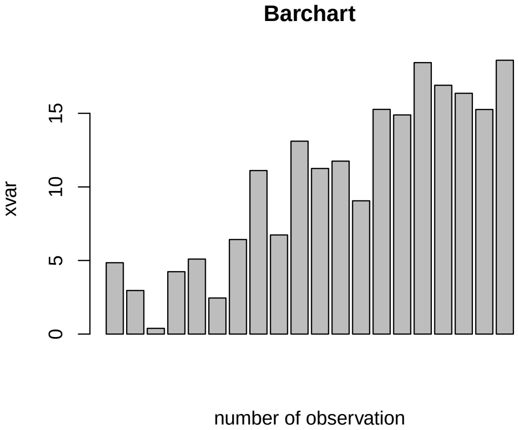
```
plot(x = 1:10, y = 1:10 + runif(10,-1,1), type = "l")
```



Graphics with the Base Package

Barchart: barplot(x) generates a barchart where the height of the bar given by the vector x

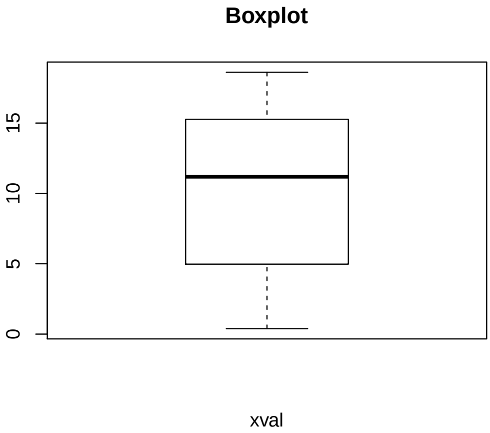
```
barplot(dat$xvar, xlab = "number of observation",
        ylab = "xvar", main = "Barchart")
```



Graphics with the Base Package

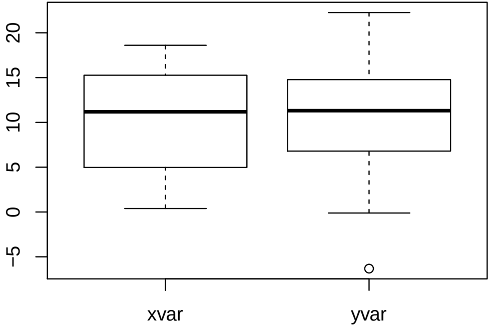
Boxplots: generated using the boxplot()-command

```
boxplot(dat$xvar, xlab = "xval", main = "Boxplot")
```



Graphics with the Base Package

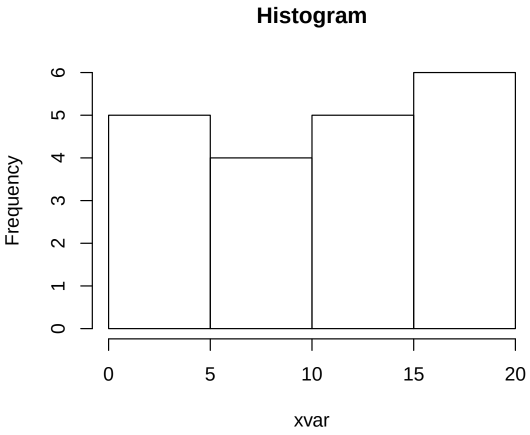
```
boxplot(dat, xlab = "xval", main = "Boxplot of both variables")
```



Graphics with the Base Package

Histogram: The hist()-comannd generates histograms. Usefull ist the option breaks which denotes the bounds of the underlying classes.

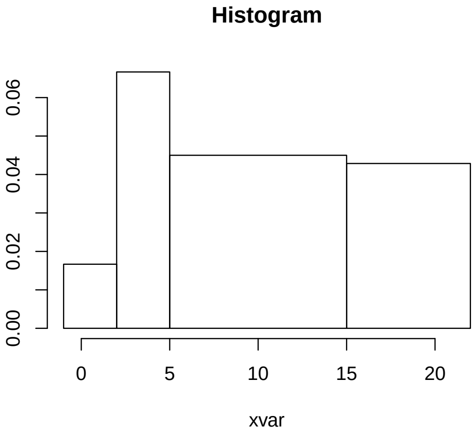
```
hist(dat$xvar, xlab = "xvar", main = "Histogram")
```



Graphics with the Base Package

```
hist(dat$xvar, breaks = c(-1,2,5,15,22),
      xlab = "xvar", ylab = "", main = "Histogram")
```

underlying classes:$(-1, 2], (2, 5], (5, 15], (15, 22]$

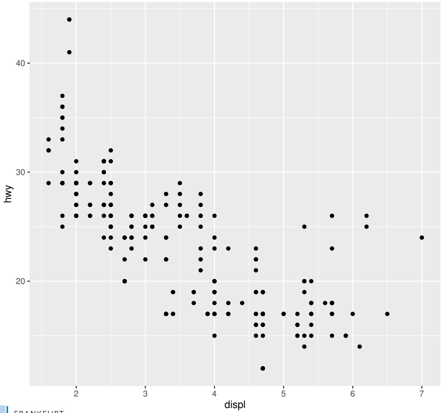


Graphics with ggplot

Source: chapter 3 of [1].

- First example: Do cars with big engines use more fuel than cars with small engines?
- We use the mpg dataset part of ggplot2 package describing fuel economy data from 1999 to 2008 for 38 popular models of cars

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



Graphics with ggplot

- ggplot() creates a coordinate system that you can add layers to.
- The first argument of ggplot() is the dataset to use in the graph. So ggplot(data = mpg) creates an empty graph. You complete your graph by adding one or more layers to ggplot().
- The function geom_point() adds a layer of points to your plot, which creates a scatterplot. ggplot2 comes with many geom functions that each add a different type of layer to a plot. Each geom function in ggplot2 takes a mapping argument, so-called “aesthetic mappings”, i.e. we determine which variables are to be displayed on the X- and Y-axes, and which variables are used to group the data. The function we use for this is called aes().

Graphics with ggplot

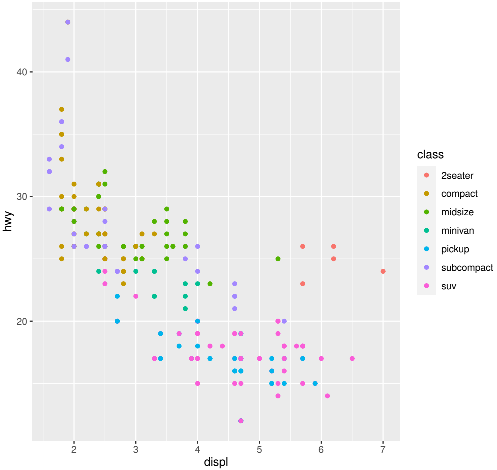
Aesthetics:

- Some of the values in the diagramm are far away from the others. One possible explanation is that hwy depends on the values of the variable class. To visualize the impact of the values of the variable class is mapping colors to the different values of class in the diagram.
- You can add a third variable, like class, to a two dimensional scatterplot by mapping it to an aesthetic. An aesthetic is a visual property of the objects, (for example the size, the shape, or the color) in your plot. You can display a point in different ways by changing the values of its aesthetic properties. We use the word “level” to describe aesthetic properties. We can change the levels of a point’s size, shape, and color to make the point small.

Graphics with ggplot

ggplot2 will automatically assign a unique level of the aesthetic (here a unique color) to each unique value of the variable. ggplot2 will also add a legend that explains which levels correspond to which values.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



Graphics with ggplot

One common problem when creating ggplot2 graphics is to put the + in the wrong place: it has to come at the end of the line, not the start. In other words, make sure you haven¡¯t accidentally written code like this:

```
ggplot(data = mpg)
 + geom_point(mapping = aes(x = displ, y = hwy))
```

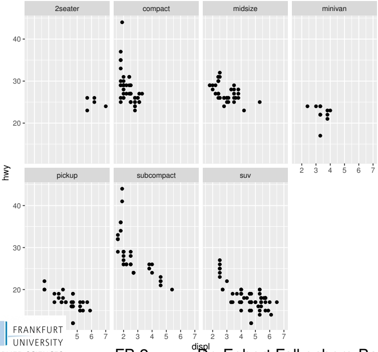
Graphics with ggplot

Facets: One way to add additional variables is with aesthetics. Another way, particularly useful for categorical variables, is to split your plot into facets, subplots that each display one subset of the data.

To facet your plot by a single variable, use facet_wrap().

The first argument of facet_wrap() should be a formula, which you create with ˜followed by a variable name (here “formula” is the name of a data structure in R, not a synonym for “equation”). The variable that you pass to facet_wrap() should be discrete.

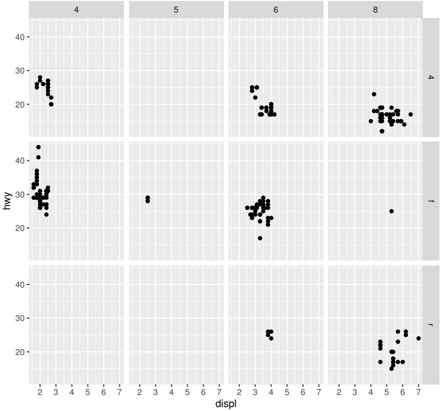
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```



Graphics with ggplot

To facet your plot on the combination of two variables, add facet_grid() to your plot call. The first argument of facet_grid() is also a formula. This time the formula should contain two variable names separated by a ~

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(drv ~ cyl)
```



Graphics with ggplot

- facets depending on drv in rows

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(drv ~ .)
```

- facets depending on cyl in columns

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(. ~ cyl)
```

Graphics with ggplot

Geometric Objects: Different visual objects can be used to represent data. In ggplot2 syntax, we say that they use different geoms.

- A geom is the geometrical object that a plot uses to represent data.
- For example: bar charts use bar geoms, line charts use line geoms, boxplots use boxplot geoms, scatterplots the point geom.
- To change the geom in your plot, change the geom function that you add to ggplot().

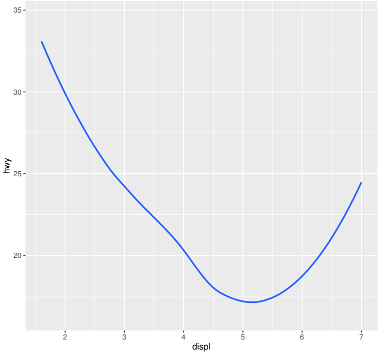
Graphics with ggplot

point geom:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```

smooth geom: a smooth line fitted to the data.

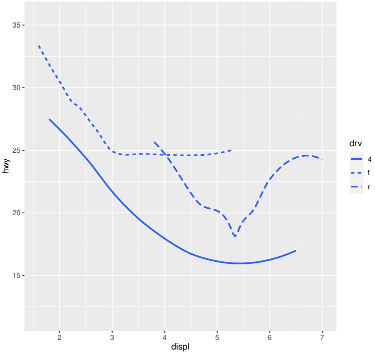
```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```



Making Graphs

Every geom function in ggplot2 takes a mapping argument. However, not every aesthetic works with every geom. You could set the shape of a point, but you couldn¡¯t set the “shape” of a line. On the other hand, you could set the linetype of a line.

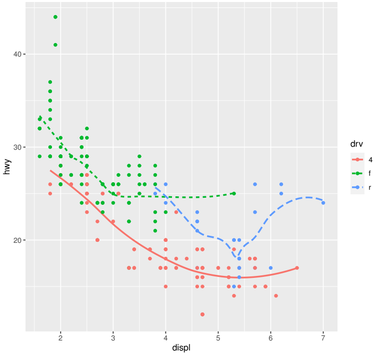
```
ggplot(data = mpg) +
  geom_smooth(mapping =  aes(x = displ, y = hwy, linetype = drv))
```



Making Graphs

To display multiple geoms in the same plot, add multiple geom functions to ggplot()

```
ggplot(data = mpg) +
  geom_smooth(mapping =  aes(x = displ, y = hwy, linetype = drv,  color = drv)) +
  geom_point(mapping = aes(x = displ, y = hwy, color = drv))
```



Making Graphs

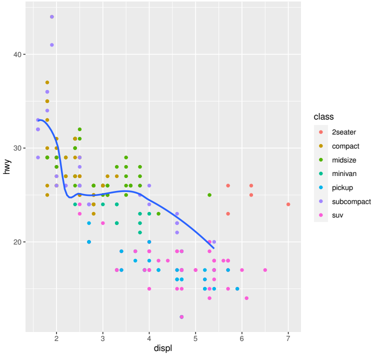
Duplication in the code can be avoided by passing a set of mappings to ggplot(). ggplot2 will treat these mappings as global mappings that apply to each geom in the graph. If you place mappings in a geom function, ggplot2 will treat them as local mappings for the layer.

```
ggplot(data = mpg,mapping =
         aes(x = displ, y = hwy, color = drv)) +
  geom_smooth(mapping = aes(linetype = drv)) +
  geom_point()
```

Making Graphs

You can use the same idea to specify different data for each layer. Here, our smooth line displays just a subset of the mpg dataset, the subcompact cars. The local data argument in geom_smooth() overrides the global data argument in ggplot() for that layer only.

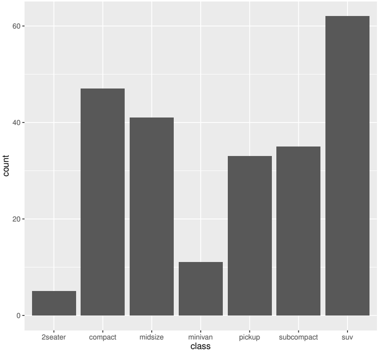
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth(data = filter(mpg, class == "subcompact"),  se = FALSE)
```



Making Graphs

Bar Charts: total number of classes in the mpg dataset

```
ggplot(data = mpg) +
  geom_bar(mapping = aes(x = class))
```



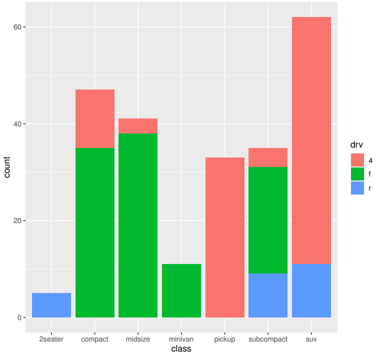
If you want to display a bar chart of proportion, rather than count you must override the default mapping of the y variable.

```
ggplot(data = mpg) +
  geom_bar(mapping = aes(x = class, y = stat(prop), group = 1))
```

Making Graphs

Again, we can additionally specify a grouping variable, based on which we color code the rectangles.

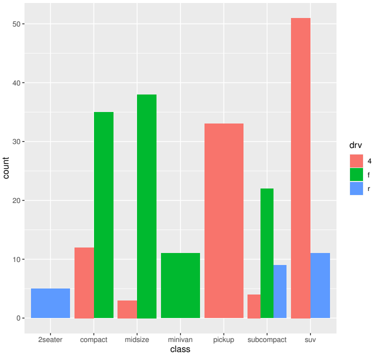
```
ggplot(data = mpg) +
  geom_bar(mapping = aes(x = class, fill = drv))
```



Making Graphs

By default, the rectangles are stacked on top of each other. If this is not desired, we can use the argument position = “dodge” of the function geom_bar().

```
ggplot(data = mpg) +
  geom_bar(mapping = aes(x = class, fill = drv), position = "dodge")
```



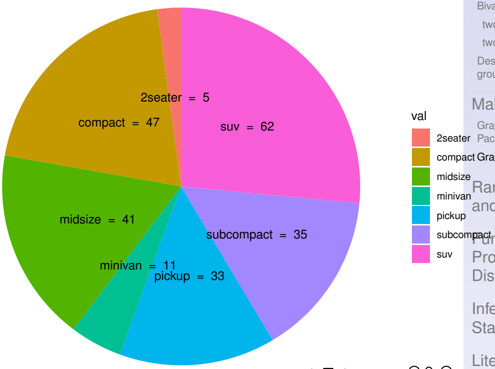
Making Graphs

Pie Charts: There are two types of bar charts: geom_bar() and geom_col(). geom_bar() makes the height of the bar proportional to the number of cases in each group. If you want the heights of the bars to represent values in the data, use geom_col() instead.
The default coordinate system is the Cartesian coordinate system where the x and y positions act independently to determine the location of each point. There are a number of other coordinate systems that are occasionally helpful. coord_polar() uses polar coordinates and can be used to create pie charts. Pie charts in ggplot are basically transformed stacked bar charts that you need geom_bar to make it work. We use a single group (x =) to bring all the values together and fill on the column of interest to divide the area.

Making Graphs

```
# a bar chart with a single bar.
ggplot(data = class.tab, mapping = aes(x = "", y = n, fill = val)) +
  geom_col(width = 1) +
# adding coord_polar()
  coord_polar(theta = "y") +
# adding text to the slices
  geom_text(mapping = aes(label = paste(val," = ",n)),
                position = position_stack(vjust = 0.5)) +
# remove axes and grey background
  theme_void()
```

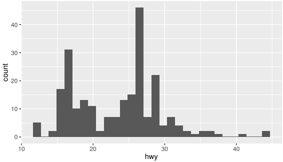
```
tab <- table(mpg$class)
class.tab <- tibble(
  val = names(tab),
  n = tab,
  per = n/ sum(tab))
```



Making Graphs

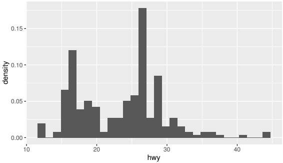
Histograms:

```
ggplot(data = mpg) +
  geom_histogram(mapping = aes(x = hwy))
```



y-values = proportion of observation in the bins rather than total number

```
ggplot(data = mpg) +
  geom_histogram(mapping = aes(x = hwy, y = ..density..))
```



Making Graphs

A histogram provides a graphical representation of the distribution of a numerical variable. For this purpose, the values of this variable are divided into discrete intervals, or bins. On the Y-axis the frequencies in the respective intervals are then displayed. Determining the size of the intervals (binwidth) is critical. If we do not specify anything, ggplot2 selects a binwidth itself, but we can also specify it ourselves with the binwidth argument.

```
ggplot(data = mpg) +
  geom_histogram(mapping = aes(x = hwy), binwidth = 2)
```

or specify the number of bins

```
ggplot(data = mpg) +
  geom_histogram(mapping = aes(x = hwy), bins = 8)
```

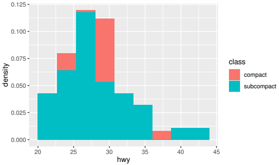
or a numeric vector giving the bin boundaries.

```
ggplot(data = mpg) +
  geom_histogram(mapping = aes(x = hwy),
                 breaks = c(0,10,15,20,25,30,35,40,60))
```

Making Graphs

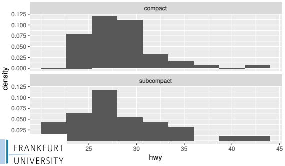
Of course, there is also the possibility to use a grouping variable for histograms. As with the Bar Chart, the histograms are stacked on top of each other. If we want them on top of each other, we use position = “identity”. Since the no of observations may differ by the values of a grouping variable, we use y = ..density..

```
    ggplot(data = mpg %>% filter(class %in% c("subcompact","compact"))) +
  geom_histogram(mapping = aes(x = hwy, y = ..density.., fill = class), bins = 10,
                position = "identity")
```



facets

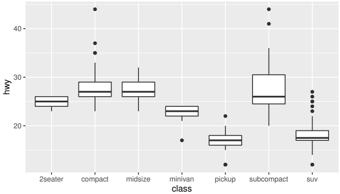
```
ggplot(data = mpg %>% filter(class %in% c("subcompact","compact"))) +
  geom_histogram(mapping = aes(x = hwy, y = ..density..), bins = 10,
                position = "identity") +
  facet_wrap(~ class, nrow = 2)
```



Making Graphs

Boxplots:

```
ggplot(data = mpg) +
  geom_boxplot(mapping = aes(x = class, y = hwy))
```



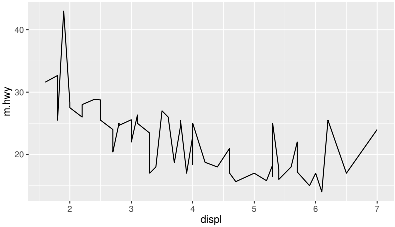
horizontal instead of vertical boxplots

```
ggplot(data = mpg) +
  geom_boxplot(mapping = aes(x = class, y = hwy)) +
  coord_flip()
```

Making Graphs

Lineplots: With the geom_line() function we can create line charts. As an example, we want to calculate the average hwy for the different values of displ and then plot it.

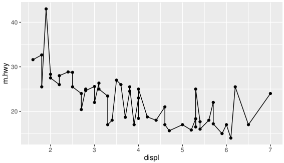
```
mpg %>% select(displ,hwy, drv) %>%
  group_by(displ,drv) %>%
  mutate(m.hwy = mean(hwy)) %>%
  select(-hwy) %>%
  unique() -> tab
ggplot(data = tab) +
  geom_line(mapping = aes(x = displ, y = m.hwy))
```



Making Graphs

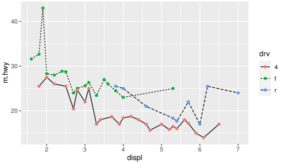
observations points additionally

```
ggplot(data = tab, mapping = aes(x = displ, y = m.hwy)) +
  geom_line() +
  geom_point()
```



grouped by drv

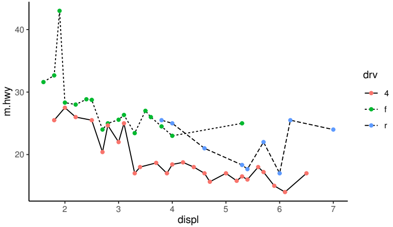
```
ggplot(data = tab, mapping = aes(x = displ, y = m.hwy)) +
  geom_line(aes(linetype = drv)) +
  geom_point(aes(color = drv))
```



Making Graphs

Labeling, themes: ggplot2 automatically chooses the gray background that ggplot2. We can change by choosing a specific theme. There are two themes which have a white background: theme_bw() and theme_classic(). They differ in that theme_classic() draws no grid lines, and only the left and bottom axis.

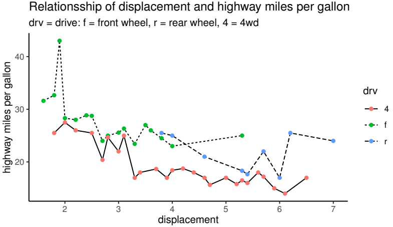
```
ggplot(data = tab, mapping = aes(x = displ, y = m.hwy)) +
  geom_line(aes(linetype = drv)) +
  geom_point(aes(color = drv)) +
  theme_classic()
```



Making Graphs

The labels of the X/Y axes can be changed with xlab() and ylab(), and we can give the plot a title with the ggtitle() function.

```
ggplot(data = tab, mapping = aes(x = displ, y = m.hwy)) +
  geom_line(aes(linetype = drv)) +
  geom_point(aes(color = drv)) +
  theme_classic() +
  xlab("displacement") +
  ylab("highway miles per gallon") +
  ggtitle("Relationsship of displacement and highway miles per gallon",
          subtitle =
          "drv = drive: f = front wheel, r = rear wheel, 4 = 4wd")
```



Random Samples and Permutations

sample(x, size = n , replace = FALSE, prob = NULL)

takes a sample of the specified size from the elements of x using either with or without replacement.

- x : either a vector of one or more elements from which to choose, or a positive integer.
- size : a non-negative integer giving the number of items to choose.
- replace : should sampling be with replacement?
- prob : a vector of probability weights for obtaining the elements of the vector being sampled.

Random Samples and Permutations

```
> x <- 1:12
> # a random permutation
> sample(x)
 [1] 11  8  6  3 10 12  7  4  5  9  1  2
> # sampling with replacement
> sample(x, replace = TRUE)
 [1]  5  9  3 11  1 12  6 12  1  7  6  1
> # 5 Bernoulli trials
> sample(c("tail","head"), 5, replace = TRUE)
[1] "head" "tail" "head" "tail" "tail"
```

Random Samples and Permutations

```
> # sampling with replacement out of an urne where 50% of the balls
> # are red, 20% of the balls are white and 30% of the balls are blue
> sample(c("red","white","blue"), size = 10,
+        replace = TRUE,
+        prob = c(0.5,0.2,0.3))
 [1] "red"   "red"   "red"   "white" "blue"  "white" "red"
 [8] "blue"  "blue"  "white"
> # sampling without replacement out of an urne with
> # 5 red balls, 2 white balls and 3 blue balls
> sample(c("red","red","red","red","red",
+          "white","white",
+          "blue","blue","blue"),
+        size = 7, replace = FALSE)
[1] "red"  "red"  "blue" "blue" "red"  "red"  "blue"
```

Functions for Probability Distributions

Every distribution that R handles has four functions. There is a root name, for example, the root name for the normal distribution is norm. This root is prefixed by one of the letters

- p for "probability", the cumulative distribution function
- q for "quantile", the inverse cumulative distribution function
- d for "density function"
- r for "random, a random variable having the specified distribution

For the normal distribution, these functions are pnorm, qnorm, dnorm, and rnorm. For the binomial distribution, these functions are pbinom, qbinom, dbinom, and rbinom. And so forth.
For continous distribution the "d" function has no practical importance, but for a discrete distribution, the "d" function calculates the density, which in this case $f(x) = P(X = x)$ and hence is useful in calculating probabilities.

Functions for Probability Distributions

R has functions to handle many probability distributions. Some of these are listed below.

Distribution	Functions
Binomial	pbinom qbinom dbinom rbinom
Chi-Square	pchisq qchisq dchisq rchisq
Exponential	pexp qexp dexp rexp
F	pf qf df rf
Geometric	pgeom qgeom dgeom rgeom
Hypergeometric	phyper qhyper dhyper rhyper
Negative Binomial	pnbinom qnbinom dnbinom rnbinom
Normal	pnorm qnorm dnorm rnorm
Poisson	ppois qpois dpois rpois
Student t	pt qt dt rt
Uniform	punif qunif dunif runif

Functions for Probability Distributions

Example:

- Binomial distribution: B(n=10, p=0.3)

```
> # random sample of size 20
> rbinom(n = 10, size = 20, prob = 0.3)
 [1] 6 6 3 7 7 8 9 4 5 8
> # probabilities: P(X=2), P(X=5), P(X=8)
> dbinom(x = c(2,5,8), size = 10, prob = 0.3)
[1] 0.233474440 0.102919345 0.001446701
> # values of the distribution function: P(X<=2), P(X<=5), P(X<=8)
> pbinom(q = c(2,5,8), size = 10, prob = 0.3)
[1] 0.3827828 0.9526510 0.9998563
> # quartiles
> qbinom(p = c(0.25,0.5,0.75), size = 10, prob = 0.3)
[1] 2 3 4
```

- Normal Distribution: $N(u = 2, \sigma = 0.5)$

```
> # random sample of size 7
> rnorm(n = 7, mean = 2, sd = 0.5)
[1] 2.471300 2.413129 1.594230 2.238124 2.510629 2.322692 2.521572
> # values of the distribution function: P(X<=1), P(X<=2), P(X<=2.5)
> pnorm(q = c(1,2,2.5), mean = 2, sd = 0.5)
[1] 0.02275013 0.50000000 0.84134475
> # quartiles
> qnorm(p = c(0.25,0.5,0.75), mean = 2, sd = 0.5)
[1] 1.662755 2.000000 2.337245
```

Inferential Statistics

Confidence intervalls and statistical tests are closely related. In R function performing statistical tests calculate confidence intervals, too. In the following confidence intervals and statistical tests are discussed together in the different cases.

Inferential Statistics: z.test()

Base R does not have a command that will compute the test of hypothesis and compute confidence interval on the mean of a normally distributed population when the standard deviation of the population is known. The package TeachingDemos contains the function z.test(), which calculates both.

Usage: z.test(x, mu = 0, stdev, alternative = c(“two.sided”, “less”, “greater”), n=length(x), conf.level = 0.95, ...)

```
> z.test(x=rnorm(n=20, mean = 1, sd = 2),mu=0,stdev=2,
+        alternative="two.sided",n=20,conf.level=0.95)

One Sample z-test

data:  rnorm(n = 20, mean = 1, sd = 2)
z = 3.1533, n = 20.00000, Std. Dev. = 2.00000, Std.
Dev. of the sample mean = 0.44721, p-value = 0.001614
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.5336746 2.2867197
sample estimates:
mean of rnorm(n = 20, mean = 1, sd = sigma) 1.410197
```

Inferential Statistics: t.test()

t.test() can be used to perform one sample or two sample tests on the mean of one or two samples from normal distributions with unknown standard deviation. In case of a two sample test it is used to determine whether the means of two groups are equal to each other. The assumption for the test is that both groups are sampled from normal distributions with equal variances.

Usage: t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"), mu = 0, paired = FALSE, conf.level = 0.95)

Inferential Statistics: t.test()

In case of a one sample test y must be NULL.

```
> t.test(x=rnorm(n=20, mean = 1, sd = 2), mu=0,
+         stdev = 2, alternative = "two.sided",
+         n = 20, conf.level = 0.95)

One Sample t-test

data:  rnorm(n = 20, mean = 1, sd = 2)
t = 3.0686, df = 19, p-value = 0.006322
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.4400656 2.3283619
sample estimates:
mean of x
 1.384214
```

Inferential Statistics: t.test()

In case of a two sample test the sample are given by x and y. The length of the two vectors x and y must be the same, since the mean of x-y is regarded. The alternative and the confidence interval refer to the difference of x and y.

```
> # two samples
> t.test(x=rnorm(n=20, mean = 1, sd = 1),
+        y=rnorm(n=20, mean = 1.2, sd = 2),
+        alternative = "less", paired = TRUE,
+        conf.level = 0.95)

Paired t-test

data:  rnorm(n = 20, mean = 1, sd = 1) and rnorm(n = 20, mean = 1.2, sd = 2)
t = -0.16594, df = 19, p-value = 0.435
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
      -Inf 0.7363194
sample estimates:
mean of the differences
            -0.07816433
```

Inferential Statistics: sigma.test()

Functions to compute tests and confidence intervals about the variance of samples from normal distributions ed are not in base R. But the package TeachingDemos contains the function sigma.test() which covers both.

Usage: sigma.test(x, sigma = 1,alternative = c(“two.sided”, “less”, “greater”), conf.level = 0.95)

```
> sigma.test(x=rnorm(n=20, mean=1, sd=2), sigma=1.5,
+            alternative = "greater", conf.level = 0.99)

One sample Chi-squared test for variance

data:  rnorm(n = 20, mean = 1, sd = 2)
X-squared = 29.576, df = 19, p-value = 0.05745
alternative hypothesis: true variance is greater than 2.25
99 percent confidence interval:
 1.838748      Inf
sample estimates:
var of rnorm(n = 20, mean = 1, sd = 2)
                              3.502416
```

Inferential Statistics: binom.test()

Performs an exact test of a simple null hypothesis and calculates a confidence about the probability of success in a Bernoulli experiment.

Usage: binom.test(x, n, p = 0.5, alternative = c(“two.sided”, “less”, “greater”), conf.level = 0.95)

Confidence intervals are obtained by a procedure given in Clopper and Pearson (1934). This guarantees that the confidence level is at least conf.level, but in general does not give the shortest-length confidence intervals.

```
> binom.test(x=20, n=30, p=0.5, alternative = "greater",
+            conf.level = 0.9)

Exact binomial test

data:  20 and 30
number of successes = 20, number of trials = 30, p-value = 0.04937
alternative hypothesis: true probability of success is greater than 0.5
90 percent confidence interval:
 0.5337188 1.0000000
sample estimates:
probability of success
             0.6666667
```

Inferential Statistics: var.test()

Performs an F test to compare the variances of two samples from normal populations.

Usage: var.test(x, y, ratio = 1, alternative = c(“two.sided”, “less”, “greater”), conf.level = 0.95)

- ratio: the hypothesized ratio of the population variances of x and y
- Confidence level for the returned confidence interval.

```
> var.test(x=rnorm(50, mean = 0, sd = 2),
+          y=rnorm(30, mean = 1, sd = 1),
+          ratio = 2, alternative = "greater", conf.level = 0.95)

F test to compare two variances

data:  rnorm(50, mean = 0, sd = 2) and rnorm(30, mean = 1, sd = 1)
F = 1.4177, num df = 49, denom df = 29, p-value = 0.1586
alternative hypothesis: true ratio of variances is greater than 2
95 percent confidence interval:
 1.595273      Inf
sample estimates:
ratio of variances
         2.835417
```

Inferential Statistics: chisq.test()

With the help of chisq.test() various statistical tests can be performed.

Usage: chisq.test(x, y = NULL, p = rep(1/length(x), length(x)))

Test of Independence: Analyze a contingency table formed by two categorical variables to evaluate whether there is a significant association between the categories of the two variables.
Null hypothesis: joint distribution of the cell counts in a 2-dimensional contingency table is the product of the row and column marginals

- x = numeric vector or matrix
- y a numeric vector; ignored if x is a matrix.

If x is a matrix with at least two rows and columns, it is taken as a two-dimensional contingency table: the entries of x must be non-negative integers. Otherwise, x and y must be vectors or factors of the same length; cases with missing values are removed, the objects are coerced to factors, and the contingency table is computed from these.

Inferential Statistics: chisq.test()

```
> cat_data <-
+   tibble(
+     hair = sample(x=c("black","brown","red","blond"), size = 100, replace = TRUE),
+     sex = sample(x=c("male","female"), size = 100, replace = TRUE)
+   )
>
> chisq.test(cat_data$hair, cat_data$sex) -> results
> results
Pearson's Chi-squared test
data:  cat_data$hair and cat_data$sex
X-squared = 1.7294, df = 3, p-value = 0.6304
> results$observed
             cat_data$sex
cat_data$hair female male
        black     10   12
        blond     11   14
        brown     11   16
        red       15   11
> results$expected
             cat_data$sex
cat_data$hair female  male
        black  10.34 11.66
        blond  11.75 13.25
        brown  12.69 14.31
        red    12.22 13.78

> # table(x,y)
> table(cat_data$hair, cat_data$sex) %>% chisq.test()
Pearson's Chi-squared test
data:  .
X-squared = 1.7294, df = 3, p-value = 0.6304
```

Inferential Statistics: chisq.test()

Test of Homogeneity: to see whether different columns (or rows) of data in a table come from the same population or not.

Example: In a study the television viewing habits of children are considered, sample of 300 first graders - 100 boys and 200 girls, each child is asked which of the TV programs A, B, C they like best.

```
> boys <- c(50,30,20)
> girls <- c(50,80,70)
> chisq.test(cbind(boys,girls))

Pearson’s Chi-squared test

data:  cbind(boys, girls)
X-squared = 19.318, df = 2, p-value = 6.384e-05
```

Inferential Statistics: chisq.test()

Remark:

- Calculation is identical to that of the chi-square test of independence; the data input, a contingency table, is also the same.
- Differences:
  - The chi-square test of independence assumes that sampling error plays a role in both which column categories were selected in the data and which row categories were selected. The test of homogeneity, by contrast, is derived from the assumption that the sample sizes for columns (or equivalently only the rows) has been pre-specified.
  - If the sample size of either the rows, or the columns, of the table are fixed, the theoretical assumptions of the chi-square test of independence are violated and the test of homogeneity should be implied instead - but you get the same conclusion regardless.

Inferential Statistics: chisq.test()

Goodness of Fit Test: If x is a matrix with one row or column, or if x is a vector and y is not given, then a goodness-of-fit test is performed. The entries of x must be non-negative integers. In this case, the hypothesis tested is whether the population probabilities equal those in p, or are all equal if p is not given.

Example: Test for uniform distribution: 60 throws of a dice

```
> # 60 throws of a dice:
> sample <- c(7,16,8,17,3,9)
> # test for uniform distribution
> chisq.test(x=sample, p=c(1/6,1/6,1/6,1/6,1/6,1/6))

Chi-squared test for given probabilities

data:  sample
X-squared = 14.8, df = 5, p-value = 0.01125
```

Inferential Statistics: wilcox.test()

With the help of wilcox.test() one-sample Wilcoxon signed rank test Wilcoxon rank sum test (Wilcoxon-Mann-Whitney U Test) and Wilcoxon signed rank test on paired samples can be conducted.

Usage: wilcox.test(x, y = NULL, alternative = c("two.sided", "less", "greater"), mu = 0, paired = FALSE, ...)

One-sample Wilcoxon signed rank test: is used to assess whether the median of the sample is equal to a known value.
Null hypothesis: Median = mu

```
vec <- c(12,16,18,24,26,31,38,40)
wilcox.test(x=vec, mu=30, conf.int=TRUE)

Output: Wilcoxon signed rank exact test
data:  vec
V = 10, p-value = 0.3125
alternative hypothesis: true location is not equal to 30
95 percent confidence interval:
 16.0 35.5
sample estimates:
(pseudo)median
          25.5
```

Inferential Statistics: wilcox.test()

Wilcoxon rank sum test: is used to compare the medians between two independent groups. It is a non-parametric alternative to the unpaired samples t-test for comparing unpaired data. It's used when the data are not normally distributed.

- x,y: numeric vectors
- paired=FALSE: value specifying that we want to compute an unpaired Wilcoxon test
- alternative: the alternative hypothesis. Allowed value is one of "two.sided" (default), "greater" or "less".

Null hypothesis: about the medians corresponding to the value of alternative

```
x <- c(0.80, 0.83, 1.89, 1.04, 1.45, 1.38, 1.91, 1.64, 0.73, 1.46)
y <- c(1.15, 0.88, 0.90, 0.74, 1.21)
wilcox.test(x, y, paired = FALSE, alternative = "greater")

Output: Wilcoxon rank sum exact test

data:  x and y
W = 35, p-value = 0.1272
alternative hypothesis: true location shift is greater than 0
```

Inferential Statistics: wilcox.test()

Wilcoxon signed rank test on paired samples: is a non-parametric alternative to the paired samples t-test for comparing the medians of paired data. It's used when the data are not normally distributed.

- x,y: numeric vectors
- paired=TRUE: value specifying that we want to compute a paired Wilcoxon test
- alternative: the alternative hypothesis. Allowed value is one of "two.sided" (default), "greater" or "less".

Null hypothesis: about the medians corresponding to the value of alternative

```
# values before treatment
before <-c(200.1, 190.9, 192.7, 213, 241.4, 196.9, 172.2, 185.5, 205.2, 193.7)
# values after treatment
after <-c(392.9, 393.2, 345.1, 393, 434, 427.9, 422, 383.9, 392.3, 352.2)
wilcox.test(before, after, paired = TRUE, alternative = "two.sided")

Output: Wilcoxon signed rank exact test
data:  before and after
V = 0, p-value = 0.001953
alternative hypothesis: true location shift is not equal to 0
```

Literature

- Hadley Wickham, Garrett Grolemund: R for Data Science, O'Reilly 2017, http://r4ds.had.co.nz
- Tidy Data: https://tidyr.tidyverse.org/articles/tidy-data.html#sec:defining
- Tibbles: https://tibble.tidyverse.org/ and https://tibble.tidyverse.org/articles/tibble.html
- R Graphics Cookbook: Practical Recipes for Visualizing Data 1st Edition, Winston Chang, O'Reilly http://www.cookbook-r.com/Graphs/
- R Graphics Cookbook, 2nd edition, Winston Chang https://r-graphics.org/index.html

Content

Generate Sequences

Data: Conversion, Information, Manipulation

Some Statistical Functions
- Univariate Data
- Bivariate Data
- Descriptive statistics by groups

Making Graphs
- Graphics with the Base Package
- Graphics with ggplot

Random Samples and Permutations

Functions for Probability Distributions

Inferential Statistics

Literature