Kovarianz, Korrelation, (lineare) Regression

Jonathan Harrington & Ulrich Reubold

18. Juni 2019

```
library(ggplot2)
epg = read.table(file.path(pfadu, "epg.txt"))
amp = read.table(file.path(pfadu, "dbdauer.txt"))

source(file.path(pfadu, "ggplotRegression.R"))
```

Die Kovarianz und die Korrelation sind zwei (verwandte) Maße für die Stärke der Beziehung zwischen 2 numerischen Variablen. Betrachten wir beispielsweise den Dataframe epg:

```
head(epg)
```

```
##         F1   F2       COG        DP SUM1278 V
## 1 391.213 1640 1.0555556 0.3750000       8 a
## 2 760.000 1820 1.3823529 0.6666667      12 a
## 3 660.000 1720 1.2857143 0.5416667      11 a
## 4 487.437 1740 0.7857143 0.2916667       6 a
## 5 300.036 2280 1.9736842 0.5833333      16 E
## 6 311.557 2280 1.9000000 0.6250000      16 E
```

- epg enthält F1- und F2-Werte zum Vokaloffset für V/k/-Folgen (mit V = /a ɛ ɪ i ɔ ʊ/), gesprochen von einem deutschen Muttersprachler.
- COG ("centre of gravity") ist ein gemessener Parameter aus der Bereich der Elektropalatographie. Er gibt ein Maß für den Schwerpunkt der Kontakte auf einem 8 x 8 Kontaktflächen großen künstlichem Gaumen. Je höher der Wert von COG, desto weiter vorne ist die Zunge.
- SUM1278 ist die Summe der aktivierten Kontaktflächen auf den jeweils zwei äußersten Kontaktflächenspalten des künstlichen Gaumens (da es acht Spalten sind, handelt es sich um die Spalten 1, 2, 7, und 8).

1. Kovarianz

Die Kovarianz ist ein Maß für die Assoziation zwischen zwei Variablen. Je mehr die Kovarianz von 0 (Null) abweicht, umso deutlicher ist die lineare Beziehung zwischen den beiden Variablen.

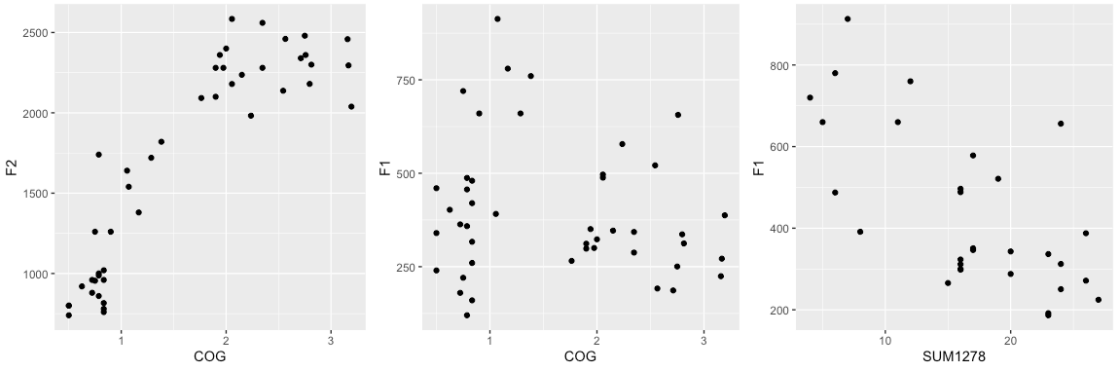
```
# mit ggplot() und gridExtra
library(gridExtra)
# F2 als Funktion von COG
p1 = ggplot(epg) +
  aes(y = F2, x = COG) +
```

```
  geom_point()

# F1 als Funktion von COG
p2 = ggplot(epg) +
  aes(y = F1, x = COG) +
  geom_point()

# F1 als Funktion von SUM1278
# wir betrachten nur /i/, /I/, /E/, und /a/
temp = with(epg, V %in% c("i", "I", "E", "a"))
p3 = ggplot(epg[temp, ]) +
  aes(y = F1, x = SUM1278) +
  geom_point()

# um die Bilder in 3 Spalten und einer Reihe zu arrangieren
grid.arrange(p1, p2, p3,  ncol = 3, nrow = 1)
```



Links: Kovarianz hoch und positiv (509,6908, s.u.); Mitte: Kovarianz nah an 0 (-24,26598); Rechts: Kovarianz mittelstark und negativ (-289,516)

Die Kovarianz berechnet sich als die Produkt-Summe der Abweichungen vom Mittelwert; betrachten wir z.B. das linke Beispiel aus der Abbildung, also F2 als Funktion von COG:

```
y = epg$F2
x = epg$COG
n = nrow(epg)

# Mittelwerte:
mx = mean(x)
my = mean(y)

# Abweichungen vom jeweiligen Mittelwert
dx = x - mean(x)
dy = y - mean(y)

# Kovarianz = Produkt-Summe der Abweichungen dividiert durch n-1
covxy = sum(dx*dy)/(n-1)
covxy

## [1] 509.6908
```

```r
# all das vorgenannte als Funktion in R
cov(x, y)
```

```
## [1] 509.6908
```

Einige Merkmale der Kovarianz

- `cov(x,y) == cov(y,x)`
- `cov(x,x) == var(x)` (`== sd(x)^2`)
- `var(x+y) == var(x)+var(y) + 2 * cov(x,y)`

Daher: wenn es keine lineare Beziehung zwischen x und y gibt, ist `cov(x,y)` 0 (Null), so dass dann gilt:

- `var(x+y) == var(x) + var(y)`

2. Korrelation

Die Korrelation (Pearson's product-moment correlation), *r*, ist dasselbe wie die Kovarianz, aber sie normalisiert für die Größe von x und y

- *r* ist die Kovarianz von x, y, dividiert durch deren Standardabweichungen
- Der große Vorteil: *r* variiert zwischen -1 und +1

```r
cov(x,y)
```

```
## [1] 509.6908
```

```r
r = cov(x,y)/(sd(x) * sd(y))
r
```

```
## [1] 0.8917474
```

```r
cor(x,y)
```

```
## [1] 0.8917474
```

```r
cov(x,y)
```

```
## [1] 509.6908
```

Auch wenn die Werte in x um den Faktor 1000 größer wären, ändert sich die Korrelation `r` nicht (im Gegensatz zur Kovarianz!):

```r
xgross = x*1000
cov(xgross,y)
```

```
## [1] 509690.8
```

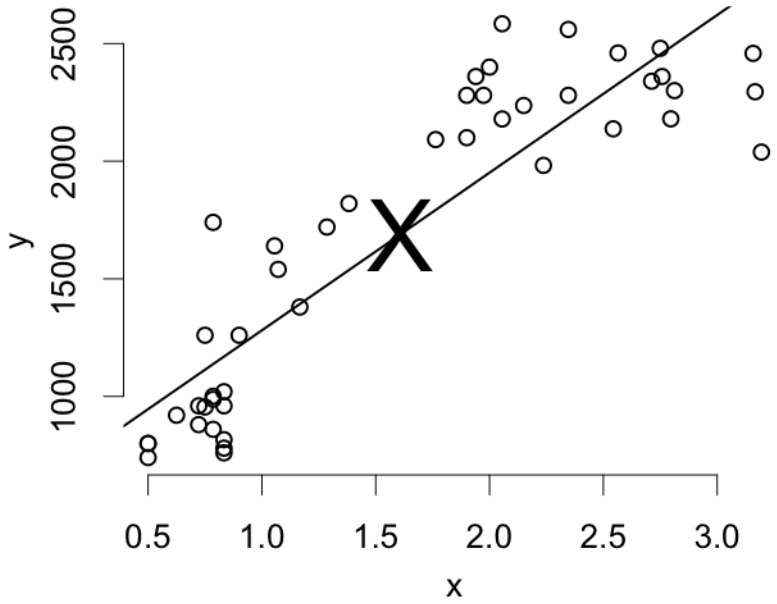
```r
cor(xgross,y)
```

```
## [1] 0.8917474
```

3. Regression

y-auf-x Regression: *y* (die abhängige Variable) soll durch *x* (die unabhängige Variable) modelliert werden, also **y durch die Werte von x eingeschätzt werden**.

Die **Regressionslinie** ist jene gerade Linie durch die Verteilung, bei welcher der Abstand der Stichprobe zu der Linie minimiert wird.



Diese Regressionslinie durchschneidet immer (m*x*, m*y*), also den Mittelwert (*X*) der Verteilung

Die Regressionslinie wird berechnet über

$$\hat{y} = bx + k$$

- *b* ist die Steigung (Engl.: 'slope'):

```
b = r * sd(y)/sd(x)
b

## [1] 670.267

# oder
b = cov(x,y)/var(x)
b

## [1] 670.267
```

- *k* ist das sog. Intercept (der Schnittpunkt auf dem y-Achsenabschnitt, wenn x == 0):

```
k = my - b*mx
k
```

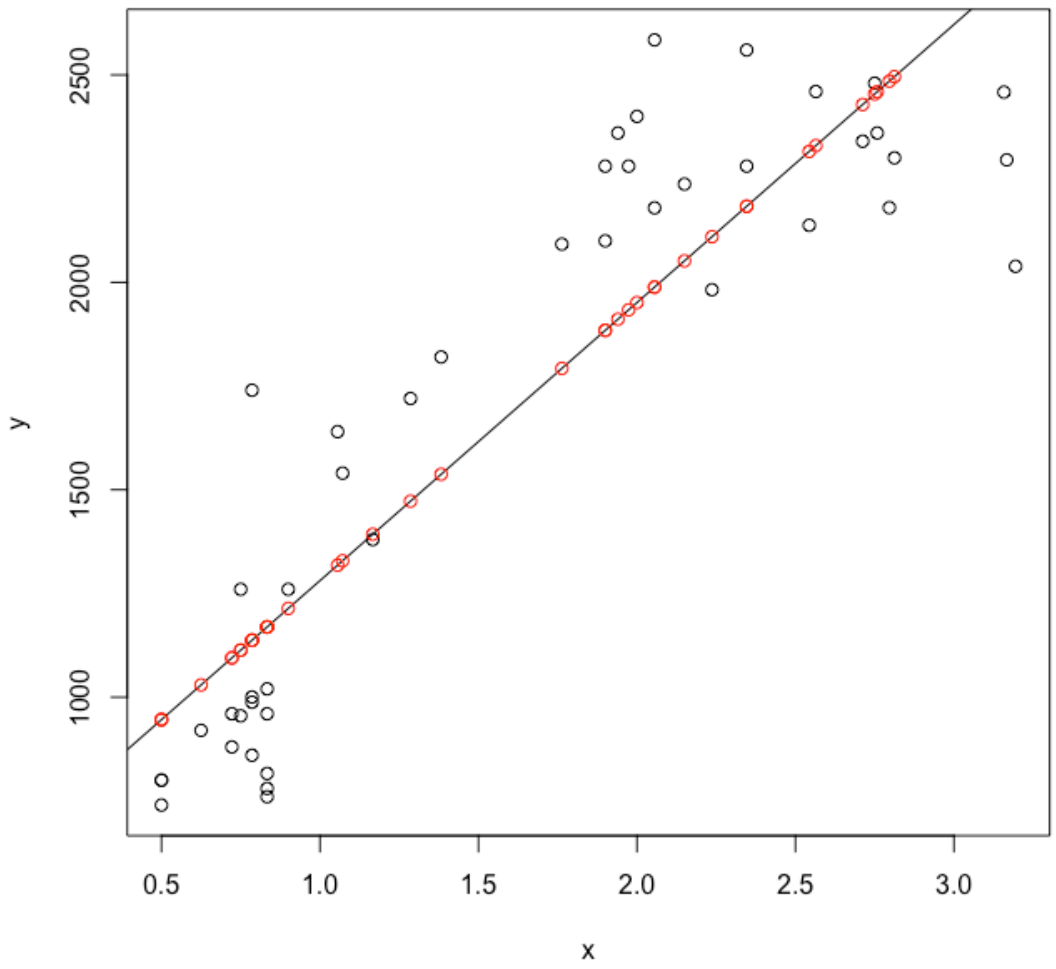
```
## [1] 610.6845
```

- ŷ sind die eingeschätzten Werte, die auf der Regressionslinie liegen:

```
yhut = b * x + k
yhut
```

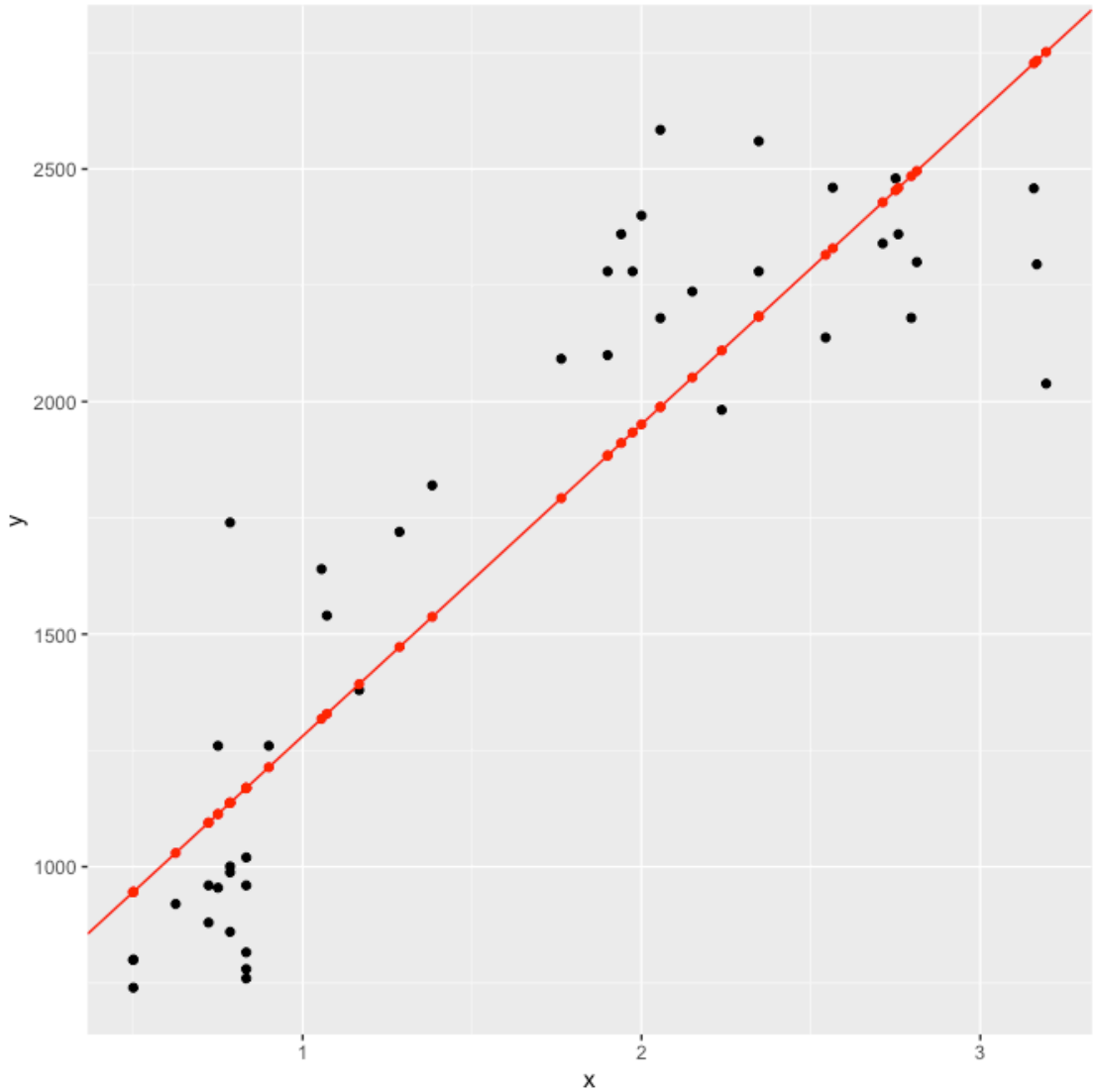
```
##  [1] 1318.1885 1537.2299 1472.4562 1137.3228 1933.5798 1884.1917 1911.0023
##  [8] 1792.4709 2329.5949 2453.9186 2428.5297 2183.2338 2183.2338 1951.2184
## [15] 1884.1917 1137.3228 1094.7661 1137.3228 1029.6013 1169.2402 1169.2402
## [22] 1094.7661 1328.8276 1113.3847 1392.6626 1213.9247 2051.7584 2109.9658
## [29] 1988.4554 1988.4554 2459.3240 2733.1965 2751.8150 2495.8103 2727.3169
## [36] 2315.4939 2484.9494 1113.3847  945.8179  945.8179  945.8179 1169.2402
## [43] 1137.3228 1169.2402 1169.2402
```

```
plot(y ~ x)
abline(k, b)
points(x, yhut, col = "red")
```



oder

```
# oder mit ggplot()
# Data-frame
df = data.frame(y, x, yhut)
# Die Abbildung
p1 = ggplot(df) + aes(y = y, x = x) + geom_point()
# Eingeschaetze Werte
p2 = geom_point(aes(y = yhut), color = "red")
# Regressionsline
p3 = geom_abline(intercept = k,
                 slope = b,
                 col = "red")
# Alle zusammen
p1 + p2 + p3
```



Error und SSE

Der *error* (auch *residuals*) ist der **Unterschied** zwischen den **tatsächlichen** und den **eingeschätzten** Werten.

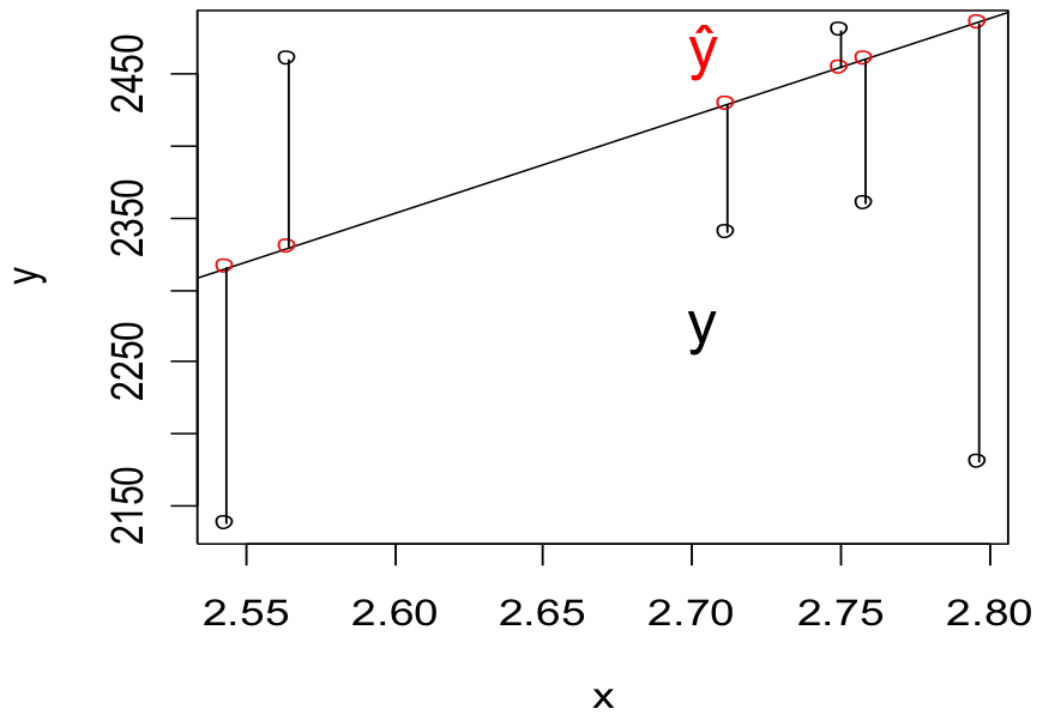
```
error = y - yhut
error
```

```
##  [1]  321.81154  282.77005  247.54375  602.67723  346.42025  395.80834
##  [7]  448.99766  299.54908  130.40514   26.08143  -88.52967   96.76616
## [13]  376.76616  448.78164  215.80834 -136.81277 -214.76614 -149.47677
## [19] -109.60130 -209.24025 -409.24025 -134.76614  211.17238  146.61533
## [25]  -12.66256   46.07529  185.00160 -127.80579  190.99459  595.86459
## [31]  -99.32395 -438.03647 -713.17499 -195.81026 -268.88693 -177.93387
## [37] -304.94945 -158.40367 -205.81793 -145.81793 -145.81793 -149.24025
## [43] -277.32277 -353.23225 -389.24025
```

Der *SSE* ('sum of the squares of the error' ist die sogenannte Summe der Quadrate, d.h. die Summe der quadrierten Fehler:

```
SSE = sum(error^2)
SSE
```

```
## [1] 3871019
```



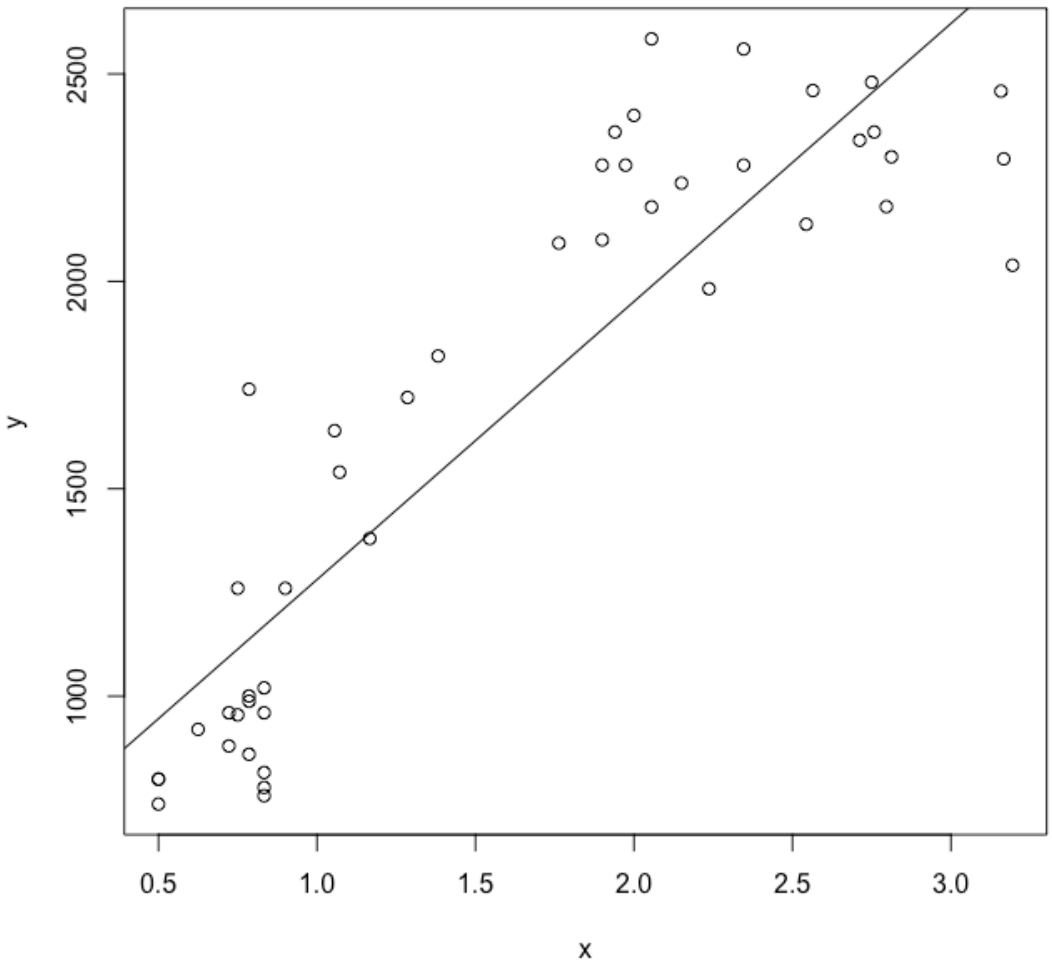
Die Regressionslinie wird auf eine solche Weise berechnet, dass *SSE* minimiert wird.

Der *SSE* wird übrigens manchmal auch *RSS* ('residual sum of squares') genannt.

Regression mit `lm()`

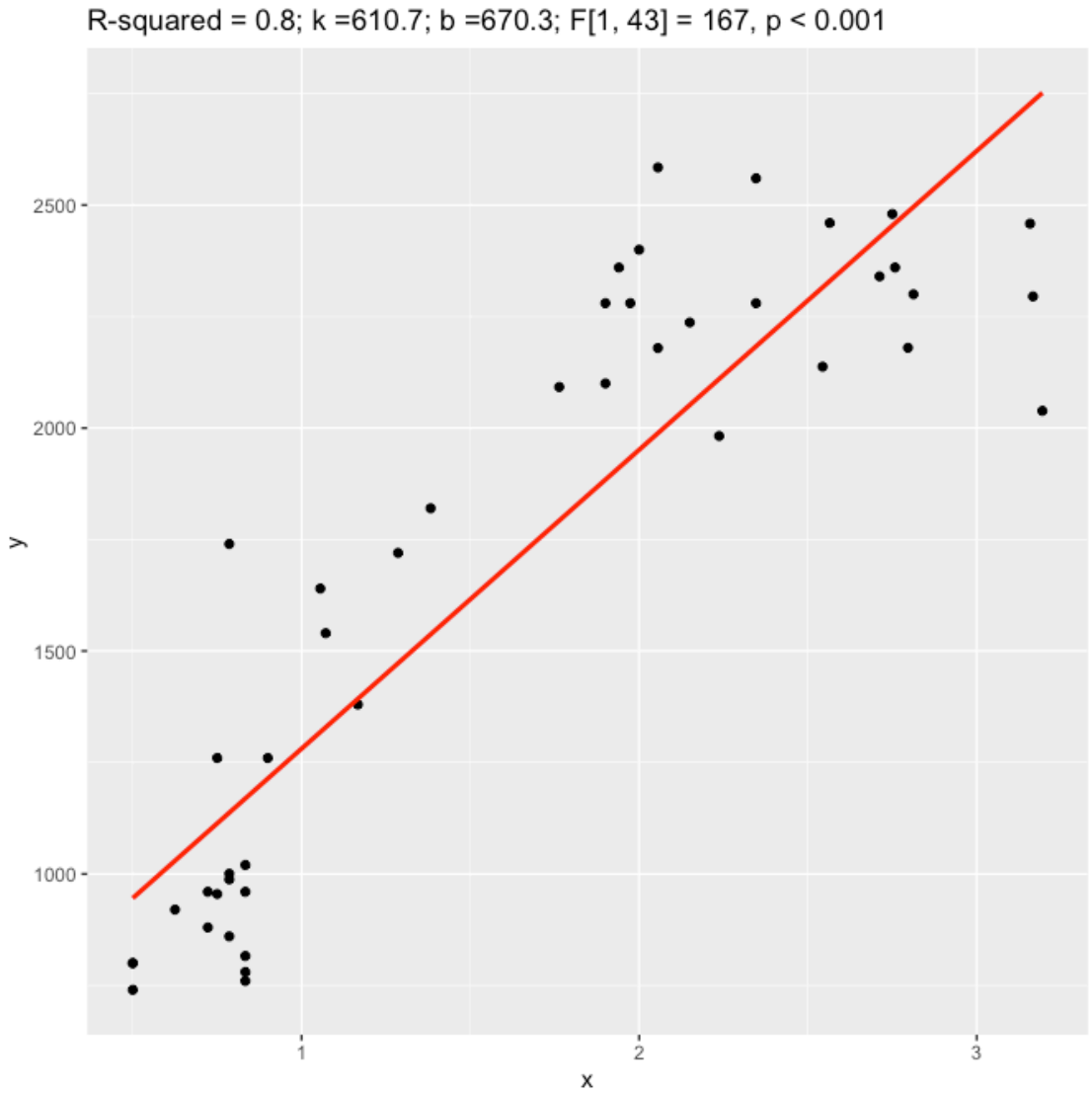
Die Funktion `lm()` berechnet für uns die obengenannten Quantitäten:

```
reg = lm(y ~ x)
plot(y ~ x)
abline(reg)
```



Füt die Abbildung können wir auch die Funktion `ggplotRegression()`[1] benutzen, in die man einfach das Regressionsmodell eingeben kann:

```
# oder mit der Funktion ggplotRegression():
ggplotRegression(reg)
```



Weitere wichtige Maße sind:

```
#Regressionskoeffizienten (Intercept und Steigung)
coef(reg)

## (Intercept)           x
##    610.6845    670.2670
```

[1] leicht adaptierte Version einer ursprünglich von Susan Johnston geschriebenen Funktion aus folgender Quelle: https://rdrr.io/github/G-Thomson/gthor/man/ggplotRegression.html

```r
# Eingeschätzte Werte
yhut = b*x + k
yhut
```

```
##  [1] 1318.1885 1537.2299 1472.4562 1137.3228 1933.5798 1884.1917 1911.0023
##  [8] 1792.4709 2329.5949 2453.9186 2428.5297 2183.2338 2183.2338 1951.2184
## [15] 1884.1917 1137.3228 1094.7661 1137.3228 1029.6013 1169.2402 1169.2402
## [22] 1094.7661 1328.8276 1113.3847 1392.6626 1213.9247 2051.7584 2109.9658
## [29] 1988.4554 1988.4554 2459.3240 2733.1965 2751.8150 2495.8103 2727.3169
## [36] 2315.4939 2484.9494 1113.3847  945.8179  945.8179  945.8179 1169.2402
## [43] 1137.3228 1169.2402 1169.2402
```

```r
# Die gleichen Werte erhalten wir mit:
yhut2 = predict(reg)
yhut2
```

```
##         1         2         3         4         5         6         7
## 1318.1885 1537.2299 1472.4562 1137.3228 1933.5798 1884.1917 1911.0023
##         8         9        10        11        12        13        14
## 1792.4709 2329.5949 2453.9186 2428.5297 2183.2338 2183.2338 1951.2184
##        15        16        17        18        19        20        21
## 1884.1917 1137.3228 1094.7661 1137.3228 1029.6013 1169.2402 1169.2402
##        22        23        24        25        26        27        28
## 1094.7661 1328.8276 1113.3847 1392.6626 1213.9247 2051.7584 2109.9658
##        29        30        31        32        33        34        35
## 1988.4554 1988.4554 2459.3240 2733.1965 2751.8150 2495.8103 2727.3169
##        36        37        38        39        40        41        42
## 2315.4939 2484.9494 1113.3847  945.8179  945.8179  945.8179 1169.2402
##        43        44        45
## 1137.3228 1169.2402 1169.2402
```

```r
# Error
error = y - yhut
error
```

```
##  [1]  321.81154  282.77005  247.54375  602.67723  346.42025  395.80834
##  [7]  448.99766  299.54908  130.40514   26.08143  -88.52967   96.76616
## [13]  376.76616  448.78164  215.80834 -136.81277 -214.76614 -149.47677
## [19] -109.60130 -209.24025 -409.24025 -134.76614  211.17238  146.61533
## [25]  -12.66256   46.07529  185.00160 -127.80579  190.99459  595.86459
## [31]  -99.32395 -438.03647 -713.17499 -195.81026 -268.88693 -177.93387
## [37] -304.94945 -158.40367 -205.81793 -145.81793 -145.81793 -149.24025
## [43] -277.32277 -353.23225 -389.24025
```

```r
# Auch hier: die gleichen Werte erhalten wir mit:
resid(reg)
```

```
##          1          2          3          4          5          6
##  321.81154  282.77005  247.54375  602.67723  346.42025  395.80834
##          7          8          9         10         11         12
##  448.99766  299.54908  130.40514   26.08143  -88.52967   96.76616
##         13         14         15         16         17         18
##  376.76616  448.78164  215.80834 -136.81277 -214.76614 -149.47677
```

```
##         19         20         21         22         23         24
## -109.60130 -209.24025 -409.24025 -134.76614  211.17238  146.61533
##         25         26         27         28         29         30
##  -12.66256   46.07529  185.00160 -127.80579  190.99459  595.86459
##         31         32         33         34         35         36
##  -99.32395 -438.03647 -713.17499 -195.81026 -268.88693 -177.93387
##         37         38         39         40         41         42
## -304.94945 -158.40367 -205.81793 -145.81793 -145.81793 -149.24025
##         43         44         45
## -277.32277 -353.23225 -389.24025

# SSE:
sum(error^2)

## [1] 3871019

# oder - den gleichen Wert - mittels
deviance(reg)

## [1] 3871019
```

Regression: drei wichtige Quantitäten

1. SSE (oder RSS) sum of the squared errors

```
SSE = sum(error^2)
SSE

## [1] 3871019

# oder
SSE = deviance(reg)
SSE

## [1] 3871019
```

2. SSY (oder SST): sum-of-the-squared deviations der tatsächlichen Werte

```
SSY = sum( (y - my)^2)
SSY

## [1] 18902691
```

3. SSR: sum of the squared-deviations von ŷ, also von den eingeschätzten Werten

```
SSR = sum((yhut - my)^2)
SSR

## [1] 15031672
```

$$SSY = SSR + SSE$$

```
SSY = SSR + SSE
SSY

## [1] 18902691
```

R-squared (R^2)

$$SSY = SSR + SSE$$

Je besser die Werte durch die Regressionslinie modelliert werden (also je geringer der Abstand zwischen y und ŷ) umso kleiner SSE, sodass im besten Fall SSE = 0 und SSY = SSR oder SSR/SSY = 1 (das hätte dann die Bedeutung: die tatsächlichen Werte sitzen auf der Linie).

$R^2 = SSR/SSY$ beschreibt auch die Proportion der Varianz in y die durch die Regressionlinie erklärt werden kann.

R^2 variiert zwischen 0 (keine 'Erklärung') und 1 (die Regressionslinie erklärt 100% der Varianz in y).

Diese Quantität SSR/SSY nennt man auch R-squared, weil sie denselben Wert hat wie der quadrierte Korrelationskoeffizient r:

```
SSR/SSY
```

```
## [1] 0.7952134
```

```
cor(x, y)^2
```

```
## [1] 0.7952134
```

Da r zwischen -1 und +1 variiert, muss R^2 zwischen 0 und 1 variieren.

Signifikanz-Test

Was ist die Wahrscheinlichkeit, dass ein lineares Verhältnis zwischen x und y besteht?

Dies kann mit einem t-test mit n-2 Freiheitsgraden berechnet werden:

$$tstat = \frac{r}{\sqrt{\frac{1-r^2}{n-2}}}$$

Von Hand ausgerechnet:

```
rsb = sqrt( (1 - r^2)/(n-2))
tstat = r/rsb
tstat
```

```
## [1] 12.92187
```

```
2 * (1 - pt(tstat, n-2))
```

```
## [1] 2.220446e-16
```

```
fstat = tstat^2
1 - pf(fstat, 1, n-2)
```

```
## [1] 2.220446e-16
```

```r
cor.test(x,y)
```

```
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = 12.922, df = 43, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.8103217 0.9393887
## sample estimates:
##       cor
## 0.8917474
```

*Die Wahrscheinlichkeit, dass die Variablen **nicht** miteinander linear assoziiert sind, ist **fast 0**. ($p < 0.001$).*

Noch einfacher bekommt man dieses Ergebnis mit einem `summary()`-Aufruf für das Modell:

```r
summary(reg)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -713.17 -195.81  -99.32  215.81  602.68
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   610.68      94.65   6.452 8.03e-08 ***
## x             670.27      51.87  12.922  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 300 on 43 degrees of freedom
## Multiple R-squared:  0.7952, Adjusted R-squared:  0.7905
## F-statistic:   167 on 1 and 43 DF,  p-value: < 2.2e-16
```

Es gibt eine signifikante lineare Beziehung zwischen COG und F2 ($R^2 = 0.80$, $F[1,43] = 167$, $p < 0.001$).

Gültigkeit einer Regression[2]

Die **Residuen ('residuals') sollen**:

- **(a) von einer Normalverteilung nicht signifikant abweichen**:

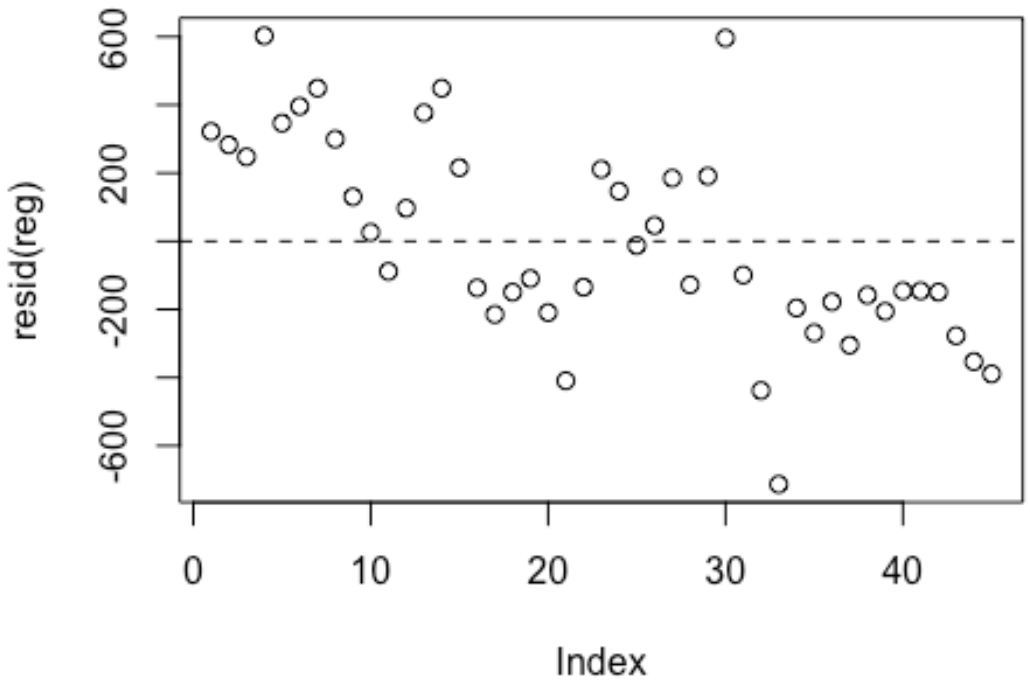
[2] siehe auch https://www.statisticssolutions.com/assumptions-of-linear-regression/

```r
shapiro.test(resid(reg))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(reg)
## W = 0.97037, p-value = 0.2987
```

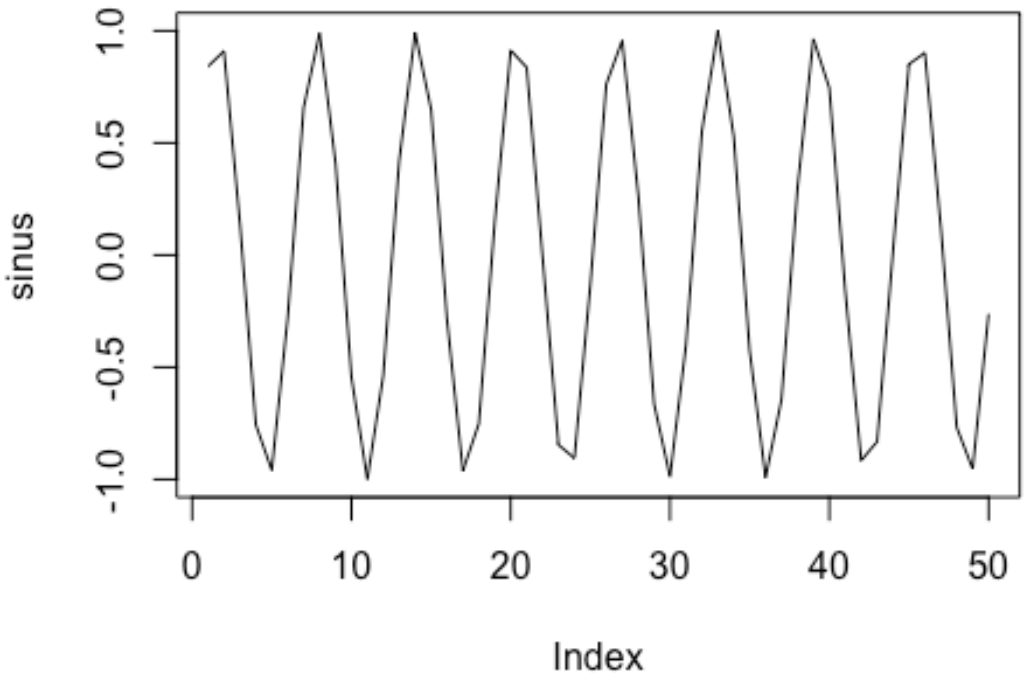
- **(b) eine konstante Varianz aufweisen.** Insbesondere sollten die Residuals nicht wesentlich größer am Anfang/Ende sein, sondern auf eine randomisierte Weise um die Null-Linie verteilt sein. Das ist hier nicht der Fall:

```r
plot(resid(reg))
abline(h=0, lty=2)
```



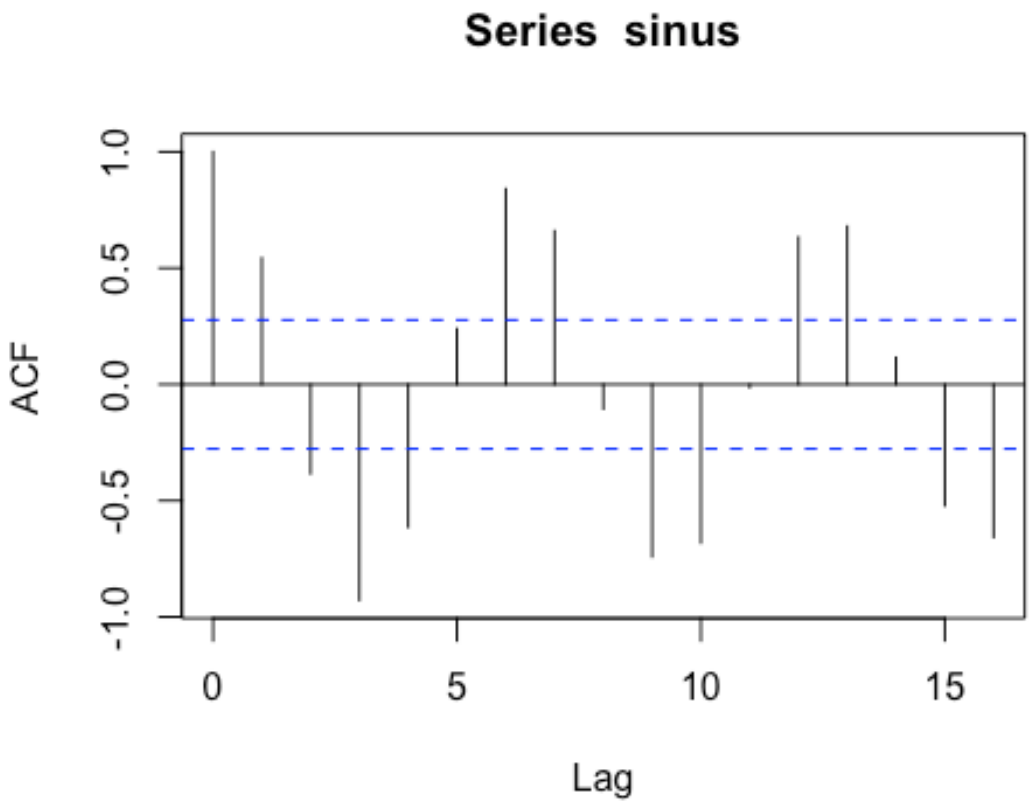
- **(c) keine Autokorrelation aufweisen.** Autokorrelation ist, wenn die Werte eines Signals mit sich selbst korreliert sind. Ein gutes Beispiel für autokorrelierte Daten wäre jene in einem stimmhaften, also periodischem, Sprachsignal, oder einfach in einer Sinusschwingung:

```r
sinus = sin(1:50)
plot(sinus, type = "l")
```



Getestet wird das dann mit der Funktion `acf()`:

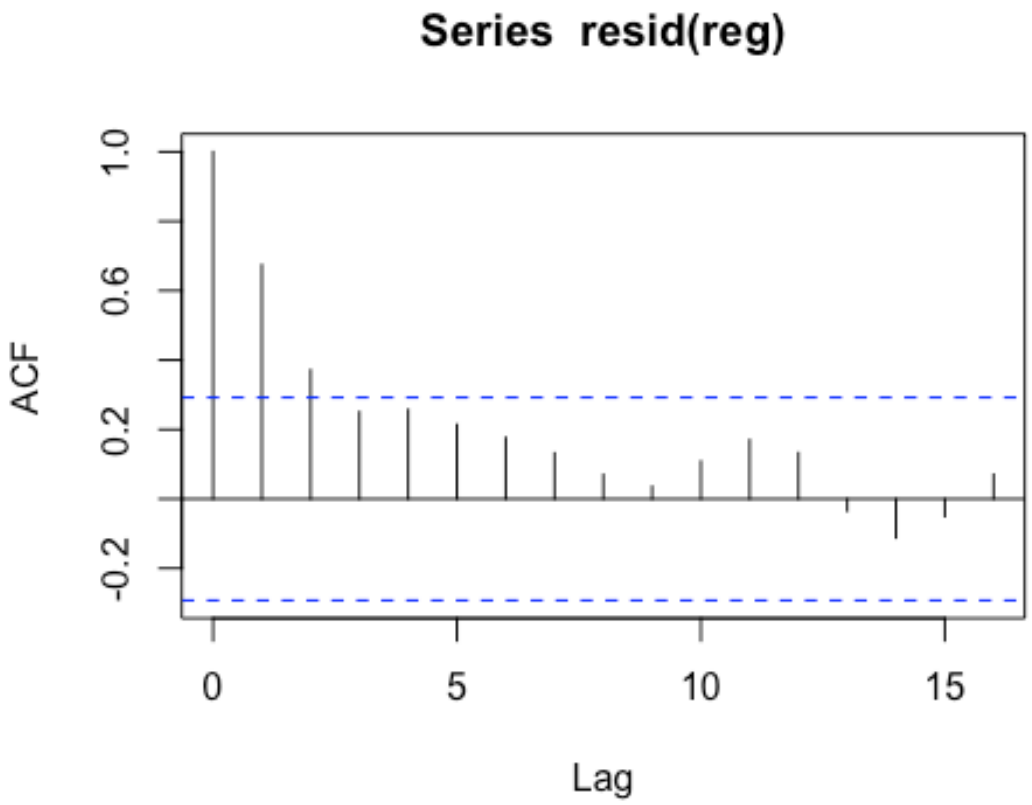
```
acf(sinus)
```



Der Grund dafür, dass Autokorrelation ungünstig ist, liegt darin begründet, dass die Residuen "auf eine *randomisierte* Weise um die Null-Linie verteilt sein" sollen (wie schon oben unter b. steht). Bei vorliegender Autokorrelation gibt es aber stattdessen eine *systematische* Variation um die Null-Linie herum, so dass die Annahme einer randomisierten Variation verletzt wird.

Die Residuen unserer Regression von oben testen wir mittels:

```
acf(resid(reg))
```



Wenn die meisten ACF-Werte innerhalb der blauen Linien liegen, gibt es keine Autokorrelation. Insbesondere sind die Werte bei lag 1 und 2 zu beobachten: diese sollten innerhalb des Vertauensintervalls (das ist jener Bereich, der durch die gestrichelten blauen Linien eingegrenz wird) liegen. Dies ist hier aber nicht der Fall.

Generell lässt sich sagen, dass die Daten aus epg besser nicht linear modelliert werden sollten. Tatsächlich folgen die Daten auch viel eher einer Hockeyschlägerkurve (wie der blauen Linie unten) als einer geraden Linie (wie wir sie oben modelliert haben und sie hier

nochmal in Rot dargestellt wird):