Programming in R for Data Science

learning by doing

Katrien Antonio

KU Leuven and UvA

2019-06-23

About the teacher

A collection of links:

- my personal website
- my GitHub page
- an e-book with more documentation.

Research team is here.

Practical information

Course material including

- R scripts, data, lecture sheets
- a collection of **cheat sheets**

are available from

https://github.com/katrienantonio/PE-Programming-R-for-data-science

Today's agenda

Learning outcomes

Today you will work on:

- optimization
- fitting distributions to data with MLE
- functional programming
- linear models
- generalized linear models *(see R markdown tutorial)*
- generalized additive models *(see R markdown tutorial)*
- decision trees *(see R markdown tutorial)*

You will cover examples of code[1] and work on **R challenges**.

[1] For a detailed discussion of each topic, see e-book.

Questions after day 1

How can I

- estimate a parametric distribution (e.g. normal, Poisson) to a given data set
- generate and store losses from the random variable $L_i = \sum_{j=1}^{N_i} Y_{ij}$ the aggregate loss of policyholder i
- create an R project.

Fitting distributions to data

Optimization

Actuaries often write functions (e.g. a likelihood) that have to be optimized.

Therefore, you'll:

- get to know some R functionalities to do **optimization**
- apply this knowledge to **fit a distribution** to a give data set.

Find the root of a function

Consider the function $f : x \mapsto x^2 - 3^{-x}$.

What is the root x of this function over the interval $[0, 1]$, so that $f(x) = 0$?

```
uniroot(function(x) x^2-3^(-x), lower = 0, upper = 1)
```

```
## $root
## [1] 0.6860224
##
## $f.root
## [1] -8.082734e-06
##
## $iter
## [1] 4
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 6.103516e-05
```

Find the root of a function

First, define the function $f(.)$

```
f <- function(x){
    x^2-3^(-x)
}
```

Then, calculate its root with `uniroot`:

```
opt <- uniroot(f, lower = 0, upper = 1)
names(opt)
```

```
## [1] "root"        "f.root"      "iter"        "init.it"     "estim.prec"
```

and evaluate $f(.)$ in the root

```
f(opt$root)
```

```
## [1] -8.082734e-06
```

Find the root of a function

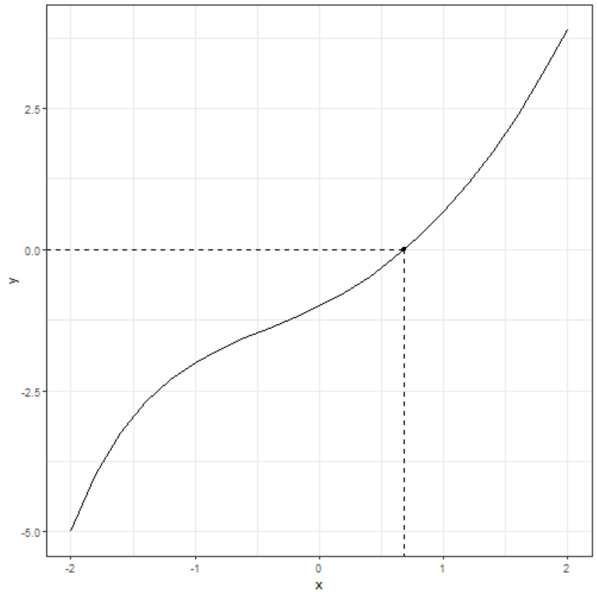
To conclude, you build a visualization (with base R).

```
range <- seq(-2, 2, by = 0.2)
plot(range, f(range), type = "l")
points(opt$root, f(opt$root), pch = 20)
segments(opt$root, -7, opt$root, 0, lty = 2)
segments(-3, 0, opt$root, 0, lty = 2)
```

Find the root of a function

To conclude, you build a visualization (in `ggplot` style).

```
data <- data.frame(x = range, y = f(range))
ggplot(data, aes(x = x, y = y)) + geom_line() +
    geom_point(x = opt$root, y = f(opt$root)) +
    geom_segment(x = opt$root, y = -7, xend = opt$root, yend = 0, lin
    geom_segment(x = -3, y = 0, xend = opt$root, yend = 0, linetype =
    theme_bw()
```

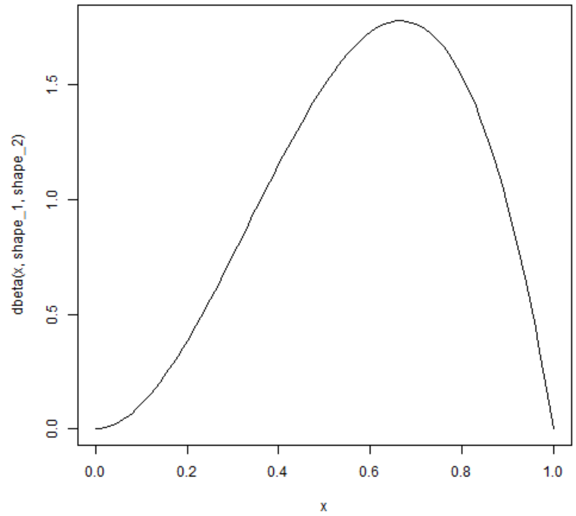


Find the maximum of a function

You look for the maximum of the beta density for a given set of parameters.

First, you sketch the beta density.

```
shape_1 <- 3; shape_2 <- 2
x <- seq(from = 0, to = 1, by = 0.01)
curve(dbeta(x, shape_1, shape_2), xlim = range(x))
```



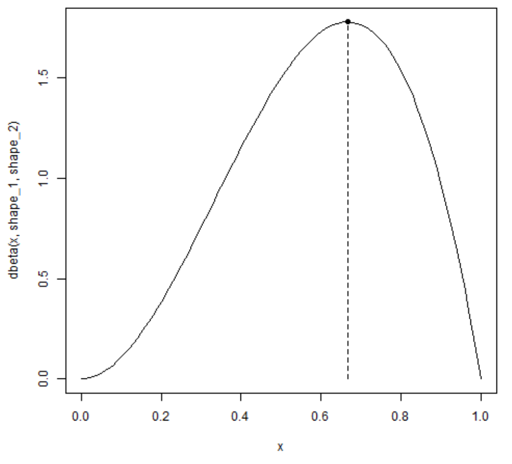
Find the maximum of a function

Now, find the maximum of this beta density

```
opt_beta <- optimize(dbeta, interval = c(0, 1), maximum = TRUE, shape
```

and visualize

```
curve(dbeta(x, shape_1, shape_2), xlim = range(x))
points(opt_beta$maximum, opt_beta$objective, pch = 20, cex = 1.5)
segments(opt_beta$maximum, 0, opt_beta$maximum, opt_beta$objective, l
```



Do Maximum Likelihood Estimation (MLE)

You generate data from a gamma distribution with given parameters.

```
nsim <- 10000
x <- rgamma(nsim, shape = 3, rate = 1.5)
```

First, you'll compare empirical mean and variance with the theoretical quantities

```
mean(x) ; var(x)
```

```
## [1] 1.98104
```

```
## [1] 1.233706
```

versus

```
3 * (1/1.5) ;  3 * (1/1.5)^2
```

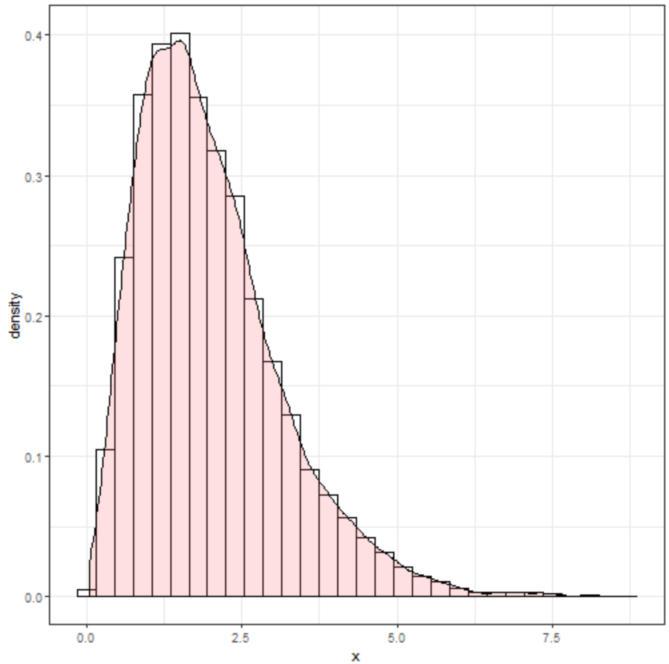
```
## [1] 2
```

```
## [1] 1.333333
```

Do Maximum Likelihood Estimation (MLE)

Picture the simulated data

```
d <- data.frame(x)
ggplot(d, aes(x = x)) + geom_histogram(aes(y = ..density..), binwidth
  geom_density(alpha = .2, fill = "#FF6666") +
  theme_bw()
```



Do Maximum Likelihood Estimation (MLE)

With Maximum Likelihood Estimation:

- $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)^t$ is the parameter vector
- the likelihood function

$$\mathcal{L}(\boldsymbol{\theta}; y_1, \ldots, y_n) = \prod_{i=1}^{n} f(y_i; \boldsymbol{\theta})$$

- the log-likelihood function

$$L(\boldsymbol{\theta}; y_1, \ldots, y_n) = \log \prod_{i=1}^{n} f(y_i; \boldsymbol{\theta}) = \sum_{i=1}^{n} \log f(y_i; \boldsymbol{\theta})$$

- the MLE $\hat{\boldsymbol{\theta}}$ maximizes $L(\boldsymbol{\theta}; y_1, \ldots, y_n)$ (or: $\mathcal{L}(\boldsymbol{\theta}; y_1, \ldots, y_n)$).

Do Maximum Likelihood Estimation (MLE)

The goal is to fit a gamma density to the generated data.

```
f <- function(p,x){
    -sum(dgamma(x, shape = p[1], rate = p[2], log = TRUE))
}
```

and optimize with `nlm`

```
nlm(f, c(1, 1), x = x)
```

```
## $minimum
## [1] 14178.54
##
## $estimate
## [1] 3.121014 1.575442
##
## $gradient
## [1] -0.003308669  0.003693534
##
## $code
## [1] 1
##
```

Do Maximum Likelihood Estimation (MLE)

Alternatively, optimize with `optim`

```
optim(c(1, 1), f, x = x)
```

```
## $par
## [1] 3.121460 1.575785
##
## $value
## [1] 14178.54
##
## $counts
## function gradient
##       67       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

MLE with fitdistr()

Alternatively, you can use `fitdistr` from the `MASS` library.

```
library(MASS)
fitdistr(x, dens = "gamma")
```

```
##       shape          rate    
##   3.12104759   1.57545939 
##  (0.04199349) (0.02299714)
```

R challenge

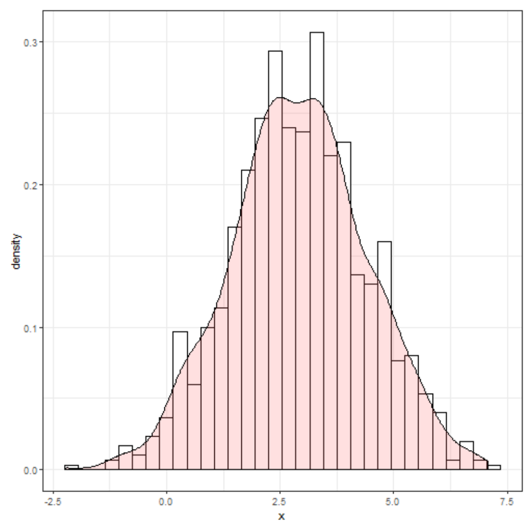
Now it's your turn:

1. generate observations from a normal distribution with given μ and σ^2
2. plot a histogram of the data
3. fit the normal distribution using MLE to the simulated data
4. examine the effect of decreasing/increasing the sample size.

R challenge solved

```
nsim <- 1000
x <- rnorm(nsim, mean = 3, sd = 1.5)
```

```
d <- data.frame(x)
ggplot(d, aes(x = x)) + geom_histogram(aes(y = ..density..), binwidth
  geom_density(alpha = .2, fill = "#FF6666") +
  theme_bw()
```



R challenge solved

```
fitdistr(x, dens = "normal")
```

```
##        mean           sd
##   3.04897207   1.49065502
##  (0.04713865) (0.03333206)
```

```
f <- function(p, x){
    -sum(dnorm(x, mean = p[1], sd = p[2], log = TRUE))
}

optim(c(1, 1), f, x = x)
```

```
## $par
## [1] 3.048983 1.490632
##
## $value
## [1] 1818.154
##
## $counts
## function gradient
##       73       NA
##
```

Functional programming

What is a functional?

A functional is a function which takes a function as input.

Example: the integral operator

$$\int_0^1 : C([0,1]) \to \mathbb{R}, f \mapsto \int_0^1 f(x)\,dx,$$

where $C([0,1])$ is the set of continuous functions on $[0,1]$.

```r
f <- function(x){
  x^2
}
integrate(f, lower = 0, upper = 1)
```

```
## 0.3333333 with absolute error < 3.7e-15
```

Why functional programming?

This approach offers:

- an intuitive alternative for loops
- code that is easy to read and interpret
- easily modifiable and reusable code
- no need to copy/paste the same code many times
- if you use something twice, put it in a function.

The purrr package in R

`purrr` is the tidyverse package for functional programming.

```
# install.packages(purrr)
require(purrr)
```

map()

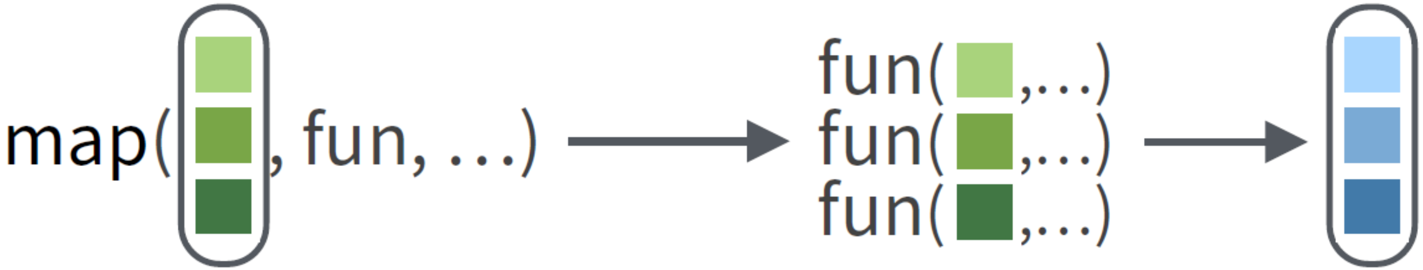


Illustration from the purrr cheat sheet

The output of `map` is stored in a `list`.

R challenge

You will now generate one draw from the distribution of the aggregate loss random variable $L = \sum_{j=1}^{N} Y_j$.

Let $N \sim \text{POI}(\lambda = 10)$ and $Y \sim \text{LogN}(\text{meanlog} = 9, \text{sdlog} = 1.75)$.

You will:

- set the seed at 1234 to reproduce the simulation; use `set.seed()`
- generate one draw from L.

R challenge solved

```r
# set the seed so we can reproduce the simulation
set.seed(1234)

expected_freq <- 10

# generate a single frequency from the poisson distribution
freq <- rpois(n = 1, lambda = expected_freq)
freq
```

```
## [1] 6
```

```r
# generate `freq` severities
# each severity represents the ultimate value of 1 claim
# we will use the lognormal distribution here
sev <- rlnorm(n = freq, meanlog = 9, sdlog = 1.75)
sev
```

```
## [1] 14046.732 15195.521  2256.730  8625.924  9874.455 98712.420
```

R challenge

You now want to generate 1000 draws from the distribution of L:

1. you generate $N_1, \ldots, N_{1000}$
2. given N_i, you generate $Y_{i1}, \ldots, Y_{iN_i}$.

Use `map` from the `purrr` package.

R challenge solved

```r
library(purrr)

# number of sims
n_sim <- 1000

# generate frequencies from the poisson distribution
freqs <- rpois(n = n_sim, lambda = expected_freq)
head(freqs)
```

```
## [1] 13  8  7  6 12 14
```

```r
# generate `freq` severities
# each severity represents the ultimate value of 1 claim
obs <- purrr::map(freqs, function(freq) rlnorm(n = freq, meanlog = 9,
head(obs)
```

```
## [[1]]
##  [1]  80356.4009  17897.3331 340100.5359  17255.5123   6110.0380
##  [6]   4648.1456   1335.0300   4503.5245  43360.1117   4187.3333
## [11]    557.3474  19858.6878   3414.7065
##
## [[2]]
```

R challenge

Now, you'll tidy the data using the following instructions.

```
library(purrr); library(dplyr)
i <- 0
obs <- purrr::map(obs, function(sev) {
  i <<- i + 1
  tibble(
    ob = i,
    sev = sev
  )
})

obs <- dplyr::bind_rows(obs)
head(obs, 10)
```

```
## # A tibble: 10 x 2
##        ob     sev
##     <dbl>   <dbl>
##  1      1  80356.
##  2      1  17897.
##  3      1 340101.
##  4      1  17256.
```

R challenge

Finally,

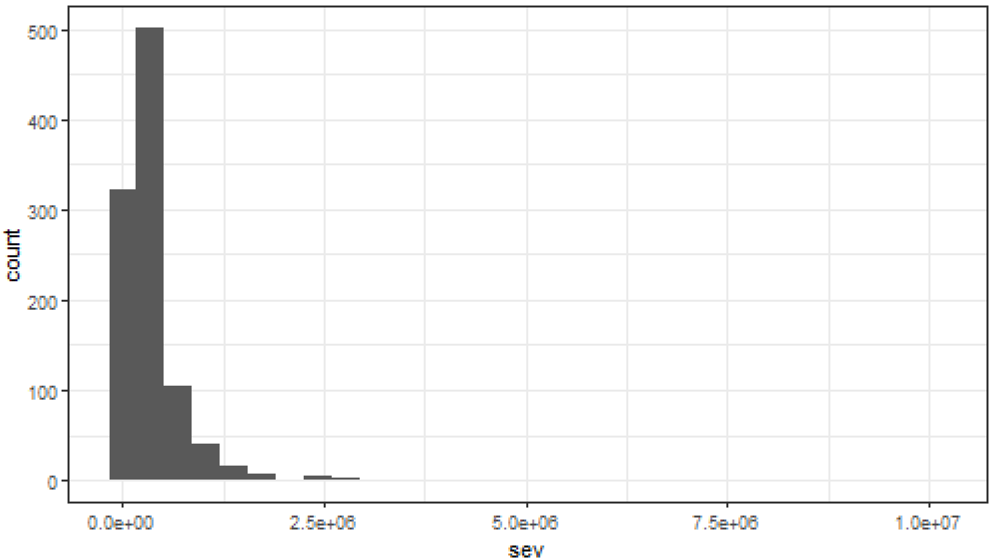
- using the tidy data structure, calculate the 1000 draws from L
- visualize the empirical distribution of L.

R challenge solved

```
obs_total <- obs %>%
  group_by(ob) %>%
  summarise(sev = sum(sev))

ggplot(obs_total, aes(sev)) + geom_histogram() + theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



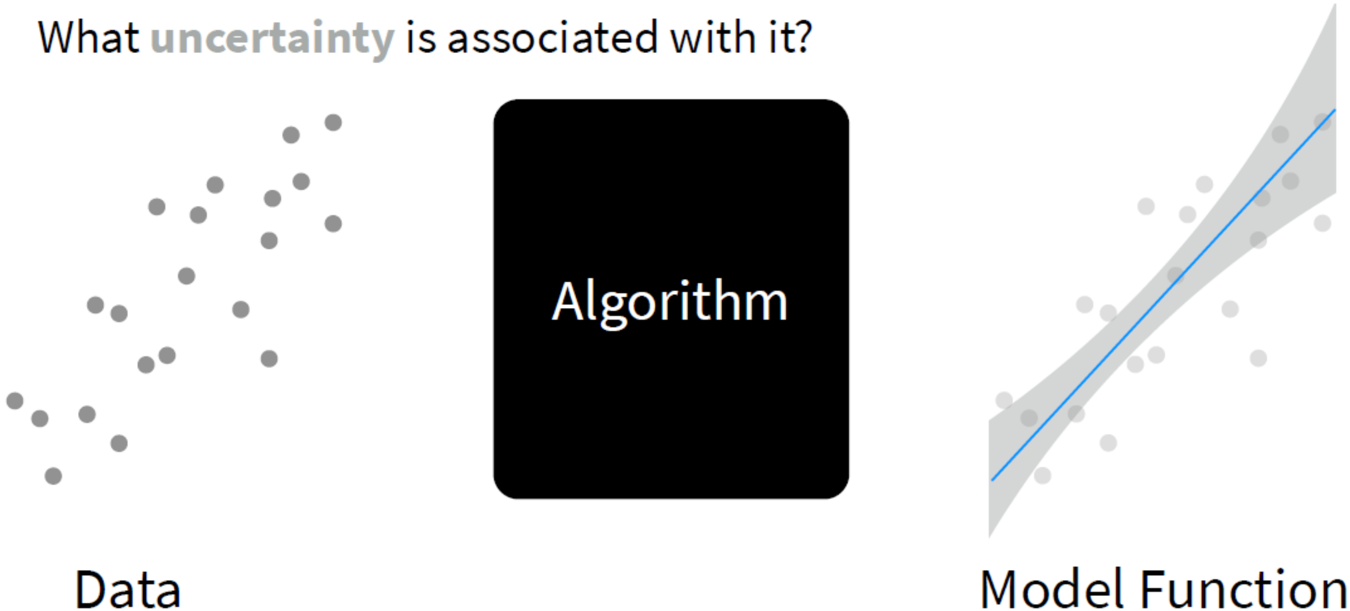
Fitting models to data

Models

Models

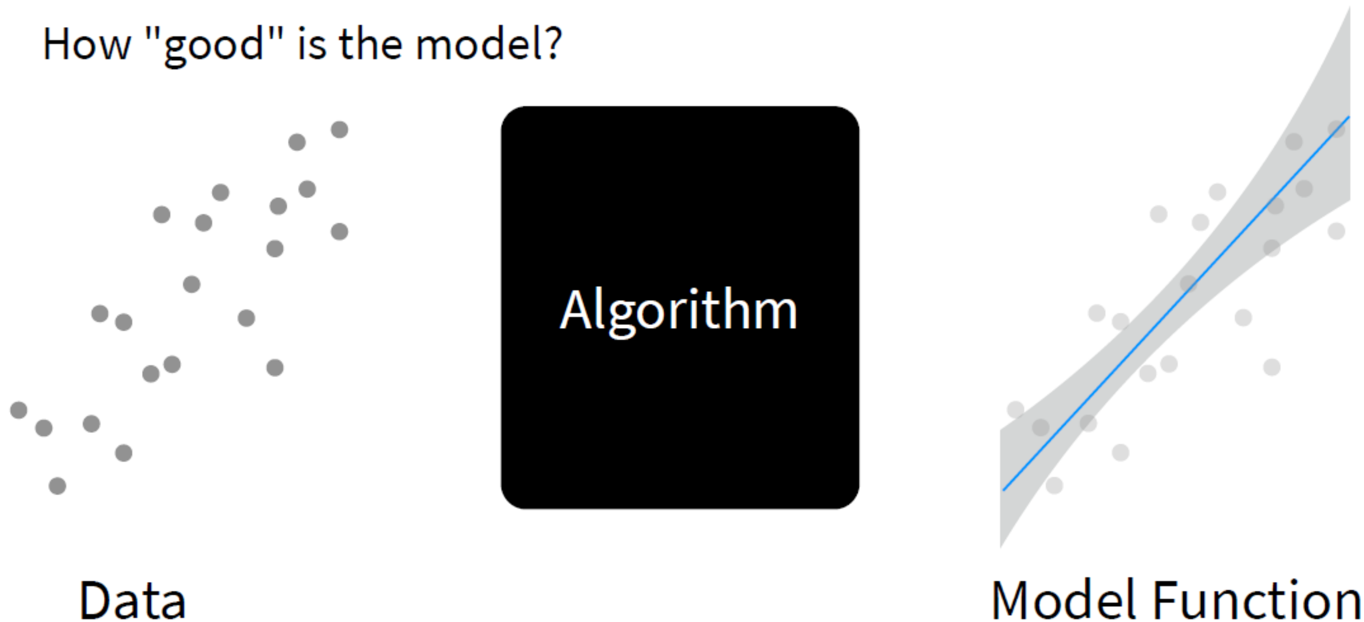
What is the **model function**?

Models

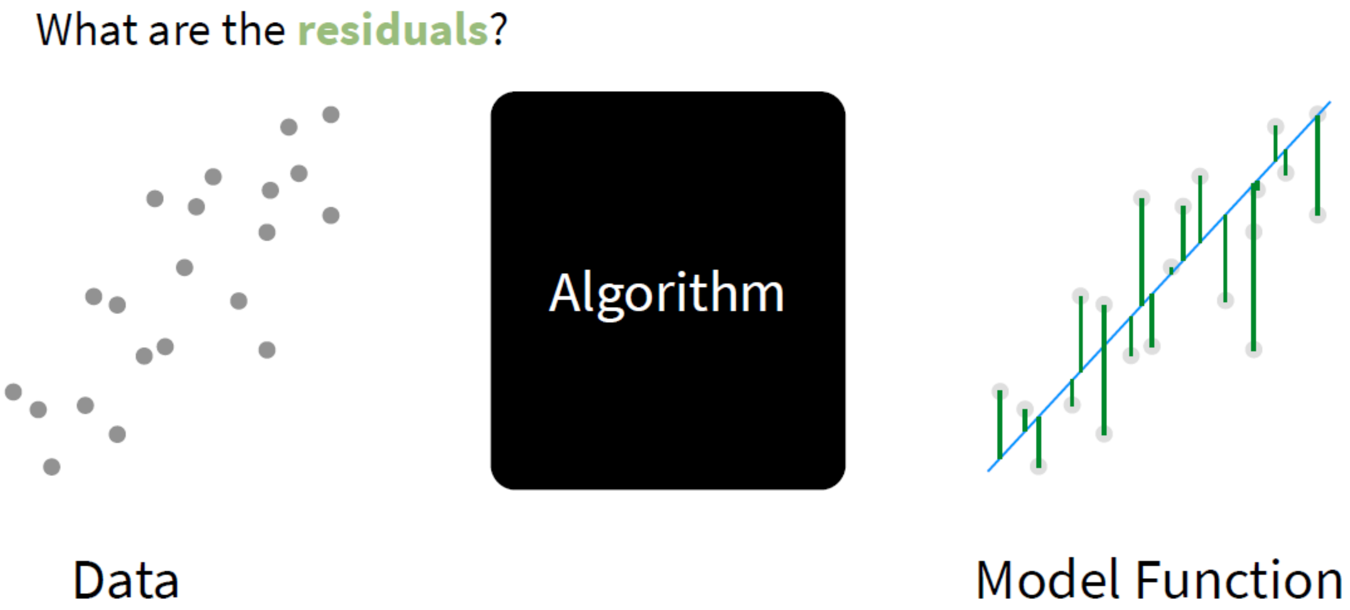


CC by RStudio

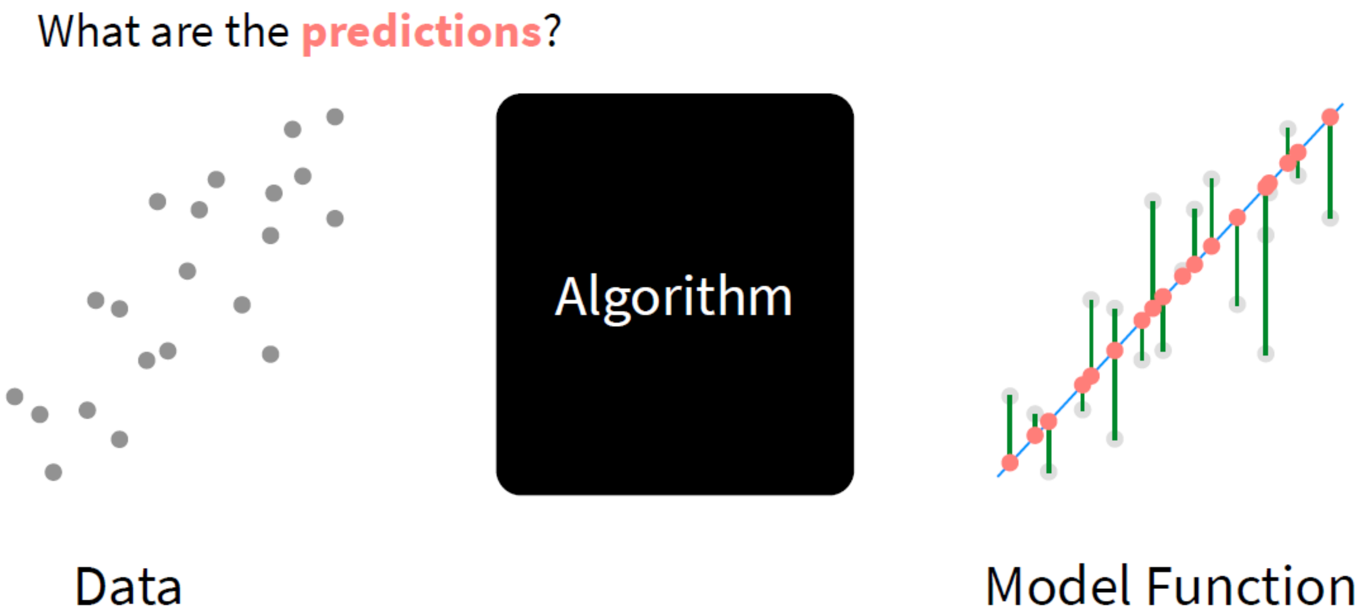
Models



Models



Models



Analyzing credit card applicants' data

Your journey as a model builder in R will start from studying **linear models** and the use of the `lm` function.

Hereto:

- you analyze Ford dealership data as registered in Milwaukee, September/October 1990
- data on 62 credit card applicants are available, including the car purchase price y (= the target, or response) and the applicant's annual income x (= feature, or covariate)
- data are in the `.csv` file `car_price`.

R challenge

Using what you've learned during Day 1:

- load the `car_price.csv` data
- with `ggplot` visualize `price` versus `income/1000`.

Explore the data

Get the data - use the instructions covered during Day 1.

```
path <- dirname(rstudioapi::getActiveDocumentContext()$path)
setwd(path)
```

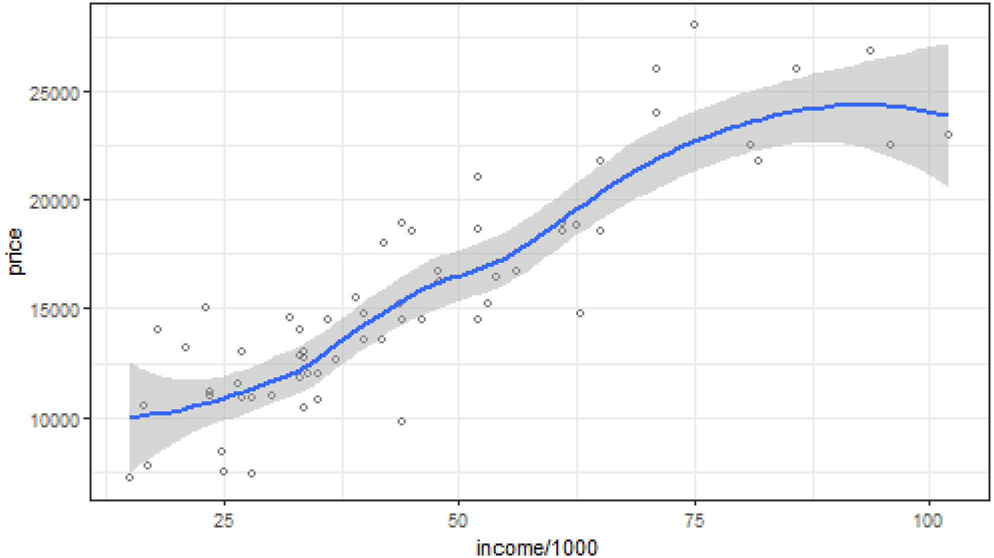
```
car_price <- read.csv("data/car_price.csv", header = TRUE)
```

Explore the data

You inspect the data with a scatterplot of `income` versus `price`:

```
ggplot(car_price, aes(x = income/1000, y = price)) +
  theme_bw() +
  geom_point(shape = 1, alpha = 1/2) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



A simple linear regression fit

You will now fit a simple regression model with `income` as predictor to purchase `price`.

That is:

$$y_i = \beta_0 + \beta_1 \cdot x_i + \epsilon_i,$$

where y_i is the car `price` for observation i, x_i the corresponding `income` and ϵ_i an error term.

β_0 is the intercept and β_1 the slope.

A formula for the model equation

Formula only needs to include the response and predictors

$$y = \beta_0 + \beta_1 \cdot x + \epsilon$$

becomes

```
y ~ x
```

Fitting a linear model then becomes

```
lm(price ~ income, data = car_price)
```

lm()

You assign the output of the `lm` function to the object `lm_car`

```
lm_car <- lm(price ~ income, data = car_price)
```

Now you inspect the results:

```
class(lm_car)
```

```
## [1] "lm"
```

```
names(lm_car)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

lm()

You inspect a summary of the fitted model

```
summary(lm_car)
```

```
##
## Call:
## lm(formula = price ~ income, data = car_price)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5364.6 -1184.7  -251.4  1334.0  6284.4
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 5.866e+03  7.498e+02   7.824  9.8e-11 ***
## income      2.113e-01  1.508e-02  14.009  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2459 on 60 degrees of freedom
## Multiple R-squared:  0.7659,    Adjusted R-squared:  0.762
## F-statistic: 196.3 on 1 and 60 DF,  p-value: < 2.2e-16
```

lm()

Some useful arguments: 'coefficients', 'residuals', 'fitted.values', 'model'

```
lm_car$coef
```

```
##  (Intercept)       income
## 5866.3339035    0.2113239
```

```
head(lm_car$residuals)
```

```
##          1          2          3          4          5          6
##  -719.2898  1146.8218  4329.8359  3258.0624 -3649.4314  -483.5697
```

```
head(lm_car$fitted.values)
```

```
##         1         2         3         4         5         6
## 14319.290  9353.178  9670.164 14741.938 11149.431 22983.570
```

Utility functions

Linear models in R come with a bunch of utility functions:

- `coef()` for retrieving coefficients
- `fitted()` for fitted values
- `residuals()` for residuals
- `summary()`, `plot()`, `predict()` and so on.

Once you master the utility functions, you'll be able to use them in the same way for model objects returned by `glm()`, `gam()`, and many others.

Visualize the `lm()` fit

To visualize this linear model fit you can use the built-in `plot` function, applied to object `lm_car`

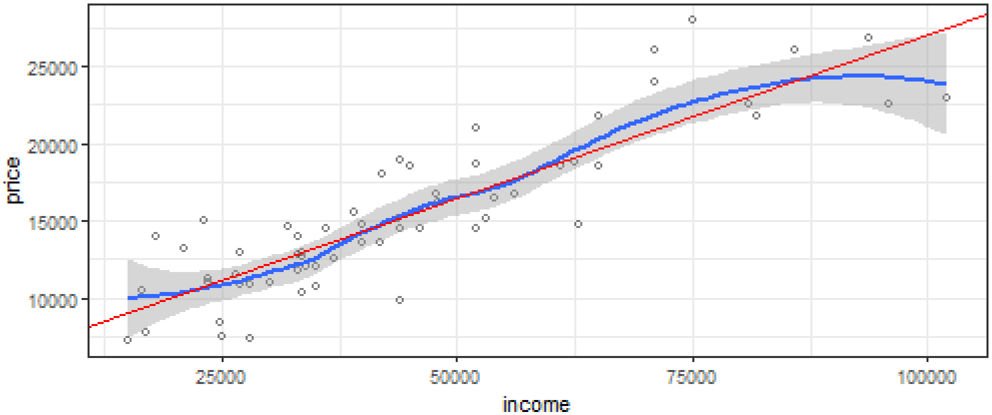
```
plot(lm_car)
```

Visualize the `lm()` fit

Or you can create your own plot

```
ggplot(car_price, aes(x = income, y = price)) +
  theme_bw() +
  geom_point(shape = 1, alpha = 1/2) +
  geom_smooth() +
  geom_abline(intercept = lm_car$coef[1], slope = lm_car$coef[2], col
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



predict()

Making predictions for new applicants:

```
new <- data.frame(income = 60000) # set up a new data frame
new_pred <- predict(lm_car, newdata = new) # call predict
new_pred
```

```
##        1
## 18545.77
```

R challenge

You'll now step from simple to multiple regression:

1. load the `pollution.csv` data set
2. read the data description here
3. create data frames of related covariates and visualize.

R challenge solved

First, you load the data

```
pollution <- read.csv("data/pollution.csv", header = TRUE)
```

Then, you combine related features

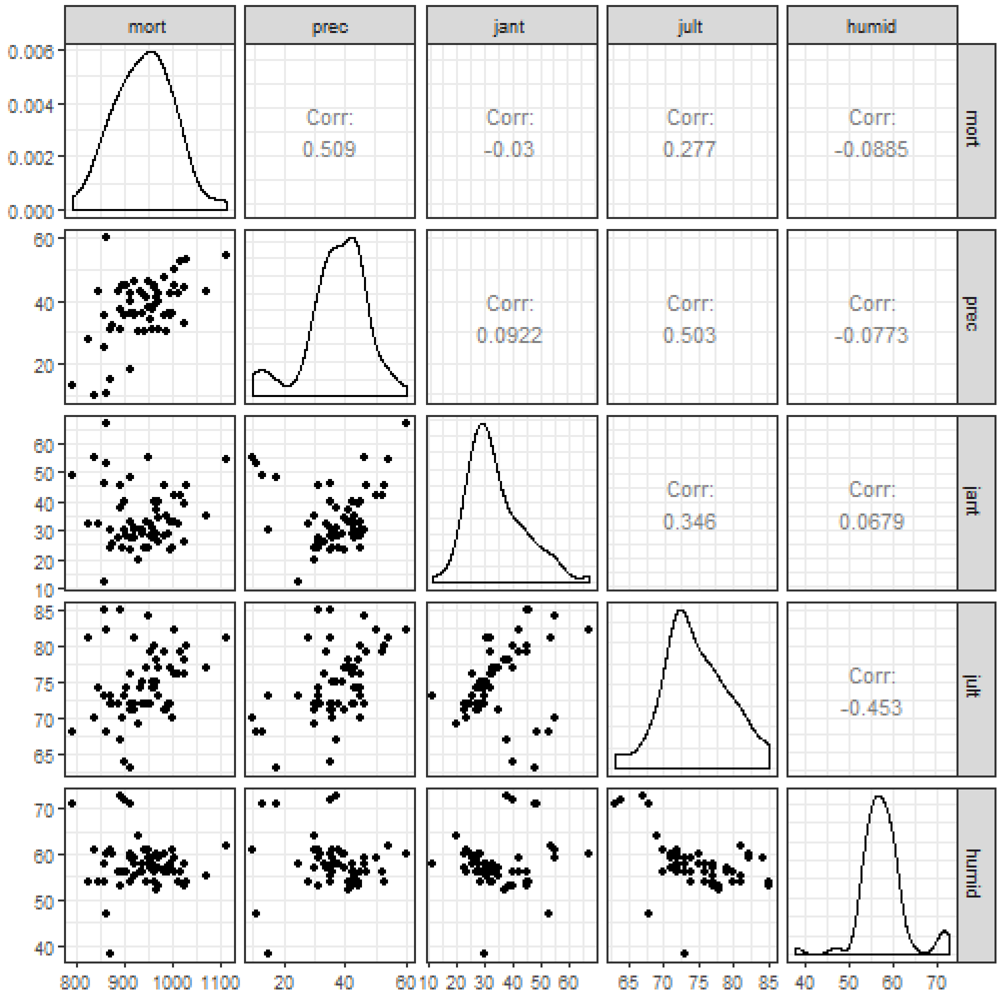
```
# weather effects
mort_poll_1 <- pollution %>% select(c(mort, prec, jant, jult, humid))
# socio-economic vars
mort_poll_2 <- pollution %>% select(c(mort, ovr65, popn, educ, hous,
# pollution effects
mort_poll_3 <- pollution %>% select(c(mort, hc, nox, so2))
```

R challenge solved

```
library(GGally)
```

```
g_1 <- ggpairs(mort_poll_1) + theme_bw()
```

```
g_1
```



mort
prec
jant
jult
humid
Corr:
0.509
Corr:
-0.03
Corr:
0.277
Corr:
-0.0885
Corr:
0.0922
Corr:
0.503
Corr:
-0.0773
Corr:
0.346
Corr:
0.0679
Corr:
-0.453
0.006
0.004
0.002
0.000
800 900 1000 1100
20 40 60
10 20 30 40 50 60
65 70 75 80 85
40 50 60 70

R challenge

Now, you'll:

- build a linear regression model to explain `mort` as a function of `so2` and `educ`
- inspect the model and fit.

R challenge solved

The linear regression of `mort` versus `so2` and `educ`:

```
lm_mort <- lm(mort ~ educ + so2, data = pollution)
summary(lm_mort)
```

```
##
## Call:
## lm(formula = mort ~ educ + so2, data = pollution)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -136.558  -30.371   -7.731   34.481  148.803
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1274.6024    89.7611  14.200  < 2e-16 ***
## educ         -32.0172     8.0195  -3.992 0.000189 ***
## so2            0.3179     0.1069   2.973 0.004321 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 50.62 on 57 degrees of freedom
```

R challenge solved

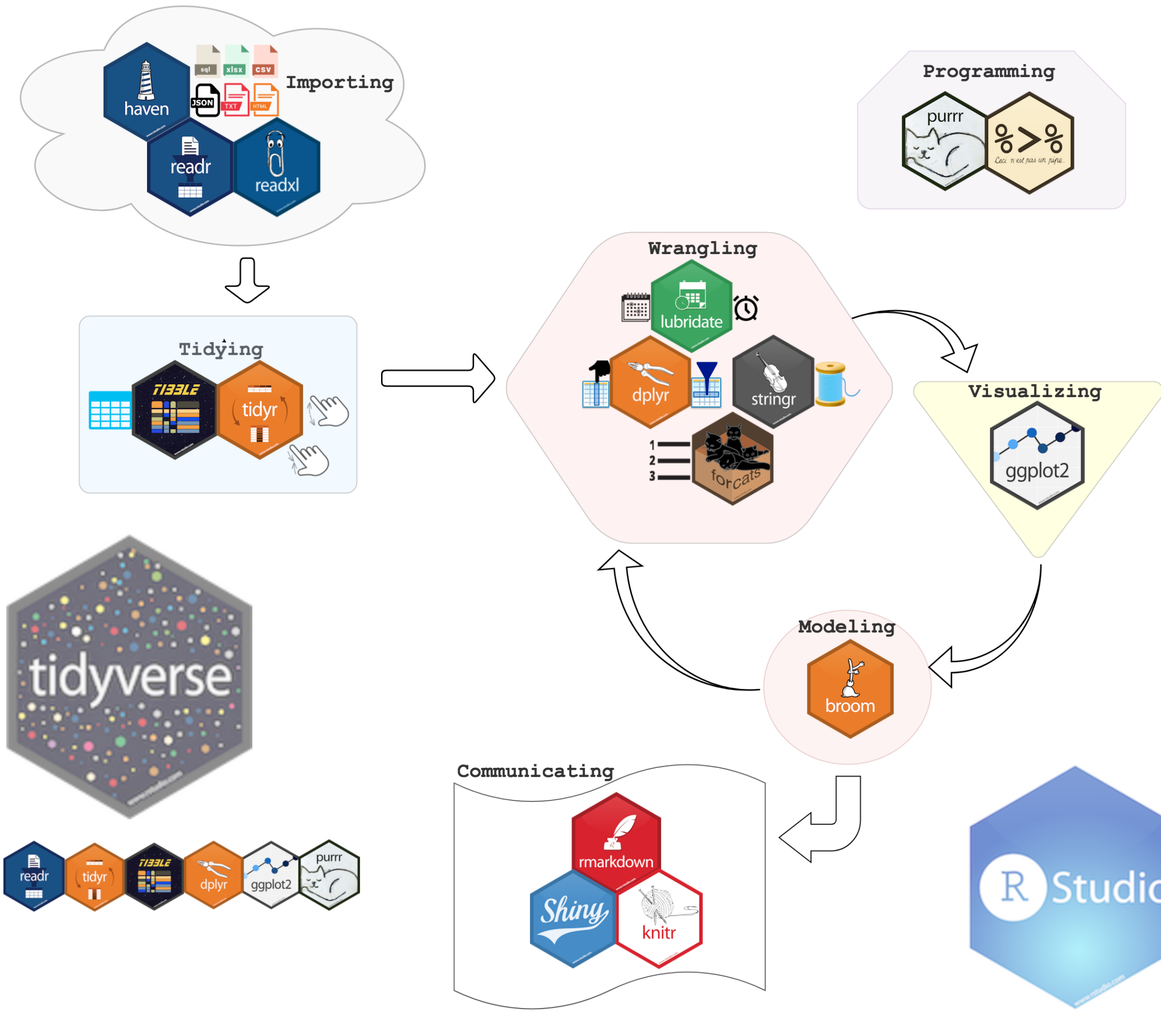
The linear regression model including all available covariates:

```
lm_mort_all <- lm(mort ~ ., data = pollution)
summary(lm_mort_all)
```

```
##
## Call:
## lm(formula = mort ~ ., data = pollution)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -68.066 -18.017   0.912  19.224  86.961
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.764e+03  4.373e+02   4.034 0.000215 ***
## prec         1.905e+00  9.237e-01   2.063 0.045071 *
## jant        -1.938e+00  1.108e+00  -1.748 0.087413 .
## jult        -3.100e+00  1.902e+00  -1.630 0.110159
## ovr65       -9.065e+00  8.486e+00  -1.068 0.291230
## popn        -1.068e+02  6.978e+01  -1.531 0.132952
## educ        -1.716e+01  1.186e+01  -1.447 0.155085
```

Tidy your model output

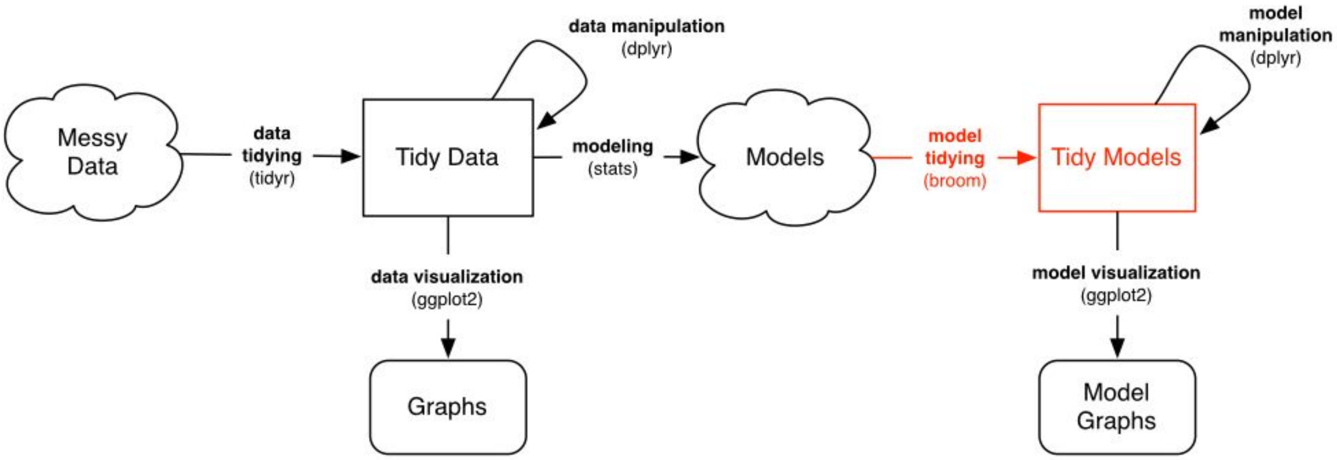
More from the tinyverse



Tidy your model output

The ouput of a model is not tidy, not even a data frame.

Use the `broom` package for tidy output.



The broom package

Broom includes three functions which work for most types of models (and can be extended to more):

1. `tidy()` - returns model coefficients, stats
2. `glance()` - returns model diagnostics
3. `augment()` - returns predictions, residuals, and other raw values.

tidy()

Returns useful model output as a data frame:

```
library(broom)
library(dplyr)
lm_car <- lm(price ~ income, data = car_price)
lm_car %>% tidy()
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) 5866.     750.          7.82 9.80e-11
## 2 income         0.211    0.0151     14.0  1.42e-20
```

Store the tibble and inspect

```
lm_fit <- lm_car %>% tidy()
```

glance()

Returns common model diagnostics as a data frame

```
lm_car %>% glance()
```

```
## # A tibble: 1 x 11
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BI
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>  <dbl> <dbl> <dbl
## 1     0.766         0.762 2459.      196. 1.42e-20     2  -571. 1148. 1154
## # ... with 2 more variables: deviance <dbl>, df.residual <int>
```

augment()

Returns data frame of model output related to original data points

```
lm_car %>% augment()
```

```
## # A tibble: 62 x 9
##    price income .fitted .se.fit .resid   .hat .sigma  .cooksd .std.resid
##    <int>  <int>   <dbl>   <dbl>  <dbl>  <dbl>  <dbl>    <dbl>      <dbl>
##  1 13600  40000  14319.    322.  -719. 0.0171  2478. 0.000759     -0.295
##  2 10500  16500   9353.    534.  1147. 0.0471  2475. 0.00564       0.478
##  3 14000  18000   9670.    516.  4330. 0.0439  2412. 0.0745        1.80
##  4 18000  42000  14742.    316.  3258. 0.0165  2443. 0.0150        1.34
##  5  7500  25000  11149.    436. -3649. 0.0315  2432. 0.0369       -1.51
##  6 22500  81000  22984.    624.  -484. 0.0644  2479. 0.00142      -0.203
##  7 21000  52000  16855.    329.  4145. 0.0179  2419. 0.0263        1.70
##  8 11000  30000  12206.    387. -1206. 0.0248  2475. 0.00314      -0.497
##  9 14500  44000  15165.    313.  -665. 0.0162  2478. 0.000611     -0.272
## 10 14800  40000  14319.    322.   481. 0.0171  2479. 0.000339      0.197
## # ... with 52 more rows
```

augment()

Combined with the original `car_price` data frame

```
lm_car %>% augment(car_price)
```

```
## # A tibble: 62 x 14
##      sex income   age married children college price .fitted .se.fit .resi
##    <int>  <int> <int>   <int>    <int>   <int> <int>   <dbl>   <dbl>  <dbl
##  1     1  40000    48       1        3       0 13600  14319.    322.  -719
##  2     0  16500    24       0        0       1 10500   9353.    534.  1147
##  3     1  18000    25       0        0       1 14000   9670.    516.  4330
##  4     1  42000    37       1        2       1 18000  14742.    316.  3258
##  5     0  25000    34       1        1       0  7500  11149.    436. -3649
##  6     1  81000    50       1        3       1 22500  22984.    624.  -484
##  7     1  52000    29       1        2       1 21000  16855.    329.  4145
##  8     0  30000    34       0        0       0 11000  12206.    387. -1206
##  9     0  44000    36       0        1       1 14500  15165.    313.  -665
## 10     1  40000    33       1        2       1 14800  14319.    322.    481
## # ... with 52 more rows, and 4 more variables: .hat <dbl>, .sigma <dbl>,
## #   .cooksd <dbl>, .std.resid <dbl>
```

R challenge

Use a pipe to model `price` against `income` for the `car_price` data set.

Then use `broom` and `dplyr` functions to extract:

1. the coefficient estimates and their related statistics
2. the `adj.r.squared` and `p.value` for the overall model.

R challenge solved

Here you go

```
res_lm <- car_price %>% lm(price ~ income, data = .) %>% tidy()
```

Now extract the coefficient estimates and their related stats

```
res_lm %>% select(c(term, estimate, std.error))
```

```
## # A tibble: 2 x 3
##   term        estimate std.error
##   <chr>          <dbl>     <dbl>
## 1 (Intercept) 5866.     750.
## 2 income         0.211    0.0151
```

R challenge solved

To get the `adj.r.squared` and `p.value` for the overall model

```
car_price %>% lm(price ~ income, data = .) %>% glance() %>% select(c
```

```
## # A tibble: 1 x 2
##   adj.r.squared  p.value
##           <dbl>    <dbl>
## 1         0.762 1.42e-20
```

R challenge

Let's explore the functionalities of `broom` for multiple linear regression:

1. load the `wages` data set using the instructions printed below
2. model `log(income)` against `education` and `height` and `sex`
3. can you interpret the coefficients?

```
library(modelr)
wages <- heights %>% filter(income > 0)
```

R challenge solved

Get the data, fit the linear model

```
library(modelr)
wages <- heights %>% filter(income > 0)
lm_fit <- lm(log(income) ~ education + height + sex, data = wages)
```

and investigate the fit

```
lm_fit %>% tidy()
```

```
## # A tibble: 4 x 5
##   term        estimate std.error statistic   p.value
##   <chr>          <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)  8.25      0.335       24.6  4.68e-127
## 2 education    0.148     0.00520     28.5  5.16e-166
## 3 height       0.00673   0.00479      1.40 1.61e-  1
## 4 sexfemale   -0.462     0.0389     -11.9  5.02e- 32
```

Multiple regression investigated

For factors, R treats the first level as the baseline level, e.g. the mean `log(income)` for a `male` is:

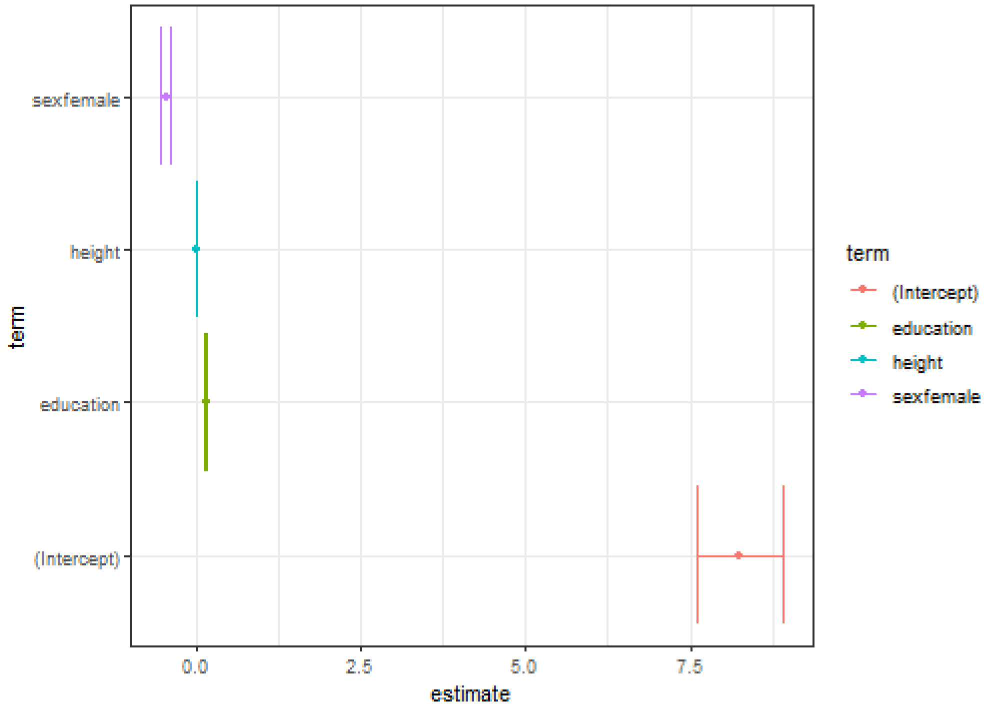
```
log(income) = 8.25 + 0.15 * education + 0 * height
```

Each additional level gets a coefficient that acts as an adjustment between the baseline level and the additional level, e.g. the mean income for a `female` is:

```
log(income) = 8.25 + 0.15 * education + 0 * height - 0.46
```

Visualize coefficients

```
td <- tidy(lm_fit, conf.int = TRUE)
ggplot(td, aes(estimate, term, color = term)) + geom_point() + geom_e
```



Popular model functions in R

function	package	fits
lm()	stats	linear models
glm()	stats	generalized linear models
gam()	mgcv	generalized additive models
glmnet()	glmnet	penalized linear models
rlm()	MASS	robust linear models
rpart()	rpart	trees
randomForest()	randomForest	random forests
xgboost()	xgboost	gradient boosting machines

Model objects working with broom

See the vignette to find out for which models `broom` is currently available

```
vignette("available-methods")
```

R challenge

You are now ready to work on a bigger R challenge:

- load the Boston Housing dataset from the course documentation, or via the `mlbench` package in R. Store the data as the object `housing`

```
library(mlbench)
data("BostonHousing")
```

- inspect the different types of variables present
- explore and visualize the distribution of our target variable `medv`
- explore and visualize the relation between `medv` and the variable `crim`.

R challenge solved

First, load the data

```
library(mlbench)
data("BostonHousing")
housing <- BostonHousing
```

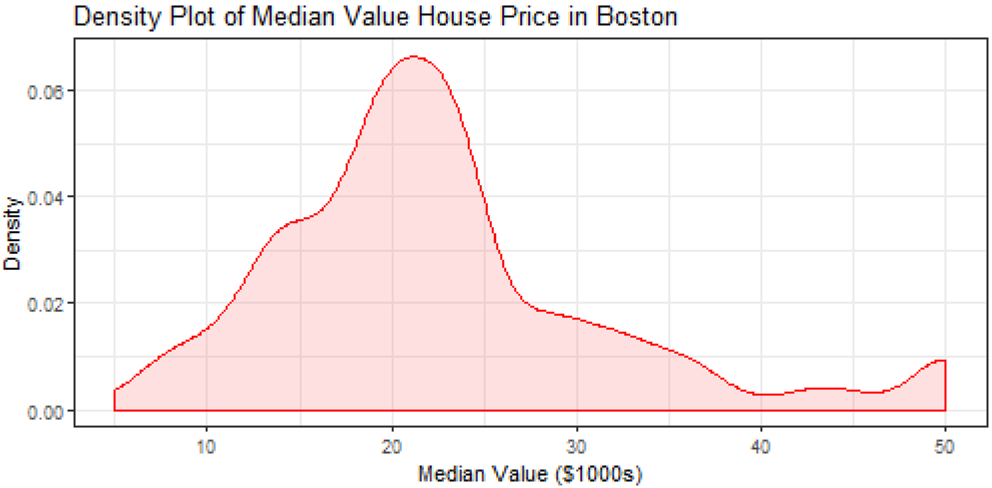
Inspect the data and figure out the data types

```
str(housing)
```

R challenge solved

Now, visualize the distribution of the (numeric) target variable

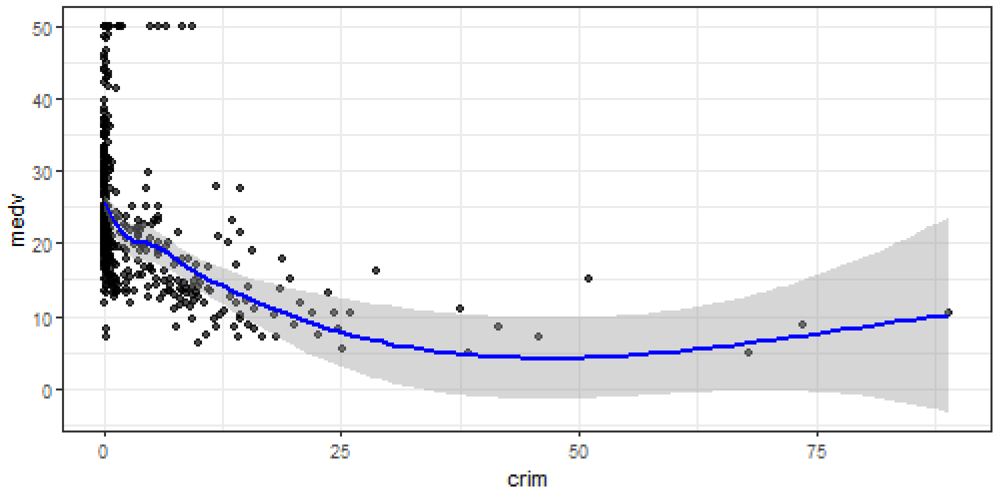
```
housing %>%
  ggplot(aes(x = medv)) +
  stat_density(alpha = 0.2, fill = "#FF6666", color = "red") +
  labs(x = "Median Value ($1000s)", y = "Density", title = "Density P
  theme_bw()
```



R challenge solved

Now, we plot the target `medv` versus covariate `crim`

```
housing %>%
  ggplot(aes(x = crim, y = medv)) +
  geom_point(alpha = 0.7) +
  geom_smooth(color = "blue") + theme_bw()
```



R challenge

Let's repeat the last graph produced for more features, i.e. `crim`, `rm`, `age`, `rad`, `tax` and `lstat`.

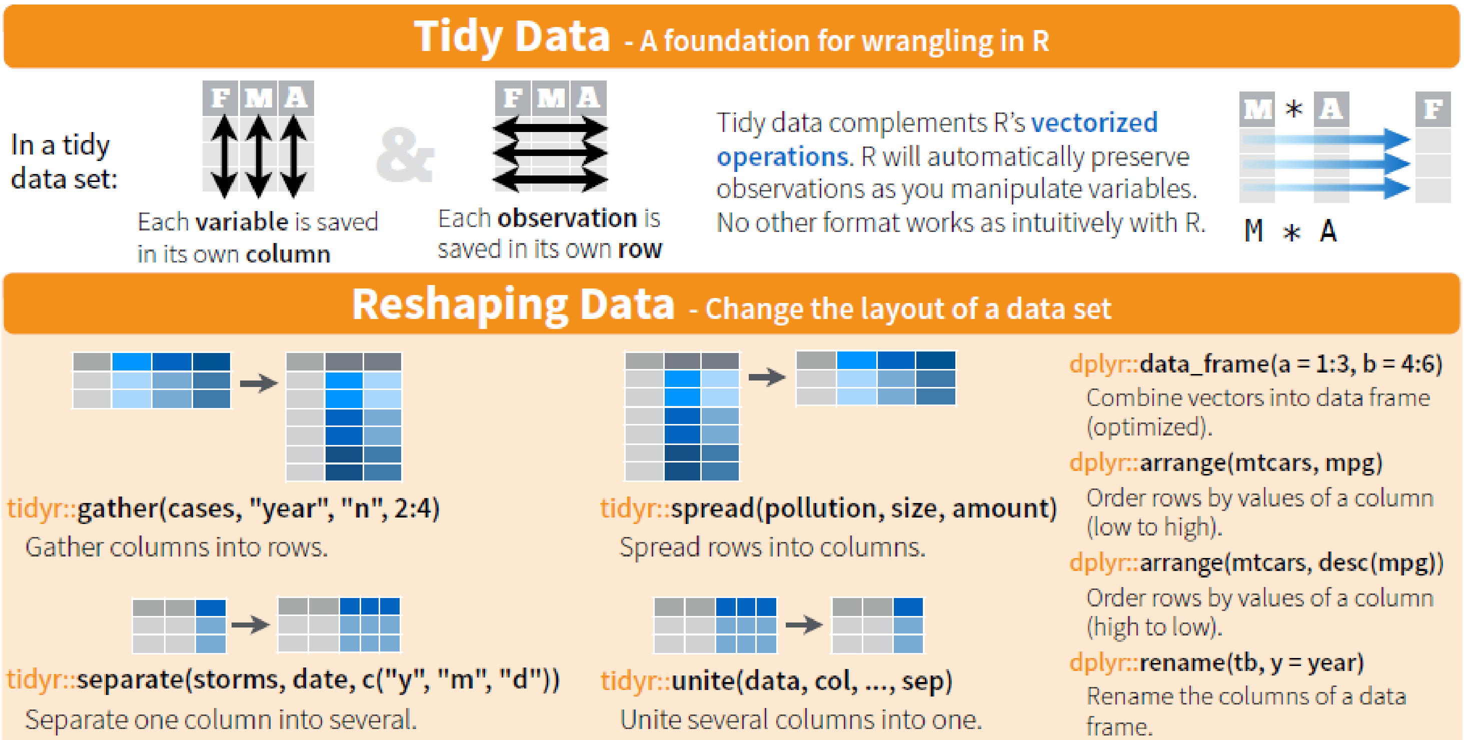
Transform the data from wide to long format ...

```
library(tidyr)
res <- housing %>% select(c(crim, rm, age, rad, tax, lstat, medv)) %>
  gather(., variable, value, crim:lstat, factor_key = TRUE)
```

In the `gather` function, the arguments are:

- data: the data object
- key: the name of new key column
- value: the name of new value column
- ...: the names of source columns that contain values
- factor_key: treat the new key column as a factor.

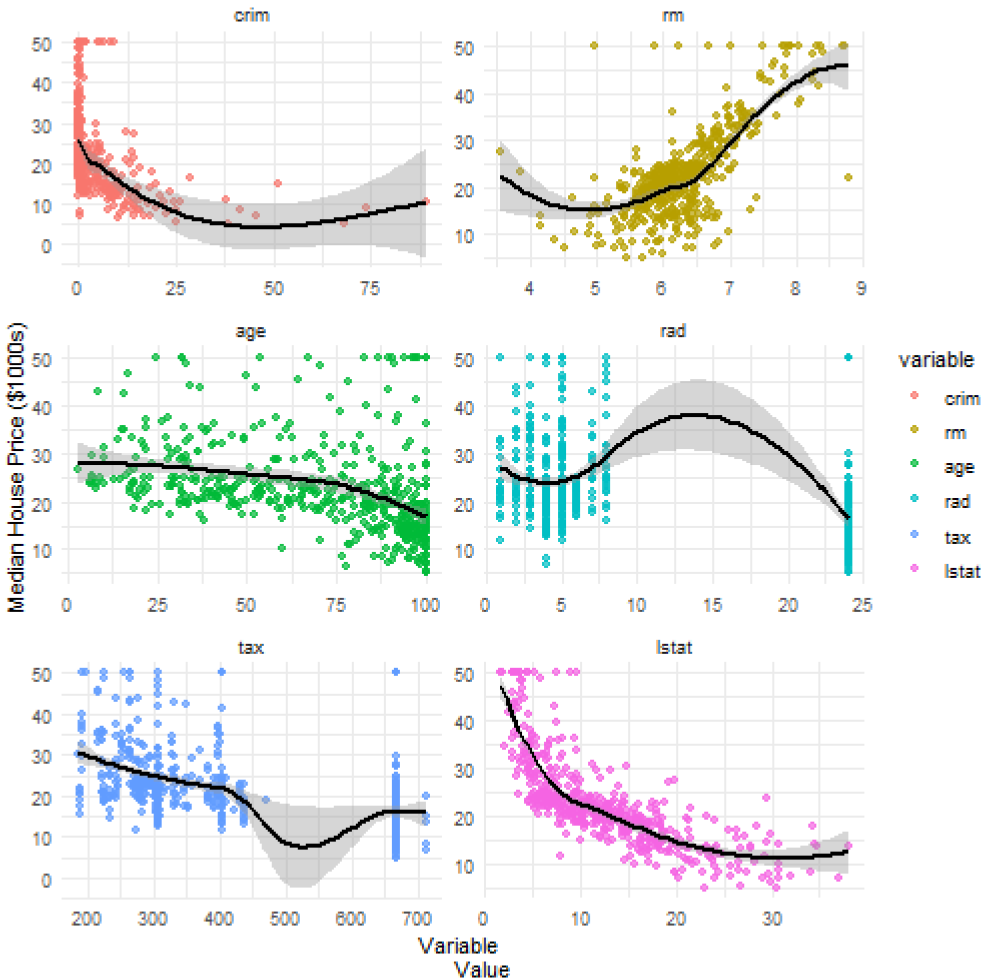
Reshaping data - from wide to long



R challenge solved

```
res %>%
  ggplot(aes(x = value, y = medv, colour = variable)) +
  geom_point(alpha = 0.7) +
  stat_smooth(color = "black") +
  facet_wrap( ~ variable, scales = "free", ncol = 2) +
  labs(x = "Variable
       Value", y = "Median House Price ($1000s)") +
  theme_minimal()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



R challenge

Now, you'll focus on the model building:

- set a seed of 123 and split your data into a train and test set using a 75/25 split

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
set.seed(123)
to_train <- createDataPartition(y = housing$medv, p = 0.75, list = FA
train <- housing %>% slice(., to_train)
test <- housing %>% slice(., -to_train)
```

R challenge solved

```
first_lm <- lm(medv ~ crim + rm + tax + lstat, data = train)
first_lm %>% glance()
```

```
## # A tibble: 1 x 11
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BI
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <int>  <dbl> <dbl> <dbl
## 1     0.672         0.668  5.19      192. 1.51e-89     5 -1165. 2342. 2366
## # ... with 2 more variables: deviance <dbl>, df.residual <int>
```

R challenge

We can see a few problems with our model immediately with a poor QQ plot in the tails and a relatively poor R-squared value.

Let us try another model:

- transform `medv` due to the positive skewness it exhibited
- examine the diagnostics for the model. What do you conclude? Is this an improvement on the first model?
- one assumption of a linear model is that the mean of the residuals is zero. You could try and test this.
- create a data frame of your predicted values and the original values. Plot this to visualize the performance of your model.

R challenge solved

```
second_lm <- lm(log(medv) ~ crim + rm + tax + lstat, data = train)

second_lm %>% glance()
```

```
## # A tibble: 1 x 11
##   r.squared adj.r.squared sigma statistic   p.value    df logLik   AIC
##       <dbl>         <dbl> <dbl>     <dbl>     <dbl> <int>  <dbl> <dbl>
## 1     0.735         0.733 0.206      261. 3.66e-107     5   64.5 -117.
## # ... with 3 more variables: BIC <dbl>, deviance <dbl>, df.residual <int>
```

```
mean(second_lm$residuals)
```

```
## [1] -3.412178e-18
```

R challenge solved

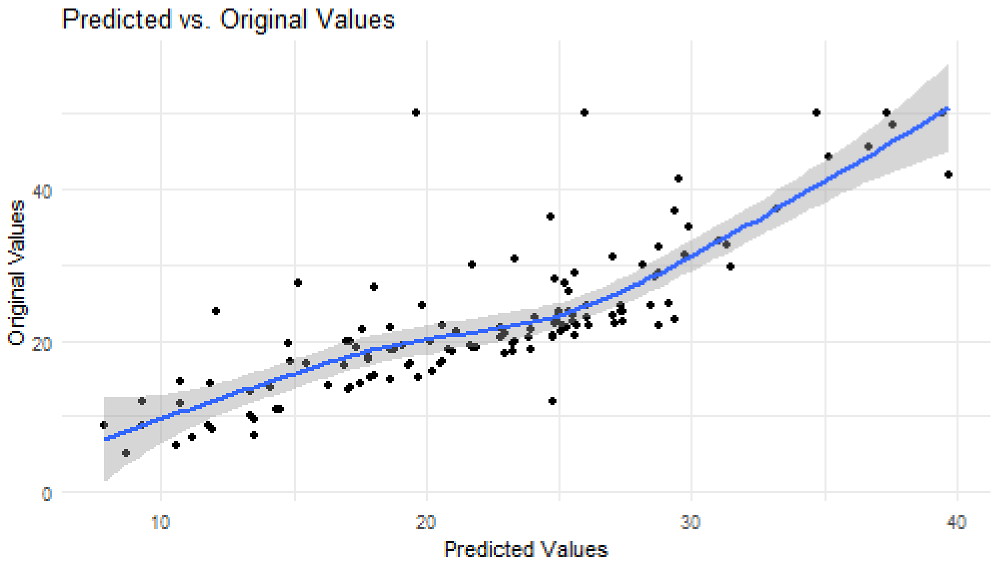
```
predicted <- predict(second_lm, newdata = test)

results <- data.frame(predicted = exp(predicted), original = test$mec
```

R challenge solved

```
results %>%
  ggplot(aes(x = predicted, y = original)) +
  geom_point() +
  stat_smooth() +
  labs(x = "Predicted Values", y = "Original Values", title = "Predic
  theme_minimal()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Rproject

Rproject

RStudio projects allow to organize your programming code.

An RProject file manages:

- working directory
- R history (.Rhistory)
- Startup code (.Rprofile)
- Startup data (.RData)

Rprojects improve collaboration and reproducibility, as these elements will be identical for all users.

Create a new Rproject by clicking: **File** > **New project…**

.Rprofile

At startup, R sources **.Rprofile** in the working directory of a project.

- Load packages required in the project.

```
require(ggplot2)
```

- Specify a company wide plotting styles.

```
company_green <- "#87D37C"

plot_style <- theme_bw() +
    theme(plot.title = element_text(size = 20,
                                    hjust = 0.5))
```

- Source custom functions required for the project.

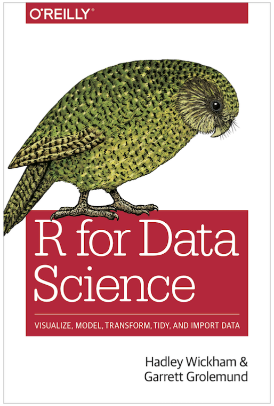
```
source(functions.R)
```

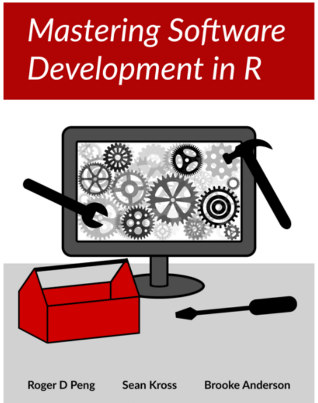
More resources

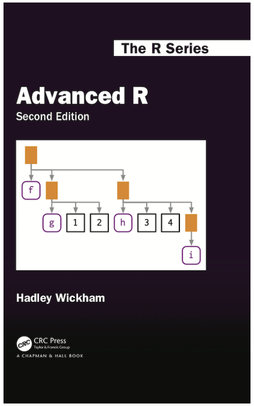
R for Data Science focuses on using the tidyverse for consistent R programming.

Mastering Software Development in R explains many common R functions through intuitive examples.

Advanced R helps you to master the R languague.







Thanks!

Slides created via the R package **xaringan**.