BST02: Using R for Statistics in Medical Research

Part D: Statistics with R

Nicole Erler

Department of Biostatistics, Erasmus Medical Center

✉ n.erler@erasmusmc.nl

24 - 28 February 2020

In this Section

- ► Common statistical tests
  - ► for continuous data
  - ► for categorical data

- ► (Generalized) linear regression

- ► Useful functions for regression models

- ► Modelling non-linear effects

t-test: `t.test()`

One-sample t-test

► compares the **mean of one sample** with a fixed value μ

t-test: `t.test()`

One-sample t-test

► compares the **mean of one sample** with a fixed value μ

Two sample / independent samples t-test

► compares the **difference between the means** of two samples with a fixed value μ

t-test: `t.test()`

One-sample t-test

► compares the **mean of one sample** with a fixed value μ

Two sample / independent samples t-test

► compares the **difference between the means** of two samples with a fixed value μ

Related samples t-test

► compares the **mean of the difference** between related observations with a fixed value μ (same as one-sample t-test)

Wilcoxon Test: `wilcox.test()`

Wilcoxon Signed Rank Test

▶ tests if **one sample** (or the difference between two paired samples) is **symmetric about** μ

Wilcoxon Test: `wilcox.test()`

Wilcoxon Signed Rank Test

► tests if **one sample** (or the difference between two paired samples) is **symmetric about** μ

Wilcoxon Rank Sum Test / Mann-Whitney test

► test for a **location shift between the distributions** of two independent samples

See also BBR Sections 7.2 & 7.3 (http://hbiostat.org/doc/bbr.pdf)

Kruskal-Wallis Rank Sum Test: `kruskal.test()`

► **extension** of the Wilcoxon rank sum test for **more than two groups**

► test for a **difference in location** of a continuous variable between multiple groups

► the **Wilcoxon rank sum test is a special case** of the Kruskal-Wallis rank sum test

Other tests for continuous data

► **Kolmogorov-Smirnov Test:** `ks.test()`
  tests if two samples are drawn from the same continuous
  distribution

► **Shapiro-Wilk Normality Test:** `shapiro.test()`

► **Friedman Rank Sum Test:** `friedman.test()`
  non-parametric test for two or more related samples

► …

Tests for Continuous Data

Demo

► Tests for Continuous Data
`R` `html`

Tests for Categorical Data / Proportions

One-sample Proportion Test

- ► tests if the **proportion in one sample** is equal to a fixed value *p*
- ► prop.test() and binom.test()

Tests for Categorical Data / Proportions

One-sample Proportion Test

- ▶ tests if the **proportion in one sample** is equal to a fixed value *p*
- ▶ `prop.test()` and `binom.test()`

Tests for Proportions in Multiple (independent) Groups

- ▶ tests if the **proportions in several samples** are equal
- ▶ `chisq.test()` and `fisher.test()` (when there are cells with 0)

See also BBR Sections 5.7 & 6 (http://hbiostat.org/doc/bbr.pdf)

Tests for Categorical Data / Proportions

Related Samples: McNemar Test

▶ Tests for **symmetry** in a 2×2 table
▶ `mcnemar.test()`

Tests for Categorical Data / Proportions

Related Samples: McNemar Test

- ▶ Tests for **symmetry** in a 2×2 table
- ▶ `mcnemar.test()`

3-Dimensional Contingency Table

- ▶ Cochrane-Mantel-Haenszel Test
- ▶ χ^2 test for **independence** of two nominal variables **within each stratum**
- ▶ `mantelhaen.test()`

Tests for Categorical Data

Demo
- ► Tests for Categorical Data
  `R` `html`

Practical
- ► Statistical Tests
  `html`

Useful Functions: Statistical Tests

Continuous Outcomes
- ► `t.test()`
- ► `wilcox.test()`
- ► `kruskal.test()`
- ► `ks.test()`
- ► `friedman.test()`
- ► `shapiro.test()`

Categorical Outcomes
- ► `prop.test()`
- ► `binom.test()`
- ► `chisq.test()`
- ► `fisher.test()`
- ► `mcnemar.test()`
- ► `mantelhaen.test()`

Pairwise tests
- ► `pairwise.prop.test()`
- ► `pairwise.t.test()`
- ► `pairwise.wilcox.test()`

Variance and Correlation
- ► `cor.test()`
- ► `bartlett.test()`
- ► `var.test()`

Multiple Testing Adjustment
- ► `p.adjust()`

Linear Regression

A standard linear regression model has the form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \varepsilon \qquad \text{with} \quad \varepsilon \sim N(0, \sigma^2)$$

where

- y is the **outcome** variable ("dependent variable")

- $x_1, \ldots, x_p$ are the **covariates** ("independent variables")

- $\beta_0, \ldots, \beta_p$ are the **regression coefficients**
  - β_0 is the intercept
  - $\beta_1, \ldots, \beta_p$ estimate the effects of the covariates

- ε is a vector of **residuals**, which we assume to be (approximately) normally distributed.

Linear Regression

To fit a **linear regression** in R we use the function `lm()`.

The most important arguments are

▶ `formula`:
  a formula object

▶ `data`:
  a data.frame (optional, but usually needed)

▶ `subset`:
  a vector specifying which observations should be used (optional)
  (works like the subset argument of the function subset())

Model Formula

A `formula` object has the form

```
outcome ~ linear predictor
```

for example

```
y ~ x1 + x2 + x3
```

Model Formula

A `formula` object has the form

```
outcome ~ linear predictor
```

for example

```
y ~ x1 + x2 + x3
```

► Variables are separated by "**+**" signs.

► An intercept is automatically included.

► One-sided formulas (omitting the outcome) are possible (used for random effects specification).

Model Formula: Interactions

Interaction terms are written using ":" or "*".

"*" includes the main effects and interaction terms, i.e.,

```
y ~ x1 * x2
```

is equivalent to

```
y ~ x1 + x2 + x1:x2
```

Model Formula: Interactions

Interaction terms are written using ":" or "*".

"*" includes the main effects and interaction terms, i.e.,

```
y ~ x1 * x2
```

is equivalent to

```
y ~ x1 + x2 + x1:x2
```

Interactions between multiple variables can be written using "()", i.e.,

```
y ~ x1 * (x2 + x3)
```

is equivalent to

```
y ~ x1 * x2 + x1 * x3
```

Model Formula: Interactions

To specify a **higher level interaction** "^" is used.

For example:

```
y ~ (x1 + x2 + x3)^3
```

will create all interactions up to 3-way and is equivalent to

```
y ~ x1 * x2 * x3
```

and equivalent to

```
y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3 + x1:x2:x3
```

Model Formula: Interactions

To specify a **higher level interaction** "ˆ" is used.

For example:

```
y ~ (x1 + x2 + x3)^3
```

will create all interactions up to 3-way and is equivalent to

```
y ~ x1 * x2 * x3
```

and equivalent to

```
y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3 + x1:x2:x3
```

and

```
y ~ (x1 + x2 + x3)^2
```

will create all two-way interactions and is equivalent to

```
y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3
```

Model Formula: Removing terms

The "–" sign can be used to remove terms from a model formula.

For example

```
y ~ x1 * x2 * x3 - x2 - x1:x3
```

is equivalent to

```
y ~ x1 + x3 + x1:x2 + x2:x3 + x1:x2:x3
```

Model Formula: Removing terms

The "−" sign can be used to remove terms from a model formula.

For example

```
y ~ x1 * x2 * x3 − x2 − x1:x3
```

is equivalent to

```
y ~ x1 + x3 + x1:x2 + x2:x3 + x1:x2:x3
```

The **intercept** can be removed from a formula by using "−1" or "+0", i.e.

```
y ~ x1 + x2 − 1
y ~ x1 + x2 + 0
```

Generalized Linear Regression (GLM)

A **generalized linear regression** model has the form

$$g(\mathbb{E}(y)) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

where $g(\ )$ is a link function and y is from the exponential family.

Generalized Linear Regression (GLM)

A **generalized linear regression** model has the form

$$g(\mathbb{E}(y)) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

where $g(\ )$ is a link function and y is from the exponential family.

For example **logistic regression** for binary y:

$$\log\left(\frac{P(y = 1)}{1 - P(y = 1)}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$$

$\log\left(\frac{p}{1-p}\right)$ is the **logit** link.

Generalized Linear Regression (GLM)

To fit a **GLM** in R we use the function `glm()`.

The most important arguments are

- ▶ `formula`:
  a `formula` object

- ▶ `family`:
  a `family` object or name of the family function, describing the error distribution and link function

- ▶ `data`:
  a `data.frame` (optional, but usually needed)

- ▶ `subset`:
  a `vector` specifying which observations should be used (optional)

Families and Link Functions

Common families & available links in R: (see also `?family`)

family	link
binomial	`logit`, `probit`, `cauchit`, `log`, `cloglog`
gaussian	`identity`, `log`, `inverse`
Gamma	`inverse`, `identity`, `log`
poisson	`log`, `identity`, `sqrt`

Families and Link Functions

Common families & available links in R: (see also `?family`)

family	link
binomial	`logit`, `probit`, `cauchit`, `log`, `cloglog`
gaussian	`identity`, `log`, `inverse`
Gamma	`inverse`, `identity`, `log`
poisson	`log`, `identity`, `sqrt`

The `family` argument in `glm()` can be specified in the following ways:

- ▶ `binomial(link = "logit")`
- ▶ `binomial()`
- ▶ `binomial`
- ▶ `"binomial"`

Families and Link Functions

Common families & available links in R: (see also `?family`)

family	link
binomial	logit, probit, cauchit, log, cloglog
gaussian	identity, log, inverse
Gamma	inverse, identity, log
poisson	log, identity, sqrt

The family argument in `glm()` can be specified in the following ways:

- ► `binomial(link = "logit")`
- ► `binomial()`
- ► `binomial`
- ► `"binomial"`

Note:
When the link is not explicitly specified (i.e. options 2-4), the default link is used.

Regression

Demo
- ► Regression Basics
  `R` `html`

Practical
- ► Linear Regression `html`

Model Evaluation

Linear model:

Evaluate the **assumptions** of a linear regression model visually, for example:

- ► Histogram of residuals
- ► Normal QQ-plot of residuals
- ► Scatterplot residuals vs fitted values
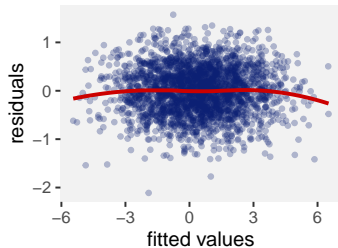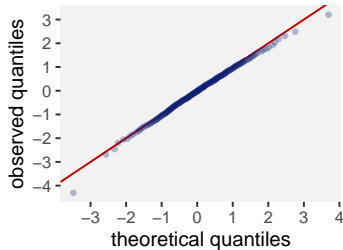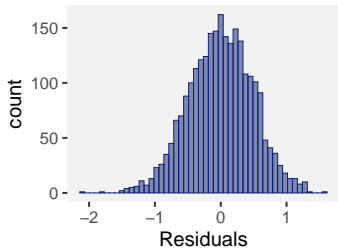
Model Comparison

Nested models:

- ▶ model is a **special case** of the other, i.e.,
- ▶ model B is a special case of model A when B can be obtained by setting some regression coefficients in A to zero

Comparison using a **likelihood ratio (LR) test**, for example:

```
anova(modelA, modelB)
anova(modelA, modelB, test = "LRT") # for a glm
```

Model Comparison

Nested models:

- ► model is a **special case** of the other, i.e.,
- ► model B is a special case of model A when B can be obtained by setting some regression coefficients in A to zero

Comparison using a **likelihood ratio (LR) test**, for example:

```
anova(modelA, modelB)
anova(modelA, modelB, test = "LRT") # for a glm
```

Non-nested models:

Comparison using information criteria, e.g.

```
AIC(modelA, modelB)
BIC(modelA, modelB)
```

The model with the **smaller** AIC (or BIC) has the **better** fit.

Model Evaluation

Demo

► Model Evaluation `R` `html`

Non-linear Effects

Default assumption: **linear effect**, i.e., $x \to y$ ➡ $x + 1 \to y + \beta, \quad \forall x$
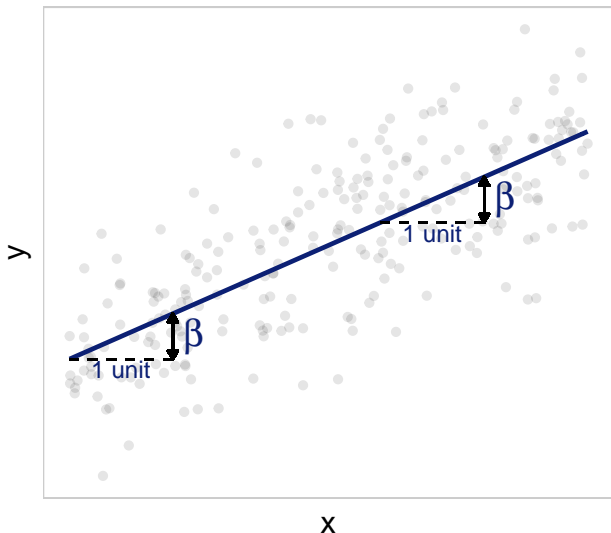
Non-linear Effects

Default assumption: **linear effect**, i.e., $x \to y \implies x + 1 \to y + \beta, \quad \forall x$

This may not always be the case:
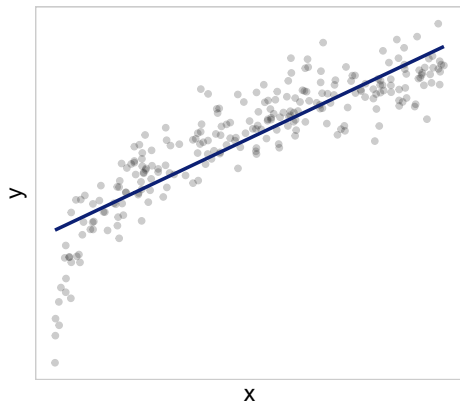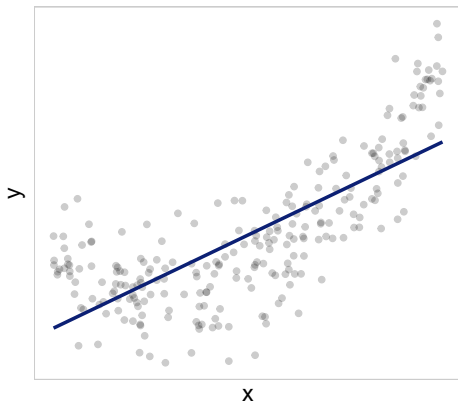
Non-linear Effects

Here, we would like to allow the **effect** of a one-unit increase **of x** to **change with the value of x**:
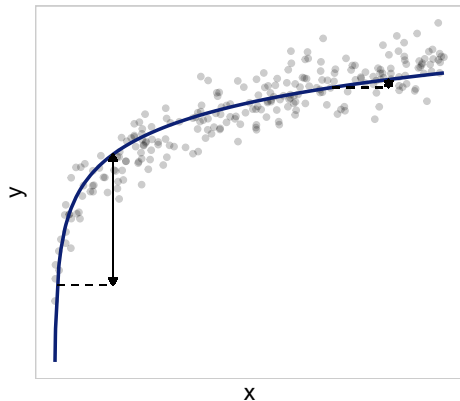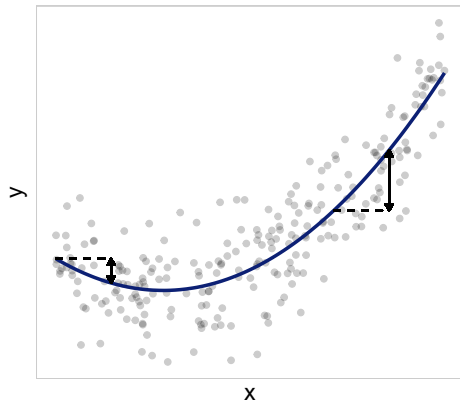
Non-linear Effects

Sometimes, we can use

- ▶ a **transformation of x**, or
- ▶ x as well as **a polynomial** of x (or a transformation).

For example:

```
y ~ x + I(x^2)
```

or

```
y ~ log(x)
```
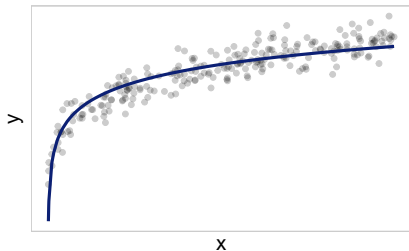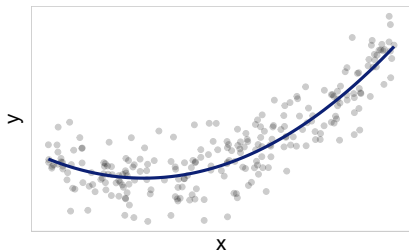
Non-linear Effects: `I()`

The function `I()` is needed to distinguish between operators that need to be interpreted as

- ▶ arithmetic operators and
- ▶ formula operators

Non-linear Effects: `I()`

The function `I()` is needed to distinguish between operators that need to be interpreted as

- ► arithmetic operators and
- ► formula operators

Example:

```
y ~ I(a + b)
```

would be the same as

```
z <- a + b
y ~ z
```

but not the same as

```
y ~ a + b
```

Complex Non-linear Effects

Non-linear effects may be **more complex** than can be modelled with a simple transformation or polynomial.
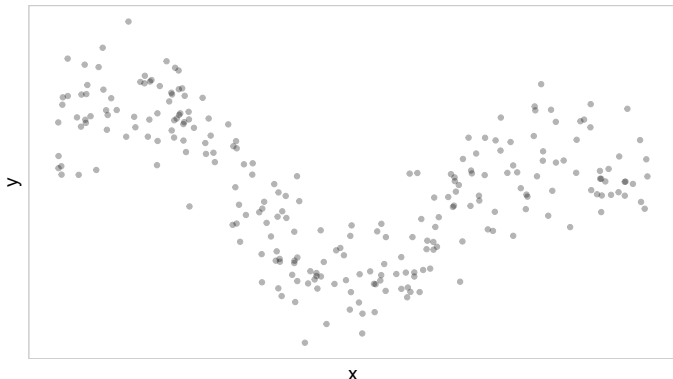
Also: the shape may depend on other covariates in the model
➡ we do not always know the shape in advance
➡ **Regression Splines** / **B-Splines**

B-Splines

A **B-Spline** is a linear combination of a set of **basis functions**.

These basis functions are defined so that they are
  ▶ a **polynomial functions** inside a given interval, and
  ▶ zero outside that interval.

The intervals are defined by a set of **knots**.

The polynomial function have a certain **degree** (i.e., constant, linear, quadratic, …)

B-Splines

A **B-Spline** is a linear combination of a set of **basis functions**.

These basis functions are defined so that they are
- ► a **polynomial functions** inside a given interval, and
- ► zero outside that interval.

The intervals are defined by a set of **knots**.

The polynomial function have a certain **degree** (i.e., constant, linear, quadratic, …)

B-Splines in R

The R package **splines** provides the functions

- ► `bs()`: B-splines
- ► `ns()`: natural cubic (B-)splines

B-Splines

Instead of $\quad y \sim \beta_0 + \beta_1 x + \ldots \quad$ we assume $\quad y \sim \beta_0 + \sum_{\ell=1}^{d} \beta_\ell B_\ell(x) + \ldots$
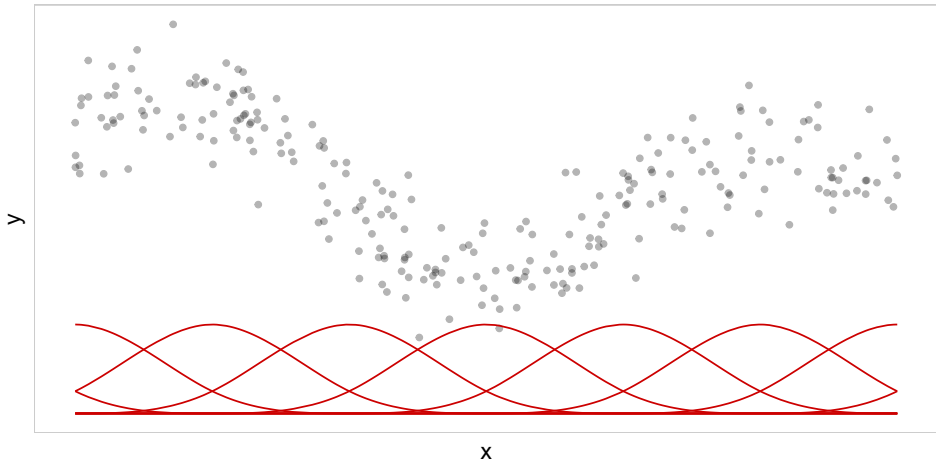
B-Splines

Instead of $\quad y \sim \beta_0 + \beta_1 x + \ldots \quad$ we assume $\quad y \sim \beta_0 + \sum_{\ell=1}^{d} \beta_\ell B_\ell(x) + \ldots$
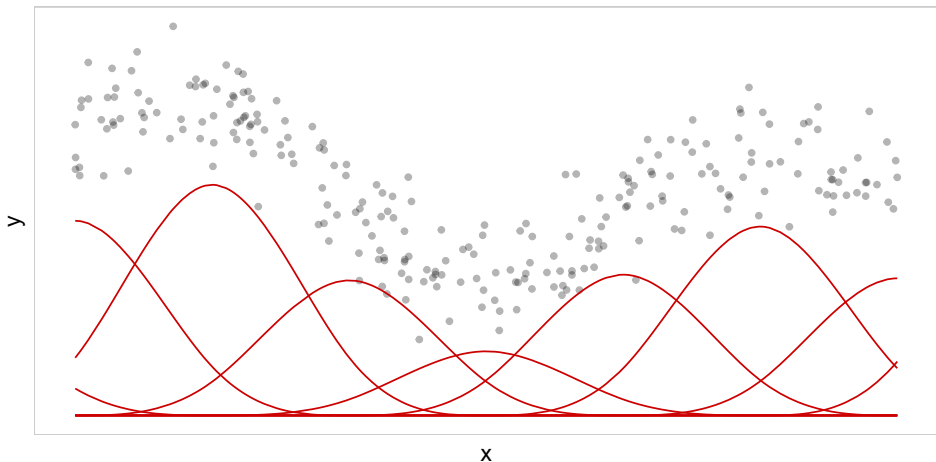
B-Splines

Instead of $y \sim \beta_0 + \beta_1 x + \ldots$ we assume $y \sim \beta_0 + \sum_{\ell=1}^{d} \beta_\ell B_\ell(x) + \ldots$
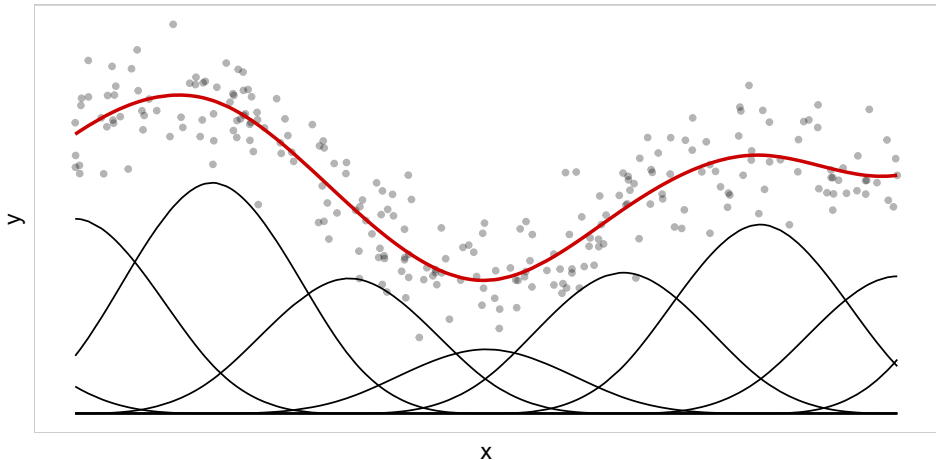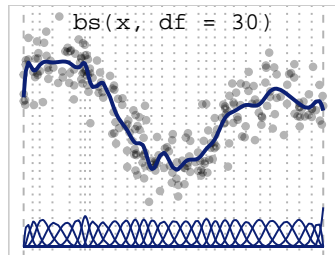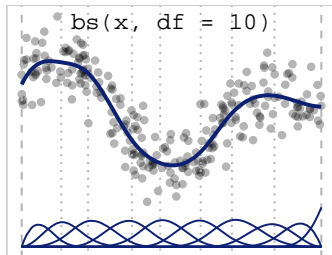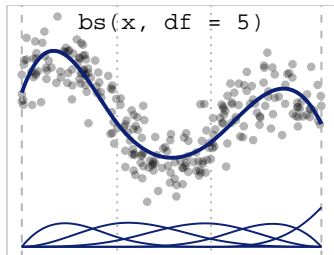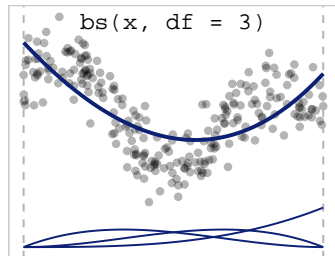
B-Splines: degree
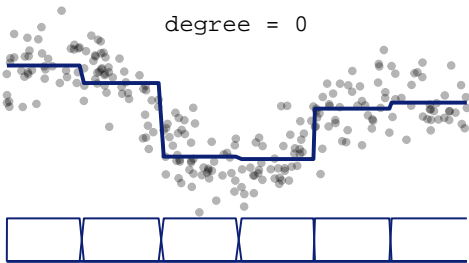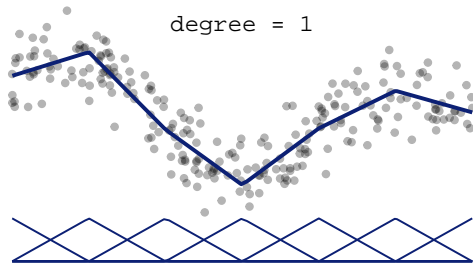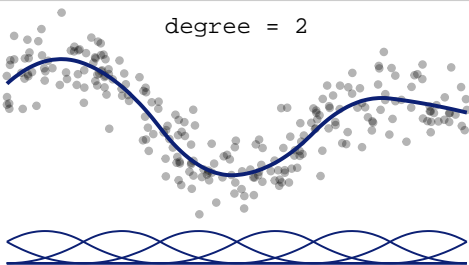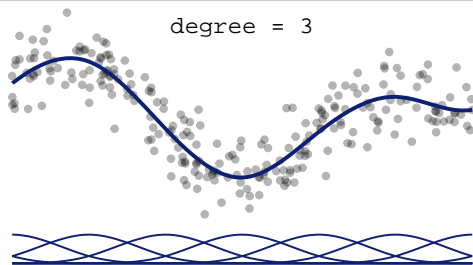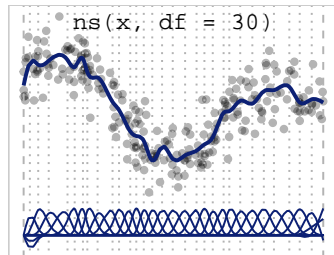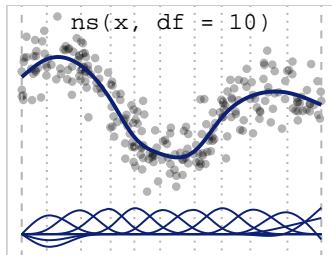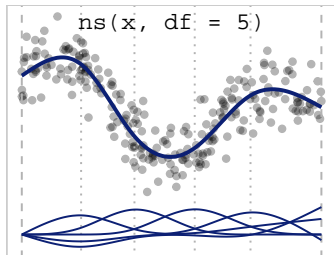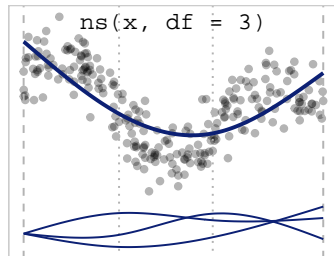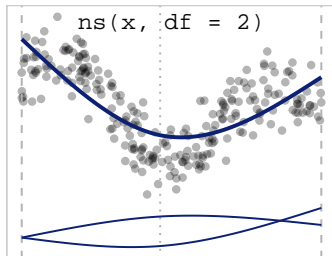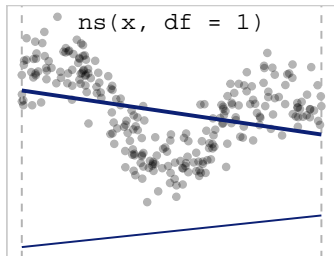
B-Splines: `df`

B-Splines: df

B-Splines in R: `bs()` & `ns()`

Important arguments of `ns()` and `bs()` are:

degree
- ▶ degree of the polynomial in each of the basis functions
- ▶ in `bs()`: default is 3
- ▶ in `ns()`: always 3 ("cubic")

B-Splines in R: `bs()` & `ns()`

Important arguments of `ns()` and `bs()` are:

degree
- ▶ degree of the polynomial in each of the basis functions
- ▶ in `bs()`: default is 3
- ▶ in `ns()`: always 3 ("cubic")

df
- ▶ degrees of freedom, i.e., "number of regression coefficients" used
- ▶ for `bs()`: has to be $\geq$ `degree`

B-Splines in R: `bs()` & `ns()`

Important arguments of `ns()` and `bs()` are:

degree

- ► degree of the polynomial in each of the basis functions
- ► in `bs()`: default is 3
- ► in `ns()`: always 3 ("cubic")

df

- ► degrees of freedom, i.e., "number of regression coefficients" used
- ► for `bs()`: has to be $\geq$ `degree`

knots

- ► position of (inner) knots
- ► if unspecified:
  - ► `df-degree` knots are used
  - ► positioned at equally spaced quantiles

B-Splines in R: `bs()` & `ns()`

Important arguments of `ns()` and `bs()` are:

degree
- ▶ degree of the polynomial in each of the basis functions
- ▶ in `bs()`: default is 3
- ▶ in `ns()`: always 3 ("cubic")

df
- ▶ degrees of freedom, i.e., "number of regression coefficients" used
- ▶ for `bs()`: has to be $\geq$ `degree`

knots
- ▶ position of (inner) knots
- ▶ if unspecified:
  - ▶ `df-degree` knots are used
  - ▶ positioned at equally spaced quantiles

Boundary.knots
- ▶ by default: `range(x)`
- ▶ outside the `Boundary.knots` the fit is extrapolated

Non-linear Effects

Demo

► Splines `R` `html`

Practicals

► Logistic Regression & More `html`

► Logistic Regression II `html`

► Custom Model Summary Function `html`

Regression

Regression Models
- ► `lm()`
- ► `glm()`

Regression Results
- ► `summary()`
- ► `coef()`, `confint()`
- ► `fitted()`, `residuals()`, `rstandard()`
- ► `AIC()`, `BIC()`
- ► `anova()`

Plots
- ► `plot()`
- ► `qqnorm()`, `qqline()`, `qqplot()`

Formulas
- ► Formula operators: +, -, *, :, ^
- ► `ns()`, `bs()`, `I()`
- ► `all.vars()`
- ► `update()`
- ► `as.formula()`