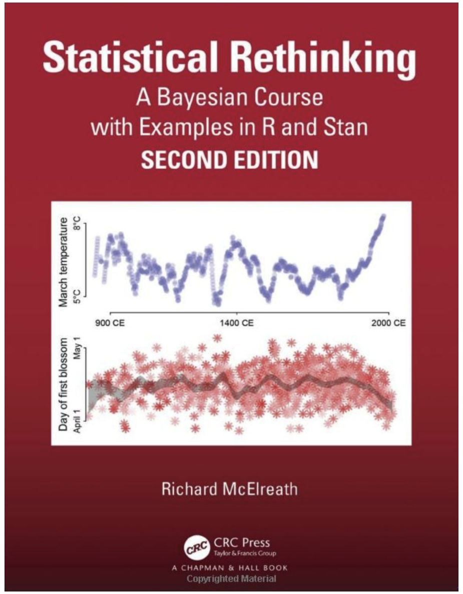
Introduction to Data Science

Advanced Modeling Practice

Prof. Dr. Ralf Lämmel & **M.Sc. Johannes Härtel**
(johanneshaertel@uni-koblenz.de)



[McElreath20]

Documentation

https://mc-stan.org/users/documentation/

The major source for this lecture.

Advanced Modeling Practice

- Logistic regression
- Dummy/index Variables
- Multilevel models (hierarchical priors)
- Missing data in a time series

Logistic Regression Models

Logistic Regression Model

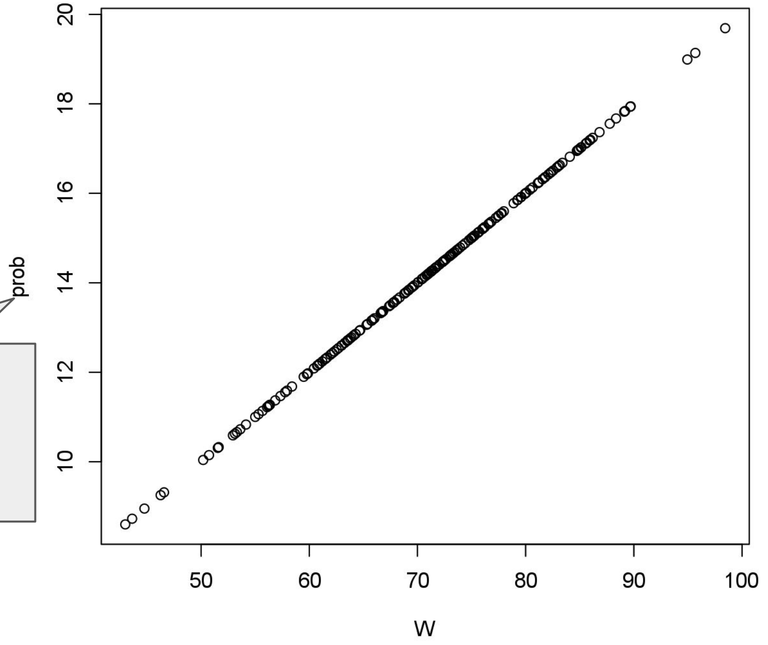
- The output variable follows a **binomial distribution**.
- Analogue to a linear model, we related any predictor variables to a binomial output.
- We can answer questions, for instance, on the probability of getting COVID-19 (binomial with 1 trial) under certain conditions described as variables.
- Interpretation of parameters slightly differs, but is mostly compared to an interpretation of a linear model.

Simulating Logistic Data: Does overweight increase the risk of getting COVID-19?

```
N <- 200

W <- rnorm(N, mean = 70, sd = 10)

prob <- 0.2 * W
```

Let’s assume that weight W increases the risk given as prob.

This is not a valid probability, as it is not between 0 and 1.

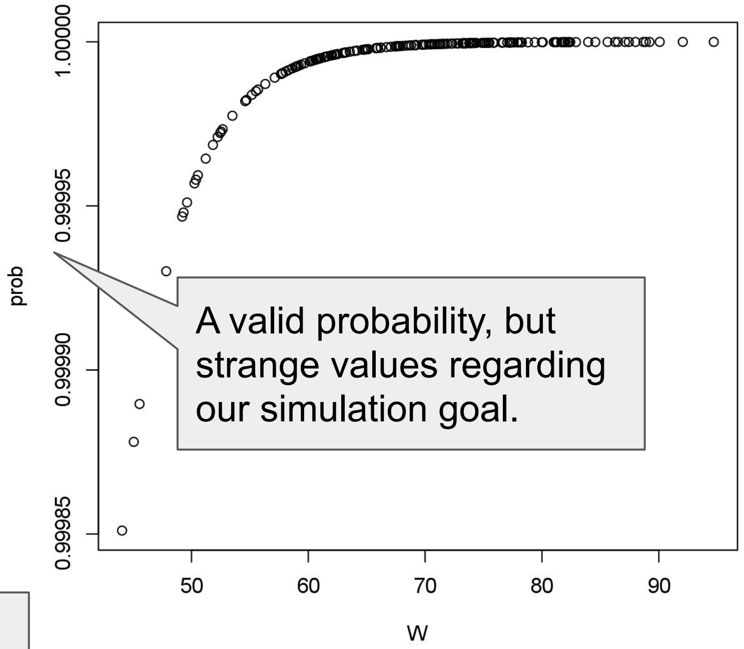


Simulating Logistic Data: We apply the inverse logistic function.

$$\frac{1}{1+e^{-x}}$$

```
N <- 200

W <- rnorm(N, mean = 70, sd = 10)

prob <- 1 / (1 + exp(-(0.2 * W)))
```

Rethinking package also has inv_logit implemented for doing this.



A valid probability, but strange values regarding our simulation goal.

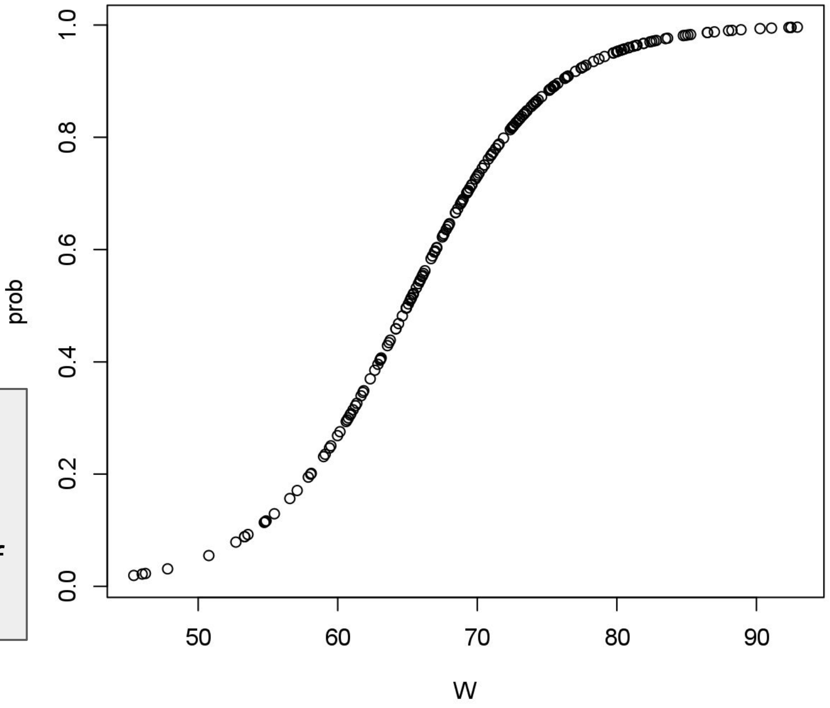
Simulating Logistic Data: Adjusting the linear component

```
N <- 200

W <- rnorm(N, mean = 70, sd = 10)

prob <- 1 / (1 + exp(-(-13 + 0.2 * W)))
```

$$\underbrace{1 / (1 + exp(-}_{\text{The inverted logistic function is one of the available "link functions".}}\underbrace{(-13 + 0.2 * W)}_{\text{This component is comparable to a linear model.}}))$$

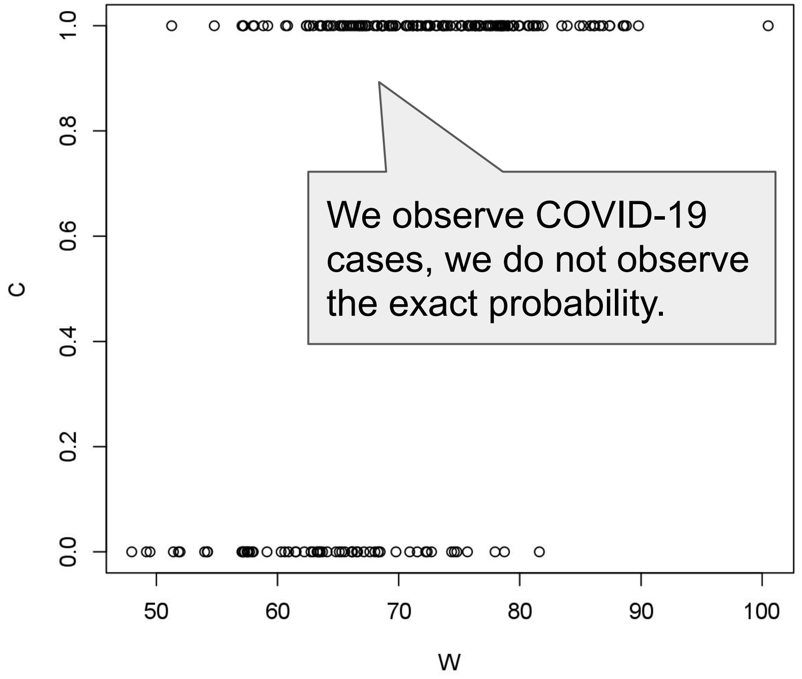
The inverted logistic function is one of the available “**link functions**”.

This component is comparable to a linear model. It is often called the **linear component** of the logistic regression.



Simulating Logistic Data: The binomial distribution creates uncertainty.

```
N <- 200

W <- rnorm(N, mean = 70, sd = 10)

prob <- 1 / (1 + exp(-(-13 + 0.2 * W)))

# Covid
C <- rbinom(N, size = 1, prob = prob)
```



Implementing the Model to find the parameters alpha and beta

```
data{
 int<lower=0> N;
 vector[N] W;
 int<lower=0,upper=1> C[N];
}
parameters{
 real alpha;
 real beta;
}
model{
 C ~ bernoulli_logit(alpha + beta * W);
}
```

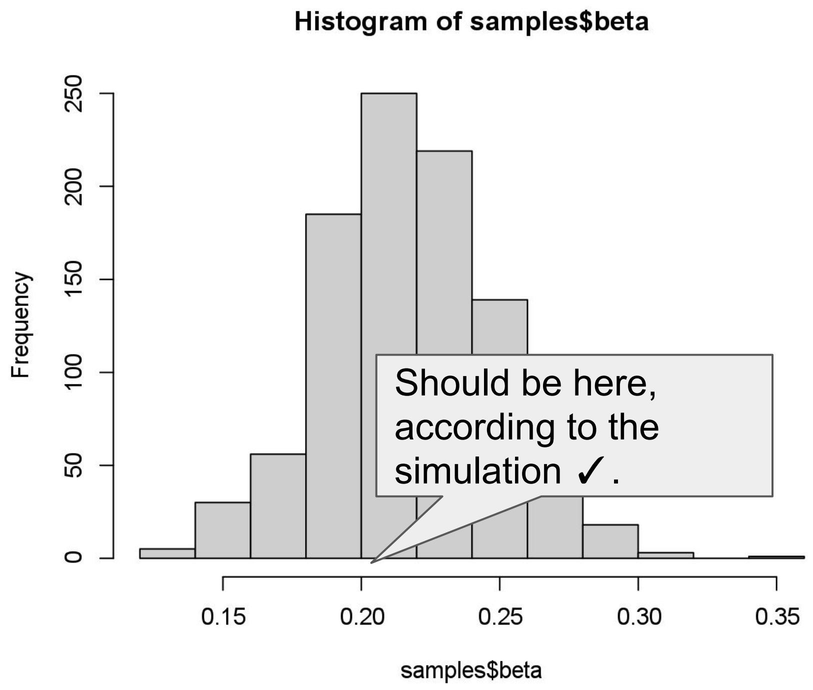
We do not need a parameter sigma, but we have **alpha and beta** (comparable to a linear model).

We use a different sampling statement for the output C, saying that if follows a **Bernoulli distribution** (binomial with 1 trial).

For details, have a look at the Stan user manual.

Examining the sample from the posterior for parameter beta:

We can infer the correct effect of weight (W) on Covid-19 (C).



Dummy/index Variables

Dummy/index Variables

- In the last lectures, we have been using parameters in a very moderate sense.
- However, we may also introduce **vectors of parameters**. This may be called dummy/index variables.
- There are different decoding schemes, we stick to the most basic.
- A drawback is that inserting many parameters increases the risk of **overfitting**.

An influenced assignment mechanism (Recap):

We **don’t know** the variable P, influencing the **assignment mechanisms**.

Students might be tempted to take Aspirin after the party, and **we cannot track this if we don’t know P**.

$$\begin{tikzcd} & \text{(Party the day before)} & \\ & \boxed{P} \arrow[dl] \arrow[dr] & \\ A \arrow[rr] & & H \\ \text{(take Aspirin)} & & \text{(Headache the day after)} \end{tikzcd}$$

Simulating the scenario (Recap)

```
# Party or not.
P <- rbinom(N, 1, 0.5)

# Assignment mechanisms now influenced by the party.
A <- rbinom(N, 1, prob = ifelse(P, 0.9, 0.1))

# We simulate headache caused by party and taking no aspirin.
mu <- -0.3 + 0.3 * P - 0.2 * A
sigma <- 0.07

H <- rnorm(N, mu, sigma)
```

The influence of P on A.

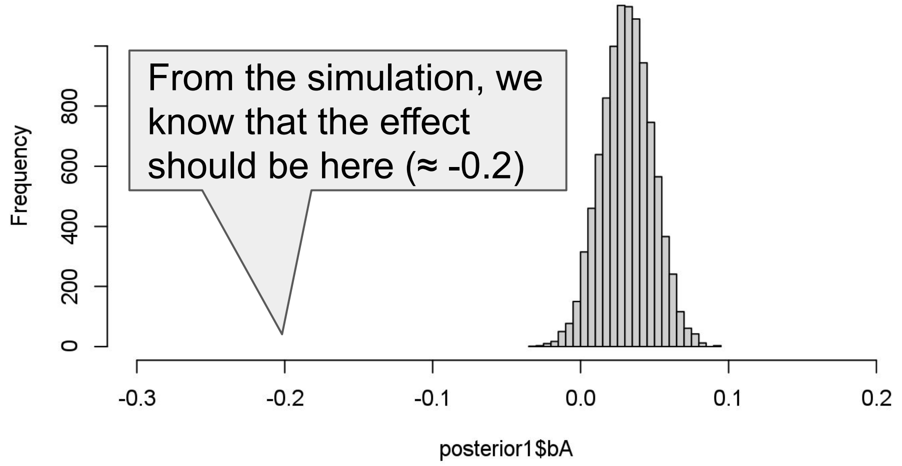
The corresponding linear model (Recap):

We don’t have P, so we may just examine the effect of A.

H_i	$\sim$	$\text{Normal}(\mu_i, \sigma)$	[likelihood]
μ_i	$=$	$\alpha + \beta_A A_i$	[linear model]
α	$\sim$	$\text{Normal}(0,1)$	[α prior]
β_A	$\sim$	$\text{Normal}(0, 1)$	[β prior]
σ	$\sim$	$\text{Uniform}(0, 3)$	[σ prior]

Wrong results when missing variable P (Recap):

We are getting a **wrong result**, with the posterior of β_A right to 0.0, suggesting that **aspirin causes headache**.



We know how to correct this if we know P.

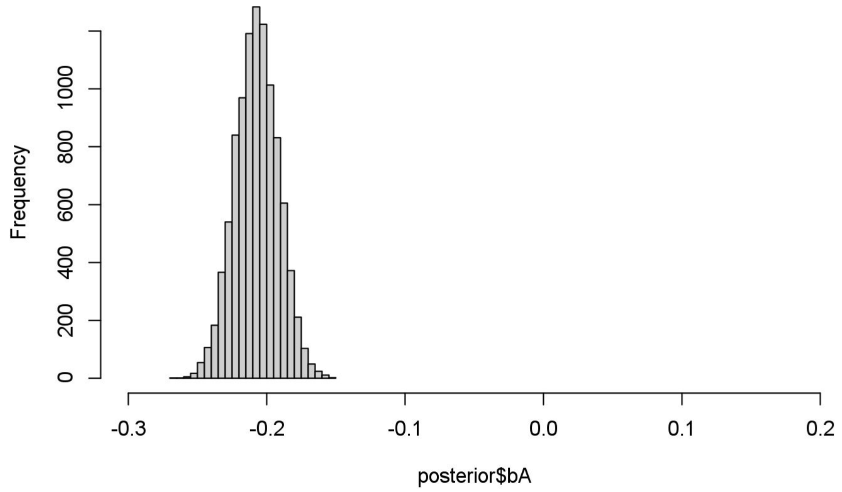
A model including P (Recap):

We use a **multiple regression model**, adding P as a predictor to **adjust for the assignment mechanism**.

$H_i \sim \text{Normal}(\mu_i, \sigma)$	[likelihood]
$\mu_i = \alpha + \beta_A A_i + \beta_P P_i$	[linear model]
$\alpha \sim \text{Normal}(0,1)$	[α prior]
$\beta_A \sim \text{Normal}(0, 1)$	[β prior]
$\beta_P \sim \text{Normal}(0, 1)$	[β prior]
$\sigma \sim \text{Uniform}(0, 3)$	[σ prior]

Correct results when including P (Recap):

We can adjust for the assignment mechanisms and again get the **correct β_A**(≈ -0.2).



Can we also correct for P without knowing it?*

(*using some other data)

We try to infer P as a parameter ps_i:

We will see that this approach will **not work due to over-parametrization**.

We remove α (related to decoding of index/dummy).

$H_i \sim$ [illegible]$ormal(\mu_i, \sigma)$ [likelihood]

$\mu_i = \beta_A A_i + ps_i$ [linear model]

$ps_i \sim \text{Normal}(0, 1)$

...

Our new index/dumy parameters ps_i.

We now have a parameter ps_i for each observation. (caution, this does not work as we face **too many parameters and too little data**)

Implementing the new but wrong model in STAN

OLD

```
data{
 int<lower=0> N;
 vector[N] H;
 vector[N] A;
}
parameters{
 real alpha;
 real beta;
 real<lower=0> sigma;
}
model{
 vector[N] mu;
 mu = alpha + beta * A;
 H ~ normal(mu,sigma);
}
```

Adding the index/dummy parameters.

NEW: Inferring P as parameter called ps:

```
data{
 int<lower=0> N;
 vector[N] H;
 vector[N] A;
}
parameters{
 real beta;
 real<lower=0> sigma;
 vector[N] ps;
}
model{
 vector[N] mu;
 mu = beta * A + ps;
 H ~ normal(mu, sigma);
}
```

A **vector** of parameters.

Why does this new model **not work:** over-parametrization

```
model{
 vector[N]
 mu = beta * A + ps;
 H ~ normal(mu, sigma);
}
```

We can rewrite the model, extracting the error.

```
model{
 vector[N] mu;
 vector[N] error;
 error ~ normal(0, sigma);
 mu = beta * A + ps;
 H = mu + error;
}
```

More rewriting.

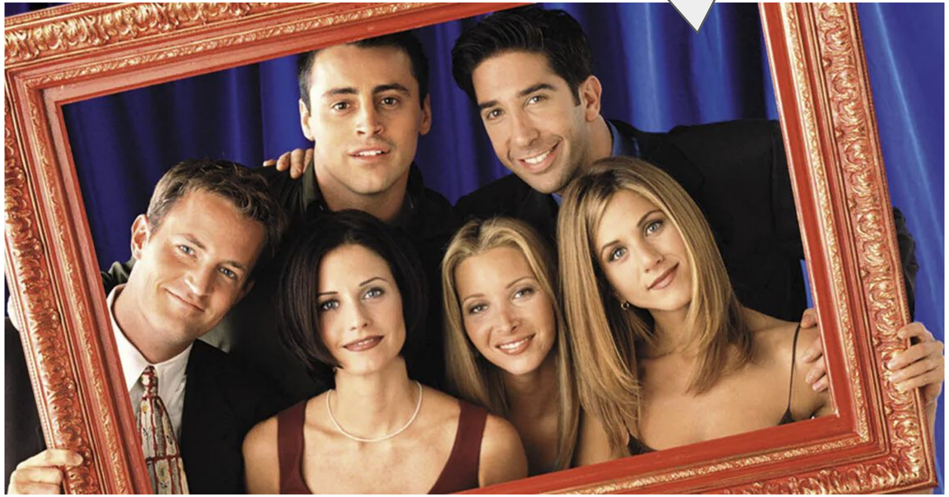
```
parameters{
 ...
 vector[N] ps;
}
model{
 vector[N] error;
 error ~ normal(0, sigma);
 H = beta * A + ps + error;
}
```

Parameters can be confused.

Parameters ps will be confused with the error.

Remedy: Employing **structure**

Fiends.



Let’s assume that **fiends are always going to parties together** and that we know the groups of friends in our set of observations.

Resolving over-parametrization using structure.

- We still don’t know P.
- However, we know that variable **P is the same** for people (observations) belonging to the **same group of friends**.
- Hence, we have fewer parameters and more “relationship” in our model.

Simulating the structured data

```r
N <- 10 # 10 groups.
M <- 20 # Each group contains 20 friends.

# Group id.
F <- rep(1:N, each = M)

# Party or not (for N groups, they only go together).
P <- rbinom(N, 1, 0.5)

# Aspirin now influenced by the party P[F] (indexed access).
A <- rbinom(N * M, 1, ifelse(P[F], 0.9, 0.1))

# We simulate headache caused by party and taking no aspirin.
mu <- -0.3 + 0.3 * P[F] - 0.2 * A
sigma <- 0.07

H <- rnorm(N * M, mu, sigma)
```

Infer P as parameter ps_j:

We now use an index notation for **person i**, in **fiend group j**, to denote observations.

$$
\begin{aligned}
H_{i,j} &\sim \text{Normal}(\mu_{i,j}, \sigma) && \text{[likelihood]} \\
\mu_{i,j} &= \beta_A A_{i,j} + ps_j && \text{[linear model]} \\
\\
ps_j &\sim \text{Normal}(0, 1) \\
&\ldots
\end{aligned}
$$

Our new index/dummy parameter p_j for each friend group.

Implementing this model in Stan

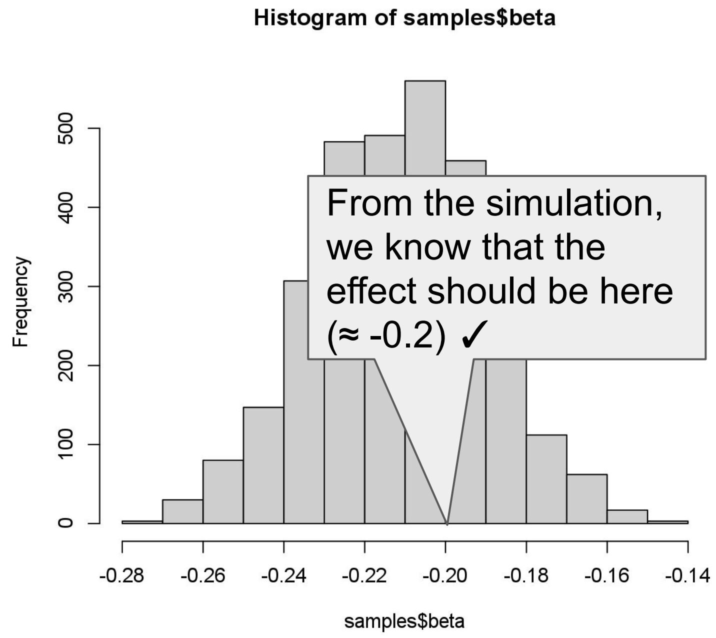
```
data{
 int<lower=0> N;
 int<lower=0> M;
 int F[N*M];
 vector[N*M] H;
 vector[N*M] A;
}
parameters{
 real beta;
 real<lower=0> sigma;
 vector[N] ps;
}
model{
 vector[N*M] mu;
 ps ~ normal(0,1);
 mu = beta * A + ps[F];
 H ~ normal(mu, sigma);
}
```

For each observation, we know to which group of friends it belongs. We use it to access the correct parameter ps.

Parameter vector.

Index access.

Finally, we manage to recover the correct effect of aspirin without knowing P.



Multilevel Models

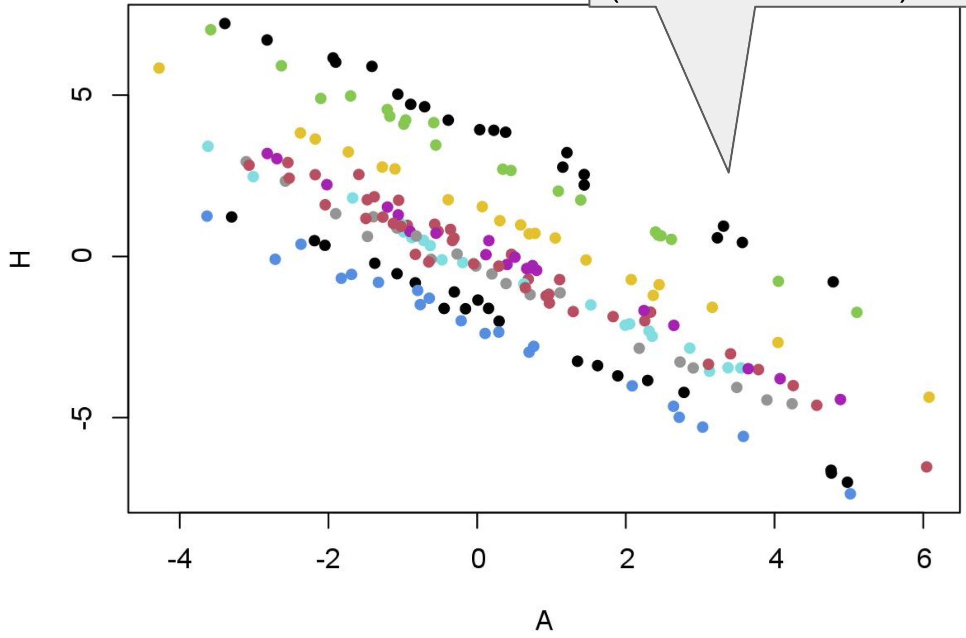
Multilevel models

- Multilevel models found on the idea of dummy/index variables, introducing vectors of parameters.
- However, they are different, since they introduce hierarchical priors (priors on priors).
- A prior can be used for regularization (decrease the risk of overfitting). A prior on another prior can be used for **self adjusted regularization**. This is also called shrinkage in the context of multilevel models.

We need to simulated slightly different data, relying solely on normal distributions (the classical multilevel model).

```
N <- 10 # 10 groups.
M <- 20 # Each group contains 20 friends (later also shown for 5).

# Group id.
F <- rep(1:N, each = M)

P <- rnorm(N,0,0.4)
A <- rnorm(N * M, P, 1)

mu <- P[F] - A

H <- rnorm(N * M, mu, 0.2)
```

Depicting the data



We see clusters of observation belonging together (shown as colors).

Implementing this model, adding priors on priors.

```
data{
 int<lower=0> N;
 int<lower=0> M;
 int F[N*M];
 vector[N*M] H;
 vector[N*M] A;
}
parameters{
 real alpha;
 real beta;
 real<lower=0> sigma;
 real<lower=0> sigma2;
 vector[N] ps;
}
model{
 vector[N*M] mu;
 ps ~ normal(0, sigma2);
 sigma2 ~ exponential(5);
 alpha ~ normal(0,1);
 mu = alpha + beta * A + ps[F];
 H ~ normal(mu, sigma);
}
```

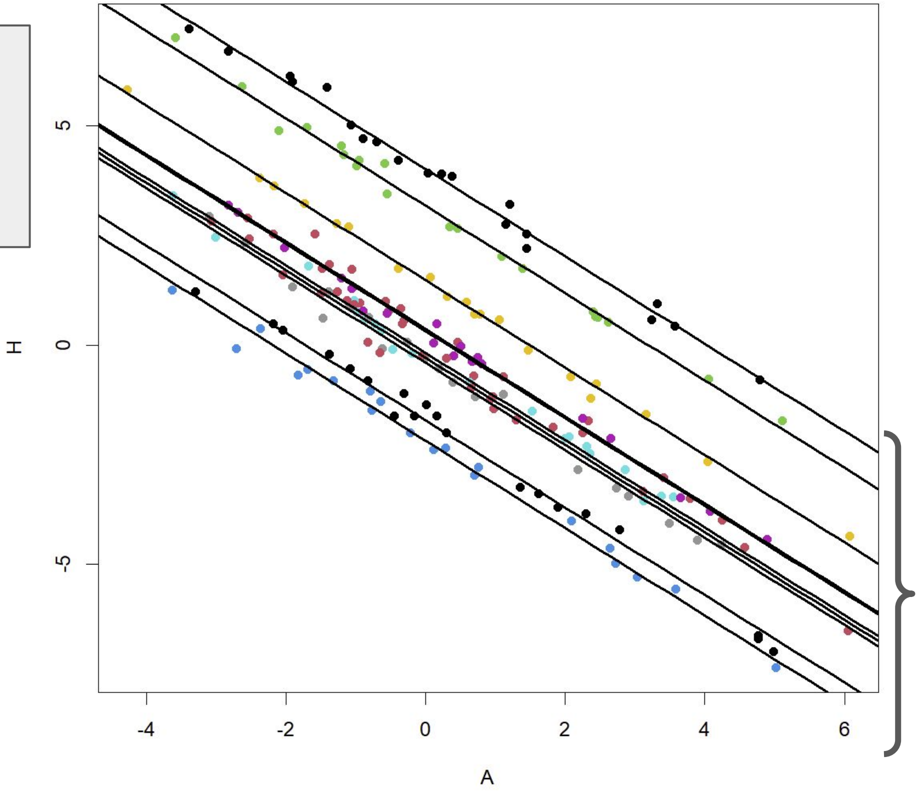
We have a prior, and a prior on a prior.

We use and exponential prior (but there are many other options).

We are getting back alpha.

Depicting the regression lines of such model

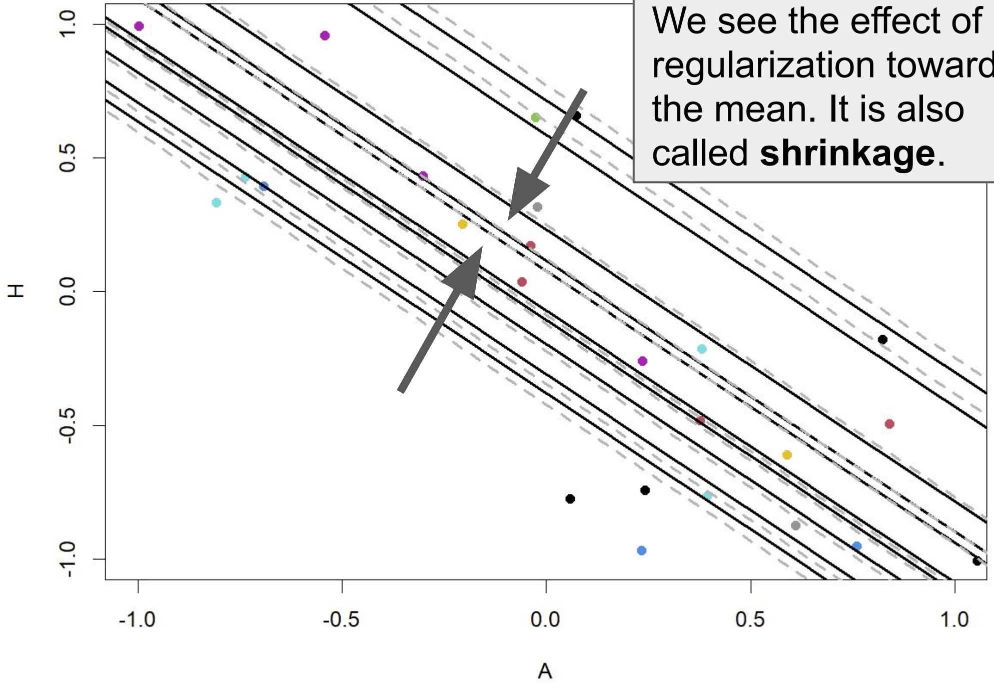
We have different regression lines for groups of friends, due to parameters ps.



This spread is described by our high-level parameter sigma2.

Comparing the multilevel model to a plain dummy/index variable strategy (for 5 friends in each group)

- **Gray lines** are dummy/index variable approach.
- **Black lines** are multilevel (self adjusting prior) approach.

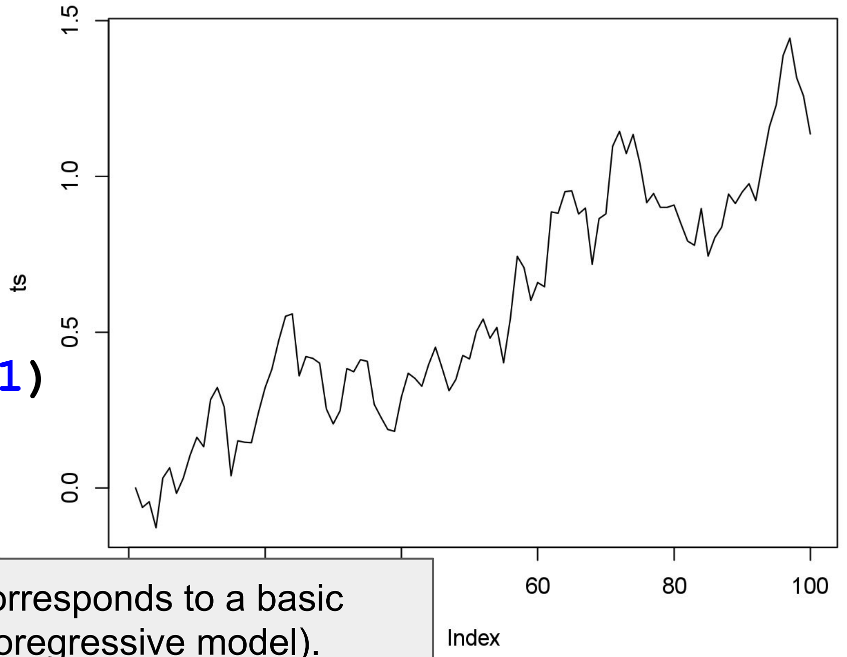


We see the effect of regularization toward the mean. It is also called **shrinkage**.

Missing data in a time series

Simulating time series data

```r
ts <- rep(NaN, 100)
ts[1] <- 0

for (i in 2:100) {
 ts[i] <- rnorm(1, ts[i - 1], 0.1)
}

plot(ts, type = "l")
```

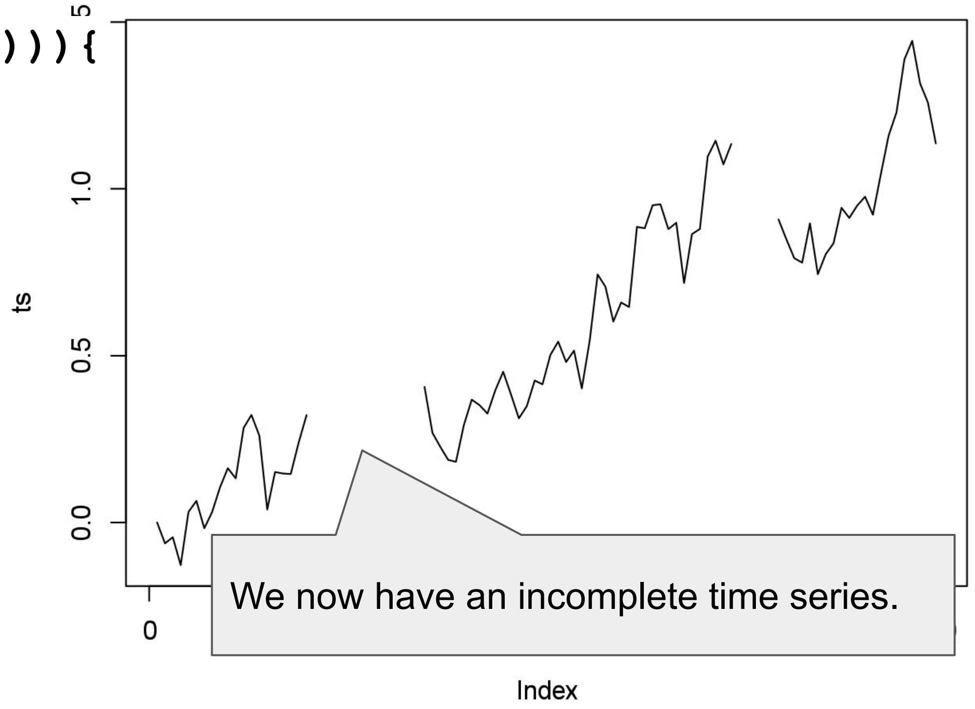


This simulation corresponds to a basic AR(1) model (autoregressive model).

Simulating missing values

```
for(del in floor(runif(4,0,97))){
 for(i in del:(del+4)){
   ts[i] <- NaN
 }
}
```

Deleting random parts of the sequence.



We now have an incomplete time series.

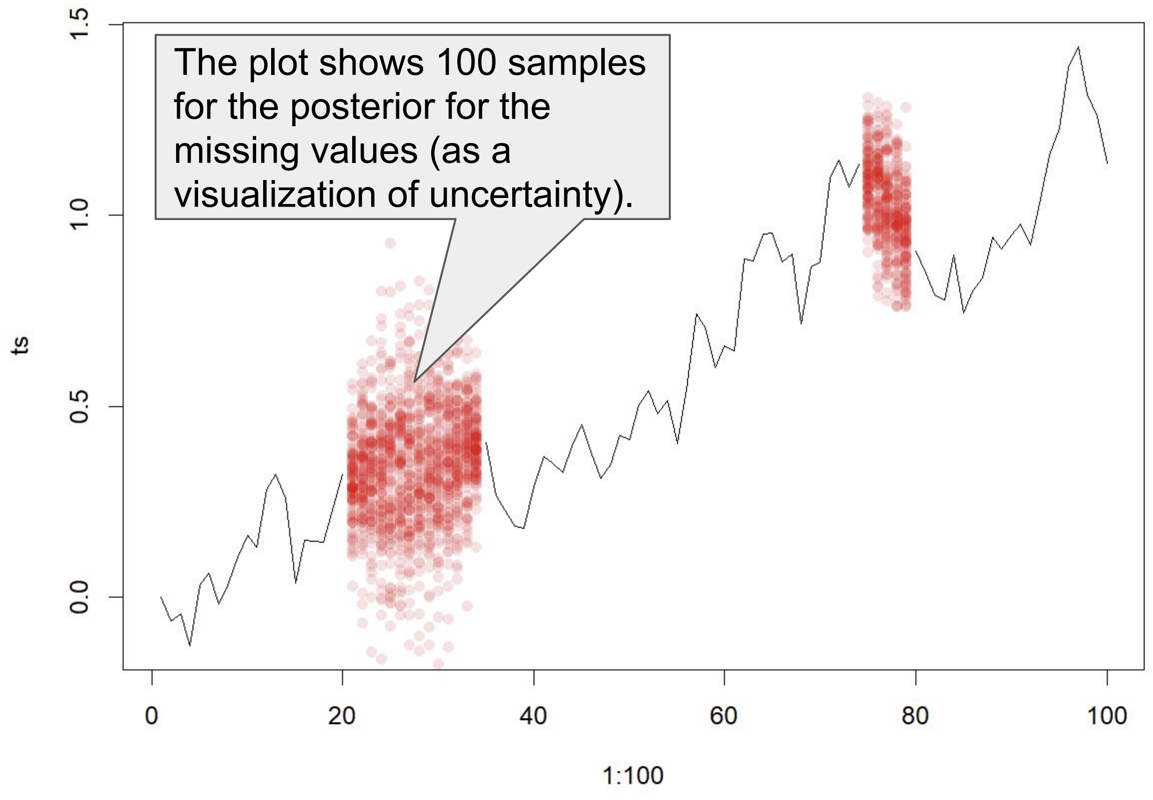
Implementing inference of the missing values (data imputation)

```
data {
 int<lower=0> N_obs;
 int<lower=0> N_mis;
 int<lower=1, upper=N_obs + N_mis> ii_obs[N_obs];
 int<lower=1, upper=N_obs + N_mis> ii_mis[N_mis];
 vector[N_obs] y_obs;
}

transformed data {
 int<lower=0> N = N_obs + N_mis;
}
```

```
parameters {
 vector[N_mis] y_mis;
 real<lower=0> sigma;
}
transformed parameters {
 vector[N] y;
 y[ii_obs] = y_obs;
 y[ii_mis] = y_mis;
}
model {
 sigma ~ gamma(1, 1);
 y[1] ~ normal(0, 100);
 y[2:N] ~ normal(y[1:(N - 1)], sigma);
}
```

Source: https://mc-stan.org/docs/2_28/stan-users-guide/sliced-missing-data.html

Depicting the observed data and the imputed data.



The plot shows 100 samples for the posterior for the missing values (as a visualization of uncertainty).

Glue code for reproduction

```
N_obs <- sum(!is.na(ts))
N_mis <- sum(is.na(ts))

ii_obs <- which(!is.na(ts))
ii_mis <- which(is.na(ts))

y_obs <- ts[!is.na(ts)]
```

Summary

- Logistic Regression to model variables following a binomial distribution.
- Dummy/Index variables as a way to use structure.
- Multilevel models implementing self adjusting priors on parameters.
- Imputation of missing values in a time series.