Report COVID19 Imperial Model

Måns Magnusson

2020-04-29

Contents

Summary / Take Away

- Using the increased number of covariates (model2) seem to be better, model-wise compared to baseline.
- There seem to be something problematic with the lockdown variable in model 1. The new model from the Imperial Collage group include an hiearchical prior here to accomodate different effects.

Current flaws to be fixed:

- IFR for Finland now set to IFR for Sweden
- Data only until the 10th of April.

Potential ways forward:

- Use hospitalizations instead of deaths. I think this has better quality and is more data.
- Focus on Finland (and Sweden) at county level, although data may be missing.
- Include serological studies to estimate cumulative infections, weekly.
- Include mobility data both together with other covariates as a single factor.
- Update data, ecdc and Oxford (until today instead of 10th of April)
- Test different alpha priors

Strange jumps in Rt

For some models there are som strange jumps in the Rt. In most cases it is small errors in data. This has hopefully been fixed with the Oxford data Version 5.

Finland is a special case. The difference comes from first having a general recommendation on movement restrictions, and then have a target movement restriction that is less certain.

Model descriptions

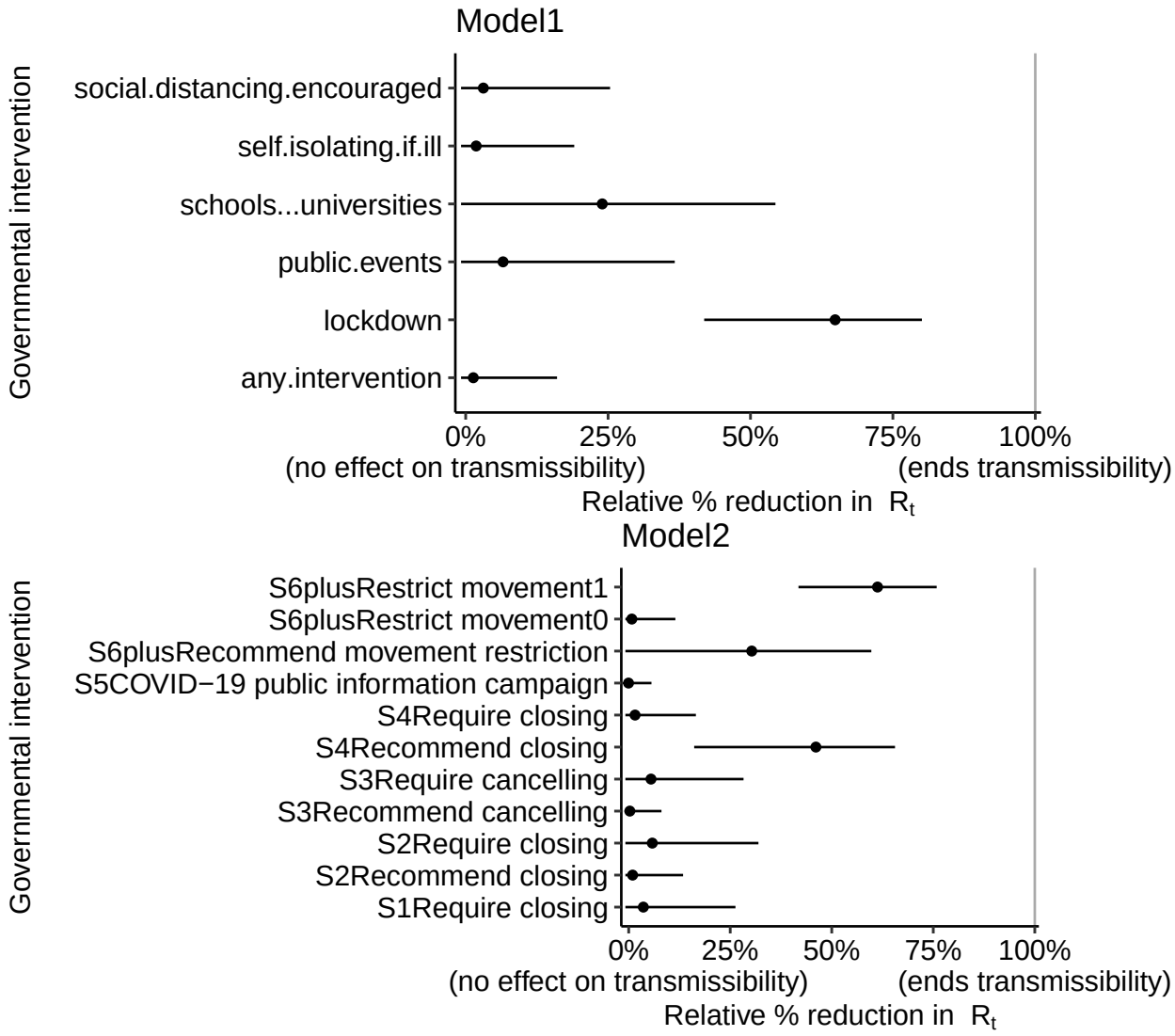
- Model 1: The baseline model, previously included, without partial pooling (the most recent model)
- Model 2: The model using S1 - S7 oxford variables
- Model 3: The model using SeverityIndex (continous variable)

Oxford covariate descriptions

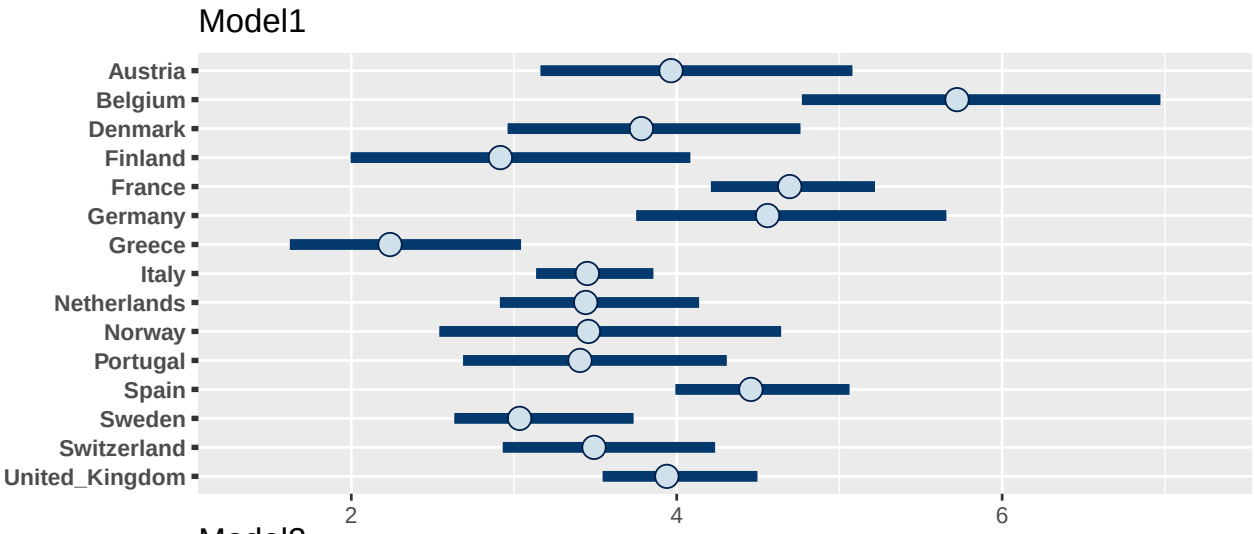
- S1: School closing
- S2: Workplace closing

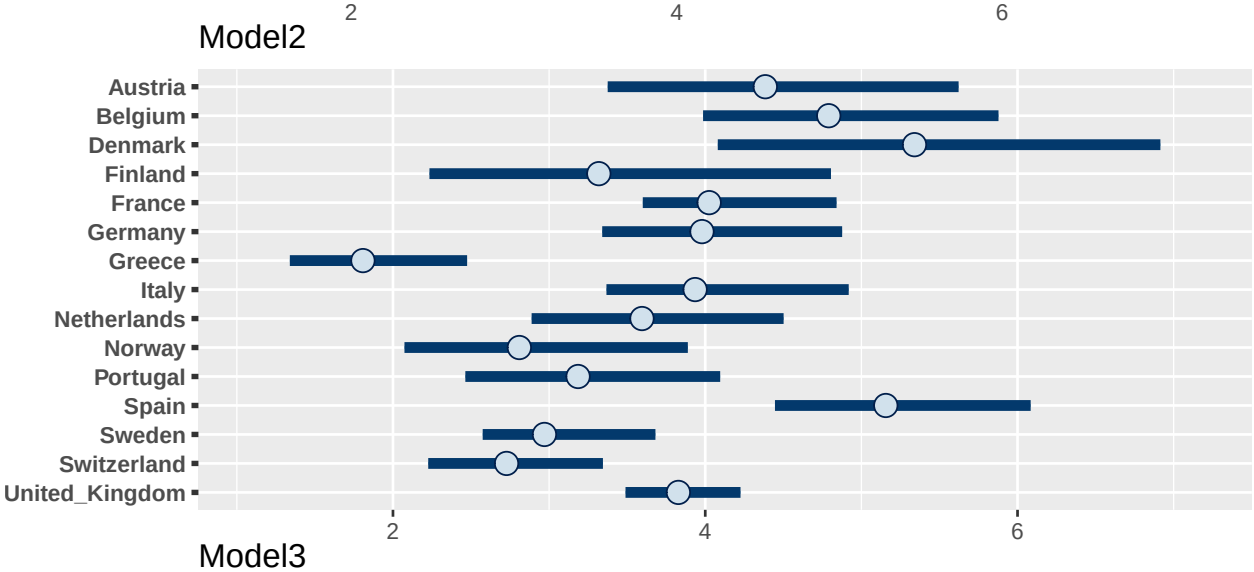
- S3: Cancel public events
- S4: Close public transport
- S5: Public information Campagins
- S6: Restrictions on internal movement (1 = General, 0 = geographic)

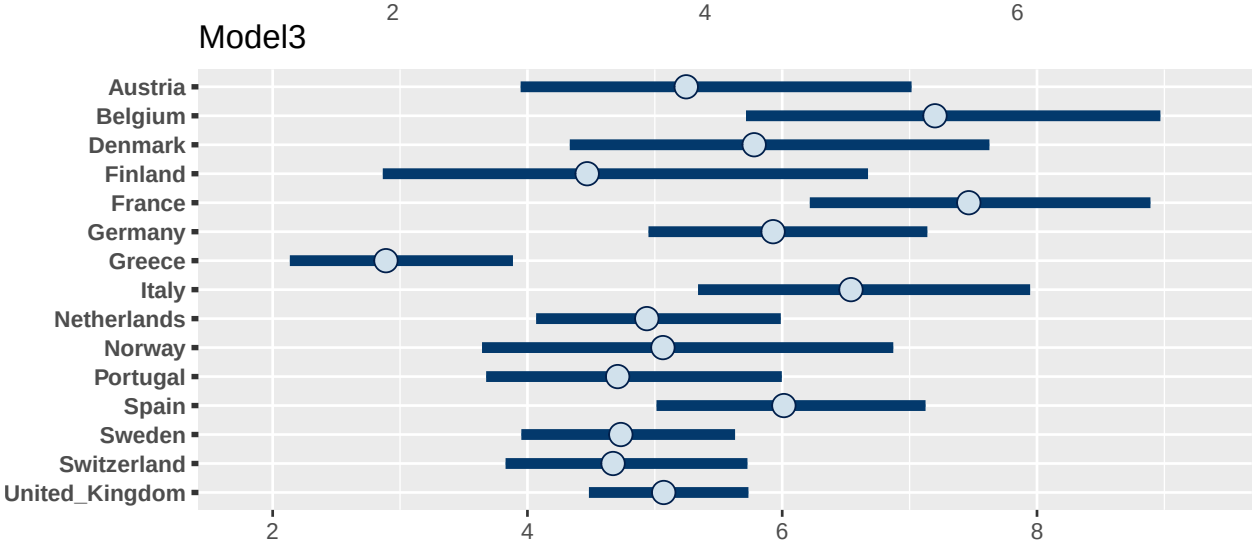
Covariate effects



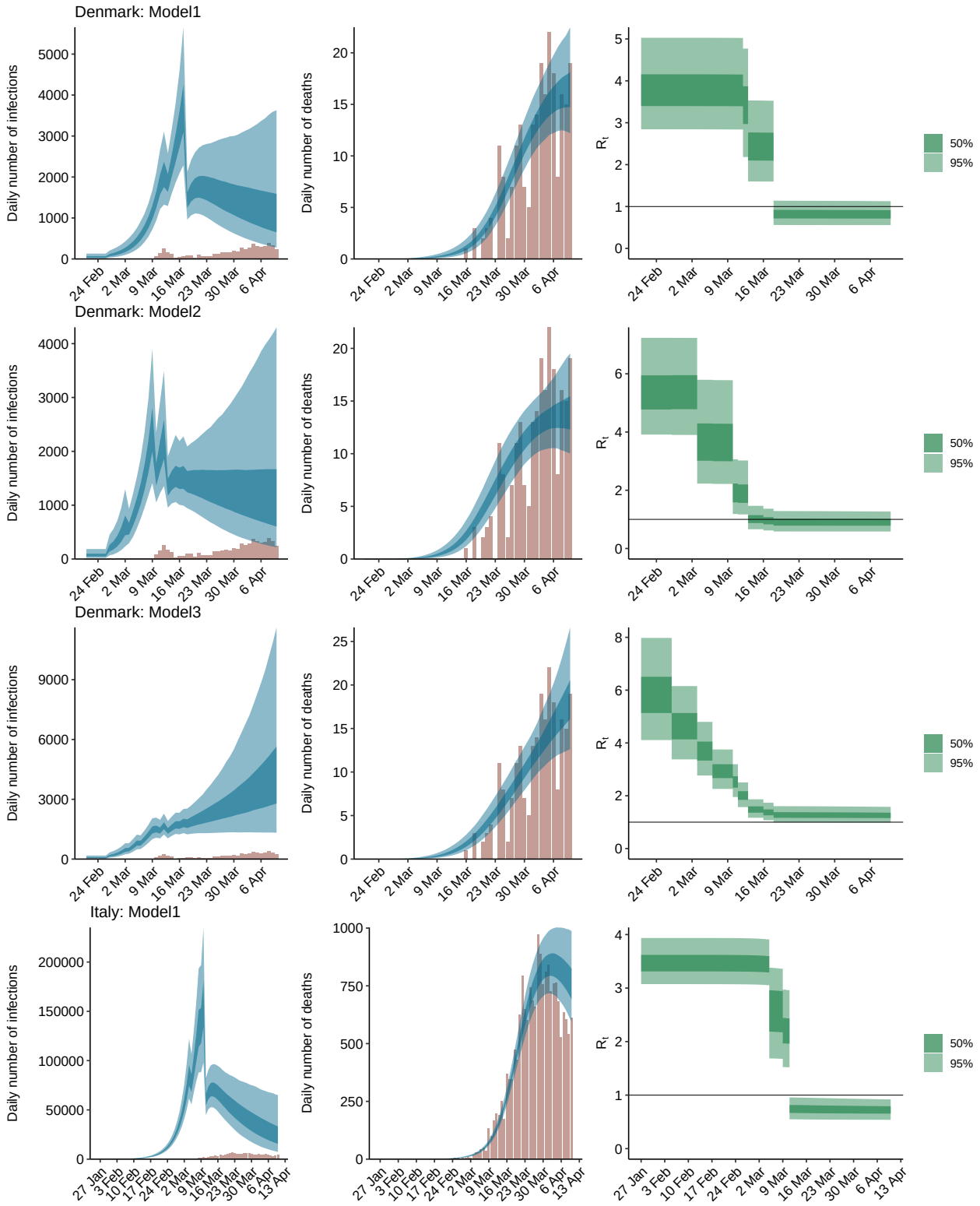
R0 for different countries

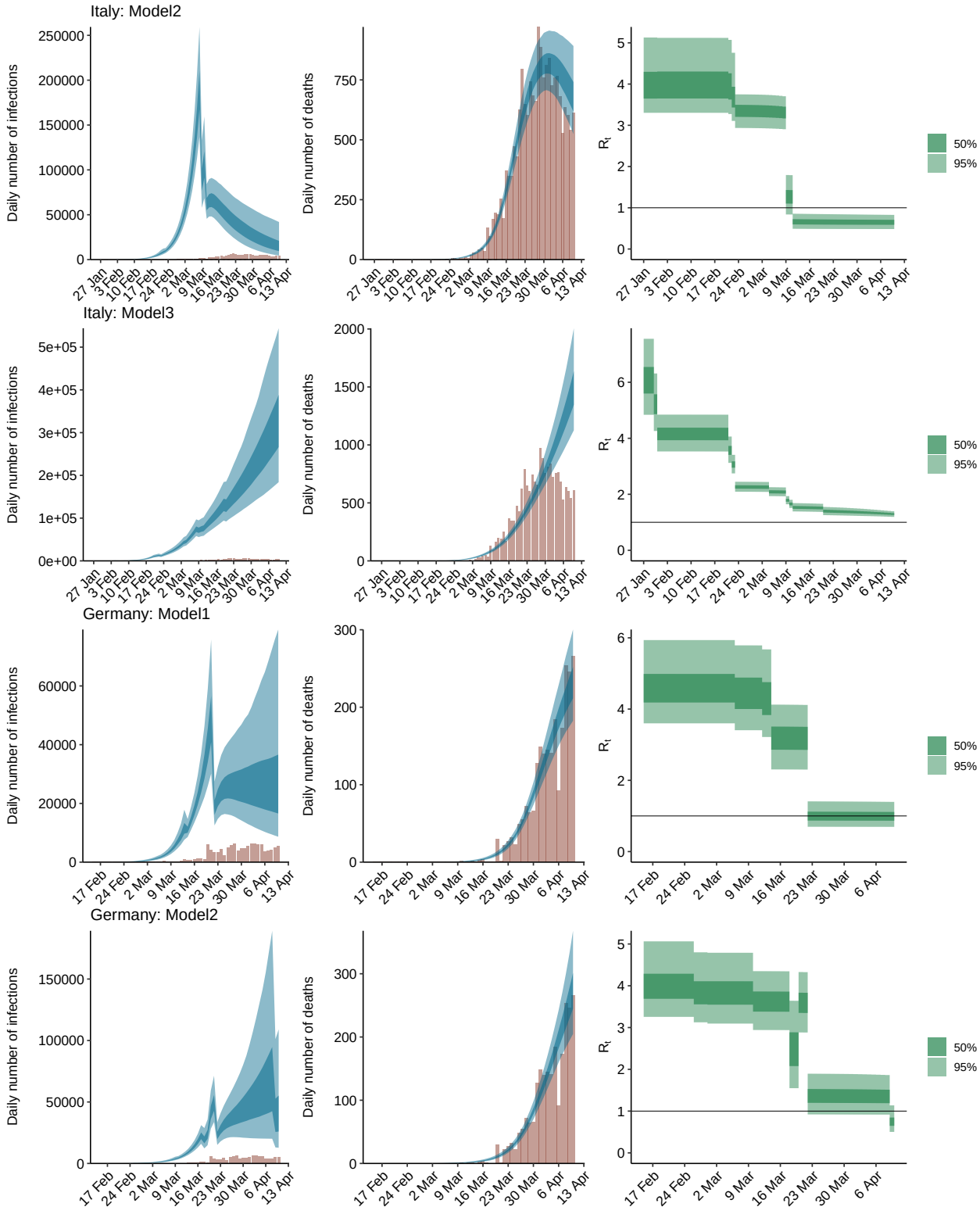


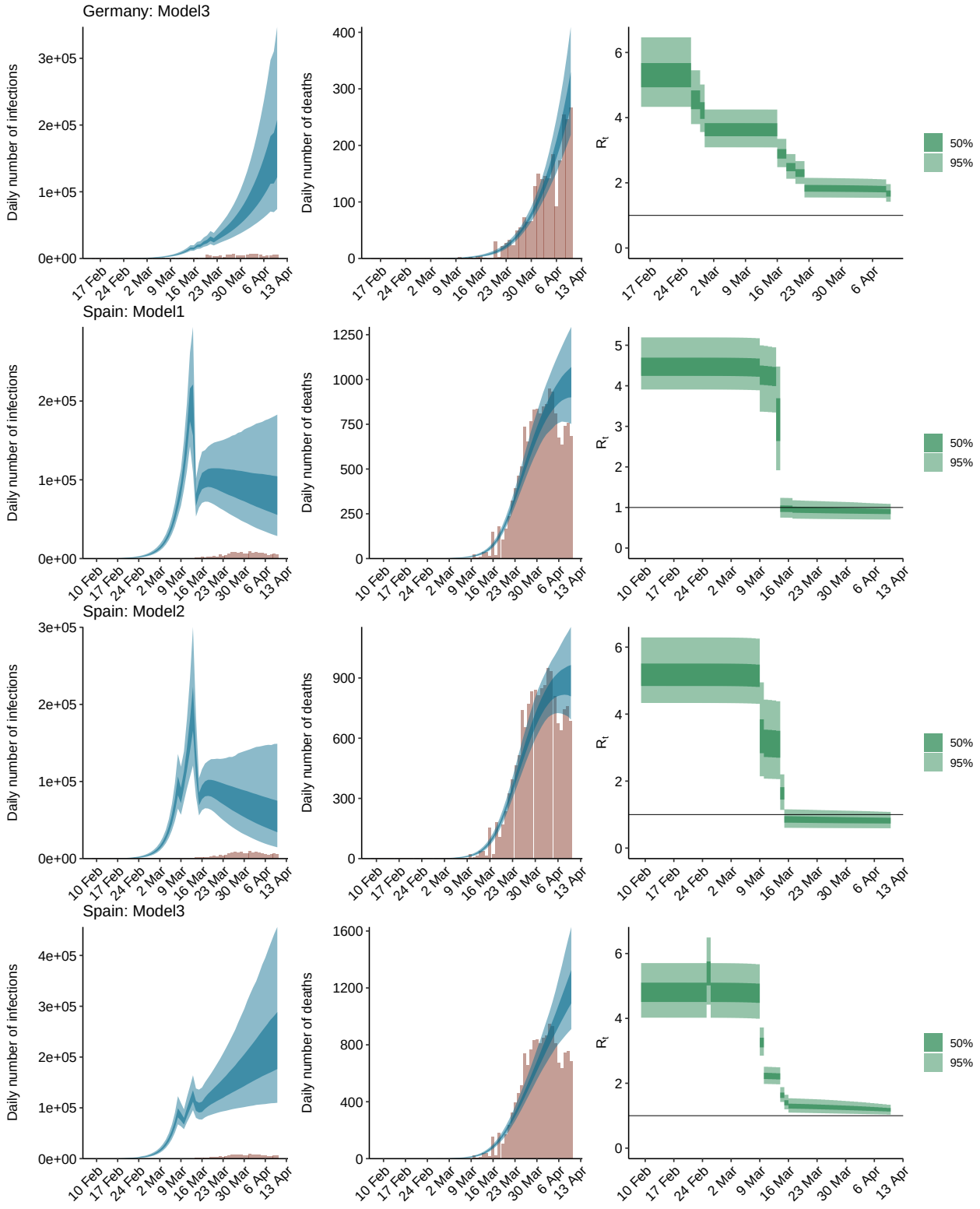


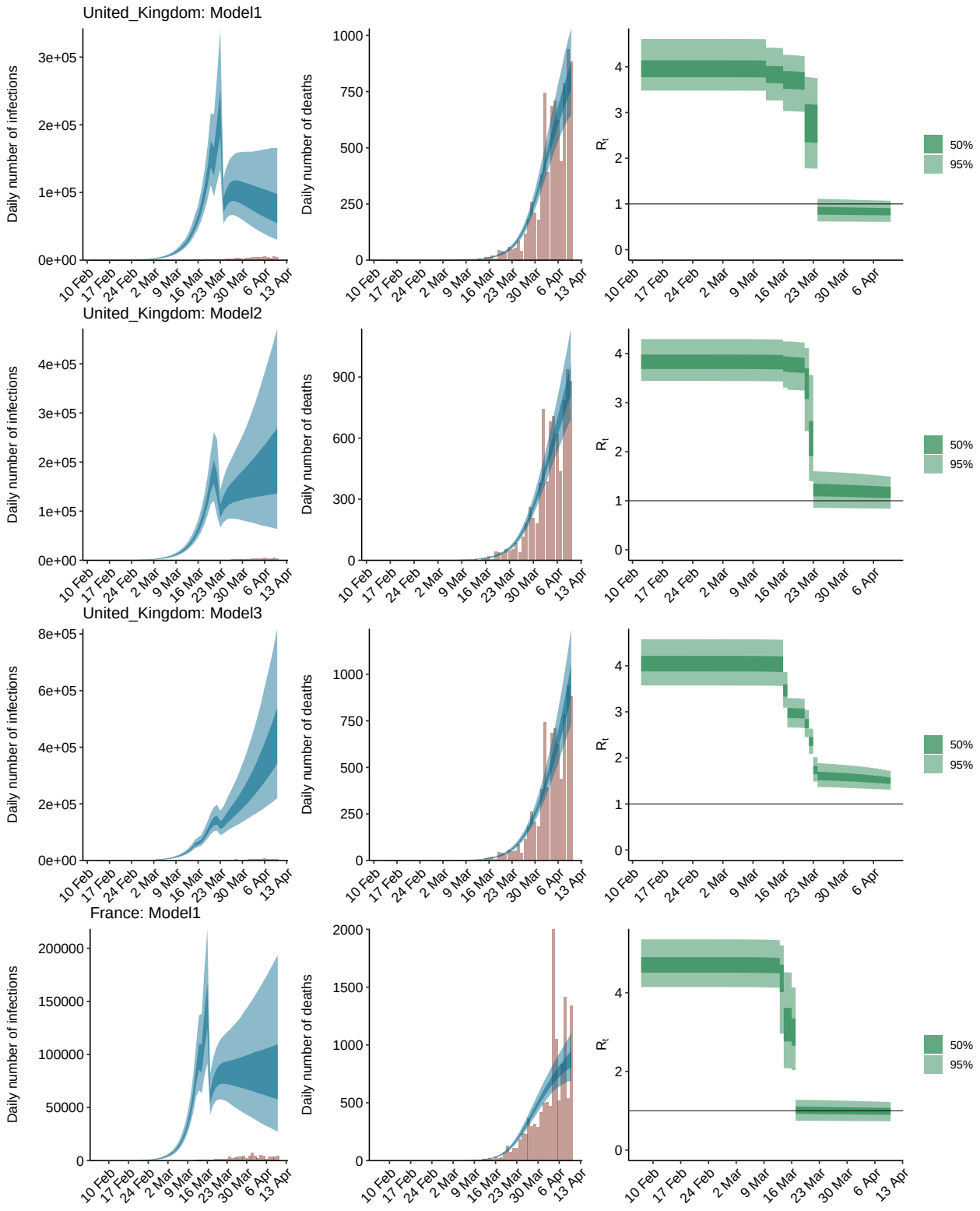


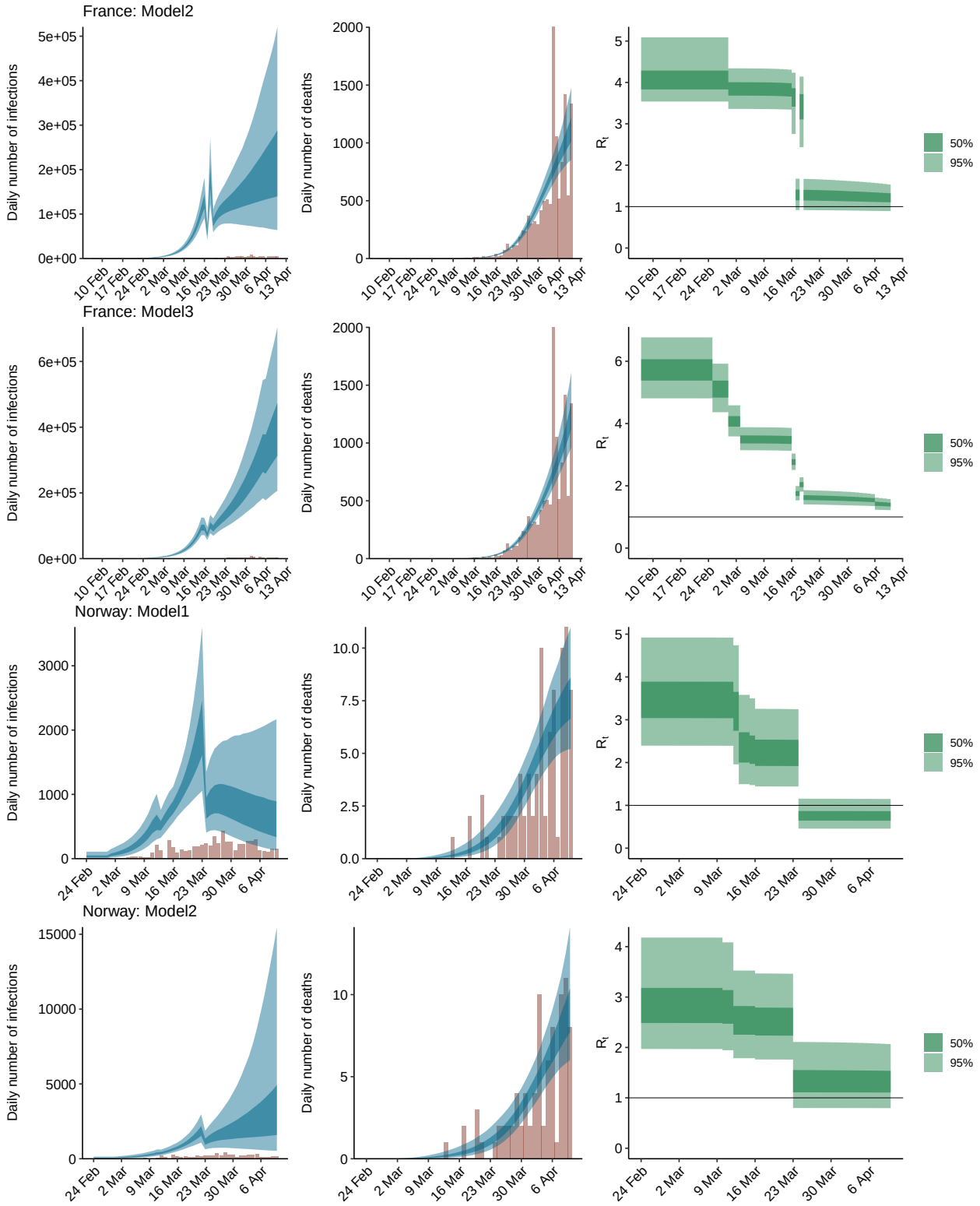
Infectious, deaths and Rt

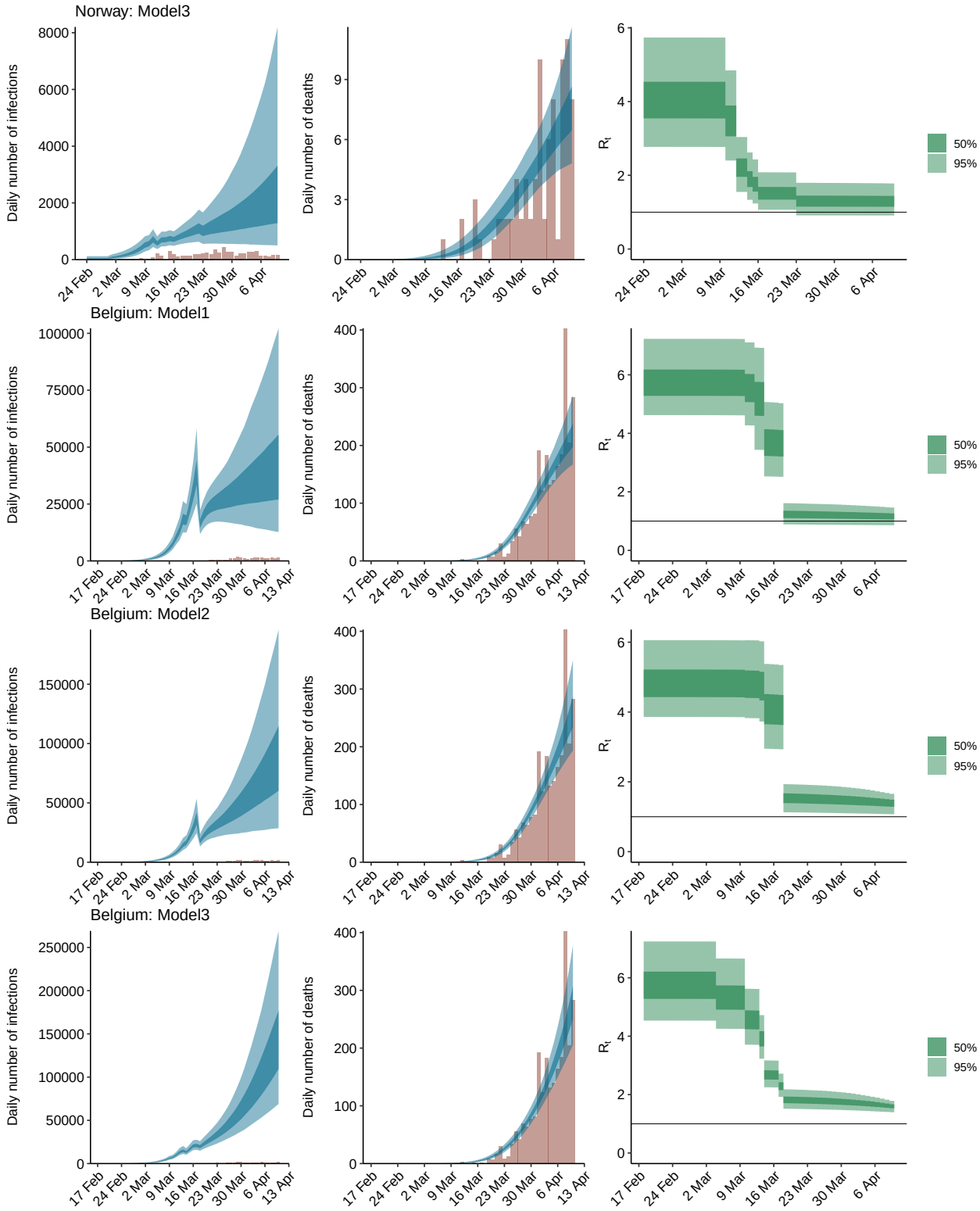


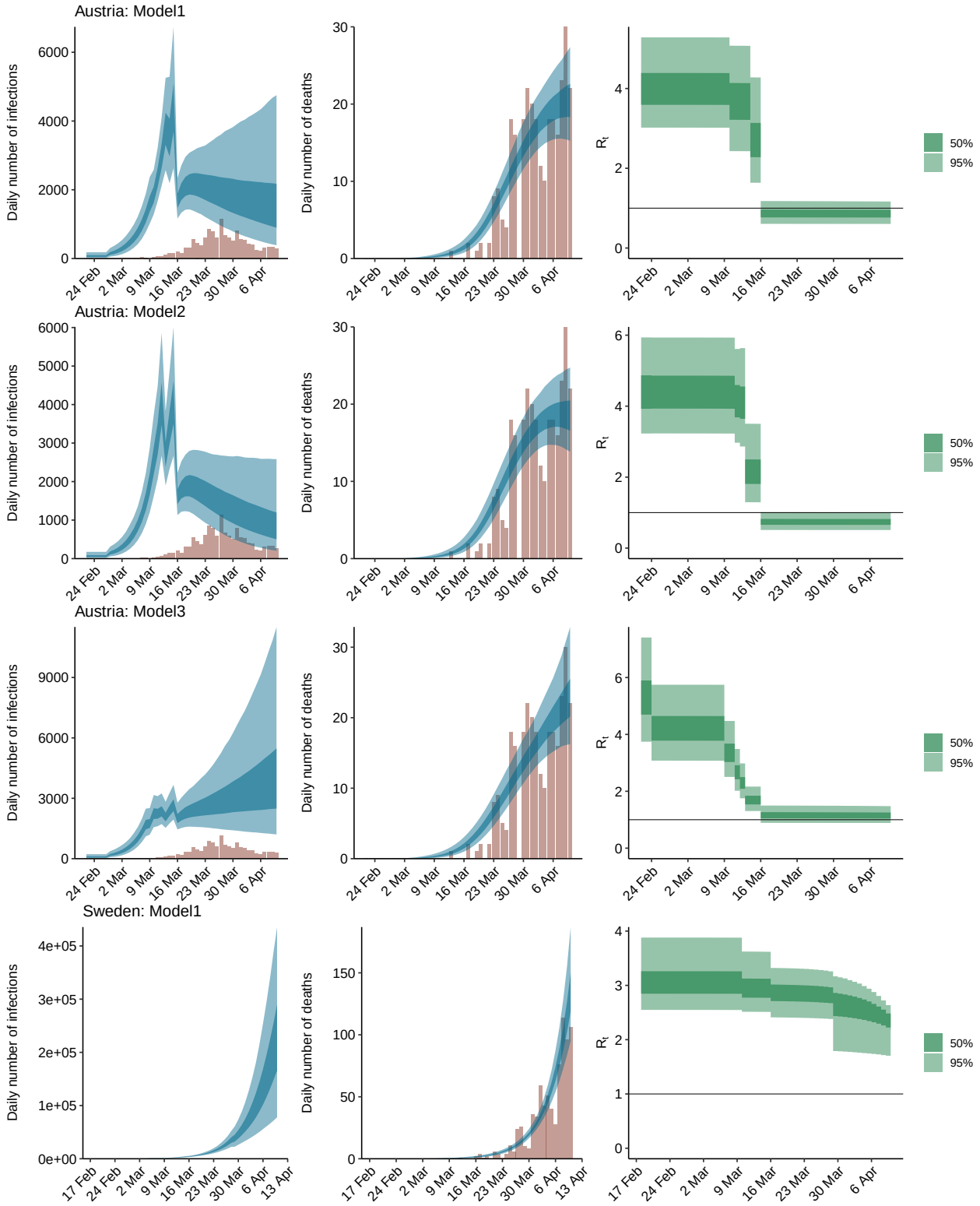

Italy: Model2
Daily number of infections
Daily number of deaths
R_t
Italy: Model3
Germany: Model1
Germany: Model2
50%
95%

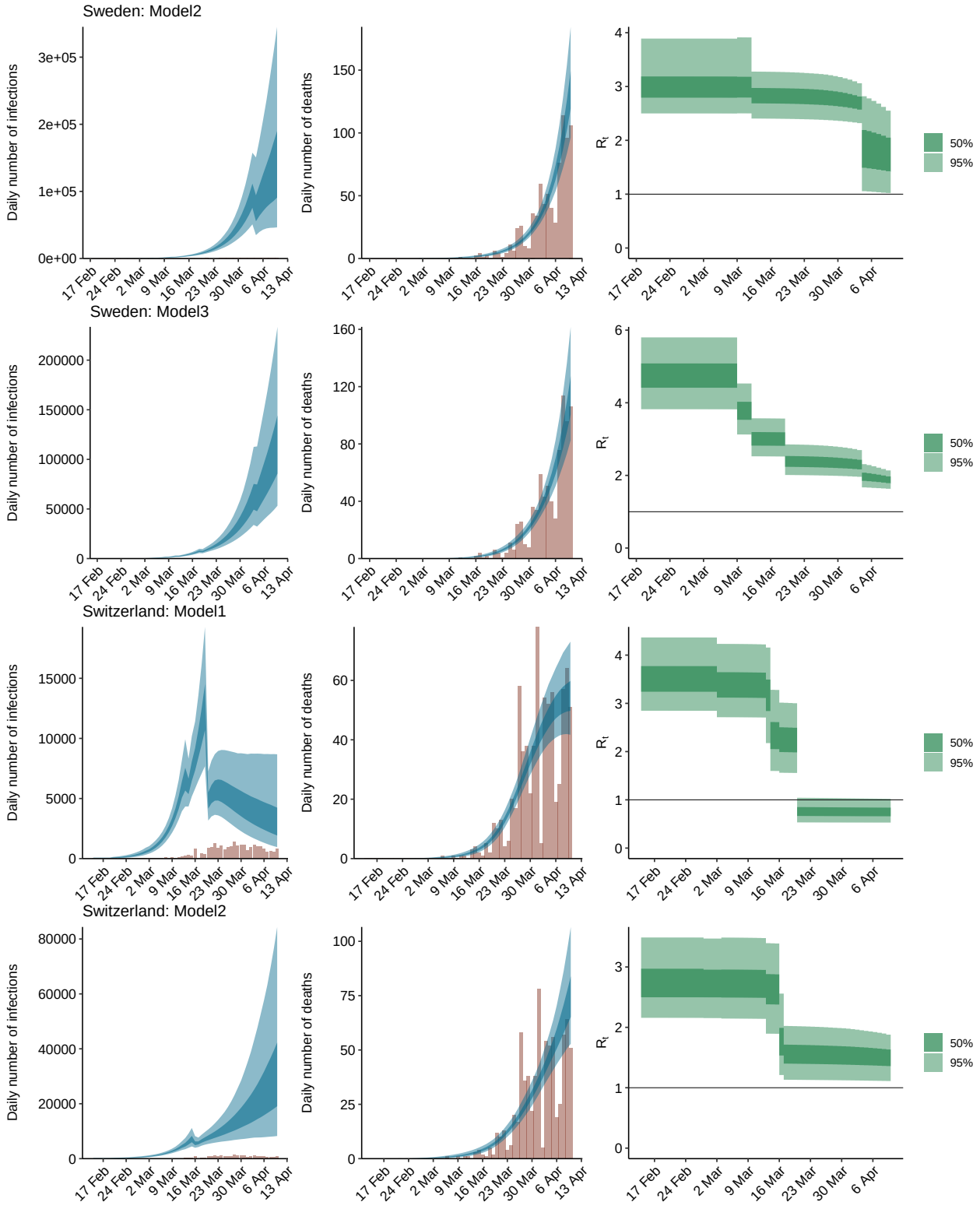

Germany: Model3
Spain: Model1
Spain: Model2
Spain: Model3
Daily number of infections
Daily number of deaths
R_t
50%
95%
17 Feb
24 Feb
2 Mar
9 Mar
16 Mar
23 Mar
30 Mar
6 Apr
13 Apr
10 Feb

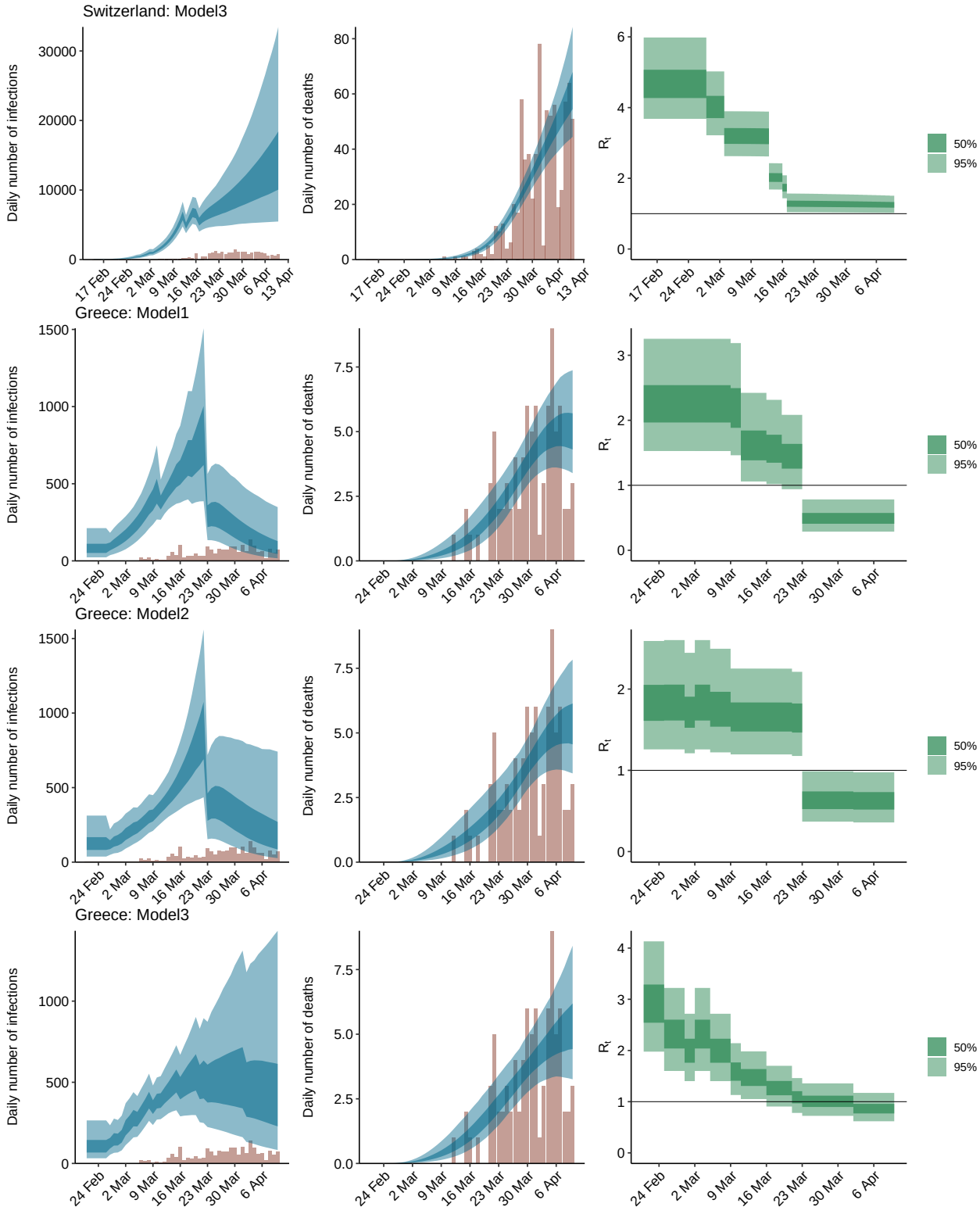

United_Kingdom: Model1
United_Kingdom: Model2
United_Kingdom: Model3
France: Model1
Daily number of infections
Daily number of deaths
R_t
50%
95%
10 Feb
17 Feb
24 Feb
2 Mar
9 Mar
16 Mar
23 Mar
30 Mar
6 Apr
13 Apr

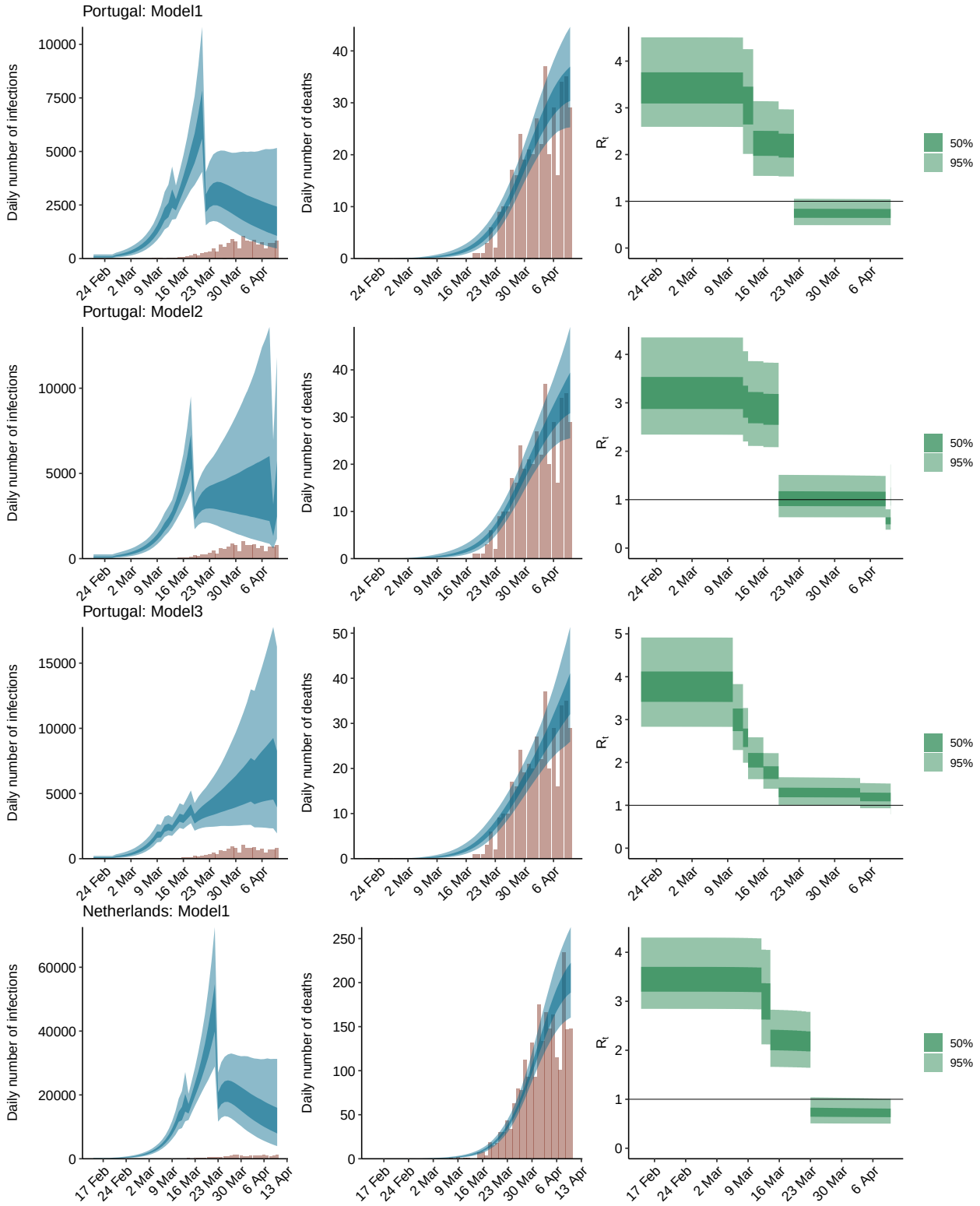

France: Model2
Daily number of infections
Daily number of deaths
R_t
France: Model3
Norway: Model1
Norway: Model2
50%
95%
10 Feb
17 Feb
24 Feb
2 Mar
9 Mar
16 Mar
23 Mar
30 Mar
6 Apr
13 Apr

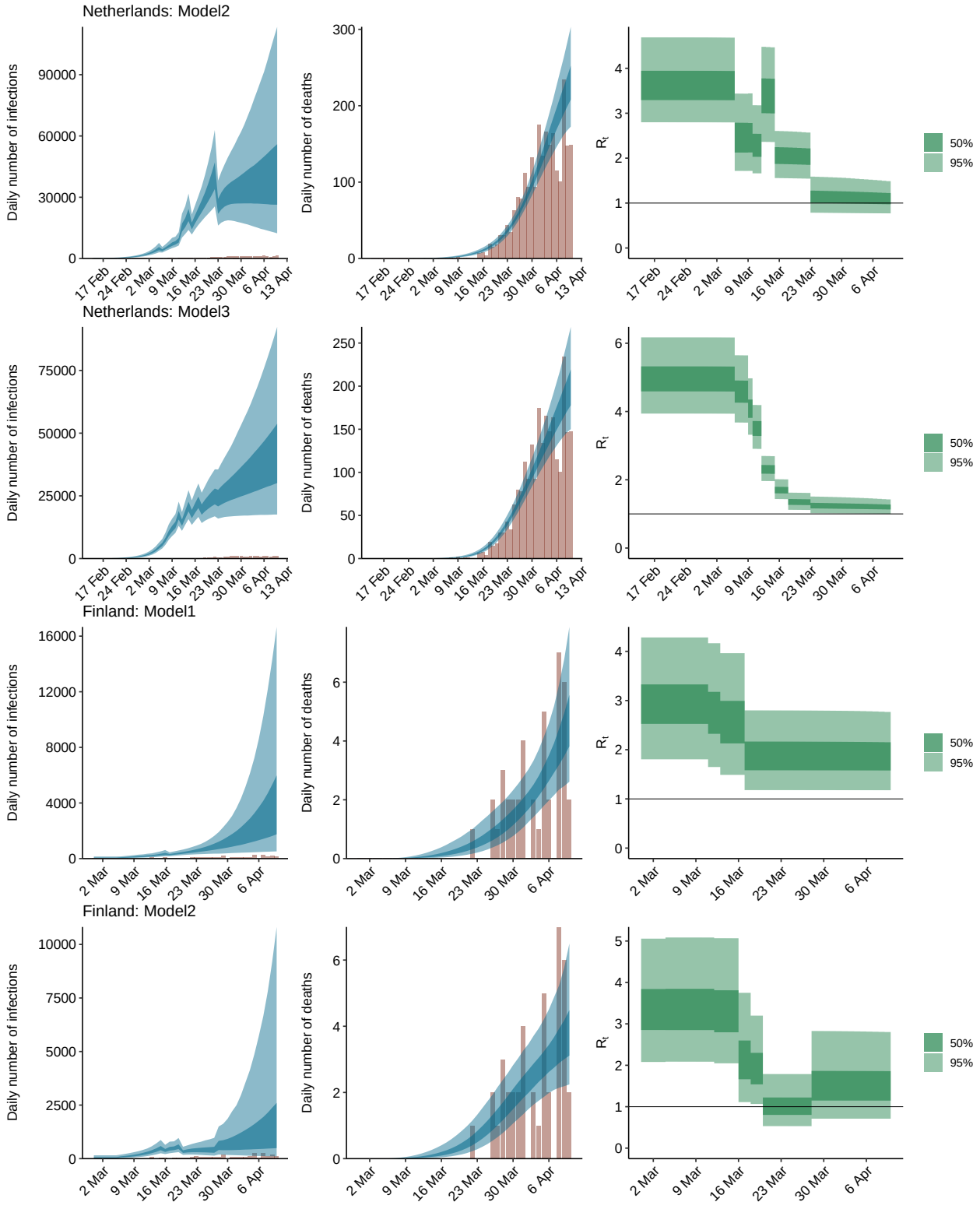

Norway: Model3
Daily number of infections
Daily number of deaths
R_t
Belgium: Model1
Belgium: Model2
Belgium: Model3
24 Feb
2 Mar
9 Mar
16 Mar
23 Mar
30 Mar
6 Apr
17 Feb
13 Apr
50%
95%

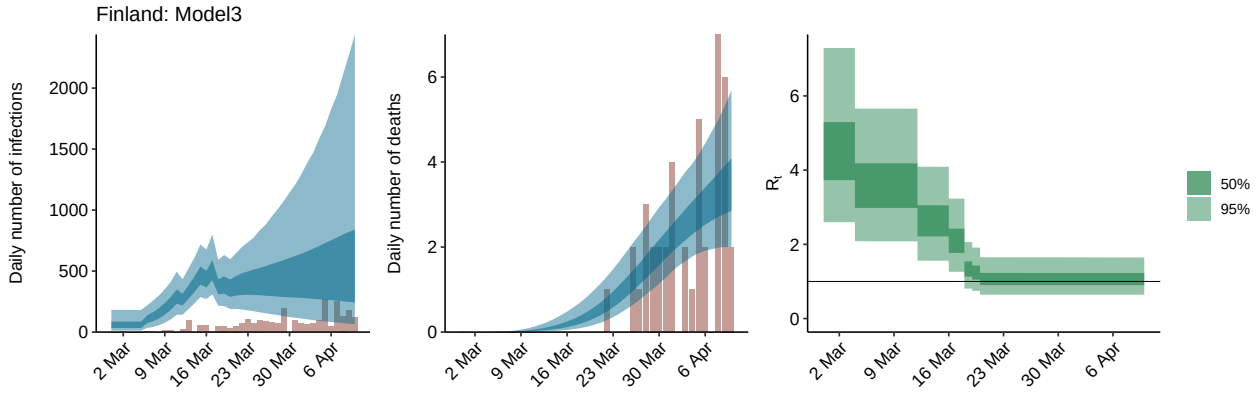

Austria: Model1
Austria: Model2
Austria: Model3
Sweden: Model1
Daily number of infections
Daily number of deaths
R_t
50%
95%
24 Feb
2 Mar
9 Mar
16 Mar
23 Mar
30 Mar
6 Apr
17 Feb
13 Apr


Sweden: Model2
Daily number of infections
Daily number of deaths
R_t
50%
95%
Sweden: Model3
Switzerland: Model1
Switzerland: Model2
17 Feb
24 Feb
2 Mar
9 Mar
16 Mar
23 Mar
30 Mar
6 Apr
13 Apr


Switzerland: Model3
Greece: Model1
Greece: Model2
Greece: Model3
Daily number of infections
Daily number of deaths
R_t
50%
95%
17 Feb
24 Feb
2 Mar
9 Mar
16 Mar
23 Mar
30 Mar
6 Apr
13 Apr


Portugal: Model1
Daily number of infections
Daily number of deaths
R_t
Portugal: Model2
Portugal: Model3
Netherlands: Model1
50%
95%
24 Feb
2 Mar
9 Mar
16 Mar
23 Mar
30 Mar
6 Apr
17 Feb
13 Apr


Netherlands: Model2
Daily number of infections
Daily number of deaths
R_t
Netherlands: Model3
Finland: Model1
Finland: Model2
50%
95%
17 Feb
24 Feb
2 Mar
9 Mar
16 Mar
23 Mar
30 Mar
6 Apr
13 Apr



Stan code

```
## S4 class stanmodel 'base_general_speed' coded as follows:
## data {
##   int <lower=1> M; // number of countries
##   int <lower=1> P; // number of covariates
##   int <lower=1> N0; // number of days for which to impute infections
##   int<lower=1> N[M]; // days of observed data for country m. each entry must be <= N2
##   int<lower=1> N2; // days of observed data + # of days to forecast
##   int cases[N2,M]; // reported cases
##   int deaths[N2, M]; // reported deaths -- the rows with i > N contain -1 and should be ignored
##   matrix[N2, M] f; // h * s
##   matrix[N2, P] X[M];
##   int EpidemicStart[M];
##   real pop[M];
##   real SI[N2]; // fixed pre-calculated SI using emprical data from Neil
## }
##
## transformed data {
##   vector[N2] SI_rev; // SI in reverse order
##   vector[N2] f_rev[M]; // f in reversed order
##
##   for(i in 1:N2)
##     SI_rev[i] = SI[N2-i+1];
##
##   for(m in 1:M){
##     for(i in 1:N2) {
##       f_rev[m, i] = f[N2-i+1,m];
##     }
##   }
## }
##
##
## parameters {
##   real<lower=0> mu[M]; // intercept for Rt
##   real<lower=0> alpha_hier[P]; // sudo parameter for the hier term for alpha
##   real<lower=0> kappa;
##   real<lower=0> y[M];
##   real<lower=0> phi;
##   real<lower=0> tau;
```

```
##   real <lower=0> ifr_noise[M];
## }
##
## transformed parameters {
##     vector[P] alpha;
##     matrix[N2, M] prediction = rep_matrix(0,N2,M);
##     matrix[N2, M] E_deaths  = rep_matrix(0,N2,M);
##     matrix[N2, M] Rt = rep_matrix(0,N2,M);
##     matrix[N2, M] Rt_adj = Rt;
##
##     {
##       matrix[N2,M] cumm_sum = rep_matrix(0,N2,M);
##       for(i in 1:P){
##         alpha[i] = alpha_hier[i] - ( log(1.05) / 6.0 );
##       }
##       for (m in 1:M){
##         /*
##         for (i in 2:N0){
##           cumm_sum[i,m] = cumm_sum[i-1,m] + y[m];
##         }
##         */
##         prediction[1:N0,m] = rep_vector(y[m],N0); // learn the number of cases in the first N0 days
##         cumm_sum[2:N0,m] = cumulative_sum(prediction[2:N0,m]);
##
##         Rt[,m] = mu[m] * exp(-X[m] * alpha);
##           Rt_adj[1:N0,m] = Rt[1:N0,m];
##         for (i in (N0+1):N2) {
##           /*
##           real convolution=0;
##           for(j in 1:(i-1)) {
##             convolution += prediction[j, m] * SI[i-j];
##           }
##           */
##           real convolution = dot_product(sub_col(prediction, 1, m, i-1), tail(SI_rev, i-1));
##
##           cumm_sum[i,m] = cumm_sum[i-1,m] + prediction[i-1,m];
##           Rt_adj[i,m] = ((pop[m]-cumm_sum[i,m]) / pop[m]) * Rt[i,m];
##           prediction[i, m] = Rt_adj[i,m] * convolution;
##         }
##
##         E_deaths[1, m]= 1e-15 * prediction[1,m];
##         for (i in 2:N2){
##           // for(j in 1:(i-1)){
##           //   E_deaths[i,m] += prediction[j,m] * f[i-j,m] * ifr_noise[m];
##           // }
##           E_deaths[i,m] = ifr_noise[m] * dot_product(sub_col(prediction, 1, m, i-1), tail(f_rev[m], i
##         }
##       }
##     }
## }
## model {
##   tau ~ exponential(0.03);
##   for (m in 1:M){
##       y[m] ~ exponential(1/tau);
```

```
##   }
##   phi ~ normal(0,5);
##   kappa ~ normal(0,0.5);
##   mu ~ normal(3.28, kappa); // citation: https://academic.oup.com/jtm/article/27/2/taaa021/5735319
##   alpha_hier ~ gamma(.1667,1);
##   ifr_noise ~ normal(1,0.1);
##   for(m in 1:M){
##     deaths[EpidemicStart[m]:N[m], m] ~ neg_binomial_2(E_deaths[EpidemicStart[m]:N[m], m], phi);
##    }
## }
##
## generated quantities {
##     matrix[N2, M] prediction0 = rep_matrix(0,N2,M);
##     matrix[N2, M] E_deaths0  = rep_matrix(0,N2,M);
##
##     {
##       matrix[N2,M] cumm_sum0 = rep_matrix(0,N2,M);
##       for (m in 1:M){
##          for (i in 2:N0){
##            cumm_sum0[i,m] = cumm_sum0[i-1,m] + y[m];
##         }
##         prediction0[1:N0,m] = rep_vector(y[m],N0);
##         for (i in (N0+1):N2) {
##           real convolution0 = 0;
##           for(j in 1:(i-1)) {
##             convolution0 += prediction0[j, m] * SI[i-j];
##           }
##           cumm_sum0[i,m] = cumm_sum0[i-1,m] + prediction0[i-1,m];
##           prediction0[i, m] =  ((pop[m]-cumm_sum0[i,m]) / pop[m]) * mu[m] * convolution0;
##         }
##
##         E_deaths0[1, m] = uniform_rng(1e-16, 1e-15);
##         for (i in 2:N2){
##           for(j in 1:(i-1)){
##             E_deaths0[i,m] += prediction0[j,m] * f[i-j,m] * ifr_noise[m];
##           }
##         }
##       }
##     }
## }
##
```