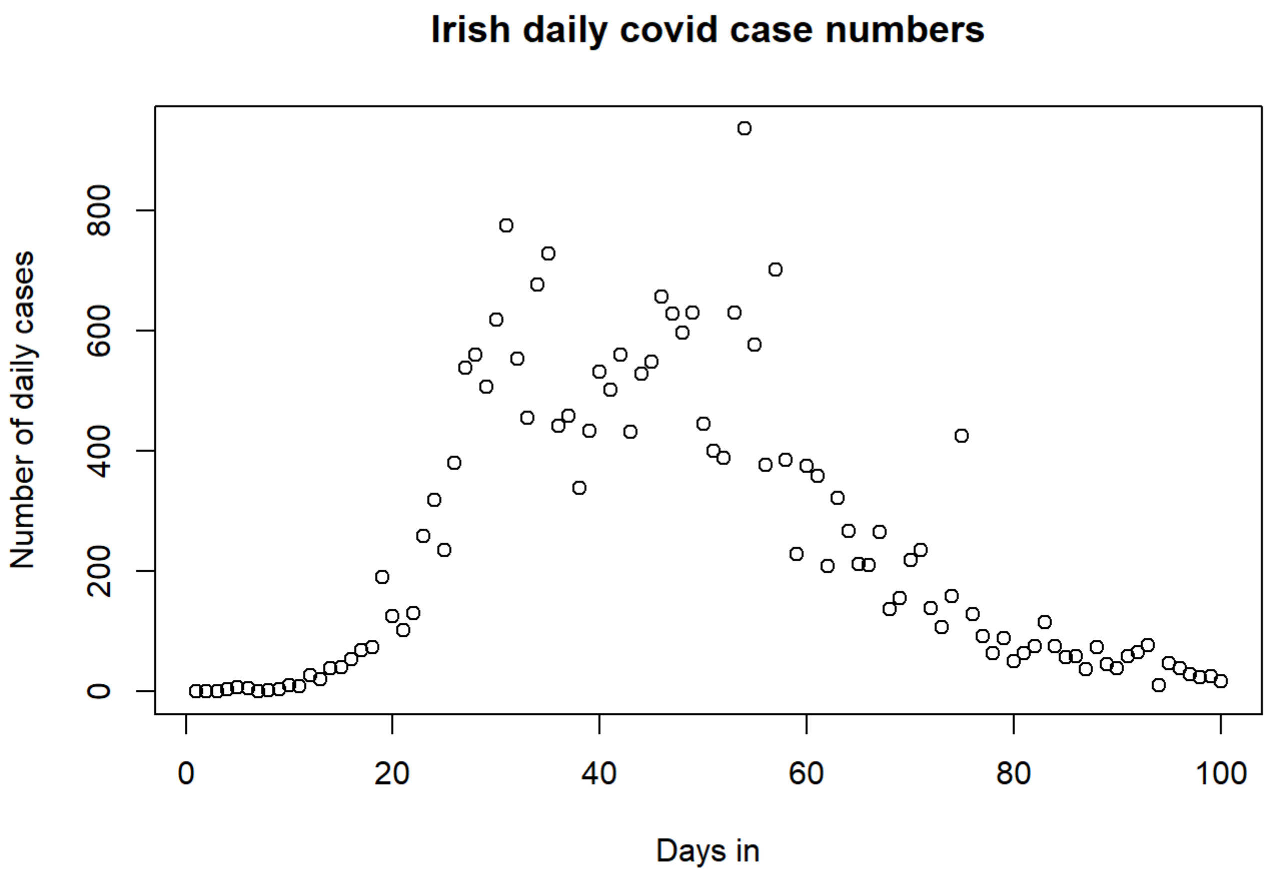
Bayesian Analysis of Covid Data

Siddhesh Bagwe

2023-03-09

```
ireland <- read.csv("./ireland1.txt")
cases <- ireland[1:100,1]
days = 1:100
plot(days, cases, xlab="Days in", ylab="Number of daily cases", main="Irish daily covid case
numbers")
```



The plot shows the cases over the days. We see that the number of cases kept on increasing with the days. After about the 50-60 day the number of cases started dropping. We will a Bayesian model for the same to predict the number of cases.

```
set.seed(123)
library(rstan)
write("
data {
  int<lower=1> N;
  int y[N];
  vector[N] t;
}

parameters {
  real<lower=0> theta1;
  real<lower=0,upper=100> theta2;
  real<lower=0,upper=1> theta3;
}

model {
    target += normal_lpdf(theta1 | 1e3,1e5);
    target += normal_lpdf(theta2| 50, 100);
    for (n in 1:N){
    target += poisson_lpmf(y[n] | theta1 * theta3 * exp(-theta3 * (t[n] - theta2)) / pow(1 +
exp(-theta3 * (t[n] - theta2)), 2));
    }
}

generated quantities {
  vector[N] y_pred;
  real log_lik[N];
  for (n in 1:N){
    y_pred[n] = poisson_rng( theta1 * theta3 * exp(-theta3 * (t[n] - theta2)) / pow(1 + exp(-
theta3 * (t[n] - theta2)), 2.0) );
    log_lik[n]=poisson_lpmf(y[n] | theta1 * theta3 * exp(-theta3 * (t[n] - theta2)) / pow(1 +
exp(-theta3 * (t[n] - theta2)), 2));
}
}"
,"m1.stan")
```

Model Description

The above model is defined for the logistic function

```
      yt - Po(lambdal(t))
```

- theta1, theta2 are normally distributed while theta 3 is uniformly distributed.
- We will use the y_pred generated to get the prediction from the model. The same can be used to get the accuracy of the model.
- The log_lik function is defined to compare the accuracy of the model.

```
set.seed(123)
data <- list(N = NROW(cases),
            y = cases,
            t = days)

fit <- stan(file="m1.stan", data = data, iter=500)
```

- The model is fit for 500 iterations using the given data.
- Here t is the number of days and y is number of cases.

```
print(fit, pars=c("theta1", "theta2", "theta3","y_pred[1]"))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=500; warmup=250; thin=1;
## post-warmup draws per chain=250, total post-warmup draws=1000.
##
##               mean se_mean     sd     2.5%      25%      50%      75%    97.5%
## theta1    25525.85    5.49 152.67 25239.09 25429.13 25522.25 25619.72 25830.41
## theta2       45.94    0.00   0.11    45.72    45.86    45.94    46.02    46.17
## theta3        0.10    0.00   0.00     0.10     0.10     0.10     0.10     0.10
## y_pred[1]    29.07    0.17   5.43    19.00    25.00    29.00    32.25    40.00
##           n_eff Rhat
## theta1      774 1.01
## theta2     1284 1.00
## theta3      827 1.00
## y_pred[1]   971 1.00
##
## Samples were drawn using NUTS(diag_e) at Mon May 29 22:02:57 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Model Summary

Here we see the summary of our model. - The model is fit with 4 chains each with 500 iterations. Out of this 250 are used as warm-up. Thus we get 250 post-warm up draws per chain giving a total of 1000 post-warm up draws. - The parameters theta1, theta2 and theta3 define the function in our model. We see the mean,Standard error, standard deviation as well as the quantiles for our parameters. These parameters are then used to get the prediction from the model. As the data is of 100 days, we have the y_pred from 1 to 100. Here we see the mean, standard deviation as well as the quantiles for prediction of day 1(y_pred[1]). - n_eff is the effective sample_size. A sample size of greater than 10% of the total post-warm draws is acceptable (100 for this model.). The summary shows that all the parameters have an acceptable n_eff. - R_hat is the measure of convergence of the chains. The acceptable value of Rhat is less than or equal to 1.1. As all the parameters have Rhat less than 1.1 the model is acceptable.

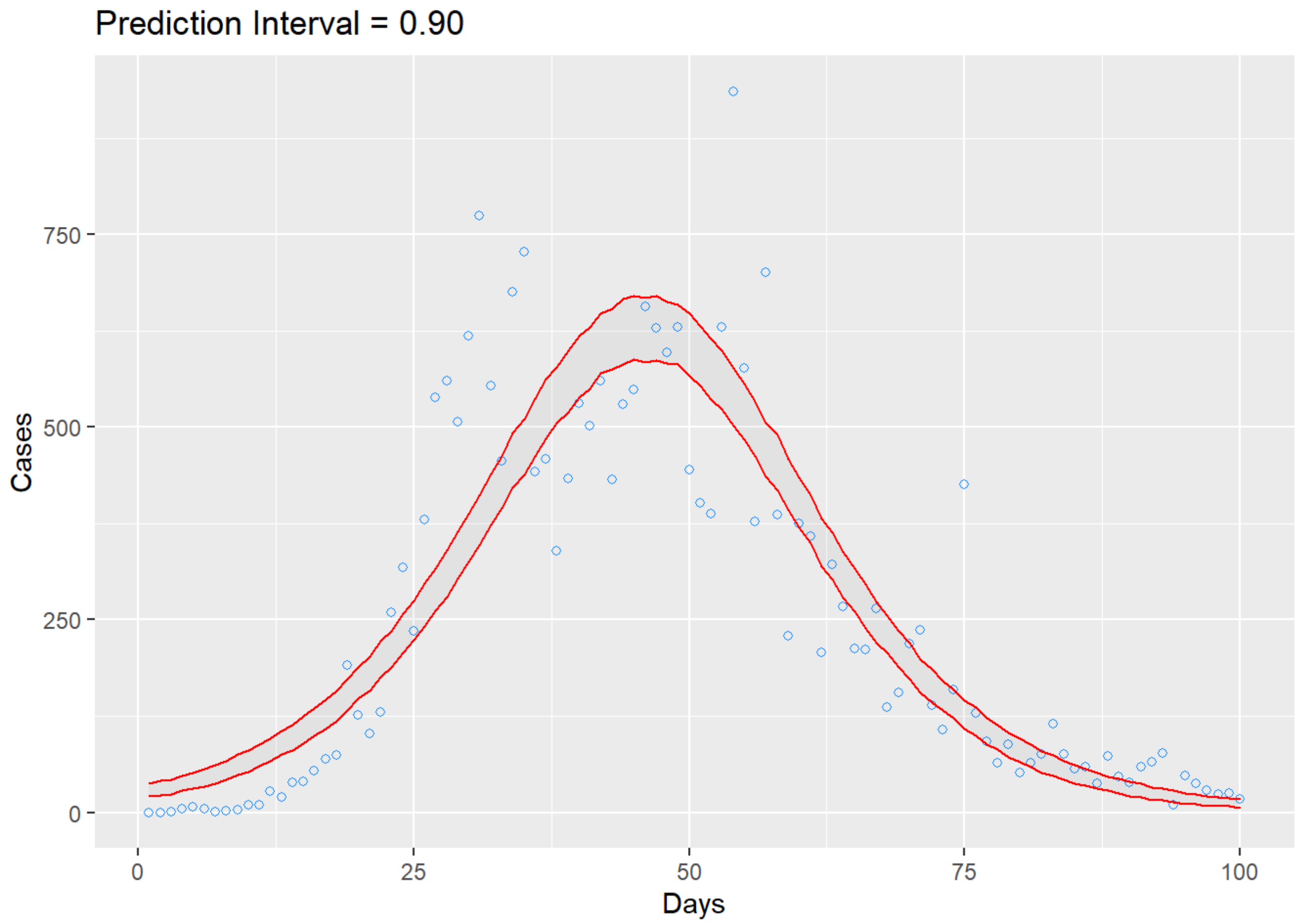
```
y_pred <- as.matrix(fit, pars=c("y_pred"))
y_phdi = HDInterval::hdi(y_pred, credMass=0.90)
pi_l = y_phdi[1,]
pi_u = y_phdi[2,]
print(y_phdi)
```

```
##       parameters
##        y_pred[1] y_pred[2] y_pred[3] y_pred[4] y_pred[5] y_pred[6] y_pred[7]
##   lower        21        22        24        29        31        34        37
##   upper        38        41        43        48        52        57        61
##       parameters
##        y_pred[8] y_pred[9] y_pred[10] y_pred[11] y_pred[12] y_pred[13]
##   lower        42        49         53         60         67         76
##   upper        67        75         81         88         96        106
##       parameters
##        y_pred[14] y_pred[15] y_pred[16] y_pred[17] y_pred[18] y_pred[19]
##   lower         81         90         99        108        119        132
##   upper        113        125        135        146        158        173
##       parameters
##        y_pred[20] y_pred[21] y_pred[22] y_pred[23] y_pred[24] y_pred[25]
##   lower        148        158        176        189        207        224
##   upper        189        202        223        235        258        275
##       parameters
##        y_pred[26] y_pred[27] y_pred[28] y_pred[29] y_pred[30] y_pred[31]
##   lower        242        262        280        304        325        347
##   upper        297        317        341        365        388        412
##       parameters
##        y_pred[32] y_pred[33] y_pred[34] y_pred[35] y_pred[36] y_pred[37]
##   lower        373        395        422        438        463        485
##   upper        440        462        493        511        537        562
##       parameters
##        y_pred[38] y_pred[39] y_pred[40] y_pred[41] y_pred[42] y_pred[43]
##   lower        505        520        539        550        570        575
##   upper        578        599        618        630        648        654
##       parameters
##        y_pred[44] y_pred[45] y_pred[46] y_pred[47] y_pred[48] y_pred[49]
##   lower        582        588        584        587        583        581
##   upper        667        670        668        671        663        659
##       parameters
##        y_pred[50] y_pred[51] y_pred[52] y_pred[53] y_pred[54] y_pred[55]
##   lower        566        554        536        523        502        484
##   upper        647        631        614        599        576        556
##       parameters
##        y_pred[56] y_pred[57] y_pred[58] y_pred[59] y_pred[60] y_pred[61]
##   lower        462        436        418        392        370        349
##   upper        534        506        490        459        434        412
##       parameters
##        y_pred[62] y_pred[63] y_pred[64] y_pred[65] y_pred[66] y_pred[67]
##   lower        319        303        279        261        239        220
##   upper        381        363        338        317        296        273
##       parameters
##        y_pred[68] y_pred[69] y_pred[70] y_pred[71] y_pred[72] y_pred[73]
##   lower        207        188        173        155        143        132
##   upper        256        235        220        199        186        171
##       parameters
##        y_pred[74] y_pred[75] y_pred[76] y_pred[77] y_pred[78] y_pred[79]
##   lower        122        108        100         88         82         72
##   upper        159        145        136        122        113        104
##       parameters
##        y_pred[80] y_pred[81] y_pred[82] y_pred[83] y_pred[84] y_pred[85]
##   lower         66         59         52         47         43         37
```

```
##   upper         96         88         79         74         67         61
##        parameters
##         y_pred[86] y_pred[87] y_pred[88] y_pred[89] y_pred[90] y_pred[91]
##   lower         35         31         28         25         21         20
##   upper         57         52         46         44         40         37
##        parameters
##         y_pred[92] y_pred[93] y_pred[94] y_pred[95] y_pred[96] y_pred[97]
##   lower         16         16         13         11         11          8
##   upper         33         31         28         25         24         21
##        parameters
##         y_pred[98] y_pred[99] y_pred[100]
##   lower          8          8           6
##   upper         20         19          17
## attr(,"credMass")
## [1] 0.9
```

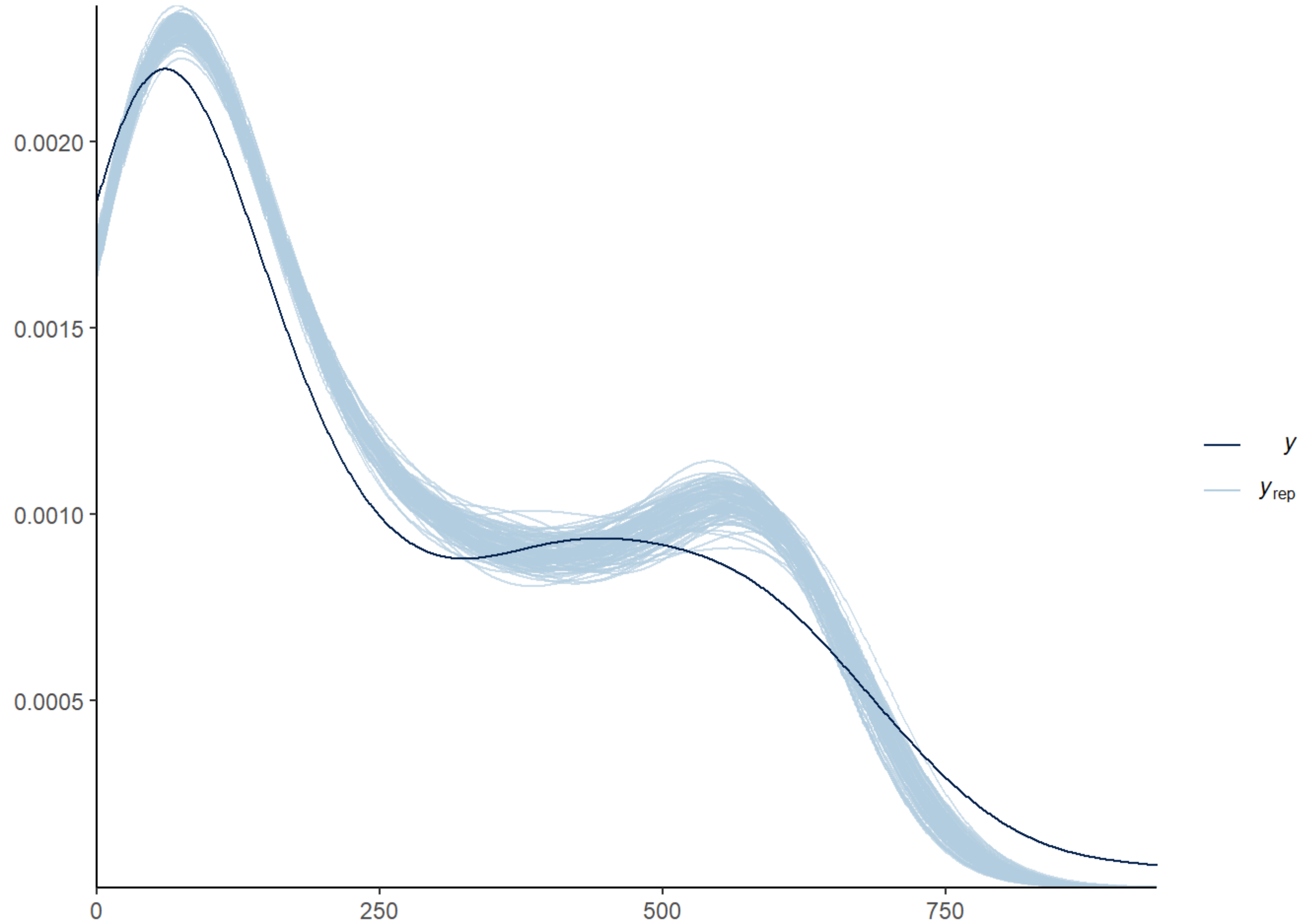
We can see the 90% posterior interval of our model. We will use this to check the accuracy of our model.

```
d1 <- as.data.frame(data)
library(ggplot2)
p <- ggplot()

p2 <- p +
  geom_point(data = d1,
      aes(t, y), shape = 1, color = 'dodgerblue') +
  ggtitle("Prediction Interval = 0.90")+
  geom_ribbon(data = d1,
      mapping = aes(t,ymin=pi_l, ymax=pi_u), alpha = .05,color = 'red')+xlab("Days")+ ylab("C
ases")

p2
```



The plot shows the ribbon of the 90% interval of our predicted data and the plot of the original data. We can see that though our model does a decent job in predicting the curve, the real values do not lie in the 90% interval for most of our data. This indicates that the model may not be a perfect fit for our data. We will check this further using the density plot of our original data to that of the posterior prediction.

```
library(bayesplot)
ppc_dens_overlay(d1$y, y_pred[1:100,])+ theme_classic()
```



Again we can see that, though the curve is similar there is quite a difference between the original and predicted values. This indicates that the model may not be a great fit.

Next we will try to use our model to make the predication for number of cases for the next five days.

```
set.seed(123)
library(rstan)
write("
data {
  int<lower=1> N;
  int<lower=1> N1;
  int y[N];
  vector[N] t;
  vector[N1] t_new;
}

parameters {
  real<lower=0> theta1;
  real<lower=0,upper=100> theta2;
  real<lower=0,upper=1> theta3;
}

model {
    target += normal_lpdf(theta1 | 1e3,1e5);
    target += normal_lpdf(theta2| 50, 100);
    for (n in 1:N){
    target += poisson_lpmf(y[n] | theta1 * theta3 * exp(-theta3 * (t[n] - theta2)) / pow(1 +
exp(-theta3 * (t[n] - theta2)), 2));
    }
}

generated quantities {
  vector[N1] y_pred;
  for (n in 1:N1){
    y_pred[n] = poisson_rng( theta1 * theta3 * exp(-theta3 * (t_new[n] - theta2)) / pow(1 + e
xp(-theta3 * (t_new[n] - theta2)), 2.0) );
}
}"
,"m2.stan")
```

Here the model is updated to predict values for t_new which is for days 101-105. We have to define the new data to fit our model.

```
set.seed(123)
t_new <- c(101,102,103,104,105)
data1 <- list(N = NROW(cases),
            N1 = as.integer(5),
            y = cases,
            t = days,
            t_new=t_new)

fit1 <- stan(file="m2.stan", data = data1, iter=500)
```

- The model is fit again with new data
- Here along with the days and cases, we also give the data t_new to get the prediction.

```
y_pred1 <- extract(fit1)$y_pred
cases_pred <- as.integer(c(mean(y_pred1[,1]),mean(y_pred1[,2]),mean(y_pred1[,3]),mean(y_pred1
[,4]),mean(y_pred1[,5])))
print(cases_pred)
```

```
## [1] 10  9  8  8  7
```

Our model has predicted the number of cases for the next five days would be 10, 9, 8, 8, 7 respectively. Although the prediction may not be accurate, it can still give us a rough idea of the number of cases that can be found over the next five days.

Again we will create our model to predict the number of cases but this time we will use the g(t) function. The g(t) function is given as

```
      g(t) = theta1 exp(-theta2*theta3^t)
```

For our model we need the value of lambda_g(t) which is the derivative of g(t) w.r.t t.

Taking derivative we get lamda_g(t) as

```
      lamda_g(t)= theta1 * -theta2 * theta3^t * exp(-theta2 * theta3^t) * log(theta3)
```

We will use this function to build our model

```
set.seed(123)
library(rstan)
write("
data {
  int<lower=1> N;
  int y[N];
  vector[N] t;
}

parameters {
  real<lower=0> theta1;
  real<lower=0,upper=100> theta2;
  real<lower=0,upper=1> theta3;
}

model {
    target += normal_lpdf(theta1 | 1e3,1e5);
    target += normal_lpdf(theta2| 50, 100);
    for (n in 1:N){
    target += poisson_lpmf(y[n] | theta1 * (-theta2) * pow(theta3,t[n]) * exp(-theta2 * pow(t
heta3,t[n])) * log(theta3));
    }
}

generated quantities {
  vector[N] y_pred;
  real log_lik[N];
  for (n in 1:N){
    y_pred[n] = poisson_rng( theta1 * (-theta2) * pow(theta3,t[n]) * exp(-theta2 * pow(theta
3,t[n])) * log(theta3));
    log_lik[n] = poisson_lpmf(y[n] | theta1 * (-theta2) * pow(theta3,t[n]) * exp(-theta2 * po
w(theta3,t[n])) * log(theta3));
}
}"
,"m3.stan")


fit2<- stan(file="m3.stan", data = data, iter=500)
```

Our model is trained for 500 iterations using the same data as for part 1.

```
print(fit2, pars=c("theta1", "theta2", "theta3","y_pred[1]"))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=500; warmup=250; thin=1;
## post-warmup draws per chain=250, total post-warmup draws=1000.
##
##                mean se_mean     sd     2.5%      25%      50%      75%    97.5%
## theta1    25558.68    6.45 161.38 25252.69 25448.54 25556.77 25666.33 25869.51
## theta2       14.19    0.01   0.19    13.80    14.08    14.20    14.32    14.54
## theta3        0.94    0.00   0.00     0.93     0.94     0.94     0.94     0.94
## y_pred[1]     0.03    0.01   0.17     0.00     0.00     0.00     0.00     1.00
##           n_eff Rhat
## theta1      626 1.00
## theta2      375 1.01
## theta3      387 1.00
## y_pred[1]   797 1.00
##
## Samples were drawn using NUTS(diag_e) at Mon May 29 22:04:27 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

-The model summary can be seen using the print function. We get the mean, sd and the quantiles for the parameters theta1, theta2, theta3 as well as the predictions.We have shown the data for the prediction of first day. - The noticeable difference is the summary of y_pred[1] of both the models. In this model we get that the lower and upper levels for y_pred[1] is 0 to 1 with a mean of 0.03 and sd of 0.17. - The n_eff and Rhat values are acceptable for this model as well.

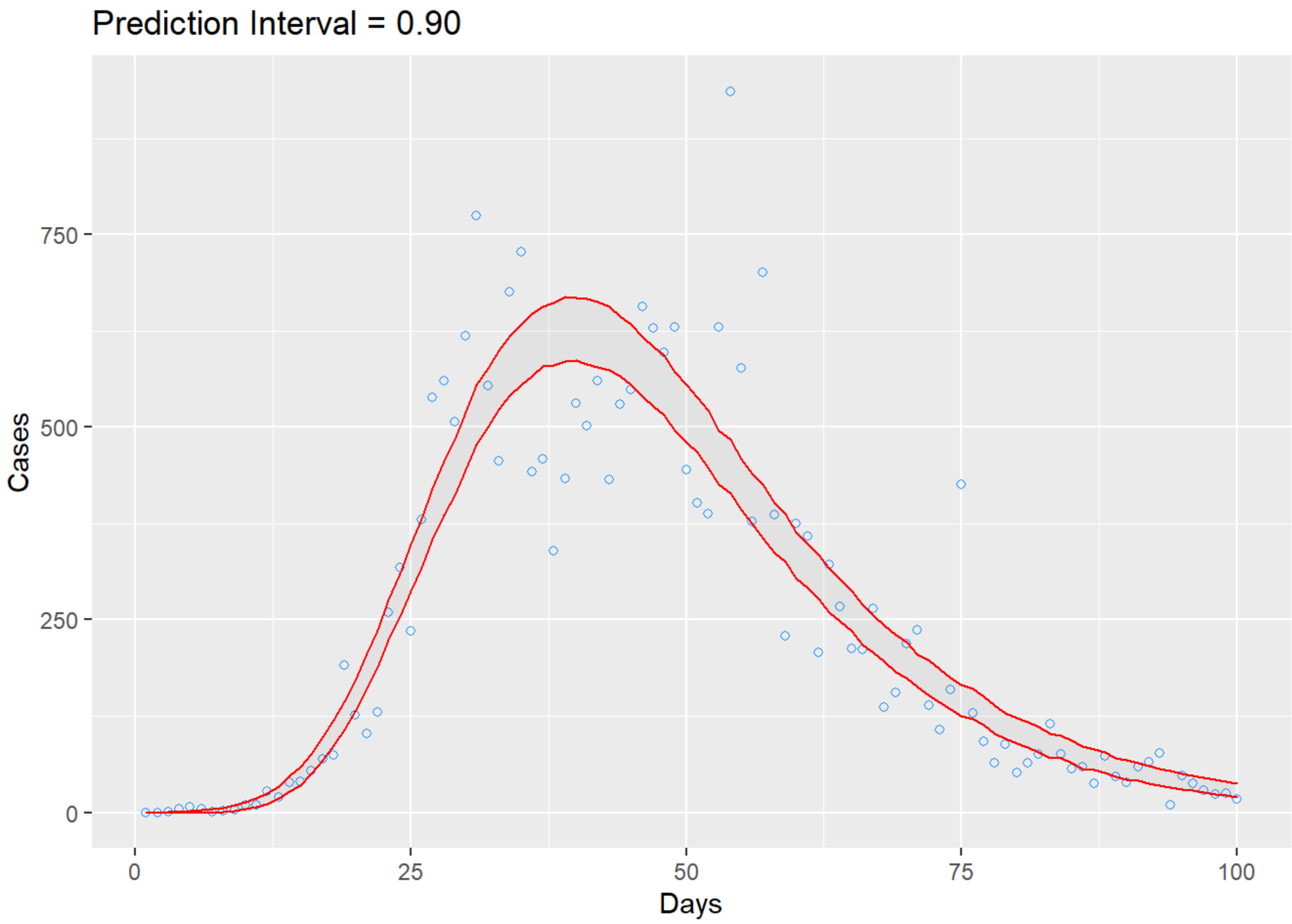
```
y_pred2 <- as.matrix(fit2, pars=c("y_pred"))
y_phdi1 = HDInterval::hdi(y_pred2, credMass=0.90)
pi_l1 = y_phdi1[1,]
pi_u1 = y_phdi1[2,]
print(y_phdi1)
```

```
##       parameters
##        y_pred[1] y_pred[2] y_pred[3] y_pred[4] y_pred[5] y_pred[6] y_pred[7]
##   lower         0         0         0         0         0         0         0
##   upper         0         0         1         1         2         3         4
##       parameters
##        y_pred[8] y_pred[9] y_pred[10] y_pred[11] y_pred[12] y_pred[13]
##   lower         1         2          4          7         11         18
##   upper         6         9         13         18         25         34
##       parameters
##        y_pred[14] y_pred[15] y_pred[16] y_pred[17] y_pred[18] y_pred[19]
##   lower         27         35         50         68         87        107
##   upper         47         59         76         98        120        144
##       parameters
##        y_pred[20] y_pred[21] y_pred[22] y_pred[23] y_pred[24] y_pred[25]
##   lower        131        161        189        224        254        287
##   upper        171        206        237        276        310        348
##       parameters
##        y_pred[26] y_pred[27] y_pred[28] y_pred[29] y_pred[30] y_pred[31]
##   lower        318        355        386        412        445        477
##   upper        381        420        456        484        519        555
##       parameters
##        y_pred[32] y_pred[33] y_pred[34] y_pred[35] y_pred[36] y_pred[37]
##   lower        500        523        541        555        566        579
##   upper        576        600        619        634        648        657
##       parameters
##        y_pred[38] y_pred[39] y_pred[40] y_pred[41] y_pred[42] y_pred[43]
##   lower        580        585        587        581        578        574
##   upper        662        669        668        667        663        656
##       parameters
##        y_pred[44] y_pred[45] y_pred[46] y_pred[47] y_pred[48] y_pred[49]
##   lower        566        555        540        527        516        496
##   upper        644        633        617        605        593        571
##       parameters
##        y_pred[50] y_pred[51] y_pred[52] y_pred[53] y_pred[54] y_pred[55]
##   lower        480        468        447        426        414        392
##   upper        555        539        522        496        484        459
##       parameters
##        y_pred[56] y_pred[57] y_pred[58] y_pred[59] y_pred[60] y_pred[61]
##   lower        375        356        337        325        304        291
##   upper        439        426        402        387        364        348
##       parameters
##        y_pred[62] y_pred[63] y_pred[64] y_pred[65] y_pred[66] y_pred[67]
##   lower        277        260        248        235        218        208
##   upper        334        316        303        287        270        256
##       parameters
##        y_pred[68] y_pred[69] y_pred[70] y_pred[71] y_pred[72] y_pred[73]
##   lower        196        182        175        163        151        143
##   upper        243        230        221        205        197        186
##       parameters
##        y_pred[74] y_pred[75] y_pred[76] y_pred[77] y_pred[78] y_pred[79]
##   lower        134        125        121        113        102         96
##   upper        175        165        160        150        139        129
##       parameters
##        y_pred[80] y_pred[81] y_pred[82] y_pred[83] y_pred[84] y_pred[85]
##   lower         89         85         78         70         70         64
```

```
##   upper        123        118        111        102         99         93
##        parameters
##         y_pred[86] y_pred[87] y_pred[88] y_pred[89] y_pred[90] y_pred[91]
##   lower         57         55         52         46         43         41
##   upper         86         82         78         71         68         64
##        parameters
##         y_pred[92] y_pred[93] y_pred[94] y_pred[95] y_pred[96] y_pred[97]
##   lower         38         35         33         30         28         26
##   upper         60         57         54         50         48         45
##        parameters
##         y_pred[98] y_pred[99] y_pred[100]
##   lower         24         22          20
##   upper         42         40          38
## attr(,"credMass")
## [1] 0.9
```

The 90% credible intervals shows that the model predicts the cases to fall between this range. We can clearly see the difference between this model and the model in part1. Plotting the credible intervals with the original data will give us an idea of the accuracy of our model.

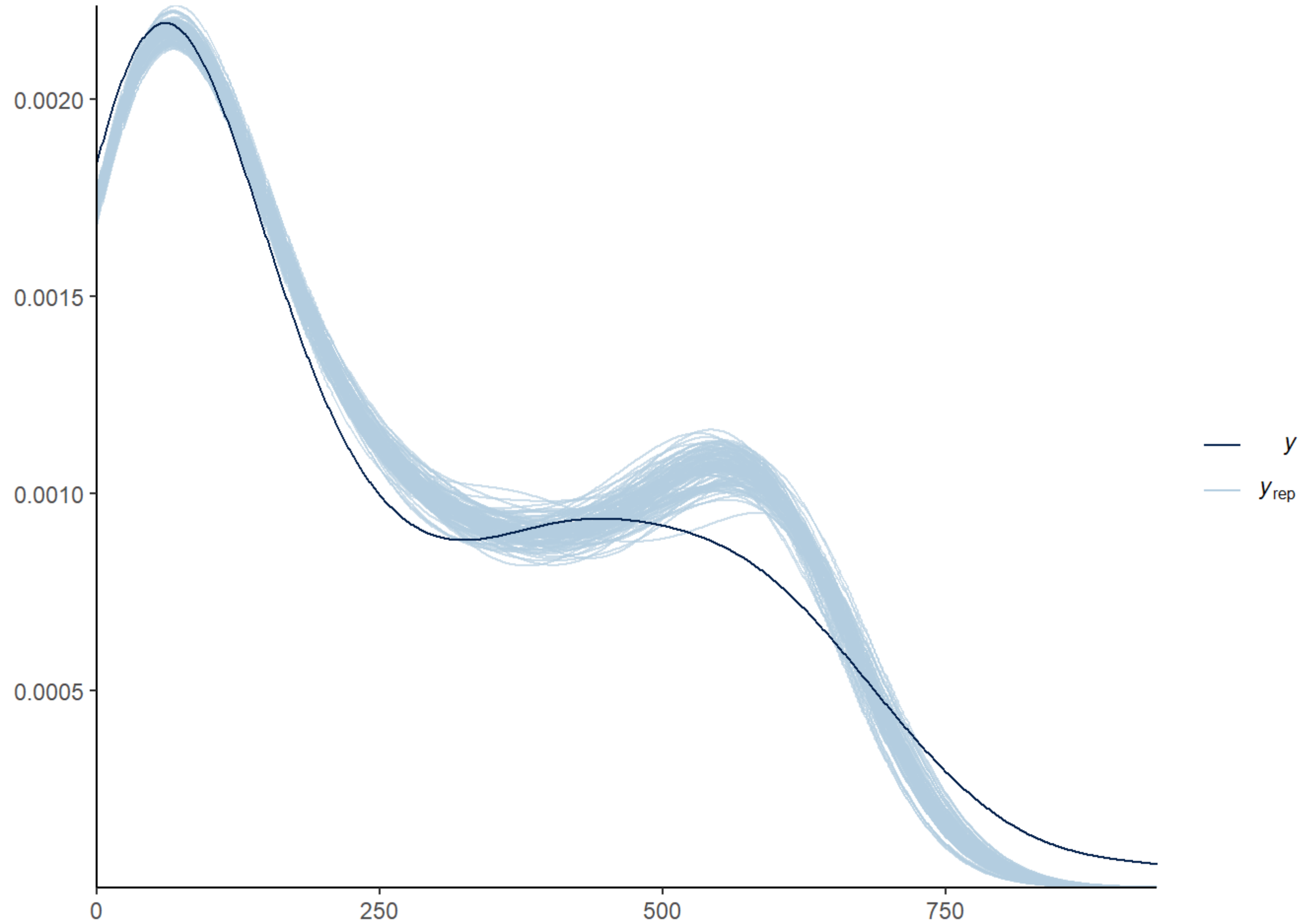
```
p3 <- p +
  geom_point(data = d1,
      aes(t, y), shape = 1, color = 'dodgerblue') +
  ggtitle("Prediction Interval = 0.90")+
  geom_ribbon(data = d1,
      mapping = aes(t, ymin=pi_l1, ymax=pi_u1), alpha = .05,color = 'red')+xlab("Days")+ ylab
("Cases")

p3
```



From the plot we can again see that the model doesn’t do a great job in predicting the values. Although the values are predicted accurately at the start and to the end but there is a lot of inconsistency in the centre.

```
library(bayesplot)
ppc_dens_overlay(d1$y, y_pred2[1:100,])+ theme_classic()
```

```
## Warning: Ignoring unknown parameters: linewidth
## Ignoring unknown parameters: linewidth
```



Through bayesplot, we can see the density curve of the original data as compared to our prediction.

```
set.seed(123)
library(rstan)
write("
data {
  int<lower=1> N;
  int<lower=1> N1;
  int y[N];
  vector[N] t;
  vector[N1] t_new;
}

parameters {
  real<lower=0> theta1;
  real<lower=0,upper=100> theta2;
  real<lower=0,upper=1> theta3;
}

model {
    target += normal_lpdf(theta1 | 1e3,1e5);
    target += normal_lpdf(theta2| 50, 100);
    for (n in 1:N){
    target += poisson_lpmf(y[n] | theta1 * (-theta2) * pow(theta3,t[n]) * exp(-theta2 * pow(t
heta3,t[n])) * log(theta3));
    }
}

generated quantities {
  vector[N1] y_pred;
  for (n in 1:N1){
    y_pred[n] = poisson_rng( theta1 * (-theta2) * pow(theta3,t_new[n]) * exp(-theta2 * pow(th
eta3,t_new[n])) * log(theta3) );
  }
}"
,"m4.stan")

fit3 <- stan(file="m4.stan", data = data1, iter=500)
```

```
y_pred3 <- extract(fit3)$y_pred
cases_pred1 <- as.integer(c(mean(y_pred3[,1]),mean(y_pred3[,2]),mean(y_pred3[,3]),mean(y_pred
3[,4]),mean(y_pred3[,5])))
print(cases_pred1)
```

```
## [1] 27 25 24 22 21
```

We see that the prediction from the model for the days 101-105 is 27, 25, 24, 22, 21 respectively. As the model gives 500 iterations, we use the mean of 500 values to get the prediction.

Model Comparison using Loo package

```
library(loo)
```

```
## This is loo version 2.5.1
```

```
## - Online documentation and vignettes at mc-stan.org/loo
```

```
## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the
'cores' argument or set options(mc.cores = NUM_CORES) for an entire session.
```

```
## - Windows 10 users: loo may be very slow if 'mc.cores' is set in your .Rprofile file (see
https://github.com/stan-dev/loo/issues/94).
```

```
##
## Attaching package: 'loo'
```

```
## The following object is masked from 'package:rstan':
##
##     loo
```

```
loo1 <- loo(fit)
loo2 <- loo(fit2)
loo_compare(loo1,loo2)
```

```
##        elpd_diff se_diff
## model2    0.0       0.0
## model1 -922.3     248.0
```

From the values of loo_compare we see that the accuracy of model 2(Gompertz) is better than that of the logistic function. So model 2 is preferred over model 1. The standard error difference too is pretty high, indicating that there is a big difference between the two. We will use the WAIC function to get more information.

Model Comparison using WAIC function

```
log_lik1 <- loo::extract_log_lik(fit)
waic1 <- loo::waic(log_lik1)
log_lik2 <- loo::extract_log_lik(fit2)
waic2 <- loo::waic(log_lik2)
loo_compare(waic(log_lik1), waic(log_lik2))
```

```
##        elpd_diff se_diff
## model2    0.0       0.0
## model1 -926.3     249.4
```

The WAIC function also suggests that model 2 is better than model 1.The standard in this case is also high.

Although model 2 is better than model 1, both the models are notable to predict the data as accurately as desired. We can improve the model fit by the following techniques

Methods to improve model fit

1. We can adjust the model parameters (theta1, theta2, theta3) to improve the fit.
2. We can add more data to improve the model. More data helps the model to learn the trend better so as to better estimate the values.
3. We can try different models and check whether a different model can be a better fit.
4. Regularization techniques can help to prevent overfitting of the data by adding a penalty term to the model, which can improve the fit.

Conclusion

Overall we see that the Gompretz model is a better model for our data but it can be eventually be improved.