# STAT 663
# Statistical Graphics and Data Exploration I

Week 13

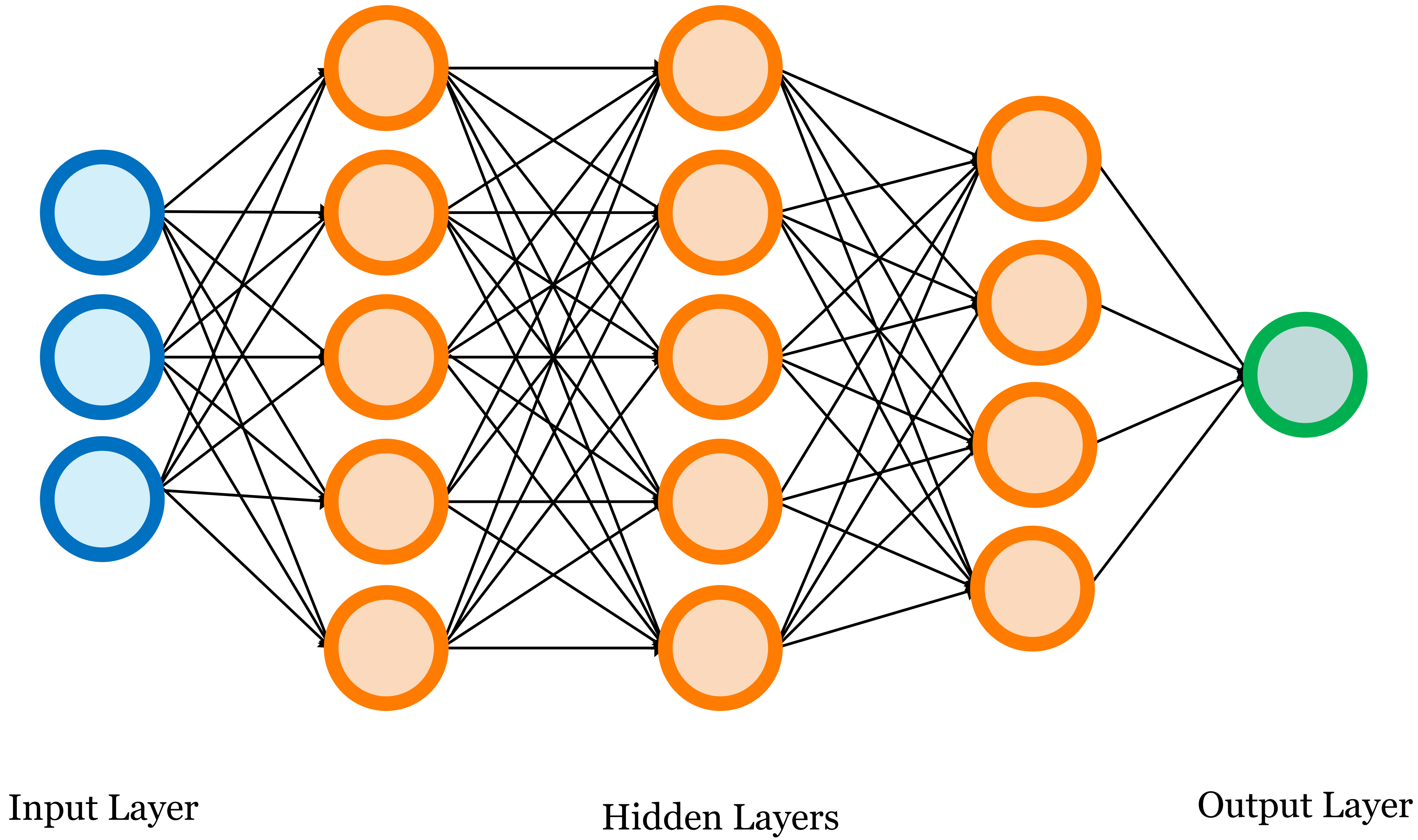
Neural Network

Lecture 2

# Lesson Objectives

- Neural Network Auto-Regressive (NNAR) models
- The nnetar() function in R
- Application of the NNAR model to COVID-19 study

# Neural Networks



# Neural Network Autoregression (NNAR) Models

- Lagged values of the time series can be used as inputs to a neural network.
- NNAR$(p, k)$: $p$ lagged inputs and $k$ nodes in the single hidden layer.
- NNAR$(p, 0)$ model is equivalent to an ARIMA$(p, 0, 0)$ model but without stationarity restrictions.
- Seasonal NNAR$(p, P, k)_m$: inputs $(Y_{t-1}, Y_{t-2}, ..., Y_{t-p}, Y_{t-m}, Y_{t-2m}, Y_{t-Pm}$ and $k$ neurons in the hidden layer.
- NNAR$(p, P, 0)_m$ model is equivalent to an ARIMA$(p, 0, 0)(P, 0, 0)_m$ model but without stationarity restrictions.

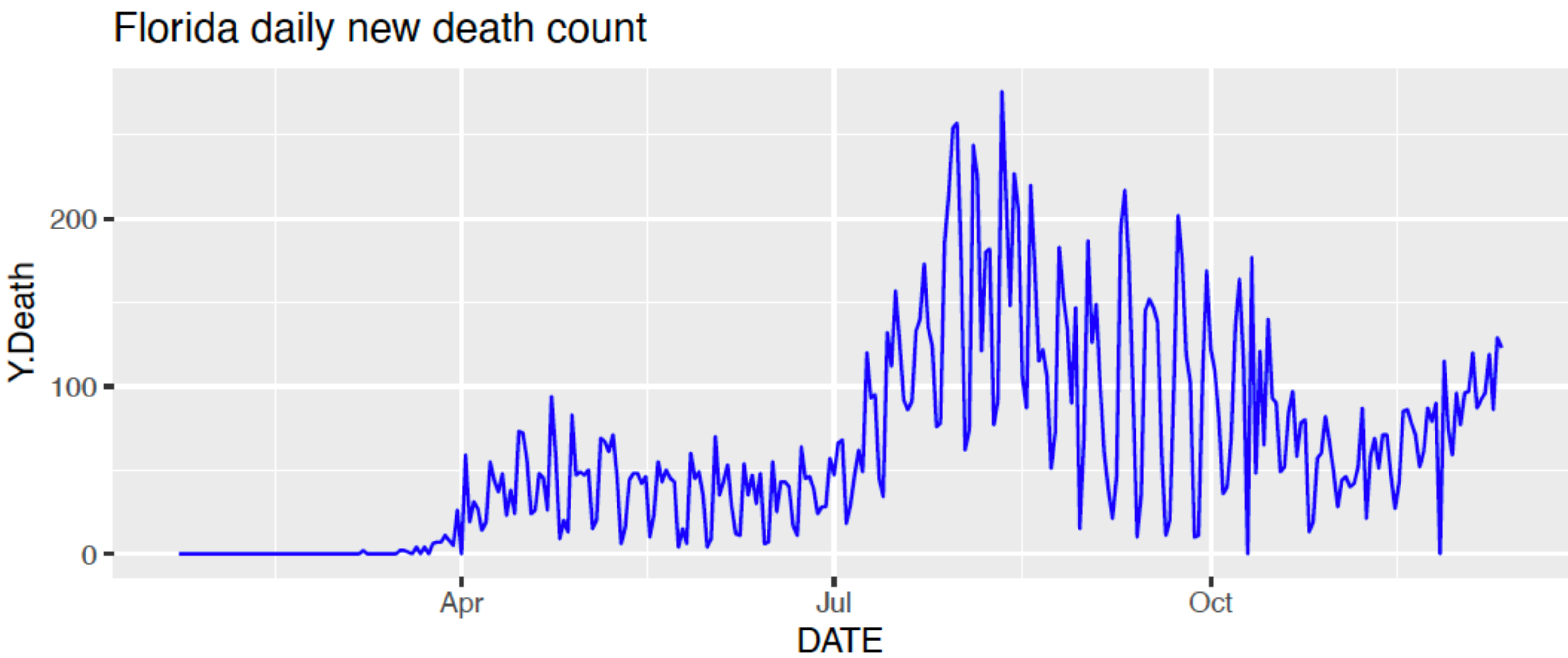
# NNAR models in R

- The `nnetar()` function in R can be used to fit an NNAR$(p, P, k)_m$ model.
- If $p$ and $P$ are not specified, they are automatically selected.
- For non-seasonal time series, default $p =$ optimal number of lags (according to the AIC) for a linear AR$(p)$ model.
- For seasonal time series, defaults are $P = 1$ and $p$ is chosen from the optimal linear model fitted to the seasonally adjusted data.
- Default $k = (p + P + 1)/2$ (rounded to the nearest integer).

# NNAR Models: Rolling Forecast

- The network is applied iteratively.
- For forecasting one step ahead, we simply use the available historical inputs.
- For forecasting two steps ahead, we use the one-step forecast as an input, along with the historical data.
- This process proceeds until we have computed all the required forecasts.

# Time Series Plot for Florida



# The nnetar function

We can also specify the options in the nnetar function.

- The repeats option of nnetar shows **the number of networks** (default = 20) to fit with different random starting weights. These networks are then averaged when producing forecasts.

- The size option provides number of nodes in the hidden layer. Default is half of the number of input nodes (including external regressors, if given) plus 1.

- We can consider a Box-Cox transformation for the data.
  - If lambda = auto, then a transformation is automatically selected using BoxCox.alpha.
  - The transformation is ignored if lambda = NULL.
  - Otherwise, data transformed before model is estimated.

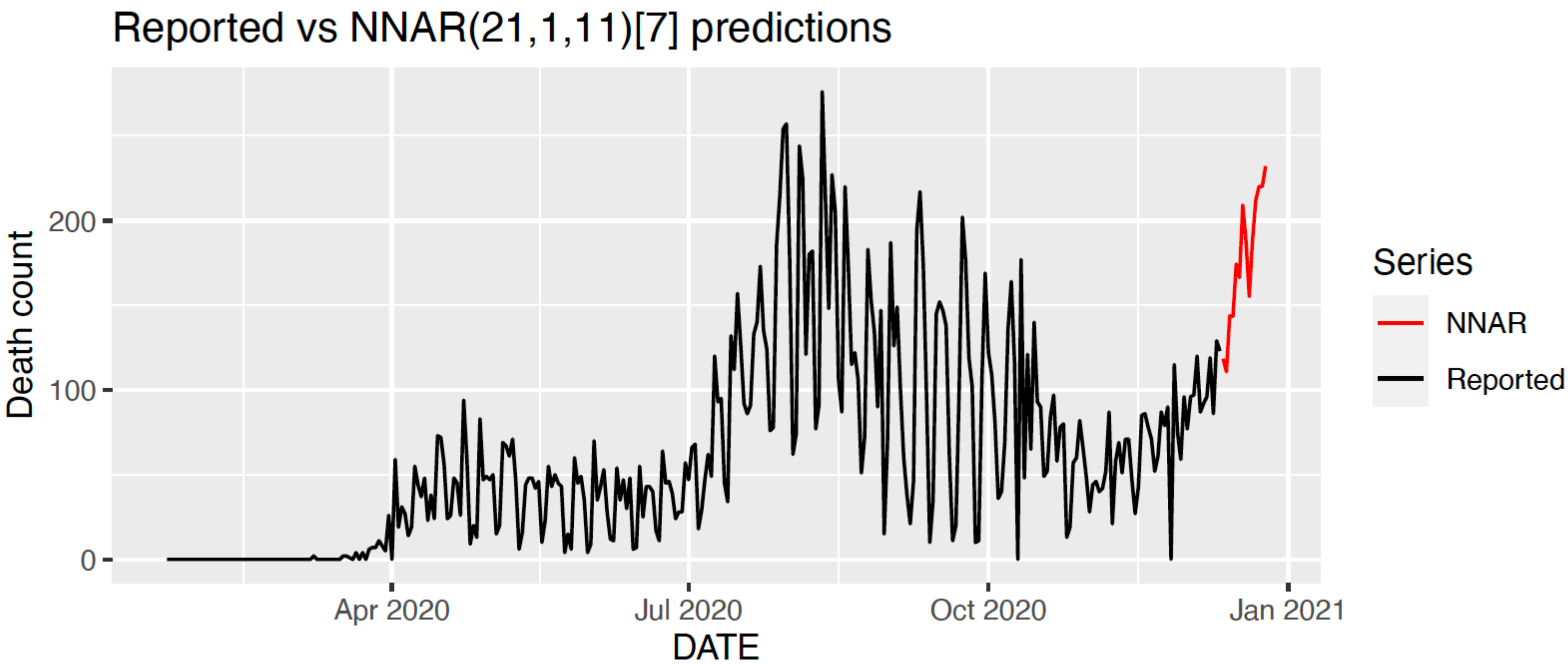
# A Time Series Plot for Florida

```
library(slid)
library(dplyr)
data(state.ts)
Florida.ts <- state.ts %>%
dplyr::filter(State == "Florida")
y <- ts(Florida.ts$Y.Death, start = c(2020,1),
frequency = 7)
```

```
library(forecast)
set.seed(2020)
fit <- nnetar(y)

fit
## Series: y
## Model: NNAR(21,1,11)[7]
## Call: nnetar(y = y)
##
## Average of 20 networks, each of which is
## a 21-11-1 network with 254 weights
## options were - linear output units
##
## sigma^2 estimated as 6.598
```

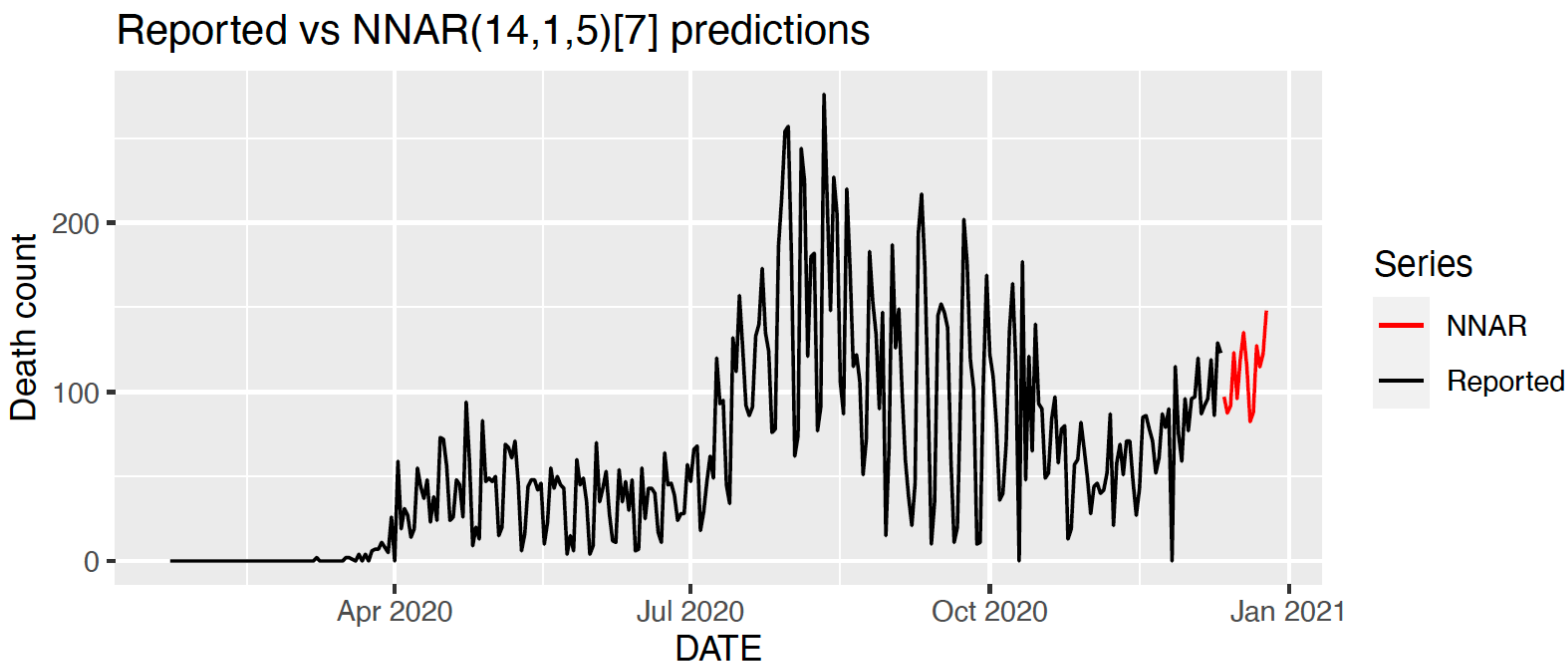
```
nnar.fc.report <- as.data.frame(forecast(fit, h = 14)) %>%
mutate(DATE = as.Date("2020-12-11") + 1:14)
names(nnar.fc.report) <- c("Y.Death", "DATE")

# plot of reported vs NNAR predictions
ggplot() +
geom_line(aes(x = DATE, y = Y.Death, color = "Reported"),
data = Florida.ts) +
geom_line(aes(x = DATE, y = Y.Death, color = "NNAR"),
data = nnar.fc.report) +
scale_color_manual(
values = c(Reported = "black", NNAR = "red")) +
labs(y = "Death count",
title = "Reported vs NNAR(21,1,11)[7] predictions") +
guides(color = guide_legend(title = "Series"))
```

# NNAR Prediction



# Log Transformation

```
fit <- nnetar(y + 1, lambda = 0, size = 5)
```



# Prediction Intervals

- Define the NNAR$(p, k)$ model as

$$Y_t = f(Y_{t-1}, \dots, Y_{t-p}) + \varepsilon_t,$$

  where $f$ is a neural network with $k$ hidden nodes in a single layer. The error series $\{\varepsilon_t\}$ is assumed to be homogeneous.

- We can simulate future sample paths of this model iteratively, by randomly generating a value for $\varepsilon_t$, either from a normal distribution, or by resampling from the historical values.
  — So if $\varepsilon^*_{T+1}$ is a random draw from the distribution of errors at time $T + 1$, then

$$Y^*_{T+1} = f(Y_T, \dots, Y_{T-p+1}) + \varepsilon^*_{T+1}$$

  is one possible draw from the forecast distribution for $Y_{T+1}$.

- We can then repeat the process to get

$$Y^*_{T+2} = f(Y^*_{T+1}, Y_T, \dots, Y_{T-p+1}) + \varepsilon^*_{T+2}$$

  and obtain a sample path $\{Y^*_{T+1}, Y^*_{T+2}, \dots, Y^*_{T+h}\}$.

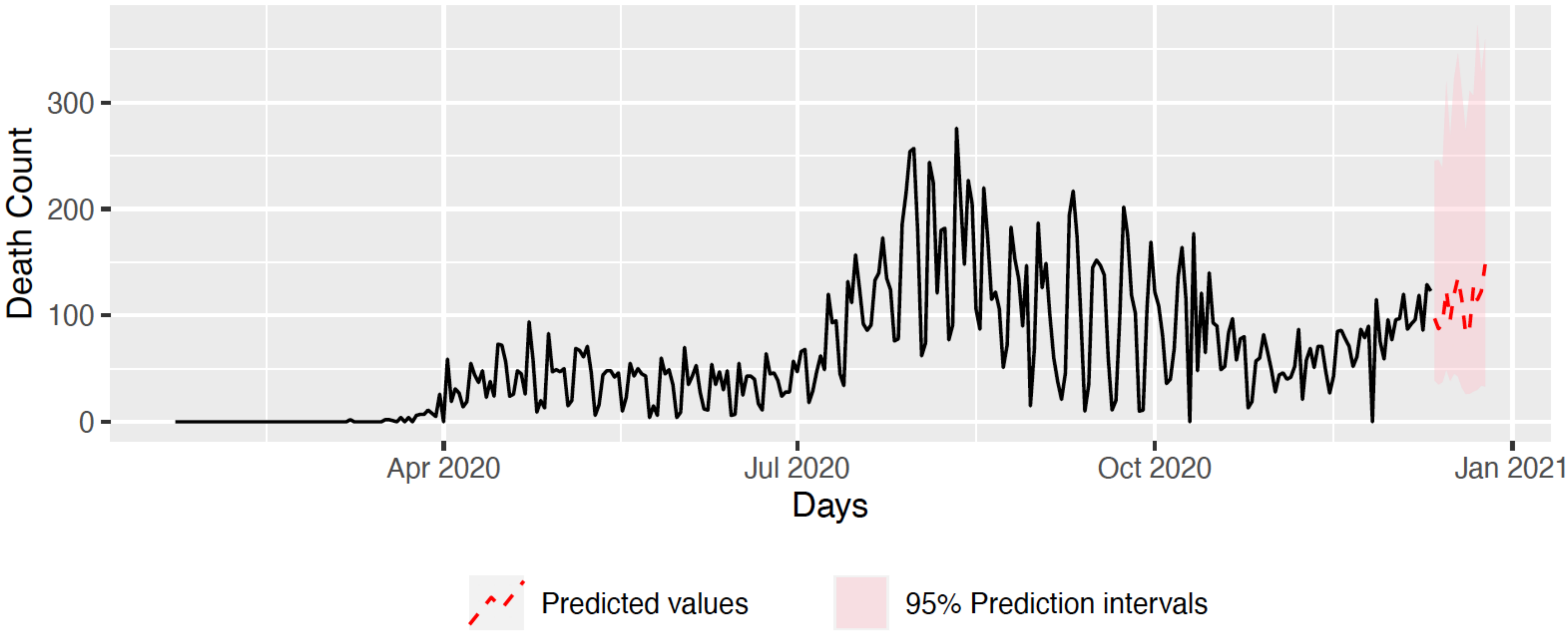
- If we generate many sample paths, we can get a good picture of the forecast distributions.

# Prediction Intervals

```
fcast <- forecast(fit, PI = TRUE, h = 14, npaths = 500)
```

```
#autoplot(fcast, xlab = "Days", ylab = "Daily new deaths")
fcast.report <- as.data.frame(fcast) %>%
mutate(DATE = as.Date("2020-12-11") + 1:14)
names(fcast.report) <- c("Y.Death", "Lo.80", "Hi.80", "Lo.95", "Hi.95", "DATE")
ggplot(Florida.ts, aes(DATE, Y.Death)) +
geom_line() +
labs(x = "Days", y = "Death Count") +
# Add prediction intervals
geom_ribbon(mapping = aes(x = DATE, y = Y.Death, ymin = Lo.95, ymax = Hi.95,
fill = '95% Prediction intervals’), data = fcast.report, alpha = 0.4) +
# Add line for predicted values
geom_line(mapping = aes(x = DATE, y = Y.Death, colour = 'Predicted values'),
linetype = "dashed", data = fcast.report, key_glyph = "timeseries") +
scale_colour_manual("", values = "red") +
scale_fill_manual("", values = "pink") +
theme(legend.position = "bottom")
```

# Prediction Intervals



# Two Statistical Cultures