Time Series Forecasting

R Tutorials for Applied Statistics

Note on required packages: The following code requires the packages `fpp2`, `quantmod`, and `scales`. The package `fpp2` contains many time series functions.
The `quantmod` package allows you to download and bring into memory financial and macroeconomic data from common sources. The package `scales` is used to change the scales in our plots. If you have not already done so, download, install, and load the library with the following code:

```
# These only needs to be executed once for your machine or RStudio Cloud project
install.packages("fpp2")
install.packages("quantmod")
install.packages("scales")


# These need to be executed every time you load R
library(fpp2)
library(quantmod)
library(scales)
```

---

1 Download Data

We will use R code to download latest available macroeconomic data from the Federal Reserve Economic Database (FRED). It is useful to browse https://fred.stlouisfed.org to identify variables you are interested in. At each variable's page, there is a code next to the name, which is used in the call below.

Below we use the `getSymbols()` function in the `quantmod` package to download and load into memory monthly unemployment rate that is not seasonally adjusted. The code for this variable in FRED is.

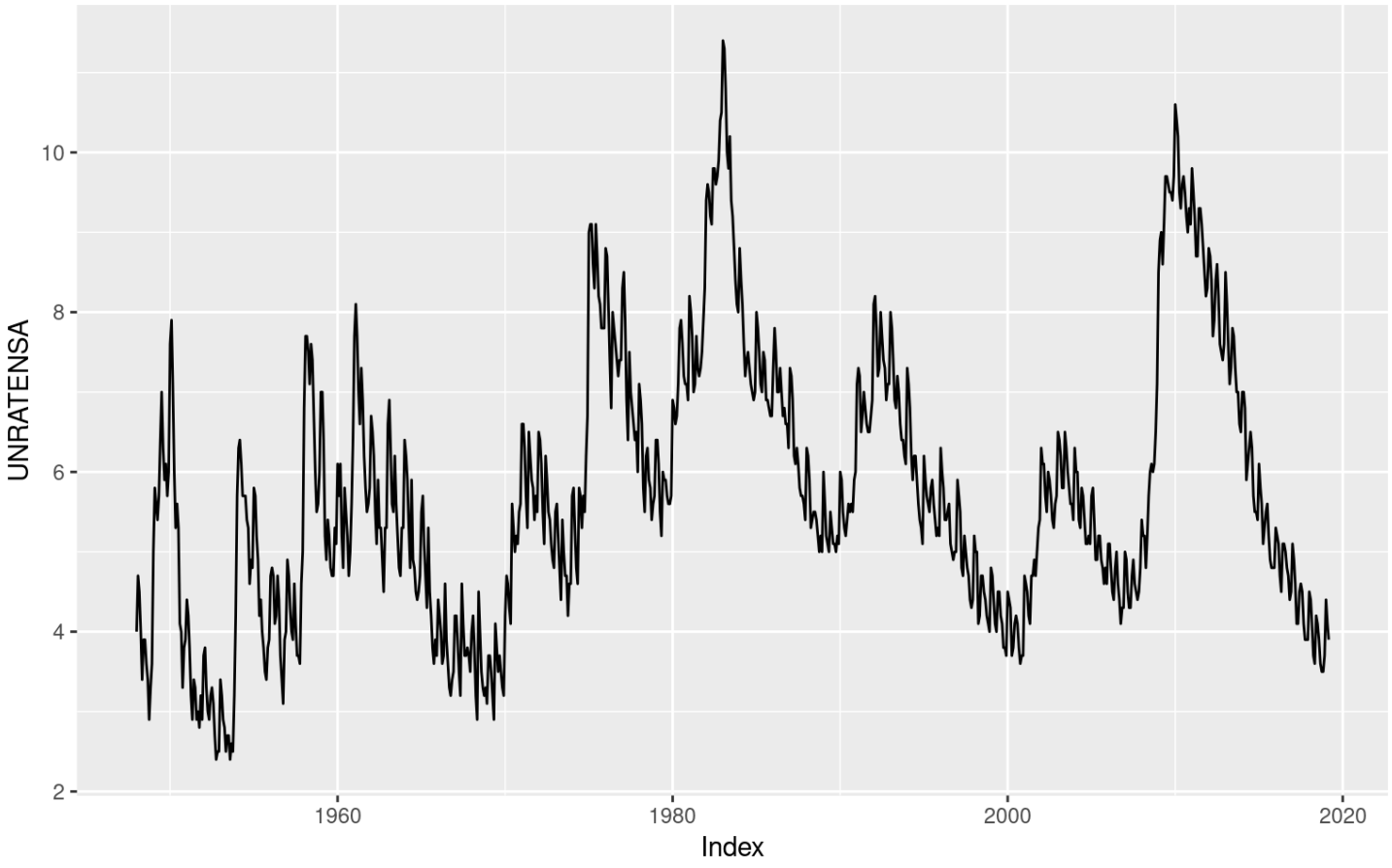
```
getSymbols("UNRATENSA", src="FRED")
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
## [1] "UNRATENSA"
```

This creates a time series object called `UNRATENSA` (short for unemployment rate - not seasonally adjusted). We can show a quick plot of our data with the following call to `autoplot()`

```
autoplot(UNRATENSA)
```



We can look at the first few observations with a call to `head()`.

```
head(UNRATENSA)
```

```
##            UNRATENSA
## 1948-01-01       4.0
## 1948-02-01       4.7
## 1948-03-01       4.5
## 1948-04-01       4.0
## 1948-05-01       3.4
## 1948-06-01       3.9
```

We can see this is monthly data. It is important to check if the frequency of the data is consistent with the monthly measurement. Since there are 12 months in a year, the frequency should equal 12.

```
frequency(UNRATENSA)
```

```
## [1] 1
```

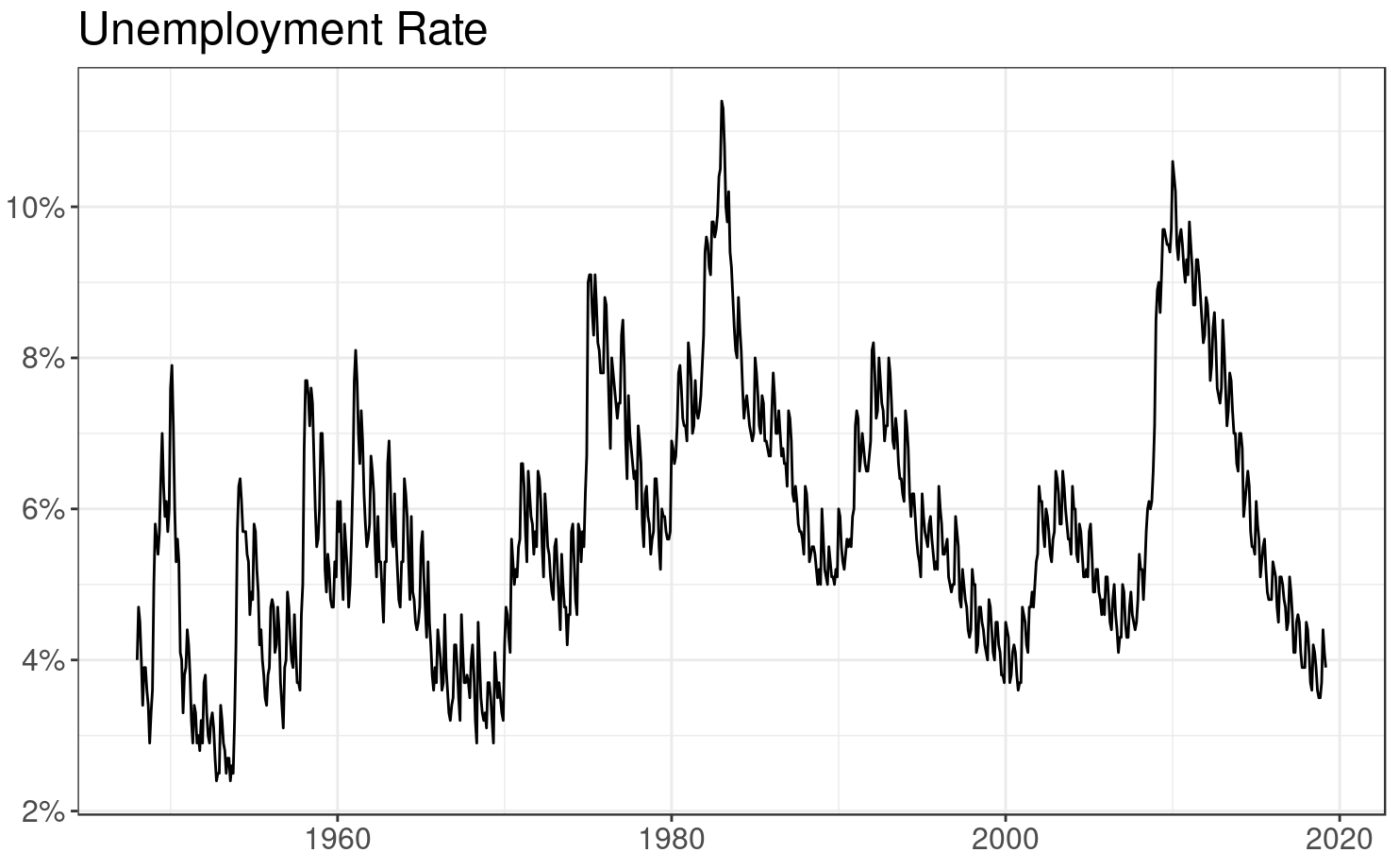
The frequency is set incorrectly to 1. We will need to create a new time series object with the same data, but with the appropriate frequency. When plotting it will be more also useful if the unemployment data is given as a decimal, as putting percentages on scales automatically multiplies the number by 100. So the data we give is our original data divided by 100.

In the code below, we create a new time series object called `unrate` with these adjustments.

```
unrate <- ts(UNRATENSA/100, frequency=12, start=c(1948,1))
```

Let's use our new time series object to create a more visually appealing plot:

```
autoplot(unrate) +
  theme_bw() +
  labs(title="Unemployment Rate", x="", y="") +
  scale_y_continuous(label=percent) +
  theme(text = element_text(size=15))
```



2 Naive Forecasts

The simplest forecast to give for any time series variable is the naive forecast. It simply uses the last observation in the sample, and guesses this will be the value for all future observations.

2.1 Random Walk Model

While it sounds stupid, the naive estimator is actually the most accurate estimator if the variable follows a random walk model:

$$x_t = x_{t-1} + \epsilon_t$$
xt=xt−1+ϵt

The random walk model simply says that the next observation in a time series is equal to the previous value plus a random step given by ϵ_tϵt.

Another reason the naive forecast is taken so seriously, is that despite its simplicity, many times it outperforms other complicated forecasts in terms of accuracy. As such, it is a useful benchmark.

2.2 Computing and Plotting Naive Forecast

We compute the naive forecast below with a call to `naive()` and save the result in a forecast object that we called `fNaive`. For all the forecasts in this tutorial, we will use a forecast horizon equal to 24 months.

```
horizon = 24
fNaive <- naive(unrate, h=horizon)
```

We can view the forecasts by calling our saved object, `fNaive`.

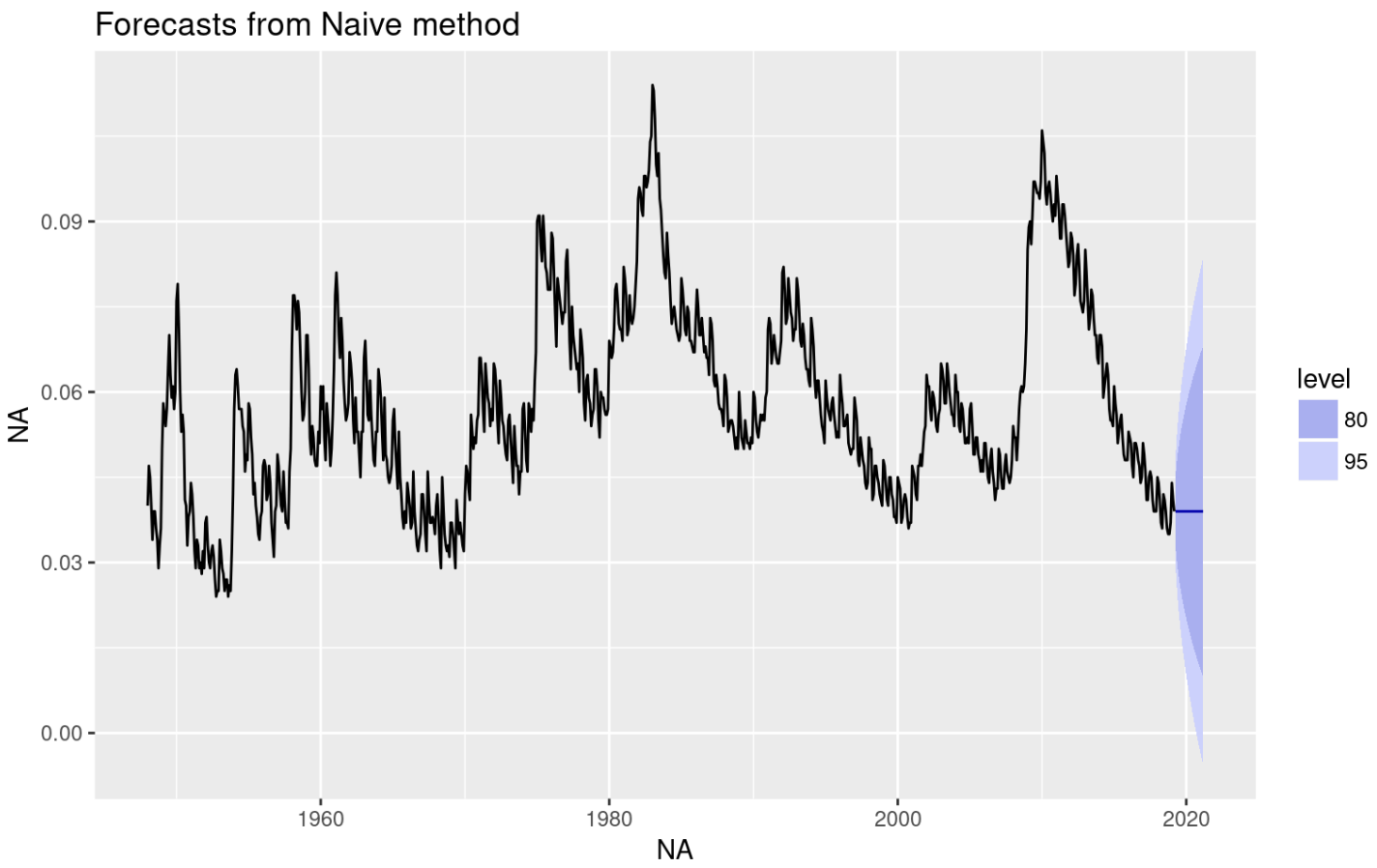
```
fNaive
```

```
##          Point Forecast       Lo 80      Hi 80         Lo 95      Hi 95
## Apr 2019          0.039 0.033050897 0.04494910  0.0299016326 0.04809837
## May 2019          0.039 0.030586698 0.04741330  0.0261329655 0.05186703
## Jun 2019          0.039 0.028695852 0.04930415  0.0232411654 0.05475883
## Jul 2019          0.039 0.027101795 0.05089821  0.0208032653 0.05719673
## Aug 2019          0.039 0.025697402 0.05230260  0.0186554321 0.05934457
## Sep 2019          0.039 0.024427734 0.05357227  0.0167136424 0.06128636
## Oct 2019          0.039 0.023260154 0.05473985  0.0149279826 0.06307202
## Nov 2019          0.039 0.022173397 0.05582660  0.0132659309 0.06473407
## Dec 2019          0.039 0.021152692 0.05684731  0.0117048979 0.06629510
## Jan 2020          0.039 0.020187286 0.05781271  0.0102284361 0.06777156
## Feb 2020          0.039 0.019269059 0.05873094  0.0088241292 0.06917587
## Mar 2020          0.039 0.018391704 0.05960830  0.0074823309 0.07051767
## Apr 2020          0.039 0.017550205 0.06044979  0.0061953699 0.07180463
## May 2020          0.039 0.016740496 0.06125950  0.0049570265 0.07304297
## Jun 2020          0.039 0.015959224 0.06204078  0.0037621747 0.07423783
## Jul 2020          0.039 0.015203589 0.06279641  0.0026065305 0.07539347
## Aug 2020          0.039 0.014471221 0.06352878  0.0014864703 0.07651353
## Sep 2020          0.039 0.013760095 0.06423990  0.0003988964 0.07760110
## Oct 2020          0.039 0.013068463 0.06493154 -0.0006588639 0.07865886
## Nov 2020          0.039 0.012394804 0.06560520 -0.0016891359 0.07968914
## Dec 2020          0.039 0.011737787 0.06626221 -0.0026939572 0.08069396
## Jan 2021          0.039 0.011096235 0.06690376 -0.0036751257 0.08167513
## Feb 2021          0.039 0.010469106 0.06753089 -0.0046342371 0.08263424
## Mar 2021          0.039 0.009855468 0.06814453 -0.0055727151 0.08357272
```

Our naive forecast simply guesses that all future values of the unemployment rate will be 3.9%, as the most recent observation is 3.9%.

We can make a quick plot of our data and our forecast with `autoplot()`.

```
autoplot(fNaive)
```

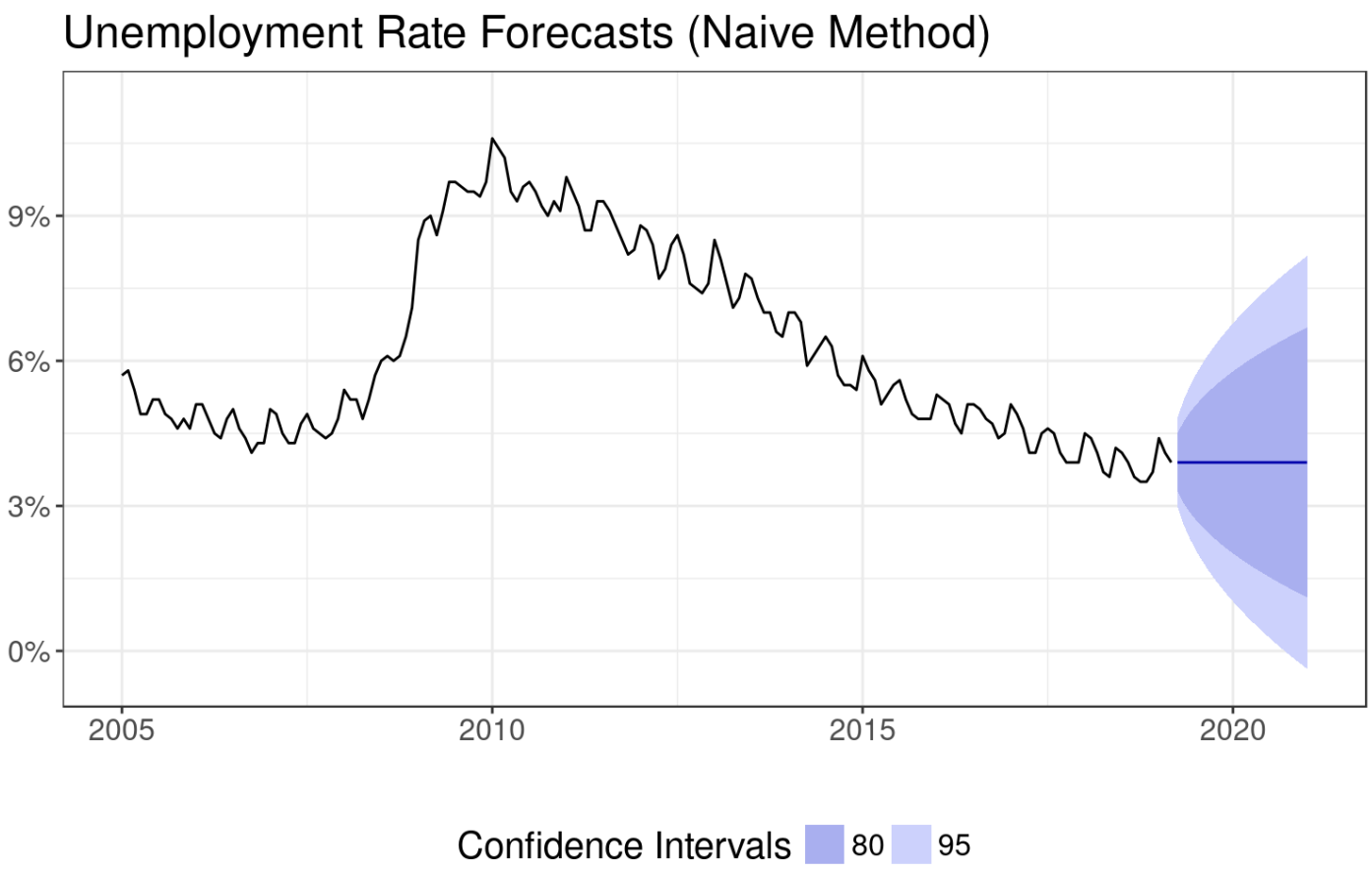


Our plot also includes an 80% and 95% confidence intervals representing the margin of error in our forecasts.

Let’s make a more visually appealing plot, looking at just data since 2005 and the two years of forecasts (through 2021).

```
autoplot(fNaive) +
  theme_bw() +
  scale_y_continuous(labels=percent) +
  scale_x_continuous(limits=c(2005,2021)) +
  theme(text = element_text(size=15)) +
  labs(title="Unemployment Rate Forecasts (Naive Method)",
       x="", y="",
       fill="Confidence Intervals") +
  theme(legend.position = "bottom")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```



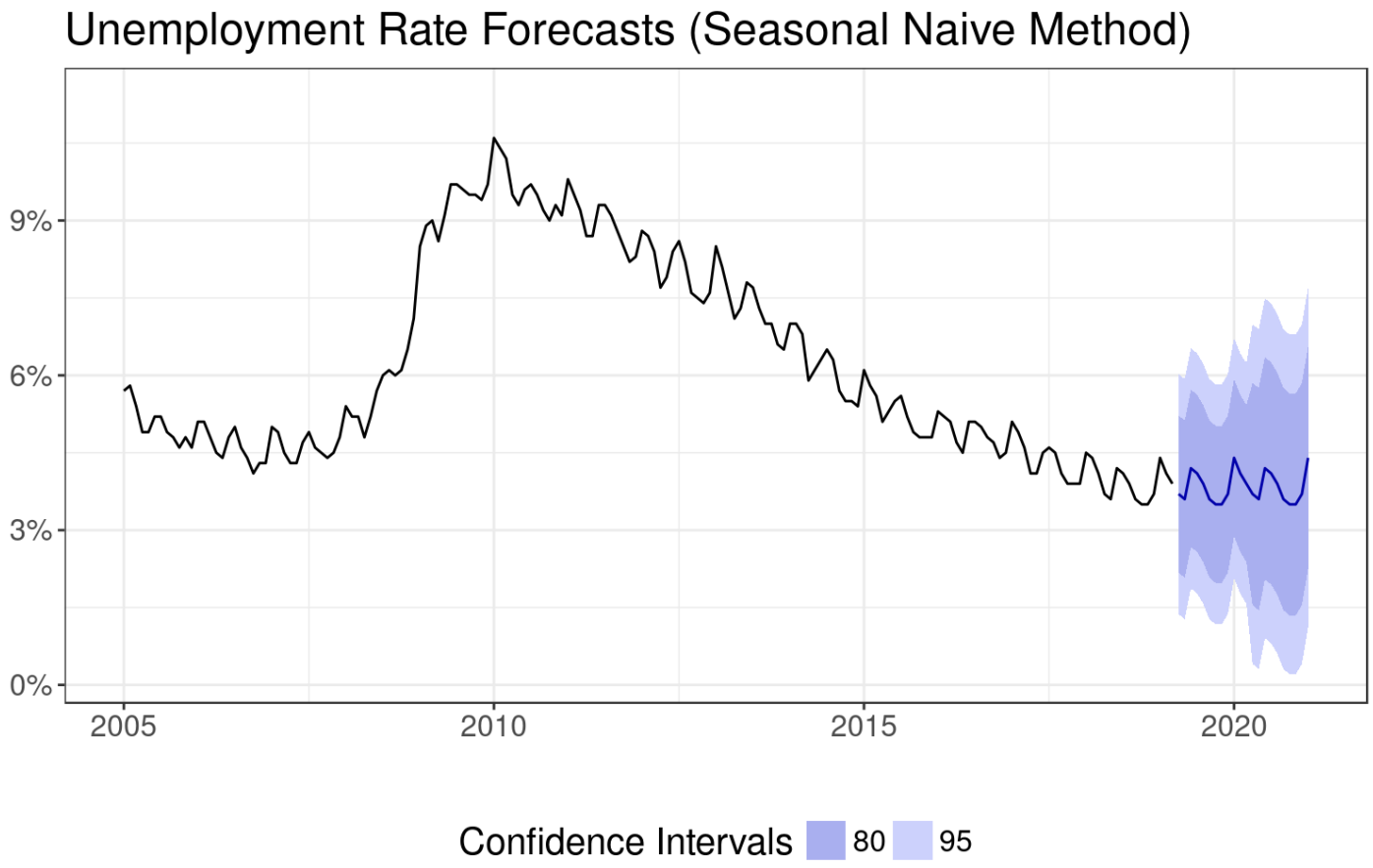
2.3 Seasonal Naive Forecast

Our time series contains predictable seasonality. A naive forecast that takes this into account is instead of using the most recent observation to predict the indefinite future, we use the most recent observation from the same month. For example, the forecast for January 2020 is equal to the value in January 2019. The forecast for February 2020 is equal to the value February 2019.

Below, we compute the seasonal naive forecast with the function `snaive()` and show its plot.

```
fSeasonNaive <- snaive(unrate, h=horizon)
autoplot(fSeasonNaive) +
  theme_bw() +
  scale_y_continuous(labels=percent) +
  scale_x_continuous(limits=c(2005,2021)) +
  theme(text = element_text(size=15)) +
  labs(title="Unemployment Rate Forecasts (Seasonal Naive Method)",
       x="", y="",
       fill="Confidence Intervals") +
  theme(legend.position = "bottom")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```



3 Autoregression Models

An autoregression model is a linear regression forecasting model where the time series variable is regressed against one or more lags of the same variable. An autoregressive model of order $p \geq 1$ $p \geq 1$ (denoted by AR(p)) is given by,

$$x_t = \rho_1 x_{t-1} + \rho_2 x_{t-2} + \ldots + \rho_p x_{t-p} + \epsilon_t$$
$$xt = \rho 1 xt-1 + \rho 2 xt-2 + \ldots + \rho p xt-p + \epsilon t$$

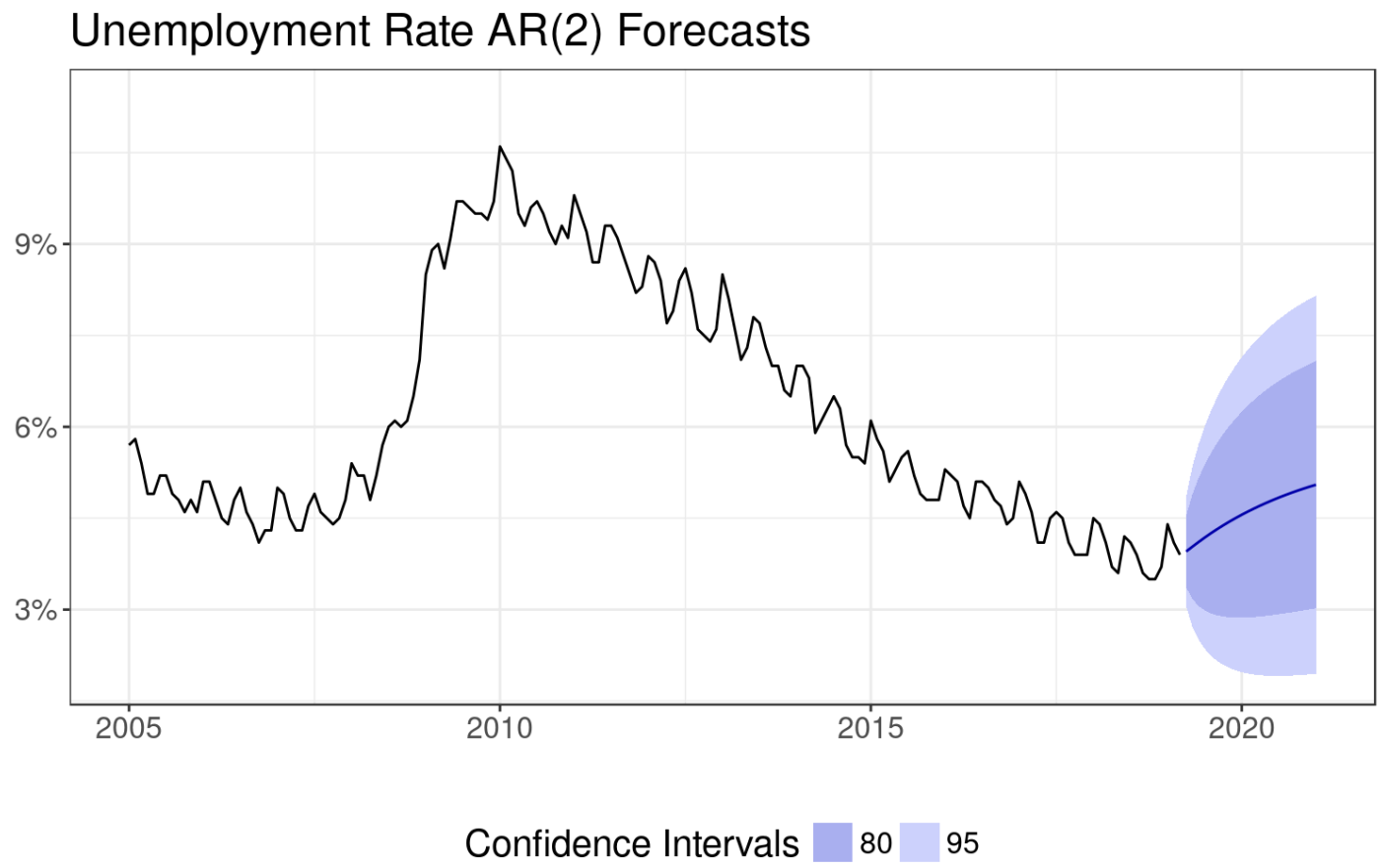
For example, we can estimate an AR(2) model on our unemployment rate data. In the code below, we estimate our model, create our forecast, and plot the forecast. We use the `Arima()` function, which can be used to estimate a large class of models including the autoregression model.

```
p = 2 # Autoregressive order
armodel <- Arima(unrate, order=c(p,0,0))

far <- forecast(armodel, h=horizon)

autoplot(far) +
  theme_bw() +
  scale_y_continuous(labels=percent) +
  scale_x_continuous(limits=c(2005,2021)) +
  theme(text = element_text(size=15)) +
  labs(title="Unemployment Rate AR(2) Forecasts",
       x="", y="",
       fill="Confidence Intervals") +
  theme(legend.position = "bottom")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```



The above forecasts are missing the seasonality component of the data. Below, we estimate a model that includes the last two lags *and* the last two observations from the same month. This allows us to use information from the most recent months *and* and

information from the same month from the most recent years. The model we are estimating is the following:

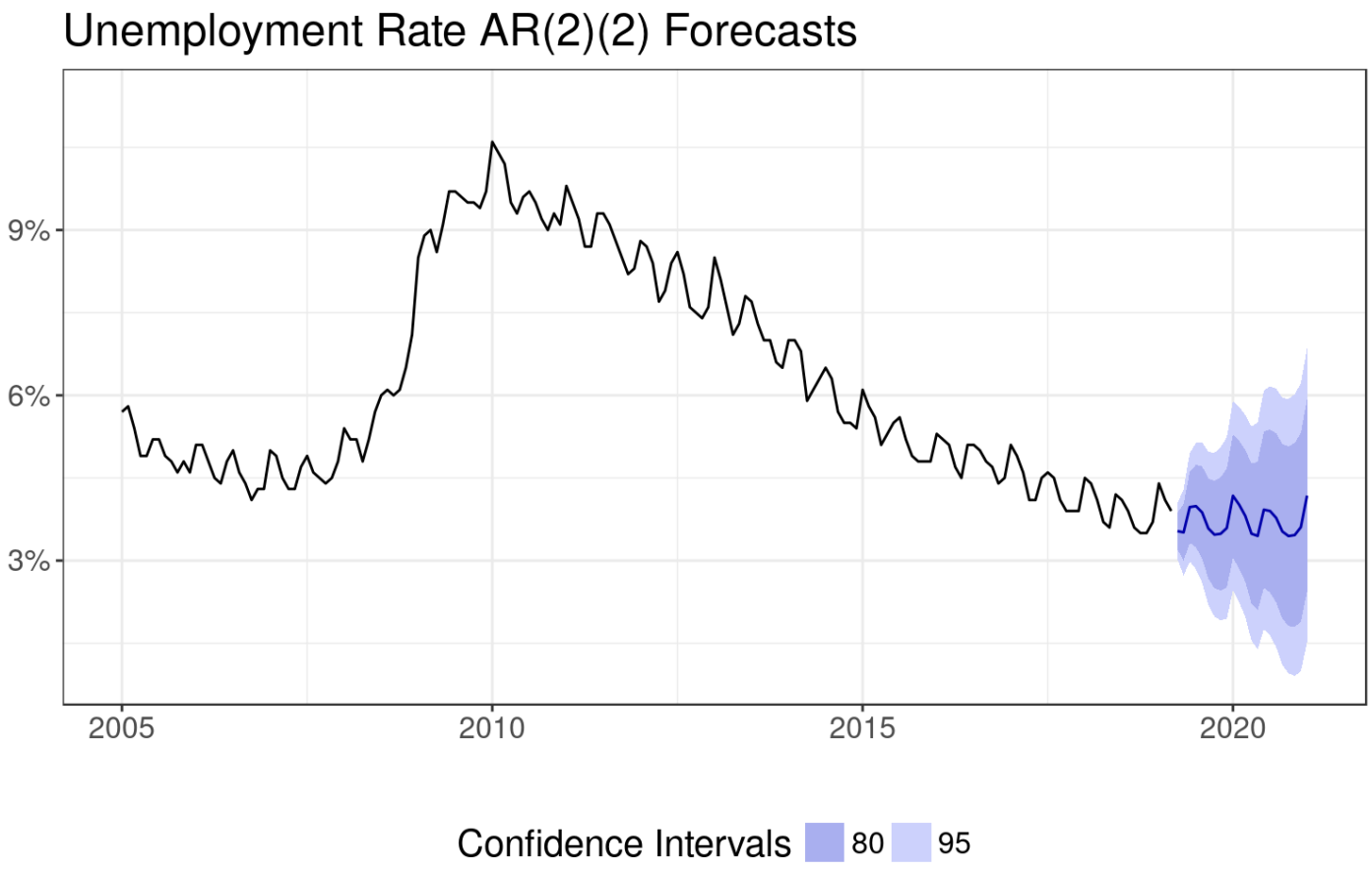
$$x_t = \rho_1 x_{t-1} + \rho_2 x_{t-2} + \phi_1 x_{t-12} + \phi_2 x_{t-24} + \epsilon_t$$
xt=ρ1xt−1+ρ2xt−2+ϕ1xt−12+ϕ2xt−24+ϵt

```
p = 2 # Autoregressive order
k = 2 # Seasonal autoregressive order
sarmodel <- Arima(unrate, order=c(p,0,0),
                  seasonal=list(order=c(k,0,0),period=12))

fsar <- forecast(sarmodel, h=horizon)

autoplot(fsar) +
  theme_bw() +
  scale_y_continuous(labels=percent) +
  scale_x_continuous(limits=c(2005,2021)) +
  theme(text = element_text(size=15)) +
  labs(title="Unemployment Rate AR(2)(2) Forecasts",
       x="", y="",
       fill="Confidence Intervals") +
  theme(legend.position = "bottom")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```

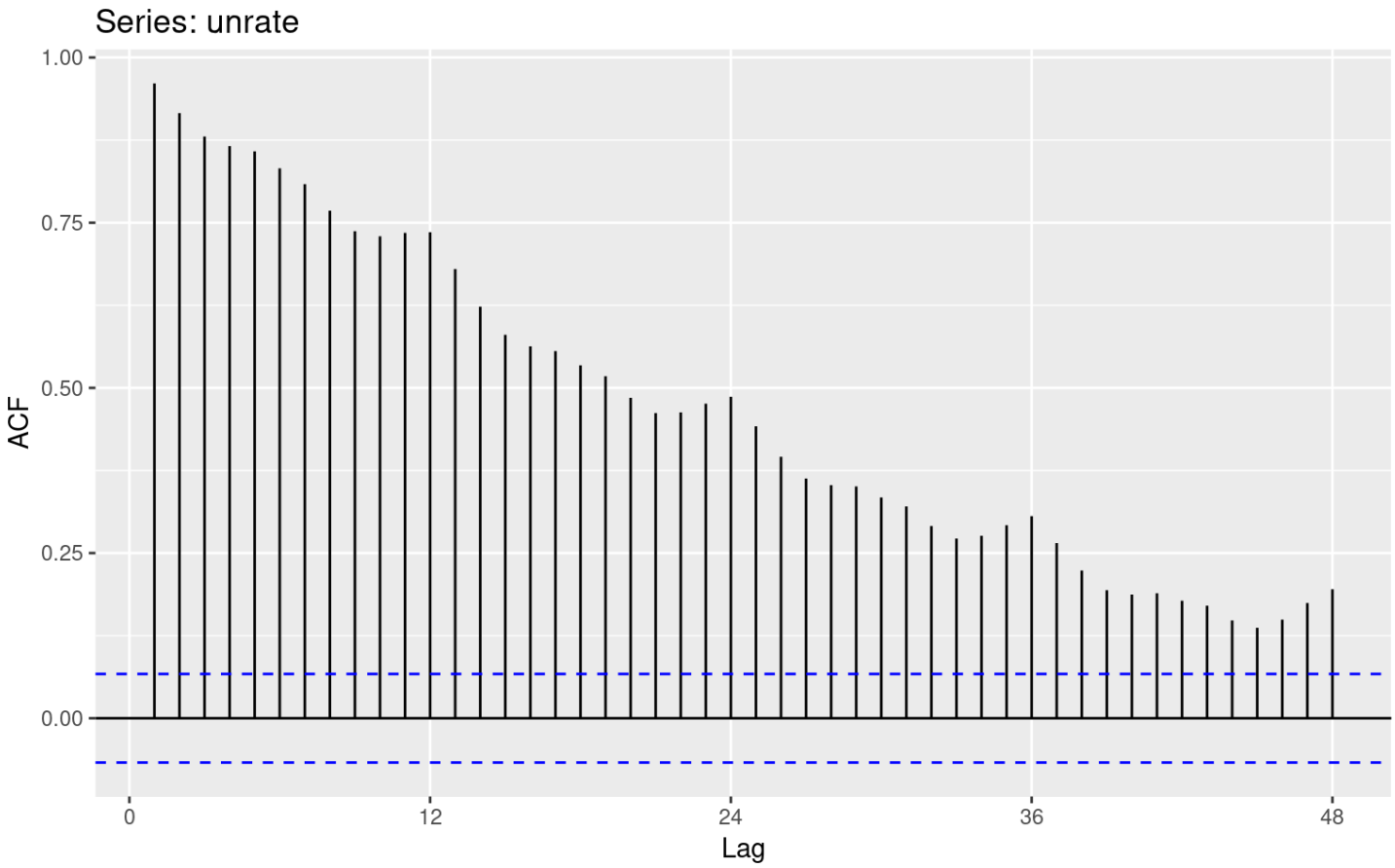


4 Autocorrelation

We can look at autocorrelation for choosing lag length. The autocorrelation coefficient is simply the Pearson correlation coefficient of the series compared to a lag of the same series.

The `ggAcf()` function produces a plot of the **autocorrelation function** for a number of lag lengths:

```
ggAcf(unrate, lag.max = 48)
```



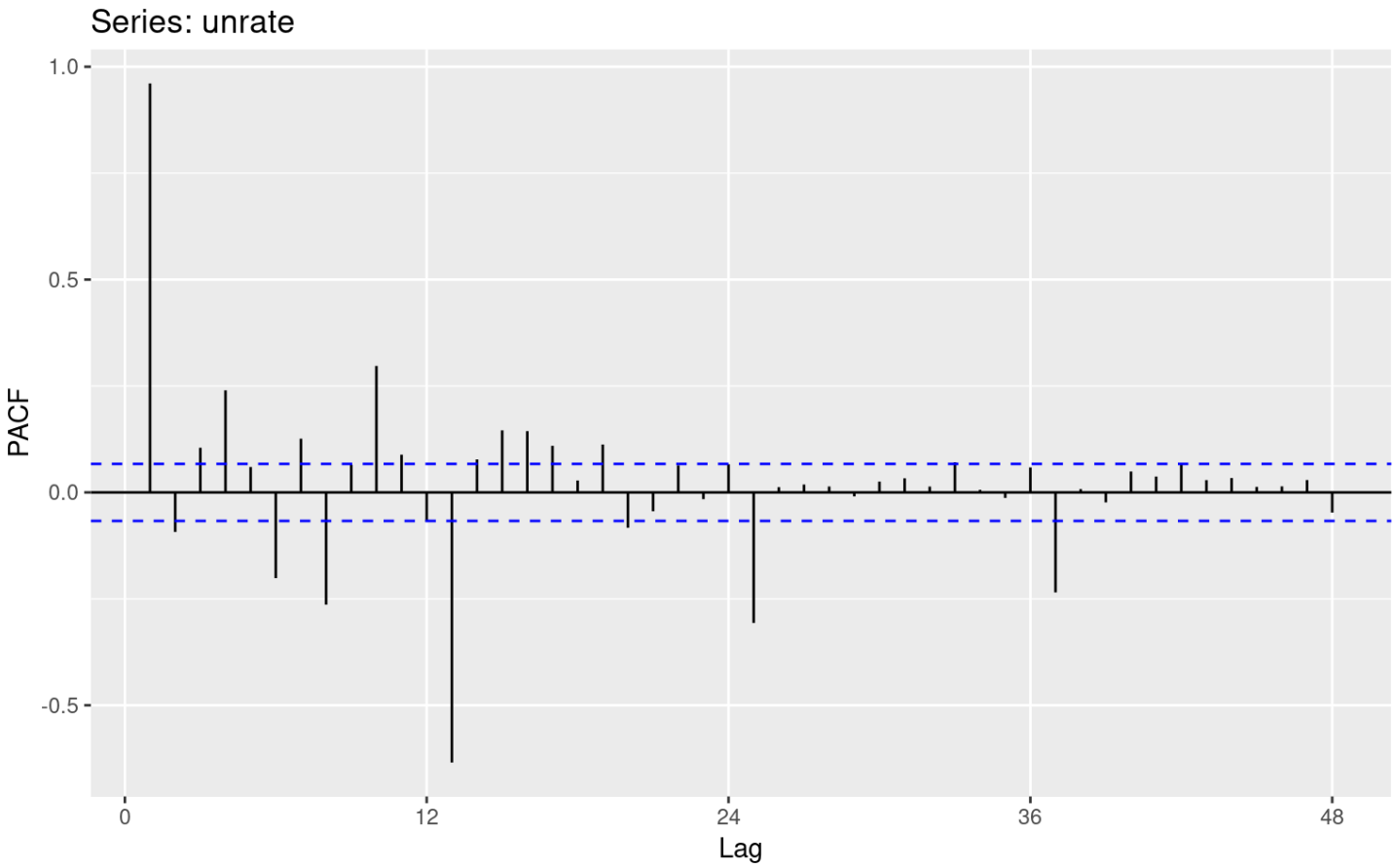
The horizontal blue dotted line shows where the correlation coefficients are statistically significantly greater than equal to zero. All the autocorrelation coefficients between 1 month lag and 48 months lag are statistically significant.

That does not mean that all 48 lags should go into the regression though. Just persistence at one lag is enough to cause correlation at longer lags. Imagine in reality, only one lag is causally important in the regression. Then x_{t-1}xt−1 is correlated with $x_t$xt. It is also true that x_{t-2}xt−2 is correlated with x_{t-1}xt−1. Since x_{t-2}xt−2 is correlated with x_{t-1}xt−1, and x_{t-1}xt−1 is correlated with $x_t$xt, then depending on the strengths of these correlations, x_{t-2}xt−2 may be correlated to $x_t$xt.

The **partial autocorrelation function** nets out explanatory power already explained by later lags. Therefore, the 2nd order partial autocorrelation is the additional explanatory power from x_{t-2}xt−2 for $x_t$xt that is not already explained by x_{t-1}xt−1.

Here is a look at the partial autocorrelation function:

```
ggPacf(unrate, lag.max = 48)
```



Many of the lags between 1 and 11 are statistically significant. We also see statistically significant seasonality, going back three years.

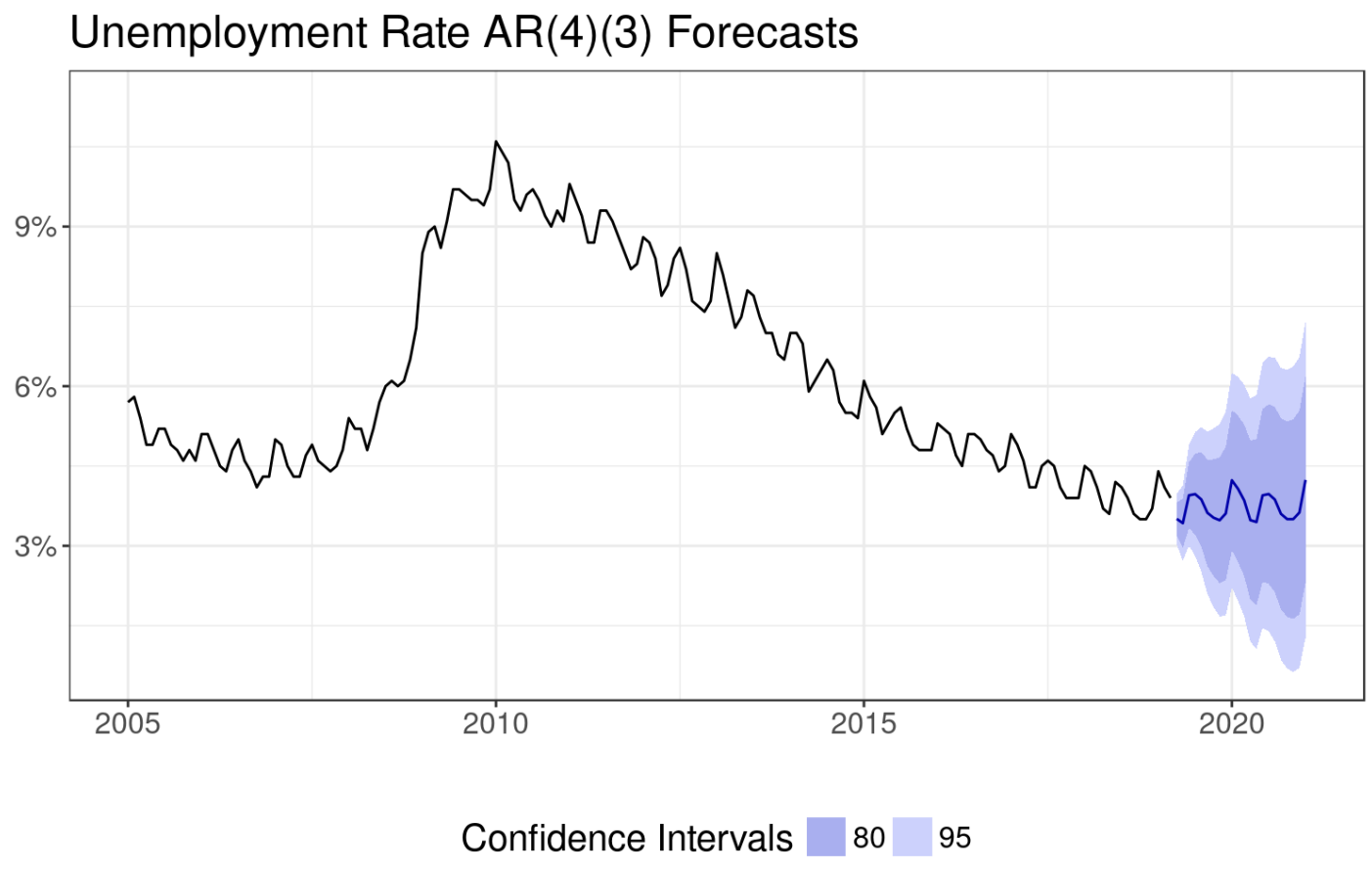
This plot suggests it may be appropriate to use more lags. We need to be careful not to include too many lags in the model, as there is much multicolinearity in these regressors, which will become increasingly difficult for the numerical optimizing procedure used within `Arima()`. We now estimate an AR(4)(3) model. This includes the four most recent lags and x_{t-12}xt−12, x_{t-24}xt−24, and x_{t-36}xt−36 as explanatory variables,

```
p = 4 # Autoregressive order
k = 3 # Seasonal autoregressive order
sarmodel_4_3 <- Arima(unrate, order=c(p,0,0),
                      seasonal=list(order=c(k,0,0),period=12))

fsar <- forecast(sarmodel_4_3, h=horizon)

autoplot(fsar) +
  theme_bw() +
  scale_y_continuous(labels=percent) +
  scale_x_continuous(limits=c(2005,2021)) +
  theme(text = element_text(size=15)) +
  labs(title="Unemployment Rate AR(4)(3) Forecasts",
       x="", y="",
       fill="Confidence Intervals") +
  theme(legend.position = "bottom")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```



5 Moving Average Models

Moving average models suppose a shock to a time series doesn't just hit in one period, but continues over several periods. A moving average model of order qq MA(q) takes the following form:

$$x_t = \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \ldots \theta_q\epsilon_{t-q}.$$
xt=ϵt+θ1ϵt−1+θ2ϵt−2+...θqϵt−q.

The current value of a time series variable is impacted by a new shock today (ϵ_tϵt) and by a shock that started last period (ϵ_1ϵ1), a shock from the period before (ϵ_{t-2}ϵt−2), and so on. Usually, but not always, the magnitudes of the coefficients θ_iθi diminish with longer lags.

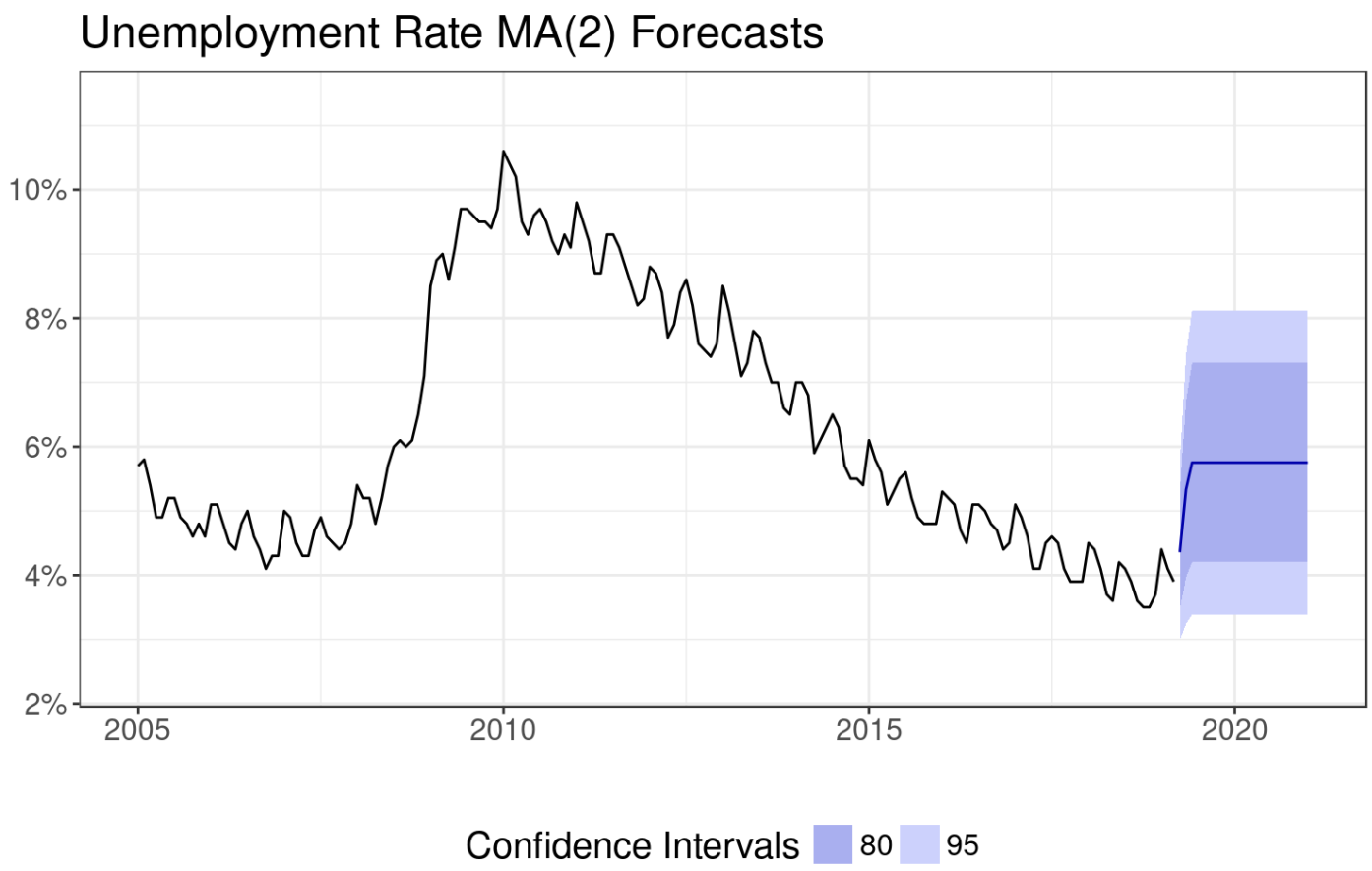
We can use the same `Arima()` function to estimate a moving average model. Consider the following MA(2).

```
q = 2 # Moving average order
mamodel <- Arima(unrate, order=c(0,0,q))

fma <- forecast(mamodel, h=horizon)

autoplot(fma) +
  theme_bw() +
  scale_y_continuous(labels=percent) +
  scale_x_continuous(limits=c(2005,2021)) +
  theme(text = element_text(size=15)) +
  labs(title="Unemployment Rate MA(2) Forecasts",
       x="", y="",
       fill="Confidence Intervals") +
  theme(legend.position = "bottom")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```



Again, we ignored an important seasonality component. In the code below, we consider the following moving average model with seasonal effects.

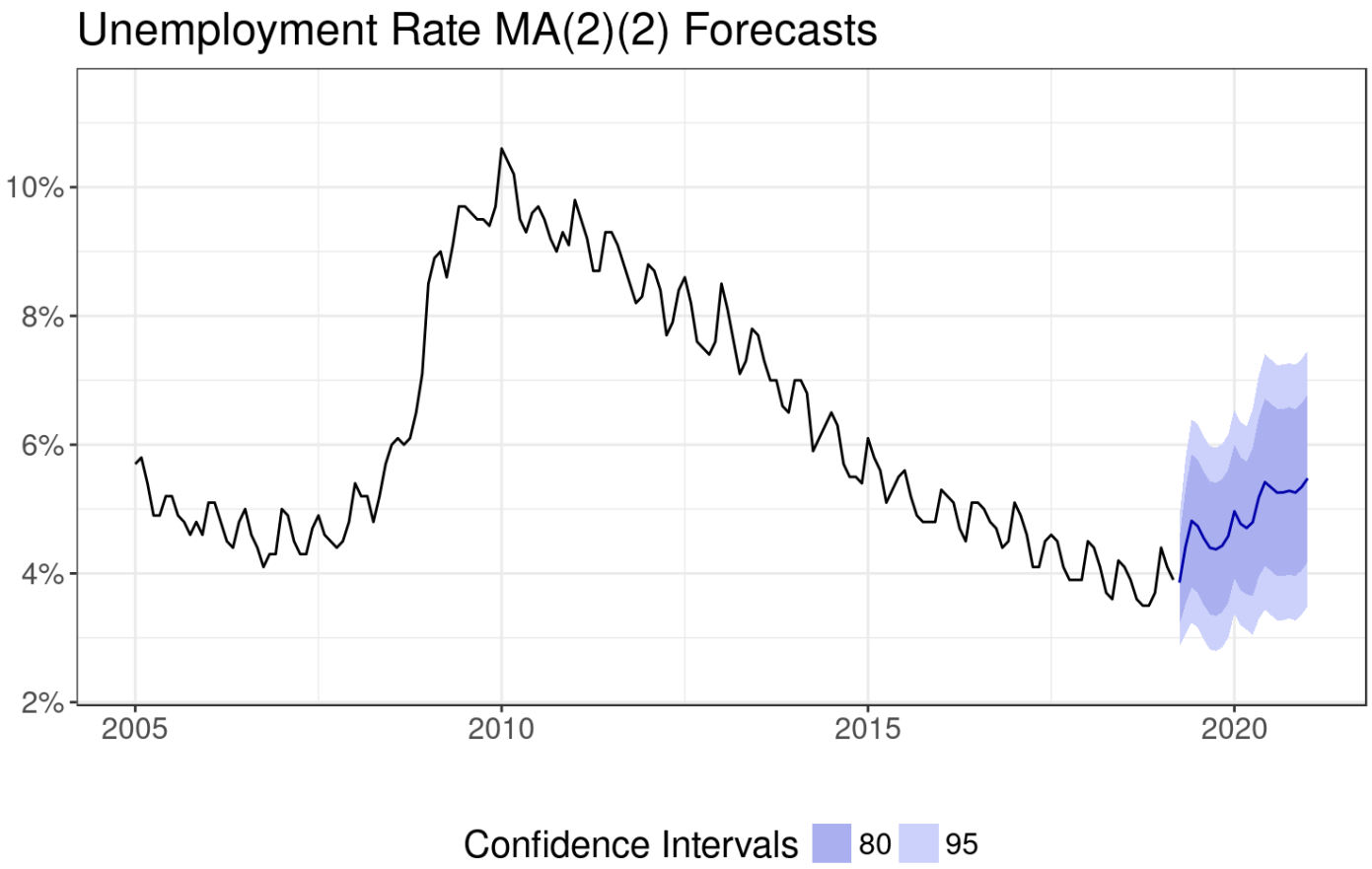
$$x_t = \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \psi_1\psi_{t-12} + \psi_2\epsilon_{t-24}$$
xt=ϵt+θ1ϵt−1+θ2ϵt−2+ψ1ψt−12+ψ2ϵt−24.

```
q <- 2 # Moving average order
m <- 2 # Seasonal moving average order
smamodel <- Arima(unrate, order=c(0,0,q),
                  seasonal = list(order=c(0,0,m), period=12))

fsma <- forecast(smamodel, h=horizon)

autoplot(fsma) +
  theme_bw() +
  scale_y_continuous(labels=percent) +
  scale_x_continuous(limits=c(2005,2021)) +
  theme(text = element_text(size=15)) +
  labs(title="Unemployment Rate MA(2)(2) Forecasts",
       x="", y="",
       fill="Confidence Intervals") +
  theme(legend.position = "bottom")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```



These forecasts suggest the unemployment rate will increase. The reason is that the unemployment rate was higher 2 years ago and 3 years ago than it is today. Our model may or may not be an appropriate description of unemployment behavior.

6 ARMA Models

We can combine what we know about autoregressive models and moving average models into one model. An ARMA(p,q) model is given by,

$$x_t = \rho_1 x_{t-1} + \rho_2 x_{t-2} + \ldots + \rho_p x_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots \theta_q \epsilon_{t-q}.$$
$$xt = \rho 1 xt-1 + \rho 2 xt-2 + \ldots + \rho p xt-p + \epsilon t + \theta 1 \epsilon t-1 + \theta 2 \epsilon t-2 + \ldots \theta q \epsilon t-q.$$

Similar to above, we can also add autoregressive terms and moving average terms for seasonality.

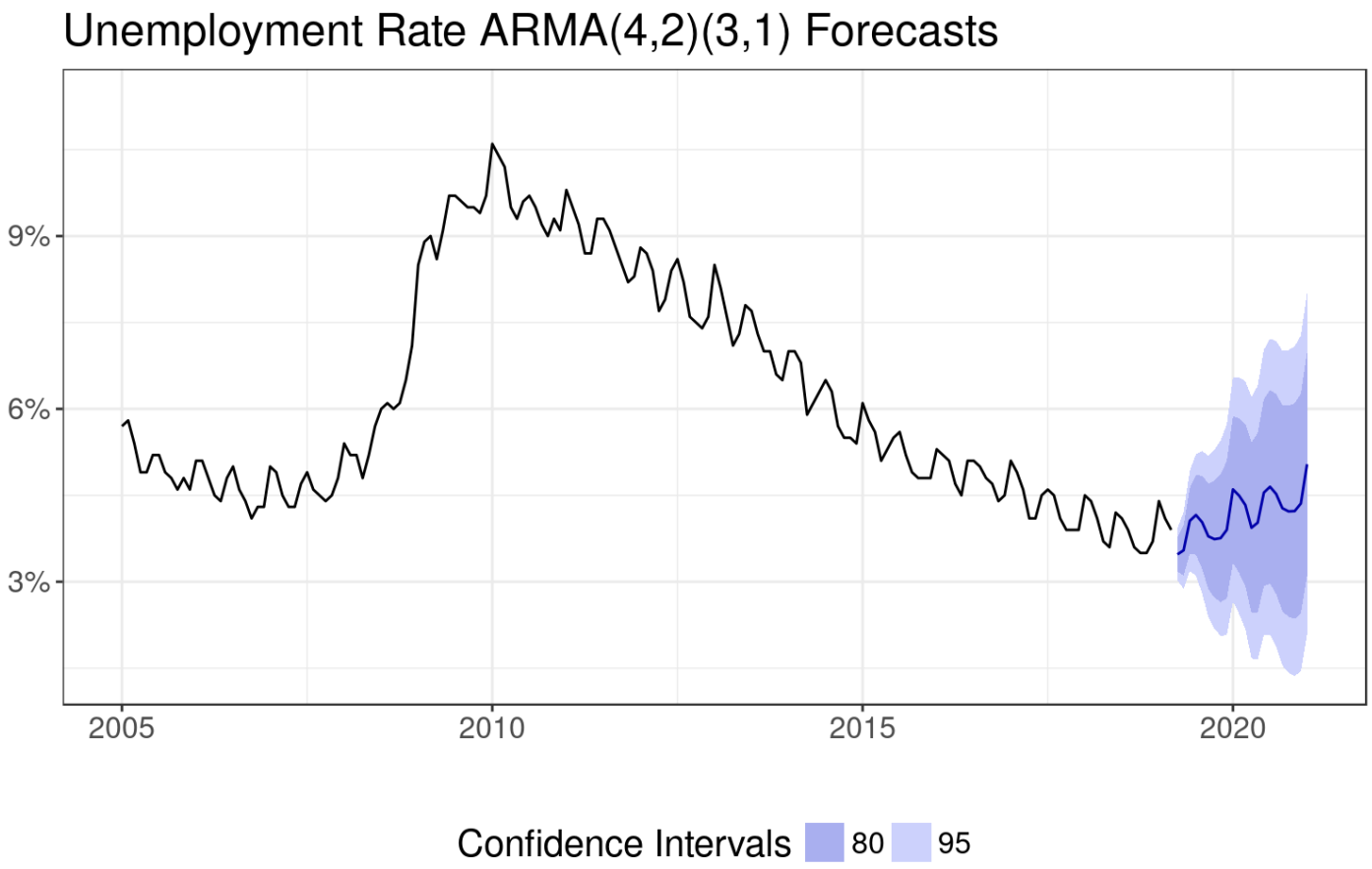
The following model has $p = 2$ $p=2$ autoregressive terms, $q = 2$ $q=2$ moving average terms, $k = 3$ $k=3$ seasonal autoregressive terms, $m = 1$ $m=1$ seasonal moving average terms:

```
p <- 2 # Autoregressive order
q <- 2 # Moving average order
k <- 3 # Seasonal autoregressive order
m <- 1 # Seasonal moving average order
armamodel <- Arima(unrate, order=c(p,0,q),
                   seasonal = list(order=c(k,0,m), period=12))

farma <- forecast(armamodel, h=horizon)

autoplot(farma) +
  theme_bw() +
  scale_y_continuous(labels=percent) +
  scale_x_continuous(limits=c(2005,2021)) +
  theme(text = element_text(size=15)) +
  labs(title="Unemployment Rate ARMA(4,2)(3,1) Forecasts",
       x="", y="",
       fill="Confidence Intervals") +
  theme(legend.position = "bottom")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```



7 Comparing Models

We can compare the fit of models in terms of their in-sample fit and their out-of-sample forecast accuracy.

We will start by breaking the sample into two parts, a **training** set and a **testing** set. We'll estimate all the above models on the training set, and then use the estimated model to forecast over the testing sample period. Since we know the actual values that occurred in the testing set, we can see how closely each model's forecasts align with the actual values.

First, let's slit the sample into the training set and testing set. We set the training set from the beginning of the sample to the end of 2009, and the test set from January 2010 through the end of the available data.

```
trainset <- window(unrate, end=c(2009,12))
testset <- window(unrate, start=c(2010,1))
```

Now, let's estimate all of the models above on the training set. We will focus on the accuracy at only a one-month horizon (the best fitting model may differ depending on the horizon considered).

```
horizon = 1

# Naive model
fNaive <- naive(trainset, h=horizon)
# Seasonal naive model
fSeasonNaive <- snaive(trainset, h=horizon)

# AR(2)
armodel <- Arima(trainset, order=c(2,0,0))
far <- forecast(armodel, h=horizon)

# AR(2)(2) Seasonal
sarmodel <- Arima(trainset, order=c(2,0,0),
                  seasonal=list(order=c(2,0,0), period=12))
fsar <- forecast(sarmodel, h=horizon)

# MA(2)
mamodel <- Arima(trainset, order=c(0,0,2))
fma <- forecast(mamodel, h=horizon)

# MA(2)(2) Seasonal
smamodel <- Arima(trainset, order=c(0,0,2),
                  seasonal = list(order=c(0,0,2), period=12))
fsma <- forecast(smamodel, h=horizon)

# ARMA(2,2)(3,1)
armamodel <- Arima(trainset, order=c(2,0,2),
                  seasonal = list(order=c(3,0,1), period=12))

farma <- forecast(armamodel, h=horizon)
```

The accuracy function reveals several measures of in-sample fit and forecast accuracy. Let's look at the naive forecast first.

```
accuracy(fNaive, testset)
```

```
##                       ME        RMSE         MAE        MPE     MAPE
## Training set 7.671602e-05 0.004781498 0.003581427 -0.2841013 6.616608
## Test set     9.000000e-03 0.009000000 0.009000000  8.4905660 8.490566
##                   MASE       ACF1
## Training set 0.4022099 0.07769048
## Test set     1.0107395         NA
```

The measures of fit include the following,

- ME: Mean error (positives and negatives may cancel each other out)
- RMSE: Root mean squared error (similar to standard error formula in regression)
- MAE: Mean absolute error (absolute values of errors are averaged)
- MPE: Mean percentage error (error as a percentage of the value, positives and negatives can cancel each other out)
- MAPE: Mean absolute percentage error (absolute value of errors as a percentage of the data values)
- MASE: Mean absolute scaled error (average of absolute values of errors scaled to number of standard deviations)
- ACF1: First-order autocorrelation of errors

Now, we look at all the forecasts fit statistics.

```
accuracy(fNaive, testset)
```

```
##                       ME        RMSE         MAE        MPE     MAPE
## Training set 7.671602e-05 0.004781498 0.003581427 -0.2841013 6.616608
## Test set     9.000000e-03 0.009000000 0.009000000  8.4905660 8.490566
##                   MASE       ACF1
## Training set 0.4022099 0.07769048
## Test set     1.0107395         NA
```

```
accuracy(fSeasonNaive, testset)
```

```
##                       ME       RMSE         MAE        MPE     MAPE
## Training set 0.0009016393 0.01236943 0.008904372 -0.7828366 15.27540
## Test set     0.0210000000 0.02100000 0.021000000 19.8113208 19.81132
##                   MASE      ACF1
## Training set 1.000000 0.9633191
## Test set     2.358392        NA
```

```
accuracy(far, testset)
```

```
##                       ME        RMSE         MAE        MPE     MAPE
## Training set 3.463581e-05 0.004711211 0.003564693 -0.6939465 6.687506
## Test set     1.052950e-02 0.010529499 0.010529499  9.9334894 9.933489
##                   MASE        ACF1
## Training set 0.4003307 0.007581665
## Test set     1.1825089          NA
```

```
accuracy(fsar, testset)
```

```
##                       ME        RMSE         MAE        MPE     MAPE
## Training set 5.467065e-05 0.002666008 0.002018923 0.01298753 3.798248
## Test set     1.256674e-03 0.001256674 0.001256674 1.18554125 1.185541
##                   MASE        ACF1
## Training set 0.2267339 -0.05694677
## Test set     0.1411300          NA
```

```
accuracy(fma, testset)
```

```
##                       ME        RMSE         MAE       MPE    MAPE
## Training set 1.145711e-05 0.006842665 0.005401246 -2.996159 10.4579
## Test set     2.088518e-02 0.020885177 0.020885177 19.702997 19.7030
##                   MASE      ACF1
## Training set 0.6065836 0.4037845
## Test set     2.3454970        NA
```

```
accuracy(fsma, testset)
```

```
##                       ME        RMSE        MAE       MPE     MAPE
## Training set 8.788744e-05 0.005244473 0.00397145 -1.691998  7.49853
## Test set     1.078725e-02 0.010787247 0.01078725 10.176648 10.17665
##                   MASE     ACF1
## Training set 0.4460113 0.495895
## Test set     1.2114551       NA
```

```
accuracy(farma, testset)
```

```
##                       ME        RMSE         MAE         MPE     MAPE
## Training set 6.377148e-05 0.002370286 0.001811179 0.003972559 3.472486
## Test set     2.131837e-03 0.002131837 0.002131837 2.011167062 2.011167
##                   MASE         ACF1
## Training set 0.2034034 -0.008490175
## Test set     0.2394147           NA
```

In terms of RMSE and MAE, the AR(2)(2) appears to be the best model for forecasting, among those considered.

There is little theory dictating how many AR or MA lags one should put in the model, and there are dozens or hundreds of combinations you could consider. There exist many automated procedures used for determining the best number of lags.
The `auto.arima()` function uses a method by Hyndman & Khandakar (2008) that iterates over many possibilities and chooses a set of lags based on a measure of in-sample fit that also includes penalties for adding more parameters.

The following code searches for the best fit ARMA(p,q)(k,m) model.

```
aa <- auto.arima(trainset,d=0,D=0)
aa
```

```
## Series: trainset
## ARIMA(5,0,1)(1,0,0)[12] with non-zero mean
##
## Coefficients:
##          ar1      ar2      ar3     ar4      ar5      ma1    sar1    mean
##       1.6695  -0.5364  -0.2004  0.0772  -0.0407  -0.6308  0.8389  0.0574
## s.e.  0.1038   0.1238   0.0752  0.0723   0.0451   0.0976  0.0203  0.0070
##
## sigma^2 estimated as 7.746e-06:  log likelihood=3317.52
## AIC=-6617.04   AICc=-6616.8   BIC=-6575.53
```

The best fitting model appears to have $p = 5$p=5 autoregressive lags, $q = 1$q=1 moving average lag, $k = 1$k=1autoregressive seasonal lag, and $m = 0$m=0 moving average seasonal lags.

Let us look examine the in-sample and out-of-sample accuracy at the one-month horizon:

```
faa <- forecast(aa, horizon=1)
accuracy(faa, testset)
```

```
##                       ME        RMSE         MAE        MPE     MAPE
## Training set 4.110157e-05 0.002768127 0.002088473 -0.1431555 3.966018
## Test set     7.676284e-03 0.009327065 0.007884719  8.5117500 8.729964
##                   MASE        ACF1 Theil's U
## Training set 0.2345447 -0.00786818        NA
## Test set     0.8854886  0.87804656  2.907489
```

The in-sample fit is similar to some of the others we examined above, and some of the above even slightly out-perform this one. The out-of-sample fit is actually worse than several of the models examined above.