MTH 325, Optional Lab #2, Spring 2024 Name _____

Instructions: Follow along with the tutorial portion of the lab. Replicate the code examples in R on your own, along with the demonstration. Then use those examples as a model to answer the questions/perform the tasks that follow. Copy and paste the results of your code to answer questions where directed. Submit your response file and the code used (both for the tutorial and part two). Your code file and your lab response file should each include your name inside.

In the main labs for this course, we consider the steps for performing an ARIMA model of a time series. In this lab we'll look at smoothing methods. The first such method we'll consider is exponential smoothing.

First we load the packages (you'll need to install fpp2). We'll consider two datasets to work with: the google stock dataset, and a second dataset on airline passengers (qcement).

```
library(tidyverse)
library(fpp2)

goog.train <- window(goog, end = 900)
goog.test <- window(goog, start = 901)

qcement.train <- window(qcement, end = c(2012, 4))
qcement.test <- window(qcement, start = c(2013, 1))
```

As noted in our other modeling labs for time series, test and training data are split differently than cross-sectional data. It's not random, but we use past data to forecast future outcomes.

Now we can model and plot the data. The alpha is the degree of smoothing.

```
ses.goog <- ses(goog.train, alpha = .2, h = 100)
autoplot(ses.goog)
```

The flat line in the forecast suggests we are not capturing the trend, so we'll try differencing to remove the trend first.

```
goog.dif <- diff(goog.train)
autoplot(goog.dif)

ses.goog.dif <- ses(goog.dif, alpha = .2, h = 100)
autoplot(ses.goog.dif)
```

In order to assess the fit, however, we will also need to compare the model to a differenced test set as well.

```
goog.dif.test <- diff(goog.test)
accuracy(ses.goog.dif, goog.dif.test)
```

This accuracy() function outputs several measurements of fit that we can compare between the training and the test data.

We can go further toward selecting the best fit model by comparing the model with different levels of smoothing (adjusting alpha).

```
alpha <- seq(.01, .99, by = .01)
RMSE <- NA
for(i in seq_along(alpha)) {
  fit <- ses(goog.dif, alpha = alpha[i], h = 100)
  RMSE[i] <- accuracy(fit, goog.dif.test)[2,2]
}

alpha.fit <- data_frame(alpha, RMSE)
alpha.min <- filter(alpha.fit, RMSE == min(RMSE))

ggplot(alpha.fit, aes(alpha, RMSE)) +  geom_line() +
  geom_point(data = alpha.min, aes(alpha, RMSE), size = 2, color = "red")
```

Here, we have created a for loop to test nearly 100 different levels of smoothing and collect the RMSE results. Then we find the best one, and finally plot the result on a graph so that we can see visually what the best setting is for our model.

The values for smoothing that work best appear to be very low, less than 0.05. Let's refit the model with the new alpha value and compare with a higher setting.

```
ses.goog.opt <- ses(goog.dif, alpha = .05, h = 100)
accuracy(ses.goog.opt, goog.dif.test)

p1 <- autoplot(ses.goog.opt) + theme(legend.position = "bottom")
p2 <- autoplot(goog.dif.test) + autolayer(ses.goog.opt, alpha = .5) +
  ggtitle("Predicted vs. actuals for the test data set")

gridExtra::grid.arrange(p1, p2, nrow = 1)
```

See resource [1] for using exponential smoothing with seasonal data.

Let's look at Bayesian smoothing for times series (BSTS). Be sure to install the bsts package first.

```
library(bsts)
data(iclaims)
ss <- AddLocalLinearTrend(list(), initial.claims$iclaimsNSA)
ss <- AddSeasonal(ss, initial.claims$iclaimsNSA, nseasons = 52)
model1 <- bsts(initial.claims$iclaimsNSA,
               state.specification = ss, niter = 1000)
```

The first step is to select a state (space) for the model. Here, we are using a linear trend. You may want to graph the data first to determine the best state space. The second call is to add seasonal information. Selecting nseasons to be 52 says that the "seasons" are weeks. The model is then fit using a Monte Carlo method (MCMC) for 1000 iterations.

From here, we can plot the model.

```
plot(model1)
plot(model1, "components")
```

From here, we can predict and plot our forecast. We are plotting the original 156 weeks (3 years) and then 12 additional weeks into the future.

```
pred1 <- predict(model1, horizon = 12)
plot(pred1, plot.original = 156)
```

A nice feature of BSTS is that we can incorporate other variables into our model besides just time.

```
model2 <- bsts(iclaimsNSA ~ ., state.specification = ss,
               niter = 1000, data = initial.claims)

model3 <- bsts(iclaimsNSA ~ ., state.specification = ss,
               niter = 1000, data = initial.claims,
               expected.model.size = 5)
```

When size is not specified, the default size is 1, but we can specify our expectation in including more variables. We can then visualize the results as before.

```
plot(model2, "comp")
plot(model3, "comp")
```

We can also visually inspect the values of the coefficients in the models to see their relative sizes.

```
plot(model2, "coef")
plot(model3, "coef")
```

Finally, we can compare all three models we built on a single graph to determine the best results.

```
CompareBstsModels(list("Model 1" = model1,
                       "Model 2" = model2,
                       "Model 3" = model3),
                  colors = c("black", "red", "blue"))
```

We can also use BSTS to predict into the future, not just model past data.

Our final method to consider here is an STL model, or Seasonal and Trend decomposition using LOESS. This function is also available in the fpp2 package along with the elecequip dataset we'll use.

```
data("elecequip")
elecequip %>%
  stl(t.window=13, s.window="periodic", robust=TRUE) %>%
  autoplot()
```

In the stl function we can specify the time window and the seasonal window, or we can use the mstl() function with some reasonably good defaults.

```
elecequip %>%
  mstl(robust=TRUE) %>%
  autoplot()
```

The output here is similar to our other time series decomposition methods.

Tasks:

1. Use the air passengers data we loaded (qcement) to perform an exponential smoothing analysis. Find the best alpha and plot the results. Describe the process, the best alpha, and include any graphs.

2. Using the same air passengers data for the previous question, construct a BSTS model. Describe the process, any choice of parameters, and include any graphs.

3. Perform an STL analysis of the air passenger data. Specify any parameters you used and the graphs produced. Describe what you see.

Resources:

1. https://www.geeksforgeeks.org/exponential-smoothing-in-r-programming/#
2. https://uc-r.github.io/ts_exp_smoothing
3. https://www.seascapemodels.org/rstats/2017/06/21/bayesian-smoothing.html
4. https://www.unofficialgoogledatascience.com/2017/07/fitting-bayesian-structural-time-series.html
5. https://bookdown.org/raghavendrajain/dengueforecasting/time-series-analysis-in-r.html#a-bayesian-structural-time-series-model
6. https://minimizeregret.com/post/2020/06/07/rediscovering-bayesian-structural-time-series/
7. https://otexts.com/fpp2/stl.html