A Quick and Amateurish Example of Time Series Analysis with R

Data

The data used here come from the forecast made by Pricing and Strategy (I think). From what I gather, they put out a monthly forecast for various things concerning applications and funding. I only looked at the SAF monthly application data.

After saving as a csv file from excel, we can read the data into R as follows:

```
SAFapps <- read.csv("SAFapps.csv", header=TRUE)
```

This saves the data into an R object called SAFapps, including the first column of header names. Next, we can look at the structure of the dataset:

```
str(SAFapps)
```

```
## 'data.frame':    70 obs. of  2 variables:
##  $ Month: Factor w/ 70 levels "2012-01","2012-02",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ SAF  : num  358707 459105 426594 368051 417596 ...
```

```
summary(SAFapps)
```

```
##       Month          SAF
##  2012-01: 1   Min.   :238856
##  2012-02: 1   1st Qu.:323692
##  2012-03: 1   Median :366313
##  2012-04: 1   Mean   :364362
##  2012-05: 1   3rd Qu.:402966
##  2012-06: 1   Max.   :459105
##  (Other):64
```

The ts() function in R converts the data into a Time Series obect in R:

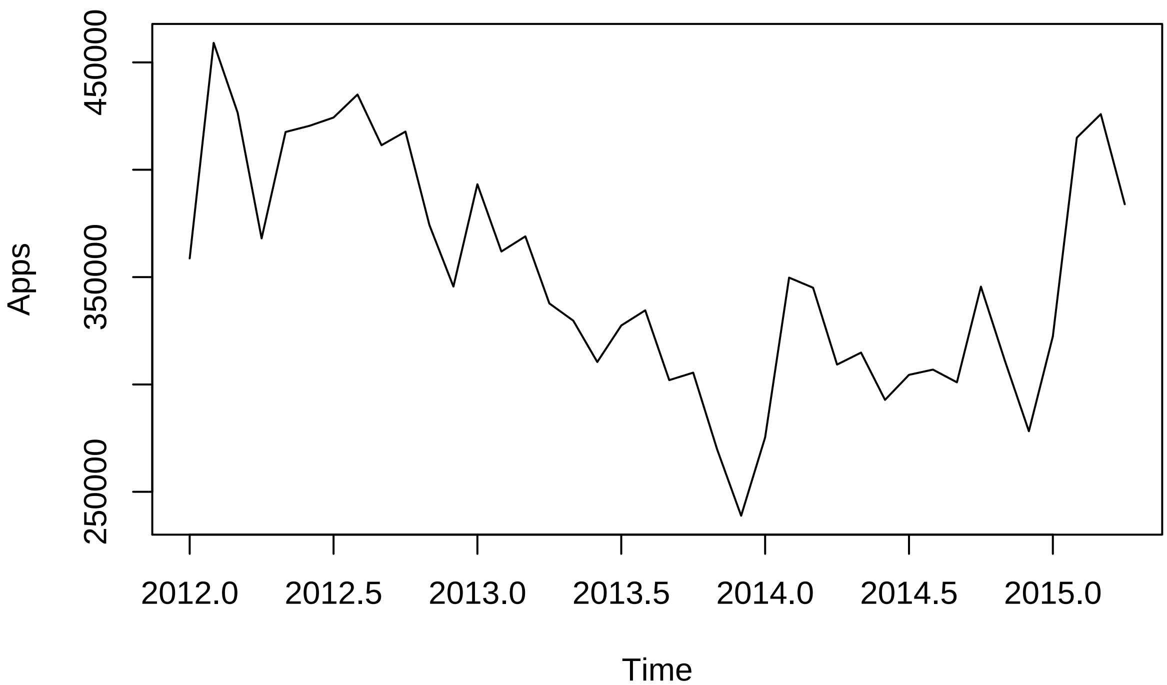
```
SAFts <- ts(SAFapps,start=c(2012,1), end=c(2015,4),frequency = 12)
head(SAFts)
```

```
##      Month    SAF
## [1,]     1 358707
## [2,]     2 459105
## [3,]     3 426594
## [4,]     4 368051
## [5,]     5 417596
## [6,]     6 420474
```

```
tail(SAFts)
```

```
##       Month    SAF
## [35,]    35 311009
## [36,]    36 278227
## [37,]    37 322438
## [38,]    38 414959
## [39,]    39 425904
## [40,]    40 383936
```

```r
plot(SAFts[,2],ylab = "Apps")
```



Exponential Smoothing with Trend and Seasonality

Holt-Winters

Simple exponential smoothing can fit a time series with no trend or seasonality. The Holt-Winters approach is a generalization that can model trend and/or seasonality. In R, the ets function is part of the forecast library and provides an implementation of the Holt-Winters method. There are various options that can be specified, but the default will find a “best-fitting” model according to certain criteria.

```r
library(forecast)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: timeDate
```

```
## This is forecast 7.1
```

```
fit.ets <- ets(SAFts[,2])
fit.ets
```

```
## ETS(M,N,A)
##
## Call:
##  ets(y = SAFts[, 2])
##
##   Smoothing parameters:
##     alpha = 0.8805
##     gamma = 1e-04
##
##   Initial states:
##     l = 364125.5818
##     s=-49391.83 -24467.85 13525.44 -4395.094 15656.25 7794.271
##            -15341.09 2562.85 987.1788 34472.92 42751.84 -24154.89
##
##   sigma:  0.0547
##
##      AIC     AICc      BIC
## 963.3327 980.1327 986.9770
```

The resulting model is ETS(M,N,A). The first letter M means the error term used was Multiplicative, the second letter N means No trend, and the last letter A means the seasonality was Additive. These can be specified when running the model to force ETS to use any particular parameterization. The smoothing parameters alpha and gamma are the parameters for the level and seasonality, respectively. AIC, AICc, and BIC are model-fitting metrics used to gauge goodness of fit and evaluate different models.

We can get measures of error with the accuracy function:

```
accuracy(fit.ets)
```

```
##                    ME     RMSE      MAE          MPE     MAPE      MASE
## Training set 554.1934 21978.83 14424.46 -0.005551813 3.953011 0.2371478
##                     ACF1
## Training set -0.03580401
```

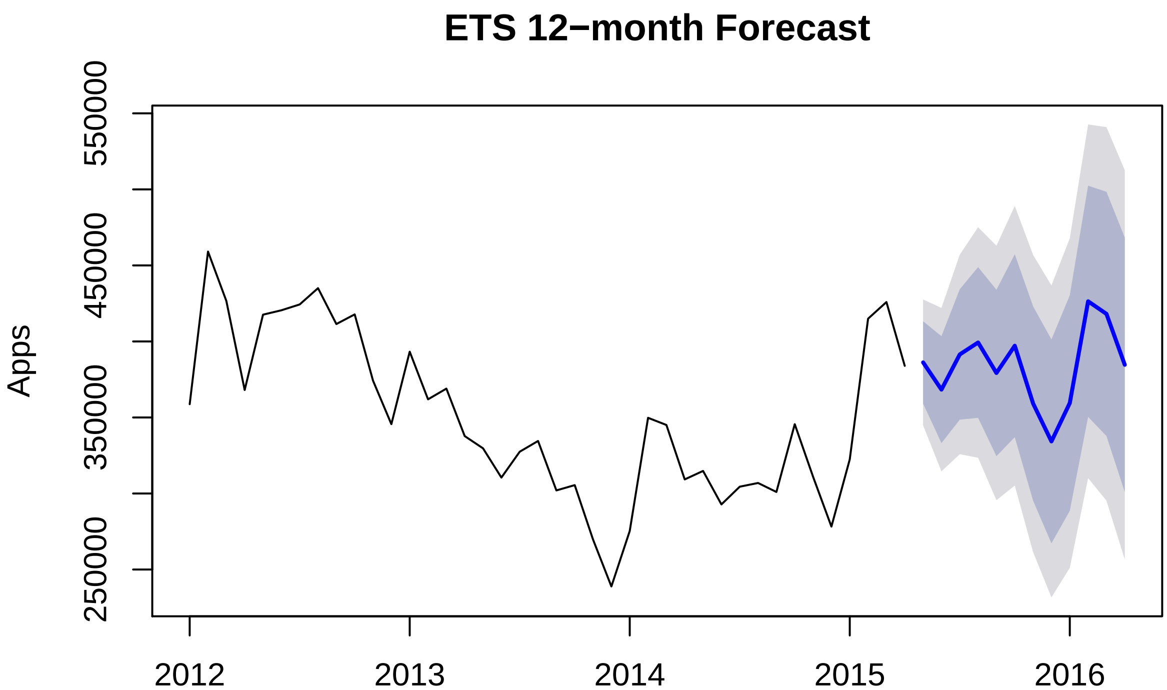
and we forecast future values:

```
pred.ets <- forecast(fit.ets,12)
pred.ets
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## May 2015       386209.9 359155.9 413264.0 344834.3 427585.6
## Jun 2015       368304.1 333165.7 403442.5 314564.6 422043.6
## Jul 2015       391434.0 348541.3 434326.7 325835.2 457032.7
## Aug 2015       399298.7 349726.5 448870.8 323484.6 475112.7
## Sep 2015       379247.8 324539.7 433955.9 295579.0 462916.6
```

```
## Oct 2015       397168.6 337041.3 457296.0 305211.8 489125.5
## Nov 2015       359175.3 295279.3 423071.3 261454.8 456895.8
## Dec 2015       334249.6 267181.9 401317.2 231678.5 436820.7
## Jan 2016       359495.2 288647.4 430343.0 251142.9 467847.6
## Feb 2016       426392.7 350362.3 502423.1 310114.2 542671.1
## Mar 2016       418117.2 337796.4 498438.0 295277.1 540957.3
## Apr 2016       384627.1 300964.4 468289.8 256676.1 512578.2
```

```
plot(pred.ets, main = "ETS 12-month Forecast", ylab = "Apps")
```



The different shades of blue are the 80% and 95% confidence intervals.

ARIMA

Auto Regressive Integrated Moving Average (ARIMA) models are another very popular way to model time series. This was not mentioned in the textbook but is a very important class of models that R is well-equipped to handle. The textbook mentioned stationarity of a series, but I don't recall if the relevance was explicitly stated. ARMA modeling requires the series to be stationary. The Integrated(I) part of ARIMA models comes from the fact that if a series is not stationary, it must be differenced at least once to achieve stationarity. Integrated means the series must be summed back later to counteract the differencing. One common test for stationarity is the Dickey-Fuller test:

```
library(tseries)
adf.test(SAFts[,2])
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  SAFts[, 2]
## Dickey-Fuller = -0.6288, Lag order = 3, p-value = 0.9679
## alternative hypothesis: stationary
```

The p-value implies the series is not stationary and so we should try to difference at least once.

```
dSAFts <- diff(SAFts)
adf.test(dSAFts[,2])
```

```
## Warning in adf.test(dSAFts[, 2]): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  dSAFts[, 2]
## Dickey-Fuller = -4.8039, Lag order = 3, p-value = 0.01
## alternative hypothesis: stationary
```

The low p-value shows we now have a stationary series.

ARIMA models are typically written ARIMA(p,d,q), where p is the autoregressive parameter, d is the number of time the series had to be differenced, and q is the moving average parameter. There are a variety of techniques to estimate these parameters, but here we will just use R's automated estimation.

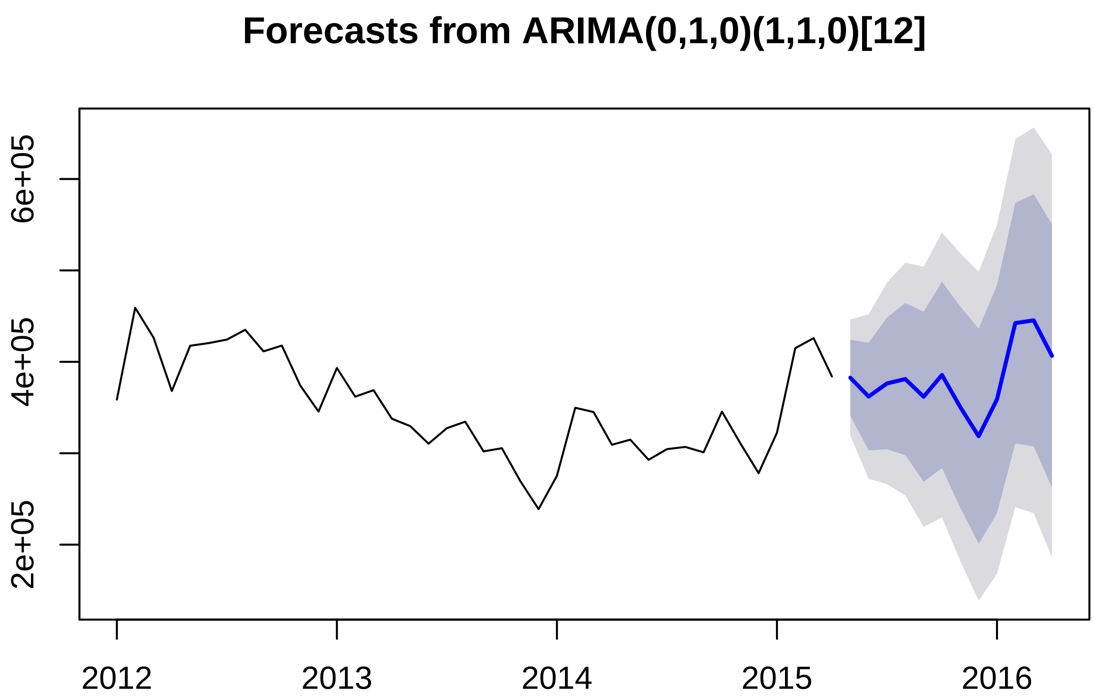
```
fit.arima <- auto.arima(SAFts[,2])
fit.arima
```

```
## Series: SAFts[, 2]
## ARIMA(0,1,0)(1,1,0)[12]
##
## Coefficients:
##          sar1
##       -0.5047
## s.e.   0.1731
##
## sigma^2 estimated as 1.054e+09:  log likelihood=-320.03
## AIC=644.07   AICc=644.57   BIC=646.66
```

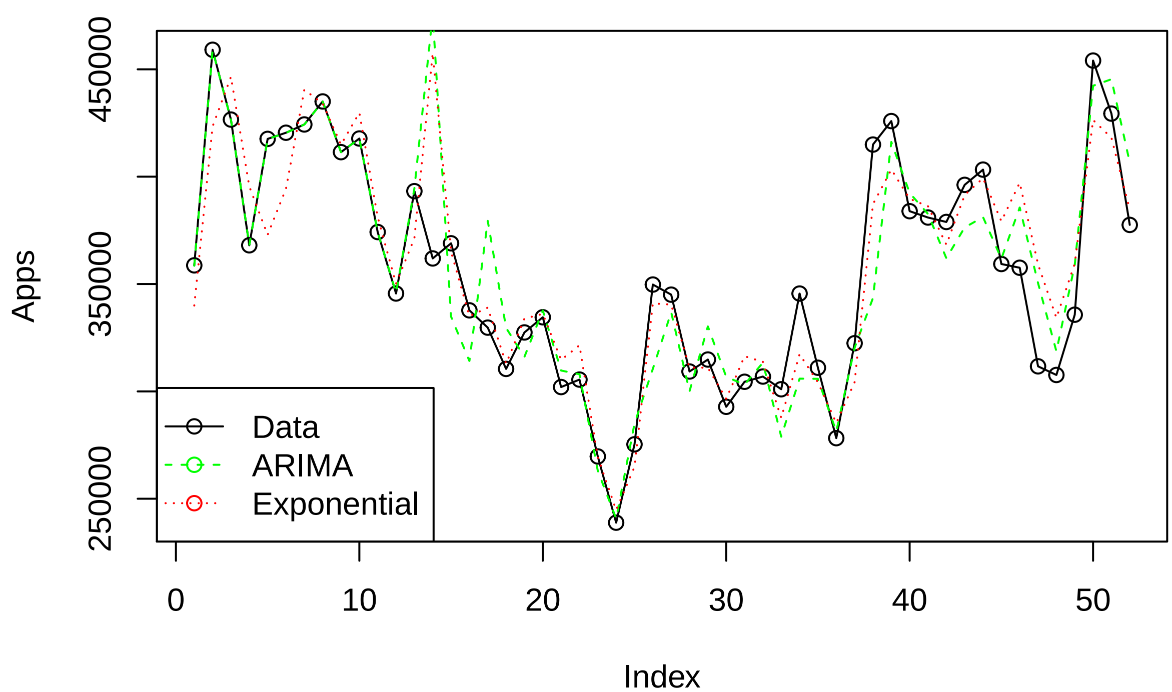
```
accuracy(fit.arima)
```

```
##                    ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 706.8997 26168.28 13527.63 0.08112678 3.919209 0.2224033
##                    ACF1
## Training set -0.0608335
```

```
pred.arima <- forecast(fit.arima,12)
plot(pred.arima)
```




```r
a <- c(pred.arima$fitted,pred.arima$mean)
e <- c(pred.ets$fitted,pred.ets$mean)
plot(SAFapps[1:52,2],type="o",ylab="Apps")
lines(a,lty=2, col="green")
lines(e,lty=3,col="red")
legend("bottomleft",lty=c(1,2,3),pch=1,col=c(1,"green","red"),c("Data","ARIMA","Exponential"))
```



Finally, we can compare the models based on fit to data the model used in estimation:

```r
accuracy(pred.arima)
```

```
##                    ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 706.8997 26168.28 13527.63 0.08112678 3.919209 0.2224033
##                    ACF1
## Training set -0.0608335
```

```r
accuracy(pred.ets)
```

```
##                    ME     RMSE      MAE          MPE     MAPE      MASE
## Training set 554.1934 21978.83 14424.46 -0.005551813 3.953011 0.2371478
##                     ACF1
## Training set -0.03580401
```

So it appears the exponential models have a better fit. We can also compare RMSE on the 12 data points we left out

```r
sqrt(sum((pred.arima$mean - SAFapps[41:52,2])^2)/12)
```

```
## [1] 21142.6
```

```r
sqrt(sum((pred.ets$mean - SAFapps[41:52,2])^2)/12)
```

```
## [1] 23478.81
```

So actually the ARIMA model does better on the 12 actual data points the model was not trained on.