Appendix A - Monthly Amtrak Ridership Forecasting Final Project

Team 3: Jimmy Nguyen, Luke Awino

12/04/2021

Contents

Libraries

```
library(astsa)
library(readr)
library(forecast)
library(zoo)
library(xts)
library(pander)
library(tidyverse)
library(tseries)
library(lubridate)

knitr::opts_chunk$set(warning = FALSE, message = FALSE)
```

Data Set

Code:

```
# Load the data set from CSV file
df <- read_csv("../Data/Amtrak Ridership Data.csv")


# Rename columns
names(df)[1] <- 'Dates'
names(df[2]) <- 'Number_of_Passengers'


# First 12 months in 1991
head(df, n = 12) %>%
  pander(style = "grid", caption = "First 12 Months - 1991")
```

Table 1: First 12 Months - 1991

Dates	Number of Passengers
Jan-91	1708917
Feb-91	1620586
Mar-91	1972715
Apr-91	1811665
May-91	1974964
Jun-91	1862356
Jul-91	1939860
Aug-91	2013264
Sep-91	1595657
Oct-91	1724924
Nov-91	1675667
Dec-91	1813863

Data Exploration

Code:

```
# convert to time series object
df<- ts(data = df[,2], start = c(1991,1),
        end = c(2013,5), frequency = 12)

print("Starting Year and Month: ")
```

```
## [1] "Starting Year and Month: "
```

```
start(df)
```

```
## [1] 1991    1
```

```
print("Final Year and Month: ")
```

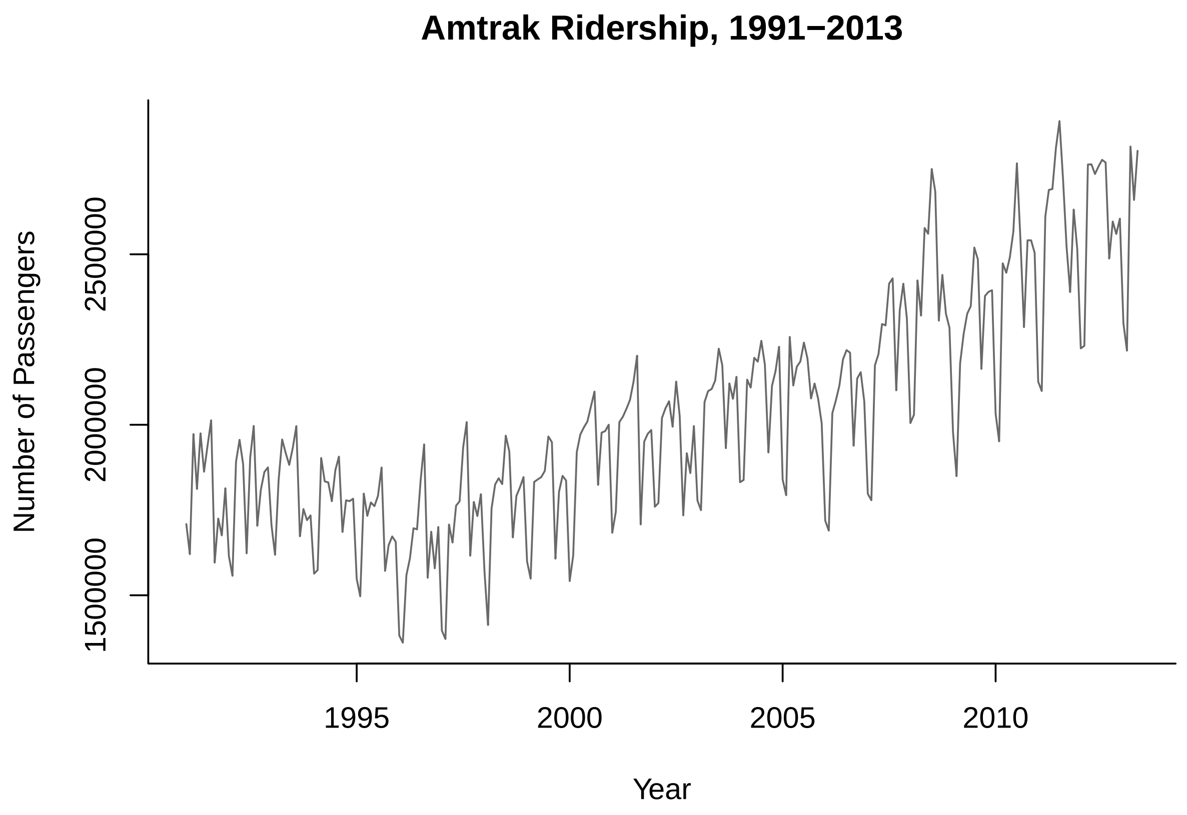
```
## [1] "Final Year and Month: "
```

```
end(df)
```

```
## [1] 2013    5
```

Code:

```
# Make a quick time-series plot
plot(df, xlab = "Year", ylab = "Number of Passengers",
     bty = "l", col = "grey41",
     main = "Amtrak Ridership, 1991-2013")
```



Analysis:

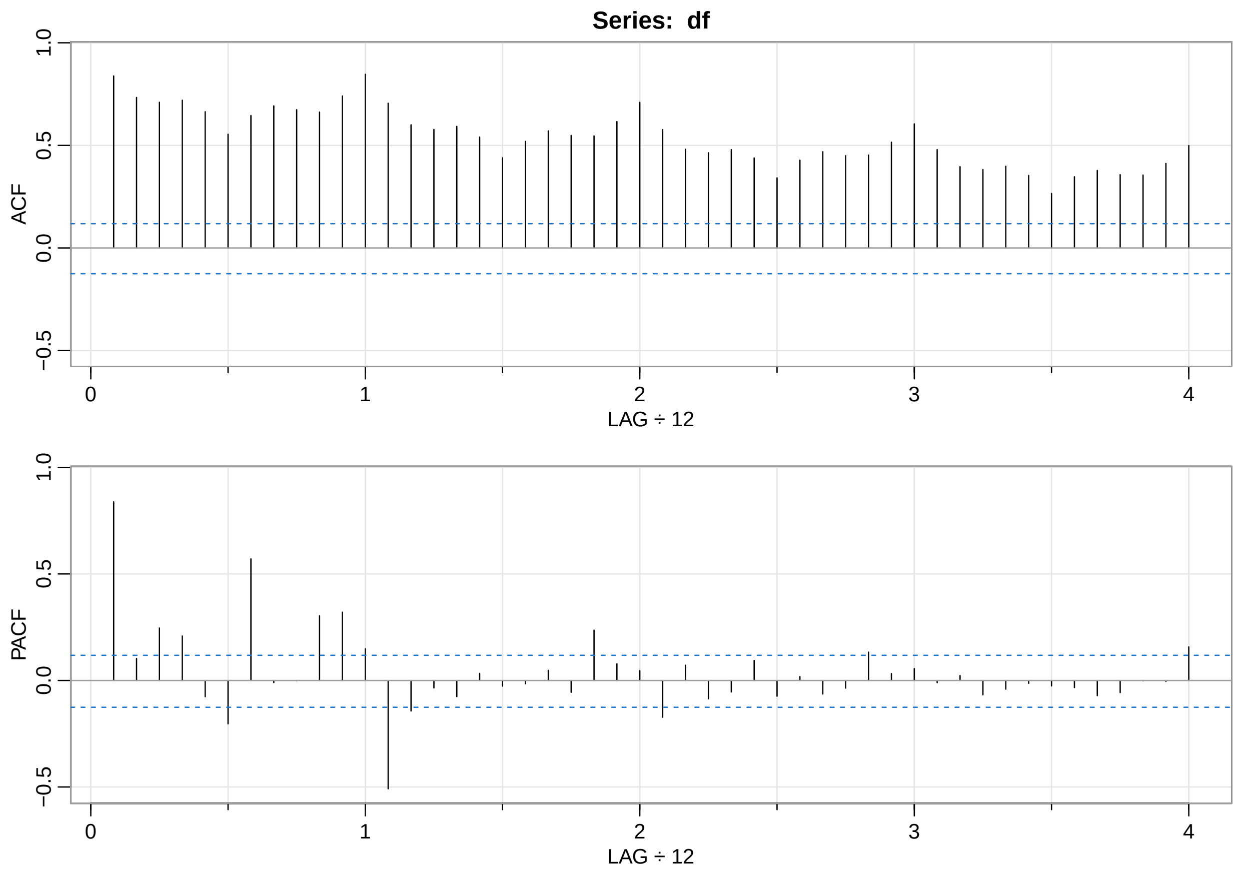
During exploratory data analysis, we were able to see that the trend of the data changed over time. It had signs of seasonality, showing it was not stationary.

ACF and PCF

Let's take a look at the auto-correlation and partial-correlation graphs for this time series data.

Code:

```
#Inspect ACF and PACF
acf_pcf <-acf2(df)
```



```
acf_pcf
```

```
##      [,1] [,2] [,3] [,4]  [,5]  [,6] [,7]  [,8] [,9] [,10] [,11] [,12] [,13]
## ACF  0.84 0.73 0.71 0.72  0.66  0.55 0.65  0.69 0.67  0.66  0.74  0.85  0.71
## PACF 0.84 0.10 0.25 0.21 -0.08 -0.20 0.57 -0.01 0.00  0.30  0.32  0.15 -0.51
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## ACF   0.60  0.58  0.59  0.54  0.44  0.52  0.57  0.55  0.55  0.62  0.71  0.58
## PACF -0.14 -0.04 -0.08  0.03 -0.03 -0.02  0.05 -0.06  0.24  0.08  0.05 -0.17
##      [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37]
## ACF   0.48  0.46  0.48  0.44  0.34  0.43  0.47  0.45  0.45  0.52  0.60  0.48
## PACF  0.07 -0.09 -0.05  0.09 -0.07  0.02 -0.06 -0.04  0.13  0.03  0.06 -0.01
##      [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48]
## ACF   0.40  0.38  0.40  0.35  0.27  0.35  0.38  0.36  0.36  0.41  0.50
## PACF  0.02 -0.07 -0.04 -0.01 -0.03 -0.03 -0.07 -0.06  0.00  0.00  0.16
```

Analysis:

The autocorrelation function shows high lag coefficients starting at lag 1, indicating seasonality.

Data Pre-processing

Smoothing Methods (Moving Average)

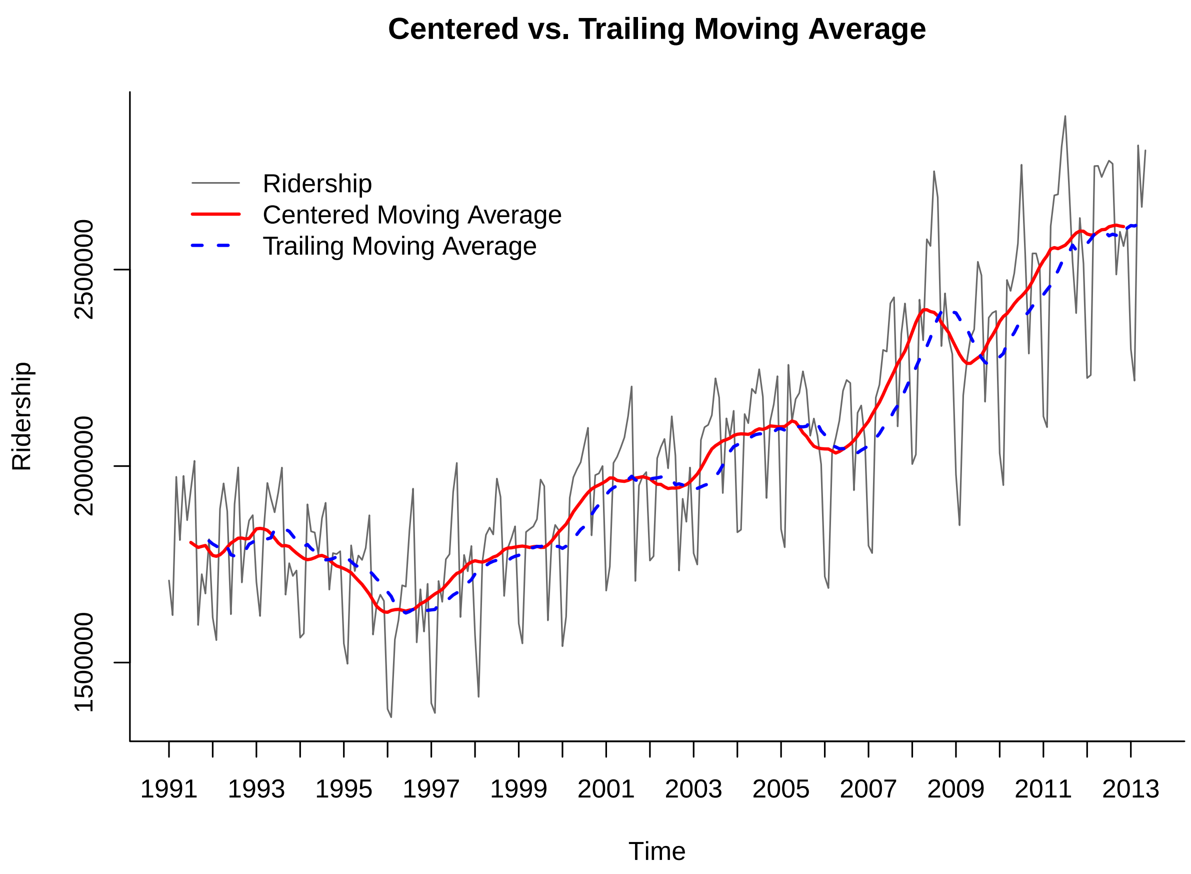
At the beginning of our data exploration, we were able to see there are components that change over time. Thus, we will need to look into data-driven methods because it deals data without a predetermined structure.

Moving Average

- This method is a simple smoother, it contains the average values across a time window, w, specified by the user. Two types of moving averages:
- a centered-moving average: useful for visualizing trends since averaging can suppress seasonality and noise
- a trailing moving average: useful for forecasting

Code:

```r
# Trailing Average
ma.trailing <- rollmean(df, k = 12, align = "right")

# Centered-Average
ma.centered <- ma(df, order = 12)

# Original Data
plot(df, ylab = "Ridership", xlab = "Time",
     bty = "l", xaxt = "n", col = 'grey41',
     main = "Centered vs. Trailing Moving Average")

# Labels
axis(1, at = seq(1991, 2013.50, 1), labels = format(seq(1991, 2013.50, 1)))

# Centered average lines
lines(ma.centered, lwd = 2, col = 'red')

# trailing moving average
lines(ma.trailing, lwd = 2, lty = 2, col ='blue')

# legend
legend(1991,2800000, c("Ridership","Centered Moving Average",
                       "Trailing Moving Average"), lty=c(1,1,2),
lwd=c(1,2,2), bty = "n", col = c("grey41", 'red', 'blue'))
```



Analysis:

Since the goal is to suppress seasonality in the data to visualize the trend, we should choose the length of a seasonal cycle. *The Amtrak ridership data indicates a choice of* $w = 12$*.*

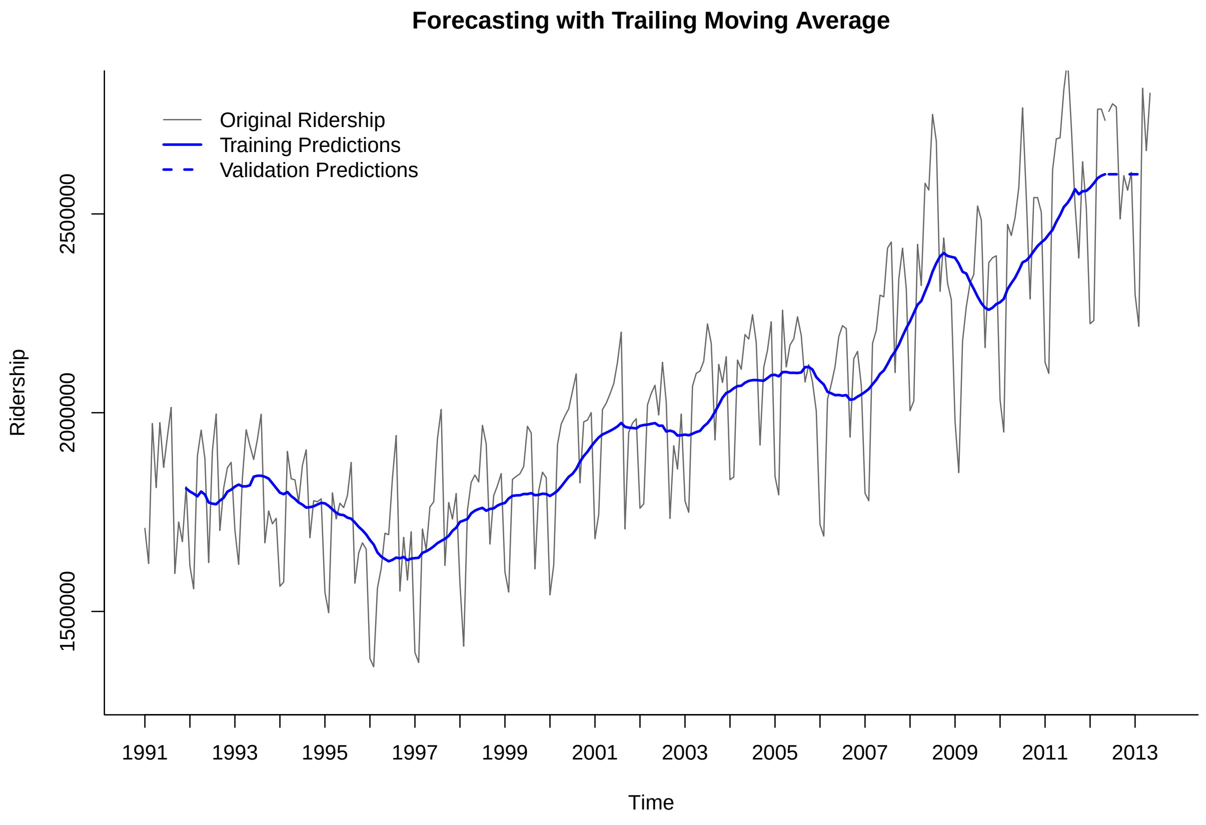
- This figure shows somewhat of a global U-shape, but the moving average looks to increase as the year passes.
- However, since centered moving averages uses data both in the past and future of a given time point, they cannot be used for forecasting because the future is typically unknown.

Trailing Moving Average

Therefore, trailing moving averages is the better approach here where the window of width is placed over the most recent available values.

```
# Validation Data
nValid <- 12

# number of training data
nTrain <- length(df) - nValid

# time window for training data
train.ts <- window(df, start = c(1991, 1), end = c(1991, nTrain))
```

```r
# time window for validation data
valid.ts <- window(df, start = c(1991, nTrain + 1), end = c(1991, nTrain + nValid))

# trailing moving average
ma.trailing <- rollmean(train.ts, k = 12, align = "right")

# last trailing moving average
last.ma <- tail(ma.trailing, 1)

# prediction by trailing moving average
ma.trailing.pred <- ts(rep(last.ma, nValid), start = c(1991, nTrain + 1),
end = c(1991, nTrain + nValid), freq = 12)

# plot training data
plot(train.ts, ylim = c(1300000, 2800000), ylab = "Ridership",
     xlab = "Time", bty="l", xaxt = "n", col = 'grey41',
xlim = c(1991,2013.50), main = "Forecasting with Trailing Moving Average")

# labels
axis(1, at = seq(1991, 2013.50, 1), labels = format(seq(1991, 2013.50, 1)))

# training model
lines(ma.trailing, lwd = 2, col = "blue")

# validation data
lines(valid.ts, col = 'grey41')

# predictions on validation
lines(ma.trailing.pred, lwd = 2, col = "blue", lty = 2)

# legend
legend(1991,2800000, c("Original Ridership","Training Predictions",
                       "Validation Predictions"), lty=c(1,1,2),
lwd=c(1,2,2), bty = "n", col = c("grey41", 'blue', 'blue'))
```



Analysis:

- The first thing to notice is that the forecasts for all the months in the validation period denoted in blue and blue dashes are identical because this method is not roll-forward next month forecasts. It is clear that the trailing moving average forecaster is inadequate for the Amtrak monthly forecast task. The reason why is because it does not capture the seasonality in the data. The forecaster predicted seasons with high ridership with lower ridership and seasons with low ridership with high ridership. This occurs because the moving average lags behind when forecasting a time series with a trend. Therefore, over-forecasting and under-forecasting in the presence of increasing and decreasing trends. So, between the smoothing methods of moving averages, it should only be use for forecasting when a series lack seasonality and trend, which is not true here for the Amtrak ridership data.
- However, there are other approaches for removing trends and seasonality, such as regression models or differencing.
- Then we can use the moving average to forecast a de-trended and de-seasonalized series.

Differencing

Differencing is a popular method for removing trend or seasonality patterns by taking the difference between two values in a series.

- For example, the lag-1 difference takes the difference between every two consecutive values $(y_t - y_{t-1})$.
- Meanwhile, differencing at lag-k means to subtract the value from k-periods back $(y_t - y_{t-k})$.

Dickey-Fuller Test

However, before using differencing as a pre-processing step, we should run a Dickey-Fuller test to see if differencing is actually needed. In other words, this test also check if the time series is stationary or not.

Code:

```
# Running the Dickey-Fuller Test on the original data without any pre-processing
adf.test(df)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df
## Dickey-Fuller = -3.8062, Lag order = 6, p-value = 0.01915
## alternative hypothesis: stationary
```

Analysis:

This may be a biased Dickey-Fuller test where we have a type 1 error. There is definitely visible seasonality happening in the data.
Since the test rejects the null hypothesis that the series is non-stationary, in this case the series was actually non-stationary. Therefore, we will ignore that it ever happened because we will need to perform a second order difference due to a trend and seasonality pattern in the series.

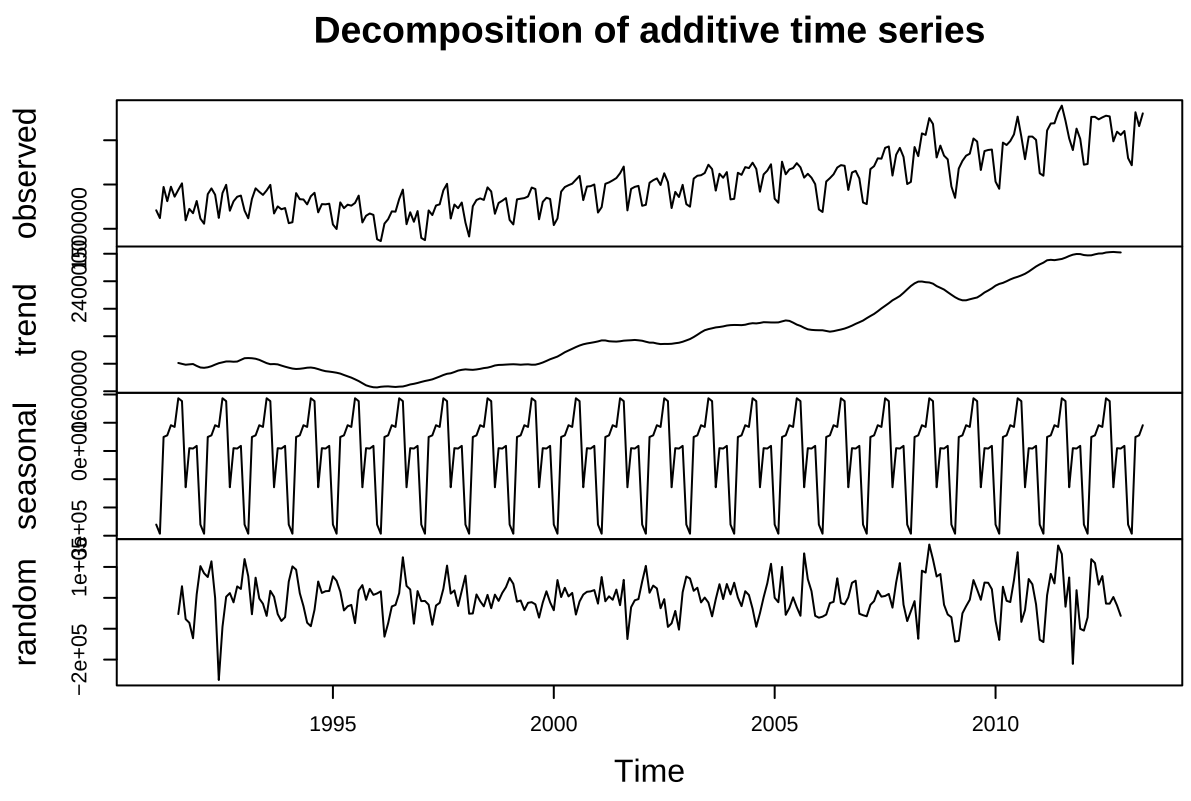
- Alternative approaches without differencing can be aggregating the monthly data into a coarser level such as yearly ridership as the total sum instead.

Other tests for Stationarity

Once again, we can visually inspect the data by decomposing it into different time series components, such as the seasonality and trend.

Code:

```
# decompose the components of the data set
d1 <-decompose(df, type = c("additive","multiplicative"))
plot(d1)
```



Analysis:

There is a strong seasonal component in the data set, there is a consistent upward trend in the dataset.

Detrending

Detrending can be used by the lag-1 difference of a series. This would remove the somewhat U-shape of Amtrak ridership series. An advantage of difference is there are no assumptions that the trend is global.

- For quadratic or exponential trends, one more step of lag-1 differencing must be applied to remove the trend.

Deseasonalizing

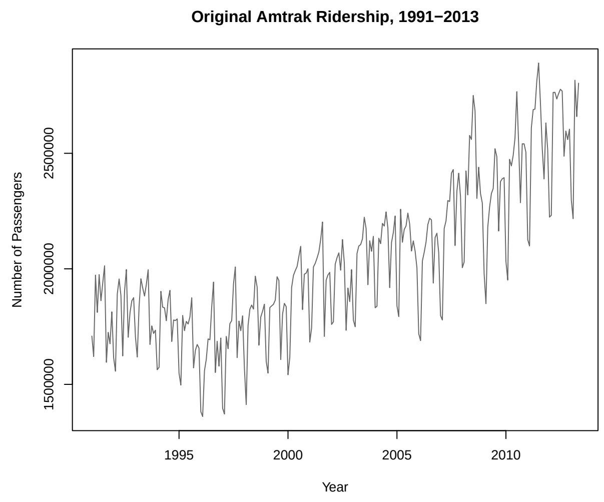
We can remove the seasonality of the Amtrak ridership data by using a lag-12 difference series. THis will remove the monthly pattern

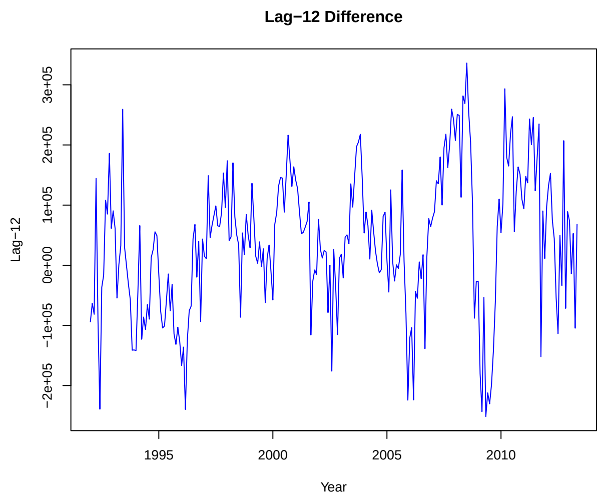
Removing seasonality and trend

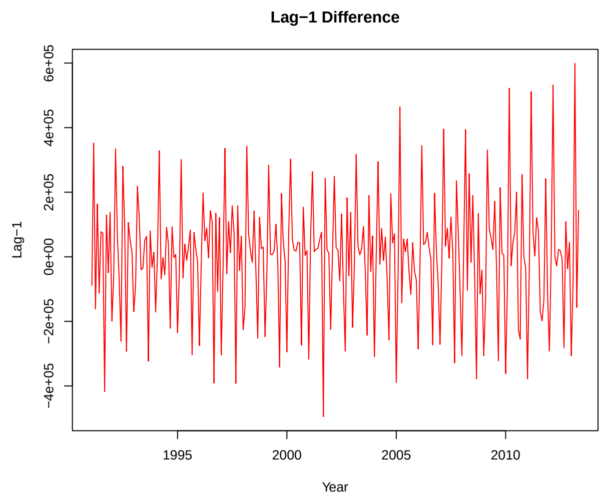
- When both of these components exist, we can apply differencing twice to the series.
- Since the Amtrak ridership data has both trend and seasonality, we will perform the doube differencing method in order to de-trend and deseasonalize it.

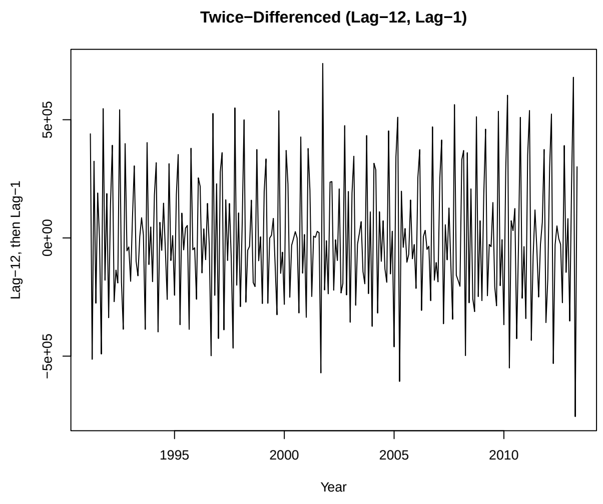
Code:

```r
par(mfrow=c(2,2))

# Original
plot(df, xlab = "Year", ylab = "Number of Passengers",
     main = "Original Amtrak Ridership, 1991-2013", col = 'grey41')

# lag-12 difference
plot(diff(df, lag = 12), xlab = "Year", ylab = "Lag-12",
     main = "Lag-12 Difference", col = 'blue')

# lag-1 difference
plot(diff(df), xlab = "Year", ylab = "Lag-1",
     main = "Lag-1 Difference", col = 'red')


# Double Differencing
plot(diff(diff(df, s = 12)), xlab = "Year", ylab = "Lag-12, then Lag-1",
     main = "Twice-Differenced (Lag-12, Lag-1)", col = 1)
```









Analysis:

- The lag-1 difference plot on the bottom left contains no visible trend compared to the original series above.
- If we are dealing with daily data ridership, then we could remove a seasonal pattern of lag-7 differences. However, since we are have monthly data, we are using a lag-12 difference series as shown in the top right figure where the monthly pattern is absent.
- Lastly, since there are both a seasonality and trend, the double differencing effect on the bottom right panel is a series without trend or monthly seasonality.

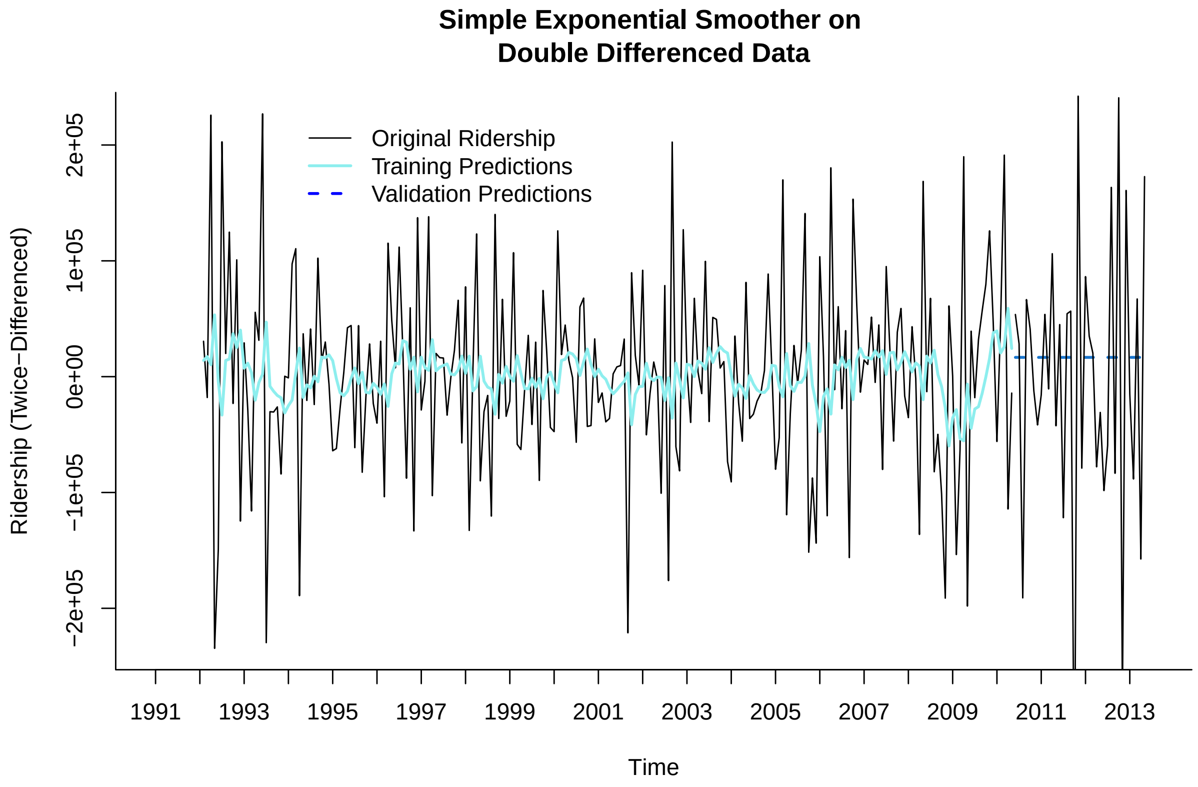
Data Modeling

Simple Exponential Smoothing

Exponential smoothing works very similar to forecasting with a moving average, except it takes the weighted average over all the past values of a series. By doing so, the weights will decrease exponentially into the past. This is valuable because we give weight to recent information more than the older information. This method is also very popular due to its low computation costs, easy automation, and good performance. However, it is important to note that using exponential smoothing for forecasting assumes no trend or seasonality in a series. So the idea is similar to before with moving averages, by first removing the trend and seasonality, then apply the exponential smoothing forecaster.

Code:

```r
# remove trend and seasonality by doing double-differencing
diff_twice <- diff(diff(df, lag = 12), lag = 1)

# Number of validation data
nValid <- 36

# number of training data
nTrain <- length(diff_twice) - nValid

# specified time window for training data
train.ts <- window(diff_twice, start = c(1992, 2), end = c(1992, nTrain + 1))

# specified time window for validation data
valid.ts <- window(diff_twice, start = c(1992, nTrain + 2),
                   end = c(1992, nTrain + 1 + nValid))

# Additive, no trend, no seasonality model using a constant (learning rate) of 0.2
ses <- ets(train.ts, model = "ANN", alpha = 0.2)

# make predictions using model
ses.pred <- forecast(ses, h = nValid, level = 0)

# training data then validation forecasts
plot(ses.pred, ylab = "Ridership (Twice-Differenced)", xlab = "Time",
bty = "l", xaxt = "n", xlim = c(1991,2013.50), main = "Simple Exponential Smoother on
Double Differenced Data", flty = 2)

# labels
axis(1, at = seq(1991, 2013, 1), labels = format(seq(1991, 2013, 1)))

# training model - predictions
lines(ses.pred$fitted, lwd = 2, col = "darkslategray2")

# validaiton data
lines(valid.ts)

# legend
legend(1994,230000, c("Original Ridership","Training Predictions",
                        "Validation Predictions"), lty=c(1,1,2),
lwd=c(1,2,2), bty = "n", col = c("black", 'darkslategray2', 'blue'))
```



Analysis:

For forecasting with simple exponential smoothing, the data trained on was the double-difference ridership data that contains no seasonality or trend. Then we fit the simple exponential smoothing model to the training data set with $\alpha = 0.2$ as default. This smoothing model is under the *ets* framework using the model with additive (A), no trend (N), and no seasonality (N), denoted as "ANN". The forecasts for the validation set for each month remained as the same value similar to the moving average forecast model. This would mean that the simple exponential forecaster or ANN model is also inadequate for the monthly forecasting task. The reason why is because it also does not capture the seasonality in the data.

- The simple exponential smoothing models did a poor job at forecasting this time series without trend or seasonality.
- Another solution is to use a more complex and sophisticated exponential smoothing that is able to model data with both trend and seasonality.

Additive Trends

Double exponential smoothing can be used on a series that contain an additive trend. This is also called the Holt's linear trend model. The local trend is estimated and is updated as more data comes in. The equation is specified by $F_{t+k} = L_t + kT_t$, where the k-step-ahead forecast is a combination of the level estimate at time t $L(t)$ and the trend estimate at time t (Tt). There is also two smoothing constants α and β which determine the rate of learning and can be constants between 0 and 1 set by the user (Higher values = faster learning). However, there are two types of errors in an exponential model:

- Additive error (additive trend): This is where errors are assumed to have a fixed magnitude, meaning the forecasts contain not only the level + trend but also an additional error. $y_{t+1} = L_t + T_t + e_t$
- Multiplicative error (additive trend): This is where the size of the error grows as the level of the series increase, or the error is a percentage increase in the current level plus trend. $y_{t+1} = (L_t + T_t) * (1 + e_t)$

Multiplicative trends

Although additive trend models assumes the level changes from one period to the next by a fixed amount, multiplicative trends assumes it changes by a factor instead. Thus, the formula is different specified by: $F_{t+k} = L_t * T_t^k$

- For the additive error, it will be $y_{t+1} = L_t * T_t + e_t$ for the multiplicative trend model instead.
- While the multiplicative error stays the same as $y_{t+1} = (L_t + T_t) * (1 + e_t)$

Exponential smoothing with both a trend and seasonality

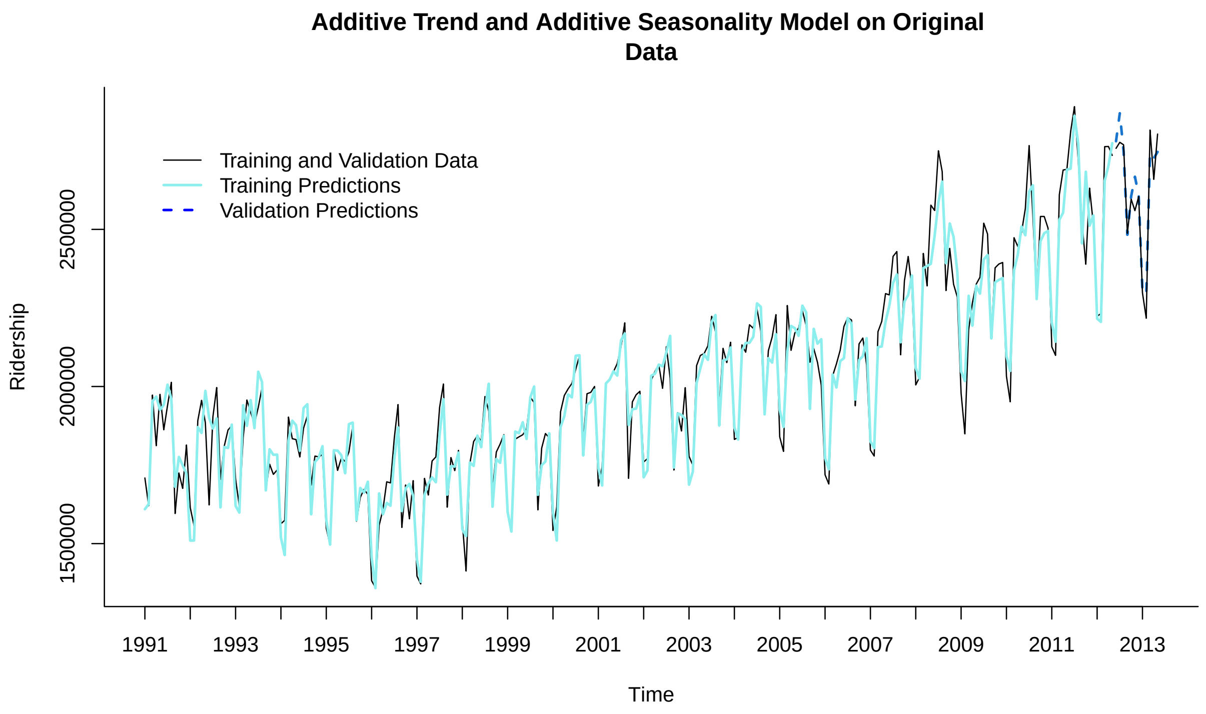
A further extension of the double exponential smoothing where the k-step-ahead forecasts also takes into consideration the seasonality of the current period. While the trend is considered from the additive and multiplicative seasonality. - This is specified as: $F_{t+k} = (L_t + kT_t)S_{t+k-M}$ where M denotes the number of seasons in a series (e.g., M = 12 for monthly seasonality). This is an adaptive method that allows the components (levels, trends, and seasonality) to change over time.

In this case we would make **June 2012 to May 2013** the validation period.

Code:

```r
# Validation Data
nValid <- 12

# number of training data
nTrain <- length(df) - nValid

# time window for training data
train.ts <- window(df, start = c(1991, 1), end = c(1991, nTrain))

# time window for validation data
valid.ts <- window(df, start = c(1991, nTrain + 1), end = c(1991, nTrain + nValid))

# multiplicative error with additive trend and additive seasonality model
maa <- ets(train.ts, model = "MAA")

# Forecasts
maa.pred <- forecast(maa, h = nValid, level = 0)
```

```
# training data and validation data - forecasts
plot(maa.pred, ylab = "Ridership", xlab = "Time",
main = "Additive Trend and Additive Seasonality Model on Original
Data", flty = 2, bty="l", xaxt = "n")

# labels
axis(1, at = seq(1991, 2013, 1), labels = format(seq(1991, 2013, 1)))

# training data - forecasts
lines(maa.pred$fitted, lwd = 2, col = "darkslategray2")

# validation data only
lines(valid.ts)

# legend
legend(1991,2800000, c("Training and Validation Data","Training Predictions",
                       "Validation Predictions"), lty=c(1,1,2),
lwd=c(1,2,2), bty = "n", col = c("black", 'darkslategray2', 'blue'))
```



```
## ETS(M,Ad,A)
##
## Call:
##  ets(y = train.ts, model = "MAA")
##
##   Smoothing parameters:
##     alpha = 0.5146
##     beta  = 1e-04
##     gamma = 0.2007
##     phi   = 0.8692
##
##   Initial states:
##     l = 1868559.4037
##     b = -3079.4402
##     s = 23126.2 11233.09 11079.87 -128642.4 177366.8 187591.1
##            81511.92 89042.49 49616.47 43289.88 -288877.1 -256338.3
##
##   sigma:  0.0357
##
##      AIC     AICc      BIC
## 7181.096 7183.970 7244.979
```

Analysis:

This approach uses Holt-Winter's method to build an additive trend and seasonality model with multiplicative error on the original data without a second order difference. This falls under the ets framework using the "MAA" model. However, what if the choice was to use an multiplicative trend and seasonality model, and so forth. There is an automated approach for model selection. In this case we would make **June 2012 to May 2013** the validation period.

Automated Exponential Model Sections

By leaving out the model parameter inside the ets function, this will fit several different models and choose the best one based on the lowest AIC score.

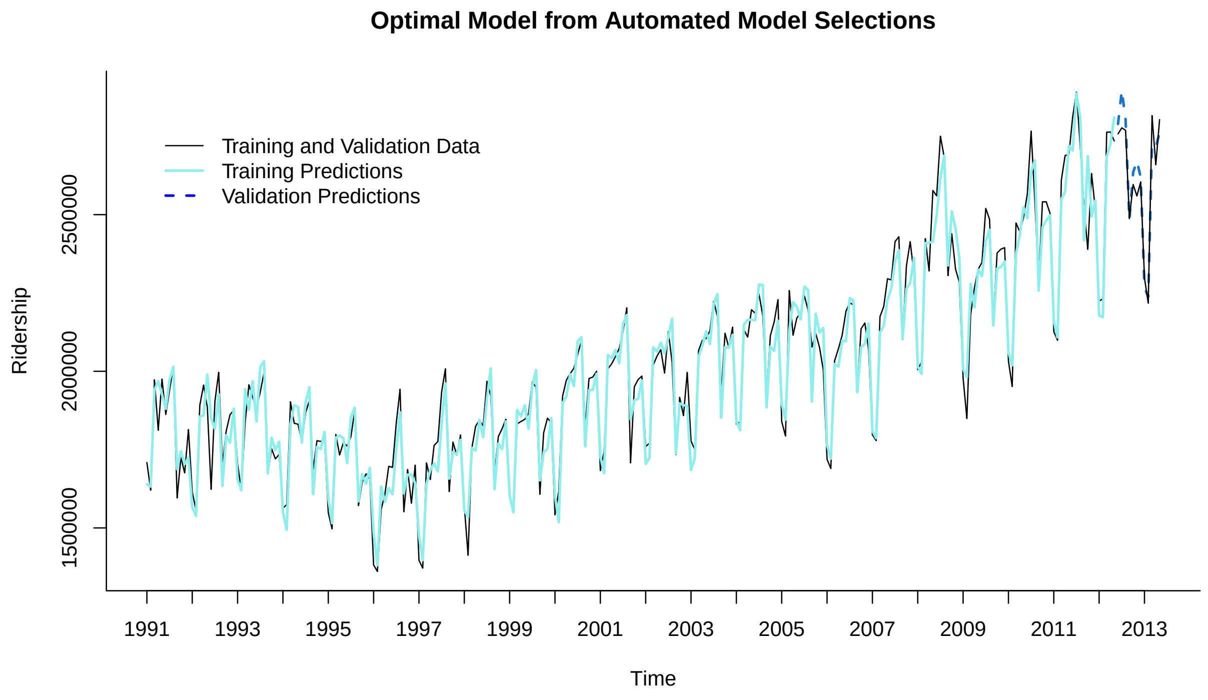
Code:

```
# Validation Data
nValid <- 12

# number of training data
nTrain <- length(df) - nValid

# time window for training data
train.ts <- window(df, start = c(1991, 1), end = c(1991, nTrain))

# time window for validation data
valid.ts <- window(df, start = c(1991, nTrain + 1), end = c(1991, nTrain + nValid))

#automated selection - optimal model
optimal <- ets(train.ts, restrict = FALSE, allow.multiplicative.trend = TRUE)
optimal
```

```
## ETS(M,Ad,M)
##
## Call:
##  ets(y = train.ts, restrict = FALSE, allow.multiplicative.trend = TRUE)
##
##   Smoothing parameters:
##     alpha = 0.5545
##     beta  = 6e-04
##     gamma = 0.1467
##     phi   = 0.98
##
##   Initial states:
##     l = 1860206.6141
##     b = -2853.0529
##     s = 1.0049 0.9822 0.9979 0.9348 1.1141 1.0792
##            1.0222 1.0606 1.0344 1.0276 0.8596 0.8826
##
##   sigma:  0.0339
##
##      AIC     AICc      BIC
## 7155.162 7158.036 7219.045
```

```
# Optimal Forecasts
op.pred <- forecast(optimal, h = nValid, level = 0)

# training data - then validation forecasts
plot(op.pred, ylab = "Ridership", xlab = "Time",
     bty = "l", xaxt = "n",
main = "Optimal Model from Automated Model Selections", flty = 2)

# labels
```

```
axis(1, at = seq(1991, 2013, 1), labels = format(seq(1991, 2013, 1)))

# training data - forecasts
lines(op.pred$fitted, lwd = 2, col = "darkslategray2")

# validation data
lines(valid.ts)

# legend
legend(1991,2800000, c("Training and Validation Data","Training Predictions",
                       "Validation Predictions"), lty=c(1,1,2),
lwd=c(1,2,2), bty = "n", col = c("black", 'darkslategray2', 'blue'))
```



Analysis:

The optimal model chosen was a multiplicative error, additive trend, and multiplicative seasonality with an improved AIC score of 7155.16 which was lower than the previous ANN model with an AIC of 7181.09

Regression-Based Models

There are different types of common trends and seasonality that can be modeled by regression-based models estimated during the training period and used to forecast on future data. Such trends include linear, exponential, or polynomial, while the different seasonality are additive and multiplicative seasons.

- Linear trends = a series is increasing/decreasing linearly over time
- Quadratic/polynomials functions = These are used to capture the more complex trends.

Linear Trend

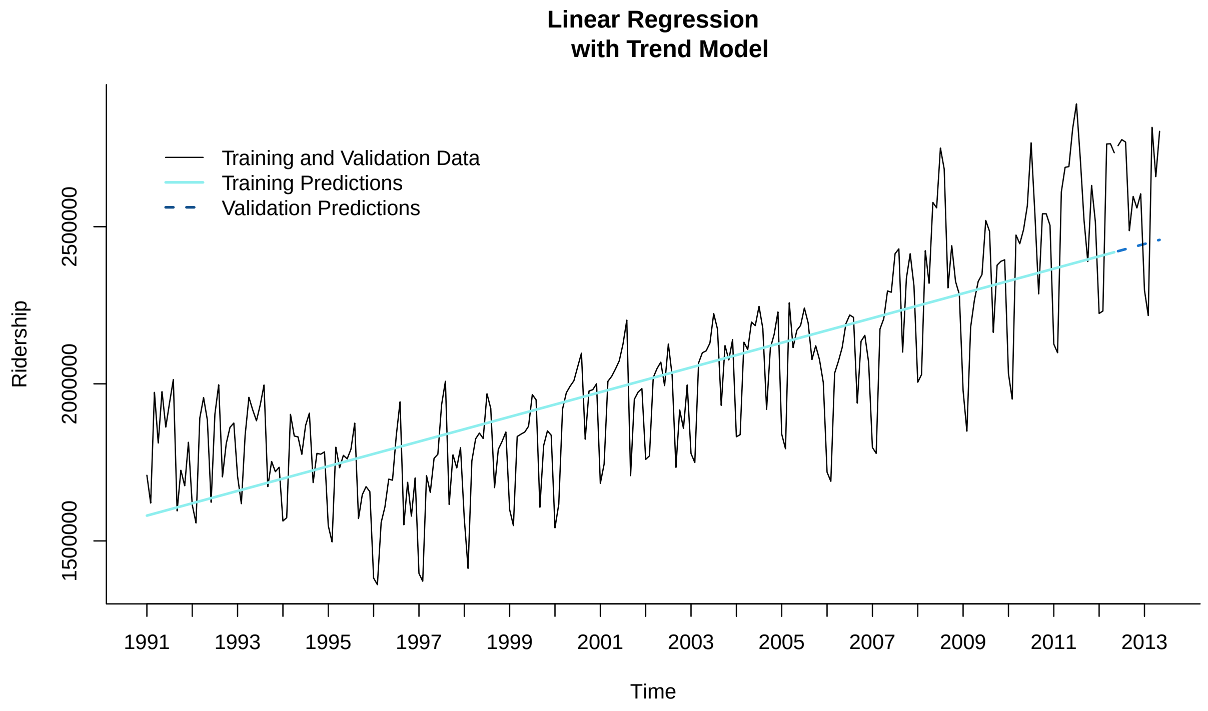
We can start with creating a linear regression model using a predictor index by time and the output, y, which is the number of ridership each month. Keep in mind that a linear regression model will capture the global linear trend in a time series.

However, before we move on we will need to partition the series into training and validation periods. In this case we would make **June 2012 to May 2013** the validation period. This will allow us to keep 12 months in the validation set to provide monthly forecasts for the following year and evaluate each forecasts individually from their respective years.

Code:

```r
# Validation Data
nValid <- 12

# number of training data
nTrain <- length(df) - nValid

# time window for training data
train.ts <- window(df, start = c(1991, 1), end = c(1991, nTrain))

# time window for validation data
valid.ts <- window(df, start = c(1991, nTrain + 1), end = c(1991, nTrain + nValid))

# trend of data
trend <- time(df)

# linear regression model
train.lm <- tslm(train.ts ~ trend)


# validation forecasts
train.lm.pred <- forecast(train.lm, h = nValid, level = 0)

# Plot training data along with validation forecasts
plot(train.lm.pred, ylab = "Ridership", xlab = "Time",
     bty = "l", xaxt = "n", main = "Linear Regression
     with Trend Model", flty = 2)

# labels
axis(1, at = seq(1991, 2013.9, 1), labels = format(seq(1991, 2013.9, 1)))

# validation data
lines(valid.ts)
```

```r
# training model forecasts
lines(train.lm.pred$fitted, lwd = 2, col = "darkslategray2")

# legend
legend(1991,2800000, c("Training and Validation Data","Training Predictions",
                        "Validation Predictions"), lty=c(1,1,2),
lwd=c(1,2,2), bty = "n", col = c("black", 'darkslategray2', 'dodgerblue4'))
```



```r
summary(train.lm)
```

```
##
## Call:
## tslm(formula = train.ts ~ trend)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -483625 -116651   24726  119568  504688
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1577216.9    24624.3   64.05   <2e-16 ***
## trend          3274.7      165.5   19.79   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 196800 on 255 degrees of freedom
## Multiple R-squared:  0.6057, Adjusted R-squared:  0.6041
## F-statistic: 391.6 on 1 and 255 DF,  p-value: < 2.2e-16
```

```
print(paste0("AIC: ", round(AIC(train.lm),2)))
```

```
## [1] "AIC: 6998.97"
```

Analysis:

This linear trend does not fit the global trend at all. The global trend in fact is not linear. This is why we will consider other models for this series. Also the reason why this linear trend model performed poorly is because we did not model seasonality.

Model with Trend and Seasonality

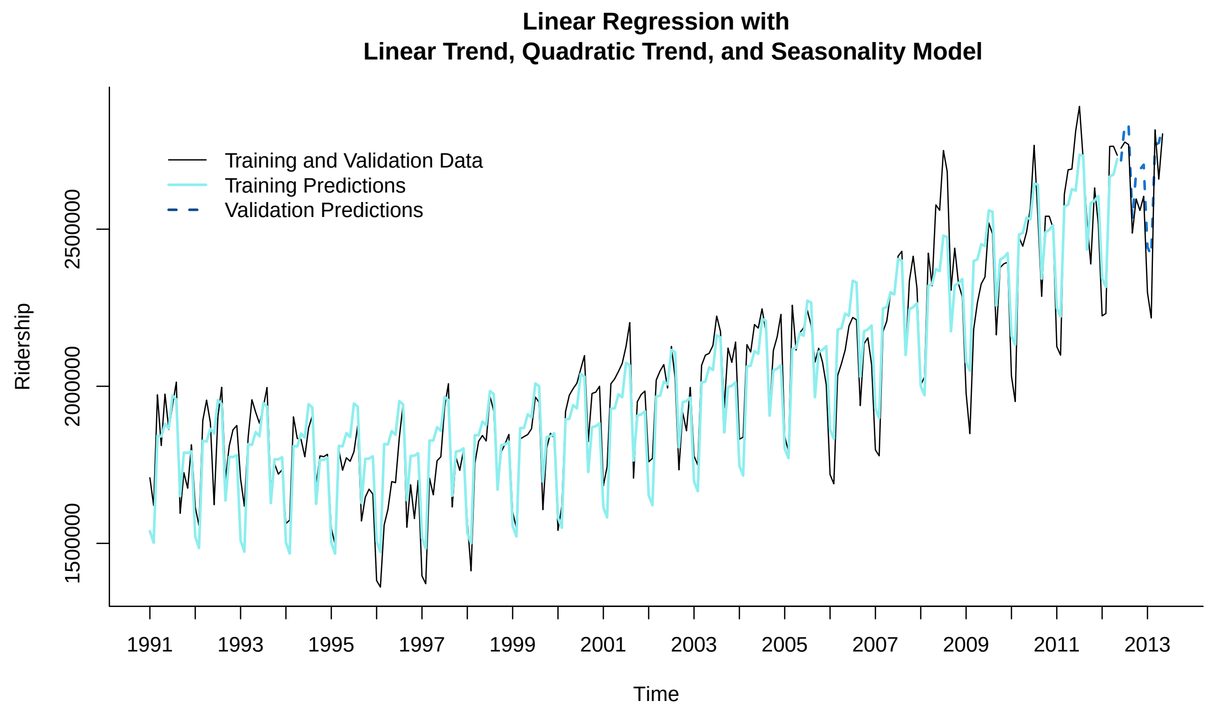
We can model a series with both trend and seasonality by adding predictors of both types. For example, we can build a new model with both linear and quadratic trend. Then we can add a monthly seasonality which was a factor of 12, however it is redundant to use all 12. In this case, except season 1 which was January. In total, we will have 13 predictors, with $tandt^2$ for the trends and the 11 dummy variables for months.

Code:

```
# Validation Data
nValid <- 12

# number of training data
nTrain <- length(df) - nValid

# time window for training data
train.ts <- window(df, start = c(1991, 1), end = c(1991, nTrain))

# time window for validation data
valid.ts <- window(df, start = c(1991, nTrain + 1), end = c(1991, nTrain + nValid))

# trend of data
trend <- time(df)

# seasonality
season <- factor(cycle(df))

# linear model with linear trend, quadratic trend and seasonality
model.lm <- tslm(train.ts ~ trend + I(trend^2) +  season)
model.lm
```

```
##
## Call:
## tslm(formula = train.ts ~ trend + I(trend^2) + season)
##
## Coefficients:
## (Intercept)        trend   I(trend^2)      season2      season3      season4
##  1540670.38     -1732.47        19.39    -35131.64    307758.22    305931.76
##     season5      season6      season7      season8      season9     season10
##   347646.93    334705.28    441363.24    430168.00    123687.04    263579.06
##    season11     season12
##   263809.12    270626.31
```

```r
# validation forecasts
model.lm.pred <- forecast(model.lm, h = nValid, level = 0)

# Plot training data along with validation forecasts
plot(model.lm.pred, ylab = "Ridership", xlab = "Time",
     bty = "l", xaxt = "n", flty = 2,
     main = "Linear Regression with
     Linear Trend, Quadratic Trend, and Seasonality Model")

# labels
axis(1, at = seq(1991, 2013.9, 1), labels = format(seq(1991, 2013.9, 1)))

# validation data
lines(valid.ts)

# training model forecasts
lines(model.lm.pred$fitted, lwd = 2, col = "darkslategray2")

# legend
legend(1991,2800000, c("Training and Validation Data","Training Predictions",
                       "Validation Predictions"), lty=c(1,1,2),
lwd=c(1,2,2), bty = "n", col = c("black", 'darkslategray2', 'dodgerblue4'))
```



```r
# summary of model
summary(model.lm)
```

```
##
```

```
## Call:
## tslm(formula = train.ts ~ trend + I(trend^2) + season)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -257653  -64600    1511   67372  270695
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.541e+06  2.630e+04  58.575  < 2e-16 ***
## trend       -1.732e+03  3.229e+02  -5.365 1.89e-07 ***
## I(trend^2)   1.939e+01  1.212e+00  15.991  < 2e-16 ***
## season2     -3.513e+04  2.881e+04  -1.220    0.224
## season3      3.078e+05  2.881e+04  10.683  < 2e-16 ***
## season4      3.059e+05  2.881e+04  10.619  < 2e-16 ***
## season5      3.476e+05  2.881e+04  12.067  < 2e-16 ***
## season6      3.347e+05  2.916e+04  11.480  < 2e-16 ***
## season7      4.414e+05  2.916e+04  15.138  < 2e-16 ***
## season8      4.302e+05  2.916e+04  14.754  < 2e-16 ***
## season9      1.237e+05  2.916e+04   4.242 3.15e-05 ***
## season10     2.636e+05  2.916e+04   9.040  < 2e-16 ***
## season11     2.638e+05  2.916e+04   9.048  < 2e-16 ***
## season12     2.706e+05  2.916e+04   9.281  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 95550 on 243 degrees of freedom
## Multiple R-squared:  0.9114, Adjusted R-squared:  0.9067
## F-statistic: 192.4 on 13 and 243 DF,  p-value: < 2.2e-16
```

```
print(paste0("AIC: ", round(AIC(model.lm),2)))
```

```
## [1] "AIC: 6639.16"
```

```
accuracy(model.lm.pred, valid.ts)
```

```
##                          ME      RMSE      MAE        MPE     MAPE      MASE
## Training set  1.806244e-12  92906.36 76159.61 -0.2116348 3.915444 0.7704515
## Test set     -7.274081e+04 100660.82 87295.84 -2.9639564 3.485683 0.8831086
##                    ACF1 Theil's U
## Training set  0.6770800        NA
## Test set     -0.1252874  0.424056
```

Analysis:

The AIC score has dropped significantly compared to the model with only linear trend. In comparison with the exponential smoothing model, this score has dropped drastically, making the linear, quadratic, and monthly seasonality model the best.

AR1 Model

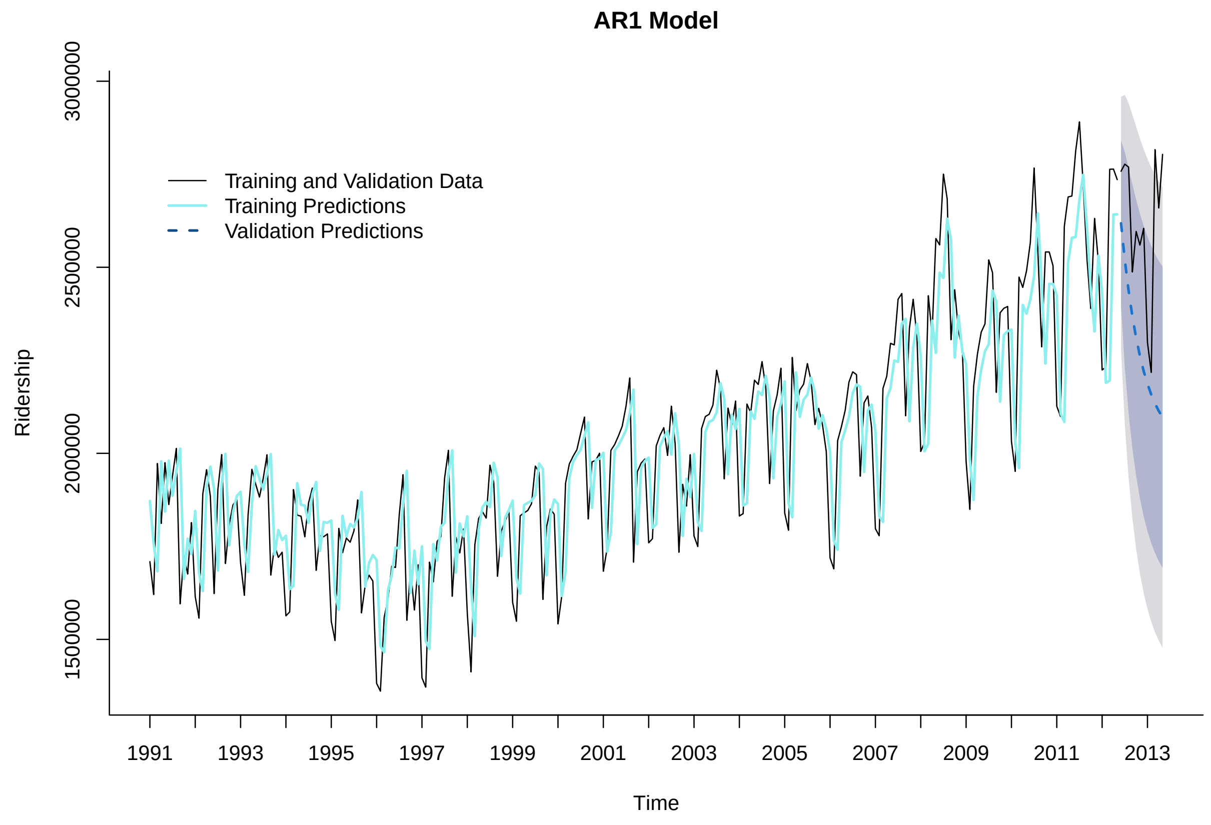
The following is an Auto-regression model

Code:

```r
# Validation Data
nValid <- 12

# number of training data
nTrain <- length(df) - nValid

# time window for training data
train.ts <- window(df, start = c(1991, 1), end = c(1991, nTrain))

# time window for validation data
valid.ts <- window(df, start = c(1991, nTrain + 1), end = c(1991, nTrain + nValid))

#create basic AR model
ar1 = arima(train.ts, order=c(1,0,0))
ar1
```

```
##
## Call:
## arima(x = train.ts, order = c(1, 0, 0))
##
## Coefficients:
##          ar1   intercept
##       0.8388  2008415.12
## s.e.  0.0346    65740.01
##
## sigma^2 estimated as 3.002e+10:  log likelihood = -3465.35,  aic = 6936.7
```

```r
#forecast ar1 model
ar1.pred <- forecast(ar1, h = nValid)

# Plot training data along with validation forecasts
plot(ar1.pred, ylab = "Ridership", xlab = "Time",
     bty = "l", xaxt = "n", flty = 2,
     main = "AR1 Model")

# labels
axis(1, at = seq(1991, 2013.9, 1), labels = format(seq(1991, 2013.9, 1)))

# validation data
lines(valid.ts)

# training model forecasts
lines(ar1.pred$fitted, lwd = 2, col = "darkslategray2")

# legend
legend(1991,2800000, c("Training and Validation Data","Training Predictions",
                       "Validation Predictions"), lty=c(1,1,2),
lwd=c(1,2,2), bty = "n", col = c("black", 'darkslategray2', 'dodgerblue4'))
```



```r
#summary
summary(ar1)
```

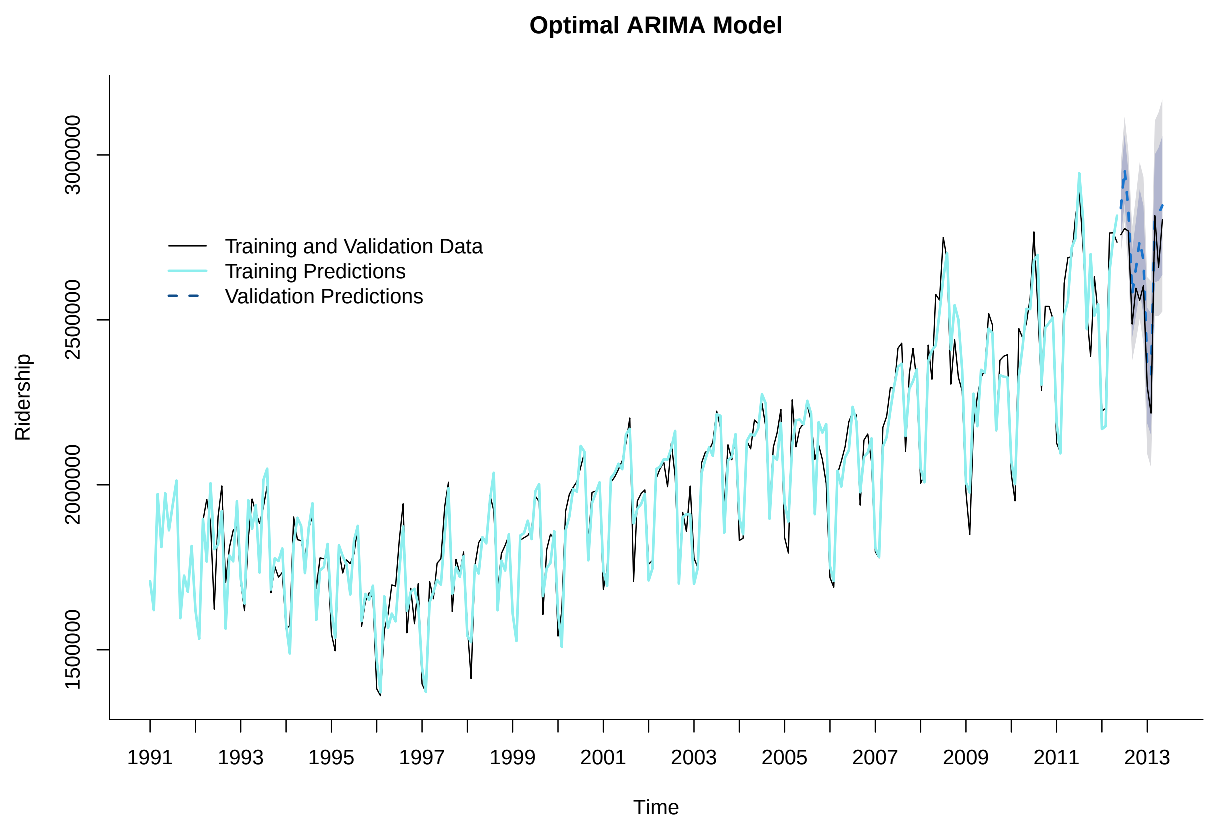
```
##
## Call:
## arima(x = train.ts, order = c(1, 0, 0))
##
## Coefficients:
##          ar1   intercept
##       0.8388  2008415.12
## s.e.  0.0346    65740.01
##
## sigma^2 estimated as 3.002e+10:  log likelihood = -3465.35,  aic = 6936.7
##
## Training set error measures:
##                    ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 1492.036 173259.5 129640.3 -0.6899507 6.683191 0.9819261
##                    ACF1
## Training set -0.0747508
```

Autoselection for Optimal ARIMA Parameters

We used the auto-arima function to get the best parameters for an optimal ARIMA model

Code:

```
#auto arima
# Validation Data
nValid <- 12

# number of training data
nTrain <- length(df) - nValid

# time window for training data
train.ts <- window(df, start = c(1991, 1), end = c(1991, nTrain))

# time window for validation data
valid.ts <- window(df, start = c(1991, nTrain + 1), end = c(1991, nTrain + nValid))

# autoselection for optimal arima parameters
autoarima <- auto.arima(train.ts)

#forecast ar1 model
autoarima.pred <- forecast(autoarima, h = nValid)

# Plot training data along with validation forecasts
plot(autoarima.pred, ylab = "Ridership", xlab = "Time",
     bty = "l", xaxt = "n", flty = 2,
     main = "Optimal ARIMA Model")

# labels
axis(1, at = seq(1991, 2013.9, 1), labels = format(seq(1991, 2013.9, 1)))

# validation data
lines(valid.ts)

# training model forecasts
lines(autoarima.pred$fitted, lwd = 2, col = "darkslategray2")

# legend
legend(1991,2800000, c("Training and Validation Data","Training Predictions",
                       "Validation Predictions"), lty=c(1,1,2),
lwd=c(1,2,2), bty = "n", col = c("black", 'darkslategray2', 'dodgerblue4'))
```



```r
#summary
summary(autoarima)
```

```
## Series: train.ts
## ARIMA(2,1,1)(2,1,1)[12]
##
## Coefficients:
##           ar1      ar2     ma1    sar1    sar2     sma1
##       -0.5596  -0.2037  0.1352  0.0763  0.0017  -0.7054
## s.e.   0.4954   0.1855  0.5050  0.1184  0.0952   0.0976
##
## sigma^2 estimated as 4.92e+09:  log likelihood=-3069.4
## AIC=6152.79   AICc=6153.27   BIC=6177.27
##
## Training set error measures:
##                    ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 2825.372 67501.85 50835.07 0.06815184 2.555057 0.5142615
##                      ACF1
## Training set -0.005063544
```

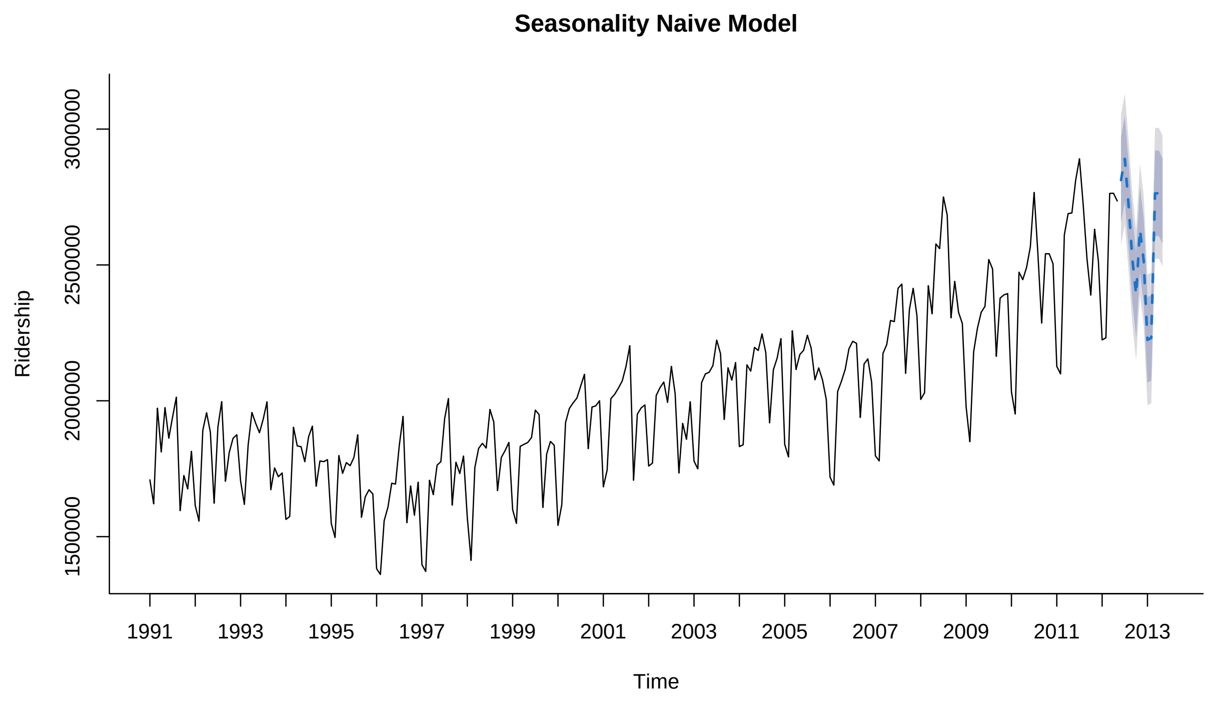
Naive Forecasts

Instead of sophisticated modeling, naive forecasts are basically the most recent values of the ridership data. While a seasonal naive forecast is from the recent value from the most identical season. For example, if we are forecasting August 2012, the forecasts for this month would be the value from August 2011 instead.

Ironically, naive forecasts are actually good at achieving great performance even though its easy to understand and deploy. Also, this naive model should be used as a baseline to compare other model's predictive performance against.

Code:

```r
# Seasonality naive forecasts
snaive.pred <- snaive(train.ts, h = nValid)

# Plot training data along with validation forecasts
plot(snaive.pred, ylab = "Ridership", xlab = "Time",
     bty = "l", xaxt = "n", flty = 2,
     main = "Seasonality Naive Model")

# labels
axis(1, at = seq(1991, 2013.9, 1), labels = format(seq(1991, 2013.9, 1)))
```



Results and Model Selection

Code:

Table 2: Optimal Exponential Model

	ME	RMSE	MAE	MPE	MAPE	MASE
Training set	6027	66103	52183	0.2028	2.624	0.5279
Test set	-21561	61488	49641	-0.824	1.866	0.5022

Table 3: Regression with Monthly Seasonality and Linear and Quadratic Trends Model

	ME	RMSE	MAE	MPE	MAPE	MASE
Training set	1.806e-12	92906	76160	-0.2116	3.915	0.7705
Test set	-72741	100661	87296	-2.964	3.486	0.8831

Table 4: AR1 Model

	ME	RMSE	MAE	MPE	MAPE	MASE
Training set	1492	173260	129640	-0.69	6.683	1.311
Test set	326770	388025	326770	12.16	12.16	3.306

Table 5: Optimal ARIMA Model

	ME	RMSE	MAE	MPE	MAPE	MASE
Training set	2825	67502	50835	0.06815	2.555	0.5143
Test set	-92458	107742	93867	-3.587	3.637	0.9496

Table 6: Seasonal Naive Model

	ME	RMSE	MAE	MPE	MAPE	MASE
Training set	38708	122794	98851	1.595	4.854	1
Test set	12386	91056	77756	0.4937	2.959	0.7866

Analysis:

The key performance metric is RMSE in the test set. Since this metric is on the same scale as the observed data values. For the seasonal naive model, its RMSE score will serve as the baseline performance to beat, meaning if any model has a lower RMSE value in the test set.

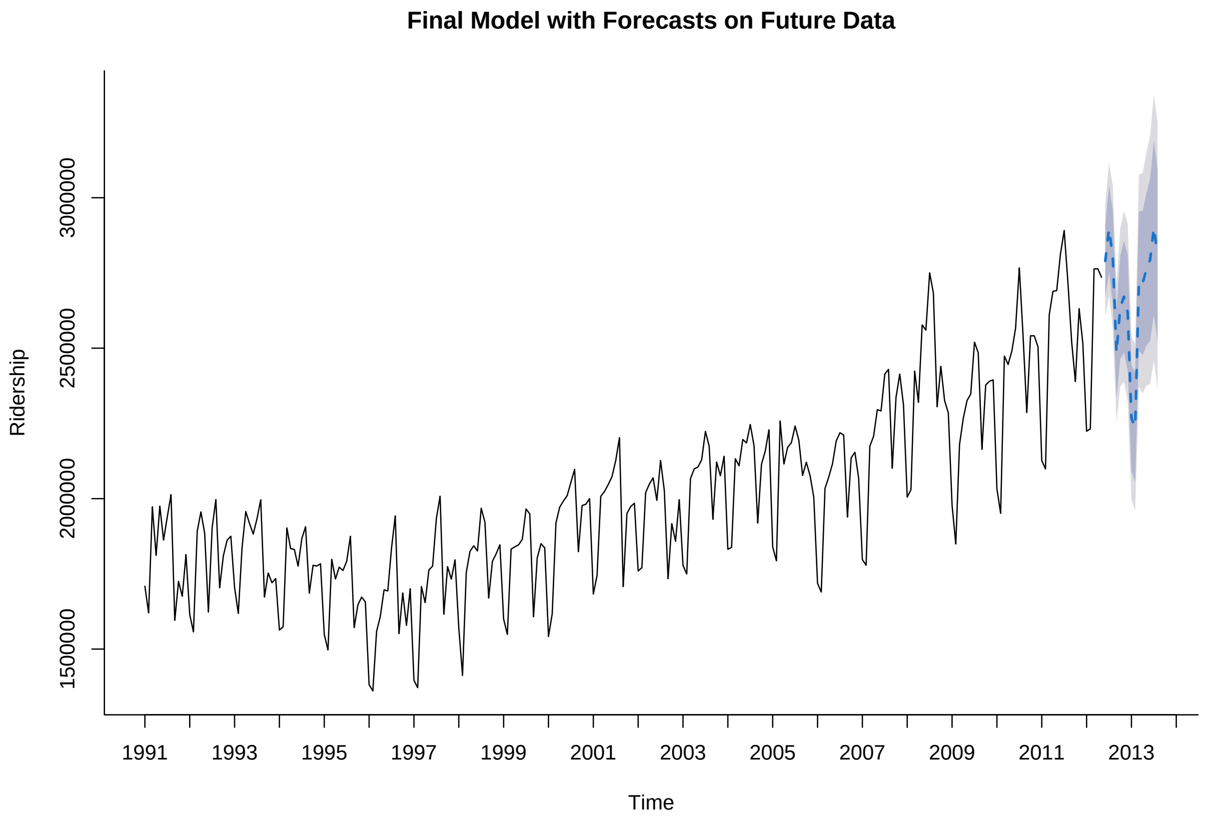
- Seasonal Naive Model - test set's RMSE: 91,056
- Optimal exponential model - test set's RMSE score: 61,488.
- Regression-based model - test set's RMSE score: 100,061
- AR1 Model - test's RMSE score: 388,025
- Optimal ARIMA Model - test's RMSE score: 107,742

The final model selected to forecast the months of June to August of 2013 will be the optimal exponential model with the lowest RMSE score of 61,488 on the test set.

Conclusion

Code:

```r
# the final model selected was the optimal exponential model
final_model <- optimal

#forecasts for June to August 2013 (3 months ahead)
final.pred <- forecast(final_model, h = nValid + 3)

# plot of predictions
plot(final.pred, ylab = "Ridership", xlab = "Time",
     bty = "l", xaxt = "n", flty = 2,
     main = "Final Model with Forecasts on Future Data")

# labels
axis(1, at = seq(1991, 2014, 1), labels = format(seq(1991, 2014, 1)))
```



June to August 2013 Forecasts:

Table 7: June to August 2013 - Ridership Forecasts

	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
Jun 2013	2793575	2523876	3063274	2381106	3206044
Jul 2013	2900579	2611706	3189452	2458786	3342372
Aug 2013	2807559	2519628	3095489	2367207	3247910