Hannah Roach
3/29/2019

Lab Exercise 10: Time Series Analysis with ARIMA

Purpose:	This lab is designed to investigate and practice Time Series Analysis with ARIMA models (Box-Jenkins-methodology). After completing the tasks in this lab you should able to: • Use R functions for ARIMA models • Apply the requirements for generating appropriate training data • Validate the effectiveness of the ARIMA models

Tasks:	Tasks you will complete in this lab include: • Use the R –Studio environment to code ARIMA models • Use the ODBC connection to the database to create the weekly sales data from the retail database • Prepare the data (sorting and rendering the data as a Time series) • Generate a model and evaluate how well it predicts the results and compare the results with original data

_

Workflow Overview

1. Set the Working Directory
2. Establish the ODBC Connection
3. Open Connections to ODBC Database
4. Get Data from the Database
5. Review, Update, and Prepare DataFrame ”msales” File for ARIMA Modeling
6. Convert “sales” into Time Series Type Data
7. Plot the Time Series
8. Analyze the ACF and PACF
9. Difference the Data to Make it Stationary
10. Plot ACF and PACF for the Differenced Data
11. Fit the ARIMA Model
12. Generate Predictions
13. Compare predicted values with actual values

LAB Instructions

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>Log in with GPADMIN credentials on to R-Studio.</td></tr>
<tr><td>2</td><td><u>Set the working directory</u>
Set working directory to ~/LAB10/ , execute the command:
<code>setwd("~/LAB10")</code>

• (Or using the “Tools” option in the tool bar in the RStudio environment.)</td></tr>
<tr><td>3</td><td><u>Establish the ODBC Connection:</u>

Load the RODBC package using the following command:
<code>library('RODBC')</code>

<code>Minimize your screen</code></td></tr>
<tr><td>4</td><td><u>Open Connections to ODBC Database:</u>

1. Before connecting to the ODBC database make sure the file, /etc/odbc.ini, is properly set to point to database “training1”.
2. If not, edit the line that starts with, "Database =" within the file /etc/odbc.ini, to point to, “training1”.

OPEN PUTTY and connect to your BE IP
LOGIN: GPADMIN
Type cd/etc/
Type nano odbc.ini
If Database does not say training1 changed to training1
Then hit CTRL X type y to save and enter
Type exit to close PUTTY

OPEN your minimized SAFARI and continue

3. Ensure the username(uid) and password (pwd) are provided correctly in the following command:

<code>ch <- odbcConnect("Greenplum",uid="gpadmin",
      case="postgresql",pwd="changeme")</code></td></tr>
<tr><td></td><td></td></tr>
</table>

Step	Action

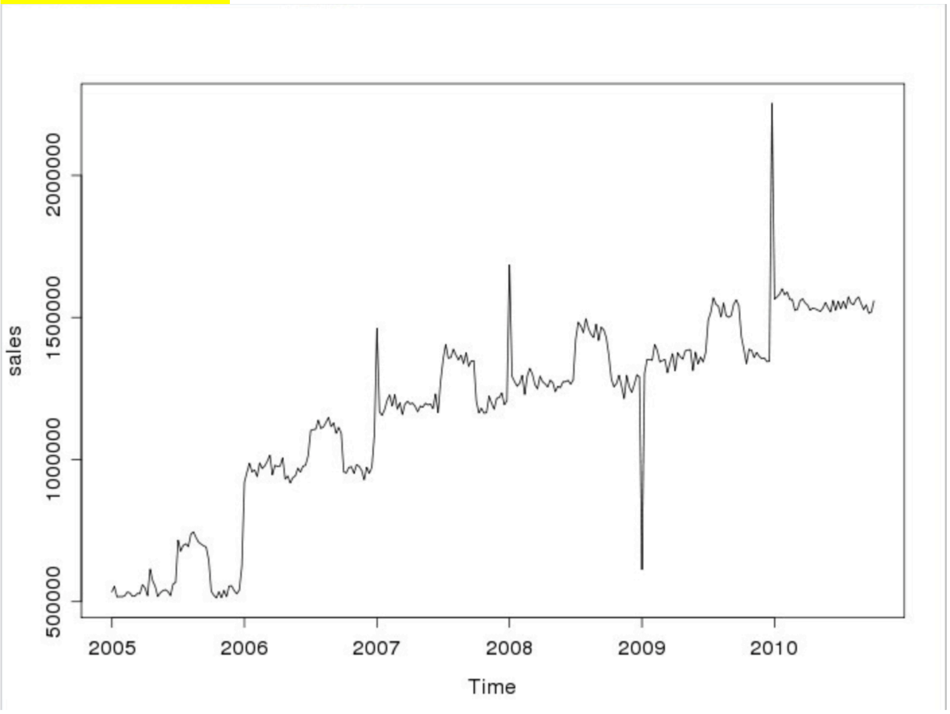
5	**Get Data from the Database:**
	
	1. Drop the table, weekly_sales, from the schema ddemo:
	
	`sqlDrop(ch,"ddemo.weekly_sales")`
	2. Execute an SQL query using the sqlQuery command, creating a table, weekly_sales, in which the weekly sales are grouped first by year and the week within a year:
	
	`> sqlQuery(ch,`
	`"CREATE TABLE`
	`     ddemo.weekly_sales (`
	`       sale INTEGER,`
	`       Y1 INTEGER,`
	`       W1 INTEGER )`
	`       DISTRIBUTED BY (sale);`
	`INSERT INTO ddemo.weekly_sales`
	`SELECT`
	`   SUM((o.item_price*o.item_quantity)) as sale ,`
	`   EXTRACT(YEAR FROM o.order_datetime) as y1,`
	`   CASE`
	`     WHEN (EXTRACT(WEEK FROM o.order_datetime) = 53) THEN 52`
	`     ELSE`
	`     EXTRACT(WEEK FROM o.order_datetime)`
	`   END w1`
	`FROM`
	`   ddemo.order_lineitems o`
	`group`
	`   by y1,w1`
	`;`
	`"`
	`)`
	• **Note:** Note the use of the **EXTRACT** function to obtain the year and the week within the year from the order_datetime field
	• The sales number is accumulated for each week
	    • The ISO Standard for numbering weeks within a year may lead to a year containing 52 or 53 weeks. In order to work with Time Series data you need a consistent "periodicity" with the data. You must accumulate the vales for week 53 with that of week 52 in the same year. Case statement is used to accumulate the week 53 (if exists) along with week 52 data
	1. Get the results from the table into data frame msales. Execute the command:
	
	`msales <- (sqlFetch(ch,"ddemo.weekly_sales"))`
	
	4. Close the ODBC channel.
	`odbcClose(ch)`

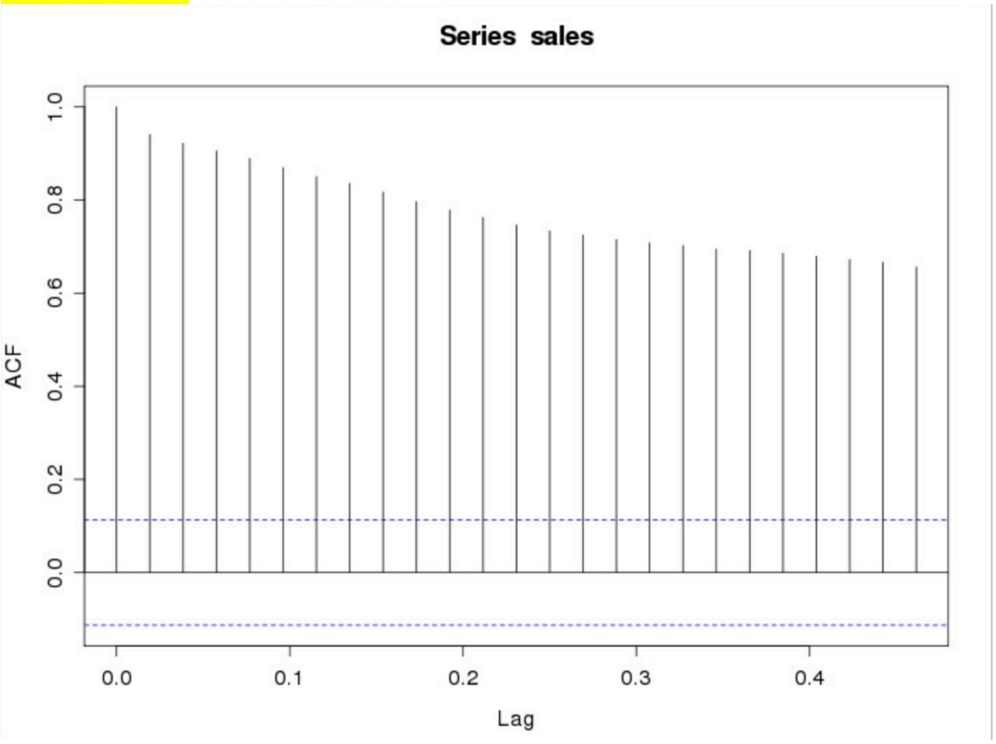
Step	Action

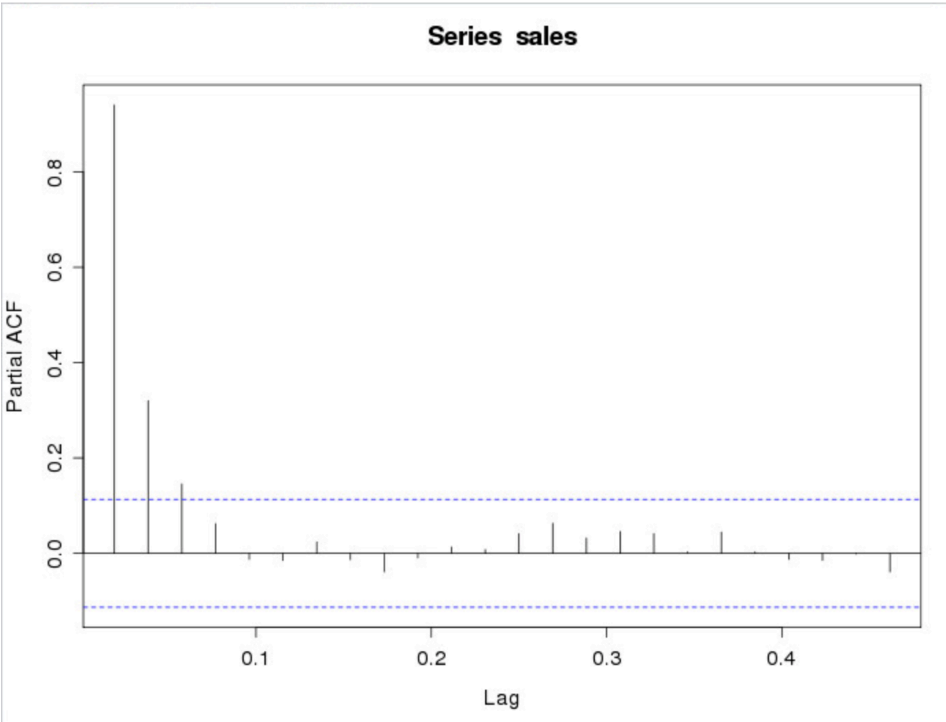
6	**<u>Review, Update and Prepare DataFrame "msales" File for ARIMA Modeling:</u>**
	1. Sort the data in the order of Year and Week:
	Use the R function "order" :
	
	`attach(msales)`
	`msales <- msales[order(y1,w1),]`
	`detach(msales)`
	
	2. Extract 300 values from "asales" for modeling and 12 values to compare with the predictions done by the model. Store them in two different vectors: "sales" and "csales". Use the following command:
	
	`>sales <- c(rep(0,300))`
	`>csales <- c(rep(0,12))`
	`>sales[1:300] <- msales[1:300, 1]`
	`>csales[1:12] <- msales[301:312, 1]`

7	**<u>Convert "sales" into Time Series Type Data:</u>** Convert the "sales" into a time series. • This "transformation" is required for most of the time–series functions, since a time series contains more information than the values themself, namely information about dates and frequencies at which the time series has been recorded. `> sales <- ts(sales,start=2005,frequency=52)`

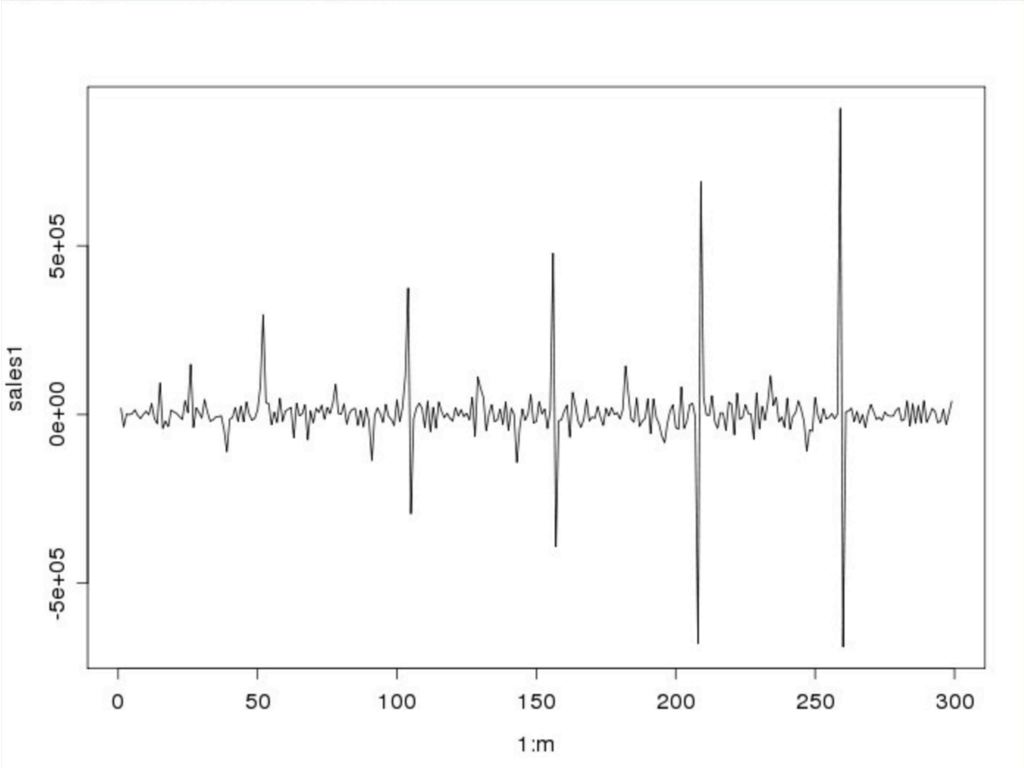
Step	Action

8	**<u>Plot the Time Series:</u>**
	
	1. Plot the Time Series using the following command:
	
	`plot(sales,type="l")`
	`***SCREENSHOT`
	
	2. Review the plot of the Time Series.
	3. Identify the seasonality features in the graph. – It appears that once a year there is a dip spike in sales from June to August and another spike in December.
	4. Is the data Seasonal (Do you see patterns that repeat at a particular frequency)? Yes
	5. Is the data stationary? The data does not appear to be stationary because there is a linear trend in the data and the data appears to be seasonal.
	6. Is there a trend to the data? Yes, there is an upward trend in the data.

9	**Analyze the ACF and PACF :**
	
	The next step in analyzing time series is to examine the **autocorrelations (ACF)** and **partial autocorrelations (PACF).** R provides the functions acf( ) and pacf( ) for computing and plotting of ACF and PACF.
	
	1. Use the **parameter function "par"** to set the plot window to display both the ACF and PACF plots. Plot ACF and PACF on the same graph using the command:
	
	`> par(mfrow=c(2,1))`
	`> acf(sales)`
	`> pacf(sales)`
	**SCREENSHOT
	

Step	Action
	 2. Using the plot generated in the plot window: • Does the ACF tail off quickly? Shows that the AR model handles seasonality in the data after 0.6. ACF 0.2, 0.4, and 0.6 have significance. • What does the ACF indicate with respect to stationarity of the data? It shows that the data is not stationary. • What will you do to make the data stationary? Take the difference of the linear trend.

Step	Action

10	**<u>Difference the Data to Make it Stationary:</u>**
	
	To difference the data and make it stationary you should use the "diff" function in R. The "diff" function takes the pair of each observation and differences it from the one previous to it.
	
	1. Run the following code:
	`> #Difference the series and plot it`
	`> sales1 <- diff(sales)`
	`> m <- length(sales1)`
	`> par(mfrow=c(1,1))`
	`> plot(1:m,sales1,type="l")`
	***SCREENSHOT
	
	
	1. Do you see any trend to the data now? No
	2. Is seasonality still shown in the data? Yes, in the y-axis.
	3. How does the trend of oscillating spikes change as you move from left to right on this plot? It increases

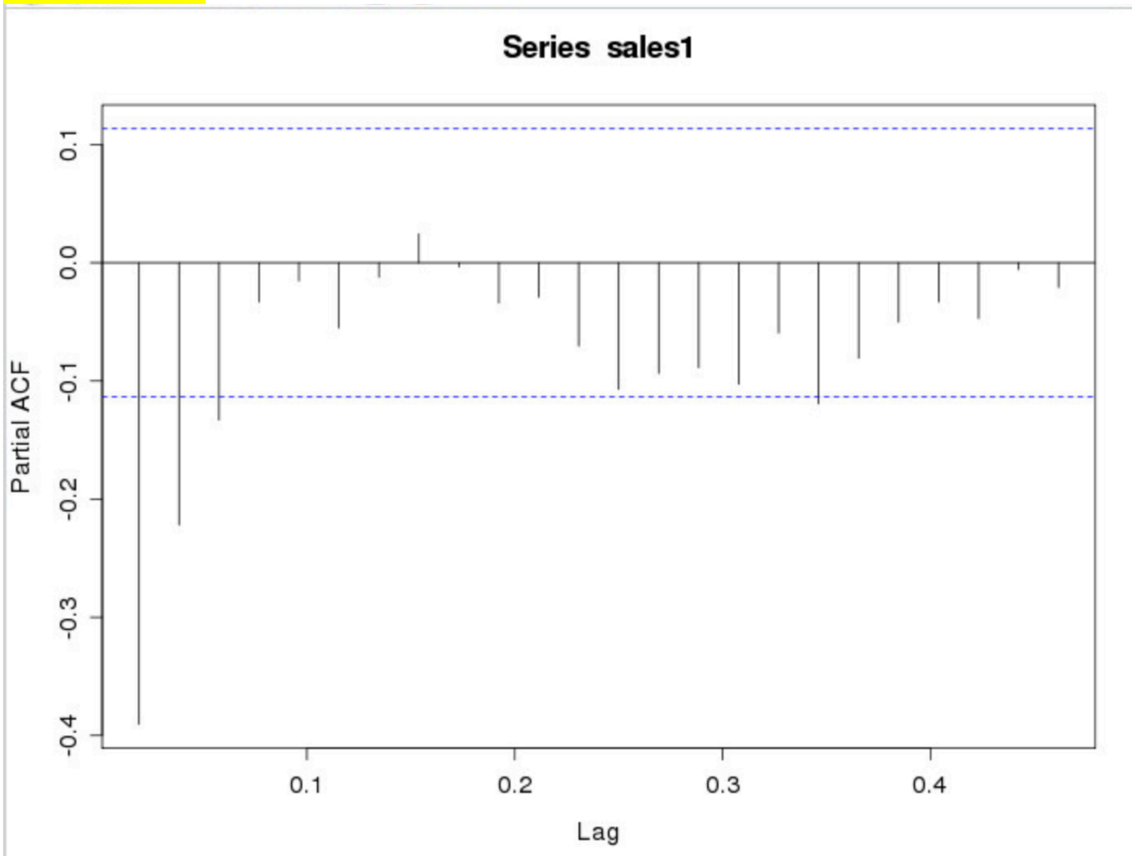
<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>11</td><td>

<u>Plot ACF and PACF for the Differenced Data:</u>

1. Run the following code:

```
> #Plot ACF and PACF on the same graph
> par(mfrow=c(2,1))
> acf(sales1)
> pacf(sales1)
```

***SCREENSHOT



2. Do you see the ACF tailing off quickly? Yes

</td></tr>
</table>

12

Fit the ARIMA Model:

Once you configure the data and review the seasonality elements, you are ready to fit an ARIMA model. Selecting the correct model to fit an ARIMA model is a bit of an art. Algorithms are used to select the correct parameters.

- Use the data configuration and the principle of "parsimony" to fit a basic model
- Use the statement ARIMA
- Use the time series
- Use "sales" and not the "diff(sales)" that we computed in step 10. ARIMA will automatically invoke the "diff" function based on the parameters specified.
- You need to specify the order (p,d,q) where "p" is the order of AR , "q" order of MA and "d" is the number of differences.
- Fit (1,1,0) so the model will difference once, use AR and MA parameters as 1 and 0.
- Use a seasonal statement since the data seems to have a seasonal component to it as you see spikes every 52 weeks.
- In the seasonal statement, you have a "list" where you put in the "order" and a (time) "period" which will be "52".
- Use "include.mean = false" - R will force a mean and continue any trend seen in the past for the model. This automatically defaults to "True". You turn the automatic default off with this statement.
- **Note:** You can also experiment with not using this statement during this lab.

1. Type in the following code:

```
sales.fit <- arima (sales,
                     order=c(1,1,0),
                     seasonal = list(order=c(1,1,0),period=52),
                     include.mean=FALSE)
> sales.fit
**SCREENSHOT
```

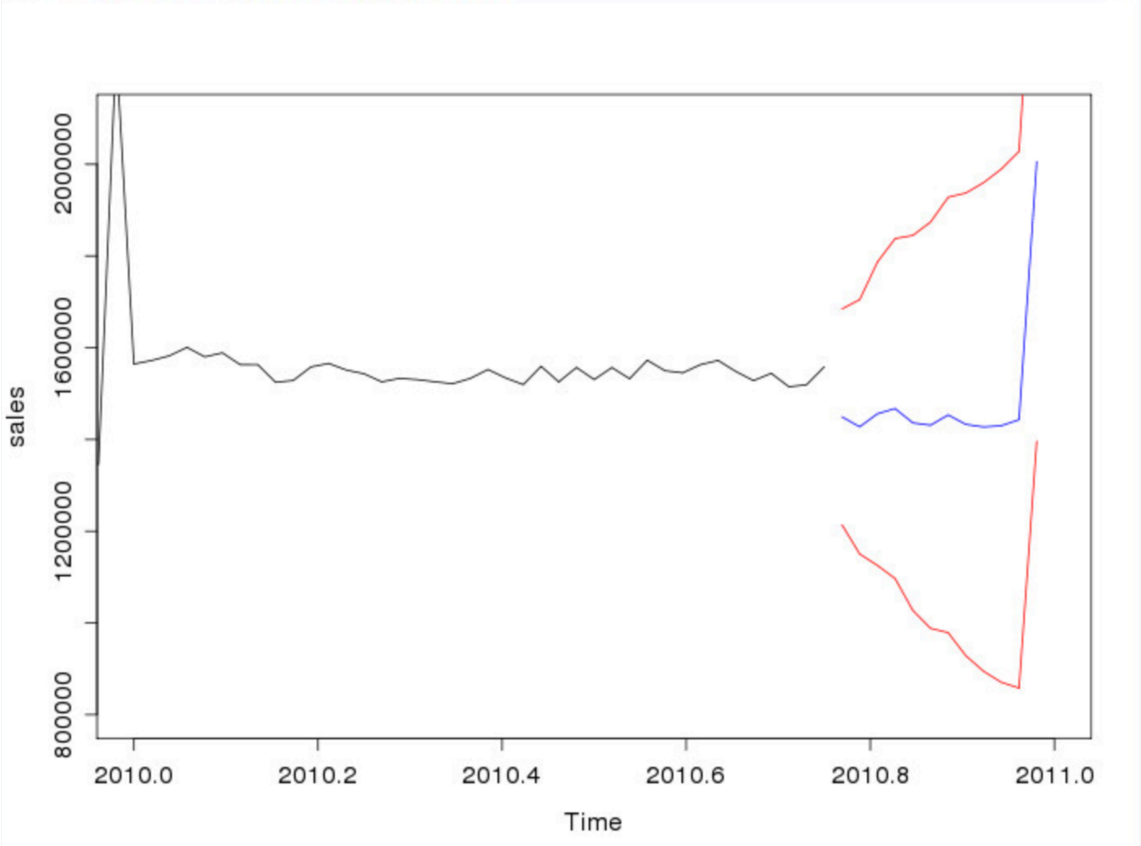
```
> sales.fit

Call:
arima(x = sales, order = c(1, 1, 0), seasonal = list(order = c(1, 1, 0), period = 52),
    include.mean = FALSE)

Coefficients:
          ar1      sar1
      -0.3769   -0.3770
s.e.   0.0608    0.0657

sigma^2 estimated as 1.384e+10:  log likelihood = -3238.4,  aic = 6482.8
```

2. Review the output displayed.
3. Record the coefficients for the AR term and the seasonal AR terms and the standard errors. AR term: -0.3769, seasonal AR terms: -0.3770, standard errors: 0.0608, 0.0657
4. What is your observation on the standard errors compared to the coefficients? The standard errors are positive whereas the coefficients are negative. The standard error is much smaller than coefficients.

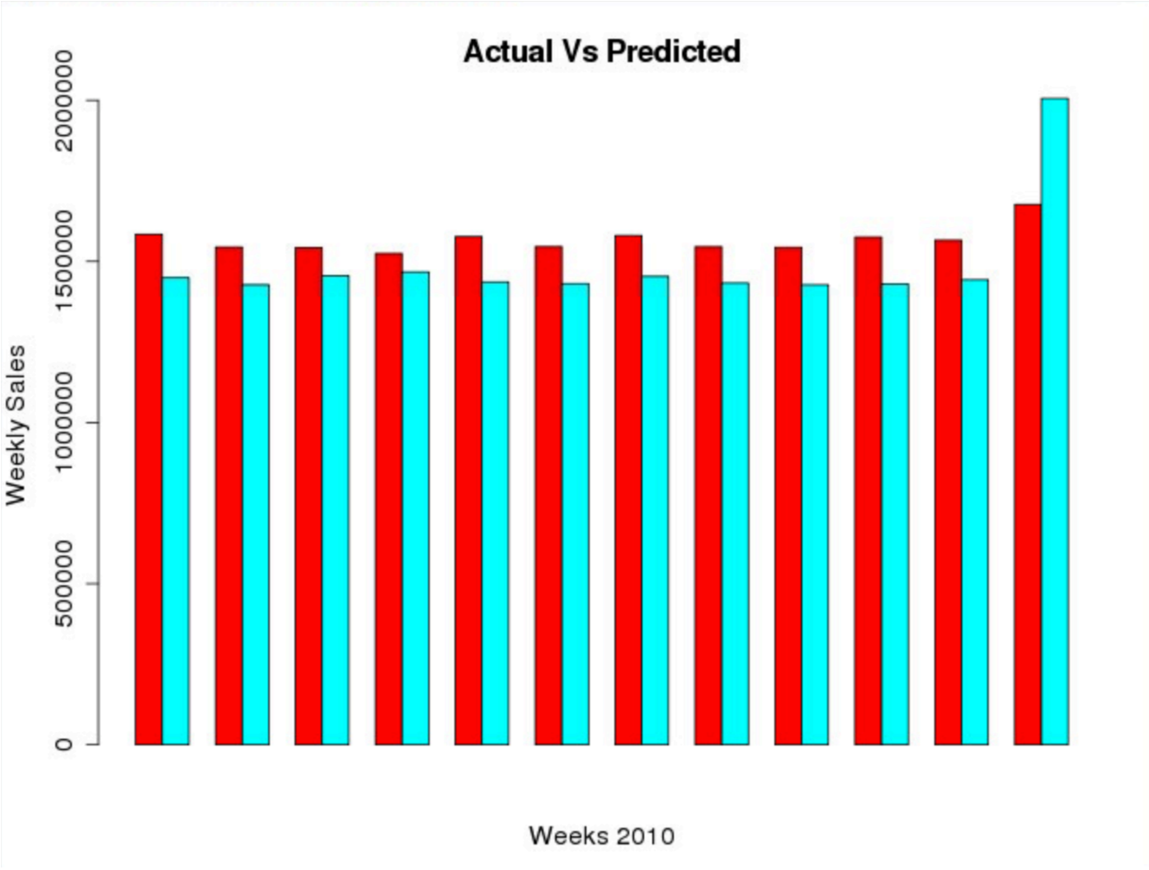
- **Note:** The model gives you the "log likelihoods" that provide important input on model selection.

Step	Action

Step	Action

13	**Generate Predictions:**
	
	1. Use the fitted model "sales.fit" and the "predict" statement for 12 periods ahead:
	
	`sales.predict <- predict (sales.fit, n.ahead=12)`
	
	• **Note:** You can see the predictions by typing "sales.fit".
	
	2. Plot the predictions using plot statements:
	
	`> par(mfrow=c(1,1))`
	`> plot (sales,xlim=c(2009.50,2011))`
	`> lines (sales.predict$pred,col="blue")`
	`> lines (sales.predict$pred+2*sales.predict$se,col="red")`
	`> lines (sales.predict$pred-2*sales.predict$se,col="red")`
	**SCREENSHOT
	
	• **Note:** The first plot statement plots the original "sales" data
	• EMC has restricted the "xlim" value to zoom in on the values just prior to the predicted values
	• The "blue" line indicates the mean of the predictions
	• The "red" lines denote the upper and lower bounds of the prediction

Step	Action

14	**<u>Compare predicted values with actual values</u>**
	
	1. Compare the predicted values with the actual values and plot the actual Values Predicted as a barplot. Use the following code:
	
	`> #comparing the predictions with the actual values`
	`> par(mfrow=c(1,1))`
	`> x <- c(rep(0,24))`
	`> x[1:12] <- msales[1:12]`
	`> x[13:24] <- as.numeric(sales.predict$pred)`
	`> forbar <- matrix(x,ncol=12,byrow=TRUE)`
	`> colnames(forbar)<- asales[1:12,3]`
	`> rownames(forbar)<- c("Actual","Predicted")`
	`> barplot(forbar,beside=TRUE,`
	`+         main="Actual Vs Predicted",`
	`+         ylab="Weekly Sales",`
	`+         col=rainbow(2),`
	`+         xlab="Weeks 2010")`
	
	2. **SCREENSHOT
	
	

End of Lab Exercise