Week 2 - Linear Regression & Deterministic Trend Models

Jonathan Thong

2023-03-03

Linear Regression

As a first step, let's make sure that our workspace is clear so that we are not accidentally calling variables from other projects that may still be in our environment. We do this by running the following line:

```
rm(list = ls())
```

The data set "cars" contains 50 observations of two variables, car speeds and the corresponding distances taken to stop. The data is pre-loaded into R and can be printed in the console by simply typing **cars** into the console. However, rather than printing the entire data set, we can just have a look at the first few observations using **head()**:

```
head(cars)
```

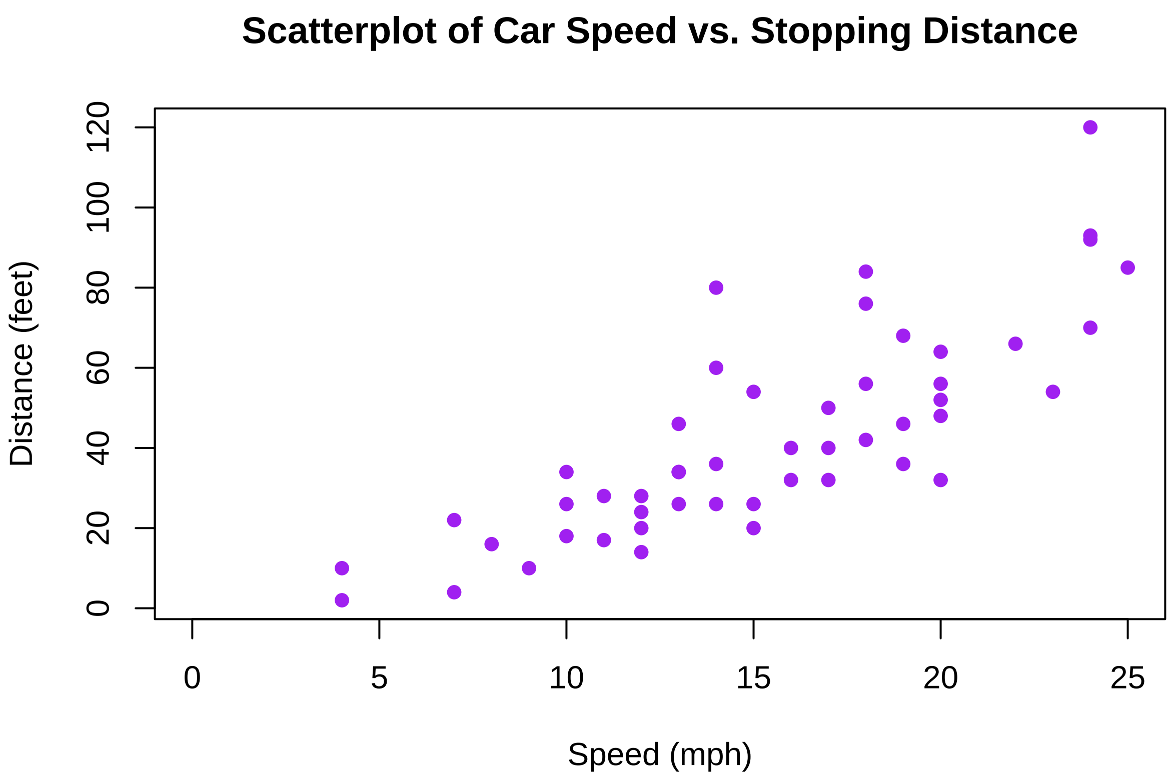
```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

Let's take this data set and place it in a data frame called **cardata** and we will attach the data using **attach()** so that we can simply type out the names our variables to access them rather than having to constantly refer to them as columns of the data frame:

```
cardata <- data.frame(cars)
attach(cardata)
```

As a first step, let's generate a scatter plot of the two variables to get a sense of the degree to which they are linearly related.

```
plot(speed,dist,
     main ="Scatterplot of Car Speed vs. Stopping Distance",
     xlab = "Speed (mph)",
     ylab = "Distance (feet)",
     xlim = c(0,25),
     col = "purple",
     pch = 16)
```



Let's now compute a simple linear regression in which we set the stopping distance as our dependent variable and car speed as our explanatory variable. We use the **lm()** function to compute our estimates and associated statistics and we will store the regression output in an object called **linreg1**:

```
linreg1 <- lm(formula = dist ~ speed, data = cardata)
```

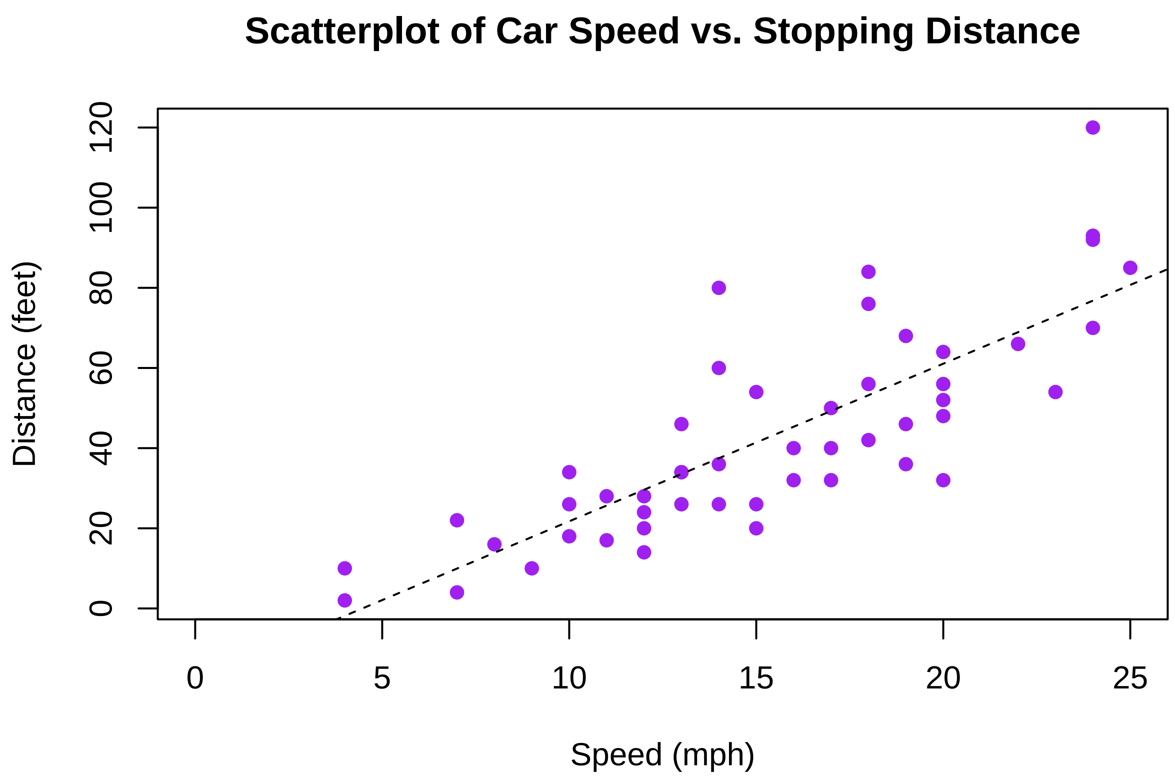
To see the results of the regression, we use the **summary()** function

```
summary(linreg1)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cardata)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

To plot the estimated regression line (i.e., the line of best fit!), we use the command **abline()**

```
plot(speed,dist,
     main ="Scatterplot of Car Speed vs. Stopping Distance",
     xlab = "Speed (mph)",
     ylab = "Distance (feet)",
     xlim = c(0,25),
     col = "purple",
     pch = 16)
abline(linreg1, lty = "dashed")
```



Note that the lm function will include an intercept term as a default. In the event that we want to set the intercept term equal to zero, we will need to type the following:

```
linreg2 <- lm(formula = dist ~ 0 + speed, data = cardata)
```

Setting the intercept term to zero *a priori* might be sensible in this case as we know from the laws of physics that the stopping distance of a stationary car must necessarily be zero! Let's have a look at the estimation results:

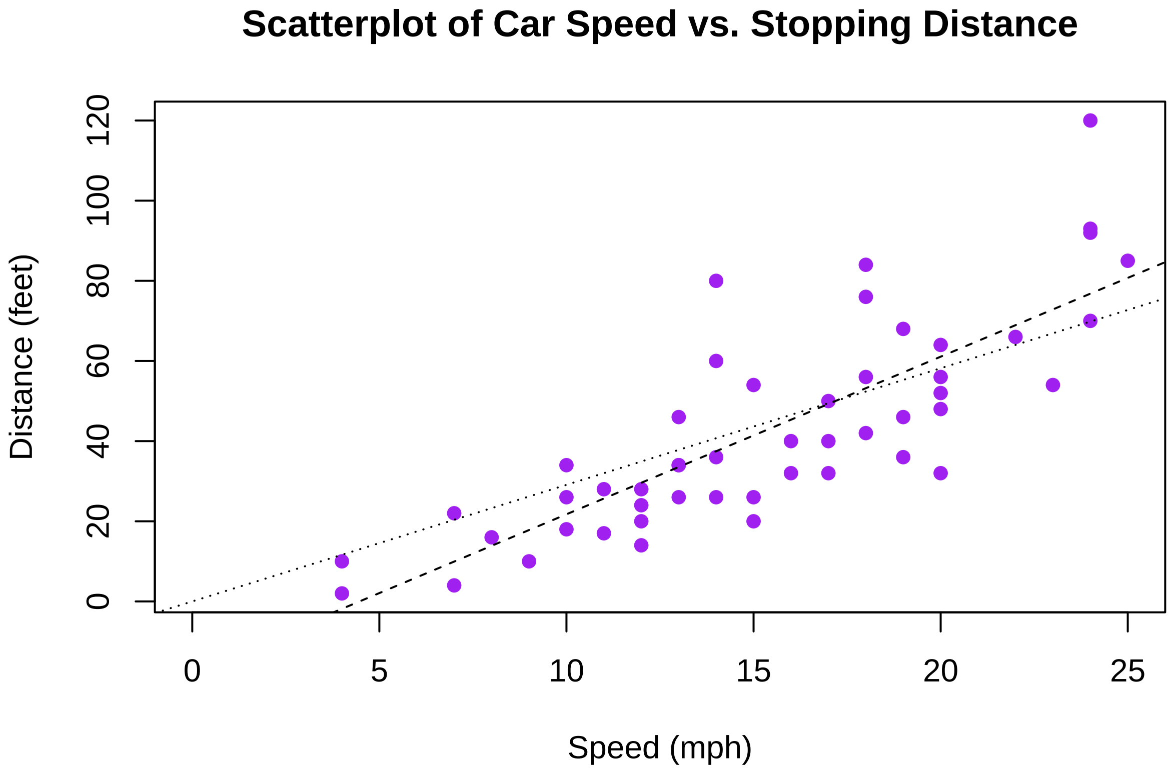
```
summary(linreg2)
```

```
##
## Call:
## lm(formula = dist ~ 0 + speed, data = cardata)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -26.183 -12.637  -5.455   4.590  50.181
##
## Coefficients:
##       Estimate Std. Error t value Pr(>|t|)
## speed   2.9091     0.1414   20.58   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.26 on 49 degrees of freedom
## Multiple R-squared:  0.8963, Adjusted R-squared:  0.8942
## F-statistic: 423.5 on 1 and 49 DF,  p-value: < 2.2e-16
```

We can then use the **abline()** again to compare the two fitted lines:

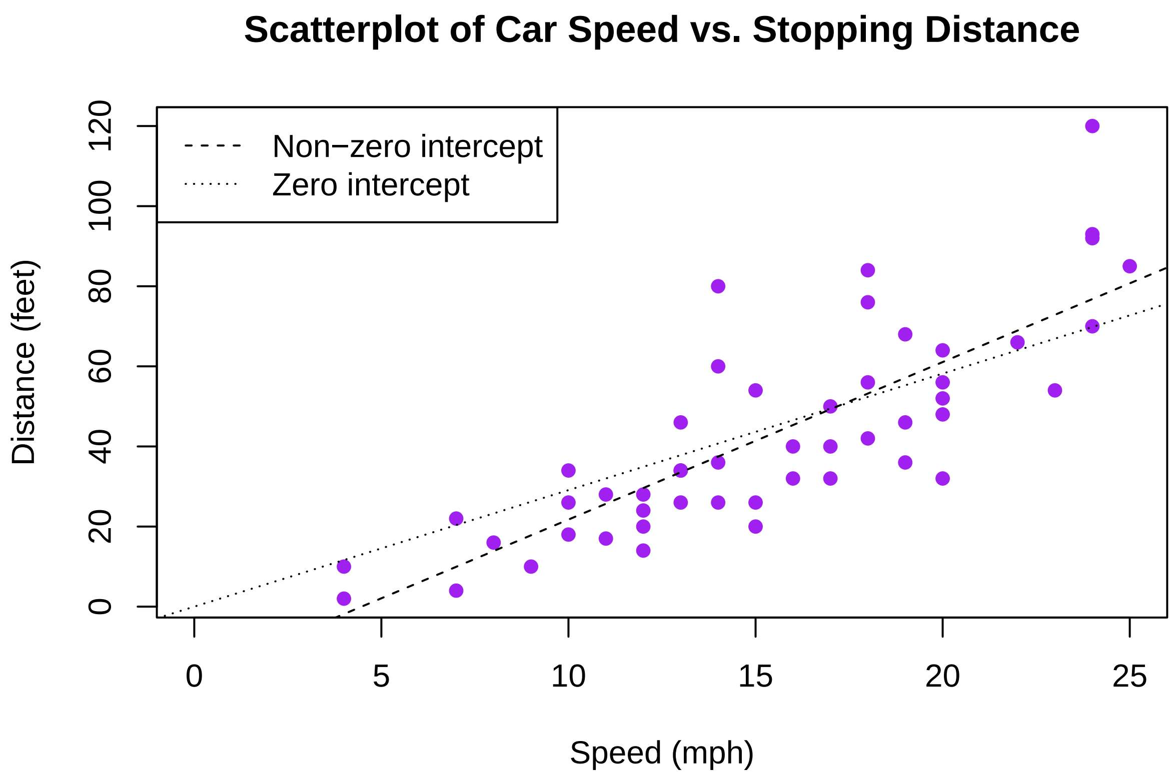
```
plot(speed,dist,
     main ="Scatterplot of Car Speed vs. Stopping Distance",
     xlab = "Speed (mph)",
     ylab = "Distance (feet)",
     xlim = c(0,25),
     col = "purple",
     pch = 16)
abline(linreg1, lty = "dashed")
abline(linreg2, lty = "dotted")
```



We can also add a legend using **legend()**

```
plot(speed,dist,
     main ="Scatterplot of Car Speed vs. Stopping Distance",
     xlab = "Speed (mph)",
     ylab = "Distance (feet)",
     xlim = c(0,25),
     col = "purple",
     pch = 16)
abline(linreg1, lty = "dashed")
abline(linreg2, lty = "dotted")
```

```r
legend(x = "topleft",
       legend = c("Non-zero intercept", "Zero intercept"),
       lty = c("dashed", "dotted"))
```



Now that we know how to perform a simple linear regression, let's proceed to a multiple regression using violent crime rates by US State. This is a data set called **USArrests** that is also pre-loaded into R, so we won't need to import it. Looking at the first few observations:

```r
head(USArrests)
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

Let's store the data in a data frame called **crimdata** and attach it. As good practice, we should also detach the **cardata** data frame now that we are no longer using it. This is to ensure that we do not accidentally call variables from data frames we are not using that happen to have the same name.

```r
crimdata <- data.frame(USArrests)
attach(crimdata)
detach(cardata)
```

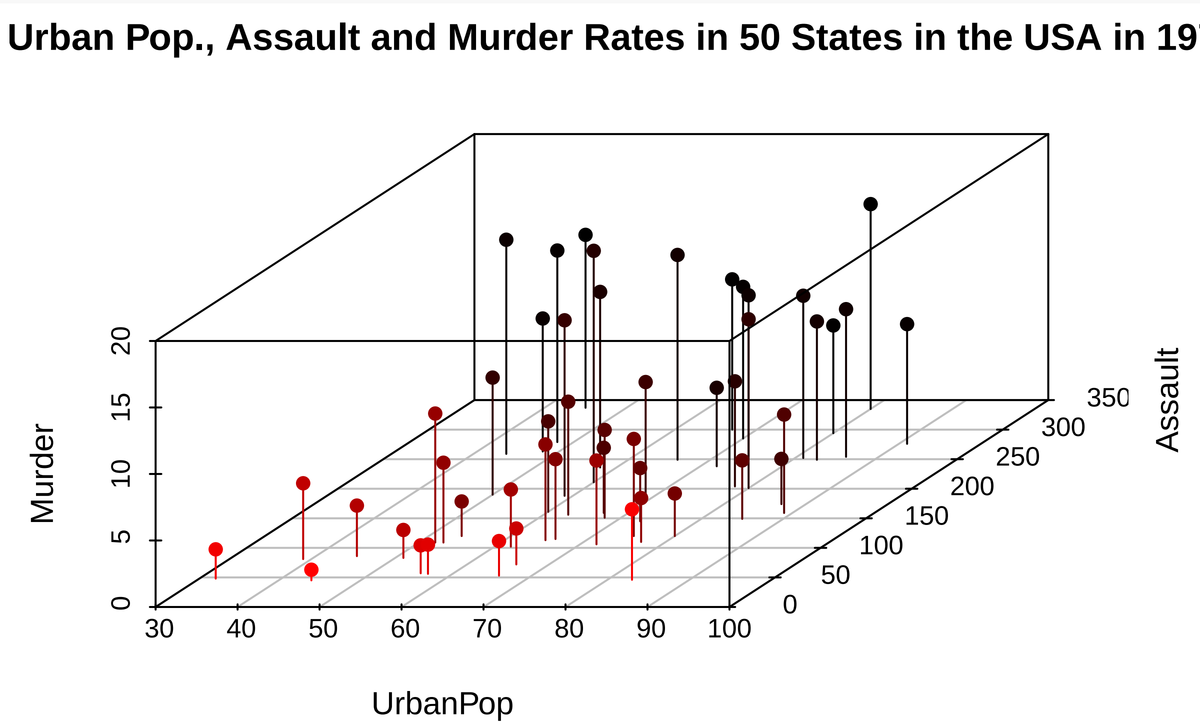
Then, running a multiple regression simple involves using the same **lm()** function, but now we specify multiple explanatory variables. So let Murder be our dependant variable and UrbanPop and Assault be our explanatory variables:

```r
linreg3 <- lm(formula = Murder ~ UrbanPop + Assault)
summary(linreg3)
```

```
##
## Call:
## lm(formula = Murder ~ UrbanPop + Assault)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5530 -1.7093 -0.3677  1.2284  7.5985
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.207153   1.740790   1.842   0.0717 .
## UrbanPop    -0.044510   0.026363  -1.688   0.0980 .
## Assault      0.043910   0.004579   9.590 1.22e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.58 on 47 degrees of freedom
## Multiple R-squared:  0.6634, Adjusted R-squared:  0.6491
## F-statistic: 46.32 on 2 and 47 DF,  p-value: 7.704e-12
```

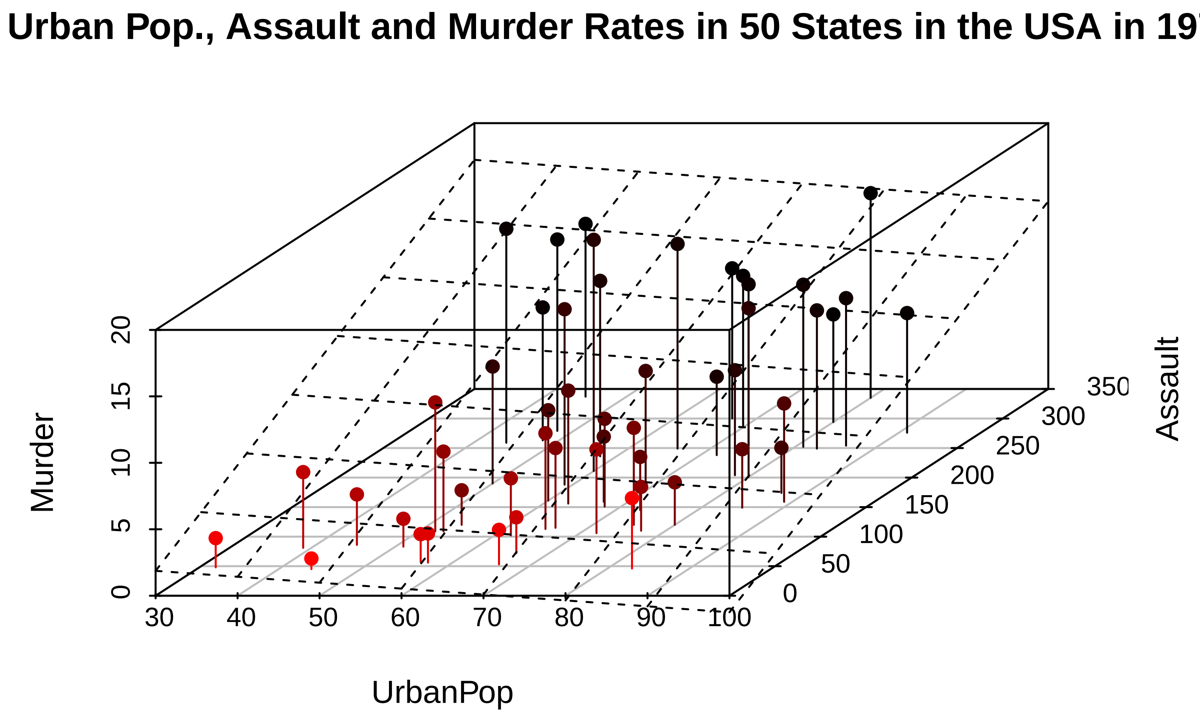
Now that we are working with a regression model with two explanatory variables, visualising our data and estimation results requires a 3-D plotting package. We will use the **scatterplot3d** package. Once we have installed and loaded the package, we can generate a plot of the data by running the following line:

```
library(scatterplot3d)
scatterplot3d(UrbanPop,Assault, Murder,
              pch = 16,
              highlight.3d = TRUE,
              type = "h",
              main = "Urban Pop., Assault and Murder Rates in 50 States in the USA in 1973" )
```



Since our fitted values now take two inputs, it would be visualised as a plane in this three dimensional space. We can plot it using the following code:

```r
s3d <- scatterplot3d(UrbanPop,Assault, Murder, pch = 16, highlight.3d = TRUE, type = "h", main = "Urban
s3d$plane3d(linreg3)
```



Fitting Deterministic Trend Models

Now that we understand how to estimate linear regressions, we can proceed to specify and estimate some simple time series models with deterministic trends. Let's work with a time series of quarterly e-commerce sales in the US obtained from the FRED database at the following link https://fred.stlouisfed.org/series/ECOMSA. After downloading the data into a csv file named *ECOMSA.csv*, we import it and store it in a data frame called **ecomdata**

```r
ecomdata <- read.csv("ECOMSA.csv")
```

Looking at the entire data frame, we notice that the last few rows of the data frame are empty and contain NAs.

```r
ecomdata
```

```
##          date ecomsa
## 1  1/10/1999   4476
## 2    1/1/2000   5691
## 3    1/4/2000   6465
## 4    1/7/2000   7419
## 5  1/10/2000   7840
## 6    1/1/2001   8135
## 7    1/4/2001   8336
## 8    1/7/2001   8335
## 9  1/10/2001   9314
## 10   1/1/2002   9904
## 11   1/4/2002  10742
## 12   1/7/2002  11543
## 13 1/10/2002  12231
## 14   1/1/2003  12738
```

```
## 15  1/4/2003  13773
## 16  1/7/2003  14833
## 17 1/10/2003  15588
## 18  1/1/2004  16697
## 19  1/4/2004  17519
## 20  1/7/2004  18506
## 21 1/10/2004  19631
## 22  1/1/2005  20801
## 23  1/4/2005  22233
## 24  1/7/2005  23653
## 25 1/10/2005  24364
## 26  1/1/2006  26417
## 27  1/4/2006  27367
## 28  1/7/2006  28842
## 29 1/10/2006  30138
## 30  1/1/2007  31728
## 31  1/4/2007  33524
## 32  1/7/2007  34841
## 33 1/10/2007  35784
## 34  1/1/2008  36012
## 35  1/4/2008  36509
## 36  1/7/2008  36287
## 37 1/10/2008  33051
## 38  1/1/2009  34127
## 39  1/4/2009  35279
## 40  1/7/2009  37400
## 41 1/10/2009  38117
## 42  1/1/2010  39284
## 43  1/4/2010  41301
## 44  1/7/2010  43469
## 45 1/10/2010  45076
## 46  1/1/2011  47041
## 47  1/4/2011  48864
## 48  1/7/2011  50207
## 49 1/10/2011  53217
## 50  1/1/2012  55316
## 51  1/4/2012  56593
## 52  1/7/2012  58589
## 53 1/10/2012  60884
## 54  1/1/2013  62310
## 55  1/4/2013  64071
## 56  1/7/2013  65982
## 57 1/10/2013  68408
## 58  1/1/2014  70372
## 59  1/4/2014  73396
## 60  1/7/2014  75811
## 61 1/10/2014  77724
## 62  1/1/2015  80365
## 63  1/4/2015  82996
## 64  1/7/2015  85775
## 65 1/10/2015  88482
## 66  1/1/2016  91385
## 67  1/4/2016  94381
## 68  1/7/2016  97174
```

```
## 69 1/10/2016 100449
## 70  1/1/2017 104503
## 71  1/4/2017 108728
## 72  1/7/2017 111854
## 73 1/10/2017 117867
## 74  1/1/2018 121808
## 75  1/4/2018 125257
## 76  1/7/2018 128222
## 77 1/10/2018 131668
## 78  1/1/2019 133162
## 79  1/4/2019 138200
## 80  1/7/2019 146201
## 81 1/10/2019 152210
## 82  1/1/2020 159853
## 83               NA
## 84               NA
## 85               NA
## 86               NA
## 87               NA
## 88               NA
## 89               NA
## 90               NA
## 91               NA
## 92               NA
## 93               NA
```

We can easily remove these rows using the **na.omit()** function in the following way:

```
ecomdata <- na.omit(ecomdata)
ecomdata
```

```
##         date ecomsa
## 1  1/10/1999   4476
## 2   1/1/2000   5691
## 3   1/4/2000   6465
## 4   1/7/2000   7419
## 5  1/10/2000   7840
## 6   1/1/2001   8135
## 7   1/4/2001   8336
## 8   1/7/2001   8335
## 9  1/10/2001   9314
## 10  1/1/2002   9904
## 11  1/4/2002  10742
## 12  1/7/2002  11543
## 13 1/10/2002  12231
## 14  1/1/2003  12738
## 15  1/4/2003  13773
## 16  1/7/2003  14833
## 17 1/10/2003  15588
## 18  1/1/2004  16697
## 19  1/4/2004  17519
## 20  1/7/2004  18506
## 21 1/10/2004  19631
## 22  1/1/2005  20801
## 23  1/4/2005  22233
```

```
## 24  1/7/2005  23653
## 25 1/10/2005  24364
## 26  1/1/2006  26417
## 27  1/4/2006  27367
## 28  1/7/2006  28842
## 29 1/10/2006  30138
## 30  1/1/2007  31728
## 31  1/4/2007  33524
## 32  1/7/2007  34841
## 33 1/10/2007  35784
## 34  1/1/2008  36012
## 35  1/4/2008  36509
## 36  1/7/2008  36287
## 37 1/10/2008  33051
## 38  1/1/2009  34127
## 39  1/4/2009  35279
## 40  1/7/2009  37400
## 41 1/10/2009  38117
## 42  1/1/2010  39284
## 43  1/4/2010  41301
## 44  1/7/2010  43469
## 45 1/10/2010  45076
## 46  1/1/2011  47041
## 47  1/4/2011  48864
## 48  1/7/2011  50207
## 49 1/10/2011  53217
## 50  1/1/2012  55316
## 51  1/4/2012  56593
## 52  1/7/2012  58589
## 53 1/10/2012  60884
## 54  1/1/2013  62310
## 55  1/4/2013  64071
## 56  1/7/2013  65982
## 57 1/10/2013  68408
## 58  1/1/2014  70372
## 59  1/4/2014  73396
## 60  1/7/2014  75811
## 61 1/10/2014  77724
## 62  1/1/2015  80365
## 63  1/4/2015  82996
## 64  1/7/2015  85775
## 65 1/10/2015  88482
## 66  1/1/2016  91385
## 67  1/4/2016  94381
## 68  1/7/2016  97174
## 69 1/10/2016 100449
## 70  1/1/2017 104503
## 71  1/4/2017 108728
## 72  1/7/2017 111854
## 73 1/10/2017 117867
## 74  1/1/2018 121808
## 75  1/4/2018 125257
## 76  1/7/2018 128222
## 77 1/10/2018 131668
```

```
## 78  1/1/2019 133162
## 79  1/4/2019 138200
## 80  1/7/2019 146201
## 81 1/10/2019 152210
## 82  1/1/2020 159853
```

Also, if we check the class of each column in our data frame using the **sapply()** function, we notice that the dates have been stored as character strings as opposed to dates.

```
sapply(ecomdata, class)
```

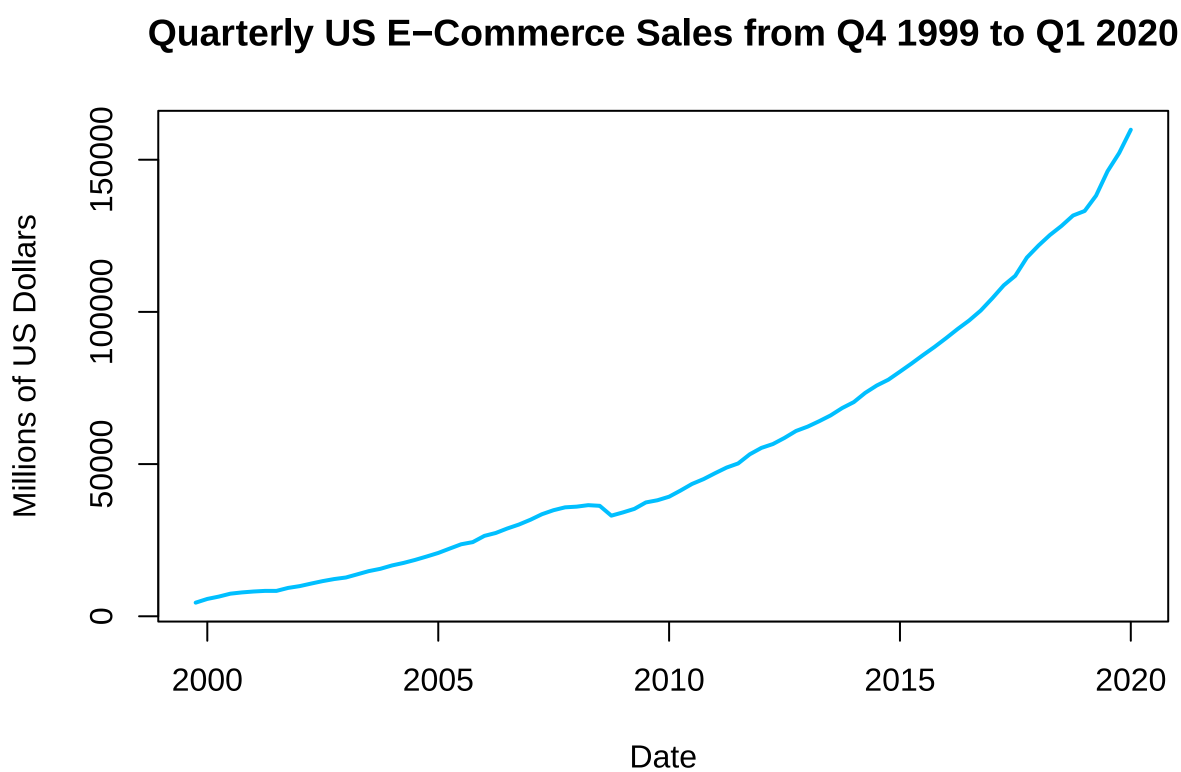
```
##        date      ecomsa
## "character"   "integer"
```

We will have to convert the character strings in the date column into a format that R recognises as dates. We can do this using the **as.Date()** function:

```
ecomdata$date <- as.Date(ecomdata$date, "%d/%m/%Y")
```

Having dealt with the missing observations, let's plot the data:

```
plot(ecomdata$date,ecomdata$ecomsa,
     main = "Quarterly US E-Commerce Sales from Q4 1999 to Q1 2020",
     xlab = "Date",
     ylab = "Millions of US Dollars",
     type = "l",
     lwd = 2.0,
     col = "deepskyblue")
```



So we can see a clear upward trend in the data. Let's try to fit a linear trend specification to our data. To do this, we first need to create a time trend variable (i.e., a deterministic counting variable that indexes the time periods):

```
T <- length(ecomdata$date)
time <- seq(1,T)
```

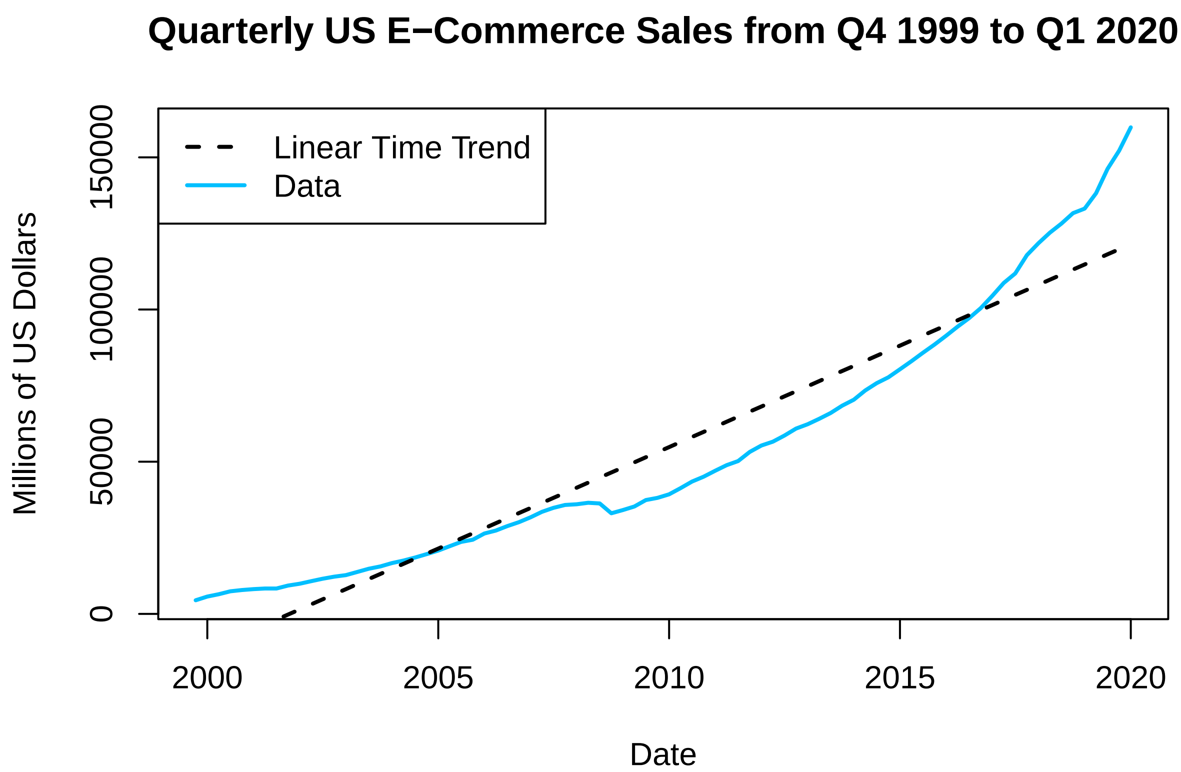
Having created this variable, we can proceed to estimate the following regression:

```
trendmod1 <- lm(formula = ecomdata$ecomsa ~ time)
summary(trendmod1)
```

```
##
## Call:
## lm(formula = ecomdata$ecomsa ~ time)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15509 -11315  -2088   7483  38381
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -15221.07    2762.05  -5.511 4.23e-07 ***
## time          1666.99      57.81  28.834  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12390 on 80 degrees of freedom
## Multiple R-squared:  0.9122, Adjusted R-squared:  0.9111
## F-statistic: 831.4 on 1 and 80 DF,  p-value: < 2.2e-16
```

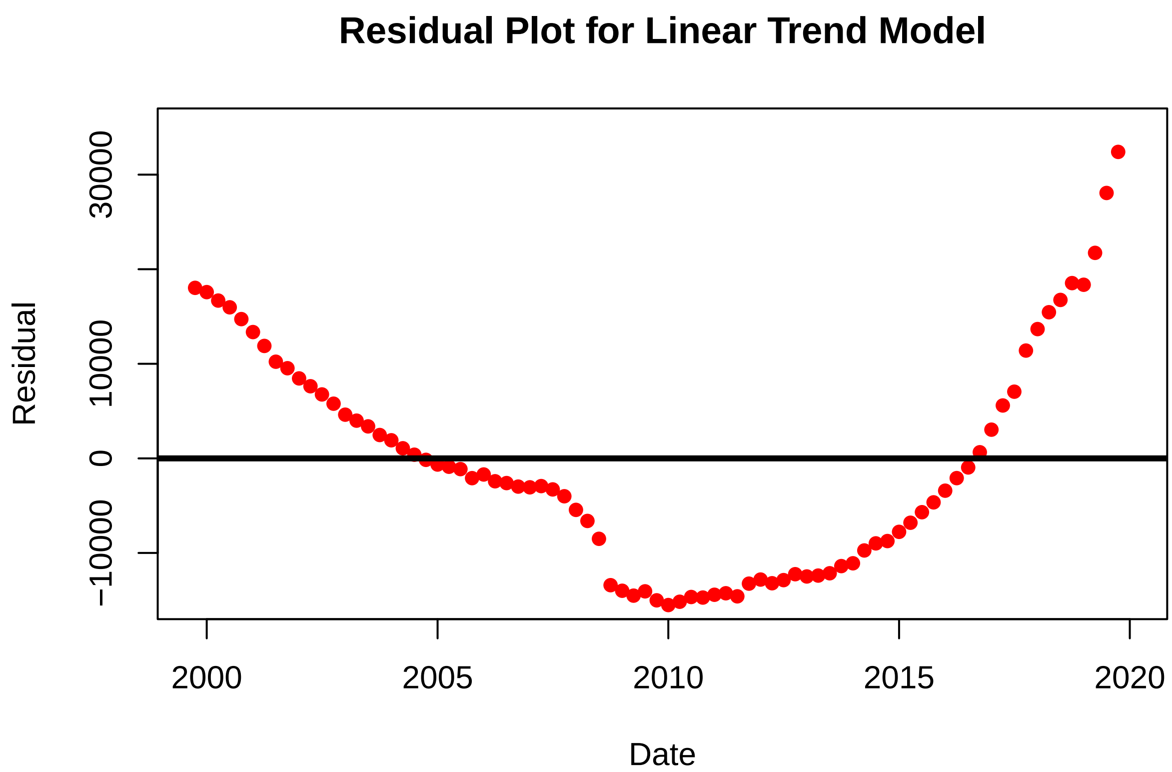
To plot the estimated linear trend along with our data we simply use the **lines()** function after our original plot code:

```
plot(ecomdata$date,ecomdata$ecomsa,
     main = "Quarterly US E-Commerce Sales from Q4 1999 to Q1 2020",
     xlab = "Date",
     ylab = "Millions of US Dollars",
     type = "l",
     lwd = 2.0,
     col = "deepskyblue")
lines(ecomdata$date, predict(trendmod1), type = 'l', lty = "dashed", lwd = 2.0)

legend(x = "topleft",
       legend = c("Linear Time Trend", "Data"),
       lty = c("dashed", "solid"),
       lwd = 2.0,
       col = c("black", "deepskyblue"))
```



Notice that the linear trend does not appear to fit the data very well. Let's have a look at the residuals from our regression model:

```
trendmod1.res <- resid(trendmod1)

plot(ecomdata$date, trendmod1.res,
     main = "Residual Plot for Linear Trend Model",
     ylim = c(-15000,35000),
     xlab = "Date",
     ylab = "Residual",
     col = "red",
     pch =16)
abline(0,0, lwd = 3)
```



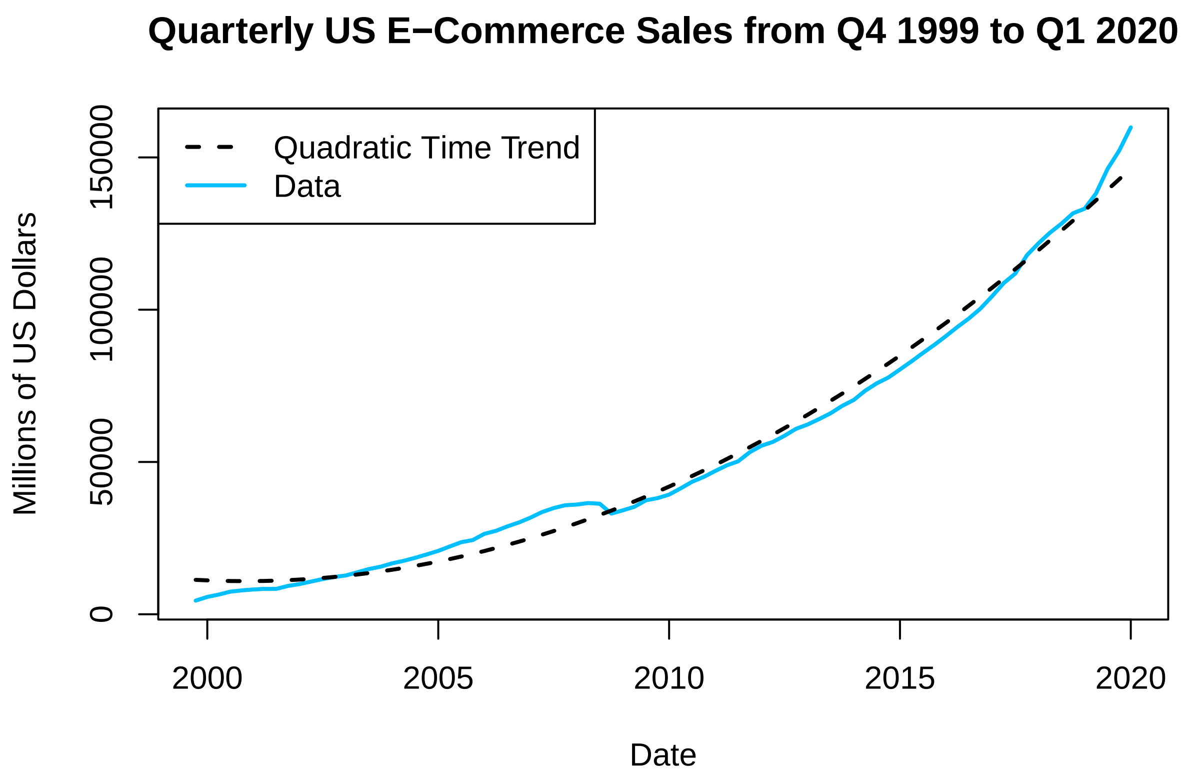
The pattern in our residuals reflects the fact that E-commerce sales appear to be rising at an increasing rate over time. So let's try fitting a quadratic trend. We can do this by generating a quadratic time trend and adding it to our regression model:

```
time2 <- time^2
trendmod2 <- lm(formula = ecomdata$ecomsa ~ time + time2)
summary(trendmod2)
```

```
##
## Call:
## lm(formula = ecomdata$ecomsa ~ time + time2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -6818  -2968  -1512   2499  13533
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 11513.5028  1423.9597   8.086 5.94e-12 ***
## time         -242.6205    79.1819  -3.064  0.00299 **
## time2          23.0074     0.9244  24.889  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4194 on 79 degrees of freedom
## Multiple R-squared:  0.9901, Adjusted R-squared:  0.9898
## F-statistic:  3939 on 2 and 79 DF,  p-value: < 2.2e-16
```

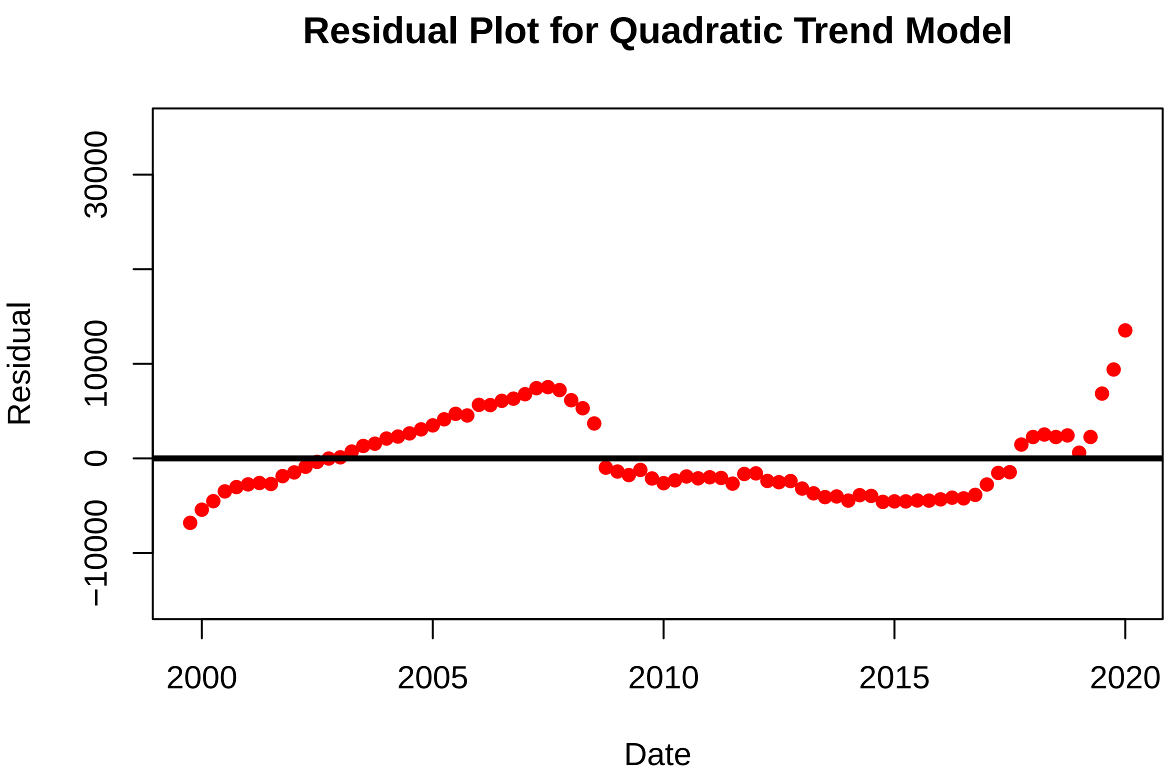
Plotting the estimated quadratic trend, we obtain:

```r
plot(ecomdata$date,ecomdata$ecomsa,
     main = "Quarterly US E-Commerce Sales from Q4 1999 to Q1 2020",
     xlab = "Date",
     ylab = "Millions of US Dollars",
     type = "l",
     lwd = 2.0,
     col = "deepskyblue")
lines(ecomdata$date, predict(trendmod2), type = 'l', lty = "dashed", lwd = 2.0)

legend(x = "topleft",
       legend = c("Quadratic Time Trend", "Data"),
       lty = c("dashed", "solid"),
       lwd = 2.0,
       col = c("black", "deepskyblue"))
```



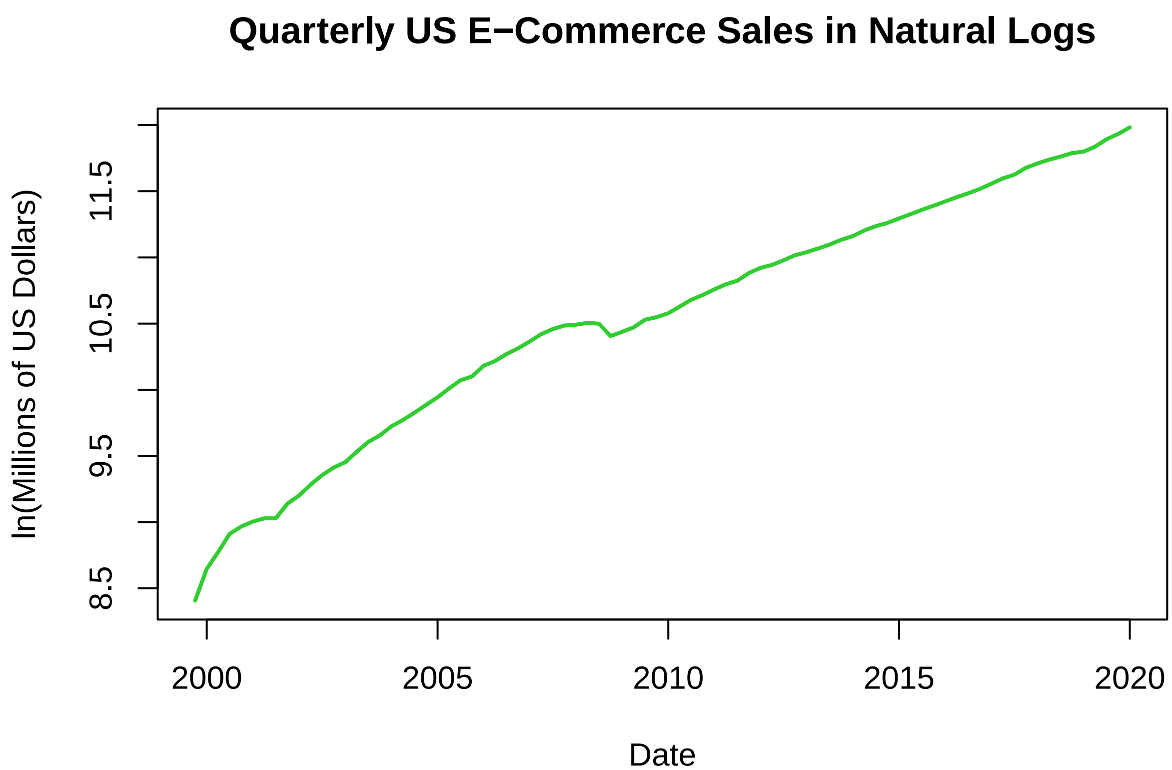
Looking at the residuals we can see that we are doing a little bit better. The magnitude of the residuals is much smaller:

```r
trendmod2.res <- resid(trendmod2)

plot(ecomdata$date, trendmod2.res,
     main = "Residual Plot for Quadratic Trend Model",
     ylim = c(-15000,35000),
     xlab = "Date",
     ylab = "Residual",
     col = "red",
     pch =16)
abline(0,0, lwd = 3)
```



Let's now try an exponential trend model. We can estimate such a model using linear regression if we transform the original data by taking natural logs. Let's generate a plot of our transformed data:

```
ln.ecomsa <- log(ecomdata$ecomsa)

plot(ecomdata$date,ln.ecomsa,
     main = "Quarterly US E-Commerce Sales in Natural Logs",
     xlab = "Date",
     ylab = "ln(Millions of US Dollars)",
     type = "l",
     lwd = 2.0,
     col = "limegreen")
```



Then we can proceed to estimate a linear trend model on this transformed data

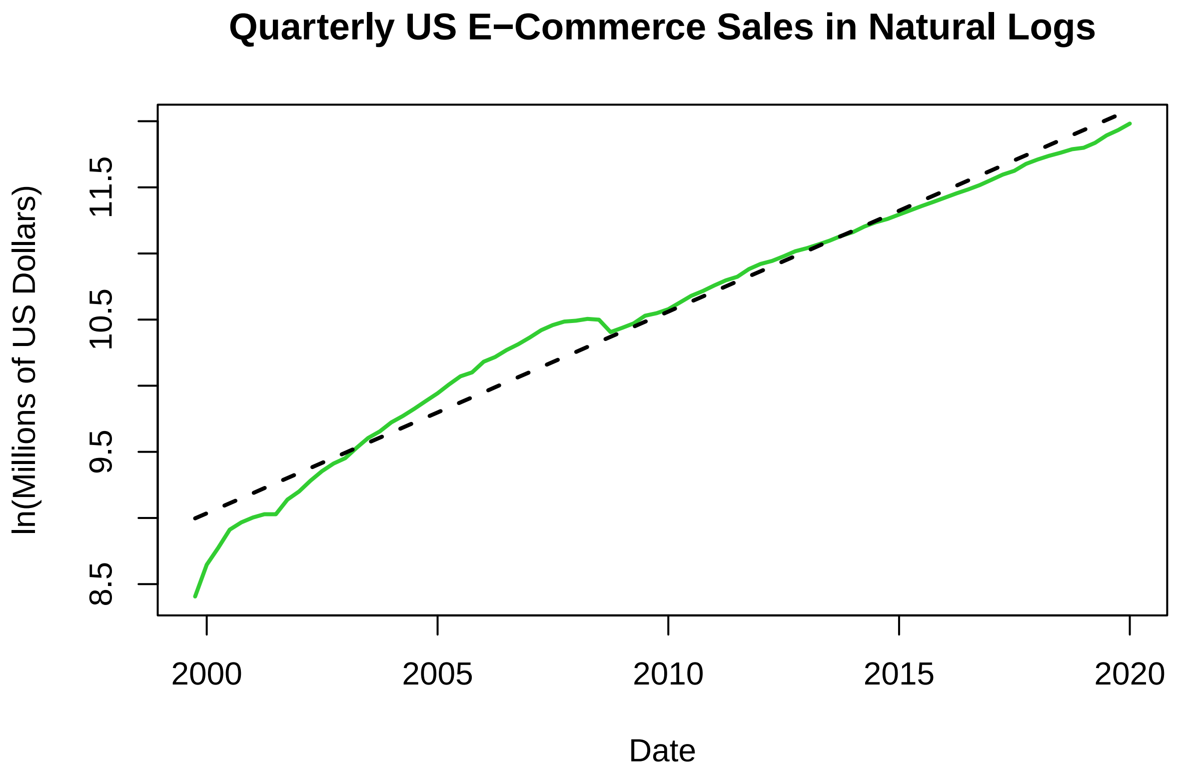
```
trendmod3 <- lm(formula = ln.ecomsa ~ time)
summary(trendmod3)
```

```
##
## Call:
## lm(formula = ln.ecomsa ~ time)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.59060 -0.07227  0.00180  0.05281  0.27938
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 8.9589548  0.0338976  264.29   <2e-16 ***
## time        0.0381316  0.0007095   53.74   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1521 on 80 degrees of freedom
## Multiple R-squared:  0.973,  Adjusted R-squared:  0.9727
## F-statistic:  2888 on 1 and 80 DF,  p-value: < 2.2e-16
```

Plotting the fitted trend on our transformed data, we obtain:

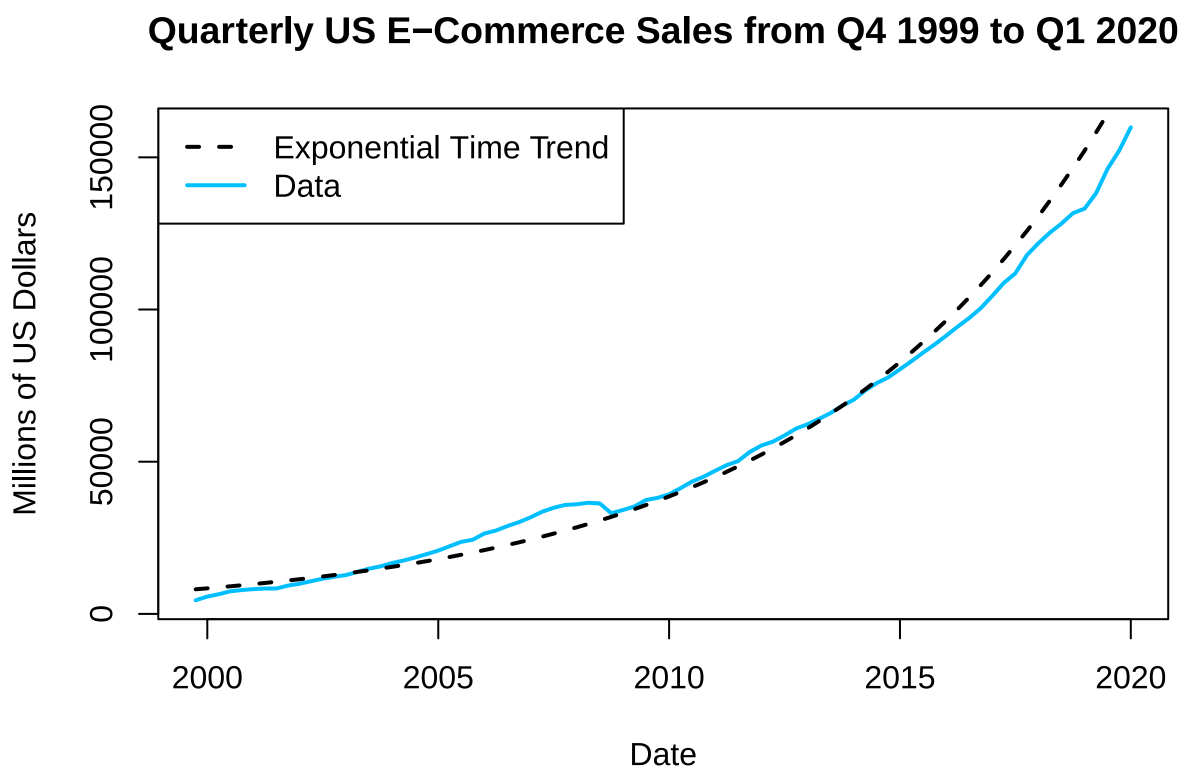
```
plot(ecomdata$date,ln.ecomsa,
     main = "Quarterly US E-Commerce Sales in Natural Logs",
     xlab = "Date",
     ylab = "ln(Millions of US Dollars)",
     type = "l",
```

```
     lwd = 2.0,
     col = "limegreen")
lines(ecomdata$date, predict(trendmod3),type = 'l', lty = "dashed", lwd = 2.0)
```



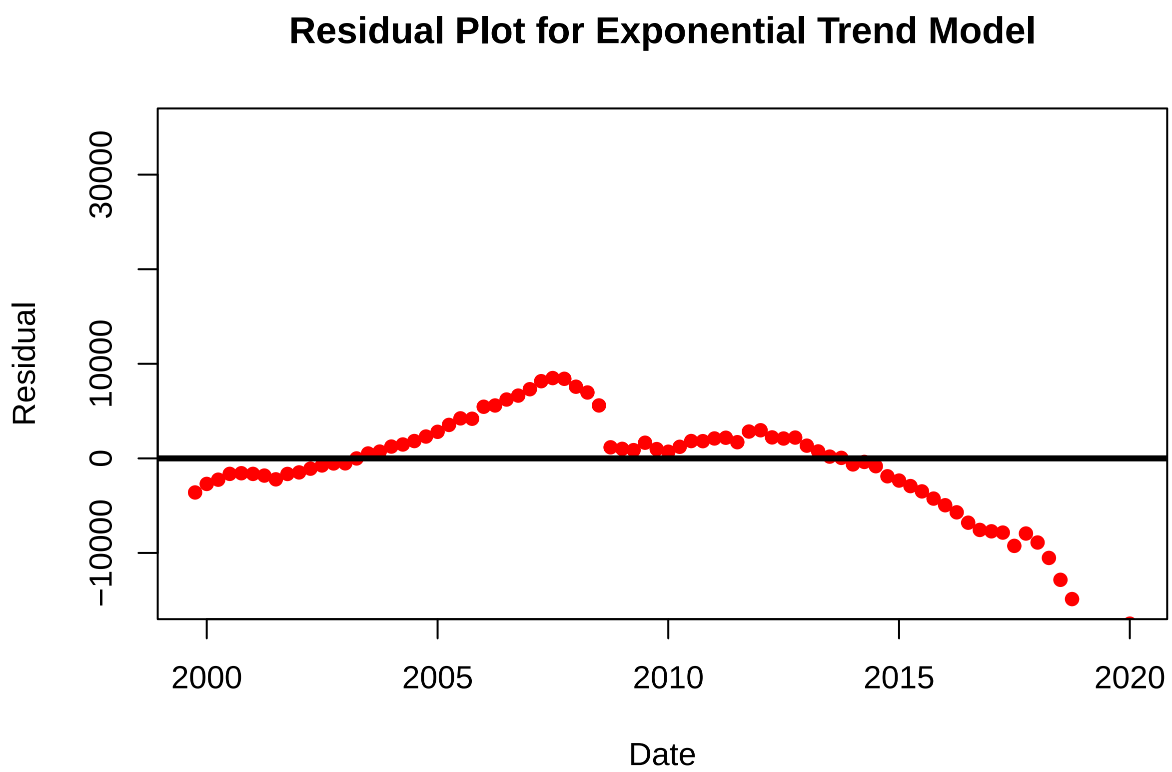
Now let's suppose that we wanted to plot our estimated exponential trend model back in the original scale of the data. All we need to do is extract the coefficients from our estimated model and plug them into our original exponential equation:

```
c3 <- coef(trendmod3) # This picks out the coefficients from the linear regression we just computed
exp.trend <- exp(c3[1])*exp(c3[2]*time) # This draws out the exponential trend

plot(ecomdata$date,ecomdata$ecomsa,
     main = "Quarterly US E-Commerce Sales from Q4 1999 to Q1 2020",
     xlab = "Date",
     ylab = "Millions of US Dollars",
     type = "l",
     lwd = 2.0,
     col = "deepskyblue")
lines(ecomdata$date, exp.trend, type = 'l', lty = "dashed", lwd = 2.0)

legend(x = "topleft",
       legend = c("Exponential Time Trend", "Data"),
       lty = c("dashed", "solid"),
       lwd = 2.0,
       col = c("black", "deepskyblue"))
```



To obtain the residuals scaled in the original units of the variable, we simply compute:

```
trendmod3.res <- ecomdata$ecomsa - exp.trend

plot(ecomdata$date, trendmod3.res,
     main = "Residual Plot for Exponential Trend Model",
     ylim = c(-15000,35000),
     xlab = "Date",
     ylab = "Residual",
     col = "red",
     pch =16)
abline(0,0, lwd = 3)
```



Having estimated our trend models, let's compute forecasts the next 12 quarters. We can do this using the **predict()** function. As a first step, let's define the forecast horizon (i.e., the set of time periods for which we would like to compute the forecasts). We will call this object **horizon**. Note that the **predict()** requires that **horizon** be formatted as a data frame:

```
h <- 12
horizon <- data.frame(time = seq(from = T+1, to = T+h), time2 = seq(from = T+1, to = T+h)^2)
```

Having defined the forecast horizon, we can proceed to compute our point forecasts as:

```
forecastmod1 <- predict(trendmod1, newdata = horizon)
forecastmod1
```

```
##        1        2        3        4        5        6        7        8
## 123139.2 124806.2 126473.2 128140.2 129807.2 131474.2 133141.2 134808.2
##        9       10       11       12
## 136475.2 138142.2 139809.2 141476.1
```

```
forecastmod2 <- predict(trendmod2, newdata = horizon)
forecastmod2
```

```
##        1        2        3        4        5        6        7        8
## 149873.8 153473.4 157119.0 160810.7 164548.3 168332.0 172161.7 176037.4
##        9       10       11       12
## 179959.1 183926.8 187940.6 192000.3
```

```
forecastmod3 <- predict(trendmod3, newdata = horizon)
forecastmod3
```

```
##        1        2        3        4        5        6        7        8
## 12.12388 12.16201 12.20014 12.23827 12.27640 12.31454 12.35267 12.39080
##        9       10       11       12
## 12.42893 12.46706 12.50519 12.54333
```

Now that we have our point forecasts, let's plot them alongside our data. First, let's create a new date vector that includes our original dates plus the dates spanned by our forecast horizon:

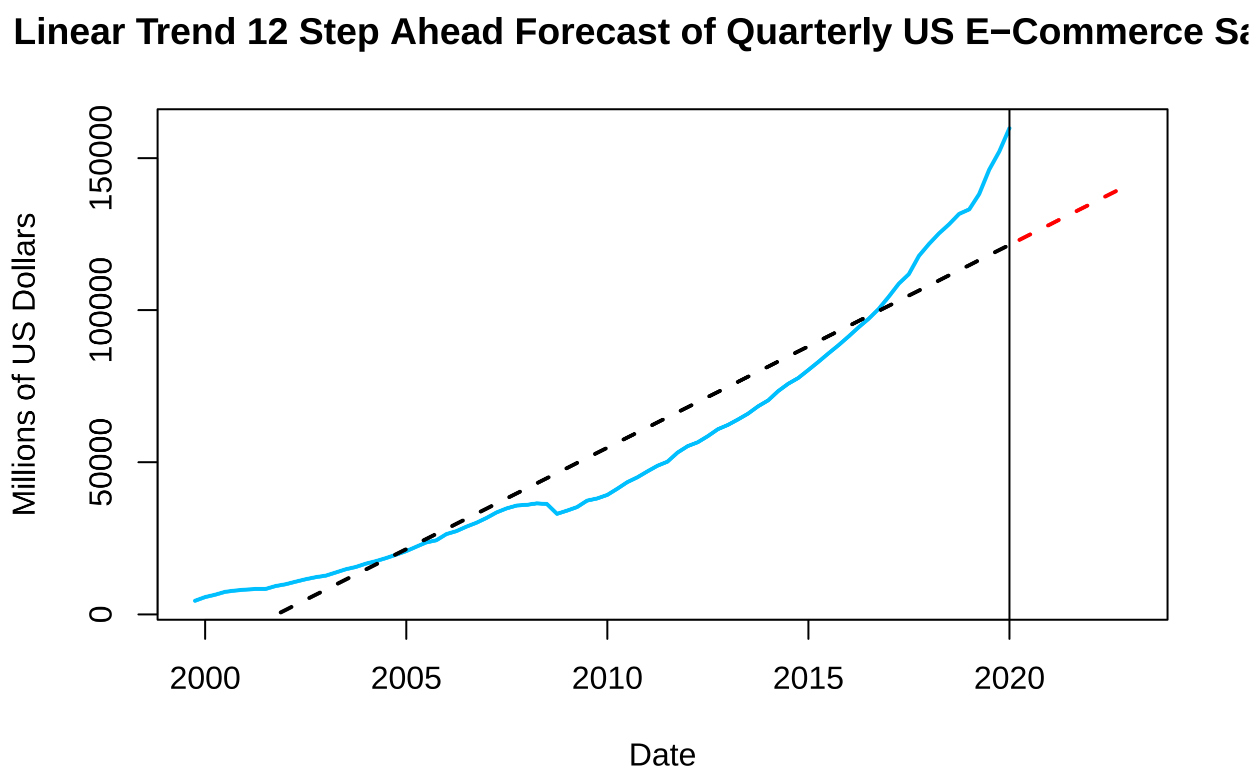
```
date.for <- seq(ecomdata$date[T], by = "quarter", length.out = h+1)
date.for <- date.for[1:h+1]

datenew <- c(ecomdata$date,date.for)
ecomsanew <- c(ecomdata$ecomsa,rep(NA,h))
```

Then, we will need to ensure that our point forecasts are aligned with the correct dates

```
forecastmod1 <- c(rep(NA,T),forecastmod1)
forecastmod2 <- c(rep(NA,T),forecastmod2)
forecastmod3 <- c(rep(NA,T),forecastmod3)
```

Then, we can plot our forecasts along with our original data:

```
plot(datenew,ecomsanew,
     main = "Linear Trend 12 Step Ahead Forecast of Quarterly US E-Commerce Sales",
     xlab = "Date",
     ylab = "Millions of US Dollars",
     type = "l",
     lwd = 2.0,
     col = "deepskyblue")
lines(datenew,forecastmod1, type = 'l', lty = 'dashed', lwd = 2.0, col = "red")
lines(ecomdata$date, predict(trendmod1), type = 'l', lty = "dashed", lwd = 2.0)
abline (v = datenew[T])
```



Finally, let's compute the AIC and BIC for our linear and quadratic trend models:

```
AIC(trendmod1)
```

```
## [1] 1782.343
```

```
BIC(trendmod1)
```

```
## [1] 1789.563
```

```
AIC(trendmod2)
```

```
## [1] 1605.632
```

```
BIC(trendmod2)
```

```
## [1] 1615.258
```

An important thing to note is that we cannot compare these values to the AIC and BIC from our exponential trend model. This is because we estimated **trendmod3** using our e-commerce data scaled in logs. This will mean that the log likelihood (and thus the AIC and BIC) of **trendmod3** will be scaled differently to **trendmod1** and **trendmod2**. To get around this issue, we will need to estimate the exponential trend model in the original scale of the data using nonlinear least squares:

```
ecomsa <- ecomdata$ecomsa
trendmod3.b <- nls(formula = ecomsa ~ a*exp(b*time), start=list(a=8000, b=0.04))
summary(trendmod3.b)
```

```
##
## Formula: ecomsa ~ a * exp(b * time)
##
## Parameters:
##    Estimate Std. Error t value Pr(>|t|)
## a 9.969e+03  2.119e+02   47.05   <2e-16 ***
## b 3.363e-02  3.064e-04  109.77   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2622 on 80 degrees of freedom
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 3.175e-06
```

```
AIC(trendmod3.b)
```

```
## [1] 1527.634
```

```
BIC(trendmod3.b)
```

```
## [1] 1534.855
```