Week 3 - Forecast Intervals, Seasonality & Decompositions

Jonathan Thong

2023-03-10

Confidence Intervals vs. Prediction Intervals

Let's start by making sure that our environment is clear:

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

Then let's import the quarterly E-commerce sales data that we have been working with so far:

```
ecomdata <- read.csv("ECOMSA.csv")
ecomdata <- na.omit(ecomdata)
ecomdata$date <- as.Date(ecomdata$date, "%d/%m/%Y")
```

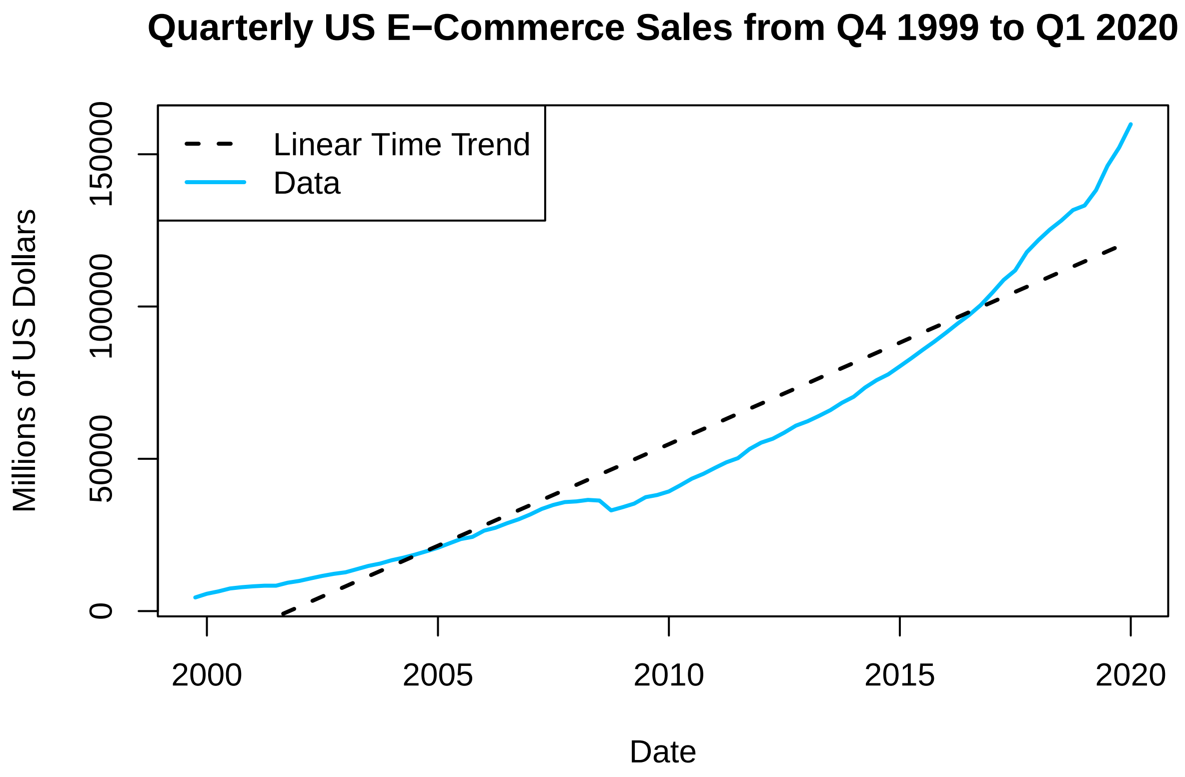
Let's work with the linear trend model that we estimated in the previous week:

```
T <- length(ecomdata$date)
time <- seq(1,T)
trendmod1 <- lm(formula = ecomdata$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
```

```
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")
```

```r
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"))
```



In the previous week, we saw how we could generate point forecasts using the **predict()** function. As it turns out, we can compute confidence and forecast intervals using the same function by simply adding in an extra argument. However, to make sure that we understand what the function is doing, we will first compute these intervals directly using the formulas provided in the lecture slides, then we will show how these are equivalent to the intervals computed by **predict()**.

Recall that the formula for the confidence interval for a simple linear regression is given by:

$$\hat{y}_h \pm t_{(1-\frac{\alpha}{2},T-2)}\sqrt{\frac{\sum_{t=1}^{T}(y_t-\hat{y}_t)^2}{T-2}\left(\frac{1}{n}+\frac{(x_h-\overline{x})^2}{\sum_{t=1}^{T}(x_t-\overline{x})}\right)}$$

While the formula for the prediction interval for a simple linear regression is given by:

$$\hat{y}_h \pm t_{(1-\frac{\alpha}{2},T-2)}\sqrt{\frac{\sum_{t=1}^{T}(y_t-\hat{y}_t)^2}{T-2}\left(1+\frac{1}{n}+\frac{(x_h-\overline{x})^2}{\sum_{t=1}^{T}(x_t-\overline{x})}\right)}$$

So let's compute these intervals direction for our 12-step horizon. First, let's map the objects in the above equations to the things that we have computed so far:

```r
h <- 12
y <- ecomdata$ecomsa
```

```r
x <- time

yhat <- trendmod1$fitted.values

b0 <- trendmod1$coefficients[1]
b1 <- trendmod1$coefficients[2]

xh <- data.frame(time = seq(from = T+1, to = T+h))
yh <- b0 + b1*xh
```

Then, let's compute the two components inside the radical (i.e., the symbol of the square root):

```r
mse <- sum((y-yhat)^2)/(T-2)

# We use this to compute the confidence interval
ci.se.fit <- (1/T + (xh - mean(x))^2/(sum((x-mean(x))^2)))

# We use this to compute the prediction interval
pi.se.fit <- (1+(1/T) + (xh - mean(x))^2/(sum((x-mean(x))^2)))
```

Setting our confidence level to $(1-\alpha) = 0.95$, we will use the **qt()** function to compute the value of a Student t with T-2 degrees of freedom that corresponds to a tail probability of $\frac{\alpha}{2} = 0.025$:

```r
t.val <- qt(0.975, T-2)
```

Putting it all together, we obtain the upper and lower bounds of our confidence and prediction intervals:

```r
ci.upper <- yh + t.val*sqrt(mse*ci.se.fit)
ci.lower <- yh - t.val*sqrt(mse*ci.se.fit)

pi.upper <- yh + t.val*sqrt(mse*pi.se.fit)
pi.lower <- yh - t.val*sqrt(mse*pi.se.fit)
```

Then, we can collect all of these objects into a data frame for a neat presentation:

```r
# Confidence interval
ci <- data.frame(yh, ci.lower, ci.upper)
colnames(ci) = c('yh', 'lower', 'upper')
print(ci)
```

```
##          yh    lower    upper
## 1  123139.2 117642.6 128635.9
## 2  124806.2 119209.3 130403.1
## 3  126473.2 120775.5 132170.9
## 4  128140.2 122341.2 133939.2
## 5  129807.2 123906.4 135708.0
## 6  131474.2 125471.1 137477.3
## 7  133141.2 127035.3 139247.1
## 8  134808.2 128599.1 141017.2
## 9  136475.2 130162.5 142787.8
## 10 138142.2 131725.5 144558.8
## 11 139809.2 133288.1 146330.2
## 12 141476.1 134850.4 148101.9
```

```r
# Prediction interval
pi <- data.frame(yh,pi.lower,pi.upper)
colnames(pi) = c('yh', 'lower', 'upper')
```

```
print(pi)
```

```
##          yh    lower    upper
## 1  123139.2  97874.2 148404.3
## 2  124806.2  99519.2 150093.3
## 3  126473.2 101163.7 151782.8
## 4  128140.2 102807.7 153472.7
## 5  129807.2 104451.2 155163.2
## 6  131474.2 106094.2 156854.2
## 7  133141.2 107736.6 158545.7
## 8  134808.2 109378.6 160237.7
## 9  136475.2 111020.1 161930.2
## 10 138142.2 112661.1 163623.2
## 11 139809.2 114301.7 165316.7
## 12 141476.1 115941.7 167010.6
```

Having computed these intervals directly from the formulas, let's see how they compare to the intervals produced by the **predict()** function. To generate intervals along with our point forecasts, we simply have to specify the interval in the argument of the function. First, let's compute the confidence interval:

```
predict(trendmod1, newdata = xh, interval = 'confidence')
```

```
##         fit      lwr      upr
## 1  123139.2 117642.6 128635.9
## 2  124806.2 119209.3 130403.1
## 3  126473.2 120775.5 132170.9
## 4  128140.2 122341.2 133939.2
## 5  129807.2 123906.4 135708.0
## 6  131474.2 125471.1 137477.3
## 7  133141.2 127035.3 139247.1
## 8  134808.2 128599.1 141017.2
## 9  136475.2 130162.5 142787.8
## 10 138142.2 131725.5 144558.8
## 11 139809.2 133288.1 146330.2
## 12 141476.1 134850.4 148101.9
```

Then, let's compute the prediction interval:

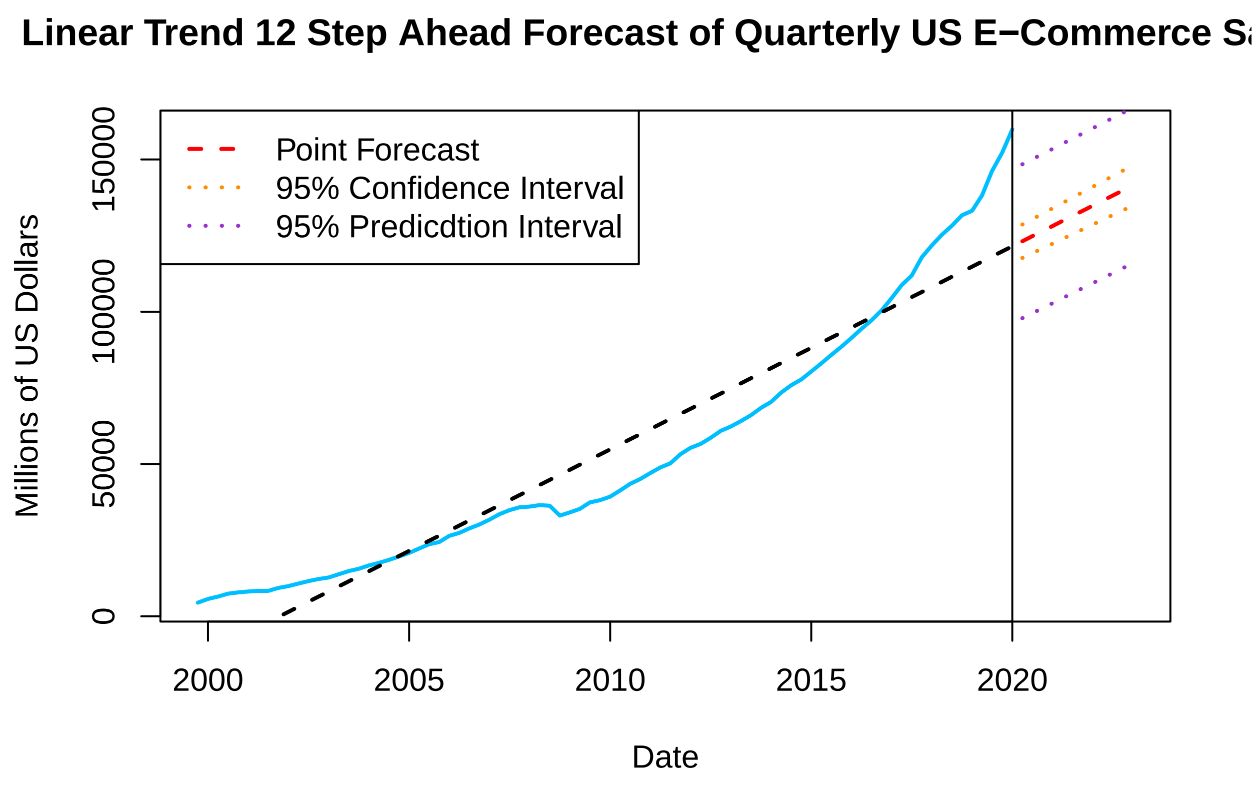
```
predict(trendmod1, newdata = xh, interval = 'prediction')
```

```
##         fit      lwr      upr
## 1  123139.2  97874.2 148404.3
## 2  124806.2  99519.2 150093.3
## 3  126473.2 101163.7 151782.8
## 4  128140.2 102807.7 153472.7
## 5  129807.2 104451.2 155163.2
## 6  131474.2 106094.2 156854.2
## 7  133141.2 107736.6 158545.7
## 8  134808.2 109378.6 160237.7
## 9  136475.2 111020.1 161930.2
## 10 138142.2 112661.1 163623.2
## 11 139809.2 114301.7 165316.7
## 12 141476.1 115941.7 167010.6
```

Note here that the predict function defaults to a 95% interval. If you wish to set another level of confidence you will have to specify it in the function.

As you can see, both methods produce the same values. So now we know how the predict function works! Having computed the confidence and prediction intervals, the final step is to plot them along with our point forecasts!

```r
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))

point.forecast <- predict(trendmod1, newdata = xh)
point.forecast <- c(rep(NA,T),point.forecast)

ci.forecast <- data.frame(predict(trendmod1, newdata = xh, interval = 'confidence'))
pi.forecast <- data.frame(predict(trendmod1, newdata = xh, interval = 'prediction'))

ci.forecast.lwr <- c(rep(NA,T),ci.forecast$lwr)
ci.forecast.upr <- c(rep(NA,T),ci.forecast$upr)

pi.forecast.lwr <- c(rep(NA,T),pi.forecast$lwr)
pi.forecast.upr <- c(rep(NA,T),pi.forecast$upr)

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,point.forecast, type = 'l', lty = 'dashed', lwd = 2.0, col = "red")
lines(datenew,ci.forecast.lwr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorange")
lines(datenew,ci.forecast.upr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorange")
lines(datenew,pi.forecast.lwr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorchid")
lines(datenew,pi.forecast.upr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorchid")
lines(ecomdata$date, predict(trendmod1), type = 'l', lty = "dashed", lwd = 2.0)
abline (v = datenew[T])

legend(x = "topleft",
       legend = c("Point Forecast", "95% Confidence Interval", "95% Predicdtion Interval"),
       lty = c("dashed", "dotted", "dotted"),
       lwd = 2.0,
       col = c("red", "darkorange", "darkorchid"))
```



Seasonality

Let us now proceed to look at ways in which to model seasonal fluctuations. We will be working with a new dataset so it would be wise of us to clear our environment to ensure that we don't mix up any of the variables we have created and worked with in the previous section.

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

We will be working with a monthly time series of US housing starts starting from January 1946 to November 1995 contained in the file *ushstarts.csv*. Each data point represents the number of new residential constructions (in tens of thousands) begun during the month.

```
y <- read.csv("ushstarts.csv")
head(y)
```

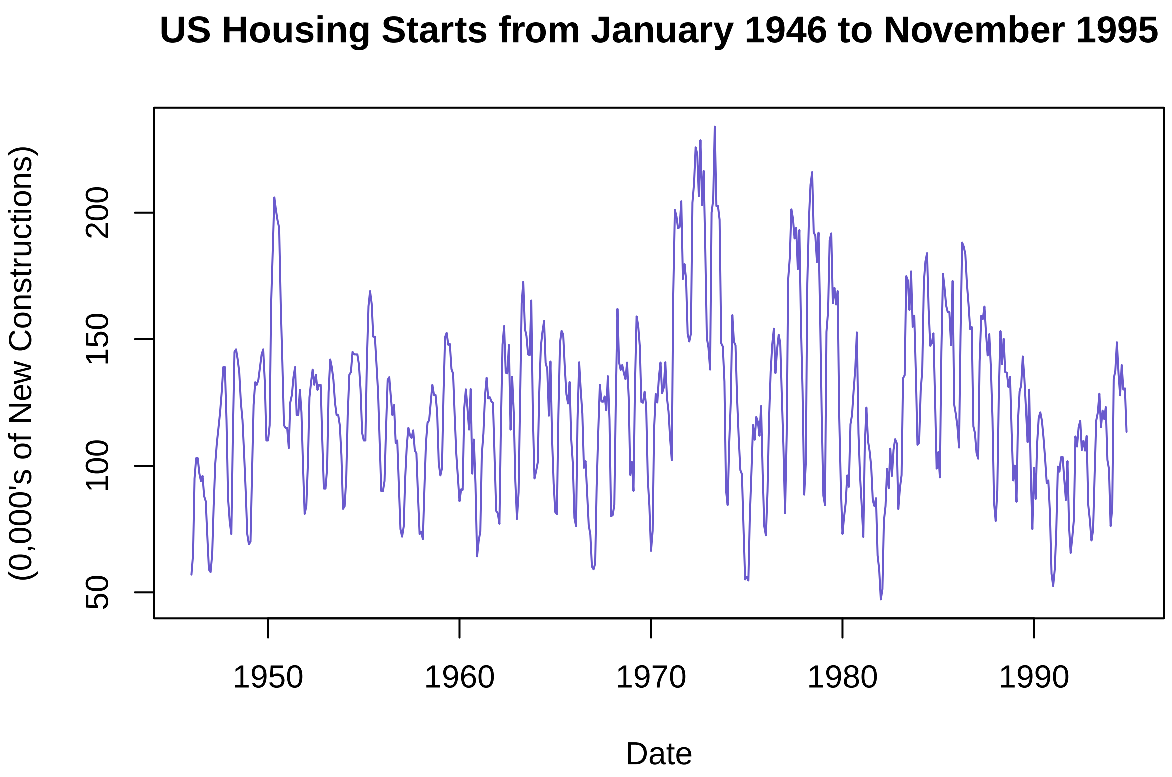
```
##   hstarts
## 1      57
## 2      65
## 3      95
## 4     103
## 5     103
## 6      97
```

Looking at the data we notice that the csv file does not contain dates. This means that we have to generate the dates ourselves:

```
date <- seq(as.Date("1946/1/1"), as.Date("1994/11/1"), by = "month")
```

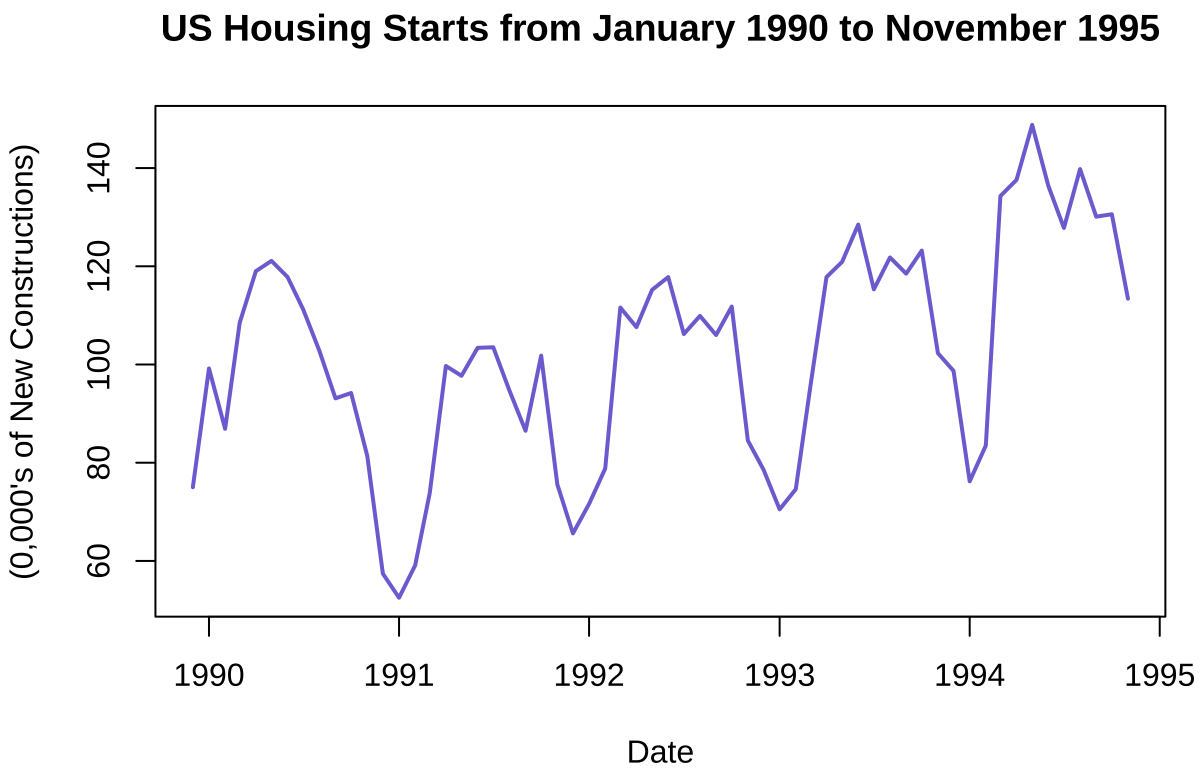
As a first step, let's plot the data

```
plot(date,y$hstarts,
     main = "US Housing Starts from January 1946 to November 1995",
     xlab = "Date",
     ylab = "(0,000's of New Constructions)",
     type = 'l',
     lwd = 1,
     col = "slateblue")
```



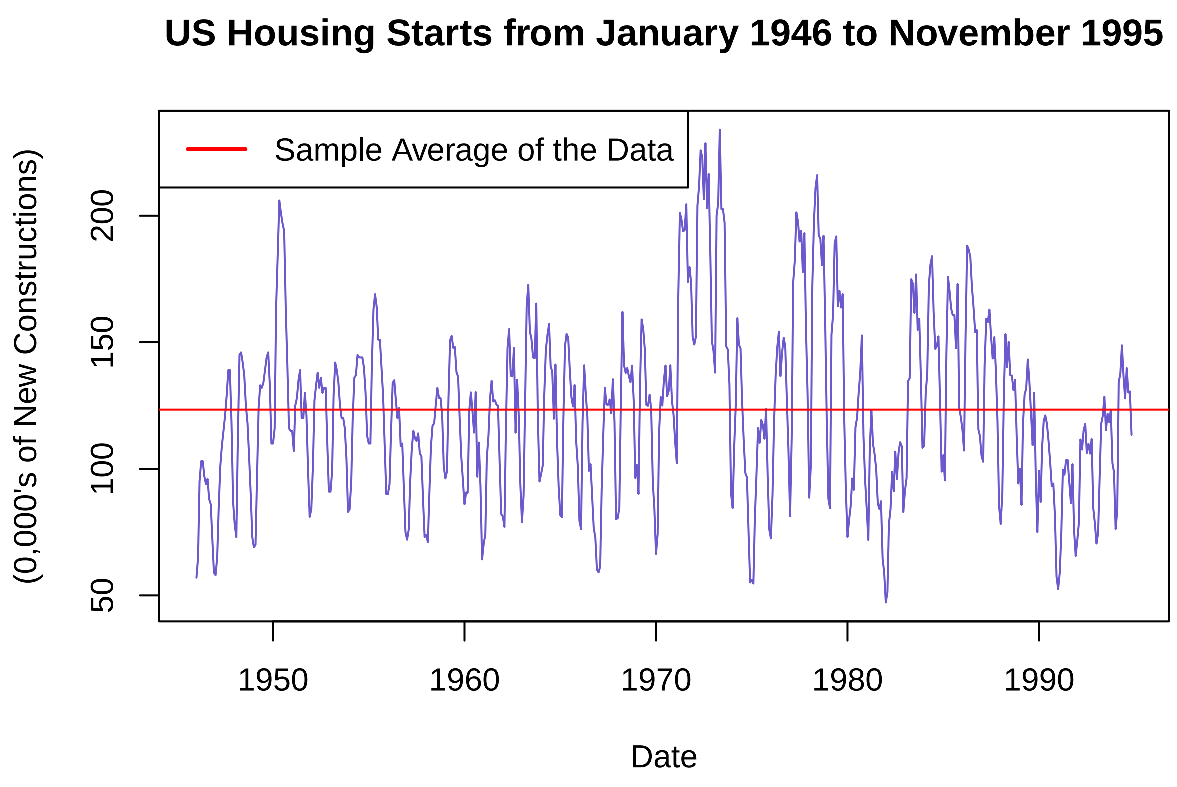
Given the length of the data set, it is a little difficult to discern the seasonal fluctuations that are present. So let's zoom into a narrow subset of the data, say, the last five years:

```
T <- length(date)
time <- seq(1,T)
T_zm <- T - 59

plot(date[T_zm:T],y$hstarts[T_zm:T],
     main = "US Housing Starts from January 1990 to November 1995",
     xlab = "Date",
     ylab = "(0,000's of New Constructions)",
     type = 'l',
     lwd = 2,
     col = "slateblue")
```



Here we can see the seasonal fluctuations clearly. The number of new constructions falls at the same time every year and peak just before the middle of the year. One way to look at these fluctuations is as season specific averages. If the fluctuations are very similar at common points in time (i.e., same month, quarter, day of week), then we can model these season specific averages as seasonal factors.

```
plot(date,y$hstarts,
     main = "US Housing Starts from January 1946 to November 1995",
     xlab = "Date",
     ylab = "(0,000's of New Constructions)",
     type = 'l',
     lwd = 1,
     col = "slateblue")
abline(h = mean(y$hstarts), col = "red")

legend(x = "topleft",
       legend = c("Sample Average of the Data" ),
       lty = c("solid"),
       lwd = 2.0,
       col = c("red"))
```



Let's suppose we would like to model our seasonality at the same frequency as our observations. To create our dummy variables, we use the **rep()** function to create repeating sequences.

```
# First we have to specify the number of times we have to repeat each sequence
s = 12
nyears = ceiling(T/s)

M1 <- rep(c(1,0,0,0,0,0,0,0,0,0,0,0), nyears)
M2 <- rep(c(0,1,0,0,0,0,0,0,0,0,0,0), nyears)
M3 <- rep(c(0,0,1,0,0,0,0,0,0,0,0,0), nyears)
M4 <- rep(c(0,0,0,1,0,0,0,0,0,0,0,0), nyears)
M5 <- rep(c(0,0,0,0,1,0,0,0,0,0,0,0), nyears)
M6 <- rep(c(0,0,0,0,0,1,0,0,0,0,0,0), nyears)
M7 <- rep(c(0,0,0,0,0,0,1,0,0,0,0,0), nyears)
M8 <- rep(c(0,0,0,0,0,0,0,1,0,0,0,0), nyears)
M9 <- rep(c(0,0,0,0,0,0,0,0,1,0,0,0), nyears)
M10 <- rep(c(0,0,0,0,0,0,0,0,0,1,0,0), nyears)
M11 <- rep(c(0,0,0,0,0,0,0,0,0,0,1,0), nyears)
M12 <- rep(c(0,0,0,0,0,0,0,0,0,0,0,1), nyears)

# Since our data does not consist of exactly 49 years of observations, we have
# to make sure our dummy variables conform to the sample.

M1 <- M1[1:T]
M2 <- M2[1:T]
M3 <- M3[1:T]
M4 <- M4[1:T]
M5 <- M5[1:T]
M6 <- M6[1:T]
M7 <- M7[1:T]
```

```
M8 <- M8[1:T]
M9 <- M9[1:T]
M10 <- M10[1:T]
M11 <- M11[1:T]
M12 <- M12[1:T]
```

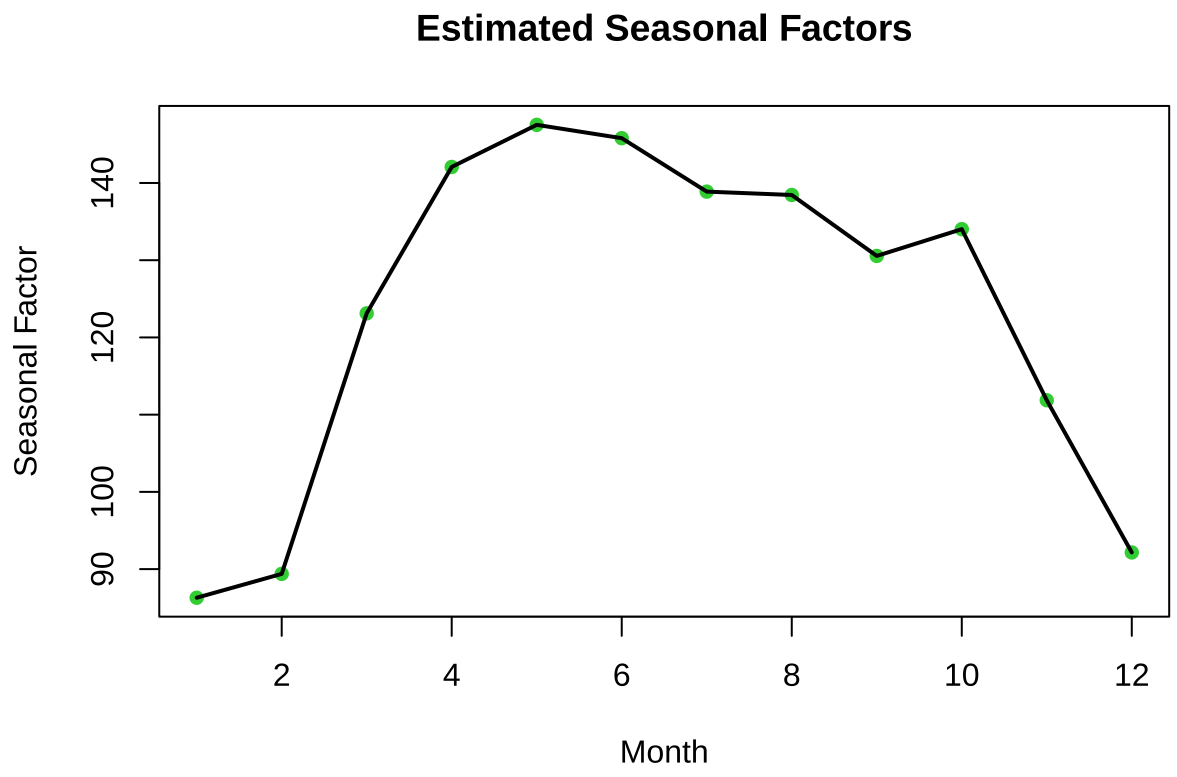
Having created our seasonal dummy variables, we can proceed to estimate the seasonal factors using the **lm()** function:

```
seasmod <- lm(formula = y$hstarts ~ 0 + M1 + M2 + M3 + M4 + M5 + M6 + M7 + M8 + M9 + M10 + M11 + M12)
summary(seasmod)
```

```
##
## Call:
## lm(formula = y$hstarts ~ 0 + M1 + M2 + M3 + M4 + M5 + M6 + M7 +
##     M8 + M9 + M10 + M11 + M12)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -57.976 -17.504  -2.158  13.447  90.155
##
## Coefficients:
##     Estimate Std. Error t value Pr(>|t|)
## M1    86.294      3.952   21.84   <2e-16 ***
## M2    89.382      3.952   22.62   <2e-16 ***
## M3   123.116      3.952   31.16   <2e-16 ***
## M4   142.076      3.952   35.95   <2e-16 ***
## M5   147.527      3.952   37.33   <2e-16 ***
## M6   145.802      3.952   36.90   <2e-16 ***
## M7   138.882      3.952   35.14   <2e-16 ***
## M8   138.445      3.952   35.03   <2e-16 ***
## M9   130.553      3.952   33.04   <2e-16 ***
## M10  134.020      3.952   33.91   <2e-16 ***
## M11  111.865      3.952   28.31   <2e-16 ***
## M12   92.158      3.993   23.08   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 27.66 on 575 degrees of freedom
## Multiple R-squared:  0.9544, Adjusted R-squared:  0.9535
## F-statistic:  1004 on 12 and 575 DF,  p-value: < 2.2e-16
```

We can plot the seasonal factors easily by extracting the coefficient estimates using the **coef()** function:

```
seasonal.f <- coef(seasmod)
plot(seasonal.f, main = "Estimated Seasonal Factors", xlab = "Month", ylab = "Seasonal Factor", col = "
lines(seasonal.f, lwd = 2.0)
```



Having estimated our model, we can proceed to use it generate point and interval forecasts for the 25 months following the end of our sample. First, let's generate our forecasting variables (i.e., the dummy variables that correspond to our forecast horizon):

```
h = 24
nyears.f = h/s + 1

M1.h <- rep(c(1,0,0,0,0,0,0,0,0,0,0,0), nyears.f)
M2.h <- rep(c(0,1,0,0,0,0,0,0,0,0,0,0), nyears.f)
M3.h <- rep(c(0,0,1,0,0,0,0,0,0,0,0,0), nyears.f)
M4.h <- rep(c(0,0,0,1,0,0,0,0,0,0,0,0), nyears.f)
M5.h <- rep(c(0,0,0,0,1,0,0,0,0,0,0,0), nyears.f)
M6.h <- rep(c(0,0,0,0,0,1,0,0,0,0,0,0), nyears.f)
M7.h <- rep(c(0,0,0,0,0,0,1,0,0,0,0,0), nyears.f)
M8.h <- rep(c(0,0,0,0,0,0,0,1,0,0,0,0), nyears.f)
M9.h <- rep(c(0,0,0,0,0,0,0,0,1,0,0,0), nyears.f)
M10.h <- rep(c(0,0,0,0,0,0,0,0,0,1,0,0), nyears.f)
M11.h <- rep(c(0,0,0,0,0,0,0,0,0,0,1,0), nyears.f)
M12.h <- rep(c(0,0,0,0,0,0,0,0,0,0,0,1), nyears.f)

M1.h <- M1.h[12:36]
M2.h <- M2.h[12:36]
M3.h <- M3.h[12:36]
M4.h <- M4.h[12:36]
M5.h <- M5.h[12:36]
M6.h <- M6.h[12:36]
M7.h <- M7.h[12:36]
M8.h <- M8.h[12:36]
M9.h <- M9.h[12:36]
M10.h <- M10.h[12:36]
```

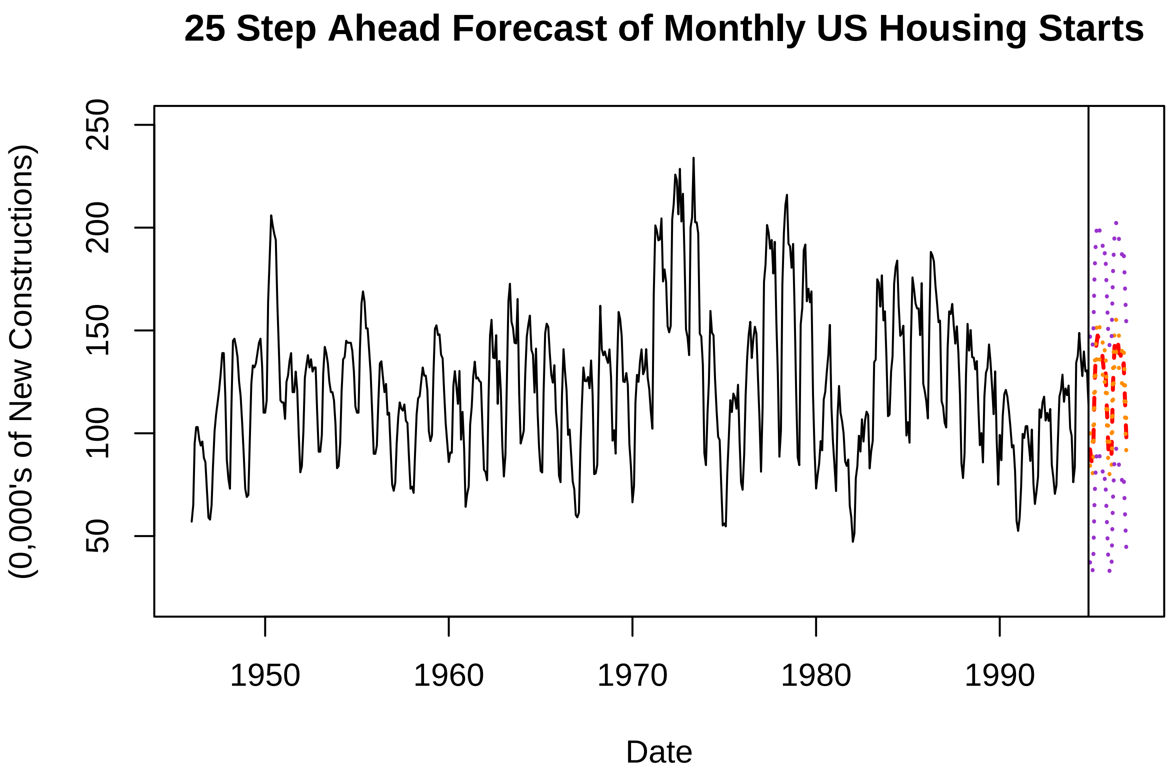
```r
M11.h <- M11.h[12:36]
M12.h <- M12.h[12:36]

xh = data.frame(M1.h,M2.h,M3.h,M4.h,M5.h,M6.h,M7.h,M8.h,M9.h,M10.h,M11.h,M12.h)
colnames(xh) = c('M1','M2','M3','M4','M5','M6','M7','M8','M9','M10','M11','M12')
```

With our appropriately defined predictor variables stored in the data frame **xh**, we can simply apply the **predict()** function to generate our point and interval forecasts:

```r
point.forecast <- predict(seasmod, newdata = xh)
point.forecast <- c(rep(NA,T),point.forecast)

ci.forecast <- data.frame(predict(seasmod, newdata = xh, interval = 'confidence'))
pi.forecast <- data.frame(predict(seasmod, newdata = xh, interval = 'prediction'))

ci.forecast.lwr <- c(rep(NA,T),ci.forecast$lwr)
ci.forecast.upr <- c(rep(NA,T),ci.forecast$upr)

pi.forecast.lwr <- c(rep(NA,T),pi.forecast$lwr)
pi.forecast.upr <- c(rep(NA,T),pi.forecast$upr)
```

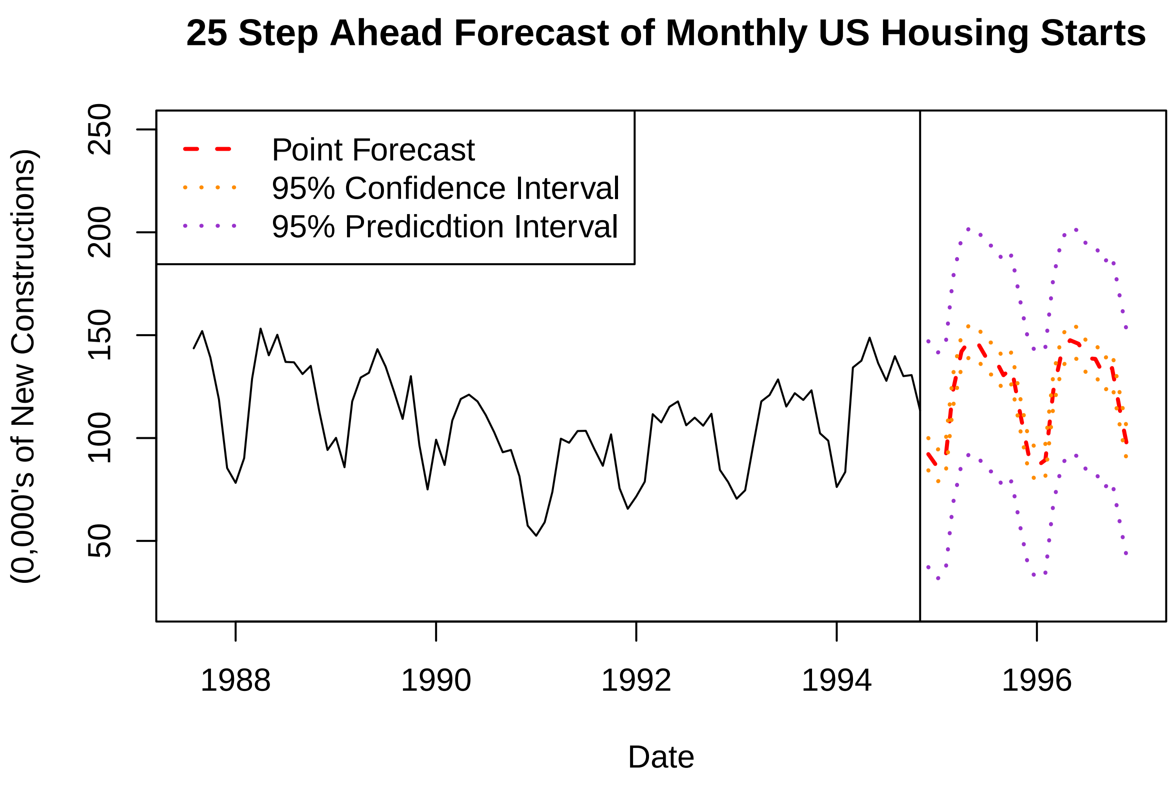
Then, we can plot our forecasts using the same code that we have utilised in the previous section:

```r
datenew <- seq(as.Date("1946/1/1"), as.Date("1996/12/1"), by = "month")
hstartsnew <- c(y$hstarts, rep(NA,25))

plot(datenew,hstartsnew,
     main = " 25 Step Ahead Forecast of Monthly US Housing Starts",
     xlab = "Date",
     ylab = "(0,000's of New Constructions)",
     ylim = c(20,250),
     type = "l",
     lwd = 1,
     col = "black")
lines(datenew,point.forecast, type = 'l', lty = 'dashed', lwd = 2.0, col = "red")
lines(datenew,ci.forecast.lwr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorange")
lines(datenew,ci.forecast.upr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorange")
lines(datenew,pi.forecast.lwr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorchid")
lines(datenew,pi.forecast.upr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorchid")
abline (v = datenew[T])
```



Notice that the forecasts are not very legible due to the length of our sample. We may wish to restrict our attention to the end of the sample:

```
plot(datenew[500:612],hstartsnew[500:612],
     main = " 25 Step Ahead Forecast of Monthly US Housing Starts",
     xlab = "Date",
     ylab = "(0,000's of New Constructions)",
     ylim = c(20,250),
     type = "l",
     lwd = 1,
     col = "black")
lines(datenew,point.forecast, type = 'l', lty = 'dashed', lwd = 2.0, col = "red")
lines(datenew,ci.forecast.lwr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorange")
lines(datenew,ci.forecast.upr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorange")
lines(datenew,pi.forecast.lwr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorchid")
lines(datenew,pi.forecast.upr, type = 'l', lty = 'dotted', lwd =2.0, col = "darkorchid")
abline (v = datenew[T])

legend(x = "topleft",
       legend = c("Point Forecast", "95% Confidence Interval", "95% Predicdtion Interval"),
       lty = c("dashed", "dotted", "dotted"),
       lwd = 2.0,
       col = c("red", "darkorange", "darkorchid"))
```



Smoothing Using Moving Averages and Classical Decomposition

Let's stick with our housing starts data and use it to demonstrate some smoothing using moving averages. To compute moving averages, we will use the **ma()** function in the **forecast** package. So please make sure to install (by running the line **install.packages("forecast")** and load it by running the line:

```
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

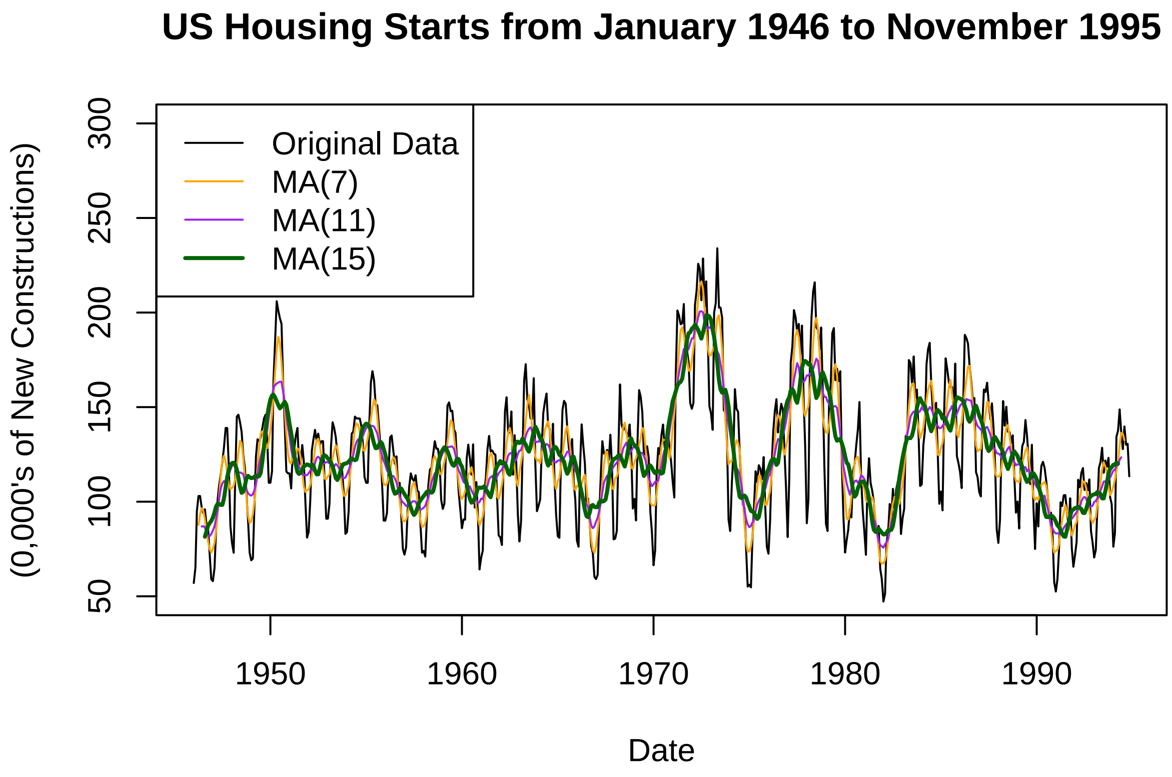
Then, we can compute the moving averages using **ma()** in the following way:

```
hstarts.ma7 <- ma(y$hstarts, order = 7)
hstarts.ma11 <- ma(y$hstarts, order = 11)
hstarts.ma15 <- ma(y$hstarts,order = 15)
```

Plotting these moving averages against the orginal data, we obtain:

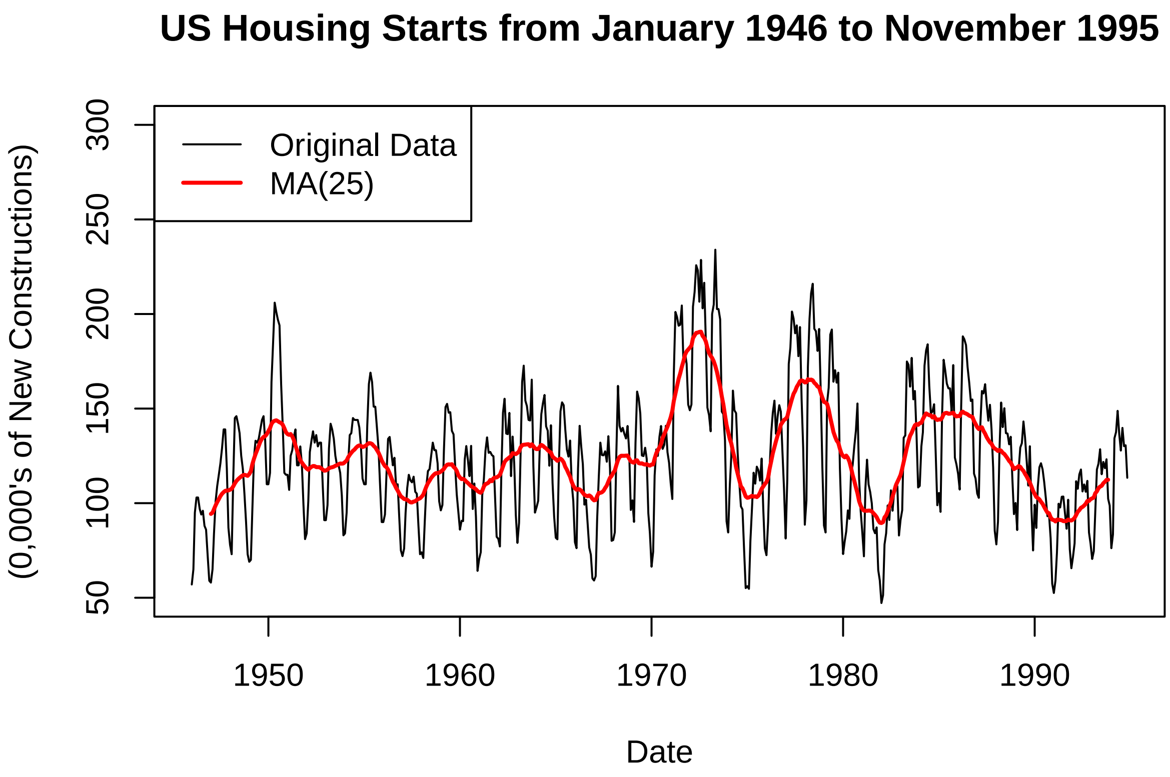
```
plot(date,y$hstarts,
     main = "US Housing Starts from January 1946 to November 1995",
     xlab = "Date",
     ylab = "(0,000's of New Constructions)",
     ylim = c(50,300),
     type = 'l',
     lwd = 1,
     col = "black")
lines(date, hstarts.ma7,lwd = 1, col = "orange")
lines(date, hstarts.ma11,lwd = 1, col = "purple")
lines(date, hstarts.ma15,lwd = 2, col = "darkgreen")
```

```
legend(x = "topleft",
       legend = c("Original Data", "MA(7)", "MA(11)", "MA(15)" ),
       lty = c("solid","solid","solid","solid"),
       lwd = c(1.0,1.0,1.0,2.0),
       col = c("black","orange","purple","darkgreen"))
```



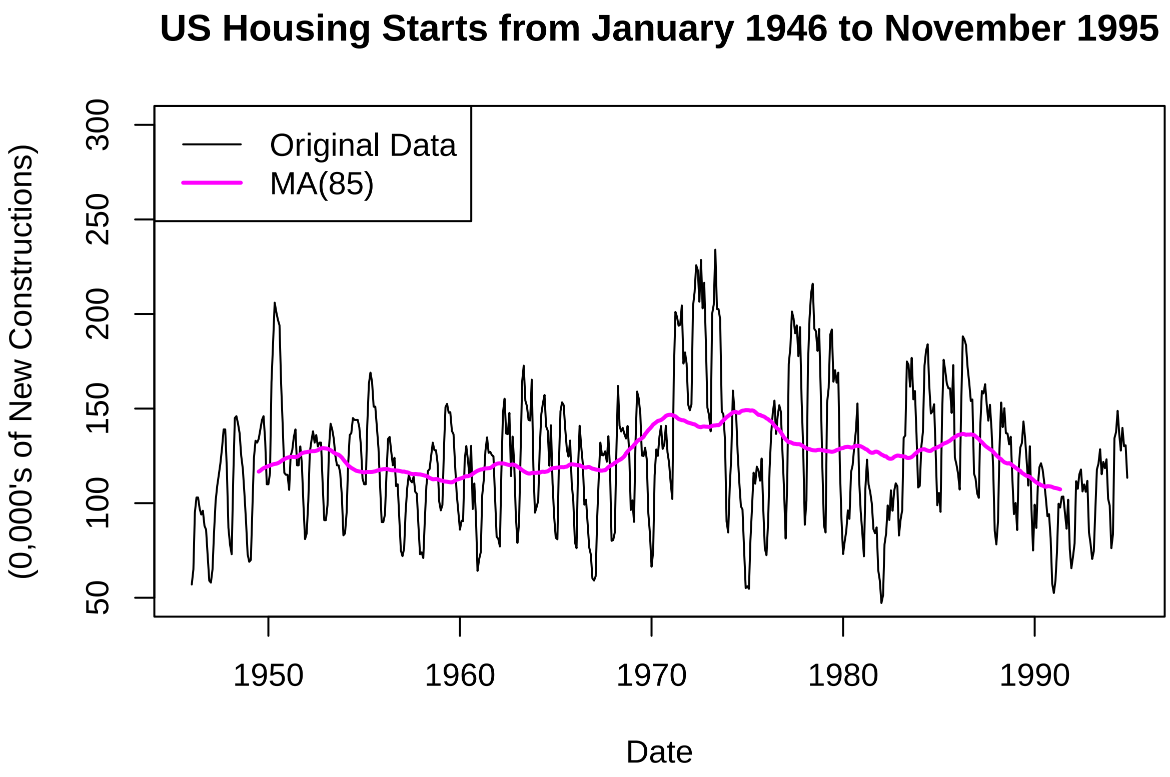
Notice that the smoothed series at $m = 15$ still contain a fair amount of short run fluctuations. Let's see how things look when we set $m = 25$

```
hstarts.ma25 <- ma(y$hstarts, order = 25)

plot(date,y$hstarts,
     main = "US Housing Starts from January 1946 to November 1995",
     xlab = "Date",
     ylab = "(0,000's of New Constructions)",
     ylim = c(50,300),
     type = 'l',
     lwd = 1,
     col = "black")
lines(date, hstarts.ma25,lwd = 2, col = "red")

legend(x = "topleft",
       legend = c("Original Data", "MA(25)" ),
       lty = c("solid", "solid"),
       lwd = c(1.0,2.0),
       col = c("black","red"))
```



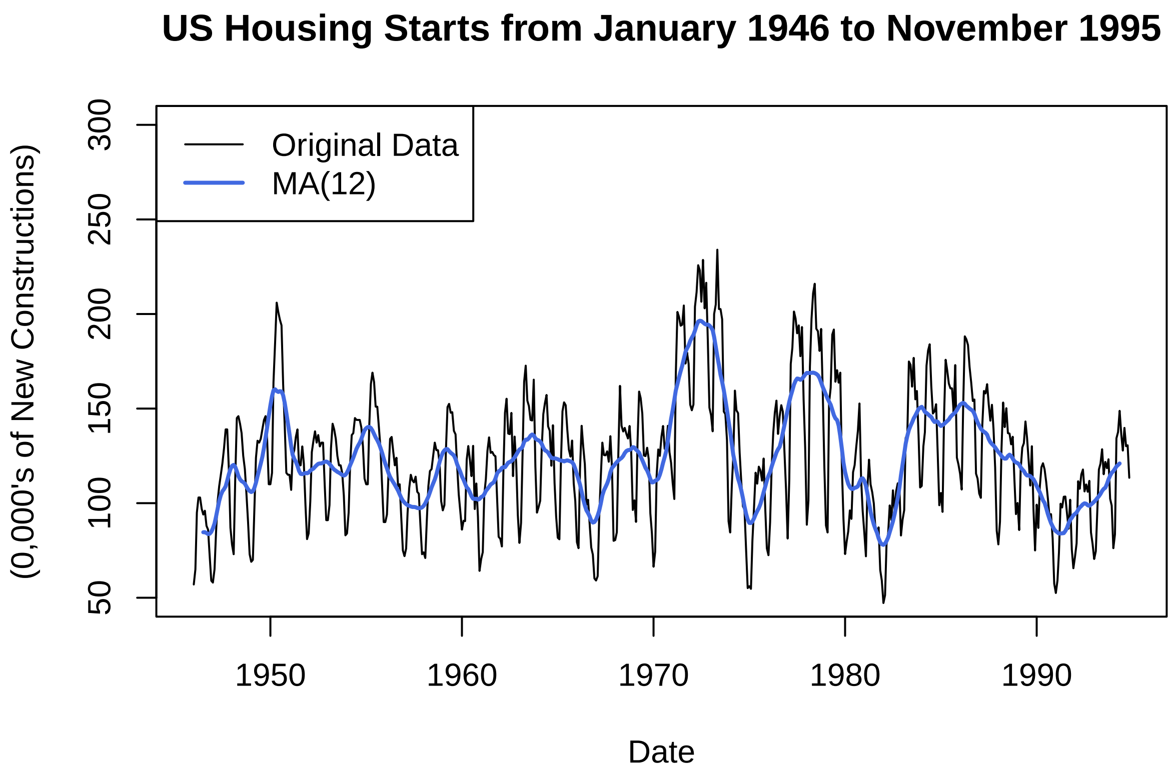
When we make our smoothing window too big, we end up with an over-smoothed series that is uninformative:

```
hstarts.ma85 <- ma(y$hstarts, order = 85)

plot(date,y$hstarts,
     main = "US Housing Starts from January 1946 to November 1995",
     xlab = "Date",
     ylab = "(0,000's of New Constructions)",
     ylim = c(50,300),
     type = 'l',
     lwd = 1,
     col = "black")
lines(date, hstarts.ma85,lwd = 2, col = "magenta")

legend(x = "topleft",
       legend = c("Original Data", "MA(85)" ),
       lty = c("solid", "solid"),
       lwd = c(1.0,2.0),
       col = c("black","magenta"))
```



So far we have chosen our moving average orders arbitrarily. If we think a bit more carefully, it should be clear to us that we should be choosing m to reflect the periodicity of the seasonality that is present in our data. Thus we should set $m = 12$ and include **centre = TRUE** in the argument of the **ma()** function to compute the averaged (centred) MA-12.

```
hstarts.ma12 <- ma(y$hstarts, order = 12, centre = TRUE)
plot(date,y$hstarts,
     main = "US Housing Starts from January 1946 to November 1995",
     xlab = "Date",
     ylab = "(0,000's of New Constructions)",
     ylim = c(50,300),
     type = 'l',
     lwd = 1,
     col = "black")
lines(date, hstarts.ma12,lwd = 2, col = "royalblue")

legend(x = "topleft",
       legend = c("Original Data", "MA(12)" ),
       lty = c("solid", "solid"),
       lwd = c(1.0,2.0),
       col = c("black","royalblue"))
```



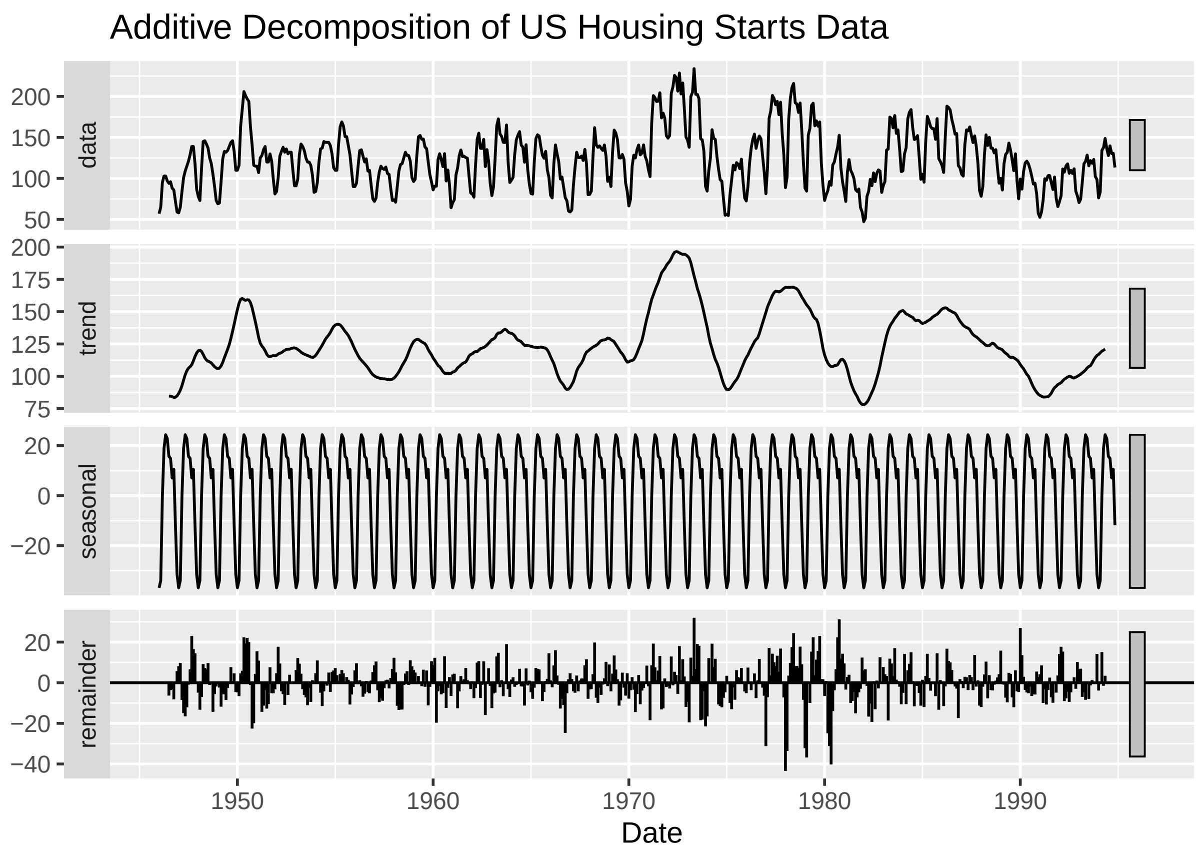
Now that we understand how to compute seasonal and smoothed trend-cycle components directly, let's now use the **decompose()** function to compute these components in a single line of code.

To use the **decompose()** function, we first need to format our data as a time series object using the **ts()** function:

```r
hstarts <- ts(y$hstarts, start = c(1946,1), end = c(1994,11), frequency = 12)
```

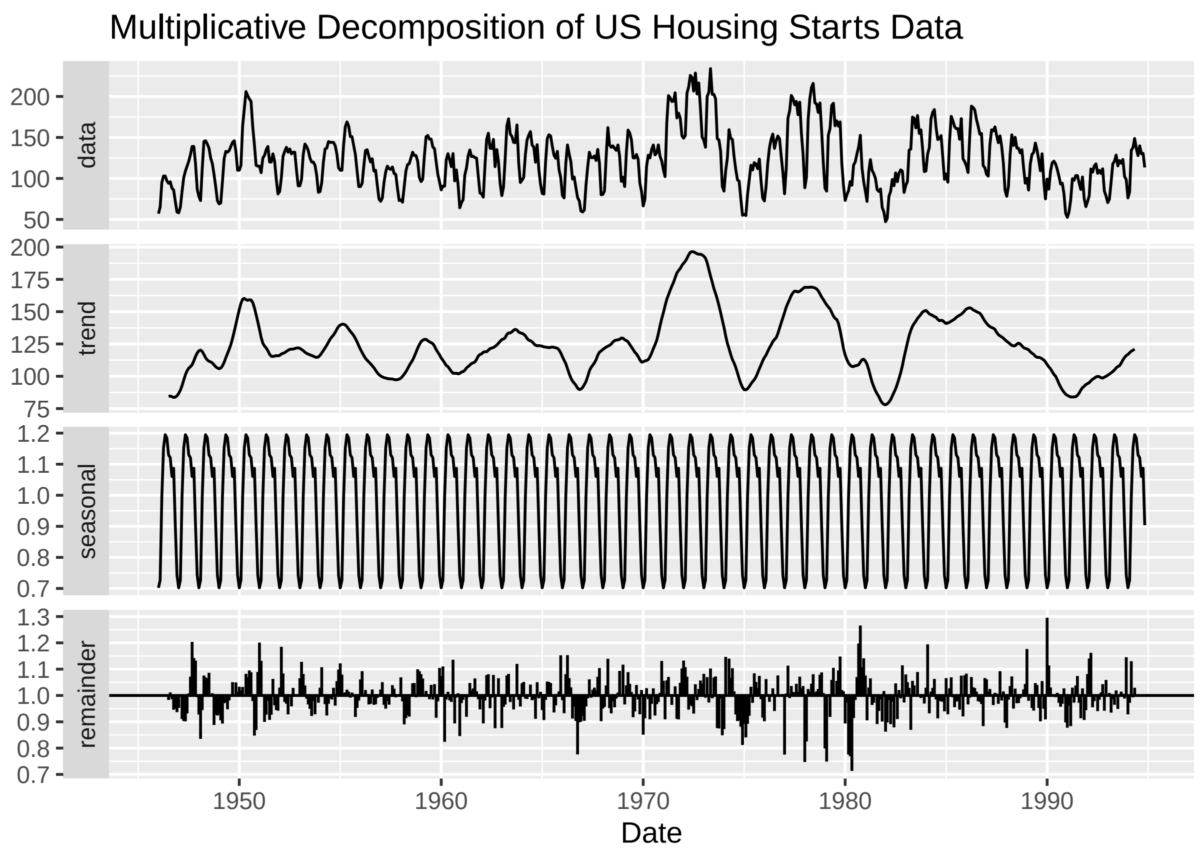
Then, we can compute and plot the decomposition using the following lines of code:

```r
hstarts.decomp.add <- decompose(hstarts, type = "additive")

autoplot(hstarts.decomp.add,
         main = "Additive Decomposition of US Housing Starts Data",
         xlab = "Date")
```



We can generate a multiplicative decomposition by simply changing the option in the decompose function:

```
hstarts.decomp.mul <- decompose(hstarts, type = "multiplicative")

autoplot(hstarts.decomp.mul,
         main = "Multiplicative Decomposition of US Housing Starts Data",
         xlab = "Date")
```



Multiplicative Decomposition of US Housing Starts Data
data
trend
seasonal
remainder
Date
1950
1960
1970
1980
1990