R Lab - Time Series

Tanvir

2023-12-11

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

R is very powerful when it comes to Time Series Analysis, however the methods that we are looking at this chapter are very basic, so probably you won't appreciate the use of R for these methods. Sorry guys.... ECO204 is just the starting, so we can't cover the advanced methods here and also I think some of the concepts of time series is beyond the scope of this course. However, if you are interested in Time Series Analysis, then you can take some courses in Econometrics and hopefully you will learn more. Nevertheless, here are some resources,

- Forecasting: Principles and Practice
- Time Series Analysis and Its Applications: With R Examples

Time Series Objects and Plots

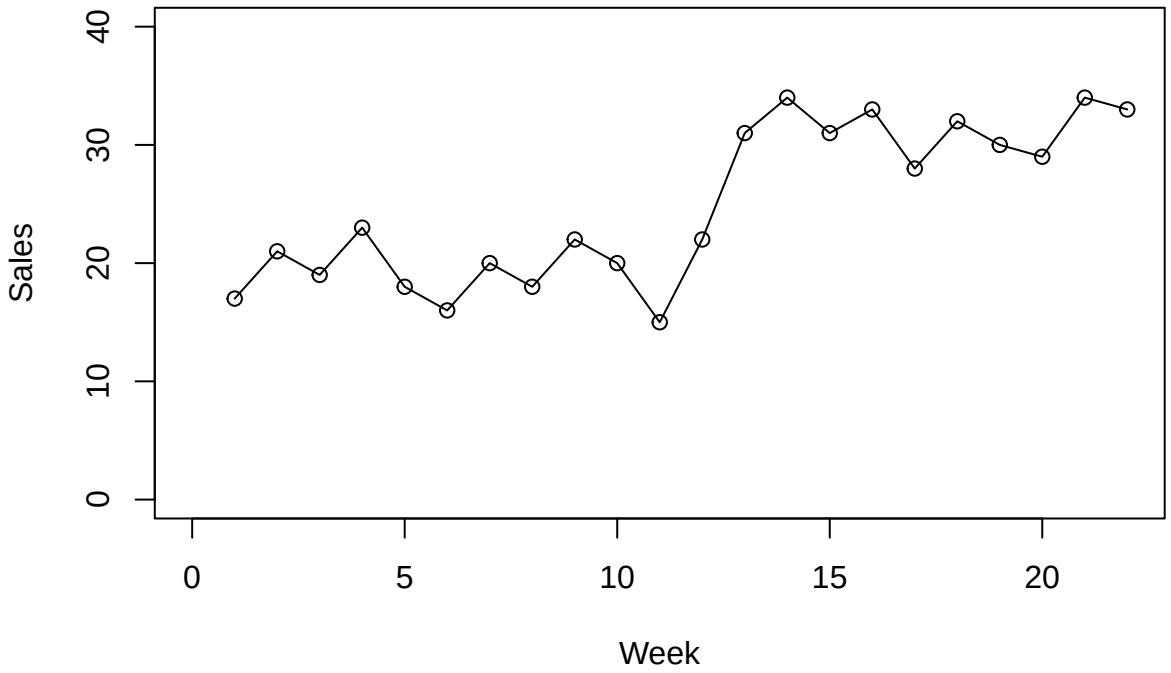
In R for time series analysis there is a special object. We can convert any vector to time series objects by calling the function `ts()`. The `ts()` function takes two arguments, the first one is the data and the second one is the frequency of the data. If we have consecutive time periods for a short amount of time, probably you don't need to do anything with the frequency, but complicated time structure like seasonality or cycles, possibly you should be careful.

Okay let's see an example, where we will start with a vector and then convert it to a time series object.

```
raw_data <- c(17, 21,  19,  23,  18, 16, 20, 18, 22, 20, 15, 22, 31, 34, 31, 33, 28, 32, 30, 29, 34, 33

# here frequency is 1 since we observe the data in every week, we don't need to
# set this but I did it just for clarity
gas_revised <- ts(raw_data, frequency = 1)

# "o" is used to show the circle in the plots
plot.ts(gas_revised , xlim = c(0, 22), ylim = c(0, 40), type = "o",
        xlab = "Week", ylab = "Sales")
```



One very important thing in the time series is the lagged values. We can get the lagged values of a time series object with the function `lag()`. Careful this function will only work for time series objects not for any arbitrary vector. Let's see how we can use this function.

```
# This will creae a new object called gas_lag which is the lagged value of the gas_revised object
lag(gas_revised, k = -1)
```

```
## Time Series:
## Start = 2
## End = 23
## Frequency = 1
##  [1] 17 21 19 23 18 16 20 18 22 20 15 22 31 34 31 33 28 32 30 29 34 33
```

Note, we used `k = -1`. The way you should understand this, we are in the period "0" currently. Since we want to get the data for the past period, so we used `k = - 1`. If we use `k = 1` then we will see value in the future, i.e., one period ahead. We can use the function `cbind()` to combine the original time series object and the lagged time series data.

```
cbind(gas_revised, lag(gas_revised, k = -1))
```

```
## Time Series:
## Start = 1
## End = 23
## Frequency = 1
##    gas_revised lag(gas_revised, k = -1)
##  1          17                       NA
##  2          21                       17
##  3          19                       21
##  4          23                       19
##  5          18                       23
##  6          16                       18
##  7          20                       16
##  8          18                       20
##  9          22                       18
## 10          20                       22
```

```
## 11          15                      20
## 12          22                      15
## 13          31                      22
## 14          34                      31
## 15          31                      34
## 16          33                      31
## 17          28                      33
## 18          32                      28
## 19          30                      32
## 20          29                      30
## 21          34                      29
## 22          33                      34
## 23          NA                      33
```

We can also different lagged values, for different k, you can try this. In R we can also look at directly the difference between the time series object and the lagged time series object. We can do that using the `diff()` function. Since I don't need this I won't use this function now, maybe later!

Horizontal Pattern

When the plot shows a horizontal pattern, then we can say that probably the time series object is stationary. For Horizontal pattern we learned two techniques to forecast,

- 1. Naive Forecast
- 2. Moving Average Forecast (Equal or Unequal Weights)

Let's see them now

1. Naive Forecast

In this method we simply take the last observation (lagged observation) as the forecast for the current period. So the equation is

$$F_t = y_{t-1}$$

This means that the forecast for the current period is the last observation. Let's see how we can do this in R. We will use the gasoline data from the Anderson book. First we will load the data, create a time series object and plot it

```
library(readxl)
raw_data <- read_excel("/home/tanvir/Documents/ownCloud/Git_Repos/EWU_repos/3_Fall_2023/eco_204/ewu-eco
```

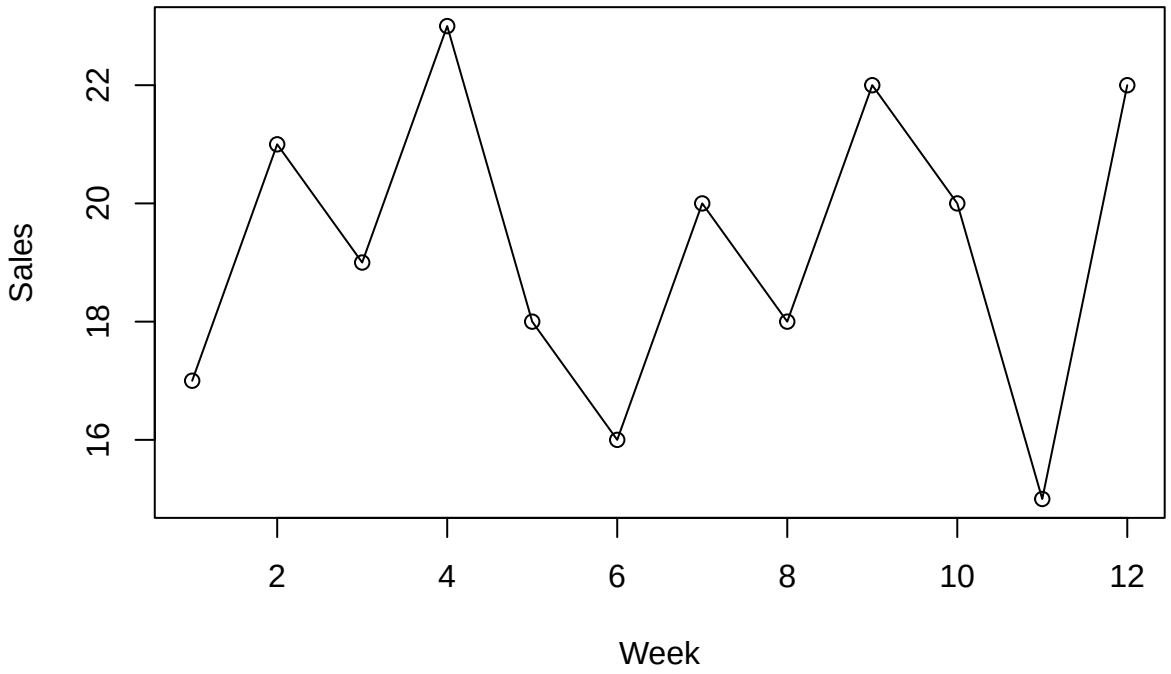
```
# take only the sales column and convert it to a time series object
gas_data <- ts(raw_data$Sales, frequency = 1)


# see the data
gas_data
```

```
## Time Series:
## Start = 1
## End = 12
## Frequency = 1
##  [1] 17 21 19 23 18 16 20 18 22 20 15 22
```

Let's plot the time series object

```
plot.ts(gas_data, type = "o", xlab = "Week", ylab = "Sales")
```



Now we will do the forecast using the naive method and then see `MFE`, `MAE` or other summary measures of the forecast. We will use the `accuracy()` function from the package `forecast` to do that. If you don't have the package installed, then you can install it using the command `install.packages("forecast")`

```
# get the naive forecast
gas_forecast_naive <- lag(gas_data, k = -1)

# load the library forecast
library(forecast)
```

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

```
# get the summary measures for accuracy of the forecast
accuracy(gas_forecast_naive, gas_data)
```

```
##                  ME     RMSE      MAE       MPE     MAPE       ACF1 Theil's U
## Test set 0.4545455 4.033947 3.727273 0.1082169 19.24431 -0.4553872         1
```

Note that the `accuracy()` function takes two arguments, the first one is the forecast and the second one is the actual data. The output of the `accuracy()` function is a table, which shows the summary statistics of the forecast.

- The first column `ME` is Mean Forecast Error (MFE), which is the average of the forecast errors.
- The second column `RMSE` is Root Mean Square Error (RMSE), which is the square root of the average of the squared forecast errors. If you square the RMSE, then you will get the MSE (Mean Squared Error).
- The third column `MAE` is Mean Absolute Error (MAE), which is the average of the absolute forecast errors.
- The fourth column `MPE` is Mean Percentage Error (MPE), which is the average of the percentage forecast

errors.

- The fifth column `MAPE` is Mean Absolute Percentage Error (MAPE), which is the average of the absolute percentage forecast errors.

You can ignore the last two columns. Finally we can plot the forecast and the actual data together to see how the forecast looks like. In this case we will use the `plot.ts()` function. To do this first we need to make a matrix of the forecast and the actual data using the `cbind()` function. Then we will use the `plot.ts()` function to plot the data. Let's see how we can do that.

```
# make a matrix of the forecast and the actual data
gas_forecast_naive_m <- cbind( gas_data, gas_forecast_naive)
gas_forecast_naive_m
```

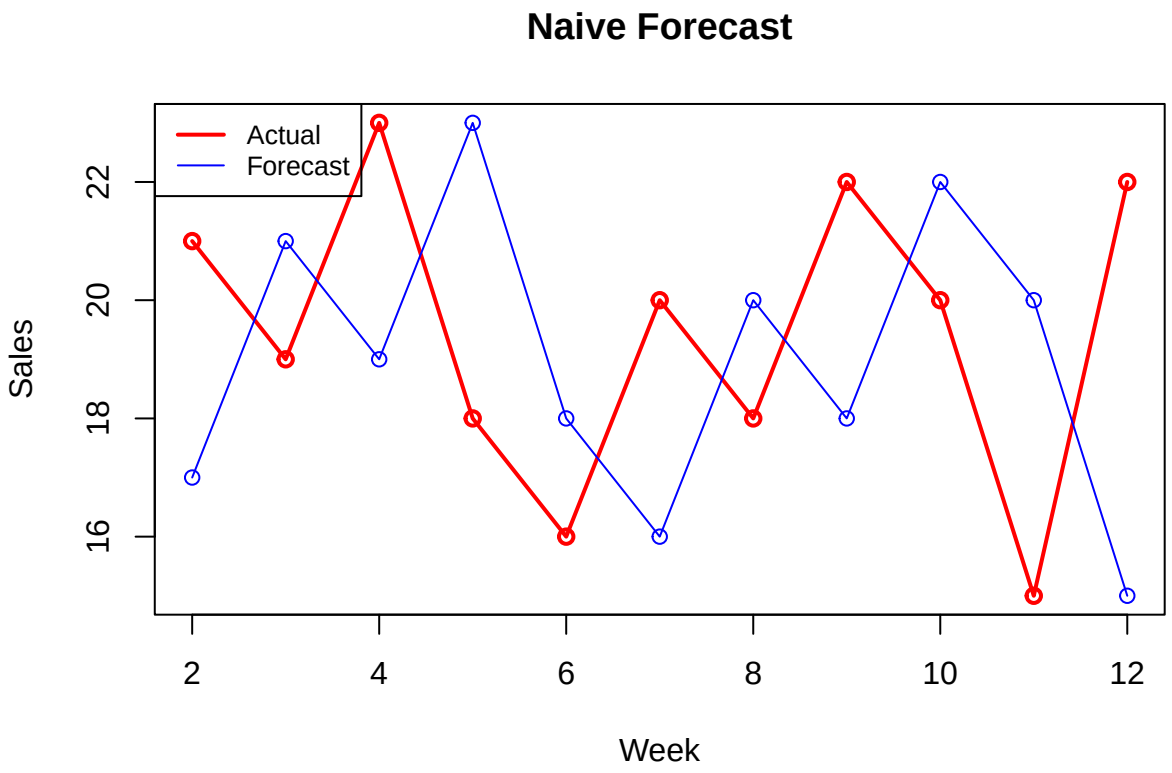
```
## Time Series:
## Start = 1
## End = 13
## Frequency = 1
##    gas_data gas_forecast_naive
##  1       17                 NA
##  2       21                 17
##  3       19                 21
##  4       23                 19
##  5       18                 23
##  6       16                 18
##  7       20                 16
##  8       18                 20
##  9       22                 18
## 10       20                 22
## 11       15                 20
## 12       22                 15
## 13       NA                 22
```

```
# remove the NAs from the matrix
gas_forecast_naive_m <- na.omit(gas_forecast_naive_m)
gas_forecast_naive_m
```

```
## Time Series:
## Start = 2
## End = 12
## Frequency = 1
##    gas_data gas_forecast_naive
##  2       21                 17
##  3       19                 21
##  4       23                 19
##  5       18                 23
##  6       16                 18
##  7       20                 16
##  8       18                 20
##  9       22                 18
## 10       20                 22
## 11       15                 20
## 12       22                 15
```

```
# plot the data
plot.ts(gas_forecast_naive_m,  plot.type = "single", xlab = "Week", ylab = "Sales", col = c("red", "blu
```

```
legend("topleft", legend=c("Actual", "Forecast"),col=c("red", "blue"), cex=0.8, lwd = c(2, 1))
```



2. Moving Average Forecasts (Equal / Unequal Weights)

In this method we take the average of the last few observations as the forecast for the current period. The number of observations that we take the average is called the order. For example, if we take the average of the last 3 observations, then the order is 3. If we write this then,

$$F_t = \frac{y_{t-1} + y_{t-2} + y_{t-3}}{3}$$

We will use the function `rollmean()` from the package `xts` to do that. Let's see how we can do that.

```
library(xts)
```

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

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

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

```
gas_ma <- rollmean(gas_data, k = 3,  align = "right", fill = NA)
gas_ma
```

```
## Time Series:
## Start = 1
## End = 12
## Frequency = 1
##  [1] NA NA 19 21 20 19 18 18 20 20 19 19
```

Now this is the data set of forecasts. The problem with the `rollmean()` function is that it does not give us 3 NAs at the beginning of the data set (which we should have), it gives two, but we can fix that using the `lag()` function. So using `lag` and then `cbind()` we can get the data set of forecasts.

```
cbind(gas_data, lag(gas_ma, k = -1))
```

```
## Time Series:
## Start = 1
## End = 13
## Frequency = 1
##    gas_data lag(gas_ma, k = -1)
##  1       17                   NA
##  2       21                   NA
##  3       19                   NA
##  4       23                   19
##  5       18                   21
##  6       16                   20
##  7       20                   19
##  8       18                   18
##  9       22                   18
## 10       20                   20
## 11       15                   20
## 12       22                   19
## 13       NA                   19
```

Finally we can use the `accuracy()` function to calculate the summary statistics of the forecast.

```
# fix the lag
gas_forecast_ma <- lag(gas_ma, k = -1)

# get the summary measures for accuracy of the forecast
accuracy(gas_forecast_ma, gas_data)
```

```
##           ME     RMSE      MAE       MPE     MAPE       ACF1 Theil's U
## Test set  0 3.197221 2.666667 -2.310057 14.35661 -0.2065217 0.7135559
```

Also here we can plot the forecast and the actual data together to see how the forecast looks like.

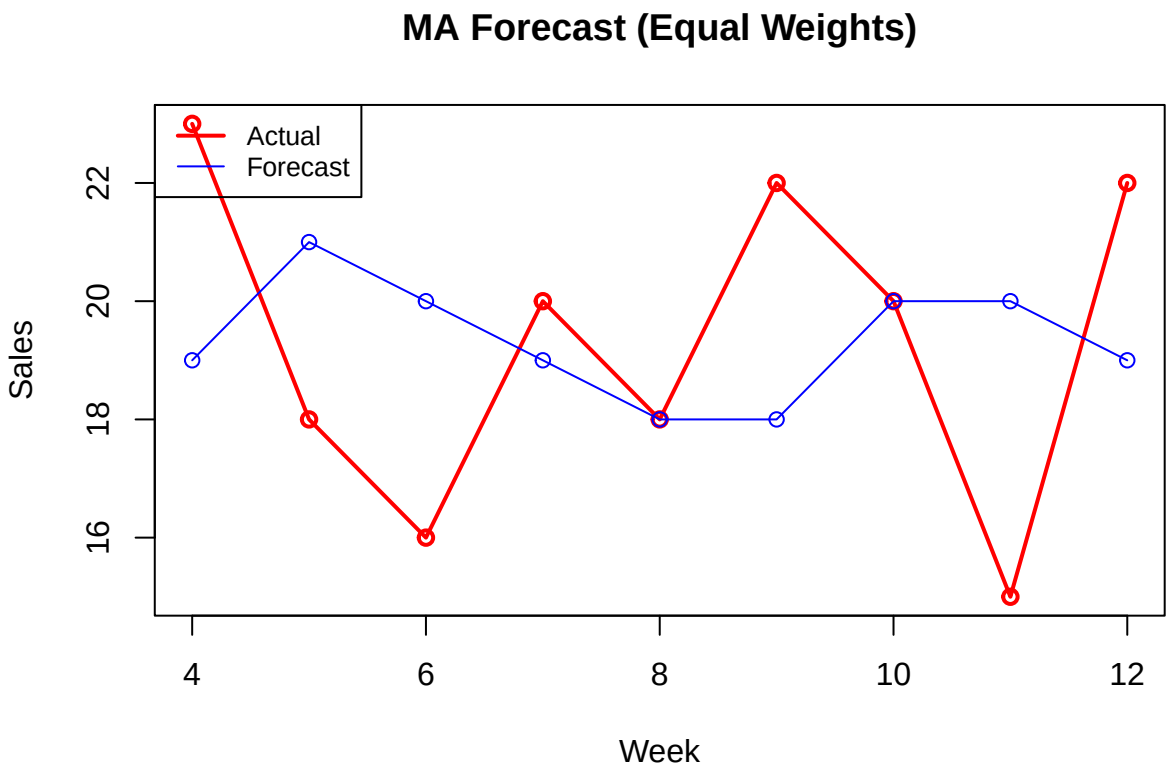
```
# make a matrix of the forecast and the actual data
gas_forecast_ma_m <- cbind( gas_data, gas_forecast_ma)

# remove the NAs from the matrix
gas_forecast_ma_m <- na.omit(gas_forecast_ma_m)
gas_forecast_ma_m
```

```
## Time Series:
## Start = 4
## End = 12
## Frequency = 1
##    gas_data gas_forecast_ma
##  4       23              19
##  5       18              21
##  6       16              20
##  7       20              19
##  8       18              18
##  9       22              18
## 10       20              20
```

```
## 11       15              20
## 12       22              19
```

```
# plot the data
plot.ts(gas_forecast_ma_m,  plot.type = "single", xlab = "Week", ylab = "Sales", col = c("red", "blue")
legend("topleft", legend=c("Actual", "Forecast"),col=c("red", "blue"), cex=0.8, lwd = c(2, 1))
```



Finally we will do the moving average with unequal weights. For this we can just give weights for the three observations that we are taking the average. Let's see how we can do that.

```
gas_ma_unequal <- rollapply(gas_data, width = 3, FUN = function(x){weighted.mean(x, w = c(1/6, 2/6, 3/6)

gas_ma_unequal
```

```
## Time Series:
## Start = 1
## End = 12
## Frequency = 1
##  [1]       NA       NA 19.33333 21.33333 19.83333 17.83333 18.33333 18.33333
##  [9] 20.33333 20.33333 17.83333 19.33333
```

Again we can use the `accuracy()` function to calculate the summary statistics of the forecast. Let's see how we can do that.

```
accuracy(lag(gas_ma_unequal, k = -1), gas_data)
```

```
##                    ME    RMSE      MAE       MPE     MAPE       ACF1 Theil's U
## Test set 0.05555556 3.3898 2.981481 -2.129945 15.99248 -0.2994209 0.7723224
```

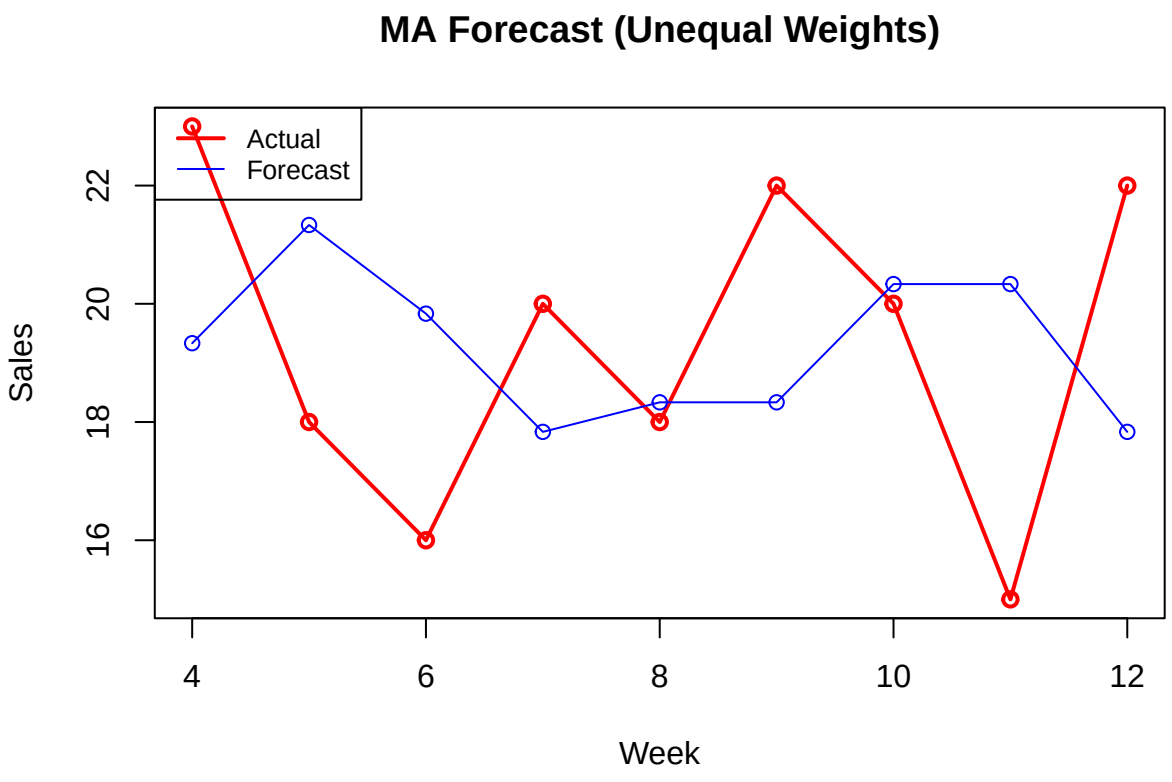
Again Finally we can plot the forecast and the actual data together to see how the forecast looks like.

```r
# make a matrix of the forecast and the actual data
gas_forecast_ma_unequal_m <- cbind( gas_data, lag(gas_ma_unequal, k = -1))

# remove the NAs from the matrix
gas_forecast_ma_unequal_m <- na.omit(gas_forecast_ma_unequal_m)
gas_forecast_ma_unequal_m
```

```
## Time Series:
## Start = 4
## End = 12
## Frequency = 1
##    gas_data lag(gas_ma_unequal, k = -1)
##  4       23                    19.33333
##  5       18                    21.33333
##  6       16                    19.83333
##  7       20                    17.83333
##  8       18                    18.33333
##  9       22                    18.33333
## 10       20                    20.33333
## 11       15                    20.33333
## 12       22                    17.83333
```

```r
# plot the data
plot.ts(gas_forecast_ma_unequal_m,  plot.type = "single", xlab = "Week", ylab = "Sales", col = c("red",
legend("topleft", legend=c("Actual", "Forecast"),col=c("red", "blue"), cex=0.8, lwd = c(2, 1))
```



3. Exponential Smoothing

We didn't cover exponential smoothing!

Trend Pattern

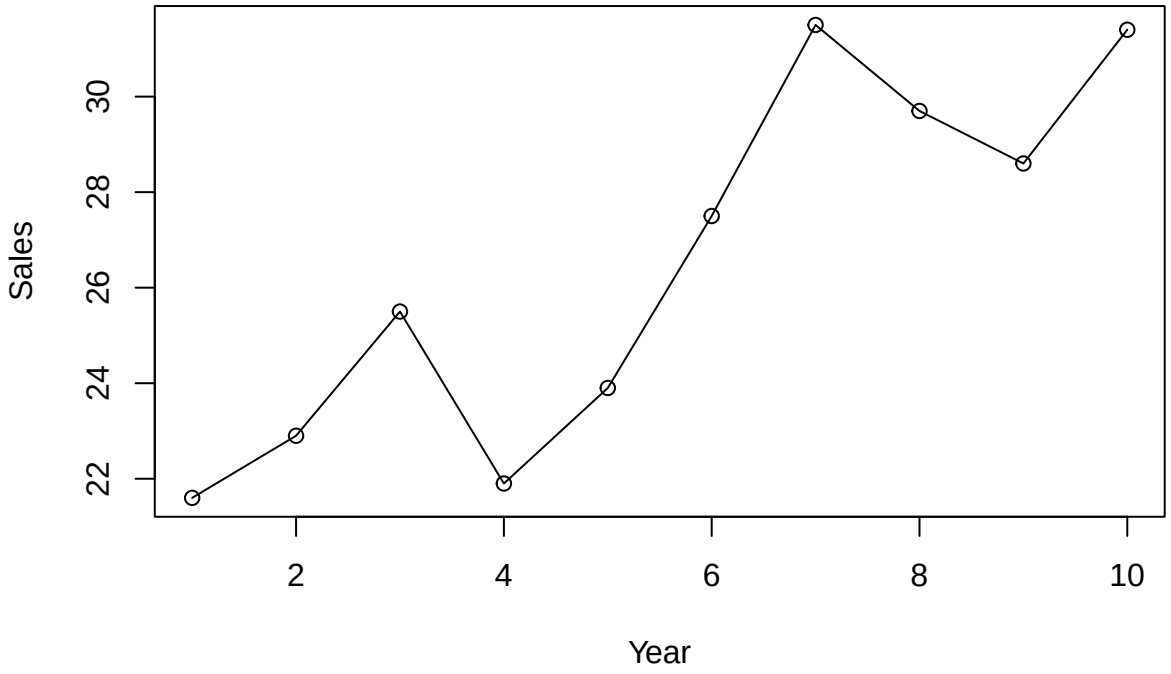
Now we will forecasting with trend pattern. For trend pattern we can use forecast using linear or non-linear trend pattern.

1. Linear Trend Pattern

Let's load the Bicycle data from the Anderson book. As usual, first we will load the data, create a time series object and plot it

```
raw_data <- read_excel("/home/tanvir/Documents/ownCloud/Git_Repos/EWU_repos/3_Fall_2023/eco_204/ewu-eco
bicycle_data <- ts(raw_data$Sales, frequency = 1)

plot.ts(bicycle_data, type = "o", xlab = "Year", ylab = "Sales")
```



The data shows a trend pattern. Now we will use the `tslm()` function to fit a linear trend model. Let's see how we can do that.

```
bicycle_lm <- tslm(bicycle_data ~ trend)
summary(bicycle_lm)
```

```
##
## Call:
## tslm(formula = bicycle_data ~ trend)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -2.900 -1.275  0.200  0.500  3.400
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  20.4000     1.3382   15.24  3.4e-07 ***
## trend         1.1000     0.2157    5.10  0.00093 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.959 on 8 degrees of freedom
## Multiple R-squared:  0.7648, Adjusted R-squared:  0.7354
## F-statistic: 26.01 on 1 and 8 DF,  p-value: 0.0009295
```

Now we will use the `predict()` function to forecast the data. Let's see how we can do that.

```
bicycle_data_forecast_table <- forecast(bicycle_lm)

bicycle_data_forecast_table
```

```
##    Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 11           32.5 29.18618 35.81382 27.02921 37.97079
## 12           33.6 30.12574 37.07426 27.86434 39.33566
## 13           34.7 31.04746 38.35254 28.67002 40.72998
## 14           35.8 31.95382 39.64618 29.45033 42.14967
## 15           36.9 32.84701 40.95299 30.20891 43.59109
## 16           38.0 33.72895 42.27105 30.94893 45.05107
## 17           39.1 34.60128 43.59872 31.67306 46.52694
## 18           40.2 35.46538 44.93462 32.38362 48.01638
## 19           41.3 36.32243 46.27757 33.08252 49.51748
## 20           42.4 37.17340 47.62660 33.77140 51.02860
```

```
# we only need the point forecast now
bicycle_data_forecast <- bicycle_data_forecast_table$fitted
```

Now we will calculate the summary statistics of the forecast. Same technique as before.

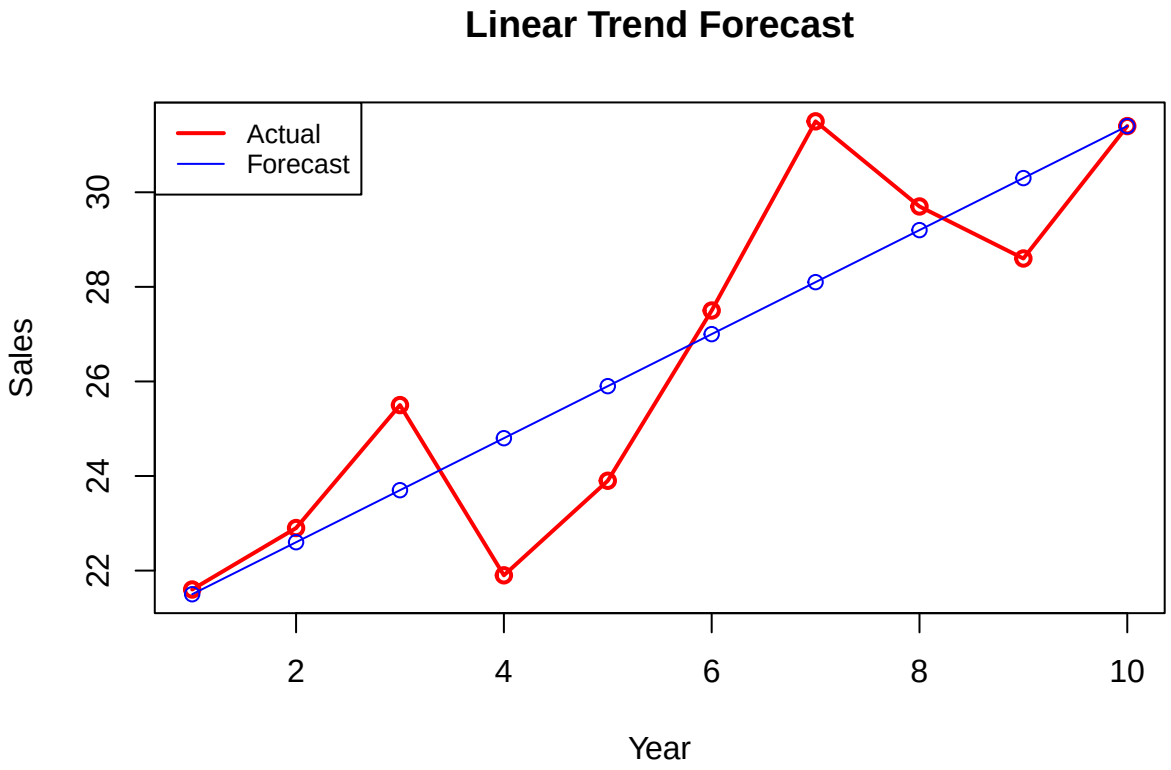
```
accuracy(bicycle_data_forecast, bicycle_data)
```

```
##                      ME     RMSE  MAE        MPE     MAPE       ACF1 Theil's U
## Test set -3.552844e-16 1.752142 1.32 -0.4427101 5.068143 0.08794788 0.6752526
```

Finally we will plot the actual data and the forecasted data together.

```
bicycle_data_forecast_m <- cbind(bicycle_data, bicycle_data_forecast)
bicycle_data_forecast_m <- na.omit(bicycle_data_forecast_m)


plot.ts(bicycle_data_forecast_m , type = "o", plot.type = "single", xlab = "Year", ylab = "Sales", col =
legend("topleft", legend=c("Actual", "Forecast"),col=c("red", "blue"), cex=0.8, lwd = c(2, 1))
```

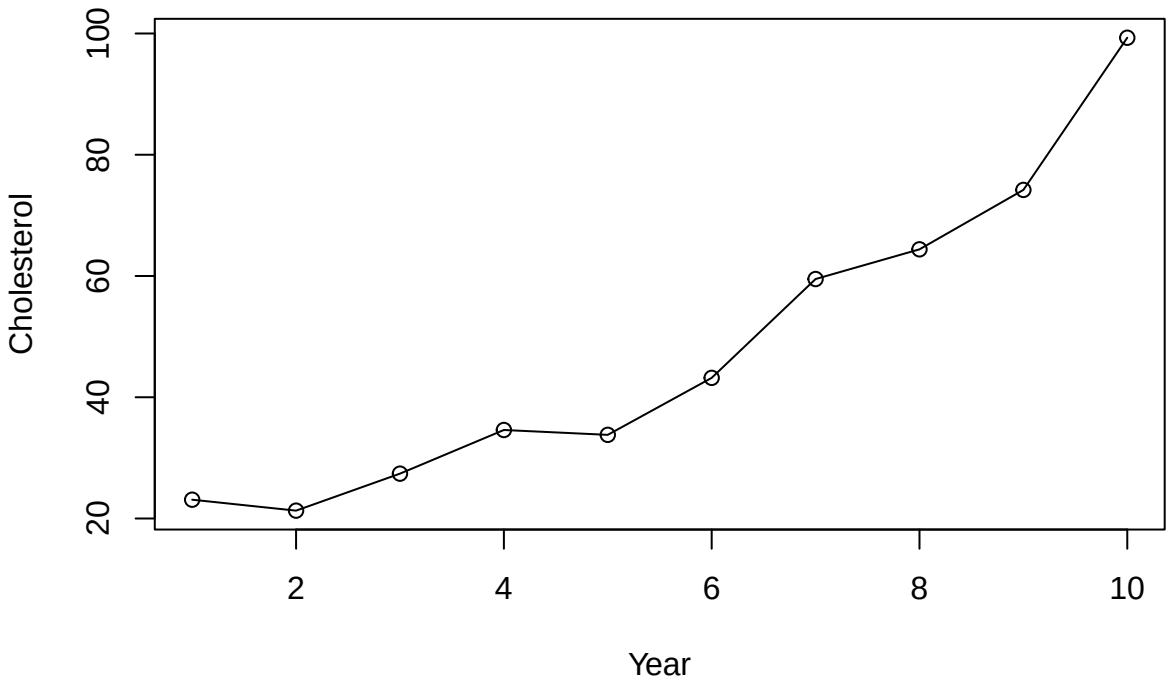


Finally we can fit a quadratic trend model. Let's see how we can do that. In this case we can use the Cholesterol data from the Anderson book.

```
raw_data <- read_excel("/home/tanvir/Documents/ownCloud/Git_Repos/EWU_repos/3_Fall_2023/eco_204/ewu-eco

cholesterol_data <- ts(raw_data$Revenue , frequency = 1)

plot.ts(cholesterol_data, type = "o", xlab = "Year", ylab = "Cholesterol")
```



```r
cholesterol_lm <- tslm(cholesterol_data ~ trend + I(trend^2))

summary(cholesterol_lm)
```

```
##
## Call:
## tslm(formula = cholesterol_data ~ trend + I(trend^2))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.6767 -2.2459 -0.7102  3.3248  4.9023
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  24.1817     4.6761   5.171  0.00129 **
## trend        -2.1060     1.9529  -1.078  0.31662
## I(trend^2)    0.9216     0.1730   5.326  0.00109 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.976 on 7 degrees of freedom
## Multiple R-squared:  0.9812, Adjusted R-squared:  0.9758
## F-statistic: 182.5 on 2 and 7 DF,  p-value: 9.137e-07
```

```r
cholesterol_data_forecast_table <- forecast(cholesterol_lm)

cholesterol_data_forecast <- cholesterol_data_forecast_table$fitted

accuracy(cholesterol_data_forecast, cholesterol_data)
```

```
##          ME     RMSE      MAE        MPE     MAPE       ACF1 Theil's U
## Test set  0 3.326378 2.872576 -0.4662172 6.287116 -0.3101741 0.3657351
```

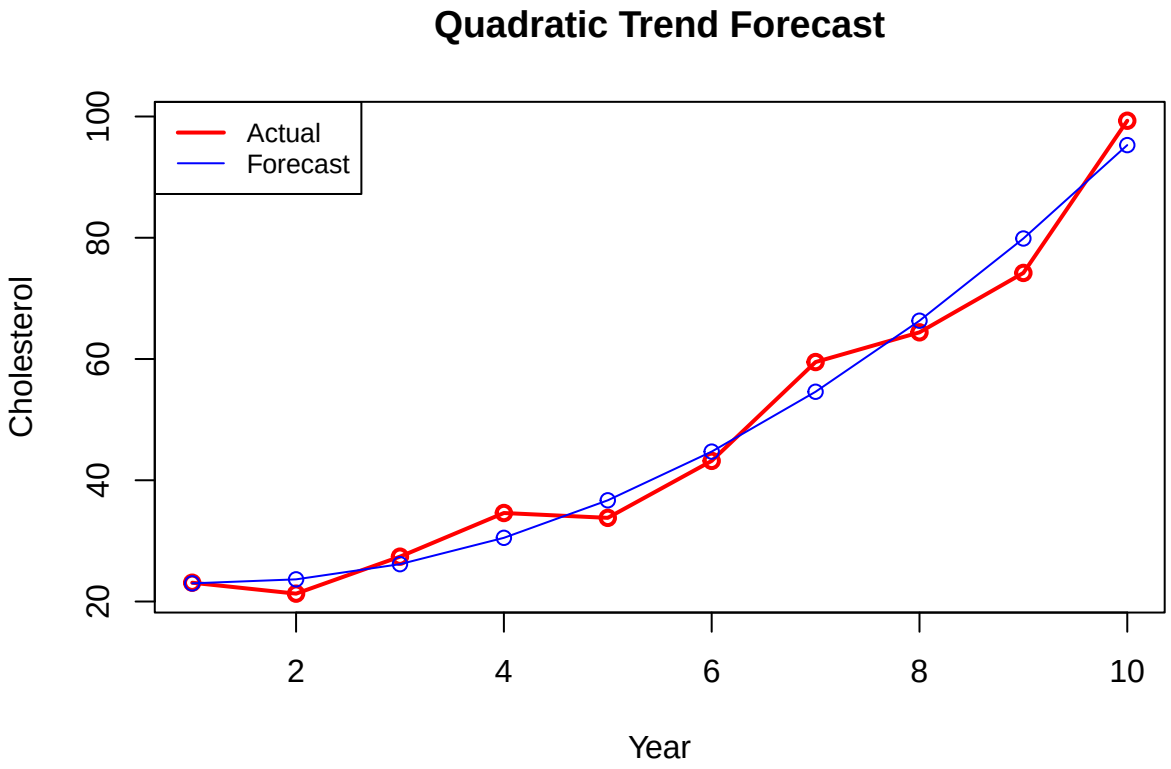
Finally we will plot the actual data and the forecasted data together.

```r
cholesterol_data_forecast_m <- cbind(cholesterol_data, cholesterol_data_forecast)
cholesterol_data_forecast_m <- na.omit(cholesterol_data_forecast_m)

cholesterol_data_forecast_m
```

```
## Time Series:
## Start = 1
## End = 10
## Frequency = 1
##    cholesterol_data cholesterol_data_forecast
##  1             23.1                  22.99727
##  2             21.3                  23.65606
##  3             27.4                  26.15803
##  4             34.6                  30.50318
##  5             33.8                  36.69152
##  6             43.2                  44.72303
##  7             59.5                  54.59773
##  8             64.4                  66.31561
##  9             74.2                  79.87667
## 10             99.3                  95.28091
```

```r
plot.ts(cholesterol_data_forecast_m , type = "o", plot.type = "single", xlab = "Year", ylab = "Cholester
legend("topleft", legend=c("Actual", "Forecast"),col=c("red", "blue"), cex=0.8, lwd = c(2, 1))
```



3. Seasonal Pattern

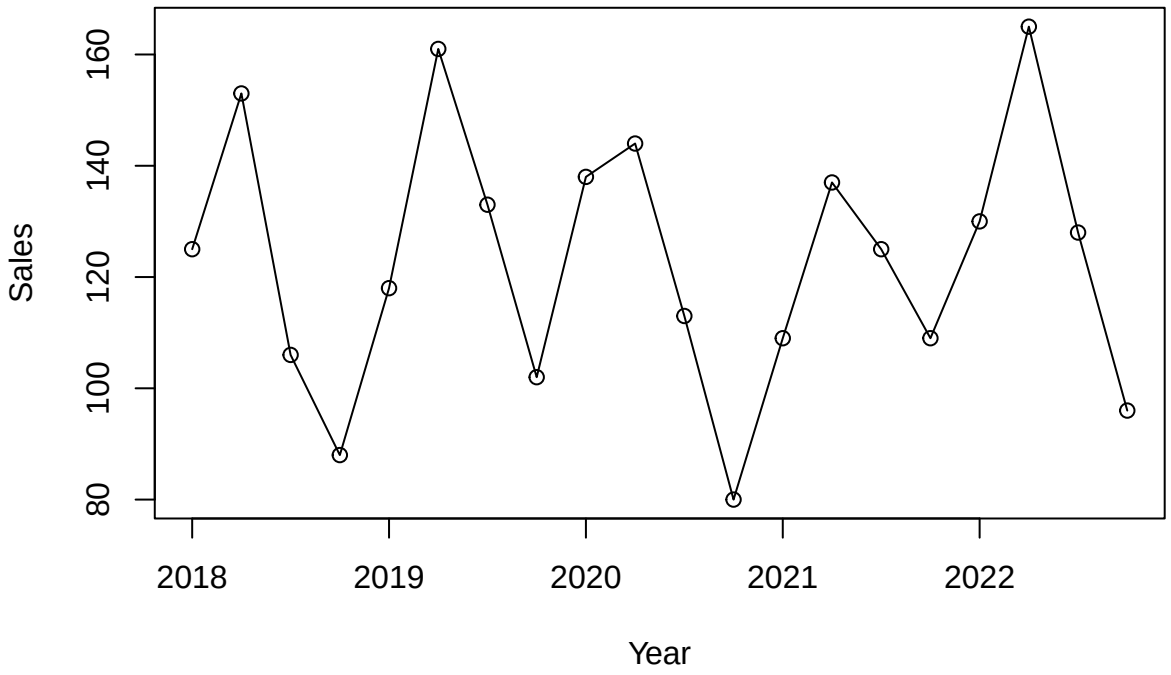
1. Seasonality Without Trend

Now we will start looking into Seasonal Pattern's data. Let's load a seasonal data from the Anderson book. As usual, first we will load the data, create a time series object and plot it

```r
raw_data <- read_excel("/home/tanvir/Documents/ownCloud/Git_Repos/EWU_repos/3_Fall_2023/eco_204/ewu-eco

# Note that we used frequency = 4 because the data is quarterly data.
umbrella_data <- ts(raw_data$Sales, frequency = 4, start = c(2018,1))

plot.ts(umbrella_data, type = "o", xlab = "Year", ylab = "Sales")
```



Now let's fit a linear model with seasonal dummies. The function `seasonaldummy()` from the package `forecast` can be used to create seasonal dummies.

```
seasonaldummy(umbrella_data)
```

```
##       Q1 Q2 Q3
##  [1,]  1  0  0
##  [2,]  0  1  0
##  [3,]  0  0  1
##  [4,]  0  0  0
##  [5,]  1  0  0
##  [6,]  0  1  0
##  [7,]  0  0  1
##  [8,]  0  0  0
##  [9,]  1  0  0
## [10,]  0  1  0
## [11,]  0  0  1
## [12,]  0  0  0
## [13,]  1  0  0
## [14,]  0  1  0
## [15,]  0  0  1
## [16,]  0  0  0
## [17,]  1  0  0
## [18,]  0  1  0
## [19,]  0  0  1
## [20,]  0  0  0
```

Finally we fit a linear model with seasonal dummies.

```
umbrella_lm <- tslm(umbrella_data ~ season)
summary(umbrella_lm)
```

```
##
## Call:
```

```
## tslm(formula = umbrella_data ~ season)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -15.0   -8.0    1.0    7.5   14.0
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  124.000      5.065  24.484 4.15e-14 ***
## season2       28.000      7.162   3.909 0.001249 **
## season3       -3.000      7.162  -0.419 0.680892
## season4      -29.000      7.162  -4.049 0.000931 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.32 on 16 degrees of freedom
## Multiple R-squared:  0.7989, Adjusted R-squared:  0.7611
## F-statistic: 21.18 on 3 and 16 DF,  p-value: 8.104e-06
```

Now we can calculate the forecast using the `forecast()` function. Let's see how we can do that.

```
Umrella_data_forecast_table <- forecast(umbrella_lm)


# we only need the point forecast now
Umrella_data_forecast <- Umrella_data_forecast_table$fitted

Umrella_data_forecast
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 2018  124  152  121   95
## 2019  124  152  121   95
## 2020  124  152  121   95
## 2021  124  152  121   95
## 2022  124  152  121   95
```

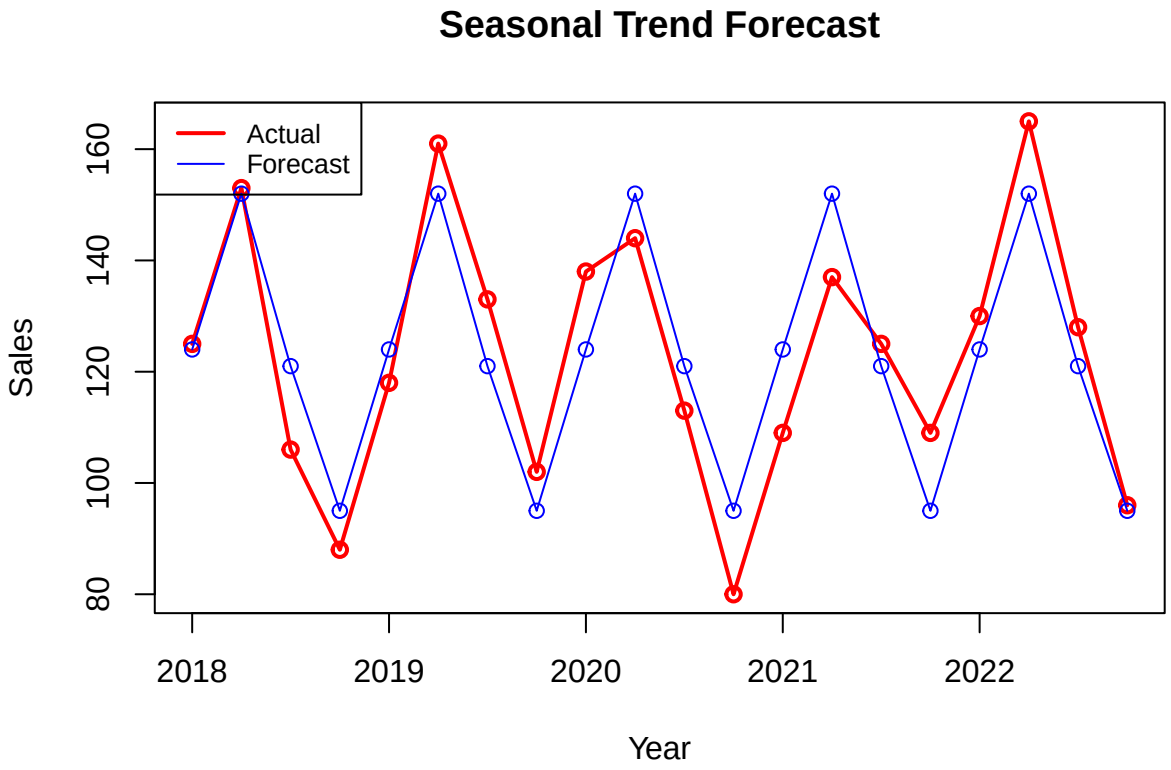
```
accuracy(Umrella_data_forecast, umbrella_data)
```

```
##                      ME     RMSE MAE        MPE     MAPE      ACF1 Theil's U
## Test set -2.842225e-15 10.12917 8.9 -0.7581649 7.570416 0.5589669 0.3616559
```

Finally we will plot the actual data and the forecasted data together.

```
# Nothing to omit here
Umrella_data_forecast_m <- cbind(umbrella_data, Umrella_data_forecast)

plot.ts(Umrella_data_forecast_m , type = "o", plot.type = "single", xlab = "Year", ylab = "Sales", col =
legend("topleft", legend=c("Actual", "Forecast"),col=c("red", "blue"), cex=0.8, lwd = c(2, 1))
```



1. Seasonality With Trend

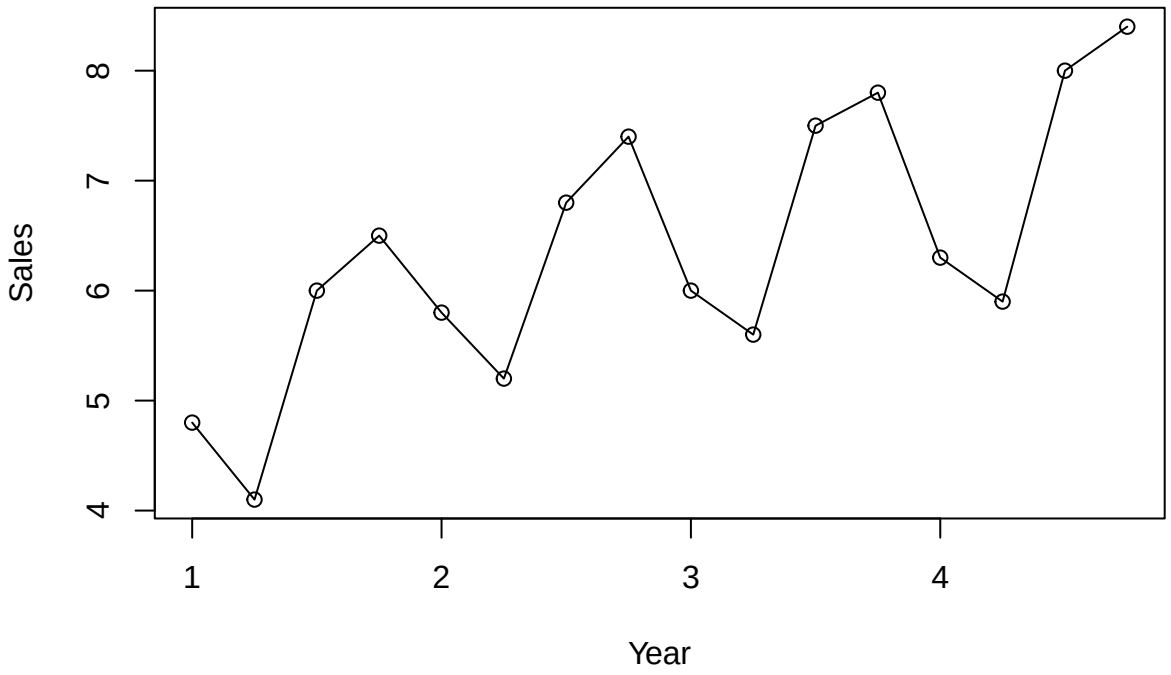
We can also do seasonality and trend together. Here we will use the Smartphones data from the Anderson book.

```
raw_data <- read_excel("/home/tanvir/Documents/ownCloud/Git_Repos/EWU_repos/3_Fall_2023/eco_204/ewu-eco

smartphone_data <- ts(raw_data$`Sales (1000s)`, frequency = 4, start = c(1,1))
smartphone_data
```

```
##   Qtr1 Qtr2 Qtr3 Qtr4
## 1  4.8  4.1  6.0  6.5
## 2  5.8  5.2  6.8  7.4
## 3  6.0  5.6  7.5  7.8
## 4  6.3  5.9  8.0  8.4
```

```
plot.ts(smartphone_data, type = "o", xlab = "Year", ylab = "Sales")
```



Now we will fit a linear model with seasonal dummies and trend. Let's see how we can do that.

```
smartphone_lm <- tslm(smartphone_data ~ trend + season)
summary(smartphone_lm)
```

```
##
## Call:
## tslm(formula = smartphone_data ~ trend + season)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.2988 -0.1569 -0.0075  0.1150  0.3663
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.70563    0.13756  34.207 1.60e-12 ***
## trend        0.14562    0.01211  12.023 1.14e-07 ***
## season2     -0.67063    0.15368  -4.364  0.00113 **
## season3      1.05875    0.15511   6.826 2.85e-05 ***
## season4      1.36312    0.15745   8.657 3.06e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2167 on 11 degrees of freedom
## Multiple R-squared:  0.9763, Adjusted R-squared:  0.9676
## F-statistic: 113.2 on 4 and 11 DF,  p-value: 7.376e-09
```

Now we will calculate the forecast using the `forecast()` function. Let's see how we can do that.

```
smartphone_data_forecast_table <- forecast(smartphone_lm)

# we only need the point forecast now
smartphone_data_forecast <- smartphone_data_forecast_table$fitted
```

```
smartphone_data_forecast
```

```
##      Qtr1    Qtr2    Qtr3    Qtr4
## 1 4.85125 4.32625 6.20125 6.65125
## 2 5.43375 4.90875 6.78375 7.23375
## 3 6.01625 5.49125 7.36625 7.81625
## 4 6.59875 6.07375 7.94875 8.39875
```

```
accuracy(smartphone_data_forecast, smartphone_data)
```

```
##                     ME      RMSE      MAE        MPE     MAPE      ACF1
## Test set 5.551115e-17 0.1796481 0.141875 -0.1032015 2.450953 0.3913853
##          Theil's U
## Test set 0.1458527
```

We can also plot the actual data and the forecasted data together.

```
# Nothing to omit here
smartphone_data_forecast_m <- cbind(smartphone_data, smartphone_data_forecast)

plot.ts(smartphone_data_forecast_m , type = "o", plot.type = "single", xlab = "Year", ylab = "Sales", c
legend("topleft", legend=c("Actual", "Forecast"),col=c("red", "blue"), cex=0.8, lwd = c(2, 1))
```