PES University, Bangalore

Established under Karnataka Act No. 16 of 2013

UE20CS312 - Data Analytics - Worksheet 3a - Basic Forecasting Techniques

Designed by Vishruth Veerendranath, Dept. of CSE - vishruth@pesu.pes.edu

Time Series Data and Basic Forecasting Techniques

Time Series data is any data that is collected at regular time intervals, with changing observations at every time interval. Processing time series data effectively can help gain meaningful insights into how a quantity changes with time.

Forecasting a quantity into the future is an essential task, that predicts future values at any particular time. Forecasts can be made using various techniques like Exponential Smoothing to much more complex techniques such as Recurrent Neural Networks. Let's try to process time-series data and forecast values using basic techniques!

Prerequisites

- Revise the following concepts
  - Components of Time-Series Data
  - Single, Double and Triple Exponential Smoothing
  - Regression (Refer to worksheets and slides from Unit 2)
  - Croston's Forecasting
  - Time-Series Accuracy Metrics

Task

Let's imagine it is 2012 and you are in the market to buy an Orange Ultrabook Laptop for college. But this laptop is rare to find and expensive. You would want to put your Data Analytics skills to use, and predict the best time to buy your laptop, such that you can get it for the best price! You would also like to predict when the Orange Ultrabook would be in stock and when it would have high demand.

An electronics store collected sales data for their store weekly, from *February 2010* to *October 2012*, a period of 143 months. You have gotten your hands on this, and will use it to predict how the prices will change in the future.

The data for the following tasks can be downloaded from this Github Repository.

Data Dictionary

```
Date - Date on which data was collected (end of the week)
Sales - Weekly sales of the store (in $)
Holiday_Flag - Boolean Flag. 0 for normal week and 1 for holiday season
Temperature - Average temperature during the week
Fuel_Price - Average price during the week (in $/gallon)
CPI - Consumer Price Index
Unemployment - Average percentage of Unemployment in the city
Laptop_Demand - Number of Orange Ultrabook laptops sold during the week
```

Data Ingestion and Preprocessing

- Read the file into a `data.frame` object

```
df <- read.csv('sales.csv')
head(df)
```

```
##   X       Date   Sales Holiday_Flag Temperature Fuel_Price      CPI
## 1 0 05-02-2010 2135144            0       43.76      2.598 126.4421
## 2 1 12-02-2010 2188307            1       28.84      2.573 126.4963
## 3 2 19-02-2010 2049860            0       36.45      2.540 126.5263
## 4 3 26-02-2010 1925729            0       41.36      2.590 126.5523
## 5 4 05-03-2010 1971057            0       43.49      2.654 126.5783
## 6 5 12-03-2010 1894324            0       49.63      2.704 126.6043
##   Unemployment Laptop_Demand
## 1        8.623             0
## 2        8.623             0
## 3        8.623             0
## 4        8.623             0
## 5        8.623             1
## 6        8.623             0
```

- Pick out the `Sales` column in the `data.frame`. Most of our time-series analysis will be done on this column.

```
sales <- df$Sales
head(sales)
```

```
## [1] 2135144 2188307 2049860 1925729 1971057 1894324
```

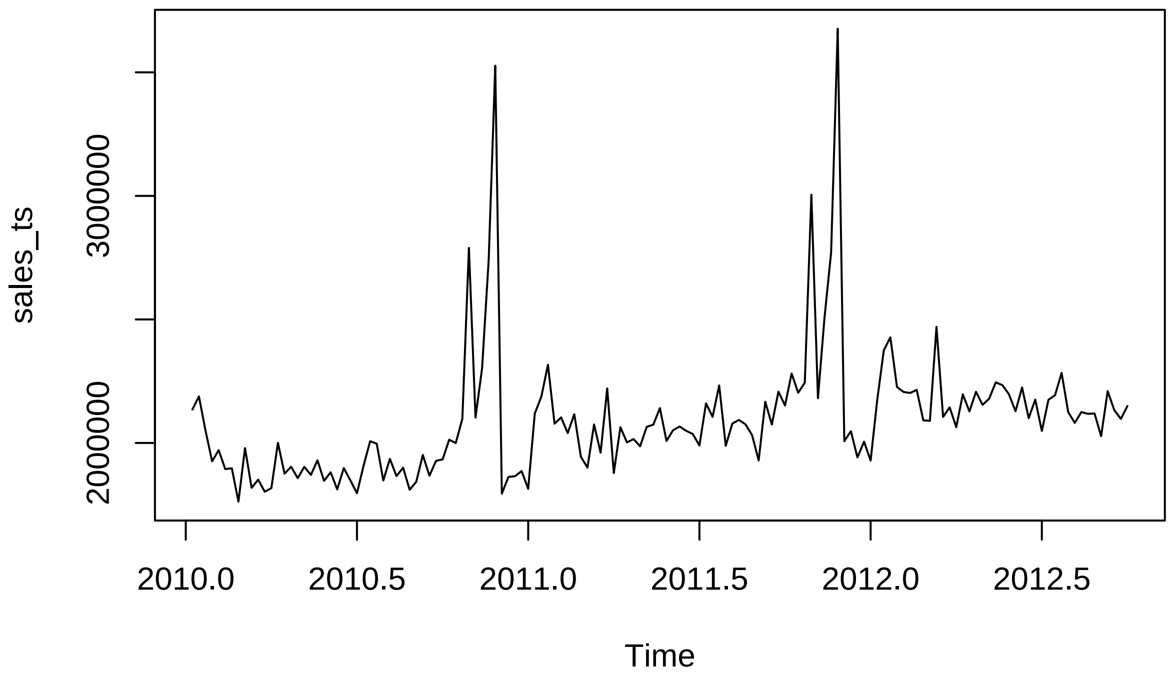
- The `ts` function is used to create the `ts` object in R. Frequency is 52 as it is weekly data. The start is specified like `start= c(y, m, d)` as we are dealing with weekly data. If it was monthly data we can omit the `d` and for yearly data we can omit the `m` as well.(`c` is the combine function in R)

```
sales_ts <- ts(sales, frequency = 52, start=c(2010, 2, 5))
sales_ts
```

```
## Time Series:
## Start = c(2010, 2)
## End = c(2012, 40)
## Frequency = 52
##   [1] 2135144 2188307 2049860 1925729 1971057 1894324 1897429 1762539 1979247
##  [10] 1818453 1851520 1802678 1817273 2000626 1875597 1903753 1857534 1903291
##  [19] 1870619 1929736 1846652 1881337 1812208 1898428 1848427 1796638 1907639
##  [28] 2007051 1997181 1848404 1935858 1865821 1899960 1810685 1842821 1951495
##  [37] 1867345 1927610 1933333 2013116 1999794 2097809 2789469 2102530 2302505
##  [46] 2740057 3526713 1794869 1862476 1865502 1886394 1814241 2119086 2187847
##  [55] 2316496 2078095 2103456 2039818 2116475 1944164 1900246 2074953 1960588
##  [64] 2220601 1878167 2063683 2002362 2015563 1986598 2065377 2073951 2141211
##  [73] 2008345 2051534 2066542 2049047 2036231 1989674 2160057 2105669 2232892
##  [82] 1988490 2078420 2093139 2075577 2031406 1929487 2166738 2074549 2207742
##  [91] 2151660 2281217 2203029 2243947 3004702 2180999 2508955 2771397 3676389
## [100] 2007106 2047766 1941677 2005098 1928721 2173374 2374661 2427640 2226662
## [109] 2206320 2202451 2214967 2091593 2089382 2470206 2105301 2144337 2064066
## [118] 2196968 2127661 2207215 2154138 2179361 2245257 2234191 2197300 2128363
## [127] 2224499 2100253 2175564 2048614 2174514 2193368 2283540 2125242 2081181
## [136] 2125105 2117855 2119439 2027620 2209835 2133026 2097267 2149594
```

- Visualize the Time-Series of `Sales` column

```
plot.ts(sales_ts)
```



Points

The problems in this worksheet are for a total of 10 points with each problem having a different weightage.

- *Problem 1*: 1 point
- *Problem 2*: 3 points
- *Problem 3*: 2 points
- *Problem 4*: 2 point
- *Problem 5*: 2 points

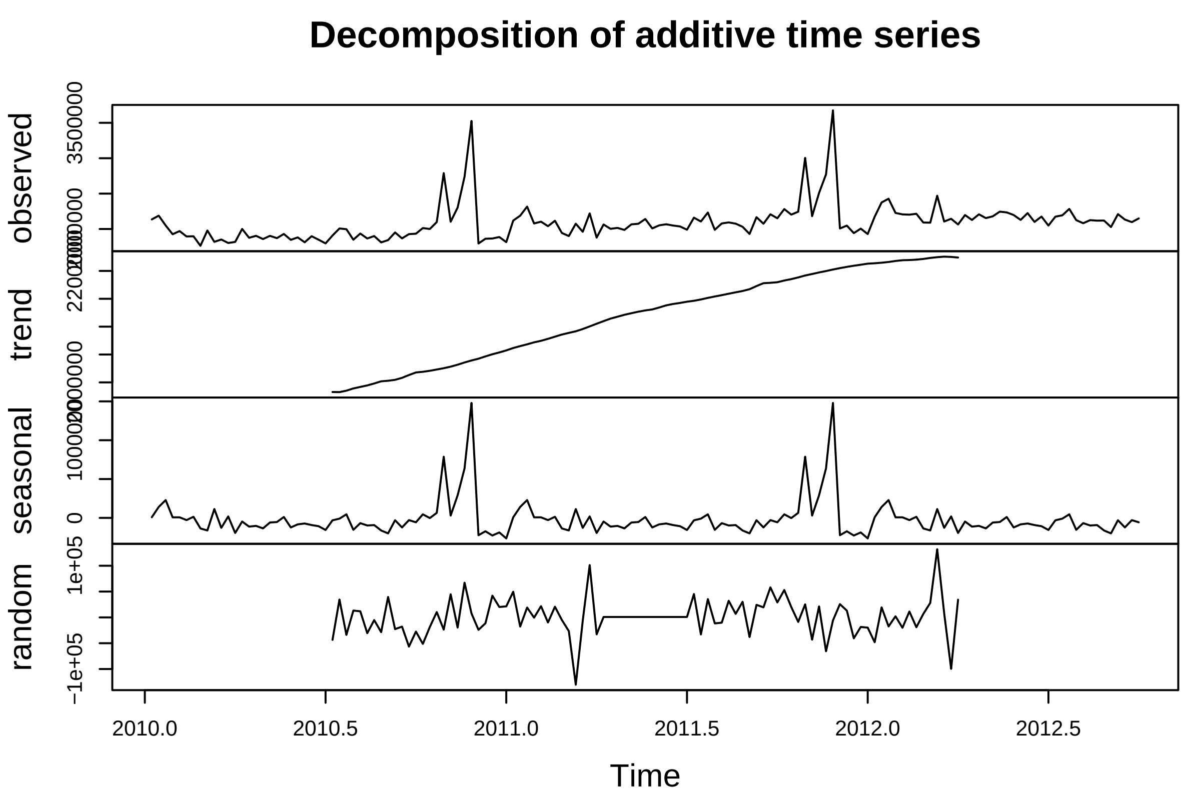
Problem 1 (1 Point)

Decompose the `Sales` column into trend, seasonal and random components. Plot these components as well (Hint: Look at the `decompose` function).

```
sales_decomp <- decompose(sales_ts)
sales_decomp
```

```
## $x
## Time Series:
## Start = c(2010, 2)
## End = c(2012, 40)
## Frequency = 52
##   [1] 2135144 2188307 2049860 1925729 1971057 1894324 1897429 1762539 1979247
##  [10] 1818453 1851520 1802678 1817273 2000626 1875597 1903753 1857534 1903291
##  [19] 1870619 1929736 1846652 1881337 1812208 1898428 1848427 1796638 1907639
##  [28] 2007051 1997181 1848404 1935858 1865821 1899960 1810685 1842821 1951495
##  [37] 1867345 1927610 1933333 2013116 1999794 2097809 2789469 2102530 2302505
##  [46] 2740057 3526713 1794869 1862476 1865502 1886394 1814241 2119086 2187847
##  [55] 2316496 2078095 2103456 2039818 2116475 1944164 1900246 2074953 1960588
##  [64] 2220601 1878167 2063683 2002362 2015563 1986598 2065377 2073951 2141211
##  [73] 2008345 2051534 2066542 2049047 2036231 1989674 2160057 2105669 2232892
##  [82] 1988490 2078420 2093139 2075577 2031406 1929487 2166738 2074549 2207742
##  [91] 2151660 2281217 2203029 2243947 3004702 2180999 2508955 2771397 3676389
## [100] 2007106 2047766 1941677 2005098 1928721 2173374 2374661 2427640 2226662
## [109] 2206320 2202451 2214967 2091593 2089382 2470206 2105301 2144337 2064066
## [118] 2196968 2127661 2207215 2154138 2179361 2245257 2234191 2197300 2128363
## [127] 2224499 2100253 2175564 2048614 2174514 2193368 2283540 2125242 2081181
## [136] 2125105 2117855 2119439 2027620 2209835 2133026 2097267 2149594
##
## $seasonal
## Time Series:
## Start = c(2010, 2)
## End = c(2012, 40)
## Frequency = 52
##   [1]    7743.257  140588.437  229129.855    6732.743    7212.515  -28485.929
##   [7]   13806.741 -136568.204 -162005.366  113629.830 -128524.725   18906.555
##  [13] -194301.096  -47011.003 -112863.472 -103021.707 -135461.949  -59506.806
##  [19]  -53672.187   11300.607 -123305.628  -83596.948  -72410.992  -92419.978
##  [25] -107109.365 -155908.834  -31501.633  -10069.600   45895.071 -153957.503
##  [31]  -67843.488  -98114.971  -92834.414 -162665.952 -199669.152  -30369.650
##  [37] -122892.464  -29026.035  -57034.928   45504.872   -2405.866   64389.720
##  [43]  787712.682   29638.265  290475.194  637207.431 1479881.931 -223555.384
##  [49] -172591.306 -227075.422 -187462.192 -264511.559    7743.257  140588.437
##  [55]  229129.855    6732.743    7212.515  -28485.929   13806.741 -136568.204
##  [61] -162005.366  113629.830 -128524.725   18906.555 -194301.096  -47011.003
##  [67] -112863.472 -103021.707 -135461.949  -59506.806  -53672.187   11300.607
##  [73] -123305.628  -83596.948  -72410.992  -92419.978 -107109.365 -155908.834
##  [79]  -31501.633  -10069.600   45895.071 -153957.503  -67843.488  -98114.971
##  [85]  -92834.414 -162665.952 -199669.152  -30369.650 -122892.464  -29026.035
##  [91]  -57034.928   45504.872   -2405.866   64389.720  787712.682   29638.265
##  [97]  290475.194  637207.431 1479881.931 -223555.384 -172591.306 -227075.422
## [103] -187462.192 -264511.559    7743.257  140588.437  229129.855    6732.743
## [109]    7212.515  -28485.929   13806.741 -136568.204 -162005.366  113629.830
## [115] -128524.725   18906.555 -194301.096  -47011.003 -112863.472 -103021.707
## [121] -135461.949  -59506.806  -53672.187   11300.607 -123305.628  -83596.948
## [127]  -72410.992  -92419.978 -107109.365 -155908.834  -31501.633  -10069.600
## [133]   45895.071 -153957.503  -67843.488  -98114.971  -92834.414 -162665.952
## [139] -199669.152  -30369.650 -122892.464  -29026.035  -57034.928
##
## $trend
## Time Series:
## Start = c(2010, 2)
## End = c(2012, 40)
## Frequency = 52
##   [1]      NA      NA      NA      NA      NA      NA      NA      NA      NA
##  [10]      NA      NA      NA      NA      NA      NA      NA      NA      NA
##  [19]      NA      NA      NA      NA      NA      NA      NA      NA 1982713
##  [28] 1982554 1985113 1989142 1991880 1994552 1998057 2001910 2002897 2004603
##  [37] 2008119 2013186 2017790 2018982 2020807 2023101 2025417 2028216 2031730
##  [46] 2035719 2039307 2042498 2046580 2050474 2053728 2057390 2061673 2065048
##  [55] 2068263 2071877 2074594 2078151 2082025 2085836 2088792 2091695 2095757
##  [64] 2100443 2105236 2109913 2114445 2117804 2121279 2124103 2126843 2129129
##  [73] 2130870 2134350 2138172 2140686 2142560 2144802 2146425 2148743 2151608
##  [82] 2154105 2156523 2159076 2161586 2163951 2167187 2172806 2177998 2178657
##  [91] 2179711 2182780 2185266 2188314 2191768 2194475 2197218 2199759 2202470
## [100] 2205025 2207283 2209294 2211126 2213033 2213738 2214721 2216051 2217853
## [109] 2219194 2219528 2220242 2221495 2223285 2224643 2225620 2225120 2224038
## [118]      NA      NA      NA      NA      NA      NA      NA      NA      NA
## [127]      NA      NA      NA      NA      NA      NA      NA      NA      NA
## [136]      NA      NA      NA      NA      NA      NA      NA      NA
##
## $random
## Time Series:
## Start = c(2010, 2)
## End = c(2012, 40)
## Frequency = 52
##   [1]          NA          NA          NA          NA          NA
##   [6]          NA          NA          NA          NA          NA
##  [11]          NA          NA          NA          NA          NA
##  [16]          NA          NA          NA          NA          NA
##  [21]          NA          NA          NA          NA          NA
##  [26]          NA -43572.3913  34566.5723 -33827.1360  13219.4112
##  [31]  11820.9387 -30616.3929  -5263.3446 -28559.3436  39593.4238
##  [36] -22738.9795 -17881.0100 -56549.6898 -27421.8665 -51370.7113
##  [41] -18606.7116  10318.8423 -23660.1762  44675.4337 -19700.4475
##  [46]  67131.0721   7524.6758 -24073.9025 -11512.4692  42104.0584
##  [51]  20128.2333  21362.5768  49669.7269 -17789.5569  19102.5927
##  [56]   -514.6002  21648.9085  -9846.5347  20643.3808  -5103.6898
##  [61] -26540.0261 -130371.1968  -6644.3285 101251.4701 -32767.0705
##  [66]    780.9629    780.9629    780.9629    780.9629    780.9629
##  [71]    780.9629    780.9629    780.9629    780.9629    780.9629
##  [76]    780.9629    780.9629    780.9629  45134.3171 -33004.6465
##  [81]  35389.0618 -11657.4854 -10259.0129  32178.3187   6825.2704
##  [86]  30121.2694 -38031.4980  24300.9052  19442.9358  58111.6156
##  [91]  28983.7923  52932.6371  20168.6374  -8756.9165  25222.1020
##  [96] -43113.5079  21262.3733 -65569.1463  -5962.7500  25635.8283
## [101]  13074.3950 -40542.1326 -18566.3075 -19800.6510 -48107.8011
## [106]  19351.4827 -17540.6669   2076.5260 -20086.9827  11408.4604
## [111] -19081.4550   6665.6156  28101.9519 131933.1226   8206.2543
## [116] -99689.5443  34328.9963          NA          NA          NA
## [121]          NA          NA          NA          NA          NA
## [126]          NA          NA          NA          NA          NA
## [131]          NA          NA          NA          NA          NA
## [136]          NA          NA          NA          NA          NA
## [141]          NA          NA          NA
##
## $figure
##  [1]    7743.257  140588.437  229129.855    6732.743    7212.515  -28485.929
##  [7]   13806.741 -136568.204 -162005.366  113629.830 -128524.725   18906.555
## [13] -194301.096  -47011.003 -112863.472 -103021.707 -135461.949  -59506.806
## [19]  -53672.187   11300.607 -123305.628  -83596.948  -72410.992  -92419.978
## [25] -107109.365 -155908.834  -31501.633  -10069.600   45895.071 -153957.503
## [31]  -67843.488  -98114.971  -92834.414 -162665.952 -199669.152  -30369.650
## [37] -122892.464  -29026.035  -57034.928   45504.872   -2405.866   64389.720
## [43]  787712.682   29638.265  290475.194  637207.431 1479881.931 -223555.384
## [49] -172591.306 -227075.422 -187462.192 -264511.559
##
## $type
## [1] "additive"
##
## attr(,"class")
## [1] "decomposed.ts"
```

```r
plot(sales_decomp)
```



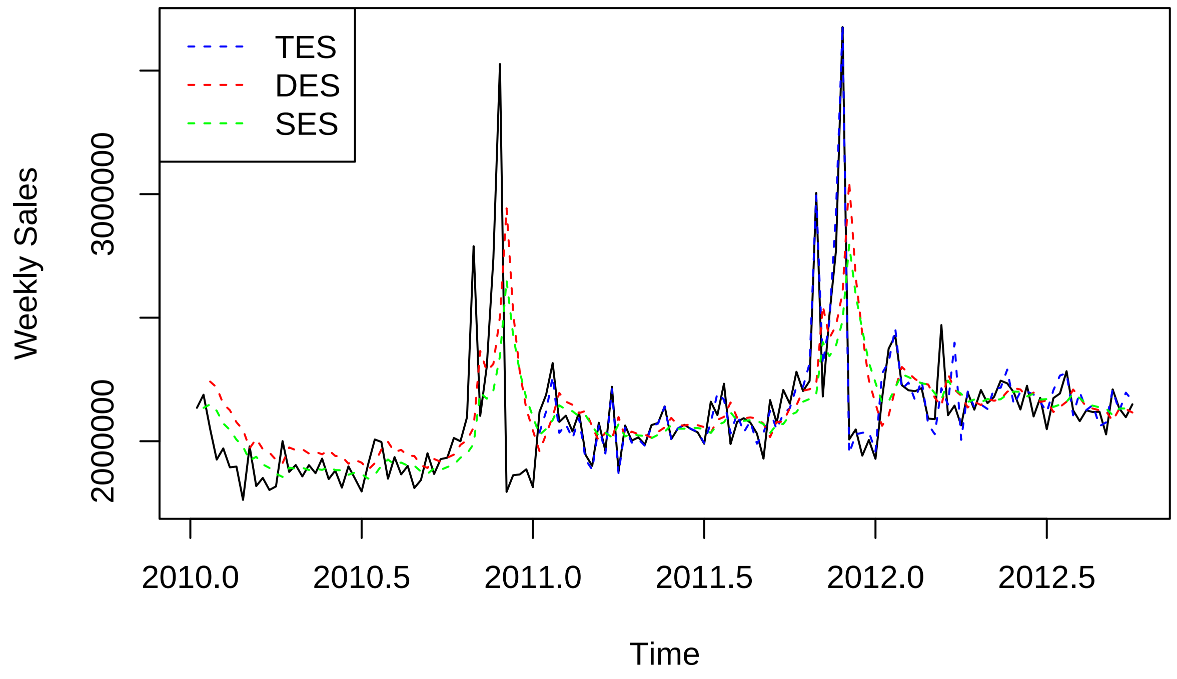
Problem 2 (3 Points)

- Perform forecasts using Single, Double and Triple Exponential Smoothing.
- Plot the forecasts of all three forecasts (different colours) against the true values. (Hint: use `lines`)
- **Use only one function needed for all 3 forecasts**, only changing parameters to get each of the 3 models (Hint: Think about alternate names)

```r
tes_fc <- HoltWinters(sales_ts)
des_fc <- HoltWinters(sales_ts, gamma=FALSE)
ses_fc <- HoltWinters(sales_ts, gamma=FALSE, beta=FALSE)

plot(sales_ts, ylab="Weekly Sales")
lines(tes_fc$fitted[,1], lty="dashed", col="blue")
lines(des_fc$fitted[,1], lty="dashed", col="red")
lines(ses_fc$fitted[,1], lty="dashed", col="green")

legend(x="topleft", legend=c('TES', 'DES', 'SES'), col=c('blue', 'red', 'green'), lty="dashed")
```



Problem 3 (2 Points)

- Forecast `Sales` values by Regression using all other columns. Print summary of regression model.
- Plot the predicted values against original as well. (Hint: Regression model predictions will not be a Time Series, so need to get both to common index/x-axis)
- (Hint: Will not work unless one column is dropped/transformed before including it in the regression. Use the `lm` function to get linear model)

Note: This is Multiple Linear Regression, that is, using all the columns for regression

- Option 1: Convert Date from string to Date

```
df$Date = as.Date(df$Date)

reg = lm(Sales ~ ., data = df)
summary(reg)
```

```
##
## Call:
## lm(formula = Sales ~ ., data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -454716 -105957  -23533   49902 1381881
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -3.007e+06  8.528e+06  -0.353  0.72492
## X               8.075e+03  4.168e+03   1.937  0.05482 .
## Date           -3.938e+00  6.415e+00  -0.614  0.54034
## Holiday_Flag    8.282e+04  8.084e+04   1.024  0.30745
## Temperature    -4.739e+03  1.441e+03  -3.288  0.00129 **
## Fuel_Price     -1.783e+04  1.126e+05  -0.158  0.87444
## CPI             8.405e+03  5.830e+04   0.144  0.88558
## Unemployment    1.644e+05  1.131e+05   1.454  0.14837
## Laptop_Demand   2.260e+03  4.479e+03   0.505  0.61471
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 237300 on 134 degrees of freedom
## Multiple R-squared:  0.2503, Adjusted R-squared:  0.2055
## F-statistic: 5.591 on 8 and 134 DF,  p-value: 4.082e-06
```

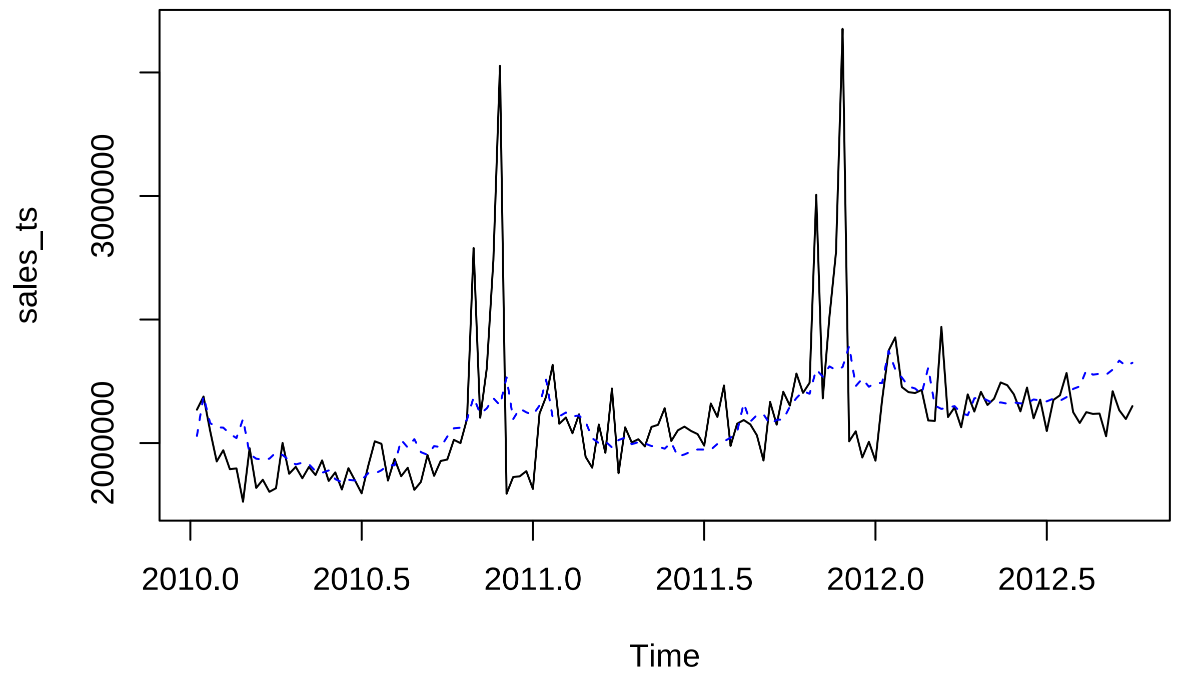
- Option 2: Drop Date column from regression

```
reg = lm(Sales ~ . - Date, data = df)
summary(reg)
```

```
##
## Call:
## lm(formula = Sales ~ . - Date, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -470922  -93407  -18493   47960 1373888
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -253212    7235433  -0.035  0.97213
## X                  7488       4048   1.850  0.06651 .
## Holiday_Flag      81698      80629   1.013  0.31275
## Temperature       -4747       1438  -3.301  0.00123 **
## Fuel_Price       -23079     112042  -0.206  0.83711
## CPI               10047      58099   0.173  0.86297
## Unemployment     148385     109794   1.351  0.17880
## Laptop_Demand      2130       4463   0.477  0.63397
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 236700 on 135 degrees of freedom
## Multiple R-squared:  0.2482, Adjusted R-squared:  0.2092
## F-statistic: 6.365 on 7 and 135 DF,  p-value: 1.792e-06
```

```
plot.ts(sales_ts)

reg_pred <- ts(predict(reg), frequency = 52, start=c(2010,2,5))
lines(reg_pred, lty="dashed", col="blue")
```



Problem 4 (2 Points)

Plot the `Laptop_Demand` column as a time series. Identify the forecasting required for this type of Time-series, and forecast the values for all 143 weeks (Hint: Look at functions in the `forecast` package)

Forecasting type is Croston's forecasting method for intermitted demand, as the demand is 0 for a lot of weeks.

```
library(forecast)
```

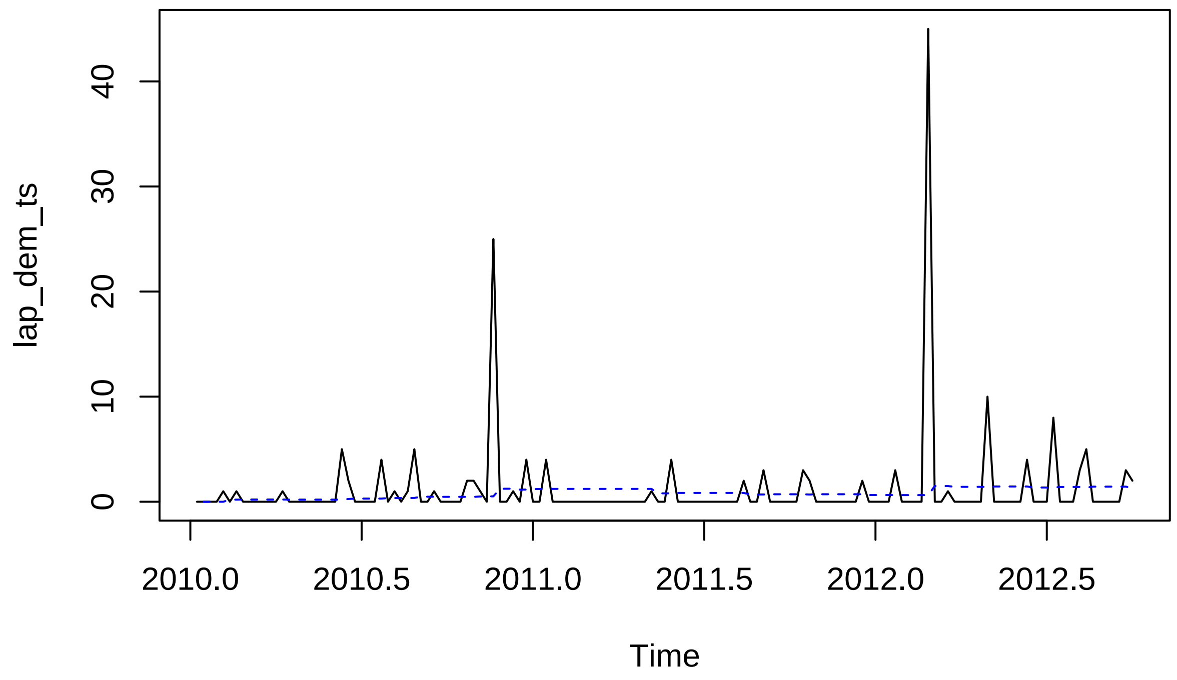
```
## Warning: package 'forecast' was built under R version 4.1.2

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

```
lap_dem <- df$Laptop_Demand
lap_dem_ts <- ts(lap_dem, frequency = 52, start = c(2010, 2, 5))
plot.ts(lap_dem_ts)

crost_fc <- croston(lap_dem_ts, h = 143)

lines(crost_fc$fitted, lty="dashed", col="blue")
```



Problem 5 (2 Points)

Evaluate the accuracy of all 3 Exponential Smoothing models (from Problem 2) and Regression models using the MAPE and RMSE metrics. Comment on which is the best Exponential Smoothing method, and if Regression is better than Exponential Smoothing? Provide a reasoning for why the best model is better suited for the `Sales` data (Bonus Point: reasoning for why the 2 other models perform similarly)

```
library(Metrics)
```

```
##
## Attaching package: 'Metrics'

## The following object is masked from 'package:forecast':
##
##     accuracy
```

```
ses_mape <- mape(sales_ts, ses_fc$fitted[,1])
des_mape <- mape(sales_ts, des_fc$fitted[,1])
tes_mape <- mape(sales_ts, tes_fc$fitted[,1])

ses_rmse <- rmse(sales_ts, ses_fc$fitted[,1])
des_rmse <- rmse(sales_ts, des_fc$fitted[,1])
tes_rmse <- rmse(sales_ts, tes_fc$fitted[,1])

reg_mape <- mape(sales_ts, predict(reg))
reg_rmse <- rmse(sales_ts, predict(reg))

c('SES Metrics:', 'MAPE:', ses_mape, 'RMSE:', ses_rmse)
```

```
## [1] "SES Metrics:"       "MAPE:"               "0.0614366285537106"
## [4] "RMSE:"               "244672.518751483"
```

```r
c('DES Metrics:', 'MAPE:', des_mape, 'RMSE:', des_rmse)
```

```
## [1] "DES Metrics:"       "MAPE:"               "0.0682364867241992"
## [4] "RMSE:"               "255053.813209909"
```

```r
c('TES Metrics:', 'MAPE:', tes_mape, 'RMSE:', tes_rmse)
```

```
## [1] "TES Metrics:"       "MAPE:"               "0.0198317461250846"
## [4] "RMSE:"               "62937.0764794706"
```

```r
c('Regression Metrics', 'MAPE:', reg_mape, 'RMSE:', reg_rmse)
```

```
## [1] "Regression Metrics" "MAPE:"               "0.0574461601494901"
## [4] "RMSE:"               "230012.096613479"
```

Triple Exponential Smoothing (Holt Winter's) is the best Exponential Smoothing method, and is also better than Regression. This is because TES takes seasonality into account, which the other models do not

Bonus Point: SES and DES perform similarly becuase although DES takes trend into account, the `Sales` data doesn't have a huge trend.