

R for Beginners - R for Finance Code File-1

This R code book written by [Rohit Dhankar](https://github.com/RohitDhankar) . GitHub - <https://github.com/RohitDhankar>

Code and Data > <https://github.com/RohitDhankar/R-Beginners-Online-Virtual-Learning-Session>

Good practice to keep track of current Working Directory , list all Objects in R ENVIRONMENT - specially so when committing changes to Git or any other version control Remote directory.

R for Finance

```
library("forecast");
library("ggplot2");
library("ggfortify")

##
## Attaching package: 'ggfortify'

## The following object is masked from 'package:forecast':
##
##      gglagplot

library("tseries")

# Set Seed -- ensure reproducible results

set.seed(123)

infy_df <- read.csv("~/Desktop/R_Own/R_Finance/DATA_Files/INFY.csv")
str(infy_df)

## 'data.frame':   494 obs. of  1 variable:
##  $ Close.Price: num  1176 1135 1150 1092 1086 ...

summary(infy_df)

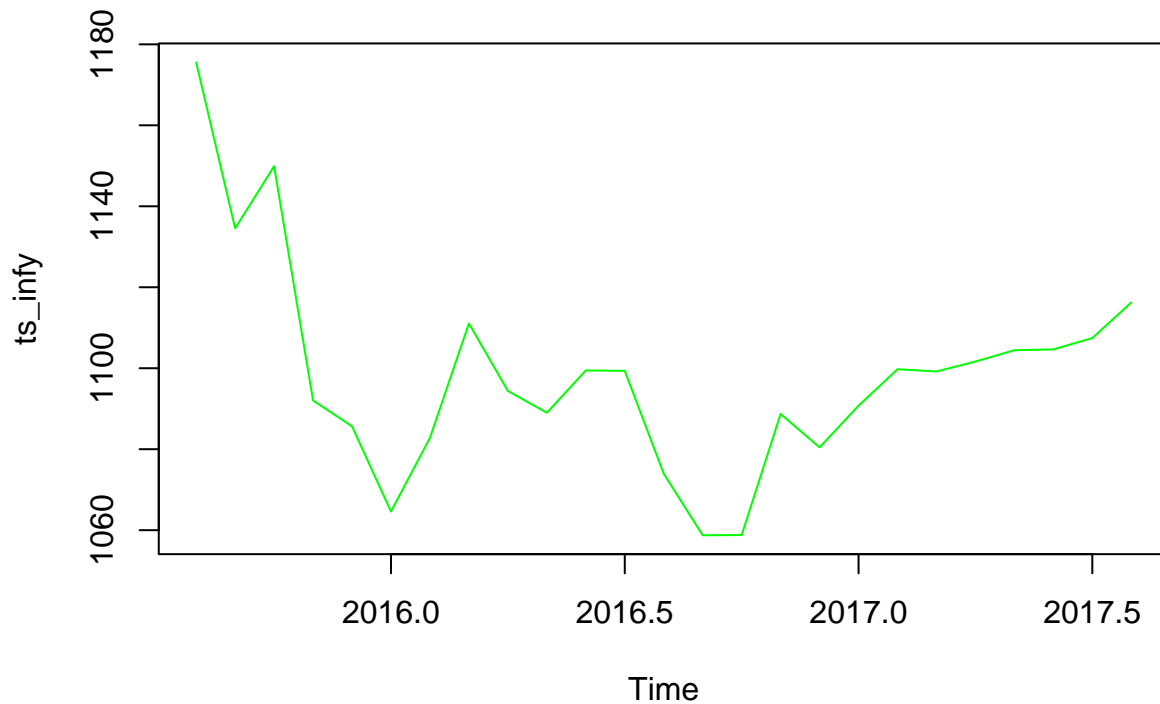
##      Close.Price
##      Min.       : 911.1
##      1st Qu.:  982.0
##      Median :1049.1
##      Mean    :1060.4
##      3rd Qu.:1134.1
##      Max.     :1267.6

start_date <- infy_df$Date[1] ## [1] 19-Aug-2015
len_df<-length(infy_df$Date)
end_date <-infy_df$Date[len_df] ## [1] 17-Aug-2017

# Convert DF to TS

ts_infy <-ts(infy_df, start = c(2015,8), end =  c(2017,8), frequency = 12) #
plot.ts(ts_infy,main="NSE-INFY-STOCK_PRICE_DAILY_CLOSING",col="green")
```

NSE-INFY-STOCK_PRICE_DAILY_CLOSING



```
#
#

#?ts() ## Uncomment to seek help
# frequency ---
# the number of observations per unit of time.
# deltat ---
# the fraction of the sampling period between successive
# observations; e.g., 1/12 for monthly data. Only one of frequency
# or deltat should be provided.
#
ts_infy1 <-ts(infy_df, start = c(2015,8), end = c(2017,8), frequency = 24,names = "NSE-INFY-STOCK_PRICE_DAILY_CLOSING")
#
str(ts_infy1);summary(ts_infy1)

## Time-Series [1:49] from 2015 to 2017: 1176 1135 1150 1092 1086 ...
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      1059   1095    1107    1117   1141    1182
#
typeof(ts_infy1) ## [1] "double"

## [1] "double"
#
typeof(ts_infy) ## [1] "double"

## [1] "double"
#
class(ts_infy)
```

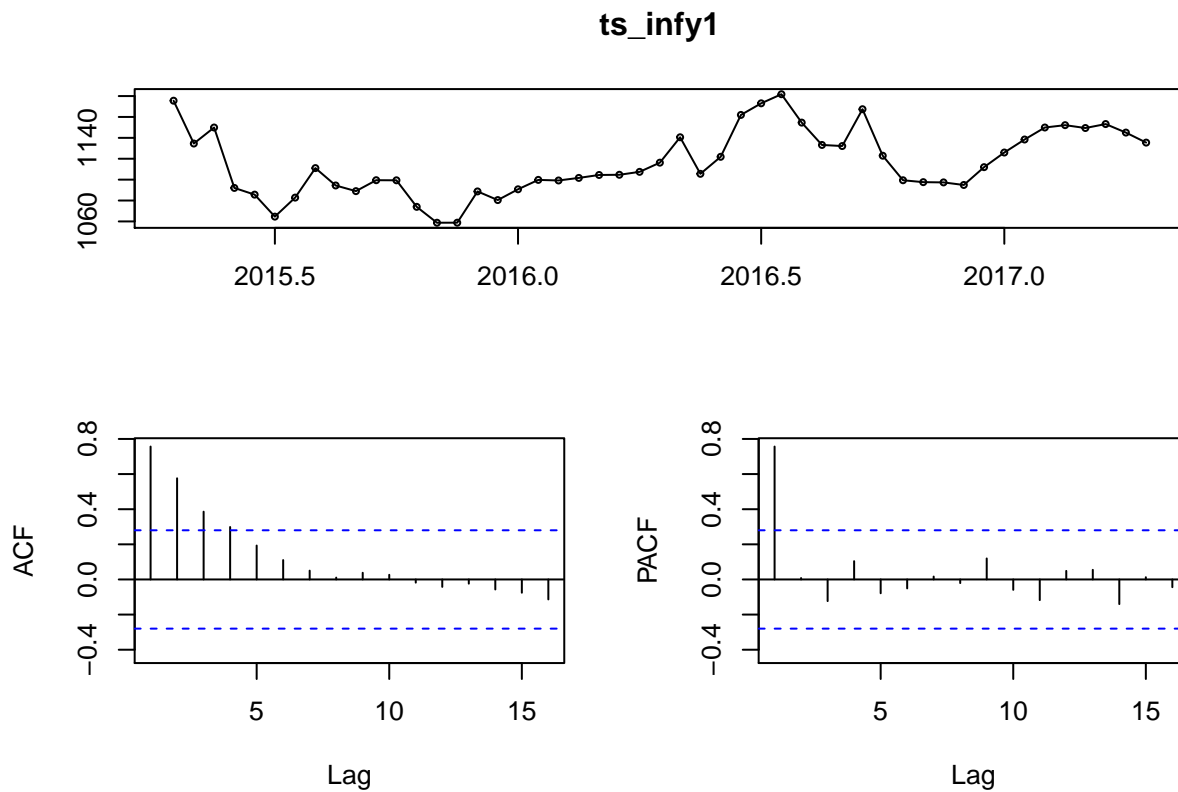
```
## [1] "ts"
#
head(ts_infy1);head(time(ts_infy1));tail(time(ts_infy1))

## Time Series:
## Start = c(2015, 8)
## End = c(2015, 13)
## Frequency = 24
## [1] 1175.55 1134.55 1149.90 1092.05 1085.65 1064.60

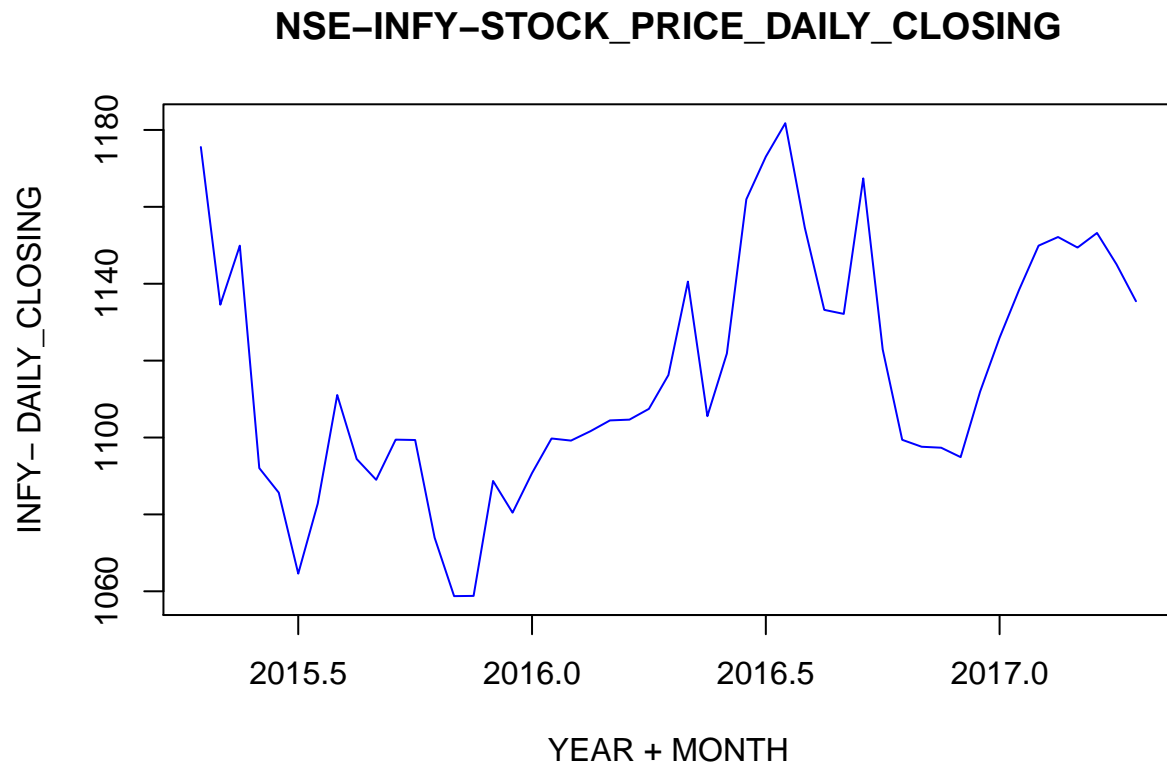
## Time Series:
## Start = c(2015, 8)
## End = c(2015, 13)
## Frequency = 24
## [1] 2015.292 2015.333 2015.375 2015.417 2015.458 2015.500

## Time Series:
## Start = c(2017, 3)
## End = c(2017, 8)
## Frequency = 24
## [1] 2017.083 2017.125 2017.167 2017.208 2017.250 2017.292
# Head of TS # view Head - sampled times # view Tail of sampled times
dim(ts_infy1);

## NULL
#
tsdisplay(ts_infy1)
```



```
#
plot.ts(ts_infy1,main="NSE-INFY-STOCK_PRICE_DAILY_CLOSING",col="blue",xlab="YEAR + MONTH",ylab="INFY- D
```



Decomposition of TIME SERIES :-

" to construct, from an observed time series, a number of component series (that could be used to reconstruct the original by additions or multiplications) where each of these has a certain characteristic or type of behaviour.

SOURCE – WIKI – https://en.wikipedia.org/wiki/Decomposition_of_time_series

There are many lively discussions with regards which functions from the STATS package to be used for decomposition i am referring a few below here - i shall experiment with a couple of methods and see what works best for best.

```
#
```

Various Sources —

STACK_EXCHANGE – <https://stats.stackexchange.com/questions/9506/stl-trend-of-time-series-using-r?rq=1>

STACK_EXCHANGE – <https://stats.stackexchange.com/questions/85987/which-is-better-stl-or-decompose>

ESTIMATING and ELIMINATING THE TREND

We need to identify , then estimate and eliminate the trend component if present in the Time Series Model .We also endeavour to eliminate the other TWO Deterministic Features of the TS MODEL

We are dealing with DISCRETE data of Stock Price Closing Values as taken from the NSE on 18 AUG 17

SOURCE — <http://userwww.sfsu.edu/efc/classes/biol710/timeseries/TimeSeriesAnalysis.html>

Learning pointers to be highlighted from the above cited source -

STATIONARY TIME SERIES DATA -

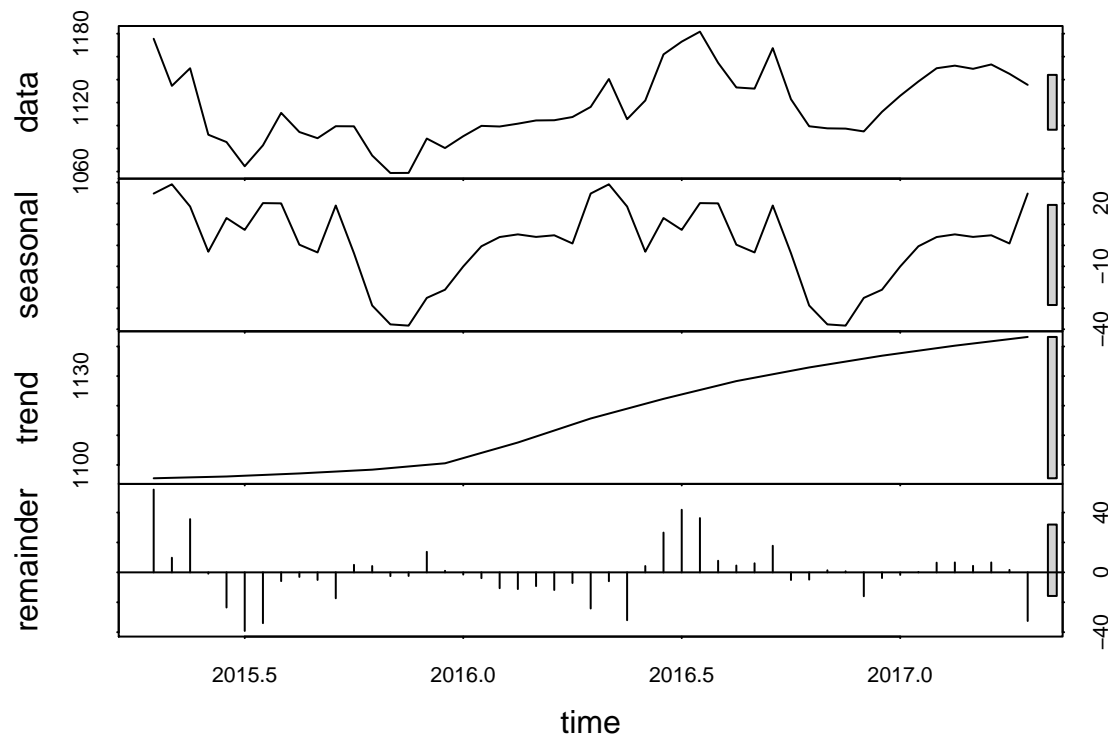
Definitions – TBD

NON STATIONARY TIME SERIES DATA -

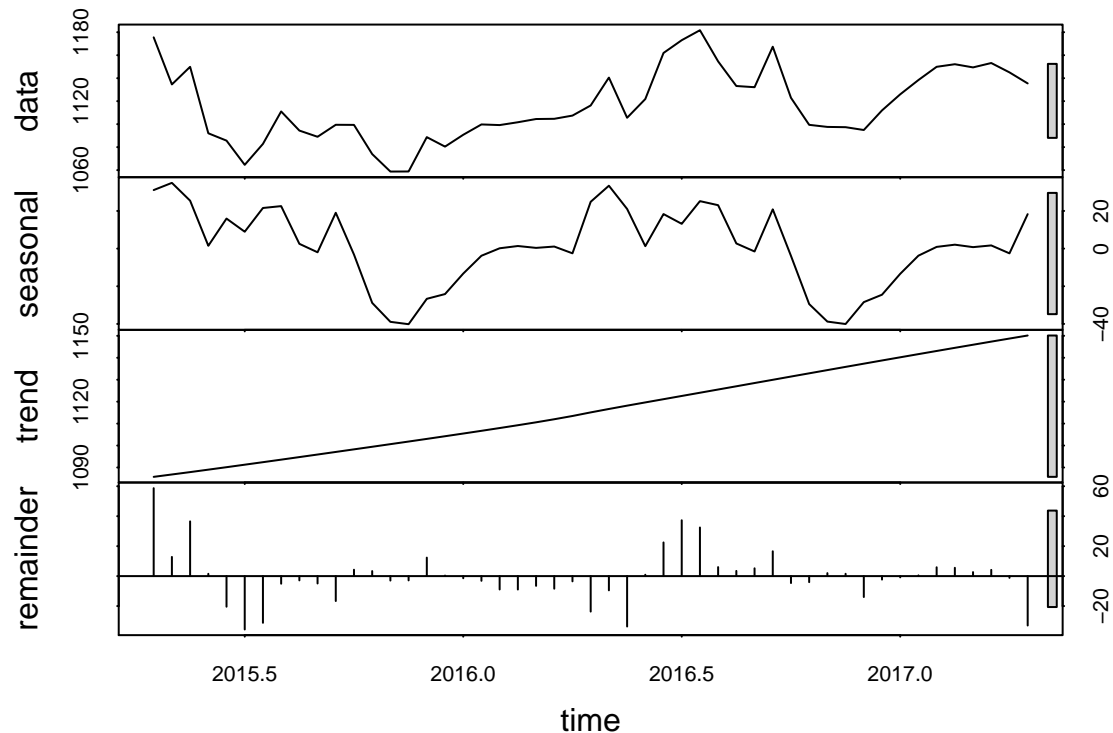
Quoting verbatim from the source mentioned above – “Box and Jenkins developed the AutoRegressive Integrative Moving Average (ARIMA) model which combined the AutoRegressive (AR) and Moving Average (MA) models developed earlier with a differencing factor that removes in trend in the data.”

Further Reading – Multiple Sources :- WIKI —

```
##?stl() # stl {stats}  
# Decompose a time series into seasonal, trend and irregular  
# components using loess, acronym STL.  
  
require(graphics)  
  
plot(stl(ts_infy1, "per"))
```



```
plot(stl(ts_infy1, s.window = 7, t.window = 50, t.jump = 1))
```



```
#plot(stllc <- stl(log(co2), s.window = 21))
#summary(stllc)
```

```
dcomp_infy_1 <- stl(ts_infy1, s.window = 12, t.window = 10) ; head(dcomp_infy_1$time.series) ; str(dcomp_infy_1)
```

```
## Time Series:
## Start = c(2015, 8)
## End = c(2015, 13)
## Frequency = 24
##      seasonal      trend remainder
## 2015.292 20.854384 1142.521 12.174863
## 2015.333 19.671386 1129.986 -15.107597
## 2015.375 10.343246 1118.022 21.534308
## 2015.417 -10.910514 1106.855 -3.894247
## 2015.458  6.329533 1096.497 -17.176779
## 2015.500  3.019051 1086.892 -25.311028

## List of 8
## $ time.series: Time-Series [1:49, 1:3] from 2015 to 2017: 20.85 19.67 10.34 -10.91 6.33 ...
## .. attr(*, "dimnames")=List of 2
## .. ..$ : NULL
## .. ..$ : chr [1:3] "seasonal" "trend" "remainder"
## $ weights      : num [1:49] 1 1 1 1 1 1 1 1 1 1 ...
## $ call         : language stl(x = ts_infy1, s.window = 12, t.window = 10)
## $ win          : Named num [1:3] 12 10 25
## .. attr(*, "names")= chr [1:3] "s" "t" "l"
## $ deg          : Named int [1:3] 0 1 1
## .. attr(*, "names")= chr [1:3] "s" "t" "l"
## $ jump         : Named num [1:3] 2 1 3
```

```
##   ..- attr(*, "names")= chr [1:3] "s" "t" "l"
##   $ inner      : int 2
##   $ outer      : int 0
##   - attr(*, "class")= chr "stl"
```

```
typeof(dcomp_infy_1) # LIST
```

```
## [1] "list"
```

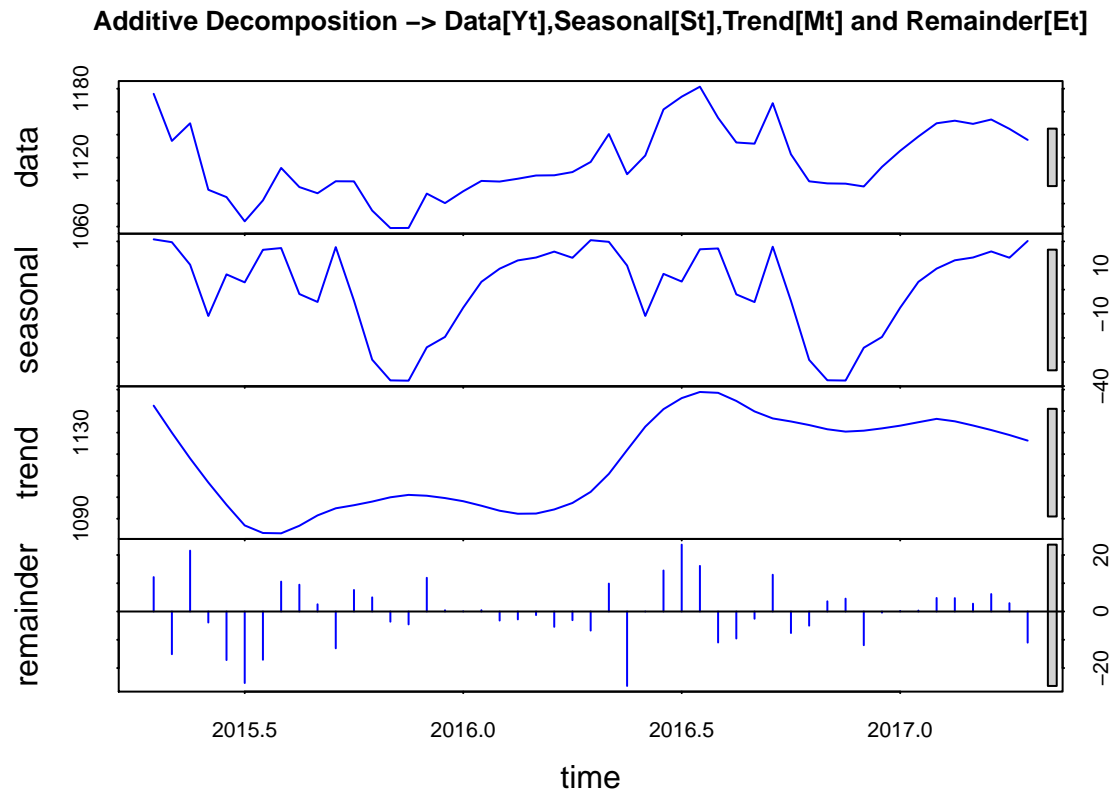
```
summary(dcomp_infy_1)
```

```
## Call:
## stl(x = ts_infy1, s.window = 12, t.window = 10)
##
## Time.series components:
##      seasonal      trend      remainder
## Min.      :-37.72765   Min.      :1083.233   Min.      :-26.341639
## 1st Qu.: -7.48136     1st Qu.:1096.497   1st Qu.: -5.424921
## Median :  6.32953     Median :1121.906   Median :  0.074595
## Mean    :  0.41636     Mean    :1116.466   Mean     : -0.376051
## 3rd Qu.: 15.76663     3rd Qu.:1133.521   3rd Qu.:  5.006186
## Max.    : 20.85438     Max.    :1148.884   Max.     : 23.671716
## IQR:
##      STL.seasonal STL.trend STL.remainder data
##      23.25        37.02    10.43         45.65
##      % 50.9        81.1     22.9         100.0
##
## Weights: all == 1
##
## Other components: List of 5
## $ win : Named num [1:3] 12 10 25
## $ deg : Named int [1:3] 0 1 1
## $ jump: Named num [1:3] 2 1 3
## $ inner: int 2
## $ outer: int 0
```

```
# We focus on the IQR Values ---
```

```
# IQR:
#      STL.seasonal STL.trend STL.remainder data
#      23.25        37.02    10.43         45.65
#      % 50.9        81.1     22.9         100.0
```

```
plot(dcomp_infy_1,main="Additive Decomposition -> Data[Yt],Seasonal[St],Trend[Mt] and Remainder[Et]",col
```



```
#
# Start the clock!
# ptm <- proc.time()
#
# vec_gross_sale <- p_sale_count_rnd*p_sale_cost_rnd
#
# summary(vec_gross_sale)
#
# proc.time() - ptm
#
#
# As seen below in our case
# ELAPSED time - 1st 0.011 , 2nd - 0.012
# Thus the WALL CLOCK or REAL / ELAPSED
# timings are almost same .
#
# The USER TIME and SYSTEM TIME's in our case
# add upto -
# 1st - 0.008
# 2nd - 0.012
#
# Thus it would seem we are better off
# with Vector Multiplication
#
# But we also need to consider
# once we have the "vec_gross_sale"
# we will need to add it to our "mdf"
```



```

# Kindly also note the Timings will
# differ for each system - also for each run
# of the chunk of code on same sys

# Definition of user Time --- The 'user time' is the CPU time
# charged for execution of user instructions of the calling process.
#
# REFER- https://stat.ethz.ch/R-manual/R-devel/library/base/html/proc.time.html

# Now to multiply TWO Columns of the DF
# Also called COLUMNAR VECTORS

# Again start the clock!
# ptm <- proc.time()
#
# mdf$gross_sale<- mdf$p_sale_count_rnd*mdf$p_sale_cost_rnd
#
# proc.time() - ptm
# #
# str(mdf)
# #
# summary(mdf)
# #
# write.csv(mdf,file="Mkt_DATA_Files/mdf.csv")
# ## Writes to Sub Directory - DATA_Files
# #

```