R Project: Time Series Forecasting of Property Sales in England and Wales

Gaurav Kumar

2023-06-25

Forecasting Property Sales in England and Wales

This project aims to build a forecasting model to predict property sales in England and Wales based on monthly data from Jan 1995 to Nov 2016. The intention is to predict property sales for next one year ie from Dec 2016 to Nov 2017.

Step 1: Loading 'forecast' package which is the main package to conduct forecasting

```
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 4.2.2
```

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

Step 2: Importing the dataset

```
data <- read.csv("propertysales.csv")
```

Step 3: Data exploration : Viewing first 6 rows of data

```
head(data)
```

```
##        X Total_Sales Flats_Sales Terraced_Sales Semidetached_Sales
## 1 Jan-95       50497        7561          16309              15428
## 2 Feb-95       50856        7390          16425              15491
## 3 Mar-95       71357       10127          23084              21941
## 4 Apr-95       60336        8551          19079              18448
## 5 May-95       67860        9594          21550              20923
## 6 Jun-95       77005       10852          23818              23587
##   Detached_Sales All_AveragePrice Flats_AveragePrice Terraced_AveragePrice
## 1          11198            55130              48751                 42440
## 2          11550            54195              47049                 41458
## 3          16205            54478              47430                 42090
## 4          14258            55969              48374                 42931
## 5          15793            55639              48080                 42853
## 6          18748            56248              48904                 42942
```

```
##   Semi.detached_AveragePrice Detached_AveragePrice
## 1                      53606                 91095
## 2                      53526                 88296
## 3                      53202                 88582
## 4                      54175                 90857
## 5                      54330                 89607
## 6                      54011                 90441
```

Step 4: Data exploration: Viewing summary of data

```r
summary(data)
```

```
##       X              Total_Sales      Flats_Sales    Terraced_Sales 
##  Length:263         Min.   : 27532   Min.   : 4497   Min.   : 8196  
##  Class :character   1st Qu.: 64328   1st Qu.:11135   1st Qu.:18992  
##  Mode  :character   Median : 84349   Median :14910   Median :25887  
##                     Mean   : 83336   Mean   :15191   Mean   :25687  
##                     3rd Qu.:100917   3rd Qu.:18619   3rd Qu.:31501  
##                     Max.   :142750   Max.   :38646   Max.   :44972  
##  Semidetached_Sales Detached_Sales  All_AveragePrice Flats_AveragePrice
##  Min.   : 7613      Min.   : 6178   Min.   : 54195   Min.   : 46586    
##  1st Qu.:17844      1st Qu.:15150   1st Qu.: 82611   1st Qu.: 83156    
##  Median :23002      Median :19260   Median :161388   Median :150537    
##  Mean   :23035      Mean   :19199   Mean   :140662   Mean   :132864    
##  3rd Qu.:28256      3rd Qu.:23733   3rd Qu.:185512   3rd Qu.:172855    
##  Max.   :40074      Max.   :32950   Max.   :220292   Max.   :217184    
##  Terraced_AveragePrice Semi.detached_AveragePrice Detached_AveragePrice
##  Min.   : 41240        Min.   : 53026             Min.   : 87909       
##  1st Qu.: 60822        1st Qu.: 75659             1st Qu.:133033       
##  Median :128121        Median :157257             Median :256277       
##  Mean   :110761        Mean   :132102             Mean   :218228       
##  3rd Qu.:150080        3rd Qu.:173143             3rd Qu.:283438       
##  Max.   :177897        Max.   :205165             Max.   :332403       
```

Step 5: Creating a time-series object:
A dataframe cannot be used directly for forecasting. It has to be converted to a time-series object first. Here, "Total_Sales" feature of data is selected for forecasting. Frequency is 12 because data is monthly.

```r
psts <- ts(data$Total_Sales, start = c(1995,1), end = c(2016,11), frequency = 12)
```

Step 6: Viewing the time-series object

```r
psts
```

```
##         Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
## 1995  50497  50856  71357  60336  67860  77005  67889  73320  71529  65088
## 1996  51471  54475  73169  68252  83595  79978  90201  96488  84843  91547
## 1997  70750  70384  81414  83628  96923 101011 108808 104745  94995 101821
## 1998  64730  66497  81670  84519  91728  97105 109611  94242  90208  93760
## 1999  63530  65911  88825  94175  94169 106227 126193 114320 108228 113622
## 2000  72281  77068 102762  94538  98130 112302 101678 100823  94327  88910
## 2001  69087  71732  96341  94222 105664 120902 120583 132913 104779 112757
```

```
## 2002  76717  82700 110415 106557 141076 111555 134946 134874 111165 113383
## 2003  85399  81101  86266  89160  98359 101346 109894 115622 111015 125027
## 2004  90492  90263 105047 118810 109162 123447 131790 113736  98157  94423
## 2005  57029  59745  72160  84349  85182  97903 102855 100640 102199  94280
## 2006  77522  80562 105745  96992 107083 128150 119167 125702 120783 117911
## 2007  90597  89460 109782  98497 111506 127693 120633 128673 101485 106241
## 2008  59824  63766  60358  64943  67334  61177  54518  49951  43282  46735
## 2009  27532  28488  37541  40503  46570  54410  64057  59198  59387  66680
## 2010  36444  43097  52307  53022  52932  63760  68286  62307  58332  59484
## 2011  38210  40183  47492  51405  49644  58678  63137  65239  64439  58763
## 2012  44476  45468  62139  43947  53261  60669  59944  65837  53504  59752
## 2013  43714  45538  55288  51586  67165  67551  74942  81273  72902  82700
## 2014  71030  70450  74393  76672  84272  88790  91844  94631  85365  93443
## 2015  62810  64217  74848  70915  80893  93425 100545  91670  90504  99747
## 2016  68945  75546 142750  59510  67935  81999  83820  82324  75663  64785
##         Nov    Dec
## 1995  67945  73525
## 1996  99251  91737
## 1997  86611  92933
## 1998  86683  89072
## 1999 110023 108752
## 2000  89023  97102
## 2001 116805  99761
## 2002 118988 109540
## 2003 117396 121075
## 2004  83795  85704
## 2005  97243 108444
## 2006 119965 127079
## 2007 104539  84675
## 2008  37707  41590
## 2009  62031  79669
## 2010  57139  57745
## 2011  61211  64582
## 2012  64614  57306
## 2013  89480  86552
## 2014  80181  85784
## 2015  90640  95815
## 2016  31227
```

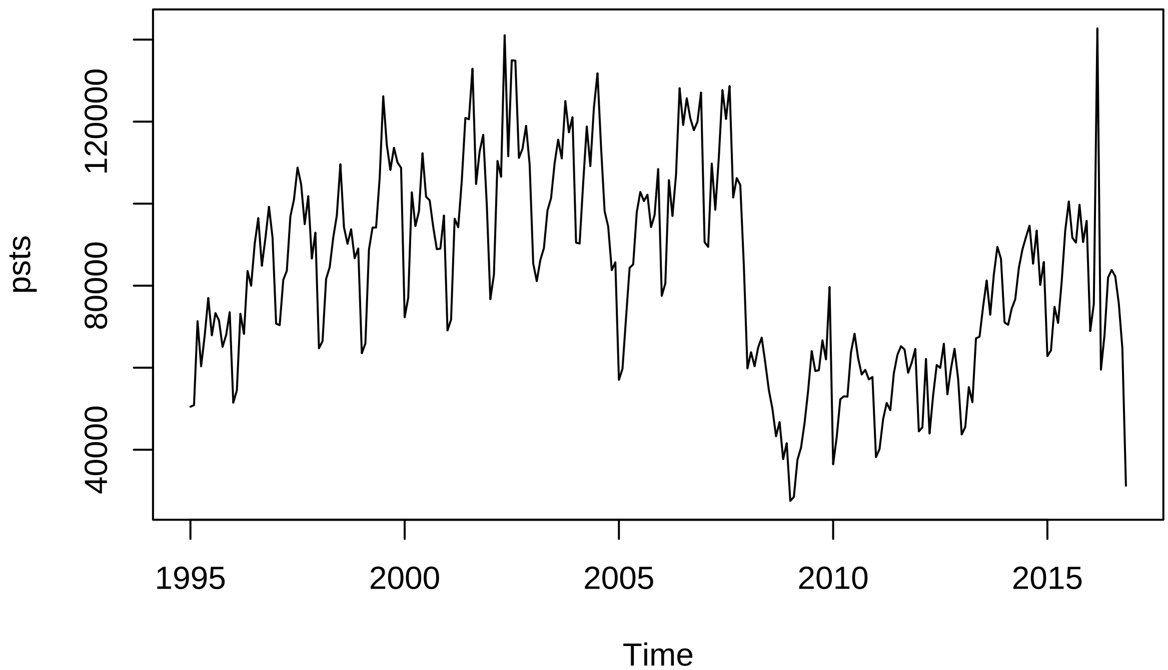
Step 7: Checking the length of time-series object

```
length(psts)
```

```
## [1] 263
```

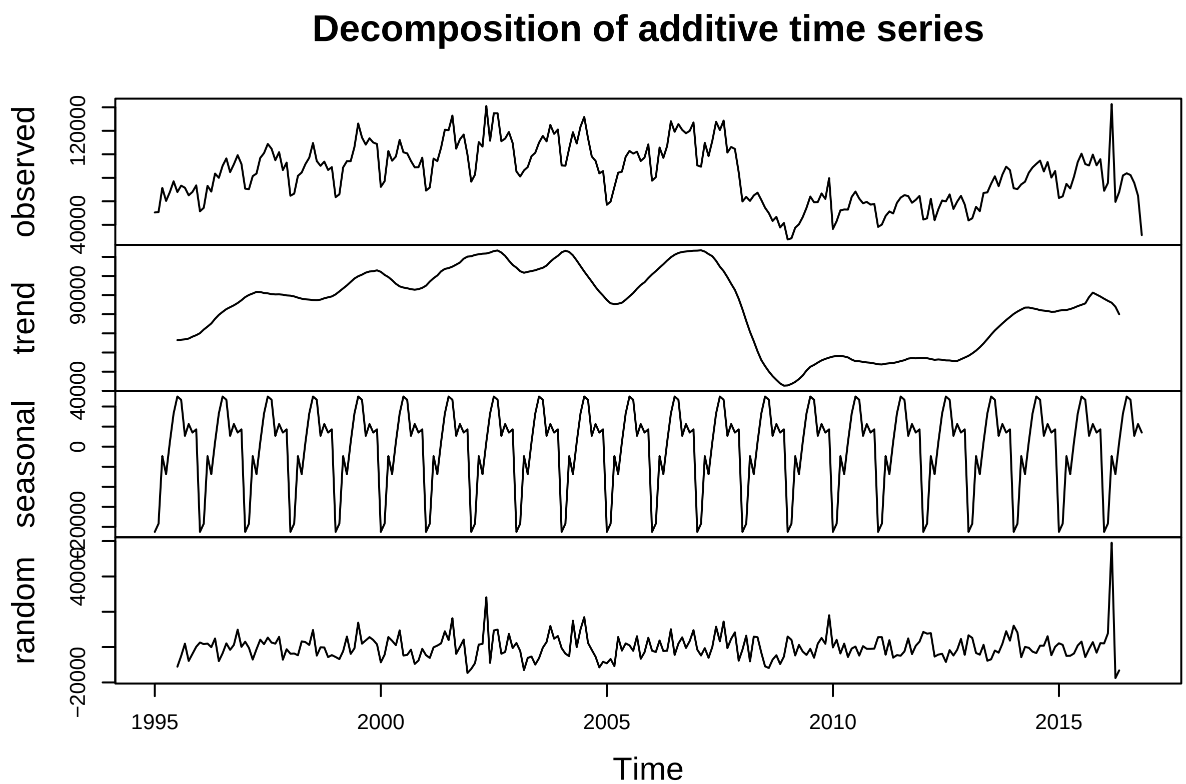
Step 8: Plotting the time-series object

```
plot(psts)
```



Step 9: Decomposing the time series to understand series better:

```
plot(decompose(psts))
```



The decomposition shows presence of seasonality in time series.

Step 10: Partitioning data into train and test data sets:
Here, last twenty months are excluded to form training data set. Consequently, last 20 months will form test data set.

```
train <- subset(psts, end = length(psts)-20)
test <- tail(psts,20)
```

Step 11: Viewing the training data set

```
train
```

```
##         Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
## 1995  50497  50856  71357  60336  67860  77005  67889  73320  71529  65088
## 1996  51471  54475  73169  68252  83595  79978  90201  96488  84843  91547
## 1997  70750  70384  81414  83628  96923 101011 108808 104745  94995 101821
## 1998  64730  66497  81670  84519  91728  97105 109611  94242  90208  93760
## 1999  63530  65911  88825  94175  94169 106227 126193 114320 108228 113622
## 2000  72281  77068 102762  94538  98130 112302 101678 100823  94327  88910
## 2001  69087  71732  96341  94222 105664 120902 120583 132913 104779 112757
## 2002  76717  82700 110415 106557 141076 111555 134946 134874 111165 113383
## 2003  85399  81101  86266  89160  98359 101346 109894 115622 111015 125027
## 2004  90492  90263 105047 118810 109162 123447 131790 113736  98157  94423
## 2005  57029  59745  72160  84349  85182  97903 102855 100640 102199  94280
## 2006  77522  80562 105745  96992 107083 128150 119167 125702 120783 117911
```

```
## 2007   90597   89460  109782   98497  111506  127693  120633  128673  101485  106241
## 2008   59824   63766   60358   64943   67334   61177   54518   49951   43282   46735
## 2009   27532   28488   37541   40503   46570   54410   64057   59198   59387   66680
## 2010   36444   43097   52307   53022   52932   63760   68286   62307   58332   59484
## 2011   38210   40183   47492   51405   49644   58678   63137   65239   64439   58763
## 2012   44476   45468   62139   43947   53261   60669   59944   65837   53504   59752
## 2013   43714   45538   55288   51586   67165   67551   74942   81273   72902   82700
## 2014   71030   70450   74393   76672   84272   88790   91844   94631   85365   93443
## 2015   62810   64217   74848
##          Nov     Dec
## 1995   67945   73525
## 1996   99251   91737
## 1997   86611   92933
## 1998   86683   89072
## 1999  110023  108752
## 2000   89023   97102
## 2001  116805   99761
## 2002  118988  109540
## 2003  117396  121075
## 2004   83795   85704
## 2005   97243  108444
## 2006  119965  127079
## 2007  104539   84675
## 2008   37707   41590
## 2009   62031   79669
## 2010   57139   57745
## 2011   61211   64582
## 2012   64614   57306
## 2013   89480   86552
## 2014   80181   85784
## 2015
```

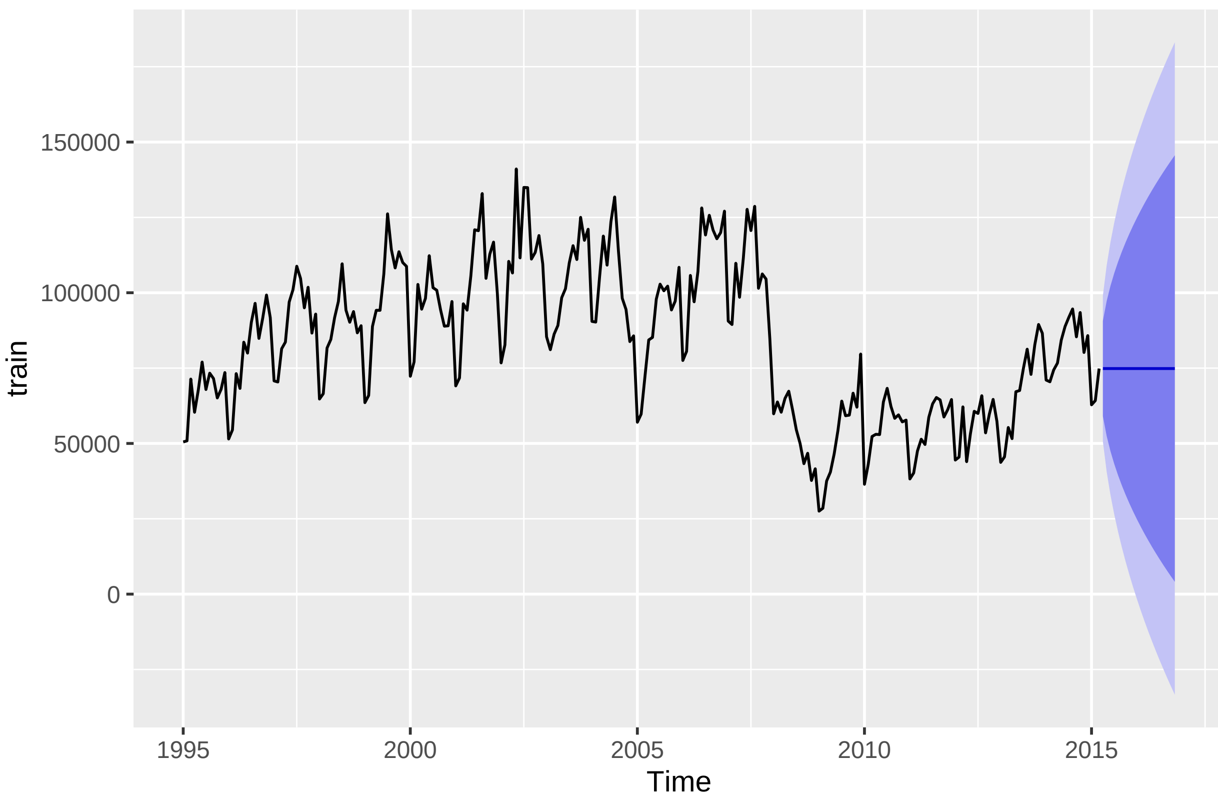
Now, starting with **Simple Forecasting Methods**: Under this, 4 methods would be deployed, which are:
1. Naive forecast
2. Seasonal Naive forecast
3. Mean forecast
4. Drift forecast

Step 12: So, first starting with *Naive forecast* where forecast is the copy of last value in the data set.

```
fcnaive <- naive(train, h = 20)
```

Step 13: Plotting naive forecast:

```
autoplot(train) + autolayer(fcnaive)
```

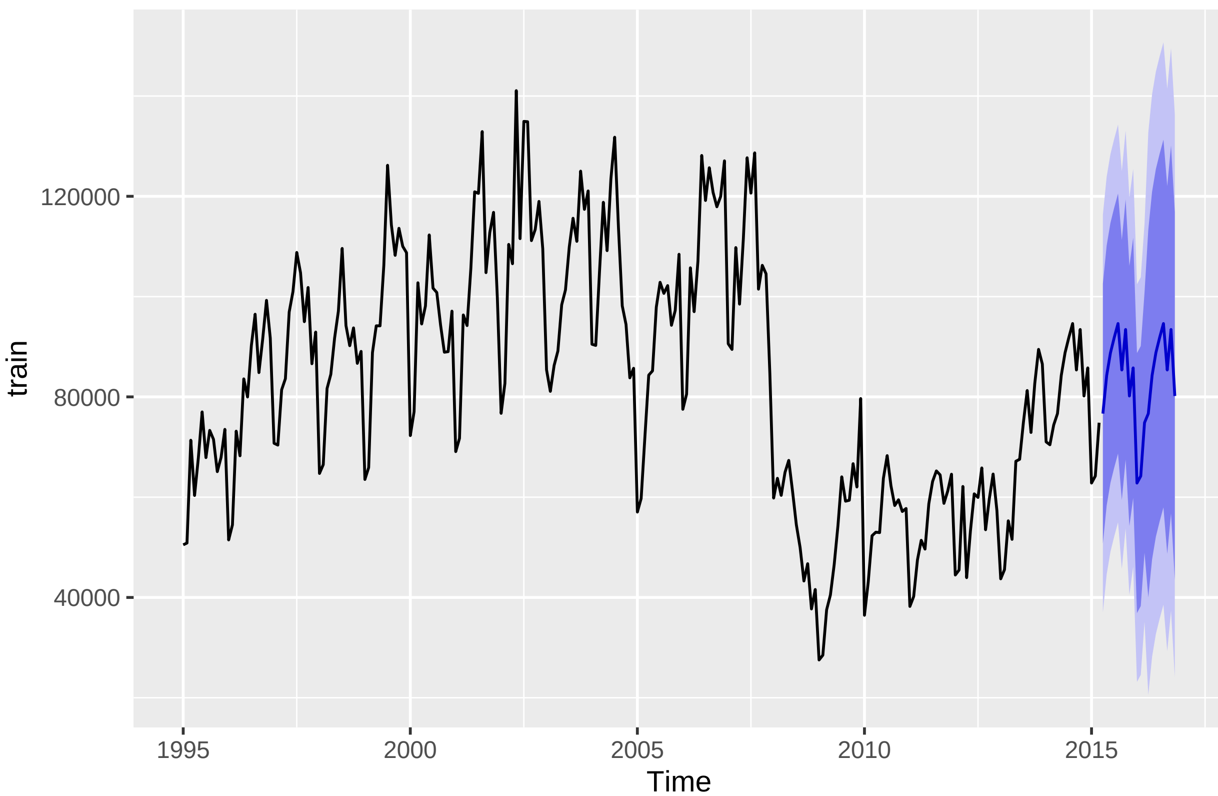


Step 14: Performing *Seasonal Naive* forecast: Here, the forecast will be based on historical values of the particular previous season.

```
fcsnaive <- snaive(train, h = 20)
```

Step 15: Plotting Seasonal Naive forecast:

```
autoplot(train) + autolayer(fcsnaive)
```

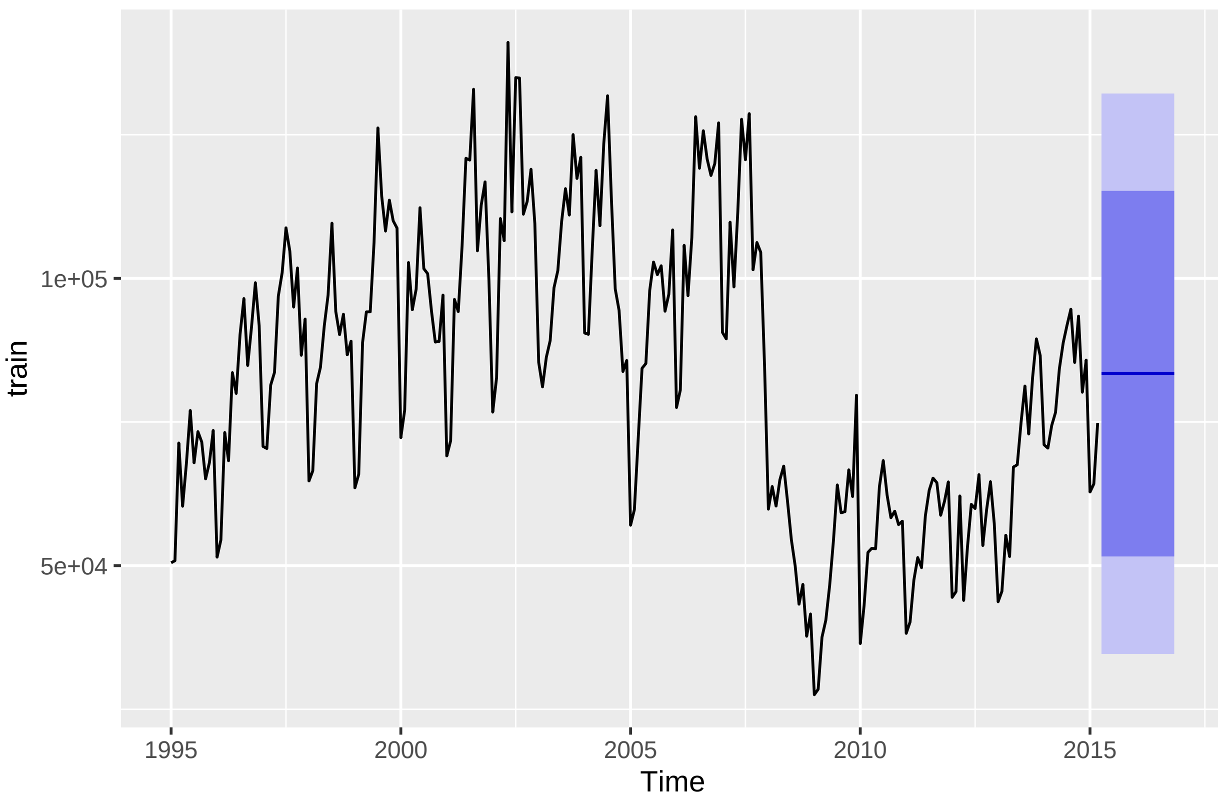


Step 16: Performing *Mean* forecast:
Here, the forecast is based on the mean of historical data.

```
fcmean <- meanf(train, h = 20)
```

Step 17: Plotting Mean forecast

```
autoplot(train) + autolayer(fcmean)
```

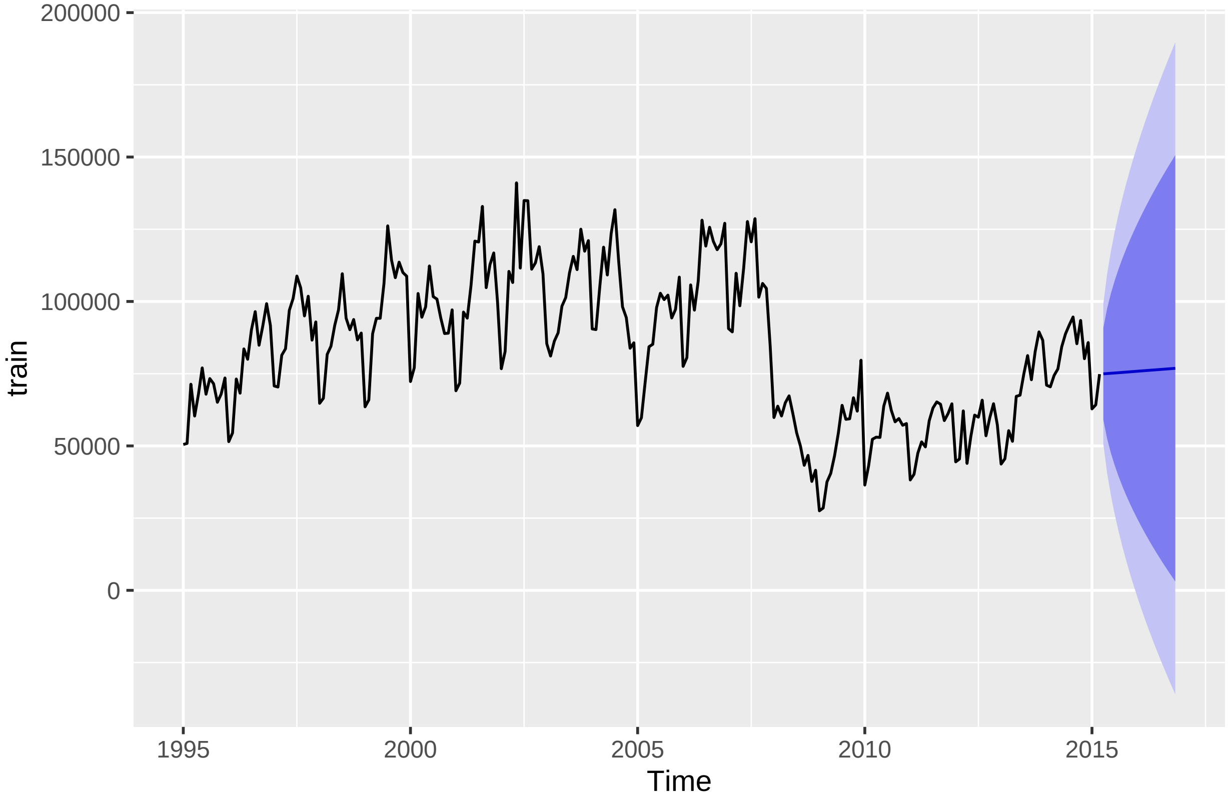


Step 18: Performing *Drift* forecast:
Here, the forecast is based on average change seen in historical data.

```
fcdrift <- rwf(train, h = 20, drift = TRUE)
```

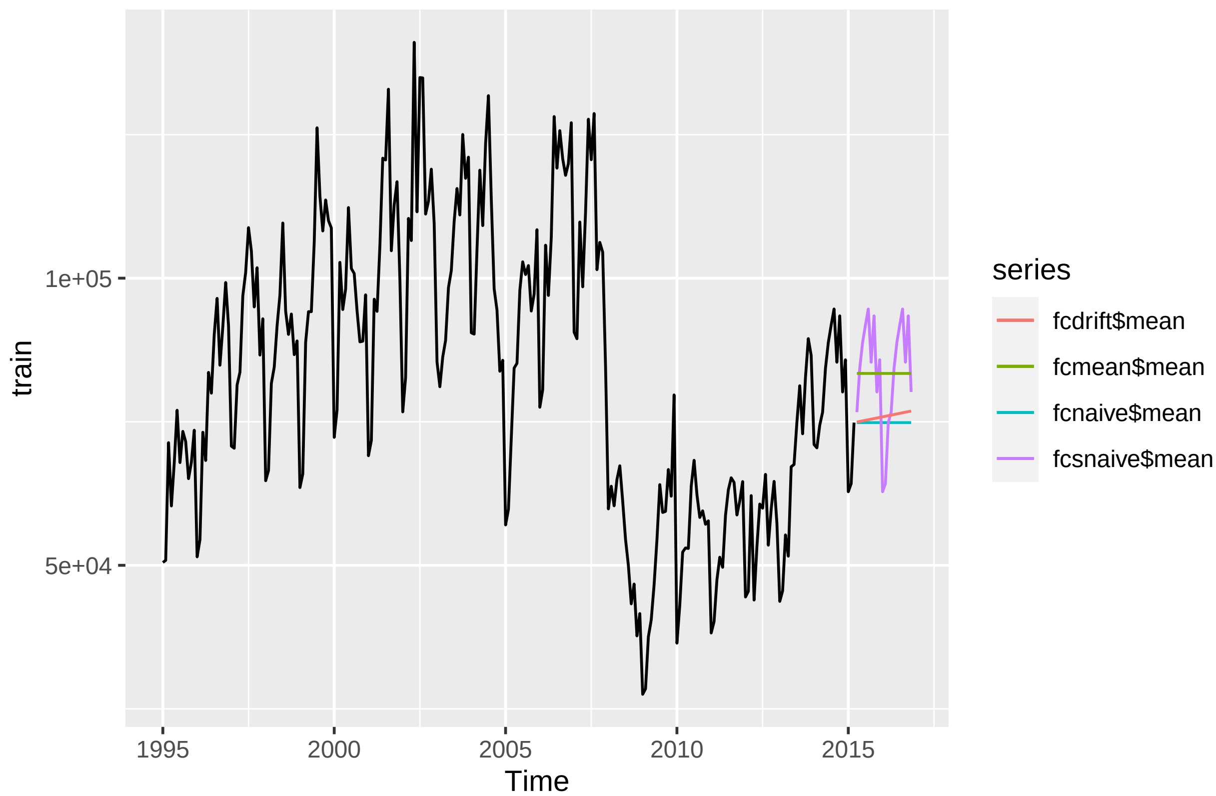
Step 19: Plotting Drift forecast:

```
autoplot(train) + autolayer(fcdrift)
```



Step 20: Plotting all four methods used above:

```
autoplot(train) + autolayer(fcnaive$mean) + autolayer(fcsnaive$mean) + autolayer(fcdrift$mean) + autola
```



Step 21: Generating the performance metrics of all 4 methods deployed above:

```
# Naive Forecast
accuracy(fcnaive,psts)
```

```
##                     ME     RMSE       MAE       MPE     MAPE      MASE
## Training set  100.624 12348.01  9283.996 -1.150741 12.06621 0.6160241
## Test set     7584.900 22378.34 16162.000  1.281270 21.80933 1.0724027
##                     ACF1 Theil's U
## Training set -0.17105672        NA
## Test set      0.03491632  1.008418
```

```
# Seasonal Naive Forecast
accuracy(fcsnaive,psts)
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set   803.6797 20229.22 15070.83 -2.866127 20.72605 1.0000000
## Test set     -1469.8500 21608.62 14533.35 -9.797985 21.70629 0.9643363
##                   ACF1 Theil's U
## Training set 0.8590534        NA
## Test set     0.2497280  1.025775
```

```
# Mean Forecast
accuracy(fcmean,psts)
```

```
##                          ME     RMSE      MAE       MPE     MAPE      MASE
## Training set  3.702746e-12 24655.34 20838.15 -10.88582 29.61719 1.3826809
## Test set     -9.779107e+02 21076.43 14799.78 -10.01242 22.40877 0.9820149
##                     ACF1 Theil's U
## Training set 0.87118813        NA
## Test set     0.03491632 0.9783045
```

```r
# Drift Forecast
accuracy(fcdrift,psts)
```

```
##                         ME     RMSE       MAE        MPE     MAPE      MASE
## Training set 5.992635e-14 12347.60  9266.532 -1.2842390 12.04635 0.6148654
## Test set     6.528348e+03 22268.83 16020.762 -0.2457653 21.99498 1.0630311
##                     ACF1 Theil's U
## Training set -0.17105672        NA
## Test set      0.05151713  1.008491
```

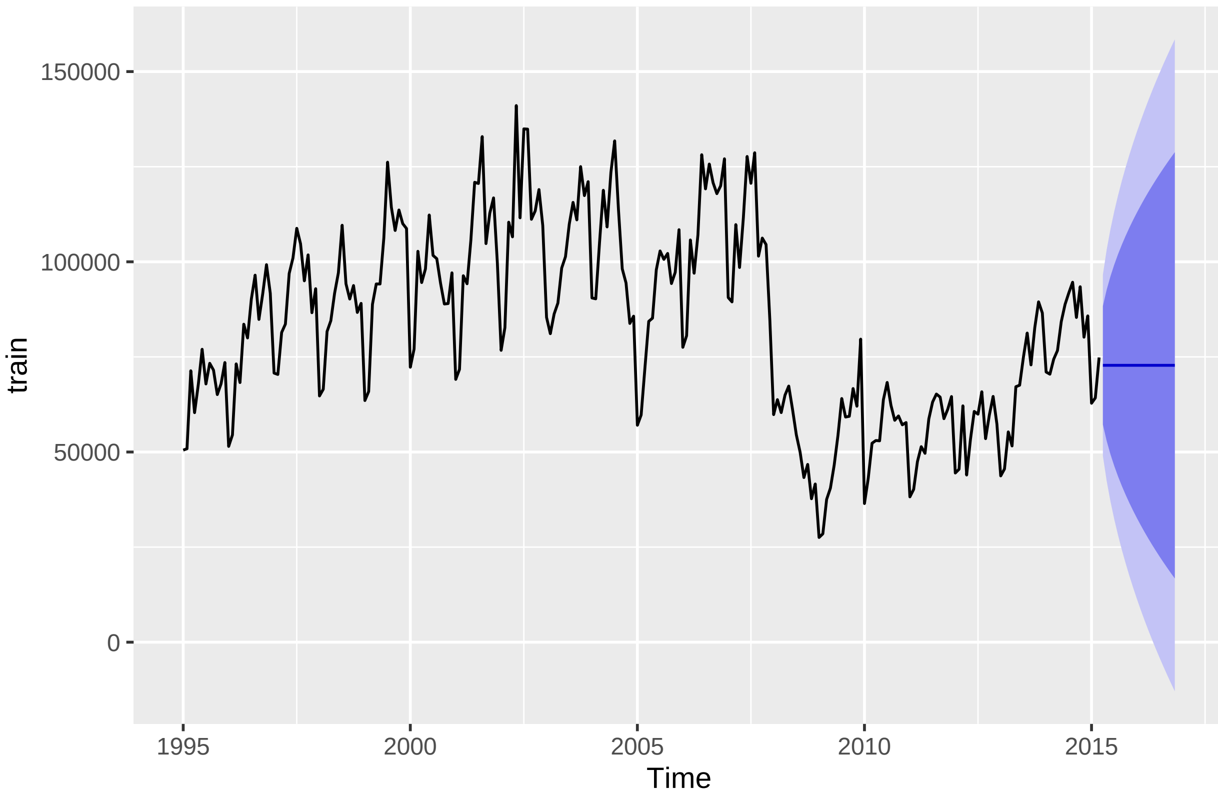
Step 22: Performing **Exponential Smoothing** forecast methods: Under this, 4 methods will be used:
1. Simple Exponential Smoothing (SES)
2. Holt Linear Method
3. Additive Holt-Winters Method
4. Multiplicative Holt-Winters Method

Starting, with *Simple Exponential Smoothing* forecast:

```r
fcses <- ses(train, h = 20)
```

Step 23: Plotting SES forecast:

```r
autoplot(train) + autolayer(fcses)
```

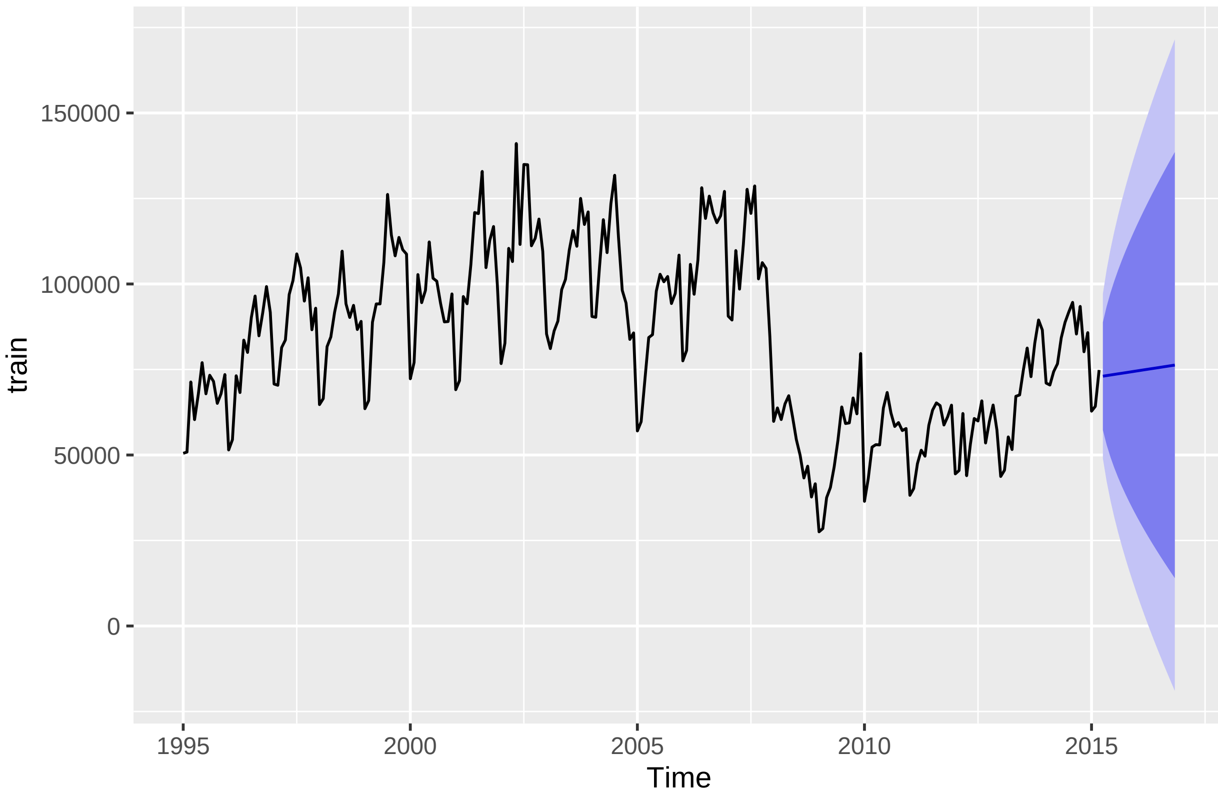


Step 24: Performing *Holt Linear Method* forecast:

```
fcholt <- holt(train, h = 20)
```

Step 25: Plotting Holt Linear forecast:

```
autoplot(train) + autolayer(fcholt)
```

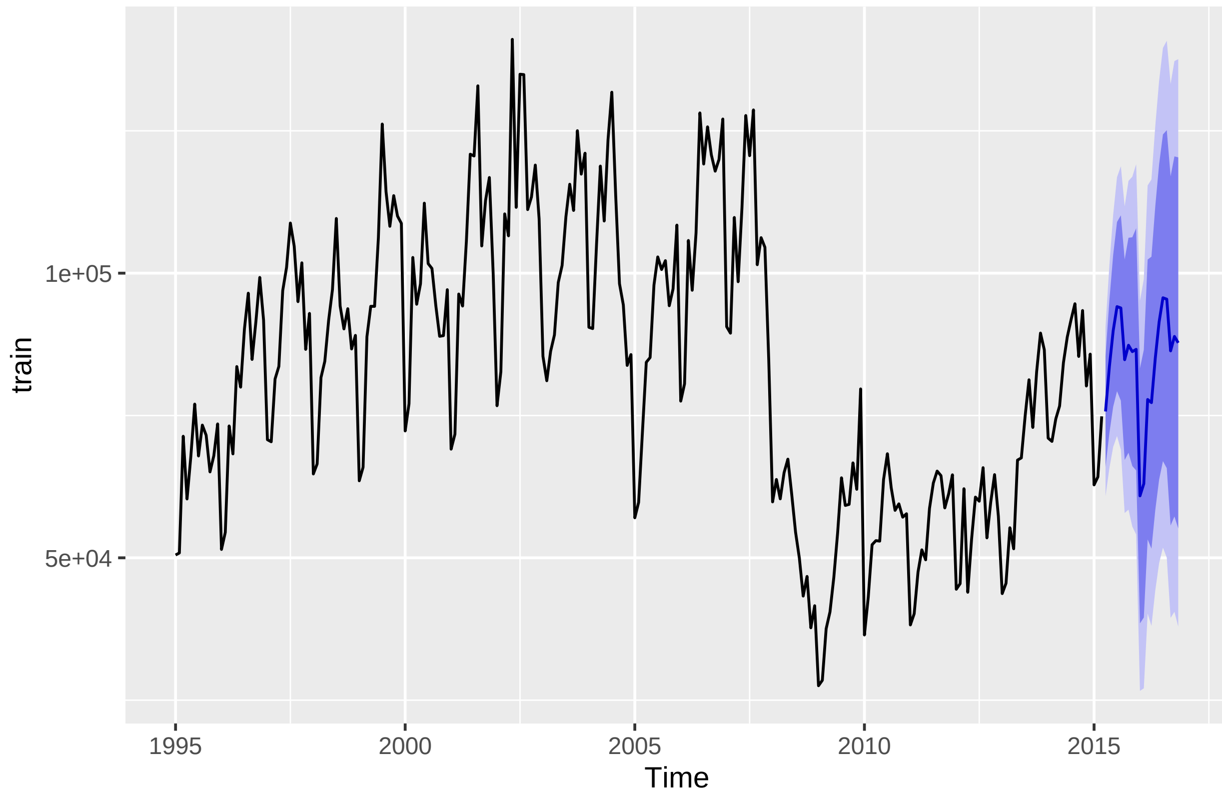


Step 26: Performing *Additive Holt-Winters Method* forecast:

```
fchw <- hw(train, h = 20)
```

Step 27: Plotting Additive Holt-Winters forecast:

```
autoplot(train) + autolayer(fchw)
```

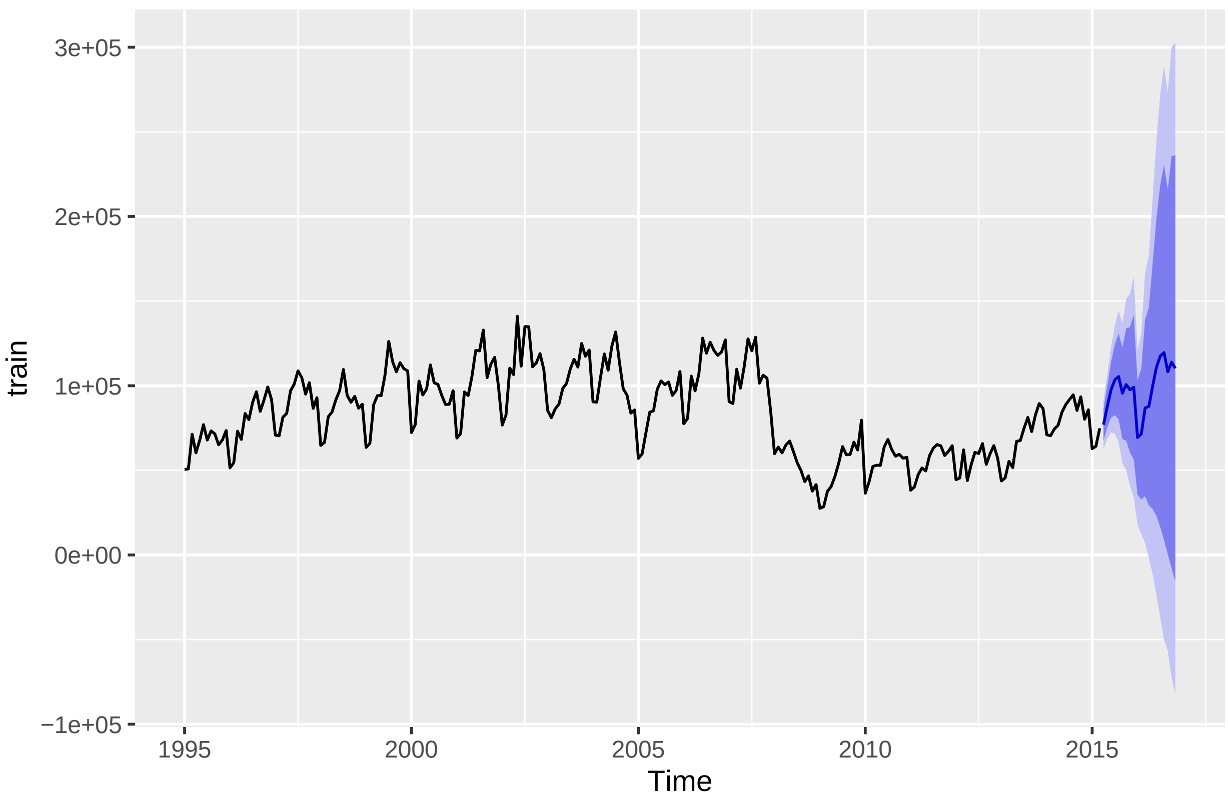


Step 28: Performing *Multiplicative Holt-Winters Method* forecast:

```
fchwm <- hw(train, h = 20, seasonal = "multiplicative")
```

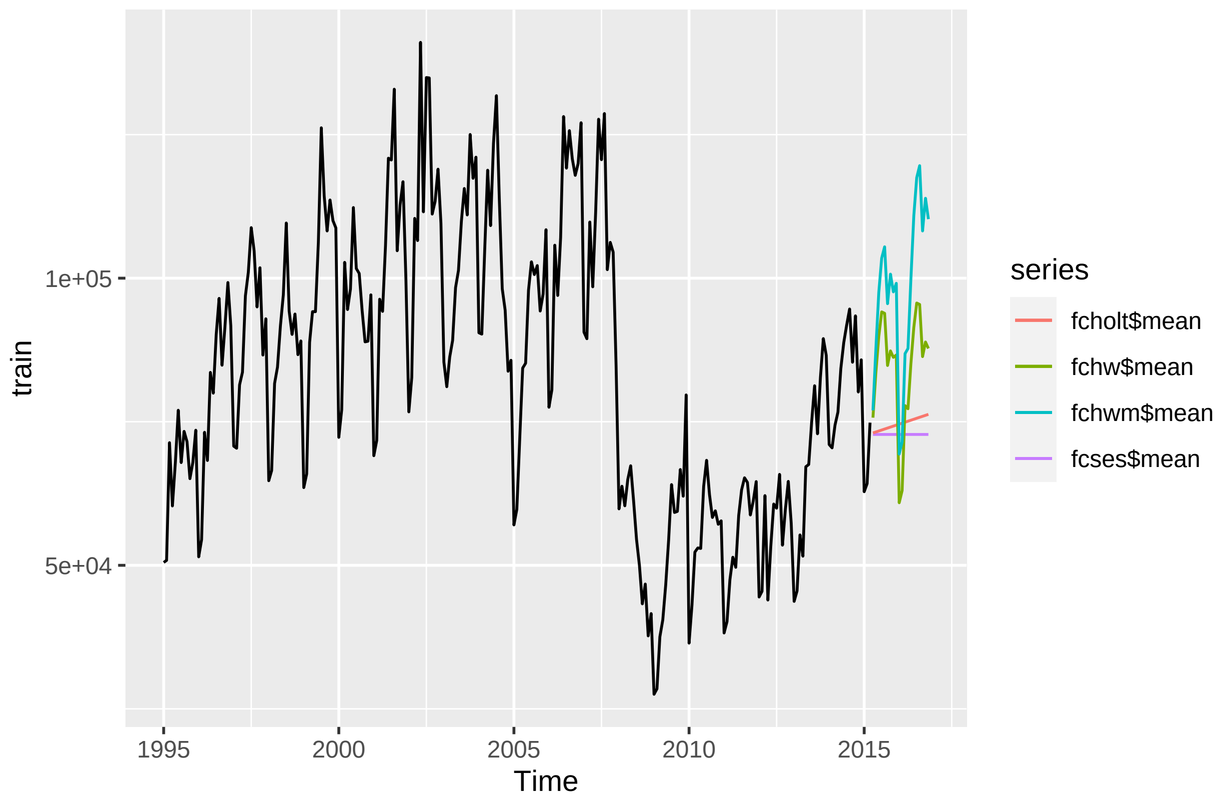
Step 29: Plotting Multiplicative Holt-Winters forecast:

```
autoplot(train) + autolayer(fchwm)
```



Step 30: Plotting all 4 methods under Exponential Smoothing together

```
autoplot(train) + autolayer(fcses$mean) + autolayer(fcholt$mean) + autolayer(fchw$mean) + autolayer(fchw
```



Step 31: Generating the performance metrics of all 4 methods under Exponential Smoothing deployed above:

```
# Simple Exponential Smoothing
accuracy(fcses,test)
```

```
##                      ME     RMSE       MAE       MPE     MAPE     MASE
## Training set  111.0939 12097.98  9089.867 -1.328005 11.79913 0.603143
## Test set     9646.2802 23158.37 16986.552  4.000072 22.31032 1.127114
##                    ACF1 Theil's U
## Training set 0.01241591        NA
## Test set     0.03491632  1.034367
```

```
# Holt Linear Method
accuracy(fcholt,test)
```

```
##                      ME     RMSE       MAE       MPE     MAPE      MASE
## Training set -952.4854 12192.96  9104.988 -2.555360 11.93878 0.6041464
## Test set     7766.0084 22824.48 16501.590  1.293776 22.32585 1.0949356
##                     ACF1 Theil's U
## Training set 0.005696305        NA
## Test set     0.063311362  1.028222
```

```
# Additive Holt-Winters Method
accuracy(fchw,test)
```

```
##                      ME      RMSE      MAE         MPE      MAPE      MASE
## Training set  -429.2029  7336.257  5464.28  -0.7067046  6.997023 0.3625732
## Test set     -2151.6235 22044.775 14873.95 -11.2135313 22.930014 0.9869363
##                     ACF1 Theil's U
## Training set -0.07699233        NA
## Test set      0.26275944   1.06068
```

```
# Multiplicative Holt-Winters Method
accuracy(fchwm,test)
```

```
##                        ME      RMSE       MAE          MPE      MAPE      MASE
## Training set    -67.77009  7482.735  5672.104  -0.04867551  7.124933 0.3763631
## Test set     -15559.99699 30350.868 21554.289 -30.01165678 34.463217 1.4301991
##                     ACF1 Theil's U
## Training set -0.02007707        NA
## Test set      0.43168049   1.45967
```

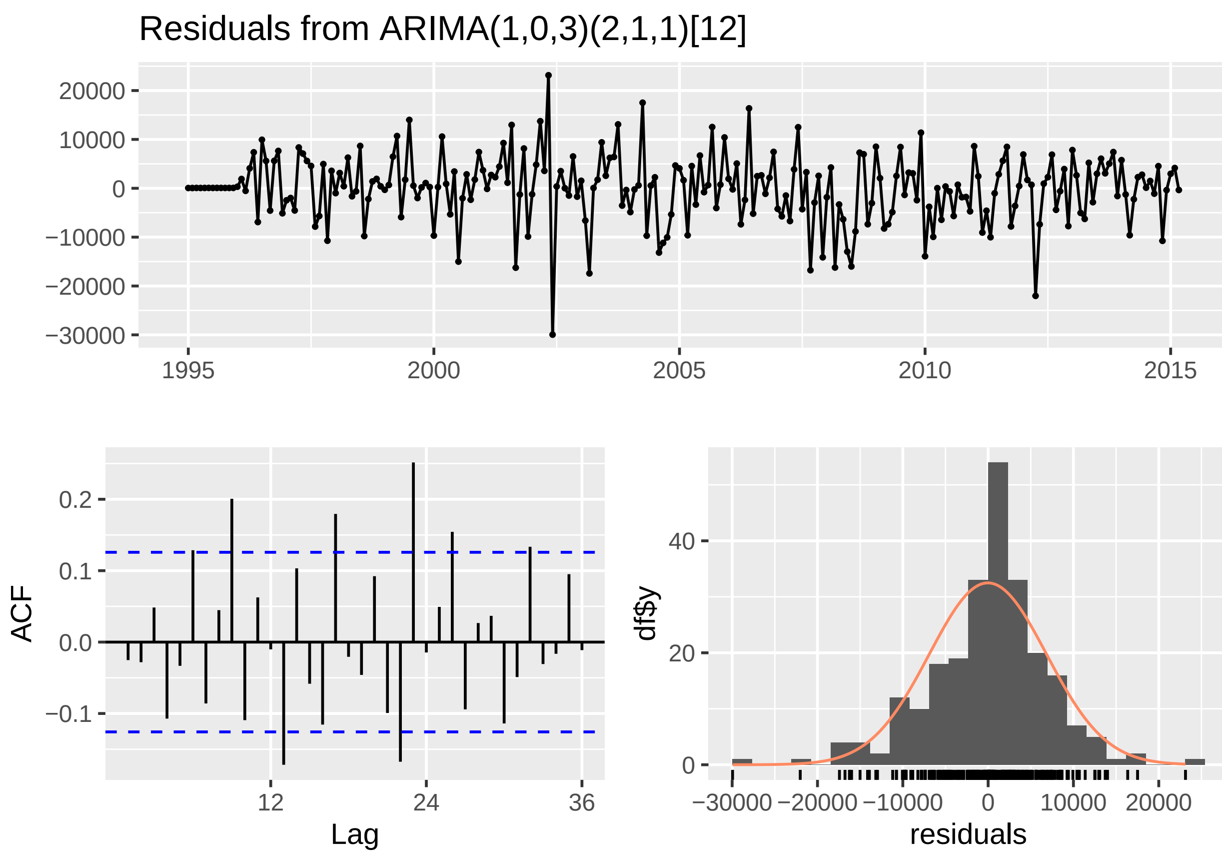
Step 32: Performing Auto ARIMA model for forecasting

```
fcauto <- auto.arima(train)
fcauto
```

```
## Series: train
## ARIMA(1,0,3)(2,1,1)[12]
##
## Coefficients:
##          ar1      ma1     ma2     ma3    sar1     sar2     sma1
##       0.9359  -0.3693  0.2552  0.2734  0.0797  -0.1004  -0.8845
## s.e.  0.0280   0.0735  0.0730  0.0750  0.0907   0.0820   0.0839
##
## sigma^2 = 51258914:  log likelihood = -2385.05
## AIC=4786.1   AICc=4786.75   BIC=4813.64
```

Step 33: Performing residual diagnostics on Auto ARIMA model

```
checkresiduals(fcauto)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,0,3)(2,1,1)[12]
## Q* = 78.512, df = 17, p-value = 7.039e-10
##
## Model df: 7.   Total lags used: 24
```

The aim of residual diagnostics is to find out whether residuals represent a white noise. If not, then model needs to be modified.
Mainly we look for two items:
1. Residuals are normally distributed
2. ACF (Auto Correlation Plot) is not statistically significant. This means values should either lie in between blue lines or not far from blue lines.
Here, in our example values does not lie far from blue line.Thus, it represents white noise. Though, this is not the best representation of white noise but this model is better than all other models based on performance metric.

Step 34:Generating the performance metric for Auto ARIMA:

```
fcauto_test <- forecast(fcauto, h = 20)
accuracy(fcauto_test,psts)
```

```
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -25.42274  6873.937  5010.642 -0.6259232 6.601257 0.3324728
```

```
## Test set     811.95659 21239.245 13529.334 -6.9973761 20.401478 0.8977165
##                      ACF1 Theil's U
## Training set -0.02517246        NA
## Test set      0.21942317  1.015061
```

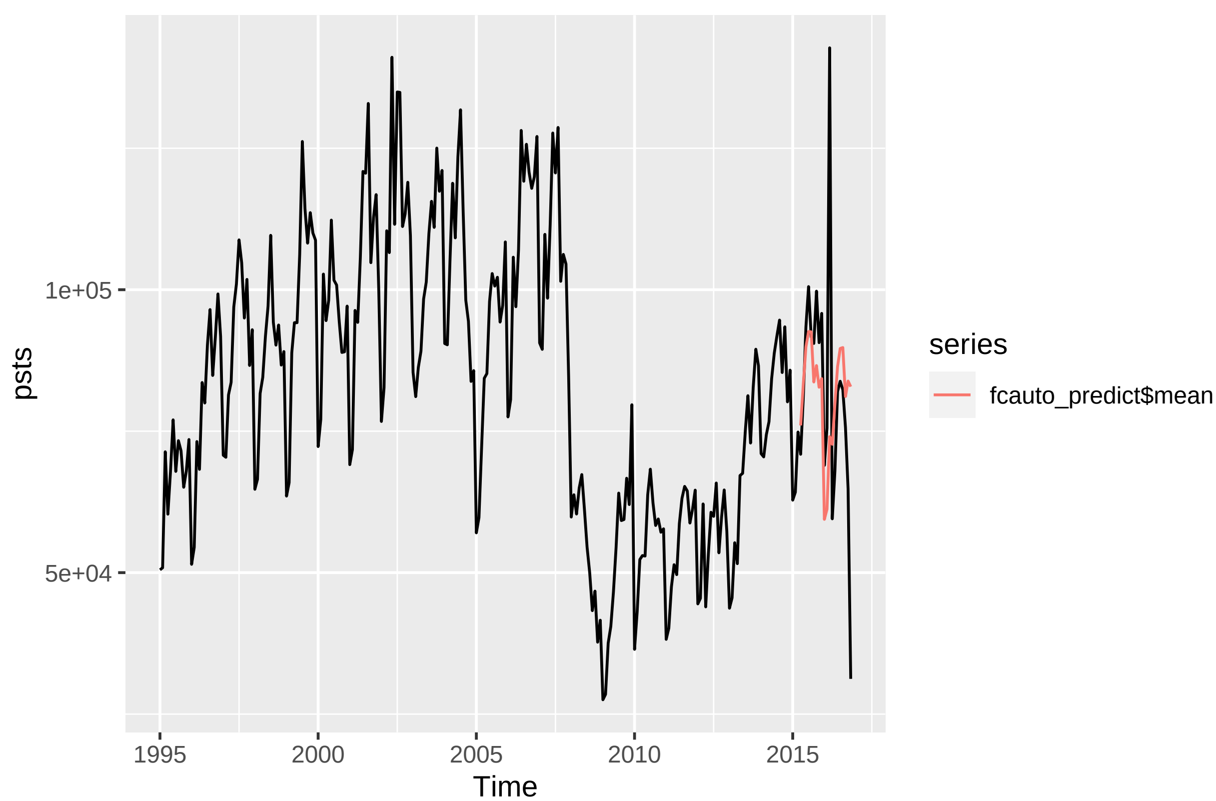
Analysis
In order to find the best method, we have to look at the values of each performance metric. The model having lowest value for each metric is the best one. If there are different models having lowest value than the model with more number of metrics with lowest value is the best.So, reviewing all above performance metrics clearly suggests that Auto ARIMA is the best forecasting model of all.

Step 35: Predicting property sales for next 20 months:

```
fcauto_predict <- forecast(fcauto, h = 20)
```

Step 36: Plotting the forecast of model with actual values

```
autoplot(psts) + autolayer(fcauto_predict$mean)
```



Step 37: Generating forecast for next one year i.e. till Nov 2017

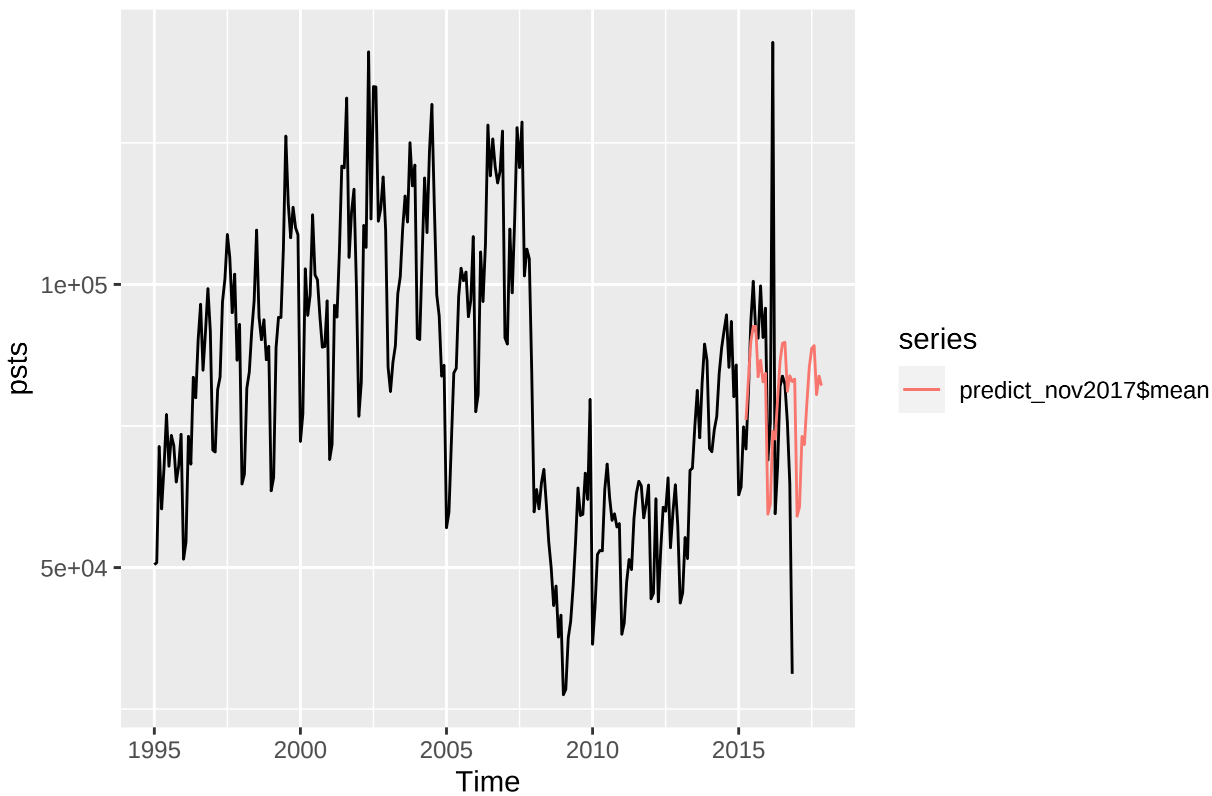
```
predict_nov2017 <- forecast(fcauto, h = 32)
predict_nov2017
```

```
##          Point Forecast    Lo 80     Hi 80    Lo 95     Hi 95
```

```
## Apr 2015       76010.47 66827.06  85193.87 61965.66  90055.28
## May 2015       83243.60 72688.66  93798.54 67101.21  99385.99
## Jun 2015       89883.78 77099.87 102667.70 70332.47 109435.10
## Jul 2015       92608.21 76822.80 108393.63 68466.51 116749.92
## Aug 2015       92553.33 74545.60 110561.05 65012.89 120093.77
## Sep 2015       83691.42 63941.89 103440.95 53487.12 113895.72
## Oct 2015       86597.21 65439.90 107754.52 54239.89 118954.52
## Nov 2015       82765.22 60447.62 105082.82 48633.40 116897.05
## Dec 2015       84299.53 61013.11 107585.95 48686.03 119913.04
## Jan 2016       59415.39 35313.52  83517.26 22554.76  96276.02
## Feb 2016       61204.33 36409.32  85999.34 23283.64  99125.03
## Mar 2016       74038.99 48652.49  99425.49 35213.69 112864.30
## Apr 2016       72674.50 46365.84  98983.16 32438.88 112910.12
## May 2016       79805.02 52865.91 106744.12 38605.20 121004.83
## Jun 2016       86389.84 58815.96 113963.73 44219.22 128560.46
## Jul 2016       89616.19 61397.48 117834.90 46459.40 132772.99
## Aug 2016       89777.57 61006.21 118548.93 45775.57 133779.56
## Sep 2016       81119.01 51872.49 110365.53 36390.32 125847.70
## Oct 2016       83861.58 54205.50 113517.66 38506.52 129216.64
## Nov 2016       82863.67 52853.38 112873.95 36966.90 128760.44
## Dec 2016       83284.29 52967.14 113601.44 36918.21 129650.37
## Jan 2017       59058.39 28476.06  89640.71 12286.75 105830.02
## Feb 2017       60736.72 29922.99  91550.45 13611.19 107862.25
## Mar 2017       73130.28 42115.39 104145.17 25697.10 120563.46
## Apr 2017       71727.74 40505.08 102950.41 23976.80 119478.69
## May 2017       78934.80 47542.97 110326.62 30925.15 126944.45
## Jun 2017       85347.02 53800.56 116893.49 37100.87 133593.18
## Jul 2017       88688.38 56999.71 120377.06 40224.73 137152.03
## Aug 2017       89191.49 57379.01 121003.97 40538.50 137844.48
## Sep 2017       80545.32 48625.06 112465.58 31727.50 129363.14
## Oct 2017       83828.43 51814.38 115842.49 34867.16 132789.71
## Nov 2017       82141.93 50045.87 114237.98 33055.25 131228.61
```

Step 38: Plotting the future forecast:

```
autoplot(psts) + autolayer(predict_nov2017$mean)
```



The data of property prices has been obtained from https://www.doogal.co.uk/PropertySales. One can cross-check the prices from the model with actual prices. It can be noticed that forecast values are quite near the actual values.