Time Series

Assignment:

Apply time series analysis to the sales of new one-family houses, USA, from 1963 through 2018 (https://www.census.gov/construction/nrs/historical_data/index.html) .

First, I prepared the environment, loaded required libraries. In order to facilitate data loading and clean-up, I saved the original file in .xlsx format (instead of an older .xls) and used the opemxlsx package to load the data set:

```
> ### Time Series
> ###Median House Sales Prices
>
> rm(list=ls()) #Clear the environment
> setwd("YOUR_PATH") #Set working directory for the assignment
> getwd() #Check working directory
[1] "YOUR_PATH"
>
> #Load libraries
> library(openxlsx)
> library(stats)
> library(tseries)

    'tseries' version: 0.10-46

    'tseries' is a package for time series analysis and computational finance.

    See 'library(help="tseries")' for details.
> ###Load Data - file was previouesly saved as .xlsx (instead of xls)
> prices <- read.xlsx("pricereg_cust.xlsx", sheet = 2, startRow = 4, colNames = TRUE, rowNames =
FALSE, rows = c(4:228), fillMergedCells = FALSE)
```

I loaded the second spreadsheet from the file and only the rows that contained the price information, but a quick look at the data using head(prices), tail(prices), and str(prices) commands indicated that the data still required some additional cleanup.

```
> #Check data
> head(prices)
  Period     United Northeast Midwest South  West     United Northeast Midwest South West
1   <NA> States 1         NA      NA    NA     NA States 1       <NA>    <NA>  <NA> <NA>
2 1963Q1      17800     20800   17500 16800 18000     19300       (NA)    (NA)  (NA) (NA)
```

```
3 1963Q2      18000      20600    17700 15800 18900      19400       (NA)     (NA)   (NA)  (NA)
4 1963Q3      17900      19600    17800 15900 19000      19200       (NA)     (NA)   (NA)  (NA)
5 1963Q4      18500      20600    19100 15800 19500      19600       (NA)     (NA)   (NA)  (NA)
6 1964Q1      18500      20300    18700 16500 19600      19600       (NA)     (NA)   (NA)  (NA)
> tail(prices)
    Period United Northeast Midwest  South   West United Northeast Midwest  South   West
219 2017Q2 318200    472200  288300 285400 386300 376900    583500  322300 336100 432700
220 2017Q3 320500    445800  278500 295300 385500 373200    536200  316500 334000 451400
221 2017Q4 337900    496500  285600 298500 409700 399700    683200  322800 340400 486800
222 2018Q1 331800    437500  291200 295800 408000 374600    514600  326300 329900 456800
223 2018Q2 315600    453300  277600 285300 423400 378400    576600  316600 327200 483700
224 2018Q3 325200    484900  292100 289400 404300 390200    693900  343000 338700 462000
> #check structure of the df
> str(prices)
'data.frame':      224 obs. of  11 variables:
 $ Period   : chr  NA "1963Q1" "1963Q2" "1963Q3" ...
 $ United   : chr  "States 1 " "17800" "18000" "17900" ...
 $ Northeast: num  NA 20800 20600 19600 20600 20300 19800 20200 21400 21000 ...
 $ Midwest  : num  NA 17500 17700 17800 19100 18700 19800 18900 20800 21900 ...
 $ South    : num  NA 16800 15800 15900 15800 16500 16800 16800 16700 17400 ...
 $ West     : num  NA 18000 18900 19000 19500 19600 20100 20600 21500 21600 ...
 $ United   : chr  "States 1 " "19300" "19400" "19200" ...
 $ Northeast: chr  NA "(NA)" "(NA)" "(NA)" ...
 $ Midwest  : chr  NA "(NA)" "(NA)" "(NA)" ...
 $ South    : chr  NA "(NA)" "(NA)" "(NA)" ...
 $ West     : chr  NA "(NA)" "(NA)" "(NA)" ...
```

So, I deleted the first row from the data frame that was inserted due to merged cells in the original file. Dropped columns containing information for the average sale prices (kept the median prices only), renamed a column and converted a column with median US prices to numeric (it was originally loaded as character) using the following code:

```
> ###Data clean-up
> prices <- prices[-1,] #delete first row
> prices<- prices[,-c(7:11)] #drop columns 7 through 11 (average sale prices)
> names(prices)[2] <- "US" #Rename the second column
> prices$US <- as.numeric(prices$US) #convert US prices to numeric
```

To check the results, I used head(prices), tail(prices), and str(prices) commands again:

```
> #check results
> head(prices)
  Period    US Northeast Midwest South  West
2 1963Q1 17800     20800   17500 16800 18000
3 1963Q2 18000     20600   17700 15800 18900
4 1963Q3 17900     19600   17800 15900 19000
5 1963Q4 18500     20600   19100 15800 19500
6 1964Q1 18500     20300   18700 16500 19600
7 1964Q2 18900     19800   19800 16800 20100
```

```
> tail(prices)
    Period     US Northeast Midwest  South   West
219 2017Q2 318200    472200  288300 285400 386300
220 2017Q3 320500    445800  278500 295300 385500
221 2017Q4 337900    496500  285600 298500 409700
222 2018Q1 331800    437500  291200 295800 408000
223 2018Q2 315600    453300  277600 285300 423400
224 2018Q3 325200    484900  292100 289400 404300
> str(prices)
'data.frame':	223 obs. of  6 variables:
 $ Period   : chr  "1963Q1" "1963Q2" "1963Q3" "1963Q4" ...
 $ US       : num  17800 18000 17900 18500 18500 18900 18900 19400 20200 19800 ...
 $ Northeast: num  20800 20600 19600 20600 20300 19800 20200 21400 21000 21900 ...
 $ Midwest  : num  17500 17700 17800 19100 18700 19800 18900 20800 21900 20800 ...
 $ South    : num  16800 15800 15900 15800 16500 16800 16800 16700 17400 16400 ...
 $ West     : num  18000 18900 19000 19500 19600 20100 20600 21500 21600 22100 ...
```

The above output shows that the data is ready to be converted to a time series object. The dataset contains 223 data points - median quarterly single-family house prices for the US market and its four regions (Northeast, Midwest, South, and West) for the period from the first quarter of 1963 to the third quarter of 2018. For this assignment, I used the median sales prices observed in the South region.

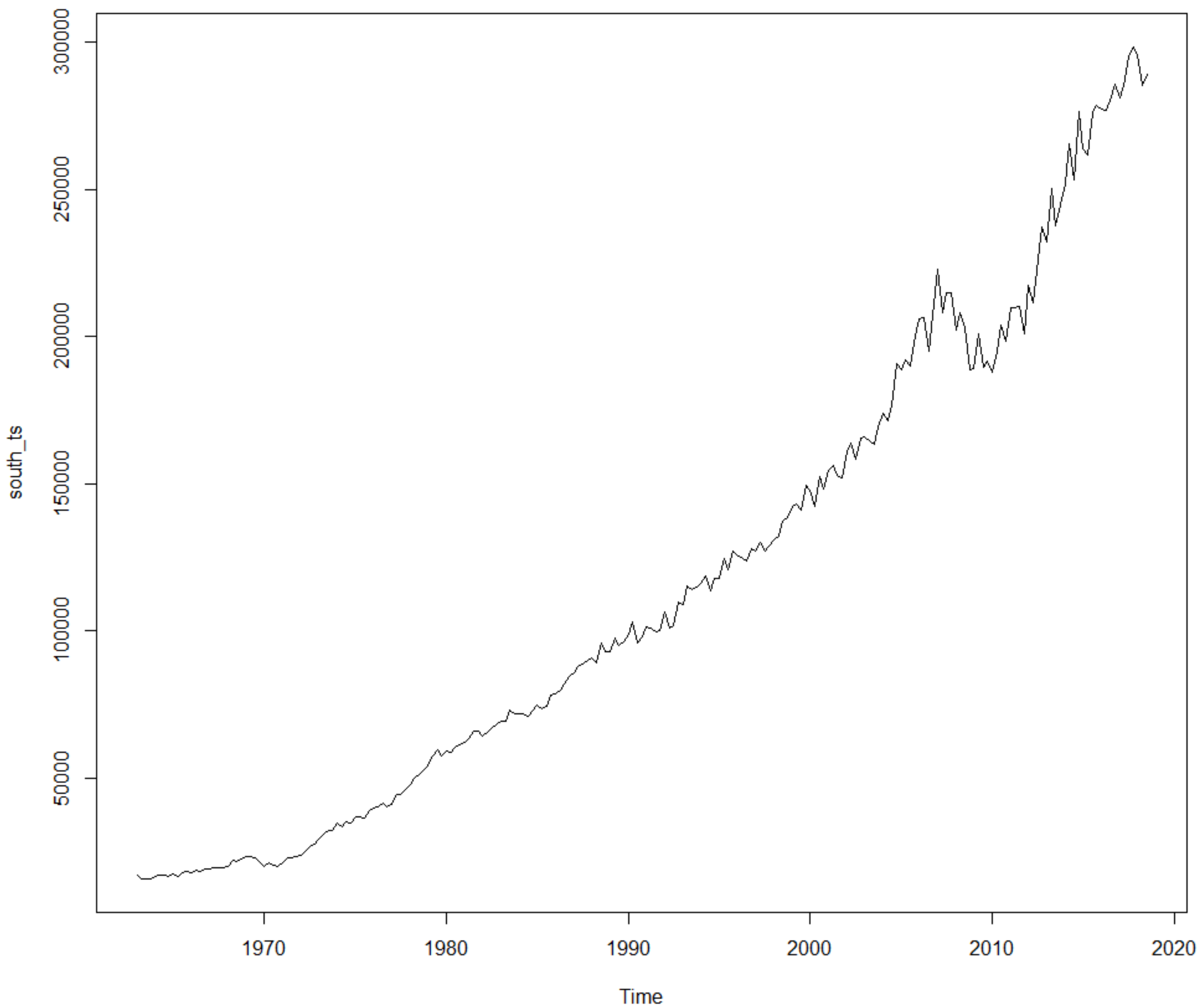
First, I created a time series object and checked its content to look for irregularities:

```
> ###Create time series object for the median prices, South region
> south_ts <- ts(prices[,5], start=c(1963,1), end=c(2018,3), frequency = 4)
> #check content
> south_ts
      Qtr1   Qtr2   Qtr3   Qtr4
1963  16800  15800  15900  15800
1964  16500  16800  16800  16700
1965  17400  16400  17700  18100
1966  17800  18800  18200  18900
1967  18900  19400  19400  19600
1968  19800  22000  21700  22400
1969  23100  23300  23000  21100
1970  20000  21100  20400  20000
1971  21300  22700  22900  23200
1972  23900  25200  26600  27500
1973  29100  30900  32200  32100
1974  34800  33600  35100  34900
1975  36700  36800  36500  39000
1976  39800  40400  41400  40000
1977  41200  44300  44600  46100
1978  47700  49900  51300  53000
1979  54300  57400  59700  57300
1980  59000  58700  60400  61200
1981  62300  63500  66000  66100
```

```
1982  64400  65700  67300  68100
1983  69200  69300  73200  71900
1984  71900  71900  71100  73000
1985  74800  73600  74300  78300
1986  78400  79800  81900  84400
1987  85800  88000  88900  89800
1988  91000  89300  95800  93000
1989  93000  97500  94900  96500
1990  98900 103000  95900  98000
1991 101300 100900  99700 100000
1992 106500 101000 102000 110000
1993 109000 115500 114000 115000
1994 116200 118500 113700 117900
1995 118000 124500 121000 127000
1996 125500 125000 123900 127900
1997 127100 129900 127000 129000
1998 131000 132300 137300 138500
1999 142500 143000 141100 149600
2000 148000 142500 152300 148400
2001 154700 156100 152800 152100
2002 160900 164000 158200 165400
2003 165800 164600 163400 169400
2004 173800 171400 176700 190900
2005 188600 192000 190000 200000
2006 205900 206700 195100 207400
2007 222900 208300 214900 214900
2008 202200 208100 203300 188700
2009 189300 201000 189700 191800
2010 187900 195200 203900 198500
2011 209800 209900 210300 201200
2012 217300 211700 226200 237500
2013 232400 250200 237800 245100
2014 252000 265400 253200 276400
2015 263900 261800 276100 278700
2016 277400 277100 280200 285900
2017 281400 285400 295300 298500
2018 295800 285300 289400
```

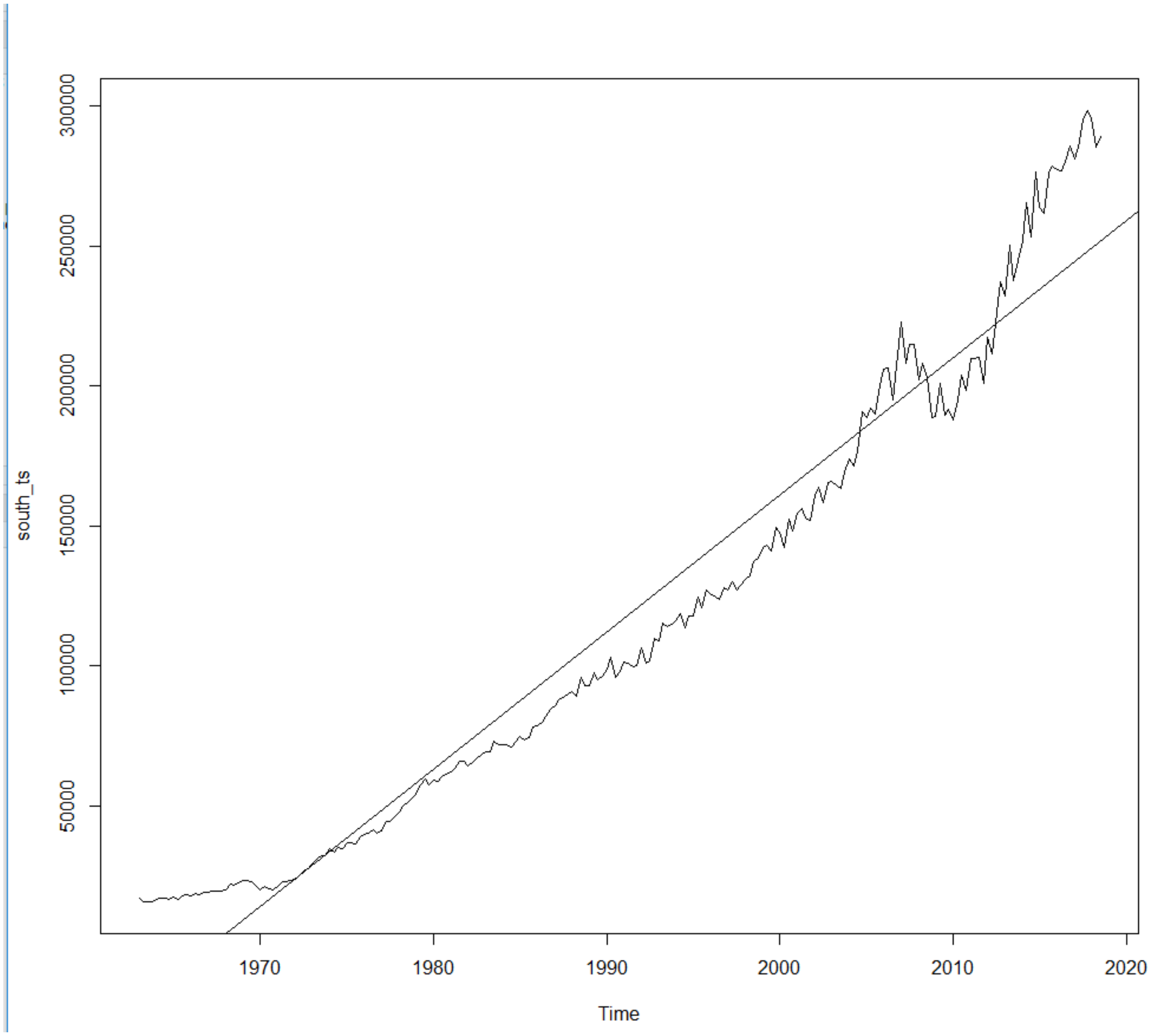
Next, in order to familiarize myself with the time series, I plot.ts() to visualize it:

```
> ##EDA
> #plot time series
> plot.ts(south_ts)
```



In order to proceed with the time series analysis, the data needs to be stationary (approximately the same mean, constant variance, no trends or seasonal fluctuations). The above plot for south_ts demonstrates that these time series are not stationary. First of all, there is an obvious upward trend that is demonstrated with a regression line on the next graph:

```
> #trendline
> abline(reg=lm(south_ts~time(south_ts)))
```



The graph also demonstrates local maximum and minimums for the median sale prices. In the second part of 2007, the prices reached its local maximum before crushing down to their minimum at the beginning of 2010. After that, the median sale prices gradually returned to the previous growth trajectory, however, the variance increased significantly compared to the period before the 2000s.

A formal test for stationary data confirmed this observation.

Hypothesis: Ho: -Data is not stationary;

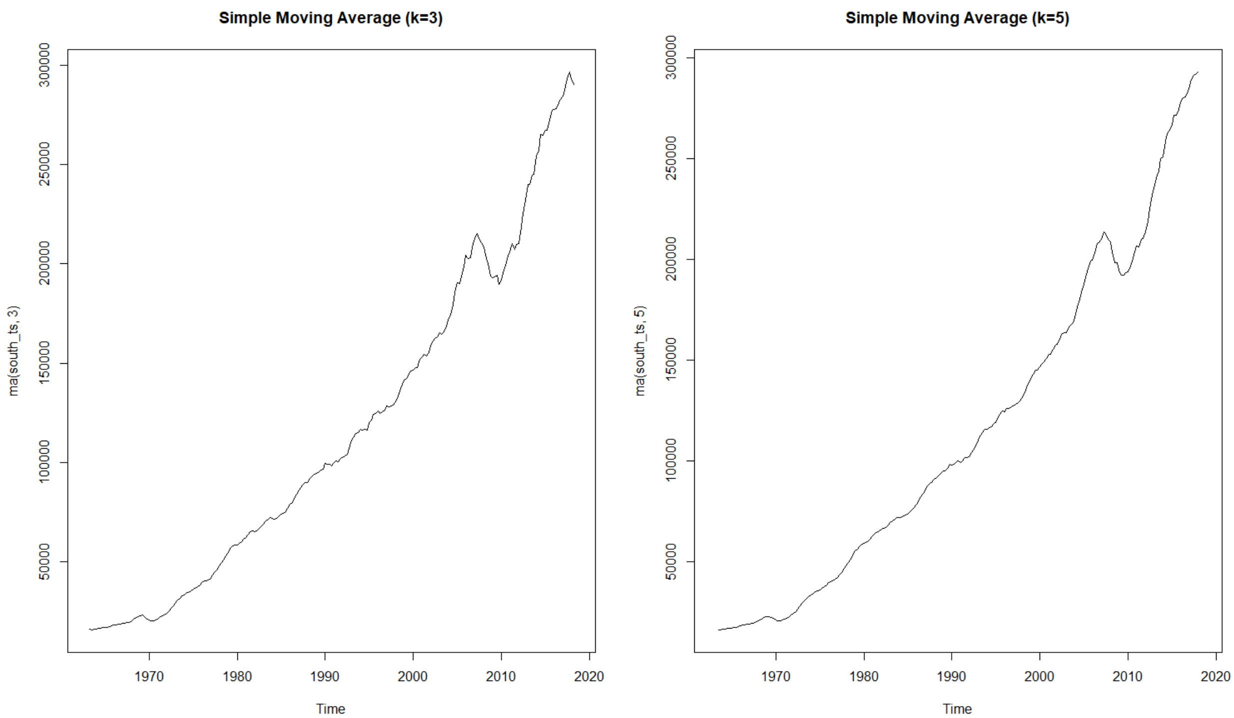
Ha: Data is stationary.

The augmented Dickey-Fuller test returned p-values of 0.6468 which is well above the significance level of 0.05 which provided no evidence allowing us to reject the null hypothesis stating that this data is not stationary.

```
> #test for stationary time series
> adf.test(south_ts, alternative = "stationary")

        Augmented Dickey-Fuller Test

data:  south_ts
Dickey-Fuller = -1.8314, Lag order = 6, p-value = 0.6468
alternative hypothesis: stationary
```

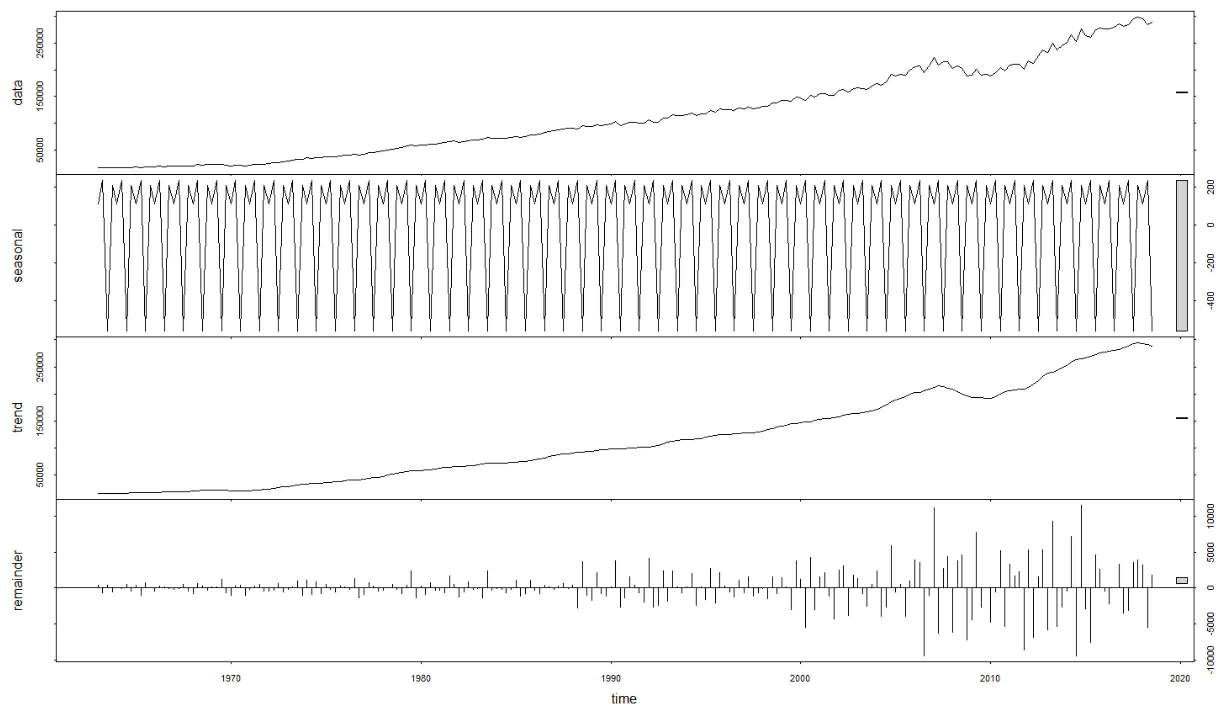
Smoothing with simple moving averages in this case did not help transform data to stationary, it just made the trend line more apparent:

```
> #smoothing using simple moving average ms() from the forecast package
> par(mfrow=c(1,2))
> plot(ma(south_ts, 3), main= "Simple Moving Average (k=3)")
> plot(ma(south_ts, 5), main= "Simple Moving Average (k=5)")
> par(mfrow=c(1,1))
```



In addition to the well-pronounced trend, the time series can potentially include seasonal component. In order to choose an appropriate transformation, we need to decompose the time series into the trend, seasonal and irregular components. I used decomposition by loess smoothing with the stl() function:

```
> #decomposition by loess smoothing
> south_decomposed <- stl(south_ts, s.window = "periodic")
> plot(south_decomposed)
```



The above output shows the initial time series, and extracted seasonal component, the main trend and the remainder component. There is a distinct seasonal pattern, in order to evaluate seasonal influences we check estimated values for the components:

```
> south_decomposed$time.series
         seasonal    trend   remainder
1963 Q1  112.4627 16285.22   402.31698
1963 Q2  235.9598 16190.14  -626.10210
1963 Q3 -557.3817 16109.87   347.50964
1963 Q4  208.9589 16112.02  -520.97987
```

```
1964 Q1  112.4627  16332.61    54.93013
1964 Q2  235.9598  16617.53   -53.48946
1964 Q3 -557.3817  16827.84   529.54497
1964 Q4  208.9589  16891.96  -400.91508
```

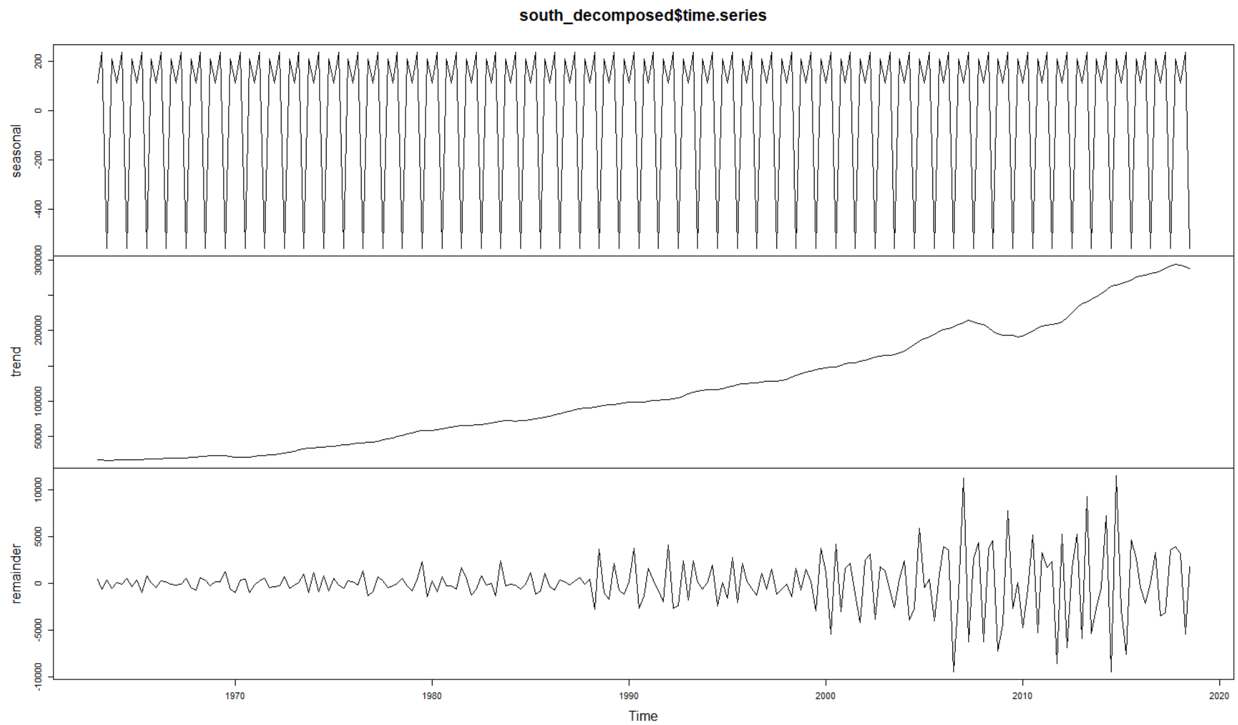
Part of the output omitted …

```
2015 Q1  112.4627 266681.66 -2894.12713
2015 Q2  235.9598 269160.93 -7596.89333
2015 Q3 -557.3817 272033.16  4624.21978
2015 Q4  208.9589 275855.95  2635.08835
2016 Q1  112.4627 277771.64  -484.10268
2016 Q2  235.9598 279016.78 -2152.74219
2016 Q3 -557.3817 280729.98    27.40413
2016 Q4  208.9589 282361.22  3329.82096
2017 Q1  112.4627 284815.14 -3527.59888
2017 Q2  235.9598 288277.92 -3113.87894
2017 Q3 -557.3817 292264.95  3592.42676
2017 Q4  208.9589 294362.76  3928.28059
2018 Q1  112.4627 292431.81  3255.73121
2018 Q2  235.9598 290532.42 -5468.37813
2018 Q3 -557.3817 288192.44  1764.94263
```

The above partial output shows that in the initial units (dollars) every year due to seasonal component home prices decrease by 557.3817 in the third quarter and then start increasing by 208.9589 in the fourth quarter, with the seasonal gains continuing into the first quarter (+112.4627) and the second quarter (+235.9589) of the next year just to drop again the third quarter.

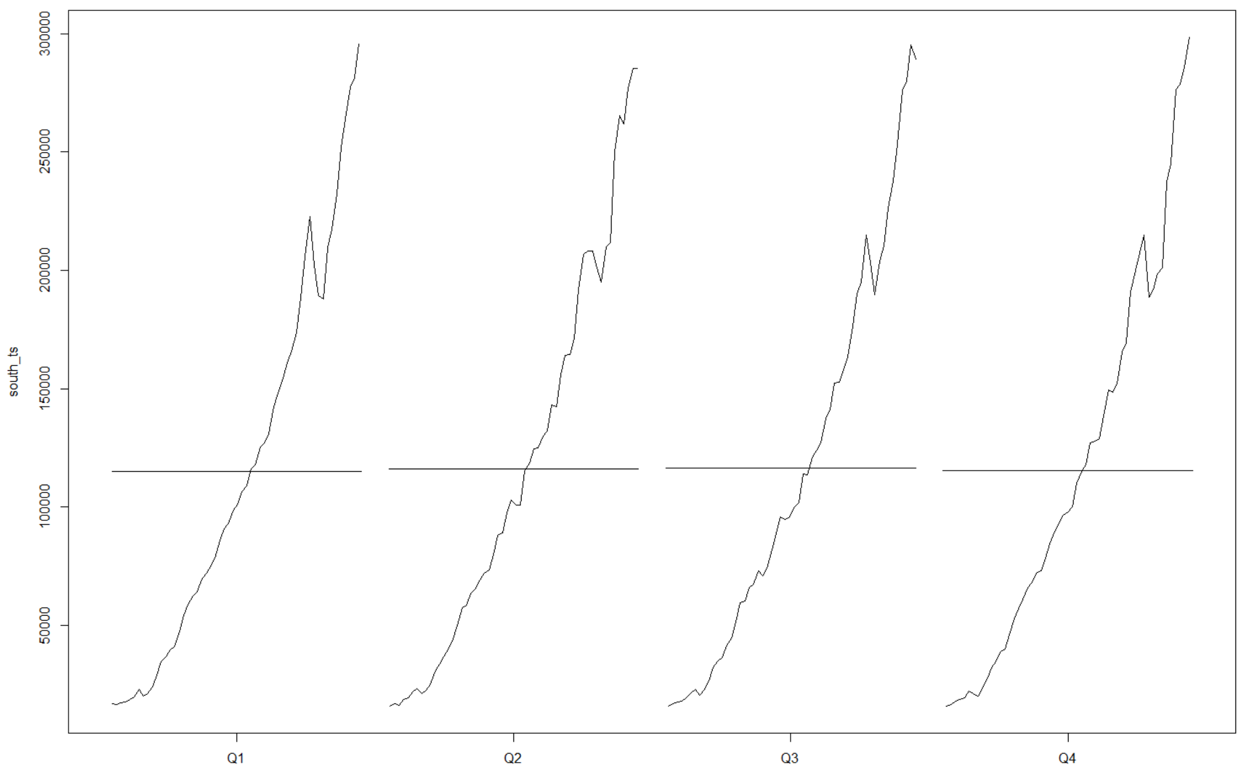
In addition, the plot below shows the increasing variability of the initial time series, since we have not applied the log transformation yet:

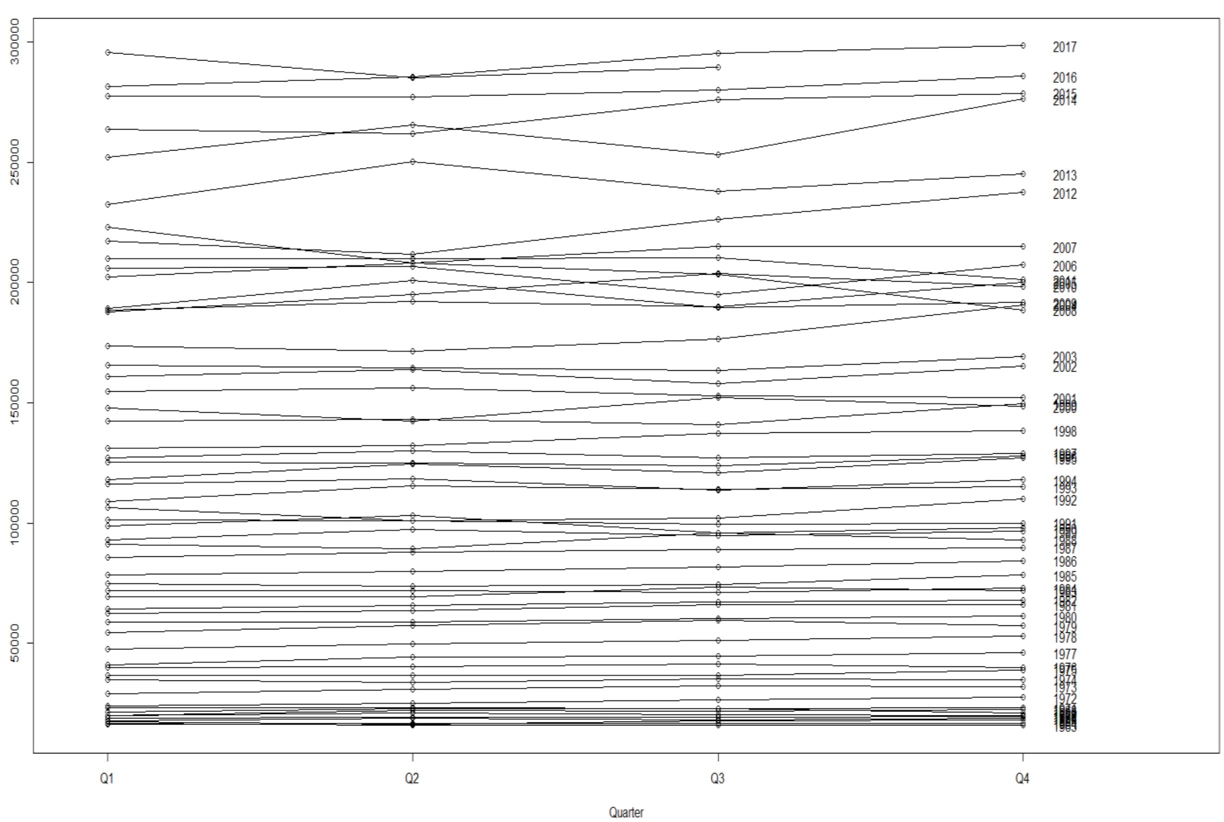
```
> plot(south_decomposed$time.series)
```



Two additional plots are helpful in visualizing the seasonal component – monthplot() and seaonplot() from the forecast package.

```
> plot(south_decomposed$time.series)
> #Visualizing seasonal components
> par(mfrow=c(2,1))
> monthplot(south_ts, xlab=", ylab=")
> seasonplot(south_ts, year.labels="TRUE", main="")
> par(mfrow=c(1,1))
```


south_ts
300000
250000
200000
150000
100000
50000
Q1
Q2
Q3
Q4

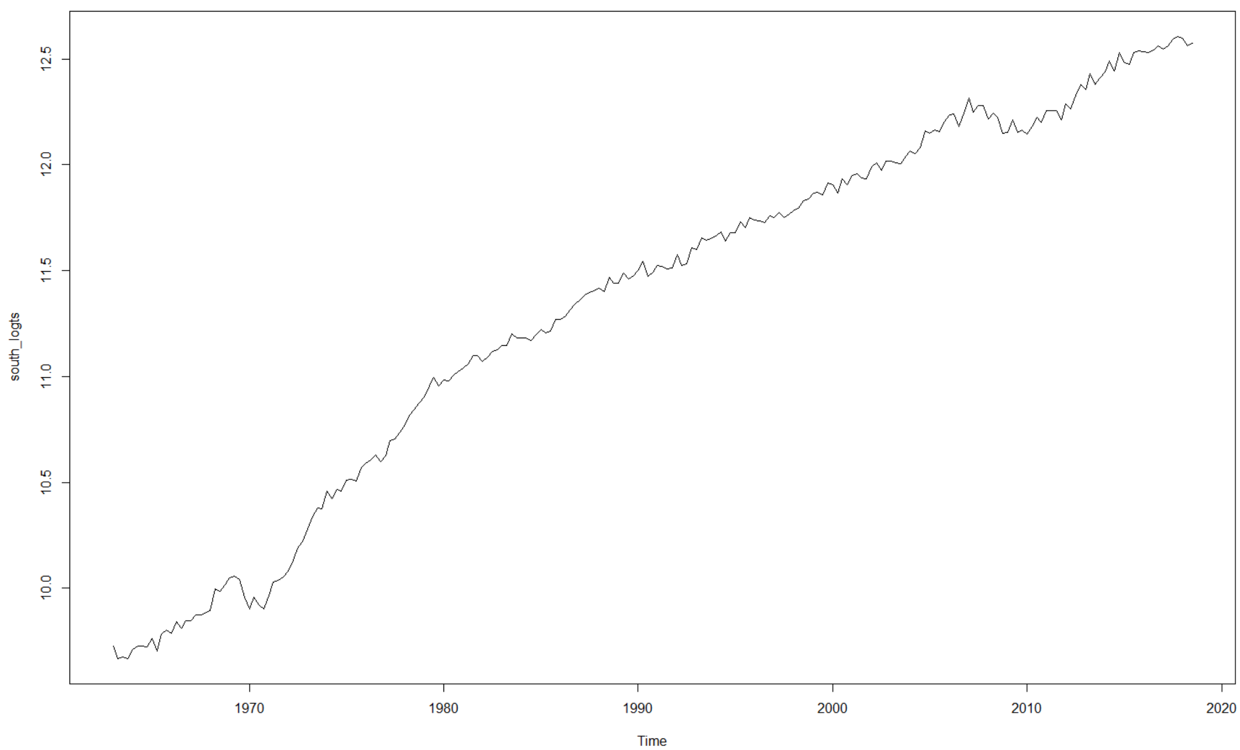

300000
250000
200000
150000
100000
50000
2017
2016
2013
2012
2007
2006
2003
2002
1998
1992
1986
1985
1978
1977
1972
Q1
Q2
Q3
Q4
Quarter

The month plot shows subseries for each quarter, and it appears that the trend increases for each quarter in approximately the same way. The season plot, displaying subseries by year, shows that in most years, the same seasonal pattern was observed. The exceptions mainly coincide with the time frame of the previously mentioned real estate crises.

So, in order to make the time series suitable for forecasting, it is necessary to eliminate the influence of the main trend, decrease variability and eliminate seasonality. I used log-transformation to decrease variance and differencing to eliminate trend and seasonality.

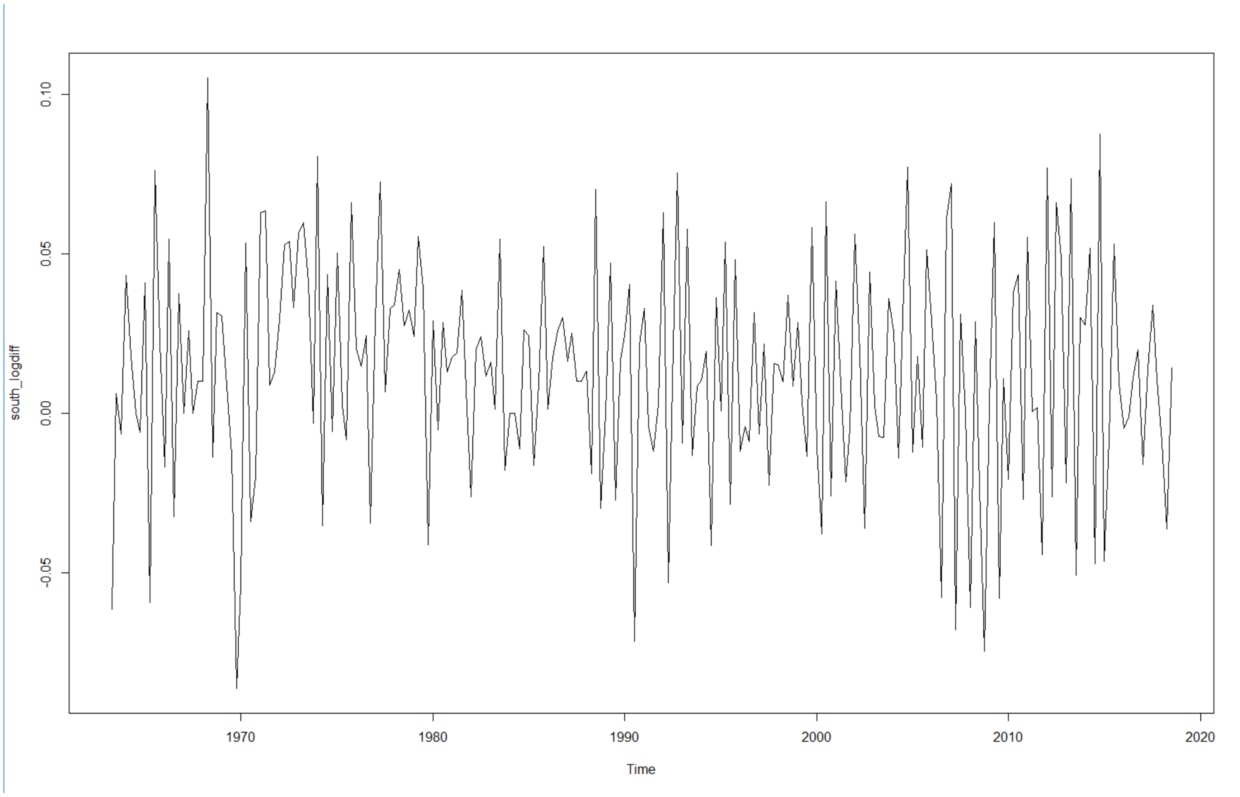
```
> #Log transformation
> south_logts <- log(south_ts) #stabilizging using log function
> plot.ts(south_logts)
```

The log transformation stabilized the time series:



Applying differencing produced a time series that visually appear to be stationary in means and variance, as the level of the series stays roughly constant over time and the variance of the series appears roughly constant over time.

```
> #Differencing
> south_logdiff <- diff(south_logts, difference =1)
> plot.ts(south_logdiff)
```



I used the augmented Dickey-Fuller test to formally test is the data is stationary.

Hypothesis: Ho: -Data is not stationary;

Ha: Data is stationary.

```
> #checking for stationary ts
> adf.test(south_logdiff, alternative = "stationary")

         Augmented Dickey-Fuller Test

data:  south_logdiff
Dickey-Fuller = -5.662, Lag order = 6, p-value = 0.01
```

```
alternative hypothesis: stationary

Warning message:
In adf.test(south_logdiff, alternative = "stationary") :
  p-value smaller than printed p-value
```

The test returned the test statistic of -5.662 and the p-value of less than 0.01, which is lower than the significance level of 0.05.; It means that we can reject the null hypothesis in favor of the alternative hypothesis, stating that the data is in fact stationary.

The next step – model building.

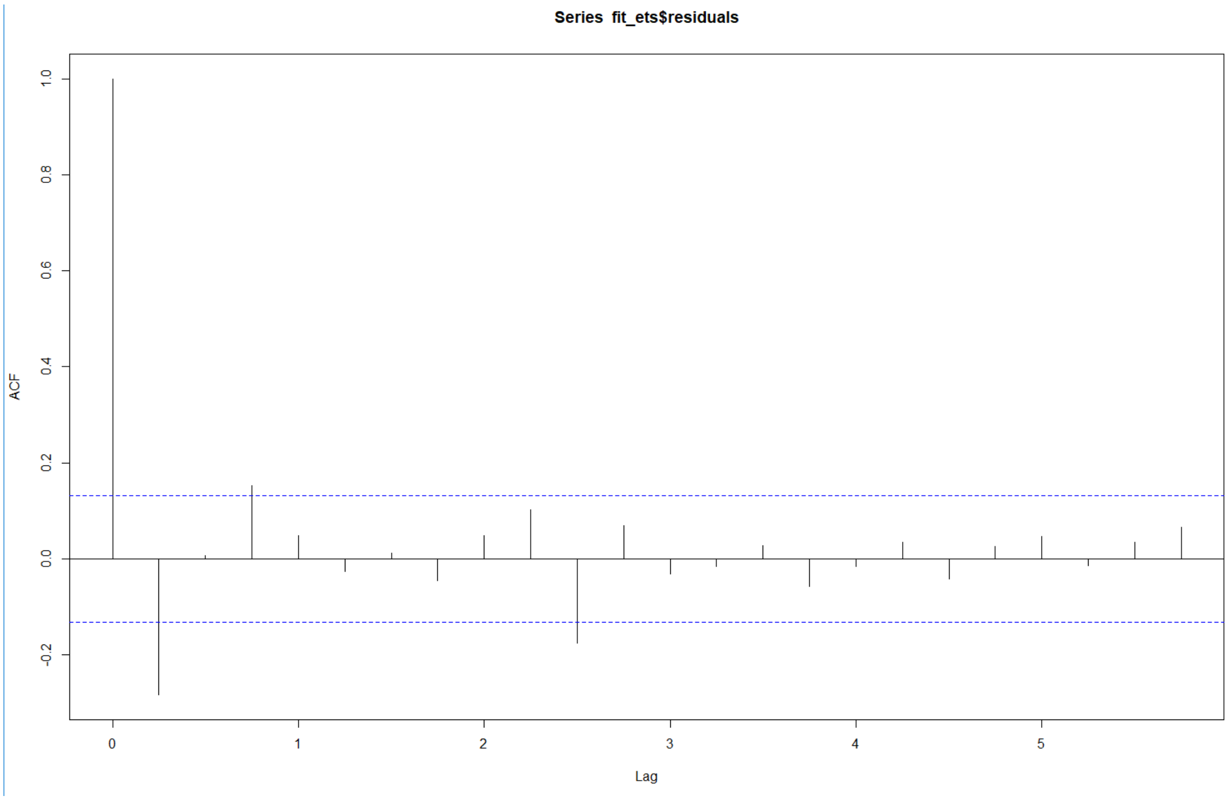
First, I used the simple exponential smoothing model that uses a weighted average of existing time-series values to make predictions. The weights are exponentially decreasing the impact of the older observations on the average. I used ets() command from the forecast package on the south_logdiff series, as it requires stationary data, to automatically fit the model.

```
> fit_ets <- ets(south_logdiff) #autoselect the best fit
> fit_ets
ETS(A,N,N)

Call:
 ets(y = south_logdiff)

  Smoothing parameters:
    alpha = 1e-04

  Initial states:
    l = 0.0128

  sigma:  0.0349

      AIC      AICc       BIC
-286.2652 -286.1551 -276.0572
```

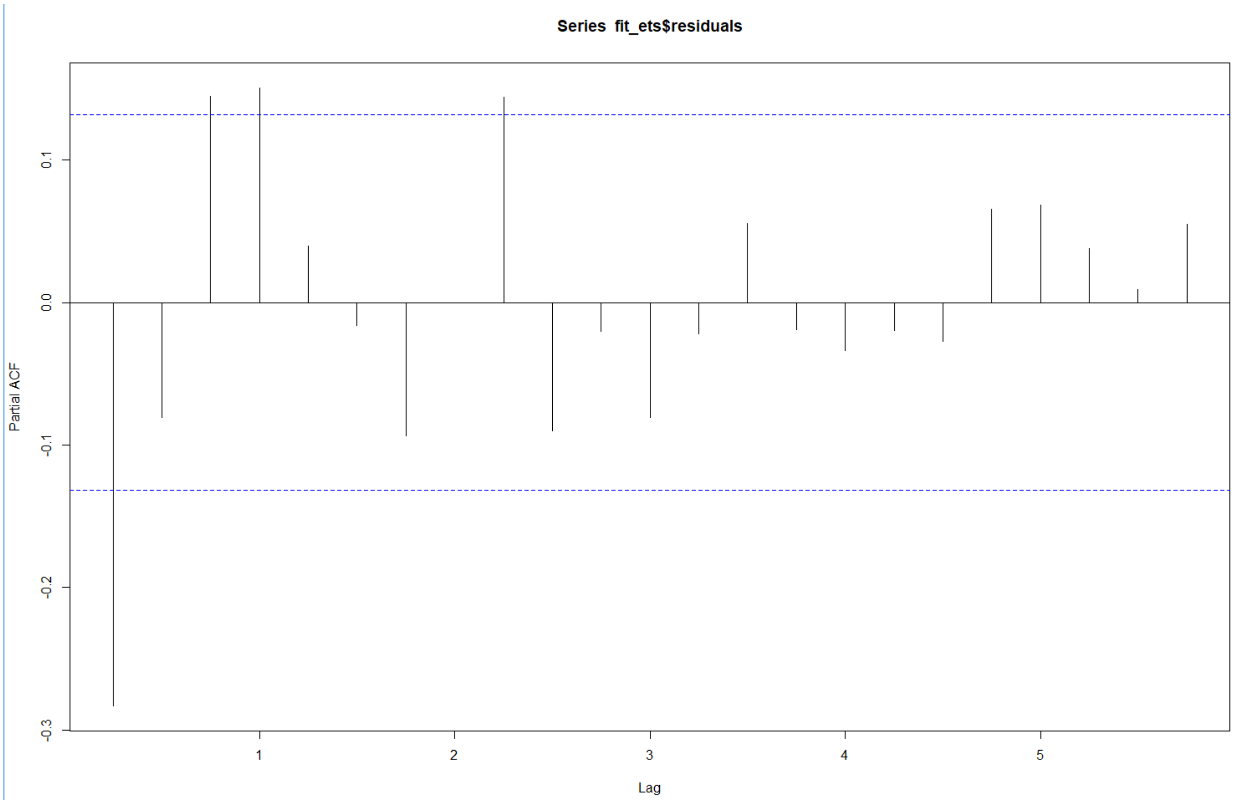
The resulting ETS(A,N,N) model means that the errors are additive, and there is no trend and no seasonal component. The low level of alpha parameter controlling the rate of decay for the weights indicates that older observations play a substantial role in forecasting.

I checked the residuals from the model by plotting autocorrelation and the partial autocorrelation of the residuals:

```
> #check the residuals
> acf(fit_ets$residuals)
```




```
> pacf(fit_ets$residuals)
```



Both graphs show that autocorrelation and partial autocorrelation eventually trail off to zero, but this process is not consistent.

A formal test of independence between the successive errors:

Hypothesis: Ho: Zero autocorrelation (the model does not exhibit a lack of fit);

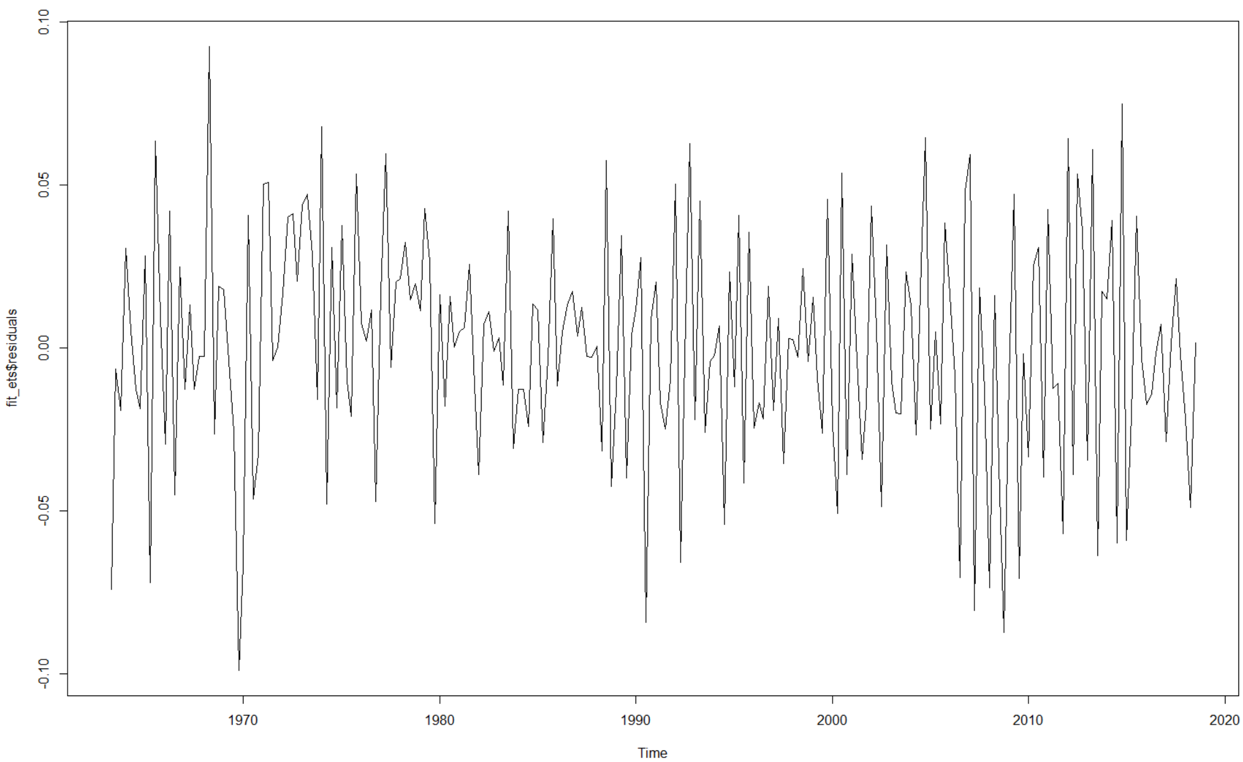
Ha: Non-zero autocorrelation (the model exhibits lack of fit).

```
> Box.test(fit_ets$residuals, type="Ljung-Box") #test independence between successive errors

        Box-Ljung test

data:  fit_ets$residuals
X-squared = 18.028, df = 1, p-value = 2.177e-05
```

The test returned p-values of 2.177e-05 which is considerably lower than the significance level of 0.05, which means that there is enough evidence to reject the null hypothesis in favor of the alternative hypothesis stating that the model needs to be improved as there is autocorrelation between the residuals.

Plotting residuals also suggests that while mean is close to zero, the variance of the residuals is not constant.

```
> plot.ts(fit_ets$residuals)
```



These results suggest that the simple exponential smoothing method does not provide an adequate predictive model for the median home prices.

Next candidate model is the Holt-Winters exponential smoothing model.

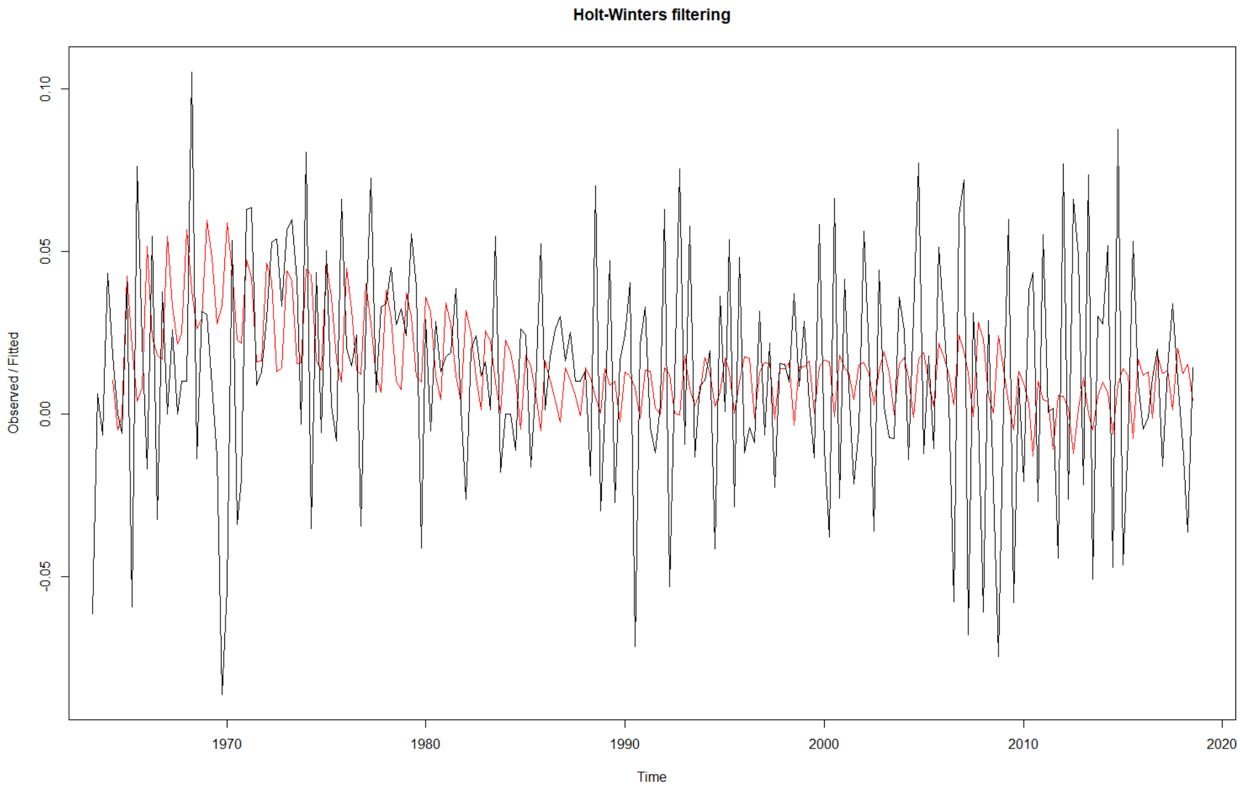
Using HoltWinters() command to automatically select the parameters resulted in the following model:

```
> fit_hw <- HoltWinters(south_logdiff) #automatically select parameters
```

```
> fit_hw
Holt-Winters exponential smoothing with trend and additive seasonal component.

Call:
HoltWinters(x = south_logdiff)

Smoothing parameters:
 alpha: 0.01152048
 beta : 0.4461007
 gamma: 0.06587624

Coefficients:
            [,1]
a    0.0188216413
b    0.0001621310
s1   0.0009564722
s2  -0.0080478505
s3  -0.0071512620
s4  -0.0137169486
```

The alpha parameter, which controls exponential decay for the level, is 0.01152048. It means that old observations have relatively high weights. The beta parameter controlling exponential decay for the slope 0.4464007 means that more newer observations are included in determining the slope. The gamma smoothing parameter, which controls exponential decay of the seasonal component, is 0.06587624. Again, it signifies a relatively high importance of older observations in the model.   The model is applied to the logged time series, so an additive model could be fit.

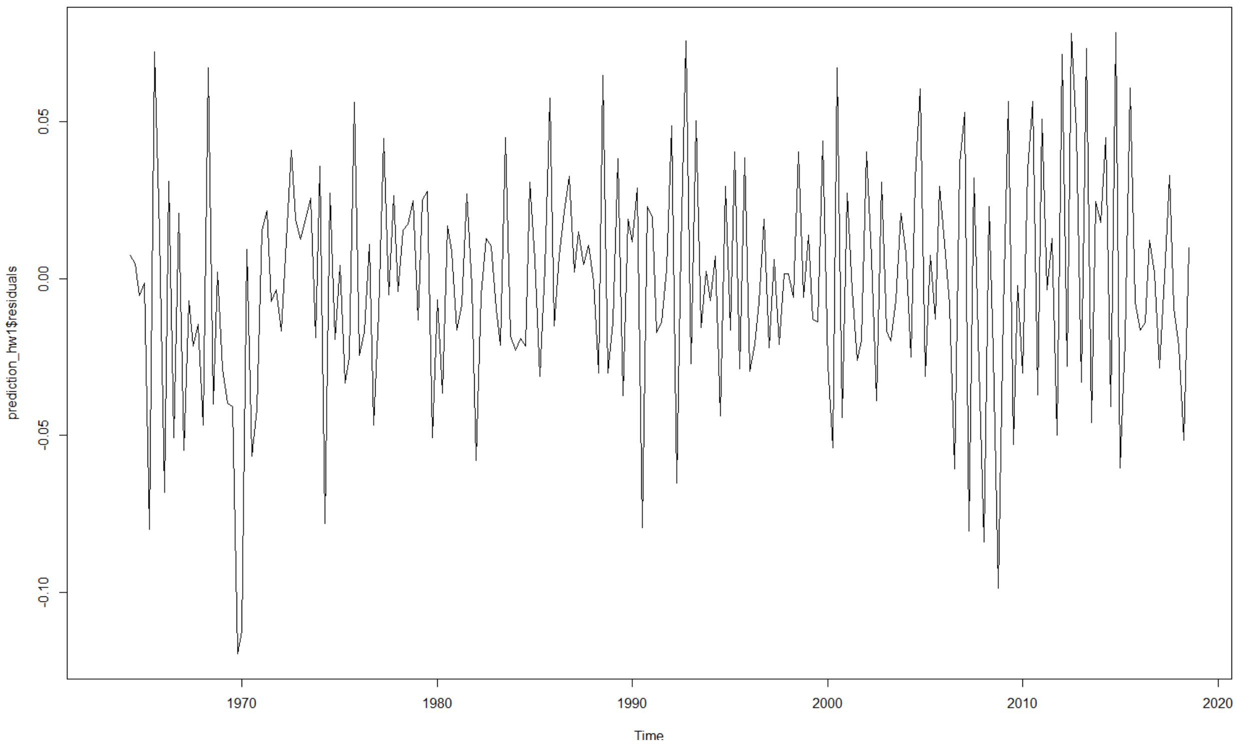
```
> #plot observed and fit
> plot(fit_hw)
```



Plotting observed and fit values on the same graph shows that while overall the fit values followed the observed values, in some periods there was a significance difference (e.g., 1970).

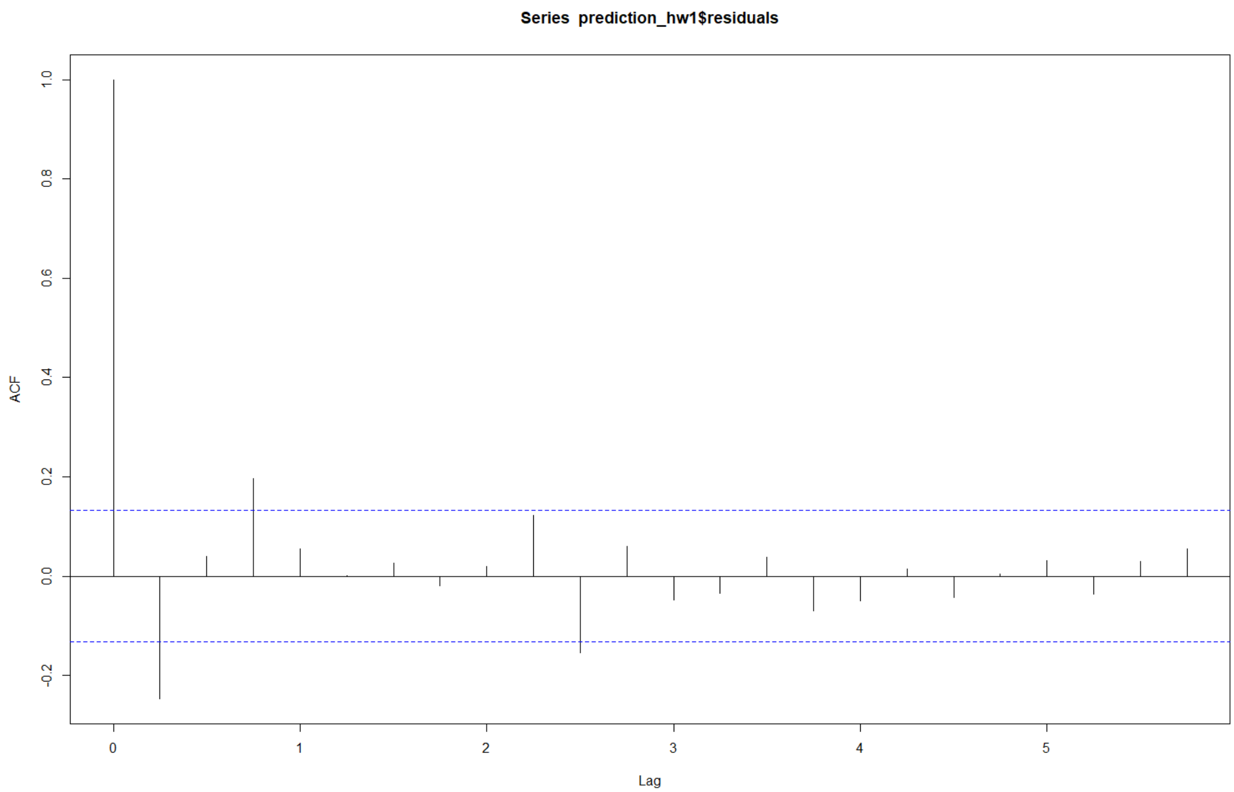
Plotting residuals did not show any significant sources of concern, as the means seems to be close to zero and the variance seems to be consistent.

```
> plot.ts(prediction_hw1$residuals)
```

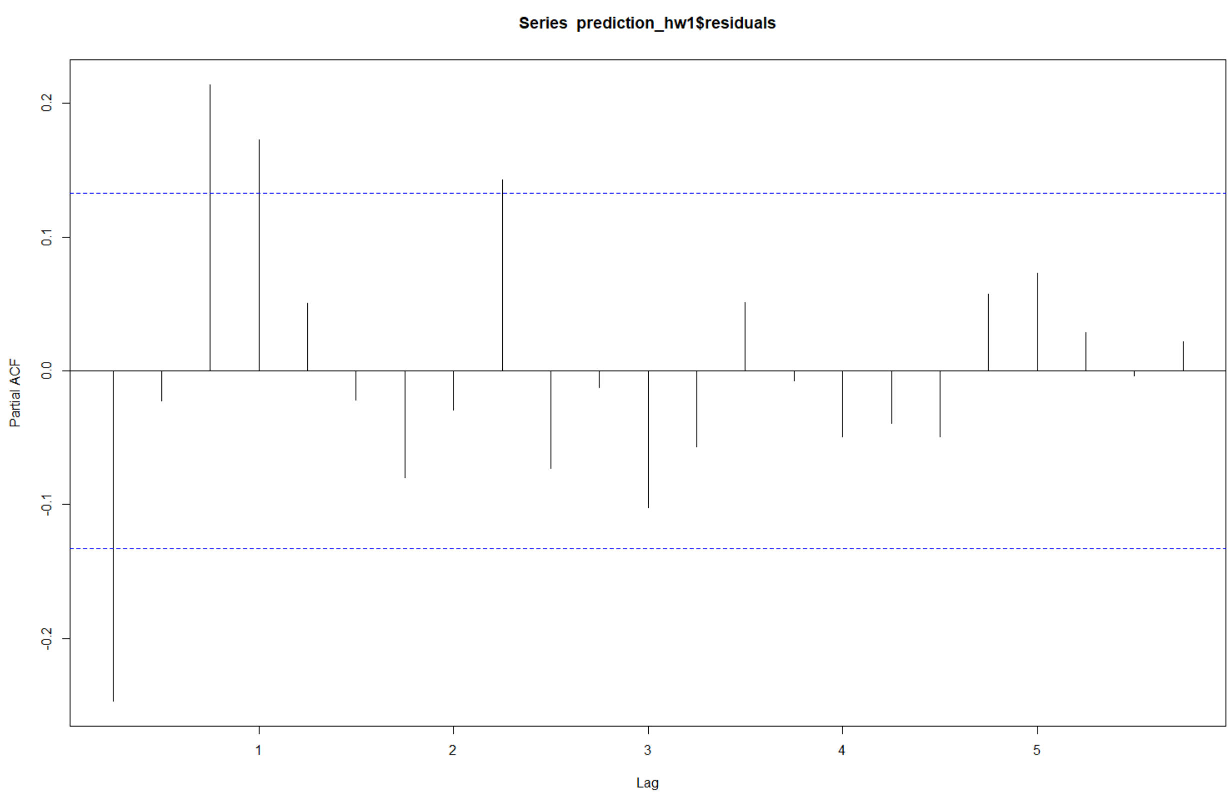


I checked the residuals by plotting the autocorrelation and partial autocorrelation of the residuals.

```
> prediction_hw1 <- forecast(fit_hw)
> acf(prediction_hw1$residuals, na.action = na.omit)
```




```
> pacf(prediction_hw1$residuals, na.action = na.omit)
```



Both autocorrelation and partial autocorrelation of the residuals eventually get close to zero, but this process is not consistent. So, in order to make sure that the successive residuals are independent, I used the Box-Ljung test.

Hypothesis: Ho: Zero autocorrelation (the model does not exhibit lack of fit);

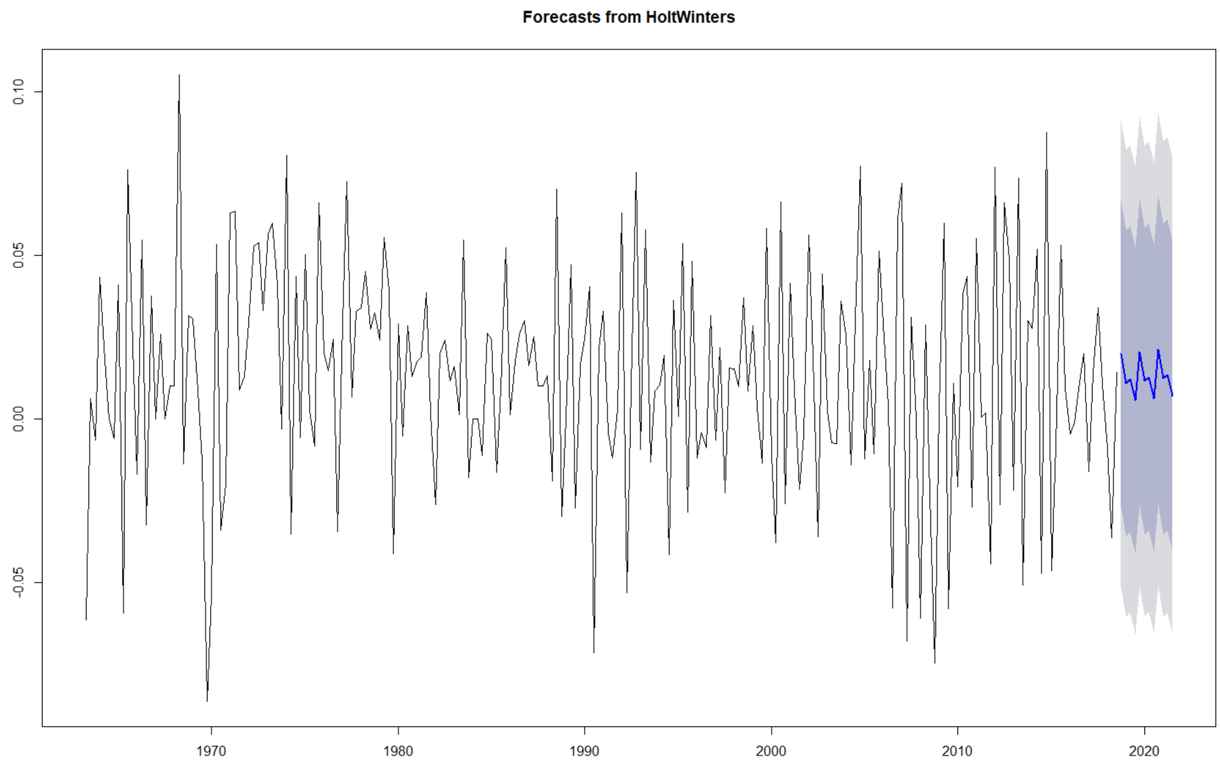
Ha: Non-zero autocorrelation (the model exhibits lack of fit).

```
> Box.test(prediction_hw1$residuals, type="Ljung-Box") #test independence between successive erro
rs

	Box-Ljung test

data:  prediction_hw1$residuals
X-squared = 13.492, df = 1, p-value = 0.0002396
```

The above output returned the p-value of 0.002396, which is lower than the significance level of 0.05. It means that we do not have enough evidence to reject the null hypothesis stating that the autocorrelation between the residuals is equal to zero and the model does not exhibit a lack of fit. It is suitable for forecasting.

Next step is to use the Holt-Winters model for forecasting.

```
> ##Use Holt-Winters model for forecast
> prediction_hw <- forecast(fit_hw, h=12) #forecast values of the next three years
> prediction_hw #predicated values in log thousands
        Point Forecast          Lo 80      Hi 80       Lo 95      Hi 95
2018 Q4    0.019940245 -0.02674338 0.06662386 -0.05145619 0.09133668
2019 Q1    0.011098053 -0.03559205 0.05778815 -0.06030829 0.08250440
2019 Q2    0.012156772 -0.03454442 0.05885796 -0.05926653 0.08358008
2019 Q3    0.005753217 -0.04096490 0.05247133 -0.06569598 0.07720241
2019 Q4    0.020588769 -0.02634917 0.06752671 -0.05119662 0.09237416
2020 Q1    0.011746577 -0.03522351 0.05871667 -0.06008797 0.08358113
2020 Q2    0.012805297 -0.03420639 0.05981699 -0.05909288 0.08470347
2020 Q3    0.006401741 -0.04066221 0.05346569 -0.06557636 0.07837984
2020 Q4    0.021237293 -0.02614660 0.06862119 -0.05123012 0.09370470
2021 Q1    0.012395101 -0.03506549 0.05985569 -0.06018961 0.08497981
2021 Q2    0.013453821 -0.03409756 0.06100520 -0.05926974 0.08617738
2021 Q3    0.007050265 -0.04060713 0.05470766 -0.06583542 0.07993595
> plot(prediction_hw)
```



The above output shows predictions for the next 3 years (12 quarter periods) as applied to the logged differenced data. So, after, converting it back into the initial units (thousands of dollars) the predicted changes in mean prices for each quarter compared to the previous quarter are as follow:

```
> #converting prediction back into the initial units
> low_hw<- exp(prediction_hw$lower)
> upper_hw <- exp(prediction_hw$upper)
> mean_hw <- exp(prediction_hw$mean)
> results_hw <- cbind(mean_hw, low_hw, upper_hw)
> dimnames (results_hw)[[2]] <- c("mean", "Lo 80", "Lo 95", "Hi 80", "Hi 95")
> results_hw
            mean     Lo 80     Lo 95    Hi 80    Hi 95
2018 Q4 1.020140 0.9736111 0.9498453 1.068893 1.095638
2019 Q1 1.011160 0.9650339 0.9414742 1.059491 1.086003
2019 Q2 1.012231 0.9660454 0.9424555 1.060625 1.087172
2019 Q3 1.005770 0.9598628 0.9364155 1.053872 1.080261
2019 Q4 1.020802 0.9739949 0.9500918 1.069859 1.096775
2020 Q1 1.011816 0.9653896 0.9416817 1.060475 1.087173
2020 Q2 1.012888 0.9663720 0.9426192 1.061642 1.088394
2020 Q3 1.006422 0.9601534 0.9365275 1.054921 1.081533
2020 Q4 1.021464 0.9741923 0.9500600 1.071030 1.098235
```

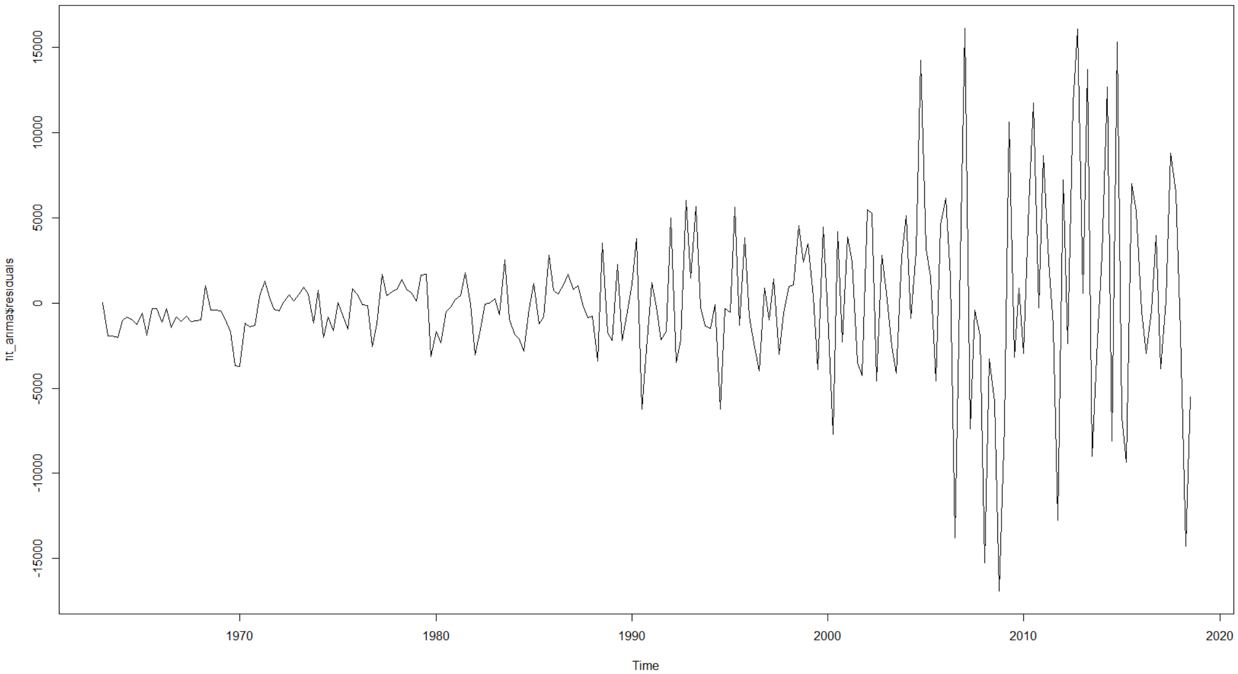
```
2021 Q1 1.012472 0.9655422 0.9415860 1.061683 1.088695
2021 Q2 1.013545 0.9664772 0.9424525 1.062904 1.090000
2021 Q3 1.007075 0.9602063 0.9362849 1.056232 1.083218
```

The next model to explore is ARIMA (autoregressive integrated moving average). Previously, differencing the initial time series once helped to transform it to a stationary time series. So, I specified d=1 and used auto.arima() command to automatically find the p and q parameters.

```
> #ARIMA model autofit
> #differenecing 1 time so for ARIMA (p,d,q) model d=1
> fit_arima <- auto.arima(south_ts, d=1) #autofit arima model to the initial ts
> fit_arima
Series: south_ts
ARIMA(1,1,2)(0,0,1)[4] with drift

Coefficients:
         ar1      ma1     ma2     sma1     drift
      0.7750  -1.2898  0.5253  -0.0043  1197.017
s.e.  0.1065   0.1085  0.0675   0.0783   318.208

sigma^2 estimated as 21567512:  log likelihood=-2187.15
AIC=4386.31   AICc=4386.7   BIC=4406.72
```

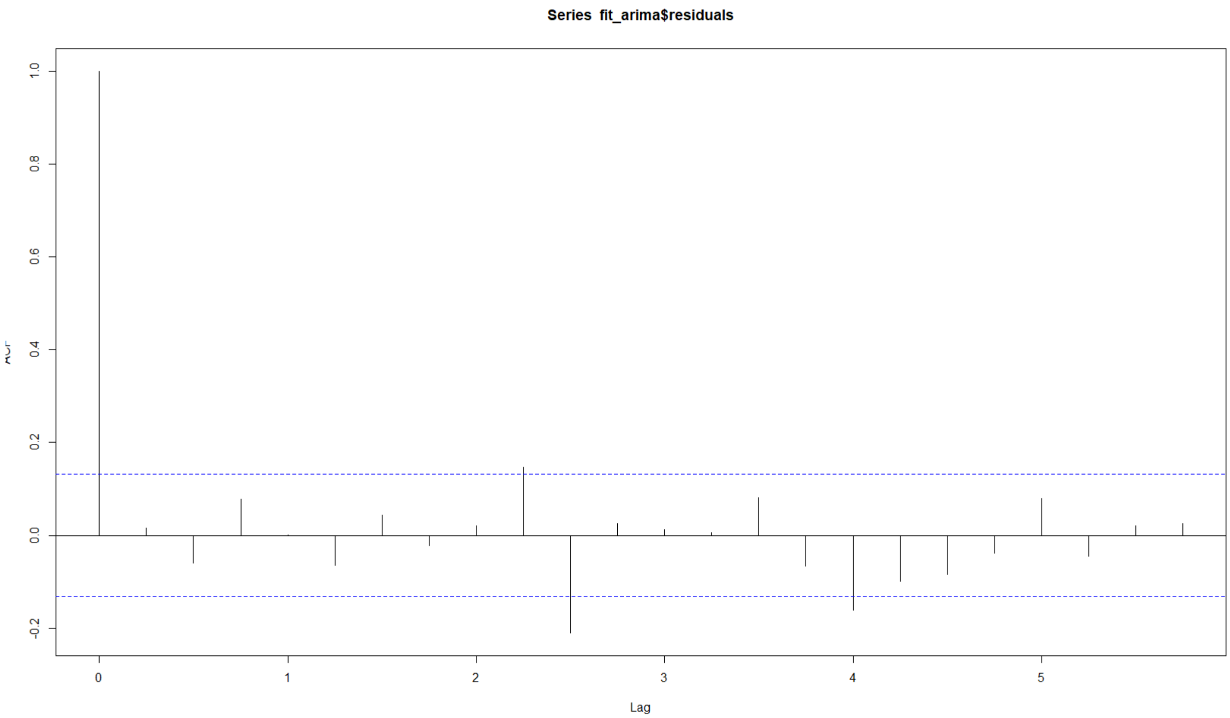
As for previous model, I looked at the residuals to see if the model is a good fit for the time series.

```
> #look at the  residuals to check nodel fit
> plot.ts(fit_arima$residuals)
```

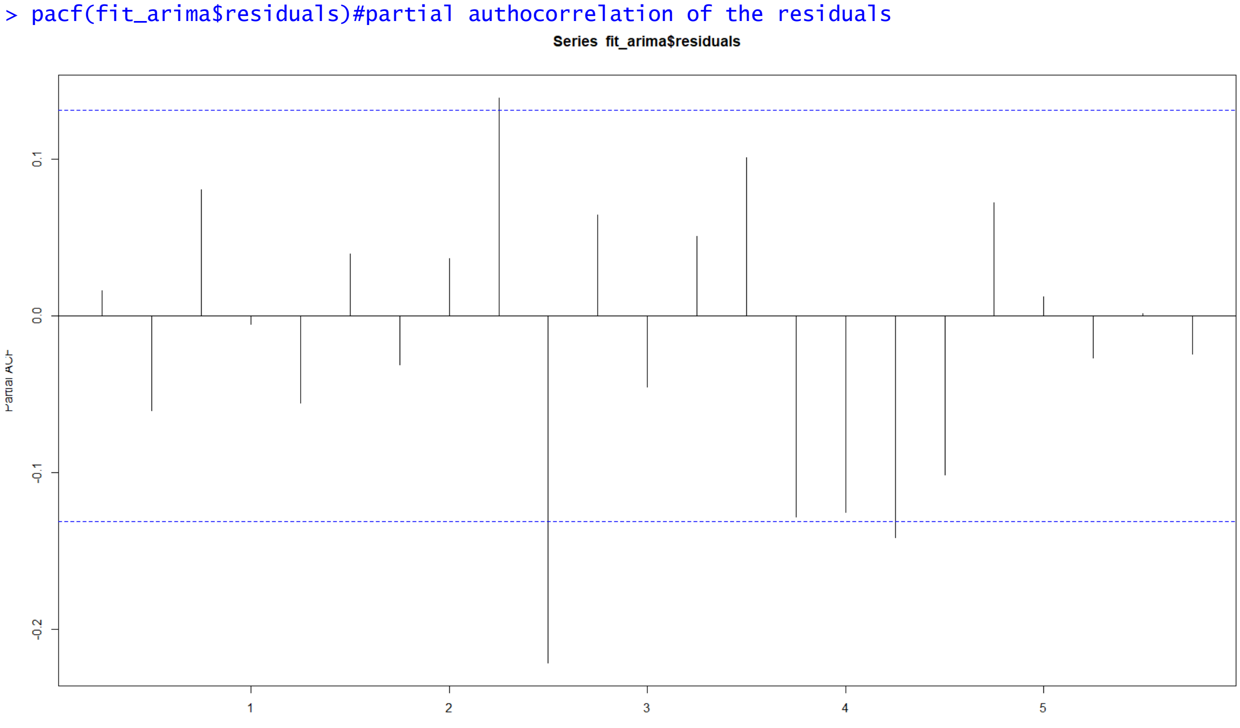


The plot showed, that the residuals seem to have a mean of 0 but an increasing variance. The correlogram and partial correlogram of the residuals did not provide consistent results, as while they mostly were close enough to zero, partial autocorrelation exhibits some fluctuations.

```
> acf(fit_arima$residuals)#autocorrelation of the residuals
```




```
> pacf(fit_arima$residuals)#partial authocorrelation of the residuals
```



I performed the Box-Ljung test to look at the independence of the residuals with the following hypothesis:

Ho: Zero autocorrelation (the model does not exhibit lack of fit);

Ha: Non-zero autocorrelation (the model exhibits lack of fit).

```
> Box.test(fit_arima$residuals, type = "Ljung-Box")

	Box-Ljung test

data:  fit_arima$residuals
X-squared = 0.056915, df = 1, p-value = 0.8114
```

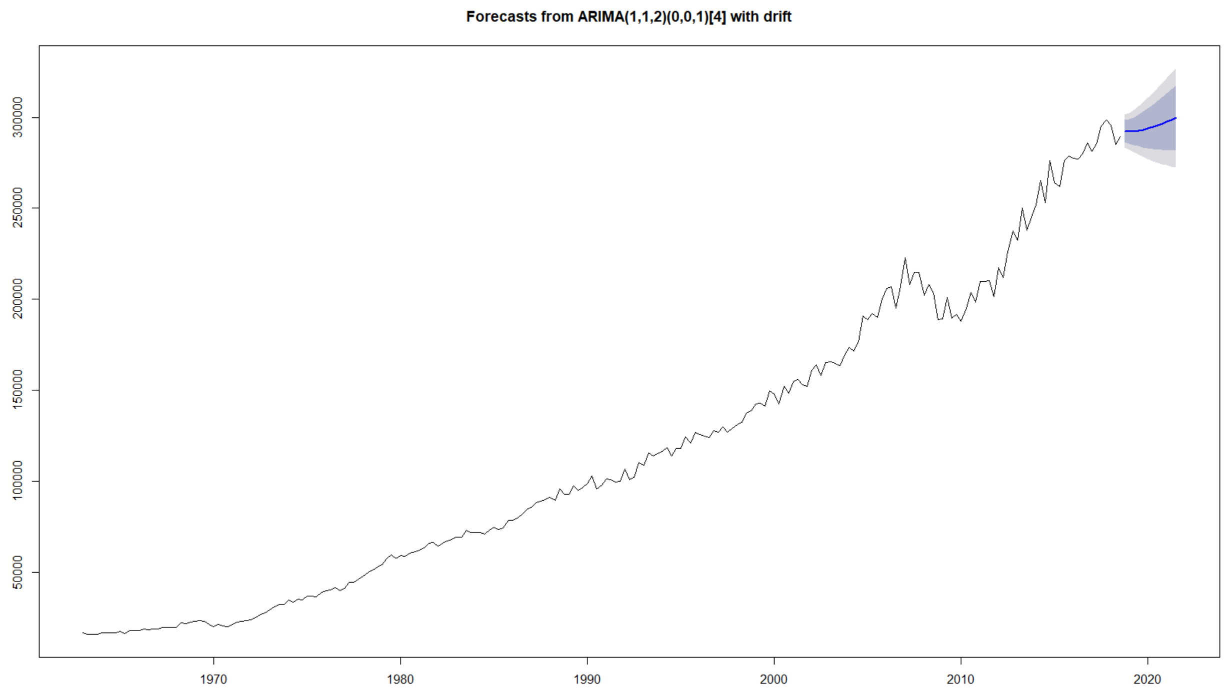
The test returned the p=value of 0.8114, which means that there is no evidence to reject the null hypothesis, stating that there is no autocorrelation between the residuals and that the model's fit does not need to be improved.

This allowed me to proceed with using the model for predictions.

```
> acf(fit_arima$residuals)#autocorrelation of the residuals
> pacf(fit_arima$residuals)#partial authocorrelation of the residuals
> #Using ARIMA model for predictions
> prediction_arima <- forecast(fit_arima, h=12) #forecast for the next 12 quarters (3 years)
> prediction_arima
        Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
2018 Q4       292477.3 286525.6 298428.9 283375.0 301579.5
2019 Q1       292262.5 285647.2 298877.7 282145.3 302379.6
2019 Q2       292402.4 284852.0 299952.8 280855.0 303949.8
2019 Q3       292728.9 284078.0 301379.7 279498.5 305959.2
2019 Q4       293252.9 283432.2 303073.5 278233.5 308272.2
2020 Q1       293940.7 282915.3 304966.1 277078.8 310802.6
2020 Q2       294743.1 282518.9 306967.4 276047.7 313438.5
2020 Q3       295634.3 282235.0 309033.7 275141.9 316126.8
2020 Q4       296594.4 282053.3 311135.4 274355.7 318833.0
2021 Q1       297607.7 281962.6 313252.8 273680.6 321534.8
2021 Q2       298662.4 281952.5 315372.2 273106.9 324217.9
2021 Q3       299749.1 282013.4 317484.8 272624.7 326873.5
```

The above results are in the initial units (US dollars) as the model was built using the original time series. Below is the visual presentation of the predictions, including 80% and 95% confidence intervals:

```
> plot(prediction_arima)
```



Overall, both Holt-Winters and ARIMA models generated predictions of continually increasing median single home prices for the next three years, however, the point predictions are slightly different. An advantage of the auto.arima() command used that it can handle both seasonal and non-seasonal ARIMA. In addition, ARIMA can handle differencing internally and produce forecast in the initial units, which are easily visualized. However, fitting models using the initial time series and the transformed times series also caused inability to directly compare the models, as all error measures were calculated in different units and the AIC cannot reliably be used to compare different types of models (e.g., ets and ARIMA) as they treat the initial values differently.