exercise_part01

Forecasting: Time Series Exploration

Original Source

Exercise 1

Read data from the file sales.csv

```
df <- read.csv("sales.csv")
```

Exercise 2

Transform the data into a time series object of the ts type (indicate that the data is monthly, and the starting period is Jan 1992). Print data.

```
series <- ts(df, frequency = 12, start = c(1992,1))
print(series)
```

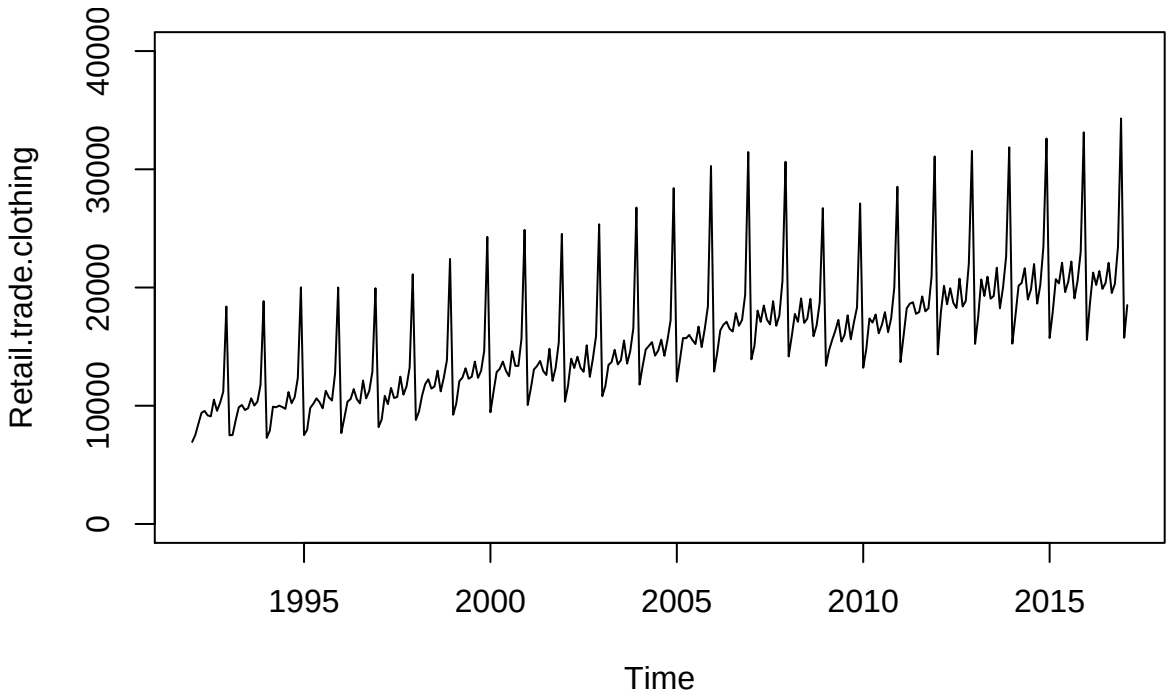
```
##        Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov
## 1992  6938  7524  8475  9401  9558  9182  9103 10513  9573 10254 11187
## 1993  7502  7524  8766  9867 10063  9635  9794 10628 10013 10346 11760
## 1994  7280  7902  9921  9869 10009  9893  9735 11157 10217 10730 12354
## 1995  7518  7961  9815 10168 10620 10301  9784 11264 10710 10439 12751
## 1996  7684  8991 10349 10570 11405 10554 10202 12134 10623 11250 12875
## 1997  8194  8835 10840 10131 11505 10654 10734 12461 10942 11635 13244
## 1998  8800  9499 10863 11825 12239 11451 11633 12971 11214 12384 13854
## 1999  9237 10171 12081 12386 13167 12280 12461 13734 12357 12948 14643
## 2000  9447 11170 12841 13124 13735 12953 12500 14610 13375 13369 15675
## 2001 10060 11450 13067 13362 13787 12935 12600 14818 12104 13218 15352
## 2002 10344 11730 13977 13195 14150 13210 12873 15113 12445 14006 15911
## 2003 10804 11662 13452 13691 14730 13496 13854 15522 13567 14601 16555
## 2004 11790 13344 14760 15058 15379 14237 14667 15588 14224 15570 17230
## 2005 12046 13878 15727 15708 15989 15559 15218 16697 14960 16509 18402
## 2006 12893 14474 16386 16848 17103 16505 16275 17832 16767 17253 19391
## 2007 13927 15077 18045 17096 18474 17289 16883 18850 16765 17614 20550
## 2008 14170 15877 17764 17098 19081 17006 17366 19038 15881 16791 18753
## 2009 13382 14681 15560 16334 17260 15429 16002 17650 15624 17046 18324
## 2010 13208 14809 17384 17034 17717 16129 16834 17915 16224 17403 19995
## 2011 13692 15920 18241 18634 18756 17772 17925 19246 17984 18242 20941
## 2012 14340 17909 20144 18586 19938 18709 18265 20751 18400 18860 22040
## 2013 15230 17484 20681 19285 20914 19056 19280 21676 18244 19992 22611
## 2014 15252 17608 20146 20388 21631 18984 19848 21987 18640 20214 23473
## 2015 15733 17919 20711 20346 22093 19612 20499 22197 19080 20563 23007
## 2016 15564 18712 21264 20221 21389 19883 20381 22086 19538 20306 23438
## 2017 15748 18511
##        Dec
## 1992 18395
## 1993 18851
## 1994 20016
```

```
## 1995 20002
## 1996 19944
## 1997 21118
## 1998 22418
## 1999 24286
## 2000 24875
## 2001 24534
## 2002 25350
## 2003 26760
## 2004 28406
## 2005 30276
## 2006 31462
## 2007 30635
## 2008 26718
## 2009 27110
## 2010 28526
## 2011 31085
## 2012 31551
## 2013 31860
## 2014 32604
## 2015 33123
## 2016 34300
## 2017
```

Exercise 3

Plot the time series. Ensure that the y axis starts from zero.

```
plot(series, ylim=c(0,40000))
```



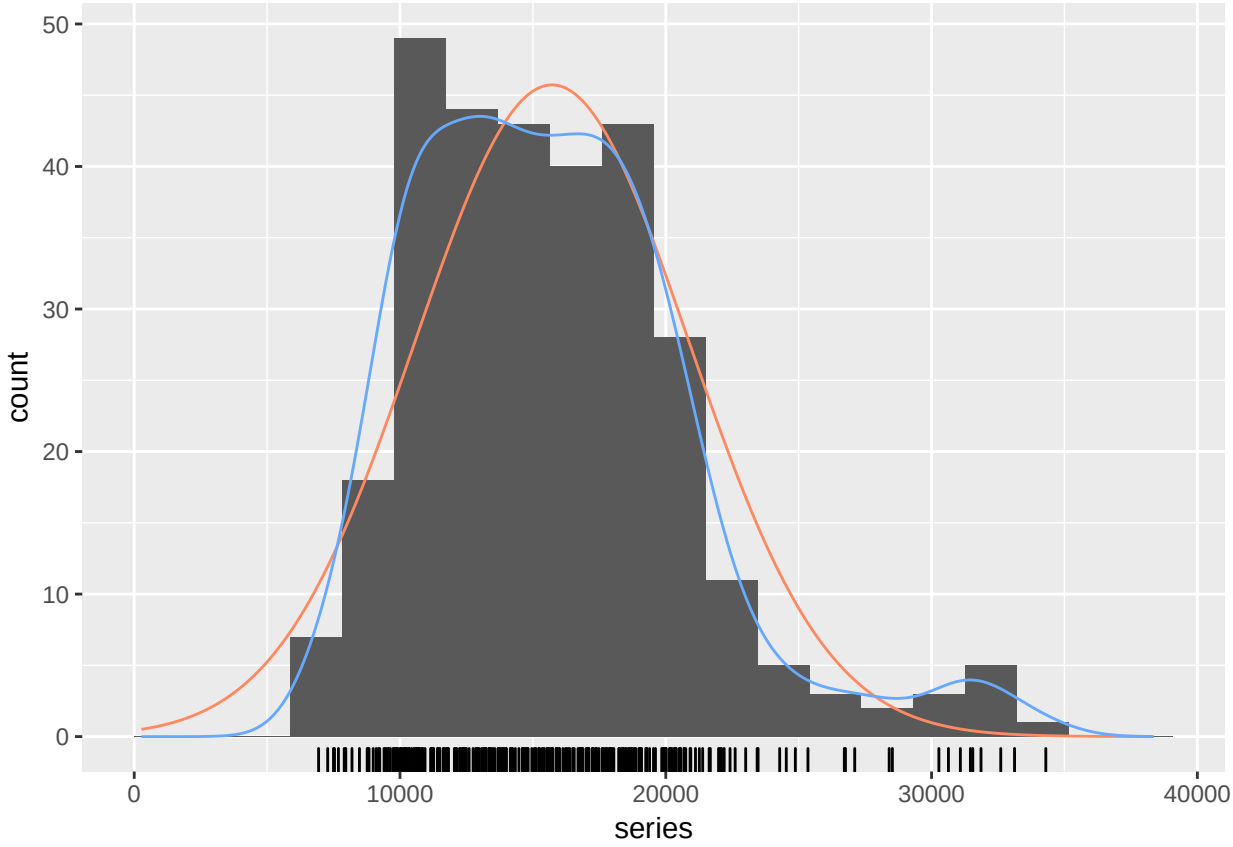
Exercise 4

Use the gghistogram function from the **forecast** package to visually inspect the distribution of time series values. Add a kernel density estimate and a normal density function to the plot.

```
require(forecast)
```

```
## Loading required package: forecast
```

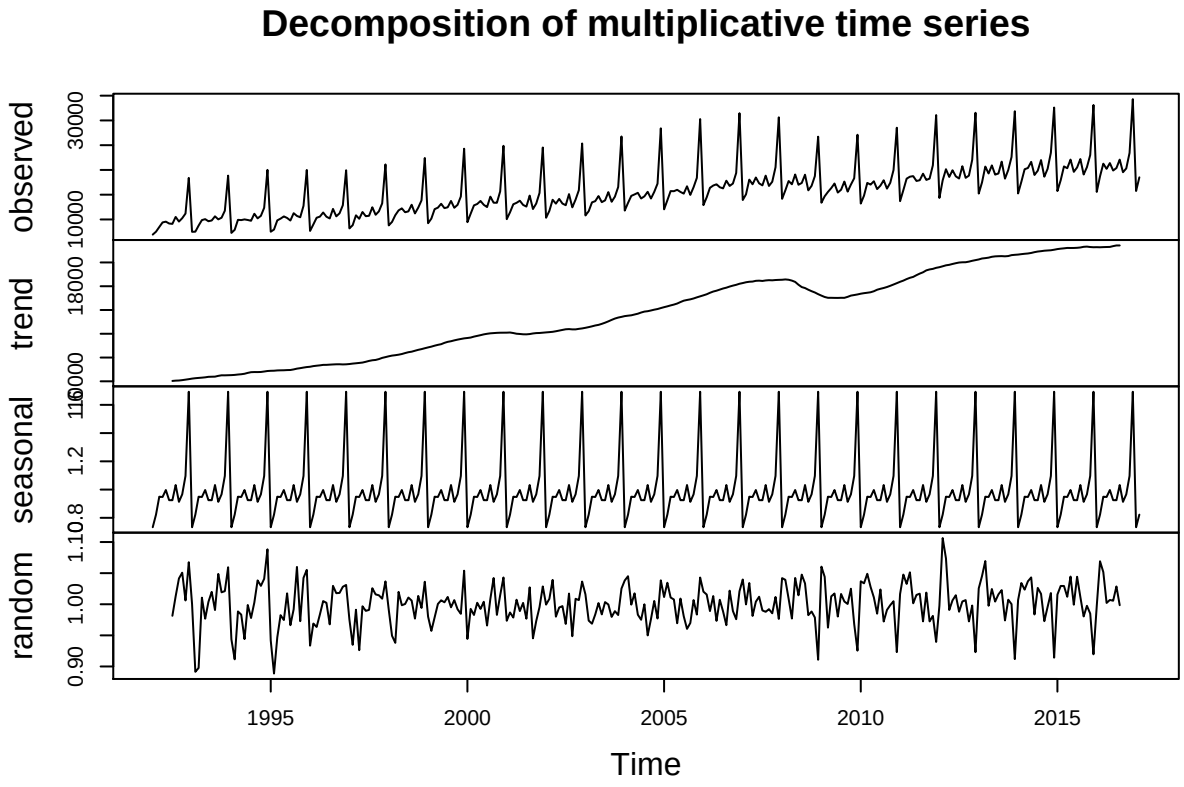
```
gghistogram(series, add.normal = TRUE, add.kde = TRUE)
```



Exercise 5

Use the **decompose** function to break the series into seasonal, trend, and irregular components (apply multiplicative decomposition). Plot the decomposed series.

```
series_decomposed <- decompose(series, type='multiplicative')
plot(series_decomposed)
```



Exercise 6

Explore the structure of the decomposed object, and find seasonal coefficients (multiples). Identify the three months with the greatest coefficients, and the three months with the smallest coefficients. (Note that the coefficents are equal in different years).

```
seasonal_coef <- data.frame(series_decomposed$figure)
seasonal_coef <- cbind(seasonal_coef, seq(1:12))
seasonal_coef[order(-seasonal_coef[,1]), ]
```

```
##    series_decomposed.figure seq(1:12)
## 12                1.6926382        12
## 11                1.0966024        11
## 8                 1.0321913         8
## 5                 0.9969609         5
## 10                0.9663104        10
## 3                 0.9495724         3
## 4                 0.9473508         4
## 6                 0.9249434         6
## 7                 0.9245634         7
## 9                 0.9130038         9
## 2                 0.8225737         2
## 1                 0.7332894         1
```

Exercise 7

Check whether the times eries is trend-stationary (i.e. its mean and variance are constant with respect to a trend) using function **kpss.test** from the **tseries** package. (Note that the null hypothesis of the test is that

the series is trend-stationary).

```
require(tseries)
```

```
## Loading required package: tseries
```

```
kpss.test(series)
```

```
## Warning in kpss.test(series): p-value smaller than printed p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  series
## KPSS Level = 5.551, Truncation lag parameter = 4, p-value = 0.01
```

Exercise 8

Use the **diff** function to create a differenced time series (i.e. a series that includes differences between the values of the original series), and test it for trend stationarity.

```
series_differenced <- diff(series)
kpss.test(series_differenced)
```

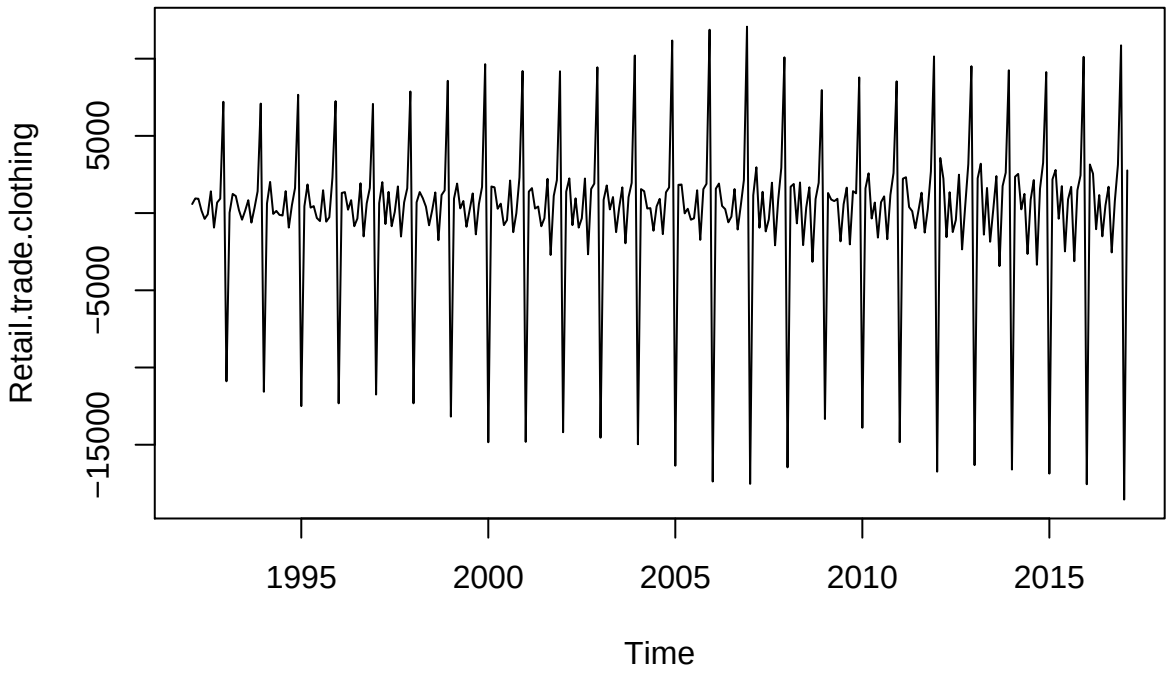
```
## Warning in kpss.test(series_differenced): p-value greater than printed p-
## value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  series_differenced
## KPSS Level = 0.013917, Truncation lag parameter = 4, p-value = 0.1
```

Exercise 9

Plot the differenced time series.

```
plot(series_differenced)
```



Exercise 10

Use the **Acf** and **Pacf** functions from the **forecast** package to explore autocorrelation of the differenced seires. Find at which lags correlation between lagged values is tstatistically significant at 5% level.

```
require(forecast)
Acf(series_differenced)
```

```
Pacf(series_differenced)
```