Time Series - Final Project

by Krystian Bułat, Agnieszka Tracz, Karolina Grzech and Zuzanna Maciszewska

You have two tasks below:

TASK 1:

Consider the monthly *sales of products* (see file dane_KKAZ.csv at UPeL)).

1. Make *sales of products* forecasts for each month of the year 2024.

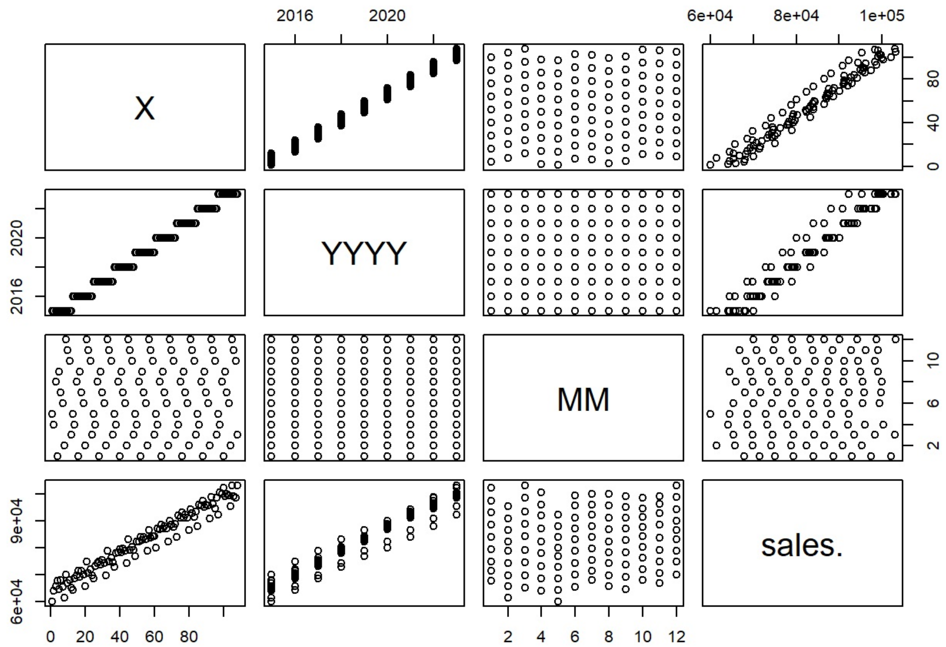
SOLUTION of TASK 1

We are loading our data, which consists of 4 columns and 108 observations. The first column pertains to the observation index, the next two are associated with dates, and the last one contains information about sales. In this process, we add an additional column containing complete date information (day and month), and then proceed to remove the initial three columns from our database.

```
library(readxl)
```

```
## Warning: pakiet 'readxl' został zbudowany w wersji R 4.2.1
```

```
dane <- read.csv("dane_KKAZ.csv")

plot(dane)
```

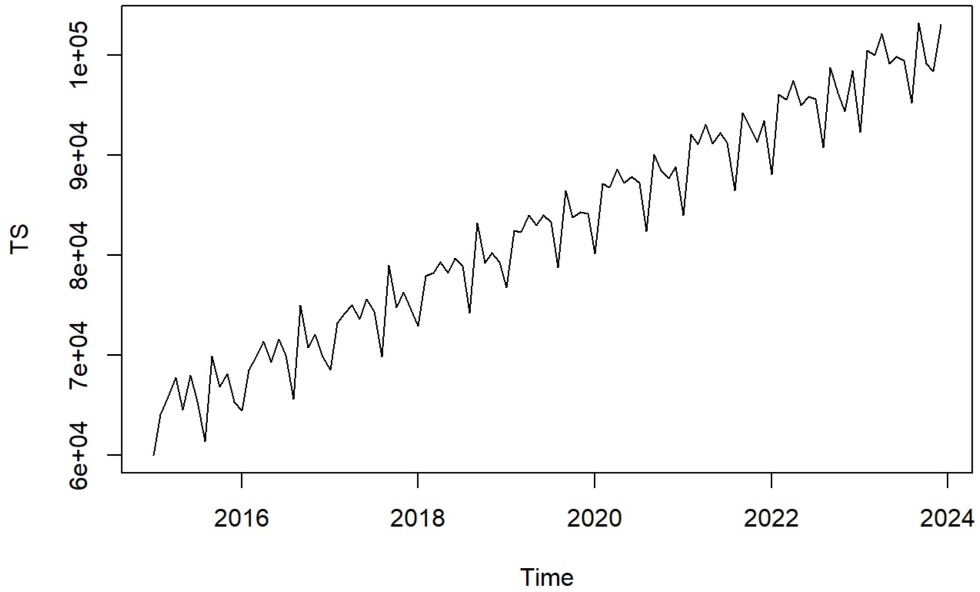



```
dane$data <- paste(dane$MM, dane$Y)

dane1 <- dane[,5:4]

summary(dane1)
```

```
##      data               sales.
##  Length:108         Min.   : 59977
##  Class :character   1st Qu.: 73540
##  Mode  :character   Median : 83111
##                     Mean   : 82585
##                     3rd Qu.: 92122
##                     Max.   :103195
```

We are creating a time series composed of date and sales, and then plotting its graph.

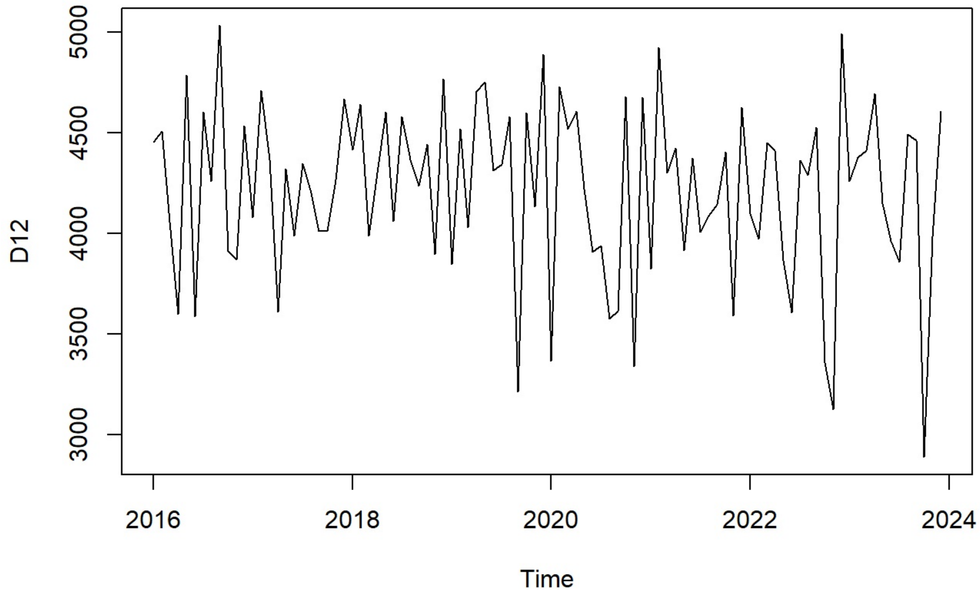
```
TS <- ts(dane1$sales , start = c(2015,1), end = c(2023,12) ,frequency=12 )

plot(TS)
```



The variance in our time series does not exhibit significant growth, so there is no necessity to apply the Box-Cox transformation. However we can see trends and seasonality in the data, we decided to choose a period based on the nature of our data, so here period=year

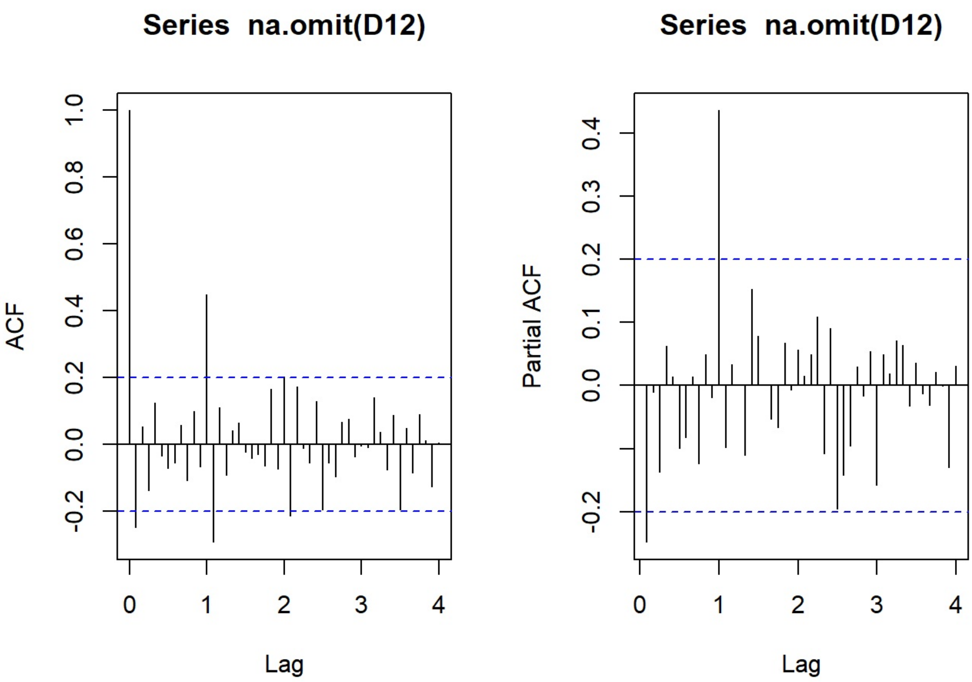
In the next step, we use the diff function to eliminate the seasonality from our data.

```
D12<-diff(TS,lag=12)
plot(D12)
```



The seasonality has been successfully removed, so D=1. We are plotting the autocorrelation function (ACF) and partial autocorrelation function (PACF) graphs for deseasonalized time series.

```
par(mfrow=c(1,2))
LagMax<-48
acf(na.omit(D12), lag.max=LagMax) #Watch out for missing data
pacf(na.omit(D12), lag.max=LagMax) #Watch out for missing data
```



They are similar to the theoretical ACF and PACF of SARIMA models, the sample ACF tends to zero so it may suggest d=0, so we will consider ARIMA(p,0,q)x(P,1,Q)12

Seasonsal Component: It seems that at the seasons, the sample ACF and PACF are cutting off at lag 1s (s = 12). Therefore, we can try the pairs: P=1& Q=0; P=0& Q=1; P=1& Q=1.

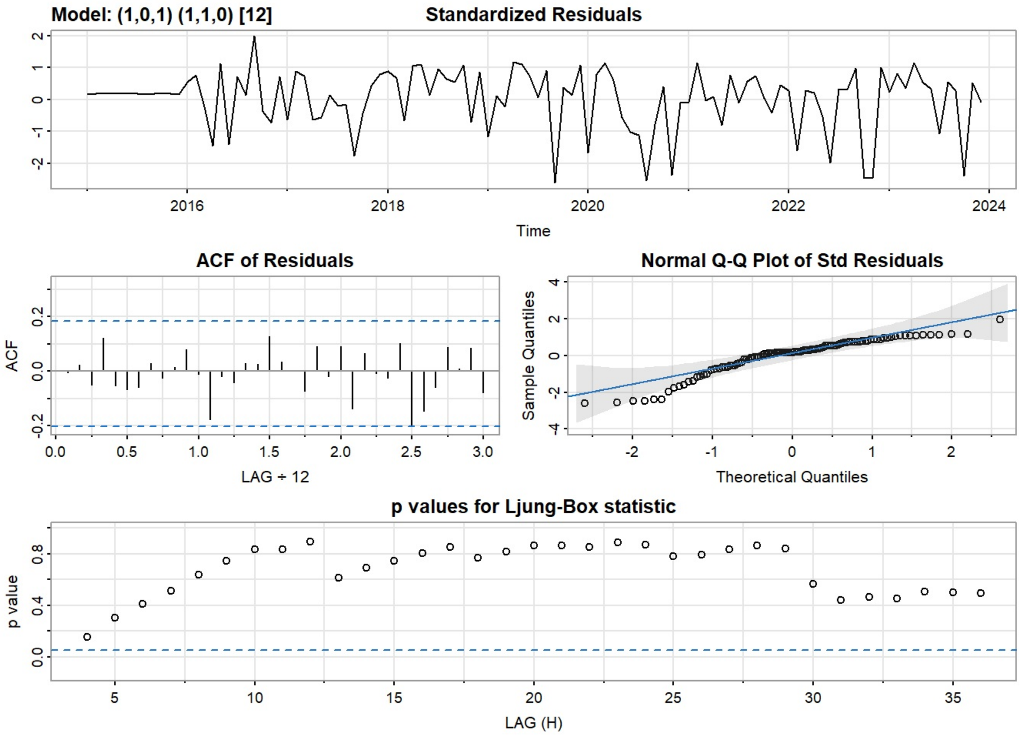
Non-SeasonsalComponent: At the lower lags, the sample ACF and PACF are tailing off. Since we prefer the low number of parameters we can try p=q=1

```
library(astsa)
```

```
## Warning: pakiet 'astsa' został zbudowany w wersji R 4.2.2
```

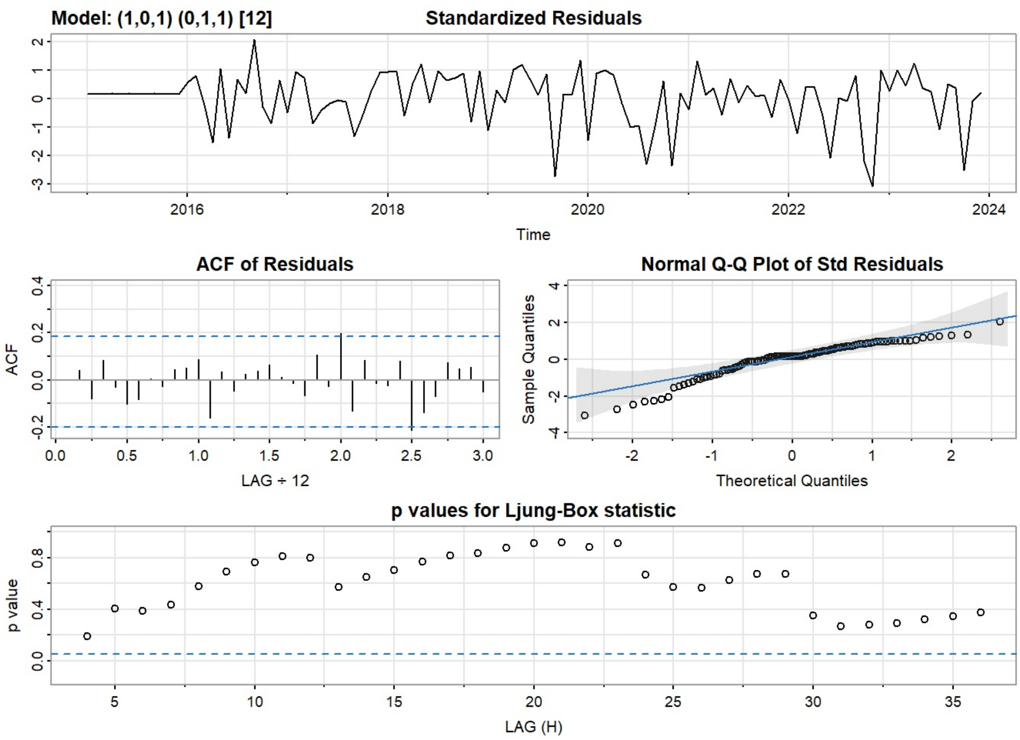
```
model_1 <- sarima(TS,1,0,1,1,1,0,12)# p,d,q, P,D,Q, s
```

```
## initial  value 6.065772
## iter   2 value 5.942886
## iter   3 value 5.899347
## iter   4 value 5.898358
## iter   5 value 5.898083
## iter   6 value 5.898080
## iter   7 value 5.898075
## iter   8 value 5.898070
## iter   9 value 5.898056
## iter  10 value 5.898029
## iter  11 value 5.898008
## iter  12 value 5.897997
## iter  13 value 5.897995
## iter  14 value 5.897994
## iter  15 value 5.897978
## iter  16 value 5.897960
## iter  17 value 5.897944
## iter  18 value 5.897939
## iter  19 value 5.897938
## iter  19 value 5.897938
## final  value 5.897938
## converged
## initial  value 5.913800
## iter   2 value 5.913141
## iter   3 value 5.912174
## iter   4 value 5.912090
## iter   5 value 5.911919
## iter   6 value 5.911637
## iter   7 value 5.911048
## iter   8 value 5.910537
## iter   9 value 5.910220
## iter  10 value 5.910211
## iter  10 value 5.910211
## final  value 5.910211
## converged
```



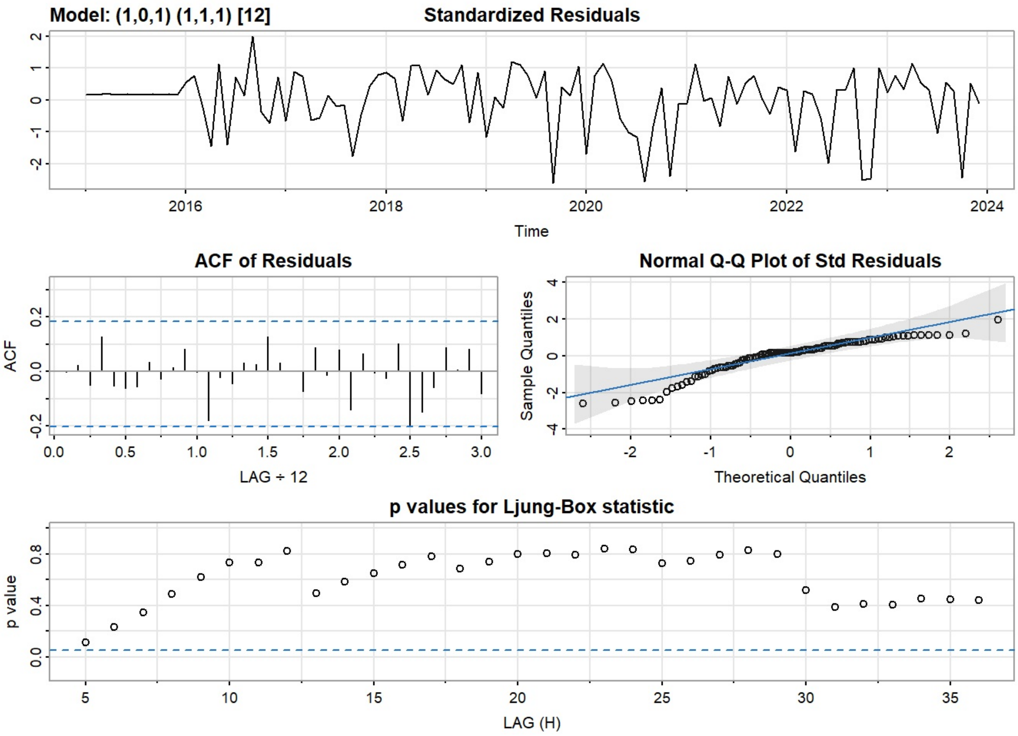

```
model_2 <- sarima(TS,1,0,1,0,1,1,12)
```

```
## initial  value 6.069513 
## iter   2 value 5.985460
## iter   3 value 5.946823
## iter   4 value 5.946379
## iter   5 value 5.945937
## iter   6 value 5.945890
## iter   7 value 5.945515
## iter   8 value 5.945135
## iter   9 value 5.944918
## iter  10 value 5.944867
## iter  11 value 5.944866
## iter  12 value 5.944864
## iter  12 value 5.944864
## iter  12 value 5.944864
## final  value 5.944864 
## converged
## initial  value 5.941643 
## iter   2 value 5.941449
## iter   3 value 5.941409
## iter   4 value 5.941405
## iter   5 value 5.941403
## iter   6 value 5.941402
## iter   7 value 5.941396
## iter   8 value 5.941390
## iter   9 value 5.941383
## iter  10 value 5.941381
## iter  11 value 5.941381
## iter  11 value 5.941381
## iter  11 value 5.941381
## final  value 5.941381 
## converged
```




```
model_3 <- sarima(TS,1,0,1,1,1,1,12)
```

```
## initial  value 6.065772
## iter   2 value 6.059913
## iter   3 value 5.901160
## iter   4 value 5.899855
## iter   5 value 5.898270
## iter   6 value 5.897121
## iter   7 value 5.897082
## iter   8 value 5.897080
## iter   9 value 5.897078
## iter  10 value 5.897072
## iter  11 value 5.897060
## iter  12 value 5.897037
## iter  13 value 5.897028
## iter  14 value 5.897016
## iter  15 value 5.897014
## iter  16 value 5.897013
## iter  17 value 5.897012
## iter  18 value 5.897010
## iter  19 value 5.897008
## iter  20 value 5.897005
## iter  21 value 5.897000
## iter  22 value 5.896992
## iter  23 value 5.896987
## iter  24 value 5.896986
## iter  25 value 5.896985
## iter  25 value 5.896985
## iter  25 value 5.896985
## final  value 5.896985
## converged
## initial  value 5.915308
## iter   2 value 5.913706
## iter   3 value 5.912859
## iter   4 value 5.912590
## iter   5 value 5.912200
## iter   6 value 5.911708
## iter   7 value 5.911573
## iter   8 value 5.911380
## iter   9 value 5.911140
## iter  10 value 5.910717
## iter  11 value 5.910281
## iter  12 value 5.910018
## iter  13 value 5.910016
## iter  14 value 5.910014
## iter  15 value 5.910013
## iter  16 value 5.910010
## iter  17 value 5.910007
## iter  18 value 5.910005
## iter  19 value 5.910003
## iter  20 value 5.910001
## iter  21 value 5.909998
## iter  22 value 5.909997
## iter  23 value 5.909996
## iter  23 value 5.909996
## final  value 5.909996
## converged
```




```
c(model_1$AIC, model_1$AICc,model_1$BIC)
```

```
## [1] 14.76247 14.76704 14.89603
```

```
c(model_2$AIC, model_2$AICc,model_2$BIC)
```

```
## [1] 14.82481 14.82938 14.95837
```

```
c(model_3$AIC, model_3$AICc,model_3$BIC);
```

```
## [1] 14.78287 14.78981 14.94314
```

```
model_1$ttable
```

```
##          Estimate     SE t.value p.value
## ar1        0.4087 0.6200  0.6592  0.5114
## ma1       -0.5736 0.5624 -1.0200  0.3104
## sar1       0.5103 0.0926  5.5119  0.0000
## constant 351.7106 4.0596 86.6362  0.0000
```

```
model_2$ttable
```

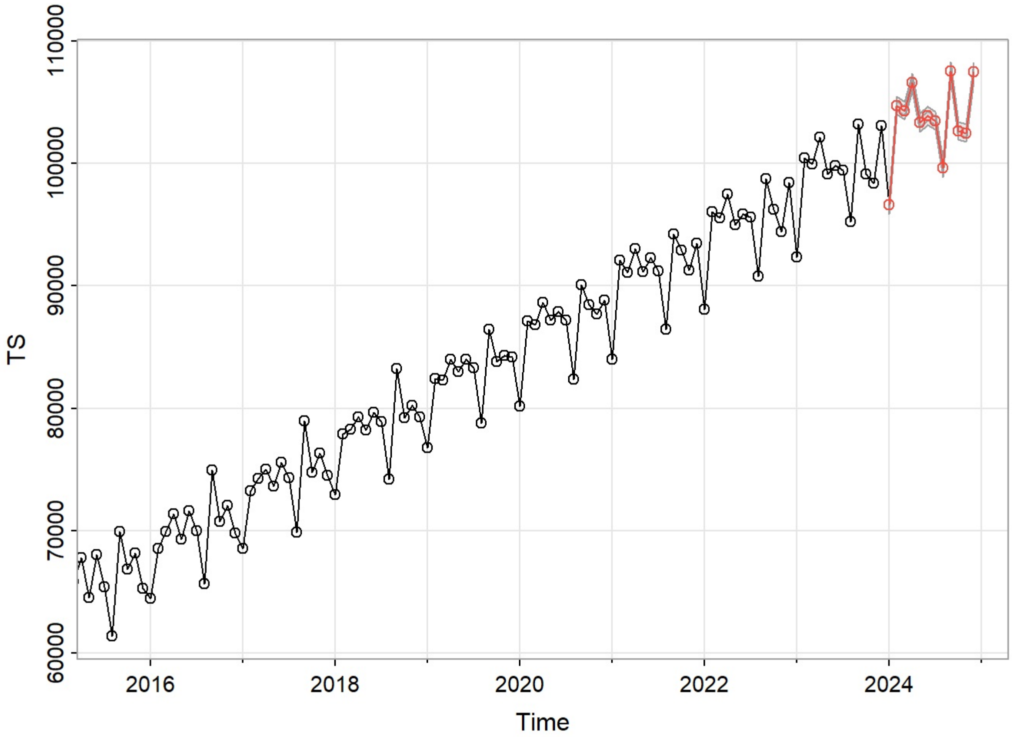
```
##          Estimate     SE  t.value p.value
## ar1        0.1622 0.5743   0.2824  0.7783
## ma1       -0.3783 0.5446  -0.6947  0.4890
## sma1       0.4056 0.0877   4.6225  0.0000
## constant 351.9200 3.2383 108.6727  0.0000
```

```
model_3$ttable
```

```
##          Estimate     SE t.value p.value
## ar1        0.4142 0.5629  0.7358  0.4637
## ma1       -0.5808 0.5078 -1.1438  0.2557
## sar1       0.5384 0.1614  3.3365  0.0012
## sma1      -0.0369 0.1818 -0.2028  0.8397
## constant 351.6790 4.0851 86.0892  0.0000
```

We can see that all the metrics prefer the first model, so our final fit is ARIMA(1,0,1)×(1,1,0)12

```
pred<-sarima.for(TS,n.ahead=12,1,0,1,1,1,0,12)
```



{r}

TASK 2:

Consider the Krystian’s time series from Lab 1 task 4.2 (https://upel.agh.edu.pl/mod/forum/discuss.php?d=8847) (Netflix Stock Price)

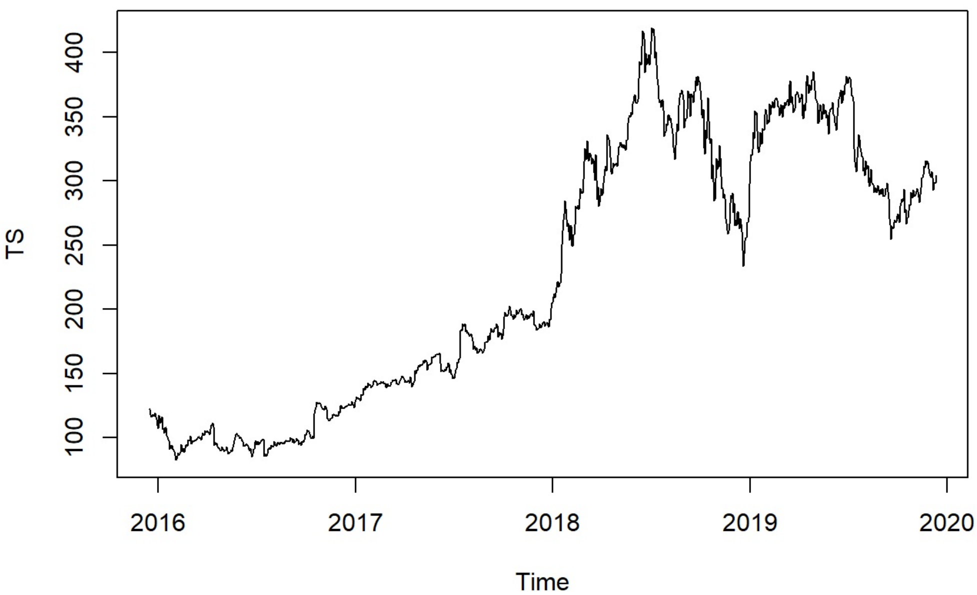
1. Make all necessary transformations to get a stationary time series (noise).
2. If possible, choose the ARMA (p, q) model for stationary noise.

SOLUTION of TASK 2

We are loading our data. The dataset comprises 7 columns and consists of 1007 observations. The first column contains date information, while the fifth column represents the ‘close’ values. In our analysis, we will construct a time series based on these two columns.

```
dane <- read.csv("NFLX.csv")
TS <- dane[, c(1,5)]
```

```
TS <- ts(TS$Close, start = c(2015, 241), end = c(2019,  239), frequency = 252)
plot(TS)
```



```r
library(tseries)
```

```
## Warning: pakiet 'tseries' został zbudowany w wersji R 4.2.3
```

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

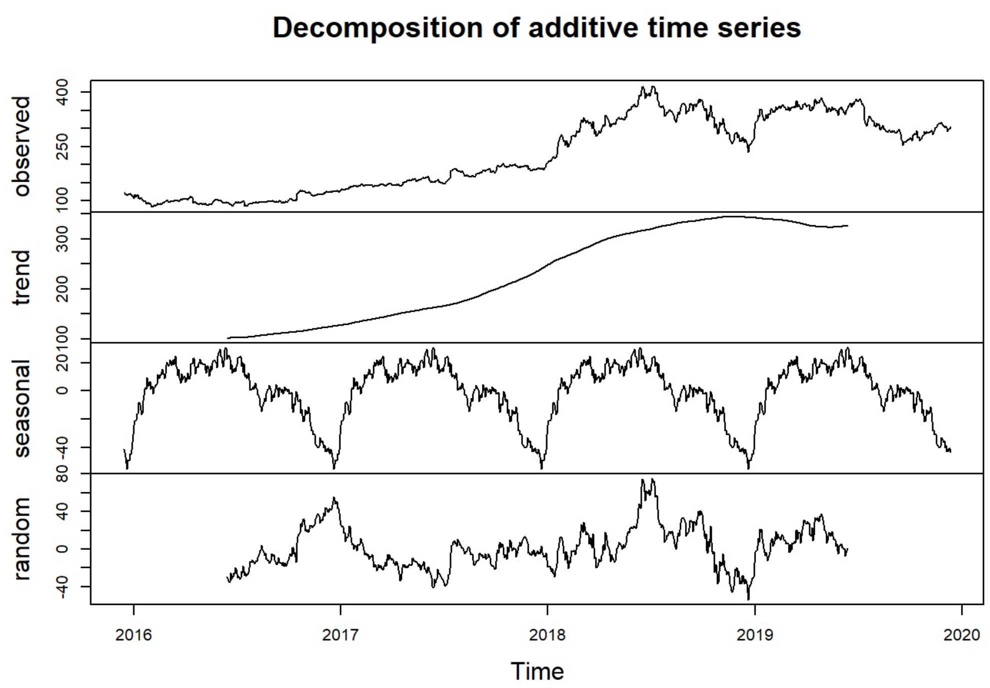
```r
adf.test(TS)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  TS
## Dickey-Fuller = -1.4872, Lag order = 10, p-value = 0.7954
## alternative hypothesis: stationary
```

The p-value is greater than 0.05, so we cannot reject the null hypothesis, indicating that the time series is non-stationary.

We utilize the 'decompose' function to decompose the time series, separating it into its individual components such as trend, seasonality, and remainder.

```r
decomposed <- decompose(TS)
plot(decomposed)
```



The increasing variance observed in the above plot indicates the need for a Box-Cox transformation. Employing Box-Cox helps stabilize the variance, a crucial step in achieving a more stationary and predictable time series.

```r
library(forecast)
```

```
## Warning: pakiet 'forecast' został zbudowany w wersji R 4.2.3
```

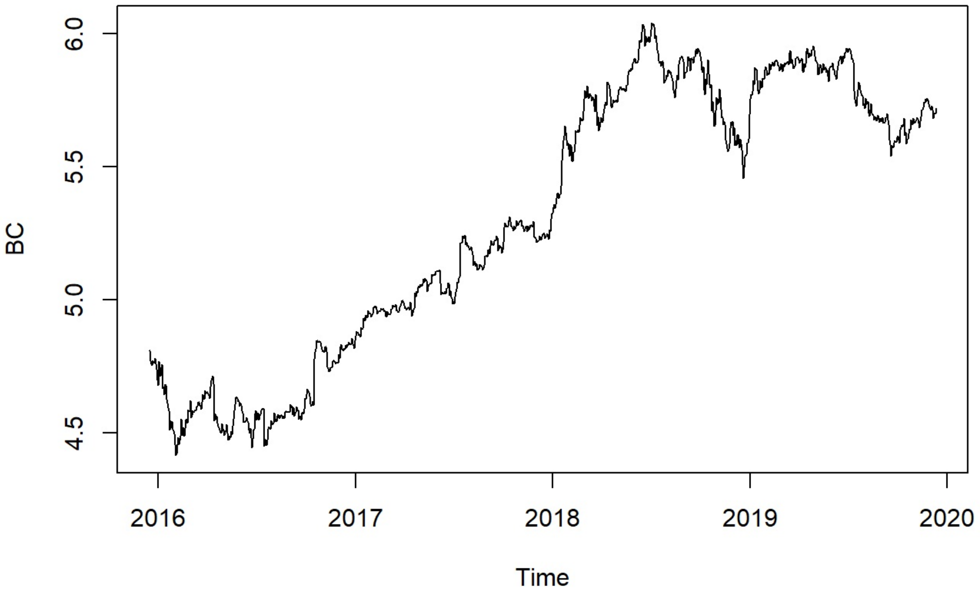
```
##
## Dołączanie pakietu: 'forecast'
```

```
## Następujący obiekt został zakryty z 'package:astsa':
##
##     gas
```

```r
lambda <- BoxCox.lambda(TS,lower=0)
lambda
```

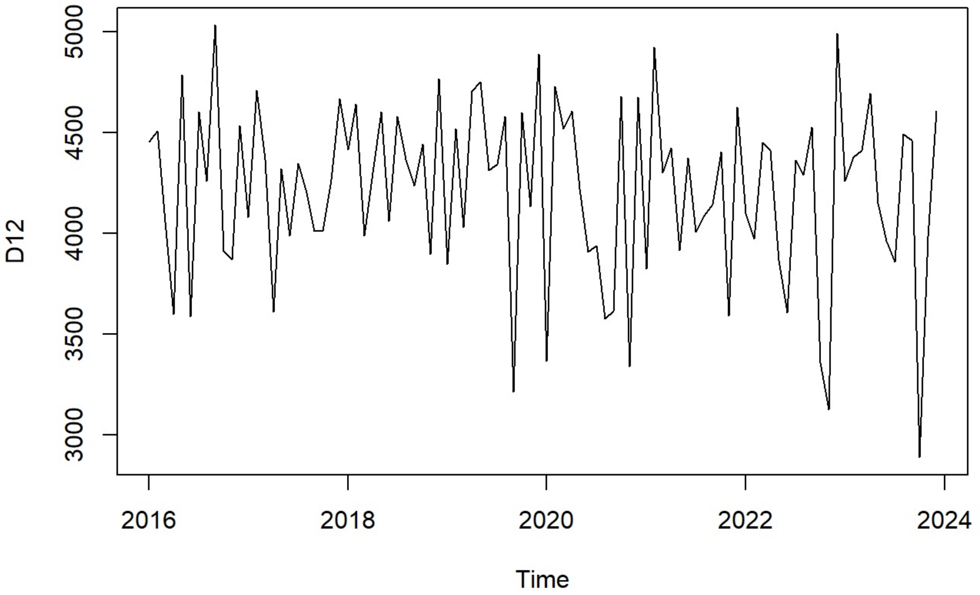
```
## [1] 4.102259e-05
```

```
BC <- BoxCox(TS,lambda)
plot(BC)
```



We can observe the presence of a trend in the data. We apply the ‘diff’ function to compute differences between consecutive values, aiming to remove the trend component and make the time series more stationary for further analysis.

```
D1<-diff(BC,lag=1)
plot(D12)
```



```
adf.test(D1)
```

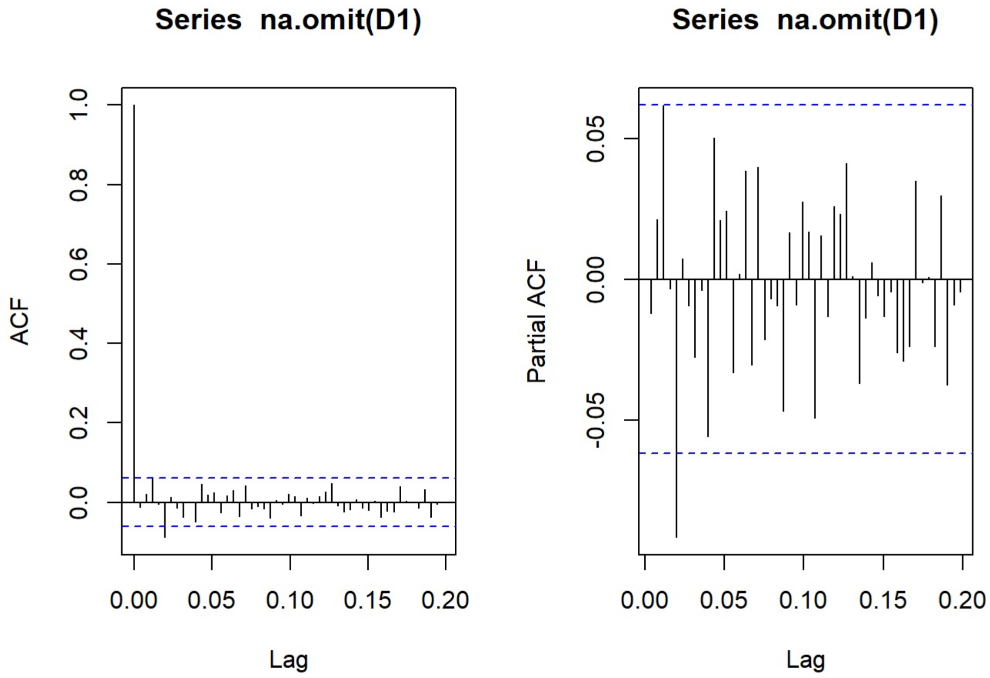
```
## Warning in adf.test(D1): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  D1
## Dickey-Fuller = -10.006, Lag order = 10, p-value = 0.01
## alternative hypothesis: stationary
```

The obtained p-value is now equal to 0.01, so we reject the null hypothesis of non-stationarity.

We can proceed to the analysis of ACF and PACF.

```
par(mfrow=c(1,2))
LagMax<-50
acf(na.omit(D1), lag.max=LagMax)
pacf(na.omit(D1), lag.max=LagMax)
```



The PACF seems to be cut off after lag 5 and ACF has tails off, so we decide to choose AR(5).