ARIMA_Workshop

Workshop 3 2020

Dr. Sparkle L. Malone

The primary objectives of the workshop:

1. Create a time series object.
2. Decompose a time series.
3. Test for stationarity and detecting autocorrelation.
4. Choose the order of an ARIMA model.
5. Use explanatory series to improve models.

Data:

The data provided in ARIMA_Workshop includes one data set of daily rates of NEE for a tower site in a mangrove scrub (mangroves) along the coast of Florida. The dataset includes gap-filled NEE (g C m^2 day^{-1}). Data was obtained from an eddy covariance tower in Everglades National Park at the Florida Coastal Everglades Long-term Ecological Research site TS/Ph-7. Data includes NEE (g C m^2 day^{-1}), total daily PAR (par; W m^{-2} day^{-1}), air temperature (tair; C), water temperature minimum and maximum, (water.tmax and water.min; C), and water salinity minimum, maximum, and mean (salinity.min, salinity.max, salinity.mean; ppt).

```
load("~/OneDrive - Florida International University/Teaching/Workshops/Workshops/ARIMA/ARIMA_Workshop.RD
```

Libraries:

```
library(zoo)
library(tseries)
library(forecast)
library(xts)
```

Part 1:

1. Create timeseries objects:

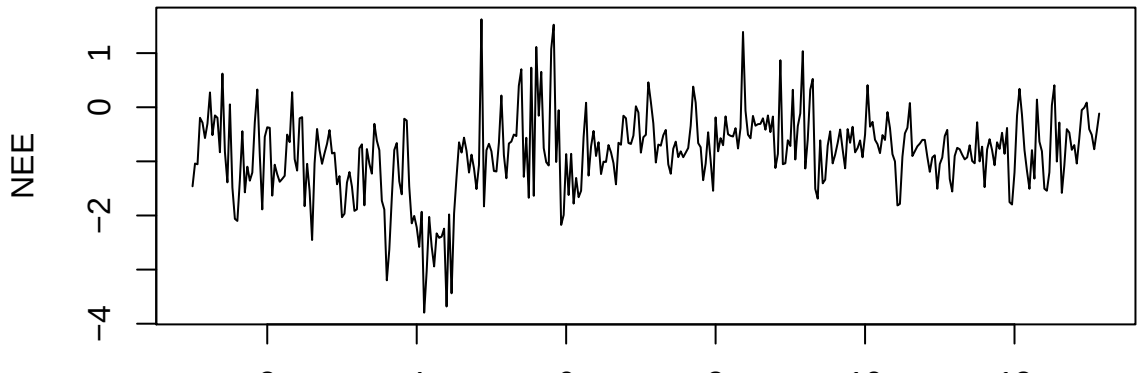
In the frequency parameter in the ts() object, we are specifying periodicity of the data, i.e., number of observations per period. Since we are using daily data, we have 30 observations per month.

```
nee <- ts( mangroves$nee, start= 1, frequency=30)
```

Visualize data:

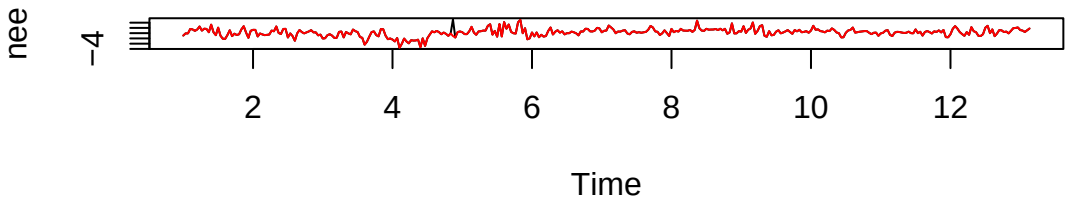
A good starting point is to plot the timeseries and visually examine it for any outliers, volatility, or irregularities.

```
par(mfrow=c(1,1), mai=c(0.25,0.8,0.1, 0.1))
plot( nee, typ="l", ylab= "NEE", xlab="")
```



You want to remove any outliers that could bias the model by skewing statistical summaries. R provides a convenient method for removing time series outliers: tsclean() as part of its forecast package. tsclean() identifies and replaces outliers using series smoothing and decomposition.

```
plot(nee)
lines(tsclean(nee), col="red")
```




```
nee <- tsclean(nee)
```

2. Decompose the timeseries:

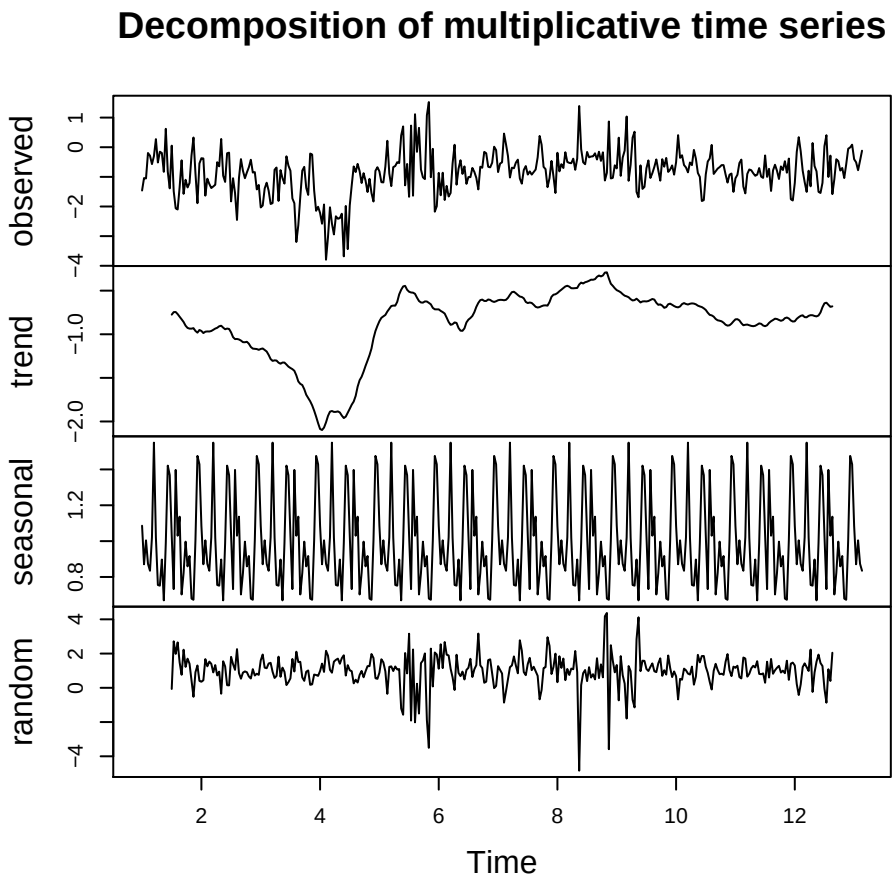
Time series analysis involves trying to separate the time series into the seasonal, trend and irregular components. Deconstructing the series will help you understand its behavior and prepare a foundation for building an ARIMA model.

The Seasonal component refers to fluctuations in the data related to calendar cycles. Usually, seasonality is fixed at some number; for instance, quarter or month of the year.

Trend component is the overall pattern of the series. It consists of decreasing or increasing patterns that are not seasonal. This is estimated using moving averages.

The part of the series that can't be attributed to the seasonal or trend components is referred to as residual or error.

```
nee.d <- decompose(nee, 'multiplicative')
plot(nee.d)
```



3.a. Test for stationarity

Fitting an ARIMA model requires the series to be stationary. A series is stationary when its mean, variance, and autocovariance are time invariant. This assumption makes intuitive sense: Since ARIMA uses previous lags of series to model its behavior, modeling stable series with consistent properties involves less uncertainty.

The augmented Dickey-Fuller (ADF) test is a formal statistical test for stationarity. The null hypothesis assumes that the series is non-stationary. ADF procedure tests whether the change in Y can be explained by lagged value and a linear trend. If contribution of the lagged value to the change in Y is non-significant and there is a presence of a trend component, the series is non-stationary and null hypothesis will not be rejected.

```
# p-value < 0.05 indicates the TS is stationary

adf.test(nee )
```

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

```
##
##  Augmented Dickey-Fuller Test
##
## data:  nee
## Dickey-Fuller = -4.4703, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

3.b. Detecting Autocorrelation:

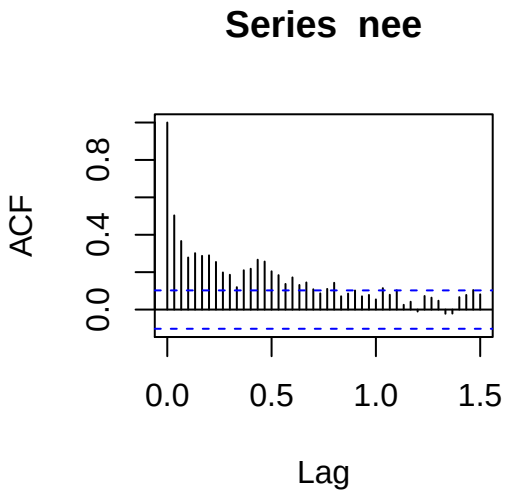
Autocorrelation plots (also known as ACF or the auto correlation function) are a useful visual tool in determining whether a series is stationary. ACF plots display correlation between a series and its lags.

If the series is correlated with its lags then, generally, there are some trend or seasonal components and therefore its statistical properties are not constant over time.

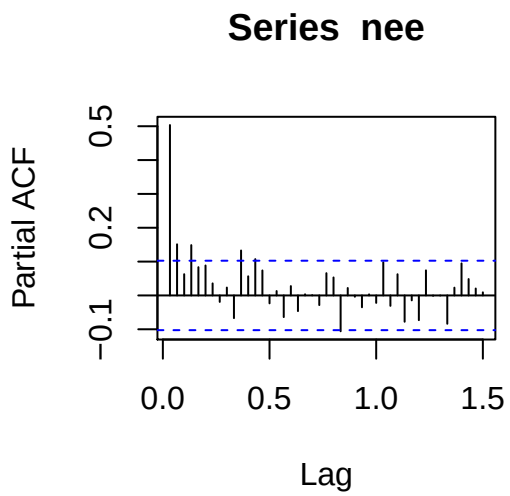
ACF plots can help in determining the order of the MA (q) model. Partial autocorrelation plots (PACF), display correlation between a variable and its lags that is not explained by previous lags. PACF plots are useful when determining the order of the AR(p) model.

R plots 95% significance boundaries as blue dotted lines.

```
acf(nee, lag.max=45)
```




```
pacf(nee, lag.max=45)
```



There are significant autocorrelations with many lags in the nee series, as shown by the ACF plot above. However, this could be due to carry-over correlation from early lags, since the PACF plot only shows a spike at certain lags.

There are significant auto correlations at lags 1 and 2 and beyond. Partial correlation plots show a significant spike at multiple lags. This suggests that we might want to test models with AR or MA components of an order associated with the lags.

4. Fitting an ARIMA Model:

Now let's fit a model. The forecast package allows the user to explicitly specify the order of the model using the arima() function, or automatically generate a set of optimal (p, d, q) using auto.arima(). This function searches through combinations of order parameters and picks the set that optimizes model fit criteria. While auto.arima() can be very useful, it is still important to complete the steps above in order to understand the series and interpret model results.

```
arima.nee1 <-auto.arima(nee, trace=TRUE)
```

```
##
##  Fitting models using approximations to speed things up...
##
##  ARIMA(2,1,2)(1,0,1)[30] with drift         : 692.3276
##  ARIMA(0,1,0)            with drift         : 812.1839
##  ARIMA(1,1,0)(1,0,0)[30] with drift         : 754.1
##  ARIMA(0,1,1)(0,0,1)[30] with drift         : 716.6018
##  ARIMA(0,1,0)                               : 810.171
##  ARIMA(2,1,2)(0,0,1)[30] with drift         : 700.2522
##  ARIMA(2,1,2)(1,0,0)[30] with drift         : 690.6777
##  ARIMA(2,1,2)            with drift         : 699.6979
##  ARIMA(2,1,2)(2,0,0)[30] with drift         : 693.2668
##  ARIMA(2,1,2)(2,0,1)[30] with drift         : 683.2512
##  ARIMA(2,1,2)(2,0,2)[30] with drift         : 684.2198
##  ARIMA(2,1,2)(1,0,2)[30] with drift         : 693.9678
##  ARIMA(1,1,2)(2,0,1)[30] with drift         : 687.2495
##  ARIMA(2,1,1)(2,0,1)[30] with drift         : 688.6439
##  ARIMA(3,1,2)(2,0,1)[30] with drift         : 684.767
##  ARIMA(2,1,3)(2,0,1)[30] with drift         : 680.7186
##  ARIMA(2,1,3)(1,0,1)[30] with drift         : 691.8705
##  ARIMA(2,1,3)(2,0,0)[30] with drift         : 693.7771
##  ARIMA(2,1,3)(2,0,2)[30] with drift         : 680.1553
##  ARIMA(2,1,3)(1,0,2)[30] with drift         : Inf
##  ARIMA(1,1,3)(2,0,2)[30] with drift         : 685.9808
##  ARIMA(3,1,3)(2,0,2)[30] with drift         : 687.4705
##  ARIMA(2,1,4)(2,0,2)[30] with drift         : 682.0784
##  ARIMA(1,1,2)(2,0,2)[30] with drift         : 688.6414
##  ARIMA(1,1,4)(2,0,2)[30] with drift         : 687.389
##  ARIMA(3,1,2)(2,0,2)[30] with drift         : Inf
##  ARIMA(3,1,4)(2,0,2)[30] with drift         : Inf
##  ARIMA(2,1,3)(2,0,2)[30]                    : 679.2322
##  ARIMA(2,1,3)(1,0,2)[30]                    : 691.5612
##  ARIMA(2,1,3)(2,0,1)[30]                    : 679.7025
##  ARIMA(2,1,3)(1,0,1)[30]                    : 689.9659
##  ARIMA(1,1,3)(2,0,2)[30]                    : Inf
##  ARIMA(2,1,2)(2,0,2)[30]                    : 682.9001
##  ARIMA(3,1,3)(2,0,2)[30]                    : 686.0237
##  ARIMA(2,1,4)(2,0,2)[30]                    : 681.2665
##  ARIMA(1,1,2)(2,0,2)[30]                    : 688.0739
##  ARIMA(1,1,4)(2,0,2)[30]                    : 686.5791
##  ARIMA(3,1,2)(2,0,2)[30]                    : 684.2254
##  ARIMA(3,1,4)(2,0,2)[30]                    : 688.1532
##
##  Now re-fitting the best model(s) without approximations...
```

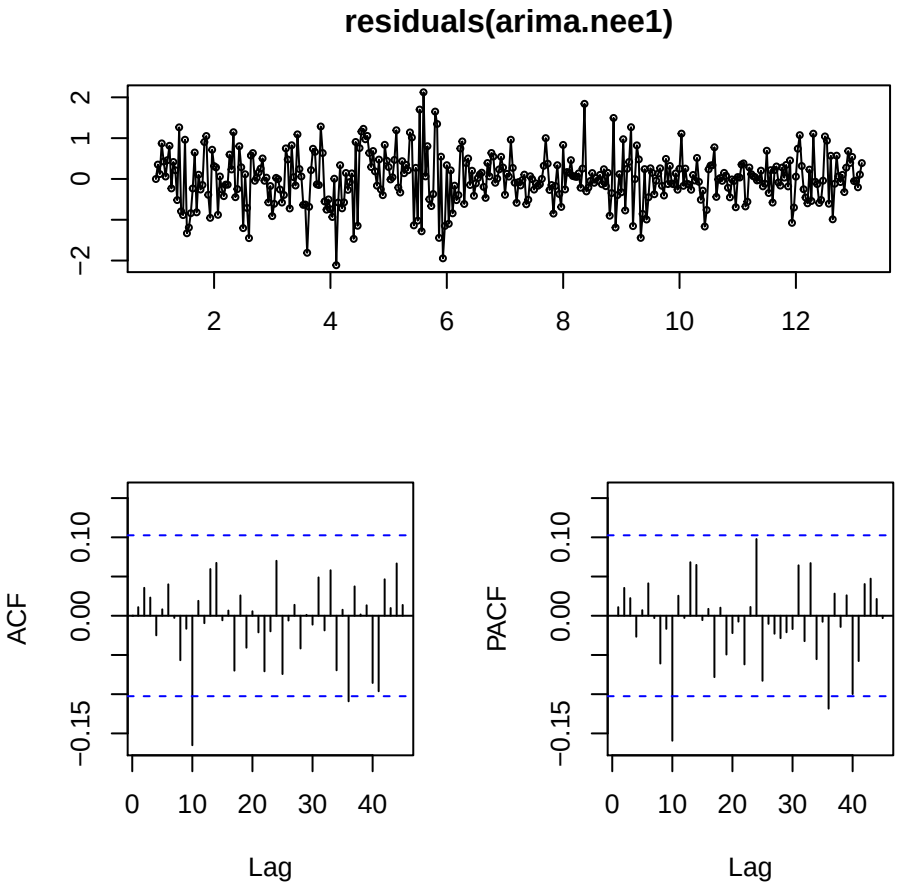
```
##
##  ARIMA(2,1,3)(2,0,2)[30]                    : 706.7137
##
##  Best model: ARIMA(2,1,3)(2,0,2)[30]
```

So now we have fitted a model, but does it make sense? Can we trust this model? We can start by examining ACF and PACF plots for model residuals. If model order parameters and structure are correctly specified, we would expect no significant autocorrelations present.

Ideally, residuals should look like white noise, meaning they are normally distributed. A convenience function tsdisplay() can be used to plot these model diagnostics.

Residuals plots show a smaller error range, more or less centered around 0. We can observe that AIC is smaller for the second model structure as well:

```
tsdisplay(residuals(arima.nee1), lag.max=45)
```



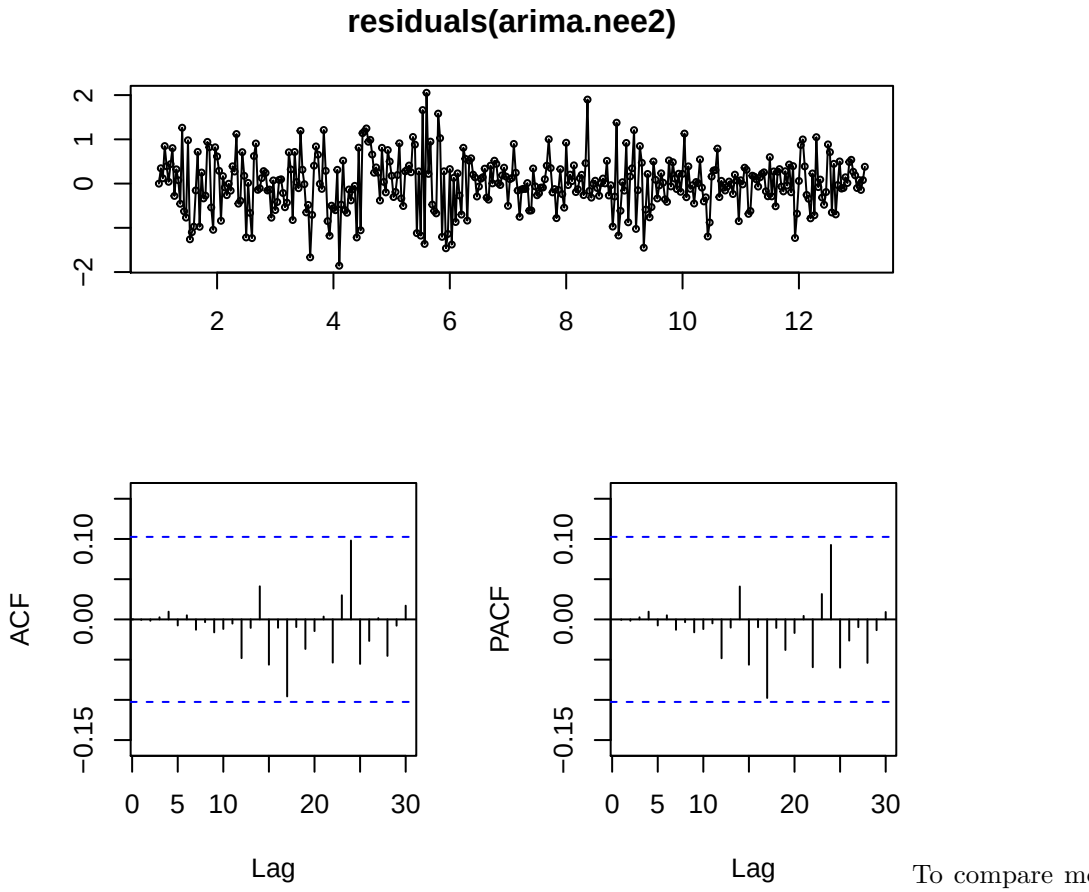
There is a clear pattern present in ACF/PACF and model residuals plots repeating at lag 10. This suggests that our model may be better off with a different specification, such as p = 10 or q = 10.

We can repeat the fitting process allowing for the MA(10 or 35) component and examine diagnostic plots again. This time, there are no significant autocorrelations present. If the model is not correctly specified, that will usually be reflected in residuals in the form of trends, skewedness, or any other patterns not captured by the model.

```
arima.nee2 <-arima(nee , order=c(10,1,3), seasonal= list(order=c(2,0,2)))
```

```
## Warning in arima(nee, order = c(10, 1, 3), seasonal = list(order = c(2, :
## possible convergence problem: optim gave code = 1
```

```
tsdisplay(residuals(arima.nee2), lag.max= 30)
```

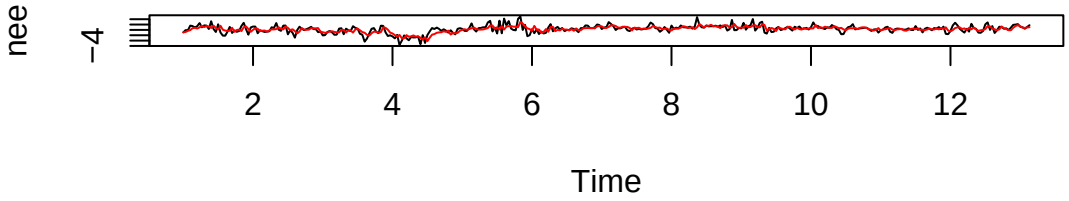


To compare models, you use the AIC. You also want to compare observed versus predicted values.

```
 # You want to minimize AIC

AIC(arima.nee1, arima.nee2)
```

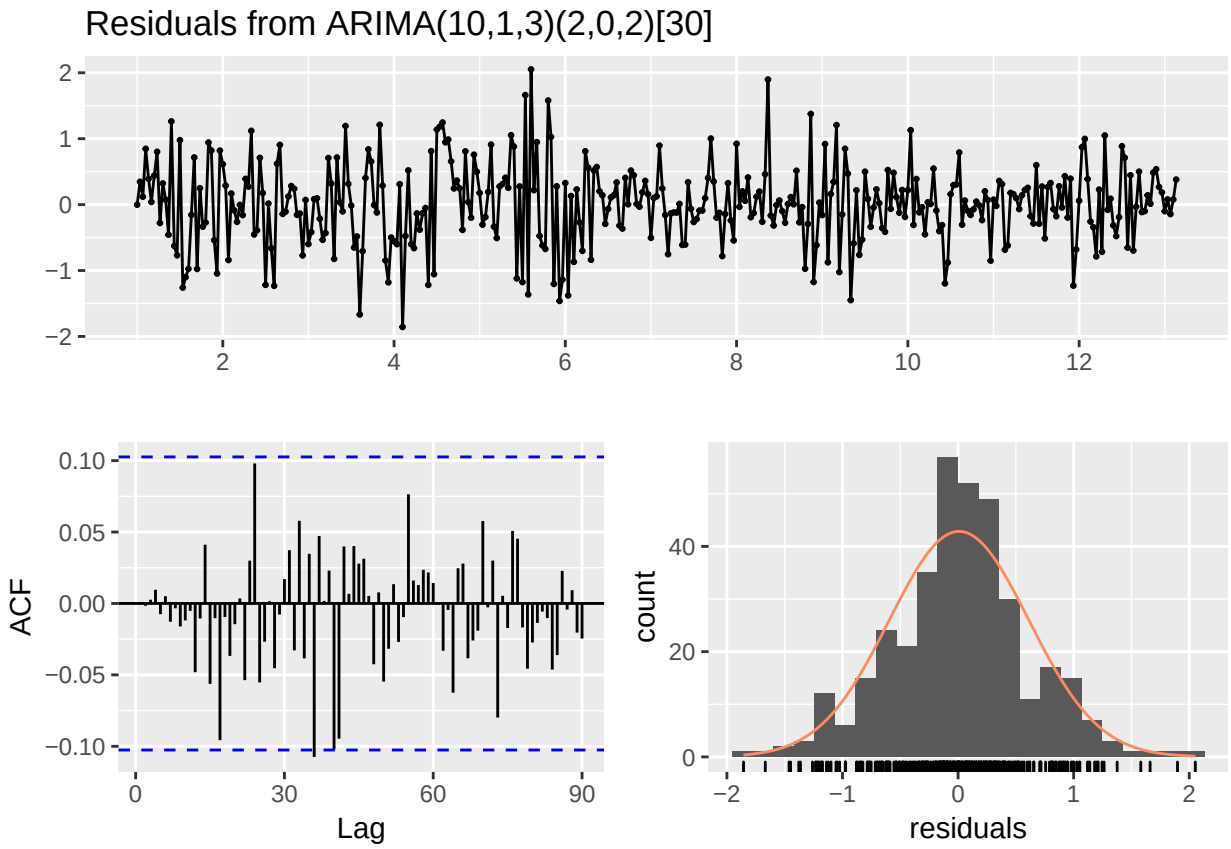
```
##             df      AIC
## arima.nee1 10 706.0904
## arima.nee2 18 704.7663
```

```
par(mfrow=c(1,1))

plot(nee , typ="l"); lines(fitted(arima.nee2),col="red")
```



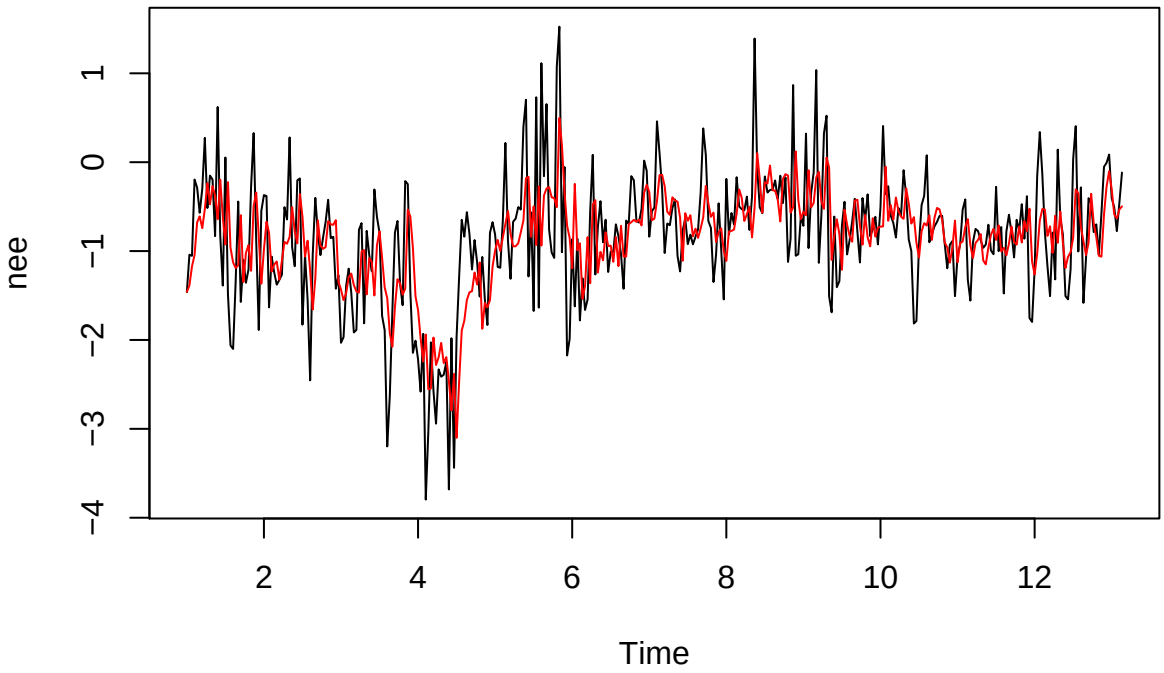
Next, we just need test for independence. The Ljung-Box is a test of independence at all lags up to the one specified. Instead of testing randomness at each distinct lag, it tests the "overall" randomness based on a number of lags, and is therefore a portmanteau test. It is applied to the residuals of a fitted ARIMA model, not the original series, and in such applications the hypothesis actually being tested is that the residuals from the ARIMA model have no autocorrelation.

```
# Measuring for significant difference from white noise.
# You need a p-value greater than 0.05!

checkresiduals(arima.nee2, lag=36)
```



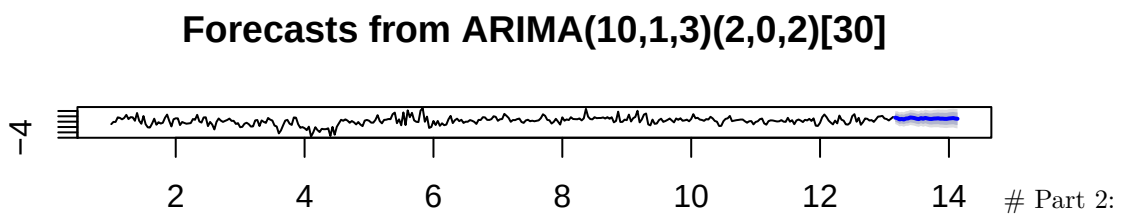

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(10,1,3)(2,0,2)[30]
## Q* = 23.111, df = 19, p-value = 0.2325
##
## Model df: 17.   Total lags used: 36
```

```
par(mfrow=c(1,1))
plot(nee , typ="l"); lines(fitted(arima.nee2),col="red")
```



Now that you have captured the temporal dynamics in nee, the next step maybe to forecast NEE if you are interested in making projections. You can do this using the *forecast function. For fun, lets forecast nee for 30 days.

```
plot(forecast(arima.nee2, h=30))
```



Part 2:

If you are interested in evaluating potential drivers of nee, you can prewritten potential explanatory variables you want to explore.

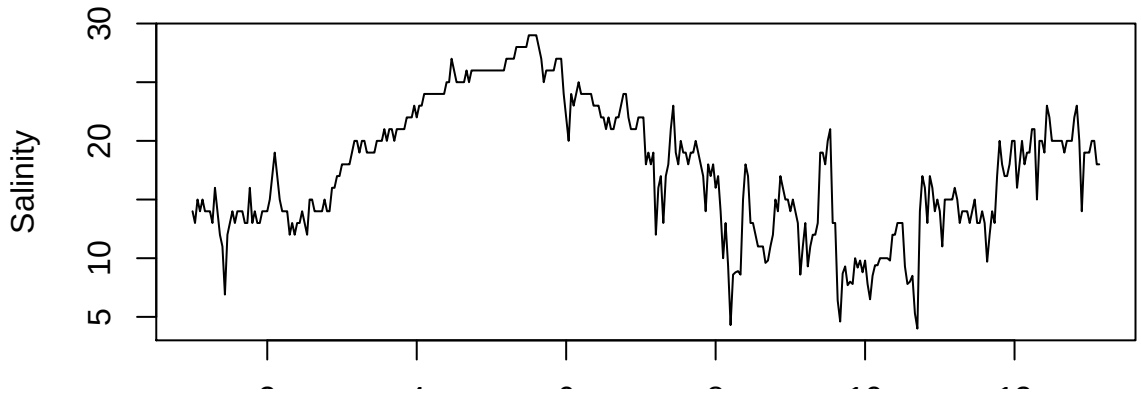
1. Create timeseries objects.
2. Decompose the time series.
3. Test for stationarity and detecting autocorrelation.
4. Explore correlations.
5. Compare models with and without explanatory drivers.

1. Create a timeseries object.

```
sal <- ts(mangroves$salinity.max, start= 1, frequency=30)
```

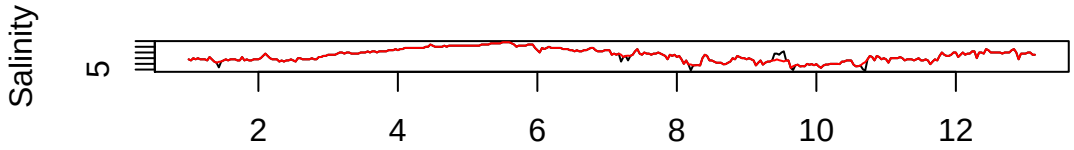
Visualize data:

```
par(mfrow=c(1,1), mai=c(0.25,0.8,0.1, 0.1))
plot(sal , typ="l", ylab= "Salinity", xlab="")
```



You want to remove any outliers that could bias the model by skewing statistical summaries.

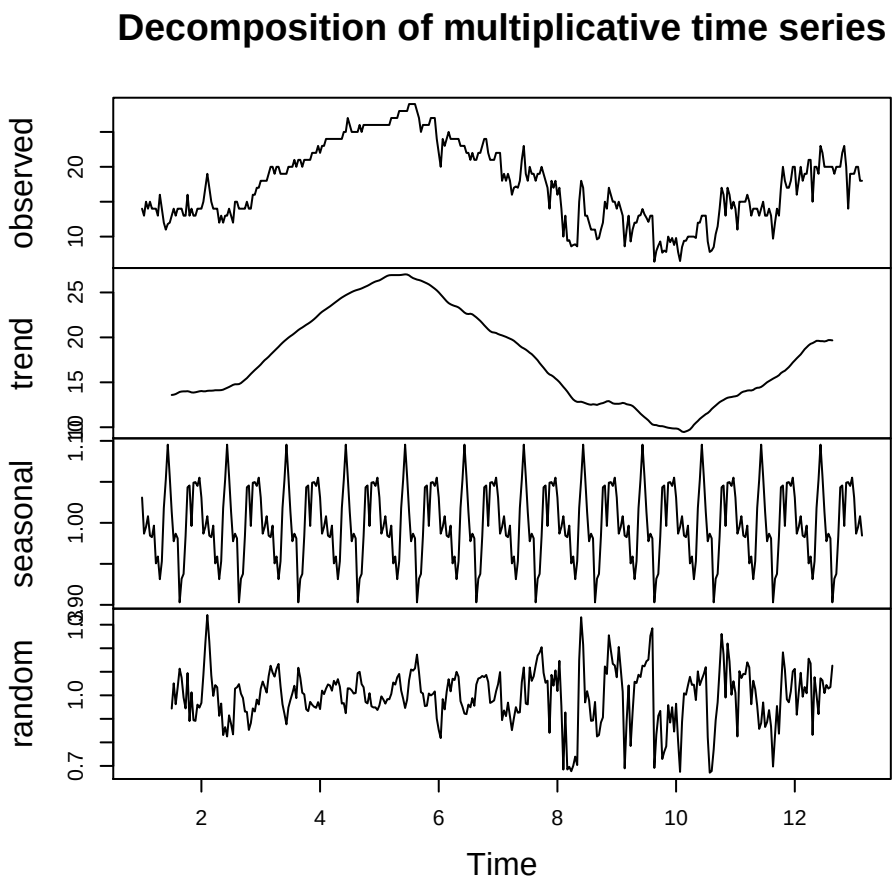
```
plot(sal , typ="l", ylab= "Salinity", xlab="")
lines(tsclean(sal) , col="red")
```




```
sal <- tsclean(sal)
```

2. Decompose the time series.

```
sal.d <- decompose(sal, 'multiplicative')
plot(sal.d)
```



3.a. Test for stationarity

If your null hypothesis is not rejected, you can try differencing the time series using the function *diff()*

```
# p-value < 0.05 indicates the TS is stationary

adf.test(sal)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  sal
## Dickey-Fuller = -1.7132, Lag order = 7, p-value = 0.698
## alternative hypothesis: stationary
```

```
adf.test(diff(sal))
```

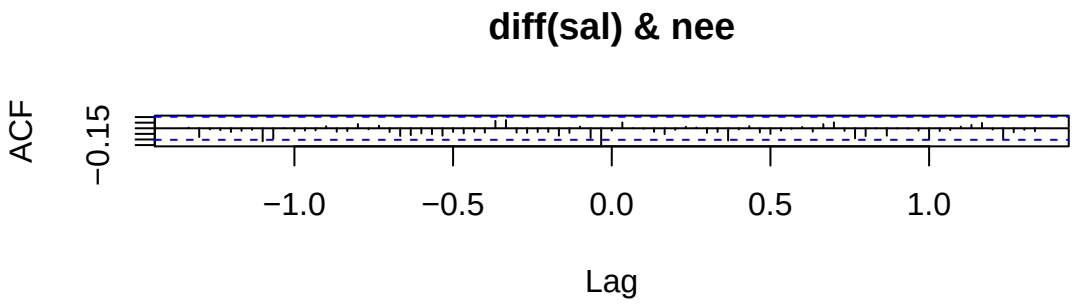
```
## Warning in adf.test(diff(sal)): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff(sal)
## Dickey-Fuller = -8.9764, Lag order = 7, p-value = 0.01
## alternative hypothesis: stationary
```

4. Explore correlations.

Look for significant lags.

```
ccf( diff(sal),nee, na.action = na.pass, lag.max=40, plot=TRUE)
```



5. Explore Models of NEE.

```
arima.nee3 <-auto.arima(nee, xreg=c(diff(sal),0), trace=TRUE)
```

```
##
##  Fitting models using approximations to speed things up...
##
##  Regression with ARIMA(2,1,2)(1,0,1)[30] errors : 690.6315
##  Regression with ARIMA(0,1,0)            errors : 811.6241
##  Regression with ARIMA(1,1,0)(1,0,0)[30] errors : 749.6281
##  Regression with ARIMA(0,1,1)(0,0,1)[30] errors : 714.2142
##  ARIMA(0,1,0)                               : 809.5996
##  Regression with ARIMA(2,1,2)(0,0,1)[30] errors : 699.309
##  Regression with ARIMA(2,1,2)(1,0,0)[30] errors : 689.2625
##  Regression with ARIMA(2,1,2)            errors : 698.7717
##  Regression with ARIMA(2,1,2)(2,0,0)[30] errors : 691.3713
##  Regression with ARIMA(2,1,2)(2,0,1)[30] errors : 681.362
##  Regression with ARIMA(2,1,2)(2,0,2)[30] errors : 682.4585
##  Regression with ARIMA(2,1,2)(1,0,2)[30] errors : 692.4389
##  Regression with ARIMA(1,1,2)(2,0,1)[30] errors : 684.9212
##  Regression with ARIMA(2,1,1)(2,0,1)[30] errors : 686.9106
##  Regression with ARIMA(3,1,2)(2,0,1)[30] errors : 682.9556
##  Regression with ARIMA(2,1,3)(2,0,1)[30] errors : 679.4947
##  Regression with ARIMA(2,1,3)(1,0,1)[30] errors : 690.3674
##  Regression with ARIMA(2,1,3)(2,0,0)[30] errors : Inf
##  Regression with ARIMA(2,1,3)(2,0,2)[30] errors : 678.9617
##  Regression with ARIMA(2,1,3)(1,0,2)[30] errors : 692.1447
##  Regression with ARIMA(1,1,3)(2,0,2)[30] errors : 684.2355
##  Regression with ARIMA(3,1,3)(2,0,2)[30] errors : 685.7004
##  Regression with ARIMA(2,1,4)(2,0,2)[30] errors : 680.5748
##  Regression with ARIMA(1,1,2)(2,0,2)[30] errors : Inf
##  Regression with ARIMA(1,1,4)(2,0,2)[30] errors : 685.5068
##  Regression with ARIMA(3,1,2)(2,0,2)[30] errors : 683.8335
##  Regression with ARIMA(3,1,4)(2,0,2)[30] errors : 687.8471
##  ARIMA(2,1,3)(2,0,2)[30]                    : 677.9777
##  ARIMA(2,1,3)(1,0,2)[30]                    : Inf
##  ARIMA(2,1,3)(2,0,1)[30]                    : 678.4147
##  ARIMA(2,1,3)(1,0,1)[30]                    : 688.496
##  ARIMA(1,1,3)(2,0,2)[30]                    : Inf
```

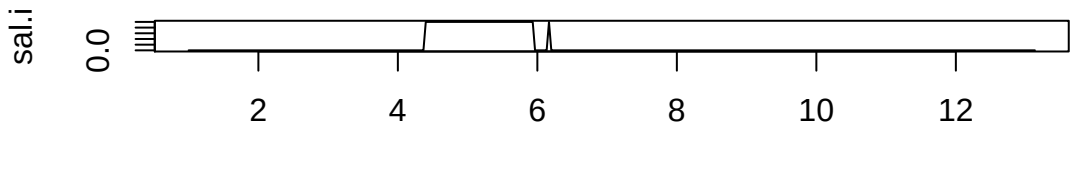
```
##  ARIMA(2,1,2)(2,0,2)[30]                    : 681.0913
##  ARIMA(3,1,3)(2,0,2)[30]                    : 684.3019
##  ARIMA(2,1,4)(2,0,2)[30]                    : 679.7665
##  ARIMA(1,1,2)(2,0,2)[30]                    : 685.7646
##  ARIMA(1,1,4)(2,0,2)[30]                    : 684.6679
##  ARIMA(3,1,2)(2,0,2)[30]                    : Inf
##  ARIMA(3,1,4)(2,0,2)[30]                    : 686.4496
##
##  Now re-fitting the best model(s) without approximations...
##
##  ARIMA(2,1,3)(2,0,2)[30]                    : 706.1645
##
##  Best model: Regression with ARIMA(2,1,3)(2,0,2)[30] errors
```

```
#compare to current model

AIC(arima.nee2, arima.nee3 )
```

```
##            df      AIC
## arima.nee2 18 704.7663
## arima.nee3 11 705.4145
```

So, adding salinity to nee did not improve the model. Maybe extreme salinity is more important. Lets create a salinity index to ID when salinity values are greater than 25n ppt.

```
sal.i <- sal
sal.i[sal.i < 25 ]<- 0
sal.i[sal.i >= 25 ]<- 1

plot(sal.i)
```



Now try adding the extreme salinity indicator into the model to see if this is an improvement:

```
arima.nee4 <-auto.arima(nee, xreg=sal.i, trace=TRUE)
```

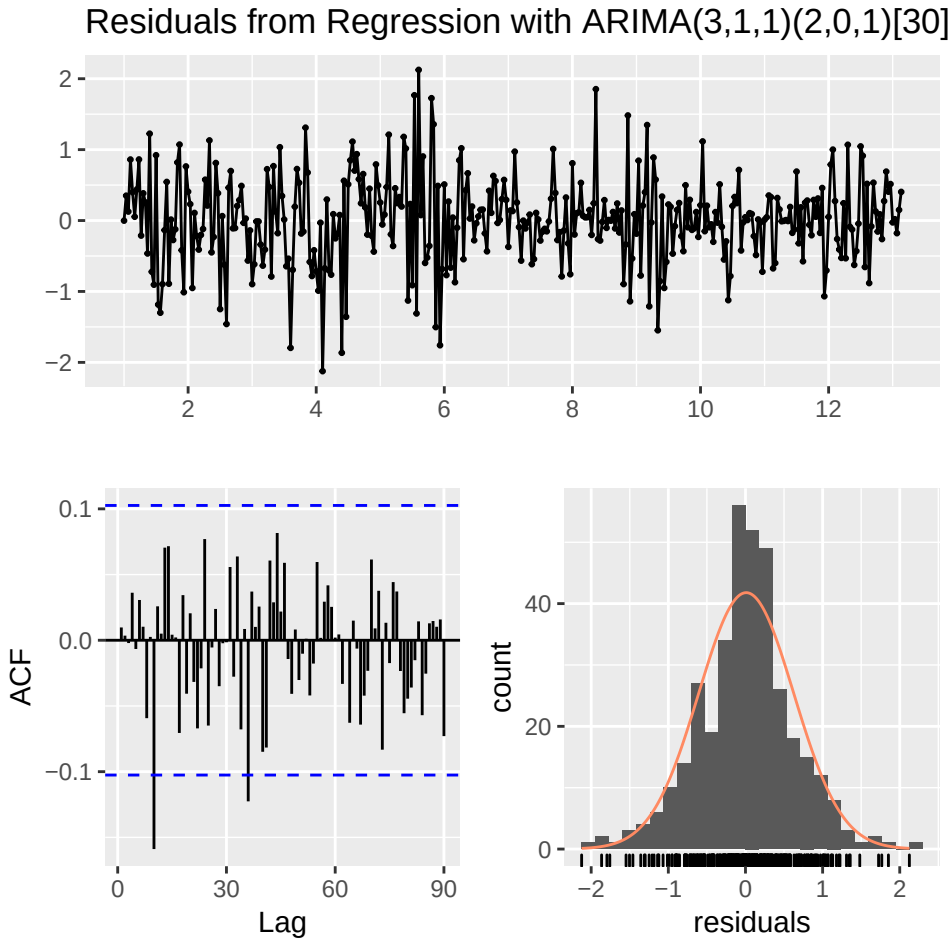
```
##
##  Fitting models using approximations to speed things up...
##
##  Regression with ARIMA(2,1,2)(1,0,1)[30] errors : 692.3299
##  Regression with ARIMA(0,1,0)            errors : 812.1719
##  Regression with ARIMA(1,1,0)(1,0,0)[30] errors : 754.9082
##  Regression with ARIMA(0,1,1)(0,0,1)[30] errors : 718.091
##  ARIMA(0,1,0)                               : 810.1477
##  Regression with ARIMA(2,1,2)(0,0,1)[30] errors : 700.5987
##  Regression with ARIMA(2,1,2)(1,0,0)[30] errors : 690.8439
##  Regression with ARIMA(2,1,2)            errors : 699.7661
##  Regression with ARIMA(2,1,2)(2,0,0)[30] errors : 693.383
##  Regression with ARIMA(2,1,2)(2,0,1)[30] errors : 682.688
##  Regression with ARIMA(2,1,2)(2,0,2)[30] errors : 683.7001
##  Regression with ARIMA(2,1,2)(1,0,2)[30] errors : 694.1394
##  Regression with ARIMA(1,1,2)(2,0,1)[30] errors : 686.7376
```

```
##  Regression with ARIMA(2,1,1)(2,0,1)[30] errors : Inf
##  Regression with ARIMA(3,1,2)(2,0,1)[30] errors : 681.9797
##  Regression with ARIMA(3,1,2)(1,0,1)[30] errors : 692.0179
##  Regression with ARIMA(3,1,2)(2,0,0)[30] errors : 693.8084
##  Regression with ARIMA(3,1,2)(2,0,2)[30] errors : 682.7728
##  Regression with ARIMA(3,1,2)(1,0,0)[30] errors : 689.9044
##  Regression with ARIMA(3,1,2)(1,0,2)[30] errors : 693.959
##  Regression with ARIMA(3,1,1)(2,0,1)[30] errors : 680.032
##  Regression with ARIMA(3,1,1)(1,0,1)[30] errors : 690.809
##  Regression with ARIMA(3,1,1)(2,0,0)[30] errors : 692.2766
##  Regression with ARIMA(3,1,1)(2,0,2)[30] errors : Inf
##  Regression with ARIMA(3,1,1)(1,0,0)[30] errors : 688.7484
##  Regression with ARIMA(3,1,1)(1,0,2)[30] errors : 692.6679
##  Regression with ARIMA(3,1,0)(2,0,1)[30] errors : 717.1632
##  Regression with ARIMA(4,1,1)(2,0,1)[30] errors : 682.9725
##  Regression with ARIMA(2,1,0)(2,0,1)[30] errors : 740.972
##  Regression with ARIMA(4,1,0)(2,0,1)[30] errors : 708.9094
##  Regression with ARIMA(4,1,2)(2,0,1)[30] errors : 684.9766
##  ARIMA(3,1,1)(2,0,1)[30]                     : 679.0062
##  ARIMA(3,1,1)(1,0,1)[30]                     : 688.7987
##  ARIMA(3,1,1)(2,0,0)[30]                     : 690.5547
##  ARIMA(3,1,1)(2,0,2)[30]                     : 679.765
##  ARIMA(3,1,1)(1,0,0)[30]                     : 686.7338
##  ARIMA(3,1,1)(1,0,2)[30]                     : 690.6468
##  ARIMA(2,1,1)(2,0,1)[30]                     : 687.1294
##  ARIMA(3,1,0)(2,0,1)[30]                     : 715.2093
##  ARIMA(4,1,1)(2,0,1)[30]                     : 682.2748
##  ARIMA(3,1,2)(2,0,1)[30]                     : 680.869
##  ARIMA(2,1,0)(2,0,1)[30]                     : 738.9596
##  ARIMA(2,1,2)(2,0,1)[30]                     : 681.6579
##  ARIMA(4,1,0)(2,0,1)[30]                     : 707.0037
##  ARIMA(4,1,2)(2,0,1)[30]                     : 684.1723
##
##  Now re-fitting the best model(s) without approximations...
##
##  ARIMA(3,1,1)(2,0,1)[30]                     : 701.2818
##
##  Best model: Regression with ARIMA(3,1,1)(2,0,1)[30] errors
```

```
AIC(arima.nee2,arima.nee4 )
```

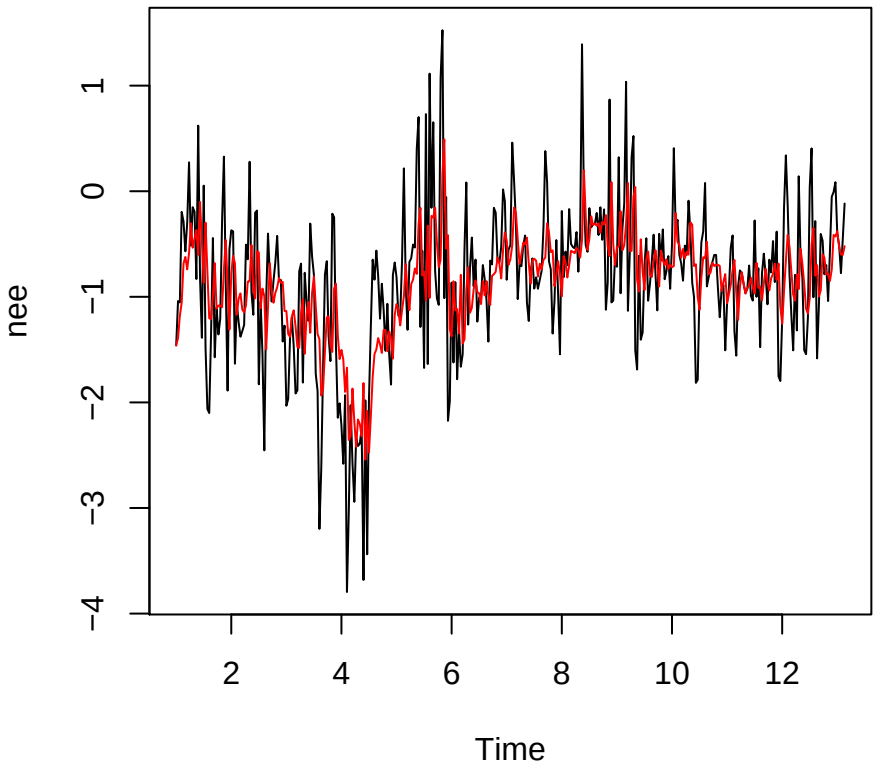
```
##            df      AIC
## arima.nee2 18 704.7663
## arima.nee4  9 700.7734
```

```
checkresiduals(arima.nee4, lag=36)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(3,1,1)(2,0,1)[30] errors
## Q* = 37.32, df = 28, p-value = 0.112
##
## Model df: 8.   Total lags used: 36
```

```
par(mfrow=c(1,1))
plot(nee , typ="l"); lines(fitted(arima.nee4),col="red")
```



There are other potential explanatory series. Can you develop a better model?