Time Series

Julius Ongteco

5/7/2021

Libraries

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(fpp2)
```

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

## -- Attaching packages ---------------------------------------------- fpp2 2.4 --

## v ggplot2   3.3.3     v fma       2.4
## v forecast  8.14      v expsmooth 2.3

##
```

```
library(ggplot2)
library(openxlsx)
library(readxl)
```

Initialize

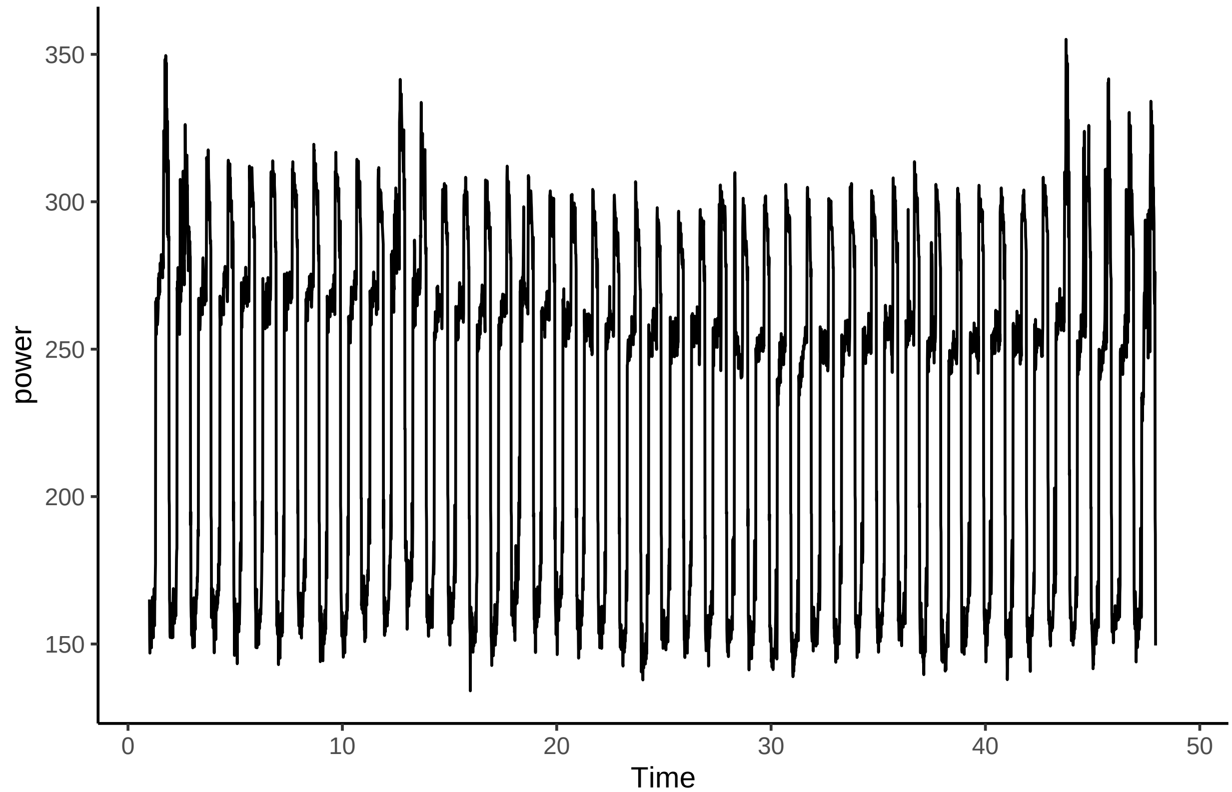
```r
# aesthetic
theme_set(theme_classic())

# read file
e <- read_xlsx("Elec-train.xlsx")
e
```

```
## # A tibble: 4,603 x 3
##    Timestamp     `Power (kW)` `Temp (C°)`
##    <chr>                <dbl>       <dbl>
##  1 1/1/2010 1:15         165.        10.6
##  2 1/1/2010 1:30         152.        10.6
##  3 1/1/2010 1:45         147.        10.6
##  4 1/1/2010 2:00         154.        10.6
##  5 1/1/2010 2:15         154.        10.6
##  6 1/1/2010 2:30         159         10.6
##  7 1/1/2010 2:45         158.        10.6
##  8 1/1/2010 3:00         163.        10.6
##  9 1/1/2010 3:15         152.        10
## 10 1/1/2010 3:30         149.        10
## # ... with 4,593 more rows
```

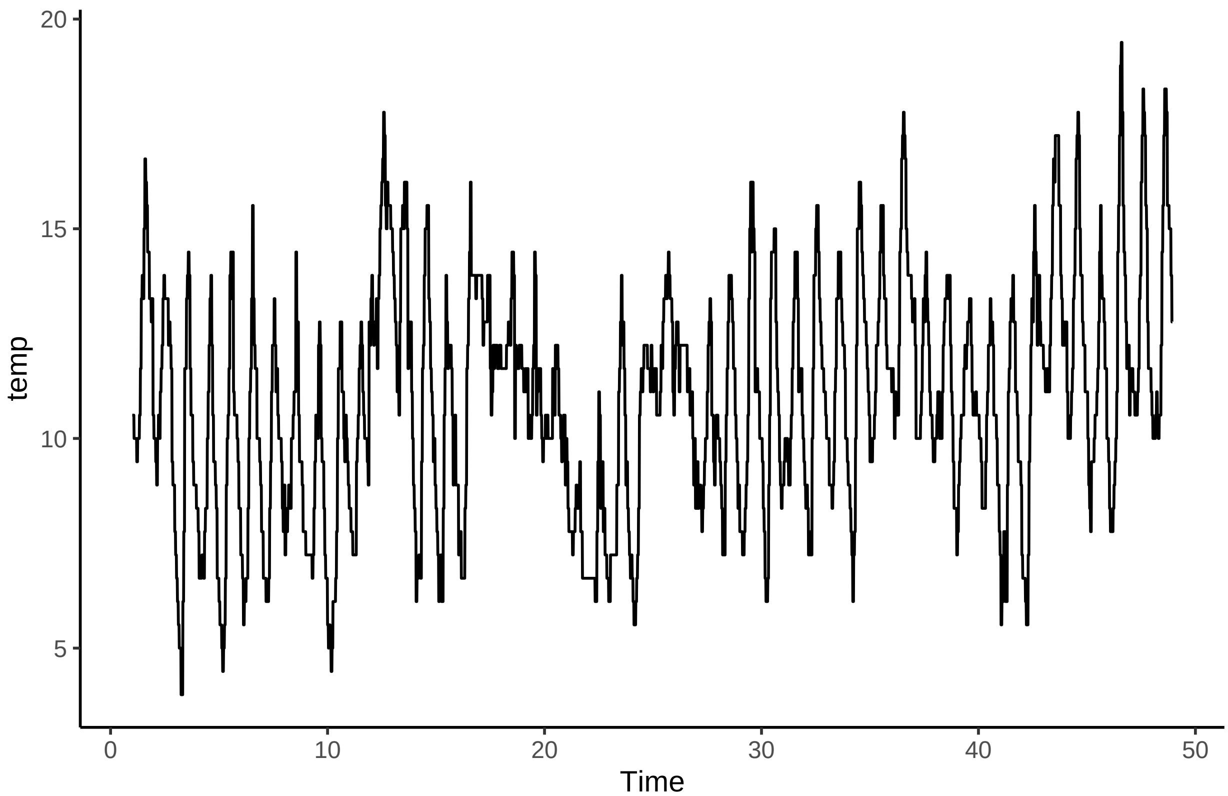
```r
# renamed columns
names(e) <- c('date', 'power', 'temp')

# power.
power <- ts(e$power, freq=96)

# temp.
temp <- ts(e$temp, freq=96)

# date.
date <- as.POSIXct(e$date,format="%m/%d/%Y %H:%M",tz=Sys.timezone())

# split data sets into one that needs to be predicted and one having values of e.
predict_e <- e[!complete.cases(e),]
values_of_e <- e[complete.cases(e),]

# predict temperature.
pred_temp <- ts(predict_e$temp, start = c(1,nrow(values_of_e)+1), end=c(1,nrow(e)), frequency=96)
```

Join our variables.

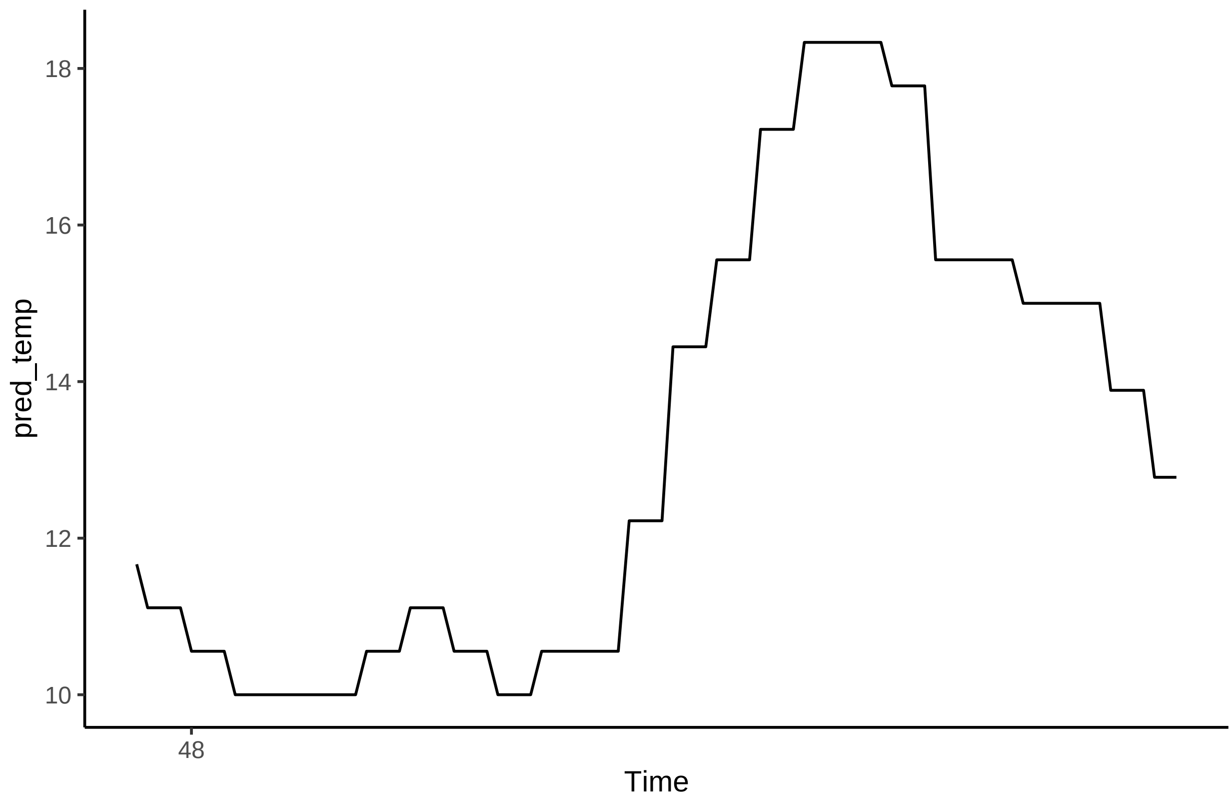
```r
elec <- ts.union(power, temp, pred_temp)

autoplot(power)
```




```
autoplot(temp)
```

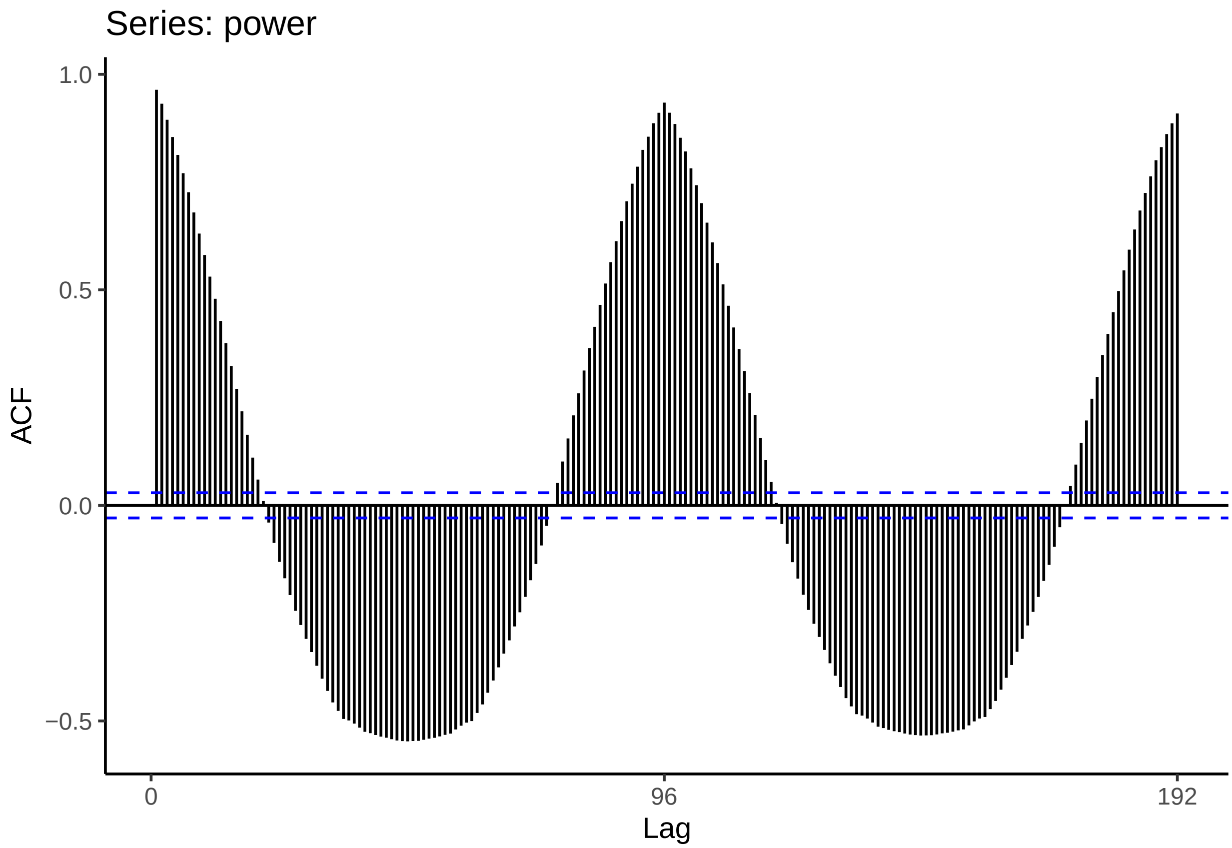



```
autoplot(pred_temp)
```

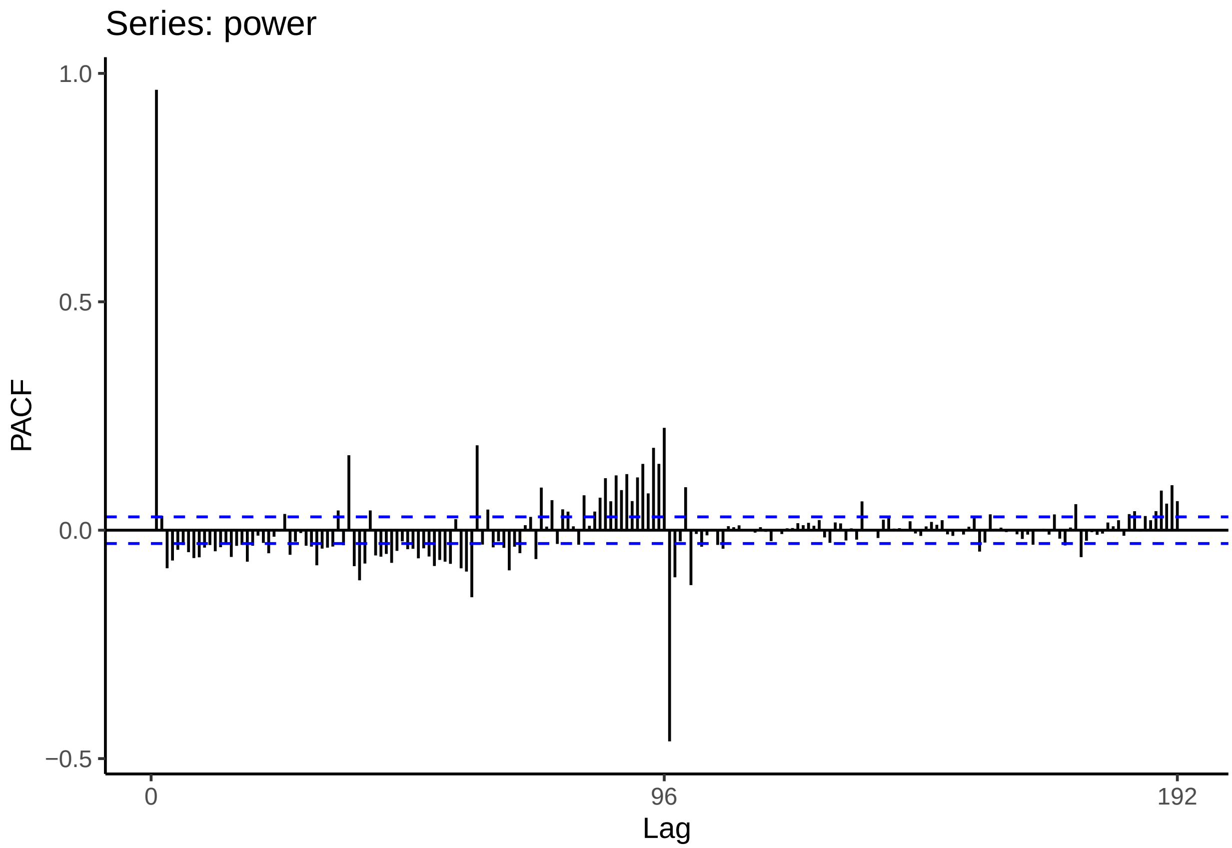


Power graph indicates seasonality but no apparent trend. Temp graph also shows seasonality which possibly could be cyclic. let's confirm out of curiosity using an acf or pacf plot. I'm personally just enjoying this.

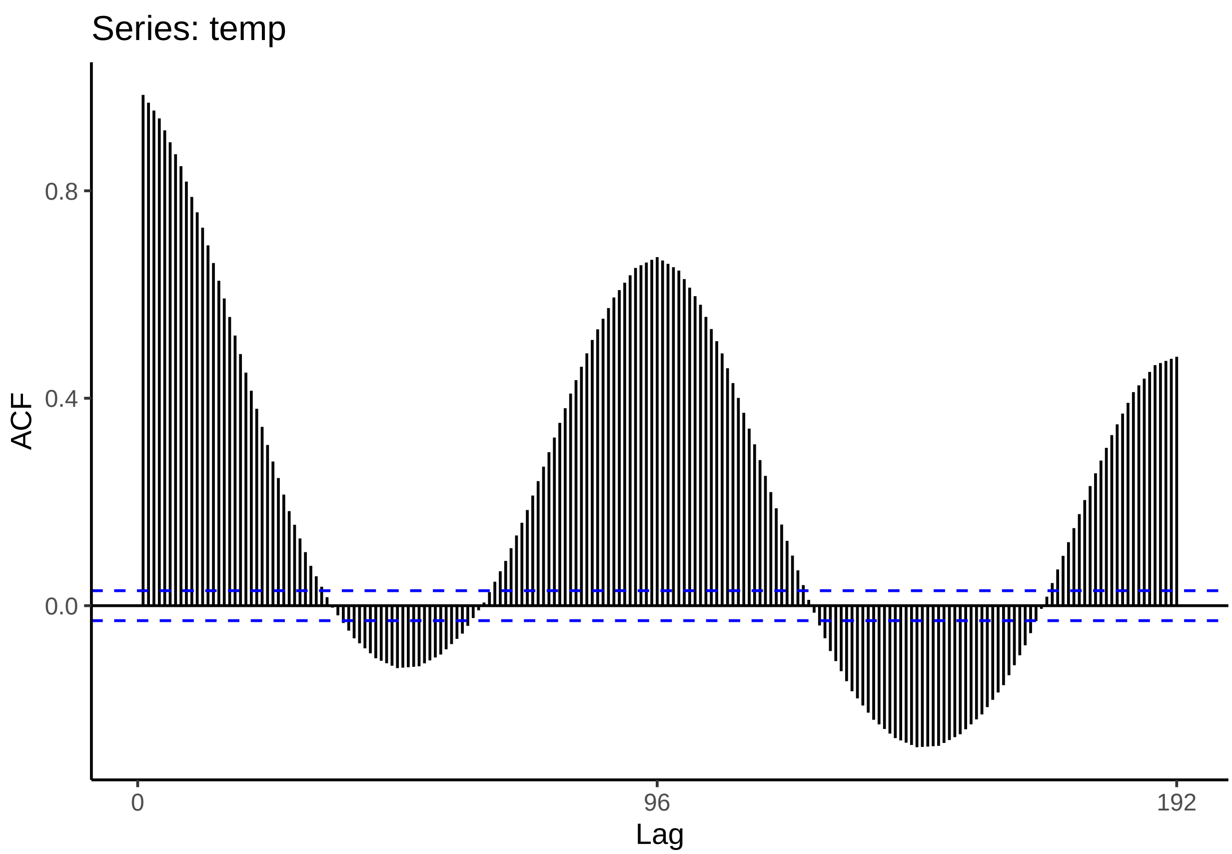
```
ggAcf(power)
```



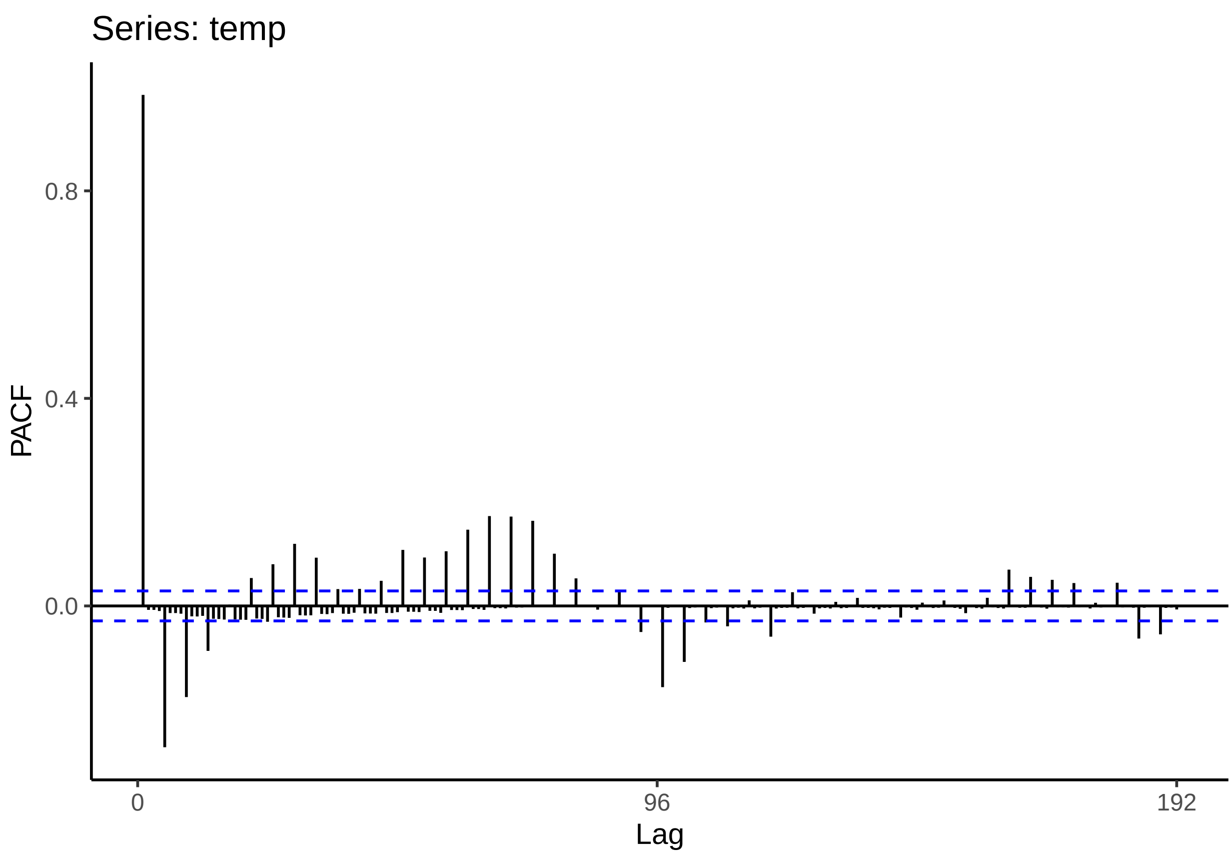

```
ggPacf(power)
```




```
ggAcf(temp)
```




```
ggPacf(temp)
```



autocorrelations are statistically significantly different from zero. which indicates seasonality. probably means this can possibly not be a stationary time series.

```
Box.test(power,lag=10,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  power
## X-squared = 28470, df = 10, p-value < 2.2e-16
```

```
Box.test(temp,lag=10,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  temp
## X-squared = 37388, df = 10, p-value < 2.2e-16
```

There are significant differences in both variables as p value is less than 0.05.

Train Test Split.

```
e_train <- window(elec, start=c(1,1),end=c(1,nrow(values_of_e)-96))
e_test <- window(elec, start=c(1,nrow(values_of_e)-95),end=c(1,nrow(values_of_e)))
```

Forecasting without Temperature

Test the Models!

Simple Exponential Smoothing.

```
SES=ses(e_train[,"power"],alpha=NULL,beta=NULL,gamma=NULL)
SES_f<-predict(SES,n.ahead=96)
```

Base Holt Winters with Nonseasonal HW Smoothing.

```
LES=HoltWinters(e_train[,"power"],alpha=NULL,beta=NULL,gamma=FALSE)
LES_f<-predict(LES,n.ahead=96)
```

Holt "Damped"

```
LES_damped=HoltWinters(e_train[,"power"],alpha=NULL,beta=FALSE,gamma=FALSE)
LES_damped_f<-predict(LES_damped,damped=TRUE, phi=0.9, n.ahead=96)
```

Holt additive

```
SES_additive=HoltWinters(e_train[,"power"],alpha=NULL,beta=NULL,gamma=NULL,seasonal = 'additive' )
SES_add<-predict(SES_additive, n.ahead=96)
```

Holt Multiplicative

```
SES_multi=HoltWinters(e_train[,"power"],alpha=NULL,beta=NULL,gamma=NULL,seasonal = 'multi' )
SES_m<-predict(SES_multi, n.ahead=96)
```

Arima (Example)

```
ar = Arima(e_train[,"power"], order=c(0,0,0))
ar_f = forecast(ar,h=96)
```

Auto Arima Optimal

```
aa = auto.arima(e_train[,"power"], seasonal=FALSE)
aa_f = forecast(aa,h=96)
```

AA to make it work a little harder.

```
aa2 <- auto.arima(e_train[,"power"], seasonal=FALSE,
  stepwise=FALSE, approximation=FALSE)
aa2_f = forecast(aa2,h=96)
```

SARIMA

```
sarima = auto.arima(e_train[,"power"])
sarima_f = forecast(sarima,h=96)
```

Neural Network

```
nn = nnetar(e_train[,"power"],h=96)
nn_f = forecast(nn, h=96)
```

RSME

```
print(paste0("RMSE (SES): ",sqrt(mean((SES_f$mean-e_test[,"power"])^2))))
```

```
## [1] "RMSE (SES): 13.0336300725396"
```

```
print(paste0("RMSE (HW): ",sqrt(mean((LES_f-e_test[,"power"])^2))))
```

```
## [1] "RMSE (HW): 138.667761014417"
```

```
print(paste0("RMSE (HW damped): ",sqrt(mean((LES_damped_f-e_test[,"power"])^2))))
```

```
## [1] "RMSE (HW damped): 88.8847800356967"
```

```
print(paste0("RMSE (HW add): ",sqrt(mean((SES_add-e_test[,"power"])^2))))
```

```
## [1] "RMSE (HW add): 16.865425974985"
```

```
print(paste0("RMSE (HW multi): ",sqrt(mean((SES_m-e_test[,"power"])^2))))
```

```
## [1] "RMSE (HW multi): 13.9237614756714"
```

```
print(paste0("RMSE (Arima): ",sqrt(mean((ar_f$mean-e_test[,"power"])^2))))
```

```
## [1] "RMSE (Arima): 60.8416061105818"
```

```
print(paste0("RMSE (Auto Arima): ",sqrt(mean((aa_f$mean-e_test[,"power"])^2))))
```

```
## [1] "RMSE (Auto Arima): 37.7158299627763"
```

```
print(paste0("RMSE (Auto Arima deep): ",sqrt(mean((aa2_f$mean-e_test[,"power"])^2))))
```

```
## [1] "RMSE (Auto Arima deep): 37.5620762288198"
```

```
print(paste0("RMSE (Sarima): ",sqrt(mean((sarima_f$mean-e_test[,"power"])^2))))
```

```
## [1] "RMSE (Sarima): 19.7890715067741"
```

```
print(paste0("RMSE (Neural Network): ",sqrt(mean((nn_f$mean-e_test[,"power"])^2))))
```

```
## [1] "RMSE (Neural Network): 18.0480494609491"
```

Based on the results, SES seems to outperform the other models. Going to be using this for predicting values.

Forecasting with Temperature

We want to be able to predict Power using other time series models. We add seasonality and trend into the mix.

```
lm=tslm(power~temp+season+trend,data=e_train)
summary(lm)
```

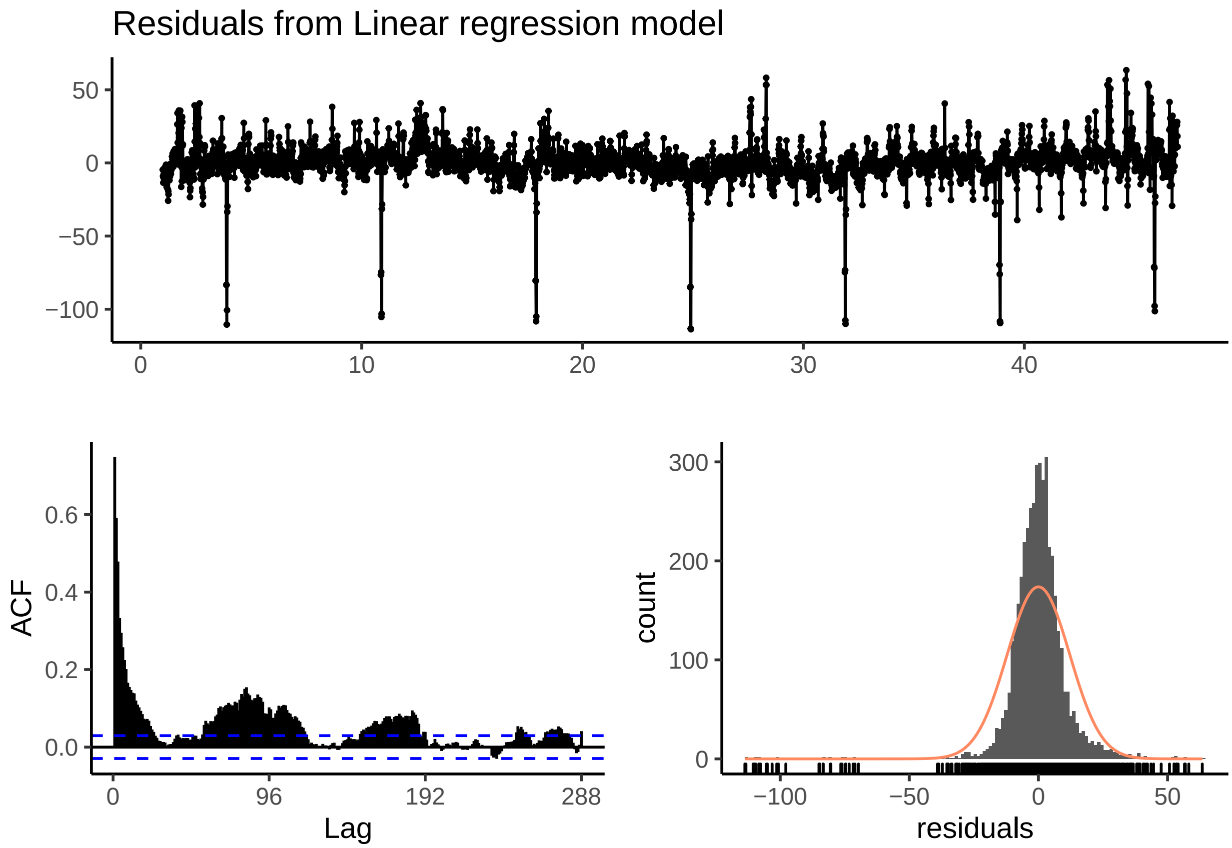
```
##
## Call:
## tslm(formula = power ~ temp + season + trend, data = e_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -113.691   -4.999    0.046    4.847   63.435
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.563e+02  2.006e+00  77.888  < 2e-16 ***
## temp         1.278e+00  9.568e-02  13.356  < 2e-16 ***
## season2     -9.151e+00  2.555e+00  -3.581 0.000346 ***
## season3     -9.314e+00  2.555e+00  -3.645 0.000271 ***
## season4     -5.991e+00  2.555e+00  -2.344 0.019101 *
## season5     -6.371e+00  2.556e+00  -2.493 0.012704 *
## season6     -3.587e+00  2.556e+00  -1.404 0.160529
## season7     -4.620e+00  2.556e+00  -1.808 0.070697 .
## season8     -3.114e+00  2.556e+00  -1.219 0.223063
## season9     -7.453e+00  2.556e+00  -2.916 0.003563 **
## season10    -8.834e+00  2.556e+00  -3.456 0.000553 ***
## season11    -3.300e+00  2.556e+00  -1.291 0.196717
## season12    -1.638e+00  2.556e+00  -0.641 0.521728
## season13    -2.429e+00  2.557e+00  -0.950 0.342155
## season14    -3.345e+00  2.557e+00  -1.308 0.190852
## season15    -1.763e+00  2.557e+00  -0.689 0.490562
## season16    -2.115e+00  2.557e+00  -0.827 0.408009
## season17    -1.148e+00  2.557e+00  -0.449 0.653386
## season18    -3.316e-01  2.557e+00  -0.130 0.896828
## season19     1.687e+00  2.557e+00   0.660 0.509357
## season20     1.211e+00  2.557e+00   0.473 0.635917
## season21     4.666e+00  2.557e+00   1.825 0.068071 .
## season22     1.229e+01  2.557e+00   4.805 1.60e-06 ***
## season23     1.359e+01  2.557e+00   5.316 1.11e-07 ***
## season24     1.383e+01  2.557e+00   5.410 6.63e-08 ***
## season25     1.887e+01  2.557e+00   7.379 1.90e-13 ***
```

```
## season26      1.766e+01  2.557e+00   6.909 5.61e-12 ***
## season27      1.323e+01  2.557e+00   5.174 2.40e-07 ***
## season28      1.693e+01  2.557e+00   6.622 3.99e-11 ***
## season29      1.007e+02  2.556e+00  39.385  < 2e-16 ***
## season30      9.851e+01  2.556e+00  38.533  < 2e-16 ***
## season31      9.591e+01  2.556e+00  37.515  < 2e-16 ***
## season32      9.534e+01  2.556e+00  37.293  < 2e-16 ***
## season33      9.837e+01  2.556e+00  38.490  < 2e-16 ***
## season34      9.304e+01  2.556e+00  36.403  < 2e-16 ***
## season35      9.517e+01  2.556e+00  37.240  < 2e-16 ***
## season36      9.608e+01  2.556e+00  37.593  < 2e-16 ***
## season37      9.236e+01  2.559e+00  36.088  < 2e-16 ***
## season38      9.326e+01  2.559e+00  36.441  < 2e-16 ***
## season39      9.463e+01  2.559e+00  36.977  < 2e-16 ***
## season40      9.499e+01  2.559e+00  37.119  < 2e-16 ***
## season41      9.668e+01  2.566e+00  37.674  < 2e-16 ***
## season42      9.480e+01  2.566e+00  36.942  < 2e-16 ***
## season43      9.549e+01  2.566e+00  37.210  < 2e-16 ***
## season44      9.611e+01  2.566e+00  37.454  < 2e-16 ***
## season45      9.570e+01  2.576e+00  37.155  < 2e-16 ***
## season46      9.905e+01  2.576e+00  38.458  < 2e-16 ***
## season47      9.800e+01  2.576e+00  38.050  < 2e-16 ***
## season48      9.640e+01  2.576e+00  37.427  < 2e-16 ***
## season49      9.789e+01  2.582e+00  37.909  < 2e-16 ***
## season50      9.815e+01  2.582e+00  38.010  < 2e-16 ***
## season51      9.785e+01  2.582e+00  37.894  < 2e-16 ***
## season52      9.657e+01  2.582e+00  37.399  < 2e-16 ***
## season53      9.680e+01  2.590e+00  37.378  < 2e-16 ***
## season54      9.740e+01  2.590e+00  37.611  < 2e-16 ***
## season55      9.767e+01  2.590e+00  37.714  < 2e-16 ***
## season56      9.835e+01  2.590e+00  37.978  < 2e-16 ***
## season57      9.803e+01  2.592e+00  37.815  < 2e-16 ***
## season58      9.803e+01  2.592e+00  37.816  < 2e-16 ***
## season59      9.709e+01  2.592e+00  37.453  < 2e-16 ***
## season60      9.698e+01  2.592e+00  37.410  < 2e-16 ***
## season61      9.651e+01  2.586e+00  37.325  < 2e-16 ***
## season62      9.830e+01  2.586e+00  38.016  < 2e-16 ***
## season63      9.641e+01  2.586e+00  37.284  < 2e-16 ***
## season64      9.460e+01  2.586e+00  36.585  < 2e-16 ***
## season65      1.142e+02  2.575e+00  44.335  < 2e-16 ***
## season66      1.267e+02  2.575e+00  49.223  < 2e-16 ***
## season67      1.392e+02  2.575e+00  54.058  < 2e-16 ***
## season68      1.413e+02  2.575e+00  54.891  < 2e-16 ***
## season69      1.396e+02  2.566e+00  54.410  < 2e-16 ***
## season70      1.384e+02  2.566e+00  53.953  < 2e-16 ***
## season71      1.380e+02  2.566e+00  53.775  < 2e-16 ***
## season72      1.377e+02  2.566e+00  53.686  < 2e-16 ***
## season73      1.437e+02  2.563e+00  56.043  < 2e-16 ***
## season74      1.407e+02  2.563e+00  54.877  < 2e-16 ***
## season75      1.380e+02  2.563e+00  53.849  < 2e-16 ***
## season76      1.371e+02  2.563e+00  53.487  < 2e-16 ***
## season77      1.384e+02  2.561e+00  54.038  < 2e-16 ***
## season78      1.349e+02  2.561e+00  52.694  < 2e-16 ***
## season79      1.345e+02  2.561e+00  52.539  < 2e-16 ***
```

```
## season80     1.347e+02  2.561e+00  52.602  < 2e-16 ***
## season81     1.323e+02  2.558e+00  51.696  < 2e-16 ***
## season82     1.305e+02  2.558e+00  51.005  < 2e-16 ***
## season83     1.292e+02  2.558e+00  50.492  < 2e-16 ***
## season84     1.275e+02  2.558e+00  49.841  < 2e-16 ***
## season85     1.103e+02  2.557e+00  43.154  < 2e-16 ***
## season86     1.085e+02  2.557e+00  42.444  < 2e-16 ***
## season87     1.028e+02  2.557e+00  40.210  < 2e-16 ***
## season88     1.032e+02  2.557e+00  40.373  < 2e-16 ***
## season89     2.833e+01  2.556e+00  11.081  < 2e-16 ***
## season90     2.995e+01  2.556e+00  11.715  < 2e-16 ***
## season91     5.943e-01  2.556e+00   0.233 0.816160
## season92    -2.636e+00  2.570e+00  -1.026 0.305112
## season93    -3.087e+00  2.570e+00  -1.201 0.229657
## season94    -8.937e+00  2.570e+00  -3.478 0.000510 ***
## season95    -3.033e+00  2.570e+00  -1.180 0.237916
## season96     6.070e-03  2.570e+00   0.002 0.998115
## trend       -3.697e-03  1.507e-04 -24.530  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.26 on 4313 degrees of freedom
## Multiple R-squared:  0.9555, Adjusted R-squared:  0.9545
## F-statistic: 955.7 on 97 and 4313 DF,  p-value: < 2.2e-16
```

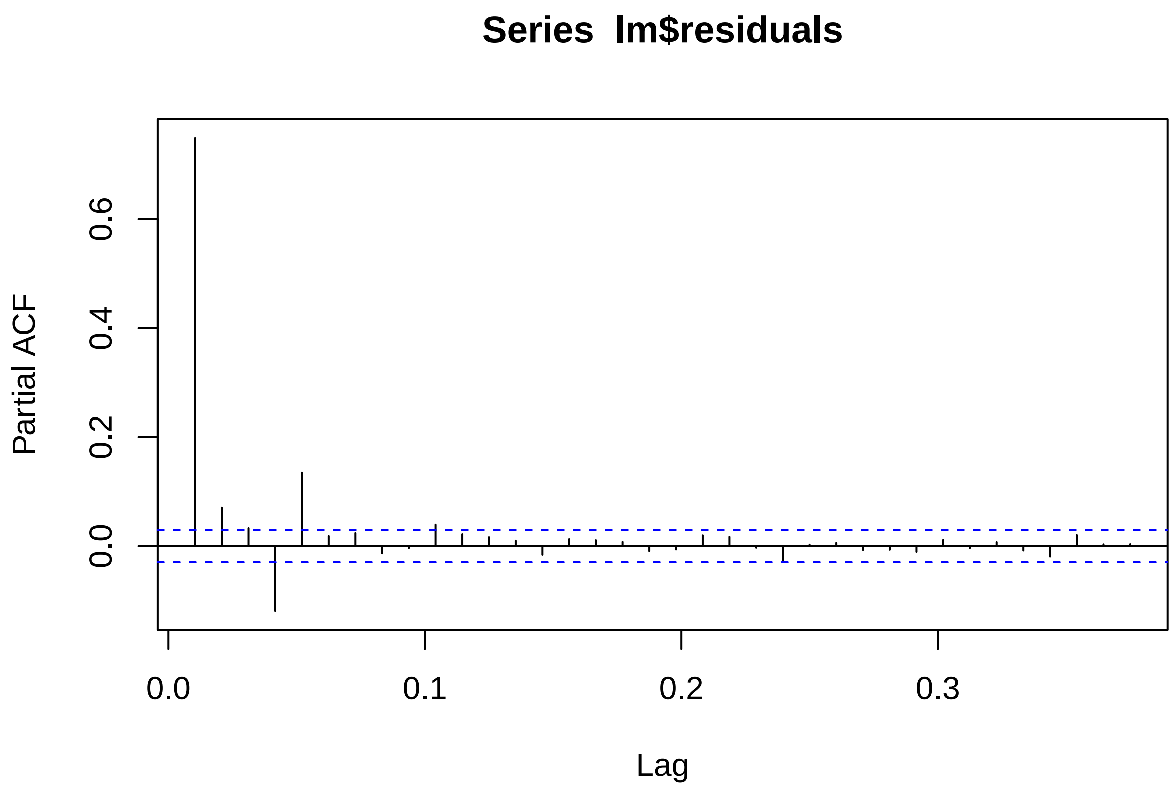
We can say that temperature is a significant feature, given that trend and majority of the seasons are below 0.05. So it needs to be added to the forecast.

```
checkresiduals(lm)
```



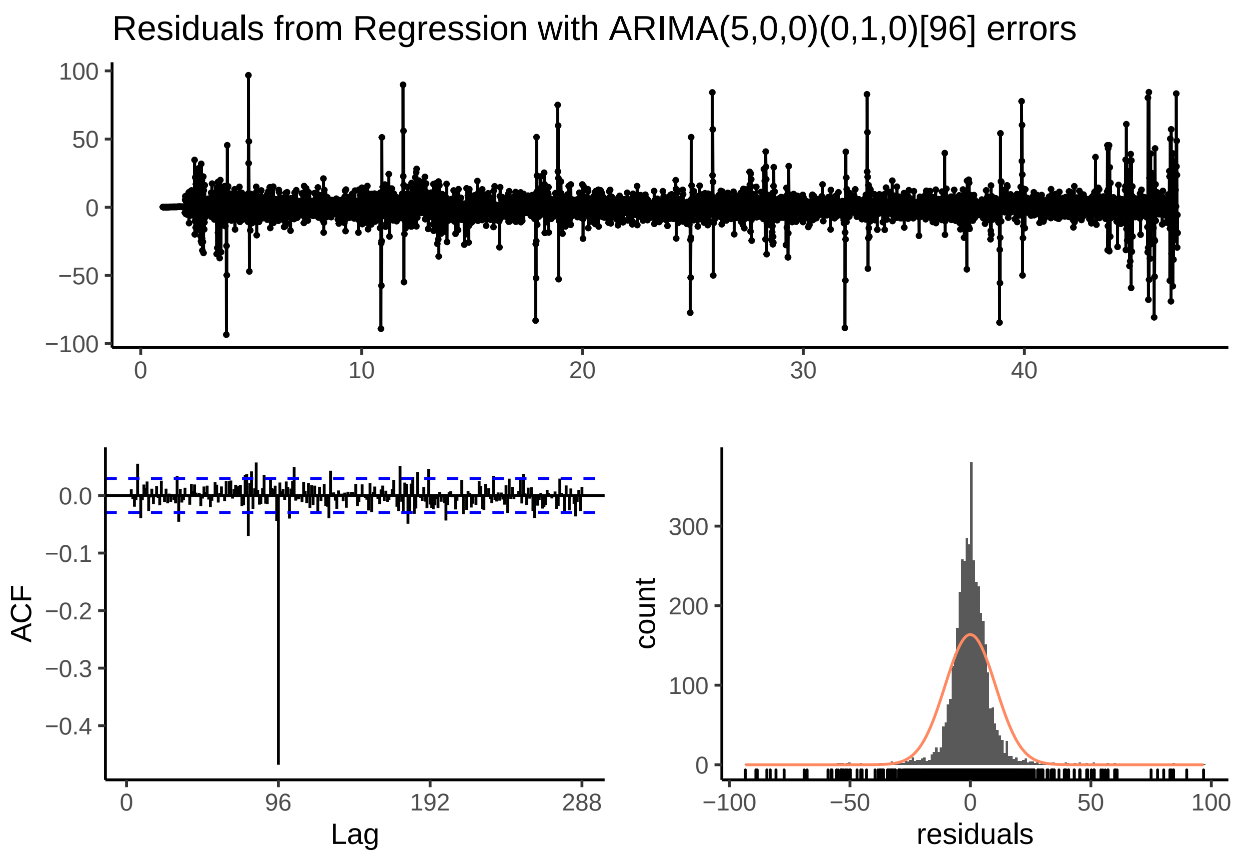

```
##
##  Breusch-Godfrey test for serial correlation of order up to 192
##
## data:  Residuals from Linear regression model
## LM test = 2692.9, df = 192, p-value < 2.2e-16
```

```
pacf(lm$residuals)
```



We fit the best Arima model with temperature.

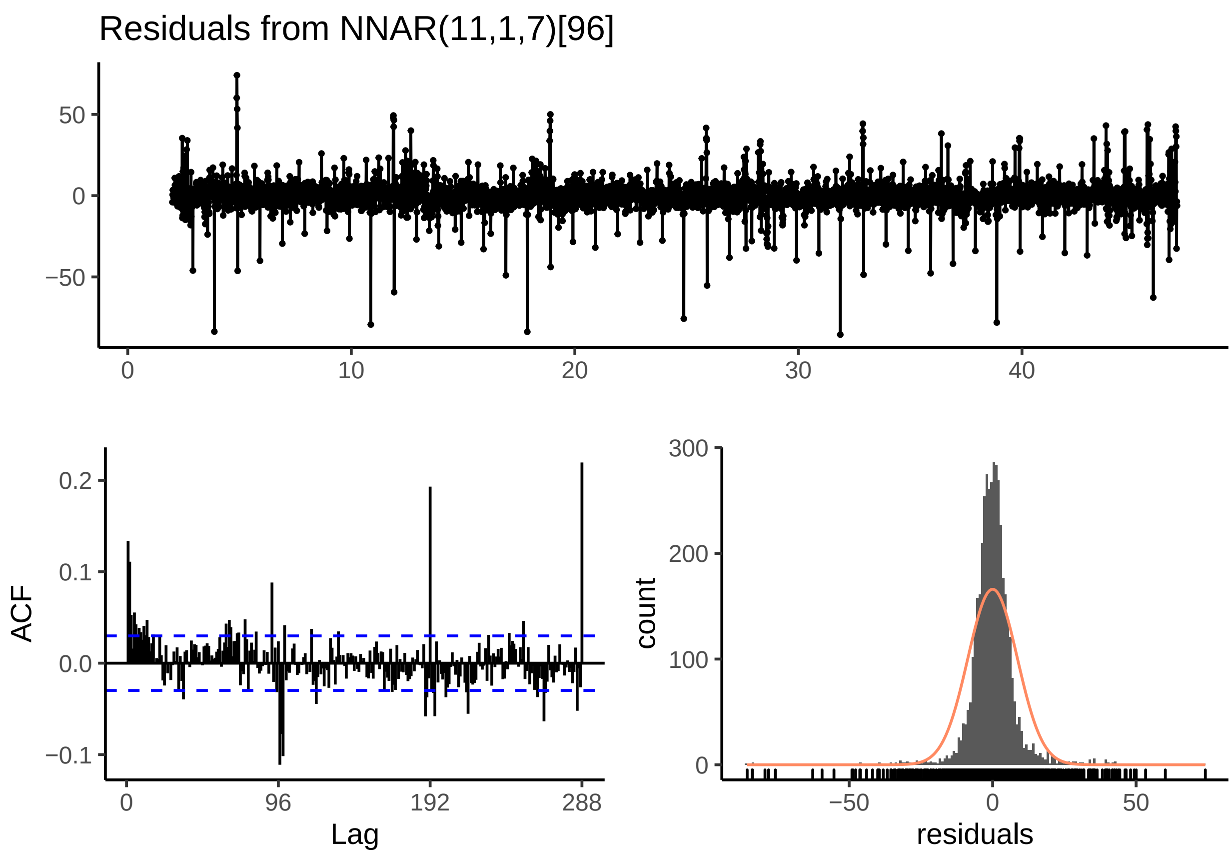
```
s_2 = Arima(e_train[,"power"],xreg=e_train[,"temp"],order=c(5,0,0),seasonal = c(0,1,0))
checkresiduals((s_2))
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(5,0,0)(0,1,0)[96] errors
## Q* = 1349.1, df = 186, p-value < 2.2e-16
##
## Model df: 6.   Total lags used: 192
```

```
nn_2=nnetar(e_train[,"power"],xreg=e_train[,"temp"])
checkresiduals(nn_2)
```

```
## Warning in modeldf.default(object): Could not find appropriate degrees of
## freedom for this model.
```



How about we test a Vectorial Auto Regressive Model?

Grouped Time Series models. Let's go! We take both columns.

```
e_var <- e_train
e_var <- e_var[, c("power","temp")]
```

```
library(vars)
```

```
## Loading required package: MASS

##
## Attaching package: 'MASS'

## The following objects are masked from 'package:fma':
##
##     cement, housing, petrol

## The following object is masked from 'package:dplyr':
##
##     select

## Loading required package: strucchange

## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich

## Loading required package: urca

## Loading required package: lmtest
```

```
VARselect(e_var, lag.max=10, type="const")
```

```
## $selection
## AIC(n)  HQ(n)  SC(n) FPE(n)
##     10      9      9     10
##
## $criteria
##                1         2         3         4         5         6         7
## AIC(n)  3.931438  3.932675  3.925397  3.920633  3.812948  3.813634  3.812562
## HQ(n)   3.934511  3.937797  3.932568  3.929852  3.824215  3.826950  3.827926
## SC(n)   3.940150  3.947194  3.945723  3.946767  3.844889  3.851382  3.856117
## FPE(n) 50.980258 51.043358 50.673213 50.432378 45.283732 45.314810 45.266255
##                8         9        10
## AIC(n)  3.806452  3.764471  3.763353
## HQ(n)   3.823865  3.783932  3.784864
## SC(n)   3.855815  3.819641  3.824331
## FPE(n) 44.990540 43.140866 43.092691
```

AIC leads me to select an order equal to 10.

```
var <- VAR(e_var, p=10, type="const")
summary(var)
```

```
##
## VAR Estimation Results:
## =========================
## Endogenous variables: power, temp
## Deterministic variables: const
## Sample size: 4401
## Log Likelihood: -20728.756
## Roots of the characteristic polynomial:
## 0.9427 0.9427 0.8946 0.8946 0.8671 0.8671 0.846 0.792 0.792 0.7468 0.7468 0.7099 0.7099 0.6867 0.6867
## Call:
## VAR(y = e_var, p = 10, type = "const")
##
##
## Estimation results for equation power:
## ======================================
```

```
## power = power.l1 + temp.l1 + power.l2 + temp.l2 + power.l3 + temp.l3 + power.l4 + temp.l4 + power.l5
##
##           Estimate Std. Error t value Pr(>|t|)
## power.l1   0.91598    0.01510  60.656  < 2e-16 ***
## temp.l1    0.06536    0.50841   0.129  0.89772
## power.l2   0.11405    0.02049   5.566 2.76e-08 ***
## temp.l2   -0.25183    0.70661  -0.356  0.72156
## power.l3  -0.02151    0.02055  -1.047  0.29530
## temp.l3    0.09467    0.70007   0.135  0.89244
## power.l4  -0.01906    0.02057  -0.926  0.35431
## temp.l4    1.44055    0.70015   2.057  0.03970 *
## power.l5  -0.02596    0.02080  -1.248  0.21210
## temp.l5   -0.99833    0.70677  -1.413  0.15787
## power.l6   0.02297    0.02097   1.096  0.27330
## temp.l6    0.30180    0.70137   0.430  0.66700
## power.l7   0.03456    0.02096   1.649  0.09924 .
## temp.l7   -0.62854    0.68692  -0.915  0.36023
## power.l8  -0.03208    0.02096  -1.530  0.12599
## temp.l8    2.03564    0.68687   2.964  0.00306 **
## power.l9  -0.01649    0.02098  -0.786  0.43185
## temp.l9   -0.43062    0.69135  -0.623  0.53341
## power.l10 -0.03927    0.01548  -2.536  0.01124 *
## temp.l10  -1.12260    0.49695  -2.259  0.02393 *
## const     10.00981    1.13559   8.815  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 14.83 on 4380 degrees of freedom
## Multiple R-Squared: 0.9336,  Adjusted R-squared: 0.9333
## F-statistic:  3081 on 20 and 4380 DF,  p-value: < 2.2e-16
##
##
## Estimation results for equation temp:
## =====================================
## temp = power.l1 + temp.l1 + power.l2 + temp.l2 + power.l3 + temp.l3 + power.l4 + temp.l4 + power.l5 +
##
##             Estimate Std. Error t value Pr(>|t|)
## power.l1  -2.761e-04  4.488e-04  -0.615    0.538
## temp.l1    9.813e-01  1.511e-02  64.942  < 2e-16 ***
## power.l2   2.096e-06  6.090e-04   0.003    0.997
## temp.l2    2.454e-03  2.100e-02   0.117    0.907
## power.l3  -1.698e-04  6.108e-04  -0.278    0.781
## temp.l3   -1.213e-03  2.081e-02  -0.058    0.954
## power.l4   6.034e-03  6.114e-04   9.868  < 2e-16 ***
## temp.l4    1.870e-01  2.081e-02   8.987  < 2e-16 ***
## power.l5  -5.168e-03  6.182e-04  -8.360  < 2e-16 ***
## temp.l5   -1.841e-01  2.101e-02  -8.765  < 2e-16 ***
## power.l6  -6.924e-04  6.232e-04  -1.111    0.267
## temp.l6    5.490e-03  2.085e-02   0.263    0.792
## power.l7   3.259e-04  6.230e-04   0.523    0.601
## temp.l7    1.708e-04  2.042e-02   0.008    0.993
## power.l8   3.587e-03  6.231e-04   5.757 9.14e-09 ***
## temp.l8    1.438e-01  2.041e-02   7.044 2.16e-12 ***
```

```
## power.l9  -3.055e-03  6.235e-04  -4.901 9.90e-07 ***
## temp.l9   -1.451e-01  2.055e-02  -7.062 1.90e-12 ***
## power.l10 -2.470e-04  4.602e-04  -0.537    0.591
## temp.l10  -1.776e-02  1.477e-02  -1.202    0.229
## const      2.232e-01  3.375e-02   6.613 4.23e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 0.4407 on 4380 degrees of freedom
## Multiple R-Squared: 0.9738,  Adjusted R-squared: 0.9737
## F-statistic:  8131 on 20 and 4380 DF,  p-value: < 2.2e-16
##
##
##
## Covariance matrix of residuals:
##          power   temp
## power 219.8665 0.1392
## temp    0.1392 0.1942
##
## Correlation matrix of residuals:
##        power   temp
## power 1.0000 0.0213
## temp  0.0213 1.0000
```

```
serial.test(var, lags.pt=10, type="PT.asymptotic")
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var
## Chi-squared = 38.815, df = 0, p-value < 2.2e-16
```

Forecast

```
library(forecast)
s2_f <- forecast(s_2,xreg=e_test[,"temp"],96)
nn2_f <- forecast(nn_2, xreg=e_test[,"temp"],96)
v_f <- forecast(var, xreg=e_test[,"temp"], h=96)

print(paste0("RMSE (Sarima): ",sqrt(mean((s2_f$mean-e_test[,"power"])^2))))
```

```
## [1] "RMSE (Sarima): 20.039676980261"
```

```
print(paste0("RMSE (NN): ",sqrt(mean((nn2_f$mean-e_test[,"power"])^2))))
```

```
## [1] "RMSE (NN): 18.2354605993451"
```

```
print(paste0("RMSE (Var - Power): ",sqrt(mean((v_f$forecast$power$mean-e_test[,"power"])^2))))
```

```
## [1] "RMSE (Var - Power): 49.0577087151196"
```

```
print(paste0("RMSE (Var - Temp): ",sqrt(mean((v_f$forecast$temp$mean-e_test[,"temp"])^2))))
```

```
## [1] "RMSE (Var - Temp): 3.3911558446532"
```

Testing VAR here wouldn't work since we only have two columns and as much as Var - Temp has 3.39 RMSE, that's not what i want to use or predict so here we give Neural Networks (at 17.7032301905825) - the nod for Forecasting with Temperature.

Preparing models for saving.

```
# SES model to predict without temperature.

SES_pred=ses(na.omit(elec[,"power"],alpha=NULL,beta=NULL,gamma=NULL))
ses_pred_f=forecast(SES_pred,xreg=pred_temp, n.ahead=96)

SES_pred_multi=HoltWinters(na.omit(elec[,"power"],alpha=NULL,beta=NULL,gamma=NULL,seasonal = 'multi' ))
SES_pred_multi_f<-predict(SES_pred_multi, n.ahead=96)

NN_pred= nnetar(na.omit(elec[,"power"]), xreg=elec[1:nrow(values_of_e),"temp"])
nn_pred_f=forecast(NN_pred,xreg=pred_temp, h=96)
```

Predictions, Save CSV.

Initially, SES was my forecasting pick without temperature.

But in the end, the observations of SES weren't enough to predict the next day as it only generated values for 10 observations when we need 96.

This means I have to choose the next lowest predictor and that was Holt Winters Multi at 13.9237614756714.

As for the forecasts with temperature, Neural Networks wins.

Now we write to CSV.

```
library(xts)
```

```
##
## Attaching package: 'xts'

## The following objects are masked from 'package:dplyr':
##
##     first, last
```

```
# sp <- as.data.frame(ses_pred_f$mean[1:96])
sm <- as.data.frame(SES_pred_multi_f[1:96])
np <- as.data.frame(nn_pred_f$mean[1:96])

names(sm) <- c("Prediction without Temperature")
names(np) <- c("Prediction with Temperature")

new <- sm
new <- cbind(sm, np)
new
```

```
##    Prediction without Temperature Prediction with Temperature
## 1                        145.8903                    153.7513
## 2                        143.8029                    156.1772
## 3                        141.2950                    153.3954
## 4                        141.8876                    156.3812
## 5                        147.2551                    156.1634
## 6                        144.3237                    156.6316
## 7                        137.2356                    153.0678
## 8                        134.0645                    152.7835
## 9                        134.5679                    154.7701
## 10                       137.7683                    155.4734
## 11                       138.6246                    155.9497
## 12                       133.8825                    156.1880
## 13                       131.9974                    156.2286
## 14                       125.9711                    155.8101
## 15                       123.6377                    155.7690
## 16                       127.2008                    157.1656
## 17                       127.7419                    158.4920
## 18                       124.7421                    157.9634
## 19                       122.7602                    157.3045
## 20                       124.2979                    157.2136
## 21                       125.0456                    157.8514
## 22                       125.5668                    158.1279
## 23                       124.3608                    157.9097
## 24                       125.6249                    158.0031
## 25                       125.7768                    158.3182
## 26                       130.3395                    159.2394
## 27                       141.3704                    160.4332
## 28                       141.3375                    161.8778
## 29                       145.7570                    174.2335
## 30                       140.4098                    171.9313
## 31                       131.5136                    168.3507
## 32                       133.9902                    166.4775
## 33                       138.9867                    166.4138
## 34                       218.4472                    229.3442
## 35                       219.5554                    241.5280
## 36                       219.3415                    243.7895
## 37                       215.9591                    239.8887
## 38                       223.8903                    239.4937
## 39                       220.2540                    238.5006
## 40                       219.8954                    239.0926
## 41                       222.1516                    242.5753
## 42                       222.5110                    241.5090
## 43                       226.4865                    245.3000
## 44                       228.9188                    252.3104
## 45                       229.6786                    262.2989
## 46                       230.1832                    269.9512
## 47                       228.9153                    268.9638
## 48                       230.7467                    268.1266
## 49                       232.4754                    272.1999
## 50                       237.2737                    280.1098
## 51                       231.2851                    273.1587
## 52                       243.3654                    279.7564
## 53                       239.4165                    284.3937
```

```
## 54                    235.2253                    278.6242
## 55                    238.4838                    278.7333
## 56                    244.8126                    282.0317
## 57                    237.2859                    280.6560
## 58                    241.9395                    282.5094
## 59                    246.0476                    283.6380
## 60                    246.6461                    280.7577
## 61                    258.3714                    285.2338
## 62                    248.3369                    287.3054
## 63                    244.2686                    278.7479
## 64                    243.4862                    281.5772
## 65                    241.6694                    282.3405
## 66                    239.6929                    280.6696
## 67                    253.5084                    286.2912
## 68                    244.9637                    281.0687
## 69                    237.9346                    280.9160
## 70                    240.9730                    282.1367
## 71                    243.9171                    275.6340
## 72                    276.3771                    284.3273
## 73                    302.6919                    292.9702
## 74                    301.1728                    291.9735
## 75                    306.2093                    303.1659
## 76                    297.5817                    300.4792
## 77                    292.0235                    298.3446
## 78                    300.5035                    304.7959
## 79                    290.8368                    301.6278
## 80                    288.2034                    304.4645
## 81                    288.0550                    307.1188
## 82                    286.8207                    303.4701
## 83                    285.5215                    307.2465
## 84                    279.0651                    305.8983
## 85                    281.6604                    302.9489
## 86                    280.8058                    304.8625
## 87                    276.5391                    302.9824
## 88                    275.6485                    301.9375
## 89                    277.2433                    301.2528
## 90                    258.4152                    293.9322
## 91                    257.4927                    289.3958
## 92                    256.3355                    286.2474
## 93                    256.1010                    283.4061
## 94                    186.3203                    237.9368
## 95                    182.3607                    216.5476
## 96                    150.5263                    188.8202
```

```
# we save our two columns in one file.
write.xlsx(new, "Predict.xls", row.names=FALSE, append=TRUE)
```

In the end. save files are predict no temp final and predict with temp.