ENE 434 Lab 6 Assignment Final

Ethan McChristian, Mitchell Tyse, Sebastian Borgen

#loading in data (lab stuff)

```
if (!require('pacman')) install.packages('pacman')
```

```
## Loading required package: pacman
```

```
library(pacman)
p_load(tidyverse, fpp3, lubridate)
```

```
power_df = read_csv("https://raw.githubusercontent.com/emcchri5/codebank/main/power_df.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   FRE = col_logical(),
##   date = col_date(format = "")
## )
## i Use `spec()` for the full column specifications.
```

```
colnames(power_df)[23] = 'ets_price'
power_df$date = yearmonth(power_df$date)
power_ts = tsibble(power_df, index = date)
power_ts = power_ts %>% mutate(
  DK1 = DK1/1000
)
```

```
#scenarios

#no change in ets price
scen1 = new_data(power_ts, 12) %>%
  mutate(ets_price = rep(power_ts$ets_price[128],12))

#constant increase of .5EUR
scen2 = new_data(power_ts, 12) %>%
  mutate(
    ets_price = rep(power_ts$ets_price[128],12) + cumsum(rep(.5,12))
         )
```

#Assignment ##Question 1 In a dynamic regression model, it may make sense to include lagged variables as exogenous regressors. In the model of DK1 prices, include both contemporaneous and lagged carbon permit prices. How does this change your model? (You may want to read Ch 10.6 in fpp3).

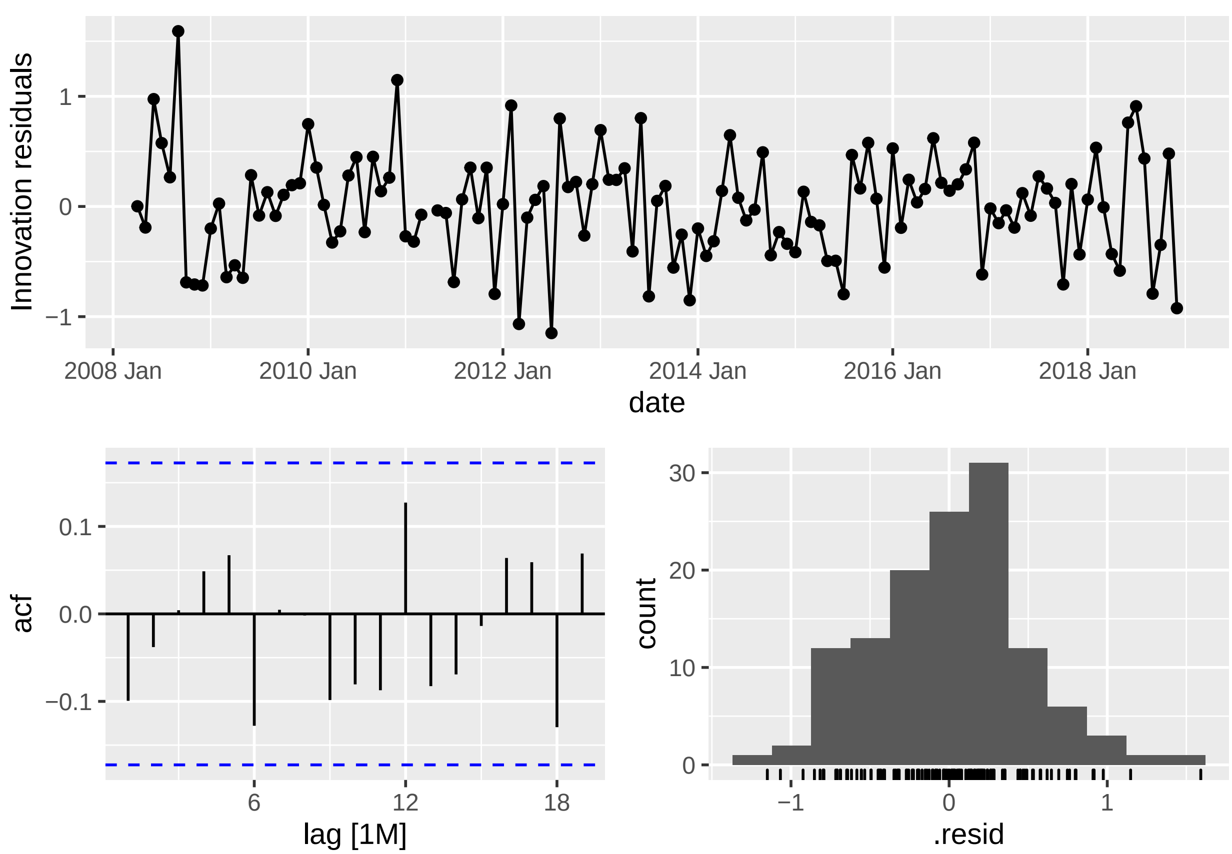
```
armax3 = power_ts %>% fill_gaps() %>% model(
  modWithEts = ARIMA(DK1 ~ ets_price + pdq(2,1,0)),
  mod_ets_lagged = ARIMA(DK1 ~ ets_price + pdq(2,1,0) +
                lag(ets_price) + lag(ets_price, 2) + lag(ets_price, 3)))
glance(armax3) %>% arrange(AICc) #better fit!!
```

```
## # A tibble: 2 x 8
##   .model         sigma2 log_lik   AIC  AICc   BIC ar_roots  ma_roots
##   <chr>           <dbl>   <dbl> <dbl> <dbl> <dbl> <list>    <list>
## 1 mod_ets_lagged  0.209   -78.1  170.  171.  190. <cpl [2]> <cpl [0]>
## 2 modWithEts      0.233   -87.1  182.  182.  194. <cpl [2]> <cpl [0]>
```

```
armax2 = power_ts %>% fill_gaps() %>% model(
  modWithEts = ARIMA(DK1 ~ ets_price + pdq(2,1,0)),
  modWOutEts = ARIMA(DK1 ~ pdq(2,1,0))
  )
```

```
armax3 %>%
  select(modWithEts) %>%
  gg_tsresiduals()
```

```
## Warning: Removed 1 rows containing missing values (geom_point).

## Warning: Removed 1 rows containing non-finite values (stat_bin).
```

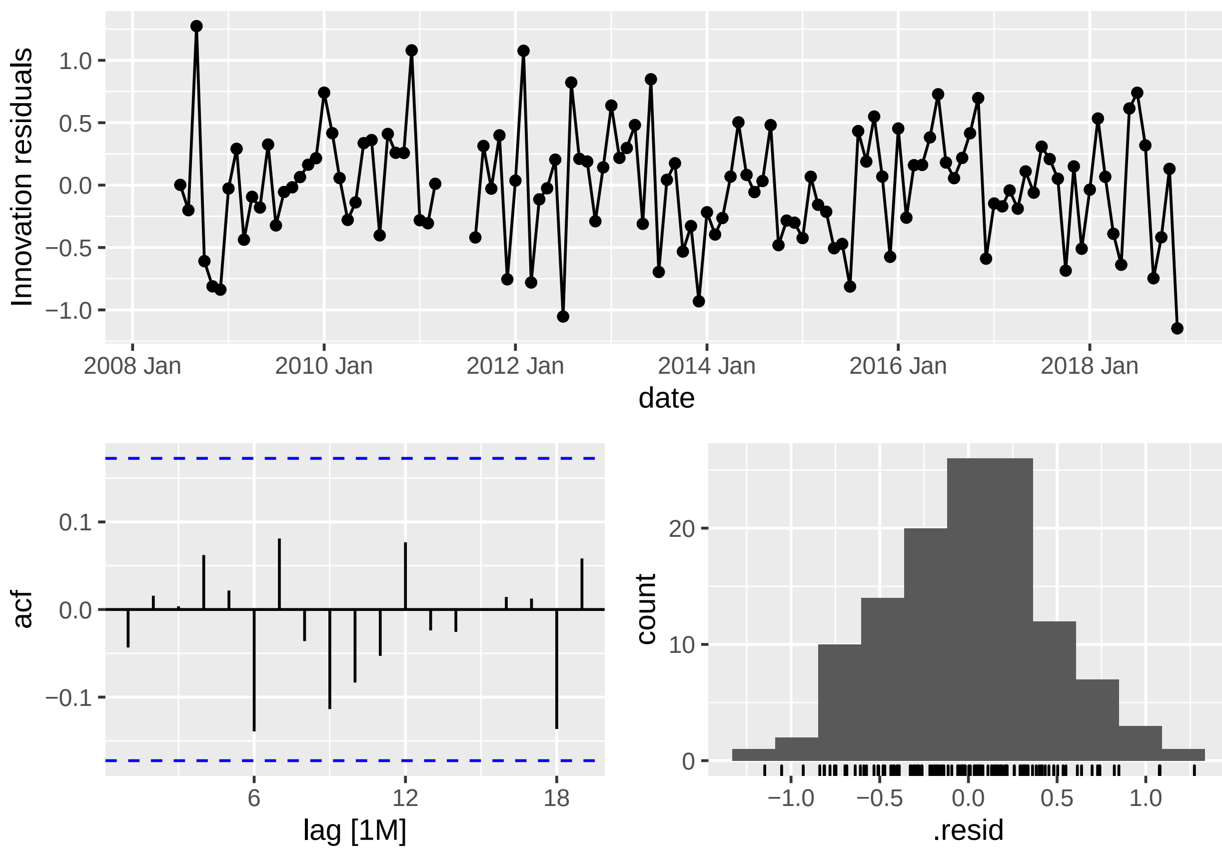


```
armax3 %>%
  select(mod_ets_lagged) %>%
  gg_tsresiduals()
```

```
## Warning: Removed 3 row(s) containing missing values (geom_path).
```

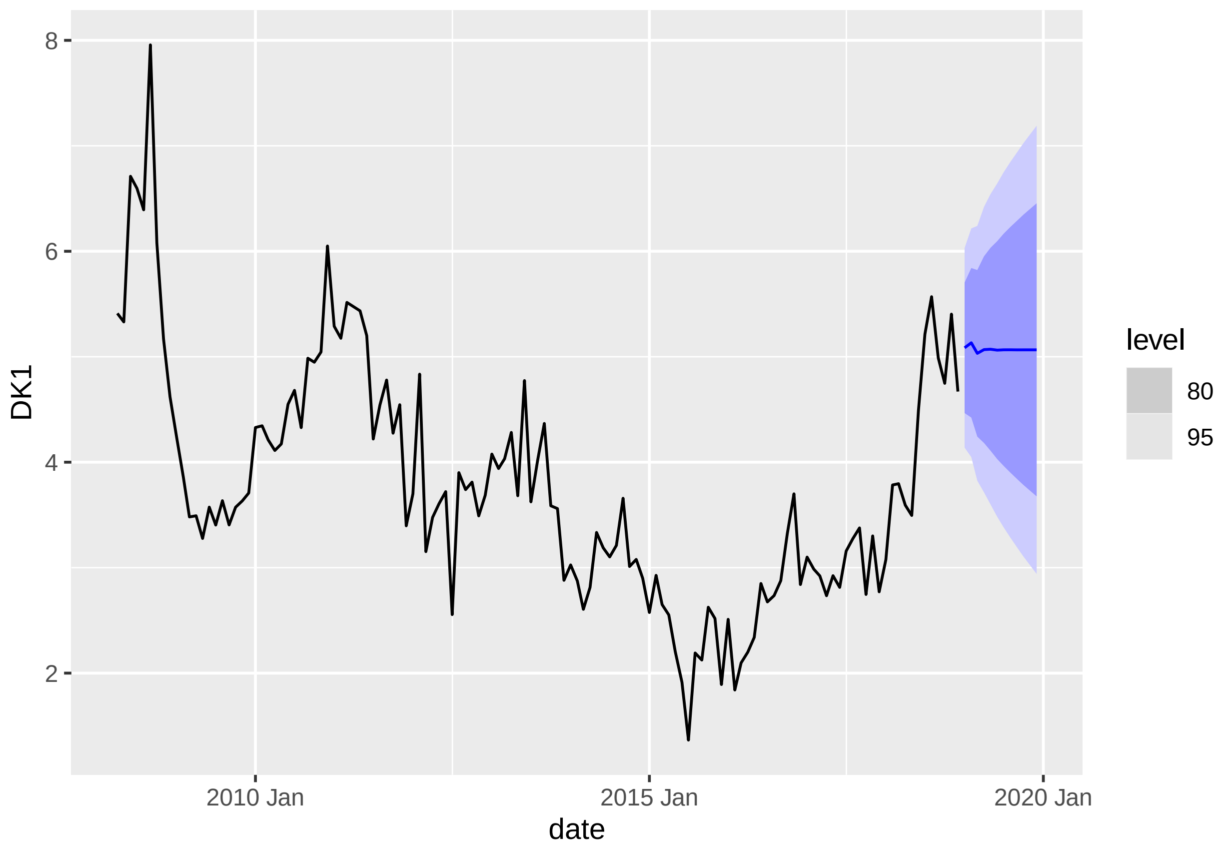
```
## Warning: Removed 7 rows containing missing values (geom_point).
```

```
## Warning: Removed 7 rows containing non-finite values (stat_bin).
```

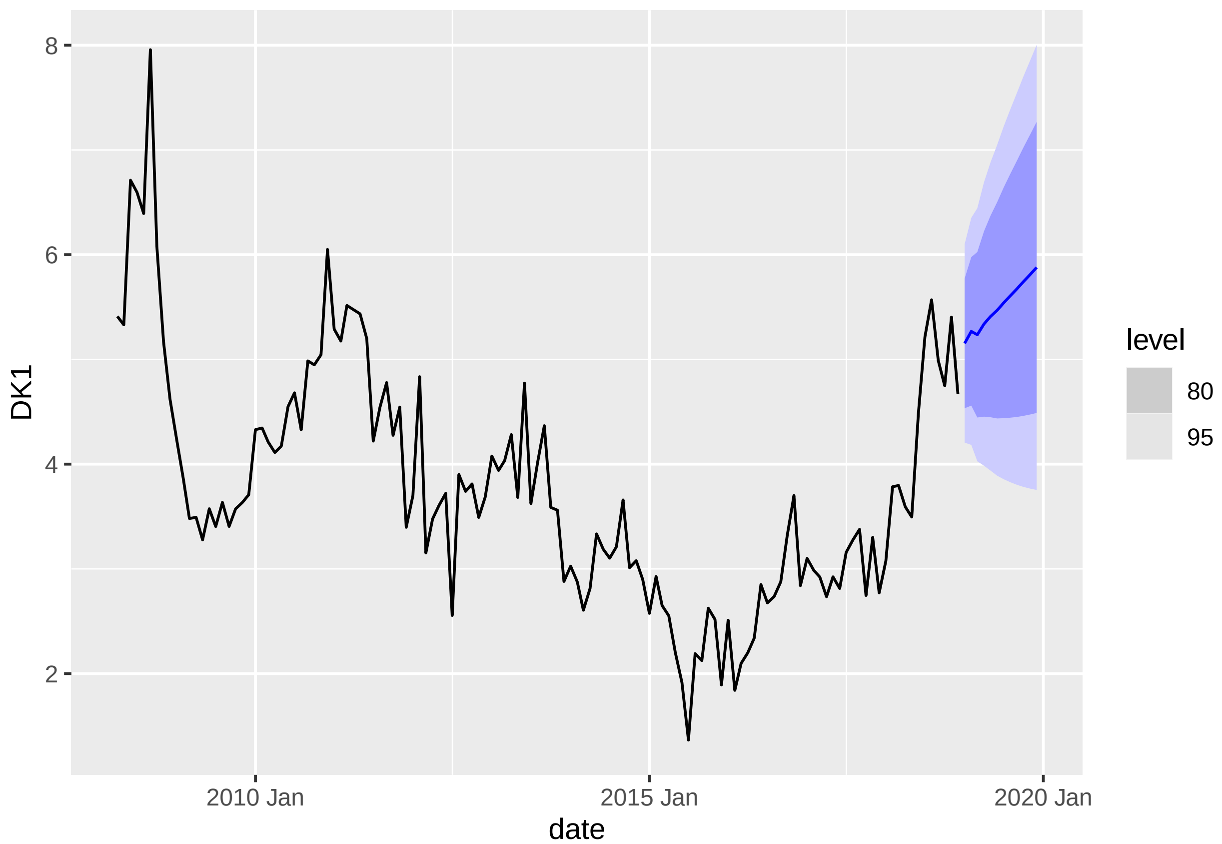



```
fcast3 = armax2 %>% select(modWithEts) %>% forecast(new_data=scen1)
fcast4 = armax2 %>% select(modWithEts) %>% forecast(new_data=scen2)
fcast5 = armax3 %>% select(mod_ets_lagged) %>% forecast(new_data=scen1)
fcast6 = armax3 %>% select(mod_ets_lagged) %>% forecast(new_data=scen2)
```

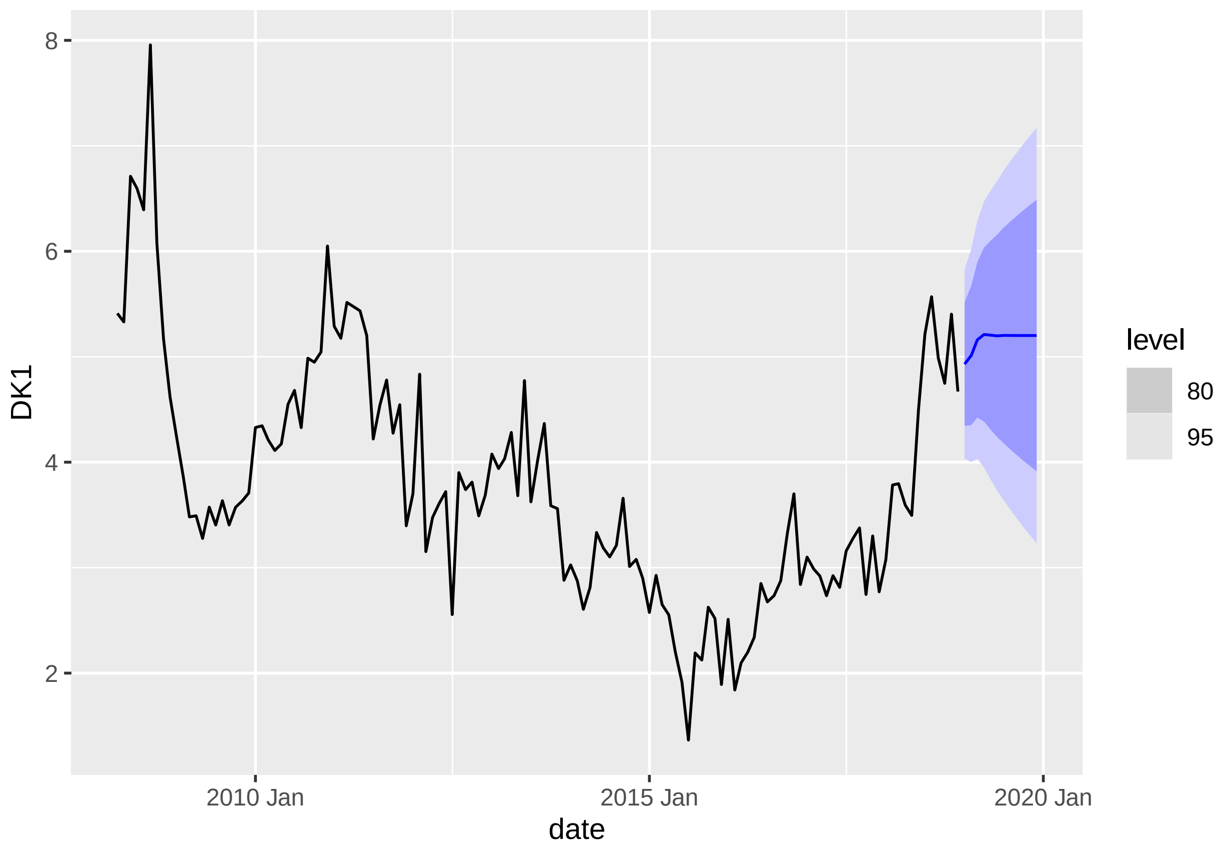
```
fcast3 %>% autoplot(power_ts) #constant carbon prices, ets model (no lags)
```



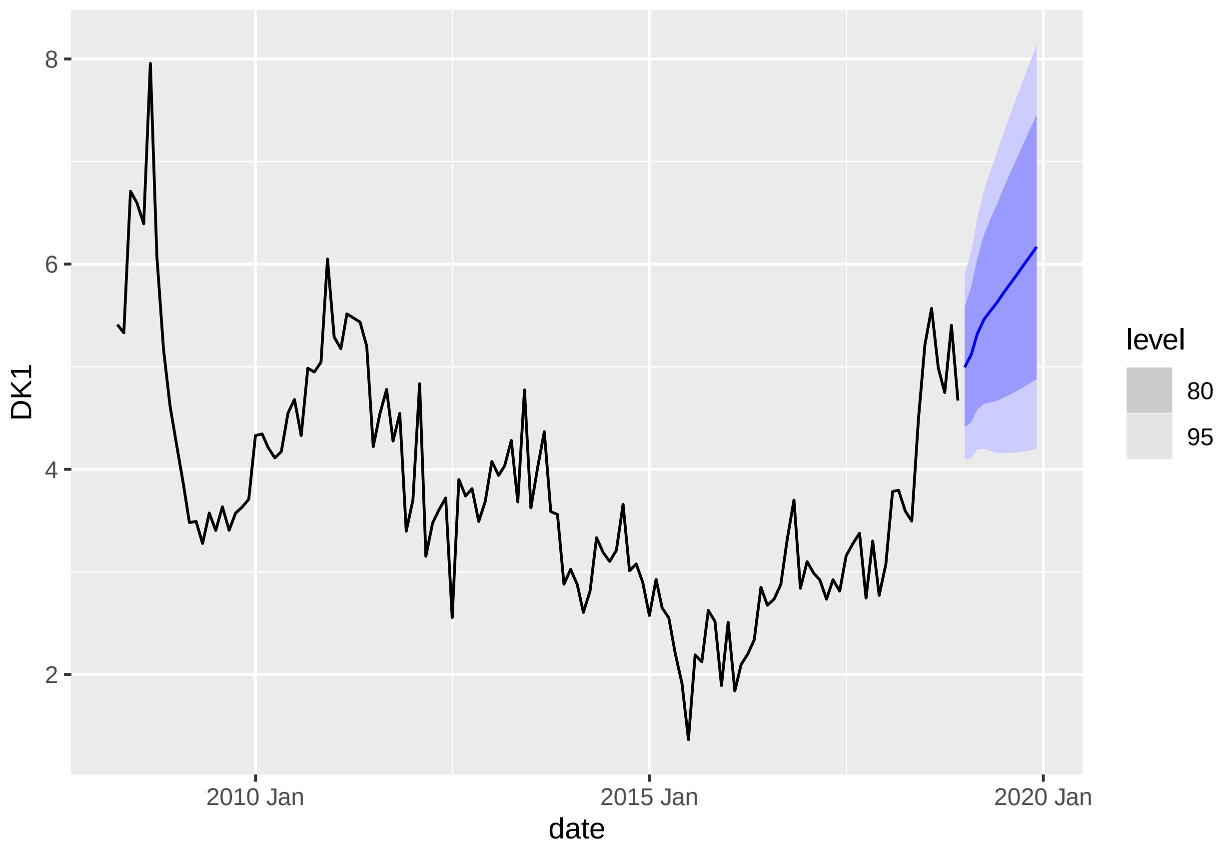

```
fcast4 %>% autoplot(power_ts) #increasing carbon prices, ets model
```




```
fcast5 %>% autoplot(power_ts) #constant carbon prices, lagged model
```




```
fcast6 %>% autoplot(power_ts) #increasing carbon prices, lagged model
```



Answer: Without lagged variables, the error margins go lower and price doesn't seem to respond as drastically to carbon prices in the predictions. When including lagged variables, the carbon prices seem to respond more and the error margins are smaller. The AICC is lower with lagged variables.

##Question 2 From ENTSOE-E or statnett, download hourly consumption data for Norway for 2017 and 2018. Join this with the 2019 data in order to create one long time series for Norwegian consumption. Then model the seasonality in the data (at monthly, weakly and daily level), with fourier terms. ###importing data

```
cons_2017 <- read_csv('https://raw.githubusercontent.com/emcchri5/codebank/main/NO_Energy_Cons_2017.csv
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   `Time(Local)` = col_character(),
##   Production = col_double(),
##   Consumption = col_double()
## )
```

```
cons_2018 <- read_csv('https://raw.githubusercontent.com/emcchri5/codebank/main/NO_Energy_Cons_2018.csv
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   `Time(Local)` = col_character(),
```

```
##   Production = col_double(),
##   Consumption = col_double()
## )
```

```
cons = read_csv2("http://jmaurit.github.io/analytics/labs/data/consumption-no-areas_2019_hourly.csv")
```

```
## i Using ',' as decimal and '.' as grouping mark. Use `read_delim()` for more control.
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Date = col_character(),
##   Hours = col_character(),
##   NO1 = col_double(),
##   NO2 = col_double(),
##   NO3 = col_double(),
##   NO4 = col_double(),
##   NO5 = col_double(),
##   NO = col_double()
## )
```

###getting hour columns

```
cons_2017 <- cons_2017 %>%
  separate('Time(Local)', sep = ' ', into=c('date','time','timezone')) %>%
  separate(time, sep = ':', into=c('hour', 'minute', 'second'))
cons_2018 <- cons_2018 %>%
   separate('Time(Local)', sep = ' ', into=c('date','time','timezone')) %>%
  separate(time, sep = ':', into=c('hour', 'minute', 'second'))
cons <- cons %>% separate(Hours, sep = '-', into=c('start','end'))
```

###converting hour to numeric

```
cons_2017 <- cons_2017 %>%
  mutate(hour = as.numeric(hour))
cons_2018 <- cons_2018 %>%
  mutate(hour = as.numeric(hour))
```

###binding 2017 and 2018 data (easy part)

```
cons_20178 <- rbind(cons_2017, cons_2018)
cons_20178 <- cons_20178 %>%
  select(date, hour, Consumption)
cons_2019 <- cons %>%
  select(Date, start, NO)
```

###wrangling date values to be parallel

```
cons_20178 <- cons_20178 %>%
  rename('cons' = 'Consumption')
cons_20178$date <- gsub('\\.', '/', cons_20178$date)
```

```r
cons_2019$time <- gsub('\\s+', '', cons$start)
cons_2019$hour <- as.numeric(cons_2019$time)
cons_2019 <- cons_2019 %>%
  select(Date, hour, NO) %>%
  rename('date' = 'Date', 'cons' = 'NO')
```

###binding datasets 2017/2018 and 2019, fixing data to tsibble

```r
cons_total <- rbind(cons_20178, cons_2019)

cons_total["period"] = dmy_h(paste(cons_total$date, cons_total$hour))

#check for NA's
cons_total[!complete.cases(cons_total), ]
```

```
## # A tibble: 1 x 4
##   date        hour  cons period
##   <chr>      <dbl> <dbl> <dttm>
## 1 31/03/2019     2    NA 2019-03-31 02:00:00
```

```r
#replace NA's
cons_total[["cons"]][cons_total$period==as_datetime("2019-03-31 02:00:00")] = cons_total[["cons"]][cons_
#check for duplicates
duplicates(cons_total)
```

```
## Using `period` as index variable.

## # A tibble: 6 x 4
##   date        hour  cons period
##   <chr>      <dbl> <dbl> <dttm>
## 1 29/10/2017     2 13168 2017-10-29 02:00:00
## 2 29/10/2017     2 13108 2017-10-29 02:00:00
## 3 28/10/2018     2 15134 2018-10-28 02:00:00
## 4 28/10/2018     2 15123 2018-10-28 02:00:00
## 5 27/10/2019     2 13760 2019-10-27 02:00:00
## 6 27/10/2019     2 13685 2019-10-27 02:00:00
```

```r
#run this 3 times  until duplicates(cons_total is gone)
dupRow = duplicates(cons_total)[2,]
```

```
## Using `period` as index variable.
```

```r
cons_total = cons_total %>% rows_delete(dupRow, by=c("period", 'cons'))
```

```
## Ignoring extra columns: date, hour
```

```r
dupRow = duplicates(cons_total)[2,]
```

```
## Using `period` as index variable.
```

```r
cons_total = cons_total %>% rows_delete(dupRow, by=c("period", 'cons'))
```

```
## Ignoring extra columns: date, hour
```

```r
dupRow = duplicates(cons_total)[2,]
```

```
## Using `period` as index variable.
```

```r
cons_total = cons_total %>% rows_delete(dupRow, by=c("period", 'cons'))
```

```
## Ignoring extra columns: date, hour
```

```r
duplicates(cons_total) #should be an empty 0x4 dataframe
```

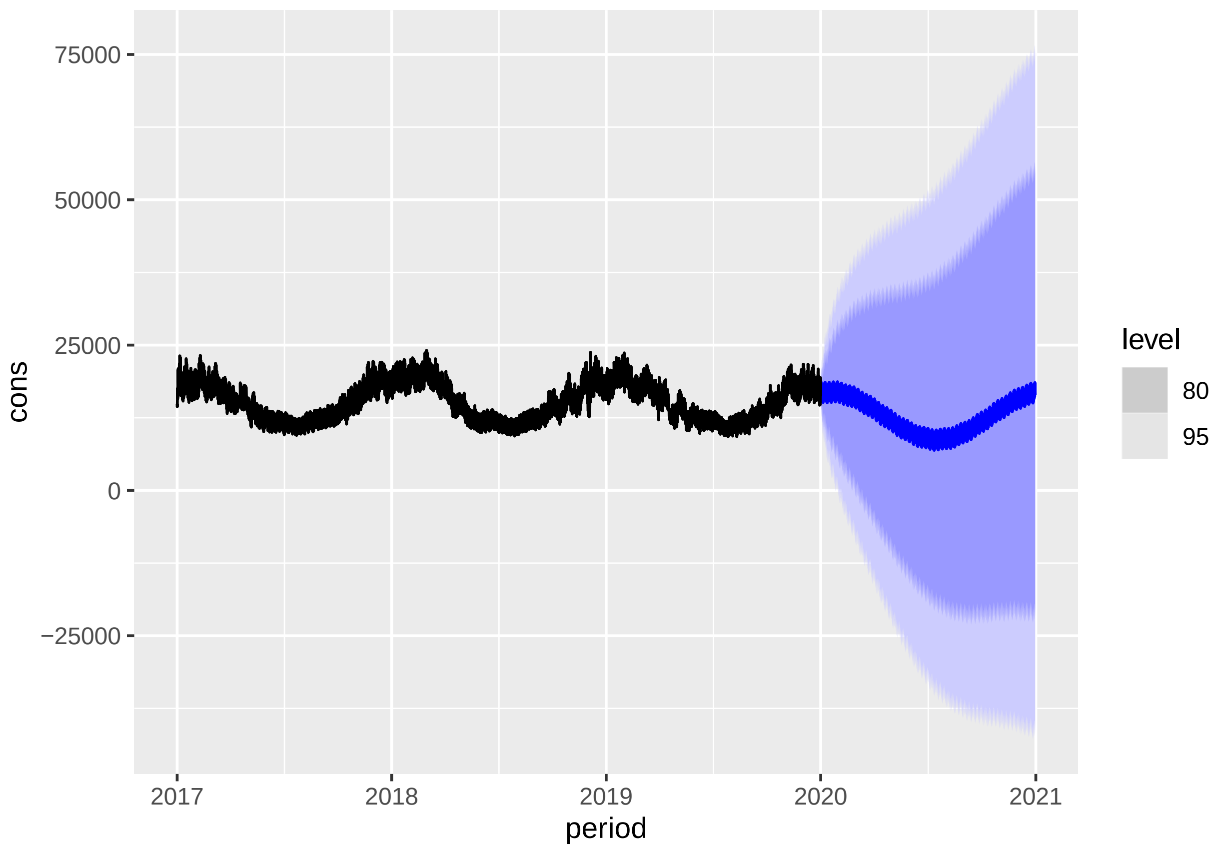
```
## Using `period` as index variable.

## # A tibble: 0 x 4
## # ... with 4 variables: date <chr>, hour <dbl>, cons <dbl>, period <dttm>
```

```r
cons_total_ts <- cons_total %>%
  tsibble(index=period) %>%
  fill_gaps()
```

```r
smod3 = cons_total_ts %>% model(
  fmod7 = ARIMA(cons ~ fourier(period = 24, K = 1) + fourier(period = 24*7, K = 1) + fourier(period = 24
)
glance(smod3)
```

```
## # A tibble: 1 x 8
##   .model sigma2  log_lik     AIC    AICc     BIC ar_roots   ma_roots
##   <chr>   <dbl>    <dbl>   <dbl>   <dbl>   <dbl> <list>     <list>
## 1 fmod7  63599. -182599. 365226. 365226. 365340. <cpl [5]> <cpl [0]>
```

```r
forecast <- smod3 %>%
  forecast(h = 24*365)
forecast %>% autoplot(cons_total_ts)
```



##Question 3 Create a VAR model for consumption and prices in 2019 using Danish data (You can find it on ENTSOE_E or at the Danish TSO's energy data site. Create a 30 day forecast. Load in actual data for january 2020–how does your forecast look? Include wind power in Denmark as a variable. How does this affect the model and forecast? #data wrangling, getting dan consumption ready

```
DAN <- read.csv('https://raw.githubusercontent.com/emcchri5/codebank/main/Total%20Load%20Denmark.csv')
DAN <- DAN %>%
  rename('total_load' = 'Actual.Total.Load..MW....CTA.DK', 'datetime' = 'Time..CET.CEST.') %>%
  select(datetime, total_load)
DAN <- DAN %>%  separate('datetime', sep = ' - ', into=c('start','end')) %>%
  separate('start', sep = ' ', into = c('date', 'time')) %>%
  select(-end)
```

#data wrangling, getting dan prices together

```
DAN_price <- read.csv('https://raw.githubusercontent.com/emcchri5/codebank/main/EL_prices_2019.csv')
DAN_price <- DAN_price %>% select("ï..HourUTC","PriceArea","SpotPriceEUR") %>%
  rename(Time = 'ï..HourUTC', price = "SpotPriceEUR") %>%
  filter(PriceArea == 'DK1') %>%
  separate('Time', sep = 'T', into=c('date','time')) %>%
  separate('time', sep = "[+]", into=c('tid','useless')) %>%
  select(-useless,-PriceArea) %>%
  separate('tid', sep = ':', into= c('hour', 'minute','second'))
DAN_price$time <- paste(DAN_price$hour, DAN_price$minute, sep= ":")
DAN_price <- DAN_price %>%
  select(-hour,-minute,-second)
```

```r
DAN_price <- DAN_price %>%
  separate('date', sep = '-', into = c('year', 'month', 'day'))
DAN_price$dizzy <- paste(DAN_price$day, DAN_price$month, sep = '.')
DAN_price$date <- paste(DAN_price$dizzy, DAN_price$year, sep = '.')
DAN_price <- DAN_price %>%
  select(-year,-month,-day,-dizzy)
```

#data wrangling… merging sets and tsibble stuff

```r
DAN_data <- left_join(DAN_price, DAN)
```

```
## Joining, by = c("time", "date")
```

```r
DAN_data <- DAN_data %>%
  separate('time',sep = ':', into = c('hour','minute'))
DAN_data$date <- gsub('\\.', '-', DAN_data$date)
DAN_data["period"] = dmy_h(paste(DAN_data$date, DAN_data$hour))
DAN_data[!complete.cases(DAN_data), ]
```

```
##      price hour minute       date total_load              period
## 6599 22.47   02     00 31-03-2019         NA 2019-03-31 02:00:00
```

```r
DAN_data[["total_load"]][DAN_data$period==as_datetime("2019-03-31 02:00:00")] = DAN_data[["total_load"]]
dupRow = duplicates(DAN_data)[2,]
```

```
## Using `period` as index variable.
```

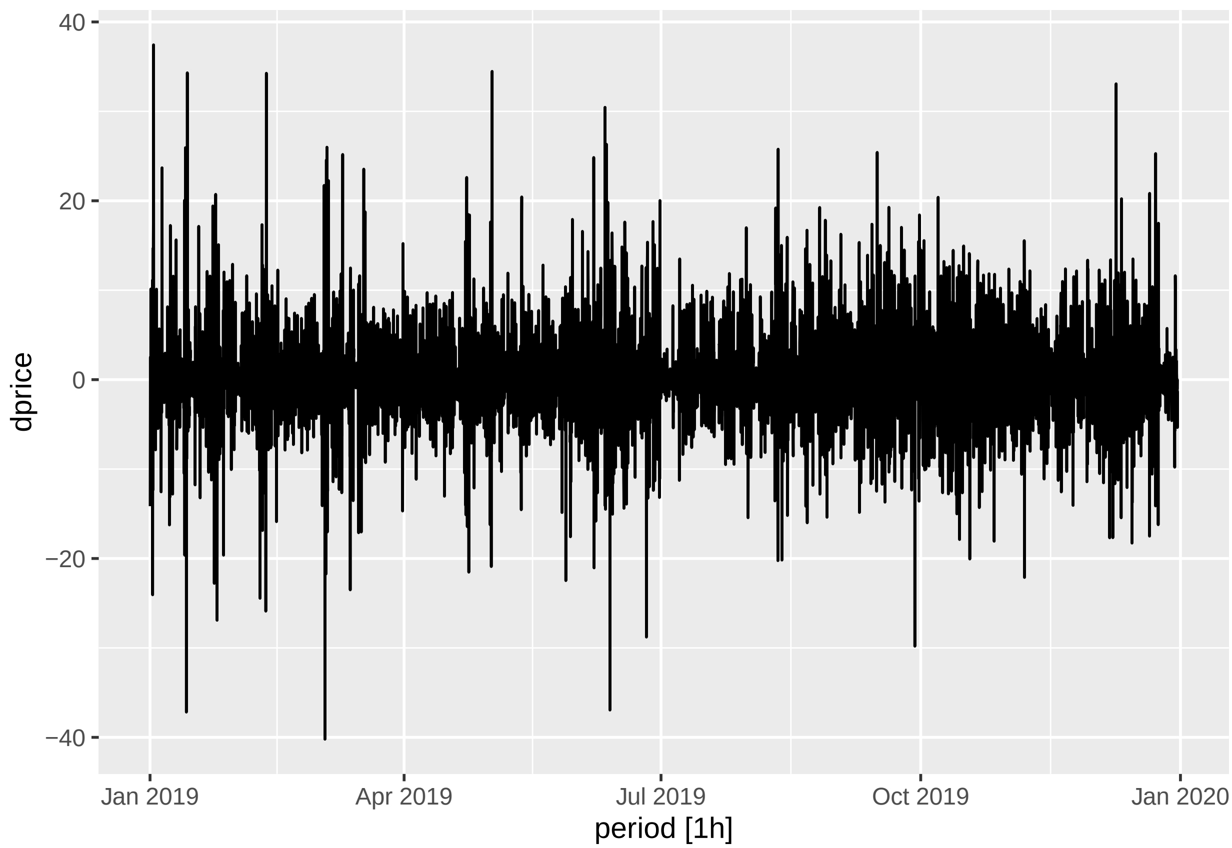
```r
DAN_data = DAN_data %>% rows_delete(dupRow, by=c("period", 'total_load'))
```

```
## Ignoring extra columns: price, hour, minute, date
```

```r
DAN_data <- DAN_data %>%
  select(-hour, -minute) %>%
  tsibble(index='period')
```

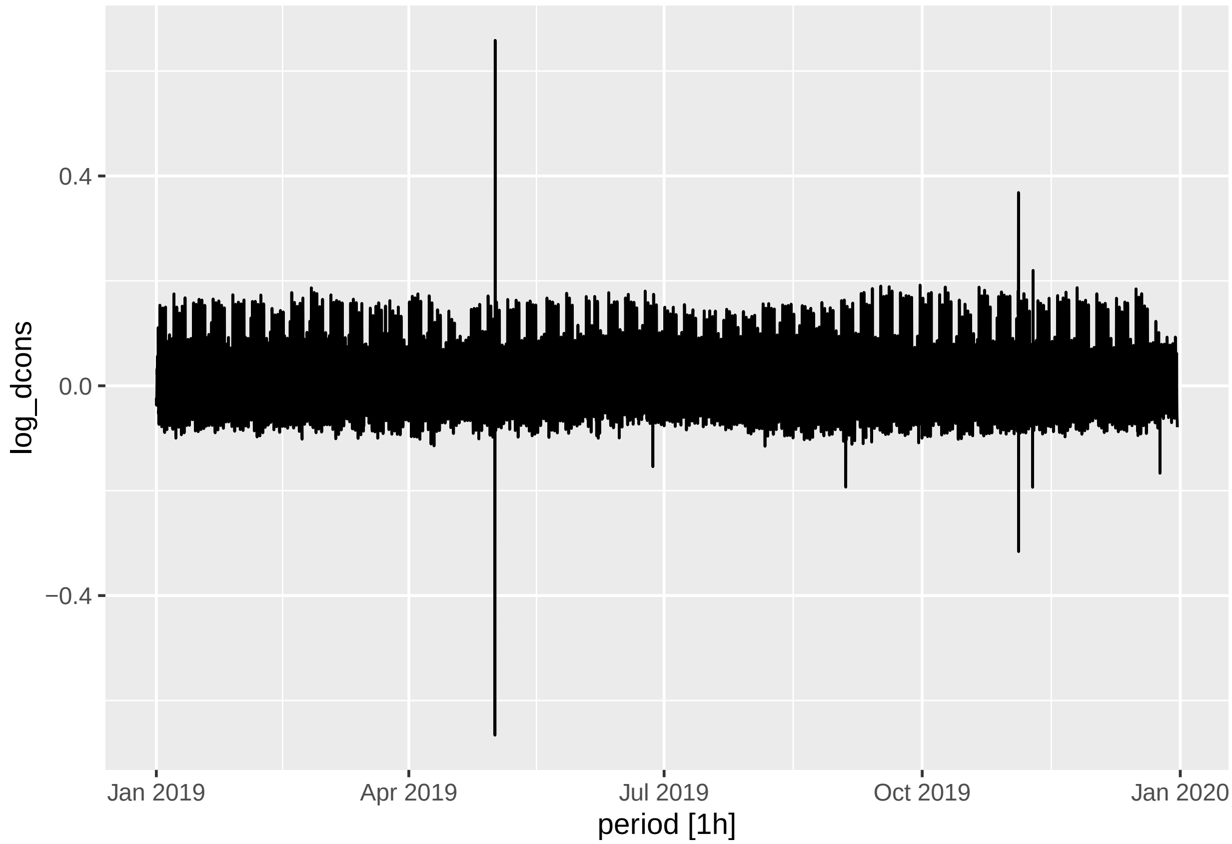
```r
DAN_data_diff <- DAN_data %>% mutate(
  dprice = difference(price), #can't log price since it's negative
  log_dcons = difference(log(total_load))
)
DAN_data_diff<- na.omit(DAN_data_diff) %>%
  select(period, price, total_load,dprice,log_dcons)
DAN_data_diff %>% select(dprice) %>% autoplot()
```

```
## Plot variable not specified, automatically selected `.vars = dprice`
```




```
DAN_data_diff %>% select(log_dcons) %>% autoplot()
```

```
## Plot variable not specified, automatically selected ‘.vars = log_dcons‘
```




```r
varmod = DAN_data_diff %>%
  model(
    mod1 = VAR(vars(dprice, log_dcons))
  )
varmod2 = DAN_data_diff %>%
  model(
    mod2 = VAR(vars(price, total_load))
  )
varmod %>% report()
```

```
## Series: dprice, log_dcons
## Model: VAR(5)
##
## Coefficients for dprice:
##       lag(dprice,1)  lag(log_dcons,1)  lag(dprice,2)  lag(log_dcons,2)
##              0.3304            5.8563        -0.0127           -5.4569
## s.e.         0.0109            1.3342         0.0117            1.7484
##       lag(dprice,3)  lag(log_dcons,3)  lag(dprice,4)  lag(log_dcons,4)
##             -0.1250            8.6842        -0.1061           -3.7577
## s.e.         0.0117            1.7915         0.0118            1.7442
##       lag(dprice,5)  lag(log_dcons,5)
##             -0.0809           -8.8356
## s.e.         0.0114            1.2462
##
## Coefficients for log_dcons:
```

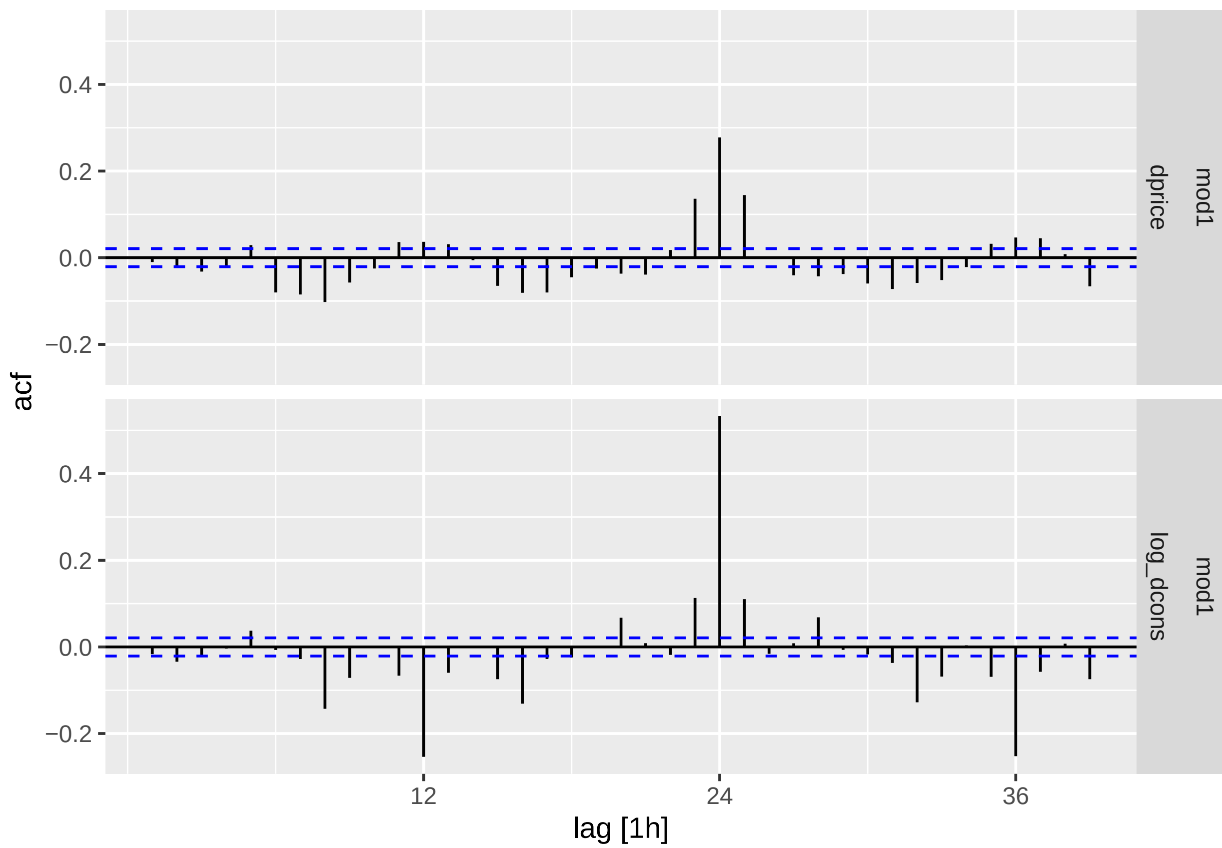
```
##       lag(dprice,1)  lag(log_dcons,1)  lag(dprice,2)  lag(log_dcons,2)
##              0.0025            0.8471          8e-04           -0.3418
## s.e.         0.0001            0.0108          1e-04            0.0141
##       lag(dprice,3)  lag(log_dcons,3)  lag(dprice,4)  lag(log_dcons,4)
##              -3e-04            0.1719         -1e-04            0.0572
## s.e.          1e-04            0.0145          1e-04            0.0141
##       lag(dprice,5)  lag(log_dcons,5)
##              -3e-04           -0.1648
## s.e.          1e-04            0.0101
##
## Residual covariance matrix:
##            dprice log_dcons
## dprice    17.3836    0.0284
## log_dcons  0.0284    0.0011
##
## log likelihood = -7456.95
## AIC = 14961.91   AICc = 14962.05 BIC = 15131.7
```

```
varmod2 %>% report()
```

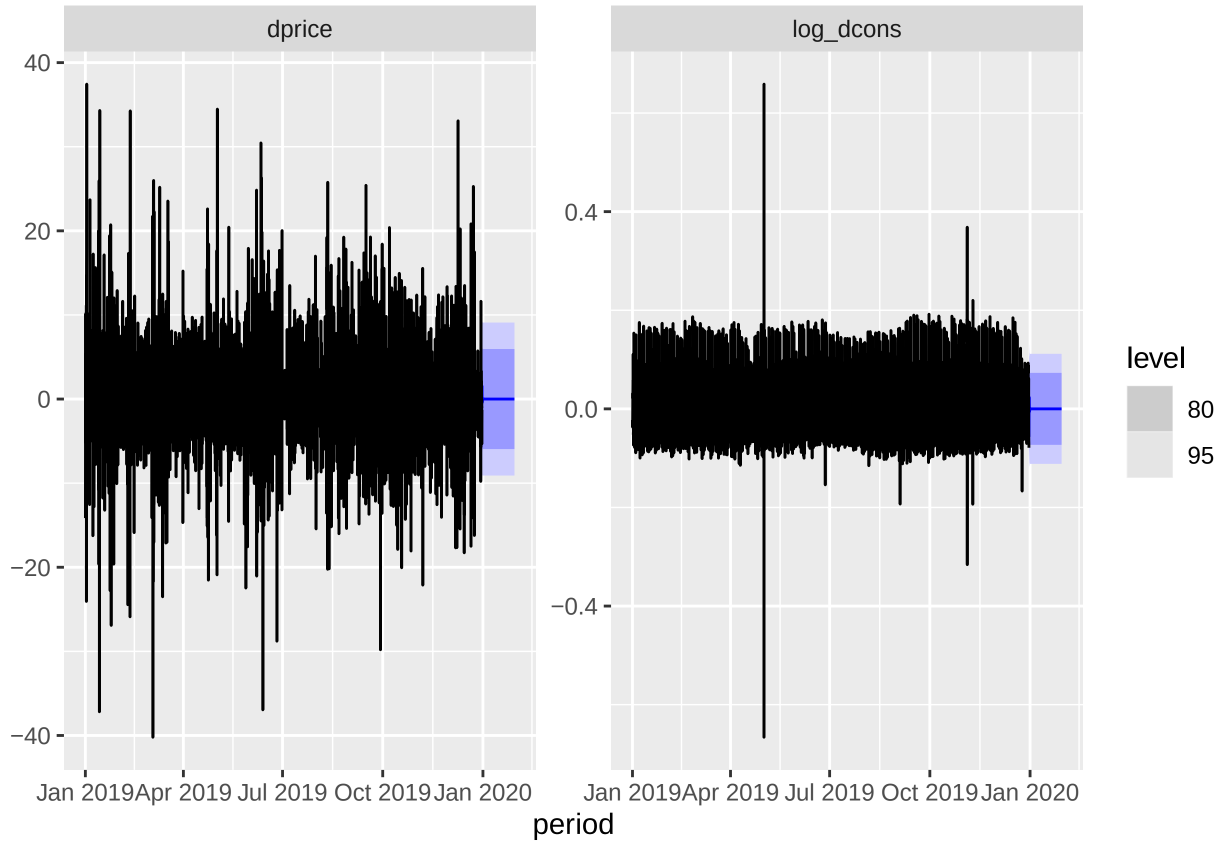
```
## Series: price, total_load
## Model: VAR(5) w/ mean
##
## Coefficients for price:
##       lag(price,1)  lag(total_load,1)  lag(price,2)  lag(total_load,2)
##             1.3084             4e-04       -0.3350            -0.0015
## s.e.        0.0109             3e-04        0.0177             0.0007
##       lag(price,3)  lag(total_load,3)  lag(price,4)  lag(total_load,4)
##            -0.1199            0.0034        -0.001            -0.0044
## s.e.        0.0181            0.0007         0.018             0.0007
##       lag(price,5)  lag(total_load,5)  constant
##             0.0767             2e-03    3.4157
## s.e.        0.0114             3e-04    0.2916
##
## Coefficients for total_load:
##       lag(price,1)  lag(total_load,1)  lag(price,2)  lag(total_load,2)
##            10.2512            1.6529       -5.6549            -0.9634
## s.e.        0.3402            0.0109        0.5559             0.0210
##       lag(price,3)  lag(total_load,3)  lag(price,4)  lag(total_load,4)
##            -4.0151            0.4016        0.1255            -0.1295
## s.e.        0.5673            0.0230        0.5627             0.0207
##       lag(price,5)  lag(total_load,5)  constant
##             0.1500           -0.0467  292.6803
## s.e.        0.3589            0.0101    9.1396
##
## Residual covariance matrix:
##               price total_load
## price       16.9157   102.4901
## total_load 102.4901 16622.3355
##
## log likelihood = -79364.01
## AIC = 158780 AICc = 158780.2 BIC = 158964
```

It's such a horrible model. The forecasting brough me back here to check. Where did we go wrong?

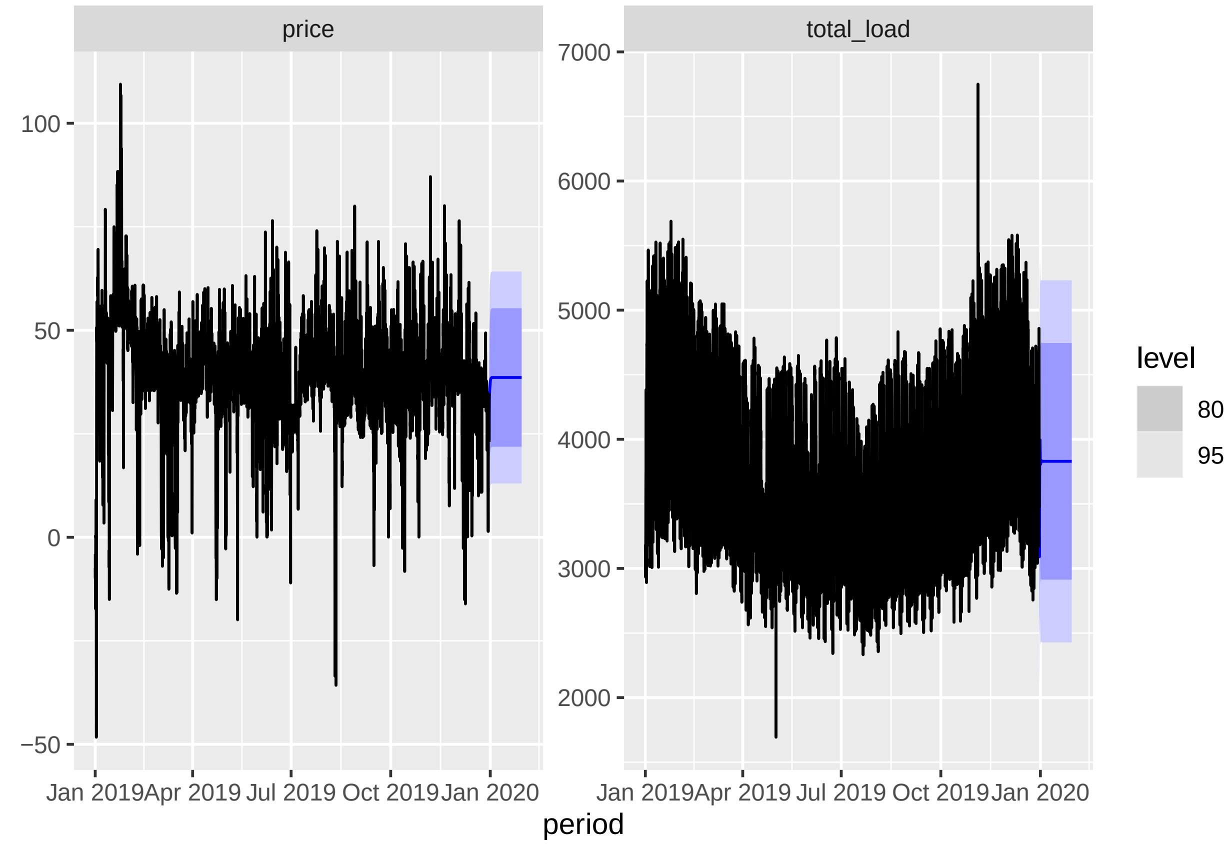
```
varmod %>%
  augment() %>%
  ACF(.innov) %>%
  autoplot() #failed to account for daily seasonality (or 12 hours for that matter)
```




```
varmod %>%
  forecast(h=30*24) %>%
  autoplot(DAN_data_diff)
```




```
varmod2 %>%
  forecast(h=30*24) %>%
  autoplot(DAN_data_diff)
```



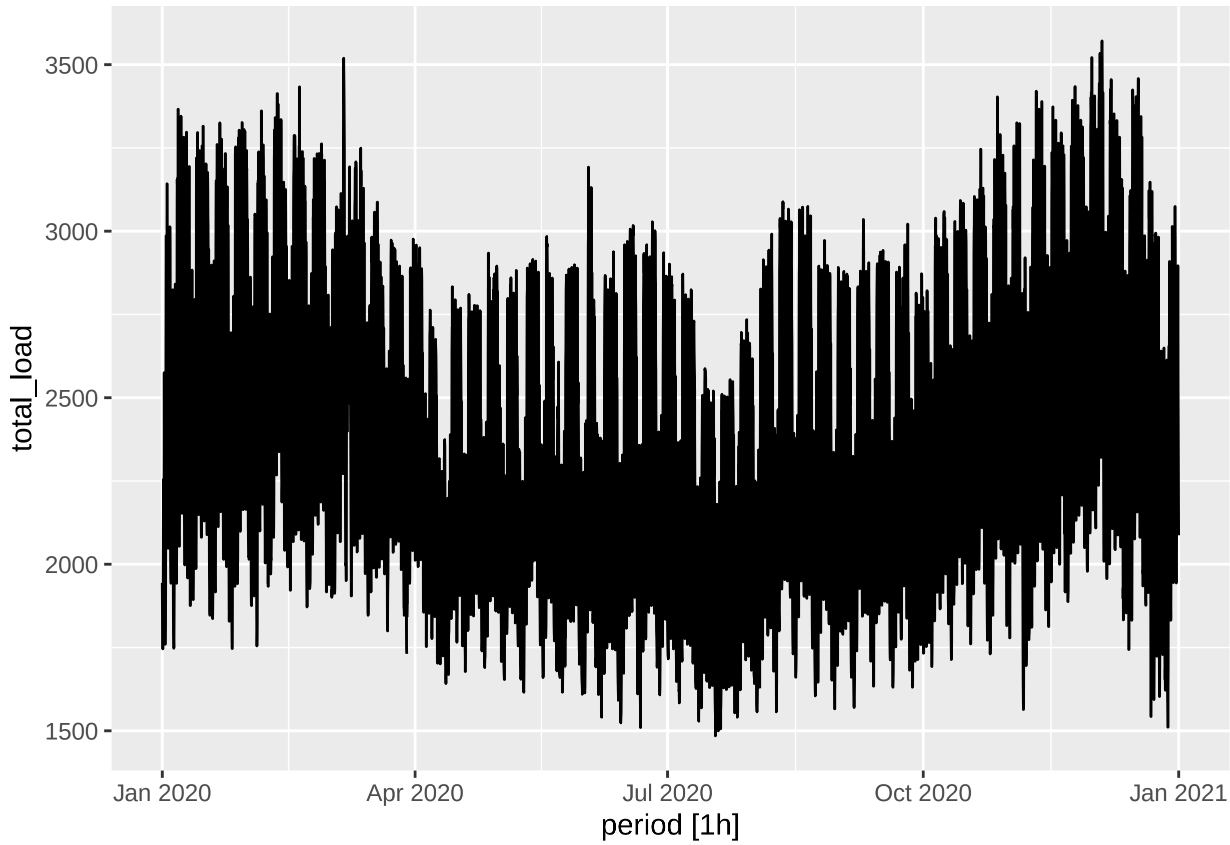
Why are the forecasts straight lines? This seem absolutely wrong. The expected difference is the mean, pretty much.

```
DAN_2020 <- read.csv('https://raw.githubusercontent.com/emcchri5/codebank/main/consumption%202020.csv')
DAN_2020 <- DAN_2020 %>%
  rename('total_load' = 'Actual.Total.Load..MW....BZN.DK1', 'datetime' = 'Time..CET.CEST.') %>%
  select(datetime, total_load)
DAN_2020 <- DAN_2020 %>%  separate('datetime', sep = ' - ', into=c('start','end')) %>%
  separate('start', sep = ' ', into = c('date', 'time')) %>%
  select(-end)

DAN_2020 <- DAN_2020 %>%
  separate('time', sep = ':', into=c('hour','minute'))

DAN_2020["period"] = dmy_h(paste(DAN_2020$date, DAN_2020$hour))
DAN_2020 <- DAN_2020 %>%
  select(-hour, -minute, -date)
dupRow = duplicates(DAN_2020)[2,]
```

```
## Using `period` as index variable.
```

```
DAN_2020 = DAN_2020 %>% rows_delete(dupRow, by=c("period", 'total_load'))
DAN_2020 <- DAN_2020 %>%
  tsibble(index = 'period')
DAN_2020 %>% select(total_load) %>% autoplot()
```

```
## Plot variable not specified, automatically selected `.vars = total_load`
```



I can't perform filters on this tsibble format… also our 95% confidence interval didn't work for January. I think it's safe to say we've failed this question. Where did we go wrong? Why is our prediction flat? How are these predictions supposed to look? Maybe a phone call/zoom meeting/etc. could help clear this up. I'm not exactly sure how submissions work in this class and how big of a deal this inadequacy is, but we hope that this can be a learning opportunity rather than a learning hindrance!