Introduction to Time Series Forecasting

BOUN ETM-58D Spring 2018

May 7, 2018

Contents

What is time series data? 1

Forecasting 6

- Fundamental Methods . . . 7
  - Autocorrelation . . . 7
  - Time Series Decomposition . . . 9
  - Moving Averages . . . 10
  - Autoregression . . . 11
- Forecasting Errors . . . 13

Some models 14

- ARIMA . . . 14
  - `auto.arima` in R . . . 14
- ETS . . . 15
  - Simple Exponential Smoothing . . . 15
  - Holt's Method . . . 15
  - Holt-Winter's Model . . . 16
  - Auto ETS . . . 17
- Prophet . . . 17

What is time series data?

The title time series comes from the data having an ordered index (e.g. days, weeks, quarters), usually (and preferably) with equal distances. It has the concept of future, now and past; which means, you cannot randomly sample from the data like you do .

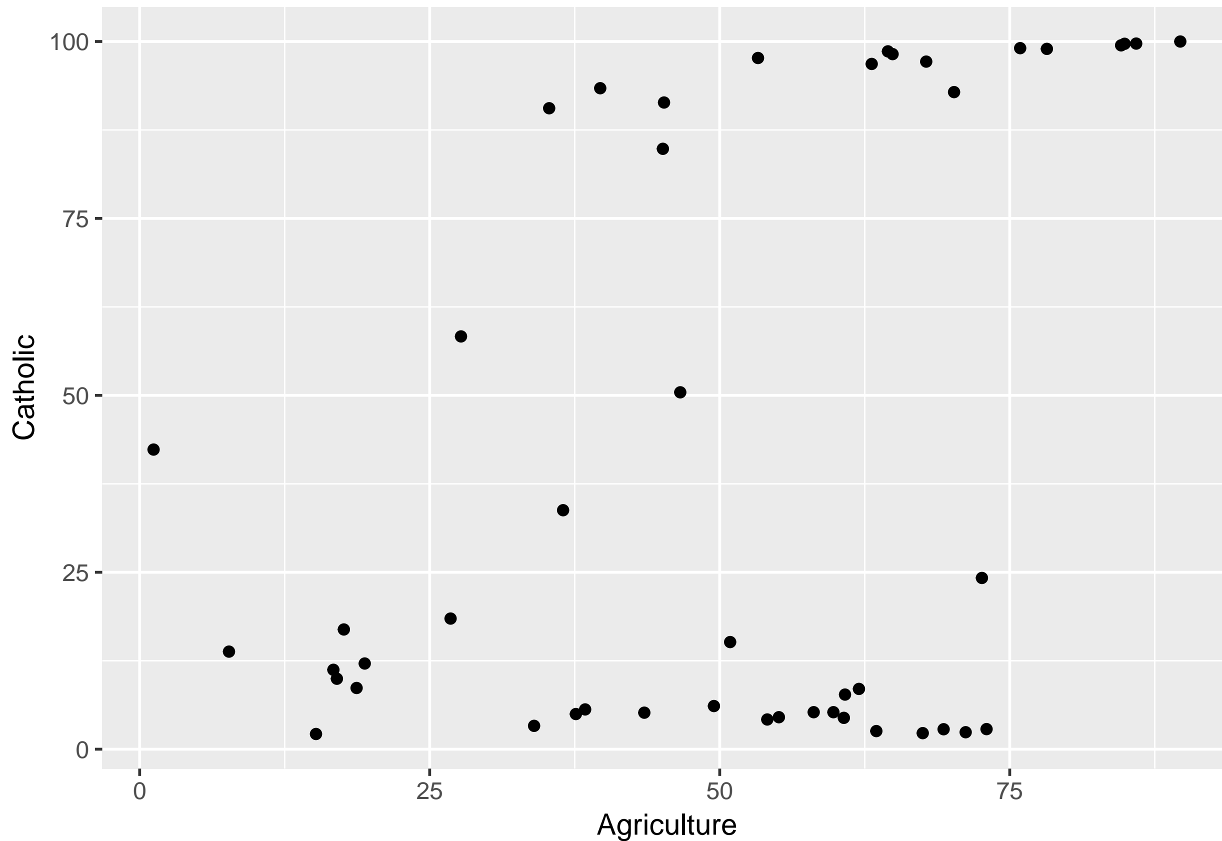
Let's start with some non-time series data, such as the `swiss` data (it is included in base R). You will see the indexes are Swiss regions and you can order them in any way you want or take random samples.

```r
head(swiss[,c("Agriculture","Catholic")])
```

```
##              Agriculture Catholic
## Courtelary          17.0     9.96
## Delemont            45.1    84.84
## Franches-Mnt        39.7    93.40
```

```
## Moutier               36.5    33.77
## Neuveville            43.5     5.16
## Porrentruy            35.3    90.57
```

```
ggplot(swiss,aes(x=Agriculture,y=Catholic)) + geom_point()
```



Now, let's see some time series data. The data we are going to use is electricity consumption data which can be found in EPİAŞ Şeffaflık / EXIST Transparency platform (click here). You can also **download** the sample data from 2015 to yesterday (May 6, 2018) from here.

Let's check the consumption data. The first column is a time index in "year-month-day hour:minute:second" format and the second column is the value column, in this case hourly electricity consumption of Turkey.

```
print(consumption_data)
```

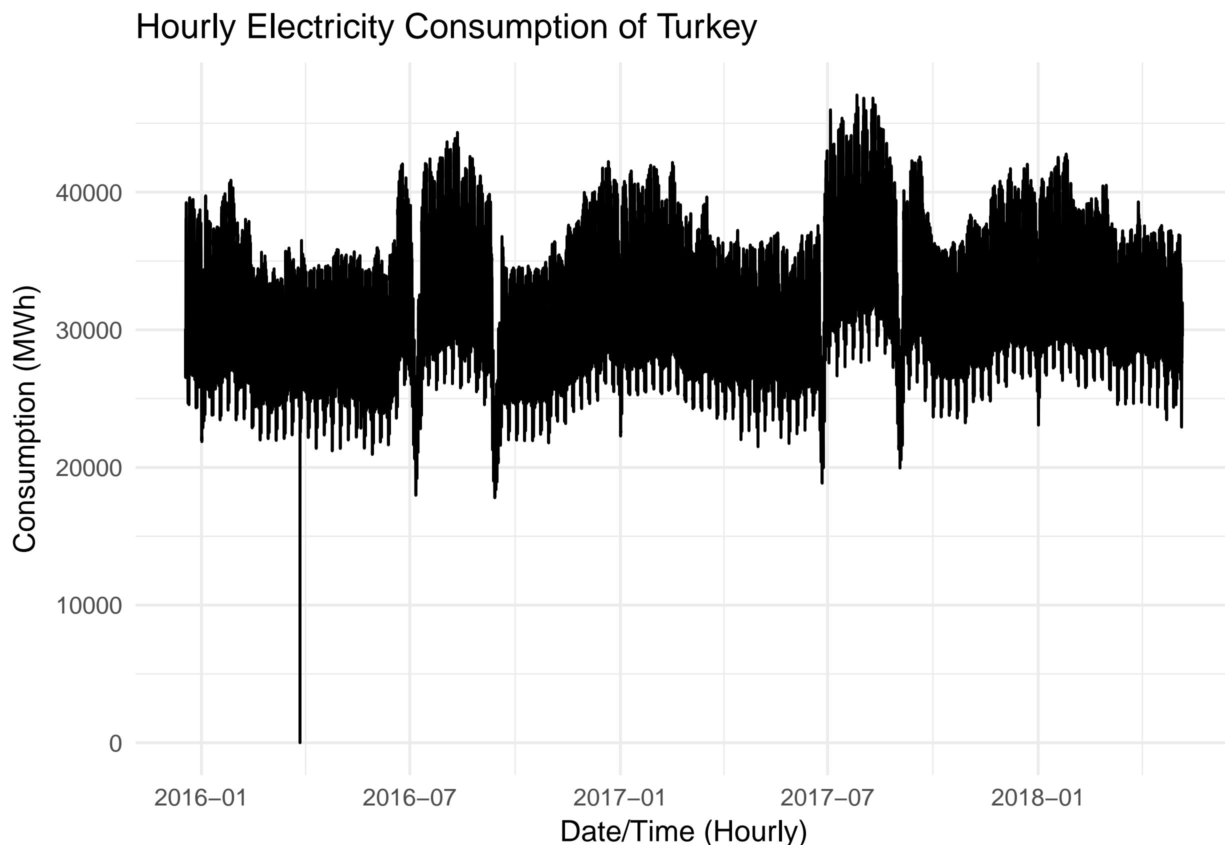
```
## # A tibble: 20,904 x 2
##    date_time           consumption
##    <dttm>                    <dbl>
##  1 2015-12-18 00:00:00       30070
##  2 2015-12-18 01:00:00       28311
##  3 2015-12-18 02:00:00       27351
##  4 2015-12-18 03:00:00       26740
##  5 2015-12-18 04:00:00       26526
##  6 2015-12-18 05:00:00       27230
##  7 2015-12-18 06:00:00       28319
##  8 2015-12-18 07:00:00       29790
##  9 2015-12-18 08:00:00       34311
```

```
## 10 2015-12-18 09:00:00       37337
## # ... with 20,894 more rows
```

It is also useful to know some date-time transformations to work with the data but we will not cover the topic here. Instead you can learn quickly from these two sources date time with R, lubridate tutorial. This tutorial uses `lubridate` functions (they are amazing).

Let's plot the consumption data. This time, instead of the scatter plot, we use the line plot. Because every data point is connected to previous and next data points.

```
# consumption_data <-
#     readr::read_csv("~/Downloads/GercekZamanliTuketim-01012015-06052018.csv",
#                         skip=1,col_names=FALSE,
#                     locale=readr::locale(decimal_mark=",",grouping_mark=".",tz="Turkey")) %>% rename(Date
# save(consumption_data,file="/Users/berkorbay/Dropbox/Courses_given/BOUN_ETM_58D_2017S/Lecture_Notes/w


ggplot(consumption_data,aes(x=date_time,y=consumption)) +
    labs(x="Date/Time (Hourly)",
         y="Consumption (MWh)",
         title="Hourly Electricity Consumption of Turkey") +
    geom_line() +
    theme_minimal()
```

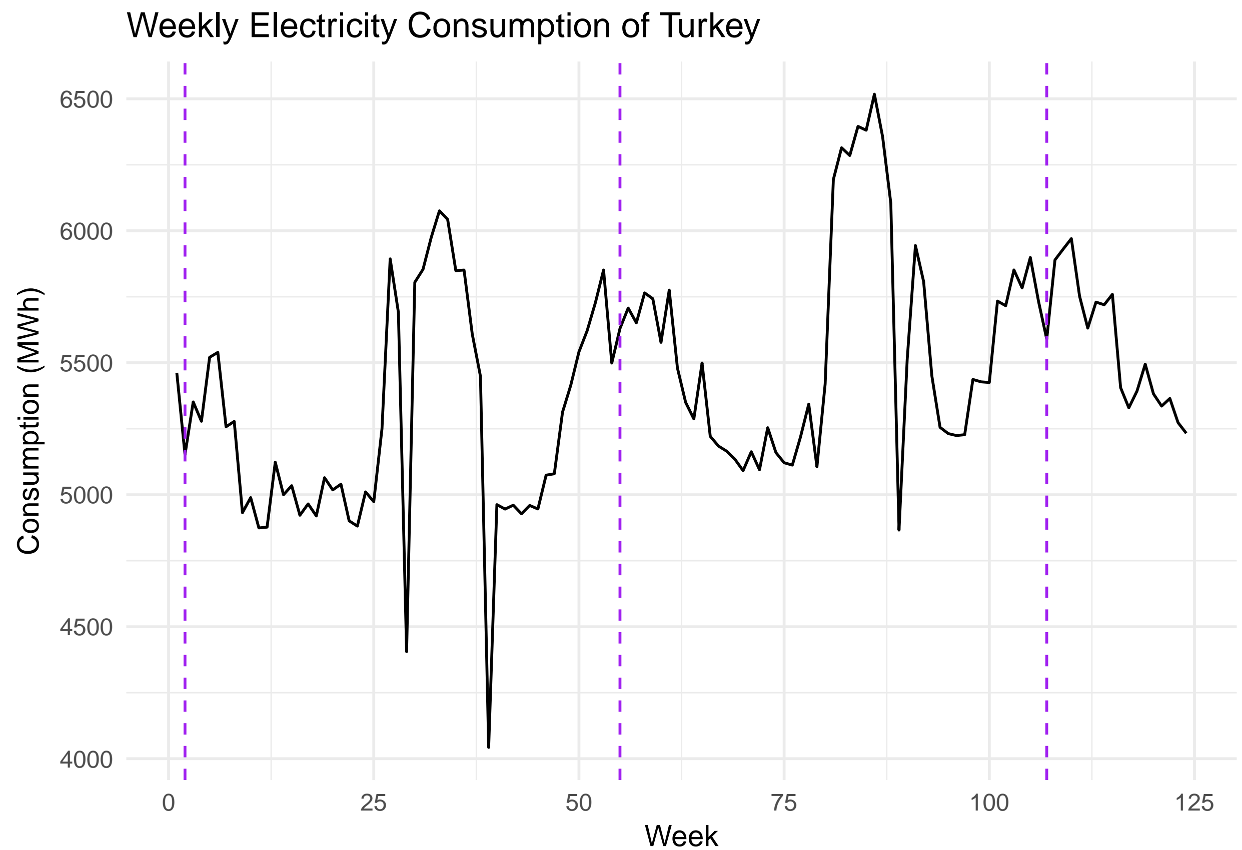


Time series data is messy. Do you see the zero value? It is because of the summer/winter time shifts. You need to clean or fix this kind of pecularities before you can work with the time series data. But instead, today we are going to aggregate consumption data to weekly GWh values from hourly MWh data.

```r
consumption_weekly <-
    consumption_data %>%
    group_by(Date=lubridate::as_date(date_time)) %>%
    summarise(consumption=sum(consumption,na.rm=T)/1000) %>%
    ungroup() %>%
    filter(Date >= "2015-12-21") %>%
    # arrange(desc(Date)) %>%
    mutate(wday=lubridate::wday(Date,week_start=1)) %>%
    group_by(wday) %>%
    mutate(weeknum=row_number()) %>%
    ungroup() %>%
    group_by(weeknum) %>%
    summarise(consumption=sum(consumption,na.rm=T)) %>%
    ungroup()
```

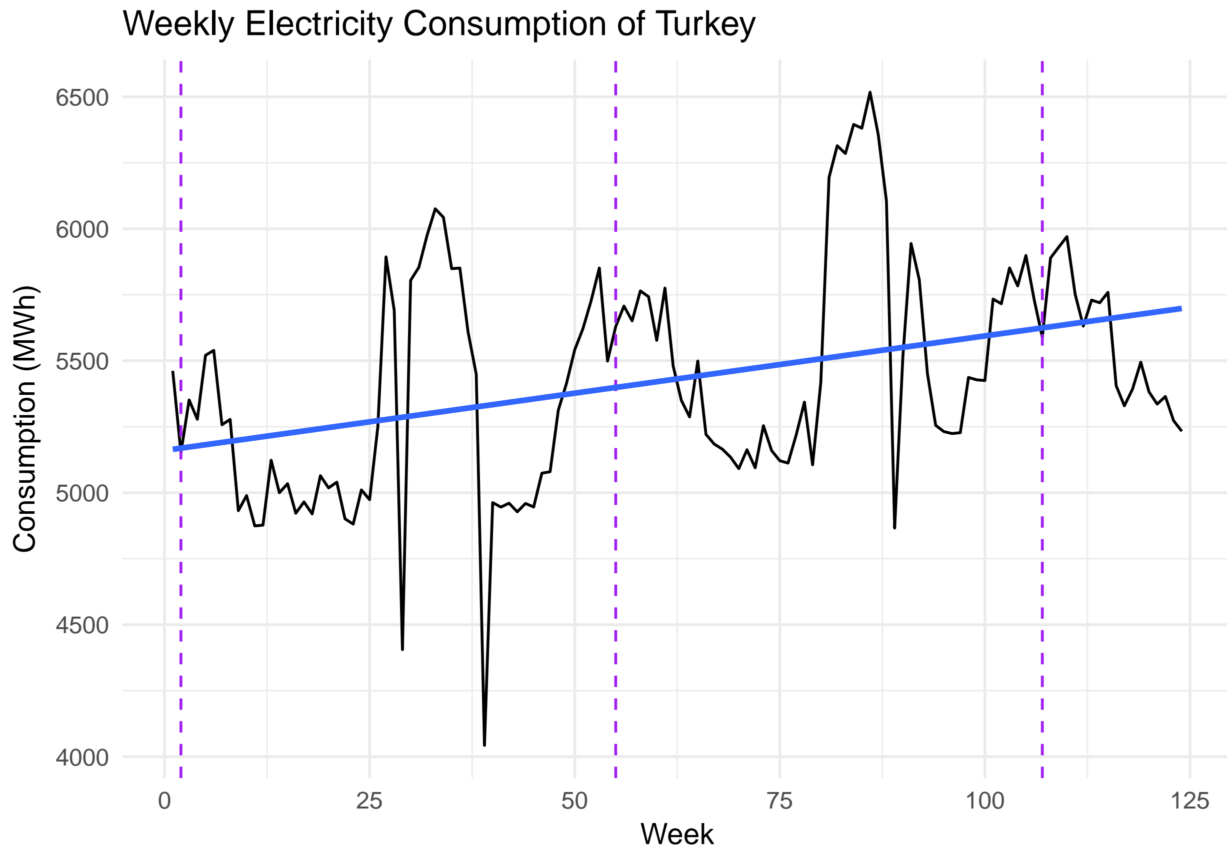
The reason we took such a lengthy transformation process is to make sure every week is exactly 7 days. Now let's plot our weekly data. It is a fun roller coaster (i.e. seasonality). Purple dashed lines indicate year starts.

```r
cons_plot <-
ggplot(consumption_weekly,aes(x=weeknum,y=consumption)) +
    labs(x="Week",
         y="Consumption (MWh)",
         title="Weekly Electricity Consumption of Turkey") +
    geom_line() +
    geom_vline(xintercept=c(2,55,107),color="purple",size=0.5,linetype="dashed") +
    theme_minimal()

cons_plot
```



Time series data has its own characteristics. Let's start with identifying the sudden consumption decreases. What might they be? It is always possible that data is wrong. Also check whether consumption is increasing over time (i.e. trend).

```
cons_plot + geom_smooth(method="lm",se=FALSE)
```



Even though they can be used for other purposes such as anomaly detection, the main purpose of time series analysis is forecasting. Forecasts are made due to our need to know about the future. What will be the weather tomorrow? What will be the stock market next year? How many units can I sell in the next quarter? If we can see the future with some degree of clarity, we can plan ahead. For instance, we can arrange our production capacity, invest in the most profitable stocks or just take an umbrella on the way out.

Forecasting is never easy and it is rarely stationary. There are many methods and pitfalls. Currently we increasingly use sophisticated tools and mathematical models in forecasting but in the past it was all about human intuition or divination methods such as haruspex.

Forecasting

Forecasting starts with the analysis and identification of components. Fundamental components are trend (e.g. increasing consumption) and seasonality (e.g. ice cream sales). There is also cyclicality (e.g. bear and bull markets). There are special days (e.g. holidays) and their interactions (e.g. weekend vs 4-day holiday). Other external variables are also important (e.g. temperature, daylight). Events (e.g. discount campaigns) according to forecasts can also affect the forecast.

Beware: We also have noise, some randomness. Sometimes, it seems like it can be modeled but you cannot actually. It might also be hard to distinguish noise from signal.

You can (try to) forecast almost anything that will happen in the future and R is your best friend. There are amazing R functions making your life way easier and superb R packages for almost any kind of forecasting method. `forecast` package of Rob Hyndman and its associated book (see here) are very fine resources which we will partly follow here.

So let's load the libraries. Install them if you do not have the libraries.

```r
library(forecast)
library(fma)
library(fpp)
```

Fundamental Methods

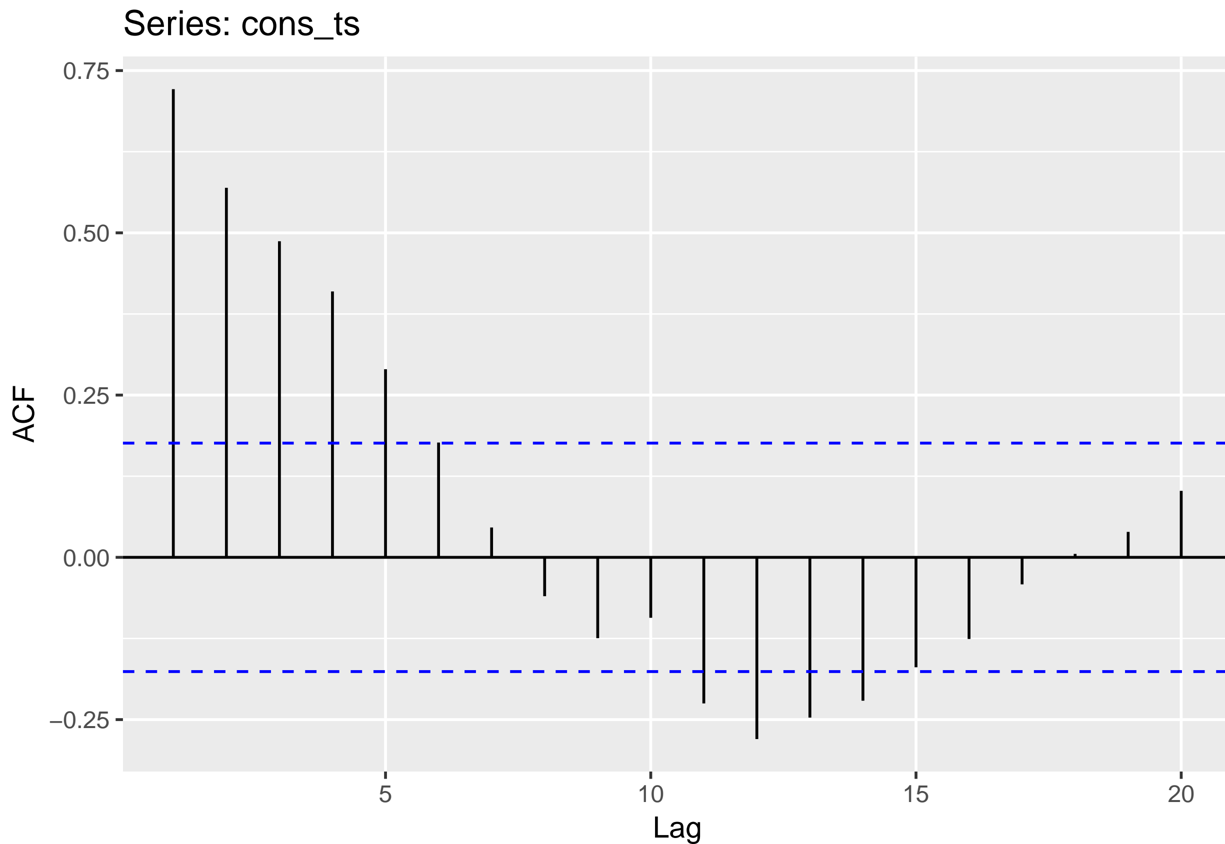
Autocorrelation

Autocorrelation is correlation but instead of among different variables, we check for correlation between lagged series (e.g. today vs yesterday) of the same variable.

```r
cons_ts <- as.ts(consumption_weekly$consumption,deltat=1/52,frequency=52)
print(cons_ts)
```

```
## Time Series:
## Start = 1
## End = 124
## Frequency = 1
##   [1] 5461.870 5151.477 5351.725 5278.316 5520.635 5539.354 5257.580
##   [8] 5277.803 4931.996 4989.307 4874.317 4877.194 5123.416 5000.016
##  [15] 5034.354 4922.256 4965.309 4919.686 5064.609 5018.301 5040.058
##  [22] 4901.160 4881.155 5010.810 4973.707 5249.778 5893.636 5691.209
##  [29] 4405.583 5804.768 5853.943 5974.774 6075.727 6043.160 5848.992
##  [36] 5851.157 5608.761 5448.794 4042.791 4962.589 4945.646 4960.470
##  [43] 4927.765 4959.392 4946.075 5074.122 5079.516 5312.894 5414.136
##  [50] 5541.272 5621.493 5727.062 5851.498 5498.768 5629.653 5707.258
##  [57] 5651.176 5764.715 5742.655 5577.219 5775.314 5479.987 5350.096
##  [64] 5287.304 5499.080 5221.240 5184.441 5165.101 5134.689 5091.152
##  [71] 5162.964 5094.551 5254.048 5159.827 5121.293 5112.272 5219.161
##  [78] 5343.421 5105.879 5419.578 6194.232 6314.642 6285.170 6395.449
##  [85] 6380.908 6517.536 6356.117 6104.603 4866.081 5513.819 5944.302
##  [92] 5807.028 5450.772 5255.386 5231.599 5224.247 5227.445 5436.747
##  [99] 5427.664 5425.230 5733.742 5716.017 5851.673 5783.542 5898.963
## [106] 5732.149 5589.169 5889.455 5930.545 5970.350 5751.984 5631.136
## [113] 5729.768 5719.665 5759.217 5405.711 5329.232 5393.010 5494.758
## [120] 5382.244 5335.816 5364.685 5273.187 5232.958
```

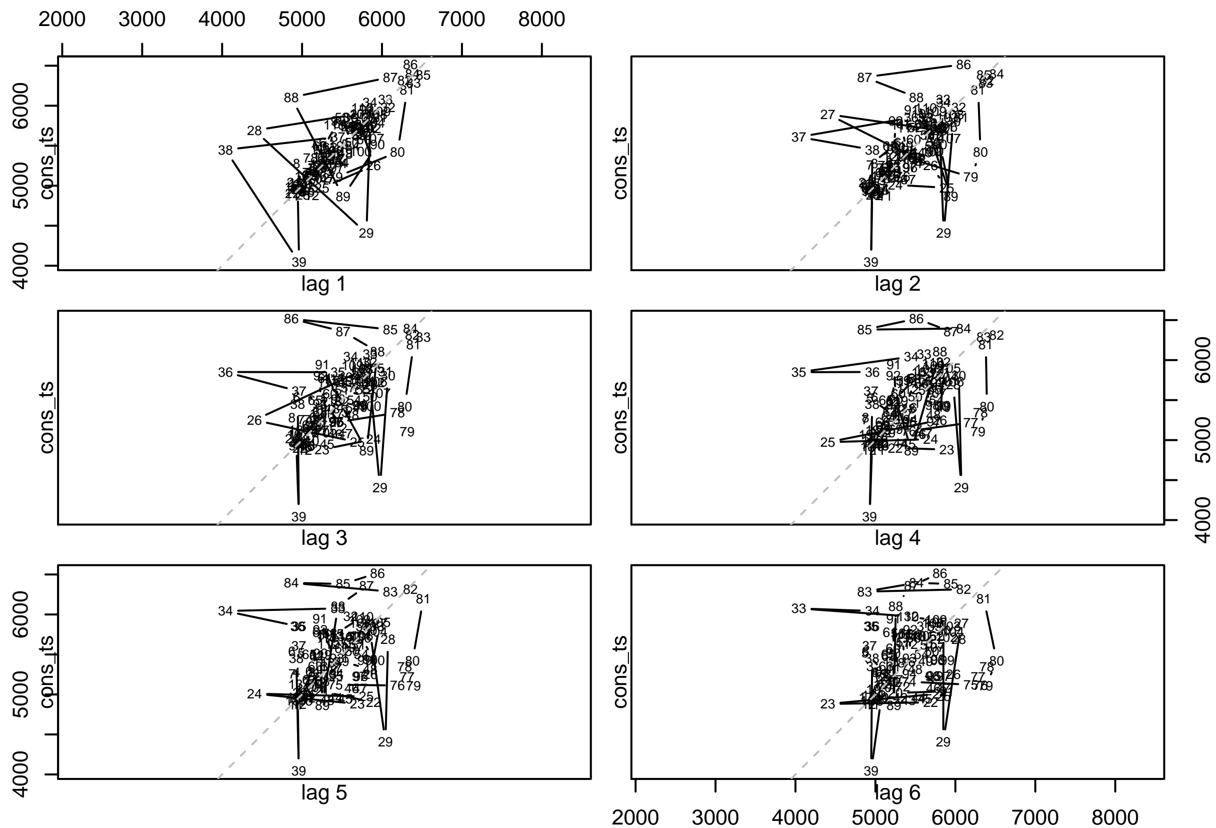
```r
ggAcf(cons_ts)
```



Basically, between the blue lines are insignificant indicators of autocorrelation. In our case, up to lag 5 we can say there is some degree of autocorrelation. Between 11 and 14 lags, we see negative values meaning the turn of the cyclical activity.

Now let's draw the lag plots. Lag plots are scatterplots that show the level of the consumption vs X periods (e.g. weeks) before.

```
lag.plot(cons_ts, lags=6, do.lines=TRUE)
```



Time Series Decomposition

We talked about the seasonality and trend components of the time series. Now let's build a new time series object giving the frequency and starting points.

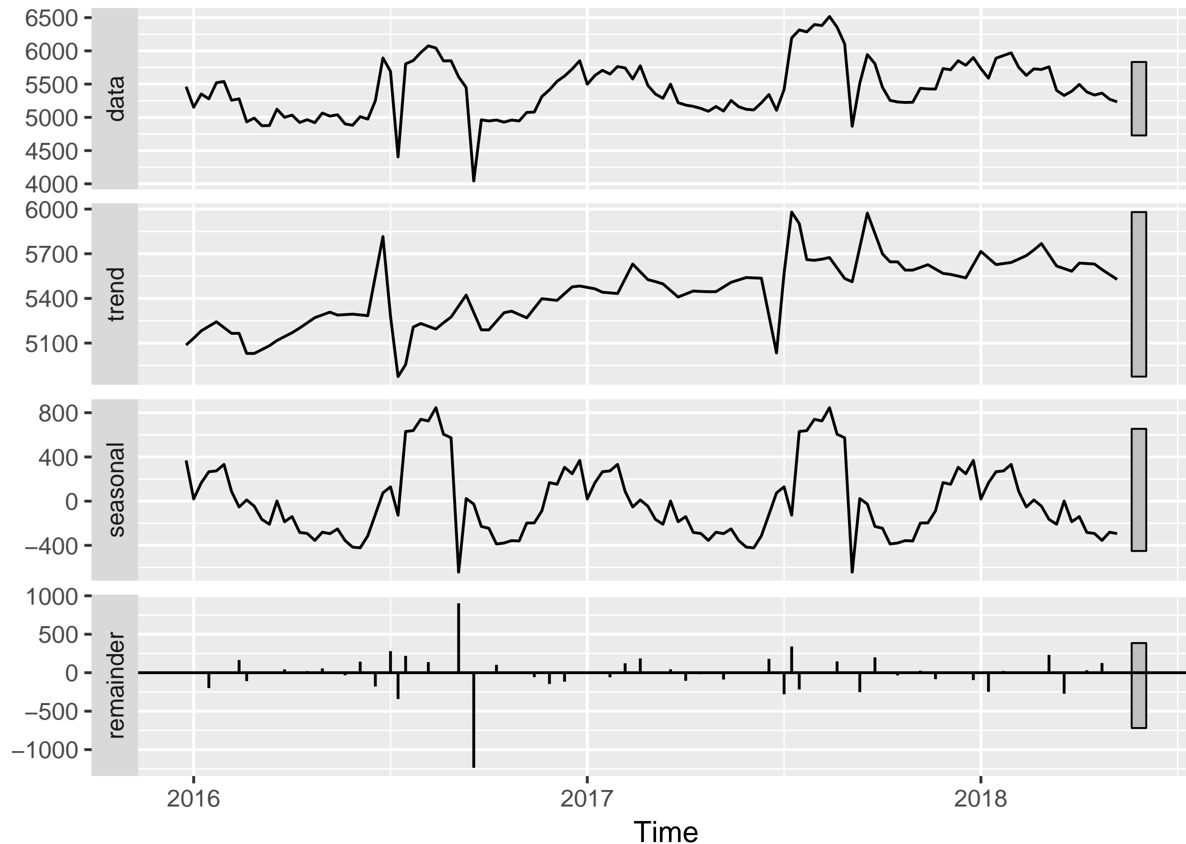
```
cons_ts2<-ts(consumption_weekly$consumption,start=c(2015,52),frequency=52)
print(cons_ts2)
```

```
## Time Series:
## Start = c(2015, 52)
## End = c(2018, 19)
## Frequency = 52
##   [1] 5461.870 5151.477 5351.725 5278.316 5520.635 5539.354 5257.580
##   [8] 5277.803 4931.996 4989.307 4874.317 4877.194 5123.416 5000.016
##  [15] 5034.354 4922.256 4965.309 4919.686 5064.609 5018.301 5040.058
##  [22] 4901.160 4881.155 5010.810 4973.707 5249.778 5893.636 5691.209
##  [29] 4405.583 5804.768 5853.943 5974.774 6075.727 6043.160 5848.992
##  [36] 5851.157 5608.761 5448.794 4042.791 4962.589 4945.646 4960.470
##  [43] 4927.765 4959.392 4946.075 5074.122 5079.516 5312.894 5414.136
##  [50] 5541.272 5621.493 5727.062 5851.498 5498.768 5629.653 5707.258
##  [57] 5651.176 5764.715 5742.655 5577.219 5775.314 5479.987 5350.096
##  [64] 5287.304 5499.080 5221.240 5184.441 5165.101 5134.689 5091.152
##  [71] 5162.964 5094.551 5254.048 5159.827 5121.293 5112.272 5219.161
##  [78] 5343.421 5105.879 5419.578 6194.232 6314.642 6285.170 6395.449
##  [85] 6380.908 6517.536 6356.117 6104.603 4866.081 5513.819 5944.302
##  [92] 5807.028 5450.772 5255.386 5231.599 5224.247 5227.445 5436.747
```

```
##  [99] 5427.664 5425.230 5733.742 5716.017 5851.673 5783.542 5898.963
## [106] 5732.149 5589.169 5889.455 5930.545 5970.350 5751.984 5631.136
## [113] 5729.768 5719.665 5759.217 5405.711 5329.232 5393.010 5494.758
## [120] 5382.244 5335.816 5364.685 5273.187 5232.958
```

We are going to use a method called STL (Seasonal and Trend decomposition using Loess). Loess (Local Weighted Scatterplot Smoother) is the nonlinear smoothing curve; similar to regression, but obviously nonlinear.

```
stl(cons_ts2,t.window=5,s.window="periodic", robust=TRUE) %>% autoplot()
```



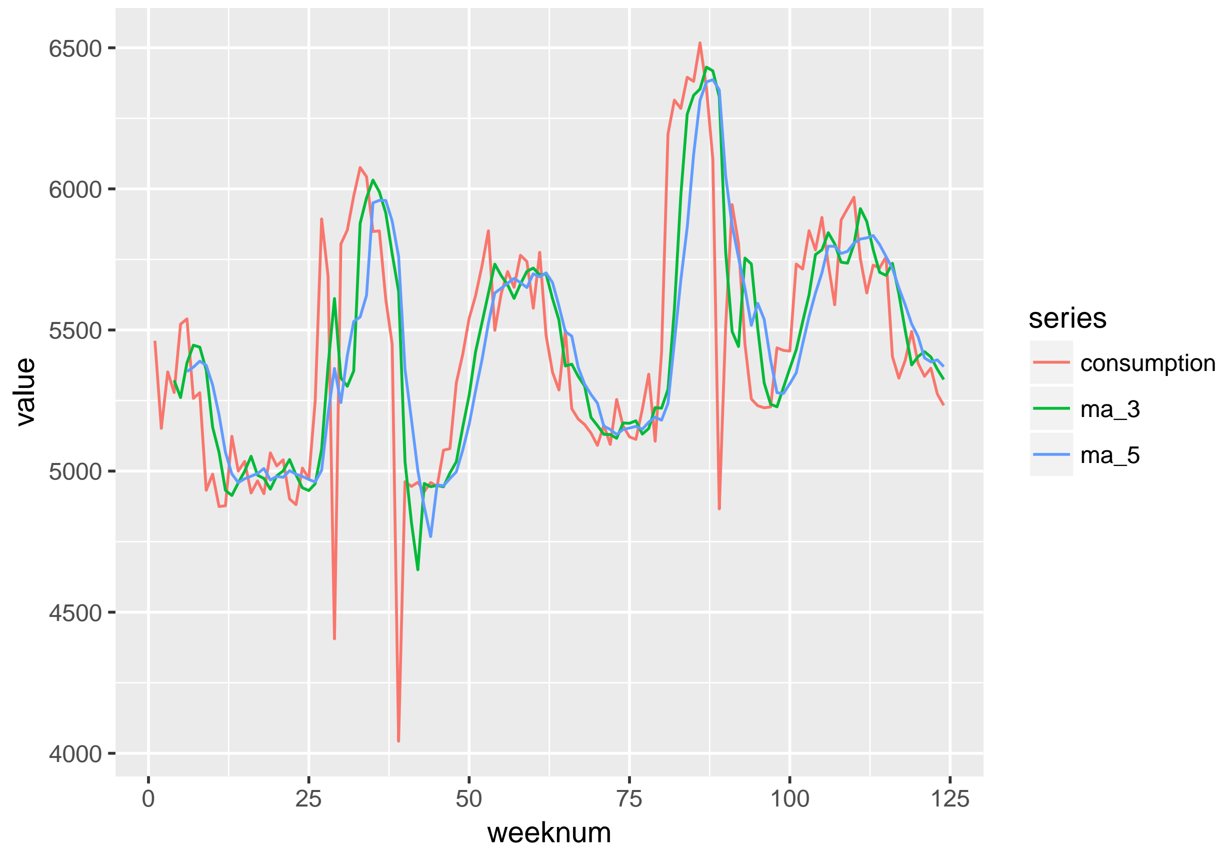
STL is actually not bad at all. There are some specific errors on localized points but the rest look like ok.

Moving Averages

Simple moving average ($MA(X)$) just takes the last X observation's average. Let's try to forecast the next week with MA(3) and MA(5). The forecast seems a bit lagged, isn't it?

```
the_plot <-
consumption_weekly %>%
    mutate(ma_3=lag(rollmean(consumption,3,align="right",fill=NA),1),ma_5=lag(rollmean(consumption,5,al
    gather(key=series,value,-weeknum) %>%
    ggplot(.,aes(x=weeknum,y=value,color=series)) +
    geom_line()

the_plot
```

```
## Warning: Removed 8 rows containing missing values (geom_path).
```



Autoregression

Autoregression is regression with lagged values of the time series. The formula for AR(k) is given below.

$$x_t = \beta_0 + \beta_1 x_{t-1} + \beta_1 x_{t-2} + \cdots + \beta_1 x_{t-k} + \epsilon$$

You can calculate AR(k) with `ar` function. Let's try AR(5). The model spits out the coefficients.

```
the_model <- ar(cons_ts,FALSE,5)
print(the_model)
```

```
##
## Call:
## ar(x = cons_ts, aic = FALSE, order.max = 5)
##
## Coefficients:
##       1        2        3        4        5
##  0.6382   0.0496   0.0879   0.0838  -0.1063
##
## Order selected 5  sigma^2 estimated as  86093
```

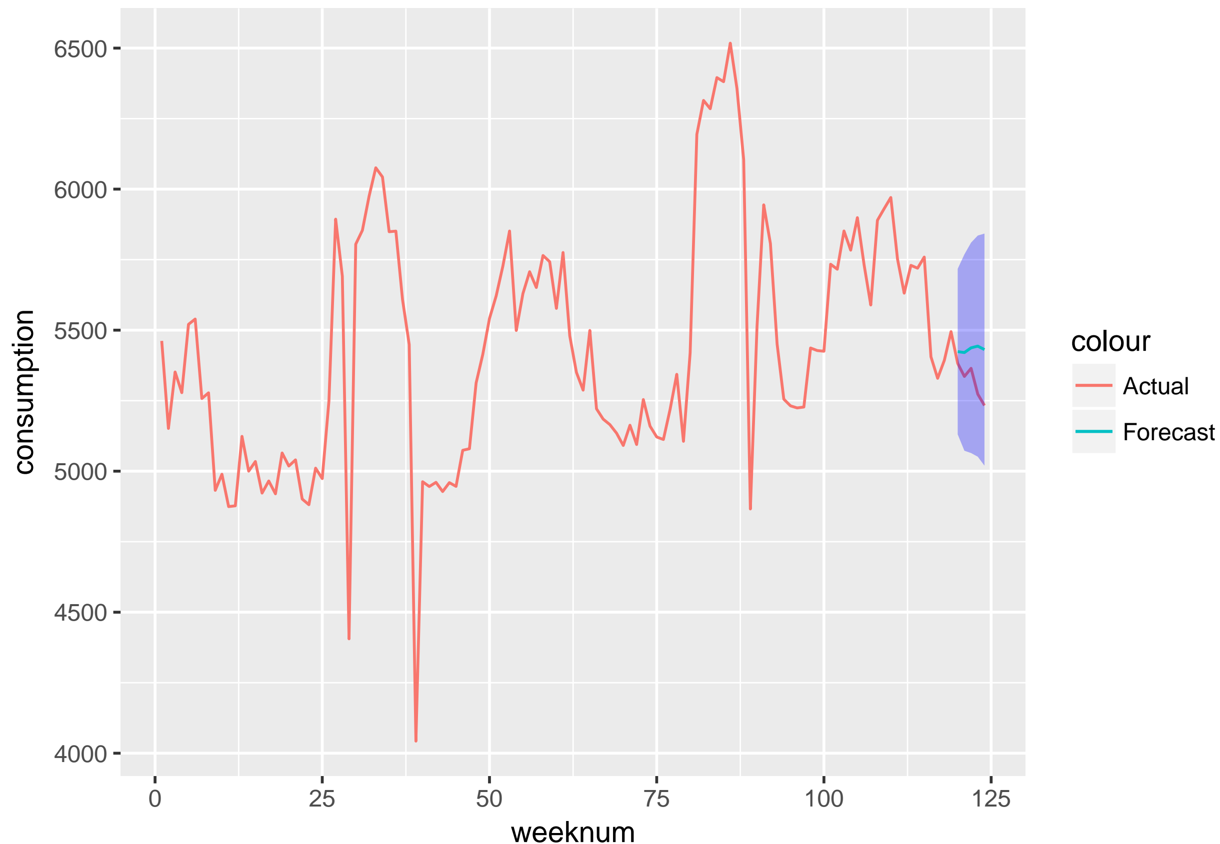
Let's predict the consumption for the last 5 periods.

```r
the_prediction <- predict(the_model,newdata=cons_ts[-(120:124)],n.ahead=5)
print(the_prediction)
```

```
## $pred
## Time Series:
## Start = 120
## End = 124
## Frequency = 1
## [1] 5423.962 5420.590 5437.342 5443.394 5431.044
##
## $se
## Time Series:
## Start = 120
## End = 124
## Frequency = 1
## [1] 293.4161 348.0843 373.0156 392.0446 411.4034
```

Now let's finally plot it vs the actual values.

```r
    the_plot <-
    consumption_weekly %>% left_join(.,
        tibble(weeknum=120:124,
            forecast=as.vector(the_prediction$pred),
            hi=forecast+the_prediction$se,lo=forecast-the_prediction$se),by="weeknum") %>%
        ggplot(.,aes(x=weeknum)) +
        geom_line(aes(y=consumption,color="Actual")) +
        geom_ribbon(aes(ymin=lo,ymax=hi),fill="blue",alpha=0.3) +
        geom_line(aes(y=forecast,color="Forecast"))

the_plot
```



Forecasting Errors

There are multiple error measures for forecasting errors. Some are meaningful in certain concepts, some are not. There are usually a few metrics analysts follow for their specific problems. Two main error types are absolute scale errors and relative scale errors.

In absolute scale errors, simple difference is (Actual-Forecast) calculated. The absolute of this error is Absolute Error. Mean of this absolute error is Mean Absolute Error (MAE).

Some cases require scaled values such as financial time series or very volatile base valued series. Then we scale the error with the actual value (e.g. (Actual-Forecast)/Actual). It is called relative or percentage error. Mean value of the absolute percentage error is called MAPE.

Here is a handy function to get the descriptive error statistics of your forecasts.

```
accu=function(actual,forecast){
  n=length(actual) #Length of the vector, number of periods
  error=actual-forecast #Also known as BIAS or deviance, NOT A METRIC
  meanval=mean(actual) #Mean Error
  stdev=sd(actual) #Standard Deviation of Error
  CV=stdev/meanval #Coefficient of Variation
  AD=abs(actual-meanval) #Absolute Deviation, NOT A METRIC
  R2=1-sum(error^2)/sum((actual-meanval)^2) #R-squared
  #AdjR2=1-(1-R2)*(n-1)/(n-k-1)
  DB=sum(diff(error)^2)/sum(error^2)
  #FE=sqrt(sum(error^2)/(n-k))
  FBias=sum(error)/sum(actual) #Relative total error
```

```
  MPE=sum(error/actual)/n #Mean Percentage Error
  MAPE=sum(abs(error/actual))/n #Mean Absolute Percentage Error
  RMSE=sqrt(sum(error^2)/n) #Root Mean Squared Error
  MAD=sum(abs(error))/n #Mean Absolute Deviation
  MADP=sum(abs(error))/sum(abs(actual)) #Mean Absolute Deviation Percentage
  MASE=MAD/mean(abs(diff(actual))) #Mean Absolute Standard Error
  RAE= sum(abs(error))/sum(abs(actual-meanval)) #Relative Absolute Error
  WMAPE=MAD/meanval #Weighted MAPE
  l=data.frame(n,meanval,stdev,CV,R2,DB,FBias,MPE,MAPE,RMSE,MAD,MADP,MASE,RAE,WMAPE)
  return(l)
}
```

Let's try the error measurement for in-sample estimation of STL.

```
stl_model <- stl(cons_ts2,t.window=5,s.window="periodic", robust=TRUE)
comparison_table <-
stl_model$time.series %>% as.tibble() %>% transmute(prediction=seasonal+trend) %>% bind_cols(consumptio
accu(comparison_table$consumption,comparison_table$prediction)
```

```
##     n  meanval    stdev         CV        R2     DB         FBias
## 1 124 5431.405 421.4474 0.07759455 0.8399875 2.2028 -0.0007076739
##            MPE       MAPE     RMSE      MAD       MADP      MASE       RAE
## 1 -0.001646773 0.01271655 167.9044 66.13872 0.01217709 0.3713248 0.1954779
##        WMAPE
## 1 0.01217709
```

Some models

ARIMA, ETS and Prophet models are briefly introduced in this part.

ARIMA

ARIMA(p,d,q) is Autoregression Integrated Moving Average model for time series forecasting. The model takes both autoregressive and moving average components. p is for AR part, d is for differencing and q is for MA part. See this tutorial for more information.

Some special cases of ARIMA are equivalent to ETS.

`auto.arima` in R

```
arima_model <- auto.arima(cons_ts)
summary(arima_model)
```

```
## Series: cons_ts
## ARIMA(0,1,1)
##
## Coefficients:
##           ma1
```

```
##       -0.3235
## s.e.   0.0948
##
## sigma^2 estimated as 91957:  log likelihood=-876.97
## AIC=1757.94   AICc=1758.04   BIC=1763.57
##
## Training set error measures:
##                     ME     RMSE      MAE        MPE    MAPE     MASE
## Training set -2.191192 300.7885 178.9205 -0.2665584 3.38283 1.004519
##                    ACF1
## Training set 0.03639094
```

For our time series data, ARIMA(0,1,1) is the best fit which means the first order differencing (I) with MA(1). Best ARIMA setting is determined by a metric called AIC (Akaike Information Criterion). AIC is the penalization of log-likelihood with the number of parameters used.

ETS

Exponential time series smoothing consist of three different types; exponential smoothing, double exponential smoothing and holt-winters. In the most basic model, the next period is determined by a weighted average of the current period and past periods.

$$\hat{y_{t+1}} = \alpha y_t + \alpha(1-\alpha)y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + \dots$$

The α parameter between 0 and 1 is called the smoothing parameter.

Simple Exponential Smoothing

Simple exponential smoothing always returns the last point forecast. Further periods are equivalent to the last forecast but the upper/lower limits are affected.

```
ses(cons_ts)
```

```
##     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 125       5255.688 4870.214 5641.161 4666.156 5845.219
## 126       5255.688 4790.992 5720.384 4544.996 5966.379
## 127       5255.688 4723.433 5787.942 4441.675 6069.700
## 128       5255.688 4663.533 5847.842 4350.065 6161.310
## 129       5255.688 4609.159 5902.216 4266.907 6244.468
## 130       5255.688 4559.015 5952.360 4190.220 6321.156
## 131       5255.688 4512.247 5999.129 4118.693 6392.682
## 132       5255.688 4468.251 6043.124 4051.407 6459.968
## 133       5255.688 4426.586 6084.789 3987.686 6523.689
## 134       5255.688 4386.917 6124.458 3927.018 6584.357
```

Holt's Method

Holt's method adds a trend parameter to SES. It is also called double exponential smoothing.

```r
holt(cons_ts)
```

```
##     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 125       5254.098 4868.598 5639.599 4664.526 5843.671
## 126       5253.019 4788.288 5717.751 4542.274 5963.765
## 127       5251.940 4719.625 5784.256 4437.834 6066.047
## 128       5250.862 4658.608 5843.115 4345.088 6156.635
## 129       5249.783 4603.107 5896.458 4260.778 6238.787
## 130       5248.704 4551.829 5945.578 4182.926 6314.481
## 131       5247.625 4503.919 5991.330 4110.225 6385.025
## 132       5246.546 4458.775 6034.317 4041.754 6451.337
## 133       5245.467 4415.956 6074.977 3976.840 6514.094
## 134       5244.388 4375.128 6113.648 3914.970 6573.806
```

Holt-Winter's Model

In this model we also add the seasonality parameters. We will use the monthly consumption data to forecast with Holt Winter's.

```r
consumption_monthly <- consumption_weekly %>% group_by(month=ceiling(weeknum/4)) %>% summarise(consumpt
cons_ts3 <- ts(consumption_monthly$consumption,start=c(2015,12),frequency=12)
print(cons_ts3)
```

```
##           Jan      Feb      Mar      Apr      May      Jun      Jul
## 2015
## 2016 21595.37 19672.81 20080.04 19967.91 19833.18 21808.33 22039.07
## 2017 22687.18 22735.76 21892.70 21069.86 20483.36 20647.44 21088.04
## 2018 23084.98 23109.74 23284.01 22614.36 21599.24 21206.65
##           Aug      Sep      Oct      Nov      Dec
## 2015                                     21243.39
## 2016 23819.04 20062.93 19793.27 20412.61 22303.96
## 2017 25189.49 25359.17 22131.23 21162.00 21517.09
## 2018
```

```r
hw(cons_ts3)
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## Jul 2018       23093.18 21678.44 24507.93 20929.51 25256.86
## Aug 2018       25962.95 24548.21 27377.70 23799.28 28126.63
## Sep 2018       24412.27 22997.52 25827.02 22248.60 26575.94
## Oct 2018       22395.15 20980.40 23809.90 20231.48 24558.82
## Nov 2018       22272.95 20858.20 23687.70 20109.27 24436.62
## Dec 2018       23501.18 22086.43 24915.93 21337.50 25664.85
## Jan 2019       24319.72 22904.97 25734.47 22156.04 26483.39
## Feb 2019       23896.28 22481.53 25311.03 21732.61 26059.96
## Mar 2019       23809.01 22394.25 25223.76 21645.33 25972.68
## Apr 2019       23240.26 21825.51 24655.02 21076.59 25403.94
## May 2019       22581.96 21167.21 23996.71 20418.28 24745.64
## Jun 2019       22998.65 21583.90 24413.41 20834.97 25162.33
## Jul 2019       24058.46 22495.49 25621.43 21668.11 26448.81
## Aug 2019       26928.23 25365.26 28491.20 24537.87 29318.58
```

```
## Sep 2019       25377.54 23814.57 26940.51 22987.19 27767.90
## Oct 2019       23360.42 21797.45 24923.40 20970.07 25750.78
## Nov 2019       23238.22 21675.25 24801.19 20847.86 25628.58
## Dec 2019       24466.45 22903.47 26029.43 22076.09 26856.82
## Jan 2020       25284.99 23722.01 26847.97 22894.62 27675.36
## Feb 2020       24861.56 23298.58 26424.54 22471.19 27251.93
## Mar 2020       24774.28 23211.30 26337.26 22383.90 27164.66
## Apr 2020       24205.54 22642.55 25768.52 21815.16 26595.92
## May 2020       23547.23 21984.24 25110.22 21156.85 25937.62
## Jun 2020       23963.93 22400.93 25526.92 21573.53 26354.32
```

Auto ETS

If you want R to automatically determine the best parameter setting for ETS, use the `ets` function. It determines the nature of error, trend and seasonality components automatically and returns the best model possible.

```
ets(cons_ts3)
```

```
## ETS(M,N,N)
##
## Call:
##  ets(y = cons_ts3)
##
##   Smoothing parameters:
##     alpha = 0.9999
##
##   Initial states:
##     l = 21011.9714
##
##   sigma:  0.0646
##
##      AIC     AICc      BIC
## 561.5532 562.4421 565.8552
```

Prophet

Prophet is Facebook's special forecasting library. In their own words "At its core, the Prophet procedure is an additive regression model with four main components:

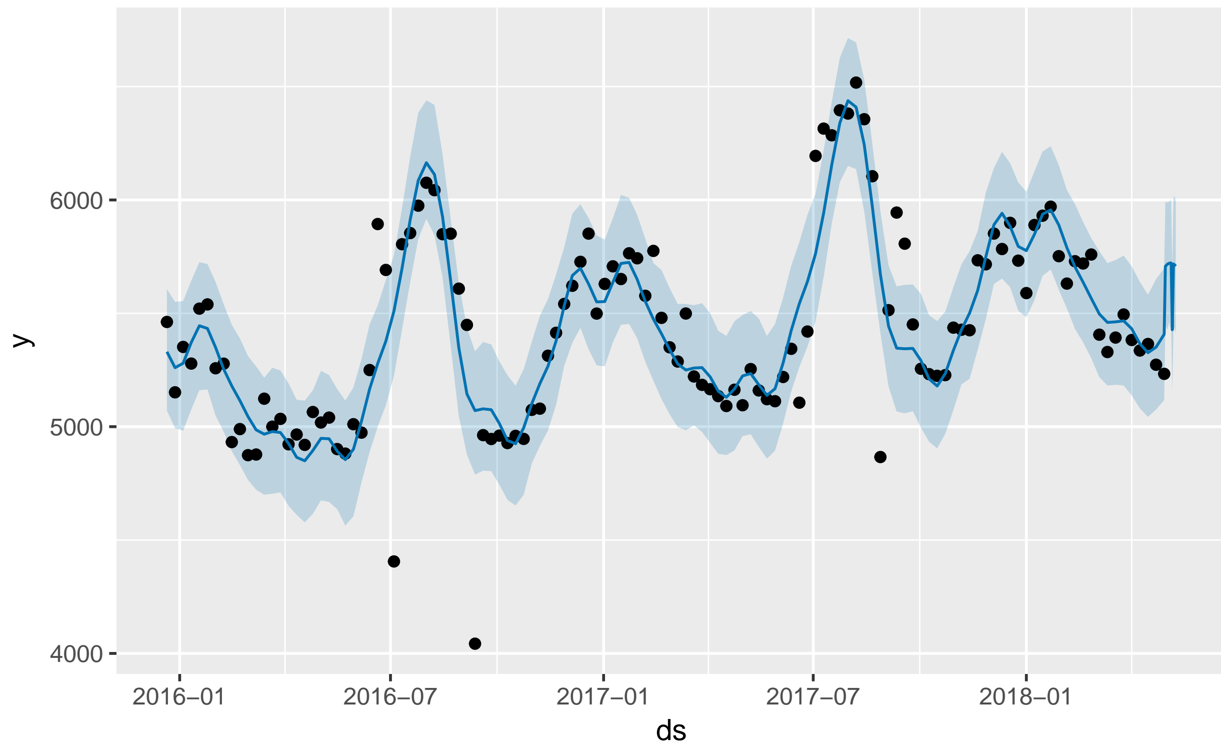
- A piecewise linear or logistic growth curve trend. Prophet automatically detects changes in trends by selecting changepoints from the data.
- A yearly seasonal component modeled using Fourier series.
- A weekly seasonal component using dummy variables.
- A user-provided list of important holidays."

You can see the details from here.

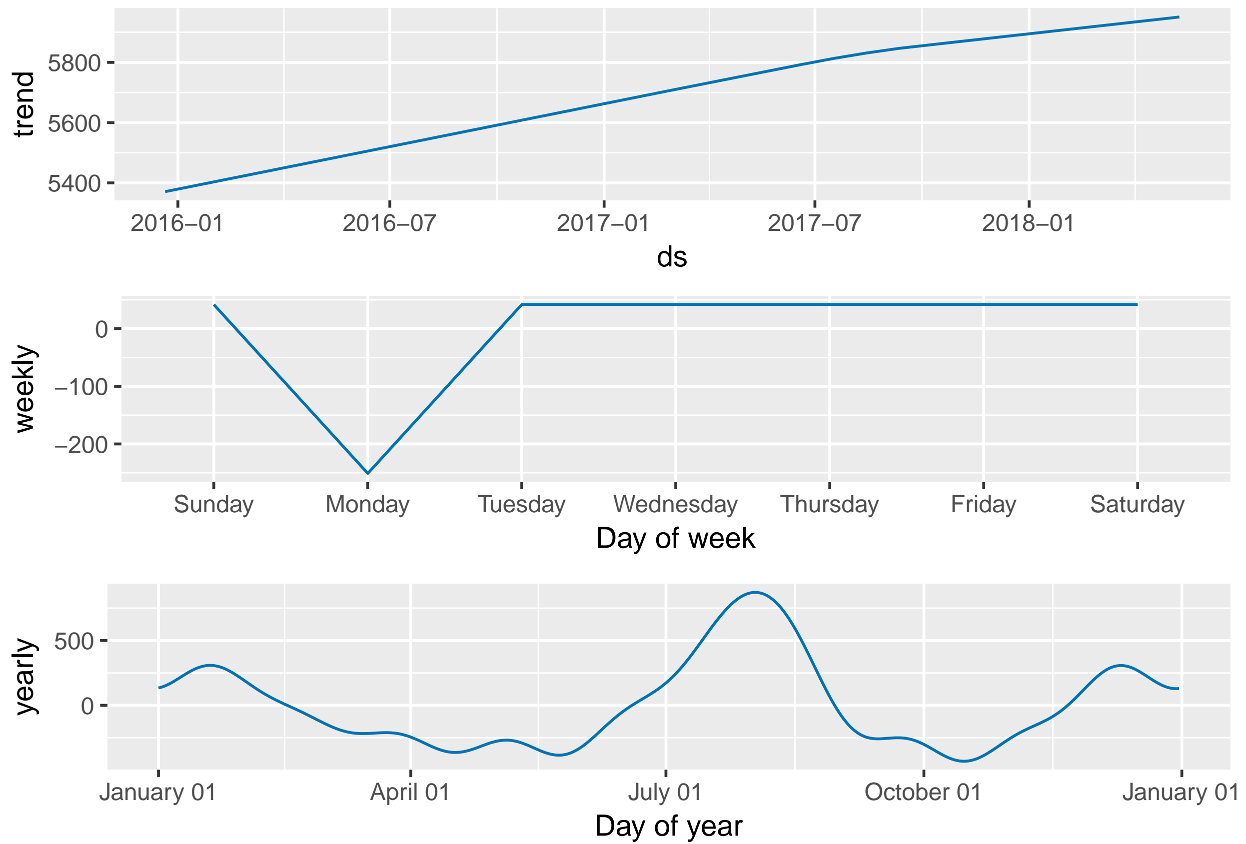
```
library(prophet)
consumption_prophet <- consumption_weekly %>% transmute(ds=seq(from=as.Date("2015-12-21"),to=as.Date("2
prophet_model <- prophet(consumption_prophet,weekly.seasonality=TRUE)
```

```
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.

## Initial log joint probability = -2.3182
## Optimization terminated normally:
##   Convergence detected: relative gradient magnitude is below tolerance
```

```r
future <- make_future_dataframe(prophet_model,periods=10)
prophet_forecast <- predict(prophet_model, future)

the_plot <- plot(prophet_model,prophet_forecast)
the_plot
```




```r
prophet_plot_components(prophet_model,prophet_forecast)
```



trend
5800
5600
5400
2016–01
2016–07
2017–01
2017–07
2018–01
ds
weekly
0
–100
–200
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week
yearly
500
0
January 01
April 01
July 01
October 01
January 01
Day of year