Advanced Time Series Methods

Seun Odeyemi

3/4/2018

```
# runif(1, 0, 10^8)
set.seed(77159275) #for reproducibility of results
```

```
rm(list =ls())
```

```
devtools::session_info()
```

```
##  setting  value
##  version  R version 3.4.3 (2017-11-30)
##  system   x86_64, linux-gnu
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  tz       Zulu
##  date     2018-03-05
##
##  package   * version date       source
##  backports   1.1.2   2017-12-13 CRAN (R 3.4.3)
##  base      * 3.4.3   2017-12-01 local
##  compiler    3.4.3   2017-12-01 local
##  datasets  * 3.4.3   2017-12-01 local
##  devtools    1.13.5  2018-02-18 CRAN (R 3.4.3)
##  digest      0.6.15  2018-01-28 CRAN (R 3.4.3)
##  evaluate    0.10.1  2017-06-24 CRAN (R 3.4.3)
##  graphics  * 3.4.3   2017-12-01 local
##  grDevices * 3.4.3   2017-12-01 local
##  htmltools   0.3.6   2017-04-28 CRAN (R 3.4.3)
##  knitr       1.20    2018-02-20 CRAN (R 3.4.3)
##  magrittr    1.5     2014-11-22 CRAN (R 3.4.3)
##  memoise     1.1.0   2017-04-21 CRAN (R 3.4.3)
##  methods   * 3.4.3   2017-12-01 local
##  Rcpp        0.12.15 2018-01-20 CRAN (R 3.4.3)
##  rmarkdown   1.9     2018-03-01 CRAN (R 3.4.3)
##  rprojroot   1.3-2   2018-01-03 CRAN (R 3.4.3)
##  stats     * 3.4.3   2017-12-01 local
##  stringi     1.1.6   2017-11-17 CRAN (R 3.4.3)
##  stringr     1.3.0   2018-02-19 CRAN (R 3.4.3)
##  tools       3.4.3   2017-12-01 local
##  utils     * 3.4.3   2017-12-01 local
##  withr       2.1.1   2017-12-19 CRAN (R 3.4.3)
##  yaml        2.1.17  2018-02-27 CRAN (R 3.4.3)
```

Loading some useful libraries

```
#library(XLConnect)
library(dplyr)
library(ggplot2)
```

```r
#library(forecast)
library(fpp2)
library(readxl)
library(data.table)
```

Set Working Directory

```r
setwd("/home/sdotserver1/projects/")
```

Dynamic Regression

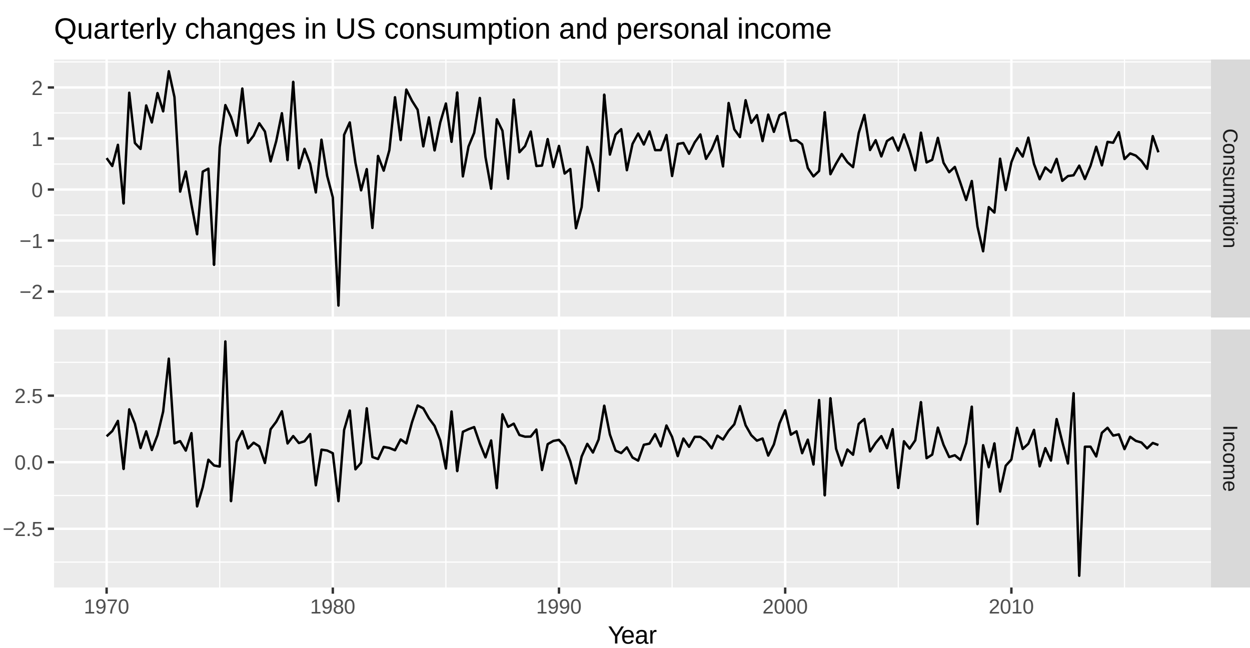
So far we have used time series models that use only the history of the time series, but do not use any other information. But often there is additional information that is available that will help us make better predictions. For example, if you are forecasting monthly sale then you could use the advertising expenditure for the month to improve your forecast. Or perhaps you can include information about competitor activity. Dynamic regression is one way of **combining this external information with the history of the time series in a single model**. The model looks like a standard linear regression model:

$$y_t = \beta_0 + \beta_1 x_{1,t} + ... + \beta_r x_{r,t} + e_t$$

- y_t modeled as function of r explanatory variables $x_{1,t}, ..., x_{r,t}$. This provide the external information you will want to use when forecasting
- In dynamic regression, we allow e_t to be an ARIMA process. This ARIMA process is where the historical information about the time series is incorporated
- In ordinary regression, we assume that e_t is white noise

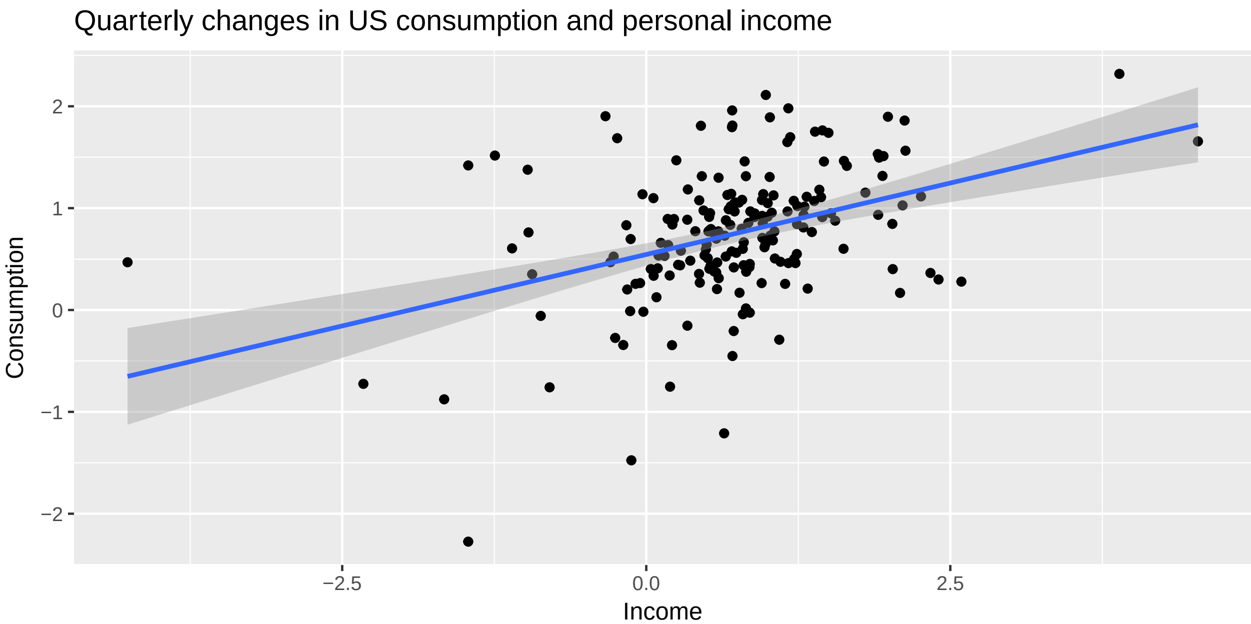
Let's look at an example using time series data containing the personal consumption and income in the US from 1960 to 2016

```r
autoplot(uschange[,1:2], facets = TRUE) +
  xlab("Year") + ylab("") +
  ggtitle("Quarterly changes in US consumption and personal income")
```



These two time series show quaterly changes in US consumption and quaterly changes in US personal income. You might want to forecast **consumption** and use **income** as a predictor variable. If there is a drop in income, you might expect consumption to drop as well and vice versa. The scatter plot below shows the relationship between the two variables.

```
ggplot(aes(x = Income, y = Consumption),
       data = as.data.frame(uschange)) +
  geom_point() +
  geom_smooth(method = "lm") +
  ggtitle("Quarterly changes in US consumption and personal income")
```



Clearly, there is a positive relationship between them as we expected. It is not a particularly strong relationship, but it does provide some useful information that will help give us better forecast of consumption.

Fitting a dynmaic regression model is not much more difficult than fitting an ARIMA model, you still use the `auto.arima()` function. It just needs one more argument, `xreg`, which contains a matrix of predicted variables you want to include in the model. When you include an `xreg` argument, `auto.arima` will fit a dynamic regression model rather than a regular ARIMA model. In this case, it has fitted a linear regression to the income variable, and then choosen an ARIMA (1, 0, 2) model for the errors. As usual, the ARIMA coefficients are not particularly interpretable, but the regression coefficient is interpretable.

```
fit <- auto.arima(uschange[,"Consumption"],
                  xreg = uschange[,"Income"])
summary(fit)
```

```
## Series: uschange[, "Consumption"]
## Regression with ARIMA(1,0,2) errors
##
## Coefficients:
##          ar1      ma1     ma2  intercept    xreg
##       0.6922  -0.5758  0.1984     0.5990  0.2028
## s.e.  0.1159   0.1301  0.0756     0.0884  0.0461
##
## sigma^2 estimated as 0.3219:  log likelihood=-156.95
## AIC=325.91   AICc=326.37   BIC=345.29
##
```

```
## Training set error measures:
##                       ME      RMSE       MAE     MPE     MAPE      MASE
## Training set 0.001714366 0.5597088 0.4209056 27.4477 161.8417 0.6594731
##                     ACF1
## Training set 0.006299231
```

Here we see that the consumption change increased by `0.20`% when income changes by `1`%. In dynamic regression models, the regression part takes care of the predicted variable, while the ARIMA part takes care of the short-term dynamics. As with all forecasting models, you should check that the residuals look like white noise.

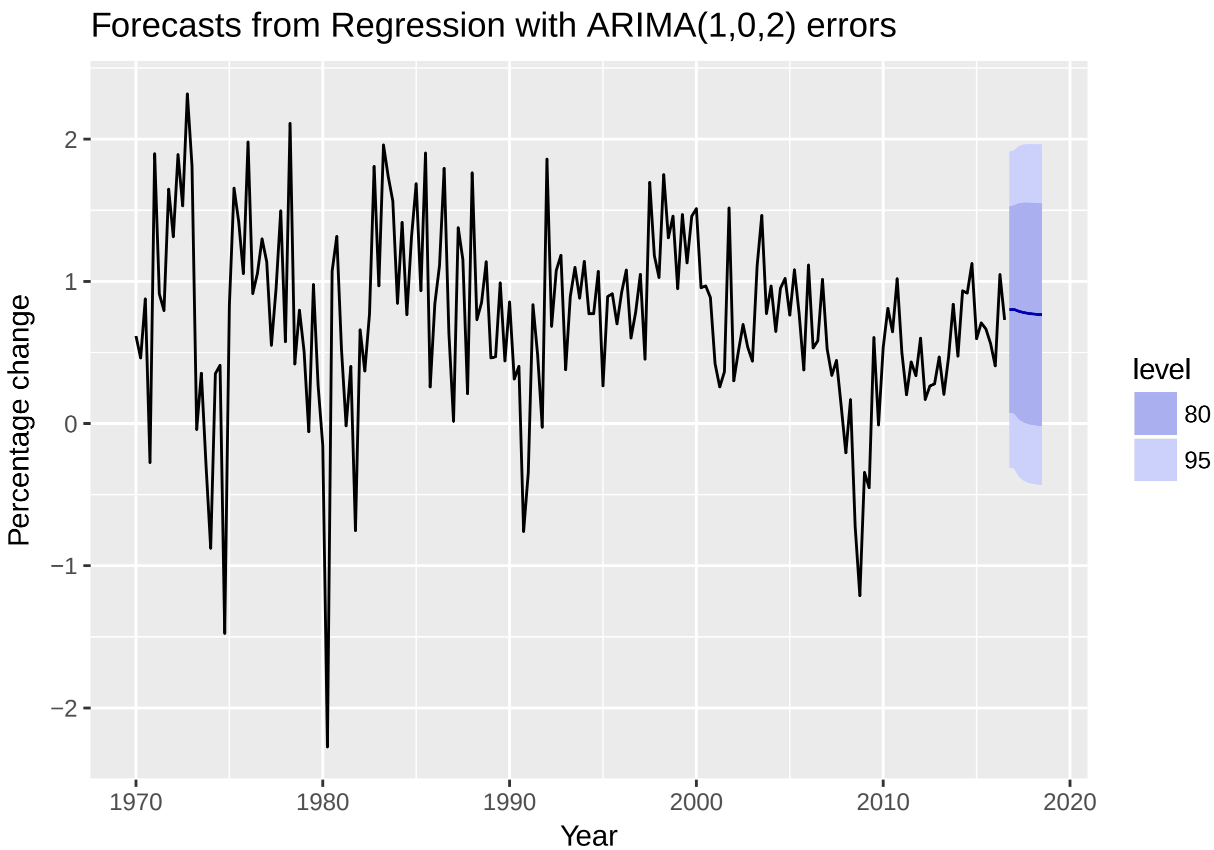
```
checkresiduals(fit)
```


```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(1,0,2) errors
## Q* = 5.8916, df = 3, p-value = 0.117
##
## Model df: 5.   Total lags used: 8
```

The Lyung-Box test here is above 0.05, which means these residuals look like white noise. To forecast with dynamic models, you need to provide future values of the predictors. Either you can forecast this in a separate model or you can do a scenario forecasting where you look at the effect of different values of the predictors on the forecast. The future values of the predictors need to be passed to the `forecast` function using the `xreg` argument just as the past values were included in the `auto.arima` function.

```
fcast <- forecast(fit, xreg = rep(0.8, 8))
autoplot(fcast) +
```

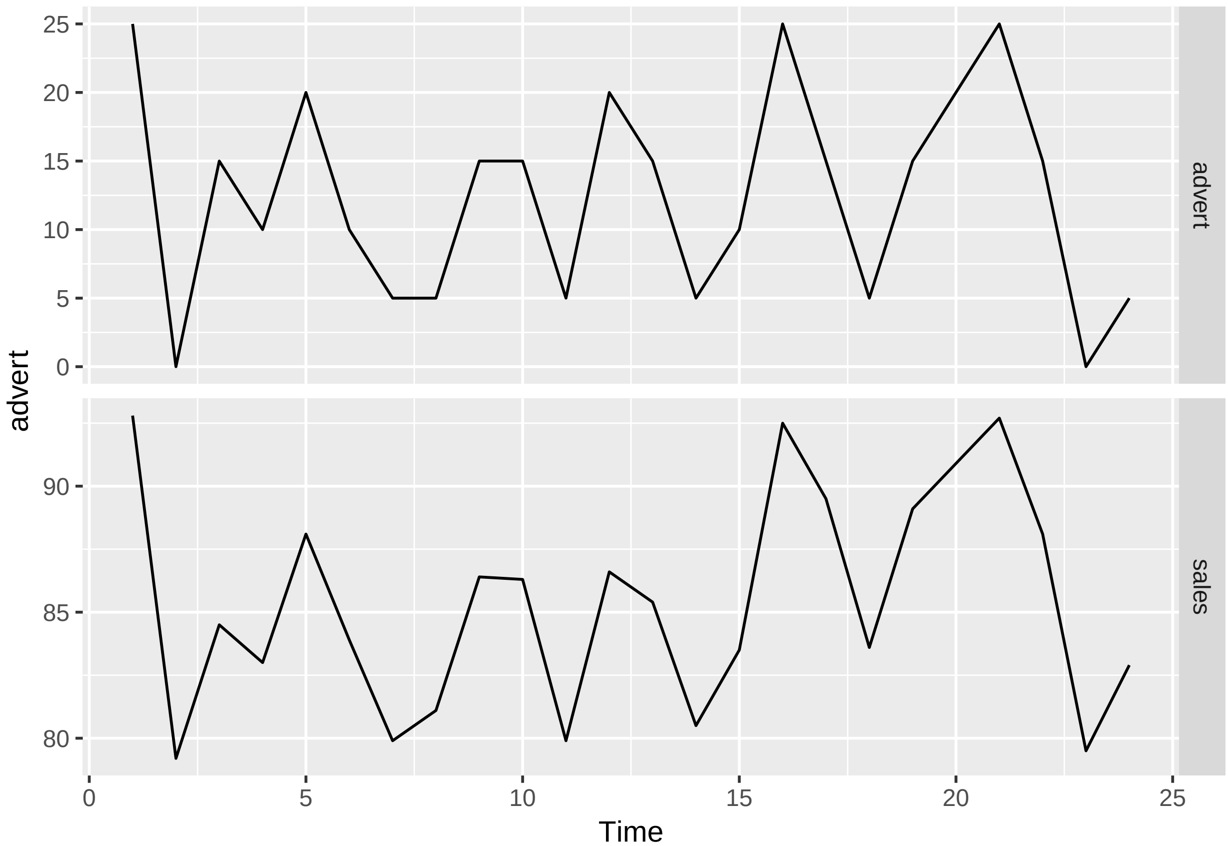
```r
  xlab("Year") + ylab("Percentage change")
```



Here, we have assumed the future income change of 0.8% per quarter for the next 8 quarters.

Forecasting Sales Allowing for Advertising Expenditure

```r
# Time plot of both variables
autoplot(advert, facets = TRUE)
```

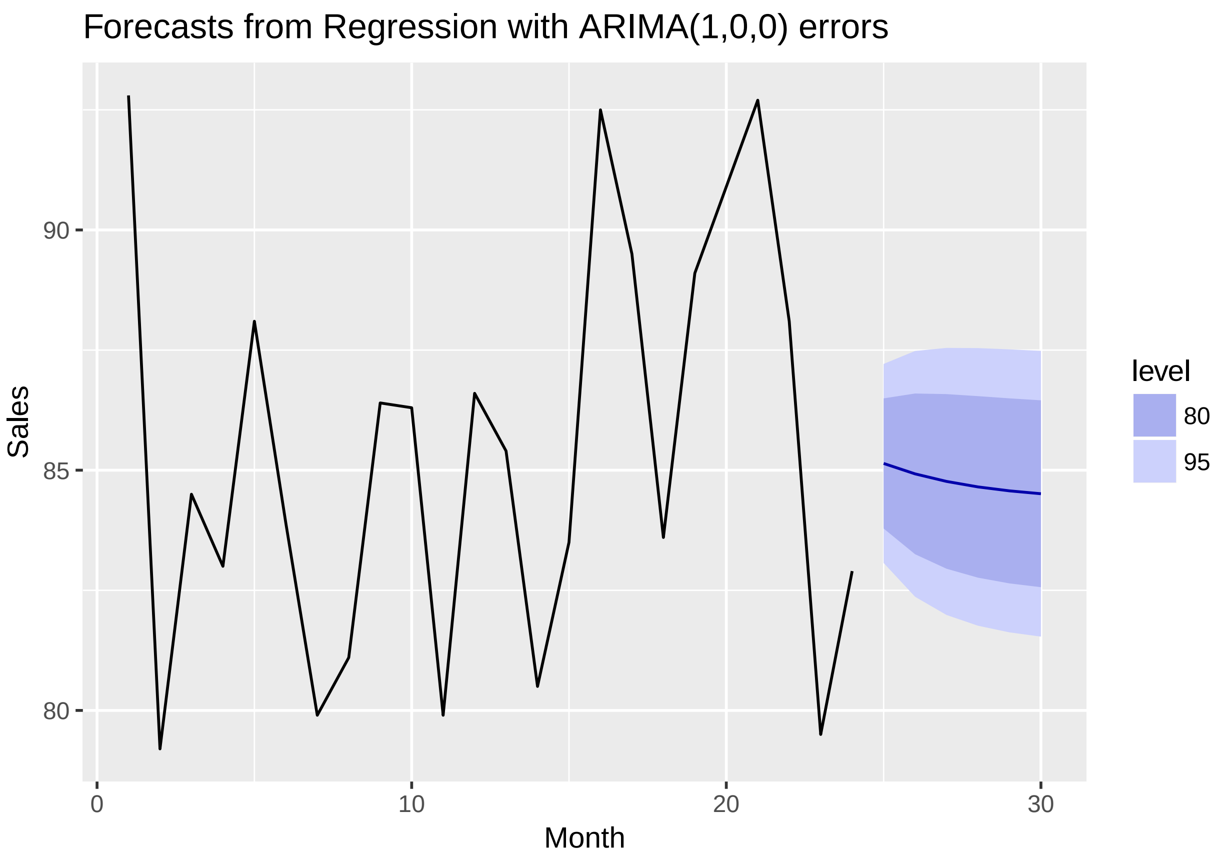



```r
# Fit ARIMA model
fit <- auto.arima(advert[, "sales"], xreg = advert[, "advert"], start.q = 0, max.q = 1, stationary = TRU
# fit2 <- auto.arima(advert[, "sales"], xreg = advert[, "advert"], stationary = TRUE)
summary(fit)
```

```
## Series: advert[, "sales"]
## Regression with ARIMA(1,0,0) errors
##
## Coefficients:
##          ar1  intercept   xreg
##       0.7247    79.2725  0.508
## s.e.  0.1339     0.7349  0.022
##
## sigma^2 estimated as 1.116:  log likelihood=-34.15
## AIC=76.29   AICc=78.4   BIC=81
##
## Training set error measures:
##                       ME     RMSE       MAE         MPE      MAPE
## Training set -0.03570439 0.988353 0.7612276 -0.06588987 0.8951198
##                   MASE       ACF1
## Training set 0.1650164 0.02381244
```

```r
# summary(fit2)

# Check model. Increase in sales for each unit increase in advertising
salesincrease <- coefficients(fit)[3]

# Forecast fit as fc
```

```
fc <- forecast(fit, xreg = rep(10, 6))

# Plot fc with x and y labels
autoplot(fc) + xlab("Month") + ylab("Sales")
```



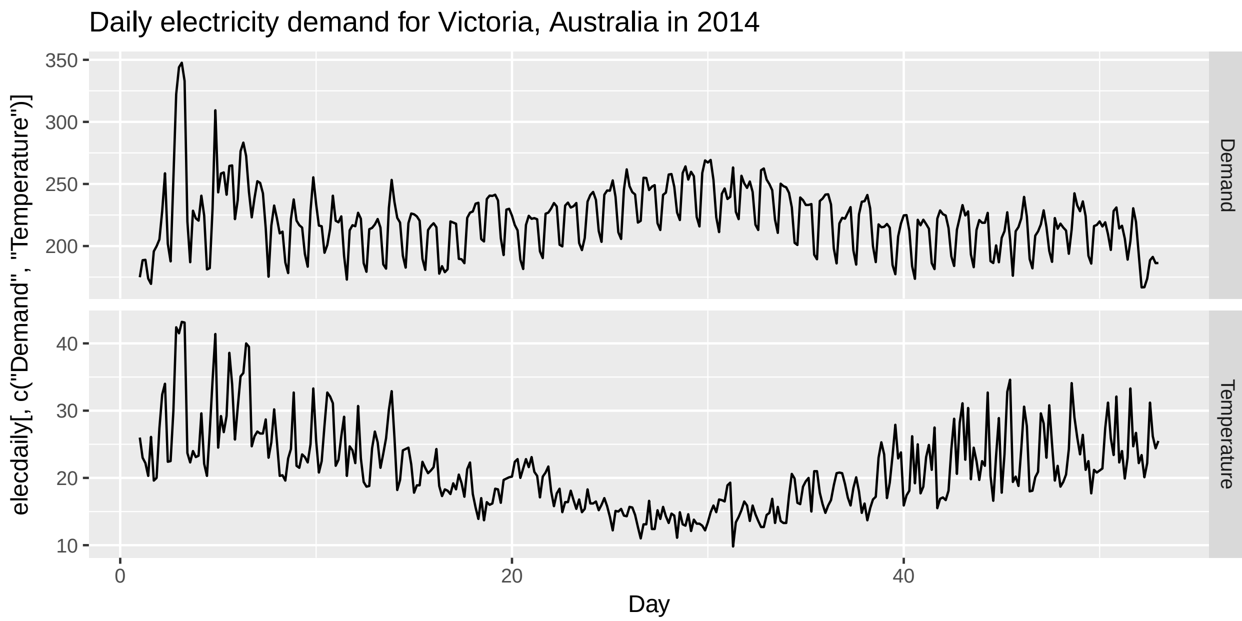
According to the `auto.arima` function, for every **$1** of advertising investment, there is **50 cent** increase in sales.

Forecasting electricity demand

You can also model daily electricity demand as a function of temperature. As you may have seen on your electric bill, more electricity is used on hot days due to air conditioning and on cold days due to heating.

In this exercise, you will fit a quadratic regression model with an ARMA error. One year of daily data are stored as elec including total daily demand, an indicator variable for workdays (a workday is represented with 1, and a non-workday is represented with 0), and daily maximum temperatures. Because there is weekly seasonality, the frequency has been set to 7.

```
# Time plots of demand and temperatures
autoplot(elecdaily[, c("Demand", "Temperature")], facets = TRUE)+
  xlab("Day") +
  ggtitle("Daily electricity demand for Victoria, Australia in 2014")
```




```
# Matrix of regressors
xreg <- cbind(MaxTemp = elecdaily[, "Temperature"],
              MaxTempSq = elecdaily[, "Temperature"] ^ 2,
              Workday = elecdaily[, "WorkDay"])
# Fit model
fit <- auto.arima(elecdaily[, "Demand"] , xreg = xreg)
summary(fit)
```

```
## Series: elecdaily[, "Demand"]
## Regression with ARIMA(2,1,2)(2,0,0)[7] errors
##
## Coefficients:
##           ar1     ar2      ma1      ma2    sar1    sar2    drift
##       -0.0622  0.6731  -0.0234  -0.9301  0.2012  0.4021  -0.0191
## s.e.   0.0714  0.0667   0.0413   0.0390  0.0533  0.0567   0.1091
##       xreg.MaxTemp  xreg.MaxTempSq  xreg.Workday
##            -7.4996          0.1789       30.5695
## s.e.        0.4409          0.0084        1.2891
##
## sigma^2 estimated as 43.72:  log likelihood=-1200.7
## AIC=2423.4   AICc=2424.15   BIC=2466.27
##
## Training set error measures:
##                      ME     RMSE      MAE         MPE     MAPE      MASE
## Training set 0.05300191 6.511663 4.716611 -0.07616974 2.137864 0.3238475
##                     ACF1
## Training set -0.03189628
```

```
# Forecast fit one day ahead
forecast(fit, xreg = cbind(20,20^2,1))
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 53.14286       185.4008 176.9271 193.8745 172.4414 198.3602
```

Great job! Now you've seen how multiple independent variables can be included using matrices.

Dynamic Harmonic Regression

One particularly useful kind of regression is called **dynamic harmonic regression**. . Fourier was a French mathematician who showed that a series of sine and cosine terms of the right frequencies can approximate any periodic function. You can use them for seasonal patterns when forecasting. `Fourier` terms come in pairs consisting of a `sine` and `cosine`. The frequency of these terms are called the **harmonic frequencies**, and they increase with K. These `Fourier` terms are predictors in our dynamic regression model; the more terms you include in the model, the more complicated our seasonal pattern will be. We choose uppercase K for how many terms gets included. The α_k and γ_k are coefficients in our regression model. Because the seasonality is being modeled by the `Fourier` terms, you normally use a non-seasonal ARIMA model for the error. One important difference in handling seasonality this way rather than using a seasonal ARIMA model is that `Fourier` terms as shown in the seasonal pattern does not change over time. Whereas the seasonal ARIMA model allows the seasonal pattern to evolve over time.

$$s_k(t) = sin\left(\frac{2\pi kt}{m}\right)$$

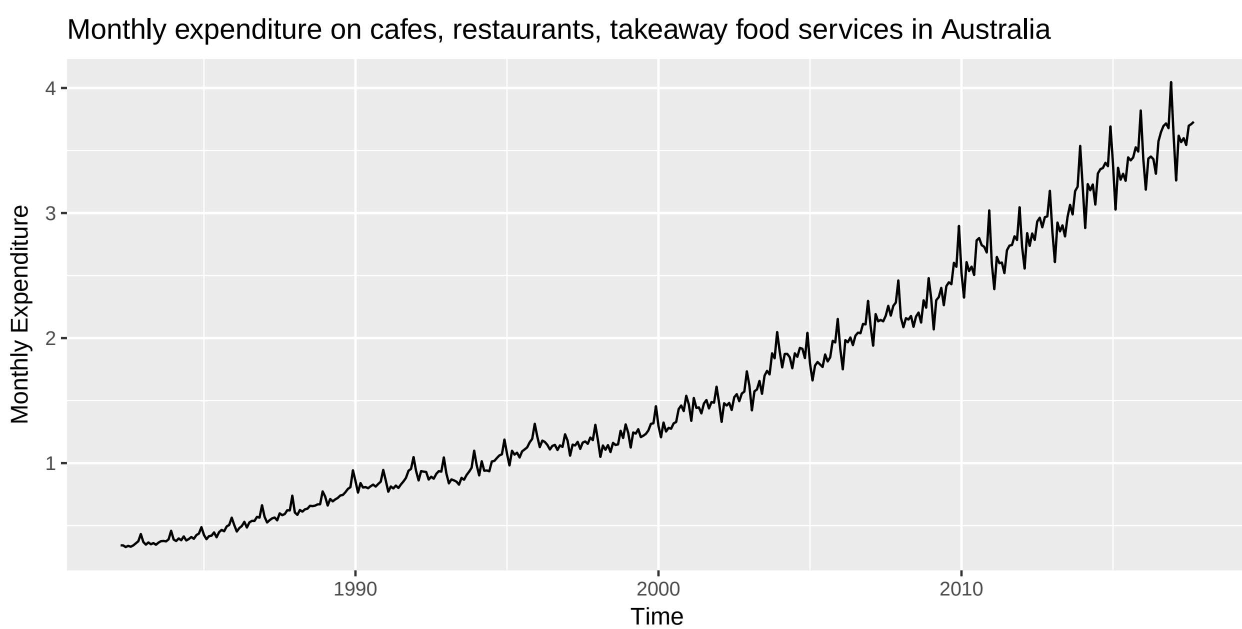
$$c_k(t) = cos\left(\frac{2\pi kt}{m}\right)$$

$$y_t = \beta_0 + \sum_{k=1}^{k}[\alpha_k s_k(t) + \gamma_k c_k(t)] + e_t$$

- m = seasonal period
- Every periodic function can be approximated by sums of sin and cos terms for large enough K
- Regression coefficients: α_k and γ_k
- e_t can be modeled as a non-seasonal ARIMA process
- Assumes seasonal pattern is unchanging

Let's see an example using the time series object `auscafe`, which contains data on monthly expenditure on eating out in Australia from April 1982 to September 2017.

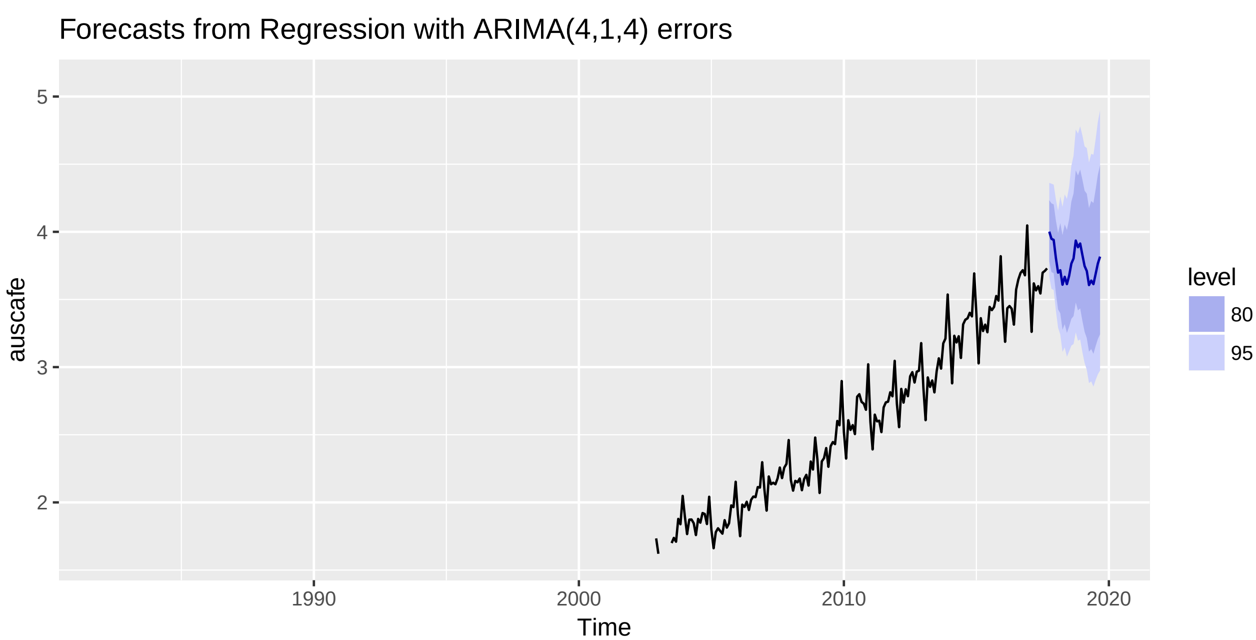
```
autoplot(auscafe) +
  ylab("Monthly Expenditure") +
  ggtitle("Monthly expenditure on cafes, restaurants, takeaway food services in Australia ")
```



```r
fit <- auto.arima(auscafe, xreg = fourier(auscafe, K = 1), seasonal = FALSE, lambda = 0)
summary(fit)
```

```
## Series: auscafe
## Regression with ARIMA(4,1,4) errors
## Box Cox transformation: lambda= 0
##
## Coefficients:
##           ar1      ar2      ar3      ar4     ma1      ma2     ma3     ma4
##       -0.9330  -0.3118  -0.3984  -0.6154  0.4752  -0.5480  0.2390  0.6840
## s.e.   0.0576   0.0921   0.0865   0.0543  0.0535   0.0704  0.0589  0.0439
##         S1-12    C1-12
##       -0.0348  -0.0197
## s.e.   0.0024   0.0024
##
## sigma^2 estimated as 0.00194:  log likelihood=727.15
## AIC=-1432.3   AICc=-1431.66   BIC=-1387.73
##
## Training set error measures:
##                      ME       RMSE        MAE       MPE     MAPE      MASE
## Training set 0.01446027 0.07711295 0.0536669 0.8835616 3.351414 0.5136254
##                     ACF1
## Training set -0.04278696
```

```r
fit %>%
  forecast(xreg = fourier(auscafe, K = 1, h = 24)) %>%
  autoplot() + ylim(1.6, 5.1)
```

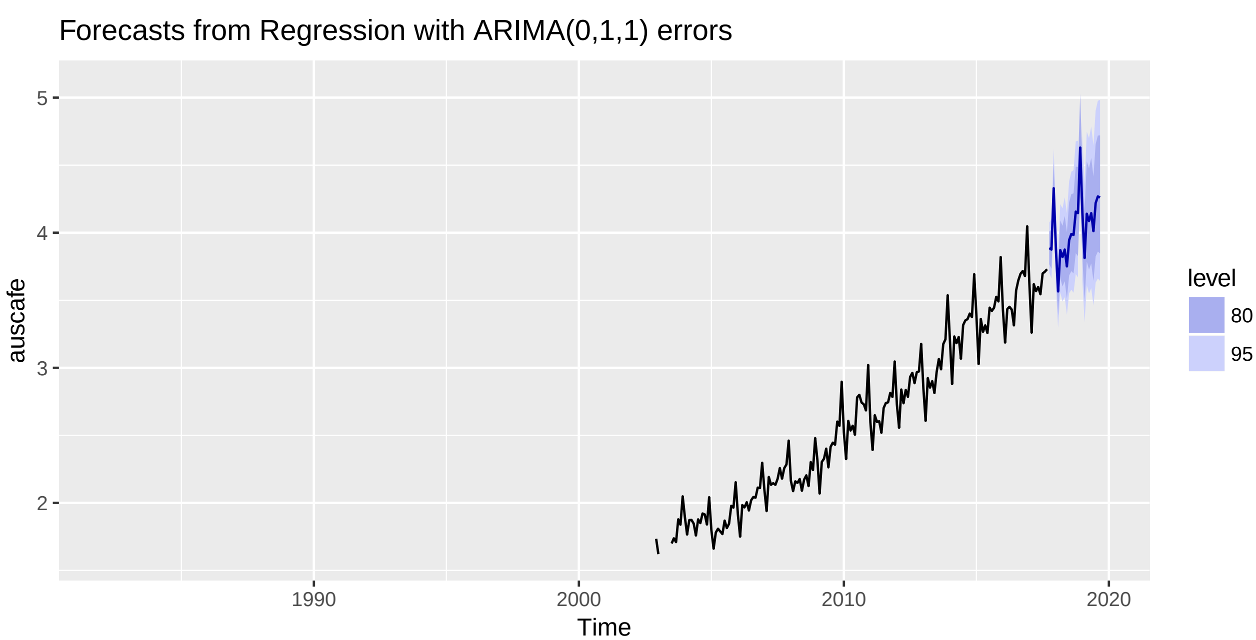


Notice that we set `seasonal = FALSE` meaning the ARIMA error in the model should be non-seasonal. I have also used a BoxCox transformation by setting $\lambda = 0$ because the variance increases with the level of the series. You can use the `fourier()` function to generate all the `Fourier` terms to be included in our model. You just have to select the value of K which indicates how complicated the seasonal pattern will be. When forecasting you use the `fourier()` function again to generate future values of the predictors. It must have the same value of K that was used in fitting the model. By adding the `h` argument, the `fourier()` function knows you want the future value and not past values. `h` is the forecast horizon. Using $K = 1$ does not capture the seasonal pattern very well. Let's increase the value of K to see how the forecast change.

```
fit <- auto.arima(auscafe, xreg = fourier(auscafe, K = 5), seasonal = FALSE, lambda = 0)
summary(fit)
```

```
## Series: auscafe
## Regression with ARIMA(0,1,1) errors
## Box Cox transformation: lambda= 0
##
## Coefficients:
##           ma1   drift    S1-12    C1-12   S2-12    C2-12   S3-12   C3-12
##       -0.3467  0.0056  -0.0345  -0.0193  0.0130  -0.0212  0.0343  0.0035
## s.e.   0.0445  0.0008   0.0023   0.0023  0.0014   0.0014  0.0012  0.0012
##         S4-12   C4-12    S5-12   C5-12
##        0.0122  0.0157  -0.0198  0.0182
## s.e.   0.0011  0.0011   0.0011  0.0011
##
## sigma^2 estimated as 0.0005901:  log likelihood=982.96
## AIC=-1939.92   AICc=-1939.03   BIC=-1887.24
##
## Training set error measures:
##                        ME       RMSE        MAE         MPE     MAPE
## Training set -0.001599654 0.03789639 0.02793053 -0.03097366 1.904107
##                   MASE       ACF1
## Training set 0.2673124 0.05371972
```

```
fit %>%
  forecast(xreg = fourier(auscafe, K = 5, h = 24)) %>%
  autoplot() + ylim(1.6, 5.1)
```



As K increases, the seasonal pattern start to look more like the past data. You also notice that the ARIMA error model gets simpler as there is less signal in the residuals when K is larger. The best way to select K is to try a few different values and then select the value of K that gives the lowest AIC_c value. In this case it is $K = 5$. The model can include other predictor variables as well as the `Fourier` terms. They just need to be added to the `xreg` matrix. The advantage to using `Fourier` terms compared with other methods of modeling seasonality is that they can handle seasonality when the seasonal period, m is very large. For example, with weekly data where m is approximately 52. Daily data where m could be 365, if there is annual seasonality. And sub-daily data where it could be even higher.

$$y_t = \beta_0 + \beta_1 x_{t,1} + ... + \beta_{t,r} x_{t,r} + \sum_{k=1}^{k} [\alpha_k s_k(t) + \gamma_k c_k(t)] + e_t$$

The whole process is mostly automated. The only thing you must do yourself is select K

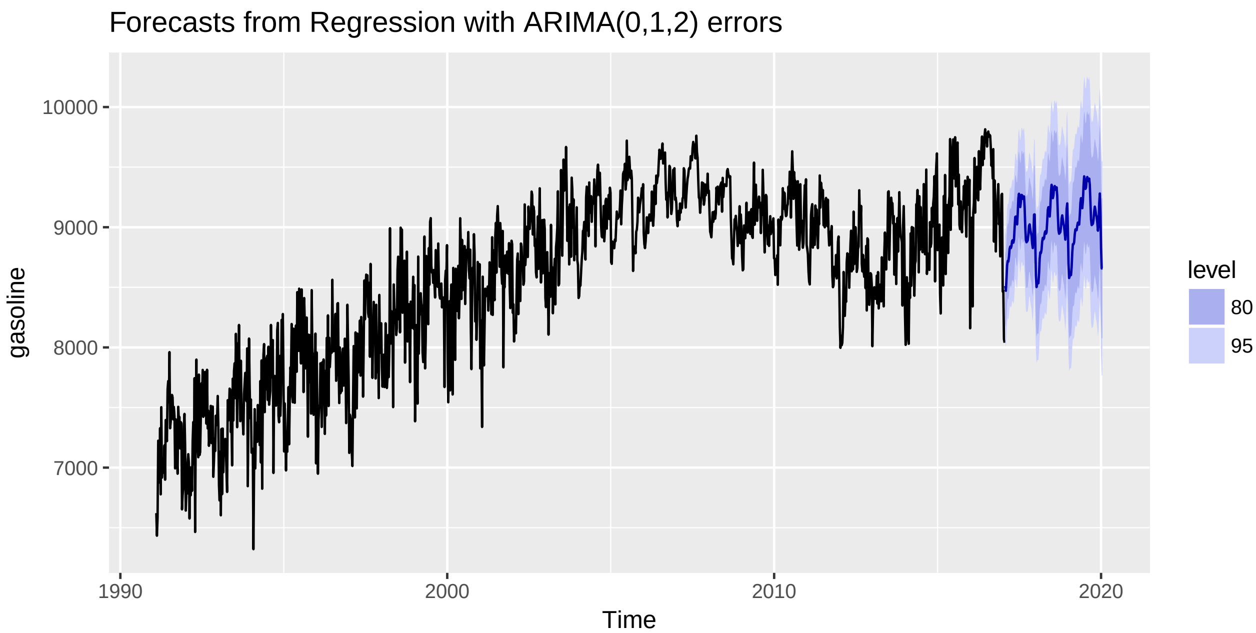
- Other predictors variables can be added as well: $x_{t,1}, ..., x_{t,r}$
- Choose K to minimize AIC_c
- K cannot be more than m/2
- This is particularly useful for weekly data, daily data, and sub-daily data.

Forecasting weekly data

With weekly data, it is difficult to handle seasonality using ETS or ARIMA models as the seasonal length is too large (approximately 52). Instead, you can use harmonic regression which uses sines and cosines to model the seasonality.

The `fourier()` function makes it easy to generate the required harmonics. The higher the order (K), the more "wiggly" the seasonal pattern is allowed to be. With $K = 1$, it is a simple sine curve. You can select the value of K by minimizing the AICc value. As you saw in the video, fourier() takes in a required time series, required number of Fourier terms to generate, and optional number of rows it needs to forecast.

```r
# Set up harmonic regressors of order 13
harmonics <- fourier(gasoline, K = 13)

# Fit regression model with ARIMA errors
fit <- auto.arima(gasoline, xreg = harmonics, seasonal = FALSE)

# Forecasts next 3 years
newharmonics <- fourier(gasoline, K = 13, h = 156)
fc <- forecast(fit, xreg = newharmonics)

# Plot forecasts fc
autoplot(fc)
```



Great. The point predictions look to be a bit low.

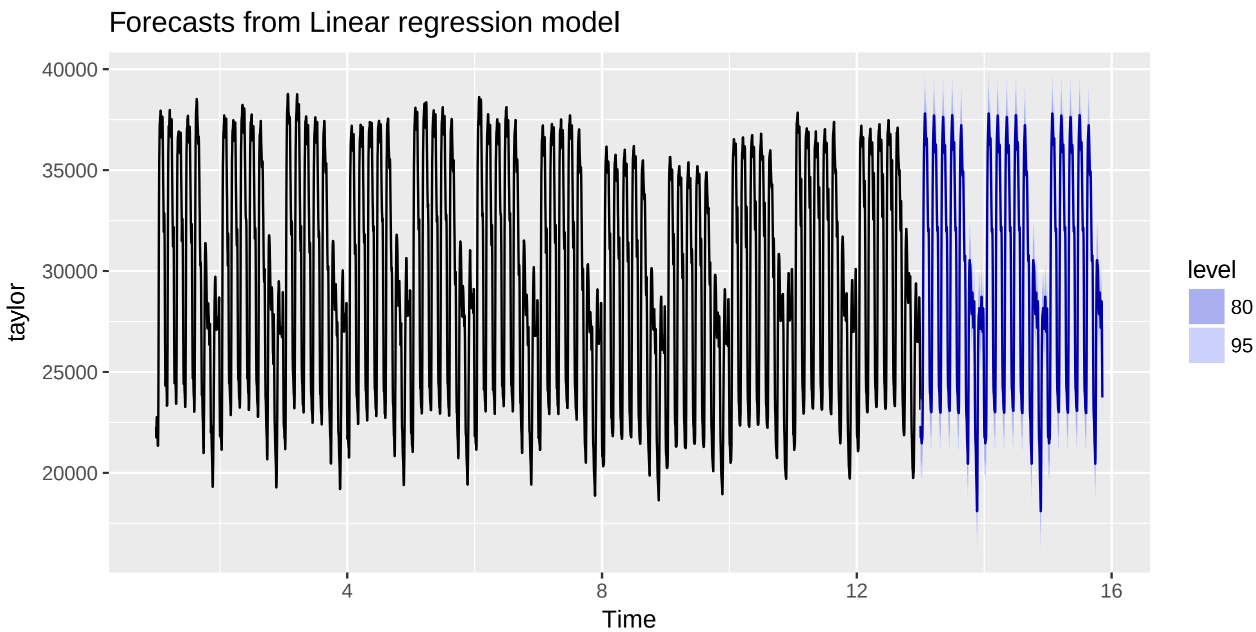
Harmonic regression for multiple seasonality

Harmonic regressions are also useful when time series have multiple seasonal patterns. For example, `taylor` contains half-hourly electricity demand in England and Wales over a few months in the year 2000. The seasonal periods are 48 (daily seasonality) and 7 x 48 = 336 (weekly seasonality). There is not enough data to consider annual seasonality.

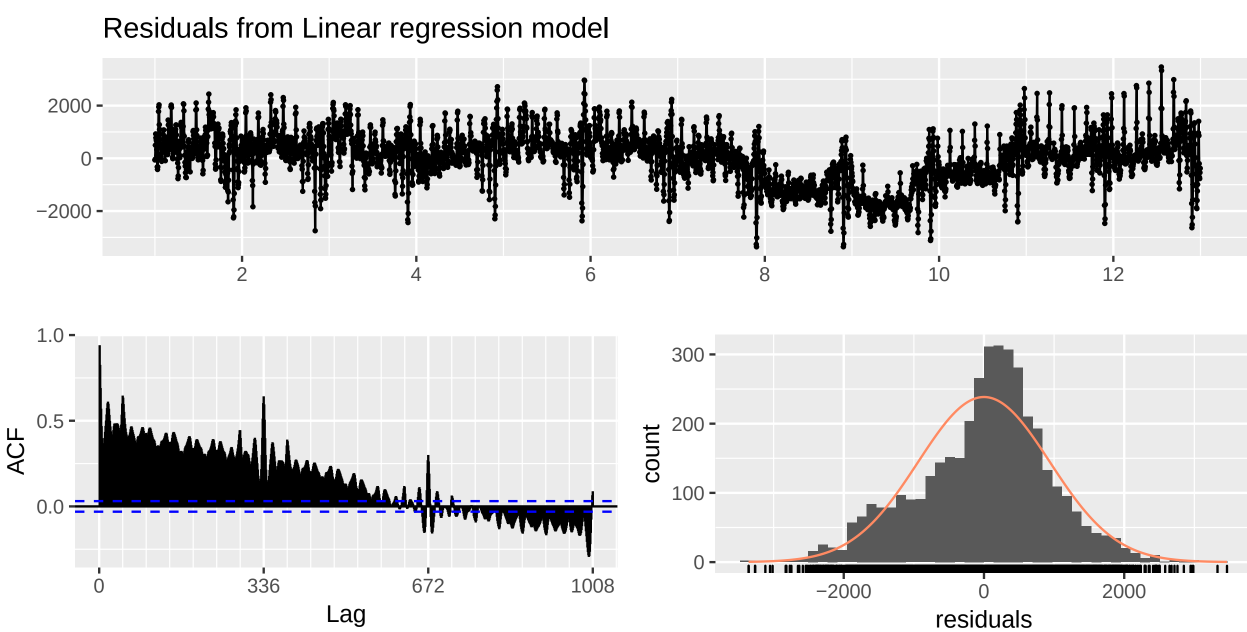
`auto.arima()` would take a long time to fit a long time series such as this one, so instead you will fit a standard regression model with Fourier terms using the `tslm()` function. This is very similar to `lm()` but is designed to handle time series. With multiple seasonality, you need to specify the order K for each of the seasonal periods.

`tslm()` is a newly introduced function, so you should be able to follow the pre-written code for the most part. The taylor data are loaded into your workspace.

```
# Fit a harmonic regression using order 10 for each type of seasonality
fit <- tslm(taylor ~ fourier(taylor, K = c(10, 10)))

# Forecast 20 working days ahead
fc <- forecast(fit, newdata = data.frame(fourier(taylor, K = c(10, 10), h = 960)))

# Plot the forecasts
autoplot(fc)
```




```r
# Check the residuals of fit
checkresiduals(fit)
```




```
##
##  Breusch-Godfrey test for serial correlation of order up to 672
##
## data:  Residuals from Linear regression model
## LM test = 3938.9, df = 672, p-value < 2.2e-16
```

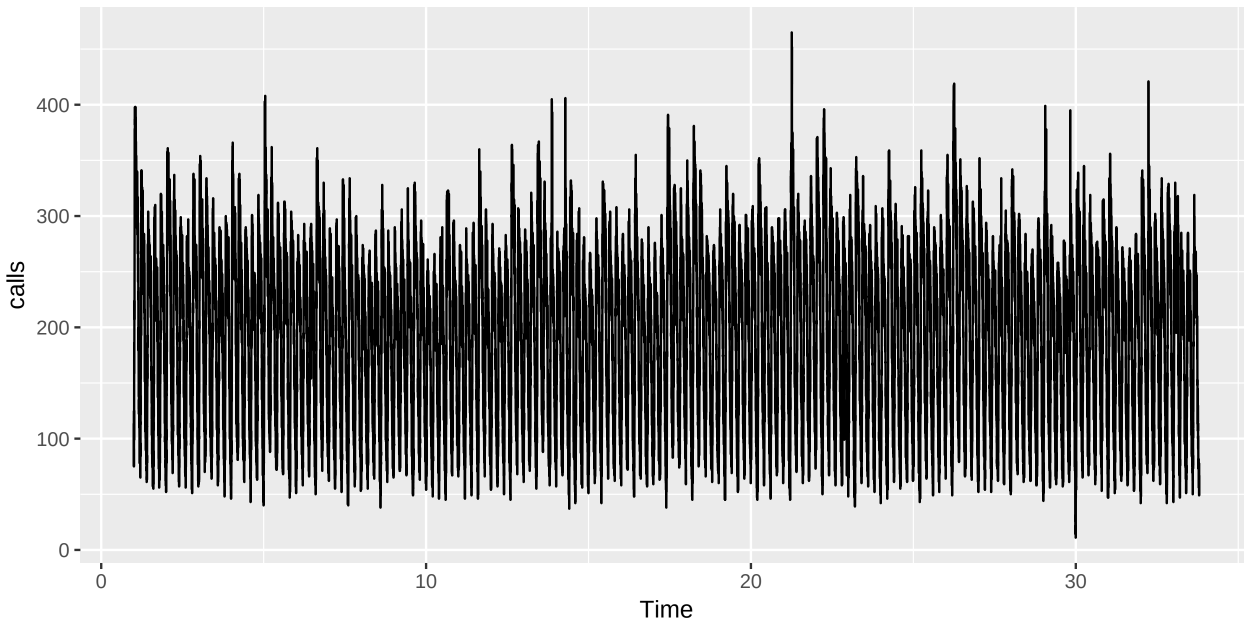
As you can see, `auto.arima()` would have done a better job.

Forecasting call bookings

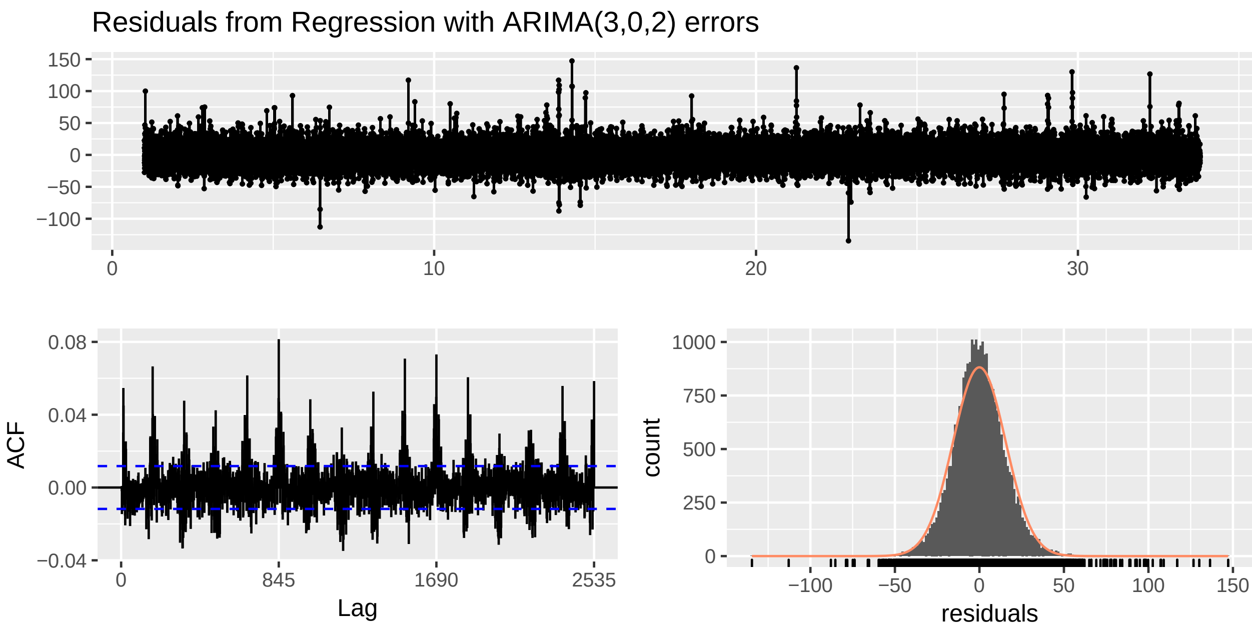
Another time series with multiple seasonal periods is `calls`, which contains 20 consecutive days of 5-minute call volume data for a large North American bank. There are 169 5-minute periods in a working day, and so the weekly seasonal frequency is 5 x 169 = 845. The weekly seasonality is relatively weak, so here you will just model daily seasonality. `calls` is pre-loaded into your workspace.

The residuals in this case still fail the white noise tests, but their autocorrelations are tiny, even though they are significant. This is because the series is so long. It is often unrealistic to have residuals that pass the tests for such long series. The effect of the remaining correlations on the forecasts will be negligible.

```
# Plot the calls data
autoplot(calls)
```

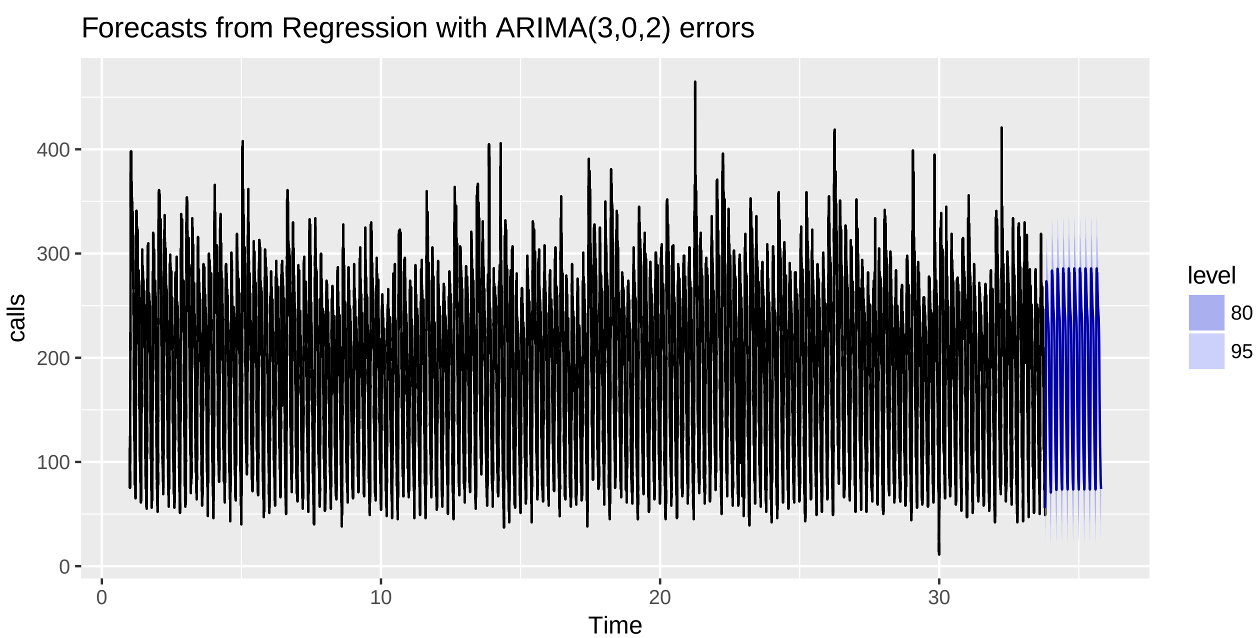



```
# Set up the xreg matrix
xreg <- fourier(calls, K = c(10,0))

# Fit a dynamic regression model
fit <- auto.arima(calls, xreg = xreg, seasonal = FALSE, stationary = TRUE)

# Check the residuals
checkresiduals(fit)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(3,0,2) errors
## Q* = 6861.7, df = 1664, p-value < 2.2e-16
##
## Model df: 26.   Total lags used: 1690
```

```r
# Plot forecasts for 10 working days ahead
fc <- forecast(fit, xreg =  fourier(calls, c(10, 0), h = 1690))
autoplot(fc)
```



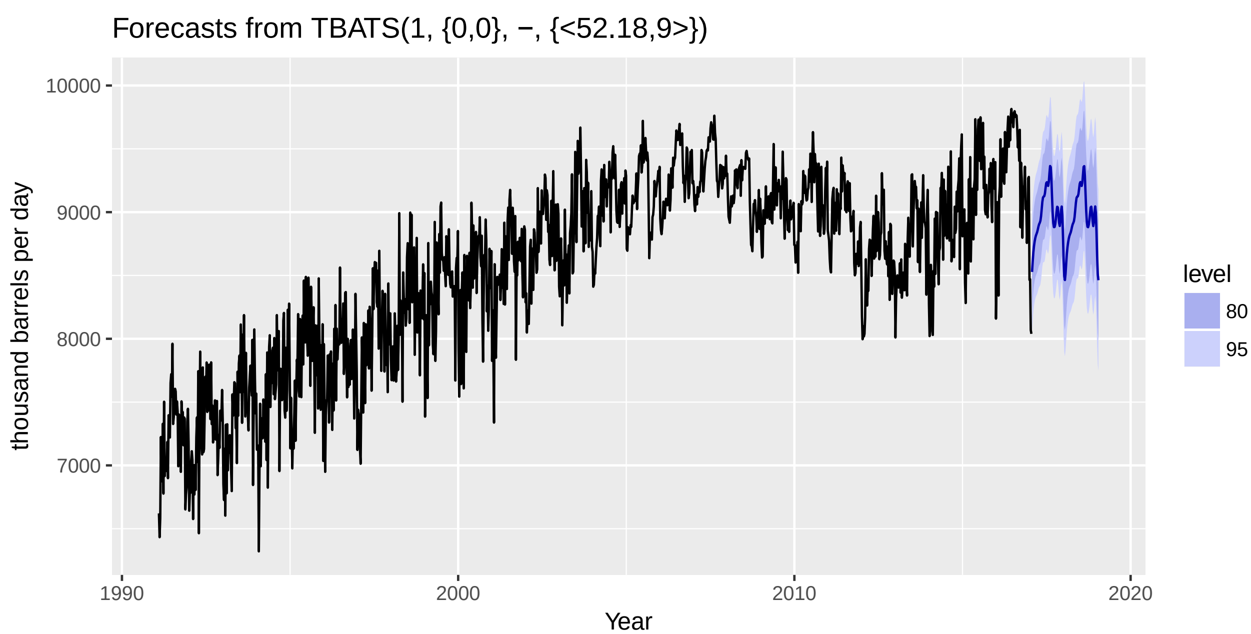
Great! Now you've gotten a lot of experience using complex forecasting techniques.

TBATS models

A TBATS model combines many of the components of models we've already used into one single automated framework. It includes trigonometric terms for seasonality. These are similar to the `Fourier` terms we used in harmonic regression, except here the seasonality can change over time. It includes a `BoxCox` transformation for heterogeneity. It has `ARMA` errors for short-term dynamics as we saw in the dynamic regression. It has level and trend terms similar to an `ets()` model. Everything is automated. This makes them very convenient but also somewhat dangerous, as sometimes the automatic choices are not so good. Let's look at some examples.

- Handles non-integer seasonality, multiple seasonal periods
- Entirely automated
- Prediction intervals o#en too wide
- Very slow on long series

```
gasoline %>% tbats() %>% forecast() %>%
autoplot() +
  xlab("Year") + ylab("thousand barrels per day")
```

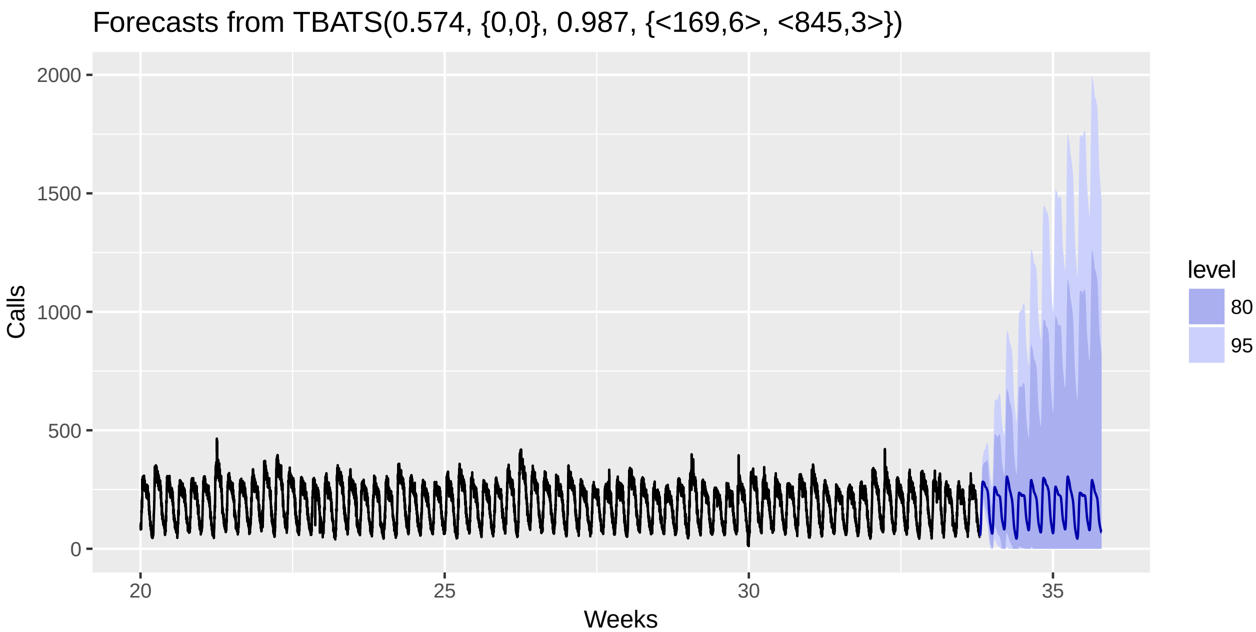


See how easy the `tbats()` function is easy to use. Just pass the `ts` object to the `tbats()` function, and then the results to the `forecast()` function. The title of the graph shows what choices have been made. The first **1** is the `BoxCox` parameter – meaning no transformation was required. The next part is the ARMA error – meaning p = 0 and q = 0, so a simple white noise error was used. The third part is the damping parameter for trend – a dash (-) means no damping. So this pretty simple so far: no transformation, no ARMA error, no damping. The last part tells us about the `Fourier` terms: the seasonal period is 52.18 (the number of weeks in a year). There were 14 fourier-like terms selected.

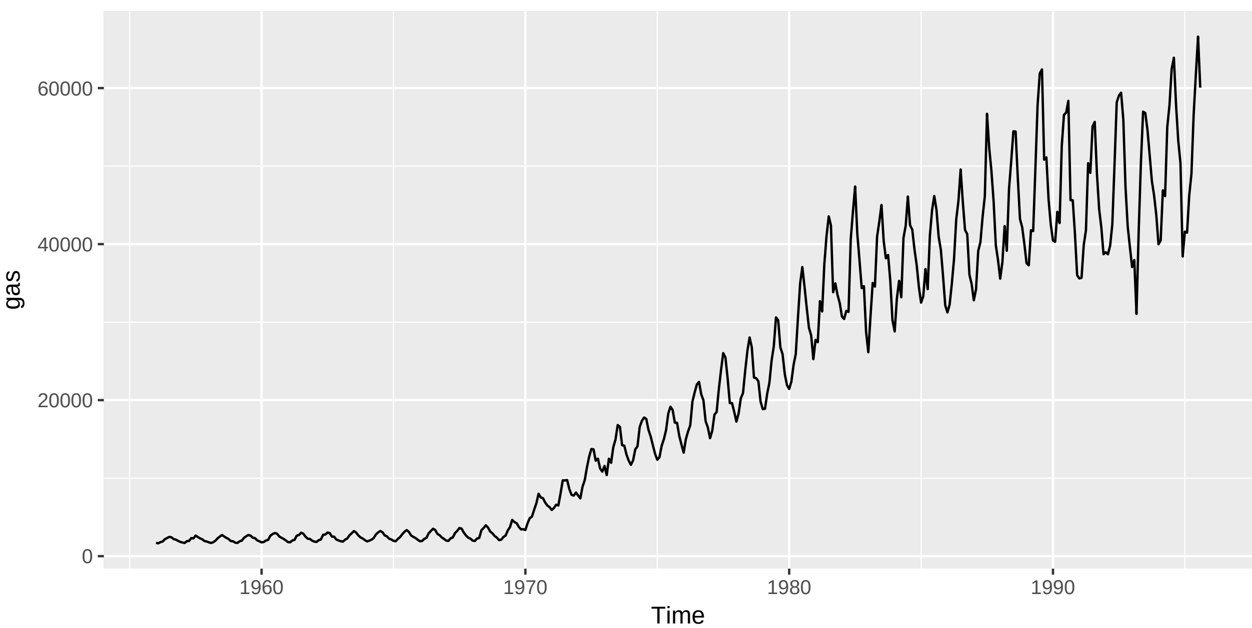
The forecast looks ok, although perhaps they are a little low.

This next example contains forecast of call volumes every 5 minute to an American bank.

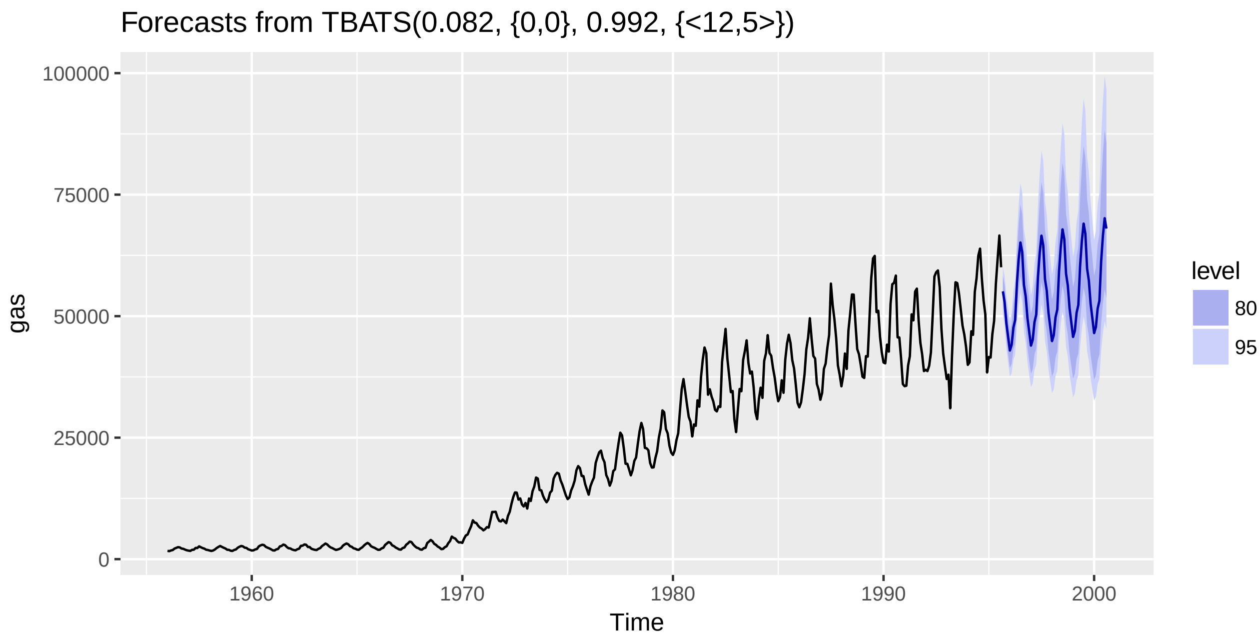
```
calls %>% window(start = 20) %>%
  tbats() %>% forecast() %>%
  autoplot() + xlab("Weeks") + ylab("Calls")
```




```
# Plot the gas data
autoplot(gas)
```




```
# Fit a TBATS model to the gas data
fit <- tbats(gas)

# Forecast the series for the next 5 years
fc <- forecast(fit, h = 60)

# Plot the forecasts
autoplot(fc)
```




```r
# Record the Box-Cox parameter and the order of the Fourier terms
lambda <- 0.082
K <- 5
```

Amazing! Just remember that completely automated solutions don't work every time.