Time Series Regression

Intro to Time Series Analysis

Representing and plotting time series data

A time series can be thought of as a list of numbers, along with information regarding the points in time those numbers refer to. Such information is often stored in R as a *ts* object.

Let's see how we can create a ts object using the data stored in a data frame:

```
# read the data from the "data/yearly_eatout_expenditures.csv" file
yearly_exp <- read.csv("data/yearly_eatout_expenditures.csv")
str(yearly_exp)
```

```
## 'data.frame':    17 obs. of  2 variables:
##  $ year   : int  2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 ...
##  $ tot_exp: num  16.1 17.6 18.1 20.3 22.5 ...
```

The data is about total yearly expenditure in billion AUD on eating out (in cafes, restaurants and takeaway) in Australia in the period 2000-2016.

Transform this data frame into a ts object and print the results:

```
yearly_exp_ts <- ts(yearly_exp$tot_exp, start=2000)

yearly_exp_ts
```

```
## Time Series:
## Start = 2000
## End = 2016
## Frequency = 1
##  [1] 16.1328 17.6170 18.0945 20.3288 22.4583 22.2311 24.1840 26.2601
##  [9] 26.3498 29.0435 31.8246 32.1919 34.3818 35.8945 39.2950 40.7665
## [17] 42.6097
```

To examine this time series visually we'll use plotting functions of the **fpp2** R package.

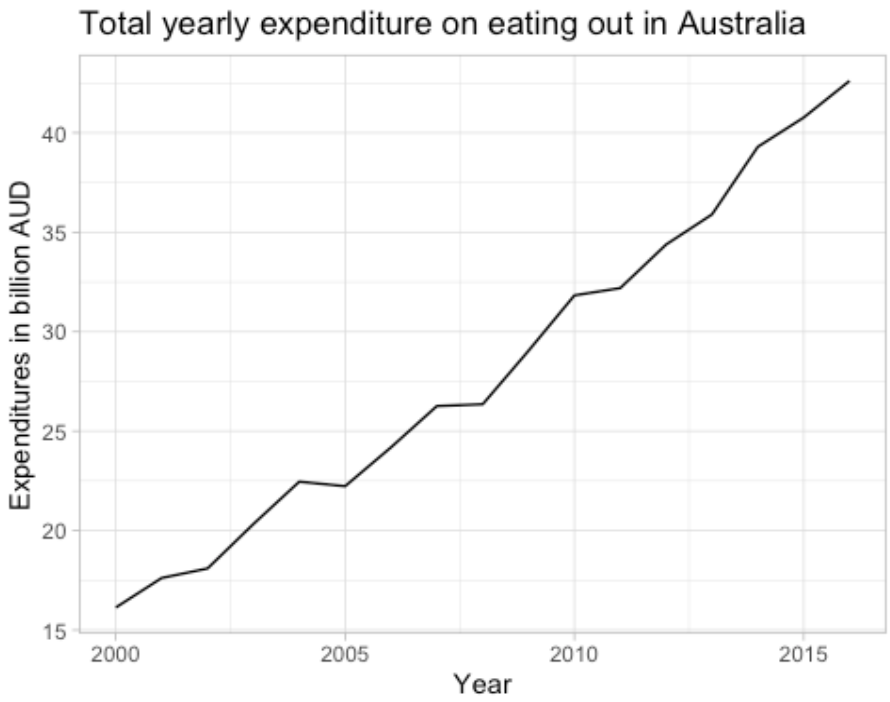
Note 1: the *fpp2* package is built on top of *ggplot2*, so it will automatically load *ggplot2*.

Note 2: majority of the functions for time series analysis and regression that we'll use in this lab come from the fpp2 package.

```
# install.packages('fpp2')
library(fpp2)
```

Plot the time series:

```
autoplot(yearly_exp_ts) +
  xlab("Year") +
  ylab("Expenditures in billion AUD") +
  ggtitle("Total yearly expenditure on eating out in Australia") +
  theme_light()
```



For observations that are more frequent than once per year, we have to add the frequency argument when creating a ts object. The frequency is the number of observations within a seasonal "pattern" (eg, daily, weekly, monthly, quarterly).

Let's now examine if Australians tended to spend more or less on eating out depending on the season. To that end, we will use the data about total monthly expenditures for the same time period.

```
monthly_exp <- read.csv("data/monthly_eatout_expenditures.csv")
head(monthly_exp)
```

```
##   year month expenditure
## 1 2000   Jan      1.2974
## 2 2000   Feb      1.2067
## 3 2000   Mar      1.3246
## 4 2000   Apr      1.2525
## 5 2000   May      1.2821
## 6 2000   Jun      1.2747
```

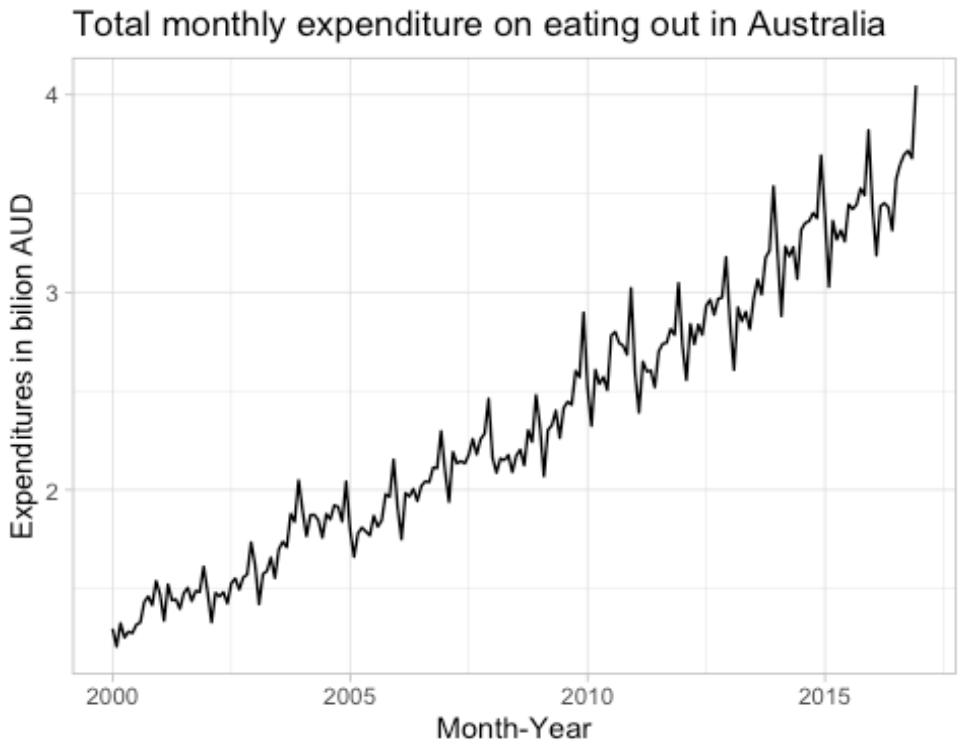
Transform the loaded data into a ts object and print time series data for the first two years:

```
monthly_exp_ts <- ts(monthly_exp$expenditure, start = c(2000, 1), frequency =
12)
head(monthly_exp_ts, 2*12)
```

```
##         Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
## 2000 1.2974 1.2067 1.3246 1.2525 1.2821 1.2747 1.3179 1.3288 1.4324 1.4608
## 2001 1.4708 1.3380 1.5212 1.4413 1.4461 1.3976 1.4774 1.5046 1.4372 1.4885
##         Nov    Dec
## 2000 1.4163 1.5386
## 2001 1.4829 1.6114
```

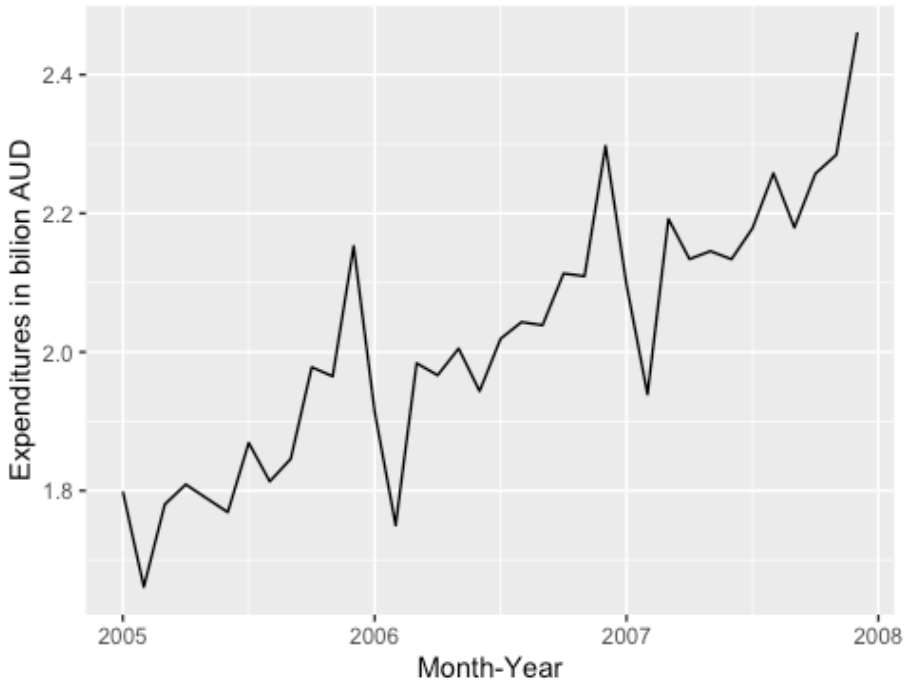
Plot the monthly time series:

```
autoplot(monthly_exp_ts) +
  xlab("Month-Year") +
  ylab("Expenditures in bilion AUD") +
  ggtitle("Total monthly expenditure on eating out in Australia") +
  theme_light()
```



To take a closer look at the monthly pattern, we can take a slice of the time series. This can be done with the window() function by specifying start and/or end arguments:

```
autoplot(window(monthly_exp_ts, start=c(2005,1), end=c(2007,12))) +
  xlab("Month-Year") + ylab("Expenditures in bilion AUD")
```



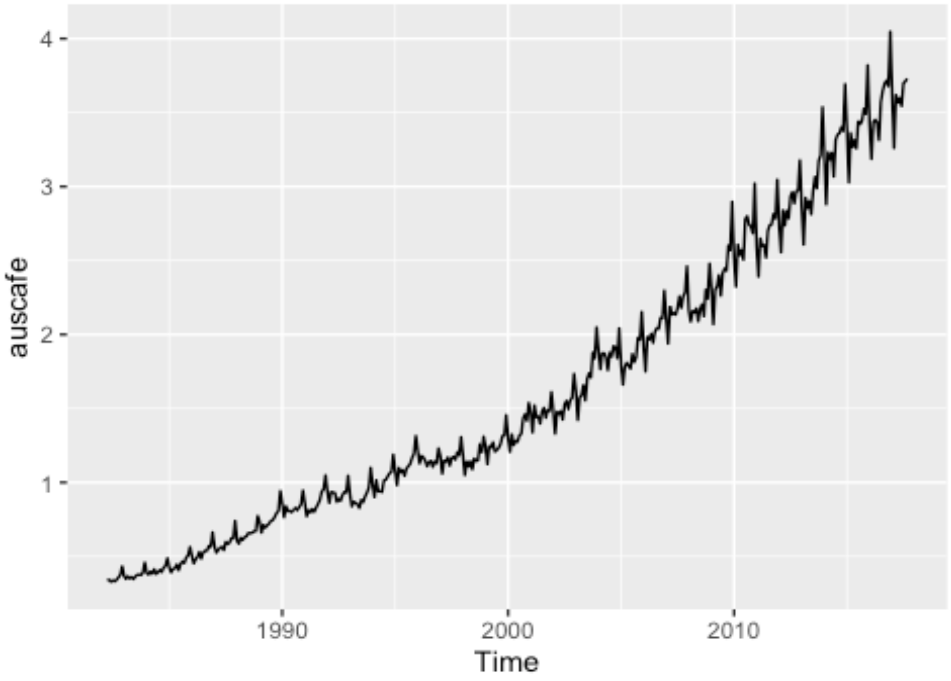
Time series components

Trend

A trend exists when there is a long-term increase or decrease in the data. It can also be a "changing direction" trend, as the one that starts as an increasing trend and then changes to a decreasing trend. Previously examined expenditure data showed an increasing trend.

As another example of a time series with a rising trend we can take the original data set (auscafe) from which the two above examined were extracted:

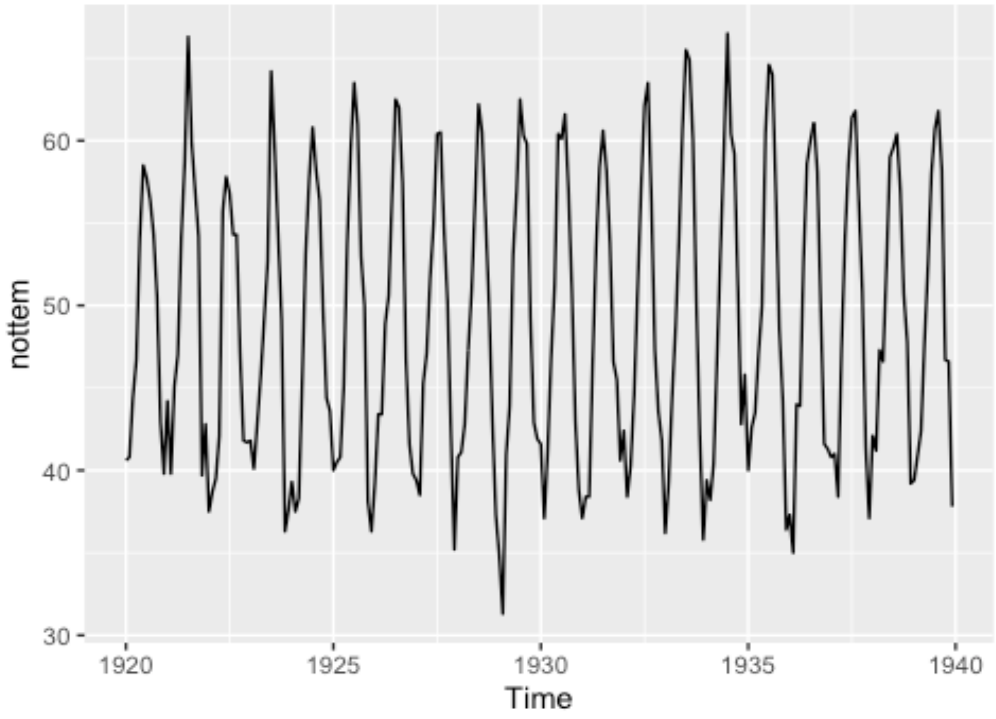
```
autoplot(auscafe)
```



Seasonality

A seasonal pattern occurs when a time series is affected by seasonal factors such as the time of the year or the day of the week. As an example, we can take a look at the data set with average monthly temperatures (in degrees Fahrenheit) at Nottingham, 1920–1939:

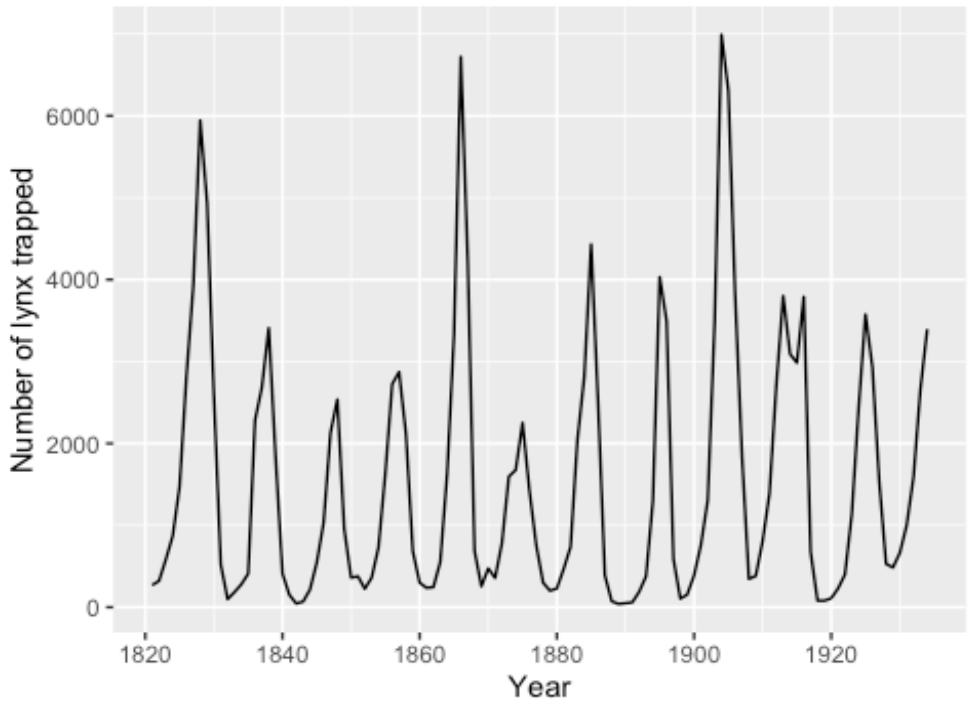
```
autoplot(nottem)
```



Cycle

A cycle occurs when the data exhibit rises and falls, often due to economic conditions. Unlike seasonal patterns, these are not of a fixed frequency. The following example shows an aperiodic cycles of approximately 10 years.

```
# Annual numbers of lynx trappings for 1821–1934 in Canada
autoplot(lynx) + xlab("Year") + ylab("Number of lynx trapped")
```

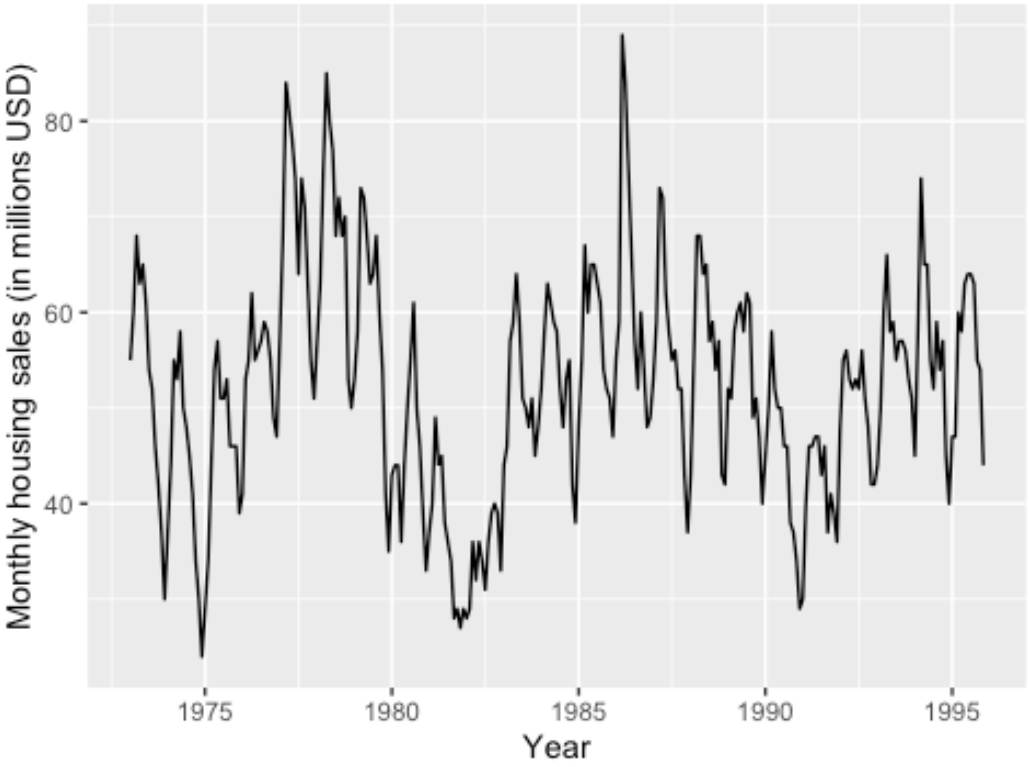


How to differentiate cycles from seasonal patterns:

- fluctuations that are not of a fixed frequency are cyclic; if the frequency is unchanging and associated with some aspect of the calendar, then it is a seasonal pattern.
- average length of a cycle tends to be longer than that of a seasonal pattern
- magnitude of a cycle tends to be more variable than in seasonal patterns

The two patterns - seasonal and cyclic - can also be combined. For example, the monthly sales of new one-family houses in the USA (1973-1995) show a strong seasonality within each year, as well as a cyclic pattern with period about 6 – 10 years:

```
autoplot(hsales) + xlab("Year") + ylab("Monthly housing sales (in millions
USD)")
```



When choosing a forecasting method, one needs to first identify the presence and combination of time series components (trend, season, cycle) in the data, and choose a method that is able to capture those patterns properly. See Section 2.3 of the Forecasting: Principle and Practices book for further examples and explanations.

Lagged values and autocorrelations

Lagged values of a time series are values from a previous point in time. For lag=1, and observation in time t, that would be observation at time t-1. In general, for lag=k, it would be observation at time t-k. For example, for seasonal monthly data, lag=1 refers to the value observed in the previous month.

We can compute lagged values using the lag() function from the stats package:

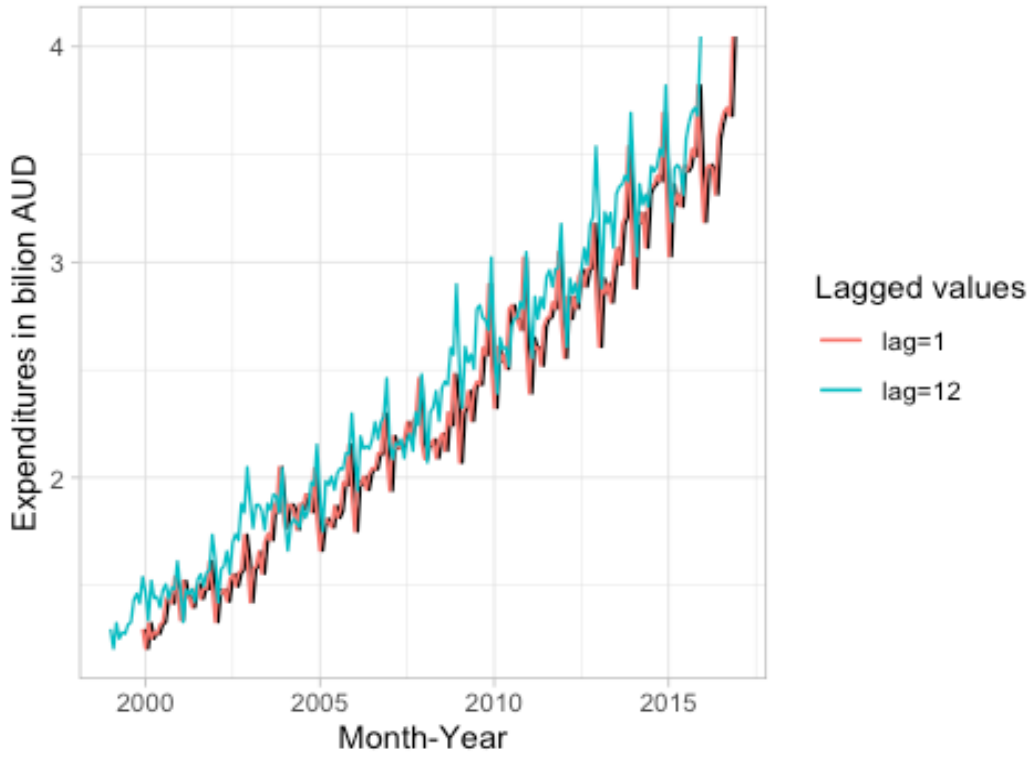
```
monthly_exp_lag1 <- stats::lag(monthly_exp_ts, k=1)
head(monthly_exp_lag1)
```

```
##         Jan    Feb    Mar    Apr    May Jun Jul Aug Sep Oct Nov    Dec
## 1999                                                             1.2974
## 2000 1.2067 1.3246 1.2525 1.2821 1.2747
```

This can be better observed when plotted:

```
autoplot(monthly_exp_ts) +
  autolayer(stats::lag(monthly_exp_ts, k=1), series = "lag=1") +
  autolayer(stats::lag(monthly_exp_ts, k=12), series = "lag=12") +
  xlab("Month-Year") + ylab("Expenditures in bilion AUD") +
```

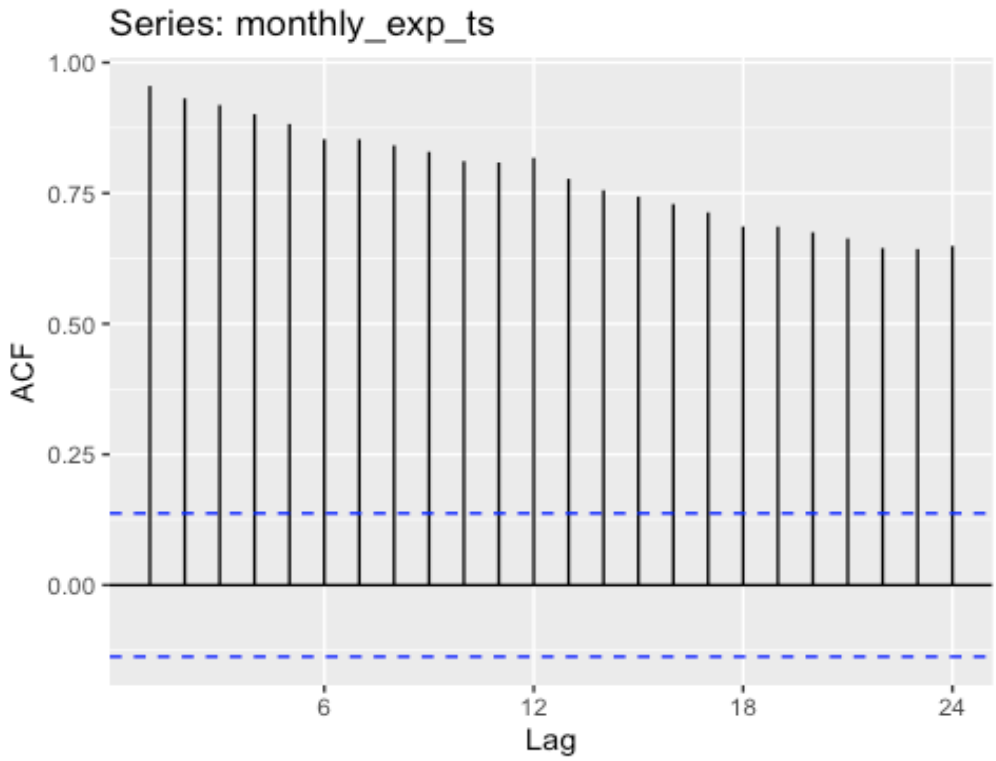
```
  guides(color=guide_legend(title = "Lagged values")) +
  theme_light()
```



Correlation measures the extent of a linear relationship between two variables. **Autocorrelation** measures the extent of linear relationship between lagged values of a time series.

A convenient way to examine autocorrelation is by plotting them:

```
ggAcf(monthly_exp_ts)
```



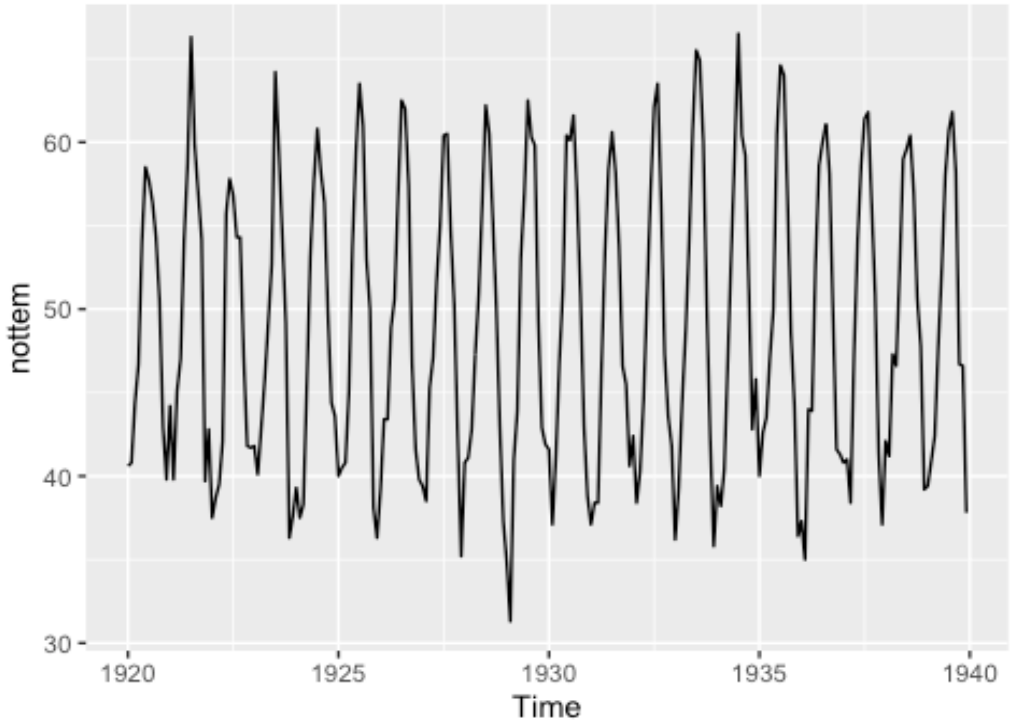
Note: the dashed blue lines indicate whether the autocorrelations are significantly different from zero.

Why / how autocorrelations can be useful?

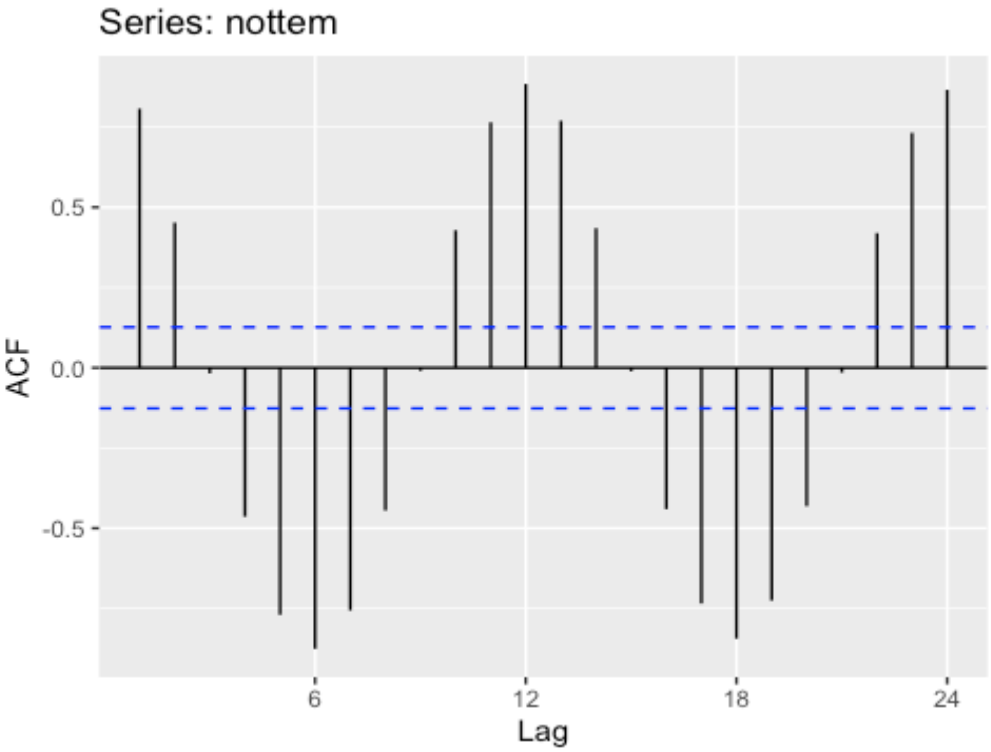
One reason is that they can help in identifying the kind of pattern(s) present in the time series:

- When data have a trend, autocorrelations for small lags tend to be large and positive (nearby values are similar), and they slowly decrease as the lags increase; this can be observed on the above given plot.
- When data are seasonal, autocorrelations are larger for the seasonal lags (lags at multiples of the seasonal frequency) than for other lags. As an example, recall the time series with average monthly temperature in Nottingham

```
autoplot(nottem)
```

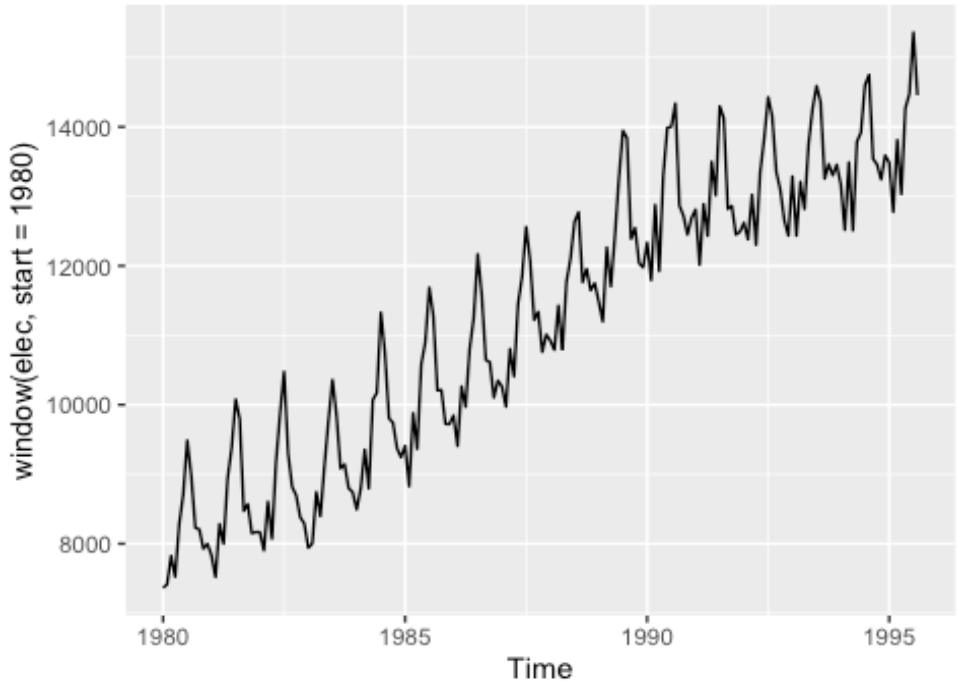



```
ggAcf(nottem)
```

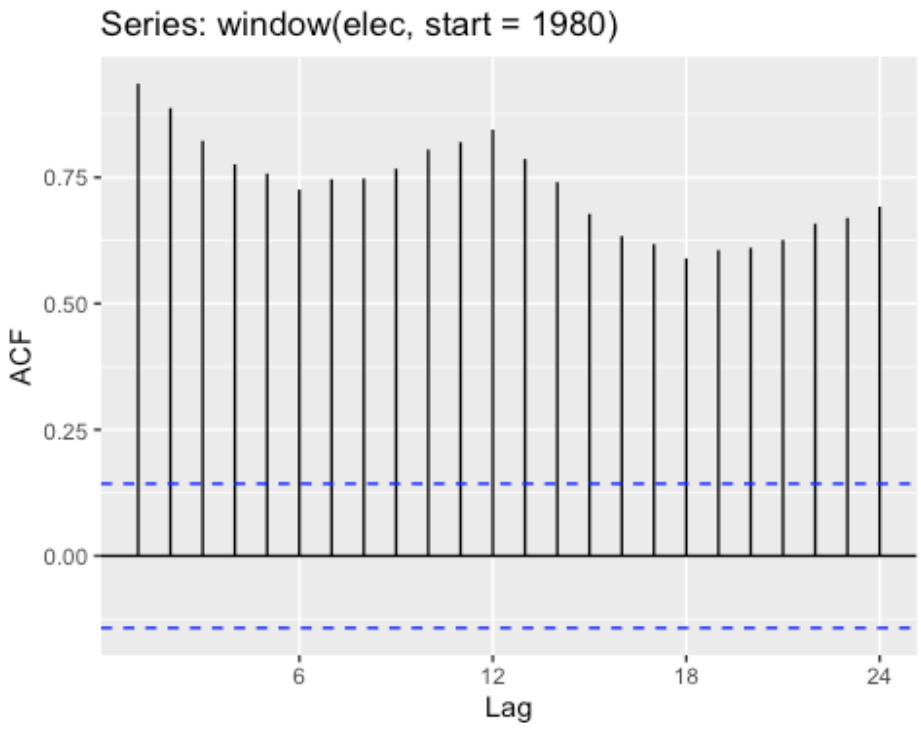


- When data are both seasonal and have a trend, a combination of the above effects are visible in the ACF charts. To illustrate that, we'll use as an example Australian monthly electricity production from 1980 till 1995.

```
autoplot(window(elec, start=1980))
```

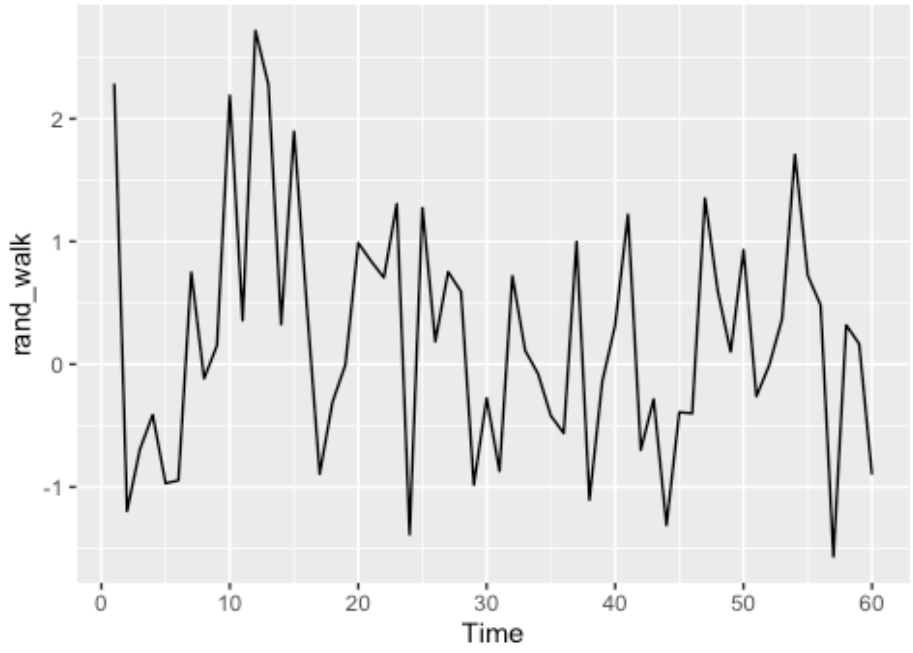



```
ggAcf(window(elec, start=1980))
```

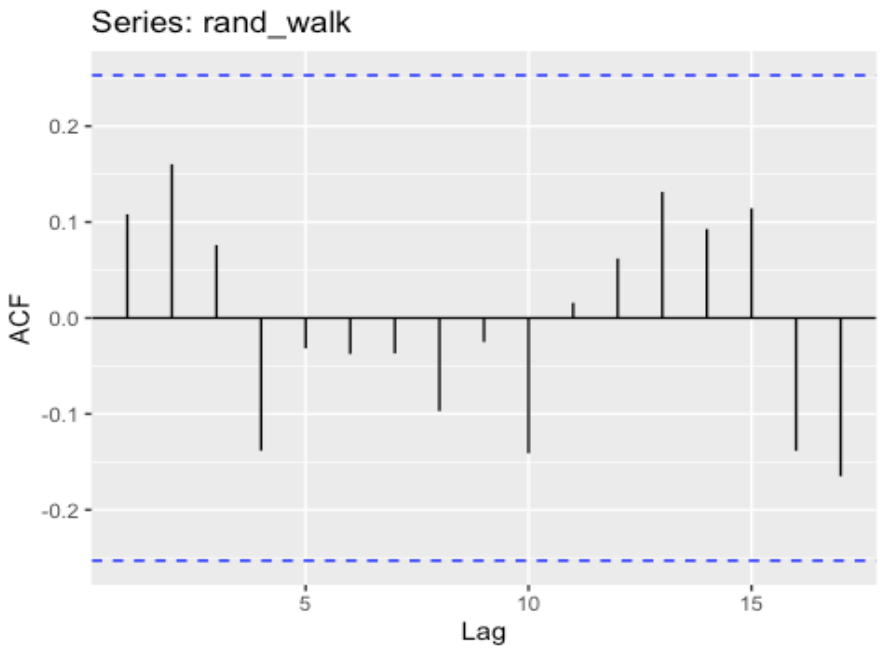


White noise. Time series that show no autocorrelation are called white noise. They are used for comparison purpose, to facilitate the detection of patterns in time series data

```
set.seed(7)
rand_walk <- ts(rnorm(60))
autoplot(rand_walk)
```




```
ggAcf(rand_walk)
```



The other way that autocorrelations and ACF plot can be useful is for the creation of predictor variables, that is, variables to be used as input variables for time series forecasting.

Time Series Regression Models

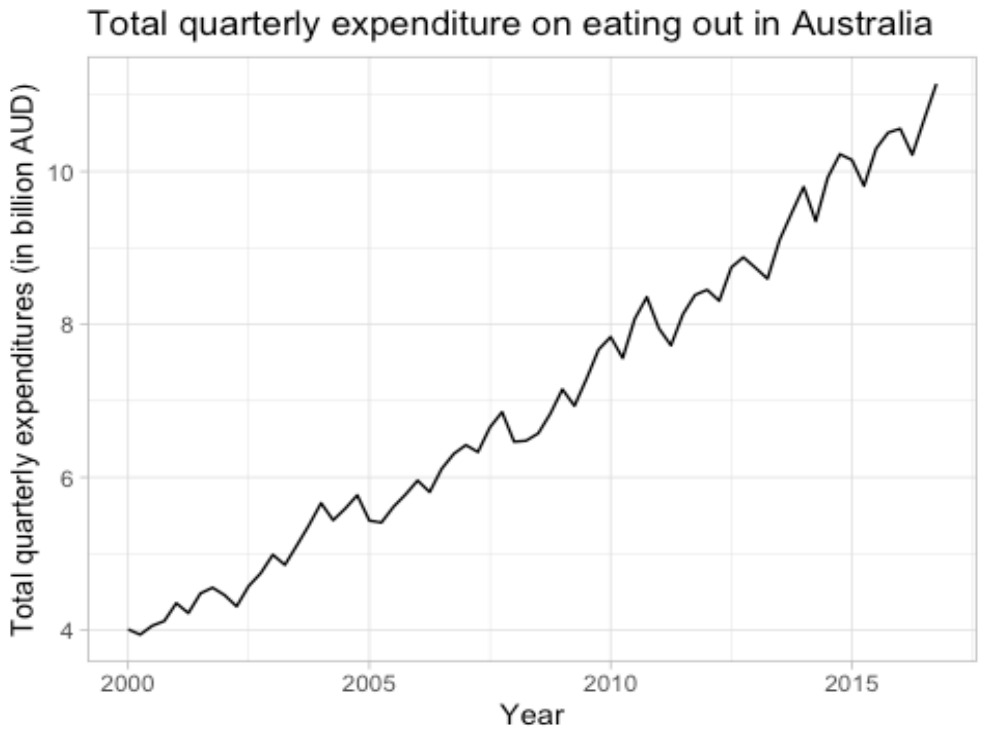
For univariate time series regression, we'll continue with the data about Australian eat out expenditures, but we'll now consider them at the quarterly level (just for the sake of having convenient data granularity for the analysis):

```
# read the data from the "data/quarterly_eatout_expenditures.csv" file
quarterly_exp <- read.csv("data/quarterly_eatout_expenditures.csv")
str(quarterly_exp)

## 'data.frame':    68 obs. of  3 variables:
##  $ year    : int  2000 2000 2000 2000 2001 2001 2001 2001 2002 2002 ...
##  $ quartile: Factor w/ 4 levels "Q1","Q2","Q3",..: 1 2 3 4 1 2 3 4 1 2 ...
##  $ tot_exp : num  4.01 3.94 4.06 4.12 4.35 ...
```

Create and plot a ts object:

```
quarterly_exp_ts <- ts(quarterly_exp$tot_exp, start = c(2000,1), frequency =
4)
autoplot(quarterly_exp_ts) +
  xlab("Year") + ylab("Total quarterly expenditures (in billion AUD)") +
  ggtitle("Total quarterly expenditure on eating out in Australia") +
theme_light()
```



Splitting the time series data into training and test sets

Before proceeding with model building, we need to split the data into training and test sets. Unlike before, with time series, we cannot randomly pick observations for training and test sets, but have to consider their chronological order. With seasonal data, we should also consider the frequency of seasonality. For instance, in the example we've been examining, the season consists of 4 observations (quartiles). So, a test set of 8 observations (= 2 years) seams as a meaningful choice (even a smaller would do if we are interested in short term forecasts).

```
exp_test <- tail(quarterly_exp_ts, 8)
exp_test
```

```
##         Qtr1    Qtr2    Qtr3    Qtr4
## 2015 10.1497  9.8108 10.2985 10.5075
## 2016 10.5591 10.2174 10.6867 11.1465
```

```
exp_train <- window(quarterly_exp_ts, end=c(2014, 4))
tail(exp_train, 4)
```

```
##         Qtr1    Qtr2    Qtr3    Qtr4
## 2014  9.7978  9.3490  9.9227 10.2255
```

Let's also load the content of the 'util.R' script with some auxiliary functions that we'll need along the way.

```
source('util.R')
```

Now, we can proceed with the model creation.

Regression using trend and season as predictors

If we know that our data has a (linear) trend and/or a seasonal pattern we can use trend and/or season as predictors in a (time series) regression model. The data in our example seem to have both, so, we will build a regression model using trend and season as predictors.

Note that the seasonal component in this case is quarterly. Season will be represented with 3 "dummy" variables, that is, binary (0/1) variables that when combined determine the quartile:

Q1: 0 0 0

Q2: 1 0 0

Q3: 0 1 0

Q4: 0 0 1

So, let's create a regression model using the tslm() function

```
exp_tslm <- tslm(exp_train ~ trend + season)
summary(exp_tslm)
```

```
##
## Call:
## tslm(formula = exp_train ~ trend + season)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.49179 -0.15388 -0.01787  0.14168  0.56012
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.633826   0.079288  45.831  < 2e-16 ***
## trend        0.099236   0.001757  56.470  < 2e-16 ***
## season2     -0.259823   0.085918  -3.024  0.00378 **
## season3     -0.040513   0.085972  -0.471  0.63934
## season4      0.077378   0.086061   0.899  0.37252
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2352 on 55 degrees of freedom
## Multiple R-squared:  0.9833, Adjusted R-squared:  0.982
## F-statistic: 807.8 on 4 and 55 DF,  p-value: < 2.2e-16
```

The output of the summary function is the same as we saw earlier when using the lm() function and should be interpreted in the same way. The only difference is in relation to the *season* predictor. Note that the missing season value (season1 or Q1) corresponds to the intercept of the linear model. Therefore, when interpreting the season variables we do so with respect to the season1, that is, with respect to the intercept. So, for example, season2 proved to be a significant predictor with coefficient -0.259823 meaning that the value of the output variable - expenditures on eating out - in Q2 is predicted to be by 0.259823 billion AUD smaller than in Q1. Note that this is not specific to time series regression models, but is the way factor variables are interpreted in linear models.

Examine the elements of the tslm model:

```
names(exp_tslm)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "contrasts"     "xlevels"       "call"          "terms"
## [13] "model"         "data"          "x"             "method"
```

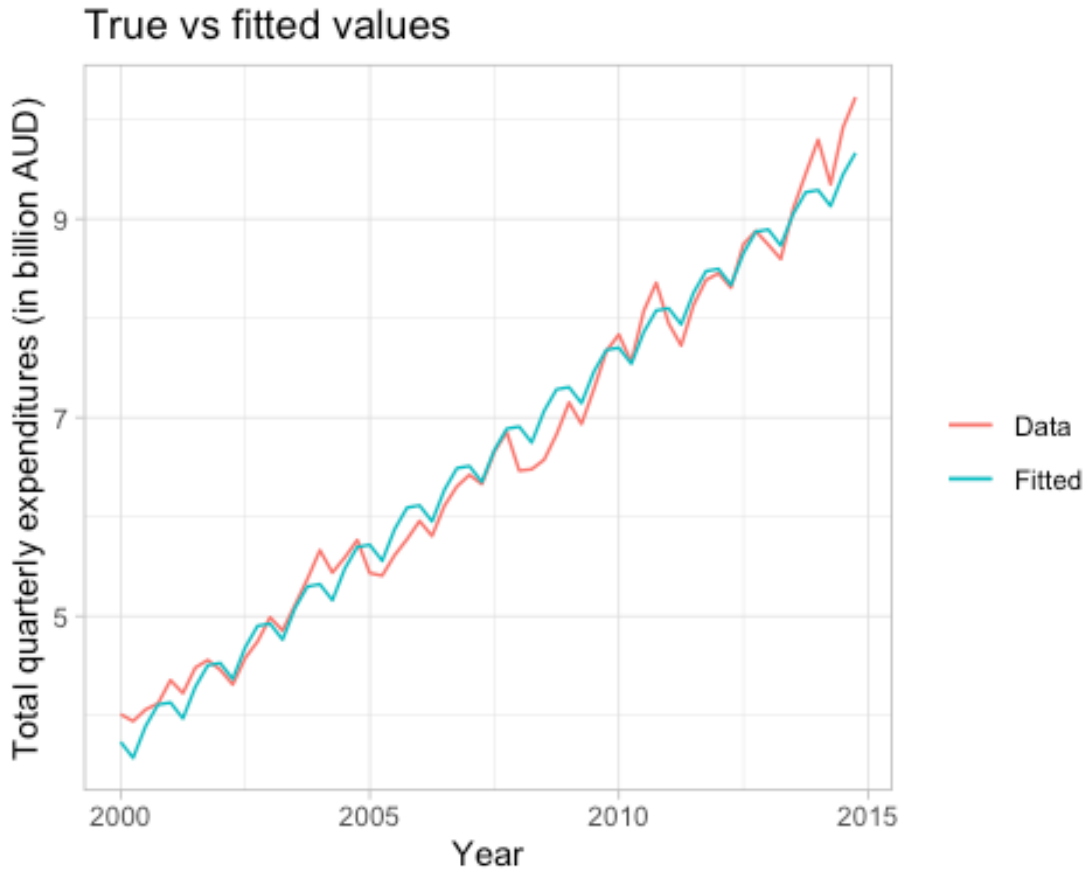
Note a large overlap with the lm model. For example, we can get coefficients of the linear model with:

```
coefficients(exp_tslm)
```

```
## (Intercept)       trend     season2     season3     season4
##  3.63382616  0.09923634 -0.25982301 -0.04051268  0.07737765
```

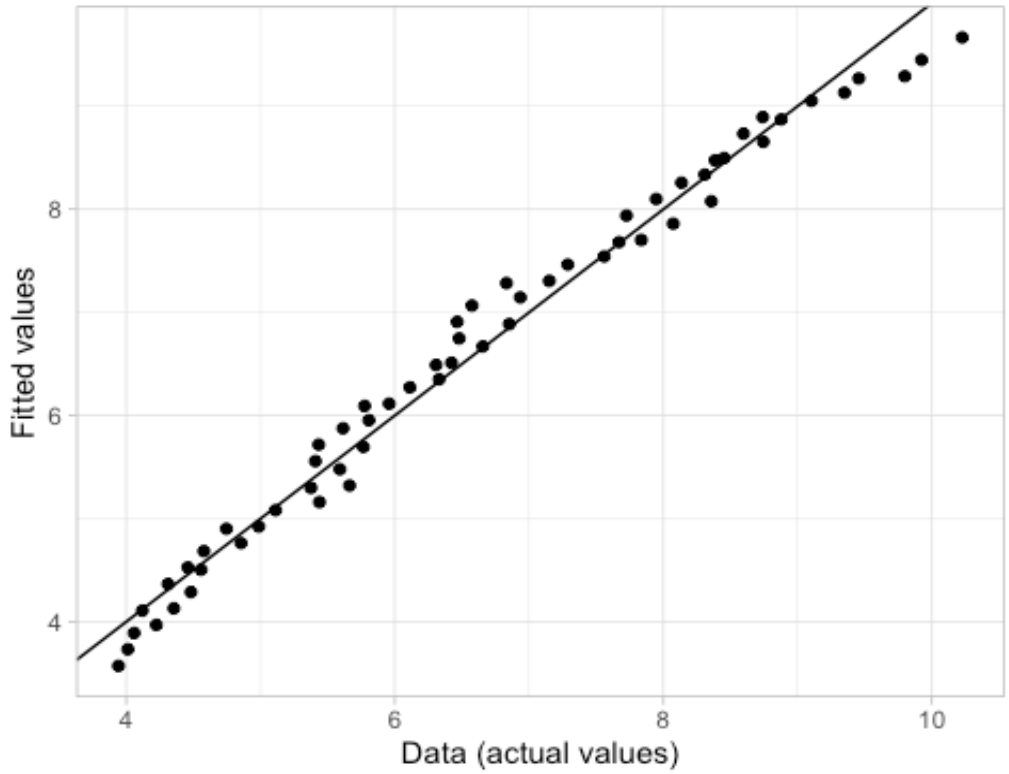
To get an initial insight how good our predictions are, we can plot the fitted against the actual time series:

```r
autoplot(exp_train, series = "Data") +
  autolayer(fitted(exp_tslm), series = "Fitted") +
  guides(color = guide_legend(title = "")) +
  xlab("Year") + ylab("Total quarterly expenditures (in billion AUD)") +
  ggtitle("True vs fitted values") +
  theme_light()
```



We can also examine this by creating a scatter plot of actual (x-axis) and fitted values (y-axis), expecting to see them concentrated along the diagonal line that goes through the origin.

```r
# Note that the true_vs_fitted_plot() function is defined in the util.R
script
true_vs_fitted_plot(true_vals = exp_train, fitted_vals = fitted(exp_tslm))
```



Making predictions (forecasts)

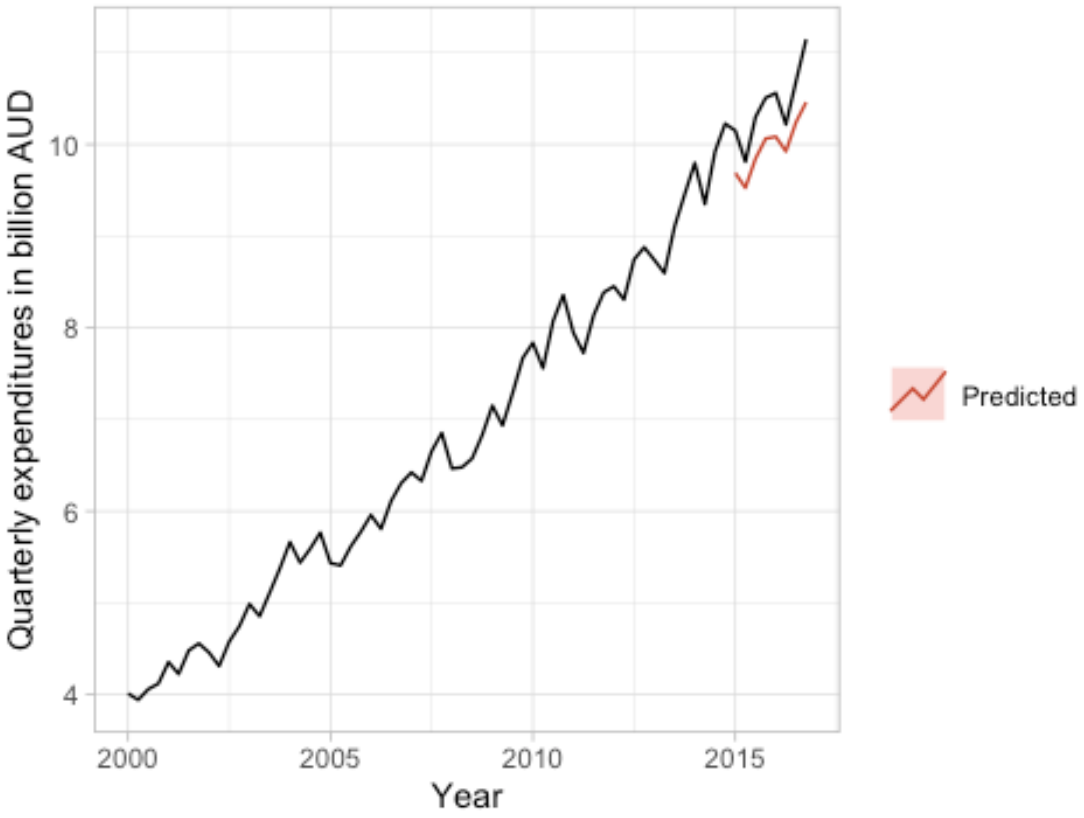
Now, we use the test set to make predictions

```
pred_tslm <- forecast(exp_tslm, newdata = as.data.frame(exp_test))
tail(pred_tslm$fitted, 8)
```

```
##          Qtr1     Qtr2     Qtr3     Qtr4
## 2013 8.893352 8.732765 9.051312 9.268439
## 2014 9.290297 9.129711 9.448258 9.665384
```

Plot the predictions

```
autoplot(quarterly_exp_ts) +
  autolayer(pred_tslm, series = "Predicted", PI = FALSE) +
  guides(color=guide_legend(title="")) +
  ylab("Quarterly expenditures in billion AUD") +
  xlab("Year") +
  theme_light()
```



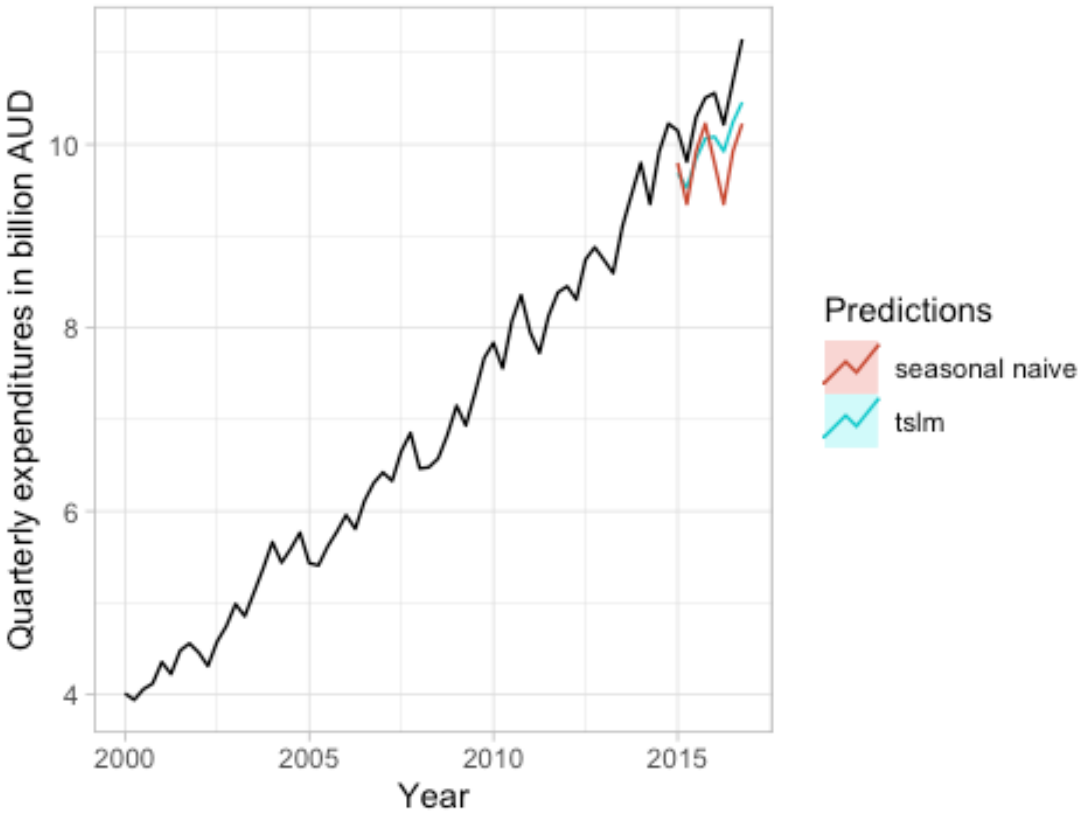
To get a first impression of how good our predictions are, we will compare them with the predictions of some simple forecasting methods that are often used as baselines in time series analysis.

A suitable method in our case would be the *seasonal naive method*. The *naive method* simply sets all forecasts to be equal to the value of the last observation; such forecasts are also known as *random walk forecasts*. For seasonal data, the naive method is somewhat altered, so that it sets each forecast to be equal to the last value of the same season (e.g., to set value for temp. in Jan, it will use temp. in Jan of the previous year).

```
exp_snaive <- snaive(exp_train, h=length(exp_test))
```

Note: **h** is the length of the forecast period

```
autoplot(quarterly_exp_ts) +
  autolayer(pred_tslm, series = "tslm", PI = FALSE) +
  autolayer(exp_snaive, series = "seasonal naive", PI = FALSE) +
  guides(color=guide_legend(title="Predictions")) +
  ylab("Quarterly expenditures in billion AUD") +
  xlab("Year") +
  theme_light()
```



The plot suggests that the (seasonal) naive method is not that bad. We will now verify that by computing a set of evaluation measures often used for assessing the accuracy of forecasts.

Evaluation measures

A variety of evaluation measures are used to assess the forecasting performance. We will focus only on a small subset of the most frequently used measures.

Scale-dependent measures include:

- MAE = Mean Absolute Error. It is computed as: $mean(|e_t|)$, where e_t is the error (residual) in time t.
- RMSE = Root Mean Squared Error. It is computed as: $sqrt(mean(e_t^2))$

Scale-independent measures include:

- MAPE = Mean Absolute Percentage Error. It is computed as: $mean(|p_t|)$, where $p_t = 100 * e_t/y_t$, e_t is the error (residual) in time t, and y_t is the prediction at time t.
- MASE = Mean Absolute Scaled Error, where scaling is done using (seasonal) naive predictions; MASE is < 1 if the forecast are better than those of the (seasonal) naïve method.

The advantage of the latter group of measures is that they are unit-free, and thus can be used to compare predictions (of the same or different models) across data sets.

The accuracy() function computes these plus many other evaluation measures:

```
?accuracy
accuracy(pred_tslm, exp_test)
```

```
##                         ME      RMSE       MAE        MPE     MAPE
## Training set -1.480839e-17 0.2252313 0.1784808 0.03683804 2.838162
## Test set      4.431945e-01 0.4582607 0.4431945 4.22365621 4.223656
##                    MASE        ACF1 Theil's U
## Training set 0.4184532  0.68440399        NA
## Test set     1.0390820 -0.08465492  1.228686
```

We'll select measures we are interested in, computed on the test set using our regression model and the seasonal naive model:

```
eval_masures <- c('MAE', 'RMSE', 'MAPE', 'MASE')
tslm_eval <- accuracy(pred_tslm, exp_test)[2, eval_masures]
snaive_eval <- accuracy(exp_snaive, exp_test)[2, eval_masures]
rbind(tslm_eval, snaive_eval)
```

```
##                   MAE      RMSE     MAPE     MASE
## tslm_eval   0.4431945 0.4582607 4.223656 1.039082
## snaive_eval 0.5982750 0.6445421 5.703488 1.402673
```

Our regression model is clearly better than the seasonal naive model.

Checking assumptions of the linear model

There is a set of assumptions that a linear regression model has to satisfy (as we discussed before, in the linear regression lab). These assumptions are expressed in relation to the model's residuals. The key assumptions for time series linear regression models include:

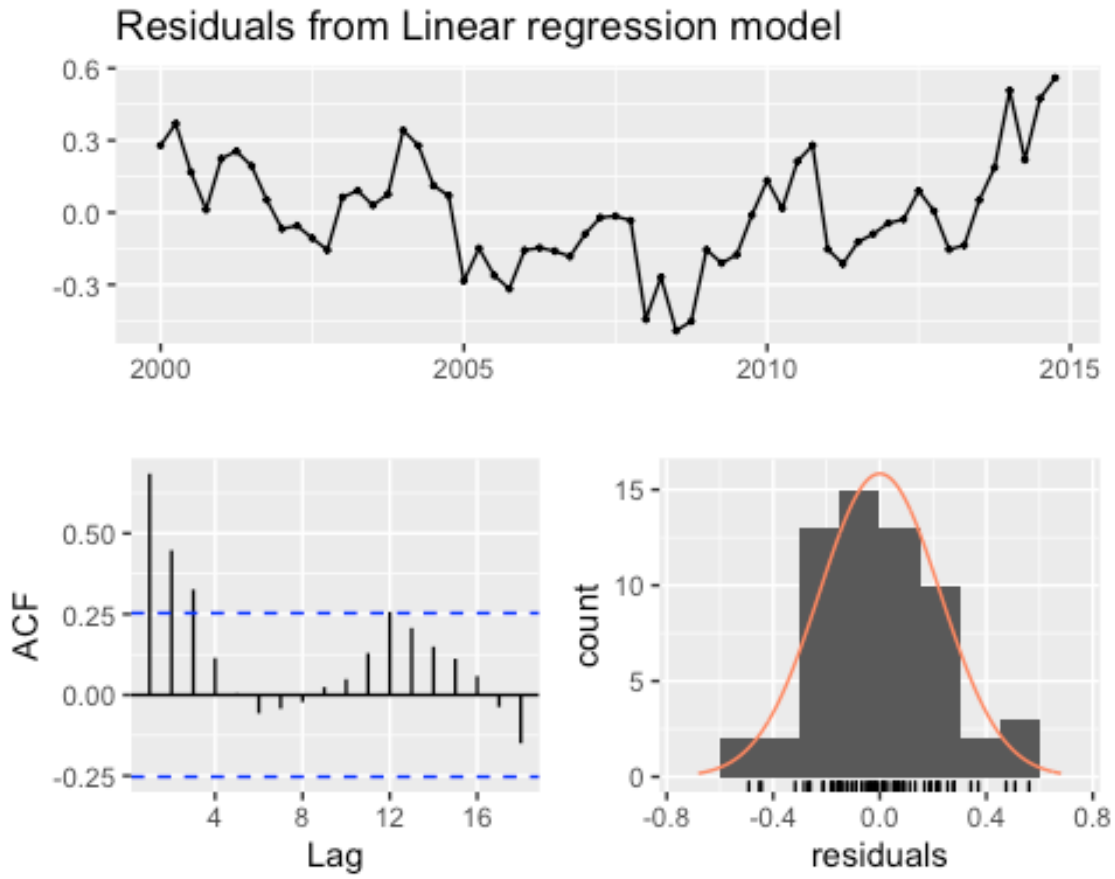
1. the residuals have mean value equal to zero; otherwise the predictions will be systematically biased
2. the residuals are not autocorrelated; otherwise there is more information in the data that can be exploited (ie, the model failed to capture all the patterns in the data).
3. the residuals are unrelated to the predictor variables; otherwise there is more information that should be included in the model (additional variables).

Two additional assumptions, required for easy computation of prediction intervals:

4. the residuals are normally distributed
5. the residuals have a constant variance.

To check for all the above assumptions except assumption 3, we'll use the checkresiduals() function as it conveniently gathers different methods for checking residuals:

```
checkresiduals(exp_tslm)
```




```
##
##  Breusch-Godfrey test for serial correlation of order up to 8
##
## data:  Residuals from Linear regression model
## LM test = 32.797, df = 8, p-value = 6.699e-05
```

The first plot shows the difference between the actual and the fitted values (i.e. residuals) for the overall time period covered by the data. It allows for checking assumption 5 - homoschedasticity of residuals, that is, if residuals have constant variance. The pattern on the plot indicates that this is not the case here.

The second plot (first in the second row) is the ACF (autocorrelations function) plot for the residuals. It is used for checking assumption 2 - that residuals are not autocorrelated. The fact that some of the autocorrelations are above the significance level (dashed blue line) suggests that this assumption is not satisfied. That is further confirmed by the Breusch-Godfrey test. This test is used for testing the hypothesis (H0) that there is no autocorrelation in the residuals up to a certain specified order (in this case up to 8, as we were making 8 predictions). A significant test (small p-value) indicates a significant autocorrelation remaining in the residuals.

The third plot (second in the second row) allows for checking assumptions 1 and 4. However, in this particular case, it is difficult to say with certainty if the residuals follow normal distribution and whether mean is zero. We can further check that as follows:

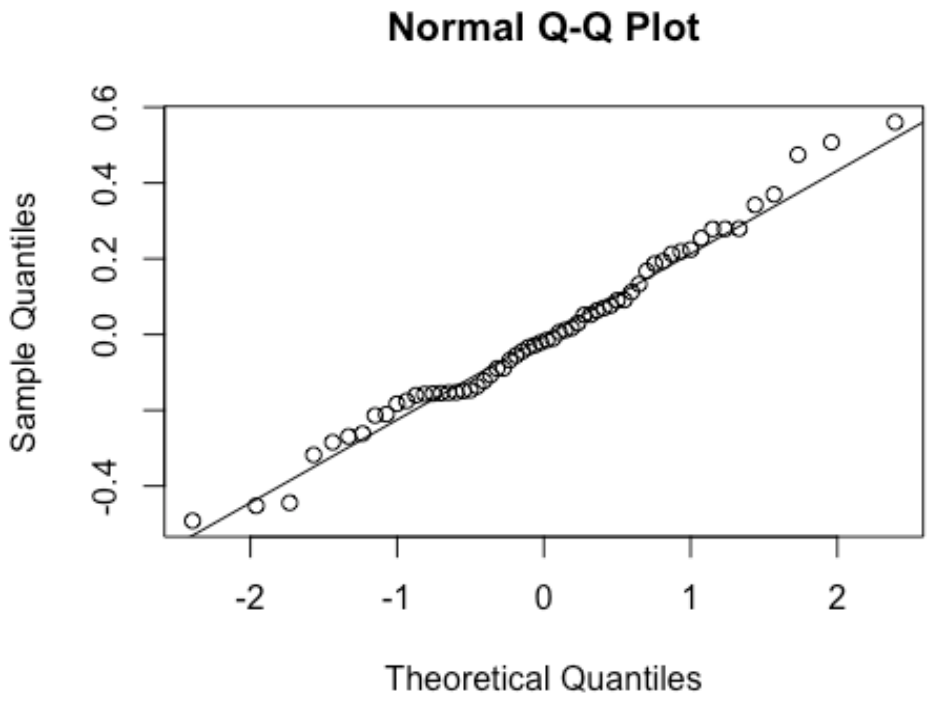
```r
mean(residuals(exp_tslm))
```

```
## [1] -2.09251e-18
```

```r
shapiro.test(residuals(exp_tslm))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(exp_tslm)
## W = 0.98337, p-value = 0.5861
```

```r
qqnorm(residuals(exp_tslm))
qqline(residuals(exp_tslm))
```



To sum up, we have found that residuals are autocorrelated, meaning that our model does not capture all the patterns present in the data. Furthermore, the finding that residuals do not have constant variance indicates that prediction intervals need to be computed through bootstrapping or some other complex method.

Note that we haven't examined assumption 3. This is because in this case predictors are derived from the time series itself, making this assumption inapplicable. We'll examine it in the context of the next model.

Multivariate (time series) regression models

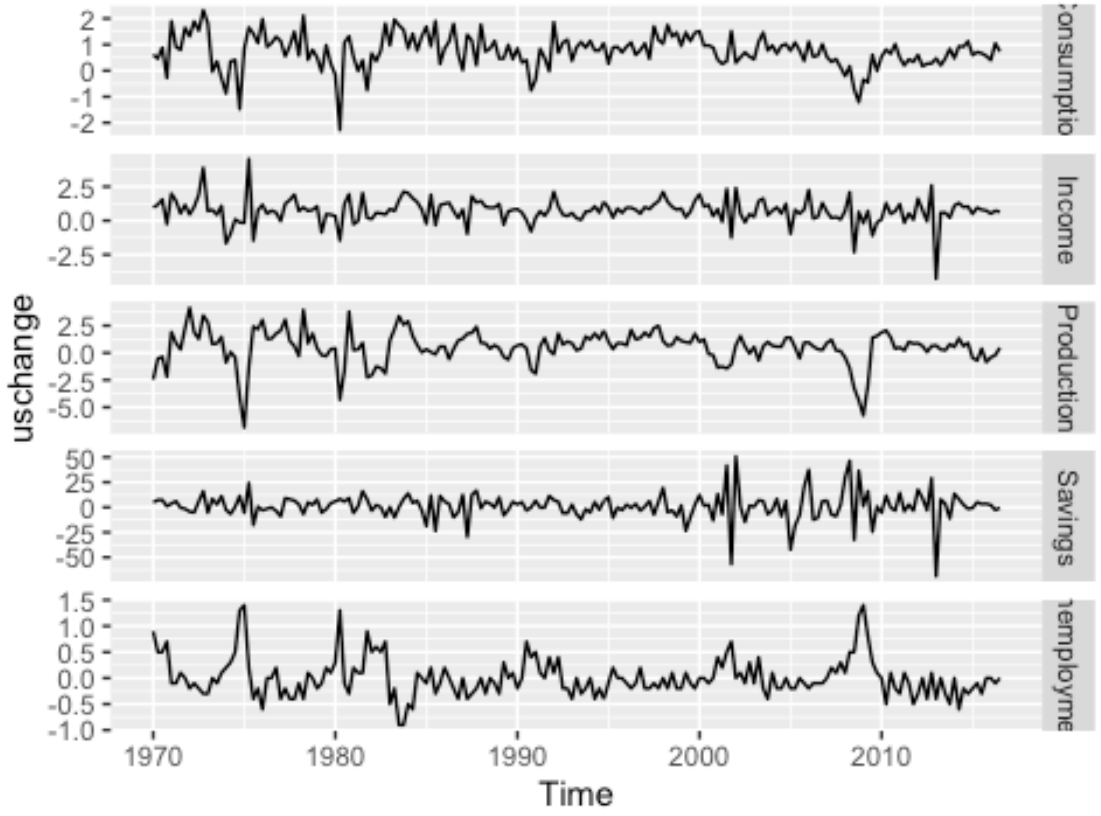
Let's now consider a multivariate regression problem with time series (TS) data, that is, predicting one TS based on two or more of other (related) TS. To do that, we'll use another (richer) dataset:

```
?uschange
head(uschange)
```

```
##         Consumption     Income Production   Savings Unemployment
## 1970 Q1   0.6159862  0.9722610 -2.4527003 4.8103115          0.9
## 1970 Q2   0.4603757  1.1690847 -0.5515251 7.2879923          0.5
## 1970 Q3   0.8767914  1.5532705 -0.3587079 7.2890131          0.5
## 1970 Q4  -0.2742451 -0.2552724 -2.1854549 0.9852296          0.7
## 1971 Q1   1.8973708  1.9871536  1.9097341 3.6577706         -0.1
## 1971 Q2   0.9119929  1.4473342  0.9015358 6.0513418         -0.1
```

It is a data set of percentage changes in quarterly personal consumption expenditure, personal disposable income, production, savings, and the unemployment rate for the US, from 1960 to 2016.

```
autoplot(uschange, facets = TRUE)
```



We'll set as our objective to create a model that can predict the change in quarterly personal consumption based on the quarterly change in:

- personal disposable income
- production,
- savings,
- the unemployment rate.

We'll start by splitting the data into training and test sets. To decide on the split, let's take a closer look at the length of the time series, their beginning and end

```
dim(uschange)
```

```
## [1] 187   5
```

```
head(uschange)
```

```
##         Consumption     Income Production   Savings Unemployment
## 1970 Q1   0.6159862  0.9722610 -2.4527003 4.8103115          0.9
## 1970 Q2   0.4603757  1.1690847 -0.5515251 7.2879923          0.5
## 1970 Q3   0.8767914  1.5532705 -0.3587079 7.2890131          0.5
## 1970 Q4  -0.2742451 -0.2552724 -2.1854549 0.9852296          0.7
## 1971 Q1   1.8973708  1.9871536  1.9097341 3.6577706         -0.1
## 1971 Q2   0.9119929  1.4473342  0.9015358 6.0513418         -0.1
```

```
tail(uschange)
```

```
##         Consumption    Income Production    Savings Unemployment
## 2015 Q2   0.7081439 0.9549595 -0.6970216  5.0239177         -0.1
## 2015 Q3   0.6649696 0.8016627  0.3806061  3.1809298         -0.3
## 2015 Q4   0.5616798 0.7400626 -0.8455464  3.4827860          0.0
## 2016 Q1   0.4046822 0.5190254 -0.4179305  2.2365341          0.0
## 2016 Q2   1.0477074 0.7237208 -0.2033188 -2.7215011         -0.1
## 2016 Q3   0.7295978 0.6447008  0.4749184 -0.5728579          0.0
```

Note that the last year (2016) lacks the last quartile (Q4).

Since the time period is long (1970-2016), we can take the last 5 years for the test set

```
uschange_test <- tail(uschange, 4*4+3)
head(uschange_test)
```

```
##         Consumption      Income Production    Savings Unemployment
## 2012 Q1   0.6010900  1.62204885 0.88651817  17.625305         -0.3
## 2012 Q2   0.1694296  0.76689543 0.62923586   8.969497          0.0
## 2012 Q3   0.2641603 -0.05071452 0.07880166  -3.049222         -0.4
## 2012 Q4   0.2787719  2.59106697 0.63305509  29.046704          0.1
## 2013 Q1   0.4686129 -4.26525047 0.67713243 -68.788267         -0.4
## 2013 Q2   0.2054580  0.58146541 0.30744961   7.816477          0.0
```

```
uschange_train <- window(uschange, end=c(2011, 4))
```

Now, create a linear model with Income, Production, Savings, and Unemployment as predictors:

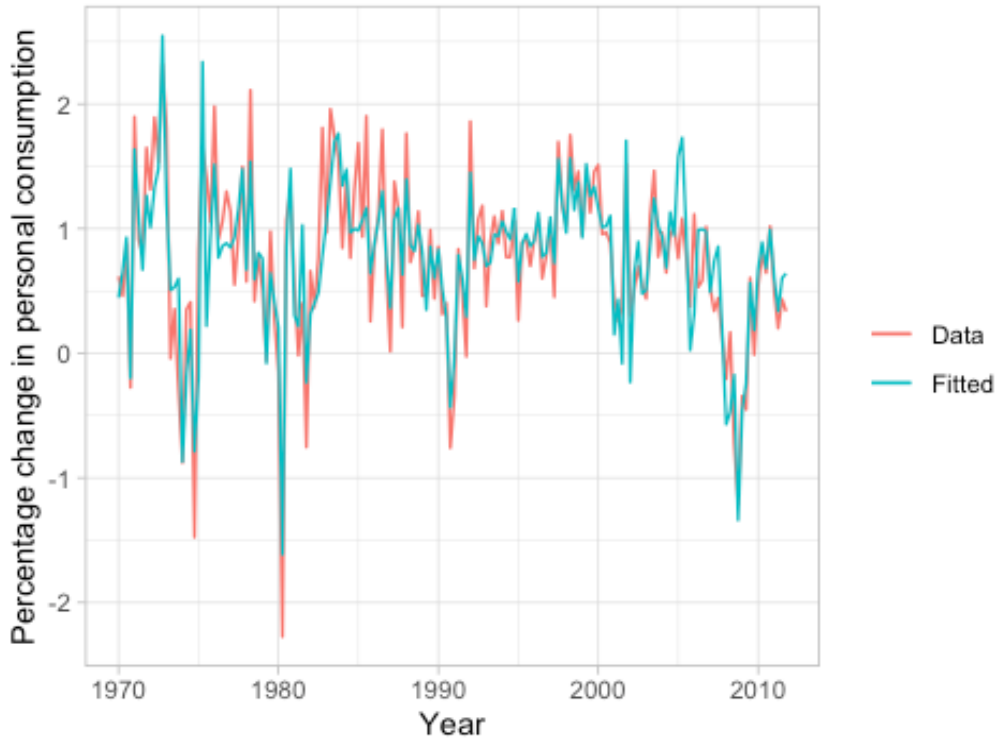
```
usch_tslm <- tslm(Consumption ~ Income + Production + Savings + Unemployment,
                  data = uschange_train)
summary(usch_tslm)
```

```
##
## Call:
## tslm(formula = Consumption ~ Income + Production + Savings +
##     Unemployment, data = uschange_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.89047 -0.19532 -0.03704  0.15917  1.20013
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.285751   0.043069   6.635 4.58e-10 ***
## Income        0.712491   0.045648  15.608  < 2e-16 ***
## Production    0.037131   0.028689   1.294   0.1974
## Savings      -0.044460   0.002951 -15.068  < 2e-16 ***
## Unemployment -0.253606   0.121178  -2.093   0.0379 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3417 on 163 degrees of freedom
## Multiple R-squared:  0.756,  Adjusted R-squared:   0.75
## F-statistic: 126.3 on 4 and 163 DF,  p-value: < 2.2e-16
```

The model identifies all variables except Production as important predictors. As expected, disposable income positively affects consumption, whereas savings and unemployment are negative associated. Adjusted R^2 indicates that our model explains 75% of variability in the Consumption.

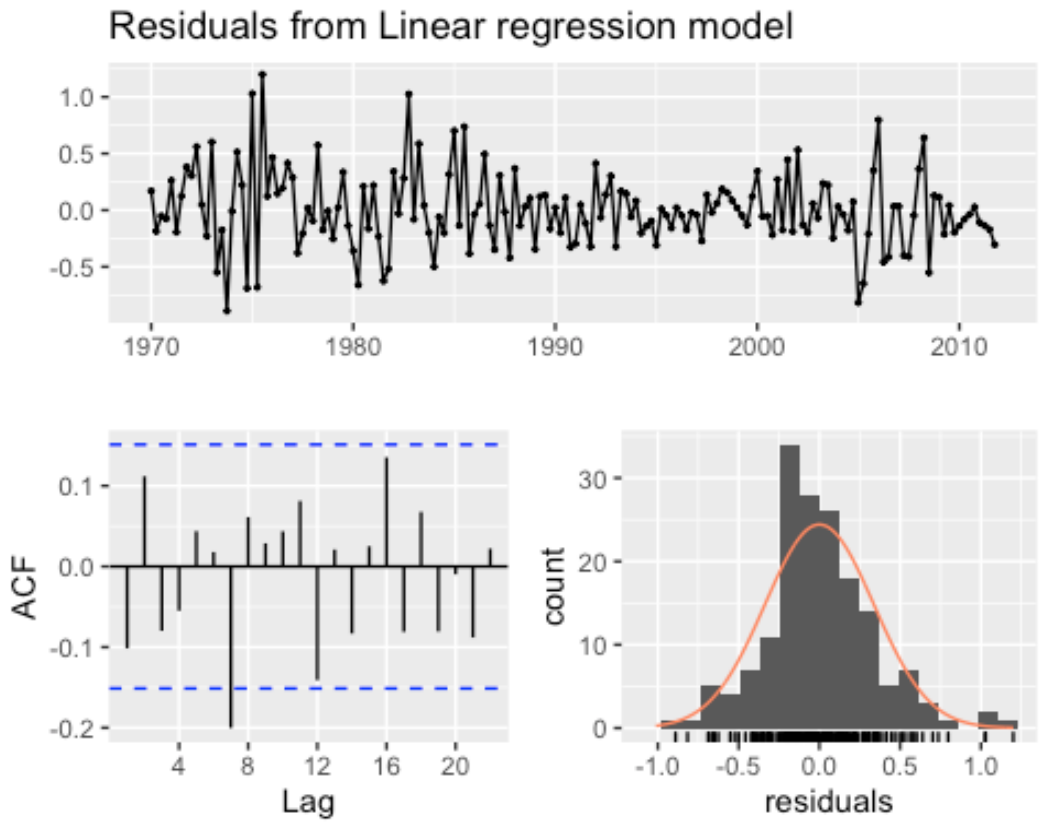
We can visually examine how well the fitted values match the data:

```
autoplot(uschange_train[,'Consumption'], series = 'Data') +
  autolayer(fitted(usch_tslm), series = 'Fitted') +
  ylab("Percentage change in personal consumption") +
  xlab("Year") +
  guides(color = guide_legend(title = "")) +
  theme_light()
```



We should also examine residuals before proceeding to prediction:

```
checkresiduals(usch_tslm)
```




```
##
##  Breusch-Godfrey test for serial correlation of order up to 8
##
## data:  Residuals from Linear regression model
## LM test = 14.523, df = 8, p-value = 0.06911
```

```
mean(residuals(usch_tslm))
```

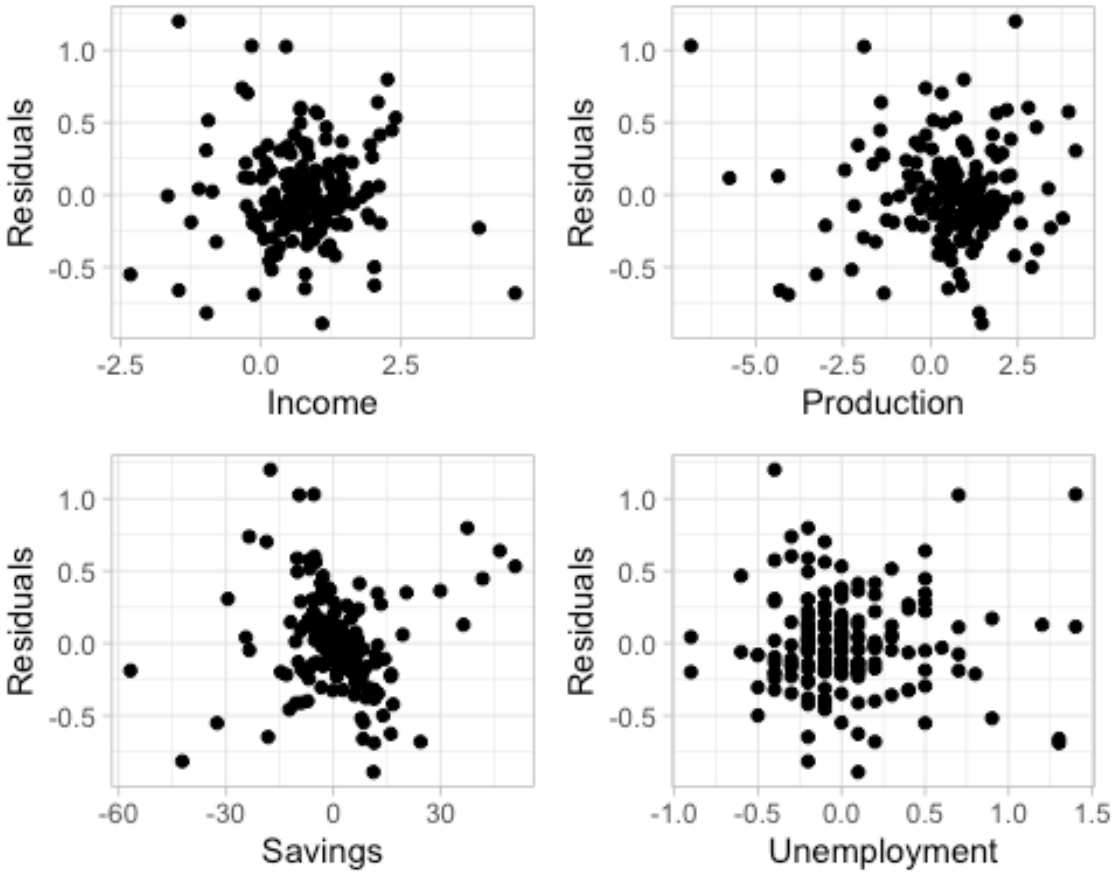
```
## [1] -1.26792e-17
```

The two key assumptions - mean value equal to zero and no autocorrelations - are satisfied. On the other hand, normal distribution of residuals and homoschedasticity do not apply.

We also need to verify the assumption that residuals are unrelated to the predictor variables. We will do that by plotting each predictor variable against the residuals.

```
uschange_train_df <- as.data.frame(uschange_train)
uschange_train_df$Residuals <- as.numeric(residuals(usch_tslm))

# the following function is defined in the util.R script
residuals_against_predictors_plot(df = uschange_train_df,
                                  predictor_lbls =
colnames(uschange_train_df)[2:5],
                                  residuals_lbl = 'Residuals')
```



That the points are randomly scattered on the plots confirms that predictors and residuals are not associated.

All in all, the model is generally fine. However, prediction intervals have to computed in a more complex manner (eg via bootstrapping), but that is out of scope of this lab.

Evaluate the model on the test set

In the case of multivariate time series, we need to distinguish between two types of predictions (forecasts), depending on what is assumed to be known when the predictions are computed:

- *Ex-ante forecasts* are those that are made using only the information that is available at the time when predictions are made. The tricky part here is that in order to make such predictions, the model requires forecasts of the predictor variables.
- *Ex-post forecasts* use the actual observations of the predictors, once these become available. These are not genuine forecasts, but are useful for studying the behaviour of forecasting models.

A comparison of ex-ante and ex-post forecasts can help in detecting the source of forecast errors: if they originate primarily in the poor forecast of the predictors or are due to a poor forecasting model.

The predictions that we will do obviously belong to the ex-post category.

Make predictions on the test set:

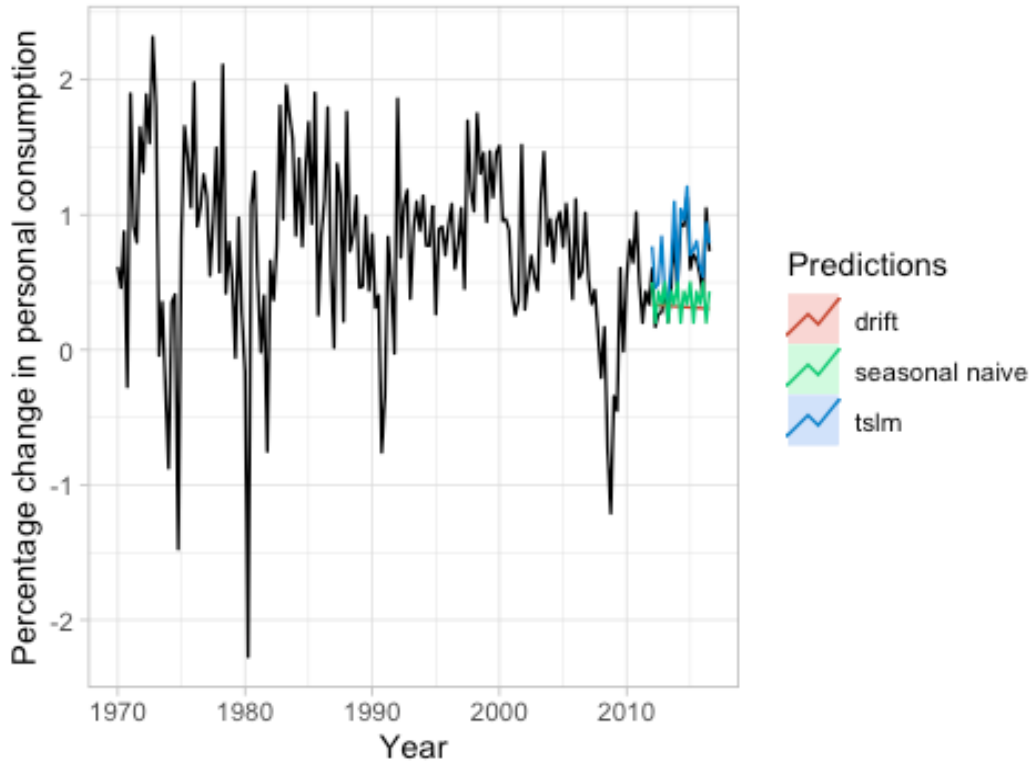
```
usch_tslm_pred <- forecast(usch_tslm, newdata = data.frame(uschange_test))
```

As a baseline model, in addition to the seasonal naive model, we can use the *drift model*. The drift model is a variation of the naïve model, which allows the forecasts to change over time, where the amount of change (drift) is equal to the average change seen in the historical data.

```
pred_horizon <- nrow(uschange_test)
usch_snaive <- snaive(uschange_train[,'Consumption'], h=pred_horizon)
usch_drift <- rwf(uschange_train[,'Consumption'], h=pred_horizon, drift =
TRUE)
```

Compare the predictions of the linear model and the two baselines, first, by plotting them:

```
autoplot(uschange[,'Consumption']) +
  autolayer(usch_tslm_pred, series = "tslm", PI = FALSE) +
  autolayer(usch_drift, series = 'drift', PI = FALSE) +
  autolayer(usch_snaive, series = 'seasonal naive', PI = FALSE) +
  ylab("Percentage change in personal consumption") +
  xlab("Year") +
  guides(color = guide_legend(title = "Predictions")) +
  theme_light()
```



The plot shows that in this case simple methods are not able to capture the pattern in the data.

Now, use the accuracy() function to evaluate the models' performance using evaluation measures (MAE, RMSE, MAPE, MASE):

```
models <- list(usch_snaive, usch_drift, usch_tslm_pred)
perf_eval <- lapply(models, function(m) {
  accuracy(m, uschange_test[,'Consumption'])[2,eval_masures]})

perf_eval_df <- do.call(rbind.data.frame, perf_eval)
colnames(perf_eval_df) <- eval_masures
rownames(perf_eval_df) <- c('snaive', 'drift', 'tslm')
perf_eval_df

##              MAE      RMSE     MAPE      MASE
## snaive 0.2768647 0.3890224 37.40000 0.4044021
## drift  0.3259415 0.3979125 50.25134 0.4760860
## tslm   0.1451565 0.1898651 40.48077 0.2120226
```

The regression model is significantly better than the simple forecasting models.

Acknowledgement. This annotated script is partially based on several chapters of the book "Forecasting: Principles and Practice" by Rob J Hyndman and George Athanasopoulos. The book is available online at: https://otexts.com/fpp2/