Code Explanation (HW2)

Anthony Tricarico

Table of contents

1 Explanation of code

Here follows a simple explanation of the most used pieces of code encountered so far which can be useful for different purposes.

1.1 Loading libraries

These are the libraries we will be using for now. I will import some extra ones because I might need them later on.

```r
library(tsibbledata)
library(TSA)
library(ggplot2)
library(fpp3)
library(gridExtra)
library(tidyquant)
library(forecast)
library(readr)
library(latex2exp)
library(fma)

setwd("~/Desktop/UniTrento/Tutorato/Software R (English)/DAF/homeworks/Homework2")
```

1.2 Useful functions to generate sequences

It is common in the exercises encountered so far to have to generate some dates in a specific range, either because those are not present in the original dataset or because they are not in the correct datatype (i.e., not formatted as dates). The following functions are useful to generate dates (or better, vectors of dates) which can then be bound to the original dataset (i.e., added as a column)

The built-in *R* function *seq* generates sequences of number in the range specified by its arguments (*from* and *to*) in steps specified by the argument *by*.

```r
seq(from = 1, to = 10, by = 2)
```

```
[1] 1 3 5 7 9
```

Even though the built-in date type in *R* is at heart a number (when converted to its integer form) it is often better to work with functions that are specific to dates such as ***seq.Date()*** which returns a vector of dates employing a similar logic to *seq*.

```r
#Generates dates from January 1st 2020 to December 31st 2020
#in monthly steps (i.e., one date for each month)
seq.Date(from = as.Date("2020-01-01"), to = as.Date("2020-12-31"), by = "month")
```

```
 [1] "2020-01-01" "2020-02-01" "2020-03-01" "2020-04-01" "2020-05-01"
 [6] "2020-06-01" "2020-07-01" "2020-08-01" "2020-09-01" "2020-10-01"
[11] "2020-11-01" "2020-12-01"
```

Tip

Notice how you can pass arguments into R functions. Arguments can be passed in either by following an order of positions or by calling the label (i.e., the name) associated to that argument and pass the input after the = sign. To experiment with this idea notice that:
`seq.Date(from = as.Date("2020-01-01"), to = as.Date("2020-12-31"), by = "month")`
and
`seq.Date(as.Date("2020-01-01"), as.Date("2020-12-31"), "month")`
produce exactly the same result. This is useful and applies to all functions in R. Try it for yourself!

1.3 Tsibbles

Tsibbles (short form Time Series Tibbles) allow us to model time series data in R. Tsibbles have much in common with tibbles (i.e., common data frames which you can think of as Excel spreadhseets), however they are indexed by dates. You can think of an index as a column that adds structure to the data you are analyzing (in this case you are adding a temporal structure to your dataset). This allows to perform operations that require data to be ordered chronologically (e.g., computing autocorrelation with ACF(), plotting correlograms with autoplot(), and so on)

```r
obs <- seq(1, 10) #a vector containing numbers from 1 to 10
dates <- seq.Date(from = as.Date("2020-01-01"), to = as.Date("2020-01-10"), by = "day")

tbl <- tibble(obs, dates)

tsbl <- tsibble(
  tbl,
  index = dates
)

tsbl
```

```
# A tsibble: 10 x 2 [1D]
     obs dates
   <int> <date>
 1     1 2020-01-01
 2     2 2020-01-02
 3     3 2020-01-03
 4     4 2020-01-04
 5     5 2020-01-05
 6     6 2020-01-06
 7     7 2020-01-07
 8     8 2020-01-08
 9     9 2020-01-09
10    10 2020-01-10
```

1.4 Computing statistics for Time Series

So far in the course you've explored some useful statistics that can be computed to describe your time series, spot patterns in your data, and determine the presence of the source of potential problems for your forecasts (e.g., autocorrelation). This is a brief recap of what has been done so far in the exercises.

1.4.1 Autocorrelation

Autocorrelation refers to how much correlated (if any) the observations in a time series are. Keep in mind that autocorrelation computes the correlation between a variable and its *lagged* values. A simple example follows below:

```r
hours <- read_csv("hours.dat")
```

```
Rows: 60 Columns: 1
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (1): hours

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

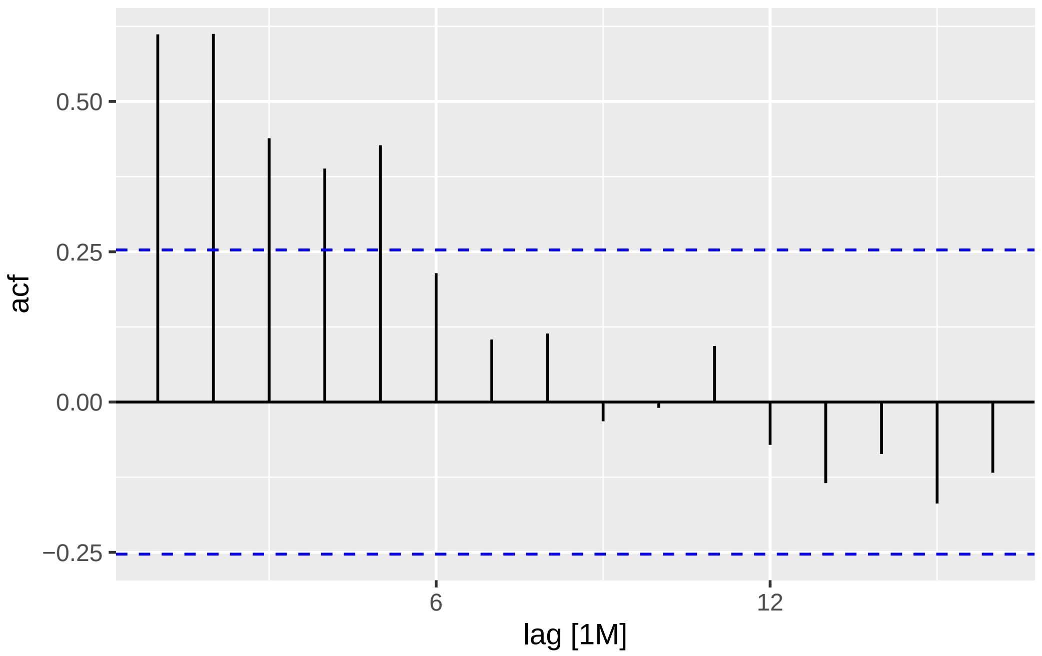
```r
hours$month <- seq.Date(as.Date("1982/07/01"), as.Date("1987/06/01"), "month")

hours <- hours %>%
  mutate(month = yearmonth(month)) #convert full dates to year-month representation

(hours_ts <- as_tsibble(hours, index = month))
```

```
# A tsibble: 60 x 2 [1M]
   hours    month
   <dbl>    <mth>
 1  38.9 1982 Jul
 2  39   1982 Aug
 3  38.9 1982 Sep
 4  39   1982 Oct
 5  39.3 1982 Nov
 6  39.7 1982 Dec
 7  39.2 1983 Jan
 8  38.8 1983 Feb
 9  39.6 1983 Mar
10  39.8 1983 Apr
# i 50 more rows
```

```r
(auto_corr <- ACF(hours_ts, y = hours)) #the outside () make the output appear directly
```

```
# A tsibble: 16 x 2 [1M]
        lag      acf
   <cf_lag>    <dbl>
 1       1M  0.612
 2       2M  0.612
 3       3M  0.439
 4       4M  0.388
 5       5M  0.427
 6       6M  0.214
 7       7M  0.104
 8       8M  0.114
 9        9M -0.0320
10       10M -0.00960
11       11M  0.0932
12       12M -0.0712
13       13M -0.135
14       14M -0.0864
15       15M -0.169
16       16M -0.118
```

```
autoplot(auto_corr, y = hours)
```



1.5 Data Transformations

Transformations refer to the use of mathematical functions to correct for some undesirable features in the data while retaining as much information as possible. To better understand what this refers to, let's have a look at Exercise 2 from HW2.

First, let's have a look at what the `sample()` function does.

```
id_number <- 254978

set.seed(id_number)
sample(aus_retail$`Series ID`,1)
```

```
[1] "A3349480L"
```

You can notice that `sample()` returns a randomly sampled element of the Series ID column in the aus_retail dataframe. We use set.seed() to make sure that every time we run the function it will return the same series. This is done to preserve code reproducibility since when you send your R script to someone else they will get the same result out of it (i.e., "A3349480L").

```
id_number <- 254978

set.seed(id_number) #set seed for code reproducibility
#(the sample() function depends on pseudo-random generation
#and this ensures we get always the same sample)

#filter the dataset called "aus_retail"
myseries <- aus_retail |>
  filter(`Series ID` == sample(aus_retail$`Series ID`,1))

myseries
```

```
# A tsibble: 441 x 5 [1M]
# Key:       State, Industry [1]
   State     Industry                           `Series ID`    Month Turnover
   <chr>     <chr>                              <chr>          <mth>    <dbl>
 1 Queensland Other recreational goods retailing A3349480L   1982 Apr     11.1
 2 Queensland Other recreational goods retailing A3349480L   1982 May     11.7
 3 Queensland Other recreational goods retailing A3349480L   1982 Jun     11.5
 4 Queensland Other recreational goods retailing A3349480L   1982 Jul     13.1
 5 Queensland Other recreational goods retailing A3349480L   1982 Aug     13
 6 Queensland Other recreational goods retailing A3349480L   1982 Sep     13
 7 Queensland Other recreational goods retailing A3349480L   1982 Oct     12
 8 Queensland Other recreational goods retailing A3349480L   1982 Nov     13.2
 9 Queensland Other recreational goods retailing A3349480L   1982 Dec     16.2
10 Queensland Other recreational goods retailing A3349480L   1983 Jan     12
# i 431 more rows
```

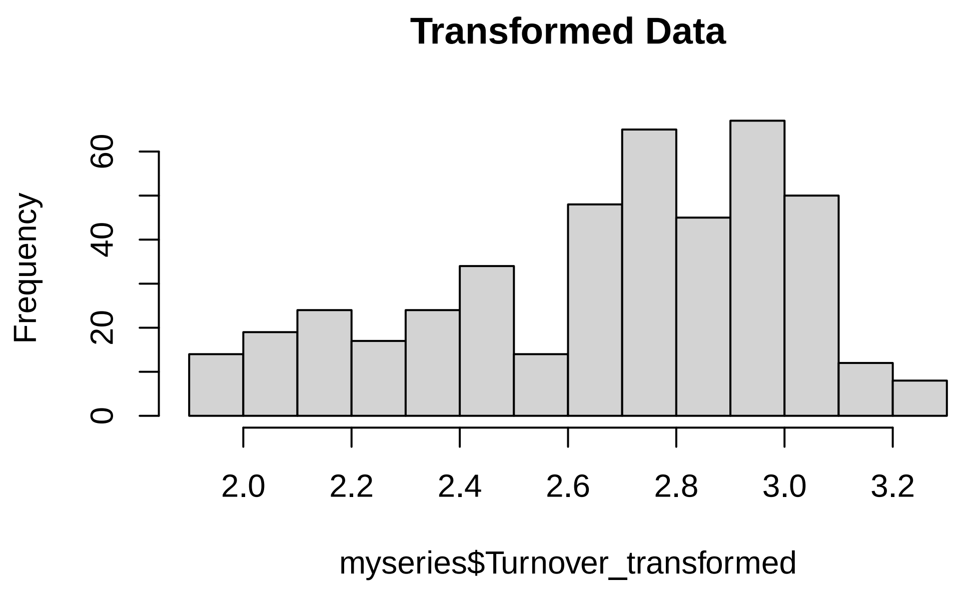
Knowing that sample() returns "A3349480L" helps you also understand what is going on in the cell above. In short, the aus_retail dataframe is being filtered and only the rows which have Series ID equal to "A3349480L" are kept and assigned to the variable *`myseries`* (meaning we can later refer to this filtered dataframe by the name *`myseries`*).

Now we move on and explore Box-Cox transformations, which are transformations used to normalize data that do not appear to be normally distributed. These transformations are also used to make "the size of the seasonal variation about the same across the whole series" as pointed out in Chapter 3.1 of your textbook. Therefore, they tend to alleviate the problems associated to seasonality in time series and exploit these patterns to derive more information which are useful for time series modeling! You can spot deviations from normality in different ways, either through data visualizations or through specific tests (don't worry for now you don't need to know about this stuff, just focus on visualizations)

```r
guer <- features(myseries, Turnover, features = guerrero)
#use guerrero function to determine a suitable lambda for the box-cox transformation
lambda <- guer$lambda_guerrero
#assign the computed lambda to a variable with the same name (lambda)
myseries$Turnover_transformed <- box_cox(myseries$Turnover, lambda = lambda)
#add the transformed turnover column to the original dataset
#and assign the name "Turnover_transformed"

myseries
```

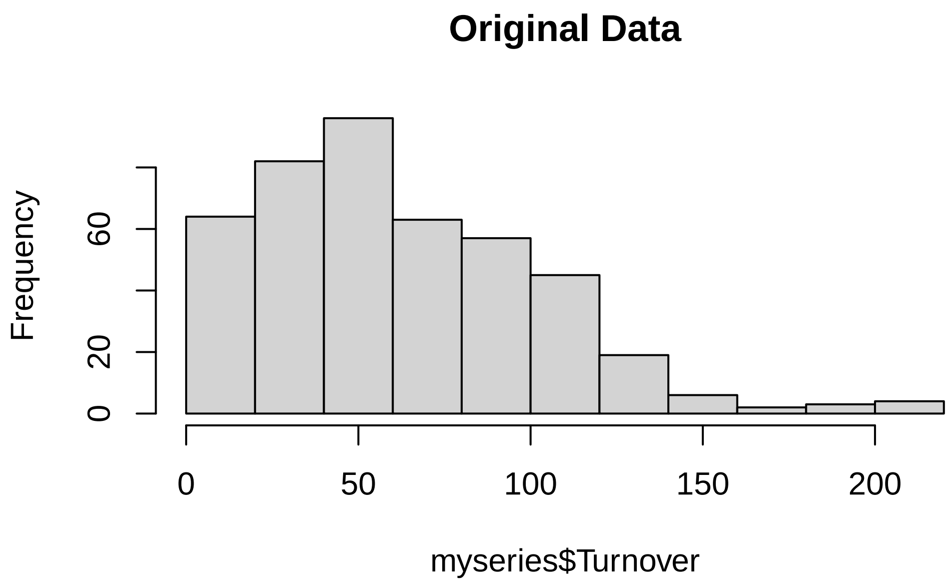
```
# A tsibble: 441 x 6 [1M]
# Key:       State, Industry [1]
   State      Industry         `Series ID`    Month Turnover Turnover_transformed
   <chr>      <chr>            <chr>          <mth>    <dbl>                <dbl>
 1 Queensland Other recreati~ A3349480L   1982 Apr     11.1                 1.90
 2 Queensland Other recreati~ A3349480L   1982 May     11.7                 1.94
 3 Queensland Other recreati~ A3349480L   1982 Jun     11.5                 1.93
 4 Queensland Other recreati~ A3349480L   1982 Jul     13.1                 2.00
 5 Queensland Other recreati~ A3349480L   1982 Aug     13                   2.00
 6 Queensland Other recreati~ A3349480L   1982 Sep     13                   2.00
 7 Queensland Other recreati~ A3349480L   1982 Oct     12                   1.95
 8 Queensland Other recreati~ A3349480L   1982 Nov     13.2                 2.01
 9 Queensland Other recreati~ A3349480L   1982 Dec     16.2                 2.13
10 Queensland Other recreati~ A3349480L   1983 Jan     12                   1.95
# i 431 more rows
```

```r
hist(myseries$Turnover_transformed, main = "Transformed Data")
```



```
#visualize the improvements in normality with this plot...
```

```
hist(myseries$Turnover, main = "Original Data")
```

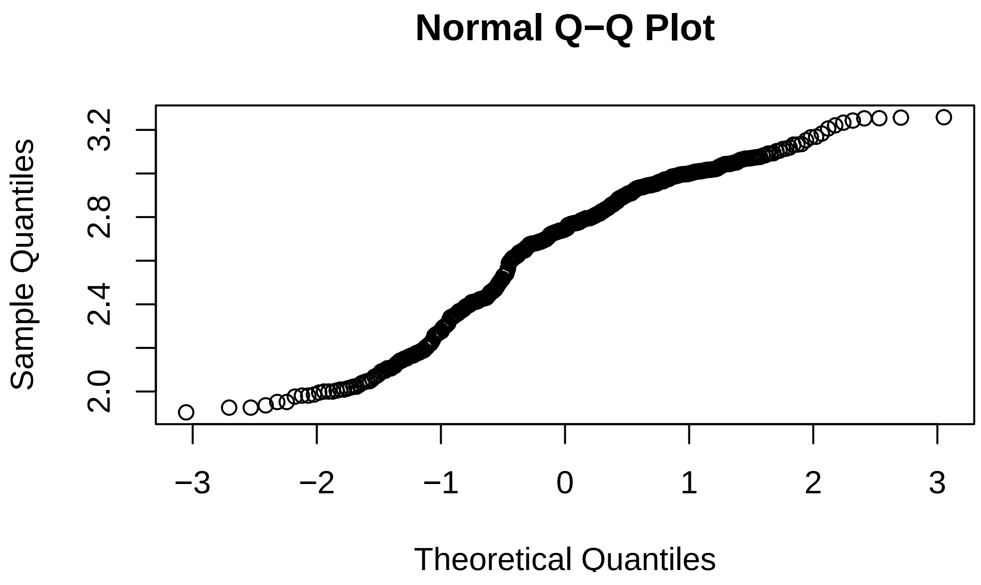


```
#... compared to this more skewed distribution
```

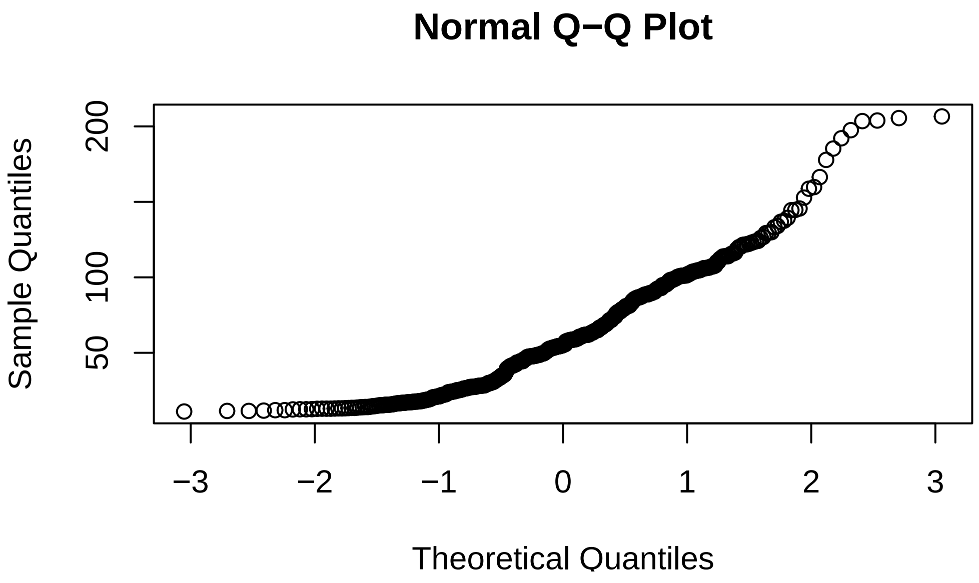
Also, you can use a quantile-quantile plot which shows how much of the data falls within specified quantiles (represented on the x-axis). Normality here is represented by points that are closer to the diagonal line.

```
qqnorm(y = myseries$Turnover_transformed)                                    (1)
```

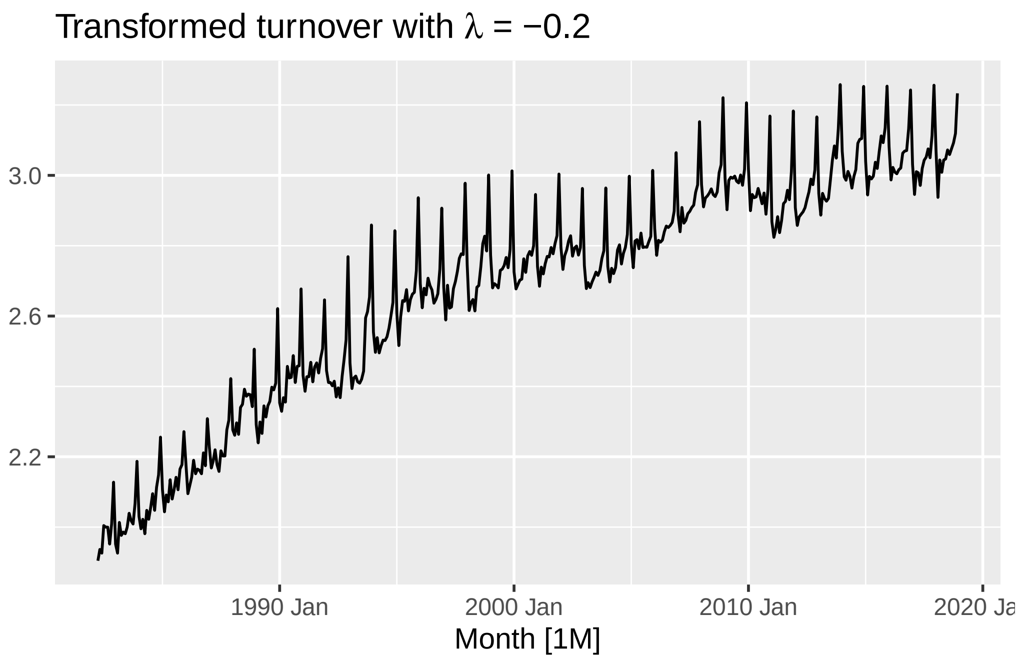
(1) quantile-quantile plot to show the normality of the data (closer to the diagonal is better)



```r
qqnorm(y = myseries$Turnover)
```



```r
autoplot(myseries, box_cox(Turnover, lambda)) +
  labs(y = "",
       title =TeX(paste0("Transformed turnover with $\\lambda$ = ",
                          round(lambda,2))))
```



it is concluded that a box-cox transformation with lambda ≈ -0.20 improves the fit to normality of the data as shown by the quantile-quantile plot and the histogram.

1.6 Data visualization

We just scratched the surface of what data visualization can do, so in Exercise 3 of HW2 we delve a little deeper into that.

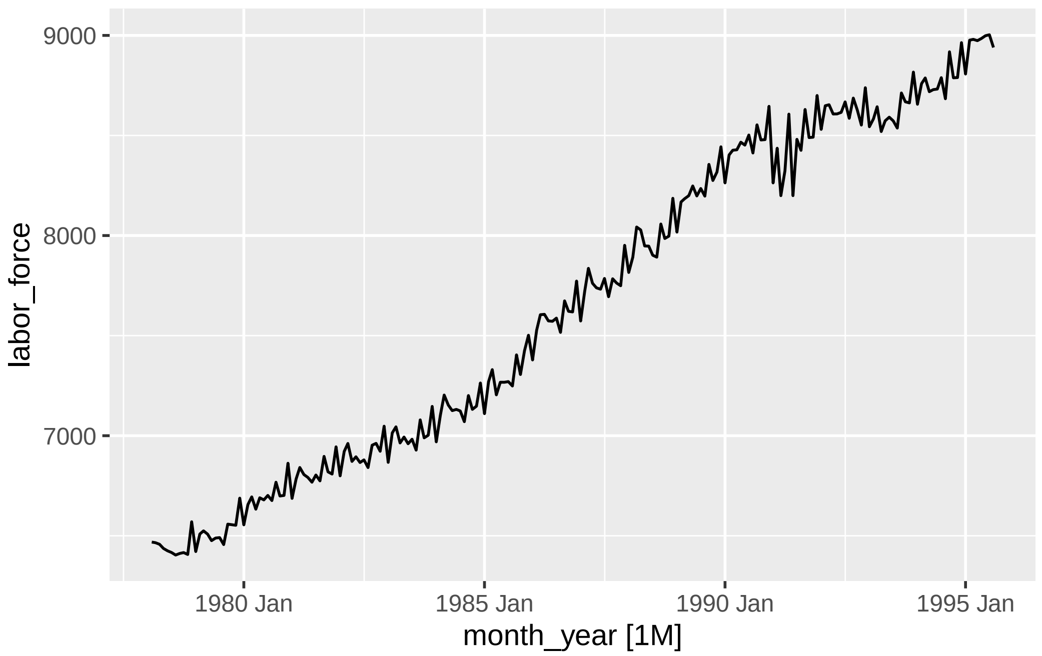
```r
labour <- fma::labour

dates <- seq.Date(from = as.Date("1978-02-01"), to = as.Date("1995-08-01"), by = "month")

labour <- as.data.frame(labour)
labour$date <- dates
names(labour) <- c("labor_force", "month_year")

labour <- labour %>%
  mutate(month_year = yearmonth(month_year))

(labour_ts <- as_tsibble(labour, index = month_year))
```

```
# A tsibble: 211 x 2 [1M]
   labor_force month_year
         <dbl>      <mth>
 1       6468.   1978 Feb
 2       6465.   1978 Mar
 3       6458.   1978 Apr
 4       6437.   1978 May
 5       6425.   1978 Jun
 6       6417.   1978 Jul
 7       6404.   1978 Aug
 8       6412.   1978 Sep
 9       6416.   1978 Oct
10       6407.   1978 Nov
# i 201 more rows
```

As usual, we import the dataset and perform the routine steps to organize it into a nice tsibble complete with a column for dates. Now, we are ready to produce some plots to explore some features of this time series.

1.6.1 Time Plot

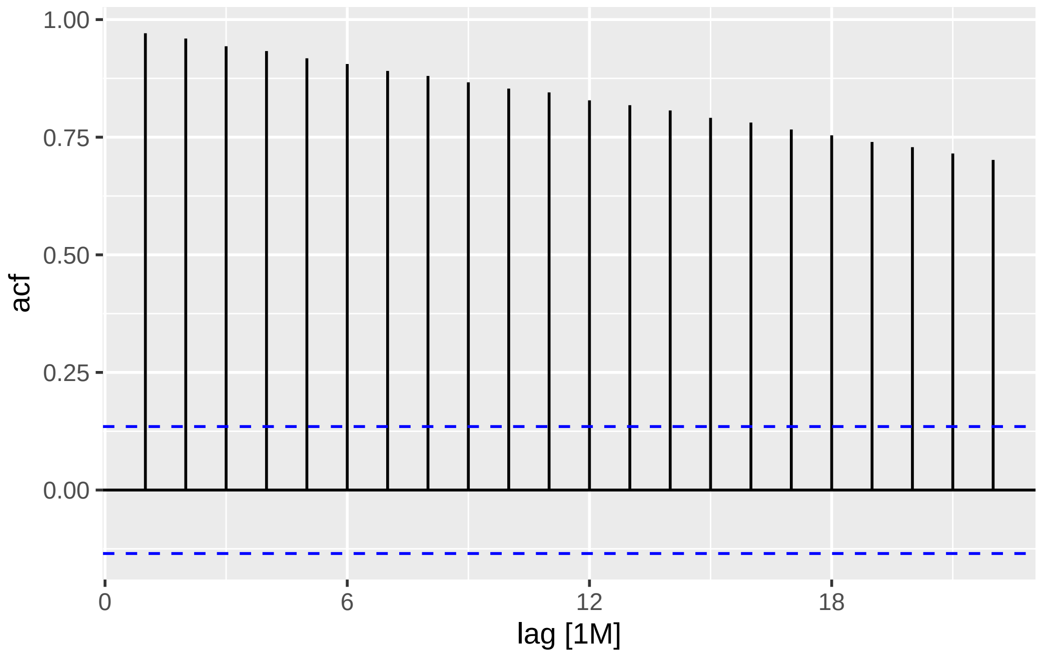
```r
autoplot(labour_ts, .vars = labor_force) #time plot
```



You can plot a time plot very easily with `autoplot()`. This allows you to see trends in data.

1.6.2 Correlogram

```r
## correlogram
autocorr <- ACF(labour_ts, labor_force)
autoplot(autocorr) #huge autocorrelation across months
```



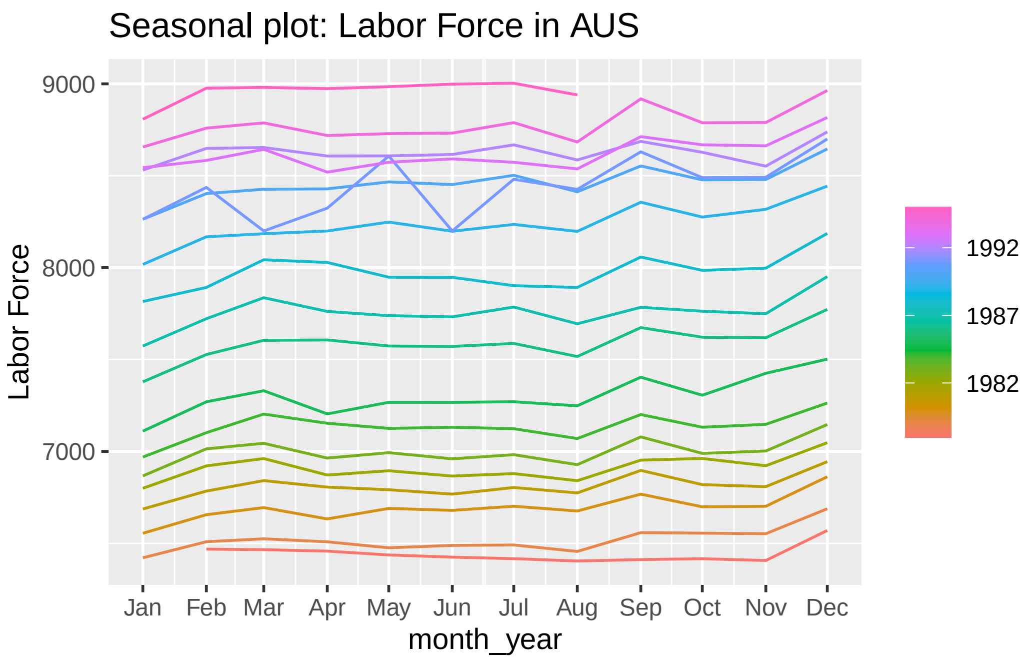
To compute a correlogram you follow these steps:

1. you estimate the auto correlation function with `ACF(dataset, name_of_column_in_dataset)` and assign it to a variable (autocorr in my example above)
2. then you pass that variable as an argument in `autoplot()` and that will do the trick.

1.6.3 Seasonality Plots

They allow you to check for the presence of seasonality (trends based on time periods) in your data.

```
## seasonality plot
gg_season(labour_ts, labor_force) +
  labs(y = "Labor Force",
       title = "Seasonal plot: Labor Force in AUS")
```

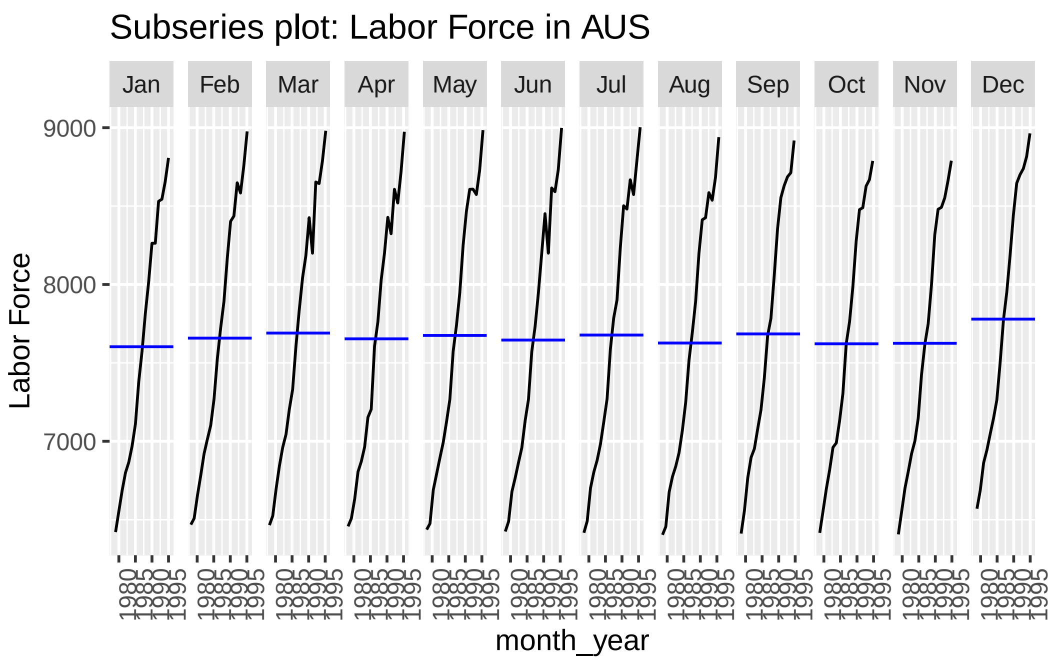


For these plots you use `gg_season()` and specify the dataset and the column to plot. You can add (literally with a + sign) other layers to your plot. In my example I added y labels and a title. If you learn this syntax you can be sure that you can use it on any other ggplot you'll ever do in your life.

From this seasonal plot we learn that every year around December more people join the labor force.

1.6.4 Subseries Plots

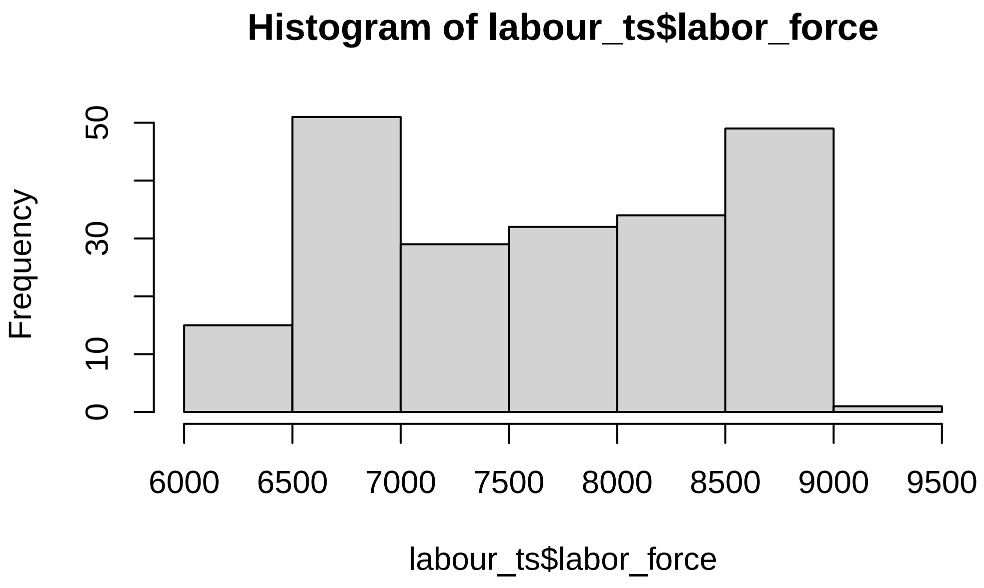
```
gg_subseries(labour_ts, labor_force) +
  labs(y = "Labor Force",
       title = "Subseries plot: Labor Force in AUS")
```



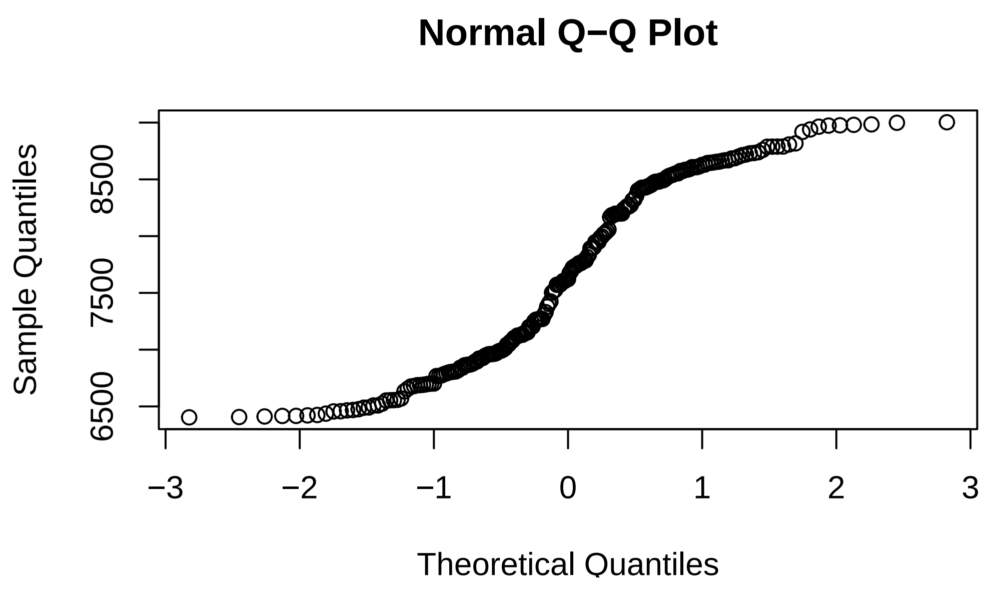
In these plots we can see how the series evolves across months in different years. The syntax is the same as the one for seasonality plots, except for the function `gg_subseries()` of course.

```
hist(labour_ts$labor_force)                                                      (1)
```

(1) the distribution appears to be bimodal, therefore a smoothing through box-cox would make sense



```
qqnorm(y = labour_ts$labor_force)
```

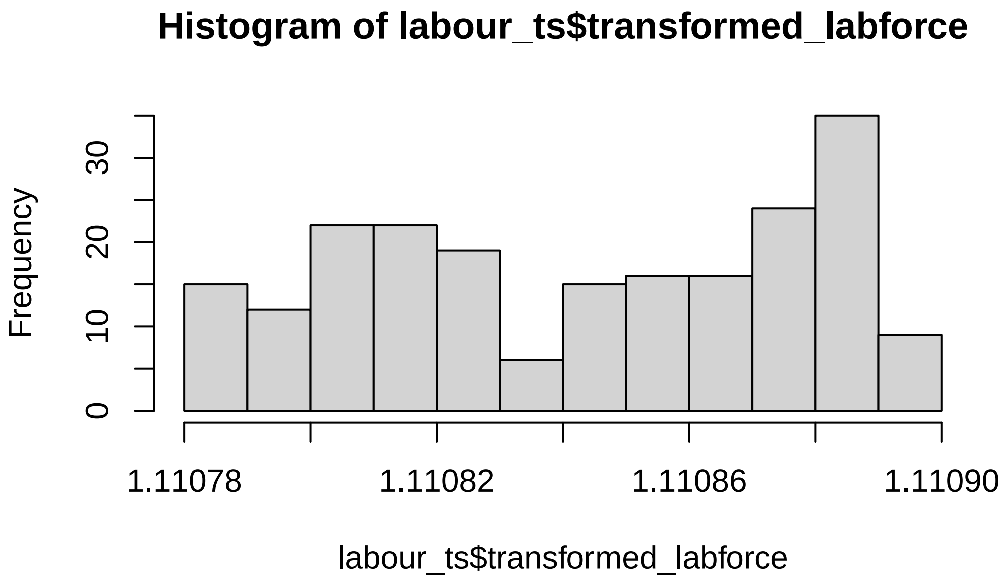


```r
lambda <- features(labour_ts, .var = labor_force, features = guerrero)$lambda_guerrero ①
transformed_labforce <- box_cox(labour_ts$labor_force, lambda)                       ②

labour_ts$transformed_labforce <- transformed_labforce                                 ③
labour_ts
```

① estimate the value of lambda using the Guerrero method through the `features()` extraction
② perform the `box_cox()` transformation on the `labor_force` column of the data using the `lambda` estimated at point 1
③ create a new column in `labour_ts` called `transformed_labforce` which contains the values of the box_cox transformation

```
# A tsibble: 211 x 3 [1M]
   labor_force month_year transformed_labforce
         <dbl>      <mth>                <dbl>
 1       6468.   1978 Feb                 1.11
 2       6465.   1978 Mar                 1.11
 3       6458.   1978 Apr                 1.11
 4       6437.   1978 May                 1.11
 5       6425.   1978 Jun                 1.11
 6       6417.   1978 Jul                 1.11
 7       6404.   1978 Aug                 1.11
 8       6412.   1978 Sep                 1.11
 9       6416.   1978 Oct                 1.11
10       6407.   1978 Nov                 1.11
# i 201 more rows
```

```r
hist(labour_ts$transformed_labforce)
```



```
#however the distribution remains highly bimodal and normality
#does not seem to be improved by much
```

For the last step of the exercise we compute again the Box-Cox transformation for this time series, but it does not do much to improve upon the normality of this specific time series.