FORECASTING AND PREDICTIVE MODELING

LECTURE 3

Lect. PhD. Oneț-Marian Zsuzsanna

Babeș - Bolyai University
Computer Science and Mathematics Faculty

2024 - 2025

In Lecture 2...

- Working in R
  - Data representations
  - Working with data
  - Plotting

Today

- Time series visualizations

Time series visualizations

- The first thing to do when working with a time series is to plot it. We can use the function *autoplot* for this.

```
olympic_running |> autoplot()

olympic_running |>
  filter(Length == 100 & Sex == "men") |>
  autoplot()
```

Time series plot

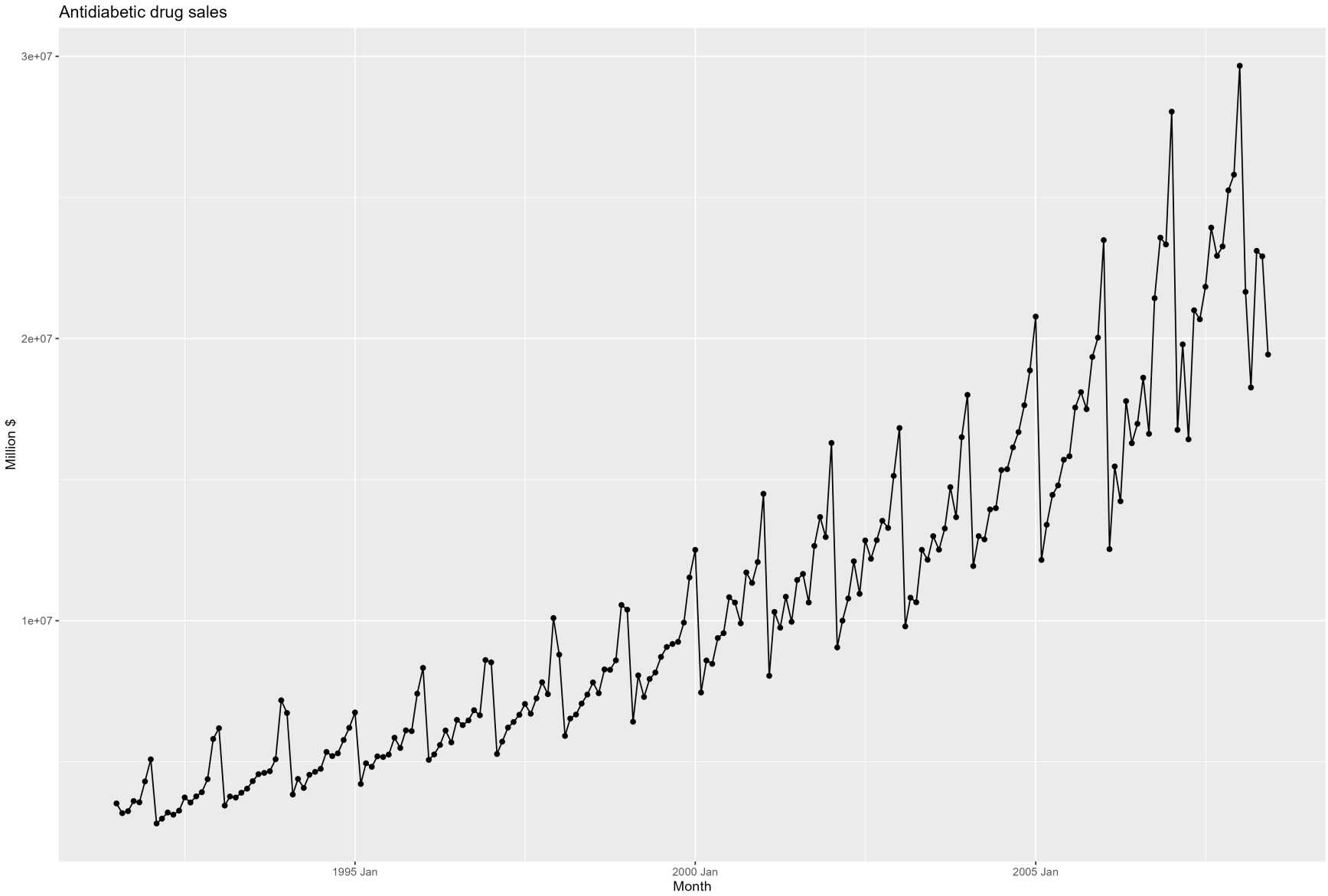
- Let us start with a data set called *PBS*, containing monthly drug prescription data from Australia from 1991 July to 2008 June. The drugs are divided into categories. We will only use a part of this data

```
antidiabetic <- PBS |>
  filter(ATC2 == "A10") |>
  select(Month, Concession, Type, Cost) |>
  summarize(TotalCost = sum(Cost)) |>
  mutate(Cost = TotalCost / 1000000)

antidiabetic |> autoplot(TotalCost) +
  labs(x = "Month", y = "Million $", title = "Antidiabetic drug sales")
```

- Similarly, we can create the following tsibbles:
  - A11 - Vitamins
  - A12 - Mineral supplements
  - J07 - Vaccines
  - N02 - Analgesics

Time series plot

- Sometimes, it is hard to identify the exact values on the time series plot. We can also solve this problem by showing the actual points, adding a *geom_point* layer to the plot.


Antidiabetic drug sales
3e+07
2e+07
1e+07
Million $
1995 Jan
2000 Jan
2005 Jan
Month

Trend, season, cycle

- When visually analyzing a time series plot, there are three important terms that can be used to describe it.
  - **trend** - describes a long term increase or decrease in the data, even if it is not linear.
  - **seasonal pattern** - appears when the data values are affected by factors related to the calendar like time of the year or day of the week. Seasonality always has a fixed known frequency, which can be at most a year.
  - **cycle** - when there are rises and falls, but not of fixed frequency, caused in general by economic conditions. Their duration is usually at least 2 years.

- There is a fourth term that can be used to characterize time series data: **stationarity**.

- A time series is stationary if its statistical properties (mean, standard deviation) do not depend on the time at which the series is observed. This means that the time series cannot have trend or seasonality.

- Whether a time series is stationary or not, can be inspected visually, but there are also statistical tests that can be used to check this (these will be discussed later).

- In case of seasonal data, determining the length of a season is very important (it is a value which is used later in analysis).
- Another important thing to determine in case of seasonal data is whether the magnitude is constant over time (we call this additive effect) or whether it grows or shrinks (multiplicative effect).
- When you have many points on your plot (we have 204), determining the exact length might not be easy. In such cases we can *zoom in* on the data, by plotting only parts of it. We can use the *head*, *tail* or *slice* functions.

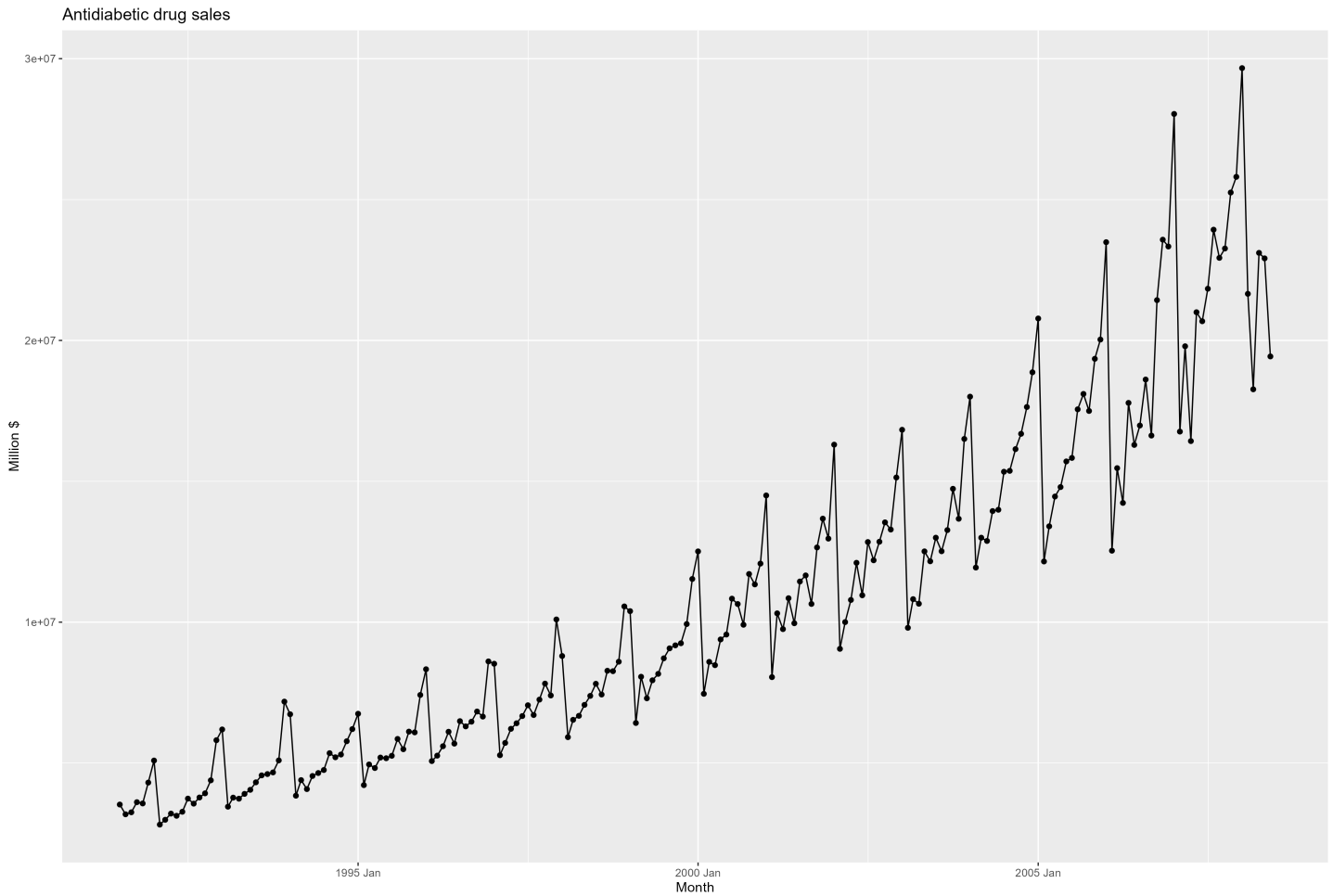
```
A10 |>
  head(n = 35) |>
  autoplot() +
  geom_point()


A10 |>
   slice(10:40) |>
   autoplot()
```

Trend, season, cycle examples

- Let us see a few examples of time series from the PBS data.

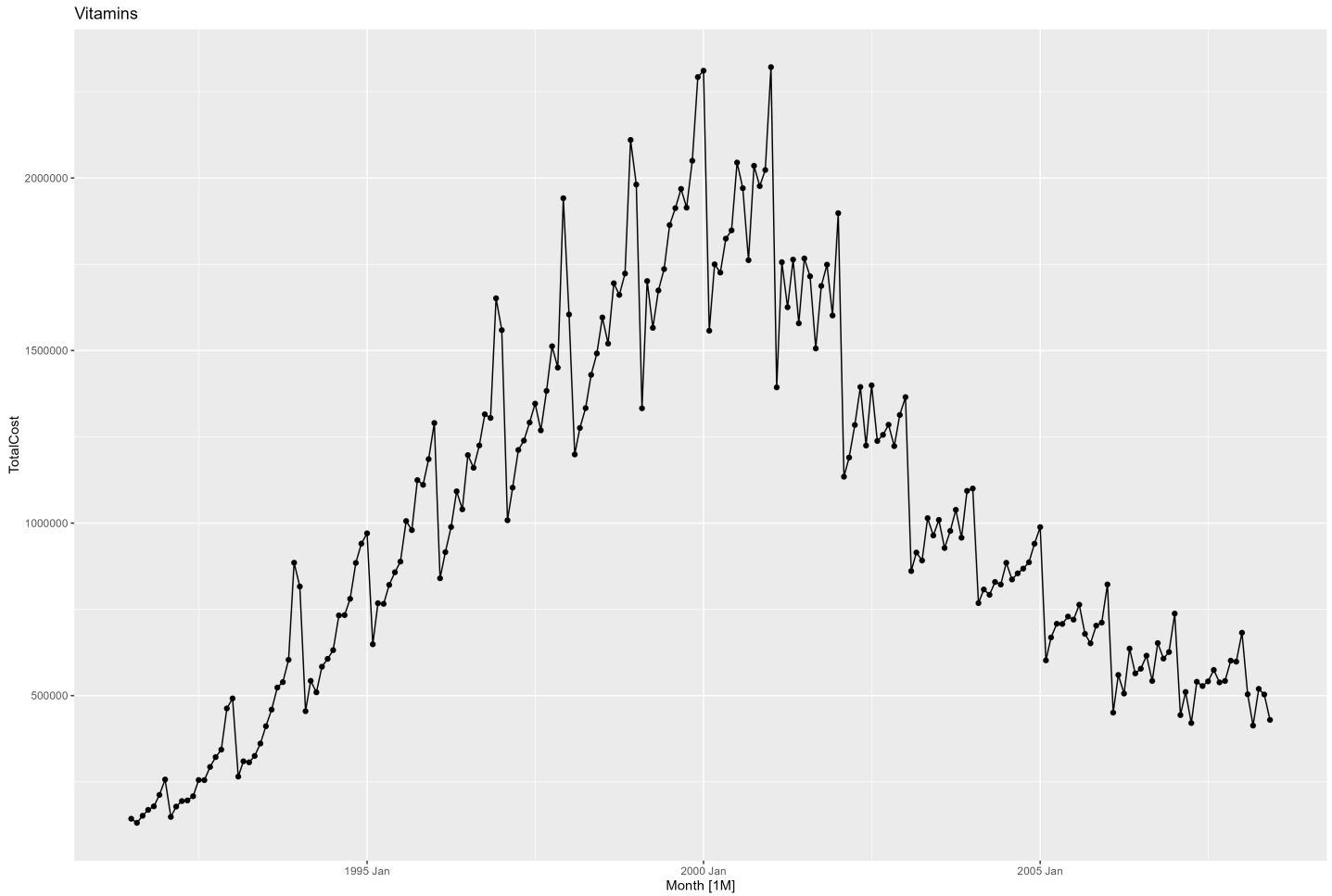
```
A10 |> autoplot(TotalCost) +
  geom_point() +
  labs(title = "Antidiabetic drugs")

A11 |> autoplot(TotalCost) +
  geom_point() +
  labs(title = "Vitamins")

A12 |> autoplot(TotalCost) +
  geom_point() +
  labs(title = "Mineral supplements")

N02|> autoplot(TotalCost) +
  geom_point() +
  labs(title = "Analgesics")
```

Trend, season, cycle example

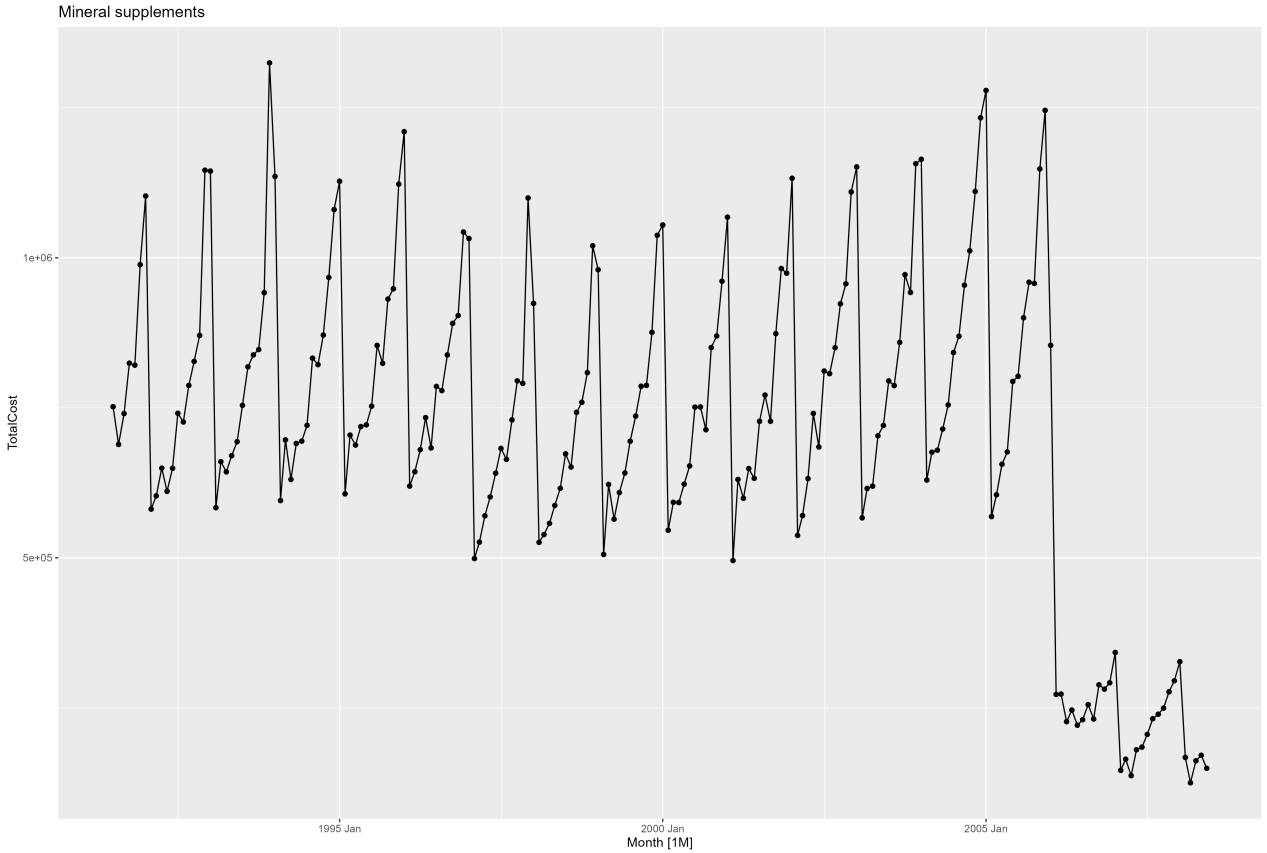


- Increasing trend, multiplicative seasonality

Trend, season, cycle example

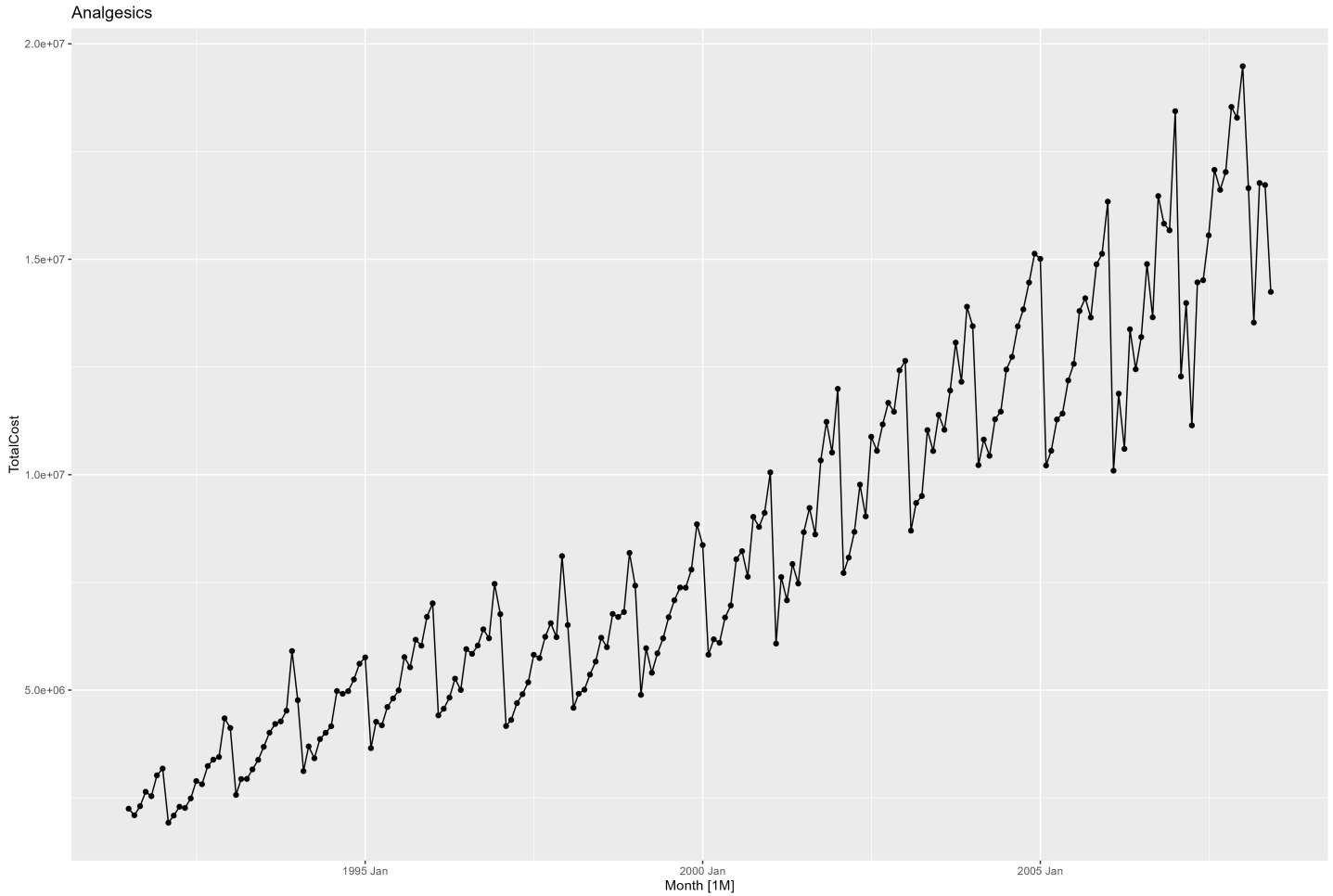


- Increasing trend until 2000, then a decreasing trend.
- Clear seasonality (not easy to determine if it is additive or multiplicative).



- Sudden drop in values in 2006 February.
- Until 2006 February there does not seem to be a trend (or only a very small one) and there is not enough data after 2006 to determine if there is a trend.
- There is seasonality, it seems to be additive (but not enough data to clearly see it after 2006)

Trend, season, cycle example

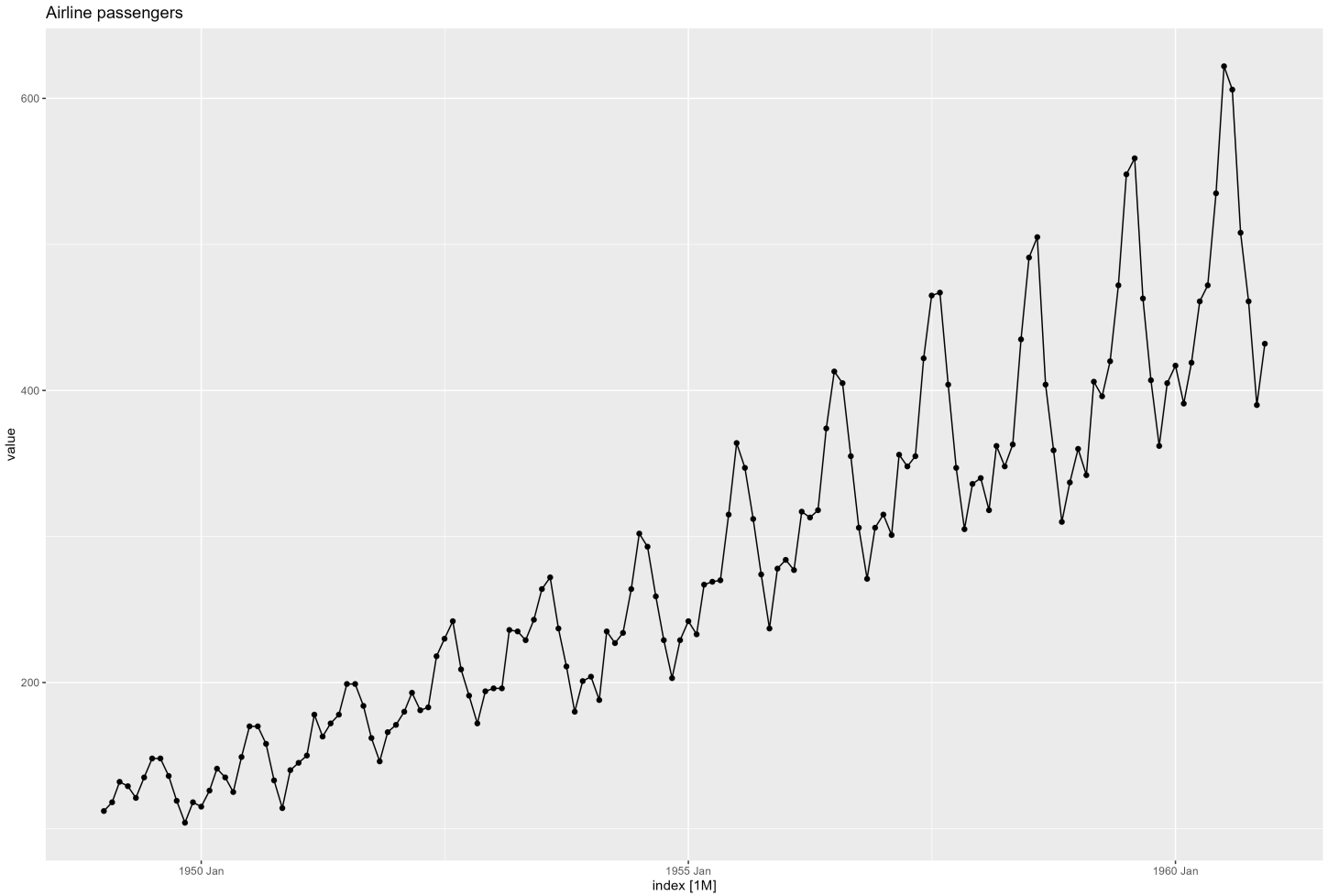


- Clearly increasing trend and multiplicative seasonality (very similar to antidiabetic drugs)

Trend, season, cycle example

- Monthly total data about airline passengers from 1949 to 1960, in thousands (data from datasets package in R)

```
AirPassengers |>
  as_tsibble() |>
  autoplot() +
  geom_point() +
  labs(title = "Airline passengers")
```

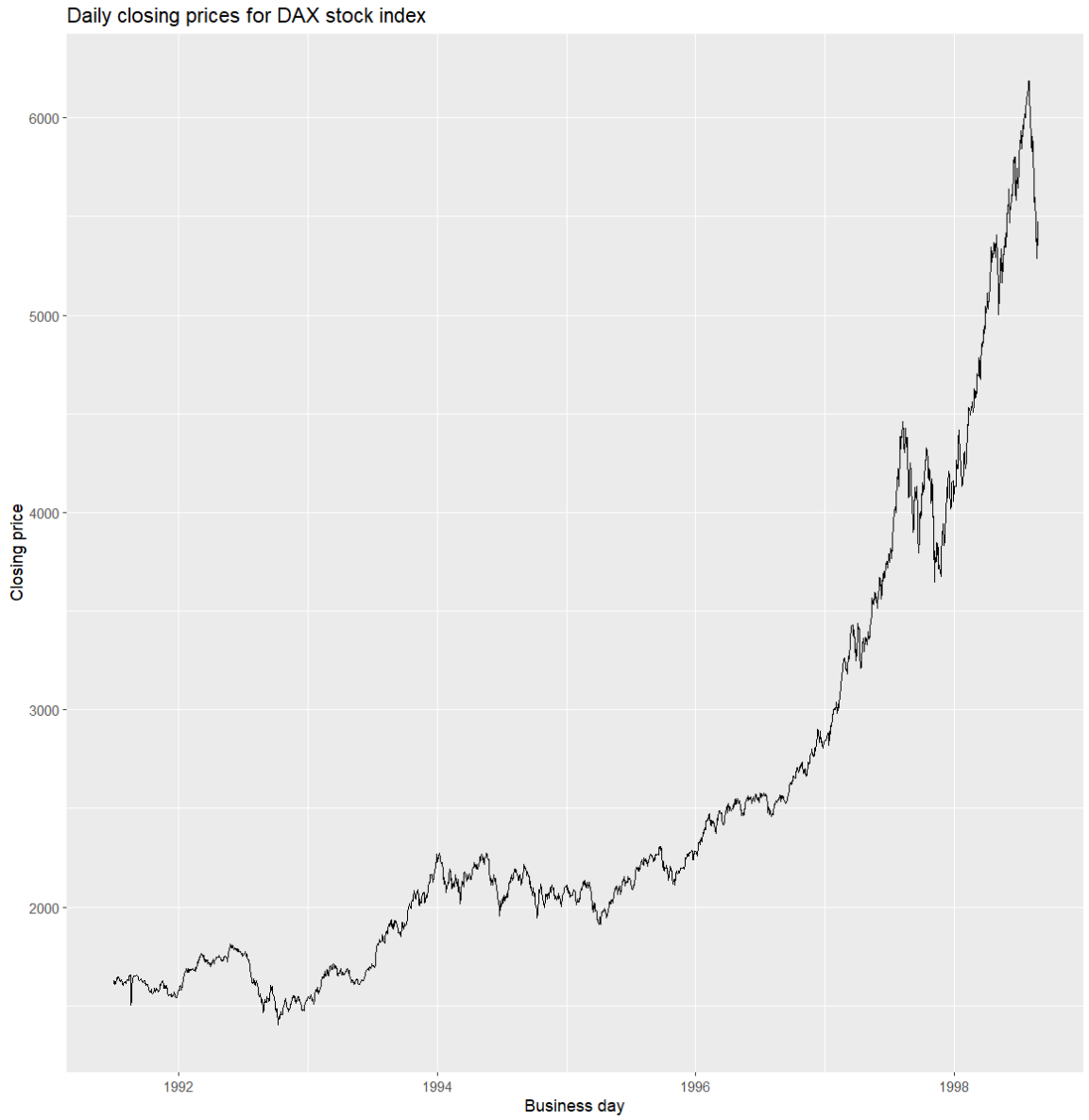


- We have an increasing trend and multiplicative seasonality with a frequency of a year

Trend, season, cycle example

- Daily closing prices of the DAX stock index between 1991 - 1998 (taken from the EuStockMarkets data set from R)

```
EuStockMarkets |>
    as_tsibble() |>
    filter(key == "DAX") |>
    autoplot(value) +
    labs(x = "Business day", y = "Closing price",
     title = "Daily closing prices for DAX stock index")
```

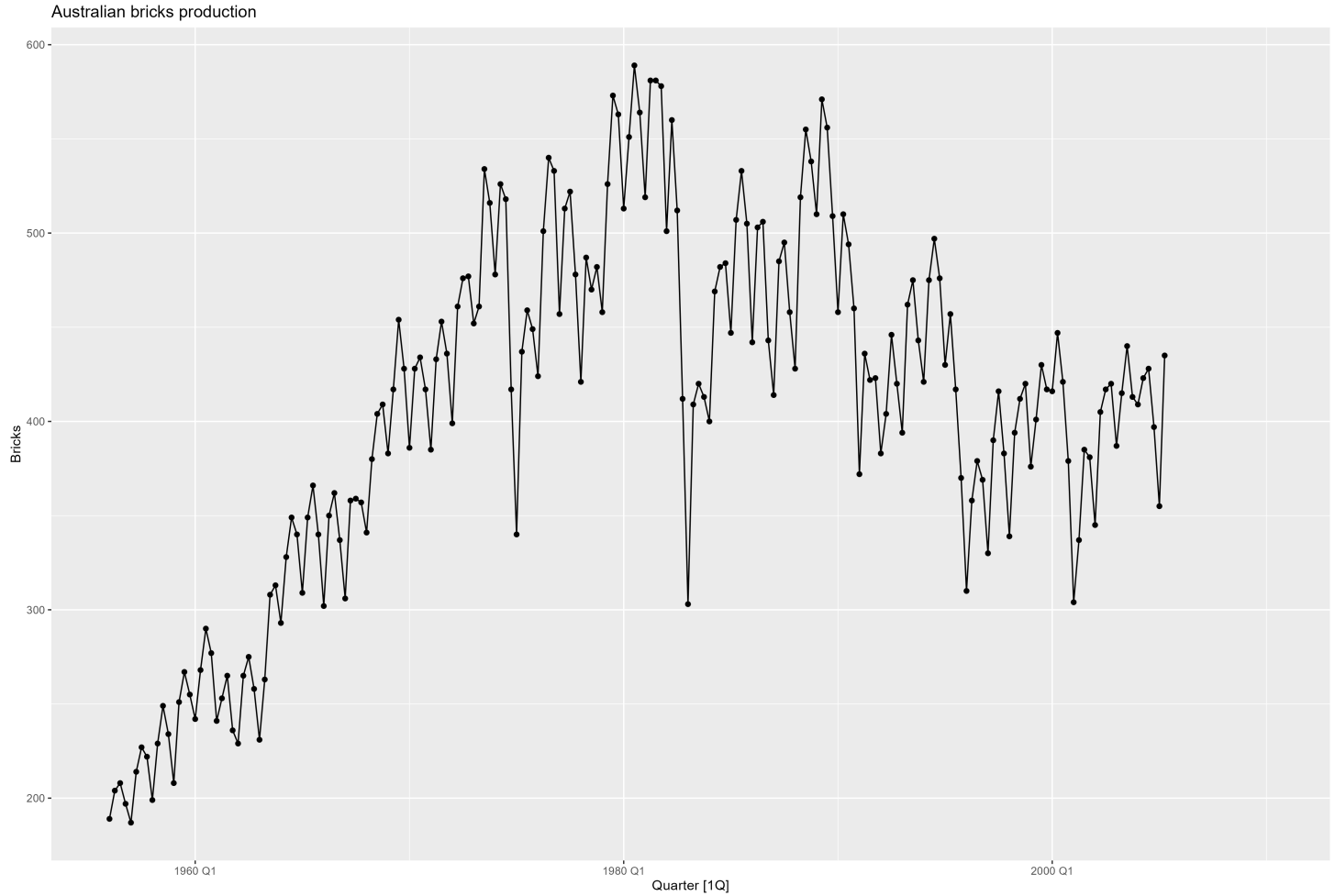


- We have an increasing trend and no seasonality or cycle.

Trend, season, cycle examples

- Data about quarterly clay brick production between 1956 - 1994 (taken from the *fma* package in R)

```
aus_bricks <- aus_production |>
  select(Bricks)

aus_bricks |>
  autoplot() +
  geom_point() +
  labs(title = "Australian bricks production")
```

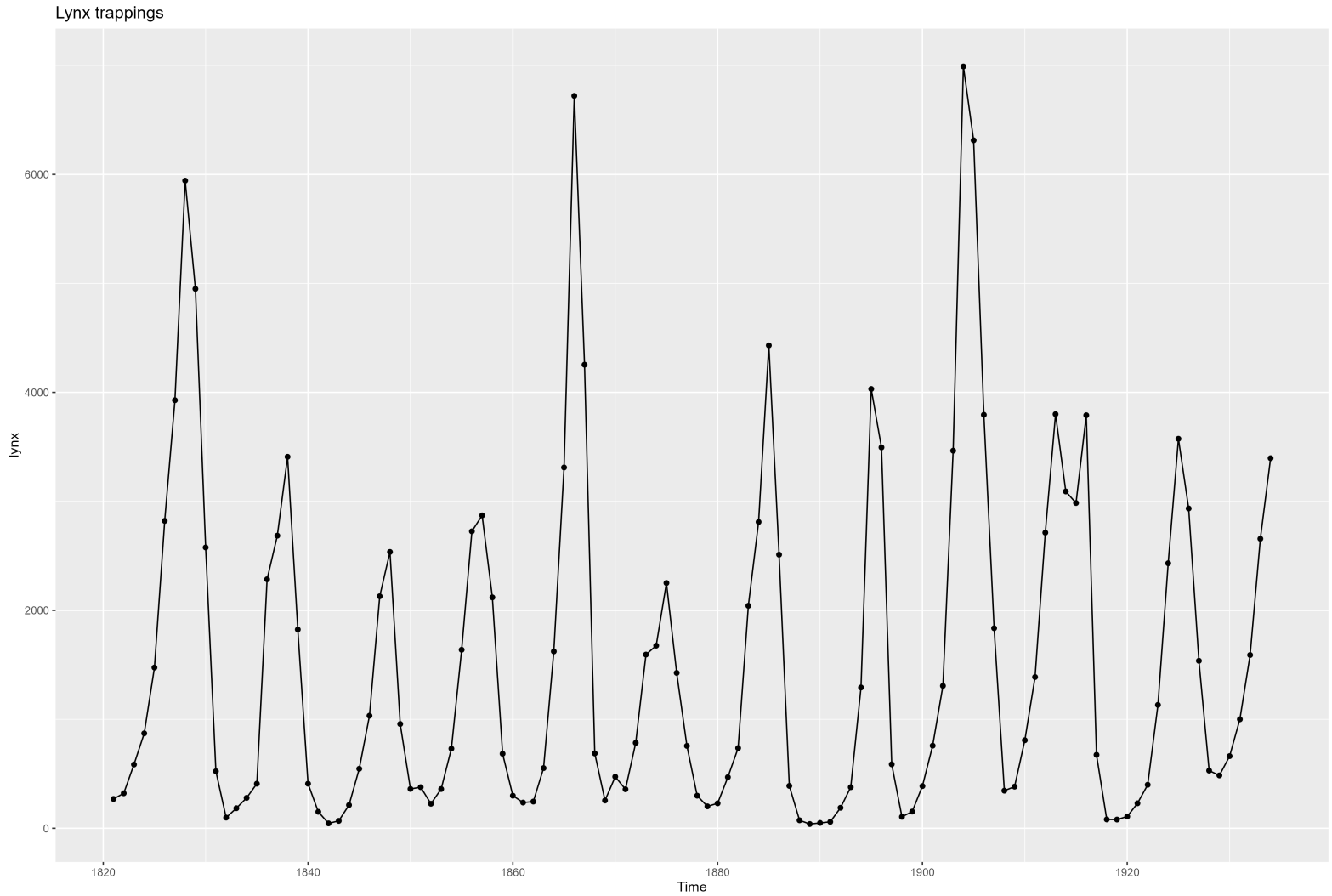


- We have a seasonality with a frequency of a year, but also cycles, showing periods of recession (1975, 1983, 1991) between which the series increases and decreases.

Trend, season, cycle examples

- Data about annual number of lynx trapped in McKenzie river district of northwest Canada between 1821-1934

```
lynx |>
  autoplot() +
  geom_point() +
  labs(title = "Lynx trappings")
```

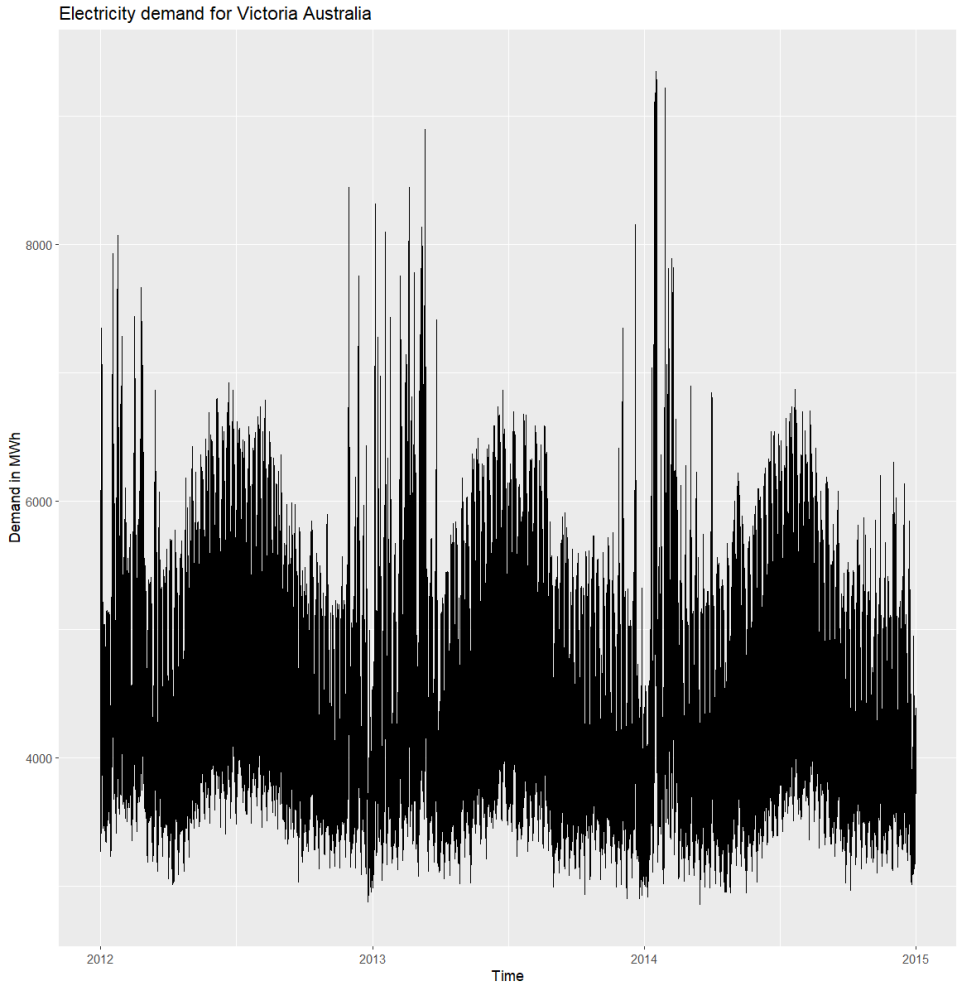


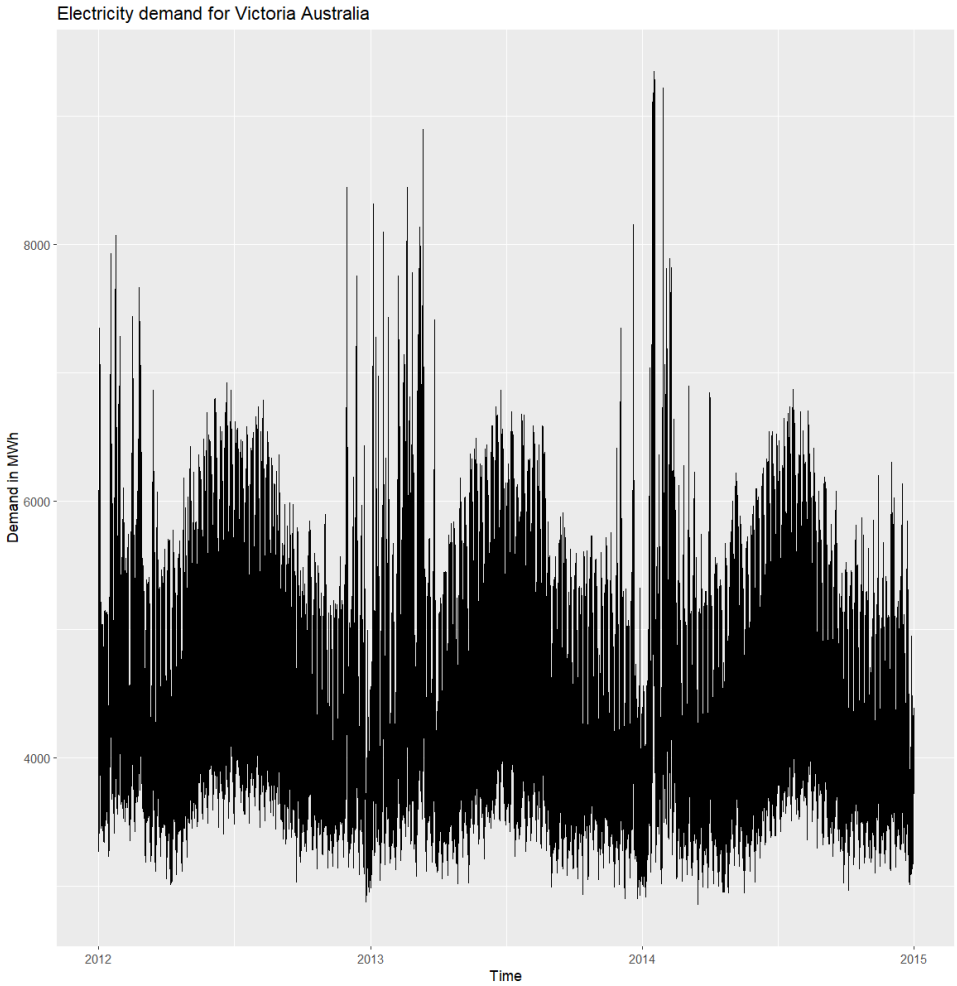
- It might look like seasonal data, but since it is annual, it cannot be seasonal. The length of a "pattern" is several years, so we actually have cycles of 8-11 years.

Trend, season, cycle examples

- Half-hourly electricity demand for Victoria, Australia, between 2012-01-01 and 2014-12-31 (52608 observations). Contains information about the electricity demand and the temperature of Melbourne. (taken from package *tsibbledata* in R).

```
vic_elec |>
  autoplot(Demand) +
  labs(title = "Electricity demand")
```


Electricity demand for Victoria Australia
Demand in MWh
8000
6000
4000
2012
2013
2014
2015
Time

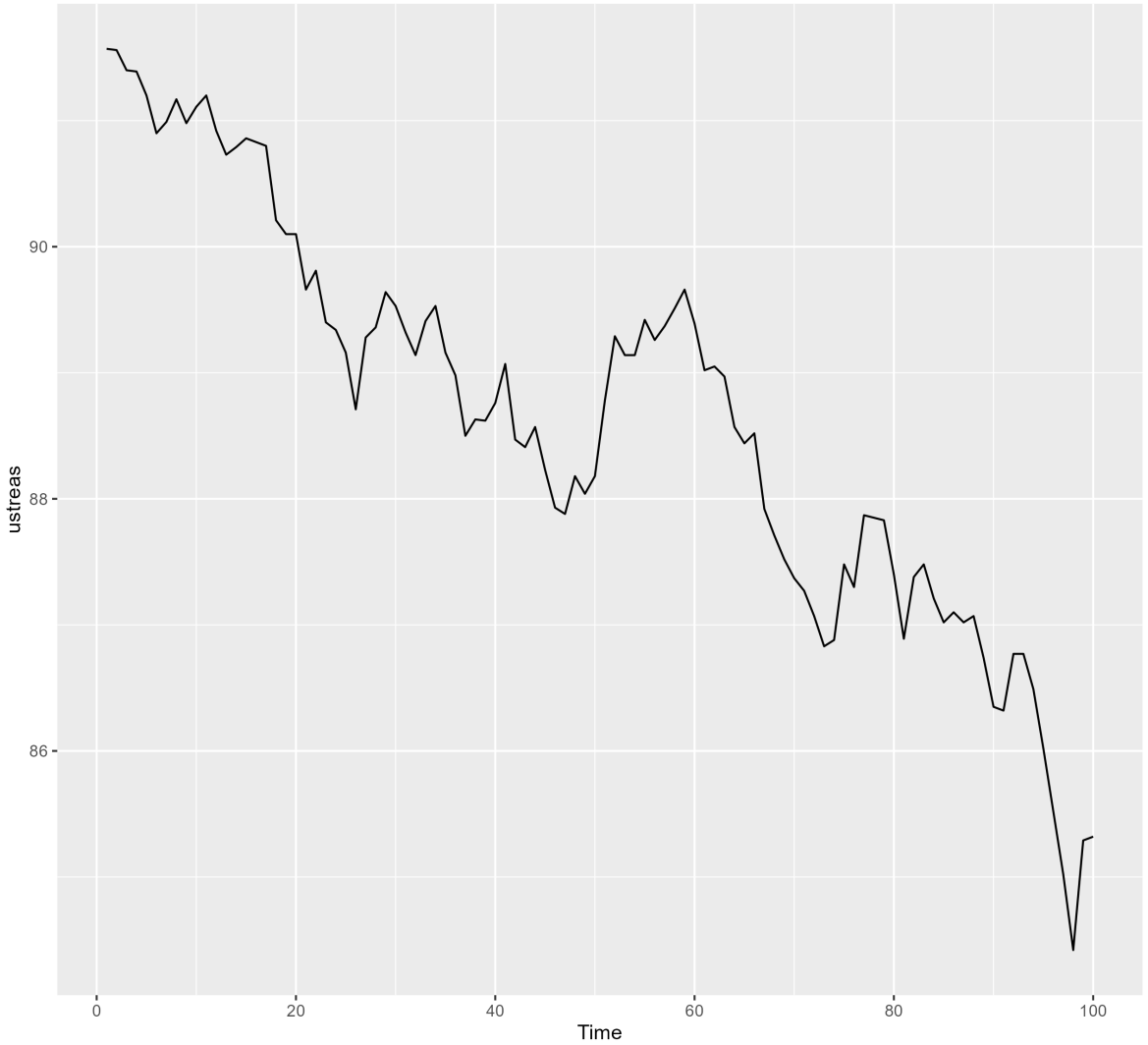


- There is no trend, but it looks like we have yearly seasons. If we take a closer look at the data (for example, the first 3 weeks) we can see that we have a daily season as well. We can have data with two seasons with different length.

Trend, season, cycle examples

- US treasury bill contracts on the Chicago market for 100 consecutive trading days in 1981

```
ustreas |>
  autoplot() +
  geom_point() +
  labs(title = "Us treasury bill contracts")
```



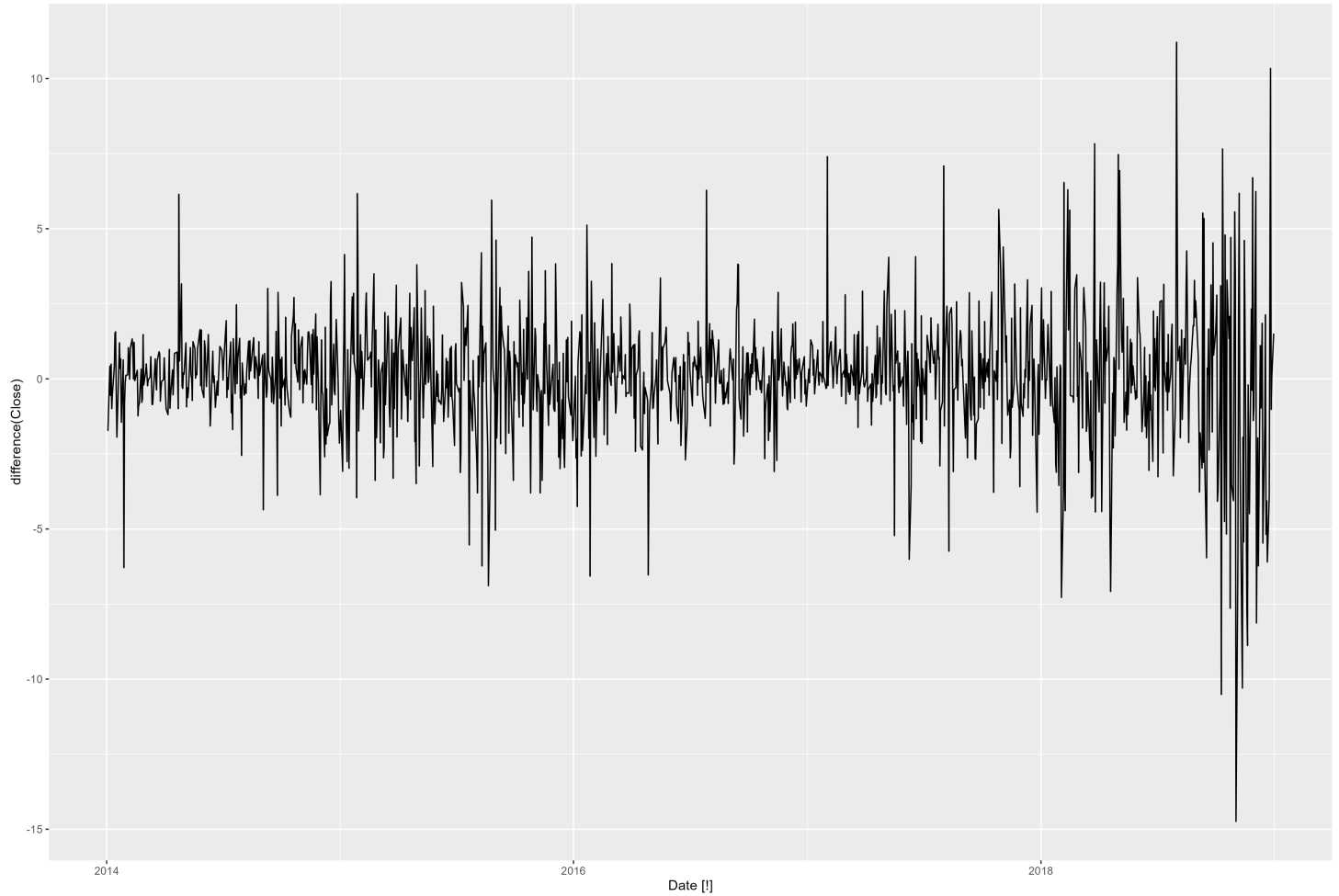
- There is a downward trend, but no season or cycle.

Trans, season, cycle examples

- When working with trading data, sometimes the actual stock price is less important, than the change from the previous day.

- Let us look at the change for Apple stock prices in the 2014-2018 period.

```
apple <- gafa_stock |>
  filter(Symbol == "AAPL")

apple |>
    autoplot(difference(Close))
```

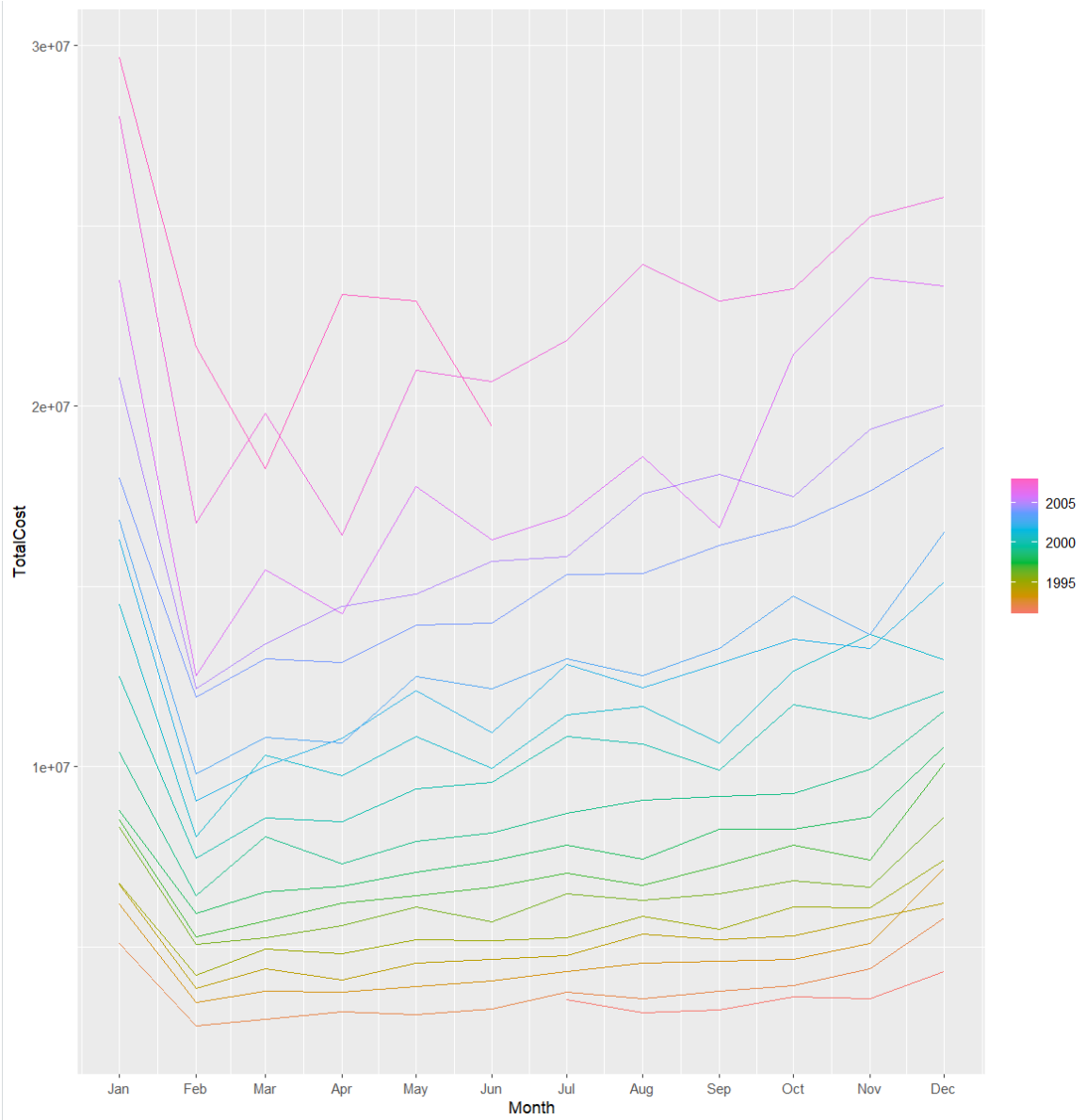


- There is no trend, no cycle, no season. It is stationary.

Seasonal plots

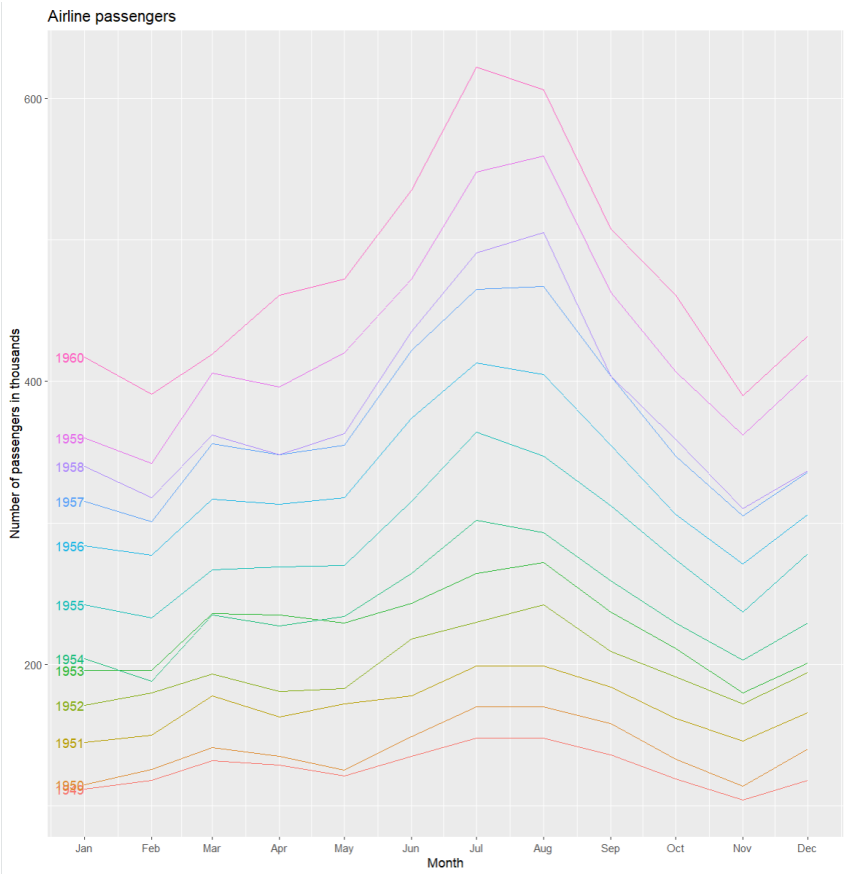
- Once we have identified that there is a seasonality in our time series, we can use a *seasonal* plot to visualize the data.
- A *seasonal* plot simply plots every season *"in parallel"*, so we can see the underlying patterns more clearly and it is easier to detect anomalies or changes in the pattern.
- We can use the *gg_season* for creating seasonal plots.

```
A10 |>
    gg_season(TotalCost)
```

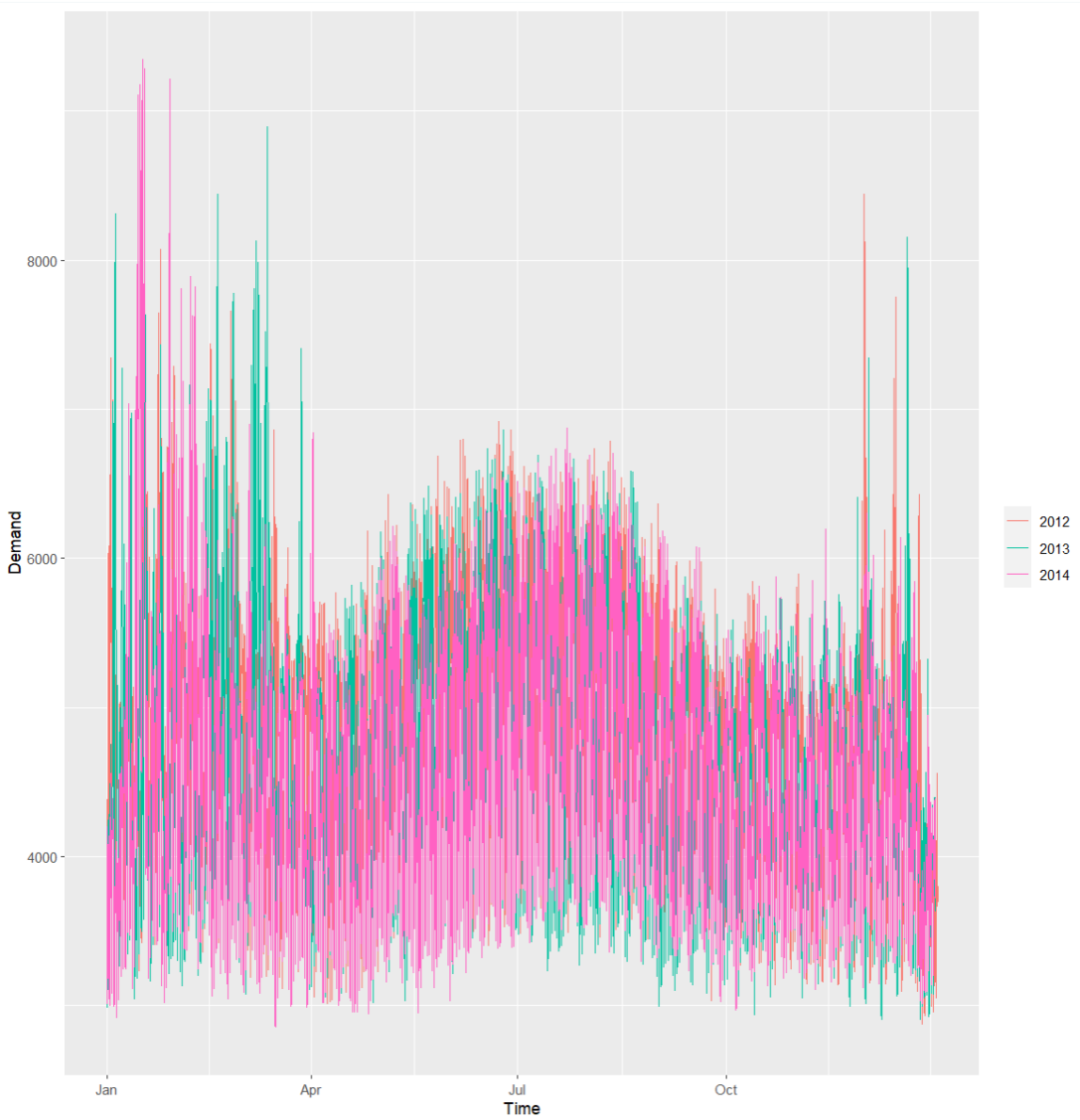

3e+07
2e+07
1e+07
TotalCost
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month
2005
2000
1995

- We can see that the lines are mostly parallel $\rightarrow$ there is a seasonal pattern in data.
- We can see that the values in January are high, which drop for February.
- We can see that in 2008 March there was a small number of sales (in general March has higher sales than February).
- Also we can see that over the years the sales have increased in general (more recent lines are above the older ones - this is actually the increasing trend of the time series).

```
AirPassengers |>
  as_tsibble() |>
  gg_season(value, labels = "left", period = 12) +
  labs (x = "Month", y = "Number of passengers in thousands", title = "Airline
      passengers")
```

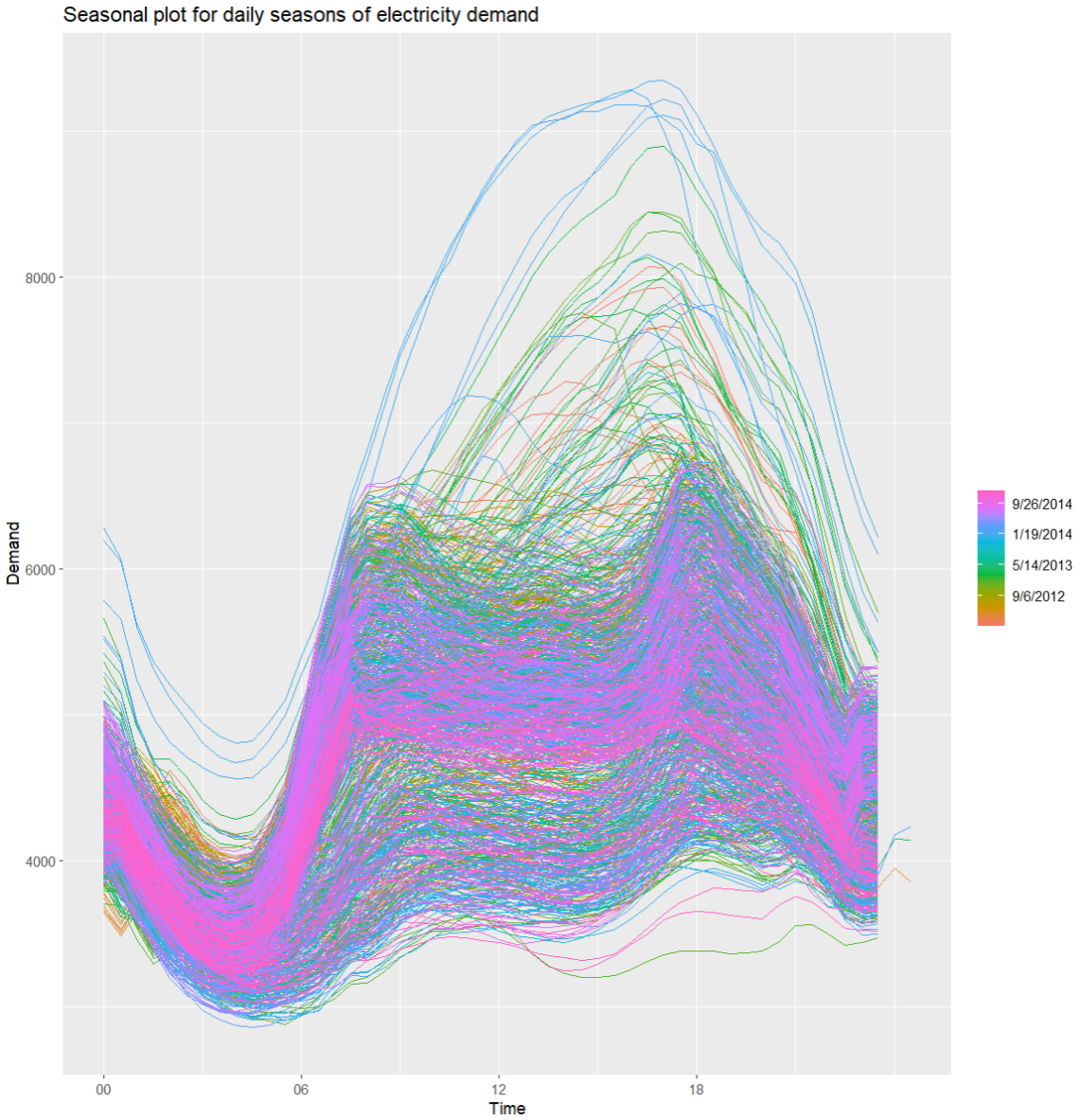


```
vic_elec |> gg_season(Demand)
```



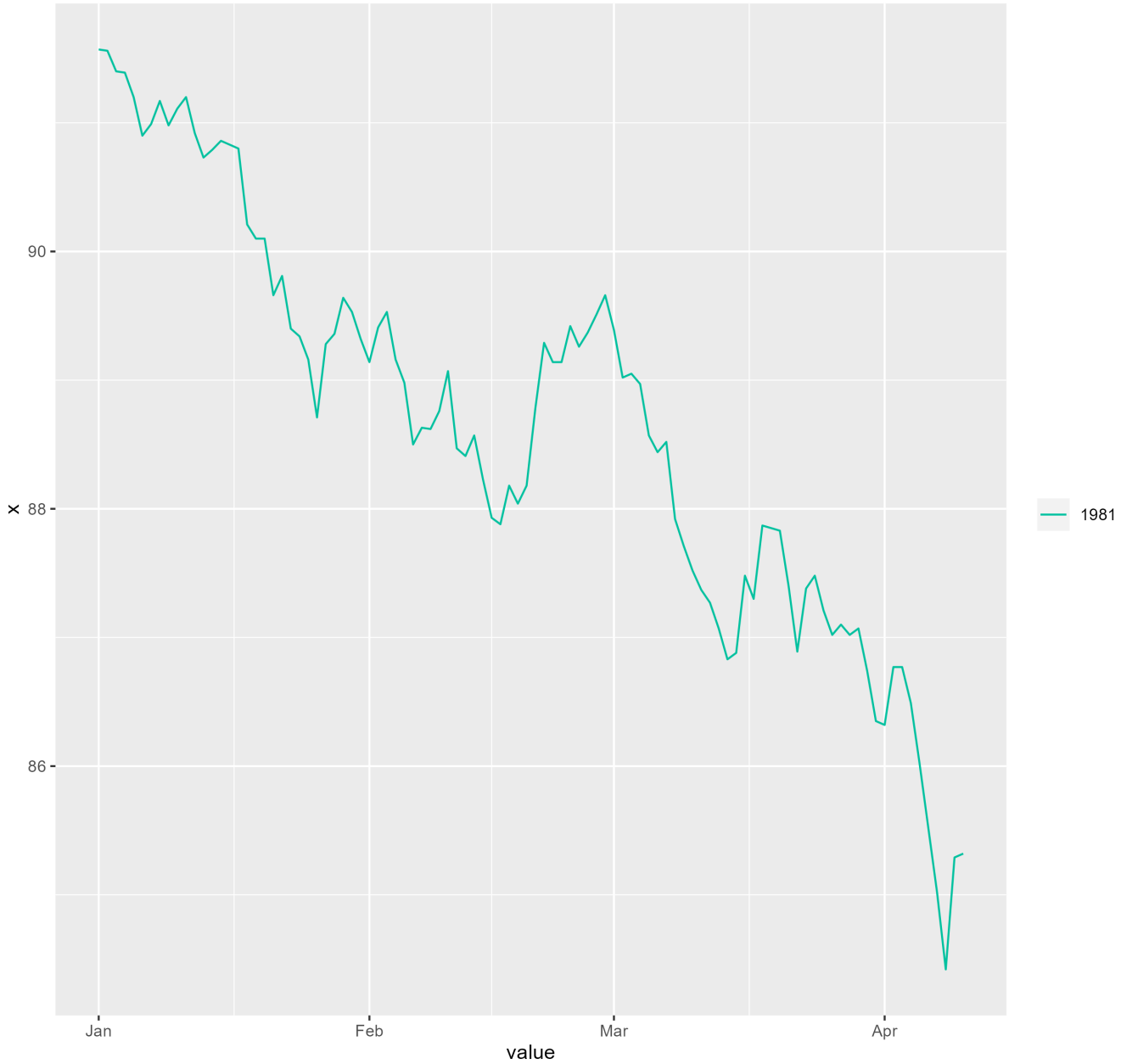
- We saw from the time series plot that the *vic_elec* data has two seasons. We can plot the daily season as well.

```
vic_elec |>
    gg_season(Demand, period = "day") +
    labs(title = "Seasonal plot for daily seasons of electricity demand")
```

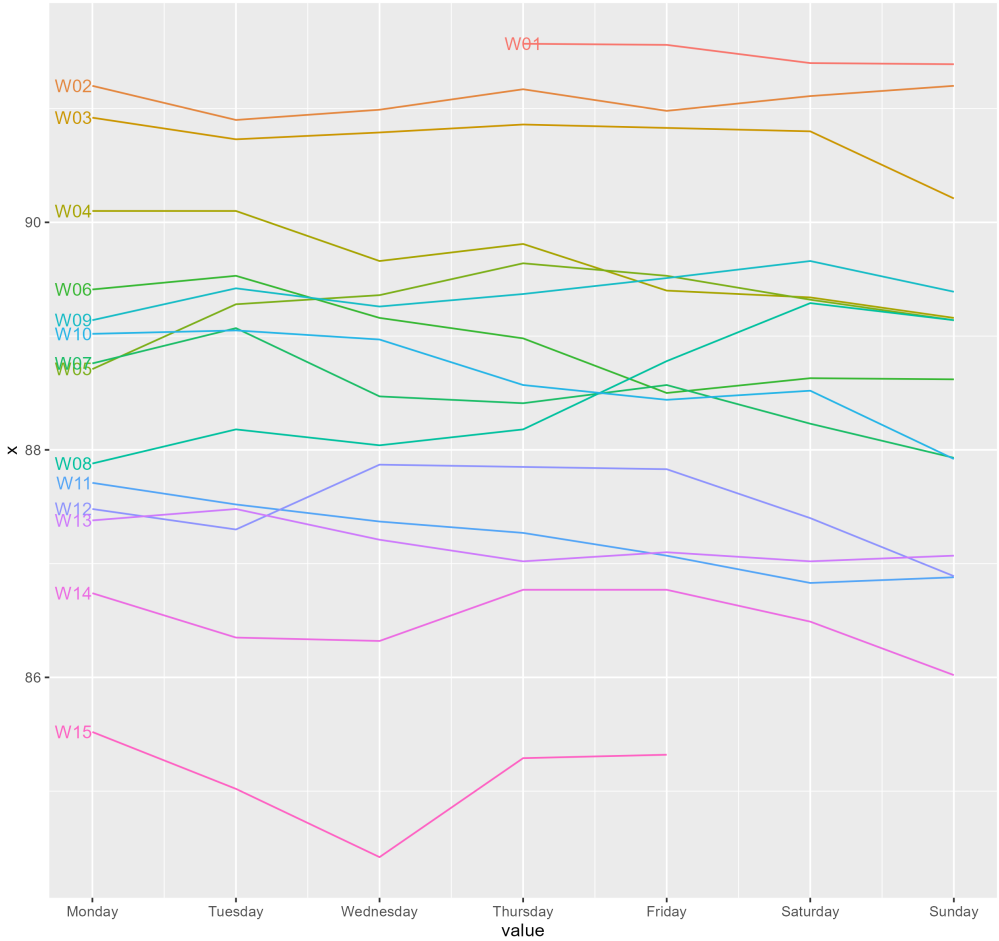

Seasonal plot for daily seasons of electricity demand
Demand
8000
6000
4000
00
06
12
18
Time
9/26/2014
1/19/2014
5/14/2013
9/6/2012

- Let's see what happens if we plot a time series without seasons, for example the *ustreas* time series. Since it is a *ts* object, it cannot be directly used in *gg_season*, it does not have *time* information.
- We know that the data was collected over 100 days, so we need to generate a sequence of 100 days to match with the time series. This can be done with the *seq* function. **Obs.** The description of the time series mentions business days and we will generate regular days, including weekends.
- Then we will transform both the time series and the sequence of says into tibble, bind them and transform into a tsibble, using the generated days as index.

```
ustreas |> as_tsibble() -> ustreasTsibble

ustreasTsibble
days <- seq(as.Date("1981-01-01"), by = "day", length = 100)

ustreasTsibble  <- tsibble(
  Date = days,
  Values = ustreas,
  index = Date
)
ustreasTsibble
ustreasTsibble |> autoplot()
```

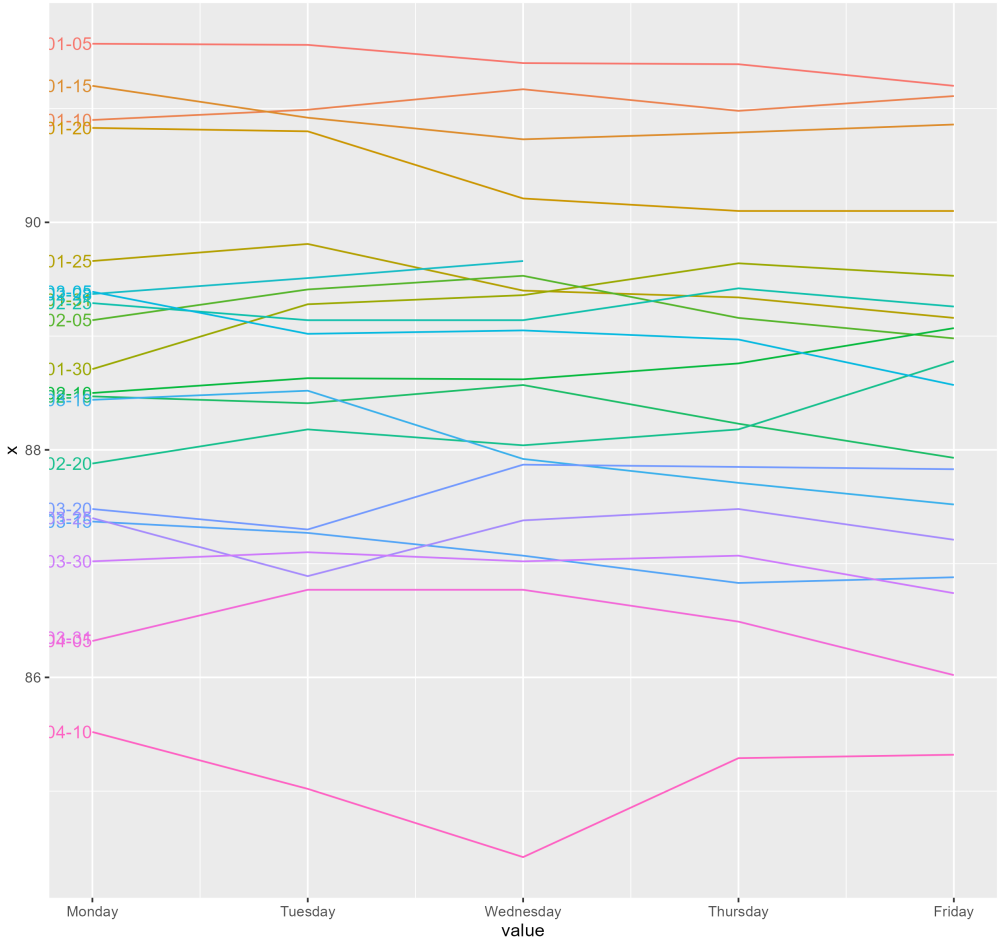


- By default, gg_season uses a period of a year.



- If we set the period to be a week, it does not seem to have a pattern in it. (But we can see the downward trend, upper lines are from January, then in the middle we have February and at the bottom of the plot we have the data for March and April)

- When the period is set to a month, we only have 4 lines, but it still does not seem to contain any seasonal pattern (other than a descending trend).



- When the period is set to 5 days (kind of like a business week), we still does not seem to have any seasonal pattern.
- While there are better alternatives (and we will talk about them), seasonal plots can be used to see if there are seasons in data, in this case, there are not.

Missing data

- In case a time series, we can have two types of missing data: implicit missing data (when actual time stamps are missing) or explicit missing data (when time stamps are present, but there are no values associated to them)
- If we have implicit missing values in the time series *gg_season* will not work.
- If we have explicit missing values in the time series, *gg_season* will just leave an empty space in the plot.
- We can transform implicit missing values into explicit ones using the *fill_gaps* function.

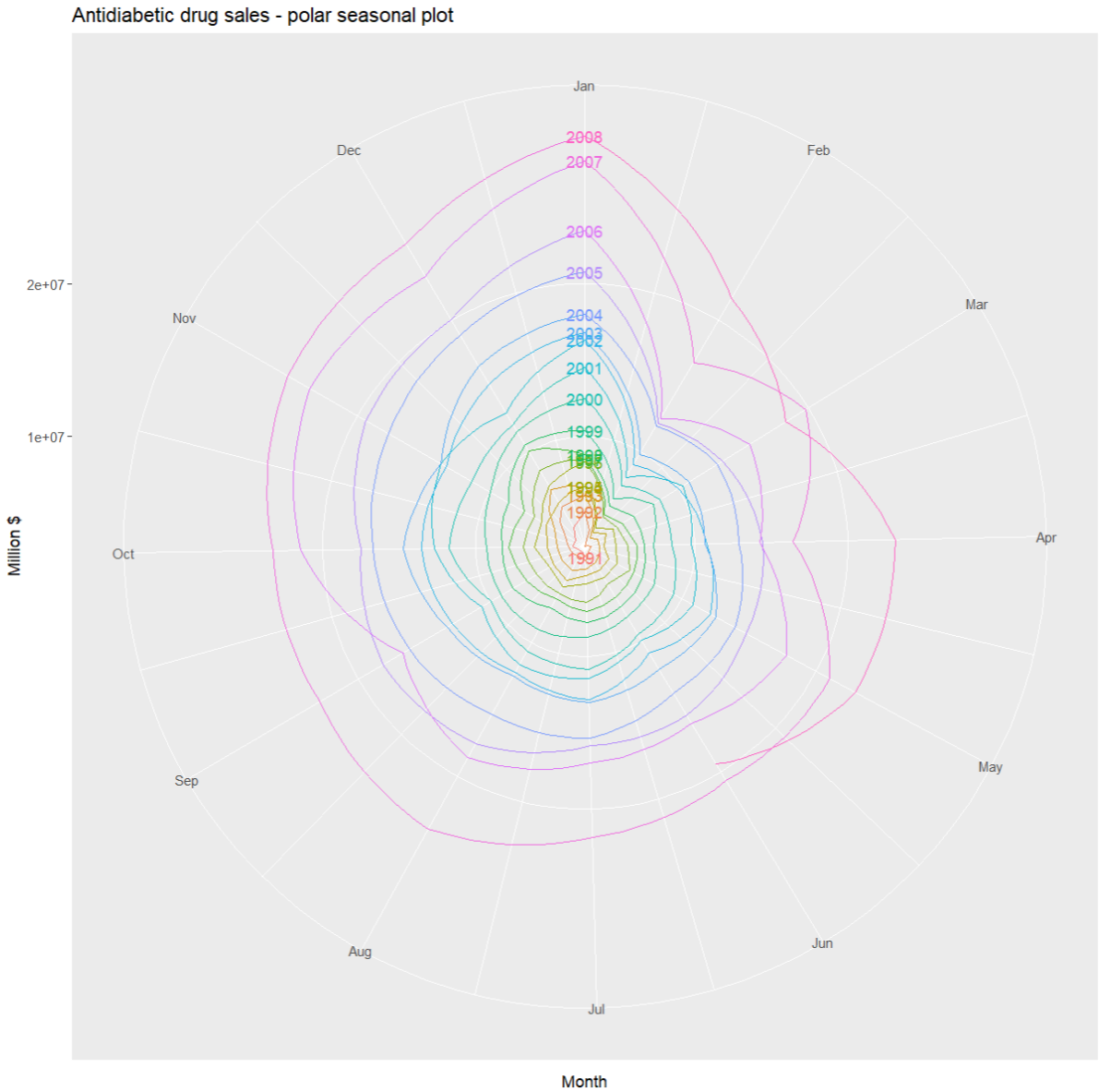
Transforming missing values

- *Missing data* can be found frequently at day-level time series, where only business days are considered (for example stock market information).
- One such example is the already mentioned EuStockMarkets time series.

```
EuStockMarkets
EuStockMarkets |>
  as_tsibble() |>
  filter(key == "DAX") -> DaxStock


DaxStock |>
  mutate(Date = as.Date(index)) |>
  update_tsibble(index = Date) |>
  select(Date, value) ->DaxStock

DaxStock
DaxStock |> fill_gaps(.full = TRUE) -> DaxStockNoMissing
```

- Another option when creating seasonal plots is to use polar coordinates for it, by setting the value of the *polar* parameter to TRUE.

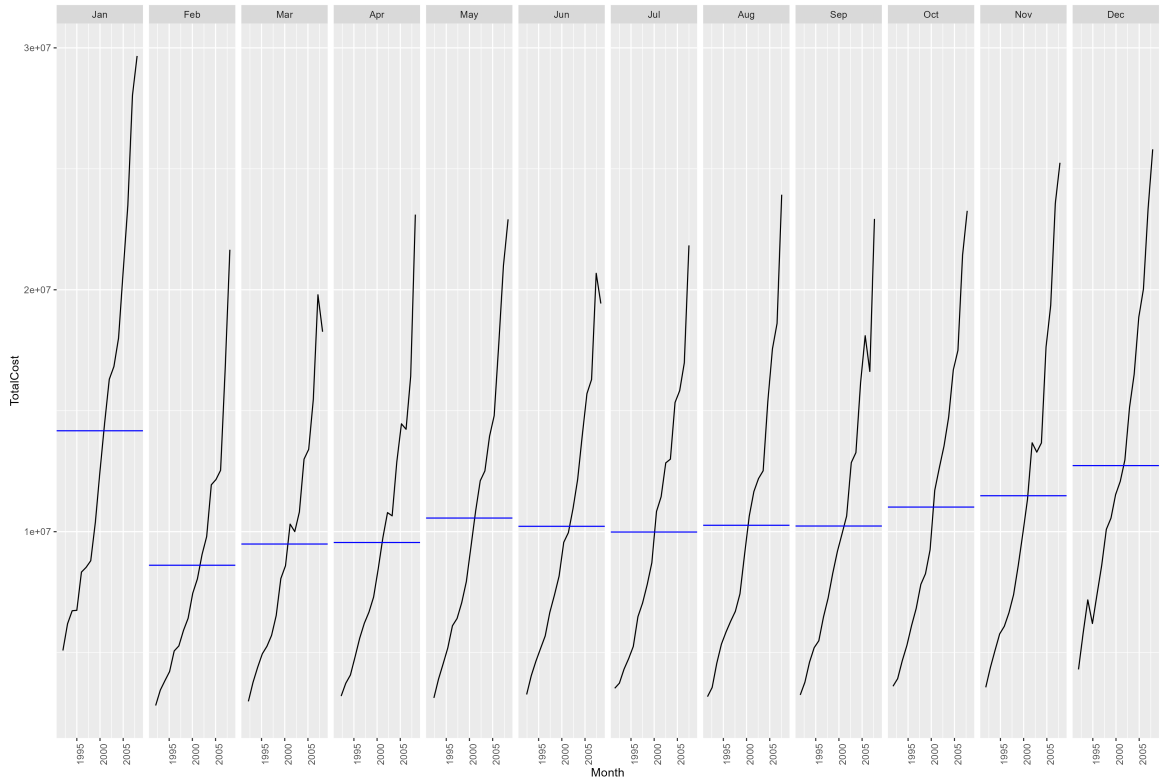
```
gg_season(A10, labels="left", polar = TRUE) +
 ylab("Million $") +
 ggtitle("Antidiabetic drug sales – polar seasonal plot")
```


Antidiabetic drug sales - polar seasonal plot
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
2e+07
1e+07
Million $
2008
2007
2006
2005
2004
2001
2000
1999
1991
Month

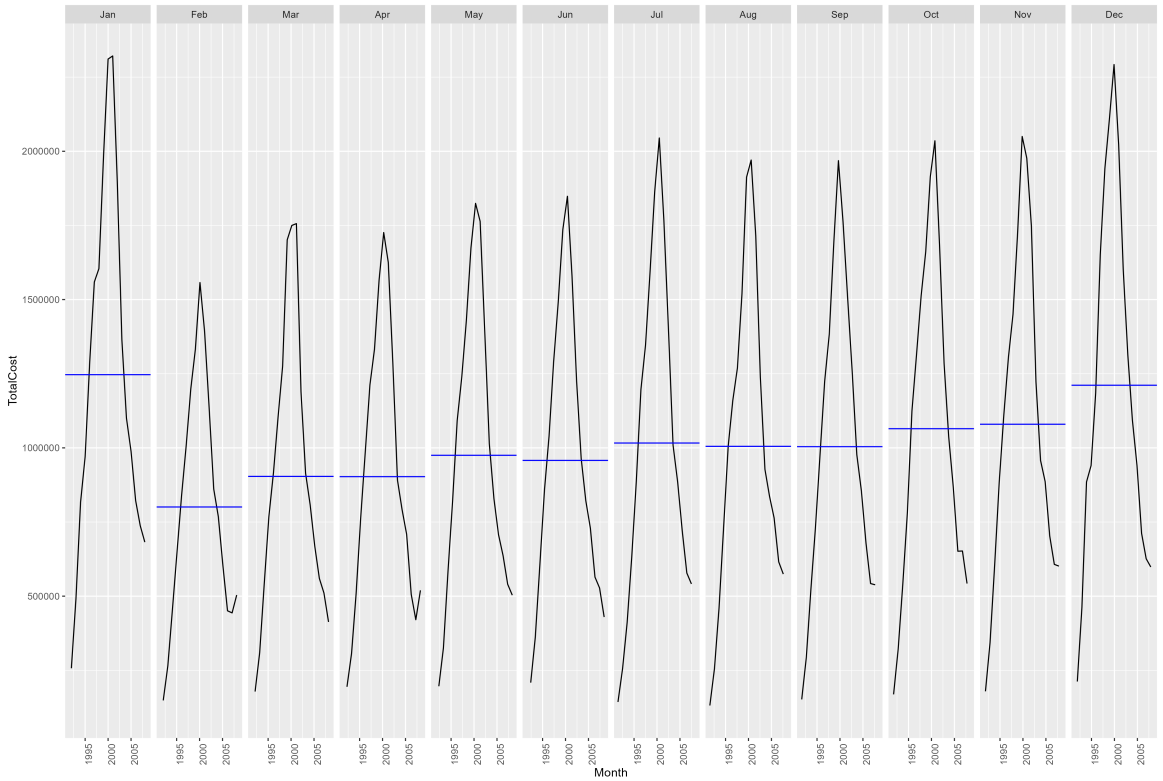
Seasonal subseries plot

- Another plot suitable for seasonal data is the *seasonal subseries plot*, which creates a separate plot for each season of your data for points from successive time periods. For each subplot, a horizontal line is also drawn which is the mean of the values.
- Such a plot can reveal:
  - Changes within a given season over time
  - Changes between different seasons
- On the other hand, it is complicated to compare the time series to each other using the seasonal subseries plot.
- You can create a seasonal subseries plot with the *gg_subseries* function.

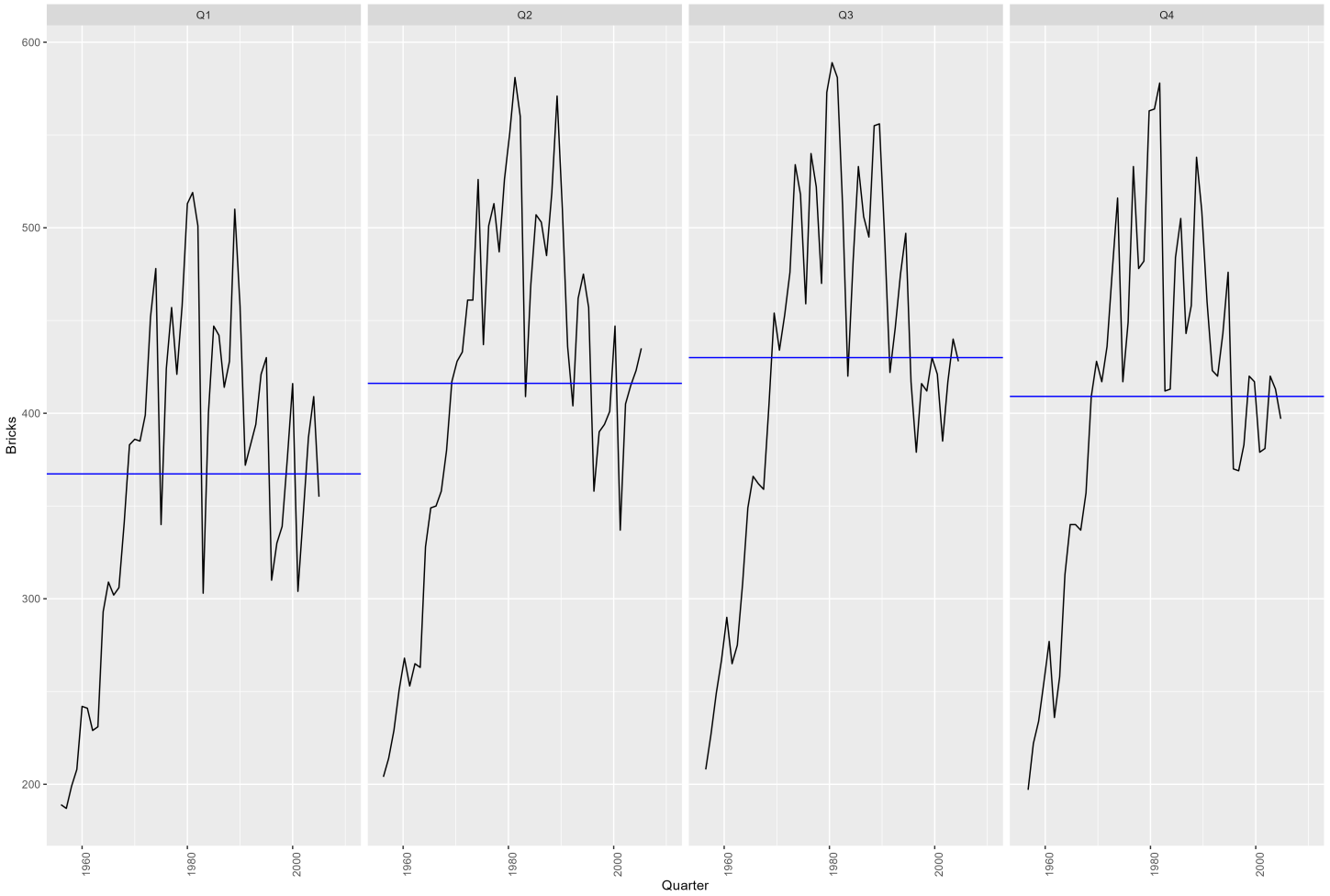
```
A10 |>
    gg_subseries()
```



```
A11 |>
    gg_subseries()
```



```
aus_bricks |> gg_subseries()
```



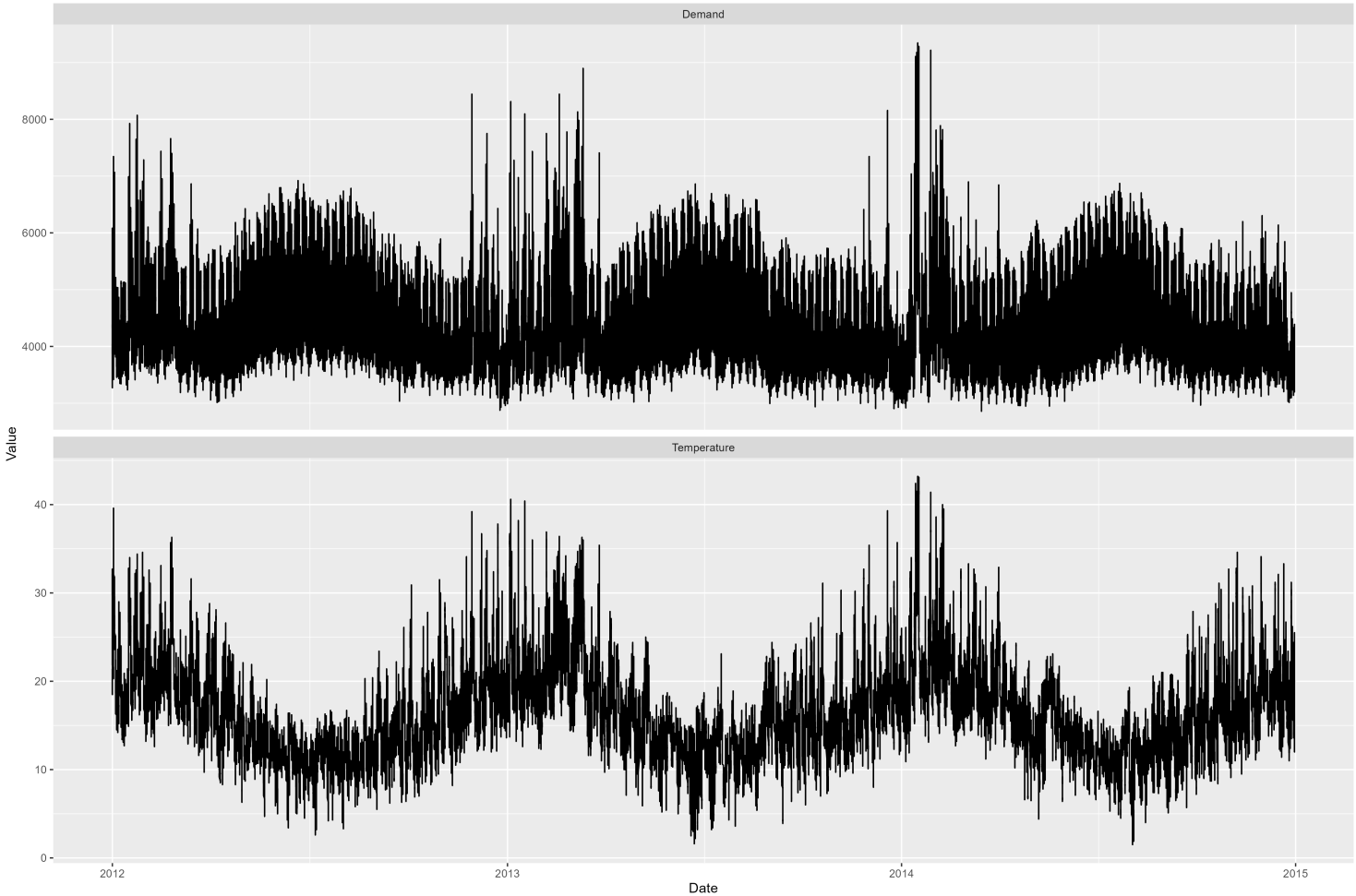
Scatter plot

- Time series, seasonal and seasonal subseries plots can be used to visualize one single time series. Sometimes we want to see relations between time series.
- For example: the *vic_elec* time series contains three attributes:
  - Demand - Total electricity demand for Victoria, Australia (every half-hour)
  - Holiday - TRUE if the current day is a Holiday, FALSE otherwise
  - Temperature - temperature for Melbourne (every half hour)
- Demand and temperature are time series, so we can plot them.

- *autoplot* cannot handle two time series, unless they are in the long data format (and *vic_elec* is in the wide format). So we will first transform it, using *pivot_longer*.

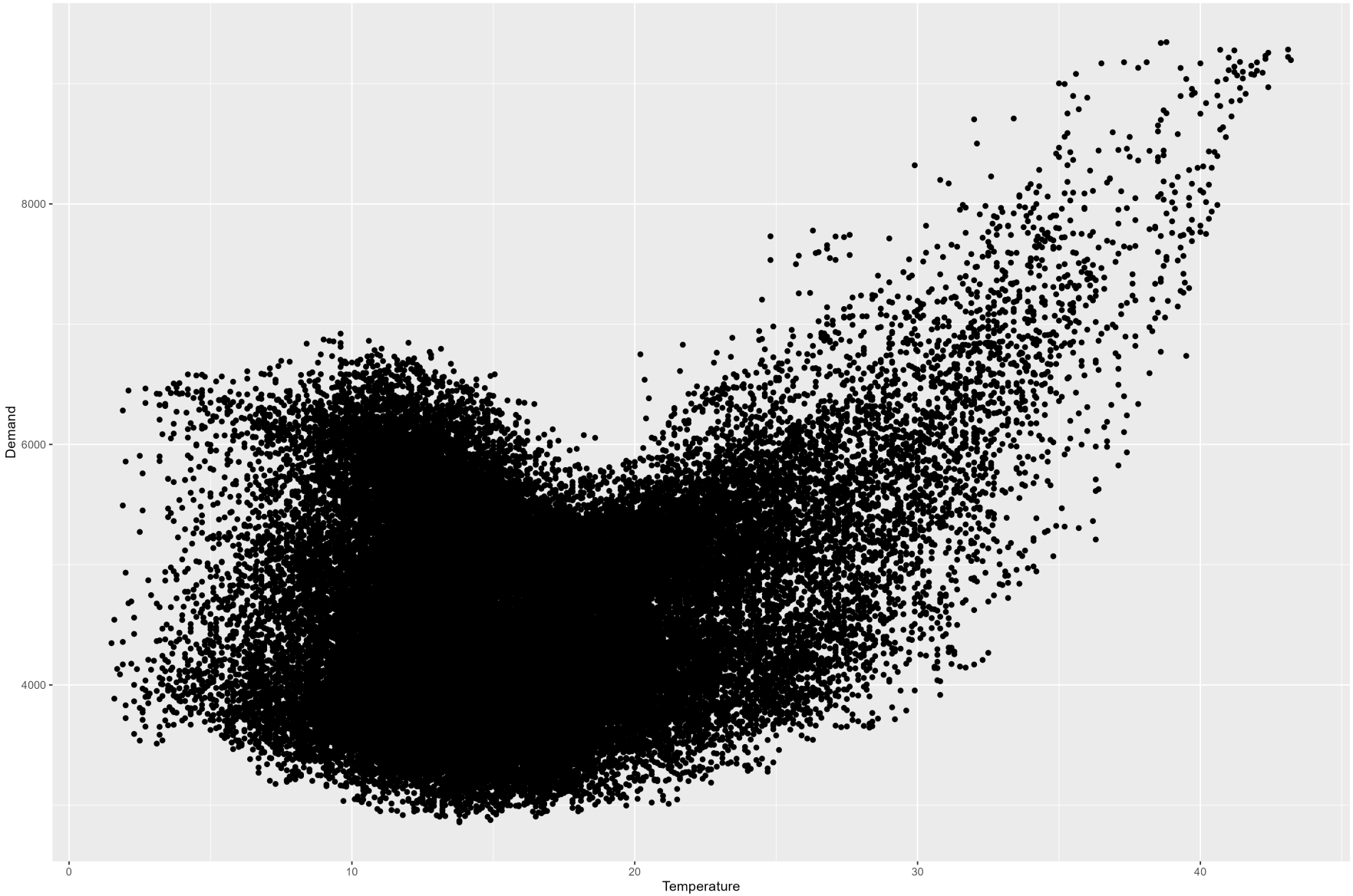
- Now we could autoplot it, but since the scales are so different (Temperature is between 1.5 and 43.20, while Demand is between 2858 and 9345) Temperature will not really be visible.
- We should plot them in two different facets and for this we need *ggplot*.

```
pivot_longer(vic_elec, cols = c("Demand", "Temperature"),
   names_to = "Attribute", values_to = "Value") -> vic_elec_long
autoplot(vic_elec_long)
ggplot(data = vic_elec_long) +
  geom_line(mapping = aes(x = Date, y = Value)) +
  facet_wrap(vars(Attribute), scales= "free_y", nrow = 2, ncol = 1)
```


Demand
8000
6000
4000
Temperature
40
30
20
10
0
Value
2012
2013
2014
2015
Date

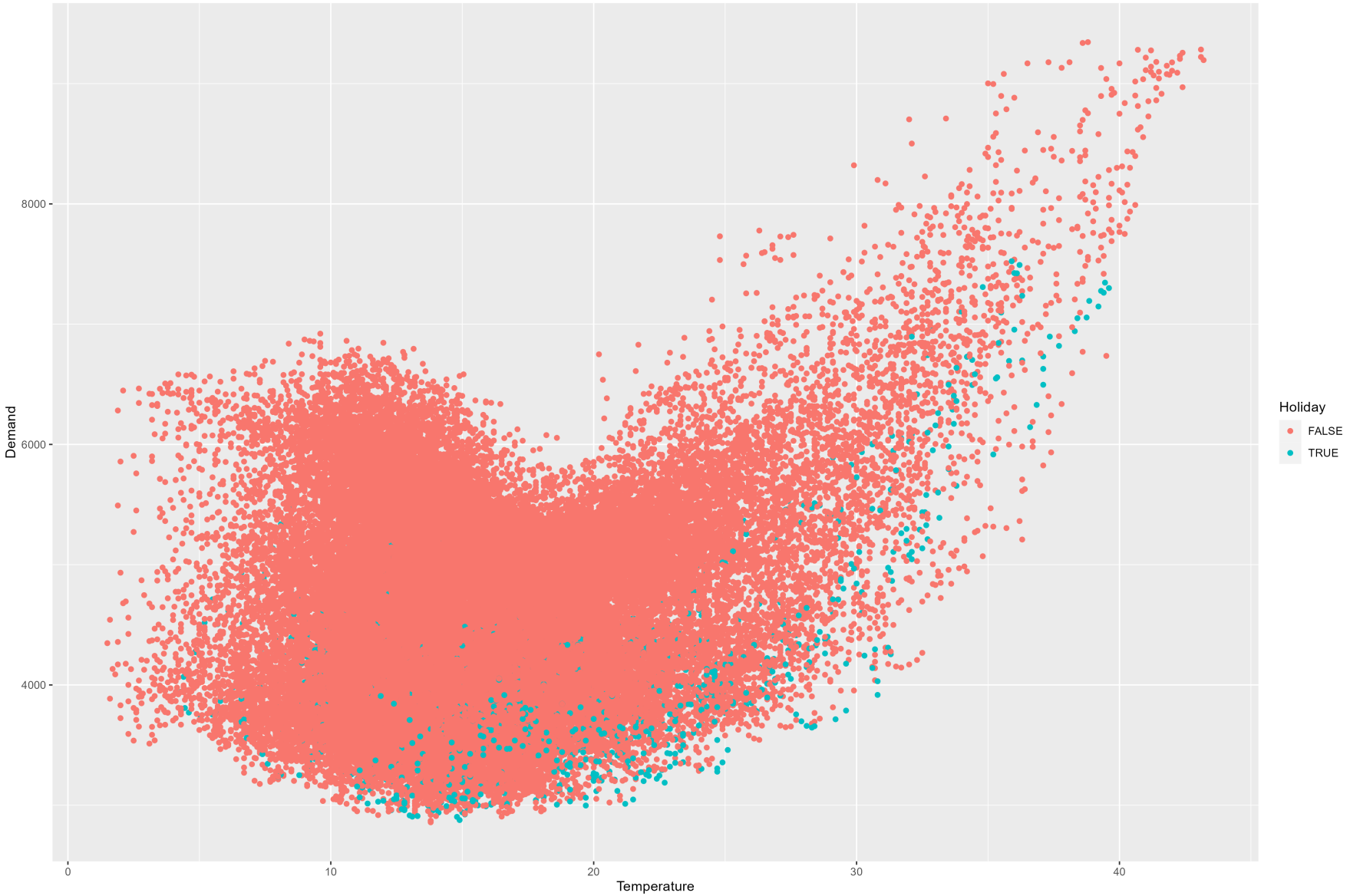
- While we might be able to see that there are some common patterns in both series, it is more helpful if we plot the two series together, one against the other in a scatter plot, where each point of the plot is a (Temparature, Demand) pair of observations.

```
ggplot(data = vic_elec) +
  geom_point(mapping = aes(x = Temperature, y = Demand))
```


Demand
8000
6000
4000
0
10
20
30
40
Temperature

- Remember, *vic_elec* contained an attribute denoting if the current day is a holiday or not. By adding a color aesthetic for this, we can see whether the relation between Temperature and Demand is the same on holidays as in regular days.

```
ggplot(data = vic_elec) +
  geom_point(mapping = aes(x = Temperature, y = Demand, color = Holiday))
```


Demand
8000
6000
4000
0
10
20
30
40
Temperature
Holiday
FALSE
TRUE

Correlation

- Scatterplots help us visualize relationships between time series. In the previous example we could see that electricity demand increases when temperatures increase (people turn on air-conditioning), but to a limited extent, demand increases when temperatures are low (due to some using electricity for heating).
- If we want a numerical value for how strong is the relationship between two variable we can compute the *correlation coefficient*.
- Correlation takes values between -1 (strong negative correlation) and 1 (strong positive correlation). Values around 0 mean that the variables are not correlated.

Correlations

- Correlations in R can be computed with the *cor* function.
- There are several types of correlations, the *cor* function can compute 3 of them:
  - Pearson (the default) - measures linear relationship
  - Spearman - measures the strength of monotonic association between two variables
  - Kendall - Similar to Spearman (as in works on ranks), but it is better when there are few data points and many rank ties.

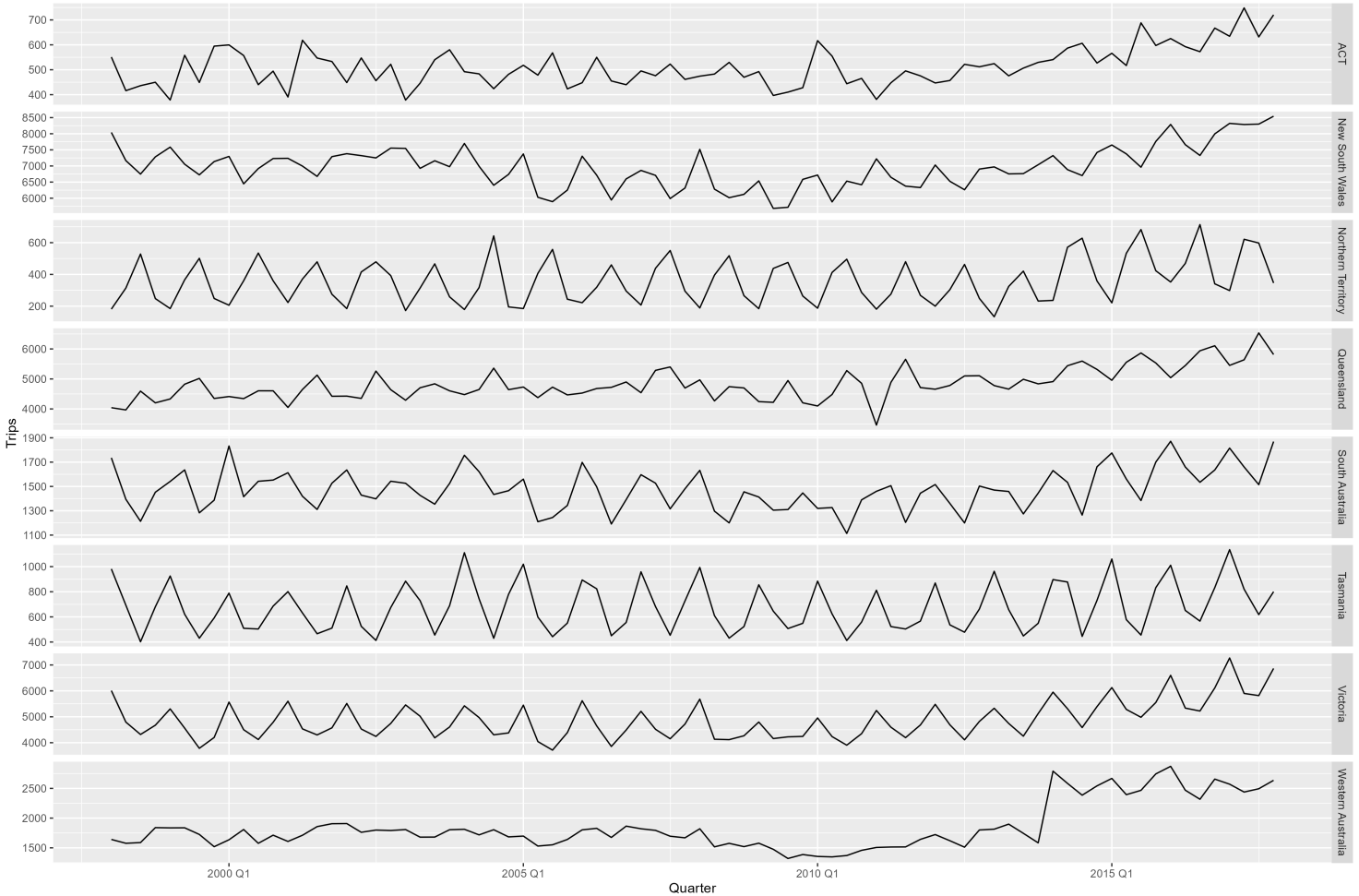
- Pearson correlation for the electricity demand and temperature data is 0.26, which is quite low. The reason for this is that the relationship between these two variables is not linear.
- Spearman and Kendall correlations are even lower (0.11 and 0.08).
- This shows that it is important to plot the data as well, not just rely on numerical correlation values.

- Correlations should be used very carefully when we work with time series!!!
- Two time series (for which we compute correlations) are actually connected by time, which might influence (increase) the results.
- This is especially true in case of time series which have a trend.
- Spurious Correlations: `https://www.tylervigen.com/spurious-correlations`
- Nice reading and example about what is the problem when computing correlation between two trended time series: `https://www.svds.com/avoiding-common-mistakes-with-time-series/`

- Sometimes we have data which consists of several time series (one example is the *vic_elec* data set containing two time series).
- In this case, we might want to create scatter plots for every combination of variables.
- For example: the *tourism* data set contains 304 time series (total of 24320 observations), each of them representing the quarterly visitor nights for different regions of Australia, between 1998 - 2017. It contains the following attributes:
  - Quarter
  - Region (76 different values)
  - State (8 different values)
  - Purpose (4 possible values: Business, Holiday, Visiting, Other)
  - Number of trips
  - The keys of the data set are the Region, State and Purpose (but Region defines the State automatically)

- To reduce the number of time series, let's group the observations by state (which means that we will have one observation per state) and compute the total number of trips for each of them.
- Now we only have 640 observations (8 time series * 4 quarters * 20 years)
- Let's plot the 8 time series (using facets)

```
visitors <- tourism |>
  group_by(State) |>
  summarize(Trips = sum(Trips))

ggplot(data = visitors) +
  geom_line(mapping = aes(x = Quarter, y = Trips)) +
  facet_grid(vars(State), scales = "free_y")
```

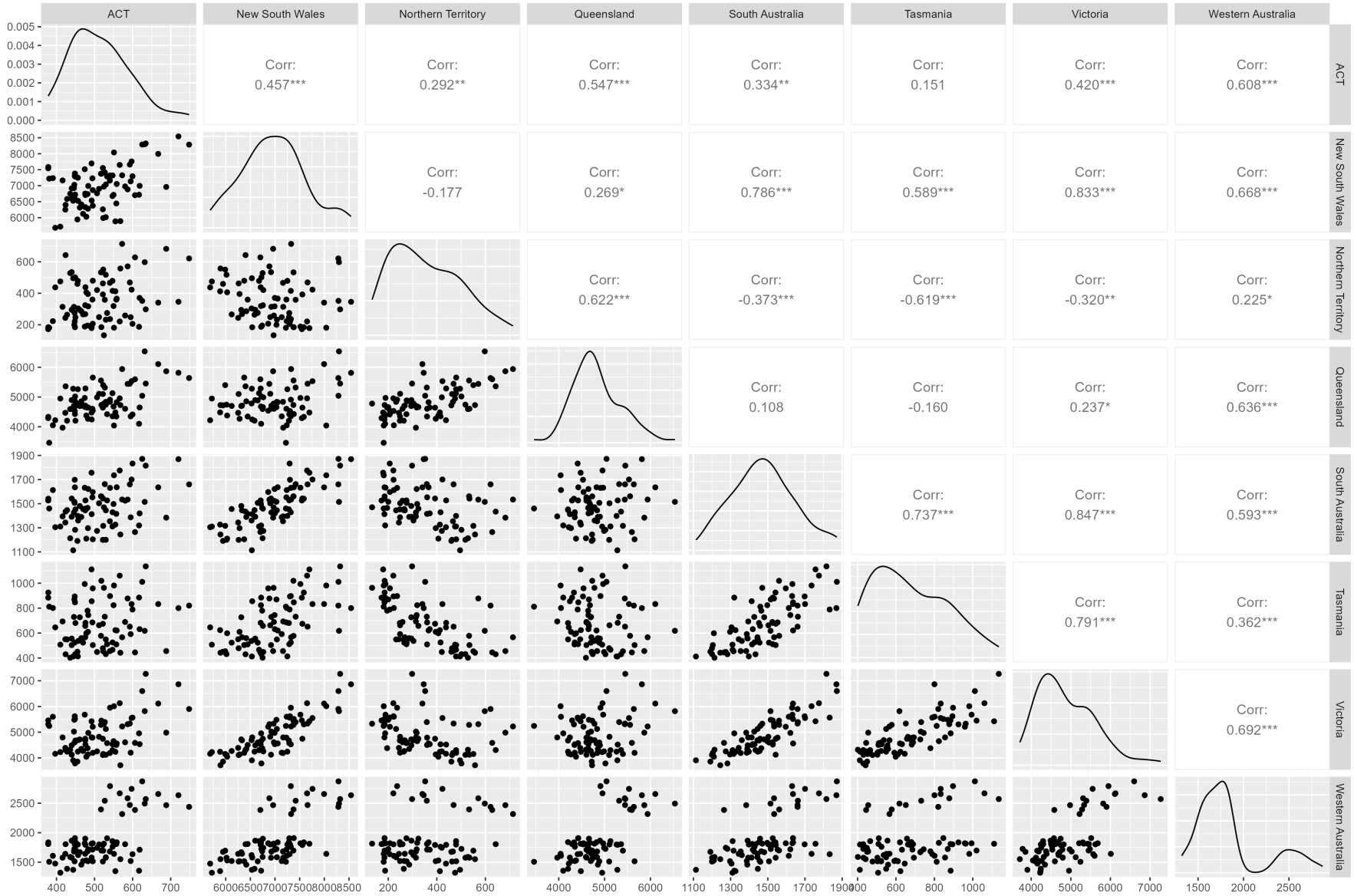

Trips
Quarter
2000 Q1
2005 Q1
2010 Q1
2015 Q1
ACT
New South Wales
Northern Territory
Queensland
South Australia
Tasmania
Victoria
Western Australia

- We might want to visualize scatter plots between pairs of variables, which can be done with the *ggpairs* function, for which we need to install the *GGally* package.

- *ggpairs* takes the time series to be considered as columns (needs wide format) but we have the long one. So we will use *pivot_wider* to transform it first.

- *ggpairs* by default considers all attribute pairs. But we have the Quarter attribute which should not be considered. So we will use the *columns* parameter to show which attributes to consider.

```
visitors_wide <- pivot_wider(visitors, values_from= Trips, names_from=State)
ggpairs(visitors_wide, columns = 2:9)
```


ACT
New South Wales
Northern Territory
Queensland
South Australia
Tasmania
Victoria
Western Australia
Corr: 0.457***
Corr: 0.292**
Corr: 0.547***
Corr: 0.334**
Corr: 0.151
Corr: 0.420***
Corr: 0.608***
Corr: -0.177
Corr: 0.269*
Corr: 0.786***
Corr: 0.589***
Corr: 0.833***
Corr: 0.668***
Corr: 0.622***
Corr: -0.373***
Corr: -0.619***
Corr: -0.320**
Corr: 0.225*
Corr: 0.108
Corr: -0.160
Corr: 0.237*
Corr: 0.636***
Corr: 0.737***
Corr: 0.847***
Corr: 0.593***
Corr: 0.791***
Corr: 0.362***
Corr: 0.692***

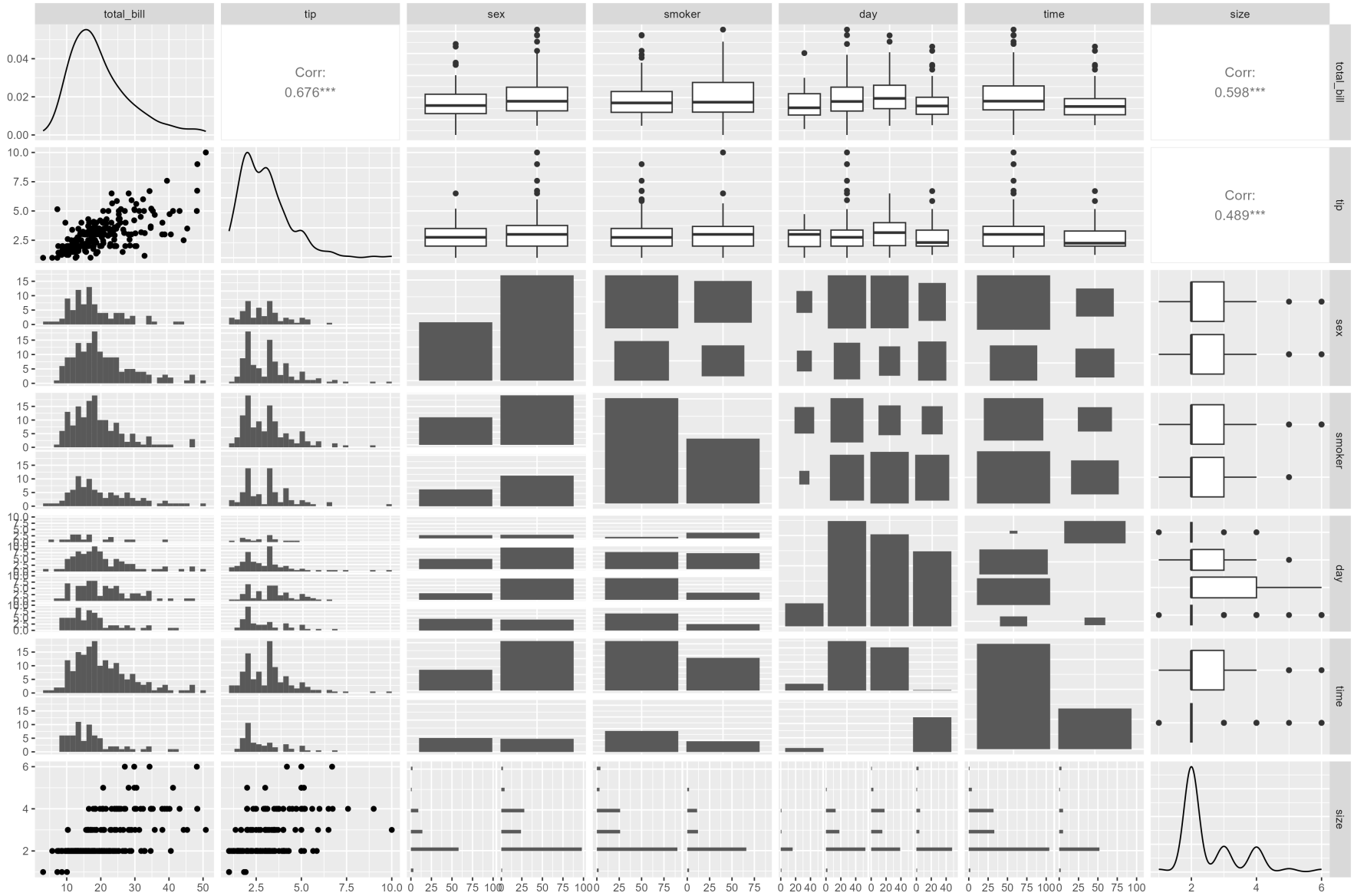
- On the main diagonal we have density plots of the attributes (how frequent are different values of the attribute).
- Below the main diagonal we have the scatterplots. For each scatterplot, the variable on the y-axis is the one whose name is on the corresponding row, while the variable on the x-axis is the one whose name is on the corresponding column.
- Above the main diagonal we have correlations.

- *ggpairs* is a very versatile function, it has many options. The *switch* parameter determines where the line and column labels should be (by default it is right side and bottom), the value *"x"* specifies that the column labels should be at the bottom, *"y"* specifies that the row labels should be at the left side and *"both"* swaps both labels.
- The diagram has three main sections:
  - the main diagonal (*diag*)
  - below the main diagonal (*lower*)
  - above the main diagonal (*upper*).
- The data on the main diagonal can be either *continuous* or *discrete*. The other two sections can be *continuous* (both), *discrete* (both) or *combo* (one continuous and one discrete). For each tipe of data you can set the type of visualization to use.
- Moreover, you can define aesthetics as well (just like in case of *ggplot*).

- Let's see a few examples of what you can do with the *ggpairs* function, considering a non-time series data set, *tips* (from package *reshape*), containing information recorded by a waiter about received tips. It contains the following attributes:
  - total bill - the total amount to be payed, numeric.
  - tip - the received tip, numeric
  - sex - the sex of the bill payer, discrete
  - smoker - whether there were smokers among the customers, Yes or No
  - day - day of week, discrete (Sunday, Saturday, Thursday, Friday)
  - time - time of the day, discrete (Dinner, or Lunch)
  - size - size of the party, discrete/numeric (numbers 1 to 6)
- **Obs:** Inspiration for these visualizations was taken from `https://ggobi.github.io/ggally/articles/ggpairs.html`

- Let's start by visualizing everything.

```
ggpairs(tips)
```


total_bill
tip
sex
smoker
day
time
size
Corr:
0.676***
Corr:
0.598***
Corr:
0.489***

- We can see that on the main diagonal we have density plots (for continuous values - total_bill, tip and size) or bar plots with frequencies (the other attributes).
- For the lower part, we have:
  - scatter plot, when both attributes are numerical (combinations of tip, total_bill and size).
  - histogram (facethist), when we have a numerical and a categorical attribute
  - bar plots (but for the combination of possible values) when we have two discrete variables.
- For the upper part we have:
  - boxplots - when we have a continuous and a discrete variable
  - correlation - when we have two continuous variables
  - count plot - when we have two discrete variables (area of the figure is proportional to the count for the combination of values).

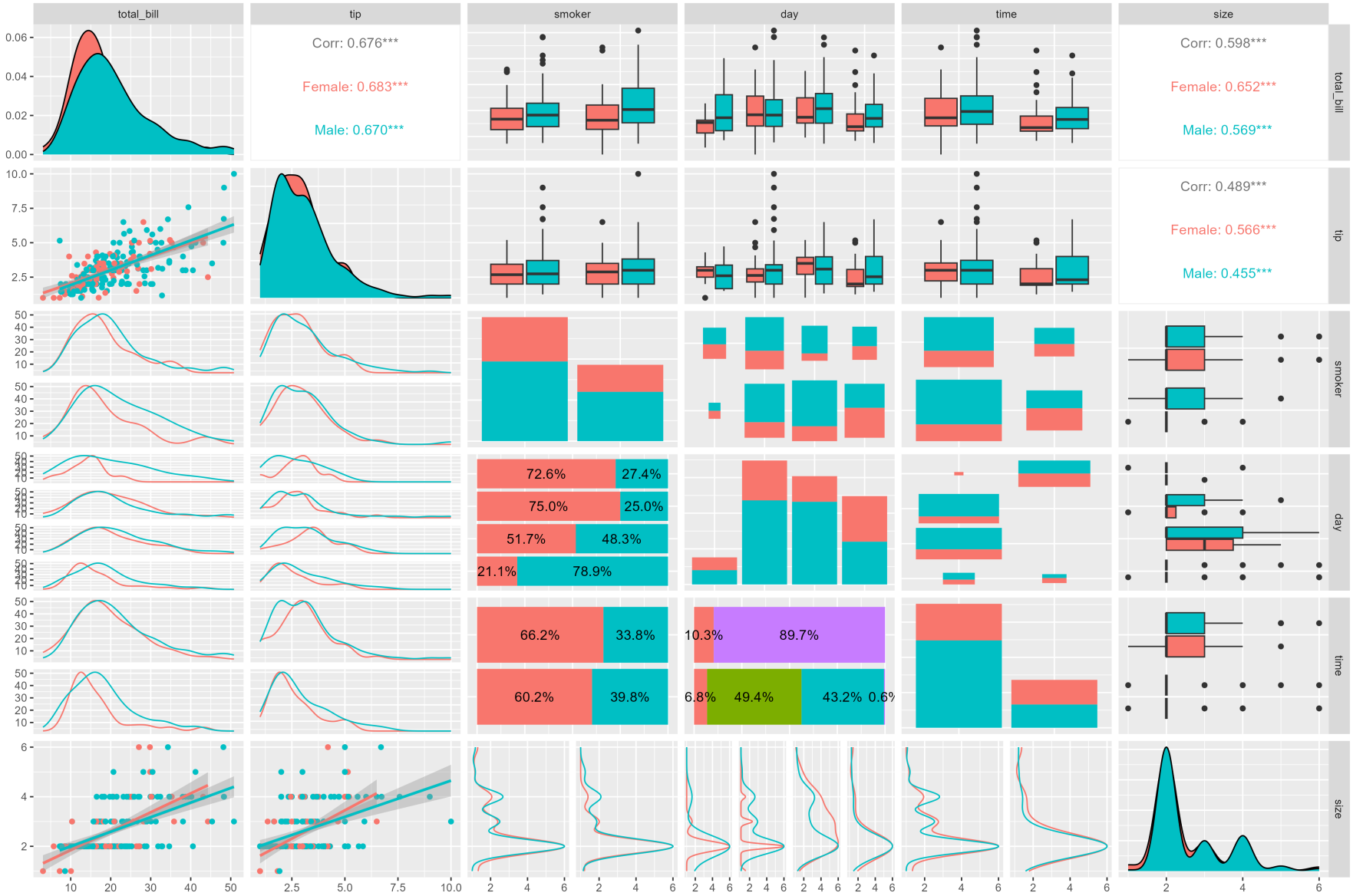
- Let's simplify a little, by removing *sex* and making it an aesthetic.
- We will have the same type of plots, but one attribute less, and everything will be colored according to sex.

```
ggpairs(tips, columns =c(1,2,4,5,6,7), mapping = aes(colour = sex))
```


total_bill
tip
smoker
day
time
size
Corr: 0.676***
Female: 0.683***
Male: 0.670***
Corr: 0.598***
Female: 0.652***
Male: 0.569***
Corr: 0.489***
Female: 0.566***
Male: 0.455***

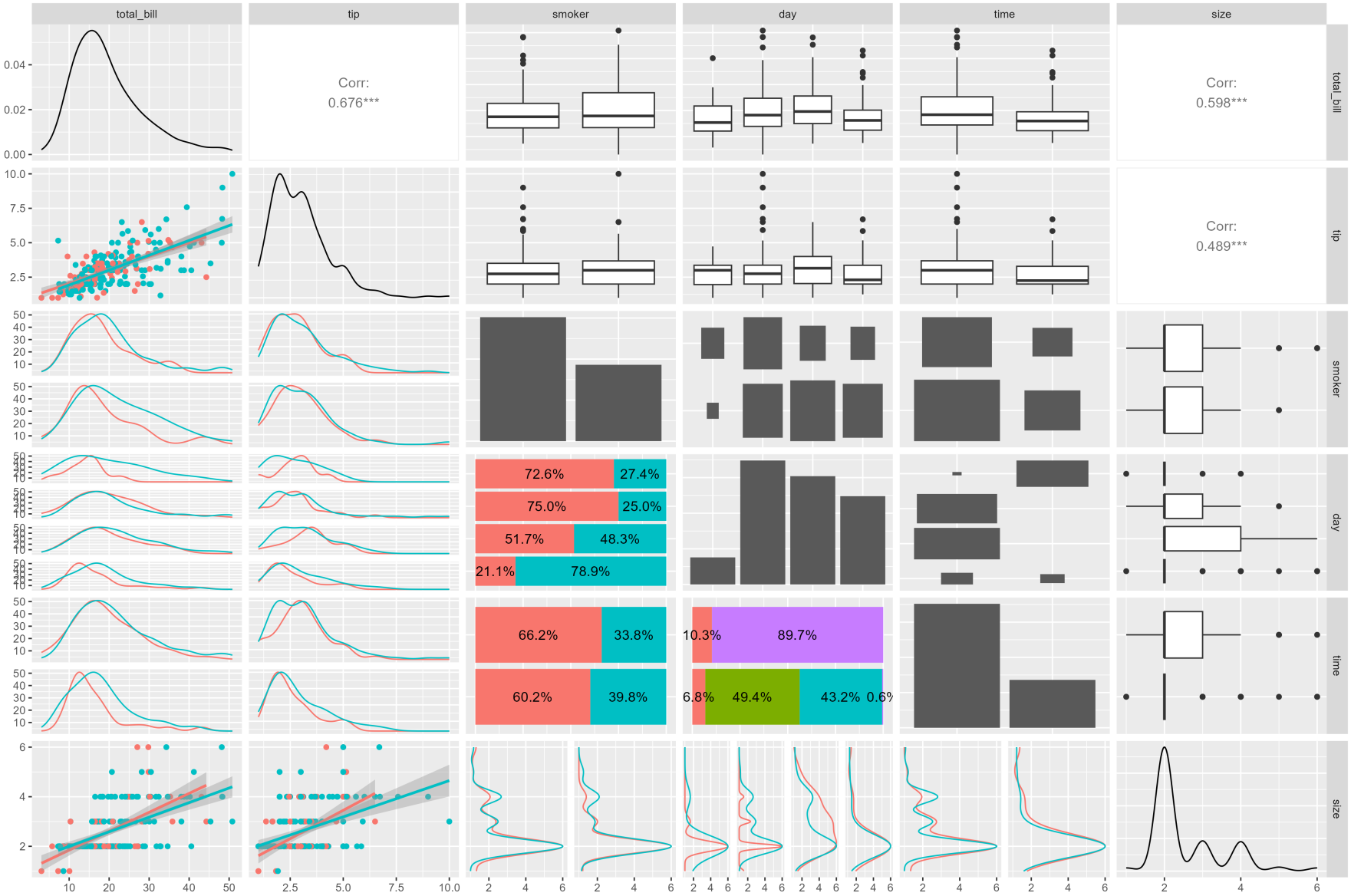
- In order to modify the chart types, we need to use the *lower* and *upper* parameters. They take as value a list, with three name-value pairs.
- The names are *continuous*, *combo*, *discrete*, while the values are the types of the plot to use.
- The complete list of possible plot types for each combination (with examples) can be found here: `https://ggobi.github.io/ggally/articles/ggally_plots.html`
- In the example below we only set custom plot types for the lower part of the matrix. We can similarly set values to the *upper* part as well and to the *diag* part, with the observation that for the diagonal we only have values for *continuous* and *discrete* variables.

```
ggpairs(tips, columns =c(1,2,4,5,6,7), mapping = aes(colour = sex),
    lower = list(continuous = "smooth", combo = "facetdensity", discrete = "
     rowbar"))
```


total_bill
tip
smoker
day
time
size
Corr: 0.676***
Female: 0.683***
Male: 0.670***
Corr: 0.598***
Female: 0.652***
Male: 0.569***
Corr: 0.489***
Female: 0.566***
Male: 0.455***
72.6%
27.4%
75.0%
25.0%
51.7%
48.3%
21.1%
78.9%
66.2%
33.8%
60.2%
39.8%
10.3%
89.7%
6.8%
49.4%
43.2%
0.6%

- You can add the aesthetic also as part of the upper or lower list, and in this case it will only be applied to the plots on that side.

```
ggpairs(tips, columns =c(1,2,4,5,6,7),
   lower = list(continuous = "smooth", combo = "facetdensity", discrete = "
     rowbar", mapping = aes(color = sex)))
```


total_bill
tip
smoker
day
time
size
Corr:
0.676***
Corr:
0.598***
Corr:
0.489***
72.6%
27.4%
75.0%
25.0%
51.7%
48.3%
21.1%
78.9%
66.2%
33.8%
60.2%
39.8%
10.3%
89.7%
6.8%
49.4%
43.2%
0.6%