Tidy Time-series Data Analysis

Dr. Kam Tin Seong
Assoc. Professor of Information Systems

School of Computing and Information Systems,
Singapore Management University

2022-7-14 (updated: 2022-07-17)

Content

By the end of this session, you will be able to:

- import and wrangling time-series data by using appropriate tidyverse methods,
- visualise and analyse time-series data,
- calibrate time-series forecasting models by using exponential smoothing and ARIMA techniques, and
- compare and evaluate the performance of forecasting models.

Getting Started

For the purpose of this hands-on exercise, the following R packages will be used.

```
pacman::p_load(tidyverse, lubridate, zoo,
               timetk, modeltime,
               trelliscopejs, seasonal,
               tsibble, feasts, fable)
```

- **lubridate** provides a collection to functions to parse and wrangle time and date data.
- **zoo** provides an S3 class with methods for indexed totally ordered observations, such as discrete irregular time series.
- **timetk** provides methods for analysing and visualising time series data stored in tibble data frame object.
- **seasonal** provides easy-to-use interface to X-13-ARIMA-SEATS.

- tsibble, feasts, fable and fable.prophet are belong to **tidyverts**, a family of tidy tools for time series data handling, analysis and forecasting.
  - **tsibble** provides a data infrastructure for tidy temporal data with wrangling tools. Adapting the tidy data principles, tsibble is a data- and model-oriented object.
  - **feasts** provides a collection of tools for the analysis of time series data. The package name is an acronym comprising of its key features: Feature Extraction And Statistics for Time Series.
- **trelliscopejs** is an interface for creating Trelliscope displays in R environment.

Importing the data

First, `read_csv()` of **readr** package is used to import *visitor_arrivals_by_air.csv* file into R environment. The imported file is saved an tibble object called *ts_data*.

```
ts_data <- read_csv(
  "data/visitor_arrivals_by_air.csv")
```

In the code chunk below, `dmy()` of **lubridate** package is used to convert data type of Month-Year field from Character to Date.

```
ts_data$`Month-Year` <- dmy(
  ts_data$`Month-Year`)
```

Conventional base `ts` object versus `tibble` object

tibble object

```r
ts_data
```

```
## # A tibble: 144 × 34
##    `Month-Year` `Republic of South Africa` Canada   USA Bangladesh Brunei China
##    <date>                            <dbl>  <dbl> <dbl>      <dbl>  <dbl> <dbl>
##  1 2008-01-01                         3680   6972 31155       6786   3729 79599
##  2 2008-02-01                         1662   6056 27738       6314   3070 82074
##  3 2008-03-01                         3394   6220 31349       7502   4805 72546
##  4 2008-04-01                         3337   4764 26376       7333   3096 76112
##  5 2008-05-01                         2089   4460 26788       7988   3586 64808
##  6 2008-06-01                         2515   3888 29725       8301   5284 55238
##  7 2008-07-01                         2919   5313 33183       9004   4070 80747
##  8 2008-08-01                         2471   4519 27427       7913   4183 66625
##  9 2008-09-01                         2492   3421 21588       7549   3160 52649
## 10 2008-10-01                         3023   4756 25112       7527   2983 54423
## # … with 134 more rows, and 27 more variables: `Hong Kong SAR (China)` <dbl>,
## #   India <dbl>, Indonesia <dbl>, Japan <dbl>, `South Korea` <dbl>,
## #   Kuwait <dbl>, Malaysia <dbl>, Myanmar <dbl>, Pakistan <dbl>,
## #   Philippines <dbl>, `Saudi Arabia` <dbl>, `Sri Lanka` <dbl>, Taiwan <dbl>,
## #   Thailand <dbl>, `United Arab Emirates` <dbl>, Vietnam <dbl>,
## #   `Belgium & Luxembourg` <dbl>, Finland <dbl>, France <dbl>, Germany <dbl>,
## #   Italy <dbl>, Netherlands <dbl>, Spain <dbl>, Switzerland <dbl>, …
```

Conventional base `ts` object versus `tibble` object

ts object

```
ts_data_ts <- ts(ts_data)
head(ts_data_ts)
```

```
##      Month-Year Republic of South Africa Canada   USA Bangladesh Brunei China
## [1,]      13879                     3680   6972 31155       6786   3729 79599
## [2,]      13910                     1662   6056 27738       6314   3070 82074
## [3,]      13939                     3394   6220 31349       7502   4805 72546
## [4,]      13970                     3337   4764 26376       7333   3096 76112
## [5,]      14000                     2089   4460 26788       7988   3586 64808
## [6,]      14031                     2515   3888 29725       8301   5284 55238
##      Hong Kong SAR (China) India Indonesia Japan South Korea Kuwait Malaysia
## [1,]                 17103 41639     62683 37673       27937    284    31352
## [2,]                 21089 37170     47834 35297       22633    241    35030
## [3,]                 23230 44815     64688 42575       22876    206    37629
## [4,]                 17688 49527     58074 26839       20634    193    37521
## [5,]                 19340 67754     57089 30814       22785    140    38044
## [6,]                 19152 57380     70118 31001       22575    354    40419
##      Myanmar Pakistan Philippines Saudi Arabia Sri Lanka Taiwan Thailand
## [1,]    5269     1395       18622          406      5289  13757    18370
## [2,]    4643     1027       21609          591      4767  13921    16400
## [3,]    6218     1635       28464          626      4988  11181    23387
## [4,]    7324     1232       30131          644      7639  11665    24469
```

Converting `tibble` object to `tsibble` object

Built on top of the tibble, a **tsibble** (or tbl_ts) is a data- and model-oriented object. Compared to the conventional time series objects in R, for example ts, zoo, and xts, the tsibble preserves time indices as the essential data column and makes heterogeneous data structures possible. Beyond the tibble-like representation, key comprised of single or multiple variables is introduced to uniquely identify observational units over time (index).

The code chunk below converting ts_data from tibble object into tsibble object by using `as_tsibble()` of **tsibble** R package.

```
ts_tsibble <- ts_data %>%
  mutate(Month = yearmonth(`Month-Year`)) %>%
  as_tsibble(index = `Month`)
```

What can we learn from the code chunk above?

- `mutate()` of **dplyr** package is used to derive a new field by transforming the data values in Month-Year field into month-year format. The transformation is performed by using `yearmonth()` of **tsibble** package.
- `as_tsibble()` is used to convert the tibble data frame into tsibble data frame.

tsibble object

```
ts_tsibble
```

```
## # A tsibble: 144 x 35 [1M]
##    `Month-Year` `Republic of South Africa` Canada   USA Bangladesh Brunei China
##    <date>                            <dbl>  <dbl> <dbl>      <dbl>  <dbl> <dbl>
##  1 2008-01-01                         3680   6972 31155       6786   3729 79599
##  2 2008-02-01                         1662   6056 27738       6314   3070 82074
##  3 2008-03-01                         3394   6220 31349       7502   4805 72546
##  4 2008-04-01                         3337   4764 26376       7333   3096 76112
##  5 2008-05-01                         2089   4460 26788       7988   3586 64808
##  6 2008-06-01                         2515   3888 29725       8301   5284 55238
##  7 2008-07-01                         2919   5313 33183       9004   4070 80747
##  8 2008-08-01                         2471   4519 27427       7913   4183 66625
##  9 2008-09-01                         2492   3421 21588       7549   3160 52649
## 10 2008-10-01                         3023   4756 25112       7527   2983 54423
## # … with 134 more rows, and 28 more variables: `Hong Kong SAR (China)` <dbl>,
## #   India <dbl>, Indonesia <dbl>, Japan <dbl>, `South Korea` <dbl>,
## #   Kuwait <dbl>, Malaysia <dbl>, Myanmar <dbl>, Pakistan <dbl>,
## #   Philippines <dbl>, `Saudi Arabia` <dbl>, `Sri Lanka` <dbl>, Taiwan <dbl>,
## #   Thailand <dbl>, `United Arab Emirates` <dbl>, Vietnam <dbl>,
## #   `Belgium & Luxembourg` <dbl>, Finland <dbl>, France <dbl>, Germany <dbl>,
## #   Italy <dbl>, Netherlands <dbl>, Spain <dbl>, Switzerland <dbl>, …
```

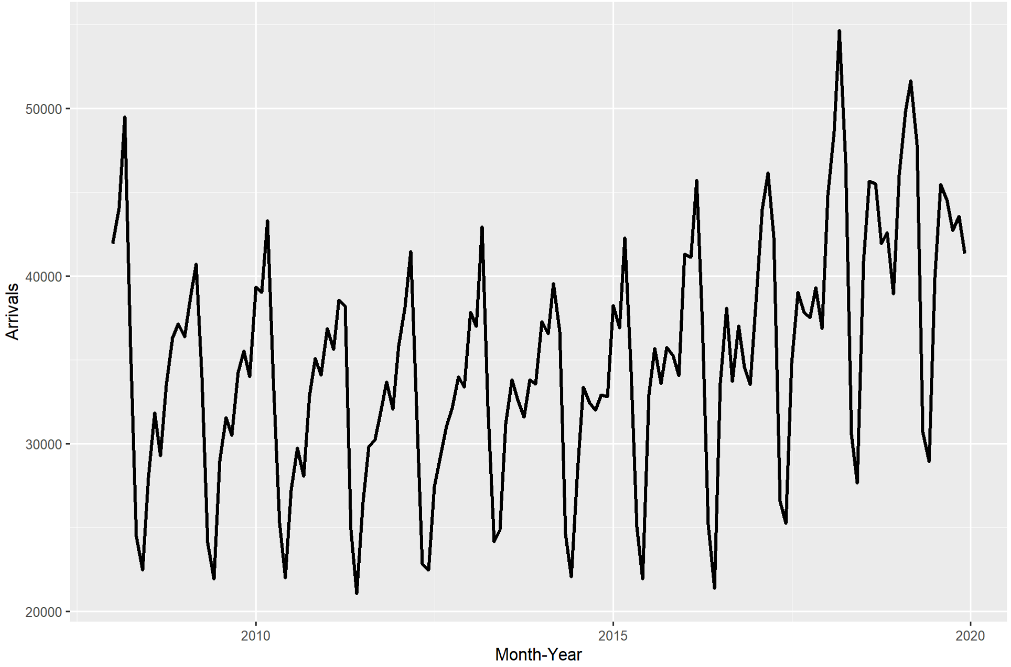
Visualising Time-series Data

In order to visualise the time-series data effectively, we need to organise the data frame from wide to long format by using `pivot_longer()` of **tidyr** package as shown below.

```r
ts_longer <- ts_data %>%
  pivot_longer(cols = c(2:34),
               names_to = "Country",
               values_to = "Arrivals")
```

Visualising single time-series: ggplot2 methods

```r
ts_longer %>%
  filter(Country == "Vietnam") %>%
  ggplot(aes(x = `Month-Year`,
             y = Arrivals))+
  geom_line(size = 1)
```



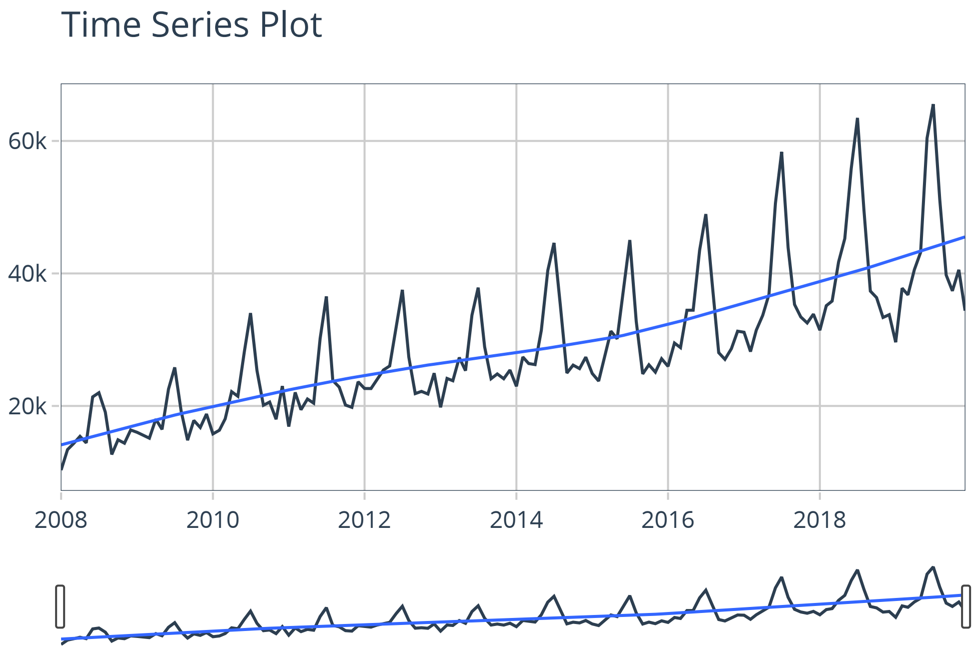
What can we learn from the code chunk above?

- `filter()` of **dplyr** package is used to select records belong to Vietnam.
- `geom_line()` of **ggplot2** package is used to plot the time-series line graph.

Visualising single time-series: timetk methods

In the code chunk below, `plot_time_series()` of **timetk** package is used to plot the time series line graph.

```
ts_longer %>%
  filter(Country == "United Kingdom") %>%
  plot_time_series(`Month-Year`, Arrivals,
                   .line_size = 0.4,
                   .smooth_size = 0.4,
                   .interactive = TRUE,
                   .plotly_slider = TRUE)
```



Plotting time-series data: ggplot2 methods

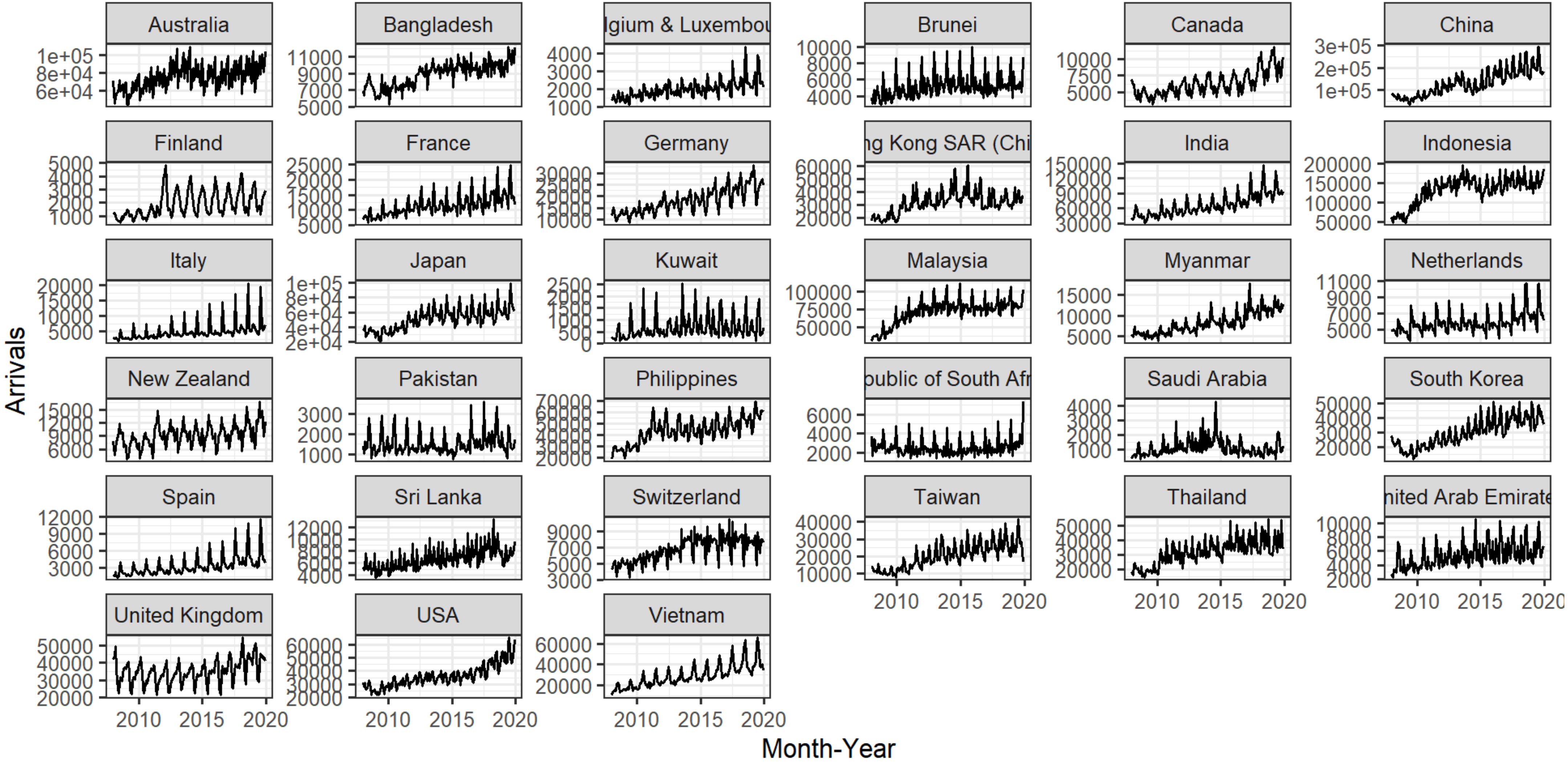
```r
ggplot(data = ts_longer,
       aes(x = `Month-Year`,
           y = Arrivals,
           color = Country))+
  geom_line(size = 0.5)
```

Visualising multiple time-series with trellis plot: ggplot2 methods

In order to provide effective comparison, `facet_wrap()` of **ggplot2** package is used to create small multiple line graph also known as trellis plot.

```
ggplot(data = ts_longer,
       aes(x = `Month-Year`,
           y = Arrivals))+
  geom_line(size = 0.5) +
  facet_wrap(~ Country,
             scales = "free_y") +
  theme_bw()
```

Visualising multiple time-series with trellis plot: ggplot2 methods

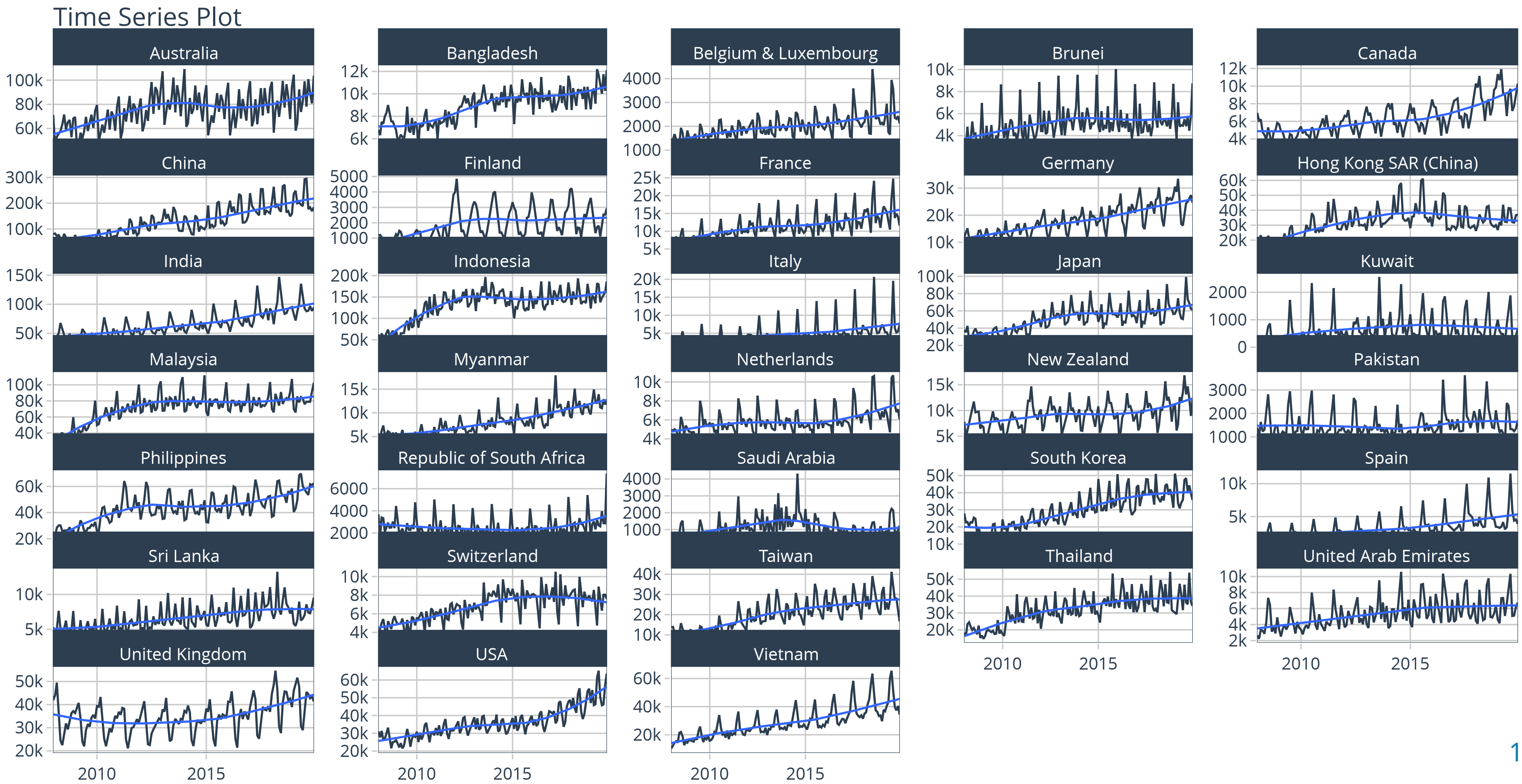


Visualising multiple time-series with interactive trellis plot: timetk methods

In the code chunk below, `plot_time_series()` of timetk package is used to prepare the trellis line graphs.

```r
ts_longer %>%
  group_by(Country) %>%
  plot_time_series(
    `Month-Year`, Arrivals,
    .line_size = 0.4,
    .facet_ncol = 5,
    .facet_scales = "free_y",
    .interactive = TRUE,
    .smooth_size = 0.4)
```

Visualising multiple time-series with interactive trellis plot: timetk methods

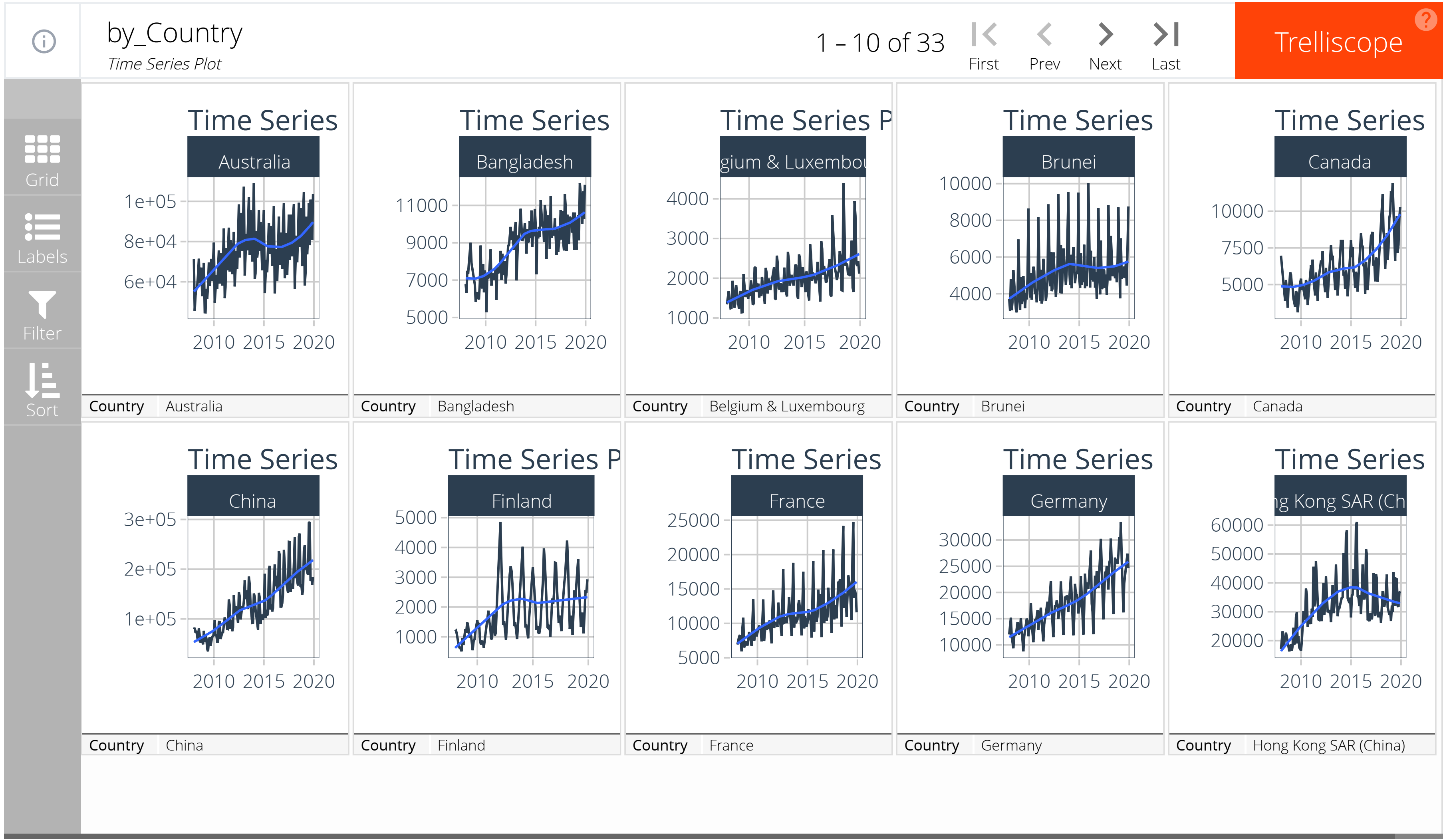


Visualising multiple time-series with interactive trellis plot: timetk methods

Beside using **plotly R** to plot interactive trellis graphs, timetk also support **trelliscopejs**.

```
ts_longer %>%
  group_by(Country) %>%
  plot_time_series(
    `Month-Year`, Arrivals,
    .line_size = 0.4,
    .facet_ncol = 5,
    .facet_nrow = 2,
    .facet_scales = "free_y",
    .interactive = TRUE,
    .smooth_size = 0.4,
    .trelliscope = TRUE,
    .trelliscope_params = list(
      width = 600,
      height = 700,
      path= "trellis/")
    )
```

Visualising multiple time-series with interactive trellis plot: timetk methods



Visual Analysis of Seasonality

- Time series datasets can contain a seasonal component.

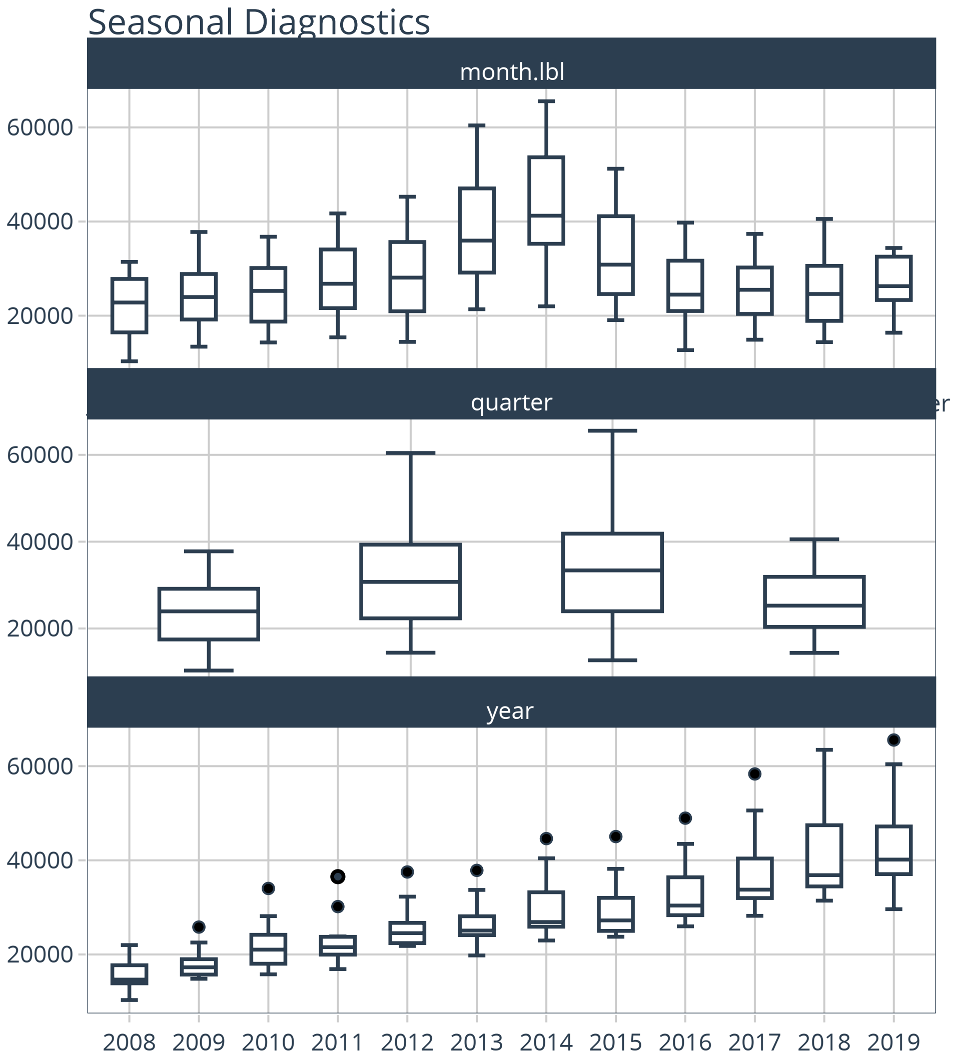
- This is a cycle that repeats over time, such as monthly or yearly. This repeating cycle may obscure the signal that we wish to model when forecasting, and in turn may provide a strong signal to our predictive models.

In this section, you will discover how to identify seasonality in time series data by using functions provides by **timetk** and **feasts** packages.

Visual Analysis of Seasonality: timetk methods

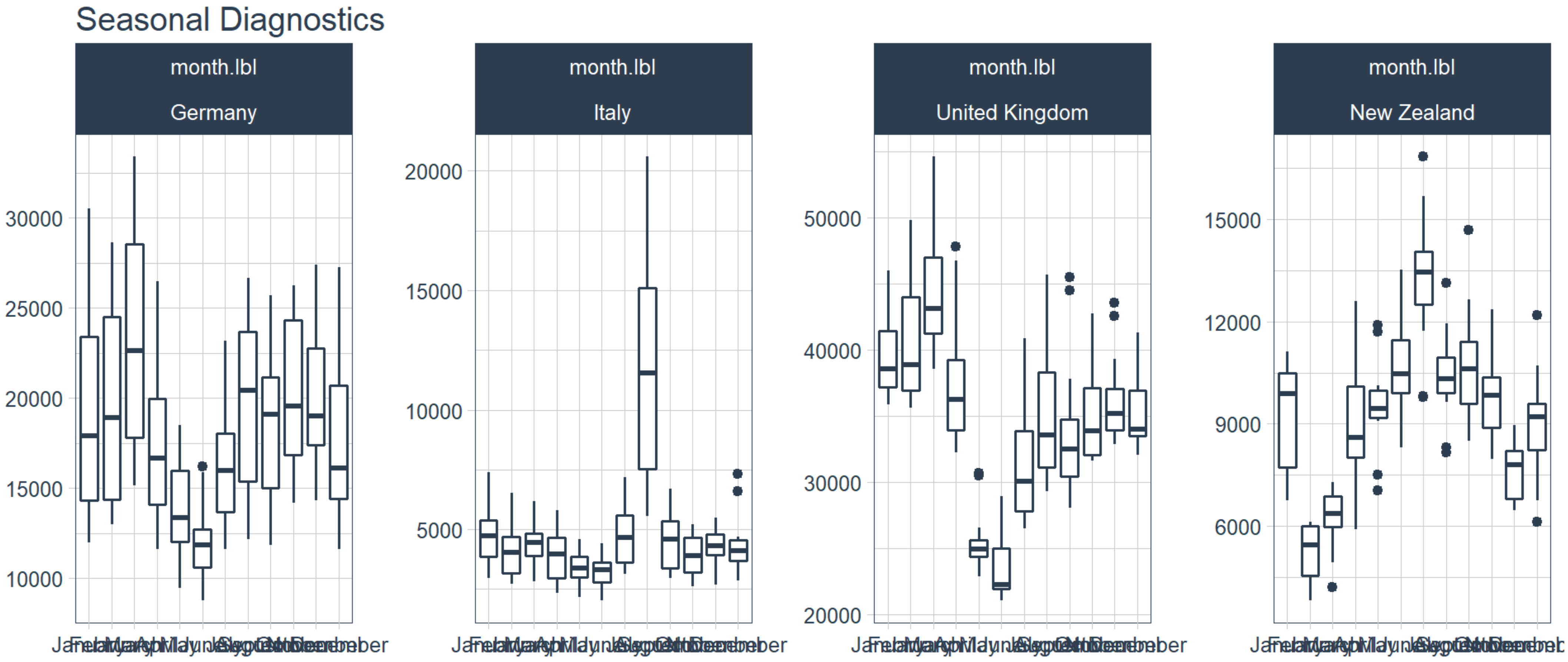
In the code chunk below, `plot_seasonal_diagnostics()` of **timetk** package is used to detect seasonal patterns visually.

```
ts_longer %>%
  filter(Country == "Vietnam") %>%
  plot_seasonal_diagnostics(
    `Month-Year`, Arrivals,
    .interactive = TRUE)
```



Visual Analysis of Seasonality: timetk methods

`plot_seasonal_diagnostics()` of **timetk** package can also be used to detect seasonal patterns of multiple time series visually.



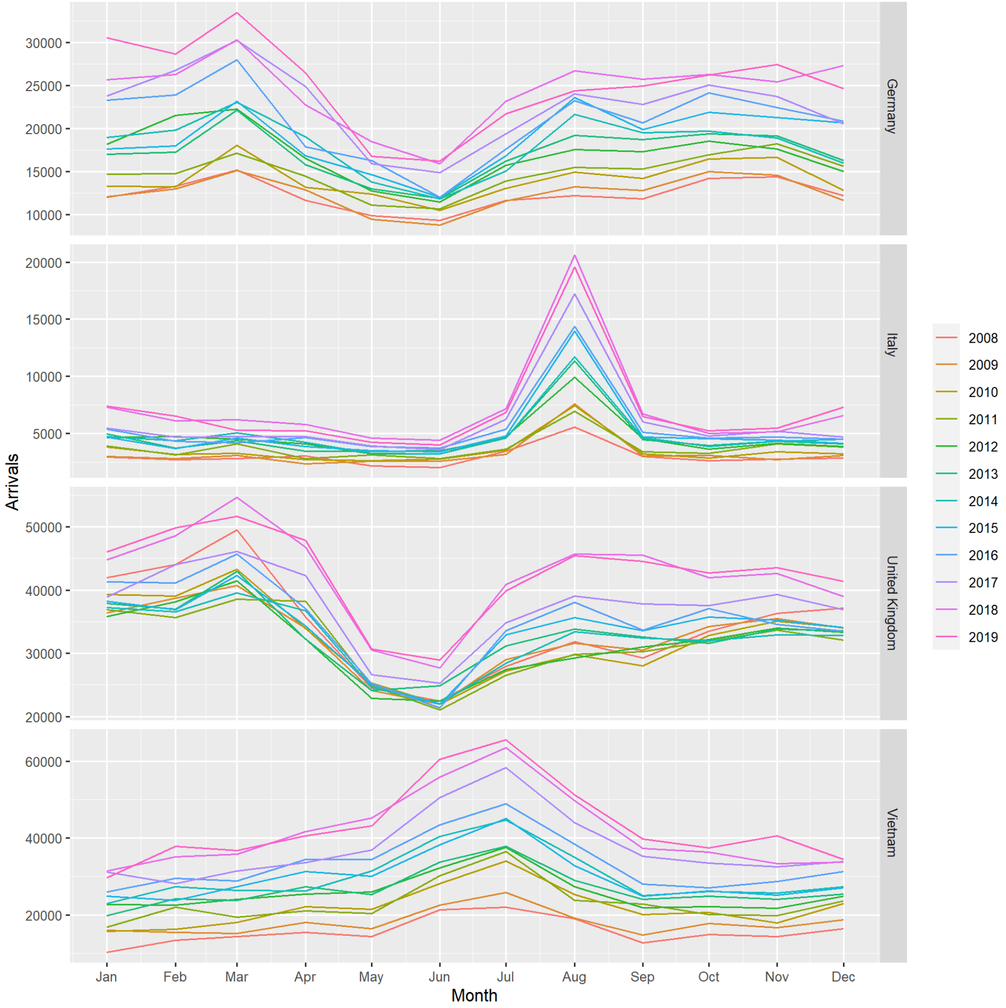
Visual Analysis of Seasonality: timetk methods

Below is the code chunk used to prepare the seasonal detection plots on previous page.

Visual Analysis of Seasonality: feasts methods

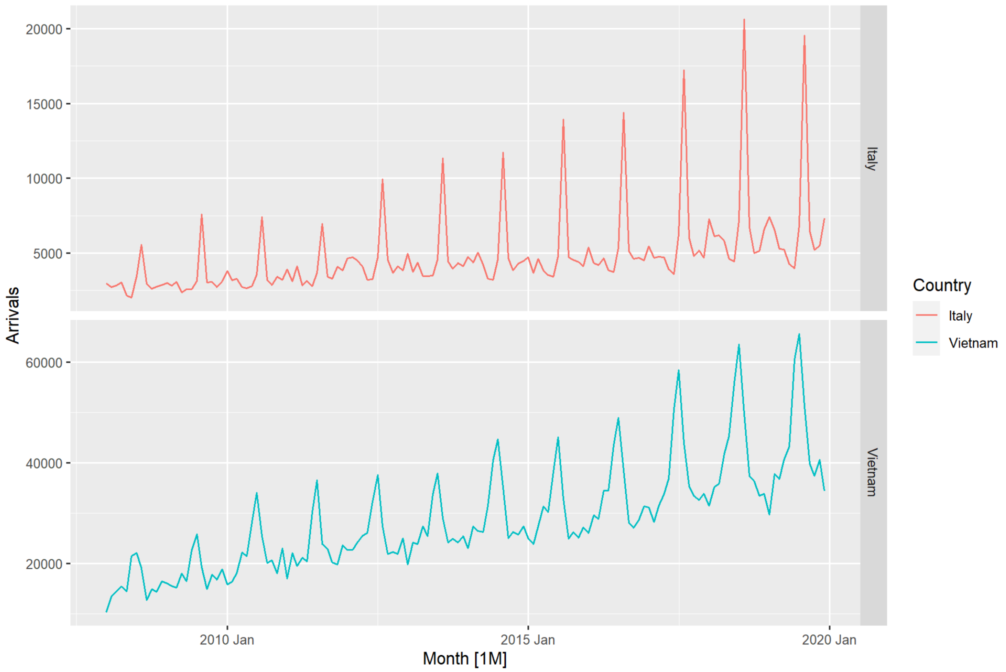
A season plot is created by using `gg_season()` of **feasts** package.

```
tsibble_longer %>%
  filter(Country == "Italy" |
         Country == "Vietnam" |
         Country == "United Kingdom" |
         Country == "Germany") %>%
  gg_season(Arrivals)
```



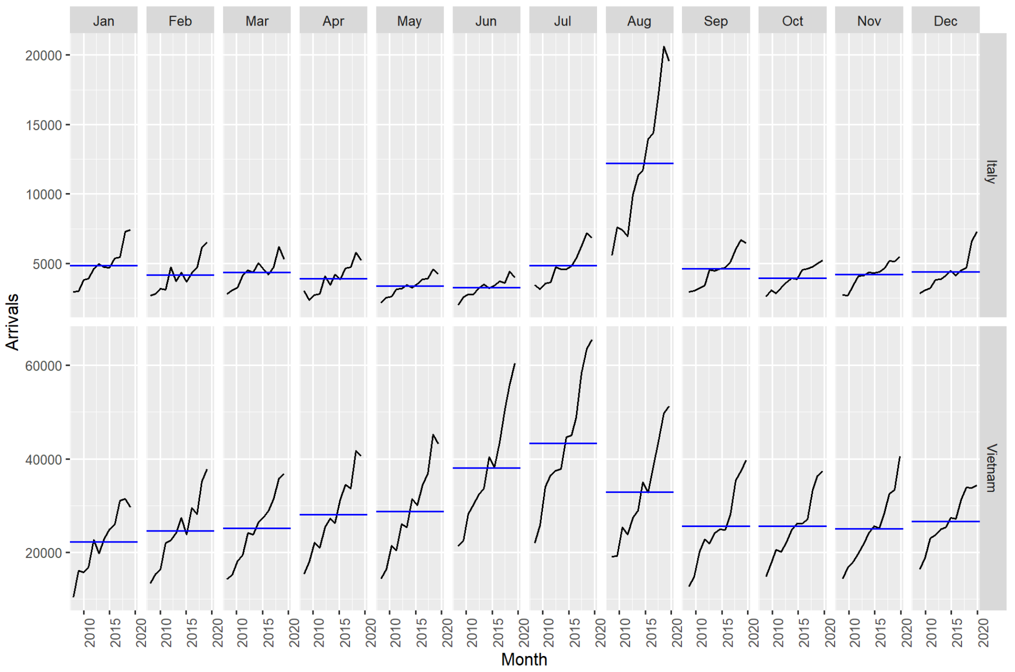
Visual Analysis of Seasonality: feasts methods

```
tsibble_longer %>%
  filter(Country == "Vietnam" |
           Country == "Italy") %>%
  autoplot(Arrivals) +
  facet_grid(Country ~ ., scales = "free_y")
```



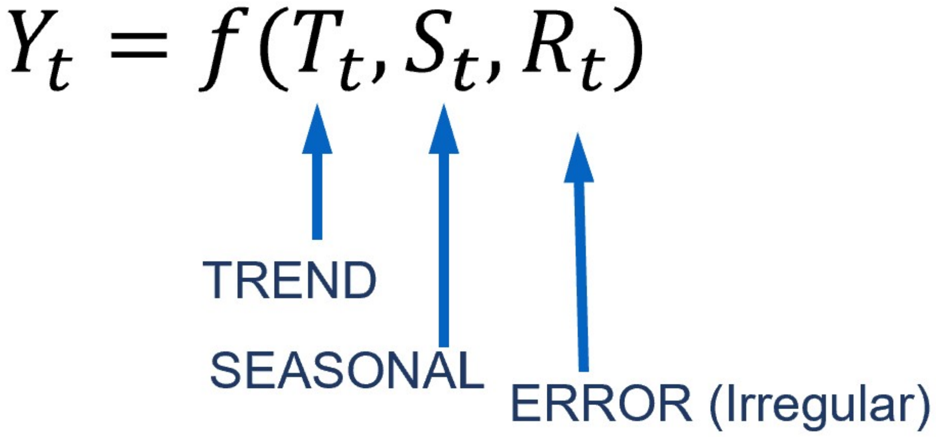
Cycle plot using `gg_subseries()` of feasts package.

```
tsibble_longer %>%
  filter(Country == "Vietnam" |
           Country == "Italy") %>%
  gg_subseries(Arrivals)
```



Time series decomposition

Time series decomposition allows us to isolate structural components such as trend and seasonality from the time-series data.



$$Y_t = f(T_t, S_t, R_t)$$

TRENDS

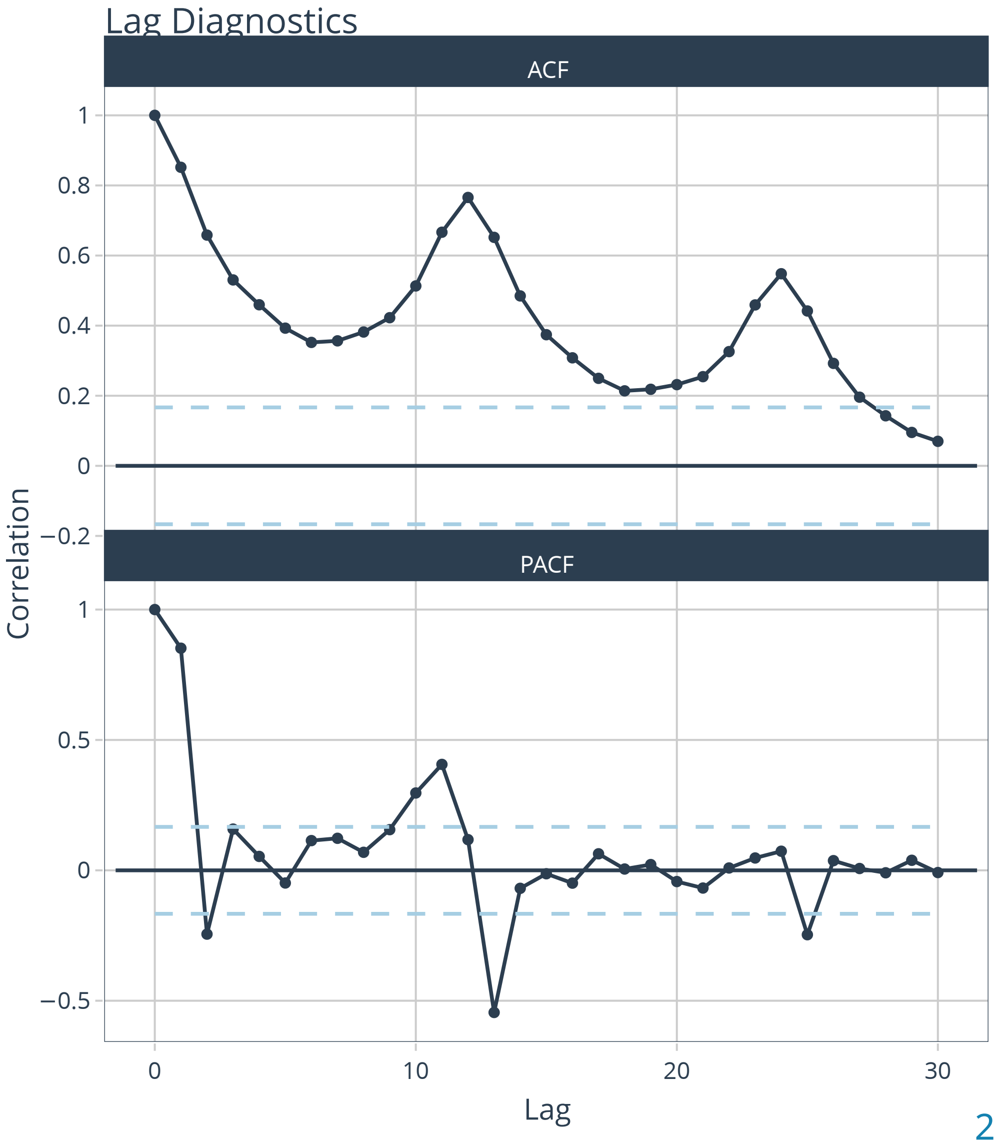
SEASONAL

ERROR (Irregular)

Time series decomposition: timetk methods

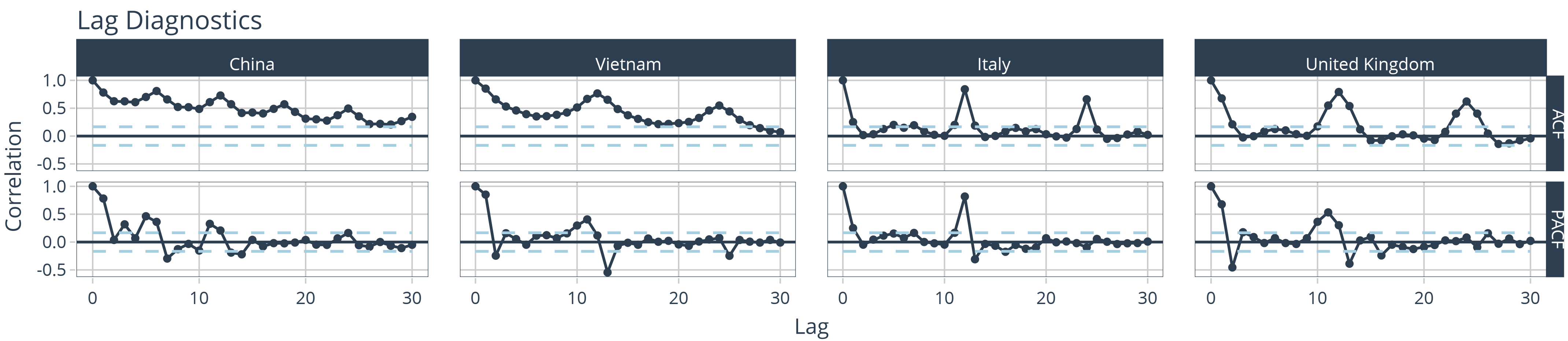
In the code chunk below, `plot_acf_diagnostics()` is used to decompose the visitor arrival from Vietnam by air data. The function Return the ACF and PACF of a target and optionally CCF's of one or more lagged predictors in interactive plotly plots.

```
ts_longer %>%
  filter(Country == "Vietnam") %>%
  plot_acf_diagnostics(
    `Month-Year`, Arrivals,
    .lags = "30 months",
    .interactive = TRUE
    )
```



Multiple Time series decomposition: timetk methods

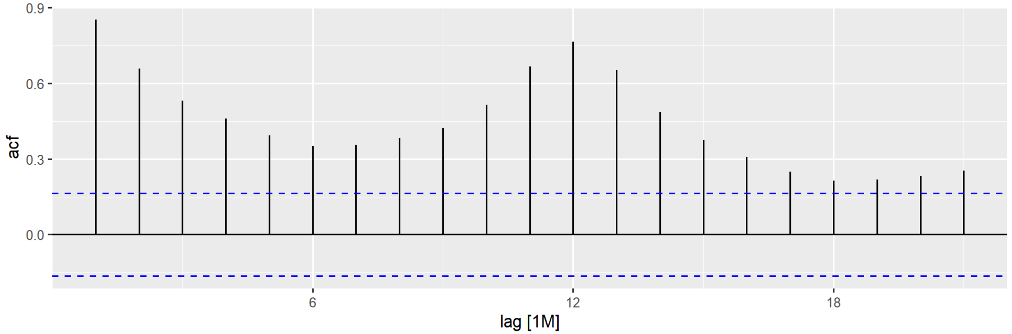
Code chunk below is used to prepare a trellis plot of ACFs for visitor arrivals from Vietnam, Italy, United Kingdom and China.



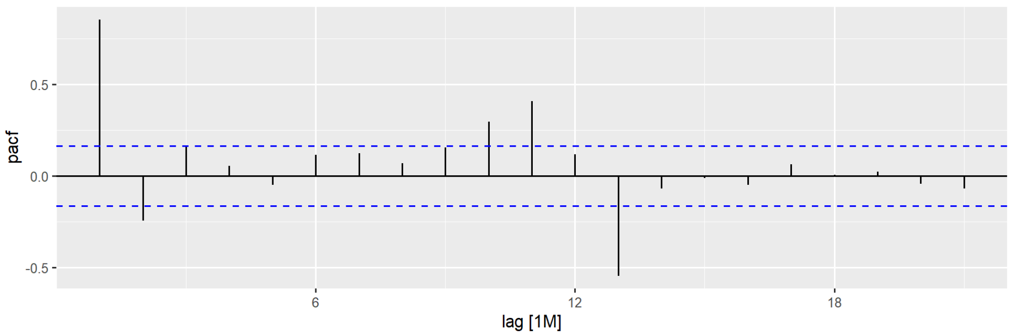
Time series decomposition: feasts methods

In **feasts** package, time series decomposition is supported by `ACF()`, `PACF()`, `CCF()`, `feat_acf()`, and `feat_pacf()`. The output can then be plotted by using `autoplot()` of **feasts** package.

```r
tsibble_longer %>%
  filter(`Country` == "Vietnam") %>%
  ACF(Arrivals) %>%
  autoplot()
```

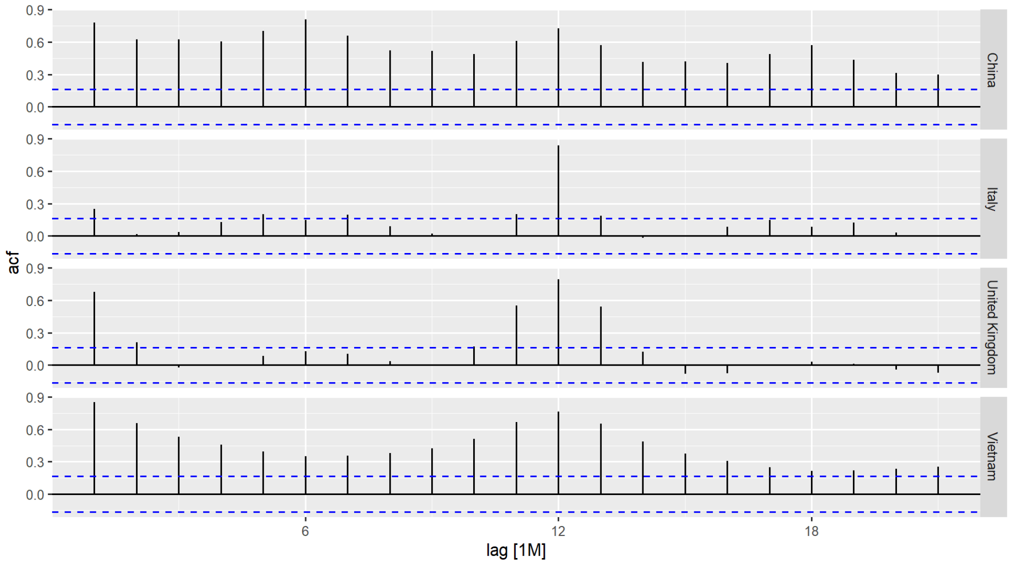


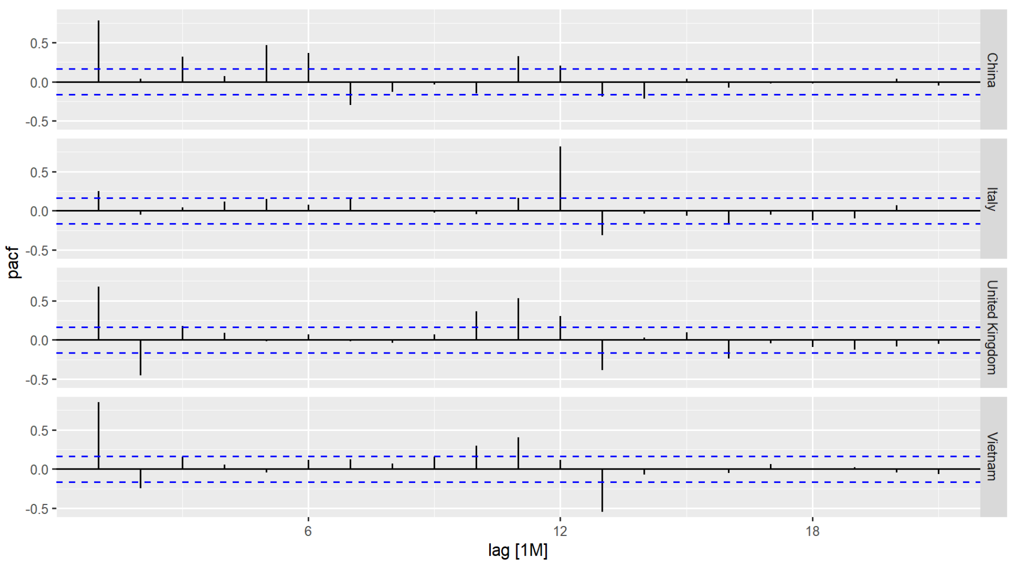
```r
tsibble_longer %>%
  filter(`Country` == "Vietnam") %>%
  PACF(Arrivals) %>%
  autoplot()
```



Time series decomposition: feasts methods

Code chunk below is used to prepare a trellis plot of ACFs for visitor arrivals from Vietnam, Italy, United Kingdom and China.

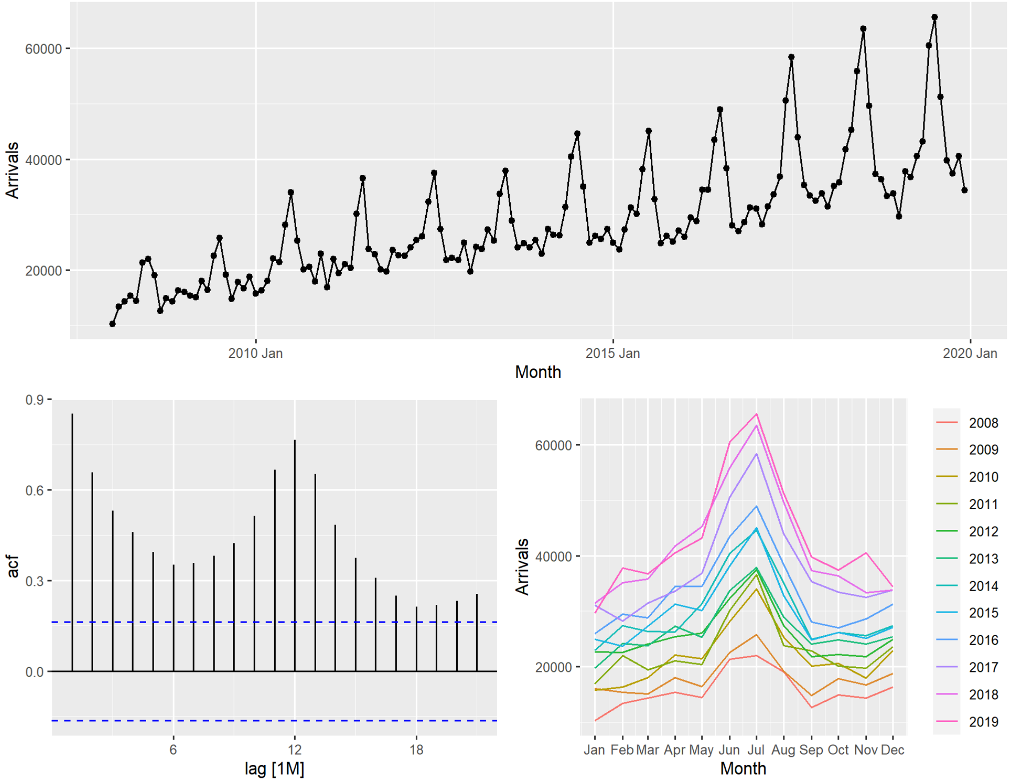




Composite plot of time series: feasts methods

One of the interesting function of feasts package time series decomposition is `gg_tsdisplay()`. It provides a composite plot by showing the original line graph on the top pane follow by the ACF on the left and seasonal plot on the right.

```
tsibble_longer %>%
  filter(`Country` == "Vietnam") %>%
  gg_tsdisplay(Arrivals)
```



STL Diagnostics

STL is an acronym for “Seasonal and Trend decomposition using Loess”, while Loess is a method for estimating nonlinear relationships. The STL method was developed by R. B. Cleveland, Cleveland, McRae, & Terpenning (1990).

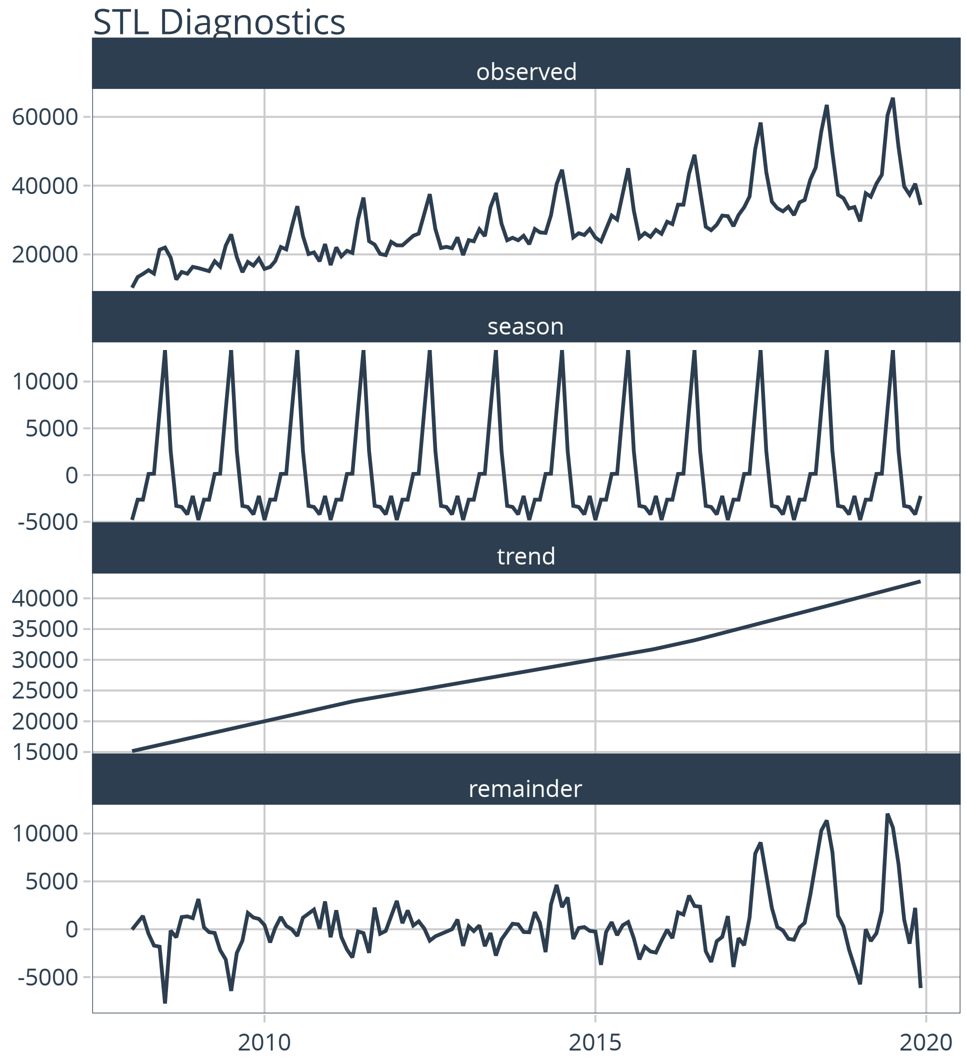
STL has several advantages over the classical, SEATS and X11 decomposition methods:

- Unlike SEATS and X11, STL will handle any type of seasonality, not only monthly and quarterly data.
- The seasonal component is allowed to change over time, and the rate of change can be controlled by the user.
- The smoothness of the trend-cycle can also be controlled by the user.
- It can be robust to outliers (i.e., the user can specify a robust decomposition), so that occasional unusual observations will not affect the estimates of the trend-cycle and seasonal components. They will, however, affect the remainder component.

STL Diagnostics: timetk methods

In the code chunk below, `plot_stl_diagnostics()` of timetk package is used to perform STL diagnostics.

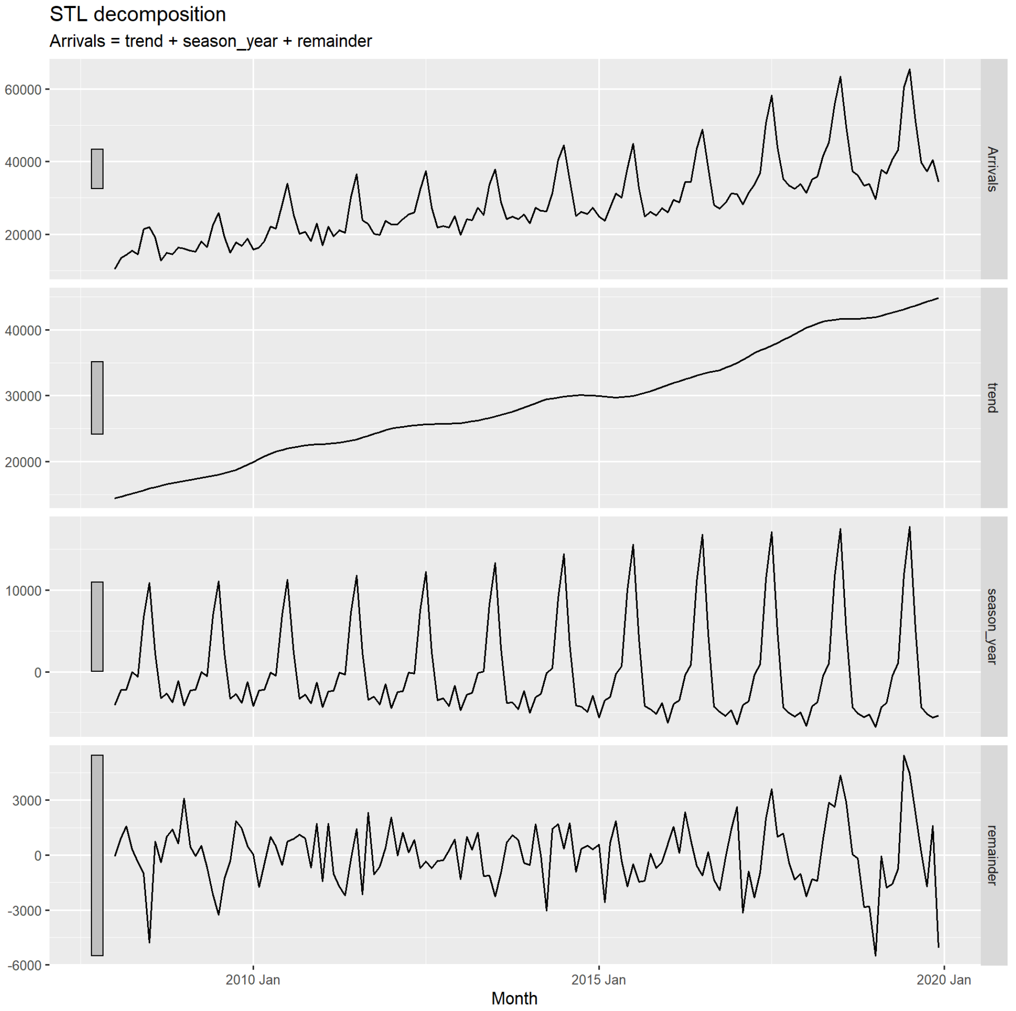
```
ts_longer %>%
  filter(Country == "Vietnam") %>%
  plot_stl_diagnostics(
    `Month-Year`, Arrivals,
    .frequency = "auto",
    .trend = "auto",
    .feature_set = c(
     "observed", "season",
     "trend", "remainder"),
    .interactive = TRUE)
```



STL Diagnostics: feasts methods

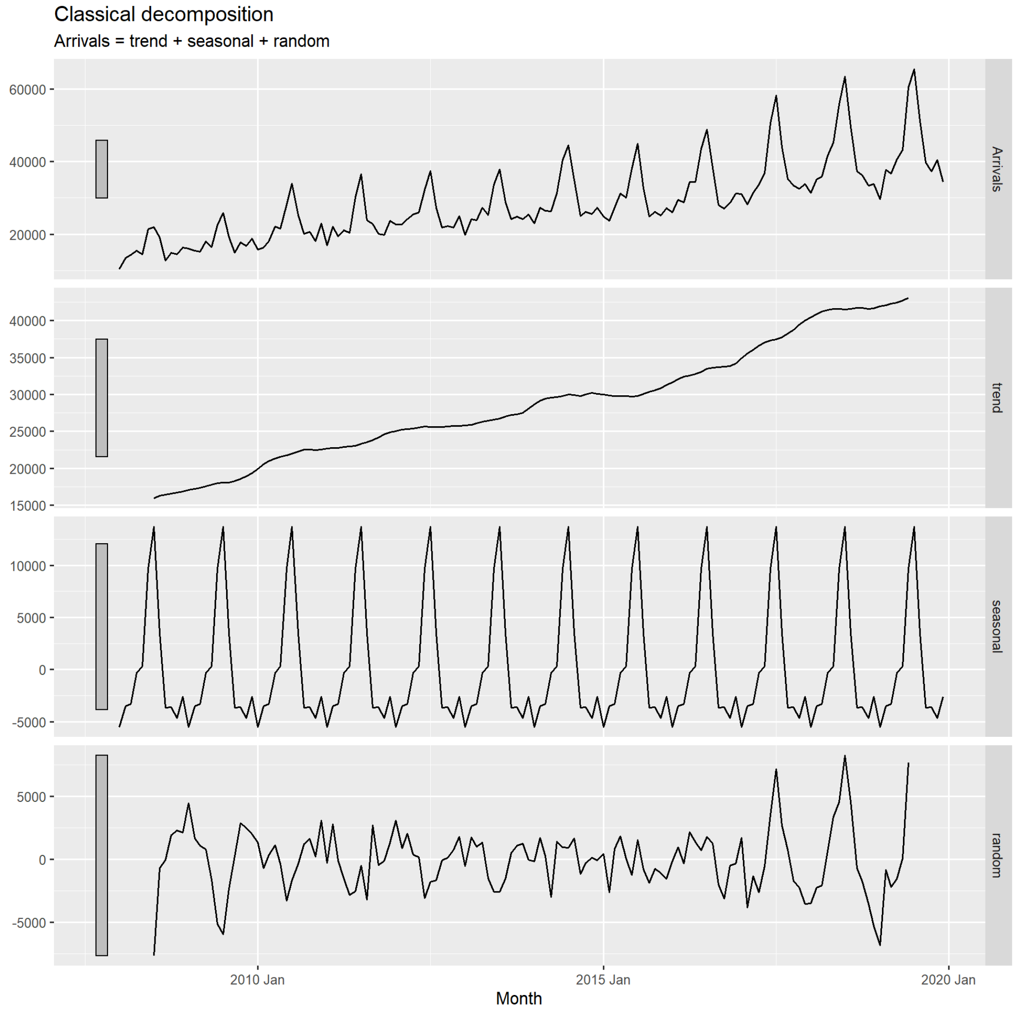
```
tsaPlot <- tsibble_longer %>%
  filter(`Country` == "Vietnam") %>%
  model(stl = STL(Arrivals)) %>%
  components() %>%
  autoplot()
```

The grey bars to the left of each panel show the relative scales of the components. Each grey bar represents the same length but because the plots are on different scales, the bars vary in size. The large grey bar in the bottom panel shows that the variation in the remainder component is smallest compared to the variation in the data. If we shrank the bottom three panels until their bars became the same size as that in the data panel, then all the panels would be on the same scale.



Classical Decomposition: feasts methods

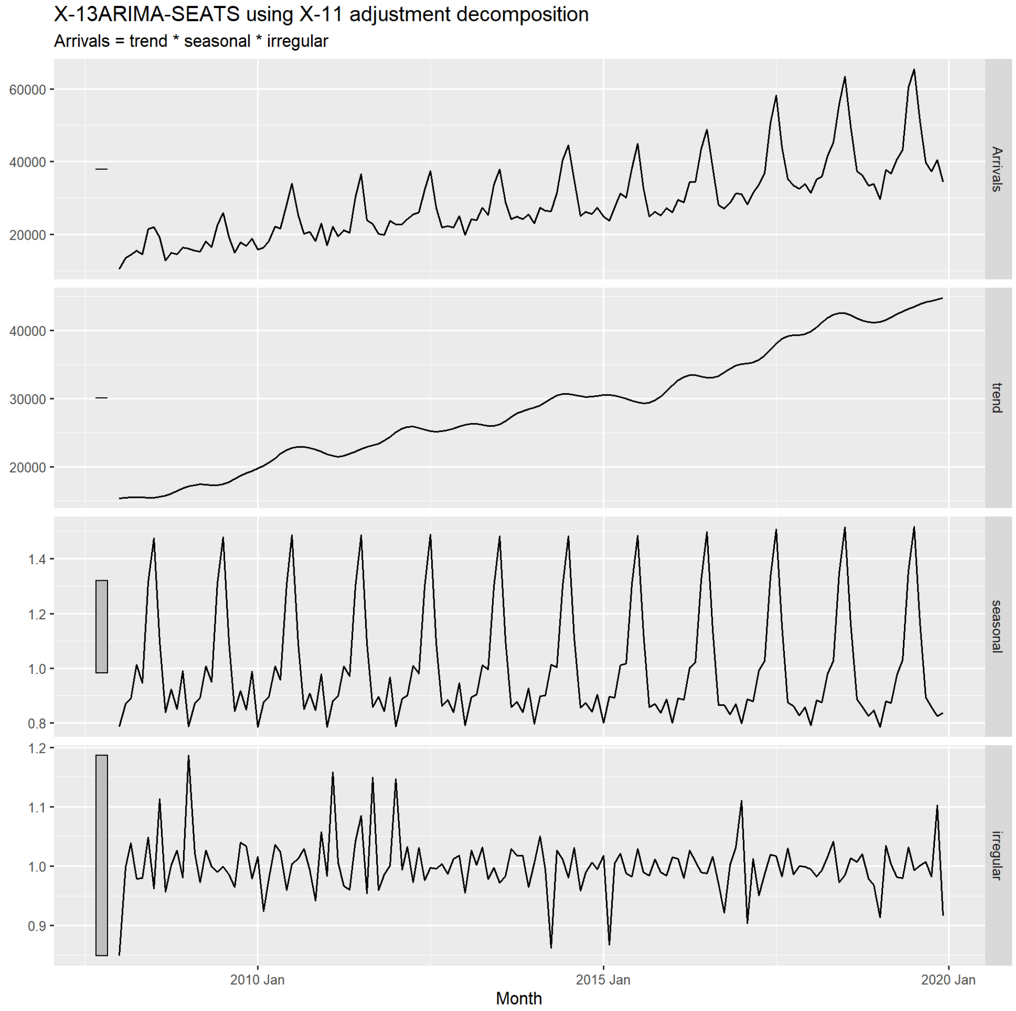
```
tsibble_longer %>%
  filter(`Country` == "Vietnam") %>%
  model(
    classical_decomposition(
      Arrivals, type = "additive")) %>%
  components() %>%
  autoplot()
```



X11 Decomposition: feasts methods

The X-11 method originated in the US Census Bureau and was further developed by Statistics Canada. It is based on classical decomposition, but includes many extra steps and features in order to overcome the drawbacks of classical decomposition that were discussed in the previous section. The process is entirely automatic and tends to be highly robust to outliers and level shifts in the time series. The details of the X-11 method are described in Dagum & Bianconcini (2016).

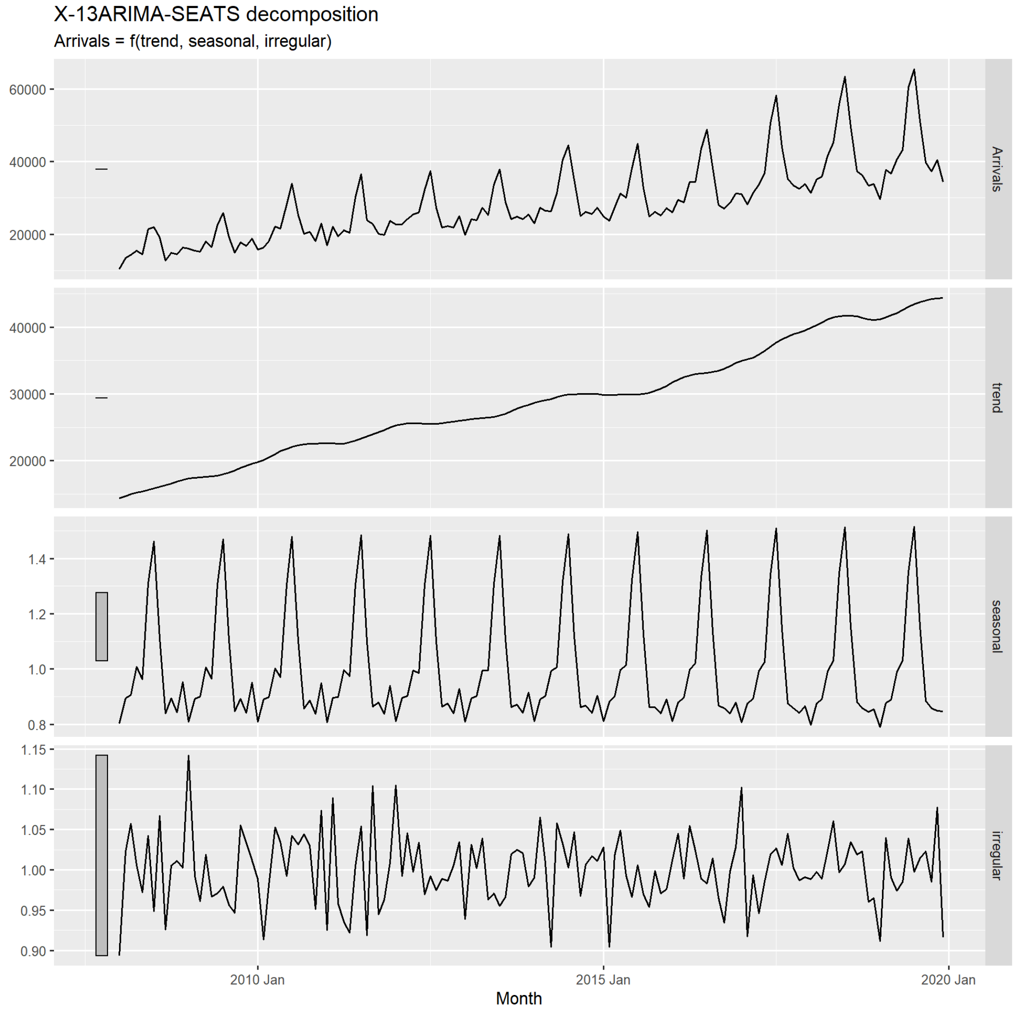
```
tsibble_longer %>%
  filter(`Country` == "Vietnam") %>%
  model(x11 = X_13ARIMA_SEATS(
    Arrivals ~ x11())) %>%
  components() %>%
  autoplot()
```



SEATS Decomposition: feasts methods

SEATS stands for *Seasonal Extraction in ARIMA Time Series*. This procedure was developed at the Bank of Spain, and is now widely used by government agencies around the world. A complete discussion of the method is available in Dagum & Bianconcini (2016).

```
tsibble_longer %>%
  filter(`Country` == "Vietnam") %>%
  model(seats = X_13ARIMA_SEATS
        (Arrivals ~ seats())) %>%
  components() %>%
  autoplot()
```



A Little Bonus for the Day

Step 1: Installing `openxlsx` package and load it onto R environment.

Step 2: Creating a workbook

```
wb <- createWorkbook()
```

Step 3: Creating a worksheet

```
addWorksheet(wb,
             sheetName = "Time Series Analysis
```

Step 4: Adding a plot

```
print(tsaPlot)

wb %>% insertPlot(
  sheet = "Time Series Analysis",
  startCol = "G",
  startRow = 3)
```

Step 5: Adding Data Table

```
Vietnam <- tsibble_longer %>%
  filter(`Country` == "Vietnam")

writeDataTable(wb,
               sheet = "Time Series Analysis",
               x = Vietnam)
```

Step 6: Saving the Workbook

```
saveWorkbook(wb, "data/tsa.xlsx",
             overwrite = TRUE)
```

Reference

Rob J Hyndman and George Athanasopoulos (2022) **Forecasting: Principles and Practice (3rd ed)**, online version.