ENV 797 - Time Series Analysis for Energy and Environment Applications | Spring 2026

Assignment 5 - Due date 02/17/26

Loo Si Min

Directions

You should open the .rmd file corresponding to this assignment on RStudio. The file is available on our class repository on Github.

Once you have the file open on your local machine the first thing you will do is rename the file such that it includes your first and last name (e.g., "LuanaLima_TSA_A05_Sp26.Rmd"). Then change "Student Name" on line 3 with your name.

Then you will start working through the assignment by **creating code and output** that answer each question. Be sure to use this assignment document. Your report should contain the answer to each question and any plots/tables you obtained (when applicable).

When you have completed the assignment, **Knit** the text and code into a single PDF file. Submit this pdf using Canvas.

R packages needed for this assignment: "readxl", "ggplot2", "forecast","tseries", and "Kendall". Install these packages, if you haven't done yet. Do not forget to load them before running your script, since they are NOT default packages.\

```
#Load/install required package here
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(tseries)
library(ggplot2)
library(Kendall)
```

```
## Warning: package 'Kendall' was built under R version 4.4.3
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(tidyverse)  #load this package so yon clean the data frame using pipes
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr   1.1.4     v stringr 1.5.1
```

```
## v forcats 1.0.0     v tibble  3.2.1
## v purrr   1.0.2     v tidyr   1.3.1
## v readr   2.1.5
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(readxl)
```

Consider the same data you used for A04 from the spreadsheet "Table_10.1_Renewable_Energy_Production_and_Consumptio
The data comes from the US Energy Information and Administration and corresponds to the December 2025
Monthly Energy Review.

```
#Importing data set - using readxl package
energy_data <- read_excel(
  path="../Data/Table_10.1_Renewable_Energy_Production_and_Consumption_by_Source.xlsx",
  skip = 12,
  sheet="Monthly Data",
  col_names=FALSE
  )
```

```
## New names:
## * `` -> `...1`
## * `` -> `...2`
## * `` -> `...3`
## * `` -> `...4`
## * `` -> `...5`
## * `` -> `...6`
## * `` -> `...7`
## * `` -> `...8`
## * `` -> `...9`
## * `` -> `...10`
## * `` -> `...11`
## * `` -> `...12`
## * `` -> `...13`
## * `` -> `...14`
```

```
#Now let's extract the column names from row 11 only
read_col_names <- read_excel(
  path="../Data/Table_10.1_Renewable_Energy_Production_and_Consumption_by_Source.xlsx",
  skip = 10,
  n_max = 1,
  sheet="Monthly Data",
  col_names=FALSE
  )
```

```
## New names:
## * `` -> `...1`
## * `` -> `...2`
## * `` -> `...3`
## * `` -> `...4`
## * `` -> `...5`
## * `` -> `...6`
## * `` -> `...7`
## * `` -> `...8`
```

```
## * `` -> `...9`
## * `` -> `...10`
## * `` -> `...11`
## * `` -> `...12`
## * `` -> `...13`
## * `` -> `...14`
```

```
colnames(energy_data) <- read_col_names
nobs <- nrow(energy_data)

nobs=nrow(energy_data)
nvar=ncol(energy_data)

head(energy_data)
```

```
## # A tibble: 6 x 14
##   Month               `Wood Energy Production` `Biofuels Production`
##   <dttm>                                 <dbl> <chr>
## 1 1973-01-01 00:00:00                     130. Not Available
## 2 1973-02-01 00:00:00                     117. Not Available
## 3 1973-03-01 00:00:00                     130. Not Available
## 4 1973-04-01 00:00:00                     125. Not Available
## 5 1973-05-01 00:00:00                     130. Not Available
## 6 1973-06-01 00:00:00                     125. Not Available
## # i 11 more variables: `Total Biomass Energy Production` <dbl>,
## #   `Total Renewable Energy Production` <dbl>,
## #   `Hydroelectric Power Consumption` <dbl>,
## #   `Geothermal Energy Consumption` <dbl>, `Solar Energy Consumption` <chr>,
## #   `Wind Energy Consumption` <chr>, `Wood Energy Consumption` <dbl>,
## #   `Waste Energy Consumption` <dbl>, `Biofuels Consumption` <chr>,
## #   `Total Biomass Energy Consumption` <dbl>, ...
```

Handling Missing Data

Q1

Using the original dataset, create a new data frame that includes only the following variables: **Date, Solar Energy Consumption and Wind Energy Consumption**. Check the class of columns, you will see that they are stored are characters instead of numbers. Because solar generation begins later in the sample, the early observations are recorded as "Not Available". Convert the data to numeric, the "Not Available" will became NAs.

You may either filter out the "Not Available" rows and then convert the column to numeric or convert first and then remove missing values using drop_na() (or na.omit()). If you are comfortable using pipes for data wrangling, please do so.

Important: Note that we dropping the missing observations instead of interpolating is becasue they only happen in the beginning of the series!

```
library(dplyr)
library(stringr)
library(tidyr)

df_sw <- energy_data %>%
  dplyr::select(
    Date = Month,
    Solar_Cons = `Solar Energy Consumption`,
```

```
    Wind_Cons  = `Wind Energy Consumption`
  )

sapply(df_sw, class)
```

```
## $Date
## [1] "POSIXct" "POSIXt"
##
## $Solar_Cons
## [1] "character"
##
## $Wind_Cons
## [1] "character"
```

```
# Convert "Not Available" -> NA, then convert to numeric, then drop missing rows
df_sw_clean <- df_sw %>%
  mutate(
    Solar_Cons = na_if(as.character(Solar_Cons), "Not Available"),
    Wind_Cons  = na_if(as.character(Wind_Cons), "Not Available"),
    Solar_Cons = str_replace_all(Solar_Cons, ",", ""),
    Wind_Cons  = str_replace_all(Wind_Cons, ",", ""),
    Solar_Cons = as.numeric(Solar_Cons),
    Wind_Cons  = as.numeric(Wind_Cons)
  ) %>%
  drop_na(Solar_Cons, Wind_Cons)

# Confirm result
str(df_sw_clean)
```

```
## tibble [501 x 3] (S3: tbl_df/tbl/data.frame)
##  $ Date      : POSIXct[1:501], format: "1984-01-01" "1984-02-01" ...
##  $ Solar_Cons: num [1:501] 0 0 0.001 0.001 0.002 0.003 0.001 0.003 0.003 0.002 ...
##  $ Wind_Cons : num [1:501] 0 0.001 0.001 0.002 0.003 0.002 0.002 0.001 0.002 0.003 ...
```

```
head(df_sw_clean)
```

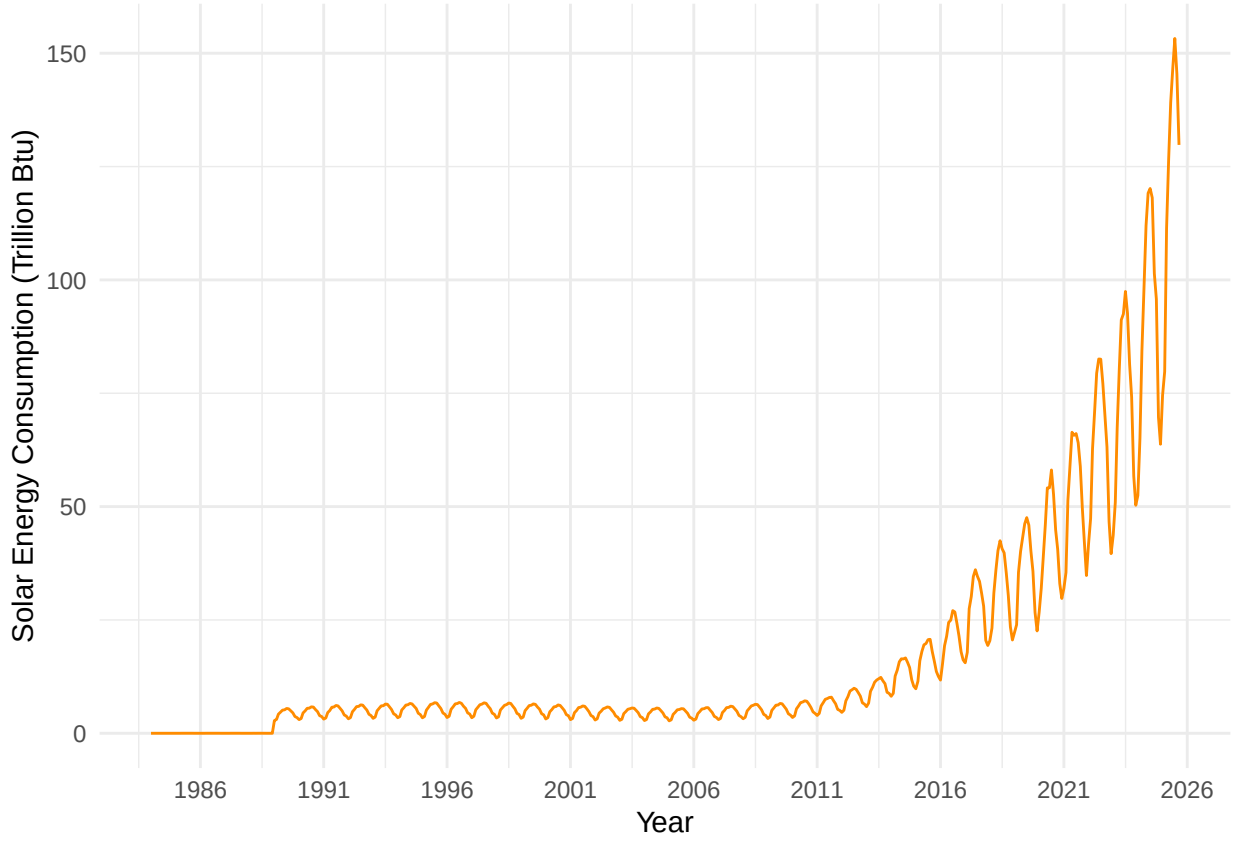
```
## # A tibble: 6 x 3
##   Date                Solar_Cons Wind_Cons
##   <dttm>                   <dbl>     <dbl>
## 1 1984-01-01 00:00:00      0         0
## 2 1984-02-01 00:00:00      0         0.001
## 3 1984-03-01 00:00:00      0.001     0.001
## 4 1984-04-01 00:00:00      0.001     0.002
## 5 1984-05-01 00:00:00      0.002     0.003
## 6 1984-06-01 00:00:00      0.003     0.002
```

Q2

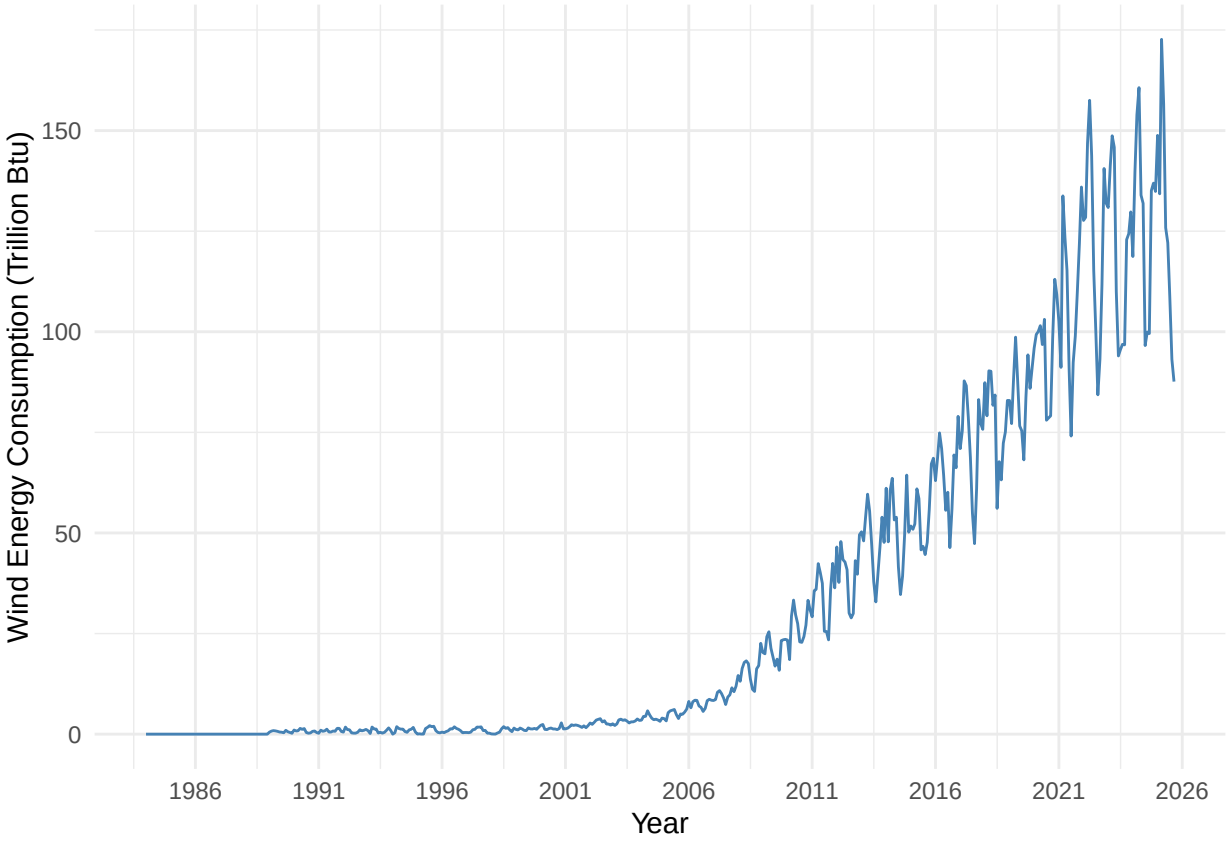
Plot the Solar and Wind energy consumption over time using ggplot. Plot each series on a separate graph. No need to add legend. Add informative names to the y axis using `ylab()`. Explore the function `scale_x_date()` on ggplot and see if you can change the x axis to improve your plot. Hint: use `scale_x_date(date_breaks = "5 years", date_labels = "%Y")")`

```
df_sw_clean <- df_sw_clean %>%
  mutate(Date2 = as.Date(Date))
```

```
ggplot(df_sw_clean, aes(x = Date2, y = Solar_Cons)) +
  geom_line(color = "darkorange") +
  ylab("Solar Energy Consumption (Trillion Btu)") +
  xlab("Year") +
  scale_x_date(date_breaks = "5 years",
               date_labels = "%Y") +
  theme_minimal()
```




```
ggplot(df_sw_clean, aes(x = Date2, y = Wind_Cons)) +
  geom_line(color = "steelblue") +
  ylab("Wind Energy Consumption (Trillion Btu)") +
  xlab("Year") +
  scale_x_date(date_breaks = "5 years",
               date_labels = "%Y") +
  theme_minimal()
```

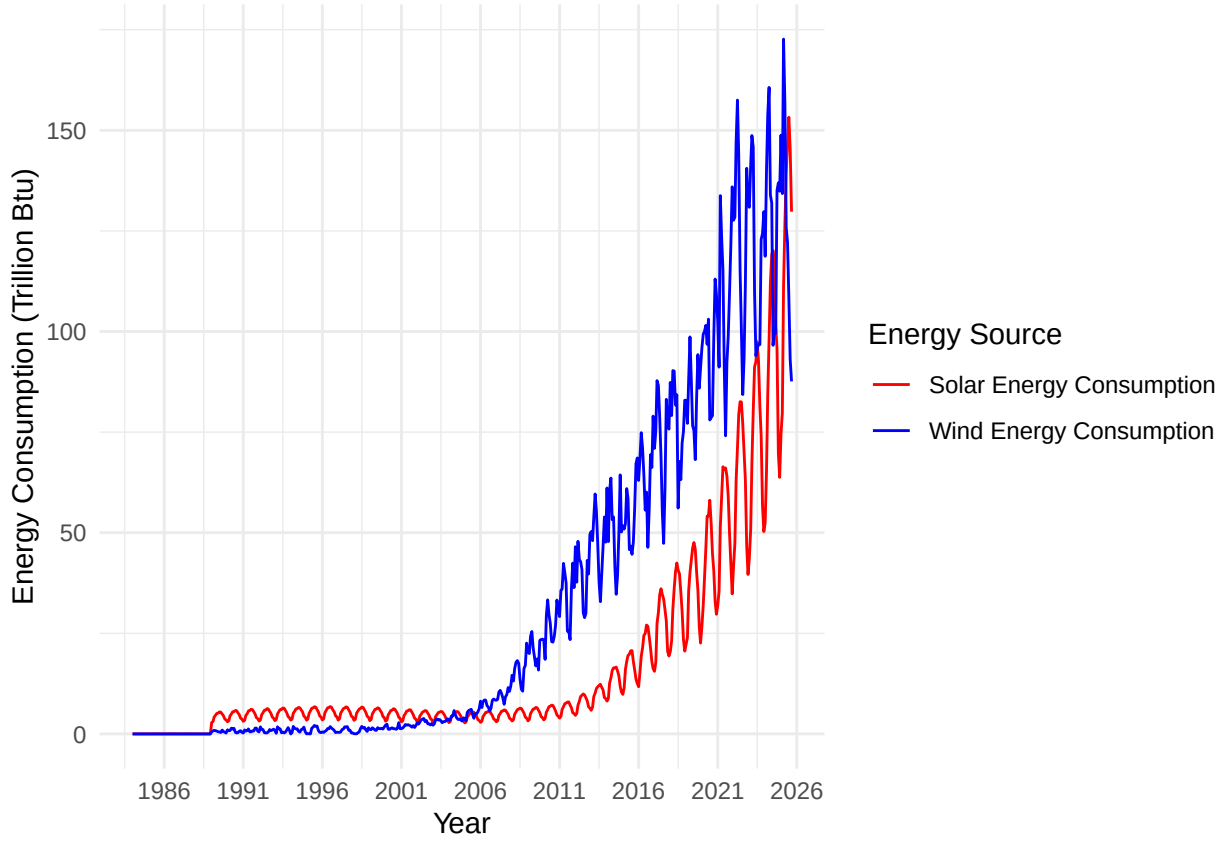


Q3

Now plot both series in the same graph, also using ggplot(). Use function `scale_color_manual()` to manually add a legend to ggplot. Make the solar energy consumption red and wind energy consumption blue. Add informative name to the y axis using `ylab("Energy Consumption)`. And use function `scale_x_date()` to set x axis breaks every 5 years.

```
library(tidyr)

# Convert Date to Date class (if not already done)
df_sw_clean <- df_sw_clean %>%
  mutate(Date = as.Date(Date))

# Reshape to long format
df_long <- df_sw_clean %>%
  pivot_longer(
    cols = c(Solar_Cons, Wind_Cons),
    names_to = "Source",
    values_to = "Consumption"
  )

# Plot both series together
ggplot(df_long, aes(x = Date, y = Consumption, color = Source)) +
  geom_line() +
  ylab("Energy Consumption (Trillion Btu)") +
  xlab("Year") +
```

```
scale_color_manual(
  values = c("Solar_Cons" = "red",
             "Wind_Cons"  = "blue"),
  labels = c("Solar Energy Consumption",
             "Wind Energy Consumption"),
  name = "Energy Source"
) +
scale_x_date(date_breaks = "5 years",
             date_labels = "%Y") +
theme_minimal()
```



Decomposing the time series

The stats package has a function called decompose(). This function only take time series object. As the name says the decompose function will decompose your time series into three components: trend, seasonal and random. This is similar to what we did in the previous script, but in a more automated way. The random component is the time series without seasonal and trend component.

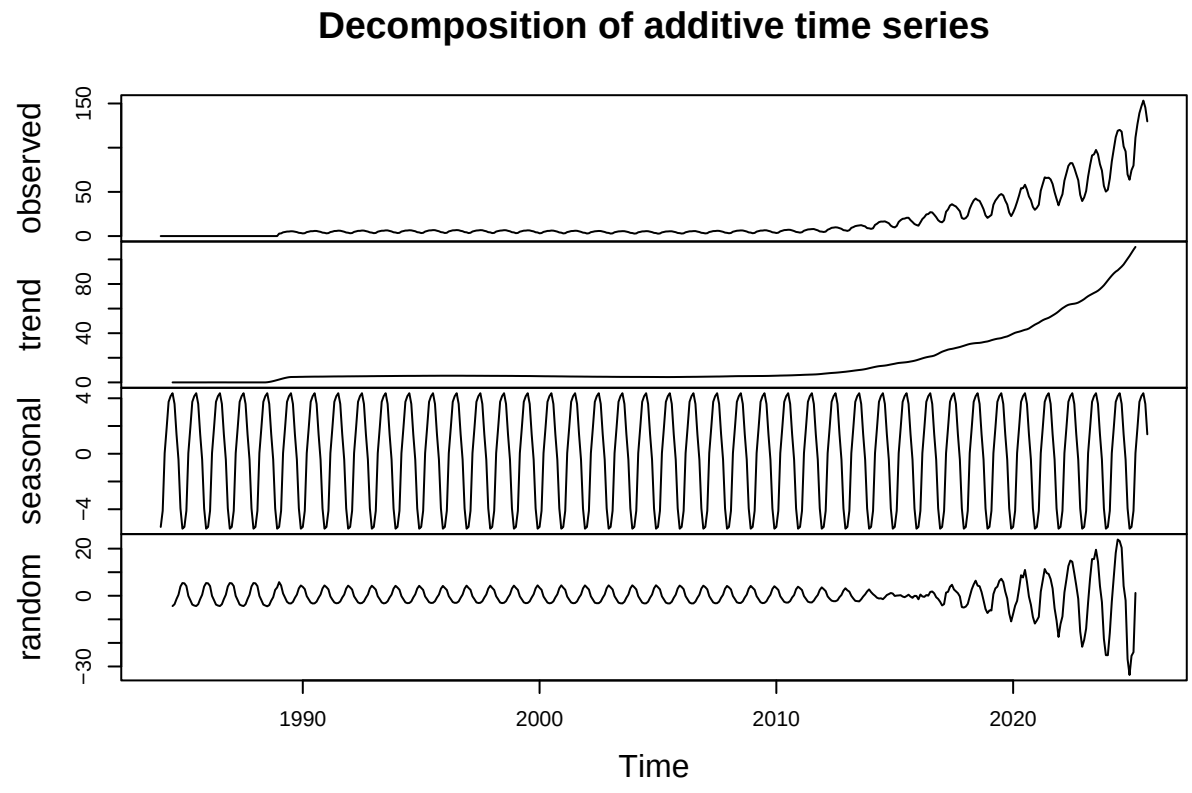
Additional info on `decompose()`.

1) You have two options: alternative and multiplicative. Multiplicative models exhibit a change in frequency over time.
2) The trend is not a straight line because it uses a moving average method to detect trend.
3) The seasonal component of the time series is found by subtracting the trend component from the original data then grouping the results by month and averaging them.
4) The random component, also referred to as the noise component, is composed of all the leftover signal which is not explained by the combination of the trend and seasonal component.

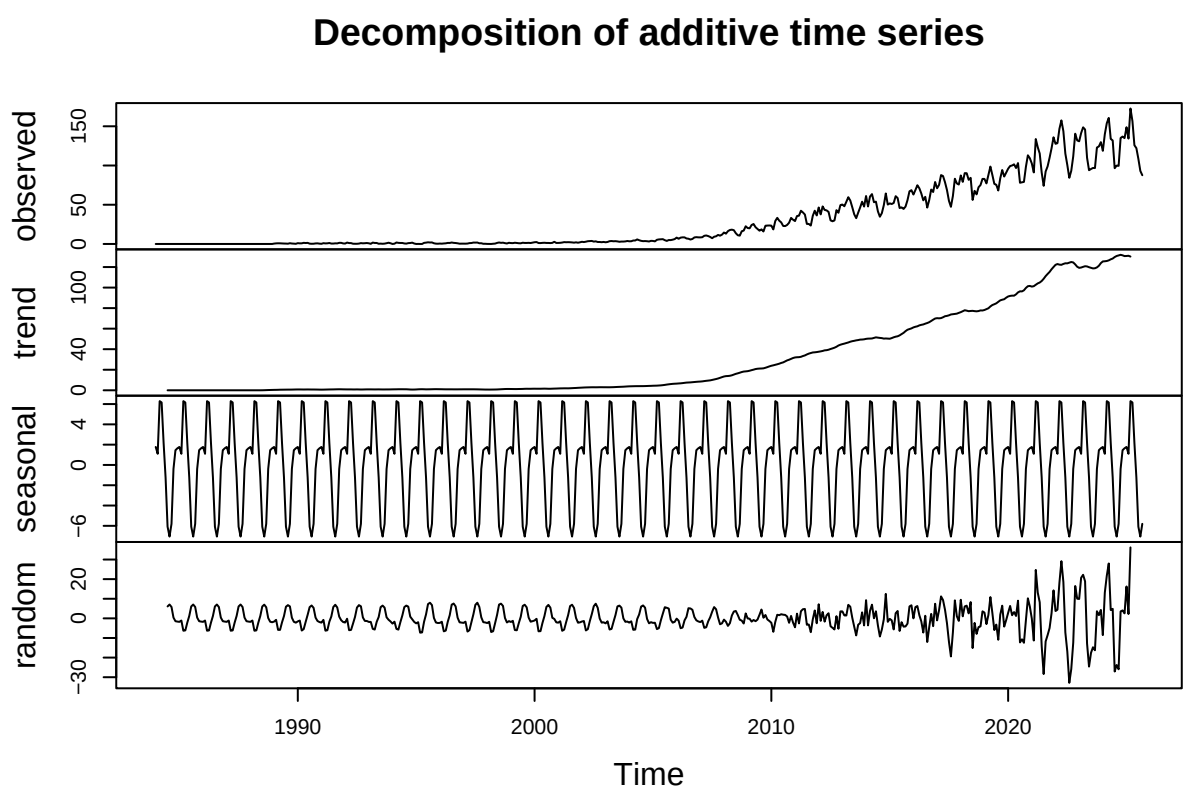
Q4

Transform wind and solar series into a time series object and apply the decompose function on them using the additive option, i.e., `decompose(ts_data, type = "additive")`. What can you say about the trend component? What about the random component? Does the random component look random? Or does it appear to still have some seasonality on it?

```r
# Extract start year and month
start_year  <- as.numeric(format(min(df_sw_clean$Date), "%Y"))
start_month <- as.numeric(format(min(df_sw_clean$Date), "%m"))

# Create ts objects
solar_ts <- ts(df_sw_clean$Solar_Cons,
               start = c(start_year, start_month),
               frequency = 12)

wind_ts  <- ts(df_sw_clean$Wind_Cons,
               start = c(start_year, start_month),
               frequency = 12)

solar_dec <- decompose(solar_ts, type = "additive")
wind_dec  <- decompose(wind_ts,  type = "additive")

plot(solar_dec)
```



```
plot(wind_dec)
```

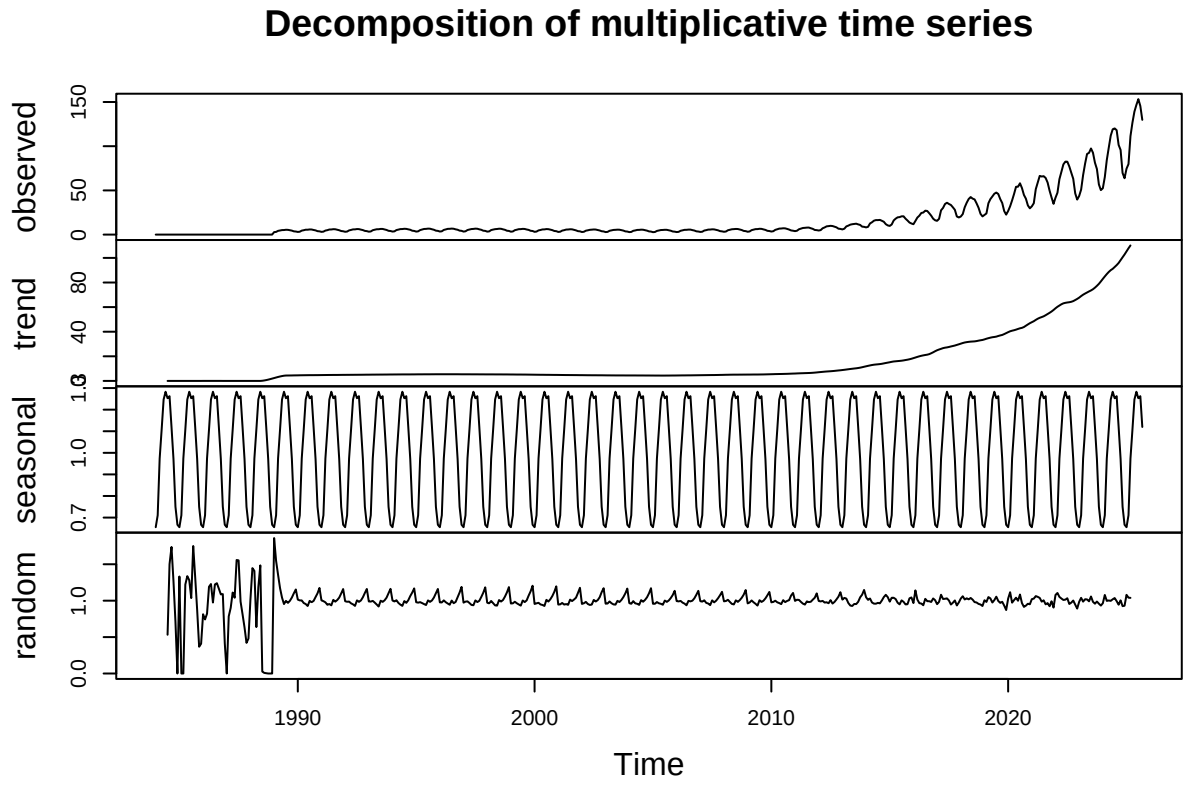


After transforming both the solar and wind consumption series into monthly time series objects and applying an additive decomposition, the trend component for both series shows a clear and strong upward movement over time. Wind begins increasing earlier, with noticeable acceleration after the early 2000s, while solar remains relatively flat for many years before rising sharply after around 2014. The trend is smooth because it is estimated using a moving average. Looking at the random component, it does not appear completely random. The variability increases over time, particularly in the later years, and there are visible patterns rather than purely irregular noise. This suggests that the additive model may not fully capture the structure of the data, especially given the increasing amplitude as the level of the series grows.

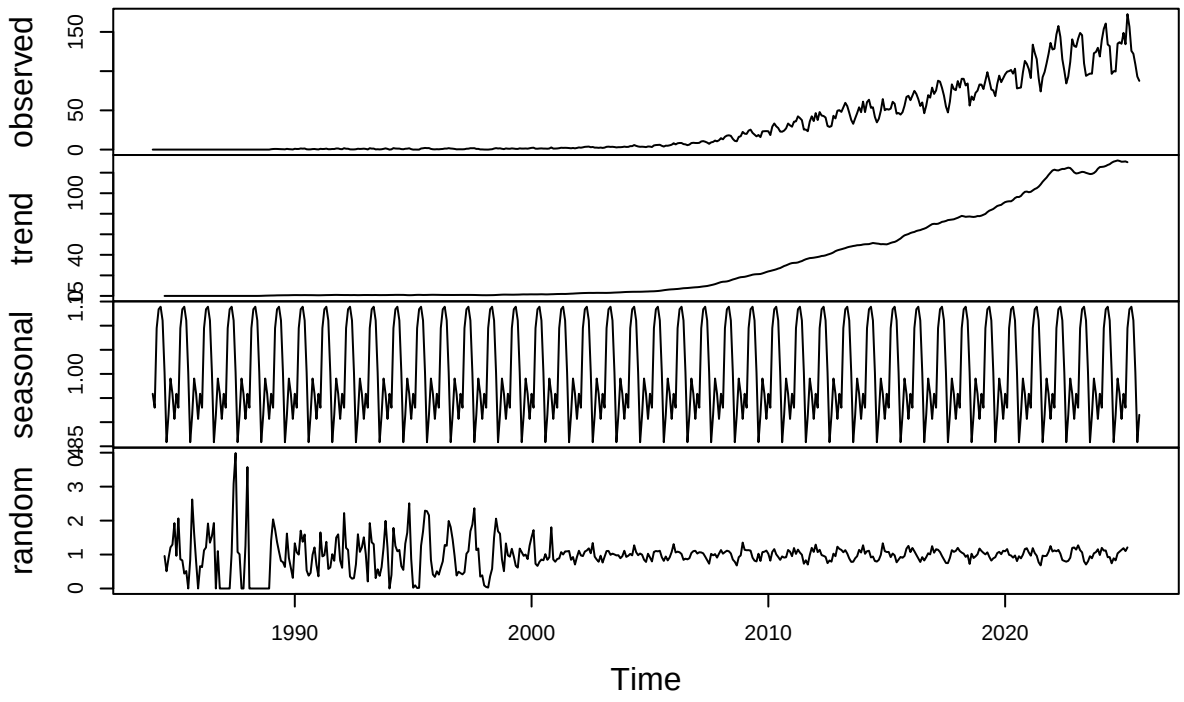
Q5

Use the decompose function again but now change the type of the seasonal component from additive to multiplicative. What happened to the random component this time?

```
solar_dec_mult <- decompose(solar_ts, type = "multiplicative")
wind_dec_mult  <- decompose(wind_ts,  type = "multiplicative")

plot(solar_dec_mult)
```




```
plot(wind_dec_mult)
```



After applying the multiplicative decomposition, the random component appears more stable over time compared to the additive case. In the additive model, the residual variability increased as the level of the series grew, especially in later years. Under the multiplicative specification, the random component now looks more centered and exhibits less systematic structure, although with more noise in the earlier years. It appears closer to true noise, with reduced evidence of remaining seasonality or trend patterns.

Q6

When fitting a model to this data, do you think you need all the historical data? Think about the data from 80s, 90s and early 20s. Are there any information from those years we might need to forecast the next six months of Solar and/or Wind consumption. Explain your response.

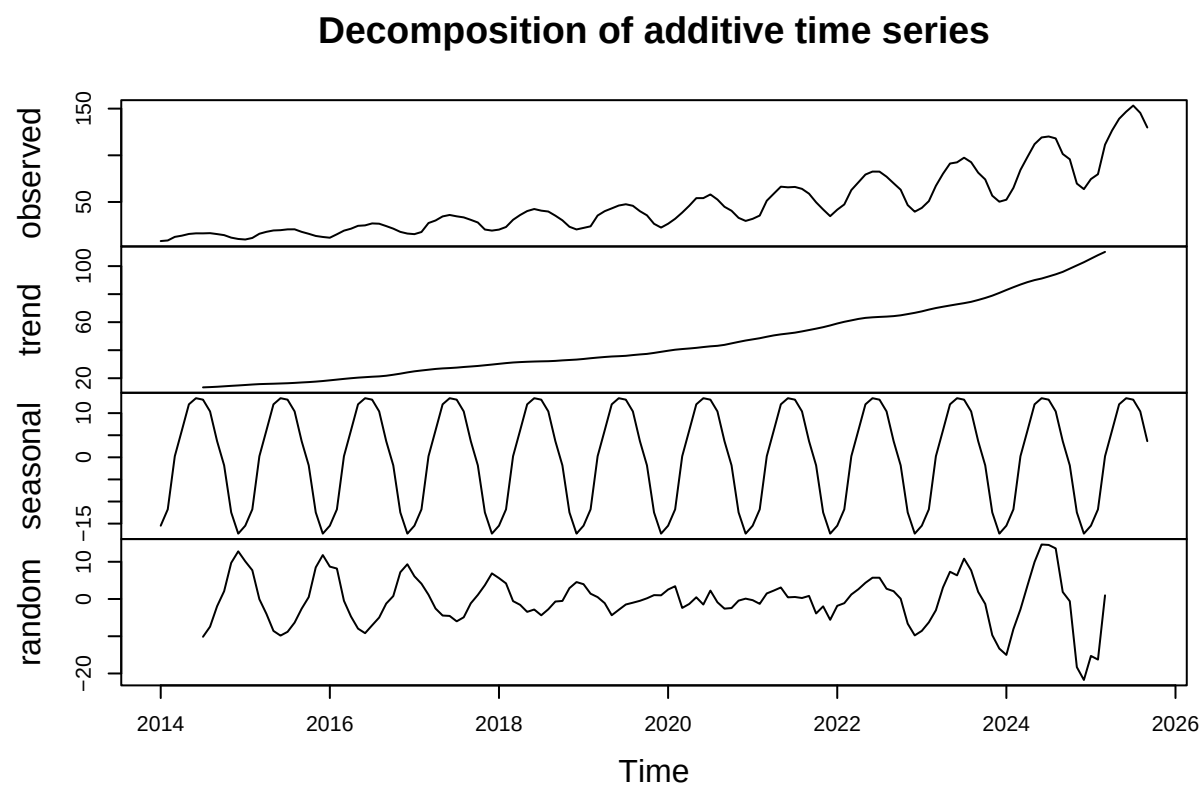
> Answer: When fitting a model to forecast the next six months of solar or wind consumption, I do not think we necessarily need all the historical data going back to the 1980s and 1990s. The early years reflect very different structural conditions, especially for solar, where consumption was close to zero and the industry was not yet developed. Those periods do not contain much useful information about the current growth dynamics or seasonal patterns driving the series today. Since both solar and wind exhibit strong upward trends and rapid expansion in the past decade, the most relevant information for short-term forecasting likely comes from more recent years, where the level, volatility, and seasonal amplitude resemble current conditions. While older data help establish long-run trend behavior, for forecasting the next six months, recent observations are more informative and better reflect the current regime of renewable energy growth.

Q7

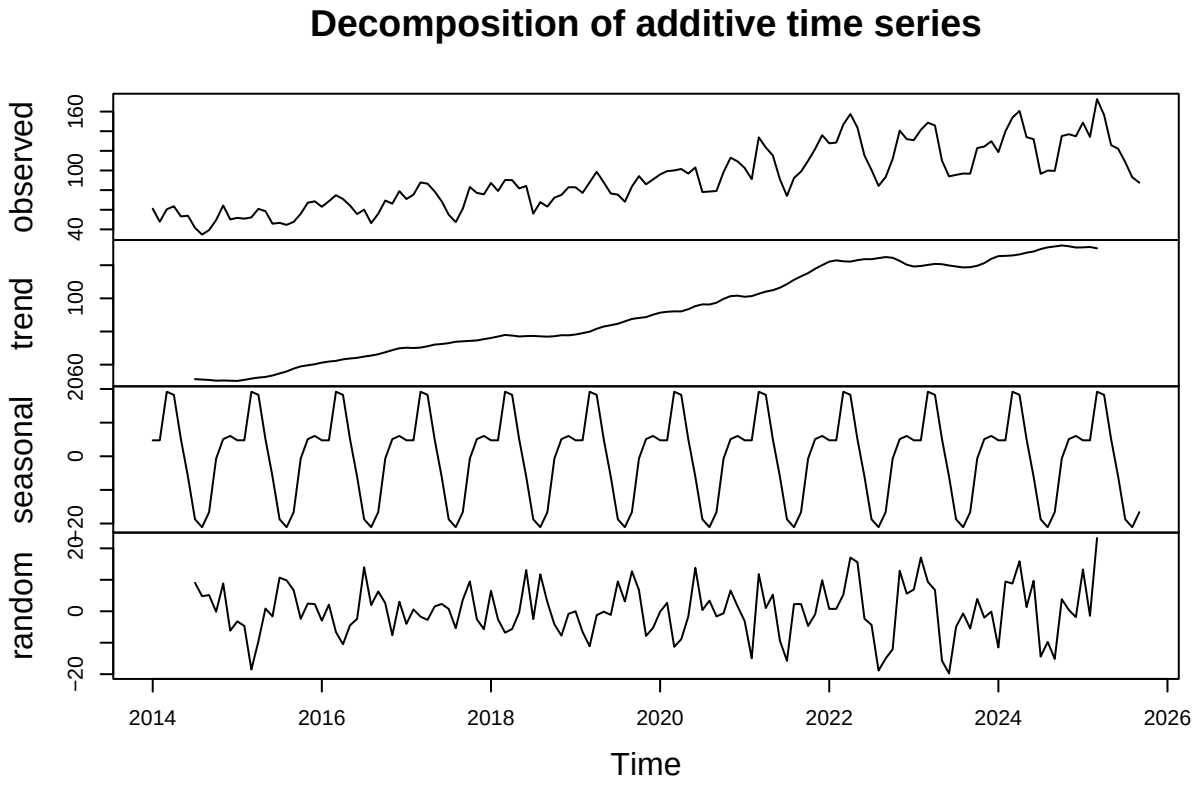
Create a new time series object where historical data starts on January 2014. Hint: use `filter()` function so that you don't need to point to row numbers, .i.e, `filter(xxxx, year(Date) >= 2014 )`. Apply the

decompose function `type=additive` to this new time series. Comment on the results. Does the random component look random?

```
library(dplyr)
library(lubridate)

df_2014 <- df_sw_clean %>%
  filter(year(Date) >= 2014)

# Create ts objects starting Jan 2014 (monthly)
solar_ts_2014 <- ts(df_2014$Solar_Cons, start = c(2014, 1), frequency = 12)
wind_ts_2014  <- ts(df_2014$Wind_Cons,  start = c(2014, 1), frequency = 12)

# Additive decomposition
solar_dec_2014_add <- decompose(solar_ts_2014, type = "additive")
wind_dec_2014_add  <- decompose(wind_ts_2014,  type = "additive")

plot(solar_dec_2014_add)
```




```
plot(wind_dec_2014_add)
```



Answer: After restricting the sample to start in January 2014 and applying additive decomposition, the trend component for both solar and wind continues to show a strong upward movement, reflecting the rapid expansion of renewable energy in the past decade. The seasonal component remains very clear and stable across months, indicating consistent seasonal patterns. However, the random component does not appear completely random. There are still visible fluctuations and periods of deviations that look cyclical. This suggests that even within the more recent sample, the additive specification may not fully capture the level-dependent variation in the data.

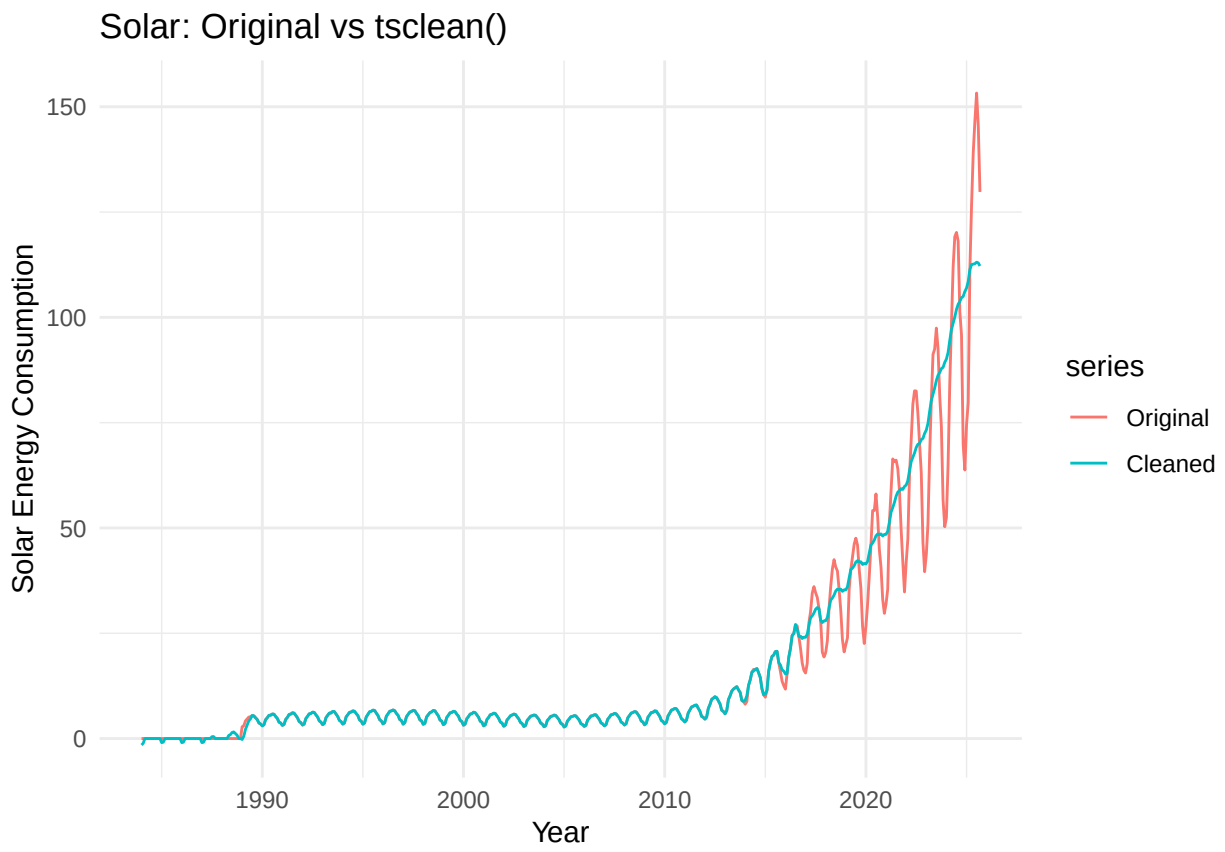
Identify and Remove outliers

Q8

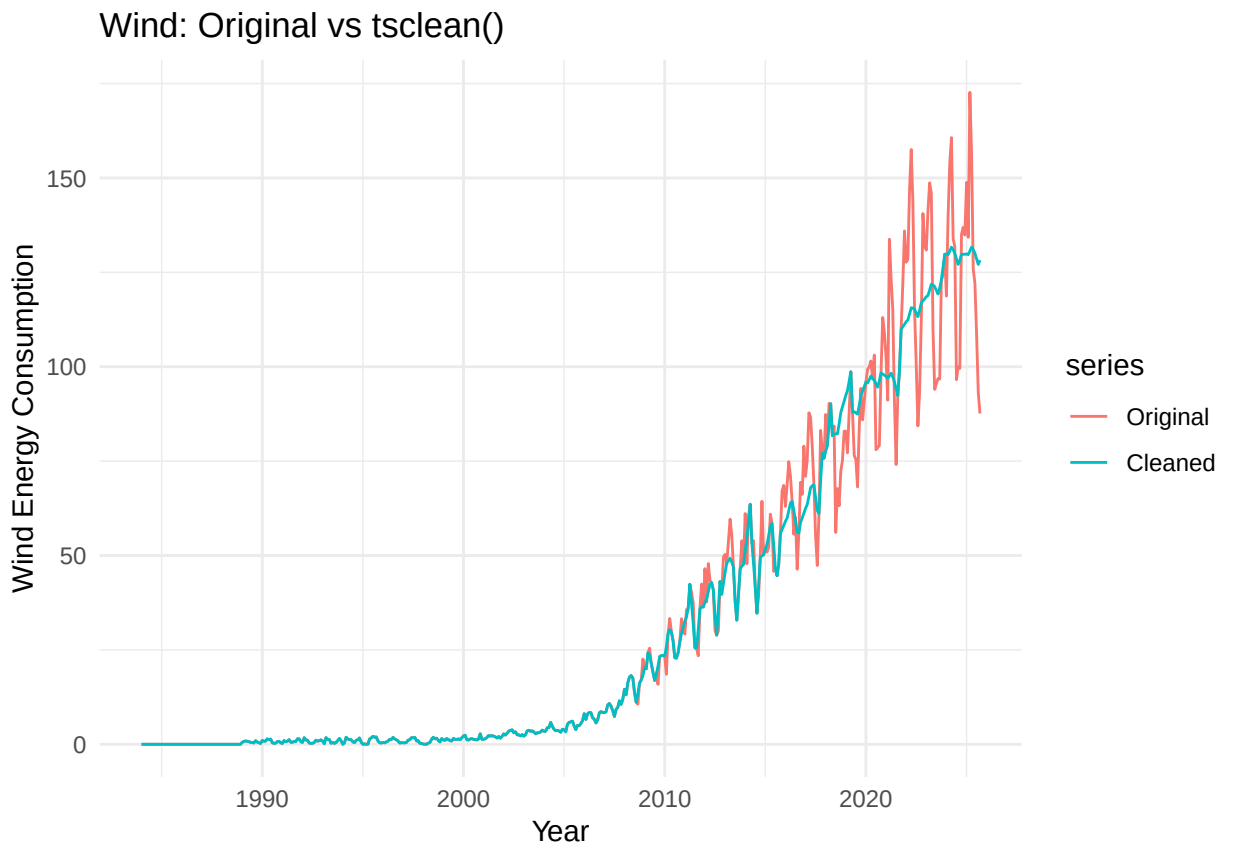
Apply the `tsclean()` to both time series object you created on Q4. Did the function removed any outliers from the series? Hint: Use `autoplot()` to check if there is difference between cleaned series and original series.

```
library(forecast)
library(ggplot2)

# Clean the full-sample ts objects from Q4
solar_ts_clean <- tsclean(solar_ts)
wind_ts_clean  <- tsclean(wind_ts)

# Overlay plots (original vs cleaned) to see differences
autoplot(cbind(Original = solar_ts, Cleaned = solar_ts_clean)) +
  ylab("Solar Energy Consumption") +
  xlab("Year") +
```

```
  ggtitle("Solar: Original vs tsclean()") +
  theme_minimal()
```




```
autoplot(cbind(Original = wind_ts, Cleaned = wind_ts_clean)) +
  ylab("Wind Energy Consumption") +
  xlab("Year") +
  ggtitle("Wind: Original vs tsclean()") +
  theme_minimal()
```



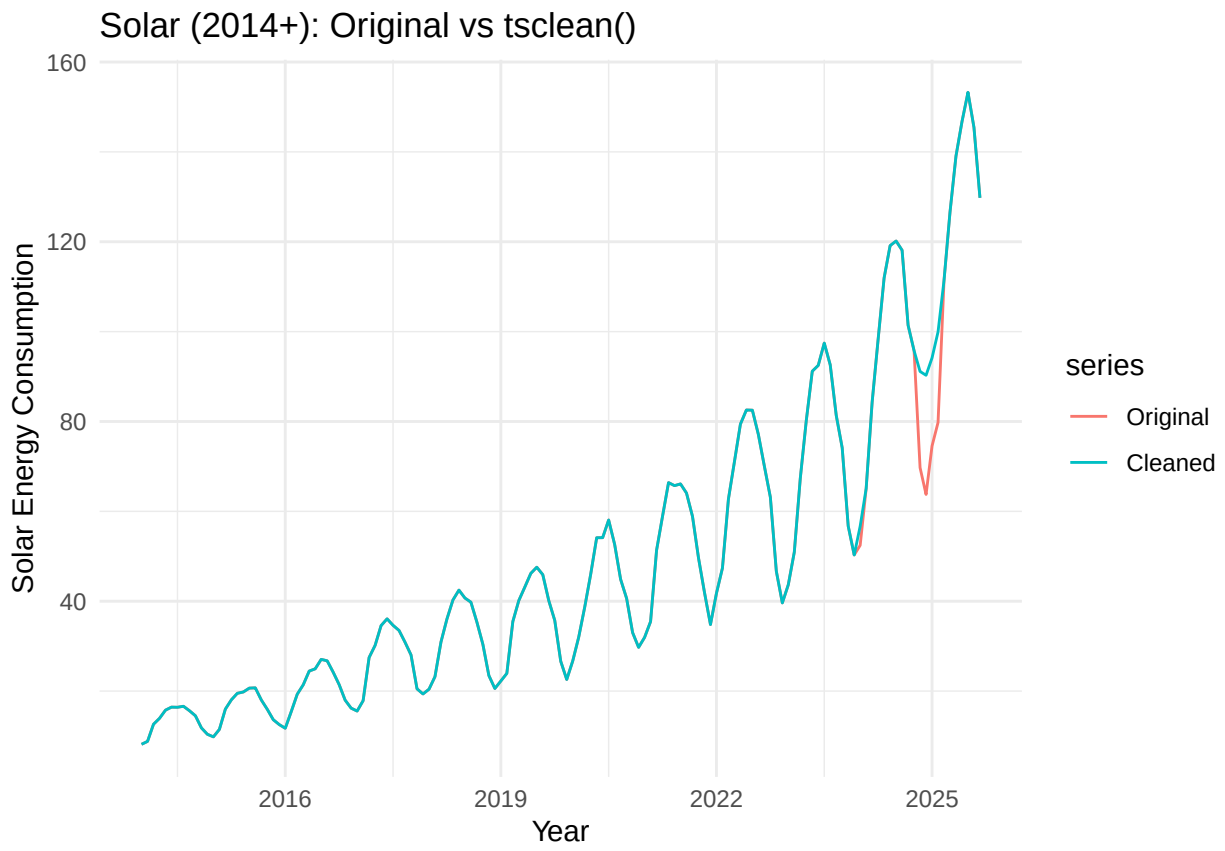

```
# which(solar_ts != solar_ts_clean)
# which(wind_ts  != wind_ts_clean)
```

After applying tsclean() to both the solar and wind time series, the function did remove and smooth several extreme values. This is particularly visible in the most recent years, where the original series shows sharp spikes and large seasonal swings. The cleaned series appears smoother and less volatile, especially at the peaks. The differences between the original and cleaned lines indicate that tsclean() identified and adjusted outliers rather than leaving the series unchanged. The effect is more noticeable for solar in the most recent period and for wind during large spikes, suggesting that the function detected unusually large deviations relative to the underlying trend and seasonal structure.

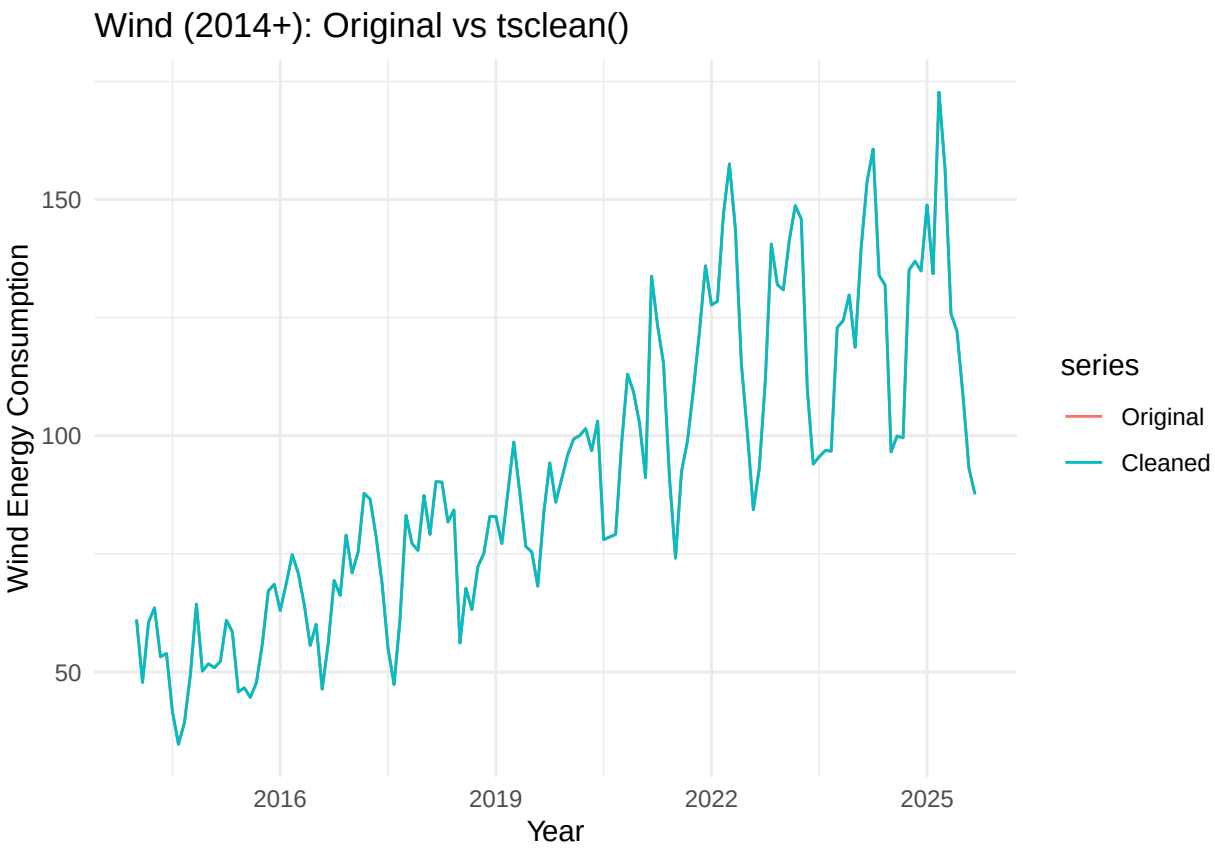
Q9

Redo number Q8 but now with the time series you created on Q7, i.e., the series starting in 2014. Using what `autoplot()` again what happened now? Did the function removed any outliers from the series?

```
# Clean 2014+ series
solar_ts_2014_clean <- tsclean(solar_ts_2014)
wind_ts_2014_clean  <- tsclean(wind_ts_2014)

# Plot comparison
autoplot(cbind(Original = solar_ts_2014,
               Cleaned  = solar_ts_2014_clean)) +
  ylab("Solar Energy Consumption") +
  xlab("Year") +
  ggtitle("Solar (2014+): Original vs tsclean()") +
  theme_minimal()
```




```
autoplot(cbind(Original = wind_ts_2014,
               Cleaned  = wind_ts_2014_clean)) +
  ylab("Wind Energy Consumption") +
  xlab("Year") +
  ggtitle("Wind (2014+): Original vs tsclean()") +
  theme_minimal()
```



Answer: When applying tsclean() to the 2014-onward series, the cleaned series almost completely overlaps with the original series for both solar and wind. Unlike the full historical sample, there are no visible adjustments or smoothing of extreme spikes. This suggests that tsclean() did not identify meaningful outliers within the recent period. The large fluctuations observed after 2014 are therefore consistent with the underlying trend and seasonal growth rather than being treated as abnormal observations. In this restricted sample, the variation appears structural rather than anomalous.