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

Assignment 5 - Due date 02/17/26

Lingyue Hao

## 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)
library(lubridate)
```

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

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

```r
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
```

```r
library(readxl)
```

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

```r
#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`
```

```r
#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!

```
energy_subset <- energy_data %>%
  select(
    Date = Month,
    `Solar Energy Consumption`,
    `Wind Energy Consumption`
  )

sapply(energy_subset, class)
```

```
## $Date
## [1] "POSIXct" "POSIXt"
##
## $`Solar Energy Consumption`
## [1] "character"
##
## $`Wind Energy Consumption`
## [1] "character"
```

```
energy_subset <- energy_subset %>%
  mutate(
    `Solar Energy Consumption` = as.numeric(`Solar Energy Consumption`),
    `Wind Energy Consumption`  = as.numeric(`Wind Energy Consumption`)
  )
```

```
## Warning: There were 2 warnings in `mutate()`.
## The first warning was:
## i In argument: `Solar Energy Consumption = as.numeric(`Solar Energy
##   Consumption`)`.
## Caused by warning:
## ! NAs introduced by coercion
## i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```
colSums(is.na(energy_subset))
```

```
##                     Date Solar Energy Consumption  Wind Energy Consumption
##                        0                      132                      120
```

```
energy_subset <- energy_subset %>%
  drop_na()

summary(energy_subset)
```

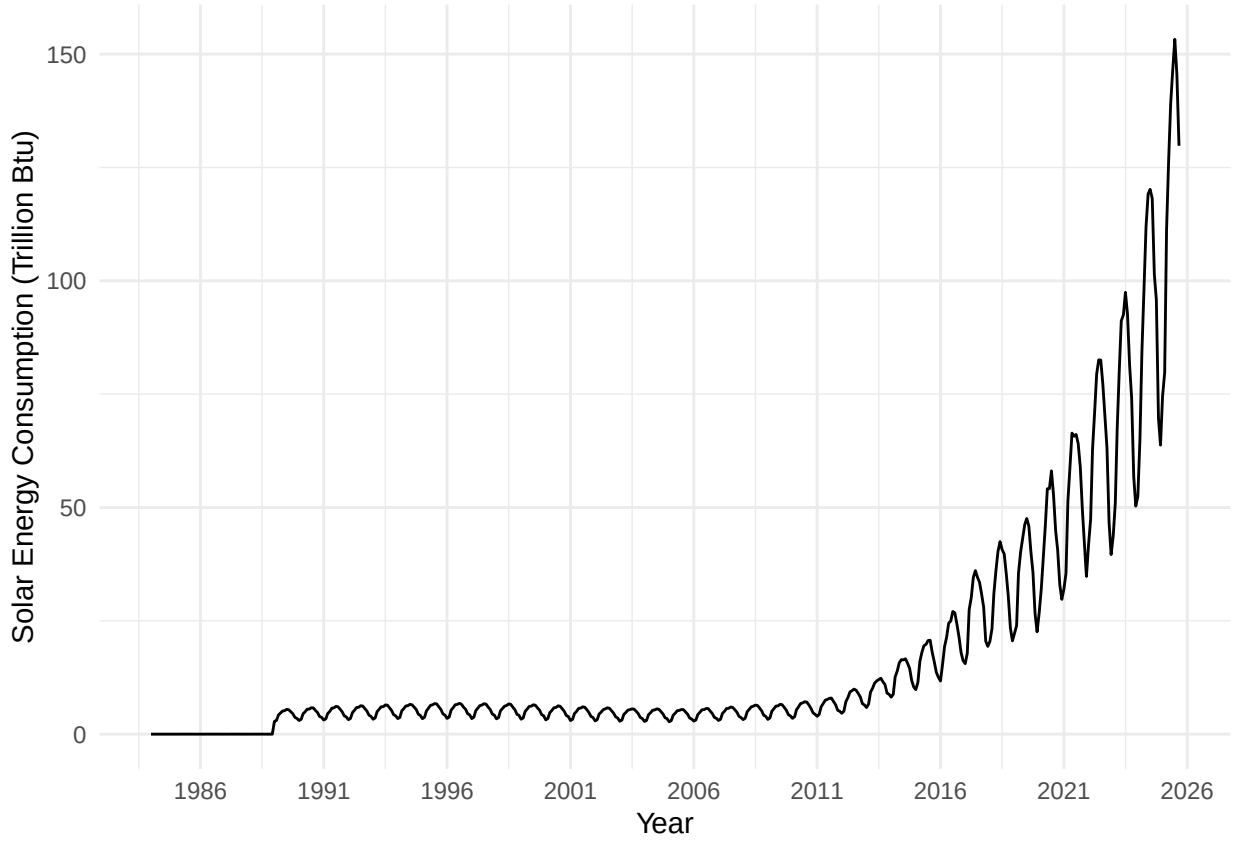
```
##       Date                        Solar Energy Consumption
##  Min.   :1984-01-01 00:00:00.00   Min.   :  0.000
##  1st Qu.:1994-06-01 00:00:00.00   1st Qu.:  3.924
##  Median :2004-11-01 00:00:00.00   Median :  5.658
##  Mean   :2004-10-31 01:26:13.65   Mean   : 16.623
##  3rd Qu.:2015-04-01 00:00:00.00   3rd Qu.: 15.758
```

```
##  Max.   :2025-09-01 00:00:00.00   Max.   :153.256
##  Wind Energy Consumption
##  Min.   :  0.000
##  1st Qu.:  0.844
##  Median :  3.999
##  Mean   : 31.305
##  3rd Qu.: 55.001
##  Max.   :172.670
```

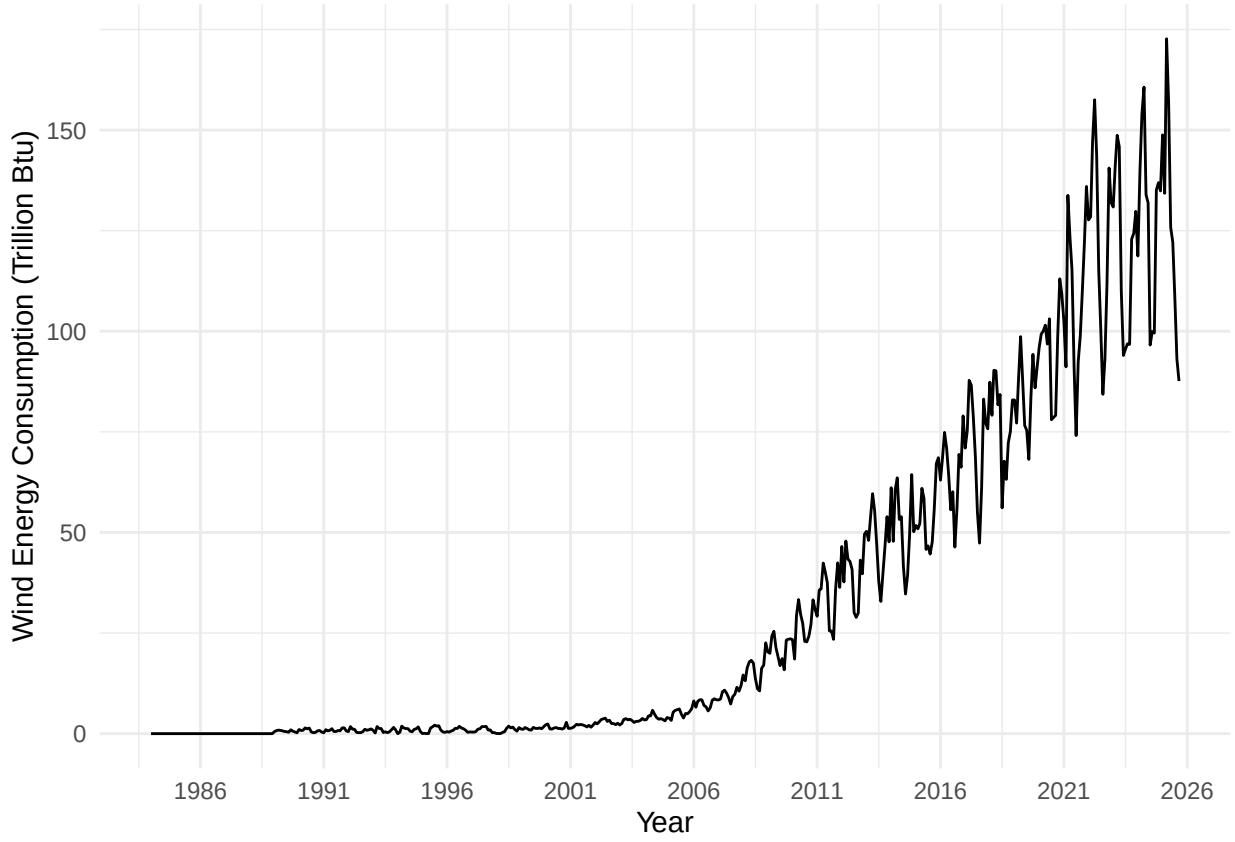
### Q2

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")")`

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

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



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

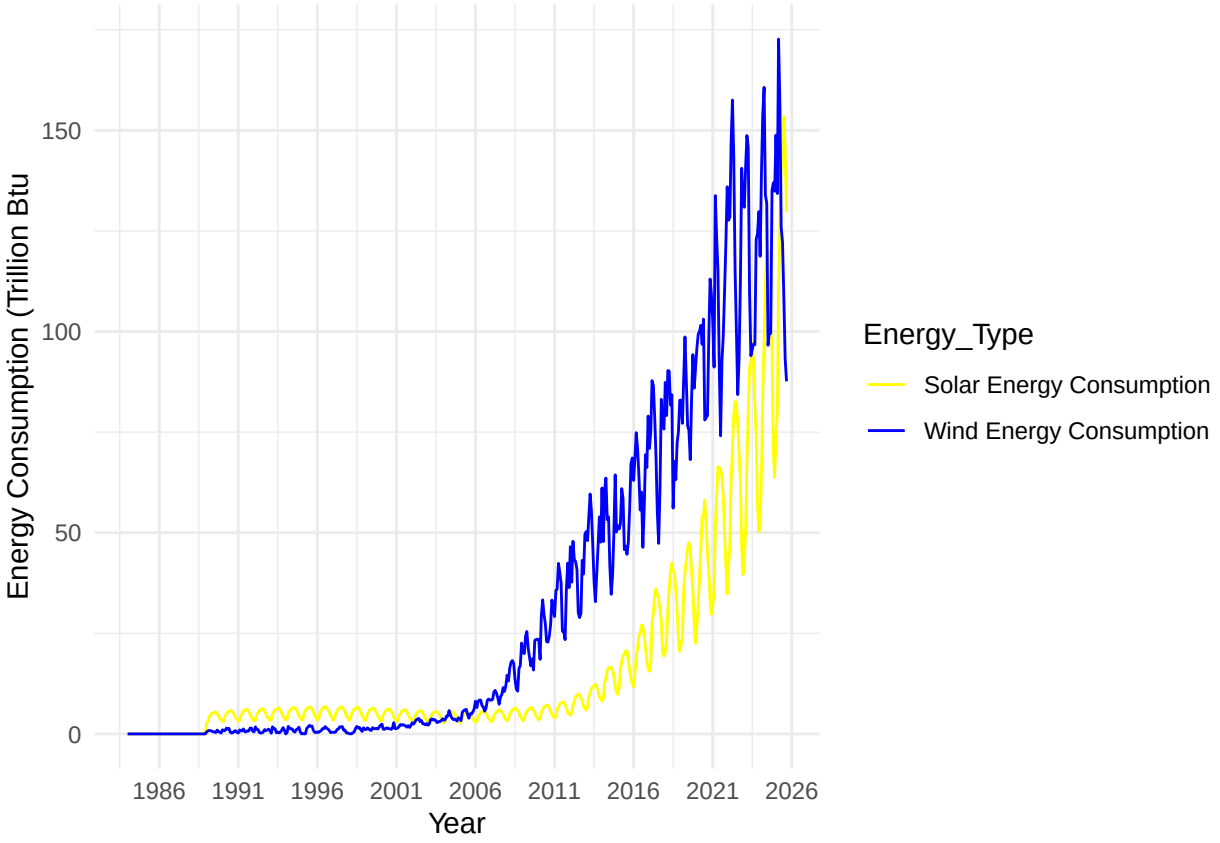



**Q3**

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.

```
energy_long <- energy_subset %>%
  pivot_longer(
    cols = c(`Solar Energy Consumption`,
             `Wind Energy Consumption`),
    names_to = "Energy_Type",
    values_to = "Value"
  )

ggplot(energy_long, aes(x = Date, y = Value, color = Energy_Type)) +
  geom_line() +
```

```
ylab("Energy Consumption (Trillion Btu") +
xlab("Year") +
scale_color_manual(
  values = c(
    "Solar Energy Consumption" = "yellow",
    "Wind Energy Consumption"  = "blue"
  )
) +
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()`.

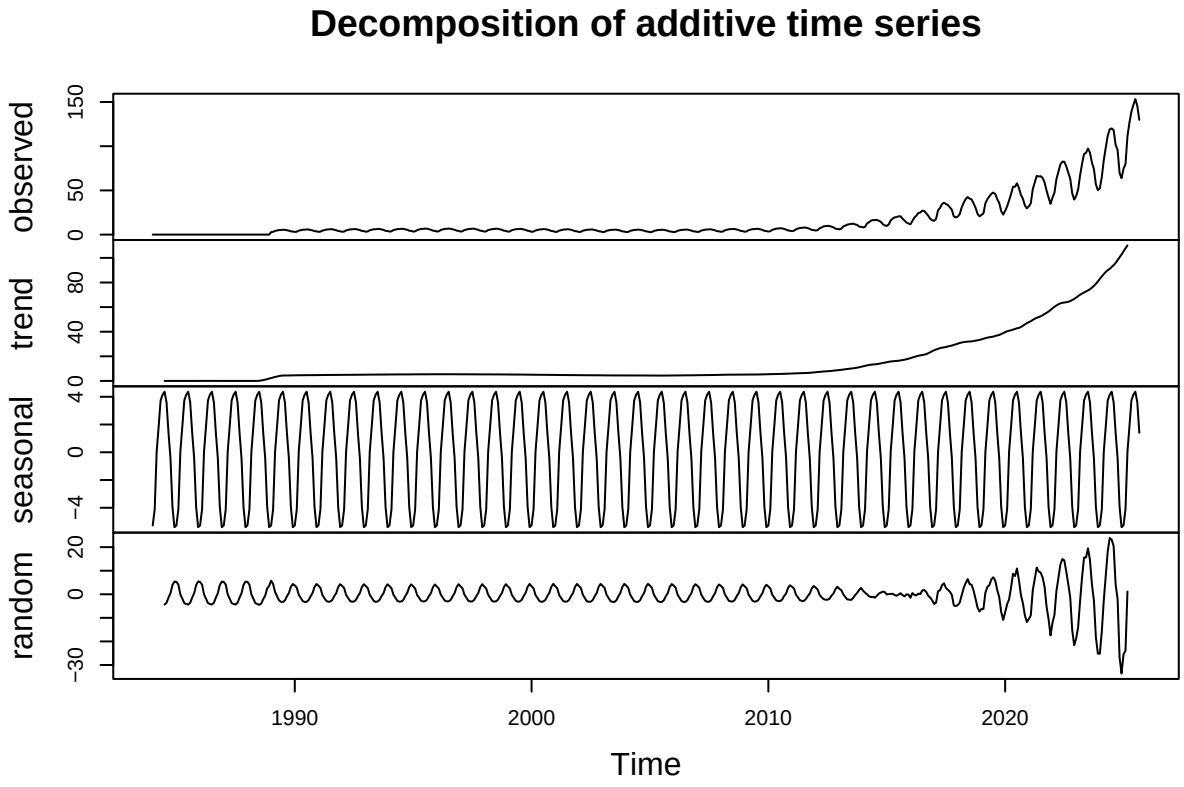
1) You have two options: alternative and multiplicative. Multiplicative models exhibit a change in frequency over time.
2) The trend is not a straight line because it uses a moving average method to detect trend.
3) The seasonal component of the time series is found by subtracting the trend component 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.

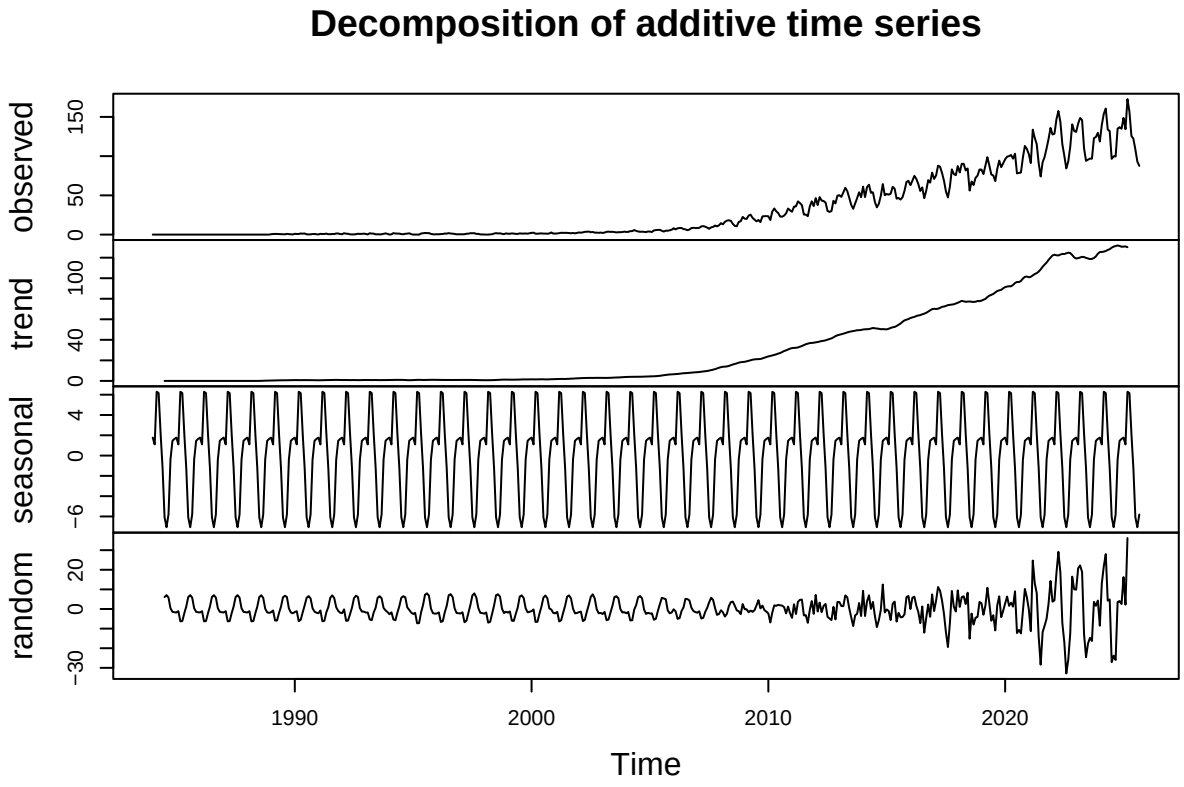
**Q4**

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?

```
start_year  <- year(min(energy_subset$Date))
start_month <- month(min(energy_subset$Date))

solar_ts <- ts(
  energy_subset$`Solar Energy Consumption`,
  start = c(start_year, start_month),
  frequency = 12
)

wind_ts <- ts(
  energy_subset$`Wind Energy Consumption`,
  start = c(start_year, start_month),
  frequency = 12
)

solar_decomp <- decompose(solar_ts, type = "additive")
wind_decomp  <- decompose(wind_ts,  type = "additive")

plot(solar_decomp)
```




```
plot(wind_decomp)
```

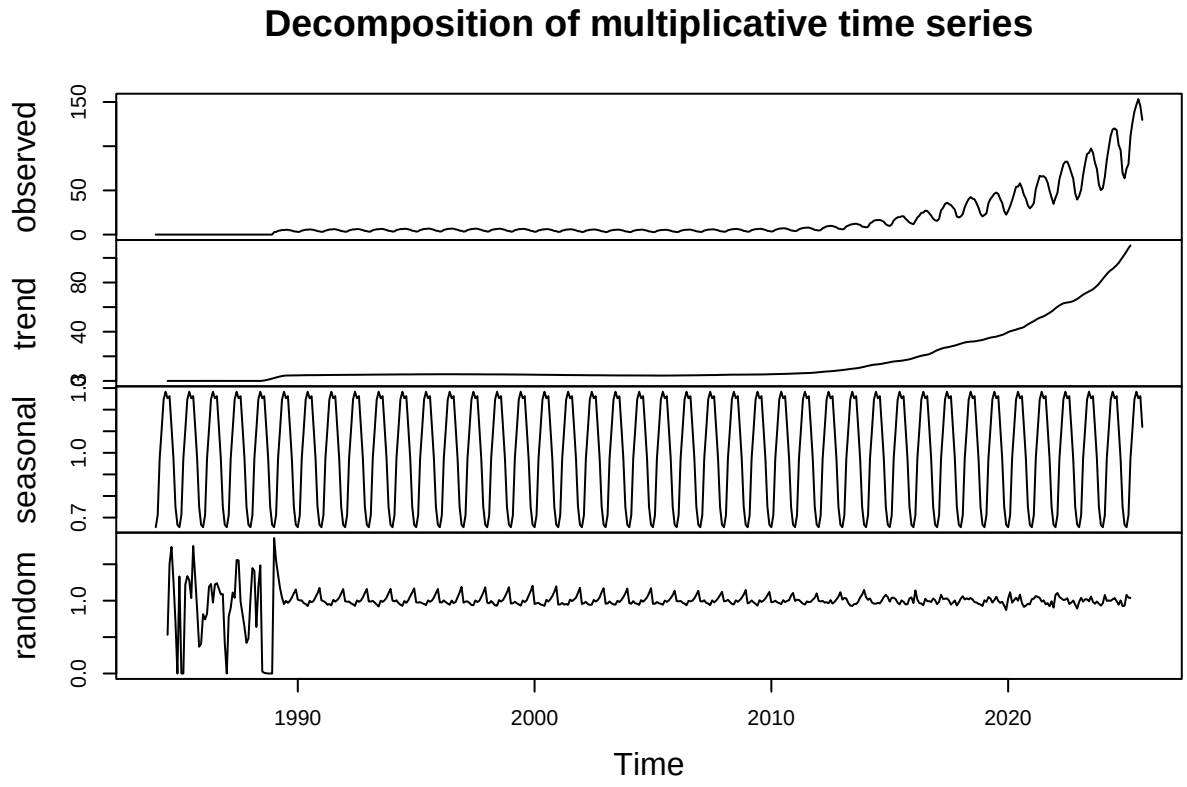



**The trend component shows a clear and strong upward pattern over time. The random component fluctuates around zero and does not display a clear repeating seasonal pattern, so most of the seasonality has been successfully removed by the additive decomposition. However, the variability of the random component appears to increase in the later part of the sample, which means it does not look completely stable over time. While it does not show obvious remaining seasonality, the changing variance suggests that the additive model may not fully capture all underlying dynamics in the series.**

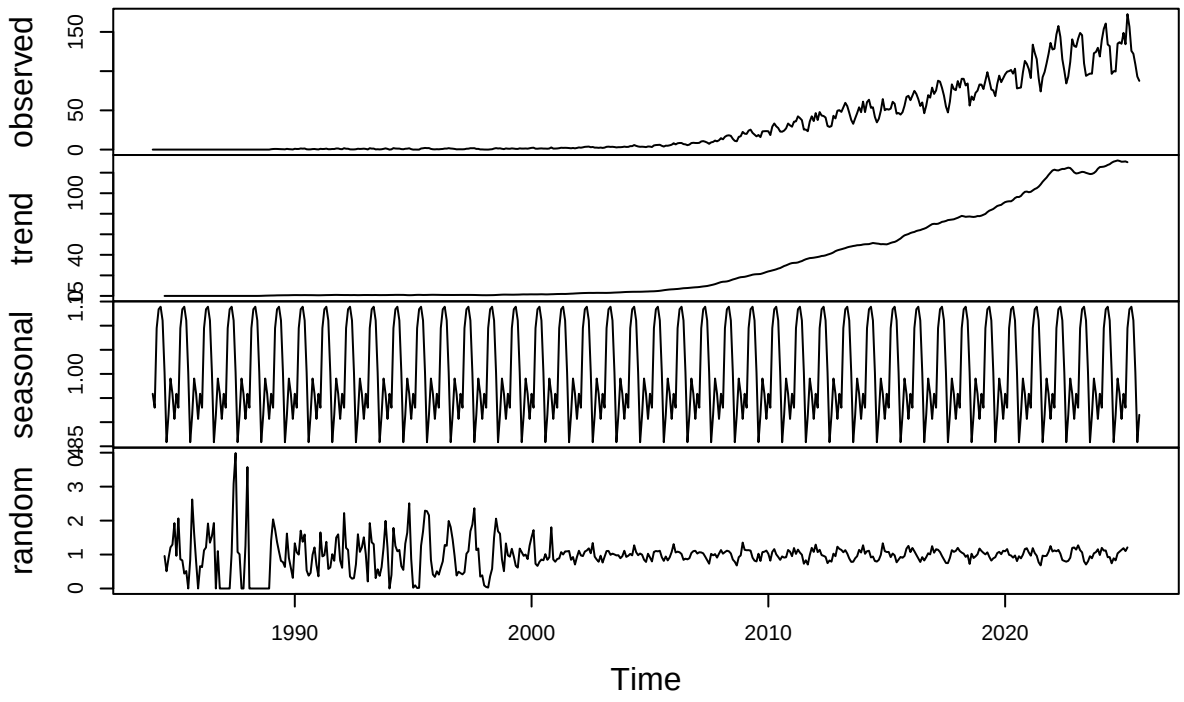
### Q5

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_decomp_mult <- decompose(solar_ts, type = "multiplicative")
wind_decomp_mult  <- decompose(wind_ts,  type = "multiplicative")

plot(solar_decomp_mult)
```




```
plot(wind_decomp_mult)
```




**The random component becomes much more stable compared to the additive case. After the early years, the residuals fluctuate closely around 1 and maintain a fairly constant spread over time. The large increase in variance seen under the additive model is no longer present. The random component does not show any clear seasonal pattern, and the fluctuations appear small and consistent relative to the level of the series.**

**Q6**

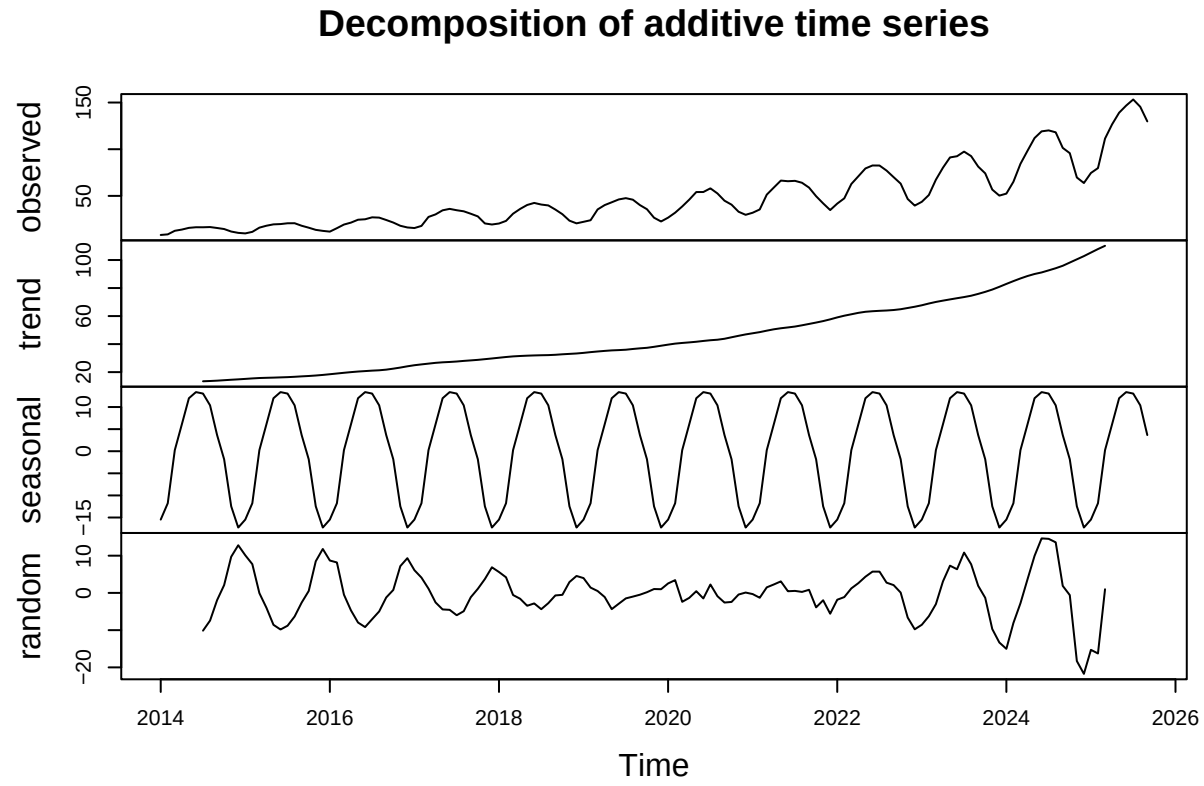
When fitting a model to this data, do you think you need all the historical data? Think about the data from 80s, 90s and early 20s. Are there any information from those years we might need to forecast the next six months of Solar and/or Wind consumption. Explain your response.

> Answer: I do not think we need all the historical data from the 80s, 90s, and early 2000s. The level of solar and wind consumption in those early years was much lower and the growth pattern has changed significantly over time. Because the series shows a strong upward trend and structural growth in recent years, the most recent data is likely more relevant for short-term forecasting.

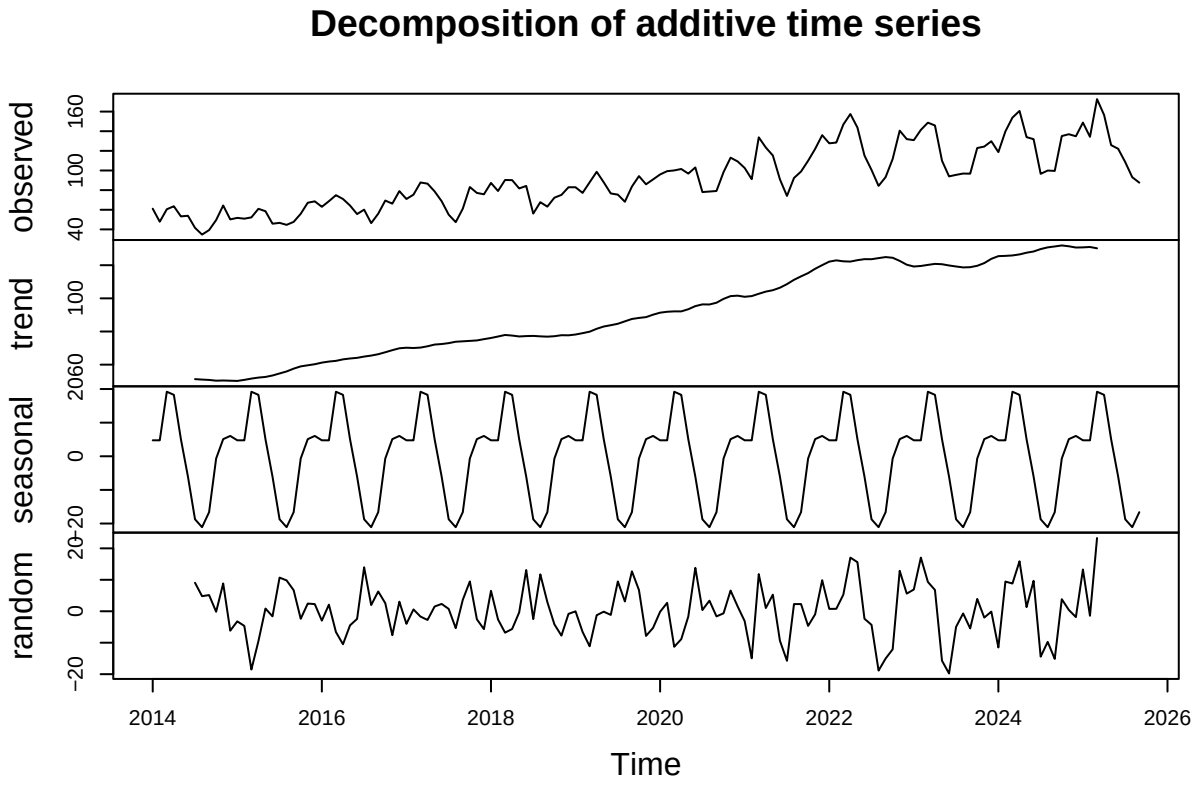
**Q7**

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?

```r
energy_2014 <- energy_subset %>%
  filter(year(Date) >= 2014)

solar_ts_2014 <- ts(
  energy_2014$`Solar Energy Consumption`,
  start = c(2014, 1),
  frequency = 12
)

wind_ts_2014 <- ts(
  energy_2014$`Wind Energy Consumption`,
  start = c(2014, 1),
  frequency = 12
)

solar_decomp_2014 <- decompose(solar_ts_2014, type = "additive")
wind_decomp_2014  <- decompose(wind_ts_2014,  type = "additive")

plot(solar_decomp_2014)
```




```r
plot(wind_decomp_2014)
```



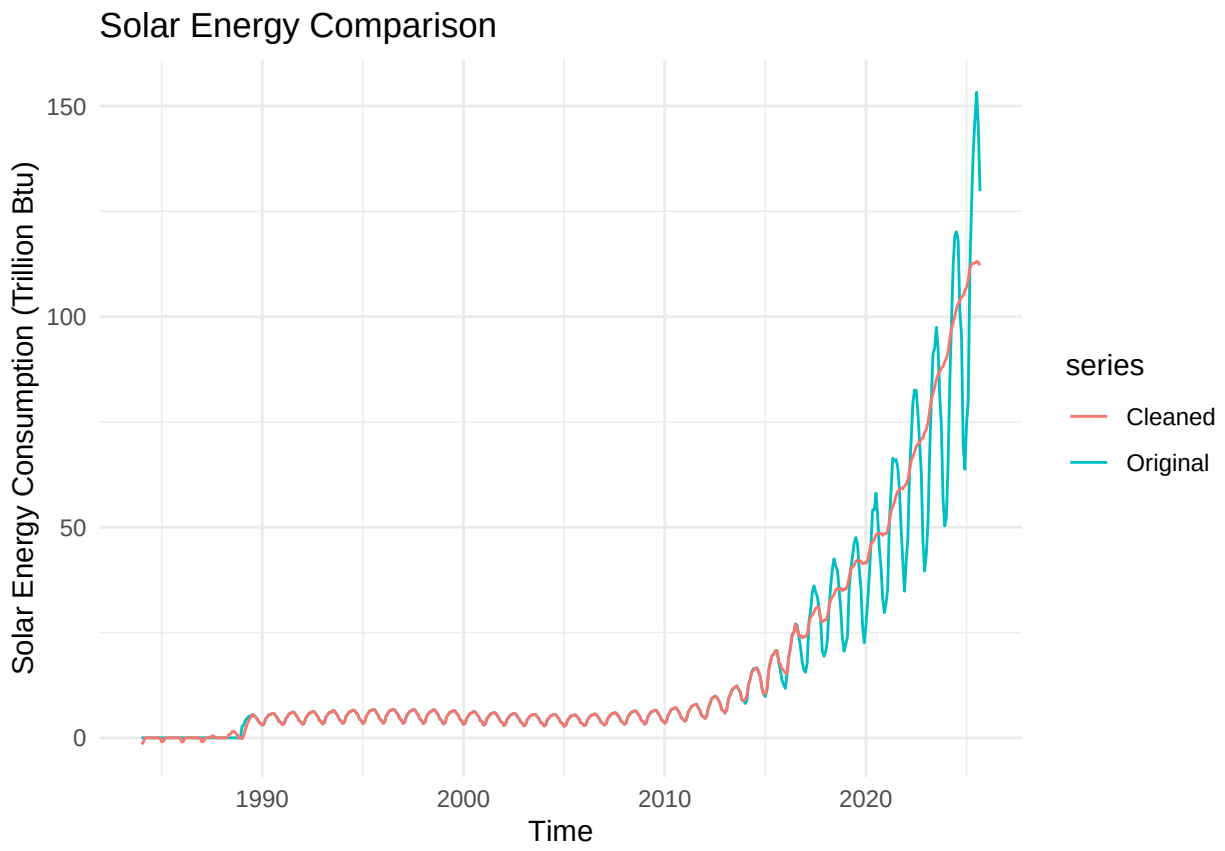

Answer: The random component looks mostly random. It fluctuates around zero without any clear repeating seasonal pattern.

## Identify and Remove outliers

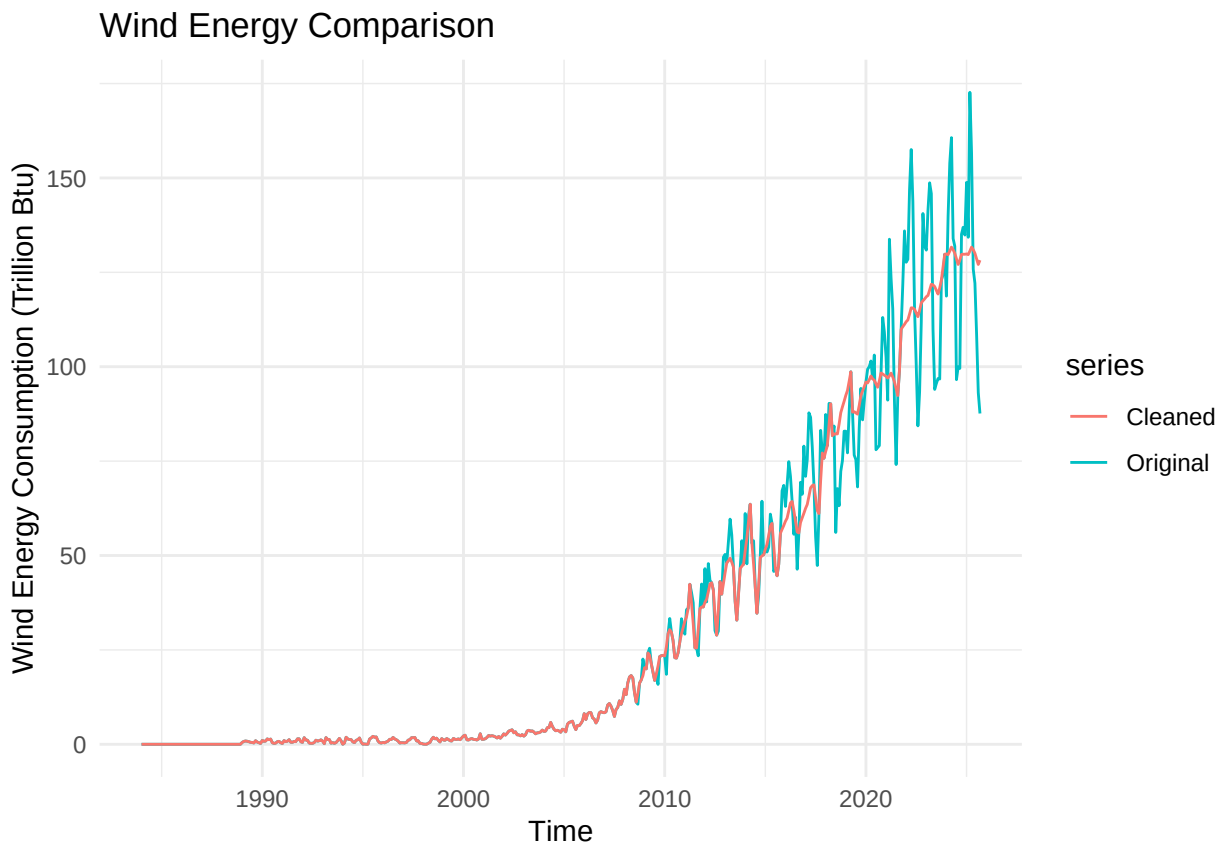
### Q8

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)

solar_clean <- tsclean(solar_ts)
wind_clean  <- tsclean(wind_ts)

autoplot(solar_ts, series = "Original") +
  autolayer(solar_clean, series = "Cleaned") +
  ggtitle("Solar Energy Comparison") +
  ylab("Solar Energy Consumption (Trillion Btu)") +
  theme_minimal()
```




```r
autoplot(wind_ts, series = "Original") +
  autolayer(wind_clean, series = "Cleaned") +
  ggtitle("Wind Energy Comparison") +
  ylab("Wind Energy Consumption (Trillion Btu)") +
  theme_minimal()
```

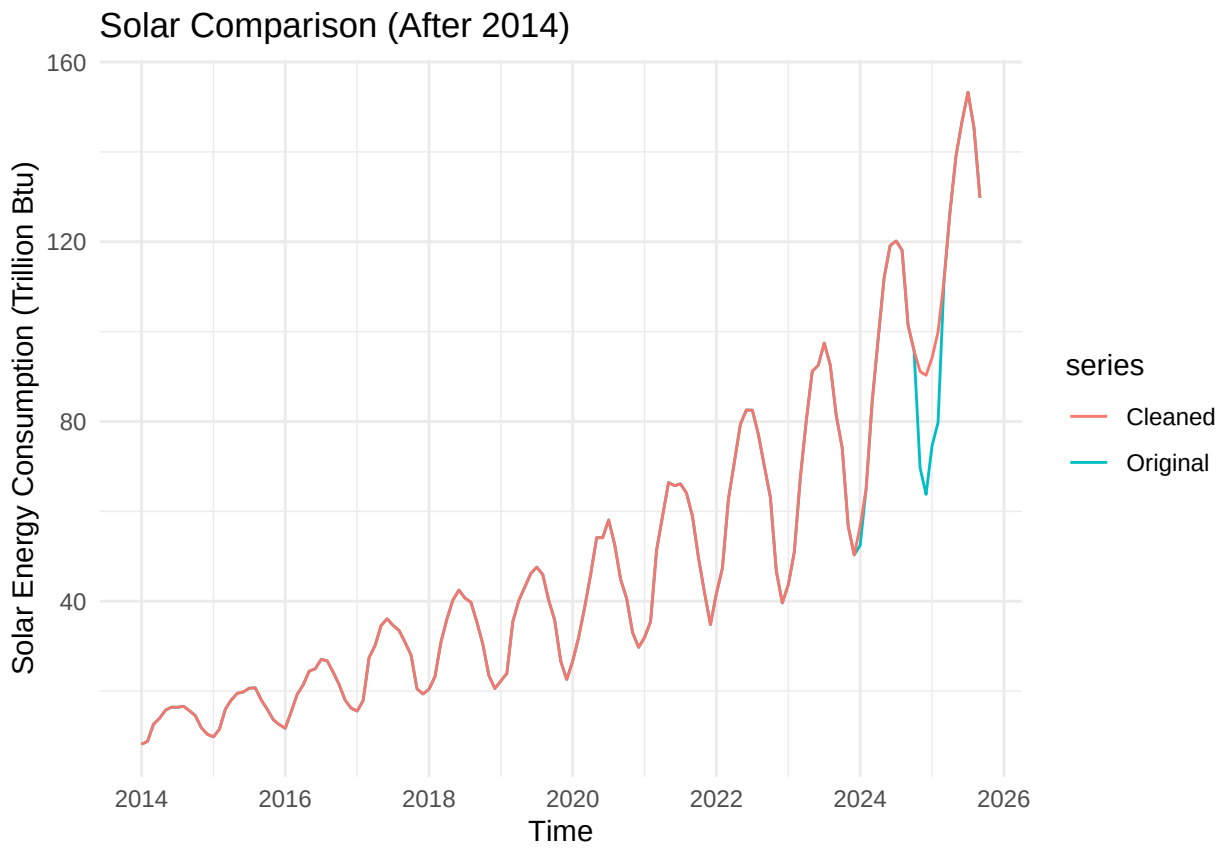



**Yes, the tsclean function removes outliers from both series.**

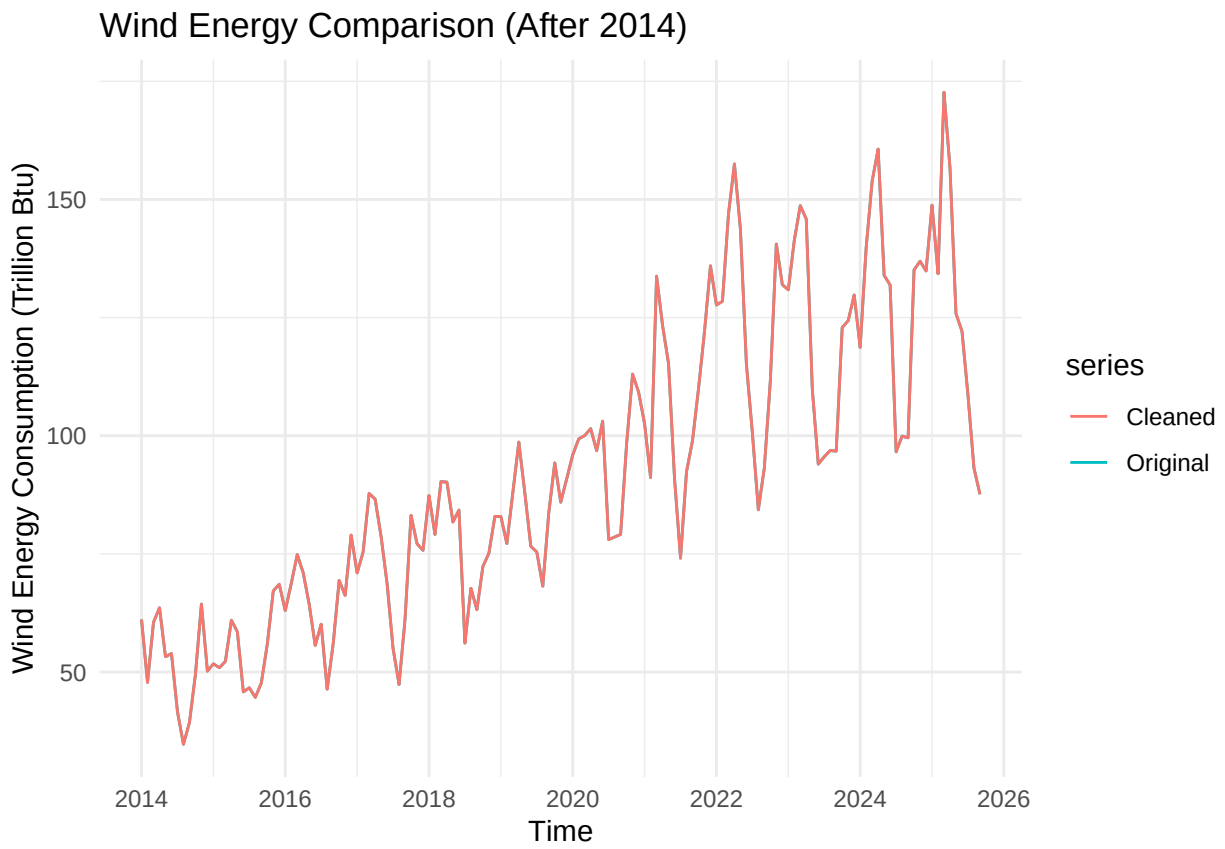
### Q9

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?

```
solar_clean_2014 <- tsclean(solar_ts_2014)
wind_clean_2014  <- tsclean(wind_ts_2014)

autoplot(solar_ts_2014, series = "Original") +
  autolayer(solar_clean_2014, series = "Cleaned") +
  ggtitle("Solar Comparison (After 2014)") +
  ylab("Solar Energy Consumption (Trillion Btu)") +
  theme_minimal()
```




```
# Compare Wind
autoplot(wind_ts_2014, series = "Original") +
  autolayer(wind_clean_2014, series = "Cleaned") +
  ggtitle("Wind Energy Comparison (After 2014)") +
  ylab("Wind Energy Consumption (Trillion Btu)") +
  theme_minimal()
```




```
sum(abs(solar_ts_2014 - solar_clean_2014) > 1e-10)
```

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

```
sum(abs(wind_ts_2014 - wind_clean_2014) > 1e-10)
```

```
## [1] 0
```

Answer: The solar series has 5 points that were adjusted by tsclean(), while the wind series has 0 changes. This matches what we see in the plots. For wind, the cleaned and original lines completely overlap. For solar, there are only a few very small adjustments.