Assignment 3

Group 4

2024-11-08

Part 1

Task A

Faults, adjusting clock summer (+02:00) time winter time (+01:00)

handle it by setting UTC as standard to transform to in the ymd_hms function.

```
power_consum_raw <- read.table("consumption_per_group_aas_hour.csv", sep=";", dec=",", header= T)
colnames(power_consum_raw)
```

```
## [1] "STARTTID"          "SLUTTID"           "KOMMUNE"           "FORBRUKSGRUPPE"
## [5] "VOLUM_KWH"         "ANTALL_MÅLEPUNKT"
```

```
power_consum_df <- power_consum_raw  %>%
  dplyr::select(c("STARTTID", "FORBRUKSGRUPPE", "VOLUM_KWH")) %>%
  mutate(STARTTID = ymd_hms(STARTTID, tz = "UTC"))

summary(power_consum_df)
```

```
##     STARTTID                     FORBRUKSGRUPPE       VOLUM_KWH
##  Min.   :2021-03-31 22:00:00.00  Length:90231       Min.   :  862.2
##  1st Qu.:2022-03-06 00:00:00.00  Class :character   1st Qu.: 3682.7
##  Median :2023-01-13 07:00:00.00  Mode  :character   Median : 8751.0
##  Mean   :2023-01-13 04:55:16.93                     Mean   : 9307.1
##  3rd Qu.:2023-11-22 14:00:00.00                     3rd Qu.:12655.9
##  Max.   :2024-09-30 21:00:00.00                     Max.   :36386.2
```

```
head(power_consum_df)
```

```
##              STARTTID FORBRUKSGRUPPE VOLUM_KWH
## 1 2021-03-31 22:00:00     Forretning  9883.478
## 2 2021-03-31 22:00:00       Industri  2848.522
## 3 2021-03-31 22:00:00         Privat 15463.043
## 4 2021-03-31 23:00:00     Forretning  9867.467
## 5 2021-03-31 23:00:00       Industri  2687.639
## 6 2021-03-31 23:00:00         Privat 14847.819
```

```
power_consum_df_long <- power_consum_df %>%
  pivot_wider(names_from = FORBRUKSGRUPPE, values_from = VOLUM_KWH)

data_range <- seq(min(power_consum_df_long$STARTTID), max(power_consum_df_long$STARTTID), by = "1 hour")

last_date_gap <- tail(data_range[!data_range %in% power_consum_df_long$STARTTID], 1)
```

```r
power_consum_df_long_cut <- power_consum_df_long %>%
  dplyr::filter(STARTTID >= last_date_gap)

head(power_consum_df_long_cut)
```

```
## # A tibble: 6 x 4
##   STARTTID            Forretning Industri Privat
##   <dttm>                   <dbl>    <dbl>  <dbl>
## 1 2021-04-30 22:00:00      8904.    2444. 13809.
## 2 2021-04-30 23:00:00      8934.    2412. 12894.
## 3 2021-05-01 00:00:00      8766.    2464. 12543.
## 4 2021-05-01 01:00:00      8967.    2548. 12502.
## 5 2021-05-01 02:00:00      9067.    2309. 12622.
## 6 2021-05-01 03:00:00      8930.    2334. 12794.
```

```r
head(power_consum_df_long_cut)
```

```
## # A tibble: 6 x 4
##   STARTTID            Forretning Industri Privat
##   <dttm>                   <dbl>    <dbl>  <dbl>
## 1 2021-04-30 22:00:00      8904.    2444. 13809.
## 2 2021-04-30 23:00:00      8934.    2412. 12894.
## 3 2021-05-01 00:00:00      8766.    2464. 12543.
## 4 2021-05-01 01:00:00      8967.    2548. 12502.
## 5 2021-05-01 02:00:00      9067.    2309. 12622.
## 6 2021-05-01 03:00:00      8930.    2334. 12794.
```

```r
tail(power_consum_df_long_cut)
```

```
## # A tibble: 6 x 4
##   STARTTID            Forretning Industri Privat
##   <dttm>                   <dbl>    <dbl>  <dbl>
## 1 2024-09-30 16:00:00      9972.    3281. 15494.
## 2 2024-09-30 17:00:00      9744.    3251. 15659.
## 3 2024-09-30 18:00:00      9294.    2844. 15680.
## 4 2024-09-30 19:00:00      8797.    2624. 15352.
## 5 2024-09-30 20:00:00      8495.    2593. 14576.
## 6 2024-09-30 21:00:00      7894.    2528. 13560.
```

```r
# Mannualy remove the head and tail that don't sum to 1 whole day

power_consum_df_long_cut_cutDay <- power_consum_df_long_cut %>%
  dplyr::filter(STARTTID > ymd_hms("2021-04-30 23:00:00 UTC")) %>%
  dplyr::filter(STARTTID < ymd_hms("2024-09-30 00:00:00 UTC"))
head(power_consum_df_long_cut_cutDay)
```

```
## # A tibble: 6 x 4
##   STARTTID            Forretning Industri Privat
##   <dttm>                   <dbl>    <dbl>  <dbl>
## 1 2021-05-01 00:00:00      8766.    2464. 12543.
## 2 2021-05-01 01:00:00      8967.    2548. 12502.
## 3 2021-05-01 02:00:00      9067.    2309. 12622.
## 4 2021-05-01 03:00:00      8930.    2334. 12794.
## 5 2021-05-01 04:00:00      8926.    2174. 13112.
## 6 2021-05-01 05:00:00      8841.    1941. 13587.
```

```r
tail(power_consum_df_long_cut_cutDay, 24)
```

```
## # A tibble: 24 x 4
##    STARTTID            Forretning Industri Privat
##    <dttm>                   <dbl>    <dbl>  <dbl>
##  1 2024-09-29 00:00:00      7809.    2039. 12014.
##  2 2024-09-29 01:00:00      8003.    2037. 11811.
##  3 2024-09-29 02:00:00      8082.    2034. 11777.
##  4 2024-09-29 03:00:00      8103.    2057. 11791.
##  5 2024-09-29 04:00:00      8435.    2079. 12148.
##  6 2024-09-29 05:00:00      8496.    2001. 12676.
##  7 2024-09-29 06:00:00      8600.    1507. 13679.
##  8 2024-09-29 07:00:00      8095.    1257. 14269.
##  9 2024-09-29 08:00:00      8140.    1255. 14333.
## 10 2024-09-29 09:00:00      9174.    1212. 14255.
## # i 14 more rows
```

```r
power_consum_sum_df <- power_consum_df_long_cut_cutDay %>%
  mutate(STARTTID = as.Date(STARTTID)) %>%
  group_by(STARTTID) %>%
  summarise_each(funs(sum))
```

```
## Warning: `summarise_each()` was deprecated in dplyr 0.7.0.
## i Please use `across()` instead.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: `funs()` was deprecated in dplyr 0.8.0.
## i Please use a list of either functions or lambdas:
##
## # Simple named list: list(mean = mean, median = median)
##
## # Auto named with `tibble::lst()`: tibble::lst(mean, median)
##
## # Using lambdas list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
head(power_consum_sum_df)
```

```
## # A tibble: 6 x 4
##   STARTTID   Forretning Industri  Privat
##   <date>          <dbl>    <dbl>   <dbl>
## 1 2021-05-01    205741.   50381. 339827.
## 2 2021-05-02    204183.   50310. 329856.
## 3 2021-05-03    259292.   82488. 329644.
## 4 2021-05-04    270244.   84911. 363652.
## 5 2021-05-05    293115.   91995. 408617.
## 6 2021-05-06    275699.   82163. 380373.
```

Final clean

```r
power_consum_df_c <- power_consum_sum_df
```

Task B

Data is daily.

Data registered on different times, might skew results.

In the excell file the dates are stored in this format '01/01/2020 22:10:00'

Relevant years: 2017-2024:

-

```
file_path <- getwd()
excel_files <- list.files(path = file_path, pattern = "*.xlsx", full.names = TRUE)
```

Data formats that might be problematic, some non unique dato. Leap years. FIles DATE formated differently, might cause issues if the read_excel function does not account for it.

```
metro_aas_df <- excel_files %>%
  lapply(read_excel) %>%
  bind_rows() %>%
  dplyr::select(c("DATO","LT", "GLOB")) %>%
  mutate(DATO = as.Date(DATO)) %>%
  group_by(DATO) %>%
  distinct(DATO, .keep_all = TRUE) %>%
  ungroup() %>%
  complete(DATO = seq(min(DATO), max(DATO), by = 'day')) # Add missing days
```

```
## New names:
## * `` -> `...30`
```

```
summary(metro_aas_df)
```

```
##       DATO                  LT               GLOB
##  Min.   :2017-01-01   Min.   :-20.111   Min.   : 0.09481
##  1st Qu.:2018-12-12   1st Qu.:  1.086   1st Qu.: 2.14463
##  Median :2020-11-22   Median :  7.049   Median : 7.76017
##  Mean   :2020-11-22   Mean   :  7.121   Mean   :10.33553
##  3rd Qu.:2022-11-02   3rd Qu.: 14.142   3rd Qu.:17.39571
##  Max.   :2024-10-13   Max.   : 24.593   Max.   :31.88862
##                       NA's   :3         NA's   :11
```

```
head(metro_aas_df)
```

```
## # A tibble: 6 x 3
##   DATO            LT  GLOB
##   <date>       <dbl> <dbl>
## 1 2017-01-01   0.543 1.34
## 2 2017-01-02  -4.44  1.48
## 3 2017-01-03  -2.37  0.794
## 4 2017-01-04  -3.18  0.909
## 5 2017-01-05 -10.8   1.85
## 6 2017-01-06  -4.18  0.530
```

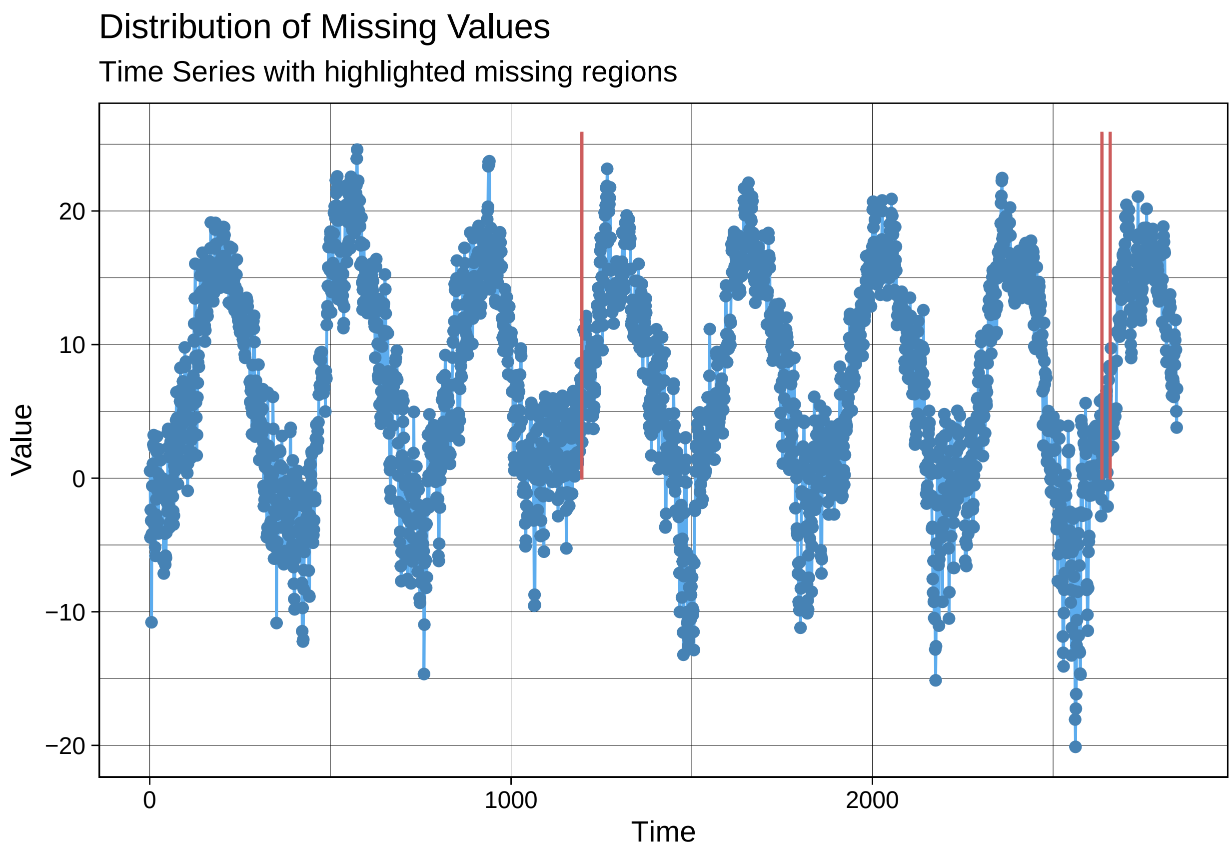
```
max(metro_aas_df$DATO)
```

```
## [1] "2024-10-13"
```

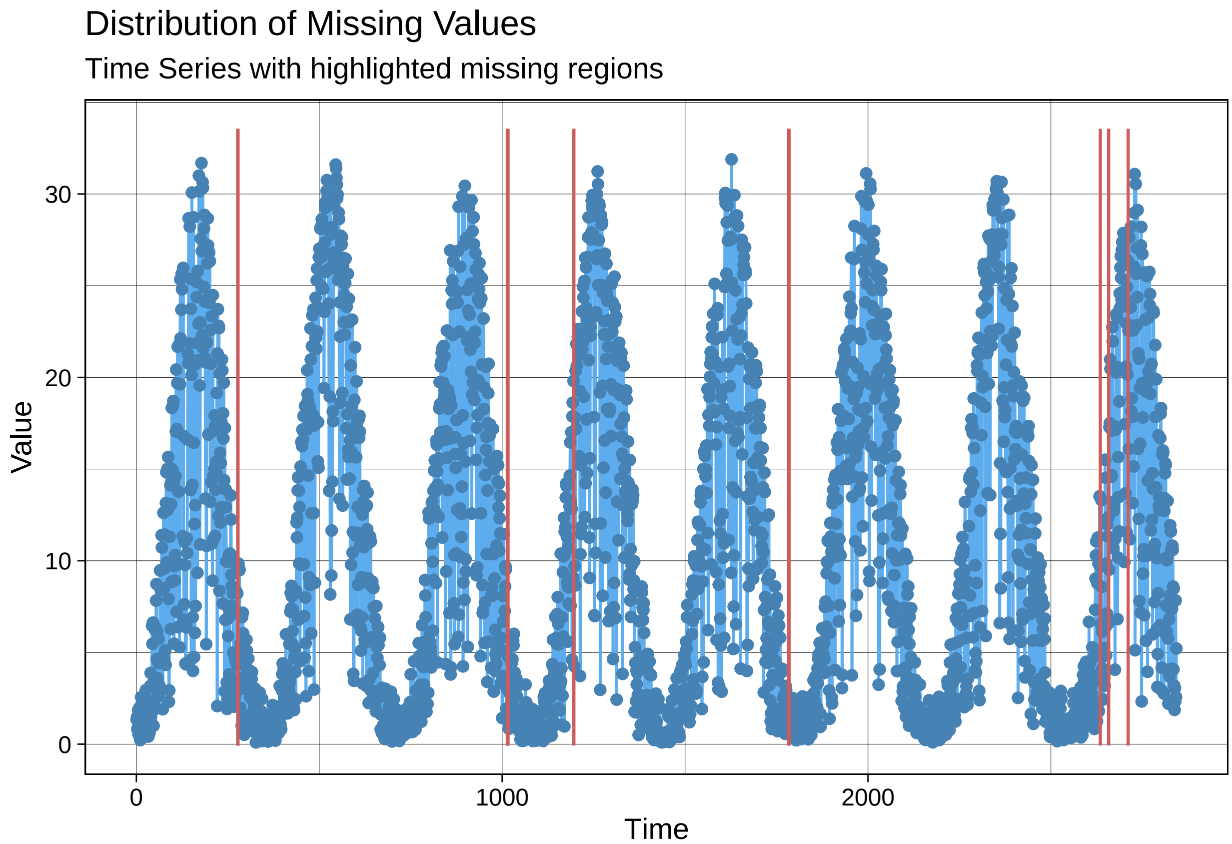
Missing data analysis

```r
metro_aas_df_c <- metro_aas_df
```

```r
ggplot_na_distribution(metro_aas_df_c$LT)
```




```r
ggplot_na_distribution(metro_aas_df_c$GLOB)
```

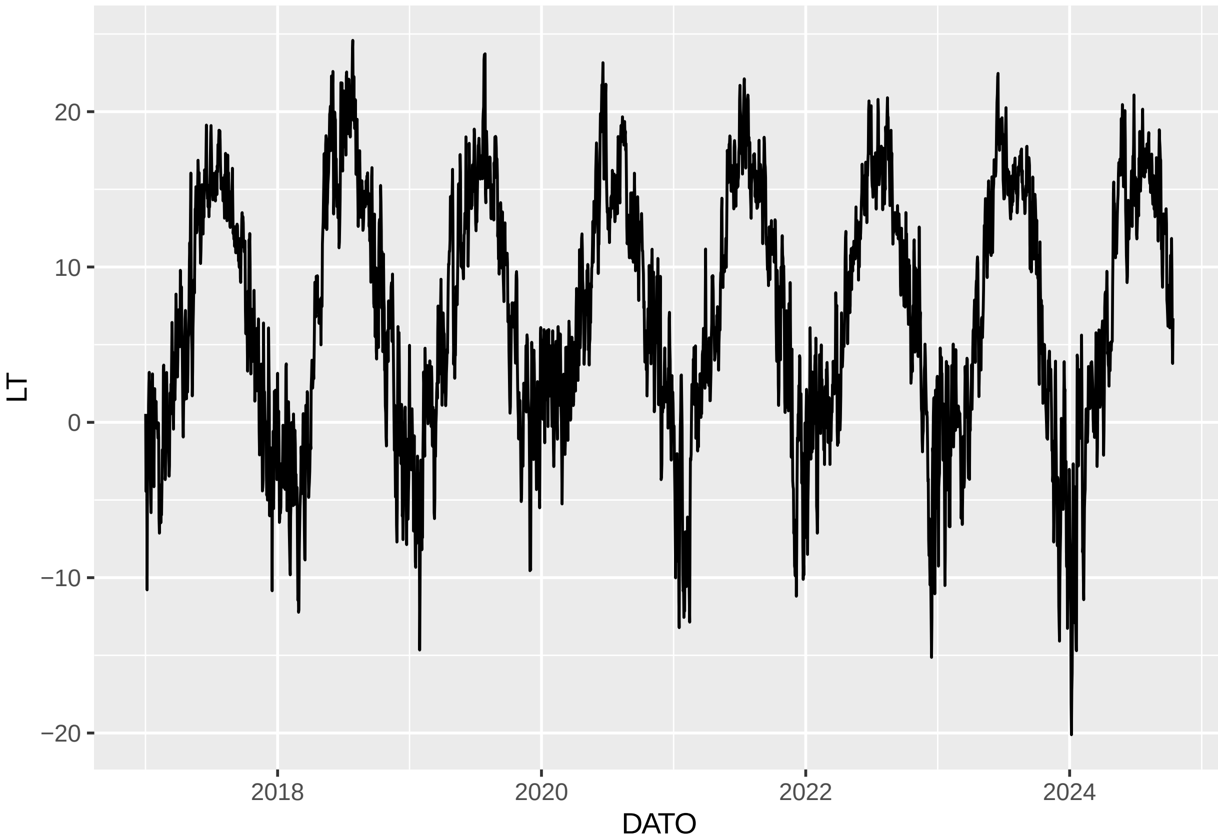


As seen in the previous graphs, there are very few missing values and they seem evenly spread out, therfore imputations seems to be a good choice to fill in the missing days.

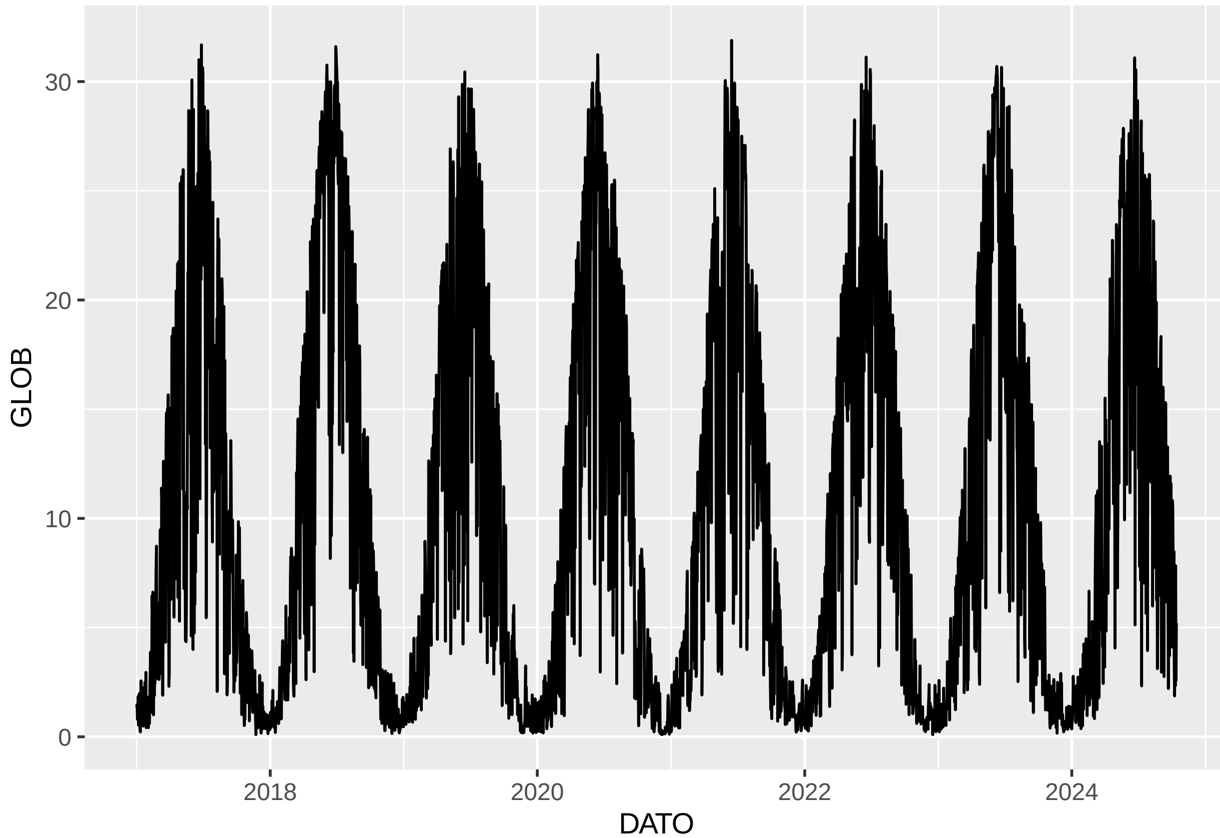
```
metro_aas_df_imputed <- imputeTS::na_ma(metro_aas_df_c , k = 4, weighting = "simple")
head(metro_aas_df_imputed)
```

```
## # A tibble: 6 x 3
##   DATO            LT  GLOB
##   <date>       <dbl> <dbl>
## 1 2017-01-01   0.543 1.34
## 2 2017-01-02  -4.44  1.48
## 3 2017-01-03  -2.37  0.794
## 4 2017-01-04  -3.18  0.909
## 5 2017-01-05 -10.8   1.85
## 6 2017-01-06  -4.18  0.530
```

```
metro_aas_df_imputed %>%
  ggplot(aes(DATO, LT)) +
  geom_line()
```




```r
metro_aas_df_imputed %>%
  ggplot(aes(DATO, GLOB)) +
  geom_line()
```



Final cleaned data

```
metro_aas_df_c <- metro_aas_df_imputed
```

Task C

Find the range of dates for each of the two data sets.

```
power_consum_dateRange <- c(min(power_consum_df_c$STARTTID), max(power_consum_df_c$STARTTID))
metro_ass_dateRange <- c(min(metro_aas_df_c$DATO), max(metro_aas_df_c$DATO))
power_consum_dateRange
```

```
## [1] "2021-05-01" "2024-09-29"
```

```
metro_ass_dateRange
```

```
## [1] "2017-01-01" "2024-10-13"
```

For the longest contiguous range of dates present in both data sets, merge the two data sets based on date.

```
merged_df <- dplyr::inner_join(power_consum_df_c, metro_aas_df_c, by = join_by(STARTTID == DATO)) %>%
  rename(DATO = STARTTID)

head(merged_df)
```

```
## # A tibble: 6 x 6
##   DATO       Forretning Industri  Privat    LT  GLOB
```

```
##   <date>          <dbl>    <dbl>   <dbl> <dbl> <dbl>
## 1 2021-05-01    205741.   50381. 339827.  6.28 20.9
## 2 2021-05-02    204183.   50310. 329856.  7.45 22.1
## 3 2021-05-03    259292.   82488. 329644.  5.97 16.3
## 4 2021-05-04    270244.   84911. 363652.  5.06  9.37
## 5 2021-05-05    293115.   91995. 408617.  3.36  5.69
## 6 2021-05-06    275699.   82163. 380373.  5.11 17.3
```

Remove data for leap days.

```r
merged_df_noLeap <- merged_df %>%
  dplyr::filter(!(month(DATO) == 2 & day(DATO) == 29))

head(merged_df_noLeap)
```

```
## # A tibble: 6 x 6
##   DATO       Forretning Industri  Privat    LT  GLOB
##   <date>          <dbl>    <dbl>   <dbl> <dbl> <dbl>
## 1 2021-05-01    205741.   50381. 339827.  6.28 20.9
## 2 2021-05-02    204183.   50310. 329856.  7.45 22.1
## 3 2021-05-03    259292.   82488. 329644.  5.97 16.3
## 4 2021-05-04    270244.   84911. 363652.  5.06  9.37
## 5 2021-05-05    293115.   91995. 408617.  3.36  5.69
## 6 2021-05-06    275699.   82163. 380373.  5.11 17.3
```

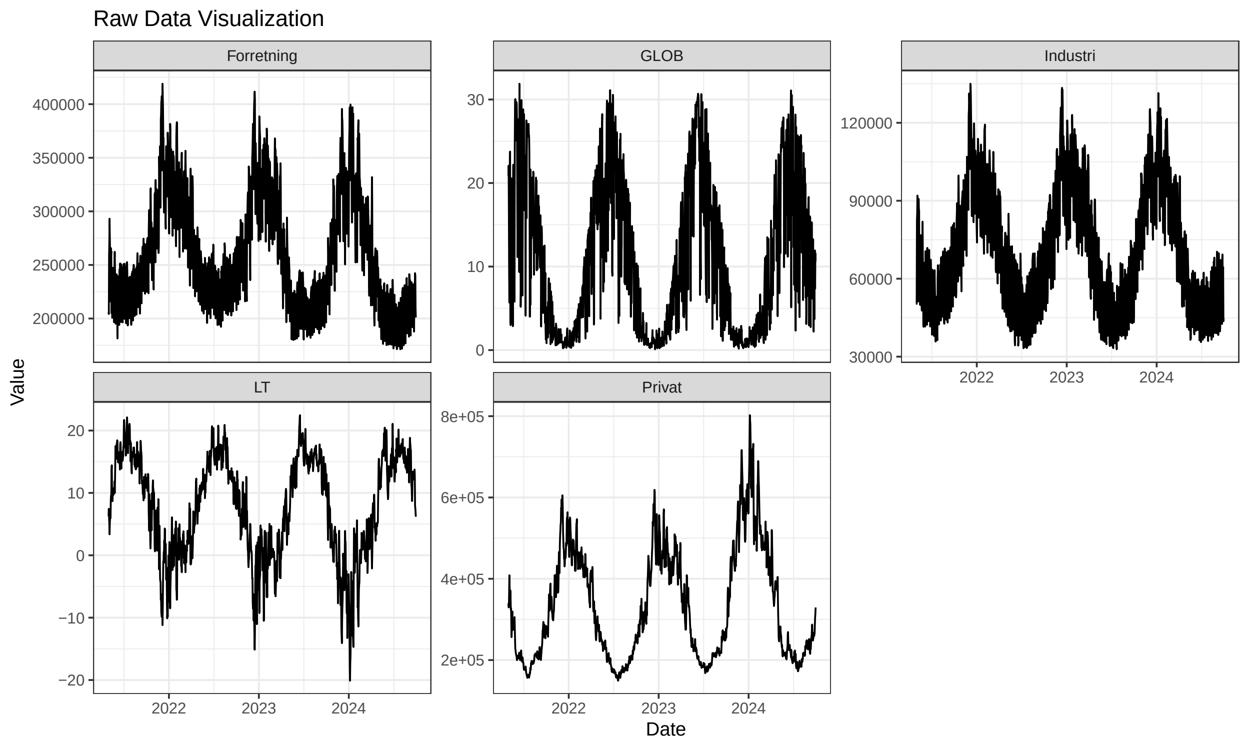
```r
merged_data <- merged_df_noLeap
```

Part 2

Task D

1. Data Visualization Forretning, Industri, Privat are all VOLUME_KWH

```r
# Plot raw data visualization
merged_data %>%
  pivot_longer(cols = c(Forretning, Industri, Privat, LT, GLOB), names_to = "Measurement", values_to =
  ggplot(aes(x = DATO, y = Value)) +
  geom_line() +
  facet_wrap(~Measurement, scales = "free_y") +
  theme_bw() +
  labs(title = "Raw Data Visualization", x = "Date", y = "Value") +
  theme(legend.position = "bottom")
```




```r
test_stationarity <- function(data, group_name) {
  print(paste("Stationarity Test for", group_name))
  # Handle NAs (Important: Use na.remove *inside* the function)
  data <- na.remove(data)
  adf.test(data) %>% print()  # Augmented Dickey-Fuller Test
  kpss.test(data) %>% print()  # KPSS Test
  cat("\n") # Newline for better output formatting
}


ts_Forr <- ts(merged_data$Forretning, frequency = 365)
test_stationarity(ts_Forr, "Forretning KWH")
```

2. Stationarity Tests

```
## [1] "Stationarity Test for Forretning KWH"
##
##  Augmented Dickey-Fuller Test
##
## data:  data
## Dickey-Fuller = -2.7124, Lag order = 10, p-value = 0.2768
## alternative hypothesis: stationary

## Warning in kpss.test(data): p-value smaller than printed p-value

##
##  KPSS Test for Level Stationarity
##
## data:  data
## KPSS Level = 0.99076, Truncation lag parameter = 7, p-value = 0.01
```

```r
ts_Indu <- ts(merged_data$Industri, frequency = 365)
test_stationarity(ts_Forr, "Industri KWH")
```

```
## [1] "Stationarity Test for Industri KWH"
##
##  Augmented Dickey-Fuller Test
##
## data:  data
## Dickey-Fuller = -2.7124, Lag order = 10, p-value = 0.2768
## alternative hypothesis: stationary

## Warning in kpss.test(data): p-value smaller than printed p-value

##
##  KPSS Test for Level Stationarity
##
## data:  data
## KPSS Level = 0.99076, Truncation lag parameter = 7, p-value = 0.01
```

```r
ts_Priv <- ts(merged_data$Forretning, frequency = 365)
test_stationarity(ts_Priv, "Privat KWH")
```

```
## [1] "Stationarity Test for Privat KWH"
##
##  Augmented Dickey-Fuller Test
##
## data:  data
## Dickey-Fuller = -2.7124, Lag order = 10, p-value = 0.2768
## alternative hypothesis: stationary

## Warning in kpss.test(data): p-value smaller than printed p-value

##
##  KPSS Test for Level Stationarity
##
## data:  data
## KPSS Level = 0.99076, Truncation lag parameter = 7, p-value = 0.01
```

```r
ts_temp <- ts(merged_data$LT, frequency = 365)
test_stationarity(ts_temp, "Temperature (LT)")
```

```
## [1] "Stationarity Test for Temperature (LT)"
##
##  Augmented Dickey-Fuller Test
##
## data:  data
## Dickey-Fuller = -2.4565, Lag order = 10, p-value = 0.3851
## alternative hypothesis: stationary
##
##
##  KPSS Test for Level Stationarity
##
## data:  data
## KPSS Level = 0.54711, Truncation lag parameter = 7, p-value = 0.03106
```

```r
ts_glob <- ts(merged_data$GLOB, frequency = 365)
test_stationarity(ts_glob, "Global Irradiation (GLOB)")
```

```
## [1] "Stationarity Test for Global Irradiation (GLOB)"
##
##  Augmented Dickey-Fuller Test
##
## data:  data
## Dickey-Fuller = -2.5149, Lag order = 10, p-value = 0.3603
## alternative hypothesis: stationary
##
##
##  KPSS Test for Level Stationarity
##
## data:  data
## KPSS Level = 0.41001, Truncation lag parameter = 7, p-value = 0.07284
```

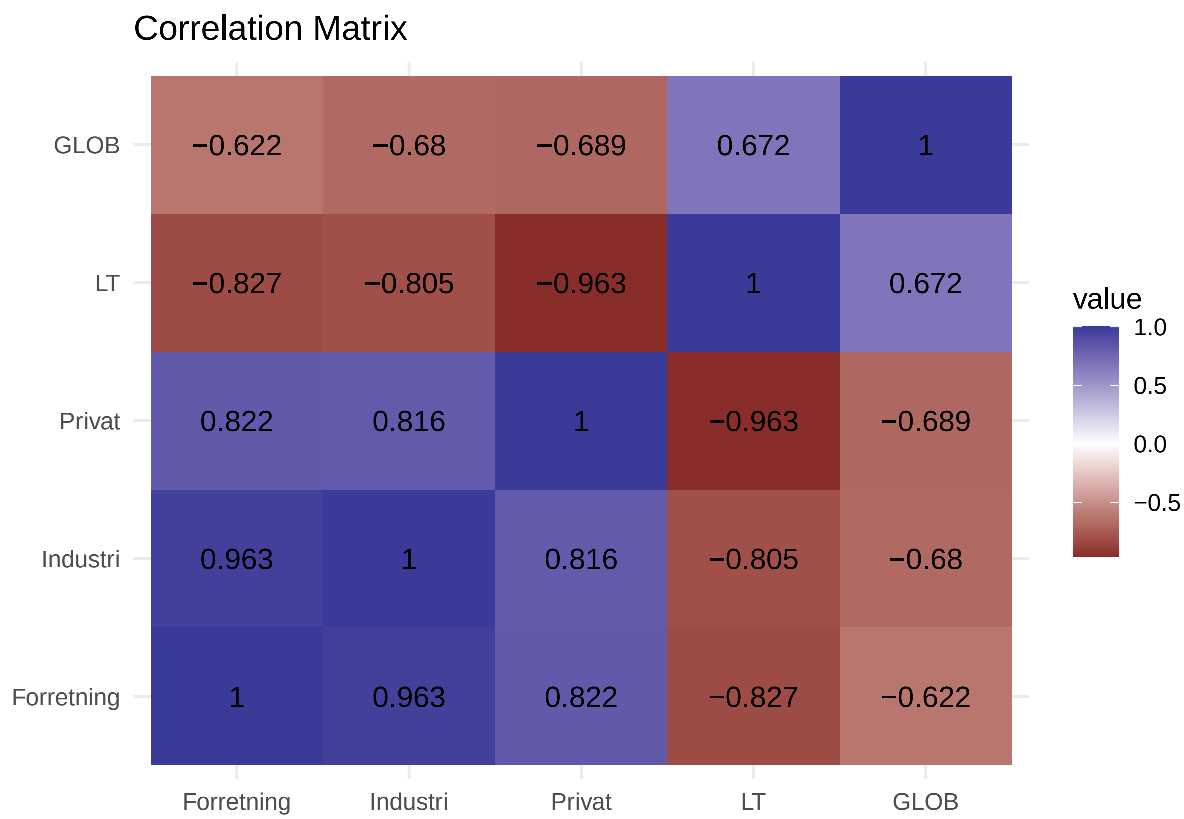
```
cor_matrix <- cor(merged_data %>% select(where(is.numeric)), use = "pairwise.complete.obs")
print(cor_matrix)
```

3. Cross-Correlation Analysis

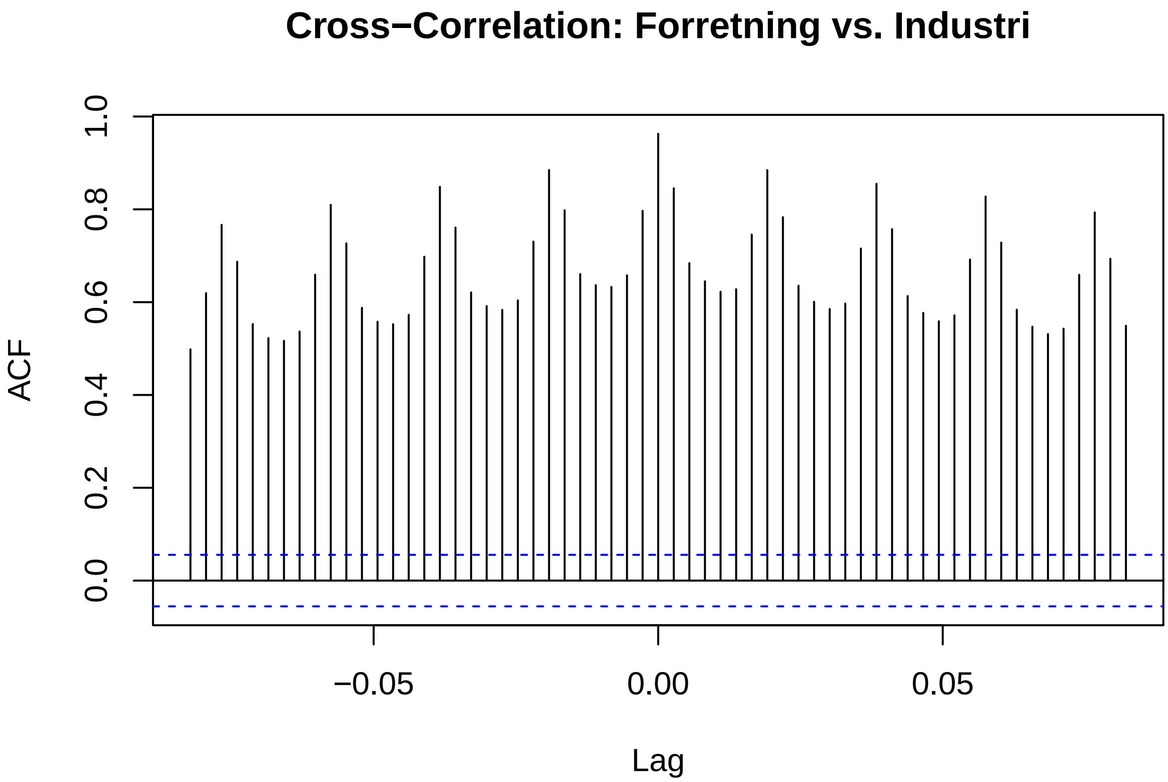
```
##             Forretning   Industri     Privat         LT       GLOB
## Forretning  1.0000000  0.9628401  0.8223101 -0.8274985 -0.6222319
## Industri    0.9628401  1.0000000  0.8164847 -0.8050398 -0.6797756
## Privat      0.8223101  0.8164847  1.0000000 -0.9631797 -0.6887294
## LT         -0.8274985 -0.8050398 -0.9631797  1.0000000  0.6716545
## GLOB       -0.6222319 -0.6797756 -0.6887294  0.6716545  1.0000000
```

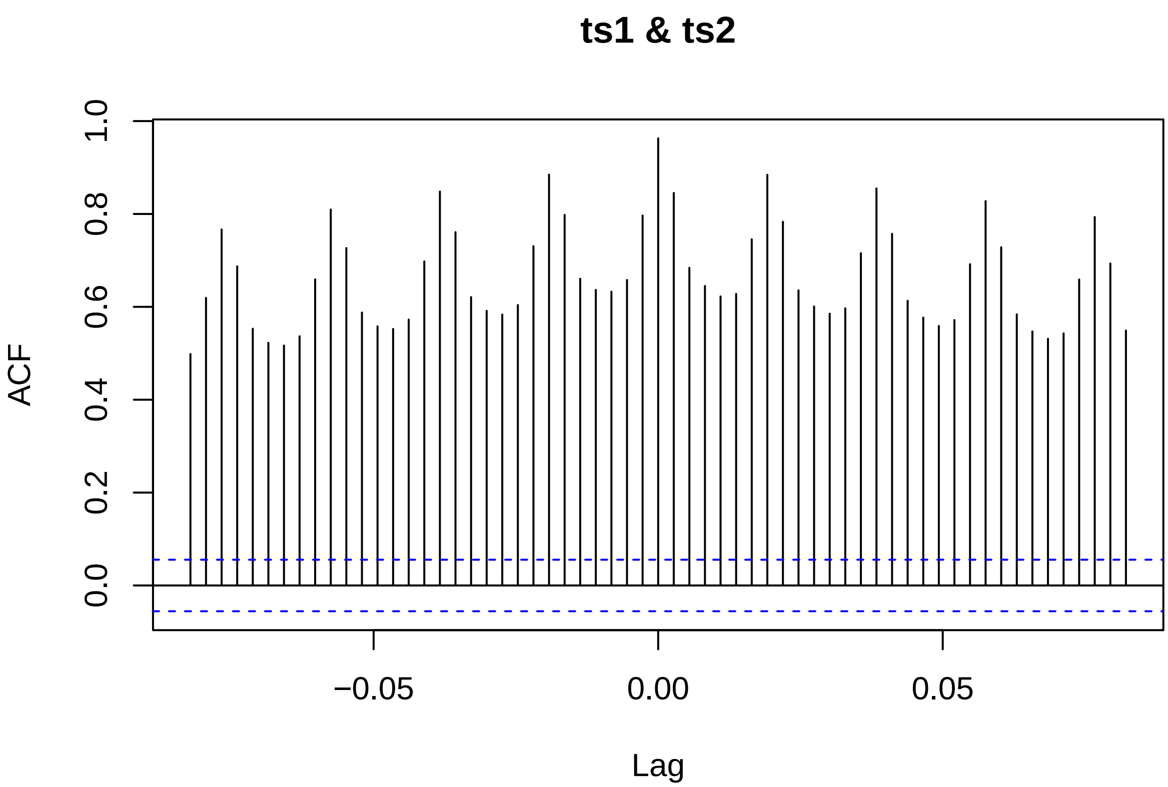
```
melt(cor_matrix) %>%
  ggplot(aes(Var2, Var1)) +
  geom_tile(aes(fill = value)) +
  geom_text(aes(fill = value, label = round(value, 3))) +
  scale_fill_gradient2() +
  labs(title = "Correlation Matrix", x = "", y = "") +
  theme_minimal()
```

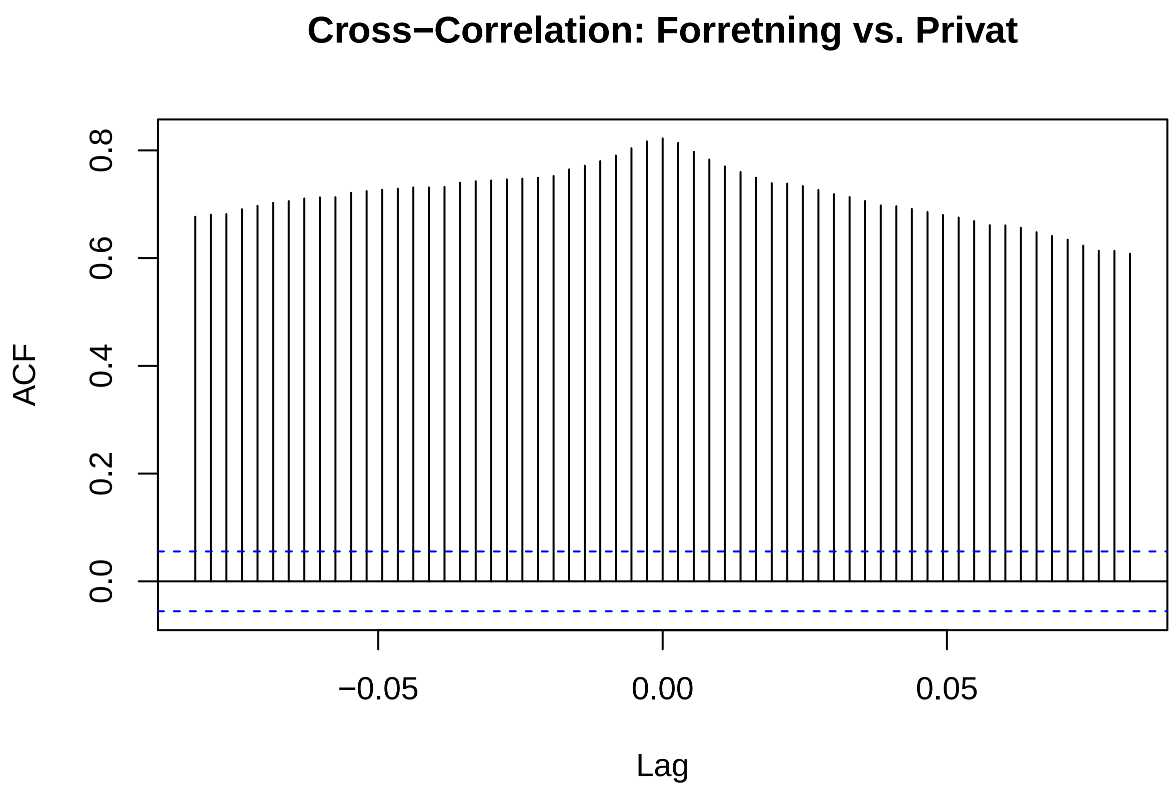
```
## Warning in geom_text(aes(fill = value, label = round(value, 3))): Ignoring
## unknown aesthetics: fill
```

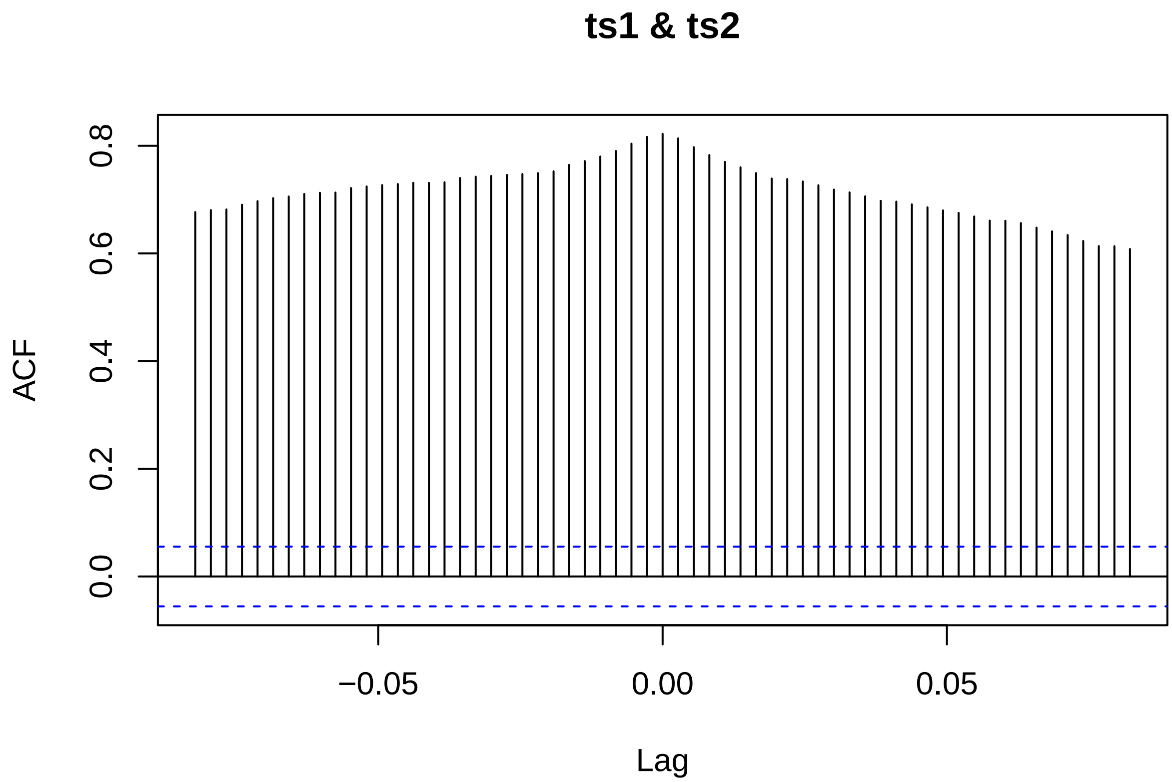


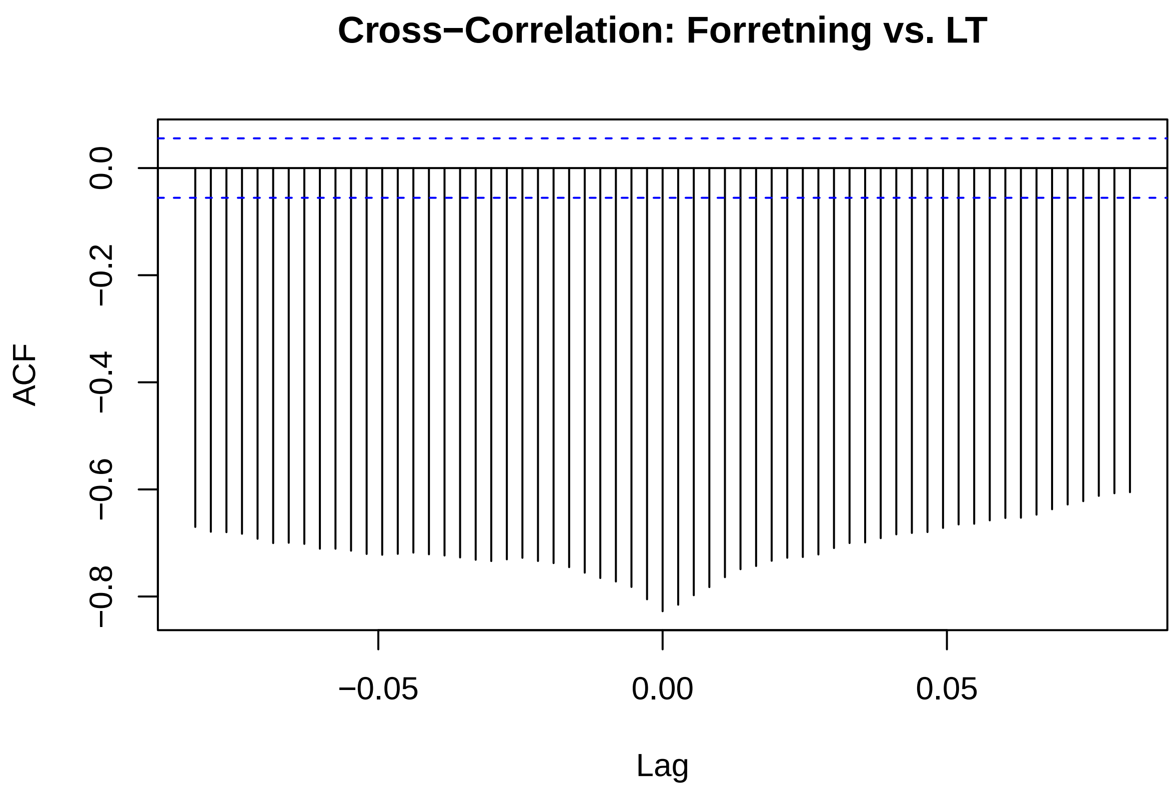

```r
numeric_cols <- names(merged_data)[sapply(merged_data, is.numeric) & names(merged_data) != "DATO"]

for (i in 1:(length(numeric_cols) - 1)) {
  for (j in (i + 1):length(numeric_cols)) {
    ts1 <- ts(merged_data[[numeric_cols[i]]], frequency = 365)
    ts2 <- ts(merged_data[[numeric_cols[j]]], frequency = 365)
    ccf_result <- ccf(ts1, ts2, lag.max = 30, na.action = na.contiguous,
                      main = paste("Cross-Correlation:", numeric_cols[i], "vs.", numeric_cols[j]))
    plot(ccf_result)
  }
}
```

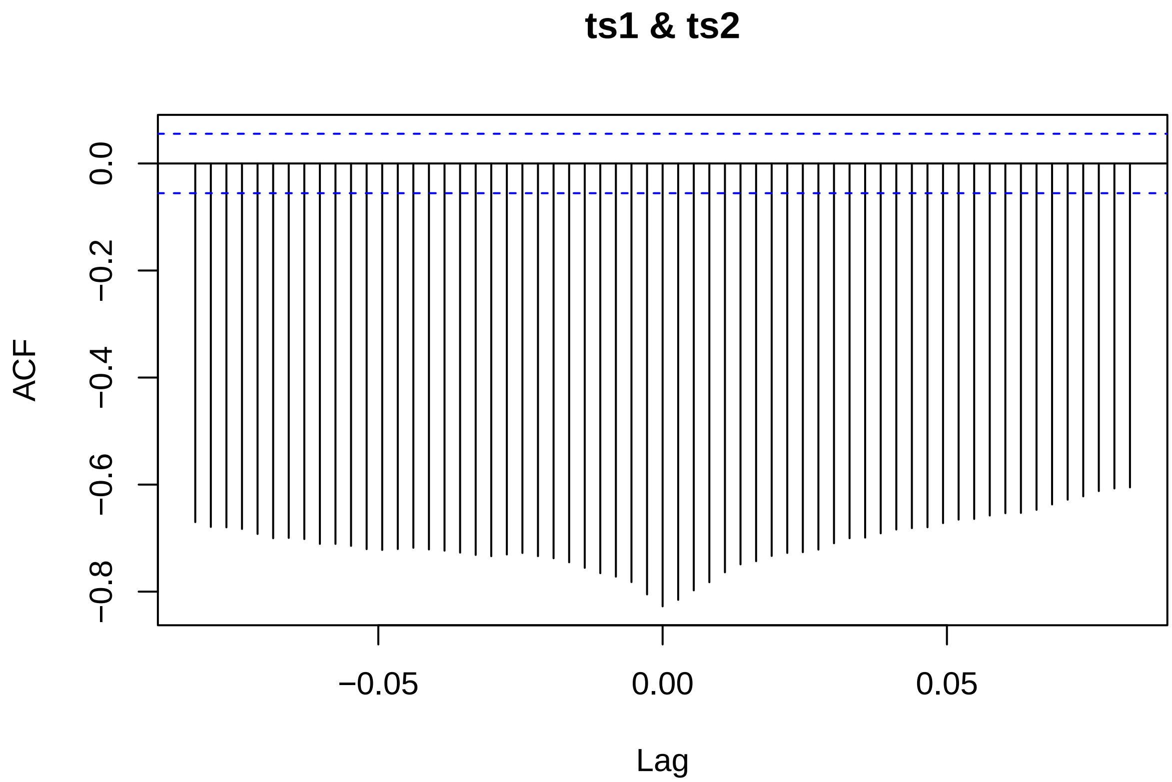

Cross−Correlation: Forretning vs. Industri
ACF
0.0
0.2
0.4
0.6
0.8
1.0
−0.05
0.00
0.05
Lag

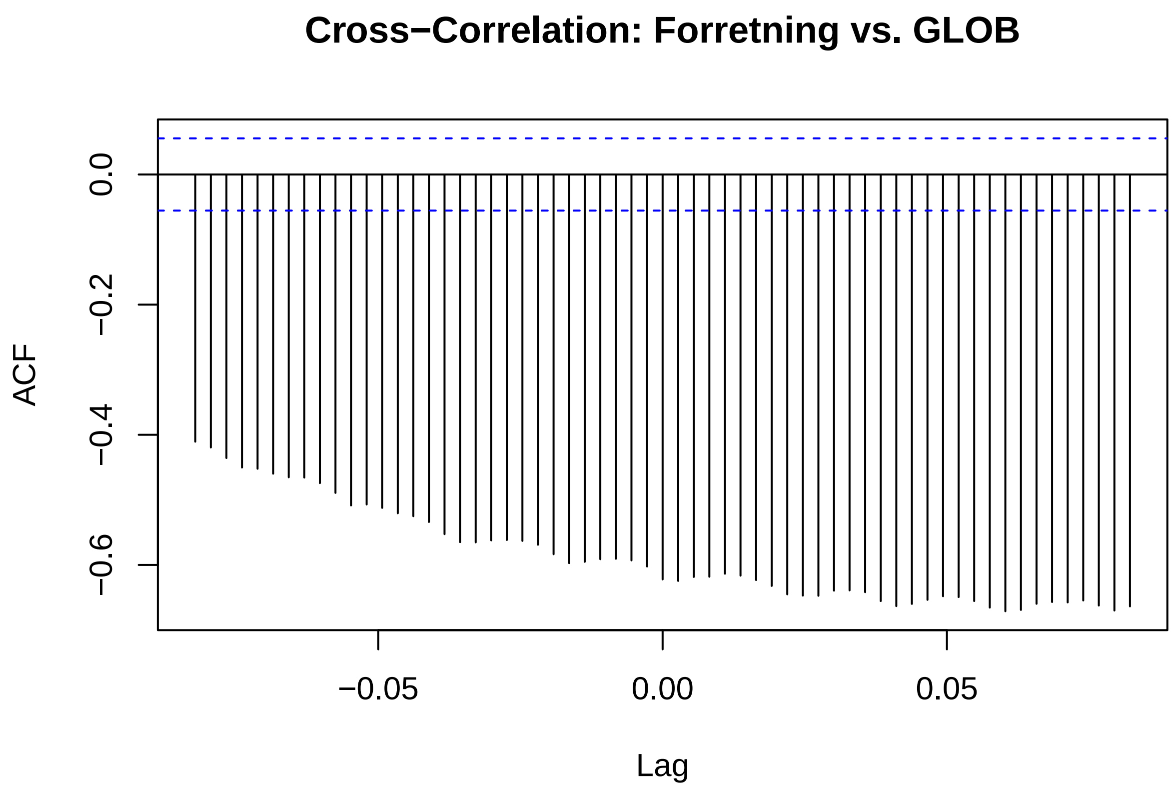

ts1 & ts2
ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.05
0.00
0.05
Lag


Cross−Correlation: Forretning vs. Privat
ACF
0.0
0.2
0.4
0.6
0.8
−0.05
0.00
0.05
Lag

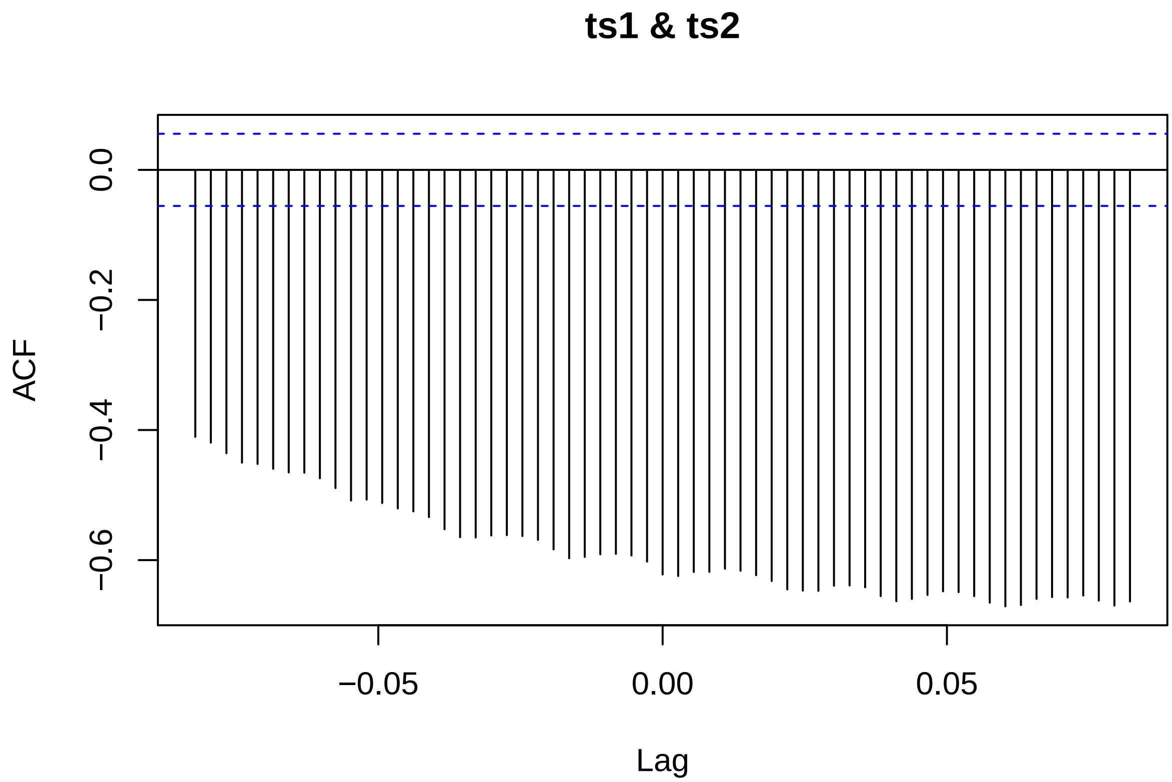

ts1 & ts2
ACF
0.0
0.2
0.4
0.6
0.8
−0.05
0.00
0.05
Lag

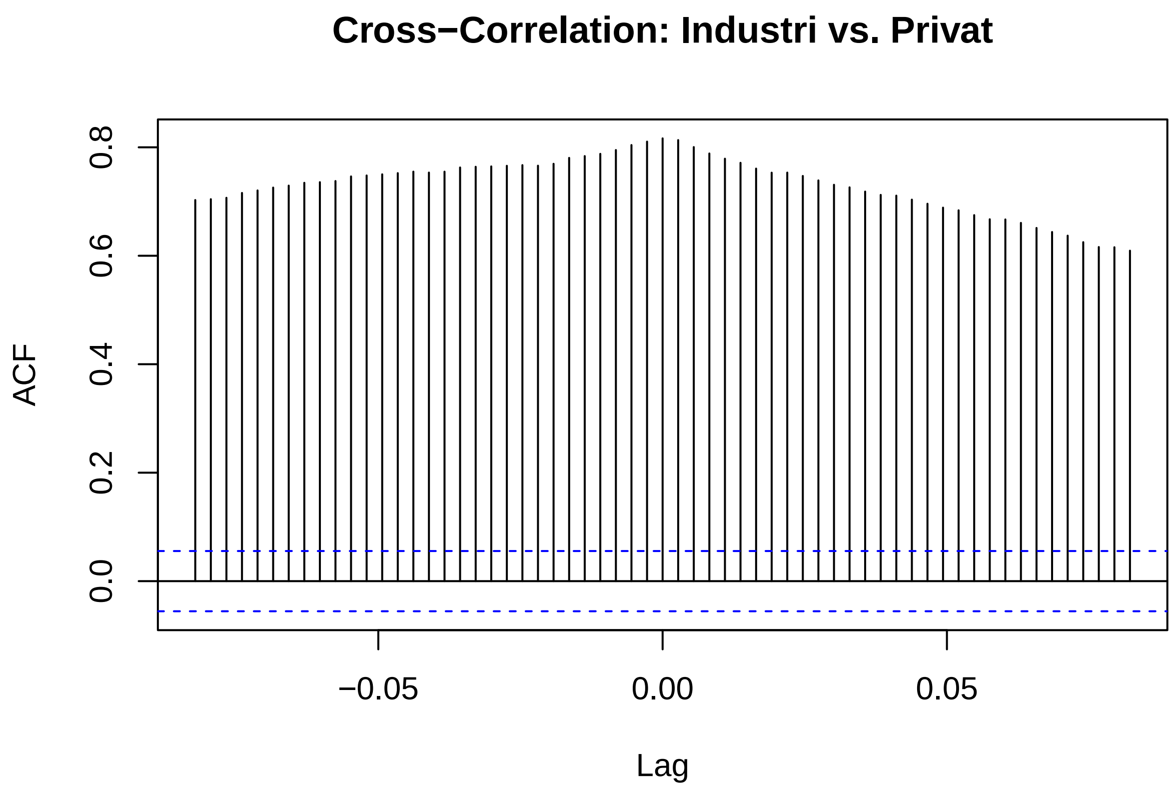

Cross−Correlation: Forretning vs. LT
ACF
0.0
−0.2
−0.4
−0.6
−0.8
−0.05
0.00
0.05
Lag


ts1 & ts2
ACF
0.0
−0.2
−0.4
−0.6
−0.8
−0.05
0.00
0.05
Lag

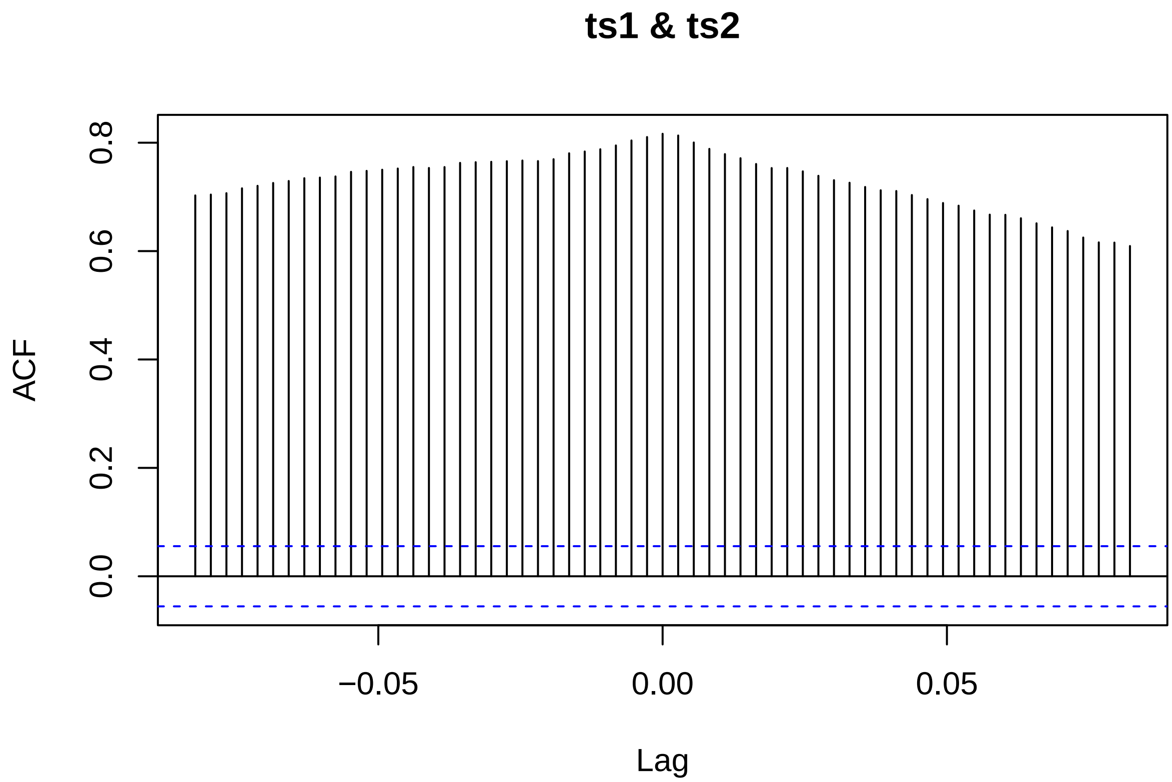


Cross−Correlation: Forretning vs. GLOB
ACF
0.0
−0.2
−0.4
−0.6
−0.05
0.00
0.05
Lag

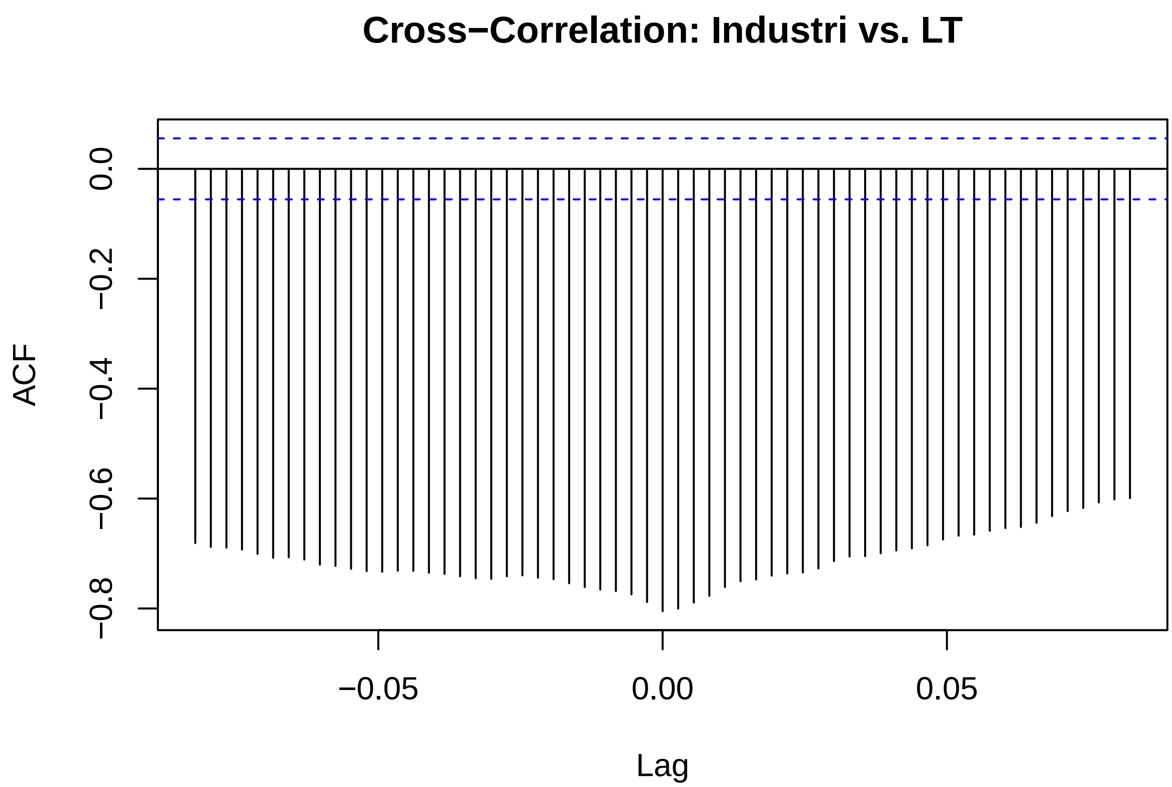

ts1 & ts2
ACF
0.0
−0.2
−0.4
−0.6
−0.05
0.00
0.05
Lag

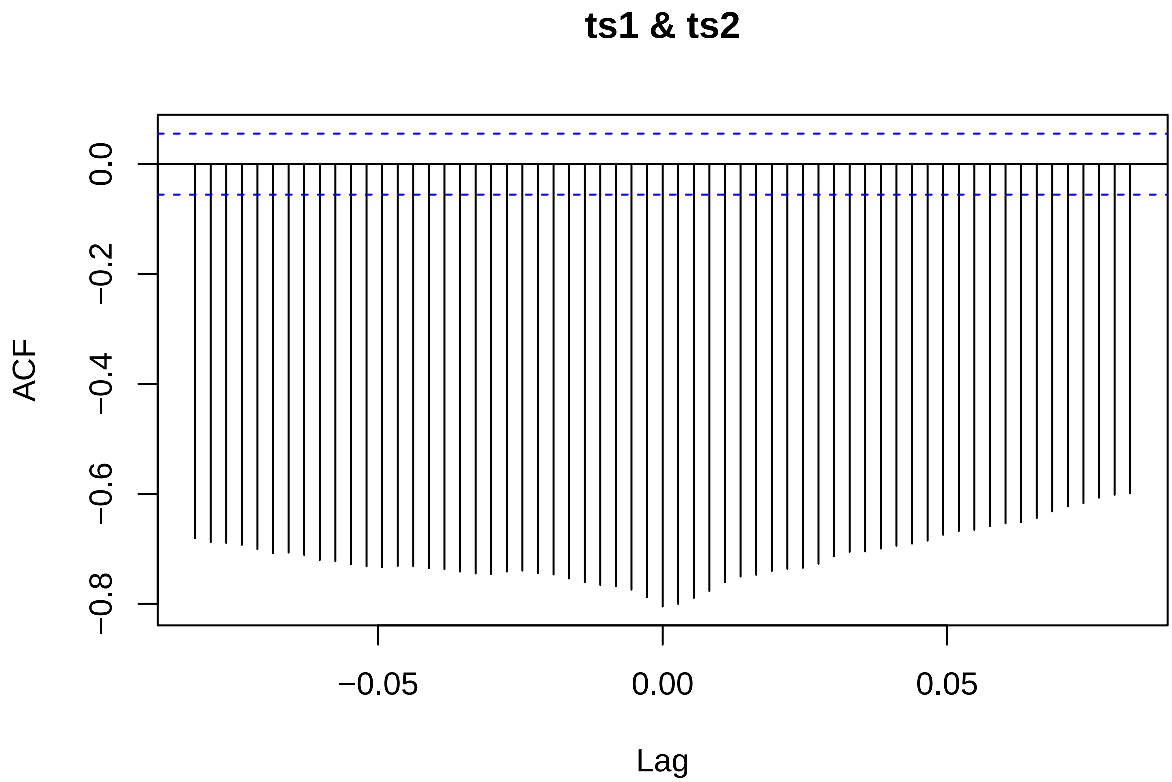


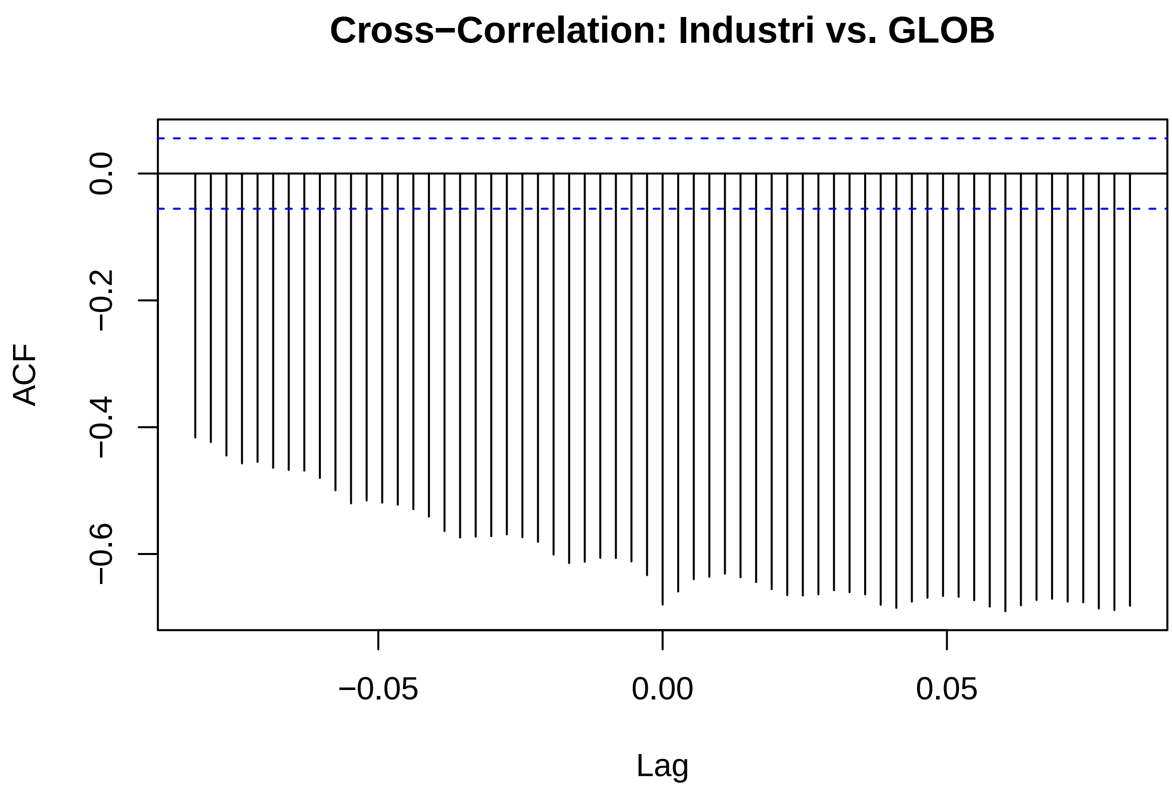
Cross−Correlation: Industri vs. Privat
ACF
0.0
0.2
0.4
0.6
0.8
−0.05
0.00
0.05
Lag



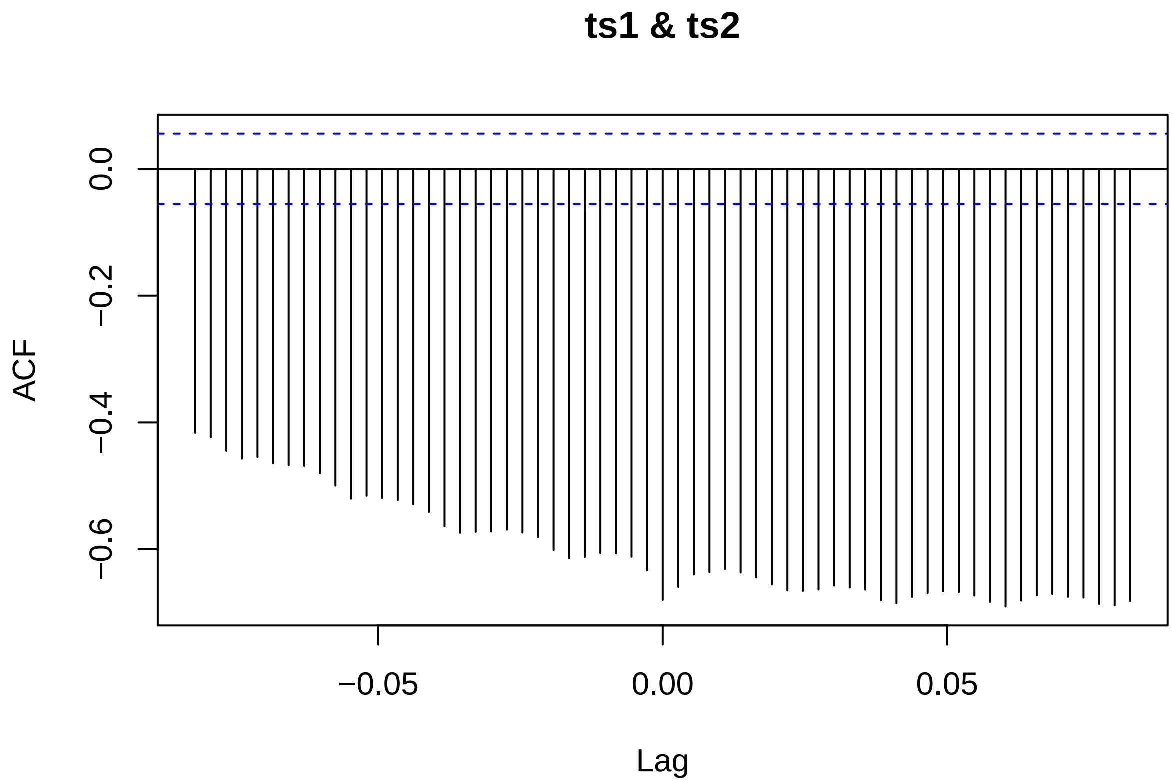
ts1 & ts2
ACF
0.0
0.2
0.4
0.6
0.8
−0.05
0.00
0.05
Lag

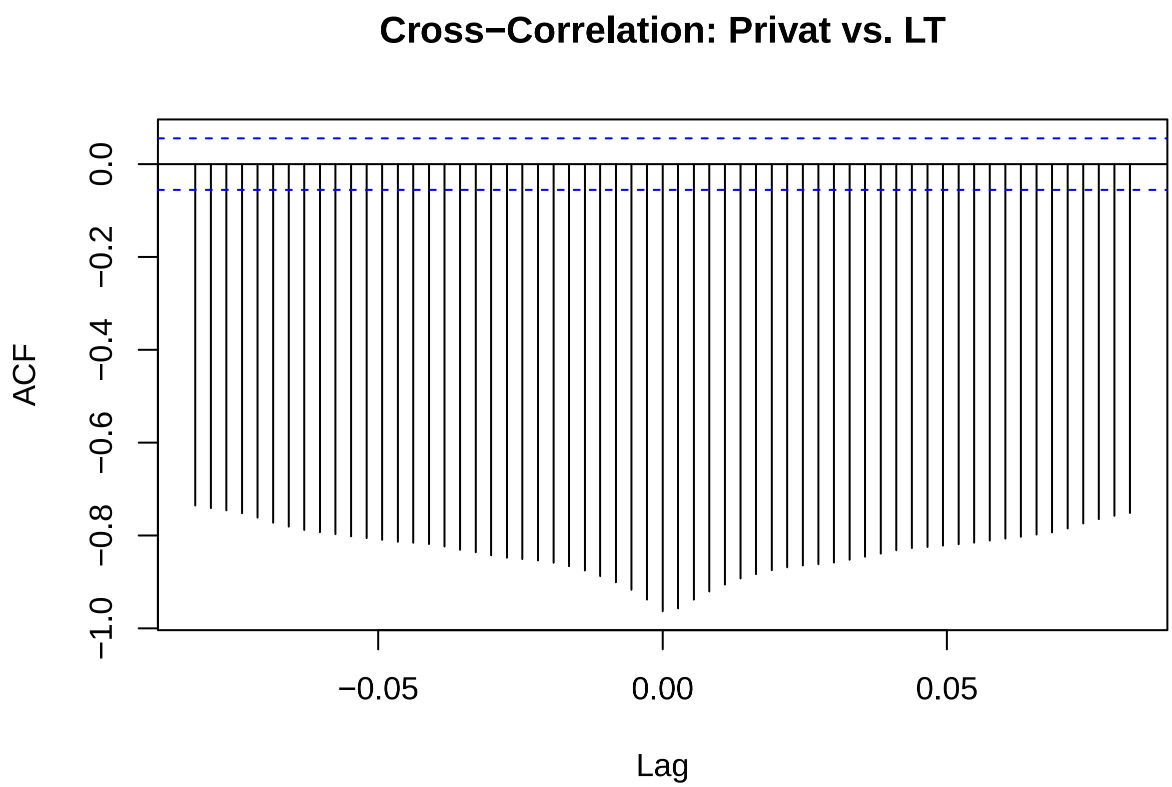

Cross−Correlation: Industri vs. LT
ACF
0.0
−0.2
−0.4
−0.6
−0.8
−0.05
0.00
0.05
Lag

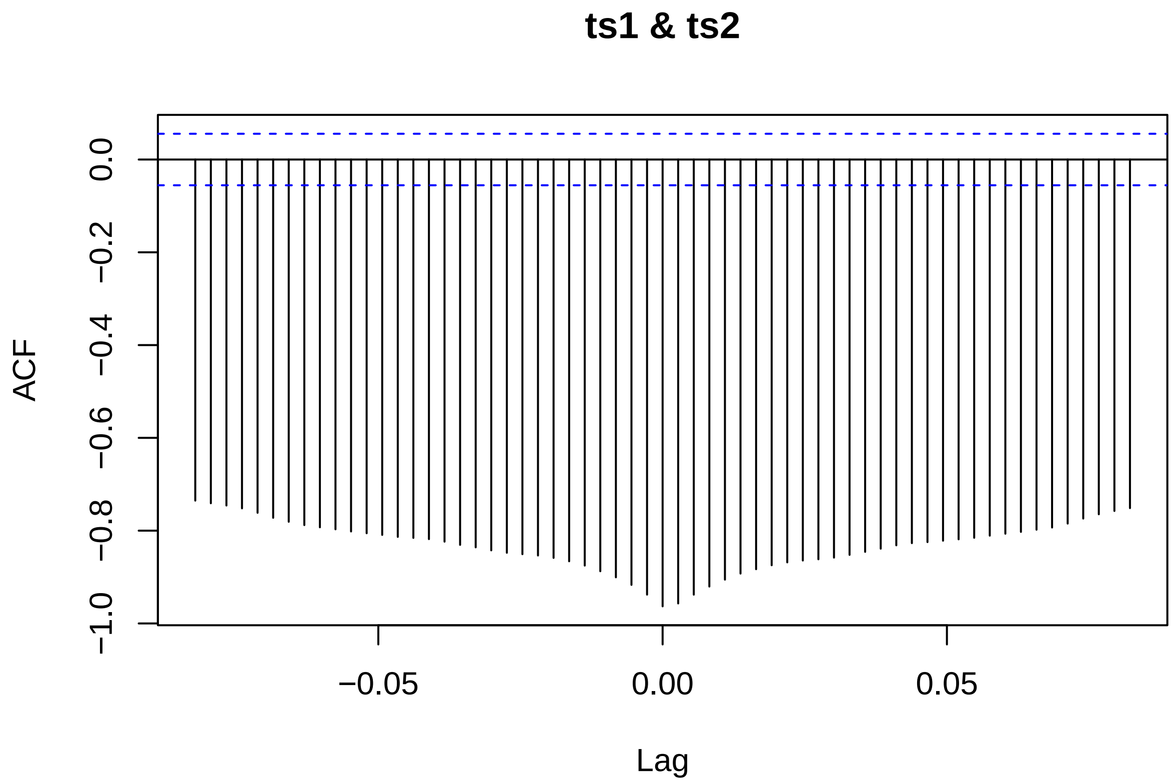

ts1 & ts2
ACF
0.0
−0.2
−0.4
−0.6
−0.8
−0.05
0.00
0.05
Lag

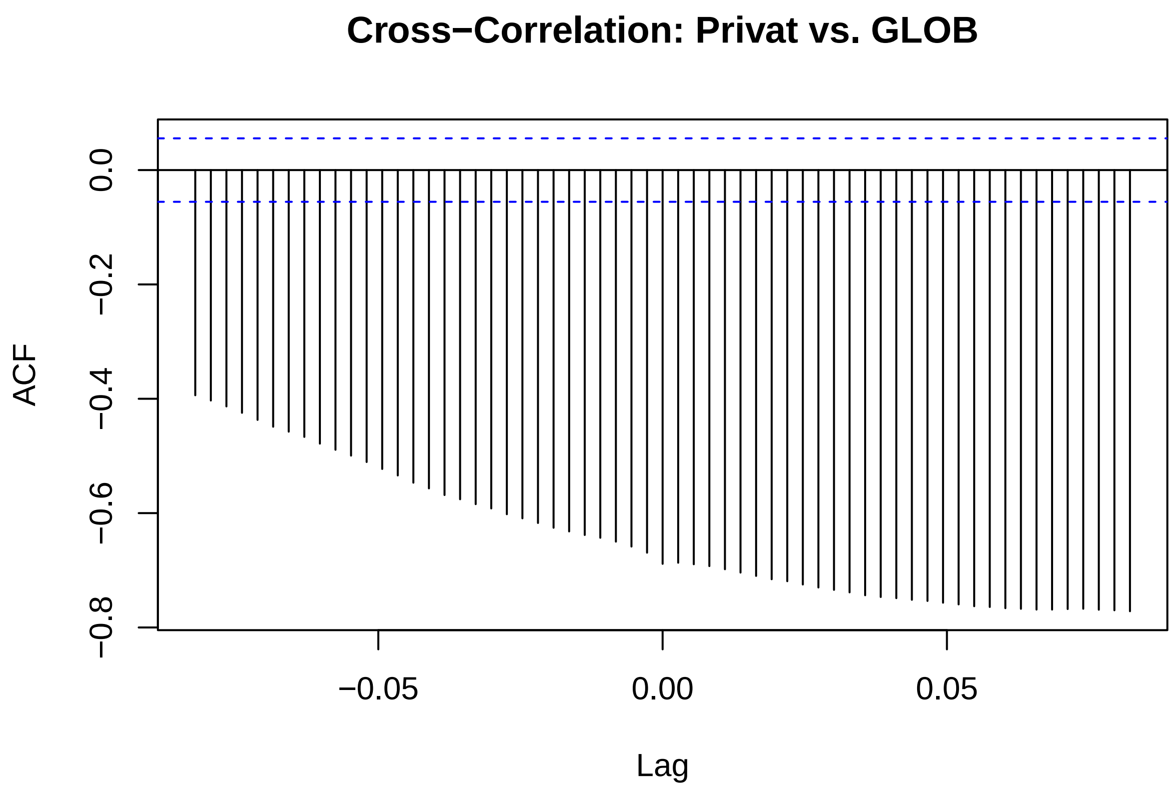


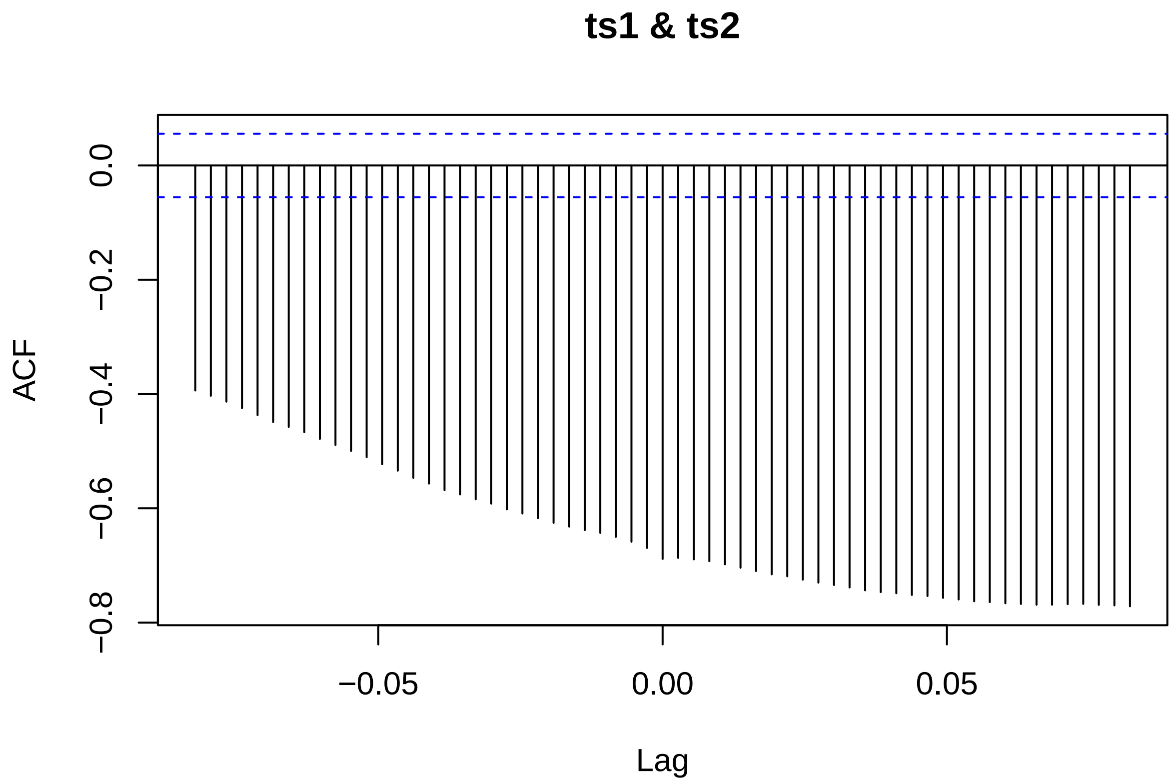
Cross−Correlation: Industri vs. GLOB
ACF
0.0
−0.2
−0.4
−0.6
−0.05
0.00
0.05
Lag

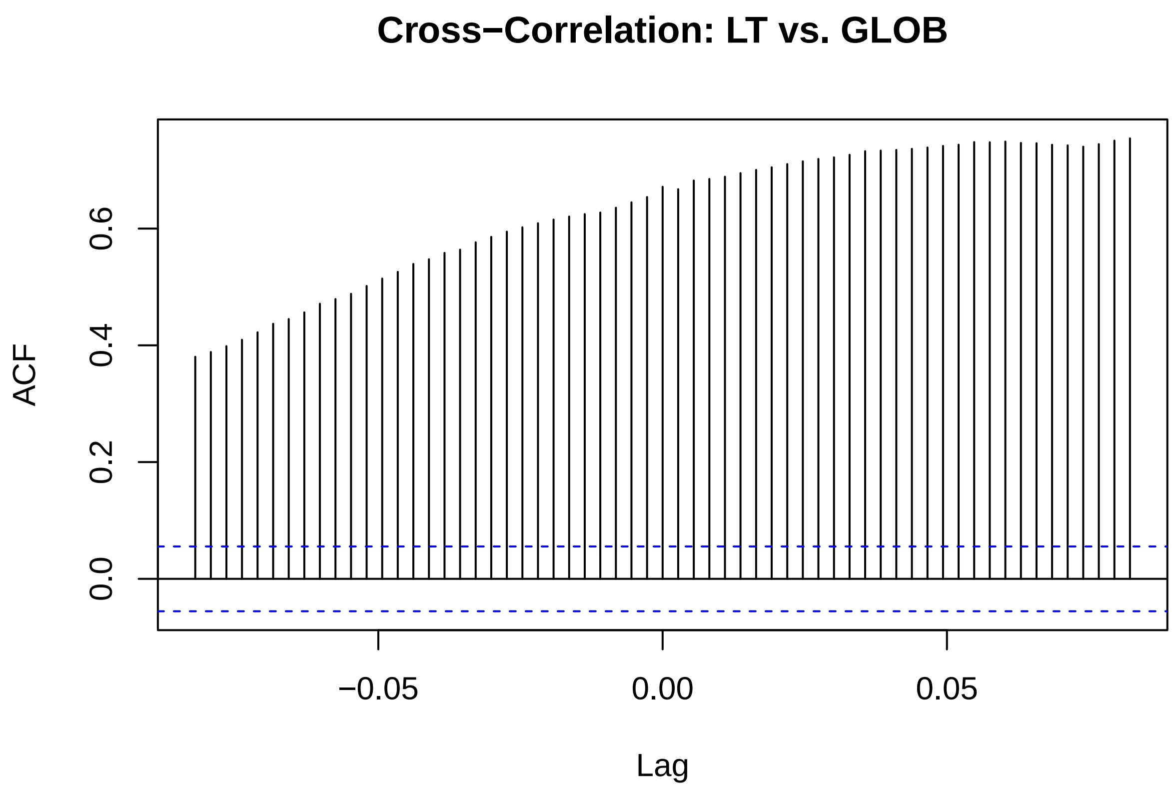

ts1 & ts2
ACF
0.0
−0.2
−0.4
−0.6
−0.05
0.00
0.05
Lag

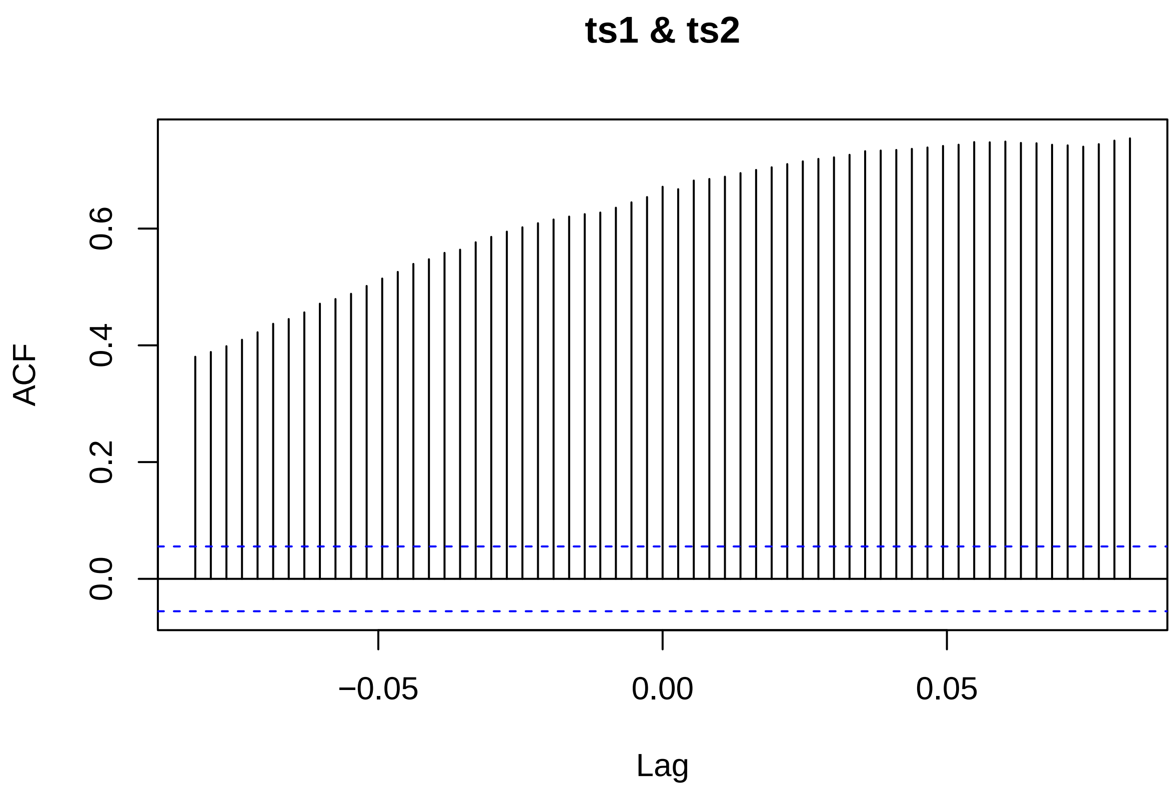

Cross−Correlation: Privat vs. LT
ACF
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−0.05
0.00
0.05
Lag


ts1 & ts2
ACF
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−0.05
0.00
0.05
Lag

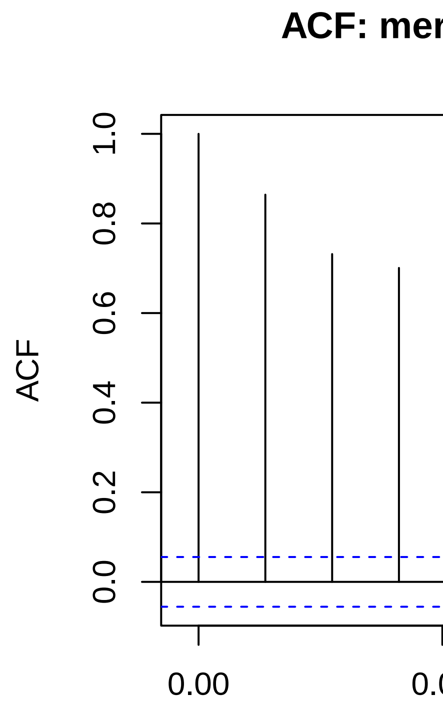

Cross−Correlation: Privat vs. GLOB
ACF
0.0
−0.2
−0.4
−0.6
−0.8
−0.05
0.00
0.05
Lag


ts1 & ts2
ACF
0.0
−0.2
−0.4
−0.6
−0.8
−0.05
0.00
0.05
Lag

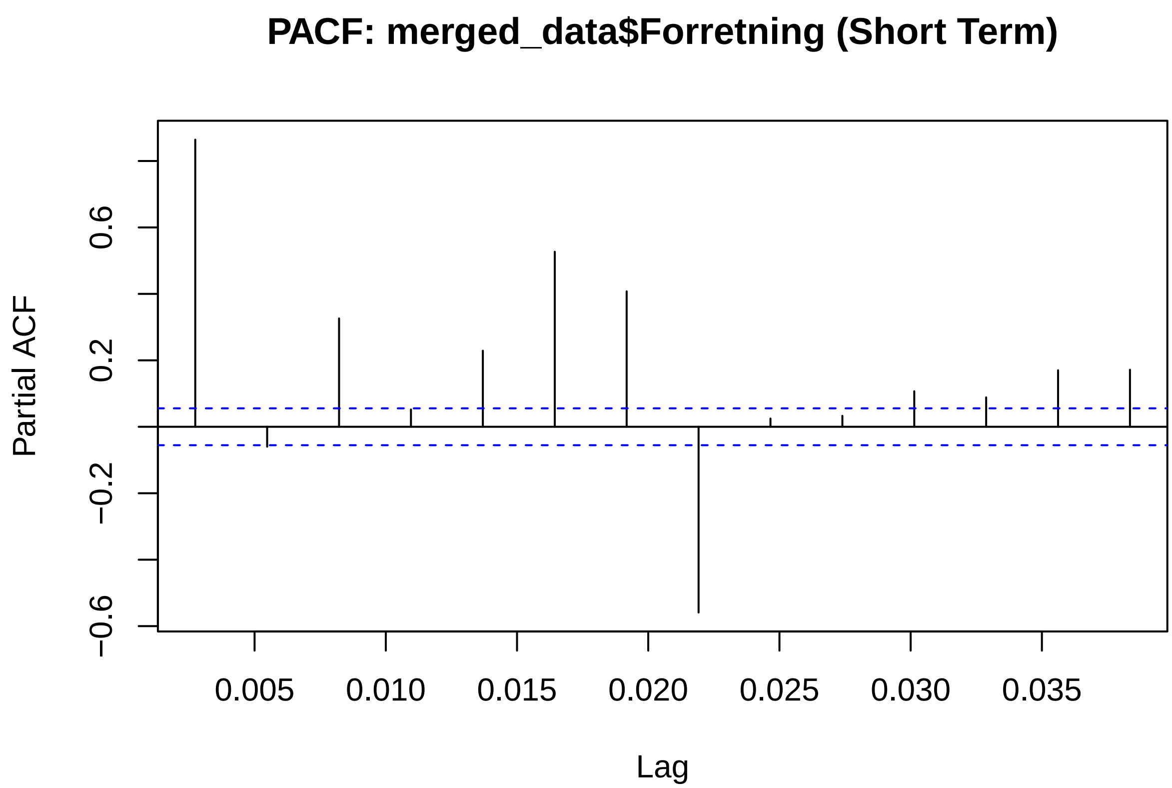

Cross−Correlation: LT vs. GLOB
ACF
0.0
0.2
0.4
0.6
−0.05
0.00
0.05
Lag

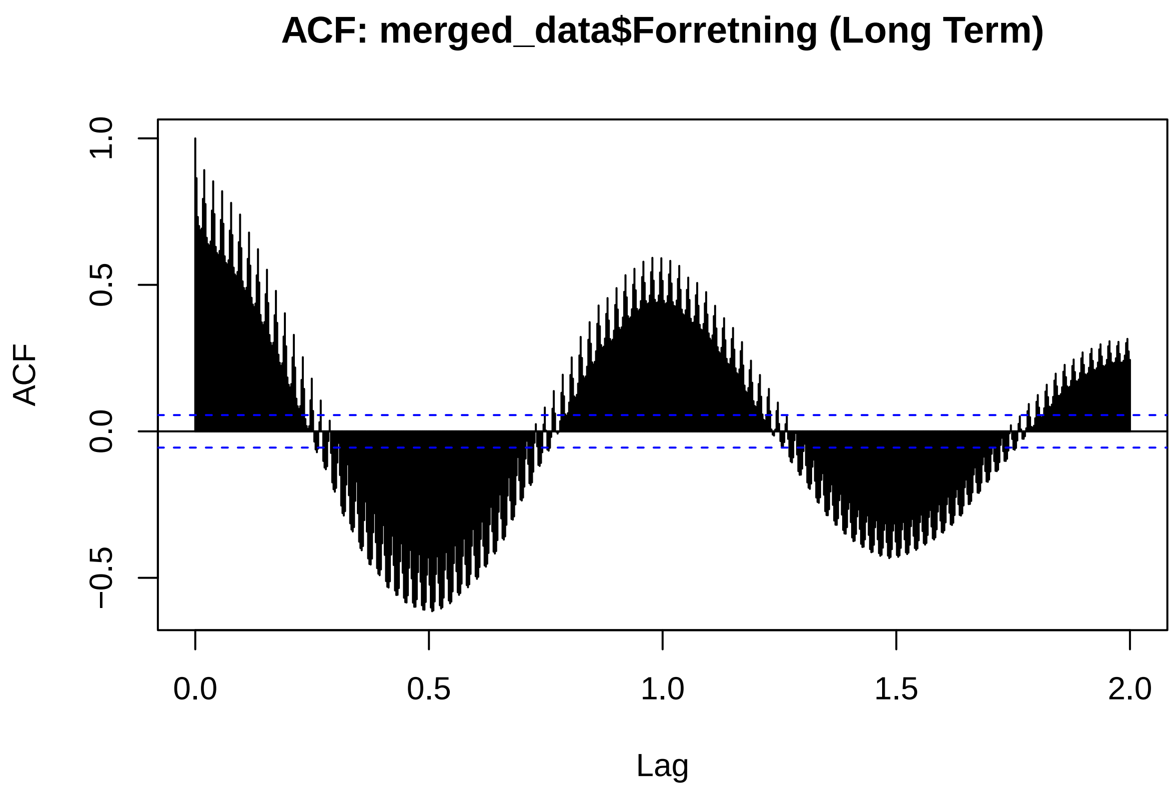


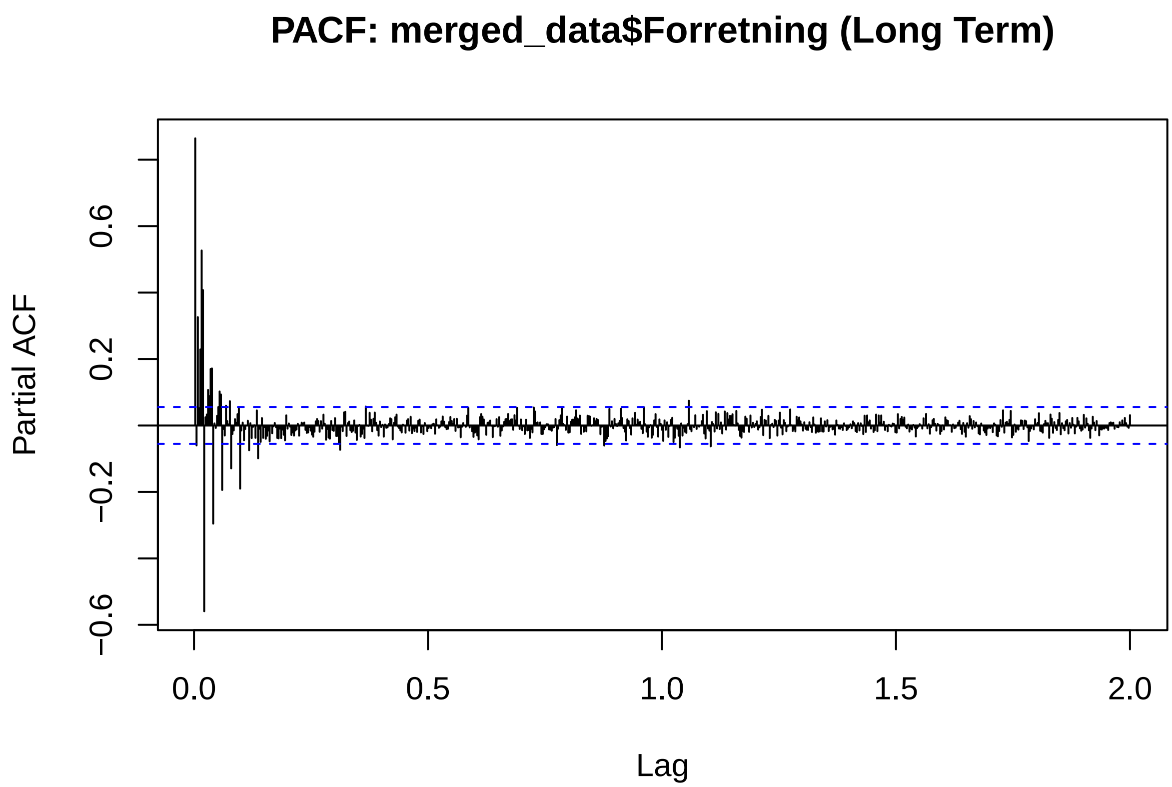

```
acf_pacf_plot <- function(data) {
  ts_data <- ts(na.remove(data), frequency = 365)

  acf(ts_data, lag.max = 14, main = paste("ACF:", deparse(substitute(data)) ,"(Short Term)"))

  pacf(ts_data, lag.max = 14, main = paste("PACF:", deparse(substitute(data)) ,"(Short Term)"))

  acf(ts_data, lag.max = 365*2, main = paste("ACF:", deparse(substitute(data)) ,"(Long Term)"))

  pacf(ts_data, lag.max = 365*2, main = paste("PACF:", deparse(substitute(data)) ,"(Long Term)"))
}

acf_pacf_plot(merged_data$Forretning)
```



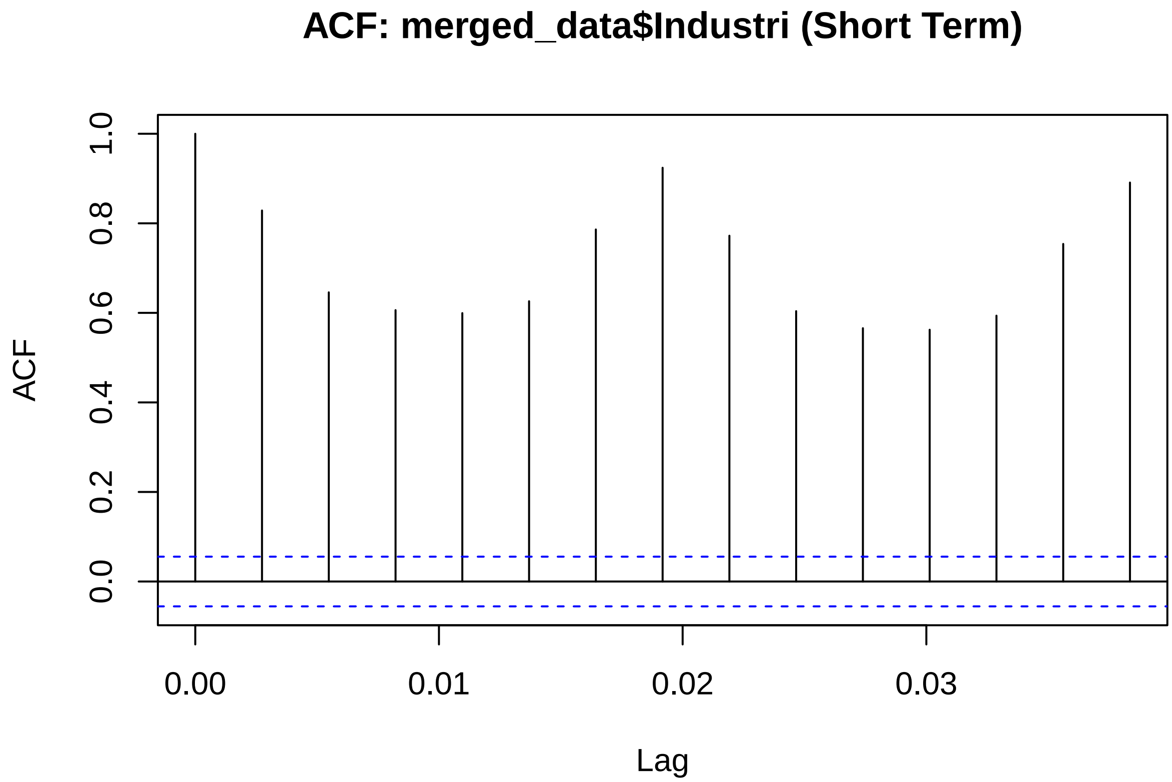
4. **Autocorrelation and Partial Autocorrelation Functions (ACF and PACF)**

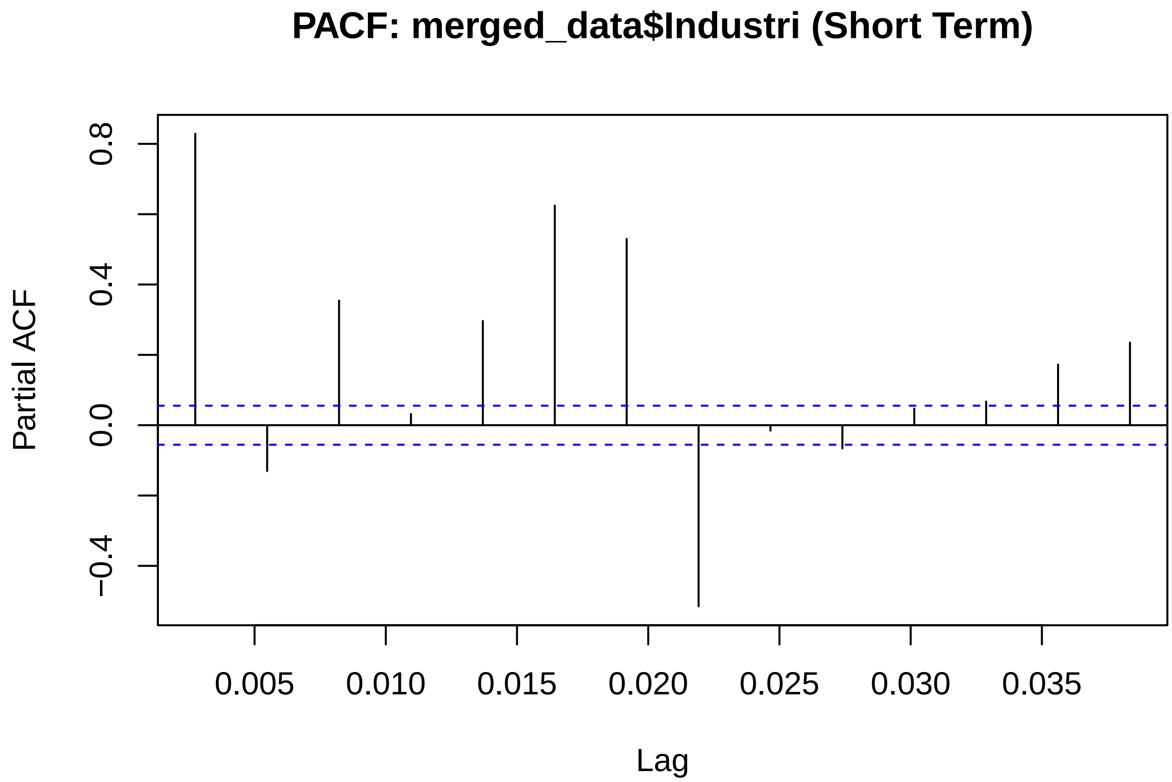


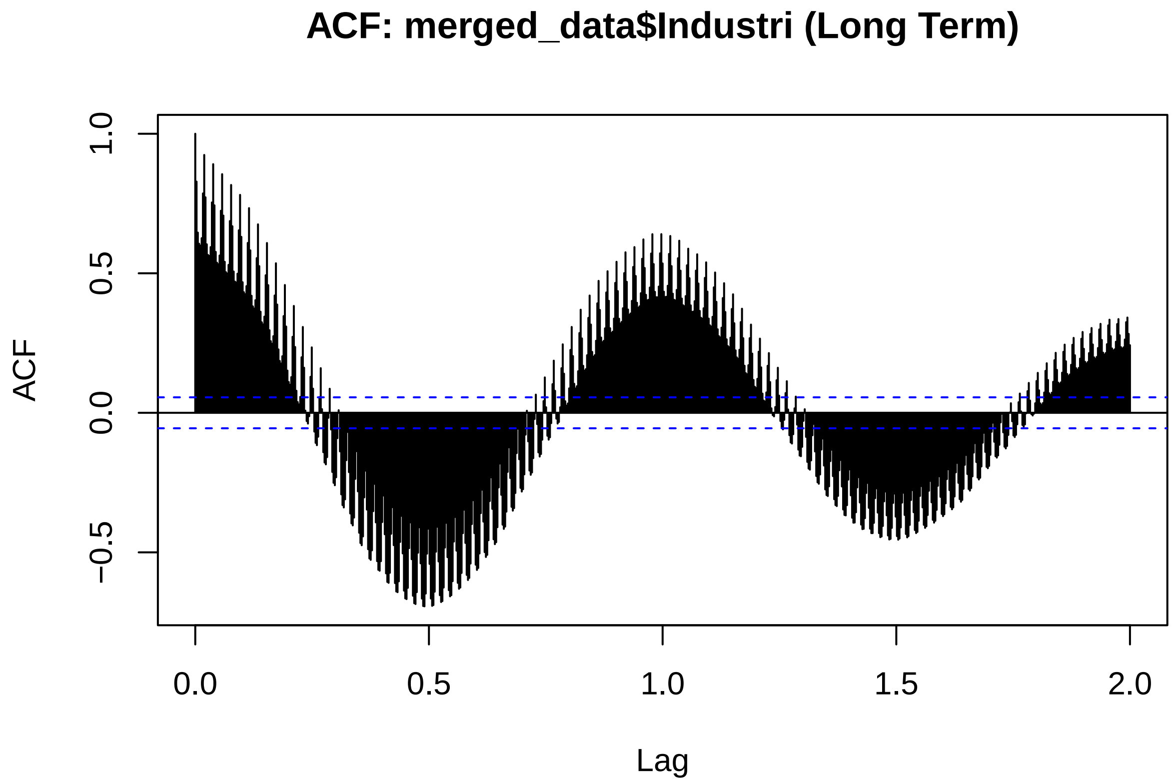

ACF: merged_data$Forretning (Long Term)
ACF
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
2.0
Lag

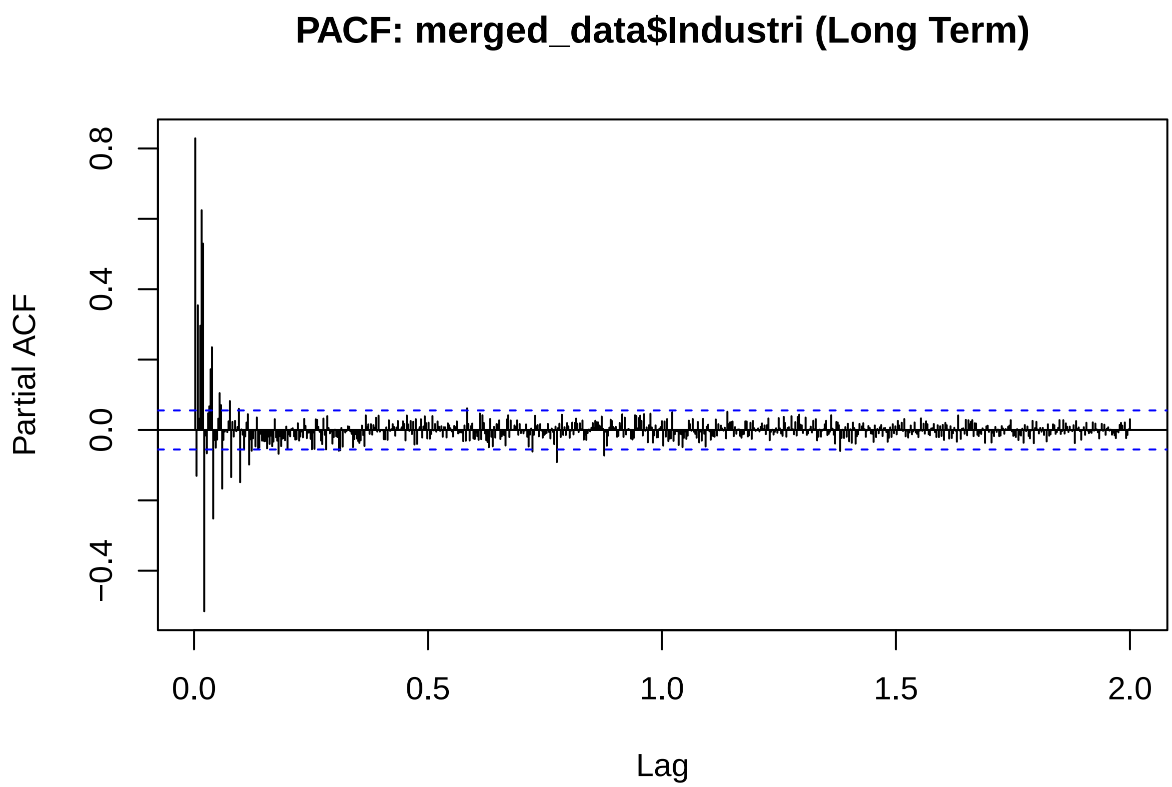



```r
acf_pacf_plot(merged_data$Industri)
```

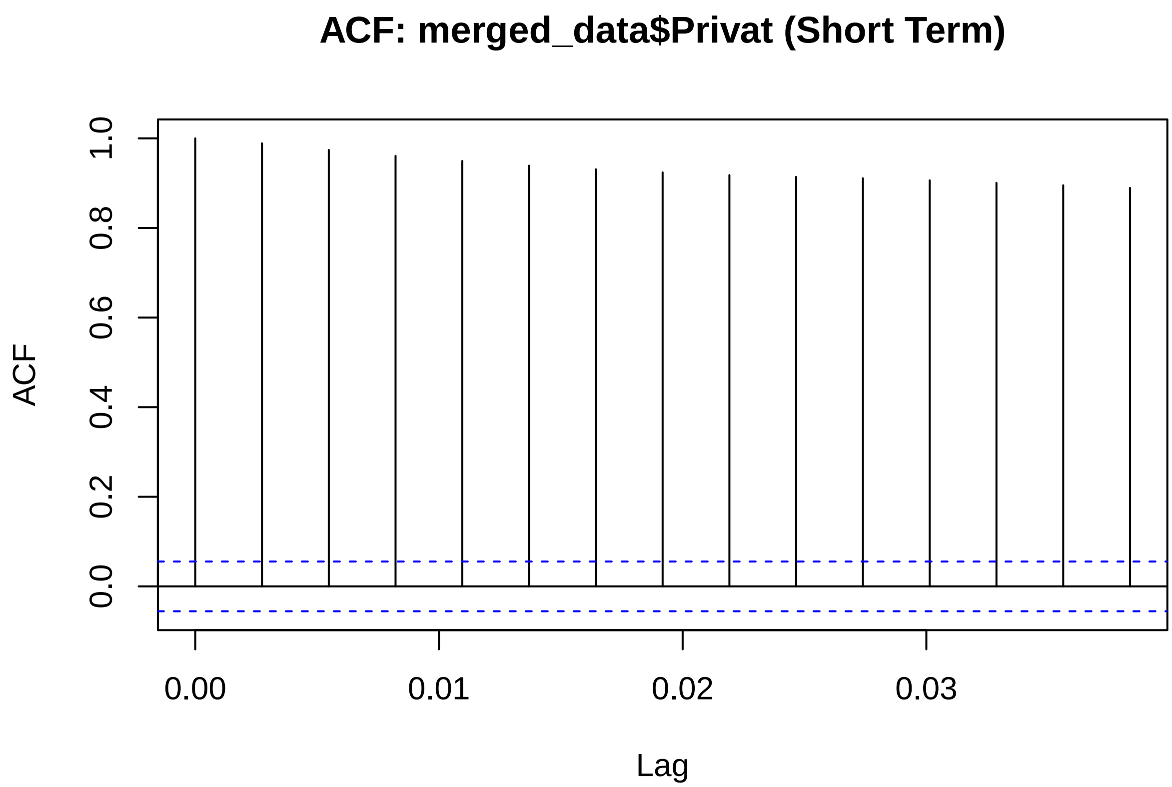

ACF: merged_data$Industri (Short Term)
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.01
0.02
0.03
Lag

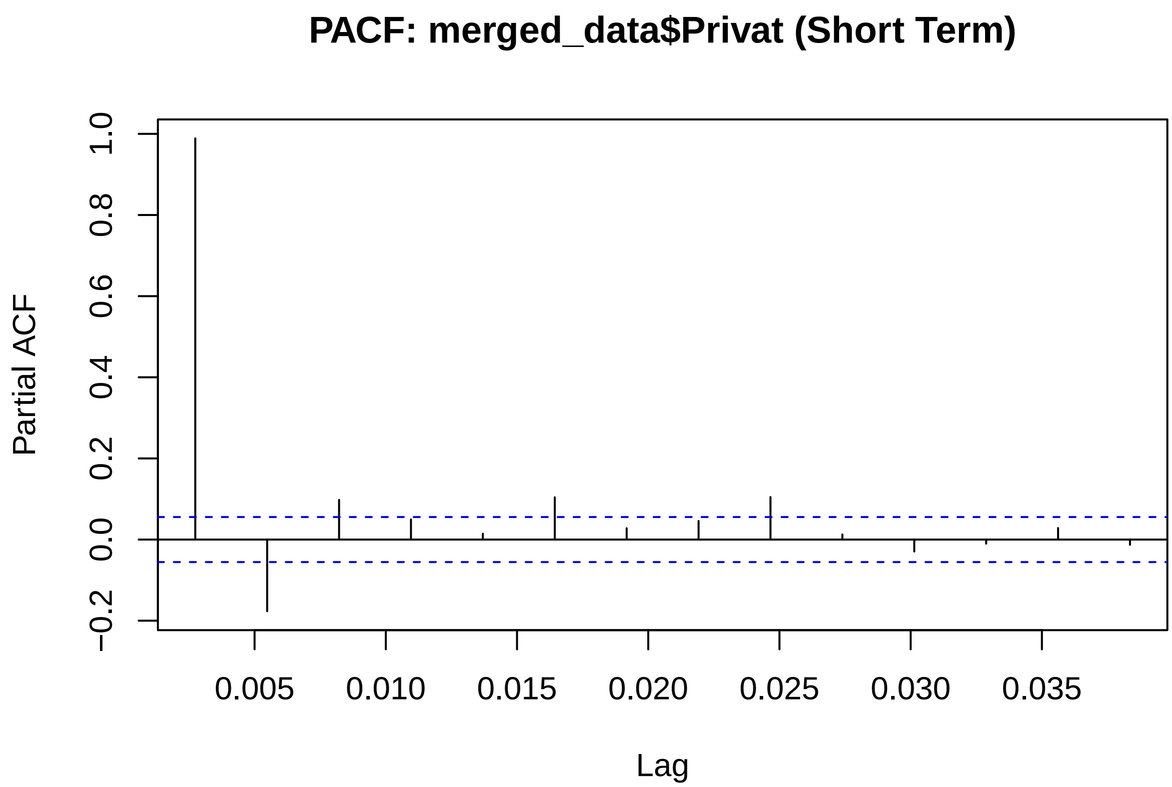

PACF: merged_data$Industri (Short Term)
Partial ACF
0.8
0.4
0.0
−0.4
0.005
0.010
0.015
0.020
0.025
0.030
0.035
Lag

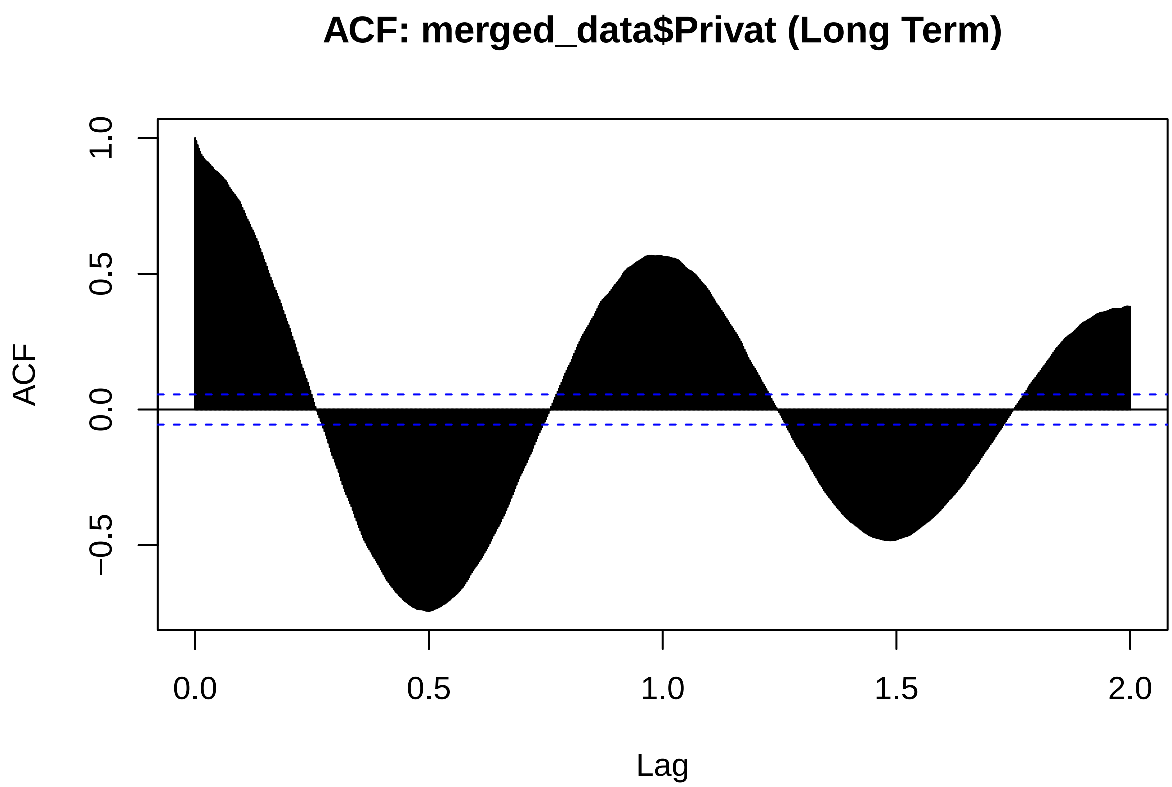

ACF: merged_data$Industri (Long Term)
ACF
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
2.0
Lag

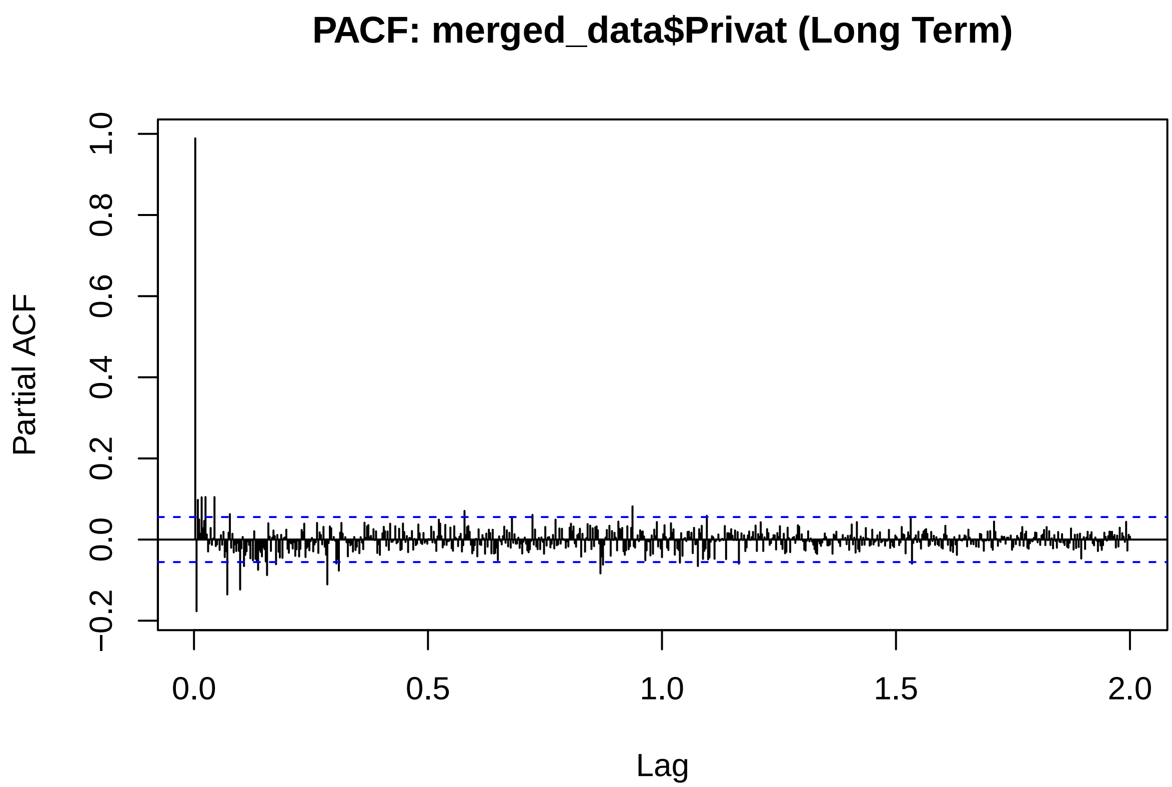



```
acf_pacf_plot(merged_data$Privat)
```

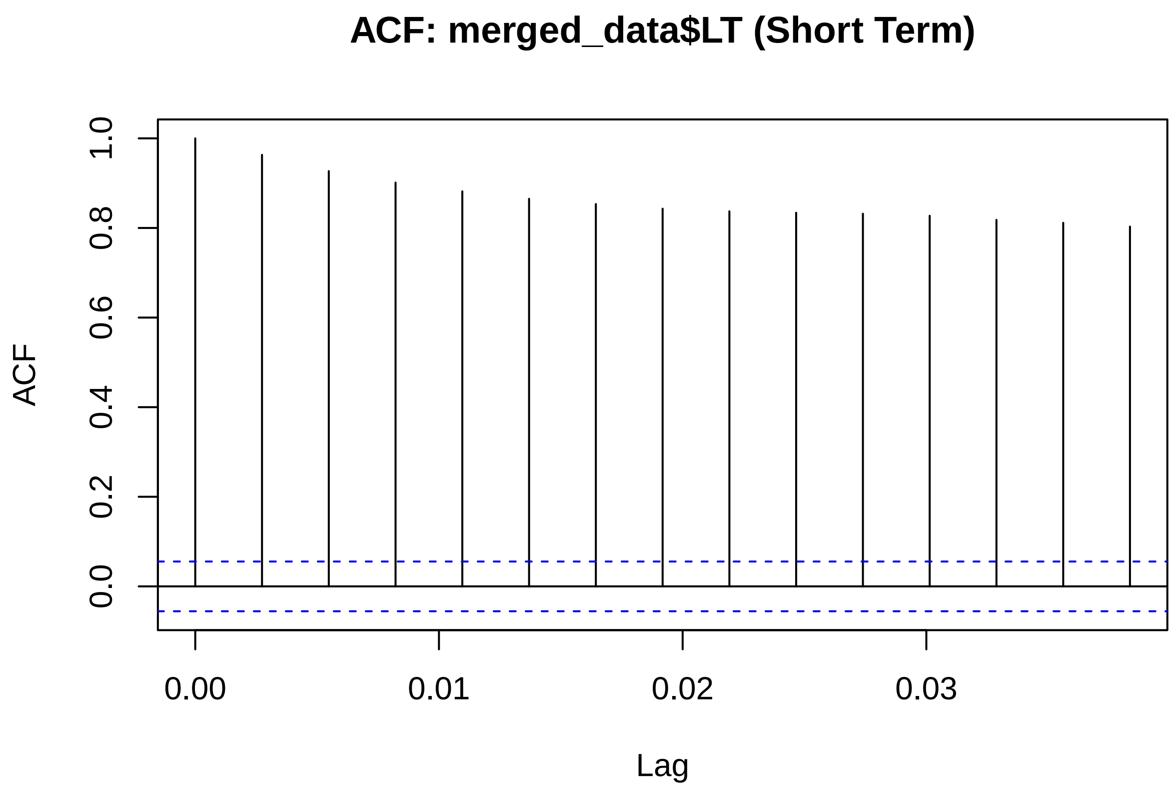

ACF: merged_data$Privat (Short Term)
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.01
0.02
0.03
Lag

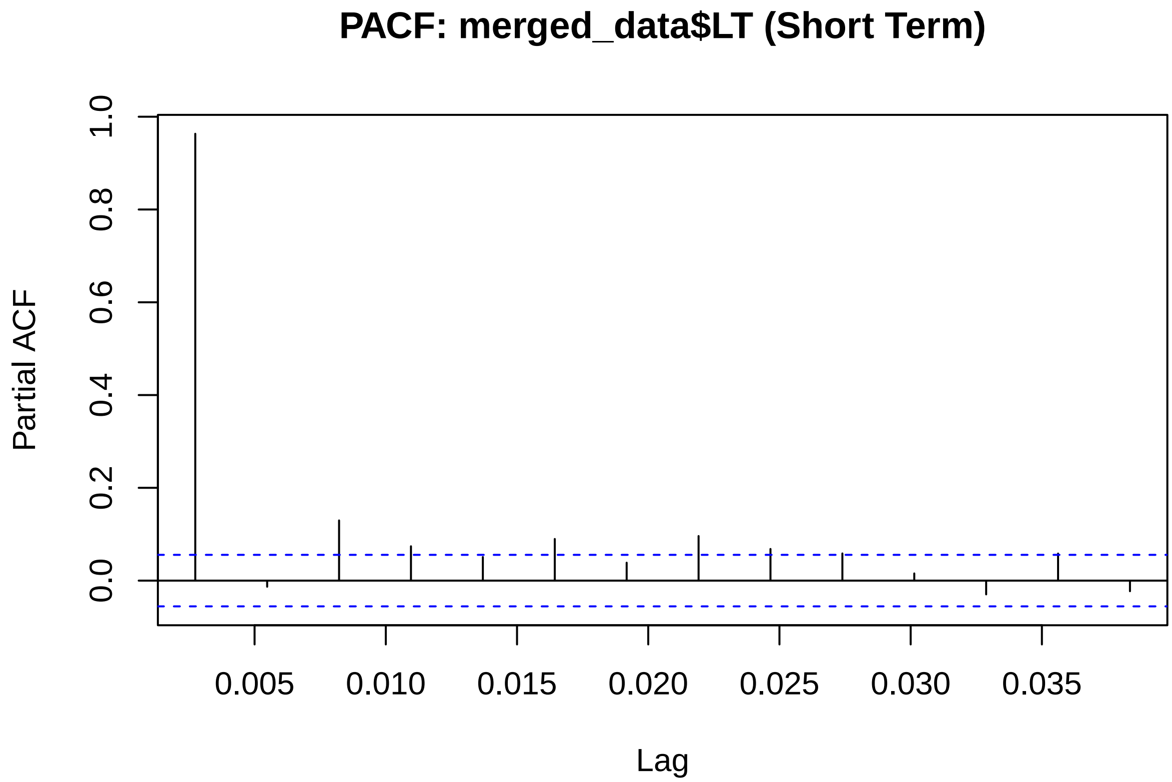

PACF: merged_data$Privat (Short Term)
Partial ACF
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
0.005
0.010
0.015
0.020
0.025
0.030
0.035
Lag

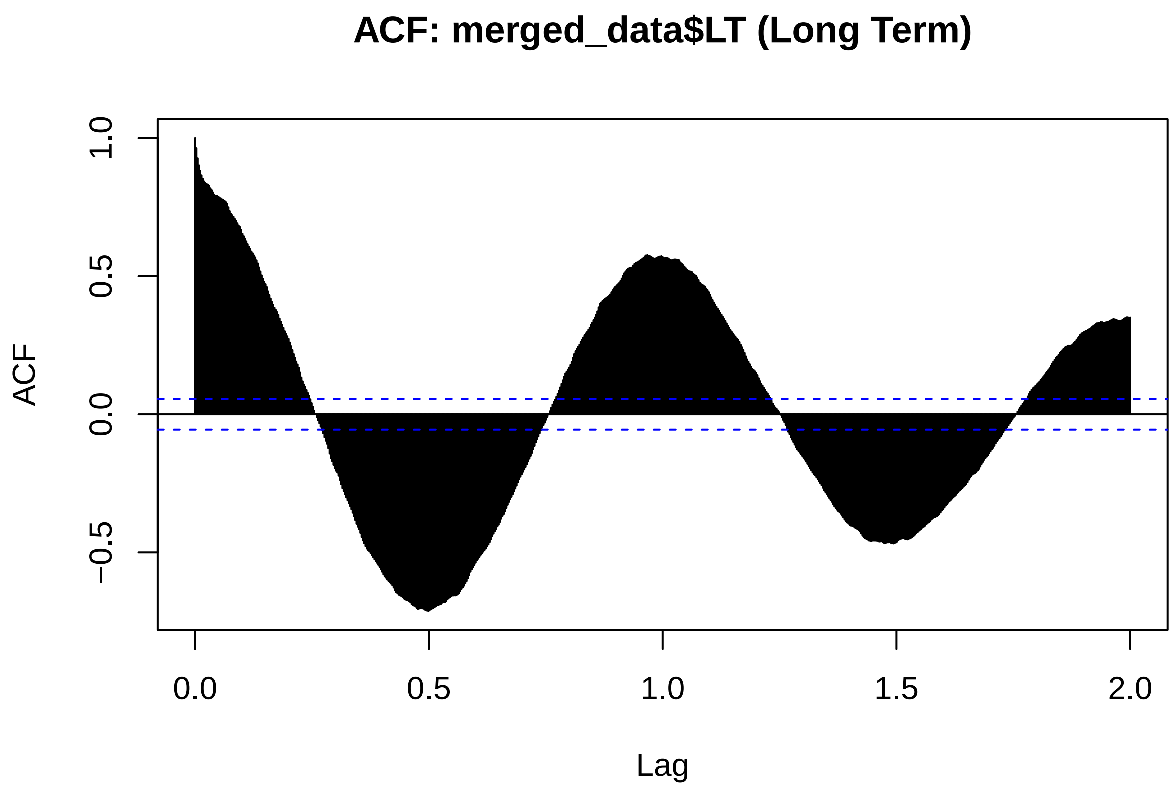

ACF: merged_data$Privat (Long Term)
ACF
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
2.0
Lag



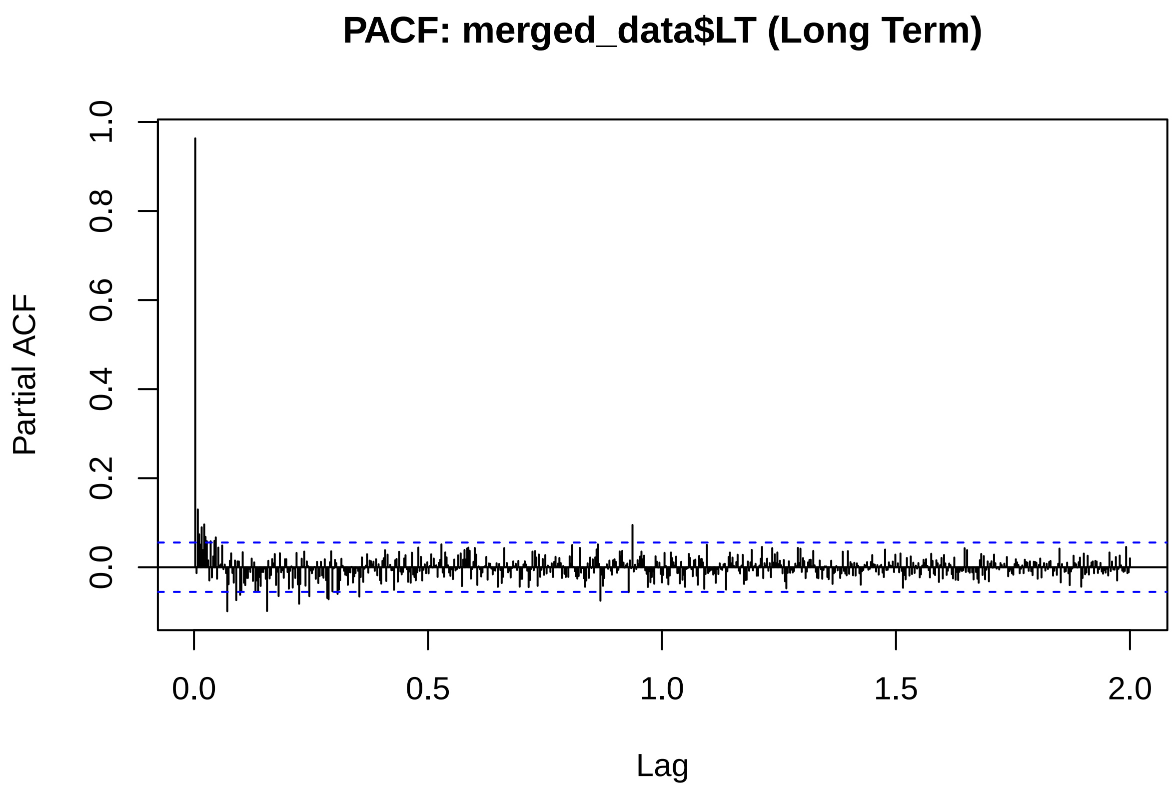

```r
acf_pacf_plot(merged_data$LT)
```


ACF: merged_data$LT (Short Term)
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.01
0.02
0.03
Lag

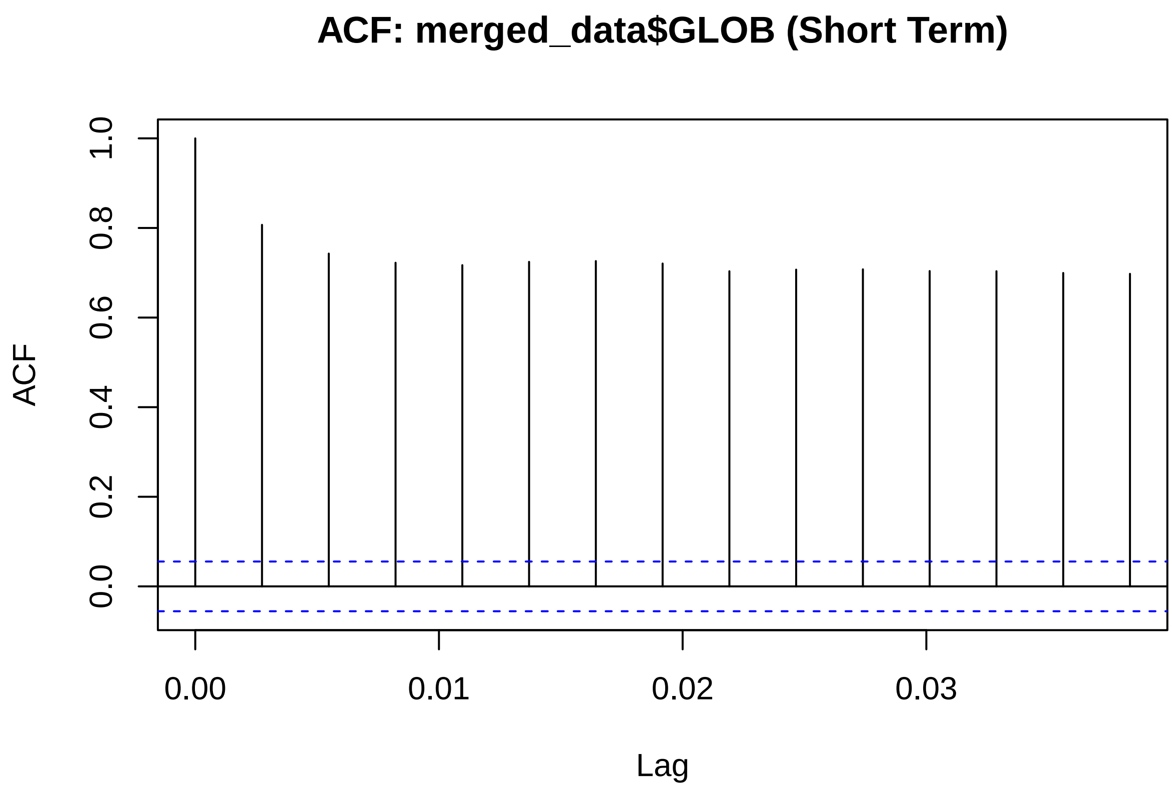

PACF: merged_data$LT (Short Term)
Partial ACF
0.0
0.2
0.4
0.6
0.8
1.0
0.005
0.010
0.015
0.020
0.025
0.030
0.035
Lag



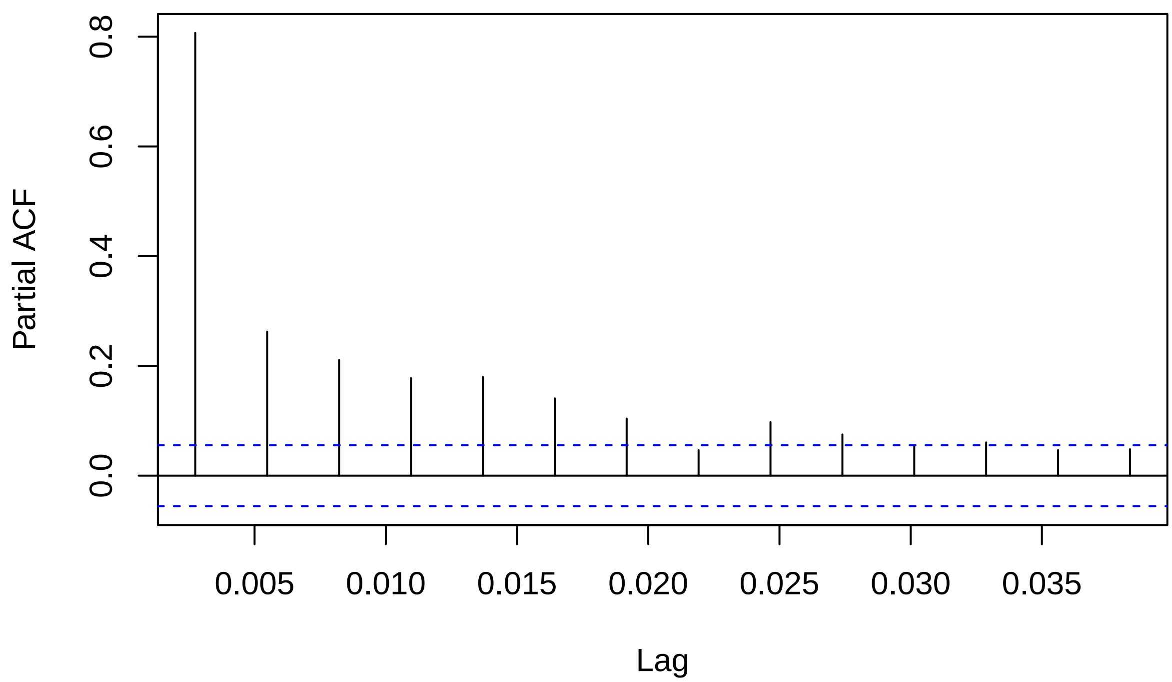
ACF: merged_data$LT (Long Term)
ACF
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
2.0
Lag

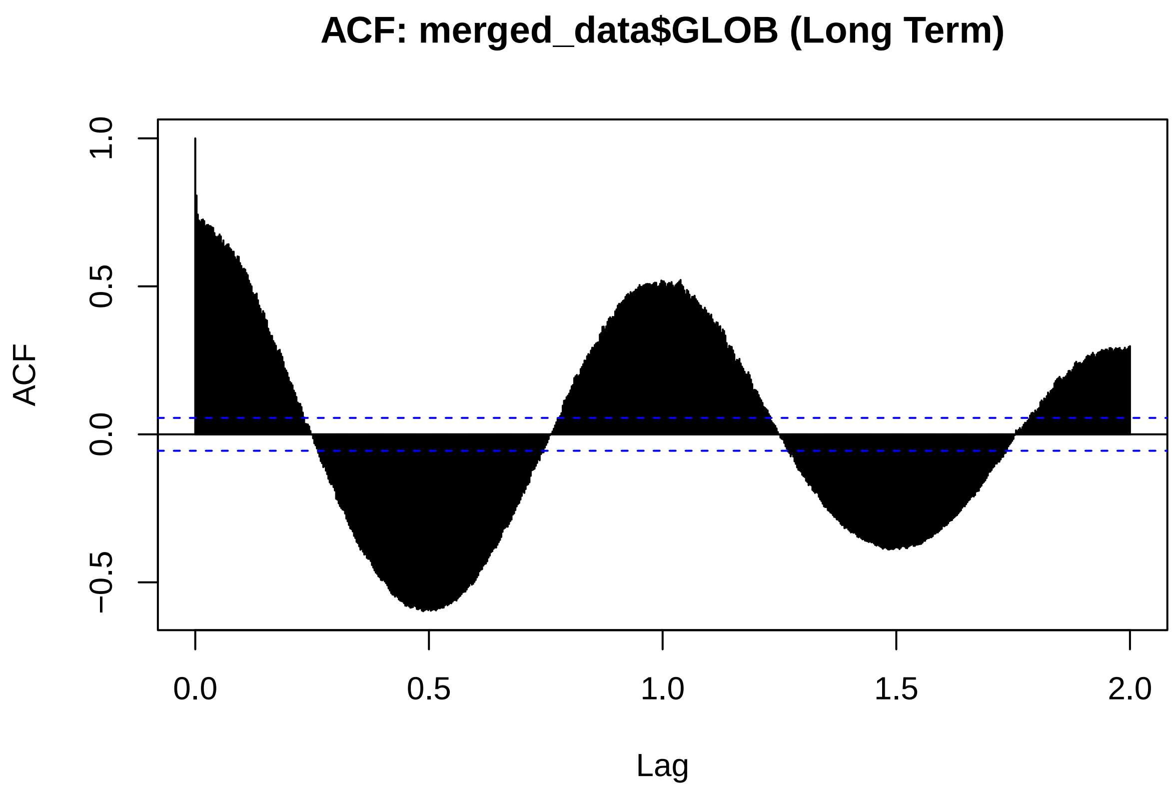


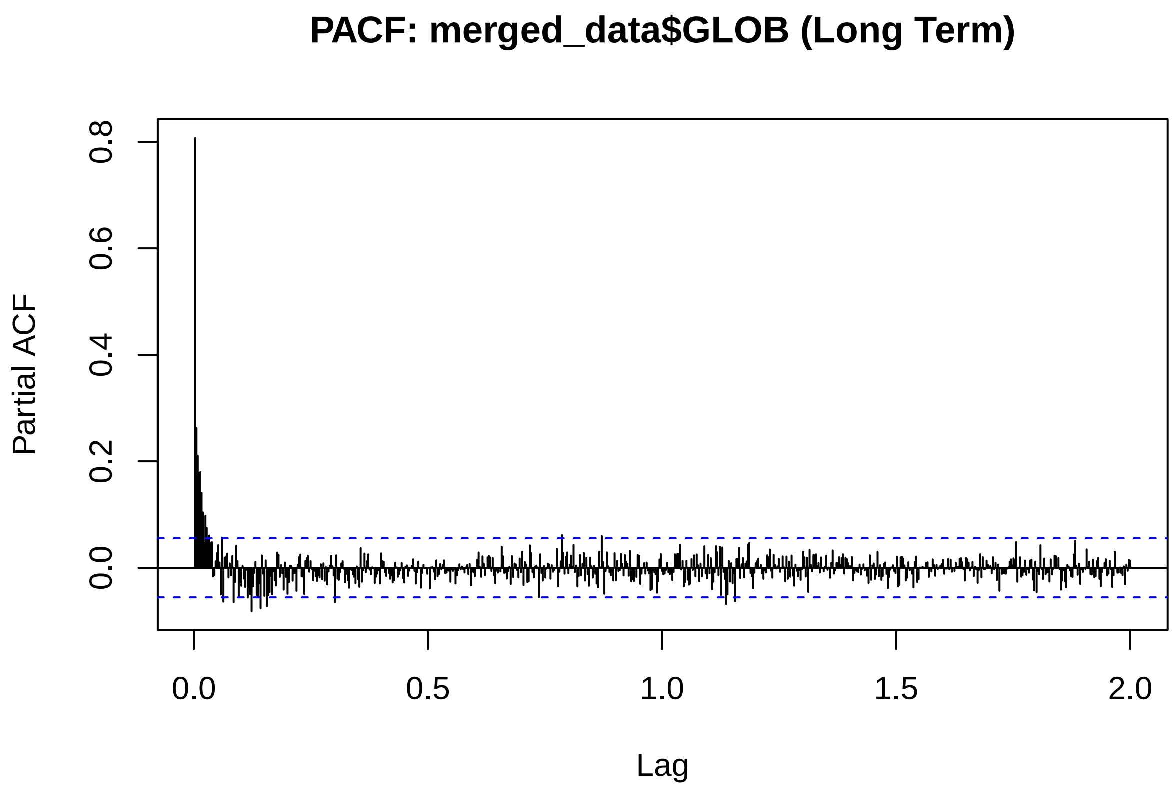

```
acf_pacf_plot(merged_data$GLOB)
```


ACF: merged_data$GLOB (Short Term)
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.01
0.02
0.03
Lag

PACF: merged_data$GLOB (Short Term)

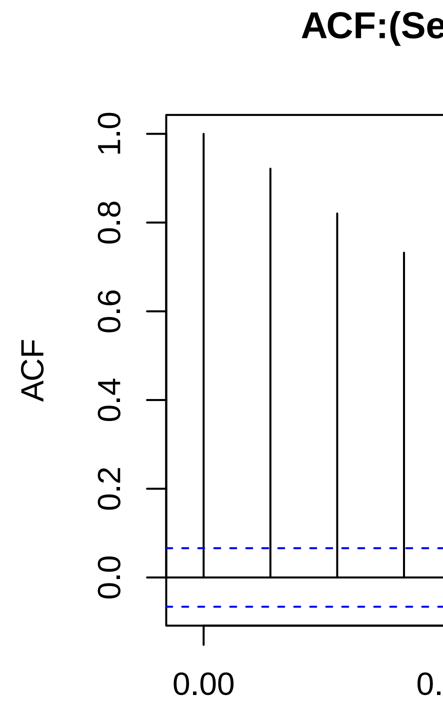



ACF: merged_data$GLOB (Long Term)
ACF
Lag
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
2.0



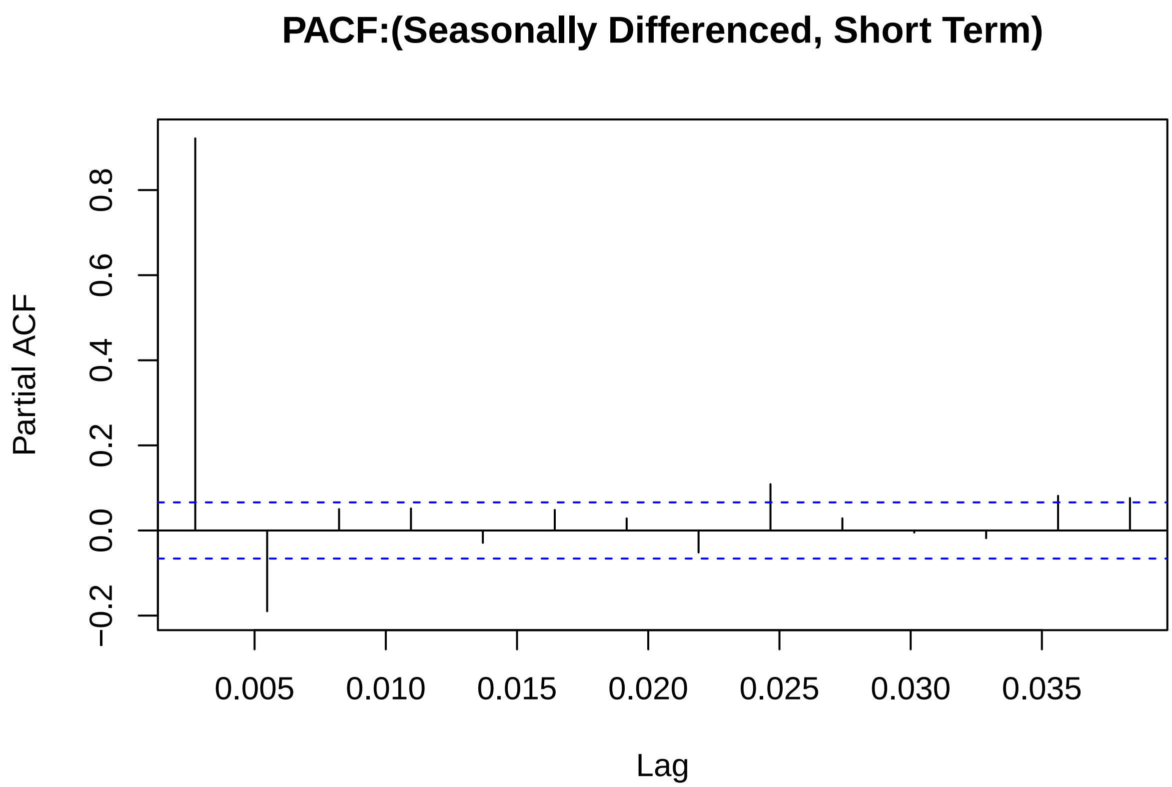
Task E: Seasonal Differencing and ACF/PACF

```
ts_data <- ts(na.remove(merged_data$Privat), frequency = 365)
diff_consumption <- diff(ts_data, lag = 365)

acf(diff_consumption, lag.max = 14, main = paste("ACF:(Seasonally Differenced, Short Term)"))
```



Seasonal Differencing of Consumption, Temperature, and Global Irradiation

```
pacf(diff_consumption, lag.max = 14, main = paste("PACF:(Seasonally Differenced, Short Term)"))
```



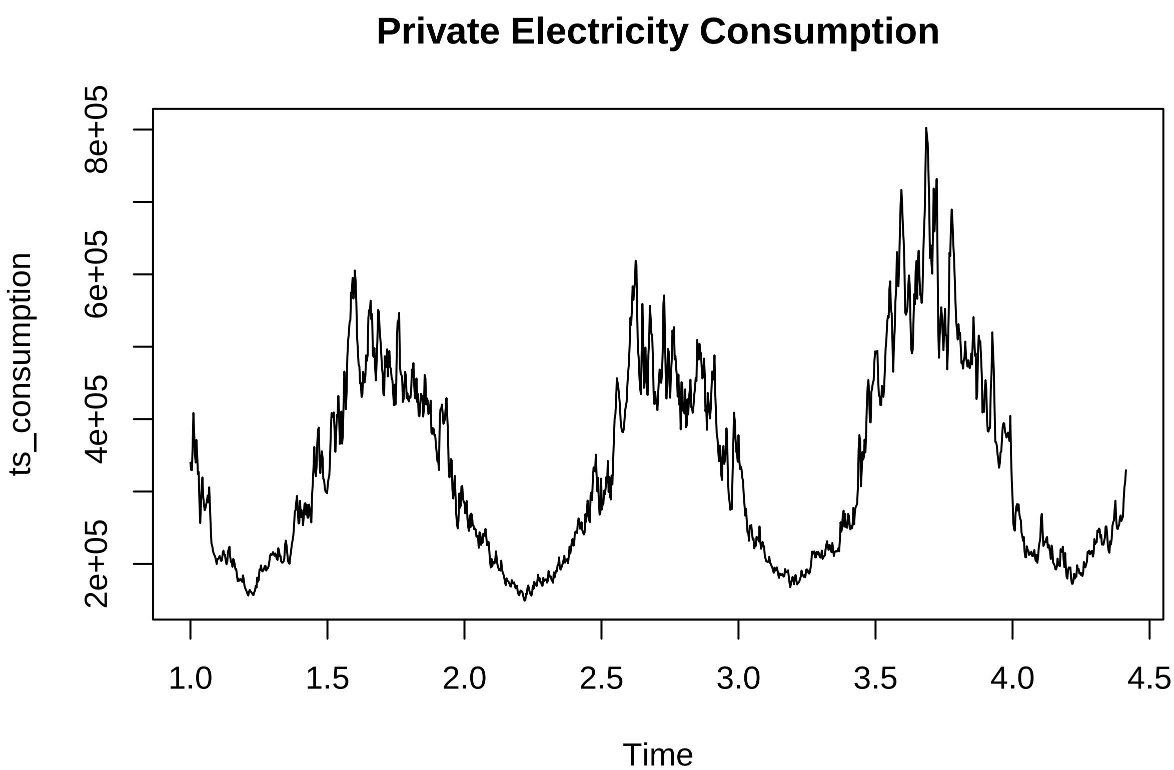

```
# Repeat for Temperature and Global Irradiation...
```

Task F: STL Decomposition Analysis

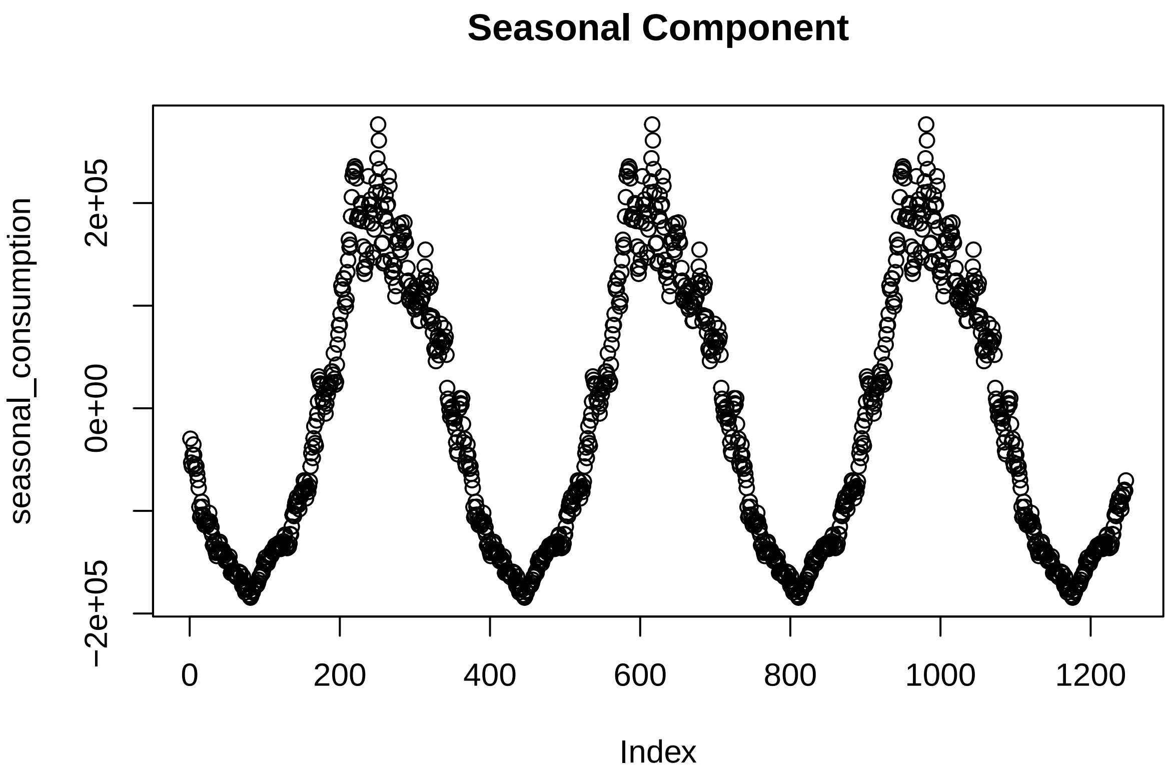
```
# Choose STL parameters
stl_parameters <- list(s.window = "periodic",  # Seasonal window for yearly data
t.window = 365,          # Trend window (yearly smoothing)
l.window = 30,           # Low-frequency window (adjust as needed)
robust = TRUE)          # Use robust STL (handles outliers better)
ma_order <- 7 # Set the order for the moving average smoother (e.g., 7 for weekly smoothing).  You can
```

```
# --- Private Electricity Consumption ---
consumption_data <- merged_data$Privat
ts_consumption <- ts(na.remove(consumption_data), frequency = 365)
# Perform STL decomposition
stl_consumption <- stlplus(ts_consumption, period = 365,
                           s.window = stl_parameters$s.window,
                           t.window = stl_parameters$t.window,
                           l.window = stl_parameters$l.window,
                           robust = stl_parameters$robust)
# Smooth the trend-cycle (AFTER STL) using a moving average
smoothed_trendcycle <- ma(stl_consumption$data[, "trend"], order = ma_order) # Using moving average.
# Extract components
seasonal_consumption <- stl_consumption$data[, "seasonal"] # Seasonal component (periodic fluctuations)
remainder_consumption <- stl_consumption$data[, "remainder"]  # Remainder component (noise or error)
trendcycle_consumption <- stl_consumption$data[, "trend"]
```

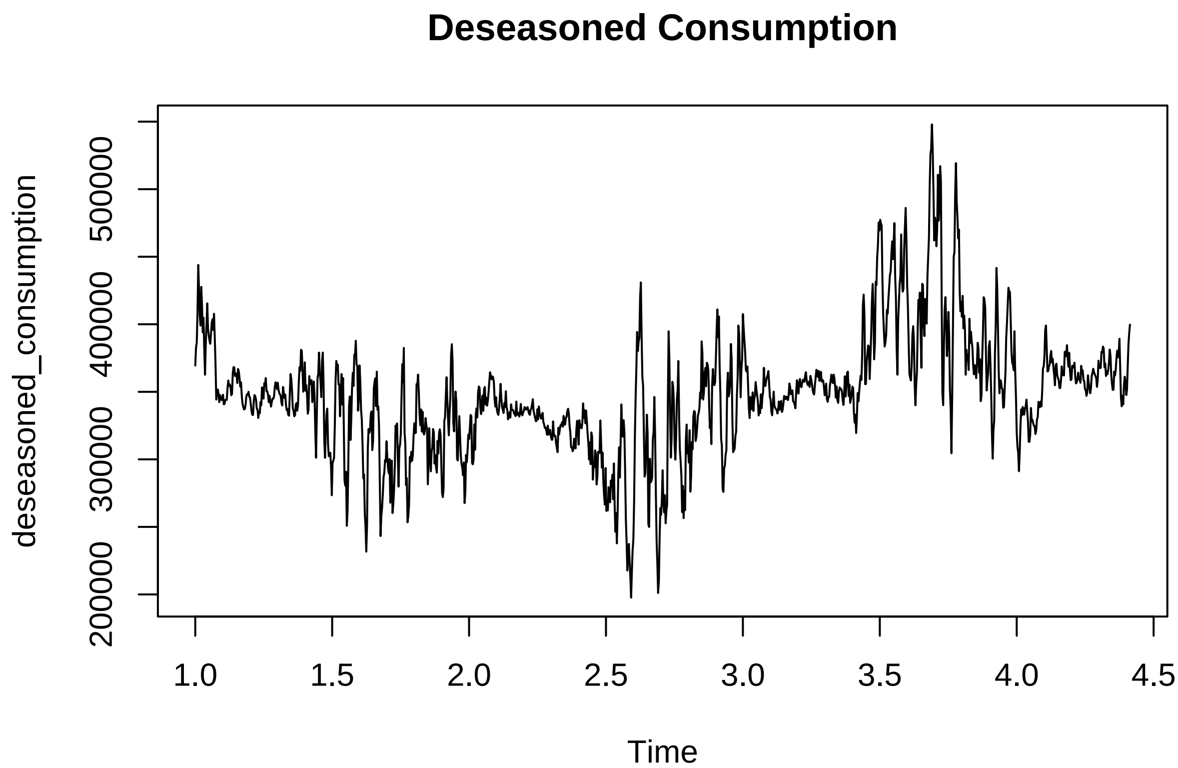
```r
if ("cycle" %in% colnames(stl_consumption$data)) {
  trendcycle_consumption <- trendcycle_consumption + stl_consumption$data[, "cycle"]
}
deseasoned_consumption <- ts_consumption - seasonal_consumption  # Deseasonalized data
# Plot components
plot(ts_consumption, main = "Private Electricity Consumption") # Original
```



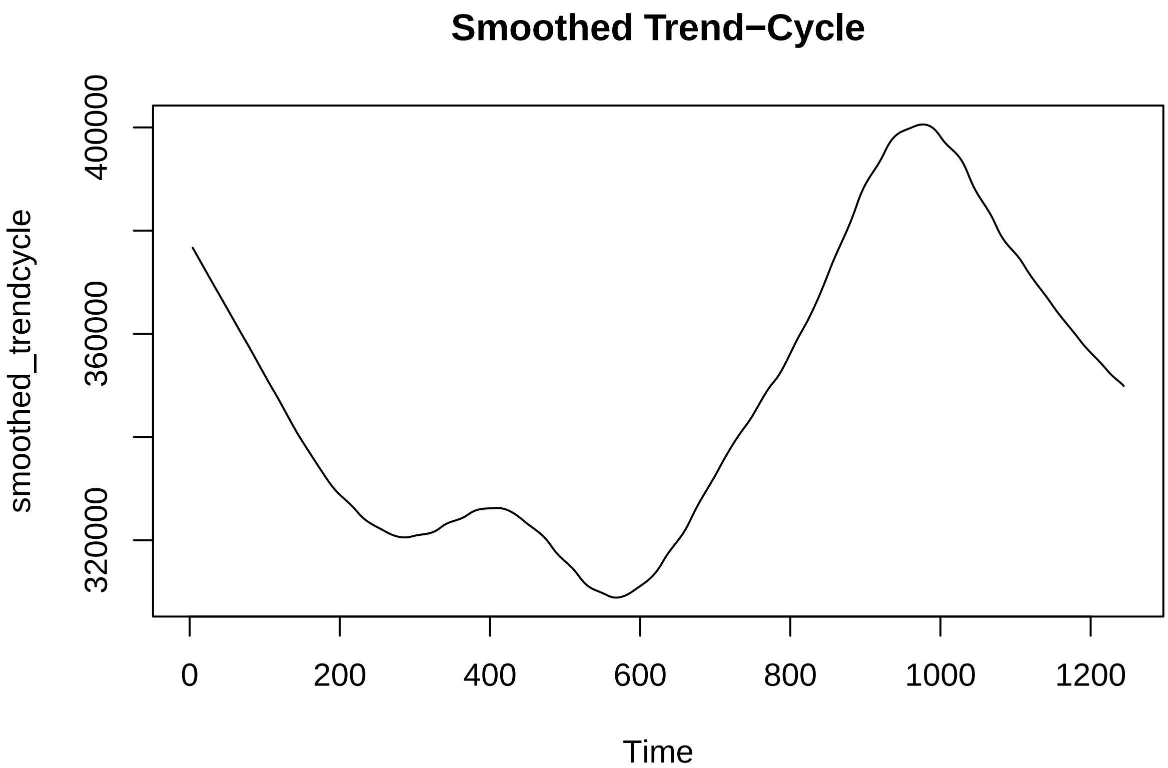

```r
plot(seasonal_consumption, main = "Seasonal Component")       # Seasonal
```



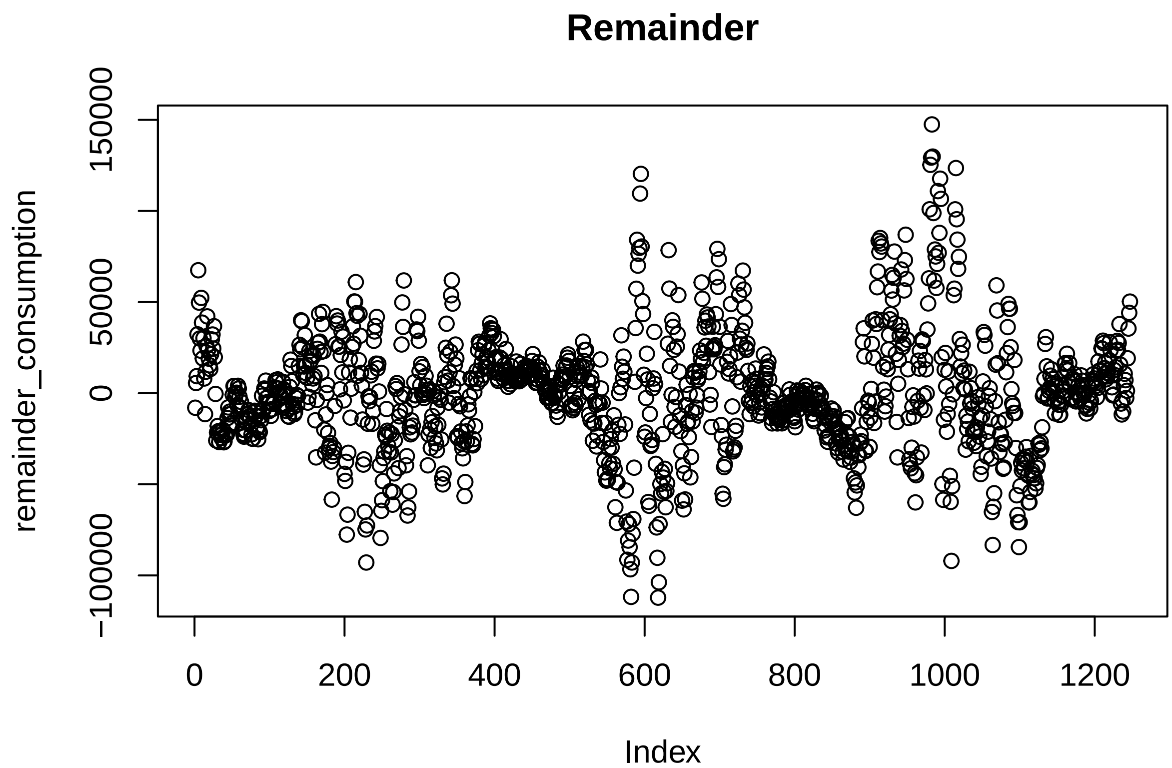

```r
plot(deseasoned_consumption, main = "Deseasoned Consumption")  # Deseasoned
```



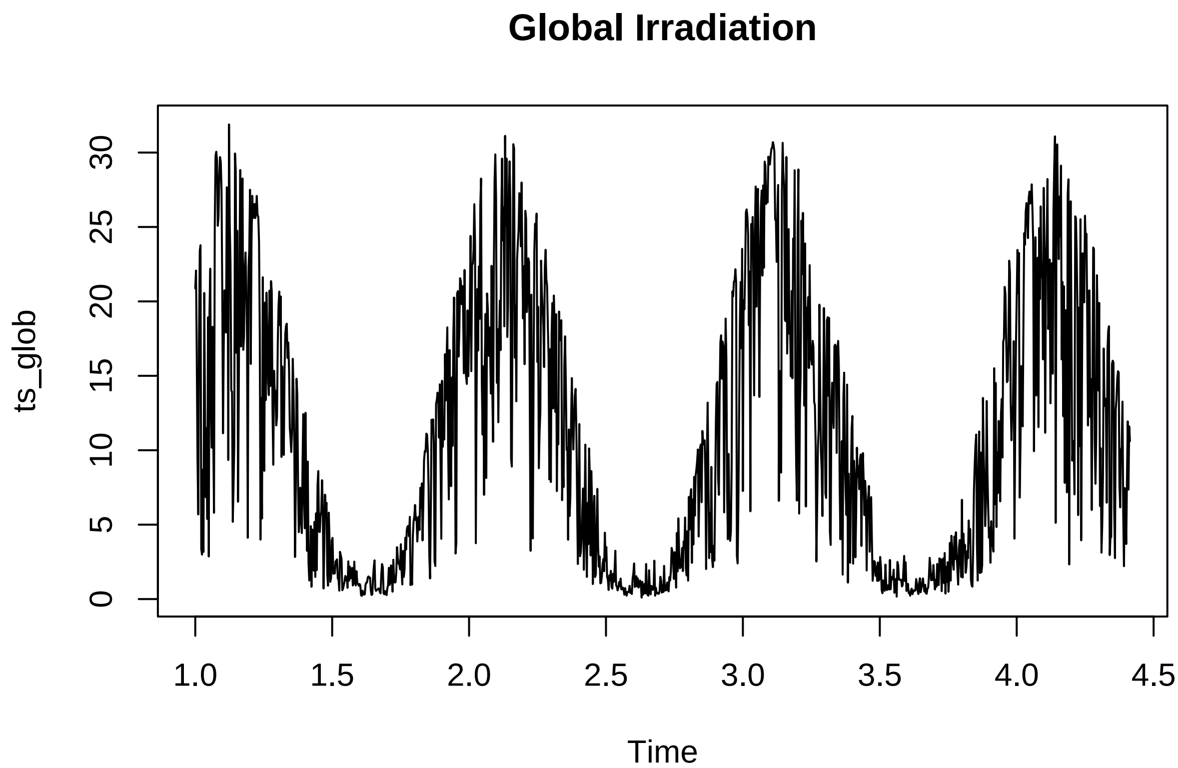

```r
plot(smoothed_trendcycle, main = "Smoothed Trend-Cycle")     # Smoothed trend-cycle
```



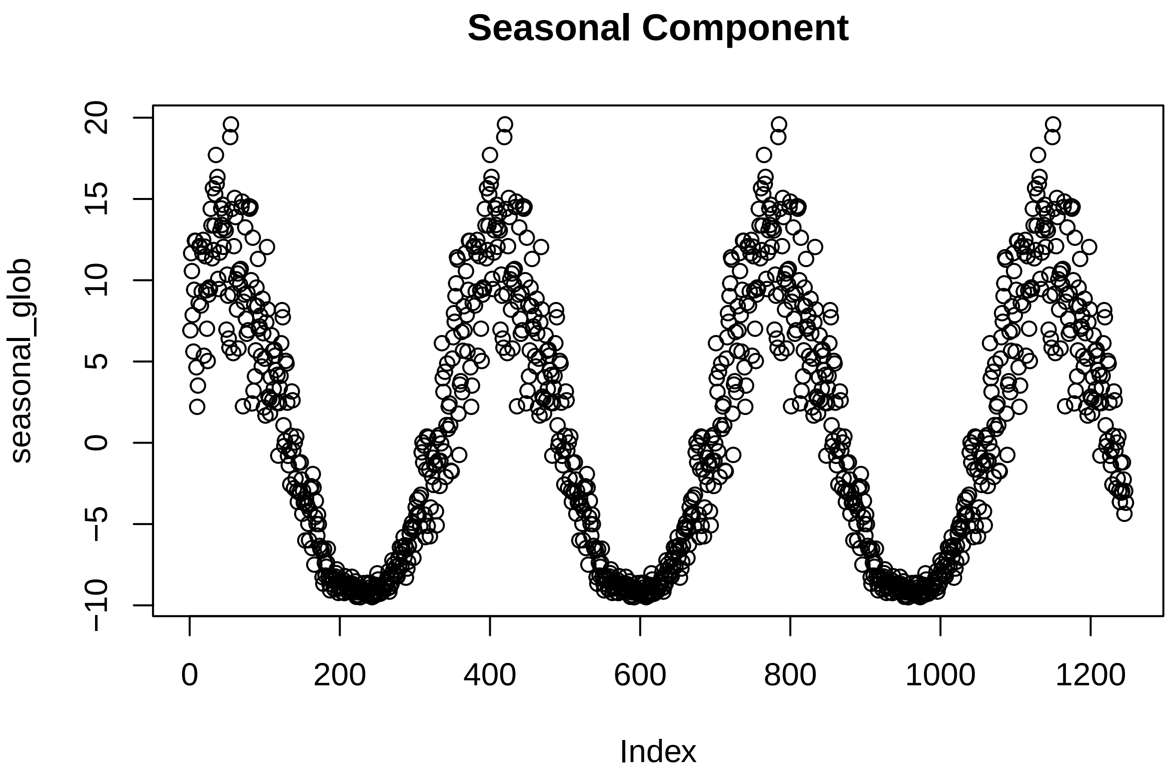

```r
plot(remainder_consumption, main = "Remainder")          # Remainder
```



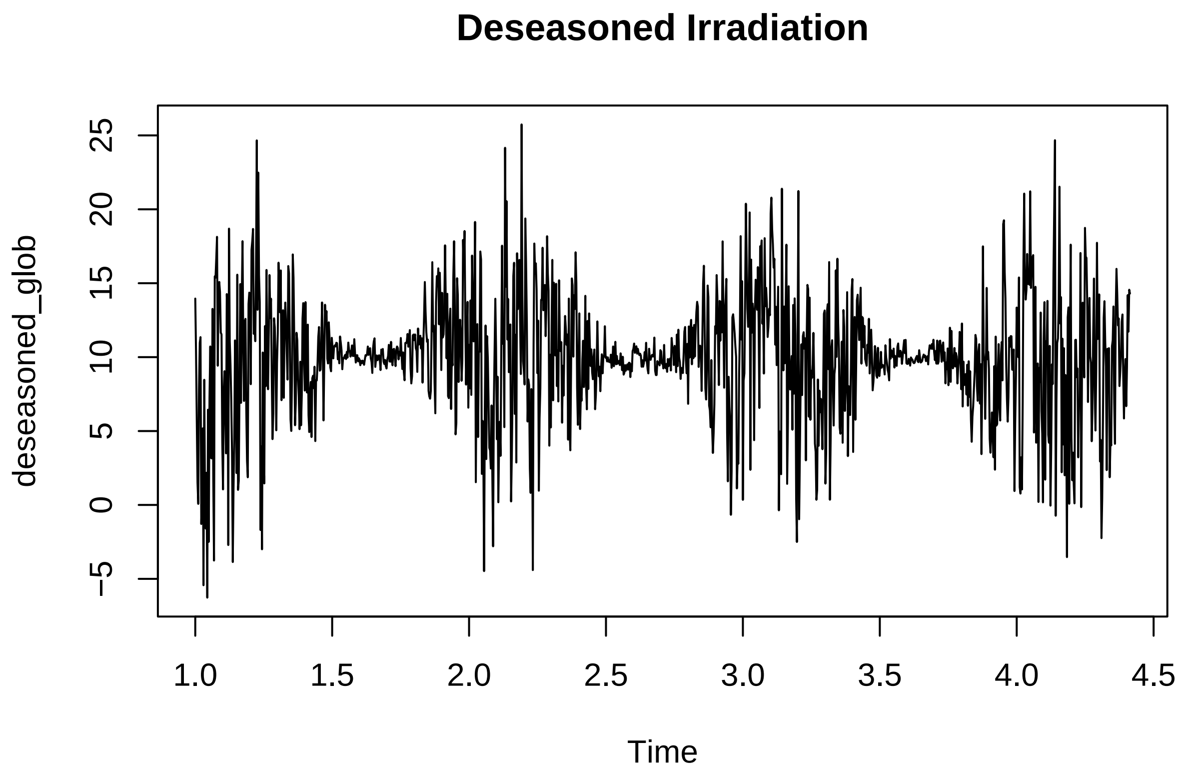

```
# --- Global Irradiation ---
ts_glob <- ts(na.remove(merged_data$GLOB), frequency = 365)
# Perform STL Decomposition
stl_glob <- stlplus(ts_glob, period = 365,
                    s.window = stl_parameters$s.window,
                    t.window = stl_parameters$t.window,
                    l.window = stl_parameters$l.window,
                    robust = stl_parameters$robust)
# Smooth the trend-cycle (using moving average, same order)
smoothed_trendcycle_glob <- ma(stl_glob$data[, "trend"], order = ma_order)
# Extract components (same approach as consumption)
seasonal_glob <- stl_glob$data[, "seasonal"]
deseasoned_glob <- ts_glob - seasonal_glob
trendcycle_glob <- stl_glob$data[, "trend"]
if ("cycle" %in% colnames(stl_glob$data)) {
  trendcycle_glob <- trendcycle_glob + stl_glob$data[, "cycle"]
}
remainder_glob <- stl_glob$data[, "remainder"]
#Plot components
plot(ts_glob, main = "Global Irradiation")
```



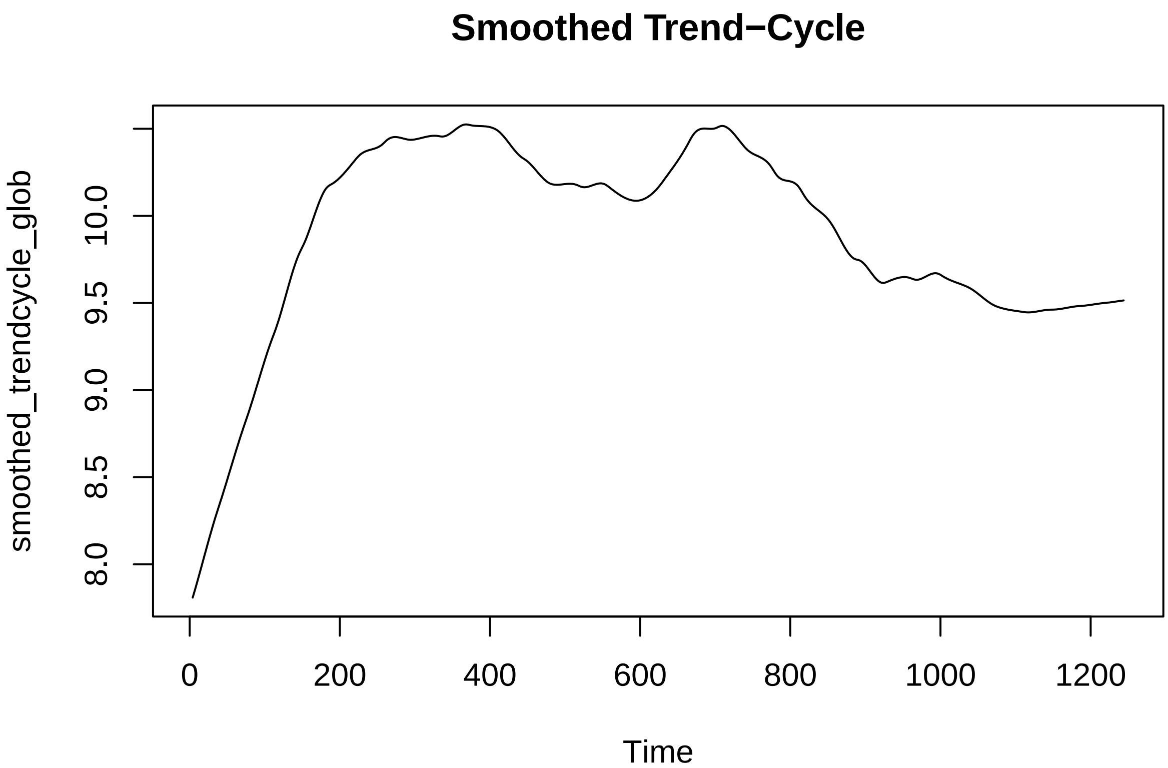

```
plot(seasonal_glob, main = "Seasonal Component")
```



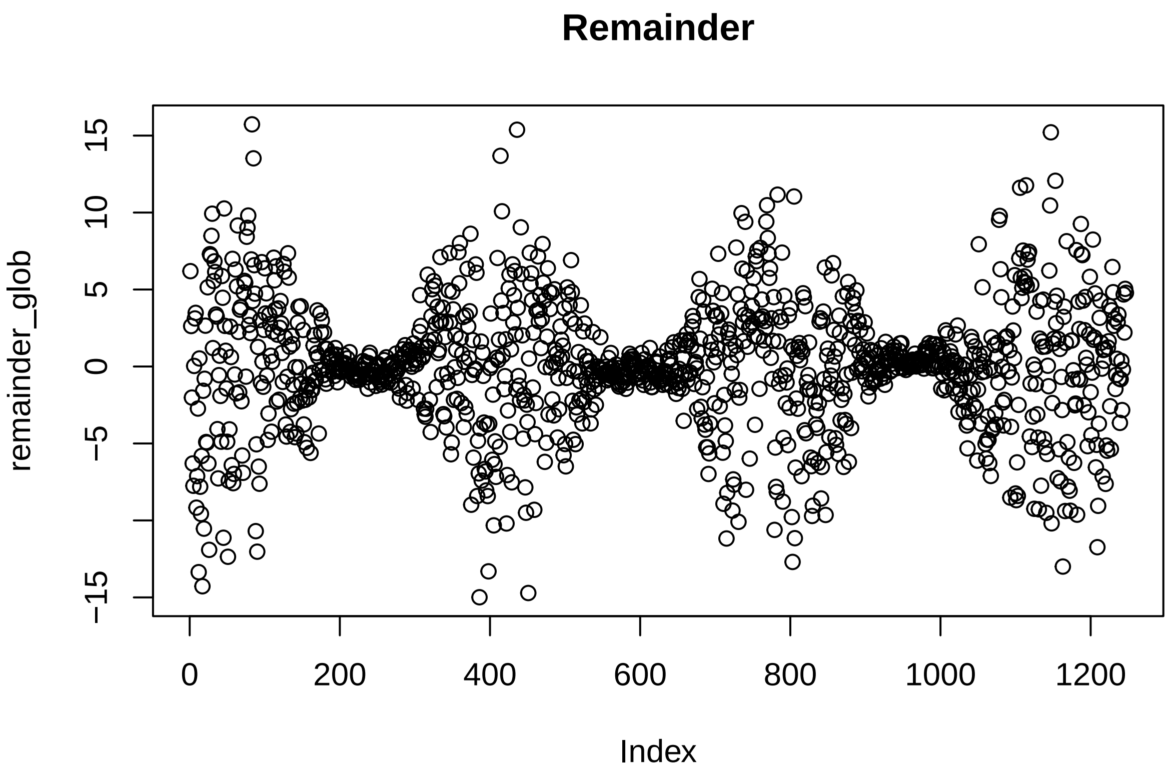

```r
plot(deseasoned_glob, main = "Deseasoned Irradiation")
```



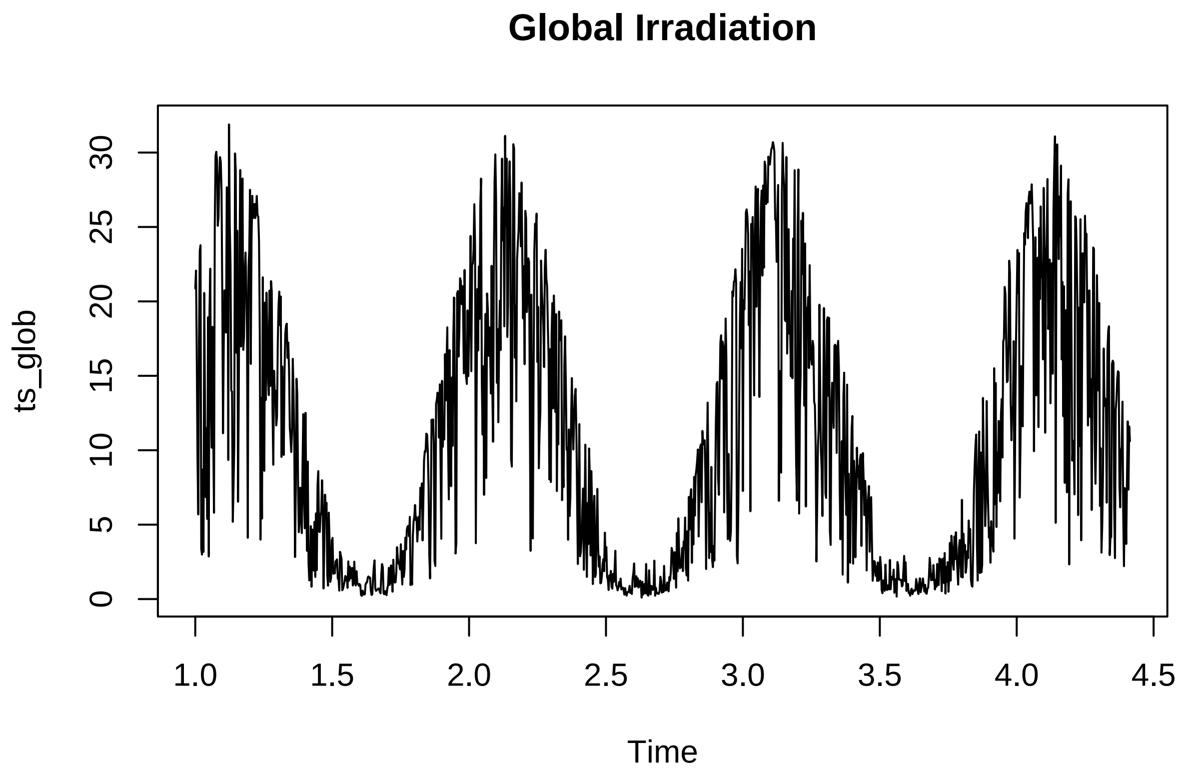

```
plot(smoothed_trendcycle_glob, main = "Smoothed Trend-Cycle") # Plot smoothed trend
```



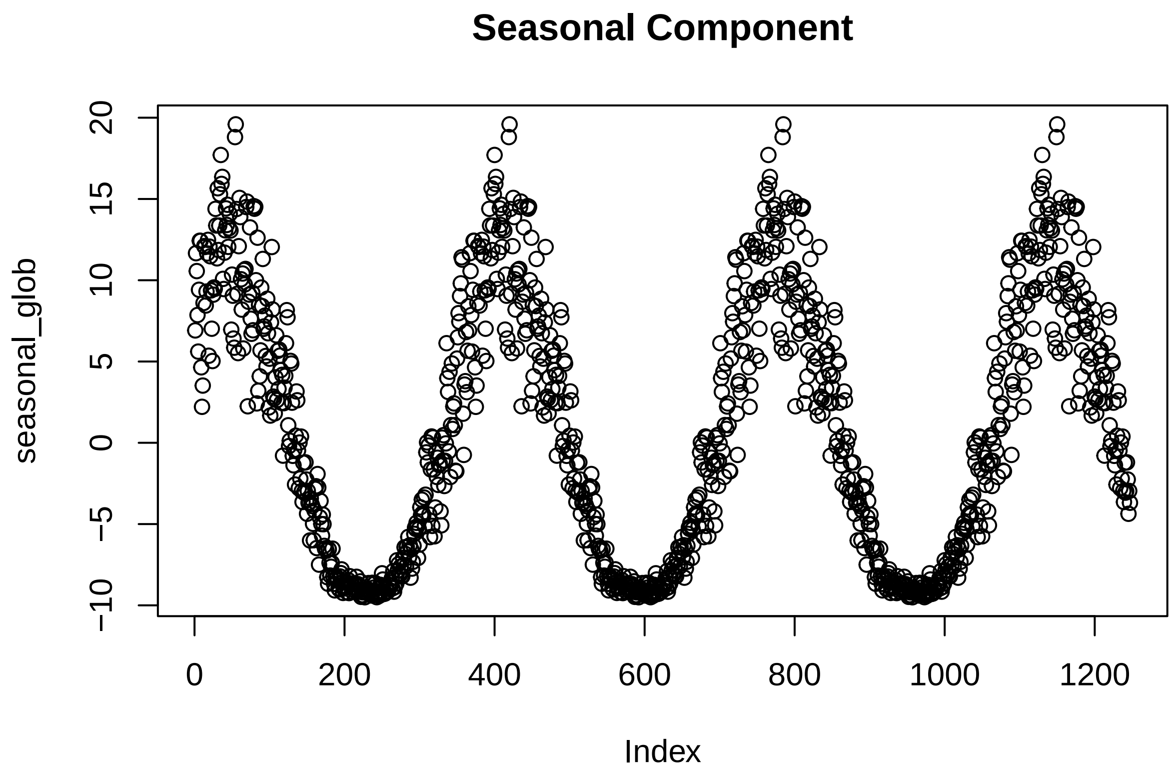

```
plot(remainder_glob, main = "Remainder")
```



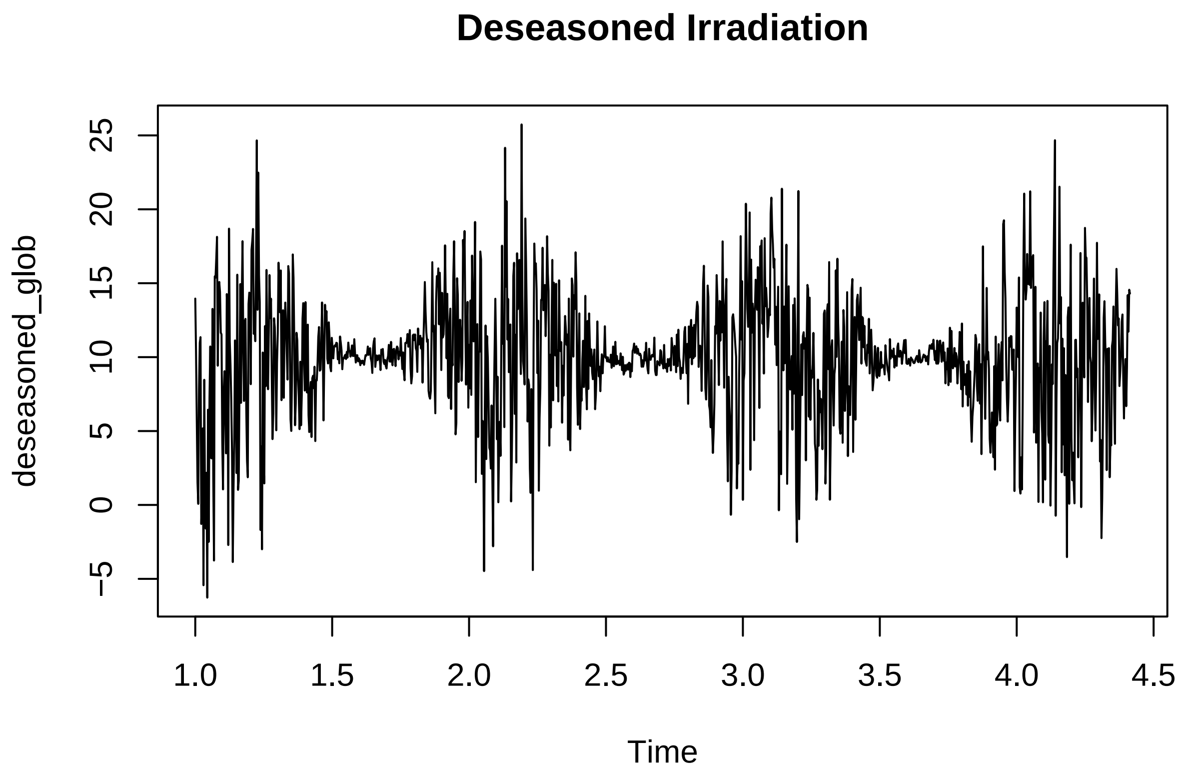

```
# --- Global Irradiation ---
ts_glob <- ts(na.remove(merged_data$GLOB), frequency = 365)
# Perform STL Decomposition
stl_glob <- stlplus(ts_glob, period = 365,
                    s.window = stl_parameters$s.window,
                    t.window = stl_parameters$t.window,
                    l.window = stl_parameters$l.window,
                    robust = stl_parameters$robust)
# Smooth the trend-cycle (using moving average, same order)
smoothed_trendcycle_glob <- ma(stl_glob$data[, "trend"], order = ma_order)
# Extract components (same approach as consumption)
seasonal_glob <- stl_glob$data[, "seasonal"]
deseasoned_glob <- ts_glob - seasonal_glob
trendcycle_glob <- stl_glob$data[, "trend"]
if ("cycle" %in% colnames(stl_glob$data)) {
  trendcycle_glob <- trendcycle_glob + stl_glob$data[, "cycle"]
}
remainder_glob <- stl_glob$data[, "remainder"]
#Plot components
plot(ts_glob, main = "Global Irradiation")
```



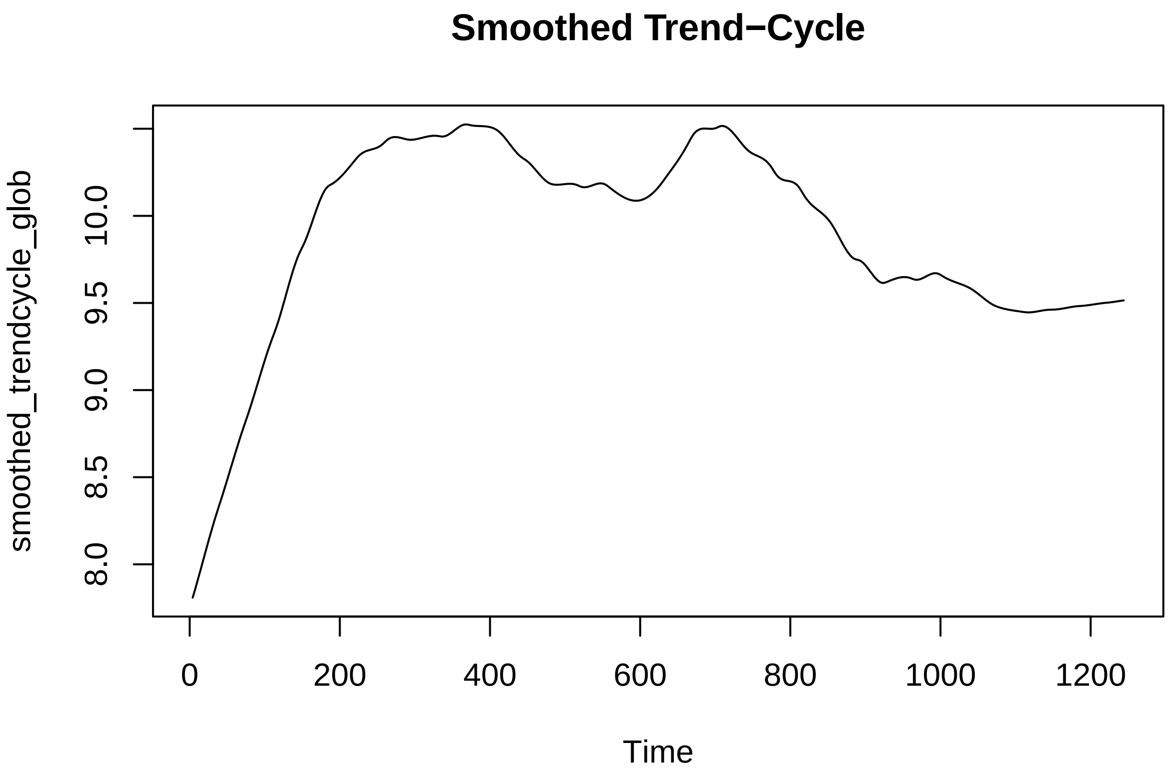

```
plot(seasonal_glob, main = "Seasonal Component")
```



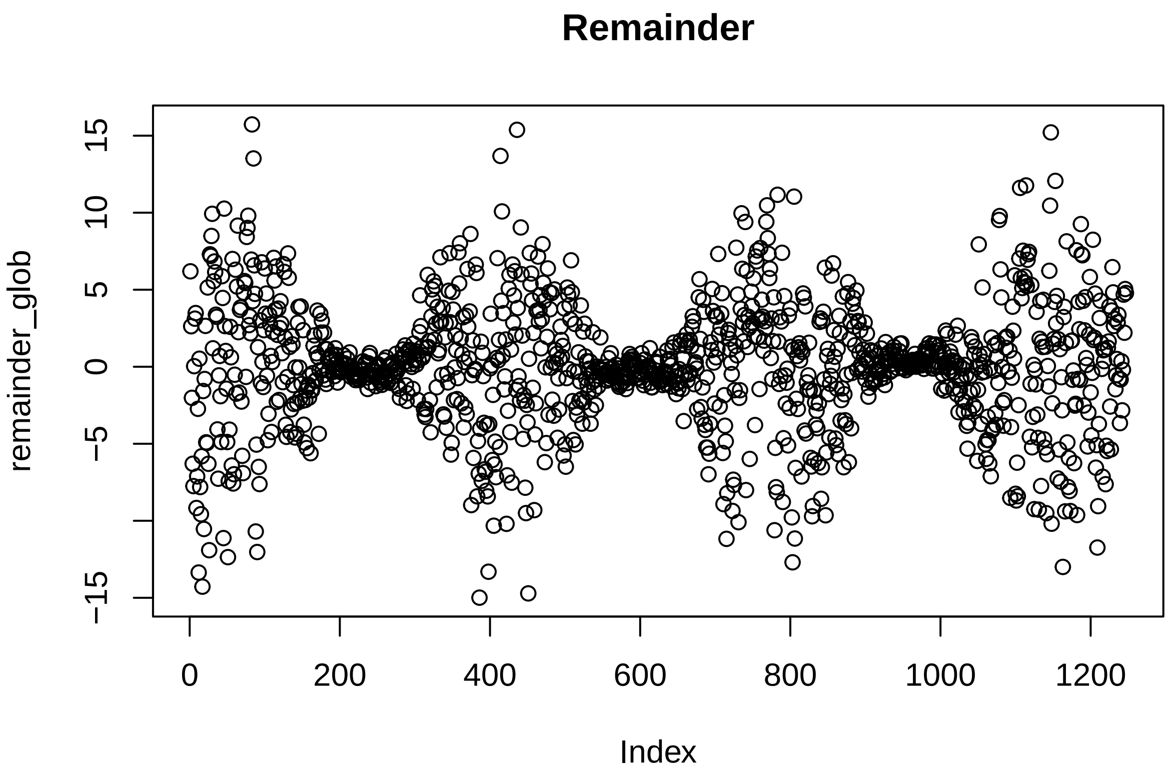

```r
plot(deseasoned_glob, main = "Deseasoned Irradiation")
```



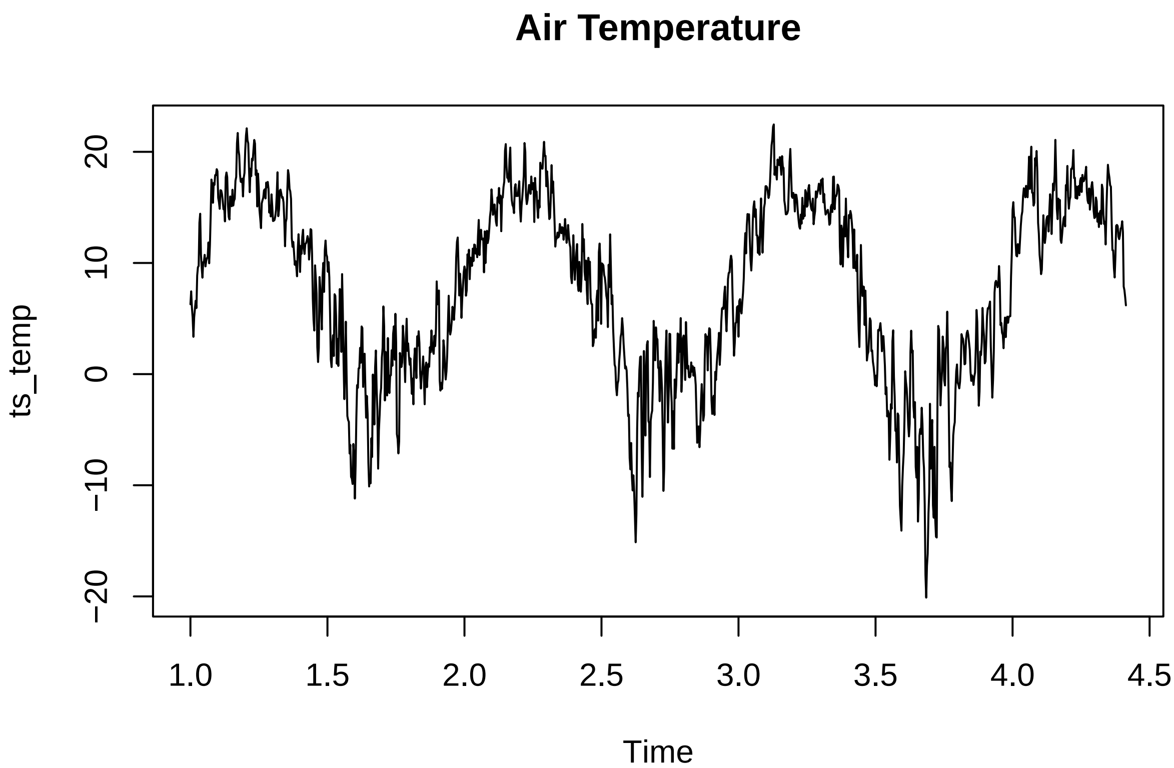

```
plot(smoothed_trendcycle_glob, main = "Smoothed Trend-Cycle") # Plot smoothed trend
```



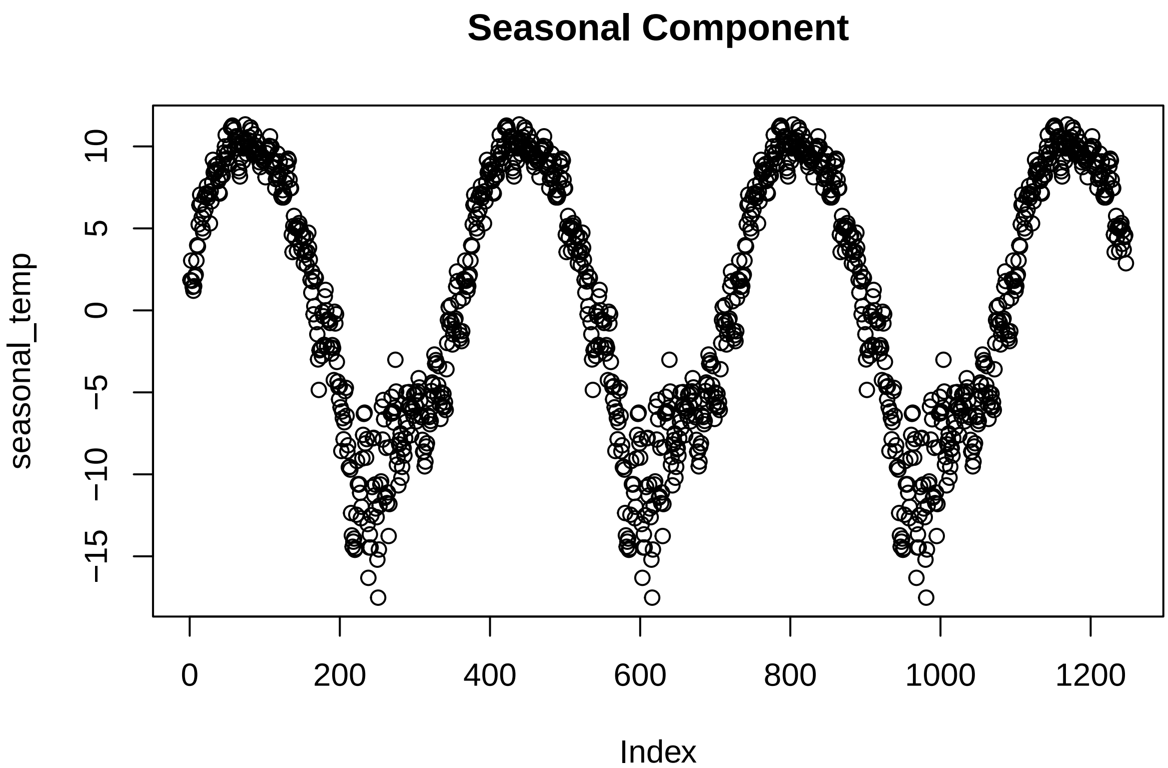

```
plot(remainder_glob, main = "Remainder")
```



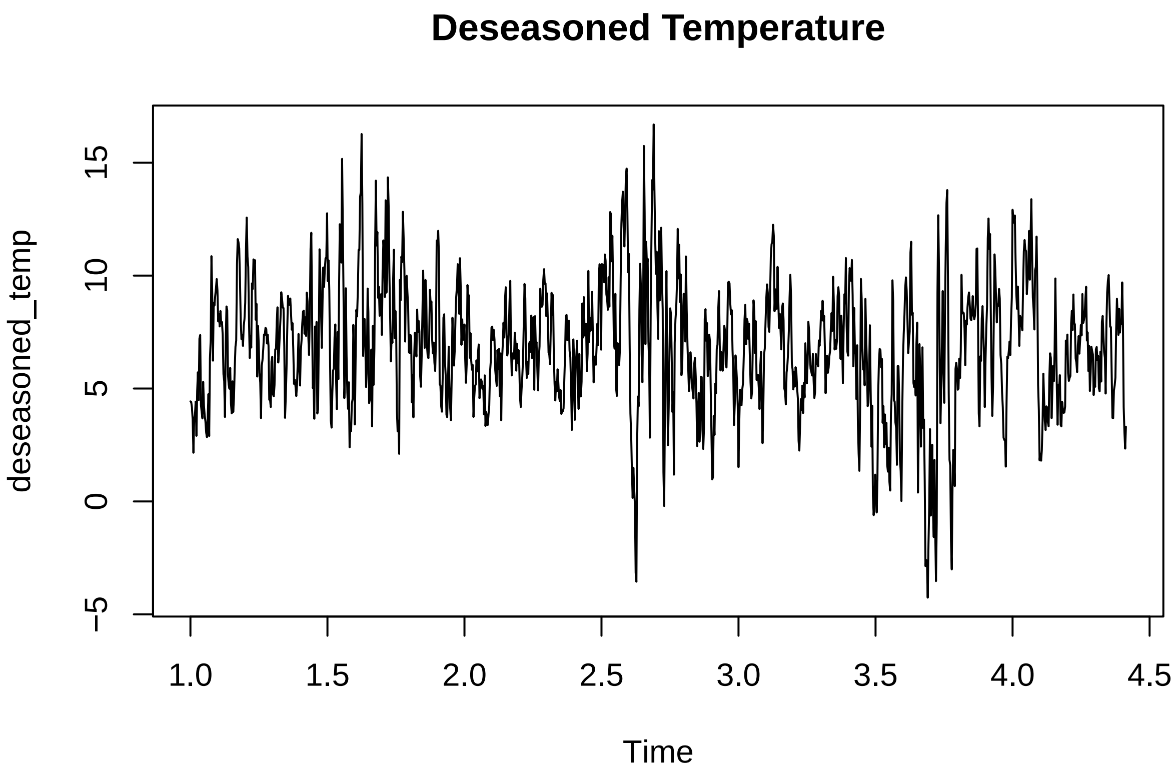

```
# --- Air Temperature ---
# ... (same structure as Global Irradiation) ...
ts_temp <- ts(na.remove(merged_data$LT), frequency = 365)
# Perform STL Decomposition
stl_temp <- stlplus(ts_temp, period = 365,
                    s.window = stl_parameters$s.window,
                    t.window = stl_parameters$t.window,
                    l.window = stl_parameters$l.window,
                    robust = stl_parameters$robust)
# Smooth the trend-cycle (using moving average, same order as others)
smoothed_trendcycle_temp <- ma(stl_temp$data[, "trend"], order = ma_order)
seasonal_temp <- stl_temp$data[, "seasonal"]
deseasoned_temp <- ts_temp - seasonal_temp
trendcycle_temp <- stl_temp$data[, "trend"]
if ("cycle" %in% colnames(stl_temp$data)) {
  trendcycle_temp <- trendcycle_temp + stl_temp$data[, "cycle"]
}
remainder_temp <- stl_temp$data[, "remainder"]
plot(ts_temp, main = "Air Temperature")
```



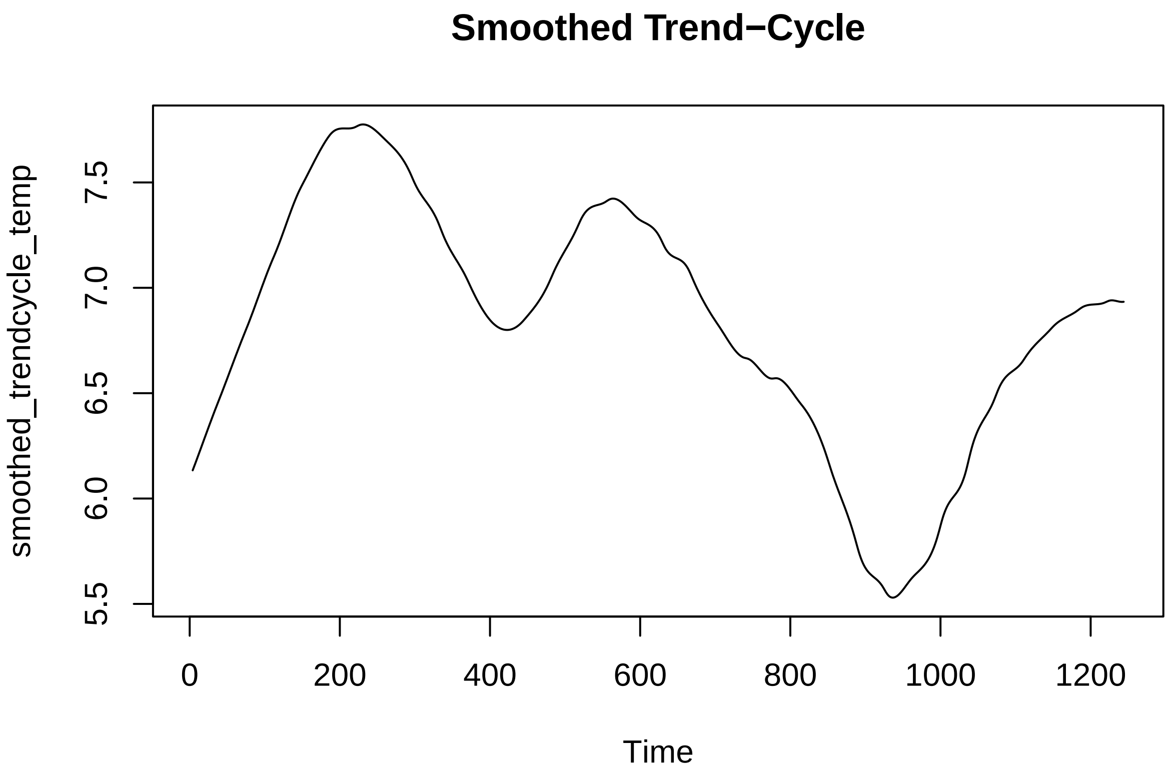

```
plot(seasonal_temp, main = "Seasonal Component")
```



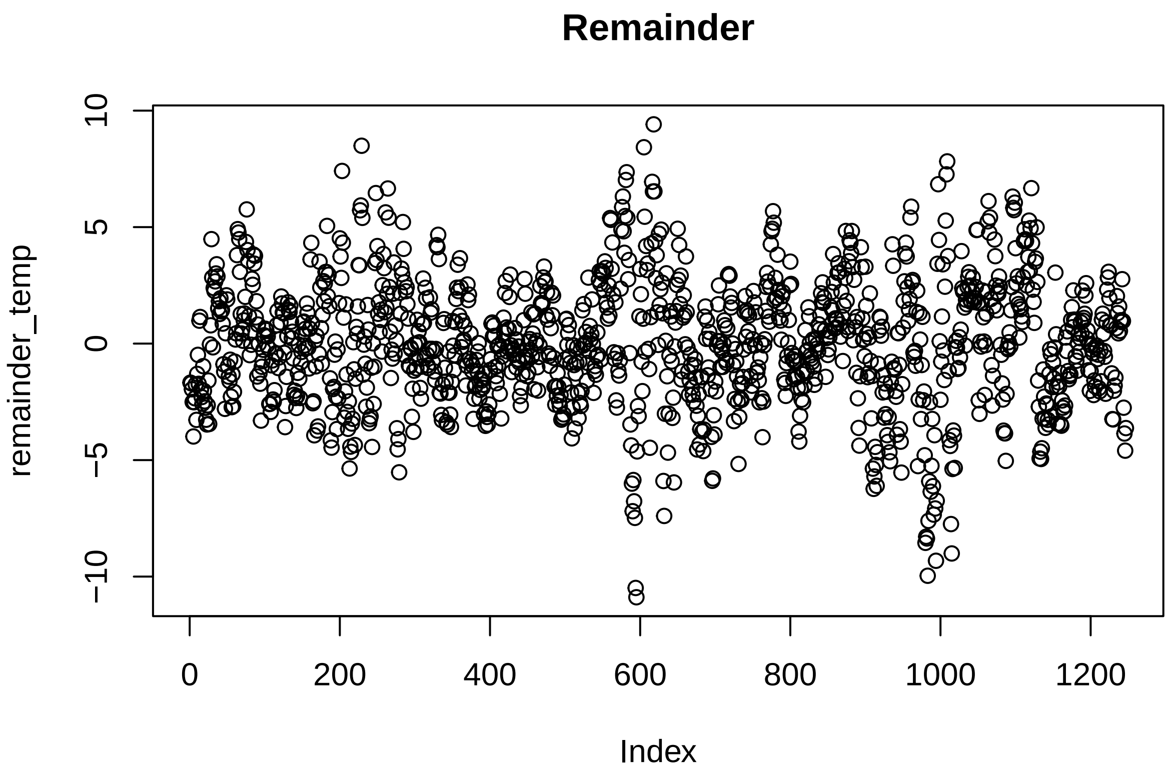

```
plot(deseasoned_temp, main = "Deseasoned Temperature")
```



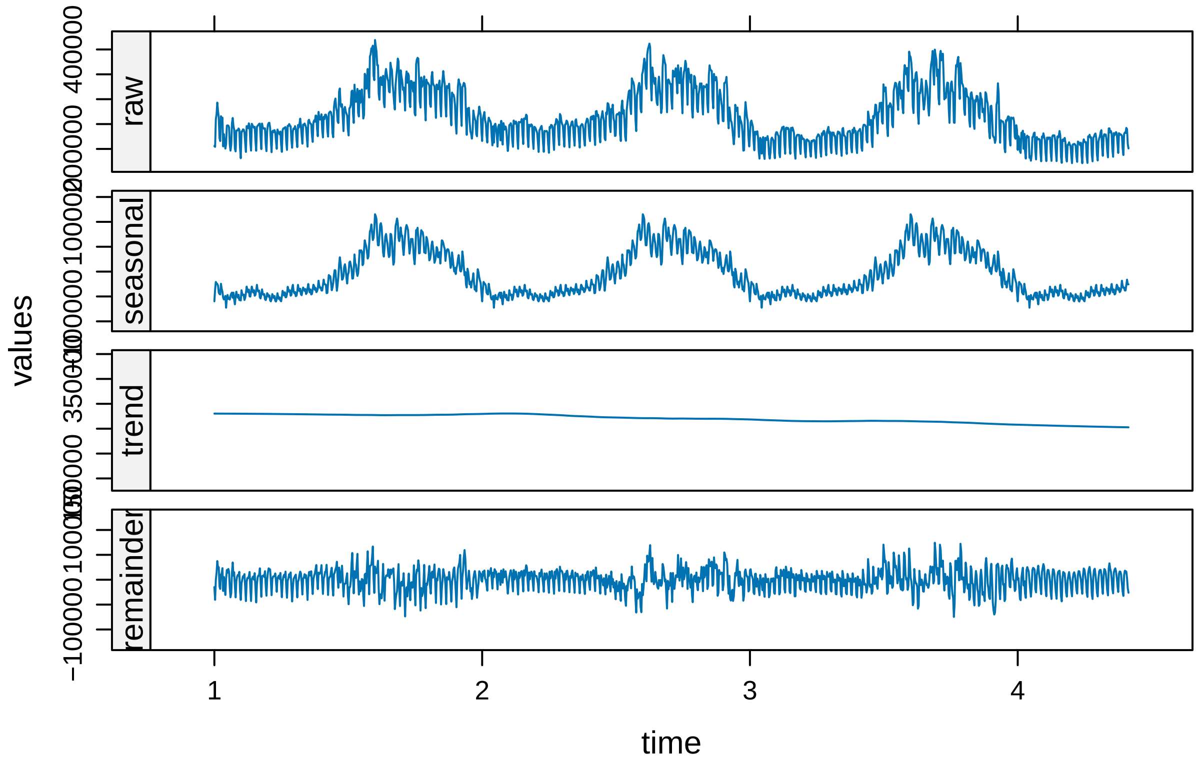

```r
plot(smoothed_trendcycle_temp, main = "Smoothed Trend-Cycle") # Plot smoothed trend
```



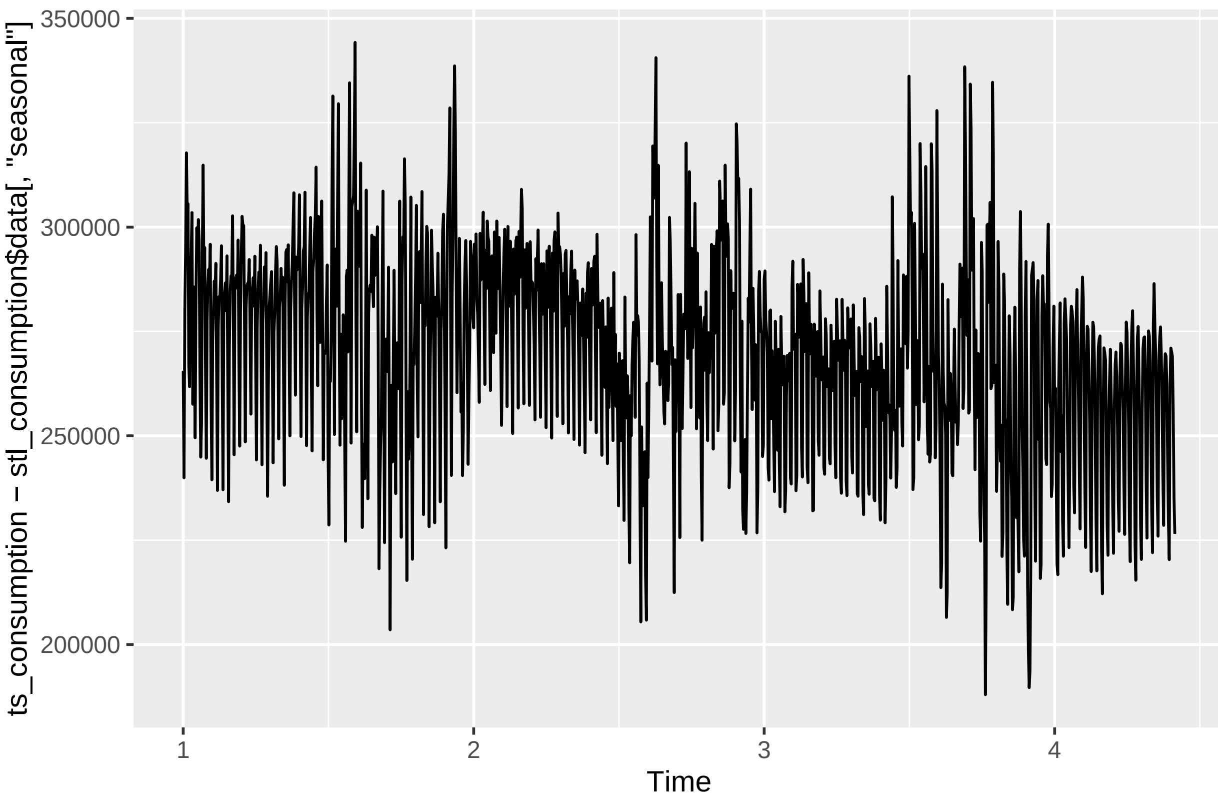

```
plot(remainder_temp, main = "Remainder")
```




```
ts_consumption <- ts(na.remove(merged_data$Forretning), frequency = 365)
stl_consumption <- stlplus(ts_consumption, period = 365, s.window = "periodic", t.window = 365, l.windo
plot(stl_consumption)
```




```r
# Remove seasonal component
autoplot(ts_consumption - stl_consumption$data[, "seasonal"])
```




```
# Repeat for Temperature and Global Irradiation...
```

Task G: Granger Causality Test

1. Hypotheses

- Null Hypothesis (H0): x does NOT Granger-cause y Past values of x do not help in predicting y.
- Alternative Hypothesis (H1): x DOES Granger-cause y. Past values of x provide statistically significant information about the future values of y.

2. Test Procedure (using AR and VAR models)

a. Univariate Autoregression (AR) Model for y:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

Model y only using its own past values (lags).

b. Vector Autoregression (VAR) Model for x and y:

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \beta_1 x_{t-1} + \cdots + \beta_p x_{t-p} + \varepsilon_t$$

Model y using both its own past values AND the past values of x.

c. Comparison: Use an F-test to compare the AR and VAR models. If the VAR model (with x) significantly improves prediction of y compared to the AR model (only y's history), reject the null hypothesis. The improvement in predictive ability is assessed by comparing residual sums of squares between restricted (AR) and unrestricted (VAR) models.V

```
# Granger Causality test function
granger_test <- function(y, x, group_name, var_name, maxlag = 5) {
  # Align time series and handle potential NA/length issues:
  # Find the maximum start date and minimum end date between the two time series
  start_date <- max(start(y), start(x))
  end_date <- min(end(y), end(x))
  # If the time series do not overlap, skip the test and show a warning
  if (start_date > end_date) {
    warning(paste("No overlapping dates for", group_name, "and", var_name))
    return(NULL)
  }
  # Window the time series to the overlapping date range
  y <- window(y, start = start_date, end = end_date)
  x <- window(x, start = start_date, end = end_date)
  # CRITICAL CHECK: Ensure time series have enough observations after windowing.
  # The Granger test needs at least 2 observations to perform the test.
  if (length(na.remove(y)) <= 1 || length(na.remove(x)) <= 1) {
    warning(paste("Time series too short for Granger test:", group_name, "or", var_name))
    return(NULL)
  }
  # Combine the two time series into a data frame for the Granger test
  combined_data <- data.frame(y = y, x = x)
  # Display a message indicating the variables being tested
  print(paste("Granger Causality Test:", var_name, "->", group_name))
  # Perform the Granger Causality Test using the grangertest() function from the lmtest package
  test_result <- tryCatch({
    # Run the Granger Causality Test with a specified maximum lag (default is 5)
    granger_result <- grangertest(y ~ x, order = maxlag, data = combined_data)
    return(granger_result)
  }, error = function(e) {
    # If an error occurs (e.g., due to data issues), catch and display it
    cat("Error in Granger test:", e, "\n")
    return(NULL)
  })
  # Print the results of the Granger Causality test if successful
  if (!is.null(test_result)) {
    print(test_result)
  } else {
    # If no result, show a message indicating no result was generated
    print(paste("No result for", var_name, "->", group_name))
  }
  # Return the test result (if any)
  return(test_result)
}
```

Example: Granger Causality Test between LT and Temperature

```
# Extract the consumption data for the specific group (e.g., 'Privat')
consumer_groups <- c("Privat", "Forretning", "Industri")
for (group in consumer_groups) {
  consumption_data <- merged_data[, group]
  ts_consumption <- ts(na.remove(consumption_data), frequency = 365)  # Convert to time series with dai
```

```r
  # Extract temperature (LT) data, assuming it is available in merged_data
  ts_temp <- ts(na.remove(merged_data$LT), frequency = 365)  # Convert to time series with daily freque

  # Perform the Granger Causality test for Consumption vs Temperature
  granger_result <- granger_test(ts_consumption, ts_temp, group, "Temperature (LT)")

  # Extract Global Irradiation (GLOB) data
  ts_glob <- ts(na.remove(merged_data$GLOB), frequency = 365)

  # Perform the Granger Causality test for Consumption vs Global Irradiation
  granger_result <- granger_test(ts_consumption, ts_glob, group, "Global Irradiation (GLOB)")

  # Perform the Granger Causality test for Temperature vs Global Irradiation
  granger_result <- granger_test(ts_temp, ts_glob, group, "Temperature (LT)")
}
```

```
## [1] "Granger Causality Test: Temperature (LT) -> Privat"
## [1] "Granger Causality Test: Global Irradiation (GLOB) -> Privat"
## [1] "Granger Causality Test: Temperature (LT) -> Privat"
## [1] "Granger Causality Test: Temperature (LT) -> Forretning"
## [1] "Granger Causality Test: Global Irradiation (GLOB) -> Forretning"
## [1] "Granger Causality Test: Temperature (LT) -> Forretning"
## [1] "Granger Causality Test: Temperature (LT) -> Industri"
## [1] "Granger Causality Test: Global Irradiation (GLOB) -> Industri"
## [1] "Granger Causality Test: Temperature (LT) -> Industri"
```

Task H: Forecasting with ARIMA Model

ARIMA Model Forecasting function

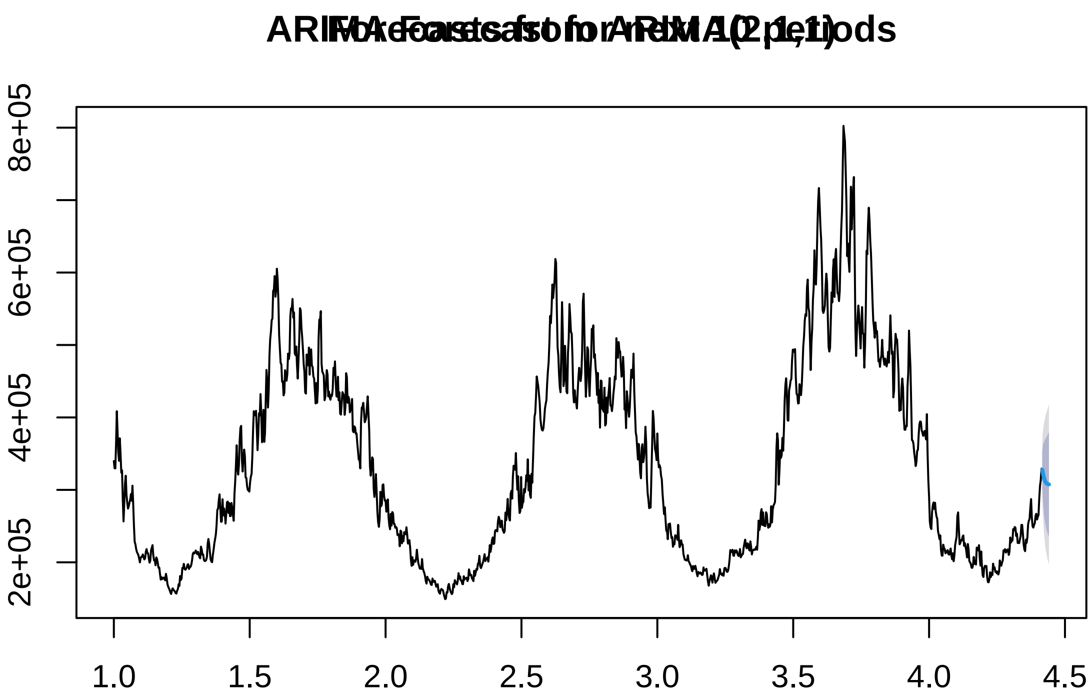
```r
arima_forecast <- function(time_series, max_order = 5, h = 10) {
  # Ensure the time series is a univariate time series (vector)
  if (is.null(time_series) || length(time_series) < 2) {
    stop("Time series is too short for forecasting")
  }

  # Check for missing data and handle it
  time_series <- na.remove(time_series)  # Remove NA values from the series

  # Fit an ARIMA model with automatic order selection
  # auto.arima function automatically selects the best ARIMA model based on AICc criterion
  model <- auto.arima(time_series, max.p = max_order, max.q = max_order, seasonal = FALSE)

  # Display the selected ARIMA model
  print(paste("Fitted ARIMA Model:", as.character(model)))

  # Forecast the next 'h' periods (e.g., 10 days)
  forecast_result <- forecast(model, h = h)

  # Plot the forecasted values along with the historical data
  plot(forecast_result)
  title(main = paste("ARIMA Forecast for next", h, "periods"))

  # Return the forecast results (point forecast, lower and upper prediction intervals)
```

```
  return(forecast_result)
}
```

Example: Forecasting Consumption for a Specific Group (Privat)

```
# Let's forecast the consumption for the 'Privat' group over the next 10 periods
# Extract consumption data for the 'Privat' group from the merged dataset
consumption_data <- merged_data$Privat
# Convert consumption data to a time series object, handling NAs
ts_consumption <- ts(na.remove(consumption_data), frequency = 365)  # Daily frequency (365 days per yea
# Apply ARIMA model forecasting for the next 10 periods (e.g., days or time units)
forecast_result <- arima_forecast(ts_consumption, max_order = 5, h = 10)
```

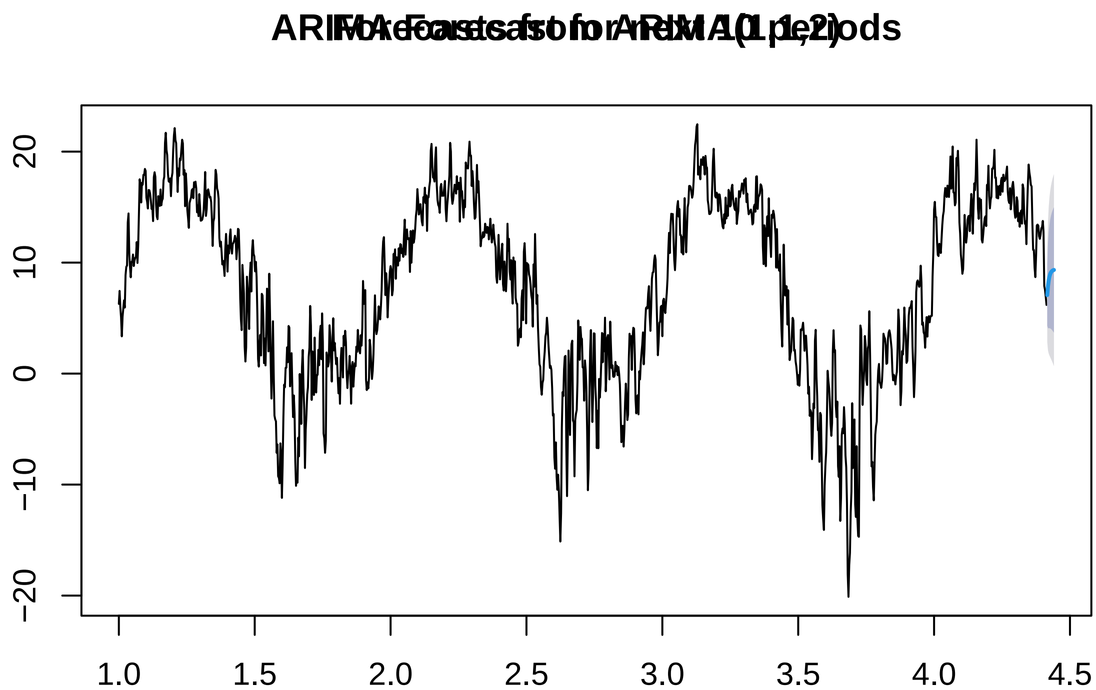
```
## [1] "Fitted ARIMA Model: ARIMA(2,1,1)"
```



Example: Forecasting Temperature (LT)

```
# Now, let's forecast the temperature (LT) for the next 10 periods
# Extract temperature data from merged dataset
temperature_data <- merged_data$LT
# Convert temperature data to a time series object
ts_temperature <- ts(na.remove(temperature_data), frequency = 365)
# Apply ARIMA model forecasting for the next 10 periods (e.g., days or time units)
forecast_result_temp <- arima_forecast(ts_temperature, max_order = 5, h = 10)
```

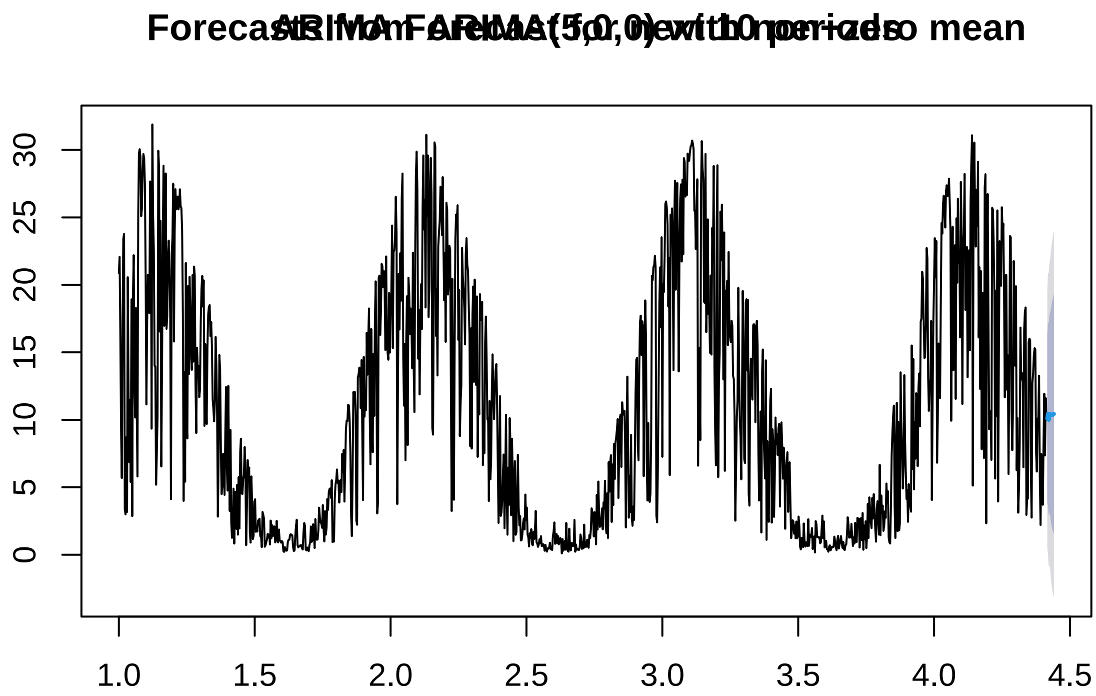
```
## [1] "Fitted ARIMA Model: ARIMA(1,1,2)"
```



Example: Forecasting Global Irradiation (GLOB)

```r
# Similarly, forecast Global Irradiation (GLOB) for the next 10 periods
# Extract global irradiation data from merged dataset
glob_data <- merged_data$GLOB
# Convert global irradiation data to a time series object
ts_glob <- ts(na.remove(glob_data), frequency = 365)
# Apply ARIMA model forecasting for the next 10 periods (e.g., days or time units)
forecast_result_glob <- arima_forecast(ts_glob, max_order = 5, h = 10)
```

```
## [1] "Fitted ARIMA Model: ARIMA(5,0,0) with non-zero mean"
```



Task I: Evaluate Forecast Accuracy for ARIMA Model

```r
# Evaluation function to calculate forecast accuracy
evaluate_forecast_accuracy <- function(forecast_result, actual_data) {
  # Ensure that both forecast and actual data are provided and have the same length
  if (length(forecast_result$mean) != length(actual_data)) {
    stop("The forecast and actual data must have the same length")
  }
  # Calculate accuracy measures: MAE, MSE, RMSE, MAPE, etc.
  accuracy_measures <- accuracy(forecast_result, actual_data)
  # Print out the accuracy measures
  print("Accuracy Measures:")
  print(accuracy_measures)
  # Return the accuracy measures for further use
  return(accuracy_measures)
}
```

Example 1: Evaluating Forecast Accuracy for Consumption (VOLUM_KWH)

```r
# Get the actual consumption data for the last 10 periods
# Assuming that actual future consumption data is available for comparison
actual_consumption <- merged_data$Privat[(length(merged_data$Privat)-9):length(merged_data$Privat)]
# Compare with the forecasted consumption
# Use the forecasted result from Task H
forecast_accuracy_consumption <- evaluate_forecast_accuracy(forecast_result, actual_consumption)
```

```
## [1] "Accuracy Measures:"
##                        ME     RMSE      MAE         MPE      MAPE      MASE
## Training set    -39.88407 19696.33 13877.23  -0.1784362  3.931527 0.9876033
## Test set     -32581.73653 44362.18 38091.36 -12.5854245 14.275539 2.7108558
##                    ACF1
## Training set 0.00248148
## Test set             NA
```

Example 2: Evaluating Forecast Accuracy for Temperature (LT)

```r
# Get the actual temperature data for the last 10 periods
actual_temperature <- merged_data$LT[(length(merged_data$LT)-9):length(merged_data$LT)]
# Compare with the forecasted temperature
# Use the forecasted result from Task H
forecast_accuracy_temperature <- evaluate_forecast_accuracy(forecast_result_temp, actual_temperature)
```

```
## [1] "Accuracy Measures:"
##                       ME     RMSE      MAE         MPE     MAPE      MASE
## Training set 0.006203914 2.151897 1.618597 -103.759734 189.2093 0.9835212
## Test set     1.860154277 3.808334 3.582179    8.581756  33.6652 2.1766677
##                      ACF1
## Training set 4.783413e-05
## Test set               NA
```

Example 3: Evaluating Forecast Accuracy for Global Irradiation (GLOB)

```r
# Get the actual global irradiation data for the last 10 periods
actual_glob <- merged_data$GLOB[(length(merged_data$GLOB)-9):length(merged_data$GLOB)]
# Compare with the forecasted global irradiation
# Use the forecasted result from Task H
forecast_accuracy_glob <- evaluate_forecast_accuracy(forecast_result_glob, actual_glob)
```

```
## [1] "Accuracy Measures:"
##                       ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -0.01252225 4.822953 3.443017 -56.68061 77.53667 0.9358132
## Test set     -2.77529483 4.109704 3.359272 -75.95699 80.94471 0.9130512
##                     ACF1
## Training set -0.02502515
## Test set              NA
```