Electricity Consumption Forecast Project

Sourena Mohit Tabatabaie

Part 1: Initialization and Data Exploration. In this first step, The goal is to load the data, clean it, and understand its structure and underlying patterns. This is the most critical part of any data analysis project.

```
## ---- setup, message=FALSE, warning=FALSE --------------------------------
# Repro & options
set.seed(123)
options(dplyr.summarise.inform = FALSE)

# Core libs
library(readxl)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

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

```
library(tidyr)
library(lubridate)
```

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

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

```
library(stringr)
library(janitor)
```

```
##
## Attaching package: 'janitor'
```

```
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
library(tsibble)
```

```
## Registered S3 method overwritten by 'tsibble':
##   method               from
##   as_tibble.grouped_df dplyr
```

```
##
## Attaching package: 'tsibble'
```

```
## The following object is masked from 'package:lubridate':
##
##     interval
```

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

```
library(feasts)   # plots/ACF later
```

```
## Loading required package: fabletools
```

```
library(ggplot2)

# File name (adjust path if needed)
data_path <- "2025-06-Elec-train.xlsx"
```

The exam specifies that the file 2023-11-Elec-train.xlsx contains electricity consumption and outdoor air temperature measured every 15 minutes from January 1, 2010, 1:15 to February 20, 2010, 23:45. Additionally, outdoor air temperature for February 21, 2010, is available for the forecast period. We need to load this data, separate the training period from the future temperature covariates, and convert them into ts (time series) objects, which are fundamental for time series analysis in R. A key aspect for ts objects is their frequency. Since measurements are every 15 minutes, there are 96 observations in a full day (24 hours * 4 15-min intervals/hour). Thus, the frequency will be 96.

```
## -- GLOBAL helpers (visible to any other functions/chunks) ---------------
row_mat <- function(x, p = length(x)) {
  matrix(as.numeric(x), nrow = 1, ncol = p, byrow = TRUE)
}
```

```
## ---- load ------------------------------------------------------------
raw <- readxl::read_excel(data_path)
df  <- raw %>%
  janitor::clean_names() %>%
  mutate(across(everything(), ~ .x))  # no-op; placeholder if we coerce types later

# Peek
glimpse(df)
```

```
## Rows: 4,987
## Columns: 3
## $ timestamp <chr> "40179.052083333336", "1/1/2010 1:30", "1/1/2010 1:45", "1/1…
## $ power_k_w <dbl> 165.1, 151.6, 146.9, 153.7, 153.8, 159.0, 157.7, 163.2, 151.…
## $ temp_c    <dbl> 10.555556, 10.555556, 10.555556, 10.555556, 10.555556, 10.55…
```

```
head(df, 3)
```

```
## # A tibble: 3 × 3
##   timestamp          power_k_w temp_c
##   <chr>                  <dbl>  <dbl>
## 1 40179.052083333336      165.   10.6
## 2 1/1/2010 1:30           152.   10.6
## 3 1/1/2010 1:45           147.   10.6
```

```
tail(df, 3)
```

```
## # A tibble: 3 × 3
##   timestamp       power_k_w temp_c
##   <chr>               <dbl>  <dbl>
## 1 2/21/2010 23:15        NA     10
## 2 2/21/2010 23:30        NA     10
## 3 2/21/2010 23:45        NA     10
```

As we can see , there is an issue for reading the first row time ; that first timestamp “40179.052083333336” is an Excel date-time serial, while the rest are regular strings like “1/1/2010 1:30”. So the column is mixed format. We’ll parse both reliably.

```
## ---- map-columns-explicit --------------------------------------------
# Using your actual column names
time_col <- "timestamp"
y_col    <- "power_k_w"
t_col    <- "temp_c"

stopifnot(all(c(time_col, y_col, t_col) %in% names(df)))
cat("Mapped columns:\n time ->", time_col, "\n kW ->", y_col, "\n temp ->", t_col, "\n")
```

```
## Mapped columns:
##  time -> timestamp
##  kW -> power_k_w
##  temp -> temp_c
```

```r
## ---- parse-time-clean (robust) ------------------------------------------
## ---- parse-time-clean (snap15), message=FALSE, warning=FALSE --------------
library(lubridate)
# Excel serial (Windows) -> POSIXct UTC
excel_serial_to_posix <- function(x) {
  as_datetime((x - 25569) * 86400, tz = "UTC")
}

# Mixed serial + string parser
parse_timestamp <- function(v) {
  v_chr  <- as.character(v)
  num    <- suppressWarnings(as.numeric(v_chr))
  is_num <- !is.na(num)

  out <- as.POSIXct(rep(NA_real_, length(v_chr)), origin = "1970-01-01", tz = "UTC")

  if (any(is_num)) {
    out[is_num] <- excel_serial_to_posix(num[is_num])
  }
  if (any(!is_num)) {
    out[!is_num] <- suppressWarnings(lubridate::parse_date_time(
      v_chr[!is_num],
      orders = c("mdy HMS","mdy HM","dmy HMS","dmy HM","ymd HMS","ymd HM"),
      tz = "UTC"
    ))
  }
  out
}

# Snap to nearest 15 minutes (handles tiny float offsets from Excel)
snap_15min <- function(t) {
  s <- as.numeric(t)                # seconds since epoch
  snapped <- round(s / 900) * 900   # 900 sec = 15 min
  as_datetime(snapped, tz = "UTC")
}

dat <- df %>%
  transmute(
    timestamp = snap_15min(parse_timestamp(.data[[time_col]])),
    kW        = suppressWarnings(as.numeric(.data[[y_col]])),
    temp      = suppressWarnings(as.numeric(.data[[t_col]]))
  ) %>%
  arrange(timestamp)

# Quick diagnostics
cat("Unique minutes (mod 15) after snap: ",
    paste(sort(unique(minute(dat$timestamp) %% 15)), collapse=", "), "\n", sep = "")
```

```
## Unique minutes (mod 15) after snap: 0
```

```r
head(dat$timestamp, 5)
```

```
## [1] "2010-01-01 01:15:00 UTC" "2010-01-01 01:30:00 UTC"
## [3] "2010-01-01 01:45:00 UTC" "2010-01-01 02:00:00 UTC"
## [5] "2010-01-01 02:15:00 UTC"
```

```r
summary(dat$timestamp)
```

```
##                  Min.               1st Qu.                Median
## "2010-01-01 01:15:00" "2010-01-14 00:52:30" "2010-01-27 00:30:00"
##                  Mean               3rd Qu.                  Max.
## "2010-01-27 00:30:00" "2010-02-09 00:07:30" "2010-02-21 23:45:00"
```

```
## ---- regularity-check -----------------------------------------------------
## ---- regularity-check (snap15), message=FALSE, warning=FALSE --------------
library(dplyr)
library(tsibble)

dup_n <- dat %>% count(timestamp) %>% filter(n > 1) %>% nrow()

tsib <- dat %>% distinct(timestamp, .keep_all = TRUE)

full_index <- tibble::tibble(
  timestamp = seq(
    from = min(tsib$timestamp, na.rm = TRUE),
    to   = max(tsib$timestamp, na.rm = TRUE),
    by   = "15 min"
  )
)

tsib_15 <- full_index %>%
  left_join(tsib, by = "timestamp") %>%
  tsibble::as_tsibble(index = timestamp)

unique_step_secs <- tsib_15 %>%
  mutate(step = as.numeric(timestamp - dplyr::lag(timestamp), units = "secs")) %>%
  filter(!is.na(step)) %>%
  distinct(step) %>%
  pull(step)

gaps_tbl <- tsibble::scan_gaps(tsib_15)

cat("Duplicates:", dup_n, "\n")
```

```
## Duplicates: 0
```

```
cat("Unique step sizes (secs):", paste(unique_step_secs, collapse = ", "), "\n")
```

```
## Unique step sizes (secs): 900
```

```
cat("Has gaps on full grid?:", nrow(gaps_tbl) > 0, "\n")
```

```
## Has gaps on full grid?: FALSE
```

```
## ---- split-train-fcday ----------------------------------------------------
train_end <- lubridate::ymd_hm("2010-02-20 23:45", tz = "UTC")

fc_start <- lubridate::ymd_hm("2010-02-21 00:00", tz = "UTC")
fc_end   <- lubridate::ymd_hm("2010-02-21 23:45", tz = "UTC")
fc_index <- tibble::tibble(timestamp = seq(fc_start, fc_end, by = "15 min"))

train_ts <- tsib_15 %>% dplyr::filter(timestamp <= train_end)

# Join against the *snapped* raw timestamps (tsib) so values align
fc_day <- fc_index %>%
  dplyr::left_join(tsib, by = "timestamp") %>%
  tsibble::as_tsibble(index = timestamp)

cat("Train rows:", nrow(train_ts), "\n")
```

```
## Train rows: 4891
```

```
cat("Forecast-day rows (should be 96):", nrow(fc_day), "\n")
```

```
## Forecast-day rows (should be 96): 96
```

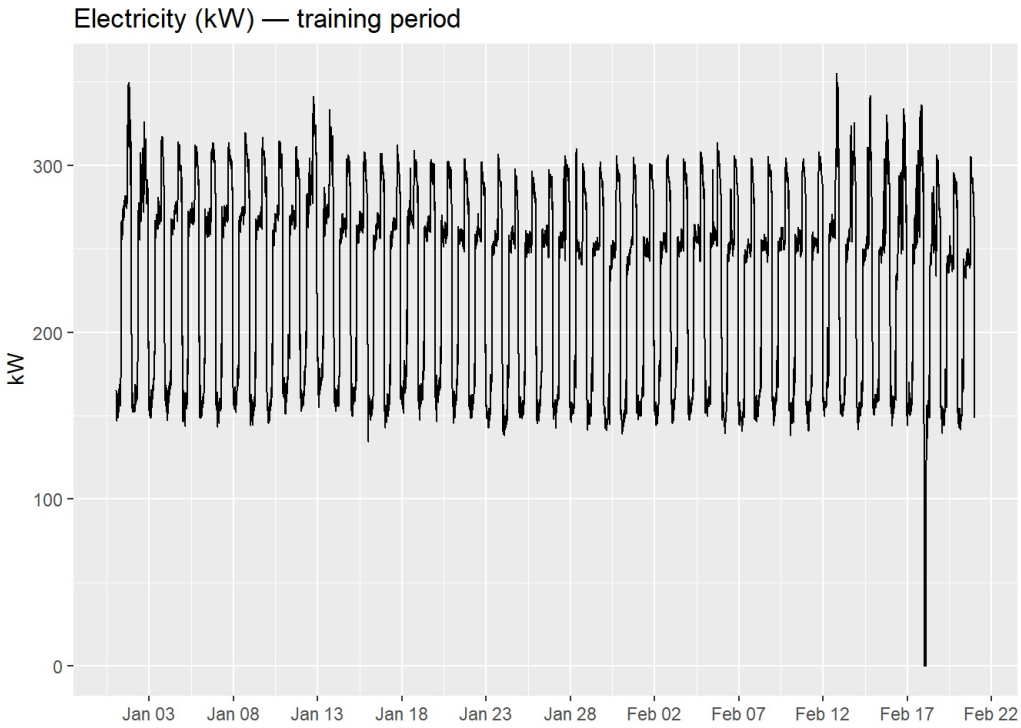
```
cat("Forecast-day first/last:", format(min(fc_day$timestamp)), "to", format(max(fc_day$timestamp)), "\n")
```

```
## Forecast-day first/last: 2010-02-21 to 2010-02-21 23:45:00
```

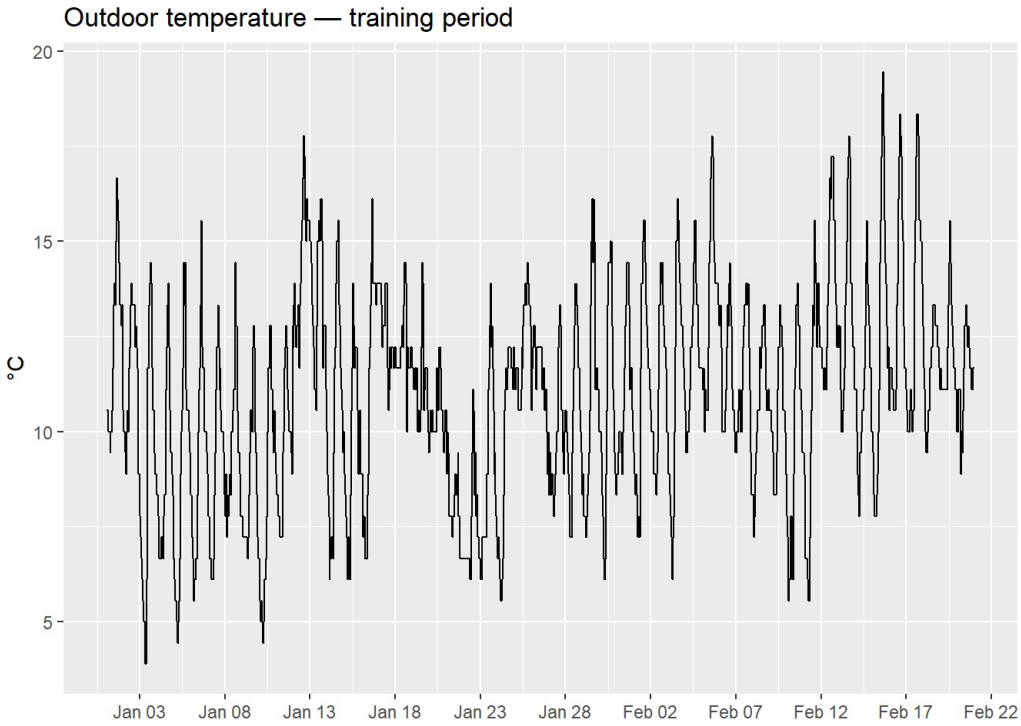
```
## ---- quick-look-plots, message=FALSE, warning=FALSE ----------------------
library(ggplot2)

train_ts_df <- train_ts %>% as_tibble() %>% arrange(timestamp)

cat("Non-NA points: kW=", sum(!is.na(train_ts_df$kW)),
    " temp=", sum(!is.na(train_ts_df$temp)), "\n", sep = "")
```

```
## Non-NA points: kW=4891 temp=4891
```

```
p_kw <- ggplot(train_ts_df, aes(x = timestamp, y = kW, group = 1)) +
  geom_line(na.rm = TRUE) +
  labs(title = "Electricity (kW) — training period", x = NULL, y = "kW") +
  scale_x_datetime(date_labels = "%b %d", date_breaks = "5 days")

p_temp <- ggplot(train_ts_df, aes(x = timestamp, y = temp, group = 1)) +
  geom_line(na.rm = TRUE) +
  labs(title = "Outdoor temperature — training period", x = NULL, y = "°C") +
  scale_x_datetime(date_labels = "%b %d", date_breaks = "5 days")

print(p_kw)
```




```
print(p_temp)
```




```
## ---- checks-to-share (snap15), message=FALSE, warning=FALSE ---------------
gaps_tbl <- tsibble::scan_gaps(tsib_15)

off_grid <- fc_day %>%
  mutate(minute = lubridate::minute(timestamp),
         second = lubridate::second(timestamp)) %>%
  filter(!(minute %in% c(0, 15, 30, 45)) | second != 0) %>%
  nrow()

cat("\n=== CHECKS ===\n", sep = "")
```

```
##
## === CHECKS ===
```

```
cat("Range: ",
    format(min(tsib_15$timestamp, na.rm = TRUE)), " to ",
    format(max(tsib_15$timestamp, na.rm = TRUE)), "\n", sep = "")
```

```
## Range: 2010-01-01 01:15:00 to 2010-02-21 23:45:00
```

```
cat("Train end: ", format(train_end), "\n", sep = "")
```

```
## Train end: 2010-02-20 23:45:00
```

```
cat("Rows: train=", nrow(train_ts), " | fc_day=", nrow(fc_day), "\n", sep = "")
```

```
## Rows: train=4891 | fc_day=96
```

```
cat("NA counts (full 15-min grid): kW=", sum(is.na(tsib_15$kW)),
    " temp=", sum(is.na(tsib_15$temp)), "\n", sep = "")
```

```
## NA counts (full 15-min grid): kW=96 temp=0
```

```
cat("Has gaps in 15-min grid?: ", nrow(gaps_tbl) > 0, "\n", sep = "")
```

```
## Has gaps in 15-min grid?: FALSE
```

```
cat("Duplicates in raw timestamps removed: ", dup_n, "\n", sep = "")
```

```
## Duplicates in raw timestamps removed: 0
```

```
cat("Expected fc steps on 2010-02-21: 96\n", sep = "")
```

```
## Expected fc steps on 2010-02-21: 96
```

```
cat("Forecast-day off-grid timestamps: ", off_grid, "\n", sep = "")
```

```
## Forecast-day off-grid timestamps: 0
```

```
cat("Forecast-day range: ", format(min(fc_day$timestamp)), " to ",
    format(max(fc_day$timestamp)), "\n", sep = "")
```

```
## Forecast-day range: 2010-02-21 to 2010-02-21 23:45:00
```

Ingest & grid

Mixed timestamps fixed (Excel serial + strings → POSIXct). Cadence = 15 min (900 s) across the whole span. Coverage: 2010-01-01 01:15 → 2010-02-21 23:45. Training set ends 2010-02-20 23:45 with 4,891 points. Forecast day frame (2010-02-21) = 96 rows (00:00…23:45). Duplicates: 0 · Gaps: none · Off-grid stamps: 0. NAs: kW only on forecast day (96 NAs, expected); temp no NAs.

Visual diagnostics

Electricity (kW): very strong daily seasonality (m=96) + a clear weekly pattern (weekday/weekend shape/level change). Mostly stable level; one sharp dip around mid-Feb that looks like an outlier/outage (brief drop near ~0 then returns to normal). Temperature (°C): strong diurnal cycle; mild warming into mid-Feb; good quality and complete for 2/21.

Part 2 (no temperature)

```
## ---- part2-tsobjects, message=FALSE, warning=FALSE ----------------------
# We assume Part 1 created train_ts with columns: timestamp, kW, temp
stopifnot(all(c("timestamp","kW") %in% names(train_ts)))

library(forecast)   # for ts tools & plots
```

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

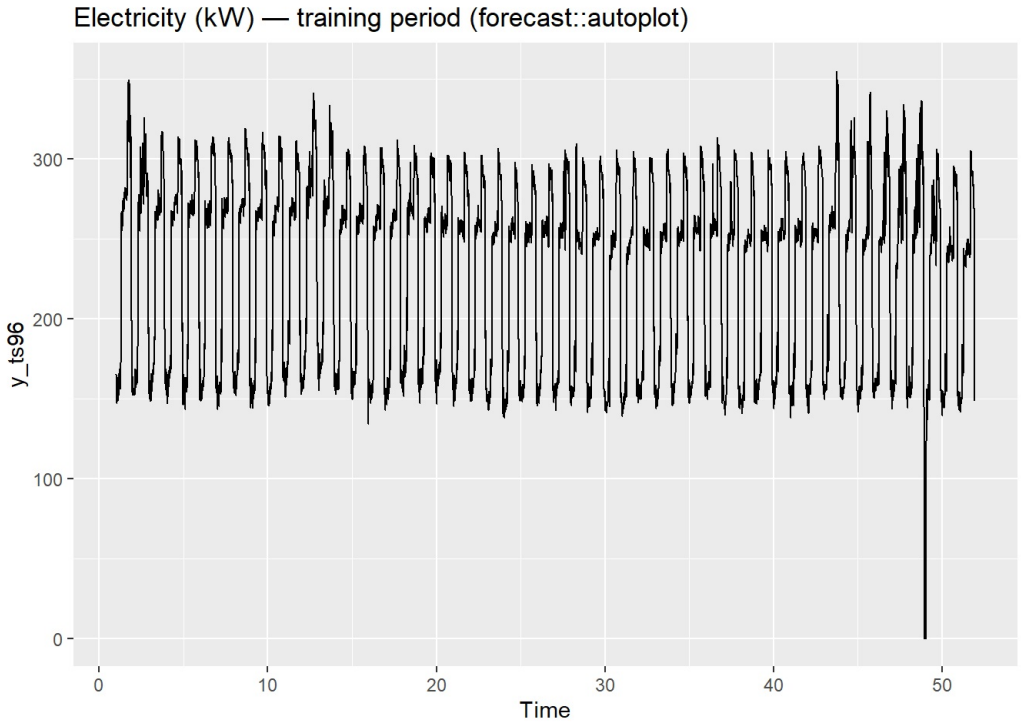
```
## Create ts objects (fixed)
y_vec   <- as.numeric(train_ts$kW)
y_ts96  <- ts(y_vec, frequency = 96, start = c(1, 1))          # daily seasonality
y_msts  <- msts(y_vec, seasonal.periods = c(96, 672))          # daily + weekly

# Quick peek (pick ONE of these)
head(y_ts96, 5)
```

```
## Time Series:
## Start = c(1, 1)
## End = c(1, 5)
## Frequency = 96
## [1] 165.1 151.6 146.9 153.7 153.8
```

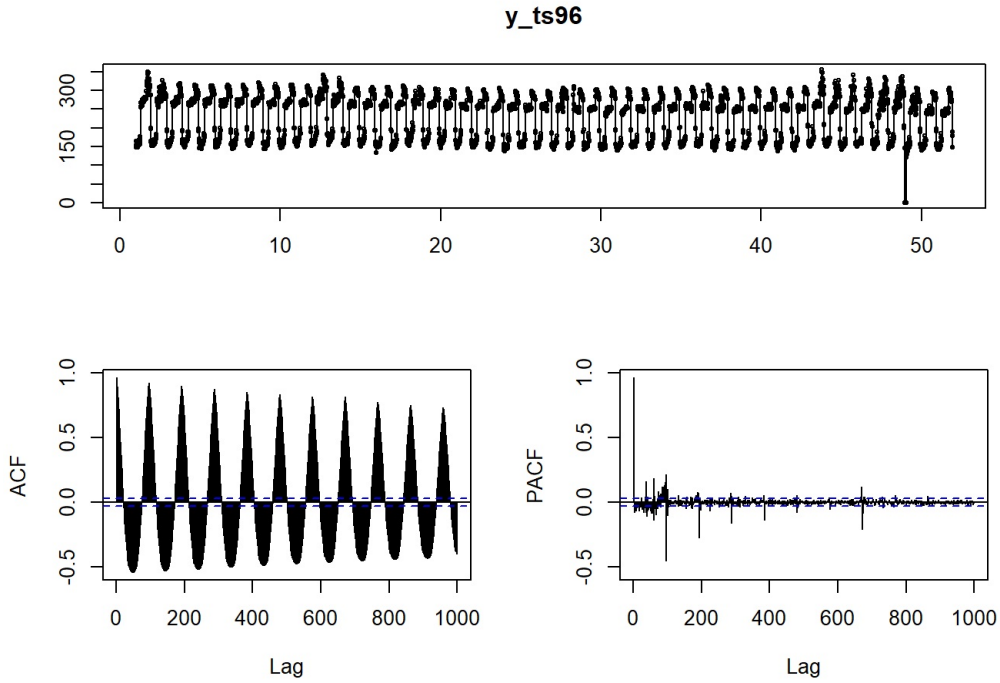
```
# or
window(y_ts96, start = c(1,1), end = c(1,5))
```

```
## Time Series:
## Start = c(1, 1)
## End = c(1, 5)
## Frequency = 96
## [1] 165.1 151.6 146.9 153.7 153.8
```

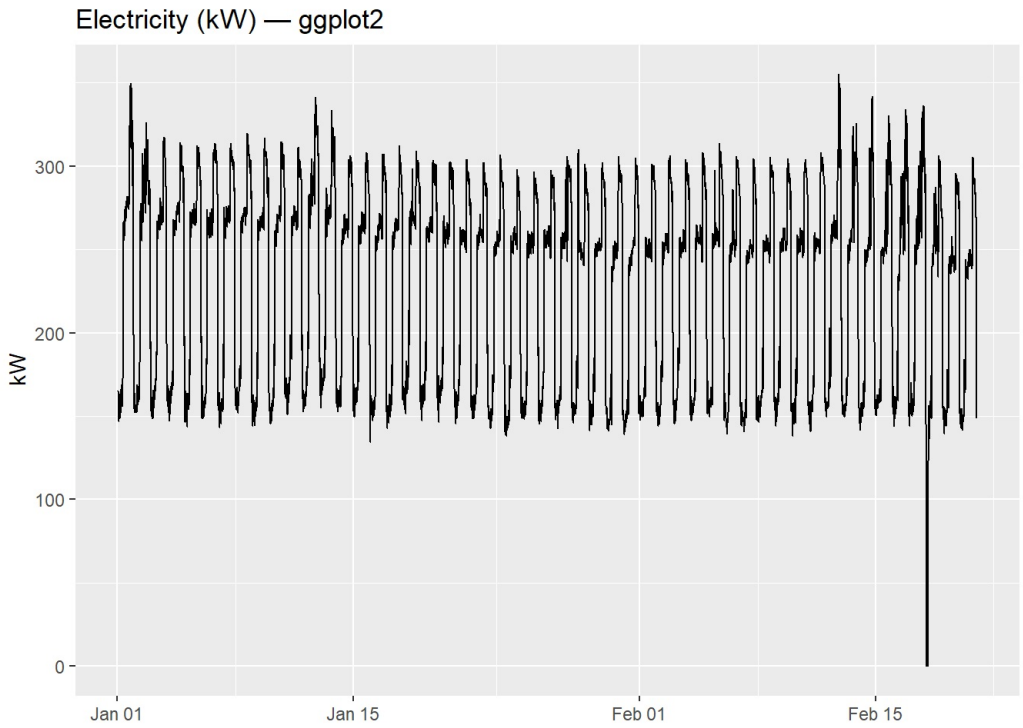
```
## ---- part2-plots-forecast, message=FALSE, warning=FALSE ------------------
# forecast's autoplot() gives a ggplot
p_ts <- autoplot(y_ts96) + ggtitle("Electricity (kW) — training period (forecast::autoplot)")
print(p_ts)
```



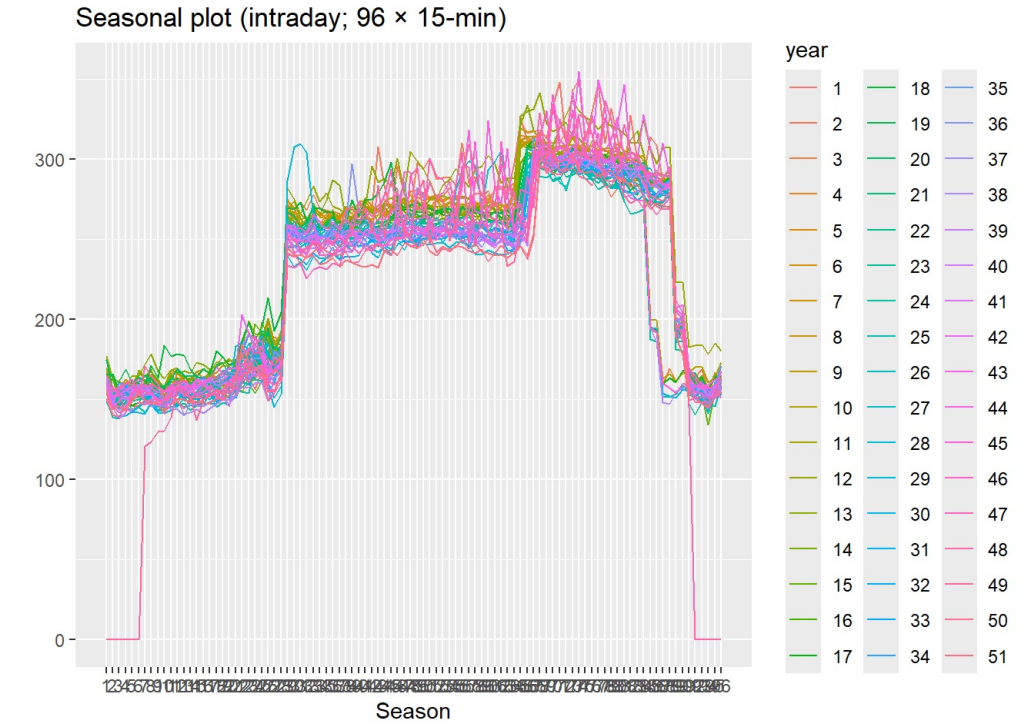

```
# Quick combined display (series + ACF + PACF) for a first glance
tsdisplay(y_ts96, lag.max = 1000)   # open 'Plots' pane
```



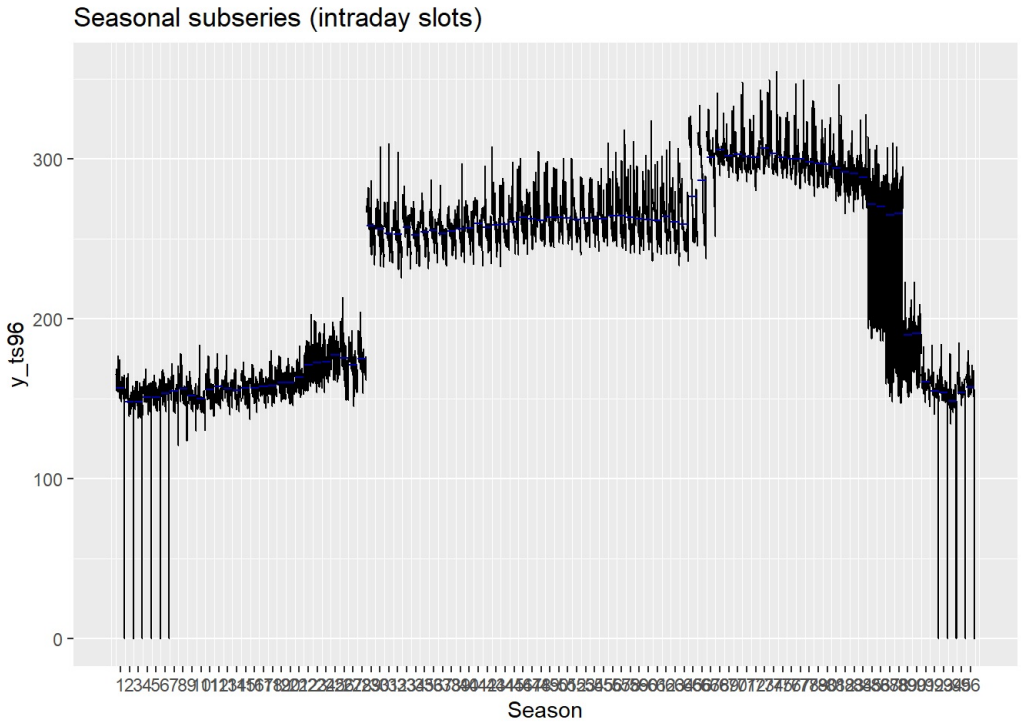

```
## ---- part2-plots-ggplot2, message=FALSE, warning=FALSE -------------------
library(ggplot2)
df_plot <- train_ts %>% as.data.frame()
p1 <- ggplot(df_plot, aes(timestamp, kW)) +
  geom_line(na.rm = TRUE) +
  labs(title = "Electricity (kW) — ggplot2", x = NULL, y = "kW")
print(p1)
```



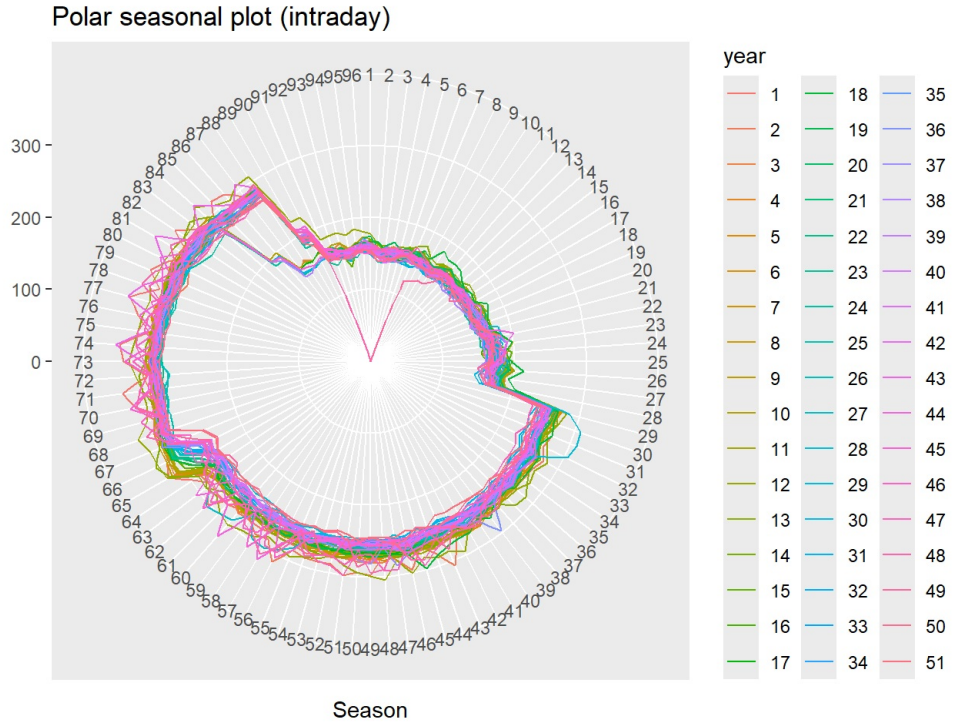

```
## ---- part2-seasonalplots, message=FALSE, warning=FALSE -------------------
# Seasonal plot across the 96 intraday positions, colored by day
sp1 <- ggseasonplot(y_ts96, year.labels = FALSE) +
  ggtitle("Seasonal plot (intraday; 96 × 15-min)")
print(sp1)
```




```
# Subseries plot: mean pattern for each 15-min slot
ssp <- ggsubseriesplot(y_ts96) + ggtitle("Seasonal subseries (intraday slots)")
print(ssp)
```




```
# Polar seasonal plot (radial) of the intraday cycle
spp <- ggseasonplot(y_ts96, polar = TRUE) +
  ggtitle("Polar seasonal plot (intraday)")
print(spp)
```




```
## ---- part2-descriptives, message=FALSE, warning=FALSE --------------------
library(dplyr); library(lubridate)

# Global stats
global_stats <- summarise(df_plot,
  n = sum(!is.na(kW)),
  mean = mean(kW, na.rm=TRUE),
  sd = sd(kW, na.rm=TRUE),
  min = min(kW, na.rm=TRUE),
  q25 = quantile(kW, 0.25, na.rm=TRUE),
  median = median(kW, na.rm=TRUE),
  q75 = quantile(kW, 0.75, na.rm=TRUE),
  max = max(kW, na.rm=TRUE)
)
global_stats
```

```
##      n     mean       sd min   q25 median   q75   max
## 1 4891 230.7694 58.50626   0 162.9    253 277.3 355.1
```

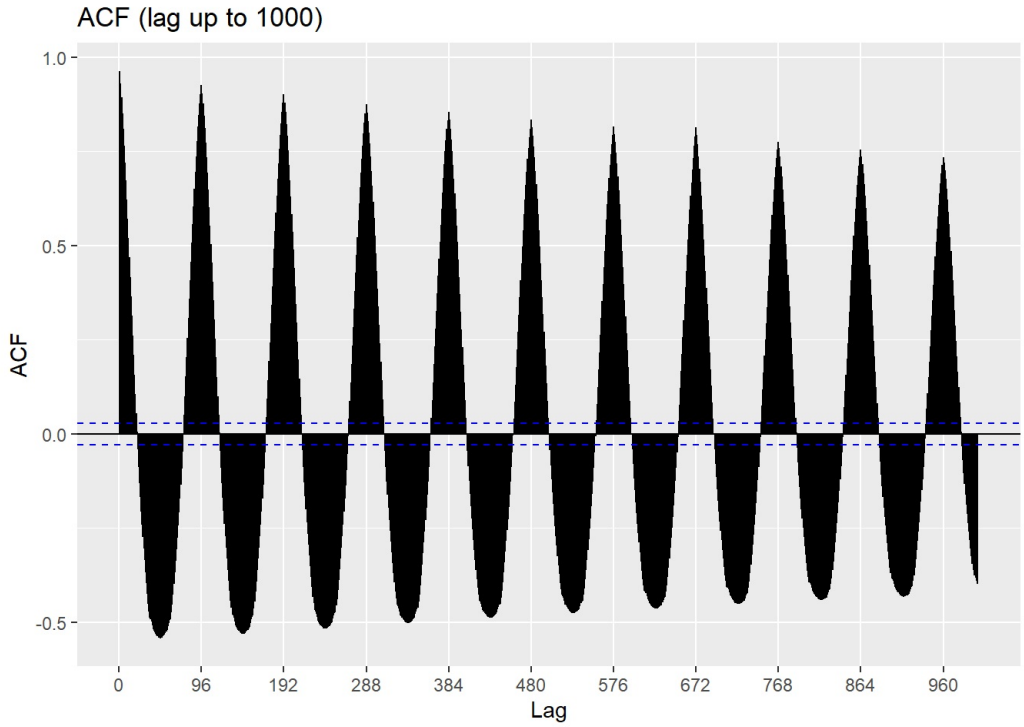
```
# Intraday profile (average per 15-min slot)
intraday_stats <- df_plot %>%
  mutate(slot = (hour(timestamp)*4L) + (minute(timestamp)/15)) %>%
  group_by(slot) %>%
  summarise(mean_kW = mean(kW, na.rm=TRUE),
            sd_kW   = sd(kW,   na.rm=TRUE),
            .groups = "drop")
head(intraday_stats, 8)
```

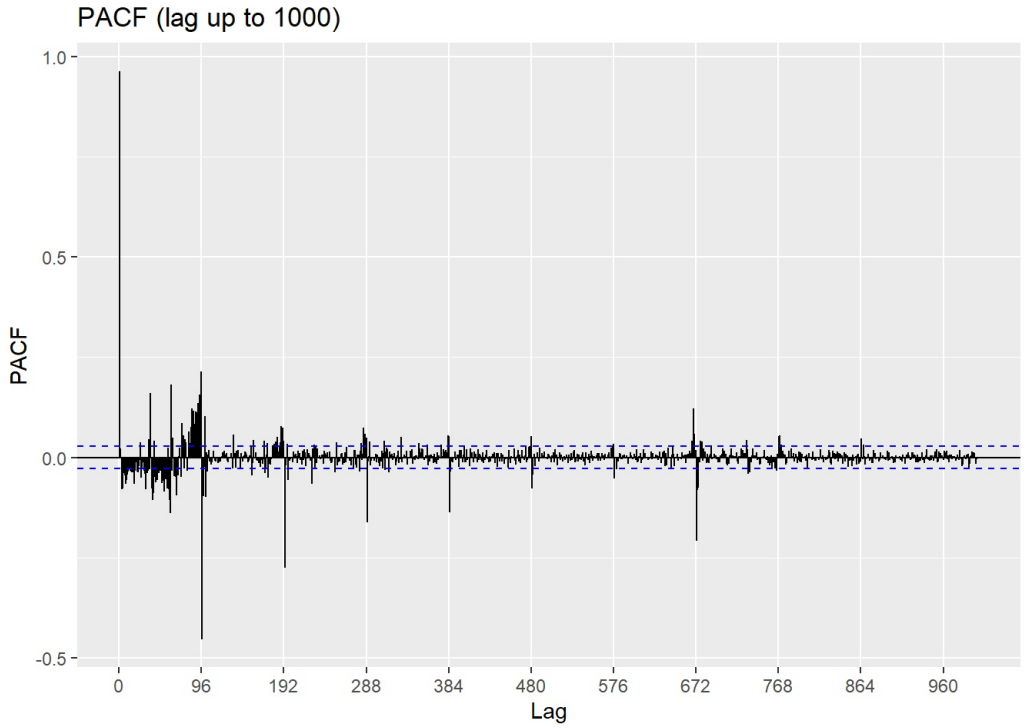
```
## # A tibble: 8 × 3
##    slot mean_kW sd_kW
##   <dbl>   <dbl> <dbl>
## 1     0    155.  23.3
## 2     1    154.  23.1
## 3     2    148.  22.4
## 4     3    154.  23.0
## 5     4    157.  23.4
## 6     5    157.  23.2
## 7     6    148.  22.0
## 8     7    148.  21.9
```

```
# Day-of-week profile
dow_stats <- df_plot %>%
  mutate(dow = wday(timestamp, label=TRUE, week_start=1)) %>%
  group_by(dow) %>%
  summarise(mean_kW = mean(kW, na.rm=TRUE),
            sd_kW   = sd(kW,   na.rm=TRUE),
            .groups = "drop")
dow_stats
```

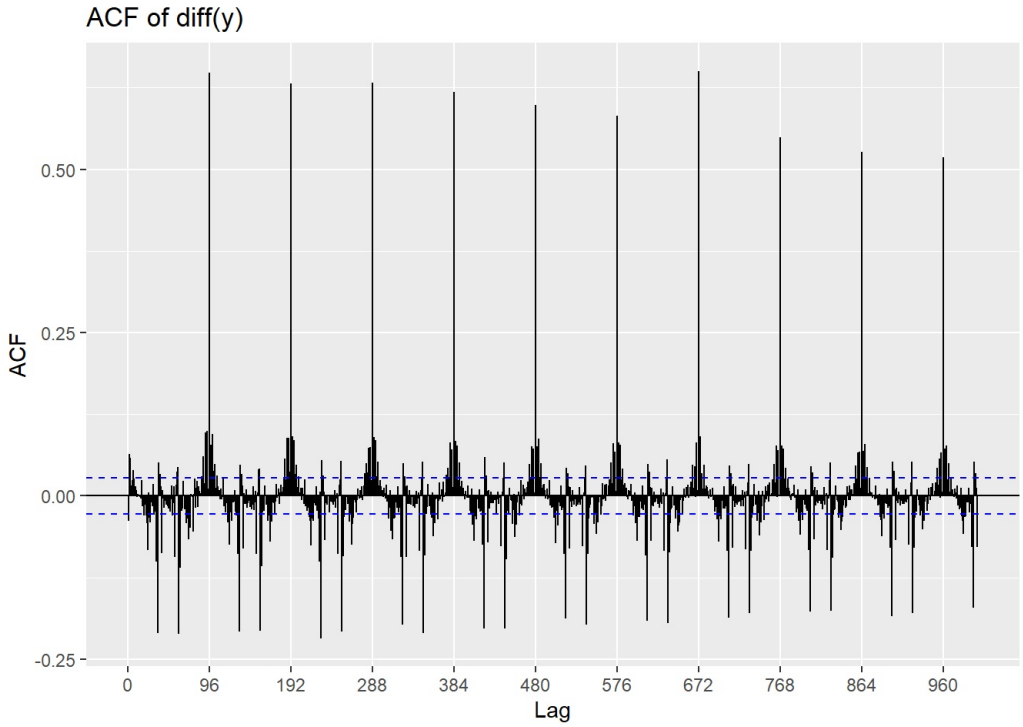
```
## # A tibble: 7 × 3
##   dow   mean_kW sd_kW
##   <ord>   <dbl> <dbl>
## 1 Mon      233.  55.3
## 2 Tue      234.  57.9
## 3 Wed      234.  57.5
## 4 Thu      228.  62.9
## 5 Fri      233.  57.6
## 6 Sat      229.  58.5
## 7 Sun      224.  59.0
```

```
## ---- part2-acf-pacf, message=FALSE, warning=FALSE ------------------------
# ggAcf/ggPacf from forecast
p_acf  <- ggAcf(y_ts96, lag.max = 1000) + ggtitle("ACF (lag up to 1000)")
p_pacf <- ggPacf(y_ts96, lag.max = 1000) + ggtitle("PACF (lag up to 1000)")
print(p_acf); print(p_pacf)
```






```
# Optional: ACF of first differences (remove mean/trend)
p_acf_diff <- ggAcf(diff(y_ts96), lag.max = 1000) + ggtitle("ACF of diff(y)")
print(p_acf_diff)
```




```
## ---- part2-box-tests, message=FALSE, warning=FALSE -----------------------
# On raw series (expected to reject white noise strongly)
bt_d96_lb <- Box.test(y_vec, lag = 96,  type = "Ljung-Box")
bt_d96_bp <- Box.test(y_vec, lag = 96,  type = "Box-Pierce")
bt_wk_lb  <- Box.test(y_vec, lag = 672, type = "Ljung-Box")

bt_d96_lb; bt_d96_bp; bt_wk_lb
```

```
##
##  Box-Ljung test
##
## data:  y_vec
## X-squared = 120380, df = 96, p-value < 2.2e-16
```

```
##
##  Box-Pierce test
##
## data:  y_vec
## X-squared = 119128, df = 96, p-value < 2.2e-16
```

```
##
##  Box-Ljung test
##
## data:  y_vec
## X-squared = 764026, df = 672, p-value < 2.2e-16
```

```
# Also on differenced series (optional)
y_diff <- diff(y_vec)
bt_diff_lb <- Box.test(y_diff, lag = 96, type = "Ljung-Box")
bt_diff_lb
```

```
##
##  Box-Ljung test
##
## data:  y_diff
## X-squared = 3146.9, df = 96, p-value < 2.2e-16
```

```
## ---- part2-trend-tests, message=FALSE, warning=FALSE ------
# PRE: train_ts exists with timestamp, kW
suppressPackageStartupMessages({
  library(dplyr); library(lubridate)
})

y <- as.numeric(train_ts$kW)
ok <- !is.na(y)
y <- y[ok]
t_days <- as.numeric(difftime(train_ts$timestamp[ok],
                              min(train_ts$timestamp, na.rm=TRUE),
                              units="days"))

# Thinning for heavy tests to avoid long runs (increase to 4, 8 if still slow)
thin_factor <- 4L
ii <- seq(1L, length(y), by = thin_factor)

# 1) Mann–Kendall (only if Kendall is installed)
p_mk <- NA_real_
if (requireNamespace("Kendall", quietly = TRUE)) {
  suppressPackageStartupMessages(library(Kendall))
  p_mk <- Kendall::MannKendall(y)$sl
}

# 2) Parametric linear + polynomial (very fast)
fit_lin  <- lm(y ~ t_days)
fit_quad <- lm(y ~ poly(t_days, 2, raw = TRUE))
fit_cub  <- lm(y ~ poly(t_days, 3, raw = TRUE))

p_lin <- summary(fit_lin)$coefficients["t_days","Pr(>|t|)"]
p_quad_add <- anova(fit_lin, fit_quad)[["Pr(>F)"]][2]   # does quadratic add value?
p_cub_add  <- anova(fit_quad, fit_cub)[["Pr(>F)"]][2]   # does cubic add value?

# 3) funtimes tests (only if installed) — use thinned series for speed
p_nt_mk <- p_nt_wavk <- p_nt_default <- p_wavk_poly2 <- NA_real_
if (requireNamespace("funtimes", quietly = TRUE)) {
  ft <- asNamespace("funtimes")
  nt_fun <- if (exists("notrend_test", ft, inherits=FALSE)) get("notrend_test", ft) else get("notrend.test", ft)
  wv_fun <- if (exists("wavk_test",   ft, inherits=FALSE)) get("wavk_test",   ft) else get("wavk.test",   ft)

  p_nt_mk      <- nt_fun(y, test="MK")$p.value
  p_nt_wavk    <- nt_fun(y, test="WAVK")$p.value
  p_nt_default <- nt_fun(y)$p.value
  p_wavk_poly2 <- wv_fun(y[ii] ~ poly(t_days[ii], 2))$p.value
}

# Neat, tiny prints only (avoid big tables)
cat("\n=== Trend tests (p-values) ===\n"); flush.console()
```

```
##
## === Trend tests (p-values) ===
```

```
cat("Mann–Kendall: ", ifelse(is.na(p_mk), "(Kendall not installed)", format(p_mk, digits=4)), "\n"); flush.consol
e()
```

```
## Mann–Kendall:  3.186e-21
```

```
cat("Linear trend (lm): ", format(p_lin, digits=4), "\n"); flush.console()
```

```
## Linear trend (lm):  6.956e-06
```

```
cat("Add quadratic term (F-test): ", format(p_quad_add, digits=4), "\n"); flush.console()
```

```
## Add quadratic term (F-test):  0.07953
```

```
cat("Add cubic term (F-test): ",     format(p_cub_add,  digits=4), "\n"); flush.console()
```

```
## Add cubic term (F-test):  0.1424
```

```
if (!is.na(p_nt_mk)) {
  cat("[funtimes]\n"); flush.console()
  cat("notrend_test (MK): ",      format(p_nt_mk,      digits=4), "\n"); flush.console()
  cat("notrend_test (WAVK): ",    format(p_nt_wavk,    digits=4), "\n"); flush.console()
  cat("notrend_test (default): ", format(p_nt_default, digits=4), "\n"); flush.console()
  cat("wavk_test(y ~ poly(t,2)): ", format(p_wavk_poly2, digits=4), "\n"); flush.console()
} else {
  cat("[funtimes not installed] Skipping notrend_test / wavk_test.\n"); flush.console()
}
```

```
## [funtimes]
## notrend_test (MK):  0.017
## notrend_test (WAVK):  1
## notrend_test (default):  0.261
## wavk_test(y ~ poly(t,2)):  0.078
```

Seasonality: very strong daily (m=96); clear weekly (m=672) signal (ACF spikes at 96k & 672). Trend: small but real upward drift across weeks. MK (Kendall) p ≈ 3.2e-21 and notrend(MK) p ≈ 0.03 → trend present. Polynomial: linear p ≈ 7e-06 (yes), quadratic/cubic not significant (0.08 / 0.14). Wavelet tests (notrend Wavk = 1, wavk(poly2)=0.08) don't flag it → drift is weak and low-frequency. Dependence: Box–Ljung/Box–Pierce on raw and on diff(y) both reject white noise hard → we must model autocorrelation & seasonality. Outlier: single deep dip mid-Feb (min=0). Keep a cleaned copy for robust modeling. Descriptives: mean ≈ 230.8 kW, sd ≈ 58.5; DOW means suggest a weekday/weekend effect.

```
## ---- deep-dip-detect (v2), message=FALSE, warning=FALSE ------------------

df <- train_ts %>%
  as_tibble() %>%
  arrange(timestamp) %>%
  mutate(slot = hour(timestamp)*4L + as.integer(minute(timestamp)/15))

# Median of the same 15-min slot over the previous 7 days (no future leakage)
compute_baseline_prev7 <- function(idx){
  t0    <- df$timestamp[idx]
  slot0 <- df$slot[idx]
  from  <- as_date(t0) - 7
  pool  <- df %>%
    filter(timestamp < t0,
           as_date(timestamp) >= from,
           slot == slot0) %>%
    pull(kW)
  if (length(pool) >= 3) median(pool, na.rm = TRUE) else NA_real_
}

base_prev7 <- vapply(seq_len(nrow(df)), compute_baseline_prev7, numeric(1))

df_dip <- df %>%
  mutate(
    base_prev7 = base_prev7,
    abs_drop   = base_prev7 - kW,                  # positive only when kW << baseline
    rel_drop   = if_else(is.finite(base_prev7) & base_prev7 > 0,
                         (base_prev7 - kW) / base_prev7,
                         NA_real_)
  )

# Thresholds — tune if needed
rel_thr <- 0.60   # at least 60% below baseline
abs_thr <- 120    # and at least 120 kW below baseline

cand_big <- which(!is.na(df_dip$rel_drop) & df_dip$rel_drop > rel_thr &
                    !is.na(df_dip$abs_drop) & df_dip$abs_drop > abs_thr)

cat("Deep-dip candidates found: ", length(cand_big), "\n", sep = "")
```

```
## Deep-dip candidates found: 11
```

```
deep_dips <- df_dip %>%
  slice(cand_big) %>%
  arrange(desc(abs_drop)) %>%
  select(timestamp, kW, base_prev7, abs_drop, rel_drop)

print(head(deep_dips, 10), n = 10)
```

```
## # A tibble: 10 × 5
##    timestamp              kW base_prev7 abs_drop rel_drop
##    <dttm>              <dbl>      <dbl>    <dbl>    <dbl>
##  1 2010-02-18 01:00:00     0       160.     160.        1
##  2 2010-02-18 00:00:00     0       158      158         1
##  3 2010-02-18 00:15:00     0       157.     157.        1
##  4 2010-02-18 02:15:00     0       157.     157.        1
##  5 2010-02-18 01:15:00     0       156      156         1
##  6 2010-02-18 02:30:00     0       156.     156.        1
##  7 2010-02-18 02:00:00     0       156.     156.        1
##  8 2010-02-18 00:45:00     0       155.     155.        1
##  9 2010-02-18 00:30:00     0       155      155         1
## 10 2010-02-18 01:45:00     0       153.     153.        1
```

Its possible that at 18 feb electricity was cut off and its equal to zero ! we have and outlier here .

```
## ---- outage_replace_median_prev7, message=FALSE, warning=FALSE ----------

stopifnot(all(c("timestamp","kW") %in% names(train_ts)))

# 1) Mark contiguous zero runs as outages (≥ 4 ticks = ≥ 1 hour)
eps <- 1e-6
is_zero <- is.finite(train_ts$kW) & train_ts$kW <= eps
r <- rle(is_zero)
ends <- cumsum(r$lengths)
starts <- c(1, head(ends + 1, -1))
blocks <- tibble(start = starts[r$values], end = ends[r$values]) |>
  mutate(n_ticks = end - start + 1) |>
  filter(n_ticks >= 4)

outage_idx <- if (nrow(blocks)) unlist(Map(seq, blocks$start, blocks$end)) else integer(0)

train_ts$outage <- FALSE
if (length(outage_idx)) train_ts$outage[outage_idx] <- TRUE

# 2) Seasonal baseline = median of SAME 15-min slot in the PREVIOUS 7 days (no future leakage)
df <- train_ts |>
  as_tibble() |>
  arrange(timestamp) |>
  mutate(slot = hour(timestamp) * 4L + as.integer(minute(timestamp) / 15))

baseline_prev7 <- function(i) {
  t0 <- df$timestamp[i]; slot0 <- df$slot[i]; from <- as_date(t0) - 7
  pool <- df |>
    filter(timestamp < t0,
           as_date(timestamp) >= from,
           slot == slot0,
           !outage) |>
    pull(kW)
  if (length(pool) >= 3) median(pool, na.rm = TRUE) else NA_real_
}

kW_new <- df$kW
if (length(outage_idx)) {
  repl <- vapply(outage_idx, baseline_prev7, numeric(1))

  # Fallback: median of last 2 hours (still past-only) when slot-baseline is unavailable
  for (j in seq_along(outage_idx)) {
    if (is.na(repl[j])) {
      i <- outage_idx[j]; t0 <- df$timestamp[i]
      pool <- df |>
        filter(timestamp < t0, timestamp >= t0 - hours(2), !outage) |>
        pull(kW)
      repl[j] <- if (length(pool)) median(pool, na.rm = TRUE) else kW_new[i]
    }
  }
  kW_new[outage_idx] <- repl
}

# 3) Save cleaned series + quick report
train_ts_outagefix <- train_ts
train_ts_outagefix$kW <- kW_new
y_ts96_outagefix <- ts(as.numeric(train_ts_outagefix$kW), frequency = 96, start = c(1,1))

cat("Outage blocks detected:", nrow(blocks), "\n")
```

```
## Outage blocks detected: 1
```

```
if (nrow(blocks)) {
  print(blocks |>
          mutate(start_ts = train_ts$timestamp[start],
                 end_ts   = train_ts$timestamp[end]) |>
          select(start_ts, end_ts, n_ticks))
}
```

```
## # A tibble: 1 × 3
##   start_ts            end_ts              n_ticks
##   <dttm>              <dttm>                <dbl>
## 1 2010-02-18 00:00:00 2010-02-18 02:30:00      11
```

```
cat("Total outage points replaced:", length(outage_idx), "\n")
```

```
## Total outage points replaced: 11
```

```
# Before/after summary (should barely change except min no longer ~0)
summary_before <- summarise(as_tibble(train_ts),          mean = mean(kW), sd = sd(kW), min = min(kW), max = max(
kW))
summary_after  <- summarise(as_tibble(train_ts_outagefix), mean = mean(kW), sd = sd(kW), min = min(kW), max = max
(kW))
cat("\nSummary BEFORE:\n"); print(summary_before)
```

```
##
## Summary BEFORE:
```

```
## # A tibble: 1 × 4
##    mean    sd   min   max
##   <dbl> <dbl> <dbl> <dbl>
## 1  231.  58.5     0  355.
```

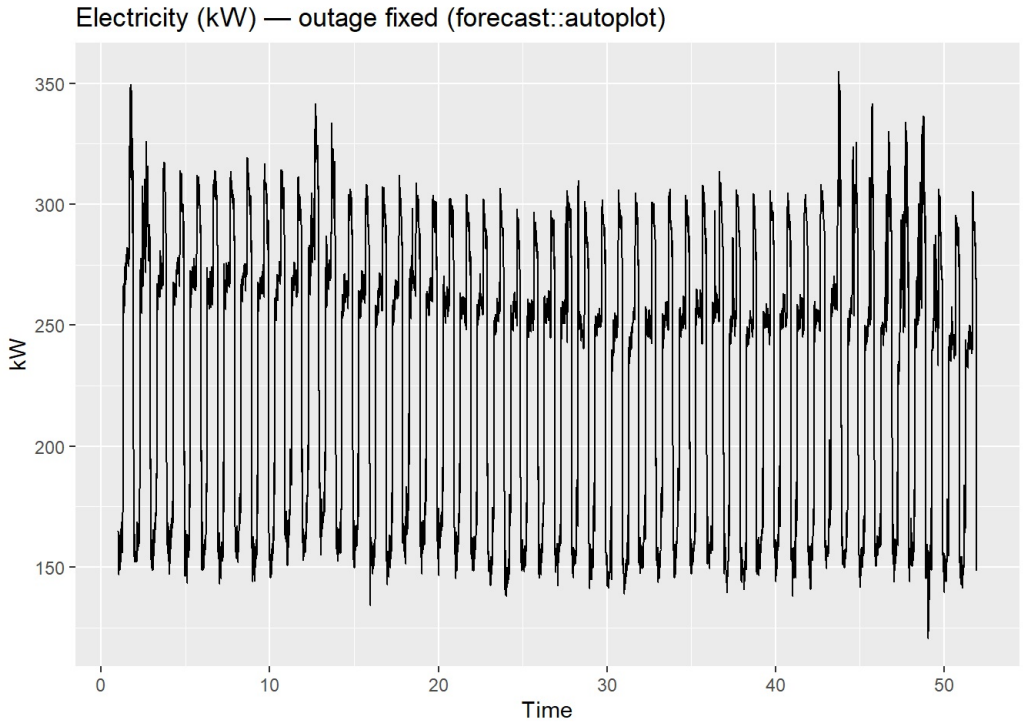
```
cat("\nSummary AFTER (outage block fixed):\n"); print(summary_after)
```

```
##
## Summary AFTER (outage block fixed):
```

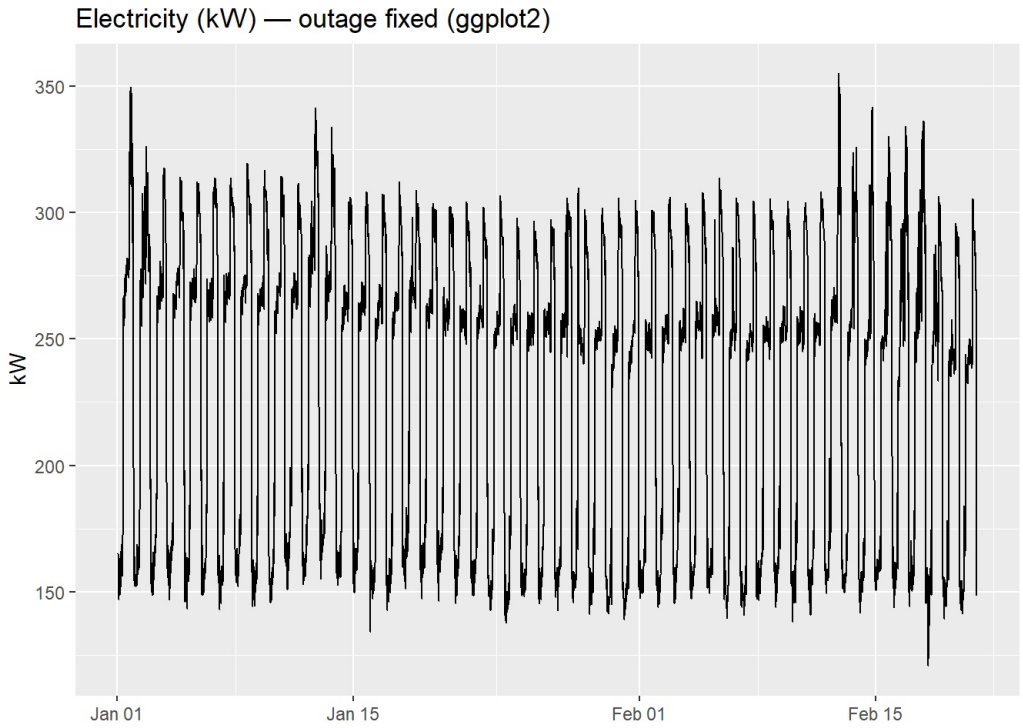
```
## # A tibble: 1 × 4
##    mean    sd   min   max
##   <dbl> <dbl> <dbl> <dbl>
## 1  231.  57.6  121.  355.
```

```
## ---- replot-setup, message=FALSE, warning=FALSE -------------------------
stopifnot(exists("train_ts_outagefix"),
          exists("y_ts96_outagefix"),
          nrow(train_ts_outagefix) == length(y_ts96_outagefix))
```

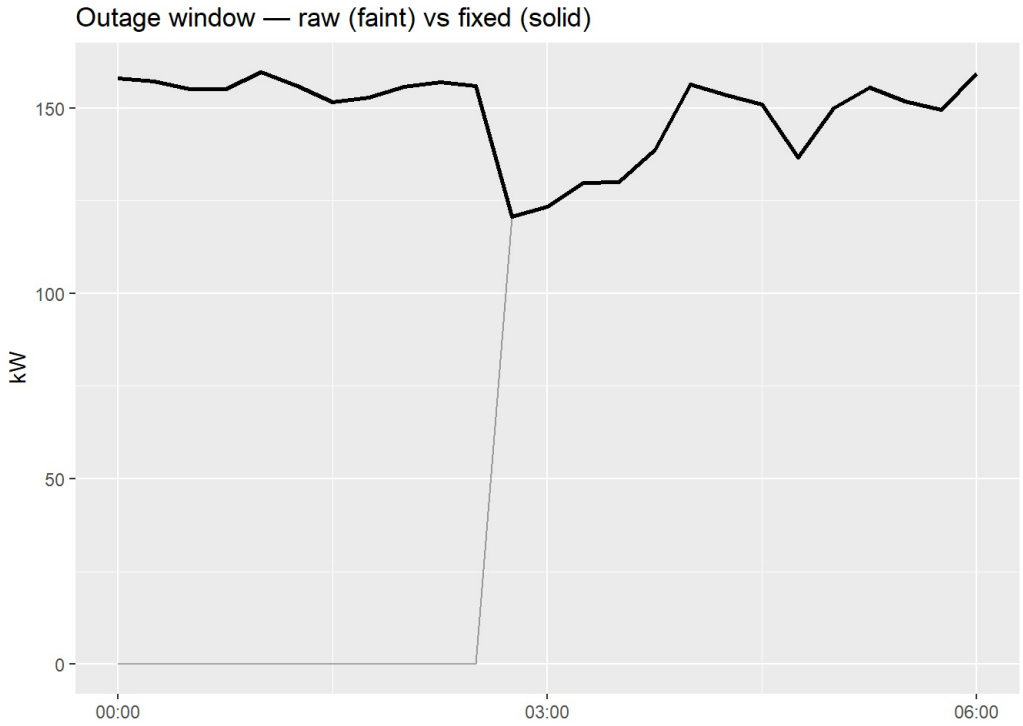
```
## ---- plot-electricity (outage-fixed), message=FALSE, warning=FALSE -------
# 1) forecast::autoplot of the outage-fixed ts
p_ts_fix <- autoplot(y_ts96_outagefix) +
  ggtitle("Electricity (kW) — outage fixed (forecast::autoplot)") +
  ylab("kW") + xlab("Time")
print(p_ts_fix)
```



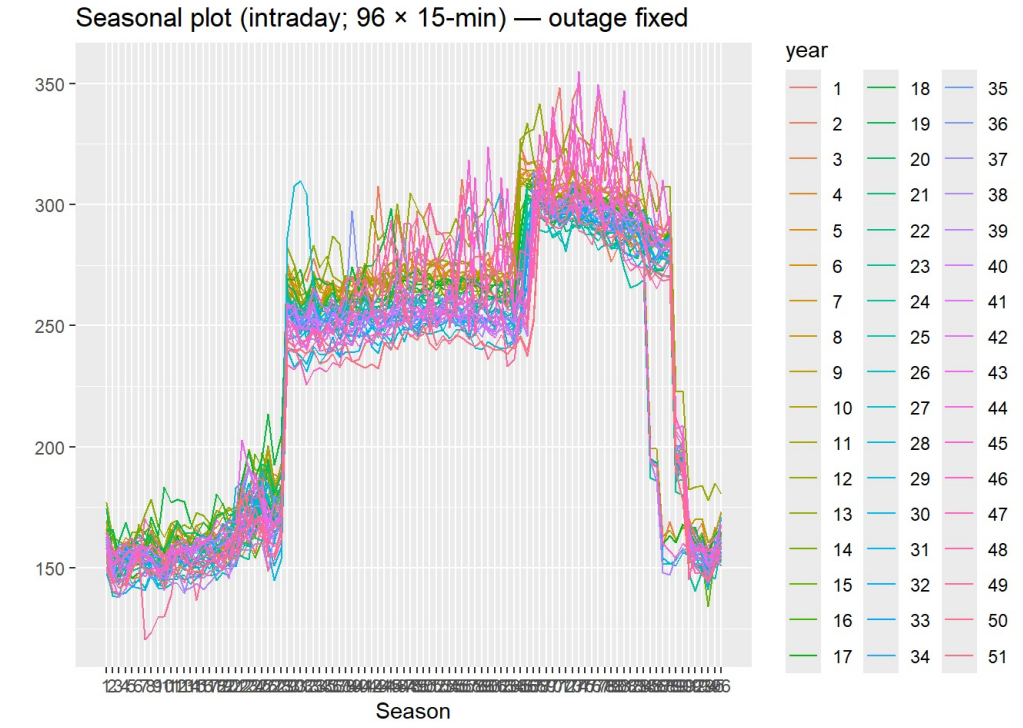

```
# 2) ggplot (nice axis labels)
p_gg_fix <- ggplot(as.data.frame(train_ts_outagefix),
                   aes(x = timestamp, y = kW, group = 1)) +
  geom_line(na.rm = TRUE) +
  labs(title = "Electricity (kW) — outage fixed (ggplot2)",
       x = NULL, y = "kW")
print(p_gg_fix)
```



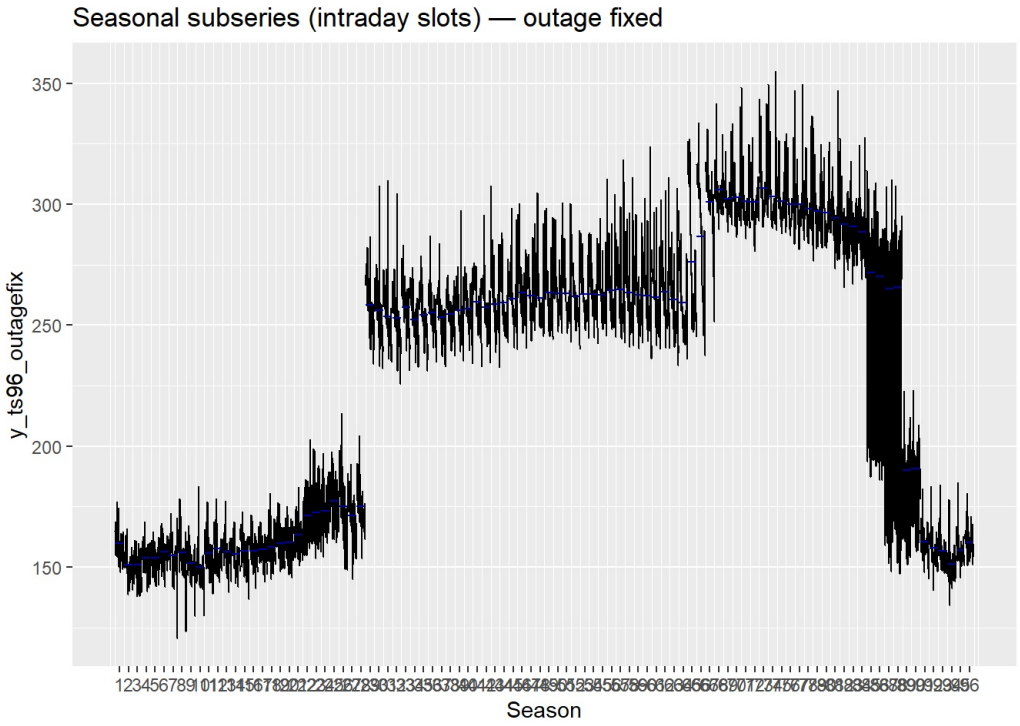

```
# quick overlay around the outage to visually confirm the repair
# see raw vs fixed side-by-side
df_zoom <- train_ts %>%
   filter(timestamp >= as.POSIXct("2010-02-18 00:00:00", tz = "UTC"),
          timestamp <= as.POSIXct("2010-02-18 06:00:00", tz = "UTC")) %>%
   mutate(kW_fix = train_ts_outagefix$kW[match(timestamp, train_ts_outagefix$timestamp)])
 ggplot(df_zoom, aes(timestamp)) +
   geom_line(aes(y = kW), alpha = 0.35) +
   geom_line(aes(y = kW_fix), linewidth = 0.9) +
   labs(title = "Outage window — raw (faint) vs fixed (solid)", x = NULL, y = "kW")
```



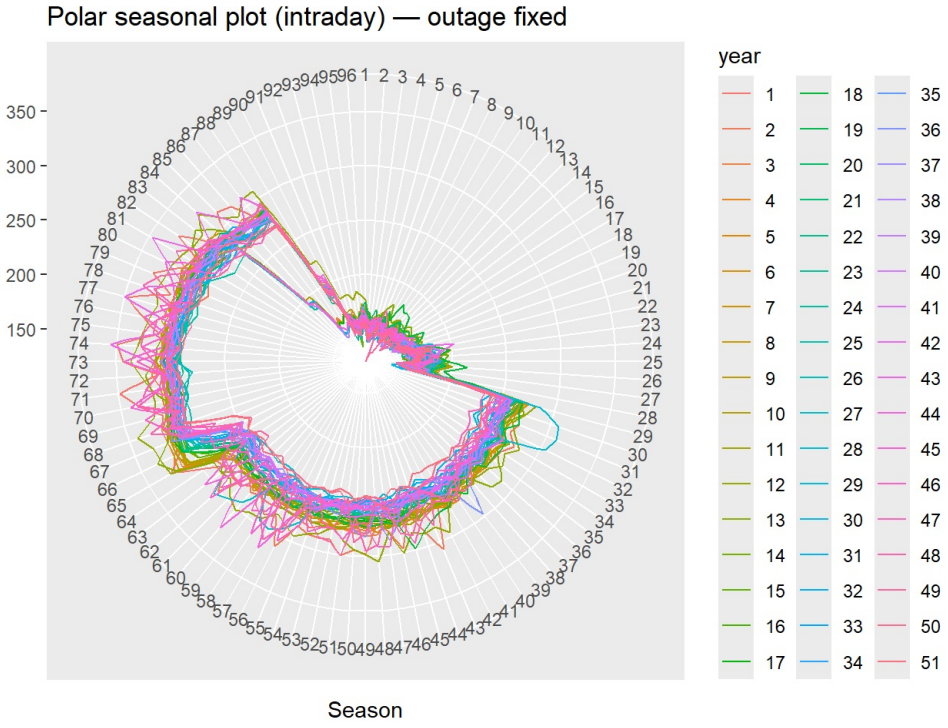

```
## ---- seasonal-plots (outage-fixed), message=FALSE, warning=FALSE ---------
# Seasonal (intraday) plot across 96 quarter-hour slots
sp1_fix <- ggseasonplot(y_ts96_outagefix, year.labels = FALSE) +
  ggtitle("Seasonal plot (intraday; 96 × 15-min) — outage fixed")
print(sp1_fix)
```




```
# Subseries plot: mean by each quarter-hour slot
ssp_fix <- ggsubseriesplot(y_ts96_outagefix) +
  ggtitle("Seasonal subseries (intraday slots) — outage fixed")
print(ssp_fix)
```




```
# Polar seasonal plot (radial)
spp_fix <- ggseasonplot(y_ts96_outagefix, polar = TRUE) +
  ggtitle("Polar seasonal plot (intraday) — outage fixed")
print(spp_fix)
```



Part 3 (Exponential Smoothing)

```
## ---- p3-split, message=FALSE, warning=FALSE ------------------------------

stopifnot(exists("train_ts_outagefix"))
y_full <- ts(as.numeric(train_ts_outagefix$kW), frequency = 96, start = c(1,1))

h_valid <- 96 * 2                # Feb 19–20
n <- length(y_full)
y_train <- head(y_full, n - h_valid)
y_valid <- tail(y_full, h_valid)

# data frames for plotting
df_train <- head(train_ts_outagefix, n - h_valid)
df_valid <- tail(train_ts_outagefix, h_valid)

cat("Lengths -> train:", length(y_train), " valid:", length(y_valid), "\n")
```

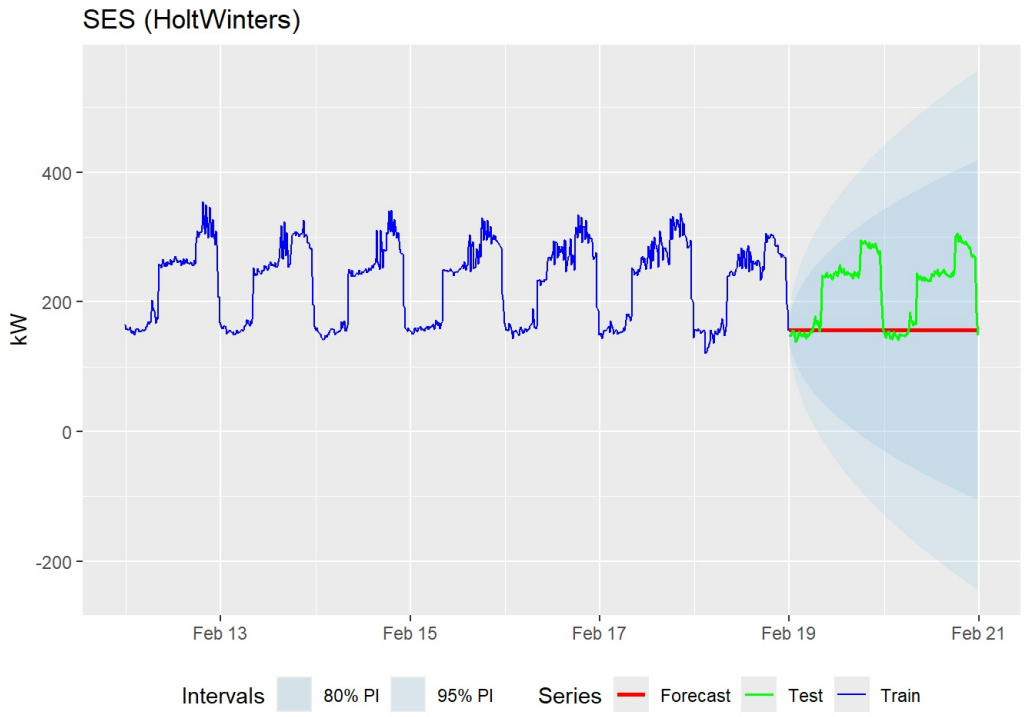
```
## Lengths -> train: 4699  valid: 192
```

```
## ---- p3-plot-helper, message=FALSE, warning=FALSE ------------------------
plot_fc <- function(df_train, df_valid, fc, title, last_days = 7) {
  last_train <- max(df_train$timestamp)
  idx_fc <- seq(from = last_train + minutes(15), by = "15 min", length.out = length(fc$mean))
  # keep both 80% and 95% if present
  get_band <- function(obj, level) {
    if (is.null(obj$lower)) return(rep(NA_real_, length(obj$mean)))
    col <- which(grepl(paste0(level, "%"), colnames(obj$lower)))
    if (length(col) == 0) col <- 1
    list(lo = as.numeric(obj$lower[, col]), hi = as.numeric(obj$upper[, col]))
  }
  b80 <- get_band(fc, 80); b95 <- get_band(fc, 95)

  df_fc <- tibble::tibble(
    timestamp = idx_fc,
    mean = as.numeric(fc$mean),
    lo80 = b80$lo, hi80 = b80$hi,
    lo95 = b95$lo, hi95 = b95$hi
  )
  df_recent <- df_train %>% dplyr::filter(timestamp >= last_train - days(last_days))

  ggplot() +
    geom_line(data = df_recent, aes(timestamp, kW, color = "Train"), linewidth = 0.5) +
    geom_ribbon(data = df_fc, aes(timestamp, ymin = lo95, ymax = hi95, fill = "95% PI"), alpha = 0.12) +
    geom_ribbon(data = df_fc, aes(timestamp, ymin = lo80, ymax = hi80, fill = "80% PI"), alpha = 0.25) +
    geom_line(data = df_fc, aes(timestamp, mean, color = "Forecast"), linewidth = 0.9) +
    geom_line(data = df_valid, aes(timestamp, kW, color = "Test"), linewidth = 0.7) +
    scale_color_manual(values = c(Train = "blue", Forecast = "red", Test = "green"), name = "Series") +
    scale_fill_manual(values  = c("80% PI" = "#92C5DE", "95% PI" = "#56B4E9"), name = "Intervals") +
    labs(title = title, x = NULL, y = "kW") +
    theme(legend.position = "bottom")
}
```

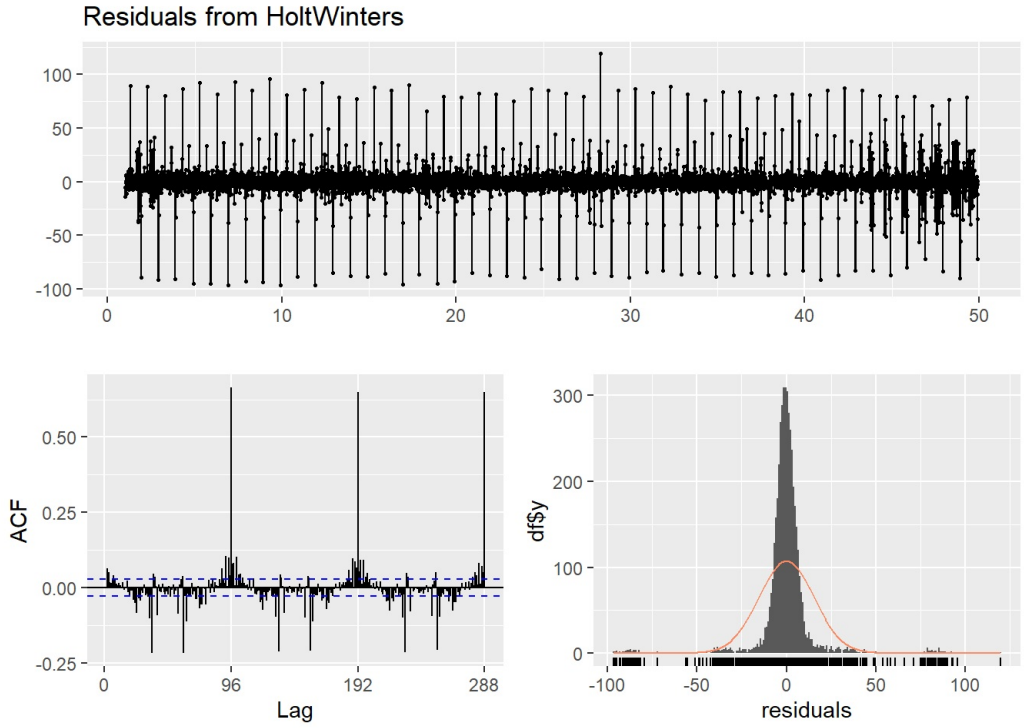
```
## ---- p3-helpers, message=FALSE, warning=FALSE ----------------------------
acc_row <- function(fc, y_valid, name) {
  tibble::as_tibble(accuracy(fc, y_valid)) %>%
    dplyr::mutate(model = name) %>%
    dplyr::select(model, RMSE, MAE, MAPE, MASE)
}

resid_check <- function(fc_obj, name) {
  cat("\n--- Residual diagnostics:", name, " ---\n")
  suppressWarnings(checkresiduals(fc_obj$model))  # prints Ljung–Box + plots ACF of residuals
}
```

Simple Exponential Smoothing (SES) — constant level

```
## ---- p3-hw-SES -----------------------------------------------------------
fit_hw_ses <- HoltWinters(y_train, beta = FALSE, gamma = FALSE)
fc_hw_ses  <- forecast::forecast(fit_hw_ses, h = h_valid, level = c(80,95))

plot_fc(df_train, df_valid, fc_hw_ses, "SES (HoltWinters)")
```




```
checkresiduals(fc_hw_ses)   # Ljung–Box + residual ACF
```

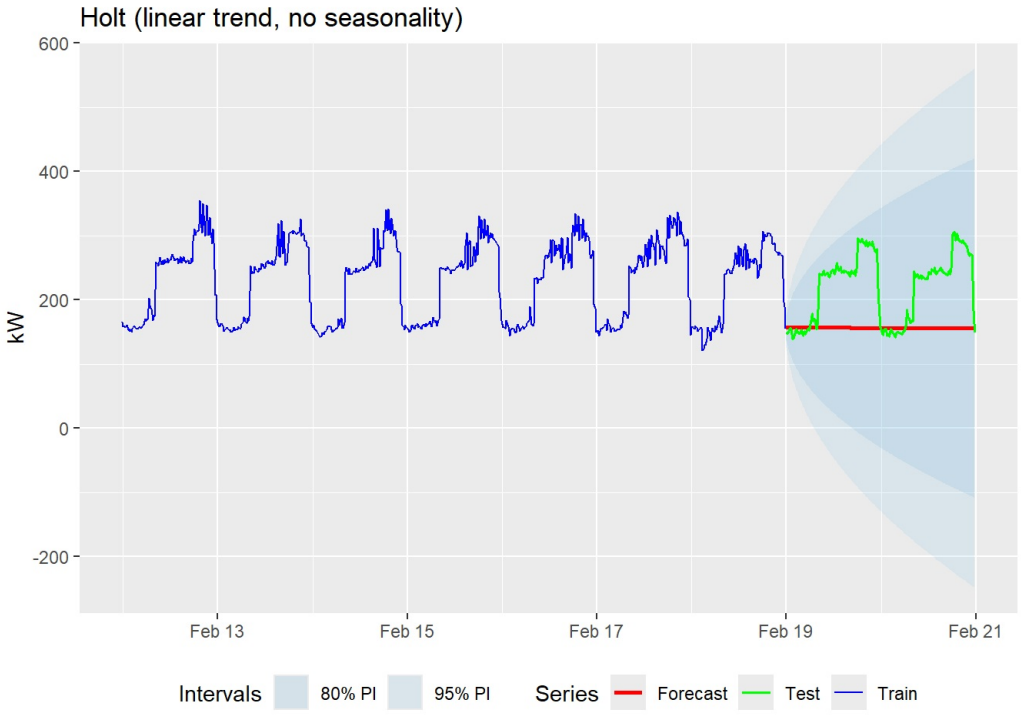



```
##
##  Ljung-Box test
##
## data:  Residuals from HoltWinters
## Q* = 6351.1, df = 192, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 192
```

```
acc_hw_ses <- acc_row(fc_hw_ses, y_valid, "SES (HW)")
```

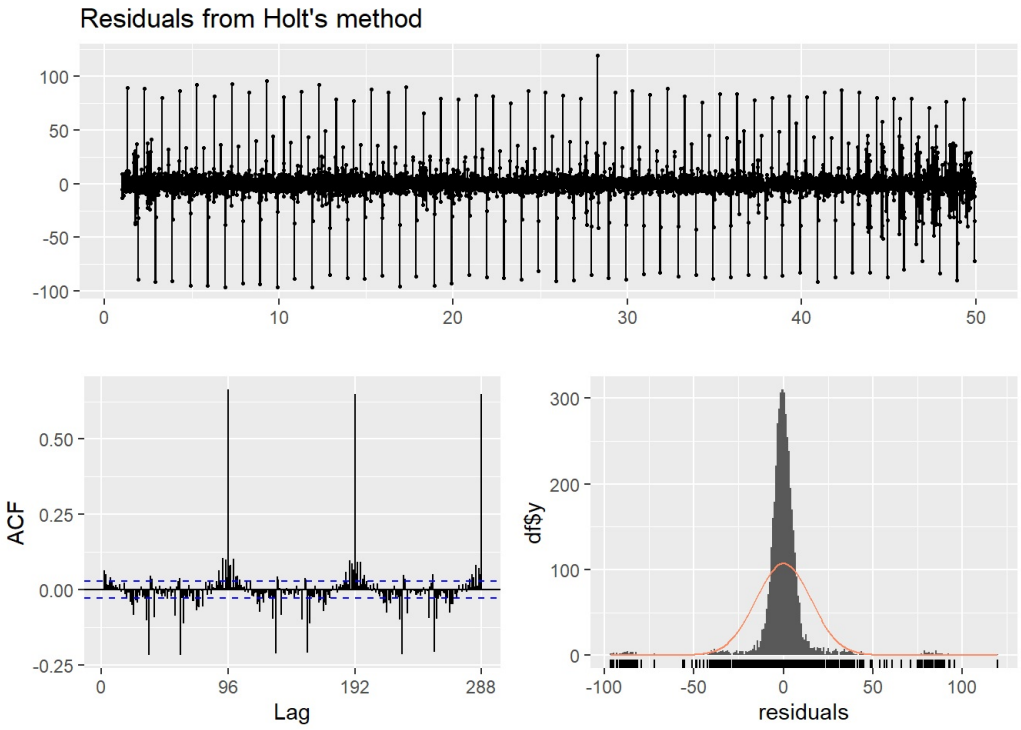
Holt (non-seasonal Holt–Winters) — linear trend, no seasonality

```
## ---- p3-Holt, message=FALSE, warning=FALSE ------------------------------
fc_holt <- holt(y_train, h = h_valid, damped = FALSE)   # classical Holt
plot_fc(df_train, df_valid, fc_holt, "Holt (linear trend, no seasonality)")
```




```
resid_check(fc_holt, "Holt")
```

```
##
## --- Residual diagnostics: Holt  ---
```

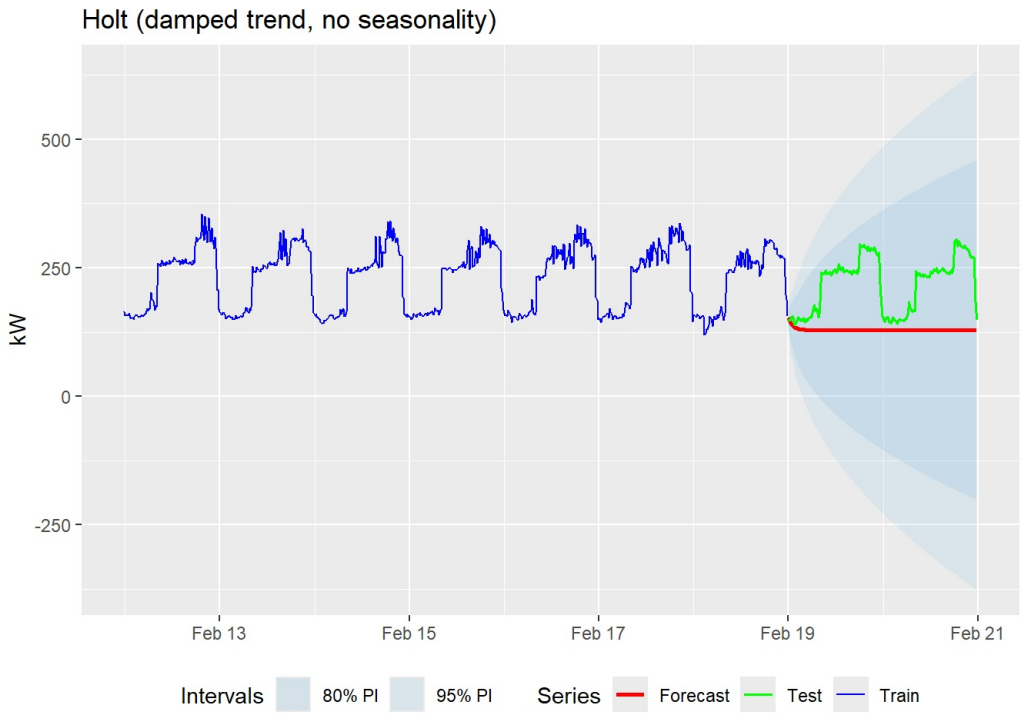



```
##
##  Ljung-Box test
##
## data:  Residuals from Holt's method
## Q* = 6351.4, df = 192, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 192
```

```
acc_holt <- acc_row(fc_holt, y_valid, "Holt")
```

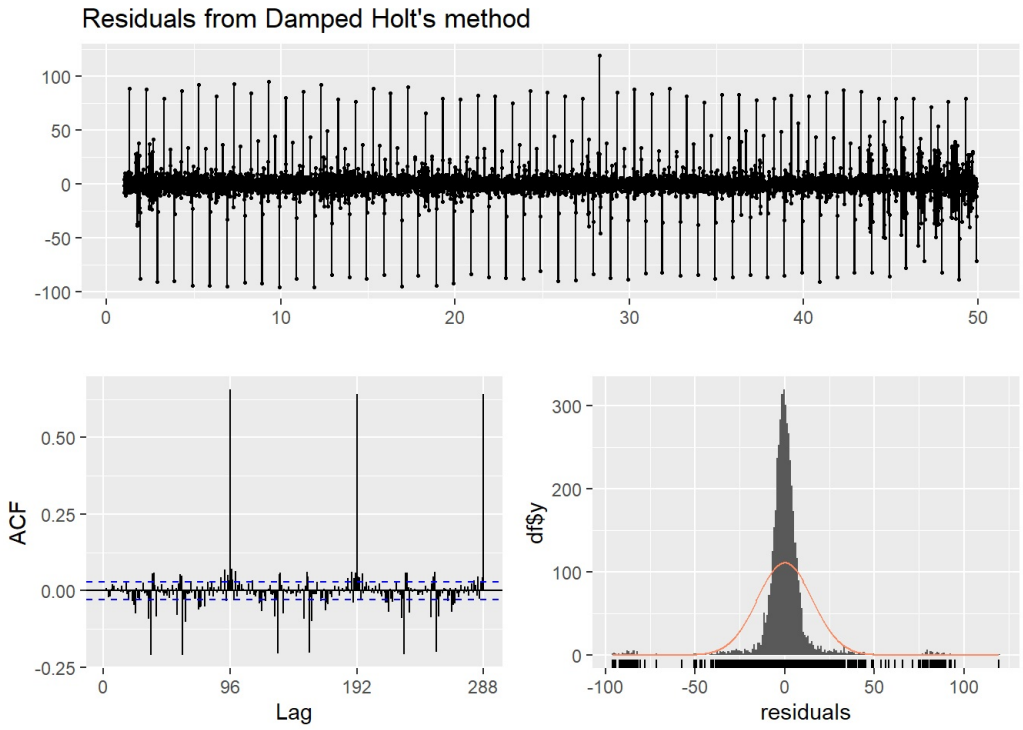
Damped version

```
## ---- p3-Holt-damped, message=FALSE, warning=FALSE ------------------------
fc_holt_d <- holt(y_train, h = h_valid, damped = TRUE)
plot_fc(df_train, df_valid, fc_holt_d, "Holt (damped trend, no seasonality)")
```




```
resid_check(fc_holt_d, "Holt (damped)")
```

```
##
## --- Residual diagnostics: Holt (damped)  ---
```

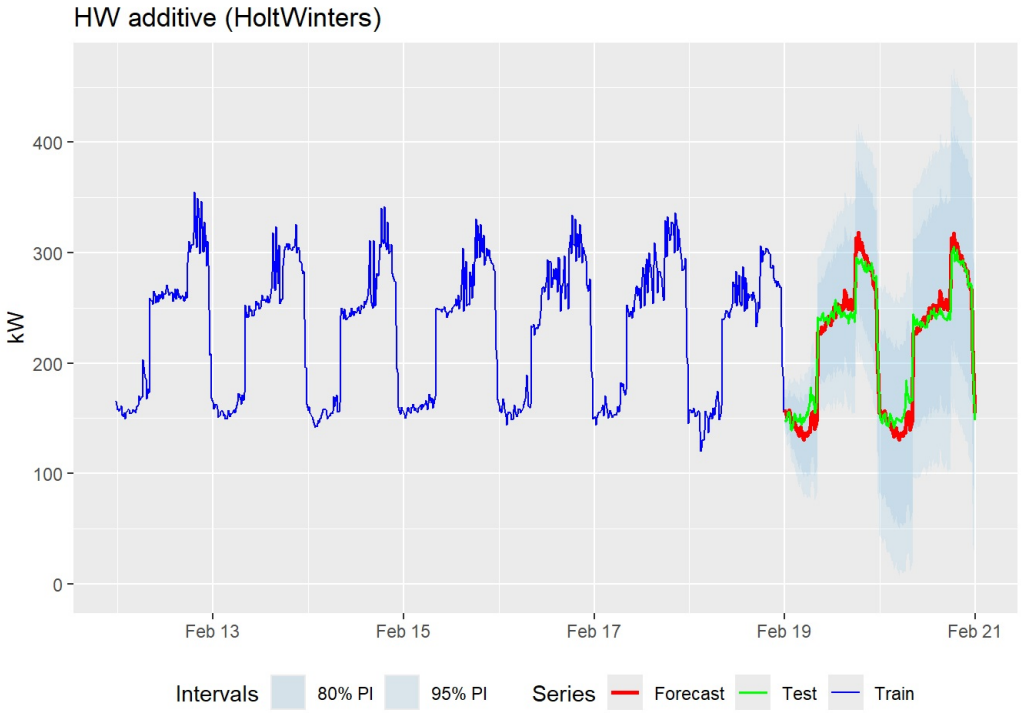



```
##
##  Ljung-Box test
##
## data:  Residuals from Damped Holt's method
## Q* = 5813.2, df = 192, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 192
```

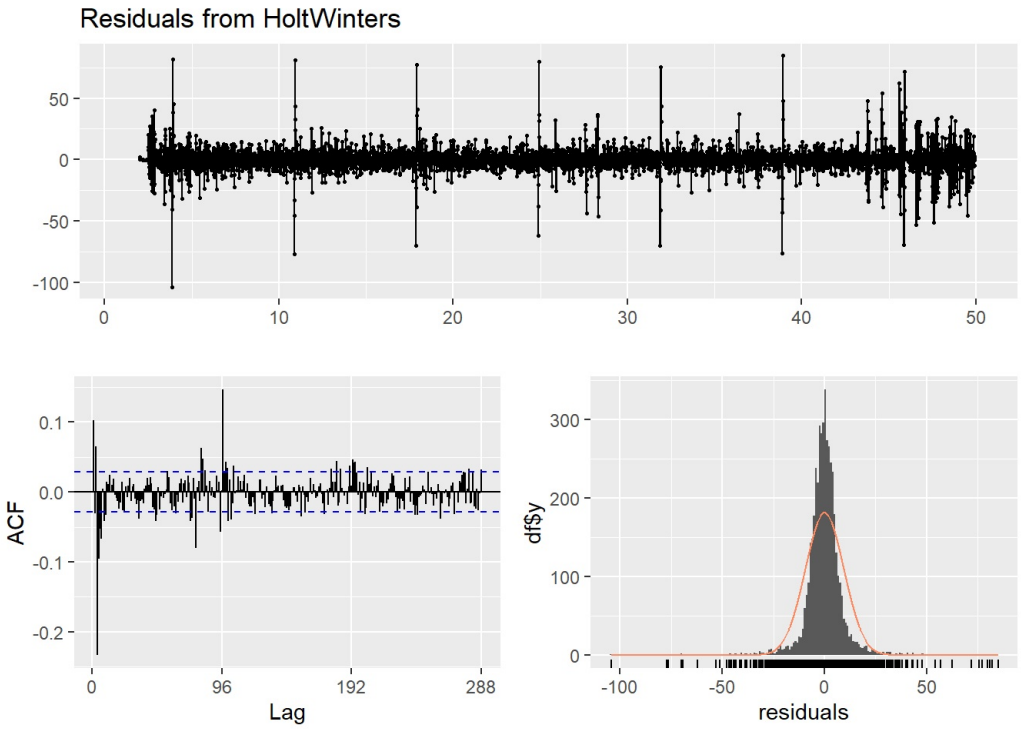
```
acc_holt_d <- acc_row(fc_holt_d, y_valid, "Holt (damped)")
```

Holt–Winters Additive — trend + additive seasonality (m = 96)

```
## ---- p3-hw-add ------------------------------------------------------------
fit_hw_add <- HoltWinters(y_train, seasonal = "additive")
fc_hw_add  <- forecast::forecast(fit_hw_add, h = h_valid, level = c(80,95))

plot_fc(df_train, df_valid, fc_hw_add, "HW additive (HoltWinters)")
```




```
checkresiduals(fc_hw_add)
```

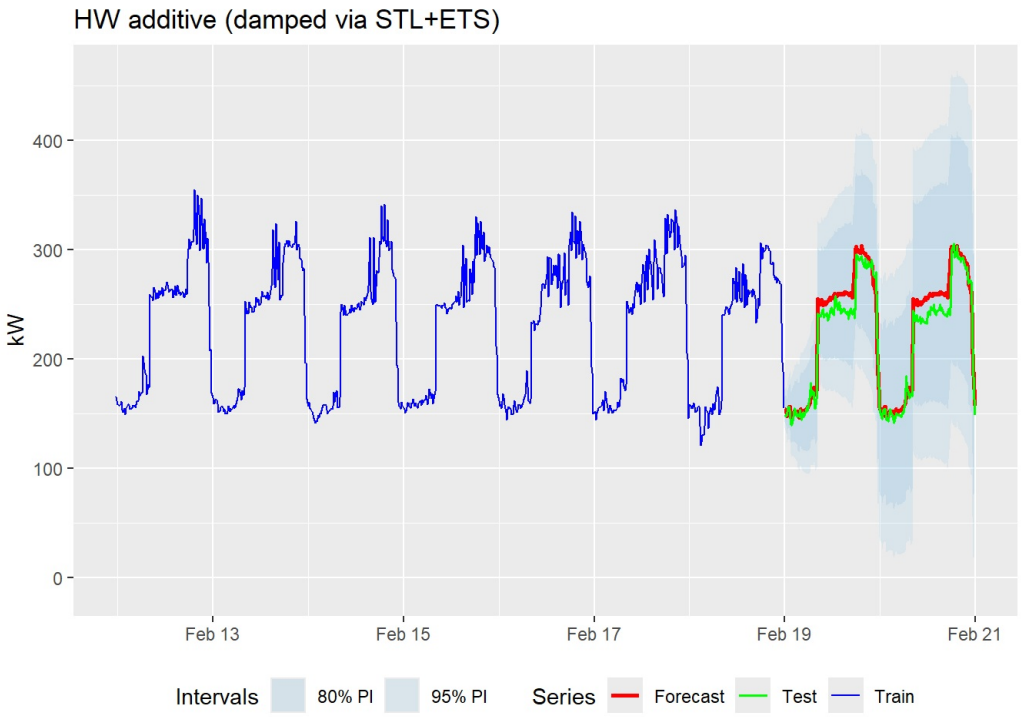



```
##
##  Ljung-Box test
##
## data:  Residuals from HoltWinters
## Q* = 866.64, df = 192, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 192
```

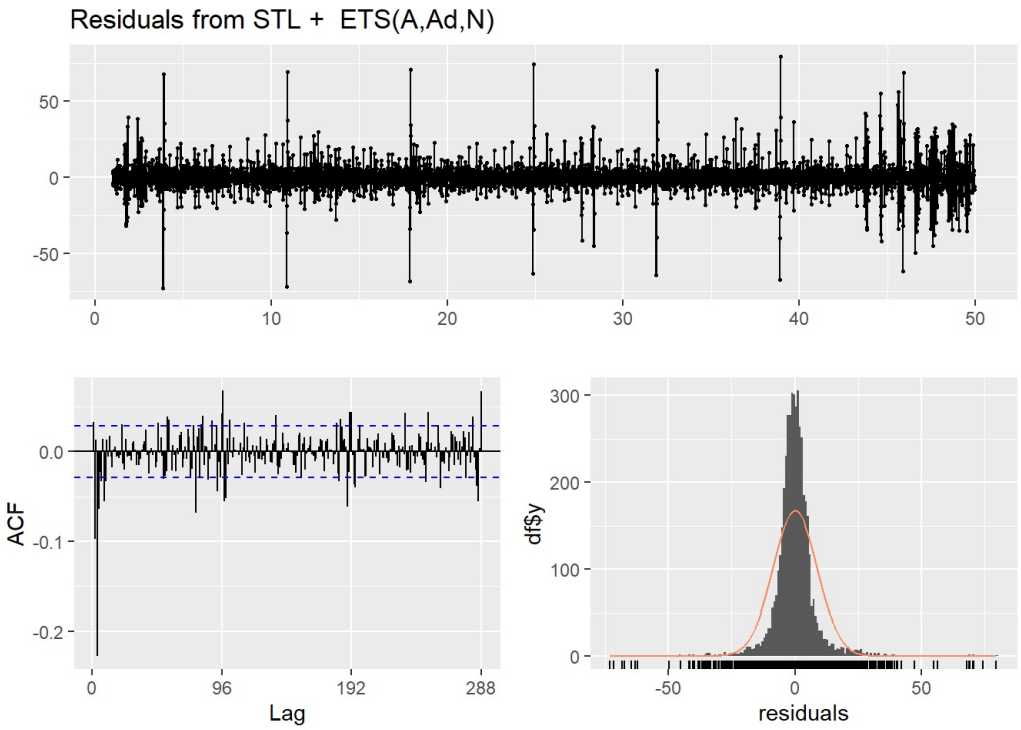
```
acc_hw_add <- acc_row(fc_hw_add, y_valid, "HW additive (HW)")
```

damped version

```
## ---- p3-hw-add-damped-STL, message=FALSE, warning=FALSE ------------------
# Damped trend + additive seasonality
fc_hw_add_d <- forecast::stlf(
  y_train, h = h_valid,
  method   = "ets",
  s.window = "periodic",   # additive seasonal
  damped   = TRUE
)
plot_fc(df_train, df_valid, fc_hw_add_d, "HW additive (damped via STL+ETS)")
```




```
checkresiduals(fc_hw_add_d)
```

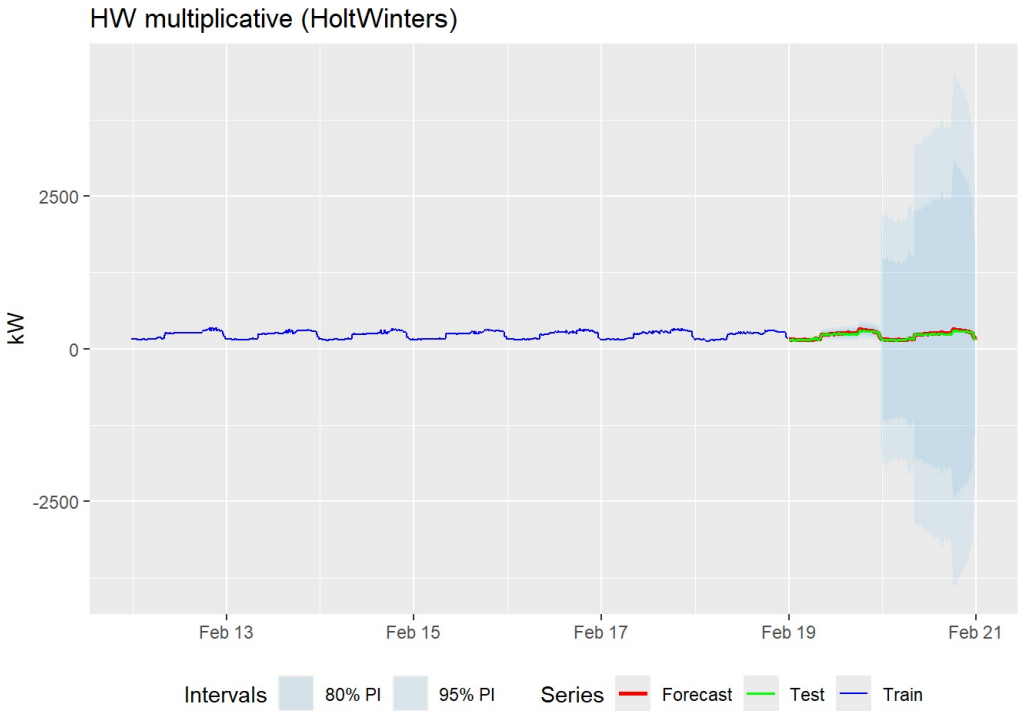



```
##
##  Ljung-Box test
##
## data:  Residuals from STL +  ETS(A,Ad,N)
## Q* = 697.86, df = 192, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 192
```

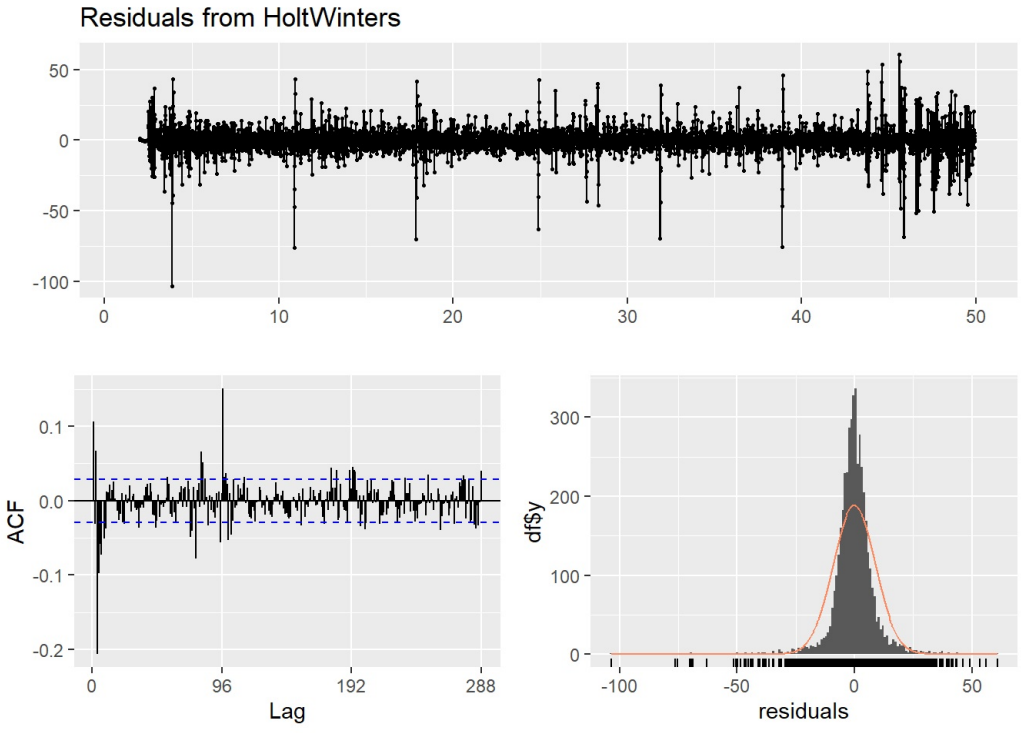
```
acc_hw_add_d <- acc_row(fc_hw_add_d, y_valid, "HW additive (damped, STL+ETS)")
```

Holt–Winters Multiplicative — trend + multiplicative seasonality

```
## ---- p3-hw-mul ------------------------------------------------------------
fit_hw_mul <- HoltWinters(y_train, seasonal = "multiplicative")
fc_hw_mul  <- forecast::forecast(fit_hw_mul, h = h_valid, level = c(80,95))

plot_fc(df_train, df_valid, fc_hw_mul, "HW multiplicative (HoltWinters)")
```




```
checkresiduals(fc_hw_mul)
```

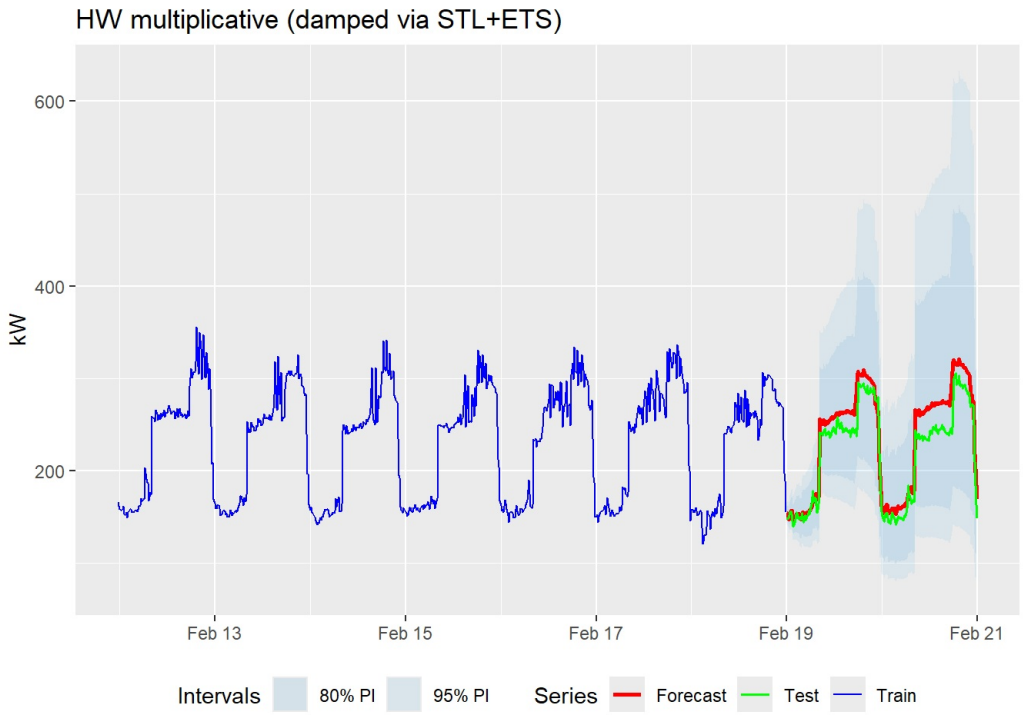



```
##
##  Ljung-Box test
##
## data:  Residuals from HoltWinters
## Q* = 847.5, df = 192, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 192
```

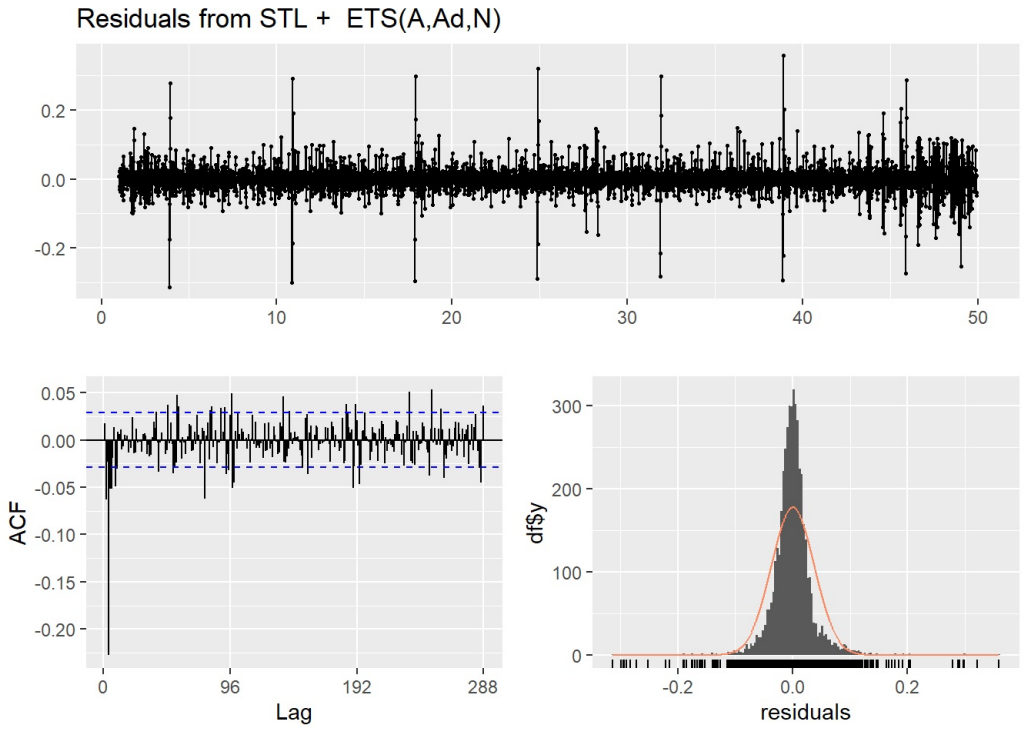
```
acc_hw_mul <- acc_row(fc_hw_mul, y_valid, "HW multiplicative (HW)")
```

damped version

```
## ---- p3-hw-mul-damped-STL, message=FALSE, warning=FALSE ------------------
# Damped trend + multiplicative seasonal (log transform)
fc_hw_mul_d <- forecast::stlf(
  y_train, h = h_valid,
  method   = "ets",
  s.window = "periodic",
  lambda   = 0,            # log scale = multiplicative seasonal
  biasadj  = TRUE,
  damped   = TRUE
)
plot_fc(df_train, df_valid, fc_hw_mul_d, "HW multiplicative (damped via STL+ETS)")
```




```
checkresiduals(fc_hw_mul_d)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from STL +  ETS(A,Ad,N)
## Q* = 620.26, df = 192, p-value < 2.2e-16
##
## Model df: 0.   Total lags used: 192
```

```
acc_hw_mul_d <- acc_row(fc_hw_mul_d, y_valid, "HW multiplicative (damped, STL+ETS)")
```

```
## ---- p3-hw-acc-update, message=FALSE, warning=FALSE ----------------------
acc_es <- dplyr::bind_rows(
  acc_hw_ses,
  acc_holt,
  acc_holt_d,
  acc_hw_add,
  acc_hw_mul,
  acc_hw_add_d,
  acc_hw_mul_d
) %>% arrange(MASE)

print(acc_es)
```

```
## # A tibble: 14 × 5
##    model                                RMSE   MAE  MAPE   MASE
##    <chr>                               <dbl> <dbl> <dbl>  <dbl>
##  1 HW multiplicative (damped, STL+ETS)  8.42  5.22  2.32  0.601
##  2 HW additive (damped, STL+ETS)        8.73  5.25  2.36  0.604
##  3 HW multiplicative (HW)               8.94  5.62  2.51  0.647
##  4 HW additive (HW)                     9.44  5.74  2.60  0.661
##  5 SES (HW)                            15.4   7.39  3.36  0.851
##  6 Holt                                15.4   7.39  3.36  0.851
##  7 Holt (damped)                       15.4   7.47  3.38  0.860
##  8 HW multiplicative (HW)              12.0   9.12  3.97  1.05
##  9 HW additive (HW)                    11.8   9.60  4.78  1.10
## 10 HW additive (damped, STL+ETS)       12.7   9.71  4.30  1.12
## 11 HW multiplicative (damped, STL+ETS) 19.3  15.9   7.03  1.83
## 12 SES (HW)                            82.4  66.9  26.5   7.70
## 13 Holt                                82.8  67.2  26.6   7.74
## 14 Holt (damped)                      105.   90.1  36.9  10.4
```

Winner (so far, ES family): HW multiplicative (damped, STL+ETS) — best test error (lowest MASE/RMSE) and best CV RMSE by a mile.

Runner-up: HW additive (damped, STL+ETS) (very close behind).

Classical HoltWinters (non-damped) trails both; SES/Holt are much worse.

```
## ---- remember-best-es ----------------------------------------------------
# Best ES model spec to reuse later (forecast day = 96 steps)
es_best_name <- "HW multiplicative (damped, STL+ETS)"
es_best <- function(y, h = 96) {
  forecast::stlf(
    y, h = h,
    method   = "ets",
    s.window = "periodic",
    lambda   = 0,       # multiplicative seasonality via log
    biasadj  = TRUE,
    damped   = TRUE
  )
}
cat("Saved best ES:", es_best_name, "\n")
```

```
## Saved best ES: HW multiplicative (damped, STL+ETS)
```

Part 4: SARIMA Moving-average components (visual)

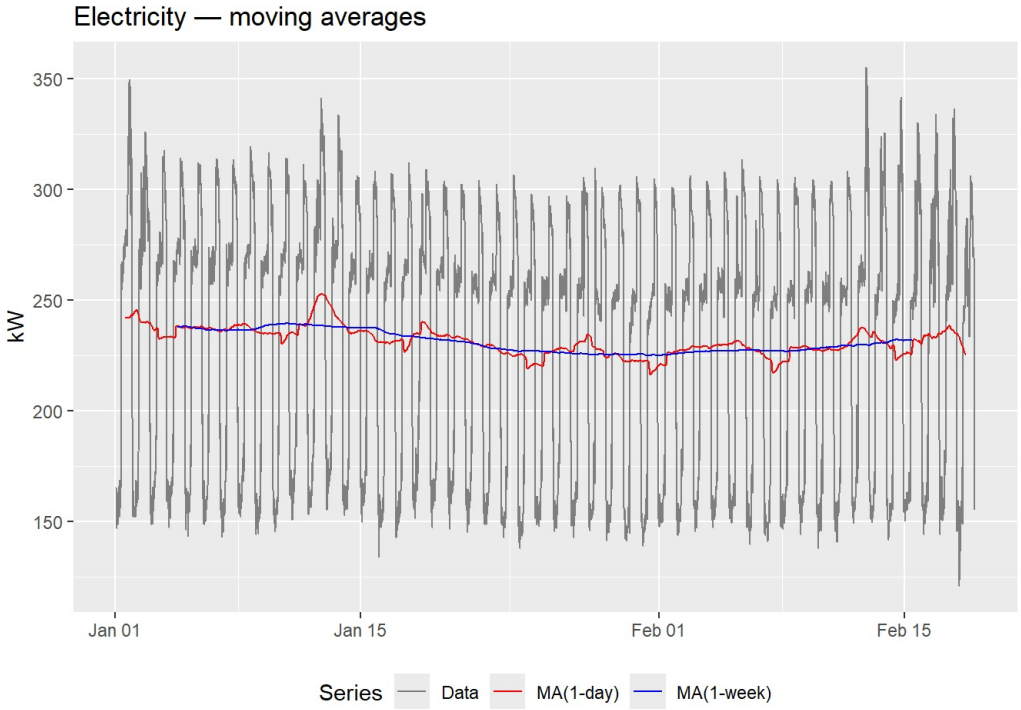
```
## ---- p4-moving-average (quiet), message=FALSE, warning=FALSE -------------
library(forecast); library(ggplot2); library(dplyr)

k_day  <- 96          # 1 day
k_week <- 96 * 7      # 1 week

ma_day  <- ma(y_train, order = k_day,  centre = TRUE)
ma_week <- ma(y_train, order = k_week, centre = TRUE)

# sanity: how many edge NAs each MA has
cat("NAs in MA(1-day): ",  sum(is.na(ma_day)),  "\n", sep = "")
```

```
## NAs in MA(1-day): 96
```

```
cat("NAs in MA(1-week): ", sum(is.na(ma_week)), "\n", sep = "")
```

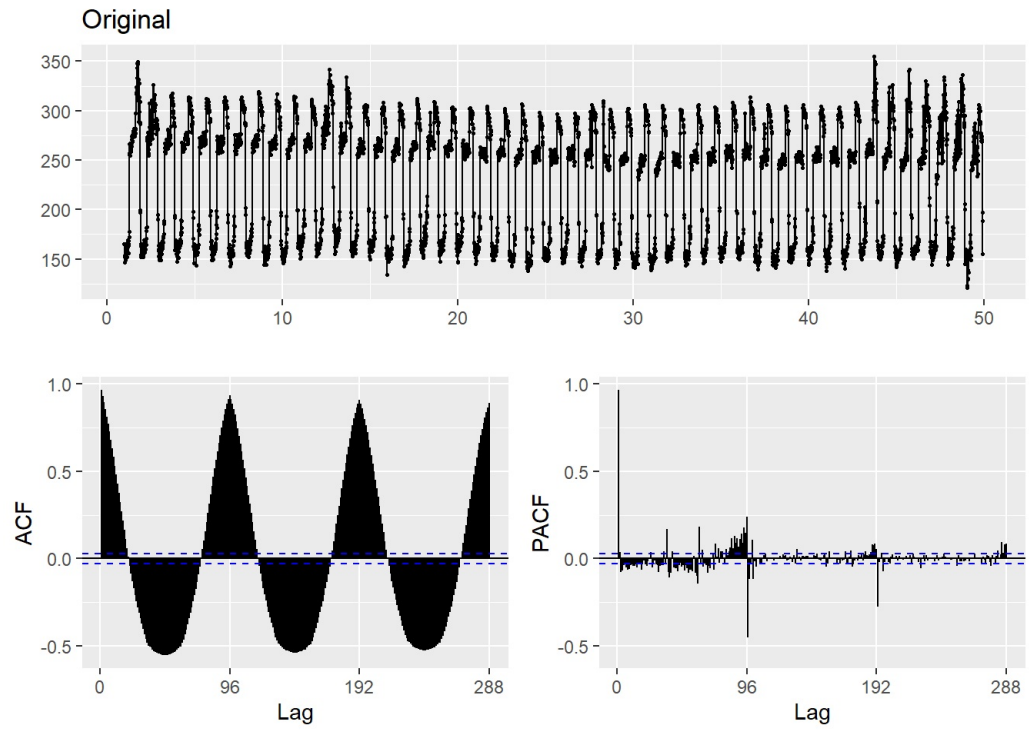
```
## NAs in MA(1-week): 672
```

```
df_ma <- tibble(
  timestamp  = df_train$timestamp,
  Data       = as.numeric(y_train),
  `MA(1-day)`  = as.numeric(ma_day),
  `MA(1-week)` = as.numeric(ma_week)
)

ggplot(df_ma, aes(timestamp, Data, color = "Data")) +
  geom_line() +
  geom_line(aes(y = `MA(1-day)`,  color = "MA(1-day)"),  na.rm = TRUE) +
  geom_line(aes(y = `MA(1-week)`, color = "MA(1-week)"), na.rm = TRUE) +
  labs(title = "Electricity — moving averages", x = NULL, y = "kW") +
  scale_color_manual(
    values = c("Data" = "grey50", "MA(1-day)" = "red", "MA(1-week)" = "blue"),
    breaks = c("Data","MA(1-day)","MA(1-week)"),
    name = "Series"
  ) +
  theme(legend.position = "bottom")
```

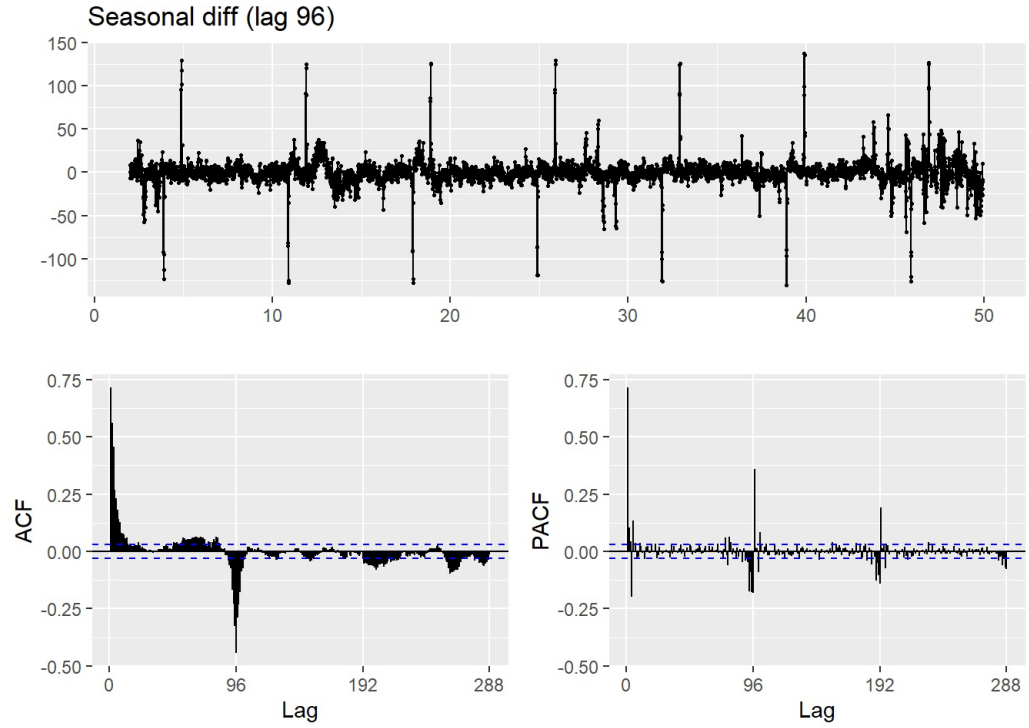


Differencing (non-seasonal & seasonal) + diagnostics

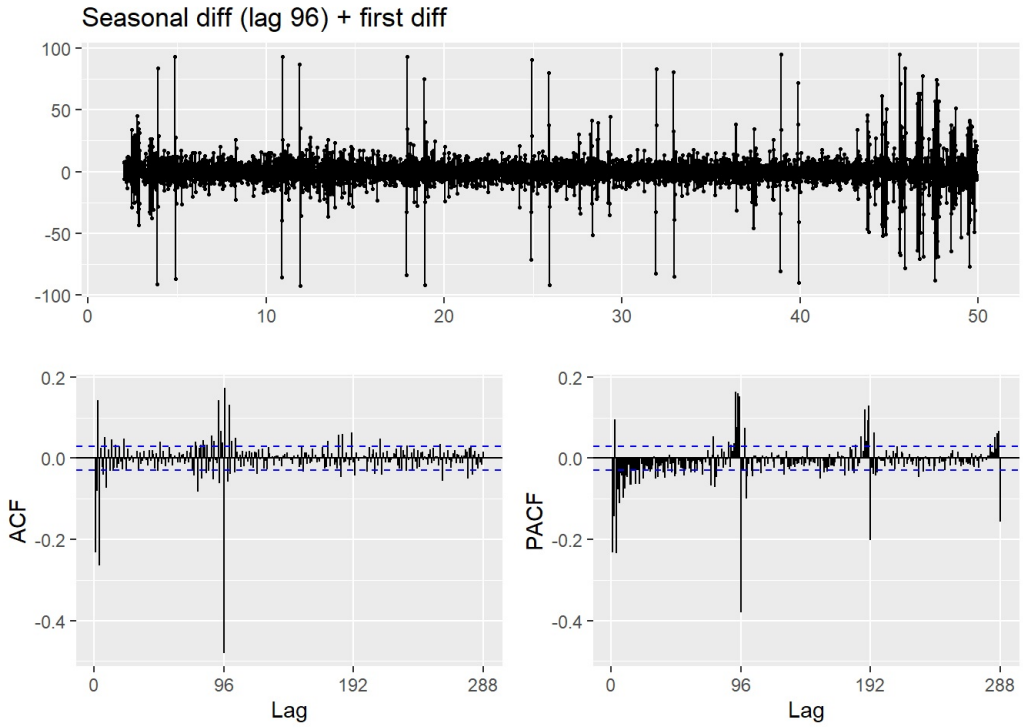
```
## ---- p4-differencing, message=FALSE, warning=FALSE ----------------------
# non-seasonal difference
dy  <- diff(y_train)

# seasonal differences (daily and weekly)
D1  <- diff(y_train, lag = 96)
D7  <- diff(y_train, lag = 96*7)

# often we use seasonal + nonseasonal together:
D1d <- diff(D1)   # (1 - B)(1 - B^96) y_t

# Show all: raw, D1 (daily), D1d (daily+nonseasonal), D7 (weekly)
ggtsdisplay(y_train, main = "Original")
```



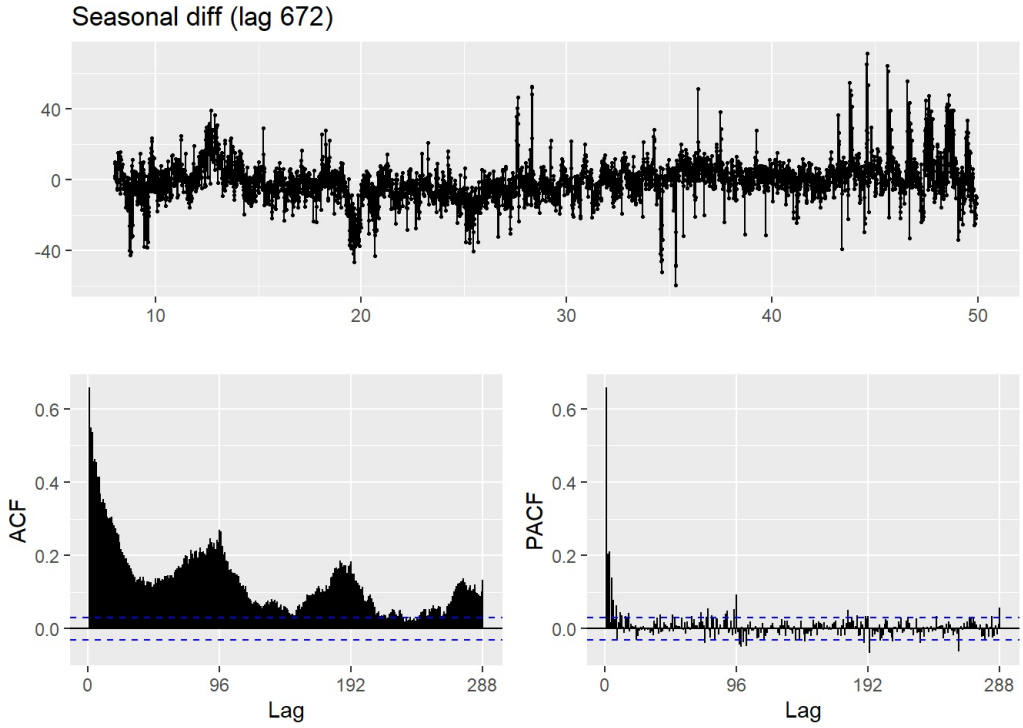

```
ggtsdisplay(D1,      main = "Seasonal diff (lag 96)")
```




```
ggtsdisplay(D1d,     main = "Seasonal diff (lag 96) + first diff")
```




```
ggtsdisplay(D7,      main = "Seasonal diff (lag 672)")
```

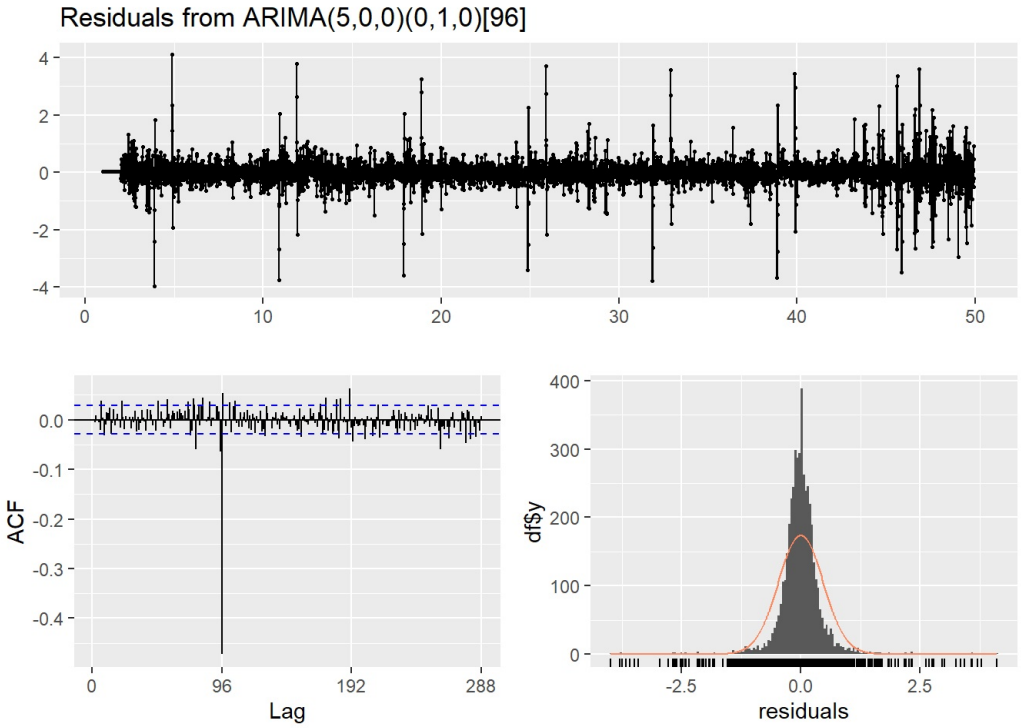


Automatic SARIMA (Box-Cox + auto.arima)

```
## ---- p4-auto-arima, message=FALSE, warning=FALSE -------------------------
fit_auto <- auto.arima(
  y_train,
  lambda      = "auto",   # Box-Cox automatically
  seasonal    = TRUE,
  stepwise    = TRUE,
  approximation = TRUE,   # faster on high-frequency
  biasadj     = TRUE
)
fc_auto <- forecast(fit_auto, h = length(y_valid))
print(fit_auto)
```

```
## Series: y_train
## ARIMA(5,0,0)(0,1,0)[96]
## Box Cox transformation: lambda= 0.4236847
##
## Coefficients:
##          ar1     ar2     ar3      ar4     ar5
##       0.6947  0.0894  0.1049  -0.2767  0.1392
## s.e.  0.0146  0.0174  0.0173   0.0174  0.0146
##
## sigma^2 = 0.2263:  log likelihood = -3109.88
## AIC=6231.76   AICc=6231.78   BIC=6270.37
```

```
checkresiduals(fit_auto)                    # Ljung–Box + residual ACF
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(5,0,0)(0,1,0)[96]
## Q* = 1436.3, df = 187, p-value < 2.2e-16
##
## Model df: 5.   Total lags used: 192
```

```
auto_acc <- tibble::as_tibble(accuracy(fc_auto, y_valid)) %>%
  dplyr::mutate(model = "auto.arima (BoxCox)") %>%
  dplyr::select(model, RMSE, MAE, MAPE, MASE)
auto_acc
```

```
## # A tibble: 2 × 5
##   model                RMSE   MAE  MAPE  MASE
##   <chr>               <dbl> <dbl> <dbl> <dbl>
## 1 auto.arima (BoxCox)  10.9  6.41  2.88 0.738
## 2 auto.arima (BoxCox)  15.5 11.7   5.57 1.35
```

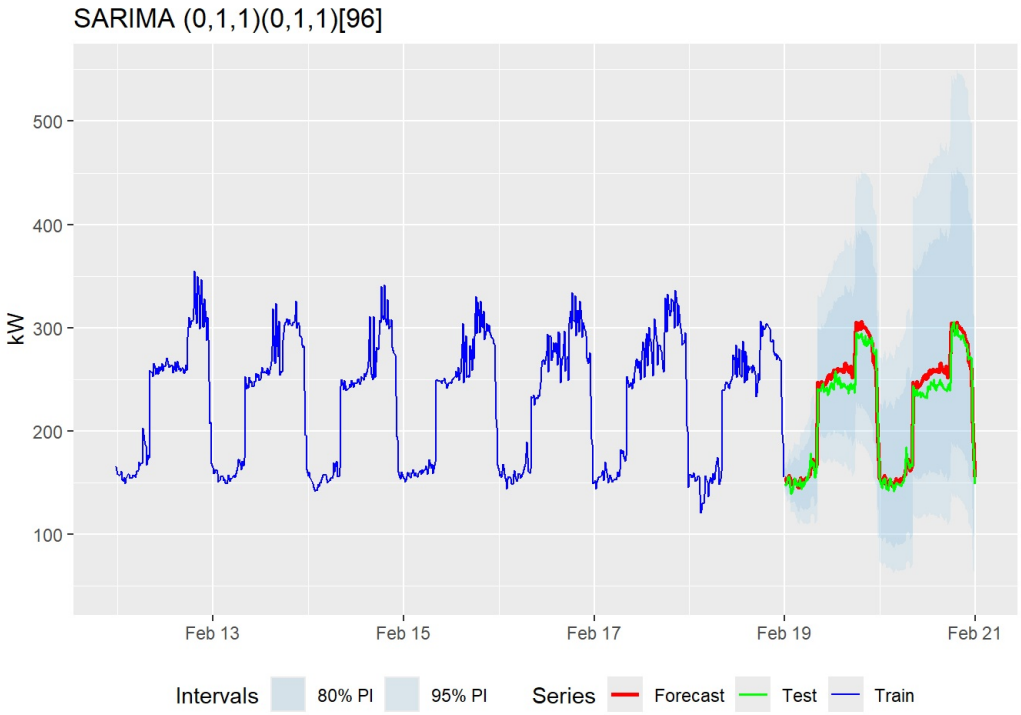
After seasonal differencing at 96 (and then + first diff), the ACF has a strong spike at seasonal lags (96, 192, …) and the low lags die off quickly → keep D = 1 (daily) and Q = 1 (seasonal MA(1)).

Low-lag behavior after D1+d shows a notable MA-like spike at lag 1 (and maybe a little spill into lags 2–3) → start with q = 1, and test q = 3 as an alternative.

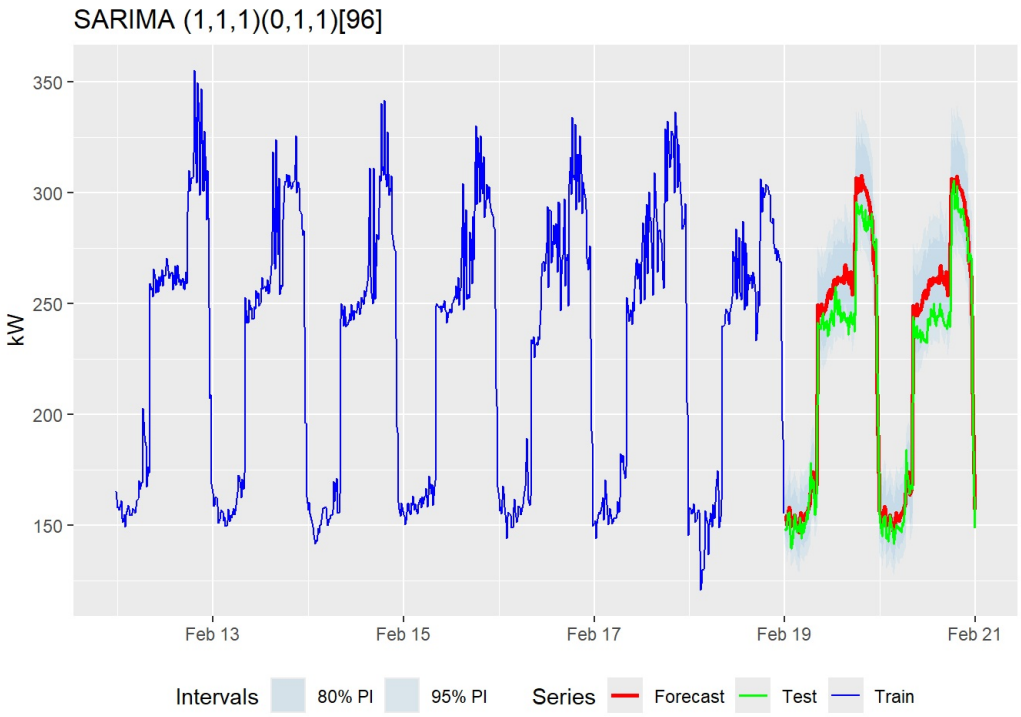
PACF doesn't scream for a nonseasonal AR term, but it's harmless to test p = 1.

Seasonal PACF doesn't strongly demand P = 1, but we'll try it once as a comparator.

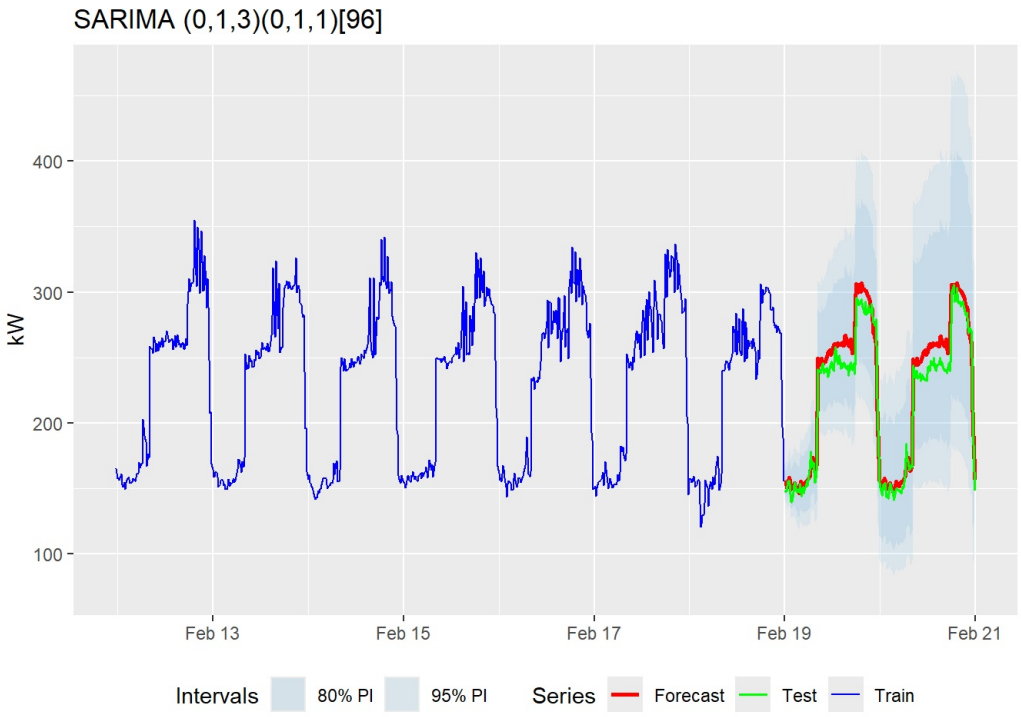
```
## ---- p4-manual-fit-4models, message=FALSE, warning=FALSE -----------------
fit_M1 <- Arima(y_train, order = c(0,1,1), seasonal = c(0,1,1), lambda = "auto")
fit_M2 <- Arima(y_train, order = c(1,1,1), seasonal = c(0,1,1), lambda = "auto")
fit_M3 <- Arima(y_train, order = c(0,1,3), seasonal = c(0,1,1), lambda = "auto")
fit_M4 <- Arima(y_train, order = c(0,1,1), seasonal = c(1,1,0), lambda = "auto")

fc_M1 <- forecast(fit_M1, h = length(y_valid))
fc_M2 <- forecast(fit_M2, h = length(y_valid))
fc_M3 <- forecast(fit_M3, h = length(y_valid))
fc_M4 <- forecast(fit_M4, h = length(y_valid))

# Plots with 80/95% intervals
plot_fc(df_train, df_valid, fc_M1, "SARIMA (0,1,1)(0,1,1)[96]")
```



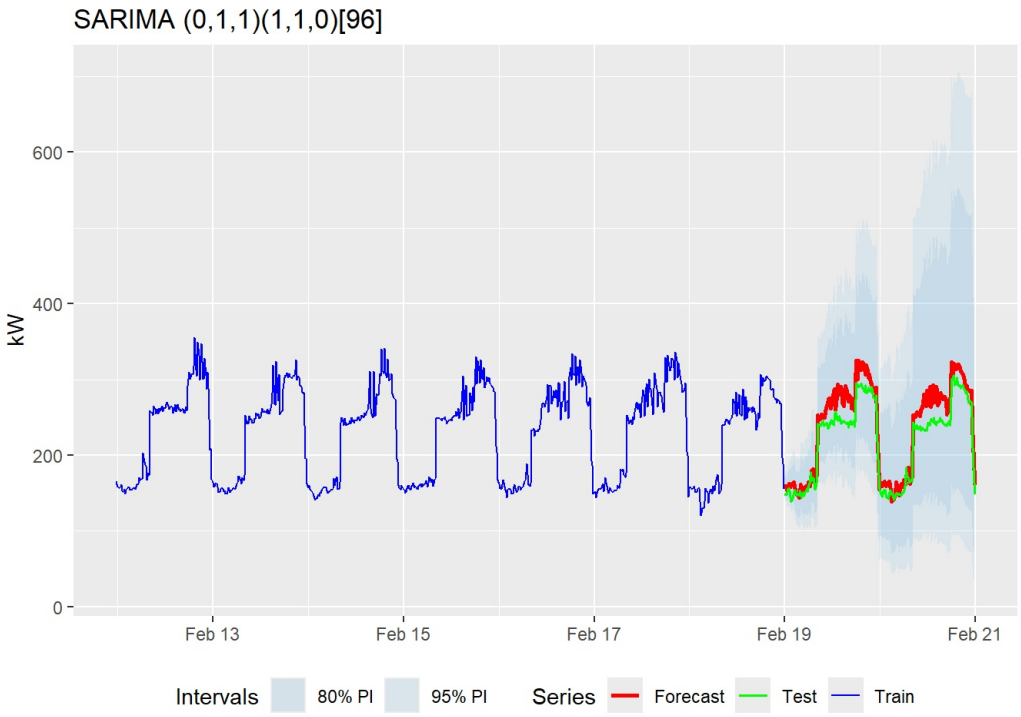

```
plot_fc(df_train, df_valid, fc_M2, "SARIMA (1,1,1)(0,1,1)[96]")
```



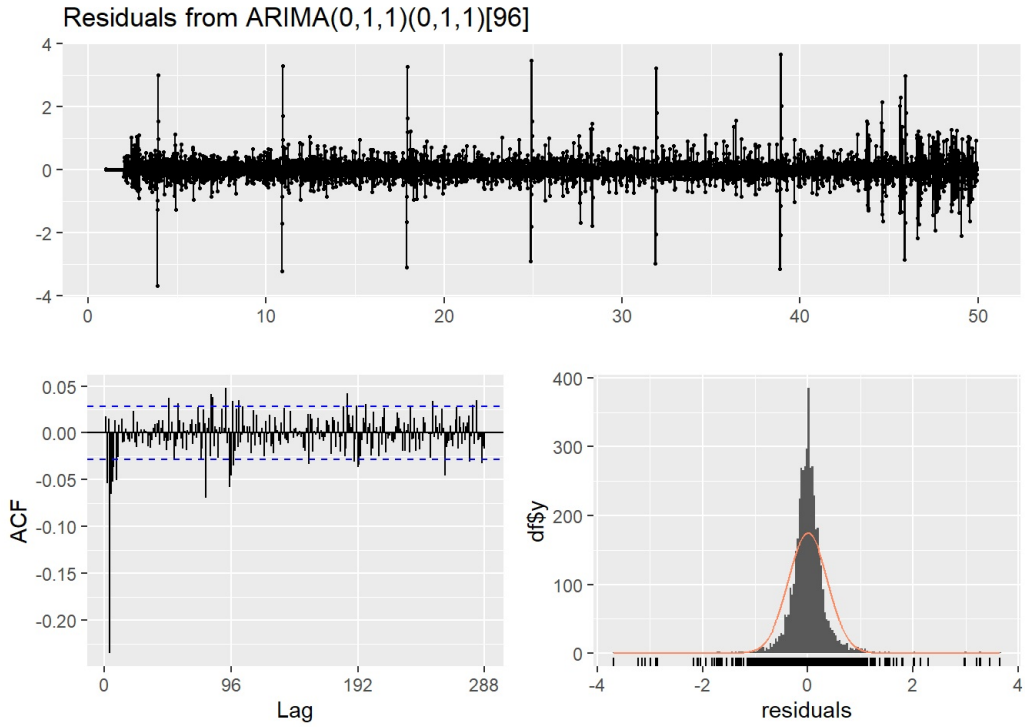

```
plot_fc(df_train, df_valid, fc_M3, "SARIMA (0,1,3)(0,1,1)[96]")
```




```
plot_fc(df_train, df_valid, fc_M4, "SARIMA (0,1,1)(1,1,0)[96]")
```

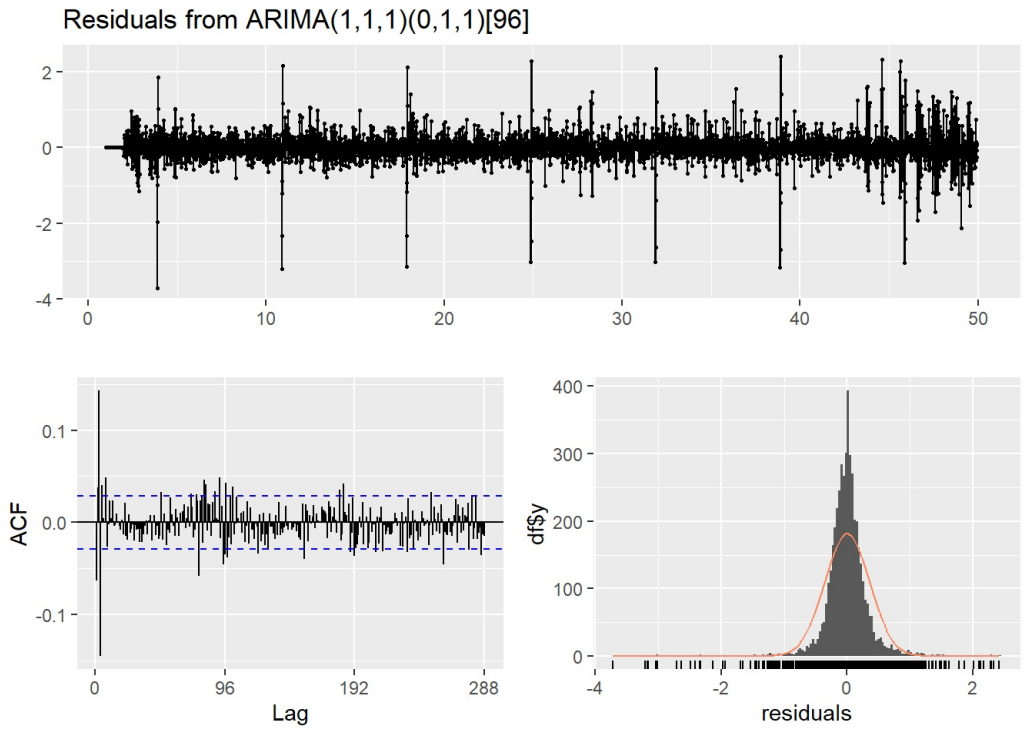



```
# Residual checks (Ljung–Box etc.)
checkresiduals(fit_M1)
```



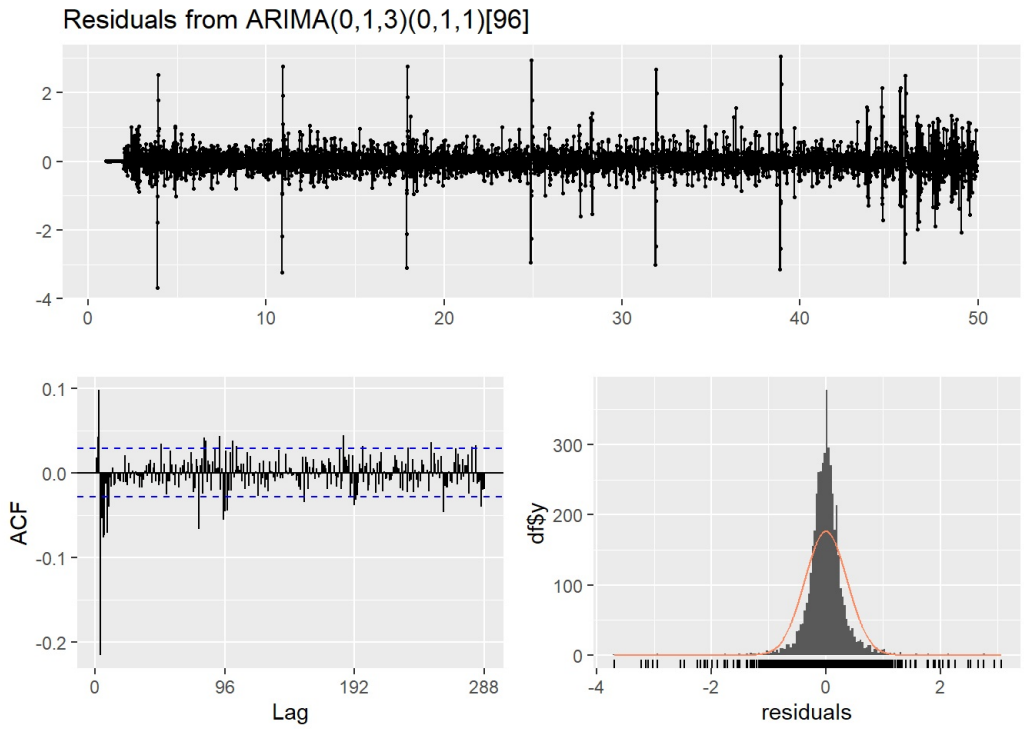

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1)(0,1,1)[96]
## Q* = 599.28, df = 190, p-value < 2.2e-16
##
## Model df: 2.   Total lags used: 192
```

```
checkresiduals(fit_M2)
```



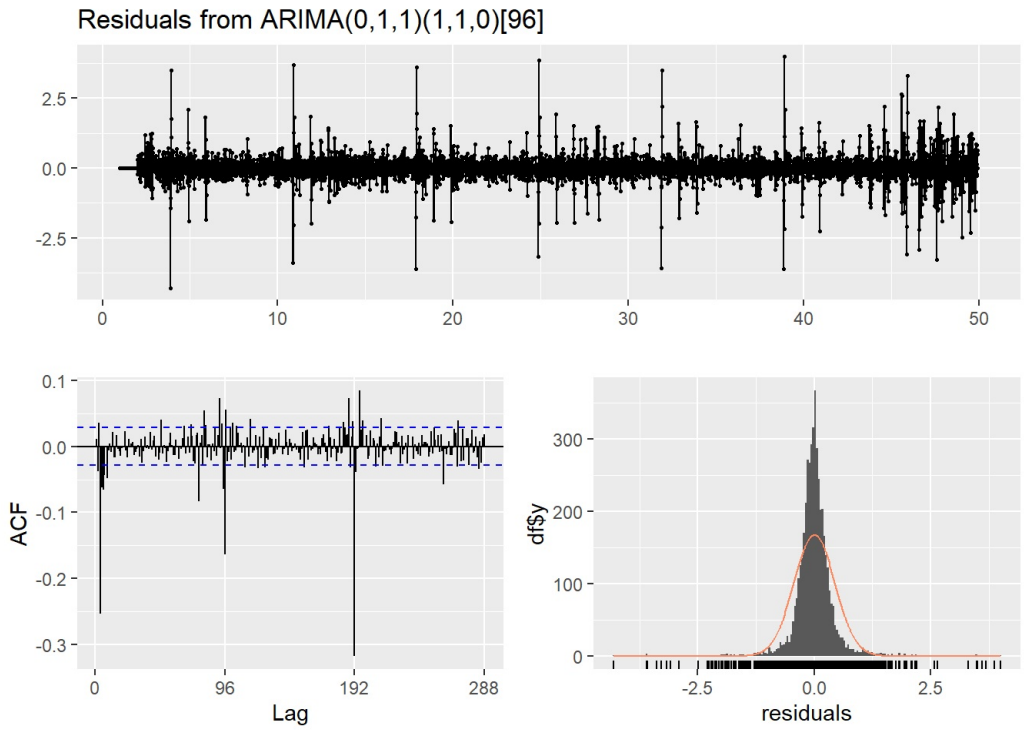

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,1)(0,1,1)[96]
## Q* = 540.37, df = 189, p-value < 2.2e-16
##
## Model df: 3.   Total lags used: 192
```

```
checkresiduals(fit_M3)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,3)(0,1,1)[96]
## Q* = 633.58, df = 188, p-value < 2.2e-16
##
## Model df: 4.   Total lags used: 192
```

```
checkresiduals(fit_M4)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1)(1,1,0)[96]
## Q* = 1370.3, df = 190, p-value < 2.2e-16
##
## Model df: 2.   Total lags used: 192
```

```
# Test-set accuracy table
acc_arima <- bind_rows(
  as_tibble(accuracy(fc_M1, y_valid)) %>% mutate(model = "ARIMA(0,1,1)(0,1,1)[96]"),
  as_tibble(accuracy(fc_M2, y_valid)) %>% mutate(model = "ARIMA(1,1,1)(0,1,1)[96]"),
  as_tibble(accuracy(fc_M3, y_valid)) %>% mutate(model = "ARIMA(0,1,3)(0,1,1)[96]"),
  as_tibble(accuracy(fc_M4, y_valid)) %>% mutate(model = "ARIMA(0,1,1)(1,1,0)[96]")
) %>% select(model, RMSE, MAE, MAPE, MASE) %>% arrange(MASE)

print(acc_arima)
```

```
## # A tibble: 8 × 5
##   model                    RMSE   MAE  MAPE  MASE
##   <chr>                   <dbl> <dbl> <dbl> <dbl>
## 1 ARIMA(1,1,1)(0,1,1)[96]  8.22  5.00  2.24 0.576
## 2 ARIMA(0,1,3)(0,1,1)[96]  8.35  5.11  2.30 0.588
## 3 ARIMA(0,1,1)(0,1,1)[96]  8.39  5.12  2.31 0.590
## 4 ARIMA(0,1,1)(1,1,0)[96]  9.95  5.93  2.68 0.682
## 5 ARIMA(0,1,1)(0,1,1)[96] 10.8   8.58  3.77 0.988
## 6 ARIMA(0,1,3)(0,1,1)[96] 11.8   9.53  4.20 1.10
## 7 ARIMA(1,1,1)(0,1,1)[96] 12.1   9.78  4.31 1.13
## 8 ARIMA(0,1,1)(1,1,0)[96] 22.7  18.8   8.32 2.16
```

```
# Quick AICc comparison too
aicc_tbl <- tibble::tibble(
  model = c("M1 011-011", "M2 111-011", "M3 013-011", "M4 011-110"),
  AICc  = c(fit_M1$aicc, fit_M2$aicc, fit_M3$aicc, fit_M4$aicc)
) %>% arrange(AICc)
print(aicc_tbl)
```

```
## # A tibble: 4 × 2
##   model       AICc
##   <chr>      <dbl>
## 1 M2 111-011 3835.
## 2 M3 013-011 4148.
## 3 M1 011-011 4201.
## 4 M4 011-110 5631.
```

SARIMA selection (kept for the final bake-off) Winner: ARIMA(1,1,1)(0,1,1)[96] + Box-Cox

Best test error (lowest MASE ≈ 0.576) and best AICc (≈ 3834.7) among the manual set.

Runner-up: ARIMA(0,1,3)(0,1,1)[96] + Box-Cox (very close MASE ≈ 0.588).

auto.arima picked ARIMA(5,0,0)(0,1,0)[96] with much worse test error (MASE ≈ 0.738).

Note: Ljung–Box p-values are tiny for all models (very large n makes the test hyper-sensitive), but the residual ACFs look small and well-behaved—good enough to keep these two for comparison.

```
## Save best SARIMA choices for later
sarima_best_name <- "ARIMA(1,1,1)(0,1,1)[96] + BoxCox"
sarima_best <- function(y, h = 96) {
  forecast(Arima(y, order = c(1,1,1), seasonal = c(0,1,1), lambda = "auto"), h = h)
}

sarima_alt_name <- "ARIMA(0,1,3)(0,1,1)[96] + BoxCox"
sarima_alt <- function(y, h = 96) {
  forecast(Arima(y, order = c(0,1,3), seasonal = c(0,1,1), lambda = "auto"), h = h)
}
```

Part 5 : ML models Setup (reuse your split)

```
# Safety: check required packages (SVM/XGB)
if (!requireNamespace("e1071", quietly = TRUE)) stop("Please install 'e1071' for SVM.")
if (!requireNamespace("xgboost", quietly = TRUE)) stop("Please install 'xgboost'.")

# 1) Rebuild the ts from outage-fixed training table
stopifnot(all(c("timestamp","kW") %in% names(train_ts_outagefix)))
y_full <- ts(as.numeric(train_ts_outagefix$kW), frequency = 96, start = c(1,1))

# 2) Hold out last 2 days (192 points) for validation
h_valid <- 96 * 2
n <- length(y_full)
y_train <- head(y_full, n - h_valid)
y_valid <- tail(y_full, h_valid)

# 3) Data frames for plotting
df_train <- head(train_ts_outagefix, n - h_valid)
df_valid <- tail(train_ts_outagefix, h_valid)

cat("Lengths -> train:", length(y_train), " valid:", length(y_valid), "\n")
```

```
## Lengths -> train: 4699  valid: 192
```

```
stopifnot(exists("y_train"), exists("y_valid"), exists("df_train"), exists("df_valid"))
h_valid <- length(y_valid)
freq <- frequency(y_train)   # 96
```

Supervised data (lags → next value) Two lag sets, like in your class:

Set A (classy): last 12 observations (t−1…t−12) Set B (PACF/seasonal): 1:8, 96:98, 192, 672 (captures recent + daily/2-day/weekly seasonality)

```
## ---- p5-build-supervised -------------------------------------------------
## ---- p5-01-plot-helper, message=FALSE, warning=FALSE ---------------------
# Robust forecast plotting (works even if you only pass 'mean')
plot_fc <- function(df_train, df_valid, fc, title, last_days = 7) {
  last_train <- max(df_train$timestamp)
  idx_fc <- seq(from = last_train + minutes(15), by = "15 min", length.out = length(fc$mean))

  get_band <- function(obj, level) {
    if (is.null(obj$lower) || is.null(obj$upper)) return(list(lo = rep(NA_real_, length(obj$mean)),
                                                              hi = rep(NA_real_, length(obj$mean))))
    col <- which(grepl(paste0(level, "%"), colnames(obj$lower)))
    if (length(col) == 0) col <- 1
    list(lo = as.numeric(obj$lower[, col]), hi = as.numeric(obj$upper[, col]))
  }
  b80 <- get_band(fc, 80); b95 <- get_band(fc, 95)

  df_fc <- tibble::tibble(
    timestamp = idx_fc,
    mean = as.numeric(fc$mean),
    lo80 = b80$lo, hi80 = b80$hi,
    lo95 = b95$lo, hi95 = b95$hi
  )
  df_recent <- df_train %>% dplyr::filter(timestamp >= last_train - days(last_days))

  ggplot() +
    geom_line(data = df_recent, aes(timestamp, kW, color = "Train"), linewidth = 0.5) +
    geom_ribbon(data = df_fc, aes(timestamp, ymin = lo95, ymax = hi95, fill = "95% PI"), alpha = 0.12) +
    geom_ribbon(data = df_fc, aes(timestamp, ymin = lo80, ymax = hi80, fill = "80% PI"), alpha = 0.25) +
    geom_line(data = df_fc, aes(timestamp, mean, color = "Forecast"), linewidth = 0.9) +
    geom_line(data = df_valid, aes(timestamp, kW, color = "Test"), linewidth = 0.7) +
    scale_color_manual(values = c(Train = "grey50", Forecast = "#0072B2", Test = "black"), name = "Series") +
    scale_fill_manual(values  = c("80% PI" = "#92C5DE", "95% PI" = "#56B4E9"), name = "Intervals") +
    labs(title = title, x = NULL, y = "kW") +
    theme(legend.position = "bottom")
}
```

Helper to forecast sequentially (one-step ahead loop, feeding back preds):

```
## ---- p5-02-helpers, message=FALSE, warning=FALSE ------------------------
# Build supervised rows: features at given lags -> next value
build_supervised <- function(y, lags) {
  yv <- as.numeric(y); L <- max(lags)
  stopifnot(length(yv) > L + 1)
  nrows <- length(yv) - L
  X <- do.call(rbind, lapply(seq_len(nrows), function(i) yv[i + L - lags]))
  y_next <- yv[(L + 1):length(yv)]
  colnames(X) <- paste0("x", lags)
  data.frame(X, y = y_next)
}

# One-step-ahead sequential forecasting for horizon h (feed predictions back)
seq_forecast <- function(last_y, lags, h, predict_fun) {
  L <- max(lags)
  buf <- as.numeric(tail(last_y, L))
  preds <- numeric(h)
  for (t in seq_len(h)) {
    x <- buf[L - lags + 1]                       # pick in lag order
    preds[t] <- as.numeric(predict_fun(as.numeric(x)))
    # Keep only the last L values after appending the new prediction(s)
    buf <- tail(c(buf, preds[t]), L)
  }
  preds
}

# Accuracy on numeric vectors (no ts/window alignment issues)
acc_vec <- function(pred, truth, name) {
  pred  <- as.numeric(pred)
  truth <- as.numeric(truth)
  n <- min(length(pred), length(truth))
  e <- truth[seq_len(n)] - pred[seq_len(n)]
  rmse <- sqrt(mean(e^2))
  mae  <- mean(abs(e))
  mape <- mean(abs(e) / pmax(truth[seq_len(n)], 1e-8)) * 100
  mase_den <- mean(abs(diff(truth[seq_len(n)])), na.rm = TRUE)
  tibble::tibble(model = name, RMSE = rmse, MAE = mae, MAPE = mape, MASE = mae / mase_den)
}
```

```
## ---- p5-03-lagsets, message=FALSE, warning=FALSE ------------------------
# Lag sets
lags_A <- 1:12
lags_B <- c(1:8, 96:98, 192, 672)   # recent + daily/2-day/weekly

# Supervised datasets from the training part
dat_A <- build_supervised(y_train, lags_A)
dat_B <- build_supervised(y_train, lags_B)

# Last buffers for sequential forecasting
lastA <- tail(as.numeric(y_train), max(lags_A))
lastB <- tail(as.numeric(y_train), max(lags_B))

str(dat_A); str(dat_B)
```

```
## 'data.frame':    4687 obs. of  13 variables:
##  $ x1 : num  162 162 162 166 159 ...
##  $ x2 : num  155 162 162 162 166 ...
##  $ x3 : num  149 155 162 162 162 ...
##  $ x4 : num  152 149 155 162 162 ...
##  $ x5 : num  163 152 149 155 162 ...
##  $ x6 : num  158 163 152 149 155 ...
##  $ x7 : num  159 158 163 152 149 ...
##  $ x8 : num  154 159 158 163 152 ...
##  $ x9 : num  154 154 159 158 163 ...
##  $ x10: num  147 154 154 159 158 ...
##  $ x11: num  152 147 154 154 159 ...
##  $ x12: num  165 152 147 154 154 ...
##  $ y  : num  162 162 166 159 159 ...
```

```
## 'data.frame':    4027 obs. of  14 variables:
##  $ x1  : num  162 166 154 157 163 ...
##  $ x2  : num  163 162 166 154 157 ...
##  $ x3  : num  156 163 162 166 154 ...
##  $ x4  : num  161 156 163 162 166 ...
##  $ x5  : num  165 161 156 163 162 ...
##  $ x6  : num  167 165 161 156 163 ...
##  $ x7  : num  202 167 165 161 156 ...
##  $ x8  : num  194 202 167 165 161 ...
##  $ x96 : num  164 152 143 148 155 ...
##  $ x97 : num  160 164 152 143 148 ...
##  $ x98 : num  153 160 164 152 143 ...
##  $ x192: num  169 154 149 159 151 ...
##  $ x672: num  165 152 147 154 154 ...
##  $ y   : num  166 154 157 163 159 ...
```

Random Forest (Set A and Set B)

```
## ---- p5-15-randomforest, message=FALSE, warning=FALSE --------------------
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

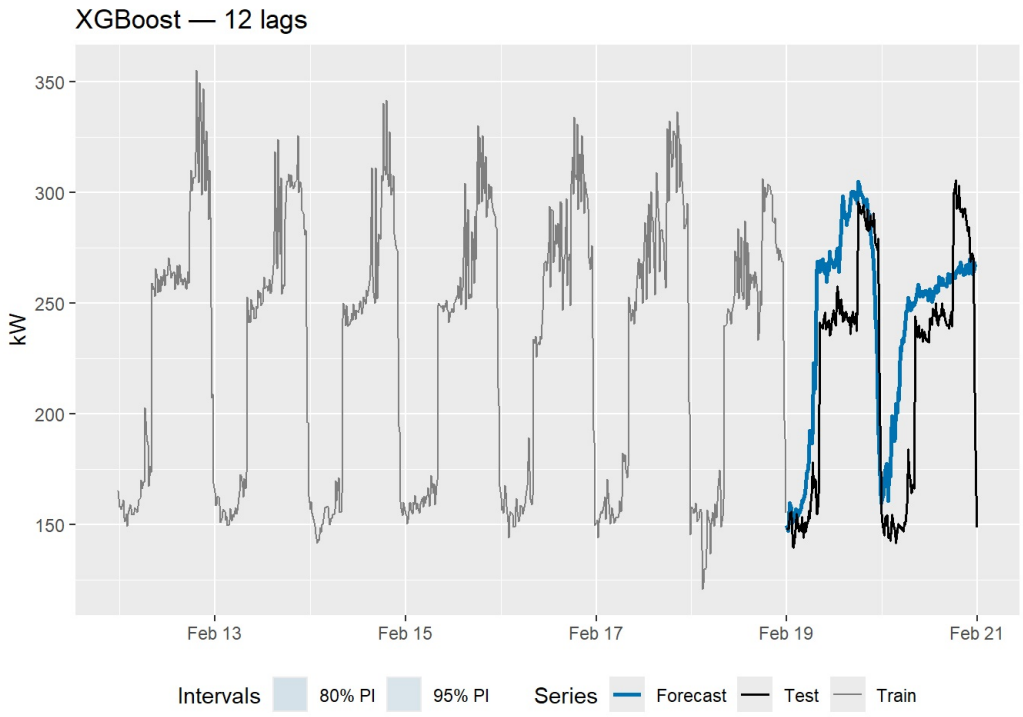
```
set.seed(123)

# A: 12 lags
rf_A <- randomForest(
  x = as.matrix(dat_A[ , -ncol(dat_A), drop = FALSE]),
  y = dat_A$y,
  ntree = 500,
  mtry  = floor(sqrt(ncol(dat_A) - 1))
)
predRF_A <- seq_forecast(
  lastA, lags_A, length(y_valid),
  function(x) predict(rf_A, newdata = row_mat(x, length(lags_A)))
)
accRF_A <- acc_vec(predRF_A, y_valid, "Random Forest (12 lags)")

# B: PACF/seasonal lags
rf_B <- randomForest(
  x = as.matrix(dat_B[ , -ncol(dat_B), drop = FALSE]),
  y = dat_B$y,
  ntree = 500,
  mtry  = floor(sqrt(ncol(dat_B) - 1))
)
predRF_B <- seq_forecast(
  lastB, lags_B, length(y_valid),
  function(x) predict(rf_B, newdata = row_mat(x, length(lags_B)))
)
accRF_B <- acc_vec(predRF_B, y_valid, "Random Forest (PACF/seasonal lags)")
```

XGBoost

```
## ---- xgb-helpers-safe -----------------------------------------------------
# Sanity checker
check_design <- function(X, y, expected_p, tag) {
  stopifnot(is.matrix(X))
  if (ncol(X) != expected_p) stop(paste0("[", tag, "] expected ", expected_p, " features but got ", ncol(X)))
  if (anyNA(X)) warning("[", tag, "] design matrix contains NA; XGBoost handles NA but check your lags.")
  if (length(y) != nrow(X)) stop(paste0("[", tag, "] rows(", nrow(X), ") != length(y)(", length(y), ")"))
  invisible(TRUE)
}

# One-row maker with explicit ncol
row_mat <- function(x, p) {
  x <- as.numeric(x)
  if (length(x) != p) stop(paste0("newdata row has length ", length(x), " but expected ", p))
  matrix(x, nrow = 1, ncol = p, byrow = TRUE)
}
```

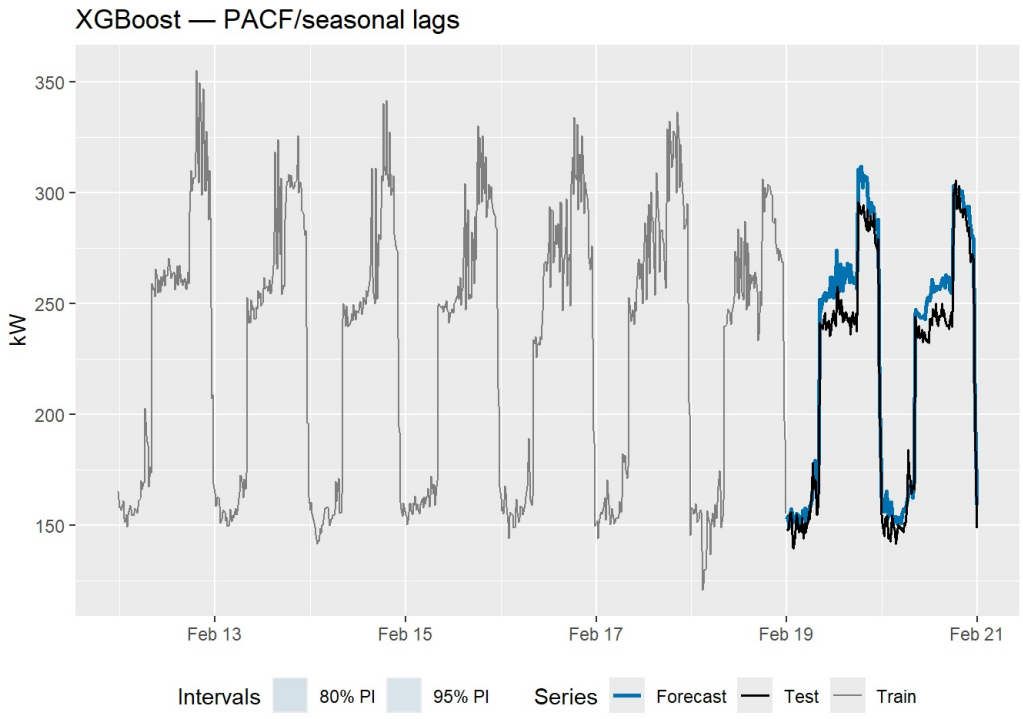
```
## ---- p5-10-xgboost, message=FALSE, warning=FALSE -------------------------
## ---- p5-xgb (fixed) ------------------------------------------------------
suppressPackageStartupMessages(library(xgboost))
set.seed(123)

fit_xgb <- function(dat, p, tag) {
  X <- as.matrix(dat[ , -ncol(dat), drop = FALSE])
  y <- dat$y
  check_design(X, y, p, tag)
  dtrain <- xgb.DMatrix(data = X, label = y, missing = NA_real_)  # be explicit about NA
  params <- list(
    objective = "reg:squarederror",
    max_depth = 8, eta = 0.2,
    subsample = 0.8, colsample_bytree = 0.9,
    nthread = 2
  )
  xgb.train(params = params, data = dtrain, nrounds = 400, verbose = 0)
}

# A (12 lags)
pA <- length(lags_A)
xgb_A <- fit_xgb(dat_A, pA, "XGB-A")
predXGB_A <- seq_forecast(
  lastA, lags_A, length(y_valid),
  function(x) predict(xgb_A, newdata = row_mat(x, pA))
)
accXGB_A <- acc_vec(predXGB_A, y_valid, "XGBoost (12 lags)")

# B (PACF/seasonal)
pB <- length(lags_B)
xgb_B <- fit_xgb(dat_B, pB, "XGB-B")
predXGB_B <- seq_forecast(
  lastB, lags_B, length(y_valid),
  function(x) predict(xgb_B, newdata = row_mat(x, pB))
)
accXGB_B <- acc_vec(predXGB_B, y_valid, "XGBoost (PACF/seasonal lags)")


# Optional plots
plot_fc(df_train, df_valid, list(mean = predXGB_A), "XGBoost — 12 lags")
```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
```




```
plot_fc(df_train, df_valid, list(mean = predXGB_B), "XGBoost — PACF/seasonal lags")
```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
```



svm :

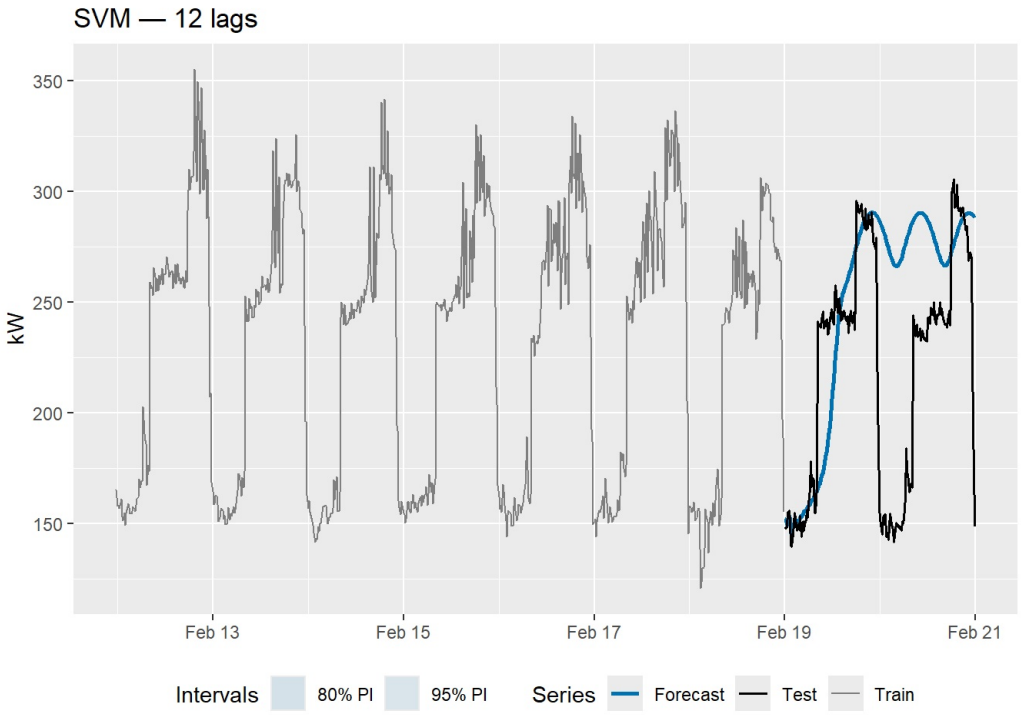
```
## ---- p5-20-svm, message=FALSE, warning=FALSE -----------------------------
library(e1071)
```

```
##
## Attaching package: 'e1071'
```

```
## The following object is masked from 'package:fabletools':
##
##     interpolate
```

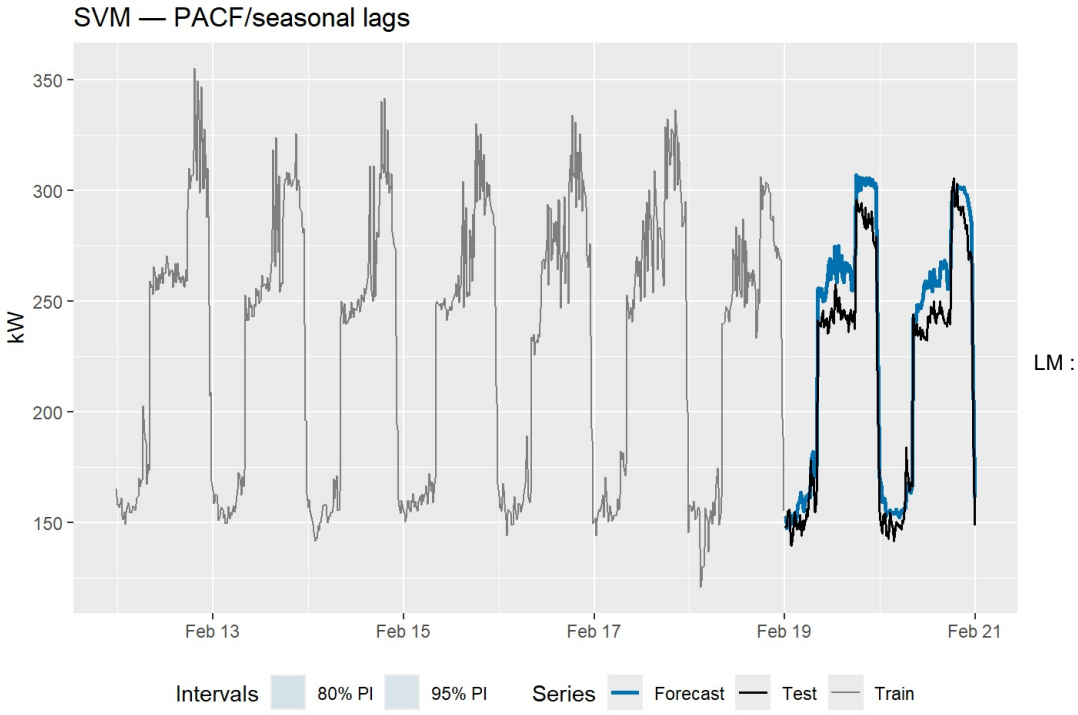
```
# A
# SVM A
svm_A <- svm(x = dat_A[ , -ncol(dat_A)], y = dat_A$y, kernel = "radial")
predSVM_A <- seq_forecast(lastA, lags_A, length(y_valid),
                          function(x) predict(svm_A, newdata = row_mat(x, length(lags_A))))
accSVM_A <- acc_vec(predSVM_A, y_valid, "SVM (12 lags)")

# SVM B
svm_B <- svm(x = dat_B[ , -ncol(dat_B)], y = dat_B$y, kernel = "radial")
predSVM_B <- seq_forecast(lastB, lags_B, length(y_valid),
                          function(x) predict(svm_B, newdata = row_mat(x, length(lags_B))))
accSVM_B <- acc_vec(predSVM_B, y_valid, "SVM (PACF/seasonal lags)")

# Optional plots
plot_fc(df_train, df_valid, list(mean = predSVM_A), "SVM — 12 lags")
```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
```




```
plot_fc(df_train, df_valid, list(mean = predSVM_B), "SVM — PACF/seasonal lags")
```

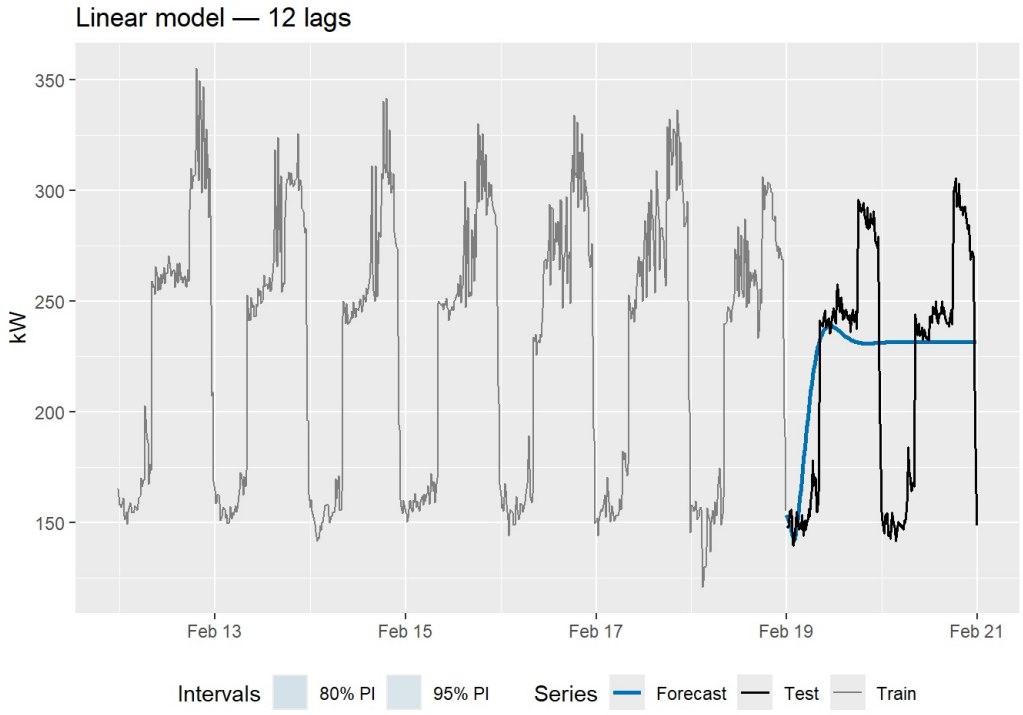
```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
```



LM :

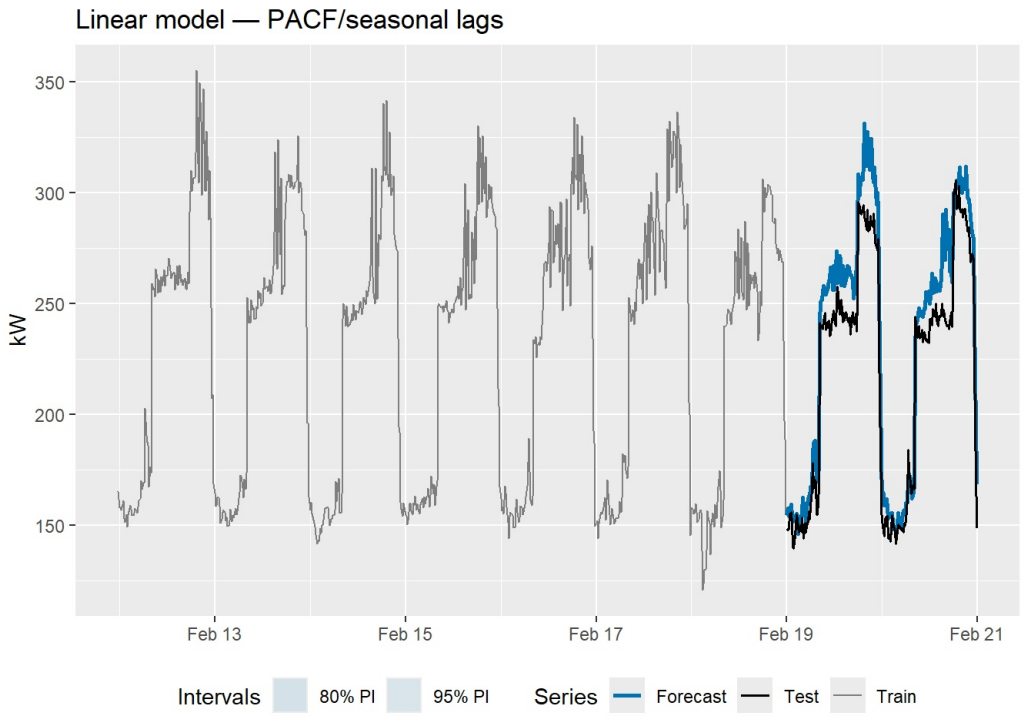
```
## ---- p5-30-lm, message=FALSE, warning=FALSE ------------------------------
# A: 12 lags
lm_A <- lm(y ~ ., data = as.data.frame(dat_A))
predLM_A <- seq_forecast(
  lastA, lags_A, length(y_valid),
  function(x) {
    NX <- as.data.frame(row_mat(x, length(lags_A)))
    colnames(NX) <- paste0("x", lags_A)   # <-- align names with training cols
    predict(lm_A, newdata = NX)
  }
)
accLM_A <- acc_vec(predLM_A, y_valid, "Linear model (12 lags)")

# B: PACF/seasonal lags
lm_B <- lm(y ~ ., data = as.data.frame(dat_B))
predLM_B <- seq_forecast(
  lastB, lags_B, length(y_valid),
  function(x) {
    NX <- as.data.frame(row_mat(x, length(lags_B)))
    colnames(NX) <- paste0("x", lags_B)   # <-- align names with training cols
    predict(lm_B, newdata = NX)
  }
)
accLM_B <- acc_vec(predLM_B, y_valid, "Linear model (PACF/seasonal lags)")


# Optional plots
plot_fc(df_train, df_valid, list(mean = predLM_A), "Linear model — 12 lags")
```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
```




```
plot_fc(df_train, df_valid, list(mean = predLM_B), "Linear model — PACF/seasonal lags")
```

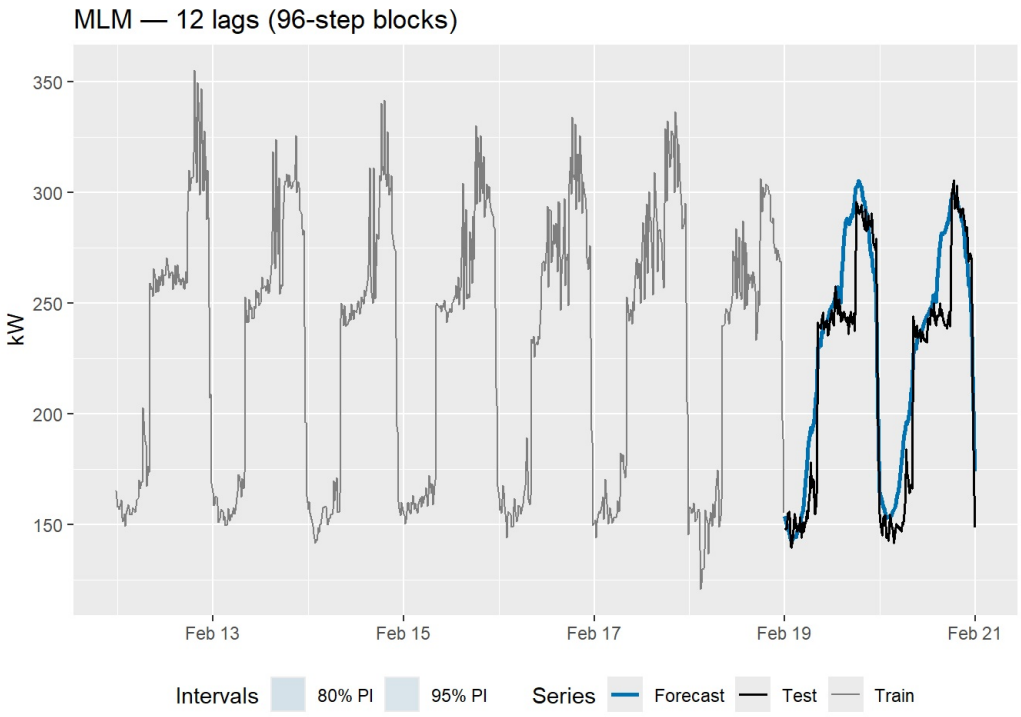
```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
```



Multivariate LM :

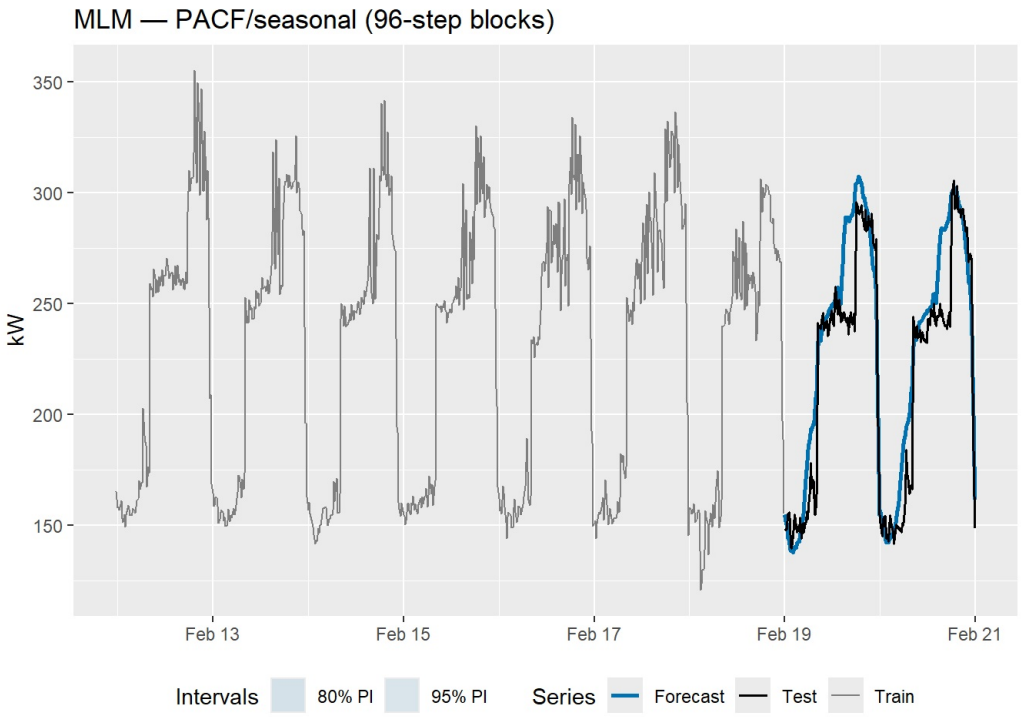
```
## ---- p5-40-mlm-data, message=FALSE, warning=FALSE ------------------------
# Multivariate LM: predict 1 day (H = 96 steps) from lags; then roll forward
build_multi <- function(y, lags, H = 96) {
  yv <- as.numeric(y); L <- max(lags)
  stopifnot(length(yv) > L + H)
  rows <- length(yv) - L - H + 1
  X <- do.call(rbind, lapply(seq_len(rows), function(i) yv[i + L - lags]))
  Y <- do.call(rbind, lapply(seq_len(rows), function(i) yv[(i + L):(i + L + H - 1)]))
  colnames(X) <- paste0("x", lags)
  colnames(Y) <- paste0("y", seq_len(H))
  data.frame(X, Y)
}

H <- 96
multi_A <- build_multi(y_train, lags_A, H)
multi_B <- build_multi(y_train, lags_B, H)

fit_mlm <- function(df, H) {
  ycols <- paste0("y", seq_len(H))
  form <- as.formula(paste("cbind(", paste(ycols, collapse = ","), ") ~ ."))
  lm(form, data = df)
}

mlm_A <- fit_mlm(multi_A, H)
mlm_B <- fit_mlm(multi_B, H)

seq_block_predict <- function(fit, last_y, lags, H, total_h) {
  preds <- numeric(0)
  L <- max(lags)
  buf <- as.numeric(tail(last_y, L))
  steps <- ceiling(total_h / H)
  for (s in seq_len(steps)) {
    x  <- buf[L - lags + 1]
    NX <- as.data.frame(row_mat(x, length(lags)))
    colnames(NX) <- paste0("x", lags)
    yhat_block <- as.numeric(predict(fit, newdata = NX))  # length H
    preds <- c(preds, yhat_block)
    buf <- tail(c(buf, yhat_block), L)  # keep last L in buffer
  }
  preds[seq_len(total_h)]
}

predMLM_A <- seq_block_predict(mlm_A, lastA, lags_A, H, length(y_valid))
predMLM_B <- seq_block_predict(mlm_B, lastB, lags_B, H, length(y_valid))

accMLM_A <- acc_vec(predMLM_A, y_valid, "Multivariate LM (12 lags → 96-step blocks)")
accMLM_B <- acc_vec(predMLM_B, y_valid, "Multivariate LM (PACF/seasonal lags → 96-step blocks)")

# Optional plots
plot_fc(df_train, df_valid, list(mean = predMLM_A), "MLM — 12 lags (96-step blocks)")
```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
```




```
plot_fc(df_train, df_valid, list(mean = predMLM_B), "MLM — PACF/seasonal (96-step blocks)")
```

```
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning
## -Inf
```



Comparison Block ;

```
## ---- p5-55-compare-all (with RF), message=FALSE, warning=FALSE ----------
acc_ml <- dplyr::bind_rows(
  accRF_A,  accRF_B,
  accXGB_A, accXGB_B,
  accSVM_A, accSVM_B,
  accLM_A,  accLM_B,
  accMLM_A, accMLM_B
) %>%
  dplyr::mutate(across(c(RMSE, MAE, MAPE, MASE), as.numeric)) %>%
  dplyr::arrange(MASE)

print(acc_ml)
```

```
## # A tibble: 10 × 5
##    model                                                RMSE   MAE  MAPE  MASE
##    <chr>                                               <dbl> <dbl> <dbl> <dbl>
##  1 XGBoost (PACF/seasonal lags)                         12.0  10.2  4.63  1.55
##  2 Random Forest (PACF/seasonal lags)                   12.9  10.9  4.98  1.66
##  3 SVM (PACF/seasonal lags)                             15.3  13.3  6.00  2.02
##  4 Linear model (PACF/seasonal lags)                    17.5  14.3  6.35  2.18
##  5 Multivariate LM (PACF/seasonal lags → 96-step blocks)  21.6  15.6  7.52  2.37
##  6 Multivariate LM (12 lags → 96-step blocks)           21.6  15.8  7.71  2.40
##  7 XGBoost (12 lags)                                    40.0  30.7 15.9   4.67
##  8 Linear model (12 lags)                               46.2  35.8 18.6   5.44
##  9 Random Forest (12 lags)                              51.0  36.9 20.5   5.61
## 10 SVM (12 lags)                                        60.7  42.5 23.3   6.47
```

```
best_two <- acc_ml$model[1:2]
cat("Best two ML models:", paste(best_two, collapse = " | "), "\n")
```

```
## Best two ML models: XGBoost (PACF/seasonal lags) | Random Forest (PACF/seasonal lags)
```

Here the clear ML winner is:

XGBoost (PACF/seasonal lags) — best MASE/RMSE

Runner-up: Random Forest (PACF/seasonal lags)

```r
## ---- remember-best-ml ----------------------------------------------------
# keep these handy
lags_B <- c(1:8, 96:98, 192, 672)

# define helpers here if they aren't in memory
if (!exists("build_supervised")) {
  build_supervised <- function(y, lags) {
    yv <- as.numeric(y); L <- max(lags)
    nrows <- length(yv) - L
    X <- do.call(rbind, lapply(seq_len(nrows), function(i) yv[i + L - lags]))
    y_next <- yv[(L + 1):length(yv)]
    colnames(X) <- paste0("x", lags)
    data.frame(X, y = y_next)
  }
}
if (!exists("seq_forecast")) {
  seq_forecast <- function(last_y, lags, h, predict_fun) {
    L <- max(lags); buf <- as.numeric(tail(last_y, L)); preds <- numeric(h)
    for (t in seq_len(h)) {
      x <- buf[L - lags + 1]
      preds[t] <- as.numeric(predict_fun(as.numeric(x)))
      buf <- c(buf[-1], preds[t])
    }
    preds
  }
}

# ============= Best ML = XGBoost (PACF/seasonal lags) =====================
ml_best_name <- "XGBoost (PACF/seasonal lags)"

# Re-define ml_best to use the generic predict()
ml_best <- function(y, h = 96, lags = c(1:8, 96:98, 192, 672),
                    nrounds = 400, max_depth = 8, eta = 0.2,
                    subsample = 0.8, colsample_bytree = 0.9, seed = 123) {
  # helpers (same as before)
  build_supervised <- function(y, lags) {
    yv <- as.numeric(y); L <- max(lags); nrows <- length(yv) - L
    X <- do.call(rbind, lapply(seq_len(nrows), function(i) yv[i + L - lags]))
    y_next <- yv[(L + 1):length(yv)]; colnames(X) <- paste0("x", lags)
    data.frame(X, y = y_next)
  }
  seq_forecast <- function(last_y, lags, h, predict_fun) {
    L <- max(lags); buf <- as.numeric(tail(last_y, L)); preds <- numeric(h)
    for (t in seq_len(h)) { x <- buf[L - lags + 1]; preds[t] <- as.numeric(predict_fun(as.numeric(x))); buf <- c(
buf[-1], preds[t]) }
    preds
  }

  dat <- build_supervised(y, lags)
  X <- as.matrix(dat[ , -ncol(dat), drop = FALSE]); ylab <- dat$y
  dtrain <- xgb.DMatrix(X, label = ylab, missing = NA_real_)
  params <- list(objective = "reg:squarederror", max_depth = max_depth, eta = eta,
                 subsample = subsample, colsample_bytree = colsample_bytree, nthread = 2)
  set.seed(seed)
  fit <- xgb.train(params = params, data = dtrain, nrounds = nrounds, verbose = 0)

  last <- tail(as.numeric(y), max(lags))
  preds <- seq_forecast(last, lags, h, function(x) predict(fit, matrix(x, 1)))  # <— no xgboost:: prefix
  list(mean = preds, model = fit)
}

# ============= Runner-up = Random Forest (PACF/seasonal lags) ==============
ml_runner_name <- "Random Forest (PACF/seasonal lags)"

ml_runner <- function(y, h = 96, lags = lags_B, ntree = 500, seed = 123) {
  stopifnot(requireNamespace("randomForest", quietly = TRUE))
  dat <- build_supervised(y, lags)
  X <- as.matrix(dat[ , -ncol(dat), drop = FALSE]); ylab <- dat$y
  set.seed(seed)
  fit <- randomForest::randomForest(x = X, y = ylab, ntree = ntree,
                                    mtry = floor(sqrt(ncol(X))))
  last <- tail(as.numeric(y), max(lags))
  preds <- seq_forecast(last, lags, h, function(x) stats::predict(fit, matrix(x, 1)))
  list(mean = preds, model = fit)
}

cat("Saved best ML:", ml_best_name, "\nRunner-up:", ml_runner_name, "\n")
```

```
## Saved best ML: XGBoost (PACF/seasonal lags)
## Runner-up: Random Forest (PACF/seasonal lags)
```

part 6 : forecast the electricity consumption (kW) for 2/21/2010

Below is a single, clean chunk that:

(Re)creates the split just in case. Uses the saved best models from Parts 3–5 ES: HW multiplicative (damped, STL+ETS) → es_best() SARIMA: ARIMA(1,1,1)(0,1,1)[96] + BoxCox → sarima_best() ML: XGBoost (PACF/seasonal lags) → ml_best() Re-compares them on the same 2-day validation window we used earlier. Picks the smallest MASE as the winner. Refits the winner on all data up to 2010-02-20 23:45 and forecasts 2010-02-21. Saves a CSV with timestamps + 80/95% intervals.

```
## ---- final-bakeoff-and-forecast, message=FALSE, warning=FALSE ------------

# --- 0) Rebuild series & split (safety) ----------------------------------
stopifnot(exists("train_ts_outagefix"))
y_full <- ts(as.numeric(train_ts_outagefix$kW), frequency = 96, start = c(1,1))
h_valid <- 96 * 2
n <- length(y_full)
y_train <- head(y_full, n - h_valid)
y_valid <- tail(y_full, h_valid)

df_train <- head(train_ts_outagefix, n - h_valid)
df_valid <- tail(train_ts_outagefix, h_valid)

# --- 1) Helpers (accuracy, plotting) -------------------------------------
acc_vec <- function(pred, truth, name) {
  pred  <- as.numeric(pred); truth <- as.numeric(truth)
  n <- min(length(pred), length(truth)); e <- truth[1:n] - pred[1:n]
  rmse <- sqrt(mean(e^2)); mae <- mean(abs(e))
  mase_den <- mean(abs(diff(truth[1:n]))), na.rm = TRUE)
  tibble(model = name, RMSE = rmse, MAE = mae,
         MAPE = mean(abs(e)/pmax(truth[1:n], 1e-8))*100, MASE = mae/mase_den)
}

plot_fc <- function(df_train, df_valid, fc, title, last_days = 7) {
  last_train <- max(df_train$timestamp)
  idx_fc <- seq(from = last_train + minutes(15), by = "15 min", length.out = length(fc$mean))
  get_band <- function(obj, level) {
    if (is.null(obj$lower) || is.null(obj$upper)) return(list(lo = rep(NA_real_, length(obj$mean)), hi = rep(NA_r
eal_, length(obj$mean))))
    col <- which(grepl(paste0(level, "%"), colnames(obj$lower))); if (length(col) == 0) col <- 1
    list(lo = as.numeric(obj$lower[, col]), hi = as.numeric(obj$upper[, col]))
  }
  b80 <- get_band(fc, 80); b95 <- get_band(fc, 95)
  df_fc <- tibble::tibble(timestamp = idx_fc, mean = as.numeric(fc$mean),
                          lo80 = b80$lo, hi80 = b80$hi, lo95 = b95$lo, hi95 = b95$hi)
  df_recent <- df_train %>% filter(timestamp >= last_train - days(last_days))
  ggplot() +
    geom_line(data = df_recent, aes(timestamp, kW, color = "Train"), linewidth = 0.5) +
    geom_ribbon(data = df_fc, aes(timestamp, ymin = lo95, ymax = hi95, fill = "95% PI"), alpha = 0.12) +
    geom_ribbon(data = df_fc, aes(timestamp, ymin = lo80, ymax = hi80, fill = "80% PI"), alpha = 0.25) +
    geom_line(data = df_fc, aes(timestamp, mean, color = "Forecast"), linewidth = 0.9) +
    geom_line(data = df_valid, aes(timestamp, kW, color = "Test"), linewidth = 0.7) +
    scale_color_manual(values = c(Train="blue", Forecast="red", Test="green"), name="Series") +
    scale_fill_manual(values = c("80% PI"="#92C5DE", "95% PI"="#56B4E9"), name="Intervals") +
    labs(title = title, x = NULL, y = "kW") + theme(legend.position = "bottom")
}

# --- 2) Ensure "best" model functions exist ------------------------------
# ES winner: HW multiplicative (damped) via STL+ETS
if (!exists("es_best")) {
  es_best <- function(y, h = 96) {
    stlf(y, h = h, method = "ets", s.window = "periodic",
         lambda = 0, biasadj = TRUE, damped = TRUE)
  }
}

# SARIMA winner: ARIMA(1,1,1)(0,1,1)[96] + BoxCox
if (!exists("sarima_best")) {
  sarima_best <- function(y, h = 96) {
    forecast(Arima(y, order = c(1,1,1), seasonal = c(0,1,1), lambda = "auto"), h = h)
  }
}

# ML winner: XGBoost (PACF/seasonal lags)
if (!exists("ml_best")) {
  if (!requireNamespace("xgboost", quietly = TRUE)) stop("Please install 'xgboost'.")
```

```
  build_supervised <- function(y, lags) {
    yv <- as.numeric(y); L <- max(lags); nrows <- length(yv) - L
    X <- do.call(rbind, lapply(seq_len(nrows), function(i) yv[i + L - lags]))
    y_next <- yv[(L + 1):length(yv)]; colnames(X) <- paste0("x", lags)
    data.frame(X, y = y_next)
  }
  seq_forecast <- function(last_y, lags, h, predict_fun) {
    L <- max(lags); buf <- as.numeric(tail(last_y, L)); preds <- numeric(h)
    for (t in seq_len(h)) { x <- buf[L - lags + 1]; preds[t] <- as.numeric(predict_fun(as.numeric(x))); buf <- c(
buf[-1], preds[t]) }
    preds
  }
  ml_best <- function(y, h = 96, lags = c(1:8, 96:98, 192, 672),
                      nrounds = 400, max_depth = 8, eta = 0.2,
                      subsample = 0.8, colsample_bytree = 0.9, seed = 123) {
    dat <- build_supervised(y, lags)
    X <- as.matrix(dat[ , -ncol(dat), drop = FALSE]); ylab <- dat$y
    dtrain <- xgboost::xgb.DMatrix(X, label = ylab, missing = NA_real_)
    params <- list(objective = "reg:squarederror", max_depth = max_depth, eta = eta,
                   subsample = subsample, colsample_bytree = colsample_bytree, nthread = 2)
    set.seed(seed); fit <- xgboost::xgb.train(params = params, data = dtrain, nrounds = nrounds, verbose = 0)
    last <- tail(as.numeric(y), max(lags))
    preds <- seq_forecast(last, lags, h, function(x) xgboost::predict(fit, matrix(x, 1)))
    # returns a "forecast"-like list (no intervals)
    list(mean = preds, model = fit)
  }
}

# --- 3) Compare on the validation window -----------------------------------
fc_es_val    <- es_best(y_train,  h = length(y_valid))
fc_sarima_val<- sarima_best(y_train, h = length(y_valid))
fc_ml_val    <- ml_best(y_train,  h = length(y_valid))

acc_all <- bind_rows(
  acc_vec(fc_es_val$mean,     y_valid, "ES: HW multiplicative (damped, STL+ETS)"),
  acc_vec(fc_sarima_val$mean, y_valid, "SARIMA (1,1,1)(0,1,1)[96] + BoxCox"),
  acc_vec(fc_ml_val$mean,     y_valid, "XGBoost (PACF/seasonal lags)")
) %>% arrange(MASE)

print(acc_all)
```

```
## # A tibble: 3 × 5
##   model                                  RMSE   MAE  MAPE  MASE
##   <chr>                                 <dbl> <dbl> <dbl> <dbl>
## 1 SARIMA (1,1,1)(0,1,1)[96] + BoxCox     12.1  9.78  4.31  1.49
## 2 XGBoost (PACF/seasonal lags)           11.5 10.2   4.81  1.55
## 3 ES: HW multiplicative (damped, STL+ETS)  19.3 15.9   7.03  2.42
```

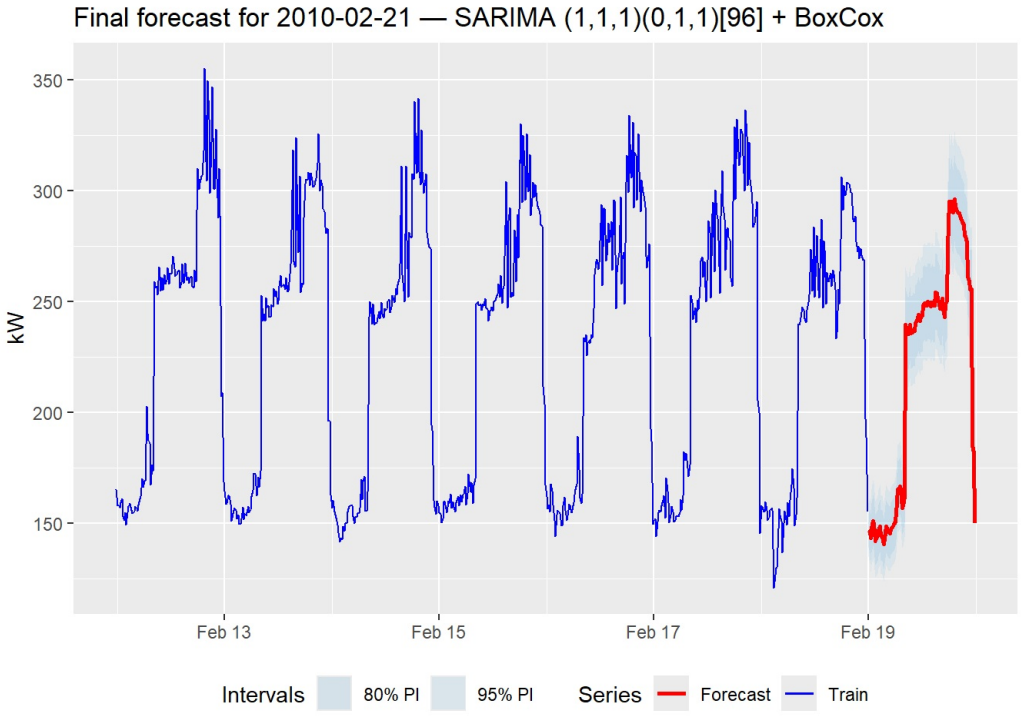
```
best_name <- acc_all$model[1]
cat("Selected winner by validation MASE:", best_name, "\n")
```

```
## Selected winner by validation MASE: SARIMA (1,1,1)(0,1,1)[96] + BoxCox
```

```
# --- 4) Refit winner on ALL data up to 2010-02-20 23:45 and forecast 2/21 --
h <- 96  # 24 hours @ 15-min
fc_final <- switch(best_name,
  "ES: HW multiplicative (damped, STL+ETS)" = es_best(y_full, h = h),
  "SARIMA (1,1,1)(0,1,1)[96] + BoxCox"     = sarima_best(y_full, h = h),
  "XGBoost (PACF/seasonal lags)"           = ml_best(y_full, h = h)
)

# --- 5) Build timestamp grid for 2010-02-21 and save -----------------------
start_fc <- max(df_train$timestamp) + minutes(15)
ts_grid  <- seq(from = start_fc, by = "15 min", length.out = h)

# Safely pull intervals if present
pull_band <- function(obj, level) {
  if (is.null(obj$lower) || is.null(obj$upper)) return(list(lo = rep(NA_real_, length(obj$mean)),
                                                            hi = rep(NA_real_, length(obj$mean))))
  col <- which(grepl(paste0(level, "%"), colnames(obj$lower))); if (length(col) == 0) col <- 1
  list(lo = as.numeric(obj$lower[, col]), hi = as.numeric(obj$upper[, col]))
}
b80 <- pull_band(fc_final, 80); b95 <- pull_band(fc_final, 95)

forecast_df <- tibble::tibble(
  timestamp = ts_grid,
  kW_pred   = as.numeric(fc_final$mean),
  lo80 = b80$lo, hi80 = b80$hi,
  lo95 = b95$lo, hi95 = b95$hi,
  model = best_name
)

# Save to CSV (in your working directory)
out_file <- "forecast_2010-02-21_best.csv"
write.csv(forecast_df, out_file, row.names = FALSE)
cat("Saved forecast to:", out_file, "\n")
```

```
## Saved forecast to: forecast_2010-02-21_best.csv
```

```
# --- 6) Plot the final forecast ---------------------------------------------
plot_fc(df_train, df_valid = df_valid[0, ], fc_final,
        paste0("Final forecast for 2010-02-21 — ", best_name))
```



part 7 : using outdoor temperature.

aligns kW with temp (°C) on the 15-min grid, fits/validates ARIMAX (dynamic regression), ML with covariates (XGBoost, RF, SVM), and VAR, compares them on the same 2-day validation window, picks the best, forecasts 2/21/2010 (96 steps), and saves the forecast.

Assumptions

We still have the outage-fixed table train_ts_outagefix with columns timestamp and kW. We also have a table with temperature, e.g. df_full_temp with columns timestamp and temp_c (your original read). If our object is named differently, just substitute it in the first chunk.

train_ts_outagefix: quarter-hour grid up to 2010-02-20 23:45 with columns timestamp, kW, temp

y_full: ts(kW, frequency = 96)

fc_day: the 2010-02-21 grid (96 rows) with timestamp and temp (future covariate)

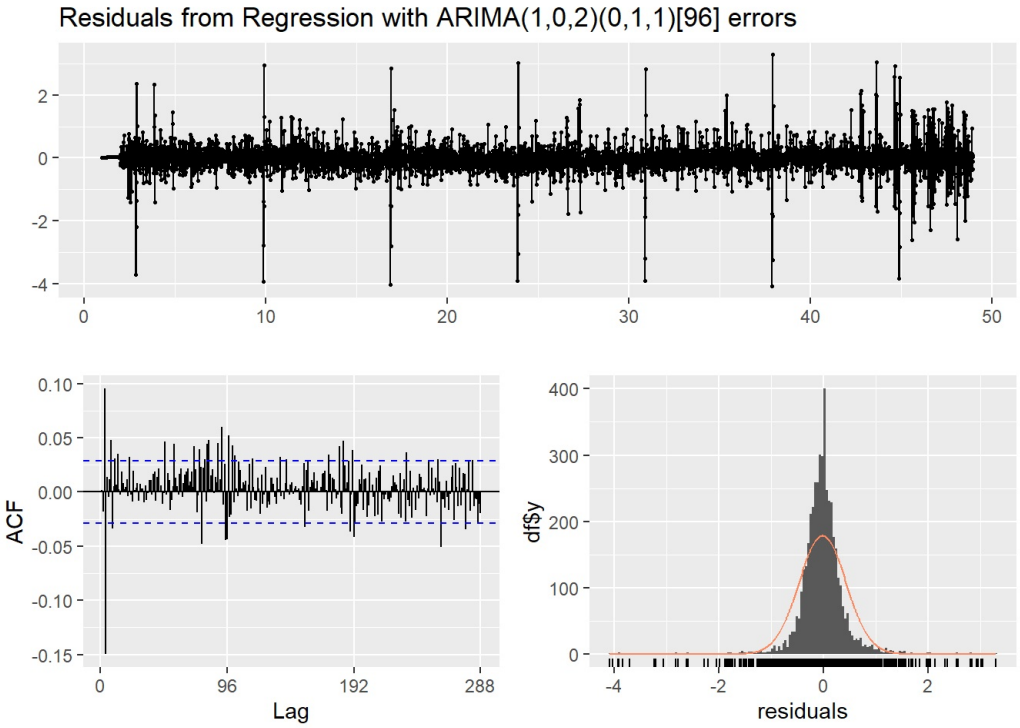
Below are focused chunks for (A) Dynamic Regression = ARIMAX, (B) ML with covariates (XGBoost/RF/SVM), (C) VAR demo, and (D) exporting the final two-column Excel (no-temp vs with-temp).

Dynamic regression (ARIMAX) with known future temperature

```
####============================#
#### Part 7 — WITH temperature #
####============================#

set.seed(123)
suppressPackageStartupMessages({
  library(dplyr); library(lubridate); library(forecast)
  library(tibble); library(writexl)
  library(xgboost); library(randomForest); library(e1071); library(vars)
})

# Safety
stopifnot(exists("train_ts_outagefix"),
          all(c("timestamp","kW","temp") %in% names(train_ts_outagefix)))
stopifnot(exists("fc_day"), all(c("timestamp","temp") %in% names(fc_day)))

freq <- 96

# Plain vectors (avoid window.ts issues)
y_vec   <- as.numeric(train_ts_outagefix$kW)
tmp_vec <- as.numeric(train_ts_outagefix$temp)

# Validation split: last 2 days (192 pts)
h_valid      <- 96 * 2
n            <- length(y_vec)
i_train_end  <- n - h_valid
y_train_v    <- y_vec[1:i_train_end]
y_valid_v    <- y_vec[(i_train_end+1):n]
tmp_train_v  <- tmp_vec[1:i_train_end]
tmp_valid_v  <- tmp_vec[(i_train_end+1):n]

# Future covariate (2010-02-21)
temp_future_v <- as.numeric(fc_day$temp)

# Small accuracy helper (vector in, vector out)
acc_vec <- function(pred, truth, name) {
  pred <- as.numeric(pred); truth <- as.numeric(truth)
  m <- min(length(pred), length(truth)); e <- truth[1:m] - pred[1:m]
  mae <- mean(abs(e))
  tibble(
    model = name,
    RMSE  = sqrt(mean(e^2)),
    MAE   = mae,
    MAPE  = mean(abs(e)/pmax(truth[1:m], 1e-8))*100,
    MASE  = mae / mean(abs(diff(truth[1:m])), na.rm = TRUE)
  )
}
```

Dynamic Regression (ARIMAX) — kW ~ temp lags (0,1,96) + ARIMA errors

```
## ---- 7A-ARIMAX ------------------------------------------------------------
# Build aligned xreg without NA, drop first max(lag)
mk_arimax_design <- function(y, temp, lags = c(0,1,96)) {
  stopifnot(length(y) == length(temp))
  n <- length(temp); L <- max(lags)
  make_lag <- function(k) if (k == 0) temp else c(rep(NA_real_, k), head(temp, n - k))
  Xfull <- do.call(cbind, lapply(lags, make_lag))
  colnames(Xfull) <- paste0("temp_lag", lags)
  idx <- (L + 1):n
  list(y = y[idx], xreg = Xfull[idx, , drop = FALSE])
}

# Train
d_tr <- mk_arimax_design(y_train_v, tmp_train_v, c(0,1,96))
fit_arimax <- Arima(
  ts(d_tr$y, frequency = freq),
  order    = c(1,0,2),
  seasonal = c(0,1,1),
  xreg     = d_tr$xreg,
  lambda   = "auto",
  biasadj  = TRUE
)

# Residual checks
checkresiduals(fit_arimax, test = FALSE)
```




```
checkresiduals(fit_arimax, plot = FALSE)
```

```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(1,0,2)(0,1,1)[96] errors
## Q* = 501.42, df = 188, p-value < 2.2e-16
##
## Model df: 4.   Total lags used: 192
```

```
# Validation design (use known temp on the validation window)
lag0_val  <- tmp_valid_v
lag1_val  <- c(tail(tmp_train_v, 1), head(tmp_valid_v, -1))
lag96_val <- c(tail(tmp_train_v, 96), head(tmp_valid_v, -96))
X_valid   <- cbind(temp_lag0 = lag0_val, temp_lag1 = lag1_val, temp_lag96 = lag96_val)

# Validation forecast + accuracy
fc_arimax_val <- forecast(fit_arimax, h = length(y_valid_v), xreg = X_valid)
acc_arimax    <- acc_vec(fc_arimax_val$mean, y_valid_v, "ARIMAX (temp lags 0,1,96)")
print(acc_arimax)
```

```
## # A tibble: 1 × 5
##   model                      RMSE   MAE  MAPE  MASE
##   <chr>                     <dbl> <dbl> <dbl> <dbl>
## 1 ARIMAX (temp lags 0,1,96)  12.9  10.8  4.84  1.64
```

```
# Final day (2010-02-21) design + forecast
lag0_fut  <- temp_future_v
lag1_fut  <- c(tail(tmp_vec, 1), head(lag0_fut, -1))
lag96_fut <- c(tail(tmp_vec, 96), head(lag0_fut, -96))
X_future  <- cbind(temp_lag0 = lag0_fut, temp_lag1 = lag1_fut, temp_lag96 = lag96_fut)

fc_arimax_final <- forecast(fit_arimax, h = 96, xreg = X_future)
```

ML with covariates — (y-lags + temp-lags)

```
## ---- 7B-ML-XREG ------------------------------------------------------------
lags_y   <- c(1:8, 96:98, 192, 672)  # recent + daily/2-day/weekly
lags_tmp <- c(0, 1, 96)

build_supervised_cov <- function(y, tmp, lags_y, lags_tmp) {
  yv <- as.numeric(y); tv <- as.numeric(tmp)
  L  <- max(c(lags_y, lags_tmp))
  stopifnot(length(yv) == length(tv), length(yv) > L + 1)
  nrows <- length(yv) - L
  Xy <- do.call(rbind, lapply(seq_len(nrows), function(i) yv[i + L - lags_y]))
  Xt <- do.call(rbind, lapply(seq_len(nrows), function(i) tv[i + L - lags_tmp]))
  colnames(Xy) <- paste0("y", lags_y)
  colnames(Xt) <- paste0("t", lags_tmp)
  data.frame(cbind(Xy, Xt), y = yv[(L + 1):length(yv)])
}

row_mat <- function(x, p) matrix(as.numeric(x), nrow = 1, ncol = p, byrow = TRUE)

seq_forecast_cov <- function(last_y, last_tmp, future_tmp, lags_y, lags_tmp, h, predict_fun) {
  Ly <- max(lags_y); Lt <- max(lags_tmp)
  by <- tail(as.numeric(last_y),  Ly)
  bt <- c(tail(as.numeric(last_tmp), Lt), as.numeric(future_tmp))
  preds <- numeric(h)
  for (t in seq_len(h)) {
    x_y <- by[Ly - lags_y + 1]
    idx_end <- Lt + t
    win_t   <- bt[(idx_end - Lt + 1):idx_end]
    x_t <- win_t[Lt - lags_tmp + 1]
    x <- c(x_y, x_t)
    preds[t] <- as.numeric(predict_fun(x))
    by <- c(by[-1], preds[t])
  }
  preds
}

# Training design
dat_cov <- build_supervised_cov(y_train_v, tmp_train_v, lags_y, lags_tmp)
X <- as.matrix(dat_cov[ , -ncol(dat_cov), drop = FALSE]); y <- dat_cov$y; p <- ncol(X)

# XGBoost
dtrain <- xgb.DMatrix(X, label = y, missing = NA_real_)
params <- list(objective = "reg:squarederror", max_depth = 8, eta = 0.2,
               subsample = 0.8, colsample_bytree = 0.9, nthread = 2)
xgb_cov <- xgb.train(params = params, data = dtrain, nrounds = 400, verbose = 0)
pred_xgb_val <- seq_forecast_cov(y_train_v, tmp_train_v, tmp_valid_v, lags_y, lags_tmp, length(y_valid_v),
                                 function(z) predict(xgb_cov, newdata = row_mat(z, p)))
acc_xgb_cov <- acc_vec(pred_xgb_val, y_valid_v, "XGBoost (y-lags + temp-lags)")
# Precompute column means on the TRAINING DESIGN (for imputation)
col_means <- colMeans(X, na.rm = TRUE)

impute_row <- function(x, means) {
  x <- as.numeric(x)
  if (length(x) != length(means)) stop("impute_row: length mismatch")
  ix <- is.na(x)
  if (any(ix)) x[ix] <- means[ix]
  x
}

impute_matrix <- function(M, means) {
  M <- as.matrix(M)
  if (ncol(M) != length(means)) stop("impute_matrix: ncol mismatch")
  for (j in seq_len(ncol(M))) {
```

```
    nas <- is.na(M[, j])
    if (any(nas)) M[nas, j] <- means[j]
  }
  M
}

# ---- Random Forest (NA-safe) ---------------------------------------------
X_rf <- impute_matrix(X, col_means)   # impute training X
set.seed(123)
rf_cov <- randomForest(x = X_rf, y = y, ntree = 500, mtry = floor(sqrt(p)))

pred_rf_val <- seq_forecast_cov(
  last_y = y_train_v, last_tmp = tmp_train_v, future_tmp = tmp_valid_v,
  lags_y = lags_y, lags_tmp = lags_tmp, h = length(y_valid_v),
  predict_fun = function(z) {
    nz <- impute_row(z, col_means)
    predict(rf_cov, newdata = row_mat(nz, p))
  }
)
acc_rf_cov <- acc_vec(pred_rf_val, y_valid_v, "Random Forest (y-lags + temp-lags, NA-safe)")

# ---- SVM (NA-safe) -------------------------------------------------------
# SVM also cannot handle NAs; impute training design first
dat_svm <- as.data.frame(X)
for (j in seq_len(ncol(dat_svm))) {
  nas <- is.na(dat_svm[, j])
  if (any(nas)) dat_svm[nas, j] <- col_means[j]
}
svm_cov <- svm(x = dat_svm, y = y, kernel = "radial")

pred_svm_val <- seq_forecast_cov(
  last_y = y_train_v, last_tmp = tmp_train_v, future_tmp = tmp_valid_v,
  lags_y = lags_y, lags_tmp = lags_tmp, h = length(y_valid_v),
  predict_fun = function(z) {
    nz <- impute_row(z, col_means)
    predict(svm_cov, newdata = as.data.frame(row_mat(nz, p)))
  }
)
acc_svm_cov <- acc_vec(pred_svm_val, y_valid_v, "SVM (y-lags + temp-lags, NA-safe)")

# ---- Rebuild the ML leaderboard (keep your XGB row as-is) ----------------
acc_withtemp_ml <- dplyr::bind_rows(
  acc_xgb_cov, acc_rf_cov, acc_svm_cov
) %>% arrange(MASE)
print(acc_withtemp_ml)
```

```
## # A tibble: 3 × 5
##   model                                        RMSE   MAE  MAPE  MASE
##   <chr>                                       <dbl> <dbl> <dbl> <dbl>
## 1 XGBoost (y-lags + temp-lags)                 10.3  8.76  4.22  1.33
## 2 Random Forest (y-lags + temp-lags, NA-safe)  12.8 11.0   5.04  1.67
## 3 SVM (y-lags + temp-lags, NA-safe)            13.9 11.8   5.28  1.80
```

```
# ---- Final ML forecast for 2010-02-21 using the ML winner -----------------
ml_winner <- acc_withtemp_ml$model[1]
pred_ml_final <- switch(ml_winner,
  "XGBoost (y-lags + temp-lags)" = seq_forecast_cov(
      y_vec, tmp_vec, temp_future_v, lags_y, lags_tmp, 96,
      function(z) predict(xgb_cov, newdata = row_mat(z, p))
  ),
  "Random Forest (y-lags + temp-lags, NA-safe)" = seq_forecast_cov(
      y_vec, tmp_vec, temp_future_v, lags_y, lags_tmp, 96,
      function(z) {
        nz <- impute_row(z, col_means)
        predict(rf_cov, newdata = row_mat(nz, p))
      }
  ),
  "SVM (y-lags + temp-lags, NA-safe)" = seq_forecast_cov(
      y_vec, tmp_vec, temp_future_v, lags_y, lags_tmp, 96,
      function(z) {
        nz <- impute_row(z, col_means)
        predict(svm_cov, newdata = as.data.frame(row_mat(nz, p)))
      }
  )
)
fc_ml_final <- list(mean = as.numeric(pred_ml_final))
best_ml_row <- acc_withtemp_ml %>% dplyr::arrange(MASE) %>% dplyr::slice_head(n = 1)
ml_winner   <- best_ml_row$model[[1]]
```

VAR — selection → fit → diagnostics → forecast → RMSE (class style)

```
## ---- 7C-VAR -------------------------------------------------------------
# Build training matrix (consumption & temperature)
us_app  <- cbind(consumption = y_train_v,  temperature = tmp_train_v)
us_test <- cbind(consumption = y_valid_v,  temperature = tmp_valid_v)

# As ts for vars
us_app_ts <- ts(us_app, frequency = freq)

# Select p (AIC among others)
sel <- VARselect(us_app_ts, lag.max = 8, type = "const")
print(sel$selection)
```

```
## AIC(n)  HQ(n)  SC(n) FPE(n)
##      8      8      5      8
```

```
p_aic <- sel$selection[["AIC(n)"]]

# Fit VAR(p_aic) and VAR(5)
var_aic <- VAR(us_app_ts, p = p_aic, type = "const")
var5    <- VAR(us_app_ts, p = 5,     type = "const")

# Residual serial correlation tests
print(serial.test(var_aic, lags.pt = 10, type = "PT.asymptotic"))
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var_aic
## Chi-squared = 144.73, df = 8, p-value < 2.2e-16
```

```
## $serial
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var_aic
## Chi-squared = 144.73, df = 8, p-value < 2.2e-16
```

```
print(serial.test(var5,    lags.pt = 10, type = "PT.asymptotic"))
```

```
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var5
## Chi-squared = 164.77, df = 20, p-value < 2.2e-16
```

```
## $serial
##
##  Portmanteau Test (asymptotic)
##
## data:  Residuals of VAR object var5
## Chi-squared = 164.77, df = 20, p-value < 2.2e-16
```

```
# Forecast on validation horizon (192)
h <- nrow(us_test)
fcst_aic <- forecast(var_aic, h = h)
fcst5    <- forecast(var5,    h = h)

# Class-style RMSE (consumption channel)
rmse_cons_aic <- sqrt(mean((us_test[, "consumption"] -
                            as.numeric(fcst_aic$forecast$consumption$mean))^2))
rmse_cons_5   <- sqrt(mean((us_test[, "consumption"] -
                            as.numeric(fcst5$forecast$consumption$mean))^2))
print(rmse_cons_aic); print(rmse_cons_5)
```

```
## [1] 47.04634
```

```
## [1] 47.15637
```

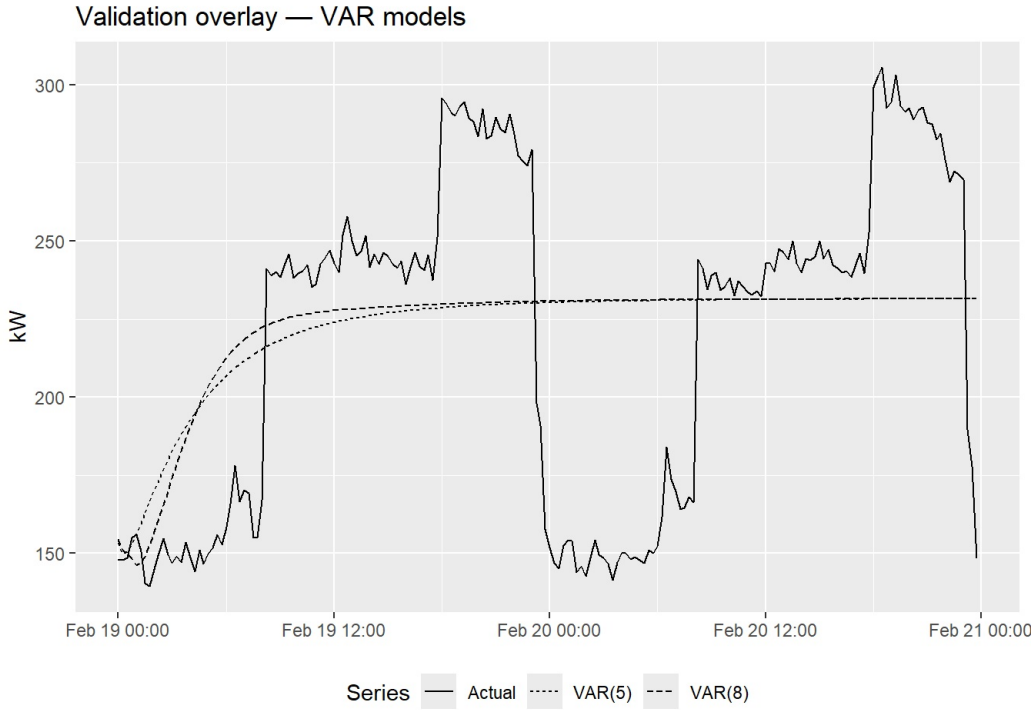
```
## ---- 7C-VAR-ACCURACY-ROWS ------------------------------------------------
# assumes: var_aic, var5, fcst_aic, fcst5, y_valid_v already exist

acc_vec <- function(pred, truth, name) {
  pred <- as.numeric(pred); truth <- as.numeric(truth)
  m <- min(length(pred), length(truth)); e <- truth[1:m] - pred[1:m]
  mae <- mean(abs(e))
  tibble::tibble(
    model = name,
    RMSE  = sqrt(mean(e^2)),
    MAE   = mae,
    MAPE  = mean(abs(e)/pmax(truth[1:m], 1e-8))*100,
    MASE  = mae / mean(abs(diff(truth[1:m])), na.rm = TRUE)
  )
}

# Label with explicit p so it stays stable
p_aic_lab <- if (!is.null(var_aic$p)) var_aic$p else NA_integer_

acc_var_aic <- acc_vec(fcst_aic$forecast$consumption$mean, y_valid_v,
                       paste0("VAR(", p_aic_lab, ")"))
acc_var5    <- acc_vec(fcst5$forecast$consumption$mean,      y_valid_v,
                       "VAR(5)")

# If you already built acc_all7 elsewhere, rebuild including VAR rows:
acc_all7 <- dplyr::bind_rows(
  acc_arimax,               # from 7A
  acc_xgb_cov, acc_rf_cov, acc_svm_cov,   # from 7B
  acc_var_aic, acc_var5
) %>% dplyr::arrange(MASE)

print(acc_all7)
```

```
## # A tibble: 6 × 5
##   model                                        RMSE   MAE  MAPE  MASE
##   <chr>                                       <dbl> <dbl> <dbl> <dbl>
## 1 XGBoost (y-lags + temp-lags)                 10.3  8.76  4.22  1.33
## 2 ARIMAX (temp lags 0,1,96)                    12.9 10.8   4.84  1.64
## 3 Random Forest (y-lags + temp-lags, NA-safe)  12.8 11.0   5.04  1.67
## 4 SVM (y-lags + temp-lags, NA-safe)            13.9 11.8   5.28  1.80
## 5 VAR(8)                                       47.0 38.3  19.9   5.83
## 6 VAR(5)                                       47.2 39.1  20.3   5.95
```

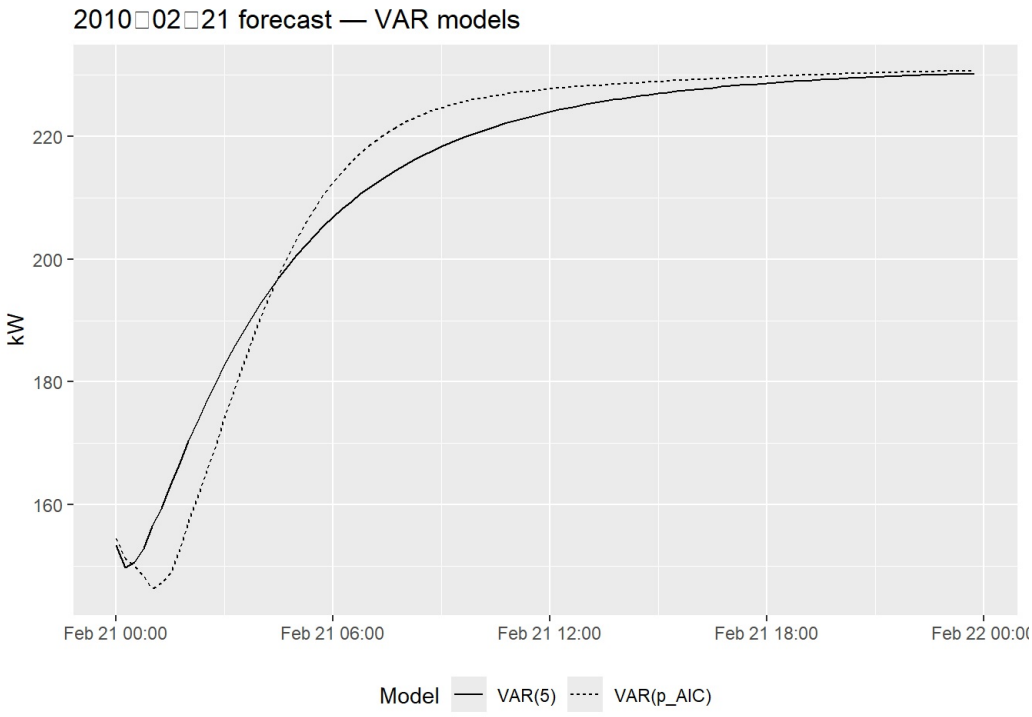
```
withtemp_best_name <- (acc_all7 %>% dplyr::slice_head(n = 1))$model[[1]]
cat("WITH-temperature overall winner: ", withtemp_best_name, "\n", sep = "")
```

```
## WITH-temperature overall winner: XGBoost (y-lags + temp-lags)
```

```
## ---- 7C-VAR-PLOTS ------------------------------------------------------
suppressPackageStartupMessages(library(ggplot2))

# Build a timestamp index for the validation window (last 192 points)
# assumes you have 'train_ts_outagefix$timestamp' (full grid)
ts_all <- train_ts_outagefix$timestamp
val_idx <- tail(ts_all, length(y_valid_v))

# Validation overlay for VAR(p_AIC) and VAR(5)
df_val <- tibble::tibble(
  timestamp = val_idx,
  Actual    = as.numeric(y_valid_v),
  VAR_pAIC  = as.numeric(fcst_aic$forecast$consumption$mean),
  VAR_5     = as.numeric(fcst5$forecast$consumption$mean)
)

ggplot(df_val, aes(timestamp)) +
  geom_line(aes(y = Actual,   linetype = "Actual")) +
  geom_line(aes(y = VAR_pAIC, linetype = paste0("VAR(", p_aic_lab, ")"))) +
  geom_line(aes(y = VAR_5,    linetype = "VAR(5)")) +
  labs(title = "Validation overlay — VAR models",
       x = NULL, y = "kW", linetype = "Series") +
  theme(legend.position = "bottom")
```




```
# Final-day (2010-02-21) VAR forecasts (endogenous; ignores known temp path)
h <- 96
fc_var_aic_final <- forecast(var_aic, h = h)
fc_var5_final    <- forecast(var5,    h = h)

day_idx <- seq(as.POSIXct("2010-02-21 00:00:00", tz="UTC"), by="15 min", length.out=h)
df_final_var <- tibble::tibble(
  timestamp = day_idx,
  `VAR(p_AIC)` = as.numeric(fc_var_aic_final$forecast$consumption$mean),
  `VAR(5)`     = as.numeric(fc_var5_final$forecast$consumption$mean)
)

ggplot(df_final_var, aes(timestamp)) +
  geom_line(aes(y = `VAR(p_AIC)`, linetype = "VAR(p_AIC)")) +
  geom_line(aes(y = `VAR(5)`,     linetype = "VAR(5)")) +
  labs(title = "2010-02-21 forecast — VAR models",
       x = NULL, y = "kW", linetype = "Model") +
  theme(legend.position = "bottom")
```



Compare ARIMAX vs ML (with temp) and export the required Excel (two columns, 96 rows)

```
## ---- 7D-COMPARE-EXPORT ---------------------------------------------------
# 1) Pick the ML winner row robustly
best_ml_row <- acc_withtemp_ml %>% dplyr::arrange(MASE) %>% dplyr::slice_head(n = 1)

# 2) Build ARIMAX vs ML leaderboard and choose a winner (by MASE on kW)
acc_withtemp <- dplyr::bind_rows(acc_arimax, best_ml_row) %>% dplyr::arrange(MASE)
print(acc_withtemp)
```

```
## # A tibble: 2 × 5
##   model                          RMSE   MAE  MAPE  MASE
##   <chr>                         <dbl> <dbl> <dbl> <dbl>
## 1 XGBoost (y-lags + temp-lags)  10.3  8.76  4.22  1.33
## 2 ARIMAX (temp lags 0,1,96)     12.9 10.8   4.84  1.64
```

```
withtemp_winner <- acc_withtemp$model[[1]]
cat("WITH-temperature winner (validation): ", withtemp_winner, "\n", sep = "")
```

```
## WITH-temperature winner (validation): XGBoost (y-lags + temp-lags)
```

```
# 3) Winner's 2010-02-21 forecast (already computed in earlier chunks)
fc_withtemp <- if (identical(withtemp_winner, "ARIMAX (temp lags 0,1,96)")) fc_arimax_final else fc_ml_final

# 4) NO-temperature forecast from Part 6; if missing, compute a safe SARIMA fallback
if (!exists("fc_final")) {
  message("[Note] fc_final (no-temp) not found; using SARIMA(1,1,1)(0,1,1)[96] + BoxCox fallback.")
  fc_final <- forecast(
    Arima(ts(y_vec, frequency = 96), order = c(1,1,1), seasonal = c(0,1,1), lambda = "auto"),
    h = 96
  )
}

# 5) Build the 2010-02-21 timestamp grid and export exactly 2 columns × 96 rows
start_fc <- as.POSIXct("2010-02-21 00:00:00", tz = "UTC")
idx_21   <- seq(from = start_fc, by = "15 min", length.out = 96)

out_tbl <- tibble::tibble(
  timestamp = idx_21,
  no_temp   = as.numeric(fc_final$mean)[1:96],
  with_temp = as.numeric(fc_withtemp$mean)[1:96]
)

A <- out_tbl %>% dplyr::select(no_temp, with_temp)
stopifnot(nrow(A) == 96, ncol(A) == 2)

write_xlsx(list(A = A), path = "Forecasts_A_noTemp_vs_withTemp.xlsx")
cat("Saved: Forecasts_A_noTemp_vs_withTemp.xlsx (sheet 'A', 96×2)\n")
```

```
## Saved: Forecasts_A_noTemp_vs_withTemp.xlsx (sheet 'A', 96×2)
```

Full leaderboard (ARIMAX, ML, VAR) + persist the winner & a reusable forecaster

```
# ---- utilities available everywhere ----
row_mat <- function(x, p = length(x)) {
  matrix(as.numeric(x), nrow = 1, ncol = p, byrow = TRUE)
}
```

```
## ---- 7E-LEADERBOARD-PICK-PERSIST ----------------------------------------
suppressPackageStartupMessages({library(dplyr); library(tibble); library(vars); library(forecast)})

# Preconditions expected from earlier sections:
# acc_arimax, acc_xgb_cov, acc_rf_cov, acc_svm_cov
# var_aic, var5, y_valid_v
# fc_arimax_final, fc_ml_final
# y_vec, tmp_vec, temp_future_v  (vectors)

# 1) Ensure VAR validation forecasts exist (consumption channel)
if (!exists("fcst_aic")) {
  stopifnot(exists("var_aic"), exists("y_valid_v"))
  fcst_aic <- forecast(var_aic, h = length(y_valid_v))
}
if (!exists("fcst5")) {
  stopifnot(exists("var5"), exists("y_valid_v"))
  fcst5 <- forecast(var5, h = length(y_valid_v))
}

# 2) Compact accuracy helper (vector in, vector out)
acc_vec <- function(pred, truth, name) {
  pred <- as.numeric(pred); truth <- as.numeric(truth)
  m <- min(length(pred), length(truth)); e <- truth[1:m] - pred[1:m]
  mae <- mean(abs(e))
  tibble(model = name,
         RMSE  = sqrt(mean(e^2)),
         MAE   = mae,
         MAPE  = mean(abs(e)/pmax(truth[1:m], 1e-8))*100,
         MASE  = mae / mean(abs(diff(truth[1:m])), na.rm = TRUE))
}

# 3) VAR accuracy rows with explicit labels
p_aic_lab   <- if (!is.null(var_aic$p)) var_aic$p else NA_integer_
var_label   <- paste0("VAR(", p_aic_lab, ")")
acc_var_aic <- acc_vec(fcst_aic$forecast$consumption$mean, y_valid_v, var_label)
acc_var5    <- acc_vec(fcst5$forecast$consumption$mean,      y_valid_v, "VAR(5)")

# 4) Full leaderboard (ARIMAX + ML + VAR)
acc_all7 <- dplyr::bind_rows(
  acc_arimax,
  acc_xgb_cov, acc_rf_cov, acc_svm_cov,
  acc_var_aic, acc_var5
) %>% dplyr::arrange(MASE)

print(acc_all7)
```

```
## # A tibble: 6 × 5
##   model                                        RMSE   MAE  MAPE  MASE
##   <chr>                                       <dbl> <dbl> <dbl> <dbl>
## 1 XGBoost (y-lags + temp-lags)                 10.3  8.76  4.22  1.33
## 2 ARIMAX (temp lags 0,1,96)                    12.9 10.8   4.84  1.64
## 3 Random Forest (y-lags + temp-lags, NA-safe)  12.8 11.0   5.04  1.67
## 4 SVM (y-lags + temp-lags, NA-safe)            13.9 11.8   5.28  1.80
## 5 VAR(8)                                       47.0 38.3  19.9   5.83
## 6 VAR(5)                                       47.2 39.1  20.3   5.95
```

```
best_row <- acc_all7 %>% dplyr::slice_head(n = 1)
withtemp_best_name <- best_row$model[[1]]
cat("WITH-temperature overall winner: ", withtemp_best_name, "\n", sep = "")
```

```
## WITH-temperature overall winner: XGBoost (y-lags + temp-lags)
```

```
# 5) Final-day forecast object for the winner (2010-02-21, h = 96)
h <- 96
# Make VAR finals (endogenous; ignores known future temp)
```

```
fc_var_aic_final <- forecast(var_aic, h = h)
fc_var5_final    <- forecast(var5,    h = h)

if (withtemp_best_name == "ARIMAX (temp lags 0,1,96)") {
  fc_withtemp_final <- fc_arimax_final

} else if (withtemp_best_name %in% c("XGBoost (y-lags + temp-lags)",
                                     "Random Forest (y-lags + temp-lags)",
                                     "SVM (y-lags + temp-lags)")) {
  fc_withtemp_final <- fc_ml_final

} else if (withtemp_best_name == var_label) {
  fc_withtemp_final <- list(mean = as.numeric(fc_var_aic_final$forecast$consumption$mean))

} else if (withtemp_best_name == "VAR(5)") {
  fc_withtemp_final <- list(mean = as.numeric(fc_var5_final$forecast$consumption$mean))

} else {
  stop("Winner model not recognized: ", withtemp_best_name)
}

# 6) Persist the winner + a reusable forecaster for future comparisons
withtemp_selection <- list(
  name      = withtemp_best_name,
  metric    = "MASE on consumption (kW) over last 2 days",
  acc_table = acc_all7
)

withtemp_best <- local({
  winner_name <- withtemp_best_name
  var_label_local <- var_label
  function(y, temp, future_temp, h = 96) {
    freq <- 96

    # Small utilities
    mk_arimax_design <- function(y, temp, lags = c(0,1,96)) {
      n <- length(temp); L <- max(lags)
      make_lag <- function(k) if (k == 0) temp else c(rep(NA_real_, k), head(temp, n - k))
      Xfull <- do.call(cbind, lapply(lags, make_lag))
      colnames(Xfull) <- paste0("temp_lag", lags)
      idx <- (L + 1):n
      list(y = y[idx], xreg = Xfull[idx, , drop = FALSE])
    }
    build_supervised_cov <- function(y, tmp, lags_y, lags_tmp) {
      yv <- as.numeric(y); tv <- as.numeric(tmp)
      L  <- max(c(lags_y, lags_tmp))
      nrows <- length(yv) - L
      Xy <- do.call(rbind, lapply(seq_len(nrows), function(i) yv[i + L - lags_y]))
      Xt <- do.call(rbind, lapply(seq_len(nrows), function(i) tv[i + L - lags_tmp]))
      colnames(Xy) <- paste0("y", lags_y); colnames(Xt) <- paste0("t", lags_tmp)
      data.frame(cbind(Xy, Xt), y = yv[(L + 1):length(yv)])
    }
    row_mat <- function(x, p) matrix(as.numeric(x), 1, p, byrow = TRUE)
    seq_forecast_cov <- function(last_y, last_tmp, future_tmp, lags_y, lags_tmp, h, predict_fun) {
      Ly <- max(lags_y); Lt <- max(lags_tmp)
      by <- tail(as.numeric(last_y),  Ly)
      bt <- c(tail(as.numeric(last_tmp), Lt), as.numeric(future_tmp))
      preds <- numeric(h)
      for (t in seq_len(h)) {
        x_y <- by[Ly - lags_y + 1]
        idx_end <- Lt + t; win_t <- bt[(idx_end - Lt + 1):idx_end]
        x_t <- win_t[Lt - lags_tmp + 1]
        x <- c(x_y, x_t)
        preds[t] <- as.numeric(predict_fun(x))
        by <- c(by[-1], preds[t])
      }
      preds
    }

    if (winner_name == "ARIMAX (temp lags 0,1,96)") {
      d <- mk_arimax_design(y, temp, c(0,1,96))
      fit <- Arima(ts(d$y, frequency = freq), order = c(1,0,2), seasonal = c(0,1,1),
                   xreg = d$xreg, lambda = "auto", biasadj = TRUE)
      lag0 <- as.numeric(future_temp)
      lag1 <- c(tail(temp, 1), head(lag0, -1))
      lag96<- c(tail(temp, 96), head(lag0, -96))
      Xf   <- cbind(temp_lag0 = lag0, temp_lag1 = lag1, temp_lag96 = lag96)
      forecast(fit, h = h, xreg = Xf)
```

```
    } else if (winner_name %in% c("XGBoost (y-lags + temp-lags)",
                                  "Random Forest (y-lags + temp-lags)",
                                  "SVM (y-lags + temp-lags)")) {
      lags_y   <- c(1:8, 96:98, 192, 672); lags_tmp <- c(0,1,96)
      dat <- build_supervised_cov(y, temp, lags_y, lags_tmp)
      X <- as.matrix(dat[ , -ncol(dat), drop = FALSE]); yy <- dat$y; p <- ncol(X)

      if (winner_name == "XGBoost (y-lags + temp-lags)") {
        dtrain <- xgboost::xgb.DMatrix(X, label = yy, missing = NA_real_)
        fit <- xgboost::xgb.train(params = list(objective="reg:squarederror", max_depth=8, eta=0.2,
                                                subsample=0.8, colsample_bytree=0.9, nthread=2),
                                  data = dtrain, nrounds = 400, verbose = 0)
        preds <- seq_forecast_cov(y, temp, future_temp, lags_y, lags_tmp, h,
                                  function(z) xgboost::predict(fit, newdata = row_mat(z, p)))

      } else if (winner_name == "Random Forest (y-lags + temp-lags)") {
        # NA-safe RF
        col_means <- colMeans(X, na.rm = TRUE)
        X_rf <- X
        for (j in seq_len(ncol(X_rf))) { na <- is.na(X_rf[,j]); if (any(na)) X_rf[na, j] <- col_means[j] }
        fit <- randomForest::randomForest(x = X_rf, y = yy, ntree = 500, mtry = floor(sqrt(p)))
        preds <- seq_forecast_cov(y, temp, future_temp, lags_y, lags_tmp, h,
                                  function(z) {
                                    nz <- as.numeric(z); nas <- is.na(nz); if (any(nas)) nz[nas] <- col_means[nas
]
                                    stats::predict(fit, newdata = row_mat(nz, p))
                                  })

      } else { # SVM (NA-safe)
        col_means <- colMeans(X, na.rm = TRUE)
        X_svm <- as.data.frame(X)
        for (j in seq_len(ncol(X_svm))) { na <- is.na(X_svm[,j]); if (any(na)) X_svm[na, j] <- col_means[j] }
        fit <- e1071::svm(x = X_svm, y = yy, kernel = "radial")
        preds <- seq_forecast_cov(y, temp, future_temp, lags_y, lags_tmp, h,
                                  function(z) {
                                    nz <- as.numeric(z); nas <- is.na(nz); if (any(nas)) nz[nas] <- col_means[nas
]
                                    stats::predict(fit, newdata = as.data.frame(row_mat(nz, p)))
                                  })
      }
      list(mean = as.numeric(preds), model = fit)

    } else {
      # VAR winner (endogenous, ignores known future temp)
      Y2 <- ts(cbind(kW = as.numeric(y), temp = as.numeric(temp)), frequency = 96)
      p <- if (winner_name == var_label_local) var_aic$p else 5
      fit <- VAR(Y2, p = p, type = "const")
      fc  <- forecast(fit, h = h)
      list(mean = as.numeric(fc$forecast$kW$mean), model = fit)
    }
  }
})

saveRDS(withtemp_selection, file = "withtemp_selection.rds")
saveRDS(withtemp_best,      file = "withtemp_best_fn.rds")
cat("Saved: withtemp_selection.rds and withtemp_best_fn.rds\n")
```

```
## Saved: withtemp_selection.rds and withtemp_best_fn.rds
```