Time Series Analysis - Assignment 01

Graduate Programme: Data Analysis & Machine-Statistical Learning

***John Maris**; math1p0004*

April 9, 2024

Part 1 (Statistical analysis of CO2 emissions trends) 30%

Estimate a model with a linear time trend for the CO2 series (468 monthly observations).

- Do we observe differences between intercepts and slopes estimated with least squares versus robust (Huber weights) least squares?
- Compare conventional (i.i.d. assumption, clearly inappropriate) standard errors and associated confidence intervals versus the ones based on Newey-West HAC Covariance Matrix Estimation.
- Repeat the previous analysis for data aggregated at annual levels.

Solution:

```
data(co2) # 468 monthly observations
co2
>>
```

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1959	315.42	316.31	316.50	317.56	318.13	318.00	316.39	314.65	313.68	313.18	314.66	315.43
1960	316.27	316.81	317.42	318.87	319.87	319.43	318.01	315.74	314.00	313.68	314.84	316.03
1961	316.73	317.54	318.38	319.31	320.42	319.61	318.42	316.63	314.83	315.16	315.94	316.85
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1996	362.09	363.29	364.06	364.76	365.45	365.01	363.70	361.54	359.51	359.65	360.80	362.38
1997	363.23	364.06	364.61	366.40	366.84	365.68	364.52	362.57	360.24	360.83	362.49	364.34

Table 1: *CO_2 measurements (in ppm) from 1959 to 1997.*

The CO_2 data depicted above is an R dataset featuring atmospheric CO_2 concentration measurements from the Mauna Loa Observatory, expressed in parts per million (ppm). This dataset is formatted as a time series containing 468 monthly observations from 1959 to 1997. Notably, the dataset incorporates interpolated values for the months of February, March, and April of 1964, where the original data was missing. In the Figure [1] below we visualize the fluctuation of CO_2 concentrations over time.

```
plot(co2, main="Atmospheric CO2 Concentrations at Mauna Loa",
     xlab="Year", ylab="Atmospheric CO2 (ppm)")
grid(lty = 1, col = "gray", lwd = 0.5656)
>>
```

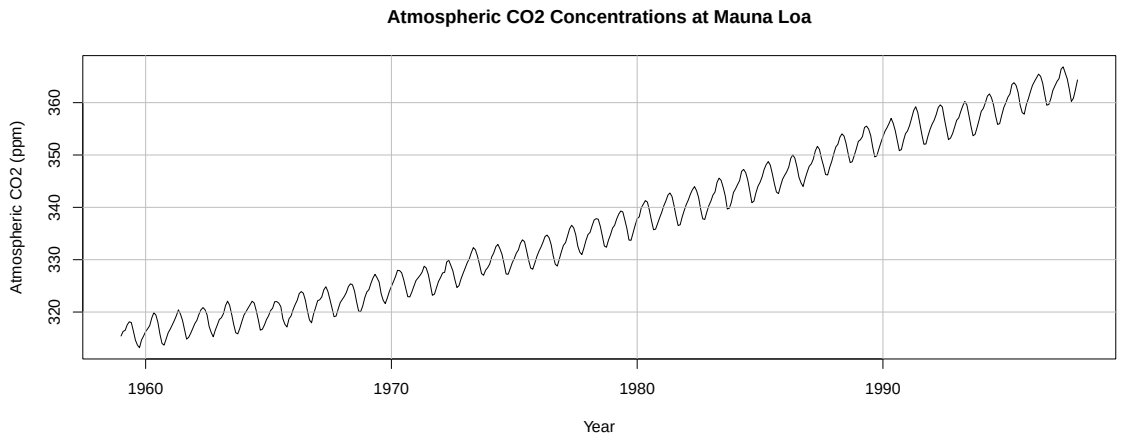

Figure 1: CO_2 *time series.*

The code below compares a model with a linear time trend $\text{CO2} = \beta_0 + \beta_1 \cdot \text{Time} + \varepsilon$, using ordinary least squares with a robust least squares model that applies Huber weights.

```
library(MASS)
co2.time <- as.numeric(time(co2))
co2.data <- as.numeric(co2)
ols.model <- lm(co2.data ~ co2.time) # OLS
robust.model <- rlm(co2.data ~ co2.time, psi = psi.huber) # Robust est. using Huber weights
```

The optimization problem solved by `rlm` can be stated as:

$$\min_{\beta} \sum_{i=1}^{n} \rho(y_i - x_i^T \beta),$$

Where ρ is the Huber loss, defined as:

$$\rho(x) = \begin{cases} \frac{1}{2}x^2 & \text{for } |x| \leq \delta, \\ \delta(|x| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases}$$

In the `rlm` function, the default tuning parameter δ (or k in `rlm()`) used in the Huber weighting function, when you do not specify it explicitly, is set to 1.345. This default value is chosen because it offers a good balance between robustness to outliers and efficiency for estimating the parameters when the errors are normally distributed.

```
summary(ols.model)
>>
Residuals:
    Min      1Q  Median      3Q     Max
-6.0399 -1.9476 -0.0017  1.9113  6.5149

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.250e+03  2.127e+01  -105.8   <2e-16 ***
co2.time     1.308e+00  1.075e-02   121.6   <2e-16 ***
---
Residual standard error: 2.618 on 466 degrees of freedom
Multiple R-squared:  0.9695,  Adjusted  R-squared:  0.9694
F-statistic: 1.479e+04 on 1 and 466 DF,  p-value: < 2.2e-16

summary(robust.model)
>>
Residuals:
     Min       1Q   Median       3Q      Max
-6.01432 -1.93128  0.03843  1.89662  6.61346

Coefficients:
            Value      Std. Error t value
(Intercept) -2258.5951    23.2419   -97.1777
co2.time        1.3119     0.0117   111.6807

Residual standard error: 2.834 on 466 degrees of freedom
```

We observe that the intercept being lower in the robust model (-2258.5951) compared to the OLS model (-2250) suggests that, after adjusting for outliers and influential observations using robust regression, the baseline level of CO2 concentration at the start of the observation period is estimated to be slightly lower.

```
plot(co2, main="Atmospheric CO2 Concentrations at Mauna Loa",
     xlab="Year", ylab="Atmospheric CO2 (ppm)")
grid(lty = 1, col = "gray", lwd = 0.5656)
abline(a = summary(ols.model)$coef[1], b = summary(ols.model)$coef[2], col = "red")
abline(a = summary(robust.model)$coef[1], b = summary(robust.model)$coef[2], col = "darkblue")
legend("topleft", legend=c("OLS", "Robust-huber"), col=c("red", "darkblue"), lty=1, cex=1.3)
```

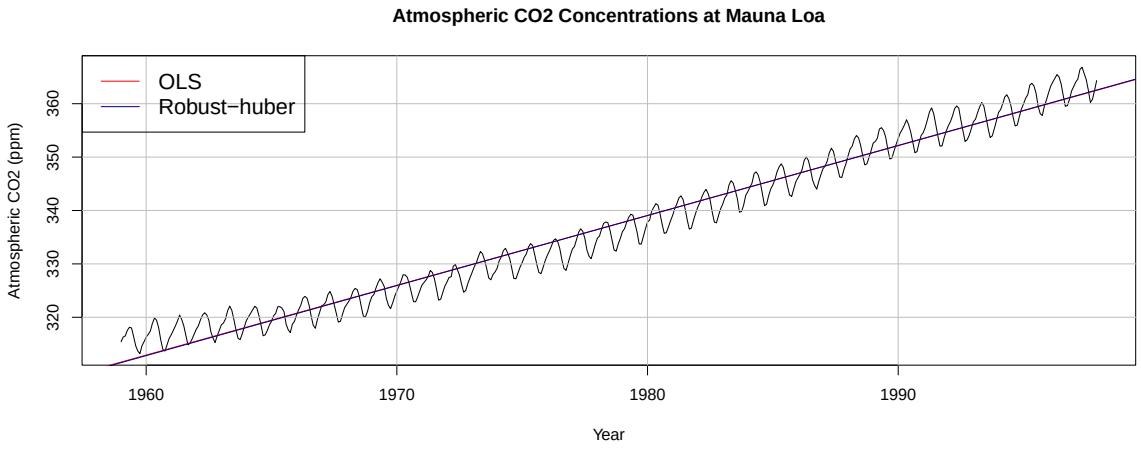

Figure 2: *OLS fit vs Robust fit.*

Now, for conventional standard errors CI (iid assumption are obviously inappropriate):

```
library(lmtest)
confint(ols.model) # conventional standard errors and CI for OLS
>>
                 2.5 %      97.5 %
(Intercept) -2291.566424 -2207.98197
co2.time        1.286373     1.32862
```

For the **robust** regression:

```
conv_CI <- function (model, alpha=0.05) {
    coefficients <-  summary(model)$coefficients[, 1]
    std_errors <-  summary(model)$coefficients[, "Std. Error"]

    df_residual <- length(model$residuals) - length(coefficients) #DF

    t_value <- qt(1 - alpha / 2, df = df_residual)
    lower_bound <- coefficients - t_value * std_errors
    upper_bound <- coefficients + t_value * std_errors

    conf_intervals <- cbind(lower_bound, upper_bound)
    colnames(conf_intervals) <- c("2.5 %", "97.5 %")
    return (conf_intervals)
}
conv_CI(robust.model)
```

Using the **Newey-West** HAC:

```
library(sandwich) # Newey-West standard errors
NW_HAC_CI <- function (model) {
```

```
    nw_se <- NeweyWest(model)
    print(coeftest(model, vcov = nw_se))

    coef <- coef(model)
    stderr <- sqrt(diag(nw_se))

    conf_interval <- t(sapply(1:length(coef), function(i) {
      qnorm(c(0.025, 0.975), mean = coef[i], sd = stderr[i])
    }))

    rownames(conf_interval) <- names(coef)
    colnames(conf_interval) <- c("2.5 %", "97.5 %")
    return (conf_interval)
}

NW_HAC_CI(ols.model)
>>
t test of coefficients:

               Estimate  Std. Error t value  Pr(>|t|)
(Intercept) -2.2498e+03  5.5593e+01 -40.469 < 2.2e-16 ***
co2.time     1.3075e+00  2.8072e-02  46.577 < 2.2e-16 ***
---
                   2.5 %       97.5 %
(Intercept) -2358.734487 -2140.813909
co2.time        1.252477     1.362517
---
```

For the **robust** regression:

```
NW_HAC_CI(robust.model)
>>
z test of coefficients:

               Estimate  Std. Error z value  Pr(>|z|)
(Intercept) -2.2586e+03  7.7461e+01 -29.158 < 2.2e-16 ***
co2.time     1.3119e+00  3.9104e-02  33.550 < 2.2e-16 ***
---
                   2.5 %       97.5 %
(Intercept) -2410.415282 -2106.774865
co2.time        1.235303     1.388589
```

The comparison between conventional iid assumption-based standard errors and those adjusted by the Newey-West HAC Covariance Matrix Estimation clearly shows the Newey-West method's more conservative approach, evidenced by increased standard errors and wider confidence intervals for both OLS and robust models.

Now we will repeat the above analysis using data aggregated at annual levels.

```r
annual_co2 <- aggregate(co2, nfrequency = 1, FUN = mean) # Aggregate to annual levels
annual_co2 <- data.frame(
  year = time(annual_co2),
  co2 = as.numeric(annual_co2)
)
head(annual_co2)
>>
```

year	co2
1959	315.8258
1960	316.7475
1961	317.4850
1962	318.2975
1963	318.8325
1964	319.4625

```r
# Original CO2 time series
plot(co2, main="Atmospheric CO2 Concentrations at Mauna Loa",
     xlab="Year", ylab="Atmospheric CO2 (ppm)", col="blue", type='l')
# vs the annual aggregate of CO2 time series
lines(aggregate(co2, nfrequency = 1, FUN = mean), col="red", lwd=2)
grid(lty = 1, col = "gray", lwd = 0.5656)
legend("topleft", legend=c("Monthly CO2", "Annual Average CO2"),
       col=c("blue", "red"), lty=1, cex=1.1)
```

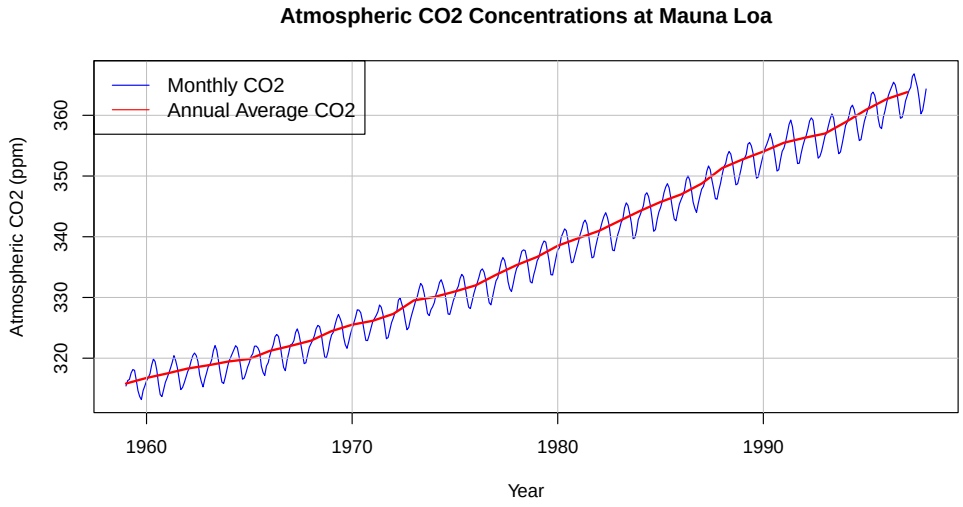

Figure 3: *CO_2 Concentration Trends: Monthly vs. Annual aggregation levels.*

```
# fit the 'naive' models
ols_model.annual <- lm(co2 ~ year, data = annual_co2) # OLS est.
robust_model.annual <- rlm(co2 ~ year, data = annual_co2, psi = psi.huber) # Robust est.

summary(ols_model.annual)
>>
Residuals:
    Min      1Q  Median      3Q     Max
-2.4467 -1.1919 -0.5012  1.2712  3.6718

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.255e+03  4.586e+01  -49.17   <2e-16 ***
year         1.310e+00  2.319e-02   56.52   <2e-16 ***
---
Residual standard error: 1.63 on 37 degrees of freedom
Multiple R-squared: 0.9885,  Adjusted R-squared: 0.9882
F-statistic:  3194 on 1 and 37 DF,  p-value: < 2.2e-16
```

For the **robust** model:

```
summary(robust_model.annual)
>>
Residuals:
    Min      1Q  Median      3Q     Max
-2.3228 -1.1301 -0.3697  1.2085  4.0306

Coefficients:
            Value      Std. Error t value
(Intercept) -2282.5429    45.4975   -50.1685
year            1.3243     0.0230    57.5755

Residual standard error: 1.761 on 37 degrees of freedom

conv_CI(ols_model.annual) # conventional (iid assumptions) CI
>>
                  2.5 %      97.5 %
(Intercept) -2348.043697 -2162.18258
year            1.263517     1.35748

NW_HAC_CI(ols_model.annual) # Newey-West HAC standard errors & CI
>>
t test of coefficients:

             Estimate Std. Error t value Pr(>|t|)
(Intercept) -2255.1131  2900.7685 -0.7774   0.4419
year            1.3105     1.4728  0.8898   0.3793
```

```
                  2.5 %      97.5 %
(Intercept) -7940.514948 3430.288671
year           -1.576119    4.197116
```

Now, for the **robust** regression:

```
conv_CI(robust_model.annual)
>>
                  2.5 %       97.5 %
(Intercept) -2374.729593 -2190.356150
year            1.277712     1.370923

NW_HAC_CI(robust_model.annual)
>>
z test of coefficients:

             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2282.5429  1824.7504 -1.2509   0.2110
year            1.3243     0.9238  1.4336   0.1517

                  2.5 %      97.5 %
(Intercept) -5858.9878957 1293.902153
year           -0.4862941    3.134929
```

Scenarios			**Standard Error (SE)**		**95% Confidence Interval (CI)**	
CO_2 levels	**Model**	**CI type**	$\hat{\beta}_0$	$\hat{\beta}_1$	$\hat{\beta}_0$	$\hat{\beta}_1$
Monthly	OLS	Conventional	21.2676	0.0107494	(-2291.57, -2207.98)	(1.286, 1.329)
		Newey-West	55.593	0.028072	(-2358.73, -2140.81)	(1.252, 1.363)
	Robust	Conventional	23.2419	0.0117473	(-2304.27, -2212.92)	(1.289, 1.335)
		Newey-West	77.461	0.039104	(-2410.42, -2106.77)	(1.235, 1.389)
Annual	OLS	Conventional	45.86	0.02319	(-2348.04, -2162.18)	(1.264, 1.357)
		Newey-West	2900.7685	1.4728	(-7940.51, 3430.29)	(-1.576, 4.197)
	Robust	Conventional	45.4975	0.0230	(-2374.73, -2190.36)	(1.278, 1.371)
		Newey-West	1824.7504	0.9238	(-5858.99, 1293.90)	(-0.486, 3.135)

Table 2: *95% Confidence intervals & standard errors for CO_2 - summary.*

In the summary Table [2] above, we observe significant differences when comparing conventional and Newey-West adjusted standard errors and confidence intervals for CO_2 trends, with the adjustments indicating potential autocorrelation and heteroskedasticity in the time series data.

For the monthly data, the robust model has slightly higher standard errors than the OLS model for both the intercept and the slope, which suggests that the robust model is giving slightly more

weight to the variability in the data, potentially due to its sensitivity to outliers. When we switch to the annual data, this relationship flips; the OLS model displays substantially higher SEs compared to the robust model under the Newey-West adjustment. This could indicate that the annual data have stronger serial correlations or more pronounced heteroskedasticity, which are being more aggressively corrected by the Newey-West adjustments in the OLS model.

Looking at the confidence intervals (CIs), the OLS model under Newey-West adjustments produces much wider intervals for the annual data than for the monthly data, indicating a greater level of uncertainty in the annual estimates. This is especially pronounced for the intercept, where the range is vast, spanning from large negative to positive values. The robust model's CIs are also wider with the Newey-West adjustments, but not to the same extreme as the OLS model, hinting at the robust model's better handling of the anomalies in the data. For the monthly data, while the Newey-West CIs are wider than the conventional ones, suggesting accounting for more uncertainty, they are still within a reasonable range.

The comparison reveals that conventional models, while providing a starting point for analysis, may underestimate the variability and complexity of environmental data.

The Newey-West adjustment, in general, tends to produce wider confidence intervals compared to the conventional approach. This is because it accounts for potential autocorrelation and heteroskedasticity in the error terms, which are not accounted for under standard assumptions. These wider intervals reflect greater uncertainty in the parameter estimates due to the possible correlated nature of the data across time. If we observe that the Newey-West adjustment does not significantly widen the confidence interval, it suggests that either the data does not exhibit strong autocorrelation and heteroskedasticity, or that the sample size is sufficiently large to mitigate their effects. In such cases, we might conclude that the estimates are quite precise, indicating low variance in the regression estimates.

Part 2 (Forecasting of solar irradiance in Hawaii) 40%

- The dataset (`HAWAiiirrad`) reports solar irradiance, measured in Hawaii (GHI). Variable (DOY) stands for day of year whereas time is reported as follows: 62 stands for 6:20, 121 for 12:10, 175 for 17:50, etc.
- We will compute and evaluate day-ahead forecasts for 53 days (`seq(22, 282, by=5)`). Our models should utilize 21 days of training data, observed just before each testing day. Obviously, in reality the length of the training period is treated as a tuning parameter.
- Our work evaluates (at least) 2 regression models [additive seasonal indicators (dummy variables) versus harmonic seasonal models] and 2 estimation methods, namely conventional least squares versus quantile regression.
- Accuracy of point forecasts should be evaluated by examining the distributions ($n = 53$) of daily normalized mean absolute errors (DNMAE). The distributions of Daily Mean Scaled Interval Score should be used to evaluate 90% forecasting intervals based on a) the Gaussian as-

sumption (`predict(object, newdata, se.fit = FALSE, interval = 'prediction')`) and *b*) Quantile Regression.

- We then repeat the previous analysis for data aggregated at an hourly level; compare findings relative to the previous task and discuss how data aggregation influences complexity of harmonic models and serial correlation of the residual series (using appropriate figures to support our arguments), after the daily profiles have been subtracted from the original measurements.

***Solution*:**

The dataset `'HAWAIirrad'` contains solar irradiance measurements from Hawaii, with variables including the day of the year (`'DOY'`), time (in a unique format), and the measured Global Horizontal Irradiance (`'GHI'`), typically in watts per square meter (W/m^2). Here's a brief overview:

Characteristic	
Start Date	March 17
#Average Daily GHI Measurements	70
#Average Monthly GHI Measurements	2,023
Total GHI Measurements	20,230
Mean GHI	458.34
Standard Deviation of GHI	284.24
Minimum GHI	0.00
25th Percentile GHI	221.77
Median GHI	444.89
75th Percentile GHI	678.81
Maximum GHI	1273.66

```
data = read.csv("HAWAIirrad.csv")
convert_time <- function(numbers) { # converts encoded. day-time to actual day-time
  converted_times <- c()
  for (number in numbers) {
    hours <- number %/% 10
    minutes <- (number %% 10) * 10
    converted_times <- c(converted_times, sprintf("%d:%02d", hours, minutes))
  }
  return(converted_times)
}
# Convert time to HH:MM format
data$daytime <- convert_time(data$time)
data$datetime <- as.POSIXct(paste("2023", data$DOY, data$daytime), format = "%Y %j %H:%M")
data
>>
```

Group.1	DOY	time	time2	GHI	daytime	datetime
76062	76	62	76062	0.4719342	6:20	2023-03-17 06:20:00
76063	76	63	76063	2.6663220	6:30	2023-03-17 06:30:00
76064	76	64	76064	5.3131420	6:40	2023-03-17 06:40:00
				⋮		
365174	365	174	365174	26.248630	17:40	2023-12-31 17:40:00
365175	365	175	365175	6.178703	17:50	2023-12-31 17:50:00

The code below visualizes the variation in solar irradiance on random days throughout the year:

```
library(ggplot2)
library(gridExtra) # arranging plots side-by-side
set.seed(5656)
random_days <- sample(unique(data$DOY), 9)
options(repr.plot.width=12.5, repr.plot.height=5.5)
plots <- lapply(random_days, function(day) {
  day_data <- subset(data, DOY == day)
  ggplot(day_data, aes(x=datetime, y=GHI)) + geom_line() +
    ggtitle(paste("Solar irradiance on day:", day)) +
    xlab("Time of day") + ylab("GHI") +
    theme_light(base_size = 12) +
    theme(axis.text.x = element_text(angle = 45, hjust = 1.1),
          axis.title = element_text(size = 14))
})
grid.arrange(grobs = plots, ncol = 3)
>>
```

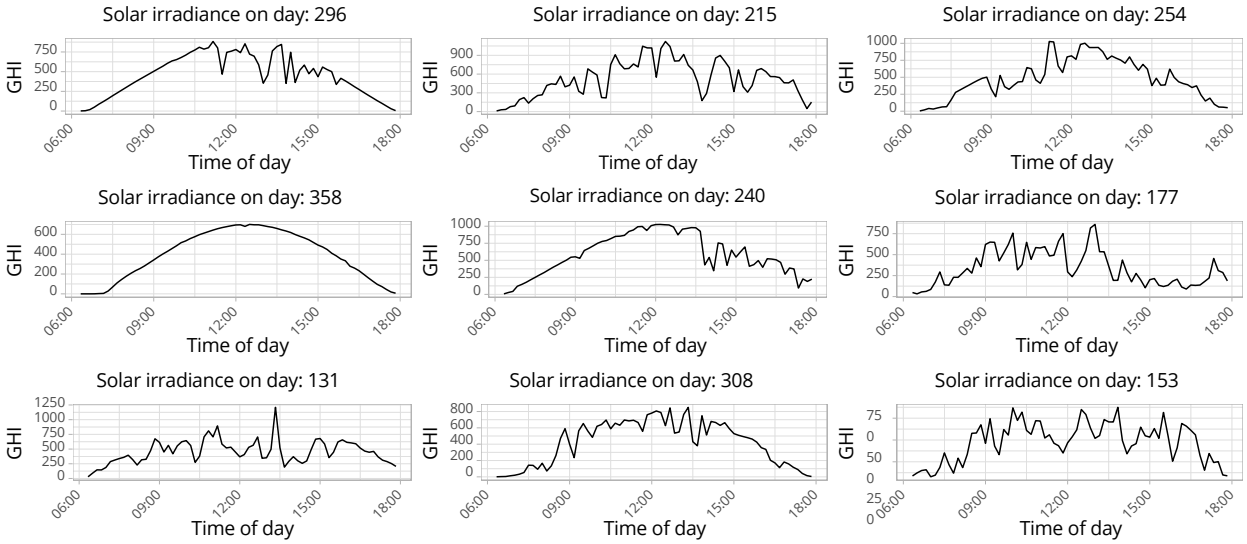

Figure 4: *Daily patterns of Solar Irradiance (GHI) across random days of the year.*

We will have to compute and evaluate day-ahead forecasts for 53 days (`seq(22,282,by=5)`). Our models will utilize 21 days of training data, observed just before each testing day. Let's implement that in R :

```
data$DOY <- as.numeric(data$DOY)
data$time <- as.factor(data$time)
train_sets <- list()
test_sets <- list()
n <- length(seq(22, 282, by=5)) #53
forecast.days <- unique(data$DOY)[seq(22,282,by=5)]

for (i in 1:n) {
    test_sets[[i]] <- subset(data, DOY==forecast.days[i])
    train_sets[[i]] <- subset(data, DOY >= forecast.days[i] - 21 & DOY <= forecast.days[i] - 1)
}
```

Let's visualize the first **21** training days:

```
first_time_window <- unique(train_sets[[1]]$DOY)

plots <- lapply(first_time_window, function(day) {
  day_data <- subset(data, DOY == day)
  ggplot(day_data, aes(x=datetime, y=GHI)) +
    geom_line(color = "darkblue") +
    ggtitle(paste("Solar irradiance on day:", day)) +
    xlab("Time of day") + ylab("GHI") +
    theme_light(base_size = 8) +
    theme(axis.text.x = element_text(angle = 45, hjust = 1.1),
          axis.title = element_text(size = 8))
})
grid.arrange(grobs = plots, ncol = 7)
>>
```

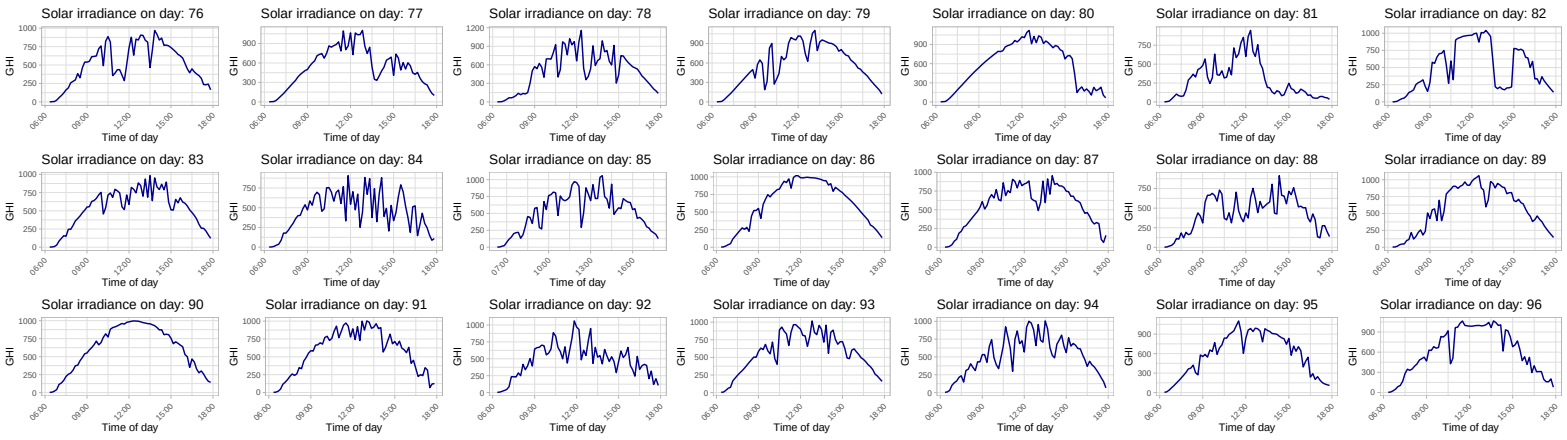

Figure 5: *First training time-window.*

We are now ready to forecast. We will be using the DNMAE metric, defined as:

$$\text{DNMAE}(y, \hat{y}) = \frac{1}{N} \sum \frac{|y - \hat{y}|}{\overline{y}} \,,$$

where y is the actual GHI on the test day, $\hat{y}$ is the predicted GHI and $\overline{y}$ is the mean of the GHI on that forecasting day.

```
library(quantreg)
library(lubridate)
# store DNMAE values for all models
dnmae_lsq_add <- vector("list", length = length(test_sets))
dnmae_lsq_harm <- vector("list", length = length(test_sets))
dnmae_qr_add <- vector("list", length = length(test_sets))
dnmae_qr_harm <- vector("list", length = length(test_sets))
# for the ploting later on
predictions_lsq_add <- list()
predictions_lsq_harm <- list()
predictions_qr_add <- list()
predictions_qr_harm <- list()
actual_values <- list()
freq <- unique(ts(table(as.numeric(data$DOY)))) # 70
# Loop over all forecast windows
time_windows = seq_along(test_sets)
for (i in time_windows) {
  test_data <- test_sets[[i]]
  train_data <- train_sets[[i]]
  # Convert time to a factor for dummy variable regression
  train_data$time_factor <- as.factor(train_data$time)
  test_data$time_factor <- as.factor(test_data$time)
  # data augm. (sins and cosines)
  s <- floor(60/2)
  for(j in 1:s) {
    train_data[[paste0("sin_", j)]] <- sin(2 * pi * j * as.numeric(train_data$time) / freq)
    train_data[[paste0("cos_", j)]] <- cos(2 * pi * j * as.numeric(train_data$time) / freq)
    test_data[[paste0("sin_", j)]] <- sin(2 * pi * j * as.numeric(test_data$time) / freq)
    test_data[[paste0("cos_", j)]] <- cos(2 * pi * j * as.numeric(test_data$time) / freq)
  }
  formula_harm <- as.formula( # For the harmonic model sins + cosines
    paste("GHI ~", paste(paste0("sin_", 1:s), collapse = " + "),
          "+", paste(paste0("cos_", 1:s), collapse = " + ")))
  # Fit models using OLS and quantile reg.
  model_harm_lsq <- lm(formula_harm, data = train_data)
  model_harm_qr <- rq(formula_harm, data = train_data, tau = 0.5)

  predictions_harm_lsq <- predict(model_harm_lsq, newdata = test_data)
  predictions_harm_qr <- predict(model_harm_qr, newdata = test_data)

  model_add_lsq <- lm(GHI ~ time_factor, data = train_data)
  predictions_add_lsq <- predict(model_add_lsq, newdata = test_data)
```

```
  model_add_qr <- rq(GHI ~ time_factor, data = train_data, tau = 0.5)
  predictions_add_qr <- predict(model_add_qr, newdata = test_data)
  # DNMAE for all models
  mean_ghi_test_day <- mean(test_data$GHI)
  dnmae_lsq_add[[i]] <- mean(abs(predictions_add_lsq - test_data$GHI)) / mean_ghi_test_day
  dnmae_lsq_harm[[i]] <- mean(abs(predictions_harm_lsq - test_data$GHI)) / mean_ghi_test_day
  dnmae_qr_add[[i]] <- mean(abs(predictions_add_qr - test_data$GHI)) / mean_ghi_test_day
  dnmae_qr_harm[[i]] <- mean(abs(predictions_harm_qr - test_data$GHI)) / mean_ghi_test_day
  # Store predictions and actual GHI values with corresponding datetime
  actual_values[[i]] <- data.frame(time = test_data$datetime, GHI = test_data$GHI)
  predictions_lsq_add[[i]] <- data.frame(time = test_data$datetime,
                                         GHI = predict(model_add_lsq, newdata = test_data))
  predictions_lsq_harm[[i]] <- data.frame(time = test_data$datetime,
                                          GHI = predict(model_harm_lsq, newdata = test_data))
  predictions_qr_add[[i]] <- data.frame(time = test_data$datetime,
                                        GHI = predict(model_add_qr, newdata = test_data))
  predictions_qr_harm[[i]] <- data.frame(time = test_data$datetime,
                                         GHI = predict(model_harm_qr, newdata = test_data))
}
dnmae_lsq_add_vec <- unlist(dnmae_lsq_add)
dnmae_lsq_harm_vec <- unlist(dnmae_lsq_harm)
dnmae_qr_add_vec <- unlist(dnmae_qr_add)
dnmae_qr_harm_vec <- unlist(dnmae_qr_harm)
```

The following function visualizes the actual Day of Year (DOY) forecasts versus the fitted ones.

```
create_plot <- function(actual_df, pred_add_lsq, pred_harm_lsq, pred_add_qr, pred_harm_qr, day_index) {
  require(ggplot2)
  require(gridExtra)
  ggplot() +
    geom_line(aes(x = time, y = GHI, color = "Actual GHI"), data = actual_df, size=1) +
    geom_line(aes(x = time, y = GHI, color = "Additive LS"), data = pred_add_lsq) +
    geom_line(aes(x = time, y = GHI, color = "Harmonic LS"), data = pred_harm_lsq) +
    geom_line(aes(x = time, y = GHI, color = "Additive QR"), data = pred_add_qr) +
    geom_line(aes(x = time, y = GHI, color = "Harmonic QR"), data = pred_harm_qr) +
    scale_color_manual(values = c("Actual GHI" = "darkblue",
                                  "Additive LS" = "aquamarine3",
                                  "Harmonic LS" = "coral",
                                  "Additive QR" = "purple",
                                  "Harmonic QR" = "orange")) +
    labs(title = paste("Solar irradiance on day:", day_index), x = "Time of day", y = "GHI") +
    theme_minimal()
}
plots <- list()
for (i in 1:53) {
  plots[[i]] <- create_plot(
    actual_values[[i]], predictions_lsq_add[[i]], predictions_lsq_harm[[i]],
    predictions_qr_add[[i]], predictions_qr_harm[[i]], forecast.days[i])
}
do.call("grid.arrange", c(plots, ncol = 3))
>>
```

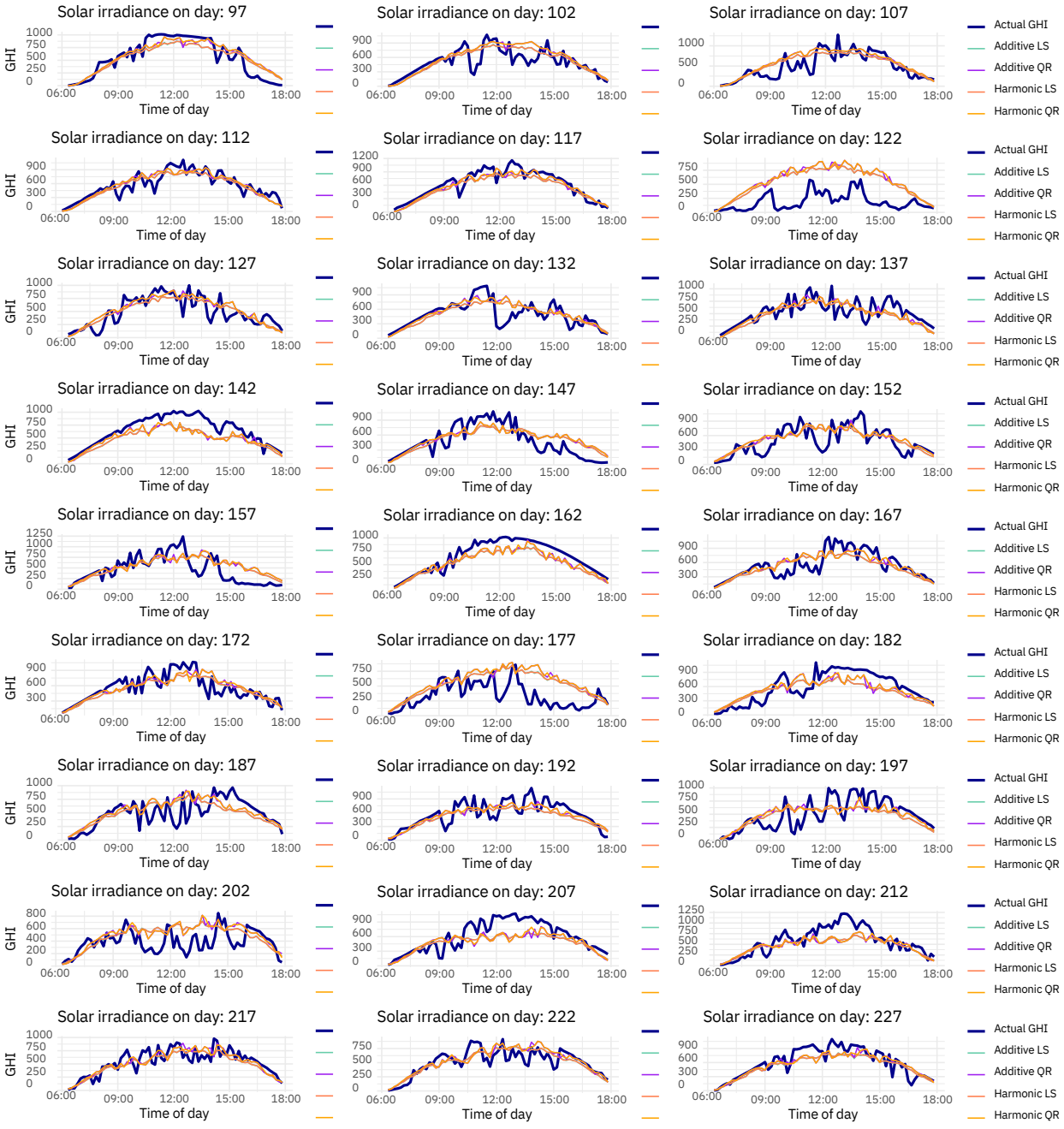

Figure 6: *First 27 actual vs. fitted DOY forecasts.*

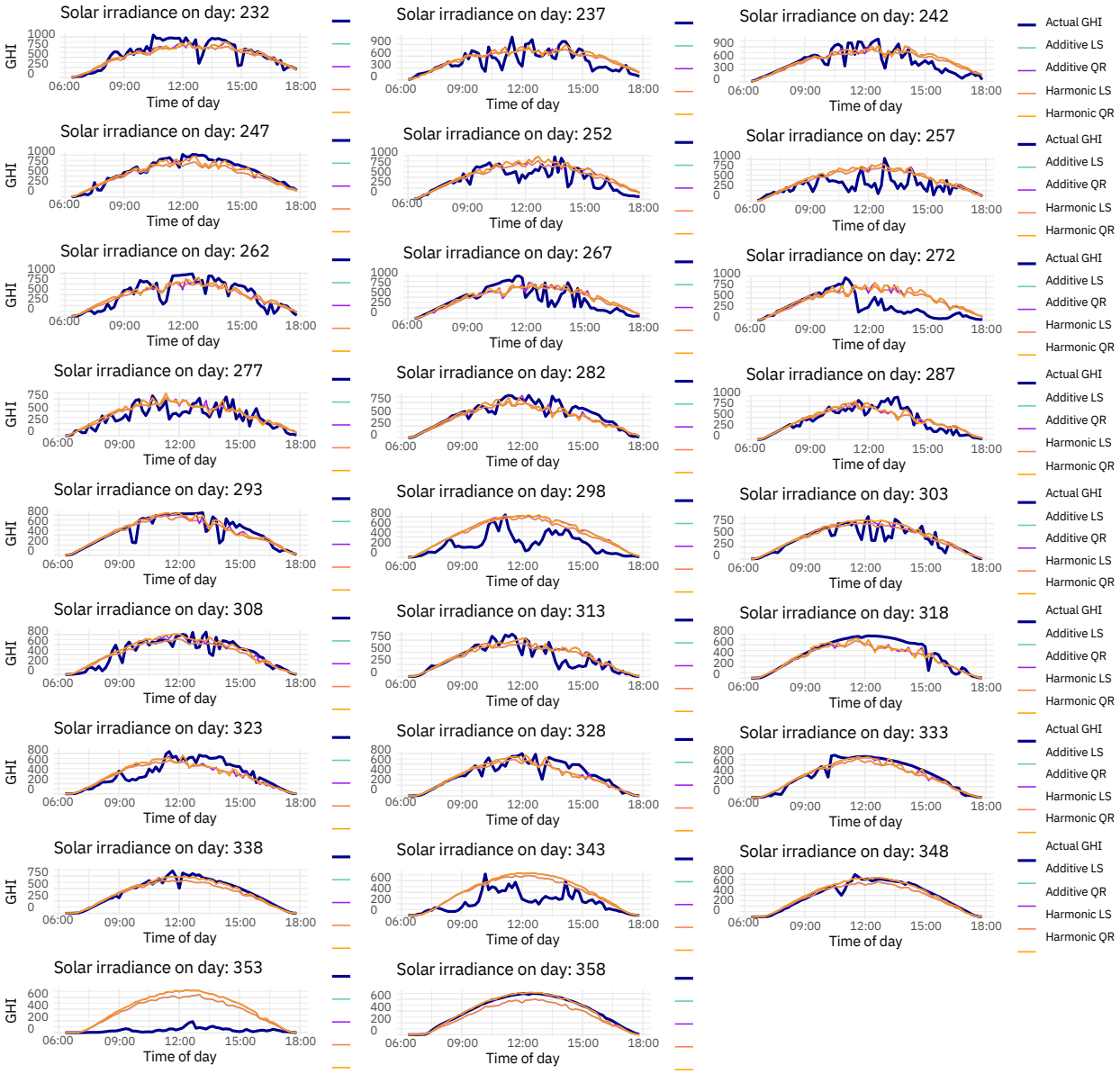

Figure 7: *28th-53th actual vs. fitted DOY forecasts.*

The figures above ([6], [7]) reveal that our forecasting models for days 122, 298, 343 and 354 significantly deviate from the actual time series. However, for the remaining forecasting days, all models appear to effectively capture the patterns of the actual GHI values. Note that, for the

harmonic models, we utilize the approach found in the **Metcalfe's** book:

$$x_t = m_t + \sum_{i=1}^{\lfloor s/2 \rfloor} \left\{ s_i \sin\left(\frac{2\pi i t}{s}\right) + c_i \cos\left(\frac{2\pi i t}{s}\right) \right\} + z_t,$$

where m_t is the trend which includes a parameter for the constant term, and s_i and c_i are unknown parameters.[1] Here, we set $s = 60$ (actual frequency is 70). The following histogram demonstrates the DNMAE values over the 53 forecasting days.

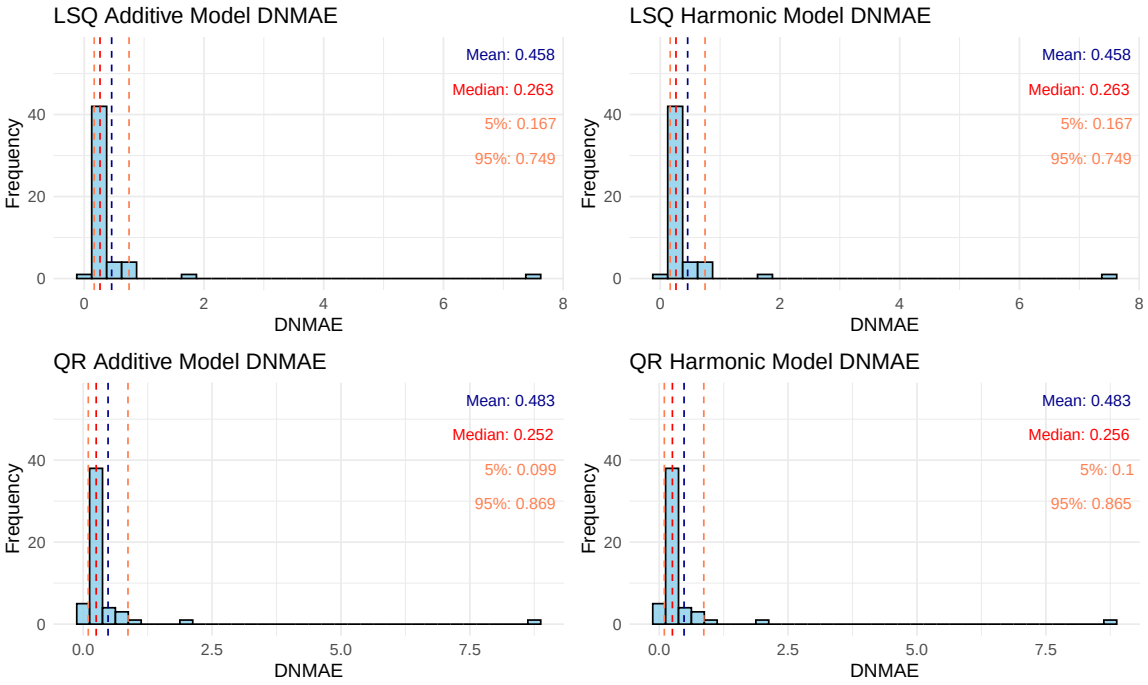

Figure 8: *DNMAE values over the 53 forecasting DOY.*

The box-plots below [9] reveal a small number of outliers (3-4), which likely correspond to the 3-4 days previously identified as being significantly off in our analysis.

[1] At first sight it may seem strange that the harmonic model has cycles of a frequency higher than the seasonal frequency of $1/s$. However, the addition of further harmonics has the effect of perturbing the underlying wave to make it less regular than a standard sine wave of period s.

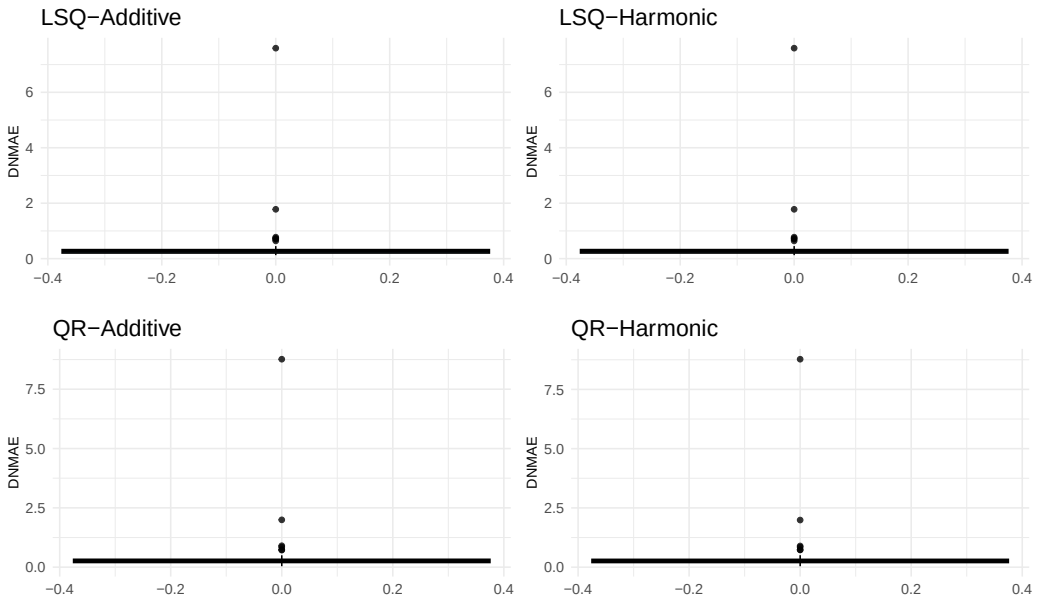

Figure 9: *Box-plots for DNMAE values over the 53 forecasting DOY.*

For the Scaled Mean Interval Score (sMIS) we will utilize the '`smis`'[2] function from the `tsRNN` package. According to M4 Forecasting competition, `smis` is given by:

$$\frac{1}{h}\sum_{t=1}^{h}\left[(U_t - L_t) + \frac{2}{\alpha}(L_t - Y_t)1(Y_t < L_t) + \frac{2}{\alpha}(Y_t - U_t)1(Y_t > U_t)\right]$$

The distributions of Daily Mean Scaled Interval Score should be used to evaluate 90% forecasting intervals based on:

- ◇ Gaussian assumption,
- ◇ Quantile Regression.

For the QR we will adopt different two approaches.

1. Assuming that the regression errors are normally distributed, an approximate 90% prediction interval associated with this forecast is given by

$$\hat{y} \pm z_{value} \cdot \hat{\sigma}_e \sqrt{1 + \frac{1}{T} + \frac{(x - \bar{x})^2}{(T-1)s_x^2}} \qquad (*)$$

with $z_{value} =$ `qnorm(0.95)` ≈ 1.645. where T is the total number of observations, $\bar{x}$ is the mean of the observed x values, s_x is the standard deviation of the observed x values and $\hat{\sigma}_e$ is the standard error of the regression:

[2]Inputs: `data:` time series (only train set), `actual:` actual values, `lower:` lower bound of prediction interval, `upper:` upper bound of prediction interval, `m:` frequency, `level:` level used for prediction interval construction.

$$\hat{\sigma}_e = \sqrt{\frac{1}{T-k-1}\sum_{t=1}^{T} e_t^2} = \frac{1}{\sqrt{T-k-1}} \cdot \|\mathbf{e}\|_2.$$

in the `smis` function, $(*)$ will be our lower and upper predictions.

2. The lower predictions will be based on a QR model with `tau=alpha/2` and for the upper QR with `tau=1-alpha/2`, where `alpha=0.1` (0.05 and 0.95 QR). Let's implement these:

```
library(quantreg)
library(tsRNN)
alpha <- 0.1  # For a 90% prediction interval
z_value <- qnorm(1-alpha/2) # = qnorm(0.95) = 1.645
# Initialize lists to store sMIS values for all models
smis_lsq_add <- vector("list", length = length(test_sets))
smis_qr_add <- vector("list", length = length(test_sets))
smis_lsq_harm <- vector("list", length = length(test_sets))
smis_qr_harm <- vector("list", length = length(test_sets))
# Loop over all forecast windows - calculate sMIS
for (i in seq_along(test_sets)) {
  test_data <- test_sets[[i]]
  train_data <- train_sets[[i]]
  # Ensure the time factor is correctly set
  train_data$time_factor <- as.factor(train_data$time)
  test_data$time_factor <- factor(test_data$time, levels = levels(train_data$time_factor))
# data augm. (sins and cosines)
  s <- floor(60/2)
  for(j in 1:s) {
    train_data[[paste0("sin_", j)]] <- sin(2 * pi * j * as.numeric(train_data$time) / freq)
    train_data[[paste0("cos_", j)]] <- cos(2 * pi * j * as.numeric(train_data$time) / freq)
    test_data[[paste0("sin_", j)]] <- sin(2 * pi * j * as.numeric(test_data$time) / freq)
    test_data[[paste0("cos_", j)]] <- cos(2 * pi * j * as.numeric(test_data$time) / freq)}
  formula_harm_lsq <- as.formula(
    paste("GHI ~", paste(paste0("sin_", 1:s), collapse = " + "),
          "+", paste(paste0("cos_", 1:s), collapse = " + ")))
  model_harm_lsq <- lm(formula_harm_lsq, data = train_data)
  formula_harm_qr <- as.formula(
    paste("GHI ~", paste(paste0("sin_", 1:s), collapse = " + "),
          "+", paste(paste0("cos_", 1:s), collapse = " + ")))
  model_harm_qr <- rq(formula_harm_qr, data = train_data, tau = 0.5)
  formula_add_qr <- GHI ~ time_factor
  model_add_qr <- rq(formula_add_qr, data = train_data, tau = 0.5)
  pred_qr_harm_median <- predict(model_harm_qr, newdata = test_data)
  pred_qr_add_median <- predict(model_add_qr, newdata = test_data)
  # sigma_hat_e for harmonic model
  residuals_harm <- test_data$GHI - pred_qr_harm_median
  T <- nrow(train_data)  # Total observations in the training set
  k_harm <- length(coef(model_harm_qr)) - 1  # Number of predictors in the harmonic model
  sigma_hat_e_harm <- sqrt(sum(residuals_harm^2) / (T - k_harm - 1))
  # for additive model
```

```
  residuals_add <- test_data$GHI - pred_qr_add_median
  k_add <- length(coef(model_add_qr)) - 1  # Number of predictors in the additive model
  sigma_hat_e_add <- sqrt(sum(residuals_add^2) / (T - k_add - 1))
  x_bar <- mean(as.numeric(train_data$time))
  s_x <- sd(as.numeric(train_data$time))
  #  prediction interval widths
  interval_width_harm <- z_value * sigma_hat_e_harm * sqrt(1 + 1/T + ((as.numeric(test_data$time) - x_bar)^2
                         / ((T - 1) * s_x^2)))
  interval_width_add <- z_value * sigma_hat_e_add * sqrt(1 + 1/T + ((as.numeric(test_data$time) - x_bar)^2
                        / ((T - 1) * s_x^2)))
  # apply intervals to median predictions
  pred_qr_harm_lower <- pred_qr_harm_median - interval_width_harm
  pred_qr_harm_upper <- pred_qr_harm_median + interval_width_harm
  pred_qr_add_lower <- pred_qr_add_median - interval_width_add
  pred_qr_add_upper <- pred_qr_add_median + interval_width_add
  # compute sMIS for QR models
  smis_qr_harm[[i]] <- smis(data = train_data$GHI, actual = test_data$GHI,
                            lower = pred_qr_harm_lower, upper = pred_qr_harm_upper,
                            m = freq, level = 1 - alpha)
  smis_qr_add[[i]] <- smis(data = train_data$GHI, actual = test_data$GHI,
                           lower = pred_qr_add_lower, upper = pred_qr_add_upper,
                           m = freq, level = 1 - alpha)
  model_add_lsq <- lm(GHI ~ time_factor, data = train_data)
  # Calculate the prediction intervals for Least Squares
  pred_lsq <- predict(model_add_lsq, newdata = test_data,
                      interval = "prediction",
                      level = 1 - alpha, se.fit=FALSE)
  pred_harm_lsq <- predict(model_harm_lsq, newdata = test_data,
                           interval = "prediction",
                           level = 1 - alpha, se.fit=FALSE)
  # sMIS for Least Squares model
  smis_lsq_add[[i]] <- smis(data = train_data$GHI, actual = test_data$GHI,
                            lower = pred_lsq[, "lwr"], upper = pred_lsq[, "upr"],
                            m = freq, level = 1 - alpha)
  smis_lsq_harm[[i]] <- smis(data = train_data$GHI, actual = test_data$GHI,
                            lower = pred_harm_lsq[, "lwr"], upper = pred_harm_lsq[, "upr"],
                            m = freq, level = 1 - alpha)
}
```

For the second approach, the code remains identical with the exception that the QR is specified as follows:

```
  model_add_qr_lower <- rq(GHI ~ time_factor, data = train_data, tau = alpha / 2)
  model_add_qr_upper <- rq(GHI ~ time_factor, data = train_data, tau = 1 - alpha / 2)
  model_qr_harm_lower <- rq(formula_harm_lsq, data = train_data, tau = alpha / 2)
  model_qr_harm_upper <- rq(formula_harm_lsq, data = train_data, tau = 1 - alpha / 2)
  pred_qr_lower <- predict(model_add_qr_lower, newdata = test_data, se.fit = FALSE)
  pred_qr_upper <- predict(model_add_qr_upper, newdata = test_data, se.fit = FALSE)
  pred_qr_harm_lower <- predict(model_qr_harm_lower, newdata = test_data, se.fit = FALSE)
  pred_qr_harm_upper <- predict(model_qr_harm_upper, newdata = test_data, se.fit = FALSE)
```

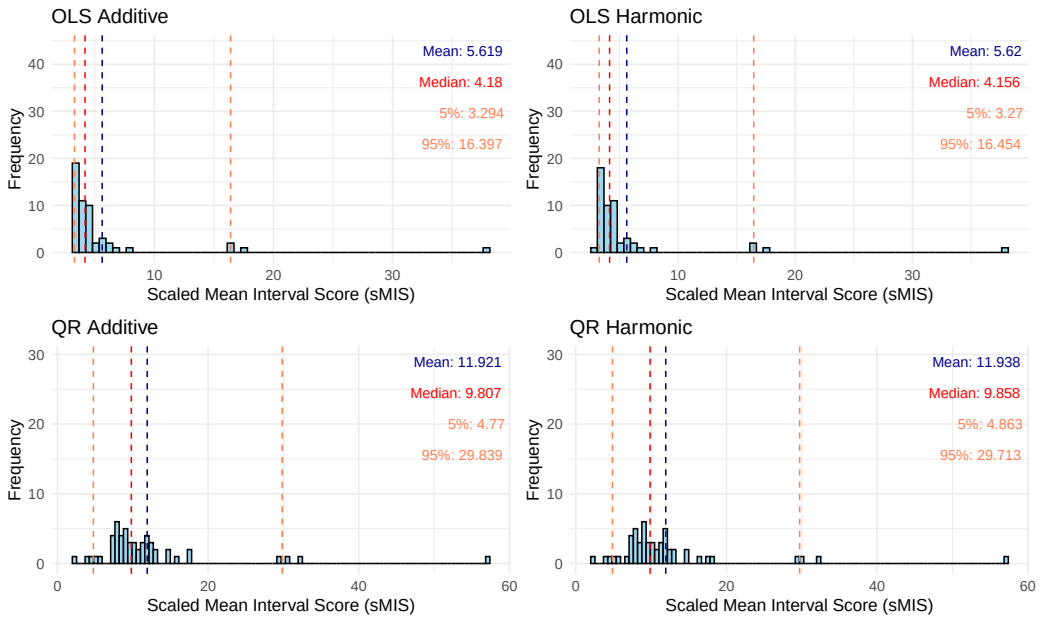

Figure 10: *Scaled Mean Interval Scores for both LS and QR (**first** approach).*

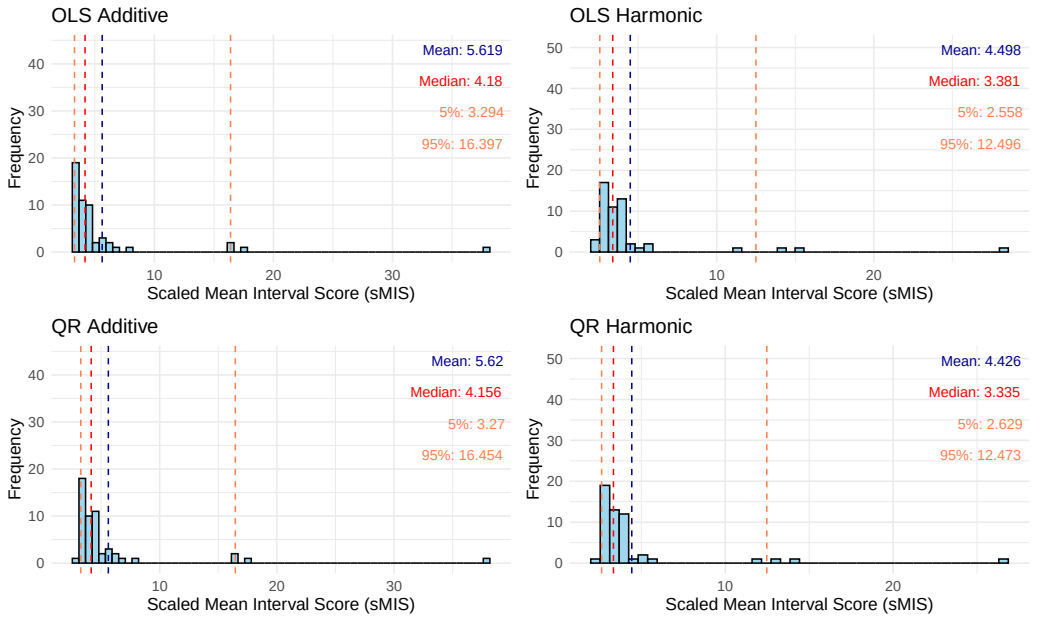

Figure 11: *Scaled Mean Interval Scores for both LS and QR (**second** approach).*

We will work now with aggregated at an hourly level time-series. To do that, we will construct an hourly-level aggregator function:

```
aggregated_data.hourly <- function(df) { # df: time.window
  time_ranges <- list(
    c(62, 65), c(70, 75), c(80, 85), c(90, 95), c(100, 105), c(110, 115),
    c(120, 125), c(130, 135), c(140, 145), c(150, 155), c(160, 165), c(170, 175))
  results <- list()

  # calc. the mean GHI for each DOY and time interval
  for(day in unique(df$DOY)) {
    day_data <- subset(df, DOY == day)
    for(range in time_ranges) {
      time_data <- subset(day_data, time >= range[1] & time <= range[2])
      if(nrow(time_data) > 0) {
        mean_ghi <- mean(time_data$GHI, na.rm = TRUE)
        # determine the actual time hour string
        actual_time_hour <- ifelse(
          range[1] == 62,
          sprintf("6:20-%d:%02d", range[2] %/% 10, (range[2] %% 10) * 6),
          sprintf("%d:%02d-%d:%02d", range[1] %/% 10, (range[1] %% 10) * 6,
                  range[2] %/% 10, (range[2] %% 10) * 6)
        )
        actual_time_hour <- gsub("30", "50", actual_time_hour)  # Ensure ending at :50
        # Determine the encoded time hour (the midpoint of the time range)
        enc_time_hour <- ifelse(range[1] == 62, 6.2, (range[1] + range[2]) / 2 / 10)
        results[[length(results) + 1]] <- list(DOY = day, GHI = mean_ghi,
                                               enc_time_hour = enc_time_hour,
                                               actual_time_hour = actual_time_hour)
      }
    }
  }
  results_df <- do.call(rbind, lapply(results, data.frame))
  rownames(results_df) <- NULL  # Clean up row names
  # Adjust the encoded time hours for subsequent rows to be whole numbers (integers)
  results_df <- transform(results_df, enc_time_hour = ifelse(enc_time_hour != 6.2,
                                                             as.integer(enc_time_hour), enc_time_hour))
  return (results_df)
}
```

We can now construct the new (aggregated) training and forecasting data:

```
train_sets.agg <- list()
test_sets.agg <- list()
n <- length(seq(22, 282, by=5)) #53
forecast.days <- unique(data$DOY)[seq(22,282,by=5)]
for (i in 1:n) {
    train.window_i <- train_sets[[i]]
    train.window_i$time <- as.numeric(as.vector(train.window_i$time))
    train_sets.agg[[i]] <- aggregated_data.hourly(train.window_i)
    test.window_i <- test_sets[[i]]
```

```
    test.window_i$time <- as.numeric(as.vector(test.window_i$time))
    test_sets.agg[[i]] <- aggregated_data.hourly(test.window_i)
    # add the datetimes
    datetimes <- subset(train_sets[[i]],
                        time %in% as.vector(train_sets.agg[[i]]$enc_time_hour *10))$datetime
    train_sets.agg[[i]]$datetime <- datetimes

    datetimes <- subset(test_sets[[i]],
                        time %in% as.vector(test_sets.agg[[i]]$enc_time_hour *10))$datetime
    test_sets.agg[[i]]$datetime <- datetimes
}
```

As we did previously, let's visualize the first **21** hourly-aggregated training DOY:

```
plots <- lapply(first_time_window, function(day) {
  agg_day_data <- subset(train_sets.agg[[1]], DOY == day)
  non_agg_day_data <- subset(train_sets[[1]], DOY == day)
  gg <- ggplot() +
    geom_line(data = non_agg_day_data, aes(x=datetime, y=GHI), color = "darkblue", size = 1, alpha = 0.7) +
    geom_line(data = agg_day_data, aes(x=datetime, y=GHI), color = "firebrick1", size = 1, alpha = 0.7) +
    ggtitle(paste("Solar irradiance on day:", day)) +
    xlab("Time of day") + ylab("GHI") +
    theme_light(base_size = 9) +
    theme(axis.text.x = element_text(angle = 45, hjust = 1.1),
          axis.title = element_text(size = 8)) +
    scale_color_manual(values = c("darkblue", "firebrick1"),
                       labels = c("Non-Aggregated", "Aggregated")) +
    guides(color = guide_legend(title = "Data Type"))
  return(gg)
})
grid.arrange(grobs = plots, ncol = 7)
>>
```

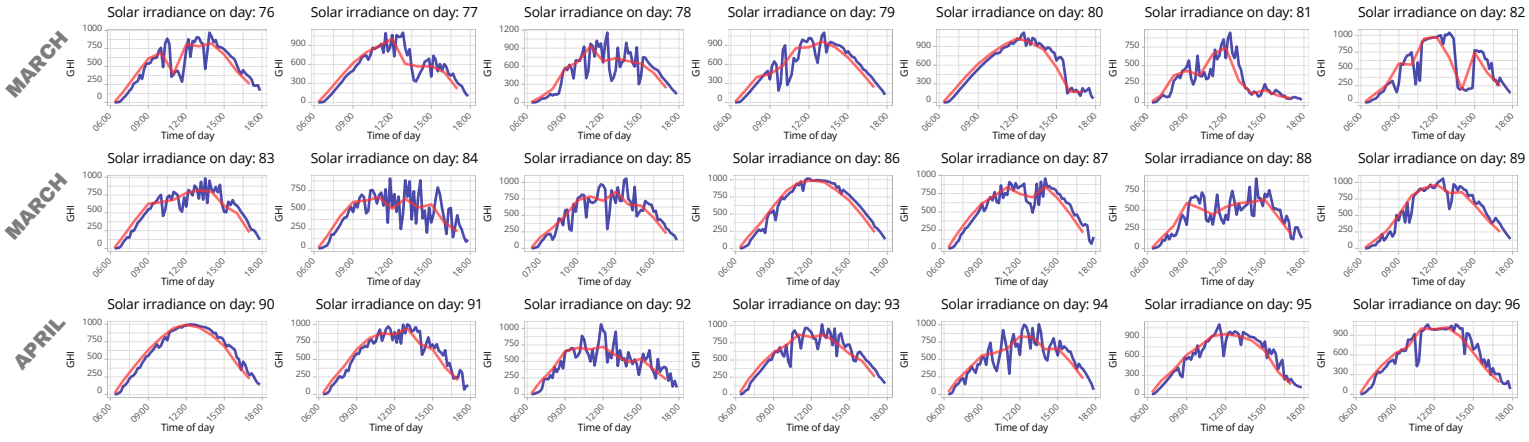

Figure 12: *First training time-window aggregated (at an hourly level) data VS original time series.*

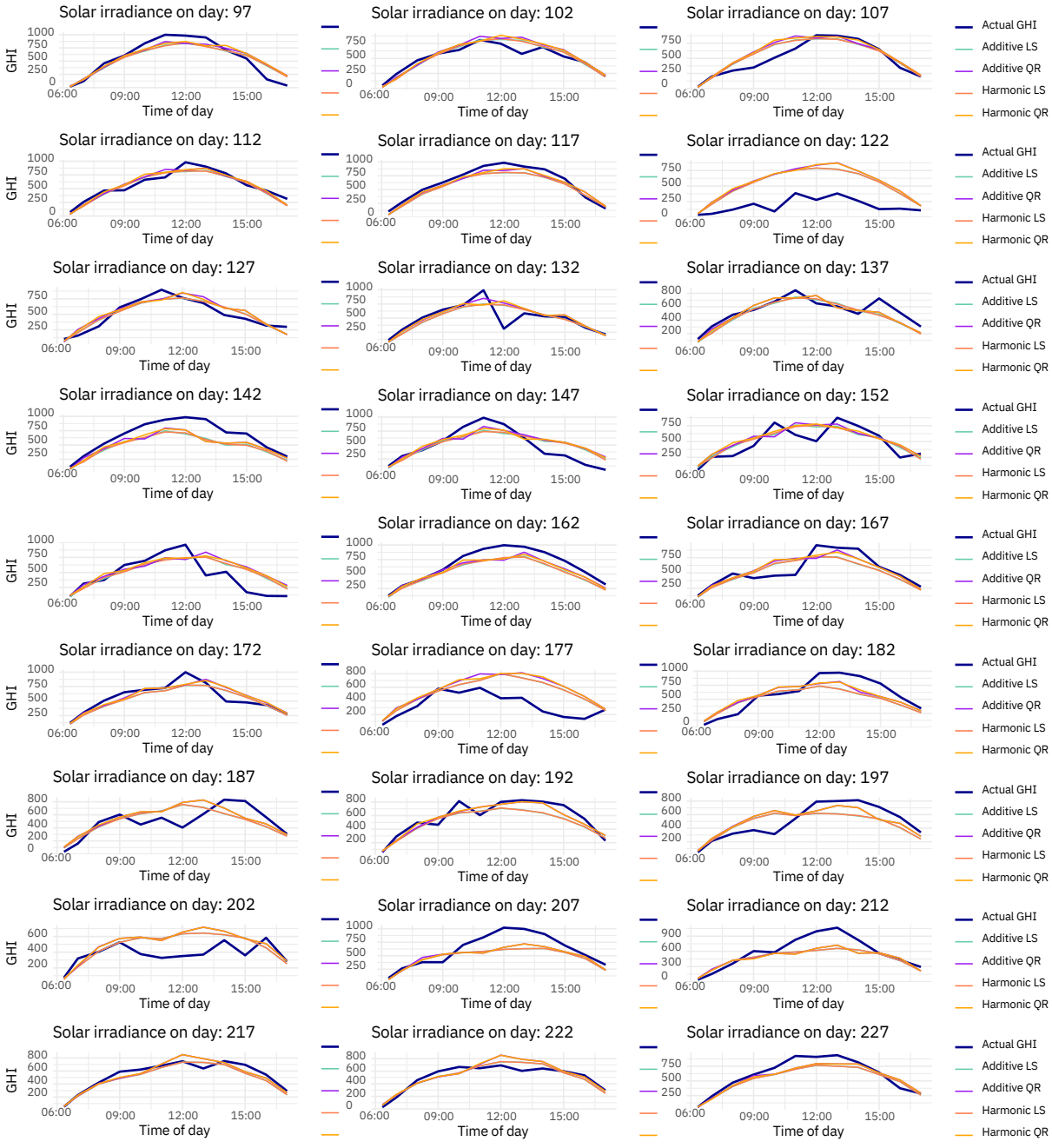

Figure 13: *First 27 actual vs. fitted DOY forecasts (hourly-level aggregated time series).*

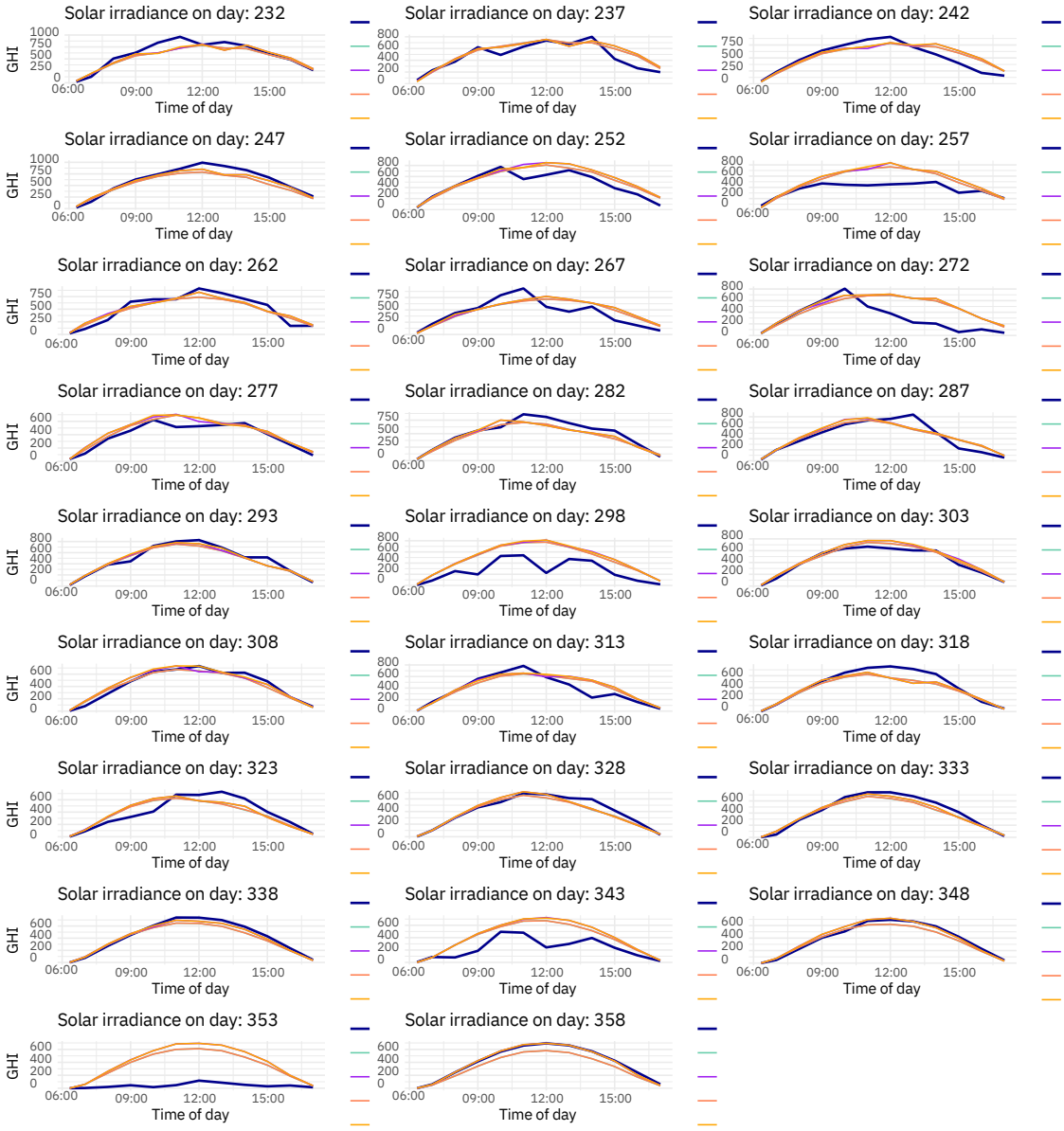

Figure 14: *28th-53th actual vs. fitted DOY forecasts (hourly-level aggregated time series).*

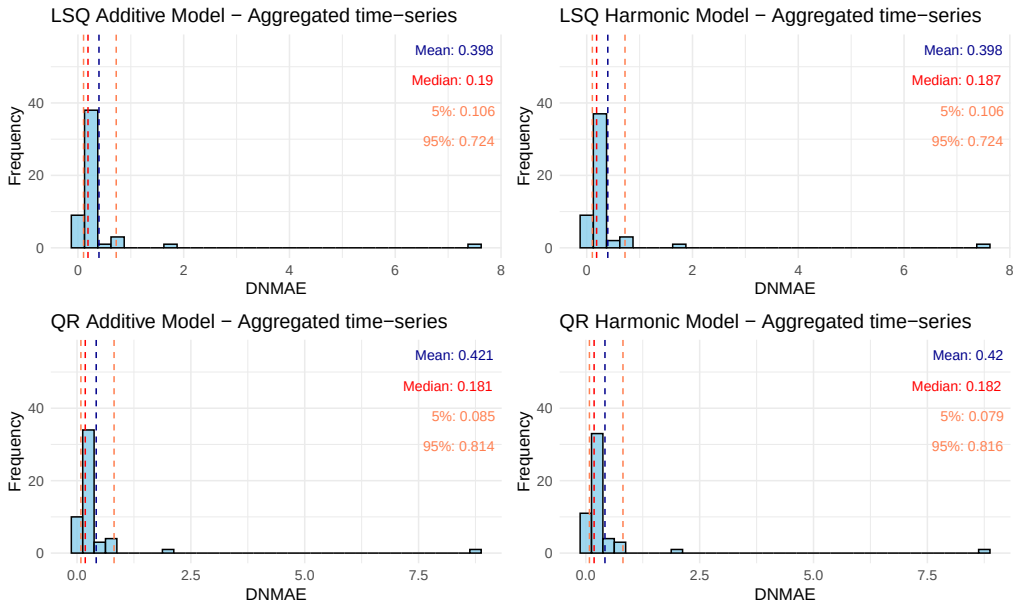

Figure 15: *DNMAE values over the 53 hourly-level forecasting DOY.*

For the Scaled Mean Interval Score (sMIS) using the first approach (∗) we get:

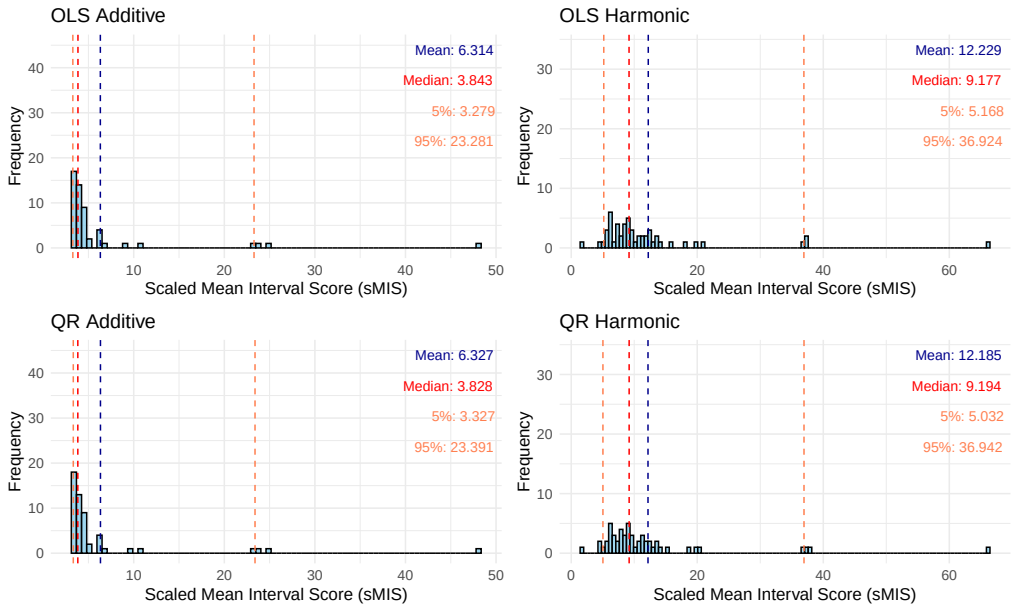

Figure 16: *Scaled Mean Interval Scores for both LS and QR (**first** approach).*

For the second approach[3], we get:

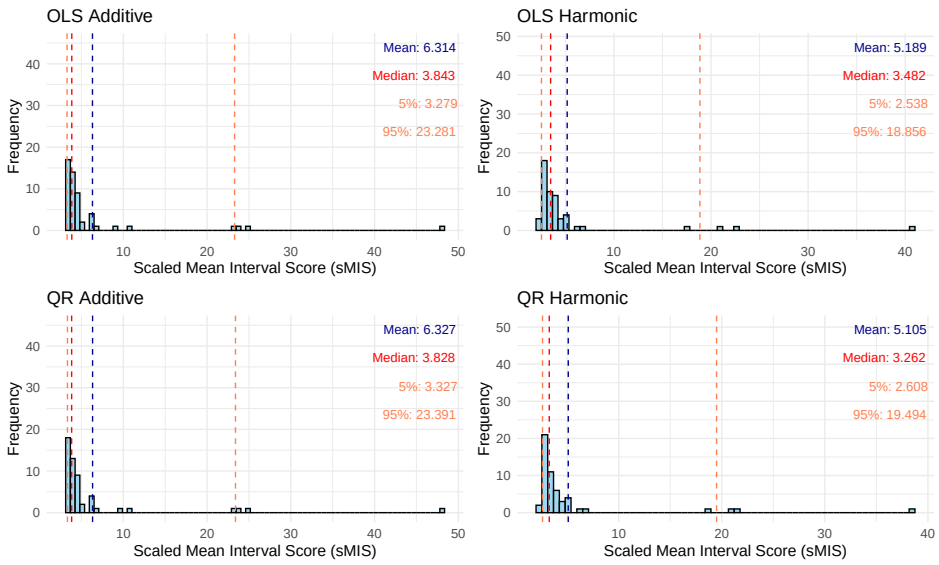

Figure 17: *Scaled Mean Interval Scores for both LS and QR (**second** approach).*

Overall, when comparing the original with the hourly-level aggregated time series results, we note a slight improvement in the hourly-level data across all models.

The ACF and PACF plots ([18], [19]) for the original solar irradiance data, collected at 10-minute intervals, exhibit a pattern that suggests a strong temporal dependence that diminishes relatively smoothly over time. The ACF shows a rapid decline from the first few lags before settling within the confidence bounds, indicating a strong initial correlation that decreases with increasing lag. This is indicative of an autoregressive nature where current values are primarily influenced by their recent past. The PACF corroborates this, displaying a significant peak at lag 2 and quickly dropping below the significance level for subsequent lags, which typically characterizes an AR(1) process.

Upon aggregating the data to an hourly level, the ACF plot [20] changes dramatically. The clear decay observed in the 10-minute data is replaced by a series of irregular spikes across the lags. This indicates that the aggregation has either introduced new dynamics or obscured the finer temporal relationships present in the higher-resolution data.

```
acf(data["GHI"])
>>
```

[3] *Recall* : For the QR, the lower-upper bounds are given by QR with `tau` $\in \{0.05, 0.95\}$.

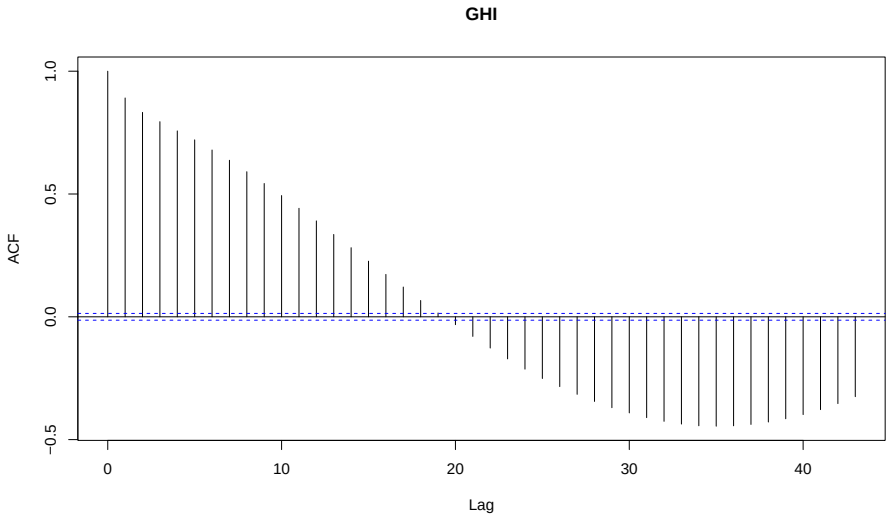

Figure 18: *Autocorrelation Function (original time-series).*

```
pacf(data["GHI"])
>>
```

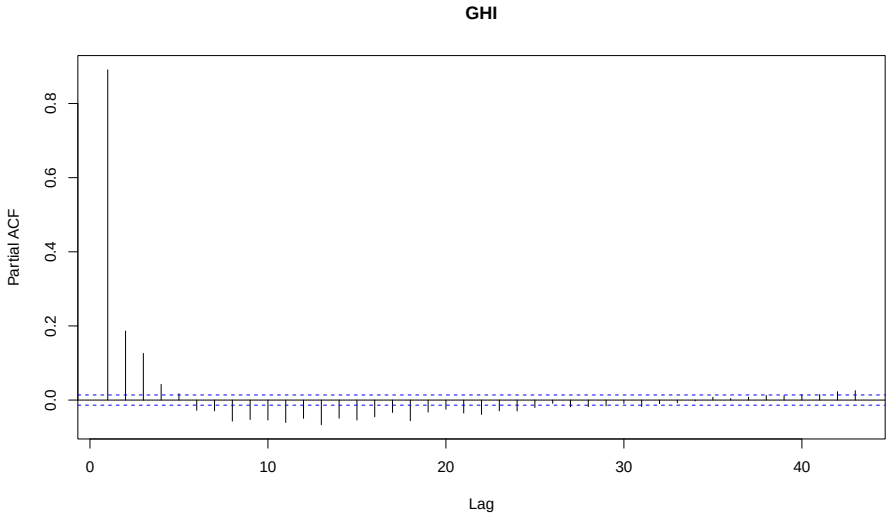

Figure 19: *Partial Autocorrelation Function (original time-series).*

It also suggests variability that might be due to environmental factors that impact solar irradiance measurements within each hour.

```
acf(data.agg["GHI"])
>>
```

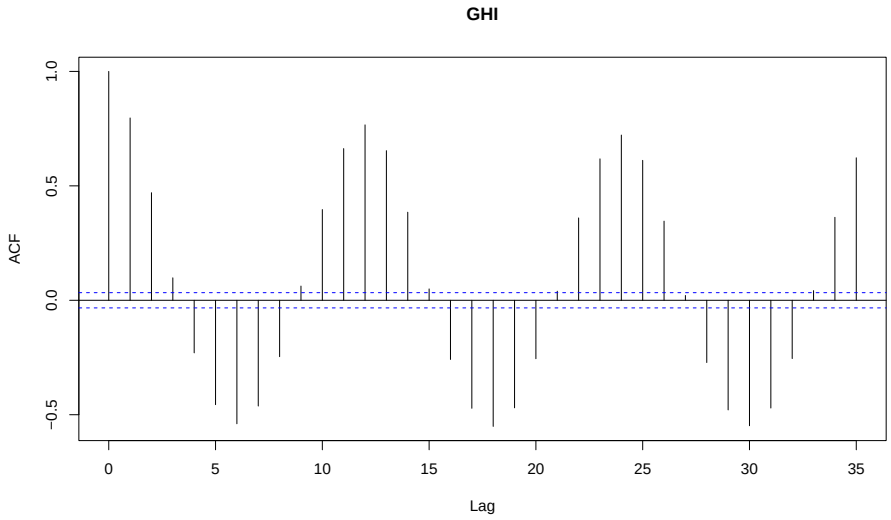

Figure 20: *Autocorrelation Function (aggregated time-series).*

```
pacf(data.agg["GHI"])
>>
```

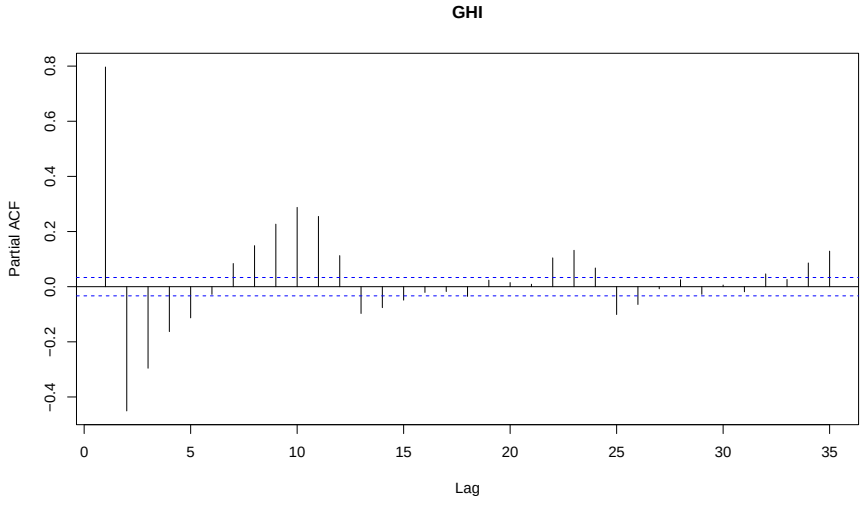

Figure 21: *Partial Autocorrelation Function (aggregated time-series).*

We can see [20] that at lag 12 and 24 there are significant peaks. This makes sense, since the sequences are repeating with a period of 12. Interestingly, note that there is a negative correlation at lags 6, 18 and 30. This is very characteristic of seasonal time series and behaviour of this sort in a correlogram is usually indicative that seasonality/periodic effects have not fully been accounted for in a model.

The PACF for the aggregated data [21] echoes this complexity, with non-sequential lags breaching the significance threshold, which indicates a less clear-cut relationship between past and current values.

Comparing the plots from both resolutions, its evident that the aggregation has a substantial impact on the apparent temporal structure of the data. While the original data seems amenable to a simple AR model, the hourly aggregated data may require a more nuanced approach, perhaps a higher-order AR model or a different time series model altogether, such as an ARIMA model with additional parameters to account for the observed irregularities. Moreover, the significance of non-sequential lags in the PACF of the hourly data hints at underlying patterns or cycles not immediately discernible, possibly calling for models that can capture such complexities or for further data decomposition to reveal any hidden periodicity or trends.

The ACF at lag k is defined as:

$$\text{ACF}(k) = \frac{\sum_{t=k+1}^{T}(Y_t - \bar{Y})(Y_{t-k} - \bar{Y})}{\sum_{t=1}^{T}(Y_t - \bar{Y})^2}.$$

The PACF gives the partial correlation of a time series with its own lagged values, controlling for the values of the time series at all shorter lags. The PACF at lag k is the correlation that results after removing the effect of any correlations due to terms at shorter lags. The mathematical definition is more complex, as it involves solving a system of equations (the Yule-Walker equations for an autoregressive process):

$$\text{PACF}(k) = Corr(Y_t - \hat{Y}_t^{(k-1)}, Y_{t-k} - \hat{Y}_{t-k}^{(k-1)}).$$

where:

- $\hat{Y}_t^{(k-1)}$ is the predicted value of Y_t based on all the values up to lag $k-1$
- $\hat{Y}_{t-k}^{(k-1)}$ is the predicted value of Y_{t-k} based on all the values up to lag $k-1$
- The predictions are usually obtained by fitting an autoregressive model of order $k-1$.

In other words, PACF at lag k is the amount of correlation between Y_t and Y_{t-k} that is not explained by correlations at all lower-order lags.

Part 3 (Traffic volumes and occupancies data in Athens) 30%

- The dataset (`APRIL_MIN_15`) contains measurements of traffic volumes ($V(t)$) and occupancies ($O(t)$) observed in seven locations in Athens for 20 week-days; data are updated every 15 minutes. The first 10 days should be utilized to train distributed lag models whereas the remaining data should be used to evaluate forecasting accuracy. We will analyze data from one certain location.
- We will develop (at least) 2 distributed lag models (DL) to forecast volumes (note that the association between volumes and occupancies is not linear). Then we compare conventional coefficient estimates, derived for instance using R package `dLagM`, against heteroscedasticity (account for heteroscedasticity).
- DL models should be evaluated in terms of their one-step forecasting accuracy. Ideally our metric should comply with the criteria of a Traffic Control Center: forecasting accuracy is more valuable during congested periods.
- Repeat the previous analyses for data aggregated at 30-min intervals. Note that, a-priori we should expect that the estimated models are simpler in this case.

***Solution*:**

In this project, we will analyze data from location **4**. Let's view a summary of this specific location:

Characteristic	Value
Total Days	20
Data Points Frequency/Day	96
Mean Volume	297.76
Median Volume	298.00
Mean Occupancy	64.01
Median Occupancy	73.59
Average Weekly Volume	1488.78
Average Weekly Occupancy	320.05

```
data <- read.csv("APRIL_MIN_15.csv")
data <- data[c(15, 4, 11)] # location 4
freq <- as.numeric(ts(table(data$DAY))[1]) # freq = 96
data.mts <- ts(data[c(2,3)], start=min(data$DAY), end=max(data$DAY), frequency = freq)

plot(data.mts, col='midnightblue')
grid(lty = 1, col = "gray", lwd = 0.5656)
>>
```

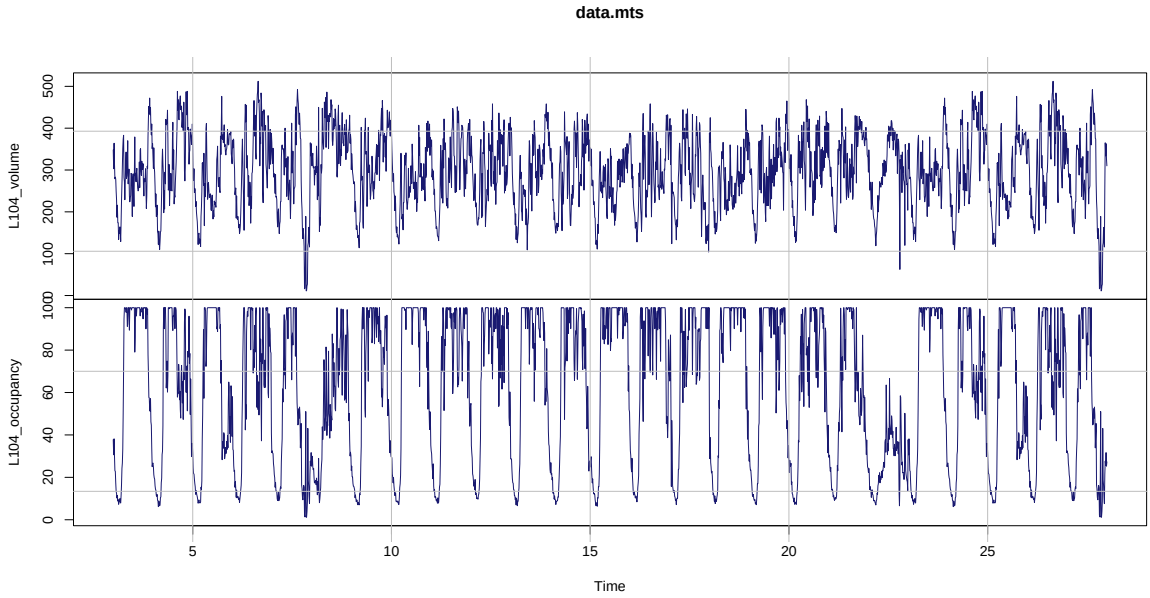

Figure 22: *Volumes & occupancies for 20 week-days (location 4).*

From the PACF of occupancies [Figure 23], we observe that only a few lags fall outside the confidence interval, as indicated by the dotted blue lines. These lags could be crucial for constructing distributed lag models, leveraging the identified lags for enhanced predictive accuracy.

```
data.vol <- data[-3]
data.occ <- data[-2]

par(mfrow=c(2,2))
acf(data.vol[-1])
pacf(data.vol[-1], main=names(data.vol[-1]))
acf(data.occ[-1])
pacf(data.occ[-1], main=names(data.occ[-1]))
>>
```

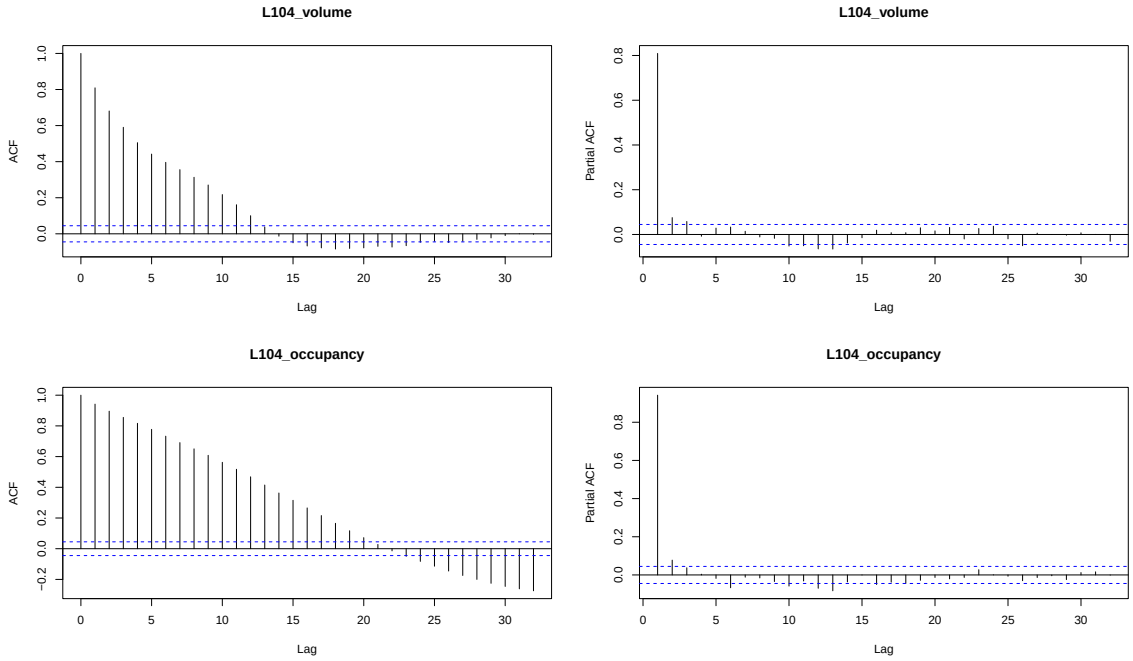

Figure 23: *Autocorrelation & Partial autocorrelation functions.*

Now let's move on to some hypothesis testing. An augmented *DickeyFuller* test (ADF) tests the null hypothesis that a unit root is present in a time series sample. The alternative hypothesis is different depending on which version of the test is used, but is usually stationarity or trend-stationarity. It is an augmented version of the DickeyFuller test for a larger and more complicated set of time series models.

The augmented DickeyFuller statistic, used in the test, is a negative number. The more negative it is, the stronger the rejection of the hypothesis that there is a unit root at some level of confidence.

```
library(tseries)
library(zoo)
adf.test(ts(read.zoo(data.vol)))
adf.test(ts(read.zoo(data.occ)))
>>
    Augmented Dickey-Fuller Test
data:  ts(read.zoo(data.vol))
Dickey-Fuller = -10.618, Lag order = 12, p-value = 0.01
alternative hypothesis: stationary

    Augmented Dickey-Fuller Test
data:  ts(read.zoo(data.occ))
Dickey-Fuller = -8.4424, Lag order = 12, p-value = 0.01
```

```
alternative hypothesis: stationary
```

The *Ljung-Box* test examines whether there is significant evidence for non-zero correlations at given lags (1-12 shown below), with the null hypothesis of independence in a given time series (a non-stationary signal will have a low p-value).

The `Ljung-Box` test uses the following hypotheses:

H_0**:** The residuals are independently distributed.

H_A**:** The residuals are not independently distributed; they exhibit serial correlation.

```
Box.test(ts(read.zoo(data.vol)), lag=12, type="Ljung-Box") # test stationary signal
Box.test(ts(read.zoo(data.occ)), lag=12, type="Ljung-Box") # test stationary signal
>>
     Box-Ljung test
data:  ts(read.zoo(data.vol))
X-squared = 4729, df = 12, p-value < 2.2e-16

     Box-Ljung test
data:  ts(read.zoo(data.occ))
X-squared = 12149, df = 12, p-value < 2.2e-16
```

Here we see a p-value much smaller than 0.005; suggests strong evidence against the null hypothesis of no serial correlation in your time series data. This indicates the presence of autocorrelation at one or more lags.

And the classical DW test:

```
library(lmtest)
dwtest(as.formula("data$L104_volume ~ data$L104_occupancy"))
>>
           Durbin-Watson test
data:  as.formula("data$L104_volume ~ data$L104_occupancy")
DW = 0.4279, p-value < 2.2e-16
alternative hypothesis: true autocorrelation is greater than 0
```

We will now split the data intro training & forecasting (50%-50%):

```
days <- unique(data$DAY)
first_ten.days <- days[seq(1,10)]
remaining.days <- days[seq(10+1, length(days))]

train.ts <- subset(data, DAY %in% first_ten.days) # 2 weeks minus WeekEnd
forecast.ts <- subset(data, DAY %in% remaining.days)
```

◇ Fit the ***distributed lag models*** models:

In a distributed-lag '`dlm`' model, the effect of an independent variable X on a dependent variable Y occurs over the time. Therefore, DLMs are dynamic models. A linear finite DLM with one

independent variable is written as follows:

$$Y_t = \alpha + \sum_{s=0}^{q} \beta_s X_{t-s} + \epsilon_t,$$

where ϵ_t is a stationary error term with $\mathbb{E}(\epsilon_t) = 0$, $Var(\epsilon_t) = \sigma^2$, $Cov(\epsilon_t, \epsilon_s) = 0$.

```
library(dLagM)
model_1.dlm = dlm(x = train.ts$L104_occupancy,
                  y = train.ts$L104_volume , q = 1)
model_2.dlm = dlm(x = train.ts$L104_occupancy,
                  y = train.ts$L104_volume , q = 2)
model_3.dlm = dlm(x = train.ts$L104_occupancy,
                  y = train.ts$L104_volume , q = 3)

AIC(model_1.dlm)
AIC(model_2.dlm)
AIC(model_3.dlm)
>>
[1] 11271.07
[1] 11221.85
[1] 11169.3
```

A more sophisticated DL (based on PACF plot) could be:

```
removed.lags <- list(x = c(3, 4, 5, 7, 8, 9, 12, 15))
linear.dlm = dlm(x = train.ts$L104_occupancy,
                 y = train.ts$L104_volume, q = 15,
                 remove = removed.lags)
summary(linear.dlm)
>>
Residuals:
     Min       1Q   Median       3Q      Max
-285.252  -64.518    0.418   62.901  215.559
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 246.63343    6.52238  37.813  < 2e-16 ***
x.t          -1.00182    0.22482  -4.456 9.36e-06 ***
x.1           0.09069    0.29589   0.306  0.75930
x.2           0.63921    0.24042   2.659  0.00798 **
x.6           1.26497    0.16655   7.595 7.45e-14 ***
x.10          0.06899    0.24163   0.286  0.77531
x.11         -0.18569    0.26357  -0.705  0.48129
x.13          0.11710    0.26407   0.443  0.65754
x.14         -0.08442    0.22575  -0.374  0.70853
---
Residual standard error: 80.98 on 936 degrees of freedom
Multiple R-squared:  0.1958,   Adjusted R-squared:  0.1889
F-statistic: 28.48 on 8 and 936 DF,  p-value: < 2.2e-16
AIC and BIC values for the model:
       AIC      BIC
1 10997.76 11046.27
```

One can observe that DLM exhibits lower `AIC`.

We will now investigate if occupancies $O(t)$ are influencing volumes $V(t)$. The ***Granger Causality*** test is used to determine whether or not one time series is useful for forecasting another.

H_0: Time series X does not Granger-cause time series Y.

H_1: Granger-causes time series Y.

The term *'Granger-causes'* means that knowing the value of time series X at a certain lag is useful for predicting the value of time series Y at a later time period.

```
grangertest(as.formula("data$L104_volume ~ data$L104_occupancy"),
            data=data)[2, c(1,3,4)]
>>
  Res.Df      F    Pr(>F)
2   1917 56.172 1.009e-13 ***
```

F: This is the F test statistic. It turns out to be `56.17196`.

$Pr(> F)$: This is the p-value that corresponds to the F test statistic. It turns out to be `1.008755e-13`.

We are also going to utilize higher order polynomial DLM: Finite distributed lag models, in general, suffer from the multicollinearity due to inclusion of the lags of the same variable in the model. To reduce the impact of this multicollinearity, a polynomial shape is imposed on the lag distribution (Judge and Griffiths, 2000). The resulting model is called Polynomial Distributed Lag model or Almon Distributed Lag Model.

Imposing a polynomial pattern on the lag distribution is equivalent to representing β parameters with another kth order polynomial model of time. So, the effect of change in X_{t-s} on the expected value of Y_t is represented as follows:

$$\frac{\partial \mathbb{E}(Y_t)}{\partial X_{ts}} = \beta_s = \gamma_0 + \gamma_1 s + \gamma_2 s^2 + \cdots + \gamma_k s^k,$$

where $s = 0, \ldots, q$[4]. Then the model becomes:

$$Y_t = \alpha + \gamma_0 Z_{t0} + \gamma_1 Z_{t1} + \gamma_2 Z_{t2} + \cdots + \gamma_k Z_{tk} + \epsilon_t.$$

In R:

```
poly3.dlm <- polyDlm(x = train.ts$L104_occupancy,
                     y = train.ts$L104_volume ,
                     q = 20, k = 3)
par(mfrow=c(2,2))
plot(poly3.dlm$model)
>>
```

[4] Judge and Griffiths, 2000

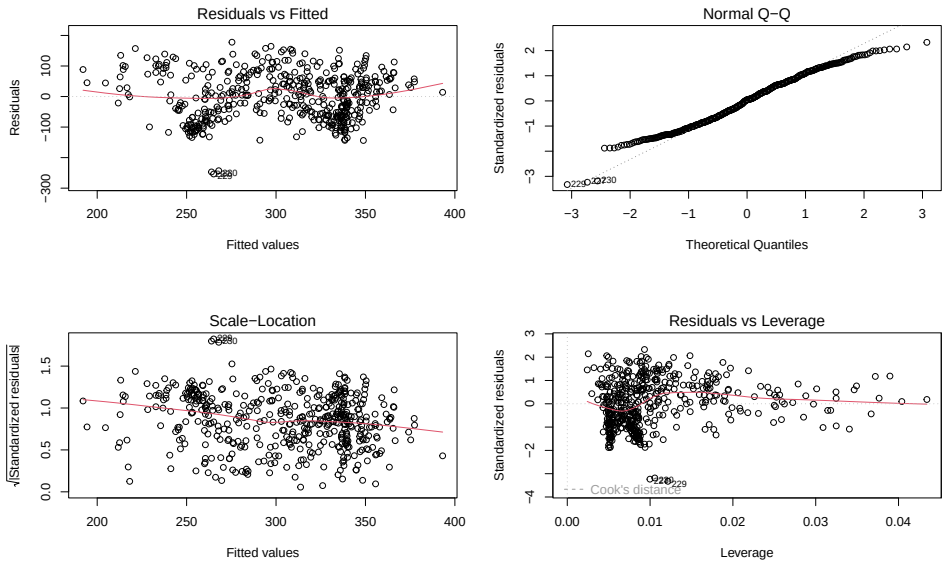

Figure 24: *Residual diagnostics - polynomial DLM of order 3 and 5 lags.*

```
library(olsrr)
#Breusch Pagan Test for Heteroskedasticity
ols_test_breusch_pagan(poly3.dlm$model)
>>
 Breusch Pagan Test for Heteroskedasticity
 -----------------------------------------
 Ho: the variance is constant
 Ha: the variance is not constant
             Data
 -------------------------------
 Response : y.t
 Variables: fitted values of y.t
        Test Summary
 -------------------------------
 DF            =    1
 Chi2          =    15.85033
 Prob > Chi2   =    6.855447e-05
```

In order to account for heteroscedasticity, we will use the following transformation for $O(t)$

```
period <- 1/freq # 1/96
train.ts$sin.occ <- sin(2*pi*train.ts$L104_occupancy/period)
forecast.ts$sin.occ <- sin(2*pi*forecast.ts$L104_occupancy/period)
data$sin.occ <- sin(2*pi*data$L104_occupancy/period)

plot(lm(train.ts$L104_volume ~ train.ts$sin.occ))
>>
```

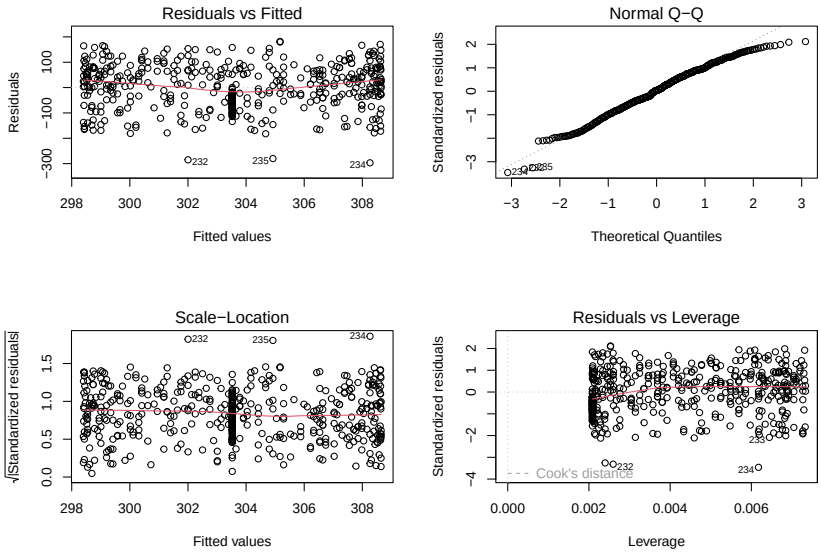

Figure 25: *Residual diagnostics - transformed occupancies.*

```
removed.lags <- list(x = c(3, 4, 5, 7, 8, 9, 12, 15))
linear.dlm = dlm(x = train.ts$sin.occ,
                 y = train.ts$L104_volume, q = 15,
                 remove = removed.lags)
ols_test_breusch_pagan(linear.dlm$model)
>>
 Breusch Pagan Test for Heteroskedasticity
 -----------------------------------------
 Ho: the variance is constant
 Ha: the variance is not constant
             Data
 -------------------------------
 Response : y.t
 Variables: fitted values of y.t
        Test Summary
 ----------------------------
 DF            =    1
 Chi2          =    0.1328455
 Prob > Chi2   =    0.7154998
```

Obviously, we don't have substantial evidence against the null hypothesis.

```
poly3.dlm <- polyDlm(x = train.ts$sin.occ, y = train.ts$L104_volume, q = 5, k = 3)
>>
Estimates and t-tests for beta coefficients:
       Estimate Std. Error t value P(>|t|)
beta.0    -3.65       4.52  -0.808   0.419
```

```
beta.1    -4.16       3.51  -1.190   0.236
beta.2    -4.50       3.14  -1.430   0.152
beta.3    -4.68       3.14  -1.490   0.136
beta.4    -4.74       3.50  -1.350   0.176
beta.5    -4.72       4.51  -1.050   0.295
```

Let's view the transformed occupancies:

```
library(ggfortify)
autoplot(ts( cbind(ts(data$sin.occ), ts(data$L104_occupancy))))
>>
```

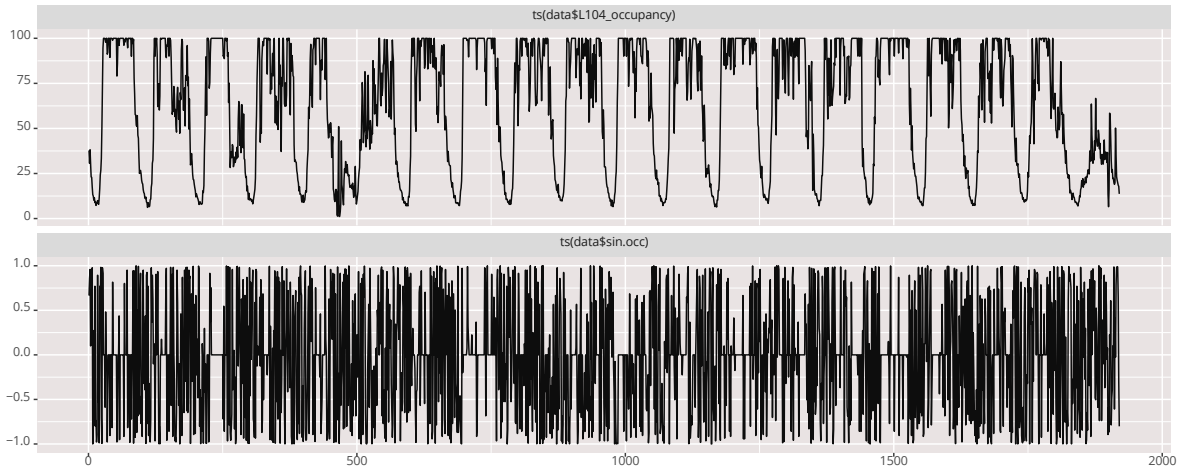

Figure 26: *Transformed occupancies.*

Let's define some accuracy metrics:

```
MAE <- function(y, forecast) {
  finite_indices <- is.finite(y) & is.finite(forecast)
  if (any(finite_indices)) {
    mean(abs(y[finite_indices] - forecast[finite_indices]), na.rm = TRUE)
  } else {
    NA  # Failure
  }
}
RMSE <- function(y, forecast) {
  finite_indices <- is.finite(y) & is.finite(forecast)
  if (any(finite_indices)) {
    sqrt(mean((y[finite_indices] - forecast[finite_indices])^2, na.rm = TRUE))
  } else {
    NA  # Or some other default value indicating failure
  }
}
```

```r
sMdAPE <- function(y, forecast) {
  finite_indices <- is.finite(y) & is.finite(forecast) & (y + forecast != 0)
  if (any(finite_indices)) {
    median(200 * abs(y[finite_indices] - forecast[finite_indices]) /
           (y[finite_indices] + forecast[finite_indices]), na.rm = TRUE)
  } else {
    NA  # Failure
  }
}
DNMAE <- function (y, forecast) MAE(y, forecast)/mean(forecast)
DNRMSE <- function (y, forecast) RMSE(y, forecast)/mean(forecast)
DNsMdAPE <- function (y, forecast) sMdAPE(y, forecast)/mean(forecast)
```

We are ready now to forecast (with `sin` transformation) on the testing days. We utilize the '`dlmForecast.main`' function that can be found here

```r
# forecasted.vals <- as.vector(predict(linear.dlm$model, x=forecast.ts)) # WRONG (returns the fitted values)
linear.dlm <- dlm(x = train.ts$sin.occ,
                  y = train.ts$L104_volume, q = 8) # full dlm (in order to forecast...)

forecast.full.dlm <- dlmForecast.main(model = linear.dlm,
                                      x = as.vector(forecast.ts$sin.occ),
                                      h = h,
                                      type = 1)$forecast
MAE(forecast.full.dlm, forecast.ts$L104_volume)
RMSE(forecast.full.dlm, forecast.ts$L104_volume)
sMdAPE(forecast.full.dlm, forecast.ts$L104_volume)
>>
[1] 66.5681345686927
[1] 80.0944295272466
[1] 19.4315016718303

DNMAE(forecast.full.dlm, forecast.ts$L104_volume)
DNRMSE(forecast.full.dlm, forecast.ts$L104_volume)
DNsMdAPE(forecast.full.dlm, forecast.ts$L104_volume)
>>
[1] 0.228072114168519
[1] 0.274415168665276
[1] 0.0665751518572334
```

For the **polynomial** DLM:

```r
# forecasted.vals <- as.vector(predict(poly3.dlm$model, x=forecast.ts)) # Do NOT run this
forecasted.vals <- ts(dLagM::forecast(model = poly3.dlm , x = as.vector(forecast.ts$sin.occ),
                                      h = length(forecast.ts$sin.occ))$forecast)
MAE(forecasted.vals, forecast.ts$L104_volume)
RMSE(forecasted.vals, forecast.ts$L104_volume)
sMdAPE(forecasted.vals, forecast.ts$L104_volume)
>>
[1] 66.2062039346244
[1] 79.8210338988611
[1] 19.540390840129
```

```
DNMAE(forecasted.vals, forecast.ts$L104_volume)
DNRMSE(forecasted.vals, forecast.ts$L104_volume)
DNsMdAPE(forecasted.vals, forecast.ts$L104_volume)
>>
[1] 0.226832087158162
[1] 0.27347847546553
[1] 0.0669482219903358
```

We will forecast now ***without*** (occupancies) transformation. We repeat the same code, this time with the original training and test data.

```
linear.dlm <- dlm(x = train.ts$L104_occupancy,
                  y = train.ts$L104_volume, q = 8)
poly3.dlm <- polyDlm(x = train.ts$L104_occupancy,
                     y = train.ts$L104_volume,
                     q = 5, k = 3)
# forecasted.vals <- as.vector(predict(linear.dlm$model, x=forecast.ts)) # wrong way to forecast
forecast.full.dlm <- dlmForecast.main(model = linear.dlm,
                                      x = as.vector(forecast.ts$L104_occupancy),
                                      h = h,
                                      type = 1)$forecast

MAE(forecast.full.dlm, forecast.ts$L104_volume)
RMSE(forecast.full.dlm, forecast.ts$L104_volume)
sMdAPE(forecast.full.dlm, forecast.ts$L104_volume)
>>
[1] 62.208094862623
[1] 74.8729951821495
[1] 18.9088379500844

DNMAE(forecast.full.dlm, forecast.ts$L104_volume)
DNRMSE(forecast.full.dlm, forecast.ts$L104_volume)
DNsMdAPE(forecast.full.dlm, forecast.ts$L104_volume)
>>
[1] 0.213133983784291
[1] 0.256525774921644
[1] 0.0647844299031013

autoplot(ts(cbind(ts(forecast.full.dlm), ts(forecast.ts$L104_volume))), facets = FALSE)
>>
```

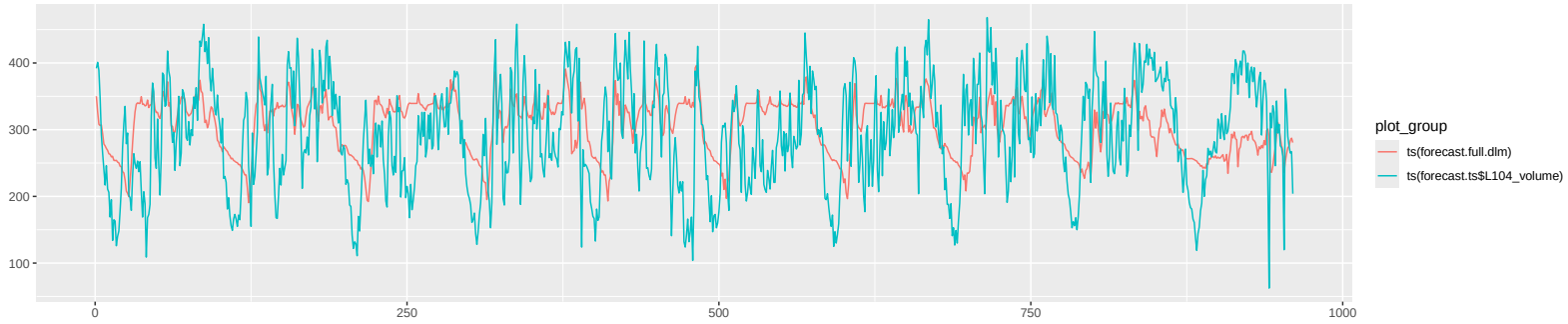

Figure 27: *Forecast volumes VS actual volumes - linear.dlm.*

For the `PolyDLM`:

```
forecasted.vals <- #as.vector(predict(poly3.dlm$model, x=forecast.ts)) # wrong way
forecasted.vals <- ts(dLagM::forecast(model = poly3.dlm , x = as.vector(forecast.ts$L104_occupancy),
                                      h = length(forecast.ts$L104_occupancy))$forecast)
MAE(forecasted.vals, forecast.ts$L104_volume)
RMSE(forecasted.vals, forecast.ts$L104_volume)
sMdAPE(forecasted.vals, forecast.ts$L104_volume)
>>
[1] 62.4896876179262
[1] 75.0113316255701
[1] 19.2099130642559

DNMAE(forecasted.vals, forecast.ts$L104_volume)
DNRMSE(forecasted.vals, forecast.ts$L104_volume)
DNsMdAPE(forecasted.vals, forecast.ts$L104_volume)
>>
[1] 0.214098761533475
[1] 0.25699973570366
[1] 0.0658159570482965

autoplot(ts(cbind(ts(forecasted.vals), ts(forecast.ts$L104_volume))), facets = FALSE)
>>
```

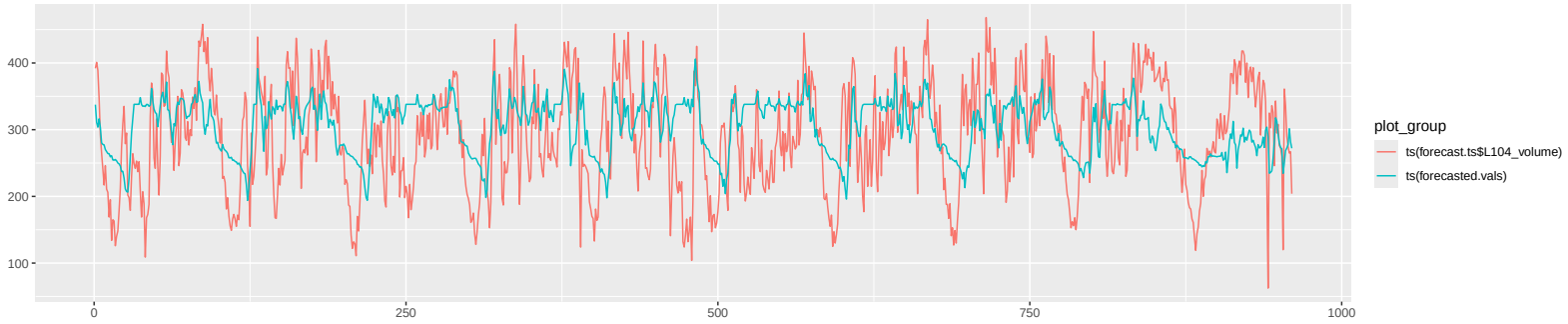

Figure 28: *Forecast volumes VS actual volumes - poly3.dlm.*

We will repeat the previous analysis for data aggregated at 30-min intervals. The code below aggregates traffic volumes and occupancies data.

```
# calc. the group index, ensuring it aligns with the data length
group_index <- gl(ceiling(nrow(data) / 2), 2, length = nrow(data))
aggregated_volume <- aggregate(data$L104_volume, list(Group = group_index), mean)
aggregated_day <- tapply(data$DAY, group_index, function(x) x[1])
# combine 'DAY' and the aggregated 'L104_volume' into a new data
vol.agg <- data.frame(DAY = aggregated_day, L104_volume = aggregated_volume$x)
aggregated_occ <- aggregate(data$L104_occupancy, list(Group = group_index), mean)
occ.agg <- data.frame(DAY = aggregated_day, L104_occupancy = aggregated_occ$x)

freq <- unique(ts(table(vol.agg$DAY)))
dim(vol.agg)
dim(occ.agg)
>>
[1] 9602
[1] 9602
```

We may now forecast:

```
linear.dlm <- dlm(x = train.ts$L104_occupancy,
                  y = train.ts$L104_volume, q = 8)
poly3.dlm <- polyDlm(x = train.ts$L104_occupancy,
                     y = train.ts$L104_volume,
                     q = 5, k = 3)
# forecasted.vals <- as.vector(predict(linear.dlm$model, x=forecast.ts)) # wrong way to forecast
forecast.full.dlm <- dlmForecast.main(model = linear.dlm,
                                      x = as.vector(forecast.ts$L104_occupancy),
                                      h = h/2,
                                      type = 1)$forecast
MAE(forecast.full.dlm, forecast.ts$L104_volume)
RMSE(forecast.full.dlm, forecast.ts$L104_volume)
sMdAPE(forecast.full.dlm, forecast.ts$L104_volume)
>>
[1] 58.4745966329275
```

```
[1] 70.8354824360449
[1] 17.2657497272355

DNMAE(forecast.full.dlm, forecast.ts$L104_volume)
DNRMSE(forecast.full.dlm, forecast.ts$L104_volume)
DNsMdAPE(fforecast.full.dlm, forecast.ts$L104_volume)
>>
[1] 0.200342475654941
[1] 0.242692668827374
[1] 0.0591549705953021

autoplot(ts(cbind(ts(forecast.full.dlm), ts(forecast.ts$L104_volume))), facets = FALSE)
>>
```

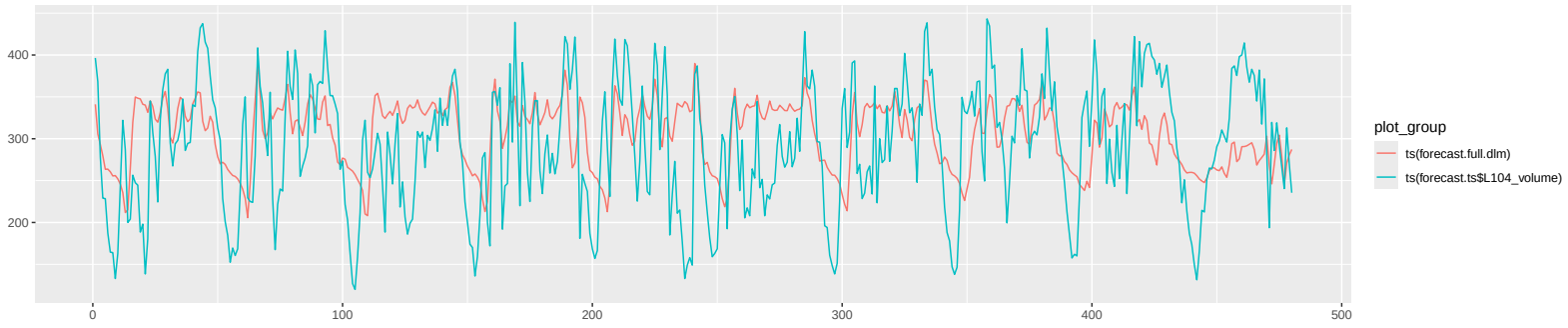

Figure 29: *Forecast volumes VS actual volumes - linear.dlm (aggregated data).*

Ultimately, for the PDLM:

```
#forecasted.vals <- as.vector(predict(poly3.dlm$model, x=forecast.ts)) # wrong!
forecasted.vals <- ts(dLagM::forecast(model = poly3.dlm , x = as.vector(forecast.ts$L104_occupancy),
                                      h = length(forecast.ts$L104_occupancy))$forecast)
MAE(forecasted.vals, forecast.ts$L104_volume)
RMSE(forecasted.vals, forecast.ts$L104_volume)
sMdAPE(forecasted.vals, forecast.ts$L104_volume)
>>
[1] 58.6915716142754
[1] 71.0081639517964
[1] 16.9856978441124

DNMAE(forecasted.vals, forecast.ts$L104_volume)
DNRMSE(forecasted.vals, forecast.ts$L104_volume)
DNsMdAPE(forecasted.vals, forecast.ts$L104_volume)
>>
[1] 0.20108586350918
[1] 0.243284300824131
[1] 0.0581954720983922
```

```
autoplot(ts(cbind(ts(forecasted.vals), ts(forecast.ts$L104_volume))), facets = FALSE)
>>
```

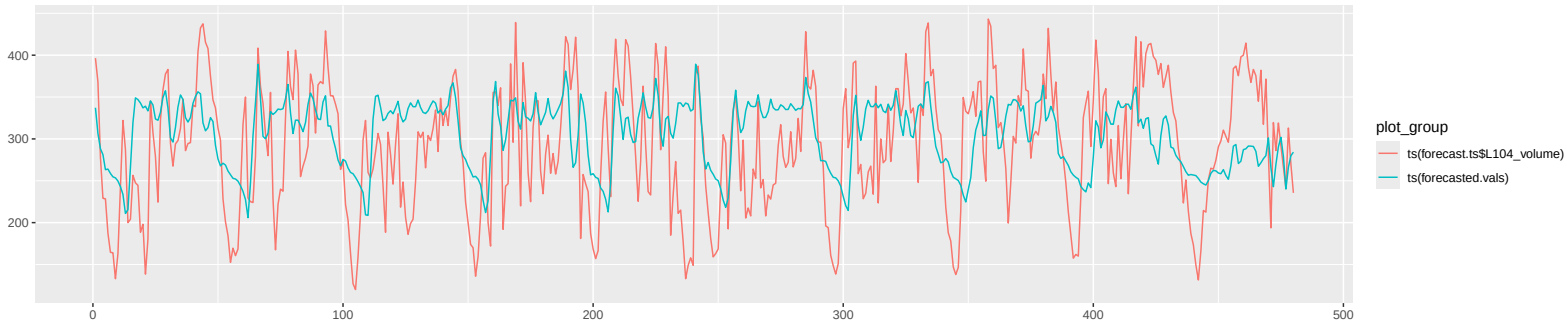

Figure 30: *Forecast volumes VS actual volumes - poly3.dlm (aggregated data).*

Further investigation could include distributed lag models with ***Koyck*** transformation (DLMK). To deal with infinite DLMs, we can use the Koyck transformation. When we apply Koyck transformation, we get the following:

$$Y_t - \phi Y_{t-1} = \alpha(1-\phi) + \beta X_t + (\epsilon_t - \phi\epsilon_{t-1}).$$

When we solve this equation for Y_t, we obtain *Koyck DLM* as follows:

$$Y_t = \delta_1 + \delta_2 Y_{t-1} + \delta_3 X_t + \nu_t,$$

where $\delta_1 = \alpha(1-\phi), \delta_2 = \phi, \delta_3 = \beta$ and the random error after the transformation is $\nu_t = (\epsilon_t - \phi\epsilon_{t-1})$.[5]

```
train.ts <- subset(data, DAY %in% first_ten.days) # 2 weeks minus WeekEnd
forecast.ts <- subset(data, DAY %in% remaining.days)

koyck.dlm <- koyckDlm(x = train.ts$L104_occupancy,
                      y = train.ts$L104_volume)
summary(koyck.dlm)
>>
Call:
"Y ~ (Intercept) + Y.1 + X.t"
Residuals:
      Min        1Q    Median        3Q       Max
-183.2367  -30.5303   -0.8655   29.4293  208.4892
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 41.42282    6.08236   6.810 1.72e-11 ***
Y.1          0.80637    0.01881  42.866  < 2e-16 ***
X.t          0.27442    0.05239   5.238 2.00e-07 ***
```

[5] Judge and Griffiths, 2000.

```
---
Residual standard error: 50.72 on 956 degrees of freedom
Multiple R-Squared: 0.6828, Adjusted R-squared: 0.6821
Wald test:  1052 on 2 and 956 DF,  p-value: < 2.2e-16
                          alpha     beta       phi
Geometric coefficients:  213.928 0.274423 0.8063703
```

We may now forecast with the **Koyck** transformation DLM:

```
# forecasted.vals <- as.vector(predict(koyck.dlm$model, x=forecast.ts)) # Do NOT run this!
koyck.forecast.values <- dLagM::forecast(model = koyck.dlm , x = as.vector(forecast.ts$L104_occupancy),
                                         h = length(forecast.ts$L104_occupancy))$forecast

MAE(koyck.forecast.values, forecast.ts$L104_volume)
RMSE(koyck.forecast.values, forecast.ts$L104_volume)
sMdAPE(koyck.forecast.values, forecast.ts$L104_volume)
>>
[1] 68.3778272395585
[1] 80.7137513942121
[1] 22.2614228364256

DNMAE(koyck.forecast.values, forecast.ts$L104_volume)
DNRMSE(koyck.forecast.values, forecast.ts$L104_volume)
DNsMdAPE(koyck.forecast.values, forecast.ts$L104_volume)
>>
[1] 0.234272384554852
[1] 0.276537055487925
[1] 0.0762708734982454

autoplot(ts(cbind(ts(koyck.forecast.values), ts(forecast.ts$L104_volume))), facets = FALSE)
>>
```

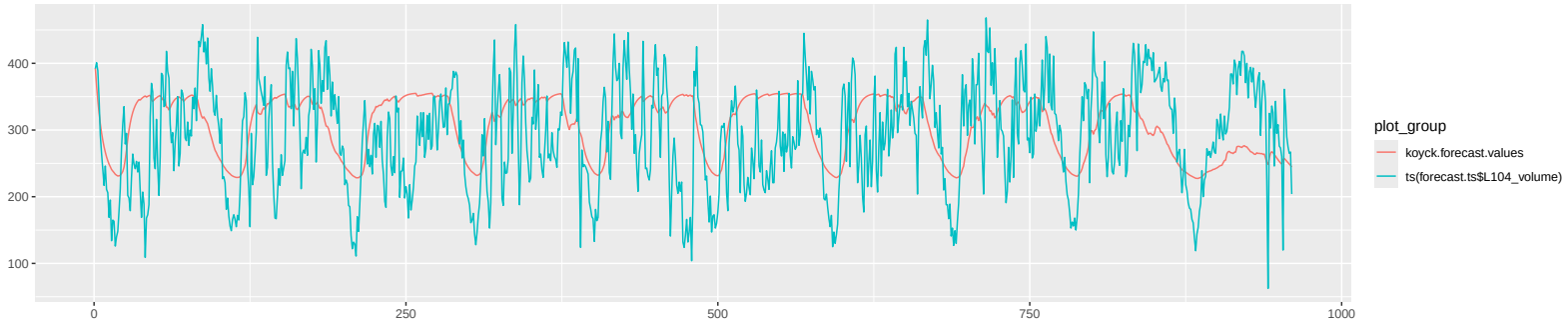

Figure 31: *Forecast volumes VS actual volumes - koyck.dlm.*

We now examine **Koyck** using the aggregated data:

```
train.ts <- subset(data.agg, DAY %in% first_ten.days) # 2 weeks minus weekends
forecast.ts <- subset(data.agg, DAY %in% remaining.days)
```

```
koyck.dlm <- koyckDlm(x = train.ts$L104_occupancy,
                      y = train.ts$L104_volume)
summary(koyck.dlm)
>>
Call:
"Y ~ (Intercept) + Y.1 + X.t"
Residuals:
     Min       1Q   Median       3Q      Max
-149.637  -34.960   -3.794   37.306  200.169

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 50.84475    9.46437   5.372 1.22e-07 ***
Y.1          0.74391    0.03001  24.787  < 2e-16 ***
X.t          0.42474    0.08296   5.120 4.44e-07 ***
---
Residual standard error: 54.12 on 476 degrees of freedom
Multiple R-Squared: 0.6065, Adjusted R-squared: 0.6048
Wald test: 387.7 on 2 and 476 DF,  p-value: < 2.2e-16
                           alpha      beta       phi
Geometric coefficients:  198.5399 0.4247442 0.7439066

koyck.forecast.values.agg <- dLagM::forecast(model = koyck.dlm , x = as.vector(forecast.ts$L104_occu
                                        h = length(forecast.ts$L104_occupancy))$forecast

MAE(koyck.forecast.values.agg, forecast.ts$L104_volume)
RMSE(koyck.forecast.values.agg, forecast.ts$L104_volume)
sMdAPE(koyck.forecast.values.agg, forecast.ts$L104_volume)
>>
[1] 65.5540761349068
[1]77.5853056659032
[1] 20.9553055033205

DNMAE(koyck.forecast.values.agg, forecast.ts$L104_volume)
DNRMSE(koyck.forecast.values.agg, forecast.ts$L104_volume)
DNsMdAPE(koyck.forecast.values.agg, forecast.ts$L104_volume)
>>
[1] 0.224597802436901
[1] 0.265818545258671
[1] 0.0717959254853034

autoplot(ts(cbind(ts(koyck.forecast.values.agg), ts(forecast.ts$L104_volume))), facets = FALSE)
>>
```

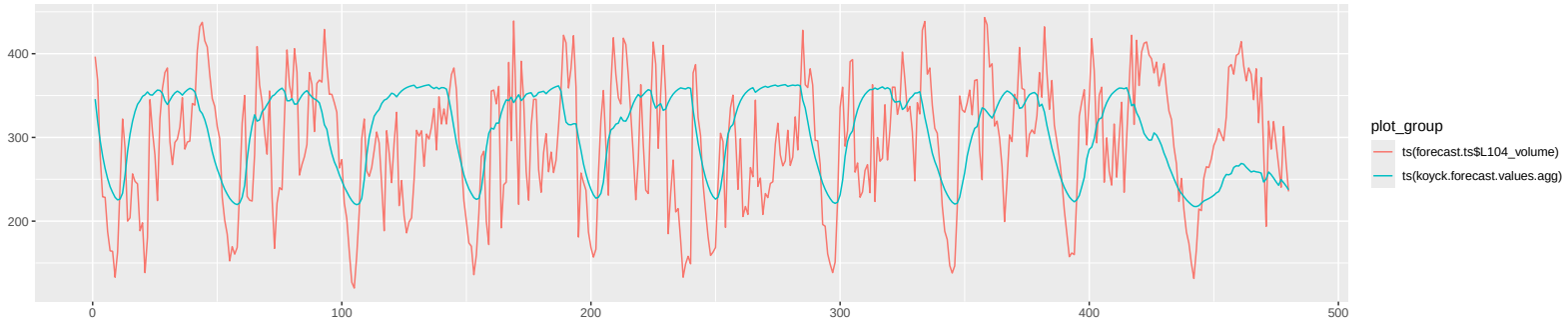

Figure 32: *Forecast volumes VS actual volumes - koyck.dlm (aggregated data).*

As we can observe from the above Figures ([31], [32]), the *Koyck DLM* appears to better fit the forecast data compared to the *polynomial* or *linear DLM*.

Some overall observations indicate that the Koyck model performs poorly compared to other models, showing the highest values across MAE, RMSE, and sMdAPE metrics. Conversely, the performance of the Linear DLM with aggregated data is the most superior, closely followed by the Polynomial DLM, which even surpasses it in terms of sMdAPE. In contrast, the performance of the Linear DLM does not significantly differ from that of the Polynomial DLM of order 3. A summary of all the results is shown on the table [3] below.

Metric	**tLDLM**	**tPDLM**	**LDLM**	**PDLM**	**aLDLM**	**aPDLM**	**Koyck**	**aKoyck**
MAE	66.57	66.21	62.21	62.49	58.47	58.69	68.38	65.55
RMSE	80.09	79.82	74.87	75.01	70.84	71.01	80.71	77.59
sMdAPE	19.43	19.54	18.91	19.21	17.27	16.99	22.26	20.96
DNMAE	0.228	0.227	0.213	0.214	0.200	0.201	0.234	0.225
DNRMSE	0.274	0.273	0.257	0.257	0.243	0.243	0.277	0.266
DNsMdAPE	0.067	0.067	0.065	0.066	0.059	0.058	0.076	0.072

Table 3: *Forecasting metrics comparison (LDLM vs PDLM vs Koyck).*

In this table, 'LDLM' stands for Linear Distributed Lag Model, 'PDLM' for Polynomial Distributed Lag Model, 't' indicates transformed occupancies, and 'a' indicates aggregated data.