Time Series Analysis - Assignment 02

Graduate Programme: Data Analysis & Machine-Statistical Learning

***John Maris**; math1p0004*

May 21, 2024

Traffic Volumes and Occupancies Time Series Analysis

This is a continuation of a previous project on traffic volumes and occupancies data from Athens.

Abstract

In this project, we are going to develop and evaluate forecasting models for a time series dataset recorded at 15-minute intervals, which we initially analyzed in a previous analysis. The dataset spans 20 days, where we'll use the first 10 days to train different models and the last 5 days to assess their forecasting accuracy. The complete code for this project can be found in my google colab here.

Overview

1. ***Koyck Distributed Lag Model (Koyck DLM):*** We'll start by implementing the Koyck DLM, using a power-transformed version of occupancies as the explanatory variable, scaled appropriately. We'll determine the best power transformation for the occupancies through a cross-validation technique[1], examining powers up to 10. For the response variable, we'll check if a power transformation (Box-Cox) is beneficial but it's not expected to improve the model.
2. ***Model Evaluation:*** After estimating the Koyck model, we'll evaluate its forecasting accuracy on the test data using the Median Relative Absolute Error (MdRAE), comparing it against a simple persistence model. [2]
3. ***Residual Analysis and ARMA Models:*** We will assess the residuals of our Koyck model for any remaining patterns or structure. If found, we'll develop an ARMA model

[1] ISLR section 5.3.3

[2] Rob J. Hyndman's paper `'Another Look at Forecast Accuracy Metrics for Intermittent Demand'`

for these residuals, comparing models built using traditional for-loop methods (from Metcalfes book) versus those generated by the `auto.arima` function from the `'forecast'` package.

4. ***Two-Stage Modeling:*** Combining the Koyck and ARMA models, we'll forecast the test period data and compare its MdRAE against that of the Koyck model alone to gauge improvements.
5. ***GARCH Model Consideration:*** Ultimately, we'll explore the need for a GARCH model to handle potential heteroscedasticity in the forecasting intervals. We'll use the Mean Scaled Interval Score (MSIS) to compare these GARCH-based intervals with more naive approaches that assume constant variance.
6. ***Autoregressive Distributed Lag (AR-DL) Model:*** Given the limitation that the Koyck model uses occupancies at time t to forecast volumes at time t, we'll repeat the previous analysis using an AR-DL model that only includes information up to time $t-1$. We'll compare the need for ARMA and GARCH models for the residuals and evaluate the forecasting accuracy using MdRAE and MSIS against the Koyck model.
7. ***Simple Regression Models with Daily Periodicity:*** Instead of using the Koyck or AR-DL models, we'll implement a simple regression model that captures daily periodicity, utilizing either dummy variables or a sinusoidal harmonic model. We'll again assess the necessity of ARMA and GARCH models for the residuals and compare the forecasting accuracy using MdRAE and MSIS against both the Koyck and AR distributed lag models.
8. ***Forecast Combination:*** To determine if forecasting accuracy improves by combining forecasts from the AR-DL model and the simple regression model, we'll use the `ForecastComb` package. The combination scheme will be trained on the 5 days between the training and testing periods, focusing on MdRAE. We will once again use ARIMA (and/or SARIMA) models, fitted on the residuals, to achieve better forecasting accuracy. For the combination scheme, we will utilize an automated procedure[3] using mean absolute percentage error as a criterion.

Part 1: Power & Box-Cox transformation for the occupancies and the volumes; Comparing the persistence model against Distributed Lag with Koyck transformation using MdRAE & GMRAE relative erros

We import the data in  (version `4.2.3`).

```
data <- read.csv("APRIL_MIN_15.csv")
data <- data[c('DAY', 'L104_volume', 'L104_occupancy')] # focusing on location 04

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)
>>
```

[3] `auto_combine(criterion = 'MAPE')`

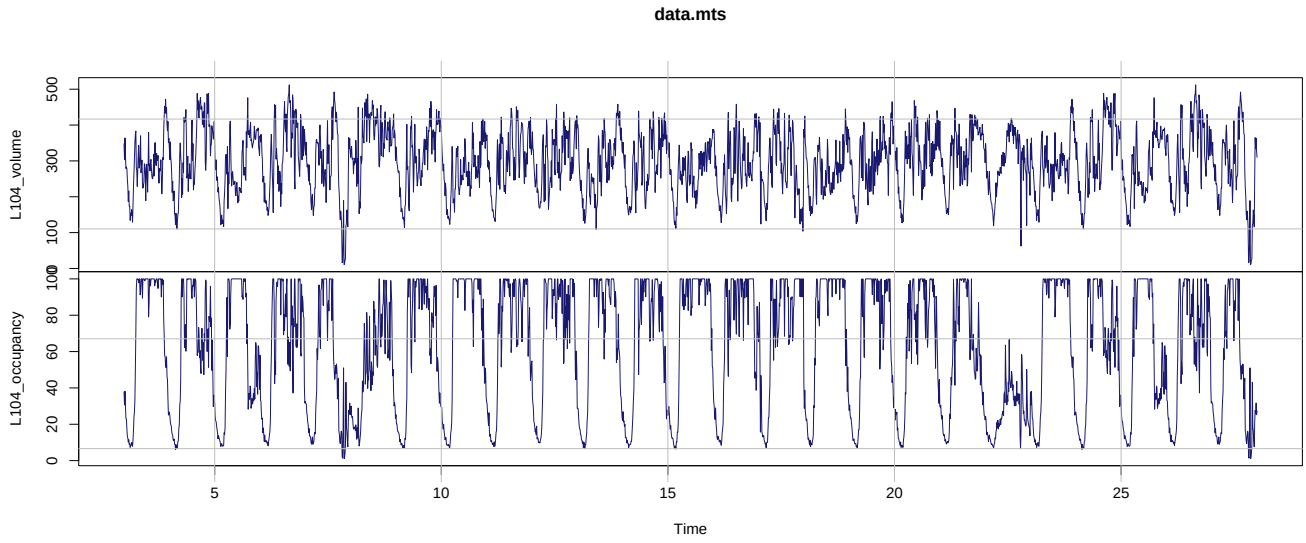


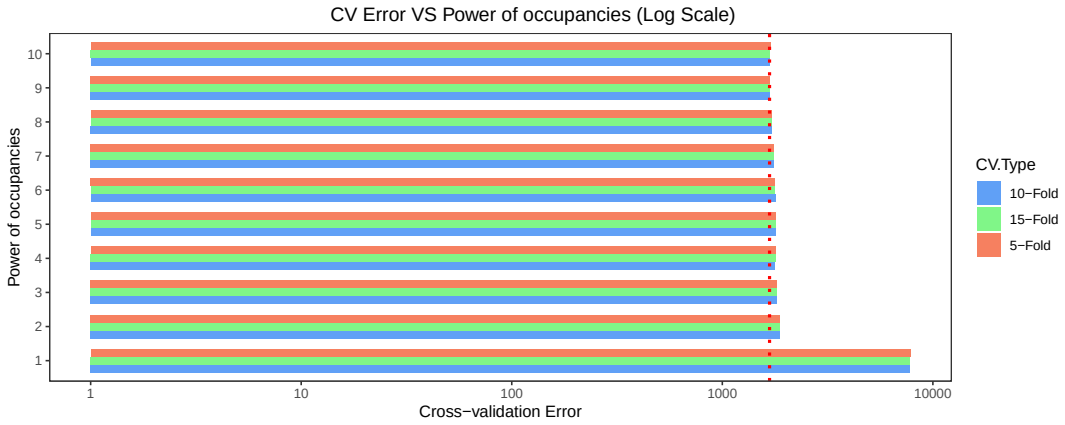
Figure 1: *Volumes and Occupancies time series.*

```
days <- unique(data$DAY) # 10 days (2 weeks without weekends)
first_10.days <- days[seq(1,10)]
last_5.days <- days[seq(15+1, length(days))]
train.ts <- subset(data, DAY %in% first_10.days) # First 2 weeks (minus weekends)
forecast.ts <- subset(data, DAY %in% last_5.days) # Last week (minus weekend)
# Check the split
100 * length(unique(train.ts$DAY))/length(days) # training data %
100 * length(unique(forecast.ts$DAY))/length(days) # testing data  %
>>
50
25
```

We will now investigate which power transformation is optimal for the Occupancies (exogenous variable) using a K-fold cross-validation procedure.

```
library(boot) # for the cv.glm

choose_power.cv <- function (data, K) {
    require(boot)
    set.seed(5656)
    cv.error.10 <- rep(0, 10)
    for (i in 1:10) {
      glm.fit <- glm(L104_volume ~ poly(L104_occupancy, i), data=data)
      cv.error.10[i] <- cv.glm(data, glm.fit, K=K)$delta[1]
    }
    return (cv.error.10)
}
```

```
cv5.errors <- choose_power.cv(data=train.ts, K=5)
cv10.errors <- choose_power.cv(data=train.ts, K=10)
cv15.errors <- choose_power.cv(data=train.ts, K=15)
>>
```




```
print(paste(" 5-Fold-CV;", "Lowest error:", min(cv5.errors), "; power: ", which.min(cv5.errors)))
print(paste("10-Fold-CV;", "Lowest error:", min(cv10.errors), "; power: ", which.min(cv10.errors)))
print(paste("15-Fold-CV;", "Lowest error:", min(cv15.errors), "; power: ", which.min(cv15.errors)))
>>
[1] " 5-Fold-CV; Lowest error: 1684.03314238064 ; power:  9"
[1] "10-Fold-CV; Lowest error: 1676.07905766231 ; power:  10"
[1] "15-Fold-CV; Lowest error: 1682.61567548372 ; power:  9"
```

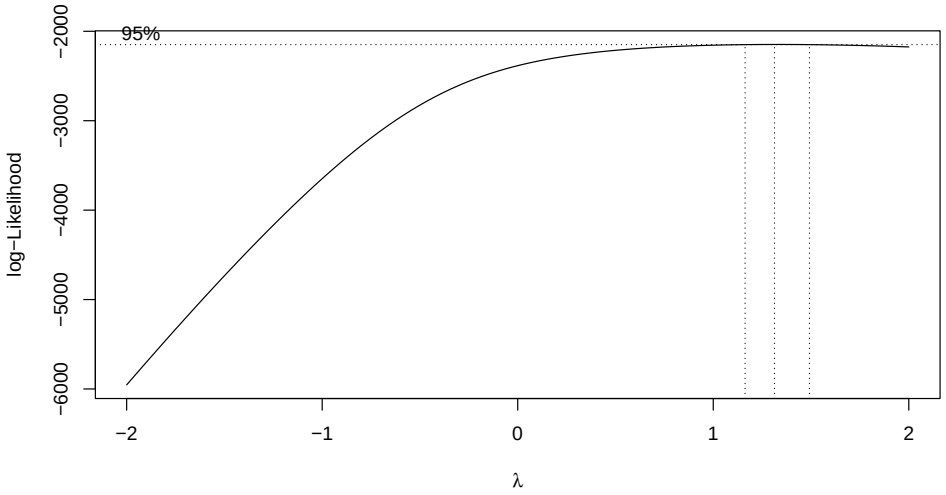
Hence, we choose $power = 9$ and we scale the new occupancies, note that, in general:

- `scaled_train = (train - mean(train)) / sd(train)`
- `scaled_test = (test - mean(train)) / sd(train)`

```
train.ts$L104_occupancy.9 <- as.vector(scale(train.ts$L104_occupancy^9)) # scale the new occ.
train.mean.occ9 <- mean(train.ts$L104_occupancy^9)
train.sd.occ9 <- sd(train.ts$L104_occupancy^9)
forecast.ts$L104_occupancy.9 <- (forecast.ts$L104_occupancy^9 - train.mean.occ9)/train.sd.occ9
```

We will now examine if there is a need for Box-Cox transformation for the Volumes (endogenous variable).

```
library(MASS)
boxcox.result <- boxcox(as.formula("train.ts$L104_volume ~ train.ts$L104_occupancy.9"))
boxcox(as.formula("train.ts$L104_volume ~ train.ts$L104_occupancy.9"))
>>
```




```
lambda.best <- boxcox.result$x[which.max(boxcox.result$y)]
train.ts$boxcox_volume <- ((train.ts$L104_volume^lambda.best) - 1)/lambda.best
lambda.best # close to 1
>>
1.31313131313131
```

Box-Cox transformation is not expected to improve our model.

```
model.original <- glm(L104_volume ~ L104_occupancy.9, data = train.ts)
model.boxcox <- glm(boxcox_volume ~ L104_occupancy.9, data = train.ts)

# k-fold cross-validation
cv_result.original <- cv.glm(train.ts, model.original, K = 10)  # 10-fold CV
cv_result.boxcox <- cv.glm(train.ts, model.boxcox, K = 10)

# delta[1] returns the raw cross-validation estimate of prediction error
print(paste("CV Error Original: ", cv_result.original$delta[1]))
print(paste("CV Error Transformed (BoxCox): ", cv_result.boxcox$delta[1]))
>>
[1] "CV Error Original:  7640.46206464299"
[1] "CV Error Transformed (BoxCox):  259642.470582121"
print(paste("AIC Original: ", AIC(model.original)))
print(paste("AIC Transformed (BoxCox): ", AIC(model.boxcox)))
>>
[2] "AIC Original:  11311.5400490128"
[2] "AIC Transformed (BoxCox):  14696.6804075154"
```

Therefore, a transformation like that won't benefit our analysis. In the code below, we make use of the `dLagM` package in order to build the DL model with the Koyck transformation:

```
library(dLagM)

koyck9.dlm <- koyckDlm(x = train.ts$L104_occupancy.9, y = train.ts$L104_volume)
summary(koyck9.dlm)
koyck9.dlm$geometric.coefficients
>>
"Y ~ (Intercept) + Y.1 + X.t"

Residuals:
     Min       1Q   Median       3Q      Max
-188.226  -31.682   -1.838   31.235  209.137

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 46.47816    6.11123   7.605 6.78e-14 ***
Y.1          0.84711    0.01934  43.802  < 2e-16 ***
X.t          9.06681    2.20528   4.111 4.27e-05 ***
---
Residual standard error: 52.64 on 956 degrees of freedom
Multiple R-Squared: 0.6584, Adjusted R-squared: 0.6577
Wald test: 972.2 on 2 and 956 DF,  p-value: < 2.2e-16
                           alpha     beta       phi
Geometric coefficients:  303.9932 9.066806 0.8471079
```

A reminder: To deal with infinite DLMs, we can use the Koyck transformation. When we apply the 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 the **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 $\nu_t = (\epsilon_t - \phi\epsilon_{t-1})$. Koyck is implemented using Instrumental Variables Regression. The Python code below uses the `IV2SLS` function from the `statsmodels.sandbox.regression.gmm` library.

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.sandbox.regression.gmm import IV2SLS

class InstrumentalVariableRegression:
    def __init__(self):
        self.model = None

    def train(self, train_data: pd.DataFrame, y: str, x: str, lags: list):
        train_data = train_data.dropna()
        endog_train = train_data[y]
        exog_train = sm.add_constant(train_data[x])
        instrument_train = sm.add_constant(train_data[lags])
```

```
        self.model = IV2SLS(endog=endog_train, exog=exog_train,
                            instrument=instrument_train).fit()
        return self.model

    def predict(self, test_data: pd.DataFrame, x: str):
        last_valid_value = test_data[x].ffill().iloc[-1]
        test_data[x] = test_data[x].fillna(last_valid_value)
        exog_test = sm.add_constant(test_data[x])
        predictions = self.model.predict(exog=exog_test)
        return predictions
```

IV2SLS stands for **Instrumental Variables Two-Stage Least Squares**. It is a statistical method used to estimate the parameters of a linear regression model when the model suffers from endogeneity, which occurs when an explanatory variable is correlated with the error term. This correlation can lead to biased and inconsistent estimates.

The IV2SLS method uses instrumentsvariables that are correlated with the endogenous explanatory variables but uncorrelated with the error termto provide consistent estimators. The process involves two stages:

1. **First Stage**: Regress the endogenous explanatory variables on the instruments to obtain fitted values.
2. **Second Stage**: Regress the dependent variable on the fitted values from the first stage.

This method corrects for the endogeneity by ensuring that the explanatory variables used in the second stage are uncorrelated with the error term.

The instrumental variables (IV) estimator[4] is used when the model has endogeneity. The key assumption is that the instruments Z (where Z is an instrument matrix) satisfy $\mathbb{E}[Z'e] = 0$, meaning the instruments are uncorrelated with the error term e.

Estimator:

$$\hat{\beta}_{IV} = \left(\frac{1}{n} \sum_{i=1}^{n} Z_i X_i' \right)^{-1} \left(\frac{1}{n} \sum_{i=1}^{n} Z_i Y_i \right)$$

where:

- $\hat{\beta}_{IV}$ is the IV estimator.
- n is the number of observations.
- Z_i is the instrument for the i-th observation.
- X_i is the endogenous explanatory variable for the i-th observation.
- Y_i is the dependent variable for the i-th observation.

[4] Instrumental Variables Estimator; Hansen's econometrics book chapter 12.9

The demeaned representation[5] for linear regression extends to the IV estimator. It considers the linear projection equation $Y_1 = X'\beta + \alpha + e$ with partitioned instruments, where X does not contain a constant and Z is partitioned such that it does not contain a constant.

Estimator:

$$\hat{\beta}_{IV} = \left(\sum_{i=1}^{n} Z_i (X_i - \bar{X})' \right)^{-1} \left(\sum_{i=1}^{n} Z_i (Y_{1i} - \bar{Y}_1) \right)$$

where:

- $\bar{X}$ is the mean of X.
- $\bar{Y}_1$ is the mean of Y_1.

The Wald estimator[6] is used when the instrument Z is binary. It estimates the effect of a binary instrument on the dependent variable.

Estimator:

$$\beta = \frac{E[Y|Z=1] - E[Y|Z=0]}{E[X|Z=1] - E[X|Z=0]}$$

The sample analog is:

$$\hat{\beta}_{IV} = \frac{\bar{Y}_1 - \bar{Y}_0}{\bar{X}_1 - \bar{X}_0}$$

where:

- $\bar{Y}_1$ and $\bar{Y}_0$ are the sample means of Y when $Z = 1$ and $Z = 0$, respectively.
- $\bar{X}_1$ and $\bar{X}_0$ are the sample means of X when $Z = 1$ and $Z = 0$, respectively.

The 2SLS estimator[7] is a generalization of the IV estimator for cases where there are more instruments than endogenous variables ($\ell \geq k$).

Estimator:

$$\hat{\beta}_{2SLS} = (X' P_Z X)^{-1} X' P_Z Y_1$$

where:

- $P_Z = Z(Z'Z)^{-1}Z'$ is the projection matrix.
- X is the matrix of endogenous explanatory variables.
- Y_1 is the vector of dependent variables.

Now, in order to compare our forecasting models with a benchmark one, we define the persistence model, which is not a trainable model, but a naive one. Our benchmark model will be the value of the volume at one time lag before, $\hat{V}_t = V_{t-1}$.

[5] Hansen 12.10
[6] Hansen 12.11
[7] Hansen 12.12

An alternative to percentages for the calculation of scale-independent measurements involves dividing each error by the error obtained using some benchmark method of forecasting. Let $r_t = \frac{e_t}{e_t^*}$ denote the relative error where e_t is the forecast error and e_t^* is the forecast error obtained from the benchmark method. Usually, the benchmark method is the naive method where F_t is equal to the last observation. Then we can define:

MdRAE: Median Relative Absolute Error is calculated as the median of the absolute values of the relative errors, median($|r_t|$). This metric provides a robust measure of forecast accuracy, as it is less sensitive to extreme values compared to other average measures.

GMRAE: Geometric Mean Relative Absolute Error is calculated as the geometric mean of the absolute values of the relative errors, gmean($|r_t|$). This metric also measures forecast accuracy but takes into account the multiplicative relationships between the errors, which can be more informative when dealing with ratios.

We may define them in R:

```
library('psych') # geometric.mean

# Relative errors (MdRAE; GMRAE)
MdRAE <- function(resids, resids_persistence) {
  r <- resids / resids_persistence
  r[is.infinite(r)] <- NA
  median(na.omit(abs(r)))
}
GMRAE <- function(resids, resids_persistence) {
  r <- resids / resids_persistence
  r[is.infinite(r)] <- NA
  geometric.mean(na.omit(abs(r)))
}
# Symmetric Median Absolute Percentage Error in the M3-competition
# (Makridakis & Hibon, 2000);  sMdAPE is an example of 'Percentage errors'
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
  }
}

library(forecast)

forecast.persistence <- function (test.data=forecast.ts$L104_volume) lag(test.data)
persistence.forecast <- forecast.persistence()
resids.persistence <- forecast.ts$L104_volume - persistence.forecast # omit NA

checkresiduals(resids.persistence)
Ljung-Box test
```

```
data:  Residuals
Q* = 52.87, df = 10, p-value = 7.873e-08

Model df: 0.   Total lags used: 10
>>
```

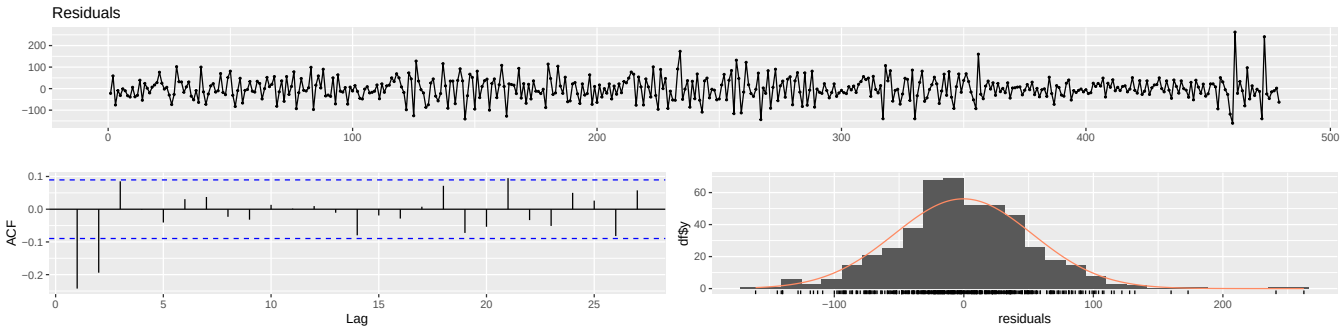
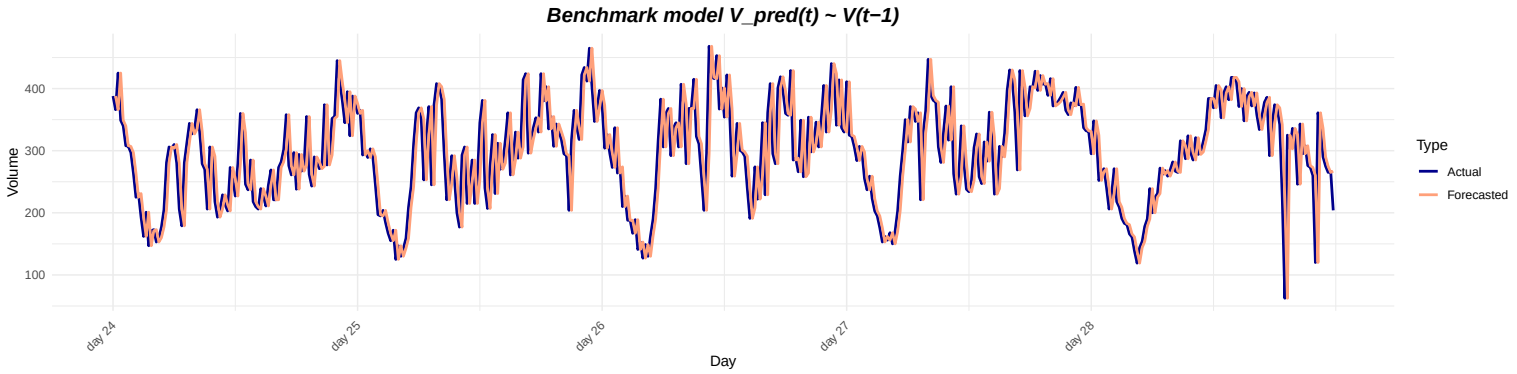


Figure 2: *Persistence model residuals diagnostics.*

```
# This plotting function can be found in my full code on Google Colab.
plot.tss(forecast.values=persistence.forecast, data.actual=forecast.ts,
         main="Benchmark model V(t) ~ V(t-1)")
>>
```



First, we examine the Koyck DL model in comparison with the persistence one:

```
test.resids.koyck9 <- forecast.ts$L104_volume - koyck.forecast.values$Estimate
resids.persistence <- forecast.ts$L104_volume - persistence.forecast

sMdAPE(forecast.ts$L104_volume, persistence.forecast)
MdRAE(test.resids.koyck9, resids.persistence)
GMRAE(test.resids.koyck9, resids.persistence)
>>
```

```
10.95 # Koyck's sMdAPE
1.02 # Koyck's MdRAE
0.92 # Koyck's GMRAE

checkresiduals(koyck.resids)
>>
   Ljung-Box test
data:  Residuals
Q* = 19.817, df = 10, p-value = 0.03103
Model df: 0.   Total lags used: 10
```

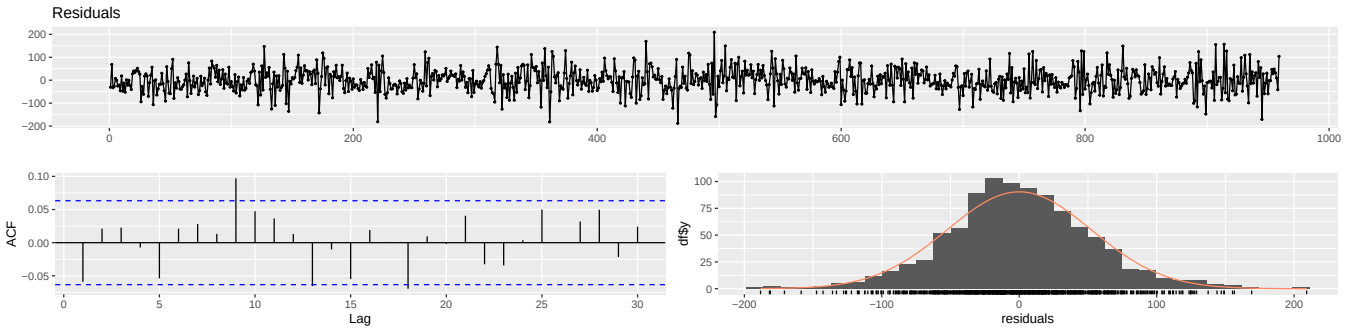
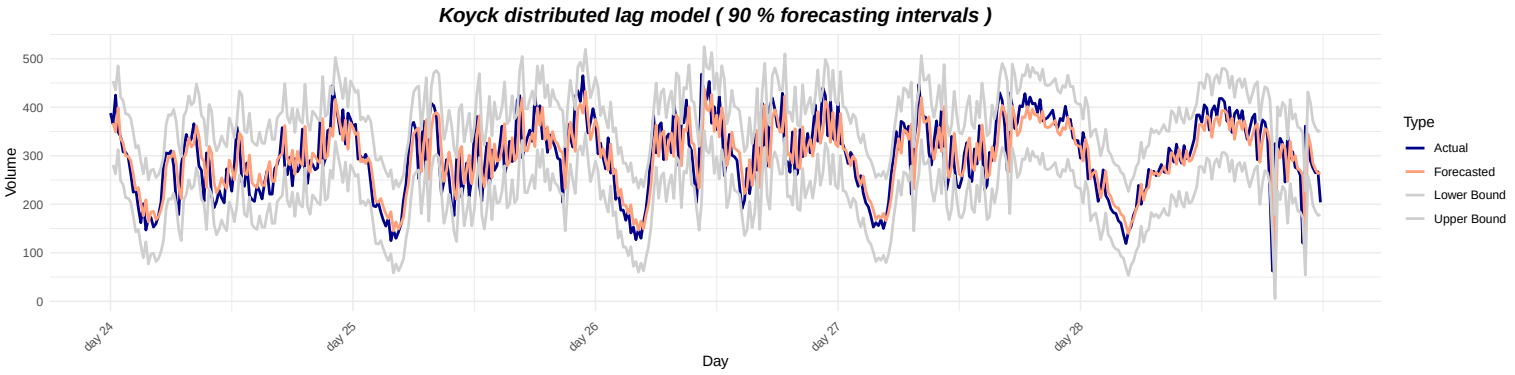


Figure 3: *Koyck DLM residuals diagnostics.*

We are ready now to forecast 15 minutes ahead using the Koyck DLM. Therefore, we build the following function, which also computes the naive prediction intervals at the 90% level:

```
predict.koyckDLM <- function (coeffs, y.test, x.test, model=NA, interval=FALSE, alpha=0.1) {
    koyck.pred <- coeffs[1] + coeffs[2] * lag(y.test) + coeffs[3] * x.test # 15 min ahead forecast
    if (interval) {
        # Naive prediction intervals
        z_value <- qnorm(1 - alpha / 2)
        # calc. the forecasting intervals
        koyck.forecast.lower <- koyck.pred - z_value * sd(residuals(model$model))
        koyck.forecast.upper <- koyck.pred + z_value * sd(residuals(model$model))

        # we need this format for the plotting later on:
        koyck.df <- data.frame("Lower Bound"=koyck.forecast.lower,
                               Estimate=koyck.pred,
                               "Upper Bound"=koyck.forecast.upper,
                               check.names=FALSE)
        return (koyck.df)
    }
    return (koyck.pred)
}

koyck.forecast.values <- predict.koyckDLM(coefficients(koyck9.dlm$model), forecast.ts$L104_volume,
                                          forecast.ts$L104_occupancy.9, model=koyck9.dlm, interval=TRUE)
```

```
plot.tss(forecast.values=koyck.forecast.values, data.actual=forecast.ts,
         main="Koyck distributed lag model", level=0.9, interval=TRUE)
         >>
```



Part 2: ARMA correction for Koyck's DL residuals & GARCH model for the volatilities

As we can observe from the MdRAE value, the Koyck model is not outperforming the persistence model. From Koyck's diagnostics [3], it is most likely that there isn't a remaining structure in the residuals. However, we will investigate if an ARMA model is needed to address any potential remaining structure. To determine the optimal order for the ARMA model, we will use the `auto.arima` function (with $d = 0$) from the `forecast` package. An Alternative is to evaluate every combination of p (AR) and q (MA) orders and select the model with the lowest AIC value.

```
best.order <- c(0, 0, 0)
best.aic <- Inf
max.p <- 5
max.q <- 5
for (i in 0:max.p) for (j in 0:max.q) {
    fit.aic <- AIC(arima(koyck.resids, order = c(i, 0, j)))
    if (fit.aic < best.aic) {
      best.order <- c(i, 0, j)
      best.arma <- arima(koyck.resids, order = best.order)
      best.aic <- fit.aic
    }
}
best.order
best.arma
>>
5  0  4 # ARMA best order
Call:
arima(x = koyck.resids, order = best.order)
Coefficients:
          ar1     ar2     ar3      ar4      ar5     ma1      ma2      ma3
      -0.0873  0.1909  0.1438  -0.8392  -0.1201  0.0308  -0.1873  -0.1294
```

```
s.e.   0.1163  0.0913  0.0671   0.0837   0.0345  0.1140   0.1022   0.0712
         ma4  intercept
      0.8333     0.0394
s.e.  0.0888     1.5159
sigma^2 estimated as 2696:  log likelihood = -5148.85,  aic = 10319.71
```

With `auto.arima`:

```
library(forecast)
best.arma.auto <- auto.arima(koyck.resids, max.p=5, max.q=5, d=0, lambda=NULL,
                             trace=TRUE, approximation=FALSE, ic='aic', stepwise=FALSE)
best.arma.auto
>>
 Best model: ARIMA(1,0,0) with zero mean
Series: koyck.resids
ARIMA(1,0,0) with zero mean
Coefficients:
          ar1
      -0.0592
s.e.   0.0323
sigma^2 = 2755:  log likelihood = -5158.47
AIC=10320.94   AICc=10320.95   BIC=10330.67

AIC(best.arma.auto) - AIC(best.arma)
>>
1.23036285543458
```

The ARMA model selected using the for loop wins in terms of Akaike information, therefore, we will proceed with that model.

Additionally, I propose a further correction for the residuals, which is not yet based on established mathematical results and may require substantial work to potentially become a standard theory. The idea involves tuning a hyperparameter (HP) that we multiply with the predicted residuals (from the ARMA model in this case) to achieve, if not the same, then potentially better predictions.[8]

```
HP_tuning.MPR <- function (fitted.resids, benchmark.resids, resids.correction.preds, train.y,
                           start=1, end=500, early.stopping=TRUE, tol=0.025, trace=FALSE, threshold=100) {
    ### fitted.resids: training residuals of the forecasting model; ###
    ### resids.correction.preds: predictions the model's residuals with a model like ARIMA or GARCH; ###
    ### benchmark.redids: residuals of the persistence model on the testing data; ###
    ### train.y: TS data vector of the training response ###
    ### early.stopping: If the first tuning parameter (mpr) is very close ('tol' threshold)
    ###                 with the last best mpr then the mpr remains invariant; ###
    ### threshold: if the best HP is > 100 then just return 1 as the MPR (unchanged) ###
    best.mdrae = Inf
    best.mpr = 1
    all.mdrae <- list()
```

[8] To verify the general applicability of this idea, we would need to investigate many datasets and a variety of models, followed by hypothesis testing to determine its impact on prediction accuracy. The hypotheses to be tested would be $\mathbb{H}_0$: $HP = 1$ versus $\mathbb{H}_1$: $HP \neq 1$.

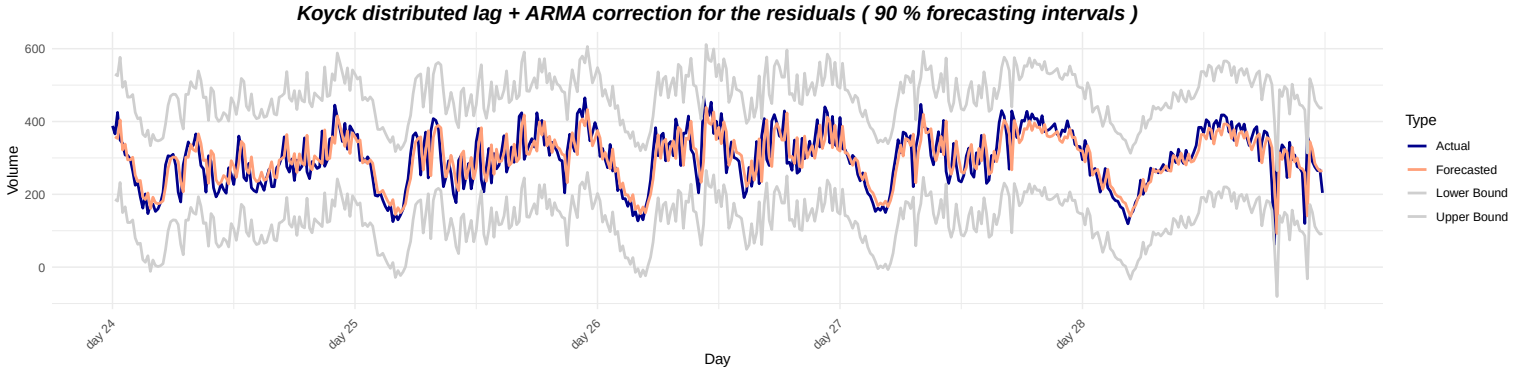
```
    all.mpr <- list()
    j <- 1 # counter
    for (mpr in seq(start, end, length.out = 1000-1)) {
        adj.preds <- as.vector(fitted.resids) + mpr * as.vector(resids.correction.preds)
        curr.residuals <- train.y - adj.preds
        curr.mdrae <- MdRAE(curr.residuals, benchmark.resids)

        if (mpr == 1) ordinary.mdrae <- curr.mdrae # save the original mdrae

        if (curr.mdrae < best.mdrae) {

            best.mdrae <- curr.mdrae
            best.mpr <- mpr

            if (trace) {
                print(paste("Best current const:", best.mpr))
                print(paste("Best current MdRAE:", best.mdrae))
                cat('\n')
            }
            all.mdrae[[j]] = curr.mdrae
            all.mpr[[j]] = best.mpr
            j = j + 1
        }
    }
    # early-stop
    if (early.stopping && (ordinary.mdrae - best.mdrae < tol)) {
        best.mdrae <- ordinary.mdrae
        best.mpr <- start # remains invariant
    }
    # check for extreme values
    if (best.mpr > threshold) {
        best.mpr <- start
        best.mdrae <- ordinary.mdrae

    }
    return(list(best.mpr=best.mpr, best.mdrae=best.mdrae,
                all.mdrae=unlist(all.mdrae), all.mpr=unlist(all.mpr)))
}

h <- dim(forecast.ts)[1] # testing days points length
best.metcalfe.arima.res.forecast <- forecast(best.arma, level=0.9, h=h)
arma.HP <- HP_tuning.MPR(fitted.resids = koyck.resids,
                         benchmark.resids = resids.persistence,
                         resids.correction.preds = as.vector(as.data.frame(
                             best.metcalfe.arima.res.forecast)[['Point Forecast']]),
                         train.y = train.ts$L104_volume)
arma.HP$best.mpr
>>
1
```

The tuning HP turns out to be 1, making it invariant. In most cases, given our predefined con-

straints, the HP remains 1. However, there are models where we find evidence against the null hypothesis $\mathbb{H}_0$: HP $= 1$. Since this theory is not yet thoroughly examined, we will stick to the original analysis, assuming the HP is always 1.

Let's find out if the ARMA is indeed beneficial for the Koyck DL model.

```
alpha <- 0.1 # 90% forecasting intercals (set alpha <- 0.05 for 95%)
z_value <- qnorm(1 - alpha/2)

best.metcalfe.arima.res.forecast <- forecast(best.arma, level=0.9, h=h)
arma.test.pred <- best.metcalfe.arima.res.forecast$mean + as.vector(koyck.forecast.values$Estimate)

arma.test.pred.lower <- as.vector(koyck.forecast.values$Lower) +
                        best.metcalfe.arima.res.forecast$lower[,1]
arma.test.pred.upper <- as.vector(koyck.forecast.values$Upper) +
                        best.metcalfe.arima.res.forecast$upper[,1]
arma.koyck.df <- data.frame("Lower Bound" = arma.test.pred.lower, Estimate=arma.test.pred,
                            "Upper Bound" = arma.test.pred.upper,  check.names = FALSE)
plot.tss(forecast.values=arma.koyck.df, data.actual=forecast.ts,
         main="Koyck distributed lag + ARMA correction for the residuals", interval=TRUE, level=0.9)
>>
```




```
MdRAE(forecast.ts$L104_volume - as.vector(arma.test.pred), resids.persistence)
GMRAE(forecast.ts$L104_volume - as.vector(arma.test.pred), resids.persistence)
>>
1.02959828982
0.93021864882559
# whithout ARMA:
MdRAE(test.resids.koyck9, resids.persistence) # just koyck predictions
GMRAE(test.resids.koyck9, resids.persistence)
>>
1.02790813899327
0.918399809012247
```

There is no substantial difference in the MdRAE; therefore, the need for an ARMA model is not required. In the code below, we perform automatic selection of the best GARCH model based on

a specified information criterion (Akaike) using the `rugarch` package. We define the `auto.garch` function, which searches for the optimal GARCH and ARMA orders within predefined limits. It iterates through all combinations of GARCH and ARMA orders, fits GARCH models, evaluates them using the information criterion, and updates the best model based on the lowest AIC value.

Since the ARMA residuals are not useful for our model, we use only the residuals from the Koyck model, fitting the GARCH model with the `auto.garch` function, setting the maximum ARMA order to zero.

```
auto.garch <- function (data, max.garch.order=c(4, 4), max.arma.order=c(4,4),
                        ic="Akaike", trace=FALSE, include.mean=FALSE, distribution.model="norm")  {
    # data: uni. ts object of the training data only.
    # max.garch.order & max.arma.order: 2dvecs; max orders to examine in the for loop.
    # ic: character; indicates which info. criterion to utilize: Akaike||Bayes||Shibata||Hannan-Quinn.
    require(rugarch)
    mgo <- max.garch.order
    mmo <- max.arma.order
    best.aic <- Inf
    start.time <- Sys.time()
    # total iterations for the progress bar
    total.iterations <- (mgo[1] * mgo[2] * (mmo[1] + 1) * (mmo[2] + 1))
    progress.bar <- txtProgressBar(min=0, max=total.iterations, style=3)
    iteration.count <- 0

    for (p1 in 1:mgo[1]) for (q1 in 1:mgo[2]) for (p2 in 0:mmo[1]) for (q2 in 0:mmo[2])  {
        spec <- ugarchspec(variance.model=list(model="sGARCH", garchOrder=c(p1,q1)),
                           mean.model=list(armaOrder=c(p2,q2), include.mean = include.mean),
                           distribution.model=distribution.model)
        garch.model <- ugarchfit(data = as.ts(data), spec = spec)
        fit.aic <- as.data.frame(t(infocriteria(garch.model)))[[ic]]
        if (trace) print(paste0("AIC of garch.order: (", p1,',',q1,
                                ") & arma.order: (", p2,',',q2,"): ", round(fit.aic, 4)))
        if (fit.aic < best.aic) {
          best.garch.order <- c(p1, q1)
          best.arma.order <- c(p2, q2)

          spec <- ugarchspec(variance.model=list(model="sGARCH", garchOrder=best.garch.order),
                     mean.model=list(armaOrder=best.arma.order, include.mean = include.mean),
                     distribution.model=distribution.model)
          garch.model <- ugarchfit(data = as.ts(data), spec = spec)

          best.aic <- fit.aic
        }
        iteration.count <- iteration.count + 1 # update progress bar
        setTxtProgressBar(progress.bar, iteration.count)
    }
    close(progress.bar)
    end.time <- Sys.time()
    time.taken <- round(end.time - start.time, 2)
```

```
    return(list(best.garch.model=garch.model, best.aic=best.aic, time=time.taken,
                best.garch.order=best.garch.order, best.arma.order=best.arma.order))
}

garch.auto.model <- auto.garch(data=as.vector(koyck.resids), max.arma.order=c(0,0))
>>
garch.auto.model$best.garch.order
>>
1   1

plot(garch.auto.model$best.garch.model, which = 'all')
>>
```

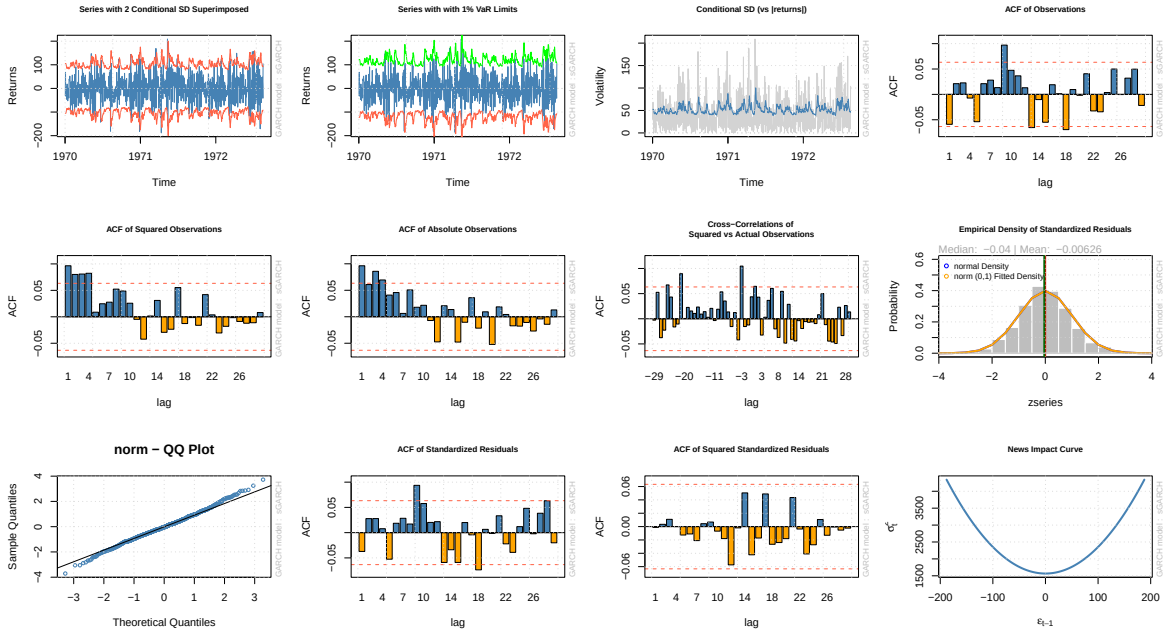


Figure 4: *Koyck's GARCH(1,1) diagnostics.*

We will now make use of the `garch` function from the `tseries` package from `R`.

```
library(tseries)
garch.model <- garch(as.ts(as.vector(koyck.resids)),
                     order=c(1, 1), grad = "numerical", trace = FALSE)
garch.model
>>
Coefficient(s):
      a0        a1        b1
278.1966    0.1170    0.7872
```

```
print(confint(garch.model))
>>
         2.5 %      97.5 %
a0 96.80350950 459.5897871
a1  0.06394752   0.1700839
b1  0.69648614   0.8778299
```

The GARCH (Generalized Autoregressive Conditional Heteroskedasticity) model extends the ARCH (Autoregressive Conditional Heteroskedasticity) model by including lagged values of the conditional variance. An ARCH(p) process models the variance of a time series as a function of the past p squared deviations from the mean. The GARCH(p, q) process generalizes this by also incorporating q lagged values of past conditional variances.

Formally, a GARCH(p, q) process is defined by:

$$\epsilon_t = w_t \sqrt{h_t},$$

where ϵ_t represents the error term or residual at time t, and w_t is white noise with zero mean and unit variance. The conditional variance h_t is given by:

$$h_t = \alpha_0 + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j h_{t-j},$$

where α_0 is a constant, α_i are the coefficients of the lagged squared residuals (the ARCH terms), and β_j are the coefficients of the lagged conditional variances (the GARCH terms). The parameters α_i and β_j must be non-negative to ensure a positive conditional variance h_t.

The GARCH(1, 1) model is a particular case of the GARCH(p, q) model with $p = 1$ and $q = 1$. This model is widely used in financial econometrics due to its simplicity and effectiveness in capturing volatility clustering, a prevalent feature in financial time series.

The GARCH$(1, 1)$ model is specified by the equations:

$$\epsilon_t = w_t \sqrt{h_t},$$

$$\hat{h}_t = \hat{\alpha}_0 + \hat{\alpha}_1 \epsilon_{t-1}^2 + \hat{\beta}_1 \hat{h}_{t-1},$$

where $\hat{\alpha}_0$ is a constant, $\hat{\alpha}_1$ is the coefficient for the lagged squared residual (the ARCH term), and $\hat{\beta}_1$ is the coefficient for the lagged conditional variance (the GARCH term).

In this model, the conditional variance h_t depends on the immediate past squared residual and the immediate past conditional variance. This structure enables the GARCH$(1, 1)$ model to effectively model periods of volatility clustering, where high-volatility periods tend to follow high-volatility periods, and low-volatility periods follow low-volatility periods.

For a GARCH$(1, 1)$ model, the standardized residuals $\hat{w}_t$ are calculated as:

$$\hat{w}_t = \frac{\epsilon_t}{\sqrt{\hat{h}_t}},$$

where $\hat{h}_t$ is the estimated conditional variance. If the GARCH$(1,1)$ model is appropriate, the standardized residual series $\hat{w}_t$ should resemble white noise with zero mean and unit variance.

```
garch.res <- garch.model$res[-1] # first is <NA>
acf(garch.res)
acf(garch.res^2)
>>
```

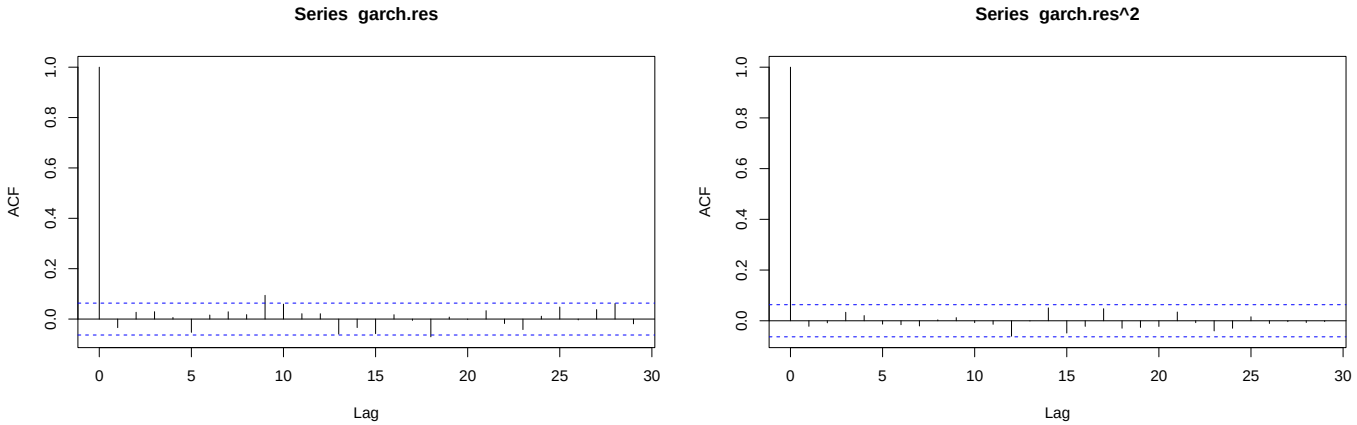
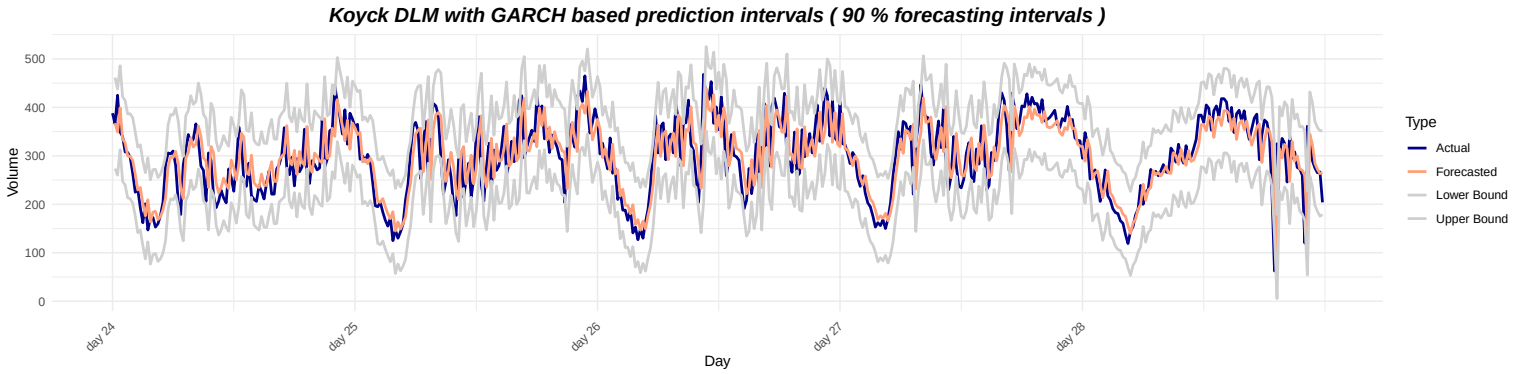


Figure 5: *Koyck's GARCH(1,1) residuals diagnostics.*

The function below computes the standardized residuals $\hat{w}_t$ using the GARCH model's coefficients:

```
garch.resids <- function(res_t, coefs, fhorizon) {
  res_t <- na.omit(res_t)
  h <- numeric(fhorizon)
  h[1] <- 0
  for (i in 2:fhorizon) {
    h[i] <- coefs[["a0"]] + coefs[["a1"]] * (res_t[i-1]^2) + coefs[["b1"]] * h[i-1]
  }
  w <- res_t / sqrt(h)
  w[1] = NA
  # return the standardized residuals
  return (w)
}

koyck.garch.pred.res <- garch.resids(test.resids.koyck9, coef(garch.model), fhorizon=h)

alpha <- 0.1 # 90% FI
z_value <- qnorm(1 - alpha/2)
# Koyck + volatilities-GARCH
garch.upper <- koyck.forecast.values$Upper + z_value * abs(koyck.garch.pred.res)
garch.lower <- koyck.forecast.values$Lower - z_value * abs(koyck.garch.pred.res)
garch.koyck.df <- data.frame("Lower Bound" = garch.lower, Estimate=koyck.forecast.values$Estimate,
```

```
                              "Upper Bound" = garch.upper,  check.names = FALSE)
plot.tss(forecast.values=garch.koyck.df,
         data.actual=forecast.ts,
         main="Koyck DLM with GARCH based prediction intervals",
         interval=TRUE, level=0.9)
>>
```



For the Mean scaled interval score (MSIS) we will utilize the 'smis' function from the tsRNN package. According to M4 Forecasting competition, smis is given by:

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

```
# Koyck + GARCH
smis(data = ts(train.ts$L104_volume[-1]),
            actual = ts(forecast.ts$L104_volume[-1]),
            lower = as.vector(garch.koyck.df$Lower)[-1],
            upper = as.vector(garch.koyck.df$Upper)[-1],
            m = freq, level = 0.9)
>>
3.17603 # a bit smaller when using garch
# Just Koyck
smis(data = ts(train.ts$L104_volume[-1]),
            actual = ts(forecast.ts$L104_volume[-1]),
            lower = as.vector(koyck.forecast.values$Lower)[-1],
            upper = as.vector(koyck.forecast.values$Upper)[-1],
            m = freq, level = 0.9)
>>
3.17672
```

Autoregressive Distributed Lag (ARDL) + ARMA residuals + GARCH

Unfortunately, the Koyck model utilizes occupancies at time t to forecast volumes at time t. In a realistic setting we would like to exploit information up to time $t-1$. We will repeat the previous analysis using an autoregressive distributed lag (ARDL) model which does not include occupancies

at time t.

The autoregressive DL model is a flexible and parsimonious infinite distributed lag model. The model ARDL(p, q) is written as

$$Y_t = \mu + \beta_0 X_t + \beta_1 X_{t-1} + \cdots + \beta_p X_{t-p} + \gamma_1 Y_{t-1} + \cdots + \gamma_q Y_{t-q} + e_t.$$

In the following function, we have created an automated procedure for tuning the p and q parameters of the ARDL model.

```
auto.ardl <- function (ardl.formula, train.data, X.t=FALSE, max.p=5, max.q=5) {
    require(dLagM)
    best.order <- c(0, 0)
    best.aic <- Inf
    X.str <- all.vars(ardl.formula)[2]
    remove_X.t <- c()
    if (!X.t) remove_X.t <- list(p = setNames(list(c(0)), X.str), q = c()) # remove X at time t
    quiet <- function(x) { # hide showing all the aics
        sink(tempfile())
        on.exit(sink())
        invisible(force(x))
    }
    start.time <- Sys.time()
    total.iterations <- max.p * max.q  # total iterations for the progress bar
    progress.bar <- txtProgressBar(min=0, max=total.iterations, style=3)
    j <- 0 # count the iter

    for (p in 1:max.p) for (q in 1:max.q) {
        fit.aic <- quiet(AIC(ardlDlm(ardl.formula, train.data, p=p, q=q, remove=remove_X.t)))
        if (fit.aic < best.aic) {
          best.order <- c(p, q)
          ardl.model <- ardlDlm(ardl.formula, train.data, p=best.order[1], q=best.order[2], remove=remove_X.t)
          best.aic <- fit.aic
        }
        j <- j + 1 # update progress bar
        setTxtProgressBar(progress.bar, j)
    }
    close(progress.bar)
    end.time <- Sys.time()
    time.taken <- round(end.time - start.time, 2)

    return(list(best.ardl = ardl.model, best.aic=best.aic, best.order=best.order, time=time.taken))
}
```

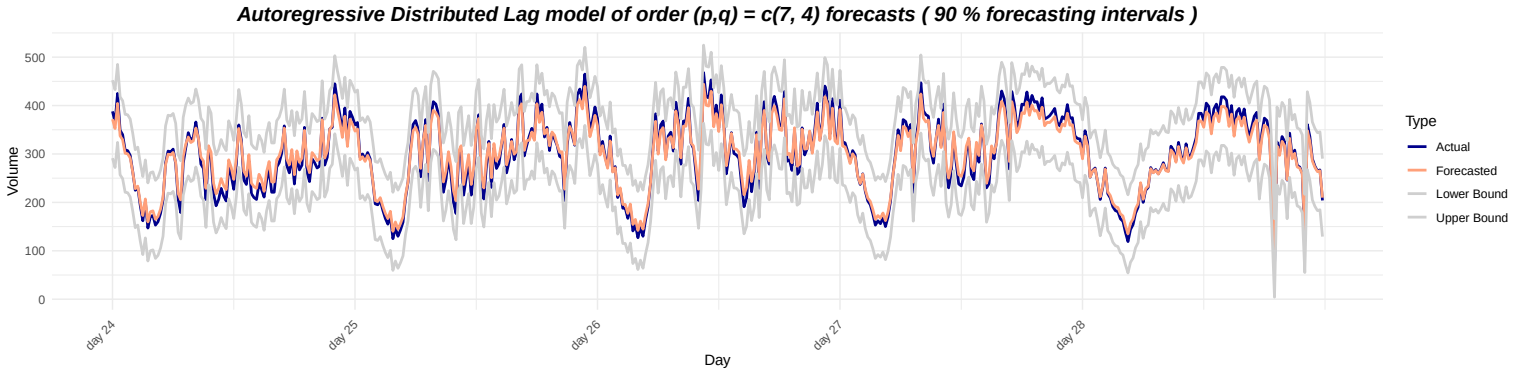
We fit the auro-ARDL model

```
ardl.model <- auto.ardl(as.formula("L104_volume ~ L104_occupancy.9"), train.ts, max.p=7, max.q=7)
ardl.model$best.ardl$model
ardl.model$best.order
ardl.model$best.aic
>>
```

```
Time series regression with "ts" data:
Start = 8, End = 960
Call: dynlm(formula = as.formula(model.text), data = data)
Coefficients:
       (Intercept)  L104_occupancy.9.1  L104_occupancy.9.2  L104_occupancy.9.3
          36.32779            -3.09219             9.68282            11.52641
L104_occupancy.9.4  L104_occupancy.9.5  L104_occupancy.9.6  L104_occupancy.9.7
          -5.92678             1.57846             2.59063            -8.85880
     L104_volume.1       L104_volume.2       L104_volume.3       L104_volume.4
           0.70109             0.14709             0.09359            -0.06118
>>
7   4  # ARDL order
>>
10128.9591901626  # Best Akaike
```

We can use the `predict` function from `R` to forecast and create the naive prediction intervals:

```
ardl.forecast_ <- predict(ardl.model$best.ardl$model,
                          forecast.ts, level=0.9,
                          interval="prediction")
ardl.forecast <- data.frame("Lower Bound" = as.vector(ardl.forecast_[,2]),
                            Estimate=as.vector(ardl.forecast_[,1]),
                           "Upper Bound" = as.vector(ardl.forecast_[,3]), check.names=FALSE)
plot.tss(forecast.values=ardl.forecast,
         data.actual=forecast.ts,
         main=paste("Autoregressive Distributed Lag model of order (p,q) =",
                    list(ardl.model$best.order), "forecasts"),
         interval=TRUE, level=0.9)
>>
```



Unlike the Koyck DLM, the prediction intervals for the ARDL model for the volumes are above zero, as they should be realistically.

```
MAE(ardl.forecast$Estimate, forecast.ts$L104_volume)
RMSE(ardl.forecast$Estimate, forecast.ts$L104_volume)
sMdAPE(ardl.forecast$Estimate, forecast.ts$L104_volume)
```

```
>>
11.521
13.05
3.842

ardl.train.resids <- as.vector(ardl.model$best.ardl$model$resid) # 953 in length
ardl.test.resids <- forecast.ts$L104_volume - ardl.forecast$Estimate
MdRAE(ardl.test.resids, resids.persistence)
GMRAE(ardl.test.resids, resids.persistence)
>>
0.3241
0.3332

checkresiduals(ardl.train.resids)
>>
   Ljung-Box test
data:  Residuals
Q* = 9.1261, df = 10, p-value = 0.5202
Model df: 0.   Total lags used: 10
```

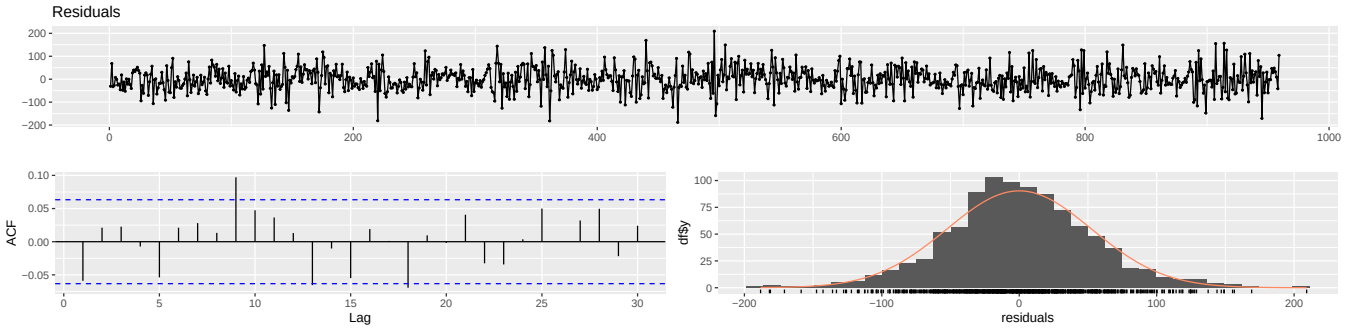


Figure 6: *ARDL(7,4) residuals diagnostics.*

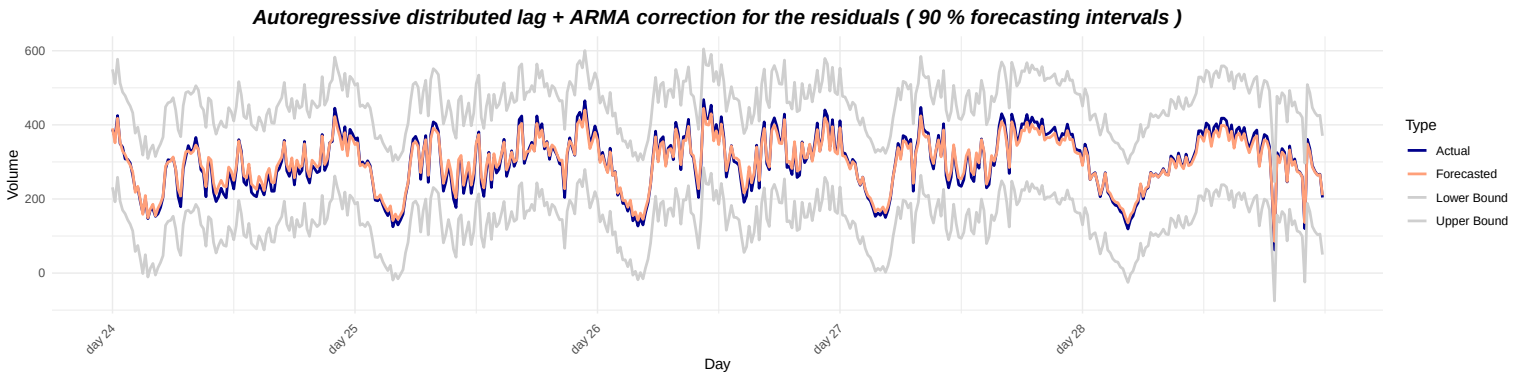
We will now investigate the need for an ARMA model for the ARDL's residuals.

```
best.order <- c(0, 0, 0)
best.aic <- Inf
max.p <- 4
max.q <- 4
for (i in 0:max.p) for (j in 0:max.q) {
    fit.aic <- AIC(arima(ardl.train.resids, order = c(i, 0, j)))
    if (fit.aic < best.aic) {
      best.order <- c(i, 0, j)
      best.arma <- arima(ardl.train.resids, order = best.order)
      best.aic <- fit.aic
    }
}
best.order
```

```
best.arma
>>
4   0   4
>>
arima(x = ardl.train.resids, order = best.order)

Coefficients:
         ar1     ar2     ar3      ar4      ma1      ma2      ma3     ma4
      0.4229  0.7023  0.4507  -0.8806  -0.4187  -0.7471  -0.4310  0.9407
s.e.  0.0290  0.0295  0.0276   0.0398   0.0253   0.0223   0.0205  0.0278
      intercept
         0.4129
s.e.     1.7485
sigma^2 estimated as 2287:  log likelihood = -5038.47,  aic = 10096.94
```

We shall now examine if this benefits our model.

```
alpha <- 0.1 # 90% FI
z_value <- qnorm(1 - alpha/2)

best.metcalfe.arma.res.forecast_ardl <- forecast(best.arma, level=0.9, h=h)
arma.test.pred <- best.metcalfe.arma.res.forecast_ardl$mean + as.vector(ardl.forecast$Estimate)
arma.test.pred.lower <- as.vector(ardl.forecast$Lower) +
                        best.metcalfe.arma.res.forecast_ardl$lower[,1]
arma.test.pred.upper <- as.vector(ardl.forecast$Upper) +
                        best.metcalfe.arma.res.forecast_ardl$upper[,1]

arma.ardl.df <- data.frame("Lower Bound" = arma.test.pred.lower, Estimate=arma.test.pred,
                      "Upper Bound" = arma.test.pred.upper,  check.names = FALSE)
plot.tss(forecast.values=arma.ardl.df,
         data.actual=forecast.ts,
         main="Autoregressive distributed lag + ARMA correction for the residuals",
         interval=TRUE, level=0.9)
>>
```

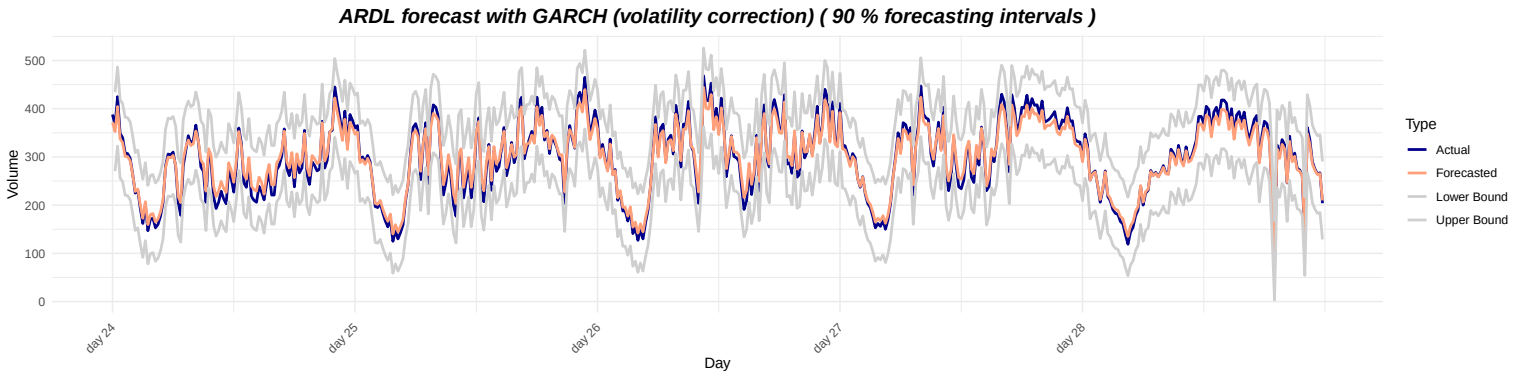



```
MdRAE(forecast.ts$L104_volume - as.vector(arma.test.pred), resids.persistence)
GMRAE(forecast.ts$L104_volume - as.vector(arma.test.pred), resids.persistence)
```

```
>>
0.3231
0.3266

MdRAE(ardl.test.resids, resids.persistence) # no arma
GMRAE(ardl.test.resids, resids.persistence) # no arma
>>
0.3241
0.3332
```

Ultimately, we will try to improve the forecasting CI by addressing a GARCH model (volatilities).

```
garch.model <- garch(as.ts(as.vector(ardl.train.resids)),
                     order=c(1, 1), grad = "numerical", trace = FALSE)
garch.model
confint(garch.model)
>>
Coefficient(s):
       a0         a1         b1
261.63834    0.09755    0.79410
         2.5 %      97.5 %
a0 66.25662118 457.0200592
a1  0.04551546   0.1495923
b1  0.68484669   0.9033577

ardl.garch.pred.res <- garch.resids(ardl.test.resids, coef(garch.model), fhorizon=h)
garch.upper.ardl <- ardl.forecast$Upper + z_value * abs(ardl.garch.pred.res)
garch.lower.ardl <- ardl.forecast$Lower - z_value * abs(ardl.garch.pred.res)

garch.df.ardl <- data.frame("Lower Bound" = garch.lower.ardl, Estimate=ardl.forecast$Estimate,
                            "Upper Bound" = garch.upper.ardl,  check.names = FALSE)
plot.tss(forecast.values=garch.df.ardl,
         data.actual=forecast.ts,
         main="ARDL forecast with GARCH (volatility correction)",
         interval=TRUE, level=0.9)
>>
```



Note that the estimation process does not change when using GARCH; only the prediction intervals are expected to be affected. For the ***MSIS***:

```
# Just ARDL
smis(data = ts(train.ts$L104_volume),
             actual = ts(forecast.ts$L104_volume),
             lower = as.vector(ardl.forecast$Lower),
             upper = as.vector(ardl.forecast$Upper),
             m = freq, level = 0.9)
>>
2.2023

# ARDL + ARMA
smis(data = ts(train.ts$L104_volume),
             actual = ts(forecast.ts$L104_volume),
             lower = as.vector(arma.test.pred.lower),
             upper = as.vector(arma.test.pred.upper),
             m = freq, level = 0.9)
>>
4.3839

# ARDL + GARCH
smis(data = ts(train.ts$L104_volume[-1]),
             actual = ts(forecast.ts$L104_volume[-1]),
             lower = garch.df.ardl$Lower[-1],
             upper = garch.df.ardl$Upper[-1],
             m = freq, level = 0.9)
>>
2.2144
```

From the above results, we can see that the GARCH model did not significantly improve the MSIS; it remains almost the same as before.

Part 3: Daily periodicity models - Additive indicators with dummy variables & Harmonic seasonal models with dynamic regression (Quantile & Least squares)

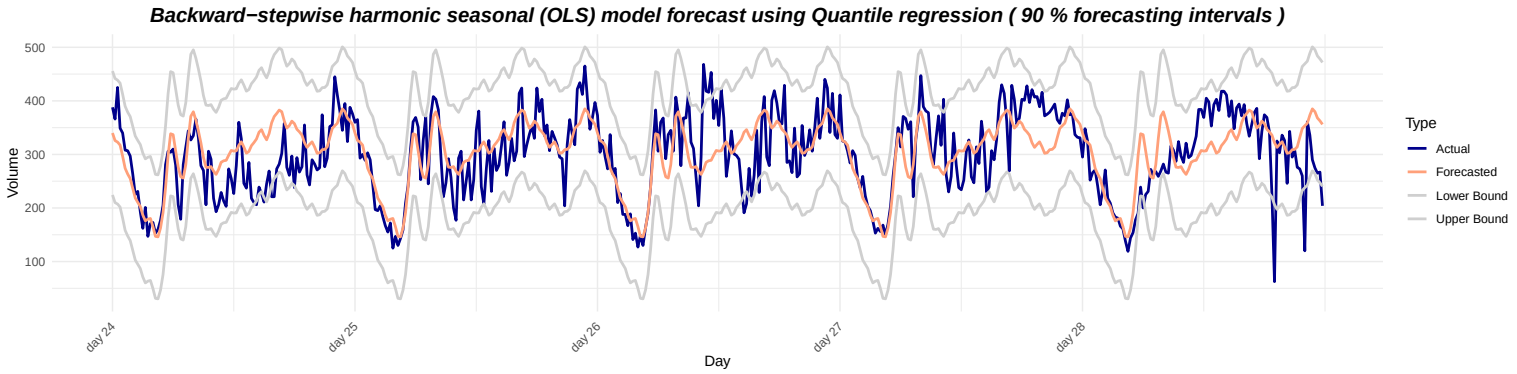
Instead of using the Koyck or ARDL models, we will employ a simple regression model that captures daily periodicity. This approach will not utilize previous information on traffic occupancies. Instead, we will use a specification based on dummy variables or a model incorporating sine and cosine functions.

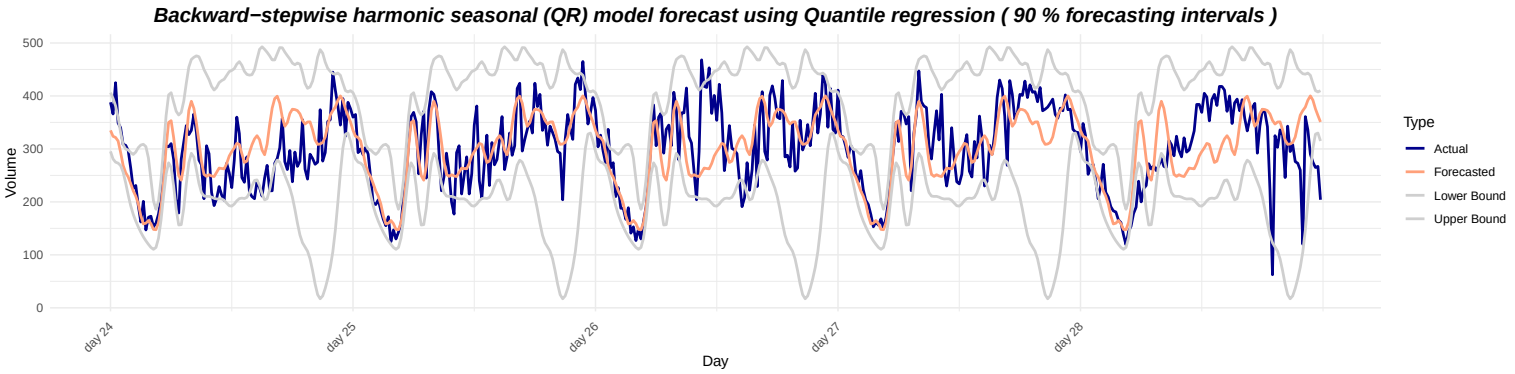
We are going to use a harmonic seasonal model with OLS and Quantile regression; we are using dynamic regression for both OLS (`dynlm`) and QR (`dynrq`), and then perform backward-stepwise model building to select the significant sinusoidal terms using AIC.

```
train.data <- train.ts
test.data <- forecast.ts
```

```r
train.data$time <- rep(1:freq, length.out = nrow(train.data))
test.data$time <- rep(1:freq, length.out = nrow(test.data))

# Frequency is equal to 96 (15-minute intervals in a 24-hour day)
s <- floor(freq / 2) - 1 # =47

# Add sinusoidal terms to the data (data aug.)
for (j in 1:s) {
    train.data[[paste0("sin_", j)]] <- sin(2 * pi * j * train.data$time / freq)
    train.data[[paste0("cos_", j)]] <- cos(2 * pi * j * train.data$time / freq)
    test.data[[paste0("sin_", j)]] <- sin(2 * pi * j * test.data$time / freq)
    test.data[[paste0("cos_", j)]] <- cos(2 * pi * j * test.data$time / freq)
}

library(quantreg) # QR
formula.harmonic <- as.formula(paste("L104_volume ~", paste(paste0("sin_", 1:s), collapse = " + "),
                                     "+", paste(paste0("cos_", 1:s), collapse = " + ")))
model_harmonic.lsq <- dynlm(formula.harmonic, data = train.data)
model_harmonic.qr <- dynrq(formula.harmonic, data = train.data, tau = 0.5)
# lower and upper preds for the QR model
alpha <- 0.1 # 90% FI
lower.q <- alpha/2
upper.q <- 1 - alpha/2
model_harmonic.qr.lower <- dynrq(formula.harmonic, data = train.data, tau = lower.q)
model_harmonic.qr.upper <- dynrq(formula.harmonic, data = train.data, tau = upper.q)
# Forecast on the testing days
predictions_harmonic.ols <- predict(model_harmonic.lsq, newdata = test.data,
                                    interval = "prediction", level = 0.9)
harmonic_ols.df <- data.frame("Lower Bound"=as.vector(predictions_harmonic.ols[,2]),
                              Estimate=as.vector(predictions_harmonic.ols[,1]),
                              "Upper Bound"=as.vector(predictions_harmonic.ols[,3]),
                              check.names = FALSE)
# for the QR
predictions_harmonic.qr <- predict(model_harmonic.qr, newdata = test.data)
predictions_harmonic.qr.lower <- predict(model_harmonic.qr.lower, newdata = test.data)
predictions_harmonic.qr.upper <- predict(model_harmonic.qr.upper, newdata = test.data)
# QR MSIS
qr.msis <- smis(data = ts(train.ts$L104_volume),
                actual = ts(forecast.ts$L104_volume),
                lower = as.vector(predictions_harmonic.qr.lower),
                upper = as.vector(predictions_harmonic.qr.upper),
                m = freq, level = 0.9)
harmonic_qr.df <- data.frame("Lower Bound"=as.vector(predictions_harmonic.qr.lower),
                              Estimate=as.vector(predictions_harmonic.qr),
                              "Upper Bound"=as.vector(predictions_harmonic.qr.upper),
                              check.names = FALSE)

 library(olsrr) # backwards stepwise
harmonic.BAIC <- ols_step_backward_aic(model_harmonic.lsq)
```

```
harmonic_BAIC.preds <- predict(harmonic.BAIC$model, newdata=test.data,
                               interval = "prediction", level = 0.9)

harmonic_BAIC.df <- data.frame("Lower Bound"=as.vector(harmonic_BAIC.preds[,2]),
                               Estimate=as.vector(harmonic_BAIC.preds[,1]),
                               "Upper Bound"=as.vector(harmonic_BAIC.preds[,3]),
                               check.names = FALSE)
harmonic.BAIC$model$terms
>>

L104_volume ~ sin_1 + sin_2 + sin_3 + sin_4 + sin_5 + sin_7 +
    sin_8 + sin_9 + sin_11 + sin_12 + sin_13 + sin_14 + cos_1 +
    cos_2 + cos_3 + cos_4 + cos_5 + cos_7 + cos_8 + cos_10 +
    cos_12 + cos_14 + cos_24

harmonic_rq.BAIC <- dynrq(formula(harmonic.BAIC$model$terms), train.data, tau=0.5)
harmonic_rq.BAIC.lower <- dynrq(formula(harmonic.BAIC$model$terms), train.data, tau=alpha/2)
harmonic_rq.BAIC.upper <- dynrq(formula(harmonic.BAIC$model$terms), train.data, tau=1-alpha/2)
harmonic_rq_BAIC.preds <- predict(harmonic_rq.BAIC, newdata=test.data)
harmonic_rq_BAIC.preds.lower <- predict(harmonic_rq.BAIC.lower, newdata=test.data)
harmonic_rq_BAIC.preds.upper <- predict(harmonic_rq.BAIC.upper, newdata=test.data)

harmonic_rq_BAIC.df <- data.frame("Lower Bound"=as.vector(harmonic_rq_BAIC.preds.lower),
                               Estimate=as.vector(harmonic_rq_BAIC.preds),
                               "Upper Bound"=as.vector(harmonic_rq_BAIC.preds.upper),
                               check.names = FALSE)
plot.tss(forecast.values=harmonic_BAIC.df, data.actual=forecast.ts, interval=TRUE, level=0.9,
         main="Backward-stepwise harmonic seasonal (OLS) model forecast using Quantile regression")
plot.tss(forecast.values=harmonic_rq_BAIC.df, data.actual=forecast.ts, interval=TRUE, level=0.9,
         main="Backward-stepwise harmonic seasonal (QR) model forecast using Quantile regression")
>>
```





For the **MdRAE** and the **GMRAE**:

```
harmonic.ols.test_resids <- forecast.ts$L104_volume - as.vector(predictions_harmonic.ols[,1])
# Harmonic model using OLS
MdRAE(harmonic.ols.test_resids, resids.persistence)
GMRAE(harmonic.ols.test_resids, resids.persistence)
>>
1.0026
1.0333

harmonic.qr.test_resids <- forecast.ts$L104_volume - as.vector(predictions_harmonic.qr)
# Harmonic model using Quantile regression
MdRAE(harmonic.qr.test_resids, resids.persistence)
GMRAE(harmonic.qr.test_resids, resids.persistence)
>>
1
0.8486
```

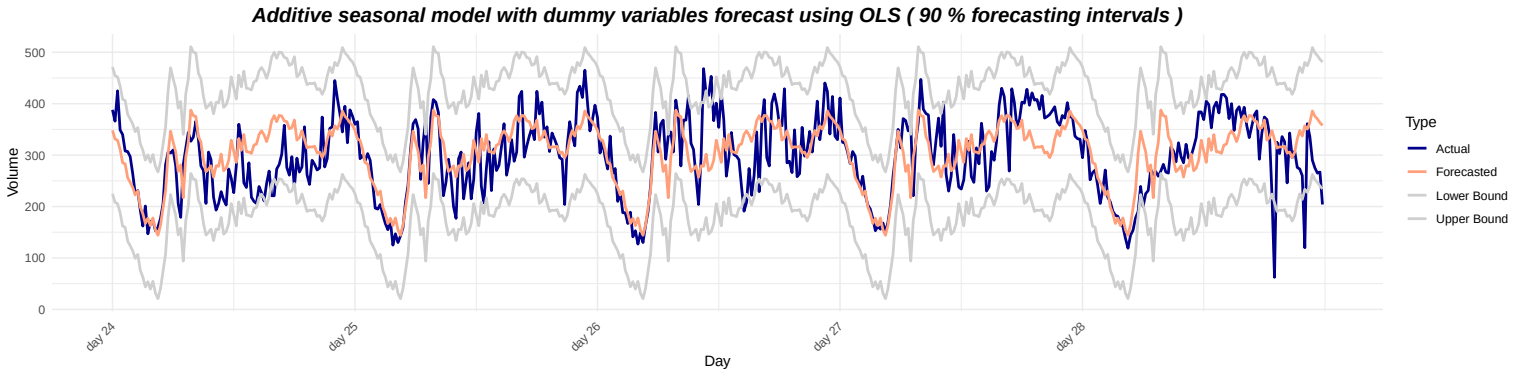
Using Backward-stepwise:

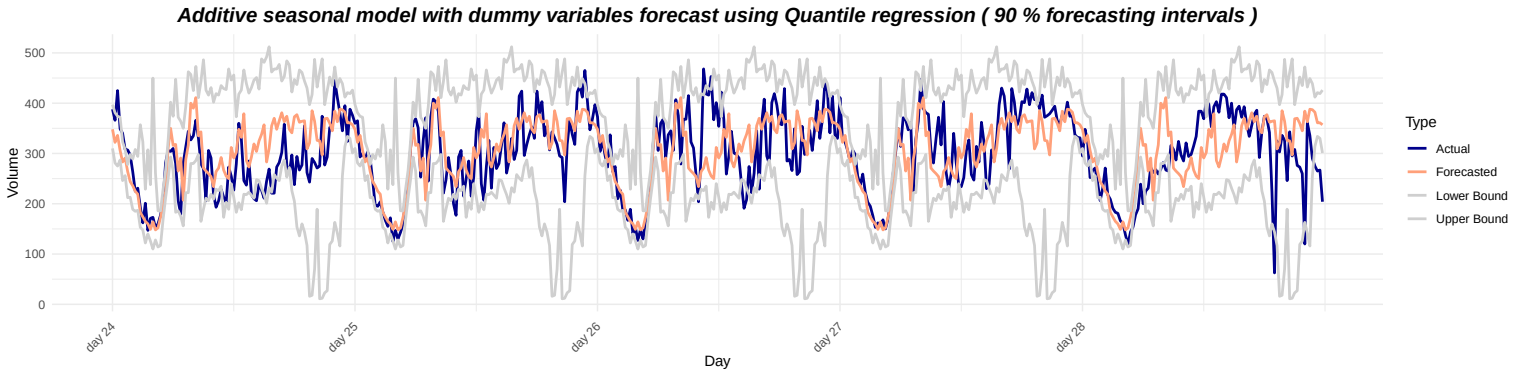
```
harmonic.BAIC.test_resids <- forecast.ts$L104_volume - as.vector(harmonic_BAIC.df$Estimate)
# Harmonic ols model using BAIC
MdRAE(harmonic.BAIC.test_resids, resids.persistence)
GMRAE(harmonic.BAIC.test_resids, resids.persistence)
>>
1.0068
0.9952

harmonic_rq.BAIC.test_resids <- forecast.ts$L104_volume - as.vector(harmonic_rq_BAIC.df$Estimate)
# Harmonic qr model using BAIC
MdRAE(harmonic_rq.BAIC.test_resids, resids.persistence)
GMRAE(harmonic_rq.BAIC.test_resids, resids.persistence)
>>
0.9975
0.9326
```

Furthermore, we will examine models that use additive indicators (dummy variables). Ultimately,

we will investigate if the log odds of volumes over occupancies can be predicted by the sum of occupancies and additive indicators, In other words, if it exhibits a decent MdRAE value.

```
### Additive Seasonal Model with Dummy Variables ####
train.data$time_factor <- as.factor(train.data$time)
test.data$time_factor <- as.factor(test.data$time)

additive.ols <- dynlm(L104_volume ~ time_factor, data = train.data)

additive.qr <- dynrq(L104_volume ~ time_factor, data = train.data, tau=0.5)
additive.qr.lower <- dynrq(L104_volume ~ time_factor, data = train.data, tau=lower.q)
additive.qr.upper <- dynrq(L104_volume ~ time_factor, data = train.data, tau=upper.q)

# Forecast on the testing days
predictions_additive.ols <- predict(additive.ols, newdata = test.data,
                                    interval = "prediction", level = 0.9)

predictions_additive.rq <- predict(additive.qr, newdata = test.data)
predictions_additive.qr.lower <- predict(additive.qr.lower, newdata = test.data)
predictions_additive.qr.upper <- predict(additive.qr.upper, newdata = test.data)

additive_ols.df <- data.frame("Lower Bound"=as.vector(predictions_additive.ols[,2]),
                              Estimate=as.vector(predictions_additive.ols[,1]),
                              "Upper Bound"=as.vector(predictions_additive.ols[,3]),
                              check.names = FALSE)
additive_qr.df <- data.frame("Lower Bound"=as.vector(predictions_additive.qr.lower),
                              Estimate=as.vector(predictions_additive.rq),
                              "Upper Bound"=as.vector(predictions_additive.qr.upper),
                              check.names = FALSE)
plot.tss(forecast.values=additive_ols.df,
         data.actual=forecast.ts, interval=TRUE, level=0.9,
         main="Additive seasonal model with dummy variables forecast using OLS")
plot.tss(forecast.values=additive_qr.df,
         data.actual=forecast.ts, interval=TRUE, level=0.9,
         main="Additive seasonal model with dummy variables forecast using Quantile regression")
>>
```





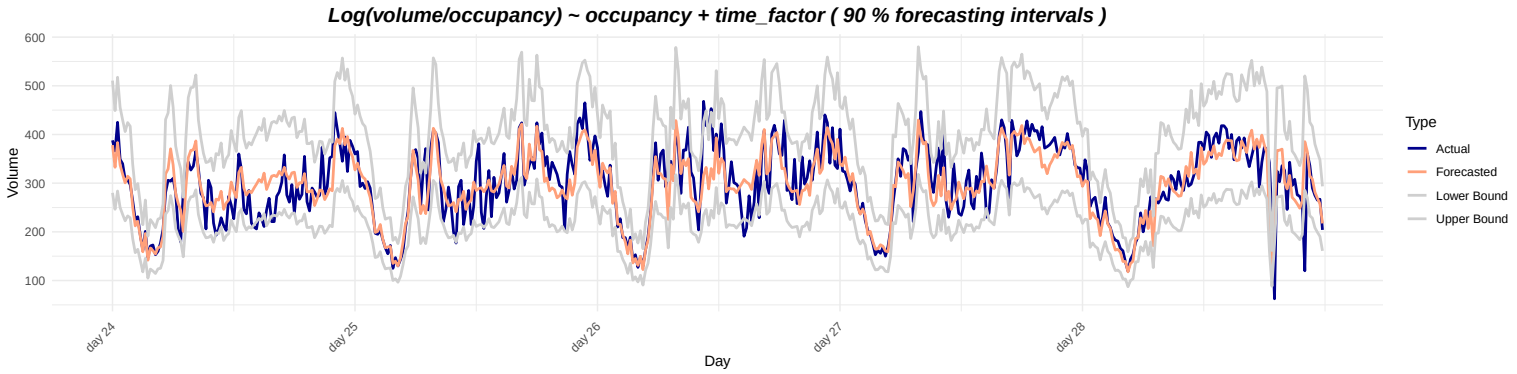
We now introduce the following model:

$$\log\left(\frac{\text{volume}}{\text{occupancy}}\right) = \beta_0 + \beta_1 \cdot \text{occupancy} + \beta_2 \cdot \text{time} + \varepsilon \qquad (*)$$

This model exhibits better results due to its ability to capture the relationship between volume and occupancy more effectively. The use of logarithmic odds provides several advantages. Firstly, taking the logarithm of the ratio stabilizes the variance of the dependent variable, making the model more robust to heteroscedasticity. Secondly, the logarithmic transformation helps in handling skewed data, making the distribution of the dependent variable more symmetric and closer to normality, which is a key assumption for many regression techniques.

Additionally, the coefficients in a log-odds model are interpretable in terms of percentage changes, making it easier to understand the impact of predictors. For instance, a coefficient represents the percentage change in the volume-to-occupancy ratio for a one-unit change in the predictor. By using the ratio of volume to occupancy, the model directly addresses the relationship between these variables, improving the overall fit and predictive accuracy. This approach helps in capturing the underlying patterns more effectively, leading to better forecasting performance. In R:

```
# model: log(V/O)
additive.logodds <- dynlm(log(L104_volume/L104_occupancy) ~ L104_occupancy + time_factor, data = train.data)
additive.logodds.preds <- forecast.ts$L104_occupancy *
                                    exp(predict(additive.logodds, newdata=test.data,
                                            interval = "prediction", level = 0.9))
additive.logodds.df <- data.frame("Lower Bound"=as.vector(additive.logodds.preds[,2]),
                                  Estimate=as.vector(additive.logodds.preds[,1]),
                                  "Upper Bound"=as.vector(additive.logodds.preds[,3]),
                                  check.names = FALSE)
plot.tss(forecast.values=additive.logodds.df,
         data.actual=forecast.ts,  interval=TRUE, level=0.9,
         main="Log(volume/occupancy) ~ occupancy + time_factor")
>>
```




```
additive.logodds.test_resids <- forecast.ts$L104_volume - as.vector(additive.logodds.df$Estimate)
MdRAE(additive.logodds.test_resids, resids.persistence)
GMRAE(additive.logodds.test_resids, resids.persistence)
>>
0.6929
0.6584

additive.ols.test_resids <- forecast.ts$L104_volume - as.vector(predictions_additive.ols[,1])
MdRAE(additive.ols.test_resids, resids.persistence)
GMRAE(additive.ols.test_resids, resids.persistence)
>>
1.0005
1.0258

additive.qr.test_resids <- forecast.ts$L104_volume - as.vector(predictions_additive.rq)
MdRAE(additive.qr.test_resids, resids.persistence)
# Omit the zeros (replace them with a small number)
additive.qr.test_resids[additive.qr.test_resids == 0] = 1e-9
GMRAE(additive.qr.test_resids, resids.persistence)
>>
1.0707
0.8674
```

Do we need ARMA or GARCH correction? To answer this, we will focus on the best-performing models in terms of MdRAE and GMRAE. For MdRAE, the additive seasonal model with dummy variables using least squares is considered. For GMRAE, the harmonic model using quantile regression is evaluated. We will not further investigate the model ($*$), as it does not offer satisfactory results in subsequent combination schemes.

```
# additive.ols
additive.ols.resids <- as.numeric(residuals(additive.ols)) # train resids
checkresiduals(additive.ols.resids)
>>
    Ljung-Box test
data:  Residuals
Q* = 2228.3, df = 10, p-value < 2.2e-16
Model df: 0.   Total lags used: 10
```

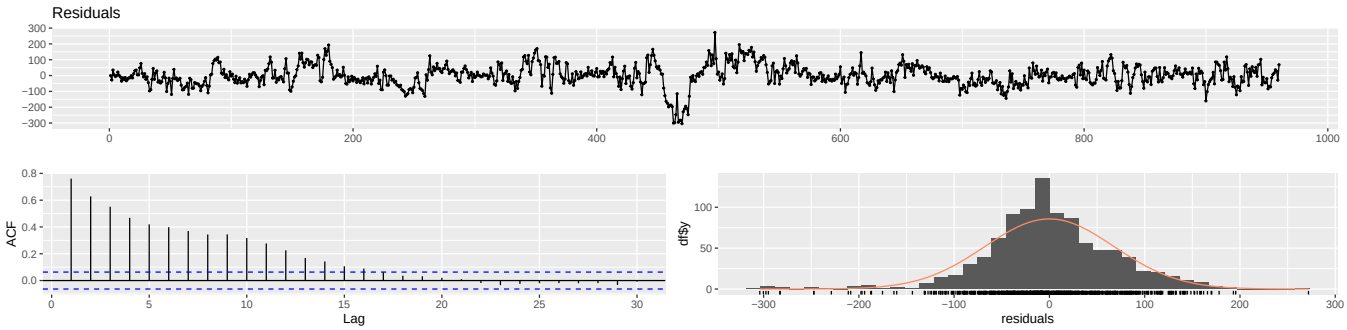


Figure 7: *Additive (dummy variables) OLS residuals diagnostics.*

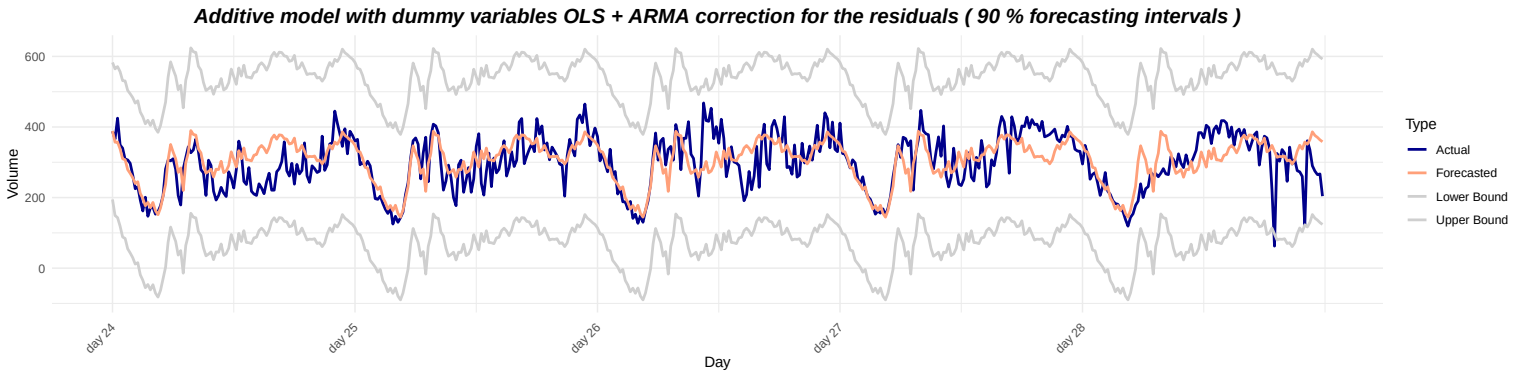
We tune the ARMA parameters:

```
best.order <- c(0, 0, 0)
best.aic <- Inf
max.p <- 5
max.q <- 5
for (i in 0:max.p) for (j in 0:max.q) {
    fit.aic <- AIC(arima(additive.ols.resids, order = c(i, 0, j)))
    if (fit.aic < best.aic) {
      best.order <- c(i, 0, j)
      best.arma <- arima(additive.ols.resids, order = best.order)
      best.aic <- fit.aic
    }
}

best.order
best.arma
>>
5   0   2
>>
arima(x = additive.ols.resids, order = best.order)
Coefficients:
         ar1      ar2     ar3      ar4     ar5      ma1     ma2  intercept
      2.1424  -1.6577  0.4993  -0.1287  0.1065  -1.4924  0.7314     0.3915
s.e.  0.1741   0.2462  0.1165   0.0816  0.0394   0.1735  0.1285     8.6180
sigma^2 estimated as 1857:  log likelihood = -4975.47,  aic = 9968.93
```

We are ready now to forecast the residuals using the ARMA$(5, 2)$.

```
best.metcalfe.arima.res.forecast <- forecast(best.arma, level=0.9, h=h)
arma.test.pred <- mpr.arma*best.metcalfe.arima.res.forecast$mean + as.vector(predictions_additive.ols[,1])

arma.test.pred.lower <- as.vector(predictions_additive.ols[,2]) +
                        mpr.arma*best.metcalfe.arima.res.forecast$lower[,1]
arma.test.pred.upper <- as.vector(predictions_additive.ols[,3]) +
                        mpr.arma*best.metcalfe.arima.res.forecast$upper[,1]
```

```
arma.add.df <- data.frame("Lower Bound" = arma.test.pred.lower, Estimate=arma.test.pred,
                          "Upper Bound" = arma.test.pred.upper, check.names = FALSE)

plot.tss(forecast.values=arma.add.df,
         data.actual=forecast.ts, interval=TRUE, level=0.9,
         main="Additive model with dummy variables OLS + ARMA correction for the residuals")
>>
```




```
# With ARMA
MdRAE(forecast.ts$L104_volume - as.vector(arma.test.pred), resids.persistence)
GMRAE(forecast.ts$L104_volume - as.vector(arma.test.pred), resids.persistence)
>>
1.0312
1.0207

# Whitout ARMA
MdRAE(additive.ols.test_resids, resids.persistence)
GMRAE(additive.ols.test_resids, resids.persistence)
>>
1.0005
1.0258

library(tseries)
garch.model <- garch(as.ts(additive.ols.resids),
                     order=c(1, 1), grad = "numerical", trace = FALSE)
garch.model
print(confint(garch.model))
>>
Coefficient(s):
       a0         a1         b1
4.131e+03  1.971e-01  2.108e-10
           2.5 %       97.5 %
a0 2852.75785496 5408.4895096
a1    0.09842748    0.2958114
b1   -0.22808854    0.2280885
```

```
arma.additive_ols.fitted <- as.vector(fitted(additive.ols)) +
                            as.data.frame(forecast(best.arma, level=0.9, h=dim(train.ts)[1]))$Point

arma.additive_ols.resids <- train.ts$L104_volume - arma.additive_ols.fitted
arma.additive_ols.test.resids <- forecast.ts$L104_volume - arma.add.df$Estimate
garch.upper.additive_ols <- arma.add.df$Upper + z_value * abs(additive_ols.garch.pred.res)
garch.lower.additive_ols <- arma.add.df$Lower - z_value * abs(additive_ols.garch.pred.res)

garch.df.additive_ols <- data.frame("Lower Bound" = garch.lower.additive_ols, Estimate=arma.add.df$Estimate,
                        "Upper Bound" = garch.upper.additive_ols,  check.names = FALSE)
```

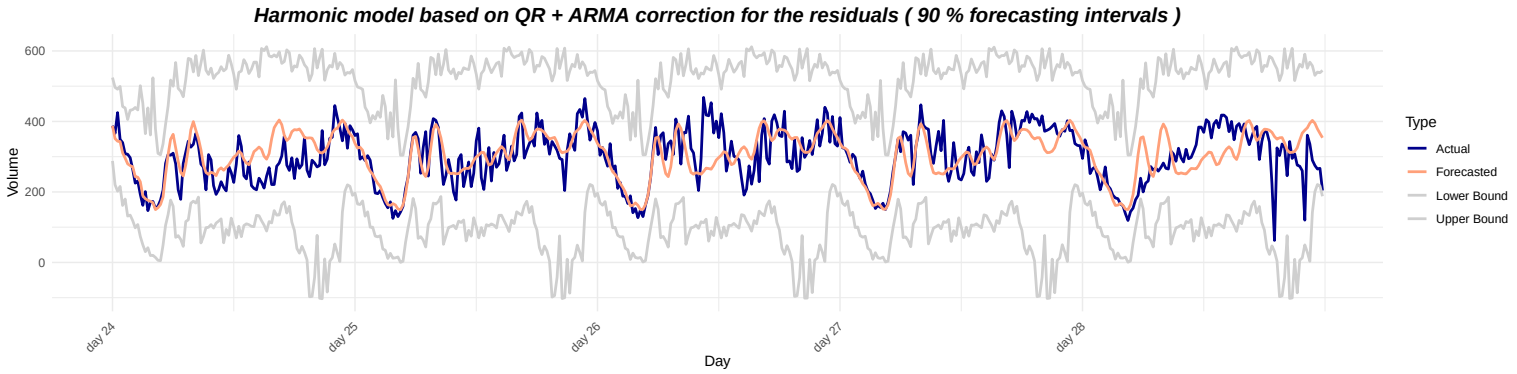
We will investigate the BAIC harmonic using QR:

```
harmonic.qr.resids <- as.vector(residuals(harmonic_rq.BAIC))
harmonic.qr.test.resids <- forecast.ts$L104_volume - as.vector(harmonic_rq_BAIC.df$Estimate)
# ARMA
best.order <- c(0, 0, 0)
best.aic <- Inf
max.p <- 10
max.q <- 10
for (i in 0:max.p) for (j in 0:max.q) {
    fit.aic <- AIC(arima(harmonic.qr.resids, order = c(i, 0, j)))
    if (fit.aic < best.aic) {
      best.order <- c(i, 0, j)
      best.arma <- arima(harmonic.qr.resids, order = best.order)
      best.aic <- fit.aic
    }
}

best.order
best.arma
>>
8  0  6
>>
arima(x = harmonic.qr.resids, order = best.order)
Coefficients:
         ar1      ar2     ar3      ar4      ar5      ar6     ar7     ar8
      0.5498  -0.0266  1.2242  -0.6517  -0.0187  -0.8636  0.5192  0.1077
s.e.  0.0751   0.0542  0.0544   0.0916   0.0672   0.0545  0.0493  0.0372
         ma1     ma2      ma3      ma4      ma5     ma6  intercept
      0.0747  0.1677  -1.0907  -0.0139  -0.0908  0.8640     2.9811
s.e.  0.0682  0.0742   0.0826   0.0509   0.0493  0.0563     8.2945
sigma^2 estimated as 2058:  log likelihood = -5027.16,  aic = 10086.32
```

We may now predict

```
best.metcalfe.arima.res.forecast <- forecast(best.arma, level=0.9, h=h)
arma.test.pred <- best.metcalfe.arima.res.forecast$mean + as.vector(harmonic_rq_BAIC.df$Estimate )
arma.test.pred.lower <- as.vector(predictions_harmonic.qr.lower) +
                        best.metcalfe.arima.res.forecast$lower[,1]
arma.test.pred.upper <- as.vector(predictions_harmonic.qr.upper) +
```

```
                                best.metcalfe.arima.res.forecast$upper[,1]

arma.harm.df <- data.frame("Lower Bound" = arma.test.pred.lower, Estimate=arma.test.pred,
                           "Upper Bound" = arma.test.pred.upper,  check.names = FALSE)
plot.tss(forecast.values=arma.harm.df,
         data.actual=forecast.ts, interval=TRUE, level=0.9,
         main="Harmonic model based on QR + ARMA correction for the residuals")
>>
```



Let's find out if this reduces the MdRAE:

```
MdRAE(forecast.ts$L104_volume - as.vector(arma.test.pred), resids.persistence)
GMRAE(forecast.ts$L104_volume - as.vector(arma.test.pred), resids.persistence)
>>
0.9831
0.9483

# Harmonic-BAIC WITHOUT ARMA
MdRAE(forecast.ts$L104_volume - as.vector(harmonic_rq_BAIC.df$Estimate), resids.persistence)
GMRAE(forecast.ts$L104_volume - as.vector(harmonic_rq_BAIC.df$Estimate), resids.persistence)
>>
0.9975
0.9326
```

We repeat the same procedure as before to examine if a GARCH correction for the residuals addresses the volatilities. After that, we obtain the following MSIS values:

```
### Harmonic QR & additive dummy var. OLS

qr_harmonic.msis <- smis(data = ts(train.ts$L104_volume),
                actual = ts(forecast.ts$L104_volume),
                lower = as.vector(predictions_harmonic.qr.lower),
                upper = as.vector(predictions_harmonic.qr.upper),
                m = freq, level = 0.9)
qr_harmonic.msis
>>
```

```
3.72680

ols_additive.msis <- smis(data = ts(train.ts$L104_volume),
                actual = ts(forecast.ts$L104_volume),
                lower = as.vector(predictions_additive.ols[,2]),
                upper = as.vector(predictions_additive.ols[,3]),
                m = freq, level = 0.9)
ols_additive.msis
>>
3.73383

# ARMA residuals
qr_harmonic_ARMA.msis <- smis(data = ts(train.ts$L104_volume),
                actual = ts(forecast.ts$L104_volume),
                lower = as.vector(arma.harm.df$Lower),
                upper = as.vector(arma.harm.df$Upper),
                m = freq, level = 0.9)

ols_additive_ARMA.msis <- smis(data = ts(train.ts$L104_volume),
                actual = ts(forecast.ts$L104_volume),
                lower = as.vector(arma.add.df$Lower),
                upper = as.vector(arma.add.df$Upper),
                m = freq, level = 0.9)

qr_harmonic_ARMA.msis
ols_additive_ARMA.msis
>>
5.95931
6.42194

### + ARMA residuals + Koyck volatilities

qr_harmonic_garch.msis <- smis(data = ts(train.ts$L104_volume)[-1],
                actual = ts(forecast.ts$L104_volume)[-1],
                lower = as.vector(garch.df.harmonic_qr$Lower)[-1],
                upper = as.vector(garch.df.harmonic_qr$Upper)[-1],
                m = freq, level = 0.9)

ols_additive_garch.msis <- smis(data = ts(train.ts$L104_volume)[-1],
                actual = ts(forecast.ts$L104_volume)[-1],
                lower = as.vector(garch.df.additive_ols$Lower)[-1],
                upper = as.vector(garch.df.additive_ols$Upper)[-1],
                m = freq, level = 0.9)

qr_harmonic_garch.msis
ols_additive_garch.msis
>>
5.9879
6.4423
```

Part 4: Forecast combination scheme

We will employ a technique frequently adopted by experts in the field: using a combination scheme of various models to capture complex patterns that individual models might miss. For this purpose, we will use the `ForecastComb` package from `R`. Here's an overview of the package:

- ◇ **`comb_BG`**: The Bayesian combination method applies Bayesian principles to combine forecasts, accounting for the uncertainty in model parameters and providing a probabilistic framework for enhanced accuracy and robustness.
- ◇ **`comb_InvW`**: The inverse-variance weighted combination method assigns weights to forecasts inversely proportional to their variance, giving higher weights to forecasts with lower variance and higher reliability.
- ◇ **`comb_MED`**: The median combination method combines forecasts by taking the median value, offering robustness against outliers and ensuring that extreme values do not disproportionately influence the combined forecast.
- ◇ **`comb_NG`**: The non-Gaussian combination method uses techniques that are robust to non-Gaussian distributions, making it useful when forecast errors do not follow a normal distribution.
- ◇ **`comb_SA`**: The simple average combination method combines forecasts by equally averaging them, effectively reducing individual forecast errors through diversification.
- ◇ **`comb_TA`**: The trimmed average combination method combines forecasts by averaging after trimming a certain percentage of the highest and lowest forecasts, thus reducing the influence of extreme values and outliers.
- ◇ **`comb_WA`**: The weighted average combination method assigns different weights to each forecast based on historical performance or other criteria, combining them to enhance overall accuracy.

Regression-Based Forecast Combination Functions:

- ◇ **`comb_CLS`**: The combined least squares method uses least squares regression to combine forecasts, minimizing the sum of squared residuals for an optimal linear combination.
- ◇ **`comb_CSR`**: The combined shrinkage regression method applies shrinkage techniques, such as ridge or lasso regression, to combine forecasts, addressing multicollinearity and overfitting by penalizing large coefficients.
- ◇ **`comb_LAD`**: The least absolute deviation regression method combines forecasts by minimizing the sum of absolute deviations, providing robustness to outliers. This method worked the best in our analysis, demonstrating superior resilience and accuracy.
- ◇ **`comb_OLS`**: The ordinary least squares regression method combines forecasts by minimizing the sum of squared residuals, assigning weights based on historical accuracy for an optimal linear combination.
- ◇ **`comb_EIG1, comb_EIG2, comb_EIG3, comb_EIG4`**: These eigenvector-based combination methods use principal component analysis (PCA) to combine forecasts based on the most significant eigenvectors, reducing dimensionality and improving efficiency by focusing on key components. Each method (EIG1 to EIG4) may apply different criteria or components for the combination.

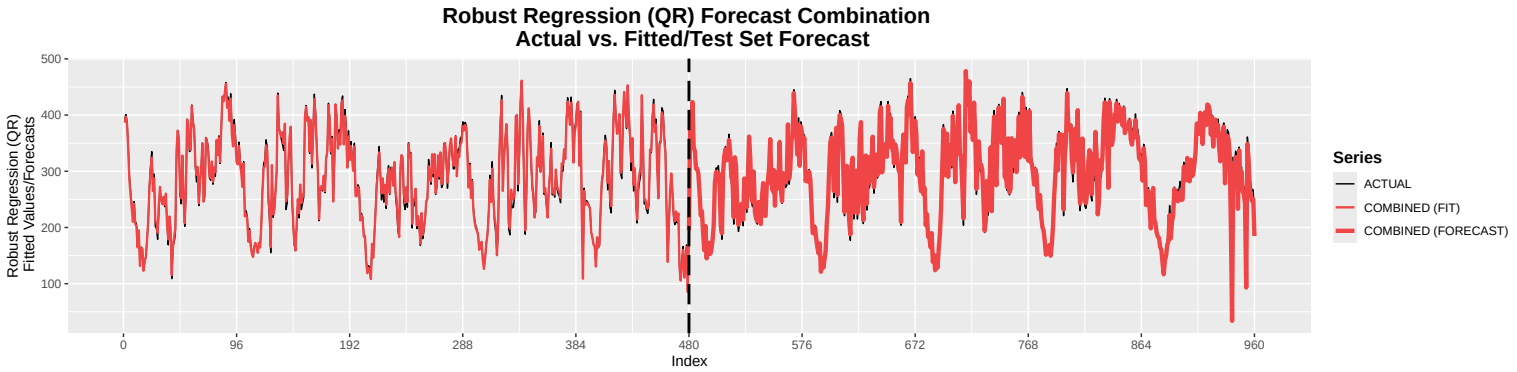
Function	Description
Data Preparation Functions	
foreccomb	Transform raw input data for forecast combination
cs_dispersion	Compute cross-sectional dispersion
Forecast Combination Functions	
Simple Forecast Combination Functions	
comb_BG	Bates/Granger (1969) forecast combination
comb_InvW	Inverse Rank forecast combination
comb_MED	Median Forecast Combination
comb_NG	Newbold/Granger (1974) forecast combination
comb_SA	Simple Average forecast combination
comb_TA	Trimmed Mean forecast combination
comb_WA	Winsorized Mean forecast combination
Regression-Based Forecast Combination Functions	
comb_CLS	Constrained Least Squares (CLS) forecast combination
comb_CSR	Complete Subset Regression forecast combination
comb_LAD	Least Absolute Deviation (LAD) forecast combination
comb_OLS	Ordinary Least Squares (OLS) forecast combination
Eigenvector-Based Forecast Combination Functions	
comb_EIG1	Standard Eigenvector forecast combination
comb_EIG2	Bias-Corrected Eigenvector forecast combination
comb_EIG3	Trimmed Eigenvector forecast combination
comb_EIG4	Trimmed Bias-Corrected Eigenvector forecast combination
Other Forecast Combination Functions	
auto_combine	Automated grid-search forecast combination
rolling_combine	Rolling forecast combination (time-varying)

In R, starting with ***ARDL+BAIC-Harmonic QR***:

```
library(ForecastComb)

# extract the training combination period
mid_week.days <- unique(data$DAY)[(!(unique(data$DAY) %in%
                    c(unique(train.ts$DAY), unique(forecast.ts$DAY))))]
comb_train.ts <- subset(data, DAY %in% mid_week.days)
comb_forecast.ts <- forecast.ts

comb_train.ts$L104_occupancy.9 <- as.vector(scale(comb_train.ts$L104_occupancy^9)) # scale the new occ.
#forecast.ts$L104_occupancy.9 <- as.vector(scale(forecast.ts$L104_occupancy^9)) # wrong!
train.mean.occ9 <- mean(comb_train.ts$L104_occupancy^9)
train.sd.occ9 <- sd(comb_train.ts$L104_occupancy^9)
comb_forecast.ts$L104_occupancy.9 <- (comb_forecast.ts$L104_occupancy^9 - train.mean.occ9)/train.sd.occ9

comb_train.ts$time <- rep(1:freq, length.out = nrow(comb_train.ts))
comb_forecast.ts$time <- rep(1:freq, length.out = nrow(comb_forecast.ts))
```

```r
s <- floor(freq / 2) - 1 #  = 47

# Add sinusoidal terms to the data
for (j in 1:s) {
    comb_train.ts[[paste0("sin_", j)]] <- sin(2 * pi * j * comb_train.ts$time / freq)
    comb_train.ts[[paste0("cos_", j)]] <- cos(2 * pi * j * comb_train.ts$time / freq)
    comb_forecast.ts[[paste0("sin_", j)]] <- sin(2 * pi * j * comb_forecast.ts$time / freq)
    comb_forecast.ts[[paste0("cos_", j)]] <- cos(2 * pi * j * comb_forecast.ts$time / freq)
}

# generate predictions for the ARDL model
pred_ardl_train <- predict(ardl.model$best.ardl$model, newdata = comb_train.ts)
pred_ardl_test <- predict(ardl.model$best.ardl$model, newdata = comb_forecast.ts)

# generate predictions for the harmonic QR model
pred_harmonic_qr_train <- predict(harmonic_rq.BAIC, newdata = comb_train.ts)
pred_harmonic_qr_test <- predict(harmonic_rq.BAIC, newdata = comb_forecast.ts)

# combine forecasts into a matrix
train_combined_forecasts <- cbind(pred_ardl_train, pred_harmonic_qr_train)
test_combined_forecasts <- cbind(pred_ardl_test, pred_harmonic_qr_test)

# actual values for the combination training period
actual_train <- comb_train.ts$L104_volume
actual_test <- comb_forecast.ts$L104_volume

# structure the data using the foreccomb function
data_comb <- foreccomb(actual_train, train_combined_forecasts,
                       newobs = actual_test,
                       newpreds = test_combined_forecasts)
# Evaluate all the forecast combination methods and return the best
best_combination.2 <- auto_combine(data_comb, criterion = "MAPE")

# display the best combination method and its details
summary(best_combination.2)
>>
Summary of Forecast Combination
-------------------------------
Method:  Robust Regression (QR)
Individual Forecasts & Combination Weights:
                       Combination Weight
pred_ardl_train                 1.2081568
pred_harmonic_qr_train         -0.1098581
Intercept (Bias-Correction):  -28.32745
Accuracy of Combined Forecast:

                     ME     RMSE      MAE        MPE     MAPE
Training Set -0.6684554 7.706319 6.360118 -0.3177154 2.399913
Test set      1.1320530 8.877062 7.322645  0.3565986 2.627385
```

```
plot(best_combination.2)
>>
```



That did a great job, with just two models! To ensure the above procedure is indeed correct, we will manually forecast using the weights of each model and report the corresponding naive prediction intervals.

```
# predictions from the ARDL model
pred_ardl_new <- predict(ardl.model$best.ardl$model, newdata = comb_forecast.ts)

# predictions from the QR harmonic model
pred_harmonic_qr_new <- predict(model_harmonic.qr, newdata = comb_forecast.ts)

# combine the new forecasts into a matrix
new_combined_forecasts <- cbind(pred_ardl_new, pred_harmonic_qr_new)

forecastcomb.ardlQRharm <- as.vector(summary(best_combination.2)$weight[1] * new_combined_forecasts[,1] +
                           + summary(best_combination.2)$weight[2] * new_combined_forecasts[,2]) +
                           + summary(best_combination.2)$Intercept
MdRAE(forecast.ts$L104_volume - forecastcomb.ardlQRharm, resids.persistence) #same as above!
>>
0.179362761644922 # same as the above!
```

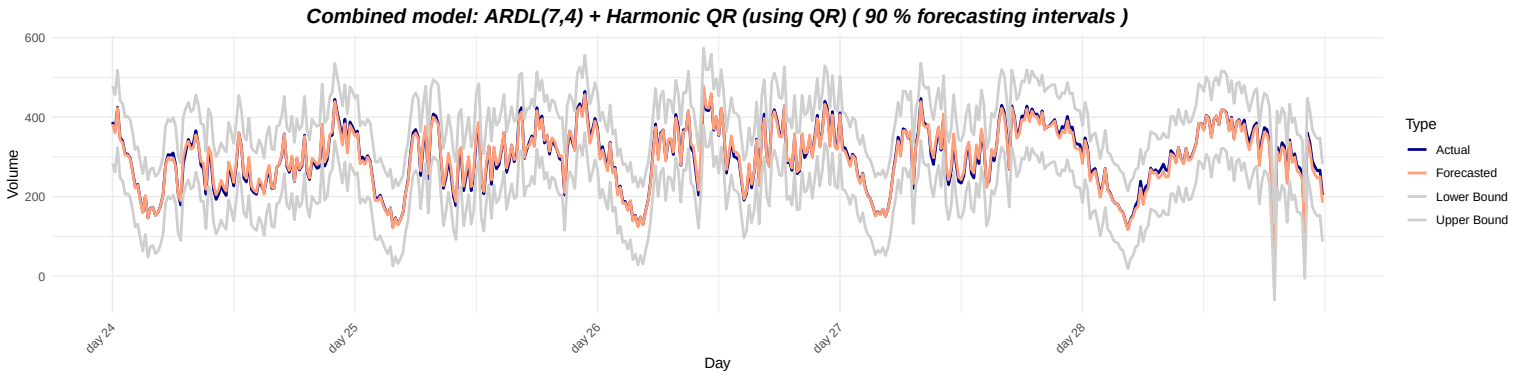
For the naive **PI**:

```
# Naive forecasting intervals for ARDL model
pred_ardl_intervals <- predict(ardl.model$best.ardl$model, newdata = comb_forecast.ts,
                               interval = "prediction", level = 0.9)
ardl_lower <- pred_ardl_intervals[, "lwr"]
ardl_upper <- pred_ardl_intervals[, "upr"]
# Naive forecasting intervals for QR harmonic model
pred_harmonic_qr_lower <- predict(model_harmonic.qr, newdata = comb_forecast.ts, tau = 0.05)
pred_harmonic_qr_upper <- predict(model_harmonic.qr, newdata = comb_forecast.ts, tau = 0.95)
# Extract weights and intercept from the best combination model
weights <- summary(best_combination.2)$weight
intercept <- summary(best_combination.2)$Intercept
```

```
# Combine the naive intervals including intercept
combined_lower <- intercept + weights[1] * ardl_lower + weights[2] * pred_harmonic_qr_lower
combined_upper <- intercept + weights[1] * ardl_upper + weights[2] * pred_harmonic_qr_upper
combined_forecast_2.df <- data.frame(
  "Lower Bound" = combined_lower,
  Estimate = combined_forecasts,
  "Upper Bound" = combined_upper,
  check.names = FALSE)
```

We now plot the combined fit with their prediction intervals.

```
plot.tss(forecast.values=combined_forecast_2.df,
         data.actual=forecast.ts, interval=TRUE, level=0.9,
         main="Combined model: ARDL(7,4) + Harmonic QR (using QR)")
>>
```

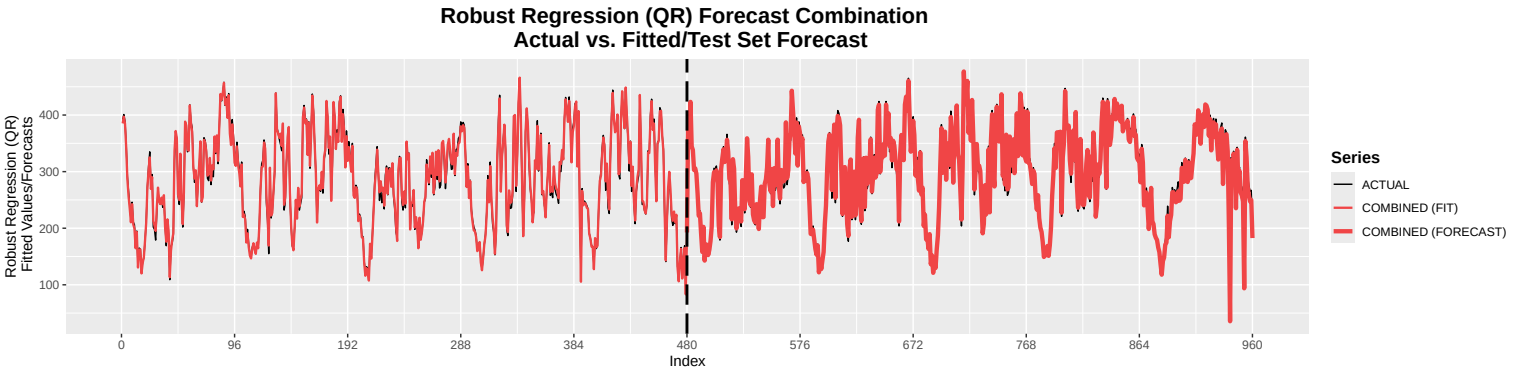



```
smis(data = ts(train.ts$L104_volume),
            actual = ts(forecast.ts$L104_volume),
            lower = as.vector(combined_forecast_2.df$Lower),
            upper = as.vector(combined_forecast_2.df$Upper),
            m = freq, level = 0.9)
>>
2.6608061408051
```

Now, we will repeat the same procedure, this time combining three models: ARDL, BAIC-harmonic QR, and additive dynlm.

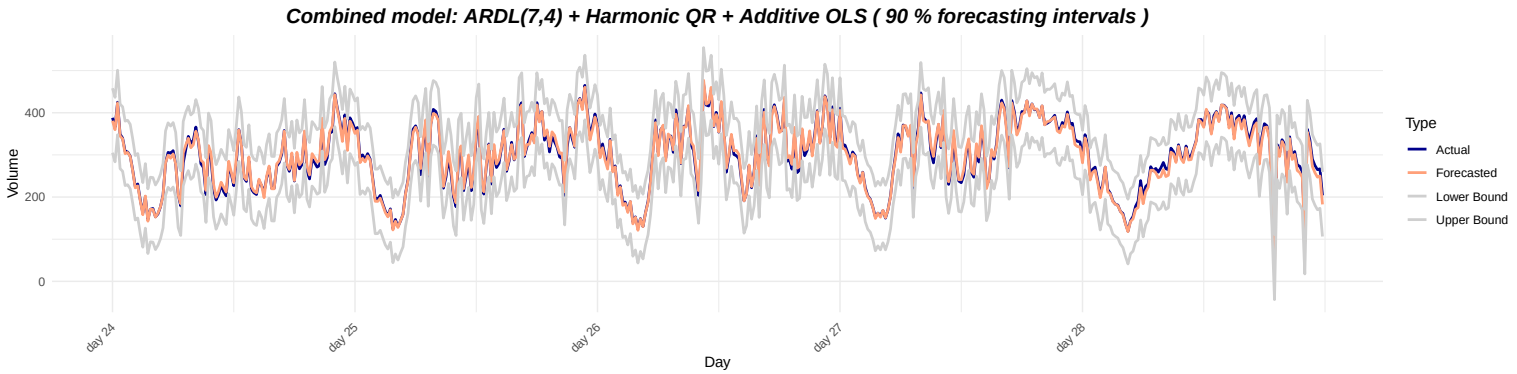
```
comb_train.ts$time_factor <- as.factor(comb_train.ts$time)
comb_forecast.ts$time_factor <- as.factor(comb_forecast.ts$time)

# generate predictions from the ARDL model
pred_ardl_train <- predict(ardl.model$best.ardl$model, newdata = comb_train.ts)
pred_ardl_test <- predict(ardl.model$best.ardl$model, newdata = comb_forecast.ts)

# generate predictions from the QR harmonic model
pred_harmonic_qr_train <- predict(harmonic_rq.BAIC, newdata = comb_train.ts)
```

```r
pred_harmonic_qr_test <- predict(harmonic_rq.BAIC, newdata = comb_forecast.ts)

# generate predictions from the additive.ols model
pred_additive_ols_train <- predict(additive.ols, newdata = comb_train.ts)
pred_additive_ols_test <- predict(additive.ols, newdata = comb_forecast.ts)

# combine forecasts into a matrix
train_combined_forecasts <- cbind(pred_ardl_train, pred_harmonic_qr_train, pred_additive_ols_train)
test_combined_forecasts <- cbind(pred_ardl_test, pred_harmonic_qr_test, pred_additive_ols_test)

# prepare actual values for the combination training period
actual_train <- comb_train.ts$L104_volume
actual_test <- comb_forecast.ts$L104_volume

# Structure the data using the foreccomb function
data_comb <- foreccomb(actual_train, train_combined_forecasts,
                       newobs=actual_test, newpreds=test_combined_forecasts)

# Evaluate all the forecast combination methods and return the best
best_combination.3 <- auto_combine(data_comb, criterion = "MAPE")

# Structure the data using the foreccomb function
data_comb <- foreccomb(actual_train, train_combined_forecasts,
                       newobs=actual_test, newpreds=test_combined_forecasts)

# Evaluate all the forecast combination methods and return the best
best_combination.3 <- auto_combine(data_comb, criterion = "MAPE")

summary(best_combination.3)
>>
Method:  Robust Regression (QR)

Individual Forecasts & Combination Weights:

                        Combination Weight
pred_ardl_train                  1.2190797
pred_harmonic_qr_train           0.0542941
pred_additive_ols_train         -0.1757467

Intercept (Bias-Correction):  -27.51733

Accuracy of Combined Forecast:

                     ME     RMSE      MAE        MPE     MAPE
Training Set -0.6803559 7.153966 5.746895 -0.2115713 2.203264
Test set      1.0060260 8.132838 6.685752  0.4016226 2.419982

plot(best_combination.3)
>>
```




```r
# residuals for combined forecasts
combined_forecasts <- best_combination.3$Forecasts_Test
combined_residuals <- actual_test - combined_forecasts

#  MdRAE for combined forecasts
mdrae_combined.3 <- MdRAE(combined_residuals, resids.persistence)
gmrae_combined.3 <- GMRAE(combined_residuals, resids.persistence)
mdrae_combined.3
gmrae_combined.3
>>
0.185188944573677
0.172857446430445

# Naive forecasting intervals for ARDL model
pred_ardl_intervals <- predict(ardl.model$best.ardl$model, newdata = comb_forecast.ts,
                               interval = "prediction", level = 0.9)
ardl_lower <- pred_ardl_intervals[, "lwr"]
ardl_upper <- pred_ardl_intervals[, "upr"]

# Naive forecasting intervals for QR harmonic model
pred_harmonic_qr_lower <- predict(model_harmonic.qr, newdata = comb_forecast.ts, tau = 0.05)
pred_harmonic_qr_upper <- predict(model_harmonic.qr, newdata = comb_forecast.ts, tau = 0.95)

# Naive forecasting intervals for additive OLS model
pred_additive_ols_intervals <- predict(additive.ols, newdata = comb_forecast.ts,
                                       interval = "prediction", level = 0.9)
additive_ols_lower <- pred_additive_ols_intervals[, "lwr"]
additive_ols_upper <- pred_additive_ols_intervals[, "upr"]

# Extract weights and intercept from the best combination model
weights <- summary(best_combination.3)$weight
intercept <- summary(best_combination.3)$Intercept

# Combine the naive intervals including intercept
combined_lower <- intercept + weights[1] * ardl_lower +
                  weights[2] * pred_harmonic_qr_lower +
```

```
                  weights[3] * additive_ols_lower

combined_upper <- intercept + weights[1] * ardl_upper +
                  weights[2] * pred_harmonic_qr_upper +
                  weights[3] * additive_ols_upper

# Create a data frame with the combined forecast and prediction intervals
combined_forecast_3.df <- data.frame(
  "Lower Bound" = combined_lower,
  Estimate = combined_forecasts,
  "Upper Bound" = combined_upper,
  check.names = FALSE)

plot.tss(forecast.values=combined_forecast_3.df,
         data.actual=forecast.ts, interval=TRUE, level=0.9,
         main="Combined model: ARDL(7,4) + Harmonic QR + Additive OLS")
>>
```




```
smis(data = ts(train.ts$L104_volume),
            actual = ts(forecast.ts$L104_volume),
            lower = as.vector(combined_forecast_3.df$Lower),
            upper = as.vector(combined_forecast_3.df$Upper),
            m = freq, level = 0.9)
>>
2.09234141251451
```

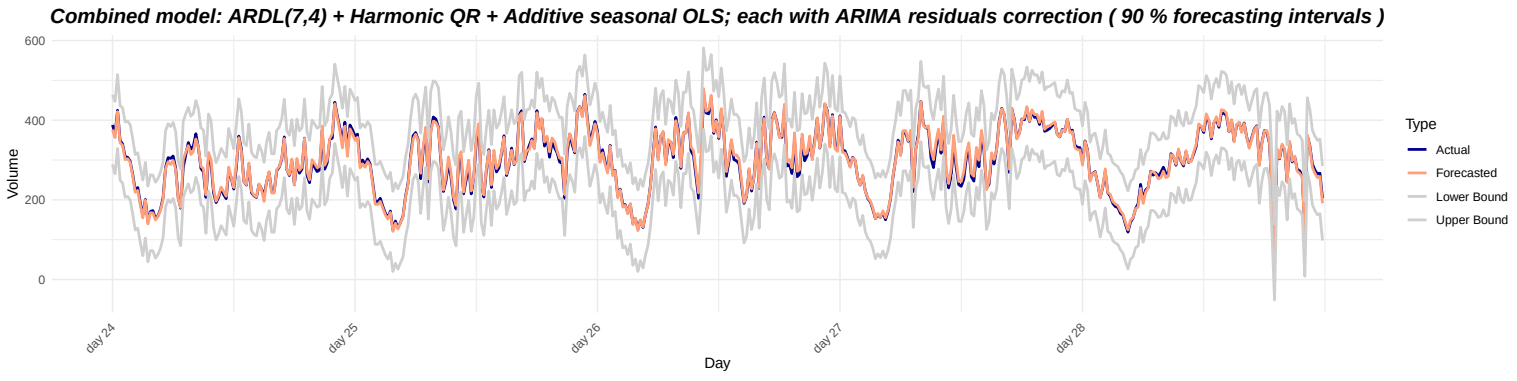
The results are very promising, but we can improve further. We will adopt ARIMA models for the residuals, with the integrated part set to 2, and tune the AR and MA degrees using the AIC criterion.

```
# predictions from the ARDL model
pred_ardl_train <- predict(ardl.model$best.ardl$model, newdata = comb_train.ts)
pred_ardl_test <- predict(ardl.model$best.ardl$model, newdata = comb_forecast.ts)

# predictions from the QR harmonic model
pred_harmonic_qr_train <- predict(harmonic_rq.BAIC, newdata = comb_train.ts)
```

```r
pred_harmonic_qr_test <- predict(harmonic_rq.BAIC, newdata = comb_forecast.ts)

# predictions from the additive.ols model
pred_additive_ols_train <- predict(additive.ols, newdata = comb_train.ts)
pred_additive_ols_test <- predict(additive.ols, newdata = comb_forecast.ts)

# residuals and convert to time series
residuals_ardl <- ts(comb_train.ts$L104_volume - pred_ardl_train)
residuals_harmonic_qr <- ts(comb_train.ts$L104_volume - pred_harmonic_qr_train)
residuals_additive_ols <- ts(comb_train.ts$L104_volume - pred_additive_ols_train)

# fit ARIMA models for the residuals of each model

# ARMA for ARDL residuals
arma_ardl <- arima(residuals_ardl, order = c(4, 2, 4))

# ARMA for QR harmonic residual
arima_harmonic_qr <- arima(residuals_harmonic_qr, order = c(3, 2, 5))

# ARMA for additive OLS residuals
arima_additive_ols <- arima(residuals_additive_ols, order = c(5, 2, 2))

# Forecast residuals with ARIMA models

h <- nrow(comb_forecast.ts) # forecasting horizon

# forecast residuals for ARDL model
forecast_arma_ardl <- forecast(arma_ardl, h = h)
arma_adjusted_ardl <- pred_ardl_test + forecast_arma_ardl$mean

# forecast residuals for QR harmonic model
forecast_arima_harmonic_qr <- forecast(arima_harmonic_qr, h = h)
arma_adjusted_harmonic_qr <- pred_harmonic_qr_test + forecast_arima_harmonic_qr$mean

# forecast residuals for additive OLS model
forecast_arima_additive_ols <- forecast(arima_additive_ols, h = h)
arma_adjusted_additive_ols <- pred_additive_ols_test + forecast_arima_additive_ols$mean

# combine the adjusted forecasts into a matrix
train_combined_forecasts <- cbind(pred_ardl_train, pred_harmonic_qr_train, pred_additive_ols_train)
test_combined_forecasts <- cbind(arma_adjusted_ardl, arma_adjusted_harmonic_qr, arma_adjusted_additive_ols)

# prepare actual values for the combination training period
actual_train <- comb_train.ts$L104_volume
actual_test <- comb_forecast.ts$L104_volume

# structure the data using the foreccomb function
data_comb <- foreccomb(actual_train, train_combined_forecasts,
                       newobs = actual_test,
                       newpreds = test_combined_forecasts)
```

```r
# Evaluate ALL the forecast combination methods and return the best
best_combination.3ARMA <- auto_combine(data_comb, criterion = "MAPE")

summary(best_combination.3ARMA)
>>
Summary of Forecast Combination
------------------------------
Method:  Robust Regression (QR)
Individual Forecasts & Combination Weights:

                        Combination Weight
pred_ardl_train                  1.2190797
pred_harmonic_qr_train           0.0542941
pred_additive_ols_train         -0.1757467
Intercept (Bias-Correction):  -27.51733

# residuals for combined forecasts
combined_forecasts <- best_combination.3ARMA$Forecasts_Test
combined_residuals <- actual_test - combined_forecasts

# Calculate MdRAE for combined forecasts
mdrae_combined <- MdRAE(combined_residuals, resids.persistence)
gmrae_combined <- GMRAE(combined_residuals, resids.persistence)

mdrae_combined
gmrae_combined
>>
0.148546875587982
0.154319830668447

# Naive forecasting intervals for ARDL model
pred_ardl_intervals <- predict(ardl.model$best.ardl$model, newdata = comb_forecast.ts,
                               interval = "prediction", level = 0.9)
ardl_lower <- pred_ardl_intervals[, "lwr"]
ardl_upper <- pred_ardl_intervals[, "upr"]

# Naive forecasting intervals for QR harmonic model
pred_harmonic_qr_lower <- predict(harmonic_rq.BAIC, newdata = comb_forecast.ts, tau = alpha/2)
pred_harmonic_qr_upper <- predict(harmonic_rq.BAIC, newdata = comb_forecast.ts, tau = 1-alpha/2)

# Naive forecasting intervals for additive OLS model
pred_additive_ols_intervals <- predict(additive.ols, newdata = comb_forecast.ts,
                                       interval = "prediction", level = 0.9)
additive_ols_lower <- pred_additive_ols_intervals[, "lwr"]
additive_ols_upper <- pred_additive_ols_intervals[, "upr"]

# adjust intervals for ARDL model
arma_lower_ardl <- ardl_lower + forecast_arma_ardl$lower[, 2]
arma_upper_ardl <- ardl_upper + forecast_arma_ardl$upper[, 2]
```

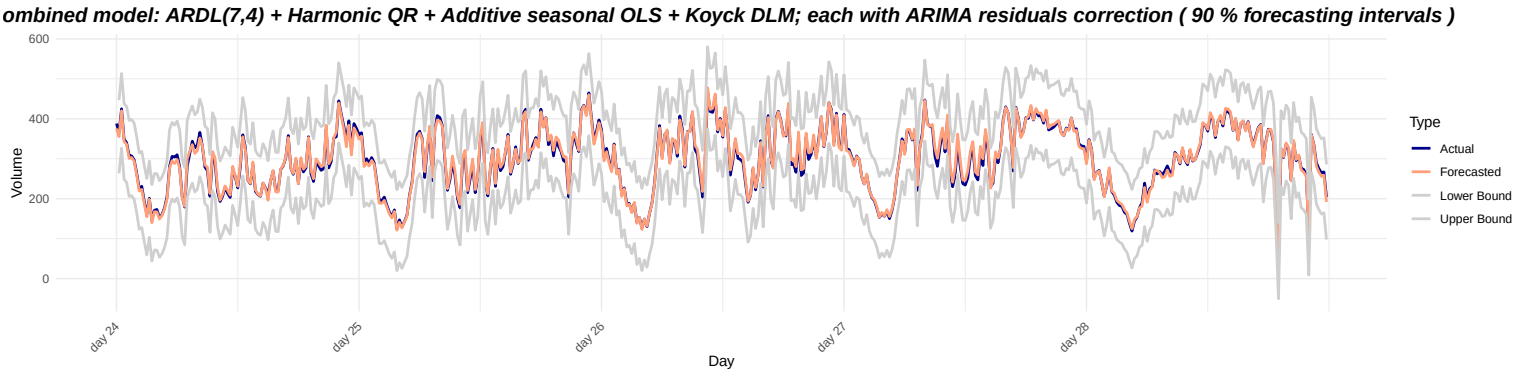
```
# adjust intervals for QR harmonic model
arma_lower_harmonic_qr <- pred_harmonic_qr_lower + forecast_arima_harmonic_qr$lower[, 2]
arma_upper_harmonic_qr <- pred_harmonic_qr_upper + forecast_arima_harmonic_qr$upper[, 2]

# adjust intervals for additive OLS model
arma_lower_additive_ols <- additive_ols_lower + forecast_arima_additive_ols$lower[, 2]
arma_upper_additive_ols <- additive_ols_upper + forecast_arima_additive_ols$upper[, 2]

# extract weights and intercept from the best combination model
weights <- summary(best_combination.3ARMA)$weight
intercept <- summary(best_combination.3ARMA)$Intercept

# combine the adjusted intervals including intercept
combined_lower <- intercept + weights[1] * arma_lower_ardl +
                  weights[2] * arma_lower_harmonic_qr +
                  weights[3] * arma_lower_additive_ols

combined_upper <- intercept + weights[1] * arma_upper_ardl +
                  weights[2] * arma_upper_harmonic_qr +
                  weights[3] * arma_upper_additive_ols

combined_forecast3ARIMA.df <- data.frame(
  "Lower Bound" = combined_lower,
  Estimate = combined_forecasts,
  "Upper Bound" = combined_upper,
  check.names = FALSE)

plot.tss(forecast.values=combined_forecast3ARIMA.df,
         data.actual=forecast.ts,
         main="Combined model: ARDL(7,4) + Harmonic QR +
               Additive seasonal OLS; each with ARIMA residuals correction",
         interval=TRUE, level=0.9)
>>
```




```
smis(data = ts(train.ts$L104_volume),
```

```
            actual = ts(forecast.ts$L104_volume),
            lower = as.vector(combined_forecast3ARIMA.df$Lower),
            upper = as.vector(combined_forecast3ARIMA.df$Upper), m = freq, level = 0.9)
>>
2.71532470629861
```

Ultimately, we present another combination: ***Koyck DLM + ARDL + BAIC-Harmonic + Additive dynlm + ARIMA (residuals)***. We repeat the same code as above. We get:




```
smis(data = ts(train.ts$L104_volume)[-1],
            actual = ts(forecast.ts$L104_volume)[-1],
            lower = as.vector(combined_forecast4ARIMA.df$Lower)[-1],
            upper = as.vector(combined_forecast4ARIMA.df$Upper)[-1], m = freq, level = 0.9)
>>
2.72839751514372
```

Conclusions

Some overall conclusions are that the combination scheme using ARIMA for the residuals is outperforming all other models. Specifically, by combining ARDL, daily-basis seasonal models, and Koyck DLM, we achieved the best MdRAE value. However, Koyck is somewhat more expensive because we do not have occupancies at time t, as mentioned above. Therefore, the best and most parsimonious choice would be the combination scheme of ARIMA residuals + ARDL + Backwards AIC Harmonic with dynamic QR and additive indicators (dummy variables), which also reported the lowest GMRAE and sMdAPE errors.

The adoption of ARIMA models significantly improved forecasting power; however, it showed that ARIMA was not very beneficial for the MSIS. On the other hand, GARCH models sometimes helped improve the MSIS, but in other cases, they did not provide any significant benefit.

Ultimately, for practical implementation, since we perform one-step ahead forecasting, we need to refit the models as soon as the next volume value is measured. This also applies to the ARIMA or ARMA models we used previously. In reality, we would need a VARIMA model instead of an ARIMA model. The following table summarizes all the results.

	sMdAPE	***MdRAE***	***GMRAE***	***MSIS***
Persistence model ($\hat{V}_t = V_{t-1}$)	10.9589	1	1	-
Koyck DL	10.9670	1.0279	0.9184	3.1767
Koyck DL + ARMA	11.1738	1.0296	0.9302	4.7882
Koyck DL + GARCH	-	-	-	3.1760
ARDL	3.8429	0.3241	0.3332	2.2023
ARDL + ARMA	3.7305	0.3231	0.3267	4.3839
ARDL + GARCH	-	-	-	2.2144
Harmonic dyn. LS	11.6064	1.0026	1.0333	3.7321
Harmonic dyn. QR	12.236	1	0.8486	3.7268
Harmonic dyn. QR + ARMA	12.236	0.9831	0.9483	5.9593
Harmonic dyn. QR + GARCH	-	-	-	5.9879
Additive dyn. LS	11.5978	1.0005	1.0258	3.7338
Additive dyn. LS + ARMA	11.4870	1.0312	1.0207	6.4219
Additive dyn. LS + GARCH	-	-	-	6.4423
Additive dyn. QR	12.5642	1.0707	0.8674	3.7671
$\log(V_t/O_t) \sim O_t + t$ dyn. LS	7.3699	0.6929	0.6585	2.9117
Harmonic dyn. LS + BAIC	11.5091	1.0069	0.9953	3.5474
Harmonic dyn. QR + BAIC	12.3757	0.9976	0.9327	3.5474
QR-Comb.2 (ARDL + BAIC-Harmonic dyn. QR)	2.1306	0.1794	0.1856	2.6608
QR-Comb.3 (ARDL + BAIC-Harmonic dyn. QR + Add. LS)	1.9725	0.1852	0.1729	2.0923
QR-Comb.3 (ARIMA) ARDL + BAIC-Harmonic dyn. QR + Add. LS	1.7474	0.1485	0.1543	2.7153
QR-Comb.4 (ARIMA) Koyck + ARDL + BAIC-Harmonic dyn. QR + Add. LS	1.7554	0.1448	0.1580	2.7284

Table 1: *Summary of MSIS, relative errors & percentage errors*