Time Series Project

Andrea D'Amicis 869008

2024-11-20

Contents

Data Exploration and Preprocessing

Dataset

In this section i make a brief exploratory analysis of time serie.

```
knitr::opts_chunk$set(message = FALSE)
# Set workspace directory
setwd("C:/Users/andre/Documents/Unimib/Magistrale/2°anno/Time Serie Management")

# Load dataset
data <- read.csv("ts2024_indexes.csv")

# Visualize head of time serie
head(data)
```

```
##              DateTime       Date Hour      X
## 1 2015-01-01 00:00:00 2015-01-01    0 0.0146
## 2 2015-01-01 01:00:00 2015-01-01    1 0.0148
## 3 2015-01-01 02:00:00 2015-01-01    2 0.0101
## 4 2015-01-01 03:00:00 2015-01-01    3 0.0060
## 5 2015-01-01 04:00:00 2015-01-01    4 0.0055
## 6 2015-01-01 05:00:00 2015-01-01    5 0.0071
```

Time serie has the following variables:

- DateTime
- Date
- Hour field
- X which refers to the time series to predict

```
# Descriptive statistics summary
require(skimr)
skim_without_charts(data)
```

Table 1: Data summary

Name	data
Number of rows	17544
Number of columns	4
Column type frequency:	
character	2
numeric	2

Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
DateTime	0	1	19	19	0	17542	0
Date	0	1	10	10	0	731	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
Hour	0	1.00	11.50	6.92	0	5.75	11.50	17.25	23.00
X	744	0.96	0.05	0.05	0	0.02	0.04	0.05	0.45

Excluding time variables, the time series is approximately 95 per cent complete, indicating that there are missing data corresponding to **744 observations but regards the daily predictions** that we have to provide in the final part of project. The values of the series are **between 0 and 0.45** with a mean of 0.04631961 and a rather moderate standard deviation of 0.04894477. The 50th percentile is 0.0368 indicating that 50% of the series has values above this value. In addition, the 75th percentile, with a value of 0.0538, is also rather low compared to the highest value in the series. I expect to see the series remaining mostly below this value and reaching with several peaks the value of the maximum.

```
library(xts)

data$DateTime <- as.POSIXct(data$DateTime, by = "hour",
                            format="%Y-%m-%d %H:%M:%S")

ts_data <- xts(data$X, order.by = data$DateTime)
```

As I expected earlier from the percentiles, we have a time series where most values are below 0.05 approximately with several peaks throughout the time series which probably are outliers but i investigate further in the next chunks after NA investigation.

```
library(xts)
library(forecast)

data$DateTime <- as.POSIXct(data$DateTime, format="%Y-%m-%d %H:%M:%S")

ts_data <- xts(data$X, order.by = data$DateTime)
```

Here i create the eXtensible Time Serie object.

```
ts_cutted <- ts_data[1:16800]
```

First of all we are taking into account the values provided without the “forecast” part.

Zero and NA values

```
# Initialize counters
count_na <- 0
count_zero <- 0

# Cycle the serie to investigate
for (i in 1:length(ts_cutted)) {
  if (is.na(ts_cutted[i])) {
    count_na <- count_na + 1  # Conta i valori NA
  } else if (ts_cutted[i] == 0) {
    count_zero <- count_zero + 1  # Conta i valori zero
  }
}

# Visualize results
cat("Number of NA values:", count_na, "\n")
```

```
## Number of NA values: 0
```

```
cat("Number of zero values:", count_zero, "\n")
```

```
## Number of zero values: 107
```

Here i count the null or zeros values of time serie and there are 107 zeros.

```
# Calculate median of time serie
median_value <- median(ts_cutted, na.rm = TRUE)

# Substitute missing values
ts_filled <- ts_cutted
ts_filled[ts_filled == 0] <- median_value
```

Substituted every zero with the **manual imputation** of median relative to entire dataset.

```
# Initialize counters
count_na <- 0
count_zero <- 0

# Cycle the serie to investigate
for (i in 1:length(ts_filled)) {
  if (is.na(ts_filled[i])) {
    count_na <- count_na + 1  # Conta i valori NA
  } else if (ts_filled[i] == 0) {
    count_zero <- count_zero + 1  # Conta i valori zero
  }
}

# Visualize results
cat("Number of NA values:", count_na, "\n")
```

```
## Number of NA values: 0
```

```r
cat("Number of zero values:", count_zero, "\n")
```

```
## Number of zero values: 0
```

Check if there are any zeros or NA after first preprocessing.

```r
ts_cutted[ts_cutted == 0] <- NA
ts_cutted_filled <- ts_cutted
# Sostituzione manuale dei valori mancanti
for (i in 1:length(ts_cutted_filled)) {
  if (is.na(ts_cutted_filled[i])) {
    start <- max(1, i - 12)  # Imposta il range della finestra
    end <- min(length(ts_cutted_filled), i + 12)
    ts_cutted_filled[i] <- round(median(ts_cutted_filled[start:end], na.rm = TRUE), 5)
  }
}
```

Substituted every zeros with the **moving median** centered on the 24-hour window (12 hours before and 12 hours after)

```r
# Initialize counters
count_na <- 0
count_zero <- 0

# Cycle the serie to investigate
for (i in 1:length(ts_cutted_filled)) {
  if (is.na(ts_cutted_filled[i])) {
    count_na <- count_na + 1  # Conta i valori NA
  } else if (ts_cutted_filled[i] == 0) {
    count_zero <- count_zero + 1  # Conta i valori zero
  }
}

# Visualize results
cat("Number of NA values:", count_na, "\n")
```

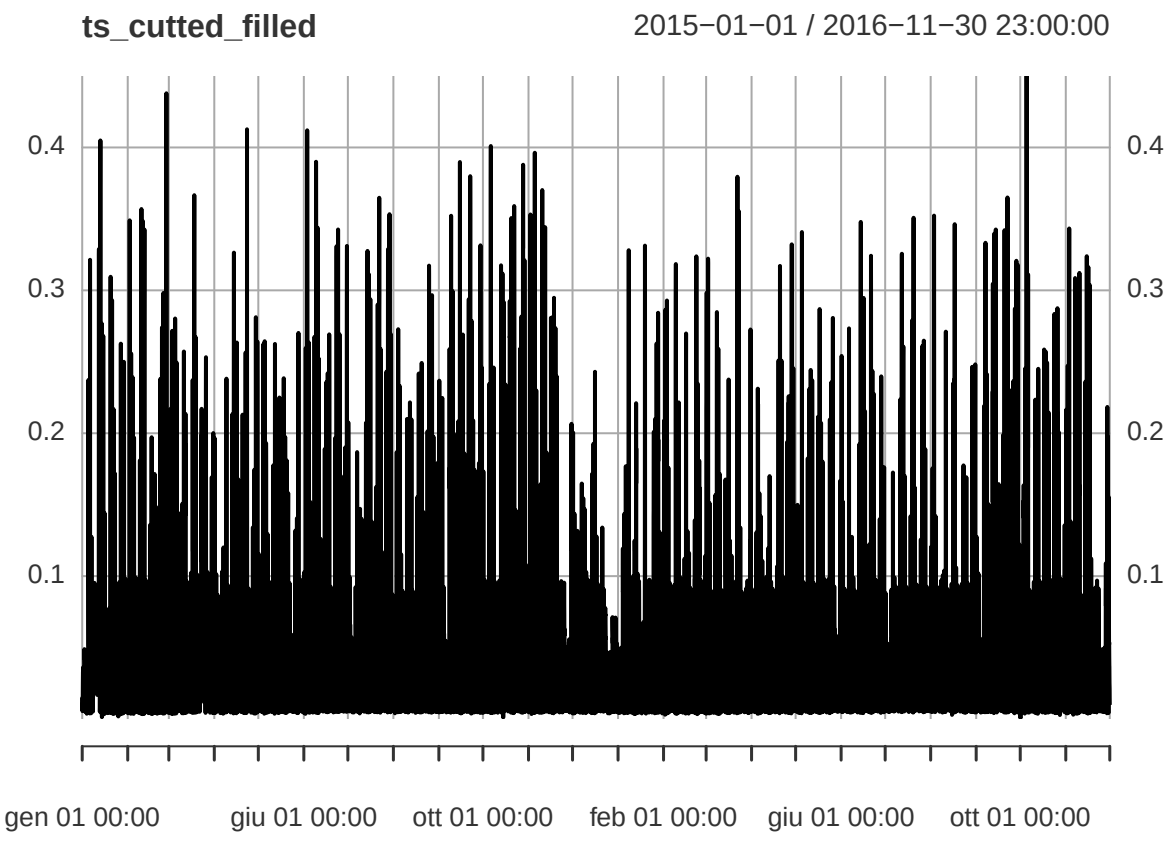
```
## Number of NA values: 0
```

```r
cat("Number of zero values:", count_zero, "\n")
```

```
## Number of zero values: 0
```

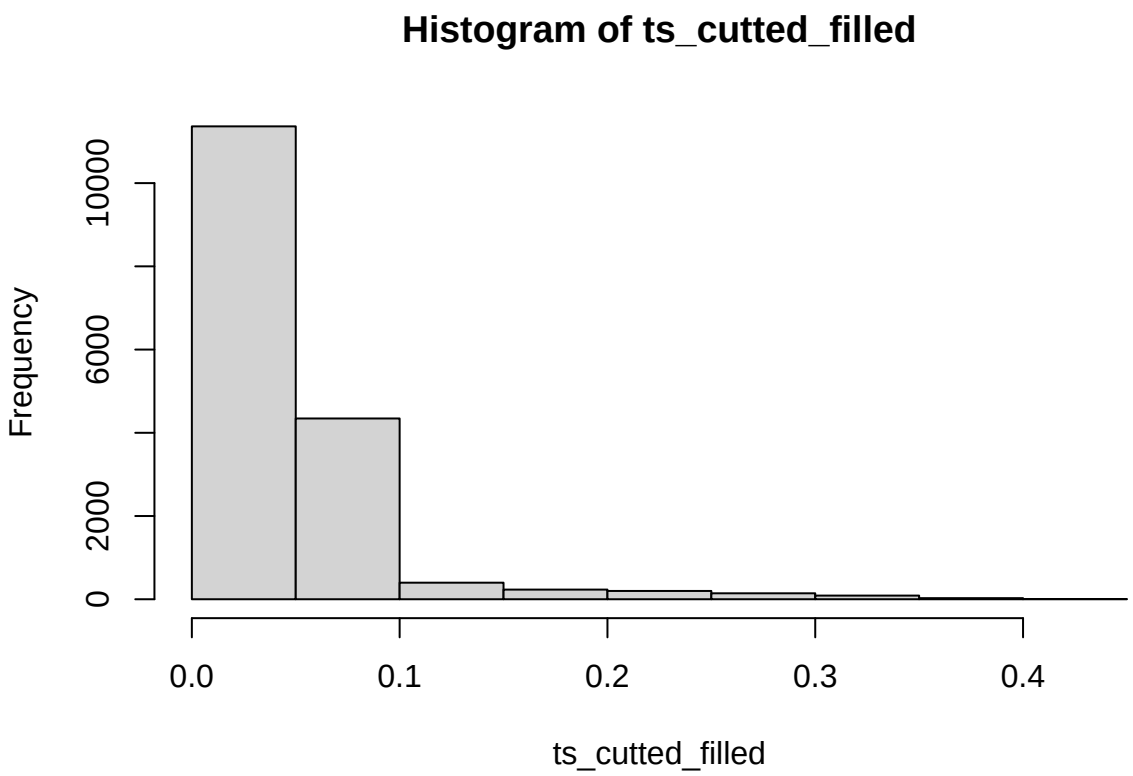
Check if there are any zeros or NA after preprocessing

```r
plot(ts_cutted_filled)
```



Outliers

```
hist(ts_cutted_filled)
```



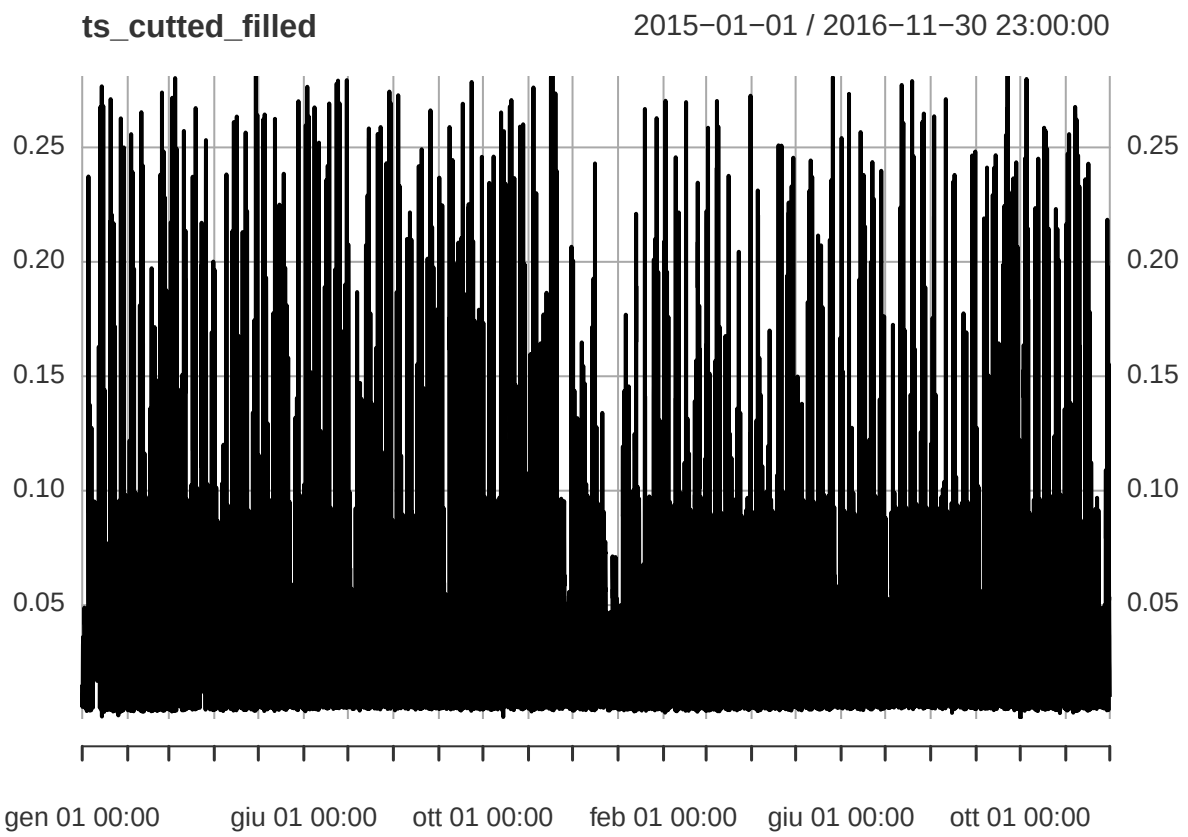
Probably the values belonging to the low frequency class are to be considered as outliers. It's reasonable to replace them with median. So, after determining a quantile i replace this values.

```
percentile_99 <- quantile(ts_cutted_filled, probs = 0.99)
count_above_99th <- sum(ts_cutted_filled > percentile_99, na.rm = TRUE)
count_above_99th
```

```
## [1] 168
```

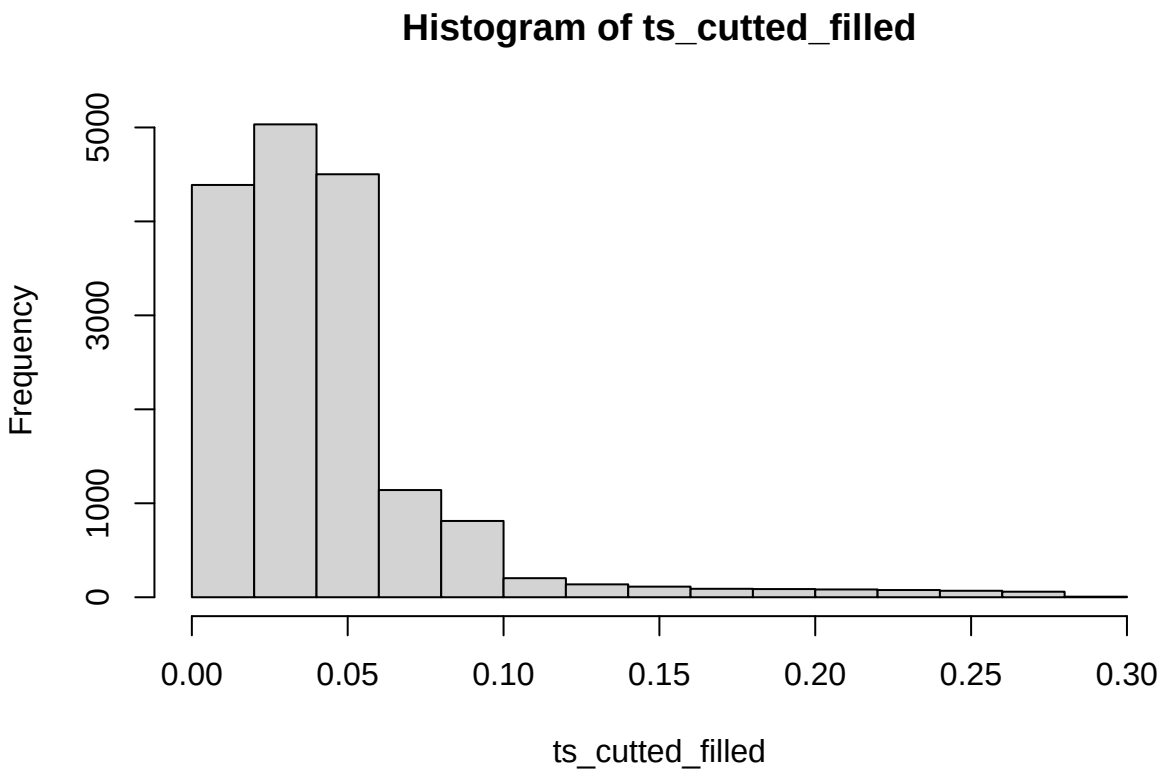
There are only 168 observations that are outliers which are very low in comparison with the whole number of instances in the dataset. Let's substitute them with median value previously calculated.

```
ts_cutted_filled[ts_cutted_filled > percentile_99] <- median_value
plot(ts_cutted_filled, type = "l")
```



Now the time serie it's very similar across the entire time space.

```
hist(ts_cutted_filled)
```



The histogram now reflect the new properties of the time serie.

```
head(ts_cutted_filled)
```

```
##                       [,1]
## 2015-01-01 00:00:00 0.0146
## 2015-01-01 01:00:00 0.0148
## 2015-01-01 02:00:00 0.0101
## 2015-01-01 03:00:00 0.0060
## 2015-01-01 04:00:00 0.0055
## 2015-01-01 05:00:00 0.0071
```

```
file_path <- "C:/Users/andre/Desktop/ts_cutted_filled.csv"
# Assumendo che ts_cutted_filled sia una serie temporale (ts o xts)
# Converti la serie temporale in un data frame
ts_cutted_filled_df <- data.frame(
  Date = time(ts_cutted_filled),  # Ottieni le date
  Value = as.numeric(ts_cutted_filled)  # Ottieni i valori numerici
)

# Salva il data frame in un file CSV
#write.csv(ts_cutted_filled_df, file_path, row.names = FALSE)
```

Dataset preprocessed

In this section i take datasets and divide it into 24 time series with daily data and then in training and test.

```
library(dplyr)
knitr::opts_chunk$set(message = FALSE)
# Set workspace directory
setwd("C:/Users/andre/Documents/Unimib/Magistrale/2°anno/Time Serie Management")

# Load datasets
data <- read.csv("ts2024_preprocessed.csv")
data_dummies <- read.csv("ts2024_dummies.csv")

#data_dummies <- data_dummies %>% select(-Lag_X)
data_dummies <- data_dummies
```

```
dim(data)
```

```
## [1] 17544     4
```

```
dim(data_dummies)
```

```
## [1] 17544    19
```

Dividing date in 24 time series

Below i create a list containing 24 time series, one for a specific hour.

```
# Supponendo che il dataset abbia una colonna 'Hour' (da 0 a 23) e 'value' (la serie temporale)
# Inizializzare una lista per contenere le 24 serie temporali
hourly_series <- list()
hourly_series_dummy <- list()

# Ciclo per filtrare e salvare ogni serie oraria
for (hour in 0:23) {
  # Filtra i dati per l'ora specifica
  hourly_series[[hour + 1]] <- data %>% filter(Hour == hour)
  hourly_series_dummy[[hour + 1]] <- data_dummies %>% filter(Hour == hour)
}

# Visualizzare la serie per una specifica ora (ad esempio, per le 12)
head(hourly_series[[13]])  # L'elemento 13 corrisponde alle 12:00 perche R indicizza da 1 e non 0
```

```
##              DateTime       Date Hour      X
## 1 2015-01-01 12:00:00 2015-01-01   12 0.0329
## 2 2015-01-02 12:00:00 2015-01-02   12 0.0489
## 3 2015-01-03 12:00:00 2015-01-03   12 0.0454
## 4 2015-01-04 12:00:00 2015-01-04   12 0.0123
## 5 2015-01-05 12:00:00 2015-01-05   12 0.0447
## 6 2015-01-06 12:00:00 2015-01-06   12 0.0489
```

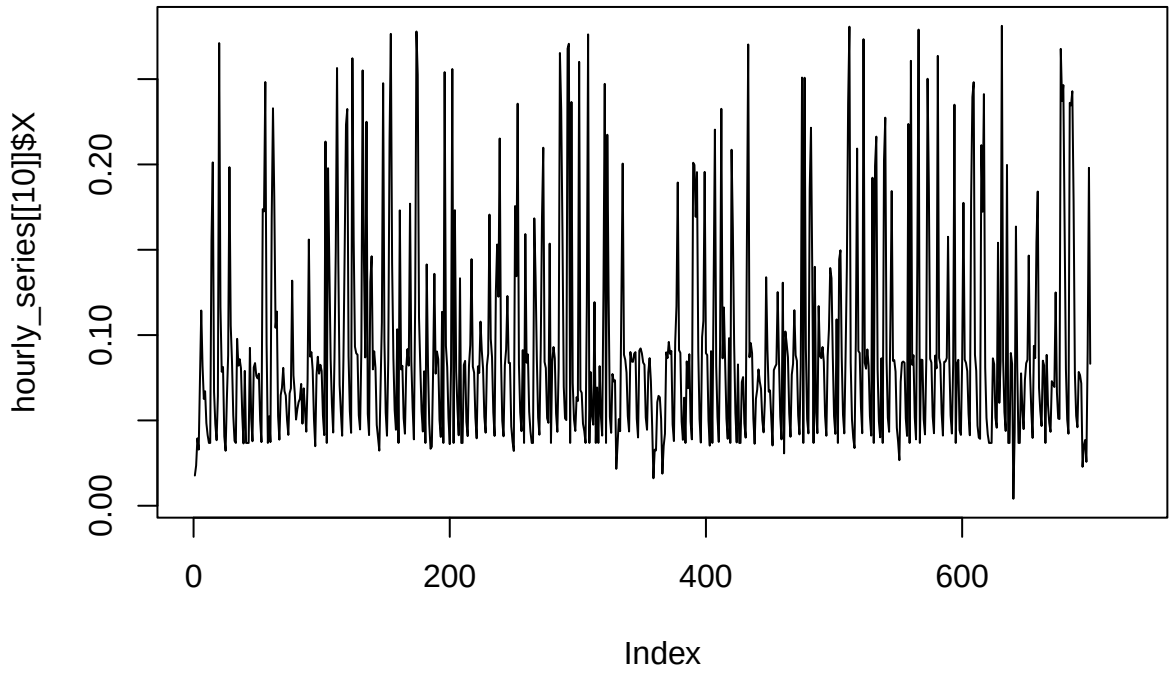
```r
nrow(hourly_series[[10]])
```

```
## [1] 731
```

```r
nrow(hourly_series_dummy[[10]])
```

```
## [1] 731
```

```r
plot(hourly_series[[10]]$X, type = "l")
```



Box-Cox and check Stationarity for 24 time series

Let's investigate the optimal values of lambda with Box-Cox transformation for each time serie.

```r
# Carica la libreria forecast
library(forecast)

# Lista per salvare i valori di lambda ottimali
optimal_lambdas <- numeric(24)

# Ciclo per calcolare il lambda ottimale per ciascuna delle 24 serie orarie
```

```r
for (hour in 0:23) {
  # Recupera la serie per l'ora 'hour' dalla lista 'hourly_series'
  hour_data <- hourly_series[[hour + 1]]$X  # Assicurati che hourly_series contenga delle serie tempora

  # Verifica che 'hour_data' sia un vettore numerico e contiene solo valori positivi
  if (is.numeric(hour_data) && all(hour_data > 0, na.rm = TRUE)) {
    # Trova il lambda ottimale usando BoxCox.lambda
    optimal_lambdas[hour + 1] <- BoxCox.lambda(hour_data)
  } else {
    optimal_lambdas[hour + 1] <- NA  # Se non è un vettore numerico o contiene valori <= 0, restituisce
  }
}

# Visualizza i risultati
print(optimal_lambdas)
```

```
##  [1] -0.99995568 -0.99995223 -0.99992471 -0.28183339 -0.03410270 -0.40158558
##  [7]  0.07960928 -0.14448631 -0.30520352 -0.33984977 -0.22154101  1.99992425
## [13]  1.99992425  1.99992425  0.48873312  0.10206000  0.08864715  1.09296178
## [19]  0.05761952  0.87154550  0.64040828 -0.75452759 -0.25330359 -0.91224536
```

λ	Transformation
-2	$\frac{1}{x^2}$
-1	$\frac{1}{x}$
-0.5	$\frac{1}{\sqrt{x}}$
0	$\log(x)$
0.5	$\sqrt{x}$
1	x
2	x^2

Below i test if the 24 time series are stationary or not.

```r
library(forecast)
library(tseries)  # Per il test ADF

# Lista per salvare i risultati di stazionarietà
stationarity_results <- data.frame(
  Hour = 0:23,
  IsStationary = rep(NA, 24),
  PValue = rep(NA, 24),
  TauStatistic = rep(NA, 24),
  CriticalValue5Pct = rep(NA, 24)
)

# Ciclo per verificare la stazionarietà di ciascuna serie
for (hour in 0:23) {
  # Recupera la serie per l'ora 'hour' dalla lista 'hourly_series'
  hour_data <- hourly_series[[hour + 1]]$X  # Assicurati che hourly_series contenga delle serie tempora

  # Verifica che 'hour_data' sia un vettore numerico
  if (is.numeric(hour_data)) {
    # Applica la trasformazione di Box-Cox se il lambda ottimale è valido
```

```r
    lambda <- optimal_lambdas[hour + 1]
    if (!is.na(lambda) && all(hour_data > 0, na.rm = TRUE)) {
      transformed_data <- BoxCox(hour_data, lambda)
    } else {
      transformed_data <- hour_data  # Nessuna trasformazione se lambda non valido
    }

    # Rimuovi eventuali NA prima del test
    transformed_data <- na.omit(transformed_data)

    # Applica il test di Dickey-Fuller aumentato (ADF)
    adf_test <- adf.test(transformed_data, alternative = "stationary")

    # Estrai il valore critico a livello del 5%
    critical_values <- c(-3.43, -2.86, -2.57)  # Esempio: dipende dal numero di osservazioni (tau2 per
    tau_statistic <- adf_test$statistic
    critical_value_5pct <- critical_values[2]  # Supponiamo il livello al 5%

    # Determina se la serie è stazionaria
    is_stationary <- tau_statistic < critical_value_5pct

    # Salva i risultati
    stationarity_results$IsStationary[hour + 1] <- is_stationary
    stationarity_results$PValue[hour + 1] <- adf_test$p.value
    stationarity_results$TauStatistic[hour + 1] <- tau_statistic
    stationarity_results$CriticalValue5Pct[hour + 1] <- critical_value_5pct
  } else {
    # Se non è un vettore numerico, restituisci NA
    stationarity_results$IsStationary[hour + 1] <- NA
    stationarity_results$PValue[hour + 1] <- NA
    stationarity_results$TauStatistic[hour + 1] <- NA
    stationarity_results$CriticalValue5Pct[hour + 1] <- NA
  }
}
```

```
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
```

```
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(transformed_data, alternative = "stationary"): p-value
## smaller than printed p-value
```

```r
# Visualizza i risultati
print(stationarity_results)
```

```
##    Hour IsStationary     PValue TauStatistic CriticalValue5Pct
## 1     0         TRUE 0.01000000    -4.917948             -2.86
## 2     1         TRUE 0.01000000    -5.837128             -2.86
## 3     2         TRUE 0.01000000    -6.839808             -2.86
## 4     3         TRUE 0.01000000    -6.895324             -2.86
## 5     4         TRUE 0.01000000    -6.445611             -2.86
## 6     5         TRUE 0.01000000    -7.580303             -2.86
## 7     6         TRUE 0.01000000    -6.412932             -2.86
## 8     7         TRUE 0.01000000    -7.458907             -2.86
## 9     8         TRUE 0.01000000    -7.478475             -2.86
## 10    9         TRUE 0.01000000    -7.588359             -2.86
## 11   10         TRUE 0.01000000    -7.111670             -2.86
## 12   11         TRUE 0.01000000    -4.679636             -2.86
## 13   12         TRUE 0.01000000    -5.266351             -2.86
## 14   13         TRUE 0.01000000    -5.490569             -2.86
## 15   14         TRUE 0.01000000    -6.621222             -2.86
## 16   15         TRUE 0.01000000    -7.116080             -2.86
## 17   16         TRUE 0.01000000    -8.229424             -2.86
## 18   17         TRUE 0.01000000    -6.939559             -2.86
## 19   18         TRUE 0.01000000    -7.507875             -2.86
## 20   19         TRUE 0.01821757    -3.815609             -2.86
## 21   20         TRUE 0.20755762    -2.875880             -2.86
## 22   21         TRUE 0.04758142    -3.445131             -2.86
## 23   22         TRUE 0.01000000    -4.254819             -2.86
## 24   23         TRUE 0.01000000    -5.195442             -2.86
```

Every time series are stationary.

Dividing in Train and Test set

Below i create 2 objects related to plain and dummies time serie. The objects contain 24 time series and, for each of them are generated 2 variables related to train set and test set. This are the structure related to final forecasts.

```
# Funzione per dividere i dati in train e test
train_test_split <- function(hourly_series, train_end) {
  train_set <- hourly_series[1:train_end, ]  # Da 1 a 'train_end' per il train
  test_set <- hourly_series[(train_end+1):nrow(hourly_series), ]  # Rimanenti per il test
  list(train = train_set, test = test_set)
}

# Impostiamo il valore di train_end
train_end <- 700  # Gli ultimi 31 vanno nel test

# Creare una lista per salvare i train e test per tutte le ore per i due dataset
train_test_series <- list()
train_test_series_dummy <- list()

# Ciclo per creare i train e test per ogni ora (0 a 23) per entrambe le serie
for (hour in 0:23) {
  # Per hourly_series
  hour_data <- hourly_series[[hour + 1]]
  train_test_series[[hour + 1]] <- train_test_split(hour_data, train_end)

  # Per hourly_series_dummy
  hour_data_dummy <- hourly_series_dummy[[hour + 1]]
  train_test_series_dummy[[hour + 1]] <- train_test_split(hour_data_dummy, train_end)
}

# Verifica per le 12:00
cat("Train set for 12:00 (original):\n")
```

```
## Train set for 12:00 (original):
```

```
dim(train_test_series[[13]]$train)
```

```
## [1] 700   4
```

```
dim(train_test_series[[13]]$test)
```

```
## [1] 31  4
```

```
cat("Train set for 12:00 (dummy):\n")
```

```
## Train set for 12:00 (dummy):
```

```
dim(train_test_series_dummy[[13]]$train)
```

```
## [1] 700  19
```

```
dim(train_test_series_dummy[[13]]$test)
```

```
## [1] 31 19
```

Dividing in Train and Validation set

```r
# Funzione per dividere i dati in train e test
train_test_split <- function(hourly_series, train_end) {
  train_set <- hourly_series[1:train_end, ]  # Da 1 a 'train_end' per il train
  test_set <- hourly_series[(train_end+1):700, ]  # Fino a 700 per il test
  list(train = train_set, test = test_set)
}

# Impostiamo il valore di train_end
train_end <- 630  # Gli ultimi 70 vanno nel test (da 631 a 700)

# Creare una lista per salvare i train e test per tutte le ore per i due dataset
train_val_series <- list()
train_val_series_dummy <- list()

# Ciclo per creare i train e test per ogni ora (0 a 23) per entrambe le serie
for (hour in 0:23) {
  # Per hourly_series
  hour_data <- hourly_series[[hour + 1]]
  train_val_series[[hour + 1]] <- train_test_split(hour_data, train_end)

  # Per hourly_series_dummy
  hour_data_dummy <- hourly_series_dummy[[hour + 1]]
  train_val_series_dummy[[hour + 1]] <- train_test_split(hour_data_dummy, train_end)
}

# Verifica per le 12:00
cat("Train set for 12:00 (original):\n")
```

```
## Train set for 12:00 (original):
```

```r
dim(train_val_series[[13]]$train)
```

```
## [1] 630   4
```

```r
dim(train_val_series[[13]]$test)
```

```
## [1] 70  4
```

```r
cat("Train set for 12:00 (dummy):\n")
```

```
## Train set for 12:00 (dummy):
```

```r
dim(train_val_series_dummy[[13]]$train)
```

```
## [1] 630  19
```

```r
dim(train_val_series_dummy[[13]]$test)
```

```
## [1] 70 19
```

```r
tail(train_val_series[[1]]$train)
```

```
##                DateTime       Date Hour      X
## 625 2016-09-16 00:00:00 2016-09-16    0 0.0070
```

```
## 626 2016-09-17 00:00:00 2016-09-17    0 0.0101
## 627 2016-09-18 00:00:00 2016-09-18    0 0.0145
## 628 2016-09-19 00:00:00 2016-09-19    0 0.0081
## 629 2016-09-20 00:00:00 2016-09-20    0 0.0061
## 630 2016-09-21 00:00:00 2016-09-21    0 0.0065
```

```r
head(train_val_series[[1]]$test)
```

```
##                DateTime       Date Hour      X
## 631 2016-09-22 00:00:00 2016-09-22    0 0.0058
## 632 2016-09-23 00:00:00 2016-09-23    0 0.0065
## 633 2016-09-24 00:00:00 2016-09-24    0 0.0108
## 634 2016-09-25 00:00:00 2016-09-25    0 0.0132
## 635 2016-09-26 00:00:00 2016-09-26    0 0.0087
## 636 2016-09-27 00:00:00 2016-09-27    0 0.0058
```

```r
head(train_test_series[[1]]$test)
```

```
##                DateTime       Date Hour  X
## 701 2016-12-01 00:00:00 2016-12-01    0 NA
## 702 2016-12-02 00:00:00 2016-12-02    0 NA
## 703 2016-12-03 00:00:00 2016-12-03    0 NA
## 704 2016-12-04 00:00:00 2016-12-04    0 NA
## 705 2016-12-05 00:00:00 2016-12-05    0 NA
## 706 2016-12-06 00:00:00 2016-12-06    0 NA
```

ARIMA (Auto Regressive Integrated Moving Average)

Auto Arima models

Univariate

```r
# Lista per memorizzare i MAE e i parametri
mae_values <- numeric(length = 24)
arima_params <- list()

# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test

  # Ottieni il valore di lambda per questa serie (da optimal_lambdas)
  lambda <- optimal_lambdas[i]

  # Trova il miglior modello ARIMA usando auto.arima
  model <- auto.arima(train_series$X,
                      d = NA,               # Lascia stimare la differenziazione regolare
                      D = 1,                # Forza una differenziazione stagionale
                      seasonal = TRUE,      # Abilita la stagionalità
```

```
                      lambda = lambda,      # Trasformazione Box-Cox, se necessaria
                      stepwise = FALSE,     # Esplora più modelli per ottimizzare
                      approximation = FALSE) # Usa calcoli esatti)

  # Fai una previsione sul test set
  forecast_values <- forecast(model, h = length(test_series$X))

  # Calcola l'errore assoluto medio (MAE)
  mae_values[i] <- mean(abs(forecast_values$mean - test_series$X))

  # Salva i parametri del modello ARIMA
  arima_params[[i]] <- list(
    p = model$arma[1],    # Parametro p (AR)
    d = model$arma[6],    # Parametro d (differencing)
    q = model$arma[2],    # Parametro q (MA)
    seasonal_p = model$arma[3],  # Parametro stagionale p
    seasonal_d = model$arma[7],  # Parametro stagionale d
    seasonal_q = model$arma[4]   # Parametro stagionale q
  )
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)
print("MAE values:")
```

```
## [1] "MAE values:"
```

```
print(mae_values)
```

```
##  [1] 0.002595853 0.001666123 0.001546887 0.004064742 0.013122270 0.028356173
##  [7] 0.045318816 0.074219510 0.045990895 0.045160746 0.017249797 0.005595339
## [13] 0.004248850 0.005278601 0.005005078 0.006415548 0.006249187 0.006112751
## [19] 0.006190386 0.006921889 0.005384159 0.004345308 0.004448492 0.002859169
```

```
print(paste("MAE:", mae))
```

```
## [1] "MAE: 0.0145144404413805"
```

```
# Visualizzare i parametri ARIMA per tutte le serie in un formato leggibile
cat("\nARIMA Parameters for Each Series:\n")
```

```
##
## ARIMA Parameters for Each Series:
```

```
for (i in 1:24) {
  cat(paste("Series", i, ":\n"))
  cat(paste("  p:", arima_params[[i]]$p, "\n"))
  cat(paste("  d:", arima_params[[i]]$d, "\n"))
  cat(paste("  q:", arima_params[[i]]$q, "\n"))
  cat(paste("  Seasonal p:", arima_params[[i]]$seasonal_p, "\n"))
  cat(paste("  Seasonal d:", arima_params[[i]]$seasonal_d, "\n"))
  cat(paste("  Seasonal q:", arima_params[[i]]$seasonal_q, "\n"))
  cat("\n")
}
```

```
## Series 1 :
##   p: 4
```

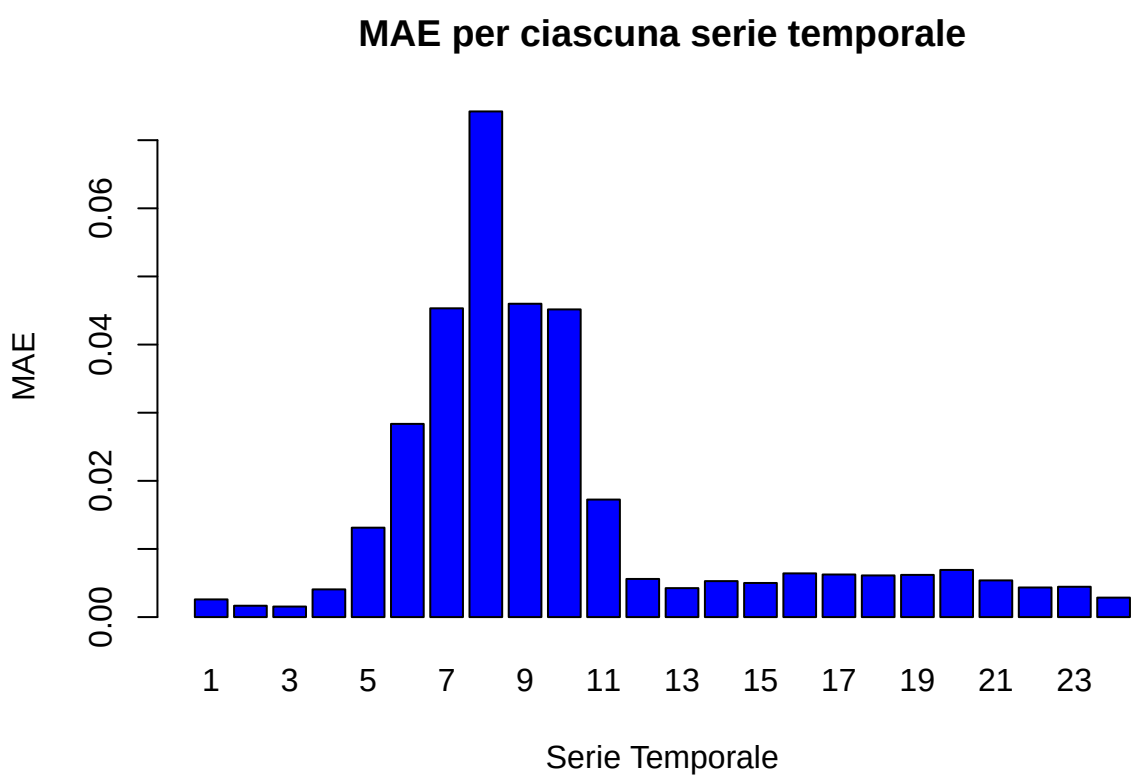
```
##   d: 0
##   q: 1
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 2 :
##   p: 2
##   d: 0
##   q: 2
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 3 :
##   p: 3
##   d: 0
##   q: 2
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 4 :
##   p: 0
##   d: 0
##   q: 5
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 5 :
##   p: 0
##   d: 0
##   q: 5
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 6 :
##   p: 5
##   d: 0
##   q: 0
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 7 :
##   p: 5
##   d: 0
##   q: 0
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
```

```
## Series 8 :
##   p: 5
##   d: 0
##   q: 0
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 9 :
##   p: 5
##   d: 0
##   q: 0
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 10 :
##   p: 4
##   d: 0
##   q: 0
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 11 :
##   p: 4
##   d: 1
##   q: 1
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 12 :
##   p: 4
##   d: 1
##   q: 1
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 13 :
##   p: 4
##   d: 1
##   q: 1
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 14 :
##   p: 2
##   d: 1
##   q: 3
##   Seasonal p: 0
##   Seasonal d: 0
```

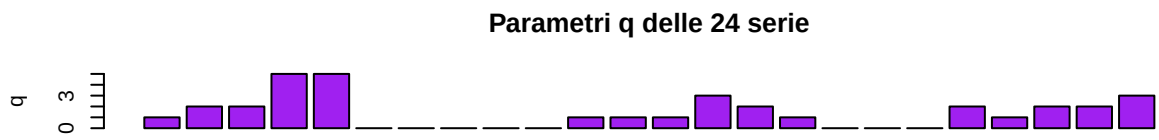
```
##   Seasonal q: 0
##
## Series 15 :
##   p: 0
##   d: 1
##   q: 2
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 16 :
##   p: 4
##   d: 1
##   q: 1
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 17 :
##   p: 5
##   d: 0
##   q: 0
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 18 :
##   p: 5
##   d: 0
##   q: 0
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 19 :
##   p: 5
##   d: 0
##   q: 0
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 20 :
##   p: 3
##   d: 1
##   q: 2
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 21 :
##   p: 4
##   d: 1
##   q: 1
```

```
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 22 :
##   p: 3
##   d: 1
##   q: 2
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 23 :
##   p: 3
##   d: 1
##   q: 2
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
##
## Series 24 :
##   p: 2
##   d: 0
##   q: 3
##   Seasonal p: 0
##   Seasonal d: 0
##   Seasonal q: 0
```

```r
library(ggplot2)
library(forecast)

# Crea un grafico a barre dei MAE per ogni serie temporale
barplot(mae_values,
        names.arg = 1:24,
        col = "blue",
        main = "MAE per ciascuna serie temporale",
        xlab = "Serie Temporale",
        ylab = "MAE")
```



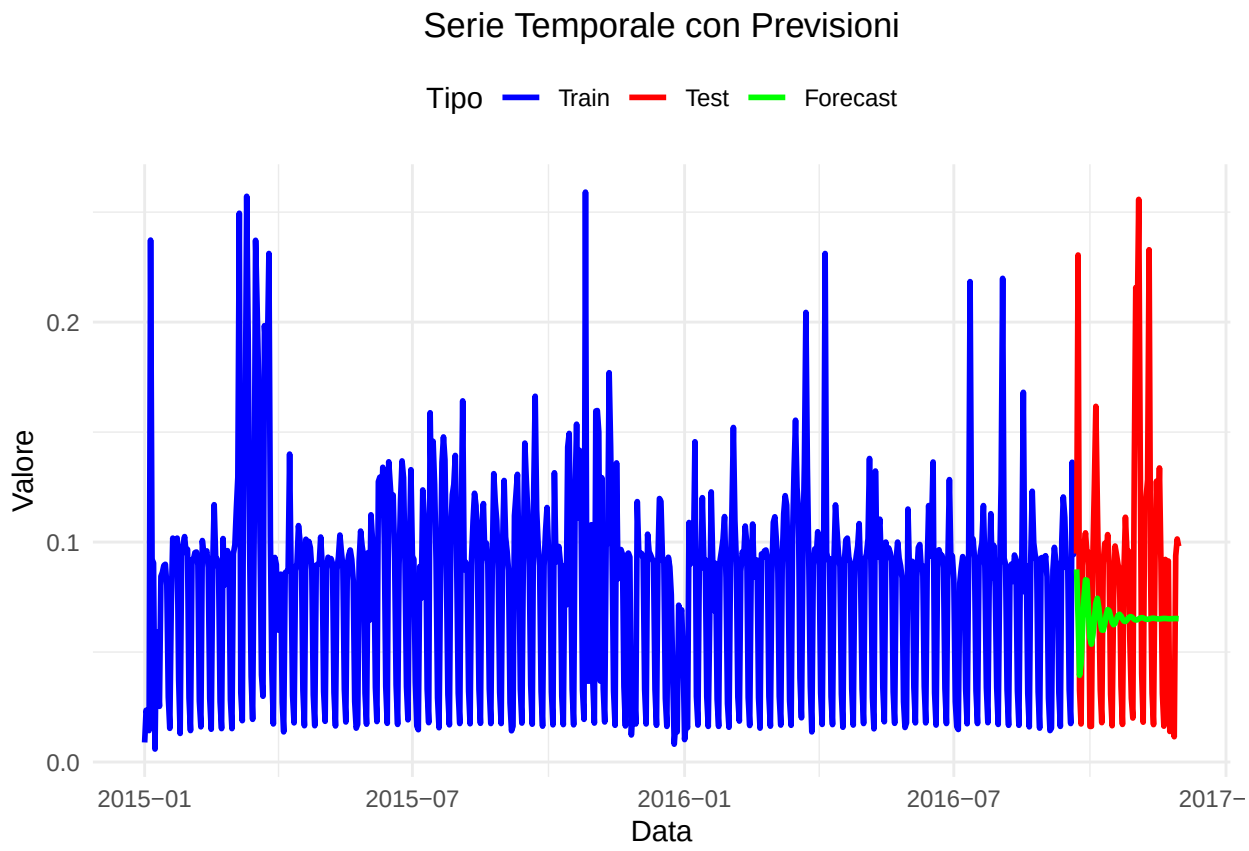

```r
# Visualizzare un grafico per i parametri p, d, q delle 24 serie temporali
p_values <- sapply(arima_params, function(x) x$p)
d_values <- sapply(arima_params, function(x) x$d)
q_values <- sapply(arima_params, function(x) x$q)

# Creare un grafico a barre per i parametri ARIMA
par(mfrow = c(3, 1)) # disposizione del grafico (3 righe, 1 colonna)
barplot(p_values, main = "Parametri p delle 24 serie", col = "green", ylab = "p")
barplot(d_values, main = "Parametri d delle 24 serie", col = "red", ylab = "d")
barplot(q_values, main = "Parametri q delle 24 serie", col = "purple", ylab = "q")
```




```r
library(ggplot2)

# Prendi la prima serie temporale (serie 1)
train_series <- train_val_series[[7]]$train
test_series <- train_val_series[[7]]$test

# Ottieni il valore di lambda per questa serie
lambda <- optimal_lambdas[7]

# Trova il miglior modello ARIMA usando auto.arima
model <- auto.arima(train_series$X,
                    d = NA,                # Lascia stimare la differenziazione regolare
                    D = 1,                 # Forza una differenziazione stagionale
                    seasonal = TRUE,       # Abilita la stagionalità
                    lambda = lambda,       # Trasformazione Box-Cox, se necessaria
                    stepwise = FALSE,      # Esplora più modelli per ottimizzare
                    approximation = FALSE)

# Fai una previsione sul test set
forecast_values <- forecast(model, h = length(test_series$X))

# Crea un data frame per la serie temporale di allenamento
train_data <- data.frame(
  Date = train_series$Date,
  Value = train_series$X,
```

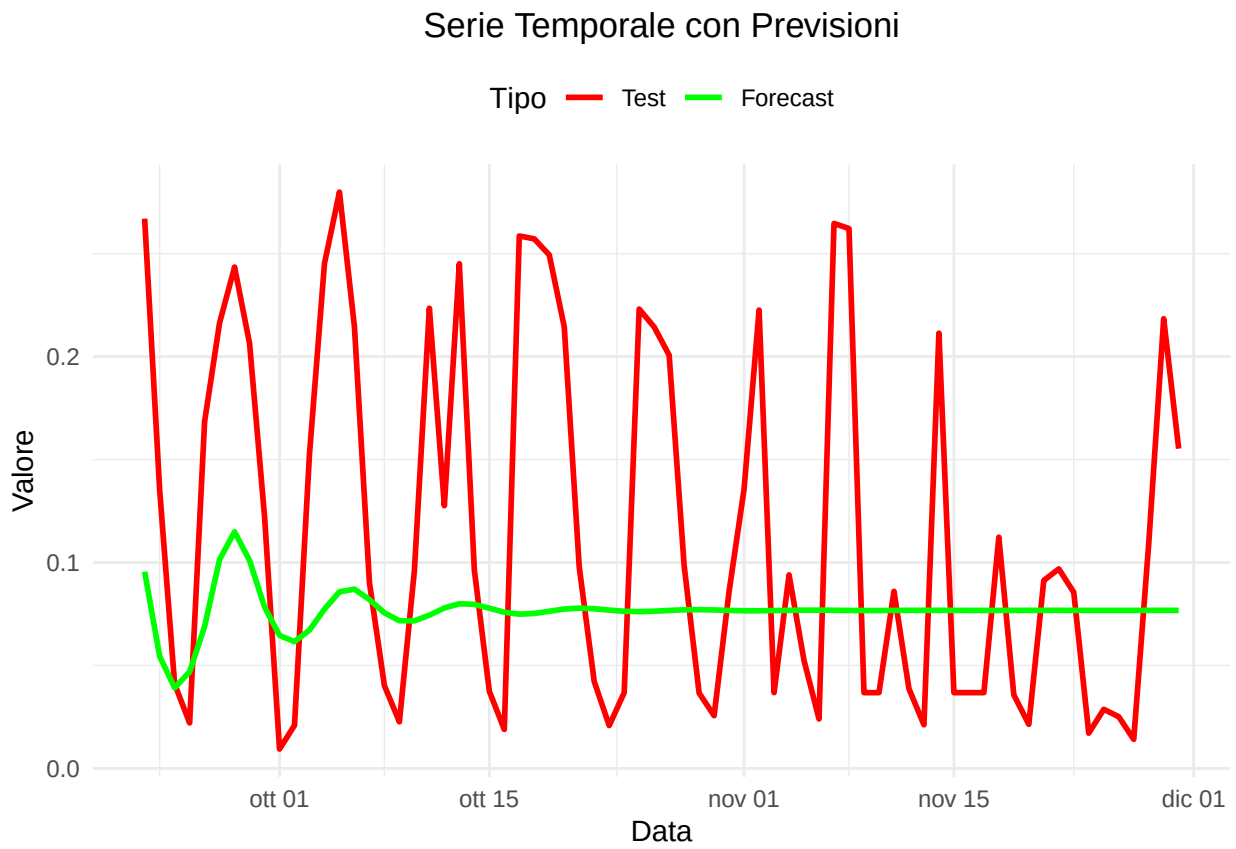
```r
  Type = "Train"
)

# Crea un data frame per la serie temporale di test
test_data <- data.frame(
  Date = test_series$Date,
  Value = test_series$X,
  Type = "Test"
)

# Crea un data frame per le previsioni
forecast_data <- data.frame(
  Date = test_series$Date,  # Usa gli stessi indici di test_series per la previsione
  Value = forecast_values$mean,
  Type = "Forecast"
)

# Assicurati che la colonna Date sia nello stesso formato per tutti i data frame
train_data$Date <- as.Date(train_data$Date)
test_data$Date <- as.Date(test_data$Date)
forecast_data$Date <- as.Date(forecast_data$Date)

# Combina i tre data frame
plot_data <- rbind(train_data, forecast_data, test_data)

# Imposta l'ordine dei livelli per il fattore 'Type'
plot_data$Type <- factor(plot_data$Type, levels = c("Train", "Test", "Forecast"))

# Creazione del grafico con ggplot2
ggplot(plot_data, aes(x = Date, y = Value, color = Type)) +
  geom_line(size = 1) +  # Linee separate per Train, Test e Forecast
  labs(
    title = "Serie Temporale con Previsioni",
    x = "Data",
    y = "Valore",
    color = "Tipo"
  ) +
  scale_color_manual(
    values = c("Train" = "blue", "Test" = "red", "Forecast" = "green")
  ) +
  theme_minimal() +
  theme(
    legend.position = "top",  # Posizionare la legenda in alto
    plot.title = element_text(hjust = 0.5)  # Centrare il titolo
  )
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```r
library(ggplot2)

# Prendi la prima serie temporale (serie 7)
train_series <- train_val_series[[8]]$train
test_series <- train_val_series[[8]]$test

# Ottieni il valore di lambda per questa serie
lambda <- optimal_lambdas[8]

# Trova il miglior modello ARIMA usando auto.arima
model <- auto.arima(
  train_series$X,
  d = NA,                # Lascia stimare la differenziazione regolare
  D = 1,                 # Forza una differenziazione stagionale
  seasonal = TRUE,       # Abilita la stagionalità
  lambda = lambda,       # Trasformazione Box-Cox, se necessaria
  stepwise = FALSE,      # Esplora più modelli per ottimizzare
  approximation = FALSE
)

# Fai una previsione sul test set
forecast_values <- forecast(model, h = length(test_series$X))

# Crea un data frame per la serie temporale di test
test_data <- data.frame(
```

```r
  Date = as.Date(test_series$Date), # Assicura che il formato della data sia coerente
  Value = test_series$X,
  Type = "Test"
)

# Crea un data frame per le previsioni
forecast_data <- data.frame(
  Date = as.Date(test_series$Date),  # Usa le date corrispondenti ai dati di test
  Value = as.numeric(forecast_values$mean),  # Estrai la media delle previsioni
  Type = "Forecast"
)

# Combina i due data frame
plot_data <- rbind(forecast_data, test_data)

# Imposta l'ordine dei livelli per il fattore 'Type'
plot_data$Type <- factor(plot_data$Type, levels = c("Test", "Forecast"))

# Creazione del grafico con ggplot2
ggplot(plot_data, aes(x = Date, y = Value, color = Type)) +
  geom_line(size = 1) +  # Linee separate per Test e Forecast
  labs(
    title = "Serie Temporale con Previsioni",
    x = "Data",
    y = "Valore",
    color = "Tipo"
  ) +
  scale_color_manual(
    values = c("Test" = "red", "Forecast" = "green")
  ) +
  theme_minimal() +
  theme(
    legend.position = "top",  # Posizionare la legenda in alto
    plot.title = element_text(hjust = 0.5)  # Centrare il titolo
  )
```



Arima models (manual)

Univariate

Arima (3,1,1) (0,1,1)(7)

The idea is to detect the overall behavior of the 24 time series and select the best parameters of Arima that achieve the lowest value of the mean of MAE.

```
# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()

# Lista per memorizzare le previsioni
forecasts_list <- list()

# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test

  # Ottieni il valore di lambda per questa serie (da optimal_lambdas)
  lambda <- optimal_lambdas[i]
```

```
  # Specifica i parametri ARIMA manualmente
  p <- 3    # Ordine autoregressivo
  d <- 1    # Ordine di differenziazione
  q <- 1    # Ordine media mobile

  # Parametri stagionali
  seasonal_p <- 0
  seasonal_d <- 1
  seasonal_q <- 1
  m <- 7   # Periodo stagionale orario

  # Costruisci il modello ARIMA manualmente
  model <- Arima(train_series$X,
                 order = c(p, d, q),
                 seasonal = list(order = c(seasonal_p, seasonal_d, seasonal_q), period = m),
                 lambda = lambda)

  # Fai una previsione sul test set
  forecast_values <- forecast(model, h = length(test_series$X))

  # Salva le previsioni in forecasts_list
  forecasts_list[[i]] <- forecast_values$mean

  # Calcola l'errore assoluto medio (MAE)
  mae_values[i] <- mean(abs(forecast_values$mean - test_series$X))

  # Salva i residui del modello
  residuals_list[[i]] <- residuals(model)
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```
print(mae_values)
```

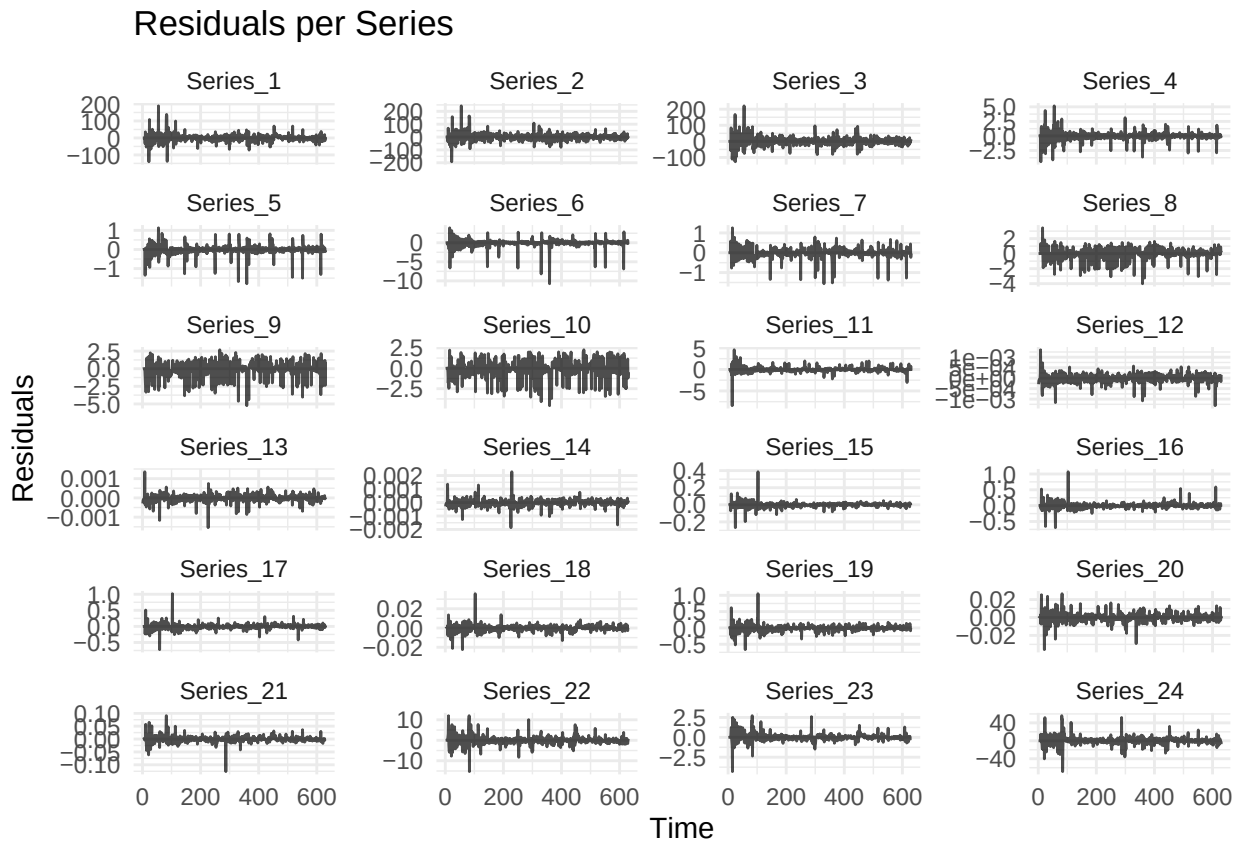
```
##  [1] 0.001705055 0.001523074 0.001544725 0.002318464 0.005589199 0.007445938
##  [7] 0.020672805 0.048379729 0.034659147 0.037532561 0.015521326 0.005087556
## [13] 0.003880511 0.004675762 0.004903802 0.005144033 0.003949046 0.003816586
## [19] 0.003809891 0.005293339 0.003652791 0.002720638 0.003355387 0.002335071
```

```
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.009563185
```

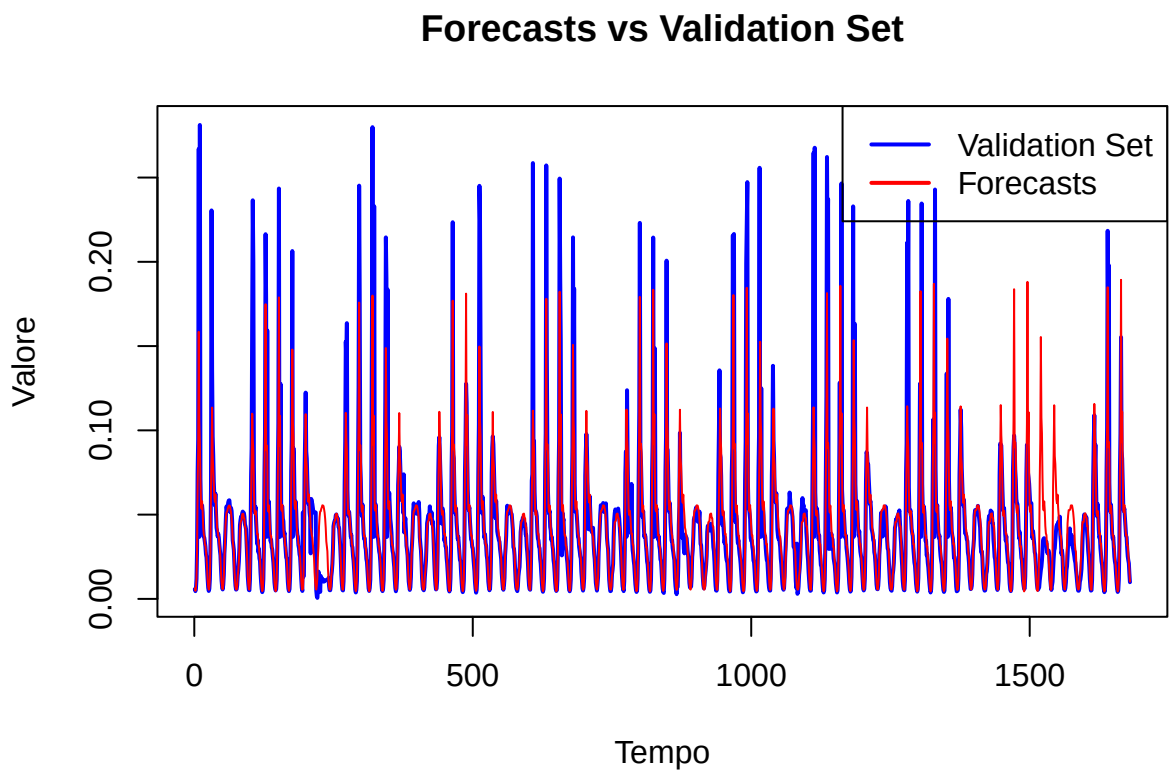
```
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
```

```r
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecasts_list)
series_length <- length(forecasts_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecasts_list[[i]][t]
```

```
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)

# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



Arima (3,1,1) (0,1,2)(7)

```
# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()
```

```r
# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test

  # Ottieni il valore di lambda per questa serie (da optimal_lambdas)
  lambda <- optimal_lambdas[i]

  # Specifica i parametri ARIMA manualmente
  p <- 3    # Ordine autoregressivo
  d <- 1    # Ordine di differenziazione
  q <- 1    # Ordine media mobile

  # Parametri stagionali
  seasonal_p <- 0
  seasonal_d <- 1
  seasonal_q <- 2
  m <- 7   # Periodo stagionale orario

  # Costruisci il modello ARIMA manualmente
  model <- Arima(train_series$X,
                 order = c(p, d, q),
                 seasonal = list(order = c(seasonal_p, seasonal_d, seasonal_q), period = m),
                 lambda = lambda)

  # Fai una previsione sul test set
  forecast_values <- forecast(model, h = length(test_series$X))

  # Salva le previsioni in forecasts_list
  forecasts_list[[i]] <- forecast_values$mean

  # Calcola l'errore assoluto medio (MAE)
  mae_values[i] <- mean(abs(forecast_values$mean - test_series$X))

  # Salva i residui del modello
  residuals_list[[i]] <- residuals(model)
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```r
print(mae_values)
```

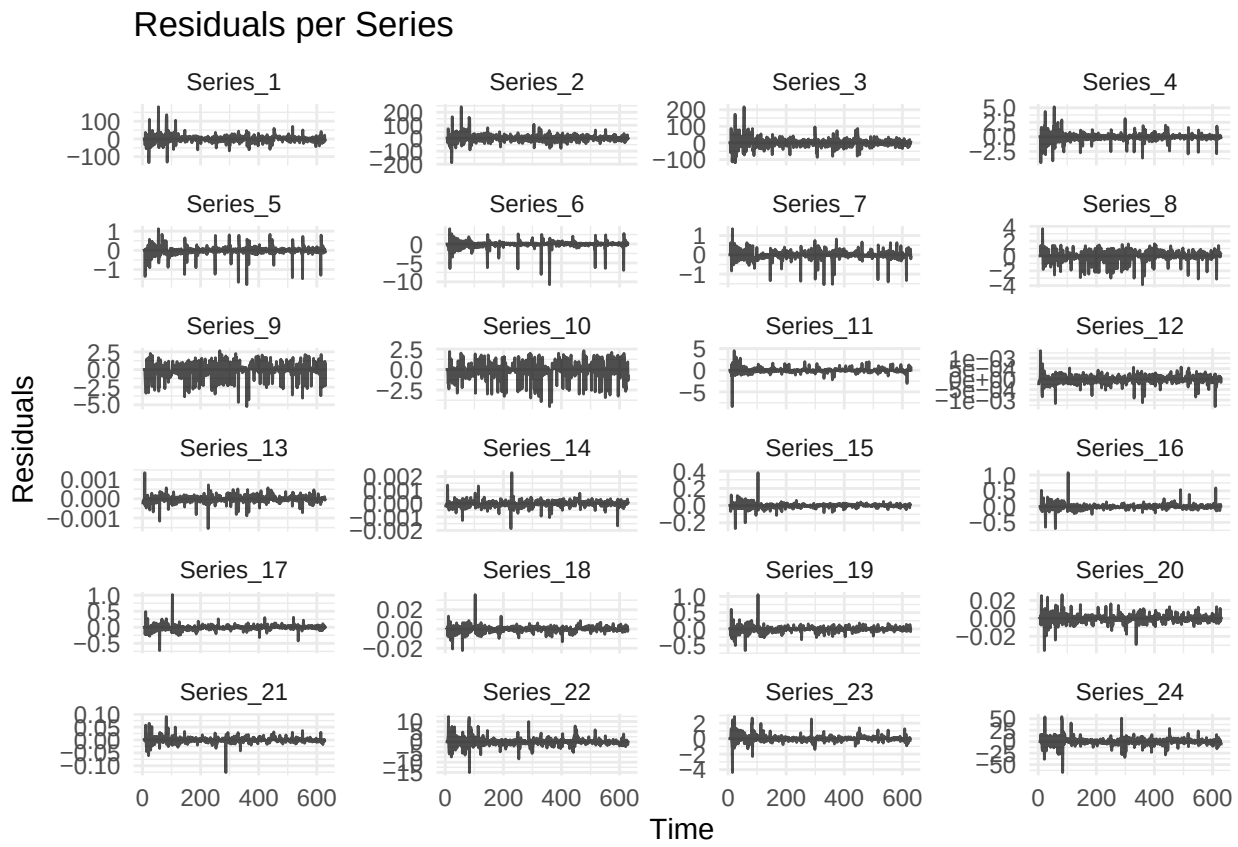
```
##  [1] 0.001677170 0.001544360 0.001528894 0.002317808 0.005613577 0.006938096
##  [7] 0.020171223 0.048752564 0.034668703 0.037610432 0.015469548 0.005158399
## [13] 0.003827822 0.004678003 0.004709634 0.005143165 0.003889474 0.003689533
## [19] 0.003846640 0.005294001 0.003739821 0.003107730 0.003645783 0.002378751
```

```
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.00955838
```

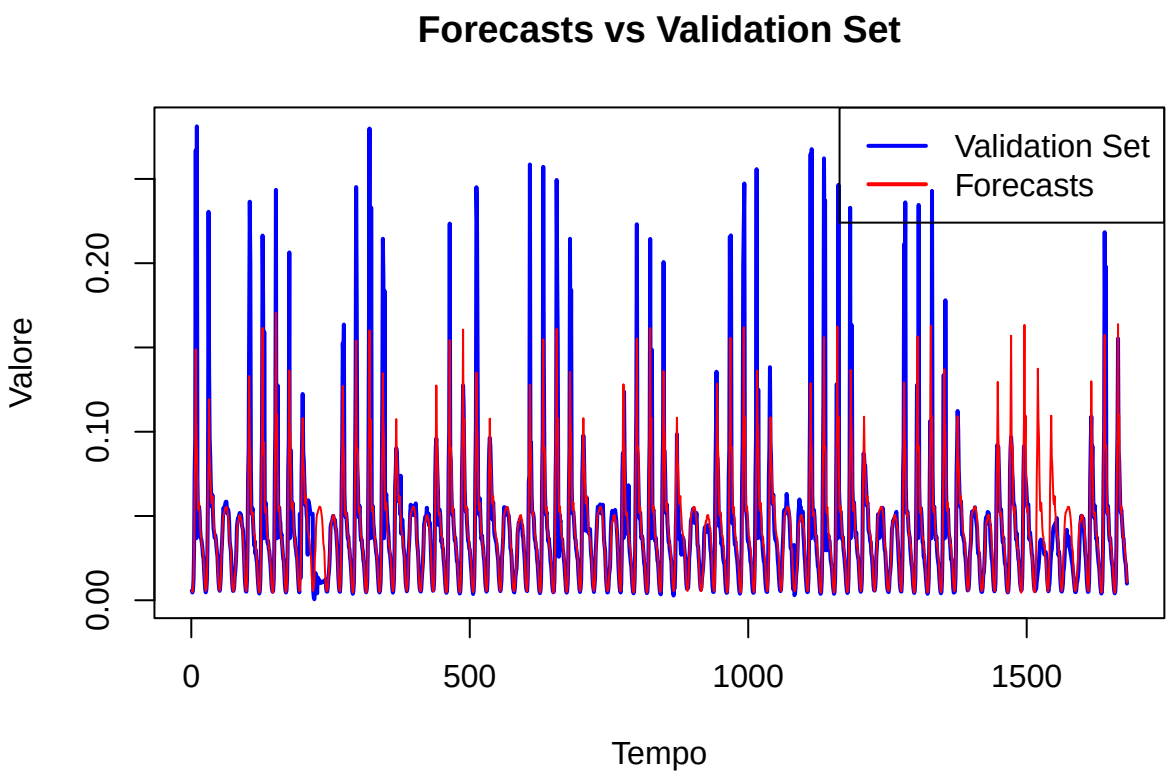
There is still one lag significant lag. I take another attempt.

```
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecasts_list)
```

```r
series_length <- length(forecasts_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecasts_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)

# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



Arima (3,1,1) (1,1,1)(7)

```
# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()

# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test

  # Ottieni il valore di lambda per questa serie (da optimal_lambdas)
  lambda <- optimal_lambdas[i]

  # Specifica i parametri ARIMA manualmente
  p <- 3    # Ordine autoregressivo
  d <- 1    # Ordine di differenziazione
  q <- 1    # Ordine media mobile

  # Parametri stagionali
  seasonal_p <- 1
  seasonal_d <- 1
  seasonal_q <- 1
```

```
  m <- 7   # Periodo stagionale orario

  # Costruisci il modello ARIMA manualmente
  model <- Arima(train_series$X,
                 order = c(p, d, q),
                 seasonal = list(order = c(seasonal_p, seasonal_d, seasonal_q), period = m),
                 lambda = lambda)

  # Fai una previsione sul test set
  forecast_values <- forecast(model, h = length(test_series$X))

  # Salva le previsioni in forecasts_list
  forecasts_list[[i]] <- forecast_values$mean

  # Calcola l'errore assoluto medio (MAE)
  mae_values[i] <- mean(abs(forecast_values$mean - test_series$X))

  # Salva i residui del modello
  residuals_list[[i]] <- residuals(model)
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```
print(mae_values)
```

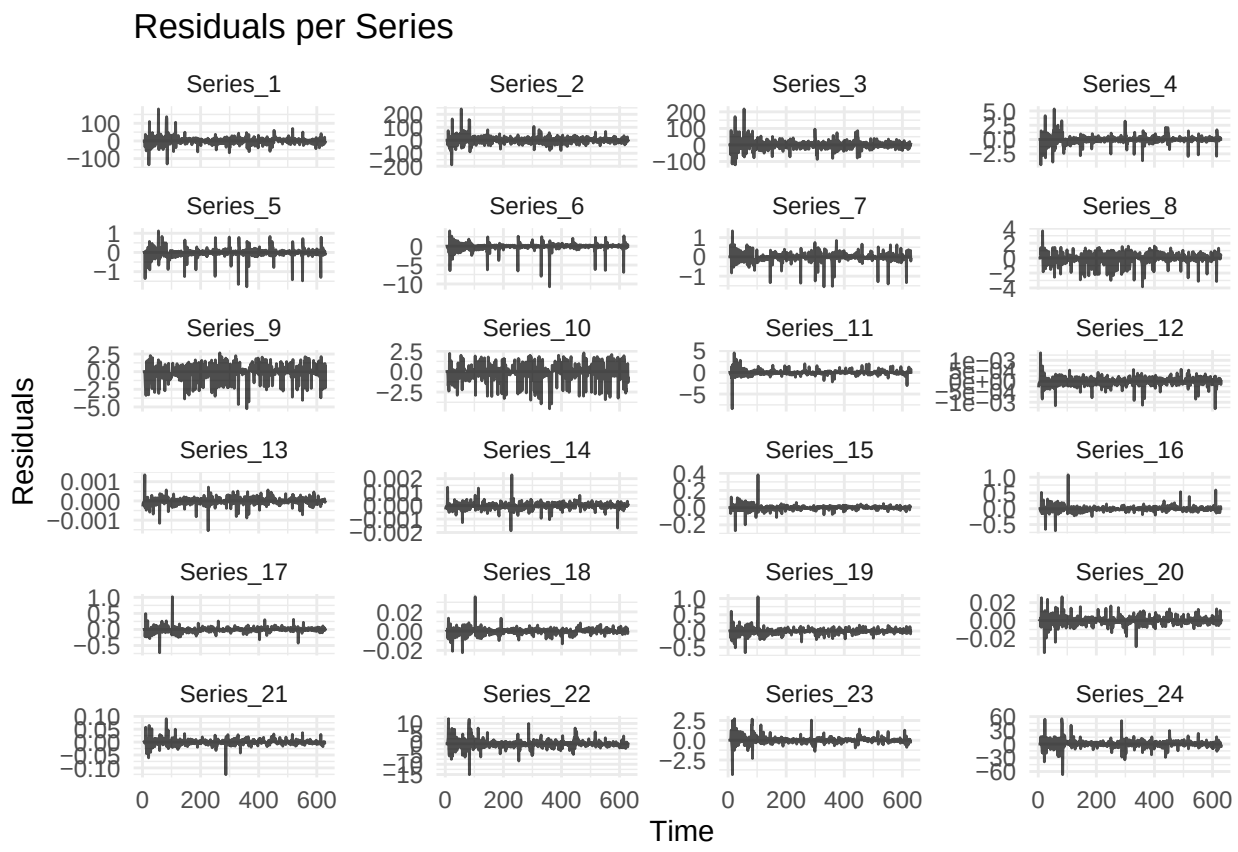
```
##  [1] 0.001680502 0.001550665 0.001524294 0.002454696 0.005613597 0.006851430
##  [7] 0.020091765 0.048671394 0.034670593 0.037573777 0.015468952 0.005245128
## [13] 0.003823701 0.004678223 0.004724275 0.005141735 0.003905711 0.003688611
## [19] 0.003834980 0.005294535 0.003734201 0.003028961 0.003593437 0.002419475
```

```
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.009552693
```

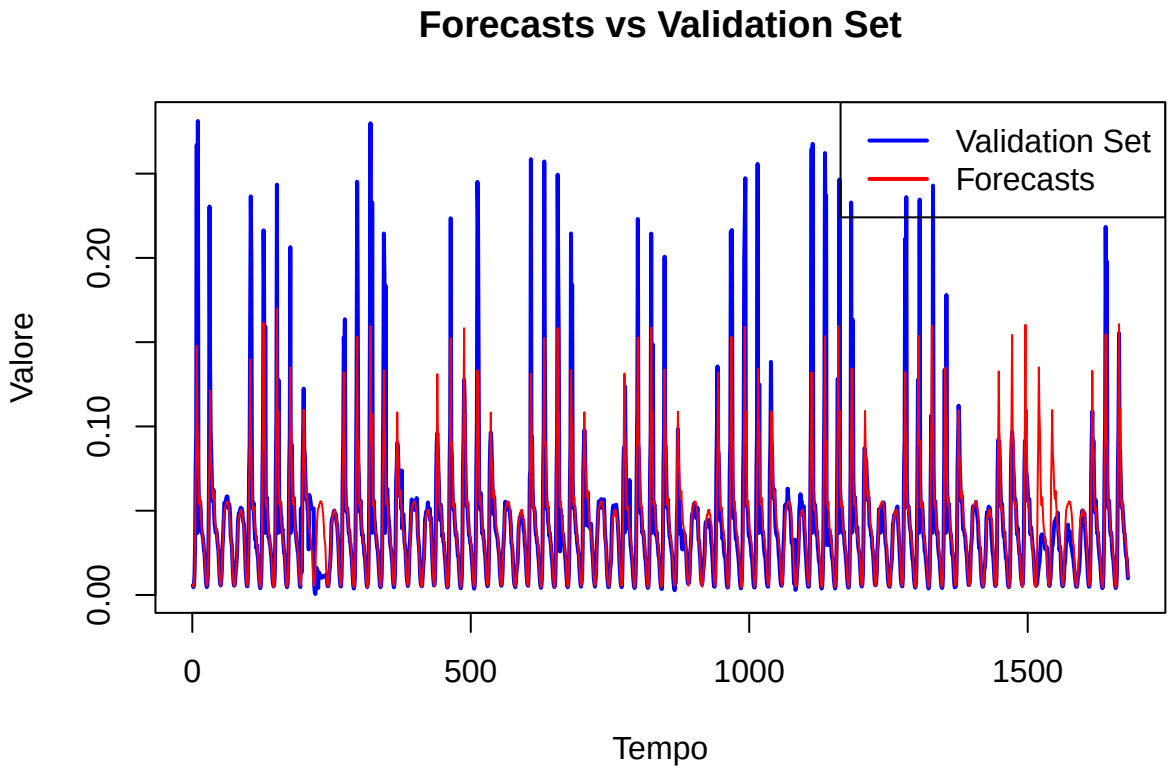
```
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
```

```
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecasts_list)
series_length <- length(forecasts_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecasts_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
```

```r
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)

# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



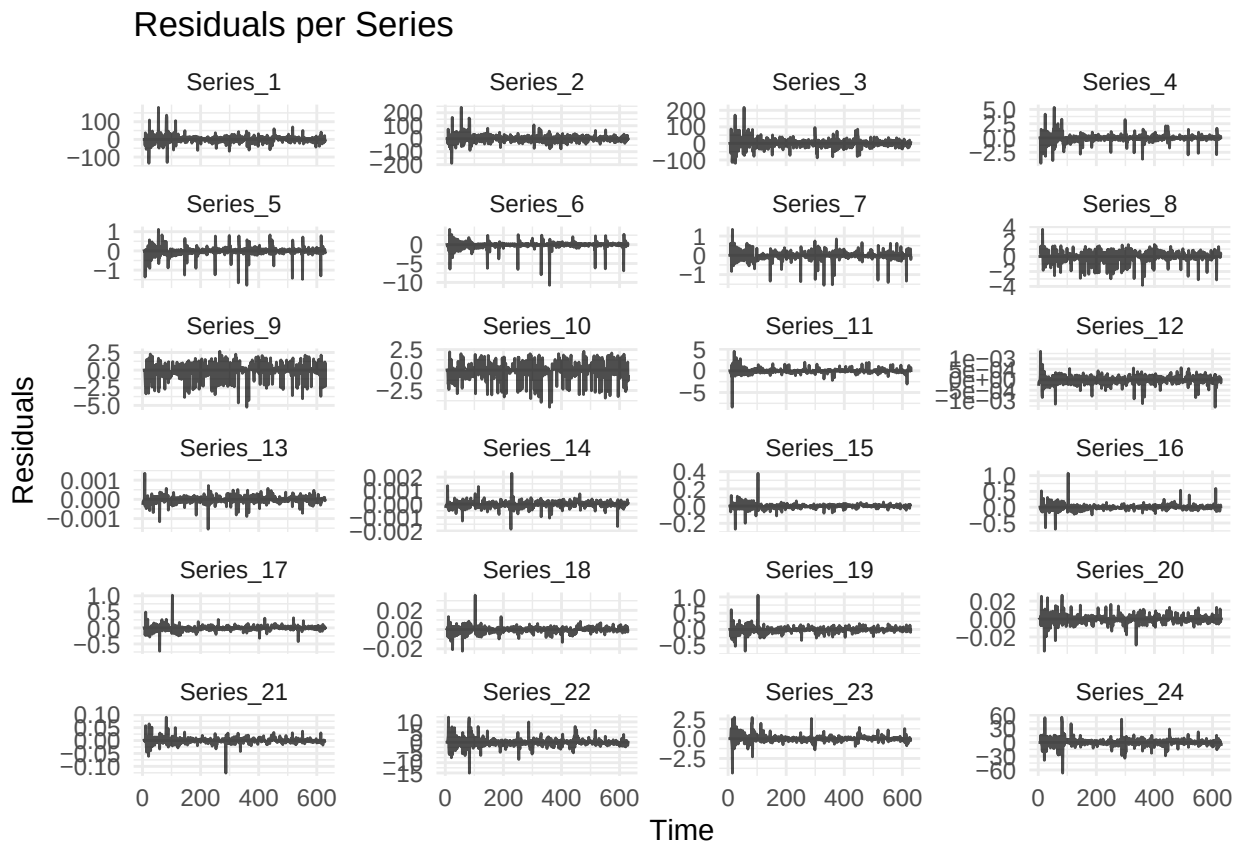
Dummies

Arima (3,1,1) (0,1,1)(7) with dummies

```r
# Liste per memorizzare i MAE e i residui
mae_values <- numeric(length = 24)
residuals_list <- list()

xreg_columns <- c("Dec24", "Dec25", "Dec26", "Jan1", "Jan6",
                  "EasterSat", "Easter", "EasterMon",
                  "EasterTue", "Aug15", "EndYear", "Valentine")


# Loop per ogni serie temporale
for (i in 1:24) {
```

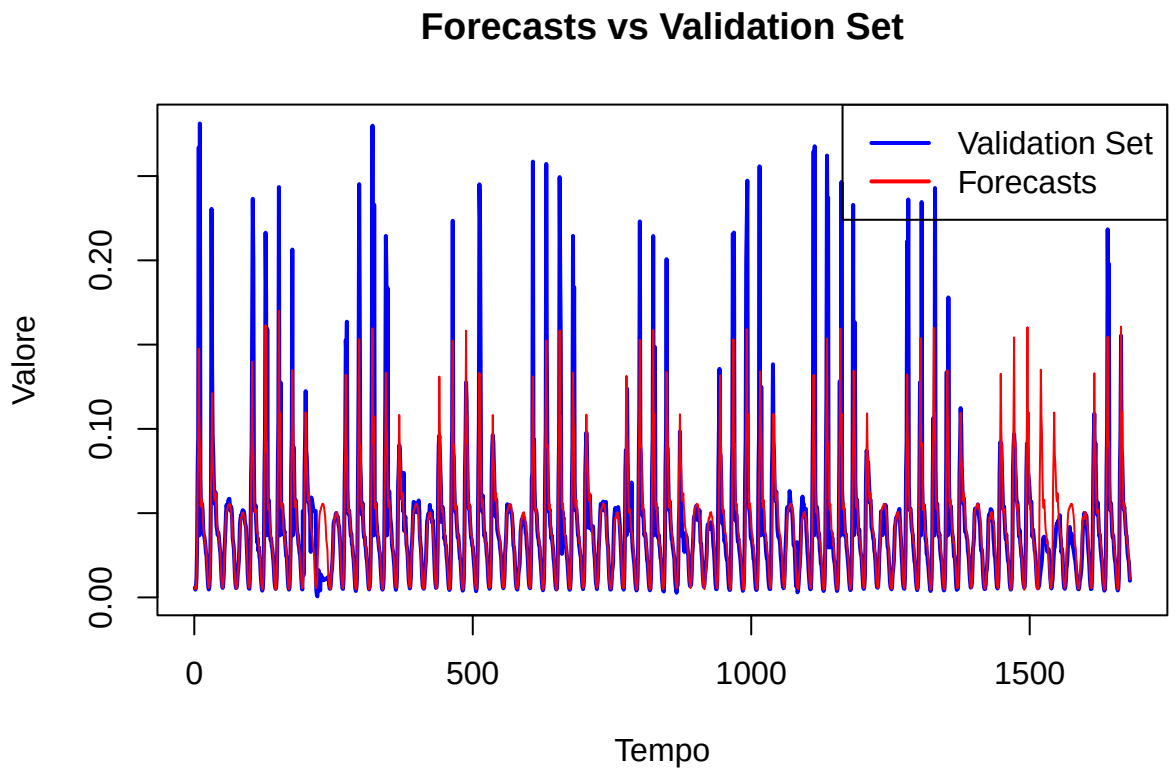
```r
  # Estrai i dati di allenamento e test per la serie corrente
  train_series <- train_val_series_dummy[[i]]$train
  test_series <- train_val_series_dummy[[i]]$test

  # Ottieni le variabili dummy (xreg) per training e test set
  train_xreg <- as.matrix(train_series[, xreg_columns, drop = FALSE])
  test_xreg <- as.matrix(test_series[, xreg_columns, drop = FALSE])

  # Converti le variabili in numerico
  train_xreg <- apply(train_xreg, 2, as.numeric)
  test_xreg <- apply(test_xreg, 2, as.numeric)

  # Specifica i parametri ARIMA manualmente
  p <- 3    # Ordine autoregressivo
  d <- 1    # Ordine di differenziazione
  q <- 1    # Ordine media mobile

  # Parametri stagionali
  seasonal_p <- 0
  seasonal_d <- 1
  seasonal_q <- 1
  m <- 7   # Periodo stagionale

  # Costruisci il modello ARIMA manualmente con regressori esterni
  model <- Arima(train_series$X,
                 order = c(p, d, q),
                 seasonal = list(order = c(seasonal_p, seasonal_d, seasonal_q), period = m),
                 lambda = optimal_lambdas[i],
                 xreg = train_xreg)

  # Fai una previsione sul test set
  forecast_values <- forecast(model, h = length(test_series$X), xreg = test_xreg)

  #Salva previsioni
  forecasts_list[[i]] <- forecast_values$mean

  # Calcola l'errore assoluto medio (MAE)
  mae_values[i] <- mean(abs(forecast_values$mean - test_series$X), na.rm = TRUE)

  # Salva i residui del modello
  residuals_list[[i]] <- residuals(model)
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
print(mae_values)
cat("MAE: ", mae, "\n")
```

```r
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecasts_list)
series_length <- length(forecasts_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
```

```r
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecasts_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)

# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



Arima (3,1,1) (0,1,2)(7) with dummies

```
# Liste per memorizzare i MAE e i residui
mae_values <- numeric(length = 24)
residuals_list <- list()

xreg_columns <- c("Dec24", "Dec25", "Dec26", "Jan1", "Jan6",
                  "EasterSat", "Easter", "EasterMon",
                  "EasterTue", "Aug15", "EndYear", "Valentine")

# Loop per ogni serie temporale
for (i in 1:24) {
  # Estrai i dati di allenamento e test per la serie corrente
  train_series <- train_val_series_dummy[[i]]$train
  test_series <- train_val_series_dummy[[i]]$test

  # Ottieni le variabili dummy (xreg) per training e test set
  train_xreg <- as.matrix(train_series[, xreg_columns, drop = FALSE])
  test_xreg <- as.matrix(test_series[, xreg_columns, drop = FALSE])

  # Converti le variabili in numerico
  train_xreg <- apply(train_xreg, 2, as.numeric)
  test_xreg <- apply(test_xreg, 2, as.numeric)

  # Specifica i parametri ARIMA manualmente
  p <- 3    # Ordine autoregressivo
  d <- 1    # Ordine di differenziazione
  q <- 1    # Ordine media mobile

  # Parametri stagionali
  seasonal_p <- 0
  seasonal_d <- 1
  seasonal_q <- 2
  m <- 7   # Periodo stagionale

  # Costruisci il modello ARIMA manualmente con regressori esterni
  model <- Arima(train_series$X,
                 order = c(p, d, q),
                 seasonal = list(order = c(seasonal_p, seasonal_d, seasonal_q), period = m),
                 lambda = optimal_lambdas[i],
                 xreg = train_xreg)

  # Fai una previsione sul test set
  forecast_values <- forecast(model, h = length(test_series$X), xreg = test_xreg)

  # Salva previsioni
  forecasts_list[[i]] <- forecast_values$mean

  # Calcola l'errore assoluto medio (MAE)
  mae_values[i] <- mean(abs(forecast_values$mean - test_series$X), na.rm = TRUE)

  # Salva i residui del modello
  residuals_list[[i]] <- residuals(model)
}
```

```r
# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

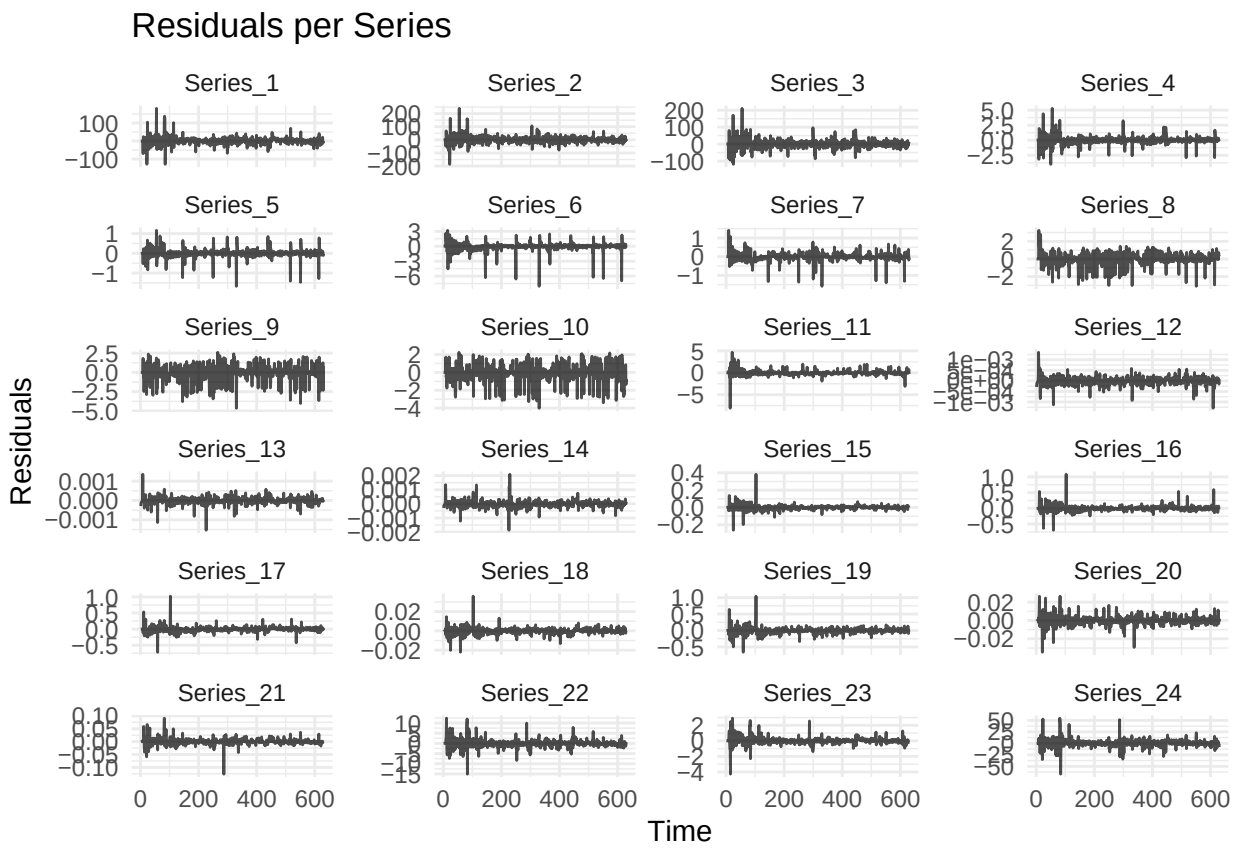
```r
print(mae_values)
```

```
##  [1] 0.001705995 0.001535358 0.001538645 0.002446706 0.005724375 0.006629295
##  [7] 0.020292216 0.048774209 0.034613006 0.037724147 0.015548211 0.005088516
## [13] 0.003851504 0.004683768 0.004752285 0.005141593 0.003899059 0.003668267
## [19] 0.003836504 0.005355312 0.003678123 0.003063932 0.003599061 0.002344511
```

```r
cat("MAE: ", mae, "\n")
```

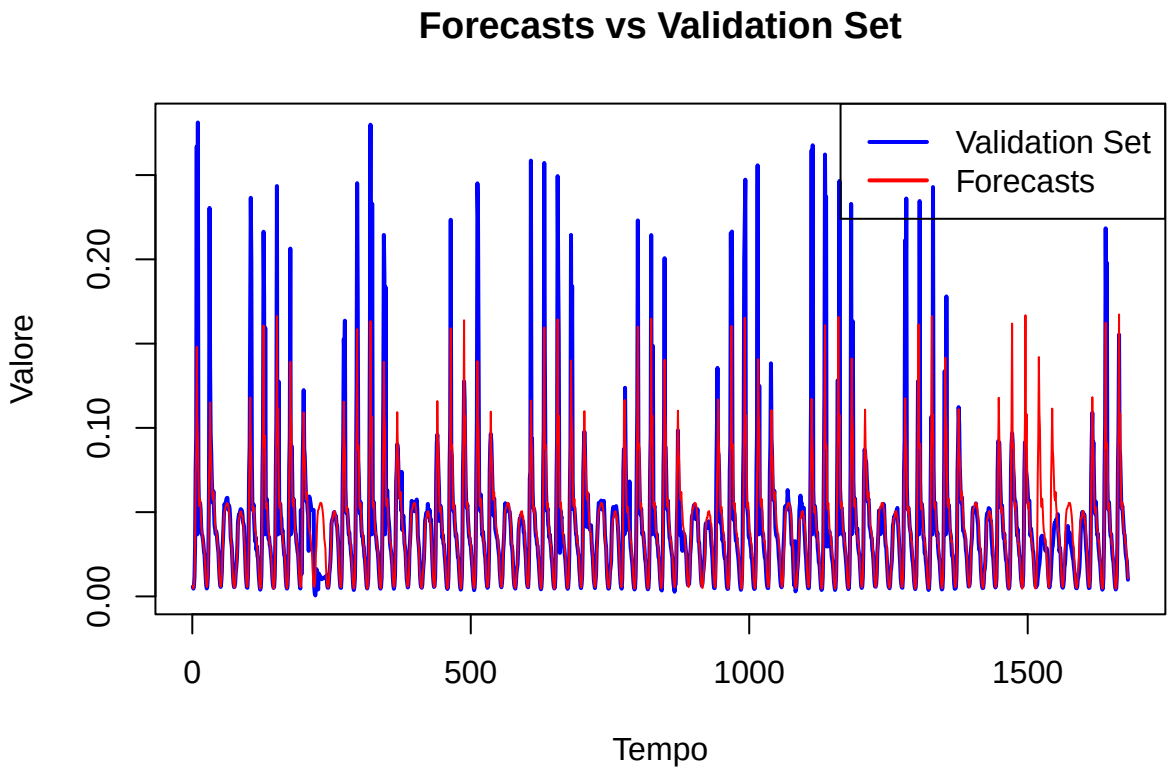
```
## MAE:  0.009562275
```

```r
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecasts_list)
series_length <- length(forecasts_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecasts_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



Arima (3,1,1) (1,1,1)(7) with dummies

```r
# Liste per memorizzare i MAE e i residui
mae_values <- numeric(length = 24)
residuals_list <- list()

xreg_columns <- c("Dec24", "Dec25", "Dec26", "Jan1", "Jan6",
                  "EasterSat", "Easter", "EasterMon",
                  "EasterTue", "Aug15", "EndYear", "Valentine")

# Loop per ogni serie temporale
for (i in 1:24) {
  # Estrai i dati di allenamento e test per la serie corrente
  train_series <- train_val_series_dummy[[i]]$train
  test_series <- train_val_series_dummy[[i]]$test

  # Ottieni le variabili dummy (xreg) per training e test set
  train_xreg <- as.matrix(train_series[, xreg_columns, drop = FALSE])
```

```r
  test_xreg <- as.matrix(test_series[, xreg_columns, drop = FALSE])

  # Converti le variabili in numerico
  train_xreg <- apply(train_xreg, 2, as.numeric)
  test_xreg <- apply(test_xreg, 2, as.numeric)

  # Specifica i parametri ARIMA manualmente
  p <- 3    # Ordine autoregressivo
  d <- 1    # Ordine di differenziazione
  q <- 1    # Ordine media mobile

  # Parametri stagionali
  seasonal_p <- 1
  seasonal_d <- 1
  seasonal_q <- 1
  m <- 7   # Periodo stagionale

  # Costruisci il modello ARIMA manualmente con regressori esterni
  model <- Arima(train_series$X,
                 order = c(p, d, q),
                 seasonal = list(order = c(seasonal_p, seasonal_d, seasonal_q), period = m),
                 lambda = optimal_lambdas[i],
                 xreg = train_xreg)

  # Fai una previsione sul test set
  forecast_values <- forecast(model, h = length(test_series$X), xreg = test_xreg)

  # Salva previsioni
  forecasts_list[[i]] <- forecast_values$mean

  # Calcola l'errore assoluto medio (MAE)
  mae_values[i] <- mean(abs(forecast_values$mean - test_series$X), na.rm = TRUE)

  # Salva i residui del modello
  residuals_list[[i]] <- residuals(model)
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

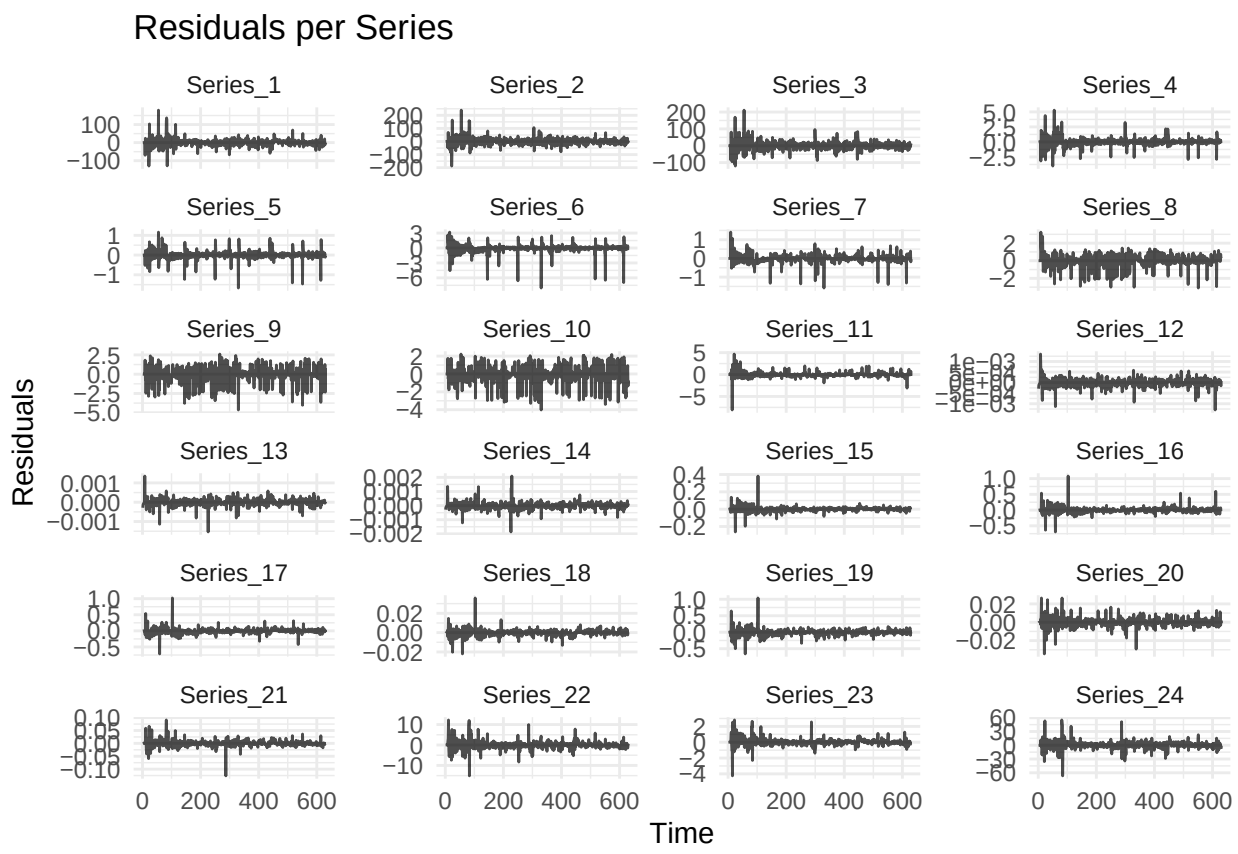
```r
print(mae_values)
```

```
##  [1] 0.001703763 0.001540520 0.001535130 0.002443403 0.005724452 0.006623899
##  [7] 0.020280809 0.048765807 0.034602049 0.037659907 0.015546976 0.005102231
## [13] 0.003846638 0.004683174 0.004759972 0.005138145 0.003914038 0.003667690
## [19] 0.003827644 0.005354637 0.003675519 0.002973251 0.003559955 0.002372195
```

```r
cat("MAE: ", mae, "\n")
```

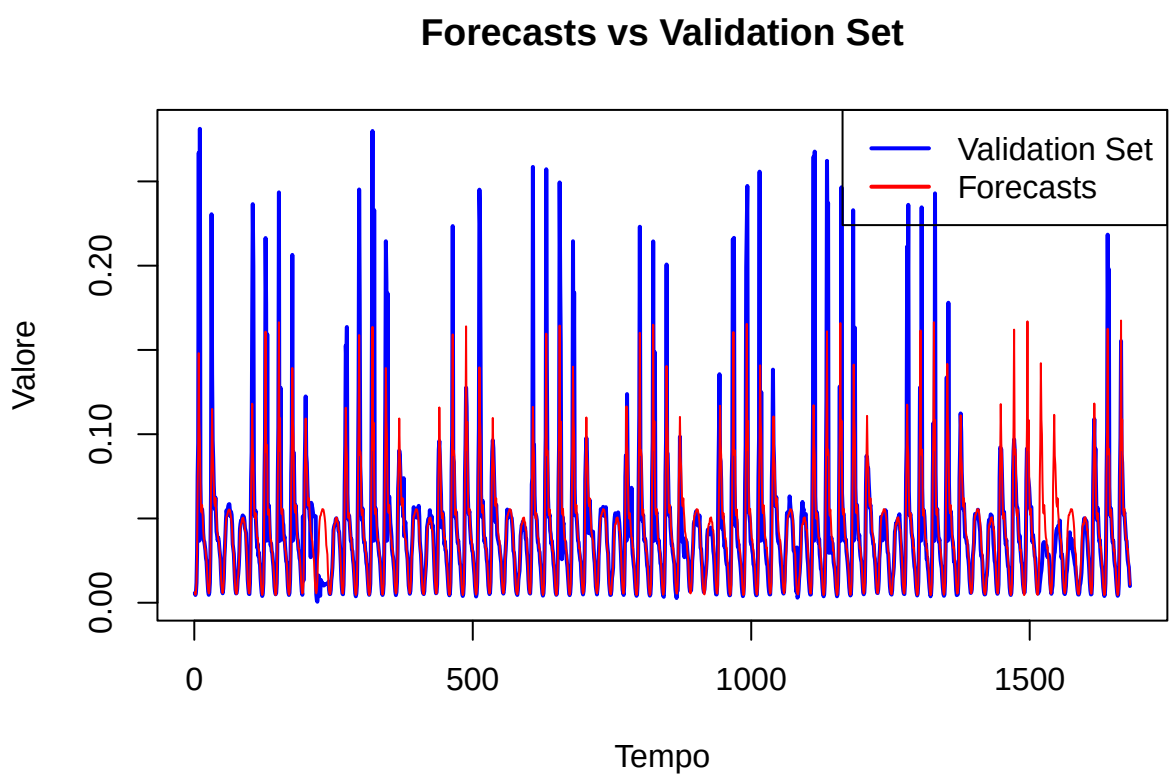
```
## MAE:  0.009554242
```

```r
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecasts_list)
series_length <- length(forecasts_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)
```

```r
# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecasts_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)

# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



Deployment final model

```r
# Inizializza una lista per salvare le previsioni per ciascuna serie temporale
forecast_results_ARIMA <- vector("list", 24)

# Colonne dei regressori (dummy)
xreg_columns <- c("Dec24", "Dec25", "Dec26", "Jan1", "Jan6",
                  "EasterSat", "Easter", "EasterMon",
                  "EasterTue", "Aug15", "EndYear", "Valentine")

# Loop su tutte le serie temporali
for (i in 1:24) {
  print(i)
  # Estrai train e test per la serie corrente
  train_series <- train_test_series_dummy[[i]]$train
  test_series <- train_test_series_dummy[[i]]$test

  # Ottieni il valore di lambda per questa serie
  lambda <- optimal_lambdas[i]

  # Estrai i regressori (dummy) per training e test set
  train_xreg <- as.matrix(train_series[, xreg_columns, drop = FALSE])
  test_xreg <- as.matrix(test_series[, xreg_columns, drop = FALSE])

  # Converti le variabili in numerico
  train_xreg <- apply(train_xreg, 2, as.numeric)
  test_xreg <- apply(test_xreg, 2, as.numeric)

  # Specifica i parametri ARIMA
  p <- 3    # Ordine autoregressivo
  d <- 1    # Ordine di differenziazione
  q <- 1    # Ordine media mobile

  # Parametri stagionali
  seasonal_p <- 0
  seasonal_d <- 1
  seasonal_q <- 1
  m <- 7  # Periodo settimanale

  # Costruisci il modello ARIMA con regressori esterni
  model <- tryCatch({
    Arima(train_series$X,
          order = c(p, d, q),
          seasonal = list(order = c(seasonal_p, seasonal_d, seasonal_q), period = m),
          lambda = lambda,
          xreg = train_xreg)
  }, error = function(e) {
    cat(sprintf("Errore nella serie %d: %s\n", i, e$message))
    NULL
  })

  # Verifica se il modello è stato creato con successo
  if (!is.null(model)) {
    # Fai una previsione sul test set con regressori esterni
```

```
    forecast_values <- tryCatch({
      forecast(model, h = length(test_series$DateTime), xreg = test_xreg)
    }, error = function(e) {
      cat(sprintf("Errore nella previsione per la serie %d: %s\n", 1, e$message))
      NULL
    })

    # Salva le previsioni nella lista
    if (!is.null(forecast_values)) {
      forecast_results_ARIMA[[i]] <- as.numeric(forecast_values$mean)
    } else {
      forecast_results_ARIMA[[i]] <- NULL
    }
  } else {
    # Se il modello non è stato creato, salva NULL
    forecast_results_ARIMA[[i]] <- NULL
  }
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
```

```
# Stampa un riepilogo delle previsioni
cat("Previsioni completate per le 24 serie temporali.\n")
```

```
## Previsioni completate per le 24 serie temporali.
```

```
forecast_results_ARIMA[[1]]
```

```
##   [1] 0.005836121 0.006454987 0.009333407 0.010276006 0.007572757 0.005774914
##   [7] 0.005943068 0.005890924 0.006461411 0.009334022 0.010264356 0.007561643
```

```
## [13] 0.005767372 0.005934583 0.005882340 0.006450946 0.009312067 0.010237738
## [19] 0.007547167 0.005758941 0.005925655 0.005873567 0.006440396 0.017352926
## [25] 0.030253252 0.007078260 0.005750531 0.005916751 0.005864819 0.006429880
## [31] 0.012396532
```

```
forecast_results_ARIMA[[2]]
```

```
##  [1] 0.004403759 0.004952013 0.006361674 0.006787382 0.005307715 0.004311449
##  [7] 0.004329202 0.004462305 0.004968931 0.006373835 0.006779151 0.005296466
## [13] 0.004303003 0.004320172 0.004452405 0.004956522 0.006353349 0.006755934
## [19] 0.005282270 0.004293625 0.004310719 0.004442363 0.004944081 0.011756614
## [25] 0.016310486 0.004498652 0.004284286 0.004301305 0.004432367 0.004931702
## [31] 0.008209508
```

```
forecast_results_ARIMA[[3]]
```

```
##  [1] 0.005211713 0.005946321 0.005280578 0.004845232 0.005264961 0.005403324
##  [7] 0.005446346 0.005496746 0.006275046 0.005487649 0.004975781 0.005387978
## [13] 0.005505674 0.005527427 0.005561624 0.006341400 0.005527195 0.005001085
## [19] 0.005410988 0.005524182 0.005541680 0.005572524 0.006351937 0.006047516
## [25] 0.004928811 0.003832444 0.005525797 0.005542437 0.005572594 0.006351310
## [31] 0.005524636
```

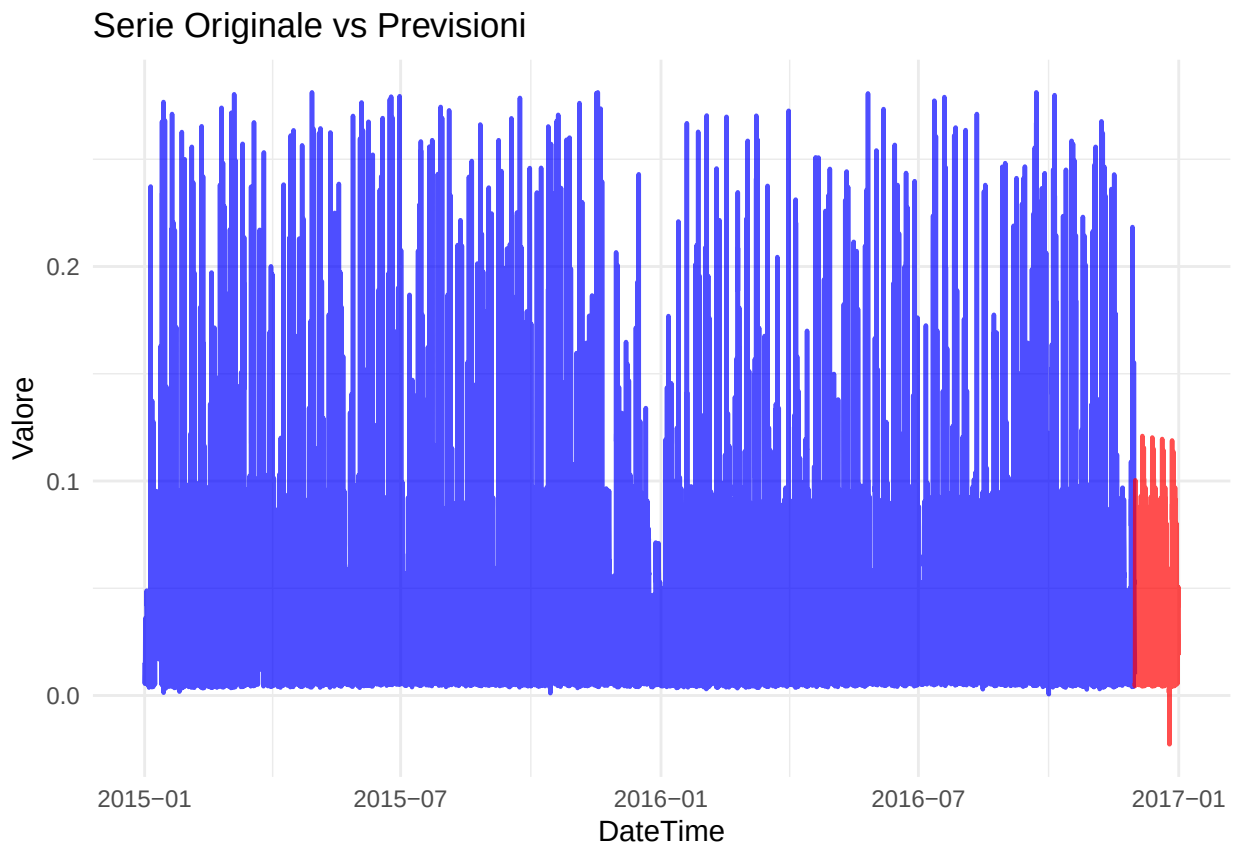
```r
# Creazione della base DateTime
start_date <- as.POSIXct("2016-12-01 00:00:00")
num_days <- length(forecast_results_ARIMA[[1]])  # Numero di giorni previsti
hourly_series_length <- num_days * 24  # Numero totale di ore

# Genera il vettore DateTime per ogni ora
datetime_vector <- seq(start_date, by = "hour", length.out = hourly_series_length)

# Inizializza un vettore per salvare tutte le previsioni
forecast_vector_ARIMA <- numeric(hourly_series_length)

# Ricomponi la serie oraria
for (i in 1:24) {
  # Inserisci le previsioni nella posizione corretta
  forecast_vector_ARIMA[seq(i, hourly_series_length, by = 24)] <- forecast_results_ARIMA[[i]]
}

# Creazione del data frame finale
forecast_dataframe <- data.frame(
  DateTime = datetime_vector,
  ARIMA = forecast_vector_ARIMA
)

# Visualizza le prime righe del risultato
head(forecast_dataframe)
```

```
##              DateTime       ARIMA
## 1 2016-12-01 00:00:00 0.005836121
## 2 2016-12-01 01:00:00 0.004403759
## 3 2016-12-01 02:00:00 0.005211713
## 4 2016-12-01 03:00:00 0.011759774
## 5 2016-12-01 04:00:00 0.030413454
## 6 2016-12-01 05:00:00 0.068222103
```

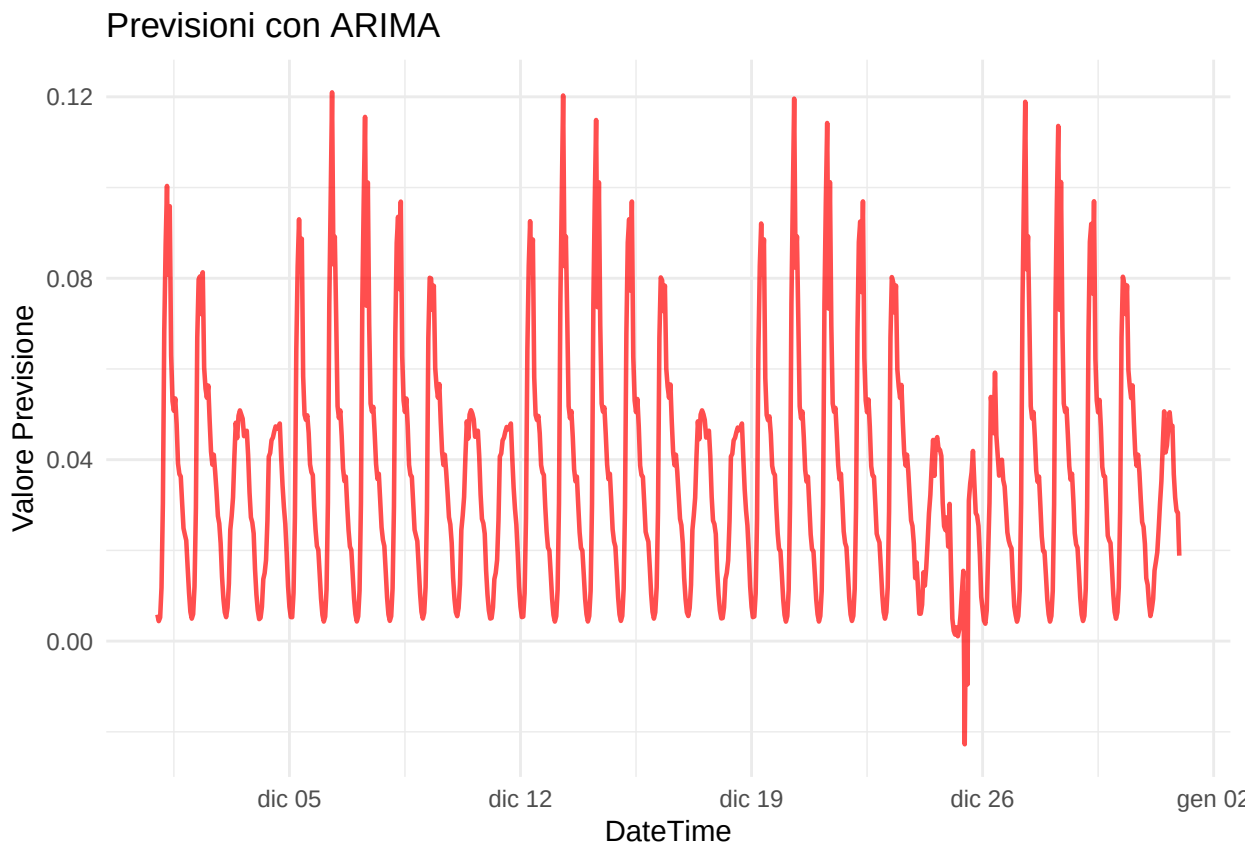
```r
dim(forecast_dataframe)
```

```
## [1] 744   2
```

```r
library(ggplot2)

# Filtra la serie originale per sovrapporre solo fino all'ultima osservazione reale
ts_train_df <- subset(ts_cutted_filled_df, Date < as.POSIXct("2016-12-01 00:00:00"))

# Combina i dati di training e le previsioni
forecast_df <- data.frame(
  DateTime = forecast_dataframe$DateTime,
  Forecast = forecast_dataframe$ARIMA
)

# Plot con ggplot2
ggplot() +
  # Serie originale (parte di training)
  geom_line(data = ts_train_df, aes(x = Date, y = Value), color = "blue", size = 0.8, alpha = 0.7) +
  # Previsioni
  geom_line(data = forecast_df, aes(x = DateTime, y = Forecast), color = "red", size = 0.8, alpha = 0.7
  # Personalizzazione
  labs(title = "Serie Originale vs Previsioni",
       x = "DateTime", y = "Valore") +
  theme_minimal()
```




```r
library(ggplot2)

# Combina i dati delle previsioni
forecast_df <- data.frame(
  DateTime = forecast_dataframe$DateTime,
  Forecast = forecast_dataframe$ARIMA
)

# Plot con ggplot2: solo previsioni
ggplot(data = forecast_df) +
  # Previsioni
  geom_line(aes(x = DateTime, y = Forecast), color = "red", size = 0.8, alpha = 0.7) +
  # Personalizzazione
  labs(title = "Previsioni con ARIMA",
       x = "DateTime", y = "Valore Previsione") +
  theme_minimal()
```



Unobserved Components Models (UCM)

```
library(xts)
library(ggplot2)
library(lubridate)
library(forecast)
#install.packages("fastDummies")
library(fastDummies)
library(Metrics)
library("KFAS")
library(KFAS)
hour <- 17
```

Univariate

LLT, seasonal dummy (7)

```
# Modello a: LLT e stagionalità dummy a periodo di 7 giorni

# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()

test_values_list <- list()  # Lista per memorizzare i valori di training
```

```r
forecast_values_list <- list()  # Lista per memorizzare i valori di forecast

# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test

  # Ottieni il valore di lambda per questa serie ed applicala (da optimal_lambdas)
  lambda <- optimal_lambdas[i]
  train_series_boxcox <- BoxCox(train_series$X, lambda)

  # Costruisci il modello UCM: trend locale lineare (LLT) + stagionalità dummy con periodo 7
  model <- SSModel(train_series_boxcox ~
                     SSMtrend(2, list(NA, NA)) +    # LLTrend di ordine 2 con varianze iniziali sconosc
                     SSMseasonal(7, NA, "dummy"),   # Stagionalità settimanale (periodo = 7)
                   H = NA)                          # Varianza del rumore di misura

  # Inizializza i valori basati sulla varianza della serie di training
  ts_var <- var(train_series_boxcox, na.rm = TRUE)
  inits <- log(c(
    ts_var / 100,    # Varianza del trend lento
    ts_var / 1000,   # Varianza del trend veloce
    ts_var / 10,     # Varianza della stagionalità giornaliera
    ts_var / 1000    # Varianza degli errori di misura
  ))

  # Adattamento del modello
  fit_mod <- fitSSM(model, inits = inits, method = "BFGS")

  # Esecuzione del filtro di Kalman per stimare lo stato latente
  kfs <- KFS(fit_mod$model)

  # Estrai le previsioni sulla scala Box-Cox
  pred_boxcox <- predict(fit_mod$model, n.ahead = length(test_series$X), interval = "none")

  # Converti le previsioni sulla scala originale usando InvBoxCox
  pred_original <- InvBoxCox(pred_boxcox, lambda)

  # Salva i valori di training e previsione nella lista
  test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
  forecast_values_list[[i]] <- pred_original  # Salva le previsioni

  # Calcola il MAE confrontando le previsioni con i valori reali
  mae_values[i] <- mean(abs(pred_original - test_series$X))

  # Salva i residui (scala originale) nella lista
  residuals_list[[i]] <- test_series$X - pred_original
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)
```

```r
# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```r
print(mae_values)
```

```
##  [1] 0.001623562 0.001520820 0.001627661 0.002310122 0.005283809 0.008530825
##  [7] 0.020439896 0.051080485 0.034803684 0.037299717 0.015798539 0.005172245
## [13] 0.004037080 0.004908775 0.004396895 0.004791180 0.003835159 0.004037759
## [19] 0.003906006 0.005088857 0.004463196 0.003475504 0.003117420 0.002185404
```

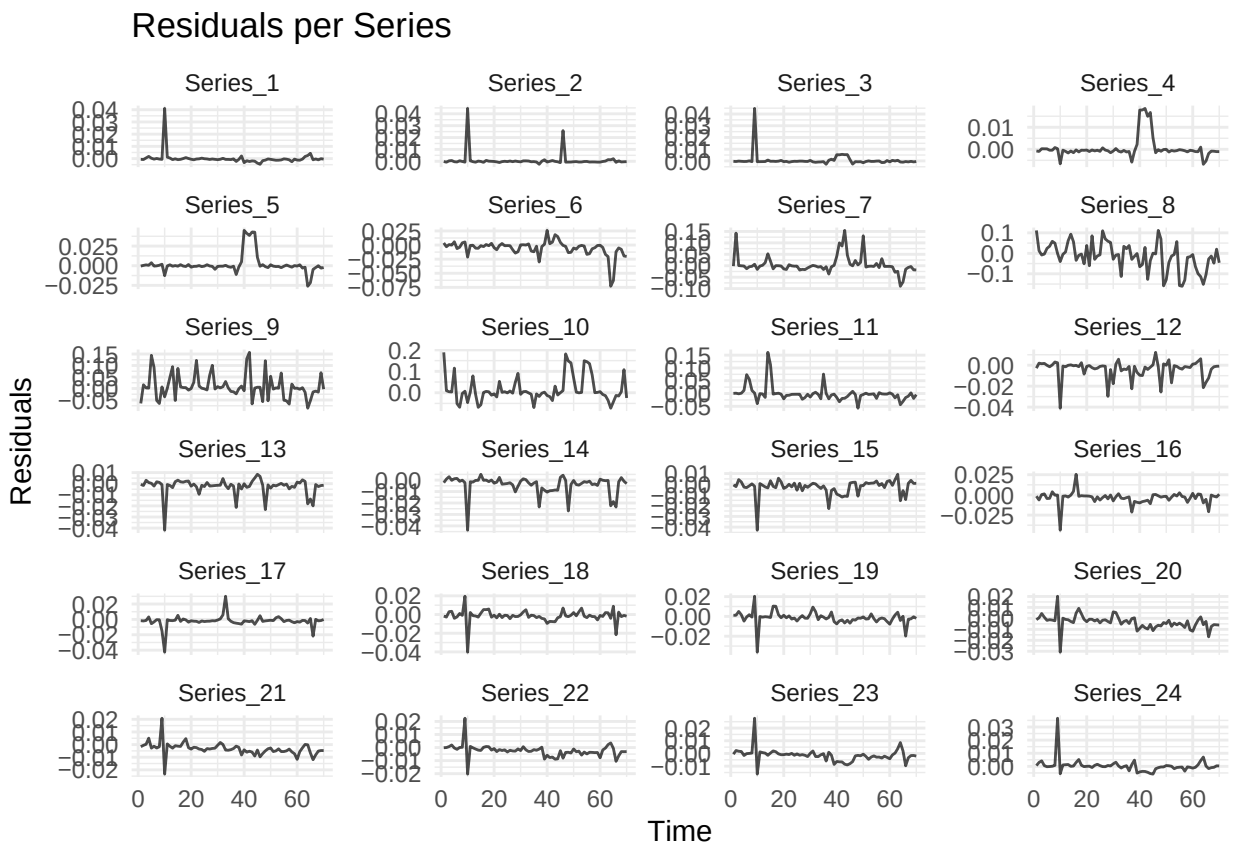
```r
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.009738942
```

```r
print(fit_mod$optim.out$convergence==0)
```

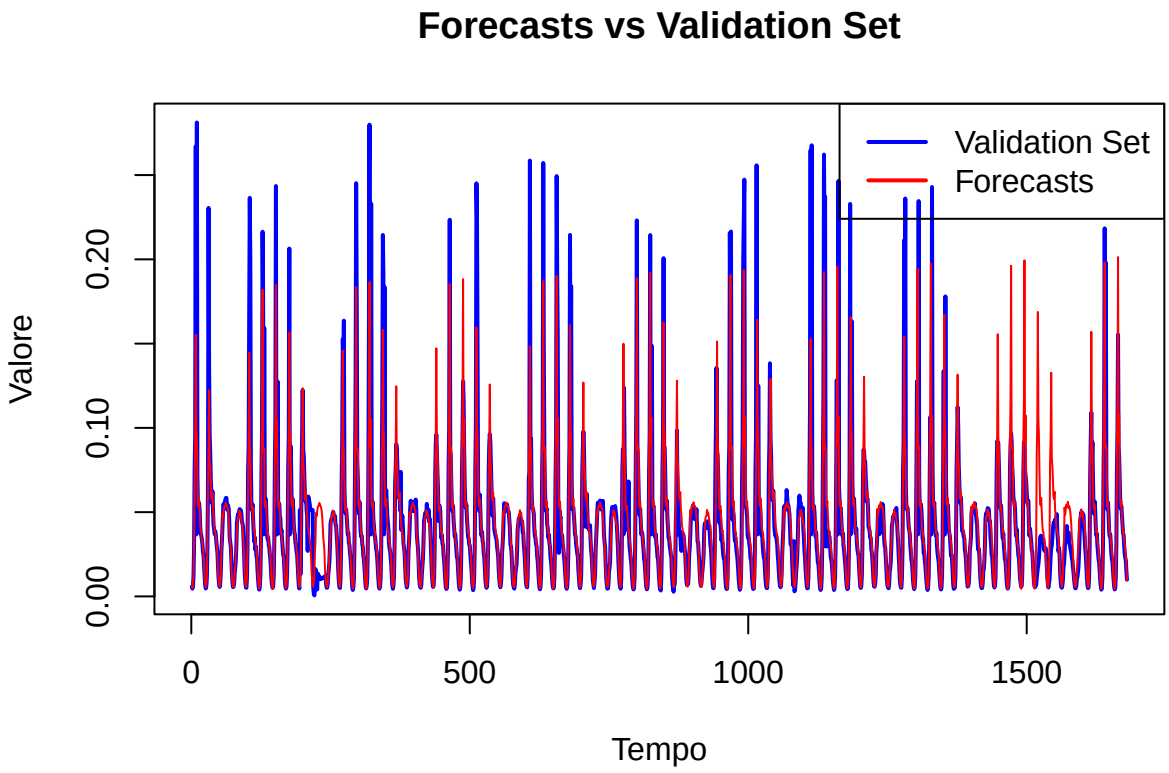
```
## [1] TRUE
```

```r
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



LLT, trigometric seasonality (7)

```r
# Modello b: LLT e stagionalità trigonometrica di periodo 7

# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()

test_values_list <- list()  # Lista per memorizzare i valori di training
forecast_values_list <- list()  # Lista per memorizzare i valori di forecast

# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test

  # Ottieni il valore di lambda per questa serie ed applicala (da optimal_lambdas)
```

```r
  lambda <- optimal_lambdas[i]
  train_series_boxcox <- BoxCox(train_series$X, lambda)

  # Costruisci il modello UCM: trend locale lineare (LLT) + stagionalità trigonometrica con periodo 7
  model <- SSModel(train_series_boxcox ~
                     SSMtrend(2, list(NA, NA)) +    # LLTrend di ordine 2 con varianze iniziali sconosc
                     SSMseasonal(7, NA, "trig"),  # Stagionalità trigonometrica settimanale (periodo =
                   H = NA)                         # Varianza del rumore di misura

  # Inizializza i valori basati sulla varianza della serie di training
  ts_var <- var(train_series_boxcox, na.rm = TRUE)
  inits <- log(c(
    ts_var / 100,    # Varianza del trend lento
    ts_var / 1000,   # Varianza del trend veloce
    ts_var / 10,     # Varianza della stagionalità giornaliera
    ts_var / 1000    # Varianza degli errori di misura
  ))

  updt <- function(pars, model) {
  model$Q[1, 1, 1] <- exp(pars[1])
  model$Q[2, 2, 1] <- exp(pars[2])
  diag(model$Q[-(1:2), -(1:2), 1]) <- exp(pars[3])
  model$H[1, 1, 1] <- exp(pars[4])
  model
  }

  # Adattamento del modello
  fit_mod <- fitSSM(model, inits = inits, updatefn = updt, method = "BFGS")

  # Esecuzione del filtro di Kalman per stimare lo stato latente
  kfs <- KFS(fit_mod$model)

  # Estrai le previsioni sulla scala Box-Cox
  pred_boxcox <- predict(fit_mod$model, n.ahead = length(test_series$X), interval = "none")

  # Converti le previsioni sulla scala originale usando InvBoxCox
  pred_original <- InvBoxCox(pred_boxcox, lambda)

  # Salva i valori di training e previsione nella lista
  test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
  forecast_values_list[[i]] <- pred_original  # Salva le previsioni

  # Calcola il MAE confrontando le previsioni con i valori reali
  mae_values[i] <- mean(abs(pred_original - test_series$X))

  # Salva i residui (scala originale) nella lista
  residuals_list[[i]] <- test_series$X - pred_original
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
```

```r
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```r
print(mae_values)
```

```
##  [1] 0.001596975 0.001520495 0.001624532 0.002312310 0.005295285 0.010996851
##  [7] 0.021502018 0.049775418 0.034802404 0.037289017 0.014582394 0.004941779
## [13] 0.003958812 0.005054565 0.004345983 0.005072928 0.004045885 0.004037042
## [19] 0.003884163 0.005034091 0.004466592 0.003475705 0.003142985 0.002265281
```

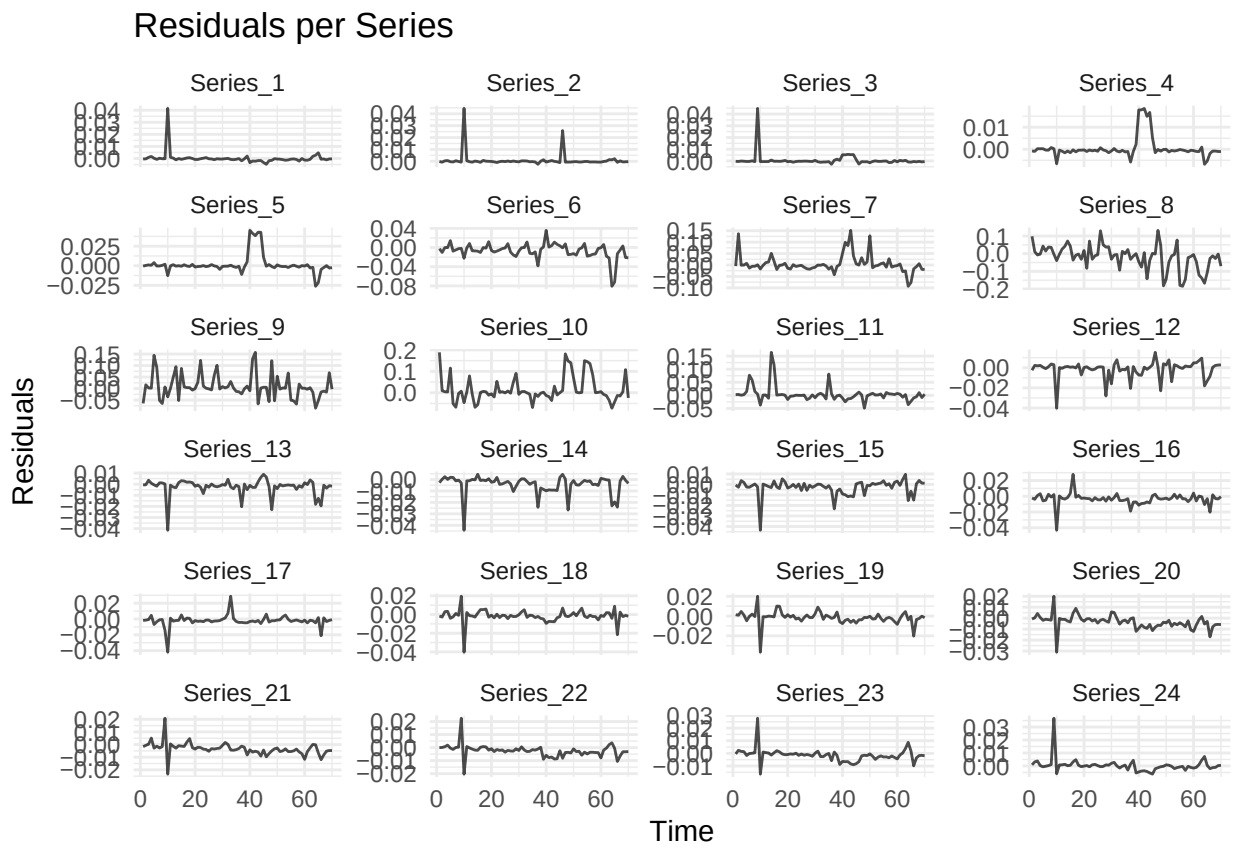
```r
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.009792646
```

```r
print(fit_mod$optim.out$convergence==0)
```

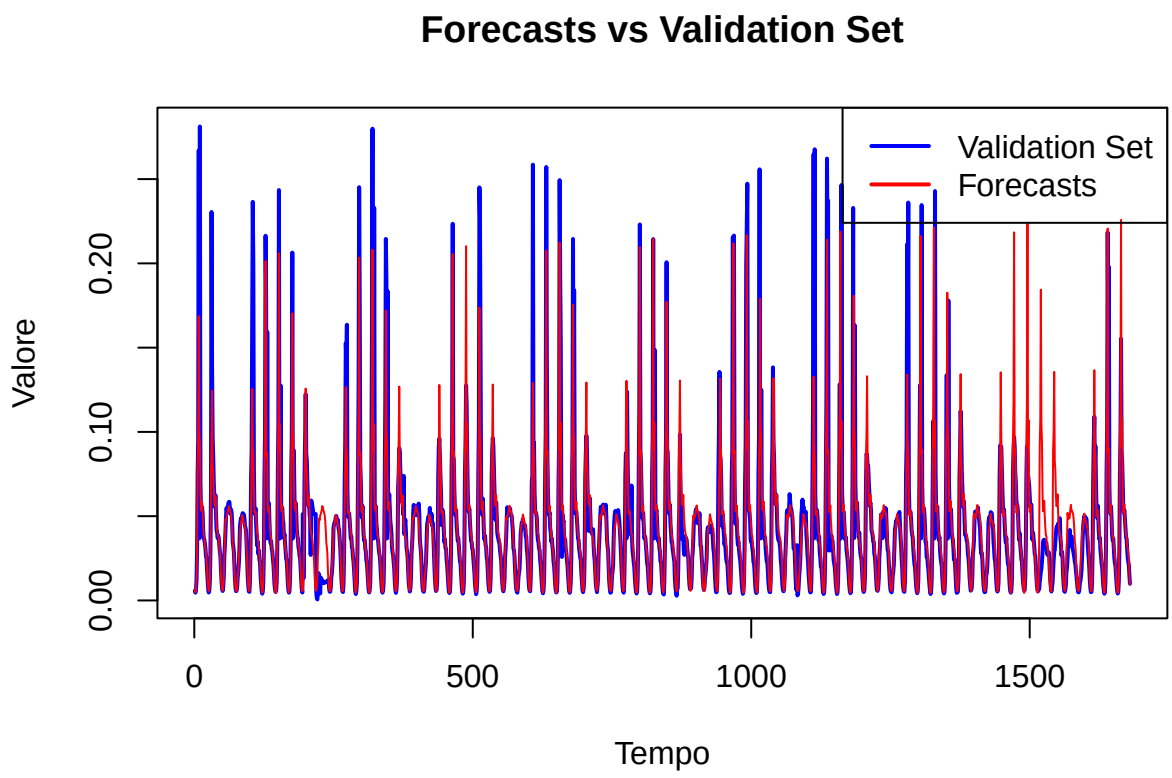
```
## [1] TRUE
```

```r
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)

# Grafico delle serie temporali
```

```r
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



LLT, seasonal dummy (7) and one trigonometric harmonic (365)

```r
# Modello c: LLT e seasonal dummy (7) and 1 trigonometric harmonic (365)

# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()

test_values_list <- list()  # Lista per memorizzare i valori di training
forecast_values_list <- list()  # Lista per memorizzare i valori di forecast

# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test

  # Ottieni il valore di lambda per questa serie ed applicala (da optimal_lambdas)
```

```r
  lambda <- optimal_lambdas[i]
  train_series_boxcox <- BoxCox(train_series$X, lambda)

  # Costruisci il modello UCM: trend locale lineare (LLT) + stagionalità trigonometrica con periodo 7
  model <- SSModel(train_series_boxcox ~
                     SSMtrend(2, list(NA, NA)) +    # LLTrend di ordine 2 con varianze iniziali sconosc
                     SSMseasonal(7, NA, "dummy") +  # # Stagionalità settimanale (periodo = 7)
                     SSMseasonal(365, NA, "trig", harmonics = 1), # Stagionalità di periodo 365
                   H = NA)                          # Varianza del rumore di misura

  # Inizializza i valori basati sulla varianza della serie di training
  ts_var <- var(train_series_boxcox, na.rm = TRUE)
  inits <- log(c(
    ts_var / 100,    # Varianza del trend lento
    ts_var / 1000,   # Varianza del trend veloce
    ts_var / 10,     # Varianza della stagionalità giornaliera
    ts_var / 1000    # Varianza degli errori di misura
  ))

  updt <- function(pars, model) {
  model$Q[1, 1, 1] <- exp(pars[1])
  model$Q[2, 2, 1] <- exp(pars[2])
  diag(model$Q[-(1:2), -(1:2), 1]) <- exp(pars[3])
  model$H[1, 1, 1] <- exp(pars[4])
  model
  }

  # Adattamento del modello
  fit_mod <- fitSSM(model, inits = inits, updatefn = updt, method = "BFGS")

  # Esecuzione del filtro di Kalman per stimare lo stato latente
  kfs <- KFS(fit_mod$model)

  # Estrai le previsioni sulla scala Box-Cox
  pred_boxcox <- predict(fit_mod$model, n.ahead = length(test_series$X), interval = "none")

  # Converti le previsioni sulla scala originale usando InvBoxCox
  pred_original <- InvBoxCox(pred_boxcox, lambda)

  # Salva i valori di training e previsione nella lista
  test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
  forecast_values_list[[i]] <- pred_original  # Salva le previsioni

  # Calcola il MAE confrontando le previsioni con i valori reali
  mae_values[i] <- mean(abs(pred_original - test_series$X))

  # Salva i residui (scala originale) nella lista
  residuals_list[[i]] <- test_series$X - pred_original
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)
```

```r
# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```r
print(mae_values)
```

```
##  [1] 0.001527042 0.001522194 0.001586656 0.002704173 0.007060525 0.015100915
##  [7] 0.021588566 0.053740370 0.035327918 0.038185161 0.014512142 0.004964293
## [13] 0.003660056 0.004334220 0.004370305 0.004065235 0.003940312 0.003472201
## [19] 0.003823143 0.004108273 0.002651202 0.002527042 0.002794629 0.002707187
```

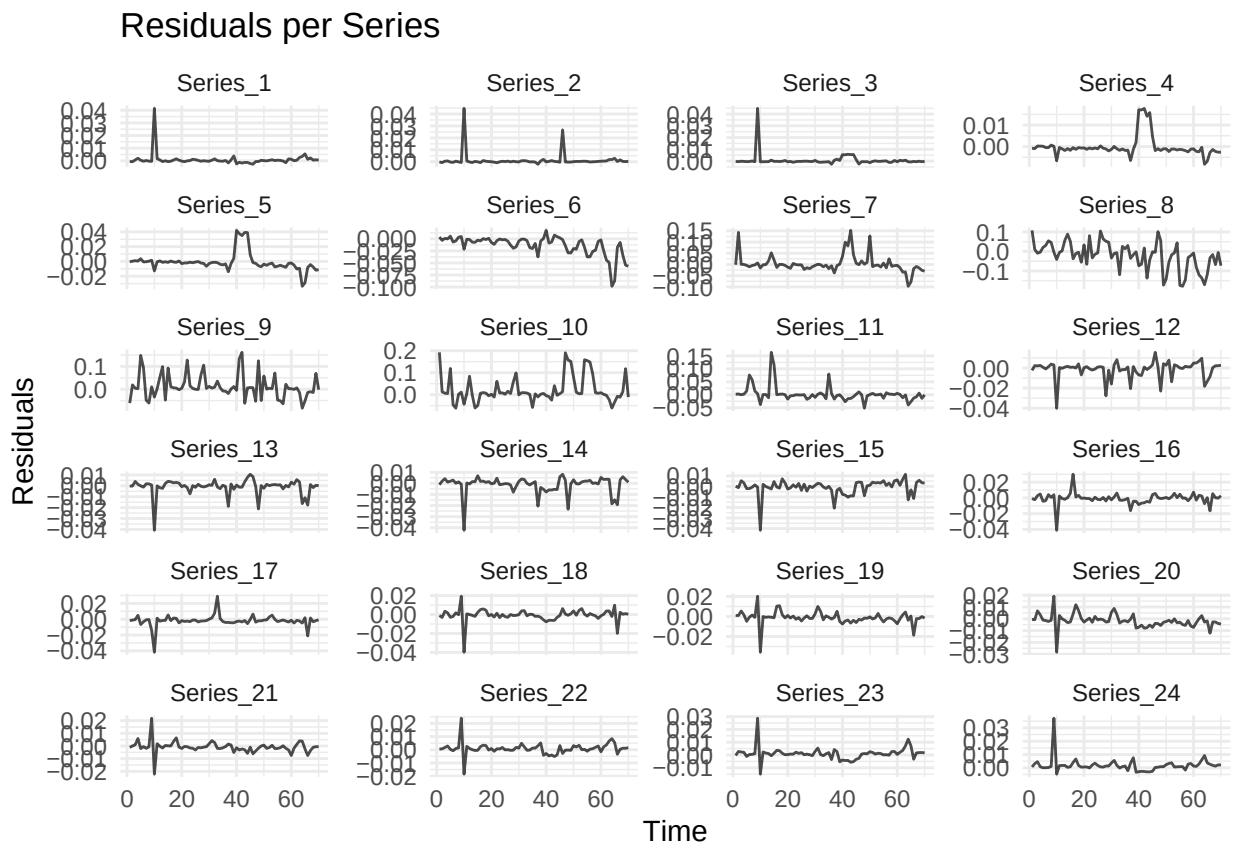
```r
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.01001141
```

```r
print(fit_mod$optim.out$convergence==0)
```

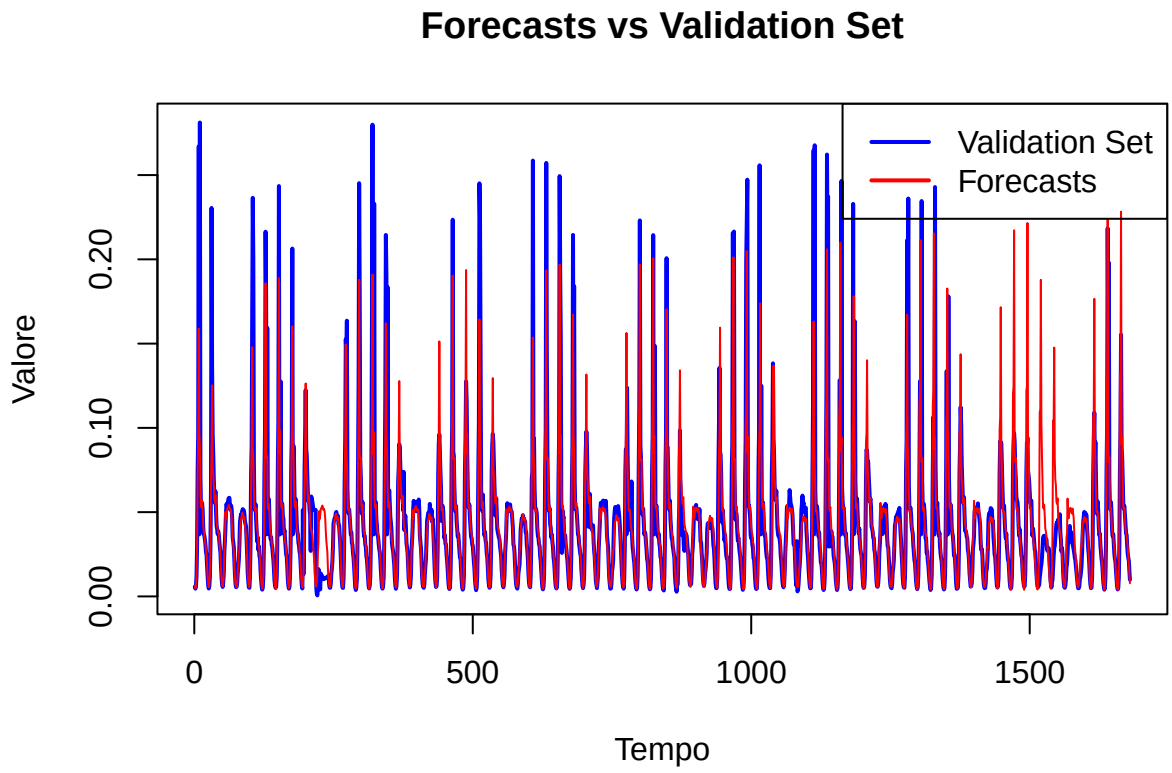
```
## [1] TRUE
```

```r
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



LLT, seasonal dummy (7) and 2 harmonics (365)

```r
# Modello b: LLT e stagionalità trigonometrica di periodo 7

# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()

test_values_list <- list()  # Lista per memorizzare i valori di training
forecast_values_list <- list()  # Lista per memorizzare i valori di forecast

# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test

  # Ottieni il valore di lambda per questa serie ed applicala (da optimal_lambdas)
```

```r
  lambda <- optimal_lambdas[i]
  train_series_boxcox <- BoxCox(train_series$X, lambda)

  # Costruisci il modello UCM: trend locale lineare (LLT) + stagionalità trigonometrica con periodo 7
  model <- SSModel(train_series_boxcox ~
                     SSMtrend(2, list(NA, NA)) +    # LLTrend di ordine 2 con varianze iniziali sconosc
                     SSMseasonal(7, NA, "dummy") +  # # Stagionalità settimanale (periodo = 7)
                     SSMseasonal(365, NA, "trig", harmonics=1:2), # Stagionalità di periodo 365 con 5 a
                   H = NA)                          # Varianza del rumore di misura

  # Inizializza i valori basati sulla varianza della serie di training
  ts_var <- var(train_series_boxcox, na.rm = TRUE)
  inits <- log(c(
    ts_var / 100,    # Varianza del trend lento
    ts_var / 1000,   # Varianza del trend veloce
    ts_var / 10,     # Varianza della stagionalità giornaliera
    ts_var / 1000    # Varianza degli errori di misura
  ))

  updt <- function(pars, model) {
  model$Q[1, 1, 1] <- exp(pars[1])
  model$Q[2, 2, 1] <- exp(pars[2])
  diag(model$Q[-(1:2), -(1:2), 1]) <- exp(pars[3])
  model$H[1, 1, 1] <- exp(pars[4])
  model
  }

  # Adattamento del modello
  fit_mod <- fitSSM(model, inits = inits, updatefn = updt, method = "BFGS")

  # Esecuzione del filtro di Kalman per stimare lo stato latente
  kfs <- KFS(fit_mod$model)

  # Estrai le previsioni sulla scala Box-Cox
  pred_boxcox <- predict(fit_mod$model, n.ahead = length(test_series$X), interval = "none")

  # Converti le previsioni sulla scala originale usando InvBoxCox
  pred_original <- InvBoxCox(pred_boxcox, lambda)

  # Salva i valori di training e previsione nella lista
  test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
  forecast_values_list[[i]] <- pred_original  # Salva le previsioni

  # Calcola il MAE confrontando le previsioni con i valori reali
  mae_values[i] <- mean(abs(pred_original - test_series$X))

  # Salva i residui (scala originale) nella lista
  residuals_list[[i]] <- test_series$X - pred_original
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)
```

```r
# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```r
print(mae_values)
```

```
##  [1] 0.001824315 0.001553705 0.001601014 0.002170653 0.006398607 0.008547852
##  [7] 0.022410140 0.047569546 0.036252904 0.038734289 0.014822391 0.004962430
## [13] 0.003892981 0.004304083 0.004331243 0.004201514 0.003584297 0.004147486
## [19] 0.003792473 0.003455460 0.002483288 0.002454228 0.002637659 0.002220172
```

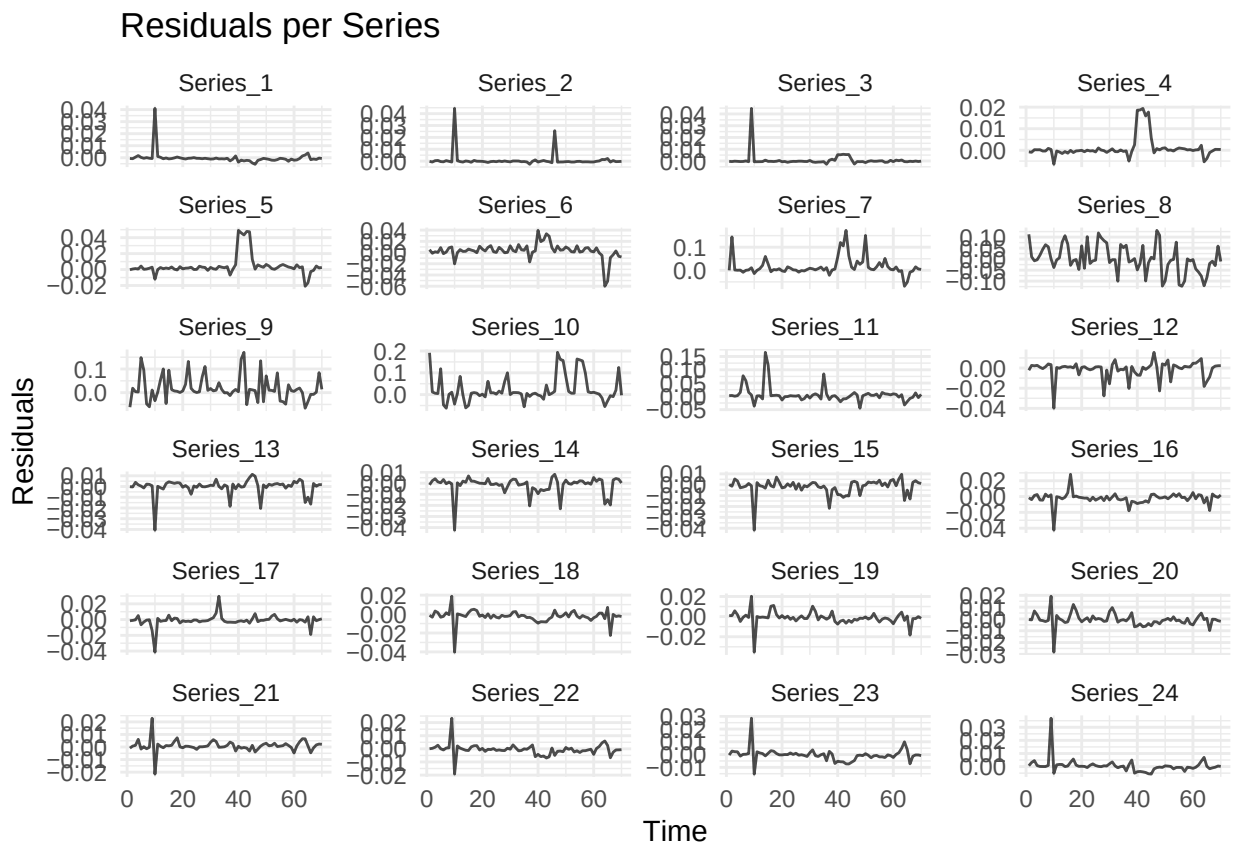
```r
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.009514697
```

```r
print(fit_mod$optim.out$convergence==0)
```

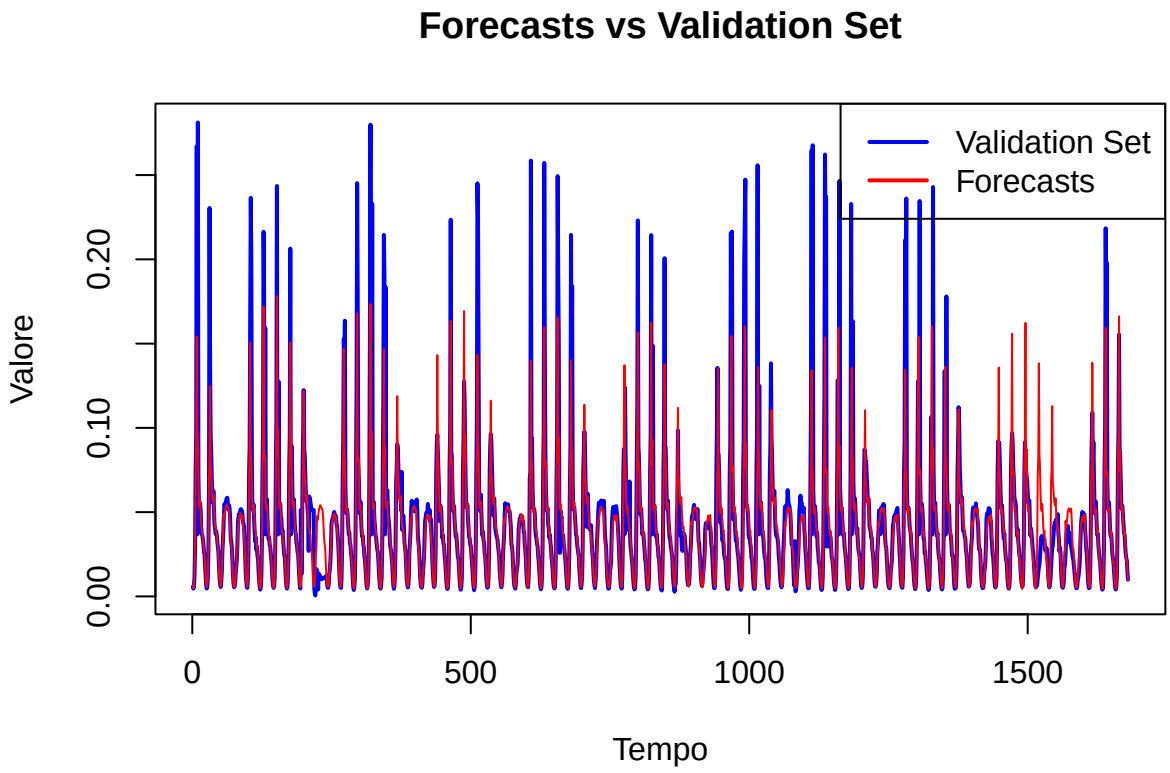
```
## [1] TRUE
```

```r
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



LLT, seasonal dummy (7) and 3 harmonics (365)

```r
# Modello b: LLT e stagionalità trigonometrica di periodo 7

# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()

test_values_list <- list()  # Lista per memorizzare i valori di training
forecast_values_list <- list()  # Lista per memorizzare i valori di forecast

# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test

  # Ottieni il valore di lambda per questa serie ed applicala (da optimal_lambdas)
```

```r
  lambda <- optimal_lambdas[i]
  train_series_boxcox <- BoxCox(train_series$X, lambda)

  # Costruisci il modello UCM: trend locale lineare (LLT) + stagionalità trigonometrica con periodo 7
  model <- SSModel(train_series_boxcox ~
                     SSMtrend(2, list(NA, NA)) +    # LLTrend di ordine 2 con varianze iniziali sconosc
                     SSMseasonal(7, NA, "dummy") +  # # Stagionalità settimanale (periodo = 7)
                     SSMseasonal(365, NA, "trig", harmonics=1:3), # Stagionalità di periodo 365 con 5 a
                   H = NA)                          # Varianza del rumore di misura

  # Inizializza i valori basati sulla varianza della serie di training
  ts_var <- var(train_series_boxcox, na.rm = TRUE)
  inits <- log(c(
    ts_var / 100,    # Varianza del trend lento
    ts_var / 1000,   # Varianza del trend veloce
    ts_var / 10,     # Varianza della stagionalità giornaliera
    ts_var / 1000    # Varianza degli errori di misura
  ))

  updt <- function(pars, model) {
  model$Q[1, 1, 1] <- exp(pars[1])
  model$Q[2, 2, 1] <- exp(pars[2])
  diag(model$Q[-(1:2), -(1:2), 1]) <- exp(pars[3])
  model$H[1, 1, 1] <- exp(pars[4])
  model
  }

  # Adattamento del modello
  fit_mod <- fitSSM(model, inits = inits, updatefn = updt, method = "BFGS")

  # Esecuzione del filtro di Kalman per stimare lo stato latente
  kfs <- KFS(fit_mod$model)

  # Estrai le previsioni sulla scala Box-Cox
  pred_boxcox <- predict(fit_mod$model, n.ahead = length(test_series$X), interval = "none")

  # Converti le previsioni sulla scala originale usando InvBoxCox
  pred_original <- InvBoxCox(pred_boxcox, lambda)

  # Salva i valori di training e previsione nella lista
  test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
  forecast_values_list[[i]] <- pred_original  # Salva le previsioni

  # Calcola il MAE confrontando le previsioni con i valori reali
  mae_values[i] <- mean(abs(pred_original - test_series$X))

  # Salva i residui (scala originale) nella lista
  residuals_list[[i]] <- test_series$X - pred_original
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)
```

```r
# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```r
print(mae_values)
```

```
##  [1] 0.001719272 0.001899349 0.002119782 0.002875357 0.005289159 0.009505776
##  [7] 0.020622661 0.044293092 0.038487707 0.039728584 0.014612448 0.039258640
## [13] 0.045067354 0.050465348 0.004313526 0.004012564 0.003400579 0.003928902
## [19] 0.003980935 0.003375286 0.002531468 0.002409869 0.002708201 0.002453759
```

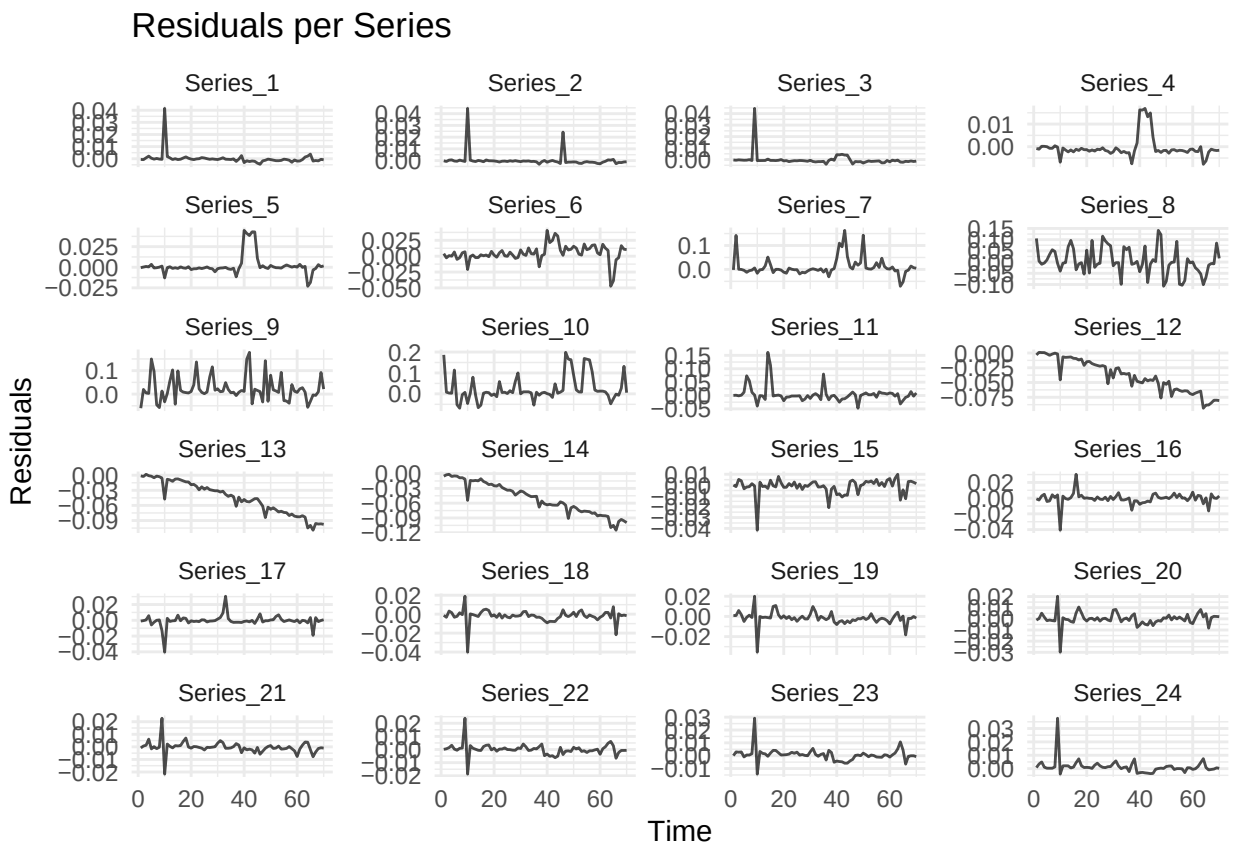
```r
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.01454415
```

```r
print(fit_mod$optim.out$convergence==0)
```

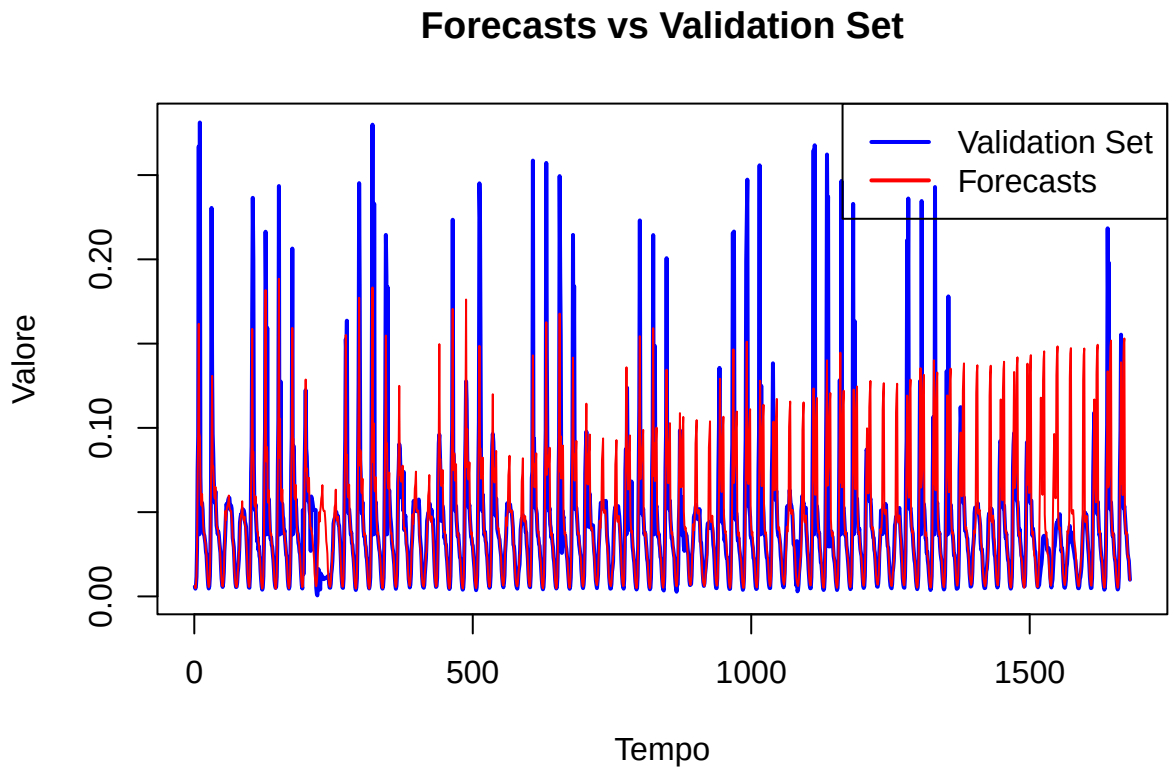
```
## [1] TRUE
```

```r
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



Dummies

LLT, seasonal dummy (7) with dummies

```r
xreg_columns <- c("Dec24", "Dec25", "Dec26", "Jan1", "Jan6",
                  "EasterSat", "Easter", "EasterMon",
                  "EasterTue", "Aug15", "EndYear", "Valentine")

# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()

test_values_list <- list()  # Lista per memorizzare i valori di training
forecast_values_list <- list()  # Lista per memorizzare i valori di forecast

# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
```

```r
train_series <- train_val_series_dummy[[i]]$train
test_series <- train_val_series_dummy[[i]]$test

# Ottieni il valore di lambda per questa serie ed applicala (da optimal_lambdas)
lambda <- optimal_lambdas[i]
train_series_boxcox <- BoxCox(train_series$X, lambda)

# Estrai le variabili dummy dal train set
train_xreg <- train_series[, xreg_columns, drop = FALSE]
test_xreg <- test_series[, xreg_columns, drop = FALSE]

# Costruisci il modello UCM: trend locale lineare (LLT) + stagionalità trigonometrica di periodo 7
model <- SSModel(train_series_boxcox ~
                 SSMtrend(2, list(NA, NA)) +    # LLTrend di ordine 2 con varianze iniziali sconosciu
                 SSMseasonal(7, NA, "dummy") +  # # Stagionalità settimanale (periodo = 7)
                 `Dec24` + `Dec25` + `Dec26` + `Jan1` + `Jan6` +
                 `EasterSat` + `Easter` + `EasterMon` + `EasterTue` +
                 `Aug15` + `EndYear` + `Valentine`,
                 data = train_series,                    # Variabili esogene (dummy) nel modello
                 H = NA)                          # Varianza del rumore di misura

# Inizializza i valori basati sulla varianza della serie di training
ts_var <- var(train_series_boxcox, na.rm = TRUE)
inits <- log(c(
  ts_var / 100,    # Varianza del trend lento
  ts_var / 1000,   # Varianza del trend veloce
  ts_var / 10,     # Varianza della stagionalità settimanale
  ts_var / 1000    # Varianza degli errori di misura
))

# Adattamento del modello
fit_mod <- fitSSM(model, inits = inits, method = "BFGS")

# Esecuzione del filtro di Kalman per stimare lo stato latente
kfs <- KFS(fit_mod$model)

na_per_pred <- c(rep(NA, 70))
empty <- SSModel(na_per_pred ~ SSMtrend(2, list(fit_mod$model$Q[1,1,1], fit_mod$model$Q[2,2,1])) +
                   SSMseasonal(7, fit_mod$model$Q[3,3,1], "dummy") +
                 `Dec24` + `Dec25` + `Dec26` + `Jan1` + `Jan6` +
                 `EasterSat` + `Easter` + `EasterMon` + `EasterTue` +
                 `Aug15` + `EndYear` + `Valentine`,
                 data = test_series,
                 H = fit_mod$model$H)

# Estrai le previsioni sulla scala Box-Cox
pred_boxcox <- predict(fit_mod$model, newdata = empty)

# Converti le previsioni sulla scala originale usando InvBoxCox
pred_original <- InvBoxCox(pred_boxcox, lambda)

# Salva i valori di training e previsione nella lista
test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
```

```
  forecast_values_list[[i]] <- pred_original  # Salva le previsioni

  # Calcola il MAE confrontando le previsioni con i valori reali
  mae_values[i] <- mean(abs(pred_original - test_series$X))

  # Salva i residui (scala originale) nella lista
  residuals_list[[i]] <- test_series$X - pred_original
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```
print(mae_values)
```

```
##  [1] 0.001644455 0.001514093 0.001639006 0.002250347 0.005134824 0.006919275
##  [7] 0.019792686 0.050027896 0.034797361 0.037413159 0.015122201 0.005179665
## [13] 0.004235793 0.004931333 0.004581387 0.005136799 0.003926401 0.003707647
## [19] 0.003903932 0.005073490 0.004386748 0.003489308 0.003131571 0.002179702
```

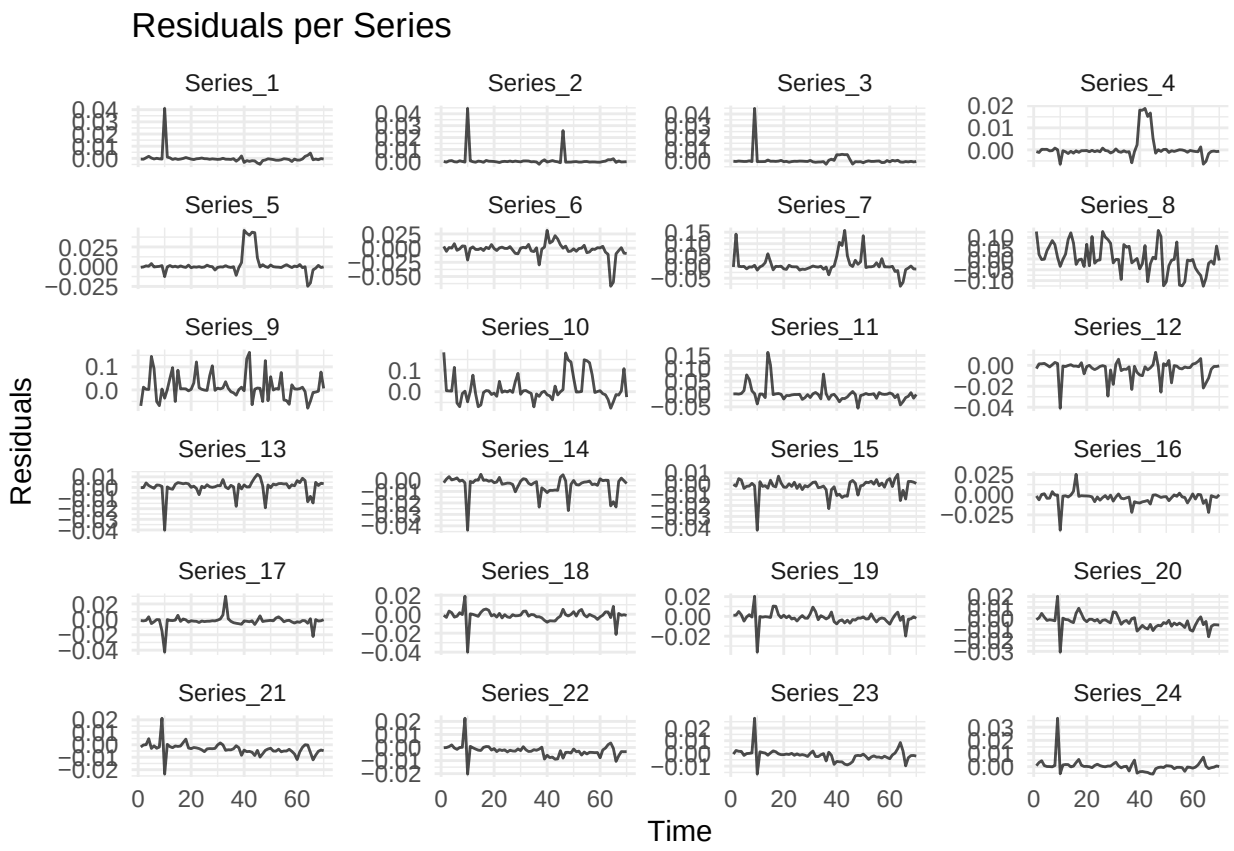
```
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.009588295
```

```
print(fit_mod$optim.out$convergence==0)
```

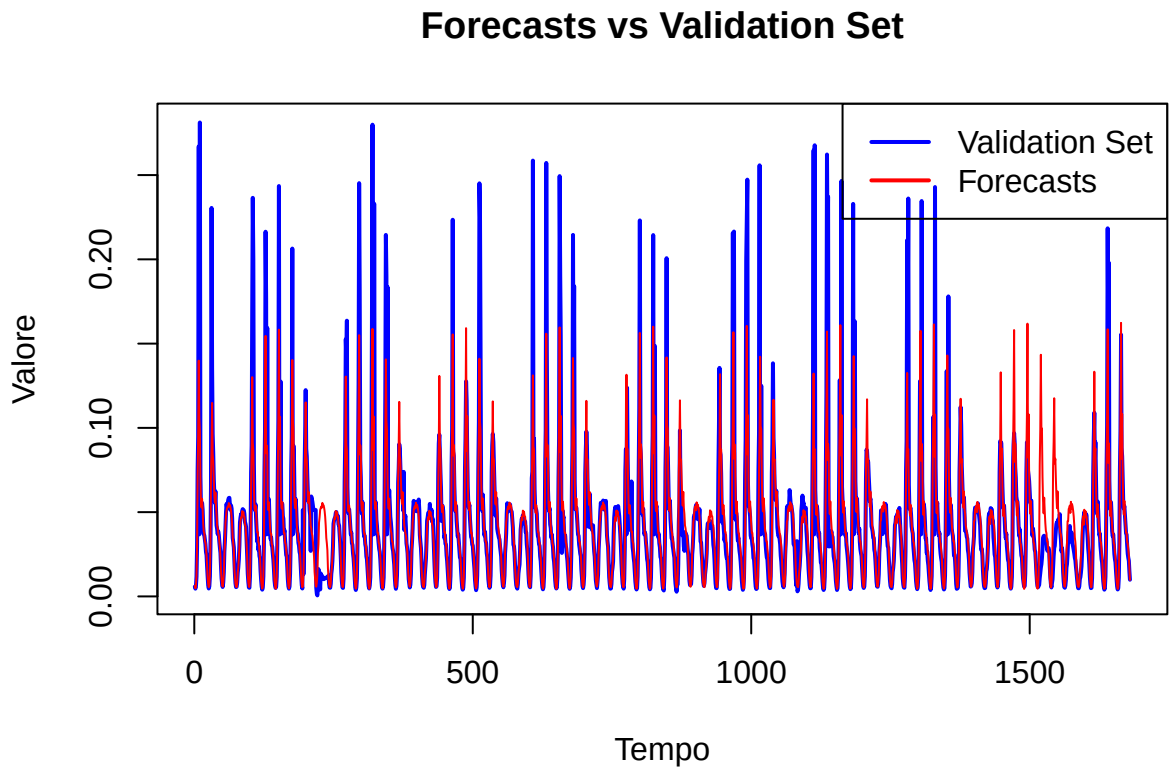
```
## [1] TRUE
```

```
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



LLT, trigometric seasonality (7) with dummies

```r
# Modello b: LLT e stagionalità trigonometrica di periodo 7

# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()

test_values_list <- list()  # Lista per memorizzare i valori di training
forecast_values_list <- list()  # Lista per memorizzare i valori di forecast

# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series_dummy[[i]]$train
  test_series <- train_val_series_dummy[[i]]$test

  # Ottieni il valore di lambda per questa serie ed applicala (da optimal_lambdas)
```

```r
lambda <- optimal_lambdas[i]
train_series_boxcox <- BoxCox(train_series$X, lambda)

# Costruisci il modello UCM: trend locale lineare (LLT) + stagionalità trigonometrica con periodo 7
model <- SSModel(train_series_boxcox ~
                   SSMtrend(2, list(NA, NA)) +    # LLTrend di ordine 2 con varianze iniziali sconosc
                   SSMseasonal(7, NA, "trig") +
                 `Dec24` + `Dec25` + `Dec26` + `Jan1` + `Jan6` +
                 `EasterSat` + `Easter` + `EasterMon` + `EasterTue` +
                 `Aug15` + `EndYear` + `Valentine`,
                 data = train_series,                    # Variabili esogene (dummy) nel modello
                 H = NA)                          # Varianza del rumore di misura

# Inizializza i valori basati sulla varianza della serie di training
ts_var <- var(train_series_boxcox, na.rm = TRUE)
inits <- log(c(
  ts_var / 100,    # Varianza del trend lento
  ts_var / 1000,   # Varianza del trend veloce
  ts_var / 10,     # Varianza della stagionalità giornaliera
  ts_var / 1000    # Varianza degli errori di misura
))

updt <- function(pars, model) {
model$Q[1, 1, 1] <- exp(pars[1])
model$Q[2, 2, 1] <- exp(pars[2])
diag(model$Q[-(1:2), -(1:2), 1]) <- exp(pars[3])
model$H[1, 1, 1] <- exp(pars[4])
model
}

# Adattamento del modello
fit_mod <- fitSSM(model, inits = inits, updatefn = updt, method = "BFGS")

# Esecuzione del filtro di Kalman per stimare lo stato latente
kfs <- KFS(fit_mod$model)

na_per_pred <- c(rep(NA, 70))
empty <- SSModel(na_per_pred ~ SSMtrend(2, list(fit_mod$model$Q[1,1,1], fit_mod$model$Q[2,2,1])) +
                   SSMseasonal(7, fit_mod$model$Q[3,3,1], "trig") +
                 `Dec24` + `Dec25` + `Dec26` + `Jan1` + `Jan6` +
                 `EasterSat` + `Easter` + `EasterMon` + `EasterTue` +
                 `Aug15` + `EndYear` + `Valentine`,
                 data = test_series,
                 H = fit_mod$model$H)

# Estrai le previsioni sulla scala Box-Cox
pred_boxcox <- predict(fit_mod$model, newdata = empty)

# Converti le previsioni sulla scala originale usando InvBoxCox
pred_original <- InvBoxCox(pred_boxcox, lambda)

# Salva i valori di training e previsione nella lista
test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
```

```
  forecast_values_list[[i]] <- pred_original  # Salva le previsioni

  # Calcola il MAE confrontando le previsioni con i valori reali
  mae_values[i] <- mean(abs(pred_original - test_series$X))

  # Salva i residui (scala originale) nella lista
  residuals_list[[i]] <- test_series$X - pred_original
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```
print(mae_values)
```

```
##  [1] 0.001626893 0.001513987 0.001635899 0.002247577 0.005134811 0.007265066
##  [7] 0.020006679 0.048813291 0.034729739 0.037409304 0.015249011 0.005027620
## [13] 0.004100649 0.004968659 0.004583938 0.004886994 0.004115883 0.003682916
## [19] 0.003882566 0.005006195 0.004375419 0.003478559 0.003152112 0.002246975
```

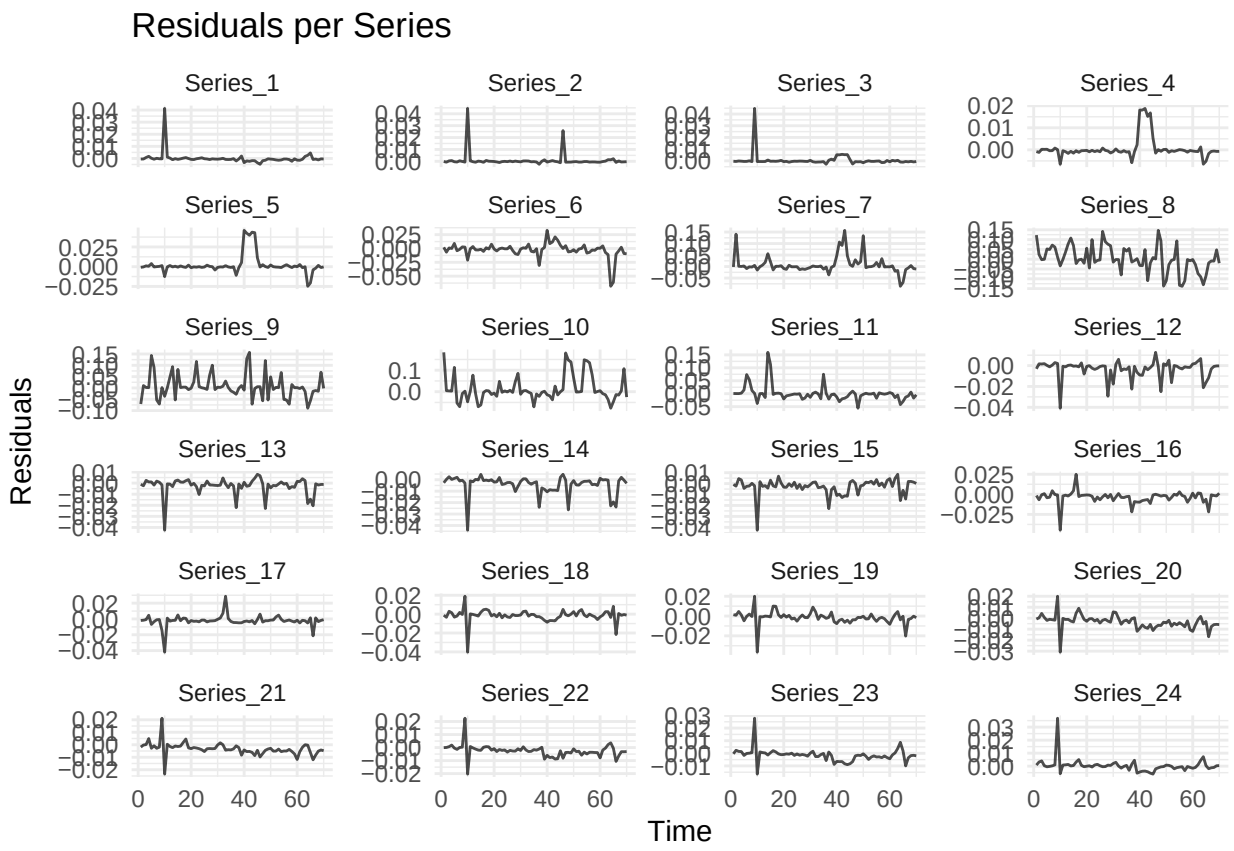
```
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.009547531
```

```
print(fit_mod$optim.out$convergence==0)
```

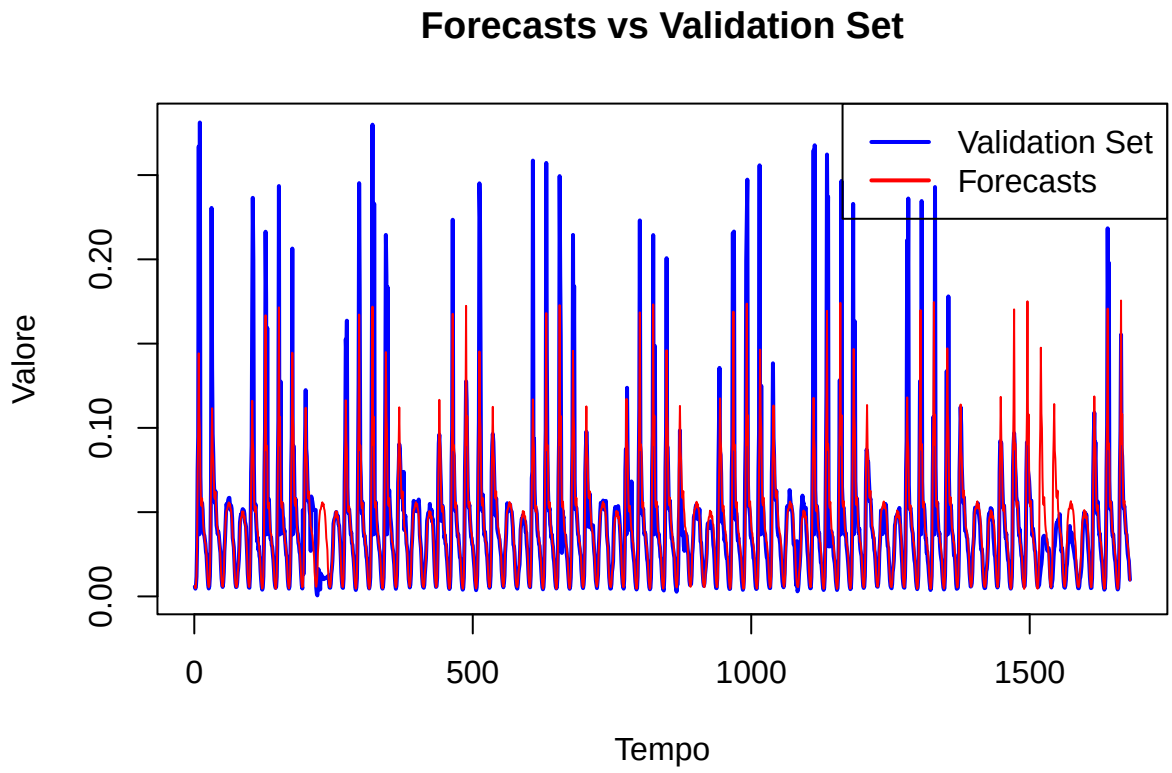
```
## [1] TRUE
```

```
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



LLT, seasonal dummy (7) and 2 harmonics (365) with dummies

```r
# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()

test_values_list <- list()  # Lista per memorizzare i valori di training
forecast_values_list <- list()  # Lista per memorizzare i valori di forecast

# Loop per ogni serie temporale
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series_dummy[[i]]$train
  test_series <- train_val_series_dummy[[i]]$test

  # Ottieni il valore di lambda per questa serie ed applicala (da optimal_lambdas)
  lambda <- optimal_lambdas[i]
```

```r
train_series_boxcox <- BoxCox(train_series$X, lambda)

# Estrai le variabili dummy dal train set
train_xreg <- train_series[, xreg_columns, drop = FALSE]
test_xreg <- test_series[, xreg_columns, drop = FALSE]

# Costruisci il modello UCM: trend locale lineare (LLT) + stagionalità trigonometrica di periodo 7
model <- SSModel(train_series_boxcox ~
                 SSMtrend(2, list(NA, NA)) +    # LLTrend di ordine 2 con varianze iniziali sconosciu
                 SSMseasonal(7, NA, "dummy") +  # # Stagionalità settimanale (periodo = 7)
                 SSMseasonal(365, NA, "trig", harmonics=1:2) + # Stagionalità di periodo 365 con 5 ar
                 `Dec24` + `Dec25` + `Dec26` + `Jan1` + `Jan6` +
                 `EasterSat` + `Easter` + `EasterMon` + `EasterTue` +
                 `Aug15` + `EndYear` + `Valentine`,
                 data = train_series,                    # Variabili esogene (dummy) nel modello
                 H = NA)                          # Varianza del rumore di misura

# Inizializza i valori basati sulla varianza della serie di training
ts_var <- var(train_series_boxcox, na.rm = TRUE)
inits <- log(c(
  ts_var / 100,    # Varianza del trend lento
  ts_var / 1000,   # Varianza del trend veloce
  ts_var / 10,     # Varianza della stagionalità settimanale
  ts_var / 1000    # Varianza degli errori di misura
))

updt <- function(pars, model) {
model$Q[1, 1, 1] <- exp(pars[1])
model$Q[2, 2, 1] <- exp(pars[2])
diag(model$Q[-(1:2), -(1:2), 1]) <- exp(pars[3])
model$H[1, 1, 1] <- exp(pars[4])
model
}

# Adattamento del modello
fit_mod <- fitSSM(model, inits = inits, updatefn = updt, method = "BFGS")

# Esecuzione del filtro di Kalman per stimare lo stato latente
kfs <- KFS(fit_mod$model)

na_per_pred <- c(rep(NA, 70))
empty <- SSModel(na_per_pred ~ SSMtrend(2, list(fit_mod$model$Q[1,1,1], fit_mod$model$Q[2,2,1])) +
                   SSMseasonal(7, fit_mod$model$Q[3,3,1], "dummy") +
                   SSMseasonal(365, fit_mod$model$Q[4,4,1], "trig", harmonics=1:2) +
                 `Dec24` + `Dec25` + `Dec26` + `Jan1` + `Jan6` +
                 `EasterSat` + `Easter` + `EasterMon` + `EasterTue` +
                 `Aug15` + `EndYear` + `Valentine`,
                 data = test_series,
                 H = fit_mod$model$H)

# Estrai le previsioni sulla scala Box-Cox
pred_boxcox <- predict(fit_mod$model, newdata = empty)
```

```
  # Converti le previsioni sulla scala originale usando InvBoxCox
  pred_original <- InvBoxCox(pred_boxcox, lambda)

  # Salva i valori di training e previsione nella lista
  test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
  forecast_values_list[[i]] <- pred_original  # Salva le previsioni

  # Calcola il MAE confrontando le previsioni con i valori reali
  mae_values[i] <- mean(abs(pred_original - test_series$X))

  # Salva i residui (scala originale) nella lista
  residuals_list[[i]] <- test_series$X - pred_original


}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```
print(mae_values)
```

```
##  [1] 0.001791269 0.001533448 0.001614899 0.002204213 0.006209471 0.007405336
##  [7] 0.022223002 0.051911505 0.034734143 0.037283798 0.014841337 0.004854760
## [13] 0.003778557 0.004329386 0.004432131 0.004239913 0.003395103 0.004178102
## [19] 0.003756121 0.003470053 0.002495539 0.002462039 0.002640290 0.002253016
```

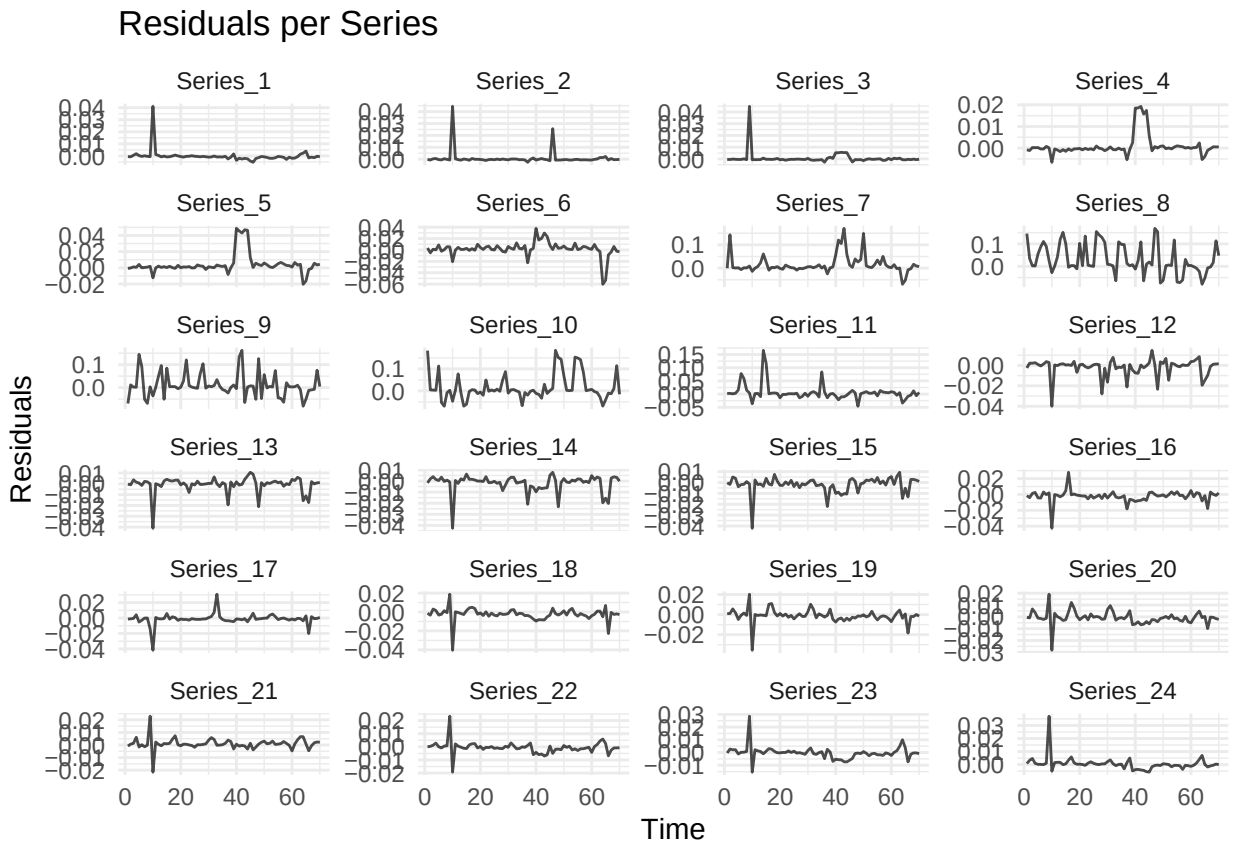
```
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.00950156
```

```
print(fit_mod$optim.out$convergence==0)
```

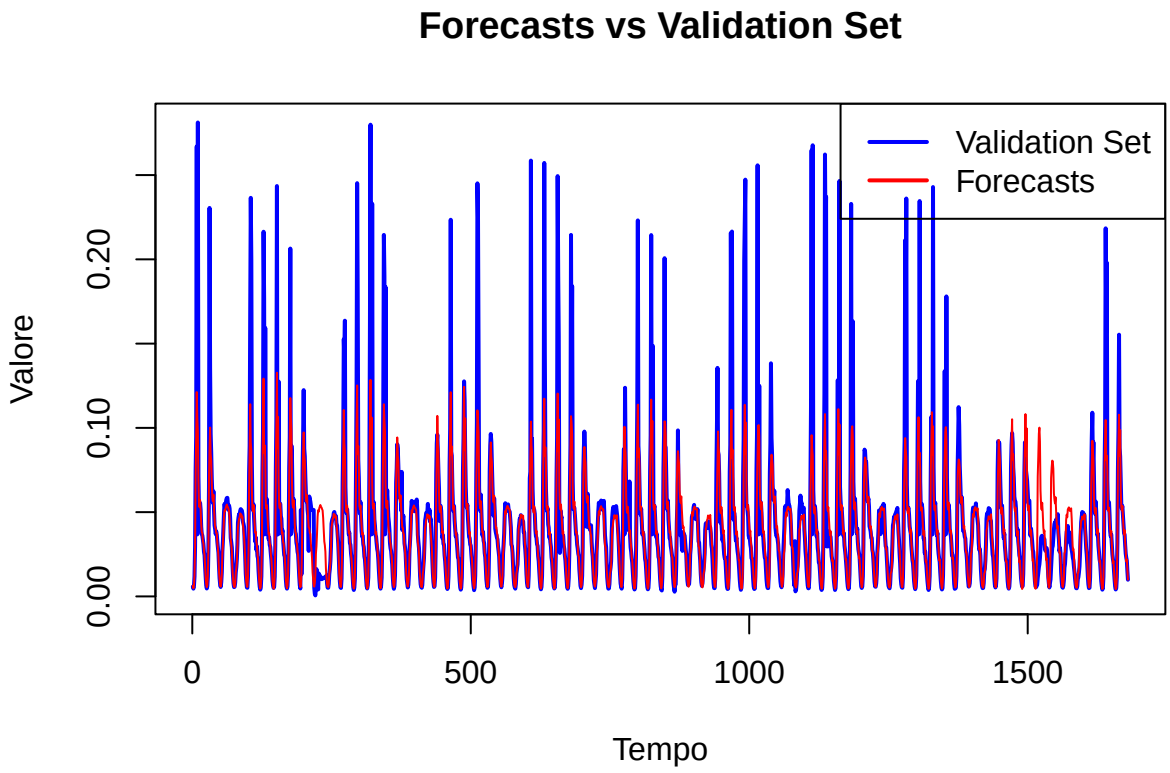
```
## [1] TRUE
```

```
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



Deployment final model

```r
# Lista per memorizzare i MAE
mae_values <- numeric(length = 24)
residuals_list <- list()

test_values_list <- list()  # Lista per memorizzare i valori di training
forecast_results_UCM <- list()  # Lista per memorizzare i valori di forecast

# Loop per ogni serie temporale
for (i in 1:24) {
  print(i)
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_test_series_dummy[[i]]$train
  test_series <- train_test_series_dummy[[i]]$test

  # Ottieni il valore di lambda per questa serie ed applicala (da optimal_lambdas)
  lambda <- optimal_lambdas[i]
```

```r
train_series_boxcox <- BoxCox(train_series$X, lambda)

# Estrai le variabili dummy dal train set
train_xreg <- train_series[, xreg_columns, drop = FALSE]
test_xreg <- test_series[, xreg_columns, drop = FALSE]

# Costruisci il modello UCM: trend locale lineare (LLT) + stagionalità trigonometrica di periodo 7
model <- SSModel(train_series_boxcox ~
                 SSMtrend(2, list(NA, NA)) +    # LLTrend di ordine 2 con varianze iniziali sconosciu
                 SSMseasonal(7, NA, "dummy") +  # # Stagionalità settimanale (periodo = 7)
                 SSMseasonal(365, NA, "trig", harmonics=1:2) + # Stagionalità di periodo 365 con 5 ar
                 `Dec24` + `Dec25` + `Dec26` + `Jan1` + `Jan6` +
                 `EasterSat` + `Easter` + `EasterMon` + `EasterTue` +
                 `Aug15` + `EndYear` + `Valentine`,
                 data = train_series,                    # Variabili esogene (dummy) nel modello
                 H = NA)                          # Varianza del rumore di misura

# Inizializza i valori basati sulla varianza della serie di training
ts_var <- var(train_series_boxcox, na.rm = TRUE)
inits <- log(c(
  ts_var / 100,    # Varianza del trend lento
  ts_var / 1000,   # Varianza del trend veloce
  ts_var / 10,     # Varianza della stagionalità settimanale
  ts_var / 1000    # Varianza degli errori di misura
))

updt <- function(pars, model) {
model$Q[1, 1, 1] <- exp(pars[1])
model$Q[2, 2, 1] <- exp(pars[2])
diag(model$Q[-(1:2), -(1:2), 1]) <- exp(pars[3])
model$H[1, 1, 1] <- exp(pars[4])
model
}

# Adattamento del modello
fit_mod <- fitSSM(model, inits = inits, updatefn = updt, method = "BFGS")

# Esecuzione del filtro di Kalman per stimare lo stato latente
kfs <- KFS(fit_mod$model)

na_per_pred <- c(rep(NA, 31))
empty <- SSModel(na_per_pred ~ SSMtrend(2, list(fit_mod$model$Q[1,1,1],
                                                fit_mod$model$Q[2,2,1])) +
                   SSMseasonal(7, fit_mod$model$Q[3,3,1], "dummy") +
                   SSMseasonal(365, fit_mod$model$Q[4,4,1], "trig", harmonics=1:2) +
                 `Dec24` + `Dec25` + `Dec26` + `Jan1` + `Jan6` +
                 `EasterSat` + `Easter` + `EasterMon` + `EasterTue` +
                 `Aug15` + `EndYear` + `Valentine`,
                 data = test_series,
                 H = fit_mod$model$H)

# Estrai le previsioni sulla scala Box-Cox
pred_boxcox <- predict(fit_mod$model, newdata = empty)
```

```
  # Converti le previsioni sulla scala originale usando InvBoxCox
  pred_original <- InvBoxCox(pred_boxcox, lambda)

  # Salva i valori di training e previsione nella lista
  test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
  forecast_results_UCM[[i]] <- pred_original  # Salva le previsioni

  # Calcola il MAE confrontando le previsioni con i valori reali
  mae_values[i] <- mean(abs(pred_original - test_series$X))

  # Salva i residui (scala originale) nella lista
  residuals_list[[i]] <- test_series$X - pred_original


}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
```

```
# Stampa un riepilogo delle previsioni
cat("Previsioni completate per le 24 serie temporali.\n")
```

```
## Previsioni completate per le 24 serie temporali.
```

```
forecast_results_UCM[[1]][1:4]
```

```
## [1] 0.006095668 0.006688912 0.009757098 0.010678105
```

```r
forecast_results_UCM[[2]][1:4]
```

```
## [1] 0.004434946 0.004940798 0.006205971 0.006520038
```

```r
forecast_results_UCM[[3]][1:4]
```

```
## [1] 0.005034438 0.005701661 0.005050244 0.004565853
```

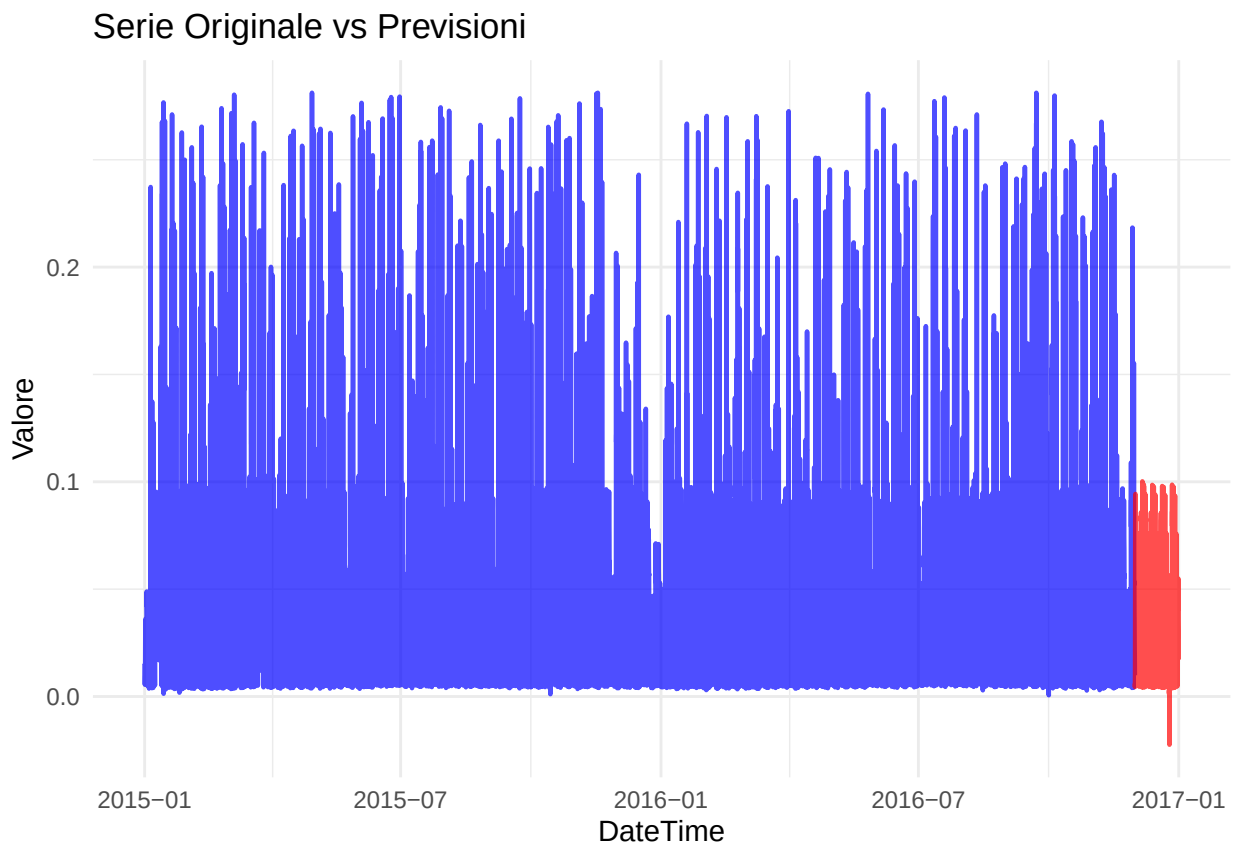
```r
# Creazione della base DateTime
start_date <- as.POSIXct("2016-12-01 00:00:00")
num_days <- length(forecast_results_UCM[[1]])  # Numero di giorni previsti
hourly_series_length <- num_days * 24  # Numero totale di ore

# Genera il vettore DateTime per ogni ora
datetime_vector <- seq(start_date, by = "hour", length.out = hourly_series_length)

# Inizializza un vettore per salvare tutte le previsioni
forecast_vector_UCM <- numeric(hourly_series_length)

# Ricomponi la serie oraria
for (i in 1:24) {
  # Inserisci le previsioni nella posizione corretta
  forecast_vector_UCM[seq(i, hourly_series_length, by = 24)] <- forecast_results_UCM[[i]]
}

# Creazione del data frame finale
forecast_dataframe <- data.frame(
  DateTime = datetime_vector,
  ARIMA = forecast_vector_ARIMA,
  UCM = forecast_vector_UCM
)

# Visualizza le prime righe del risultato
head(forecast_dataframe)
```

```
##              DateTime       ARIMA         UCM
## 1 2016-12-01 00:00:00 0.005836121 0.006095668
## 2 2016-12-01 01:00:00 0.004403759 0.004434946
## 3 2016-12-01 02:00:00 0.005211713 0.005034438
## 4 2016-12-01 03:00:00 0.011759774 0.011392291
## 5 2016-12-01 04:00:00 0.030413454 0.028460888
## 6 2016-12-01 05:00:00 0.068222103 0.049397717
```

```r
dim(forecast_dataframe)
```

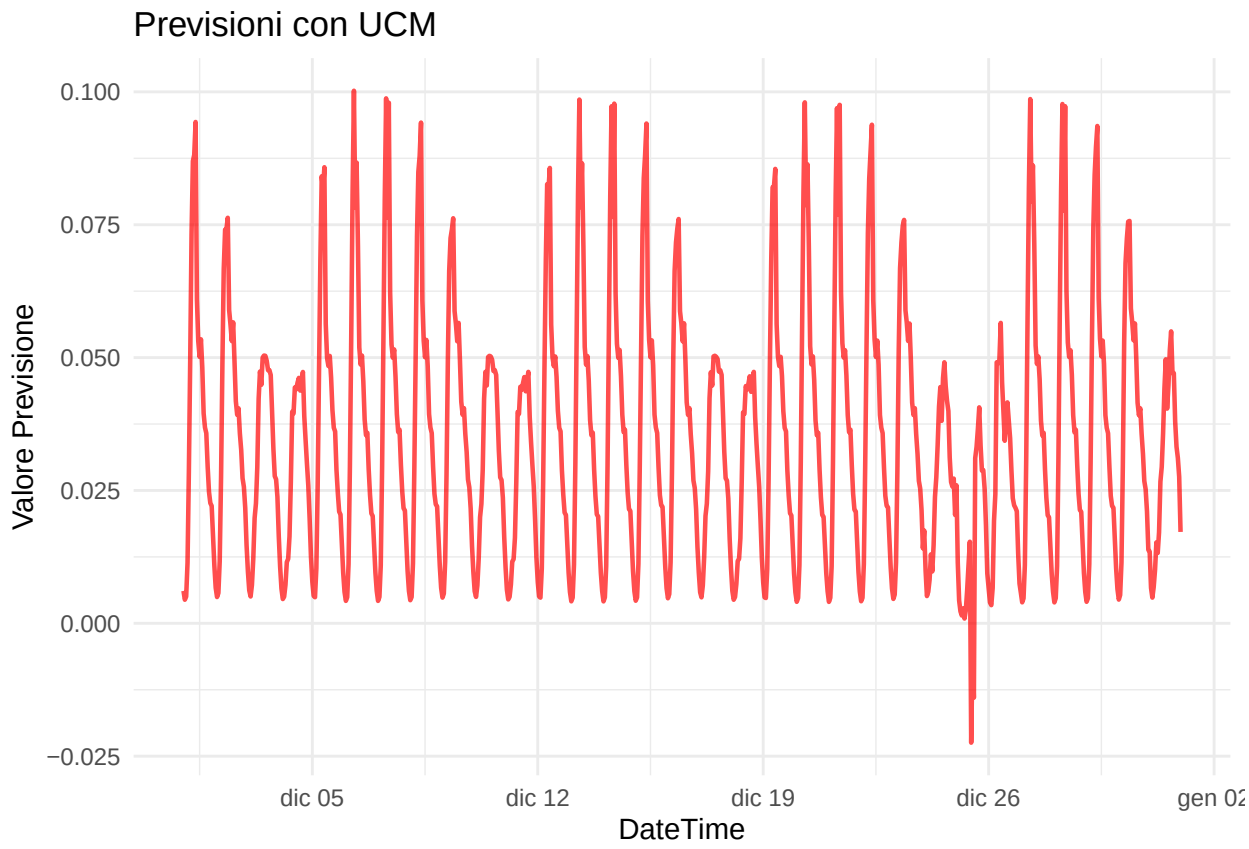
```
## [1] 744   3
```

```r
library(ggplot2)

# Filtra la serie originale per sovrapporre solo fino all'ultima osservazione reale
ts_train_df <- subset(ts_cutted_filled_df, Date < as.POSIXct("2016-12-01 00:00:00"))

# Combina i dati di training e le previsioni
forecast_df <- data.frame(
  DateTime = forecast_dataframe$DateTime,
  Forecast = forecast_dataframe$UCM
)

# Plot con ggplot2
ggplot() +
  # Serie originale (parte di training)
  geom_line(data = ts_train_df, aes(x = Date, y = Value), color = "blue", size = 0.8, alpha = 0.7) +
  # Previsioni
  geom_line(data = forecast_df, aes(x = DateTime, y = Forecast), color = "red", size = 0.8, alpha = 0.7
  # Personalizzazione
  labs(title = "Serie Originale vs Previsioni",
       x = "DateTime", y = "Valore") +
  theme_minimal()
```




```r
library(ggplot2)

# Combina i dati delle previsioni
```

```r
forecast_df <- data.frame(
  DateTime = forecast_dataframe$DateTime,
  Forecast = forecast_dataframe$UCM
)

# Plot con ggplot2: solo previsioni
ggplot(data = forecast_df) +
  # Previsioni
  geom_line(aes(x = DateTime, y = Forecast), color = "red", size = 0.8, alpha = 0.7) +
  # Personalizzazione
  labs(title = "Previsioni con UCM",
       x = "DateTime", y = "Valore Previsione") +
  theme_minimal()
```




```r
# Specifica il percorso e il nome del file CSV
file_path <- "forecast_dataframe.csv"

# Salva il data frame come CSV
write.csv(forecast_dataframe, file = file_path, row.names = FALSE)

# Conferma il salvataggio
cat("Il file è stato salvato in:", file_path, "\n")
```

```
## Il file è stato salvato in: forecast_dataframe.csv
```

Machine Learning (ML)

Preprocessing (division dataset and box cox transformation)

```r
# Carica i dati
data_dummies <- read.csv("ts2024_dummiesML.csv")

# Assumiamo che i dataset abbiano una colonna "DateTime" e che le date siano stringhe
data_dummies$DateTime <- as.POSIXct(data_dummies$DateTime, format = "%Y-%m-%d %H:%M:%S")

# Definizione dei limiti temporali
validation_start <- as.POSIXct("2016-09-22 00:00:00")
test_start <- as.POSIXct("2016-12-01 00:00:00")

# Imputazione della mediana per Lag_X
data_dummies$Lag_X[is.na(train_data$Lag_X)] <- median(train_data$Lag_X, na.rm = TRUE)

# Imputazione della mediana per Lag_X7
data_dummies$Lag_X7[is.na(train_data$Lag_X7)] <- median(train_data$Lag_X7, na.rm = TRUE)

# Imputazione della mediana per Diff_X
data_dummies$Diff_X[is.na(train_data$Diff_X)] <- median(train_data$Diff_X, na.rm = TRUE)

# Imputazione della mediana per Diff_X7
data_dummies$Diff_X7[is.na(train_data$Diff_X7)] <- median(train_data$Diff_X7, na.rm = TRUE)
```

```r
# Imputazione della mediana per Lag_X
data_dummies$Lag_X[is.na(train_data$Lag_X)] <- median(train_data$Lag_X, na.rm = TRUE)

# Imputazione della mediana per Lag_X7
data_dummies$Lag_X7[is.na(train_data$Lag_X7)] <- median(train_data$Lag_X7, na.rm = TRUE)

# Imputazione della mediana per Diff_X
data_dummies$Diff_X[is.na(train_data$Diff_X)] <- median(train_data$Diff_X, na.rm = TRUE)

# Imputazione della mediana per Diff_X7
data_dummies$Diff_X7[is.na(train_data$Diff_X7)] <- median(train_data$Diff_X7, na.rm = TRUE)

# Divisione del dataset con dummy
train_data_dummies <- subset(data_dummies, DateTime < validation_start)
validation_data_dummies <- subset(data_dummies, DateTime >=
                                  validation_start & DateTime < test_start)
test_data_dummies <- subset(data_dummies, DateTime >= test_start)

# Selezione delle colonne rilevanti
selected_columns <- c("DateTime", "Date", "Hour", "X", "Lag_X", "Lag_X7", "Diff_X", "Diff_X7", "DayOfWe
#selected_columns <- c("DateTime", "Date", "Hour", "X", "Lag_X")

# Filtra le colonne nei dataset di training, validation e test
train_data <- train_data_dummies[, selected_columns, drop = FALSE]
validation_data <- validation_data_dummies[, selected_columns, drop = FALSE]
test_data <- test_data_dummies[, selected_columns, drop = FALSE]
```

```r
# Verifica delle dimensioni
cat("Dimensioni dataset originale:\n")
```

```
## Dimensioni dataset originale:
```

```r
cat("Train: ", nrow(train_data), "\n")
```

```
## Train:  15119
```

```r
cat("Validation: ", nrow(validation_data), "\n")
```

```
## Validation:  1681
```

```r
cat("Test: ", nrow(test_data), "\n")
```

```
## Test:  744
```

```r
cat("\nDimensioni dataset con dummy:\n")
```

```
##
## Dimensioni dataset con dummy:
```

```r
cat("Train: ", nrow(train_data_dummies), "\n")
```

```
## Train:  15119
```

```r
cat("Validation: ", nrow(validation_data_dummies), "\n")
```

```
## Validation:  1681
```

```r
cat("Test: ", nrow(test_data_dummies), "\n")
```

```
## Test:  744
```

```r
# Converti i dataframe in oggetti xts
train_xts <- xts(train_data[ , -1], order.by = train_data$DateTime)
validation_xts <- xts(validation_data[ , -1], order.by = validation_data$DateTime)
test_xts <- xts(test_data[ , -1], order.by = test_data$DateTime)

train_xts_dummies <- xts(train_data_dummies[ , -1], order.by = train_data_dummies$DateTime)
validation_xts_dummies <- xts(validation_data_dummies[ , -1], order.by = validation_data_dummies$DateTi
test_xts_dummies <- xts(test_data_dummies[ , -1], order.by = test_data_dummies$DateTime)
```

```r
head(train_data)
```

```
##              DateTime       Date Hour      X  Lag_X Lag_X7  Diff_X Diff_X7
## 1 2015-01-01 00:00:00 2015-01-01    0 0.0146 0.0100     NA      NA      NA
## 2 2015-01-01 01:00:00 2015-01-01    1 0.0148 0.0146     NA  0.0002      NA
## 3 2015-01-01 02:00:00 2015-01-01    2 0.0101 0.0148     NA -0.0047      NA
## 4 2015-01-01 03:00:00 2015-01-01    3 0.0060 0.0101     NA -0.0041      NA
## 5 2015-01-01 04:00:00 2015-01-01    4 0.0055 0.0060     NA -0.0005      NA
## 6 2015-01-01 05:00:00 2015-01-01    5 0.0071 0.0055     NA  0.0016      NA
##   DayOfWeek DayOfYear
## 1         3         1
```

```
## 2         3         1
## 3         3         1
## 4         3         1
## 5         3         1
## 6         3         1
```

```r
head(train_data_dummies)
```

```
##              DateTime       Date Hour      X  Lag_X Lag_X7  Diff_X Diff_X7
## 1 2015-01-01 00:00:00 2015-01-01    0 0.0146 0.0100     NA      NA      NA
## 2 2015-01-01 01:00:00 2015-01-01    1 0.0148 0.0146     NA  0.0002      NA
## 3 2015-01-01 02:00:00 2015-01-01    2 0.0101 0.0148     NA -0.0047      NA
## 4 2015-01-01 03:00:00 2015-01-01    3 0.0060 0.0101     NA -0.0041      NA
## 5 2015-01-01 04:00:00 2015-01-01    4 0.0055 0.0060     NA -0.0005      NA
## 6 2015-01-01 05:00:00 2015-01-01    5 0.0071 0.0055     NA  0.0016      NA
##   DayOfWeek DayOfYear IsWeekend IsHoliday Season Dec24 Dec25 Dec26 Jan1 Jan6
## 1         3         1         0         1 Winter     0     0     0    1    0
## 2         3         1         0         1 Winter     0     0     0    1    0
## 3         3         1         0         1 Winter     0     0     0    1    0
## 4         3         1         0         1 Winter     0     0     0    1    0
## 5         3         1         0         1 Winter     0     0     0    1    0
## 6         3         1         0         1 Winter     0     0     0    1    0
##   EasterSat Easter EasterMon EasterTue Aug15 EndYear Valentine
## 1         0      0         0         0     0       0         0
## 2         0      0         0         0     0       0         0
## 3         0      0         0         0     0       0         0
## 4         0      0         0         0     0       0         0
## 5         0      0         0         0     0       0         0
## 6         0      0         0         0     0       0         0
```

```r
set.seed(213654897)
library(randomForest)
library(xts)
library(ggplot2)
library(lubridate)
library(forecast)
#install.packages("fastDummies")
library(fastDummies)
library(Metrics)
library("KFAS")
#install.packages("randomForest")
library(randomForest)
#install.packages("xgboost")
library(xgboost)
```

```r
# Carica i dati
data <- read.csv("ts2024_dummiesML.csv")
```

```r
# Assumiamo che i dataset abbiano una colonna "DateTime" e che le date siano stringhe
data$DateTime <- as.POSIXct(data$DateTime, format = "%Y-%m-%d %H:%M:%S")

# Definizione dei limiti temporali
validation_start <- as.POSIXct("2016-09-22 00:00:00")
test_start <- as.POSIXct("2016-12-01 00:00:00")

# Imputazione della mediana per Lag_X
data$Lag_X[is.na(train_data$Lag_X)] <- median(train_data$Lag_X, na.rm = TRUE)

# Imputazione della mediana per Lag_X7
data$Lag_X7[is.na(train_data$Lag_X7)] <- median(train_data$Lag_X7, na.rm = TRUE)

# Imputazione della mediana per Diff_X
data$Diff_X[is.na(train_data$Diff_X)] <- median(train_data$Diff_X, na.rm = TRUE)

# Imputazione della mediana per Diff_X7
data$Diff_X7[is.na(train_data$Diff_X7)] <- median(train_data$Diff_X7, na.rm = TRUE)
```

```r
# Supponendo che il dataset abbia una colonna 'Hour' (da 0 a 23) e 'value' (la serie temporale)
# Inizializzare una lista per contenere le 24 serie temporali
selected_columns <- c("DateTime", "Date", "Hour", "X", "Lag_X", "Lag_X7", "Diff_X", "Diff_X7", "DayOfWe
hourly_series <- list()
hourly_series_dummy <- list()

# Ciclo per filtrare e salvare ogni serie oraria
for (hour in 0:23) {
  # Filtra i dati per l'ora specifica
  hourly_series[[hour + 1]] <- data[, selected_columns, drop = FALSE] %>%
    filter(Hour == hour)
  hourly_series_dummy[[hour + 1]] <- data %>% filter(Hour == hour)
}

# Visualizzare la serie per una specifica ora (ad esempio, per le 12)
head(hourly_series[[13]])  # L'elemento 13 corrisponde alle 12:00 perche R indicizza da 1 e non 0
```

```
##              DateTime       Date Hour      X  Lag_X  Lag_X7  Diff_X  Diff_X7
## 1 2015-01-01 12:00:00 2015-01-01   12 0.0329 0.0298 0.00710  0.0031  0.02580
## 2 2015-01-02 12:00:00 2015-01-02   12 0.0489 0.0485 0.02355  0.0004  0.02535
## 3 2015-01-03 12:00:00 2015-01-03   12 0.0454 0.0444 0.01550  0.0010  0.02990
## 4 2015-01-04 12:00:00 2015-01-04   12 0.0123 0.0317 0.01230 -0.0194  0.00000
## 5 2015-01-05 12:00:00 2015-01-05   12 0.0447 0.0485 0.04190 -0.0038  0.00280
## 6 2015-01-06 12:00:00 2015-01-06   12 0.0489 0.0531 0.07200 -0.0042 -0.02310
##   DayOfWeek DayOfYear
## 1         3         1
## 2         4         2
## 3         5         3
## 4         6         4
## 5         0         5
## 6         1         6
```

```r
# Funzione per dividere i dati in train e test
train_test_split <- function(hourly_series, train_end) {
  train_set <- hourly_series[1:train_end, ]  # Da 1 a 'train_end' per il train
  test_set <- hourly_series[(train_end+1):nrow(hourly_series), ]  # Rimanenti per il test
  list(train = train_set, test = test_set)
}

# Impostiamo il valore di train_end
train_end <- 700  # Gli ultimi 31 vanno nel test

# Creare una lista per salvare i train e test per tutte le ore per i due dataset
train_test_series <- list()
train_test_series_dummy <- list()

# Ciclo per creare i train e test per ogni ora (0 a 23) per entrambe le serie
for (hour in 0:23) {
  # Per hourly_series
  hour_data <- hourly_series[[hour + 1]]
  train_test_series[[hour + 1]] <- train_test_split(hour_data, train_end)

  # Per hourly_series_dummy
  hour_data_dummy <- hourly_series_dummy[[hour + 1]]
  train_test_series_dummy[[hour + 1]] <- train_test_split(hour_data_dummy, train_end)
}

# Verifica per le 12:00
cat("Train set for 12:00 (original):\n")
```

```
## Train set for 12:00 (original):
```

```r
dim(train_test_series[[13]]$train)
```

```
## [1] 700  10
```

```r
dim(train_test_series[[13]]$test)
```

```
## [1] 31 10
```

```r
cat("Train set for 12:00 (dummy):\n")
```

```
## Train set for 12:00 (dummy):
```

```r
dim(train_test_series_dummy[[13]]$train)
```

```
## [1] 700  25
```

```r
dim(train_test_series_dummy[[13]]$test)
```

```
## [1] 31 25
```

```r
# Funzione per dividere i dati in train e test
train_test_split <- function(hourly_series, train_end) {
  train_set <- hourly_series[1:train_end, ]  # Da 1 a 'train_end' per il train
  test_set <- hourly_series[(train_end+1):700, ]  # Fino a 700 per il test
```

```
  list(train = train_set, test = test_set)
}

# Impostiamo il valore di train_end
train_end <- 630  # Gli ultimi 70 vanno nel test (da 631 a 700)

# Creare una lista per salvare i train e test per tutte le ore per i due dataset
train_val_series <- list()
train_val_series_dummy <- list()

# Ciclo per creare i train e test per ogni ora (0 a 23) per entrambe le serie
for (hour in 0:23) {
  # Per hourly_series
  hour_data <- hourly_series[[hour + 1]]
  train_val_series[[hour + 1]] <- train_test_split(hour_data, train_end)

  # Per hourly_series_dummy
  hour_data_dummy <- hourly_series_dummy[[hour + 1]]
  train_val_series_dummy[[hour + 1]] <- train_test_split(hour_data_dummy, train_end)
}

# Verifica per le 12:00
cat("Train set for 12:00 (original):\n")
```

```
## Train set for 12:00 (original):
```

```
dim(train_val_series[[13]]$train)
```

```
## [1] 630  10
```

```
dim(train_val_series[[13]]$test)
```

```
## [1] 70 10
```

```
cat("Train set for 12:00 (dummy):\n")
```

```
## Train set for 12:00 (dummy):
```

```
dim(train_val_series_dummy[[13]]$train)
```

```
## [1] 630  25
```

```
dim(train_val_series_dummy[[13]]$test)
```

```
## [1] 70 25
```

Without dummies

Random Forest

```
# Lista per memorizzare i MAE per ogni serie temporale
mae_values <- numeric(length = 24)

# Liste per memorizzare i valori di training, forecast e residui
test_values_list <- list()
```

```r
forecast_values_list <- list()
residuals_list <- list()

# Ciclo per le 24 serie temporali
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test

  # Trasformazione logaritmica per stabilizzare la varianza nei dati di allenamento
  train_series$Lag_X7[1] <- train_series$Lag_X7[2]
  train_series$Diff_X[1] <- train_series$Diff_X[2]
  train_series$Diff_X7[1] <- train_series$Diff_X7[2]

  # Crea il modello Random Forest (o il modello che stai utilizzando)
  rf_model <- randomForest(X ~ ., data = train_series, importance = TRUE)

  # Inizializza il vettore per le previsioni rolling
  z_hatr <- numeric(nrow(test_series))

  for (h in 1:nrow(test_series)) {
    # Crea una nuova riga con la previsione corrente
    new_row <- data.frame(
      DateTime = test_series$DateTime[h],
      Date = as.character(test_series$Date[h]),
      Hour = test_series$Hour[h],
      X = z_hatr[h],  # La previsione corrente
      Lag_X = ifelse(h > 1, z_hatr[h-1], NA),
      Lag_X7 = ifelse(h > 7, z_hatr[h-7], NA),
      Diff_X = ifelse(h > 1, z_hatr[h] - z_hatr[h-1], NA),
      Diff_X7 = ifelse(h > 7, z_hatr[h] - z_hatr[h-7], NA),
      DayOfWeek = test_series$DayOfWeek[h],
      DayOfYear = test_series$DayOfYear[h],
      stringsAsFactors = FALSE
    )

    # Aggiungi la nuova riga al dataset di addestramento
    train_data_with_pred <- rbind(train_series, new_row)

    # Rimuovi eventuali righe con NA
    train_data_with_pred_clean <- na.omit(train_data_with_pred)

    # Previsione con il nuovo dataset di allenamento aggiornato
    z_hatr[h] <- predict(rf_model, newdata = train_data_with_pred_clean[nrow(train_data_with_pred_clean
  }

  # Salva i valori di training e previsione nella lista
  test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
  forecast_values_list[[i]] <- z_hatr  # Salva le previsioni

  # Calcola MAE tra le previsioni e i valori reali sulla scala originale
  mae_values[i] <- mean(abs(z_hatr - test_series$X), na.rm = TRUE)
```

```r
  # Calcola i residui sulla scala originale
  residuals_list[[i]] <- test_series$X - z_hatr
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

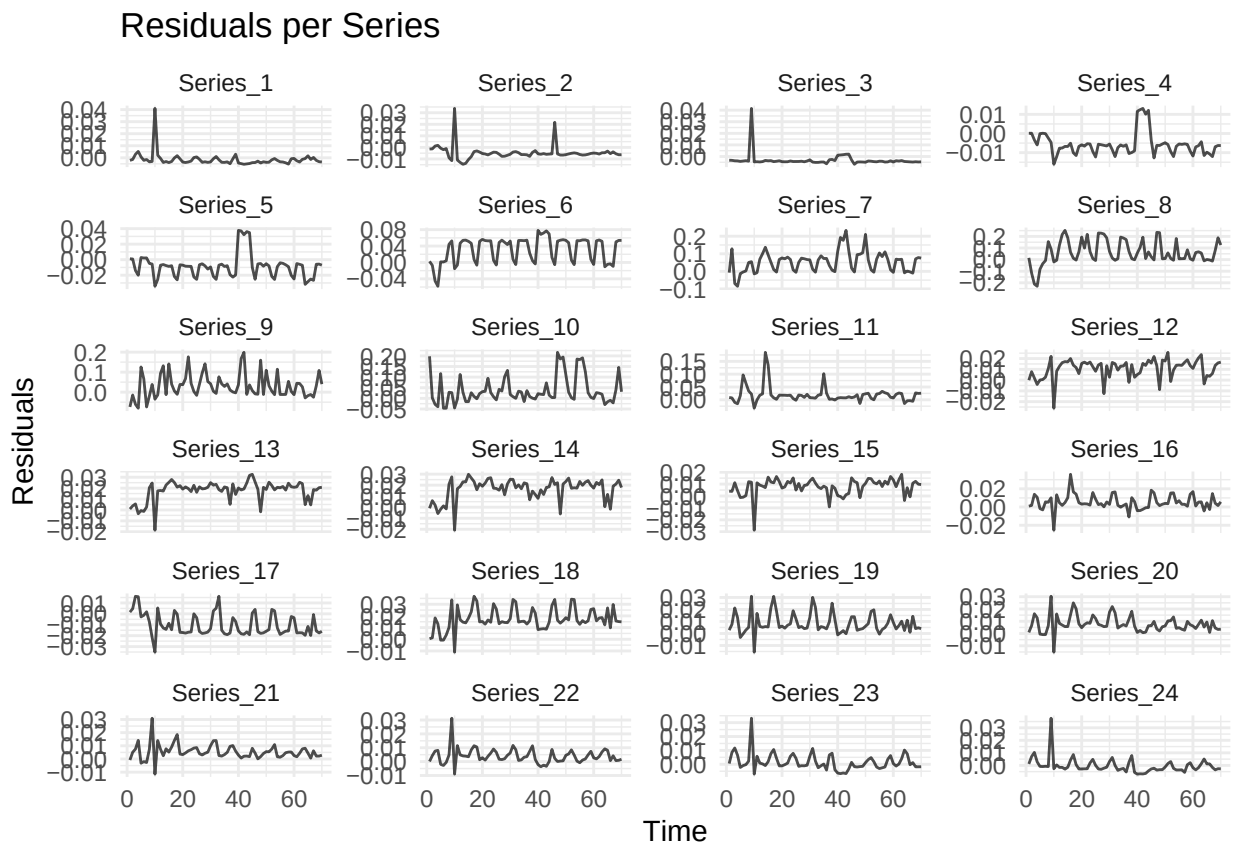
```r
print(mae_values)
```

```
##  [1] 0.003257612 0.006352170 0.004588075 0.007351069 0.014066393 0.036782663
##  [7] 0.061659601 0.085821369 0.048738405 0.049424095 0.022198188 0.012304171
## [13] 0.018336281 0.017514748 0.009229551 0.007095566 0.012474410 0.017720617
## [19] 0.009051312 0.008248263 0.006014140 0.004721578 0.003952516 0.003660869
```

```r
cat("MAE: ", mae, "\n")
```

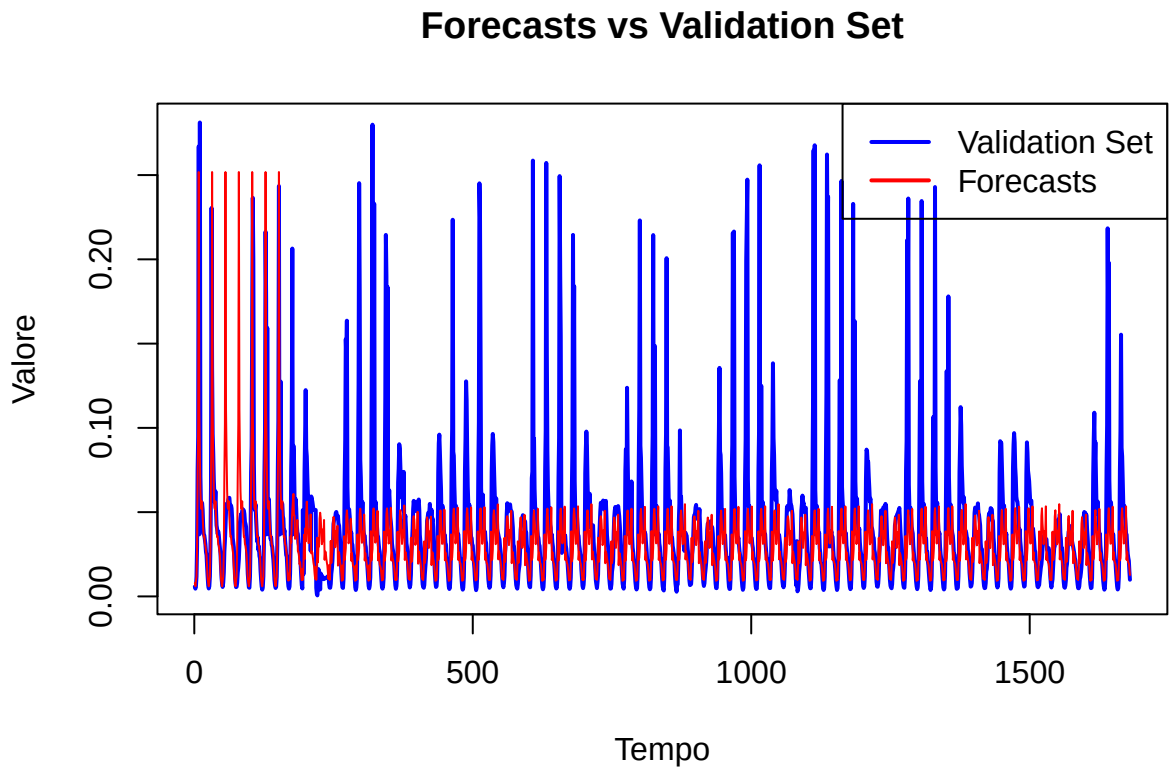
```
## MAE:  0.01960682
```

```r
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



XGBoost

```r
# Carica il pacchetto xgboost
library(xgboost)

# Lista per memorizzare i MAE per ogni serie temporale
mae_values <- numeric(length = 24)

# Liste per memorizzare i valori di training, forecast e residui
test_values_list <- list()
forecast_values_list <- list()
residuals_list <- list()

# Ciclo per le 24 serie temporali
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test
```

```r
# Sistemazione dei valori mancanti nelle lag
train_series$Lag_X7[1] <- train_series$Lag_X7[2]
train_series$Diff_X[1] <- train_series$Diff_X[2]
train_series$Diff_X7[1] <- train_series$Diff_X7[2]

# Preparazione dei dati per XGBoost
train_matrix <- as.matrix(subset(train_series, select = -c(DateTime, Date, X)))
train_labels <- train_series$X

# Parametri di XGBoost
xgb_params <- list(
  objective = "reg:squarederror",
  max_depth = 6,
  eta = 0.1,
  nthread = 2,
  verbosity = 0
)

# Addestramento del modello XGBoost
xgb_model <- xgboost(
  data = train_matrix,
  label = train_labels,
  params = xgb_params,
  nrounds = 100,
  verbose = FALSE
)

# Inizializza il vettore per le previsioni rolling
z_hatr <- numeric(nrow(test_series))

for (h in 1:nrow(test_series)) {
  # Crea una nuova riga con la previsione corrente
  new_row <- data.frame(
    DateTime = test_series$DateTime[h],
    Date = as.character(test_series$Date[h]),
    Hour = test_series$Hour[h],
    X = z_hatr[h],  # La previsione corrente
    Lag_X = ifelse(h > 1, z_hatr[h-1], NA),
    Lag_X7 = ifelse(h > 7, z_hatr[h-7], NA),
    Diff_X = ifelse(h > 1, z_hatr[h] - z_hatr[h-1], NA),
    Diff_X7 = ifelse(h > 7, z_hatr[h] - z_hatr[h-7], NA),
    DayOfWeek = test_series$DayOfWeek[h],
    DayOfYear = test_series$DayOfYear[h],
    stringsAsFactors = FALSE
  )

  # Aggiungi la nuova riga al dataset di addestramento
  train_data_with_pred <- rbind(train_series, new_row)

  # Rimuovi eventuali righe con NA
  train_data_with_pred_clean <- na.omit(train_data_with_pred)

  # Preparazione del dataset per la previsione
```

```
    pred_matrix <- as.matrix(subset(train_data_with_pred_clean, select = -c(DateTime, Date, X)))

    # Previsione con il nuovo dataset di allenamento aggiornato
    z_hatr[h] <- predict(xgb_model, newdata = pred_matrix[nrow(pred_matrix), , drop = FALSE])
  }

  # Salva i valori di training e previsione nella lista
  test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
  forecast_values_list[[i]] <- z_hatr  # Salva le previsioni

  # Calcola MAE tra le previsioni e i valori reali
  mae_values[i] <- mean(abs(z_hatr - test_series$X), na.rm = TRUE)

  # Calcola i residui
  residuals_list[[i]] <- test_series$X - z_hatr
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

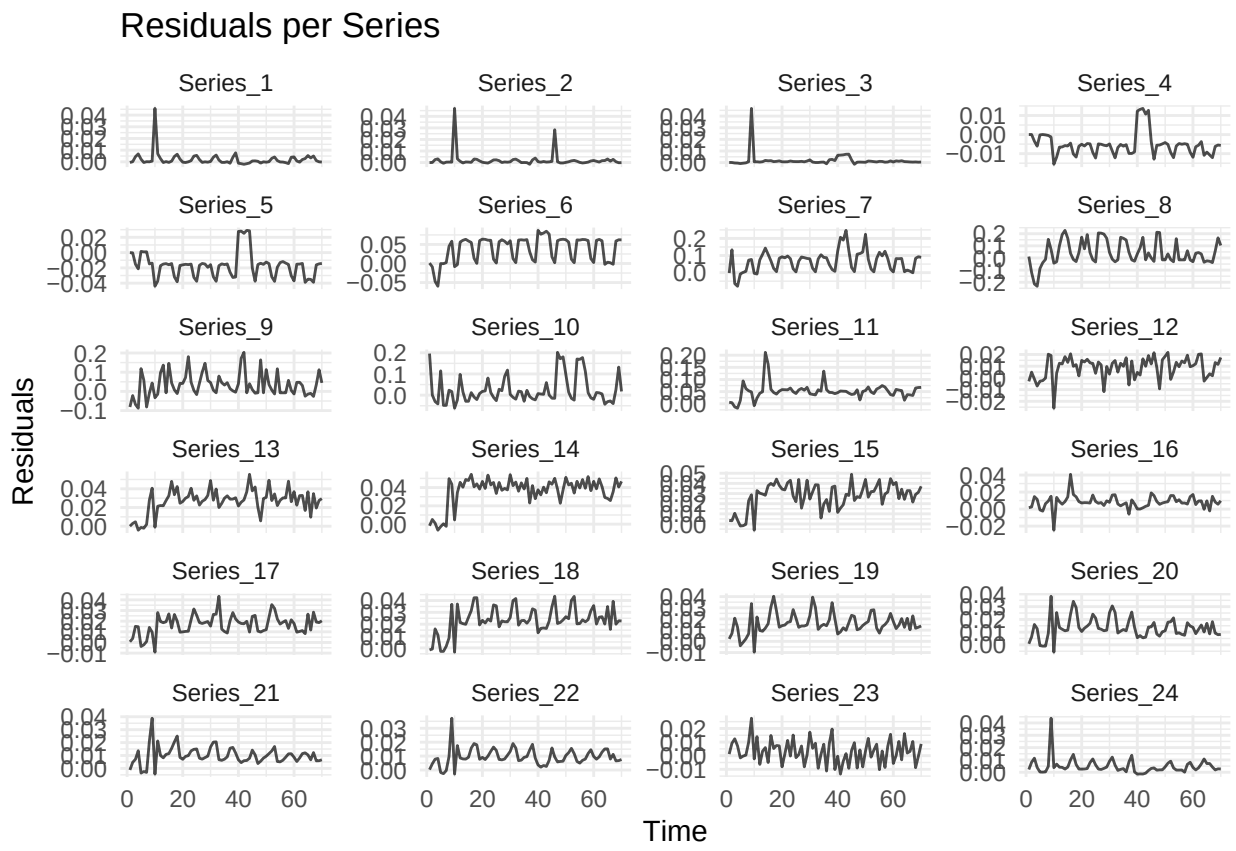
```
print(mae_values)
```

```
##  [1] 0.002540465 0.002134507 0.001970631 0.006954768 0.020901753 0.041511897
##  [7] 0.071470493 0.079723667 0.050593608 0.046351073 0.052680484 0.010941856
## [13] 0.027171122 0.036974257 0.027891236 0.009328398 0.017417489 0.023794717
## [19] 0.016416394 0.013678111 0.010435418 0.009828177 0.007193089 0.005014431
```

```
cat("MAE: ", mae, "\n")
```

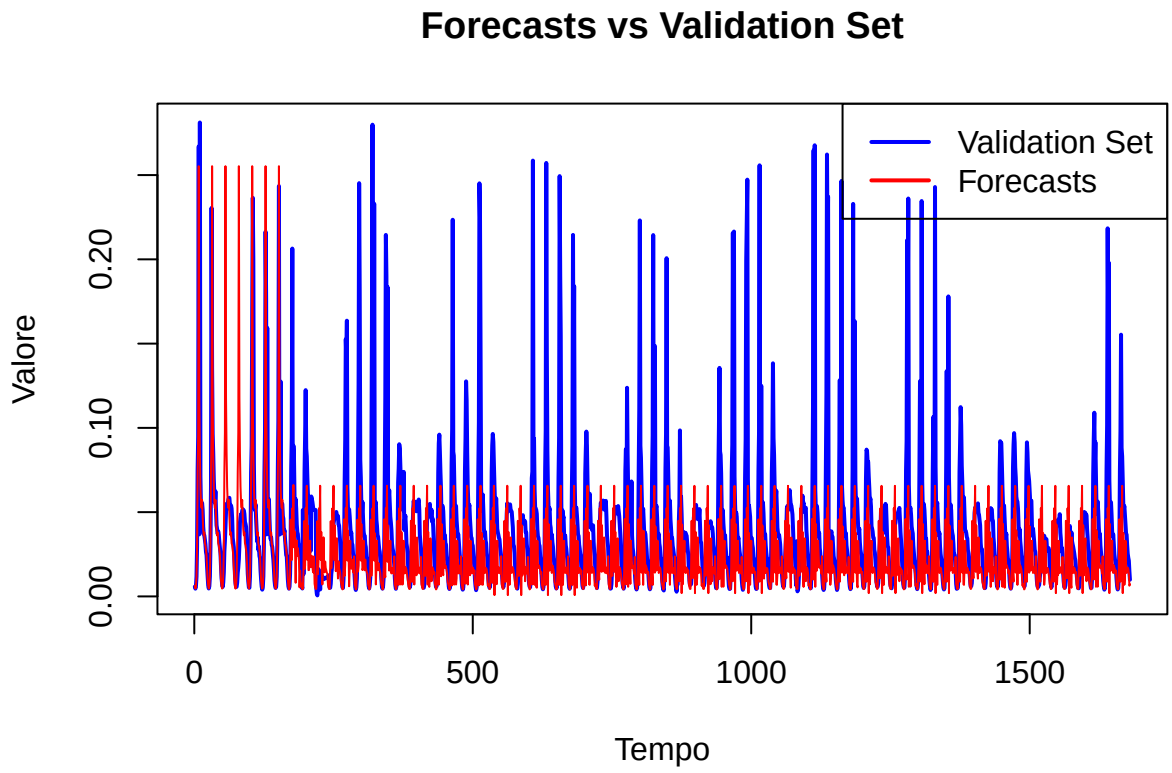
```
## MAE:  0.02470492
```

```
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



K-Nearest Neighbors (KNN)

```r
# Carica il pacchetto FNN
library(FNN)

# Lista per memorizzare i MAE per ogni serie temporale
mae_values <- numeric(length = 24)

# Liste per memorizzare i valori di training, forecast e residui
test_values_list <- list()
forecast_values_list <- list()
residuals_list <- list()

# Ciclo per le 24 serie temporali
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series[[i]]$train
  test_series <- train_val_series[[i]]$test
```

```r
# Sistemazione dei valori mancanti nelle lag
train_series$Lag_X7[1] <- train_series$Lag_X7[2]
train_series$Diff_X[1] <- train_series$Diff_X[2]
train_series$Diff_X7[1] <- train_series$Diff_X7[2]

# Preparazione dei dati per KNN (escludendo colonne non numeriche come DateTime e Date)
train_matrix <- as.matrix(subset(train_series, select = -c(DateTime, Date, X)))
train_labels <- train_series$X

# Inizializza il vettore per le previsioni rolling
z_hatr <- numeric(nrow(test_series))

for (h in 1:nrow(test_series)) {
  # Crea una nuova riga con la previsione corrente
  new_row <- data.frame(
    DateTime = test_series$DateTime[h],
    Date = as.character(test_series$Date[h]),
    Hour = test_series$Hour[h],
    X = z_hatr[h],  # La previsione corrente
    Lag_X = ifelse(h > 1, z_hatr[h-1], NA),
    Lag_X7 = ifelse(h > 7, z_hatr[h-7], NA),
    Diff_X = ifelse(h > 1, z_hatr[h] - z_hatr[h-1], NA),
    Diff_X7 = ifelse(h > 7, z_hatr[h] - z_hatr[h-7], NA),
    DayOfWeek = test_series$DayOfWeek[h],
    DayOfYear = test_series$DayOfYear[h],
    stringsAsFactors = FALSE
  )

  # Aggiungi la nuova riga al dataset di addestramento
  train_data_with_pred <- rbind(train_series, new_row)

  # Rimuovi eventuali righe con NA
  train_data_with_pred_clean <- na.omit(train_data_with_pred)

  # Preparazione del dataset per la previsione
  pred_matrix <- as.matrix(subset(train_data_with_pred_clean, select = -c(DateTime, Date, X)))

  # Applica il modello KNN per la previsione
  knn_prediction <- knn.reg(train = train_matrix, test = pred_matrix[nrow(pred_matrix), , drop = FALSE

  # Salva la previsione corrente
  z_hatr[h] <- knn_prediction$pred
}

# Salva i valori di training e previsione nella lista
test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
forecast_values_list[[i]] <- z_hatr  # Salva le previsioni

# Calcola MAE tra le previsioni e i valori reali
mae_values[i] <- mean(abs(z_hatr - test_series$X), na.rm = TRUE)

# Calcola i residui
residuals_list[[i]] <- test_series$X - z_hatr
```

```
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

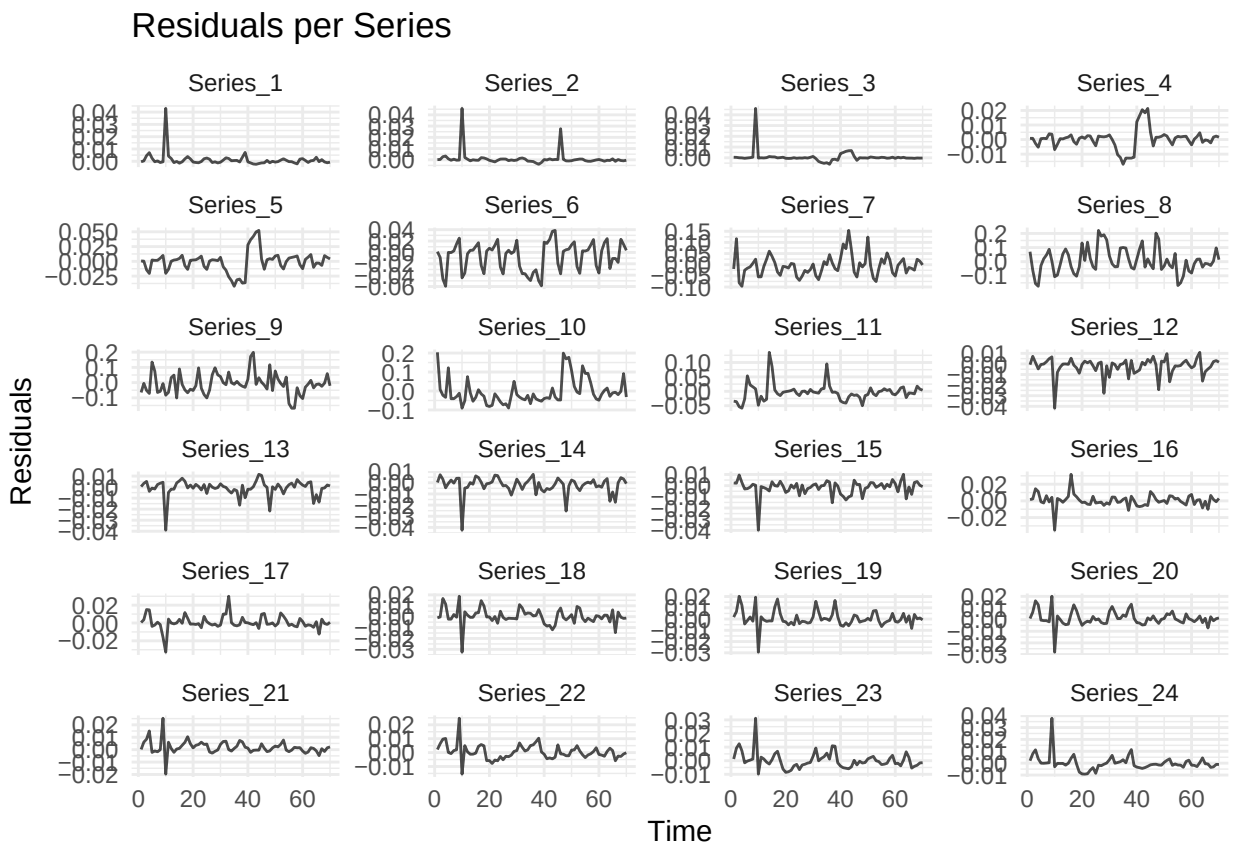
```
print(mae_values)
```

```
##  [1] 0.002074571 0.001918000 0.001804286 0.004381714 0.011961429 0.020028857
##  [7] 0.035223714 0.068924857 0.053049143 0.050837714 0.019225857 0.005154857
## [13] 0.004290000 0.004700571 0.004195429 0.004880143 0.004653857 0.004694857
## [19] 0.004709143 0.004068857 0.003468857 0.003750571 0.003995143 0.003426571
```

```
cat("MAE: ", mae, "\n")
```

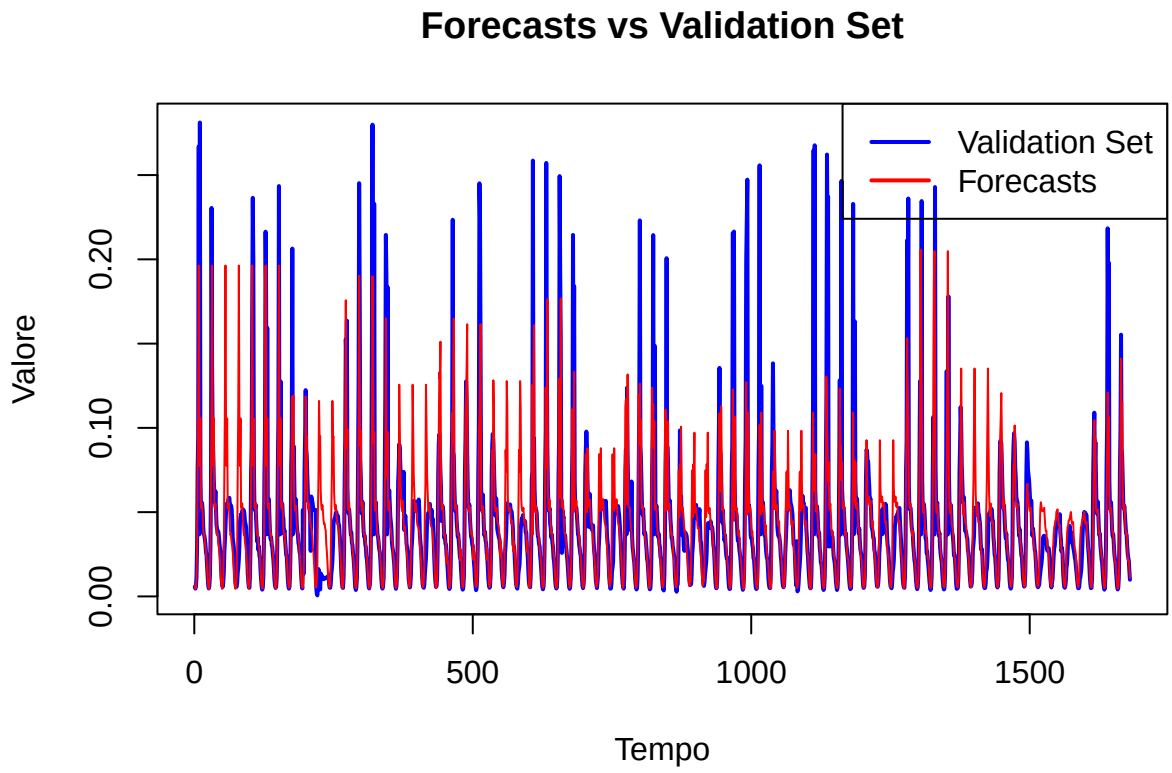
```
## MAE:  0.01355912
```

```
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



Dummies

Random Forest

```r
# Lista per memorizzare i MAE per ogni serie temporale
mae_values <- numeric(length = 24)

# Liste per memorizzare i valori di training, forecast e residui
test_values_list <- list()
forecast_values_list <- list()
residuals_list <- list()

# Ciclo per le 24 serie temporali
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series_dummy[[i]]$train
  test_series <- train_val_series_dummy[[i]]$test
```

```r
train_series$Lag_X7[1] <- train_series$Lag_X7[2]
train_series$Diff_X[1] <- train_series$Diff_X[2]
train_series$Diff_X7[1] <- train_series$Diff_X7[2]

# Crea il modello Random Forest (o il modello che stai utilizzando)
rf_model <- randomForest(X ~ ., data = train_series, importance = TRUE)

# Inizializza il vettore per le previsioni rolling
z_hatr <- numeric(nrow(test_series))

for (h in 1:nrow(test_series)) {
  # Crea una nuova riga con la previsione corrente
  new_row <- data.frame(
    DateTime = test_series$DateTime[h],
    Date = as.character(test_series$Date[h]),
    Hour = test_series$Hour[h],
    X = z_hatr[h],  # La previsione corrente
    Lag_X = ifelse(h > 1, z_hatr[h-1], NA),
    Lag_X7 = ifelse(h > 7, z_hatr[h-7], NA),
    Diff_X = ifelse(h > 1, z_hatr[h] - z_hatr[h-1], NA),
    Diff_X7 = ifelse(h > 7, z_hatr[h] - z_hatr[h-7], NA),
    DayOfWeek = test_series$DayOfWeek[h],
    DayOfYear = test_series$DayOfYear[h],
    IsWeekend = test_series$IsWeekend[h],
    IsHoliday = test_series$IsHoliday[h],
    Season = test_series$Season[h],
    Dec24 = test_series$Dec24[h],
    Dec25 = test_series$Dec25[h],
    Dec26 = test_series$Dec26[h],
    Jan1 = test_series$Jan1[h],
    Jan6 = test_series$Jan6[h],
    EasterSat = test_series$EasterSat[h],
    Easter = test_series$Easter[h],
    EasterMon = test_series$EasterMon[h],
    EasterTue = test_series$EasterTue[h],
    Aug15 = test_series$Aug15[h],
    EndYear = test_series$EndYear[h],
    Valentine = test_series$Valentine[h],
    stringsAsFactors = FALSE
  )

  # Aggiungi la nuova riga al dataset di allenamento
  train_data_with_pred <- rbind(train_series, new_row)

  # Rimuovi eventuali righe con NA
  train_data_with_pred_clean <- na.omit(train_data_with_pred)

  # Previsione con il nuovo dataset di allenamento aggiornato
  z_hatr[h] <- predict(rf_model, newdata = train_data_with_pred_clean[nrow(train_data_with_pred_clean)
}

# Salva i valori di training e previsione nella lista
test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
```

```
  forecast_values_list[[i]] <- z_hatr  # Salva le previsioni

  # Calcola MAE tra le previsioni e i valori reali
  mae_values[i] <- mean(abs(z_hatr - test_series$X), na.rm = TRUE)

  # Calcola i residui
  residuals_list[[i]] <- test_series$X - z_hatr
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

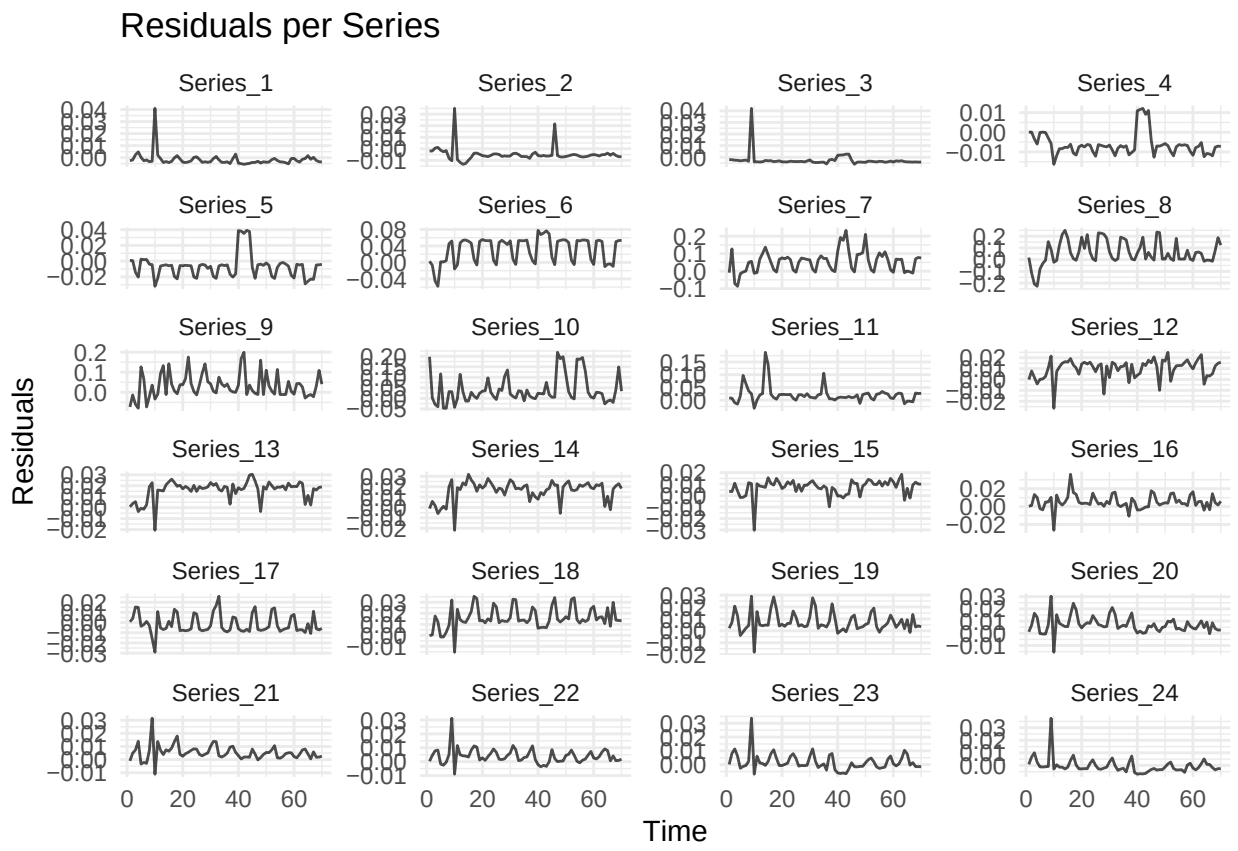
```
print(mae_values)
```

```
##  [1] 0.003320309 0.006480261 0.004033017 0.007837991 0.012324973 0.036619111
##  [7] 0.061376523 0.084174012 0.048183153 0.050759239 0.023894479 0.011662911
## [13] 0.016808970 0.017250195 0.008697250 0.007037790 0.008121311 0.016378112
## [19] 0.008200902 0.007662778 0.005876616 0.004679261 0.003915247 0.003566001
```

```
cat("MAE: ", mae, "\n")
```

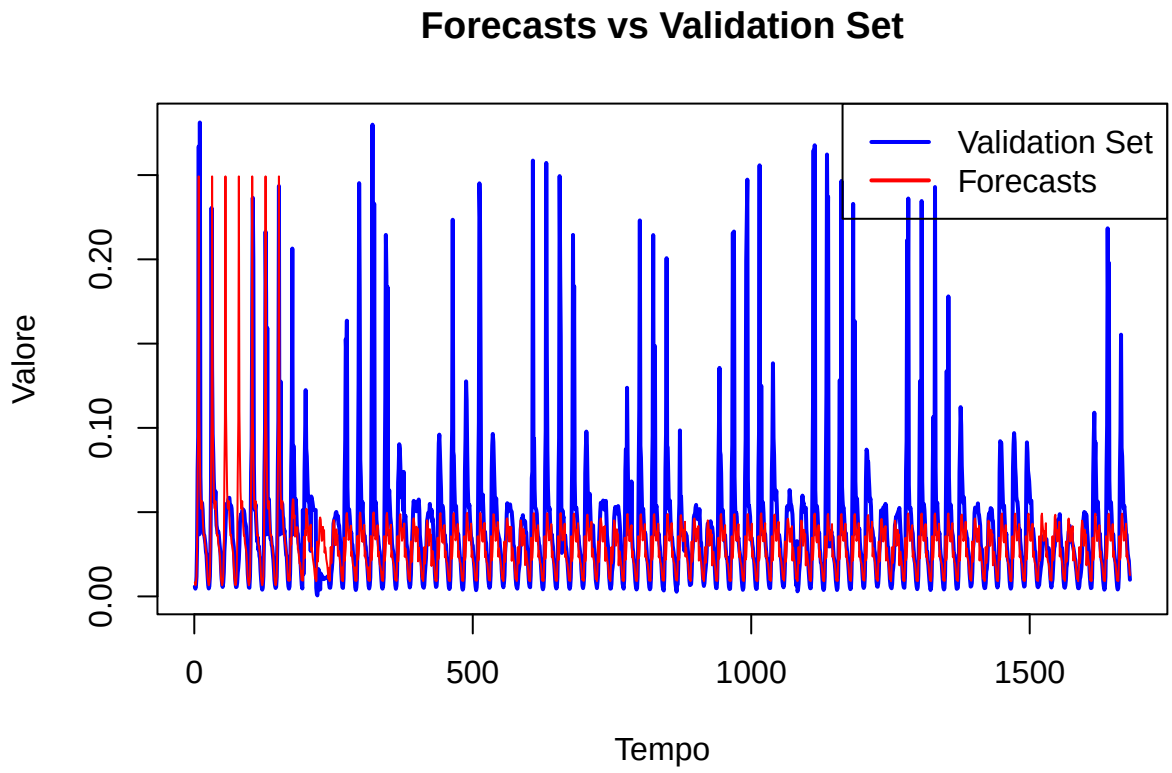
```
## MAE:  0.01911918
```

```
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



XGBoost

```r
# Lista per memorizzare i MAE per ogni serie temporale
mae_values <- numeric(length = 24)

# Liste per memorizzare i valori di training, forecast e residui
test_values_list <- list()
forecast_values_list <- list()
residuals_list <- list()

# Ciclo per le 24 serie temporali
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente
  train_series <- train_val_series_dummy[[i]]$train
  test_series <- train_val_series_dummy[[i]]$test

  # Trasformazione logaritmica per stabilizzare la varianza nei dati di allenamento
  train_series$X <- log(train_series$X + 1)  # Aggiungi 1 per evitare log(0)
```

```r
train_series$Lag_X <- log(train_series$Lag_X + 1)
train_series$Lag_X7 <- log(train_series$Lag_X7 + 1)
train_series$Lag_X7[1] <- train_series$Lag_X7[2]
train_series$Diff_X <- log(train_series$Diff_X + 1)
train_series$Diff_X[1] <- train_series$Diff_X[2]
train_series$Diff_X7 <- log(train_series$Diff_X7 + 1)
train_series$Diff_X7[1] <- train_series$Diff_X7[2]

# Crea il modello Random Forest (o il modello che stai utilizzando)
rf_model <- randomForest(X ~ ., data = train_series, importance = TRUE)

# Inizializza il vettore per le previsioni rolling
z_hatr <- numeric(nrow(test_series))

for (h in 1:nrow(test_series)) {
  # Crea una nuova riga con la previsione corrente
  new_row <- data.frame(
    DateTime = test_series$DateTime[h],
    Date = as.character(test_series$Date[h]),
    Hour = test_series$Hour[h],
    X = z_hatr[h],  # La previsione corrente
    Lag_X = ifelse(h > 1, z_hatr[h-1], NA),
    Lag_X7 = ifelse(h > 7, z_hatr[h-7], NA),
    Diff_X = ifelse(h > 1, z_hatr[h] - z_hatr[h-1], NA),
    Diff_X7 = ifelse(h > 7, z_hatr[h] - z_hatr[h-7], NA),
    DayOfWeek = test_series$DayOfWeek[h],
    DayOfYear = test_series$DayOfYear[h],
    IsWeekend = test_series$IsWeekend[h],
    IsHoliday = test_series$IsHoliday[h],
    Season = test_series$Season[h],
    Dec24 = test_series$Dec24[h],
    Dec25 = test_series$Dec25[h],
    Dec26 = test_series$Dec26[h],
    Jan1 = test_series$Jan1[h],
    Jan6 = test_series$Jan6[h],
    EasterSat = test_series$EasterSat[h],
    Easter = test_series$Easter[h],
    EasterMon = test_series$EasterMon[h],
    EasterTue = test_series$EasterTue[h],
    Aug15 = test_series$Aug15[h],
    EndYear = test_series$EndYear[h],
    Valentine = test_series$Valentine[h],
    stringsAsFactors = FALSE
  )

  # Aggiungi la nuova riga al dataset di allenamento
  train_data_with_pred <- rbind(train_series, new_row)

  # Rimuovi eventuali righe con NA
  train_data_with_pred_clean <- na.omit(train_data_with_pred)

  # Previsione con il nuovo dataset di allenamento aggiornato
  z_hatr[h] <- predict(rf_model, newdata = train_data_with_pred_clean[nrow(train_data_with_pred_clean
```

```
  }

  # Inverso della trasformazione logaritmica per riportare le previsioni alla scala originale
  z_hatr_original <- exp(z_hatr) - 1  # Inverti il logaritmo (assumendo che la trasformazione sia log(x

  # Salva i valori di training e previsione nella lista
  test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
  forecast_values_list[[i]] <- z_hatr_original  # Salva le previsioni

  # Calcola MAE tra le previsioni e i valori reali sulla scala originale
  mae_values[i] <- mean(abs(z_hatr_original - test_series$X), na.rm = TRUE)

  # Calcola i residui sulla scala originale
  residuals_list[[i]] <- test_series$X - z_hatr_original
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

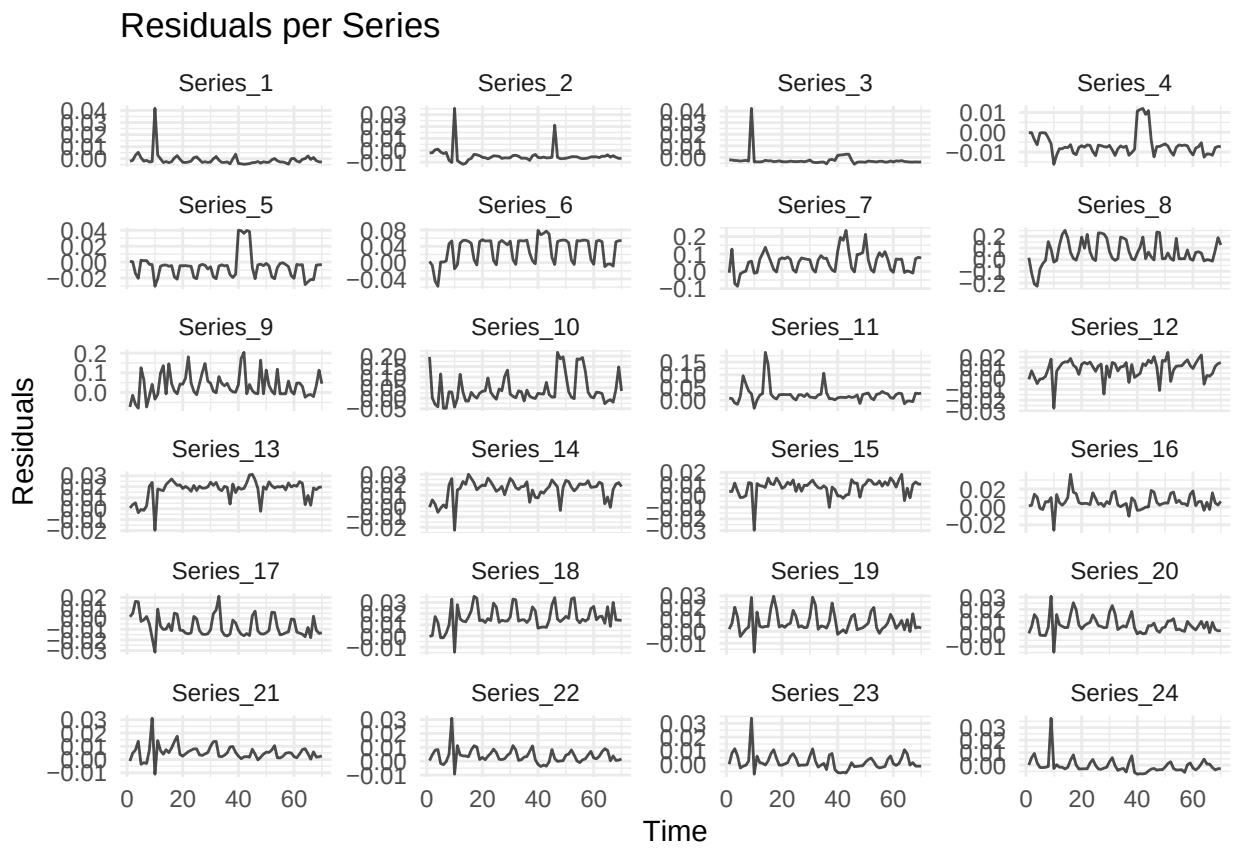
```
print(mae_values)
```

```
##  [1] 0.002914593 0.006397579 0.003890243 0.007846446 0.011544210 0.036815957
##  [7] 0.062628429 0.085658452 0.049981482 0.050148562 0.023987325 0.011023561
## [13] 0.017771878 0.016389242 0.008659298 0.007373137 0.010508572 0.016997280
## [19] 0.008307908 0.007969671 0.005861029 0.004511767 0.003852504 0.003713071
```

```
cat("MAE: ", mae, "\n")
```

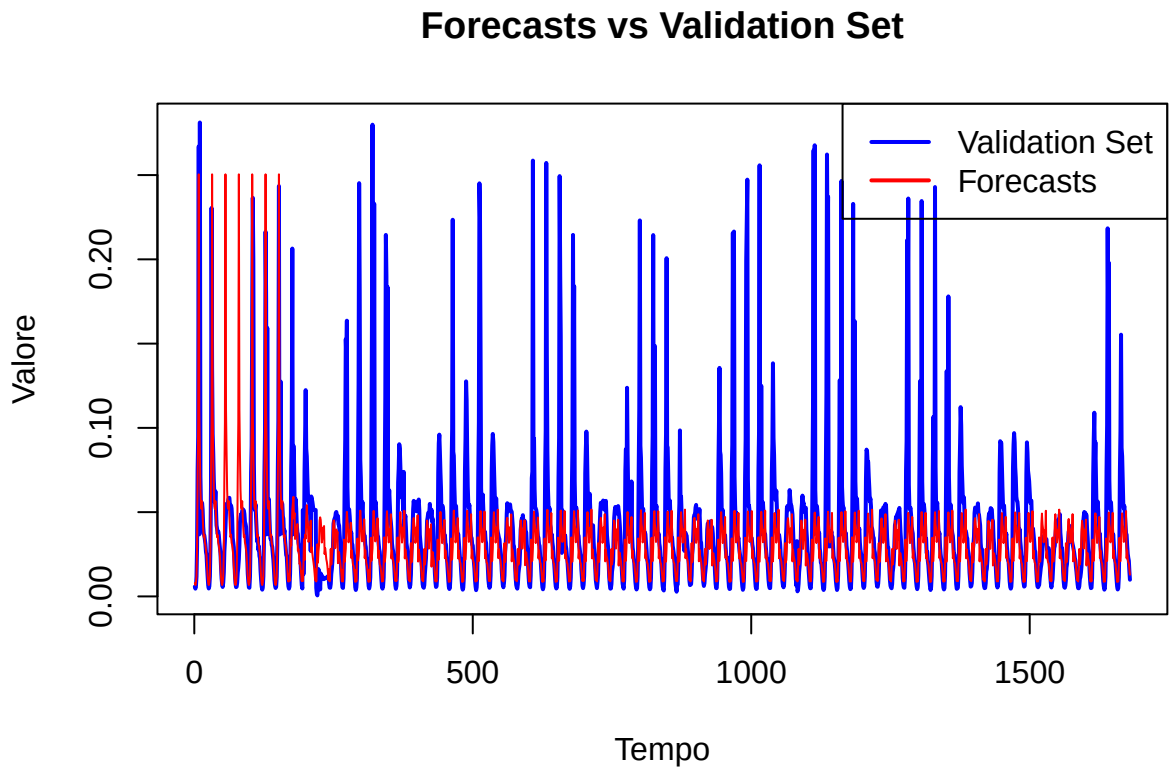
```
## MAE:  0.01936467
```

```
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```r
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)

# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)
```

```r
# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



K-Nearest Neighbors (KNN)

```r
# Carica il pacchetto FNN
library(FNN)

# Lista per memorizzare i MAE per ogni serie temporale
mae_values <- numeric(length = 24)

# Liste per memorizzare i valori di training, forecast e residui
test_values_list <- list()
forecast_values_list <- list()
residuals_list <- list()

# Ciclo per le 24 serie temporali
for (i in 1:24) {
  # Prendi la serie temporale di allenamento e test per la serie corrente con le colonne dummy
  train_series <- train_val_series_dummy[[i]]$train
```

```r
test_series <- train_val_series_dummy[[i]]$test

# Sistemazione dei valori mancanti nelle lag
train_series$Lag_X7[1] <- train_series$Lag_X7[2]
train_series$Diff_X[1] <- train_series$Diff_X[2]
train_series$Diff_X7[1] <- train_series$Diff_X7[2]

# Converti le colonne categoriali in numeriche (ad esempio, `Season`)
train_series$Season <- as.numeric(as.factor(train_series$Season))
test_series$Season <- as.numeric(as.factor(test_series$Season))

# Preparazione dei dati per KNN (escludendo colonne non numeriche come DateTime e Date)
train_matrix <- as.matrix(subset(train_series, select = -c(DateTime, Date, X)))
train_labels <- train_series$X

# Inizializza il vettore per le previsioni rolling
z_hatr <- numeric(nrow(test_series))

for (h in 1:nrow(test_series)) {
  # Crea una nuova riga con la previsione corrente
  new_row <- data.frame(
    DateTime = test_series$DateTime[h],
    Date = as.character(test_series$Date[h]),
    Hour = test_series$Hour[h],
    X = z_hatr[h],  # La previsione corrente
    Lag_X = ifelse(h > 1, z_hatr[h-1], NA),
    Lag_X7 = ifelse(h > 7, z_hatr[h-7], NA),
    Diff_X = ifelse(h > 1, z_hatr[h] - z_hatr[h-1], NA),
    Diff_X7 = ifelse(h > 7, z_hatr[h] - z_hatr[h-7], NA),
    DayOfWeek = test_series$DayOfWeek[h],
    DayOfYear = test_series$DayOfYear[h],
    IsWeekend = test_series$IsWeekend[h],
    IsHoliday = test_series$IsHoliday[h],
    Season = as.numeric(as.factor(test_series$Season[h])),
    Dec24 = test_series$Dec24[h],
    Dec25 = test_series$Dec25[h],
    Dec26 = test_series$Dec26[h],
    Jan1 = test_series$Jan1[h],
    Jan6 = test_series$Jan6[h],
    EasterSat = test_series$EasterSat[h],
    Easter = test_series$Easter[h],
    EasterMon = test_series$EasterMon[h],
    EasterTue = test_series$EasterTue[h],
    Aug15 = test_series$Aug15[h],
    EndYear = test_series$EndYear[h],
    Valentine = test_series$Valentine[h],
    stringsAsFactors = FALSE
  )

  # Aggiungi la nuova riga al dataset di addestramento
  train_data_with_pred <- rbind(train_series, new_row)

  # Rimuovi eventuali righe con NA
```

```
    train_data_with_pred_clean <- na.omit(train_data_with_pred)

    # Preparazione del dataset per la previsione
    pred_matrix <- as.matrix(subset(train_data_with_pred_clean, select = -c(DateTime, Date, X)))

    # Applica il modello KNN per la previsione
    knn_prediction <- knn.reg(train = train_matrix, test = pred_matrix[nrow(pred_matrix), , drop = FALSE

    # Salva la previsione corrente
    z_hatr[h] <- knn_prediction$pred
  }

  # Salva i valori di training e previsione nella lista
  test_values_list[[i]] <- test_series$X  # Salva la serie di allenamento
  forecast_values_list[[i]] <- z_hatr  # Salva le previsioni

  # Calcola MAE tra le previsioni e i valori reali
  mae_values[i] <- mean(abs(z_hatr - test_series$X), na.rm = TRUE)

  # Calcola i residui
  residuals_list[[i]] <- test_series$X - z_hatr
}

# Media dei MAE di tutte le serie
mae <- mean(mae_values)

# Visualizza i risultati
cat("MAE values per series:\n")
```

```
## MAE values per series:
```

```
print(mae_values)
```

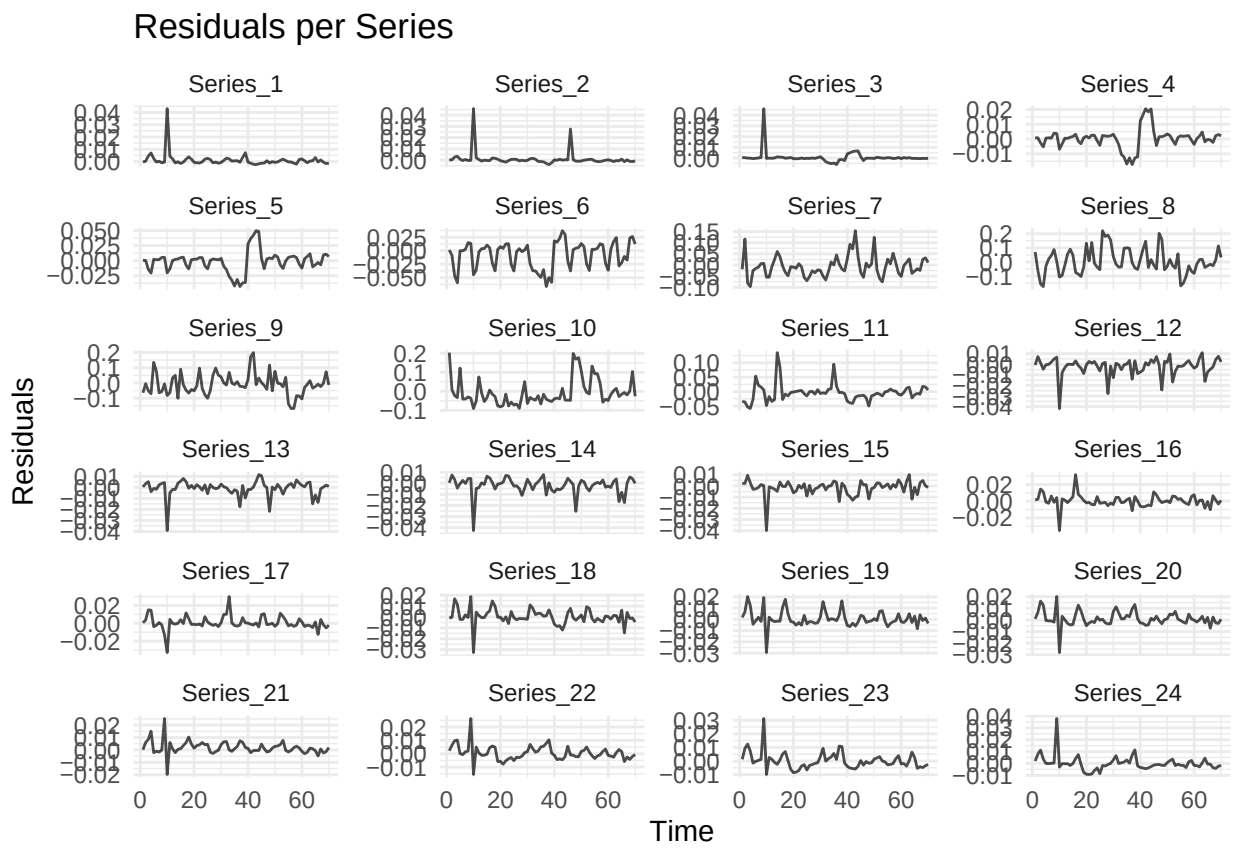
```
##  [1] 0.002086286 0.001934857 0.001804857 0.004456857 0.011904857 0.019927429
##  [7] 0.035465714 0.070131143 0.053858571 0.053410000 0.019692143 0.005223714
## [13] 0.004298571 0.004715429 0.004205143 0.004849286 0.004577857 0.004715714
## [19] 0.004762857 0.004126571 0.003506571 0.003797143 0.004050000 0.003467714
```

```
cat("MAE: ", mae, "\n")
```

```
## MAE:  0.01379039
```

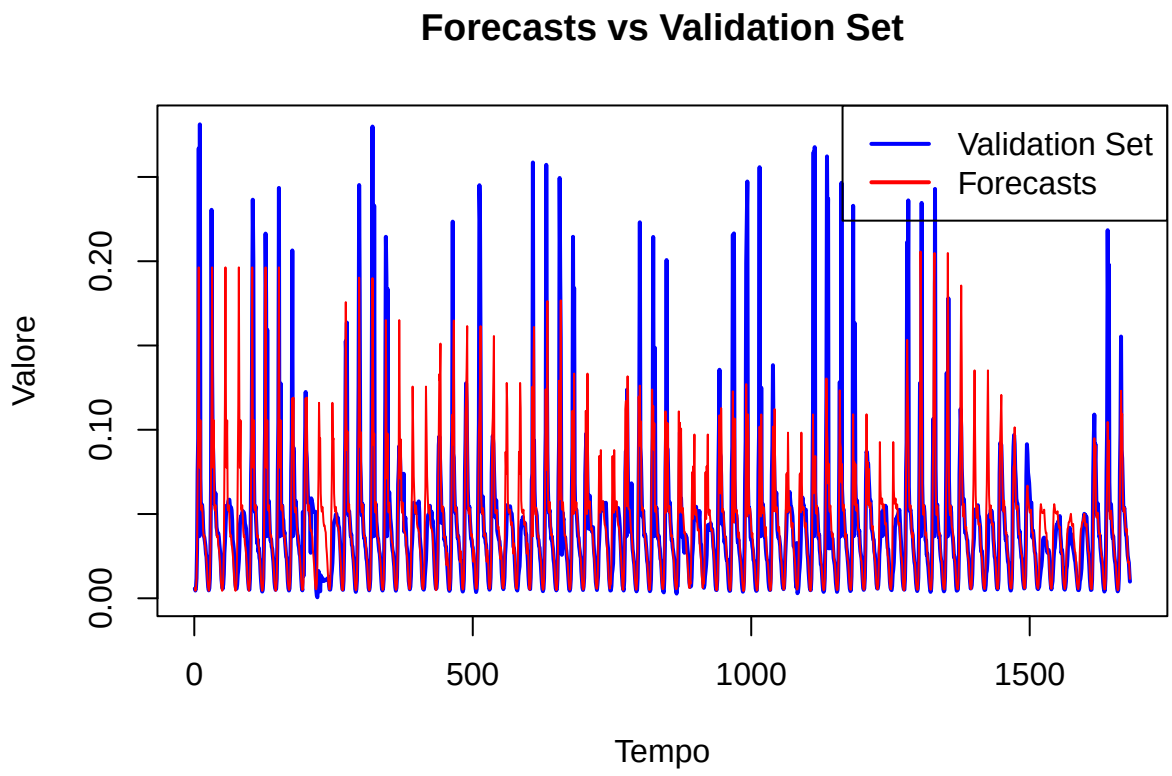
```
library(ggplot2)
library(reshape2)

# Combina i residui in un unico data frame
residuals_df <- do.call(cbind, residuals_list)
colnames(residuals_df) <- paste0("Series_", 1:24)
residuals_df <- data.frame(Time = seq_len(nrow(residuals_df)), residuals_df)
residuals_long <- melt(residuals_df, id.vars = "Time", variable.name = "Series", value.name = "Residual

# Crea il grafico con ggplot2
ggplot(residuals_long, aes(x = Time, y = Residual)) +
```

```
  geom_line(alpha = 0.7) +
  facet_wrap(~ Series, scales = "free_y", ncol = 4) +
  theme_minimal() +
  labs(title = "Residuals per Series", x = "Time", y = "Residuals")
```




```
# Numero di serie temporali e lunghezza di ogni serie
num_series <- length(forecast_values_list)
series_length <- length(forecast_values_list[[1]])

# Inizializza vettori per le previsioni e i dati di test combinati
combined_forecasts <- numeric(series_length * num_series)
combined_test <- numeric(series_length * num_series)

# Ricostruisci le serie temporali combinate ordinando i dati correttamente
for (t in 1:series_length) {
  for (i in 1:num_series) {
    index <- (t - 1) * num_series + i
    combined_forecasts[index] <- forecast_values_list[[i]][t]
    combined_test[index] <- train_val_series[[i]]$test$X[t]
  }
}

# Ottieni i valori di start_date e frequency dalla prima serie di test
start_date <- start(train_val_series[[1]]$test$X)  # Data di inizio della prima serie test
frequency <- frequency(train_val_series[[1]]$test$X)  # Frequenza della serie (e.g., giornaliera)
```

```r
# Convertili in oggetti di serie temporali
forecast_ts <- ts(combined_forecasts, start = start_date, frequency = frequency)
test_ts <- ts(combined_test, start = start_date, frequency = frequency)

# Grafico delle serie temporali
plot(test_ts, col = "blue", lwd = 2, ylim = range(c(test_ts, forecast_ts)),
     main = "Forecasts vs Validation Set", ylab = "Valore", xlab = "Tempo")
lines(forecast_ts, col = "red", lwd = 1)  # Linea rossa continua per le previsioni
legend("topright", legend = c("Validation Set", "Forecasts"),
       col = c("blue", "red"), lty = c(1, 1), lwd = 2)  # Linee continue nella legenda
```



Deployment final model

```r
# Carica il pacchetto necessario
library(FNN)

# Liste per memorizzare i valori di training, forecast e residui
test_values_list <- list()
forecast_results_KNN <- list()  # Lista per le previsioni
residuals_list <- list()  # Lista per i residui

# Loop per le 24 serie temporali
for (i in 1:24) {
  print(paste("Processing series:", i))
```

```r
# Prendi la serie temporale di allenamento e test per la serie corrente
train_series <- train_test_series_dummy[[i]]$train
test_series <- train_test_series_dummy[[i]]$test

# Sistemazione dei valori mancanti nelle lag
train_series$Lag_X7[1] <- train_series$Lag_X7[2]
train_series$Diff_X[1] <- train_series$Diff_X[2]
train_series$Diff_X7[1] <- train_series$Diff_X7[2]

# Converti colonne categoriali in numeriche (ad esempio `Season`)
train_series$Season <- as.numeric(as.factor(train_series$Season))
test_series$Season <- as.numeric(as.factor(test_series$Season))

# Preparazione dei dati per KNN (escludendo colonne non numeriche come DateTime e Date)
train_matrix <- as.matrix(subset(train_series, select = -c(DateTime, Date, X)))
train_labels <- train_series$X

# Inizializza il vettore per le previsioni rolling
z_hatr <- numeric(nrow(test_series))

for (h in 1:nrow(test_series)) {
  # Prepara la nuova riga con la previsione corrente
  new_row <- data.frame(
    DateTime = test_series$DateTime[h],
    Date = as.character(test_series$Date[h]),
    Hour = test_series$Hour[h],
    X = z_hatr[h],  # Previsione corrente
    Lag_X = ifelse(h > 1, z_hatr[h-1], NA),
    Lag_X7 = ifelse(h > 7, z_hatr[h-7], NA),
    Diff_X = ifelse(h > 1, z_hatr[h] - z_hatr[h-1], NA),
    Diff_X7 = ifelse(h > 7, z_hatr[h] - z_hatr[h-7], NA),
    DayOfWeek = test_series$DayOfWeek[h],
    DayOfYear = test_series$DayOfYear[h],
    IsWeekend = test_series$IsWeekend[h],
    IsHoliday = test_series$IsHoliday[h],
    Season = test_series$Season[h],
    Dec24 = test_series$Dec24[h],
    Dec25 = test_series$Dec25[h],
    Dec26 = test_series$Dec26[h],
    Jan1 = test_series$Jan1[h],
    Jan6 = test_series$Jan6[h],
    EasterSat = test_series$EasterSat[h],
    Easter = test_series$Easter[h],
    EasterMon = test_series$EasterMon[h],
    EasterTue = test_series$EasterTue[h],
    Aug15 = test_series$Aug15[h],
    EndYear = test_series$EndYear[h],
    Valentine = test_series$Valentine[h],
    stringsAsFactors = FALSE
  )

  # Aggiungi la nuova riga al dataset di allenamento
  train_data_with_pred <- rbind(train_series, new_row)
```

```
    # Rimuovi eventuali righe con NA
    train_data_with_pred_clean <- na.omit(train_data_with_pred)

    # Preparazione del dataset per la previsione
    pred_matrix <- as.matrix(subset(train_data_with_pred_clean, select = -c(DateTime, Date, X)))

    # Applica il modello KNN per la previsione
    knn_prediction <- knn.reg(
      train = train_matrix,
      test = pred_matrix[nrow(pred_matrix), , drop = FALSE],
      y = train_labels,
      k = 5
    )

    # Salva la previsione corrente
    z_hatr[h] <- knn_prediction$pred
  }

  # Salva i valori di test e previsioni nella lista
  test_values_list[[i]] <- test_series$X
  forecast_results_KNN[[i]] <- z_hatr
}
```

```
## [1] "Processing series: 1"
## [1] "Processing series: 2"
## [1] "Processing series: 3"
## [1] "Processing series: 4"
## [1] "Processing series: 5"
## [1] "Processing series: 6"
## [1] "Processing series: 7"
## [1] "Processing series: 8"
## [1] "Processing series: 9"
## [1] "Processing series: 10"
## [1] "Processing series: 11"
## [1] "Processing series: 12"
## [1] "Processing series: 13"
## [1] "Processing series: 14"
## [1] "Processing series: 15"
## [1] "Processing series: 16"
## [1] "Processing series: 17"
## [1] "Processing series: 18"
## [1] "Processing series: 19"
## [1] "Processing series: 20"
## [1] "Processing series: 21"
## [1] "Processing series: 22"
## [1] "Processing series: 23"
## [1] "Processing series: 24"
```

```
# Riepilogo delle previsioni completate
cat("Previsioni completate per tutte le serie temporali con KNN.\n")
```

```
## Previsioni completate per tutte le serie temporali con KNN.
```

```r
forecast_results_KNN[[1]][1:4]
```

```
## [1] 0.00646 0.00646 0.00646 0.00646
```

```r
forecast_results_KNN[[2]][1:4]
```

```
## [1] 0.00436 0.00436 0.00436 0.00436
```

```r
forecast_results_KNN[[3]][1:4]
```

```
## [1] 0.00498 0.00498 0.00498 0.00498
```

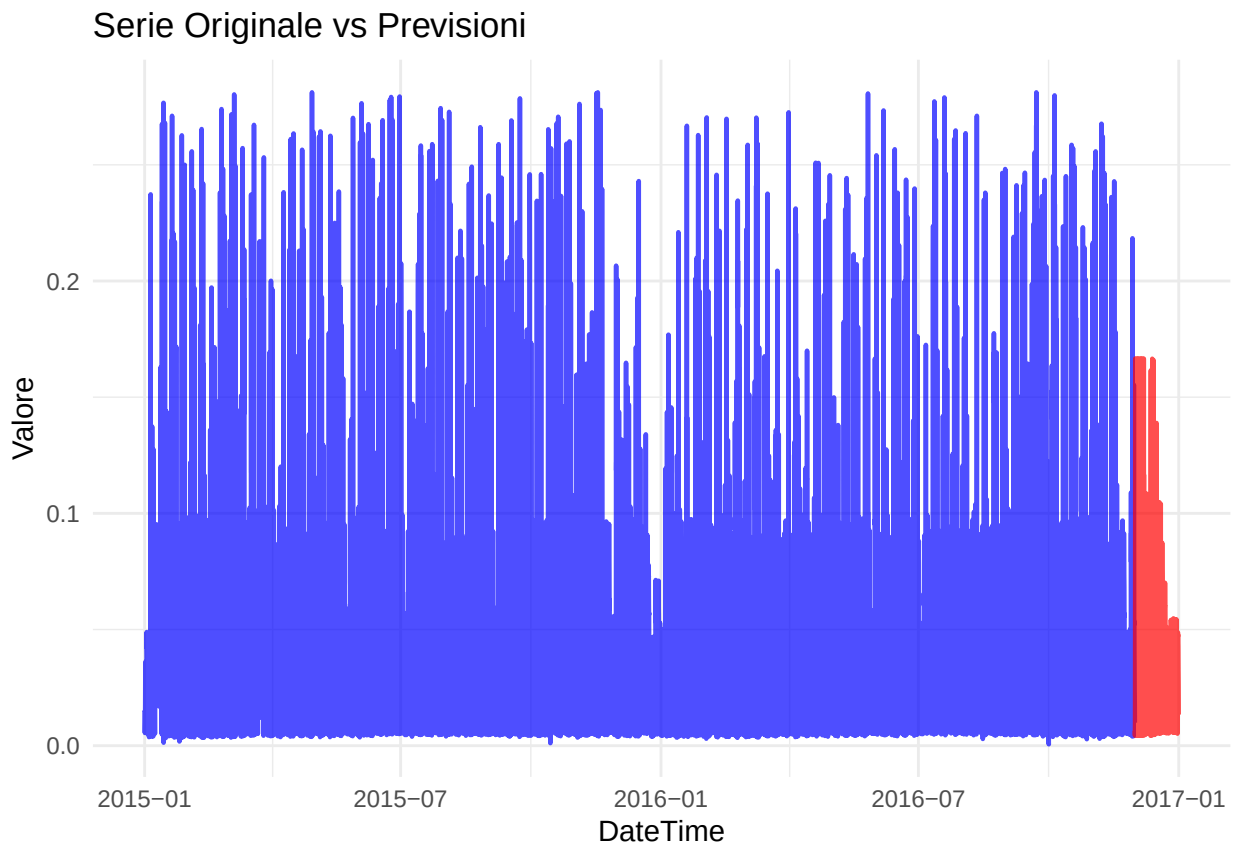
```r
# Creazione della base DateTime
start_date <- as.POSIXct("2016-12-01 00:00:00")
num_days <- length(forecast_results_KNN[[1]])  # Numero di giorni previsti
hourly_series_length <- num_days * 24  # Numero totale di ore

# Genera il vettore DateTime per ogni ora
datetime_vector <- seq(start_date, by = "hour", length.out = hourly_series_length)

# Inizializza un vettore per salvare tutte le previsioni
forecast_vector_KNN <- numeric(hourly_series_length)

# Ricomponi la serie oraria
for (i in 1:24) {
  # Inserisci le previsioni nella posizione corretta
  forecast_vector_KNN[seq(i, hourly_series_length, by = 24)] <- forecast_results_KNN[[i]]
}

# Creazione del data frame finale
forecast_dataframe <- data.frame(
  DateTime = datetime_vector,
  ARIMA = forecast_vector_ARIMA,
  UCM = forecast_vector_UCM,
  ML = forecast_vector_KNN
)

# Visualizza le prime righe del risultato
head(forecast_dataframe)
```

```
##              DateTime       ARIMA         UCM      ML
## 1 2016-12-01 00:00:00 0.005836121 0.006095668 0.00646
## 2 2016-12-01 01:00:00 0.004403759 0.004434946 0.00436
## 3 2016-12-01 02:00:00 0.005211713 0.005034438 0.00498
## 4 2016-12-01 03:00:00 0.011759774 0.011392291 0.01066
## 5 2016-12-01 04:00:00 0.030413454 0.028460888 0.02972
## 6 2016-12-01 05:00:00 0.068222103 0.049397717 0.07338
```

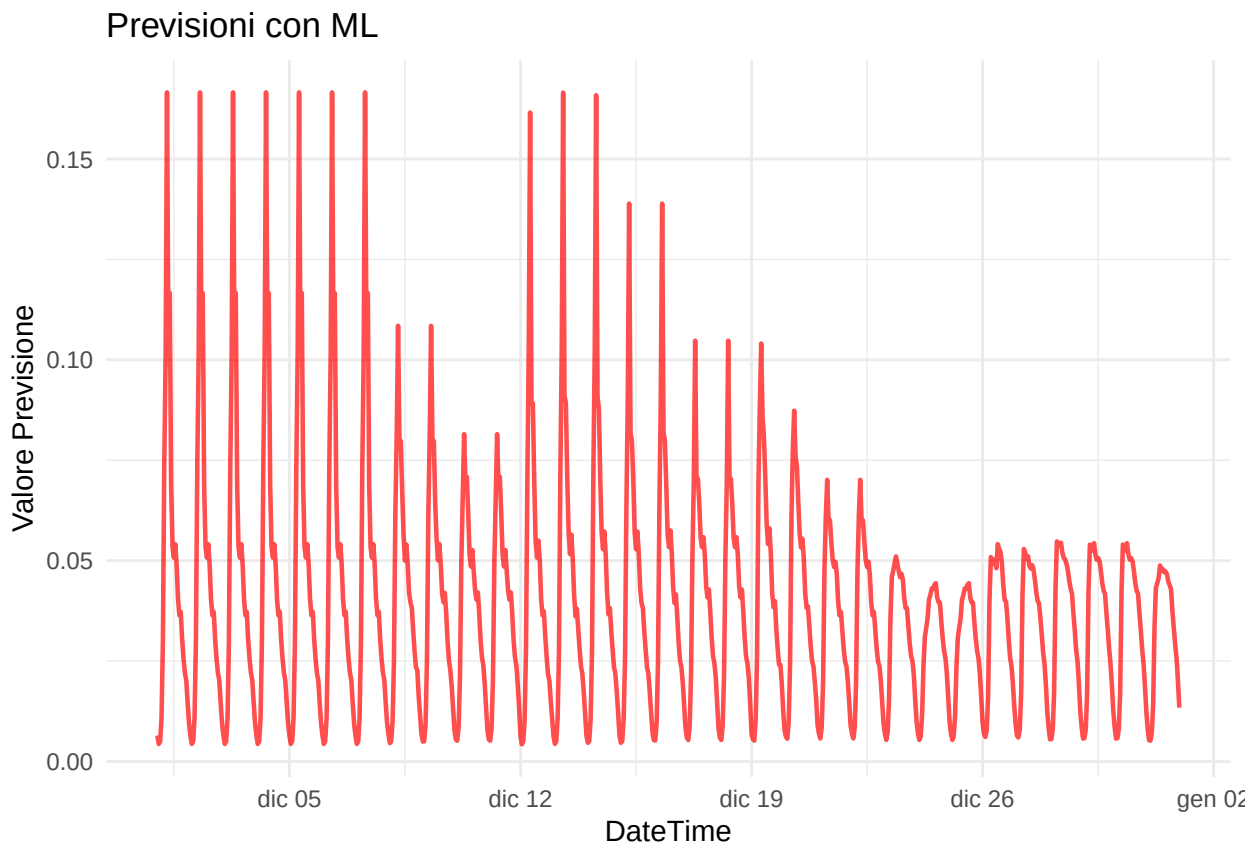
```r
dim(forecast_dataframe)
```

```
## [1] 744   4
```

```r
library(ggplot2)

# Filtra la serie originale per sovrapporre solo fino all'ultima osservazione reale
ts_train_df <- subset(ts_cutted_filled_df, Date < as.POSIXct("2016-12-01 00:00:00"))

# Combina i dati di training e le previsioni
forecast_df <- data.frame(
  DateTime = forecast_dataframe$DateTime,
  Forecast = forecast_dataframe$ML
)

# Plot con ggplot2
ggplot() +
  # Serie originale (parte di training)
  geom_line(data = ts_train_df, aes(x = Date, y = Value), color = "blue", size = 0.8, alpha = 0.7) +
  # Previsioni
  geom_line(data = forecast_df, aes(x = DateTime, y = Forecast), color = "red", size = 0.8, alpha = 0.7
  # Personalizzazione
  labs(title = "Serie Originale vs Previsioni",
       x = "DateTime", y = "Valore") +
  theme_minimal()
```




```r
library(ggplot2)

# Combina i dati delle previsioni
forecast_df <- data.frame(
  DateTime = forecast_dataframe$DateTime,
  Forecast = forecast_dataframe$ML
)

# Plot con ggplot2: solo previsioni
ggplot(data = forecast_df) +
  # Previsioni
  geom_line(aes(x = DateTime, y = Forecast), color = "red", size = 0.8, alpha = 0.7) +
  # Personalizzazione
  labs(title = "Previsioni con ML",
       x = "DateTime", y = "Valore Previsione") +
  theme_minimal()
```




```
# Specifica il percorso e il nome del file CSV
file_path <- "forecast_dataframe.csv"

# Salva il data frame come CSV
write.csv(forecast_dataframe, file = file_path, row.names = FALSE)

# Conferma il salvataggio
cat("Il file è stato salvato in:", file_path, "\n")
```

```
## Il file è stato salvato in: forecast_dataframe.csv
```