SCHOOL OF INFORMATION SCIENCES & TECHNOLOGY

DEPARTMENT OF STATISTICS
POSTGRADUATE PROGRAMM

APPENDIX

Written by
Χατζόπουλος Γεράσιμος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Που υποβλήθηκε στο Τμήμα Στατιστικής του Οικονομικού Πανεπιστημίου Αθηνών ως μέρος των απαιτήσεων για την απόκτηση Διπλώματος Μεταπτυχιακών Σπουδών στην Εφαρμοσμένη Στατιστική

Αθήνα, Αύγουστος 2023

2.2.1 LINEAR

```
# Set the seed for reproducibility

set.seed(123)

# Generate the simulated data

n <- 100  # number of observations

x <- runif(n, 0, 10)  # independent variable

noise <- rnorm(n, 0, 1)  # some noise

beta0 <- 2  # true intercept

beta1 <- 3  # true slope

y <- beta0 + beta1 * x + noise  # dependent variable

# Create a data frame

data <- data.frame(x = x, y = y)

# Fit a linear regression model

model <- lm(y ~ x, data = data)

# Print the summary of the model

summary(model)

# Plot the data and the fitted line

plot(data$x, data$y, main = "Linear Regression", xlab = "x", ylab
= "y")

abline(model, col = "red")
```

2.2.2 LOGISTIC

```
# Loading necessary libraries

library(ggplot2)

# Setting seed for reproducibility

set.seed(123)

# Generating independent variable

x <- runif(100, -10, 10)

# Generating dependent variable

z <- 1 + 3*x  # Linear function

prob <- 1/(1 + exp(-z))  # Logistic function

y <- rbinom(100, 1, prob)  # Binary variable

# Fitting logistic regression model

model <- glm(y ~ x, family = "binomial")

# Creating sorted x values

sorted_x <- sort(x)

# Making predictions (on the probability scale)

predicted_probs <- predict(model, newdata = data.frame(x = sorted_x), type = "response")

# Creating dataframe of sorted x, y and predicted probabilities

df <- data.frame(x = sorted_x, PredictedProbability = predicted_probs)

df_actual <- data.frame(x = x, y = y)

# Creating the base plot

p <- ggplot() +

    geom_point(data = df_actual, aes(x = x, y = y), colour = "blue", alpha = 0.5) +

    geom_line(data = df, aes(x = x, y = PredictedProbability), colour = "red") +
    labs(title = "Logistic Regression", x = "X", y = "Y /
Predicted Probability") +

    theme_minimal()

# Printing the plot

print(p)
```

2.2.3 TIMES SERIES ANALYSIS

```
# Loading the necessary libraries

library(forecast)

library(ggplot2)

# Setting the seed for reproducibility

set.seed(123)

# Generating a time series

data <- arima.sim(n = 100, model = list(ar = c(0.6), ma = c(0.3)))

# Converting the data to a ts object

data_ts <- ts(data)

# Fitting an ARIMA model

model <- auto.arima(data_ts)

# Forecasting future values

forecast_result <- forecast(model, h = 20)

# Visualizing the results

autoplot(forecast_result) +

  ggtitle("ARIMA Model Forecast") +

  xlab("Time") +

  ylab("Values") +

  theme_minimal()
```

2.2.4 DECISION TREES

```
# Load the required packages

library(rpart)

library(rpart.plot)

 # Create a sample dataset

 data <- iris

 # Fit the decision tree model

 tree_model <- rpart(Species ~ ., data = data)

 # Define a color palette for the nodes

 box_colors <- list("pink", "lightblue", "lightgray")  # Specify
your desired colors

# Plot the decision tree with custom colors

rpart.plot(tree_model, type = 2, extra = 101, under = TRUE,
fallen.leaves = FALSE,

+               branch = 0.6, shadow.col = "gray", box.palette =
box_colors,

+             nn = TRUE, main = "Decision Tree")
```

2.2.5 RANDOM FORESTS

```
# Loading necessary libraries

library(randomForest)

library(ggplot2)

# Setting seed for reproducibility

set.seed(123)

# Generating independent variables

x1 <- runif(100, -10, 10)

x2 <- runif(100, -10, 10)

# Generating dependent variable

y <- 1 + 2*x1 + 3*x2 + rnorm(100, 0, 0.5)

# Creating a data frame

data <- data.frame(x1 = x1, x2 = x2, y = y)

# Fitting a Random Forest model

model <- randomForest(y ~ x1 + x2, data = data)

# Making predictions

data$predicted <- predict(model, newdata = data)

# Visualizing the results

ggplot(data, aes(x = x1, y = y)) +

  geom_point(aes(color = "Actual")) +

  geom_point(aes(y = predicted, color = "Predicted")) +

  scale_color_manual(values = c("Actual" = "blue", "Predicted" =
"red")) +

  labs(title = "Random Forest Model", x = "X1", y = "Y / Predicted
Y", color = "Legend") +

  theme_minimal()
```

2.2.6 SVM

```
# Loading necessary libraries

library(e1071)

library(ggplot2)

# Setting seed for reproducibility

set.seed(123)

# Generating independent variable

x <- runif(100, -10, 10)

# Generating dependent variable

y <- 1 + 2*x + rnorm(100, 0, 0.5)

# Creating a data frame

data <- data.frame(x = x, y = y)

# Fitting an SVR model

model <- svm(y ~ x, data = data)

# Making predictions

data$predicted <- predict(model, newdata = data)

# Ordering data by 'x' for plotting

data <- data[order(data$x), ]

# Visualizing the results

ggplot(data, aes(x = x, y = y)) +

  geom_point(aes(color = "Actual")) +

  geom_line(aes(y = predicted, color = "Predicted")) +

  scale_color_manual(values = c("Actual" = "blue", "Predicted" =
"red")) +

  labs(title = "Support Vector Regression Model", x = "X", y =
"Y / Predicted Y", color = "Legend") +

  theme_minimal()
```

2.2.7 NAÏVE BAYES

```
# Loading necessary libraries
library(e1071)
library(ggplot2)
# Setting seed for reproducibility
set.seed(123)
# Generating independent variable
x <- runif(100, -10, 10)
# Generating dependent variable based on a condition
y <- ifelse(x > 0, "positive", "negative")
# Creating a data frame
data <- data.frame(x = x, y = as.factor(y))
# Fitting a Naive Bayes model
model <- naiveBayes(y ~ x, data = data)
# Making predictions
data$predicted <- predict(model, newdata = data)
# Ordering data by 'x' for plotting
data <- data[order(data$x), ]
# Visualizing the results
ggplot(data, aes(x = x, y = y)) +
  geom_point(aes(color = y), size = 3, alpha = 0.6) +
  geom_rug(data = data[data$predicted == "positive", ], sides =
"t", col = "blue", alpha = 0.5, size = 1.2) +
  geom_rug(data = data[data$predicted == "negative", ], sides =
"b", col = "red", alpha = 0.5, size = 1.2) +
  labs(title = "Naive Bayes Classification", x = "X", y = "Y /
Predicted Y", color = "Actual Y") +
  theme_minimal()
```

2.2.9 GRADIENT BOOSTING

```
# Loading necessary libraries

library(gbm)

library(ggplot2)

# Setting seed for reproducibility

set.seed(123)

# Generating independent variable

x <- runif(100, -10, 10)

# Generating dependent variable

y <- 1 + 2*x + rnorm(100, 0, 0.5)

# Creating a data frame

data <- data.frame(x = x, y = y)

# Fitting a Gradient Boosting Model

model <- gbm(y ~ x, data = data, distribution = "gaussian",
n.trees = 100, interaction.depth = 4)

# Making predictions

data$predicted <- predict(model, newdata = data, n.trees = 100)

# Ensure that the predicted values are numeric

data$predicted <- as.numeric(data$predicted)

# Ordering data by 'x' for plotting

data <- data[order(data$x), ]

# Visualizing the results

ggplot(data, aes(x = x, y = y)) +

  geom_point(aes(color = "Actual")) +

  geom_line(aes(y = predicted, color = "Predicted")) +

  scale_color_manual(values = c("Actual" = "blue", "Predicted" =
"red")) +

  labs(title = "Gradient Boosting Model", x = "X", y = "Y /
Predicted Y", color = "Legend") +
  theme_minimal()
```

2.3.4 COMPARTMENT

```
install.packages("deSolve")

library(deSolve)

library(ggplot2)

# Define the SIR model function

sir_model <- function(time, state, parameters) {

    with(as.list(c(state, parameters)), {

        # Model equations

        dS <- -beta * S * I

        dI <- beta * S * I - gamma * I

        dR <- gamma * I

        # Return the derivative of each compartment

        return(list(c(dS, dI, dR)))

    })

}

# Set initial conditions and parameter values

initial_state <- c(S = 999, I = 1, R = 0)

parameters <- c(beta = 0.2, gamma = 0.1)

# Set time points for prediction

times <- seq(0, 100, by = 0.1)

# Solve the differential equations using the SIR model

solution <- ode(y = initial_state, times = times, func =
sir_model, parms = parameters)

# Create a data frame with the solution

df <- as.data.frame(solution)

# Plot the predicted results

ggplot(df, aes(x = time)) +
    geom_line(aes(y = S, color = "Susceptible"), size = 1) +

    geom_line(aes(y = I, color = "Infected"), size = 1) +

    geom_line(aes(y = R, color = "Recovered"), size = 1) +

    labs(x = "Time", y = "Population", color = "Compartment") +

    scale_color_manual(values   =   c("Susceptible"   =   "blue",
"Infected" = "red", "Recovered" = "green")) +

    theme_minimal()
```

2.3.5 AGENT BASED

```
# Load necessary library

library(ggplot2)

library(gridExtra)


# Create a function to simulate agent movement

simulate_agent_movement <- function(num_iterations) {

  agent_positions <- data.frame(iteration = integer(), x = 
integer(), y = integer())

  agent <- c(0, 0)


  for (iteration in 1:num_iterations) {

    move <- sample(c(-1, 1), 2, replace = TRUE)

    agent <- agent + move

    agent_positions <- rbind(agent_positions, 
data.frame(iteration = iteration, x = agent[1], y = agent[2]))

  }


  return(agent_positions)

}


# Set the number of iterations

num_iterations <- 4


# Simulate agent movement

agent_positions <- simulate_agent_movement(num_iterations)


# Create a custom color palette
```

```
colors <- c("#1f77b4", "#ff7f0e", "#2ca02c", "#d62728")


# Create plots for each iteration
plots <- list()
for (i in 1:num_iterations) {
  plot <- ggplot(data = agent_positions %>% filter(iteration <= i), aes(x, y)) +
    geom_path(aes(group = iteration), color = colors[i], size = 1.5, lineend = "round") +
    geom_point(data = agent_positions %>% filter(iteration == i), size = 5, color = colors[i]) +
    geom_text(data = agent_positions %>% filter(iteration == i), aes(label = paste("Iteration", i)),
              vjust = -1.5, color = colors[i]) +  # Adjusted vjust value
    geom_path(data = agent_positions %>% filter(iteration <= i), linetype = "dashed", color = "gray", size = 0.5) +
    labs(title = paste("Agent Movement - Iteration", i), x = "X Coordinate", y = "Y Coordinate") +
    theme_minimal() +
    theme(legend.position = "none",
          plot.title = element_text(size = 16, hjust = 0.5, margin = margin(b = 15)),
          axis.title = element_text(size = 14),
          axis.text = element_text(size = 12),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_rect(fill = "white"))
  plots[[i]] <- plot
}
```

```
# Display plots

grid.arrange(grobs = plots, ncol = 2)
```

2.3.6 SPATIAL

```
# Load packages

library(sp)

library(spdep)

library(ggplot2)

# Create a 10x10 grid

grid_df <- expand.grid(x = seq(1, 10), y = seq(1, 10))

# Generate some random data for the grid

set.seed(123)

grid_df$vals <- rnorm(n = nrow(grid_df))

# Convert the grid to a spatial object

coordinates(grid_df) <- ~x+y

class(grid_df) <- "SpatialPointsDataFrame"

# Define neighbors (using dnearneigh function)

nb <- dnearneigh(coordinates(grid_df), d1 = 0, d2 = sqrt(2))

# Create spatial weights matrix

listw <- nb2listw(nb, style = "W")

# Perform Moran's I test

moran.test(grid_df@data$vals, listw)
```

```
# Calculate lagged values

grid_df@data$vals.lag <- lag.listw(listw, grid_df@data$vals)

# Perform the spatial regression

model <- lm(vals ~ vals.lag, data = as.data.frame(grid_df@data))

# Print the summary of the model

summary(model)

# Add residuals to the data frame

grid_df@data$residuals <- residuals(model)

# Convert grid to data frame for ggplot

grid_df <- as.data.frame(grid_df)

# Plot the data

ggplot(grid_df, aes(x = x, y = y, fill = vals)) +

  geom_tile() +

  scale_fill_gradient2(low = "blue", high = "red", mid = "white",

                       midpoint = median(grid_df$vals), limit =
range(grid_df$vals)) +

  theme_minimal() +

  ggtitle("Data")

# Plot residuals

ggplot(grid_df, aes(x = x, y = y, fill = residuals)) +

  geom_tile() +

  scale_fill_gradient2(low = "blue", high = "red", mid = "white",

                       midpoint   =   median(grid_df$residuals),
limit = range(grid_df$residuals)) +

  theme_minimal() +

  ggtitle("Residuals")
```

2.3.7 BAYESIAN

```
# Simulated data

cases <- c(10, 15, 20, 25, 30)  # Number of cases over time

n <- length(cases)  # Number of time points

# Prior distribution parameters

prior_alpha <- 1

prior_beta <- 1

# Bayesian updating

posterior_alpha <- prior_alpha + sum(cases)

posterior_beta <- prior_beta + n

# Posterior predictive distribution

new_cases <- rbeta(1000, posterior_alpha, posterior_beta) * 100

# Summary statistics of the posterior predictive distribution

mean_cases <- mean(new_cases)

median_cases <- median(new_cases)

credible_interval <- quantile(new_cases, c(0.025, 0.975))

# Plot the posterior predictive distribution

hist(new_cases,  breaks  =  20,  col  =  "lightblue",  xlab  =  "New
Cases", main = "Posterior Predictive Distribution", density = 10)

lines(density(new_cases), col = "red", lwd = 2)

abline(v = mean_cases, col = "red", lwd = 2, lty = 2)

legend("topright", legend = c("Mean", "95% Credible Interval"),
col = c("red", "black"), lty = c(2, 1), lwd = 2)

# Print results

cat("Mean new cases:", mean_cases, "\n")

cat("Median new cases:", median_cases, "\n")
cat("95%   Credible   Interval:",   credible_interval[1],   "-",
credible_interval[2], "\n")
```

2.3.8 GLM

```
# Load required libraries

library(ggplot2)

install.packages("scales")

library(scales)

# Simulated data for illustration purposes

set.seed(123)

date  <-  seq(as.Date("2022-01-01"),  as.Date("2022-01-31"),  by  =
"day")

cases <- rpois(length(date), lambda = 10)

# Create a data frame

data <- data.frame(date, cases)

# Fit a GLM model

model <- glm(cases ~ date, data = data, family = poisson)

# Generate predictions

new_dates  <-  seq(as.Date("2022-02-01"),  as.Date("2022-02-28"),
by = "day")

new_data <- data.frame(date = new_dates)

predicted_cases  <-  predict(model,  newdata  =  new_data,  type  =
"response")

# Combine original and predicted data

combined_data  <-  rbind(data,  data.frame(date  =  new_dates,  cases
= predicted_cases))

# Plot the actual and predicted cases

ggplot(combined_data, aes(x = date, y = cases)) +

    geom_line(color = "blue") +
    geom_point(data = data, aes(x = date, y = cases), color =
"blue", size = 2) +

    geom_point(data = combined_data[length(data$date) +
1:nrow(new_data), ], aes(x = date, y = cases), color = "red",
size = 2) +

    labs(x = "Date", y = "Number of Cases", title = "GLM
Predictive Model") +

    theme_minimal() +

    theme(plot.title = element_text(hjust = 0.5),

          axis.text.x = element_text(angle = 45, hjust = 1),

          legend.position = "none") +

    scale_x_date(labels = date_format("%b %d"))
```

2.3.9 MACHINE LEARNING

```
# Load libraries

library(randomForest)

library(ggplot2)

# Generate a synthetic dataset

set.seed(123)

num_samples <- 200

age <- rnorm(num_samples, mean = 50, sd = 10)

gender <- rbinom(num_samples, 1, 0.5)

fever <- rbinom(num_samples, 1, 0.3)

cough <- rbinom(num_samples, 1, 0.4)

fatigue <- rbinom(num_samples, 1, 0.6)

DiseaseStatus  <-  factor(rbinom(num_samples,  1,  0.5),  levels  =
c(0, 1)) # Convert to factor

# Create the dataframe

data   <-   data.frame(age,   gender,   fever,   cough,   fatigue,
DiseaseStatus)

# Shuffle the data

data <- data[sample(nrow(data)),]

# Create 80-20 split

train_index <- round(nrow(data) * 0.8)

# Create Training and Test set

data_train <- data[1:train_index, ]

data_test <- data[(train_index + 1):nrow(data), ]

# Train the random forest model
```

```
rf_model <- randomForest(DiseaseStatus ~ ., data = data_train,
ntree = 100)

# Make predictions on the test set

rf_predictions <- predict(rf_model, data_test)

# Manually calculate the accuracy

accuracy <- sum(rf_predictions == data_test$DiseaseStatus) /
nrow(data_test)

print(paste("Accuracy: ", accuracy))

# Visualize feature importance

importance    <- importance(rf_model)

varImportance <- data.frame(Variables = row.names(importance),
Importance = round(importance[ ,'MeanDecreaseGini'],2))

# Use ggplot2 to visualize the relative importance of variables

ggplot(varImportance, aes(x = reorder(Variables, Importance), y
= Importance, fill = Importance)) +

    geom_bar(stat='identity') +

    labs(x = 'Variables') +

    coord_flip() +

    theme_minimal()
```

2.5.1 POISSON

```
# Example: Modeling the number of new cases of malaria per week
lambda <- 5  # Mean number of new cases per week
# Generate Poisson-distributed data
data <- rpois(100, lambda)
# Plotting the data
hist(data, breaks = 20, probability = TRUE, main = "Poisson
Distribution",
     xlab = "Number of New Cases", ylab = "Probability")
# Plotting the probability mass function
x <- 0:15
pmf <- dpois(x, lambda)
plot(x, pmf, type = "h", lwd = 2, ylim = c(0, max(pmf) + 0.05),
     main = "Poisson PMF", xlab = "Number of New Cases", ylab =
"Probability")
```

2.5.2 NEGATIVE BINOMIAL

```
# Example: Modeling the number of reported cases of dengue fever

size <- 10   # Size parameter

prob <- 0.3  # Probability of success

# Generate negative binomial-distributed data

data <- rnbinom(100, size, prob)

# Plotting the data

hist(data, breaks = 20, probability = TRUE, main = "Negative
Binomial Distribution",

   xlab = "Number of Reported Cases", ylab = "Probability")

# Plotting the probability mass function

x <- 0:40

pmf <- dnbinom(x, size, prob)

plot(x, pmf, type = "h", lwd = 2, ylim = c(0, max(pmf) + 0.05),

   main = "Negative Binomial PMF", xlab = "Number of Reported
Cases", ylab = "Probability")
```

2.5.3 NORMAL

```
# Example: Modeling the body temperature of individuals infected
with influenza

mean <- 98.6   # Mean body temperature

sd <- 0.5      # Standard deviation

# Generate normally-distributed data

data <- rnorm(100, mean, sd)

# Plotting the data

hist(data, breaks = 20, probability = TRUE, main = "Normal
Distribution",

     xlab = "Body Temperature", ylab = "Density")

# Plotting the probability density function

x <- seq(97, 100, 0.01)

pdf <- dnorm(x, mean, sd)

plot(x, pdf, type = "l", lwd = 2, ylim = c(0, max(pdf) + 0.05),

     main = "Normal PDF", xlab = "Body Temperature", ylab =
"Density")
```

2.5.4 BETA-BINOMIAL

```
# Example: Modeling the proportion of individuals vaccinated against measles

alpha <- 10    # Shape parameter 1

beta <- 5      # Shape parameter 2

n <- 100       # Number of trials

# Generate beta-binomial-distributed data

data <- rbeta(n, alpha, beta)

data <- rbinom(n, size = 1, prob = data)

# Plotting the data

barplot(table(data), main = "Beta-Binomial Distribution",

        xlab = "Vaccination Status", ylab = "Count")
```

2.5.5 GAMMA

```
# Example: Modeling the duration of symptoms for patients with
typhoid fever

shape <- 5   # Shape parameter

rate <- 0.5  # Rate parameter

# Generate gamma-distributed data

data <- rgamma(100, shape, rate)

# Plotting the data

hist(data, breaks = 20, probability = TRUE, main = "Gamma
Distribution",

     xlab = "Duration of Symptoms", ylab = "Probability")

# Plotting the probability density function

x <- seq(0, 20, 0.1)

pdf <- dgamma(x, shape, rate)

plot(x, pdf, type = "l", lwd = 2, ylim = c(0, max(pdf) + 0.05),

     main = "Gamma PDF", xlab = "Duration of Symptoms", ylab =
"Probability")
```

2.5.6 WEIBULL

```
# Example: Modeling the time to recovery for patients with cholera

shape <- 2   # Shape parameter

scale <- 5   # Scale parameter

# Generate Weibull-distributed data

data <- rweibull(100, shape, scale)

# Plotting the data

hist(data, breaks = 20, probability = TRUE, main = "Weibull Distribution",

     xlab = "Time to Recovery", ylab = "Probability")

# Plotting the probability density function

x <- seq(0, 20, 0.1)

pdf <- dweibull(x, shape, scale)

plot(x, pdf, type = "l", lwd = 2, ylim = c(0, max(pdf) + 0.05),

     main = "Weibull PDF", xlab = "Time to Recovery", ylab = "Probability")
```

2.5.7 LOG-NORMAL

```
# Example: Modeling the distribution of incubation periods for
norovirus infection

meanlog <- 2     # Mean of the logarithm of incubation periods

sdlog <- 0.5       # Standard deviation of the logarithm of
incubation periods

# Generate log-normal-distributed data

data <- rlnorm(100, meanlog, sdlog)

# Plotting the data

hist(data, breaks = 20, probability = TRUE, main = "Log-Normal
Distribution",

     xlab = "Incubation Period", ylab = "Density")

# Plotting the probability density function

x <- seq(0, 10, 0.1)

pdf <- dlnorm(x, meanlog, sdlog)

plot(x, pdf, type = "l", lwd = 2, ylim = c(0, max(pdf) + 0.05),

     main = "Log-Normal PDF", xlab = "Incubation Period", ylab =
"Density")
```

2.5.8 BINOMIAL

```
# Example: Modeling the probability of HIV infection among intravenous drug users

n <- 10   # Number of trials

p <- 0.3  # Probability of success

# Generate binomial-distributed data

data <- rbinom(100, n, p)

# Plotting the data

barplot(table(data), main = "Binomial Distribution",

        xlab = "Number of Infections", ylab = "Count")
```

2.5.9 MULTINOMIAL

```
# Example: Modeling the distribution of disease severity among
patients with dengue fever

probs <- c(0.4, 0.3, 0.3)   # Probabilities of mild, moderate,
severe

# Generate multinomial-distributed data

data <- rmultinom(100, size = 1, prob = probs)

# Plotting the data

barplot(t(data), main = "Multinomial Distribution",

        xlab = "Disease Severity", ylab = "Count", legend =
c("Mild", "Moderate", "Severe"))
```

2.5.10 DIRICHLET

```
# Install and load the required package
install.packages("ggtern")
library(ggtern)
# Example: Modeling the distribution of different serotypes of
poliovirus
proportions <- c(0.3, 0.4, 0.3)   # Proportions of serotypes
sample_size <- 100   # Number of samples
# Generate multinomial-distributed data
data <- t(apply(rmultinom(sample_size, 1, proportions), 2,
function(x) x/sum(x)))
# Create a data frame with the normalized proportions
df <- data.frame(Serotype1 = data[, 1], Serotype2 = data[, 2],
Serotype3 = data[, 3])
# Plotting the data using a ternary plot
ggtern(data = df, aes(x = Serotype1, y = Serotype2, z =
Serotype3)) +
  geom_point() +
  theme_bw() +
  labs(title = "Dirichlet Distribution", x = "Serotype 1", y =
"Serotype 2", z = "Serotype 3")
```

3.3.2 HHH4 MODEL SPECIFICATION-FITTING

```
#required packages for our model to work
library(surveillance)

library(lubridate)

data <- data.frame(

    Time = seq(as.Date("2008/01/01"), as.Date("2020/12/01"), by
= "month"),

    NumValue = c(5, 2, 4, 2, 6, 3, 4, 14, 2, 2, 10, 2, 6, 4, 6,
3, 4, 8, 5, 10, 4, 1, 9, 4, 10, 11, 16, 27, 28, 46, 111, 242,
77, 21, 3, 4, 3, 3, 5, 3, 7, 3, 4, 17, 2, 1, 3, 4, 6, 3, 7, 8,
4, 12, 26, 33, 16, 16, 12, 9, 18, 13, 16, 21, 40, 40, 46, 97,
69, 44, 39, 13, 19, 16, 18, 12, 42, 41, 42, 107, 39, 13, 9, 9,
2, 9, 16, 18, 30, 16, 41, 71, 31, 24, 16, 11, 15, 13, 17, 38,
55, 31, 50, 85, 30, 18, 16, 5, 1, 7, 14, 23, 36, 29, 36, 59,
18, 15, 24, 4, 7, 6, 11, 29, 40, 40, 40, 58, 29, 33, 23, 15,
16, 15, 40, 70, 117, 79, 96, 158, 77, 81, 105, 50, 79, 92, 85,
34, 29, 40, 70, 342, 233, 224, 115, 19)

)

# Extract the year from the Time column

data$Year <- lubridate::year(data$Time)

# Select training data indices based on the year

train_inds <- which(data$Year %in% 2008:2018)

# Transform data to the sts class (space-time surveillance
data) to be used in #the surveillance package functions

train_data <- sts(data$NumValue,

                  start = c(lubridate::year(data[1, "Time"]),
lubridate::month(data[1, "Time"])),
                   freq = 12L)

# Define different specification options to be used in the
model

family_values <- c("Poisson", "NegBin1")

S_ar_values <- 0:3

S_end_values <- 0:3

lag_ar_values <- 1:3

# Dataframe for all combinations of the model specifications


model_specifications <- as.data.frame(

    expand.grid(

        family = family_values,

        S_ar = S_ar_values,

        S_end = S_end_values,

        lag_ar = lag_ar_values,

        mean_log_score = NA_real_,

        stringsAsFactors = FALSE),

    stringsAsFactors = FALSE)

# Create an empty list to store fitted models

fits <- vector("list", nrow(model_specifications))

# Loop over all model specifications and fit models


for(specification_ind in seq_len(nrow(model_specifications))) {

    family <- model_specifications$family[specification_ind]

    S_ar <- model_specifications$S_ar[specification_ind]

    lag_ar <- model_specifications$lag_ar[specification_ind]

    S_end <- model_specifications$S_end[specification_ind]
```

```
# Create a hhh4 model using the given specifications for
#monthly data

    fits[[specification_ind]] <- hhh4(train_data,

                                      control = list(

                                          ar = list(f =
addSeason2formula(f = ~ 1, S = S_ar, period = 12), lag =
lag_ar),

                                          end = list(f =
addSeason2formula(f = ~ 1, S = S_end, period = 12)),

                                          subset = seq(from =
lag_ar + 1, to = min(max(train_inds), 131)),

                                          family = family

                                      ))



# Generate one-step-ahead predictions for each fitted model

    one_step_ahead_preds <-
oneStepAhead(fits[[specification_ind]],

                                         tp =
min(nrow(train_data) - 1, 129))

# Compute prediction scores for each prediction

    pred_scores <- scores(one_step_ahead_preds)



# Store the mean log score, mean ranked probability score
#(RPS), mean #Dawid-Sebastiani Score (DSS) and mean spherical
#error score (SES) for #each model in the model_specifications
#dataframe

    model_specifications$mean_log_score[specification_ind] <-
mean(pred_scores[, "logs"])

    model_specifications$mean_rps[specification_ind] <-
mean(pred_scores[, "rps"])

    model_specifications$mean_dss[specification_ind] <-
mean(pred_scores[, "dss"])
```

```
    model_specifications$mean_ses[specification_ind] <-
mean(pred_scores[, "ses"])

}

# Create the model

 surveillance_fits <- list(

     model_specifications = model_specifications,

     model_fits = fits

 )

saveRDS(surveillance_fits,

        file = file.path(

            "C:/Users/forta/OneDrive/Desktop/THESIS UNTIL NOW/r
results",

            "surveillance-fits.rds"))
```

3.3.3 MODEL SELECTION

```
selection_criteria <- "aic" # how to pick "best" hhh4 model

#Set the prediction horizon to 12 time steps ahead

all_prediction_horizons <- 1:12

all_prediction_statistics <- c("log_score",
                                "pt_pred",

                                "AE",

                                "interval_pred_lb_95",

                                "interval_pred_ub_95",

                                "interval_pred_lb_50",

                                "interval_pred_ub_50")



# Identify indices of the data that fall within specified
#years

prediction_time_inds <- which(data$season %in%
paste0(2008:2019, "/", 2009:2020))

# convert dates

data$time <- ymd(data$Time)

#Add time_index column.  This is used for calculating the
#periodic kernel.

## Here, this is calculated as the number of days since some
origin date (1970-1-1 in this case).

#The origin is conventional.

data$time_index <- as.integer(data$time -  ymd(paste("1970",
"01", "01", sep = "-")))

#load surveillance fits and choose best one



surveillance_fits <- readRDS(file = file.path(

    "C:/Users/forta/OneDrive/Desktop/THESIS UNTIL NOW/r
results/surveillance-fits.rds"))

# Calculate AIC for each model

aic_by_surveillance_fit <- sapply(surveillance_fits$model_fits,
function(sfit) {

    summary(sfit)$AIC

})
```

```
# Calculate BIC for each model

bic_by_surveillance_fit <- sapply(surveillance_fits$model_fits,
function(sfit) {

    summary(sfit)$BIC

})

# Select the best model based on the selection criterias above

if(identical(selection_criteria, "log_score")) {

# If selection criteria is log_score, select model with minimum
#mean log score

    best_spec_ind <-
which.min(surveillance_fits$model_specifications$mean_log_score
)

    surveillance_fit <-
surveillance_fits$model_fits[[best_spec_ind]]

# If selection criteria is AIC, select model with minimum AIC

} else if(identical(selection_criteria, "aic")) {

    best_spec_ind <- which.min(aic_by_surveillance_fit)

    surveillance_fit <-
surveillance_fits$model_fits[[best_spec_ind]]

# If selection criteria is BIC, select model with minimum BIC

} else if(identical(selection_criteria, "bic")) {

    best_spec_ind <- which.min(bic_by_surveillance_fit)

    surveillance_fit <-
surveillance_fits$model_fits[[best_spec_ind]]

}
```

3.3.4 RESULTS

```
# Mean log score

best_model_mean_log_score <-
surveillance_fits$model_specifications$mean_log_score[best_spec
_ind]
```

```
print(paste("Mean Log Score: ", best_model_mean_log_score))
```

```
summary(surveillance_fit)
```

```
AIC(surveillance_fit)
 # Predictions of the best model

 best_predictions <- predict(surveillance_fit)

#DSS

model_specifications$mean_dss[specification_ind]
```

```
#SES

model_specifications$mean_ses[specification_ind]
```

```
#RPS
model_specifications$mean_rps[specification_ind]
```

```
# Show the predictions

 print(best_predictions)

# Ensure that the ggplot2 package is installed

if(!require(ggplot2)){

    install.packages("ggplot2")

}


# Load the package

library(ggplot2)


# Create a data frame for the predicted values
```

```
predicted_data <- data.frame(

    Time = data$Time[1:130],

    NumValue = best_predictions[,1]

)



# Plot the actual and predicted values

# Load the extrafont and ggthemes packages for additional fonts
and themes

library(extrafont)

library(ggthemes)

#one step ahead , it takes all the previous one to predict the
next #one

# Generate the plot

ggplot() +

    geom_line(data = data[1:130,], aes(x = Time, y = NumValue,
color = "Actual"), size = 1) +

    geom_line(data = predicted_data, aes(x = Time, y =
NumValue, color = "Predicted"), size = 1, linetype = "twodash")
+

    scale_color_manual(values = c("Actual" = "#1C4E80",
"Predicted" = "#FF4500"),

                       name = " ",

                       labels = c("Actual Values", "Predicted
Values")) +

    labs(title = "Actual vs Predicted Values",

         subtitle = "Comparison of actual and predicted values
over time",

         x = "Time",

         y = "Number of Cases") +

    theme_minimal(base_family = "Arial") +
    theme(
        plot.title = element_text(size = 20, face = "bold",
hjust = 0.5),
        plot.subtitle = element_text(size = 15, face =
"italic", hjust = 0.5),
        axis.title = element_text(size = 15, face = "bold"),
        legend.position = "bottom",
        legend.text = element_text(size = 12),
        legend.title = element_text(size = 15)
    )
```

```
library(Metrics)
# Calculate the RMSE
rmse_val <- rmse(data$NumValue[1:130], predicted_data$NumValue)
print(paste("RMSE: ", rmse_val))
```

```
# Calculate the MAE
mae_val <- mae(data$NumValue[1:130], predicted_data$NumValue)
print(paste("MAE: ", mae_val))
```

```
# Calculate the MAPE
mape_val <- mape(data$NumValue[1:130], predicted_data$NumValue)
print(paste("MAPE: ", mape_val))
```

```
# Calculate the residuals HEATMAP
residuals <- data$NumValue[1:130] - predicted_data$NumValue
# Create a data frame with the residuals
residuals_data <- data.frame(
  Time = data$Time[1:130],

  Month = format(data$Time[1:130], "%m"),

  Year = format(data$Time[1:130], "%Y"),

  Residuals = residuals

)



# Create the heatmap

ggplot(data = residuals_data, aes(x = Month, y = Year, fill =
Residuals)) +

  geom_tile() +

  scale_fill_gradientn(colors = rev(heat.colors(10)), name =
"Residuals") +

  labs(x = "Month", y = "Year", title = "Heatmap of Residuals")
+

  theme_minimal()



library(viridis)

library(scales)

# Generate the heatmap

ggplot(data = residuals_data, aes(x = Month, y = Year, fill =
Residuals)) +

    geom_tile(color = "white", size = 0.1) +

    scale_fill_viridis(option = "C", direction = -1, name =
"Residuals",

                       breaks = scales::pretty_breaks(n = 5)) +

    labs(title = "Heatmap of Residuals",

         subtitle = "Comparing monthly residuals over years",

         x = "Month",

         y = "Year") +
    theme_minimal(base_family = "Arial") +
    theme(
        plot.title = element_text(size = 20, face = "bold",
hjust = 0.5),
        plot.subtitle = element_text(size = 15, hjust = 0.5),
        axis.title = element_text(size = 15, face = "bold"),
        axis.text.x = element_text(angle = 90, vjust = 0.5),
        legend.position = "right",
        legend.title = element_text(size = 12),
        legend.text = element_text(size = 10)
    )
```

```
# Create a residuals plot OVER TIME
residuals_data <- data.frame(Residuals = residuals)
ggplot(residuals_data, aes(x = 1:length(Residuals), y =
Residuals)) +
  geom_point() +
  geom_hline(yintercept=0, linetype="dashed", color = "red") +
  labs(x = "Time", y = "Residuals", title = "Residuals Plot") +
  theme_minimal()


# Load the necessary libraries
library(ggthemes)


# Create the residuals plot
residuals_plot <- ggplot(residuals_data, aes(x =
1:length(Residuals), y = Residuals)) +
  geom_point(color = "steelblue", alpha = 0.5, size = 2) +

  geom_smooth(method = "loess", se = FALSE, color =
"firebrick", linetype = "dashed") +

  geom_hline(yintercept = 0, linetype = "dashed", color =
"red") +

  scale_x_continuous(breaks = seq(0,
length(residuals_data$Residuals), by = 10)) +

  labs(

    x = "Time",

    y = "Residuals",

    title = "Residuals Over Time",

    subtitle = "A plot to check the residuals of the model over
time"

  ) +

  theme_tufte() +

  theme(

    plot.title = element_text(size = 18, face = "bold", hjust =
0.5),

    plot.subtitle = element_text(size = 14, hjust = 0.5),

    axis.title = element_text(size = 14, face = "bold")

  )


print(residuals_plot)
```

```
# Create an ACF plot of the residuals

acf(residuals)
```

```
# Create a histogram of the residuals

ggplot(residuals_data, aes(x=Residuals)) +
  geom_histogram(binwidth=1, color="black", fill="white") +

  labs(x = "Residuals", y = "Frequency", title = "Histogram of 
Residuals") +

  theme_minimal()


# Load necessary libraries

library(ggthemes)


# Create the histogram

residuals_histogram <- ggplot(residuals_data, aes(x = 
Residuals)) +

    geom_histogram(aes(y = ..density..), binwidth = 1, fill = 
"steelblue", color = "black") +

    geom_density(alpha = .2, fill = "firebrick") +

    labs(

        x = "Residuals",

        y = "Density",

        title = "Histogram of Residuals",

        subtitle = "The red line represents the kernel density 
estimation"

    ) +

    theme_fivethirtyeight() +

    theme(

        plot.title = element_text(size = 18, face = "bold", 
hjust = 0.5),

        plot.subtitle = element_text(size = 14, hjust = 0.5),

        axis.title = element_text(size = 14, face = "bold")

    )
```

```
print(residuals_histogram)
```

```
# Create a Predicted vs Actual values plot
comparison_data <- data.frame(Actual = data$NumValue[1:130],
Predicted = predicted_data$NumValue)
ggplot(comparison_data, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  labs(x = "Actual", y = "Predicted", title = "Predicted vs
Actual Values") +
  theme_minimal()


# Load necessary libraries
library(ggthemes)


# Create the comparison plot
comparison_plot <- ggplot(comparison_data, aes(x = Actual, y =
Predicted)) +
    geom_point(color = "steelblue", alpha = 0.5, size = 2) +
    geom_smooth(method = "loess", se = FALSE, color =
"firebrick", linetype = "dashed") +
    geom_abline(intercept = 0, slope = 1, color = "red",
linetype = "dashed") +
    labs(
        x = "Actual Values",
        y = "Predicted Values",
        title = "Predicted vs Actual Values",
        subtitle = "A scatter plot showing the correlation
between predicted and actual values"
```

```
    ) +
    theme_fivethirtyeight() +
    theme(
        plot.title = element_text(size = 18, face = "bold",
hjust = 0.5),
        plot.subtitle = element_text(size = 14, hjust = 0.5),
        axis.title = element_text(size = 14, face = "bold")
    )

print(comparison_plot)
```

#Boxplot of Residuals

```
ggplot(residuals_data, aes(y=Residuals)) +
  geom_boxplot(fill="lightblue") +
  labs(y = "Residuals", title = "Boxplot of Residuals") +
  theme_minimal()

# Load necessary libraries
library(ggthemes)

# Create the boxplot of residuals
residuals_plot <- ggplot(residuals_data, aes(y = Residuals)) +
    geom_boxplot(fill = "steelblue", color = "black") +
    labs(
        y = "Residuals",
        title = "Boxplot of Residuals",
        subtitle = "Displaying the spread and skewness of
residuals"
```

```
    ) +

    theme_minimal() +

    theme(

        plot.title = element_text(size = 18, face = "bold",
hjust = 0.5),

        plot.subtitle = element_text(size = 14, hjust = 0.5),

        axis.title.y = element_text(size = 14, face = "bold")

    )


print(residuals_plot)
```

```
#Scatter Plot of Predictions vs. Residuals

resid_pred_data <- data.frame(Predictions =
predicted_data$NumValue, Residuals = residuals_data$Residuals)

ggplot(resid_pred_data, aes(x = Predictions, y = Residuals)) +

  geom_point() +

  geom_hline(yintercept=0, linetype="dashed", color = "red") +

  labs(x = "Predictions", y = "Residuals", title = "Scatter
Plot of Predictions vs. Residuals") +

  theme_minimal()


# Load necessary libraries

library(ggthemes)


# Create the residuals vs predictions plot

resid_pred_plot <- ggplot(resid_pred_data, aes(x = Predictions,
y = Residuals)) +

  geom_point(color = "steelblue", alpha = 0.5, size = 2) +
  geom_smooth(method = "loess", se = FALSE, color =
"firebrick", linetype = "dashed") +

  geom_hline(yintercept = 0, linetype = "dashed", color =
"red") +

  labs(

    x = "Predictions",

    y = "Residuals",

    title = "Predictions vs. Residuals",

    subtitle = "A scatter plot to assess the relationship
between model predictions and residuals"

  ) +

  theme_fivethirtyeight() +

  theme(

    plot.title = element_text(size = 18, face = "bold", hjust =
0.5),

    plot.subtitle = element_text(size = 14, hjust = 0.5),

    axis.title = element_text(size = 14, face = "bold")

  )


print(resid_pred_plot)
```

```
#density plot of residuals

ggplot(residuals_data, aes(x=Residuals)) +

  geom_density(fill="lightblue") +

  labs(x = "Residuals", y = "Density", title = "Density Plot of
Residuals") +

  theme_minimal()


# Load necessary libraries
```

```
library(ggthemes)


# Create the density plot of residuals

density_plot <- ggplot(residuals_data, aes(x = Residuals)) +

    geom_density(fill = "steelblue", alpha = 0.7) +

    geom_vline(aes(xintercept=mean(Residuals)),

               color="darkred", linetype="dashed", size=1) +

    labs(

        x = "Residuals",

        y = "Density",

        title = "Density Plot of Residuals",

        subtitle = "Visualizing the distribution of residuals"

    ) +

    annotate("text", x = Inf, y = Inf,

             label = paste("Mean =",
round(mean(residuals_data$Residuals), 2)),

             vjust = 2, hjust = 1.5, size = 4, color =
"darkred") +

    theme_minimal() +

    theme(

        plot.title = element_text(size = 18, face = "bold",
hjust = 0.5),

        plot.subtitle = element_text(size = 14, hjust = 0.5),

        axis.title = element_text(size = 14, face = "bold")

    )


print(density_plot)
```

3.4.2 KCDE R IMPLEMENTATION

```
#required packages
library(zoo)

library(tibble)

library("kcde")

library("HIDDA.forecasting")

library(caret)

data <- data.frame(

    Time = seq(as.Date("2008/01/01"), as.Date("2020/12/01"), by
= "month"),

    NumValue = c(5, 2, 4, 2, 6, 3, 4, 14, 2, 2, 10, 2, 6, 4, 6,
3, 4, 8, 5, 10, 4, 1, 9, 4, 10, 11, 16, 27, 28, 46, 111, 242, 77,
21, 3, 4, 3, 3, 5, 3, 7, 3, 4, 17, 2, 1, 3, 4, 6, 3, 7, 8, 4, 12,
26, 33, 16, 16, 12, 9, 18, 13, 16, 21, 40, 40, 46, 97, 69, 44,
39, 13, 19, 16, 18, 12, 42, 41, 42, 107, 39, 13, 9, 9, 2, 9, 16,
18, 30, 16, 41, 71, 31, 24, 16, 11, 15, 13, 17, 38, 55, 31, 50,
85, 30, 18, 16, 5, 1, 7, 14, 23, 36, 29, 36, 59, 18, 15, 24, 4,
7, 6, 11, 29, 40, 40, 40, 58, 29, 33, 23, 15, 16, 15, 40, 70,
117, 79, 96, 158, 77, 81, 105, 50, 79, 92, 85, 34, 29, 40, 70,
342, 233, 224, 115, 19)

)

# Creating a Date formatted index for the data

data$Index <- ymd(data$Time)

DENGUEFR <- zoo(data$NumValue, order.by = data$Time)
```

```
.T <- match(paste0(2018, "-12"), strftime(index(DENGUEFR), "%Y-
%m"))

index(DENGUEFR)[.T]

TEST <- lapply(.T, function (T) seq(from = T, by = 1,
length.out = 24))
#our train data

OWA <- (TEST[[1]][1]-1):(length(DENGUEFR)-1)

length(OWA)

format_period <- function (index, fmt = "%Y-%m", collapse = "
to ") {

    paste0(strftime(index(DENGUEFR)[range(index)], fmt),
collapse = collapse)

}

# Fortify and rename

DENGUEFRdat <- as.data.frame(DENGUEFR)

names(DENGUEFRdat)[1] <- "DENGUEFR"

# Reset index and rename

DENGUEFRdat <- tibble::rownames_to_column(DENGUEFRdat, "Index")

Traindata <- 1:OWA[1]

DENGUEFRdat$Index <- ymd(DENGUEFRdat$Index)

prediction_target_var <- "DENGUEFR"

predictive_vars <- "DENGUEFR"

prediction_horizon <- 1

max_lag <- 1L

max_seasonal_lag <- 0L

init_sample_size <- length(Traindata)  # "scott's rule"

bw_parameterization <- "diagonal"

### **Full** Bandwidth KCDE cost over 80h estimation time with
#8 cores atmy machine with no significant better results
```

```
### So we use the diagonal variant (which took 14 minutes for
#the France Dengue data) in my machine


## setup list describing kernel components

kernel_components <- list()


## add periodic kernel component capturing seasonality

kernel_components <- c(kernel_components, list(list(

    vars_and_offsets = data.frame(

        var_name = "Index", offset_value = 0L, offset_type =
"lag",

        combined_name = "time_index_lag0", stringsAsFactors =
FALSE

    ),

    kernel_fn = kcde::periodic_kernel,

    theta_fixed = list(period = pi / 365.2425),

    theta_est = list("bw"),

    initialize_kernel_params_fn =

        kcde::initialize_params_periodic_kernel,

    initialize_kernel_params_args = list(

        sample_size = init_sample_size

    ),

    get_theta_optim_bounds_fn =

        kcde::get_theta_optim_bounds_periodic_kernel,

    get_theta_optim_bounds_args = NULL,

    vectorize_kernel_params_fn =

        kcde::vectorize_params_periodic_kernel,

    vectorize_kernel_params_args = NULL,

    update_theta_from_vectorized_theta_est_fn =
kcde::update_theta_from_vectorized_theta_est_periodic_kernel,
    update_theta_from_vectorized_theta_est_args = NULL
    )))
## Kernel components for observed values (NumValues)
## First step is setup: create list of data frames specifying
groups of
## variables and offsets included in each kernel component
lag_values <- NULL
for(seasonal_lag in seq(from = 0, to = max_seasonal_lag)) {
    lag_values <- c(lag_values,
        seq(from = 0, to = max_lag) + 52 * seasonal_lag)
}
print(lag_values)


if (bw_parameterization == "diagonal") {
    vars_and_offsets_groups <- list()
    ## Group of variable names and offsets for our prediction
#target
    new_vars_and_offsets_group <- data.frame(
        var_name = prediction_target_var,
        offset_value = prediction_horizon, offset_type =
"horizon",
        stringsAsFactors = FALSE
    )
    vars_and_offsets_groups <- c(vars_and_offsets_groups,

list(new_vars_and_offsets_group))
```

```
    ## Groups of variable names and offsets for lagged
predictive variables

    for(lag_value in lag_values) {

        for(predictive_var in predictive_vars) {

            ## No filtering: group for lagged "raw"/unfiltered
observed incidence

            new_vars_and_offsets_group <- data.frame(

                var_name = predictive_var,

                offset_value = lag_value, offset_type = "lag",

                stringsAsFactors = FALSE

            )

            vars_and_offsets_groups <-
c(vars_and_offsets_groups,

list(new_vars_and_offsets_group))

        }

    }

    ## add combined_name column

    for (i in seq_along(vars_and_offsets_groups)) {

        vars_and_offsets_groups[[i]]$combined_name <- with(

            vars_and_offsets_groups[[i]],

            paste0(var_name, "_", offset_type, offset_value)

        )

    }


} else if (bw_parameterization == "full") {


    ## Prediction target variable

    new_vars_and_offsets_group <- data.frame(
        var_name = prediction_target_var,

        offset_value = prediction_horizon, offset_type =
"horizon",

        stringsAsFactors = FALSE

    )

    ## Lagged prediction target == predictive variables

    for(lag_value in lag_values) {

        for(predictive_var in predictive_vars) {

            ## No filtering: lagged "raw"/unfiltered observed
incidence

            new_vars_and_offsets_group <- rbind(

                new_vars_and_offsets_group,

                data.frame(

                    var_name = predictive_var,

                    offset_value = lag_value, offset_type =
"lag",

                    stringsAsFactors = FALSE

                )

            )

        }

    }

    ## Add combined_name column and put in a list for further
processing below

    new_vars_and_offsets_group$combined_name <- with(

        new_vars_and_offsets_group,

        paste0(var_name, "_", offset_type, offset_value)

    )

    vars_and_offsets_groups <- list(new_vars_and_offsets_group)
```

```
}


print(vars_and_offsets_groups)



## configure discretization

log_exp_x_minus_0.5 <- function(x) {

    temp <- exp(x) - 0.5

    temp[temp < 0] <- 0

    return(log(temp))

}

log_exp_x_plus_0.5 <- function(x) {

    return(log(exp(x) + 0.5))

}

log_round_to_integer_plus_0.5_exp <- function(x) {

    exp_x <- exp(x) + 0.5

    inds_ceil <- exp_x - floor(exp_x) >= 0.5

    exp_x[inds_ceil] <- ceiling(exp_x[inds_ceil])

    exp_x[!inds_ceil] <- floor(exp_x[!inds_ceil])

    return(log(exp_x - 0.5))

}

in_range_fn <- function(x, tolerance = 0.5 *
.Machine$double.eps^0.5) {

    vapply(X = x, FUN = function(x_i) {

        isTRUE(all.equal.numeric(

            x_i,

            log_round_to_integer_plus_0.5_exp(x_i),

            tolerance = tolerance
        ))

    }, FUN.VALUE = TRUE, USE.NAMES = FALSE)

}

discrete_var_names <- unlist(lapply(predictive_vars, function
(predictive_var)

    c(paste0(predictive_var, "_lag", rep(seq(from = 0, to =
max_lag + 52 * max_seasonal_lag), each=2)),

      paste0(predictive_var, "_horizon", rep(1:52, each=2)))))

discrete_var_range_fns <- sapply(

    X = discrete_var_names,

    FUN = function(discrete_var_name) {

    list(a = log_exp_x_minus_0.5,

         b = log_exp_x_plus_0.5,

         in_range = in_range_fn,

         discretizer = log_round_to_integer_plus_0.5_exp)

}, simplify = FALSE, USE.NAMES = TRUE)


## Second step is to actually append the kernel component
descriptions to the

## kernel_components list

kernel_components <- c(kernel_components,

    lapply(vars_and_offsets_groups, function(vars_and_offsets)
{

        lower_trunc_bds <- rep(-Inf, nrow(vars_and_offsets))

        names(lower_trunc_bds) <-
vars_and_offsets$combined_name

        upper_trunc_bds <- rep(Inf, nrow(vars_and_offsets))

        names(upper_trunc_bds) <-
vars_and_offsets$combined_name
```

```
        return(list(

            vars_and_offsets = vars_and_offsets,

            kernel_fn = kcde::log_pdtmvn_mode_centered_kernel,

            rkernel_fn =
kcde::rlog_pdtmvn_mode_centered_kernel,

            theta_fixed = list(

                parameterization = "bw-chol-decomp",

                continuous_vars = NULL,

                discrete_vars = vars_and_offsets$combined_name[

                    vars_and_offsets$combined_name %in%
discrete_var_names],

                discrete_var_range_fns =
discrete_var_range_fns,

                lower = lower_trunc_bds,

                upper = upper_trunc_bds,

                validate_in_support = FALSE

            ),

            theta_est = list("bw"),

            initialize_kernel_params_fn =

                kcde::initialize_params_log_pdtmvn_kernel,

            initialize_kernel_params_args = list(

                sample_size = init_sample_size

            ),

            get_theta_optim_bounds_fn =

                kcde::get_theta_optim_bounds_log_pdtmvn_kernel,

            get_theta_optim_bounds_args = NULL,

            vectorize_kernel_params_fn =

                kcde::vectorize_params_log_pdtmvn_kernel,

            vectorize_kernel_params_args = NULL,
            update_theta_from_vectorized_theta_est_fn =

kcde::update_theta_from_vectorized_theta_est_log_pdtmvn_kernel,
            update_theta_from_vectorized_theta_est_args = NULL
        ))
    })
)

kcde_control <- create_kcde_control(
    X_names = "Index",  # seems to work, no need to supply
as.integer(Index)
    y_names = "DENGUEFR",
    time_name = "Index",
    prediction_horizons = "this parameter is actually unused",
    kernel_components = kernel_components,
    filter_control = NULL,
    crossval_buffer = 365,  # as in the original application
    prediction_inds_not_included = NULL,
    loss_fn = neg_log_score_loss,
    loss_fn_prediction_args = list(
        prediction_type = "distribution",
        log = TRUE),
    loss_args = NULL,
    variable_selection_method = "all_included",
    par_cores = 3
)
##fit the kcde model
#it will take time
runtime <- system.time(
    kcdefit <- kcde(data = DENGUEFRdat[Traindata,],

                    kcde_control = kcde_control)

)
#check the time of fitting the model

kcdefit$runtime <- runtime


#prediction and log scores

predict_kcde_owa <- function(kcdefit, data,
prediction_time_inds,

                             nsamples = 10000)

{

    prediction_target_var <- kcdefit$kcde_control$y_names

    prediction_horizon <- 1  # this is actually a property of
kcdefit


    ## Allocate data frame to store results

    num_rows <- length(prediction_time_inds)

    scores <- data.frame(

        prediction_time_ind = prediction_time_inds,

        observed = data[prediction_time_inds,
prediction_target_var],

        pt_pred = rep(NA_real_, num_rows),

        AE = rep(NA_real_, num_rows),

        log_score = rep(NA_real_, num_rows)

    )

    samples <- matrix(NA_real_, num_rows, nsamples,

                      dimnames = list(prediction_time_inds,
NULL))
```

```
    results_row_ind <- 1L

    for(prediction_time_ind in prediction_time_inds) {

        ## Get index of analysis time in data set

        ## (time from which we predict forward)

        analysis_time_ind <- prediction_time_ind -
prediction_horizon


        ## Compute log score

        observed_prediction_target <-
as.matrix(scores$observed[results_row_ind])

        colnames(observed_prediction_target) <-

            paste0(prediction_target_var, "_horizon",
prediction_horizon)

        scores$log_score[results_row_ind] <-

            kcde::kcde_predict(

                kcde_fit = kcdefit,

                prediction_data =

                    data[seq_len(analysis_time_ind), , drop =
FALSE],

                leading_rows_to_drop = 0L,

                trailing_rows_to_drop = 0L,

                prediction_type = "distribution",

                prediction_test_lead_obs =
observed_prediction_target,

                log = TRUE

            )


        ## Sample from predictive distribution

        samples[results_row_ind,] <-
```

```
            kcde::kcde_predict(
                n = nsamples,
                kcde_fit = kcdefit,
                prediction_data =
                    data[seq_len(analysis_time_ind), , drop =
FALSE],
                leading_rows_to_drop = 0L,
                trailing_rows_to_drop = 0L,
                prediction_type = "sample"
            )

        ## Increment results row
        results_row_ind <- results_row_ind + 1L
    }

    ## Correction by subtracting 0.5
    samples <- samples - 0.5

    ## Compute point predictions
    scores$pt_pred <- apply(samples, 1, median)

    ## Compute absolute error of point prediction
    scores$AE <- abs(scores$pt_pred - scores$observed)

    ## fix orientation of the log-score
    scores$log_score <- -scores$log_score

    ## add DSS
    scores$DSS <- surveillance:::.dss(

        meanP = apply(samples, 1, mean),

        varP = apply(samples, 1, var),

        x = scores$observed)


    return(list(scores = scores,

                pdists = apply(samples, 1, ecdf)))

}


set.seed(161017)

kcdeowa_runtime <- system.time(

    kcdeowa <- predict_kcde_owa(kcdefit = kcdefit, data =
DENGUEFRdat,

                                prediction_time_inds = OWA +
1L)

)
```

3.4.3 results

```
summary(kcdeowa$scores[c("DSS", "log_score", "AE")])
```

```
#visualize

par(mar = c(5,5,1,1))

osaplot(

    quantiles = t(sapply(kcdeowa$pdists, quantile, probs=1:99/100)),

    probs = 1:99/100,

    observed = kcdeowa$scores$observed,

    scores = as.matrix(kcdeowa$scores[c("DSS", "log_score")]),

    start = 1,

    xlab = "", # temporarily remove xlab

    ylim = range(kcdeowa$scores$observed), # changing the y-limits to match your data

    fan.args = list(ln = c(0.1,0.9), rlab = NULL),
    xaxt = "n" # suppress automatic x-axis

)

# Here I extract the years from your dates

score_years <- format(as.Date(DENGUEFRdat$Index[OWA + 1L]), "%Y")

# The unique years to be shown on the x-axis

unique_years <- unique(score_years)

# The positions where the years should be placed on the x-axis

year_positions <- sapply(unique_years, function(year) {

    # For each unique year, we find the first occurrence in
score_years

    which(score_years == year)[1]

})

# Add x-axis with the desired years

axis(1, at = year_positions, labels = unique_years, cex.axis =
0.7)


# Add x-axis label

mtext("Year", side = 1, line = 2.5)
```

```
#plot original vs predicted for 2019 to 2020

library(ggplot2)


# create a new dataframe with dates, observed and predicted values

df <- data.frame(

    Date = data$Time[(OWA[1] + 1):length(data$Time)],

    Observed = kcdeowa$scores$observed,

    Predicted = kcdeowa$scores$pt_pred

)
```

```
# plot observed and predicted values with ggplot
p <- ggplot(df, aes(x = Date)) +

    # observed values as blue area
    geom_ribbon(aes(ymin = 0, ymax = Observed), fill = "blue",
alpha = 0.4) +

    # predicted values as red area
    geom_ribbon(aes(ymin = 0, ymax = Predicted), fill = "red",
alpha = 0.4) +

    # add observed and predicted lines to distinguish between
areas
    geom_line(aes(y = Observed), colour = "blue", linewidth = 1)
+
    geom_line(aes(y = Predicted), colour = "red", linewidth = 1)
+

    # add labels
    labs(x = "Year",
         y = "Value",
         title = "Observed vs Predicted Values (2019-2020)",
         caption = "Blue: Observed | Red: Predicted") +

    # improve the theme
    theme_minimal() +
    theme(
        plot.title  =  element_text(hjust  =  0.5,  face="bold",
size=20),

        axis.title = element_text(face="bold", size=14),

        plot.caption = element_text(hjust = 1, face="italic")

    ) +


    # format x-axis

    scale_x_date(date_breaks = "1 year", date_labels = "%Y")


print(p)
```

3.4.4 evaluation metrics

```
# Calculate Mean Absolute Error (MAE)

mae <- mean(kcdeowa$scores$AE)


# Calculate Root Mean Squared Error (RMSE)

rmse <- sqrt(mean(kcdeowa$scores$AE^2))


# Calculate Mean Absolute Percentage Error (MAPE)

mape <- mean(abs((kcdeowa$scores$observed -
kcdeowa$scores$pt_pred) / kcdeowa$scores$observed)) * 100


# Print the metrics

print(mae)
```

```
print(rmse)
print(mape)
```

```
#Residual Plot

# Calculate residuals

residuals <- kcdeowa$scores$pt_pred - kcdeowa$scores$observed


# Load necessary libraries

library(ggthemes)


# Create a new data frame for plotting

residuals_data <- data.frame(Predicted =
kcdeowa$scores$pt_pred, Residuals = residuals)


# Create the residuals vs predicted values plot

residuals_plot <- ggplot(residuals_data, aes(x = Predicted, y =
Residuals)) +

  geom_point(color = "steelblue", alpha = 0.5, size = 2) +

  geom_smooth(method = "loess", se = FALSE, color =
"firebrick", linetype = "dashed") +

  geom_hline(yintercept = 0, linetype = "dashed", color =
"red") +

  labs(

    x = "Predicted Values",

    y = "Residuals",

    title = "Residuals vs Predicted Values",

    subtitle = "A plot to check the residuals of the model"

  ) +

  theme_minimal() +

  theme(

    plot.title = element_text(size = 18, face = "bold", hjust =
0.5),
    plot.subtitle = element_text(size = 14, hjust = 0.5),

    axis.title = element_text(size = 14, face = "bold")

  )

print(residuals_plot)
```

```
#Forecast Error Distribution:

# Plot histogram of forecast errors

# Ensure the Metrics and ggplot2 packages are installed and loaded

library(Metrics)

library(ggplot2)


# Calculate Mean Absolute Error (MAE)

mae <- mae(kcdeowa$scores$observed, kcdeowa$scores$pt_pred)

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


# Calculate Root Mean Squared Error (RMSE)

rmse <- rmse(kcdeowa$scores$observed, kcdeowa$scores$pt_pred)

print(paste("RMSE: ", rmse))


# Calculate Mean Absolute Percentage Error (MAPE)

mape <- mape(kcdeowa$scores$observed, kcdeowa$scores$pt_pred) *
100

print(paste("MAPE: ", mape, "%"))


# Calculate absolute errors for the plot

kcdeowa$scores$AE      <-      abs(kcdeowa$scores$observed      -
kcdeowa$scores$pt_pred)
```

```
# Plot histogram of forecast errors

ggplot(kcdeowa$scores, aes(x = AE)) +

  geom_histogram(binwidth  =  1,  fill  =  "#2196F3",  color  =
"#0D47A1", alpha = 0.8) +

  labs(

    x = "Absolute Error",

    y = "Frequency",

    title = "Histogram of Forecast Errors",

    subtitle  =  paste("MAE  =",  round(mae,  2),  "  RMSE  =",
round(rmse, 2), " MAPE =", round(mape, 2), "%")

  ) +

  theme_minimal() +

  theme(

    plot.title = element_text(size = 20, face = "bold"),

    plot.subtitle = element_text(size = 14),

    axis.title = element_text(size = 14, face = "bold"),

    axis.text = element_text(size = 12),

    axis.text.x = element_text(angle = 45, vjust = 0.5, hjust =
1),

    panel.grid.major = element_blank(),

    panel.grid.minor = element_blank(),

    legend.position = "none"

  )
```

```
# Plot CDF plot

ggplot(kcdeowa$scores, aes(x = observed)) +

  stat_ecdf(geom = "step", color = "blue", size = 1.5) +
  stat_ecdf(geom = "step", aes(x = pt_pred), color = "red",
size = 1.5) +

  labs(x = "Value", y = "Cumulative Probability") +

  scale_color_manual(values = c("blue", "red"), labels =
c("Observed", "Predicted"))
```

```
# Plot density plot

# Load the required packages

library(ggplot2)

library(ggthemes)



# Create a density plot

density_plot <- ggplot(kcdeowa$scores, aes(x = AE, fill =
"Error")) +

  geom_density(alpha = 0.7, color = "#1C4E80") +

  labs(x = "Absolute Error", y = "Density", fill = "") +

  scale_fill_manual(values = c("#1C4E80")) +

  theme_economist() +

  theme(

    panel.grid.major = element_blank(),

    panel.grid.minor = element_blank(),

    panel.border = element_blank(),

    axis.line = element_line(color = "black"),

    legend.position = "none"

  )

# Apply a theme for a visually stunning appearance

final_plot <- density_plot + theme_few()

# Display the plot
```

```
final_plot
```

```
# Plot box plot

# Load the required packages

library(ggplot2)

library(ggthemes)

# Create a box plot

box_plot <- ggplot(kcdeowa$scores, aes(x = 1, y = AE)) +

  geom_boxplot(fill = "lightblue", color = "black") +

  labs(x = "", y = "Absolute Error") +

  theme_minimal() +

  theme(

    panel.grid.major = element_blank(),

    panel.grid.minor = element_blank(),

    panel.border = element_blank(),

    axis.text.y = element_text(size = 12),

    axis.title.y = element_text(size = 14, face = "bold"),

    plot.title = element_text(size = 20, face = "bold"),

    plot.subtitle = element_text(size = 16),

    plot.caption = element_text(size = 10, hjust = 0)

  )

# Apply a custom theme for a visually stunning appearance

final_plot <- box_plot +

  theme(

    plot.background = element_rect(fill = "#F5F5F5"),

    panel.background = element_rect(fill = "#F5F5F5"),

    axis.line = element_line(color = "black"),
    axis.ticks.y = element_line(color = "black")

  )

# Display the plot

final_plot
```

```
# Plot scatter plot

library(ggplot2)

library(ggthemes)



# Create the scatter plot

scatter_plot <- ggplot(kcdeowa$scores, aes(x = observed, y =
pt_pred)) +

  geom_point(color = "blue", size = 3, alpha = 0.7) +

  geom_abline(slope = 1, intercept = 0, color = "red", linetype
= "dashed", size = 1.2) +

  labs(x = "Observed", y = "Predicted") +

  theme_minimal() +

  theme(

    plot.title = element_text(size = 20, face = "bold"),

    axis.title = element_text(size = 14, face = "bold"),

    axis.text = element_text(size = 12),

    axis.line = element_line(color = "black"),

    panel.grid.major = element_blank(),

    panel.grid.minor = element_blank(),

    panel.border = element_blank(),

    legend.position = "none"

  )
```

```
# Add a title and subtitle

scatter_plot <- scatter_plot +

  ggtitle("Scatter Plot of Observed vs Predicted Values") +

  labs(subtitle = "Comparison of Model Predictions")


# Display the scatter plot

scatter_plot
```

```
# load necessary packages

library(ggplot2)

library(gridExtra)

library(GGally)


# Q-Q plot

qqplot <- ggplot(kcdeowa$scores, aes(sample = observed)) +

  geom_qq() +

  geom_abline(slope = 1, intercept = 0, color = "red") +

  labs(x = "Theoretical Quantiles", y = "Observed Quantiles",

       title = "Q-Q Plot") +

  theme_minimal()


# Residual analysis plot

residuals <- kcdeowa$scores$observed - kcdeowa$scores$pt_pred


residual_plot <- ggplot() +

  geom_point(data        =        data.frame(Predicted        =
kcdeowa$scores$pt_pred,

                               Residuals = residuals),
             aes(x = Predicted, y = Residuals), color = "blue",
size = 3) +

  geom_hline(yintercept = 0, color = "red") +

  labs(x  =  "Predicted",  y  =  "Residuals",  title  =  "Residual
Analysis") +

  theme_minimal()


# Combine plots into a single figure

combined_plot <- grid.arrange(qqplot, residual_plot, nrow = 1)


# Display the combined plot

print(combined_plot)
```

3.5.2.1 HHH4 WITH WEIGHTED SCHEMES

```
library(RColorBrewer)

library(surveillance)

library(devtools)

library(hhh4addon)

library(RColorBrewer)

library(surveillance)

library(devtools)

#the hhh4addon works only with sts structure so we convert

csv_to_sts <- function(file, names, start, end, ...){

    # read data:

    dat <- read.csv2("D:/ECDC_surveillance_data_Dengue.csv",

                     sep = ",", header = TRUE, stringsAsFactors
= FALSE)

    dat$year <- dat$week <- NA

    # handle month 12:

    is_month12 <- which(grepl("m12", dat$time))

    dat <- dat[-is_month12, ]

    # handle time variable:

    for(i in 1:nrow(dat)){

        temp  <-  as.numeric(strsplit(dat$time[i],  "w",  fixed  =
TRUE)[[1]])

        dat$year[i] <- temp[1]

        dat$month[i] <- temp[2]

    }

    dat$time <- NULL

    colnames(dat)[1:length(names)] <- names

    # restrict to selected range
    if( (tail(dat$year, 1) < end[1]) ||
        (tail(dat$year,  1)  ==  end[1])  &  tail(dat$week,  1)  <
end[2]){
        stop("Either start or end is outside of the range of the
provided data.")
    }


    dat <- subset(dat, year >= start[1])
    dat <- subset(dat, !(year == start[1] & month < start[2]))


    dat <- subset(dat, year <= end[1])
    dat <- subset(dat, !(year == end[1] & month > end[2]))


    dat$month <- NULL
    dat$year <- NULL


    # to sts object:
    stsObj <- new("sts", observed = dat, start = start,...)
    return(stsObj)
}
# Load the libraries
library(RColorBrewer)
library(surveillance)
library(hhh4addon)
data <- data.frame(
    Time  =  seq(as.Date("2008/01/01"),  as.Date("2020/12/01"),  by
= "month"),
    NumValue = c(5, 2, 4, 2, 6, 3, 4, 14, 2, 2, 10, 2, 6, 4, 6,
3, 4, 8, 5, 10, 4, 1, 9, 4, 10, 11, 16, 27, 28, 46, 111, 242, 77,
21, 3, 4, 3, 3, 5, 3, 7, 3, 4, 17, 2, 1, 3, 4, 6, 3, 7, 8, 4, 12,
26, 33, 16, 16, 12, 9, 18, 13, 16, 21, 40, 40, 46, 97, 69, 44,
39, 13, 19, 16, 18, 12, 42, 41, 42, 107, 39, 13, 9, 9, 2, 9, 16,
18, 30, 16, 41, 71, 31, 24, 16, 11, 15, 13, 17, 38, 55, 31, 50,
85, 30, 18, 16, 5, 1, 7, 14, 23, 36, 29, 36, 59, 18, 15, 24, 4,
7, 6, 11, 29, 40, 40, 40, 58, 29, 33, 23, 15, 16, 15, 40, 70,
117, 79, 96, 158, 77, 81, 105, 50, 79, 92, 85, 34, 29, 40, 70,
342, 233, 224, 115, 19)

)

# Convert Time to date type and create sts object

data$Time <- as.Date(data$Time)

data_sts <- sts(observed = matrix(data$NumValue, ncol = 1), epoch
= data$Time)



# Ensure there are no missing or infinite values in the data

if(any(is.na(data_sts@observed))){

    stop("The observed data contains NA values")

}

if(any(is.infinite(data_sts@observed))){

    stop("The observed data contains Inf values")

}



# Define controls for different lag weighting schemes

ctrls_data <- list()

ctrls_data$ar1 <- list(

    ar = list(f = addSeason2formula(f = ~ 1, S = 2), lag = 1),

    end = list(f = addSeason2formula(f = ~ 1, S = 1, period =
12)),

    family = "NegBin1"

)
```

```
ctrls_data$geom  <-  ctrls_data$ar1;  ctrls_data$geom$funct_lag  =
geometric_lag; ctrls_data$geom$max_lag <- 10

ctrls_data$pois  <-  ctrls_data$ar1;  ctrls_data$pois$funct_lag  =
poisson_lag; ctrls_data$pois$max_lag <- 10

ctrls_data$ar2  <-  ctrls_data$ar1;  ctrls_data$ar2$funct_lag  =
ar2_lag; ctrls_data$ar2$max_lag <- 2

ctrls_data$lin  <-  ctrls_data$ar1;  ctrls_data$lin$funct_lag  =
linear_lag; ctrls_data$lin$max_lag <- 10

ctrls_data$unres <- ctrls_data$ar1; ctrls_data$unres$funct_lag =
unrestricted_lag; ctrls_data$unres$max_lag <- 10


# Fit models

ctrls_data$ar1$subset <- 3:length(data$NumValue)

fit_data_ar1 <- hhh4(data_sts, ctrls_data$ar1)


# Fit models varying order p

fits_data_vary_max_lag <- list()

fits_data_vary_max_lag$geom[[1]]                                <-
fits_data_vary_max_lag$pois[[1]]                                <-
fits_data_vary_max_lag$lin[[1]]                                 <-
fits_data_vary_max_lag$unres[[1]] <- fit_data_ar1


# Adjust the max lag and run the model fits

for(max_lag in 2:10){

    # Update the subset start point to be greater than max_lag

    subset_start_point <- max_lag + 1


    ctrls_data$geom$max_lag <- max_lag

    ctrls_data$geom$subset                                      <-
subset_start_point:length(data$NumValue)
```

```
    fits_data_vary_max_lag$geom[[max_lag]]                         <-
profile_par_lag(data_sts, ctrls_data$geom)


    ctrls_data$pois$max_lag <- max_lag

    ctrls_data$pois$subset                                         <-
subset_start_point:length(data$NumValue)

    fits_data_vary_max_lag$pois[[max_lag]]                         <-
profile_par_lag(data_sts, ctrls_data$pois)


    ctrls_data$lin$max_lag <- max_lag

    ctrls_data$lin$subset                                          <-
subset_start_point:length(data$NumValue)

    fits_data_vary_max_lag$lin[[max_lag]]                          <-
profile_par_lag(data_sts, ctrls_data$lin)


    ctrls_data$unres$max_lag <- max_lag

    ctrls_data$unres$subset                                        <-
subset_start_point:length(data$NumValue)

    start_par_lag    <-    if(max_lag    ==    2)    NULL    else
c(fits_data_vary_max_lag$unres[[max_lag - 1]]$par_lag, -3)

    fits_data_vary_max_lag$unres[[max_lag]]                        <-
profile_par_lag(data_sts,   ctrls_data$unres,   start_par_lag   =
start_par_lag)


    print(max_lag)

}

AICs_vary_max_lag_data <- matrix(ncol = 5, nrow = 10, dimnames =
list(NULL, c("max_lag", "geom", "pois", "lin", "unres")))

AICs_vary_max_lag_data[, "max_lag"] <- 1:10

AICs_vary_max_lag_data[,                "geom"]                <-
unlist(lapply(fits_data_vary_max_lag$geom, AIC))
```

```
AICs_vary_max_lag_data[,                "pois"]                <-
unlist(lapply(fits_data_vary_max_lag$pois, AIC))

AICs_vary_max_lag_data[,                 "lin"]                <-
unlist(lapply(fits_data_vary_max_lag$lin, AIC))

AICs_vary_max_lag_data[,               "unres"]                <-
unlist(lapply(fits_data_vary_max_lag$unres, AIC))



# Save the results to a csv file

write.csv(AICs_vary_max_lag_data,            file              =
"D:/AICs_data_vary_max_lag.csv", row.names = FALSE)
```

3.5.2.2

```
# Define controls for different lag weighting schemes

ctrls_data <- list()

ctrls_data$ar1 <- list(

    ar = list(f = addSeason2formula(f = ~ 1, S = 2), lag = 1),

    end = list(f = addSeason2formula(f = ~ 1, S = 1, period =
12)),

    subset = 133:(10*12),

    family = "NegBin1"

)



# Update the lag weightings based on different schemes

ctrls_data$geom <- ctrls_data$ar1; ctrls_data$geom$funct_lag <-
geometric_lag; ctrls_data$geom$max_lag <- 5

ctrls_data$pois <- ctrls_data$ar1; ctrls_data$pois$funct_lag <-
poisson_lag; ctrls_data$pois$max_lag <- 5

ctrls_data$ar2  <-  ctrls_data$ar1;  ctrls_data$ar2$funct_lag  <-
ar2_lag; ctrls_data$ar2$max_lag <- 5

ctrls_data$lin  <-  ctrls_data$ar1;  ctrls_data$lin$funct_lag  <-
linear_lag; ctrls_data$lin$max_lag <- 5

ctrls_data$unres  <-  ctrls_data$ar1;  ctrls_data$unres$funct_lag
<- unrestricted_lag; ctrls_data$unres$max_lag <- 4



# Based on serial interval:
```

```
wgts_siraj <- c(0.001, 0.999*c(0.2, 0.425, 0.25, 0.125)) # cannot
asign probability 0 to lag 1

par_lag_siraj <- log(wgts_siraj[-1]/(1 - sum(wgts_siraj[-1])))

ctrls_data$siraj <- ctrls_data$geom; ctrls_data$siraj$funct_lag
<- unrestricted_lag

ctrls_data$siraj$par_lag <- par_lag_siraj


# Fit models:

fits_data <- list()


fits_data$ar1 <- hhh4(data_sts, ctrls_data$ar1)


# Adjust the subset start point to be greater than max_lag and
fit the models

subset_start_point <- max(6, ctrls_data$ar2$max_lag) + 1

ctrls_data$ar2$subset                                          <-
subset_start_point:length(data$NumValue)

fits_data$ar2 <- profile_par_lag(data_sts, ctrls_data$ar2)


subset_start_point <- max(6, ctrls_data$geom$max_lag) + 1

ctrls_data$geom$subset                                         <-
subset_start_point:length(data$NumValue)

fits_data$geom <- profile_par_lag(data_sts, ctrls_data$geom)


subset_start_point <- max(6, ctrls_data$pois$max_lag) + 1

ctrls_data$pois$subset                                         <-
subset_start_point:length(data$NumValue)

fits_data$pois <- profile_par_lag(data_sts, ctrls_data$pois)


subset_start_point <- max(6, ctrls_data$lin$max_lag) + 1
```

```
ctrls_data$lin$subset                                                    <-
subset_start_point:length(data$NumValue)

fits_data$lin <- profile_par_lag(data_sts, ctrls_data$lin)


subset_start_point <- max(6, ctrls_data$unres$max_lag) + 1

ctrls_data$unres$subset                                                  <-
subset_start_point:length(data$NumValue)

fits_data$unres <- profile_par_lag(data_sts, ctrls_data$unres)


fits_data$siraj <- hhh4_lag(data_sts, ctrls_data$siraj)


# Compute AICs:

AICs_data <- lapply(fits_data, AIC)

AICs_vary_max_lag_dengue                                                 <-
read.csv("D:/AICs_data_vary_max_lag.csv")

ref <-AICs_vary_max_lag_dengue[1, "geom"]

AICs_vary_max_lag_dengue          <-AICs_vary_max_lag_dengue-ref
library(RColorBrewer)


# Define the color palette

n_models <- 6  # Number of models

palette_name <- "Set1"  # Color palette name

cols_models_dengue <- brewer.pal(n_models, palette_name)

print(fits_data$pois$distr_lag[1:5])

print(fits_data$lin$distr_lag[1:5])

print(fits_data$geom$distr_lag[1:5])

print(fits_data$unres$distr_lag[1:5])

print(fits_data$siraj$distr_lag[1:5])

plot(1:10, AICs_vary_max_lag_dengue[1:10, "geom"], type = "b",
     xlab = "p", ylab  = "improvement in AIC", pch = 15, cex =
0.9,

     ylim = range(AICs_vary_max_lag_dengue[,2:5]))

# create figure:

par(mfrow = c(1, 2), las = 1, mar = c(4, 4, 0.5, 1))

#We can see the improvement in AIC per p


# Plot AICs for different values of p

plot(2:10, AICs_vary_max_lag_dengue[2:10, "geom"], type = "b",

     xlab = "p", ylab  = "improvement in AIC", pch = 15, cex =
0.9,

     ylim = range(AICs_vary_max_lag_dengue[,2:5]))

lines(2:10, AICs_vary_max_lag_dengue[2:10, "pois"],

      type = "b", pch = 15, cex = 0.9)

lines(2:10, AICs_vary_max_lag_dengue[2:10, "lin"],

      type = "b", pch = 15, cex = 0.9)

lines(2:10, AICs_vary_max_lag_dengue[2:10, "unres"],

      type = "b", pch = 15, cex = 0.9)
```

3.5.3

```
library(surveillance)

library(hhh4addon)


# create folder structure if necessary:
```

```
list.dirs()

dir.create("model_fits")

for(lag_structure in c("ar1", "geom", "pois", "lin", "unres",
"end", "siraj")){

    dir.create(paste0("model_fits/data_", lag_structure))

}


plot(data_sts)

names_lag_structures <- c("end", "ar1", "pois", "lin", "geom",
"unres", "siraj")




# define controls for different lag weighting schemes:

max_lag_param <- 5

max_lag_unres <- 4

ctrls_data <- list(

    ar = list(f = addSeason2formula(f = ~ 1, S = 2), lag = 1),

    end = list(f = addSeason2formula(f = ~ 1, S = 1, period =
12)),

    family = "NegBin1",

    max_lag  = max_lag_param

)


ctrls_data_end <- list(

    ar = list(f = ~-1),

    end = list(f = addSeason2formula(f = ~ 1, S = 1, period =
12)),

    family = "NegBin1"

)
```

```
# timepoints for which to fit models:

tps <- (133):(155)


# for the linear lag we adopt a grid-based optimization

vals_kappa_lin <- 1:99/100

vals_par_lag_lin <- log(vals_kappa_lin/(1 - vals_kappa_lin))


# weighting scheme based on serial intervals taken from Siraj:

wgts_siraj <- c(0.001, 0.999*c(0.2, 0.425, 0.25, 0.125)) # cannot
asign probability 0 to lag 1.

par_lag_siraj <- log(wgts_siraj[-1]/(1 - sum(wgts_siraj[-1])))

# unrestricted_lag(par_lag_siraj, 1, 5) # works


# fit models for all time points during the evaluation period

# we originally also experimented with models which have a min_lag
larger than 1,

# i.e. force the first couple of weights to 0, but this did not
yield improvements.

# therefore only run min_lag = 1

for(min_lag in 1:1){


    # adapt control settings to min_lag:

    ctrls_data_temp <- ctrls_data

    ctrls_data_temp$subset         <-         (min_lag         +
max_lag_param):nrow(data_sts)

    ctrls_data_temp$ar$lag <- ctrls_data_temp$ne$lag <- min_lag

    ctrls_data_temp$min_lag <- min_lag

    ctrls_data_temp$max_lag <- min_lag + max_lag_param - 1
    # define controls with different weighting schemes:

    ctrls_data_pois_temp           <-           ctrls_data_temp;
ctrls_data_pois_temp$funct_lag <- poisson_lag

    ctrls_data_lin_temp            <-           ctrls_data_temp;
ctrls_data_lin_temp$funct_lag <- linear_lag


    ctrls_data_unres_temp          <-           ctrls_data_temp;
ctrls_data_unres_temp$funct_lag <- unrestricted_lag

    ctrls_data_unres_temp$max_lag <- min_lag + max_lag_unres - 1


    ctrls_data_siraj_temp          <-           ctrls_data_temp;
ctrls_data_siraj_temp$funct_lag <- unrestricted_lag

    ctrls_data_siraj_temp$par_lag <- par_lag_siraj


    # run over timepoints in validation period:

    for(ind in tps){


        # steer subset via NAs:

        data_sts_temp <- data_sts

        if(ind < nrow(data_sts)){

            data_sts_temp@observed[(ind                          +
1):nrow(data_sts_temp), ] <- NA

        }


        # fit endemic-only model

        fit_data_end_temp <- hhh4(data_sts_temp, ctrls_data_end)
        # fit ar1 model:

        fit_data_ar1_temp <- hhh4(data_sts_temp,
ctrls_data_temp)




        # fit model with geometric lags:

        start_par_lag <- ifelse(ind == tps[1], 0.5,
fit_data_geom_temp$par_lag)

        fit_data_geom_temp <- profile_par_lag(data_sts_temp,
ctrls_data_temp,

                                                  start_par_lag =
start_par_lag)




        # fit model with Poisson lags:

        start_par_lag <- ifelse(ind == tps[1], 0.5,
fit_data_pois_temp$par_lag)

        fit_data_pois_temp <- profile_par_lag(data_sts_temp,
ctrls_data_pois_temp,

                                                  start_par_lag =
start_par_lag)




        # fit model with linear lags using grid-based
optimization:

        fit_data_lin_temp <- fit_par_lag(data_sts_temp,
ctrls_data_lin_temp,

                                             range_par =
vals_par_lag_lin, use_update = FALSE)$best_mo

        start_par_lag <- ifelse(ind == tps[1], 0.5,
fit_data_lin_temp$par_lag)
```

```
        fit_data_lin_temp    <-    profile_par_lag(data_sts_temp,
ctrls_data_lin_temp,
                                                            start_par_lag   =
start_par_lag)



        # fit model with unconstrained weights:
        start_par_lag <- if(ind == tps[1]){
            rep(0.5, max_lag_unres - 1)
        }else{
            fit_data_unres_temp$par_lag
        }
        fit_data_unres_temp   <-   profile_par_lag(data_sts_temp,
ctrls_data_unres_temp,
                                                              start_par_lag =
start_par_lag)
        # fit model with weights taken from Siraj 2017:
        fit_data_siraj_temp      <-      hhh4_lag(data_sts_temp,
ctrls_data_siraj_temp)



        print(ind)
    }
}
```

3.5.4

```
#fuction
forecasting_nStepAhead <- function(fit, stsObj, tp_cond, horizon,
n_sim){
  is_hhh4_lag <- (class(fit)[1] == "hhh4lag") # check if hhh4 or
hhh4lag object

  n_units <- ncol(stsObj@observed)


  fit$stsObj <- stsObj # replace stsObj in fit by original stsObj
(as subsets steered via NA)

  max_lag <- ifelse(is_hhh4_lag, fit$max_lag, 1)

  # wrapper function to handle apply commands

  dnb <- function(mu, size, x){

    dnbinom(x = x, mu = mu, size = size)

  }


  # for horizon h = 1 no simulation is necessary:

  if(horizon <= 1){

    forecast <-

      if(is_hhh4_lag){

        suppressMessages(

          oneStepAhead_hhh4lag(fit, tp = rep(tp_cond + horizon -
1, 2), type = "final")

        )

      }else{

        oneStepAhead(fit, tp = rep(tp_cond + horizon - 1, 2),
type = "final")

      }

    log_score <- scores(forecast, which = "logs")

  }else{ # for horizons >= 2 we need to simulate:

    tp_to_simulate <- tp_cond + 1:(horizon)

    # need to do one more time point due to bug in surveillance
    # generate sample paths:

    sims <- simulate(fit, subset = tp_to_simulate,

                     y.start  =  fit$stsObj@observed[tp_cond  -
(max_lag:1) + 1, , drop = FALSE],

                     simplify = TRUE, nsim = n_sim)

    sims[length(tp_to_simulate),,]  <-  stsObj@observed[tp_cond +
horizon, ] # put true observation back


    # obtain log scores for each sample path:

    sim_log_scores <- sim_mu <- matrix(NA, ncol = n_units, nrow
= n_sim)

    for(i in 1:n_sim){

      fit_temp <- fit

      fit_temp$stsObj@observed[tp_to_simulate, ] <- sims[,,i] #
plug simulated path into fit object


      # do one-step ahead forecast given the simulated path:

      forecast_temp <-

        if(is_hhh4_lag){

          suppressMessages(

            oneStepAhead_hhh4lag(fit_temp,  tp  =  rep(tp_cond  +
horizon - 1, 2), type = "final")

          )

        }else{

          oneStepAhead(fit_temp, tp = rep(tp_cond + horizon - 1,
2), type = "final")

        }


      #  store  the  predictive  means  and  size  (in  the  NB
distribution) under the respective
```

```
      # simulated path:

      sim_mu[i, ] <- forecast_temp$pred

      if(i == 1) size <- rep(exp(forecast_temp$psi), length.out
= n_units) # stays the same in all iterations



      # store the log score obtained under the respective
simulated path:

      scores_temp <- scores(forecast_temp, individual = TRUE)

      if(ncol(stsObj@observed) == 1){

        sim_log_scores[i, ] <- scores_temp["logs"]

      }else{

        sim_log_scores[i, ] <- scores_temp[, "logs"]

      }

    }

    # average over log scores obtained with different sample
paths:

    log_score <- -log(mean(exp(-
rowSums(sim_log_scores))))/ncol(sim_log_scores)

  }



  # extract characteristics of predictive distributions and the
scores:

  templ_vect <- numeric(n_units); names(templ_vect) <-
colnames(stsObj@observed)

  unit_wise_log_score <- pred_mean <- pred_var <- pred_lb50 <-
pred_ub50 <-

    pred_lb95 <- pred_ub95 <- unit_wise_pit_l <- unit_wise_pit_u
<- templ_vect



  for(unit in 1:n_units){
    # get observed value:

    obs_unit_temp <- stsObj@observed[tp_cond + horizon, unit]



    if(horizon == 1){ # for horizon 1 can extract directly from
return of oneStepAhead

      mu_unit <- forecast$pred[unit]

      size_unit   <-   rep(exp(forecast$psi),   length.out   =
n_units)[unit] # catch case of NegBin1

      support_temp <- 0:max(qnbinom(p = 0.99,

                                    mu = mu_unit,

                                    size = size_unit),

                            obs_unit_temp + 2, na.rm = TRUE)

      pred_densities_temp   <-   dnbinom(support_temp,   size   =
size_unit, mu = mu_unit)



      pred_mean[unit] <- mu_unit

      pred_var[unit] <- mu_unit + 1/size_unit*mu_unit^2

    }else{ # otherwise need to average over samples:

      support_temp <- 0:max(qnbinom(p = 0.99,

                                    mu = max(sim_mu[, unit]),

                                    size = size[unit]),

                            obs_unit_temp + 2, na.rm = TRUE)



      pred_densities_temp  <-  rowMeans(sapply(sim_mu[,  unit],
dnb,

                                               x = support_temp,

                                               size = size[unit]))

      pred_mean[unit] <- sum(support_temp*pred_densities_temp)
      pred_var[unit]  <-  sum(support_temp^2*pred_densities_temp)
- pred_mean[unit]^2

    }


    # compute limits of prediction intervals:

    pred_cumul_distr_temp <- cumsum(pred_densities_temp)

    pred_lb50[unit] <- min(which(pred_cumul_distr_temp >= 0.25))
- 1 # -1 bc support includes 0

    pred_ub50[unit] <- max(which(pred_cumul_distr_temp <= 0.75))
# no -1  as we want to be slightly conservative

    pred_lb95[unit]   <-   min(which(pred_cumul_distr_temp   >=
0.025)) - 1

    pred_ub95[unit]   <-   max(which(pred_cumul_distr_temp   <=
0.975))


    # compute unit-wise log scores:

    unit_wise_log_score[unit] <- -log(pred_densities_temp[

      obs_unit_temp + 1

      ])


    # and unit-wise PIT value:

    if(!is.na(obs_unit_temp)){

      unit_wise_pit_l[unit] <- if(obs_unit_temp == 0){

        0

      }else{

        pred_cumul_distr_temp[obs_unit_temp]  #  +  1  bc  support
includes 0

      }

      unit_wise_pit_u[unit]                                      <-
pred_cumul_distr_temp[obs_unit_temp + 1]
```

```
    }else{

      unit_wise_pit_l[unit] <- unit_wise_pit_u[unit] <- NA

    }

  }


  # return object:

  return(list(pred_mean = pred_mean, pred_var = pred_var,

              lb50 = pred_lb50, ub50 = pred_ub50,

              lb95 = pred_lb95, ub95 = pred_ub95,

              obs = stsObj@observed[tp_cond + horizon, ],

              unit_wise_log_score = unit_wise_log_score,

              unit_wise_pit_l = unit_wise_pit_l,

              unit_wise_pit_u = unit_wise_pit_u,

              multiv_log_score = log_score))

}
```

3.5.5

```
max_horizon <- 8

tps <- (133-max_horizon):(155)

n_units <- ncol(dengueSJ@observed)#data_sts?

names_lag_structures <- c("ar1", "pois", "lin", "geom", "unres",
"end", "siraj")


dir.create("logS")

dir.create("forecasts")
```

```
templ_df      <-      data.frame(data_set      =      rep("data",
n_units*max_horizon*length(tps)),

                       unit = 1,

                       prediction_horizon = NA_integer_,

                       lag_structure = "NA",

                       prediction_time = NA_integer_,

                       pred_mean = NA, pred_var = NA,

                       lb50 = NA, ub50 = NA,

                       lb95 = NA, ub95 = NA,

                       obs = NA,

                       unit_wise_log_score = NA,

                       unit_wise_pit_l = NA,

                       unit_wise_pit_u = NA,

                       multiv_log_score = NA)

templ_df$lag_structure <- as.character(templ_df$lag_structure)


results_detailed_data <- list()   # Initialization here

logS_data <- list()               # Initialization here


for(lag_structure in names_lag_structures){


    results_detailed_data[[lag_structure]] <- templ_df

    logS_data[[lag_structure]] <- matrix(ncol = max_horizon, nrow
=   length(tps),   dimnames   =   list(paste0("t_cond=",   tps),
paste0("h", 1:max_horizon)))


    print(lag_structure)


    for(ind in tps){
        set.seed(ind)


        if (lag_structure == 'ar1') {

            fit_data_temp <- fit_data_ar1_temp

        } else if (lag_structure == 'end') {

            fit_data_temp <- fit_data_end_temp

        } else if (lag_structure == 'geom') {

            fit_data_temp <- fit_data_geom_temp

        } else if (lag_structure == 'lin') {

            fit_data_temp <- fit_data_lin_temp

        } else if (lag_structure == 'pois') {

            fit_data_temp <- fit_data_pois_temp

        } else if (lag_structure == 'siraj') {

            fit_data_temp <- fit_data_siraj_temp

        }


        fit_data_temp$stsObj <- data_sts


        for(horizon in 1:max_horizon){

            if(horizon <= nrow(data_sts@observed) - ind){

                capture.output(pred_temp                       <-
forecasting_nStepAhead(fit_data_temp,  stsObj  =  data_sts,  tp  =
ind, horizon = horizon, n_sim = 1000))


                ind_row                                         <-
min(which(is.na(results_detailed_data[[lag_structure]]$predicti
on_horizon)))
results_detailed_data[[lag_structure]]$prediction_horizon[ind_r
ow] <- horizon
results_detailed_data[[lag_structure]]$lag_structure[ind_row] <-
lag_structure


results_detailed_data[[lag_structure]]$prediction_time[ind_row]
<- ind

                results_detailed_data[[lag_structure]][ind_row,
names(pred_temp)] <- pred_temp



                logS_data[[lag_structure]][paste0("t_cond=",
ind), horizon] <- pred_temp$multiv_log_score

            }

        }

        print(ind)

    }
#write our results to access them later

    write.csv(results_detailed_data[[lag_structure]],
paste0("D:/detailed", lag_structure, ".csv"))

    write.csv(logS_data[[lag_structure]],       paste0("D:/log",
lag_structure, ".csv"))

}
```

3.5.6 RESUTLS

```
setwd("dengue")



library(surveillance)

names_lag_structures <- c("ar1", "pois", "lin", "geom",
"unres", "siraj")



source("../auxiliary_functions.R")
source("../basic_settings.R")



# compute the mean log scores for our forecasts:

logS_data <- list()

mean_logS_data <- matrix(NA, ncol = 6, nrow = 8,

                           dimnames = list(paste0("h", 1:8),

                                           c("ar1", "pois",
"lin", "geom",

                                             "unres",
"siraj")))

for(lag_structure in colnames(mean_logS_data)){

    # read in results (generated in evaluate_logS_data.R)

    logS_data_temp <- read.csv(paste0("D:", lag_structure,
".csv"))


    logS_data[[lag_structure]] <- matrix(nrow = 2*12, ncol = 8,

                                         dimnames =
list(paste("t=", (132 + 1):(156)),


paste0("h", 1:8)))
```

```
    # extract scores and for different horizons and store in
list logS_data

    for(horizon in 1:8){

        logS_data[[lag_structure]][, paste0("h", horizon)] <-

            logS_data_temp[seq(from = 10 - horizon, length.out
= 2*12), paste0("h", horizon)]

    }


    # evaluate and store mean logS
    mean_logS_data[, lag_structure] <-
colMeans(logS_data[[lag_structure]])

}



# meaningful column names:

colnames(mean_logS_data) <- paste0("log_",
colnames(mean_logS_data))



##CORRECTED PART

logS_data <- list()

names_lag_structures <- c("ar1", "pois", "lin", "geom",
"unres", "siraj")

mean_logS_data <- matrix(NA, ncol =
length(names_lag_structures), nrow = 8,

                         dimnames = list(paste0("h", 1:8),
paste0("log_", names_lag_structures)))



for(lag_structure in names_lag_structures){

    # read in results (generated in evaluate_logS_data.R)

    logS_data_temp <- read.csv(paste0("D:/log_", lag_structure,
".csv"))
```

```

    logS_data[[lag_structure]] <- logS_data_temp



    # evaluate and store mean logS

    mean_logS_data[, paste0("log_", lag_structure)] <-
colMeans(logS_data[[lag_structure]][,2:9], na.rm = TRUE)

}

# meaningful column names:
```

```
colnames(mean_logS_data) <- paste0("hhh4_",
colnames(mean_logS_data))
```

##DETAILED CSV FILE

```
# You need to have 'Metrics' package installed for mae, rmse,
mape functions and 'DescTools' for mad.

library(Metrics)

library(DescTools)


evaluation_results <- list()


for(lag_structure in names_lag_structures){


    # Read detailed CSV files

    df <- read.csv(paste0("D:/detailed", lag_structure, ".csv"))


    df <- df[!is.na(df$pred_mean),] # Remove NAs


    mae_val <- mae(df$obs, df$pred_mean)

    rmse_val <- rmse(df$obs, df$pred_mean)

    residuals <- df$obs - df$pred_mean

    sum_squares_residuals <- sum(residuals^2)

    total_sum_squares <- sum((df$obs - mean(df$obs))^2)

    r_squared <- 1 - (sum_squares_residuals / total_sum_squares)


    # Additional metrics

    mape_val <- mape(df$obs, df$pred_mean)

    mad_val <- mad(df$obs - df$pred_mean)
    # More metrics

    mse_val <- mean((df$pred_mean - df$obs)^2)

    mpe_val <- mean((df$pred_mean - df$obs) / df$obs)

    md_val <- DescTools::MeanAD(df$obs, df$pred_mean)

    mape_val <- mean(abs((df$pred_mean - df$obs) / df$obs))


    evaluation_results[[lag_structure]] <- list(MAE = mae_val,
RMSE = rmse_val, R_Squared = r_squared,

                                                MAPE = mape_val,
MAD = mad_val, MSE = mse_val,

                                                MPE = mpe_val,
MD = md_val, MAPE = mape_val)

}

# Convert the evaluation results to a data frame

eval_df <- data.frame(do.call(rbind, evaluation_results))


# Make the rownames a column in the dataframe

eval_df$Model <- rownames(eval_df)


# Reorder the columns

eval_df <- eval_df[, c(ncol(eval_df), 1:(ncol(eval_df)-1))]


# Show the results in a table

print(eval_df)
```

##LOG SCORES CSV FILE

```
log_evaluation_results <- list()
```

```
for(lag_structure in names_lag_structures){


    # Read log CSV files

    df <- read.csv(paste0("D:/log_", lag_structure, ".csv"))


    # Compute statistics for each horizon

    for(i in 1:8){

        column_name <- paste0("h",i)


        mean_val <- mean(df[[column_name]], na.rm = TRUE)

        median_val <- median(df[[column_name]], na.rm = TRUE)

        sd_val <- sd(df[[column_name]], na.rm = TRUE)

        iqr_val <- IQR(df[[column_name]], na.rm = TRUE)


        log_evaluation_results[[paste0(lag_structure,       "_",
column_name)]] <- list(Mean = mean_val, Median = median_val,


SD = sd_val, IQR = iqr_val)

    }

}

# Convert the evaluation results to a data frame

log_eval_df             <-             data.frame(do.call(rbind,
log_evaluation_results))


# Make the rownames a column in the dataframe

log_eval_df$Model_Horizon <- rownames(log_eval_df)


# Reorder the columns
```

```
log_eval_df      <-      log_eval_df[,      c(ncol(log_eval_df),
1:(ncol(log_eval_df)-1))]


# Show the results in a table

print(log_eval_df)
```

```
# Boxplot: Distribution of Predicted Means for each
#model

boxplot <- ggplot() +

  theme_minimal() +

  labs(x = "Model", y = "Predicted Mean", fill = "Model") +

  ggtitle("Distribution of Predicted Means for each Model")


df_combined <- do.call(rbind, detailed_data_list)

df_combined$lag_structure  <-  factor(df_combined$lag_structure,
levels = names_lag_structures)
```

```
boxplot <- boxplot +
  geom_boxplot(data = df_combined, aes(x = lag_structure, y =
pred_mean, fill = lag_structure))


boxplot
```

```
#scatter
library(ggplot2)


# List of lag structures
names_lag_structures <- c("ar1", "pois", "lin", "geom", "unres",
"siraj")


# Create an empty list to store the dataframes
detailed_data_list <- list()
# Read the detailed data for each lag structure
for (lag_structure in names_lag_structures) {
  file_name <- paste0("D:/detailed", lag_structure, ".csv")
  df <- read.csv(file_name)
  detailed_data_list[[lag_structure]] <- df
}
# Scatter plot: Observed vs. Predicted Mean for each model
scatter_plot <- ggplot() +
  theme_minimal() +
  labs(x = "Observed", y = "Predicted Mean", color = "Model") +
  ggtitle("Observed vs. Predicted Mean")
```

```
for (lag_structure in names_lag_structures) {

  df <- detailed_data_list[[lag_structure]]

  scatter_plot <- scatter_plot +

    geom_point(data = df, aes(x = obs, y = pred_mean, color =
lag_structure))

}

scatter_plot
```

```
library(ggplot2)

##line plot

# List of lag structures

names_lag_structures <- c("ar1", "pois", "lin", "geom", "unres",
"siraj")


# Create an empty list to store the dataframes

detailed_data_list <- list()


# Read the detailed data for each lag structure

for (lag_structure in names_lag_structures) {

    file_name <- paste0("D:/detailed", lag_structure, ".csv")

    df <- read.csv(file_name)

    detailed_data_list[[lag_structure]] <- df

}


# Line graph: Predicted Mean vs. Prediction Horizon for each model

line_plot <- ggplot() +

    theme_minimal() +
    labs(x = "Prediction Horizon", y = "Predicted Mean", color =
"Model") +
    ggtitle("Predicted Mean vs. Prediction Horizon")


for (lag_structure in names_lag_structures) {
    df <- detailed_data_list[[lag_structure]]


    line_plot <- line_plot +
        geom_line(data = df, aes(x = prediction_horizon, y =
pred_mean, color = lag_structure)) +
        geom_point(data = df, aes(x = prediction_horizon, y =
pred_mean, color = lag_structure, shape = lag_structure))
}


line_plot
```

3.5.7 NAÏVE MODEL

```
naive_forecast_glmnb <- function(ts, t_cond, max_horizon = 1,
freq = 12){


  dummies_season <- as.factor(rep(1:freq, length.out =
length(ts)))


  # get training data:
  subset_training <- 1:t_cond
  dat_training <- ts[subset_training]
  dummies_season_training <- dummies_season[subset_training]


  # fit model:
```

```
  fit_nb <- glm.nb(dat_training ~ dummies_season_training)


  # subset test:

  subset_test <- t_cond + (1:max_horizon)

  dummies_season_test <- dummies_season[subset_test]


  # predict:

  mu <- predict.glm(fit_nb,

                  newdata = data.frame(dummies_season_training
= dummies_season_test),

                  type = "response")


  size <- fit_nb$theta


  pred_lb50 <- qnbinom(0.25, mu = mu, size = size)

  pred_ub50 <- qnbinom(0.75, mu = mu, size = size)

  pred_lb95 <- qnbinom(0.025, mu = mu, size = size)

  pred_ub95 <- qnbinom(0.975, mu = mu, size = size)


  obs <- ts[t_cond + (1:max_horizon)]


  unit_wise_log_score <- -dnbinom(x = obs, mu = mu, size = size,
log = TRUE)

  multiv_log_score <- NA


  return(list(prediction_time = t_cond,

              prediction_horizon = 1:max_horizon,

              pred_mean = mu, pred_var = mu + mu^2/size,
              lb50 = pred_lb50, ub50 = pred_ub50,
              lb95 = pred_lb95, ub95 = pred_ub95,
              obs = obs,
              unit_wise_log_score = unit_wise_log_score,
              multiv_log_score = multiv_log_score))
}


# function to obtain naive seasonal forecasts as suggesed by Leo:
naive_forecast_glmnb_multiv <- function(ts, t_cond, max_horizon
= 1, freq = 12){


  n_units <- ncol(ts)
  n_timepoints <- nrow(ts)


  dummies_season  <-  matrix(as.factor(rep(1:freq,  length.out  =
length(ts))),
                           ncol = n_units, nrow = n_timepoints)
  dummies_regions <- matrix(as.factor(1:n_units),
                            ncol = n_units, nrow = n_timepoints,
                            byrow = TRUE)


  # get training data:
  subset_training <- 1:t_cond


  dat_training <- as.vector(ts[subset_training, ])
  dummies_season_training                                      <-
as.vector(dummies_season[subset_training, ])
  dummies_regions_training                                     <-
as.vector(dummies_regions[subset_training, ])
```

```
  # fit model:

  fit_nb  <-  glm.nb(dat_training  ~  dummies_season_training  +
dummies_regions_training)




  # subset test:

  subset_test <- t_cond + (1:max_horizon)

  dummies_season_test  <-  as.vector(dummies_season[subset_test,
])

  dummies_regions_test <- as.vector(dummies_regions[subset_test,
])


  # predict:

  mu <- predict.glm(fit_nb,

                    newdata = data.frame(dummies_season_training
= dummies_season_test,


dummies_regions_training = dummies_regions_test),

                    type = "response")


  size <- fit_nb$theta


  pred_lb50 <- qnbinom(0.25, mu = mu, size = size)

  pred_ub50 <- qnbinom(0.75, mu = mu, size = size)

  pred_lb95 <- qnbinom(0.025, mu = mu, size = size)

  pred_ub95 <- qnbinom(0.975, mu = mu, size = size)


  obs <- as.vector(ts[subset_test, ])
  unit_wise_log_score <- -dnbinom(x = obs, mu = mu, size = size,
log = TRUE)

  matr_unit_wise_log_score <- matrix(unit_wise_log_score, ncol =
n_units)

  multiv_log_score  <-  rep(rowMeans(matr_unit_wise_log_score),
n_units)


  return(list(prediction_time = t_cond,

              unit = rep(1:n_units, each = max_horizon),

              prediction_horizon = rep(1:max_horizon, n_units),

              pred_mean = mu, pred_var = mu + mu^2/size,

              lb50 = pred_lb50, ub50 = pred_ub50,

              lb95 = pred_lb95, ub95 = pred_ub95,

              obs = obs,

              unit_wise_log_score = unit_wise_log_score,

              multiv_log_score = multiv_log_score))

}


##main

tps <- (133 - 8):(156 - 1)


max_horizon <- 8

n_units <- ncol(data_sts@observed)


library(surveillance)

library(hhh4addon)

library(MASS)
```

```
# evaluation of log scores for order larger 1 involves simulation,

# therefore set.seed

seed <- 0


# get data:

data("data_sts")


ts <- data_sts@observed

unit <- 1

t_cond <- 141

max_horizon <- 8

freq <- 12


logS_data_naive <- matrix(ncol = max_horizon, nrow = length(tps),

                          dimnames = list(paste0("t=", tps),

                                          paste0("h",
1:max_horizon)))

results_detailed_data_naive <- data.frame(data_set = rep("data",
n_units*max_horizon*length(tps)),

                                          unit = 1,

                                          prediction_horizon   =
NA_integer_,

                                          model = "naive",

                                          prediction_time      =
NA_integer_,

                                          pred_mean     =    NA,
pred_var = NA,

                                          lb50 = NA, ub50 = NA,

                                          lb95 = NA, ub95 = NA,
                                               obs = NA,
                                               unit_wise_log_score  =
NA,
                                               multiv_log_score = NA)



for(ind in tps){
    ind_rows0                                                        <-
min(which(is.na(results_detailed_data_naive$prediction_horizon)
))
    inds_rows <- seq(from = ind_rows0, length.out = max_horizon)

    pred_naive_data_temp <- naive_forecast_glmnb(ts = ts, t_cond
= ind,
                                                       max_horizon   =
max_horizon, freq = 12)
    results_detailed_data_naive[inds_rows,
names(pred_naive_data_temp)] <- pred_naive_data_temp
    # store log scores separately:
    logS_data_naive[paste0("t=",        ind),        ]        <-
pred_naive_data_temp$unit_wise_log_score

    print(ind)
}

# add NAs at top for irrelevant forecasts:
for(i in 1:(max_horizon - 1)){
    logS_data_naive[i, (1:(max_horizon - i))] <- NA
}
```

```
tail(logS_data_naive)

#log mean scores

colMeans(logS_data_naive, na.rm = TRUE)
```

results

```


# write out results:

write.csv(results_detailed_data_naive,

          file = "D:/forecasts_data_naive_glmnb.csv")

write.csv(logS_data_naive,

          file = "D:/logS_data_naive_glmnb.csv")
```

results 2

```
# Calculate evaluation metrics

mae      <-      mean(abs(results_detailed_data_naive$obs      -
results_detailed_data_naive$pred_mean), na.rm = TRUE)

rmse     <-     sqrt(mean((results_detailed_data_naive$obs     -
results_detailed_data_naive$pred_mean)^2, na.rm = TRUE))

residuals       <-       results_detailed_data_naive$obs       -
results_detailed_data_naive$pred_mean

sum_squares_residuals <- sum(residuals^2, na.rm = TRUE)

total_sum_squares   <-   sum((results_detailed_data_naive$obs   -
mean(results_detailed_data_naive$obs))^2, na.rm = TRUE)

r_squared <- 1 - (sum_squares_residuals / total_sum_squares)

mape     <-     mean(abs((results_detailed_data_naive$obs     -
results_detailed_data_naive$pred_mean)                        /
results_detailed_data_naive$obs), na.rm = TRUE) * 100

mad      <-      mean(abs(results_detailed_data_naive$obs      -
results_detailed_data_naive$pred_mean), na.rm = TRUE)

pred_var  <-  mean(results_detailed_data_naive$pred_var,  na.rm  =
TRUE)
```

```
lb50 <- mean(results_detailed_data_naive$lb50, na.rm = TRUE)

ub50 <- mean(results_detailed_data_naive$ub50, na.rm = TRUE)


# Create a dataframe with evaluation metrics

evaluation_df <- data.frame(MAE = mae, RMSE = rmse, R_Squared =
r_squared,

                            MAPE = mape, MAD = mad, Pred_Var =
pred_var,

                            LB50 = lb50, UB50 = ub50)


# Print the evaluation dataframe

evaluation_df
```

```
# Calculate MAE

mae      <-      mean(abs(results_detailed_data_naive$obs      -
results_detailed_data_naive$pred_mean), na.rm = TRUE)


# Calculate mean of observed values

mean_observed <- mean(results_detailed_data_naive$obs, na.rm =
TRUE)


# Calculate prediction accuracy

accuracy <- 1 - (mae / mean_observed)


# Print the prediction accuracy

print(accuracy)
```

3.6 ARIMA-SARIMA

```
# Load necessary packages

library(forecast)

library(ggplot2)

data <- data.frame(

    Time = seq(as.Date("2008/01/01"), as.Date("2020/12/01"), by
= "month"),

    NumValue = c(5, 2, 4, 2, 6, 3, 4, 14, 2, 2, 10, 2, 6, 4, 6,
3, 4, 8, 5, 10, 4, 1, 9, 4, 10, 11, 16, 27, 28, 46, 111, 242, 77,
21, 3, 4, 3, 3, 5, 3, 7, 3, 4, 17, 2, 1, 3, 4, 6, 3, 7, 8, 4, 12,
26, 33, 16, 16, 12, 9, 18, 13, 16, 21, 40, 40, 46, 97, 69, 44,
39, 13, 19, 16, 18, 12, 42, 41, 42, 107, 39, 13, 9, 9, 2, 9, 16,
18, 30, 16, 41, 71, 31, 24, 16, 11, 15, 13, 17, 38, 55, 31, 50,
85, 30, 18, 16, 5, 1, 7, 14, 23, 36, 29, 36, 59, 18, 15, 24, 4,
7, 6, 11, 29, 40, 40, 40, 58, 29, 33, 23, 15, 16, 15, 40, 70,
117, 79, 96, 158, 77, 81, 105, 50, 79, 92, 85, 34, 29, 40, 70,
342, 233, 224, 115, 19)

)

# Convert the data to a time series object

ts_data <- ts(data$NumValue, frequency = 12, start = c(2008, 1))


# Define maximum order

max_order <- 5


# Initialize variables to keep track of the best model

best_aic <- Inf

best_order <- c(0,0,0)


# Grid search for ARIMA parameters

for(p in 0:max_order) {

  for(d in 0:max_order) {

    for(q in 0:max_order) {

      if(p+d+q <= max_order) {

        arima_fit <- arima(ts_data, order=c(p,d,q), method="ML",
include.mean=FALSE)

        current_aic <- AIC(arima_fit)

        if(current_aic < best_aic) {

          best_aic <- current_aic

          best_order <- c(p,d,q)

        }

      }

    }
  }

}

cat("Best ARIMA order: ", best_order, "\n")
```

```
cat("Best AIC for ARIMA: ", best_aic, "\n")
```

```
# Fit ARIMA model with best order

arima_fit  <-  arima(ts_data,  order=best_order,  method="ML",
include.mean=FALSE)

#Generate  in-sample  predictions  using  the  fitted  model
arima_pred_insample <- fitted(arima_fit)
```

```
# Predict for 2008 - 2020 (in-sample prediction)

library(ggplot2)

library(ggthemes)

library(dplyr)

library(scales)
library(zoo)

#  Assuming  'ts_data'  and  'arima_pred_insample'  are  correctly
generated from your code

# Creating a data frame with actual and predicted data

data <- tibble(

  Time = as.yearmon(time(ts_data), "%m/%Y"),  # You need to have
the zoo package for as.yearmon

  Actual = as.numeric(ts_data),

  Predicted = as.numeric(arima_pred_insample)

)
```

```
# Create the plot

ggplot(data = data, aes(x = Time)) +

  geom_line(aes(y = Actual), color = 'blue', alpha = 0.7, size =
1.1) +

  geom_line(aes(y  =  Predicted),  color  =  'red',  linetype  =
"dashed", size = 1.1) +

  labs(

    title = "ARIMA: Actual vs Predicted NumValue",

    subtitle = "Best ARIMA order: (0, 1, 4) with AIC: 1538.351",

    x = "Time",

    y = "NumValue",

    color = "Series"

  ) +

  scale_x_yearmon(format   =   "%m/%Y",   n   =   10,   labels   =
date_format("%m/%Y")) +

  theme_minimal() + # apply minimal theme

  theme(

    plot.title = element_text(face = "bold"),

    text = element_text(size = 12),

    legend.position = "bottom",

    panel.grid.major = element_line(colour = "gray", linetype =
"dashed"),

    panel.grid.minor = element_blank()

  ) +

  scale_color_manual(values = c("Actual" = "blue", "Predicted" =
"red")) +

  guides(color = guide_legend(title = "Series"))
```

```r
# Load required libraries

library(Metrics)

library(forecast)


# Generate in-sample predictions using the fitted model


# Assuming you have actual and predicted data in 'ts_data' and
'arima_pred_insample'

actual <- ts_data

predicted <- arima_pred_insample


# Calculate RMSE

rmse <- sqrt(mean((actual - predicted)^2))

cat("RMSE: ", rmse, "\n")


# Calculate MAE

mae <- mean(abs(actual - predicted))

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


# Calculate MAPE

mape <- mean(abs((actual - predicted) / actual)) * 100

cat("MAPE: ", mape, "%\n")
```

```r
# Predict for 2021 - 2024 (out-of-sample forecast)

arima_forecast  <-  forecast::forecast(arima_fit,  h=4*12)    #
forecast for next 4 years

autoplot(arima_forecast, xlab="Time", ylab="NumValue", colour =
'blue', size = 1) +
  ggtitle("ARIMA: Forecasted NumValue for 2021 - 2024") +

  theme_minimal() +

  theme(plot.title = element_text(hjust = 0.5, face="bold", size
= 16),

        axis.title = element_text(face="bold", size = 14),

        plot.background = element_rect(fill = "white", color =
"black")) +

  guides(colour = guide_legend(title = "Series"))
```

```
# Now for SARIMA

# Grid search for SARIMA parameters might be too time-consuming,
so we'll use auto.arima instead

sarima_fit <- auto.arima(ts_data, seasonal = TRUE)

summary(sarima_fit)
```

```
# Check AIC and log likelihood

cat("AIC for SARIMA: ", AIC(sarima_fit), "\n")
```

```
cat("Log Likelihood for SARIMA: ", logLik(sarima_fit), "\n")
```

```


# Predict for 2008 - 2020 (in-sample prediction)

sarima_pred_insample <- fitted(sarima_fit)



# Plot actual vs predicted data
```

```
autoplot(ts_data)       +       autolayer(sarima_pred_insample,
series="SARIMA") + xlab("Time") + ylab("NumValue") +

  ggtitle("SARIMA:    Actual    vs    Predicted    NumValue")    +
guides(colour=guide_legend(title="Series"))
```

```
# Predict for 2021 - 2024 (out-of-sample forecast)

library(ggplot2)

library(forecast)

sarima_forecast <- forecast(sarima_fit, h = 4*12)  # forecast for
next 4 years


# Create a visually appealing plot

autoplot(sarima_forecast,   series="Forecast",   fill="#95d5b2",
colour="#1a1a2e") +

  xlab("Time") +

  ylab("NumValue") +

  ggtitle("SARIMA: Forecasted NumValue for 2021 - 2024") +

  theme_minimal() +

  theme(

    plot.title = element_text(hjust = 0.5, face="bold", size=14),

    axis.title.x = element_text(face="bold", size=12),

    axis.title.y = element_text(face="bold", size=12),

    legend.title = element_text(face="bold", size=12)

  ) +

  guides(fill=guide_legend(title="Prediction         Interval"),
color=guide_legend(title="Series")))
```

```
cat("SARIMA Mean Log Score: ", sarima_mean_log_score, "\n")
```

```
arima_fit  <-  arima(ts_data,  order=best_order,  method="ML",
include.mean=FALSE)

cat("ARIMA Mean Log Score: ", arima_mean_log_score, "\n")
```

```
# Assuming you have actual and predicted data in 'ts_data' and
'sarima_pred_insample'

actual <- ts_data

predicted_sarima <- sarima_pred_insample


# Convert to numeric

actual <- as.numeric(actual)

predicted_sarima <- as.numeric(predicted_sarima)


# Calculate RMSE

rmse_sarima <- sqrt(mean((actual - predicted_sarima)^2))

cat("RMSE for SARIMA: ", rmse_sarima, "\n")
```

```
# Calculate MAE

mae_sarima <- mean(abs(actual - predicted_sarima))

cat("MAE for SARIMA: ", mae_sarima, "\n")
```

```
# Calculate MAPE

mape_sarima <- mean(abs((actual - predicted_sarima) / actual)) *
100

cat("MAPE for SARIMA: ", mape_sarima, "%\n")
```

4 MODEL COMPARISON

```
# Load required packages

# Load required packages

library(dplyr)

library(ggplot2)

library(DT)



# Define the data

data <- data.frame(

    Model = c('HHH4', 'KCDE', 'AR1', 'Pois', 'Siraj', 'Naïve'),
    Mean_Log_Score   =   c(4.9200325220223,   5.448,   4.946424,
4.946437, 5.480640, 7.883698),

    RMSE = c(17.4126455037778, 82.42, 83.224, 82.32461, 102.7528,
89.22906),

    MAE = c(10.2698091163071, 52.44, 55.714, 55.76565, 78.2803,
62.25116),

    MAPE  =  c(0.650968732264653,  47.95,  0.8936543,  0.8947496,
1.647, 62.12227)

)


# Create a ranking for each column

ranking <- data %>%

    mutate(Rank_MLS = rank(Mean_Log_Score),

           Rank_RMSE = rank(RMSE),

           Rank_MAE = rank(MAE),

           Rank_MAPE = rank(MAPE))


# Display the table

datatable(ranking, options = list(pageLength = 10), rownames =
FALSE)


# Plot the graphs

p1 <- ggplot(data, aes(x = Model, y = Mean_Log_Score)) +

    geom_bar(stat = "identity", fill = "steelblue") +

    theme_minimal() +

    ggtitle("Mean Log Score by Model") +

    theme(axis.text.x = element_text(angle = 45, hjust = 1))


p2 <- ggplot(data, aes(x = Model, y = RMSE)) +
    geom_bar(stat = "identity", fill = "steelblue") +

    theme_minimal() +

    ggtitle("RMSE by Model") +

    theme(axis.text.x = element_text(angle = 45, hjust = 1))


p3 <- ggplot(data, aes(x = Model, y = MAE)) +

    geom_bar(stat = "identity", fill = "steelblue") +

    theme_minimal() +

    ggtitle("MAE by Model") +

    theme(axis.text.x = element_text(angle = 45, hjust = 1))


p4 <- ggplot(data, aes(x = Model, y = MAPE)) +

    geom_bar(stat = "identity", fill = "steelblue") +

    theme_minimal() +

    ggtitle("MAPE by Model") +

    theme(axis.text.x = element_text(angle = 45, hjust = 1))


print(p1)

print(p2)

print(p3)

print(p4)
```

AIC AND DSS

```
# Define the AIC values for each model

aic_values <- c(

  hhh4 = 910,

  ar1 = 1206,

  pois = 1206,
  lin = 1206,

  geom = 1206,

  unres = 1206,

  siraj = 1206

)


# Define the DSS values for HHH4 and KCDE models

dss_values <- c(

  hhh4 = 7.118514,

  kcde = 10.422

)


# Calculate the best model based on AIC

best_model_aic <- names(aic_values)[which.min(aic_values)]


# Calculate the best model based on DSS

best_model_dss <- names(dss_values)[which.max(dss_values)]


# Create an evaluation summary

evaluation_summary <- paste(

  "Based on AIC, the best model is", best_model_aic,

  "with an AIC value of", aic_values[best_model_aic],

  "\nBased on DSS, the best model is", best_model_dss,

  "with a DSS value of", dss_values[best_model_dss]

)


# Print the evaluation summary

cat(evaluation_summary)
```

```
Based on AIC, the best model is hhh4 with an AIC value of 910

Based on DSS, the best model is kcde with a DSS value of 10.422
```

```
# Load required packages

library(dplyr)

library(ggplot2)

library(DT)


# Define the data

data <- data.frame(

    Model = c('HHH4', 'KCDE', 'AR1', 'Pois', 'Siraj', 'Naïve'),

    Mean_Log_Score   =   c(4.9200325220223,   5.448,   4.946424,
4.946437, 5.480640, 7.883698),

    RMSE = c(17.4126455037778, 82.42, 83.224, 82.32461, 102.7528,
89.22906),

    MAE = c(10.2698091163071, 52.44, 55.714, 55.76565, 78.2803,
62.25116),

    MAPE  =  c(0.650968732264653,  47.95,  0.8936543,  0.8947496,
1.647, 62.12227)

)


# Round numeric columns to two decimal places if needed

numeric_cols <- c("Mean_Log_Score", "RMSE", "MAE", "MAPE")

data[numeric_cols]  <-  lapply(data[numeric_cols],  function(x)
ifelse(x %% 1 != 0, round(x, 2), x))


# Create a ranking for each column

ranking <- data %>%
    mutate(Rank_MLS = rank(Mean_Log_Score),
           Rank_RMSE = rank(RMSE),
           Rank_MAE = rank(MAE),
           Rank_MAPE = rank(MAPE))

# Display the table with "bootstrap" style
datatable(ranking, options = list(pageLength = 10, style =
'bootstrap'), rownames = FALSE)

# Plot the graphs
p1 <- ggplot(data, aes(x = Model, y = Mean_Log_Score)) +
    geom_bar(stat = "identity", fill = "steelblue") +
    theme_minimal() +
    ggtitle("Mean Log Score by Model") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))

p2 <- ggplot(data, aes(x = Model, y = RMSE)) +
    geom_bar(stat = "identity", fill = "steelblue") +
    theme_minimal() +
    ggtitle("RMSE by Model") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))

p3 <- ggplot(data, aes(x = Model, y = MAE)) +
    geom_bar(stat = "identity", fill = "steelblue") +
    theme_minimal() +
    ggtitle("MAE by Model") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
p4 <- ggplot(data, aes(x = Model, y = MAPE)) +

    geom_bar(stat = "identity", fill = "steelblue") +

    theme_minimal() +

    ggtitle("MAPE by Model") +

    theme(axis.text.x = element_text(angle = 45, hjust = 1))


print(p1)

print(p2)

print(p3)

print(p4)
```

THE END ☺