Application to air pollution forecasting

Raphaela Azar Sbonelo Gumede

2025-09-30

Introduction

The World Health Organization estimated that approximately seven million people die every year due to air pollution (Cape Town 2024). To manage levels of air pollution, the City of Cape Town has built eleven air quality monitoring (AQM) stations in Cape Town. These stations collect data on particulate matter 10 (PM_{10}), particulate matter 2.5 ($PM_{2.5}$), sulphur dioxide (SO_2), nitrogen dioxide (NO_2), ozone (O_3), hydrogen sulphide (H_2S), carbon monoxide (CO), benzene (C_6H_6), and lead (Pb) (Cape Town 2024). Air pollution data for Cape Town can be found on the City of Cape Town open data portal (Cape Town 2015).

The Table View station was chosen for this project. This is because it had the fewest missing observations. The data consist of our response variable, nitrogen dioxide ($\mu g/m^3$), and explanatory variables, sulphur dioxide ($\mu g/m^3$), particulate matter 10 ($\mu g/m^3$), and wind speed (m/s) from 01/01/2019 to 31/12/2019, measured hourly at the Table View station.

Exploratory data analysis

Table 1: Summary statistics of the air quality dataset from 01/01/2019 to 31/12/2019 measured hourly.

	Min.	1st Qu.	Median	Mean	SD	3rd Qu.	Max.	NA
NO2	0.0	5.0	9.0	12.729	10.724	17.0	113.0	734
PM10	0.0	12.0	17.0	19.824	12.302	24.0	158.0	298
SO2	0.0	2.0	3.0	6.202	11.299	5.0	142.0	638
Speed	0.5	2.3	3.6	3.735	1.710	4.9	11.2	918

All variables in this dataset are non-negative, and many observations are missing. Some of the time series models we work with cannot handle missing data directly, so we perform imputation. Initially, we tried natural cubic splines, but some imputed values were negative because splines are unconstrained, which is problematic for non-negative variables. We then applied linear interpolation, which preserved the statistical properties of the data reasonably well. However, linear interpolation does not enforce smoothness across the time series. To address this, we used Kalman filter-based interpolation via the `na_kalman()` function in the `imputeTS` package in `R`. This method produces smooth, non-negative estimates and is statistically optimal for estimating missing values in time series (Corey Montella). It effectively leverages the temporal structure of the data, resulting in imputed values that respect both the data's continuity and its statistical properties. This uses a local level model, which is the simplest structural time series (state space) model.

Table 2: Summary statistics of the imputed air quality dataset from 01/01/2019 to 31/12/2019.

	Min.	1st Qu.	Median	Mean	SD	3rd Qu.	Max.	NA
NO2	0.0	5.0	9.0	12.806	10.773	17.0	113.0	0
PM10	0.0	12.0	17.0	19.679	12.200	24.0	158.0	0
SO2	0.0	2.0	3.0	6.331	11.097	5.0	142.0	0
Speed	0.5	2.4	3.4	3.703	1.664	4.8	11.2	0

Notice that the summary statistics did not change much after applying the Kalman filter interpolation.

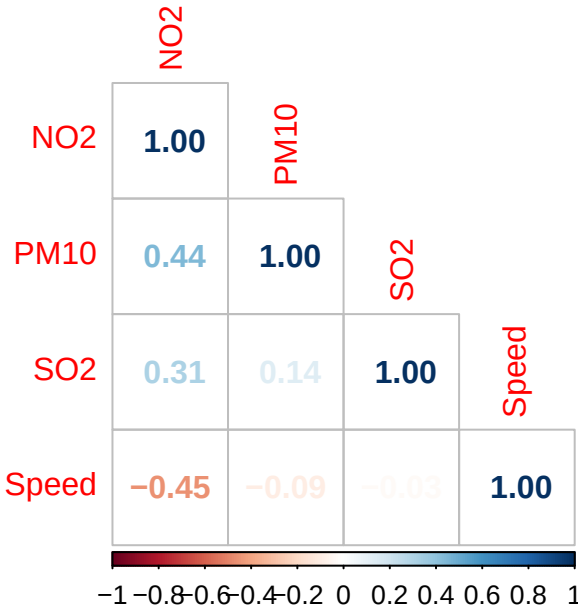

Figure 1: Correlation plot of the air quality dataset from 01/01/2019 to 31/12/2019 measured hourly.

Our response variable NO_2 appears to be moderately positively correlated with PM_{10} and SO_2, and moderately negatively correlated with Speed. These are not ideal explanatory variables since we typically would like them to be strongly correlated with the response variable. The explanatory variables are weakly correlated with one another, whether it be a positive or a negative correlation. This is ideal since some models do not work well with correlated explanatory variables, often leading to unstable point estimates and inflated standard errors.

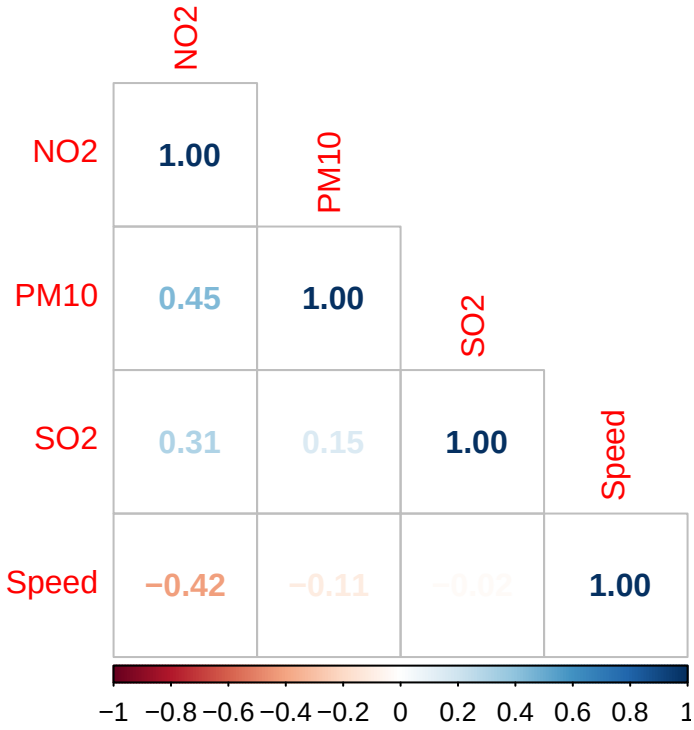

Figure 2: Correlation plot of the imputed air quality dataset from 01/01/2019 to 31/12/2019 measured hourly.

Notice that the correlations between variables did not change much after applying the Kalman filter interpolation.

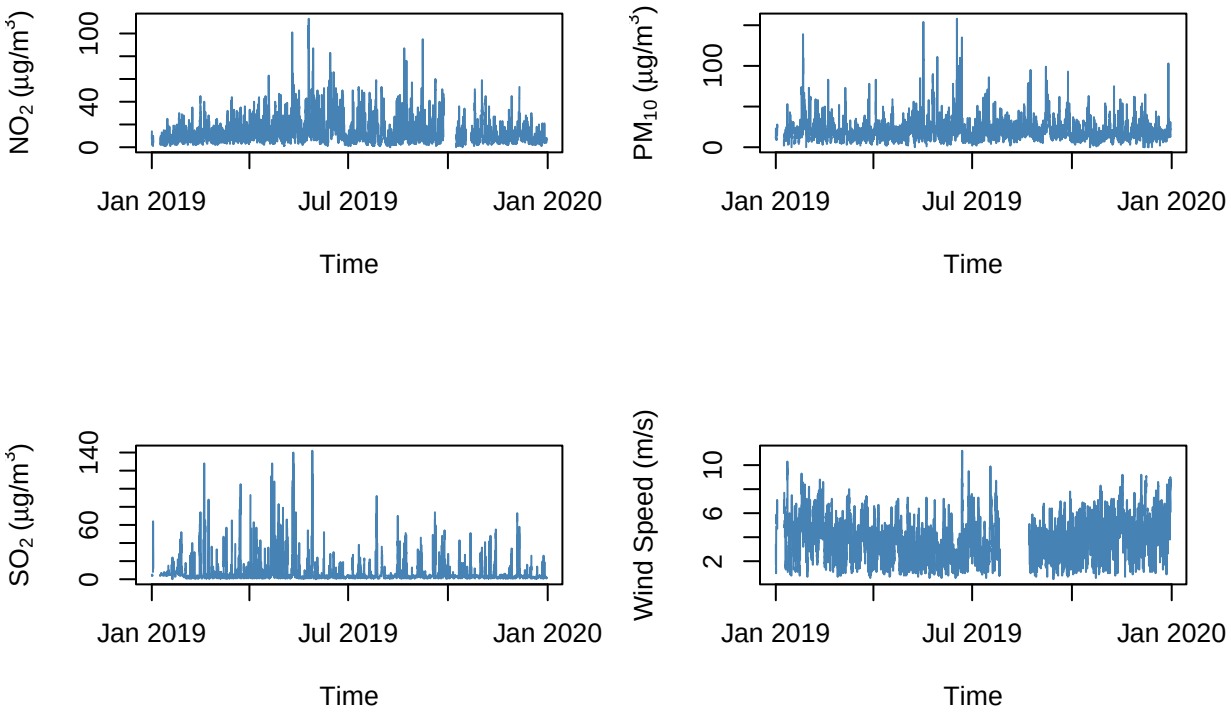

Figure 3: Scatter plots of the air quality dataset from 01/01/2019 to 31/12/2019 measured hourly.

There seems to be a weak seasonal and trend component based on the time series plots. There is no clear indication of a cyclical component. There is random variation as usual. It is challenging to visually identify the time series components since this is a noisy dataset.

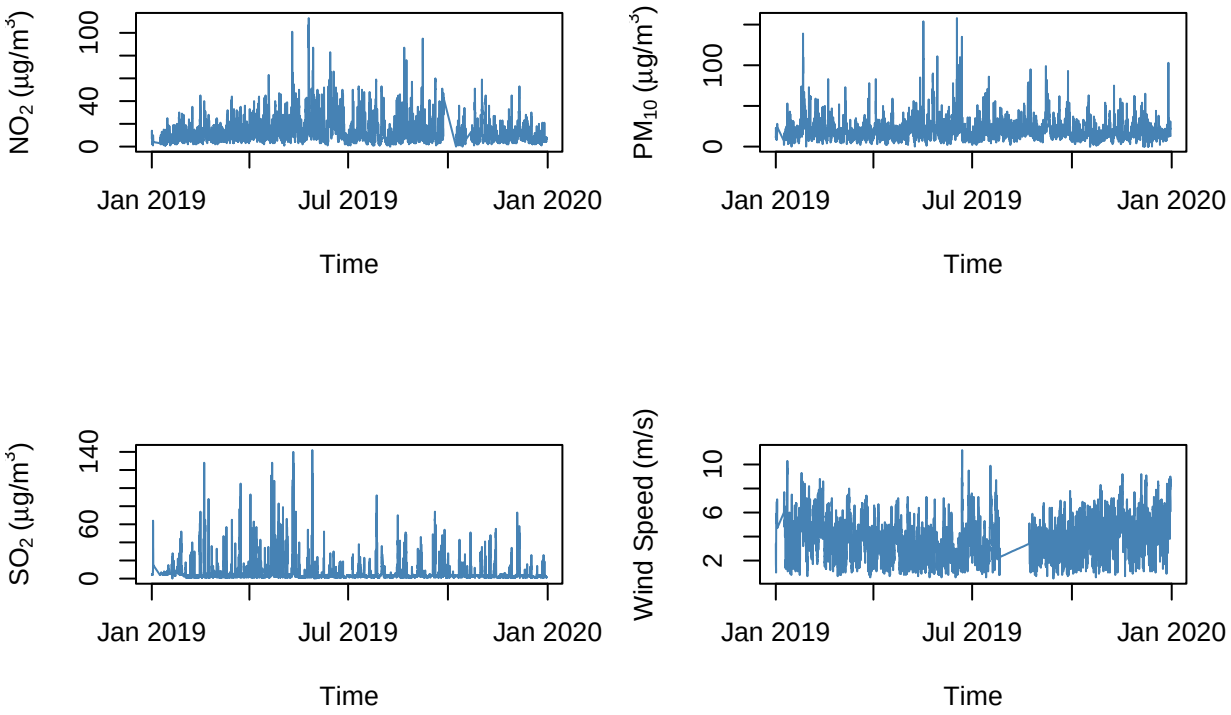

Figure 4: Scatter plots of the imputed air quality dataset from 01/01/2019 to 31/12/2019 measured hourly.

Notice that the time series plots did not change much after applying the Kalman filter interpolation.

Model formulation

Let the window from 08/02/2019 at 00:00 to 15/02/2019 at 23:00 denote our training set (response variable $\mathbf{y}_1$ and design matrix $\mathbf{X}_1$). And let the window from 16/02/2019 at 00:00 to 16/02/2019 at 23:00 denote our test set (response variable $\mathbf{y}_2$ and design matrix $\mathbf{X}_2$).

Simple forecasting models

`Mean`: The prediction is the average value. The Mean method

$$\hat{y}_{2_{n_1+h}} = \sum_{i=1}^{n_1} \frac{y_{1_i}}{n_1}.$$

`Naive`: The prediction is the last observed value. The Naive method

$$\hat{y}_{2_{n_1+h}} = y_{1_{n_1}}.$$

`Seasonal naive`: The prediction is the last observed value of the corresponding previous season. The Seasonal Naive method

$$\hat{y}_{2_{n_1+h}} = y_{1_{n_1+h-m(k+1)}},$$

where m is the seasonal period and $k = \lfloor \frac{h-1}{m} \rfloor$.

`Drift`: The prediction is the last observed value adjusted for the average trend. The Drift method

$$\hat{y}_{2_{n_1+h}} = y_{1_{n_1}} + h\left(\frac{y_{1_{n_1}} - y_{1_1}}{n_1 - 1}\right).$$

AR(1)

`AR(1)`: The prediction is based on a constant, plus a fraction of the previous value. The AR(1) model

$$\hat{y}_{2_{n_1+h}} = c + \phi y_{1_{n_1}}.$$

Gaussian process

Linear mean function:

$$m(x_i) = \sum_{j=1}^{p} x_{ij}\beta_j, \quad \text{where} \quad i \in \{1, \ldots, \text{n}\}.$$

The resulting mean vectors of the training set and test set respectively are as follows.

$$\mathbf{m}_1 = \mathbf{X}_1\boldsymbol{\beta}, \quad \mathbf{m}_2 = \mathbf{X}_2\boldsymbol{\beta}$$

Squared-exponential kernel:

$$k(x_i, x_j) = \alpha^2 \exp\left(-\frac{(x_i - x_j)^2}{2\rho^2}\right) + \delta \text{I}_{i=j}, \text{ where } \alpha, \rho \text{ are hyperparameters and } \delta = 1\text{e} - 9.$$

The resulting covariance matrices are as follows

$$\mathbf{K}_{11} = k(\mathbf{X}_1, \mathbf{X}_1) + \delta\mathbf{I}_{n_1}$$

$$\mathbf{K}_{12} = k(\mathbf{X}_1, \mathbf{X}_2)$$

$$\mathbf{K}_{22} = k(\mathbf{X}_2, \mathbf{X}_2) + \delta\mathbf{I}_{n_2}$$

The likelihood is the following.

$$\mathbf{y}_1 | \, \boldsymbol{\beta}, \alpha, \rho \sim \mathcal{N}(\mathbf{m}_1, \mathbf{K}_{11})$$

Choosing the following priors.

$$\boldsymbol{\beta} \sim \mathcal{N}(0, \mathbf{I}_p)$$

$$\alpha \sim \text{half-normal}(0, 1)$$

$$\rho \sim \text{half-normal}(24, 12)$$

The joint distribution of $[\mathbf{y}_1, \mathbf{y}_2]$ given fixed parameters $(\boldsymbol{\beta}, \alpha, \rho)$, the joint distribution of the training and set outputs is a multivariate normal:

$$\begin{bmatrix}\mathbf{y}_1 \\ \mathbf{y}_2\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{m}_1 \\ \mathbf{m}_2\end{bmatrix}, \begin{bmatrix}\mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{21} & \mathbf{K}_{22}\end{bmatrix}\right).$$

The conditional distribution of $\mathbf{y}_2 | \, \mathbf{y}_1, \boldsymbol{\beta}, \alpha, \rho$. This is just a conditional distribution of a multivariate normal distribution which is a well known result.

$$\mathbf{y}_2 | \, \mathbf{y}_1, \boldsymbol{\beta}, \alpha, \rho \sim \mathcal{N}(\mathbf{m}_2 + \mathbf{K}_{12}^{\text{T}}\mathbf{K}_{11}^{-1}(\mathbf{y}_1 - \mathbf{m}_1), \mathbf{K}_{22} - \mathbf{K}_{12}^{\text{T}}\mathbf{K}_{11}^{-1}\mathbf{K}_{12})$$

$$\pi(\mathbf{y}_2 | \, \mathbf{y}_1) = \int \pi(\mathbf{y}_2 | \, \mathbf{y}_1, \boldsymbol{\beta}, \alpha, \rho)\, \pi(\boldsymbol{\beta})\, \pi(\alpha)\, \pi(\rho)\, d\boldsymbol{\beta}\, d\alpha\, d\rho.$$

This does not have a closed form solution. We approximated the posterior samples of this distribution using `Stan`. `Stan` avoids computing explicit inverses. These quantities are calculated in `Stan` as follows.

Cholesky decomposition of $\mathbf{K}_{11}$.

$$\mathbf{K}_{11} = \mathbf{L}\mathbf{L}^{\mathrm{T}}.$$

Compute $\mathbf{w} = \mathbf{K}_{11}^{-1}(\mathbf{y}_1 - \mathbf{m}_1)$ via triangular solves:

$$\mathbf{v} = \mathbf{L}^{-1}(\mathbf{y}_1 - \mathbf{m}_1).$$

Then

$$\mathbf{w} = (\mathbf{L}^{\mathrm{T}})^{-1}\mathbf{v}.$$

Thus

$$\mathbf{w} = \mathbf{K}_{11}^{-1}(\mathbf{y}_1 - \mathbf{m}_1).$$

The predictive mean is calculated as follows.

$$\mathbf{m}_2 + \mathbf{K}_{12}^{\mathrm{T}}\mathbf{w} = \mathbf{m}_2 + \mathbf{K}_{12}^{\mathrm{T}}\mathbf{K}_{11}^{-1}(\mathbf{y}_1 - \mathbf{m}_1).$$

And the predictive covariance is calculated as follows.

$$\mathbf{A} = \mathbf{L}^{-1}\mathbf{K}_{12}.$$

Then

$$\mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{K}_{12}^{\mathrm{T}}\mathbf{L}^{-\mathrm{T}}\mathbf{L}^{-1}\mathbf{K}_{12}.$$

Thus

$$\mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{K}_{12}^{\mathrm{T}}\mathbf{K}_{11}^{-1}\mathbf{K}_{12}.$$

Then draw samples from $\mathbf{y}_2|\,\mathbf{y}_1$ via Hamiltonian Monte Carlo (HMC) in `Stan`. The algorithm proceeds as follows.

1. `Stan` samples the parameters ($\boldsymbol{\beta}$, α, ρ) from the posterior distribution $\pi(\boldsymbol{\beta}, \alpha, \rho|\,\mathbf{y}_1)$ using HMC.
2. For each parameter draw `Stan` then executes the `generated quantities` block which:

- Computes the predictive mean and predictive covariance,
- draws

  $$\mathbf{y}_2^{(s)} \sim \mathcal{N}(\text{pred mean, pred covariance})$$

  using `multi_normal_rng`. The RNG draw is not part of the HMC dynamics - it happens after the sampler proposes/accepts a parameter sample.

3. Collecting the $\mathbf{y}_2^{(s)}$ across all saved iterations gives a Monte Carlo approximation to the marginal posterior predictive

   $$\pi(\mathbf{y}_2|\,y_1) \approx \frac{1}{S}\sum_{s=1}^{S}\mathcal{N}(\mathbf{y}_2|\,\mu^{(s)}, \Sigma^{(s)}),$$

   where $\mu^{(s)}$, $\Sigma^{(s)}$ are conditional mean/covariance computed at the s-th parameter draw.

Model comparison statistics

Table 3: Average performance of the fitted models on the out-of-sample period 16/02/2019 00:00 to 16/02/2019 23:00.

	RMSE	MAE	MAPE
Mean	7.189	5.811	39.466
Naive	8.426	6.667	39.771
Seasonal naive	7.638	6.500	43.334
Drift	10.038	8.267	49.415
AR(1)	6.945	5.457	35.752
Gaussian process	3.980	3.282	29.627

Model comparison plots

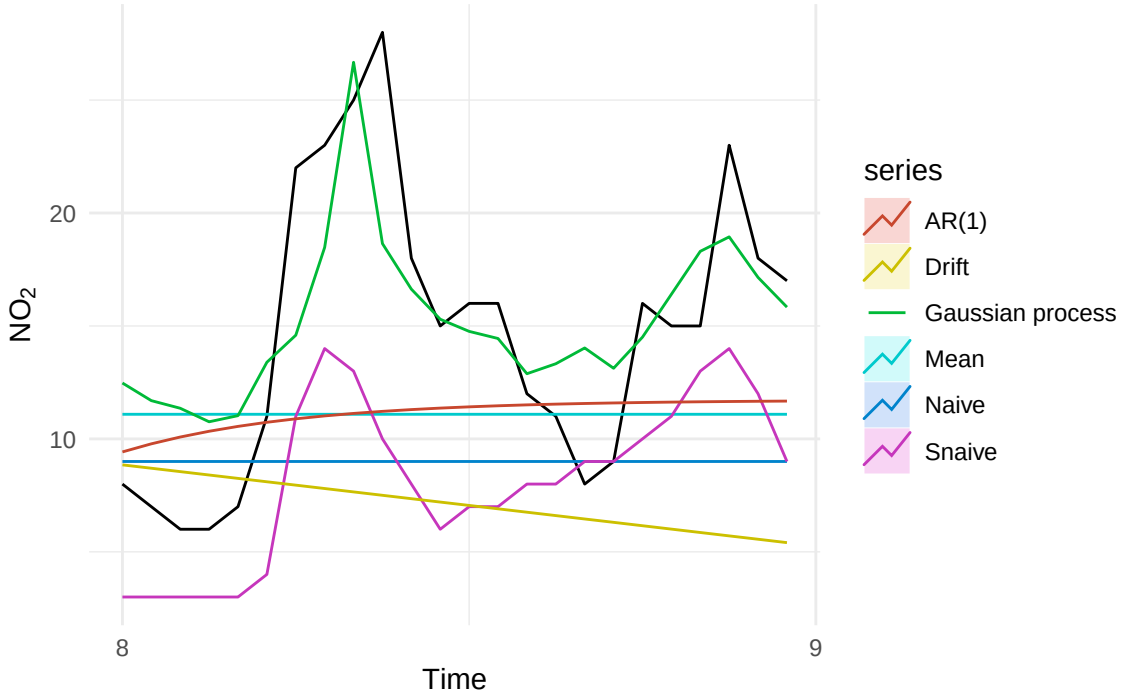

Figure 5: Time series of the fitted models on the out-of-sample period 06/02/2019 00:00 to 06/02/2019 23:00.

Cross validation

Cross-validation was conducted as follows: a period of 7 days from 08/02/2019 at 00:00 to 15/02/2019 at 23:00 was used for the training set, and one hour on 16/02/2019 at 00:00 was used as a test set. On each iteration, we increment the training set and shift the test set by one time point, after which loss functions (MSE, MAE, and MAPE) are calculated and stored in a matrix. Once all ten iterations are completed, the error functions are averaged over all iterations.

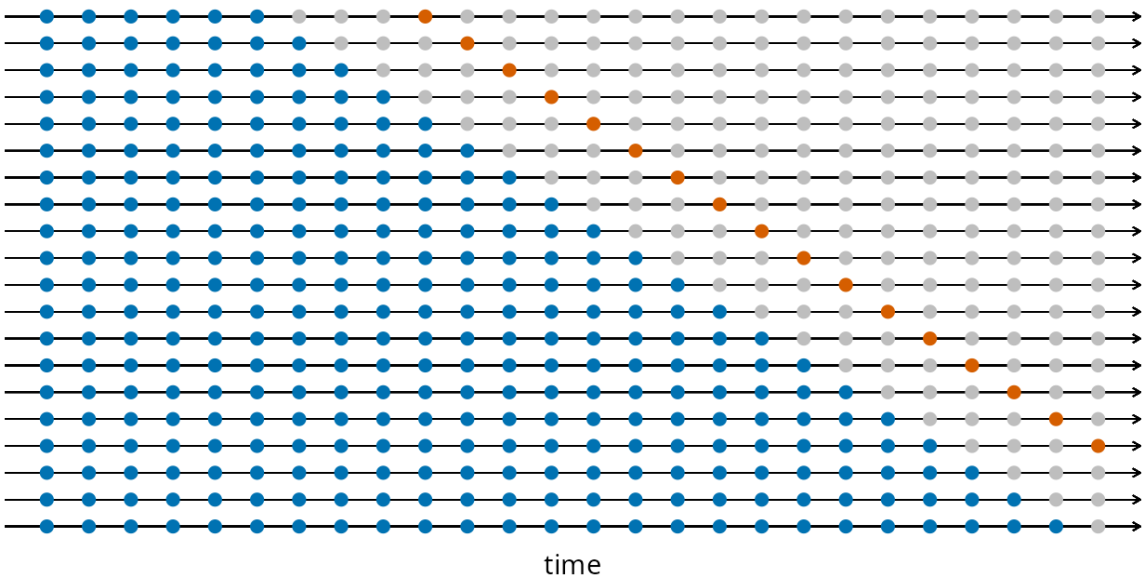

Figure 6: Illustration of the rolling-origin cross-validation scheme, showing the expanding training set and one-step-ahead test set.

Appendix

```r
# Attach the required packages.
require(package = "bayesplot")
require(package = "cmdstanr")
require(package = "corrplot")
require(package = "forecast")
require(package = "ggplot2")
require(package = "kableExtra")
require(package = "knitr")
require(package = "rstan")
require(package = "parallel")
require(package = "posterior")
color_scheme_set("brightblue")


# Load the datasets.
load(file = "../objects/extracted_data_storage.RData")
load(file = "../objects/imputed_data_storage.RData")

# Extract the datasets.
X1 <- extracted_data[[1]]
X2 <- imputed_data[[1]]

# Declare variable names.
var_name <- c("DateTime", "NO2", "PM10", "SO2", "Speed")

# Declare variable names in a proper format.
x_name <- c("DateTime",
```

```
            expression(NO[2] ~ "(" * mu * "g/m"^3 * ")"),
            expression(PM[10] ~ "(" * mu * "g/m"^3 * ")"),
            expression(SO[2] ~ "(" * mu * "g/m"^3 * ")"),
            expression("Wind Speed" ~ "(m/s)"))

# Calculate summary statistics.
summary_statistics <- function(X){
   rbind("Min." = sapply(X = X, FUN = min, na.rm = TRUE),
         "1st Qu." = sapply(X = X, FUN = quantile, 0.25, na.rm = TRUE),
         "Median" = sapply(X = X, FUN = median, na.rm = TRUE),
         "Mean" = sapply(X = X, FUN = mean, na.rm = TRUE),
         "SD" = sapply(X = X, FUN = sd, na.rm = TRUE),
         "3rd Qu." = sapply(X = X, FUN = quantile, 0.75, na.rm = TRUE),
         "Max." = sapply(X = X, FUN = max, na.rm = TRUE),
         "NA" = sapply(X = X, FUN = function(column) sum(is.na(column))))
}
S_mat1 <- summary_statistics(X = X1)
S_mat2 <- summary_statistics(X = X2)

# Display summary statistics.
kable(x = round(x = t(x = S_mat1[, 2:5]), digits = 3),
        caption = "Summary statistics of the air quality dataset from 01/01/2019 to 31/12/2019 measured

kable(x = round(x = t(x = S_mat2[, 2:5]), digits = 3),
        caption = "Summary statistics of the imputed air quality dataset from 01/01/2019 to 31/12/2019.

# Correlation plot for the original data.
par(mfrow = c(1, 2))
cor_mat1 <- cor(X1[, 2:5], use = "na.or.complete", method = "pearson")
corrplot(corr = cor_mat1, method = "number", type = "lower")

# Correlation plot for the imputed data.
cor_mat2 <- cor(X2[, 2:5], method = "pearson")
corrplot(corr = cor_mat2, method = "number", type = "lower")

# Function to sketch scatter plots.
draw_plots <- function(X, name){
   for(i in 2:5){
      filename <- paste0("../images/", name, "_", tolower(x = var_name[i]), ".png")
      plot(x = X$DateTime, y = X[[var_name[i]]],
           main = "", xlab = "Time", ylab = x_name[i],
           type = "l", col = "steelblue", cex = 0.5)
   }
}

# Scatter plots for the original data.
par(mfrow = c(2, 2))
draw_plots(X = X1, name = "extracted")

# Scatter plots for the imputed data.
par(mfrow = c(2, 2))
draw_plots(X = X2, name = "imputed")
```

```r
# Load the dataset.
load(file = "../objects/window_data_storage.RData")


R_mat <- matrix(data = NA, nrow = 3, ncol = 6,
                dimnames = list(c("RMSE", "MAE", "MAPE"),
                                c("Mean", "Naive", "Seasonal naive", "Drift", "AR(1)", "Gaussian proces

# Fit the model(s).
mean_fit <- meanf(y = y1, h = h)
naive_fit <- naive(y = y1, h = h)
snaive_fit <- snaive(y = y1, h = h)
drift_fit <- rwf(y = y1, h = h, drift = TRUE)
arima_obj <- arima(x = y1, order = c(1, 0, 0))
arima_fit <- forecast(object = arima_obj, h = h)

# Calculate the mean square error.
R_mat[1, 1] <- sqrt(mean((mean_fit$mean - y2)^2))
R_mat[1, 2] <- sqrt(mean((naive_fit$mean - y2)^2))
R_mat[1, 3] <- sqrt(mean((snaive_fit$mean - y2)^2))
R_mat[1, 4] <- sqrt(mean((drift_fit$mean - y2)^2))
R_mat[1, 5] <- sqrt(mean((arima_fit$mean - y2)^2))

# Calculate the mean absolute error.
R_mat[2, 1] <- mean(abs(mean_fit$mean - y2))
R_mat[2, 2] <- mean(abs(naive_fit$mean - y2))
R_mat[2, 3] <- mean(abs(snaive_fit$mean - y2))
R_mat[2, 4] <- mean(abs(drift_fit$mean - y2))
R_mat[2, 5] <- mean(abs(arima_fit$mean - y2))

# Calculate the mean absolute percentage error.
R_mat[3, 1] <- mean((abs(mean_fit$mean - y2) / abs(y2)) * 100)
R_mat[3, 2] <- mean((abs(naive_fit$mean - y2) / abs(y2)) * 100)
R_mat[3, 3] <- mean((abs(snaive_fit$mean - y2) / abs(y2)) * 100)
R_mat[3, 4] <- mean((abs(drift_fit$mean - y2) / abs(y2)) * 100)
R_mat[3, 5] <- mean((abs(arima_fit$mean - y2) / abs(y2)) * 100)

# Plot the forecasts.
p2 <- p2 + autolayer(object = mean_fit, series = "Mean", PI = FALSE)
p2 <- p2 + autolayer(object = naive_fit, series = "Naive", PI = FALSE)
p2 <- p2 + autolayer(object = snaive_fit, series = "Snaive", PI = FALSE)
p2 <- p2 + autolayer(object = drift_fit, series = "Drift", PI = FALSE)
p2 <- p2 + autolayer(object = arima_fit, series = "AR(1)", PI = FALSE)

cores_number <- detectCores() - 2
stan_data <- list(X1 = X1, y1 = y1, n1 = n1, p = p, X2 = X2, n2 = n2)

# Compile Stan model once.
mod <- cmdstan_model(stan_file = "squared_exponential.stan")

# Sample from a Gaussian process.
fit <- mod$sample(
    data = stan_data,
```

```
   chains = 4,
   parallel_chains = cores_number,
   seed = 2025,
   refresh = 0,
   show_messages = FALSE,
    show_exceptions = FALSE)

# fit is a CmdStanMCMC object.
y2_draws <- fit$draws("y2", format = "matrix")
y2_mean  <- colMeans(y2_draws)

y2_hat <- ts(y2_mean, start = start(y2), frequency = frequency(y2))

# Calculate the error.
R_mat[1, 6] <- sqrt(mean((y2_hat - y2)^2))
R_mat[2, 6] <- mean(abs(y2_hat - y2))
R_mat[3, 6] <- mean((abs(y2_hat - y2) / abs(y2)) * 100)
kable(x = t(round(x = R_mat, digits = 3)),
        caption = "Average performance of the fitted models on the out-of-sample period 16/02/2019 00:0

# Plot the forecasts.
p2 <- p2 + autolayer(object = y2_hat, series = "Gaussian process")
p2
```

```
functions {
  matrix se_kernel(matrix X1, matrix X2, real alpha, real rho) {
    int n1 = rows(X1);
    int n2 = rows(X2);
    matrix[n1, n2] K;

    for (i in 1:n1) {
      for (j in 1:n2) {
        // squared Euclidean distance between rows i and j
        real sqdist = dot_self(X1[i] - X2[j]);
        K[i, j] = square(alpha) * exp(-0.5 * sqdist / square(rho));
      }
    }
    return K;
  }
}

data {
  int<lower=1> n1;
  int<lower=1> p;
  matrix[n1, p] X1;
  vector[n1] y1;

  int<lower=1> n2;
  matrix[n2, p] X2;
}

transformed data {
  real delta = 1e-9;
```

```
}

parameters {
  real<lower=0> alpha;
  real<lower=0> rho;
  vector[p] beta;
}

model {
  // Priors
  alpha ~ normal(0, 1);
  rho ~ normal(24, 12); // half-normal since <lower=0>
  beta ~ normal(0, 1);

  // Build covariance for training
  matrix[n1, n1] K11 = se_kernel(X1, X1, alpha, rho)
                       + diag_matrix(rep_vector(delta, n1));
  vector[n1] mu1 = X1 * beta;

  // Likelihood
  y1 ~ multi_normal(mu1, K11);
}

generated quantities {
  vector[n2] y2;

  // Covariance blocks
  matrix[n1, n1] K11 = se_kernel(X1, X1, alpha, rho)
                       + diag_matrix(rep_vector(delta, n1));
  matrix[n1, n2] K12 = se_kernel(X1, X2, alpha, rho);
  matrix[n2, n2] K22 = se_kernel(X2, X2, alpha, rho)
                       + diag_matrix(rep_vector(delta, n2));

  // Means
  vector[n1] mu1 = X1 * beta;
  vector[n2] mu2 = X2 * beta;

  // Cholesky factor of K11
  matrix[n1, n1] L = cholesky_decompose(K11);

  // Predictive mean
  vector[n1] y1_centered = y1 - mu1;
  vector[n1] v = mdivide_left_tri_low(L, y1_centered);
  vector[n1] w = mdivide_right_tri_low(v', L)';
  vector[n2] pred_mean = mu2 + K12' * w;

  // Predictive covariance
  matrix[n1, n2] A = mdivide_left_tri_low(L, K12);
  matrix[n2, n2] pred_cov = K22 - A' * A;

  // Draw sample
  y2 = multi_normal_rng(pred_mean, pred_cov);
}
```

References

Cape Town, City of. 2015. “Open Data Portal.” https://odp-cctegis.opendata.arcgis.com/.

———. 2024. “Air Quality Management Plan.” https://resource.capetown.gov.za/documentcentre/Documents/City%20strategies,%20plans%20and%20frameworks/Air_Quality_Management_Plan_2024.pdf.