Part A: Altered Data

The creation of the altered demand history involved finding a method that would accurately model the strong multi-seasonality present during this period when trained with data between July 2017 and July 2019. Out of the examined seasonal and multi-seasonal methods, the stlf() function of the forecast package produced the best results. We used this fit to generate two years of weekly forecasts, with some added noise to obscure the exact model specification of the stlf() modeler. The noise was generated by sampling 104 random observations from a normal distribution using the rnorm() function and scaling them to 2.5% of the series mean. Random sampling was seeded with a constant to ensure reproducible results between runs.

Part B: Methods

For all the methods below, we denote observations with $y_i$ and forecasts with $\hat{y}_i$. The subscript $i$ indicates the time step.

**Simple moving average**

Simple moving average (MA) methods generate a forecast by averaging a specified number of observations in the immediate past. The size of this window $q$ is called the order of the moving average. An MA of $q$th order is specified as:

$$\hat{y}_{i+1} = \frac{1}{q} \sum_{j=i-q+1}^{i} y_j.$$ 

The ma() function from the forecast package was used with the order parameter to run simple moving averages of 5, 7, 9, and 12 weeks.

**Seasonal naïve**

The seasonal naive method (SNAIVE) uses the seasonally corresponding past observation as the forecast. For example, the forecast for February this year would be the February observation of the last year using monthly data. A seasonal naïve forecast can be computed with:
\hat{y}_{i+h} = y_{i+h-m(k+1)},

where \( h \) denotes the time step after \( i \) we wish to forecast, \( m \) denotes the length of the period, and \( k \) denotes the number of full seasons in the forecast period. This method works best when the series exhibits a strong seasonality with negligible trend and noise. The seasonal naïve forecasts were produced with the \texttt{snaive(y, h)} function from the \textit{forecast} package.

**Exponential smoothing methods**

Exponential smoothing methods (ETS)\(^1\) generate forecasts from weighted sums of the past observations so that the weights decay exponentially as we move backward in time. In their simplest form, ETS forecasts are produced with:

\[
\hat{y}_{i+1} = \alpha y_i + \alpha (1 - \alpha) y_{i-1} + \alpha (1 - \alpha)^2 y_{i-2} + \cdots,
\]

where \( 0 \leq \alpha \leq 1 \) is the weight (or smoothing) parameter that determines how rapidly the importance of past observations decreases. The ETS method can be expanded with additional parameters for trend and seasonality, which can be applied either additively or multiplicatively (i.e., do their effects increase with the level of the series). The \textit{forecast} package provides an ETS modeler with the function \texttt{ets()}. The fitted model is then used to generate the forecasts. Because the function is a modeler, the underlying model specification for the forecast may change every time the training set is altered.

**Seasonal-Trend decomposition**

Any time series can be coerced into a decomposition comprising trend, season, and error components. The meaningfulness of the decomposition depends on their respective strengths in the series: a completely random series will decompose, but the resulting trend and season components will be negligible in strength. This decomposition is most often achieved using locally estimated scatterplot smoothing (LOESS, or more commonly, the Savitsky-Golay filter) and referred to as Seasonal-Trend decomposition using LOESS (STL).

Forecasting with decomposition usually involves removing dominating components from the series, using the remainder to generate a base forecast with another method, and adjusting the base forecast with the extracted components to create the final forecast. The \texttt{stl()} function in the \textit{forecast} package decomposes the series, removes the seasonality of the series and feeds the seasonally adjusted series into the \texttt{ets()} modeler. The final forecast is generated by combining the ETS forecast with the extracted seasonal component.

**Multiseasonal-Trend decomposition**

The STL decomposition is only capable of extracting a single seasonality component from the series. Considering that multiple seasonalities are often present in daily, weekly, and monthly data, it is useful to test multiseasonal decomposition methods. The \textit{forecast} package offers a straightforward approach to this with the \texttt{stlf()} function, which calls the LOESS decomposition iteratively to extract any significant “remainder seasonalities” before feeding the seasonally adjusted series into the \texttt{ets()} modeler. The forecast is generated in the same manner as with the \texttt{stl()} function.
Autoregressive moving-average models (ARMA) are constructed using two different polynomials: the autoregressive part of order $p$, AR($p$):

$$\hat{y}_i = c_1 + \phi_1 y_{i-1} + \phi_2 y_{i-2} + \ldots + \phi_p y_{i-p},$$

and the moving-average part of order $q$, MA($q$):

$$\hat{y}_i = c_2 + \varepsilon_i + \theta_1 \varepsilon_{i-1} + \theta_2 \varepsilon_{i-2} + \ldots + \theta_q \varepsilon_{i-q}.$$

The AR($p$) is essentially multivariable linear regression using past observations going back $p$ time steps ("lagged" observations) as regressors. Equally so, the MA($q$) equates to regression on $q$ past noise terms $\varepsilon_i$. $\phi_i$ and $\theta_i$ denote the regression weights, and $c$ denotes the mean in the above equations. An ARMA forecast is computed then by combining these two with

$$\hat{y}_i = c + \varepsilon_i + \phi_1 y_{i-1} + \ldots + \phi_p y_{i-p} + \theta_1 \varepsilon_{i-1} + \ldots + \theta_q \varepsilon_{i-q}.$$ 

ARMA models assume stationarity (i.e., mean and variance remain constant over time and autocovariance depends only on past observations) of the target time series, which most time series do not initially satisfy. Stationarity can be coerced by differencing, producing a series of changes between consecutive observations, thus reducing the strength of possible trend and seasonality components. Sometimes this needs to be done more than once. ARIMA models (autoregressive integrated moving-average) are generalizations of ARMA models, where the series is differenced first to achieve stationarity. ARIMA models are specified with the order $p$ of the AR polynomial, the order $q$ of the MA polynomial, and the degree of differencing $d$.

ARIMA models can also utilize exogenous regressors (ARIMAX), in which case they follow the style of

$$\hat{y}_i = c + \varepsilon_i + \sum_{l=1}^{L} \beta_l x_l + \sum_{j=1}^{p} \phi_j y_{i-j} + \sum_{j=1}^{q} \theta_j \varepsilon_{i-j},$$

Here $x_l$ signifies the exogenous regressors and $\beta_l$ their weights. $L$ is the number of exogenous regressors. In our case, we added monthly seasonality as a dummy variable by using a one-hot encoded matrix of months. In practice, this generates additional regressors for each month, which are either “active” (1) or “inactive” (0) for any observation. An example of such a matrix for monthly data is presented below. The matrix needs only 11 columns to represent 12 months, as the final month is captured by the intercept (all variables zero).

\[\begin{array}{cccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}\]
The `forecast` package provides automatic ARIMA modeling capabilities via the `auto.arima()` function. The parameter `xregs` was used to introduce our exogenous regressor into the model.

**Dynamic regression**

When regression models for time series are built, the amount of autocorrelation in the residuals is often taken into consideration when deciding if the model is any good. If the residuals correlate significantly with each other, the model has failed to capture all the relationships contributing to the process. However, it may be possible to remedy the residual autocorrelation by modeling the residuals separately as an ARIMA process, thus capturing the more complex series dynamics in a separate component.

The `forecast` package allows fitting a regression model to a time series with the `tslm()` function. The function also returns the residuals from the fit, which can then be supplied for the `auto.arima()` function as is. The final forecast is the sum of both of these.

**TBATS**

If periodic, complex seasonality is suspected, TBATS models (Trigonometric, Box-Cox transform, ARMA errors, Trend, and Seasonal components) may be useful. They estimate Fourier term representations for the seasonalities in the Box-Cox transformed series (coercion into normality) using dynamic harmonic regression and use them in ETS state-space models, where the error term is modeled using an ARMA process. TBATS models can be automatically fitted with `tbats()` from the `forecast` package.

**Autoregressive neural networks**

The AR model uses a linear combination of lagged observations of the target series to model it. In the same vein, the autoregressive neural network (NNAR) takes lagged series observations as its input values and builds a nonlinear model of the series. In their simplest form, neural networks work by feeding their inputs through (“hidden”) layers of weighted “nodes,” whose weights are adjusted in the search for the minimum fitting error. Neural networks are good for finding complex nonlinear relationships between input values using only target values, but they lack transparency, and their structure selection is often a matter of trial and error.

We use three different autoregressive neural network implementations: a basic feed-forward neural network with one hidden layer, implemented with the `nnetar()` function from the `forecast` package and the `mlp()` function from the `nnfor` package. These use backpropagation to train the network. The `nnfor` package also offers an implementation for an extreme learning machine with the `elm()` function. The ELM is a feed-forward neural network that is trained without iterative tuning. Because the method selection system should be able to determine the best methods autonomously, we do not restrict the automatic fitting of networks enabled

**Method averages**

As any single method is limited to its parameter space and validity under its assumptions, their accuracy may suffer whenever the underlying series fails to meet those assumptions or follow the fitted parameters. A combination of several different forecasts may prove beneficial in improving the accuracy of individual forecasts, as shown by a considerable body of reviewed literature. In this study, we examine method combinations by averaging: AVG as the simple method average and W.AVG as an exponentially weighted average. Our weight-
ing scheme ranks methods based on the previous week’s performance and applies an exponentially decaying weight $\alpha = 0.5$ to them, thereby acting as a selection method:

$$\hat{y}_w = \sum_{j=1}^{N} \alpha^j \hat{y}_j,$$

where $N$ signifies the number of methods to average. $\alpha = 0.5$ was chosen for its rapid convergence to 1.

**Part C: Method Selection Visualized**

To better illustrate how the autonomous system chooses methods forecast by forecast, we plot the altered Finnish demand series (in black) and the forecasts (in white) generated by the system (here with 12-week selection period) against a background colour indicating the method chosen (Figure S1). The figure reveals that as the cyclic pattern continues, the variety of selected methods decreases to just a handful of best-performing methods. The code used for this plot can be found in our public Github repository.

**Figure S1**

![Figure S1](image)

**Part D: Peak Detection**

We tested peak detection abilities of different individual and selection methods using the real demand series from Finland. Peaks used for the accuracy analysis were chosen using the rolling median and rolling median absolute deviance (MAD). We chose to treat values over 2 MADs away from the median as anomalous (peaks and troughs) and compared the respective forecast accuracies of each method at these points. The peaks and the exclusion interval are visualized in Figure S2. The results are presented in Tables S3 and S4. The accuracy of predictions at the extreme peaks and troughs are less accurate as can be expected. The Auto-12 and Auto-3 beat all individual methods, but the differences between methods are not statistically significant at the 95% confidence level. The analysis code can be found in our public Github repository.
Figure S2

Table S3: Individual method MAPEs at peaks and troughs.

| Method | MA-5  | MA-7  | MA-9  | MA-12 | ETS   | STL   | STLF  | ARIMAX | DYNREG | TBATS | NNAR  | MLP   | ELM   | AVG   |
|--------|-------|-------|-------|-------|-------|-------|-------|--------|--------|-------|-------|-------|-------|-------|
| SNAIVE | 16.1  | 26.6  | 25.5  | 26.27 | 26.42 | 19.09 | 18.05 | 25.49  | 17.01  | 26.42 | 29.97 | 21.08 | 22.43 | 18.87 |

Table S4: Selection heuristic MAPEs at peaks and troughs.

| Method | AUTO-1 | AUTO-3 | AUTO-6 | AUTO-12 | AUTO-18 | AUTO-24 | W.AVG |
|--------|--------|--------|--------|---------|---------|---------|-------|
|        | 16.48  | 15.45  | 17.89  | 15.96   | 20.02   | 19.71   | 19.8  |

Supplement Bibliography

1. Gardner ES. Exponential smoothing: the state of the art. *J Forecast*. 1985;4(1):1-28.

2. De Livera AM, Hyndman RJ, Snyder RD. Forecasting time series with complex seasonal patterns using exponential smoothing. *J Am Stat Assoc.* 2001;106(496):1513-27.

3. Clemen RT. Combining forecasts: a review and annotated bibliography. *Int J Forecast*. 1989;5(4):559-83.

4. Finnish Red Cross Blood Service. production_forecasts. *Github*. 2021. Available from: https://github.com/FRCBS/production_forecasts
