1 Problem Description

We need to formulate and code an ordering system for an electronics company that produce one type of product. As data, we have monthly demand for the product from January-1996 to December-2005 ($\overline{D} = 93.78, \sigma_D = 12.23$ units). The company expects to use our system for the years 2006 and 2007. At each month of 2006 and 2007, the company will feed the software with a file including previous monthly demands and remaining inventory for the current month. The software would return the amount to order for the next month.

Products are not perishable, there is no discount rates nor ordering costs, only holding and shortage costs exists, and they do not change in time. There is a holding cost of \$1 per piece, so long as the total inventory is less than 90. If the inventory in a month is more than 90, then the first 90 units are charged \$1 per piece, and the rest \$2 per piece. The inventory to be considered for costs is the one left at the end of a month. The shortage cost is equal to \$3 per piece of unsatisfied demand. All product orders are made at the end of a month and received at the beginning of the next. Based on the historical data provided, the ordering can be done with precision up to the second decimal. By December 2005, the remaining inventory of the product was 73 units.

1.1 Exploratory Analysis

Data is loaded, processed, and filtered (if needed) in Python 3.6 (Anaconda distribution) using the known Pandas library. Visualizations are generated using Matplotlib/Seaborn libraries. Statistical tests and models are implemented using the popular Statsmodels package. A series of auxiliary functions are implemented in order to perform repetitive tasks and high-quality visualizations.

Given that our data corresponds to monthly consecutive demands from 1996 to 2005 (120 points) we would like to check if the series posses clear trend and seasonality. Figure 2 shows that this is the case, and furthermore that the residual of the series once its trend and seasonality has been removed is quite small. The residual is never more than 10 units, while the demand is in the order of a hundred. Figure 2 also shows that we seem to have a year seasonality, this is made clear on Figure 1. Observing the distribution of the demand for each month, the graph (left-side) shows that the demand and variance are quite distinct from one another.

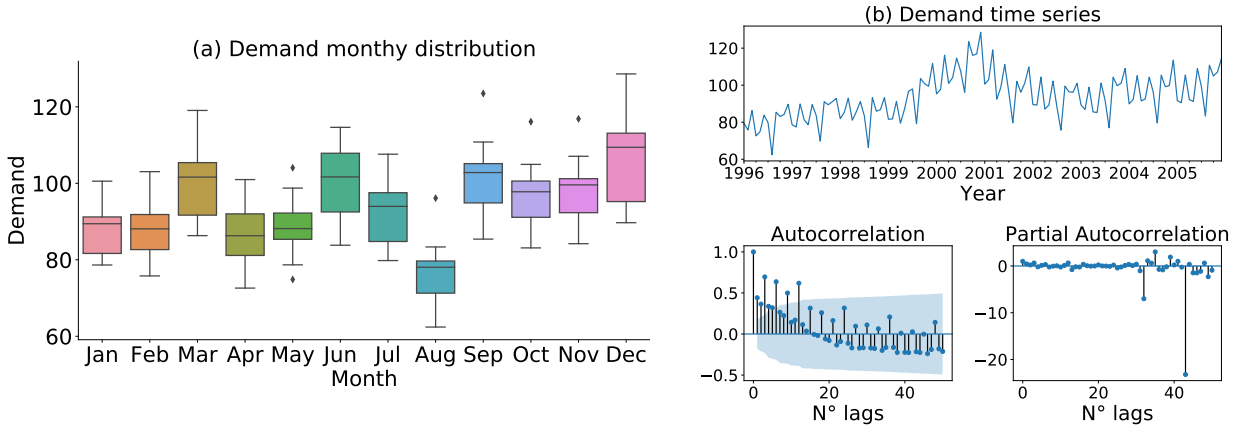

Figure 1: (a) Demand distribution per per month (full dataset). Significant differences on the demand levels can be found between months of the year. This is a relevant insight to contrast in future studies including more information regarding the product and its market, giving us the opportunity to match the patterns with certain events that affect the demand and incorporate them in our forecast model. (b) Autocorrelation and Partial autocorrelation plots of the full time series.

From the plots presented in Figure 1, we can observe how the monthly time-series present clear patterns: sales tend to be around average levels (87.73 ± 0.97) during the first half of the year (Jan-Feb and Apr-May), with significant peaks during March (100.96 ± 11.11) and June (100.35 ± 10.03). Sales experience

a strong drop during the summer (August, 76.91 ± 9.45). After the holidays, sales are increased in September (101.90 ± 10.72) reaching similar levels as in March and June. Demand reaches its maximum value and dispersion in December (106.41 ± 12.11), mainly due to the effect of the Christmas season. One possible explanation behind this pattern could be that our product is mainly demanded by students, presenting a clear yearly and quarterly seasonality.

In the same graph (right-side), both partial and simple autocorrelation (AC) can be observed for the full time-series. Here we notice that the AC values are significant for a large number of lags (12), but the partial autocorrelation (PACF) shows no significant spikes besides 42. As expected, the series are not stationary. Performing the traditional Augmented DickeyFuller test — null hypothesis indicating the presence of unit root or non-stationarity — we are not able to reject it ($p - value > 0.05$).

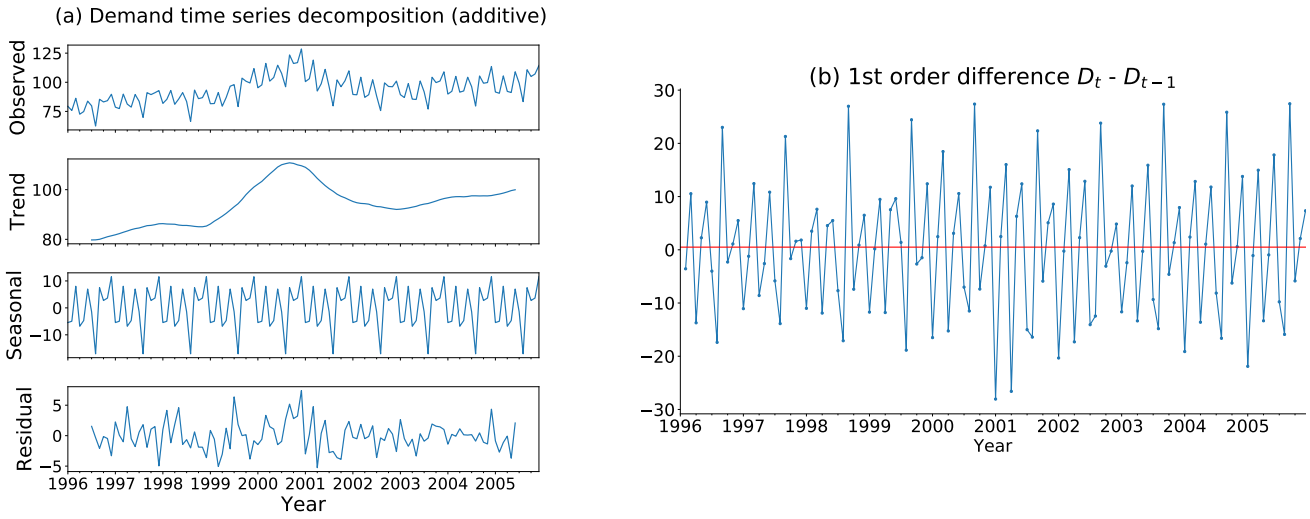

Figure 2: (a) Decomposition of monthly demand using an additive model by trend, seasonality, and residuals. (b) First order difference time series.

The first order difference is computed for the time-series $D'(t) = D_t - D_{t-1}$. From the visual results and stationary tests (Figure 2, (b), we can observe that the new time-series is in fact stationary: it presents a random behavior – no pattern is observed – and no significant ACF or PACF are found.

2 Proposed solution

2.1 Methodology And General Comments

In this section, we introduce the overall methodology of our solution and the mathematical notation associated. Let $t \in \{1, \ldots, 144\}$ represents the months from the years 1996 to 2007 (noticing that we have real demand data until 2005). Let y_t be the observed demand for month t and $\hat{y}_t$ be a forecast for that demand using information up to month $t-1$. Let ϵ_t denote a positive number that can be calculated using information up to month $t-1$. When needed, let s denote the seasonality cycle length.

Assuming that we have data until month t our algorithm suggests to order at period $t+1$ following the equation:

$$Order_{t+1} = \max\{\hat{y}_{t+1} + \epsilon_{t+1} - \text{Remaining Inventory}_t, 0\} \quad (1)$$

where f_{t+1} is the expected demand at period $t+1$ obtained by using any forecasting model. ϵ_{t+1} is a positive number which we call the bias at period $t+1$ for reasons to be explained later. The max term in the order formula is natural since we cannot order a negative amount. It is important to notice that if we knew the demand for the next period, it would always be optimal to order the exact demand minus the remaining inventory. The latter, since we have a lead time of one month, no discount rates, no limits on the ordering amount, and our product is not perishable. Of course, in practice, we expect to make

errors in any prediction system, meaning that $y_t - \hat{y}_t$ will be non-zero. Assume for now that those errors had zero mean and follow some symmetric distribution. If it were the case that holding and shortage cost were the same, it would be optimal to make $\epsilon_{t+1} = 0$. We empirically show that the zero mean assumption for the errors is adequate, but the loss is unbalanced. The holding cost is cheaper than the shortage cost. Therefore, it makes sense for ϵ_{t+1} to be positive to reduce the expensive shortage cost whenever it occurs. Then, we can understand ϵ_{t+1} as upwards biasing our demand prediction for period $t + 1$. The derivation of ϵ_{t+1} is explained in further detail in Subsection 2.4.

To obtain f_{t+1} we tested ARIMA, HoltWinters, and LSTM models. We consider these three models since they account for both seasonality and trend on its formulation. Furthermore, they are sufficiently simple considering that we have only 120 points of data, and they are fairly well known in the industry (models explained in 2.2). To test which model worked better, we used a rolling horizon approach. We compare the performance of these methods in Section 3.

It is important to mention that there could be cases where the data is not enough to run a model such as ARIMA, HoltWinters (Chatfield and Yar, 1988; Kalekar, 2004) or LSTM. For example, in the previous section, we argued that the dataset has a seasonal cycle of 12 months. Then, we cannot expect any method that uses seasonality to run or work well with less than one or two years of data. How we approach and implement our solution in these sorts of cases is discussed later in this section.

2.2 Demand Forecast

In order to evaluate multiple alternatives and check potential advantages/limitations of different approaches, we tested three classic forecasting models:

- **ARIMA(p,d,q)**: We fit a classic ARIMA model based on our visual and statistical analysis from the previous section. p seems to be significant up to 2 lags for the whole time-series, d equal to 1 is enough to obtain a stationary series and we can explore different values of q based on the PACF graph (0). Once the models are fitted using the Statsmodel library, we rank them by AIC (lower the best) and select that model for performing forecasts. The model is updated with each new observation.
- **Deep Learning**: Exploiting our familiarity with deep learning libraries in Python such as PyTorch and TensorFlow and due to its proven effectiveness in time-series forecasting (Gers et al., 2002; Karim et al., 2017; Malhotra et al., 2015), we implement a simple Long short-term memory (LSTM) network to predict the monthly demand. We stack an LSTM including a dropout layer and one final neuron for the next-step prediction. The model is trained using a mean squared error loss function and ADAM as the main optimization algorithm.
- **HoltWinters**: We fit a HolterWinters (HW) model defining a seasonality of 12 months using the statsmodels library in Python with identical functions as the popular R package (Hyndman et al., 2007). HW is a time series method that considers both seasonal and trend terms. This method is also known as the triple-smoothing method since it looks for constants to smoothly update the level, seasonal, and trend terms. For the trend, HW calculates a slope term which is then added to the level term. Both terms are smoothed at each time iteration. The method has variants in which the trend and seasonal terms are updated in an additive (stable) or multiplicative (unstable) way.

2.3 Inventory Model

In Section 3, we argued that the best forecast method was HoltWinter (HW) which by definition has unbiased errors. There, we also show that for 60 one-month predictions using a rolling horizon scheme, HW never overestimated the demand by more than 10 units. Given this scenario, considering the case of overestimating the demand for more than 90 units seems implausible with the current data. Also, in Table XX on Section 3 we show that the error made by HW depends on which month of the year the method is predicting for. Given the previous and that our dataset is limited in size, we make the following principled assumption. For a given month $t \geq s$, let $D_t = \{y_{t-ks} - \bar{y}_{t-kh}\}_{k \in k(h)}$ with $k(s)$ all strictly positive integers, such that $\bar{y}_{t-ks}$ exists. Then, D_t is the history of the errors in the same

Table 1: Performance Of Forecast Methods

	LSTM	ARIMA	HW
MAPE	3.78%	3.02%	1.76%
MSE	7.52	6.34	5.05

(a) Results obtained predicting the data from 2002 to 2005.

	HW	HW + bias
MAPE	1.75%	2.21%
MSE	5.05	7.01
Loss	3.76	3.06

(b) Adding the bias term to HW worsens its MAPE and MSE metrics, but reduces the loss function.

seasonality as t. We approximate $y_t - \bar{y}_t$ as a $\mathbb{N}(0, \sigma^2)$ random normal variable where σ equals to the standard deviation of the set D_t. Letting $Z \sim \mathbb{N}(0, \sigma^2)$, ϵ_t can be obtained as the solution of the following optimization problem:

$$\epsilon_t \in \arg\min_x \ \{3 \cdot \mathbb{E}[(Z - x)|Z \geq X]\mathbb{P}(Z \geq X) + \mathbb{E}[(x - Z)|Z \leq X]\mathbb{P}(Z \leq X)\}$$

Using a normal table is easy to see that ϵ_t is approximately 0.67σ. This way of obtaining ϵ_t is a type of news-vendor problem in which we allow the demand and quantity to order to be negative.

2.4 Small Data Case

If the monthly demand data we had, consisted of less than a year or two, it would not be adequate to run any of the methods described in Subsection 2.2. Whenever the data is so small, and no extra information is given, simple heuristics are an adequate way to go. The idea is to return a prediction with the best information available, even if the data is limited. In our case, we defined the data to be small when less than 48 months are available. For these cases, we implemented the following heuristic to predict the demand for the following month:

1. Less than 12 months of data. In this case, the prediction for the following month $\bar{y}_{t+1}$ is simply y_t. This heuristic is sometimes called persistence model.
2. Between 12 and 24 months of data. Here, we apply a simple linear regression to predict the demand for the following month. Given demands $\{y_1, \ldots, y_t\}$ for $t < 24$ we use $\{1, \ldots, t\}$ as the feature matrix – feature vector in this case – and $\{y_1, \ldots, y_t\}$ as the independent variable for the regression. Once we solve this linear regression we simply forecast which would be its value for the feature $t + 1$. The linear regression method is called from the sklearn Python package.
3. Between 24 and 48 months. In this case, we do apply one of our forecast methods to calculate $\bar{y}_{t+1}$. To calculate the bias term ϵ_{t+1} we use all available previous errors $\bar{y}_s - y_s$ with $s < t + 1$. This differs from what was proposed in the Subsection 2.4 as there the bias term only uses the errors from the same seasonality.

3 Results and Discussion

3.1 Forecast And Bias Term

Here we compare the performance of ARIMA, LSTM, and HolterWinters (HW) methods. As required by the company, we need the prediction for only one month in advance. Then, to compare the performance of the methods we use the following scheme. We use all demand information from January-1996 to December-2001 to predict the demand for January-2002. Then, we use all demand information from January-1996 to January-2002 to predict the demand for February-2002, and so on. We repeat the process until December-2005. The performance of the three methods is shown in Table 1a. Notice how HW outperforms the ARIMA and LSTM implementations.

It is important to notice how good is the prediction power of HW. It has a mean absolute percentage error (MAPE) of only 1.7% and a mean squared error (MSE) of 5.05. This tells us that overestimating

the demand for more than 90 units is highly unlikely. In fact, we use HW to predict the monthly demand for the years 2000 to 2005 and it never made a forecast who had an error of more than 15 units. This validates the assumption that, at least for the data given, we can safely remove the discontinuity on the holding cost function that occurs after the 90^{th} unit in inventory. The HW variant used here uses an 'additive' update for both the trend and seasonal terms. Using a 'multiplicative' update for the seasonal term, we obtained marginally better MAPE and MSE (1.73% and 4.89, respectively). However, we decided to use the 'additive' variant as its performance is almost the same and the 'multiplicative' variant is known to be unstable (Gelper et al., 2010). In fact, the 'multiplicative' variant is also called unstable HolterWinters as a denominator on its update may take the value of zero.

We now want to check how the bias term described in Subsection 2.4 performs against HW unmodified. To this end, we used the same rolling horizon scheme idea used to compare LSTM, ARIMA, and HW. We used the data from January-1996 to December-2000 along with HW to predict the monthly demand for the months of January-2001 to December-2005. The bias term for January-2003 is obtained by calculating the standard deviation of the errors committed by HW in January-2001 and January-2002 and multiplying it by 0.67. In the same fashion, we can obtain the security deposit for all the following months until December-2005. For example, for December-2005 we calculate the standard deviation of the errors from December-2000, December-2001, December-2002, December-2003, December-2004 and then multiply it by 0.67. In table 1b, we show the performance of HW against HW with the bias term. It is interesting to see that using HW alone performs better in terms of MAPE and MSE but has a worse loss value than when the bias term is used. The bias term perturbs the prediction of HW with the objective of reducing the shortage, whenever it occurs, at the trade-off of increasing the – cheaper – holding cost. This is shown in Figure 3. As a final comment, notice that the MAPE and MSE of the biased HW is better than the one obtained by ARIMA and LSTM.

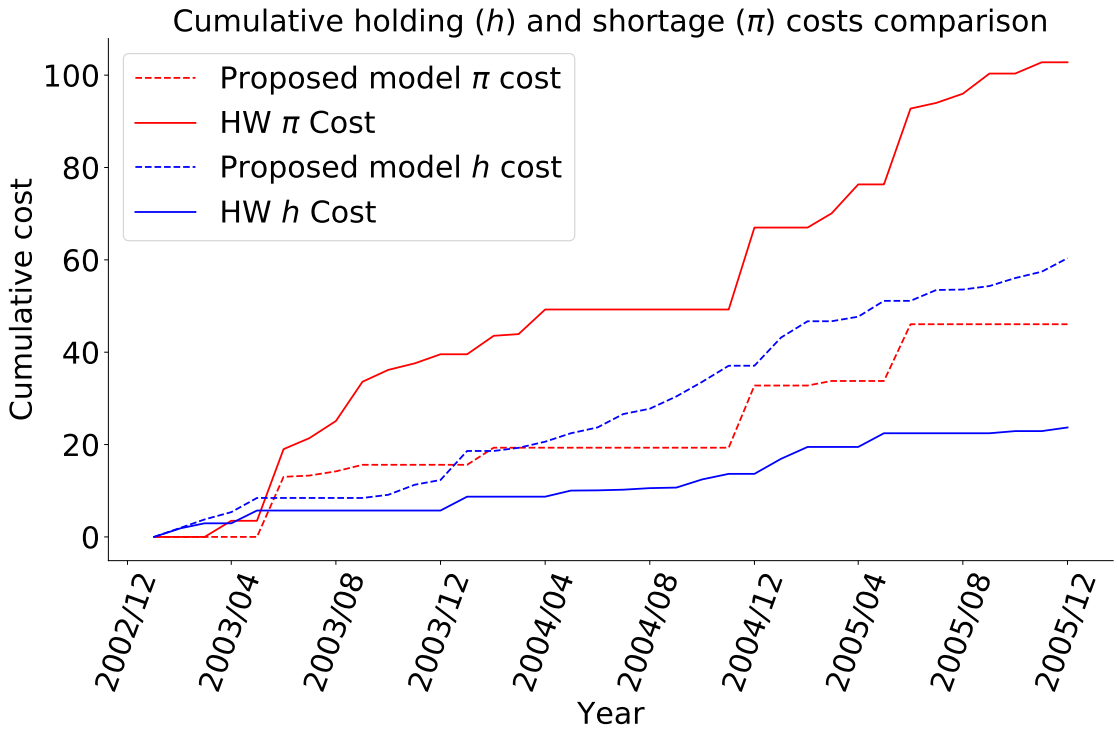

Figure 3: Cumulative holding (h) and shortage (π) costs comparison using Holter-Winter prediction and our proposed inventory model.

3.2 Limitations and Future Extensions

Despite the good performance of the proposed method, several limitations must be challenge in order to obtain a robust and flexible inventory model:

- The HoltWinters model is limited to univariate data and cannot handle irregular patterns. This

way, demand shocks triggered by events such as Christmas, Black Friday, among others, cannot be easily incorporated into the model. Alternative methods such as LSTM networks can naturally handle these situations, being more flexible and robust as more data is available.

- The optimal parameters of the model are not easy to interpret and do not capture useful information from outside the main time series.
- The current version of the proposed inventory model is not explicitly including the discontinuity of the holding cost function ($h = 2$ for each unit after the first 90 inside the warehouse). This is mainly because our analysis indicates that this situation is very unlikely, and we wanted to keep the model as simple as possible. However, our framework can naturally incorporate this component as part of the objective function of the proposed optimization model in future revisions.

References

C. Chatfield and M. Yar. Holt-winters forecasting: some practical issues. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 37(2):129–140, 1988.

S. Gelper, R. Fried, and C. Croux. Robust forecasting with exponential and holt–winters smoothing. *Journal of forecasting*, 29(3):285–300, 2010.

F. A. Gers, D. Eck, and J. Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*, pages 193–200. Springer, 2002.

R. J. Hyndman, Y. Khandakar, et al. *Automatic time series for forecasting: the forecast package for R*. Number 6/07. Monash University, Department of Econometrics and Business Statistics ..., 2007.

P. S. Kalekar. Time series forecasting using holt-winters exponential smoothing. *Kanwal Rekhi School of Information Technology*, 4329008(13), 2004.

F. Karim, S. Majumdar, H. Darabi, and S. Chen. Lstm fully convolutional networks for time series classification. *IEEE Access*, 6:1662–1669, 2017.

P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

4 Software Tutorial

We were informed that the company wants to use our prediction and inventory prediction system for the years 2006 and 2007 which are data that we do not know. This request is confusing since the lead time is one month, and at each month the company could re-run our system. Our solution to this was to offer the company two programs (callable python files). One would be used to predict the unknown demand for a next month, while the second works more as a simulator. For the second, a user would give a set of years of monthly demand and say take as the first 'n' of them are for you to train your system and following years are to test how would your system have worked. The first program is called 'PredictNextMonth.py' and the second 'Simulator.py'.

Both Python programs receive as an input a .csv or .txt file with three columns exactly with the same format as the file Ten-Year-Demand.csv given to us by the company. In that format, the file is composed of three columns, the first containing year information, the second a list with sorted counts, and the third sorted monthly demands. The rows elements are delimited by commas. Here we assume that the count number is module 12 equivalent, for example, both the count numbers 13 and 1 represent the month of January. This allows for a user to enter a file in which the first month does not necessarily correspond to January of a given year. Both python programs assume that the data is in the same folder as the program. To run the programs the user, need to be able to run Python 3.6 from the console, and having the Python packages Sklearn, Pandas, Numpy, CSV, and Statsmodels installed.

4.1 PredictNextMonth.py

This Python program assumes that the user wants to predict the next month of demand. The steps to run this program are the following:

1. Run in your computer console 'python RunOneMonth.py' in the folder in which you downloaded the python file (or point to the folder containing the file).
2. Then in the console, the program would ask the user to enter the file to be used for predicting the next month. We assume that the user wants to predict the month immediately after the last monthly data given in the file. To make the program easier, we require the user to have the .csv or .txt file to be used to be in the same folder as the program.
3. Finally, the program would require the user to enter the remaining inventory of the last month of the data.

Once the previous steps have been followed the computer will return a result through command lines that look as follows:

```
We will use the last three years of data for creating the security deposit.
This may take a minute to run.
Order Quantity for Next Month: 26.0
```

As you notice, the program says which method was used for calculating making the prediction and bias term (see more about this in the small data section 2.4)

4.2 Simulator.py

As its name implies, this python program is used to simulate how our program would have behaved under certain demand scenarios. The steps to run this program are the following:

1. Run in your computer console 'python Simulator.py' in the folder in which you downloaded the python file (or point to the folder containing the file).
2. Then, in the console, the program would ask the user for the name of the demand file. To make the program easier, we require the user to have the .csv or .txt file to be used to be in the same folder as the program.

3. The program would then require the user to enter the remaining inventory of the last month of the data used for training. As the console would say, just pressing enter assumes that 73 units would be used as remaining inventory.

4. Finally, the program would ask the user to enter how many months of data will be used as unknown data for the simulation. For example, entering 24 would mean to use all data for training up to the last two years. Then, in order, our simulator would calculate an order quantity to make for that month and then compare it to the actual demand experienced in that month. With that, it can calculate the remaining inventory at the end of the month and shortage and holding costs. It would then proceed to calculate an ordering quantity for the following month and so on.

Once the program finish it would create two .csv files in the folder, besides showing on console detailed inventories, cost and order information for the simulated months. The first '.csv' called 'Results.csv' has this detailed information, while 'Summary.csv' shows the total and average holding, shortage, and the general costs obtained.