Accelerate Multifactor Elasticity Optimization

Version 1.1

June 2023

Accelerate Multifactor Elasticity Optimization

The Optimization - Multifactor Elasticity Accelerator provides a fast implementation of a B2C demand and elasticity forecasting model. The use case is detailed in the Overview section and the other sections explain how to use the accelerator, depending on your role.

In this section:

Overview (Optimization - Multifactor Elasticity)

Optimization - Multifactor Elasticity Accelerator deploys a powerful machine learning to assess elasticity from a forecasting model. These functions allow to forecast the quantity that would be sold at a given price.

Approach

Multifactor Elasticity Accelerator relies on several steps:

- First, map the data and include as many features as possible to give enough context and information to the forecasting model.
- Build a first forecasting model and test outputs for a holding time frame in order to define the best model parameters.
- Train the model with the latest available data in order to get full knowledge of latest trends.
- Predict quantity for next periods, including expected quantity with some price variations.
- Fit an elasticity curve to match the expected quantity for these price variations and store those elasticity parameters, made available for other part of the solution.

Outputs

The results of a Multifactor Elasticity model are forecasted quantity and elasticity parameters assessed by the model. Some charts are also available to check outputs, such as demand forecast per product, the elasticity curves built from these forecasts and an evaluation providing the elasticity parameters that can be used by another model such as Accelerate Shelf Prices Optimization.

Limitations

- **Attributes / features** – Multi-factor elasticity relies on having the right features that impact the sold quantity. If those features are not available or the data are incomplete, output quality will be low.
- **Training periods** – Training period should be a complete period, meaning if the last period only contains partial data, like half a week, the outputs will be biased and probably wrong.

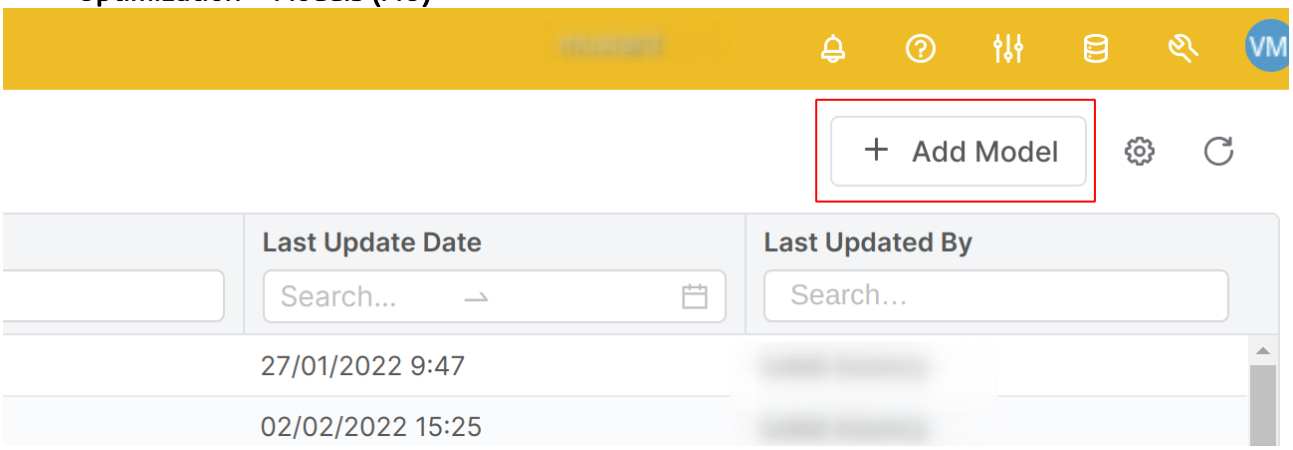
- **Sparse data** – Multi-factor elasticity relies on sold quantity from past transactions to forecast coming sales. However, if some products are new or barely sold, forecasts might be less accurate. To work around this potential issue, some aggregation are already in place, but some pre-processing or adjustment might be helpful in order to get accurate enough data.
- **No predefined extension point** – There is no out-of-the-box extension point defined for now. If you intend to add specific features, custom code should be written. (But then the accelerator becomes specific and it cannot be updated without extra effort to port those modifications.) Do not hesitate to report specific requirements and possible extension points to Pricefx.
- **Data requirements** – See Data Requirements (Optimization - Multifactor Elasticity).

Business User Reference (Optimization - Multifactor Elasticity)

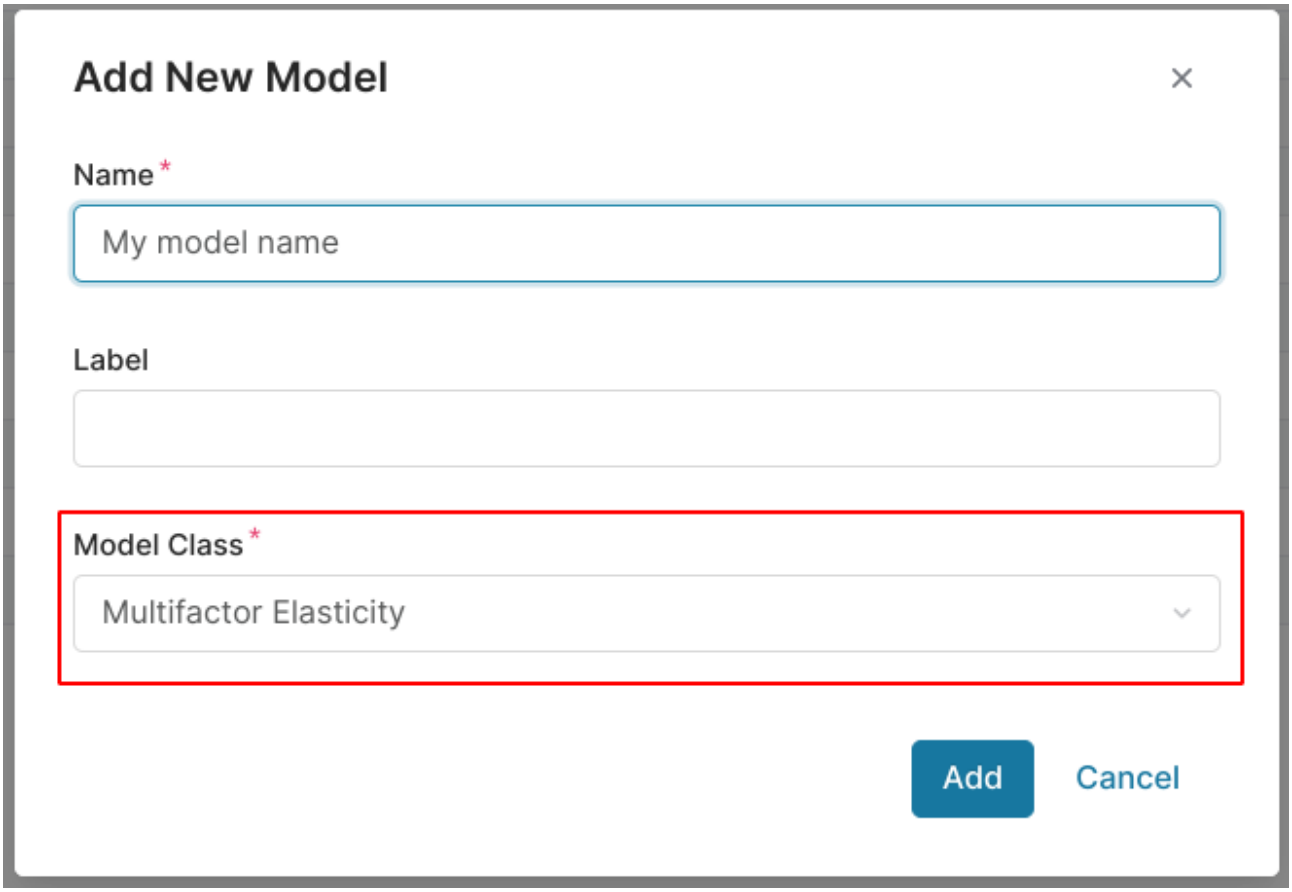
To run the Multifactor Elasticity Accelerator, you need to create a new Model Object from the **Optimization** > **Models (MO)** menu, using the Multifactor Elasticity logic to instantiate the machine learning model and give it the required parameters. The model will then be added to the list and will be editable.

New Model

1. Go to **Optimization** > **Models (MO)** and add a new model.

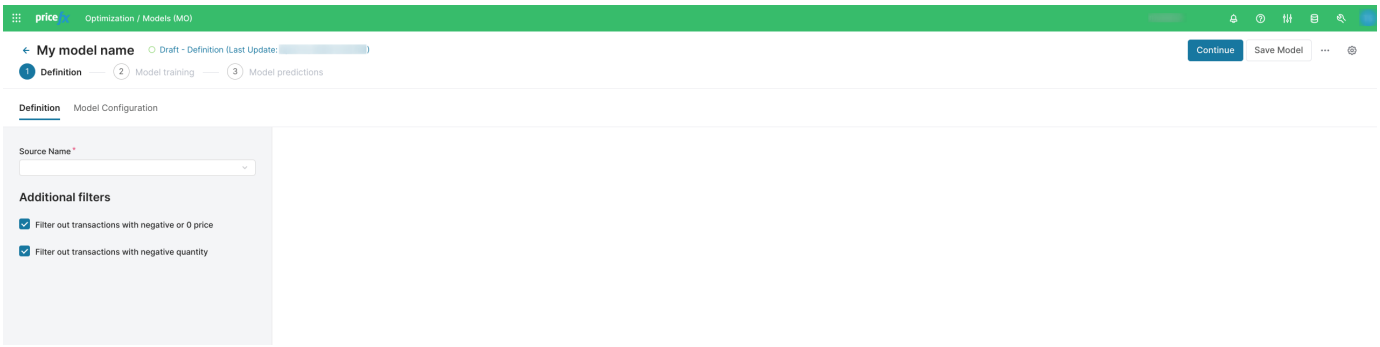


2. Define the name of the model and choose *Multifactor Elasticity* as **Model Class**.



3. The new model opens. It is based on the data you defined when deploying the Accelerator.
4. When a new model is opened for the first time, it will run a small calculation to generate a parameters table used for some configuration. This will only happen once.

The interface is:



Definition, Model Training, and Model Predictions are the steps of the model and will be explained below.

Definition Step

This step aims to define the input data and their mapping. You can also apply a filter to your data. There are two tabs:

- **Definition** defines the historical transactions and the scope of the forecast.
- **Model Configuration** defines the settings of the model.

Data Requirements

These are the data prerequisites to run a Multifactor Elasticity model.

One source is needed, it can be either a Datamart or a Data Source. It can have mixed types.

The source should contain a minimum of 2 years of transaction history, with the following columns:

Field	Required?	Comment
Period Start Date	Yes	The date of the start of the required aggregation period for the transaction (e.g. the start of the week for a weekly aggregation).
Product	Yes	
Shelf Price	Yes	Extended to the quantity.
Quantity	Yes	
Store	Yes	If there is only one store, set always the same value.
Additional categorical features	No	Here you may include any categorical columns that may influence sales, such as product hierarchies and product Pareto information.
Additional numerical features	No	Numerical features may include any numerical attributes that can be averaged over the selected time period, such as discounts, stock levels or seasonal events.

Definition Tab

In this tab, you define the transaction source and map it, as well as choose the aggregation. The general recommendations are:

- There are checkboxes at the bottom of the tab for automatic filtering of negative quantities and prices, it is recommended to keep these checked.
- **Period Start Date** is not the date of the transaction, but the date defining the period of the transaction, aggregated with the level of granularity needed – for a weekly forecast, it is the date of the Monday of each week.
- The **Shelf price** field must provide extended values.
- There is a checkbox to perform a log transformation of the quantity. This may help in some cases to produce a better forecast (for example, in cases where sales are heavily skewed towards low values – i. e., there are many long-tail products present in the data).
- **Time period field** defines the level of aggregation for the forecast – daily, weekly or monthly.
- The forecast generated in the final step may be extended up to 15 future time periods (15 weeks for a weekly forecast).
- Additional categorical and numerical features may optionally be added to the model from the source to improve the forecast.
    - Examples to include in the categorical features would be product hierarchies and product Pareto information. Categories should be unique for a given product and time period – fields such as channel and store should not be included here.

- Numerical features may include any numerical attributes that can be averaged over the selected time period, such as discounts, stock levels or seasonal events. Values do not need to be unique – they will be averaged over the time period.
- While numerical features can be included in the categorical features (for example, categories that use numerical codes), the same features should not be included in both categorical and numerical features at the same time. A warning will be displayed in this case.

Model Configuration Tab

In this tab, you may configure details of the forecasting model. The default values will work well in most cases for a weekly forecast. Some general recommendations are:

- The test set size defines how many time periods are held back to estimate the model's performance on data it has not seen before. It should be small compared to the number of time periods covered in the transaction source (up to ~20%).
- The number of past time steps for lags and differences and the rolling statistical window will help the model learn seasonal variation in the data. For example, for a weekly forecast, the default of 4 weeks will capture monthly variation. For a daily forecast it is recommended to increase these values to 30. For a monthly forecast, change to 3 (for quarterly variation) or 12 (for yearly variation) depending on data availability.
- The elasticities produced in the results step will be split according to the historical fraction of sales in each store, the store fraction window length defines how many of the most recent time periods to consider when calculating this fraction.
- The range of the elasticity prediction defines the price range of the elasticity curves calculated in the Model Predictions step. For example, a value of 50 will result in elasticity curves with a range of +/- 50% of the last known price for each product. This value is limited to +/- 100%. Note that predictions may be unreliable for ranges far outside of historical values.
- You may choose to perform automatic tuning of the model's internal parameters to improve the results. The default parameters will work in most cases but if run time is not a concern, increasing the number of tuning trials to 100 may provide small further improvements.
  - The tuning does not need to be run every time a model is calculated. If the model has been tuned before and the data has not changed significantly, you may deselect **Perform automatic model tuning** and select **Use last known parameters** from the buttons that appear to use the previously tuned parameters. If this option is selected before any tuning is done, default parameters will be used.
  - An advanced user may choose to apply their own configuration through choosing the **Use parameters from parameters table** option. The trainingParameters table, found in the parameters tables tab of the model tables may be edited according to the users' wishes.

Model Training Step

The Continue button in the Definition step will automatically run the calculation that starts the Model Training step. This calculation will gather the chosen fields from the transaction source and process the data into a time series aggregated at the chosen level. It will then enrich the data with the following extra features:

- Date related features to learn sales seasonality
- Lag, difference, and rolling statistics features chosen in the Model Configuration tab of the Definition step
- Product ages and recent sales activity features

The calculation will then split the data into a training set and test set according to the value chosen in the Model Configuration and use the training data to train a forecasting model, estimating its performance on the test set. The training results are displayed in the following three tabs:

Model Training Results Tab

This tab displays a number of charts that summarize the performance of the forecasting model:

- The Feature Importances chart shows how strong the influence of each feature is on the model predictions. This may be used to assess the impact of including features in the additional categorical and numerical features inputs. Features with low importance can be removed to reduce future training time without significantly affecting model performance. Features with very low importances are not shown. If there are a large number of features in your model you may need to extend the height of this chart to view them clearly.
- The Metrics table shows various statistics describing the performance of the model on both the training and test data.
- Scatter charts of the model predictions vs. the actual historical quantities for both the training data and the test data show how close the model's predictions are to the known truth. The black dashed line represents the line of perfect predictions. A good model should show points being close to this line for both the training and test data.

Train and Test Forecasts Tab

This tab displays forecasts alongside historical quantity for the product selected on the left panel, as well as the total forecast for all products. The test set is shaded in green. For the single product forecasts, the historical price is also displayed.

Training Curves Tab

This tab displays diagnostic metrics of the model performance over the course of the training.

Model Predictions Step

If you are happy with the training results you can click **Continue**. This will begin the final calculation for the Model Predictions step. This calculation will train a final forecasting model on the full historical data and produce a recursive forecast (for the number of time steps selected in the definition step). Using the forecast predictions with varying prices it will then calculate elasticity curves for each product for each date of the forecast and display the results in the following two tabs:

Demand Forecast Tab

This tab is very similar to the Train and Test forecasts tab in the Model Training step. In this case there is no test data – instead, the green shaded area represents the future forecast.

Elasticities Tab

This tab displays the predicted elasticity curves for the product and period start date chosen on the left. The chart shows the raw model predictions and a fitted s-curve, showing the r^2 score of the fit in the chart title. The table shows the elasticity curves for each product-date-store combination according to the calculated store fraction.

Data Requirements (Optimization - Multifactor Elasticity)

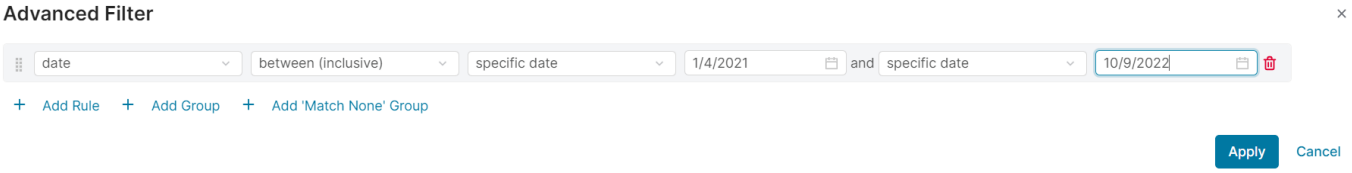
A Multifactor Elasticity model needs a Transactions Datamart to run. Here are the prerequisites for the fields.

One source is needed, it can be either a Datamart or a Data Source. It can have mixed types.

The source should contain a minimum of 2 years of transaction history, with the following columns:

Field	Required?	Comment
Period Start Date	Yes	The date of the start of the required aggregation period for the transaction (e.g. the start of the week for a weekly aggregation).
Product	Yes	Typically Product ID.
Quantity Measure	Yes	It is better to avoid negative or null values (they are filtered out by default by the model itself). It is possible to run the model with the log of the quantity. This may produce better outputs when there are small and large quantities in the dataset (e.g. many long-tail products).
Shelf Price (extended)	Yes	Typically end customer price x quantity Extended to the quantity. It is better to avoid negative or null values.
Store	Yes	Typically the store, but it can be channel or sales org. If no such feature exists, just provide a field with always the same value.
Additional categorical features	No	Here you may include any categorical columns that may influence sales. While not strictly required to run the model, this is key to getting proper outputs. Examples include Product Category, Competitor Name, Season, Channel, Special Event, Product Life cycle, Tag for promotions/discounts.
Additional numerical features	No	Numerical features may include any numerical attributes that can be averaged over the selected time period. It is highly recommended to at least include the transaction record discount rate.

It is also required to set the data filter in "Filter" to get complete periods (and not e.g. half week at the beginning or at the end of the scope).



Admin User Reference: Installation (Optimization - Multifactor Elasticity)

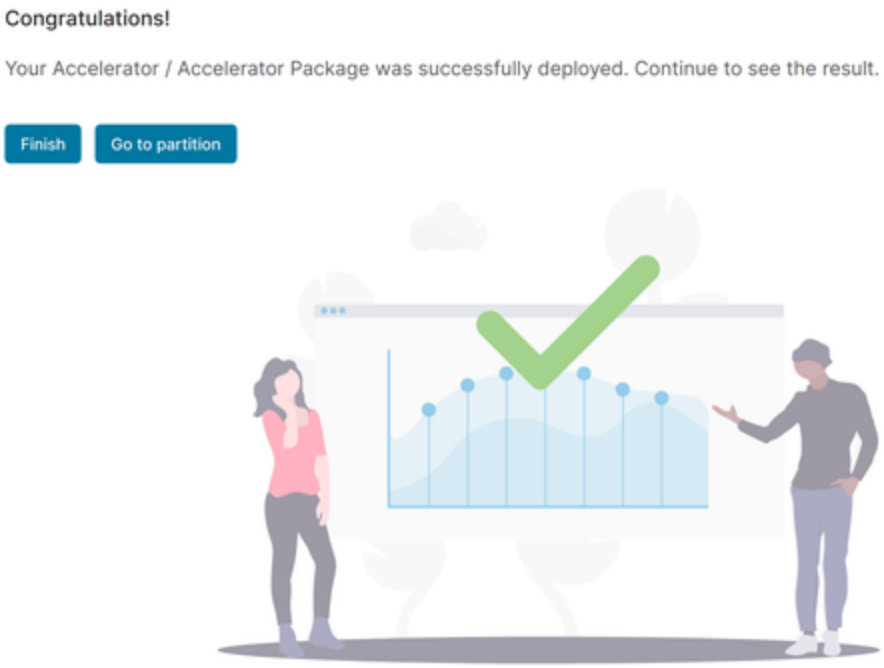
The Multifactor Elasticity Accelerator deploys the Multifactor Elasticity Model Class and the related logics.

Prerequisites

Before you start the installation of the Accelerator, ensure you have the according *Transactions* Datamart or Data Source. Note that it must contain the required fields (as specified in the Definition step) – create them if needed.

Deployment

1. Access PlatformManager at https://platform.pricefx.com/ and log in with your account or using O365.
2. Go to **Marketplace** > **Accelerator Packages**.
3. Find **Optimization - Multifactor Elasticity Package**.
4. Select your **Target Partition** from the drop-down menu.
5. Click **Continue** and wait until the deployment is complete.



Note: The Accelerator uses a Python Engine image. If the image `cregistry.pricefx.eu /engineering/pricefx-python-engine/datascience` was not yet enabled for your partition, the deployment process takes care of configuring it. Please refer to Python Engine for more information.

Technical User Reference (Optimization - Multifactor Elasticity)

This section details the Model Class and the logics that the Multifactor Elasticity Accelerator deploys. For each step, its aim, its outputs, and the main reasons to modify the logics are explained. If there is a need to modify the Python job, refer to Run Python Scripts.

In this section:

Multifactor Elasticity Model Class

The Multifactor Elasticity Model Class organizes a list of logics to create the model architecture. It is a JSON file that refers to some logics and it is transformed into an UI in the Pricefx platform that is organized in 3 steps:

1. **Definition** – Maps the transactions source and configures the model.
2. **Model Training** – Checks the quality of the model after its training.
3. **Model Predictions** – Forecasts quantities and shows elasticity functions details.

There are two types of logics: `calculation`, which writes tables in the model, and `evaluation`, whose purpose is only to display some results. The standard Model Class definition is documented in Model Class (MC).

All the logics of the Multifactor Elasticity Model Class follow a standard naming convention: first *MFE_* prefix, then the step name, then *calc* or *eval*, depending on the formula nature, then the tab it is referring to.

Library

The logic is **MFE_library**.

Aim of the logic

This logic contains some functions needed specifically for this Accelerator, such as reading its configuration from the application settings, applying the user filters in each part of the model, preprocessing the data for the charts, and many small helpers for the charts rendering. The elements `ParametersUtils` and `LabelsUtils` contain the names of many elements and fields of the models.

Common reasons to modify the logic

This lib is the place where to change the names inside the model, to reflect the user business vocabulary. Here you can also write a function to be used in different places of the model class.

Definition Step

This step runs the `MFE_definition_calc` calculation and displays two tabs: **Definition** and **Model Configuration**.

Calculation: Generate Parameter Table

- Aim of the logics
  This logic generates a parameter table containing default model training parameters. If the table already exists, it does nothing.
- Outputs of the calculation
  The `TrainingParameters` parameter table.
- Common reasons to modify the logic
  To change the default values in the table.

Definition Tab

The logics are **MFE_definition_eval** and **MFE_definition_eval_configurator**.

- Aim of the logics
  These logics define the Data Source and the mapping of the entries for the transactions, as well as some configuration of the outputs in the configurator. The main logic calls the configurator and the code for the dashboard portlets.
- Outputs of the evaluation
  Two portlets show the selected data fields for the in-scope and filtered-out transactions.
- Common reasons to modify the logics
  The mapping could be changed if a field is removed or added. Default values of the configuration could be changed.

Model Configuration Tab

The logic is **MFE_definition_eval_model_config_configurator**.

- Aim of the logics
  This logic defines more advanced configuration of the forecasting model.
- Outputs of the evaluation
  The inputs are stored to be used by the Python job which trains the model.
- Common reasons to modify the logics
  Default values of the configuration could be changed.

Model Training Step

This step runs the `MFE_train_calc` calculation and displays the tabs **Model Training Results**, **Train and Test Forecasts**, and **Train curves**.

Calculation: Train the Model

The logic is **MFE_train_calc**.

- Aim of the logic
  The aim of this logic is to gather the data and convert it into a form that can be used to train the forecasting model, store the data, train the model and store the results tables.

  The steps are as follows:

  - **StoreFraction.groovy** - This element calculates the fraction of sales for each product in each store for the user inputted `store_fraction_window` and saves it as a table.

- **DataQuery.groovy** - This logic takes the mapping and configuration from the Definition tab and defines a query on the source data that converts it from transactions to a complete aggregated time series for each product, filling missing quantity data with 0 and missing price data with the last known value.
- **DataTable.groovy** - Executes the query returned by *DataQuery.groovy* and saves it into the `Data` table.
- **Train.groovy** - This element collects the configuration for the forecasting model and triggers the Python job.
- **TrainScript.py** - This element is a Python script that loads the data, further processes them to add new feature columns based on the Model Configuration tab inputs and trains the machine learning model.

Outputs of the calculation

The Groovy elements output the `store_quantity_fraction`, and `Data` tables.

The Python job triggered in *Train.groovy* will write a set of model tables, including both the processed input data and the model results. It will also add a copy of the model as an attachment in a pickle file called `model.pkl`. This writing is done directly by the Python job and is *not related to a Groovy logic*.

The following tables are produced:

- `processed_data` - Final input data after all preprocessing.
- `X_train`, `X_test`, `y_train`, and `y_test` - Data split into training and test data.
- `train_preds` and `test_preds` - Training and test set model predictions, respectively.
- The tables with suffix `_curve` - Metrics data tables used for producing charts in the Model Training step.
- `model_parameters` - Model parameters used for training.
- `shap_values` - Data for the Feature Importances chart in the Model Training step.
- `metrics_table` - Data for the Metrics table in the Model Training step.

Common reasons to modify the logic

If you wish to change the model to use the per unit item price instead of the extended price, then change the following line in *DataQuery.groovy*:

```
(SUM($price) / NULLIF(SUM($quantity),0)) as price,
```

to

```
ISNULL((SUM($price * $quantity) / NULLIF(SUM($quantity), 0)), AVG($price))
as price,
```

This change will do two things: use the per unit item price instead of extended price to calculate the weighted average price, and default to the average price for a given time period if the total sales quantity in that time is 0.

Model Training Results Tab

The logic is **MFE_train_eval_preds_vs_actuals**.

Aim of the logic

This logic produces a dashboard with 3 charts and a table to visualize the model results. *predict_vs_actual_train.groovy* and *predict_vs_actual_test.groovy* produce two similar charts for the model training data and test data, respectively.

Outputs of the evaluation

Four portlets. Three charts showing the feature importances and scatter plots of the model predictions vs. the historical quantity for both the training and test data, and a table summarizing performance statistics.

Common reasons to modify the logics
Add, modify or remove visualizations.

Train and Test Forecasts Tab

The logics are **MFE_train_eval_forecasts** and **MFE_train_eval_forecasts_configurator**.

Aim of the logic
This logic produces two similar charts, one for a single product, the other for the sum of all products. The configurator selects a single product from the list of unique products in the `Data` table, which is used in *single_product_forecast.groovy* to display that products forecast.

Outputs of the evaluation
Two portlets showing the demand forecast for a single product and for the sum of all products. The shaded area in green represents results from the test data, the remaining data is from the training set.

Common reasons to modify the logics
Add, modify or remove visualizations.

Training Curves Tab

The logic is **MFE_train_eval_metrics**.

Aim of the logic
This logic displays the Training curve data. These are technical charts for assessing the model training.

Outputs of the evaluation
Three portlets displaying the training curves for the three different metrics in the three `_curve` tables.

Common reasons to modify the logic
To modify or remove visualizations.

Model Predictions Step

This step runs the `MFE_results_calc` calculation and displays the tabs **Demand forecast** and **Elasticities**

Calculation: Results

The logic is **MFE_results_calc**.

Aim of the logic
This calculation consists of two parts:

- **Elasticity.groovy** - This element collects the configuration for the elasticity calculation and triggers the Python job.
- **ElasticityScript.py** - This element is a Python script that loads the data and then trains a final machine learning model using the parameters in `model_parameters` on the full dataset, producing a future forecast. It then uses the predictions from this model to generate elasticity curves for each product and number of time periods of the forecast and fits a logistic ('S') curve to the data.

Outputs of the calculation
The Python job triggered in *Elasticity.groovy* will write a set of model tables. This writing is done directly by the Python job and is *not related to a Groovy logic*.

The following tables are produced:

- `elasticity_factors` - Fitted curve parameters needed to reproduce the elasticity curves.
- `elasticity_data` - Data used for plotting the elasticity curve charts.
- `forecast` - Future forecast outputs.
- `final_model_preds` - Predictions of the final model on the training data.
- `elasticity_r2_scores` - r^2 scores for the logistic curve fits.

Common reasons to modify the logic
There is no reason to edit this in general.

Demand Predictions Tab

The logic is **MFE_predictions_eval_predictions** and **MFE_predictions_eval_predictions_configurator**.

Aim of the logic
This logic produces two similar charts, one for a single product, the other for the sum of all products. The configurator selects a single product from the list of unique products in the `Data` table, which is used in *single_product_forecast.groovy* to display that products forecast.

Outputs of the evaluation
Two portlets showing the demand forecast for a single product and for the sum of all products. The shaded area in green represents results from the `forecast` table, the remaining data is from the training data.

Common reasons to modify the logic
Add, modify or remove visualizations.

Elasticities Tab

The logics are **MFE_predictions_eval_elasticities** and **MFE_predictions_eval_elasticities_configurator**.

Aim of the logics
This logic displays the results of the elasticity calculations.

Outputs of the evaluation
A portlet showing elasticity curves for the product and period start date selected in the configurator, with the r^2 score of the logistic curve fit in the title, as well as a table showing the `elasticity_factors` data with the corresponding r^2 score.

Common reasons to modify the logics
To modify the chart or the table output.

Release Notes (Optimization - Multifactor Elasticity)

Optimization - Multifactor Elasticity 1.1

This document summarizes major improvements and fixes introduced in the Accelerate Multifactor Elasticity Optimization package release version.

Version	1.1

Release Date	Jun 15, 2023

Table of contents:

New Features and Improvements

Description	ID
There is a new bar chart showing the feature importance and a new table with important metrics from training and test data sets. This helps to explain which features are important to the trained model to assist with feature selection and to increase explainability of the model.	PFPCS-6772
If not yet configured for the partition, Python Engine is enabled automatically during accelerator deployment.	PFPCS-6943
It is possible now to define the price range of the predicted elasticity curves.	PFPCS-6953
The maximum forecast length has been extended to 15 time periods.	PFPCS-6954

Fixed Issues

Bug Description	ID
In the Definition step, when you leave "Additional categorical features" or "Additional numerical features" empty (even if they are optional), you get an error during the Training step calculation.	PFPCS-6770
Fixed an issue where the default number of estimators used for model training was too low, limiting performance on large datasets. Model results are now reproducible for the same data and inputs.	PFPCS-6799
Some user inputs (number of lags, diffs, rolling window) in elasticity calculation were missing, as they were hard-coded and not using the user inputs. Those inputs are now used correctly in the calculation.	PFPCS-6958