5. Data modeling

First Model: the weights model

The weights model is used to learn and extract the **relative importance / price-driving impact of each feature** for a given family - to be used when weighting each feature in the similarity distance function.

The model used to extract the weights is primarily a LightGBM model, trained to predict the price of a CPC given its feature, which in turn allows us to **extra ct the weights of each feature** (i.e. price driving relevance) using SHAP values.

R² metrics

The R² metrics is used to measure the **model performance**, result is generally between 0 (bad) and 1 (perfect). We consider the model good enough between 0.4 and 0.9.

Note : the objective is not to have a perfect model, because fitting too well the training data often leads to a bad generalization.
In simpler words, it means that the provided data are too specific and detailed and while it allows the model to perform better on the training set, it will fail its prediction with any slight change in the new data we will provide at each campaign.

If R2 are too low, this could mean two things :

- We are missing some pricing levers (features) to explain price successfully. The ones we provide are not enough to have a good prediction on price.
- The dispersion of prices cannot be explained by data (due to human behavior, specific negotiation with the customers, etc.)
  For example, we can end up with 2 CPCs with exactly the same values for all features, with different prices.

==> In this use-case, we do not use the output of the model to predict prices directly.

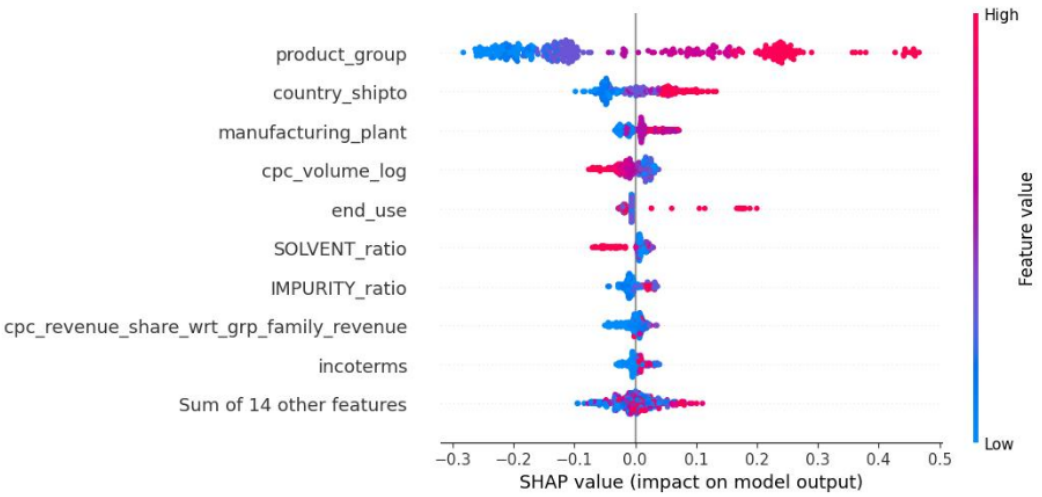
Example of R² output (for CS):

```
gbu_product_family,r2
Sulfosuccinate_Sulfosuccinamate,0.786
Specialty_Monomers,0.885
Phosphate_Esters,0.711
Guars,0.66
Alkoxylates,0.63
```

Features importance : SHAP values

What we are interested in is to understand the **importance** of each of our pricing lever (feature) in predicting the price. This is done by computing SHAP values when running the model.
This feature importance will then be used by the second model as coefficient to find neighbors for each CPC.

Specialty Monomers example (CS):



- Each dot represents a CPC
- Color is red when a CPC has a high value for the feature (y-axis), and blue for a low value. These are numeric values coming from the encoding step.
- The SHAP values (x-axis) represent the impact of a feature on the price prediction. This can be understood as the deviation from the average prices of the family due to the feature.

The strength of the SHAP value is that it is able to isolate the impact of a specific feature, ignoring all the other pricing features input in the model.

- Since we are modeling the price with a **log10** applied, and not the raw value, we can not directly interpret the x-axis as a "% price variation" on this graph.
  We need to apply the log10 inverse function (which is 10^x) to bring it back to the "normal" scale.
  We can then read it as a real percentage of price deviation from the average of the family.

- For the volume, we can see that high volumes (in red) have negative impact on price, which is expected.

Final feature importance (weight):

SHAP values are then used to define the weight (or importance) of every pricing lever.

feature_name	feature_weight
product_group	0.463
country_shipto	0.114
manufacturing_plant	0.055
cpc_volume_log	0.050
end_use	0.047
SOLVENT_ratio	0.043
IMPURITY_ratio	0.033
cpc_revenue_share_wrt_grp_family_revenue	0.026
incoterms	0.024
n_customers_per_product	0.020
n_products_per_customer	0.013
gbu_customer_seg	0.012
COMPONENT_ratio	0.012
group_volume_but_cpc_label	0.012
historical_unit_price_coalesce_ratio_on_12	0.012
RV_Gamma	0.010
RV_Lognormal	0.010
rev_outside_family_log	0.008
RV_Normal	0.008
RV_Logistic	0.007
RV_Uniform	0.007
historical_sales_coalesce_ratio_on_12	0.007
RV_Poisson	0.007

- We first compute the absolute average of every CPC SHAP value by feature (average SHAP value of every dot in previous graph).
- We then normalize the values between features so all the weights in the table sum to 1. It allows us to compare all the different features that were independent since then.
  For example, the product_group is our most important feature and explains 46% of the dispersion around the family price average.
- Random variables (RV_Gamma, RV_Normal, etc.) are also introduced into the model to check whether features are more important than a random variable in predicting price variation.

Hyperparameter grid search

For each model run, the weights model will be retrained using LightGBM parameters specified in the select_weights_model function (inside similarity library). LightGBM is sensitive to the specific hyperparameters, and may overfit if care is not taken. We therefore select the hyperparameters **individually for each family**, and find them using a **hyperparameter grid search**.

This functionality is implemented in the weights model, and can be run at the bottom of the weights model recipe for a specific family (toggled off by default) - printing out the optimal parameters. These parameters are then moved to the model parameter section for this family in the config, so future runs use these optimal parameters. We typically run the hyperparameter search when we're adding a new family, or if large changes are made to an existing one (e.g. many new features).

The grid search will train a model for each of the combinations of parameters specified in the weights model (in the library) and pick the ones with the best cross validated R2-score. To ensure best possible parameters, we have a **rich search space - making the procedure time intensive** (~30 minutes). Faster times can be achieved by reducing the amount of grid search parameters in the library implementation.

Also note that this search will try Lasso and Random Forest models, as a reference. Generally, LightGBM should be able to achieve better results than these, so if these models come out as the best, this indicates that manual tuning of LightGBM parameters should be done.

Second model : Find comparable CPCs

To find comparables, we compute a similarity distance between all CPCs two by two. This computation is based on the numeric values of the CPC features on which we apply the **feature weight from the first model** as a factor.

- The algorithm used for this is named "*K-nearest neighbors*".
- The computed distance is a "*cosine*" distance, which is commonly used in data science.
- If 2 CPCs have similar values for an **important variable**, it is more impactful than CPCs having similar values on a **low importance variable.**
- **Volume** feature is **excluded** from similarity distance calculation. Indeed, volume is treated after in its own "volume adjustment" step and the objective here is not to find comparables which have similar volume, but similar characteristics.