Project variables documentation

The objective of this page is to provide a detailed description of the global variables used in Dataiku's workflow, these variables are grouped as follows:

Environment

```
"env": "master"
```

This variable is used to differentiate the file storage between a "dev" and a "master" location.

- "dev" contains all the runs done on the dev and feature branches. To be used on said branched.
- "master" contains the automated runs of the "master" branch. This is also the only env that is used in prod automation due to the fact that we deploy the "master" project.

This variable is always set to "master" in the *Global* variables and overriden by "dev" in the *Local* variables in order to not have to do any change between design and automation.

Versioning

These two variables are used to define the version of the runs in Dataiku folders:

- the version name must be updated for each new run in dev or feature branches. On the master branch and in automation, the version name will stay the same, only the date will change.
- the "*version_date*" refers to the first forecast month in the extracted forecast data, and is updated automatically in a scenario step.

```
"version_name": "Q3.3_all_families_run",
"version_date": "2023-09-01",
```

To get the version name in the python recipes, we used the ***get_config_version()*** function from *config_helpers.py* file.

Parameter gsheet

Directly used in the gsheet settings for parameters update, this variable contains the URL of the **PROD** gsheet in the **global** variables.

It is overriden in **all design projects** (dev and master included) by the **local** variable containing the **DEV** gsheet.

Global variables

```
"parameter_gsheet": "1onxXZjM3MH-GISC47L41pUOvfwi0ySgqXxM4rLURupw"
```

Local variables

```
"parameter_gsheet": "1AK6UgFg9KeBHR3_--zGJ9S930yBV3ckLRjmfk4-qiIk"
```

Taxonomy gsheet (SpP only)

Directly used in the gsheet settings for taxonomy, this variable contains the URL of the **PROD** gsheet in the **global** variables.

It is overriden in **all design projects** (dev and master included) by the **local** variable containing the **DEV** gsheet.

Global variables

```
"taxonomy_gsheet": "1ASWg_uEPWZF_YOhydSD32gTxDL_cjSuJoV_FpdFjmGo"
```

Local variables

```
"taxonomy_gsheet": "1pz4-k47el-DsoK1iI1E9CLxWeC-v-OkWFTxUDwU2yzo"
```

GBU

Specific GBU measures and identifiers, which must be updated for each new GBU.

"*Families_in_scope*" contains the names of currently integrated product families for the GBU

"All_validated_families" contains all of the families that already have been validated on both the technical and business side. To be updated each time we a dd a new product family.

A family can therefore have been validated but not be used in current runs if it has been deactivated by the business.

```
"GBU_measures": {
    "historical_revenue": "historical_sales",
    "historical_volume": "historical_volume",
    "historical_price": "historical_unit_price"
  },
  "GBU_identifiers": {
    "id_key": "cpc",
    "product_key": "material_code",
    "customer_key": "shipto_code",
    "soldto_key": "soldto_code",
    "soldto_group_key": "soldto_group",
    "shipto_key": "shipto_code",
    "family_key": "gbu_product_family",
    "sales_key": "forecasted_sales",
    "volumes_key": "forecasted_volume",
    "prices_key": "computed_unit_price"
  },
  "families_in_scope": [
    "Sulfosuccinate_Sulfosuccinamate",
    "Specialty_Monomers",
    "Phosphate_Esters",
    "Amines"
  ],
  "all_validated_families": [
    "Sulfosuccinate_Sulfosuccinamate",
    "Phosphate_Esters",
    "Alkoxylates",
    "Amines"
  ],
```

To get these variables, we used the ***get_config_gbu_ids()*** function from *config_helpers.py* file.

The values of "*families_in_scope*" variable are used by the GBU variable "*family_key*" to select families in the scope.

Product composition

variables used to process product composition data, in particular to select the component to be used, specify the identifiers of the product, component type, measure and unit.

```
"product_composition": {
    "component_values": [
      "COMPONENT",
      "IMPURITY",
      "SOLVENT",
      "ADDITIVE",
      "Z_CONST"
    ],
    "product_identifier": "EHS_Product",
    "component_type_identifier": "Component_Type",
    "measure_identifier": "Average",
    "unit_identifier": "Unit"
  },
```

these variables are used as arguments to the ***compute_product_composition()*** function to compute the product composition features in this recipe.

Pre-processing

variables used in the various data preparation stages:

- "*preprocessing_filters*" to filter CPCs on the basis of column values *(for example, here we filter all CPCs with "SSPH" in the product_group column),* used as arguments for the ***data_filters()*** function in feature engineering recipe.
- "*imputers*" to specify the imputation strategy to be used for each characteristic with nan values, used as arguments for ***simple_imputer()*** function in feature engineering recipe.
- *"replace_with_null"* lists columns for which there are default values e.g. "Not assigned" that we want to replace by null values before other treatments like imputation. For each column as key we can have a list of the values to replace.
- "*outliers*" to activate the outliers detection and deletion based on the specified "*method*" for the selected variables "*vars_to_check*", used as arguments for ***detect_outliers()*** function in outliers recipe.
- "*categorical_encoder*" to specify the type of encoder to be used for categorical features, used as arguments for ***encoding_data()*** function in encoding recipe.
- "*ordinal_encoder*" to specify the type of encoder to be used for ordinal features, used as arguments for ***encoding_data()*** function in encoding recipe.

```
"preprocessing_filters": {
    "product_group": [
      "SSPH"
    ],
    "material_name": [
      "AEROSOL OT-100 SURF 25KG FBD WHSKIN",
      "AEROSOL OT-100 SURF 11KG W/LBL BOX"
    ],
    "end_use": [
      "Hpc-Api"
    ]
  },
"replace_with_null": {
    "end_use": [
      "Not Assigned"
    ],
    "gbu_customer_seg": [
      "Not valid",
      "Not yet assigned"
    ],
    "market_cluster": [
      "Not Identified",
      "-1"
    ]
  }, "imputers": {
    "most_frequent": [
      "manual_region_SS",
      "manual_region_SM",
      "product_group"
    ],
    "constant": {
      "n_competitors": 1,
      "historical_unit_price_coalesce_ratio_on_12": 1,
      "historical_sales_coalesce_ratio_on_12": 1,
      "historical_unit_price_ratio_3_on_12_month": 1
    },
    "mean": [
      "COMPONENT_ratio",
      "IMPURITY_ratio",
      "SOLVENT_ratio",
      "n_components"
    ]
  },
"outliers": {
    "remove_outliers": false,
    "method": "IQR",
    "vars_to_check": [
      "cpc_price_log",
      "cpc_volume_log",
      "cpc_revenue_log"
    ],
    "n_vars_out": 1
  },
"categorical_encoder": "TargetMean",
"ordinal_encoder": "Ordinal",
```


also, to create sales evolution features, a set of parameters for the ***get_interval_ratio()*** function are declared as global variables.

The function used in the sales evolution features recipe calculates a ratio of the chosen column in "*evolution_columns*" on one or several month ("*numerator_list*") in regards to another set of months ("*denominator_list*").

```
"evolution_features_params": {
    "evolution_columns": [
      "historical_sales",
      "historical_volume",
      "historical_unit_price"
    ],
    "numerator_list": [
      1,
      3,
      6
    ],
    "denominator_list": [
      12
    ]
  },
```

Weights Model

variables to customize the weight model:

- "*target*": the target variable of the model
- "*id_col*": the column index of the dataset
- "*SHAP_VISUALS*": list of the features for which we store the visuals of the SHAP values
- "*shared_features*": the features shared by all scope families, divided into "numerical_features", "categorical_features" and "ordinal_features"

```
"model": {
    "target": "cpc_price_log",
    "id_col": "cpc",
    "SHAP_VISUALS": [
      "cpc_volume_log"
    ],
    "shared_features": {
      "numerical_features": [
        "cpc_volume_log",
        "cpc_revenue_share_wrt_grp_family_revenue",
        "rev_outside_family_log",
        "n_products_per_customer",
        "n_customers_per_product",
        "historical_unit_price_coalesce_ratio_on_12",
        "historical_sales_coalesce_ratio_on_12",
        "IMPURITY_ratio",
        "COMPONENT_ratio"
      ],
      "categorical_features": [
        "incoterms",
        "manufacturing_plant",
        "product_group",
        "country_shipto",
        "end_use",
        "gbu_customer_seg"
      ],
      "ordinal_features": {
        "group_volume_but_cpc_label": [
          "0_one_cpc",
          "1_small",
          "2_medium",
          "3_big",
          "4_top"
        ]
      }
    },
```

- "*family_features*": used for specific features for each family with the same breakdown as described above

```
"family_features": {
      "Sulfosuccinate_Sulfosuccinamate": {
        "numerical_features": [
          "n_competitors"
        ],
        "categorical_features": [],
        "ordinal_features": {}
      },
      "Specialty_Monomers": {
        "numerical_features": [
          "SOLVENT_ratio"
        ],
        "categorical_features": [],
        "ordinal_features": {}
      },
      "Alkoxylates": {
        "numerical_features": [
          "SOLVENT_ratio"
        ],
        "categorical_features": [
          "lip_2",
          "chemistry"
        ],
        "ordinal_features": {}
      },
      "Phosphate_Esters": {
        "numerical_features": [
          "SOLVENT_ratio"
        ],
        "categorical_features": [],
        "ordinal_features": {}
      },
      "Guars": {
        "numerical_features": [
          "SOLVENT_ratio"
        ],
        "categorical_features": [],
        "ordinal_features": {}
      },
      "Amines": {
        "numerical_features": [
          "SOLVENT_ratio",
          "COMPONENT_ratio"
        ],
        "categorical_features": [],
        "ordinal_features": {}
      }
    }
  },
```

these variables are used in the compute Weighting dataset recipe.

Similarity Model

variables linked to the initiation of the similarity model

- "*target*": price column to use as target for the model
- "*id_col*": CPC identifier
- "*cohort_size_cap*": maximum number of comparables
- "*min_cohort_for_pricing*": minimum number of comparables allowed to recommend a price, if we have less than this number then we display the original price with no increase.
- "*max_cohort_used_for_pricing*": number of comparables used to compute the median price of a target and recommend the price.
- "*min_impact_for_pricing_euro*": minimum impact on revenues to display a recommended price for a CPC, otherwise original price will be displayed instead.
- "*price_recommendation_cap*": Relative maximum increase and decrease the model can recommend (e.g. a value of 0.3 means a maximum of 30% change based on the original price of the CPC). This is the global cap used by default for all families.
- *"local_price_recommendation_cap"*: Overrides the global cap above by a per-family cap if defined.
- "*features_weight_zero*": features that will have no importance in the model (for example, volume since it is already handled specifically in the volume adjustment step).

- "*match_percentage_similarity_threshold*": the threshold under which we stop considering a CPC comparable with the target.
- "*sim_threshold_black_list*": allows to skip above threshold for the families defined in the list.
- "*match_percentage_cols*": list of columns used for the match percentage calculation which is part of the similarity threshold definition.

```
"target": "cpc_price_log",
"id_col": "cpc",
"cohort_size_cap": 20,
"min_cohort_for_pricing": 3,
"max_cohort_used_for_pricing": 10,
"min_impact_for_pricing_euro": 15000,
"price_recommendation_cap": 0.3,
"local_price_recommendation_cap": {
    "udel": 0.2,
    "radel": 0.2,
    "veradel": 0.2,
    "amodel": 0.2,
    "ketaspire": 0.2,
    "ryton": 0.2,
    "torlon": 0.2,
    "fluids": 0.2,
    "pvdc": 0.2,
    "halar": 0.2,
    "ixef": 0.2,
    "kalix": 0.2,
    "solef": 0.2,
    "tecnoflon_fkm": 0.2,
        "tecnoflon_ffkm": 0.2
},
"features_weight_zero": [
  "cpc_volume_log",
  "group_volume_but_cpc_label"
],
"match_percentage_similarity_threshold": 0,
"sim_threshold_black_list": [],
"match_percentage_cols": {
   "shared": [
      "country_shipto",
      "end_use",
      "gbu_customer_seg",
      "product_group"
   ]
},
```

these variables are used in the compute similarity dataset and compute price recommandation recipes.

Hard boundaries

variables to apply several rules on comparables:

- "*hard_boundaries*": lists the columns for which CPC are not comparable if they do not have the same value. The lists are product family specific.
- "*hard_boundaries_inverse*": lists the columns for which CPC should not have the same value.
- "*volume_hard_boundaries*": additional hard-boundaries based on the volume difference between the CPC
    - flag : identifies if the volume hard boundary is applied for the product family
    - threshold : volume factor used for the hard boundary. For example, a threshold of 10 means that a CPC with a volume of 100 can only be compared to CPCs having a volume between 100*0,1 to 100*10.

```
"hard_boundaries": {
    "Sulfosuccinate_Sulfosuccinamate": [
      "product_group",
      "manual_region_SS"
    ],
    "Specialty_Monomers": [
      "product_group",
      "manual_region_SM"
    ],
    "Alkoxylates": [
      "lip_2",
      "chemistry"
    ],
    "Phosphate_Esters": [
      "manual_region_Ph_Esters",
      "COMPONENT_nb_Ph_Esters"
    ],
    "Guars": [
      "product_group",
      "manual_region_Guars"
    ],
    "Amines": [
      "product_group"
    ]
  },
  "hard_boundaries_inverse": [
    "shipto_code",
    "soldto_code"
  ],

  "volume_hard_boundaries": {
   "Sulfosuccinate_Sulfosuccinamate": {
      "flag": 1,
      "threshold": 10
    },
    "Sulfosuccinates_Healthcare": {
      "flag": 1,
      "threshold": 10
    },
    "Specialty_Monomers": {
      "flag": 1,
      "threshold": 10
    },
    "Alkoxylates": {
      "flag": 1,
      "threshold": 10
    },
    "Phosphate_Esters": {
      "flag": 1,
      "threshold": 10
    },
    "Guars": {
      "flag": 1,
      "threshold": 10
    },
    "Amines": {
      "flag": 1,
      "threshold": 10
    },
    "Solutions_Polymers": {
      "flag": 0,
      "threshold": 10
    },
    "Esters": {
      "flag": 0,
      "threshold": 10
    }
```

these variables are used in the compute similarity dataset recipe.

Adjustment Model

variables used for the volume and the group volume adjustment:

- "*volume_feature*": the volume feature to use for adjustment.
- "*fit_curve*": boolean allowing to activate the volume adjustment step.
- "*group_vol_labels*": defined bins to perform the adjustment.
- "*perform_group_adjustment*": to activate the group volume adjustment.
- *"small_volume_weight"*: used to reduce the weight of smaller CPC to improve curve fit.
- *"big_volume_weight"*: used to increase the weight of larger CPC to improve curve fit.
- "*small_volume_q*" / "*big_volume_q*": Quotients of the extreme volume data to be replaced by the defined weights
- "*group_adjustments*": parameters by family used to perform the group volume adjustment.

```
"adjustment_model": {
    "volume_feature": "cpc_volume_log",
    "fit_curve": true,
    "group_vol_labels": [
      "0_one_cpc",
      "1_small",
      "2_medium",
      "3_big",
      "4_top"
    ],
    "perform_group_adjustment": false,
    "small_volume_weight": 1.8,
    "big_volume_weight": 0.8,
    "small_volume_q": 0.2,
    "big_volume_q": 0.9,
    "group_adjustments": {
      "default": {
        "0_one_cpc": 0,
        "1_small": 0,
        "2_medium": 0,
        "3_big": 0,
        "4_top": 0
      }
    }
  },
```

these variables are used in the compute adjust results recipe.

Cross-validation

Variables to activate cross-validation and use the finetuned parameters for the LGBM model.

Default params are defined for families that have not been optimized yet.

- "*run_cross_val*": boolean allowing to activate the grid-search.
- "*use_hyper_params*": boolean allowing to use the saved finetuned parameters for the LGBM model for each of the families (finetuned params dataset).
- "*families_to_optimize*": families for which we optimize hyperparameters and store in the finetuned params dataset.
- "*cv_params*": lists all the hyperparameters we want to try when activating the grid search for the weights model on a specific family.

```
"run_cross_val": false,
  "use_hyper_params": true,
  "families_to_optimize": [
    "Sulfosuccinate_Sulfosuccinamate",
    "Phosphate_Esters",
    "Alkoxylates",
    "Amines"
  ],
  "default_params": {
    "n_estimators": 100,
    "max_depth": 6,
    "learning_rate": 0.1,
    "min_child_samples": 10,
    "R2 score": 0
  },
  "cv_params": {
    "lgb": {
      "n_estimators": [
        25,
        50,
        100
      ],
      "max_depth": [
        3,
        5,
        9
      ],
      "learning_rate": [
        0.05,
        0.1
      ],
      "min_child_samples": [
        10,
        20,
        50
      ]
    }
```

These variables are used in the compute optimized hyperparameters recipe.

Metrics variables

Metrics_dict contains global variables used for the monitoring of the project, through the metrics and checks.

- *"Project_error_level"* : can contain either "ERROR" or "WARNING" and will override all the local error level of the checks inside the project. ***To be used carefully only on dev environments for specific testing purposes.***

```
"metrics_dict": {
    "project_error_level": ""
  }
```

Manual_files_checks lists the families checked for each of the manual files of the project.

```
  "manual_files_checks": {
    "regions": [
      "amodel",
      "ryton",
      "tecnoflon_ffkm"
    ]
  },
```

External variables update

Some of the variables of the project can be updated in an autonomous way by the users. This allows them to change some of the business rules without any action required from a developer or data scientist. For more information on this process, please refer to the dedicated documentation here.

In order to define the variables in the scope of these updates and to provide enough information to the users, we use a dedicated dictionary from the project variables itself :

From the example below :

- "*minimal_impact_threshold*": name of the variable that will be used as identfier in the external parameters Gsheet.
- "*technical_path*": path of the variable in the Dataiku global variables dict, each level being separated by a '.' character.
- "*input_type*": helps understanding what is the requested type of the variable to update.
- "*description*": user friendly explanation of the usage of the variable.

```
"external_variables_update": {
    "minimal_impact_threshold": {
      "technical_path": "model.min_impact_for_pricing_euro",
      "input_type": "float",
      "description": "Impact threshold under which the price recommendation for a CPC will not considered.
Applies to both positive and negative impacts. Only one numerical value should be input"
    },
    "recommendation_cap": {
      "technical_path": "model.price_recommendation_cap",
      "input_type": "float",
      "description": "Absolute gap that can not be exceeded between the original and recommended price,
applicable to both negative and positive values."
    },
    "included_families": {
      "technical_path": "families_in_scope",
      "input_type": "list",
      "description": "List of the families included in the run (based on the product_family_h4 in our data) "
    },
    "all_validated_families": {
      "technical_path": "all_validated_families",
      "input_type": "list",
      "description": "List of all the families that have been validated and are available to be included in a
run (based on the product_family_h4 in our data)"
    },
    "hard_boundaries": {
      "technical_path": "hard_boundaries",
      "input_type": "dict",
      "description": "Dict of hard-boundaries"
    }
  }
```

These variables are used in the Parameters_dataset recipe.

Output filter dictionary

In this step, some CPCs can be filtered based on conditions passed by a dictionary variable, so that they do not appear in the front-end dataset displayed in the Qlik dashboard:

- The dictionary is structured as follows: for each family, we specify the feature to filter on, the type of operator and the value.
- The "type_filter" is used to specify whether the CPCs corresponding to the condition are retained or deleted.

```
"output_filters_dict": {
    "Amines": {
       "grp_of_activities": {
               "operator": "==",
               "value": "CSAGR",
               "type_filter": "keep"
       }
    }
}
```

The filter is now applied only to the Amines family, as indicated above, and only CPCs linked to the Agro market are retained in the output data.

ICM ratios dictionary

Variables to activate the use of ICM ratios.

- "use_ICM_features": boolean allowing to activate the ICM/Price ratio features.
- "interval_sizes": size of the interval of historical months to be aggregated when calculating ICM ratios (here we specify the current month and the last 3, 6, 9, 12 months).

```
"ICM_features": {
    "use_ICM_features": false,
    "interval_sizes": [1, 3, 6, 9, 12]
}
```