

Introduction

Ce notebook est le **quatrième** d'une série où je regarde des grands jeux de données, et dans chaque cas j'utilise un outil différent pour effectuer la même analyse sur le même jeu de données.

Cette fois-ci j'utilise **Talend Open Studio**, un outil ETL open source. On peut trouver chaque notebook dans la série dans mon [répertoire Github](#), y compris:

1. Pandas chunksize
2. Bibliothèque Dask
3. PySpark
4. Talend Open Studio

Il y a un peu plus d'explication dans le premier notebook (Pandas chunksize) par rapport à l'approche générale de l'analyse. Dans les autres notebooks je me concentre plus sur les éléments spécifiques à l'outil que j'utilise.

Description du jeu de données

On se servira du jeu de données des [Données de consommation d'énergie des résidences muni de SmartMeter à Londres](#), qui contient, selon le site web:

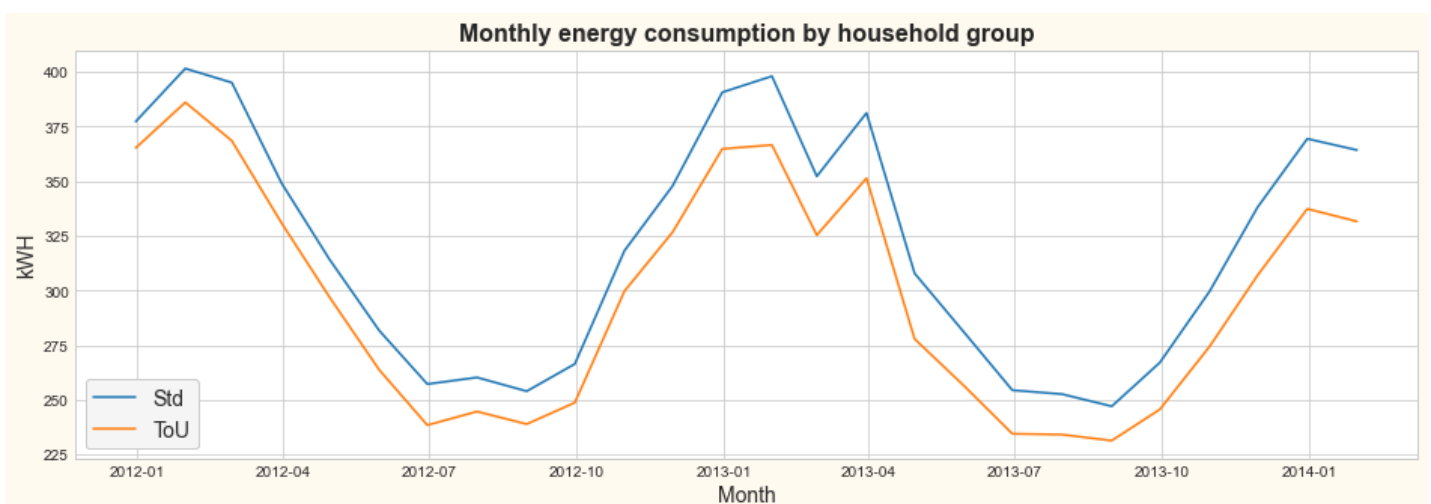
Des relevés de consommation d'énergie pour un échantillon de 5 567 résidences à Londres qui ont participé au projet de Low Carbon London (géré par UK Power Networks) entre novembre 2011 et février 2014.

Les résidences étaient divisées en deux groupes:

- Celles qui ont reçu des tarifs d'énergie Dynamic Time of Use (dTou) (décrit "Haut", "Moyen", ou "Bas") la veille du jour où le prix allait être appliqué.
- Celles qui étaient soumises au tarif Standard.

Un but du projet était d'évaluer si la connaissance du prix de l'énergie changerait le comportement par rapport à la consommation d'énergie.

Résultats



Les résultats montrent la variation saisonnière attendue et une différence nette entre les deux groupes, qui suggère qu'une connaissance du prix d'énergie aide à réduire la consommation de l'énergie.

Le reste du notebook montre comment le diagramme était produit des données brutes.

Introduction à Talend Open Studio

Talend Open Studio est un logiciel ETL (Extract-Transform-Load) gratuit. Voici comment Talend le décrit:

Grâce à Talend Open Studio, créez des pipelines de données simples en un rien de temps. Exécutez des tâches ETL et d'intégration de données simples, visualisez les données sous forme de graphiques et gérez des fichiers ; le tout dans un environnement open source installé on-premise et que vous contrôlez.

Bien qu'on effectuera à peu près les mêmes opérations qu'on a effectué dans les autres notebooks, cette fois-ci on utilisera le GUI Talend pour créer ces opérations, et où on aura besoin d'utiliser du code ce sera Java, et pas Python, puisque Open Studio est basé sur Java.

Je vais présumer une connaissance élémentaire de comment utiliser Talend Open Studio, donc ce notebook ne sera pas un guide détaillé.

NB Veuillez noter aussi que ce n'est qu'un exercice de formation pour moi - il se peut que Talend ne soit pas le meilleur outil pour quelques-unes des tâches présentées ici.

Installation

On peut télécharger le logiciel [ici](#). J'utilise une ancienne version (parce que pendant ma formation data science noter formateur l'a utilisée en disant qu'elle est fiable). Veuillez noter que vous aurez besoin de Java - v11 est nécessaire pour la dernière version d'Open Studio, mais v8 suffit pour les anciennes versions.

Fichiers Talend

Tous les fichiers de job que j'utilise sont disponibles dans ce répertoire Github dans un fichier zip dans le dossier `Talend`. Vous devriez pouvoir les importer vers votre projet Talend en tant qu'archive zip et ensuite les utiliser avec les données.

Pourtant, partout dans cet exercice vous verrez que j'utilise les chemins de fichier absolus. C'est parce qu'un chemin de fichier relatif pour Talend est relatif au dossier de l'installation et je préfère stocker les données ailleurs. Donc pour faire fonctionner les jobs Talend du répertoire Github vous devrez modifier les chemins de fichier.

Accéder les données

On peut télécharger les données sous forme de fichier zip qui contient un fichier csv de 167 million lignes. Si la commande `curl` ne fonctionne pas (il faudra un certains temps puisque c'est un fichier de 800MB), vous pouvez télécharger le fichier [ici](#) et le mettre dans le dossier `data` qui se trouve dans le dossier où ce notebook est sauvegardé.

```
In [ ]: !curl "https://data.london.gov.uk/download/smartmeter-energy-use-data-in-london-households/3527bf39-d93e-4071-8451-df2ade1ea4f2/LCL-FullData.zip" --location --create-dirs -o "data/LCL-FullData.zip"
```

Ensuite on décompresse les données. Il faudra peut-être un certain temps! Vous pouvez également le décompresser manuellement en utilisant un autre logiciel de décompression. Assurez-vous simplement que vous mettez le fichier décompressé dans un dossier qui s'appelle `data` dans le dossier où votre notebook est sauvegardé.

```
In [1]: !unzip "data/LCL-FullData.zip" -d "data"
```

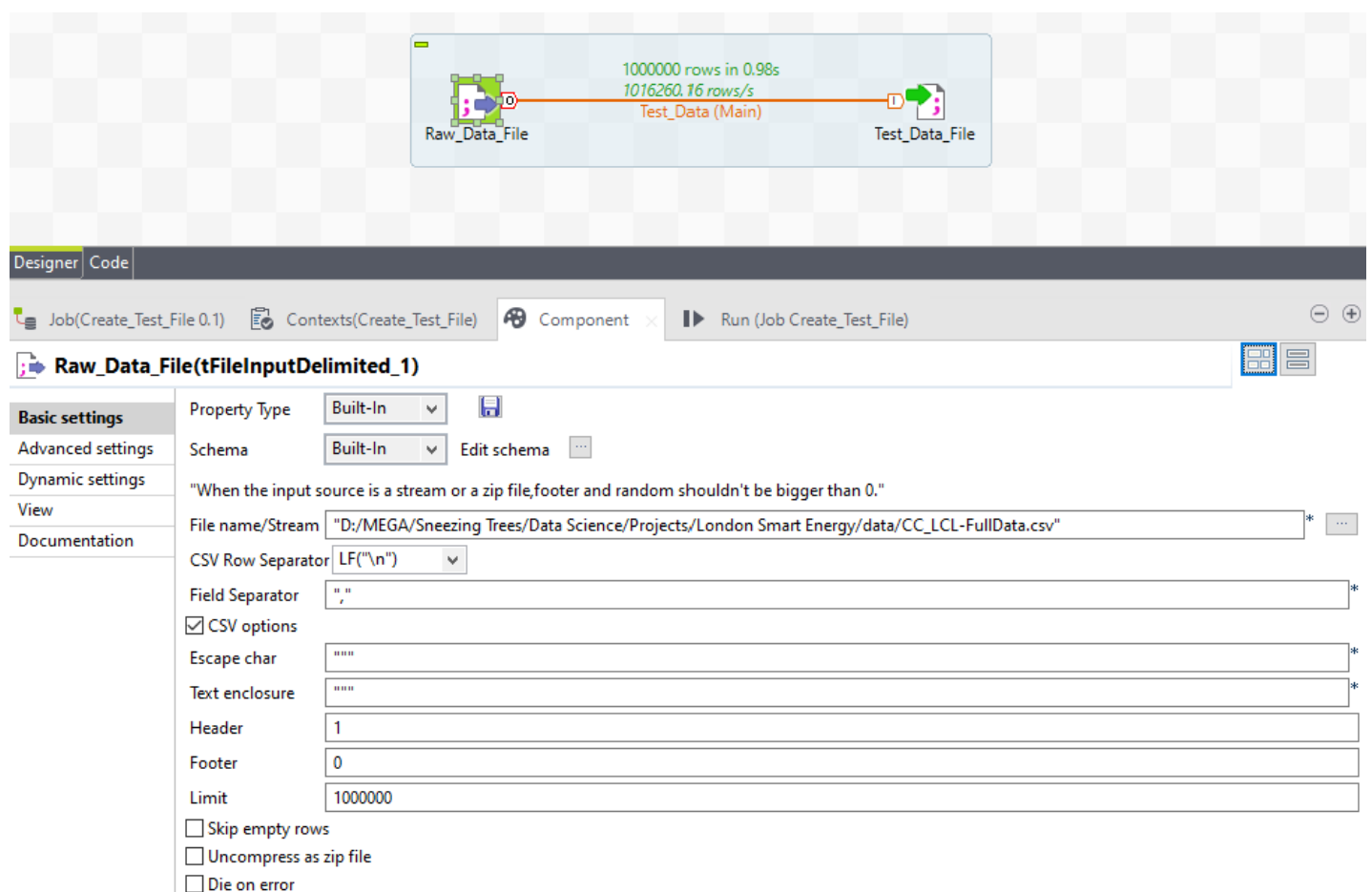
```
Archive: data/LCL-FullData.zip
  inflating: data/CC_LCL-FullData.csv
```

Examiner les données

D'abord on va créer un fichier test - un fichier qui ne contient qu'un petit sous-ensemble des données. Pour ce job "Job_Create_Test_File" on se sert d'un composant `tFileInputDelimited` et d'un composant `tFileOutputDelimited`. Le paramètre important au bas du capture d'écran pour le composant `tFileInputDelimited` - là on a défini une limite de 1 000 000 lignes.

Job_Create_Test_File

tFileInputDelimited - Raw_Data_File



The screenshot shows the Data Science Studio interface for a job named "Job_Create_Test_File". The job is running, and the progress bar indicates that 100,000 rows have been processed in 0.98 seconds at a rate of 1016260.16 rows/s. The job consists of two components: "Raw_Data_File" (tFileInputDelimited_1) and "Test_Data_File" (tFileOutputDelimited_1). The "Raw_Data_File" component is configured with a limit of 1,000,000 rows. The "Test_Data_File" component is configured with a limit of 1,000,000 rows. The job is running, and the progress bar shows 100,000 rows in 0.98s at 1016260.16 rows/s.

Raw_Data_File(tFileInputDelimited_1)

Property Type: Built-In

Schema: Built-In

"When the input source is a stream or a zip file, footer and random shouldn't be bigger than 0."

File name/Stream: "D:/MEGA/Sneezing Trees/Data Science/Projects/London Smart Energy/data/CC_LCL-FullData.csv"

CSV Row Separator: LF("\n")

Field Separator: " "

☒ CSV options

Escape char: ""

Text enclosure: ""

Header: 1

Footer: 0

Limit: 1000000

☐ Skip empty rows

☐ Uncompress as zip file

☐ Die on error

tFile_Output_Delimited - Test_Data_File

On sauvegarde notre fichier de sortie de 1 000 000 lignes comme "test-data.csv".

Job(Create_Test_File 0.1) Contexts(Create_Test_File) Component × Run (Job Create_Test_File)

Test_Data_File(tFileOutputDelimited_1)

Basic settings

Property Type: Built-In

☐ Use Output Stream

File Name: "D:/MEGA/Sneezing Trees/Data Science/Projects/London Smart Energy/data/test-data.csv" *

Row Separator: "\n"

Field Separator: ","

☐ Append

☒ Include Header

☐ Compress as zip file

Schema: Built-In Edit schema Sync columns

Et maintenant on peut examiner les données du fichier en utilisant Pandas.

In [2]: `import pandas as pd`

In [3]: `test_data = pd.read_csv("data/test-data.csv")`

In [4]: `test_data`

Out[4]:

	Household_ID	Tariff_Type	DateTime	kWh
0	MAC000002	Std	2012-10-12 00:30:00.0000000	0
1	MAC000002	Std	2012-10-12 01:00:00.0000000	0
2	MAC000002	Std	2012-10-12 01:30:00.0000000	0
3	MAC000002	Std	2012-10-12 02:00:00.0000000	0
4	MAC000002	Std	2012-10-12 02:30:00.0000000	0
...
999995	MAC000036	Std	2012-11-08 08:00:00.0000000	0.228
999996	MAC000036	Std	2012-11-08 08:30:00.0000000	0.042
999997	MAC000036	Std	2012-11-08 09:00:00.0000000	0.076
999998	MAC000036	Std	2012-11-08 09:30:00.0000000	0.07
999999	MAC000036	Std	2012-11-08 10:00:00.0000000	0.005

1000000 rows × 4 columns

Maintenant on vérifie si on a des doublons en utilisant un composant tUniqRow et un composant tLogRow. Les résultats montrent qu'il y a des doublons qu'il faudra supprimer.

Job Job_Check_For_Dupes

Execution

Run Kill Clear

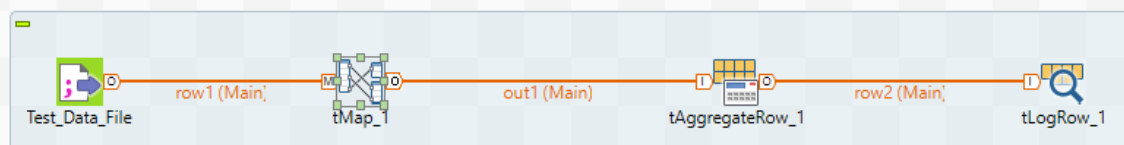
Starting job Job_Check_For_Dupes at 10:22 08/06/2022.

[statistics] connecting to socket on port 3924
[statistics] connected

tLogRow_1			
Household_ID	Tariff_Type	DateTime	kWh
MAC000002	Std	2012-10-20 00:00:00.0000000	0.2
MAC000002	Std	2012-11-20 00:00:00.0000000	0.258
MAC000002	Std	2012-12-21 00:00:00.0000000	0.238
MAC000002	Std	2013-01-21 00:00:00.0000000	0.21
MAC000002	Std	2013-02-21 00:00:00.0000000	0.216
MAC000002	Std	2013-03-24 00:00:00.0000000	0.486
MAC000002	Std	2013-04-24 00:00:00.0000000	0.147

☐ Line limit 100 ☐ Wrap

Vérifions aussi l'agrégation. On va transformer les données demi-horaire en journalières, donc pour cela on se servira d'un tMap et un tAggregateRow.



tMap Settings

Veuillez noter que:

- J'ai changé le type de kWh dans le composant tFileInputDelimited (Test_Data_File) de `String` en `Float` car on s'attend qu'ils soient des numéros décimaux.
- J'ai utilisé une expression pour convertir les valeurs DateTime en valeurs Date, mais pour des raisons de simplicité on va les laisser comme type `String`.

Find :

Var

Auto map!

row1

Column
Household_ID
Tariff_Type
DateTime
kWh

out1

Expression	Column
row1.Household_ID	Household_ID
row1.Tariff_Type	Tariff_Type
igHandling.LEFT(row1.DateTime,10)	Date
row1.kWh	kWh

Schema editor Expression editor

row1

Column	Key	Type	✓ N.	Date Pattern (...)	Length	Precisi...	Defa...	Com...
Household_ID	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		9	0		
Tariff_Type	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		3	0		
DateTime	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0			
kWh	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>		7	0		

out1

Column	Key	Type	✓ N.	Date Pattern (...)	Length	Precisi...	Defa...	Com...
Household_ID	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		9	0		
Tariff_Type	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		3	0		
Date	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		0			
kWh	<input type="checkbox"/>	Float	<input checked="" type="checkbox"/>		7	0		

Apply Ok Cancel

tAggregateRow settings

On groupe par Household_ID , Tariff_Type , and Date et on additionne les données de kWh.

tAggregateRow_1

Basic settings

Schema Built-In Edit schema Sync columns

Group by

Output column	Input column position
Household_ID	Household_ID
Tariff_Type	Tariff_Type
Date	Date

Operations

Output column	Function	Input column position	<input type="checkbox"/> Ignore null values
kWh	sum	kWh	<input type="checkbox"/>

Quand on exécute pourtant, le tLogRow affiche des erreurs - et aussi qu'une partie de l'agrégation a réussi.

Job Job_Check_Aggregation

Basic Run

Debug Run

Advanced settings

Target Exec

Memory Run

Execution

Run

Kill

Clear

Starting job Job_Check_Aggregation at 11:29 08/06/2022.

[statistics] connecting to socket on port 3907
[statistics] connected

Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:
Couldn't parse value for column kWh in row1, value is Null Details:

tLogRow_1

Household_ID	Tariff_Type	Date	kWh
MAC000025	Std	2013-05-18	5.058
MAC000035	Std	2013-04-18	15.161
MAC000025	Std	2013-05-19	5.301
MAC000035	Std	2013-04-19	15.654
MAC000025	Std	2013-05-14	5.533

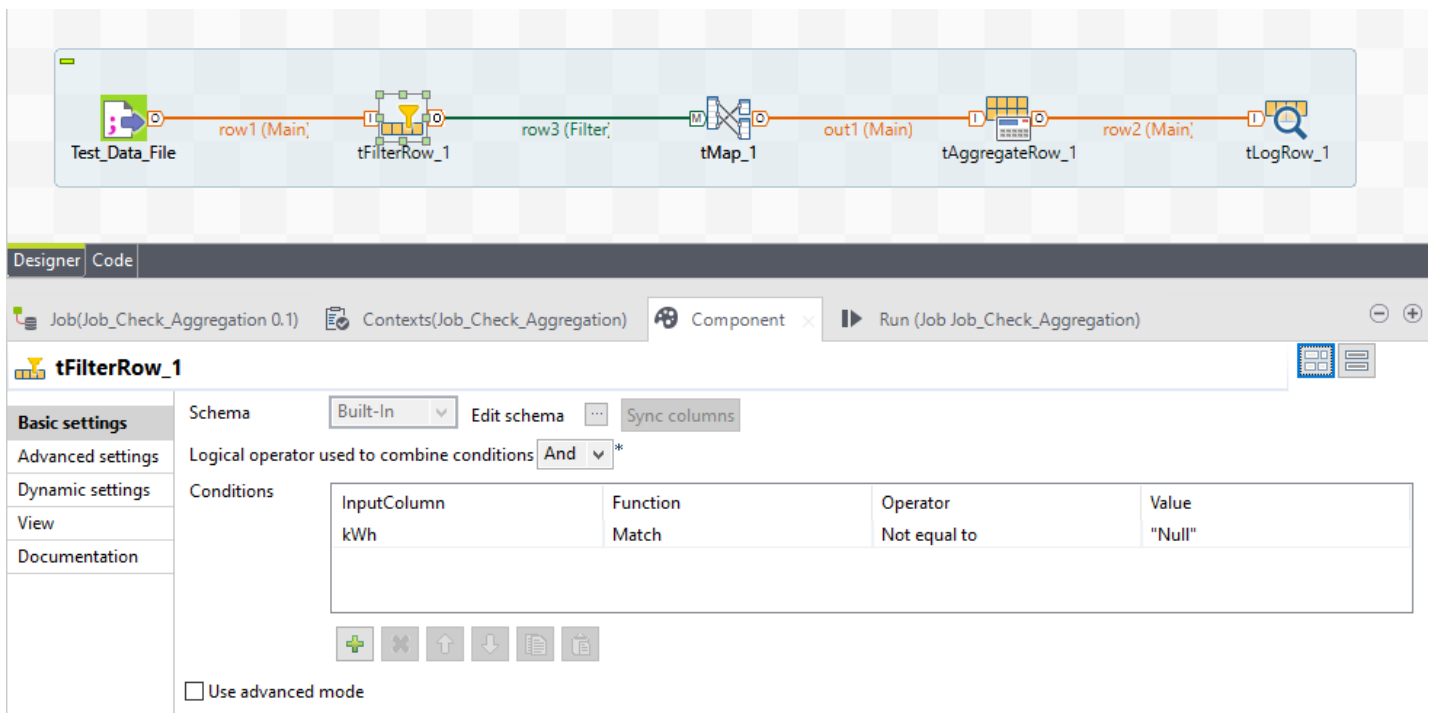
☒ Line limit

100000

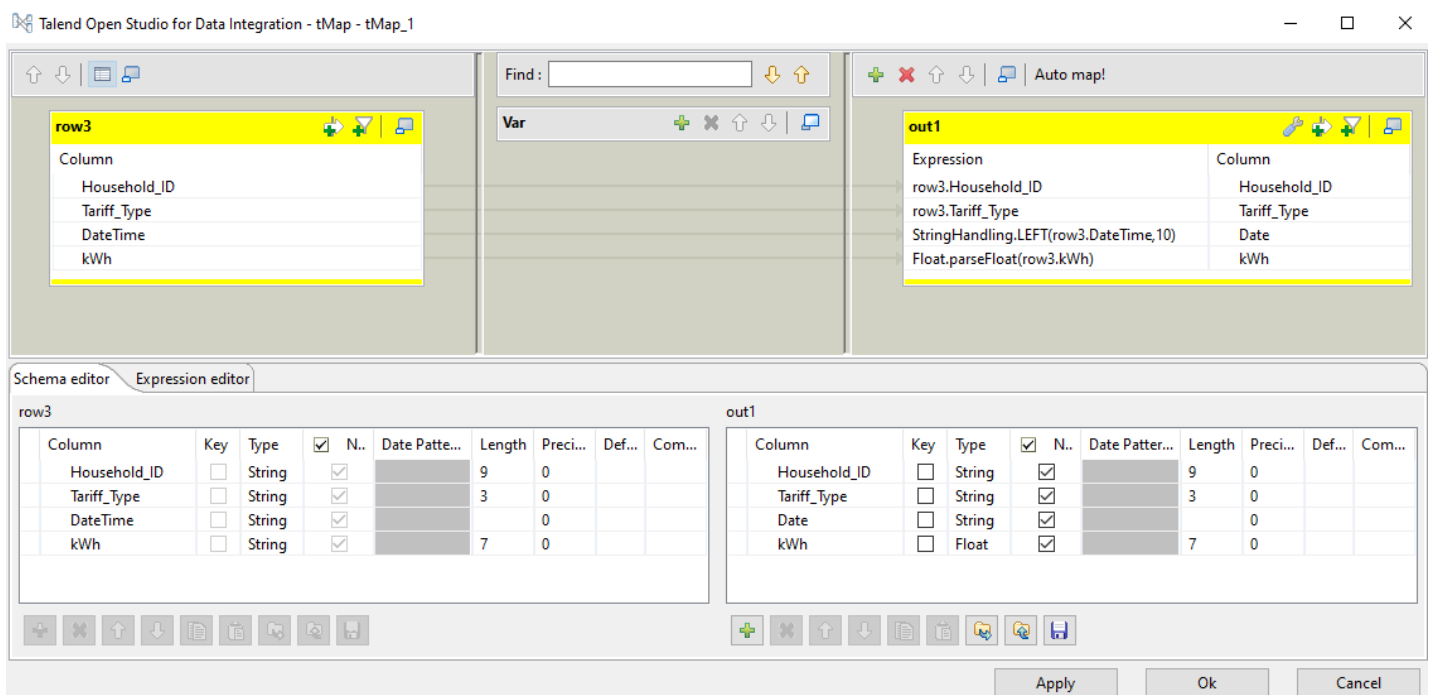
☐ Wrap

Le message d'erreur nous dit qu'il y a des valeurs "Null" que nous devons supprimer.

On les supprime en utilisant un composant `tFilterRow`. Veuillez noter pourtant que j'ai remis les données `kWh` en type `String` pour les composants `tFileInputDelimited` (`Test_Data_File`) et `tFilterRow`. Cela veut dire qu'on peut paramétrer le `tFilterRow` pour ne transmettre que les valeurs `kWh` qui ne sont pas égales à `"Null"`.



Le tMap reste inchangé sauf la transformation de kWh en type `Float` de type `String` en utilisant la fonction `Float.parseFloat()`.

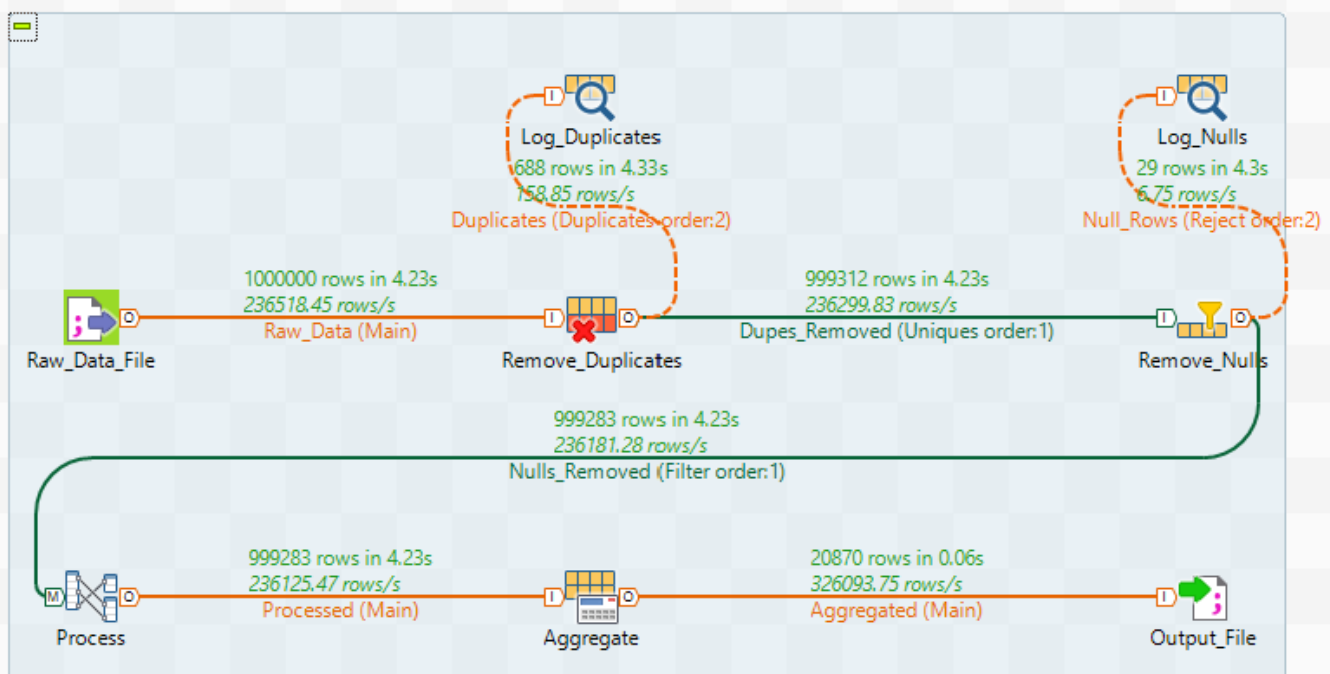


Maintenant quand on exécute on n'a pas d'erreur.

Agréger les données test

Le processus global pour les données test est affiché ci-dessous:

- Supprimer les doublons avec un tUniqRow.
- Supprimer les valeurs null avec un tFilterRow.
- Convertir les valeurs timestamp en date et les valeurs kWh en type `Float` de type `String` avec un tMap.
- Agréger avec un tAggregateRow.
- Sortir un fichier csv (test-out.csv).



On peut afficher les résultats en utilisant pandas.

```
In [5]: test_summary_data = pd.read_csv("data/test-out.csv")
```

```
In [6]: test_summary_data.sort_values(['Household_ID', 'Tariff_Type', 'Date'])
```

```
Out[6]:
```

	Household_ID	Tariff_Type	Date	kWh
14463	MAC000002	Std	2012-10-12	7.098
14464	MAC000002	Std	2012-10-13	11.087
14440	MAC000002	Std	2012-10-14	13.223
14443	MAC000002	Std	2012-10-15	10.257
14446	MAC000002	Std	2012-10-16	9.769
...
19879	MAC000036	Std	2012-11-04	2.401
19870	MAC000036	Std	2012-11-05	2.379
19873	MAC000036	Std	2012-11-06	2.352
19885	MAC000036	Std	2012-11-07	2.599
19888	MAC000036	Std	2012-11-08	0.689

20870 rows × 4 columns

Tout ce qui précède fonctionne bien pour les données test. Mais quand on l'applique aux données complètes on se trouve face à plusieurs erreurs de mémoire. Après avoir essayé plusieurs approches, j'ai conclu que:

1. La suppression des doublons est l'opération la plus exigeante en mémoire.
2. Même en utilisant des techniques telles que l'augmentation de mémoire accessible par le job (voir [ici](#)) et l'activation du paramètre Use of disk pour le composant tUniqRow (voir les paramètres avancés [ici](#)) la tâche a quand même échoué faute de mémoire.

3. La solution que j'ai trouvée - (et il est fort possible qu'il existe une meilleure) - était de diviser les fichiers originaux en plusieurs fichiers. Cependant, pour que la suppression de doublons fonctionne correctement, les divisions doivent être localisées pour qu'il n'y ait pas de doublons *entre* les fichiers, seulement *dedans* chaque fichier. Pour cette raison, on examinera la possibilité de diviser par Household ID ou par année-mois. L'une ou l'autre des approches garantit qu'il n'y aura pas de doublons entre les fichiers.

Les identifiants uniques de résidence

Voici le processus utilisé pour générer une liste d'identifiants uniques de résidence.

Deduplicate_Household_IDs(tUniqRow_1)

Schema: Built-In | Edit schema | Sync columns

Unique key: Household_ID

Key attribute: ☒ Case Sensitive: ☐

Schema of Deduplicate_Household_IDs

Column	Key	Type	✓	N..	Date Pa...	Len...	Pre...	D...	Co...
Household...	<input type="checkbox"/>	Stri...	<input checked="" type="checkbox"/>						
Tariff_Type	<input type="checkbox"/>	Stri...	<input checked="" type="checkbox"/>						
DateTime	<input type="checkbox"/>	Stri...	<input checked="" type="checkbox"/>						
kWh	<input type="checkbox"/>	Stri...	<input checked="" type="checkbox"/>						

Deduplicate_Household_IDs (Output)

Column	Key	Type	✓	N..	Date Pa...	Len...	Pre...	D...	Co...
Household...	<input type="checkbox"/>	Stri...	<input checked="" type="checkbox"/>						

Veuillez noter que le schéma du composant tUniqRow n'a que Household_ID comme données de sortie.

On peut afficher la liste comme dataframe Pandas.

```
In [7]: household_ids = pd.read_csv("data/unique-household-ids.csv", header=None, names=['Household IDs'])
```

```
In [8]: household_ids.sort_values(['Household IDs']).reset_index(drop=True)
```

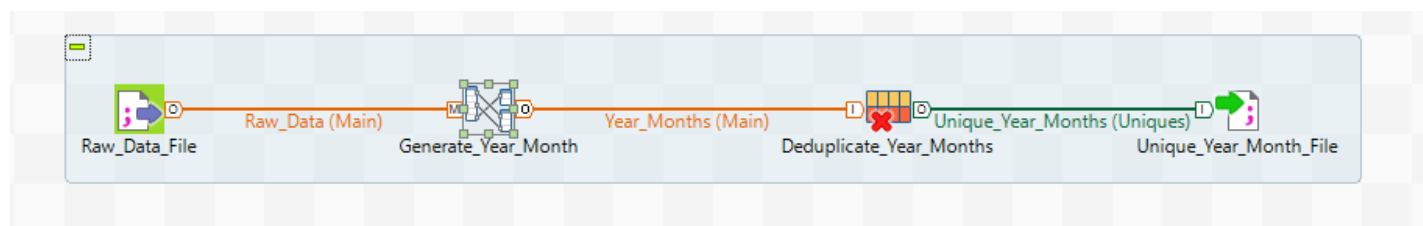
Out[8]:

Household IDs	
0	MAC000002
1	MAC000003
2	MAC000004
3	MAC000005
4	MAC000006
...	...
5561	MAC005563
5562	MAC005564
5563	MAC005565
5564	MAC005566
5565	MAC005567

5566 rows × 1 columns

Les années-mois uniques

Voici le processus utilisé pour générer une liste des années-mois uniques.



Dans le tMap on utilise une fonction StringHandling pour convertir le timestamp en format année-mois.

Talend Open Studio for Data Integration - tMap - tMap_1

Find :

Var

Auto map!

Raw_Data

Column
Household_ID
Tariff_Type
DateTime
kWh

Year_Months

Expression	Column
StringHandling.LEFT(Raw_Data.DateTime, 7)	Year_Month

Schema editor

Raw_Data

Column	Key	Type	<input checked="" type="checkbox"/> N..	Date Patte...	Leng...	Preci...	De...	Com...
Household_ID	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>					
Tariff_Type	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>					
DateTime	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>					
kWh	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>					

Year_Months

Column	Key	Type	<input checked="" type="checkbox"/> N..	Date Patte...	Leng...	Preci...	De...	Com...
Year_Month	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>					

Apply Ok Cancel

Le résultat montre qu'on a 28 mois uniques.

```
In [9]: year_months = pd.read_csv("data/unique-year-month.csv", header=None, names=['Year-Month'])
```

```
In [10]: year_months.sort_values(['Year-Month']).reset_index(drop=True)
```

```
Out[10]:
```

	Year-Month
--	------------

0	2011-11
1	2011-12
2	2012-01
3	2012-02
4	2012-03
5	2012-04
6	2012-05
7	2012-06
8	2012-07
9	2012-08
10	2012-09
11	2012-10
12	2012-11
13	2012-12
14	2013-01
15	2013-02
16	2013-03
17	2013-04
18	2013-05
19	2013-06
20	2013-07
21	2013-08
22	2013-09
23	2013-10
24	2013-11
25	2013-12
26	2014-01
27	2014-02

Diviser par année-mois semble être une approche raisonnable - facile à faire et l'approche donnera une quantité appropriée de fichiers.

Traiter le jeu de données entier

Voici les étapes qu'on utilise pour traiter toutes les données:

1. Générer une liste des année-mois uniques.

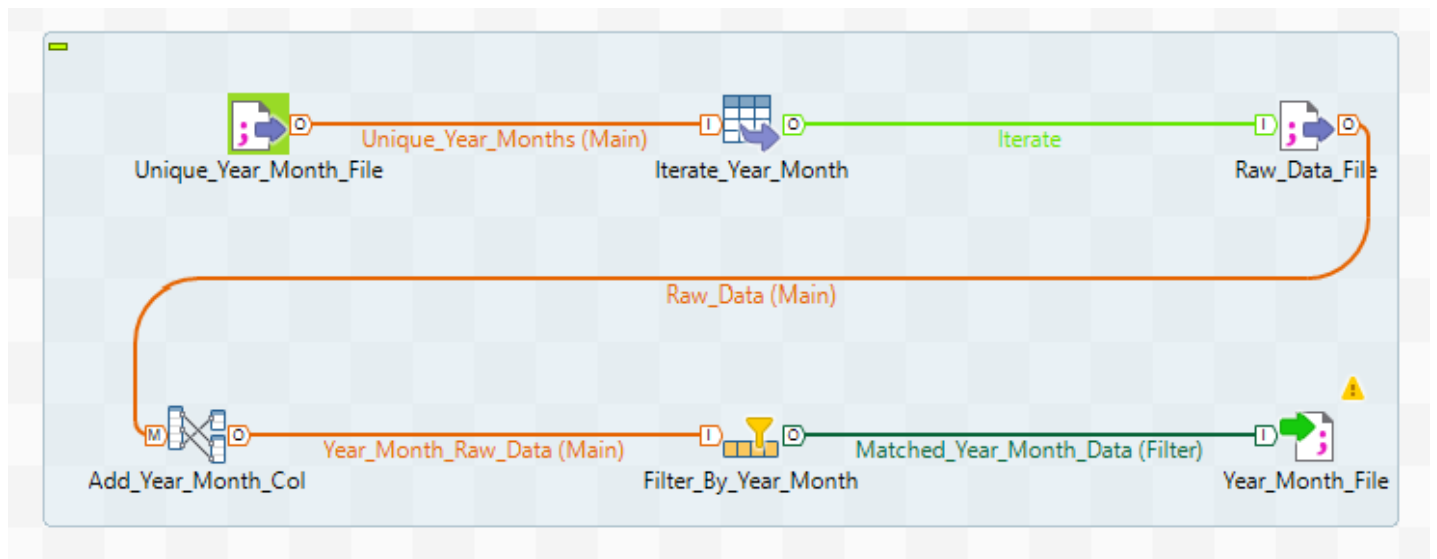
2. Utiliser la liste des année-mois pour itérer sur le jeu de données et créer un fichier pour chaque mois de données (28 en tout).
3. Supprimer les doublons dedans chacun des 28 fichiers.
4. Pour chaque fichier: Supprimer les valeurs null; Grouper par `Household_ID`, `Tariff_Type` et `Date` et additionner les valeurs `kWh`; Enregistrer les résultats agrégés dans un nouveau fichier.
5. Fusionner les 28 fichiers agrégés en un seul fichier.

Le processus global est très lent, surtout la division du fichier original, parce qu'on itère sur le jeu de données entier 28 fois. Il est fort probable qu'il y a des meilleures approches! Il faut environ une heure et demie pour exécuter le processus entier du début à la fin sur mon laptop.

1. Générer une liste des année-mois uniques

Fait ci-dessus.

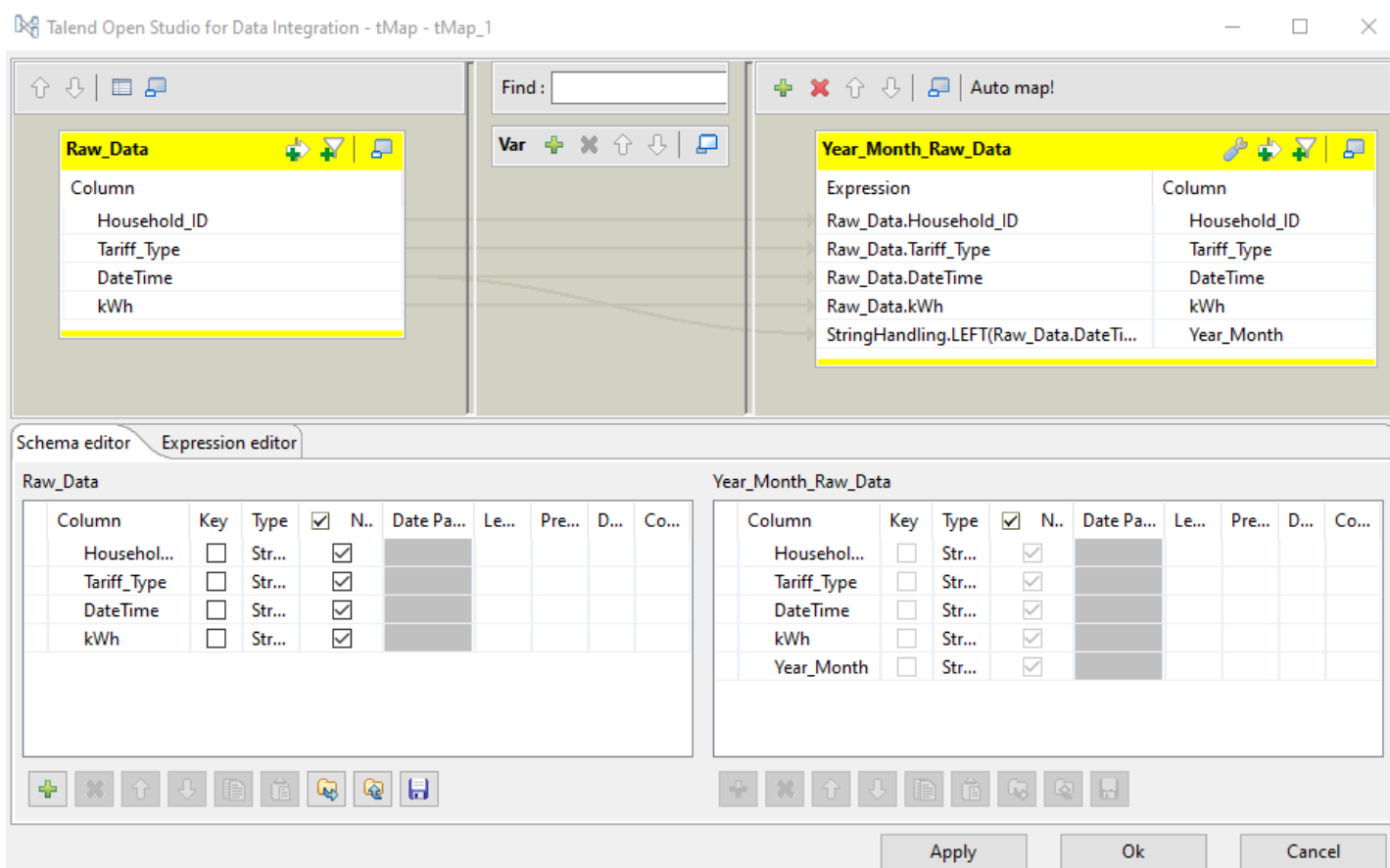
2. Diviser le fichier original en fichiers d'année-mois



On utilise un composant `tFlowToIterate` pour itérer sur le jeu de données 28 fois, en sélectionnant chaque fois les données qui correspondent au mois pertinent.

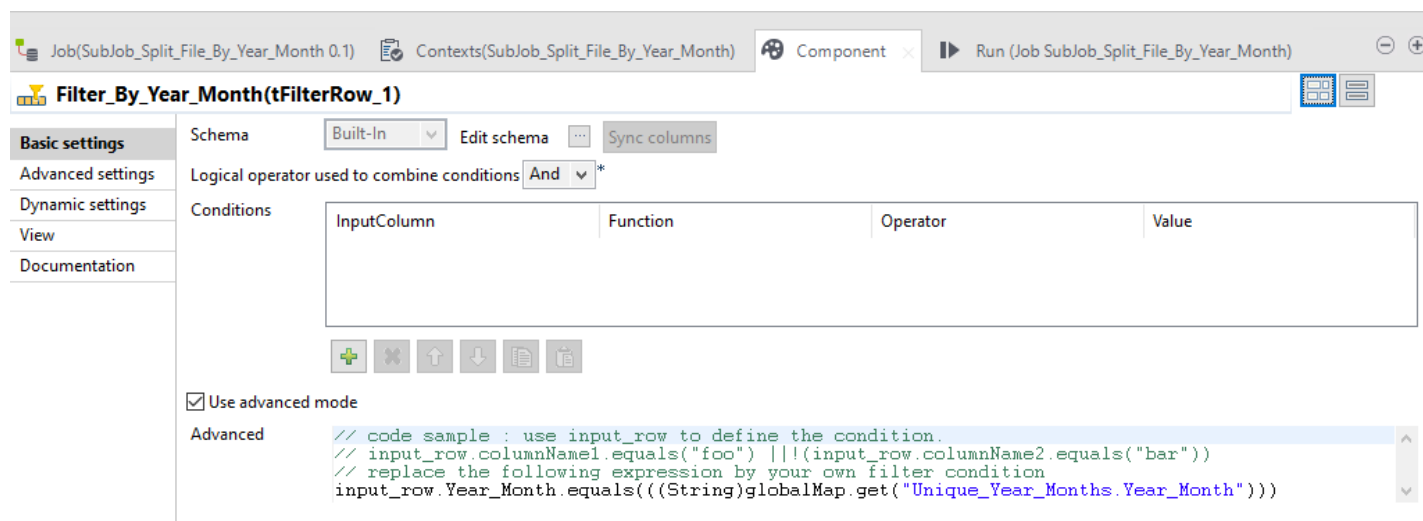
Paramètres `tMap` - `Add_Year_Month_Col`

On utilise le `tMap` pour ajouter la colonne `Year_Month` en utilisant la fonction `StringHandling`.



Paramètres tFilterRow - Filter_By_Year_Month

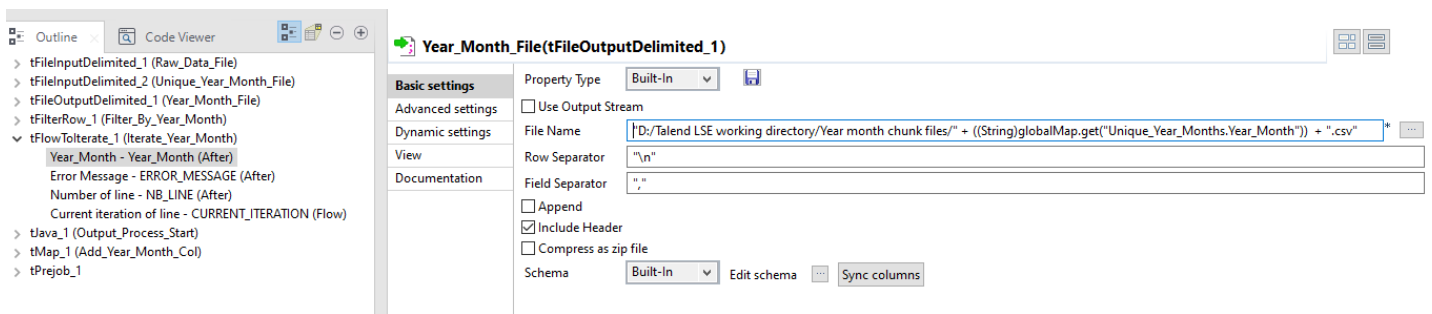
On utilise la mode avancée du composant tFilterRow pour filtrer par la valeur **Year_Month** qui a été définie par le composant tFlowTolterate.



Paramètres tFileOutputDelimited - Year_Month_File

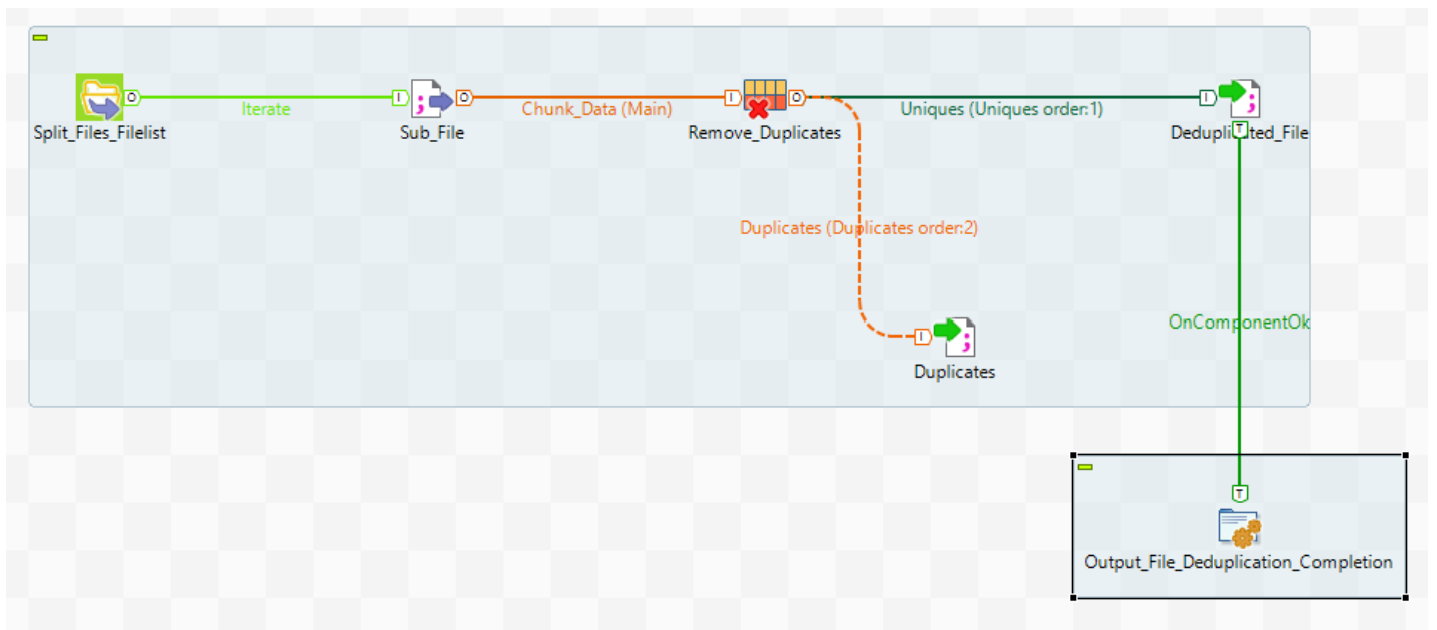
On utilise la valeur **Year_Month** stockée une deuxième fois pour définir le nom du fichier pour chacun des fichiers qui sont créés. Veuillez noter qu'on peut générer automatiquement le code

`((String)globalMap.get("Unique_Year_Months.Year_Month"))` en glissant l'élément **Year_Month - Year_Month (After)** du composant tFlowTolterate dans la fenêtre à gauche vers le champs File Name dans la fenêtre à droite.



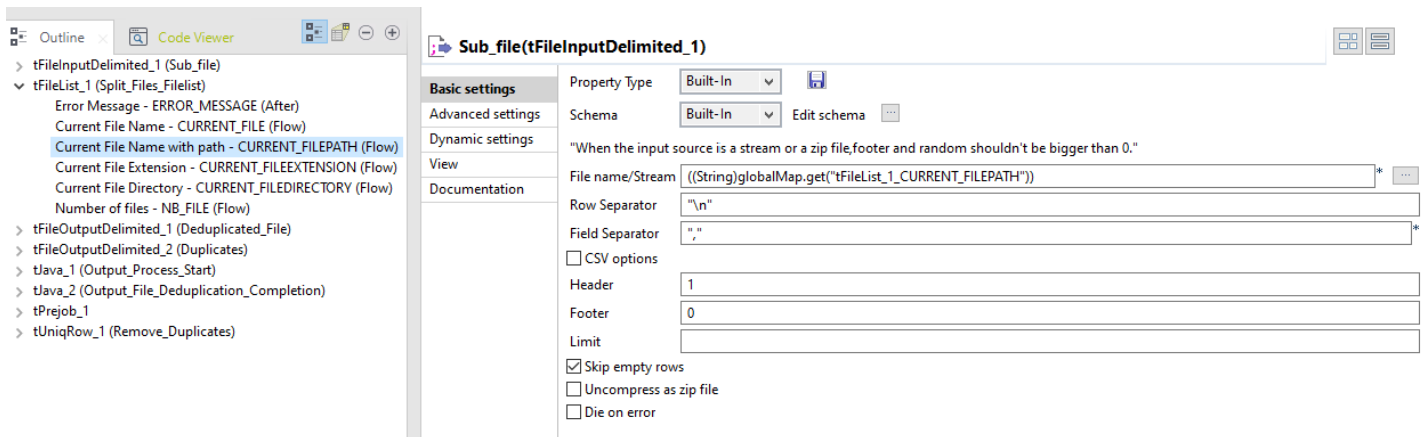
3. Supprimer les doublons dedans chacun des 28 fichiers année-mois

On utilise un composant tFileList pour itérer sur les 28 fichiers année-mois, supprimer les doublons de chaque fichier, et enregistrer les résultats dans un nouveau fichier.



Paramètres tFileInputDelimited - Sub_File

On peut glisser l'élément `Current File Name with path - CURRENT_FILEPATH (Flow)` de la fenêtre à gauche vers le champs File name/Stream dans la fenêtre à droite pour générer le code dans ce champs-là.



Paramètres tUniqRow - Remove_Duplicates (Basiques et Avancés)

Remove_Duplicates(tUniqRow_1)

Schema: Built-In Edit schema Sync columns

Unique key

Column	Key attribute	Case Sensitive
Household_ID	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Tariff_Type	<input checked="" type="checkbox"/>	<input type="checkbox"/>
DateTime	<input checked="" type="checkbox"/>	<input type="checkbox"/>
kWh	<input type="checkbox"/>	<input type="checkbox"/>

Veuillez noter qu'on utilise le paramètre Use of disk dans les paramètres avancés pour réduire la mémoire requise.

Remove_Duplicates(tUniqRow_1)

Basic settings

Advanced settings

Dynamic settings

View

Documentation

☐ Only once each duplicated key

☒ Use of disk (suitable for processing large row set)

Buffer size in memory: Medium(1million)*

Directory for temp files: D:/Talend LSE working directory/De-duplication temp directory

☐ Ignore trailing zeros for BigDecimal

☐ tStatCatcher Statistics

Paramètres tFileOutputDelimited - Deduplicated_File

On utilise encore une fois la technique de glisser un élément pour générer le nom du fichier sans les doublons.

Deduplicated_File(tFileOutputDelimited_1)

Basic settings

Advanced settings

Dynamic settings

View

Documentation

Property Type: Built-In

☐ Use Output Stream

File Name: D:/Talend LSE working directory/Deduplicated chunk files/' + ((String)globalMap.get("tFileList_1_CURRENT_FILE"))

Row Separator: \n

Field Separator: ,

☐ Append

☒ Include Header

☐ Compress as zip file

Schema: Built-In Edit schema Sync columns

Paramètres d'exécution du job

Pour ce job j'ai changé les paramètres d'exécution pour augmenter la mémoire disponible au Java Virtual Machine - en augmentant l'attribution initiale de mémoire de la valeur par défaut de 256MB -Xms256M jusqu'à 1GB -Xms1024M et l'attribution maximale de la valeur par défaut de 1GB -Xms1024M jusqu'à 4GB -Xms4096M. Peut-être qu'il aurait été possible d'éviter ceci en réduisant la taille du tampon dans le paramètre "Use of disk" du composant tUniqRow.

Basic Run

Debug Run

Advanced settings

Target Exec

Memory Run

☒ Statistics

☐ Exec time

☒ Save Job before execution

☒ Clear before run

JVM Setting

Job Run VM arguments

☒ Use specific JVM arguments

Argument	
-Xms1024M	
-Xmx4096M	

New...

Remove

Up

Down

```

graph LR
    Split_Files_Filelist[Split_Files_Filelist] --> Iterate[Iterate]
    Iterate --> Sub_file[Sub_file]
    Sub_file --> Raw_Data_Chunk_Main[Raw_Data_Chunk (Main)]
    Raw_Data_Chunk_Main --> Remove_Nulls[Remove_Nulls]
    Remove_Nulls --> Nulls_Removed_Filter[Nulls_Removed (Filter)]
    Remove_Nulls --> Preprocess[Preprocess]
    Preprocess --> Sorted_Main[Sorted (Main)]
    Sorted_Main --> Sort[Sort]
    Sort --> Aggregate_Sorted[Aggregate_Sorted]
    Aggregate_Sorted --> Aggregated_Main[Aggregated (Main)]
    Aggregated_Main --> Aggregated_Chunk[Aggregated_Chunk]
    Preprocess --> OnComponentOk[OnComponentOk]
    OnComponentOk --> Output_Process_Progress[Output_Process_Progress]
  
```

Paramètres tSortRow - Sort (Basiques et Avancés)

Sort(tSortRow_1)

Basic settings

Advanced settings

Dynamic settings

View

Documentation

Schema

Criteria

Built-In Edit schema Sync columns

Schema column	sort num or alpha?	Order asc or desc?
Household_ID	alpha	asc
Tariff_Type	alpha	asc
Date	alpha	asc

+

✕

↑

↓

📄

📅

+

Sort(tSortRow_1)

Basic settings

Advanced settings

Dynamic settings

View

Documentation

☒ Sort on disk
Temp data directory path "D:/Talend LSE working directory/Sorting temp directory" *
☒ Create temp data directory if does not exist
Buffer size of external sort 1000000 *
☐ tStatCatcher Statistics

Paramètres tAggregateSortedRow - Aggregate_Sorted

Il faut définir le nombre de lignes d'entrée pour le composant tAggregateSortedRow. On peut l'obtenir en glissant l'élément **Number of line matching the filter - NB_LINE_OK (After)** du composant tFilterRow dans la fenêtre à gauche vers le champs Input rows count dans la fenêtre à droite.

Outline

Code Viewer

Aggregate_Sorted(tAggregateSortedRow_1)

Basic settings

Advanced settings

Dynamic settings

View

Documentation

Schema

Built-In

Edit schema

Sync columns

(Enter the accurate number of input rows)

Input rows count ((Integer)globalMap.get("tFilterRow_1_NB_LINE_OK")) *

Group by

Output column

Household_ID

Tariff_Type

Date

Input column position

Household_ID

Tariff_Type

Date

Operations

Output column

kWh

Function

sum

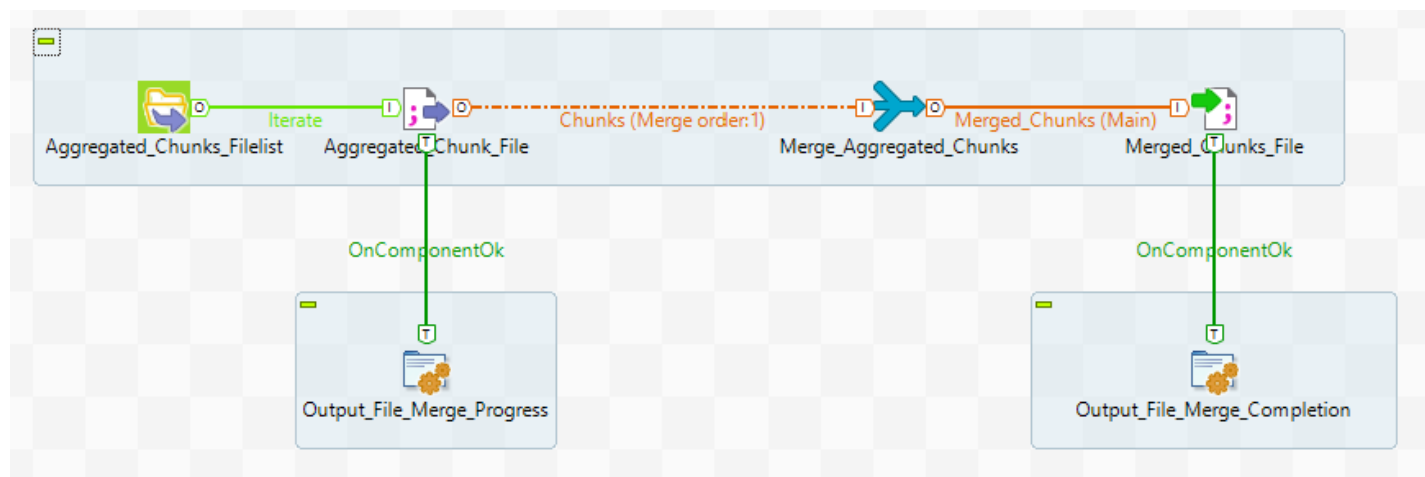
Input column position

kWh

Ignore null values

☐

5. Fusionner les 28 fichiers agrégés en un seul fichier.



Paramètres tFileInputDelimited - Aggrgated_Chunk_File

Outline

Code Viewer

Aggregated_Chunk_File(tFileInputDelimited_1)

Basic settings

Advanced settings

Dynamic settings

View

Documentation

Property Type

Built-In

Schema

Built-In

Edit schema

"When the input source is a stream or a zip file, footer and random shouldn't be bigger than 0."

File name/Stream ((String)globalMap.get("tFileList_1_CURRENT_FILEPATH")) *

Row Separator "\n"

Field Separator " "

☐ CSV options

Header 1

Footer 0

Limit

☒ Skip empty rows

☐ Uncompress as zip file

☐ Die on error

Voir les résultats

On peut voir les résultats dans un dataframe Pandas.

```
In [11]: daily_summary = (  
    pd.read_csv("data/merged-processed-chunks.csv").sort_values(  
        ['Household_ID', 'Tariff_Type', 'Date']  
    )  
    .reset_index(drop=True)  
    .rename(columns = {  
        'Household_ID' : 'Household ID',  
        'Tariff_Type' : 'Tariff Type'  
    })  
)
```

```
In [12]: daily_summary
```

```
Out[12]:
```

	Household ID	Tariff Type	Date	kWh
0	MAC000002	Std	2012-10-12	7.098
1	MAC000002	Std	2012-10-13	11.087
2	MAC000002	Std	2012-10-14	13.223
3	MAC000002	Std	2012-10-15	10.257
4	MAC000002	Std	2012-10-16	9.769
...
3510398	MAC005567	Std	2014-02-24	4.107
3510399	MAC005567	Std	2014-02-25	5.762
3510400	MAC005567	Std	2014-02-26	5.066
3510401	MAC005567	Std	2014-02-27	3.217
3510402	MAC005567	Std	2014-02-28	0.183

3510403 rows × 4 columns

Enregistrer les données agrégées

Maintenant qu'on a ramené les données à environ 3 millions lignes on devrait pouvoir les contenir dans un seul dataframe. Il vaut mieux les sauvegarder pour qu'on n'ait pas besoin de réexécuter l'agrégation chaque fois qu'on veut traiter les données.

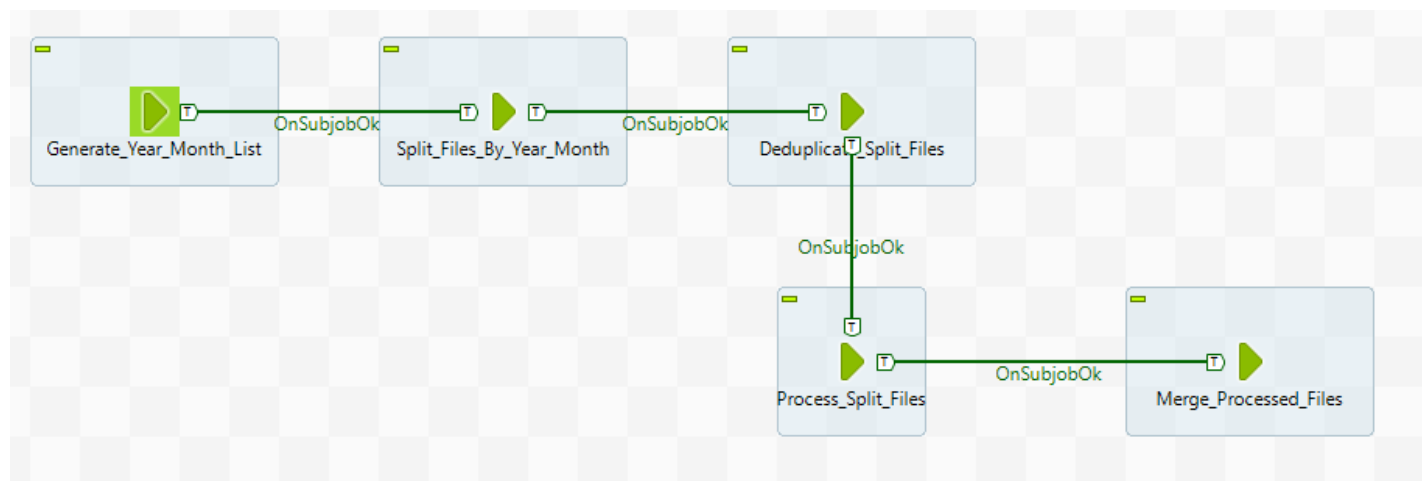
On va le sauvegarder comme fichier compressé gz - pandas reconnait automatiquement le type de fichier quand on précise l'extension.

```
In [13]: daily_summary.to_csv("data/daily-summary-data.gz", index=False)
```

Gestion du processus global

Pour exécuter tous les jobs d'un seul coup on crée un job Process_Coordinator qui déclenche les jobs séquentiellement. Il y a aussi plusieurs composants tJava dans les subjobs qui affiche le progrès dans le console

pendant l'exécution des jobs.



A partir d'ici, le reste de ce notebook contient à peu près le même traitement que tous les autres notebooks dans la série.

Analysing the data

```
In [14]: saved_daily_summary = pd.read_csv("data/daily-summary-data.gz")
```

```
In [15]: saved_daily_summary
```

Out[15]:

	Household ID	Tariff Type	Date	kWh
0	MAC000002	Std	2012-10-12	7.098
1	MAC000002	Std	2012-10-13	11.087
2	MAC000002	Std	2012-10-14	13.223
3	MAC000002	Std	2012-10-15	10.257
4	MAC000002	Std	2012-10-16	9.769
...
3510398	MAC005567	Std	2014-02-24	4.107
3510399	MAC005567	Std	2014-02-25	5.762
3510400	MAC005567	Std	2014-02-26	5.066
3510401	MAC005567	Std	2014-02-27	3.217
3510402	MAC005567	Std	2014-02-28	0.183

3510403 rows × 4 columns

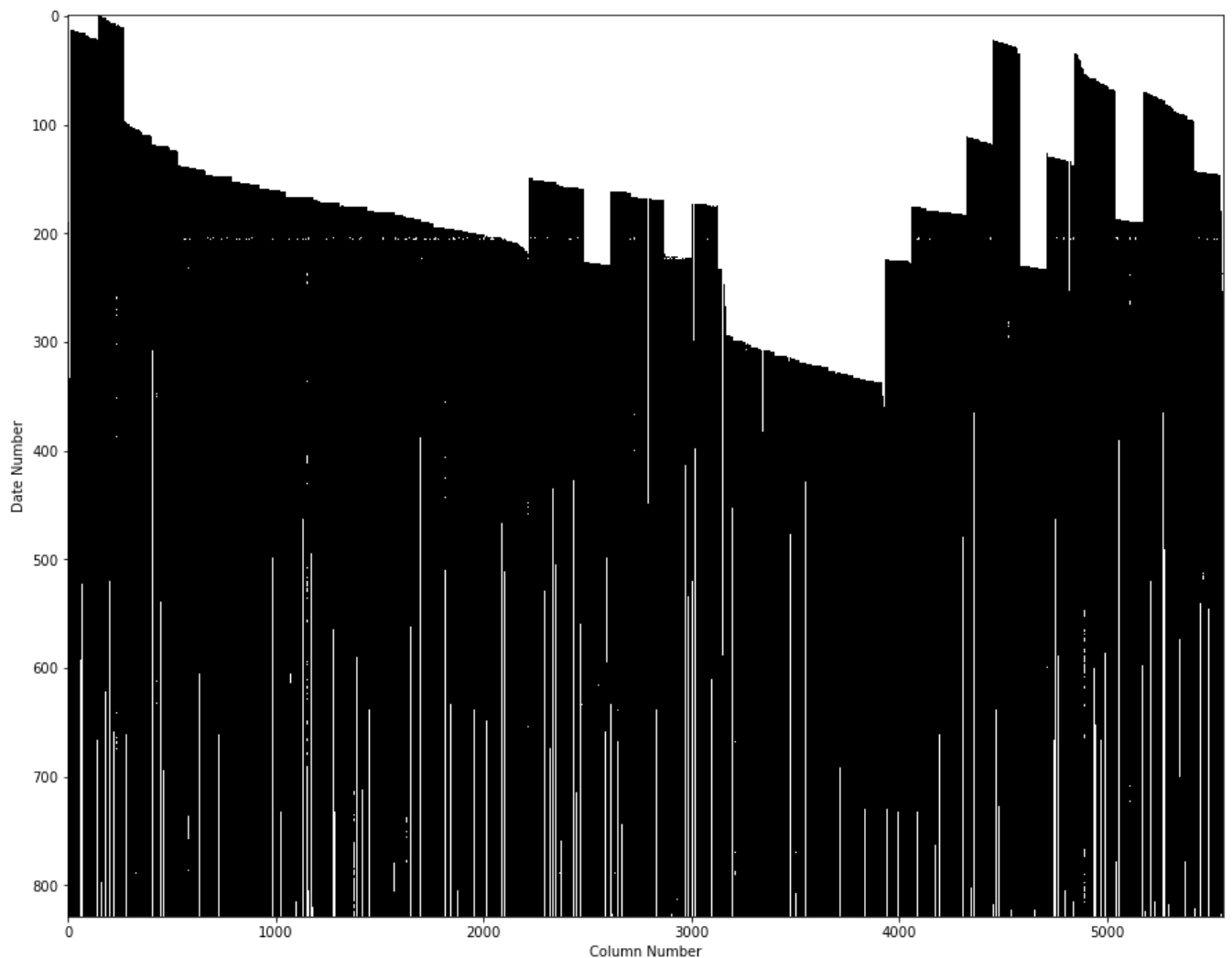
Par intérêt examinons la couverture des données. D'abord on réorganise pour avoir les résidences en colonne et les date en ligne.

```
In [16]: summary_table = saved_daily_summary.pivot_table(
    'kWh',
    index='Date',
    columns='Household ID',
    aggfunc='sum'
)
```

Ensuite on peut afficher où on a des données (noir) et où on n'en a pas (blanc).

```
In [17]: import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(15, 12))
plt.imshow(summary_table.isna(), aspect="auto", interpolation="nearest", cmap="gray")
plt.xlabel("Column Number")
plt.ylabel("Date Number");
```



Malgré une couverture un peu lacunaire, calculer par tarif sur toutes les résidences par jour devrait nous donner une comparaison utile.

```
In [18]: daily_mean_by_tariff_type = saved_daily_summary.pivot_table(
    'kWh',
    index='Date',
    columns='Tariff Type',
    aggfunc='mean'
)
daily_mean_by_tariff_type
```

Out[18]: **Tariff Type** **Std** **ToU**

Date		
2011-11-23	7.430000	4.327500
2011-11-24	8.998333	6.111750
2011-11-25	10.102885	6.886333
2011-11-26	10.706257	7.709500
2011-11-27	11.371486	7.813500
...
2014-02-24	10.580187	9.759439
2014-02-25	10.453365	9.683862
2014-02-26	10.329026	9.716652
2014-02-27	10.506416	9.776561
2014-02-28	0.218075	0.173949

829 rows × 2 columns

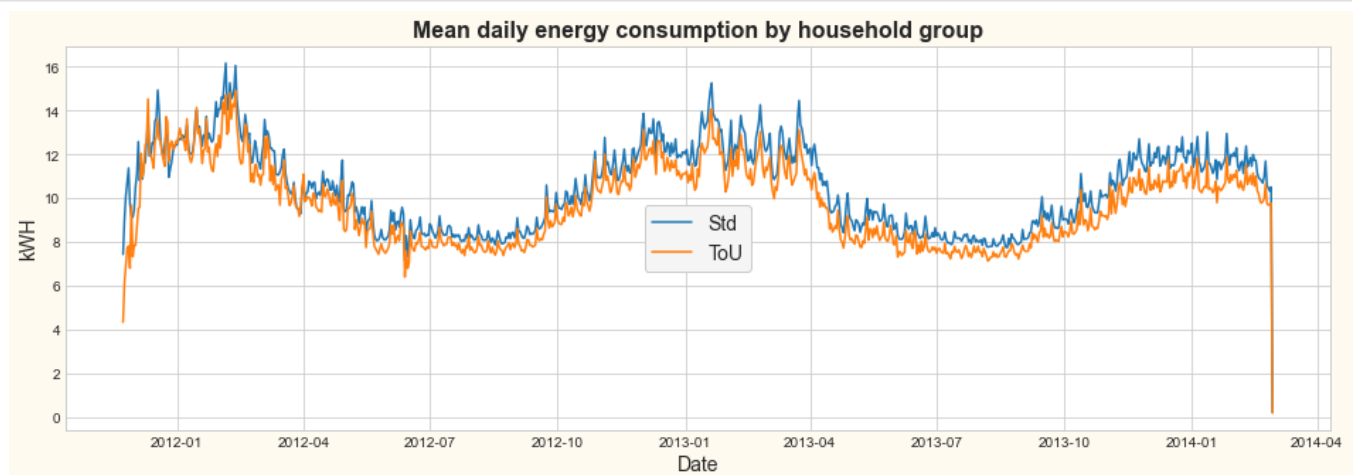
Finalement on peut tracer les deux groupes de données. Le traçage marche mieux si on convertit la date de type `string` en type `datetime`.

```
In [19]: daily_mean_by_tariff_type.index = pd.to_datetime(daily_mean_by_tariff_type.index)
```

```
In [20]: plt.style.use('seaborn-whitegrid')

plt.figure(figsize=(16, 5), facecolor='floralwhite')
for tariff in daily_mean_by_tariff_type.columns.to_list():
    plt.plot(
        daily_mean_by_tariff_type.index.values,
        daily_mean_by_tariff_type[tariff],
        label = tariff
    )

plt.legend(loc='center', frameon=True, facecolor='whitesmoke', framealpha=1, fontsize=14)
plt.title(
    'Mean daily energy consumption by household group',
    fontdict = {'fontsize' : 16, 'fontweight' : 'bold'}
)
plt.xlabel('Date', fontsize = 14)
plt.ylabel('kWh', fontsize = 14)
plt.show()
```



On dirait que la variation est saisonnière qui n'est pas étonnant vu la demande d'énergie de chauffage.

On dirait aussi qu'il y a une différence entre les deux groupes: le groupe ToU a l'air de consommer moins, mais l'affichage est trop granulaire pour voir bien. Agéons encore une fois, cette fois-ci par mois.

```
In [21]: daily_mean_by_tariff_type
```

```
Out[21]:
```

Tariff Type	Std	ToU
Date		
2011-11-23	7.430000	4.327500
2011-11-24	8.998333	6.111750
2011-11-25	10.102885	6.886333
2011-11-26	10.706257	7.709500
2011-11-27	11.371486	7.813500
...
2014-02-24	10.580187	9.759439
2014-02-25	10.453365	9.683862
2014-02-26	10.329026	9.716652
2014-02-27	10.506416	9.776561
2014-02-28	0.218075	0.173949

829 rows × 2 columns

On voit que les données commencent au cours de novembre 2011, donc on commencera le 1 décembre. On dirait que les données terminent parfaitement à la fin de février, mais la dernière valeur est suspecte puisqu'elle est très basse comparé aux autres. Il paraît probable que les données ont terminé au cours de la dernière journée, donc on finira à la fin de janvier. Peut-être qu'on a le même problème ailleurs dans les données, mais l'effet ne devrait pas être énorme parce que dans le pire des cas la consommation mensuelle d'une résidence sera réduite par deux journées (une au début et une à la fin).

```
In [22]: monthly_mean_by_tariff_type = daily_mean_by_tariff_type['2011-12-01' : '2014-01-31'].resample('M').sum()
monthly_mean_by_tariff_type
```

Out[22]:

Tariff Type	Std	ToU
-------------	-----	-----

Date		
2011-12-31	377.218580	365.145947
2012-01-31	401.511261	386.016403
2012-02-29	395.065321	368.475150
2012-03-31	349.153085	330.900633
2012-04-30	314.173857	296.903425
2012-05-31	281.666428	263.694338
2012-06-30	257.204029	238.417505
2012-07-31	260.231952	244.641359
2012-08-31	253.939017	238.904097
2012-09-30	266.392973	248.707929
2012-10-31	318.214026	299.714701
2012-11-30	347.818025	326.651435
2012-12-31	390.616106	364.754528
2013-01-31	398.004581	366.548143
2013-02-28	352.189818	325.298845
2013-03-31	381.191994	351.371278
2013-04-30	307.857771	277.856327
2013-05-31	280.762752	256.292247
2013-06-30	254.399013	234.481016
2013-07-31	252.609890	234.104814
2013-08-31	247.046087	231.347310
2013-09-30	267.024791	245.597424
2013-10-31	299.533302	274.332936
2013-11-30	338.082197	306.942424
2013-12-31	369.381371	337.331504
2014-01-31	364.225311	331.578243

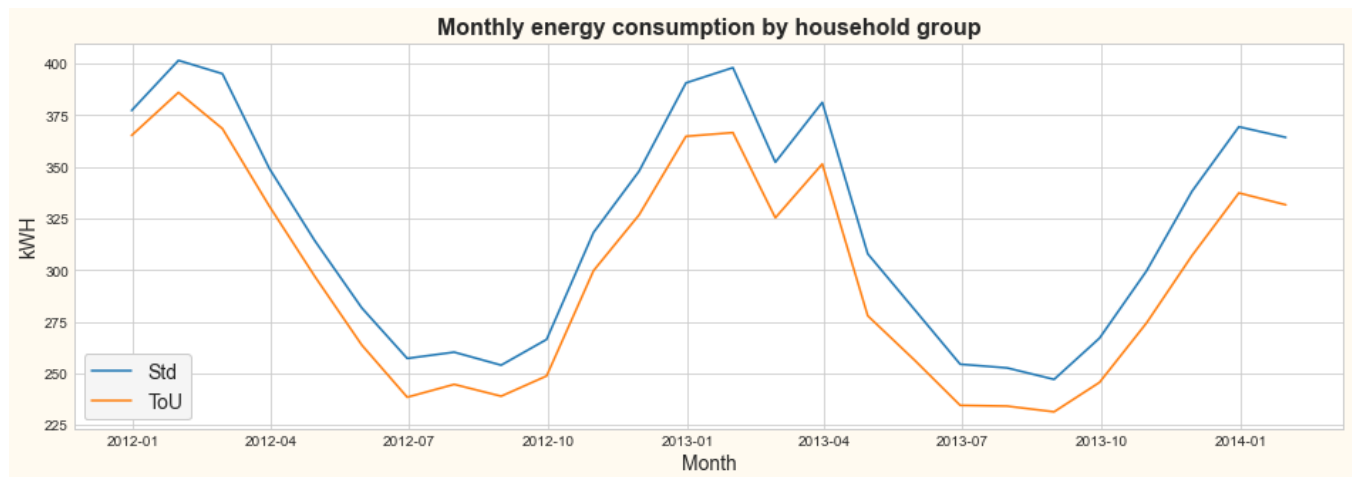
```
In [23]: plt.figure(figsize=(16, 5), facecolor='floralwhite')
for tariff in daily_mean_by_tariff_type.columns.to_list():
    plt.plot(
        monthly_mean_by_tariff_type.index.values,
        monthly_mean_by_tariff_type[tariff],
        label = tariff
    )

plt.legend(loc='lower left', frameon=True, facecolor='whitesmoke', framealpha=1, fontsize=14)
plt.title(
    'Monthly energy consumption by household group',
    fontdict = {'fontsize' : 16, 'fontweight' : 'bold'}
)
plt.xlabel('Month', fontsize = 14)
plt.ylabel('kWh', fontsize = 14)
```



```
# Uncomment for a copy to display in results
# plt.savefig(fname='images/result1-no-dupes.png', bbox_inches='tight')
```

```
plt.show()
```



Le diagramme est plus clair et il y a une différence évidente entre les deux groupes.

Veuillez noter que le diagramme ne montre pas la consommation mensuelle moyenne. Il montre la somme des moyennes journalières pour chaque mois. Pour calculer les vraies moyennes mensuelles on aurait besoin d'exclure les données journalières pour chaque résidence pendant les mois où les données n'étaient pas complètes. Notre méthode plus simple devrait nous donner une bonne approximation. The pattern is much clearer and there is an obvious difference between the two groups of consumers.