

# Introduction

Ce notebook est le **cinquième** 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 **PostgreSQL**, une base de données relationnelle 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
5. PostgreSQL

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:

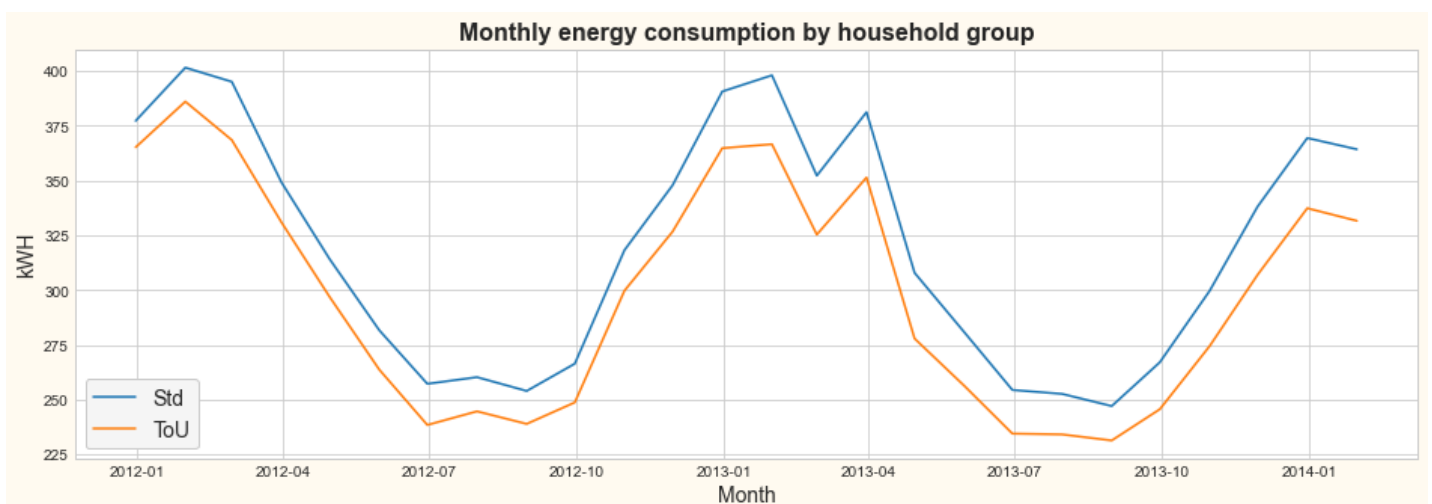
Des relevés de consommation d'énergie pour un échantillon de 5 567 résidences à Londres qui ont participé au projet de Low Carbon London (géré par UK Power Networks) entre novembre 2011 et février 2014.

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.

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

## Examiner les données

D'abord on se sert de Pandas pour créer un petit fichier test de 1 000 000 lignes.

```
In [1]: import pandas as pd
from IPython.display import HTML
```

```
In [2]: chunks = pd.read_csv('data/CC_LCL-FullData.csv', chunksize=1000000)
type(chunks)
```

```
Out[2]: pandas.io.parsers.readers.TextFileReader
```

```
In [3]: table_style = [{
    'selector' : 'caption',
    'props' : [
        ('font-size', '16px'),
        ('color', 'black'),
        ('font-weight', 'bold'),
        ('text-align', 'left')
    ]
}]

for chunk in chunks:

    display(
        chunk.describe(include='all')
        .style.set_caption('Describe')
        .set_table_styles(table_style)
    )

    display(
        chunk.head()
        .style.set_caption('Head')
```

```

.set_table_styles(table_style)
)

display(HTML('<br><span style="font-weight: bold; font-size: 16px">Info</span>'))
display(chunk.info())

test_data = chunk

break # Just the first chunk

```

## Describe

	LCLid	stdorToU	DateTime	KWH/hh (per half hour)
count	1000000	1000000	1000000	1000000
unique	30	1	39102	4801
top	MAC000018	Std	2012-11-20 00:00:00.0000000	0
freq	39082	1000000	58	45538

## Head

	LCLid	stdorToU	DateTime	KWH/hh (per half hour)
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

## Info

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 4 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   LCLid                                1000000 non-null object
1   stdorToU                            1000000 non-null object
2   DateTime                            1000000 non-null object
3   KWH/hh (per half hour)              1000000 non-null object
dtypes: object(4)
memory usage: 30.5+ MB
None

```

La colonne `KWH/hh (per half hour)` est de type `object` et pas de type `float` qui est étonnant, donc sans doute il y a des valeurs non-numériques qu'on devra traiter.

In [4]: `test_data`

Out[4]:

	LCLid	stdorToU	DateTime	KWH/hh (per half hour)
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

On enregistre notre fichier de test sous forme de fichier csv pour qu'on puisse le charger en PostgreSQL.

```
In [5]: test_data.to_csv("data/sql-test-data.csv", index=False)
```

## Charger les données

D'abord on crée une table en laquelle on va charger nos données. On ajoute une colonne `id` (qui va être utile plus tard). Veuillez noter que le type de la colonne `kWh_raw_data` est `TEXT` parce que nous soupçonnons qu'il y a des valeurs de type text qu'on devra traiter.

```
CREATE TABLE test_data
(
    id SERIAL PRIMARY KEY,
    Household_ID TEXT,
    Tariff_Type TEXT,
    Datetime TEXT,
    kWh_raw_data TEXT
);
```

Ensuite on utilise l'outil de chargement de pgAdmin pour charger les données du fichier test. On accède l'outil en faisant un clic droit sur la table qu'on veut remplir et en choisissant Import/Export Data...

Après avoir sélectionné l'onglet Import, il s'agit de préciser le fichier duquel on charge les données, le délimiteur, et le fait qu'on a des en-têtes.

Import/Export data - table 'test\_data'

OptionsColumns

Import/Export

✓ ImportExport

File Info

FilenameD:\MEGA\Sneezing Trees\Data Science\Projects\London Smart Ener

Formatcsv

EncodingSelect an item...

Miscellaneous

OID

Header

Delimiter,

Specifies the character that separates columns within each row (line) of the file. The default is a tab character in text format, a comma in CSV format. This must be a single one-byte character. This option is not allowed when using binary format.

i?

✕ Close

↺ Reset

✓ OK

Pour l'onglet de Columns, il faut just supprimer la colonne id (pusiqu'on ne charge pas les données pour celle-là). Ainsi il ne reste que les quatre colonnes de données qu'on veut charger.

Options Columns

Columns to import

household\_id x tariff\_type x datetime x kwh\_raw\_data x

An optional list of columns to be copied. If no column list is specified, all columns of the table will be copied.

NULL Strings

Specifies the string that represents a null value. The default is \N (backslash-N) in text format, and an unquoted empty string in CSV format. You might prefer an empty string even in text format for cases where you don't want to distinguish nulls from empty strings. This option is not allowed when using binary format.

Not null columns

Not null columns...

Do not match the specified column values against the null string. In the default case where the null string is empty, this means that empty values will be read as zero-length strings rather than nulls, even when they are not quoted. This option is allowed only in import, and only when using CSV format.

 Close Reset OK

On peut voir les données chargées en faisant un clic droit sur la table et en choisissant View/Edit Data. (J'ai choisi l'option de ne voir que les premières 100 lignes.)

	id [PK] integer	household_id text	tariff_type text	datetime text	kwh_raw_data text
1	1	MAC000002	Std	2012-10-12 00:30:00.0000000	0
2	2	MAC000002	Std	2012-10-12 01:00:00.0000000	0
3	3	MAC000002	Std	2012-10-12 01:30:00.0000000	0
4	4	MAC000002	Std	2012-10-12 02:00:00.0000000	0
5	5	MAC000002	Std	2012-10-12 02:30:00.0000000	0
6	6	MAC000002	Std	2012-10-12 03:00:00.0000000	0
7	7	MAC000002	Std	2012-10-12 03:30:00.0000000	0
8	8	MAC000002	Std	2012-10-12 04:00:00.0000000	0
9	9	MAC000002	Std	2012-10-12 04:30:00.0000000	0
10	10	MAC000002	Std	2012-10-12 05:00:00.0000000	0
11	11	MAC000002	Std	2012-10-12 05:30:00.0000000	0
12	12	MAC000002	Std	2012-10-12 06:00:00.0000000	0
13	13	MAC000002	Std	2012-10-12 06:30:00.0000000	0
14	14	MAC000002	Std	2012-10-12 07:00:00.0000000	0
15	15	MAC000002	Std	2012-10-12 07:30:00.0000000	0
16	16	MAC000002	Std	2012-10-12 08:00:00.0000000	0
17	17	MAC000002	Std	2012-10-12 08:30:00.0000000	0
18	18	MAC000002	Std	2012-10-12 09:00:00.0000000	0
19	19	MAC000002	Std	2012-10-12 09:30:00.0000000	0
20	20	MAC000002	Std	2012-10-12 10:00:00.0000000	0
21	21	MAC000002	Std	2012-10-12 10:30:00.0000000	0
22	22	MAC000002	Std	2012-10-12 11:30:00.0000000	0.143
23	23	MAC000002	Std	2012-10-12 12:00:00.0000000	0.663
24	24	MAC000002	Std	2012-10-12 12:30:00.0000000	0.256
25	25	MAC000002	Std	2012-10-12 13:00:00.0000000	0.155
26	26	MAC000002	Std	2012-10-12 13:30:00.0000000	0.199
27	27	MAC000002	Std	2012-10-12 14:00:00.0000000	0.125
28	28	MAC000002	Std	2012-10-12 14:30:00.0000000	0.165
29	29	MAC000002	Std	2012-10-12 15:00:00.0000000	0.14
30	30	MAC000002	Std	2012-10-12 15:30:00.0000000	0.110
Total rows: 100 of 100		Query complete 00:00:00.160			

On peut essayer de convertir nos données `kwh_raw_data` en type numerique dans une nouvelle colonne:

```
ALTER TABLE test_data ADD COLUMN kwh NUMERIC;
UPDATE test_data SET kwh = CAST(kwh_raw_data AS NUMERIC);
```







Mais la requête enchaîne une erreur:

```
ERROR: invalid input syntax for type numeric: "Null"
```

On dirait qu'on a des valeurs "Null" dans la colonne kwh\_raw\_data . Vérifions:

```
SELECT * FROM test_data WHERE kwh_raw_data = 'Null';
```

Effectivement, on voit qu'il y a 29 lignes avec une valeur "Null" dans notre jeu de données test.

Data output Messages Notifications						
     						
	id [PK] integer	household_id text	tariff_type text	datetime text	kwh_raw_data text	kwh numeric
1	3241	MAC000002	Std	2012-12-...	Null	[null]
2	38711	MAC000003	Std	2012-12-...	Null	[null]
3	70387	MAC000004	Std	2012-12-...	Null	[null]
4	106847	MAC000006	Std	2012-12-...	Null	[null]
5	131898	MAC000007	Std	2012-12-...	Null	[null]
6	183153	MAC000009	Std	2012-12-...	Null	[null]
7	163720	MAC000008	Std	2012-12-...	Null	[null]
8	208193	MAC000010	Std	2012-12-...	Null	[null]
9	618214	MAC000025	Std	2012-12-...	Null	[null]
10	231900	MAC000011	Std	2012-12-...	Null	[null]
11	256570	MAC000012	Std	2012-12-...	Null	[null]
12	344745	MAC000018	Std	2012-12-...	Null	[null]
13	422897	MAC000020	Std	2012-12-...	Null	[null]
14	461975	MAC000021	Std	2012-12-...	Null	[null]
Total rows: 29 of 29 Query complete 00:00:00.178						

On peut les supprimer très facilement:

```
DELETE FROM test_data WHERE kwh_raw_data = 'Null';
```

Et maintenant la conversion de nos données kwh devrait fonctionner:

```
UPDATE test_data SET kwh = CAST(kwh_raw_data AS NUMERIC);
```

Ça marche - mais c'est lent. 11 secondes pour les données test. On examinera cela quand on traitera le jeu de données entier.

Vérifions aussi si on a des doublons. C'est ici que la colonne id que nous avons créée est utile.

```
SELECT * FROM test_data
WHERE id IN (
  SELECT id
  FROM (
    SELECT id,
    ROW_NUMBER() OVER(
      PARTITION BY Household_ID, Tariff_Type, Datetime
      ORDER BY id
    ) AS row_num
  FROM test_data
```



```

) t
WHERE t.row_num > 1
);

```

On trouve 688 doublons dans notre jeu de données test.

Data output Messages Notifications

	id [PK] integer	household_id text	tariff_type text	datetime text	kwh_raw_data text	kwh numeric
1	7780	MAC000002	Std	2013-03-24 00:00:00.0000000	0.486	0.486
2	44738	MAC000003	Std	2013-04-24 00:00:00.0000000	1.424	1.424
3	65991	MAC000004	Std	2012-09-19 00:00:00.0000000	0	0
4	134947	MAC000007	Std	2013-02-21 00:00:00.0000000	0.179	0.179
5	160815	MAC000008	Std	2012-10-20 00:00:00.0000000	0.267	0.267
6	172726	MAC000008	Std	2013-06-25 00:00:00.0000000	0.281	0.281
7	180246	MAC000009	Std	2012-10-20 00:00:00.0000000	0.041	0.041
8	186203	MAC000009	Std	2013-02-21 00:00:00.0000000	0.098	0.098
9	189181	MAC000009	Std	2013-04-24 00:00:00.0000000	0.051	0.051
10	190670	MAC000009	Std	2013-05-25 00:00:00.0000000	0.112	0.112
11	204062	MAC000009	Std	2014-02-28 00:00:00.0000000	0.05	0.05
12	267064	MAC000012	Std	2013-07-26 00:00:00.0000000	0.05	0.05
13	298166	MAC000013	Std	2013-08-26 00:00:00.0000000	0.063	0.063
14	367551	MAC000019	Std	2012-01-15 00:00:00.0000000	0.063	0.063
15	376485	MAC000019	Std	2012-07-19 00:00:00.0000000	0.046	0.046
16	411099	MAC000020	Std	2012-04-17 00:00:00.0000000	0.056	0.056
17	448689	MAC000021	Std	2012-03-17 00:00:00.0000000	0.47	0.47
18	457623	MAC000021	Std	2012-09-19 00:00:00.0000000	0.314	0.314
19	478468	MAC000021	Std	2013-11-27 00:00:00.0000000	0.454	0.454
Total rows: 688 of 688		Query complete 00:00:04.056				

Et on peut les supprimer facilement aussi en remplaçant `SELECT *` par `DELETE` :

```

DELETE FROM test_data
WHERE id IN (
  SELECT id
  FROM (
    SELECT id,
    ROW_NUMBER() OVER(
      PARTITION BY Household_ID, Tariff_Type, Datetime
      ORDER BY id
    ) AS row_num
    FROM test_data
  ) t
  WHERE t.row_num > 1
);

```

## Agréger les données test

Le but est de **réduire** les données en les agréant d'une manière ou d'une autre. Puisque nous savons que les données sont organisées par demi-heure, on va les agréger par jour en les additionnant sur chaque période de 24 heures. Cela devrait réduire le nombre de lignes par un facteur d' environ 48.

D'abord il faut créer une colonne `Date` .

```
ALTER TABLE test_data ADD COLUMN Date TEXT;  
UPDATE test_data SET Date = LEFT(Datetime, 10);
```

```
SELECT * FROM test_data;
```

Data output

Messages

Notifications

≡+

📄

▼

📋

🗑️

🗄️

⬇️

	id [PK] integer	household_id text	tariff_type text	datetime text	kwh_raw_data text	kwh numeric	date text
1	5945	MAC000002	Std	2013-02-13 19:30:00.0000000	0.256	0.256	2013-02-13
2	5946	MAC000002	Std	2013-02-13 20:00:00.0000000	0.272	0.272	2013-02-13
3	5947	MAC000002	Std	2013-02-13 20:30:00.0000000	0.838	0.838	2013-02-13
4	5948	MAC000002	Std	2013-02-13 21:00:00.0000000	0.248	0.248	2013-02-13
5	5949	MAC000002	Std	2013-02-13 21:30:00.0000000	0.214	0.214	2013-02-13
6	5950	MAC000002	Std	2013-02-13 22:00:00.0000000	0.275	0.275	2013-02-13
7	5951	MAC000002	Std	2013-02-13 22:30:00.0000000	0.247	0.247	2013-02-13
8	5952	MAC000002	Std	2013-02-13 23:00:00.0000000	0.258	0.258	2013-02-13
9	5953	MAC000002	Std	2013-02-13 23:30:00.0000000	0.258	0.258	2013-02-13
10	5954	MAC000002	Std	2013-02-14 00:00:00.0000000	0.212	0.212	2013-02-14
11	5955	MAC000002	Std	2013-02-14 00:30:00.0000000	0.252	0.252	2013-02-14
12	5956	MAC000002	Std	2013-02-14 01:00:00.0000000	0.225	0.225	2013-02-14
13	5957	MAC000002	Std	2013-02-14 01:30:00.0000000	0.255	0.255	2013-02-14
14	5958	MAC000002	Std	2013-02-14 02:00:00.0000000	0.234	0.234	2013-02-14

Total rows: 1000 of 999971

Query complete 00:00:01.086

Et maintenant on peut agréger:

```
SELECT Household_ID, Tariff_Type, Date, SUM(kwh)  
FROM lse_test_data  
GROUP BY Household_ID, Tariff_Type, Date;
```

	household_id text	tariff_type text	date text	sum numeric
1	MAC000002	Std	2012-10-12	7.098
2	MAC000002	Std	2012-10-13	11.087
3	MAC000002	Std	2012-10-14	13.223
4	MAC000002	Std	2012-10-15	10.257
5	MAC000002	Std	2012-10-16	9.769
6	MAC000002	Std	2012-10-17	10.885
7	MAC000002	Std	2012-10-18	10.751
8	MAC000002	Std	2012-10-19	8.431
9	MAC000002	Std	2012-10-20	17.5779999
10	MAC000002	Std	2012-10-21	24.4900001
11	MAC000002	Std	2012-10-22	18.885
12	MAC000002	Std	2012-10-23	10.485
13	MAC000002	Std	2012-10-24	15.5370001
14	MAC000002	Std	2012-10-25	13.128
15	MAC000002	Std	2012-10-26	15.065
16	MAC000002	Std	2012-10-27	16.886
17	MAC000002	Std	2012-10-28	19.6289999
18	MAC000002	Std	2012-10-29	12.779
19	MAC000002	Std	2012-10-30	12.061
Total rows: 1000 of 20870			Query complete 00:00:00.428	

## Traiter le jeu de données entier

On utilisera les mêmes principes mais avec une petite modification.

### Chargement

D'abord on crée une nouvelle table.

```
CREATE TABLE full_data
(
  id SERIAL PRIMARY KEY,
  Household_ID TEXT,
  Tariff_Type TEXT,
  Datetime TEXT,
  kWh_raw_data TEXT
);
```

Et ensuite on charge les données de même manière qu'on a fait pour les données test. Il faut un moment: 7 - 10 minutes avec mon laptop en fonction des applis qui sont actifs en même temps.

### Suppression des doublons

On commence par la suppression des doublons. Il faut 25 - 30 minutes avec mon laptop.

```
DELETE FROM full_data
WHERE id IN (
  SELECT id
  FROM (
    SELECT id,
    ROW_NUMBER() OVER(
      PARTITION BY Household_ID, Tariff_Type, Datetime
      ORDER BY id
    ) AS row_num
    FROM full_data
  ) t
  WHERE t.row_num > 1
);
```

## Conversion

Maintenant on convertira les données `kwh` en type numérique. Mais au lieu de supprimer les lignes avec des valeurs `"Null"` en utilisant `UPDATE` on va créer une autre table avec une requête `SELECT` parce que c'est une méthode plus rapide. Et il vaut mieux créer la colonne `Date` en même temps. Il faut 8 - 10 minutes pour tout cela avec mon laptop.

```
CREATE TABLE lse_data AS
SELECT
  CAST(Household_ID AS TEXT) Household_ID,
  CAST(Tariff_Type AS TEXT) Tariff_Type,
  CAST(LEFT(Datetime, 10) AS TEXT) Date,
  CAST(kWh_raw_data AS NUMERIC) kWh
FROM full_data
WHERE kWh_raw_data != 'Null';
```

Ensuite on peut supprimer la table originale pour libérer l'espace sur le disque dur.

## Aggregation

Enfin on peut agréger. Et encore une fois on va créer une table, cette fois-ci pour qu'on n'ait pas besoin de réexécuter la requête si on a besoin de accéder les résultats.

```
CREATE TABLE lse_agg_data AS
SELECT Household_ID, Tariff_Type, Date, SUM(kwh) kWh
FROM lse_data
GROUP BY Household_ID, Tariff_Type, Date;
```

La dernière étape consiste en enregistrer les résultats sous forme de csv en utilisant l'outil de exportation de pgAdmin (en faisant un clic droit sur la table et en sélectionnant Import/Export Data...)

Je les ai enregistrés dans un fichier "daily-summary-data-postgresql.csv" dans le dossier "data".

## Voir les résultats

On peut voir les résultats dans un dataframe Pandas.

```
In [6]: daily_summary = (  
    pd.read_csv("data/daily-summary-data-postgresql.csv")  
)
```

```
In [7]: daily_summary
```

```
Out[7]:
```

	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

```
In [8]: daily_summary = (  
    pd.read_csv("data/daily-summary-data-postgresql.csv").sort_values(  
        ['household_id', 'tariff_type', 'date']  
    )  
    .rename(columns = {  
        'household_id' : 'Household ID',  
        'tariff_type' : 'Tariff Type',  
        'date' : 'Date',  
        'kwh' : 'kWh'  
    })  
)
```

```
In [9]: daily_summary
```

Out[9]:

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

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

```
In [12]: saved_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

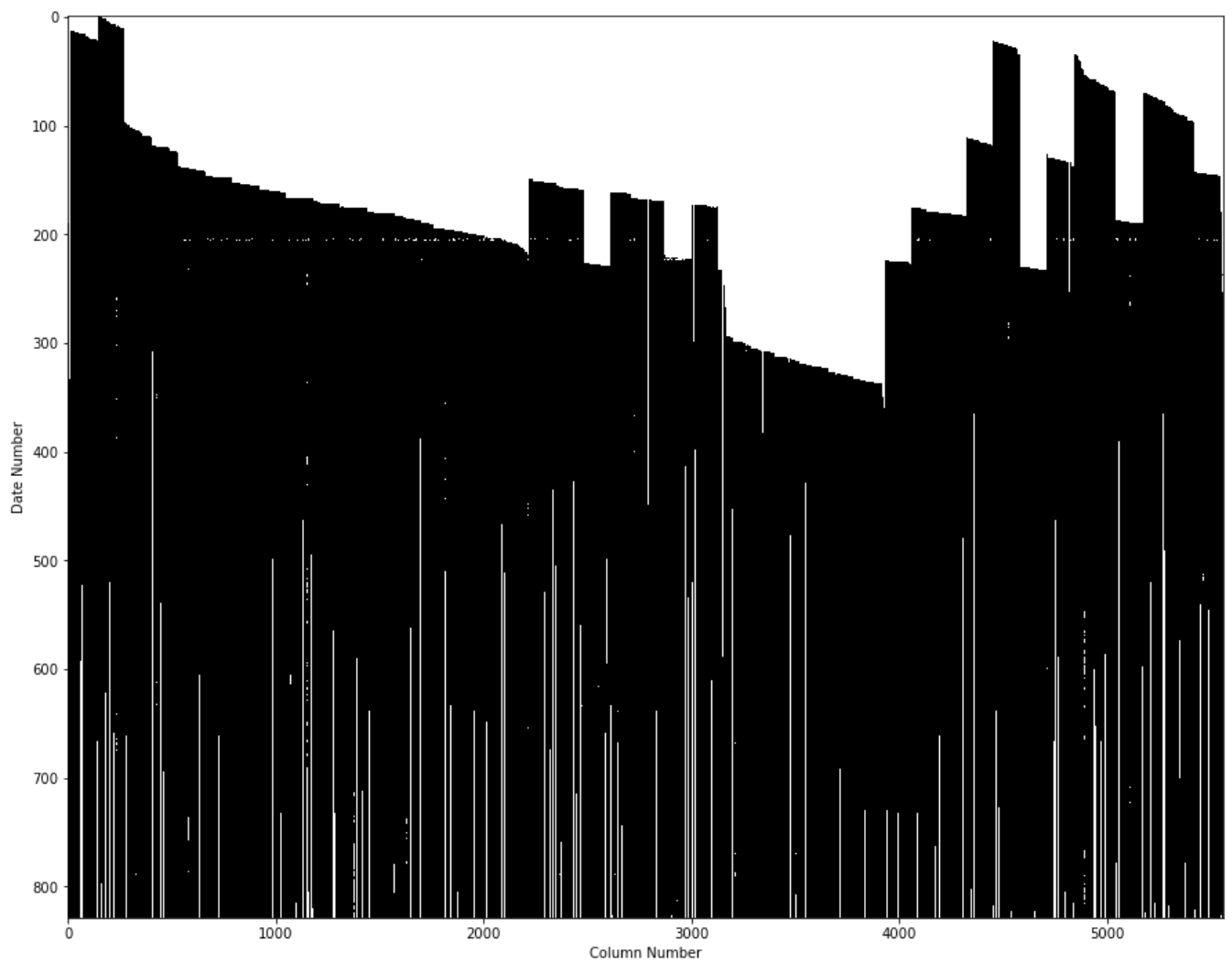
Par intérêt examinons la couverture des données. D'abord on réorganise pour avoir les résidences en colonne et les date en ligne.

```
In [13]: 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 [14]: 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 [15]: daily_mean_by_tariff_type = saved_daily_summary.pivot_table(  
    'kwh',  
    index='Date',  
    columns='Tariff Type',  
    aggfunc='mean'  
)  
daily_mean_by_tariff_type
```



Out[15]:

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

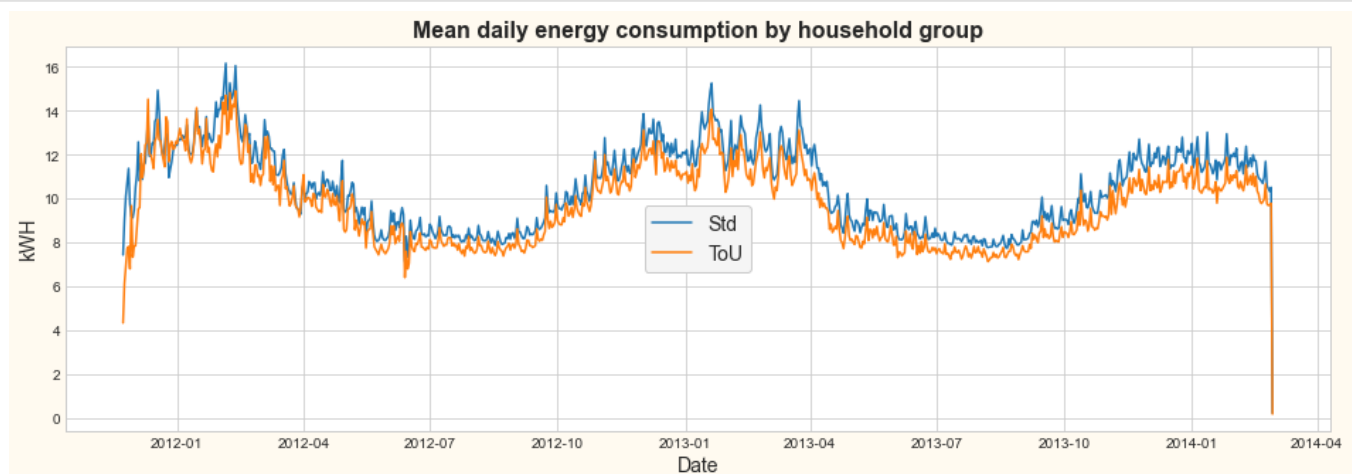
Finalement on peut tracer les deux groupes de données. Le traçage marche mieux si on convertit la date de type `string` en type `datetime`.

```
In [16]: daily_mean_by_tariff_type.index = pd.to_datetime(daily_mean_by_tariff_type.index)
```

```
In [17]: 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. Agégeons encore une fois, cette fois-ci par mois.

```
In [18]: 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

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 [19]: 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[19]:

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	238.904096
2012-09-30	266.392972	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	364.225310	331.578243

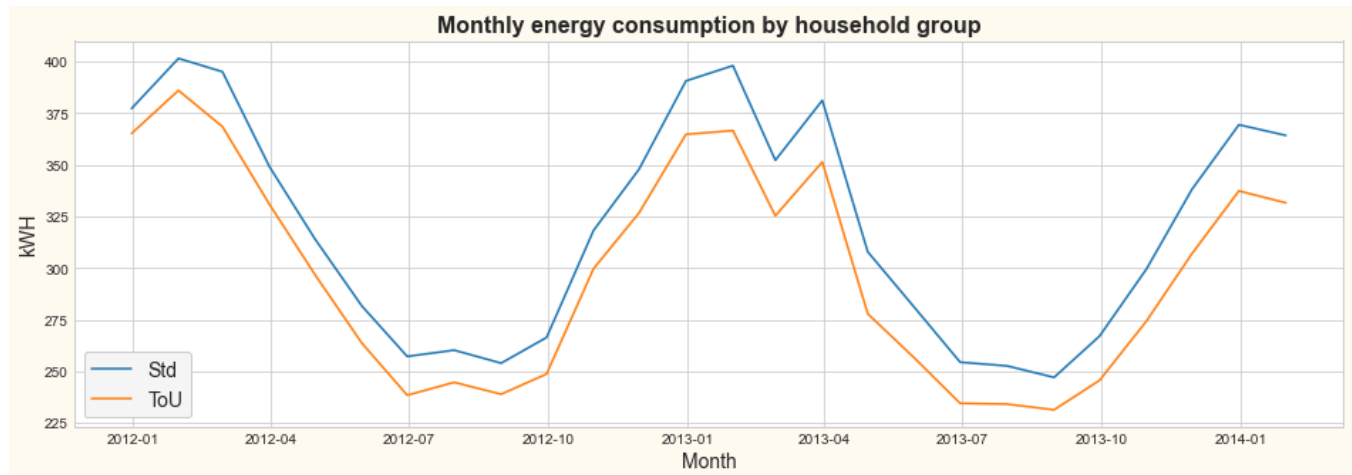
In [20]:

```
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.