# Projet Gestion Energie G11

**Léonard Simon – Maël Chambeau**

# Table des matières

# I. Introduction

Au cours du second semestre de première année à l'Ense3 nous avons pu réaliser un projet lié à notre choix de pré filière. Dans notre cas il s'est agi du projet « gestion d'énergie G11 », nous allons donc dans ce rapport revenir sur le projet, sur ces enseignements, ses aboutissements ainsi que ses conclusions. Nous structurerons ce rapport de manière chronologique afin de permettre la visualisation étapes par étapes de l'avancement du projet.

Nous définirons dans un premier temps le cadre de l'étude et présenterons le projet dans sa globalité. Nous pourrons ainsi revenir sur les objectifs et les enjeux du projet. Nous présenterons également les différents outils utiles à la réalisation sa réalisation.

Nous verrons ensuite la première étape du projet réalisée en langage Python concernant la réalisation d'un code permettant d'obtenir des informations sur l'occupation ou non d'une pièce du logement étudié.

Ensuite nous nous intéresseront à l'étape de travail sur la Rasberry Pi et le traitement d'information ayant pris le rôle d'une étape introductive au traitement des données.

Nous détaillerons ensuite comment nous avons manipulé un jeu de données fourni par les enseignants puis comment nous avons adapté cette étape au traitement d'un jeu de donnée réel récupéré sur la maison Expe-SmartHouse, sujet principal de l'étude.

Enfin nous verrons comment nous avons interprété et traité ces données et quelles conclusions nous avons pu en tirer.

Dans une dernière partie nous reviendrons sur les aboutissements du projet mais également ses échecs et comment nous pourrions ou aurions pu améliorer les résultats. Nous conclurons finalement de manière globale sur le projet et ses apprentissages.

Dans chaque partie nous adopterons un regard critique sur notre travail en tachant d'analyser quels en ont été les points positifs et négatifs mais également sur ce que nous avons pu tirer des différentes étapes de travail dans notre compréhension du travail par projet.

# II. Le projet Gestion Energie

## 1. Objet d'étude et description

L'objet d'étude de ce projet est une maison, la EXPE – SmartHouse, dotée de nombreux capteurs couvrant l'ensemble des variables pouvant être mesurées telles que la concentration en Co2 dans l'air, l'ouverture ou non d'une fenêtre, d'un interrupteur ou encore de la télévision. Toutes ces données sont stockées sur un site internet en open source, Grafana, et on peut donc les récupérer en temps réels sur la période souhaitée. L'objectif de ce projet est d'utiliser les données fournies par les capteurs pour déterminer s'il y a gaspillage d'énergie ou non. Nous n'étudierons qu'une seule pièce, la pèce à vivre de la maison. Afin d'évaluer le gaspillage d'énergie nous nous appuierons principalement sur l'ouverture de la fenêtre, le fonctionnement du chauffage et le taux de Co2 dans la pièce. Les résultats nous permettront de déterminer sur quels périodes de l'énergie est dépensée inutilement et donc de chercher des solutions ou du moins d'informer les occupants de ces pertes afin d'améliorer leur consommation d'énergie. Nous pouvons également aller plus loin en pensant par exemple à des solutions d'automatisation du chauffage et ou de l'aération de la pièce afin de limiter les pertes d'énergies.

## 2. Outils et ressources

Afin de répondre au cahier des charges du projet plusieurs ressources seront mises à notre disposition. Le principal objectif étant le développement d'un code informatique permettant l'analyse des données fournies par les capteurs, la principale ressource du projet aura été le langage Python. Nous avons utilisé différentes interfaces graphiques afin d'implémenter notre code notamment la console de l'ordinateur connectée à une Rasberry Pi en SSH ainsi que l'interface Spyder lancée à partir d'Anaconda.

La ressource première reste bien sûr les informations fournies par les capteurs équipant la SmartHouse et pouvant être récupérées à l'aide d'influxDB et visualisables sur Grafana. Ces données constituent donc la base de notre travail. Nous les récupèrerons ensuite à l'aide du logiciel Office Excel.

Ainsi, nous pouvons différencier, comme souvent dans ce type de projet, deux types de ressources, les Hardware (capteurs, Rasberry Pi…) et les Software (Excle, Spyder, Grafana, influxDB…). C'est en combinant ces ressources que nous sommes en mesure de réaliser la récupération et l'analyse des données permettant de répondre à notre problématique.

## 3. Les données

La maison est équipée au total de 360 capteurs, notre projet de recherche ne nécessite pas l'utilisation de toutes les données disponibles. En effet notre étude se concentre sur une seule pièce de la maison et sur la seule problématique du gaspillage d'énergie liée au chauffage et à l'ouverture d'une fenêtre de manière simultanée. Cela nous permet donc d'émettre au passage la remarque qu'en utilisant l'ensemble des données il est possible de répondre à un très grand nombre de problématiques qu'elles soient liées aux économies d'énergie ou à d'autres problèmes de fonctionnement.

Les données que nous retenons dans notre problème sont donc les suivantes :

- La température
- Le taux de Co2 dans l'air
- L'humidité
- L'ouverture de la fenêtre de l'évier (utilisées pour l'aération)
- Le fonctionnement ou non du chauffage
- La lumière
- Le son

# III. Extraction et nettoyage des données

## 1. Extraction des données

L'objectif de cette première partie du projet était de récupérer les données fournies par les capteurs de la SmartHouse dans des tableaux Excel. Pour cela nous avons donc écrit un code permettant de se connecter à chaque lancement du programme à la base de données INFLUXDB en ligne. Nous rentrons donc dans le code les noms d'utilisateurs et mots de passes adaptés ainsi que la procédure de connexion. Cela nous permet ensuite à l'aide des fonctions de Python de lire mais également d'écrire les données de la base de données dans un fichier Excel situé lui sur notre machine. Cela permet donc ensuite de pouvoir utiliser et modifier l'organisation de ces données bien plus facilement.

Voici la partie de code nous permettant d'écrire les données dans un fichier Excel :

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import urllib
import time
from influxdb import InfluxDBClient
import sys
import csv
##############################
#    SCRIPT SETTINGS
##############################
# Set the port where you want the bridge service to run
PORT_NUMBER = 1234
# InfluxDB Server parameters
INLUXDB_SERVER_IP = '82.65.155.71'
INLUXDB_SERVER_PORT = 8086
INFLUXDB_USERNAME = 'eleves'
INFLUXDB_PASSWORD = 'SmarthouseG2Elab'
INFLUXDB_DB_NAME = 'jeedom'
##############################

client = InfluxDBClient(INLUXDB_SERVER_IP, INLUXDB_SERVER_PORT, INFLUXDB_USERNAME, INFLUXDB_PASSWORD,

print(client.get_list_database())

client.switch_database('jeedom')

datasheet = client.query('SELECT "value" FROM "jeedom"."autogen"."3892" WHERE time > now() - 60d')

print(datasheet)

exported_data = list(datasheet.get_points())
header_list = list(exported_data[0].keys())
```

```
32
33  with open("humidité.csv", "w", newline='') as fp:
34      writer = csv.writer(fp, dialect='excel')
35      print(header_list[1:])
36      value_header = header_list[1]
37      offset = sum(c.isalpha() for c in value_header)
38      print(offset)
39      #header_list[1:] = sorted(header_list[1:], key=lambda x: int(x[offset:]))
40      header_list[1:] = ['value']
41      # print(header_list)
42      writer.writerow(header_list)
43      for line in exported_data:
44          # print(line)
45          writer.writerow([line[kn] for kn in header_list])
46
47
48
```

Le code à l'écran permet de récupérer les données concernant l'humidité et de les écrire dans un fichier .csv correspondant. Nous réalisons donc cette étape pour chaque donnée que nous souhaitons récupérer. Les données à récupérer sont identifiées dans INFLUXDB à l'aide d'un code à quatre chiffres que l'on renseigne à la ligne 26, dans le cas de l'humidité « 3892 ».

Il est également important de noter que c'est à cette étape que l'on décide de la plage de temps sur laquelle nous souhaitons récupérer les données. Nous avons fait le choix de réaliser notre étude sur 60 jours, d'il y a 60 jours à aujourd'hui. En effet la durée proposée à l'origine était plus courte et ne nous semblait pas représentative en effet comme nous allons par la suite dégager des statistiques de l'analyse des données la taille de l'échantillon doit être conséquente. Nous aurions également pu choisir des données situées dans la période hivernale pour obtenir un rôle plus important dans l'ouverture de la fenêtre et l'utilisation du chauffage mais la période du printemps étant intermédiaire elle semblait également valable. De plus il est possible de changer la période d'étude en modifiant simplement une ligne du code.

## 2. Nettoyage et préparation

Une fois les données récupérées il est important de les ordonnées afin de pouvoir les utilisées. A cette étape nous avons un tableau Excel par information ayant la forme suivante :

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | time,humidite | | | | |
| 2 | 2021-04-04T07:35:49Z,42.0 | | | | |
| 3 | 2021-04-04T07:45:22.648926Z,43.0 | | | | |
| 4 | 2021-04-04T07:54:56Z,42.0 | | | | |
| 5 | 2021-04-04T08:08:52Z,42.0 | | | | |
| 6 | 2021-04-04T08:29:02Z,43.0 | | | | |
| 7 | 2021-04-04T08:30:21.889042Z,42.0 | | | | |
| 8 | 2021-04-04T08:39:07Z,44.0 | | | | |
| 9 | 2021-04-04T08:59:18Z,44.0 | | | | |
| 10 | 2021-04-04T09:09:23Z,44.0 | | | | |
| 11 | 2021-04-04T09:27:38Z,44.0 | | | | |
| 12 | 2021-04-04T09:30:23.708758Z,43.0 | | | | |
| 13 | 2021-04-04T09:37:43Z,43.0 | | | | |
| 14 | 2021-04-04T09:45:32.063573Z,44.0 | | | | |
| 15 | 2021-04-04T09:57:53Z,42.0 | | | | |
| 16 | 2021-04-04T10:07:58Z,42.0 | | | | |
| 17 | 2021-04-04T10:28:08Z,42.0 | | | | |
| 18 | 2021-04-04T10:38:13Z,42.0 | | | | |
| 19 | 2021-04-04T10:45:18.844196Z,43.0 | | | | |
| 20 | 2021-04-04T10:58:24Z,41.0 | | | | |
| 21 | 2021-04-04T11:00:34.172355Z,42.0 | | | | |

Toutes les informations sont rangées dans la première colonne du tableau et mélangées. Le nettoyage des données consiste donc à séparer les colonnes par informations notamment ici récupérer la date et l'heure ainsi que l'information sur l'humidité dans ce cas. Une fois ces données séparées et rangées dans plusieurs colonnes du dataset concernant la donnée sur Python nous allons fusionner les dataset correspondant à chaque donnée afin de n'obtenir qu'un seul grand tableau facilement exploitable. La

bibliothèque Python de fonctions permettant de manipuler facilement les dataset que nous utilisons est la bibliothèque panda.

Il reste également à définir comme indexe de chaque dataset le temps. Ainsi le temps devient la référence et les données seront rangées à partir du temps dans le dataset final. Cependant, notre principal problème dans la suite du nettoyage est que chaque capteur ne prends pas les mesures simultanément par exemple de nombreux capteurs ne prennent une mesure que lorsqu'il y a une modification. On a donc par exemple les informations sur la fenêtre que lorsque celle-ci s'ouvre ou se ferme. Il va donc falloir compléter ces manques d'information pour pouvoir utiliser les données. En effet nous allons comparer différentes données entres elles et il nous faut donc des informations à chaque instant.

Voici la première partie du code permettant de récupérer les informations de chaque fichier Excel et de les écrire dans les dataset de la bibliothèque panda. On définit également l'index de chaque base de données sur le temps.

```
20
21  data_temp = pandas.read_csv('temp.csv', sep=",")
22  data_temp["datetime"] = data_temp["time"].apply(stringdate_to_datetime)
23
24  data_CO2 = pandas.read_csv('CO2.csv', sep=",")
25  data_CO2["datetime"] = data_CO2["time"].apply(stringdate_to_datetime)
26
27  data_humidite = pandas.read_csv('humidite.csv', sep=",")
28  data_humidite["datetime"] = data_humidite["time"].apply(stringdate_to_datetime)
29
30  data_fenetre_evier = pandas.read_csv('fenetre_evier.csv', sep=",")
31  data_fenetre_evier["datetime"] = data_fenetre_evier["time"].apply(stringdate_to_datetime)
32
33  data_heater = pandas.read_csv('heater.csv', sep=",")
34  data_heater["datetime"] = data_heater["time"].apply(stringdate_to_datetime)
35
36  data_light = pandas.read_csv('light.csv', sep=",")
37  data_light["datetime"] = data_light["time"].apply(stringdate_to_datetime)
38
39  data_sound = pandas.read_csv('sound.csv', sep=",")
40  data_sound["datetime"] = data_sound["time"].apply(stringdate_to_datetime)
41
42
43  data_temp.set_index('datetime', inplace = True)
44  data_CO2.set_index('datetime', inplace = True)
45  data_humidite.set_index('datetime', inplace = True)
46  data_fenetre_evier.set_index('datetime', inplace = True)
47  data_heater.set_index('datetime', inplace = True)
48  data_light.set_index('datetime', inplace = True)
49  data_sound.set_index('datetime', inplace = True)
```

On réorganise ensuite les données afin d'avoir une donnée toutes les 20 minutes dans chaque base de données. Lorsque nous avons trop d'information on effectue une moyenne. Pour les données ou nous n'avons pas assez d'information c'est-à-dire où les informations sont espacées de plus de 20 minutes les endroits manquants seront remplacés par des N/A (donnée non attribuée)

```
51  temp = data_temp.resample('20T').mean()
52  CO2 = data_CO2.resample('20T').mean()
53  humidite = data_humidite.resample('20T').mean()
54  fenetre_evier = data_fenetre_evier.resample('20T').mean()
55  heater = data_heater.resample('20T').mean()
56  light = data_light.resample('20T').mean()
57  sound = data_sound.resample('20T').mean()
```

Nous devons donc ensuite remplacer ces N/A par les données réelles. Par exemple lorsque l'information sur l'ouverture de la fenêtre marque N/A nous savons qu'en réalité elle est encore en

l'état de la dernière information fournie (valable pour le chauffage ou pour la lumière également). On a donc cherché à remplir ces valeurs N/A nous-même par le code (en répétant la dernière valeur mesurée ou en effectuant une moyenne/interpolation en fonction des types de capteurs) mais cela s'est avéré trop complexe… On a alors utilisé des fonctions de panda qui comblent les trous automatiquement. En l'occurrence, La fonction ffill(), « forward fill », permet de réaliser cet étape en remplaçant chaque trou d'information par la dernière information connue.

```
59  temp = temp.ffill()
60  CO2 = CO2.ffill()
61  humidite = humidite.ffill()
62  fenetre_evier = fenetre_evier.ffill()
63  heater = heater.ffill()
64  light = light.ffill()
65  sound = sound.ffill()
```

Enfin il reste une dernière étape dans la préparation des données. En effet dans certains cas, la moyenne utilisée pour obtenir une donnée toutes les 20 minutes peut créer des données fausses avec par exemple des 0.5 pour l'ouverture de la fenêtre alors que celle-ci ne peut être qu'ouverte ou fermée donc 0 ou 1. Nous allons corriger cela à l'aide de quelques lignes de code supplémentaires.

Voici le code dans le cas du chauffage :

```
 93  newheater=[]
 94  value=heater['heater']
 95  for n in range (len(value)):
 96      if ((type(value[n])) != int):
 97
 98          if (value[n]<0.5):
 99              newheater.append(0)
100          else :
101              newheater.append(1)
102
103 heater['heater']=newheater
```

Nous regroupons ensuite les DataFrame correspondants à chaque donnée :

```
103 heater['heater']=newheater
104
105 mergeData = [temp, CO2, humidite, fenetre_evier, heater, light, sound]
106
107 result = pandas.concat(mergeData, axis=1, join = 'inner')
```

Nous obtenons finalement un DataFrame de la forme suivante :

result - DataFrame

| datetime | temp | CO2 | humidite | :netre_evi | heater | light | sound |
|---|---|---|---|---|---|---|---|
| 2021-04-05 05:20:00 | 19.9 | 606.5 | 43 | 1 | 1 | 0 | 54 |
| 2021-04-05 05:40:00 | 20.1 | 671 | 42.5 | 1 | 0 | 0 | 56.3333 |
| 2021-04-05 06:00:00 | 20.1 | 703.667 | 44 | 1 | 1 | 0 | 48 |
| 2021-04-05 06:20:00 | 20.4 | 674 | 44 | 1 | 1 | 0 | 48 |
| 2021-04-05 06:40:00 | 20.4667 | 682.667 | 42.6667 | 1 | 0 | 0 | 57 |
| 2021-04-05 07:00:00 | 21.2 | 740.333 | 43.3333 | 1 | 1 | 0 | 54.6667 |
| 2021-04-05 07:20:00 | 20.9 | 785 | 43 | 1 | 0 | 0 | 49.5 |
| 2021-04-05 07:40:00 | 21.2667 | 786 | 43 | 1 | 1 | 0 | 50.6667 |
| 2021-04-05 08:00:00 | 21.2 | 789.333 | 43 | 1 | 1 | 0 | 55.6667 |
| 2021-04-05 08:20:00 | 21.6 | 870 | 43 | 1 | 0 | 0 | 51.5 |

Format Resize Background color Column min/max Save and Close Close

Ainsi, cette étape de préparation et de nettoyage des données est très importante car elle va grandement faciliter la suite du projet cet à dire l'analyse des données. La plus grande partie du travail en termes de temps aura été de préparer ces données. L'analyse prendra ensuite « beaucoup » moins de temps car notre DataFrame est propre et fonctionnelle.

## IV. Decision tree et occupation de la pièce

### 1. Decision tree

Dans cette première partie de projet l'objectif était d'implémenter un code permettant de déterminer si la pièce à vivre de la maison était occupée ou non. En effet, dans notre schéma d'analyse des données nous avons dans un premier temps décider de n'étudier les pertes d'énergies que dans le cas où la pièce était occupée. Cependant nous ne disposons pas d'information directe sur l'occupation de la pièce, nous devions donc implémenter un ensemble de fonction qui, s'appuyant sur les données fournies par les capteurs, fournissait l'occupation ou non de la pièce en fonction du temps.

Les données que nous choisissons d'utiliser sont les suivantes :

- Lumière (ON/OFF)
- Son (dB)
- Humidité (%)
- Fenêtre de l'évier (utilisée pour l'aération, CLOSE/OPEN)

La fonction principale de ce programme est appelée « Decision Tree », il s'agit d'une fonction en machine learning. Ainsi l'objectif est de lui fournir un premier set d'informations puis en réalisant un certain nombre d'itérations elle apprend et donne un résultat de plus en plus précis.

Voici l'extrait du code où nous définissons cette fonction :

```
x_train, x_test, y_train, y_test = train_test_split(data[["light","sound","humidite","fenetre_evier"

# Define the most important features
def importance (x_train, y_train) :
    classifier = tree.DecisionTreeClassifier(random_state=0, criterion='entropy', max_depth=None)
    classifier.fit(x_train, y_train)
    important = classifier.feature_importances_
    return important

most_important= importance (x_train, y_train)
print("most_important",most_important)


# implementation of decision tree

def decision_tree(x_test,y_test,x_train,y_train) :

    classifier=tree.DecisionTreeClassifier(max_depth=3)
    classifier.fit(x_train,y_train)
    prediction=classifier.predict(x_test)
    accuracy = classifier.score(x_test, y_test)
    print("accuracy", accuracy)
    features_importance= classifier.feature_importances_
    #print(" feature_ importance",features_importance)
    indices=np.argsort(features_importance)
    # plt.title('feature importance')
    # plt.barh(range(len(indices)), features_importance[indices], color = 'b', align = 'center')
    # plt.yticks(range(len(indices)), [x_train.columns.values[i] for i in indices])
    # plt.xlabel('Relative importance')
    with open ("tree.dot","w") as file :
        f=tree.export_graphviz(classifier,out_file=file)
    return(prediction, accuracy)
```

Cette fonction s'accompagne donc ensuite d'autres fonctions permettant de tester la précision du résultat fourni à chaque itération et donc d'améliorer la précision et l'efficacité du Decision Tree.

```

(prediction, accu) = decision_tree(x_test,y_test,x_train,y_train)
print("the predicted occupancy is: \n",prediction)


def accurary_dif_depths(size):
    depth = [] #abcisse
    accuracy = []  #ordonnée
     #show the accuracy from 1 to size-1
    for i in range(1, size):
        classifier=tree.DecisionTreeClassifier(max_depth=i)
        classifier.fit(x_train,y_train)
        depth.append(i)
        accuracy.append(classifier.score(x_test, y_test))  #accuracy for dpth = 1
        #print( i, classifier.score(x_test, y_test)) #to test
    #show the accuracy for 'None' last
    classifier=tree.DecisionTreeClassifier(max_depth=None)
    classifier.fit(x_train,y_train)
    depth.append(i)
    accuracy.append(classifier.score(x_test, y_test))
    plt.plot(depth, accuracy)
    return (depth, accuracy)

(depth,accuracy) = accurary_dif_depths(3)

```

```
231   def calcul_efficiency(x_test,y_test,x_train,y_train) :
232       list_effective=[]
233       list_price=[]
234       list_accuracy=[]
235       max_efficient = 0
236       #test all combinasion of sensors, to define the best deal accuracy/price
237       sensors = ["light","sound","humidite","fenetre_evier"]
238       for i in range(1, len(sensors)-0):
239           for tuple_combi in combinations(sensors, i) : #test all combinaison possible between sensors
240               combi=[]
241               for i in tuple_combi:
242                   combi.append(i)    #convert tuple into list
243               print(combi)
244               #divide new data into training and testing
245               x_test2 = x_test[combi]
246               x_train2 = x_train[combi]
```

A l'aide de ces fonctions nous devions pouvoir fournir, en fonction des autres données et donc de la période sur laquelle nous étudions notre pièce, un tableau d'occupation de la pièce précisant si oui ou non il y avait un occupant.

## 2. Autres solutions

Cependant le code ne fonctionnait que partiellement et nous n'avons pas réussi à l'intégrer dans le reste du programme. Ainsi nous avons étudié d'autre possibilités pour déterminer l'occupation ou non de la pièce. Tout d'abord nous avons tenté de lier l'occupation de la pièce à l'heure de la journée. Nous avons donc supposé que la pièce soit occupée de 17h à 23h et de 6h à 8h ce qui correspond aux heures ou les habitants sont supposés ne pas travailler et ne pas dormir. L'objectif d'ajouter au dataset une colonne remplie de 1 dans les heures ou la pièce est occupée et de 0 lorsqu'elle ne l'est pas.

Nous avons écrit le code suivant :

```
140
141   data.set_index('datetime', inplace = True)
142   hours = []
143
144   result.datetime = pandas.to_datetime(result.iloc[:,0], format = '%Y-%m-%dT%H:%M:%S')
145
146   for dt in result.datetime:
147       hours.append(dt.hour)
148   result['hour'] = hours
149
150   for i in range (len(result.datetime)):
151       if result.datetime.H[i]>17 and result.datetime.H[i]<23 :
152           hours.append(1)
153       if result.datetime.H[i]>6 and result.datetime.H[i]<8 :
154           hours.append(1)
155       else:
156           hours.append(0)
157
```

La liste hours correspond donc à notre indicateur d'occupation de la pièce. Cependant pour des raisons de connaissances informatiques nous n'avons pas réussi à trouver la syntaxe pour comparer les heures. La syntaxe « date > 17 » que l'on peut voir dans le code ci-dessus ne fonctionne donc pas. Pour des raisons de temps nous n'avons pu chercher la bonne solution. Cependant, même si cette solution apparaissait plus simple que l'utilisation du Decision Tree elle introduit dans notre analyse de nombreuses erreurs car il est probable que, sur de nombreuses journées, l'occupation de la pièce ne corresponde pas à notre supposition.

Enfin, la dernière option correspondait à l'analyse des données sans prendre en compte l'occupation de la pièce mais simplement le gaspillage d'énergie. Cela vaut dire que nous ne prenions alors plus en compte seulement les moments ou les occupants pouvaient apporter une correction directe au gaspillage mais également les moments où ils étaient absents et où l'erreur ne pouvait être réparée.

Cela signifie que nous considérons par exemple également les oublis tels qu'un départ en week-end en ayant oublié de fermer la fenêtre ou d'éteindre le chauffage. Cette nouvelle option apporte une approche différente et donc des résultats à une problématique différente. Néanmoins, cette problématique semble tout autant intéressante que la première problématique que nous nous étions posée.

# V. Résultats et analyses

## 1. Résultats

Maintenant que nous avons pu préparer au mieux nos données et choisis ou changées les conditions d'observations. Nous allons pouvoir établir des résultats à partir de notre base de données. Nous avons donc choisi d'analyser en tout temps les pertes liées à l'ouverture non nécessaire de la fenêtre lorsque le chauffage est allumé. Nous allons dans un premier temps chercher à quantifier ces pertes en unités de temps puis nous tenterons éventuellement de les quantifier en termes d'énergie perdue. Pour cela nous écrivons un petit code permettant de tester les trois conditions citées plus haut c'est-à-dire si on a le chauffage allumé, la fenêtre ouverte, et un taux de Co2 ne nécessitant pas l'ouverture de celle-ci en simultané.

Voici donc ce code :

```
compteur_waste = 0

for i in range (result.shape[0]):
    if result.CO2[i] < 1000 :
        if result.fenetre_evier[i] == 1:
            if result.heater[i] == 1:
                compteur_waste = compteur_waste + 1
```

```
minutes_waste = compteur_waste*20
jour_waste = (minutes_waste/60)/24
print("On a perdu de l'énergie pendant:", minutes_waste, "minutes")
print("Ainsi on perd de l'énergie pendant :", jour_waste, "journées, sur 60 jours")
```

On test donc la présence des trois paramètres sur tout le DataFrame donc sur la durée de 60 jours et on ajoute à chaque fois que c'est le cas +1 dans notre compteur. Puis sachant que chaque donnée correspond à une durée de 20 minutes on effectue la conversion en minute, en heure, puis en jours.

On obtient alors le résultat suivant :

```
On a perdu de l'énergie pendant: 12300 minutes
Ainsi on perd de l'énergie pendant : 8.541666666666666 journées, sur 60 jours
Soit un pourcentage de : 14.23611111111111 % du temps
```

De plus la température relativement élevée de la maison nous ayant marquée nous avons décidé de sortir un second résultat du jeu de données. Nous avons donc supposé que le chauffage est allumé à tort dès lors que la température dépasse les 22°C, en effet dans une maison la température de 22°C est très élevée et il semble étonnant d'avoir en plus le chauffage allumée alors qu'on chercherait même peut être à refroidir le logement.

Voici donc le code nous permettant d'obtenir la quantité de temps ou le chauffage est allumé lorsqu'il fait plus de 22°C :

```
117  Gaspi_heater = 0
118
119  for i in range(result.shape[0]):
120      if result.heater[i] == 1 and result.temp[i] > 22:
121          Gaspi_heater = Gaspi_heater + 1
122
```

```
133  Gaspi_heater_pourcent = ((((Gaspi_heater*20)/60)/24)/60)*100
134  print("Le chauffage a été annulé inutilement", Gaspi_heater_pourcent, "% du temps")
```

On obtient le résultat suivant :

```
Le chauffage a été annulé inutilement 29.212962962962962 % du temps
```

### 2. Analyses

Notre premier résultat nous donne un gaspillage d'énergie 14% du temps sur la période de 60% selon la première définition que nous avions faite du gaspillage lié à l'ouverture de la fenêtre, à l'allumage du chauffage et à un taux de Co2 acceptable. Il s'agit d'un chiffre très élevé d'autant plus que nous ne prenons plus en compte la présence des habitants. Ainsi si l'on suppose que dès qu'ils ne sont pas présents la fenêtre est fermée ; cela veut dire que lorsqu'ils sont présents, les moments où la fenêtre est ouverte à tort alors que le chauffage est allumé sont très nombreux voir majoritaires. Ainsi, il semble important de revoir les périodes d'allumage du chauffage.

Notre second résultat est lui encore plus surprenant en effet il montre que 29% du temps le chauffage est allumé alors qu'il fait plus de 22°C dans la pièce à vivre. Cela semble énorme. Plusieurs explications sont envisageables, ou bien les températures sont très élevées parce que justement ce chauffage est allumé à tort auquel cas il faut une nouvel fois s'inquiéter du dispositif d'allumage du chauffage si celui-ci est automatique ou informer la famille qu'il n'est pas nécessaire de vivre à une telle température.

Cependant au vu des chiffres très élevés que nous obtenons nous pouvons également remettre en cause les résultats obtenus. Il est éventuellement possible qu'une partie des valeurs que nous avons ajoutées à l'aide de la fonction ffill() soient fausses ou encore que le capteur ait fournis de fausses informations. Pourtant, en relisant le DataFrame de manière aléatoire, étant impossible de lire chaque valeur, les résultats obtenus semblent justes. Ainsi, soit la consommation sur la période observée a été mal gérée ce qui est possible suite à une succession d'erreur, soit le chauffage était mal réglé, soit certaines des données fournies par les capteurs sont fausses. Dans cette situation et avec plus de temps, il semble juste de vérifier les capteurs mais également de vérifier plus précisément les valeurs de la DataFrame. Enfin, il n'est pas impossible que l'erreur provienne directement du code et une relecture extérieure pourrait permettre de relever et d'éliminer les erreurs.

En termes de résultats nous aurions pu nous intéresser à d'autres informations tels que par exemple l'allumage de la lumière alors que personne n'était présent dans la pièce ou encore l'allumage de la télévision. Il s'agit une nouvelle de sources courantes de gaspillage d'énergie pouvant être améliorés par l'utilisation de capteurs et l'automatisation des systèmes. Cependant pour obtenir ces résultats nous aurions eu besoin de connaître précisément l'occupation de la pièce. Une nouvelle fois seul le temps à manquer pour obtenir ces résultats car nous aurions pu améliorer le fonctionnement du decision tree.

Ainsi avec la seule DataFrame que nous avons préparée et nettoyée lors de ce projet nous aurions pu obtenir de nombreux résultats concernant la consommation d'énergie dans la pièce principale de la maison. L'observation d'un nombre réduit de variable a permis de relever plusieurs incohérences dans la consommation ainsi l'automatisation d'une partie des systèmes énergétiques de la maison peut également être mise à l'étude pour améliorer encore les performances du bâtiment.

# VI. Conclusions

Au cours de ce projet nous sommes passés par toutes les étapes d'un problème de traitement de données de la récupération de celles-ci, passant par le nettoyage et la préparation et jusqu'à la rédaction du code permettant de les analyser et d'en tirer des conclusions. Les résultats que nous avons obtenus permettent de se faire une idée approximative des pertes d'énergie dans la pièce étudiée. Bien que manquant de précision à notre niveau, cette méthode montre qu'il est aujourd'hui essentiel de faire appel au machine learning et à des capteurs performants pour résoudre ce genre de problématique. Une version optimisée et applicable à tous les logements serait d'une efficacité considérable pour lutter contre le gaspillage et participer à la préservation de l'environnement.

## 1. Regard critique sur les données

Nous avons tenu à apporter un regard critique à notre travail et nous avons parfois ressenti quelques incohérences entre les données et leur exploitation. En effet, on remarque que le chauffage s'allume et s'éteint de façon très fréquente (bien supérieure à 20 minutes), ce qui réduit la fiabilité d'exploitation de cette donnée. En effet, on sait qu'il est automatisé et se régule seul donc cela n'est pas surprenant mais il difficile de dire si on perd de l'énergie en se basant sur l'état allumé ou éteint du chauffage… Il faudrait plutôt dire à l'utilisateur de baisser sa consigne lorsqu'il est absent ou lorsque la fenêtre est ouverte ! Puisque dans tous les cas le chauffage s'allume et s'éteint pour maintenir sa température, ce n'est surement pas la donnée la plus adéquat à utiliser dans notre problème.

Plus généralement, concernant les capteurs où les valeurs prises sont 0 ou 1, nous avons pris comme convention de remplacer les valeurs au-dessus de 0,5 (obtenues en moyennant plusieurs on/off dans la période de mesure) par 1 donc allumé, ce qui engendre forcément une erreur plus ou moins importante.

## 2. Suggestions pour améliorer la méthode

Voici pour finir quelques suggestions pour améliorer ce travail tant au niveau de la précision que de la cohérence :

- Revoir le mode de mesure concernant le chauffage ( VI.1.)
- Augmenter la durée sur laquelle on mène l'étude, une année par exemple pour plus de réalisme puisque les pertes changent selon les saisons.
- Diminuer la durée choisie pour prendre les mesures (20min pour nous) pour réduire de ce fait les imprécisions (fonction ffill, interpolations, moyennes …)
- Prendre en compte d'autres données comme la lumière ou la consommation d'appareils électriques pour affiner nos résultats sur les pertes.

- On pourrait également imaginer un système qui informerait en temps réel les habitants concernant le gaspillage ce qui leur permettrait d'agir.