BERNARD Pauline
CORMAN Matthieu

Projet 11: Gestion d'énergie

Professeurs encadrants :
Manar AMAYRI
Jérôme FERRARI

Matthieu Corman
Pauline Bernard

I/ Contexte

En France, l'habitat est à l'origine de 40% des émissions de CO2. Le Grenelle de l'Environnement encourage les foyers à s'équiper de systèmes de domotique. En effet, l'automatisation permet de limiter la consommation grâce à des fonctionnements précis afin d'éviter des utilisations inutiles.

Les appareils connectés permettent également de suivre en direct la consommation de la maison, de repérer immédiatement une surconsommation et d'y mettre fin rapidement. La domotique peut concerner de nombreux aspects de l'habitat. Chacun peut personnaliser comme il le souhaite le fonctionnement de ses appareils selon des scénarios pré enregistrés ou grâce à des commandes à distance. Le système peut par exemple enregistrer ce qui doit être appliqué lorsque le domicile est vide ou la nuit.

On estime que la domotique permettrait de faire des économies énergétiques de 25 à 30% concernant le chauffage, de 10% la consommation pour l'éclairage. De manière générale, la domotique permettrait de réduire jusqu'à 20% la consommation d'énergie du logement.

II / Problématique

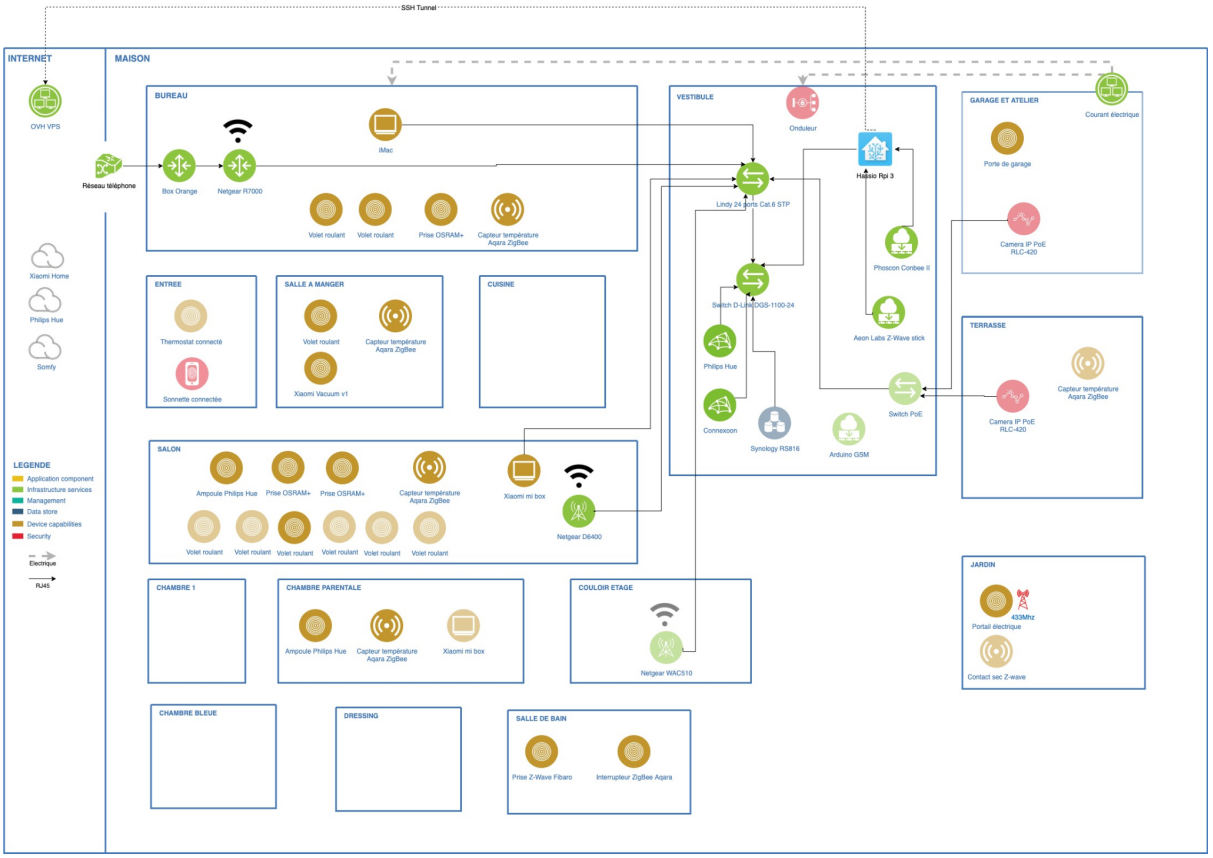
Le projet Expe-smarthouse a été initié en 2018 afin de fournir les données d'un foyer de 120 m² où vit une famille de 5 personnes. Ce projet donne accès à environ 340 points de mesure pour les scientifiques, accessibles en temps réel via un portail Grafana alimenté avec une base de données Influxdb.

Cette maison intelligente est développée sur la base de matériel et de logiciels Open Source. Le système est également flexible en ce qui concerne l'ajout de nouvelles technologies ou de nouveaux capteurs.

Les mesures remontées sont:

- Consommation d'électricité, de gaz et d'eau de chaque appareil
- Température, humidité et luminosité de chaque pièce
- Position d'ouverture de chaque porte et fenêtre
- Détecteurs de mouvement
- État de l'éclairage
- Analyse de l'air de chaque pièce
- Conditions météorologiques extérieures

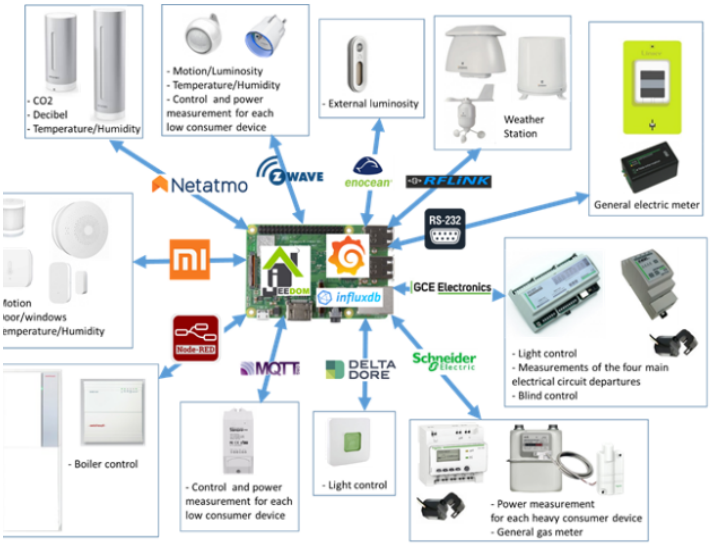
Voici un exemple de l'architecture utilisée



Source : https://domotique.yann.me/architecture.html

Notre projet consiste à prévenir d'éventuelles pertes d'énergie dans une pièce précise de la maison, le salon. Des capteurs nous fournissent les données que nous devons interpréter, traiter et confronter afin d'optimiser la consommation de cette pièce de vie. La difficulté du sujet provient de l'interprétation de ces données et des exigences que nous voulons pour réduire les dépenses énergétiques.

L'objectif de ce projet était d'apprendre à concevoir et à développer une "box" de gestion énergétique pour une zone de vie exploitant des capteurs autonomes très basse consommation. Ces capteurs permettent de construire des tableaux de bord pour l'aide à l'exploitation et l'intégration de stratégies de gestion énergétique rudimentaires.



III/ Machine learning

La première étape d'apprentissage dans le cadre de notre projet concernait le processus de machine learning. En effet, Mme Amayri nous a présenté étape par étape la construction d'un arbre de décision en langage Python à l'aide de la bibliothèque scikit-learn. L'étude a été conduite sur les données tirées des capteurs du bâtiment GreEn-ER constituées d'informations sur la température intérieure, extérieure, l'humidité, le détecteur de mouvement, la puissance électrique, la concentration en CO2, l'ouverture de la porte, les décibels et enfin notre "label" : le nombre de personne présente dans une salle. L'objectif était de créer un arbre de décision capable de dire à l'utilisateur s'il y a quelqu'un dans une pièce en fonction des données exprimées précédemment mais sans l'information label.

On peut classer les diverses informations à l'aide de la fonction feature_importance_ de la plus décisive dans notre problème, c'est-à-dire le facteur qui dépend le plus de la présence d'une personne dans la salle, à la moins pertinente. Ici, il s'agit de la puissance électrique, ce qui paraît logique.

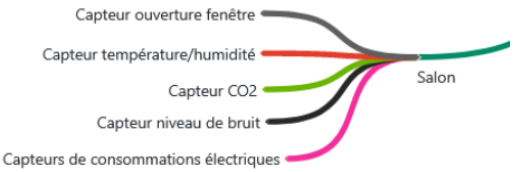
Nous avons dû discrétiser les données label en 0 et 1 pour transformer des décimaux en 0 et 1 et ainsi être plus compréhensibles pour l'arbre. Après avoir construit l'arbre, on peut le tester avec le même label qui l'a entraîné. Même avec un seul niveau de profondeur, c'est-à-dire en prenant en compte seulement l'information désignée comme la plus importante, on voit que les résultats sont très bons avec une précision de plus de 95%. Cependant, augmenter la profondeur de l'arbre, c'est-à-dire rajouter des critères de décision n'augmente pas drastiquement la précision, et il arrive même un moment où si l'on prend en compte trop de critères, l'arbre n'est plus pertinent.

Cette partie nous aura donc instruit sur comment réaliser un arbre de décision mais aussi comment le tester par la suite. Ceci peut être utile lorsqu'une partie des informations dans un problème vient à manquer, pour la compléter, on peut se servir des informations que l'on possède déjà.

IV/ Méthode

Les données des capteurs sont récupérées sur la plateforme Grafana. La création d'une base de données nous a permis d'exporter les valeurs des différents capteurs dans différents fichiers type csv. Pour le salon nous avons choisi de récupérer les données suivantes :

- Consommation de la télévision
- Consommation de la box-tv
- Etat logique de l'éclairage (Normalement éteinte)
- Luminosité
- Humidité
- Température
- Etat logique de la fenêtre (Normalement fermée)
- Bruit
- CO2



Nous voulions également récupérer les données concernant le chauffage mais nous avons rencontré des problèmes lors de l'exportation des valeurs. Celles-ci étaient nulles depuis de nombreuses semaines car nous sommes en période estivale.
Nous avons donc désormais plusieurs fichiers mais leur exploitation n'est pas encore possible car le format d'enregistrement présente une ponctuation particulière. Le but est alors de discrétiser les données, c'est-à -dire de les rendre lisibles pour notre programme Python. Chaque grandeur est ensuite mise dans une liste où une heure est affectée pour chaque mesure.

Nous voulons pouvoir comparer des données à la même heure cependant l'enregistrement des capteurs ne se fait pas à la même fréquence ni aux mêmes instants. Nous allons faire une première simplification au problème en ne gardant que les valeurs des heures rondes ainsi que des demie-heures. Désormais nous regroupons les listes des grandeurs dans un tableau afin de ne manipuler qu'un seul objet par la suite. Pour une heure donnée nous avons donc toutes les valeurs de nos capteurs et nous pouvons les exploiter.

Nous avons rencontré une difficulté pour exploiter les données temporelles dans la suite du programme. En effet le module "datetime " ne permet pas de récupérer la colonne des heures pour une grandeur donnée. Nous aurions voulu récupérer une liste avec des heures que nous aurions aussi fusionner dans notre tableau final.
Cependant nous avons dû créer nous même notre liste d'heures à partir de la première liste de la grandeur CO2. Nous avons donc une liste d'heures que nous avons seulement réussi à simplifier avec les heures pleines.

Le programme que nous cherchons à mettre au point présente l'inconvénient de perdre des données. En effet, ne garder que certaines heures réduit la taille de la liste initiale. De plus, la fusion des listes se fait en fonction de la liste la plus petite et donc on perd encore des données de certaines grandeurs. Le nombre de lignes du tableau est de 146 en sortie de programme alors que le nombre de relevés de CO2 était initialement de 746 par exemple.

Une donnée intéressante à avoir en plus de celles obtenues grâce aux capteurs serait la présence ou non de personne dans le salon. Deux solutions s'offrent à nous pour obtenir cette information. Soit nous utilisons l'arbre de décision construit dans la partie III mais en créant notre propre label "occupancy", soit nous décidons de nous même les heures où il y a une forte chance d'y avoir quelqu'un dans le salon. Malheureusement, dans les deux cas, cela revient à faire une estimation de la présence ou non. Ainsi, pour simplifier notre travail mais aussi pour éviter de créer un arbre de décision potentiellement faux (car on l'entraîne avec une prédiction), nous choisissons de créer la liste "occupancy" composée des états logiques "0" (lorsqu'il n'y a personne) et "1" (lorsqu'il y a quelqu'un) avec une prédiction qui nous paraît judicieuse en prenant compte de la vie d'une famille : on considère qu'il y a des habitants entre 7h00 et 8h00, entre 12h00 et 13h00 et entre 17h00 et 23h00.

Nous avons créé ce schéma en suivant les hypothèses suivantes :

- Les enfants partent avec leurs parents pour aller au travail avant 8h00.
- Au moins une personne rentre le midi à la maison.

- Les enfants rentrent avec les parents au voisinage de 17h00.
- Tout le monde dort vers 23h00.

Le salon est donc vide la nuit, pas d'état logique à 1 la nuit. Ce schéma a des limites. Une journée de week-end n'est pas différenciée d'une journée de travail. De plus, les passages ponctuels dans le salon la nuit par exemple ne sont pas pris en compte. Or, ils peuvent être l'occasion d'une ouverture de fenêtre ou d'utilisation de l'éclairage.

Il nous reste désormais à choisir les conditions pour les scénarios dans lesquels la maison perd de l'énergie.

V/ Consommation énergétique

C'est dans cette dernière partie que la domotique entre en jeu. Pour réduire la consommation de la maison et la rendre autonome il lui faut des consignes à suivre. Les capteurs fournissent les données mais c'est la dernière partie de notre programme qui permet de savoir si on consomme trop et si des changements, automatisations sont à prévoir.

Nous avons estimé qu'il y avait des pertes d'énergie dès lors que :
- La lumière est allumée alors que personne n'est présent dans le salon.
- La télévision et la box sont allumées alors que personne n'est dans le salon.
- La température est inférieure à 20 degrés et la fenêtre est ouverte.

Cette dernière condition est réalisée dans l'hypothèse ou le chauffage est allumé pour une température inférieure à 20 degrés. Nous n'avons pas accès aux informations concernant la chaudière. Nous voulions également mettre en place une condition sur la teneur en CO_2 de la pièce mais il est difficile de savoir si on peut ouvrir ou fermer la fenêtre sans les données de chaudière. On retombe dans le cas précédent sinon.

L'approche que nous avons construite autour de cette problématique présente quelques limites. Nous n'avons pas réussi à traiter les informations pour chaque valeur de temps donnée. Nous avons pu traiter seulement 20% de certaines données récupérées. De plus, la discrétisation des heures nous a fait perdre des informations sur la date. Le résultat en sortie de programme est le nombre d'occurrences de pertes d'énergie pour les trois jours de données. Ce nombre est une estimation basée sur des hypothèses et simplifications fortes.

Il serait intéressant de poursuivre nos travaux dans le but de combler ce manque d'exploitation lié aux heures.

Conclusion

L'étude que nous avons menée nous donne les heures et causes des pertes d'énergie qui ont lieu dans le salon. Avec ces résultats, on peut programmer une box en sachant quels sont les facteurs responsables de cette perte d'énergie. Nous avons rencontré plusieurs obstacles durant ce projet , cependant, nous avons tout de même réussi à obtenir des résultats concrets avec les données fournies.

```
# projetséance2 (1).py

import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn import metrics

#read fiche .csv

import pandas as pd
data = pd.read_csv('C:/Users/matth/Dekstop/data.csv', sep= ';')
Temperature=data['Tin']

labelcopy= []
for k in range (len(data['label'])):
    if ((type(data['label'])) != k):
        if (data['label'][k] <=0.5):
            labelcopy.append(0)
        else:
            labelcopy.append(1)

data['labelcopy']=labelcopy
print(data)

for k in range(len(data['power'])):
    if data['power'][k] ==0 and data['office_CO2_concentration'][k] < 400 and
data['CO2_corridor'][k]<400:
        print("There's no one")
    elif data['power'][k] > 0:
        if data['Door'][k]!=0:
            print("There’s at least one person")
        if data['acoustic_pressure_dB'][k]!=0:
            print("There’s at least one person")


# Train_test_split

xtrain, x_test, y_train, y_test = train_test_split (
    data[["Tin","Tout", "humidity",
"detected_motions","power","office_CO2_concentration","Door","CO2_corridor","acoustic_
pressure_dB"]],
    data[["labelcopy"]], test_size = .3, random_state = 0)


# Implement the DT

def decision_tree(x_train, x_test, y_train, y_test,D):
    clf = tree.DecisionTreeClassifier(max_depth=D)
    clf = clf.fit(x_train, y_train)
    y_test_pred = clf.predict(x_test)
    accuracy = clf.score(x_test, y_test)
    print (accuracy)
    importance = clf.feature_importances_
    print ("Features importance", importance)
    indices = np.argsort(importance)
    plt.title("Feature Importances")
    plt.bar(range(len(indices)), 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("MyTree.dot", 'w') as file:
        tree.export_graphviz(clf, out_file=file)
    print(metrics.classification_report(y_test, y_test_pred,target_names=['level
0', 'level 1']))
    return y_test_pred

x_train, x_test, y_train, y_test = train_test_split(
    data[["Tin" , "Door", "detected_motions", "Tout", "humidity"]],
    data[['labelcopy']], test_size=.2, random_state=0)
y_pred_occupancy= decision_tree(x_train, x_test,y_train, y_test,5)
```

```
65|
66| depth=[1,2,3,4, 5, 6, 7, 8, 9, 10, 11]
67| accuracy=[0.34,0.61,0.6,0.69,0.62,0.63, 0.65,0.64, 0.63, 0.69, 0.67]
68|
69| plt.plot(depth,accuracy)
70| plt.show()
71|
72|
73| # Adapt the problem to the project
74|
75|
76|
77|
78|
79|
80|
81|
```

```
# projet_vf.py

## Importations

#import matplotlib.pyplot as plt
#import numpy as np
#from sklearn.model_selection import train_test_split
#from sklearn import tree
#from sklearn import metrics
import pandas as pd
import time
import datetime

## datetime

''' On crée une fonction qui permet de modifier nos fichiers csv afin de retirer la ponctuation qui nous gêne pour extraire les données '''

def stringdate_to_datetime(stringdatetime: str):
    if "." in stringdatetime:
        stringdatetime=stringdatetime.split(".")[0]
    else:
        stringdatetime=stringdatetime.split("Z")[0]
    return datetime.datetime.fromtimestamp(time.mktime(time.strptime(stringdatetime, '%Y-%m-%dT%H:%M:%S')))

## Lecture des fichiers

''' On crée une liste pour chaque capteur à partir de la lecture d'un fichier csv et de la fonction précédente '''

TV_power= pd.read_csv('C:/Users/matth/Documents/Projet 11/TV_power.csv', sep= ',')
TV_power["datetime"]=TV_power["time"].apply(stringdate_to_datetime)

Box_TV= pd.read_csv('C:/Users/matth/Documents/Projet 11/Box_TV.csv', sep= ',')
Box_TV["datetime"]=Box_TV["time"].apply(stringdate_to_datetime)

Bruit= pd.read_csv('C:/Users/matth/Documents/Projet 11/Bruit.csv', sep= ',')
Bruit["datetime"]=Bruit["time"].apply(stringdate_to_datetime)

Lumière= pd.read_csv('C:/Users/matth/Documents/Projet 11/Lumière.csv', sep= ',')
Lumière["datetime"]=Lumière["time"].apply(stringdate_to_datetime)

Luminosité= pd.read_csv('C:/Users/matth/Documents/Projet 11/Luminosité.csv', sep= ',')
Luminosité["datetime"]=Luminosité["time"].apply(stringdate_to_datetime)

Fenêtre= pd.read_csv('C:/Users/matth/Documents/Projet 11/Fenêtre.csv', sep= ',')
Fenêtre["datetime"]=Fenêtre["time"].apply(stringdate_to_datetime)

Humidité= pd.read_csv('C:/Users/matth/Documents/Projet 11/Humidité.csv', sep= ',')
Humidité["datetime"]=Humidité["time"].apply(stringdate_to_datetime)

Température= pd.read_csv('C:/Users/matth/Documents/Projet 11/Température.csv', sep= ',')
Température["datetime"]=Température["time"].apply(stringdate_to_datetime)

CO2= pd.read_csv('C:/Users/matth/Documents/Projet 11/CO2.csv', sep= ',')
CO2["datetime"]=CO2["time"].apply(stringdate_to_datetime)

## fusion des fichiers dans un tableau

''' Ici, on cherche à regrouper les listes précédentes dans un tableau afin de travailler avec un seul objet'''

mergeData=[TV_power,Box_TV,Bruit,Lumière,Luminosité,Fenêtre,Humidité,Température,CO2]
```

```
result=pd.concat(mergeData, axis=1, join='inner')

## Trie des données par demie-heure

''' Le tableau présente des valeurs 'très différentes d'heures. On choisit de
garder les informations toutes les demie-heures '''

TV_power.set_index('datetime',inplace=True)
print("isnull",TV_power.isnull().any())
TV_power.ffill()
newTV_power=TV_power.resample('30T').sum()
#print(newTV_power)

Box_TV.set_index('datetime',inplace=True)
print("isnull",Box_TV.isnull().any())
Box_TV.ffill()
newBox_TV=Box_TV.resample('30T').sum()
#print(newBox_TV)

Bruit.set_index('datetime',inplace=True)
print("isnull",Bruit.isnull().any())
Bruit.ffill()
newBruit=Bruit.resample('30T').sum()
#print(newBruit)


Lumière.set_index('datetime',inplace=True)
print("isnull",Lumière.isnull().any())
Lumière.ffill()
newLumière=Lumière.resample('30T').sum()
#print(newLumière)


Luminosité.set_index('datetime',inplace=True)
print("isnull",Luminosité.isnull().any())
Luminosité.ffill()
newLuminosité=Luminosité.resample('30T').sum()
#print(newLuminosité)


Fenêtre.set_index('datetime',inplace=True)
print("isnull",Fenêtre.isnull().any())
Fenêtre.ffill()
newFenêtre=Fenêtre.resample('30T').sum()
#print(newFenêtre)


Humidité.set_index('datetime',inplace=True)
print("isnull",Humidité.isnull().any())
Humidité.ffill()
newHumidité=Humidité.resample('30T').sum()
#print(newHumidité)


Température.set_index('datetime',inplace=True)
print("isnull",Température.isnull().any())
Température.ffill()
newTempérature=Température.resample('30T').sum()
#print(newTempérature)


CO2.set_index('datetime',inplace=True)
print("isnull",CO2.isnull().any())
CO2.ffill()
newCO2=CO2.resample('30T').sum()
#print(newCO2)

## Fusion des listes triées par demie-heure

''' De même on fusionne les données triées par demie-heure pour ecploiter un seul
```

```
objet par la suite '''
128|
129|
mergeData2=[newTV_power,newBox_TV,newBruit,newLumière,newLuminosité,newFenêtre,newHumi
dité,newTempérature,newCO2]
130| result2=pd.concat(mergeData2, axis=1, join='inner')
131|
132| ## hours
133|
134| ''' On cherche désormais à créer une liste avec les heures de la journée à partir
des valeurs de CO2 dans le but de créer notre liste occupancy '''
135|
136| hours =[]
137| data_datetimes = pd.to_datetime(CO2.iloc[:,0],format='%Y-%m-%dT%H:%M:%S')
138| for dt in data_datetimes:
139|     hours.append(dt.hour)
140| CO2['hour']=hours
141|
142| ## Occupancy
143|
144| ''' On crée la liste occupancy qui fait la même taille que la liste heures. Le
choix des valeurs dépend de nos hypothèses de départ '''
145|
146| occupancy=[]
147| for i in range (len(hours)):
148|     if 7<=hours[i]<=8:
149|         occupancy.append(1)
150|     if 17<=hours[i]<=23:
151|         occupancy.append(1)
152|     if 12<=hours[i]<=13:
153|         occupancy.append(1)
154|     else:
155|         occupancy.append(0)
156|
157| ## Pertes d'énergie
158|
159| ''' on applique maintenant le scénario décrit dans le compte-rendu '''
160|
161| window=newFenêtre["Fenêtre"]
162| Co2=newCO2["CO2"]
163| lyt=newLumière["Lumière"]
164| box_tv=newBox_TV["Box_TV"]
165| tv_power=newTV_power["TV_power"]
166| temperature=newTempérature["Température"]
167| pertedenergie=0
168|
169| for i in range (len(result2)):
170| #    if window[i]==1 and Co2[i]<3000 and :
171| #        print('pertes d énegie car on l air n est pas sain',hours[i],'h')
172|     if lyt[i]==1 and occupancy[i]==0:
173|         pertedenergie +=1
174|         print('La lumiere est allumee mais il n y a personne',hours[i],'h')
175|     if tv_power[i]!=0 and box_tv[i]!=0 and occupancy[i]==0:
176|         pertedenergie +=1
177|         print('La télé et la box sont allumées mais il n y a personne
à',hours[i],'h')
178|     if temperature[i]<20 and window[i]==1:
179|         pertedenergie +=1
180|         print('on devrait fermer la fenêtre pour ne pasa chauffer l extérieur
à',hours[i],'h')
181|
182| print ("Il y a ", pertedenergie, "occurences de perte d'énergie en une journée")
```