RAPPORT PROJET INFORMATIQUE

SUJET 1

* Construction du programme

Afin d’analyser les données issues de campagnes de mesure du bâtiment, le programme s’est construit autour des bibliothèques et modules suivants :

* Numpy
* Matplotlib
* Statistics
* Scipy

La bibliothèque Panda n’a pas été exploitée. Sans cela, il a fallu écrire des programmes étant capables de récupérer les données du fichier CSV et construire des algorithmes pouvant les exploiter. En effet, la récupération des données donne en sortie un format Tableau, ce qui est facilement exploitable avec les opérations classiques vues en CPGE (traitement avec Numpy).

La taille du tableau a amené une grande attention quant à l’exactitude des codes. Il est arrivé durant le travail que des problèmes de taille (erreurs de type Overflow) liés à un code, aient eu des répercussions sur d’autres codes, et ce bien plus tard dans le script.

Le programme s’articule selon plusieurs étapes interdépendantes. Il répond à la problématique posée en appelant des fonctions écrites dans les étapes précédentes. Il est capable d’afficher les courbes montrant l’évolution d’un paramètre en fonction du temps, d’afficher les valeurs statistiques sur la courbe, calculer l’indice humidex, l’indice de corrélation entre deux variables (et les afficher sur une courbe)

* Etape 1 : Récupérer les données contenues dans le fichier

Ce programme provient d’un forum Python (<https://openclassrooms.com/forum/sujet/convertir-fichier-tableur-csv-en-liste-sur-python>). Comme expliqué sur le site, il prend le chemin d’accès du fichier en paramètre. Il implémente ensuite deux tableaux vides puis ouvre et lit le fichier CSV. Il ajoute dans l’un d’eux chaque ligne du tableur sous forme de liste de listes (exemple d’une liste dans la liste : ['5;27.0;24.8;59.5;0;461;2020-09-16 06:45:28 +0200']). Enfin, il récupère chaque sous-liste et l’ajoute dans le second tableau précédemment implémenté. Le tableau final [1] est retourné avec toutes les cellules d’une ligne dans une chaîne de caractères.

* Etape 2 : Transformer la chaîne de caractère en liste

Un fichier CSV voit ses données séparées par un point-virgule. La deuxième étape consiste à enlever ces éléments dans le tableau obtenu précédemment.

L’algorithme prend en argument le tableau [1]. Il consiste à implémenter un tableau [2] de la taille du tableau précédent. En parcourant chacun des éléments du tableau [1], il remplace chaque élément du tableau implémenté [2] par la chaîne de caractère correspondante en enlevant les points virgules. Python sépare donc automatiquement chaque cellule et les laisse dans une sous-liste qui représente une ligne du fichier CSV. Il retourne le tableau [2].

Le second algorithme reprend le tableau [2] en argument, implémente une liste [3] vide. Il élimine la première ligne du fichier CSV car elle ne contient que les titres des colonnes. Il parcourt ensuite chaque ligne, et pour chaque cellule, il ajoute dans la liste [3] la chaîne de caractère transformée en un élément de type Float, tout en la laissant dans sa ligne. Cette opération est nécessaire pour exploiter les données : les courbes fonctionnent avec des valeurs, Python ne considère pas une liste de caractère comme un nombre même écrite comme tel. Il reste à séparer les valeurs selon leur domaine (Identifiant, Température…) ; afin de simplifier les futures opérations, il convient de concaténer toutes les sous listes de la liste [3].

Le troisième algorithme prend donc en paramètre la liste [3], implémente une liste [4] vide. En parcourant chaque élément de chaque sous liste de la liste [3], il ajoute cet élément dans l’unique liste [4].

* Etape 3 : extraire les différents types de données

Sachant que l’on connaît le nombre de données différentes (dans notre cas six : Identifiant - Bruit – Température – Humidité – Luminosité – CO2), il est alors possible de les séparer dans des tableaux différents. Cette étape ne permet pas d’extraire les données temporelles car elles ne correspondent pas à des nombres.

Pour chaque donnée, l’algorithme est le même. Il prend en paramètre la liste [4] et implémente une liste [5] vide. Puis, l’algorithme parcourt la liste et avec un pas de six (nombre de données différentes) et commence son parcours entre la première et la sixième valeur de la liste [4] selon la donnée (par exemple, l’humidité est en quatrième position donc on ajoutera les valeurs des positions 10, 16,22…). Il ajoute alors ces valeurs à la liste [5] et la retourne.

* Etape 4 : extraire les données de temps

Cette partie consiste à enlever les secondes et la partie « +0200 » de la date. Cela permet d'alléger la lecture des tableaux lorsque l’on effectue des vérifications.

Le premier algorithme prend en argument le tableau [2] et implémente une liste [6] vide. Il parcourt le tableau et pour chaque élément, il récupère la date, les heures et minutes. Il retourne la liste [6].

* Etape 5 : afficher les courbes en spécifiant un intervalle de temps

Cette partie a pour but d’afficher les courbes des différentes données selon le capteur voulu. On peut également spécifier un intervalle de temps, si on rentre les dates sous le format approprié.

Le premier algorithme sert à trier la liste des capteurs, chaque ligne correspondant à une mesure de l’un d’eux à une date donnée. Sans cette étape, on obtiendrait in fine une courbe avec le même intervalle de temps qui se répète (pour les capteurs 1 à 5) ainsi que l’intervalle de temps du capteur 6. Il prend en argument le numéro du capteur dont on veut les données et implémente une liste [7] vide. Il parcourt la liste des index obtenue par la liste [5]. Pour chaque élément de cette liste, on pose la valeur de son index (ici l’index en tant que position dans la liste). Si l’élément de la liste correspond au numéro du capteur rentré, on ajoute son index à la liste, et enfin retourne la longueur de celle-ci.

Le second algorithme est celui qui va afficher les courbes. Il prend en argument le paramètre désiré ainsi que le numéro du capteur que l’on souhaite étudier. Le programme Python interagit avec l’utilisateur qui doit entrer les dates de début et fin dans la commande sous le format [2020-09-11]. La première partie de l’algorithme consiste à sélectionner les données du capteur choisi à l’aide du premier algorithme. La liste des capteurs est ordonnée (1,1,...1,2,...2,3,...) ; la longueur de la liste de chaque capteur permet de relever le capteur désiré en parcourant la liste jusqu’à le trouver. Tant qu’il ne l’est pas, il ajoute à une valeur implémentée à 0 la longueur de la liste du capteur analysé par le programme. On peut alors deviner l’indice de début et fin des données du paramètre dans leur liste, qui correspondent au capteur choisi.

Ensuite, on implémente une liste [8] avec les données temporelles (issues de la liste [6]) et une liste [9] avec les données du paramètre. Ces listes contiennent les données du capteur choisi. On implémente trois nouvelles listes vides [10] (données temporelles), [11] (données du paramètre), et [12]. Cette dernière servira de compteur : on aurait pu mettre une variable à la place, mais cela permet une répétition dans le programme ce qui offre une meilleure visibilité quant aux erreurs même si cela prend plus de place en mémoire vive. Ensuite, on sélectionne les données correspondant aux plages horaires entrées précédemment, à l’aide de plusieurs boucles :

* La première est une boucle **for** avec condition **if** : pour chaque élément de la liste [8], si la date de début entrée correspond à celle de l’élément, alors on l’ajoute (l’élément entier, c’est-à-dire la date, les heures et minutes) convertit au type *datetime* dans la liste [10]. On ajoute ensuite l’élément de la liste [11] correspondant au même index pour relever le paramètre. Si la condition est remplie,on ajoute le numéro de l’itération à la liste du compteur [12].
* Dans le cas où on veut les données sur une seule date, on prévoit une condition (**if**). Si tel est le cas, on affiche directement la courbe du paramètre en fonction du temps en prenant comme valeurs les éléments de la liste [10] (abscisses) et ceux de la liste [11] (ordonnées).
* Dans le cas contraire, (**else**), le compteur intervient maintenant. On récupère le dernier élément de sa liste ce qui permet de savoir quelles données restent à ajouter dans les listes [10] et [11]. Tant que (**while**) chaque élément de la liste [8] a une date différente de la date de fin entrée, le programme ajoute les éléments de la même manière que pour la première boucle **for**, en ajoutant la valeur 1 au compteur pour passer à l’élément suivant. A noter que le compteur est désormais une variable.
* Enfin, toujours dans le même cas que le précédent, on parcourt la liste (avec une boucle **for**) à partir du dernier élément ajouté jusqu’à la fin de la liste. Si (**if**) l’élément de la liste [8] possède la même date que celle entrée de fin, alors il ajoute les éléments de la même manière que la première boucle **for**. Après cela, il affiche la courbe de la même manière que cité précédemment.
* Etape 6 : afficher les valeurs statistiques

A ce stade, on peut afficher les valeurs statistiques sur la courbe, de telle sorte qu’on obtient une droite horizontale ayant pour abscisse la valeur statistique souhaitée. Il y a également le calcul de l’indice humidex et celui des indices de corrélation de type Pearson, Spearman et Kendall. Le coefficient de corrélation de Pearson permet d'analyser les relations linéaires et le **coefficient de corrélation de Spearman** les relations non-linéaires monotones.

L’algorithme possède la même structure que le second de la précédente partie. Il prend en argument le numéro du capteur, le paramètre souhaité, et un second paramètre. Voici les différences :

* On interagit avec l’utilisateur en lui demandant quel paramètre il souhaite afficher. Pour ne pas faire planter le programme, on propose l’orthographe des fonctions proposées dans l’**input**.
* Pour le calcul des coefficients de corrélation, on a besoin d’implémenter une liste supplémentaire [13] égale par défaut à une liste [14] comprenant une valeur quelconque (cela n’a pas une grande importance, on aurait pu mettre une variable quelconque mais c’est juste pour n’avoir à manipuler que des listes). Si (**if**) l’utilisateur n’entre pas de paramètre, le programme prend la liste par défaut et la multiplie par la longueur de la liste de l’unique paramètre entré. Sinon (**else**), le programme implémente une liste [14 bis] contenant les valeurs du second paramètre de la même façon que le programme précédent pour les listes [8] et [9].
* La boucle **for** reste inchangée, on ajoute seulement une étape pour la liste [14] ou [14 bis] (c’est l’étape qui ajoute les valeurs du second paramètre par rapport à l’intervalle de temps).
* [*Calcul de l’indice humidex*] : dans le cas (**if**) où l'on souhaite l’indice sur une seule date, le calcul de l’indice humidex s’opère. Le programme récupère les valeurs de température et d’humidité de la même manière que précédemment. Puis, il implémente deux listes [15] et [16] et opère de la même manière que la boucle **for**. Il récupère ensuite la moyenne des valeurs de température et d’humidité contenues dans les listes [15] et [16]. Enfin, il calcule l’indice avec la formule et l’affiche dans la console.

On pourrait laisser le calcul à la fin avec tous les autres mais en procédant de la sorte, cela permet au programme d’éviter des opérations inutiles.

* Si tel n’est pas le cas (**else**), la boucle est la même que dans le programme précédent. De même avec la condition (**if**) qui suit.
* [*Calcul de l’indice humidex*] : si on a choisi une plage horaire, le programme effectue la même chose qu’à l’autre étape de calcul.
* Sinon, si (**elif**) l’on veut afficher [*l’indice de corrélation]*, le programme implémente deux listes avec les données de chaque paramètre. Il les affiche ensuite ainsi que l’indice de corrélation sur une courbe ayant pour abscisse les données temporelles. On affiche également la valeur de l’indice dans la console.
* Sinon (**else**), on affiche enfin la courbe avec les données du paramètre en fonction du temps ainsi que la fonction souhaitée. Elle apparaît sous la forme d’une droite horizontale dont l'abscisse correspond à sa valeur. Dans le cas où l’on entre une fonction erronée, le programme retourne un message indiquant cette erreur avec un **print**.

* Etape 7 : calcul de l’indice de corrélation

Le calcul de l’indice de corrélation est en fait un calcul de deux corrélations comme expliqué précédemment (un pour les relations linéaires l’autre pour les relations non linéaires). Il se fait à l’aide d’une fonction de la librairie “scipy.stats”.

* Etape 8 : réponse à la problématique du sujet

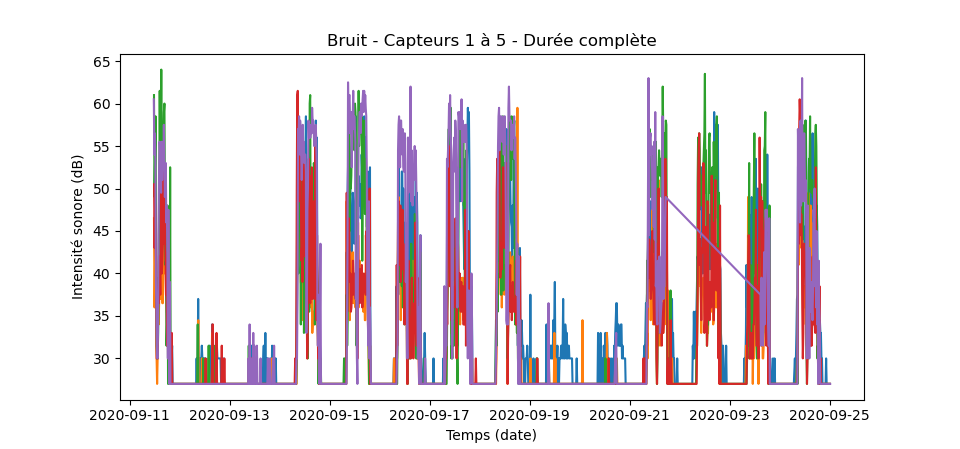
Afin de relever les anomalies dans les données, on utilise les programmes précédents ainsi que deux derniers programmes permettant d’avoir les courbes de chaque capteur sur une plage horaire souhaitée.

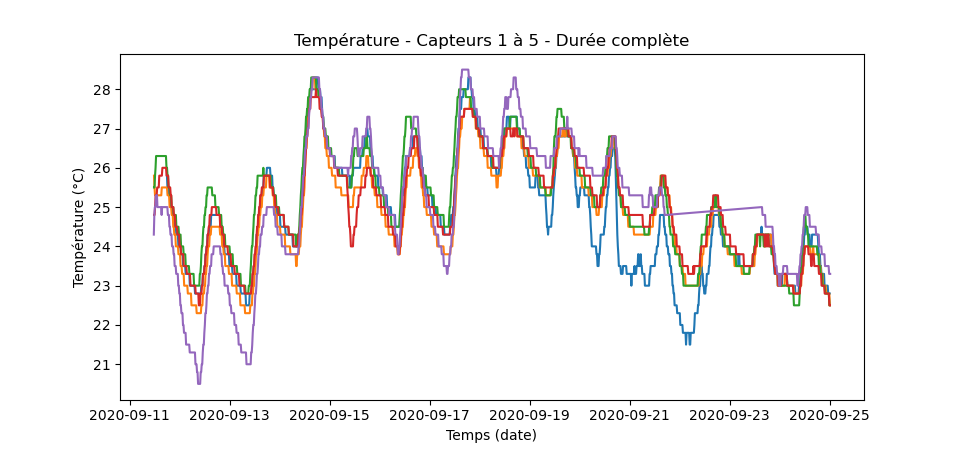
Le premier algorithme est presque exactement le même que celui permettant d’afficher les courbes à l’étape 5 (cf. la fonction **affichercourbe**). La seule différence se trouve dans les arguments : au lieu de mettre un **input**, le programme prend comme argument les dates de début et fin. En effet, si on prenait l’algorithme de la partie 5 tel quel, cela poserait problème par la suite.

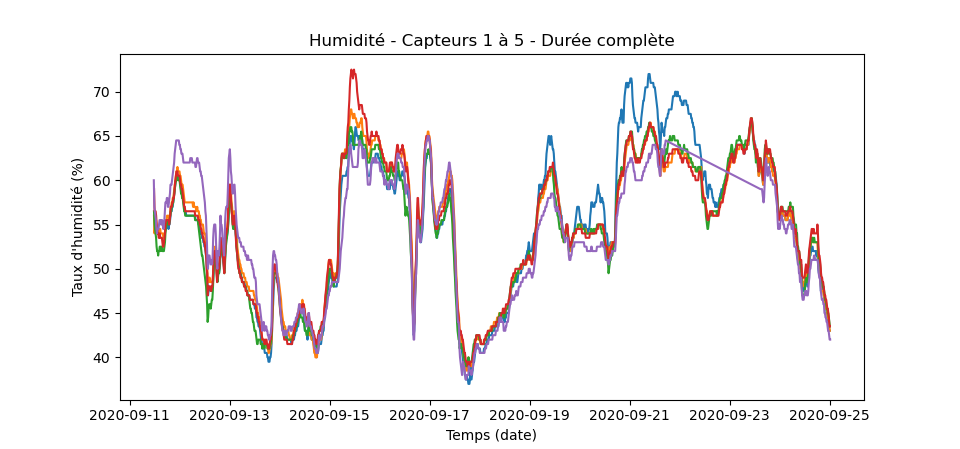
Le deuxième algorithme prend en argument le paramètre dont on souhaite voir les valeurs sur tous les capteurs. Le programme interagit avec l’utilisateur pour entrer la plage horaire désirée. Puis, il utilise le premier algorithme en lui donnant comme argument le paramètre, le numéro du capteur (on utilise l’algorithme pour chaque capteur du 1 au 5), et les dates de début et fin. Il retourne donc les courbes du paramètre sur tous les capteurs.

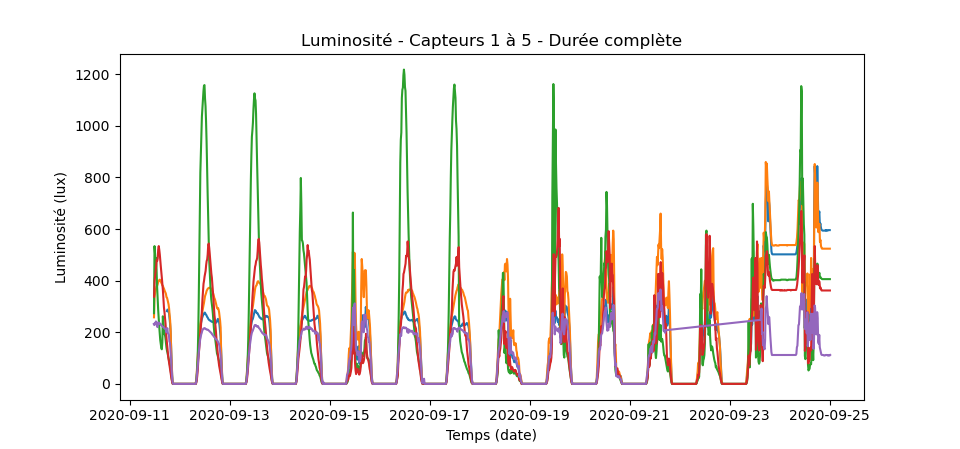
On ne prend pas en compte le capteur 6 car sa période de service diffère de celle des autres capteurs, ce qui constitue une anomalie. Cependant, l’allure des courbes correspondant à ce capteur est la même que celles des autres capteurs. Pour afficher les valeurs sur ce capteur, on utilise le programme de l’étape 5.

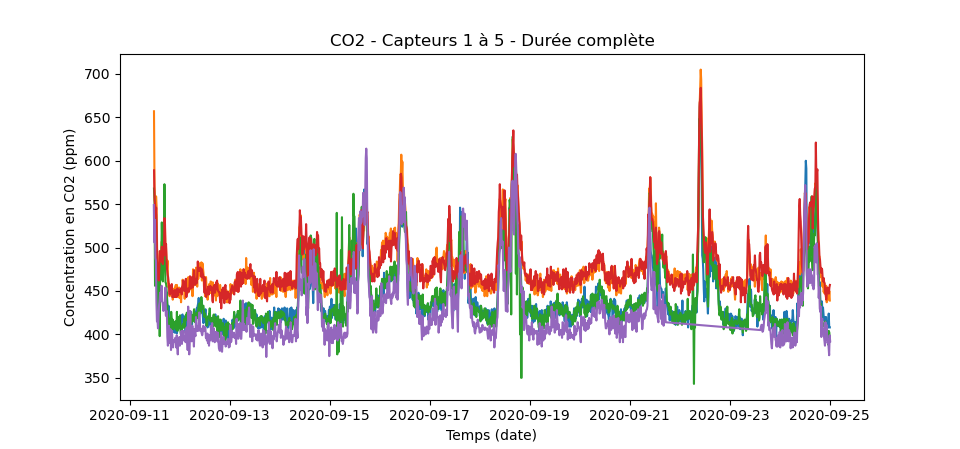
Voici les résultats pour chaque paramètre sur leur plage horaire totale (hormis le capteur 6) :











On remarque plusieurs anomalies :

1. La courbe violette correspondant au capteur 5 semble avoir un problème entre le 21 Septembre 2019 vers 16h et le 23 Septembre vers 16h également. En effet, il manque des données sur toute la plage horaire pour tous les paramètres.
2. Le capteur 3 détecte une plus grande intensité lumineuse que les autres entre le 12 Septembre 2020 et le 14 Septembre 2020, entre le 16 et le 17, entre le 19 et le 20, et le 24. A noter qu’entre le 23 et le 24, tous les capteurs (sauf le capteur 5) ont vu leur valeur de luminosité augmenter plus que la normale. La norme stipule que la luminosité dans les bâtiments doit être d’environ 750 lux au maximum, alors que le capteur 3 a détecté jusqu’à 1200 lux.

On remarque également que le capteur 1 ne commence qu’à mesurer à partir de 18h le 11 Septembre 2020 alors que les autres commencent dès 12h.

1. D’après la norme, le taux d’humidité doit être compris entre 40% et 70% ce qui est le cas sur toute la plage horaire, sauf pour le capteur 1 entre le 20 et le 22 et le capteur 4 au 15 Septembre où il a dépassé les 70%. Le taux a également chuté en dessous de 40% pour tous les capteurs le 17 Septembre.
2. A propos du capteur 6, on a dit que ses courbes étaient semblables à celles des autres capteurs. On peut supposer que le capteur a implémenté les mauvaises dates lors de ses mesures.

Afin de détecter les anomalies de mesure (c'est-à-dire si l’écart relatif entre deux mesures est trop important), on pose un dernier algorithme. Il prend en argument le paramètre dont on cherche les données et le numéro du capteur à analyser. Il sélectionne les données du capteur de la même manière que précédemment.

Il implémente deux listes qui vont récupérer les données de temps et du paramètre dont les écarts relatifs entre la valeur de la donnée du paramètre et la valeur précédente dépasse 50%, sauf si (**if** puis **else**) le paramètre considéré est la lumière : l’écart relatif doit être dépassé de 100% car il y a des valeurs qui sont nulles. En effet, ce n’est pas une erreur, les détecteurs ne mesurent pas de lumière lorsque les locaux ne sont pas occupés. Si tel est le cas, le calcul de l’écart relatif sera biaisé car on obtiendrait 1 dès que , avec la valeur considérée par l’algorithme et la valeur précédente.

Cela veut aussi dire que durant le calcul, si la valeur précédente est nulle, il y a une erreur car on ne peut pas diviser par 0 (, avec la valeur considérée par l’algorithme et la valeur précédente qui serait égale à 0). Ainsi, on additionne au dénominateur un nombre très petit (par exemple , on calcule donc )afin d’esquiver l’erreur en ne modifiant quasiment pas le résultat. La vérification se fait avec **if**.

Dans le cas où il y aurait une valeur nulle dans d’autres paramètres, on conserve la formule corrigée de l’écart relatif pour les autres paramètres mais on laisse la majoration de l’écart relatif à 50%.

Ensuite, on affiche dans la console un tableau (dont la première colonne contient les données temporelles, la seconde contient les valeurs des anomalies). Enfin on affiche la courbe du paramètre (contenant toutes ses données) par rapport au temps. On intègre une boucle **for** qui pour chaque anomalie l’affiche sur la courbe à travers une barre verticale.

Quant aux **horaires d’occupation des bureaux**, on peut le deviner grâce aux courbes. En effet, elles affichent des courbes ayant sensiblement la même forme, et ces courbes possèdent un certain intervalle creux qui se répète. La courbe la plus explicite est celle de la luminosité : le capteur détecte logiquement lorsque la lumière est éteinte ce qui permet de savoir que personne n’est censé se trouver dans le bâtiment dans cette plage horaire. On relève les plages suivantes pour différents intervalles creux (entre la dernière trace de lumière le soir et la première trace le lendemain) : [20h50-07h20] ; [20h50-07h30] ; [20h50-07h30] ; [21h22-07h34] ; [20h49-07h32]...

On en déduit que les horaires d’occupation des bureaux sont plus ou moins [7h30-21h].