**Rapport explicatif du projet de programmation**

Groupe 15 : Lucie Geulin et Marion Auriel

Sujet 2

* **Introduction** :

L’objectif de ce projet était de **déterminer les similarités entre différents capteurs à partir de mesures contenues dans un fichier CSV.**

* **Méthode de travail :**

Nous avons au départ rencontré une difficulté pour travailler de façon coordonnée car nous n’arrivions pas à utiliser Github correctement. Nous avons donc utilisé pendant plusieurs séances un Google Drive dans lequel nous déposions les nouvelles versions de notre programme. Deux semaines avant le rendu du projet, nous avons pris le temps de comprendre comment utiliser Github et avons donc commencé à l’utiliser pour partager nos avancées sur le projet. Nous avons quand même déposé toutes les versions antérieures de notre programme (que nous avions mises au fur et à mesure sur Google Drive) sur Github afin de vous montrer l’évolution de notre programme.

Pour que notre programme et notre rapport final ne soient pas perdus parmi tous ces fichiers, nous avons créé un dossier dans le répertoire contenant uniquement le rapport et le programme final.

* **Démarche :**

Afin de répondre à cette problématique, nous avons tout d’abord cherché à importer le document CSV sur Python puis nous avons récupéré les données de ce fichier dans une liste de listes. (1)

L’objectif du programme étant de comparer les similarités des capteurs, nous avons séparé les données par capteur. (2)

Dans la continuité de l’étape précédente, nous avons séparé les caractéristiques (noise, temperature, humidity, lum, CO2, date) issues de chaque capteur dans des listes différentes. (3)

Ensuite, nous avons programmé toutes les fonctions demandées par le sujet. (4)

Puis, nous avons tracé les courbes représentant une caractéristique donnée pour chaque capteur, ainsi que ses valeurs statistiques. (5)

Nous nous sommes ensuite attelées au bonus, demandant la représentation des courbes, en fonction de dates spécifiques. (6)

Enfin, nous avons programmé une fonction capable d’afficher si deux capteurs étaient similaires par rapport à une caractéristique (bruit, lumière, …) (7)

Finalement, nous avons cherché à déterminer l’occupation des bureaux. (8)

* **Programme :**

1. **Récupération des données contenues dans le fichier .csv.**

La première difficulté rencontrée a été de récupérer les données contenues dans le fichier csv. Pour cela, nous avons d’abord transformer le fichier .csv en un dossier .txt directement en enregistrant ce dernier sur notre ordinateur. Nous avons fait ce choix car nous sommes plus à l’aise avec ce type de fichier.

Nous avons donc créé une liste pour chaque ligne. Après avoir observé comment le fichier .txt était formé, nous avons compris que le séparateur : ’ \t’, permettait de séparer les différentes lignes du fichier. Nous avons donc utilisé la commande split(‘\t’), afin de séparer les différentes lignes. Chaque ligne a donc été insérée dans une nouvelle liste. La liste *donnees*, est donc une liste contenant chaque ligne dans une liste.

1. **Séparation des données par capteurs.**

La deuxième étape fut de séparer dans des listes différentes, les données par capteurs. Pour cela, il a suffi de tester pour chaque liste de la liste *donnees*, quel était le premier élément. En effet, si 1 était inscrit, alors, on récupérait la liste et on l’ajouter dans une nouvelle liste nommée c1. Idem pour les autres capteurs.

1. **Séparation par caractéristiques des données de chaque capteur.**

De la même façon que précédemment, nous avons récupéré pour chaque capteur les différentes caractéristiques dans des listes différentes. Pour cela, nous avons utilisé l’indice de position des données dans les différentes listes. En effet, chaque donnée de même type se trouvait à la même position dans les différentes listes. Par exemple, le bruit se trouvait en première position de chaque liste capteur.

Pour la liste des temps, un « +0200\n » était présent après chaque date. Nous n’avions pas besoin de ces caractères pour notre étude donc nous les avons retirés lors de la création des listes comportant les dates et heures.

1. **Programmation des différentes fonctions demandées.**

* **Valeurs statistiques :**

Nous avons tout d’abord écrit de façon classique les fonctions calculant les différentes valeurs statistiques d’une liste (maximum, minimum, écart-type, moyenne, variance et médiane). Pour le calcul de la moyenne, nous avons décidé de choisir la moyenne arithmétique classique car elle nous semblait être la plus parlante et la plus pertinente. Pour le calcul de la médiane, nous avons dû au préalable trier la liste par ordre croissant. Nous avons utilisé pour cela un algorithme de tri par insertion car cet algorithme est efficace sur des listes de la taille de celles traitées dans notre projet.

Nous avons ensuite écrit des programmes renvoyant chacun une courbe ou un point légendé(e) correspondant à la valeur statistique associée.

* **Indice humidex :**

Nous avons trouvé après des recherches les formules suivantes pour calculer l’indice humidex :

Où T correspond à la température mesurée et Trosée le point de rosée, calculé comme ceci :

avec où correspond à l’humidité relative, T à la température, a=17,27 et b=237,7.

Nous avons donc transcrit ses formules sur Python pour accéder à l’indice humidex d’un couple humidité relative/température.

* **Coefficient de corrélation :**

Nous avons trouvé que l’indice de corrélation pouvait être calculé de la façon suivante :

où et correspondent aux écarts-types et cov(X,Y) correspond à la covariance de X et de Y.

Cette dernière se calcule de la façon suivante :

avec N le nombre de valeurs des listes X et Y (supposées de même longueur), et et les moyennes des listes X et Y.

Nous avons donc encore une fois écrit ces formules sur Python afin d’accéder au coefficient de corrélation de deux listes.

1. **Tracé des courbes en fonction du temps.**

Une première difficulté rencontrée a été de choisir quelle base de temps nous souhaitions prendre. Après observation de la liste des dates, nous avons remarqué que les mesures étaient réalisées à intervalle de temps régulier (toutes les 15 minutes). Nous avons donc au départ choisi de ne pas nous soucier du temps. Nous prendrions en abscisse une liste contenant le nombre de mesures réalisées.

Cependant, cette technique ne nous permettait pas de comparer sur un même graphique les données de deux capteurs différents, car les mesures n’étaient pas toujours réalisées au même instant. Nous avons donc décidé de convertir chaque date en seconde. La référence prise serait 0 pour la première date à 0h00. Les mesures étant réalisées entre le 11 août 2019 et le 25 août 2019, (mesures réalisées durant un même mois), il a été ajouté 24h converties en secondes, pour chaque nouvelle journée passée par rapport à la référence (le 11 août).

Ensuite, nous avons utilisé la commande *matplotlib.pyplot.plot(temps, données)*, afin de tracer les courbes demandées.

Nous avons ensuite modifié l’axe des abscisses afin qu’il affiche les jours correspondant plutôt que leur « nombre » de secondes qui était automatiquement affiché mais qui n’était pas très parlant.

1. **Option : Intervalle de temps spécifiques.**

Tout d’abord, nous avons utilisé la commande :’input’, afin que le serveur nous demande de rentrer l’intervalle de temps souhaité.

Afin de restreindre, le liste des temps aux dates spécifiées, nous avons créé une nouvelle liste des temps nommée : *c1\_date* (resp. c2,..). Nous avons utilisé une compression de liste. En parcourant la liste de départ, nous n’avons ajouté à la nouvelle liste uniquement les dates contenues dans l’intervalle demandé.

Afin que les dates demandées correspondent aux dates de mesures des capteurs, nous avons utilisé la commande assert. Cette dernière vérifie la validité du champ entré par l’utilisateur, ici la date devait être comprise entre le 11 août 2019 et le 25 août 2019.

1. **Affichage des similarités.**

L’affichage des similarités a évolué suite aux échanges avec les professeurs.  Au départ, nous souhaitions que le programme indique visuellement les similarités. Ce premier programme comparait les similarités à 95% de deux capteurs vis-à-vis de la moyenne, la médiane, l’écart-type, la variance, le coefficient de corrélation ou encore l’indice humidex.

Nous avions utilisé la méthode .format afin d’afficher les différentes similarités.

Cependant, ces similarités n’étaient pas représentatives de la réalité, car deux capteurs pouvaient obtenir une même moyenne avec des valeurs totalement disparates par exemple.

Nous avons donc cherché à calculer la distance entre la moyenne des mesures sur un intervalle de temps d’une heure pour des caractéristiques identiques. En effet, nous avons remarqué que la valeur des mesures changeait de manière significative au bout d’une heure.

Pour cela nous avons procédé en plusieurs étapes.

Pour cela, nous avons tout d’abord réalisé une liste : *base\_temps* nous permettant de faire des recherches de points dans un même intervalle pour tous les capteurs. Nous avons ensuite récupéré les indices des points dans cet intervalle de temps. Nous avons utilisé ces indices pour découper nos listes de caractéristiques dans ces intervalles. Enfin, nous avons déterminé la moyenne des données dans cet intervalle afin de pouvoir calculer la distance moyenne entre deux courbes sur un même intervalle de temps.

Nous avons déterminé la distance moyenne de la totalité de la liste.

Cependant, une fois encore, la moyenne n’était pas l’outil le plus adéquat pour comparer les similarités. Nous nous sommes donc orientés vers un calcul de variance autour de la moyenne, mais nous ne savions pas quelle référence paraissait pertinente au vu de cette variance.

Finalement, nous avons choisi de travailler avec un calcul de métrique relative aux distances. Pour cela nous avons cherché à normaliser les distances obtenues pour chaque capteur.

Afin de normaliser nos valeurs, nous avons utilisé la formule suivante :

Soit X une liste

i∊|0,lenX|

Le maximum et le minimum sont déterminés sur un même intervalle, et en comparant tous les capteurs (c1/c2, c1/c3…) (le minimum est différent sur le premier intervalle de temps et le deuxième par exemple).

Après avoir normalisé toutes nos listes, nous avons pu déterminer la distance moyenne en pourcentage entre tous les capteurs pour une même caractéristique. Nous avons pu en déduire que les capteurs avec les plus faibles distances moyennes étaient les plus similaires.

1. **Bonus : Programme de détermination de l’occupation des bureaux.**

Nous avons séparé ce programme en deux parties. En effet, afin d’avoir les moments où les bureaux sont occupés, nous avions besoin de connaître les jours correspondant aux week-ends, et de connaître les horaires des bureaux.

* **Horaires d’occupation des bureaux :**

Pour trouver les horaires des bureaux, nous avons choisi d’étudier la luminosité de ceux-ci car cette donnée nous semblait la plus pertinente. En effet, le bruit, la température, le taux d’humidité et le CO2 ne dépendent pas uniquement de la présence de personnes dans les bureaux alors que la lumière est nécessaire au travail.

Pour comparer la luminosité à différents moments de la journée, nous devions trouver un seuil permettant de trancher sur la possibilité de travailler ou non avec cette quantité de lumière. D’après différentes sources, la luminosité dans les lieux de travail est généralement supérieure à 150 lux. Nous avons donc pris cette valeur comme référence : lorsque la luminosité est supérieure à cette valeur, nous considérons que les bureaux sont occupés.

Nous avons écrit dans un premier temps un programme qui renvoie la liste des horaires pour lesquels la luminosité est supérieure à ce seuil de 150 lux. Cette liste correspond donc aux horaires d’occupation des bureaux pour un capteur. Nous avons ensuite fait évoluer ce programme pour qu’il renvoie uniquement l’heure de début et de fin d’occupation des bureaux chaque journée.

Le programme renvoie des listes cohérentes pour les capteurs 1 et 3. Cependant, on remarque que pour les autres capteurs (2,4,5 et 6), la luminosité ne diminue pas les deux dernières nuits. Elles sont donc comptabilisées par notre programme comme des horaires de travail. Cette incohérence peut être due à un oubli d’éteindre la lumière deux nuits de suite, ou à un mauvais fonctionnement des capteurs. Cette deuxième option nous semble plus probable car les valeurs de luminosité pour ces mêmes capteurs lors des deux derniers jours sont beaucoup plus importantes que celles au début des mesures (par exemple avec le capteur 6, le 14 août à 10h30 la luminosité était de 272 lux alors qu’elle est de 728 lux le 25 août à la même heure).

* **Détermination des jours de la semaine :**

Afin de déterminer les jours d’occupation des bureaux, nous avons utilisé la donnée du bruit car elle variait de façon significative 2 jours sur 7, ce que nous avons interprété comme la variation due aux week-ends.

Nous avons écrit tout d’abord un programme qui sépare les mesures du bruit par journée pour un capteur donné : il renvoie une liste de sous-listes contenant chacune les mesures du bruit pour une journée. Ce programme a été utilisé pour notre deuxième programme : celui-ci renvoie une liste contenant les moyennes du bruit pour chaque jour pour un capteur donné. Enfin, ces deux programmes ont été utilisés dans notre programme final, qui nous renvoie une liste contenant les jours de la semaine et une autre les jours du week-end. Pour cela, nous avons décidé que si la moyenne du bruit pendant une journée était supérieure à 29,5 décibels, cette journée était en semaine tandis que si elle était inférieure à 29,5 décibels, la journée tombait un week-end.

Enfin, nous avons rassemblé ces deux programmes (occupation\_bu et weekend) dans un programme (horaire\_sem). Ce dernier permet de retirer de la liste renvoyée par occupation\_bu les jours retenus comme étant les samedis et dimanche grâce au programme weekend. Il renvoie donc une liste présentant pour chaque jour de la semaine l’heure de début et de fin de travail.

* **Conclusion :**
* Similarités
* Apprentissage de l’utilisation de nouveaux logiciels comme Github
* Entraînement pour le projet data