Rapport Python

Le but du projet Python était de nous faire travailler le design d’algorithme, leur implémentation et l’utilisation d’outil de gestion de code informatique. Notre sujet consistait à trouver des anomalies dans les données fournies, issues d’une campagne de mesure au sein d’un bâtiment de bureau. Les données sont proposées par Kandu et sont composées entre autres de la mesure de la température ambiante (°C), de l’humidité relative (%), du niveau sonore (dBA), du niveau lumineux (lux), de la quantité de CO2 (ppm).

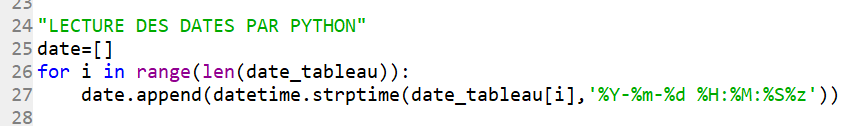


Dans la première partie de l’algorithme, le but était de lire les données CSV. Pour cela, nous avons utilisé la bibliothèque panda (l.8) en utilisant la fonction « pd.read » (l.15) et ouvert notre fichier CSV nommé : EIVP\_KM.csv. Chaque colonne du tableau étant séparé d’un « ; » nous avons décidé de créer une séparation des colonnes du tableau par ce signe. Dans un premier temps, nous avions créé une séparation à partir du nom des colonnes, mais l’algorithme a rencontré des difficultés à reconnaitre le nom de nos variables.

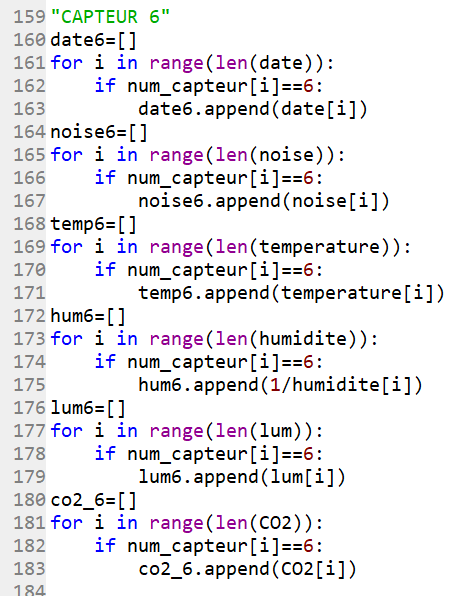
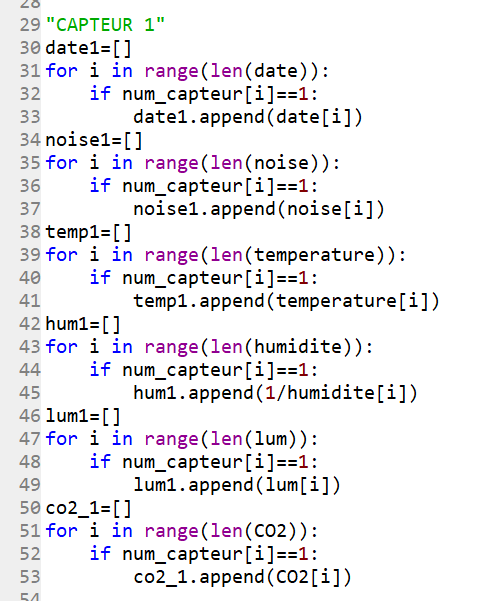
Pour créer notre graphe, nous devions obtenir des listes composées des différentes valeurs de chacune des colonnes. Dans un premier temps, nous avons voulu utiliser la fonction Dataframe de la bibliothèque. Finalement, nous avons opté pour l’utilisation de la fonction tolist (l.16 à l.22), car cette dernière nous semblait plus intuitive ; ainsi dans chaque liste se trouvent les données de tous les capteurs. Nous avons donc dû remédier à deux problèmes : le premier est le fait que les dates se trouvant dans la colonne, n’étaient pas reconnaissables par Python ; le deuxième est le fait qu’aucune des listes n’étaient séparées par capteur.

Par conséquent, le graphe créé reprenait les valeurs de chaque capteur ce qui le rendait illisible.

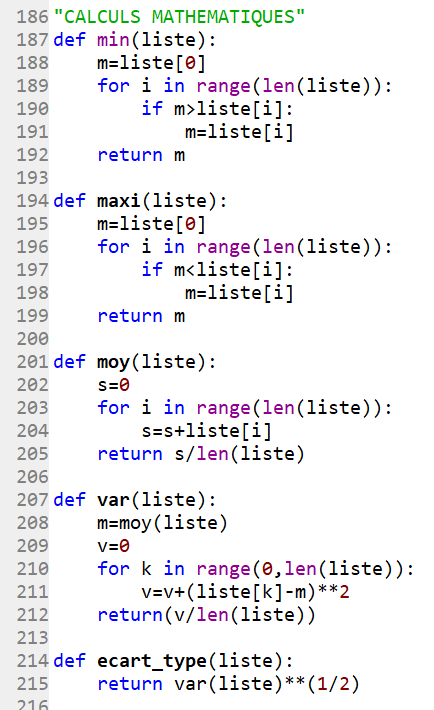
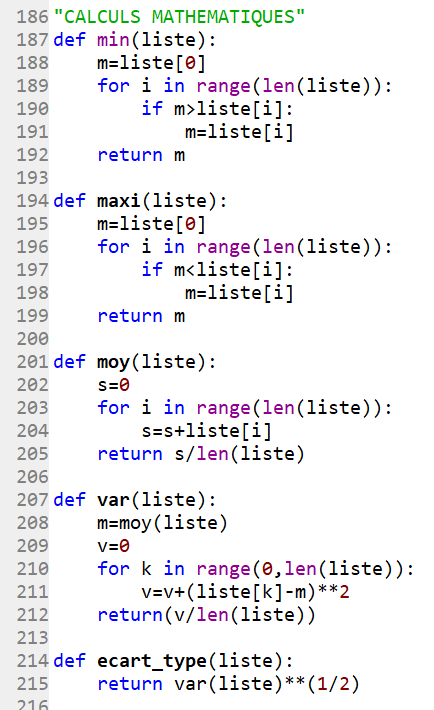


Pour remédier au problème de la date, nous avons importé la bibliothèque datetime (l.9) et créé une liste vide. Ainsi, nous avons transformé, les données de la colonne send\_at du tableau CSV (l.17) en des données pouvant être lus et traitées correctement par Python. Puis nous les avons ajoutées à la liste vide pour obtenir une liste de date fonctionnelle (l.25 à l.27).

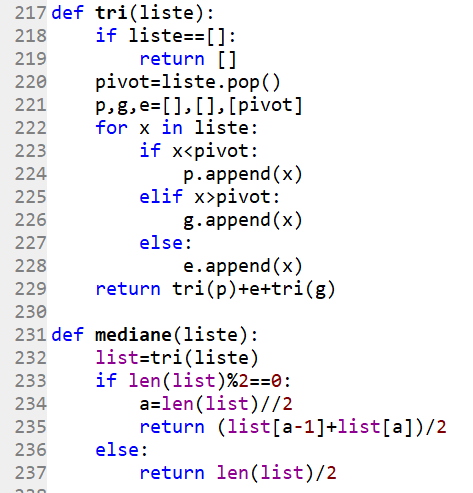
Ensuite, nous avons voulu remédier au fait que dans chaque liste (température, humidité, etc…) les valeurs de tous les capteurs étaient présents. Puisque le graphe n’était alors pas exploitable car illisible, nous avons créé de nouvelles listes, chacune associée à un numéro de capteur, contenant les valeurs d’une grandeur choisie. Ci-dessous, nous avons l’inventaire des listes des grandeurs qu’ont mesurées les capteurs 1 et 6. Enfin il était nécessaire de le faire pour chaque liste car toutes doivent posséder le même nombre de valeurs.

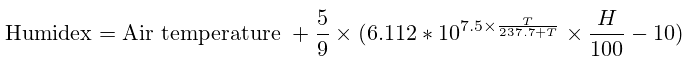


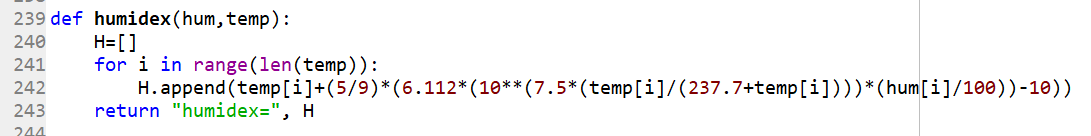
Ensuite, nous avons créé des fonctions permettant de déterminer certaines valeurs caractéristiques pour chaque liste dans le but de les reporter ensuite sur le graphe associé. Ainsi, les fonctions créées permettent de calculer dans l’ordre : le minimum, le maximum, la moyenne, la variance et l’écart-type.



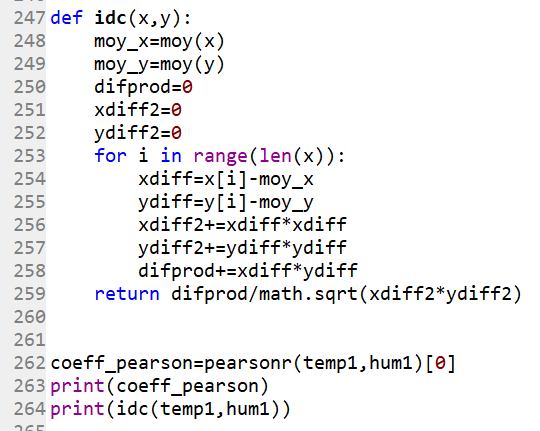
Pour déterminer la médiane d’une liste, il faut que celle-ci soit triée préalablement. Ainsi, nous avons utilisé un algorithme de tri (l.217) pour pouvoir créer ensuite une fonction médiane valable (l.231).



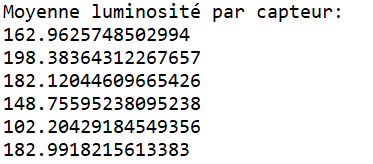
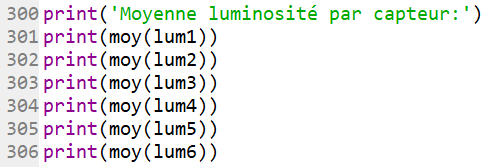
Par la suite, il est demandé de calculer l’indice humidex. Il se calcule à l’aide de cette formule :

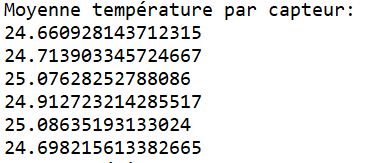
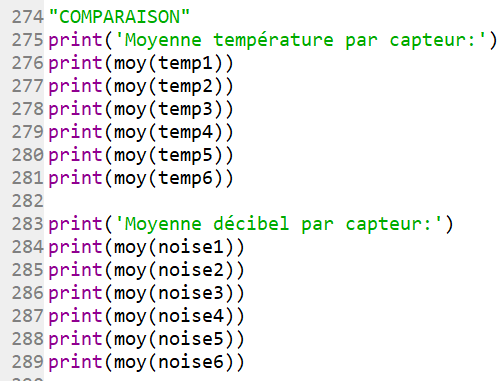
Avec T la température et H l’humidité relative. Ainsi, nous avons créé une fonction qui calcule pour chaque ligne du tableau CSV de départ l’indice humidex correspondant.

Par la suite, il était demandé de calculer l’indice de corrélation.

Pour vérifier les résultats de notre fonction, nous avons importé la fonction pearsonr depuis la bibliothèque scipy.stats. Cette fonction permet de calculer le coefficient de corrélation de Pearson.

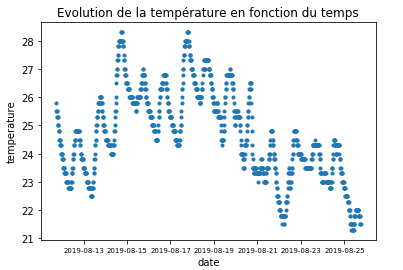
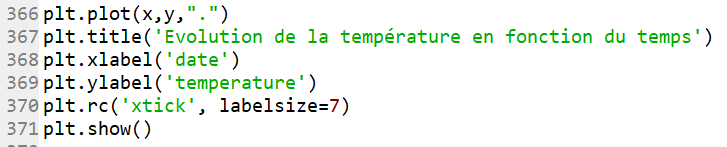
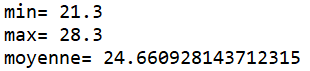
/Pour vérifier les résultats de notre fonction créée, nous avons utilisé la bibliothèque scipy.stats et la fonction correspondant à l’indice de corrélation déjà implémentée dans Python. Les résultats obtenus étaient similaires à ceux fournis par notre fonction, d’où la bonne vérification des résultats.





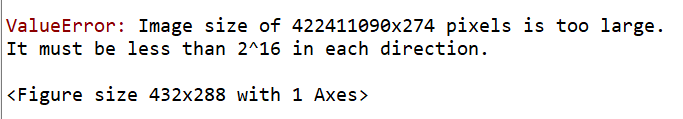
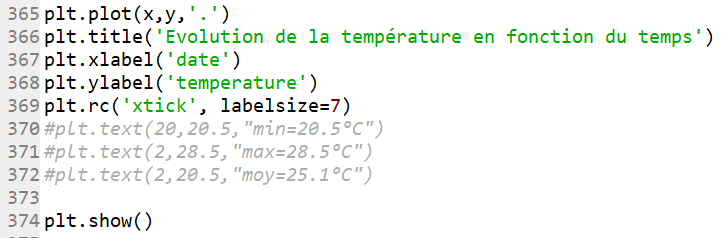
Afin de faire apparaître les anomalies, nous avons voulu comparer les différents capteurs entre eux pour vérifier qu’aucun ne soit défectueux. Ainsi, nous avons calculé différentes moyennes comme celle de la luminosité ou de la température. Il est alors ressorti que le capteur 5 avait tendance à obtenir une moyenne différente des cinq autres, par exemple environ 102 lux (l.280) alors que tous les autres obtiennent une moyenne autour de 150-190 lux. Cependant, la différence n’était pas assez significative pour que l’on puisse en déduire quoi que se soit.

Il était aussi demandé d’afficher des courbes montrant l’évolution d’une variable en fonction du temps. Pour se faire nous avons utilisé la bibliothèque mathplotlib.pyplot (ligne 10).



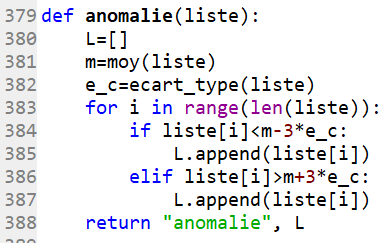
Par exemple, ici, se trouve le graphe de la température en fonction du temps du capteur 1. Nous avons préféré travailler avec des points sur le graphique plutôt que des lignes pour plus de clarté. Le x de la légende représente la liste du temps et le y la liste des différentes températures.

Nous avons ensuite essayé d’afficher les valeurs caractéristiques sur le graphe.



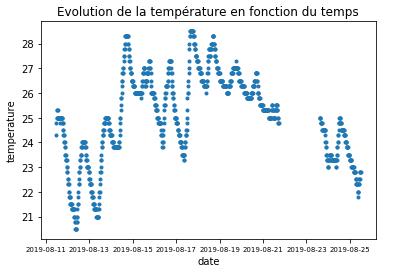
En voulant insérer différentes valeurs comme le minimum, le maximum et la moyenne sur la courbe, par des lignes de codes (ici décommentées), nous nous sommes retrouvés devant une erreur inconnue. Ainsi, il nous a été impossible d’afficher les valeurs caractéristiques sur la courbe.

Enfin, nous avons voulu créer une fonction permettant de déterminer les anomalies des différents capteurs et des différentes données.



Pour se faire nous avons mis dans une liste toutes les données supérieures à la moyenne plus trois fois l’écart type ou inférieures à la moyenne moins trois fois l’écart type. Or, cette fonction nous renvoyait uniquement des listes vides. Ainsi, nous en avons conclu que les anomalies n’étaient pas dues au changement d’environnement comme une trop forte luminosité ou un trop plein de CO2 dans les bureaux.

Nous avons alors recréé des graphes des différentes données en fonction du temps et avons remarqué une anomalie lors de l’affichage de graphe du capteur 5.



En effet, on remarque une interruption totale des valeurs pendant la journée du 22. De ce fait, nous en avons conclu à une défaillance totale du capteur 5 pendant la journée du 22 août 2019. Et en effet, pour chaque graphe d’une donnée comme la lumière ou l’humidité en fonction du temps, on remarque une interruption soudaine des points du graphique au moment de la journée du 22.

Utilisation de GitHub :

Github est une plateforme en ligne qui permet aux utilisateurs de stocker et de partager différentes données et fichiers qu’ils ont créés.

Dans un premier temps, nous avons utilisé l’interface GitBash pour utiliser GitHub, après quelques essais non fructueux pour lier nos dépôts locaux et dépôts distants, nous avons décidé d’utiliser GitHub Destock que nous trouvions plus intuitif.  Nous avons donc créé un dépôt distant en créant un repository sur Github dont on a autorisé l’accès à chacun de nous. Chacun pouvait alors déposer ses fichiers sur le dépôt distant. Pour cela, chaque membre du groupe envoie du dépôt local au dépôt distant ses modifications du fichier en utilisant la commande git push. Et chacun des membres du groupe les récupère en utilisant Fetch origin.

Ainsi, GitHub Destock nous permet une mise en commun rapide de nos avancées et une trace de toute modification exécutée par nos camarades donc de suivre l’évolution de l’ensemble du projet.