Dupont Clément
Chesné Amaury

Rapport - projet Informatique

Sujet 2 :
Mesurer les similarités des capteurs pour chaque dimension, qu'en concluez-vous ?
Proposer et implémenter un algorithme permettant de mesurer la similarité automatiquement et de la montrer sur les courbes.

Collaboration sur l'algorithme:

Notre premier obstacle fut l'installation et la compréhension de la plateforme GitHub, nous ne comprenions pas vraiment comment partager notre travail et comment utiliser au mieux les répertoires.
En demandant à nos camarades nous avons installé GitHub Desktop ce qui nous a débloquer dans l'utilisation de GitHub. Grâce à cela nous arrivions à importer les travaux fait depuis nos ordinateur sur le répertoire contenant l'algorithme.
L'historique des modifications proposé par GitHub nous a permis de comprendre les ajouts de chacun et en quoi ceux-ci modifient le programme, permettant une compréhension complète du programme pour tout le monde.
En parallèle de GitHub nous utilisions Spyder pour vérifier la syntaxe de nos programmes et des possibles erreurs qui ferait cracher le programme.
Nous testons régulièrement nos programmes sur Spyder.

Codage de l'algorithme:

- Notre premier obstacle concernant le codage de l'algorithme était la manière de traiter des fichiers CSV. Nous ne savions pas comment nous atteler aux traitement des fichiers.

Nous savions que nous pouvions le faire avec la bibliothèque Pandas donc nous avons cherché à résoudre ce problème grâce à celle-ci.
Nous avons donc utilisé la commande « pd.read_csv » pour lire le fichier et ensuite la commande « .tolist() » pour convertir les valeurs des différentes colonnes en listes.

Avec ceci fait nous allions donc pouvoir commencer à traiter nos données.

- Nous avons choisis de commencer par écrire les fonctions permettant de calculer les valeurs statistiques de nos données, notamment:

La moyenne arithmétique, la valeur maximale, la valeur minimale, l'écart-type, la médiane, la variance servant à calculer l'écart-type.

Mais également des fonctions permettant de calculer les températures de rosée, de trier les listes par ordre croissant, de calculer l'humidex qui dépend de la température de rosée et la corrélation entre deux variables.

Nous avons fait comme cela pour avoir un répertoire de fonctions que nous rappellerons dans la suite du programme.

Nous allons dans la suite de notre programme tracer des courbes grâce à la bibliothèque matplotlib.pyplot.

- Ces courbes seront les différentes données du fichier CSV en fonction du temps.

À l'aide de la commande « input » nous laissons à l'utilisateur la possibilité de choisir premièrement l'intervalle de temps voulu ou sinon de tracer la courbe sur toutes les valeurs de temps existantes. Mais également l'utilisateur peut choisir la donnée qui l'intéresse.

Cependant, il faut faire attention à ce que l'utilisateur rentre la date correctement si toutefois il décide de choisir un intervalle de temps et que celle-ci existe dans nos fichiers sinon le programme renverra une erreur.

Sur les courbes des données du document CSV on fera apparaître la droite horizontale de la valeur minimale et maximale et en dessous de la courbe le programme renverra la valeur du maximum, du minimum, la médiane, la moyenne la variance et l'écart type.

On tracera ensuite la courbe de l'humidex en fonction du temps.

On renverra l'indice de corrélation et le graphique avec les données pour lesquelles et indices a été calculé. L'indice de corrélation sera calculé entre deux données que l'utilisateur choisira et sur l'intervalle de temps choisi par celui-ci.

Les programmes qui nous ont pris du temps étaient les programmes « inter », « tri_int », et « tri_capteur ».

- La fonction **inter** renvoie une liste avec les dates et les horaires compris dans l'intervalle de date donné.

Les trois arguments de cette fonction sont: une liste d'horaires associés à des dates, une date et son horaire de début, une date et son horaire de fin. Nous avons rajouté une heure de début et une heure de fin pour les dates rentrées par l'utilisateur car l'utilisateur va juste rentrer des dates. Or dans notre tableau il y a des dates plus des heures. C'est pourquoi nous avons ajouté des heures pour pouvoir comparer les dates de début et de fin à celle dans notre liste de temps.
On va créer une liste(1) vide que l'on va remplir.

On va parcourir la liste si les éléments de la liste son entre les dates et horaires de fin et de début on les met dans la nouvelle liste(1). Puis on renvoie la liste(1) qui est composée des dates et des horaires triées.
La complexité moyenne de cet algorithme est O(n).

- La fonction **tri_int** est une fonction qui à partir de la liste de temps complète triée, de la donnée que l'on veut étudier et un intervalle de temps nous renvoie toute les données triées.

Elle est donc composée de trois arguments, une liste de temps (la liste avec tous les temps du tableau) qui doit être triée dans l'ordre croissant, la liste de la donnée que l'on veut étudier (la liste avec toutes les données d'une catégorie du tableau) et la liste correspondant à l'intervalle de temps renvoyée par la fonction inter.
On vient créer une liste vide nommée L que nous allons remplir et on vient associer une variable j à la valeur 0.
On parcourt la liste de temps complète triée dans l'ordre croissant, tant que la valeur de la liste de temps d'indice j est différente de la valeur de la liste de l'intervalle de temps d'indice i alors on rajoute plus un à j et on met la valeur de la liste de notre donnée d'indice j dans la liste L qui est au départ vide, puis on réassortie la variable j à 0. On renvoie ensuite la liste L. En fait on regarde l'indice de temps qu'on veut (dans la liste complète de temps et non triée), cet indice est le même que celui de la donnée que l'on veut.
Ainsi nous avons à la fin une liste avec toutes les données associées à l'intervalle de temps souhaité.
La complexité moyenne de cet algorithme est O(n).

- Le programme **tri_capteur** est un programme qui nous a servi à trier les données en fonction des capteurs qui les avaient enregistrées.

On rentrera par exemple dans la fonction tri_capteur la liste « humidity » qui représente toutes les données pour l'humidité et elle triera les données de l'humidité en renvoyant une liste qui associera les données de l'humidité aux capteurs qui les ont enregistrés.
La fonction a comme argument la liste de la donnée que nous voulons trier par capteur.
Pour ce faire nous allons créer six listes, appelées cap1,…, cap6.
Nous allons faire prendre à i des valeurs allant de 0 à 5.
Si la valeur de la liste capteur d'indice i est égale 0 on mettre la valeur de la liste de donnée d'indice i dans cap6, pour 1 on la mettra dans cap1, pour 2 on la mettra dans cap2, pour 3 on la mettra dans cap3, pour 4 on la mettra dans cap4, pour 5 on la mettra dans cap5.
On renverra ensuite les listes cap1, cap2, cap3, cap4, cap5, cap6.

Par la suite nous afficherons les valeurs mesurées par tous les capteurs pour toutes les données sur différent graphe. Nous savons cependant que ces graphiques ne sont pas rigoureusement juste car des éléments des listes cap1,cap2... d'indice j n'ont pas forcément étaient tous mesurés au même moment. Cela explique que sur les graphiques il peut y avoir des courbes bien différentes d'un capteur à l'autre.

Mesure de la similarité entre chaque capteur pour les différentes données:

- Nous allons trouver la similarité entre les différents capteurs à l'aide du programme **similarité**.

Cette tâche est celle qui nous à pris le plus de temps car au début nous ne comprenions pas la consignes et ce que signifiait la similarité.

La fonction a comme argument la liste de temps triée, les capteurs, les données pour laquelle nous voulons calculer la similarité des capteurs et les horaires de début et fin.
Nous allons tout d'abord créer un intervalle de temps en rappelant la fonction **inter** que nous avons écrit plus tôt.
Cette fonction inter va nous servir à rappeler la fonction **tri_int** qui va trier les valeurs des capteurs par ordres chronologiques. En rappelant la fonction **tri_capteur** on va attribuer les différentes valeurs contenues dans la liste de notre données aux différents capteurs qui les ont enregistrées.
Les capteurs n'ayant pas nécessairement enregistrés les mêmes valeurs sur un même intervalle de temps, nous allons donc comparer la longueur des listes comprenant les valeurs des capteurs pour ainsi comparer la similarités sur autant de valeurs que contient la liste la plus courte.
Pour comparer les valeurs nous allons faire la grosse approximation que les valeurs classées par ordres chronologiques dans les listes de capteurs ont été acquis à des différences de temps très faible et que deux valeurs correspondant à un même indice dans deux listes appartenants à des capteurs différents ont été pris à un moment à peu près similaire.
Nous allons créer 15 listes qui vont stocker la différences en valeurs absolue des valeurs de chaque capteur avec les valeurs des 5 autres capteurs.
En appelant la fonction **moy** nous allons ensuite faire la moyenne des valeurs présentes dans chacune de ces 15 listes.
Nous allons créer 6 listes contenant les 5 moyennes correspondant à différence de valeurs des listes des capteurs.
Puis à l'aide d'une commande if nous allons demander de renvoyer pour les 6 capteurs lequel lui est le plus similaire en se basant sur le fait que plus la moyenne de la différence est faible plus le capteur sera similaire.

Pour le programme de similarité, malgré de nombreux essais nous n'avons pas réussi à le faire tourner. Il semblerait que l'erreur soit au niveau des appels if...elif...else.
Le programme est censé nous renvoyer 6 phrases (une par capteur) et nous exécutons le programme pour les 5 données (bruit, humidité,...etc) donc en tout 30 phrases mais au lieu de cela il en renvoie 30 par capteur et seulement la similarité relative au capteur 1. Donc il semblerait que ce soit un problème d'affichage lié certainement aux boucles if et for. Nous n'arrivons pas à corriger l'erreur.

On ne savait pas comment comparer deux chaînes de caractères bien que ce soit des nombres, au final en utilisant des outils de comparaisons classique type supérieur et inférieure nous avons réussi à résoudre le problème.

Du fait que nous avons appelé beaucoup de fonctions différentes, nous avons eu du mal quant à la rigueur demandée pour ne pas se tromper dans les arguments, surtout avec les listes de temps, qui parfois était triée ou non.

Les capteurs n'acquièrent pas autant de mesures sur tout l'intervalle de temps, c'est pourquoi les longueurs des listes avec les données récoltées ne sont pas les mêmes. Ceci va donc poser un problème dans la fonction **tri_int** : il y aura donc un out of range.

Concernant la mesure de similarité, n'arrivant pas à trouver une solution précise nous avons opté pour une solution qui nous paraîtrait logique avec néanmoins de grosses approximations qui pourraient complètement fausser nos résultats.
Nous sommes conscients que cette solution n'est pas rigoureusement juste.

En ce qui concerne Github, les fonctions pull et push nous ont permis de faciliter le travail de groupe dans ce projet d'informatique. En effet grâce à github nous avons pu communiquer et partager nos programmes facilement. Nous avons créé un nouveau répertoire pour vous y déposer le travail (le programme et le compte rendu).