

RAPPORT DE PROJET PROGRAMMATION

SAGOT EMMA – HAMID IKRAM (Groupe C)

SUJET 1

Objectifs :

Explorer et analyser des données 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), ainsi que trouver des anomalies dans les données, proposer et implémenter un algorithme permettant de les relever automatiquement et de les montrer sur les courbes. Environnement de développement utilisé : Spyder.

I – Méthode de travail

Afin d'obtenir correctement les résultats demandés, nous avons essayé de mettre toute la complexité dans la première partie du programme. Nous avons traité tout ce qui concerne les instructions et conditions concernant les variables et arguments, ainsi que la définition des intervalles dans la même partie afin d'éviter les répétitions dans la suite du programme. Nous nous sommes servis du site suivant pour l'utilisation de quelques fonctions : <http://www.python-simple.com/python-numpy/statistics-numpy.php>

La complexité du programme se trouve dans le fait qu'il doit être optionnel : le programme doit s'exécuter même si les actions ou variables rentrées n'existent pas. Par défaut, si les points de départ ou d'arrivée ne sont pas renseignés, toutes les dates seront affichées sur le graphe. Si le fait de donner un intervalle était obligatoire, le programme serait moins complexe.

II – Choix des bibliothèques

Tout projet de programmation traitant de la manipulation d'un grand nombre de données nécessite l'utilisation des modules numpy et matplotlib pour manipuler ces données sous forme de tableau, puis les afficher sous forme de graphe. Ces deux modules nous étaient familiers car nous nous en étions déjà servis dans nos études.

En revanche, nous avons dû apprendre à utiliser les modules sys et DateTime. Le module DateTime permet une manipulation beaucoup plus facile des dates et nous a servi à simplifier l'écriture de l'échantillonnage temporel (ex : 2019-08-20 11:46:11 +0200) des différents capteurs qui s'est avérée lourde à manipuler sur la durée. Le module sys permet entre autres choses de lister les variables du programme et les manipuler. Pour apprendre à utiliser ces deux modules nous nous sommes servis de la documentation disponible en ligne et nous avons pu compter sur les explications de certains proches développeurs.

III – Utilisation de Git et GitHub

Dans un premier temps, la prise en main de l'outil GitHub a été laborieuse car il nous était entièrement nouveau mais il s'est par la suite révélé très utile pour mener à bien ce projet. Comprendre la notion de copie en local du « Repository » a été la partie la plus complexe. Il a donc fallu apprendre à utiliser les commandes de bases (à saisir dans le Terminal). Notre utilisation de GitHub étant très élémentaire, nous nous sommes uniquement servies des commandes copy, add, commit, push et pull, sans chercher à créer d'autres branches. Une fois la prise en main de ces commandes acquises, nous avons pu utiliser GitHub à la manière d'un « google doc » pour récupérer les modifications de notre binôme et les comprendre grâce à l'ajout de commentaires pertinents.

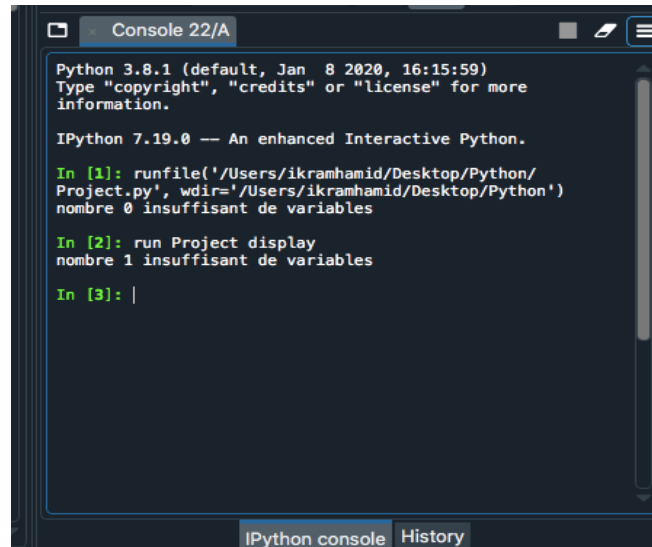
IV – Détail du code

1ère partie - Définition et déclaration des différentes actions et variables :

Afin que le programme se lance dans une ligne de commande, si on lance Powershell terminal, on a utilisé l'instruction : `if __name__ == "__main__":`

Cette première partie sert à différencier les paramètres : « display », qui montre que les graphes , « displayStat », qui montre les graphes et les statistiques et « Corrélation », qui a un autre objectif différents des deux. Pour « display » et « displayStat », on a qu'une seule variable, mais pour corrélation on a deux variables, donc il faut donc différencier ces variables, on a utilisé des instructions et des conditions qui vont permettre l'exécution du programme , même si le variable n'est pas suffisant pour éviter qu'il se plante, comme le montre la capture d'écran ci-dessous :

Nous avons pris un intervalle qui commence de 1 car le 0 correspond du nom du fichier, ce qui n'est pas vraiment utile pour nous. Cette première partie du programme a été réservée une aux définitions de tous les arguments dont on aura besoin tout au long du programme grâce à la fonction args.pop().



```
Python 3.8.1 (default, Jan 8 2020, 16:15:59)
Type "copyright", "credits" or "license" for more
information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/ikramhamid/Desktop/Python/
Project.py', wdir='/Users/ikramhamid/Desktop/Python')
nombre 0 insuffisant de variables

In [2]: run Project display
nombre 1 insuffisant de variables

In [3]: |
```

IPython console History

2ème partie - Les intervalles :

Après avoir défini nos différentes variables et pour éviter la complexité, il faut s'assurer qu'on a au moins deux variables, que les variables dont on a besoin pour la suite du programme. Si ces deux variables sont présentes, on peut donc définir l'intervalle qu'on veut avec start_date et end_date.

3ème partie - Lecture du fichier CSV :

Permet de lire le fichier CSV et donner les noms et types de chaque colonne du fichier. Tous ces données du CSV seront ainsi dans la variable data.

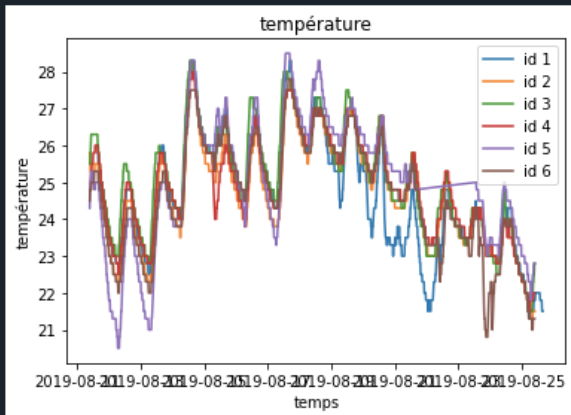
4ème partie - Les intervalles et le filtrage :

Pour display et displayStat : Il faut filtrer les axes des abscisses (le temps) et des ordonnées (la température), on a utilisé un booléenne boofilter. Si la date indiquée n'est pas dans l'intervalle (entre strat_date et end_date), l'information est fausse, sinon elle est vraie. On se servira de la variable data où on a toutes les données, il suffit de choisir la colonne dont on a besoin (temps, température, humidité....). Au cas où il y a pas d'intervalles, toute la colonne sera affichée. Pour la corrélation : même principe mais avec deux variables.

5ème partie - Affichage des courbes :

On utilise la fonction plot() pour l'affichage. On affiche les courbes des différentes variables (température, humidité...) en fonction du temps, donc la fonction plot prend en argument le temps (times) et la variable (values). Afin d'afficher les courbes d température par exemple il faut mettre les commandes : run nom_du_fichier display température

```
In [4]: run Project display température
```



<Figure size 432x288 with 0 Axes>

```
In [5]: x
```

6ème partie - Calcul de l'indice Humidex :

La formule de l'indice Humidex est la suivante :

$$\text{Humidex} = T_{\text{air}} + 0.5555 \left[6.11 \times e^{5417.7530 \left(\frac{1}{273.16} - \frac{1}{273.15 + T_{\text{rosée}}} \right)} - 10 \right]$$

Pour le calculer on a utilisé la fonction CalculateHumidex(T, RH), afin de définir toutes les variables dont on a besoin pour sa formule. Affichage de la courbe avec l'indice Humidex.



7ème partie - Calcul des statistiques :

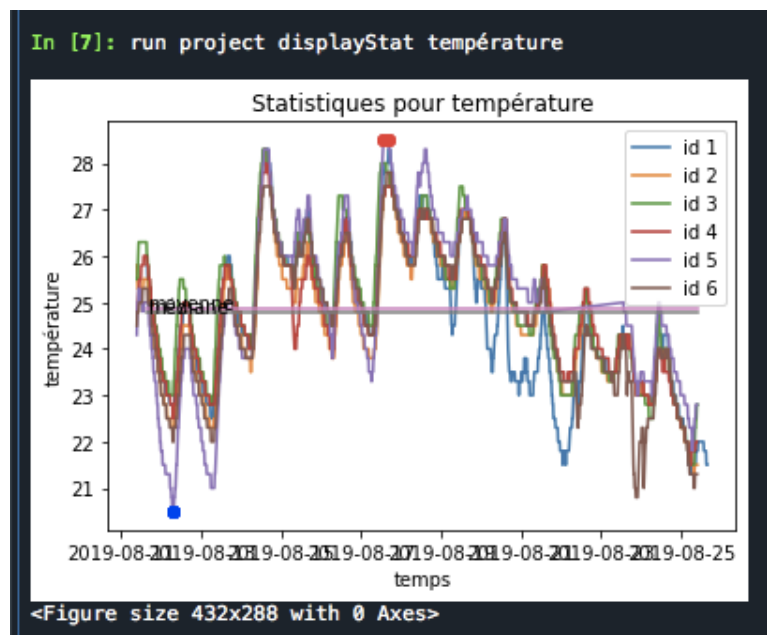
Min/Max: On commence par une première condition if pour s'assurer que l'action est displayStat, sinon le programme va pas s'exécuter. On utilise la fonction plotpoints() qui permet d'afficher le min et max des courbes en choisissant le format d'affichage et la couleur. Cette fonction prend donc comme argument la valeur des min ou max (value) et le format de leur affichage(format). bo et ro désignent la couleur et le symbole(b pour bleu et r pour rouge, ainsi que le symbole comme). Les valeurs min et max sont présentes dans plusieurs reprises, d'où l'utilisation de « indices =where(values==value)» qui permet de nous donner quand la valeur a été mesurée et pour quels indices. À partir de ces indices, on aura le temps exacte correspondant à la valeur(min ou max) en utilisant l'instruction « itimes=times(indices) » pour localiser la valeur

dans la courbe qui sera ensuite affichée à l'aide des fonctions `plotpoints(np.min(values), « bo »)` et `plotpoints(np.max(values), « ro »)`.

Moyenne : Contrairement au min et max, la moyenne n'est pas forcément une valeur qui appartient à la courbe. On a choisi donc de mettre la valeur de la moyenne dans une ligne dans la courbe, sinon on risque de ne pas apercevoir cette valeur sur la courbe. On a utilisé la fonction `mean()` de la bibliothèque `numpy` qui permet de calculer la moyenne. La fonction `plt.plot([times[0], times[-1]], [mean, mean])` permet d'afficher cette ligne de moyenne qui s'étend tout au long de la courbe.

Médiane : Pareil que pour la moyenne, elle sera affichée sous forme de ligne, et on utilise la fonction `median()` pour la calculer.

Affichage des statistiques sur la courbes de la température en fonction du temps :



On a choisi de mettre les statistiques ci-dessous comme commentaires pour éviter d'avoir plusieurs éléments sur la courbe, ça risque de nous perturber pour la suite du programme.

Écart type: Le même principe en utilisant la fonction `std()`.

Variance : Le même principe en utilisant la fonction `var()`.

8ème partie - La corrélation :

Filtre : On part du même principe utilisé précédemment pour le filtrage, mais en utilisant deux variables. **Indice de corrélation :** On calcule le coefficient de corrélation en utilisant la

fonction `corrcoef()` en lui donnant les valeurs des deux valeurs des variables. Le `print("indice de corrélation entre", variable1, "et", variable2, ":", corrcoef[0][1])` permettra d'afficher la valeur de l'indice dans la console.

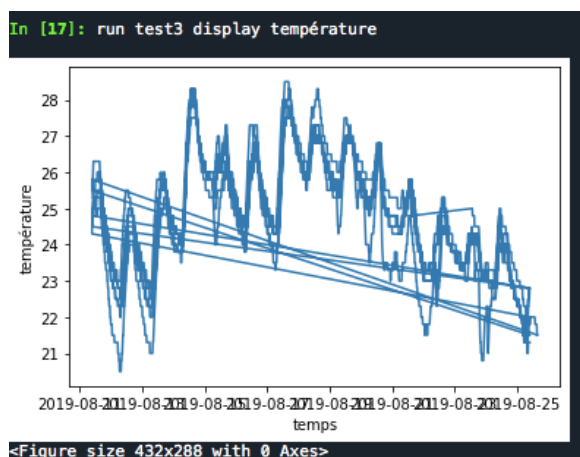
9ème partie - le bonus (affichage des deux courbes représentant les deux variables en fonction du temps sur le même graphe et indication dans la légende la valeur de l'indice de corrélation) :

Pour afficher une seule courbe, on utilise la fonction `plot.plot()` en lui donnant les abscisses et les ordonnées. Mais en affichant deux courbes dans le même graphe, on ne peut pas mettre les ordonnées de chacune de ces courbes au même endroit car leurs unités et valeurs sont différentes. On a donc un axe à gauche pour les valeurs de température et un axe à droite pour l'humidité. On utilise les fonctions `axe1_set_ylabel` pour le premier axe des ordonnées et `axe2_set_ylabel` pour le deuxième axe des ordonnées. Afin de différencier les deux courbes, on a choisi deux couleurs différentes en utilisant « `color=tab:red` » pour le rouge et « `color=tab:blue` » pour le bleu.

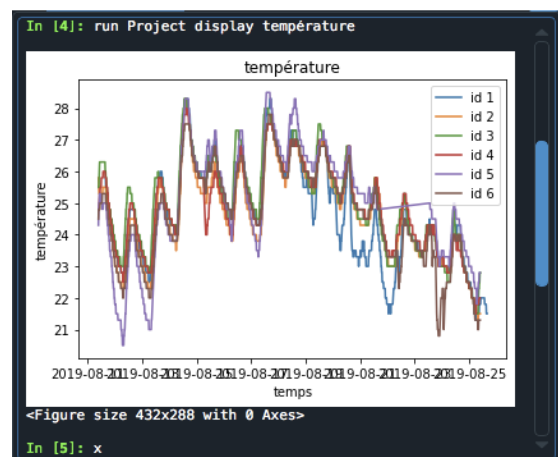
10ème partie – repérage de l'anomalie et proposition de solution :

Plusieurs graphes apparaissent car les dates se répètent pour différents indices. Par exemple, si on prend l'exemple du graphe suivant, les fonctions ont des images différents (températures) pour les mêmes abscisses(temps).

On conclut qu'il y a six capteurs avec différentes valeurs mesurées. Il faut ajouter une variable extra qui nous permettra de choisir l'appareil dont on a besoin pour calculer et afficher les données. Chaque courbe affichée correspondra à un capteur précis. Pour cela on utilise la fonction `uniqueids = set(ids)`



Graphe avant introduction variable extra



Graphe après introduction variable extra

11ème partie - le bonus (Trouvez automatiquement les périodes horaires d'occupations des bureaux)

Si on regarde le graphe, on peut voir que à 7h30, l'indice Humidex augmente et entre 16h30 et 17h, il diminue, ce qui indique les horaires d'ouverture et de fermeture des bureaux.