CHAPUIS Yukio
VANWORMHOUDT Lola

Sujet 2 - Groupe B
Promo 62

PROJET DE PROGRAMMATION INFORMATIQUE

EIVP - ÉCOLE SUPÉRIEURE DU GÉNIE URBAIN

UTILISATION DE GITHUB

Lien GitHub de notre projet : https://github.com/YukiBozu/groupe_42_eivp1_projet1.git

Github est un site internet de partage, de gestion et de développement de programmes informatiques.

Nous pensons avoir compris le fonctionnement et l'utilisation de Github dans ses grandes lignes. Cependant, nous avons peu utilisé Github pendant ce projet informatique. En effet, nous avons préféré nous concentrer sur la compréhension et l'apprentissage du langage Python plutôt que sur l'apprentissage de Github. De plus, nous nous sentions davantage à l'aise avec d'autres logiciels de partage de fichiers et de développement de programme informatique. Nous avons utilisé Google Drive pour le partage des fichiers (tests, fichiers csv, explications…).
Néanmoins, nous pensons que Github peut s'avérer être un outil très pratique pour la conservation et la visualisation de l'historique des modifications d'un projet collaboratif. Nous penserons à utiliser Github à l'avenir, notamment pour les projets menés en groupe à l'EIVP, Github permettant d'intégrer des plugins issus de différents logiciels que nous utilisons fréquemment dans le cadre de ces projets (fichier QGIS, fichier Excel…).

Pour la rédaction de ce programme, nous avons travaillé à l'aide d'un éditeur comme Pyzo et Visual Studio Code, en parallèle d'expérimentations sur Google Colaboratory. L'éditeur permettant d'avoir une vision d'ensemble de la structure du code mais n'étant pas suffisamment interactif pour l'essai de commandes.

SOMMAIRE

INTRODUCTION

I. Contexte

Le programme à réaliser, sous forme d'un script, doit permettre l'analyse de données issues de campagnes de mesure au sein d'un bâtiment de bureaux.

II. Données à traiter

Ces mesures se présentent sous la forme d'un fichier CSV, comportant près de 7900 données, collectées par 6 capteurs.

Le bruit, la température, le taux d'humidité, la lumière et le taux de CO2 sont les dimensions collectées à un instant donné.

Le but est de mesurer les similarités des capteurs pour chaque dimension, et de discuter ces résultats. Puis, il s'agira de proposer et d'implémenter un algorithme permettant de mesurer automatiquement la similarité.

	A	B	C	D	E	F	G	H	I	J
1		id	noise	temp	humidity	lum	co2	sent_at		
2	0	1	35.5	25.8	55.0	282	448	2019-08-11 17:48:06+02:00		
3	1	1	44.5	25.5	55.0	288	429	2019-08-11 18:03:03+02:00		
4	2	1	34.5	25.5	55.0	286	417	2019-08-11 18:18:03+02:00		
5	3	1	37.5	25.5	54.5	282	433	2019-08-11 18:33:03+02:00		
6	4	1	36.0	25.3	55.0	274	403	2019-08-11 18:48:03+02:00		
7	5	1	30.0	25.3	55.0	254	410	2019-08-11 19:03:03+02:00		
8	6	1	33.0	25.3	55.5	234	407	2019-08-11 19:18:03+02:00		
9	7	1	30.0	25.3	56.0	200	407	2019-08-11 19:33:03+02:00		
10	8	1	27.0	25.0	56.5	148	416	2019-08-11 19:48:02+02:00		
11	9	1	30.0	25.0	56.5	90	416	2019-08-11 20:03:06+02:00		
12	10	1	31.5	25.0	57.0	36	411	2019-08-11 20:18:09+02:00		
13	11	1	27.0	24.8	57.0	8	406	2019-08-11 20:33:03+02:00		
14	12	1	27.0	24.8	57.5	0	415	2019-08-11 20:48:03+02:00		
15	13	1	27.0	24.8	58.0	0	403	2019-08-11 21:03:02+02:00		
16	14	1	27.0	24.5	58.5	0	412	2019-08-11 21:18:03+02:00		
17	15	1	27.0	24.5	59.0	0	402	2019-08-11 21:33:03+02:00		
18	16	1	27.0	24.5	59.5	0	411	2019-08-11 21:48:03+02:00		
19	17	1	27.0	24.5	60.0	0	412	2019-08-11 22:03:02+02:00		
20	18	1	27.0	24.3	60.0	0	402	2019-08-11 22:18:02+02:00		
21	19	1	27.0	24.3	60.5	0	401	2019-08-11 22:33:02+02:00		
22	20	1	27.0	24.3	60.5	0	404	2019-08-11 22:48:02+02:00		
23	21	1	27.0	24.3	60.5	0	413	2019-08-11 23:03:02+02:00		

Portion du fichier CSV comportant les données

Le programme comprend différentes parties : d'abord, il s'agit de préparer le futur programme en important les données et packages nécessaires à la réalisation des courbes et calculs. Puis, il s'agit de créer les fonctions (exemple : action du calcul de l'indice humidex, action d'affichage des valeurs statistiques…). Enfin, il s'agit d'écrire le programme en lui-même.

PROGRAMME

I. Préparation

• Importation des packages mathématiques et des packages graphiques nécessaires

```
import sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist, squareform
from datetime import date
```

L'importation de la libraire python « pandas » va permettre de manipuler plus facilement les données à analyser (manipulation de tableaux de données avec variables).
L'importation de « sys » va permettre la manipulation des arguments.
L'importation de la bibliothèque « matplotlib.pyplot » va permettre de tracer des courbes pouvant être personnalisées.
L'importation de « spicy.spatial.distance » va permettre le calcul de la distance euclidienne entre plusieurs données.
L'importation de « datetime » va permettre de gérer les données relatives au temps : dates, heures et intervalles de temps.

• Définition des différents éléments utilisés

```
FICHIER = 'EIVP_KM.csv'
CAPTEURS = ['noise', 'temp', 'humidity', 'lum', 'co2']
TYPES_CAPTEURS = {'id':'Int8', 'noise':np.float32, 'temp':np.float32, 'humidity':np.float32,
                  'lum':np.float32 , 'co2':np.float32, 'sent_at':str}
VARIABLES = CAPTEURS + ['humidex']s
UNITES = {'lum':'lux', 'noise':'dBa', 'temp':'°C', 'humidity':'%', 'co2':'ppm', 'humidex':''}
```

Il s'agit ici de préciser les différents éléments qui vont être utilisés au sein du programme. Puis on définit, via des listes les variables analysées par les capteurs : leur nom, leur type, l'indice humidex ainsi que leurs unités respectives.

• Importation du tableur et mise en forme des données

```
def lecture_fichier(path):
    df=pd.read_csv(path, sep=';', header=0, index_col=0, dtype=TYPES_CAPTEURS,
parse_dates=['sent_at'])
    df['humidex']=df.apply(lambda x: calcul_humidex(x['temp'],x['humidity']), axis=1)
    return df
```

Avec ces lignes de commande, on choisit la mise en forme des données. La commande « sep=';' » instaure un séparateur pour la clarté de lecture et la commande « parse_dates=['sent_at']) » analyse la représentation textuelle d'une date. Enfin, la ligne df['humidex']=df.apply(lambda x: calcul_humidex(x['temp'],x['humidity']), axis=1) va ajouter une colonne dans le tableau. Cette colonne comporte les résultats des calculs d'humidex.

	id	noise	temp	humidity	lum	co2	sent_at	humidex
0	1	35.5	25.8	55.0	282.0	448.0	2019-08-11 17:48:06+02:00	35.93
1	1	44.5	25.5	55.0	288.0	429.0	2019-08-11 18:03:03+02:00	35.45
2	1	34.5	25.5	55.0	286.0	417.0	2019-08-11 18:18:03+02:00	35.45
3	1	37.5	25.5	54.5	282.0	433.0	2019-08-11 18:33:03+02:00	35.36
4	1	36.0	25.3	55.0	274.0	403.0	2019-08-11 18:48:03+02:00	35.13

Portion du fichier CSV avec la colonne « humidex »

```
def dates_croissantes(date1, date2):
    """ Remet les dates dans l'ordre si besoin
    """
    if date1 and date2 and date2 < date1:
        return date2, date1
    else:
        return date1, date2
```

Pour s'assurer que les dates entrées en option soient cohérentes, on crée la fonction « dates_croissantes », qui vise à remettre les dates dans l'ordre si besoin, la date 1 devant être toujours inférieure à la date 2.

II. Définition des fonctions

• Fonction « intervale_dates»

```
def intervale_dates(df, date_deb, date_fin):
    date_deb, date_fin = dates_croissantes(date_deb, date_fin)
    if date_deb == '' or date_deb == None:
        date_deb = df['sent_at'].min()
    if date_fin == '' or date_fin == None:
        date_fin = df['sent_at'].max()
    if date_deb < df['sent_at'].min():
        date_deb = df['sent_at'].min()
    if date_fin > df['sent_at'].max():
        date_fin = df['sent_at'].max()
    if date_deb > df['sent_at'].max():
        date_deb = df['sent_at'].max()
    return date_deb, date_fin
```

Cette fonction permet de modifier les dates si celles-ci ne correspondent pas aux dates recueillies par les capteurs. Par exemple, si la date de fin entrée en commande est supérieure à la date maximale des capteurs, alors la date entrée sera rapportée à la date maximale : if date_fin > df['sent_at'].max(): date_fin = df['sent_at'].max(). Idem pour la date de début.

• Fonction « commande_display »

```
def commande_display(df, args):
    variable = args.variable
    display(df, variable, args.start_date, args.end_date)
def display(df, nom_variable, date_deb, date_fin, withStat=False):
    date_deb, date_fin = intervale_dates(df, date_deb, date_fin)
    # filtrage par rapport aux dates
    df_filtre = df[(df['sent_at'] >= str(date_deb)) & (df['sent_at'] <= str(date_fin))]
    # tri des données selon les dates
    df_filtre_tri = df_filtre.sort_values(by=['sent_at'])
     # Sans les stats:
    if not withStat:
        titre = nom_variable
        df_filtre_tri.plot(x="sent_at", y=nom_variable, color='blue', rot=30)
        plt.title(titre)
        plt.show()
```

La fonction « display » permet d'afficher des courbes montrant l'évolution d'une variable (nom_variable) en fonction du temps (sent_at). Optionnellement, il est possible de préciser un intervalle de temps (date_deb, date_fin) grâce à la commande filtre.

```
# Avec les stats:
    else:
        if len(df_filtre_tri.index) > 1:
            # calcul des statistiques
            data = df_filtre_tri[nom_variable]
            moyenne = data.mean()
            ecart_type = data.std()
            variance = data.var()
            mediane = data.median()
            min = data.min()
            max = data.max()
            # affichage de la courbe et des stats
            _, ax = plt.subplots()
            df_filtre_tri.plot(x="sent_at", y=nom_variable, color='blue', rot=30, ax=ax)
            plt.axhline(moyenne, color='r', label='moyenne')
            plt.axhline(min, color='g', label='min')
            plt.axhline(max, color='c', label='max')
            plt.axhline(mediane, color='y', label='mediane')
            ax.fill_between(x=df_filtre_tri["sent_at"], y1=(moyenne - ecart_type), y2=(moyenne +
ecart_type), facecolor='blue', alpha=0.5, label='intervale écart type')
            labels = [f"moyenne = {moyenne:.2f} {UNITES[nom_variable]}",
                      f"min = {min:.2f} {UNITES[nom_variable]}",
                      # "min = {:.2f} {}".format(min, UNITES[nom_variable]),
                      f"max = {max:.2f} {UNITES[nom_variable]}",
                      f"médiane = {mediane:.2f} {UNITES[nom_variable]}",
                      f"écart type = {ecart_type:.2f} {UNITES[nom_variable]}"]
            handles, _ = ax.get_legend_handles_labels()
            plt.legend(handles=handles[1:], labels=labels)
            titre = f"Statistiques de {nom_variable}"
            plt.title(titre)
            plt.show()
        # Avec les stats mais dates hors de l'intervalle donc df de longueur <= 1
        else:
```

```
            date_min = df['sent_at'].min().date()
            date_max = df['sent_at'].max().date()
            print(f'enter a date between [{date_min} ; {date_max}]')
            # print('enter a date between [{} ; {}]'.format(df['sent_at'].min(),
df['sent_at'].max()))
    return

def commande_displayStat(df, args):
    variable = args.variable
    display(df, variable, args.start_date, args.end_date, withStat=True)
```

De plus, il est possible, grâce à « display_stat » d'afficher des courbes explicitant les valeurs statistiques d'une variable (nom_variable) en fonction du temps (sent_at). La moyenne, la valeur minimum, la valeur maximum, la médiane, la variance et l'écart-type sont les valeurs statistiques affichées sur les courbes. La valeur numérique de ces données statistiques est affichée dans le terminal. Il est également possible de préciser un intervalle de temps (date_deb, date_fin) grâce à la commande filtre.

• Fonction « calcul_humidex »

```
def calcul_humidex(temp, humidity):
    exp = (7.5 * temp) / (237.7 + temp)
    humidex = temp + 5/9 * 6.112 * pow(10, exp) * (humidity / 100)
    return humidex
```

La définition de l'indice humidex s'établit via une formule mathématique faisant intervenir deux paramètres : la température (temp) et l'humidité (humidity).

• Fonction « commande_correlation »

```
def commande_correlation(df, args):
    variable1 = args.variable1
    variable2 = args.variable2
    display_correlation(df, variable1, variable2, args.start_date, args.end_date)
def display_correlation(df, variable1, variable2, date_deb, date_fin):
    date_deb, date_fin = intervale_dates(df, date_deb, date_fin)
    # filtrage par rapport aux dates
    df_filtre = df[(df['sent_at'] >= str(date_deb)) & (df['sent_at'] <= str(date_fin))]
    # tri des données selon les dates
    df_filtre_tri = df_filtre.sort_values(by=['sent_at'])
    # calcul de la corrélation entre les deux variables
    corr = df_filtre_tri[variable1].corr(df_filtre_tri[variable2])
    # affichage
    _, ax = plt.subplots()
    df_filtre_tri.plot(kind='line', x='sent_at', y=variable1, color='blue', ax=ax, rot=30)
    df_filtre_tri.plot(kind='line', x='sent_at', y=variable2, color='red', ax=ax, rot=30)
    titre = "Corrélation (%s, %s) = %f" % (variable1, variable2, corr)
    plt.title(titre)
    plt.show()
```

La fonction « commande_correlation » permet de calculer l'indice de corrélation entre deux variables, et de l'afficher sur une courbe. Optionnellement, il est possible de préciser un intervalle de temps (date_deb, date_fin) grâce à la commande filtre.

• Fonction « commande_display_capteurs »

```
def commande_display_capteurs(df, args):
```

```
    variable = args.variable
    mode = args.mode
    display_capteurs(df, variable, mode, args.start_date, args.end_date)
def display_capteurs(df, variable, mode, date_deb, date_fin):
    date_deb, date_fin = intervale_dates(df, date_deb, date_fin)
    # filtrage par rapport aux dates
    df_filtre = df[(df['sent_at'] >= str(date_deb)) & (df['sent_at'] <= str(date_fin))]
    # tri des données selon les dates
    df_filtre_tri = df_filtre.sort_values(by=['sent_at'])
    # extraction des capteurs
    capteurs = df_filtre_tri['id'].unique()
    # choix d'un graphe fusionné ou séparé
    if mode=='f':
        _, ax = plt.subplots()
    else:
        ax = None
    # affichage de chaque capteur
    colors = ['r', 'b', 'g', 'k', 'm', 'y', 'b']
    for i, capteur in enumerate(capteurs):
        df_capteur = df_filtre_tri[df_filtre_tri['id'] == capteur]
        label = "capteur %s" % capteur
        df_capteur.plot(x="sent_at", y=variable, label=label, color=colors[i], ax=ax, rot=30)
    titre = "Capteurs : dimension %s" % (variable)
    plt.title(titre)
    plt.show()
```

La fonction « display_capteurs », non demandée dans le sujet, nous permet d'afficher la courbe d'une variable pour un ou plusieurs capteurs. Il est possible d'afficher les graphiques des 6 capteurs séparément ou ensemble : pour cela, il est nécessaire de préciser « f » (pour « fusionné », c'est-à-dire afficher les courbes de chaque capteur de façon superposée) ou « s » (pour « séparé ») au sein de la ligne de commande. On peut aussi préciser un intervalle de temps (date_deb, date_fin) grâce à la commande filtre. Cette fonction nous a aidé à appréhender les similarités entre certains capteurs.

• Fonction « commande_similarités »

```
def commande_similarités(df, args):
    variable = args.variable
    affichage = args.affichage
    similarités(df, variable, affichage, args.start_date, args.end_date)
def similarités(df, variable, affichage, date_deb, date_fin):
    date_deb, date_fin = intervale_dates(df, date_deb, date_fin)
    # filtrage par rapport aux dates
    df_filtre = df[(df['sent_at'] >= str(date_deb)) & (df['sent_at'] <= str(date_fin))]
    # extraction des capteurs
    nom_capteurs = df_filtre['id'].unique()
    # calcul moyenne journalière de chaque capteur
    capteurs_moy = []
    for i, capteur in enumerate(nom_capteurs):
        df_capteur = df_filtre[df_filtre['id'] == capteur]
        df_capteur_date = df_capteur.set_index('sent_at')
        df_capteur_date_moy = df_capteur_date.resample('D').mean()
        df_capteur_date_moy.fillna(method="bfill", inplace=True)
        capteurs_moy.append(df_capteur_date_moy[variable])
    if affichage == 'm':
        # affichage d'une matrice de similarité
        # creation du dataframe regroupant les données des capteurs en colonnes
```

```
        capteurs_concat = pd.concat(capteurs_moy, axis=1)
        capteurs_concat.columns = nom_capteurs
        # calcul des distances euclidiennes entre colonnes dans une matrice
        capteurs_matrice = capteurs_concat.to_numpy()
        dist_matrice = squareform(pdist(capteurs_concat.T, metric='euclidean'))
        # affichage de la matrice
        fig, ax = plt.subplots()
        cax = ax.matshow(dist_matrice, interpolation='nearest')
        fig.colorbar(cax)
        titre = "Distance Capteurs: dimension %s" % (variable)
        plt.title(titre)
        plt.show()
    elif affichage == 'c':
        # affichage des courbes
        fig, ax = plt.subplots()
        colors = ['r', 'b', 'g', 'k', 'm', 'y', 'b']
        labels = []
        for i, capteur_moy in enumerate(capteurs_moy):
            label = "capteur %s" % nom_capteurs[i]
            labels.append(label)
            capteur_moy.plot(x="sent_at", y=variable, label=label, color=colors[i], ax=ax, rot=30)
        handles, _ = ax.get_legend_handles_labels()
        plt.legend(handles=handles[1:], labels=labels)
        titre = "Capteurs: dimension %s" % (variable)
        plt.title(titre)
        plt.show()
```

La fonction «commande_similarités » permet de calculer la distance euclidienne, pour une variable donnée, entre chacun des capteurs. Il est possible d'afficher ces similarités grâce à une courbe ou une matrice. Pour cela, il est nécessaire de préciser « c » (pour courbe) ou « m » (pour matrice) au sein de la ligne de commande. On peut aussi préciser un intervalle de temps (date_deb, date_fin) grâce à la commande filtre.

III. Écriture du programme

```
#------------------------- MAIN -----------------------------------------------
COMMANDES_ACCEPTEES = ["display", "displayStat", "corrélation"]
if __name__ == '__main__':
    import argparse
    # devrait lire la première ligne du fichier csv pour récupérer les noms de variable et les
mettre dans choices
    _DESCRIPTION = "TP EIVP"
    parser = argparse.ArgumentParser(description=_DESCRIPTION)
    # parser.add_argument("commande", help=f"commande parmi {COMMANDES_ACCEPTEES}")
    subparsers = parser.add_subparsers(help="commande 'display', 'displayStat' ou 'corrélation'",
dest='subparser_name')
    # parse pour la commande display
    parser_a = subparsers.add_parser('display', help="display variable [start_date] [end_date]")
    parser_a.add_argument("variable", choices=VARIABLES, help="nom de la variable à plotter")
    parser_a.add_argument('start_date', nargs='?', type=date.fromisoformat, help="start_date
(optionnel)")
    parser_a.add_argument('end_date', nargs='?', type=date.fromisoformat, help="end_date
(optionnel)")
    parser_a.set_defaults(func=commande_display)
```

```
    # parse pour la commande displayStat
    parser_b = subparsers.add_parser('displayStat', help="displayStat variable [start_date]
[end_date]")
    parser_b.add_argument("variable", choices=VARIABLES, help="nom de la variable à plotter")
    parser_b.add_argument('start_date', nargs='?', type=date.fromisoformat, help="start_date
(optionnel)")
    parser_b.add_argument('end_date', nargs='?', type=date.fromisoformat, help="end_date
(optionnel)")
    parser_b.set_defaults(func=commande_displayStat)
    # parse pour la commande corrélation
    parser_c = subparsers.add_parser('corrélation', aliases=['corr', 'correlation'],
help="corrélation variable1 variable2 [start_date] [end_date]")
    parser_c.add_argument("variable1", choices=VARIABLES, help="nom de la variable à plotter")
    parser_c.add_argument("variable2", choices=VARIABLES, help="nom de la variable à plotter")
    parser_c.add_argument('start_date', nargs='?', type=date.fromisoformat, help="start_date
(optionnel)")
    parser_c.add_argument('end_date', nargs='?', type=date.fromisoformat, help="end_date
(optionnel)")
    parser_c.set_defaults(func=commande_correlation)
    # parse pour la commande display_capteurs
    parser_a = subparsers.add_parser('display_capteurs', help="display_capteurs variable fusion
[start_date] [end_date]")
    parser_a.add_argument("variable", choices=CAPTEURS, help="nom de la variable à plotter")
    parser_a.add_argument('mode', choices=['f', 's'], help="type d'affichage: fusion ou séparation")
    parser_a.add_argument('start_date', nargs='?', type=date.fromisoformat, help="start_date
(optionnel)")
    parser_a.add_argument('end_date', nargs='?', type=date.fromisoformat, help="end_date
(optionnel)")
    parser_a.set_defaults(func=commande_display_capteurs)
    # parse pour la commande similarités
    parser_a = subparsers.add_parser('similarités', help="similarités variable affichage
[start_date] [end_date]")
    parser_a.add_argument("variable", choices=VARIABLES, help="nom de la variable à plotter")
    parser_a.add_argument('affichage', choices=['m', 'c'], help="type d'affichage: matrice ou
courbes")
    parser_a.add_argument('start_date', nargs='?', type=date.fromisoformat, help="start_date
(optionnel)")
    parser_a.add_argument('end_date', nargs='?', type=date.fromisoformat, help="end_date
(optionnel)")
    parser_a.set_defaults(func=commande_similarités)
    args = parser.parse_args()
    if args.subparser_name:
        df = lecture_fichier(FICHIER)
        args.func(df, args)
    else:
        print("commande 'display', 'displayStat', 'corrélation' ou 'similarités'")
```

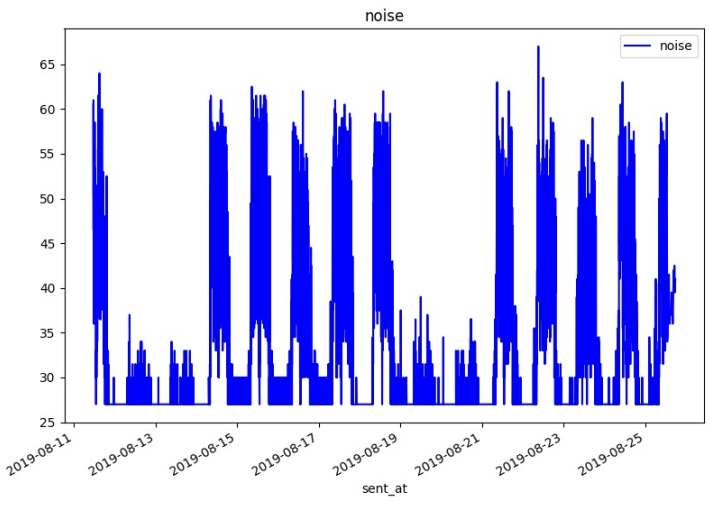
Pour l'écriture globale du programme, nous avons décidé d'utiliser le module « parser » car celui-ci permettait de créer un programme assez compact. Les quatre commandes acceptées par le programme sont les suivantes : « display », « displayStat », « corrélation » et « similarités ». L'utilisation du module « parser » nous a facilité la rédaction des messages d'erreur, du filtrage des dates et des indications d'affichage.

IV. Tests

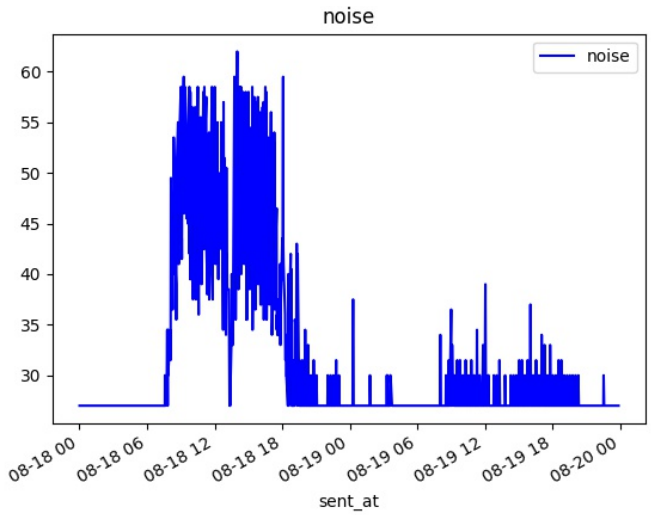
Les quatre commandes pouvant être exécutées sont les commandes « display », « displayStat », « corrélation » et « similarités ». Pour l'ensemble de ces commandes, il est possible de préciser un intervalle de temps.

- Commande « display »

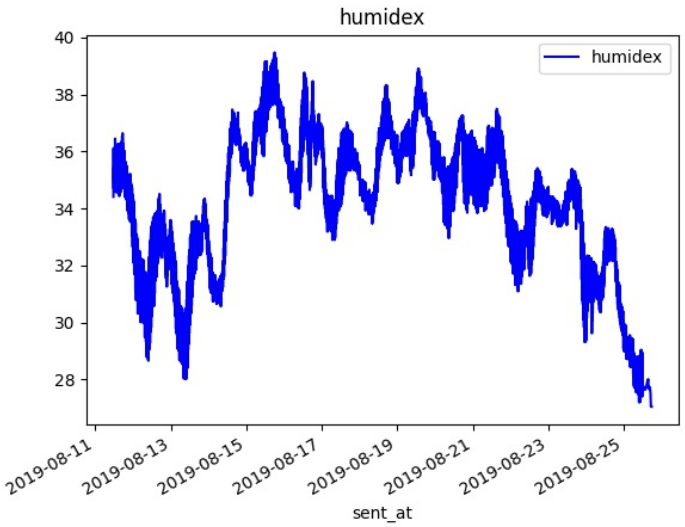
Tests avec différentes variables, et avec des intervalles de dates différents.



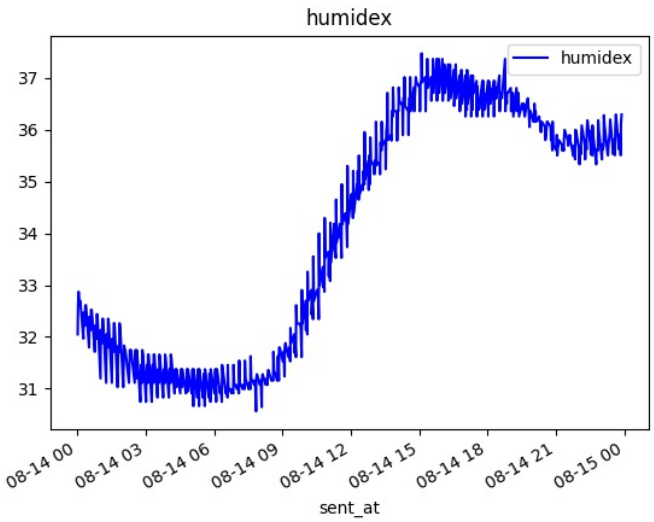
Exemple avec la variable « noise » sans précision de date.



Exemple avec la variable « noise» avec intervalle de temps.



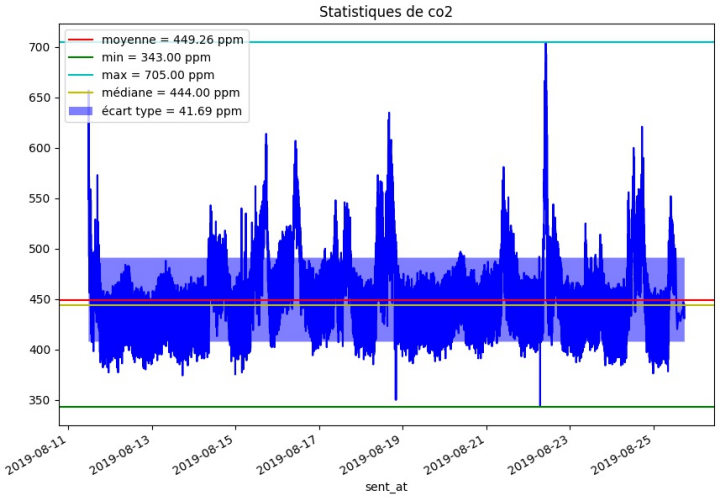
Exemple avec la variable « humidex » sans précision de date.



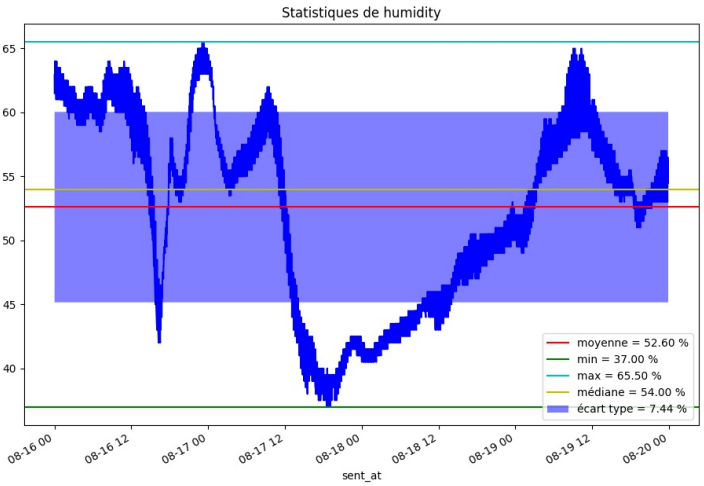
Exemple avec la variable « humidex » avec intervalle de temps.

- Commande « displayStat »

Tests avec différentes variables, et avec des intervalles de dates différents.



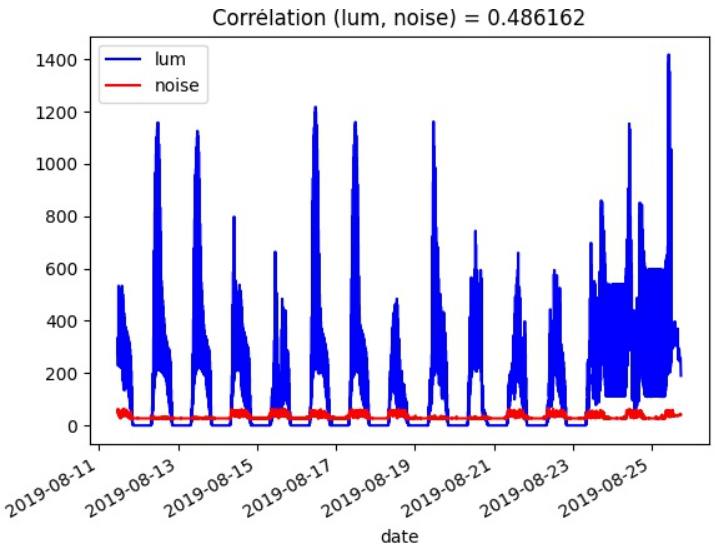
Exemple avec la variable « co2» sans précision de date.



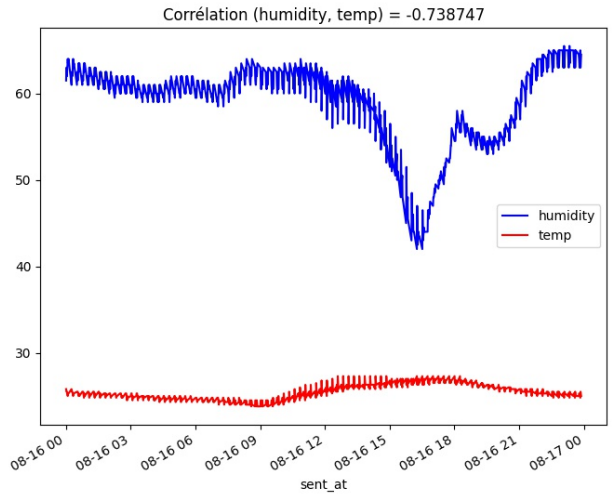
Exemple avec la variable « humidity » avec intervalle de temps.

- Commande « corrélation »

Tests avec différentes variables, et avec des intervalles de dates différents. Nous avons utilisé la moyenne dite de « Spearman » pour calculer les indices de corrélation. En effet, cette moyenne permet d'estimer la corrélation entre deux variables qui peuvent être liées mais de façon autre que linéaire. Par exemple, les calculs des indices de corrélation impliquant la variable « noise » s'appuient sur une échelle logarithmique. Pour la commande displayStat, nous n'avions pas besoin de tenir compte de cette échelle, cette commande n'impliquant qu'une seule variable.



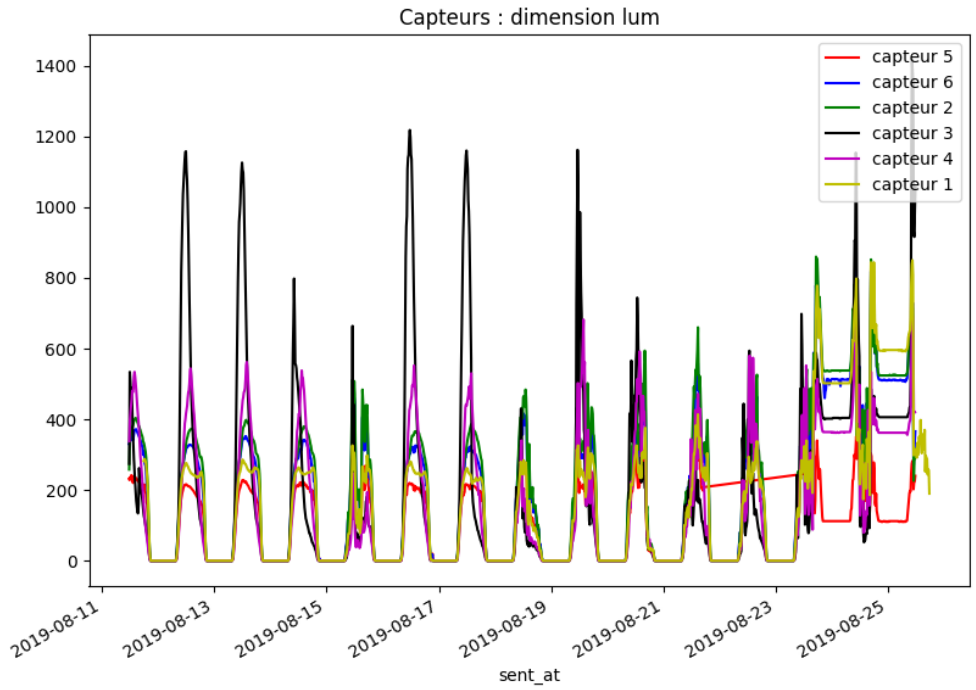
Exemple avec « lum» et « noise » sans précision de date.



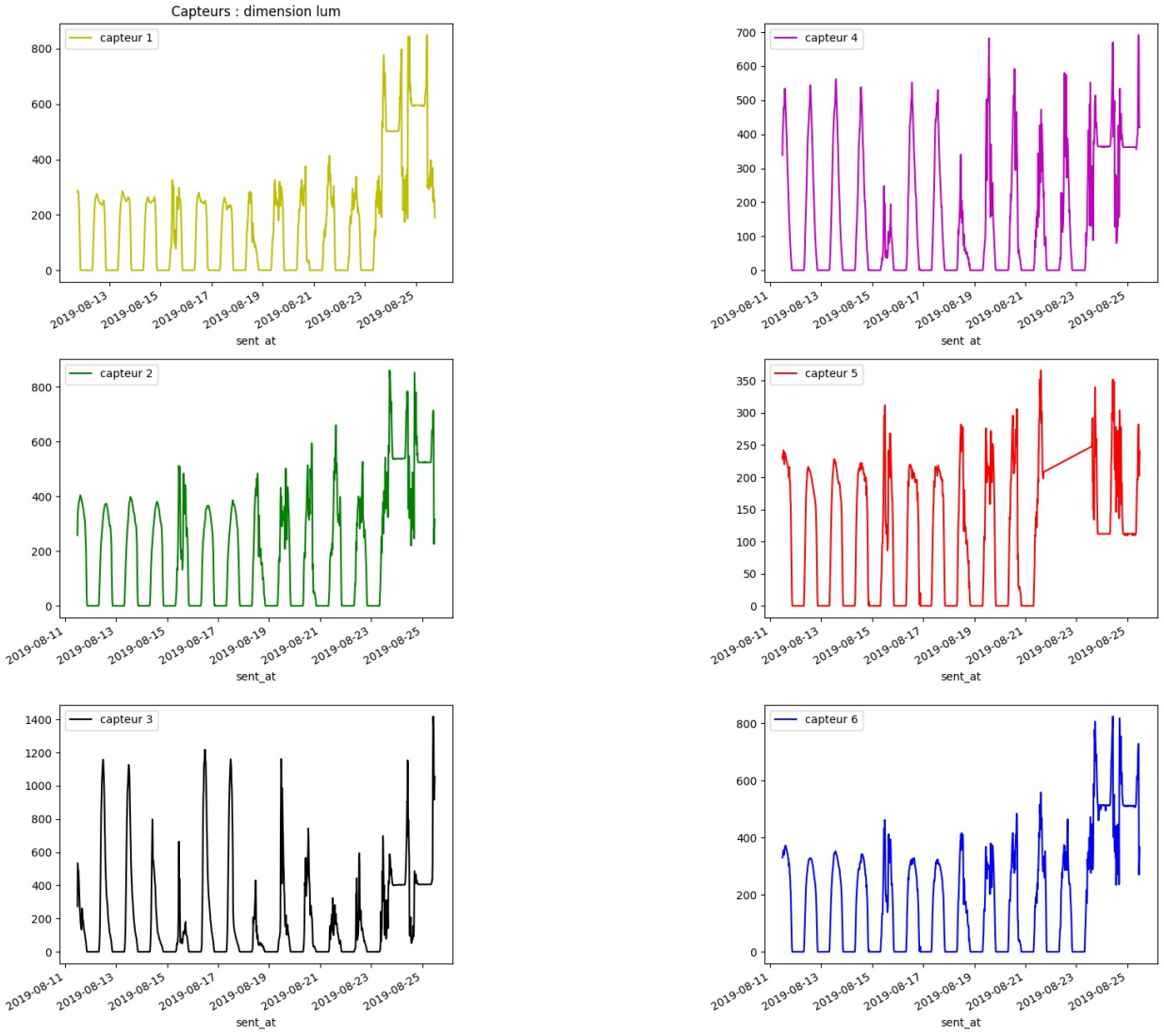
Exemple avec « humidity » et « temp » avec intervalle de temps.

- Commande « display_capteurs »

Tests avec différentes variables, et avec des modes d'affichage différents. Il est également possible de faire des tests en précisant des dates.



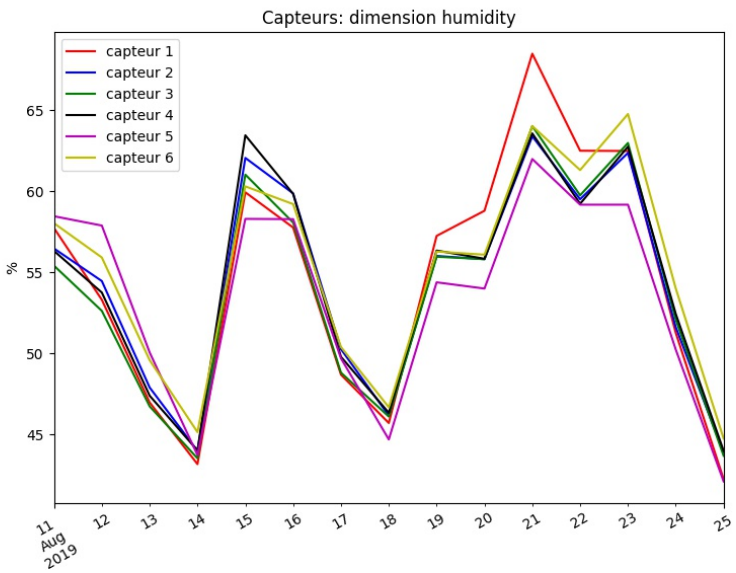
Exemple avec l'affichage superposant les courbes de chaque capteur



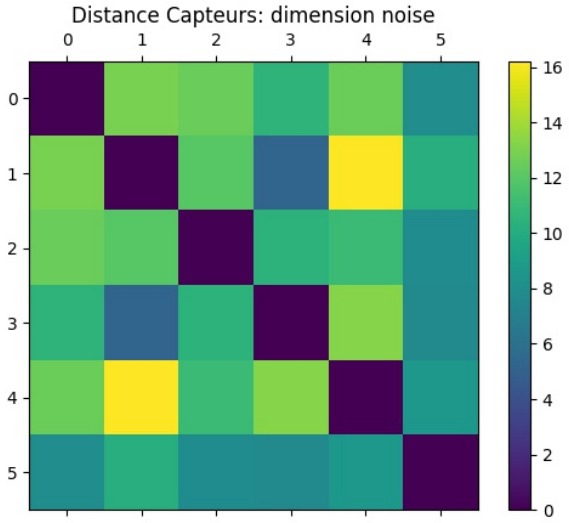
Exemple avec l'affichage séparant les courbes pour chaque capteur

• Commande « similarités »

Tests avec différentes variables, et avec des intervalles des modes d'affichage différents. Il est également possible de faire des tests en précisant des dates.



Exemple avec « humidity » et affichage avec une courbe.



Exemple avec « noise» et affichage d'une matrice.

INTERPRÉTATION

Nous n'avons pas réalisé la question bonus, qui consistait à déterminer les horaires d'occupation des bureaux. Cependant, en analysant la corrélation entre les variables lumière (lum), bruit (noise) et le dioxyde de carbone (co2). En comparant ces variables deux-à-deux, nous avons pu déterminer différents horaires et périodes : jour/nuit, jours en semaine/jours de week-end.
De plus, les anomalies de certains capteurs étaient visibles.

Pour affiner ces données, nous avons voulu estimer la position cardinale de chaque capteur. Pour cela, nous utilisé la fonction display_capteurs avec la variable lumière (lum), et la commande similarités avec les variables bruit (noise) et la variable sur le dioxyde de carbone (co2).

Avec cette analyse sommaire des données, nous avons établir quelques suppositions concernant la position de chaque capteur au sein du bâtiment.

Concernant la position des capteurs par rapport aux sources lumineuses, on peut supposer d'après les courbes, que les capteurs 3 et 4 se situeraient près de fenêtres exposées à l'est.
Concernant la position des capteurs par rapport aux émissions de co2, on peut supposer que les capteurs 2 et 4 sont assez éloignés des postes de travail.

Le capteur 5 présente parfois des données assez éloignées des autres capteurs. On peut donc penser qu'il présente des anomalies, ou qu'il est situé loin des fenêtres et porche des postes de travail. Pour le reste des capteurs, certaines données nous paraissent assez difficiles à mettre en relation.

DIFFICULTÉS RENCONTRÉES

Avant ce projet de programmation informatique, nous n'avions jamais fait de programmation. En effet, étudiants en double-cursus architecte-ingénieur, nous n'avons pas fait de classes préparatoires, au sein desquelles des cours d'algorithme et de programmation informatique sont dispensés.

Au commencement de cet exercice, il nous a fallu apprendre les bases de l'algorithmie et de la programmation informatique. Nous nous sommes familiarisés avec les notions de boucles, de variables, de booléens, de chaînes, de listes…etc. Afin de commencer le projet rapidement et efficacement, nous avons essayé d'acquérir des connaissances sur les différentes notions utiles à la réalisation de celui-ci.

Se familiariser et comprendre les notions basiques de la programmation informatique a plutôt été aisé. En effet, c'est plutôt lors de la création des fonctions et du programme que nous avons éprouvé quelques difficultés.

La difficulté majeure rencontrée lors de l'élaboration du programme a été de distinguer la commande corrélation de la commande similarités. En effet, la commande corrélation donne une mesure de l'intensité et du sens de la relation linéaire entre deux variables. La commande similarités effectue, pour une variable définie, une moyenne journalière pour chaque capteur, et compare celle-ci avec les moyennes des autres capteurs. Nous avons également eu quelques difficultés à régler les dates (dates croissantes, dates non valides…) et à afficher les messages d'erreurs au bon endroit au sein du code. De plus, certaines données nous ont parues difficiles à mettre en corrélation et à analyser, comme par exemple la lumière et l'humidité.

Pour parvenir à résoudre ces difficultés, nous avons consulté de nombreux sites internet à propos du langage python et du package pandas.

À l'inverse, certaines actions nous ont paru plutôt aisées à mettre en oeuvre : modification du graphisme des courbes, ajout d'une colonne de données dans le tableur initial, filtrage par rapport aux dates…

CONCLUSION

Ce projet nous a permis de nous familiariser avec les fonctions basiques du langage Python (importation de fichier, mise en forme des données, commandes essentielles…).

Nous avons choisi d'utiliser les bibliothèques existantes (Pandas, Numpy…) pour la praticité du programme. Nous pensons néanmoins que l'utilisation de ces bibliothèques peut faire perdre en performance et rapidité au programme.

La première version du programme contenait davantage de lignes de codes et davantage d'informations (annotations, commentaires…). Nous avons décidé, afin d'améliorer la clarté et la visibilité du programme, de le rendre le plus compact possible. C'est pourquoi nous avons utilisé le module « parser ».