

ROMDHANE Adem
CHERIF Amine

SAe_1.05

Dans le cadre de cette SAé, nous avons choisie de prendre la problématique :

Projet n°16 : déterminer les zones les plus enneigées

Nous avons choisie de faire en sorte de sélectionner un csv au choix pour faire en sorte de calculer ses villes qui ont le plus de hauteur de neige(calcul de moyen et de hauteur min max)

Pour ce projet nous avons choisie de prendre les fichiers csv :

Données climatologiques de base – quotidiennes

Ceux-là avaient une partie neige, nous avons choisie de prendre la hauteur de la neige car nous avons jugé que c'était le meilleur moyen de savoir les zones les plus enneigées. Dans le csv il est représenté comme HNEIGEF, qui veut dire Hauteur de neige fraîche, il se situe dans le row[56]. Nous avons essayé de faire une interface graphique pour rendre le rendu final plus agréable.

Voici l'ensemble des documents texte et vidéo qui nous ont servi pour ce projet :

Playlist :

Python GUI's With Tkinter

<https://youtube.com/playlist?list=PLCC34OHNc0toC6GglhF3ncJ5rLwQrLGnV&si=0t8hH-YekvSgwrJK>



Documentation python :

<https://docs.python.org/fr/3.13/library/tk.html>

Nous avons à la fois choisi de traiter à la fois la hauteur de la neige d'une ville sur un intervalle de temps choisie par l'utilisateur et le calcul de de la ville qui avait le plus de neige (min, moyen et max) pour le csv choisi par l'utilisateur, il aura le choix entre ses 2 possibilité avec le csv sélectionner.

Explication du code:

Nous avons choisi de crée une interface, nous nous sommes documentées principalement sur youtube car nous découvrons la chose, nous allons à la fois expliquer le code et montrer comment nous l'avons compris:

```
import csv
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
from tkinter import filedialog
```

Pour créer une application avec une interface utilisateur graphique, on utilise Matplotlib pour générer des graphiques et Tkinter pour concevoir l'interface. Grâce à ttk , des widgets avancés comme les combobox peuvent être intégrés. De plus, on peut afficher des boîtes de dialogue interactives pour, par exemple, ouvrir des fichiers ou confirmer des actions

```

table = []
période = []
ville1 = set()

def charger_donnees():
    global table, ville1
    fichiers = filedialog.askopenfilenames(
        title="Sélectionnez les fichiers CSV",
        filetypes=[("Fichiers CSV", "*.csv")]
    )
    if not fichiers:
        messagebox.showerror("Erreur", "Aucun fichier sélectionné.")
        return

    table.clear()
    ville1.clear()

    for fichier in fichiers:
        try:
            with open(fichier, newline='', encoding='utf-8') as csvfile:
                reader = csv.reader(csvfile, delimiter=',')
                for row in reader:
                    table.append(row)
                    if len(row) > 1:
                        ville1.add(row[1].lower())
        except Exception as e:
            messagebox.showerror("Erreur", f"Impossible de lire {fichier}: {str(e)}")

    ville_combobox['values'] = sorted(ville1)
    messagebox.showinfo("Chargement réussi", "Les données ont été chargées avec succès!")

```

Fonction permettant à l'utilisateur de charger des fichiers CSV et d'extraire les noms de villes uniques. L'utilisateur sélectionne les fichiers à l'aide d'une boîte de dialogue. Les villes extraites sont ajoutées dans une box. Si une ligne contient au moins deux colonnes (Cette vérification permet de s'assurer que seules les lignes ayant des informations valides sont traitées), le nom de la ville est extrait, converti en minuscule et ajouté à l'ensemble ville1 pour garantir l'unicité des villes. En cas d'erreur de lecture, un message d'erreur est affiché. La box est mise à jour avec les villes disponibles triées par ordre alphabétique.

Vidéo:

https://youtu.be/Aim_7fC-inw?si=QGXpb9Fb8yHKqFzR



```
def afficher_graphique():
    global période
    ville = ville_combobox.get().lower()
    datemin = datemin_entry.get()
    datemax = datemax_entry.get()

    if not ville or not datemin or not datemax:
        messagebox.showerror("Erreur", "Veuillez remplir tous les champs.")
        return
    if ville not in ville1:
        messagebox.showerror("Erreur", f"La ville '{ville}' n'est pas valide. Sélectionnez une ville dans la liste ou entrez une ville valide.")
        return

    période = []
    for row in table:
        if len(row) > 50 and row[1].lower() == ville and datemin <= row[5] <= datemax:
            try:
                hauteur_neige = float(row[50]) if row[50].strip() else None
                if hauteur_neige is not None:
                    période.append((row[5], hauteur_neige))
            except ValueError:
                continue
```

La fonction `afficher_graphique()` vérifie d'abord si tous les champs (ville, date de début et date de fin) sont remplis. Si l'un des champs est vide ou si la ville n'est pas valide, un message d'erreur s'affiche pour informer l'utilisateur. Ensuite, elle parcourt les données pour extraire les dates et hauteurs de neige correspondantes à la ville et à la période spécifiées. Si une hauteur de neige est valide (convertible en nombre flottant), elle est ajoutée à la liste `période` avec la date associée. Si une erreur de conversion survient (par exemple, une valeur non numérique ou manquante), elle est ignorée, et la ligne concernée est passée.

```

if période:
    période.sort(key=lambda x: x[0])
    dates = [item[0] for item in période]
    hauteurs = [item[1] for item in période]
    step = max(1, len(dates) // 10)
    affichage_dates = [dates[i] for i in range(0, len(dates), step)]

    plt.figure(figsize=(10, 6))
    plt.plot(dates, hauteurs, marker='o', label="Hauteur de neige")
    plt.title(f"Évolution de la hauteur de neige à {ville} entre {datemin} et {datemax}")
    plt.xlabel("Date")
    plt.ylabel("Hauteur de neige (cm)")
    plt.xticks(ticks=range(0, len(dates), step), labels=affichage_dates, rotation=45, fontsize=8)
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.legend()
    plt.tight_layout()
    plt.show()
else:
    messagebox.showinfo("Aucune donnée", f"Aucune donnée trouvée pour {ville} entre {datemin} et {datemax}.")

```

Ce code trie une liste de période en fonction de la première valeur de chaque tuple (les dates). Ensuite, il extrait deux listes séparées : date pour les premières valeurs (dates) et hauteurs pour les deuxièmes valeurs (hauteurs). Enfin, il sélectionne un sous-ensemble des dates, affichage_dates en prenant une date toutes les étapes pour limiter l'affichage à environ 10 dates. Cette partie a été rajoutée à la fin car nous avons remarqué que sur un intervalle trop grand, les dates à l'abscisse du graphique étaient trop nombreuses et donc illisibles. On affiche le graphique.

```

def zones_les_plus_enneigees():
    if not table:
        messagebox.showerror("Erreur", "Aucune donnée chargée. Veuillez charger les fichiers CSV d'abord.")
        return

    neige_par_ville = {}
    for row in table:
        if len(row) > 50:
            try:
                ville = row[1].lower()
                hauteur_neige = float(row[50]) if row[50].strip() else 0
                if ville in neige_par_ville:
                    neige_par_ville[ville].append(hauteur_neige)
                else:
                    neige_par_ville[ville] = [hauteur_neige]
            except ValueError:
                continue

    villes_avec_stats = []
    for ville, hauteurs in neige_par_ville.items():
        totale = sum(hauteurs)
        moyenne = totale / len(hauteurs)
        maximale = max(hauteurs)
        villes_avec_stats.append((ville, totale, moyenne, maximale))

```

Classement des hauteurs de neiges:

Les hauteurs de neige sont regroupées par ville dans un dictionnaire nommé neige_par_ville. Pour chaque ville, la fonction calcule trois statistiques : la hauteur totale, la moyenne et la hauteur maximale de neige enregistrée. Ces résultats sont stockés dans une liste qui fournit un aperçu des zones les plus enneigées.

```

villes_avec_stats.sort(key=lambda x: x[3], reverse=True)
resultats = "Zones les plus enneigées (basées sur la hauteur maximale de neige) :\n"
for i, (ville, totale, moyenne, maximale) in enumerate(villes_avec_stats[:5]):
    resultats += f"{i+1}. {ville.capitalize()} - Max : {maximale} cm, Moyenne : {moyenne:.2f} cm, Total : {totale:.2f} cm\n"

messagebox.showinfo("Zones les plus enneigées", resultats)

```

Sélectionne les 5 premières ligne et génère un texte affiché pour chacune : le rang, le nom, la hauteur maximale, la moyenne, et le total de neige, avec un format clair et précis. Cela met en évidence les zones les plus enneigées.

```

root = tk.Tk()
root.title("Analyse de la hauteur de neige")
file_frame = tk.Frame(root)
file_frame.pack(pady=10)
load_button = tk.Button(file_frame, text="Charger les fichiers CSV", command=charger_donnees)
load_button.pack()
ville_frame = tk.Frame(root)
ville_frame.pack(pady=10)
tk.Label(ville_frame, text="Choisissez ou entrez une ville :").pack(side=tk.LEFT)
ville_combobox = ttk.Combobox(ville_frame, width=30)
ville_combobox.pack(side=tk.LEFT)
date_frame = tk.Frame(root)
date_frame.pack(pady=10)
tk.Label(date_frame, text="Date de départ (aaaaammjj) :").grid(row=0, column=0, padx=5, pady=5)
datemin_entry = tk.Entry(date_frame, width=15)
datemin_entry.grid(row=0, column=1, padx=5, pady=5)
tk.Label(date_frame, text="Date limite (aaaaammjj) :").grid(row=1, column=0, padx=5, pady=5)
datemax_entry = tk.Entry(date_frame, width=15)
datemax_entry.grid(row=1, column=1, padx=5, pady=5)
graph_button = tk.Button(root, text="Afficher le graphique", command=afficher_graphique)
graph_button.pack(pady=10)
zones_button = tk.Button(root, text="Zones les plus enneigées", command=zones_les_plus_enneigees)
zones_button.pack(pady=10)
root.mainloop()

```

Ce morceau de code contient:

La création de l'interface Tkinter

La Sélection des fichiers

La Sélection ou saisie de la ville

L'Entrée des dates

Boutons pour les graphiques et les zones enneigées

Et enfin le lancement de l'interface.

Fonctionnement :

Vous pouvez prendre n'importe quel fichier tant qu'il est quotidien et qu'il comprend les paramètres:

Données climatologiques de base - quotidiennes

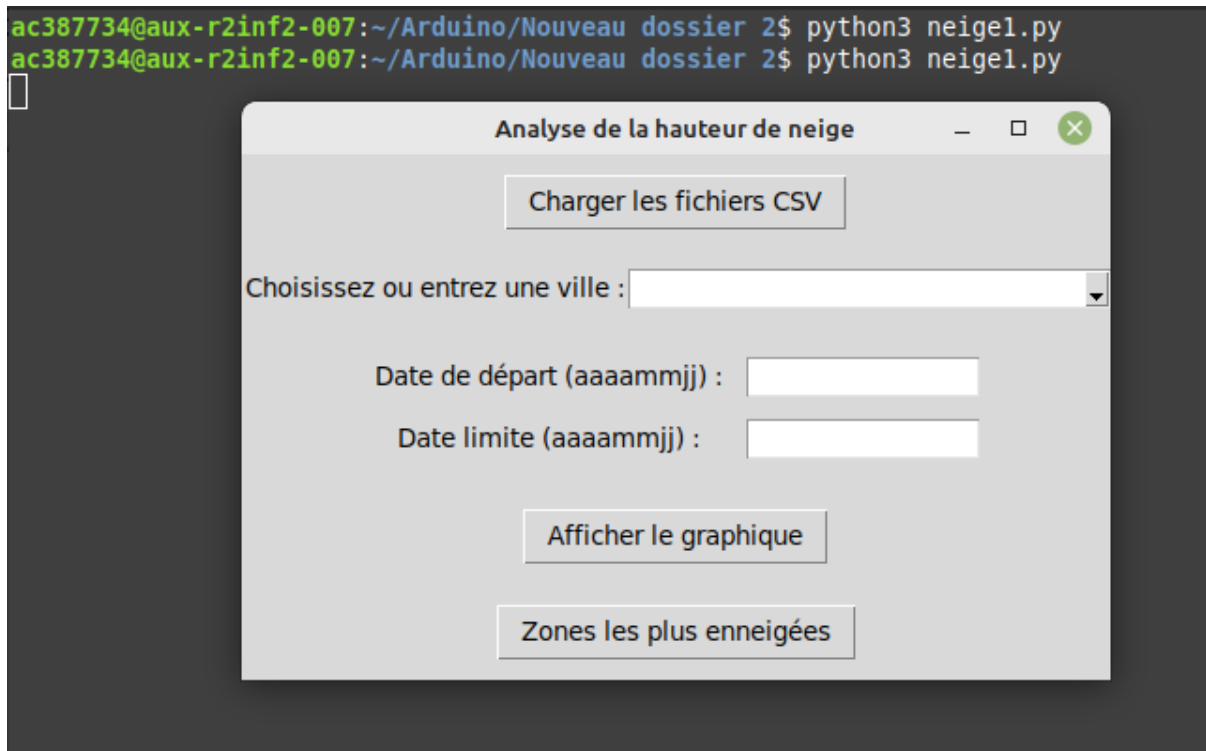
Présentation

📄 QUOT_departement_01_période_1950-2023_autres-parametres
Mis à jour il y a 16 jours — csv.gz (4.2Mo) — 1544 téléchargements

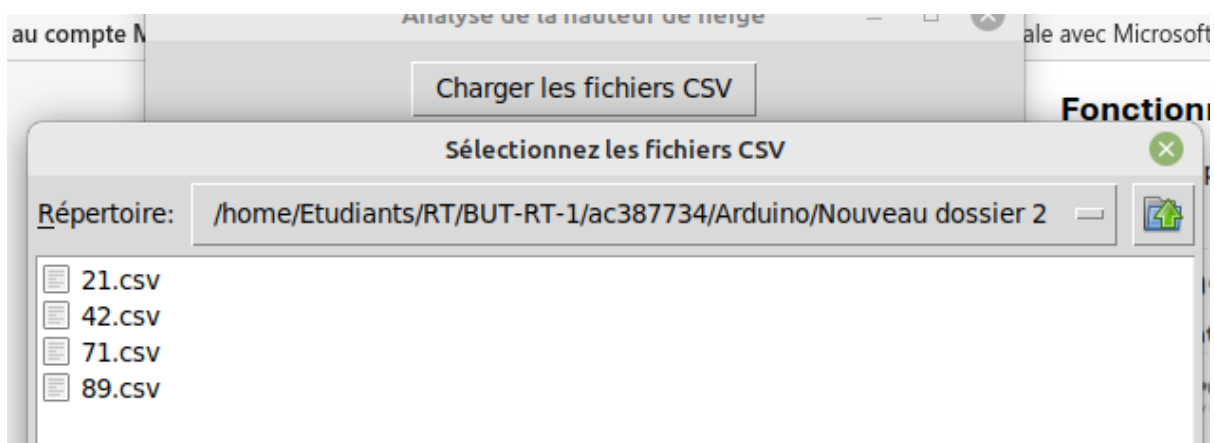
▼ Voir les données



Nous avons choisie de prendre les 4 département de la bourgogne et de les renommés avec leur n° de département. Voici ce que fait notre code une fois run (sur linux) :



L'interface s'affiche avec ses différents boutons.



On sélectionne le csv souhaité, ici ça sera 89. Si le csv est bien chargé, il y aura un message de confirmation.

Analyse de la hauteur de neige

Charger les fichiers CSV

Choisissez ou entrez une ville : auxerre

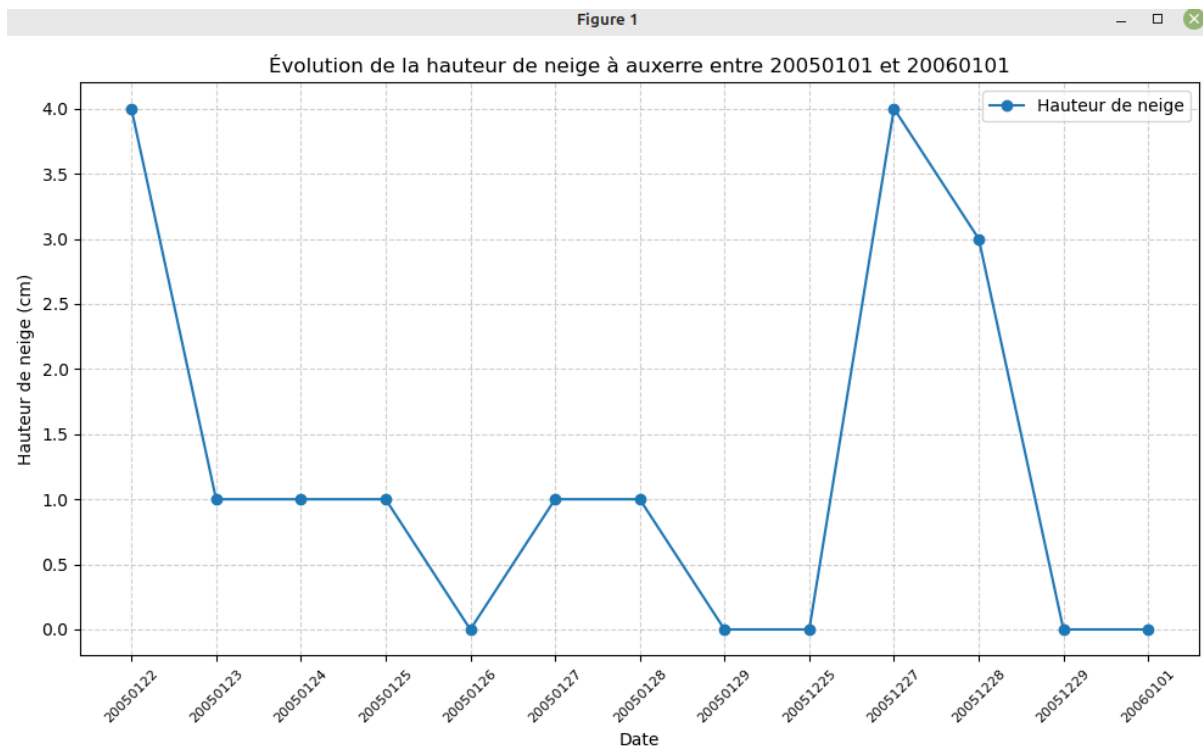
Date de départ (aaaaammjj) : 20050101

Date limite (aaaaammjj) : 20060101

Afficher le graphique

Zones les plus enneigées


On sélectionne Auxerre et on a choisi un intervalle d'un an (prioriser les intervalles courts pour un graphique plus lisible)



Voici un exemple de graphique que l'on peut avoir.

On sélectionne maintenant “zones les plus enneigées”.

Zones les plus enneigées



Zones les plus enneigées
(basées sur la hauteur
maximale de neige) :

1. Cruzy_sapc - Max : 16.0 cm,
Moyenne : 0.00 cm, Total :
45.00 cm

2. Quarre-les-tombes - Max :
16.0 cm, Moyenne : 0.04 cm,
Total : 393.00 cm

3. Auxerre - Max : 15.0 cm,
Moyenne : 0.01 cm, Total :
173.00 cm

4. St leger vauban - Max : 15.0
cm, Moyenne : 0.03 cm, Total :
240.00 cm

5. Arces - Max : 13.0 cm,
Moyenne : 0.01 cm, Total :
147.00 cm

OK

Nous avons maintenant une liste des zones sellons leur hauteur d'enneigement.