LES TROPHÉES NSI

Édition 2023

DOSSIER DE CANDIDATURE
PRÉSENTATION DU PROJET

Ce document est l'un des livrables à fournir lors du dépôt de votre projet : 4 pages maximum (hors documentation).

Pour accéder à la liste complète des éléments à fournir, consultez la page **Préparer votre participation**.

Vous avez des questions sur le concours ? Vous souhaitez des informations complémentaires pour déposer un projet ? Contactez-nous à **info@trophees-nsi.fr**.

NOM DU PROJET : PySky

> PRÉSENTATION GÉNÉRALE :

La spécialité Numérique et Science Informatique (NSI) nous rappelle ce qu'il y a de plus beau avec l'informatique : nous pouvons l'utiliser en coordination avec toute discipline qui nous passionne, nous donnant les clés nécessaires pour créer nos propres projets. C'est avec cela en tête que nous avons entrepris PySky, un projet à la croisée de la data science et de la météorologie. L'idée originale était de cartographier des données extraites en ligne, via des Application Programming Interface (API), à l'aide de Python et de librairies en open source. Mais au cours d'améliorations successives, notre projet devint une interface complète, une alternative open source aux nombreux sites de météo ; elle répertorie maintenant des données quantitatives (i.e. température, précipitation) obtenues et des données qualitatives (i.e. "Sunny", "Cloudy", "Rainy", etc) obtenues à partir d'analyses qualitatives d'images à travers la France Métropolitaine, tout en offrant la possibilité de remonter dans le temps.

> ORGANISATION DU TRAVAIL :

Nom	Rôle
Ava	Traitement des données et de leur préparation pour la cartographie
Oscar	Mise en place des analyses qualitatives ; développement de la page web
Paul	Valorisation de l'équipe: vidéo et présentation ; Mise en place des modèles climatologiques
Benjamin	Extraction des données et APIs ; débuggage liée aux requêtes utilisateurs
Hippolyte	Recherche de l'API ; Projection des données sur la carte avec Folium

LES ÉTAPES DU PROJET :

(1) La naissance de PySky

Ayant déjà fait plusieurs projets de NSI en collaboration les uns avec les autres, il n'était que logique de garder l'équipe actuelle. Mais ce n'est pas pour autant que le choix du projet était aussi simple. En effet, nous avions tous des idées différentes, toutes aussi intéressantes.

La météorologie est un rouage fondamental de notre société, abordé en Enseignement Scientifique, et la grande majorité des systèmes sur lesquels nous dépendons quotidiennement en dépendent : sécurité des transports, agriculture, production énergétique, etc. Et cette importance ne fait que croître, avec des évènements climatiques extrêmes en raison du changement climatique. Mais cette demande de notre société n'a pas toujours son offre. En effet, même si le grand public peut accéder aux données fiables, ces dernières sont cryptiques si elles ne sont pas cartographiées, et les interfaces de cartographies en ligne sont en closed source, à but lucratif, et ne montrent qu'un nombre limité de données.

C'est donc en se rappelant de l'importance de la météorologie et de tous les liens que nous pourrions faire avec le programme de NSI que nous nous sommes résolus à créer PySky, une interface météorologique en open source cartographiant des bases de données en lignes.

(2) La mise en place du projet : la répartition des tâches

Dès le début, on savait que l'idée de base de notre projet comportait 3 parties principales: (1) on extrait les données météorologiques en ligne en utilisant des APIs open source, (2) on les traite sous le format Comma Separated Values (.csv) pour (3) les mettre sous forme de carte thématique choroplèthe. Nous avons chargé Benjamin et Hippolyte de la première étape, et Oscar, Paul et Ava des deux dernières.

Nous savions par avance quels éléments du programme nous pourrions utiliser à notre avantage, notamment des éléments du programme de Terminale que nous pouvions apprendre en avance de par les TPs que nous a offert notre professeur. C'est ainsi que nous avons appris à utiliser la librairie Folium comme base de la cartographie en python ou à utiliser des APIs. Quant au programme de première, nous pouvons utiliser nos compétences en programmation Python et en traitement de données .csv. Au cours des évolutions successives de notre projet, d'autres parties du programme de première nous sont venu en aide, comme l'algorithme de classification K-Nearest Neighbours (KNN) à plusieurs reprises ou l'utilisation de librairie Python dont nous savions lire les spécifications.

(3) Développement des bases du projet

Nous avons commencé par rechercher les APIs nécessaires pour obtenir les données météorologiques voulues. Ici, l'obstacle était de trouver les données que l'on voulait qui correspondaient à un certain standard de précision, pour l'entièreté de la France, et ce avec une API rapide et gratuite. Nous avons, au début, utilisé les données du site api.met.no pour la température.

Ayant obtenu des données qualitatives nous devons désormais les traiter. Nous associons toutes les données à leurs numéros de département défini par l'INSEE. Nous obtenons ainsi un tableau sous le format csv avec nos données quantitatives en direct.

Pour cartographier les données, nous nous inspirons du système Geographic Information System (GIS), permettant de superposer plusieurs couches des données et de les afficher ou non. Il nous faut d'abord une carte de référence sur laquelle on superpose toutes les données : nous avons choisi celle offerte par OSM. Ensuite, nous affichons nos données quantitatives, comme la température ou la précipitation, sous forme de carte choroplèthe, c'est à dire que la couleur du département, dont les contours des départements sont définis par un fichier GeoJSON en open source, dépend de l'intensité de la mesure. Nous obtenons ainsi différentes couches choroplèthes qui peuvent être affichées ou non.

(4) Implémentation successives de fonctionnalités supplémentaires

En découvrant que le site infoclimat.fr propose des caméras offrant des prises du ciel à travers la france métropolitaine, il nous est venu à l'esprit d'utiliser ces images pour établir une mesure qualitative de la météo (i.e. "Sunny", "Cloudy", "Rainy", "Foggy", "Night"). Pour ce faire, nous avons utilisé une méthode d'analyse proche du "visual sentiment analysis", en nous inspirant de la publication de l'ANITS "Weather Forecasting using Digital Image Processing. Notre protocole est le suivant : (1) extraire l'image en direct, (2) tronquer l'image pour que le ciel soit l'élément central, (3) calculer l'intensité moyenne en "grayscale" de l'image pour (4) en déduire l'intensité nuageuse du ciel, (5) et donc une estimation qualitative des conditions météorologiques avec l'algorithme de classification KNN. Pour afficher ces évaluations qualitatives de la météo, nous utilisons des icônes placées au milieu de chaque département. L'utilisateur peut appuyer sur cette icône pour accéder aux détails des données.

L'intention du projet étant en partie d'offrir à l'utilisateur plus de données que les interfaces cartographiques en ligne, nous nous devions d'offrir plus de couches choroplèthe à l'utilisateur. C'est ainsi que nous nous sommes mis à la recherche d'une nouvelle API qui nous permettrait d'extraire et de traiter plus de données ; nous avons trouvé les APIs de open-meteo.com. Pour clarifier les scripts d'extraction, nous avons aussi pris les données de température qui étaient prises avec la première API et les avons remplacées avec celles de cette nouvelle API. Nous avons ainsi ajouté 10 mesures supplémentaires à notre carte.

Avec toutes ces données, une nouvelle possibilité s'offrait à nous : implémenter des modèles de prédiction météorologiques complexes. Nous nous sommes donc intéressés aux événements climatiques extrêmes communs en France rendus de plus en plus fréquents en raison du changement climatique, comme la sécheresse, la canicule, ou les inondations.

Pour la sècheresse, nous avons utilisé la mesure proposée à la page 13 du rapport n°1173 de l'Organisation Météorologique Mondiale (OMM) intitulé "Manuel des indicateurs et indices de sécheresse" : multiplie le ratio de la précipitation actuelle et de la précipitation "normale" par 100,

nous donnant un pourcentage, avec des pourcentages plus bas correspondants à un risque accrue de sécheresse.

Pour la canicule, puisque cette dernière se définit à partir d'un seuil de température différente pour chaque département, nous avons pris les seuils offert par le rapport "Système d'alerte canicule et santé : principes, fondements et évaluation" de l'Institut de Veille Sanitaire et comparé ce seuil et la température maximale du jour par département, nous rendant une valeur booléen représentant la présence d'une canicule.

Pour le risque d'inondation, nous avons placé nos départements, ainsi que 6 inondations en France depuis 2010, dans un repère à quatre dimensions avec les coordonnées suivantes: (précipitation journalière; température; débit moyen des rivières; d'humidité relative). En calculant la somme des distances entre un département et les 6 inondations, nous avons établi une mesure de "proximité" entre les conditions de chaque département et les conditions d'une inondation. Nous utilisons le processus de "z-score normalization" pour calculer l'intensité de la variance de la somme des distance d'un département par rapport aux sommes des distances de tous les autres. Nous obtenons ainsi une mesure pour chaque département a qu'elle point les données relevant d'une inondation sont atypiques, et donc une probabilité qu'il y ait une inondation.

Enfin, puisque l'API open-meteo.com nous offrait aussi la possibilité d'obtenir des données archivées ou des prévisions, nous voulions pouvoir permettre à l'utilisateur de remonter ou d'avancer dans le temps directement depuis la page web. Ainsi, nous avons modifié notre processus d'extraction de données et l'interface pour permettre à l'utilisateur de choisir la date des données cartographiées. Bien que le temps d'attente soit long, mais remédiable (voire V), la fonction n'en reste pas moins intéressante du point de vue données et expérience utilisateur.

(5) Raffinement de l'UI

Puisqu'un de nos objectifs de base était de rendre nos données accessibles au grand public, nous devions améliorer la User Interface (UI). C'est ainsi que nous avons utilisé nos compétences acquises en NSI d'HTML, de CSS et de JavaScript pour créer une page web qui intégrait notre carte comme iframe. Pour aider l'utilisateur à naviguer la carte, nous avons rajouté un popup "aide" qui lui permettrait de récupérer les informations nécessaires pour interagir avec la carte. En outre, nous devons aussi améliorer l'interface de la carte en elle-même. C'est ainsi que nous avons retiré les légendes des couches thématiques et opté pour une obtention des données via des pop ups accessibles lors d'un clic sur les icônes départementales. A cette fin, nous avons désactivé toutes les couches par défaut, pour que l'utilisateur décide quelle couche il veut voir. Enfin, nous avons offert la possibilité de changer entre la carte d'Open Street Map (OSM) et une carte simplifiée, de sorte à répondre aux préférences de l'utilisateur.

> FONCTIONNEMENT ET OPÉRATIONNALITÉ :

Au cours des différentes étapes présentées ci-dessus, nous avons rencontré différentes difficultés mineurs, liées notamment à la synchronisation des tables entre elles ou des difficultés liées à l'affichage avec le module Folium des cartes. Mais ces erreurs ont été surmontées, et leurs erreurs subséquentes ont été corrigées sans trop de difficultés.

Plutôt, le grand obstacle de notre projet était d'implémenter l'affichage des données météorologiques archivées. Si pendant l'intervalle de génération des nouvelles données l'utilisateur faisait une autre requête, le tableau présentait des défauts mineurs. Ce qui pouvait causer des problèmes lors du "rendering" de la carte.

Nous avons diagnostiqué que le problème venait en fait du fonctionnement simultané de nos subprocess et du parsing de la requête utilisateur, qui pouvait interférer avec le module pandas et son extraction des données. Pour contrer cette erreur, nous avons implémenté deux solutions pour éviter une nouvelle requête prématurée lors de la génération d'un fichier: nous avons rajouté un "cooldown" du bouton faisant des requête et nous avons ajouté des indications, comme un timer ou une animation de chargement sur la carte, pour avertir l'utilisateur. Enfin, même si l'erreur de génération venait à se produire, l'impact qu'elle aurait sur la cartographie ne serait que minime car la mauvaise génération

produit toujours le même nombre de données, un nombre suffisant pour cartographier exactement 94% des départements métropolitains.

Le projet est actuellement en fin de phase de débuggage. L'implémentation de tants de fonction a malheureusement causé de nombreux bugs mineurs : nous avons réparés ceux que nous avons trouv´

> OUVERTURE :

Nous voyons deux directions apparentes pour le développement futur de PiSky, mais elles sont antithétiques ; nous pourrions améliorer l'experience utilisateur, avec des temps de génération plus faibles et plus de styles de cartes, mais en contrepartie moins de données ; sinon, nous pourrions améliorer le coté data de PiSky, avec des données plus nombreuses et précises, des modèles plus complexes, et ce pour le monde entier, mais en contrepartie un temps de génération plus important.

En étant réalistes, nous développerons probablement PiSky en gardant un certain équilibre entre UI et data. A cette fin, nous avons plusieurs idées de ce que nous pourrions faire.

Tout d'abord, nous pouvons créer des tables de données pour toutes les dates que l'utilisateur peut sélectionner lors de la génération de la carte, pour améliorer la fluidité de l'expérience utilisateur lorsqu'il voyage dans le temps. En allant encore plus loin, nous pouvons générer ces tables sur un serveur personnel et les rendre accessibles, pour qu'elles puissent être importées plutôt que générées par l'utilisateur, ce qui diminuerait drastiquement le temps pour rendre la carte.

Ensuite, nous pouvons, avec ce nouveau temps gagné en hébergeant les données, ajouter plus de données pour plus de pays, si ce n'est pas tous.

Enfin, nous pourrons améliorer la précision de chacun des types de données. (1) Nos données qualitatives pourraient être prises d'APIs plus spécialisés, ce qui ajouterait une certaine complexité à l'extraction, mais aussi une certaine précision aux données elles-mêmes. Bien que nous ayons pris des APIs spécialisés pour la qualité de l'air et le débit moyen des rivières, nous avons décidé de ne pas encore utiliser ces APIs spécialisés pour toutes nos données, comme celles d'open-meteo.com, puisqu'elles proposaient moins de données et sur une plage temporelle plus étroite. (2) Nos données qualitatives pourraient utiliser plus de webcams en France, nous permettant donc d'établir une moyenne de condition météo à partir de plusieurs webcams dans un département plutôt que de déterminer à partir d'une seule. Il serait aussi possible, si la précision du processus de détermination était améliorée, d'ajouter plus d'états météo. (3) Finalement, nos données modélisées pourraient êtres améliorées en implémentant des modèles plus complexes. Malheureusement, la nature stochastique des catastrophes naturelles rendent cette possibilité compliquée sans entreprendre des études de météorologie mêlée à de l'informatique.

Bien que nous ayons de nombreuses pistes d'amélioration pour PySky, un de ses intérêts fondamentaux est que c'est en open source : la communauté peut aussi développer le projet dans la direction qu'ils souhaitent. Il est donc aussi question de diffuser PySky pour le développer. Pour ce faire, nous pouvons l'upload sur un site web ou établir une présence sur les réseaux sociaux. Globalement, PySky est un projet dont les utilisations sont multiples, et dont la diffusion peut se faire dans de nombreux secteurs, comme l'anticipation de catastrophes naturelles, la mise en sécurité des populations lors de ces dernières, ou même le développement durable et pérenne, pour n'en citer que quelques-uns.

DOCUMENTATION

Lien Github de notre projet : https://github.com/Oscar-T24/Trophees-NSI-2023

Prérequis:

Navigateur:

1. Firefox ou Chrome pour ***recuperation_donneeswebcam.py***
2. Tout navigateur avec les protocoles Interaction Homme Machine (IHM) à jour (Testé sur Firefox, Safari et Google Chrome)

Langage de programmation: Version de Python ultérieure à 3.10 (pour l'utilisation de "match…case" dans ***cartographie.py***)

Modules utilisés:

Module	**Commande d'installation pip**	**Description**
pandas	pip3 install pandas	Traitement de tableaux
folium	pip3 install folium	Permet de créer une carte
csv	Natif	Traitement de données csv (tableaux et fichiers temporaires)
random	Natif	Random
datetime	Natif	Extraction de l'heure
geopy.geocoders	pip3 install geopy	Données géographiques
re	Natif	extraire les entiers d'une chaine de caractere
cv2	pip3 install opencv-python	Traitement d'image
numpy	pip3 install numpy	Math
requests		Récupérer (fetch) les données des API et des caméras
json	Natif	transferts de données depuis les API
subprocess	Natif	executer un script python depuis un autre, avec des paramètres (argparse)
time	Natif	temps
argparse	Natif	executer un script python avec un argument
sys	Natif	permet d'exécuter des actions

		systemes relatif au système d'exploitation de l'utilisateur
indice (script)	script	
tronque (script)	script	
PIL	pip3 install PIL	utilisation du module Image pour manipuler les images
io	Natif	utilisation du module Bytes pour décoder les images reçues depuis infoclimat.fr
math	Natif	
tkinter	pip3 install tkinter	créer des fenêtres externes pour que l'utilisateur puisse interagir avec le code dans preparation_dataset_machine_learning.py
selenium	pip3 install selenium	automatiser les processus de recherche sur les navigateurs pour récupérer les données des webcams
bs4	pip3 install bs4	récupérer le contenu html d'une page web
shelljob		Shell et console
flask	pip3 install flask	émuler un serveur avec python pour créer une interface web avec l'utilisateur et afficher la carte
os	Natif	Permet d'effacer données météo.csv après l'utilisation

Scripts nécessaires:

Script	**Utilité**
cartographie.py	Créer le fichier html avec la carte, lui implémente les différentes couches choroplèthe
ecriture_coordonnees_departements.py	Extrait les coordonnées de chaque départements et leur identifiant ; places ces données extraites dans un fichier csv référencé dans les scripts en aval
filtrage_donnees_webcam.py	Réduit le nombre d'erreur recontré lors de **préparation_dataset_à_trier.py** en retirant certaines webcams superflus du fichier csv avec le lien des images en directe
indice.py	prend une image en paramètre et renvoi un indice multipliée par 100 , qui sera pris en compte dans la détermination qualitative de la météo
KNN_meteo.py	Détermine l'état météo pour chaque département selon la colonne indice du csv
main.py	Exécute les scripts principaux à l'appel de l'utilisateur pour générer une carte
main2.py	Actualise la base de donnée des webcams en france (un lien + un code département)

nromales.py	script permettant de calculer les valeurs normales par mois pour les 101 départements à partir des données des n dernières années, avec n compris entre 10 et 82
preparation_dataset_à_trier.py	Extrait les données de l'API, exécution des modèles, création du csv principal
preparation_dataset_machinelearning_supervié.py	Permet à KNN d'obtenir des catégories selon des inputs utilisateur
recuperation_donneeswebcam.py	Recupère les liens des webcams proposés sur infoclimat.fr en leur assignant un code département INSEE respectif
rendering.py	Initialise un serveur FLASK qui gère la partie web
run.py	Exécution de tous les fichiers
tronque.py	Formate l'image pour qu'elles soit correctement analysée dans indice.py. Notamment, il ne garde que le ciel pour rendre la détermination de l'indice plus fiable et précise

Utilisation:

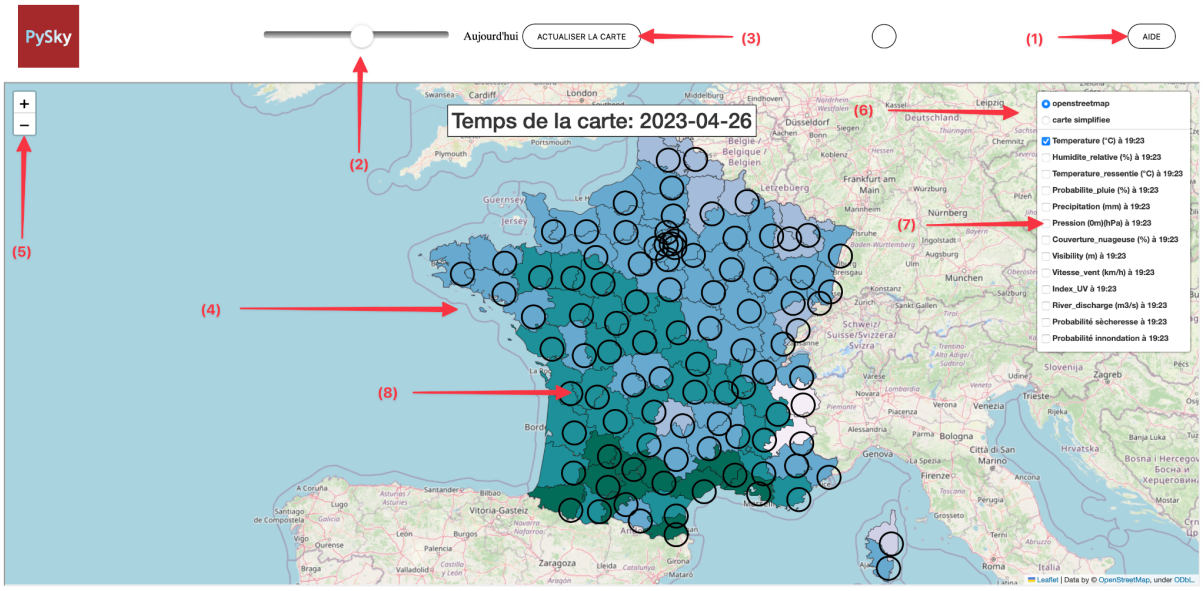
Initialisation:

Lancer le fichier run.py

Si vous rencontrez un problème, lancez d'abord preparation_dataset_a_trier.py et ensuite rendering.py.

Amusez-vous bien!

Interface utilisateur:



Trophees NSI - 2023. Site crée par Oscar, Paul, Hippolyte, Ava et Benjamin - 1ère à l'EJM Paris

(1) Afficher de l'aide relative aux symboles et aux actions utilisateur
(2) Sélectionner une date dans les 10 jours précédents aux 5 jours futurs
(3) Actualiser la carte avec la date paramétrée avec le slider
(4) Permet de visualiser les différentes données météorologiques
(5) Zoomer sur la carte
(6) Permet d'accéder aux couches et aux deux cartes proposées
(7) Afficher / cacher les différents choroplèthes (désactivées par défaut)
(8) Afficher les données météorologiques précises pour chaque département

Extraction des données:

(1) Données quantitatives:

Extraction des données suivantes avec leurs APIs respectives:

Intitulé de la données	**Lien d'extraction API**
Température	https://api.open-meteo.com/v1/forecast?latitude={x}&longitude={y}&hourly=temperature_2m,relativehumidity_2m,apparent_temperature,precipitation_probability,precipitation,pressure_msl,cloudcover,visibility,windspeed_10m,uv_index&forecast_days=1&start_date={dateiso}&end_date={dateiso}
Humidité relative	
Température ressentie	
Probabilité pluie	
Précipitation	
Pression	
Couverture nuageuse	
Visibilité	
Vitesse du vent	
Index UV	
Qualité de l'air	https://air-quality-api.open-meteo.com/v1/air-quality?latitude={x}&longitude={y}&hourly=pm2_5&start_date={dateiso}&end_date={dateiso}
Débit des rivières	https://flood-api.open-meteo.com/v1/flood?latitude={x}&longitude={y}&daily=river_discharge_mean&start_date={dateiso}&end_date={dateiso}&forecast_days=1

(2) Données modélisées

```
index_secheresse = [precipitation[i] / float(normale[i][mois]) * 100 for i in range(len(normale))]
```

Pour la sècheresse, nous avons utilisé la mesure proposée à la page 13 du rapport n°1173 de l'Organisation Météorologique Mondiale (OMM) intitulé "Manuel des indicateurs et indices de sécheresse" : multiplie le ratio de la précipitation actuelle et de la précipitation "normale" par 100, nous donnant un pourcentage, avec des pourcentages plus bas correspondants à un risque accrue de sécheresse.

```
seuils = [(1, 35), (2, 33), (3, 34), (4, 36), (5, 34), (6, 31), (7, 35), (8, 33), (9, 34), (10, 35), (11, 35), (12, 36),
          (13, 35), (14, 31), (15, 32), (16, 36), (17, 35), (18, 35), (19, 36), (21, 34), (22, 31), (23, 34),
          (24, 36), (25, 33), (26, 36), (27, 34), (28, 34), (29, 32), ('2A', 33), ('2B', 33), (30, 36), (31, 36),
          (32, 36), (33, 35), (34, 35), (35, 34), (36, 35), (37, 35), (38, 34), (39, 34), (40, 35), (41, 35), (42, 35),
          (43, 32), (44, 34), (45, 34), (46, 36), (47, 36), (48, 33), (49, 34), (50, 31), (51, 34), (52, 34), (53, 34),
          (54, 34), (55, 34), (56, 32), (57, 34), (58, 34), (59, 33), (60, 34), (61, 34), (62, 33), (63, 34), (64, 34),
          (65, 34), (66, 35), (67, 34), (68, 35), (69, 34), (70, 34), (71, 34), (72, 35), (73, 34), (74, 34), (75, 31),
          (76, 33), (77, 34), (78, 33), (79, 35), (80, 33), (81, 36), (82, 36), (83, 35), (84, 36), (85, 34), (86, 35),
          (87, 34), (88, 34), (89, 35), (90, 33), (91, 35), (92, 31), (93, 31), (94, 31), (95, 35)]
temperatures = [e for e in temperature]
canicule = [1 if temperatures[i] >= seuils[i][1] else 0 for i in range(len(seuils))] + ["", "", "", "", ""]
```

Pour la canicule, puisque cette dernière se définit à partir d'un seuil de température différente pour chaque département, nous avons pris les seuils offert par le rapport "Système d'alerte canicule et santé : principes, fondements et évaluation" de l'Institut de Veille Sanitaire et comparé ce seuil et la température maximale du jour par département, nous rendant une valeur booléen représentant la présence d'une canicule.

```
# Determination d'innondation inspirée par KNN se basant sur des valeurs de precipitation, de temperature, de débit des rivières et d'humidité relative dura

probabilite_flood = []

var2010 = (143.6, 17.7, 234, 75.1)
garonne2013 = (36.4, 27.1, 2171, 63.6)
languedoc2014 = (54.7, 16.3, 867, 85.6)
seine2016 = (10.2, 20.3, 4180, 75.7)
aude2018 = (94.4, 14.4, 769, 90.3)
occitanie2020 = (57.5, 17.8, 924, 80.2)


Pi-ckle
def distance_4d(t1, t2):
    return sqrt(sum([(t1[i] - t2[i]) ** 2 for i in range(4)]))

Pi-ckle
def distance_flood(t):
    if t[2] is None:
        return 0
    liste_distances = [distance_4d(t, e) for e in [var2010, garonne2013, languedoc2014, seine2016, aude2018, occitanie2020]]
    return round(sum(liste_distances), 3)

for i in range(len(precipitation)): # On a que les données d'humidité pour la france metropolitaine
    try:
        probabilite_flood.append(distance_flood((precipitation[i], temperature[i], river_discharge[i], humidite_relative[i])))
    except ValueError:
        probabilite_flood.append("")

Pi-ckle
def z_score_normalization(distance, ensemble_distances=probabilite_flood):
    """
    float, list --> float
    Processus de normalisation z-score: Determination de l'intensité de la variance de la distance pour KNN par rapport a la moyenne de ensemble_distances
    """
    return abs((distance - numpy.mean(ensemble_distances)) / numpy.std(ensemble_distances))

probabilite_floodv2 = [z_score_normalization(e) if e != 0 else None for e in probabilite_flood]
```

Pour le risque d'inondation, nous avons placé nos départements, ainsi que 6 inondations en France depuis 2010, dans un repère à quatre dimensions avec les coordonnées suivantes: (précipitation journalière; température; débit moyen des rivières; d'humidité relative). En calculant la somme des distances entre un département et les 6 inondations, nous avons établi une mesure de "proximité" entre les conditions de chaque département et les conditions d'une inondation. Nous utilisons le processus de "z-score normalization" pour calculer l'intensité de la variance de la somme des distance d'un département par rapport aux sommes des distances de tous les autres. Nous obtenons ainsi une mesure pour chaque département a qu'elle point les données relevant d'une inondation sont atypiques, et donc une probabilité qu'il y ait une inondation.

(3) Données qualitatives

1. Traitement des photos
    a. Extraction des données des webcams via infoclimat.fr
    b. Tronquage des données
    c. Calculer l'intensité moyenne en grayscale de l'image
    d. En déduire l'intensité nuageuse, et donc un indice
2. Détermination de l'état météo
    a. Prise de l'indexe et multiplication par 100 pour y accorder plus d'importance

b. Détermination utilisateur de la météo pour quelques images nécessaire via détermination dataset_machine_learning_supervisé.py pour initialiser l'algorithme KNN
c. Création d'un repère avec chaque descripteur comme axe
d. Calcul de la distance entre les départements et les évaluations connues
e. Classification KNN de chaque département et extraction de cette classe