



Création d'un tableau de bord interactif des prévisions météorologiques sous Python

Manuel de formation



GLOBAL
CENTER ON
ADAPTATION



Deltares

Création d'un tableau de bord interactif des prévisions météorologiques sous Python



Manuel de formation

Auteur(s)

Thomas Stolp
Dorien Lugt

PR4932.10

November 2023

Table des matières

1	Pour démarrer	2
1.1	GitHub Codespaces	2
1.2	Jupyter Notebooks	3
2	Introduction aux interfaces de programmation d'applications (API)	5
2.1	API REST	5
2.2	Structure	5
2.3	Réponse	6
2.4	Clients API	6
3	API de la station TAHMO	8
3.2	Récupération et traçage des données de précipitations journalières	9
4	API météorologique Open-Meteo	11
4.1	Open-Meteo	11
4.2	ECMWF IFS	11
5	Construire un tableau de bord	14
5.1	Introduction à Solara	14
5.2	Créer un tableau de bord de prévision	16

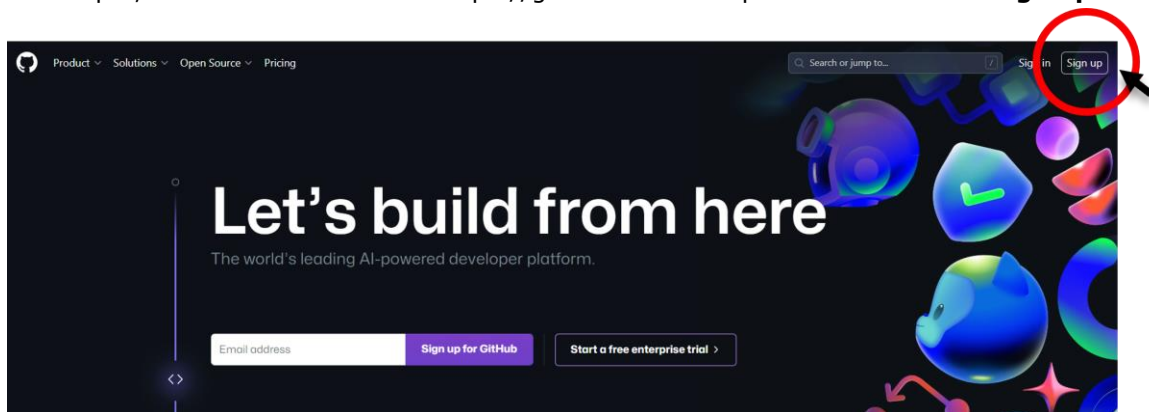
1 Pour démarrer

Dans ce chapitre, nous présentons l'environnement que nous utiliserons dans le cadre de cette formation, à savoir **GitHub Codespaces** et les **Jupyter Notebooks**. Si ces outils vous sont déjà familiers, vous pouvez passer directement au chapitre 2.

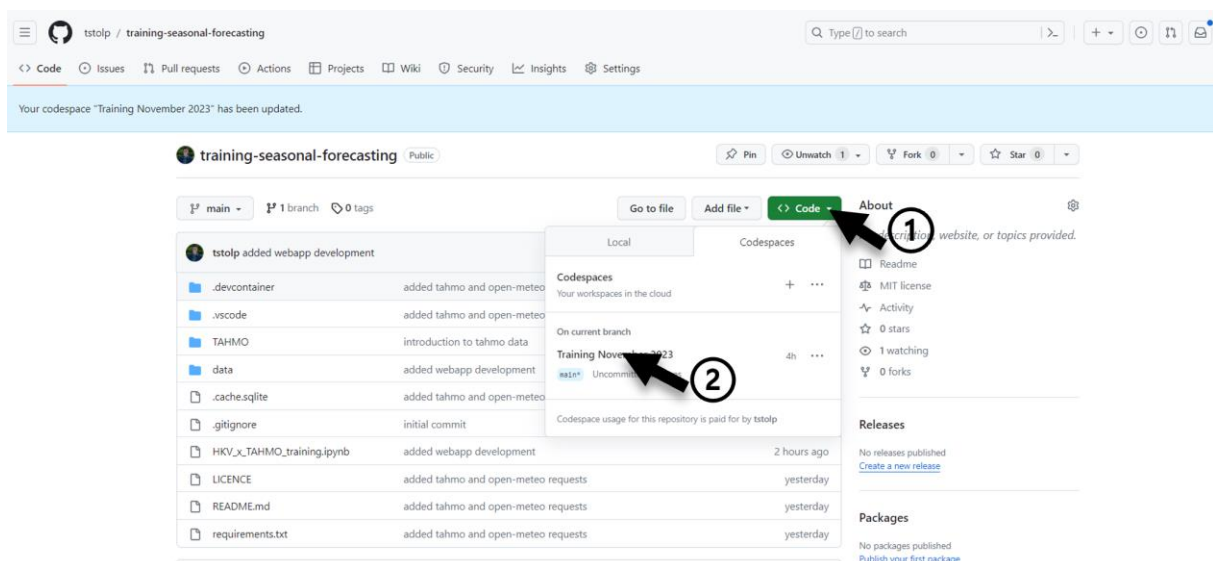
1.1 GitHub Codespaces

GitHub Codespaces est un environnement de développement en mode cloud fourni par GitHub, conçu pour rationaliser le processus de codage et de collaboration. Avec GitHub Codespaces, vous pouvez créer et gérer votre environnement de développement entièrement en ligne, éliminant ainsi le besoin d'une installation et d'une configuration locales complexes. Un répertoire GitHub a été créé pour cette formation, accessible dans <https://github.com/tstolp/training-seasonal-forecasting>.

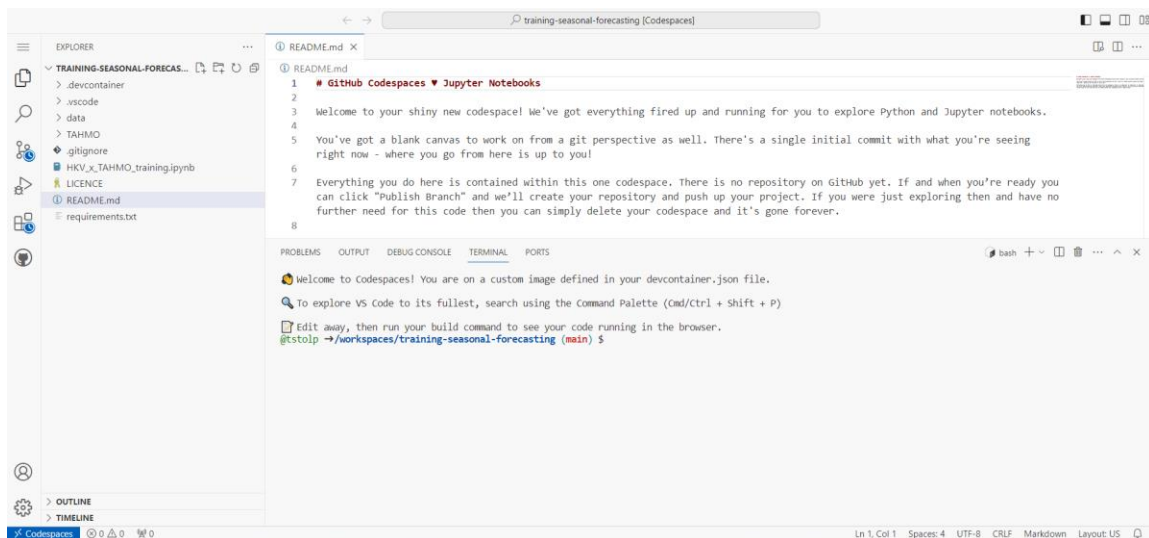
La création d'un compte est requise pour utiliser les GitHub Codespaces. Si vous n'avez pas encore de compte, veuillez suivre ce lien : <https://github.com> et cliquez sur le bouton « **Sign up** ».



Une fois connecté, allez dans le répertoire (<https://github.com/tstolp/training-seasonal-forecasting>) et cliquez sur le bouton vert « **Code** ».



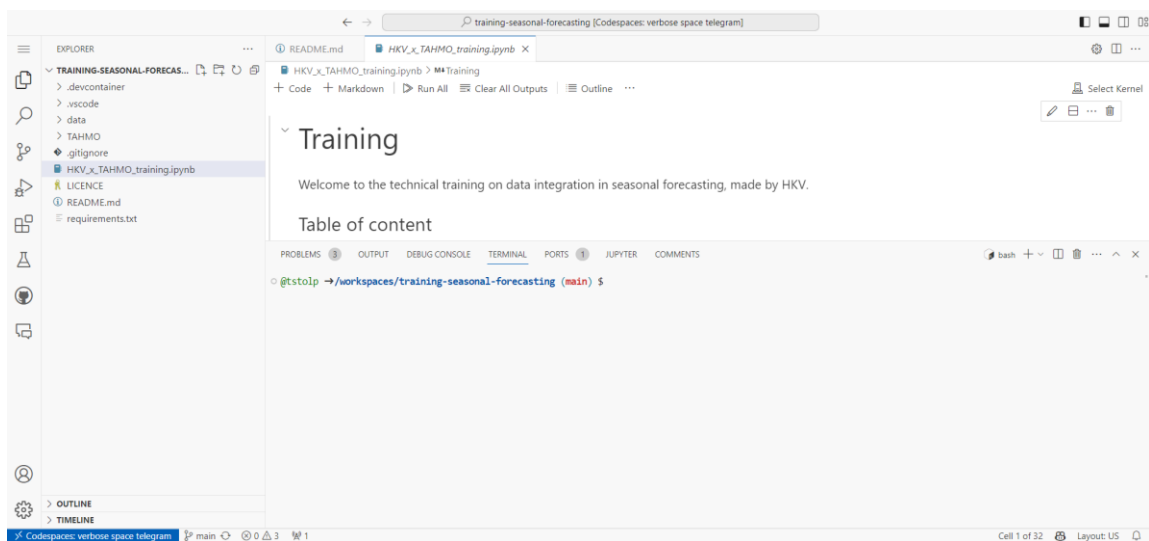
Ensuite, cliquez sur « **Create a new Codespaces on main** ». Une fois que vous avez ouvert le lien vers Codespaces, un environnement de développement est créé pour vous (cela peut prendre une minute).



Vous verrez la fenêtre suivante, cliquez sur le bloc-notes dans la partie gauche de l'écran « **training-notebook.ipynb** ». Cela ouvrira le bloc-notes utilisé pour la formation. Passez à la section suivante pour en savoir plus sur Jupyter Notebooks et sur la manière d'interagir avec le code.

1.2 Jupyter Notebooks

Une fois que vous avez ouvert le bloc-notes Jupyter Notebook, vous voyez des cellules qui sont les composants de base. Vous pouvez exécuter une cellule en cliquant sur le bouton **run** qui se trouve en haut côté gauche de la cellule. Vous pouvez également utiliser le raccourci clavier « **shift + entrer** » ou « **control + entrer** ». Le premier raccourci permet d'exécuter la cellule et de passer automatiquement à la suivante.



Vous pouvez fermer la fenêtre du bas, il s'agit du terminal que nous n'utiliserons pas dans cette formation.

2 Introduction aux interfaces de programmation d'applications (API)

Dans ce chapitre, nous présentons brièvement les API, en particulier les API RESTfull, et nous expliquons comment vous pouvez utiliser des clients API dans Python.

2.1 API REST

Une API (interface de programmation d'application) est un protocole qui définit comment des systèmes peuvent communiquer entre eux. Une API REST est construite sur la base des principes de conception REST (Representational State Transfer ou Transfert d'État Représentatif). REST est très flexible, c'est la raison pour laquelle on le trouve partout sur Internet. Il utilise les protocoles HTTP standard, à savoir :

1. **GET** : Cette méthode est utilisée pour demander des données à une ressource spécifiée.
2. **POST** : Cette méthode est utilisée pour soumettre des données à une ressource spécifiée.
3. **PUT** : Utilisé pour mettre à jour les données.
4. **DELETE** : Permet de supprimer des données.

Une API a besoin d'un **point de terminaison**, c'est-à-dire d'une URL spécifique à laquelle l'API envoie des requêtes et de laquelle elle reçoit des réponses. En termes plus simples, un point de terminaison d'une API est un chemin précis sur un serveur que l'API utilise pour exécuter une fonction particulière. Chaque point de terminaison représente une opération ou une ressource spécifique de l'API.

2.2 Structure

Les points de terminaison d'une API sont généralement structurés de façon hiérarchique, à travers lesquels les clients (applications ou systèmes) interagissent avec la fonctionnalité fournie par l'API. Les points de terminaison sont essentiels pour spécifier le type d'opération à effectuer, l'emplacement d'une ressource ou les données à récupérer ou à manipuler.

Voici un aperçu des composants d'un point de terminaison API :

- **URL de base** : Fournit l'adresse racine de l'API, l'endroit où l'API est hébergée. Par exemple, "https://api.example.com".
- **Chemin d'accès** : Vient après l'URL de base et représente la ressource spécifique. Par exemple, « /users ».
- **Méthode HTTP** : La méthode en soi pour récupérer les données, par exemple GET.

Vous pouvez également fournir des paramètres et des en-têtes pour les requêtes à l'API. Ces paramètres et en-têtes jouent un rôle crucial dans la personnalisation et la spécification des détails de votre requête. Voici une explication du fonctionnement des paramètres et des en-têtes dans les requêtes API. Ils sont généralement ajoutés sous forme de paires clé-valeur et sont annexés à l'URL du point de terminaison. Il existe deux types principaux de paramètres :

1. **Paramètres de requête** : '?key1=value1&key2=value2'
2. **Paramètres du chemin** : '/resource/{paramètre}'

De nombreuses API nécessitent une authentification. Pour les utiliser, vous avez souvent besoin d'une clé API, un identifiant unique qui vous donne accès. L'authentification est un exemple d'en-tête, elle fournit des informations supplémentaires sur la requête ou le client qui fait la requête.

2.3 Réponse

Il est essentiel de comprendre la documentation de l'API. Cette documentation fournit les détails sur les points de terminaison disponibles, les formats de requête, les paramètres et les structures de réponse. Les réponses sont généralement au format JSON ou XML. Lorsqu'une requête échoue ou réussit, un code d'état HTTP est souvent renvoyé.

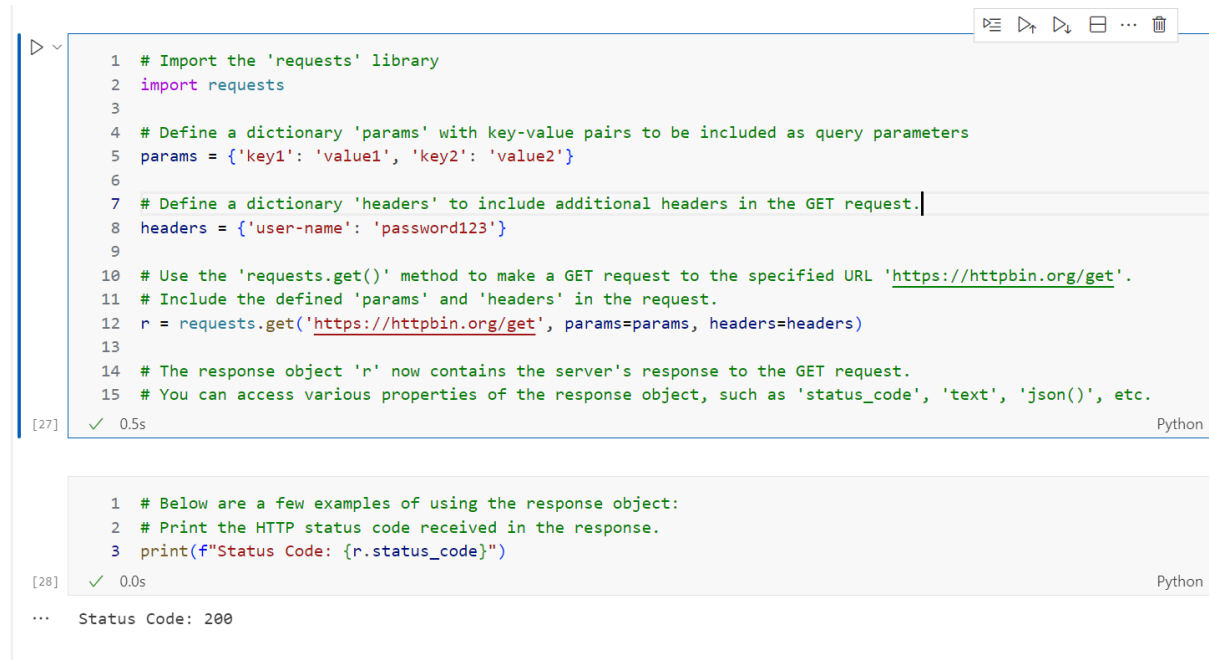
- **200 : OK**
La requête a été acceptée.
- **400 : Mauvaise requête**
La requête ne peut être satisfaite en raison d'une syntaxe incorrecte ou de paramètres non valides.
- **401 : Non autorisé**
L'authentification est requise et les informations d'identification fournies ne sont pas valides.
- **403 : Interdit**
Le serveur a compris la requête, mais il refuse de l'autoriser.
- **404 : Non trouvé**
La ressource demandée n'a pas pu être trouvée sur le serveur.
- **500 : Erreur interne du serveur**
La ressource demandée n'a pas pu être trouvée sur le serveur.

2.4 Clients API

Un client API est un logiciel ou un code qui utilise les fonctionnalités d'une API en envoyant des requêtes (éventuellement avec des paramètres et des en-têtes) et en traitant les réponses. Python utilise habituellement la bibliothèque des requêtes (<https://pypi.org/project/requests/>) pour envoyer des requêtes HTTP et traiter les réponses.

Le site web **httpbin.org** est un point de terminaison API qui vous permet de tester et d'inspecter les données que vous envoyez à un service. Nous pouvons l'utiliser pour tester le module de requête sous Python. Dans la cellule ci-dessous, nous créons une requête GET en utilisant la commande « `request.get()` » avec les paramètres de la requête et l'en-tête fournis comme

arguments. Dans la réponse, nous pouvons obtenir le code de statut 200, ce qui signifie que notre demande a abouti.



```
1 # Import the 'requests' library
2 import requests
3
4 # Define a dictionary 'params' with key-value pairs to be included as query parameters
5 params = {'key1': 'value1', 'key2': 'value2'}
6
7 # Define a dictionary 'headers' to include additional headers in the GET request.
8 headers = {'user-name': 'password123'}
9
10 # Use the 'requests.get()' method to make a GET request to the specified URL 'https://httpbin.org/get'.
11 # Include the defined 'params' and 'headers' in the request.
12 r = requests.get('https://httpbin.org/get', params=params, headers=headers)
13
14 # The response object 'r' now contains the server's response to the GET request.
15 # You can access various properties of the response object, such as 'status_code', 'text', 'json()', etc.
```

[27] ✓ 0.5s Python

```
1 # Below are a few examples of using the response object:
2 # Print the HTTP status code received in the response.
3 print(f"Status Code: {r.status_code}")
```

[28] ✓ 0.0s Python

... Status Code: 200

3 API de la station TAHMO

Dans ce chapitre, nous décrivons comment vous pouvez accéder au point de terminaison de l'API TAHMO et récupérer des données pour une variété de variables et de stations. Nous allons créer une visualisation interactive des mesures de précipitations d'une des stations.

3.1.1 API TAHMO

L'Observatoire hydrométéorologique transafricain (TAHMO) gère un réseau de stations météorologiques installés à travers le continent Africain. Les données de ces stations peuvent être récupérées à l'aide de l'API. Nous pouvons utiliser le client API-V2 qui se trouve sur la page GitHub de TAHMO (<https://github.com/TAHMO/API-V2-Python-examples>).

Dans cette formation, nous utilisons les identifiants démo qui nous donnent accès aux données de trois stations au total. Les informations d'identification se trouvent déjà dans le bloc-notes, tout ce que nous avons à faire est d'exécuter la cellule et nous pouvons démarrer.

```
1 # Import the TAHMO module
2 import TAHMO
3
4 # The demo credentials listed below give you access to three pre-defined stations.
5 api = TAHMO.apiWrapper()
6
7 # set the credentials
8 api.setCredentials('demo', 'DemoPassword1!')
```

[12] ✓ 0.0s Python

TAHMO.apiWrapper() est un client qui gère les demandes et les réponses de l'API pour nous. Dans la cellule suivante, nous envoyons une requête pour récupérer les stations auxquelles nous avons accès. Nous pouvons voir dans la réponse que ces stations sont identifiées par des codes.

```
1 # list other stations that are available
2 stations = api.getStations()
3 print('Account has access to stations: %s' % ', '.join(list(stations)))
```

[13] ✓ 0.4s Python

... API request: services/assets/v2/stations
Account has access to stations: TA00134, TA00252, TA00567

3.1.2 Variables

Ensuite, nous pouvons obtenir les variables enregistrées et disponibles pour ces stations. Pour ce faire, nous utilisons la méthode « **getVariables()** ». Exécutez la cellule dans le bloc-notes et vous obtiendrez une liste de variables avec les abréviations et les unités correspondantes. Dans cette formation, nous nous concentrerons sur les précipitations, mais la requête de données pour d'autres variables se fait de la même manière.

Variable	Numéro court	Unité
Pression atmosphérique	ap	kPa
Profondeur de l'eau	dw	mm
Conductivité électrique des précipitations	ec	mS/cm
Conductivité électrique de l'eau	ew	mS/cm
Distance de la foudre	ld	km
Événements de foudre	le	-
Rayonnement à ondes courtes	ra	W/m2
Humidité du sol	sm	m3/m3
Température du sol	st	degrés Celsius
Température de l'air en surface	te	degrés Celsius
Pression de vapeur	vp	kPa
Rafales de vent	wg	m/s
Vitesse du vent	ws	m/s
Température du capteur d'humidité	ht	degrés Celsius
Niveau de l'axe X	tx	degrés
Niveau de l'axe Y	ty	degrés
Pourcentage de la batterie de l'enregistreur	lb	-
Pression de référence de l'enregistreur	lp	kPa
Température de l'enregistreur	lt	degrés Celsius
Précipitations cumulées	cp	mm
Niveau d'eau	wl	m
Vitesse de l'eau	wv	m/s
Précipitations	pr	mm
Humidité relative	rh	-
Direction du vent	wd	degrés
Conductivité électrique du sol	se	mS/cm
Température de l'eau	tw	degrés Celsius
Débit d'eau	wq	m3/s

3.2 Récupération et traçage des données de précipitations journalières

Dans ce paragraphe, nous allons récupérer les données réelles d'une station météorologique à l'aide d'une requête GET. Nous avons choisi d'utiliser la station « TA00567 ». Avec le code suivant, nous pouvons obtenir les coordonnées et le nom de cette station. Il s'agit de la station météorologique de l'Accra Girls Senior High School, une école de jeunes filles au Ghana.

```

1 # choose a station
2 station = 'TA00567'
3
4 # get the data
5 station_data = api.getStations()[station]
6
7 print()
8 print( f"Station name = {station_data['location']['name']}" )
9 print( f"Longitude = {station_data['location']['longitude']:.02f}" )
10 print( f"Latitude = {station_data['location']['latitude']:.02f}" )

```

[45] ✓ 0.9s Python

API request: services/assets/v2/stations

Station name = Accra Girls SHS
Longitude = -0.19
Latitude = 5.60

Dans la cellule ci-dessous, nous créons la requête Get en utilisant le code court pour les précipitations et en précisant un intervalle de temps pour obtenir uniquement les données de l'année en cours. La réponse est un **pandas DataFrame**, qui est une structure de données composée de lignes et de colonnes. Dans le cas présent, il s'agit de deux colonnes : les précipitations mesurées et l'heure de la mesure.

```

1 # Example 3: Retrieve a pandas dataframe containing the time serie of surface air observations and save to C
2
3 startDate = '2023-01-01'
4 endDate = '2023-11-22'
5 variables = ['pr']
6
7 df = api.getMeasurements(station, startDate=startDate, endDate=endDate, variables=variables)
8 df.index.name = 'Timestamp'
9 df.to_csv('data/timeseries.csv', na_rep='', date_format='%Y-%m-%d %H:%M')
10 print('Timeseries saved to file "timeseries.csv"')
11

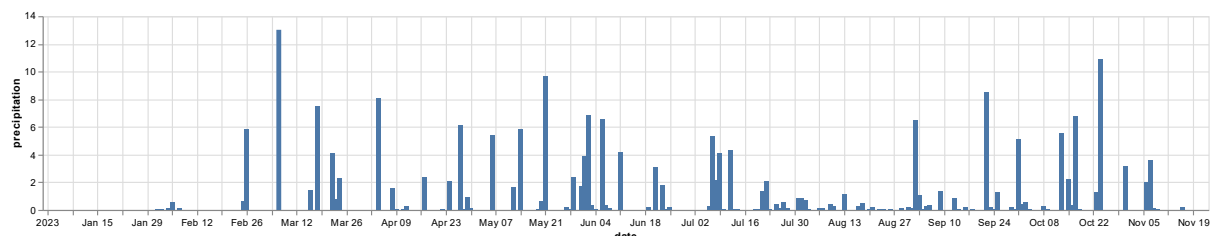
```

[46] ✓ 5.6s Python

API request: services/measurements/v2/stations/TA00567/measurements/controlled

Timeseries saved to file "timeseries.csv"

Ensuite, nous pouvons afficher les données dans un diagramme à barres. Dans ce bloc-notes, nous utilisons **Vega-Altaïr** (<https://altair-viz.github.io/>) pour créer une visualisation interactive dans laquelle nous pouvons zoomer et survoler les barres pour voir les données exactes et les précipitations mesurées.



4 API météorologique Open-Meteo

Dans ce chapitre, nous montrons comment l'API Open-Meteo peut être utilisée pour obtenir des données de prévision pour un lieu donné (coordonnées de latitude et de longitude). Nous visualiserons ces données dans un graphique interactif.

4.1 Open-Meteo

Open-Meteo (<https://open-meteo.com>) est une API météorologique open source et gratuite pour une utilisation non commerciale (pas besoin d'identifiants !). Elle utilise les prévisions météorologiques fournies par des services météorologiques tels que le Centre européen pour les prévisions météorologiques à moyen terme (CEPMMT). Normalement, ces prévisions météorologiques numériques sont difficiles à utiliser en raison des connaissances requises pour manipuler les projections et les formats de données.

4.1.1 API disponibles

Il existe plusieurs API sur Open-Meteo, ci-dessous un aperçu :

1. **Forecast API** - prévisions météorologiques jusqu'à 14 jours.
2. **Historical weather API** - données météorologiques passées (horaires) remontant à 1940.
3. **Ensemble models API** - modèle d'ensemble combinant plus de 200 prévisions individuelles.
4. **Climate Change API** - prévisions climatiques du GIEC (Groupe d'experts intergouvernemental sur l'évolution du climat) à une résolution de 10 km.

4.2 ECMWF IFS

Avec l'API de prévision Open-Meteo, il est possible de requérir des prévisions météorologiques à accès ouvert du CEPMMT, générées par le modèle météorologique IFS. L'accès aux données ouvertes est limité à une résolution de 40 km et à des valeurs trihoraires. Malgré cette limitation, le modèle reste d'une précision impressionnante dans la prévision des phénomènes météorologiques à grande échelle.

4.2.1 API de prévision

Nous demanderons les prévisions du modèle CEPMMT en utilisant la méthode GET à partir de l'url de base : <https://api.open-meteo.com/v1/forecast>. Nous pouvons donner quelques paramètres tels que le nom de la variable (nous utilisons « precipitation ») et le nombre de jours de la prévision (par exemple 10).

Une autre option dans l'API de prévision est le paramètre « past_days ». En définissant ce paramètre, on peut renvoyer les données météorologiques passées. Dans cet exemple, nous l'avons fixé à 90 jours.

Dans le bloc de code ci-dessous, nous avons créé une fonction qui renvoie les données obtenues en réponse à notre demande GET. Les données sont sous le format JSON et la fréquence des mesures devra être convertie en sommes journalières. Nous utilisons la latitude et la longitude de la station météo TAHMO à l'école l'Accra Girls Senior High School.

```

1 import requests
2
3
4 def get_ecmwf_precipitation(lon, lat):
5     """Retrieve the ECMWF precipitation forecast from the Open-Meteo API and return a JSON object"""
6
7     base_url = "https://api.open-meteo.com/v1/forecast"
8
9     # Specify the parameters for the ECMWF precipitation forecast
10    params = {
11        "longitude": lon,
12        "latitude": lat,
13        "daily": "precipitation_sum",
14        "past_days": 90,
15        "timezone": "auto",
16        "hourly": "precipitation",
17        "start": "current",
18        "forecast_days": 10,
19        "models": "ecmwf_ifs04"}
20
21    try:
22        # Make a request to the Open-Meteo API
23        response = requests.get(base_url, params=params)
24        data = response.json()
25        return data
26    except requests.RequestException as e:
27        print(f"Error: {e}")
28
29    data = get_ecmwf_precipitation(lon=station_data['location']['longitude'], lat=station_data['location']['latitude'])
30
31

```

Pour traiter les données, nous définissons la fonction suivante, qui convertit les types de données et renomme les colonnes.

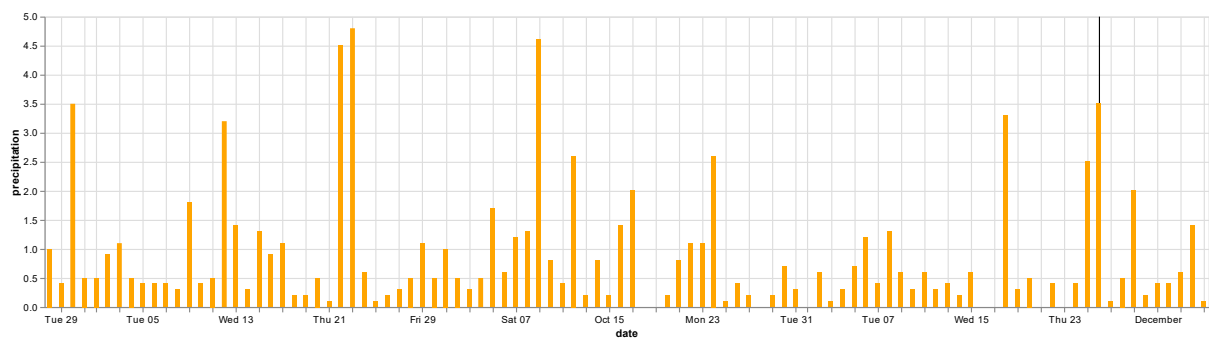
```

1 def process_ecmwf_precip_data(data):
2     """Load the precipitation data from the Open-Meteo API and return a pandas dataframe"""
3     df = pd.DataFrame.from_dict(data['hourly'])
4     df['time'] = pd.to_datetime(df['time'])
5     df.loc[:, 'date'] = df['time'].dt.date
6     df['date'] = pd.to_datetime(df['date'])
7     df = df[['date', 'precipitation']].dropna()
8     df = df.groupby('date').max().reset_index().set_index('date')
9     return df
10
11 df_ecmwf = process_ecmwf_precip_data(data)
12 df_ecmwf.head()

```

	date	precipitation
0	2023-08-28	1.0
1	2023-08-29	0.4
2	2023-08-30	3.5
3	2023-08-31	0.5
4	2023-09-01	0.5

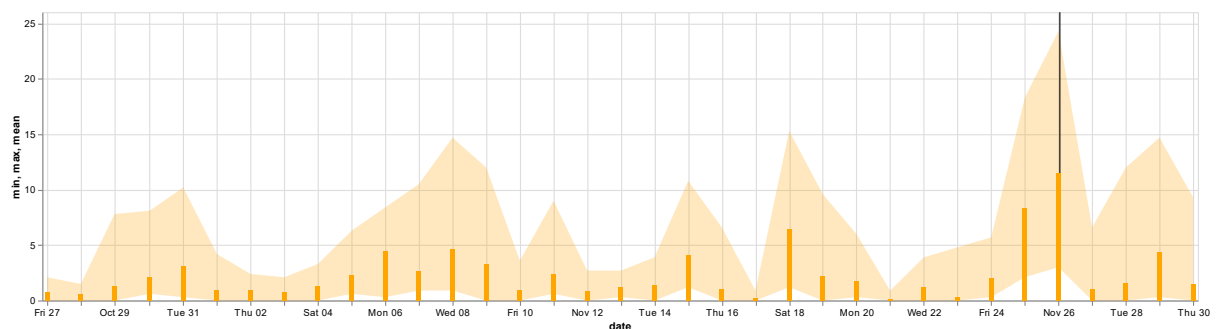
Nous pouvons tracer ces données, en plaçant une règle verticale noire à la date du jour, de manière à ce que nous puissions voir où la prévision commence.



4.2.2 API d'ensemble

Dans le cas d'une prévision d'ensemble, plusieurs versions d'un modèle météorologique numérique sont utilisées pour générer un ensemble varié de résultats potentiels. Chaque membre est initié avec des conditions initiales ou des paramètres de modèle légèrement différents, en tenant compte des incertitudes et des variations atmosphériques et en produisant une collection de prévisions perturbées.

Avec l'API d'ensemble Open-Meteo, nous pouvons récupérer les ensembles CEPMMT IFS. Dans le bloc-notes, nous montrons comment demander des données à l'aide de l'API d'ensemble et tracer les valeurs minimales et maximales de manière à obtenir une gamme d'événements plausibles.



5 Construire un tableau de bord

Dans ce chapitre, nous présentons Solara, un module permettant de créer des applications web fonctionnant sous Jupyter Notebooks. Nous allons créer un tableau de bord pour les prévisions d'ensemble des précipitations dans certaines stations météorologiques TAHMO.

5.1 Introduction à Solara

Solara (<https://solara.dev>) est une bibliothèque Open Source permettant de créer des applications web dans un Jupyter Notebook, mais elles fonctionneront également avec des frameworks web professionnels. Elle s'appuie sur IPywidgets qui permet d'utiliser des contrôles interactifs tels que des curseurs et des cases à cocher.

5.1.1 Composants et gestion des événements

Solara utilise des « composants » qui sont des blocs de construction de l'application et qui sont réutilisables. Il en existe deux types : les composants **widgets** et les **composants fonctionnels**. Les composants Widget sont des IPywidgets qui permettent de créer une application interactive. Les composants fonctionnels combinent l'état logique et d'autres composants pour créer des applications complexes et dynamiques.

A l'intérieur de la fonction, vous pouvez créer la structure du composant en utilisant les composants intégrés de Solara ou en créant des composants personnalisés pour répondre à vos besoins spécifiques. Dans l'exemple ci-dessous, nous créons un menu de sélection dans lequel vous pouvez sélectionner les codes des stations TAHMO.



La philosophie de conception de Solara consiste à créer d'abord tous les composants dont vous avez besoin dans votre application ou votre tableau de bord, puis à mettre en page ces composants dans une « page ». Pour plus d'informations sur la mise en page, consultez le site <https://solara.dev/docs/howto/layout>.

Les composants sont réutilisables. Dans l'exemple suivant, nous créons une page qui contient deux éléments du composant StationSelect.



```
1 import solara
2
3 @solara.component
4 def StationSelect():
5     solara.Select(label='station', values=['TA00134', 'TA00252', 'TA00567'], value='TA00134')
6
7 @solara.component
8 def Page():
9     StationSelect()
10    StationSelect()
11
12 Page()
```

[7] ✓ 0.0s Python

station
TA00134

station
TA00134

Les composants peuvent capturer des données utilisateur et répondre à des événements. Pour gérer les entrées des utilisateurs, vous définissez des fonctions de rappel et les connectez au composant à l'aide du système de gestion des événements de Solara.



```
1 import solara
2
3 def on_value_change(change):
4     print(f"Value changed!")
5
6 @solara.component
7 def StationSelect():
8     solara.Select(label='station', values=['TA00134', 'TA00252', 'TA00567'], value='TA00134', on_value=on_value_change)
9
10 StationSelect()
```

[3] ✓ Python

station
TA00567

Value changed!

Dans l'exemple ci-dessus, nous avons créé un composant qui utilise une fonction de rappel (on_value_change) et la relie au traitement des événements (on_value).

5.1.2 Arguments et état

Les composants que nous avons créés peuvent accepter des arguments, c'est-à-dire les valeurs transmises lorsque nous appelons les fonctions. De cette manière, il est possible d'avoir des éléments du même composant mais avec un comportement ou une apparence différents.

Il est également possible de gérer l'état d'un composant dans Solara en créant une variable réactive avec solara.reactive(). Les variables réactives stockent des valeurs qui peuvent changer

au fil du temps et, en fonction de cette valeur, peuvent déclencher d'autres composants dans votre application.



```
1 import solara
2
3 station = solara.reactive('TA00134')
4
5 def set_station(value):
6     station.value = value
7
8 @solara.component
9 def StationDisplay():
10     solara.Info(f"Station: {station.value}")
11
12 @solara.component
13 def StationSelect():
14     solara.Select(label='station', values=['TA00134', 'TA00252', 'TA00567'], value=station.value, on_value=set_station)
15
16 @solara.component
17 def Page():
18     StationSelect()
19     StationDisplay()
20
21
22 Page()
```

[10] ✓ 0.0s Python

station
TA00567

Station: TA00567

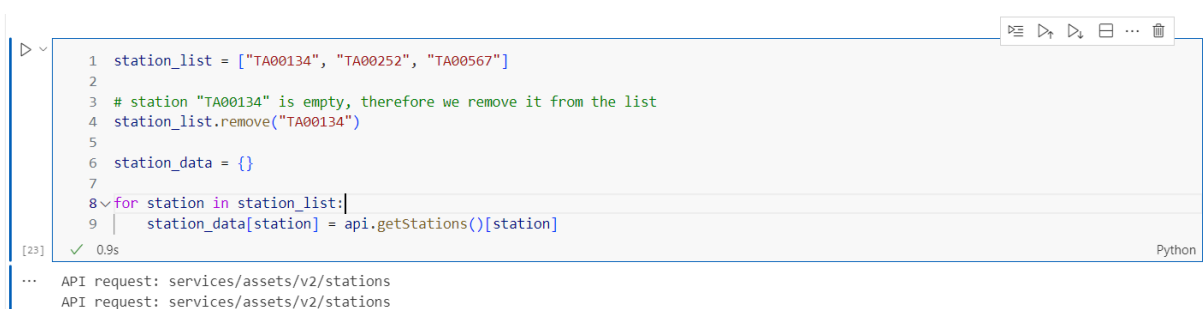
Dans l'exemple ci-dessus, nous créons une variable d'état (ou variable réactive) « station » dont la valeur par défaut est « TA00134 », mais qui peut prendre d'autres valeurs. En sélectionnant une autre valeur dans le menu de sélection, nous constatons que la variable station est modifiée. Nous montrons cela en affichant la valeur de station avec un composant solara.info.

5.2 Créer un tableau de bord de prévision

Dans cette section, nous travaillerons sur un tableau de bord réel pour voir les prévisions d'ensemble pour les stations TAHMO.

5.2.1 Créer une carte avec IPyleaflet

La première étape consiste à créer une carte afin de visualiser l'emplacement de la station météorologique. Pour ce faire, nous utiliserons le module IPyleaflet. Commençons par créer un dictionnaire avec les données de la station :



```
1 station_list = ["TA00134", "TA00252", "TA00567"]
2
3 # station "TA00134" is empty, therefore we remove it from the list
4 station_list.remove("TA00134")
5
6 station_data = {}
7
8 for station in station_list:
9     station_data[station] = api.getStations()[station]
```

[23] ✓ 0.9s Python

API request: services/assets/v2/stations
API request: services/assets/v2/stations

Il est à noter que la station « TA00134 » n'a pas enregistré de données de précipitations au cours des derniers mois, nous excluons donc cette station de notre liste.

Ci-dessous, nous créons nos variables d'état et le menu de sélection que nous avons créé plus tôt.

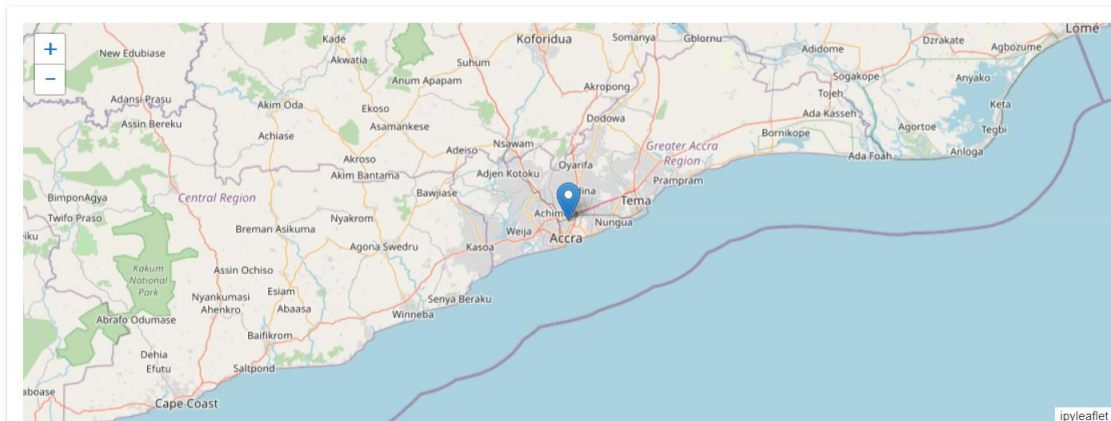
```
1 import solara
2 import ipyleaflet
3 import TAHO
4 from ipywidgets import HTML
5
6 # Create a TAHO API wrapper and set credentials
7 api = TAHO.apiWrapper()
8 api.setCredentials('demo', 'DemoPassword1!')
9
10 station_default = 'TA00252'
11 center_default = (station_data[station_default]['location']['latitude'], station_data[station_default]['location']['longitude'])
12 zoom_default = 9
13
14 # Define reactive variables for station data
15 station = solara.reactive(station_default)
16 zoom = solara.reactive(zoom_default)
17 center = solara.reactive(center_default)
18
19 def set_station(value):
20     station.value = value
21     center.value = (station_data[value]['location']['latitude'], station_data[value]['location']['longitude'])
22
23 @solara.component
24 def StationSelect():
25     """Solara component for a station selection dropdown."""
26     solara.Select(label="station", values=station_list, value=station.value, on_value=set_station, style={"z-index": "10000"})
27
```

Nous ajoutons un composant « visualisation » qui affiche la station sur les cartes et zoome automatiquement sur une autre station lorsque celle-ci est sélectionnée dans le menu déroulant.

```
28 @solara.component
29 def View():
30     """Solara component for displaying a map view with a marker for the selected station."""
31     ipyleaflet.Map(element=center.value,
32                    zoom=9,
33                    on_center=center.set,
34                    scroll_wheel_zoom=True,
35                    layers=[ipyleaflet.TileLayer(element=url=ipyleaflet.basemaps.OpenStreetMap.Mapnik.build_url()) + [ipyleaflet.Marker.element
36                    ]
37
38 @solara.component
39 def Page():
40     """Solara component for a page with two cards: View and StationSelect."""
41     with solara.Column(style={"min-width": "500px", "height": "500px"}):
42         with solara.Row():
43             StationSelect()
44             with solara.Card():
45                 View()
46
47 Page()
```

La sortie de ce bloc de code est illustrée ci-dessous. Sélectionnez une autre station et le centre de la carte deviendra le nouvel emplacement.

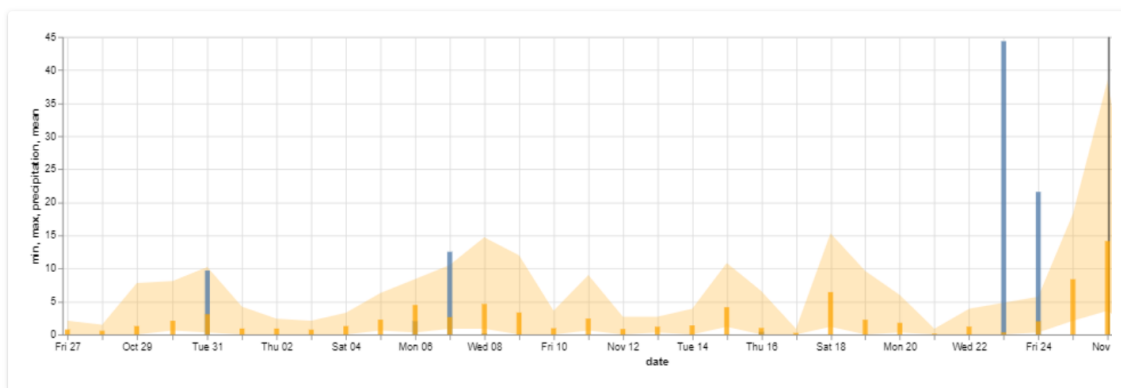
station
TA00567



5.2.2 Créer un graphique de séries temporelles avec les mesures et les prévisions de précipitations

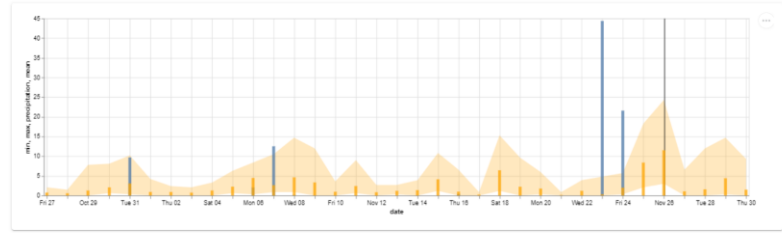
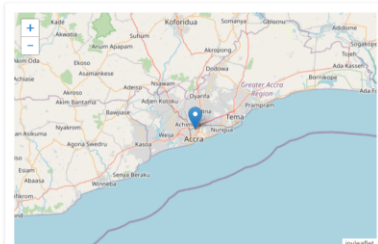
La deuxième figure présente les mesures de précipitations à la station TAHMO (en bleu) et la moyenne des prévisions d'ensemble (en orange). La bande orange indique les précipitations journalières minimales et maximales des ensembles. Exécutez le code dans le Notebook et voyez si vous obtenez le même résultat.

station
TA00567



5.2.3 Création d'un tableau de bord final

Dans le bloc-notes, nous combinons la carte et le tracé des séries temporelles. Lorsque vous sélectionnez une autre station, les deux se mettent à jour, une requête est faite pour obtenir les prévisions d'ensemble pour la nouvelle station.



Vous pouvez créer un tableau de bord à l'intérieur du bloc-notes, comme c'est le cas durant cette formation, mais il est également possible d'exécuter un script python à l'aide du serveur Solara. Pour ce faire, il suffit d'ouvrir un nouveau terminal et d'utiliser la commande suivante. L'utilisation de l'argument `-host` était nécessaire sur mon ordinateur local mais cela peut fonctionner sans cela dans l'environnement Codespaces. Essayez-le.

```
PROBLEMS 5 OUTPUT PORTS 24 TERMINAL JUPYTER
bash - training-seasonal-forecasting + v [ ] [ ] ... ^ x
vscode → /workspaces/training-seasonal-forecasting (main) $ solara run sol.py --host=0.0.0.0
```

Vous obtiendrez une URL que vous pourrez ouvrir dans votre propre navigateur !



HKV

Lelystad

Botter 11-29
8232 JN Lelystad
Pays-Bas

Delft

Informaticalaan 8
2628 ZD Delft
Pays-Bas

Amersfoort

Berkenweg 7
3818 LA Amersfoort
Pays-Bas

0320 294242
info@hkv.nl
www.hkv.nl