TP Charette

Etudiant : SOSSOU Didi Orlog

Détection de capteurs défaillants ou en maintenance

Cellule d'installation des packages (si indisponibles dans l' environnement d' execution)

In [ ]:
```
! pip install pandas matplotlib seaborn scipy
```

Importation des bibliotheques necessaires au reste du code

Chargement de *matplotlib* pour les graphiques, *seaborn* pour les visualisations statistiques, *numpy* pour les calculs numériques et *pandas* pour la gestion des données tabulaires.

In [68]:
```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from scipy import stats
from scipy.stats import binom
import warnings

# Desactivation des avertissements pour une meilleure lisibilité
# (peut être commenté si vous souhaitez voir les avertissements)
warnings.filterwarnings("ignore")
```

1 - Chargement de données

Dans cette partie nous allons charger les données en nous servant du fichier "capteur_sol_enrichis.csv".

Nous affichons le Dataframe obtenu pour nous assurer de la bonne importaion.

In [69]:
```
# Lire le jeu de données
df = pd.read_csv("capteur_sol_enrichis.csv", sep=',')

# Dimension du DataFrame
print("Dimension du DataFrame : ", df.shape) # Dimension du DataFrame

# Afficher les 10 premieres lignes
df.head(10)
```

```
Dimension du DataFrame :  (61200, 6)
```

Out[69]:

	id_capteur	zone	date_heure	humidite_sol	temperature_sol	etat_capteur
0	SC_A_1	zoneA	2024-04-01 00:00:00	23.51	12.85	DEFAILLANT
1	SC_A_1	zoneA	2024-04-01 01:00:00	22.58	12.88	DEFAILLANT
2	SC_A_1	zoneA	2024-04-01 02:00:00	14.19	11.61	DEFAILLANT
3	SC_A_1	zoneA	2024-04-01 03:00:00	25.16	12.01	DEFAILLANT
4	SC_A_1	zoneA	2024-04-01 04:00:00	24.90	11.05	DEFAILLANT
5	SC_A_1	zoneA	2024-04-01 05:00:00	11.22	13.26	DEFAILLANT
6	SC_A_1	zoneA	2024-04-01 06:00:00	23.50	12.70	DEFAILLANT
7	SC_A_1	zoneA	2024-04-01 07:00:00	30.61	12.59	DEFAILLANT
8	SC_A_1	zoneA	2024-04-01 08:00:00	31.74	11.95	DEFAILLANT
9	SC_A_1	zoneA	2024-04-01 09:00:00	25.35	12.74	DEFAILLANT

Nos données sont bien chargées et on a 61200 lignes et 6 colonnes.

2 - Filtrage dont l' etat des capteurs est OK

Nous allons appliquer un filtre sur nos données importées pour éliminer les lignes contenant des capeturs en bon état.

In [70]:
```
# Filter les capteurs dont etat_capteur est "OK" pour ne garder que les capteurrs defectueux
df_defectueux = df[df['etat_capteur'] != 'OK']

# Dimension du DataFrame des capteurs defectueux
print("Dimension du DataFrame des capteurs defectueux : ", df_defectueux.shape)

# Afficher les 10 premieres lignes des capteurs defectueux
df_defectueux.head(10)
```

```
Dimension du DataFrame des capteurs defectueux :  (12960, 6)
```

Out[70]:

	id_capteur	zone	date_heure	humidite_sol	temperature_sol	etat_capteur
0	SC_A_1	zoneA	2024-04-01 00:00:00	23.51	12.85	DEFAILLANT
1	SC_A_1	zoneA	2024-04-01 01:00:00	22.58	12.88	DEFAILLANT
2	SC_A_1	zoneA	2024-04-01 02:00:00	14.19	11.61	DEFAILLANT
3	SC_A_1	zoneA	2024-04-01 03:00:00	25.16	12.01	DEFAILLANT
4	SC_A_1	zoneA	2024-04-01 04:00:00	24.90	11.05	DEFAILLANT
5	SC_A_1	zoneA	2024-04-01 05:00:00	11.22	13.26	DEFAILLANT
6	SC_A_1	zoneA	2024-04-01 06:00:00	23.50	12.70	DEFAILLANT
7	SC_A_1	zoneA	2024-04-01 07:00:00	30.61	12.59	DEFAILLANT
8	SC_A_1	zoneA	2024-04-01 08:00:00	31.74	11.95	DEFAILLANT
9	SC_A_1	zoneA	2024-04-01 09:00:00	25.35	12.74	DEFAILLANT

Apres avoir filtré les lignes de capteurs en bon etat pour ne garder que ceux defectueux, nous constatons que le dataframe est reduit à 12960 lignes pour 6 colonees.

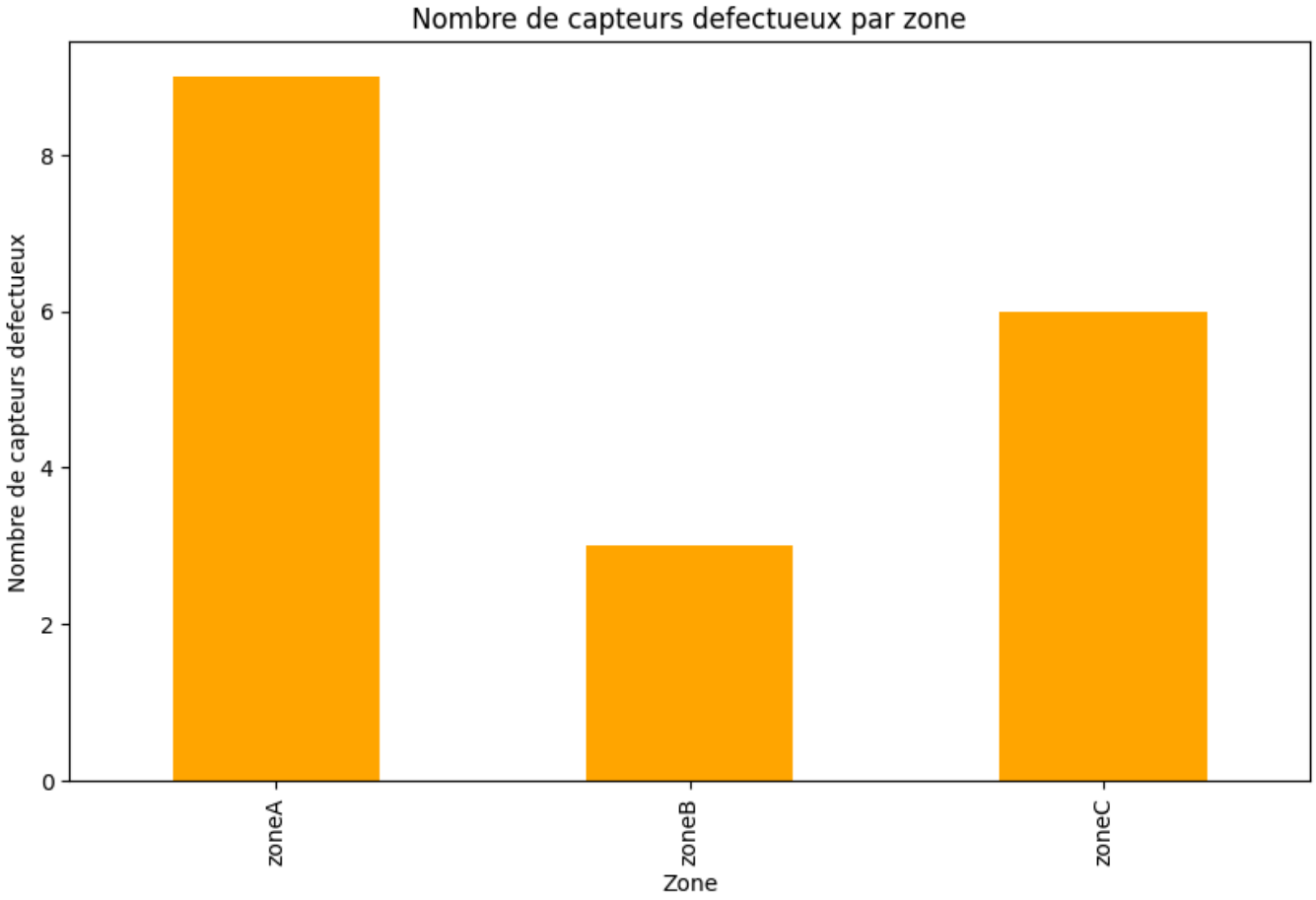
3 - Comptage du nombre de capteurs par zone affectés

Nous allons à present compter le nombre de capteurs defectueux par zone.

In [71]:
```
# Nous allons à present compter le nombre de capteurs defectueux par zone, en tenant compte
compteur_defectueux_par_zone = df_defectueux.groupby('zone')['id_capteur'].nunique()

# Affichage
print("Nombre de capteurs defectueux par zone :")
print(compteur_defectueux_par_zone)

# Visualisation graphique
compteur_defectueux_par_zone.plot(kind='bar', figsize=(10, 6), color='orange')
plt.title('Nombre de capteurs defectueux par zone')
plt.xlabel('Zone')
plt.ylabel('Nombre de capteurs defectueux')
```

```
Nombre de capteurs defectueux par zone :
zone
zoneA    9
zoneB    3
zoneC    6
Name: id_capteur, dtype: int64
```

Out[71]:
```
Text(0, 0.5, 'Nombre de capteurs defectueux')
```



La zone A est celle qui a le plus de capteurs défectueux : 9 , contre 6 pour la zone C et 3 pour la zone B

4 - Identification des créneaux horaires d'indisponibilité.

In [72]:
```
# Conversion de date_heure en datetime et extraction de l'heure et de la date
df_defectueux['date_heure'] = pd.to_datetime(df_defectueux['date_heure'])
df_defectueux['heure'] = df_defectueux['date_heure'].dt.hour
df_defectueux['date'] = df_defectueux['date_heure'].dt.date
```

```
# Fréquence d'indisponibilité par capteur et par heure
indispo_par_heure = df_defectueux.groupby(['id_capteur', 'heure']).size().reset_index(name='
indispo_pivot = indispo_par_heure.pivot(index='id_capteur', columns='heure', values='fréquen
capteurs_uniques = df_defectueux['id_capteur'].unique()

résultat_indisponibilité = {}

for capteur in capteurs_uniques:
    df_capteur = df_defectueux[df_defectueux['id_capteur'] == capteur]
    heures_indispo = df_capteur['heure'].unique()
    heures_indispo.sort()
    résultat_indisponibilité[capteur] = heures_indispo

# Visualisation des indisponibilités
plt.figure(figsize=(16, 10))
sns.heatmap(indispo_pivot, cmap='YlOrRd', annot=True, fmt='.0f', linewidths=.5)
plt.title("Fréquence d'indisponibilité par capteur et par heure", fontsize=16)
plt.xlabel('Heure de la journée', fontsize=14)
plt.ylabel('ID Capteur', fontsize=14)
plt.tight_layout()
plt.show()
```

Tous les capteurs defectueux le sont à toutes les heures durant toutes les journées presentes dans le dataset, soit un total de 720 heures.

Analysons la disponibilité par zone

In [73]:
```
# le nombre total de capteurs par zone dans le dataset complet
capteurs_totaux_par_zone = df.groupby('zone')['id_capteur'].nunique()

# nombre de capteurs défectueux par zone
capteurs_defectueux_par_zone = df_defectueux.groupby('zone')['id_capteur'].nunique().fillna(

# taux d'indisponibilité par zone
```

```
taux_indisponibilite = (capteurs_defectueux_par_zone / capteurs_totaux_par_zone) * 100

# la disponibilité temporelle des capteurs par zone
heures_totales_par_zone = df.groupby('zone').size() / df.groupby('zone')['id_capteur'].nuniq
# Nombre d'heures d'indisponibilité par zone
heures_indispo_par_zone = df_defectueux.groupby('zone').size() / df_defectueux.groupby('zone

print("=== ANALYSE DES TAUX D'INDISPONIBILITÉ PAR ZONE ===")
resultats = pd.DataFrame({
    'Capteurs totaux': capteurs_totaux_par_zone,
    'Capteurs défectueux': capteurs_defectueux_par_zone.fillna(0),
    'Taux indisponibilité (%)': taux_indisponibilite.round(2).fillna(0),
    'Jours d\'observation': heures_totales_par_zone.round(1).fillna(0),
    'Jours d\'indisponibilité': heures_indispo_par_zone.round(1).fillna(0)
}).fillna(0)
resultats
```

```
=== ANALYSE DES TAUX D'INDISPONIBILITÉ PAR ZONE ===
```

Out[73]:

zone	Capteurs totaux	Capteurs défectueux	Taux indisponibilité (%)	Jours d'observation	Jours d'indisponibilité
zoneA	20	9.0	45.0	30.0	30.0
zoneB	25	3.0	12.0	30.0	30.0
zoneC	20	6.0	30.0	30.0	30.0
zoneD	20	0.0	0.0	30.0	0.0

5 - Stratégie de redondance.

Approche globale

- Pour chaque zone indexée par i:
  - T_i : taux d'indisponibilité observé en pourcentage.
  - On en déduit la probabilité de panne

$$p_i = \frac{T_i}{100} \quad \big(\text{entre 0 et 1}\big).$$

  - $N_{0,i}$: nombre initial de capteurs déployés.
- Nous introduisons deux statistiques centrales sur l'ensemble des zones :

$$\bar{p} = \frac{1}{n}\sum_{i=1}^{n} p_i, \quad \sigma_p = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\big(p_i - \bar{p}\big)^2}.$$

- α : niveau de confiance voulu (par exemple 0,95 pour 95 %).
- $k \in [0, 1]$: paramètre de tolérance, exprimé en nombre d'écarts-type.

Plutôt que de fixer une couverture uniforme pour toutes les zones, on définit une **couverture minimale** dépendant de la fiabilité moyenne :

$$C(k) \;=\; \mathrm{clip}\Big(1 - \big(\bar{p} + k\,\sigma_p\big),\, 0,\, 1\Big).$$

- Quand la fiabilité moyenne $\bar{p}$ est faible ou très variable σ_p grand, $C(k)$ baisse automatiquement : on demande moins d'actifs.
- Plus k est grand, plus on accepte de fluctuations, donc $C(k)$ diminue.

Pour chaque zone i :

- On souhaite au moins

$$m_i = \lceil C(k) \times N_{0,i} \rceil$$

  capteurs « actifs » (non en panne).

- On modélise le nombre de capteurs en service par la variable aléatoire

$$X_i \sim \text{Binomial}\big(N_i,\, 1 - p_i\big),$$

  où $N_i \geq N_{0,i}$ est le nombre total (initial + redondants).

- On choisit N_i le plus petit possible tout en garantissant

$$P\big(X_i \geq m_i\big) = 1 - F_{X_i}(m_i - 1) \;\geq\; \alpha.$$

- La redondance requise dans la zone i est alors

$$r_i = N_i - N_{0,i}.$$

Pour la réussite de cette approche il nous faut donc chercher le K optimal. Cette recherche sera l'objet de la prochaine section.

Recherche du K optimal

Pour trouver la meilleure tolérance k^* :

- On balaie k dans $[0, 1]$.
- Pour chaque k , on calcule :
  - $C(k)$ (seuil de couverture dynamique),
  - la somme totale des redondances $R(k) = \sum_i r_i$.
- On normalise

$$R_{\text{norm}}(k) = \frac{R(k) - \min R}{\max R - \min R}, \quad C_{\text{norm}}(k) = \frac{C(k) - \min C}{\max C - \min C}.$$

- On définit la fonction objectif

$$J(k) = R_{\text{norm}}(k) \;+\; \big(1 - C_{\text{norm}}(k)\big).$$

- On choisit

$$k^* = \arg\min_{k \in [0,1]} J(k).$$

In [74]:
```
# Taux de panne observé par zone
resultats['p_obs'] = resultats['Taux indisponibilité (%)'] / 100.0

alpha   = 0.95
p_mean  = resultats['p_obs'].mean()
```

```
p_std   = resultats['p_obs'].std()

k_vals   = np.linspace(0, 1, 100)
tot_red  = []
C_vals   = []

for kv in k_vals:
    C_val = float(np.clip(1 - (p_mean + kv * p_std), 0, 1))
    C_vals.append(C_val)
    red_counts = []
    for _, row in resultats.iterrows():
        total   = int(row['Capteurs totaux'])
        p_obs   = row['p_obs']
        couv_req = int(np.ceil(C_val * total))
        N = total
        while 1 - binom.cdf(couv_req - 1, N, 1 - p_obs) < alpha:
            N += 1
        red_counts.append(N - total)
    tot_red.append(sum(red_counts))

# Optimisation du compromis entre redondance et couverture
tr      = np.array(tot_red)
cv      = np.array(C_vals)
tr_n    = (tr - tr.min())/(tr.max()-tr.min())
cv_n    = (cv - cv.min())/(cv.max()-cv.min())
obj     = tr_n + (1 - cv_n)
best_i  = np.argmin(obj)
k_comp  = k_vals[best_i]

print(f"k compromis qui minimise redondants et maximise couverture) = {k_comp:.2f}")
k = k_comp
```

```
k compromis qui minimise redondants et maximise couverture) = 0.17
```

Notre k optimal déterminé est 0.17

Determinons le nombre de capteurs par zone pour le k optimal trouvé précedemment

Dans cette étape, nous appliquons notre stratégie avec le k optimal

In [75]:

```
print(f"Taux de panne moyen p_obs : {p_mean:.3f}")
print(f"Écart-type de p_obs : {p_std:.3f}")

# seuil de couverture  C à partir de la distribution de p_obs
C = 1 - (p_mean + k * p_std)
C = float(np.clip(C, 0.0, 1.0))
print(f"Taux de couverture C calculé : {C:.3f}")

# la redondance par zone
capteurs_red = []
for _, row in resultats.iterrows():
    total  = int(row['Capteurs totaux'])
    p_obs  = row['p_obs']
    # nombre minimal de capteurs vivants à garantir
    couv_req = int(np.ceil(C * total))
    if couv_req <= 0:
        capteurs_red.append(0)
        continue

    N = total
    # on cherche N minimal tel que P(X ≥ couv_req) ≥ alpha
    while 1 - binom.cdf(couv_req - 1, N, 1 - p_obs) < alpha:
```

```
        N += 1
    capteurs_red.append(N - total)


resultats['C_zone']  = C
resultats['capteurs_redondants_a_ajouter'] = capteurs_red
resultats[[
    'Capteurs totaux',
    'Taux indisponibilité (%)',
    'C_zone',
    'capteurs_redondants_a_ajouter'
]]
```

```
Taux de panne moyen p_obs : 0.218
Écart-type de p_obs : 0.198
Taux de couverture C calculé : 0.748
```

Out[75]:

zone	Capteurs totaux	Taux indisponibilité (%)	C_zone	capteurs_redondants_a_ajouter
zoneA	20	45.0	0.748491	16
zoneB	25	12.0	0.748491	0
zoneC	20	30.0	0.748491	7
zoneD	20	0.0	0.748491	0

Notre stratégie de redondance s'active à un seuil de couverture de 74,8 %. Elle consiste à placer 16 capteurs en Zone A , 7 en Zone C, 0 en Zone B et D

II- Détection de capteurs défaillants par détection de valeurs aberrantes

1 - Chargement des données sans labels.

Dans cette partie nous allons charger les données en nous servant du fichier "capteur_sol_sans_label.csv".

Nous affichons le Dataframe obtenu pour nous assurer de la bonne importaion.

In [76]:
```
# Lire le jeu de données
df_sans_label = pd.read_csv("capteur_sol_sans_label.csv", sep=',')

# Dimension du DataFrame
print("Dimension du DataFrame : ", df_sans_label.shape) # Dimension du DataFrame

# Afficher les 10 premieres lignes
df_sans_label.head(10)
```

```
Dimension du DataFrame :  (61200, 5)
```

Out[76]:

	id_capteur	zone	date_heure	humidite_sol	temperature_sol
0	SC_A_1	zoneA	2024-04-01 00:00:00	23.51	12.85
1	SC_A_1	zoneA	2024-04-01 01:00:00	22.58	12.88
2	SC_A_1	zoneA	2024-04-01 02:00:00	14.19	11.61
3	SC_A_1	zoneA	2024-04-01 03:00:00	25.16	12.01
4	SC_A_1	zoneA	2024-04-01 04:00:00	24.90	11.05
5	SC_A_1	zoneA	2024-04-01 05:00:00	11.22	13.26
6	SC_A_1	zoneA	2024-04-01 06:00:00	23.50	12.70
7	SC_A_1	zoneA	2024-04-01 07:00:00	30.61	12.59
8	SC_A_1	zoneA	2024-04-01 08:00:00	31.74	11.95
9	SC_A_1	zoneA	2024-04-01 09:00:00	25.35	12.74

Nos données sont bien chargées et on a 61200 lignes et 5 colonnes.

Verifions l' existence de valeurs manquantes dans le dataset

In [77]:
```
# Valurs manquantes
print("Valeurs manquantes dans le DataFrame :")
print(df_sans_label.isnull().sum())
```

```
Valeurs manquantes dans le DataFrame :
id_capteur         0
zone               0
date_heure         0
humidite_sol       0
temperature_sol    0
dtype: int64
```

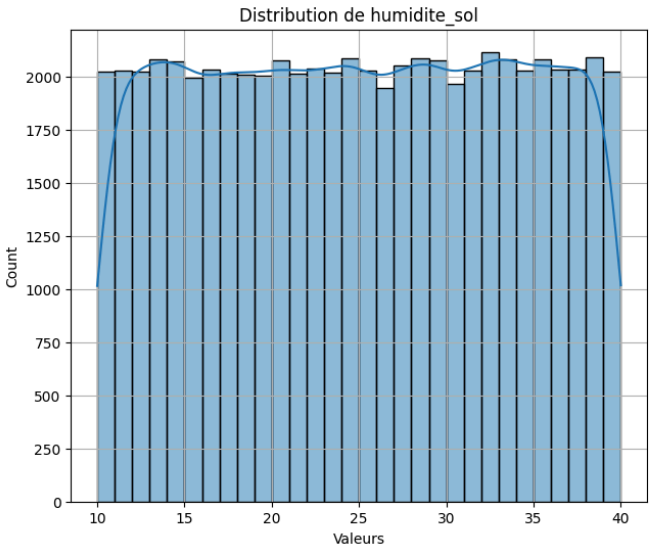
Il n' y a pas de données manquantes.

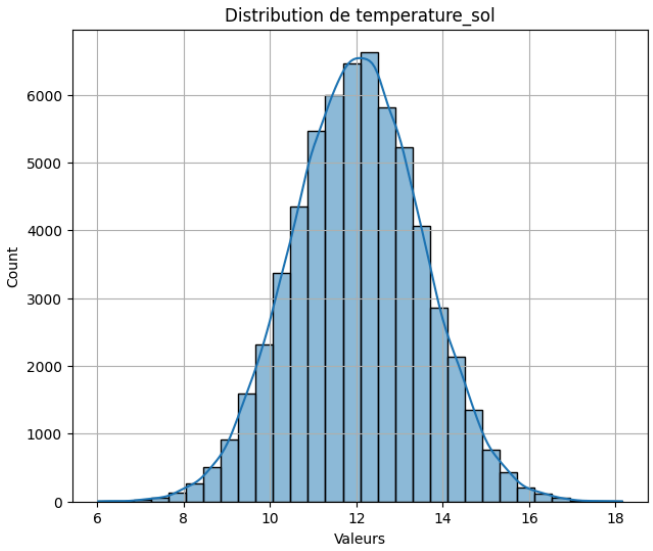
2 - Détection d'anomalies par Z-score et IQR.

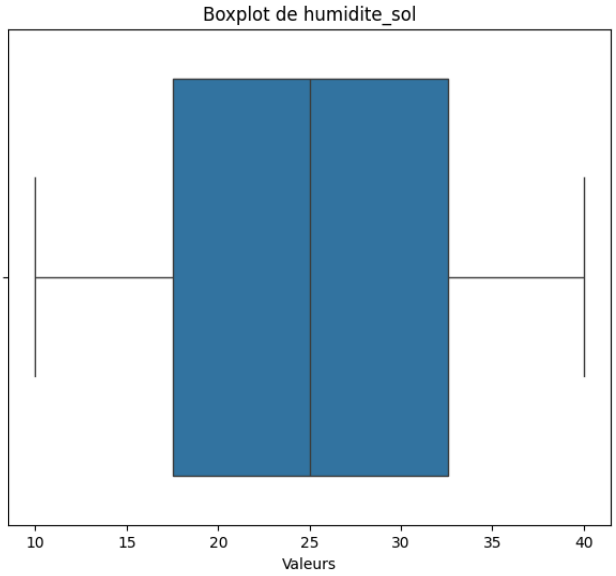
Nous allons procéder dans cette partie à la detection des anomalies sur notre dataset. Mais en amont , nous allons afficher les distributions de chaque colonne numerique.

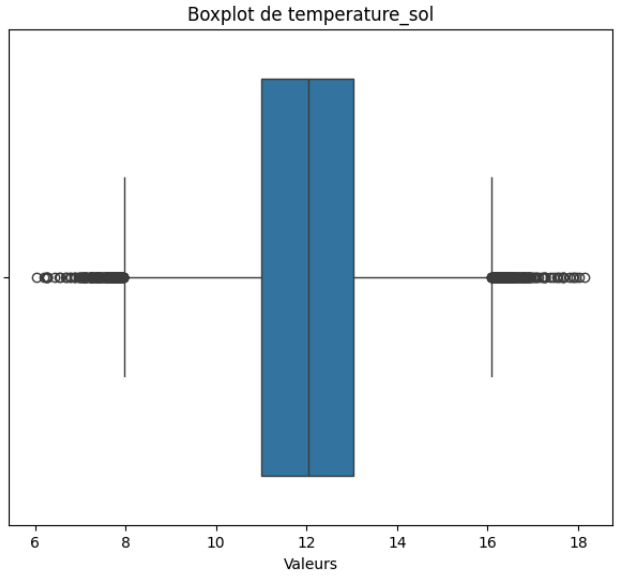
In [78]:
```
# Selectionner les colonnes numériques
colonnes_numeriques = df_sans_label.select_dtypes(include=[np.number]).columns.tolist()

# Afficher la distribution des colonnes numériques ("humidite_sol", "temperature_sol") , boti
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))
sns.histplot(df_sans_label['humidite_sol'], kde=True, bins=30, ax=axes[0])
axes[0].set_title('Distribution de humidite_sol')
sns.histplot(df_sans_label['temperature_sol'], kde=True, bins=30, ax=axes[1])
axes[1].set_title('Distribution de temperature_sol')
for ax in axes:
    ax.set_xlabel('Valeurs')
    ax.grid()

# Afficher les boxplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))
sns.boxplot(x=df_sans_label['humidite_sol'], ax=axes[0])
axes[0].set_title('Boxplot de humidite_sol')
sns.boxplot(x=df_sans_label['temperature_sol'], ax=axes[1])
```

```
axes[1].set_title('Boxplot de temperature_sol')
for ax in axes:
    ax.set_xlabel('Valeurs')
```









On constate :

- que temperature_sol suit une loi gaussienne, la methode du Z-score devrait bien fonctionner sur cette colonne. Quant à humidité sol, on va utiliser l' IQR.
- que humidite_sol ne contient pas de valeurs aberrantes.

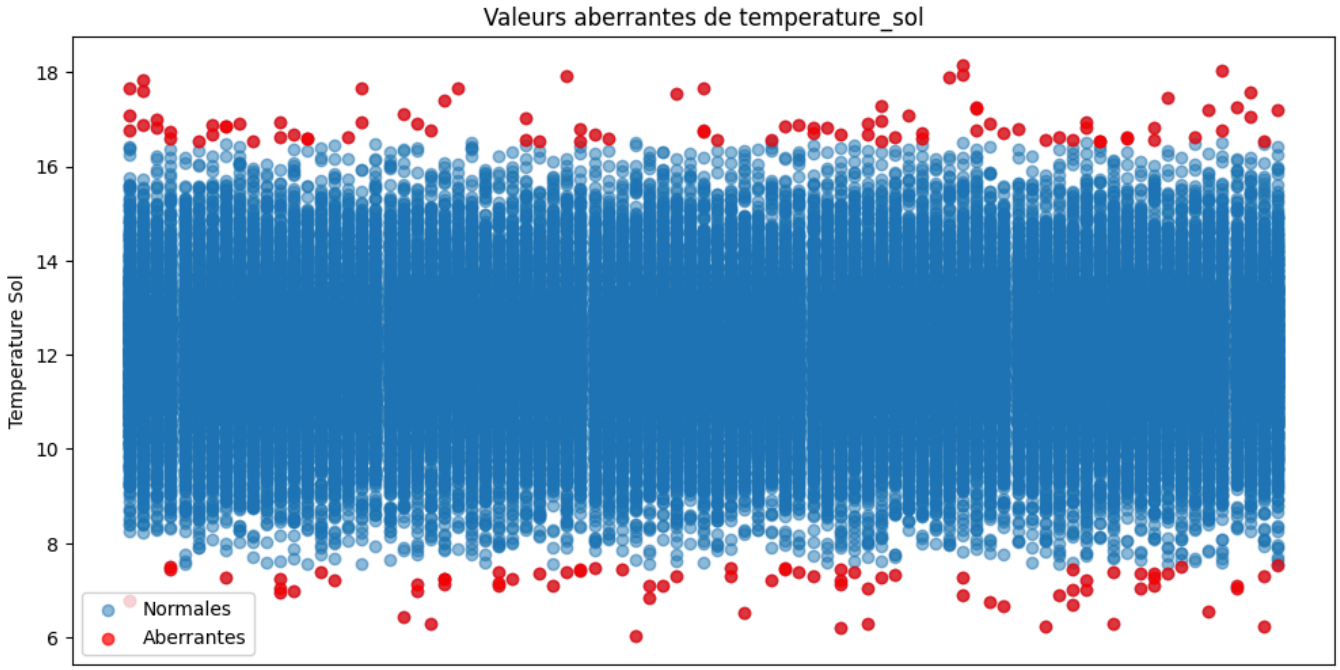
Cas de temperature_sol

In [79]:
```
# Detection des valeurs aberantes avec la méthode du Z-score sur température_sol
z_scores = np.abs(stats.zscore(df_sans_label['temperature_sol']))
seuil_z = 3
valeurs_aberrantes_z = df_sans_label[z_scores > seuil_z]

# Compter le nombre de valeurs aberrantes
nombre_valeurs_aberrantes_z = valeurs_aberrantes_z.shape[0]
print(f"Nombre de lignes contenant des valeurs aberrantes pour temperature_sol : {nombre_val

# Nombre de capteur concernés par ces valeurs
capteurs_aberrants = valeurs_aberrantes_z['id_capteur'].nunique()
print(f"Nombre de capteurs uniques concernés par les valeurs aberrantes : {capteurs_aberrant

# Visualisation graphique des valeurs abberrantes
plt.figure(figsize=(12, 6))
plt.scatter(df_sans_label['id_capteur'], df_sans_label['temperature_sol'], label='Normales',
plt.scatter(valeurs_aberrantes_z['id_capteur'], valeurs_aberrantes_z['temperature_sol'], col
```

```
plt.xticks([])  # Désactiver l'affichage des labels de l'axe x
plt.xticks(rotation=45)
plt.title('Valeurs aberrantes de temperature_sol')
plt.ylabel('Temperature Sol')
plt.legend()
```

```
Nombre de lignes contenant des valeurs aberrantes pour temperature_sol : 165
Nombre de capteurs uniques concernés par les valeurs aberrantes : 75
```

Out[79]: <matplotlib.legend.Legend at 0x20daae0dd10>



La methode du Z-score detecte 165 lignes contenant des valeurs aberrantes dans la colonne temperature_sol

3 - Attribution automatique des labels DEFAILLANT ou OK.

Dans cette section nous allons attribuer les labels manquants en nous basant sur le fait qu'il y a defaillance dès que les données sont aberrantes. Nous utiliserons les valeurs aberrantes détectées dans temperature_sol

In [80]:
```
df_label_reconstitue =  df_sans_label.copy()

df_label_reconstitue['etat_capteur'] = 'OK'  # Initialiser tous les labels à 'OK'

# Attribution du label 'DEFAILLANT' pour les valeurs aberrantes
df_label_reconstitue.loc[valeurs_aberrantes_z.index, 'etat_capteur'] = 'DEFAILLANT'

df_label_reconstitue.head(10)
```

Out[80]:

	id_capteur	zone	date_heure	humidite_sol	temperature_sol	etat_capteur
0	SC_A_1	zoneA	2024-04-01 00:00:00	23.51	12.85	OK
1	SC_A_1	zoneA	2024-04-01 01:00:00	22.58	12.88	OK
2	SC_A_1	zoneA	2024-04-01 02:00:00	14.19	11.61	OK
3	SC_A_1	zoneA	2024-04-01 03:00:00	25.16	12.01	OK
4	SC_A_1	zoneA	2024-04-01 04:00:00	24.90	11.05	OK
5	SC_A_1	zoneA	2024-04-01 05:00:00	11.22	13.26	OK
6	SC_A_1	zoneA	2024-04-01 06:00:00	23.50	12.70	OK
7	SC_A_1	zoneA	2024-04-01 07:00:00	30.61	12.59	OK
8	SC_A_1	zoneA	2024-04-01 08:00:00	31.74	11.95	OK
9	SC_A_1	zoneA	2024-04-01 09:00:00	25.35	12.74	OK

L' attribution des labels pour l' etat du capteur a été réussi.

- Comparaison des labels attribués aux labels originaux du fichier enrichi.

In [81]:
```
# Comparaison des labels attribués aux labels originaux du fichier enrichi
df_comparaison = df[['id_capteur', 'etat_capteur']].merge(df_label_reconstitue[['id_capteur'

# Calcul de la précision
précision = (df_comparaison['etat_capteur_original'] == df_comparaison['etat_capteur_reconst
print(f"Précision de la reconstitution des labels : {précision:.2f}")
```

```
Précision de la reconstitution des labels : 0.79
```

Nous arrivons à correctement reconstituer 79 % des labels des capteurs.

4 - Génération de fichiers labellisés.

Apres l' attribution des labels et le calcul de la précision de notre methode d' attribution, nous allons sauvegarder les fichier généré dans cette partie.

In [82]:
```
# Génération du fichier CSV avec les labels
df_label_reconstitue.to_csv("capteur_sol_label_reconstitue.csv", index=False)
```

Le fichier est bien enrégistré au format csv.