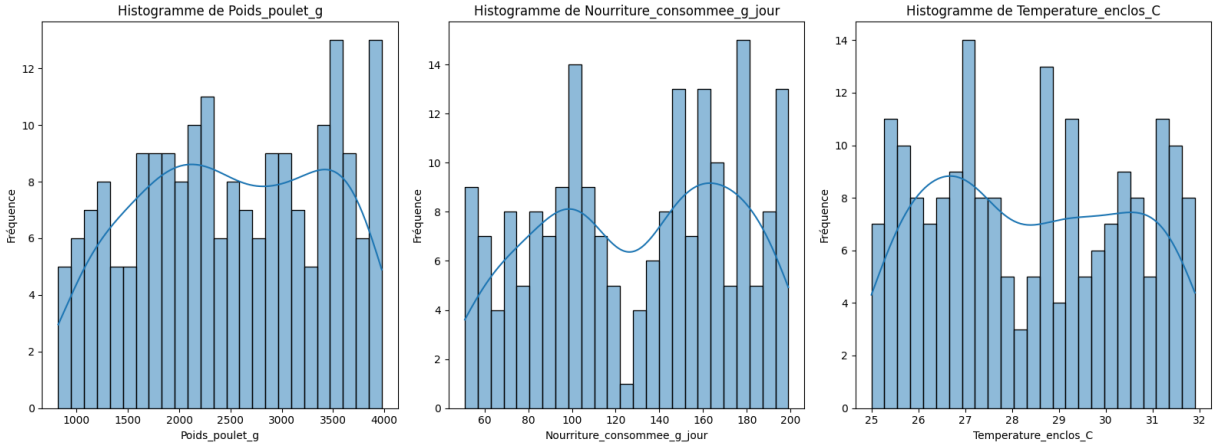
TP2 ATDN2

Alexis XIONG

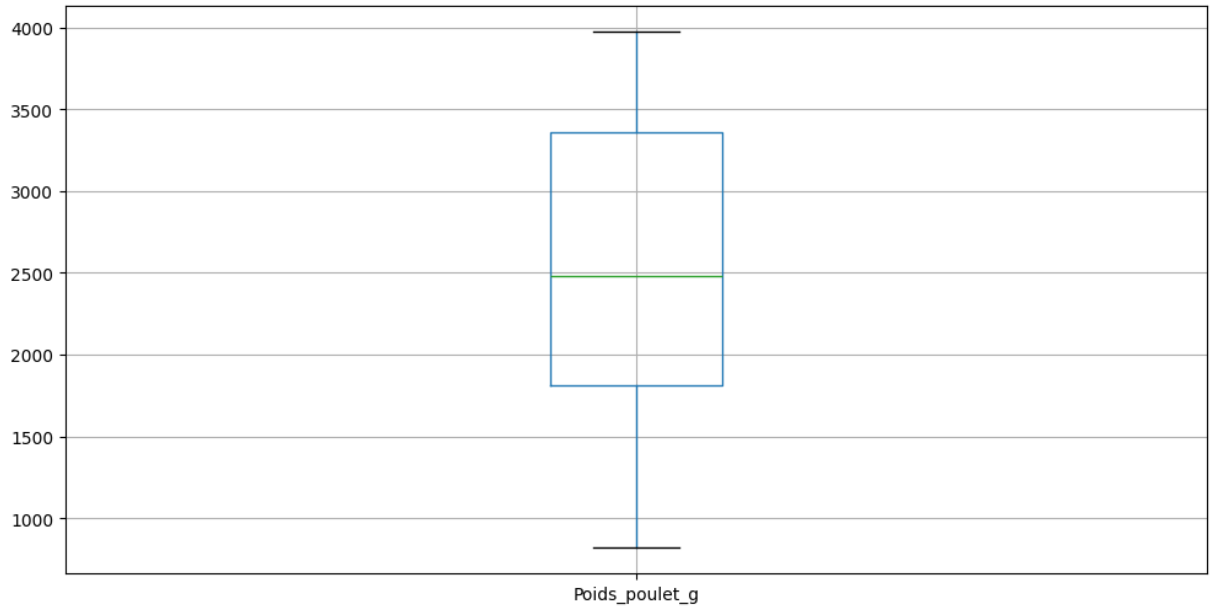
Exercice 1

In [88]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv('donnees_elevage_poulet.csv')

variables = ['Poids_poulet_g', 'Nourriture_consommee_g_jour', 'Temperature_e
stats = data[variables].describe()
variance = data[variables].var()

print(stats)
print(variance)

plt.figure(figsize=(16, 6))
for i, var in enumerate(variables):
    plt.subplot(1, 3, i + 1)
    sns.histplot(data[var], bins=25, edgecolor='black', kde = True)
    plt.title(f'Histogramme de {var}')
    plt.xlabel(var)
    plt.ylabel('Fréquence')

plt.tight_layout()
plt.show()

for i in variables:
    data[[i]].boxplot(figsize=(12, 6))
    plt.suptitle("Histogrammes des variables")
    plt.show()
```

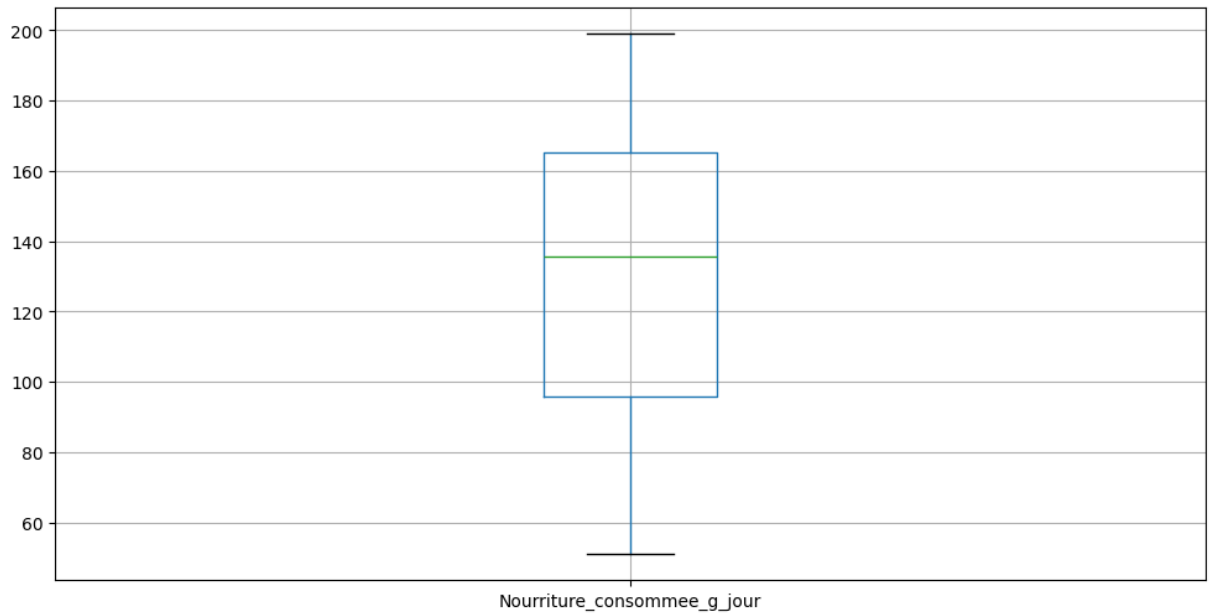
```
       Poids_poulet_g  Nourriture_consommee_g_jour  Temperature_enclos_C
count      200.000000                   200.000000            200.000000
mean      2509.580000                   129.745000             28.389000
std        898.436875                    44.006166              2.065724
min        821.000000                    51.000000             25.000000
25%       1810.750000                    95.750000             26.600000
50%       2481.500000                   135.500000             28.500000
75%       3356.500000                   165.250000             30.300000
max       3974.000000                   199.000000             31.900000
Poids_poulet_g                 807188.817688
Nourriture_consommee_g_jour      1936.542688
Temperature_enclos_C                4.267215
dtype: float64
```

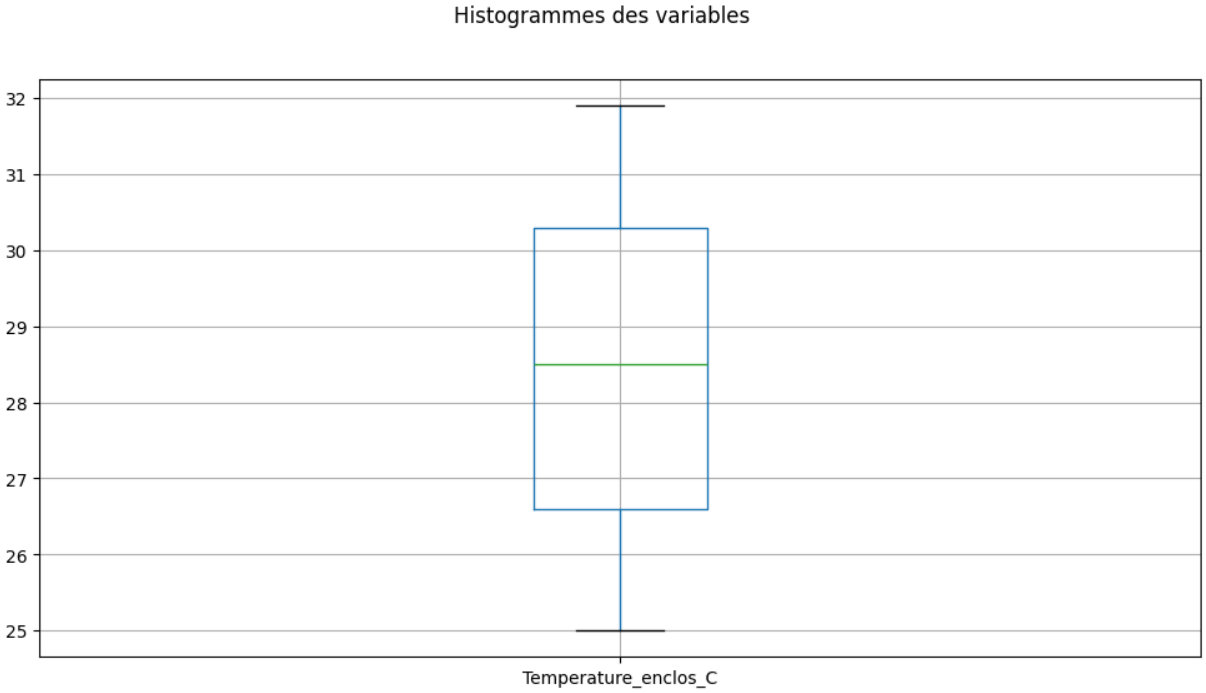


Histogrammes des variables



Histogrammes des variables





Les histogrammes nous permettent de voir la forme de la distribution des variables et les boxplots nous montrent les potentielles valeurs aberrantes.

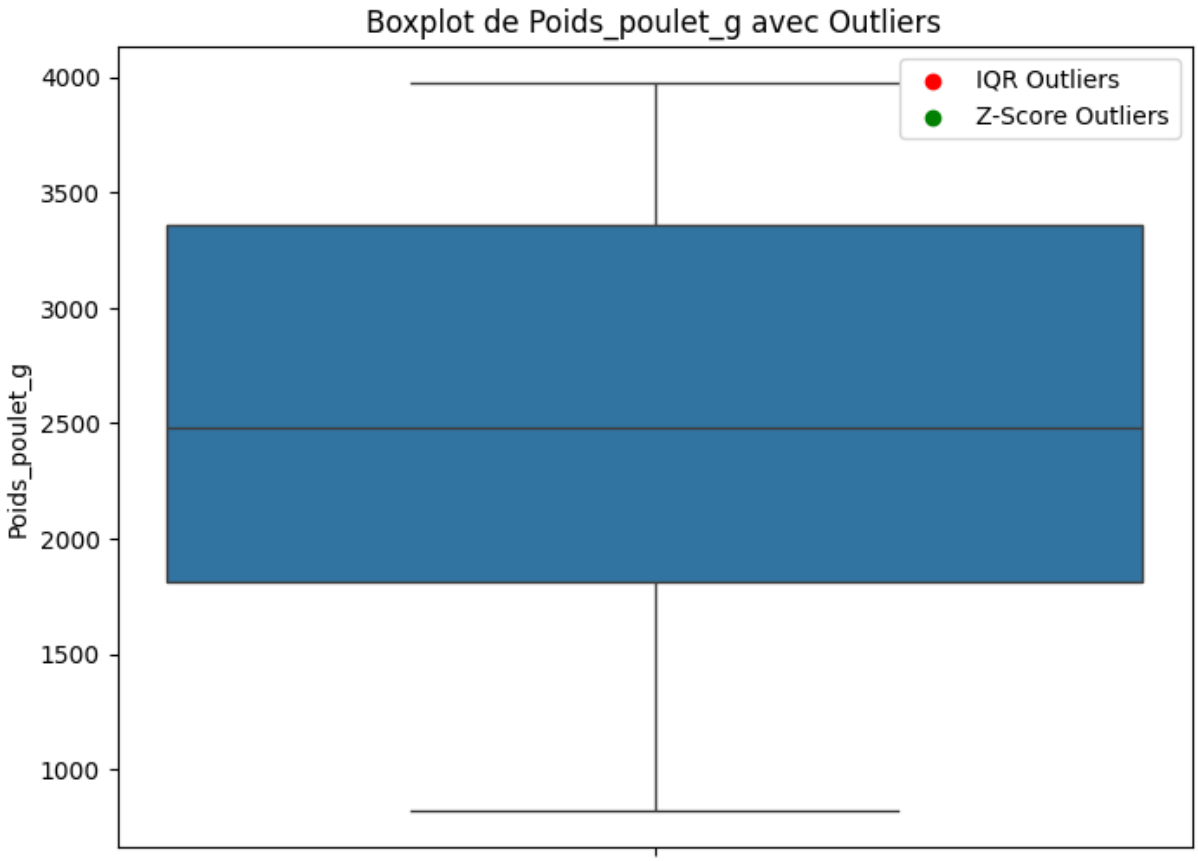
Selon les graphiques, les données semblent homogènes mis à part la température des enclos où l'on peut observer une valeur beaucoup plus basse que les autres.

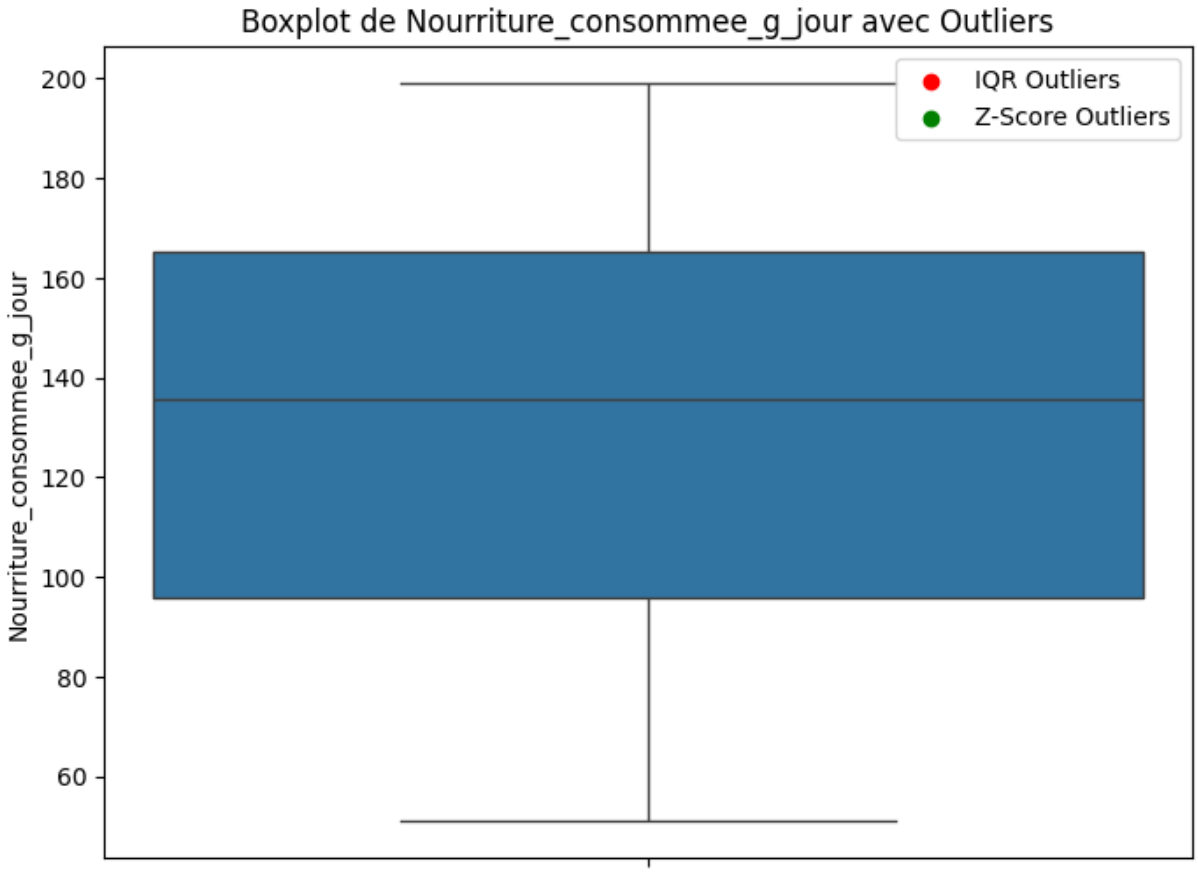
Exercice 2

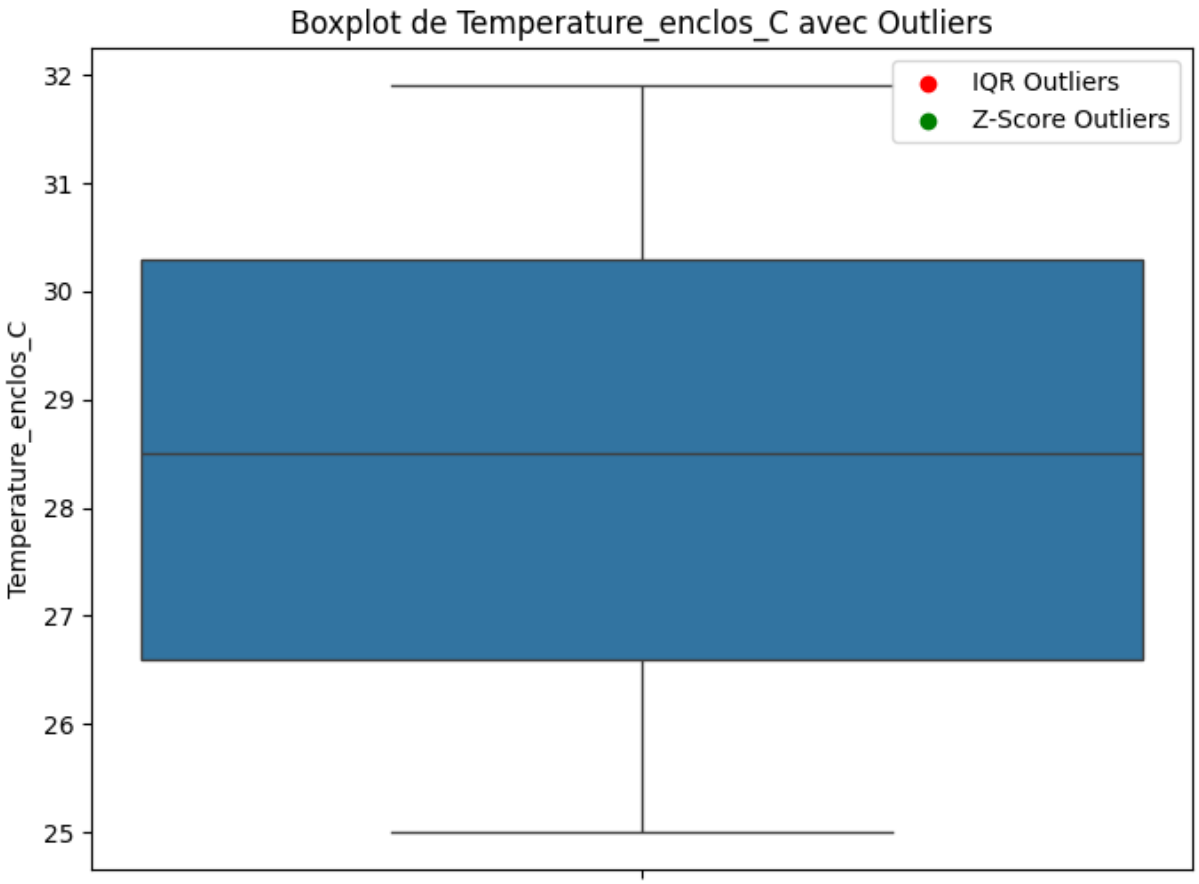
In [89]:
```
import numpy as np
import seaborn as sns

def methode_IQR(data, variable, k=1.5):
    Q1 = data[variable].quantile(0.25)
    Q3 = data[variable].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - k * IQR
    upper_bound = Q3 + k * IQR
    outliers = data[(data[variable] < lower_bound) | (data[variable] > upper
    return outliers

def methode_Z_score(data, variable, threshold=2):
    moyenne = data[variable].mean()
    ecart_type = data[variable].std()
    z_score = np.abs((data[variable] - moyenne) / ecart_type)
    outliers = data[z_score > threshold]
    return outliers

outliers_iqr = {}
outliers_zscore = {}

for var in variables:
```

```
    outliers_iqr[var] = methode_IQR(data, var)
    outliers_zscore[var] = methode_Z_score(data, var)

    print(f"Variable : {var}")
    print(f"  IQR : {len(outliers_iqr[var])} outliers")
    print(f"  Z-Score : {len(outliers_zscore[var])} outliers")

for var in variables:
    plt.figure(figsize=(8, 6))
    sns.boxplot(y=data[var])

    outliersIQR = outliers_iqr[var][var].values
    outliersZSCORE = outliers_zscore[var][var].values

    plt.scatter(np.zeros_like(outliersIQR), outliersIQR, color='red', label=
    plt.scatter(np.zeros_like(outliersZSCORE), outliersZSCORE, color='green'

    plt.title(f'Boxplot de {var} avec Outliers')
    plt.ylabel(var)
    plt.legend()
    plt.show()
```

```
Variable : Poids_poulet_g
  IQR : 0 outliers
  Z-Score : 0 outliers
Variable : Nourriture_consommee_g_jour
  IQR : 0 outliers
  Z-Score : 0 outliers
Variable : Temperature_enclos_C
  IQR : 0 outliers
  Z-Score : 0 outliers
```


Boxplot de Poids_poulet_g avec Outliers
IQR Outliers
Z-Score Outliers
Poids_poulet_g
4000
3500
3000
2500
2000
1500
1000


Boxplot de Nourriture_consommee_g_jour avec Outliers
IQR Outliers
Z-Score Outliers
Nourriture_consommee_g_jour
200
180
160
140
120
100
80
60



Pour trouver des outliers, pour le cas de IQR, la variable k va influencer le nombre de outlier, plus il tend vers 0 et plus il trouvera des outliers. Pour le cas du Z-Score, à partir du moment ou thresold descend en dessous de 1.5, le nombre d'outlier augmente vite.

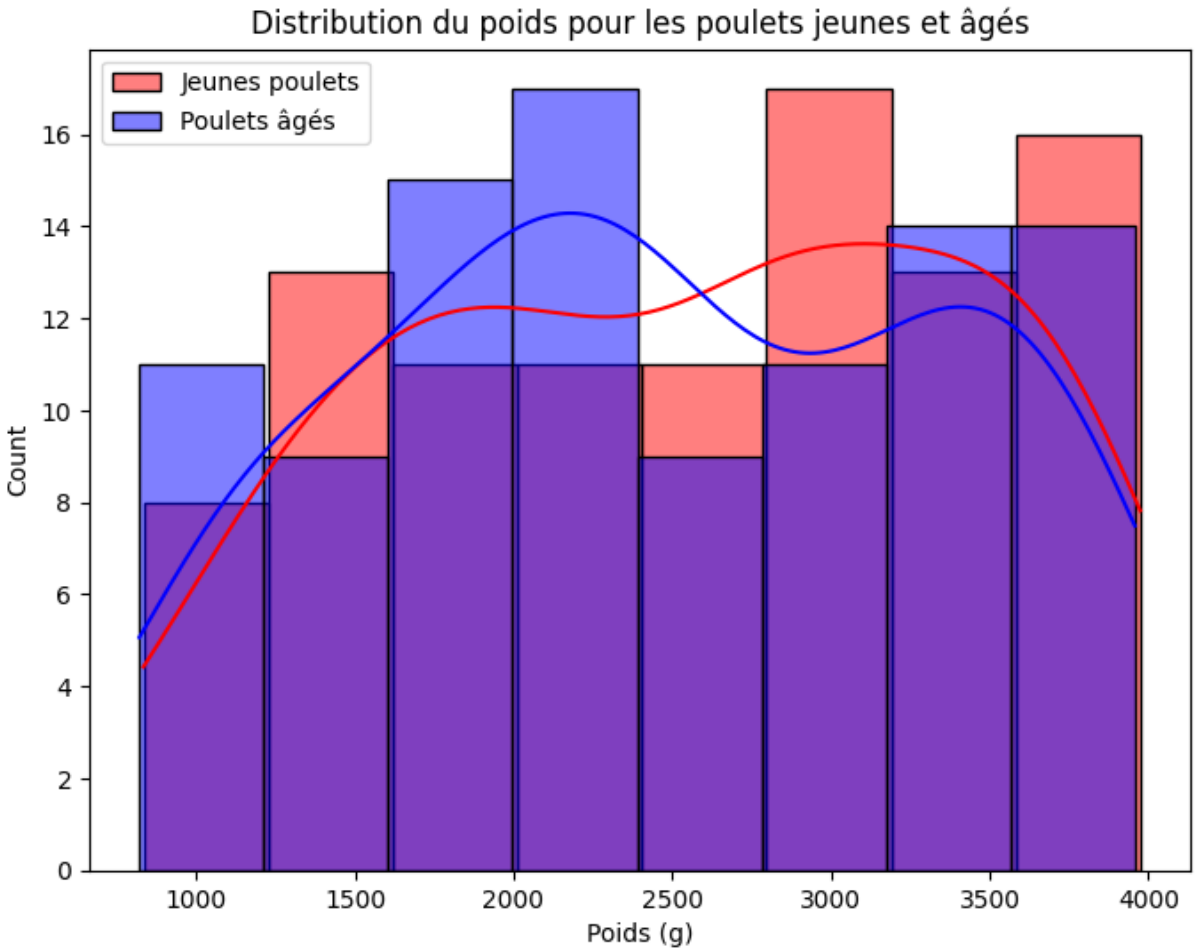
Les outliers détectés peuvent être des cas réalistes ou bien des erreurs. Dans le cas des erreurs, il est préférable de les supprimer car ils peuvent biaiser les résultats que l'on va obtenir. Mais si les erreurs sont réalistes, ils faut les conserver afin d'avoir une réelle disparité des données.

Exercice 3

In [90]:
```
from scipy import stats
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

# Shapiro-Wilk
print("Test de Shapiro-Wilk :")
for var in variables:
    stat, p = stats.shapiro(data[var])
    shapiro_test = stats.shapiro(data[var])
    print(f"  Statistique = {stat}, p-value = {p}")
    if shapiro_test.pvalue > 0.05:
        print(f"  => {var} suit une distribution normale")
    else:
```

```
        print(f"  => {var} ne suit pas une distribution normale")

# t student
print("Test t de Student :")

# Comme je ne sais pas quoi comparer, je vais comparer l'age et le poids des
median_age = data['Age_poulet_jours'].median()
group1 = data[data['Age_poulet_jours'] < median_age]['Poids_poulet_g']
group2 = data[data['Age_poulet_jours'] >= median_age]['Poids_poulet_g']

t_stat, t_p = stats.ttest_ind(group1, group2)
if t_p > 0.05:
    print("Il n'y a pas de grande différence.")
else:
    print("Il y a une grande différence.")
print(f"  Statistique t = {t_stat}, p-value = {t_p}")

plt.figure(figsize=(8, 6))
sns.histplot(group1, kde=True, label='Jeunes poulets', color='red')
sns.histplot(group2, kde=True, label='Poulets âgés', color='blue')
plt.title("Distribution du poids pour les poulets jeunes et âgés")
plt.xlabel("Poids (g)")
plt.legend()
plt.show()

# Cette fois je divise l'age des poules en 3 parties : jeune, moyen, âgé
data['Age_group'] = pd.cut(data['Age_poulet_jours'], bins=3, labels=['Jeune'

groups = [data[data['Age_group'] == group]['Poids_poulet_g'] for group in da

f_stat, f_p = stats.f_oneway(*groups) # Anova
print(f"  Statistique F = {f_stat}, p-value = {f_p}")
if t_p > 0.05:
    print("Il n'y a pas de grande différence.")
else:
    print("Il y a une grande différence.")
```

```
Test de Shapiro-Wilk :
  Statistique = 0.9568221670349863, p-value = 9.098264233228524e-06
  => Poids_poulet_g ne suit pas une distribution normale
  Statistique = 0.9448708208372757, p-value = 6.230563751996703e-07
  => Nourriture_consommee_g_jour ne suit pas une distribution normale
  Statistique = 0.943209717135969, p-value = 4.4060638371198676e-07
  => Temperature_enclos_C ne suit pas une distribution normale
Test t de Student :
Il n'y a pas de grande différence.
  Statistique t = 0.5863209665844412, p-value = 0.5583276593550339
```




```
  Statistique F = 0.22116942475925008, p-value = 0.8017796241820304
Il n'y a pas de grande différence.
```

D'après les résultats avec Shapiro, on remarque qu'aucune des trois variables ne suit une distribution normale.

Pour le test t de Student, on remarque que p-value est supérieure à 0.05. De plus, sur le graphique on remarque que les deux graphiques sont assez similaires donc il n'y a pas de différence significative entre les moyennes des deux groupes.

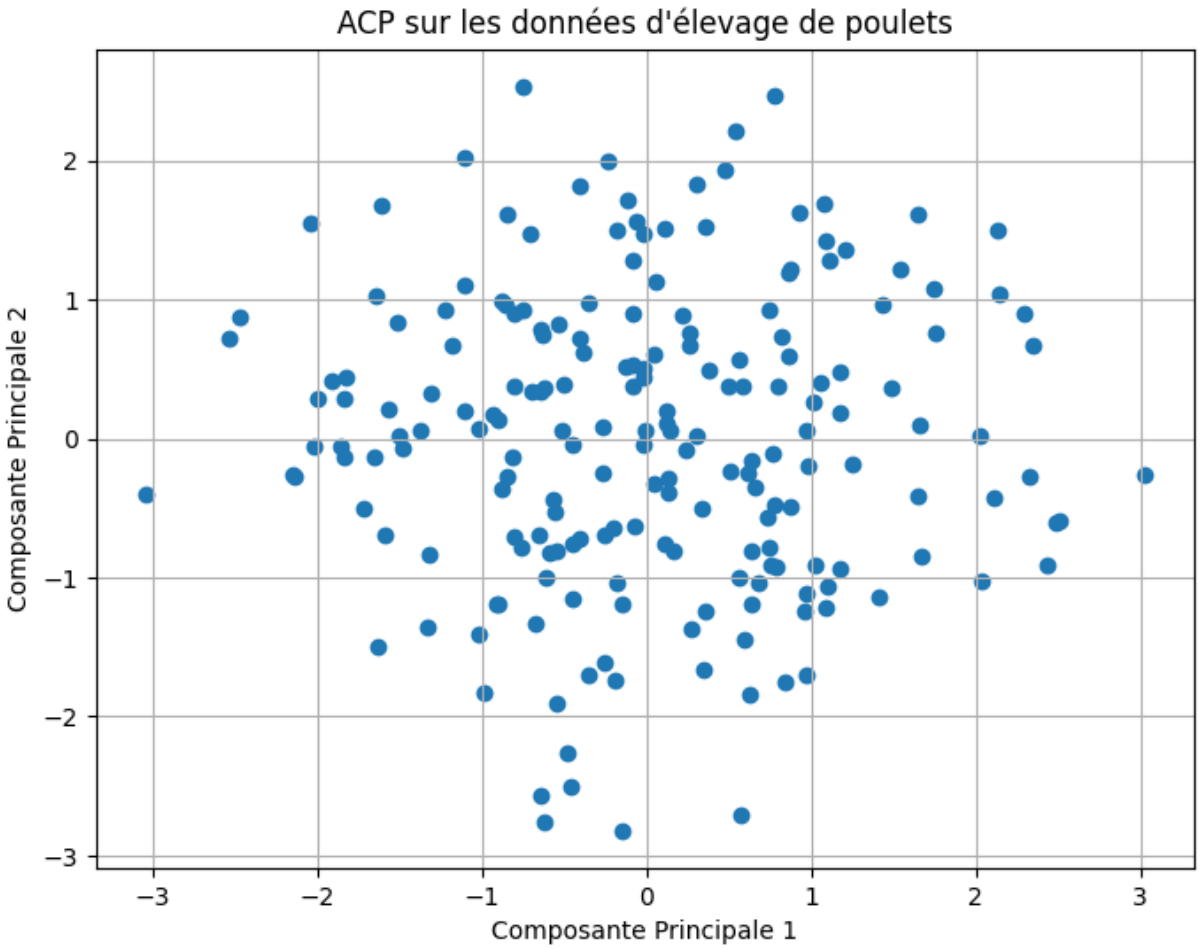
Pour le test avec la table ANOVA, il m'indique également que chaque p-value est supérieure à 0.05. Il n'y a donc pas de différence significative entre les moyennes des deux groupes.

Exercice 4

In [91]:
```
from sklearn.preprocessing import StandardScaler

data = pd.read_csv('donnees_elevage_poulet.csv')

variables = ['Poids_poulet_g', 'Nourriture_consommee_g_jour', 'Temperature_e
data_subset = data[variables]
```

```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_subset)

matrice_de_covariance = np.cov(data_scaled, rowvar=False)

# Vecteurs propres calculs
valeurs, vecteurs = np.linalg.eig(matrice_de_covariance)
valeurs_triés = valeurs[np.argsort(valeurs)[::-1]]
vecteurs_triés = vecteurs[:, np.argsort(valeurs)[::-1]]

print("Matrice de covariance :")
print(matrice_de_covariance)

print("\nValeurs propres triées :")
print(valeurs_triés)

print("\nVecteurs propres triés :")
print(vecteurs_triés)

composantes_principales = 2

vecteurs_sousensemble = vecteurs_triés[:, 0:composantes_principales]
X_reduced = np.dot(data_scaled, vecteurs_sousensemble)

principal_df = pd.DataFrame(X_reduced, columns = ['Composante Principale 1',

plt.figure(figsize=(8,6))
plt.scatter(principal_df['Composante Principale 1'], principal_df['Composant
plt.xlabel('Composante Principale 1')
plt.ylabel('Composante Principale 2')
plt.title('ACP sur les données d\'élevage de poulets')
plt.grid()
plt.show()
```

```
Matrice de covariance :
[[ 1.00502513 -0.08194588  0.01915299  0.07643343 -0.04073578  0.02797066
  -0.1190977  -0.02983038]
 [-0.08194588  1.00502513 -0.18661903 -0.02405565 -0.10737917 -0.06560509
  -0.0453192   0.05806111]
 [ 0.01915299 -0.18661903  1.00502513 -0.02704035 -0.06423386  0.05927977
  -0.0021131   0.09818369]
 [ 0.07643343 -0.02405565 -0.02704035  1.00502513 -0.00174419 -0.010359
   0.02476582  0.05087189]
 [-0.04073578 -0.10737917 -0.06423386 -0.00174419  1.00502513  0.0298505
  -0.03029875  0.06285473]
 [ 0.02797066 -0.06560509  0.05927977 -0.010359    0.0298505   1.00502513
  -0.01496235  0.07228268]
 [-0.1190977  -0.0453192  -0.0021131   0.02476582 -0.03029875 -0.01496235
   1.00502513 -0.09485815]
 [-0.02983038  0.05806111  0.09818369  0.05087189  0.06285473  0.07228268
  -0.09485815  1.00502513]]

Valeurs propres triées :
[1.27246941 1.15189625 1.10921505 1.05997308 1.01983233 0.95158043
 0.78352396 0.6917105 ]

Vecteurs propres triés :
[[ 0.30000215 -0.19015388 -0.68853542  0.09674953 -0.17035845 -0.10544206
  -0.57745754 -0.12962048]
 [-0.56060745 -0.45084781  0.07098187 -0.2515305   0.10660948 -0.15664932
  -0.13787455 -0.59936261]
 [ 0.54823689  0.21332517  0.07819074 -0.4812339   0.09284161  0.31992439
   0.0776947  -0.54701113]
 [ 0.10039135 -0.1339213  -0.33766657  0.34321119  0.77411356 -0.00153442
   0.35540901 -0.12084829]
 [ 0.17387055 -0.08345863  0.40990537  0.74102354 -0.19159612  0.1598618
  -0.12769807 -0.40877099]
 [ 0.389066   -0.08146334  0.21928201 -0.04946821 -0.02290654 -0.87595753
   0.14010977 -0.0632662 ]
 [-0.17160059  0.63506821  0.16087804  0.0453308   0.42934794 -0.19393787
  -0.55999652 -0.061297  ]
 [ 0.27207681 -0.52935995  0.39844609 -0.15612201  0.36078749  0.17753788
  -0.40699842  0.36780953]]
```



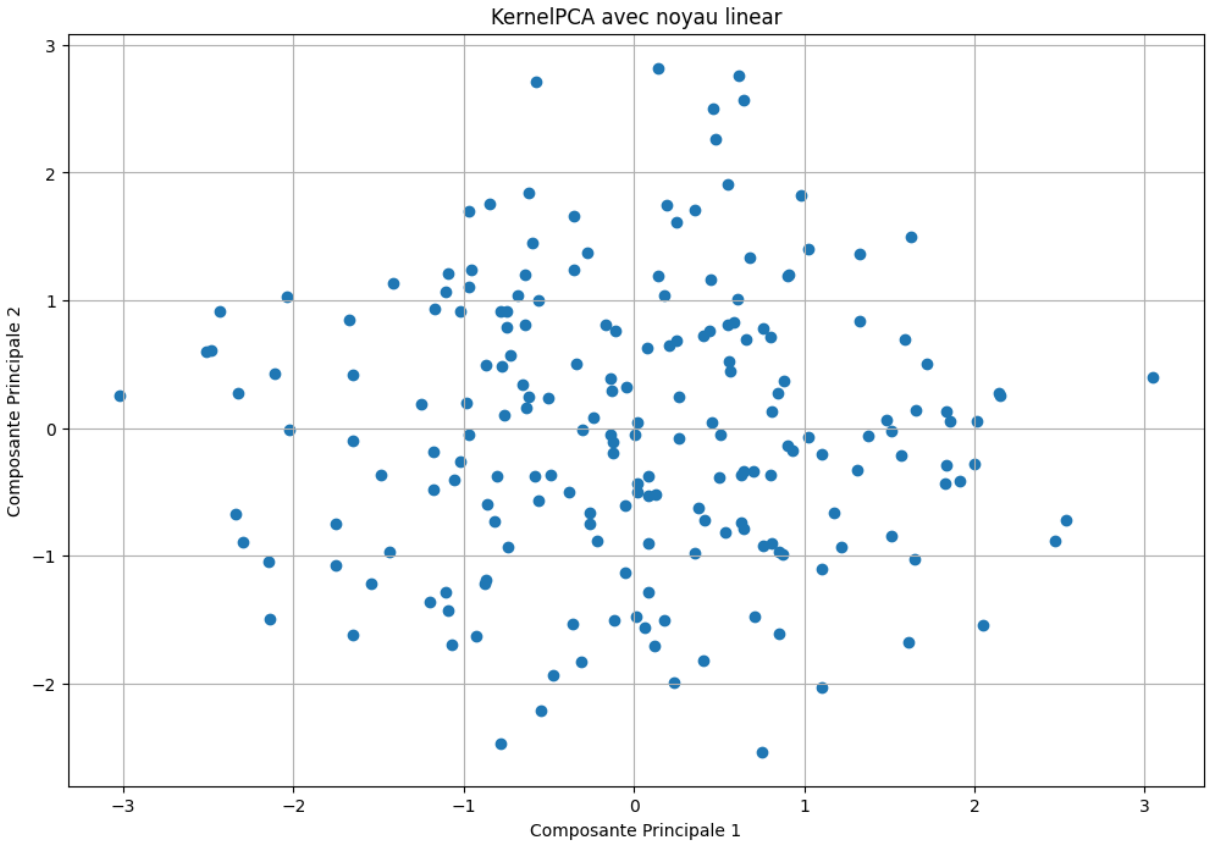
Exercice 5

In [92]:

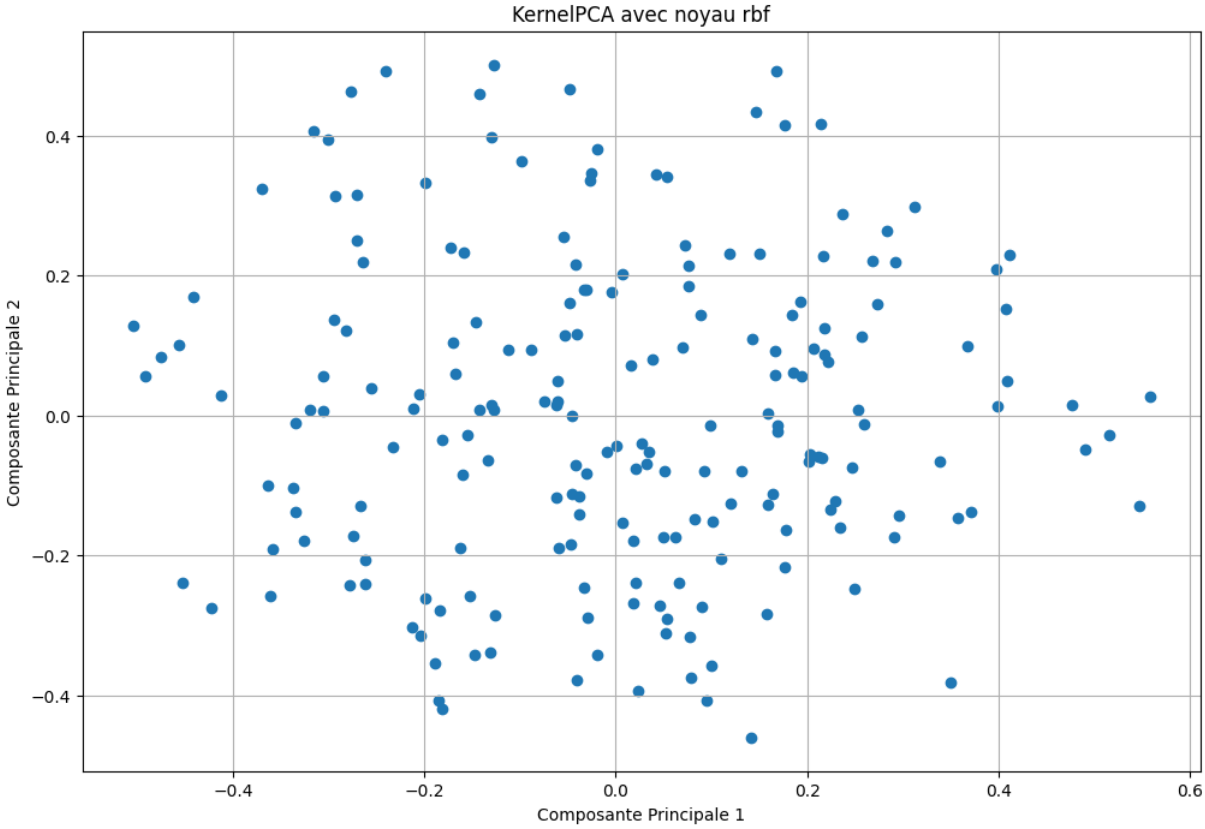
```
from sklearn.decomposition import KernelPCA

data = pd.read_csv('donnees_elevage_poulet.csv')

variables = ['Poids_poulet_g', 'Nourriture_consommee_g_jour', 'Temperature_e
data_subset = data[variables]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(data_subset)

noyaux = ['linear', 'rbf', 'poly']

for i in noyaux:
    print(f"{i} :")

    kpca = KernelPCA(n_components=2, kernel=i)
    X_kpca = kpca.fit_transform(X_scaled)

    principal_df = pd.DataFrame(X_kpca, columns=['Composante Principale 1',

    plt.figure(figsize=(12, 8))
    plt.scatter(principal_df['Composante Principale 1'], principal_df['Compo
    plt.xlabel('Composante Principale 1')
```

```
    plt.ylabel('Composante Principale 2')
    plt.title(f'KernelPCA avec noyau {i}')
    plt.grid(True)
    plt.show()
```

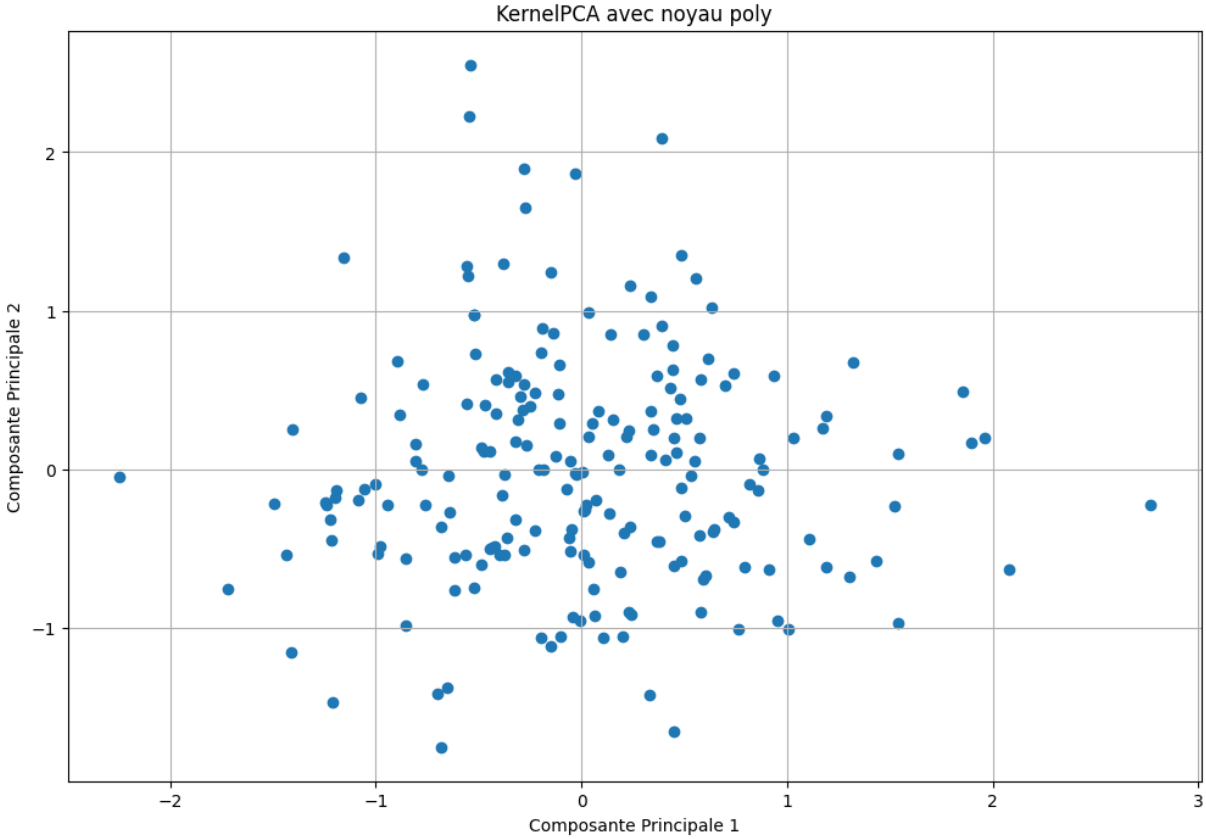
linear :



rbf :



`poly :`



L'ACP à noyaux est plus efficace lorsque les données ne sont pas linéairement séparables.

On peut comparer tous les graphiques obtenus et on remarque que l'ACP à noyaux avec un noyau polynomial semble être le graphique le plus performant car en effet les nuages de points sont beaucoup plus proches les unes des autres.

Exercice 6

In [93]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score

data = pd.read_csv('donnees_elevage_poulet.csv')

data['Survie'] = np.where(data['Taux_survie_%'] >= 90, 1, 0)

colonnes = ['Poids_poulet_g', 'Nourriture_consommee_g_jour', 'Temperature_er
X = data[colonnes]
y = data['Survie']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, rar

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
prediction = rf_model.predict(X_test)
precision = accuracy_score(y_test, prediction)
f1 = f1_score(y_test, prediction)

print("RandomForestClassifier")
print(f" Score de Précision : {precision}")
print(f" F1-score : {f1}")

importances = rf_model.feature_importances_

print("\nImportances des variables avec Random Forest:")
for i, j in zip(colonnes, importances):
    print(f"  {i}: {j}")
```

```
RandomForestClassifier
 Score de Précision : 0.7
 F1-score : 0.7

Importances des variables avec Random Forest:
  Poids_poulet_g: 0.16487941507814163
  Nourriture_consommee_g_jour: 0.13344370406663444
  Temperature_enclos_C: 0.12879070671036816
  Humidite_%: 0.15269873828091907
  Age_poulet_jours: 0.14201720606949025
  Gain_poids_jour_g: 0.12877594443167648
  Cout_elevage_FCFA: 0.14939428536276986
```

D'après le code, les variables les plus importantes sont le poids des poulets ainsi que le taux d'humidité. On peut supposer que pour le taux d'humidité, un taux trop élevé favorise la prolifération de bactéries et de moisissures et peut infecter

les poulets et donc diminuer leur chance de survie donc il est important de faire attention à cette variable. Pour le poids, on peut supposer que leur poids peut indiquer si le poulet est en bonne santé, s'il mange pas assez, cela pourrait être un signe de début de maladie ou une dégradation de sa santé.

Exercice 7

In [94]:
```
from sklearn.ensemble import AdaBoostRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score

data = pd.read_csv('donnees_elevage_poulet.csv')

colonnes = ['Poids_poulet_g', 'Nourriture_consommee_g_jour', 'Temperature_er
X = data[colonnes]
y = data['Gain_poids_jour_g']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, rar

ada = AdaBoostRegressor(n_estimators=50, random_state=42)
ada.fit(X_train, y_train)

gb = GradientBoostingRegressor(n_estimators=50, random_state=42)
gb.fit(X_train, y_train)

ada_y_pred = ada.predict(X_test)
gb_y_pred = gb.predict(X_test)

ada_mse = mean_squared_error(y_test, ada_y_pred)
ada_r2 = r2_score(y_test, ada_y_pred)

gb_mse = mean_squared_error(y_test, gb_y_pred)
gb_r2 = r2_score(y_test, gb_y_pred)

print("AdaBoostRegressor")
print(f"  MSE: {ada_mse}")
print(f"  R2: {ada_r2}")

print("GradientBoostingRegressor")
print(f"  MSE: {gb_mse}")
print(f"  R2: {gb_r2}")

# IQR
Q1 = data['Gain_poids_jour_g'].quantile(0.25)
Q3 = data['Gain_poids_jour_g'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = data[(data['Gain_poids_jour_g'] < lower_bound) | (data['Gain_poic

data_no = data[~data.index.isin(outliers.index)]

X_train_no, X_test_no, y_train_no, y_test_no = train_test_split(data_no[colc
```

```python
ada_no = AdaBoostRegressor(n_estimators=50, random_state=42)
ada_no.fit(X_train_no, y_train_no)
ada_no_predict = ada_no.predict(X_test)

gb_no = GradientBoostingRegressor(n_estimators=50, random_state=42)
gb_no.fit(X_train_no, y_train_no)
gb_no_predict = gb_no.predict(X_test)

ada_no_mse = mean_squared_error(y_test, ada_no_predict)
ada_no_r2 = r2_score(y_test, ada_no_predict)

gb_no_mse = mean_squared_error(y_test, gb_no_predict)
gb_no_r2 = r2_score(y_test, gb_no_predict)

print("AdaBoostRegressor sans outliers")
print(f"  MSE: {ada_no_mse}")
print(f"  R2: {ada_no_r2}")

print("GradientBoostingRegressor sans outliers")
print(f"  MSE: {gb_no_mse}")
print(f"  R2: {gb_no_r2}")
```

```
AdaBoostRegressor
  MSE: 27.83490175406657
  R2: -0.5806955001876322
GradientBoostingRegressor
  MSE: 26.944375632019597
  R2: -0.5301240756373893
AdaBoostRegressor sans outliers
  MSE: 27.83490175406657
  R2: -0.5806955001876322
GradientBoostingRegressor sans outliers
  MSE: 26.944375632019597
  R2: -0.5301240756373893
AdaBoostRegressor sans outliers
  MSE: 27.83490175406657
  R2: -0.5806955001876322
GradientBoostingRegressor sans outliers
  MSE: 26.944375632019597
  R2: -0.5301240756373893
```

On peut remarquer que les modèles choisis ne sont pas performants car leur score R2 est négatif. On peut supposer que les modèles ne sont pas adaptés pour la prédiction du gain de poids.

Après de multiples tentatives pour voir des différences sans et avec outliers, je n'arrive pas à avoir des résultats qui sont différents. Par conséquent je ne peux donc pas répondre à la dernière question.