TP1

March 28, 2025

1 Analyse de Données et Méthodes d'Ensemble

[15]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore, shapiro
from sklearn.decomposition import KernelPCA, PCA
from sklearn.ensemble import RandomForestClassifier, AdaBoostRegressor,␣
 ↪GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, mean_squared_error,␣
 ↪r2_score
```

1.1 Partie 1 : Analyse exploratoire des données

[4]:
```
data = pd.DataFrame({'poids':np.random.normal(2.5,0.5,100),
                     'nourriture':np.random.normal(1.2,0.3,100),
                     'température':np.random.normal(25,2,100)})
data
```

[4]:

	poids	nourriture	température
0	2.397766	1.594355	27.614211
1	1.860867	0.991491	25.679102
2	2.502766	1.909469	25.712154
3	2.943343	1.309094	28.789658
4	2.463610	1.239253	23.339024
..	…	…	…
95	1.771609	1.597872	24.420769
96	2.601971	1.389973	29.754458
97	1.946321	1.090645	27.515166
98	2.396936	1.287460	23.295369
99	2.020381	1.496021	24.028675

[100 rows x 3 columns]

Le dataset comporte 100 données de 3 variables (poids, nourriture consommée par jour et température) quantitatives continues. Ces données ont été générées aléatoirement suivant une distribution

normale.

1.1.1 Exercice 1 : Statistiques descriptives

Calcul de la moyenne, médiane, écart-type, variance et les quartiles pour les variables poids, nourriture et température.

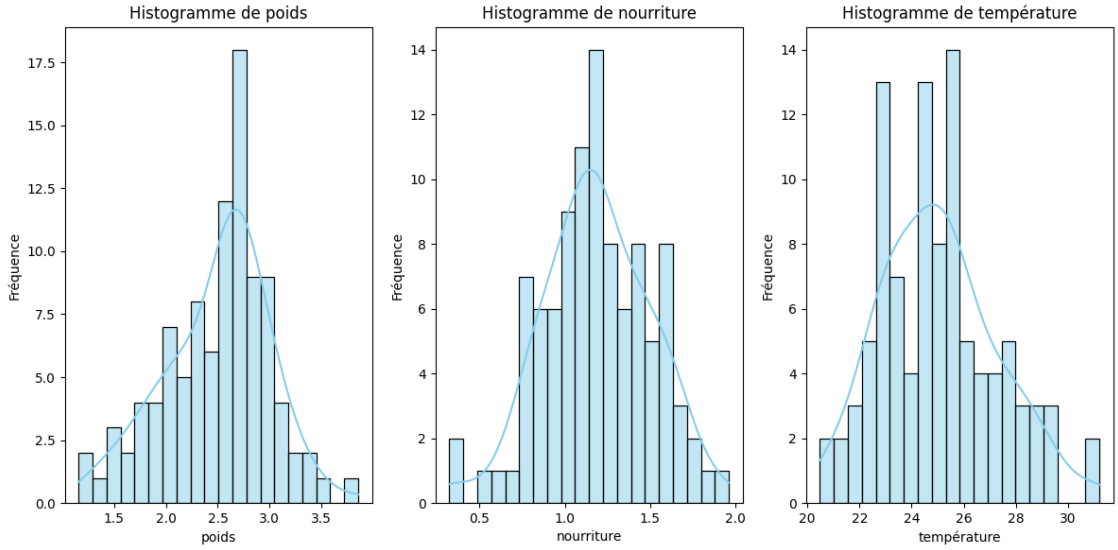
[89]:
```
v=['poids','nourriture','température']
# tableau des stats
stats = pd.DataFrame({
    "Moyenne": data[v].mean(),
    "Médiane": data[v].median(),
    "Écart-type": data[v].std(),
    "Variance": data[v].var(),
    "Q1 (25%)": data[v].quantile(0.25),
    "Q3 (75%)": data[v].quantile(0.75)}).T
stats
```

[89]:

	poids	nourriture	température
Moyenne	2.493610	1.186442	25.022620
Médiane	2.604481	1.185354	24.799534
Écart-type	0.508445	0.312631	2.235462
Variance	0.258516	0.097738	4.997289
Q1 (25%)	2.161524	0.979927	23.320431
Q3 (75%)	2.799703	1.414287	26.265196

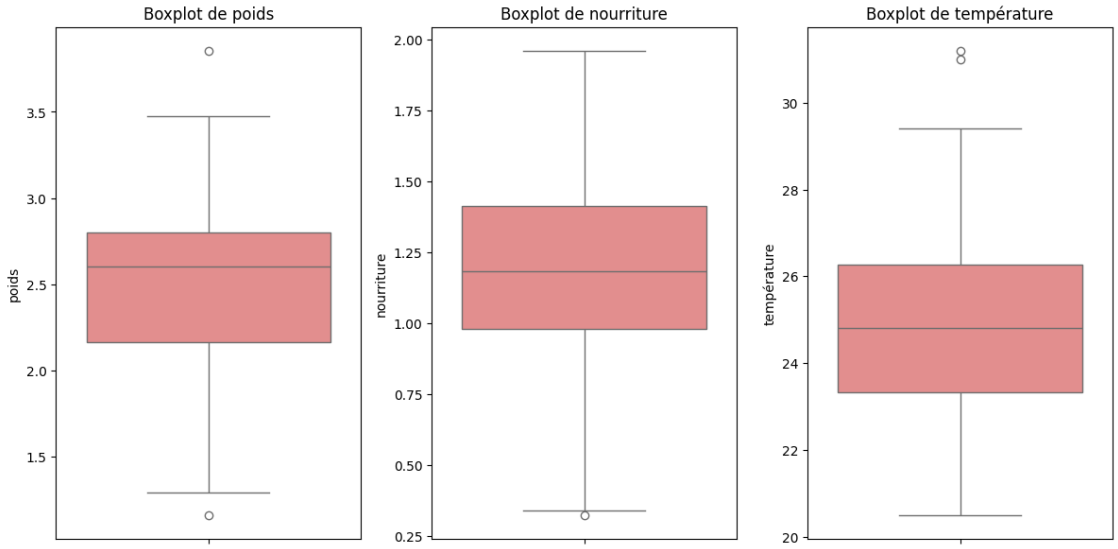
Interprétation Pour les 3 variables, la moyenne est très proche de la médiane. Cela signifie que que leurs distributions est équilibrée.
La valeur de l'écart type est petite pour le poids et la nourriture, ce qui indique une certaine stabilité. Càd que les poulets se nourrissent de la même façon et leurs poids n'est pas très différent. Les valeurs faibles de la variance indiquent que les données sont peu dispersées.
Pour la température, les valeurs de l'écart type est un peu élevée, ce qui indique que les conditions d'élevage peuvent être différentes.

[90]:
```
# Création des histogrammes
plt.figure(figsize=(12, 6))
for i, var in enumerate(v, 1):
    plt.subplot(1, 3, i)
    sns.histplot(data[var], bins=20, kde=True, color='skyblue')
    plt.title(f'Histogramme de {var}')
    plt.xlabel(var)
    plt.ylabel('Fréquence')
plt.tight_layout()
plt.show()
```



Dans les histogrammes, on observe la présence de quelques barres isolées, ce qui peut indiquer la présence de valeurs aberrantes.
Les histogrammes du poids et nourriture sont en forme de cloche, signifie que les données sont équilibrées (elles suivent une distribution normale).
L'histogramme de la température est faiblement asymétrique e, cela peut signifier la présence de quelques valeurs extremes(ayant des valeurs plus faibles que les autres qui sont assez grandes).

[91]:
```
# Création des boxplots
plt.figure(figsize=(12, 6))
for i, var in enumerate(v, 1):
    plt.subplot(1, 3, i)
    sns.boxplot(y=data[var], color='lightcoral')
    plt.title(f'Boxplot de {var}')
plt.tight_layout()
plt.show()
```



On remarque la présence de points en dehors des moustaches, ce sont les valeurs aberrantes. Les boites ne sont pas très étendues, on peut dire que les données ne sont pas très dispersées.

D'après l'analyse statistique descriptive, on constate que les données du dataset suivent une distribution normale, elles sont relativement équilibrées et peu dispersées, avec la présence de quelques valeurs aberrantes (les poulets sont nourris équitablement mais sont élevés dans des conditions de températures un peu différentes les uns des autres).
Cela dit que les données doivent être nettoyées pour se débarrasser des valeurs aberrantes et normalisées pour mettre toutes les variables sur la même échelle (unités différentes)

1.1.2 Exercice 2 : Détection des outliers

[92]:
```
# avec IQR
Q1 = data[['poids', 'nourriture', 'température']].quantile(0.25)
Q3 = data[['poids', 'nourriture', 'température']].quantile(0.75)
IQR = Q3 - Q1
outliers_iqr = ((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR)))

print("Outliers IQR:\n", outliers_iqr.sum())
```

```
Outliers IQR:
 poids          2
nourriture      1
température     2
dtype: int64
```

Dans la méthode des IQR, on prend en compte le 1er quartile (médiane des permiers 50% des donnéess) et le 3ème quartile (médiane des 50% dernières des données), l'IQR étant la différence entre ces deux. S'il y a des données qui se trouvent en dehors de la plage, elles sont considérées comme des outliers.
Une seule donnée pour la nourriture et 2 pour le poids et la température.

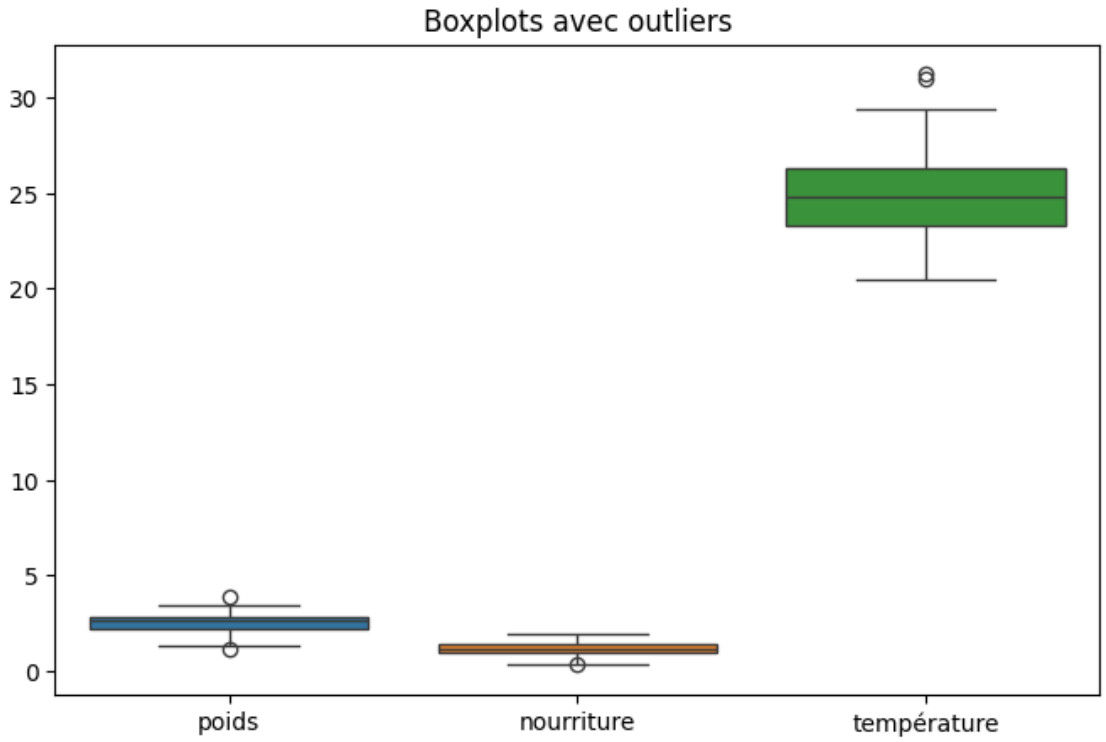
[93]:
```
# avec Z-score
data['z_score_poids'] = zscore(data['poids'])
data['z_score_nourriture'] = zscore(data['nourriture'])
data['z_score_temperature'] = zscore(data['température'])
outliers_z = ((data[['z_score_poids', 'z_score_nourriture',␣
 ↪'z_score_temperature']] > 3) | (data[['z_score_poids', 'z_score_nourriture',␣
 ↪'z_score_temperature']] < -3) )

print("Outliers Z-score:\n", outliers_z.sum())
```

```
Outliers Z-score:
 z_score_poids          0
z_score_nourriture      0
z_score_temperature     0
dtype: int64
```

Le Z-score mesure la relation d'une donnée avec la moyenne et l'écart type du dataset. Les valeurs seuils étant -3 et 3 (selon la distance à la moyenne), les résultats indiquent que selon cette méthode, il n'existe aucune valeur aberrante.

Les résultats données par Z-score indiquent qu'aucune valeur n'est aberrante sauf pour la nourriture. Tandis que la méthode de l'IQR repère l'existence de certains outliers. Comme Z-score utilise la moyenne et l'écart-type pour identifier les outliers et que les données de notre dataset suivent une distribution normale, il peut être influencé par cette distribution, alors que l'IQR est moins influencé par la distribution des données.

[94]:
```
# Visualisation
plt.figure(figsize=(8, 5))
sns.boxplot(data=data[['poids', 'nourriture', 'température']])
plt.title("Boxplots avec outliers")
plt.show()
```



Comme les données suivent une distribution normale, selon la règle empirique (qui désigne le pourcentage de données qui se situent à l'intérieur d'un, de deux ou de trois écarts-types de la moyenne). Les valeurs aberrantes ne sont pas forcément des erreurs. Donc, on les garde.

1.1.3 Exercice 3 : Tests paramétriques

[98]:
```
# 5. Test de Shapiro-Wilk
for c in v:
    stat, p_value = shapiro(data[c])
    print(f"Test de Shapiro-Wilk pour {c} : p-value = {p_value}")
```

```
Test de Shapiro-Wilk pour poids : p-value = 0.09608639031648636
Test de Shapiro-Wilk pour nourriture : p-value = 0.7985832691192627
Test de Shapiro-Wilk pour température : p-value = 0.2125396728515625
```

Le test shapiro utilise le rapport de deux estimations de la variance. Dans le cas d'une variable normale, ces deux estimations coïncident et le rapport est voisin de 1, alors que si la variable n'est pas normale le rapport est plus petit que 1.
La valeur p est utilisée pour rejeter ou conserver (ne pas rejeter) l'hypothèse nulle dans un test d'hypothèse. Si la valeur p calculée est inférieure au seuil de signification, qui est dans la plupart des cas de 5 %, l'hypothèse nulle est rejetée, sinon elle est maintenue.
Dans ce cas, l'hypothèse dit que toutes les données sont normalement distribuées. les valeurs p sont supérieures à 0.05, l'hypothèse est donc acceptée et les données suivent une loi normale.

[ ]:

1.2 Partie 2 : Réduction de dimensionnalité

1.2.1 Exercice 4 : Analyse en Composantes Principales (ACP)

[3]:
```
data = pd.read_csv("donnees_elevage_poulet.csv")
data
```

[3]:

	Poids_poulet_g	Nourriture_consommee_g_jour	Temperature_enclos_C	Humidite_%	Age_poulet_jours	Gain_poids_jour_g	Taux_survie_%	Cout_elevage_FCFA
0	3974	52	27.6	79.3	24	12.0	81.1	2682
1	1660	152	31.7	62.5	42	12.2	89.1	6626
2	2094	186	30.1	64.8	29	18.8	90.4	8424
3	1930	111	29.2	87.0	63	13.8	92.9	1933
4	1895	100	26.1	78.2	21	5.5	93.0	4598
..	…	…	…	…	…	…	…	…
195	2106	165	27.4	87.9	44	7.9	80.6	2390
196	3480	143	30.1	85.8	109	6.7	94.6	2742
197	3475	96	31.3	72.9	94	11.3	98.2	2925
198	1772	148	31.2	82.5	57	16.9	97.1	1619
199	2568	104	30.5	82.6	21	16.2	99.3	2274

```
[200 rows x 8 columns]
```

[108]:
```
# ACP manuelle
X = data[['Poids_poulet_g', 'Nourriture_consommee_g_jour',␣
 ↪'Temperature_enclos_C']].values
X = (X - X.mean(axis=0)) / X.std(axis=0)

# Matrice de covariance
mat_cov = np.cov(X.T)

# valeurs propres et vecteurs propres
val_prop, vect_prop = np.linalg.eig(mat_cov)
print("Valeurs propres :\n", val_prop)
print("Vecteurs propres :\n", vect_prop)
```

```
Valeurs propres :
 [1.21640154 0.99098099 0.80769285]
Vecteurs propres :
 [[ 0.32703009 -0.91550074  0.2343282 ]
 [-0.69352582 -0.06406901  0.71757725]
 [ 0.64192933  0.39718201  0.65587589]]
```

[115]:
```
valeurs_propres = [1.2164, 0.9910, 0.8077]
variance_expliquee = val_prop / sum(val_prop)
print(variance_expliquee)
```

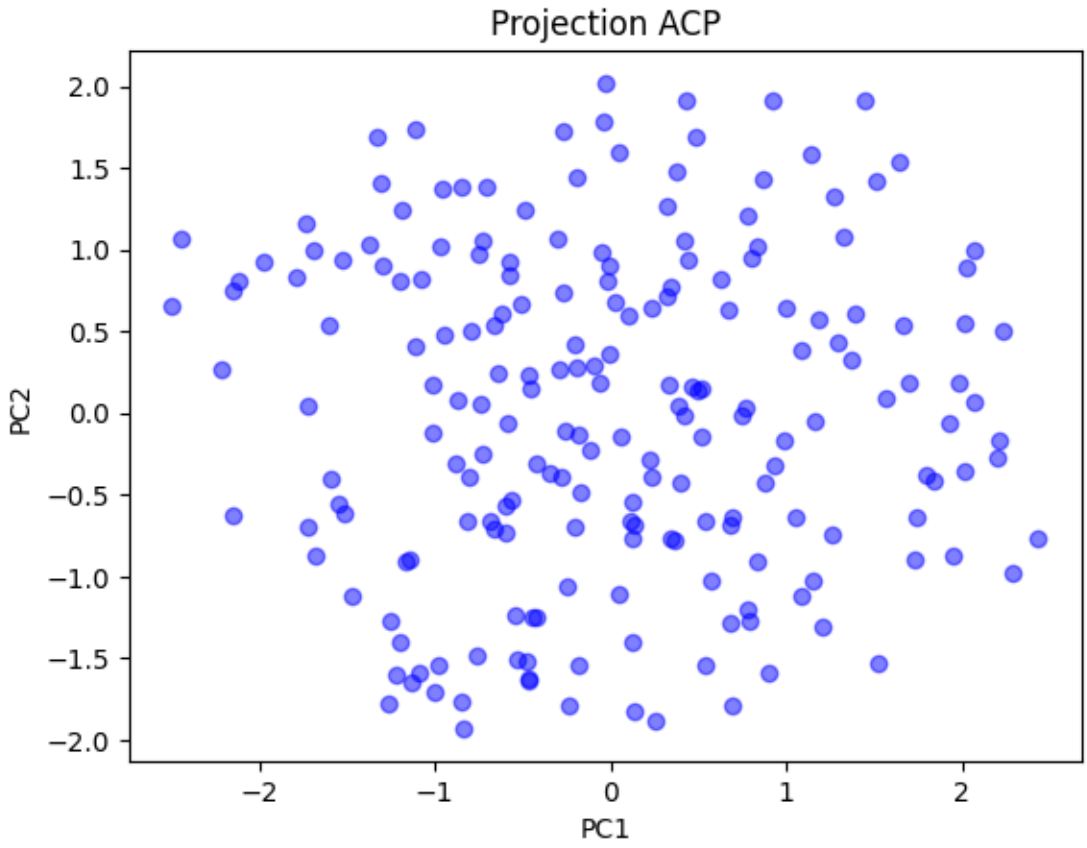
```
[0.40343984 0.32867536 0.26788479]
```

Les valeurs propres représentent la variance expliquée par chaque composante principale, plus elle est grande, plus la composante est importante.
La PC1 explique 40% de la variance, PC2 explique 33% de la variance et PC3 explique 27% de la variance. On peut donc projter les données dans les 2 premières dimensions sans perte d'informations.

Pour les vecteurs propres : Chaque colonne correspond à un vecteur propre et indique la contribution de chaque variable dans la nouvelle composante. PC1 : la température et poids sont fortement positifs, nourriture négative : PC1 capture une relation entre poids et température vs nourriture.
PC2 : Poids domine fortement en négatif, température en positif : PC2 capture principalement les variations du poids.
PC3 : Nourriture et température sont les plus importants ici.

[109]:
```
# Projection sur les 2 premières composantes
X_pca = X @ vect_prop[:, :2]

# Visualisation
plt.scatter(X_pca[:, 0], X_pca[:, 1], c='blue', alpha=0.5)
plt.xlabel("PC1")
plt.ylabel("PC2")
```

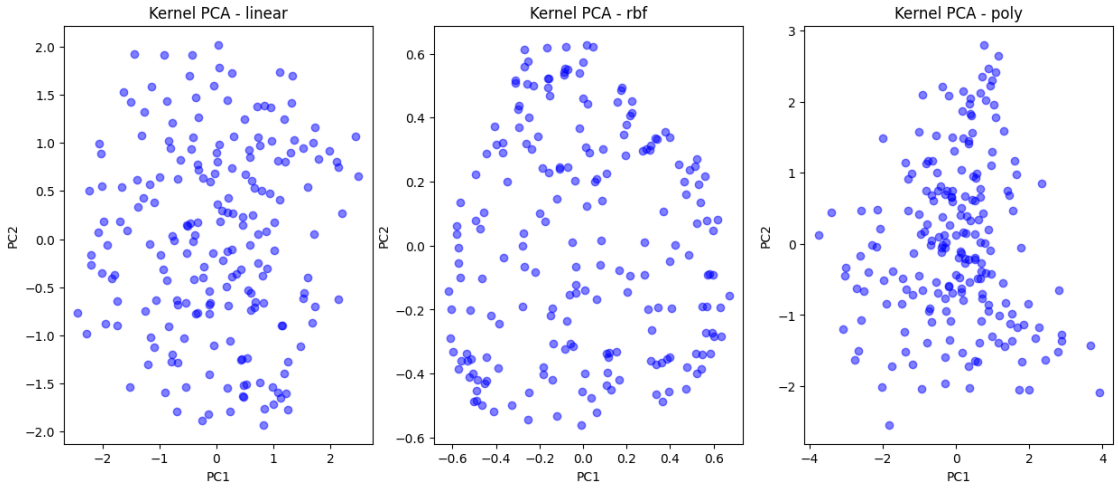
```
plt.title("Projection ACP")
plt.show()
```



Les points sont répartis de manière homogène, pas de distinction visible. Tous les points se trouvent sur une plage entre -2 et 2, ce qui indique que les valeurs ne sont pas trop dispersés. Les 2 composantes représentent bien la variance.
Les 2 premières composantes expliquent plus de 70% de la variance des données, donc on les garde.

1.2.2 Exercice 5 : ACP à Noyau

[126]:
```
kernels = ["linear", "rbf", "poly"]
fig, axes = plt.subplots(1, 3, figsize=(15, 6))

for ax, kernel in zip(axes, kernels):
    X_kpca = KernelPCA(n_components=2, kernel=kernel).fit_transform(X)
    ax.scatter(X_kpca[:, 0], X_kpca[:, 1], c='blue', alpha=0.5)
    ax.set_title(f"Kernel PCA - {kernel}")
    ax.set_xlabel("PC1")
    ax.set_ylabel("PC2")
plt.show()
```



Noyau linéaire : identique à la projection précédente.
Noyau RBF : On remarque une séparation entre les données, cela indique que les données ont des relations non linéaires et ce noyau arrive à les capturer.
L'ACP à noyau est utilise dans le cas de relations non linéaires entre les variables et quand l'ACP classique ne permet pas de bien séparer les données.

1.3 Partie 3 : Méthodes d'ensemble

1.3.1 Exercice 6 : Bagging

[7]:
```
X = data.drop(columns=['Taux_survie_%'])
y = data['Taux_survie_%'] >= 90

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_standardized)


# Division des données
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2,␣
 ↪random_state=42)

# Modèle
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Prédictions
y_pred = rf.predict(X_test)
```

```
# Evaluation
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy : {accuracy}")
print(f"F1-score : {f1}")
```

```
Accuracy : 0.675
F1-score : 0.6486486486486487
```

[6]:
```
count_true = y.value_counts()[True]
count_false = y.value_counts()[False]
print(f"True: {count_true}, False: {count_false}")
```

```
True: 102, False: 98
```

Le modèle a correctement prédit 67% des cas dans l'ensemble de test.
F1-score de 64% montre que le modèle est capable de capturer les cas positifs tout en limitant les erreurs

[11]:
```
# variables importantes
imp = rf.feature_importances_
for feature, i in zip(X.columns, imp):
    print(f"{feature}: {i:.2f}")
```

```
Poids_poulet_g: 0.50
Nourriture_consommee_g_jour: 0.50
```

Un poids plus élevé indique une bonne santé et une meilleure alimentation favorise la survie.

1.3.2 Exercice 7 : Boosting

[ ]:
```

```