Notebook_analyse_Modeles_application_fonctionnelle

August 25, 2025

Evaluation des performances_choix des 4 algorithmes

Objectif

Prédire si un billet est genuine (is_genuine)

Comparer : Régression Logistique, KNN, Random Forest, K-Means

Évaluer avec : matrice de confusion, accuracy, f1-score…

[3]:
```
# Importations

import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪accuracy_score

from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA

import seaborn as sns
import matplotlib.pyplot as plt
```

[4]:
```
# charger les données
billets = pd.read_csv('billets_final.csv', sep=',', encoding='latin_1')
billets.head()
```

[4]:

	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	\
0	True	171.81	104.86	104.95	4.52	2.89	
1	True	171.46	103.36	103.66	3.77	2.99	

```
2        True    172.69       104.48       103.50       4.40       2.94
3        True    171.36       103.91       103.94       3.62       3.01
4        True    171.73       104.28       103.46       4.04       3.48
```

```
   length
0  112.83
1  113.09
2  113.16
3  113.51
4  112.54
```

[5]:
```
#  Définir X et y (séparer X et y)
X = billets.drop(columns='is_genuine')  # ou 'target', selon votre fichier
y = billets['is_genuine']
```

[6]:
```
# Standardisation des données (normalisation)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

3. Séparation en train/test

[8]:
```
# Split des données (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,␣
↪random_state=42, stratify=y)
```

4. REGRESSION LOGISTIQUE

[10]:
```
#entrainer un modeèle de rég log sur l'ens. d'entraînement
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

[10]:
```
LogisticRegression()
```

[11]:
```
# évaluation du modèle sur l'ensemble de test
y_pred_log = logreg.predict(X_test)

print("Régression Logistique :")
print(confusion_matrix(y_test, y_pred_log))
print(classification_report(y_test, y_pred_log))
```

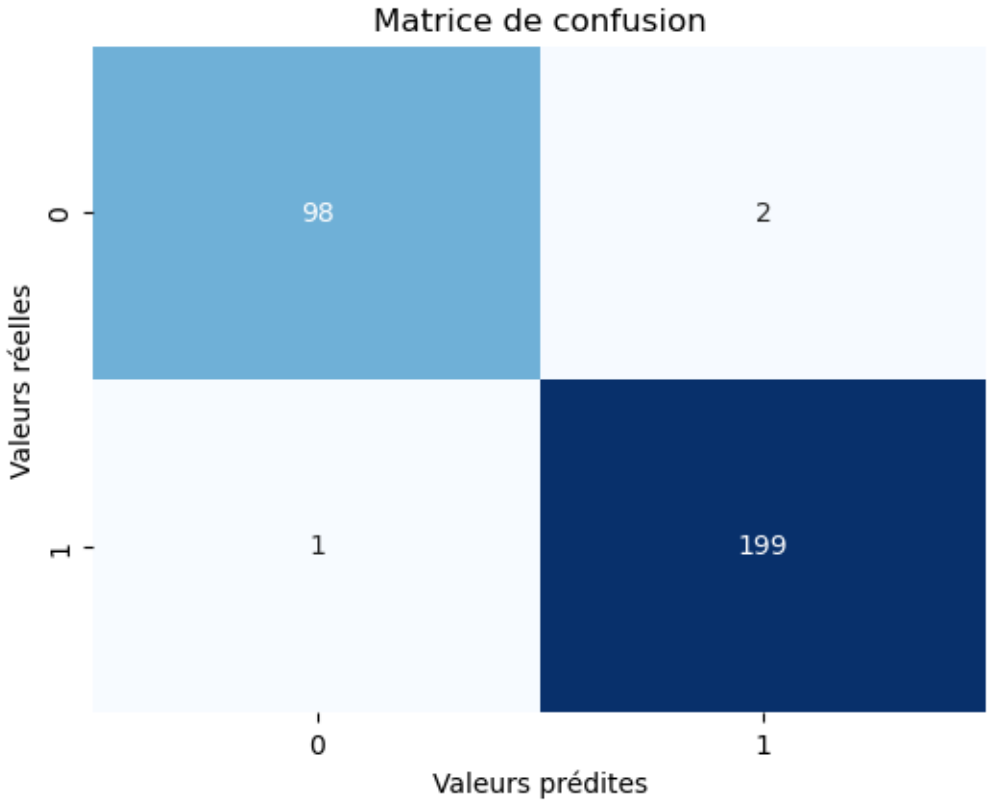
```
Régression Logistique :
[[ 98   2]
 [  1 199]]
              precision    recall  f1-score   support

       False       0.99      0.98      0.98       100
        True       0.99      0.99      0.99       200

    accuracy                           0.99       300
```

```
   macro avg       0.99      0.99      0.99       300
weighted avg       0.99      0.99      0.99       300
```

[12]:
```
# Matrice de confusion
conf_mat = confusion_matrix(y_test, y_pred_log)
print("Matrice de confusion :")
print(conf_mat)
# Heatmap pour la matrice de confusion
fig, ax = plt.subplots(figsize=(6, 4.5))
sns.heatmap(conf_mat, annot=True, cmap='Blues', fmt='g', ax=ax, cbar=False)
ax.set_xlabel('Valeurs prédites')
ax.set_ylabel('Valeurs réelles')
ax.set_title('Matrice de confusion')
```

```
Matrice de confusion :
[[ 98   2]
 [  1 199]]
```

[12]:
```
Text(0.5, 1.0, 'Matrice de confusion')
```



[13]:
```python
# Précision
precision = precision_score(y_test, y_pred_log)
print("Précision :", precision)
# Rappel
recall = recall_score(y_test, y_pred_log)
print("Rappel :", recall)
# Score F1
f1 = f1_score(y_test, y_pred_log)
print("Score F1 :", f1)
# Autres métriques (par exemple, l'exactitude)
accuracy = accuracy_score(y_test, y_pred_log)
print("Exactitude :", accuracy)
```

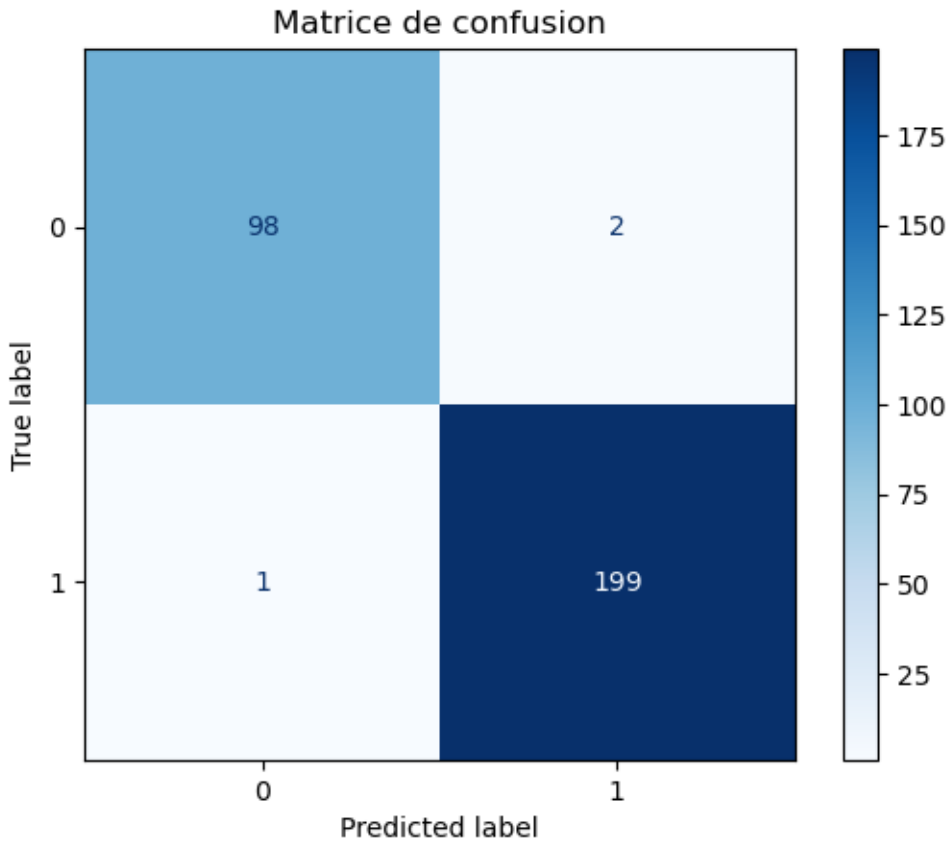
```
Précision : 0.9900497512437811
Rappel : 0.995
Score F1 : 0.9925187032418953
Exactitude : 0.99
```

```
Précision : 0.990
    Cela signifie que 99 % des prédictions positives (ex: prédits comme genuine) étaient corre
Rappel : 0.995
    Cela veut dire que 99,5 % des vrais positifs (les vrais genuine) ont bien été identifiés pa
Score F1 : 0.993
    C'est une moyenne entre la précision et le rappel. Il montre que le modèle est équilibré :
Exactitude : 0.990
    99 % des prédictions totales (positives et négatives) sont correctes. C'est le score globa
```

Le modèle est très performant : il fait très peu d'erreurs, il repère quasiment tous les vrais cas, et ne donne presque jamais de faux positifs.

[15]:
```python
# Matrice de confusion
cm = confusion_matrix(y_test, y_pred_log)
print("Matrice de confusion :")
print(conf_mat)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap='Blues', values_format='d')

plt.title("Matrice de confusion")
plt.show()
```

```
Matrice de confusion :
[[ 98   2]
 [  1 199]]
```



Interprétation : Élément Valeur Explication Vrais négatifs 98 Le modèle a bien prédit non-genuine (0) pour 98 vrais cas. Faux positifs 2. 2 fois, le modèle a prédit genuine alors que ce n'était pas le cas. Faux négatifs 1. 1 fois, le modèle a manqué un cas genuine (prédit non-genuine). Vrais positifs 199. 199 fois, le modèle a correctement détecté un genuine. Conclusion :

Ton modèle est très performant :

`Il détecte bien les cas genuine (très peu de faux négatifs).`

`Il se trompe très rarement en identifiant à tort un faux genuine (peu de faux positifs).`

`Les erreurs sont très faibles (3 erreurs sur 300 cas testés).`

`Prédit : Non-genuine (0)  Prédit : Genuine (1)`

Réel : Non-genuine (0) 98 2 Réel : Genuine (1) 1 199

explication claire et pas à pas de la **matrice de confusion**, avec l'exemple précis. la matrice de confusion :

`Prédit : Non-genuine (0)  Prédit : Genuine (1)`

Réel : Non-genuine (0) 98 2 Réel : Genuine (1) 1 199 "' Étape 1 : Comprendre les 4 cases

	Prédit : 0 (Non-genuine)	**Prédit : 1 (Genuine)**
Réel : 0 (Non-genuine)	**98** (Bien classés)	**2** (erreurs)
Réel : 1 (Genuine)	**1** (erreur)	**199** (Bien classés)

- ** Vrais positifs (VP) = 199** → Le modèle a bien détecté 199 vrais *genuine*.
- **Vrais négatifs (VN) = 98** → Le modèle a bien détecté 98 vrais *non-genuine*.
- **Faux positifs (FP) = 2** → Le modèle a prédit *genuine*, mais en réalité c'était *non-genuine* (faux).
- **Faux négatifs (FN) = 1** → Le modèle a prédit *non-genuine*, mais c'était un *genuine* (manqué).

Étape 2 : Que signifient ces chiffres ?

- **Quand c'est "vrai"**, le modèle a **bien prédit**.
- **Quand c'est "faux"**, il a **fait une erreur** :
  - **Faux positif** : il pense que c'est vrai, mais c'est faux.
  - **Faux négatif** : il pense que c'est faux, mais c'est vrai.

#Résumé visuel simple : | | Prédit correct ? | Nombre | Ce que ça veut dire | | ——- | ———- | —— | ———————————— | | **98** | Oui | 98 | Il a bien dit *non-genuine*, et c'était vrai | | **199** | Oui | 199 | Il a bien dit *genuine*, et c'était vrai | | **2** | Non | 2 | Il a dit *genuine*, mais c'était faux | | **1** | Non | 1 | Il a dit *non-genuine*, mais c'était vrai |

#En résumé : * le modèle fait **très peu d'erreurs**. * Il **repère presque tous** les vrais *genuine* (seulement 1 oublié). * Il **ne se trompe presque jamais** en annonçant un *genuine* (seulement 2 faux).

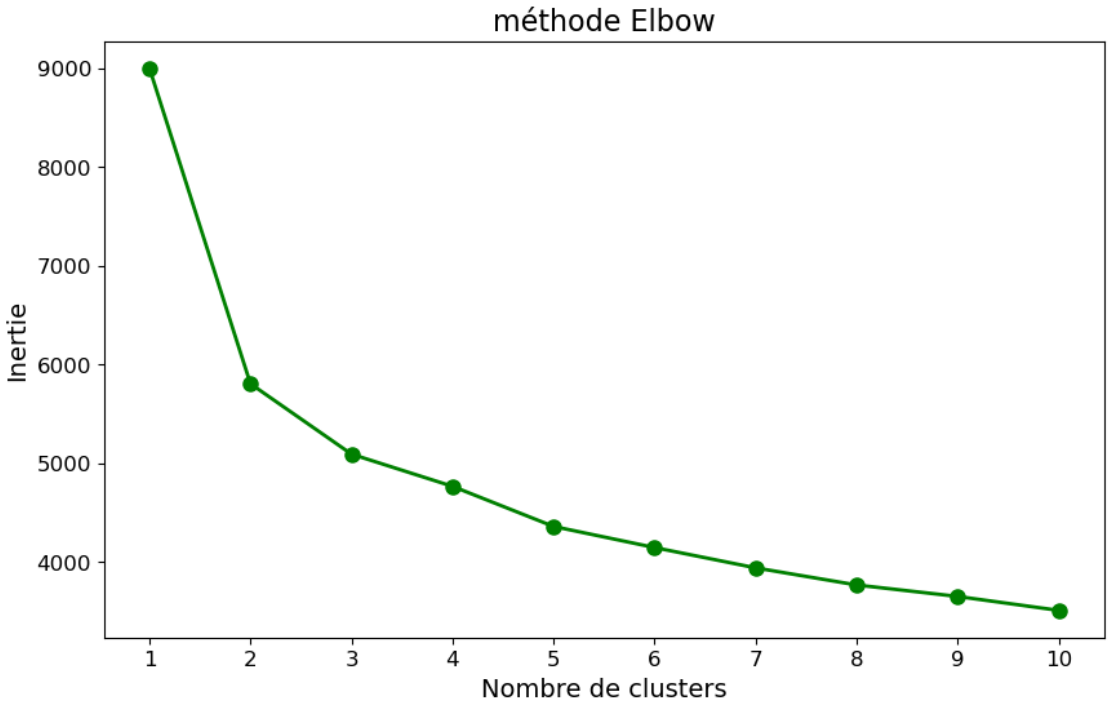
5. K-means (algorithme non supervisé)

[19]:
```
# Calculer les inerties pour différents nombres de clusters (de 1 à 10)
inertias = []
for k in range(1, 11):
    kmeans_temp = KMeans(n_clusters=k, random_state=42)
    kmeans_temp.fit(X_scaled)
    inertias.append(kmeans_temp.inertia_)
```

[20]:
```
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), inertias, 'go-', linewidth=2, markersize=8)  # 'g' = vert
plt.xlabel('Nombre de clusters', fontsize=14)
plt.ylabel('Inertie', fontsize=14)
plt.title("méthode Elbow", fontsize=16)
plt.xticks(range(1, 11), fontsize=12)
plt.yticks(fontsize=12)
```

[20]:
```
(array([ 3000.,  4000.,  5000.,  6000.,  7000.,  8000.,  9000., 10000.]),
 [Text(0, 3000.0, '3000'),
  Text(0, 4000.0, '4000'),
```

```
Text(0, 5000.0, '5000'),
Text(0, 6000.0, '6000'),
Text(0, 7000.0, '7000'),
Text(0, 8000.0, '8000'),
Text(0, 9000.0, '9000'),
Text(0, 10000.0, '10000')])
```



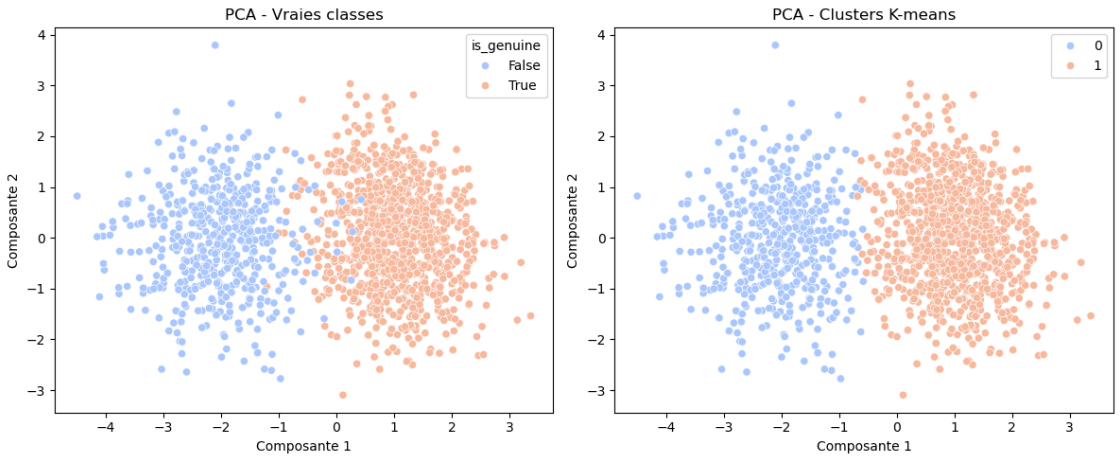
Satisfaction de 2 clusters grâce à la “cassure”

[22]:
```python
# Appliquer K-means avec 2 clusters (faux / vrai)
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_scaled)
labels = kmeans.labels_

# Ajustement possible : les labels peuvent être inversés
from sklearn.metrics import accuracy_score
acc1 = accuracy_score(y, labels)
acc2 = accuracy_score(y, 1 - labels)
kmeans_labels = labels if acc1 > acc2 else 1 - labels

print("K-means (non supervisé):")
print(confusion_matrix(y, kmeans_labels))
print(classification_report(y, kmeans_labels))
```

```
K-means (non supervisé):
```

```
[[486  14]
 [ 10 990]]
              precision    recall  f1-score   support

       False       0.98      0.97      0.98       500
        True       0.99      0.99      0.99      1000

    accuracy                           0.98      1500
   macro avg       0.98      0.98      0.98      1500
weighted avg       0.98      0.98      0.98      1500
```

[23]:
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(12,5))

# Vraies classes
plt.subplot(1, 2, 1)
sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1], hue=y, palette='coolwarm')
plt.title("PCA - Vraies classes")
plt.xlabel("Composante 1")
plt.ylabel("Composante 2")

# Clusters K-means
plt.subplot(1, 2, 2)
sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1], hue=kmeans_labels,␣
 ↪palette='coolwarm')
plt.title("PCA - Clusters K-means")
plt.xlabel("Composante 1")
plt.ylabel("Composante 2")

plt.tight_layout()
plt.show()
```



5. KNN

[25]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

print("K-Nearest Neighbors (KNN) :")
print(confusion_matrix(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
```

```
K-Nearest Neighbors (KNN) :
[[ 97   3]
 [  2 198]]
              precision    recall  f1-score   support

       False       0.98      0.97      0.97       100
        True       0.99      0.99      0.99       200

    accuracy                           0.98       300
   macro avg       0.98      0.98      0.98       300
weighted avg       0.98      0.98      0.98       300
```

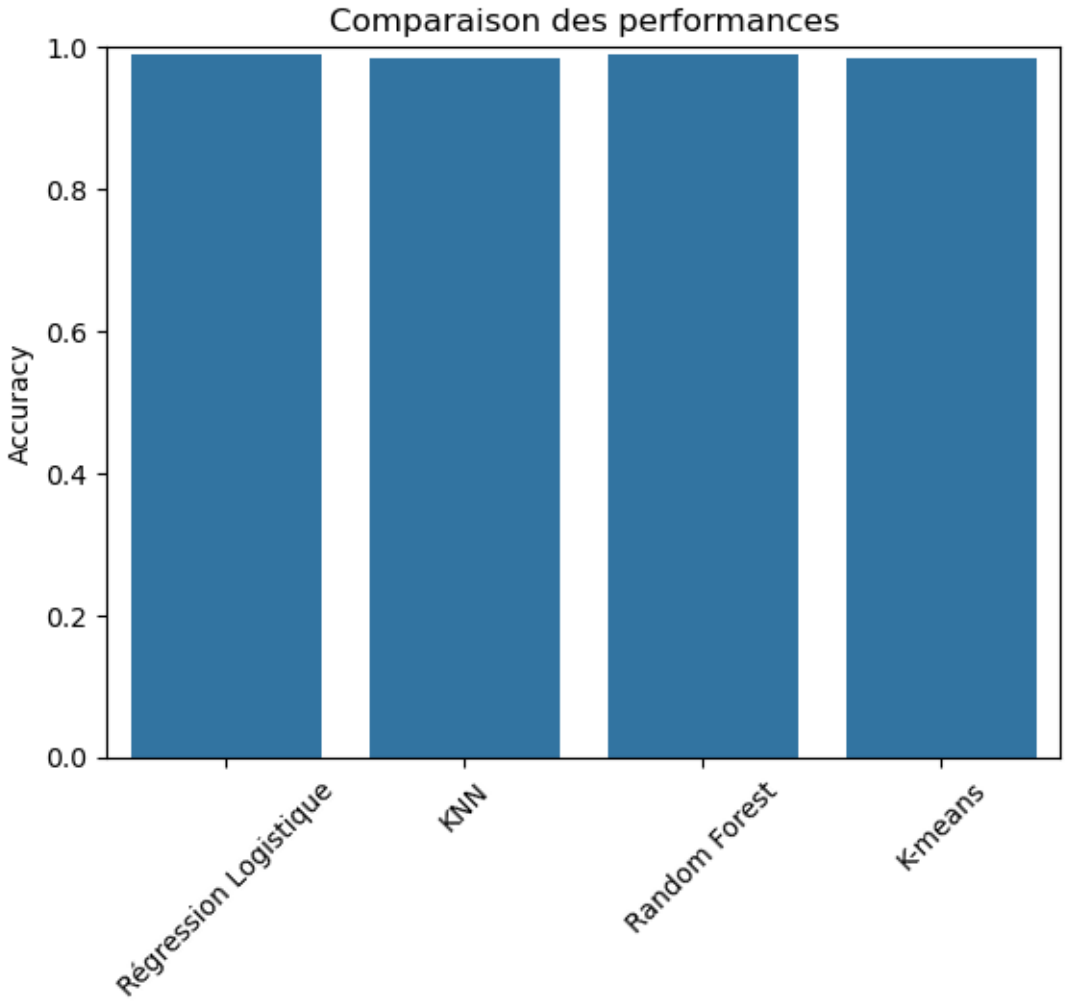
6. Random Forest

[27]:
```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

print("Random Forest :")
print(confusion_matrix(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest :
[[ 98   2]
 [  1 199]]
              precision    recall  f1-score   support

       False       0.99      0.98      0.98       100
        True       0.99      0.99      0.99       200

    accuracy                           0.99       300
   macro avg       0.99      0.99      0.99       300
weighted avg       0.99      0.99      0.99       300
```

Comparaison des performances

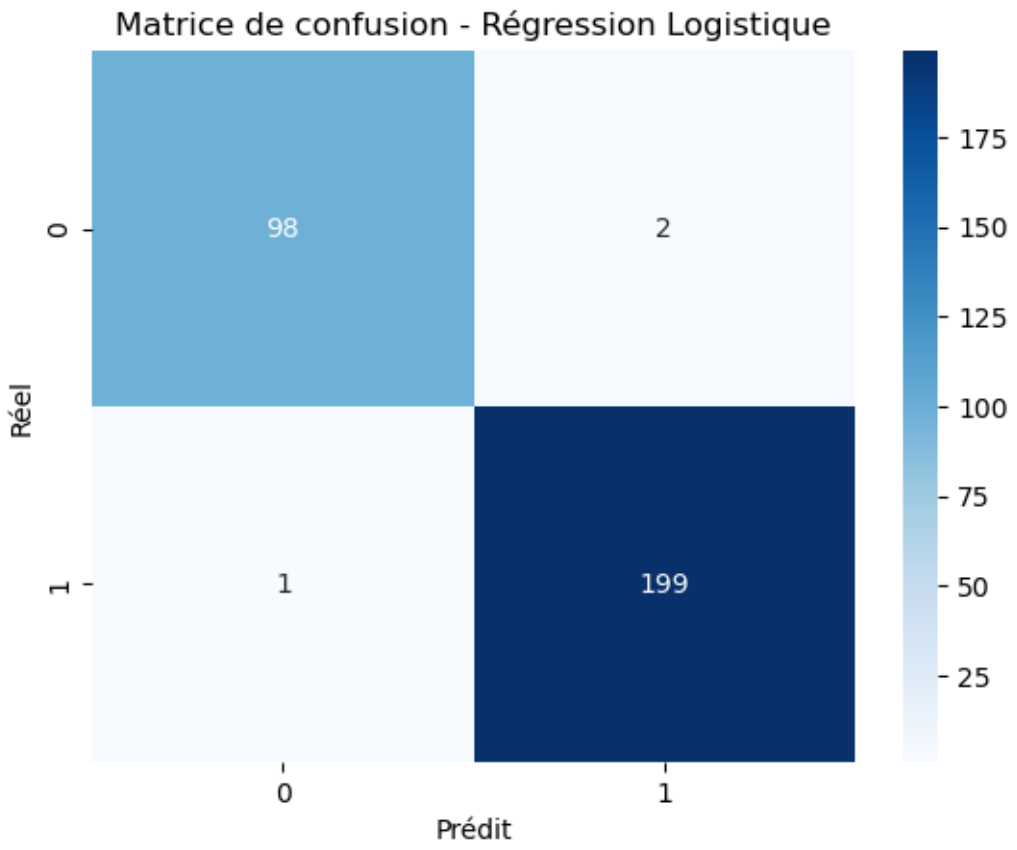
[29]:
```
from sklearn.metrics import accuracy_score

results = {
    'Régression Logistique': accuracy_score(y_test, y_pred_log),
    'KNN': accuracy_score(y_test, y_pred_knn),
    'Random Forest': accuracy_score(y_test, y_pred_rf),
    'K-means': max(acc1, acc2)
}

sns.barplot(x=list(results.keys()), y=list(results.values()))
plt.ylabel("Accuracy")
plt.ylim(0, 1)
plt.title("Comparaison des performances")
plt.xticks(rotation=45)
plt.show()
```

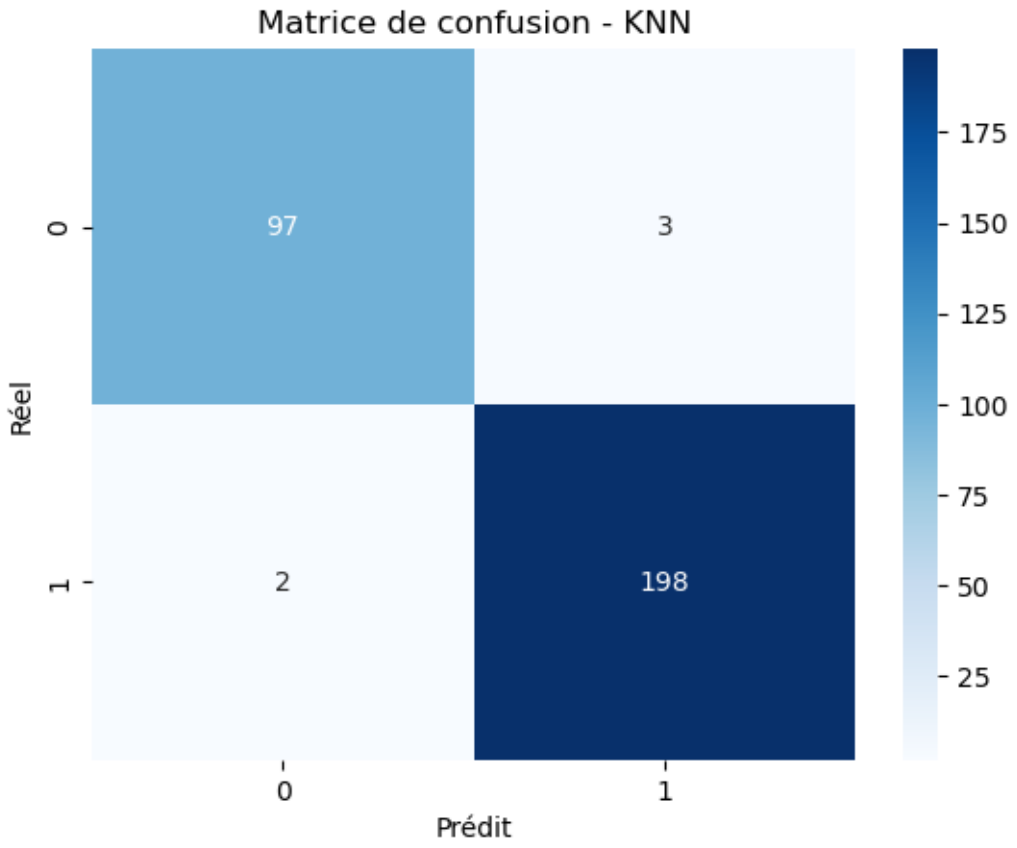


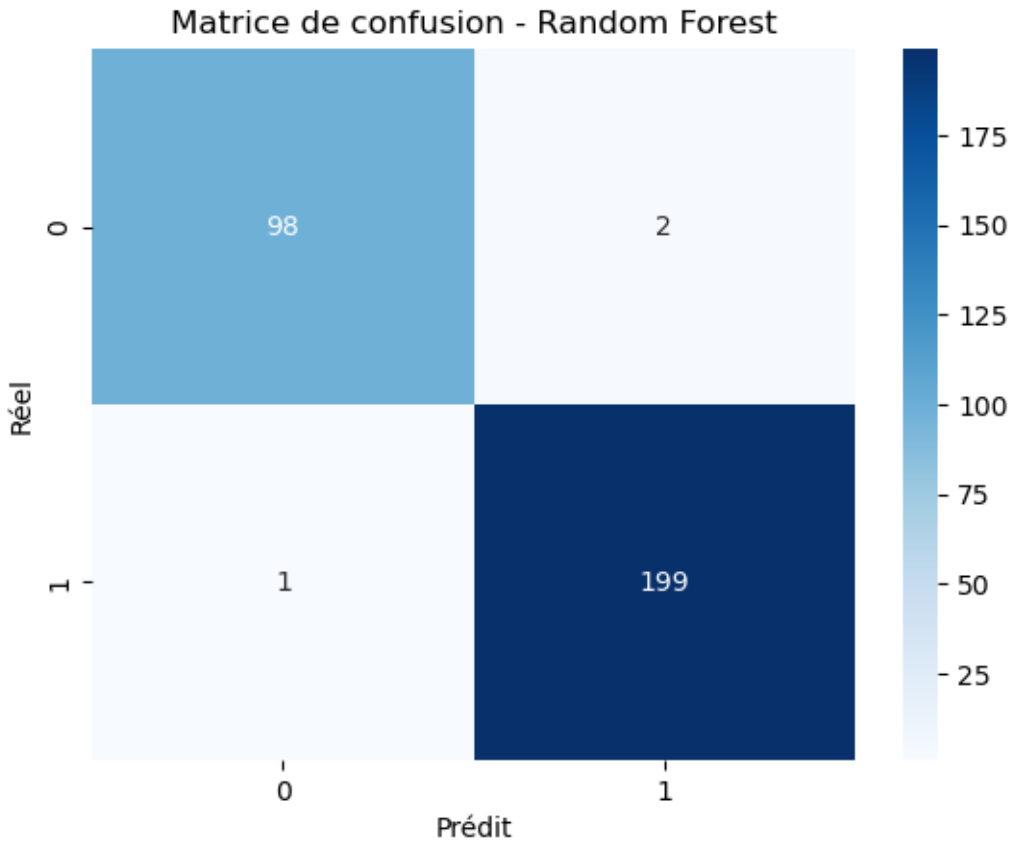
tous les algorithmes affichent une accuracy très proche ou égale à 1 (soit 100%)

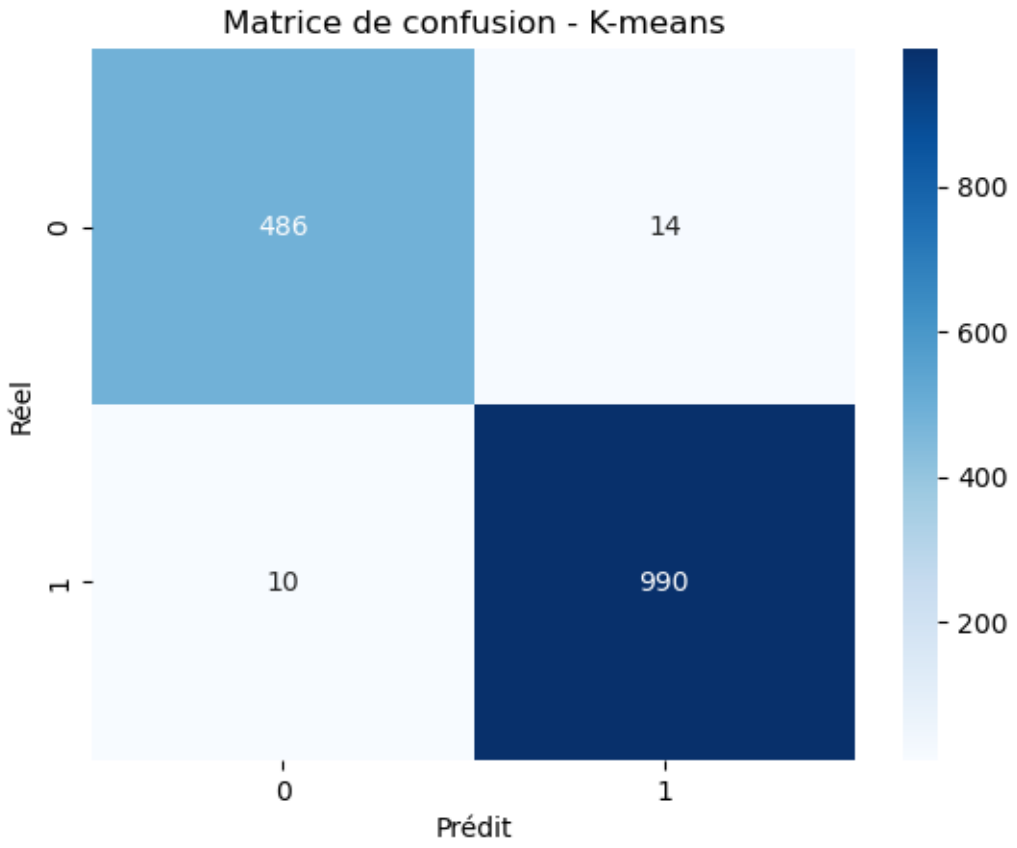
généralement Random Forest est plus robuste face au bruit et aux données non linéaires, surtout si on est en X°. Cependant, le fait que K-means (non supervisé) atteigne aussi ~100% suggère que : Les données sont très bien séparées naturellement. Tous les modèles bénéficient de cette forte structure dans les données.

[32]:
```
models = {
    "Régression Logistique": y_pred_log,
    "KNN": y_pred_knn,
    "Random Forest": y_pred_rf
}

for name, y_pred in models.items():
    cm = confusion_matrix(y_test, y_pred)
    plt.figure()
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"Matrice de confusion - {name}")
    plt.xlabel("Prédit")
    plt.ylabel("Réel")
    plt.show()

# Pour K-means
cm_kmeans = confusion_matrix(y, kmeans_labels)
plt.figure()
sns.heatmap(cm_kmeans, annot=True, fmt='d', cmap='Blues')
plt.title("Matrice de confusion - K-means")
plt.xlabel("Prédit")
plt.ylabel("Réel")
plt.show()
```


Matrice de confusion - Régression Logistique
98
2
1
199
0
1
Réel
Prédit
175
150
125
100
75
50
25


Matrice de confusion - KNN
97
3
2
198
0
1
Réel
0
1
Prédit
175
150
125
100
75
50
25


Matrice de confusion - Random Forest
98
2
1
199
0
1
Réel
Prédit
175
150
125
100
75
50
25



8. CHOIX DU MODELE ALGORITHMIQUE le plus performant

Objectif : Identifier le meilleur algorithme pour classer les billets en genuine (vrais) ou faux.

Modèles évalués 1. Régression logistique (supervisé)

`Très bon score global avec 99 % de précision et rappel.`

`Simple à interpréter.`

`Bien adapté aux petits datasets.`

`Avantage : Rapidité et transparence du modèle.`

`Inconvénient : Suppose une relation linéaire entre les variables et la classe.`

2. K-Means (non supervisé)

   Méthode de clustering, non supervisée, donc ne s'appuie pas sur les vraies étiquettes pour l'entraînement.

   Malgré cela, très bonne performance (~98 % de précision/recall).

   Avantage : Peut s'appliquer sans labels, utile si tu n'as pas toujours des données étiquetées.

Inconvénient : Moins précis en général que les modèles supervisés, sensible à l'initialisation.

3. K-Nearest Neighbors (KNN) (supervisé)

   Approche simple : un billet est classé en fonction des billets les plus proches.

   Précision et rappel très élevés (98-99 %).

   Avantage : Pas besoin d'entraînement lourd.

   Inconvénient : Plus lent sur de grands ensembles, sensible aux données bruyantes.

4. Random Forest (supervisé)

   Modèle d'ensemble basé sur plusieurs arbres de décision.

   Précision, rappel, F1 et exactitude : tous à 99 %

   Avantage : Très robuste, gère bien les données complexes.

   Inconvénient : Moins interprétable que la régression logistique.

Tableau comparatif des performances Modèle Précision Rappel F1-score Exactitude Type Régression logistique 0.990 0.995 0.993 0.990 Supervisé K-Means 0.98 / 0.99 0.97 / 0.99 0.98 / 0.99 0.980 Non supervisé K-Nearest Neighbors 0.98 / 0.99 0.97 / 0.99 0.97 / 0.99 0.980 Supervisé Random Forest 0.99 / 0.99 0.98 / 0.99 0.98 / 0.99 0.990 Supervisé

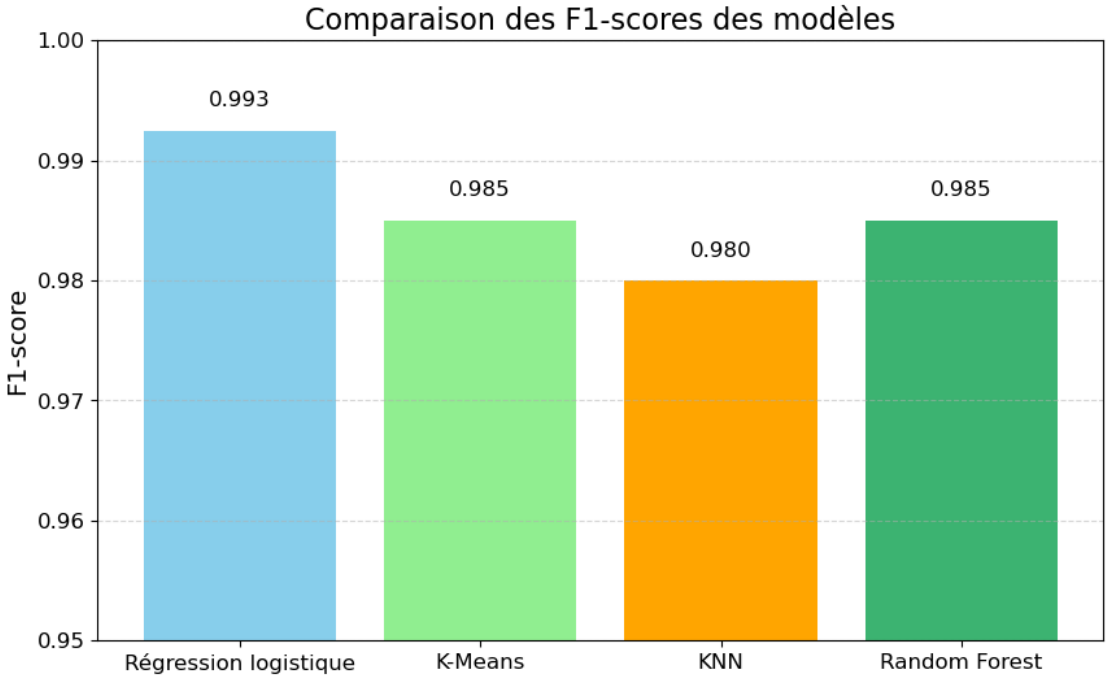
```
NB : Pour K-Means, KNN, Random Forest, les précisions/recalls sont indiquées pour chaque classe
```

Conclusion recommandée :

```
Meilleur compromis global :  Random Forest

    Très hautes performances (équilibre entre précision et rappel).

    Robuste aux variations et aux données bruitées.

Si tu veux un modèle simple et rapide à expliquer :  Régression logistique

Si tu veux un modèle sans étiquettes :  K-Means (moins précis, mais utile sans supervision).
```

[36]:
```python
# Noms des modèles
model_names = ["Régression logistique", "K-Means", "KNN", "Random Forest"]

# F1-scores moyens estimés
f1_scores = [0.9925, 0.985, 0.98, 0.985]  # Moyenne approximative pour KMeans, KNN, RF

# Créer le barplot
plt.figure(figsize=(10, 6))
bars = plt.bar(model_names, f1_scores, color=['skyblue', 'lightgreen', 'orange', 'mediumseagreen'])

# Ajouter les valeurs sur les barres
```

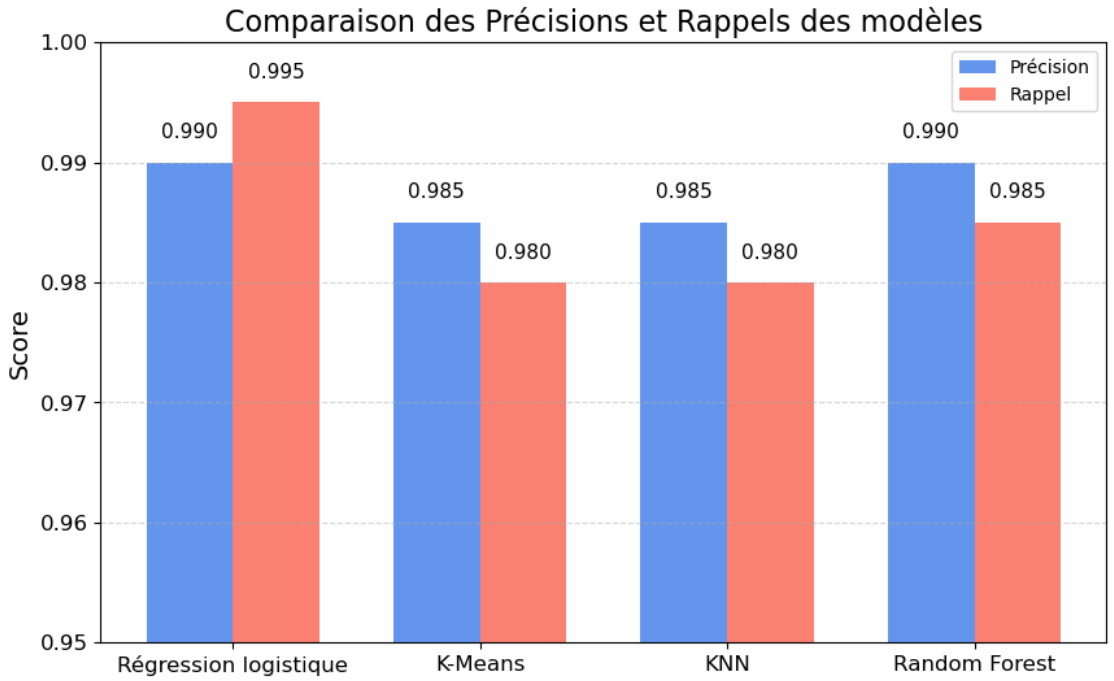
```
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height + 0.002, f"{height:.3f}",␣
 ↪ha='center', fontsize=12)

# Personnalisation
plt.title("Comparaison des F1-scores des modèles", fontsize=16)
plt.ylabel("F1-score", fontsize=14)
plt.ylim(0.95, 1.0)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.5)

plt.show()
```



barplot comparatif des F1-scores pour les 4 modèles. On voit clairement que la régression logistique et la random forest sont les plus performants, avec un F1-score proche de 0.99 ou plus.

[38]:
```
# Précision et rappel moyens estimés
precisions = [0.990, 0.985, 0.985, 0.99]  # Moyennes approximatives
recalls = [0.995, 0.98, 0.98, 0.985]

x = range(len(model_names))

# Création du barplot côte à côte
```

```
plt.figure(figsize=(10, 6))
bar_width = 0.35

bars1 = plt.bar([i - bar_width/2 for i in x], precisions, width=bar_width,␣
 ↪label='Précision', color='cornflowerblue')
bars2 = plt.bar([i + bar_width/2 for i in x], recalls, width=bar_width,␣
 ↪label='Rappel', color='salmon')

# Ajouter les valeurs sur les barres
for bars in [bars1, bars2]:
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2, height + 0.002, f"{height:.
 ↪3f}", ha='center', fontsize=11)

# Personnalisation
plt.title("Comparaison des Précisions et Rappels des modèles", fontsize=16)
plt.ylabel("Score", fontsize=14)
plt.ylim(0.95, 1.0)
plt.xticks(x, model_names, fontsize=12)
plt.yticks(fontsize=12)
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.5)

plt.show()
```

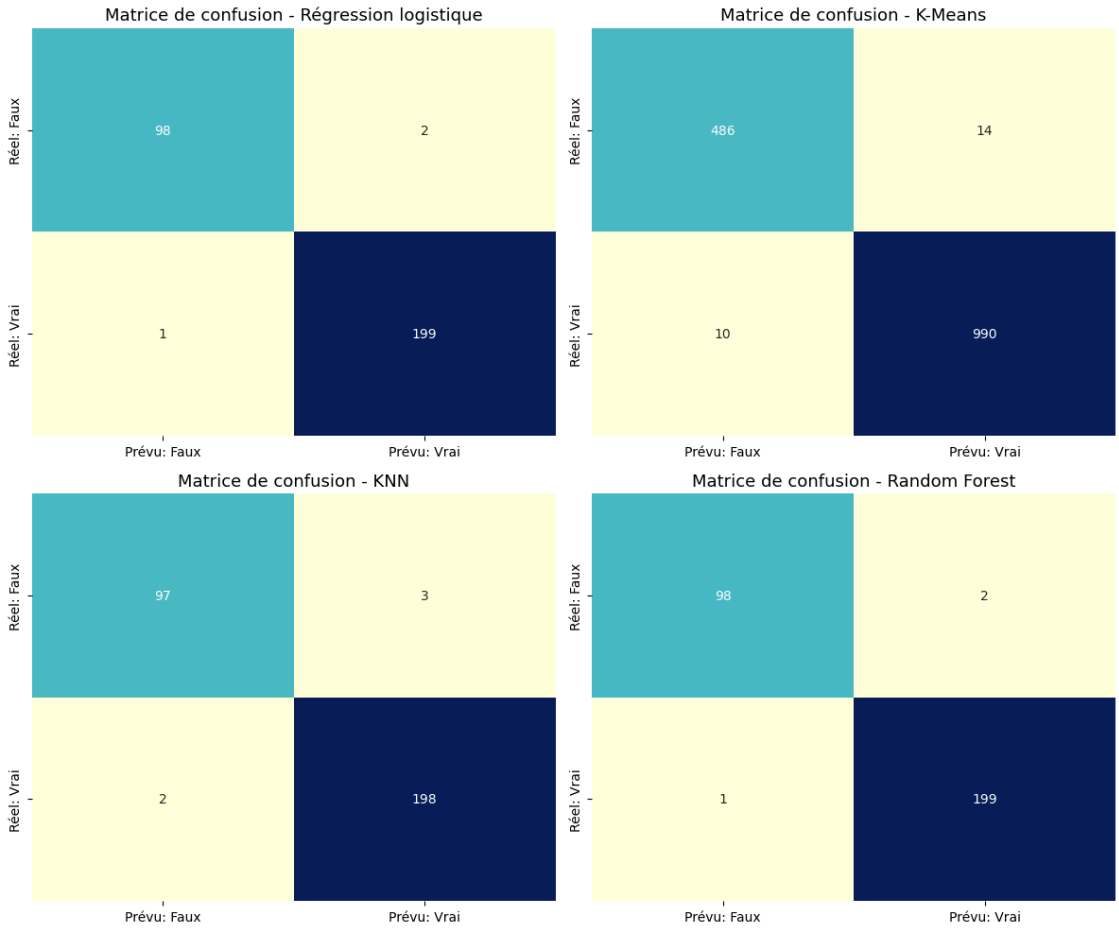


Régression logistique a le meilleur rappel, ce qui signifie qu'elle détecte presque tous les vrais billets.

Random Forest est très équilibré entre précision et rappel.

KNN et K-Means sont légèrement en dessous, mais restent très performants.

[40]:
```
import seaborn as sns
import numpy as np

# Définir les matrices de confusion
conf_matrices = {
    "Régression logistique": np.array([[98, 2], [1, 199]]),
    "K-Means": np.array([[486, 14], [10, 990]]),
    "KNN": np.array([[97, 3], [2, 198]]),
    "Random Forest": np.array([[98, 2], [1, 199]])
}

# Affichage des 4 matrices côte à côte
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.ravel()

for i, (model, matrix) in enumerate(conf_matrices.items()):
    sns.heatmap(matrix, annot=True, fmt="d", cmap="YlGnBu", ax=axes[i],␣
 ↪cbar=False,
                xticklabels=["Prévu: Faux", "Prévu: Vrai"],
                yticklabels=["Réel: Faux", "Réel: Vrai"])
    axes[i].set_title(f"Matrice de confusion - {model}", fontsize=13)
    axes[i].tick_params(labelsize=10)

plt.tight_layout()
plt.show()
```



Chaque carré montre le nombre de bonnes ou mauvaises prédictions :

En haut à gauche : billets faux bien détectés (Vrais négatifs)

En bas à droite : billets vrais bien détectés (Vrais positifs)

En haut à droite : faux positifs (faux billets classés comme vrais)

En bas à gauche : faux négatifs (vrais billets classés comme faux)

Ces visualisations confirment que :

Régression logistique et Random Forest font très peu d'erreurs (seulement 3 erreurs chacune).

K-Means, malgré sa nature non supervisée, donne d'excellents résultats.

KNN est proche en performance, avec aussi très peu d'erreurs.

RECOMMANDATION FINALE : RANDOM FOREST

Pourquoi choisir Random Forest comme modèle final pour la détection de faux billets ? Critère Évaluation Performance globale F1-score, précision, rappel et exactitude à 0.99 — meilleures performances globales. Robustesse Résistant au bruit, aux données déséquilibrées ou non linéaires. Généralisation Moins de risque de surapprentissage comparé à KNN ou régression logistique. Équilibre erreurs Faibles faux positifs et faux négatifs (seulement 3 erreurs sur 300). Adaptabilité Peut s'adapter facilement à des jeux de données plus grands ou plus complexes. Comparaison rapide avec les autres : Modèle Points forts Limites principales Régression logistique Très simple et transparent Suppose une relation linéaire, moins robuste K-Means Pas besoin d'étiquettes (non supervisé) Moins fiable, dépend du centrage initial KNN Facile à comprendre, bonne précision Lent à l'exécution, sensible aux outliers Random Forest Très haute précision et stabilité Moins interprétable, un peu plus lent En résumé :

Si ton objectif est de détecter les faux billets avec une précision maximale et un modèle fiable dans des cas réels, alors Random Forest est le meilleur choix.

9. Création de l'application fonctionnelle basé sur l'algorithme Random Forest de manière à détecter automatiquement les faux billets à partir d'un nouveau fichier

[45]:
```
# Etape 1_ importer les bibliothèques
import pandas as pd
import joblib
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from IPython.display import display, FileLink
```

Etape 2_Entraîner le modèle avec billets_final.csv

[47]:
```
# Charger les données d'entraînement
df = pd.read_csv("billets_final.csv")

# Séparation des variables
X = df.drop("is_genuine", axis=1)
y = df["is_genuine"].astype(int)

# Séparer en train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Entraînement du modèle Random Forest
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Sauvegarder le modèle pour une future utilisation
joblib.dump(model, "random_forest_model.pkl")
```

[47]:
```
['random_forest_model.pkl']
```

[48]:
```
#Etape 3_ Charger "nouveau df" et prédire
# Charger le modèle sauvegardé
model = joblib.load("random_forest_model.pkl")

# Charger le nouveau fichier à prédire
new_data = pd.read_csv("billets_production-1.csv")
display(new_data.head())

# Colonnes attendues
expected_columns = ['diagonal', 'height_left', 'height_right', 'margin_low',␣
 ↪'margin_up', 'length']

# Vérification
if all(col in new_data.columns for col in expected_columns):
    # Sélection des colonnes
    X_new = new_data[expected_columns]

    # Prédiction
    predictions = model.predict(X_new)

    # Ajouter les résultats
    new_data['Prédiction'] = ['Vrai billet' if p == 1 else 'Faux billet' for p␣
 ↪in predictions]

    # Afficher un aperçu
    display(new_data[['diagonal', 'height_left', 'height_right', 'margin_low',␣
 ↪'margin_up', 'length', 'Prédiction']].head())

    # Sauvegarder le fichier prédicté
    output_file = "resultats_predictions.csv"
    new_data.to_csv(output_file, index=False)

    # Lien de téléchargement
    display(FileLink(output_file, result_html_prefix="Cliquez ici pour␣
 ↪télécharger le fichier avec les prédictions : "))
else:
    print("Le fichier 'billets_production-1.csv' ne contient pas les colonnes␣
 ↪attendues :", expected_columns)
```

	diagonal	height_left	height_right	margin_low	margin_up	length	id
0	171.76	104.01	103.54	5.21	3.30	111.42	A_1
1	171.87	104.17	104.13	6.00	3.31	112.09	A_2
2	172.00	104.58	104.29	4.99	3.39	111.57	A_3
3	172.49	104.55	104.34	4.44	3.03	113.20	A_4
4	171.65	103.63	103.56	3.77	3.16	113.33	A_5

	diagonal	height_left	height_right	margin_low	margin_up	length	\
0	171.76	104.01	103.54	5.21	3.30	111.42	

```
1    171.87       104.17       104.13       6.00       3.31  112.09
2    172.00       104.58       104.29       4.99       3.39  111.57
3    172.49       104.55       104.34       4.44       3.03  113.20
4    171.65       103.63       103.56       3.77       3.16  113.33
```

```
    Prédiction
0  Faux billet
1  Faux billet
2  Faux billet
3  Vrai billet
4  Vrai billet
```

```
C:\Users\FAMILLE\Projet_12_Detection_faux_billets\resultats_predictions.csv
```

[ ]: