PROJET 12 Détectez des faux billets (ONCFM)

Sommaire

Étape 1 - Importation des librairies et chargement des fichiers

1.1 - Importation des librairies

In [5]:
```
#Importation des librairies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1.2 - Chargement des fichiers

In [7]:
```
#Importation du fichier
billets = pd.read_csv("billets.csv", sep=';')
```

Étape 2 - Analyse exploratoire des fichiers

2.1 - Analyse exploratoire du fichier billets

In [10]:
```
#Affichage du dataset
billets
```

Out[10]:

	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length
0	True	171.81	104.86	104.95	4.52	2.89	112.83
1	True	171.46	103.36	103.66	3.77	2.99	113.09
2	True	172.69	104.48	103.50	4.40	2.94	113.16
3	True	171.36	103.91	103.94	3.62	3.01	113.51
4	True	171.73	104.28	103.46	4.04	3.48	112.54
...	...	...	...	...	...	...	...
1495	False	171.75	104.38	104.17	4.42	3.09	111.28
1496	False	172.19	104.63	104.44	5.27	3.37	110.97
1497	False	171.80	104.01	104.12	5.51	3.36	111.95
1498	False	172.06	104.28	104.06	5.17	3.46	112.25
1499	False	171.47	104.15	103.82	4.63	3.37	112.07

1500 rows × 7 columns

In [11]:
```
#Description du dataset
billets.describe()
```

Out[11]:

	diagonal	height_left	height_right	margin_low	margin_up	length
count	1500.000000	1500.000000	1500.000000	1463.000000	1500.000000	1500.00000
mean	171.958440	104.029533	103.920307	4.485967	3.151473	112.67850
std	0.305195	0.299462	0.325627	0.663813	0.231813	0.87273
min	171.040000	103.140000	102.820000	2.980000	2.270000	109.49000
25%	171.750000	103.820000	103.710000	4.015000	2.990000	112.03000
50%	171.960000	104.040000	103.920000	4.310000	3.140000	112.96000
75%	172.170000	104.230000	104.150000	4.870000	3.310000	113.34000
max	173.010000	104.880000	104.950000	6.900000	3.910000	114.44000

In [12]:
```
# Informations sur le df billets
billets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   is_genuine    1500 non-null   bool   
 1   diagonal      1500 non-null   float64
 2   height_left   1500 non-null   float64
 3   height_right  1500 non-null   float64
 4   margin_low    1463 non-null   float64
 5   margin_up     1500 non-null   float64
 6   length        1500 non-null   float64
dtypes: bool(1), float64(6)
memory usage: 71.9 KB
```

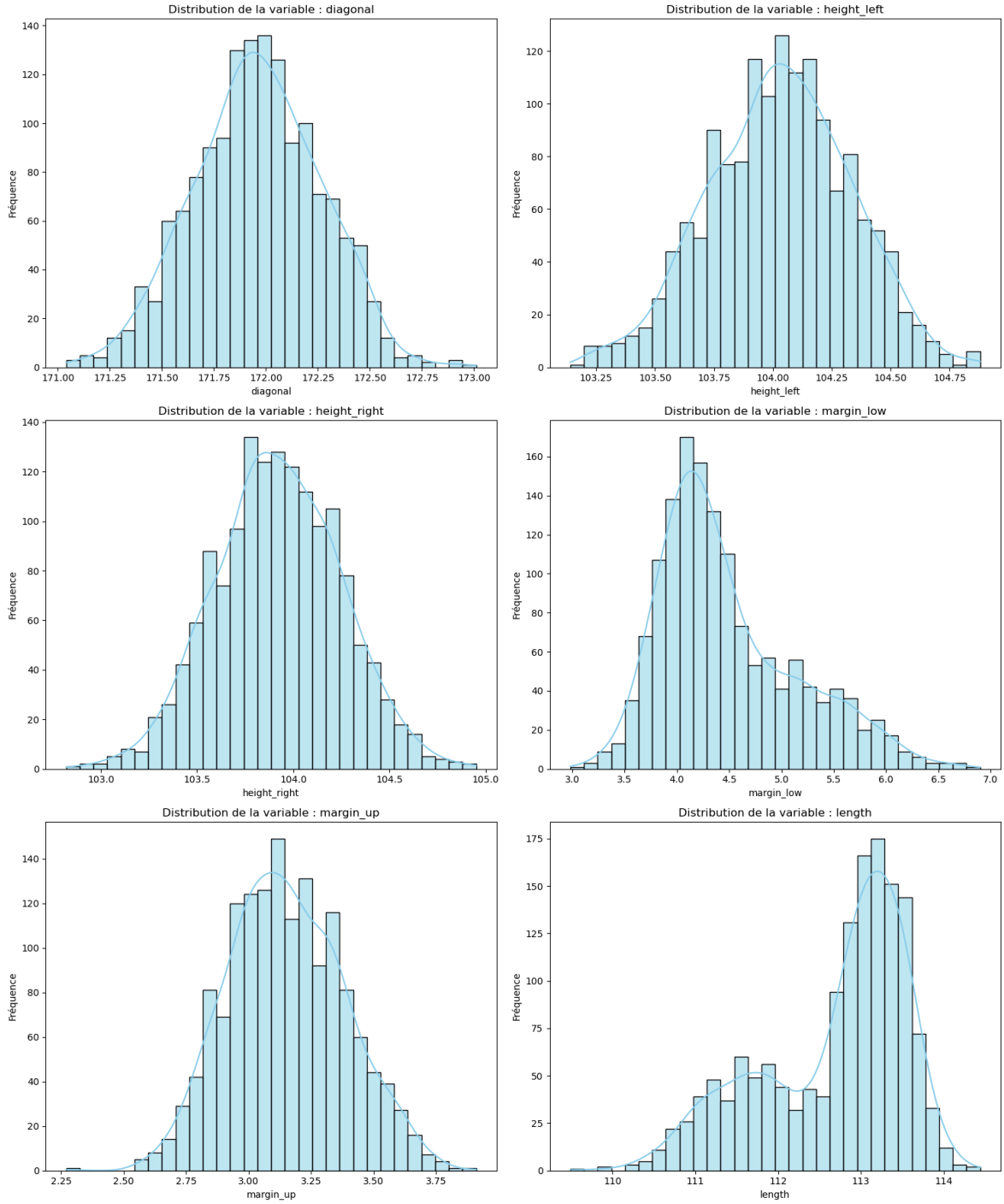
In [13]:
```python
# Vérification de la présence de doublon
billets.duplicated().sum()
```

Out[13]: 0

Il n'y a pas de doublons dans le fichier billets

Distribution des variables

In [16]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Sélection des colonnes numériques uniquement
numeric_cols = billets.select_dtypes(include='number').columns

# Taille de la figure
plt.figure(figsize=(15, len(numeric_cols) * 3))

# Boucle sur chaque variable
for i, col in enumerate(numeric_cols, 1):
    plt.subplot((len(numeric_cols) + 1) // 2, 2, i)
    sns.histplot(billets[col], kde=True, bins=30, color='skyblue', edgecolor='bl
    plt.title(f'Distribution de la variable : {col}')
    plt.xlabel(col)
    plt.ylabel('Fréquence')

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



In [17]:
```
#Vérification des manquants
billets.isna().sum()
```

Out[17]:
```
is_genuine       0
diagonal         0
height_left      0
height_right     0
margin_low      37
margin_up        0
length           0
dtype: int64
```

Il y a 37 valeurs manquantes dans la colonnes margin_low pour traiter cela nous allons réaliser

une régression linéaire pour les combler en attendant d'avoir les bonnes valeurs.

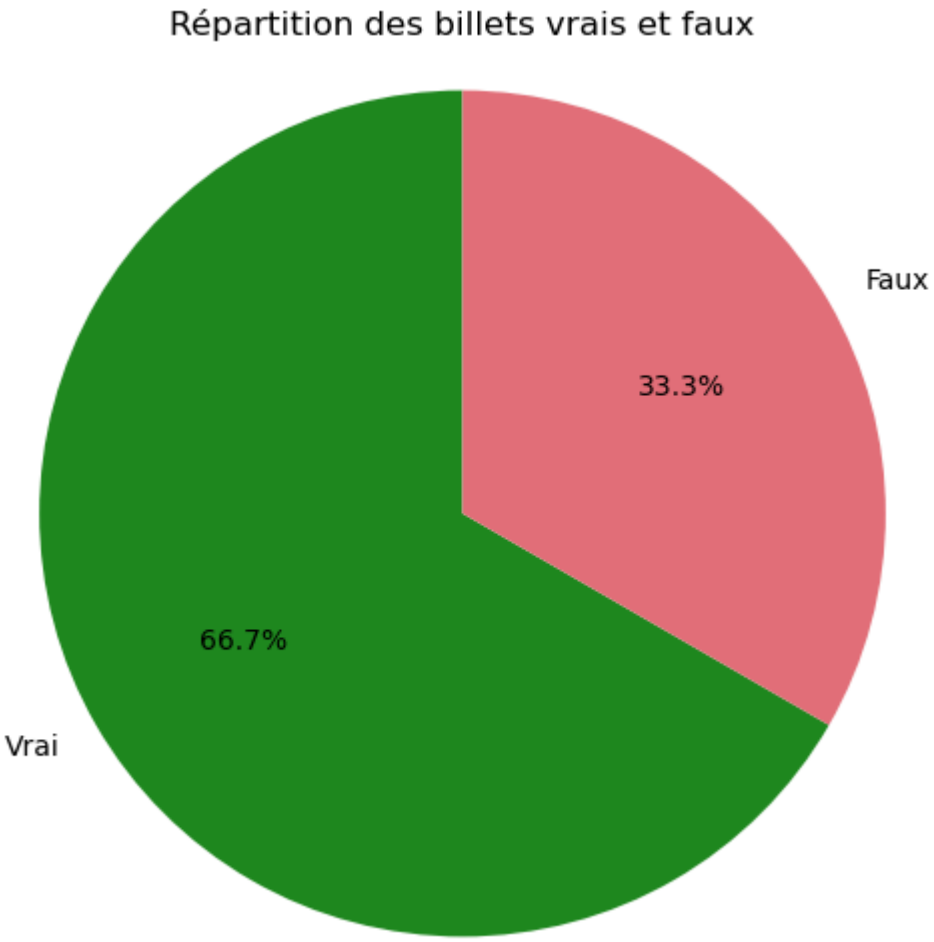
2.2 - Régression linéaire

In [20]:
```
#Importation des librairies
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
```

In [21]:
```
billets.count()
```

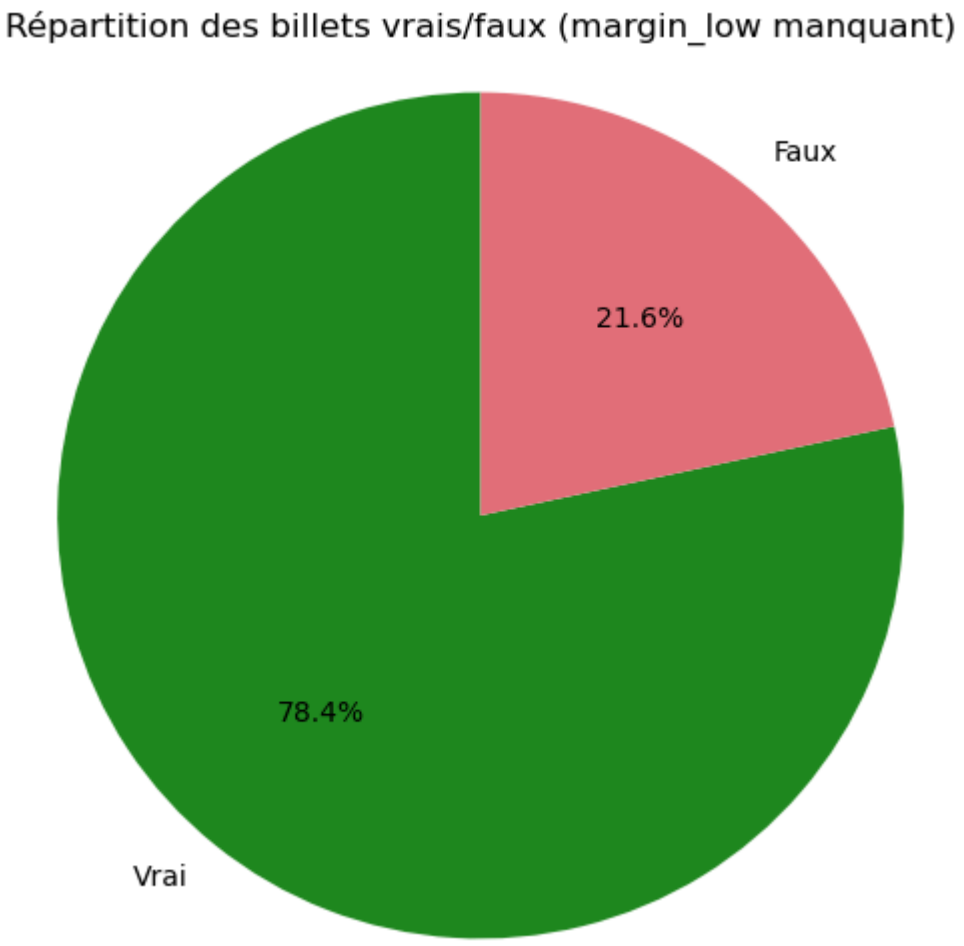
Out[21]:
```
is_genuine      1500
diagonal        1500
height_left     1500
height_right    1500
margin_low      1463
margin_up       1500
length          1500
dtype: int64
```

In [22]:
```
#Pourcenatage de billets vrais et faux
# Calcul des pourcentages
pourcentages = billets['is_genuine'].value_counts(normalize=True)

# Création du camembert
plt.figure(figsize=(6, 6))
plt.pie(pourcentages, labels=['Vrai', 'Faux'], autopct='%1.1f%%', startangle=90,
plt.title("Répartition des billets vrais et faux")
plt.axis('equal')
plt.show()
```



Il y a 33% de faux billets contre 67% de vrais billets

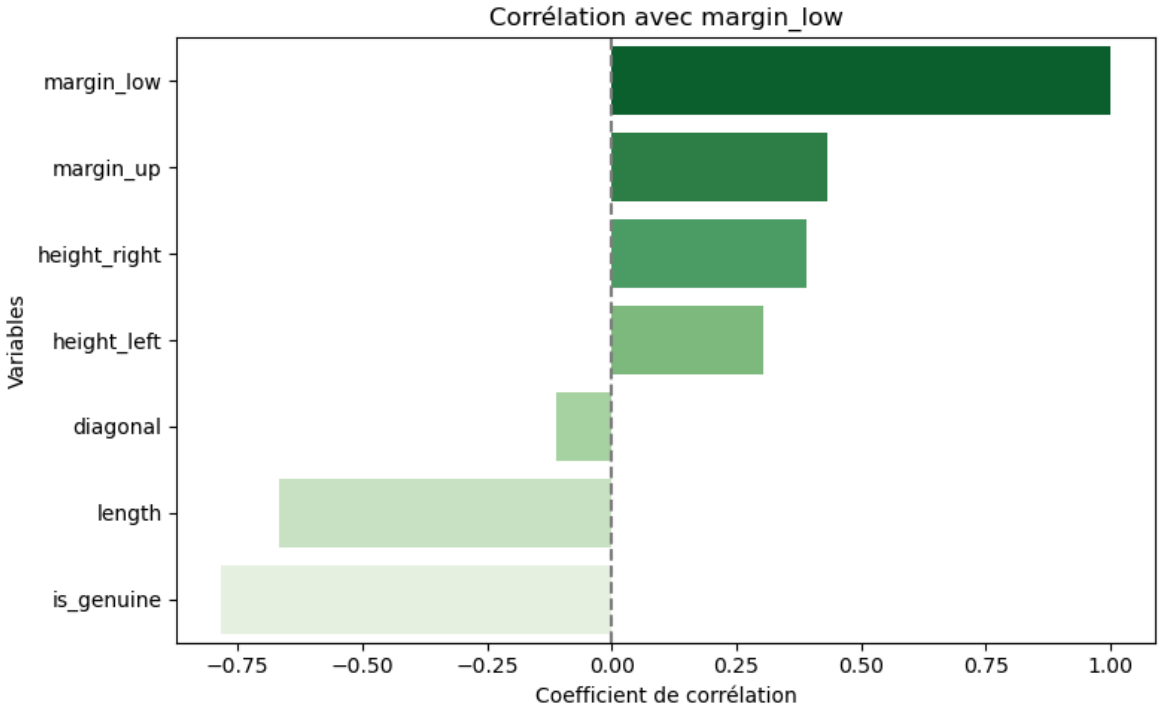
In [24]:
```
# Affichage de la répartition des billets vrais et faux parmi ceux qui ont une v
# Filtrer les billets avec margin_low manquant
subset = billets[billets['margin_low'].isna()]['is_genuine'].value_counts(normal

# Couleurs douces
colors = ['#228B22', '#E4717A']

# Affichage du camembert
plt.figure(figsize=(6, 6))
plt.pie(subset, labels=['Vrai', 'Faux'], autopct='%1.1f%%', startangle=90, color
plt.title("Répartition des billets vrais/faux (margin_low manquant)")
plt.axis('equal')
plt.show()
```



Pour ce qui est de la répartition des vrais/faux billets pour les manquants de margin_low = Il y a 22% de faux billets contre 78% de vrais billets

In [26]:
```
#Analyse de la corrélation
# Calcul des corrélations avec margin_low
correlations = billets.corr(numeric_only=True)['margin_low'].sort_values(ascendi

# Affichage en barres horizontales
plt.figure(figsize=(8, 5))
sns.barplot(x=correlations.values, y=correlations.index, palette='Greens_r')
plt.title("Corrélation avec margin_low")
plt.xlabel("Coefficient de corrélation")
plt.ylabel("Variables")
plt.axvline(x=0, color='gray', linestyle='--')
plt.tight_layout()
plt.show()
```

```
C:\Users\PC\AppData\Local\Temp\ipykernel_38764\2487935208.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.barplot(x=correlations.values, y=correlations.index, palette='Greens_r')
```



In [27]:
```
from sklearn.linear_model import LinearRegression
import pandas as pd

# Séparation des données complètes et incomplètes
train = billets[billets['margin_low'].notna()]
test = billets[billets['margin_low'].isna()]

X_train = train[['margin_up', 'height_left', 'height_right', 'length', 'diagonal
y_train = train['margin_low']

model = LinearRegression()
model.fit(X_train, y_train)

print("R² :", model.score(X_train, y_train))
```

```
R² : 0.4773366973063957
```

Ici R^2 = 0.477 → Le modèle capte donc presque la moitié de l'information utile pour prédire margin_low. C'est un modèle modérément explicatif.

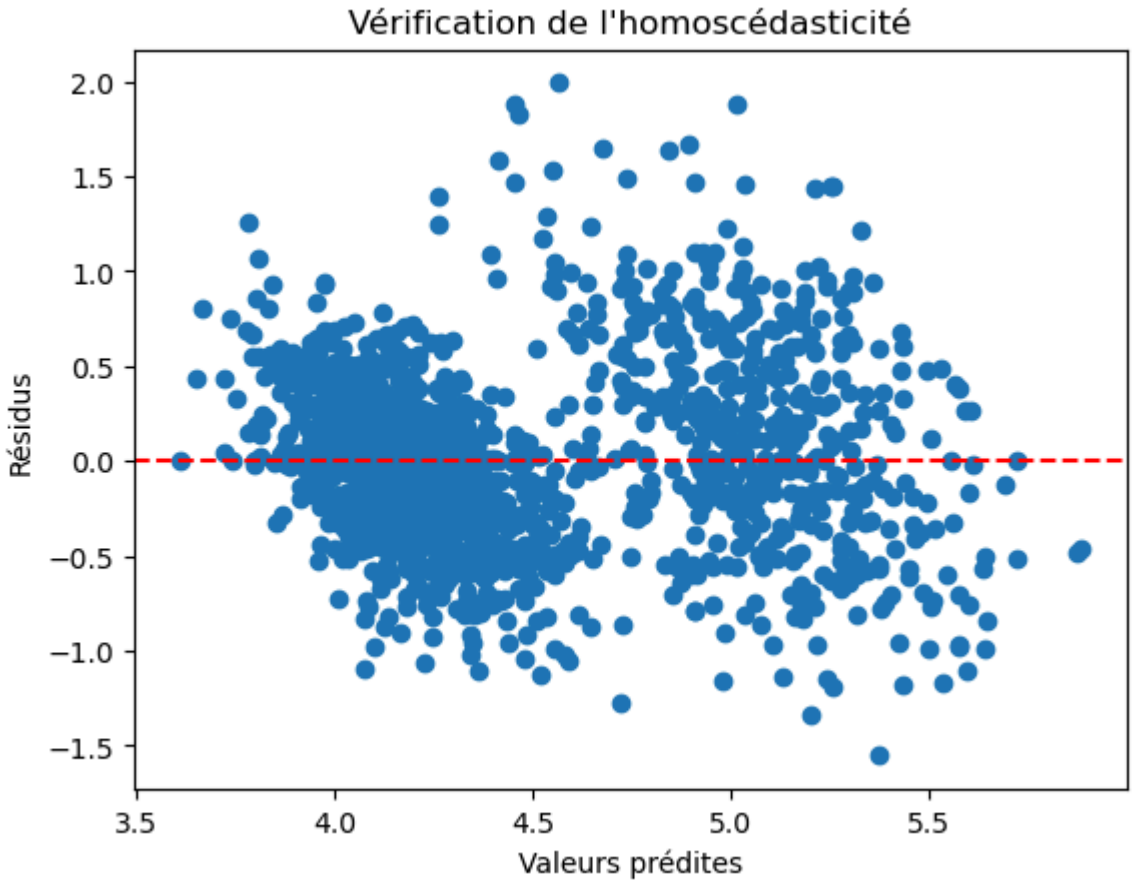
In [29]:
```
# Prédire margin_low pour les lignes où il manque
X_test = test[['margin_up', 'height_left', 'height_right', 'length', 'diagonal']
billets.loc[billets['margin_low'].isna(), 'margin_low'] = model.predict(X_test)

#Réintégrer dans le DataFrame complet
billets.loc[billets['margin_low'].isna(), 'margin_low'] = test['margin_low']
```

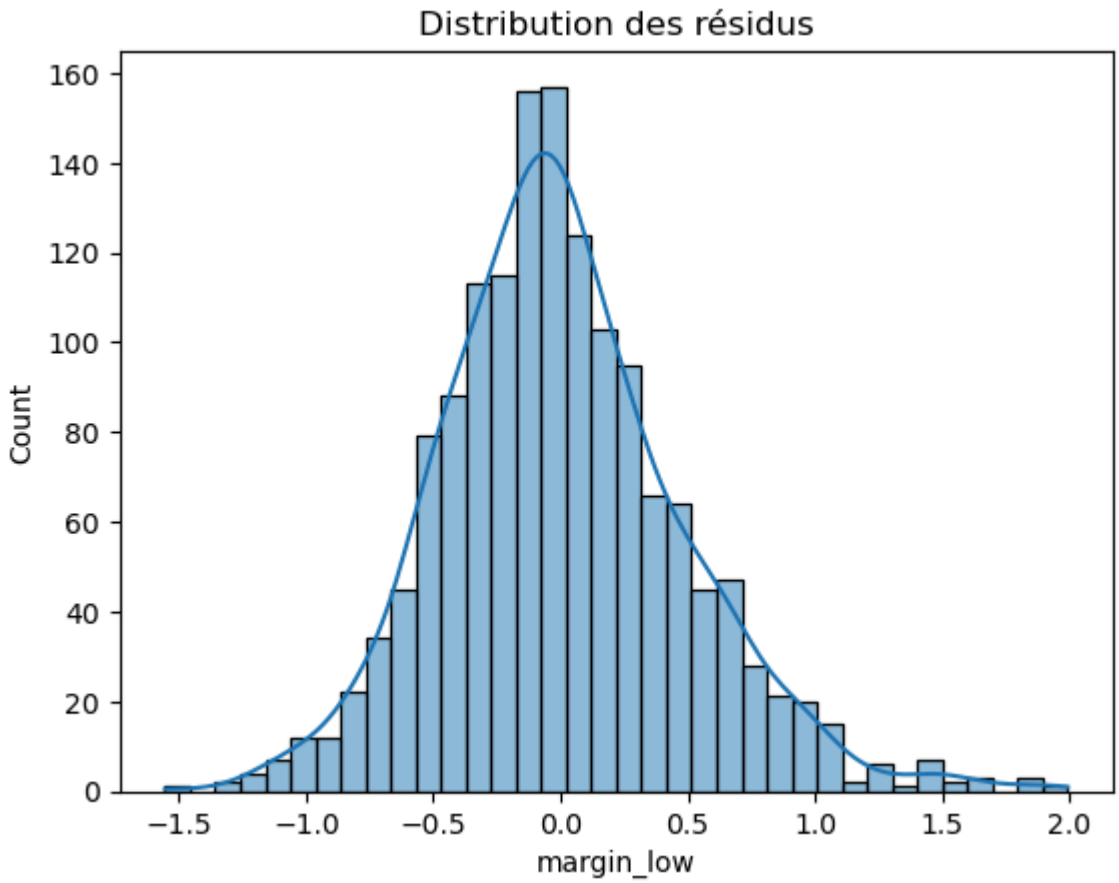
In [30]:
```
# Importation des librairies
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
from scipy import stats
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Création d'une variable combinée pour réduire la colinéarité
billets['height_mean'] = (billets['height_left'] + billets['height_right']) / 2

# Séparation des données complètes / incomplètes
train = billets[billets['margin_low'].notna()].copy()
test = billets[billets['margin_low'].isna()].copy()

# Entrainement
X_train = train[['margin_up', 'length', 'height_mean']]
y_train = train['margin_low']

# Vérification de la colinéarité avec VIF
vif_data = pd.DataFrame()
vif_data["Variable"] = X_train.columns
vif_data["VIF"] = [variance_inflation_factor(X_train.values, i) for i in range(X
print("VIF des variables explicatives :\n", vif_data)

# Création et entraînement du modèle
model = LinearRegression()
model.fit(X_train, y_train)

# Évaluation du modèle
r2 = model.score(X_train, y_train)
print(f"\nCoefficient de détermination R² : {r2:.3f}")

# Vérification des coefficients
coef = pd.DataFrame({
    'Variable': X_train.columns,
    'Coefficient': model.coef_
}).sort_values(by='Coefficient', ascending=False)
print("\nCoefficients de la régression :\n", coef)

# Vérification des hypothèses du modèle
y_pred = model.predict(X_train)
residus = y_train - y_pred

# Homoscédasticité
plt.scatter(y_pred, residus)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel("Valeurs prédites")
plt.ylabel("Résidus")
plt.title("Vérification de l'homoscédasticité")
plt.show()

# Normalité des résidus
sns.histplot(residus, kde=True)
plt.title("Distribution des résidus")
plt.show()

stat, p_value = stats.shapiro(residus)
print(f"Test de Shapiro-Wilk : stat={stat:.3f}, p-value={p_value:.3f}")
if p_value > 0.05:
    print("Les résidus suivent une distribution normale (p > 0.05)")
else:
    print("Les résidus ne semblent pas parfaitement normaux (p < 0.05)")

# Vérification des corrélations entre variables explicatives
```

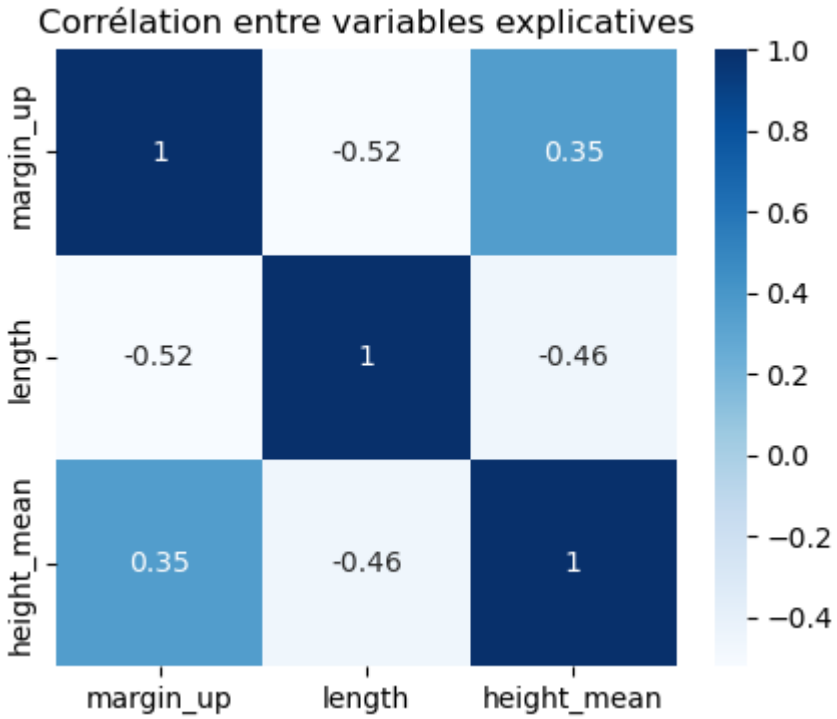
```python
corr = X_train.corr()
plt.figure(figsize=(5, 4))
sns.heatmap(corr, annot=True, cmap='Blues')
plt.title("Corrélation entre variables explicatives")
plt.show()

# Imputation des valeurs manquantes dans margin_low
if not test.empty:
    X_test = test[['margin_up', 'length', 'height_mean']]
    billets.loc[billets['margin_low'].isna(), 'margin_low'] = model.predict(X_te

# Vérification finale
print(f"Valeurs manquantes restantes dans margin_low : {billets['margin_low'].is
```

```
VIF des variables explicatives :
       Variable           VIF
0    margin_up    260.694257
1       length  16635.328184
2  height_mean  19056.357739

Coefficient de détermination R² : 0.481

Coefficients de la régression :
       Variable  Coefficient
2  height_mean     0.436950
0    margin_up     0.259612
1       length    -0.414433
```






```
Test de Shapiro-Wilk : stat=0.985, p-value=0.000
Les résidus ne semblent pas parfaitement normaux (p < 0.05)
```




```
Valeurs manquantes restantes dans margin_low : 0
```

Interprétation :

- Corrélation proche de 1 ou -1 → forte colinéarité
- Corrélation proche de 0 → pas de colinéarité notable

Le modèle de régression linéaire obtient un R^2 de 0.481, indiquant une précision moyenne.

Cependant, les valeurs prédites restent cohérentes avec la tendance générale des données. Comme seulement 37 valeurs sur 1500 sont manquantes, l'impact de cette approximation est négligeable pour la suite de notre analyse. Nous conservons donc les résultats de la régression linéaire pour compléter les valeurs manquantes de margin_low

In [33]:
```
# Suppression de la colonne height_mean
billets = billets.drop(columns='height_mean')
```

In [34]:
```
#Verication de la présence de valeurs manquantes
billets.isna().any()
```

Out[34]:
```
is_genuine      False
diagonal        False
height_left     False
height_right    False
margin_low      False
margin_up       False
length          False
dtype: bool
```

Étape 2 - Les algorithmes

In [36]:
```
# Supprimer les lignes avec des valeurs manquantes
billets_clean = billets.dropna()
```

In [37]:
```
#Vérification de la présence de valeur négatives
billets_clean.isna().any()
```

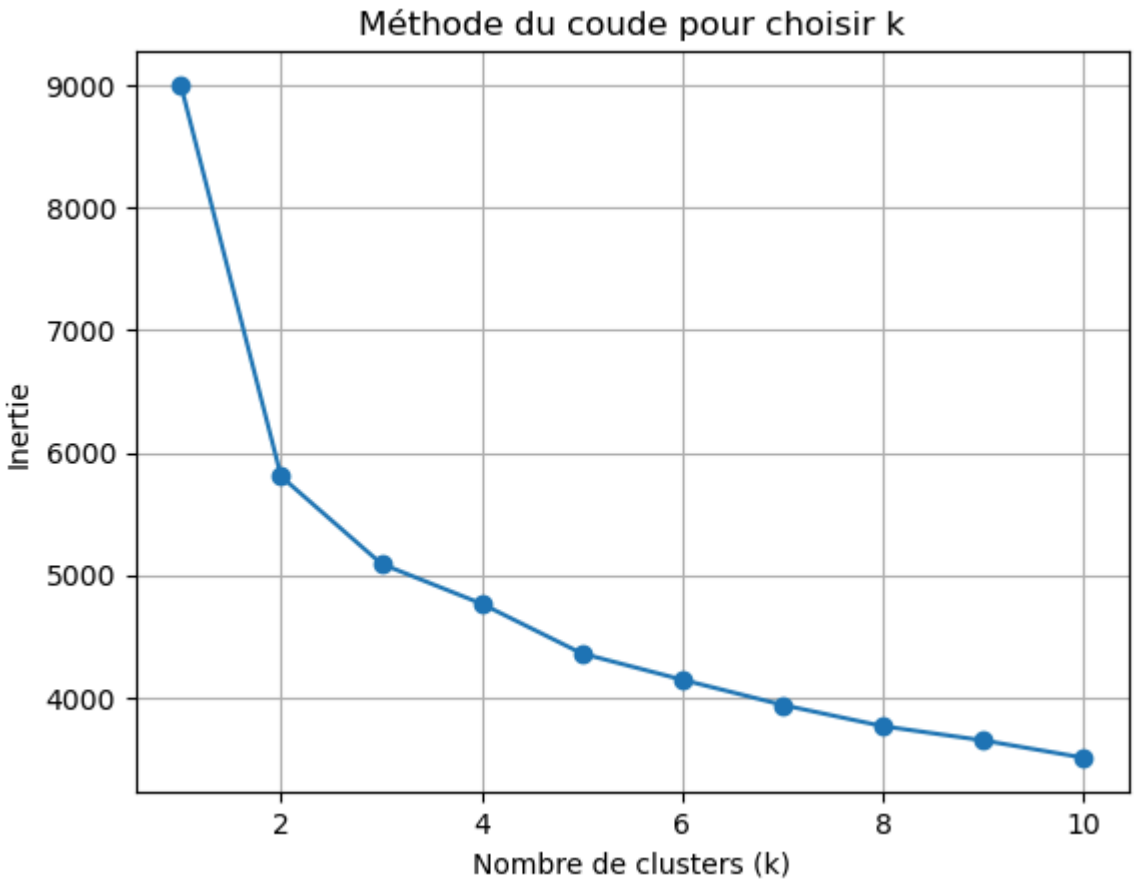
Out[37]:
```
is_genuine      False
diagonal        False
height_left     False
height_right    False
margin_low      False
margin_up       False
length          False
dtype: bool
```

3.1 - Kmeans

In [39]:
```
# On retire la colonne cible (is_genuine) pour ne garder que les variables numér
X = billets_clean.drop(columns=['is_genuine'])
```

In [40]:
```
# Scaling des données pour appliquer le K_means
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [41]:
```python
# Méthode du coude
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Teste plusieurs valeurs de k
inertia = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Affiche la courbe du coude
plt.plot(K_range, inertia, marker='o')
plt.xlabel('Nombre de clusters (k)')
plt.ylabel('Inertie')
plt.title('Méthode du coude pour choisir k')
plt.grid(True)
plt.show()
```



In [42]:
```python
import pandas as pd
```

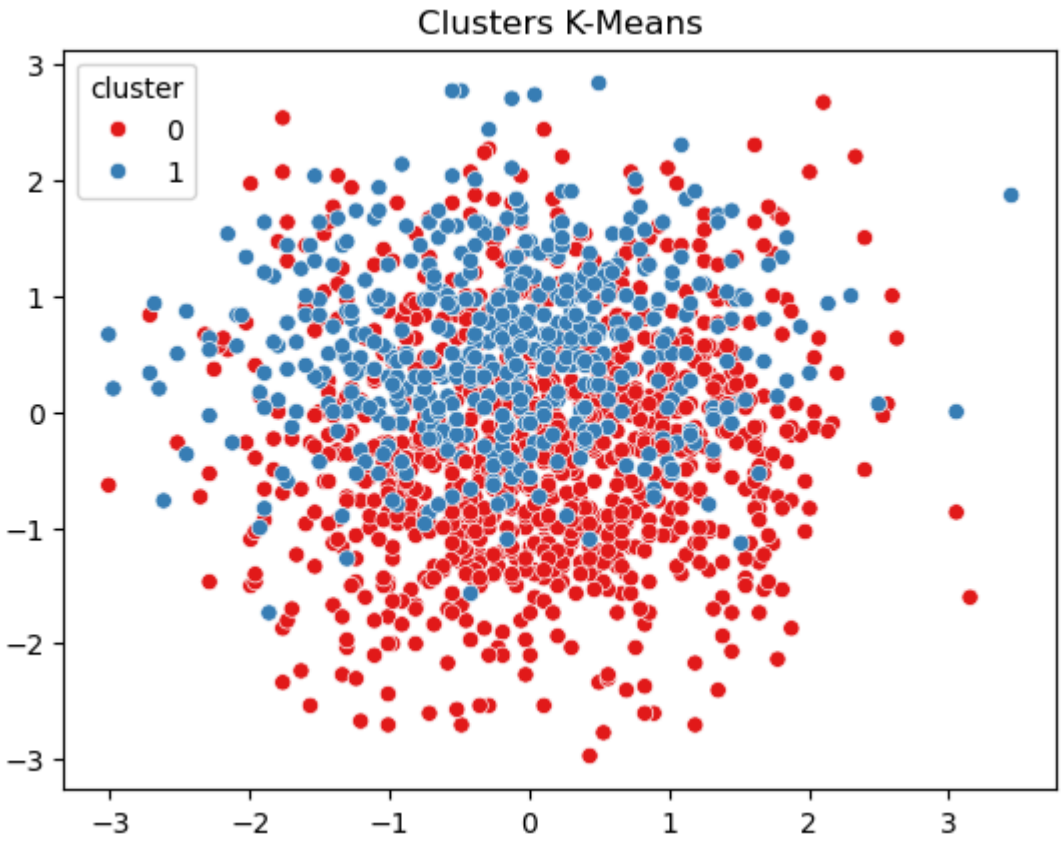
In [43]:
```python
# Standardisation des données
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [44]:
```python
# Appliquer K-Means
from sklearn.cluster import KMeans
```

```
# Choix du nombre de clusters
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_scaled)

# Ajout des labels au DataFrame
billets_clean['cluster'] = kmeans.labels_
```

In [45]:
```
#Comparer avec la vérité (is_genuine)
import seaborn as sns
import matplotlib.pyplot as plt

# Affichage croisé entre cluster et vérité
print(pd.crosstab(billets_clean['is_genuine'], billets_clean['cluster']))

# Visualisation
sns.scatterplot(x=X_scaled[:, 0], y=X_scaled[:, 1], hue=billets_clean['cluster']
plt.title("Clusters K-Means")
plt.show()
```

```
cluster       0    1
is_genuine
False        14  486
True        990   10
```



In [46]:
```
# Comptage des valeurs 0 et 1 dans la colonne 'cluster'
compte_cluster = billets_clean['cluster'].value_counts()
print(compte_cluster)
```

```
cluster
0    1004
1     496
Name: count, dtype: int64
```

In [47]:
```
#Evoluer la qualité du clustering
from sklearn.metrics import silhouette_score

score = silhouette_score(X_scaled, kmeans.labels_)
print("Score de silhouette :", score)
```

```
Score de silhouette : 0.3427474069132479
```

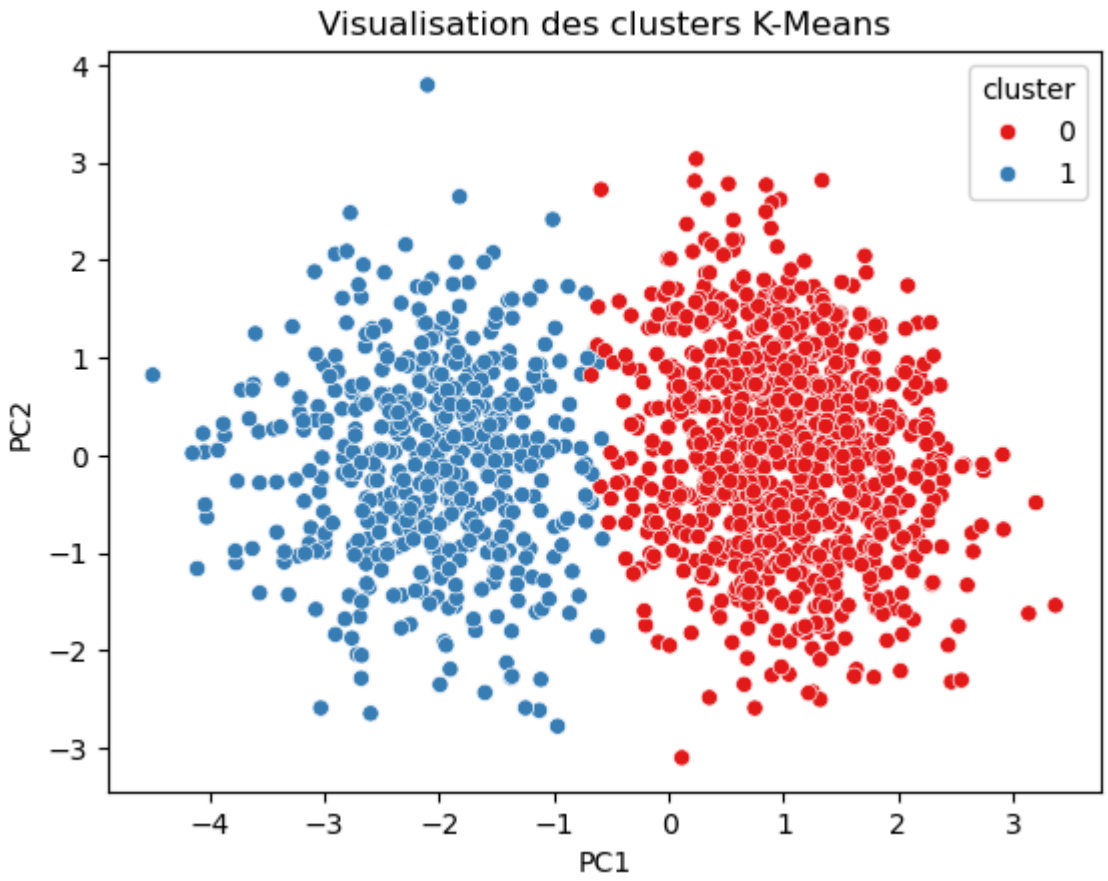
On va maintenant comparer les clusters avec la vérité pour voir si le kmeans correspond au fichier

In [49]:
```
pd.crosstab(billets_clean['is_genuine'], billets_clean['cluster'])
```

Out[49]:

cluster	0	1
is_genuine		
False	14	486
True	990	10

In [50]:
```
# Visualisation de l'ACP
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

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

# Ajoute les composantes au DataFrame
billets_clean['PC1'] = X_pca[:, 0]
billets_clean['PC2'] = X_pca[:, 1]

# Affiche les clusters
sns.scatterplot(data=billets_clean, x='PC1', y='PC2', hue='cluster', palette='Se
plt.title("Visualisation des clusters K-Means")
plt.show()
```



In [51]:
```
billets_clean.groupby('cluster').mean()
```

Out[51]:

cluster	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	l
0	0.986056	171.987729	103.945129	103.805588	4.119186	3.052540	113.1
1	0.020161	171.899153	104.200383	104.152520	5.220867	3.351734	111.6

K-Means ne connaît pas les vraies classes (is_genuine). Il regroupe les données par similarité, sans savoir ce qu'est un vrai ou un faux billet. Contrairement aux modèles supervisés, K-Means ne peut pas prédire si un nouveau billet est vrai ou faux sans une étape supplémentaire.

Quand K-Means est utile:

- Pour explorer les données sans étiquettes.
- Pour visualiser des regroupements naturels.
- Pour vérifier si les vrais/faux billets se séparent bien sans supervision.

3.2 - Regression logistique

In [55]:
```
# Import des bibliothèques
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

In [56]:
```
# Toutes les colonnes sauf la cible
X = billets_clean.drop(columns=['is_genuine'])
# la variable cible
y = billets_clean['is_genuine']
```

In [57]:
```
# Standardisation
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

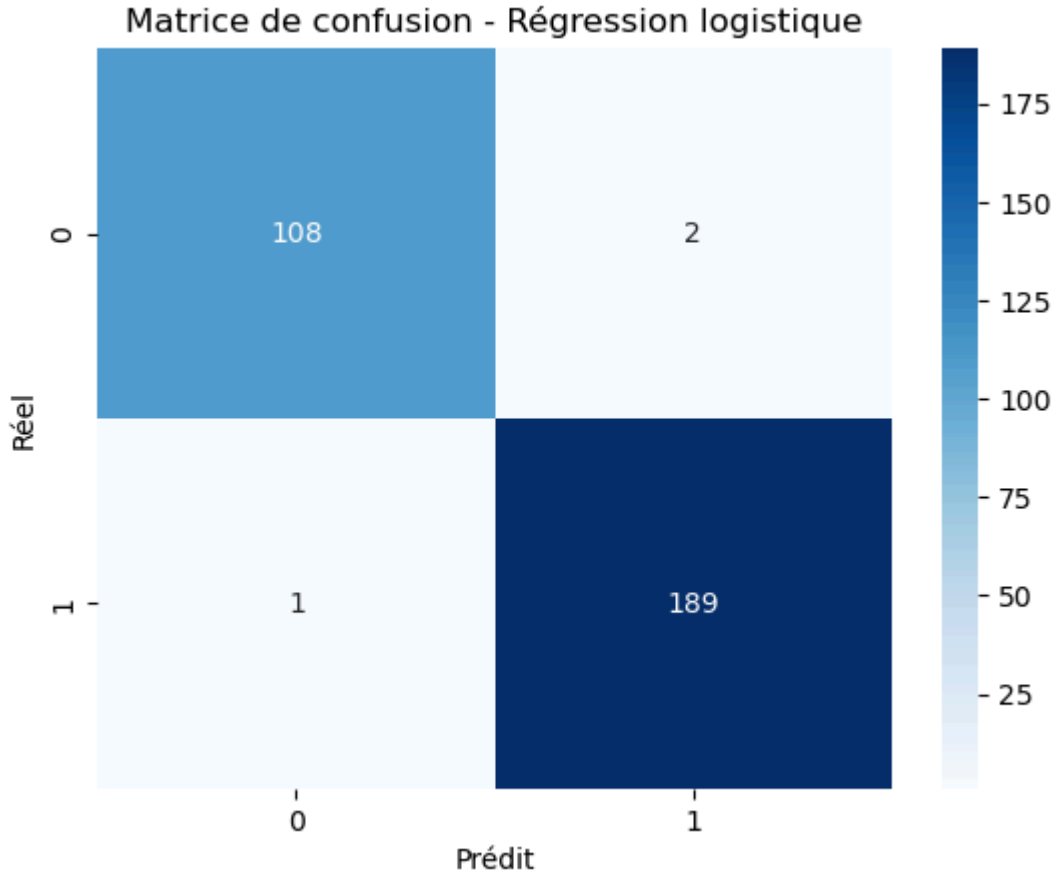
In [58]:
```
# Entrainement
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
```

In [59]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[59]:
```
LogisticRegression()
```

In [60]:
```
y_pred = model.predict(X_test)

# Rapport de classification
print(classification_report(y_test, y_pred))

# Matrice de confusion
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title("Matrice de confusion - Régression logistique")
plt.xlabel("Prédit")
plt.ylabel("Réel")
plt.show()
```

```
              precision    recall  f1-score   support

       False       0.99      0.98      0.99       110
        True       0.99      0.99      0.99       190

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



Matrice de confusion du modèle de Régression Logistique :

Cette matrice évalue la capacité du modèle à détecter les billets vrais (0) et faux (1).

- 108 vrais billets correctement identifiés (case en haut à gauche)
- 189 faux billets correctement identifiés (case en bas à droite)
- 2 faux billets classés à tort comme vrais (case en haut à droite)
- 1 vrai billet classé à tort comme faux (case en bas à gauche)

Performance : La précision globale est de 99 % avec seulement 3 erreurs sur 300 billets

In [62]:
```
coefficients = pd.Series(model.coef_[0], index=X.columns)
print(coefficients.sort_values())
```

```
margin_low     -1.878690
margin_up      -1.000363
cluster        -0.365802
diagonal       -0.166566
height_right   -0.144235
height_left    -0.035074
PC2             0.095701
PC1             1.830765
length          2.851442
dtype: float64
```

Application

In [64]:
```
import pandas as pd
import joblib

# Charger le modèle et le scaler
model = joblib.load('logistic_model.pkl')
scaler = joblib.load('scaler.pkl')
```

In [65]:
```
# Chargement du fichier csv à tester
df = pd.read_csv('billets_production.csv', sep=',', encoding='utf-8')
```

In [66]:
```
df
```

Out[66]:

	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

In [67]:
```
# Colonnes nécessaires
colonnes = ['diagonal', 'height_left', 'height_right', 'margin_low', 'margin_up'

# Vérifie que toutes les colonnes sont là
if not all(col in df.columns for col in colonnes):
    raise ValueError("Le fichier doit contenir les colonnes suivantes : " + ", "

# Supprime les lignes incomplètes
df_clean = df.dropna(subset=colonnes)

# Sélectionne les variables explicatives
X_new = df_clean[colonnes]
```

In [68]:
```
# Standardisation
X_scaled = scaler.transform(X_new)

# Prédictions
predictions = model.predict(X_scaled)

 # probabilité d'être vrai
probabilities = model.predict_proba(X_scaled)[:, 1]

# Ajout des résultats
df_clean['is_genuine_pred'] = predictions
df_clean['proba'] = probabilities
```

In [69]:
```
# Affiche les billets prédits comme faux
faux_billets = df_clean[df_clean['is_genuine_pred'] == False]
print(" Billets prédits comme faux :")
display(faux_billets[['diagonal', 'height_left', 'height_right', 'margin_low', '
```

```
# Affiche tous les résultats
print(" Résultats complets :")
display(df_clean[['diagonal', 'height_left', 'height_right', 'margin_low', 'marg
```

Billets prédits comme faux :

	diagonal	height_left	height_right	margin_low	margin_up	length	proba
0	171.76	104.01	103.54	5.21	3.30	111.42	0.001752
1	171.87	104.17	104.13	6.00	3.31	112.09	0.000183
2	172.00	104.58	104.29	4.99	3.39	111.57	0.000349

Résultats complets :

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

3.3 - KNN

In [71]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

In [72]:
```
# Supprimer les lignes avec des valeurs manquantes
billets = billets.dropna()
```

In [73]:
```
billets_clean = billets
```

In [74]:
```
X = billets_clean.drop(columns=['is_genuine'])
y = billets_clean['is_genuine']
```

In [75]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [76]:
```
#Séparer en test
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
```

In [77]:
```
#Créer et entraîner le modèle KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

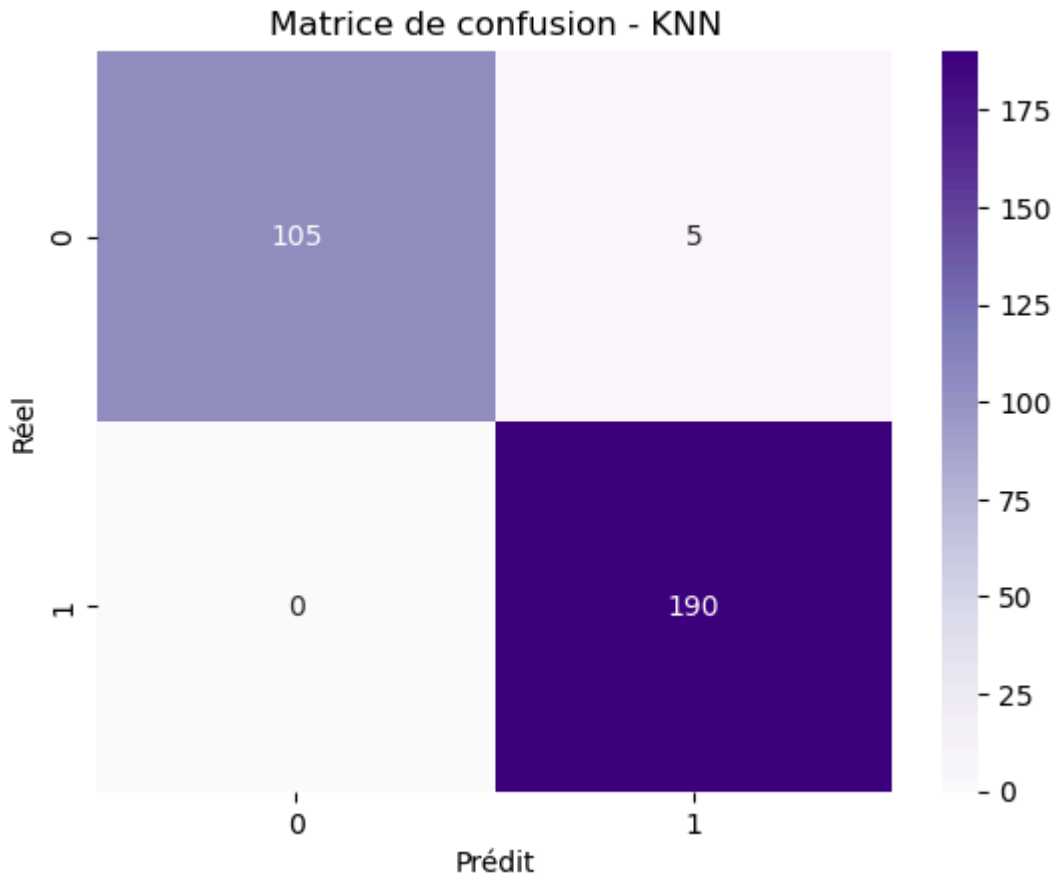
Out[77]:

```
KNeighborsClassifier()
```

In [78]:

```
#Évaluer le modèle
y_pred = knn.predict(X_test)

# Rapport de classification
print(classification_report(y_test, y_pred))

# Matrice de confusion
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Purples
plt.title("Matrice de confusion - KNN")
plt.xlabel("Prédit")
plt.ylabel("Réel")
plt.show()
```

```
              precision    recall  f1-score   support

       False       1.00      0.95      0.98       110
        True       0.97      1.00      0.99       190

    accuracy                           0.98       300
   macro avg       0.99      0.98      0.98       300
weighted avg       0.98      0.98      0.98       300
```



Matrice de confusion du modèle KNN :

Cette matrice évalue la capacité du modèle K-Nearest Neighbors à classer les billets en deux catégories (0 = vrai, 1 = faux).

- 105 vrais billets correctement identifiés comme vrais (case en haut à gauche)
- 190 faux billets correctement identifiés comme faux (case en bas à droite)
- 5 faux billets classés à tort comme vrais (case en haut à droite)
- 0 vrai billet classé à tort comme faux (case en bas à gauche)

Performance : La précision globale est de 98,3 % avec 5 erreurs sur 300 billets.

In [80]:
```
# Chargement du fichier à tester
nouveau_billet = pd.read_csv('billets_production.csv', sep=',', encoding='utf-8'
```

In [81]:
```
#Suppression des Nan
nouveau_billet = nouveau_billet.dropna()
```

In [82]:
```
# Sélectionner uniquement les colonnes utilisées pour la prédiction
colonnes = ['diagonal', 'height_left', 'height_right', 'margin_low', 'margin_up'
X_new = nouveau_billet[colonnes]

# Standardiser les données
nouveau_scaled = scaler.transform(X_new)

# Prédictions
nouveau_billet['is_genuine_pred'] = knn.predict(nouveau_scaled)
nouveau_billet['proba'] = knn.predict_proba(nouveau_scaled)[:, 1]

# Afficher les faux billets
print(nouveau_billet[nouveau_billet['is_genuine_pred'] == 0])
```

```
   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

   is_genuine_pred  proba
0            False    0.0
1            False    0.0
2            False    0.0
```

3.4 - Random Forest

In [84]:
```
# Import des librairies
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import StratifiedKFold, cross_val_score, train_test
import seaborn as sns
import matplotlib.pyplot as plt
```

In [85]:
```
billets_clean
```

Out[85]:

	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length
0	True	171.81	104.86	104.95	4.52	2.89	112.83
1	True	171.46	103.36	103.66	3.77	2.99	113.09
2	True	172.69	104.48	103.50	4.40	2.94	113.16
3	True	171.36	103.91	103.94	3.62	3.01	113.51
4	True	171.73	104.28	103.46	4.04	3.48	112.54
...	...	...	...	...	...	...	...
1495	False	171.75	104.38	104.17	4.42	3.09	111.28
1496	False	172.19	104.63	104.44	5.27	3.37	110.97
1497	False	171.80	104.01	104.12	5.51	3.36	111.95
1498	False	172.06	104.28	104.06	5.17	3.46	112.25
1499	False	171.47	104.15	103.82	4.63	3.37	112.07

1500 rows × 7 columns

In [86]:
```
# Définir X et y
X = billets_clean.drop(columns=['is_genuine'])
y = billets_clean['is_genuine']
```

In [87]:
```
#Standardisation
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Mise en place de la validation croisée

In [89]:
```
# Validation croisée
rf = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [90]:
```
# Définir une validation croisée stratifiée pour garder le même équilibre de cla
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

In [91]:
```
# Calculer les scores d'accuracy sur les 5 folds
scores = cross_val_score(rf, X_scaled, y, cv=cv, scoring='accuracy')
print("Scores de validation croisée :", scores)
print("Moyenne de l'accuracy :", scores.mean())
print("Écart-type :", scores.std())
```

```
Scores de validation croisée : [0.99666667 0.98666667 0.98666667 0.99666667 0.986
66667]
Moyenne de l'accuracy : 0.9906666666666666
Écart-type : 0.00489897948556636
```

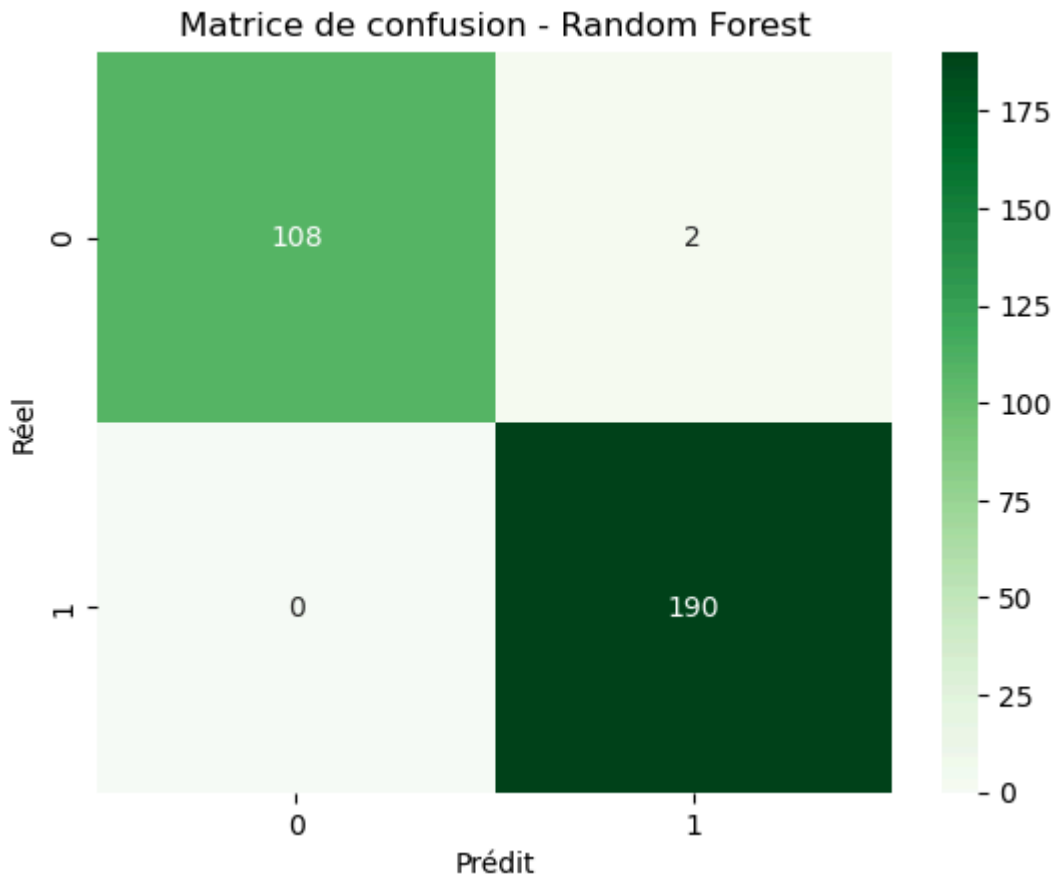
In [92]:
```
#Split final train/test pour évaluation finale
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
```

In [93]:
```
#Entraînement du modèle final sur l'ensemble d'entraînement
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

In [94]:
```
#Évaluation finale
print("\nRapport de classification :")
print(classification_report(y_test, y_pred))
```

```
Rapport de classification :
              precision    recall  f1-score   support

       False       1.00      0.98      0.99       110
        True       0.99      1.00      0.99       190

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

In [95]:
```
#Affichage de la heatmap
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Greens'
plt.title("Matrice de confusion - Random Forest")
plt.xlabel("Prédit")
plt.ylabel("Réel")
plt.show()
```



Matrice de confusion du modèle Random Forest :

Cette matrice évalue la capacité du modèle Random Forest à détecter les billets vrais (0) et faux (1).

- 108 vrais billets correctement identifiés (case en haut à gauche)
- 190 faux billets correctement identifiés (case en bas à droite)
- 2 faux billets classés à tort comme vrais (case en haut à droite)
- 0 vrai billet classé à tort comme faux (case en bas à gauche)

Performance : La précision globale est de 99,3 % avec seulement 2 erreurs, le Random Forest surpasse les autres c'est le modèle qu'on va retenir pour l'application car il est le plus précis.

In [97]:
```
# Charger le fichier à tester
nouveau_billet = pd.read_csv('billets_production.csv', sep=',', encoding='utf-8'
```

In [98]:
```
#Affichage du df à tester
nouveau_billet
```

Out[98]:

	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

In [99]:
```
# Colonnes utilisées
colonnes = ['diagonal', 'height_left', 'height_right', 'margin_low', 'margin_up'

# Vérification que toutes les colonnes existent
missing_cols = [col for col in colonnes if col not in nouveau_billet.columns]
if missing_cols:
    raise ValueError(f"Colonnes manquantes dans le DataFrame : {missing_cols}")

X_new = nouveau_billet[colonnes].copy()

# Standardiser
X_scaled_new = scaler.transform(X_new)

# Prédictions
nouveau_billet['is_genuine_pred'] = rf.predict(X_scaled_new)
nouveau_billet['proba'] = rf.predict_proba(X_scaled_new)[:, 1]
```

In [100…
```
# Colonnes utilisées
colonnes = ['diagonal', 'height_left', 'height_right', 'margin_low', 'margin_up'
X_new = nouveau_billet[colonnes]

# Standardiser
X_scaled_new = scaler.transform(X_new)

# Prédictions
nouveau_billet['is_genuine_pred'] = rf.predict(X_scaled_new)
nouveau_billet['proba'] = rf.predict_proba(X_scaled_new)[:, 1]
```

In [101…

```
# Faux billets
faux_billets = nouveau_billet[nouveau_billet['is_genuine_pred'] == False]
print("Billets prédits comme faux :")
display(faux_billets[colonnes + ['proba']])

# Vrais billets
vrais_billets = nouveau_billet[nouveau_billet['is_genuine_pred'] == True]
print("Billets prédits comme authentiques :")
display(vrais_billets[colonnes + ['proba']])
```

```
Billets prédits comme faux :
```

	diagonal	height_left	height_right	margin_low	margin_up	length	proba
0	171.76	104.01	103.54	5.21	3.30	111.42	0.01
1	171.87	104.17	104.13	6.00	3.31	112.09	0.00
2	172.00	104.58	104.29	4.99	3.39	111.57	0.00

```
Billets prédits comme authentiques :
```

	diagonal	height_left	height_right	margin_low	margin_up	length	proba
3	172.49	104.55	104.34	4.44	3.03	113.20	0.99
4	171.65	103.63	103.56	3.77	3.16	113.33	1.00

Exportation du fichier billets_clean

In [175…

```
# Export du fichier billets_clean
billets_clean.to_csv("billets_clean.csv", index=False, encoding='utf-8')
```