DÉTECTEZ DES FAUX BILLETS AVEC R OU PYTHON

Par Claude Olukoya

MA MISSION

Mon entreprise vient de décrocher un contrat avec l'Organisation nationale de lutte contre le faux-monnayage (ONCFM) et elle souhaite m'y envoyer en tant que senior data analyst pour cette mission.

Ils mettent en place des méthodes d'identification des faux billets en euros pour lutter contre la contrefaçon et construire une application de machine learning. Celle-ci leur permettra, après avoir scanné des billets (longueur, hauteur, largeur, etc.), de faire une prédiction sur la nature des billets (vrai billet ou faux billet). 4 algorithms à tester :

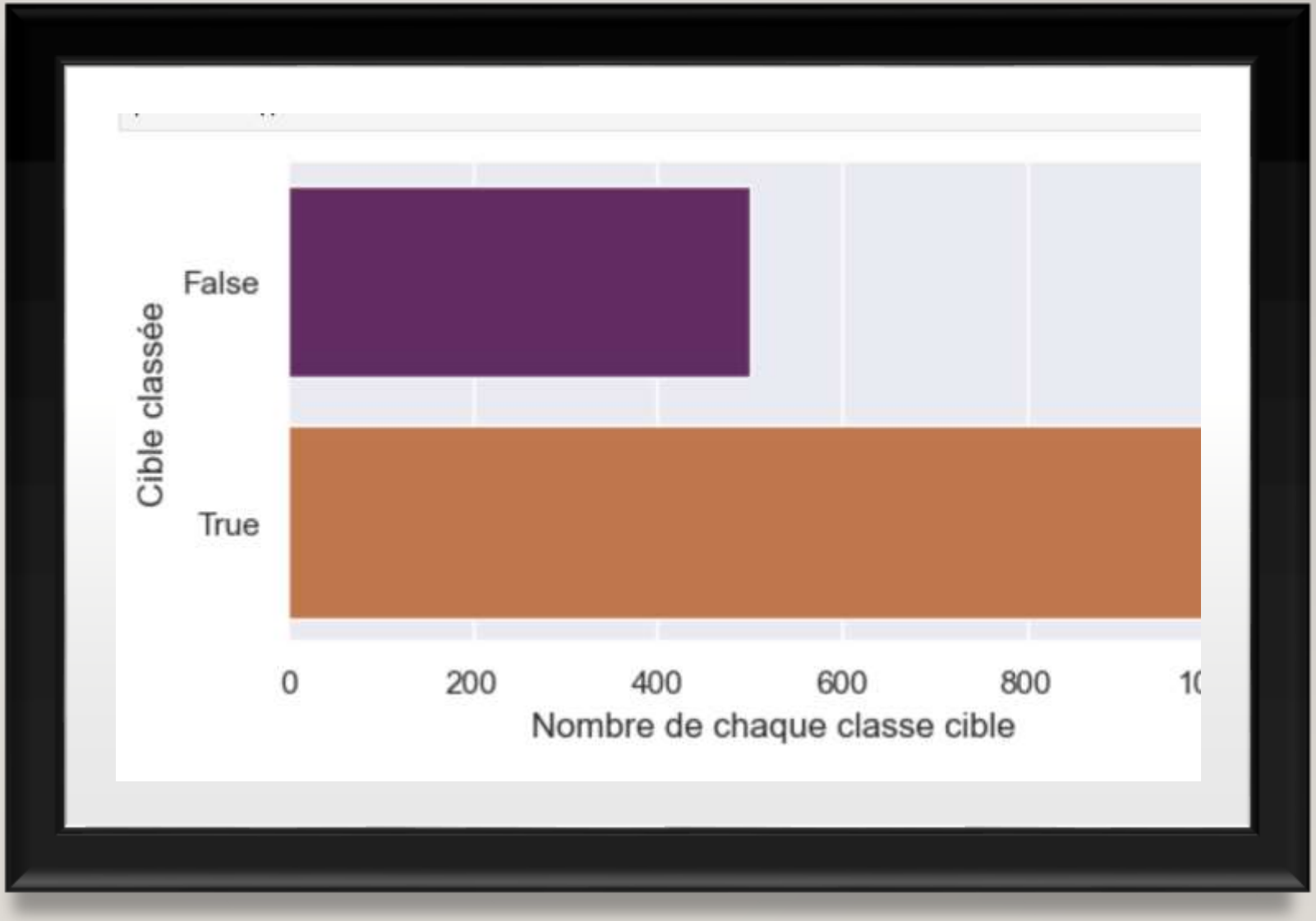
1. K-Means
2. Régression logistique
3. KNN
4. Random Forest

LE FICHIER (CSV) DE DÉPART

Billets.csv

PARTIE I : NETTOYAGE & EXPLORATION

Nous constatons qu'il y a 1,000 vrais billets et 500 faux billets



- En utilisant le package *missingno* on dessine une matrice de nullité. Nous constatons qu'il y a 37 valeurs manquantes dans la colonne margin_low.

IMPUTATION DES VALEURS MANQUANTES AVEC 'OLS'

Étape 1: *Backward elimination (Elimination inverse)* : utilisée en apprentissage automatique pour identifier le meilleur sous-ensemble de caractéristiques parmi un ensemble donné. Elle consiste à supprimer de manière itérative les caractéristiques qui ne sont pas prédictives de la variable cible ou qui ont le moins de pouvoir prédictif.

Étape 2: Une fois qu'on a choisi le nombre de features : *is_genuine & margin_up,* on fait la prédiction à l'aide de OLS (Ordinary Least Squares) nous aide à trouver la ligne la plus adaptée qui prédit le résultat en fonction des données dont nous disposons.

IMPUTATION DES VALEURS MANQUANTES

La méthode manuelle

Étape 1 : Séparer les valeurs nulles du DataFrame et les considérer comme test_df

Étape 2 : Retirer les valeurs nulles du DataFrame et les considérer comme train_data

Étape 3 : Créer X_train et y_train à partir du train_data

Étape 4 : Construire le modèle de Régression Linéaire

Étape 5 : Créer le X_train à partir test_df

Étape 6 : Appliquer le modèle sur X_test du test_df et formuler des prédictions

Étape 7 : Remplacer les valeurs manquantes par des valeurs prédites

[ ]:

DIFFERENCE ENTRE LES VRAIS ET FAUX BILLETS

	diagonale	height_left	height_right	margin_low	margin_up	length
Les différences entre vrais & faux billets	0.08271	-0.21727	-0.30512	-1.006586	-0.26659	1.42647

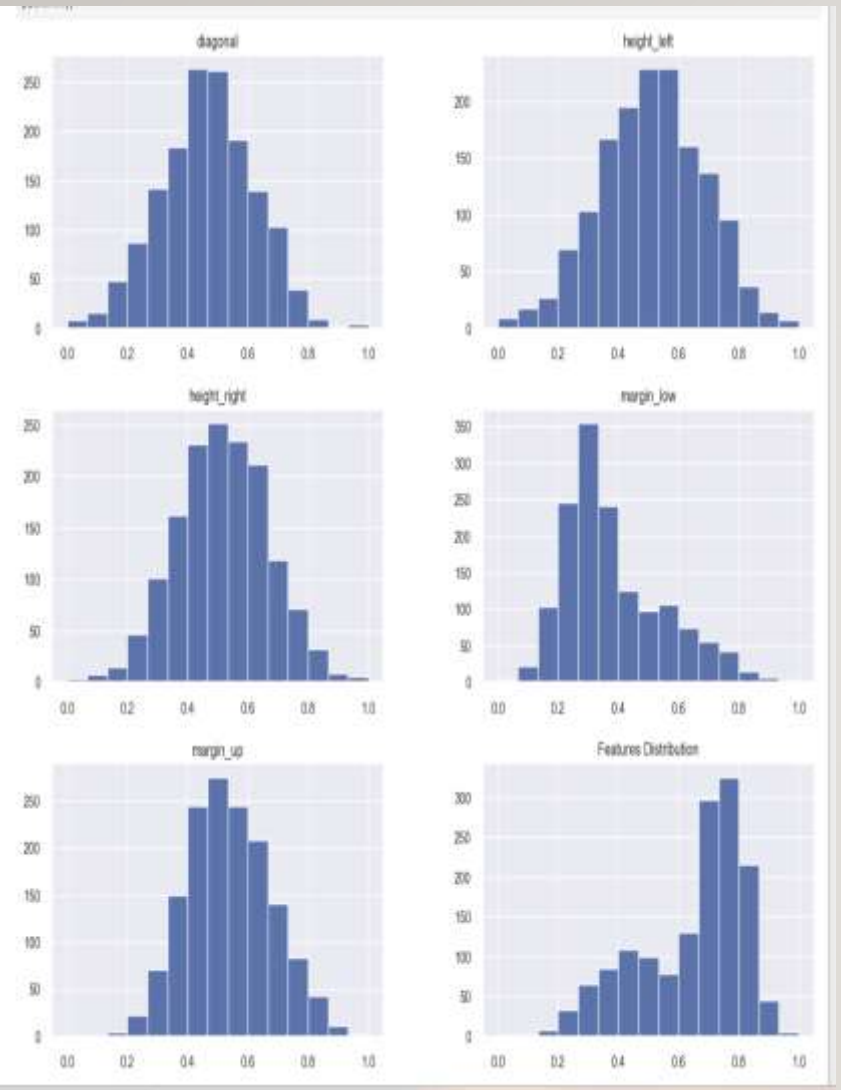
On peut en déduire que :

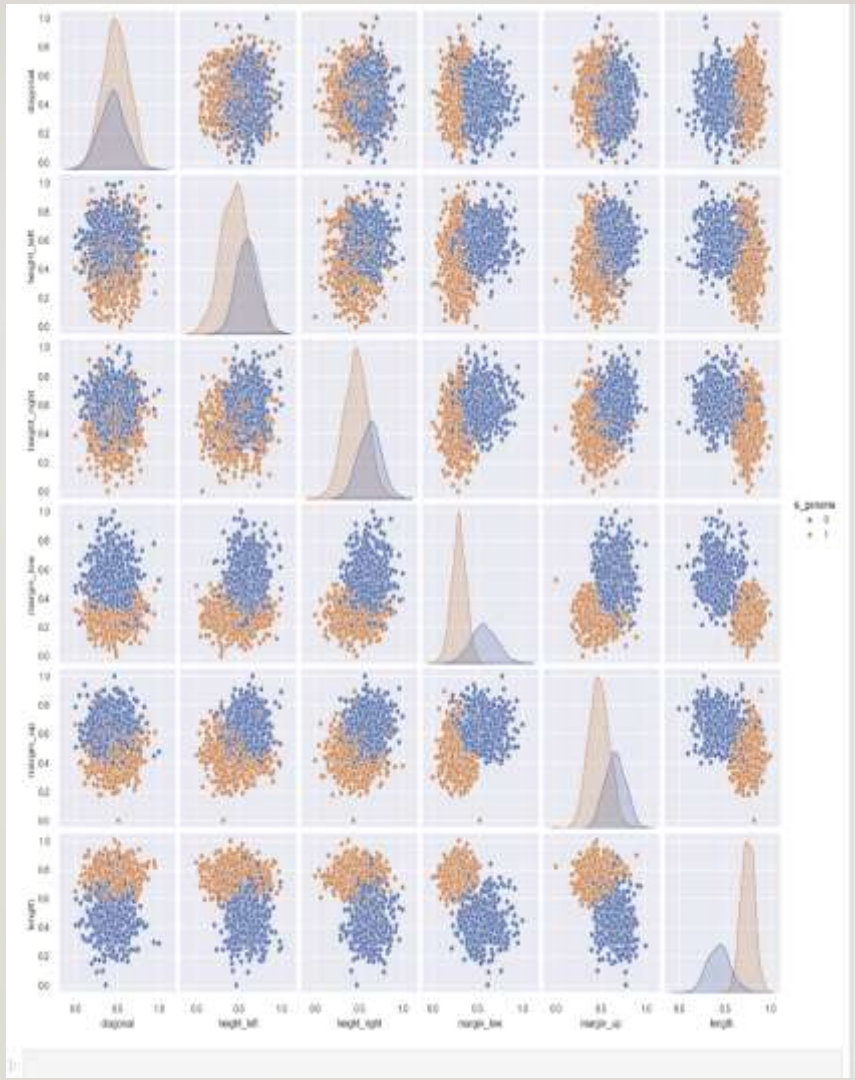
Les vrais billets ont une diagonale légèrement plus importante

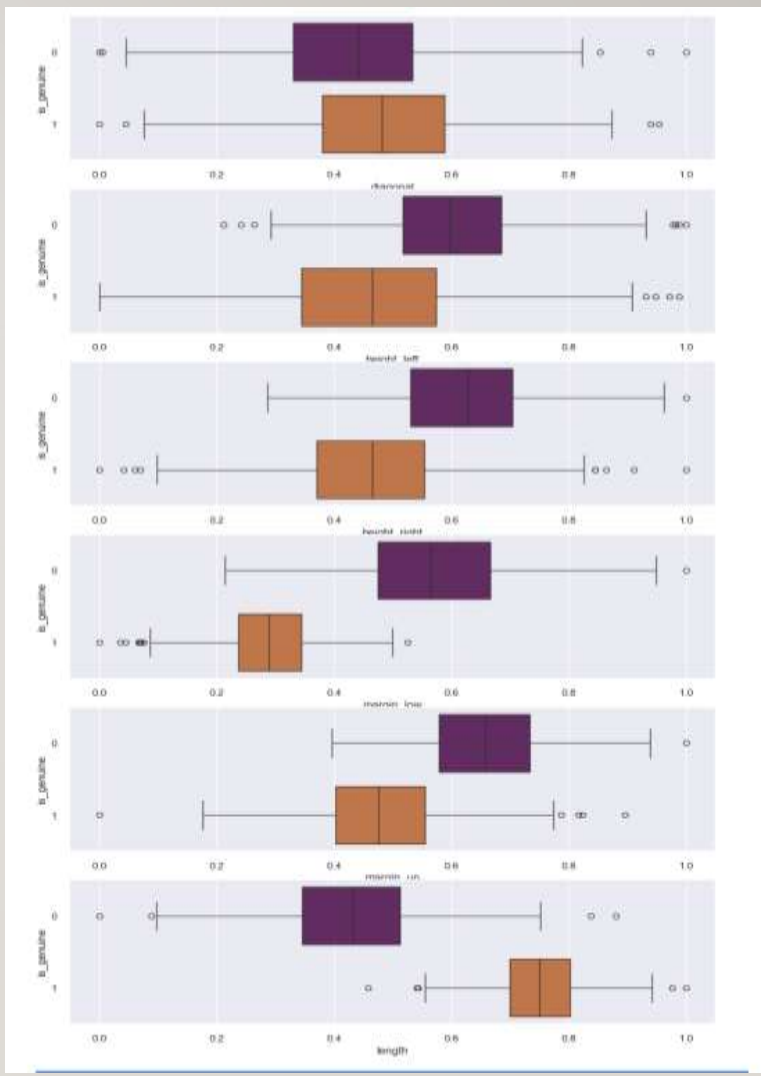
La hauteur et la marge des faux billets sont lègèrement plus importantes

Les vrais billets sont plus grands en largeur

LES DISTRIBUTIONS







LES HYPOTHÈSES DE VALIDITÉ À VÉRIFIER

1. **COLINÉARITE DES VARIABLES :**

signifie qu'une forte corrélation existe entre elles, ce qui rend difficile, voire impossible, l'estimation fiable de leurs coefficients de régression individuels

Nous constatons que les variables *length* et *is_genuine* ont une très forte corrélation.

2. **HOMOSCÉDASTICITÉ :**

fait référence à une condition dans laquelle la variance du résidu, ou terme d'erreur, dans un modèle de régression est constante.

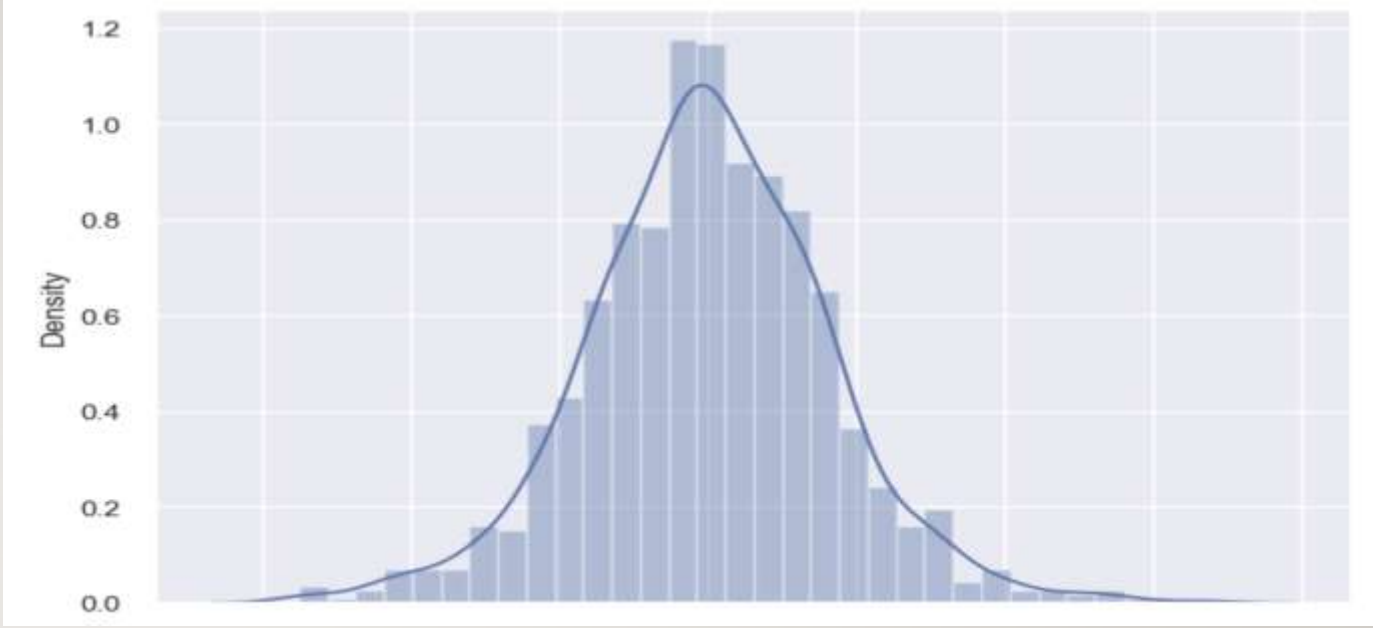
Avec une statistique de test de 29.98 et une valeur p de 9.01^{e}-24, l'assumption n'est pas validée.

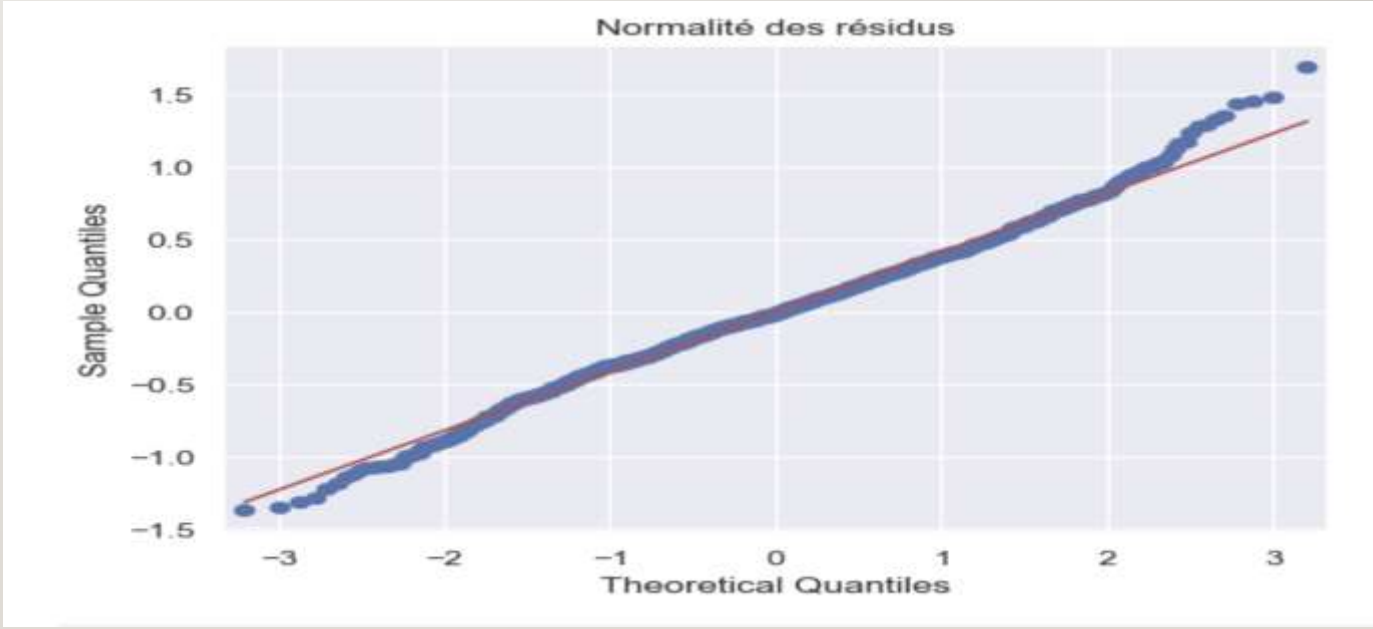
```
]: import numpy as np
   from scipy.stats import levene

   # Récupération des résidus
   residuals = ols_model.resid

   # la valeur prédite
   predicted = ols_model.fittedvalues

   # Division des résidus en groupes en fonction de la variable prédite
   groups = pd.qcut(predicted, q=5, labels=False)

   # le test de Levene sur les résidus groupés
   levene_test = levene(*[residuals[groups == i] for i in np.unique(groups)], center='mean')

   #  la statistique de test et la valeur p
   test_statistic = levene_test.statistic
   p_value = levene_test.pvalue

   # On affiche les résultats
   print("Test de Levene:")
   print("Statistique de test:", test_statistic)
   print("Valeur p:", p_value)

   Test de Levene:
   Statistique de test: 29.983642592506712
   Valeur p: 9.014373222445792e-24
```

3. NORMALITÉ DES RÉSIDUS

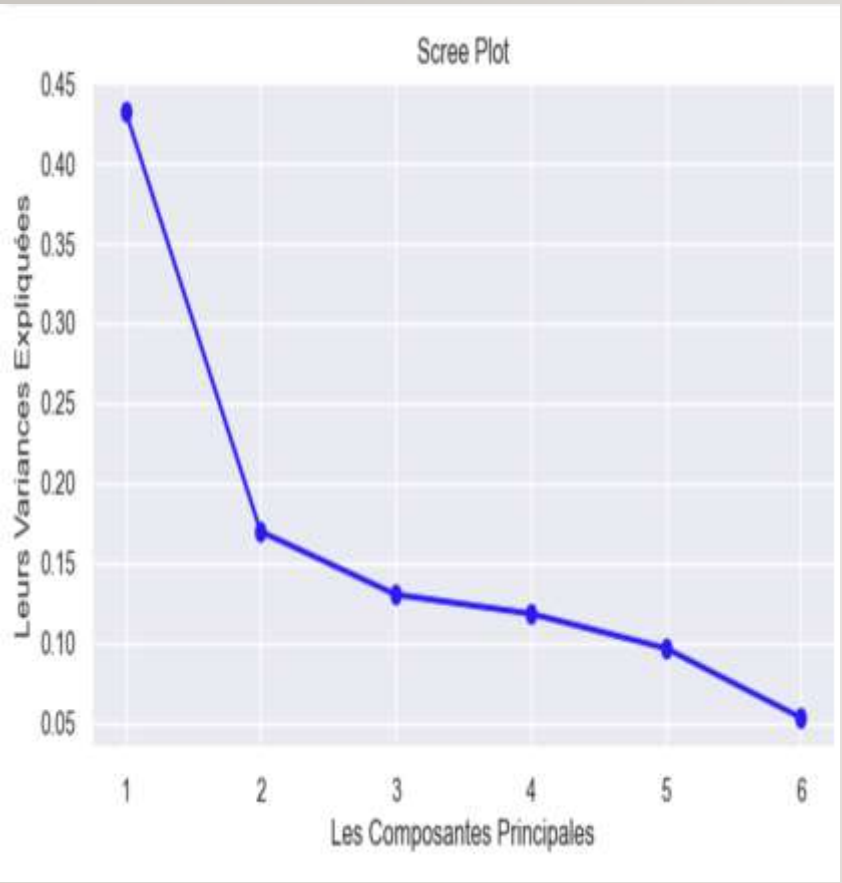
La normalité est l'hypothèse selon laquelle les résidus sous-jacents sont normalement distribués, ou approximativement ainsi.

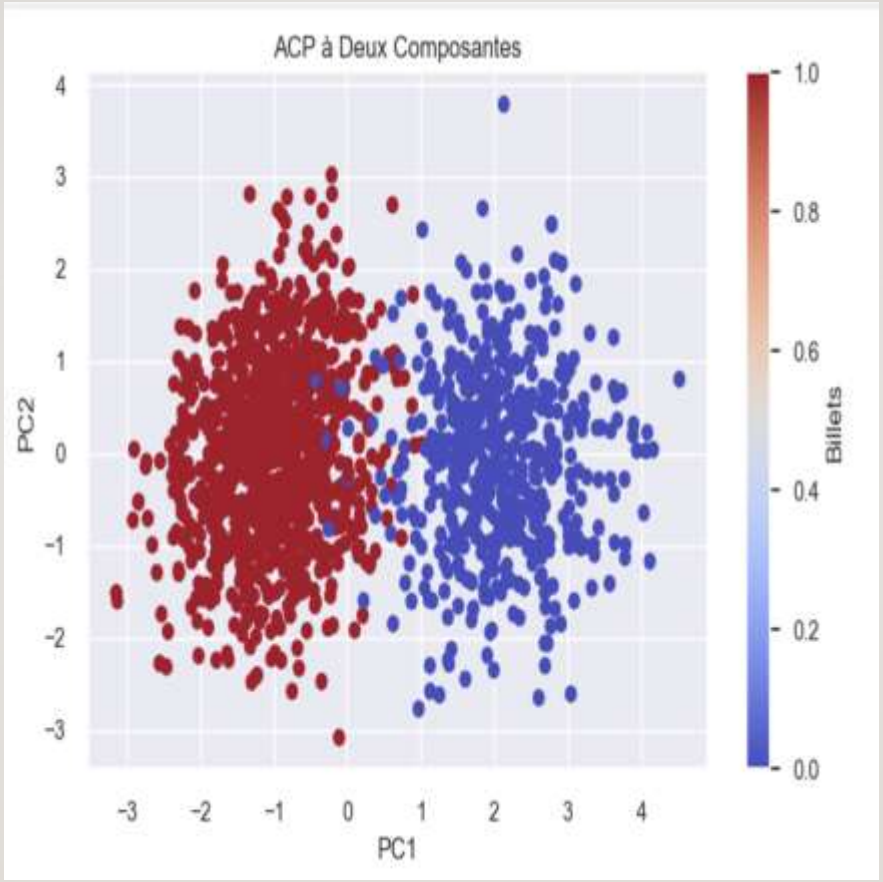
Le distplot montre une distribution normale mais le test Shapiro ne valide pas cette assomption.

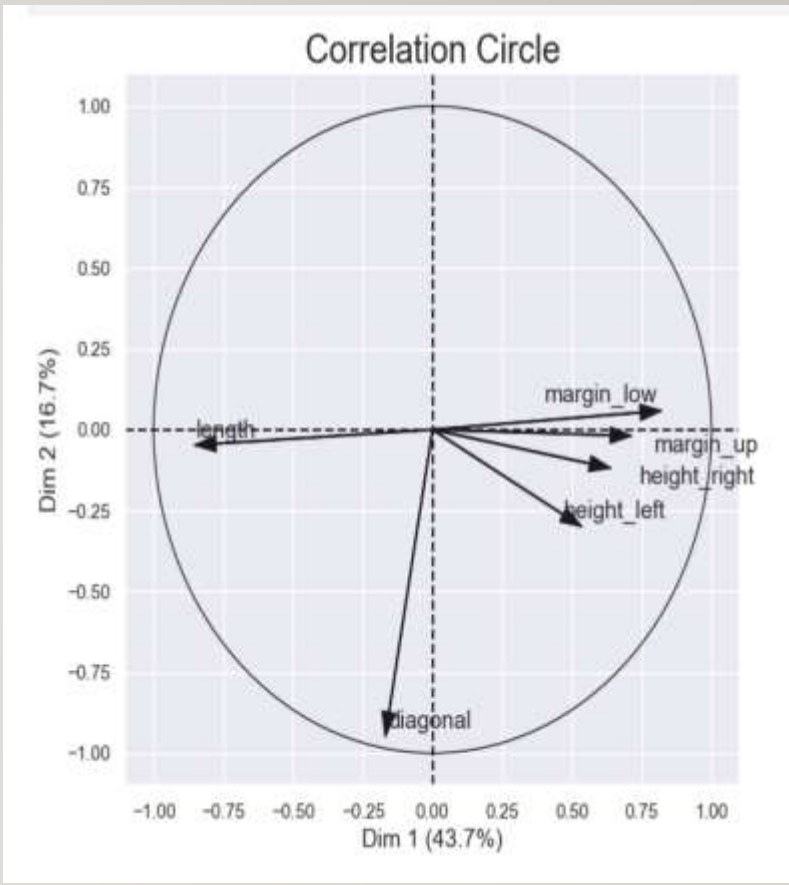




ACP





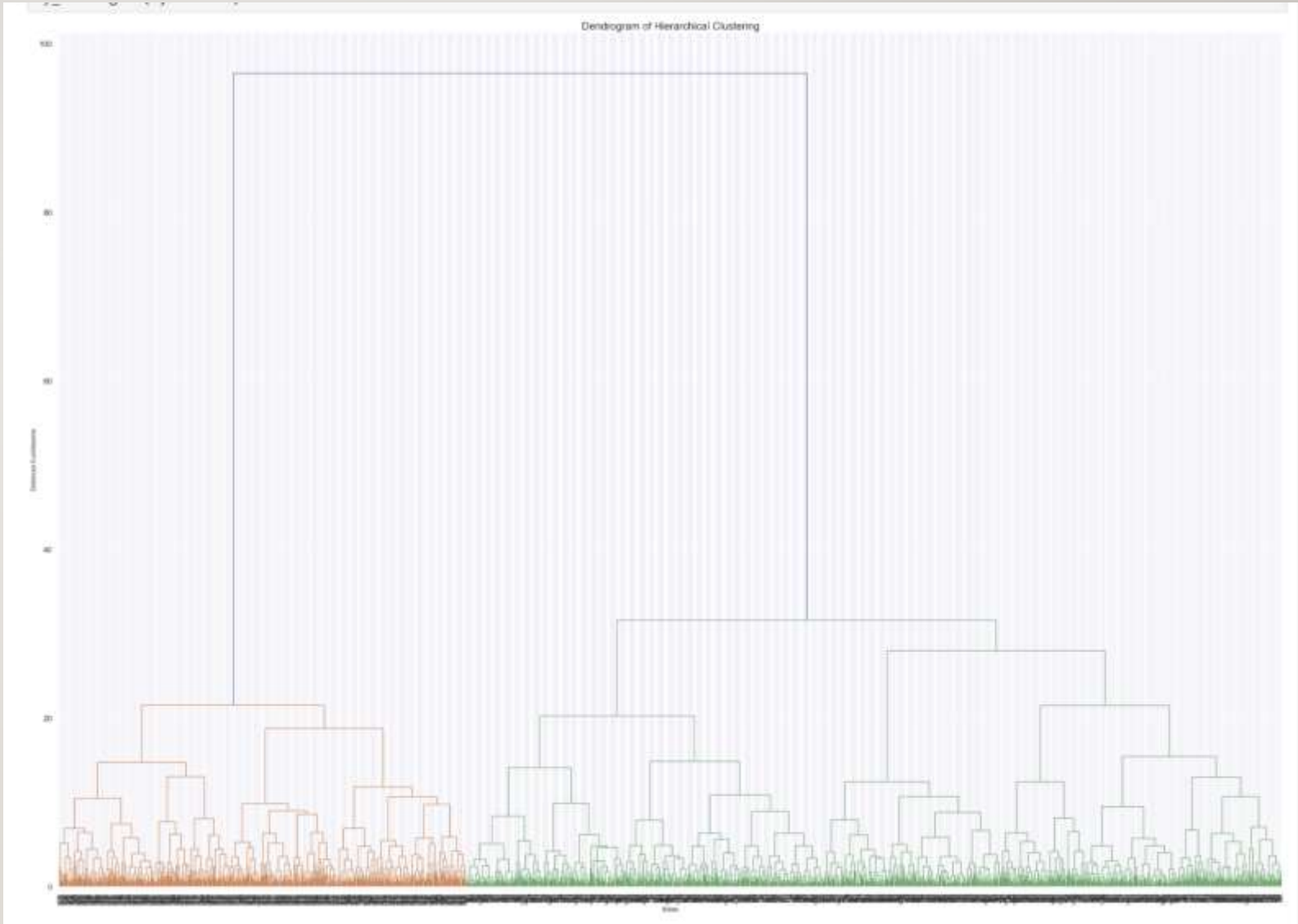


Du scree Plot, on prend les 2 premières composantes principales (60% de la variance totale)

Sur la projection sur les individus PC1 et PC2, on voit clairement deux clusters qui expliquent les vrais et faux billets.

Classifcation Ascendante Hierarchique (CAH)

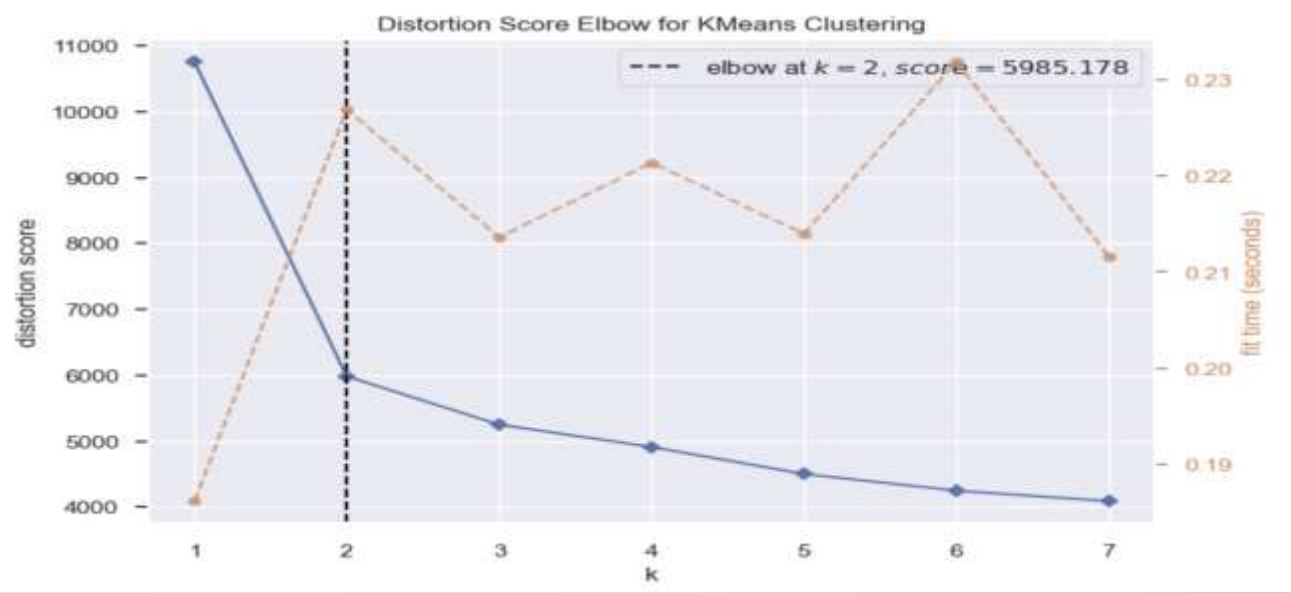
J'ai essayé une autre méthode de clustering, en l'occurrence CAH pour voir les clusters. On voit clairement 2 clusters dans ce dendogramme

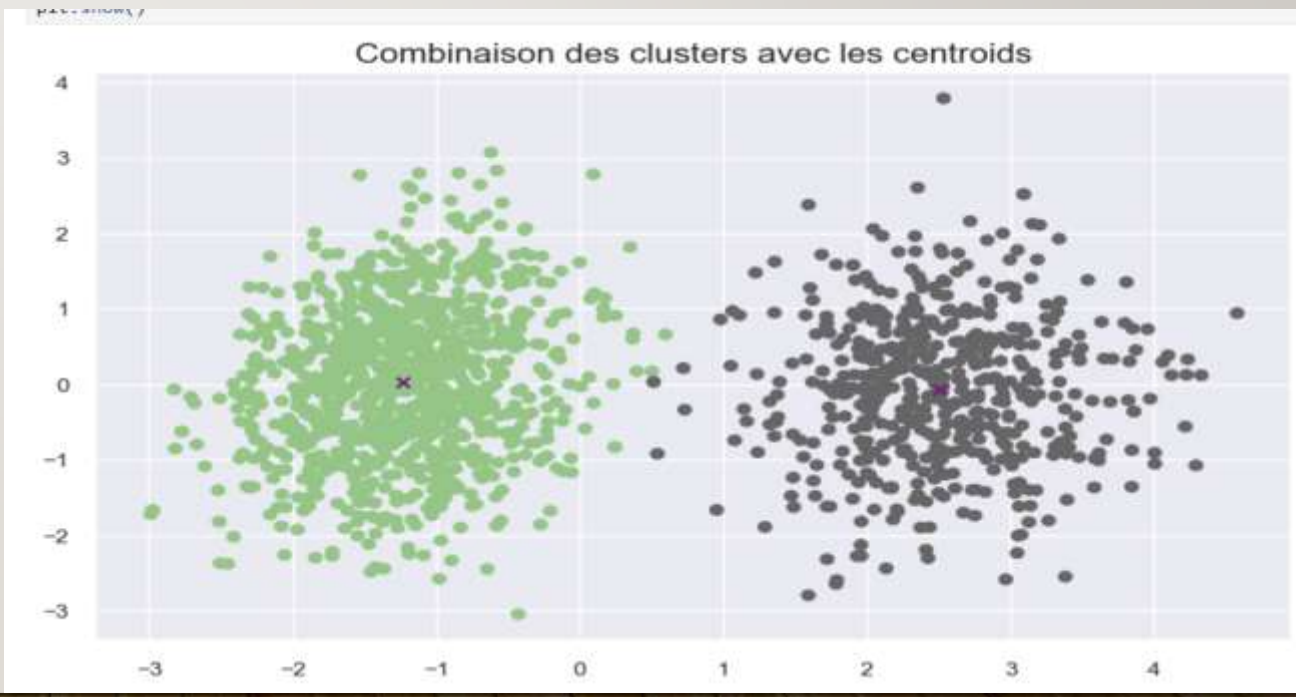


K-Means

De la 2ème méthode de clustering K-Means, on en déduit que le nombe optimal de k = 2 avec la méthode du Coude.

Après avoir entraîné le modèle sur les données, on voit clairement les deux clusters avec les centroides qui distinguent les vrais et faux billets.





PRÉDICTION DES BILLETS : 4 ALGORITHMES À TESTER

I. LA REGRESSION LOGISTIQUE

La régression logistique est un algorithme d'apprentissage automatique supervisé qui accomplit des tâches de classification binaire en prédisant la probabilité d'un résultat, d'un événement ou d'une observation.

- Vérifier les 6 assomptions avant d'entraîner le modèle
- Utilisation de DecisionTreeClassifier pour sélectionner les features les plus pertinents
- L'évaluation de la performance donne un F1 score de 98.99%

```
LogisticRegression()

[824]: # On enregistre les prédictions dans une variable y_pred

       y_pred = log_reg.predict(X_test)

[825]: result = log_reg.score(X_train, y_train)

[826]: # Afficher le score du model

       result = round(result * 100, 2)
       print("Le score du model est : ", "{}".format(result),"%")

       Le score du model est :  99.08 %

[827]: # On trace une matrice de confusion pour comparer la précision entre les vraies valeurs avec les valeurs prédites

       cm3 = confusion_matrix(y_test, y_pred)

[828]: cm3

[828]: array([[ 99,   3],
              [  1, 197]], dtype=int64)

[829]: plt.figure(figsize=(6,4))
       sns.heatmap(cm3,annot=True,linewidths=1, fmt = 'd')
       plt.xlabel('Prédit')
       plt.ylabel('Vrai')
       plt.show()
```

2. K-MEANS

Bien que K-means lui-même ne prédise pas directement, il peut s'agir d'une étape puissante de prétraitement ou d'ingénierie des fonctionnalités qui améliore les modèles prédictifs en : Identifiant des clusters ou des segments de clientèle significatifs.

- On choist le nombre optimal de K clusters (2) avec la méthode Elbow

- On crée une colonne dans notre table en ajoutant les labels calculé par le model

- Le f1 score donne 98.49%

```
KMeans(max_iter=100, n_clusters=2, n_init=1, random_state=2)
```

```
# On réalise notre prediction que l'on va stocker dans une variable
prediction = km.predict(X_test)
```

```
# On crée une colonne dans notre table en ajoutant les labels calculé par le model
X_test["cluster"] = prediction
```

```
X_test.head()
```

	diagonal	height_left	height_right	length	margin_low	margin_up
471	172.14	103.91	103.65	113.54	4.190000	3.23
9	172.47	104.07	104.02	113.45	4.040000	3.25
1499	171.90	104.28	104.20	111.49	5.228983	3.24
54	172.02	103.58	103.80	112.40	3.950000	3.46
1411	171.74	104.32	103.64	111.56	5.470000	3.28

```
# On trace une matrice de confusion
cm1 = confusion_matrix(y_test, X_test["cluster"])
```

```
cm1
```

```
array([[ 98,   4],
       [  2, 196]], dtype=int64)
```

```
plt.figure(figsize=(6,4))
sns.heatmap(cm1, annot=True,linewidths=1, fmt = 'd')
plt.xlabel('Prédit')
plt.ylabel('Vrai')
plt.show()
```

3. K-Nearest Kneighbors Classifier (KNN)

L'algorithme des k-plus proches voisins (KNN) est un classificateur d'apprentissage supervisé non paramétrique, qui utilise la proximité pour effectuer des classifications ou des prédictions sur le regroupement d'un point de données individuel.

- On partitionne notre dataset en X-train, X-test
- On entraîne notre modèle
- On prédit X_test
- Notre modèle nous donne un score de 98.99%

```
KNeighborsClassifier(n_neighbors=3)
```

```
y_pred = knn.predict(X_test)
```

```
print(y_pred)
```

```
[1 1 0 1 0 0 1 1 1 0 0 0 0 1 1 1 0 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 0 1 0 0 1
 1 1 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 1 1 0 1 0 1 1 1 0
 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 1 0 1 1 1 1 0 1
 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1
 0 1 1 0 1 1 1 1 0 1 1 1 0 1 1 0 1 1 0 0 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 0 1 1 1
 1 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1
 1 1 1 1 1 0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 0 1 0 1 0 1 1 0 0 0 1 0 1
 1 1 1 1 1 0 1 0 1 1 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 1 1 1 0 1
 1 0 0 1]
```

```
res = knn.score(X_test, y_test)
```

```
res = round(res * 100, 2)
print("Le score du model est : ", "{}".format(res),"%")
```

```
Le score du model est :  98.67 %
```

```
# Matrice de Confusion

cm_ = confusion_matrix(y_test, y_pred)
```

```
cm_
```

```
array([[ 99,   3],
       [  1, 197]], dtype=int64)
```

```
plt.figure(figsize=(6,4))
sns.heatmap(cm_,annot=True,linewidths=1, fmt = 'd')
plt.xlabel('Prédit')
plt.ylabel('Vrai')
plt.show()
```

4. RANDOM FOREST

Un algorithme qui génère une 'forêt d'arbres'. Il les combine ensuite pour éviter le surapprentissage et produire des prédictions plus précises.

- On partitionne notre dataset en X-train, X-test
- On entraîne notre modèle
- On prédit X_test
- Notre modèle nous donne un score de 99.24%