Détectez des faux billets avec R ou Python

Samira MAHJOUB

SOMMAIRE

- Présentation du projet
- Exploration des données
- Corrélation entre les variables
- Comparaison de la distribution des variables
- Imputation des valeurs manquantes: Régression linéaire multiple
- Évaluation du modèle de régression linéaire multiple("Analyse des résidus")
- Vérification de la cohérence des résultats obtenus

- Modélisations: test des modèles de classifications

  - ACP
  - Méthode des K-Means (clustering non supervisé)
  - Régression logistique (classification supervisé)
  - Méthode du K-NN (le plus proche voisin)
  - Méthode de Randomforest

- Comparaison des performances des modèles

- Application finale

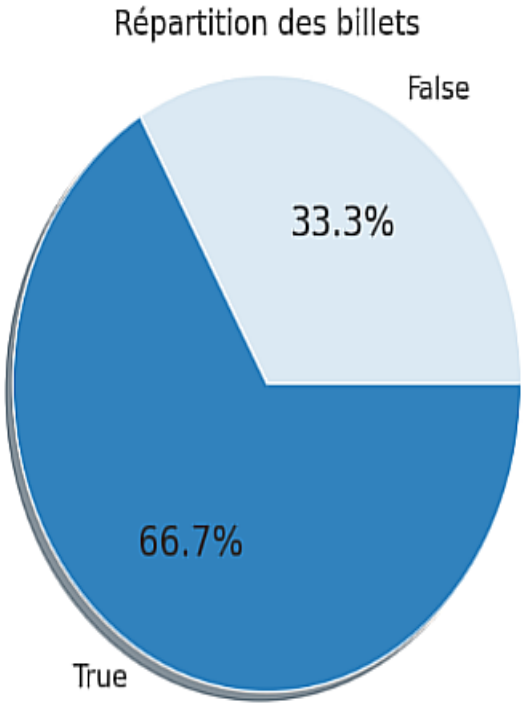
Présentation du projet

- ONCFM souhaite mettre en place un algorithme qui soit capable de différencier automatiquement les vrais des faux billets.

- Le projet consiste à développer un modèle analytique pour l'ONCFM permettant de distinguer les vrais billets d'euros des faux, en se basant sur des caractéristiques quantifiables des billets.

Exploration des données

is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length
False	500	500	500	492	500	500
True	1000	1000	1000	971	1000	1000




```
(decrire_dataframe(billets).reset_index().style
    .bar(axis='index', subset=['count'], color='green')
    .applymap(is_zero, subset=['count', 'unique'], color='pink'))
```

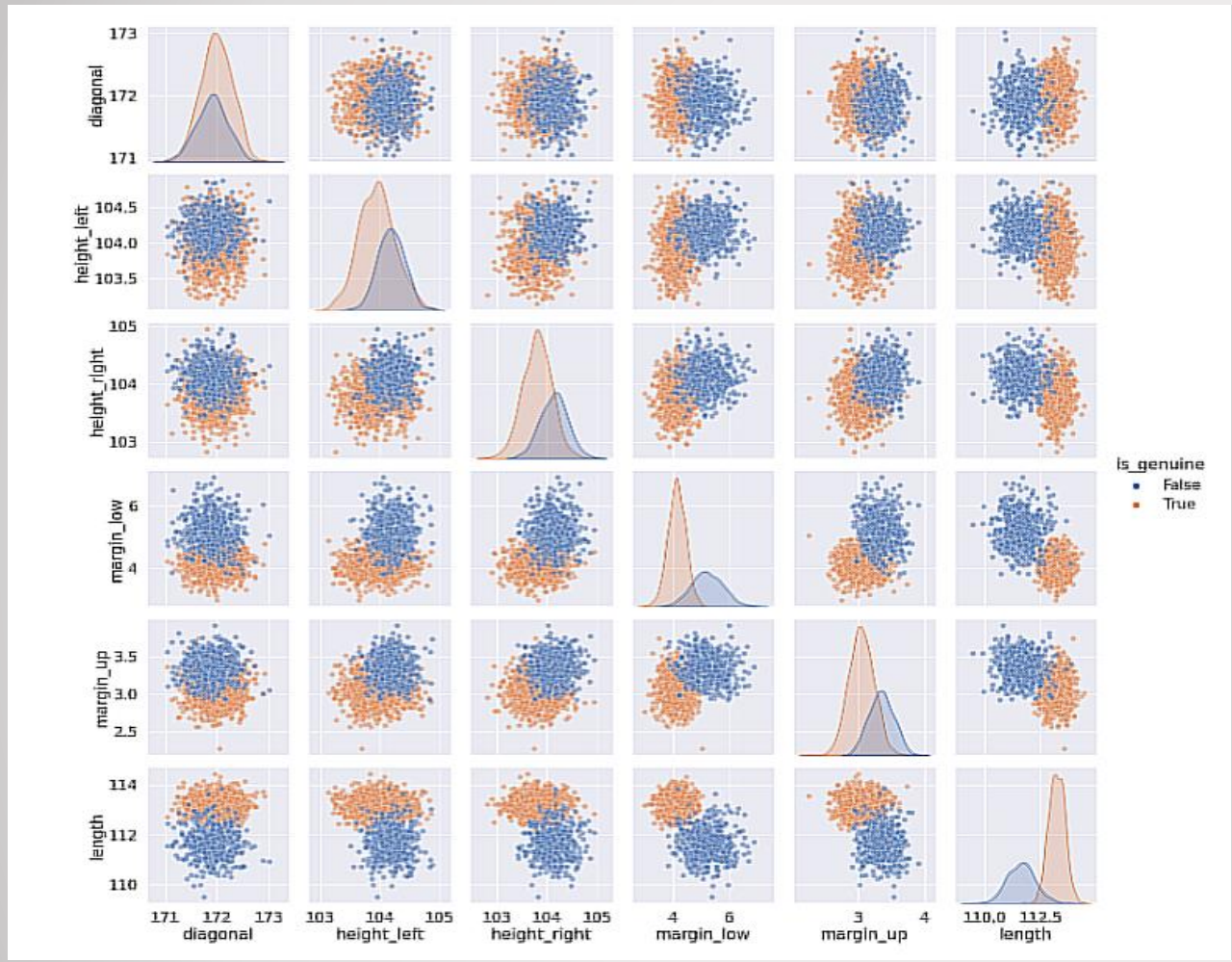
	column	count	unique	dtype	count_null	null%	duplicated
0	is_genuine	1500	2	bool	0	0.000000	0
1	diagonal	1500	159	float64	0	0.000000	0
2	height_left	1500	155	float64	0	0.000000	0
3	height_right	1500	170	float64	0	0.000000	0
4	margin_low	1463	285	float64	37	2.470000	0
5	margin_up	1500	123	float64	0	0.000000	0
6	length	1500	336	float64	0	0.000000	0

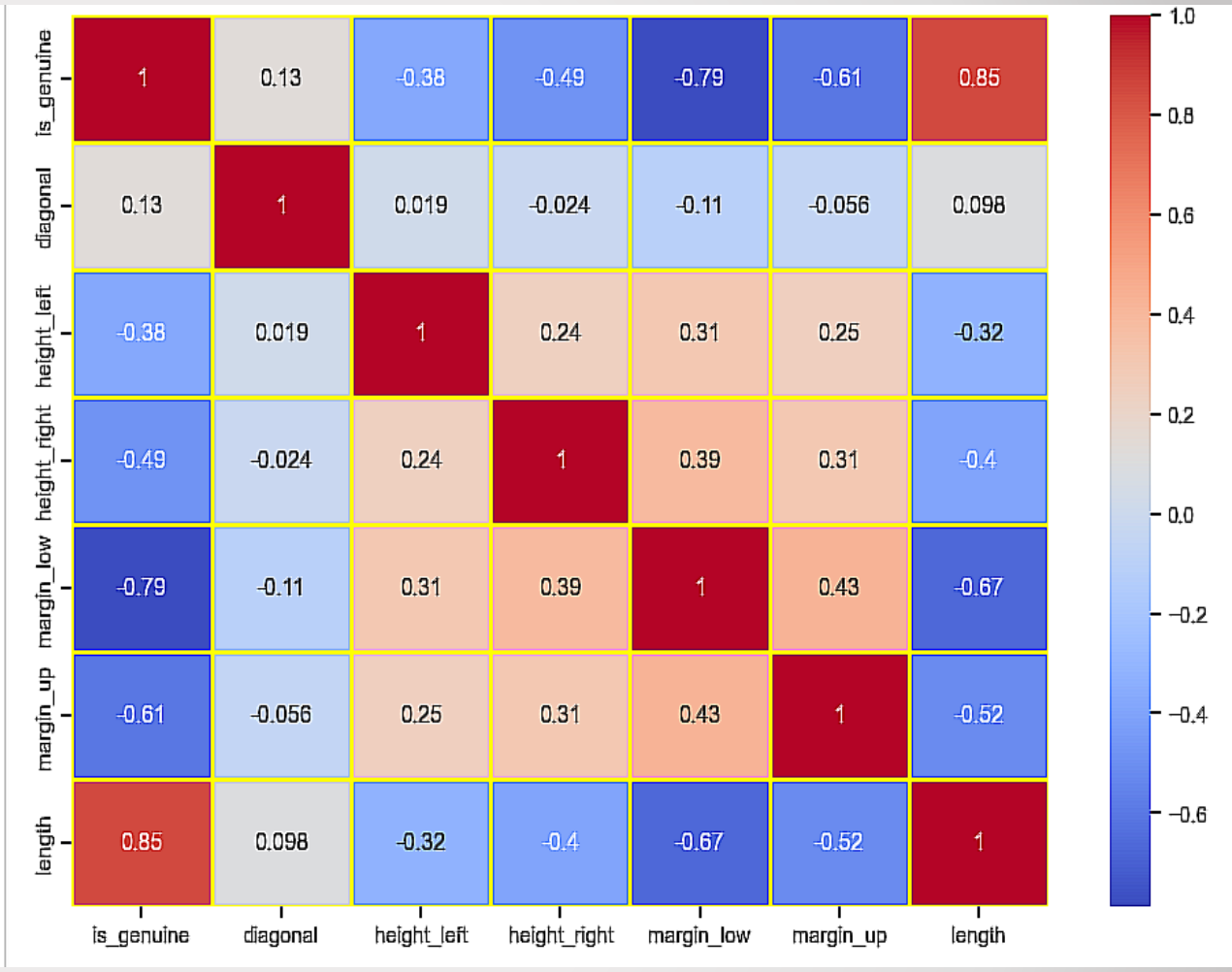
```
#on s'interesse à la colonne "is_genuine"
#Verifier le nombre de billets vrais et faux
billets['is_genuine'].value_counts()
```

```
True     1000
False     500
Name: is_genuine, dtype: int64
```

- Ces données contiennent les informations de 1500 billets de banque, dont 1000 sont authentiques(True) et 500 billets sont faux (False).
- **37 valeurs manquantes** sur la variable **margin_low.**
- L'objectif est de compléter ces valeurs manquantes

Corrélation entre les variables



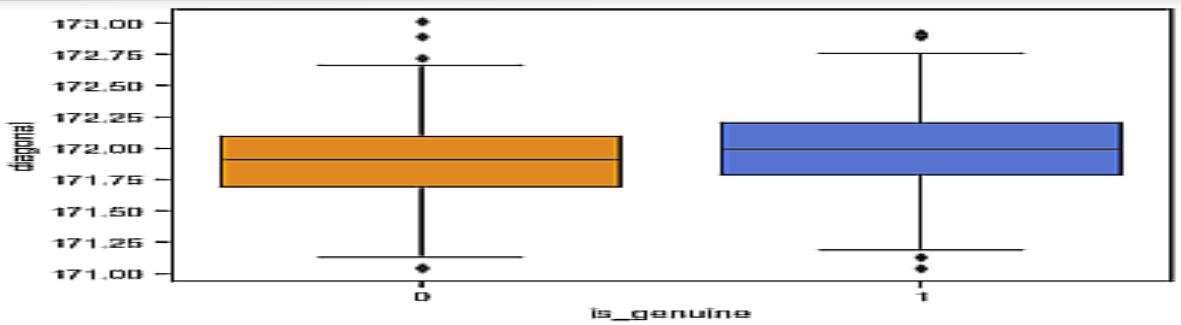


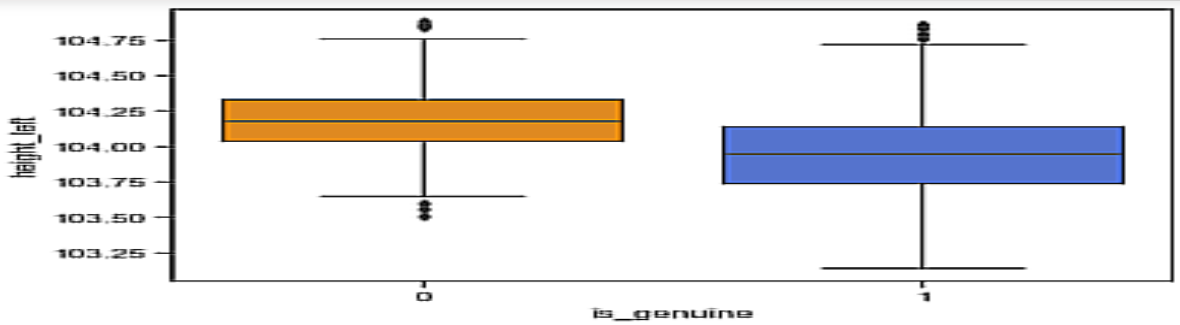
La heatmap nous montre que " **Margin_low**" est bien corrélée négativement avec "**is_genuine**" (-0,79) et positivement avec "margin_up", "height_right" et "height_left".

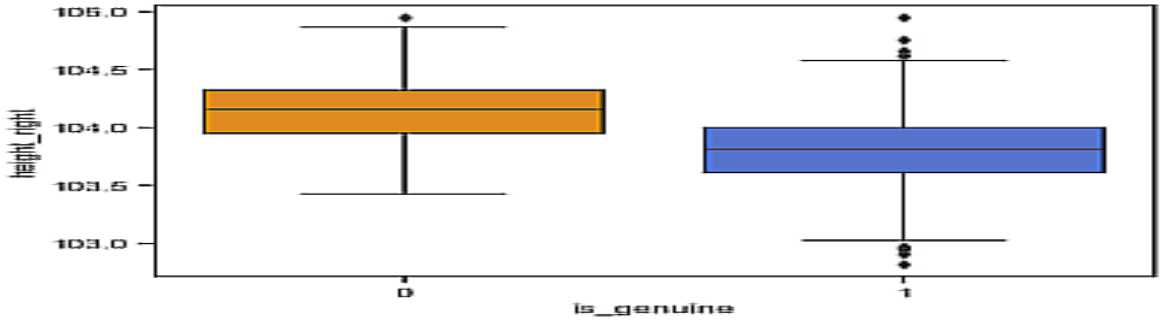
- La variable la plus corrélée est "**lenght**" et "**is_genuine**« **(coeff =0,85)**.

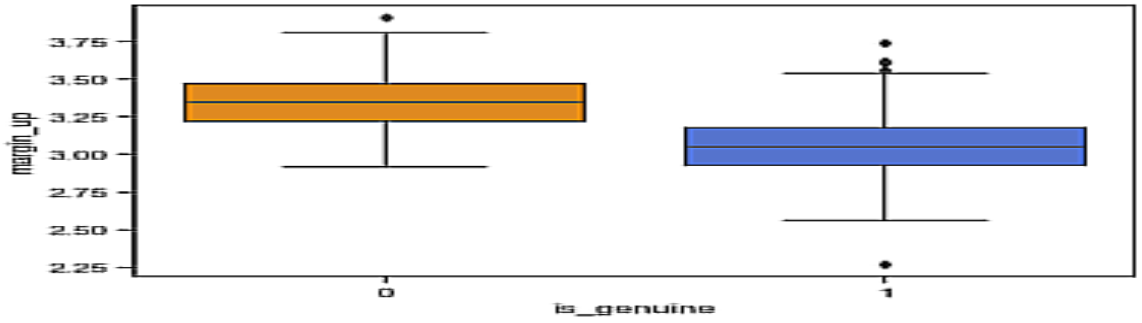
- Pour rappel, plus la corrélation est marquée, plus le résultat du coefficient de corrélation tend vers 1 ou -1. Plus il tend vers 0, moins les variables sont corrélées.

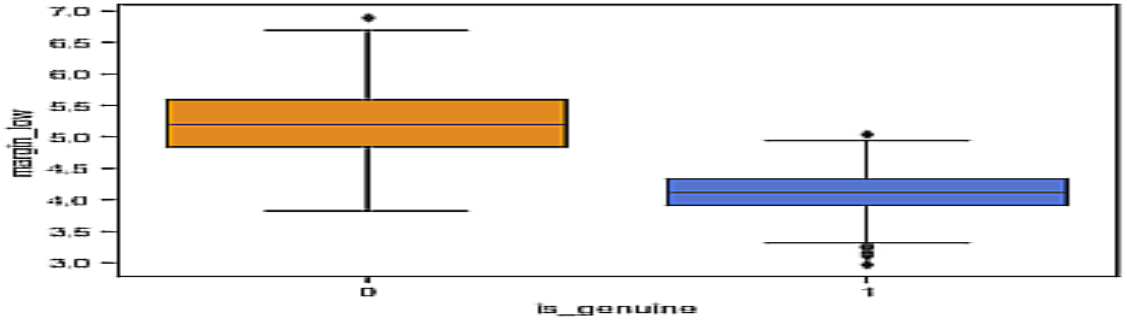
Comparaison de la distribution des variables

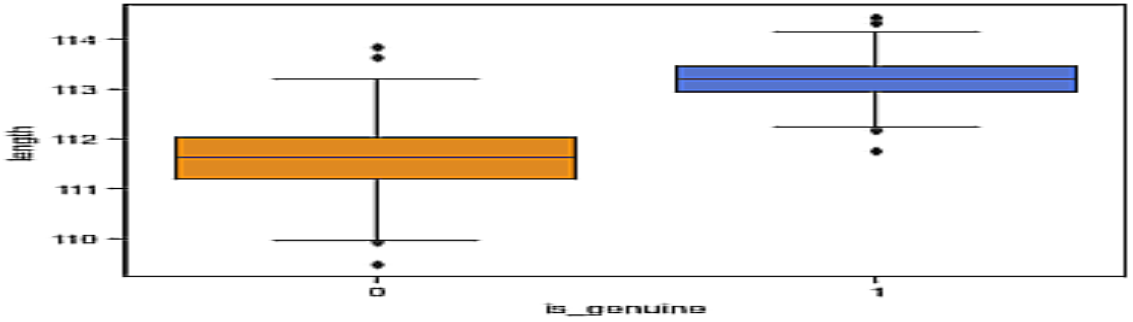












Les boxplots montrent que les vrais billets ont des dimensions généralement plus grandes et des marges inférieures plus petites par rapport aux faux billets. Ces caractéristiques peuvent être clés pour distinguer l'authenticité des billets.

Imputation des valeurs manquantes: Régression linéaire multiple

➢ **Définition**

- **La régression linéaire multiple est une technique statistique qui permet d'examiner la relation entre une variable dépendante et deux ou plusieurs variables indépendantes (ou explicatives).**
- On a constaté durant l'exploration des données que nous avions des valeurs manquantes dans la colonne "Margin_low".
- Nous souhaitons pouvoir "prédire" ces données à l'aide d'un model de Machine Learning.
- le modèle de régression linéaire multiple pour imputer les valeurs manquantes de margin_low.

➢ **Préparation des données**

- Appliquons la régression linéaire multiple à l'échantillon "billets". Modélisons 'margin_low' en fonction de toutes les autres variables
- On commence par séparer les individus qui ont une valeur manquante dans "Margin_low". Cela correspond à 37 individus

```
[ ]: # On crée un Dataset avec les individus qui ont un margin_low = NaN
billets_nan = billets.loc[billets["margin_low"].isna()]

# On créé un nouveau Dataset sans les valeurs manquantes
billets_sans_nan =billets.dropna()
billets_sans_nan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1463 entries, 0 to 1499
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   is_genuine    1463 non-null   int64
 1   diagonal      1463 non-null   float64
 2   height_left   1463 non-null   float64
 3   height_right  1463 non-null   float64
 4   margin_low    1463 non-null   float64
 5   margin_up     1463 non-null   float64
 6   length        1463 non-null   float64
dtypes: float64(6), int64(1)
memory usage: 91.4 KB
```

Régression linéaire multiple

- *Avec Statmodels: "Méthode Backward Régression"

```
# On utilise la fonction my_backward_selected afin de trouver les varibles descriptives les plus pertinentes

columns = ['margin_low','diagonal','is_genuine','height_left','height_right','margin_up','length']
reg_backward = my_backward_selected(billets_sans_nan[columns],'margin_low')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             margin_low   R-squared:                       0.617
Model:                            OLS   Adj. R-squared:                  0.616
Method:                 Least Squares   F-statistic:                     1174.
Date:                Fri, 10 Nov 2023   Prob (F-statistic):          1.24e-304
Time:                        12:47:15   Log-Likelihood:                -774.73
No. Observations:                1463   AIC:                             1555.
Df Residuals:                    1460   BIC:                             1571.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      5.9263      0.198     30.003      0.000       5.539       6.314
margin_up     -0.2119      0.059     -3.612      0.000      -0.327      -0.097
is_genuine    -1.1632      0.029    -40.477      0.000      -1.220      -1.107
==============================================================================
Omnibus:                       22.365   Durbin-Watson:                   2.041
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               39.106
Skew:                           0.057   Prob(JB):                     3.22e-09
Kurtosis:                       3.793   Cond. No.                         65.0
==============================================================================
```

- Les variables descriptives intéressantes sont : "**is_genuine**" "**margin_up**"
- On peut donc utiliser ce modèle (avec les variables "is_genuine"et"margin_up") à des fins de prévision de la variable 'margin_low' !
- Le *R*2 vaut environ 0.617, et le *R*2 ajusté est d'environ 0.616 et une p_value <0,05 ce qui suggère que les résultats sont statistiquement significatifs, et donc que 'margin_up' et 'is_genuine' sont des prédicteurs significatifs de 'margin_low'.

- *Avec Train_test_split : *''Fonction de scikit-learn utilisée pour diviser les données en ensembles d'entraînement et de test ''*

- *'' y'' présente les valeurs de ma variable à prédire(margin_low)*
- *''X'' présente l'ensemble de mes variables explicatives, à partir desquelles je vais pouvoir prédire y.*

- *Je sépare mes données en un groupe d'entrainement et un groupe de test, ensuite j'instancie notre modèle, je l'entraine et enfin on peut prédire les valeurs de y_pred.*

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
]: # Entraîner notre modèle de régression linéaire sur les données d'entrainement

   model_01 = sm.OLS(y_train, X_train)
   resultat = model_01.fit()

   #Prediction
   y_test = resultat.predict(X_test)

   print(resultat.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.618
Model:                            OLS   Adj. R-squared:                  0.617
Method:                 Least Squares   F-statistic:                     942.0
Date:                Sat, 11 Nov 2023   Prob (F-statistic):          2.89e-244
Time:                        09:49:48   Log-Likelihood:                -606.17
No. Observations:                1170   AIC:                             1218.
Df Residuals:                    1167   BIC:                             1234.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          5.6622      0.220     25.769      0.000       5.231       6.093
is_genuine    -1.1373      0.032    -35.354      0.000      -1.200      -1.074
margin_up     -0.1337      0.065     -2.051      0.040      -0.262      -0.006
==============================================================================
Omnibus:                       21.780   Durbin-Watson:                   1.948
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               41.200
Skew:                           0.034   Prob(JB):                     1.13e-09
Kurtosis:                       3.917   Cond. No.                         65.4
==============================================================================
```

Évaluation du modèle

Avant de valider l'utilisation de la régression linéaire, nous devons confirmer la validité de plusieurs hypothèses afin de s'assurer que nous pouvons appliquer cette régression. Voici les hypothèses à vérifier :

- **Normalité :** Pour satisfaire cette condition, les erreurs résiduelles de notre modèle devraient idéalement se conformer à une distribution normale, centrée autour de zéro. En d'autres termes, les résidus devraient être distribués normalement avec une moyenne égale à zéro.

➢ **Ici, l'hypothèse de normalité est remise en cause (p-value = 6.2e-06 < 0.05).**

```
#Testez la normalité des résidus

from scipy.stats import shapiro

statistic, p_value = shapiro(residuals)

if p_value > 0.05:
    print("Les résidus semblent suivre une distribution normale (hypothèse non rejetée).")
    print("P-value :",p_value)
else:
    print("Les résidus ne suivent pas une distribution normale (hypothèse rejetée).")
    print("P-value :",p_value)
```

```
Les résidus ne suivent pas une distribution normale (hypothèse rejetée).
P-value : 7.6132646427146 57e-06
```

- **Homoscédasticité :** L'homoscédasticité signifie que la variance des erreurs résiduelles est constante à tous les niveaux de la variable prédite. Cela implique que la variance des erreurs devrait rester uniforme à tous les niveaux de la variable explicative.

➢ Dans notre cas le test de "Breusch Pagan" indique une p-valeur inférieure à 5 %, on rejette l'hypothèse *H*0 selon laquelle les variances sont constantes.

➢ **La variance des erreurs n'est pas constante, ce qui est un problème pour la régression linéaire, affectant la fiabilité de vos résultats.**

```
import statsmodels.api as sm
import statsmodels.stats.diagnostic as smd

# Testez l'homoscédasticité (c'est-à-dire la constance de la variance) des résidus :
_, pval, __, f_pval = smd.het_breuschpagan(resultat.resid, X_train)
print('p value test Breusch Pagan:', pval)
```

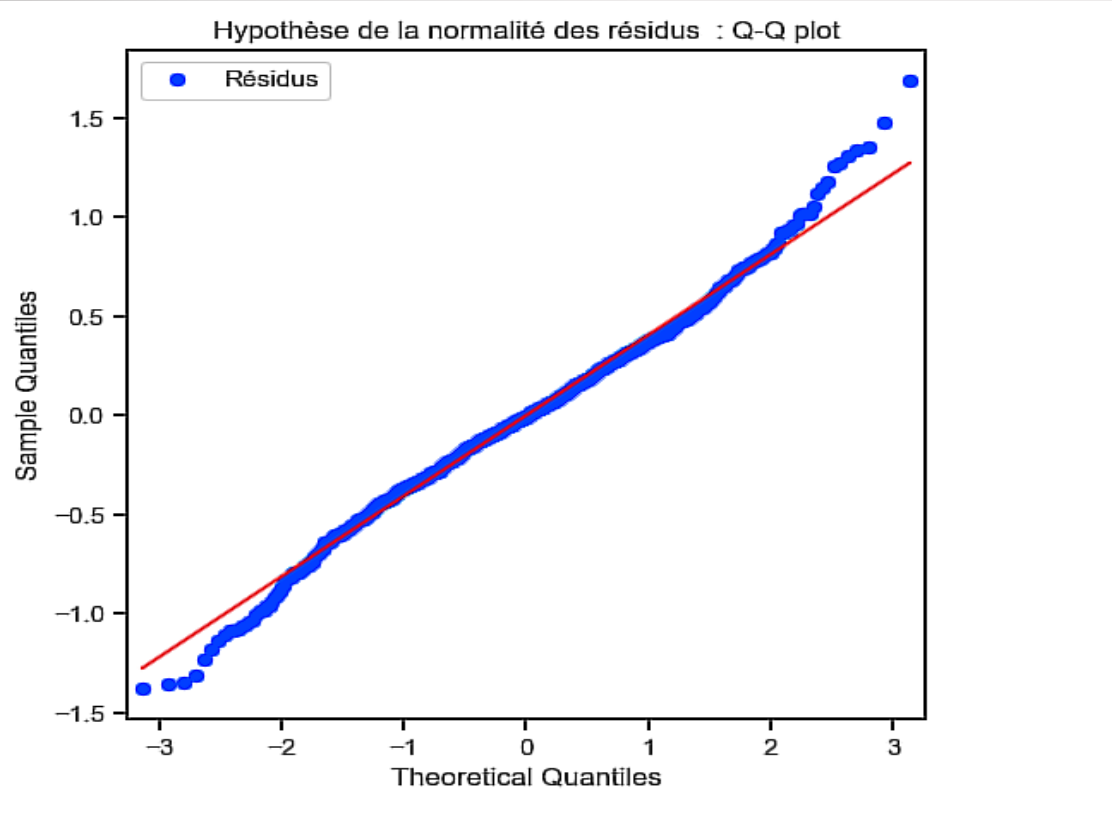
```
p value test Breusch Pagan: 1.6223530361976033e-27
```

- **Multi colinéarité :** Cette hypothèse concerne la relation entre les variables explicatives dans notre modèle de régression. Elle stipule qu'il ne devrait pas y avoir de forte corrélation linéaire entre les variables indépendantes.

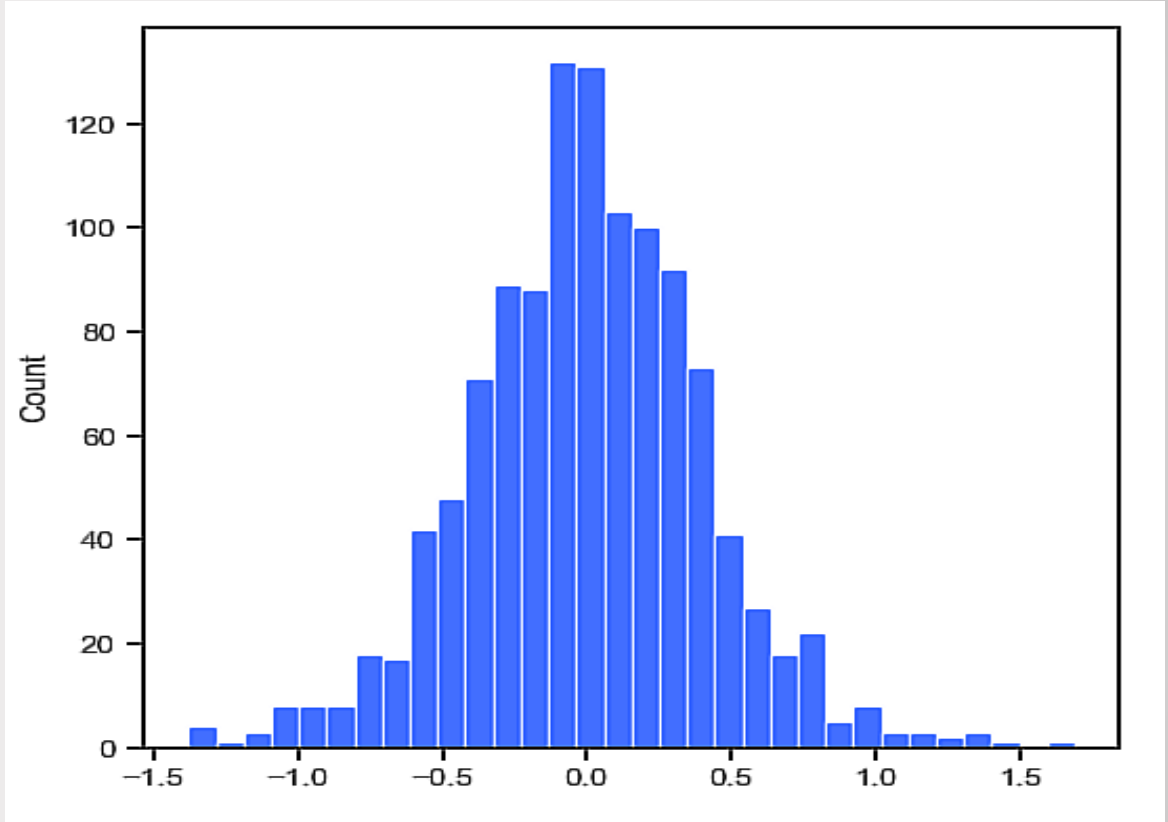
➢ **Les valeurs de Facteur d'Inflation de la Variance (VIF) pour 'is_genuine' et 'margin_up' sont environ 1.6, ce qui est bien en dessous du seuil commun de 10 ce qui suggère une absence de multi colinéarité significative.**

```
Variance Inflation Factor (VIF):
      Variable          VIF
0        const  341.438603
1   is_genuine    1.617936
2    margin_up    1.617936
```

Régression linéaire multiple



Le diagramme Quantile-Quantile est un outil graphique permettant d'évaluer la pertinence de l'ajustement d'une distribution donnée à un modèle théorique. Ce diagramme représente la normalité des résidus.



L'analyse des résidus montre qu'ils s'alignent approximativement avec une distribution symétrique et, compte tenu de la taille de l'échantillon qui excède 30, on peut conclure que les prédictions du modèle linéaire gaussien sont plausibles, même en l'absence d'une normalité parfaite des résidus.

Vérification de la cohérence des résultats obtenus

Sans la prédiction du modèle de Régression Linéaire

	margin_low
count	1463.000000
mean	4.485967
std	0.663813
min	2.980000
25%	4.015000
50%	4.310000
75%	4.870000
max	6.900000

Avec la prédiction du modèle de Régression Linéaire

	margin_low
count	1500.000000
mean	4.482809
std	0.659780
min	2.980000
25%	4.030000
50%	4.310000
75%	4.870000
max	6.900000

- **On crée le modèle de régression linéaire avec une prédiction qui permettra de remplacer les valeurs manquantes.**
- **Sur le deuxième tableau, le count nous montre que toutes nos valeurs ont bien été remplacées : nous n'avons plus de valeurs manquante dans le dataset.**

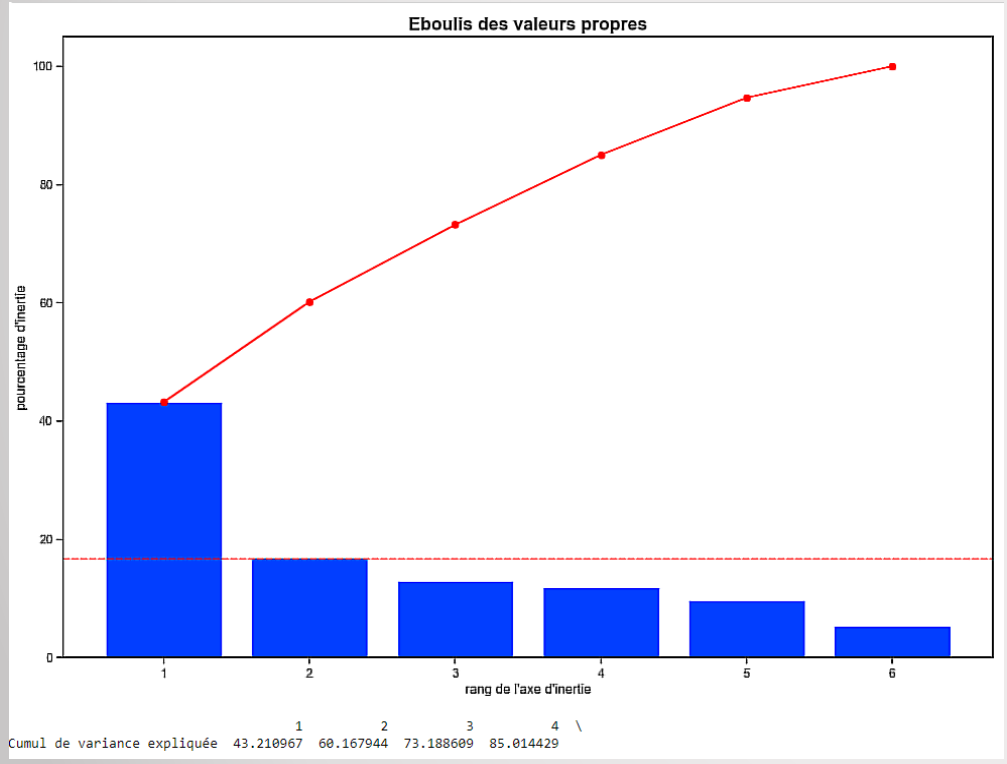
Modélisations: test des modèles de classifications

➢ **Prédiction de l'authenticité des billets**

Pour la mise en pace d'un modèle capable de prédire au mieux si un billet et vrai ou faux, nous allons tester plusieurs méthodes : K-Means, LogisticRegression, KNN, RandomForest.

➢ **ACP : Algorithme de réduction de dimensionnalité**

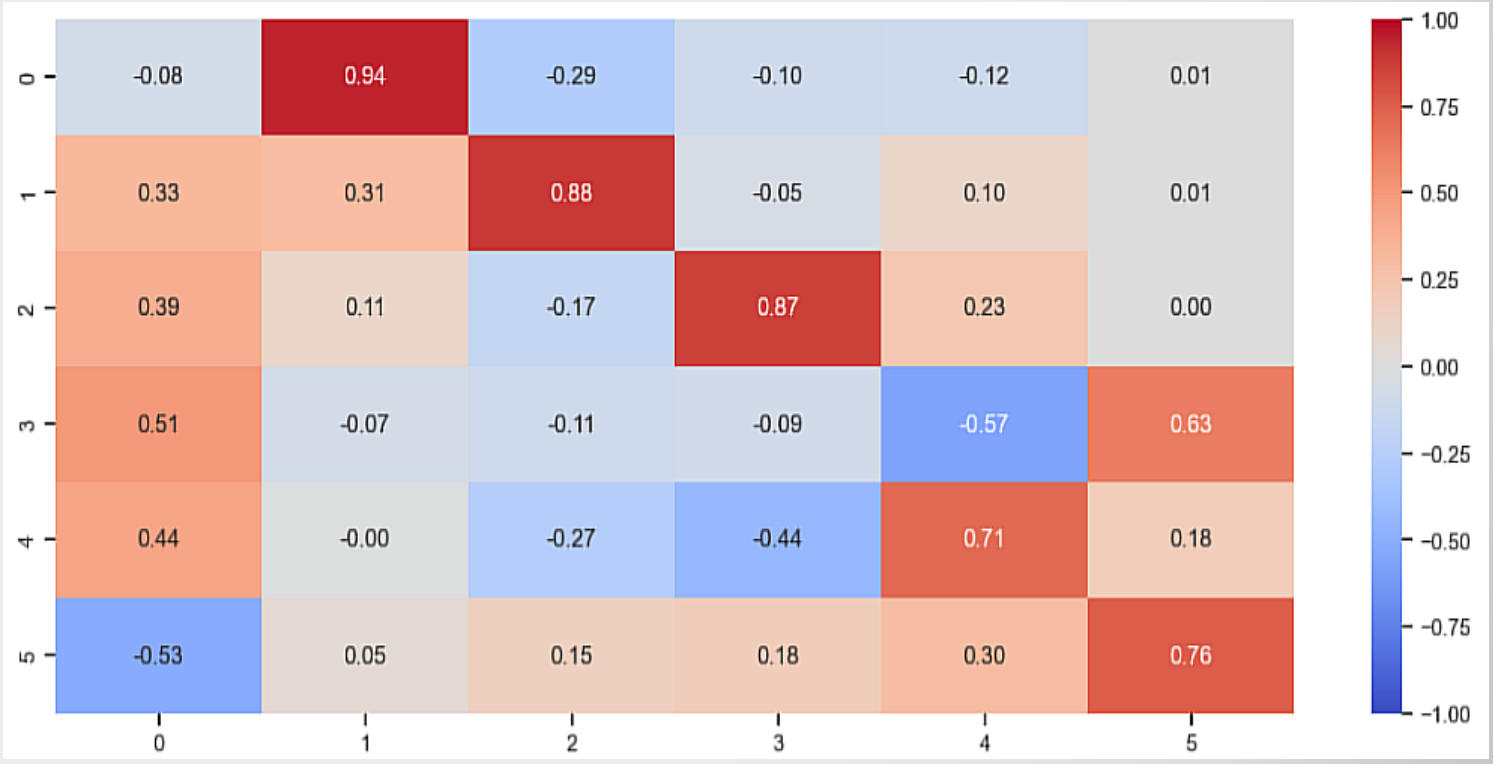
Création de l'éboulis des valeurs propres



	1	2	3	4	\
Cumul de variance expliquée	43.210967	60.167944	73.188609	85.014429	

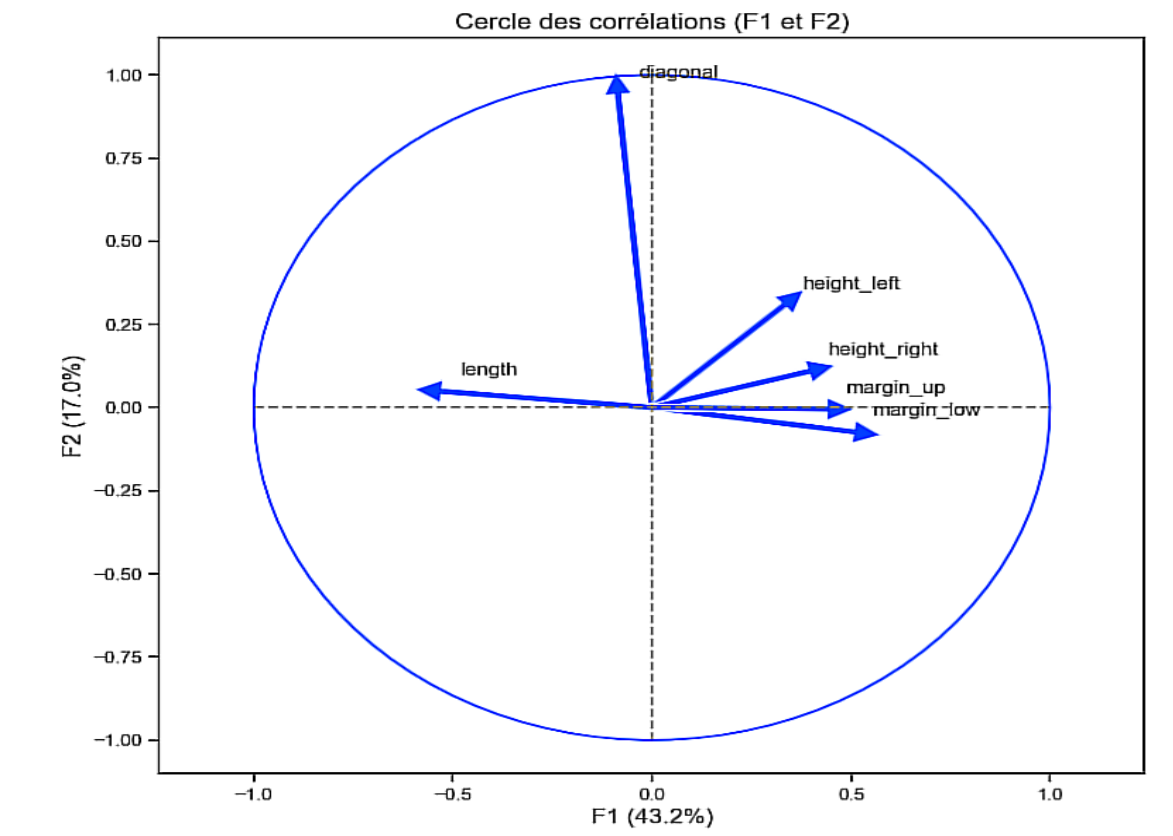
Nous analysons donc seulement les variables F1 et F2 qui représentent 60,2% de l'inertie totale (variance).

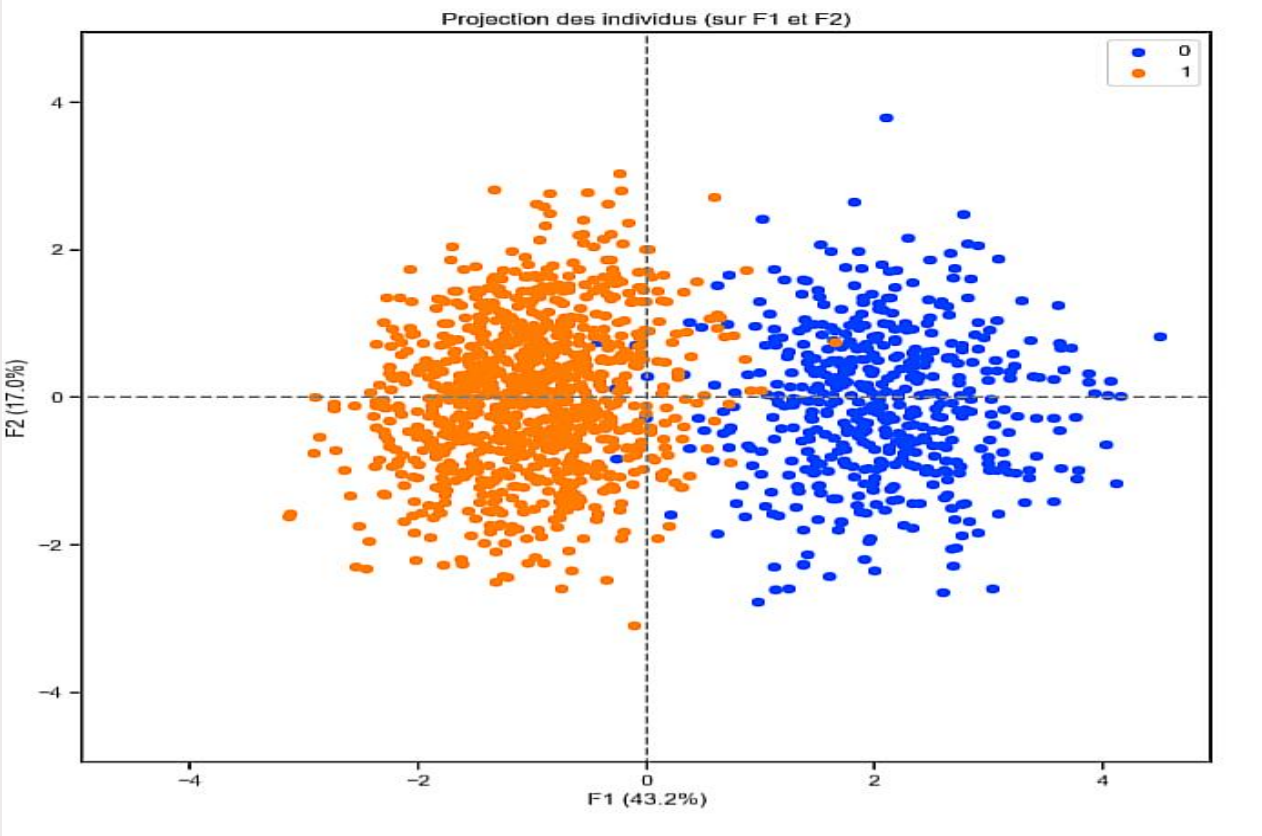
Heatmap



Je choisis 2 composantes principales, car les 2 premières dimensions représentent 60% des informations importantes.

Cercle des corrélations et projection des individus sur le plan factoriel





- le cercle des corrélations indique que les variables les plus liées à la première composante F1 sont length, margin_low, et aussi margin_up, height_right et height_left. En revanche, la variable diagonal ne semble pas avoir de relation significative.

- Il est apparent que c'est uniquement la première composante qui permet une séparation efficace entre les vrais et les faux billet
- Il sera intéressant d'analyser les clusters et leur centroïde pour mieux comprendre sur quoi nos deux ensembles varient.

Méthode des KMeans (clustering non-supervisé)

- **L'algorithme kmeans est un algorithme itératif qui minimise la somme des distances entre chaque individu et le centroïde du cluster ; c'est la variabilité intra cluster. Attribuer un cluster à chaque objet (ou sujet, ou point), de façon aléatoire.**

➢ Préparation des données et matrice de confusion

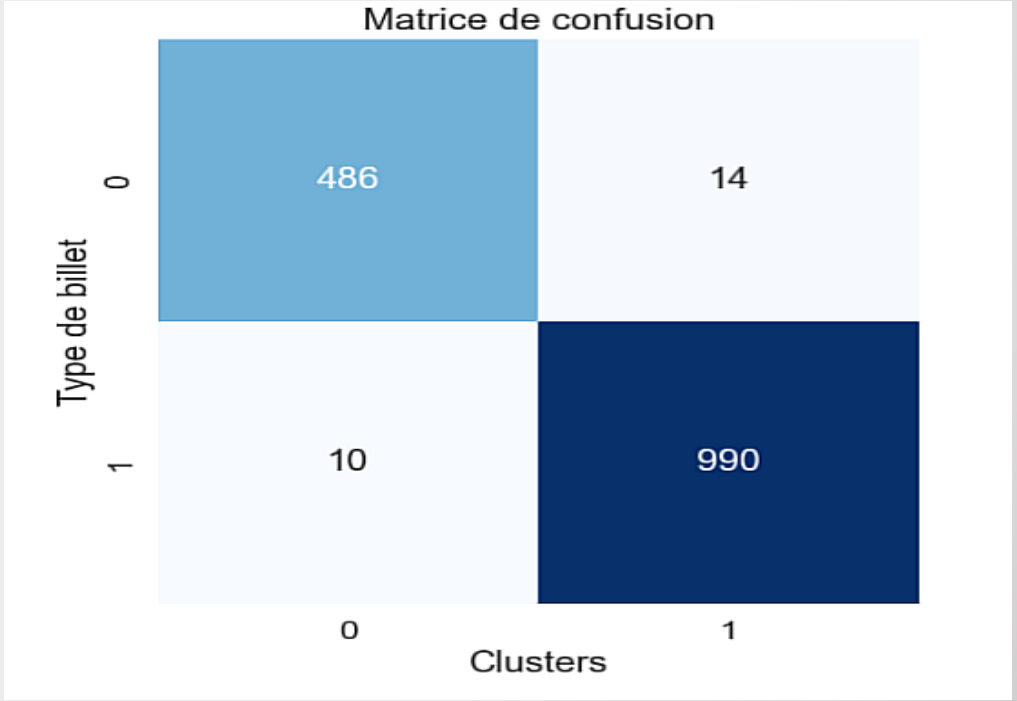
```
y_vrai= kmeans_model .is_genuine
y_pred= kmeans.labels_
```

```
# Récupération des clusters attribués à chaque individu

kmeans_model ['cluster']= kmeans.labels_

kmeans_model .head()
```

	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length	cluster
0	1	171.81	104.86	104.95	4.52	2.89	112.83	0
1	1	171.46	103.36	103.66	3.77	2.99	113.09	1
2	1	172.69	104.48	103.50	4.40	2.94	113.16	1
3	1	171.36	103.91	103.94	3.62	3.01	113.51	1
4	1	171.73	104.28	103.46	4.04	3.48	112.54	1



La matrice de confusion montre les résultats d'une classification par l'algorithme K-mean

KMeans

- **Évaluation du modèle avec les scores**

```
# Calculons nos scores de performance :

kmeans_accuracy = accuracy_score(y_vrai,y_pred).round(4)
print("Accuracy :",kmeans_accuracy)

kmeans_precision = precision_score(y_vrai,y_pred).round(4)
print("Precision :",kmeans_precision)

kmeans_recall = recall_score(y_vrai,y_pred).round(4)
print("Recall :",kmeans_recall)
```

```
Accuracy : 0.984
Precision : 0.9861
Recall : 0.99
```

Classification report

```
from sklearn.metrics import classification_report
print(classification_report(y_vrai,y_pred, ))
```

	precision	recall	f1-score	support
0	0.98	0.97	0.98	500
1	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

- **Exactitude(Accuracy):** mesure la globalité des prédictions correctes.
- **La précision(Precision):** se concentre sur la qualité des prédictions positives
- **Le rappel(Recall):** se concentre sur la capacité du modèle à trouver tous les exemples positifs
- **Notre modèle présente une performance de 98% (taux de prédiction correcte sur toutes les données de test)**
- **Une précision de 98% pour le modèle de K-means est généralement considérée comme très élevée, ce qui suggère que le modèle a réussi à bien classifier les données dans les clusters appropriés.**

Régression logistique (classification supervisé)

- La régression logistique est une technique d'apprentissage supervisé utilisée pour prédire la probabilité d'une variable cible catégorielle, souvent binaire, en fonction d'une ou plusieurs variables indépendantes. Ce modèle permet d'étudier les relations entre un ensembles de variables prédictives (explicatives) et une variable à prédire y.

➢ Préparation des données et matrice de confusion

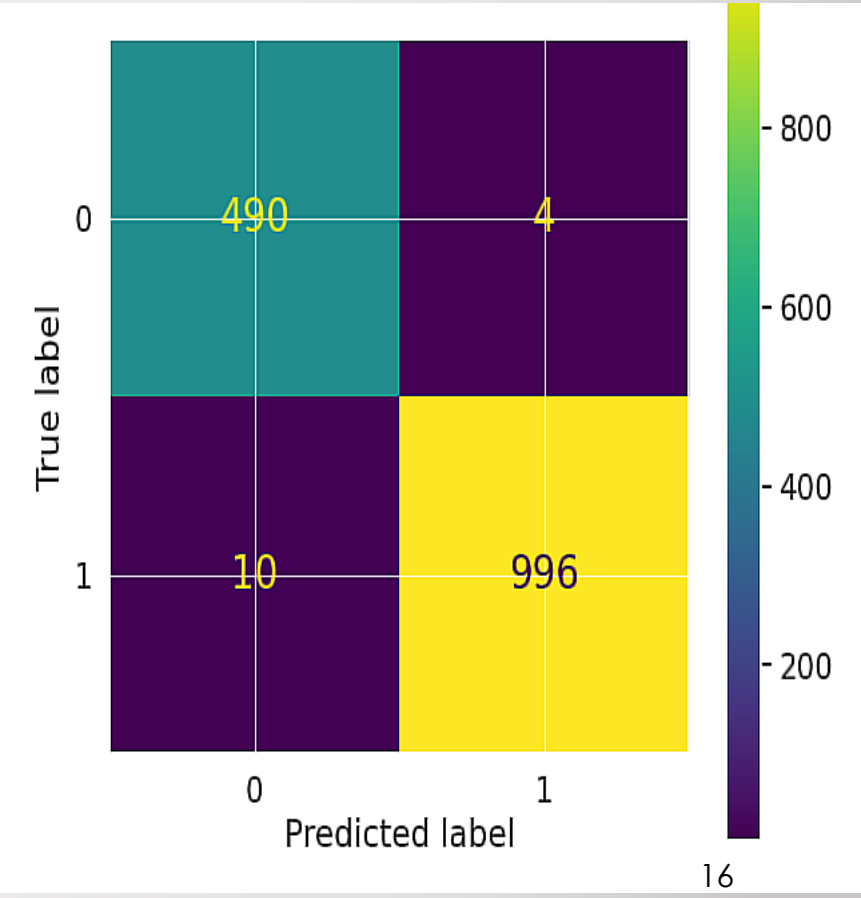
1. Fractionnement des données en un groupe d'entrainement et un groupe de test
2. Initialisation et entraînement du modèle
3. Prédiction(utilisation du modèle)
4. Calcul du score et évaluation du modèle(matrice de confusion, courbe AUC ROC, Score F1,Validation croisée…

```
# Séparons nos données en un groupe d'entrainement et un groupe de test.
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state =5)
```

```
#Initialisation et entraînement du modèle
rl_model = LogisticRegression()
rl_model.fit(x_train, y_train)
```

```
LogisticRegression()
```

```
# Prédiction sur l'ensemble de test
y_pred_log= rl_model.predict(x_test)
```



''La matrice de confusion nous donne de très bon résultats''

Régression logistique

➢ Évaluation du modèle de régression logistique

1. **L'AUC ROC:** **La zone sous la courbe (AUC : Area Under the Curve) est une mesure de la performance du modèle. Une AUC de 1.0 indique une performance parfaite, où le modèle a un taux de vrais positifs de 1 et un taux de faux positifs de 0.**
   Dans notre cas, la valeur A.U.C est de 1 , ce qui confirme que le modèle a une excellente capacité à distinguer les billets authentiques des faux.

2. **Score F1:** **est une mesure harmonique de la précision et du rappel d'un modèle de classification.**

```
from sklearn.metrics import f1_score
f1_log =f1_score(y_test,y_pred_log)
f1_log.round(4)
```

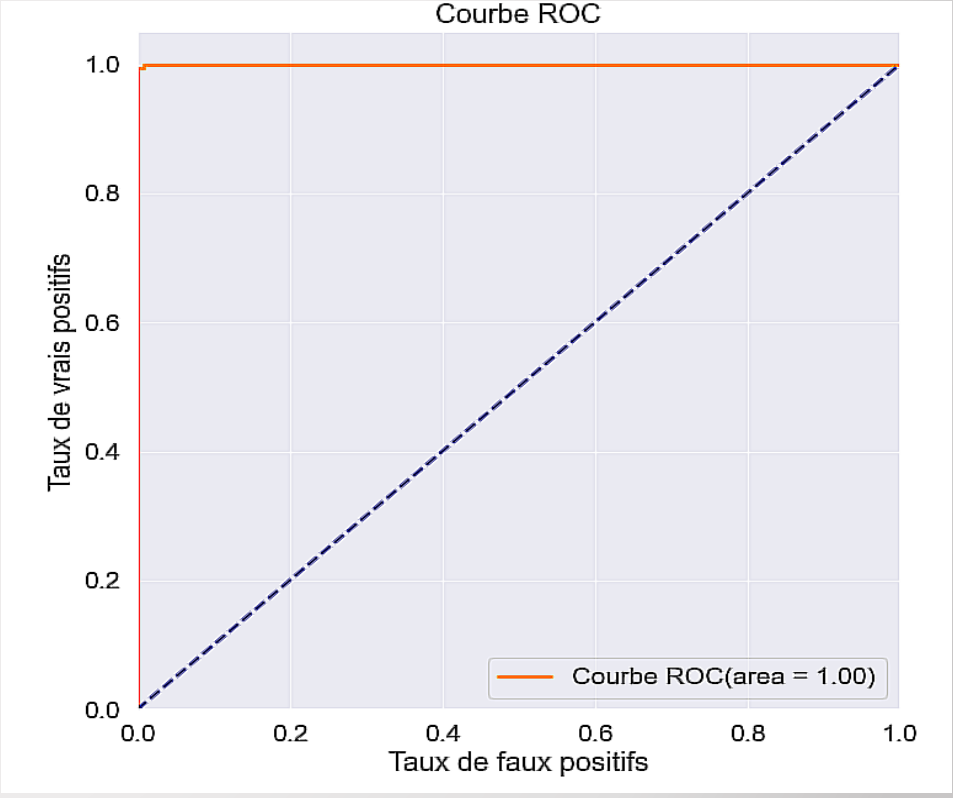
```
0.9974
```

Score F1 =0.9974, ce qui indique une excellente combinaison de précision et de rappel modèle très performant.

3. Accuracy/Precision/Recall :

```
Cross-Validated Predictions:
[0 1 1 ... 0 0 0]
Accuracy : 0.9906666666666667
Precision : 0.9900596421471173
Recall : 0.996
Confusion Matrix:
 [[490  10]
 [  4 996]]
```

Grâce à une validation croisée, nous avons évalué la performance de notre modèle de classification. Les métriques de performance révèlent que le modèle est extrêmement performant, avec une matrice de confusion démontrant un nombre très réduit de faux positifs (10).

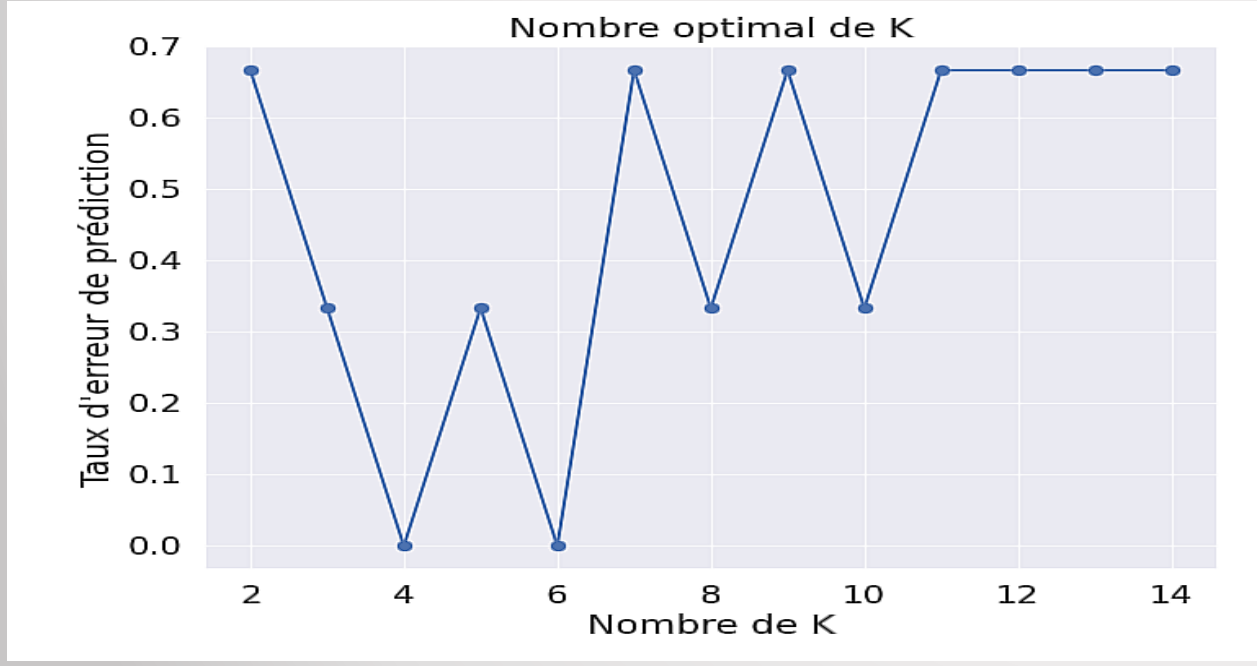


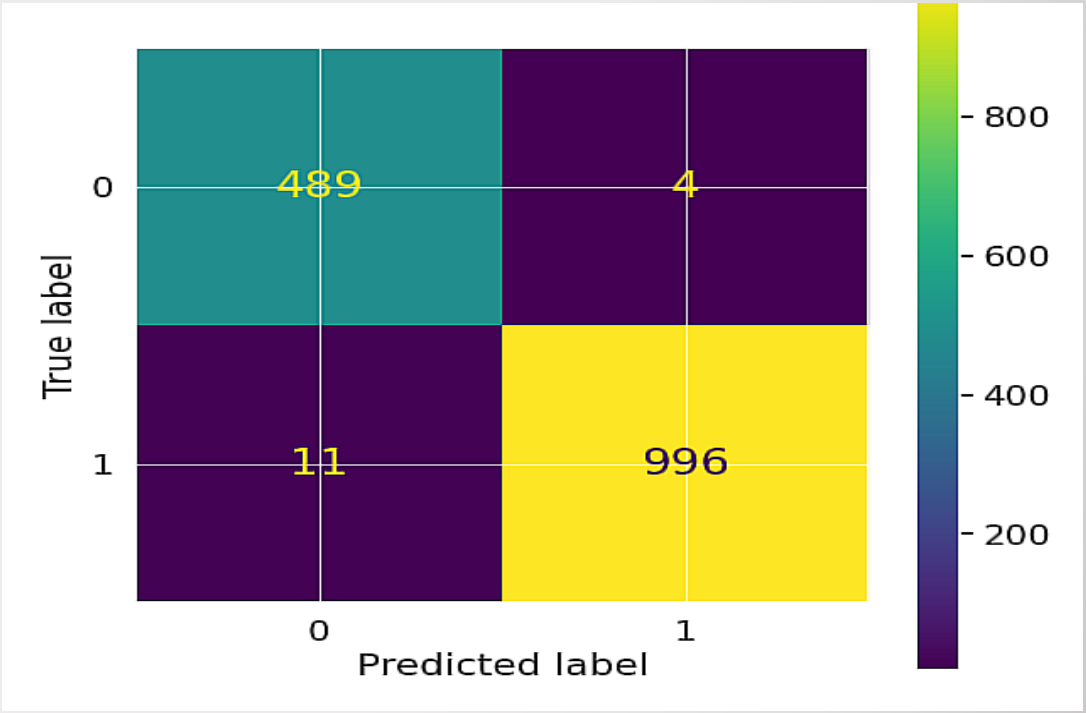
Méthode K-NN (le plus proche voisin)

- Le KNN, ou "k-plus proches voisins", est une méthode simple utilisée en machine Learning pour prédire la catégorie d'un nouvel élément en se basant sur les catégories des k éléments les plus proches de lui dans l'ensemble de données.

➢ Préparation des données et matrice de confusion

1. Fractionnement des données en un groupe d'entrainement et un groupe de test
2. Choisir le nombre optimal de K (le meilleur "k« est celui qui donne le taux d'erreur le plus bas sur un ensemble de validation)
3. Instanciation du modèle avec le meilleur k
4. Entrainement du modèle sur nos données train
5. Prédiction(utilisation du modèle)
6. Évaluation du modèle (matrice de confusion, les scores de performances ,courbe ROC,…).





La matrice de confusion nous donne de très bon résultats indiquant très peu de faux positifs (4) par rapport aux vrais positifs (996) et vrais négatifs (489).

Méthode K-NN

➢ Évaluation du modèle de régression logistique

1. **L'AUC ROC :** **dans notre cas, la valeur A.U.C est égale à 1 , ce qui confirme que le modèle a une excellente capacité à distinguer les billets authentiques des faux.**

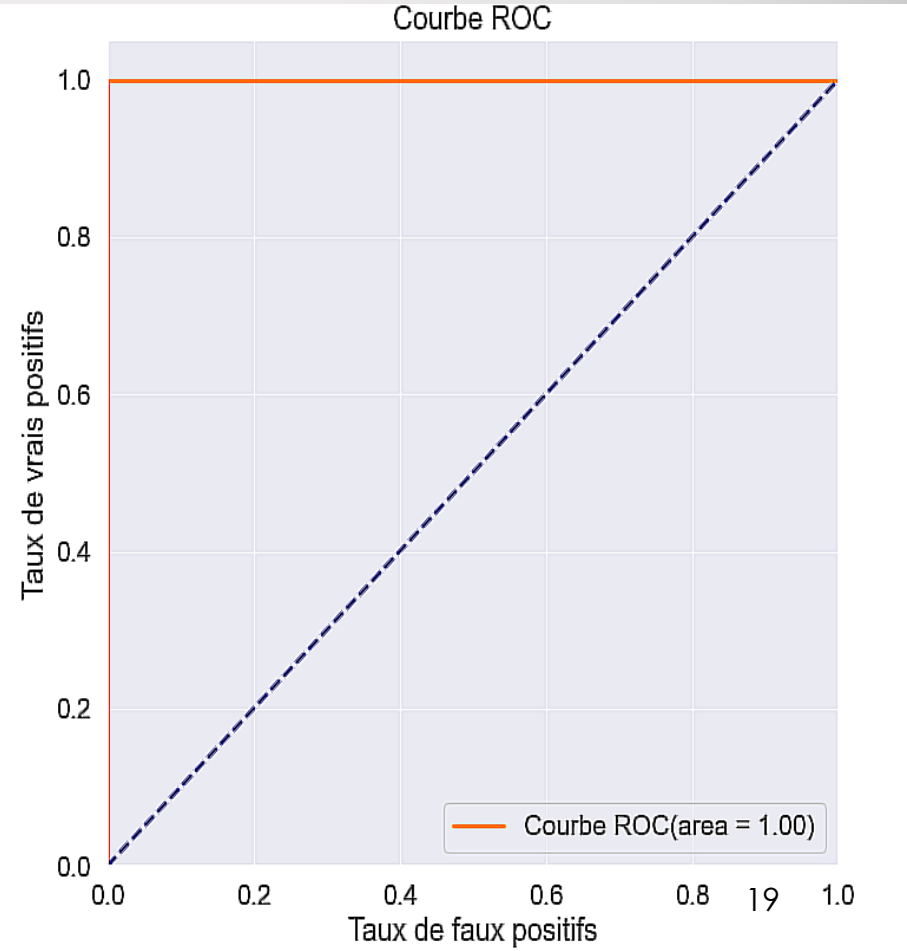
1. **Score F1 :** **f1 = 1 ce qui indique une excellente combinaison de précision et de rappel = modèle très performant.**

```
: f1_knn =f1_score(y_test,y_pred_knn)
  f1_knn.round(4)
```

```
1.0
```

2. **Accuracy/Precision/Recall :**

```
Cross-Validated Predictions:
[0 1 1 ... 0 0 0]
Accuracy : 0.99
Precision : 0.9890764647467726
Recall : 0.996
Confusion Matrix:
 [[489  11]
 [  4 996]]
```



Notre modèle K-NN fait très peu d'erreur de prédiction, donc il est excellent, il présente une fiabilité de 99%.

Méthode de Randomforest

- La forêt aléatoire est une technique d'apprentissage automatique qui construit de nombreux arbres de décision pour améliorer la robustesse et la précision des prédictions.
- Chaque arbre donne une prédiction et le résultat final est obtenu en combinant les prédictions de tous les arbres.
- Elle est particulièrement efficace pour gérer des ensembles de données complexes, tant pour la classification que pour la régression.

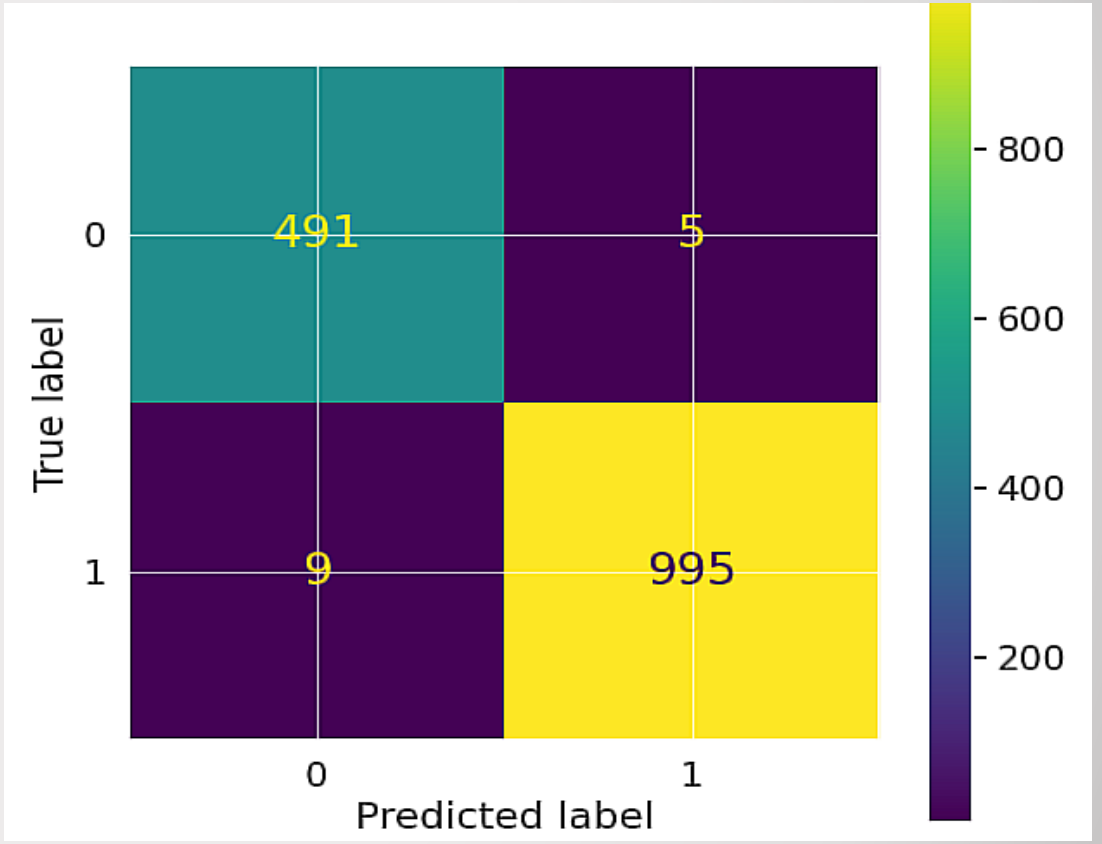
➢ Préparation des données et matrice de confusion

1. Séparation des données en un groupe d'entrainement et un groupe de test
2. Entrainement du modèle sur nos données train
3. Prédiction sur l'ensemble de test

```
# Séparons nos données en un groupe d'entrainement et un groupe de test.
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=5)
```

```
# Initialisation et entraînement du modèle
rf_model= RandomForestClassifier(random_state=5)
rf_model.fit(x_train, y_train)
```

```
# Prédiction sur l'ensemble de test
y_pred_rf = rf_model.predict(x_test)
```



La matrice de confusion de notre modèle Random Forest nous donne également de très bon résultats indiquant très peu de faux positifs (5) par rapport aux vrais positifs (995) et vrais négatifs (491).

Randomforest

➢ Évaluation du modèle de régression logistique

1. **L'AUC ROC : dans notre cas, la valeur A.U.C est égale à 1 , ce qui suggère que le modèle a correctement classifié tous les billets.**

2. **Score F1 : f1_score = 0.9949, ce qui indique une excellente combinaison de précision et de rappel**

```
: f1_rf =f1_score(y_test,y_pred_rf)
  f1_rf.round(4)
```

```
0.9949
```

1. **Accuracy/Precision/Recall :**

```
Cross-Validated Predictions:
[0 1 1 ... 0 0 0]
Accuracy : 0.9906666666666667
Precision : 0.9910358565737052
Recall : 0.995
Confusion Matrix:
 [[491   9]
 [  5 995]]
```

Les métriques de performance révèlent que le modèle est extrêmement performant, avec une matrice de confusion démontrant un nombre très réduit de faux positifs (9).

Comparaison des performances des modèles

- tableau de performance

	kmeans	Regression_Logistique	kNN	Randomforest
Accuracy	0.9840	0.9967	1.0	0.9933
Precision	0.9861	1.0000	1.0	0.9949
Rappel	0.9900	0.9949	1.0	0.9949

- Ce tableau montre les métriques de performance pour nos modèles de classifications:

KMeans, Régression Logistique, kNN (k-Nearest Neighbors), et Random Forest. Les métriques fournies sont l'Accuracy (Exactitude), la Precision (Précision), et le Recall (Rappel).

- Tous les modèles semblent bien performer sur l'ensemble de données, avec kNN et Régression Logistique en tête.

Évaluation de Modèles Classificateurs(Validation Croisée)

	Regression_Logistique	kNN	Randomforest
Accuracy	0.990000	0.990667	0.992000
Precision	0.989999	0.990674	0.992011
Recall	0.990000	0.990667	0.992000

- Ce tableau présente les métriques de performance obtenues par validation croisée pour trois modèles : Régression Logistique, kNN, et Random Forest.
- La validation croisée est un bon indicateur que la Régression Logistique pourrait être le modèle le plus robuste et le plus généralisable parmi ceux évalués ici.
- Nous allons donc utiliser ce modèle pour notre application finale afin de prédire
- l' authenticité ou pas de nouveaux billets.

Application finale

```
detection_faux_billets('billets_production(1).csv',rl_model)
```

id	diagonal	height_left	height_right	margin_low	margin_up	length	Authenticite	Probabilite_authenticite
A_1	171.76	104.01	103.54	5.21	3.30	111.42	Faux	0.002983
A_2	171.87	104.17	104.13	6.00	3.31	112.09	Faux	0.000777
A_3	172.00	104.58	104.29	4.99	3.39	111.57	Faux	0.001405
A_4	172.49	104.55	104.34	4.44	3.03	113.20	Vrai	0.946759
A_5	171.65	103.63	103.56	3.77	3.16	113.33	Vrai	0.999508

- Les trois premiers billets sont classés comme faux avec des probabilités d'authenticité très faibles, tandis que les deux derniers sont classés comme vrais avec des probabilités élevées, suggérant que le modèle est confiant dans ses prédictions.
- Le modèle prédit correctement la présence de billets vrais et faux, identifiant trois comme faux et deux comme vrais, ce qui indique une bonne séparation des classes et démontre la performance du modèle.

MERCI