ONCFM

Détectez des faux billets

Sommaire

1. Exploration des données

1 variable qualitative : **True** or **False**

6 variables quantitatives : dimensions géométriques des billets

1500 billets

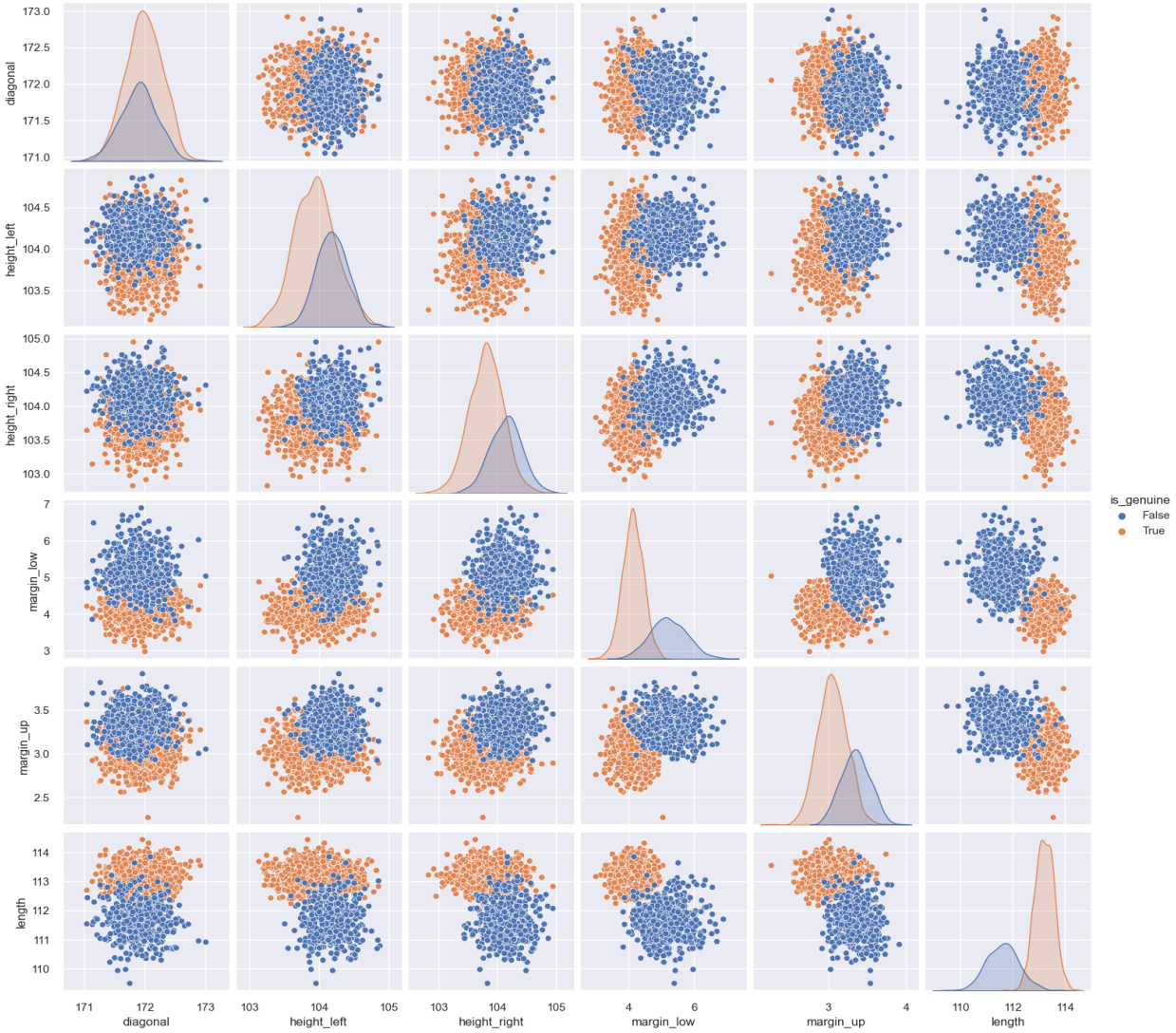
	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length
0	True	171.81	104.86	104.95	4.52	2.89	112.83
1	True	171.46	103.36	103.66	3.77	2.99	113.09
2	True	172.69	104.48	103.50	4.40	2.94	113.16
3	True	171.36	103.91	103.94	3.62	3.01	113.51
4	True	171.73	104.28	103.46	4.04	3.48	112.54

- 1000 billets authentiques (True)
- 500 billets faux (False)

37 valeurs manquantes sur la variable margin_low (2,5% des valeurs de la variable)

L'objectif va être de compléter ces valeurs manquantes

1. Exploration des données
Corrélation entre les variables



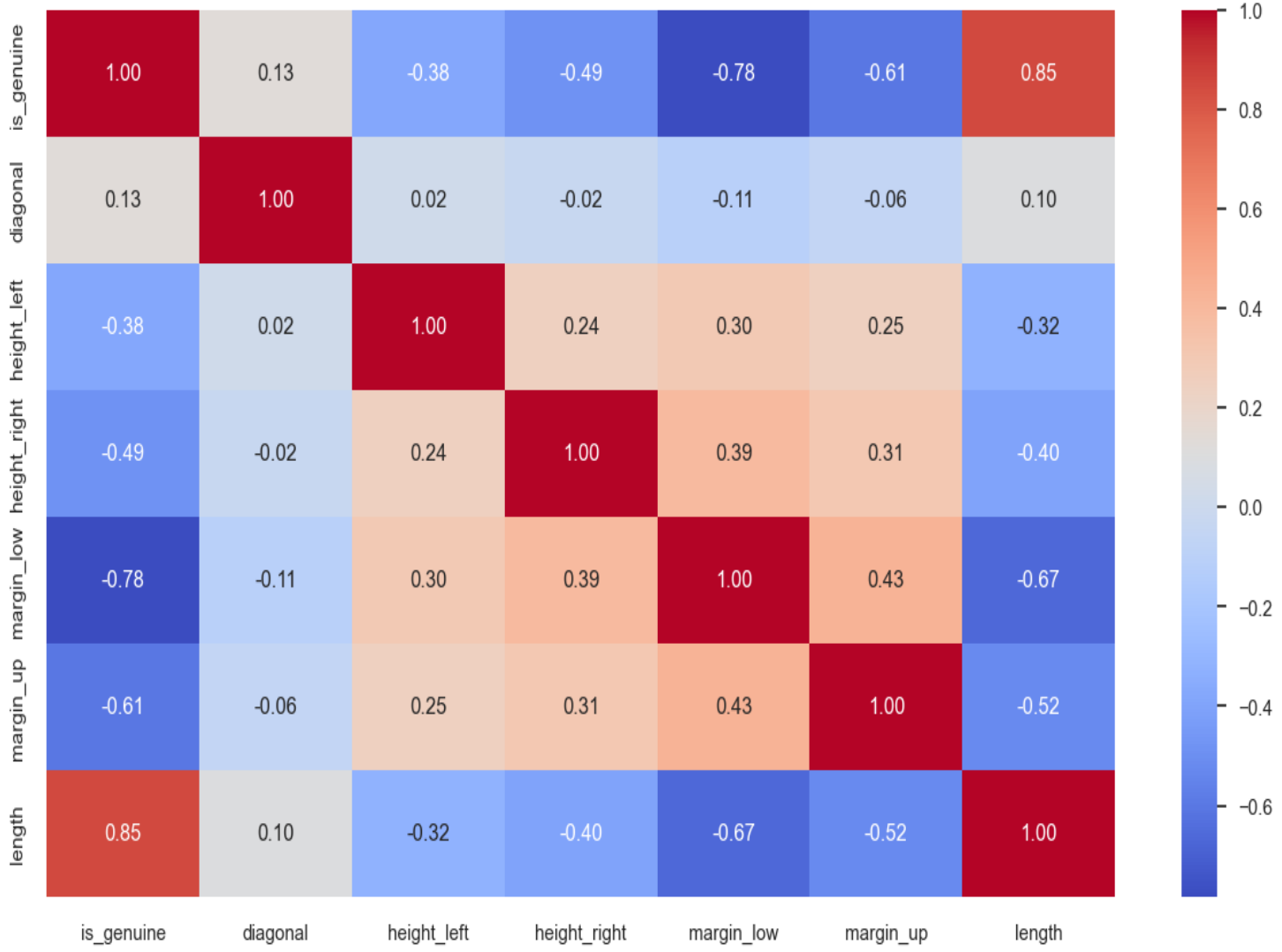
Avec un pairplot, je vais pouvoir afficher **la corrélation entre les variables deux à deux.**
Je vais pouvoir constater les premières tendances entre les vrais et les faux billets et constater des différences de dimensions moyennes.
Pour l'ensemble du projet, les vrais billets apparaitront en **orange** et les faux billets en **bleu**.

Par exemple, si je m'attarde sur la variable margin_low, je vais pouvoir remarquer qu'elle est corrélée positivement avec height_right et margin_up et négativement avec length.

Les vrais billets ont tendance à avoir une longueur supérieure et les faux, une marge basse et une marge haute qui sont plus importantes.

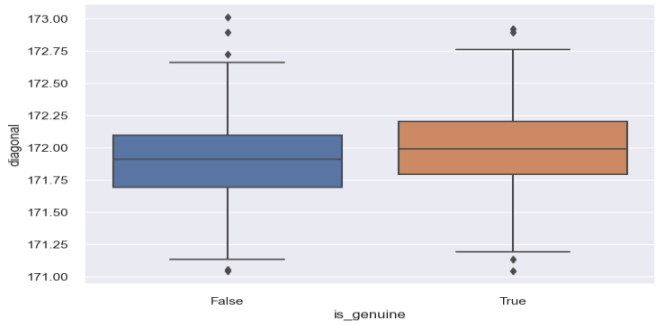
1. Exploration des données
Corrélation entre les variables

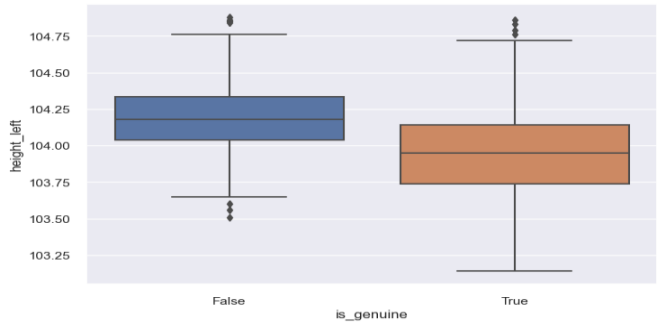
Pour plus de précision, nous pouvons voir les **corrélations avec une heatmap** et afficher les **coefficients de corrélation**.
Si nous regardons la corrélation entre la variable margin_low et la variable length nous pouvons donc confirmer la corrélation négative avec un coefficient de -0,67.
La variable margin_low est également corrélée positivement avec les variables height_left, height_right et margin_up.
Il y a une toute petite corrélation négative avec la variable diagonal.

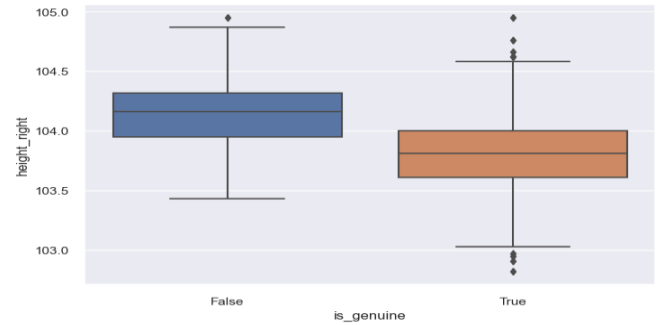


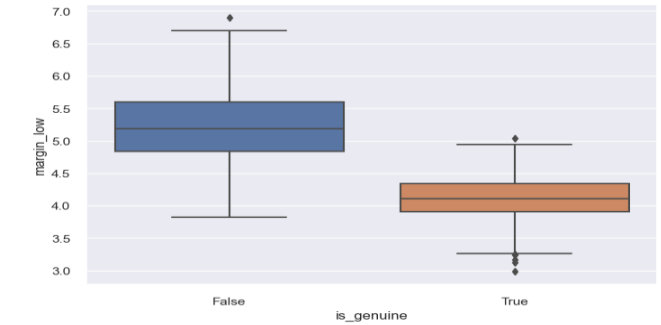
1. Exploration des données
Distribution des variables

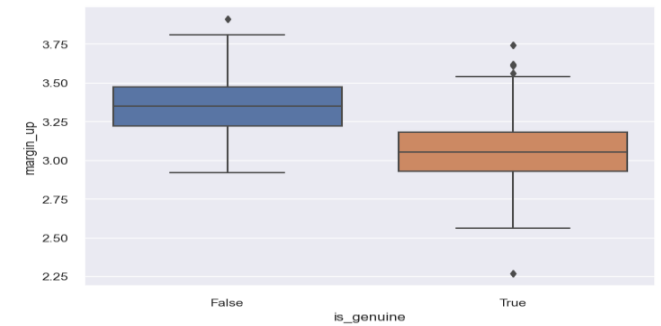
Boxplots

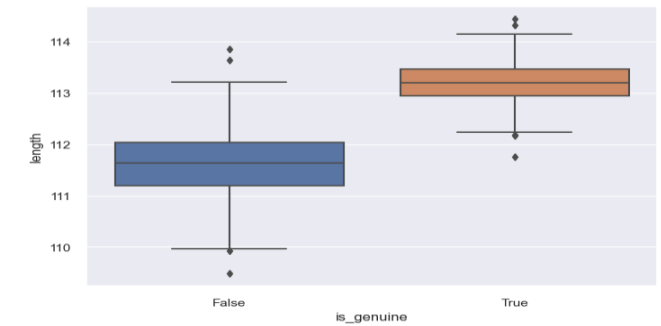












1. Exploration des données
Tests de Shapiro et Kolmogorov-Smirnov

```
# Test de Shapiro-Wilk

alpha = 0.05

for i in Df_Billet.iloc[:,:]:
    stat, pvalue = stats.shapiro(Df_Billet[i])
    print("p_value: ", pvalue)
    if pvalue > alpha:
        print(i, ": Les données suivent une loi normale\n")
    else:
        print(i, ": Nous rejetons l'hypothèse nulle, les données ne suivent pas une loi normale\n")
```

```
p_value:  0.0
is_genuine : Nous rejetons l'hypothèse nulle, les données ne suivent pas une loi normale

p_value:  0.32343590259552
diagonal : Les données suivent une loi normale

p_value:  0.0509396530687809
height_left : Les données suivent une loi normale

p_value:  0.9806053638458252
height_right : Les données suivent une loi normale

p_value:  1.0
margin_low : Les données suivent une loi normale

p_value:  0.000810406228993088
margin_up : Nous rejetons l'hypothèse nulle, les données ne suivent pas une loi normale

p_value:  7.863947037789753e-28
length : Nous rejetons l'hypothèse nulle, les données ne suivent pas une loi normale
```

Les données Diagonal, height_left, height_right et margin_low suivent une loi normale en revanche les données margin_up, length ne suivent pas une loi normale.

1. Exploration des données
Tests de Shapiro et Kolmogorov-Smirnov

Test de Kolmogorov-Smirnov

```
liste_var=Df_Billet_sans_nan.drop(columns=['margin_low','is_genuine'])
for var in liste_var :
    print(var,":",st.ks_2samp(Df_Billet[var],list(np.random.normal(np.mean(Df_Billet[var]), np.std(Df_Billet[var]), 1000))))

    if pvalue < alpha:
        print("Les données suivent une loi normale\n")
    else:
        print("Nous rejetons l'hypothèse nulle, les données ne suivent pas une loi normale")
```

```
diagonal : KstestResult(statistic=0.042, pvalue=0.23579917101222045, statistic_location=171.75, statistic_sign=1)
Les données suivent une loi normale

height_left : KstestResult(statistic=0.03966666666666667, pvalue=0.29623261897937675, statistic_location=103.93977969564332, st
atistic_sign=-1)
Les données suivent une loi normale

height_right : KstestResult(statistic=0.044, pvalue=0.19178172870967158, statistic_location=103.92829094337036, statistic_sign=
-1)
Les données suivent une loi normale

margin_up : KstestResult(statistic=0.043, pvalue=0.21292189001678022, statistic_location=3.1, statistic_sign=1)
Les données suivent une loi normale

length : KstestResult(statistic=0.16366666666666665, pvalue=1.77615807858617e-14, statistic_location=112.92871562685946, statis
tic_sign=-1)
Les données suivent une loi normale
```

2. Implémentation des valeurs manquantes
1. Régression linéaire simple : définition

La régression linéaire simple est une méthode statistique permettant de trouver une relation linéaire entre une variable explicative X et une variable à expliquer y.
Cette relation va pouvoir s'écrire avec la fonction linéaire suivante, qui va être représentée par une droite :

$$y = a.x + b$$

y = la variable que vous essayez de prédire (variable dépendante).
X = la variable que vous utilisez pour prédire y (variable indépendante).
a = la pente (si a>0, la droite est croissante, si a=0, la droite est horizontale, si a<0, la droite est décroissante)
b = l'intercept : l'ordonnée du point d'intersection de la droite avec l'axe vertical en x = 0

Afin de mieux comprendre les relations entre les variables, nous avons construit des modèles de régression linéaire pour prédire margin_low.

2. Implémentation des valeurs manquantes
1. Régression linéaire simple : préparation des données

y (margin_low) correspond aux valeurs que nous souhaitons prédire

X correspond à la variable prédictive à partir de laquelle nous allons trouver y

Avec la fonction **train_test_split**, appliquée à X et y, je vais séparer mes données en **un groupe d'entrainement et un groupe de test**. Je choisis pour mon découpage, 80% de données d'entrainement et 20% de données de test.

```
# On fractionne le dataset en 80/20 pour avoir plus de donnée d'entraitenement que de test

train, test = train_test_split(Df_Billet_sans_nan, test_size=0.2, random_state=42)
```

Nous pouvons ensuite **instancier un modèle** depuis le package linear_regression de scikit-learn.
Puis nous l'**entrainons sur nos données d'entrainement** avec la fonction fit.
Nous pouvons ensuite l'utiliser pour faire des **prédictions** avec la fonction predict appliquée à nos données X_test.

```
regLin=LinearRegression()
```

```
# Entraînement des données

regLin.fit(X_train,y_train)
```

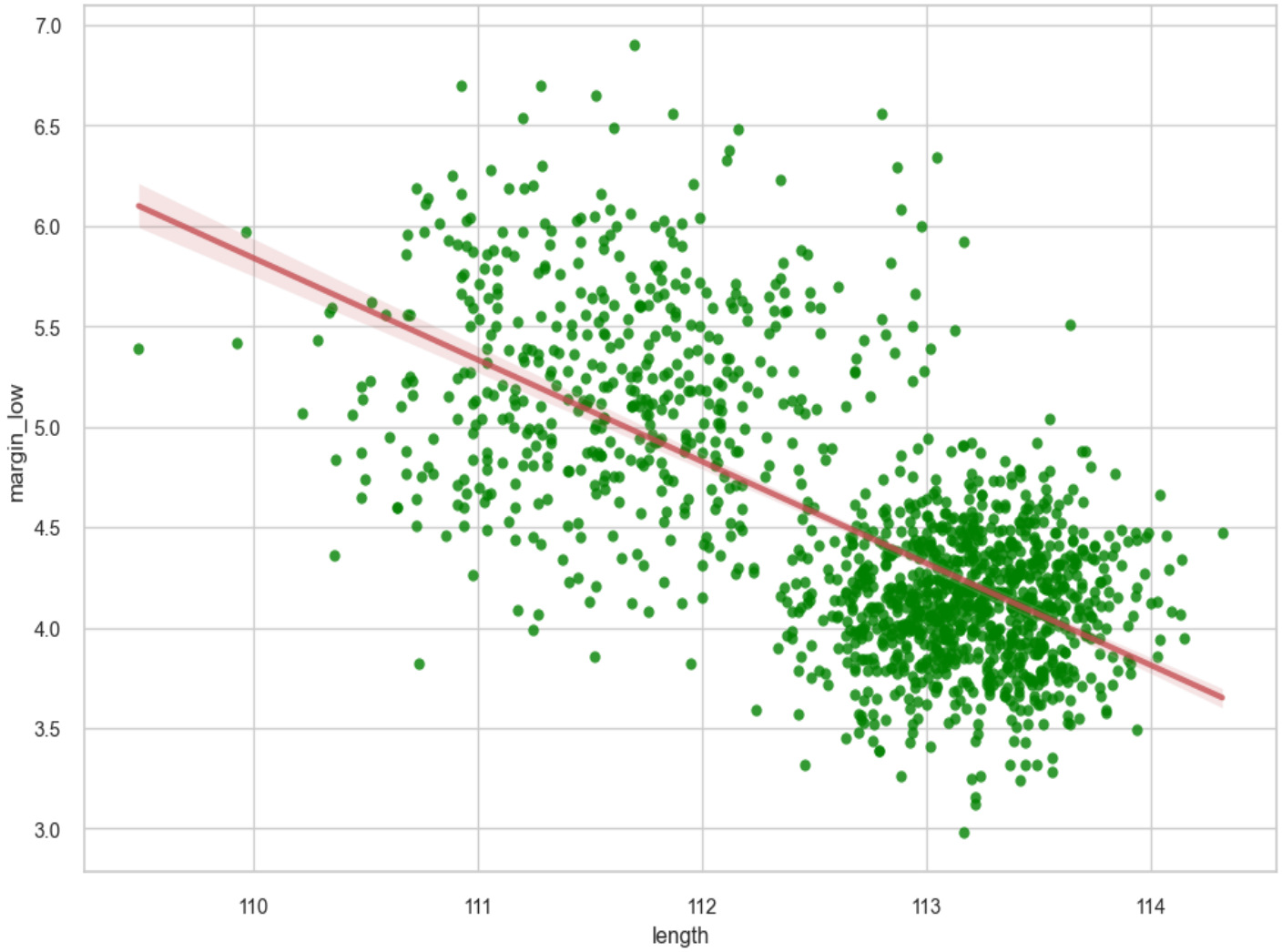
```
# Prédiction
y_pred=regLin.predict(X_test)
y_pred[:10]
```

```
array([4.98085775, 4.09270387, 4.31723716, 4.1326209 , 4.03282833,
       4.14758979, 4.88106518, 4.11765202, 4.06775573, 4.01286982])
```

2. Implémentation des valeurs manquantes
1. Régression linéaire simple : droite de regression

La droite de régression montre une relation linéaire négative entre length et margin_low.

Plus la longueur du billet (length) augmente, plus la marge basse (margin_low) tend à diminuer.



2. Implémentation des valeurs manquantes
1. Régression linéaire simple : évaluation du modèle

- Le coefficient de détermination R^2

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

```
R2_RLS = r2_score(y_test, y_pred)
R2_RLS
```

```
0.5136070700267961
```

Nous allons pouvoir **évaluer la performance de notre modèle**.
Pour cela nous allons commencer par calculer le **coefficient de détermination R^2** de notre modèle.

Le R^2 mesure la qualité de prédiction d'une régression linéaire et en cela va faciliter la comparaison entre différents modèles de régression linéaire.
Le R^2 correspond à l'erreur du modèle divisé par l'erreur d'un modèle basique qui prédit tout le temps la moyenne de la variable à prédire. Le score R2 est d'autant plus élevé que le modèle est performant, et vaut au maximum 1, lorsque toutes les prédictions sont exactes.

2. Implémentation des valeurs manquantes
1. Régression linéaire simple : évaluation du modèle

- Cross validation

Je peux obtenir les scores de ma cross validation avec la **fonction cross_val_score**, appliquée à l'ensemble de mes données X et y.
Je peux également, si je le souhaite effectuer ma cross validation dans le cadre d'**une boucle qui va se répéter autant de fois qu'il y a de folds**.

Je vais ainsi pouvoir calculer les scores suivant :

- la MAE (l'erreur absolue moyenne) est la moyenne arithmétique des valeurs absolues des écarts entre les valeurs prédites et les valeurs réelles.
- Le RMSE (l'erreur quadratique moyenne) est un coefficient de dispersion qui va nous donner une idée de la dispersion des écarts de prédiction.
- La MAPE (Mean Absolute Percentage Error) est la moyenne des écarts en valeur absolue par rapport aux valeurs observées. C'est donc un pourcentage et par conséquent un indicateur pratique de comparaison.

MAE = 0.34
MSE = 0.21
RMSE = 0.452
Median Absolute Error = 0.26

2. Implémentation des valeurs manquantes
2. Régression linéaire multiple : préparation des données

- Définition

En régression linéaire multiple, plusieurs variables explicatives sont combinées pour prédire une variable cible, chaque variable contribuant au modèle avec un coefficient reflétant son importance.

On souhaite cette fois expliquer, de manière linéaire, une variable Y (variable à expliquer), aléatoire en fonction de *p* variables (X1,…,Xp), et non plus d'une seule variable.

- Préparation des données

De la même façon que dans la régression linéaire simple, je définis :
- y comme les valeurs de ma variable à prédire (margin_low),
- X, cette fois, l'ensemble de mes variables explicatives, à partir desquelles je vais pouvoir prédire y.

2. Implémentation des valeurs manquantes
2. Régression linéaire multiple : préparation des données

```
# Régression Linéaire miltiple avec sklearn

X2 = Df_Billet_sans_nan.drop(columns = ['margin_low','is_genuine'])
y2 = Df_Billet_sans_nan.margin_low
```

J'applique les mêmes fonctions pour séparer mes données en un groupe d'entrainement et un groupe de test, puis on instancie notre modèle, on l'entraîne et on peut prédire les valeurs y_pred.

```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2,y2,test_size=0.2,random_state=42)
```

```
reg_multi=LinearRegression()
```

```
reg_multi.fit(X_train2,y_train2)
```

```
LinearRegression
LinearRegression()
```

```
# Prédiction
y_pred2=reg_multi.predict(X_test2)
y_pred2[:10]
```

```
array([4.95663909, 4.29838247, 4.37773033, 4.09528905, 4.02274623,
       4.09556812, 4.85333759, 4.12787564, 4.20096875, 3.86186446])
```

2. Implémentation des valeurs manquantes
2. Régression linéaire multiple : évaluation du modèle

Je vais ensuite calculer les coefficients R^2, MAE, RMSE MAPE et **comparer les deux modèles** pour choisir le plus performant.

```
R2_RLM = r2_score(y_test2, y_pred2)
R2_RLM
```

0.5457956683532407

```
MAE_RLM = mean_absolute_error(y_test2,y_pred2)
print('MAE =', mean_absolute_error(y_test2,y_pred2))
```

MAE = 0.3353706932764837

```
MSE_RLM = mean_squared_error(y_test2,y_pred2)

print('MSE =', mean_squared_error(y_test2,y_pred2))
```

MSE = 0.19144954947493334

```
RMSE_RLM = np.sqrt(mean_squared_error(y_test2,y_pred2))

print('RMSE =',np.sqrt(mean_squared_error(y_test2,y_pred2)))
```

RMSE = 0.4375494823159243

```
Median_Absolute_Error_M= median_absolute_error(y_test2,y_pred2)
print('Median Absolute Error =',median_absolute_error(y_test2,y_pred2))
```

Median Absolute Error = 0.2623802208694581

2. Implémentation des valeurs manquantes
2. Comparaison des deux modèles

Je vais ensuite calculer les coefficients R^2, MAE, RMSE MAPE et **comparer les deux modèles** pour choisir le plus performant.

	Régression Linéaire Simple	Régression Linéaire Multiple
R^2	0.513607	0.545796
MAE	0.344331	0.335371
MSE	0.205017	0.191450
RMSE	0.452788	0.437549
Median Absolute Error	0.262870	0.262380

⇒ Nous pouvons dire que la régression linéaire multiple donne de meilleurs résultats (légère amélioration des performances)

En effet, le R^2 est meilleur, ce qui est cohérent car nous prenons en compte plus de variables (corrélées).
Et d'autre part les erreurs de prédiction sont légèrement moindre.

Nous allons donc utiliser la régression linéaire multiple pour prédire nos valeurs manquantes sur la variable margin_low.

2. Implémentation des valeurs manquantes

2. Régression linéaire multiple : Prédiction des valeurs manquantes

Création d'un **nouveau Dataframe** avec uniquement les lignes contenant les valeurs manquantes.

Enregistrement des valeurs de la colonne is_genuine dans une variable 'is_genuine' pour pouvoir l'enlever et l'ajouter à nouveau plus tard.

Suppression des colonnes is_genuine et margin_low de mon DataFrame pour me retrouver uniquement avec l'ensemble de mes variables X.

Prédiction des valeurs margin_low manquantes avec la fonction predict appliquée à ce DataFrame contenant mes variables X.

Réintroduction de la variable is_genuine puis **concaténation de ce Dataframe** au Dataframe contenant toutes les lignes sauf celles ayant des valeurs manquantes

Obtention d'**un seul dataframe complet** (vérification de la cohérence des valeurs prédites)

3. Modélisations

1. Analyse en composantes principales

L'Enjeu de l'ACP a deux objectifs principaux.

Elle permet d'étudier :

- *La variabilité entre les individus, c'est-à-dire quelles sont les différences et les ressemblances entre individus.*
- *Les liaisons entre les variables : y a-t-il des groupes de variables très corrélées entre elles qui peuvent être regroupées en de nouvelles variables synthétiques ?*
- Ces objectifs seront possibles à travers la création de variable synthétique et en maximisant la variance de nos données. Dans le cadre d'une ACP centrée-réduite nous serons obligés de faire que la moyenne soit égale à 0 et l'écart-type à 1

3. Modélisations

1. Analyse en composantes principales

```
# On enregistre le nom de nos colonnes et de nos variables
names=Df_Billet2.index
features=Df_Billet2.columns
```

```
# On va entraîner et transformer nos données afin d'avoir une moyenne à 0 et une STD à 1

Scaler=StandardScaler()
```

```
X_Scaled=Scaler.fit_transform(X)
```

```
# On vérifie que Moyennes= 0 et STD=1
idx=["mean","std"]
# Création Dataframe de IDX
pd.DataFrame(X_Scaled).describe().round(2).loc[idx, :]
```

	0	1	2	3	4	5	6
mean	0.0	0.0	0.0	-0.0	-0.0	-0.0	0.0
std	1.0	1.0	1.0	1.0	1.0	1.0	1.0

```
# Nous allons travailler sur 7 composantes

n_components= 7
```

```
pca=PCA(n_components=n_components)
```

```
pca.fit(X_Scaled)
```

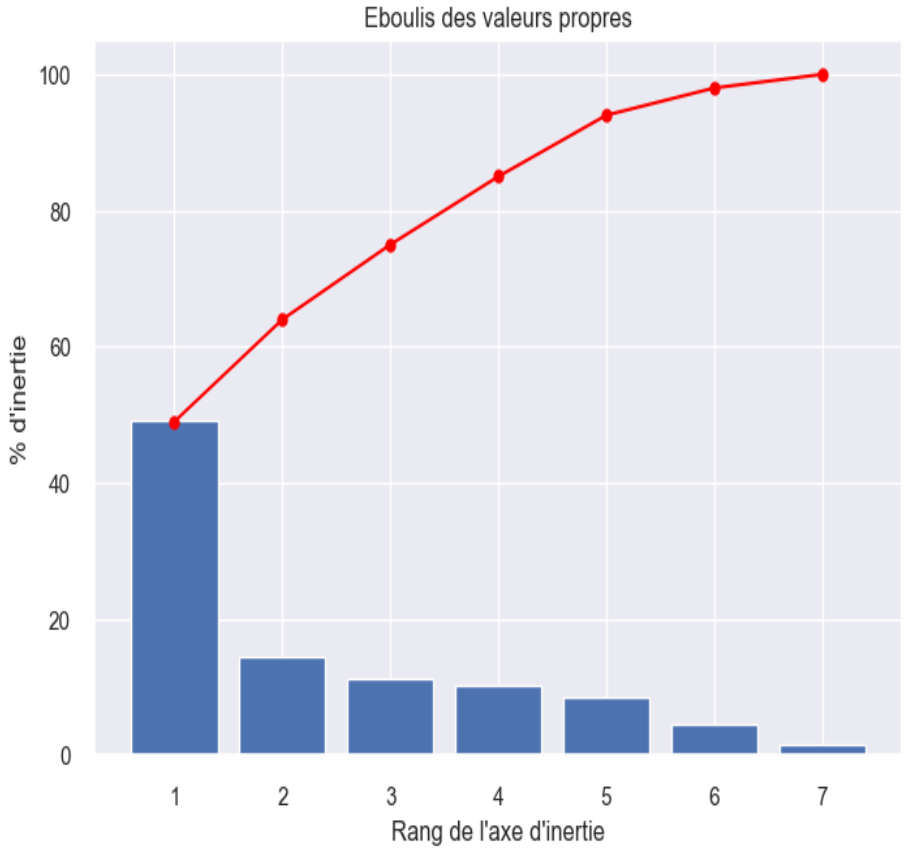
```
PCA
PCA(n_components=7)
```

```
# Recherche de la variance captée par chaque nouvelle composante.
pca.explained_variance_ratio_
```

```
array([0.49247095, 0.14570397, 0.11277902, 0.10227683, 0.08448045,
       0.04623784, 0.01605093])
```

3. Modélisations

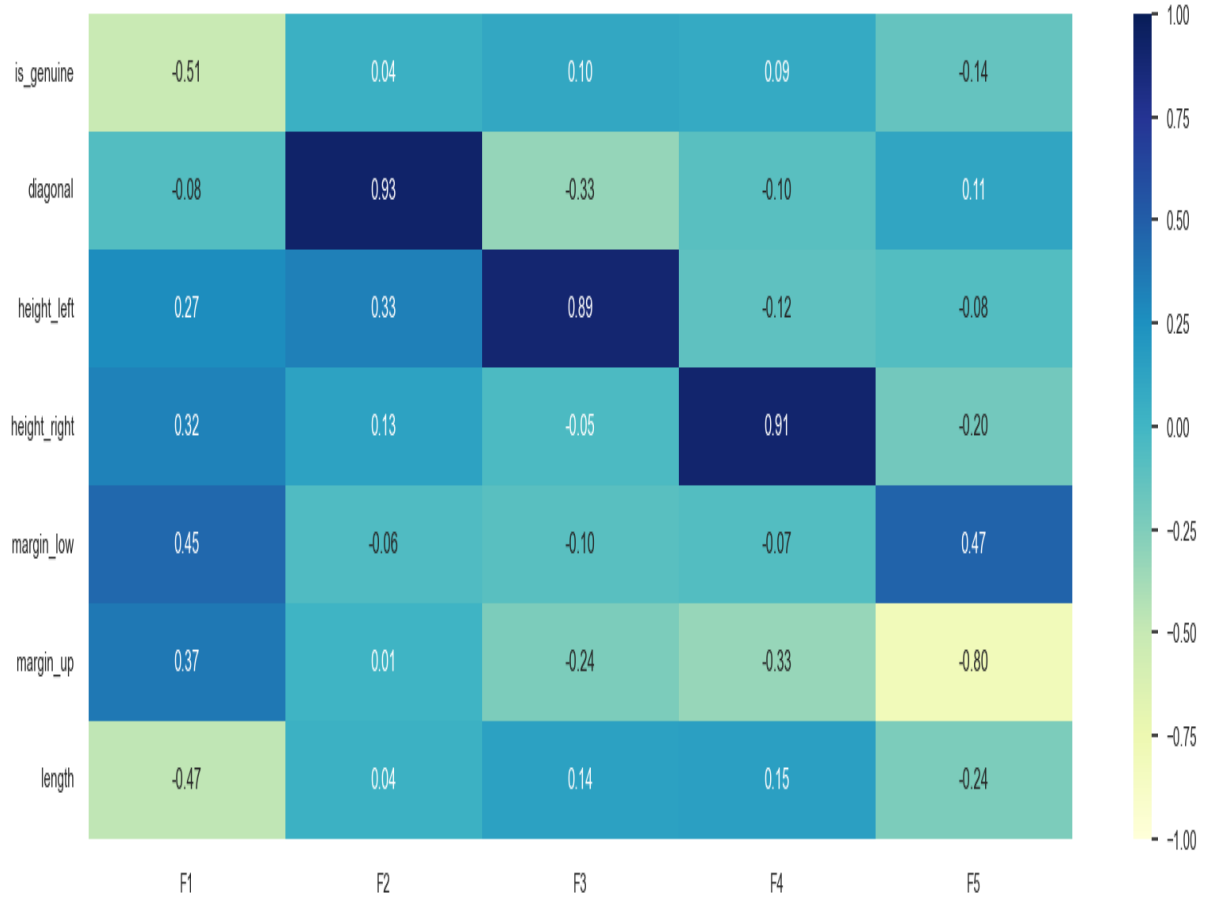
1. Analyse en composantes principales: valeurs propres



Le graphique de l'éboulis montre la proportion de variance expliquée par chaque composante principale. Les premières composantes capturent la majorité de l'information dans les données.

3. Modélisations

1. Analyse en composantes principales



Cette matrice montre l'importance de chaque variable dans la construction des composantes principales. Les valeurs absolues des coefficients indiquent l'influence des variables sur les composantes.

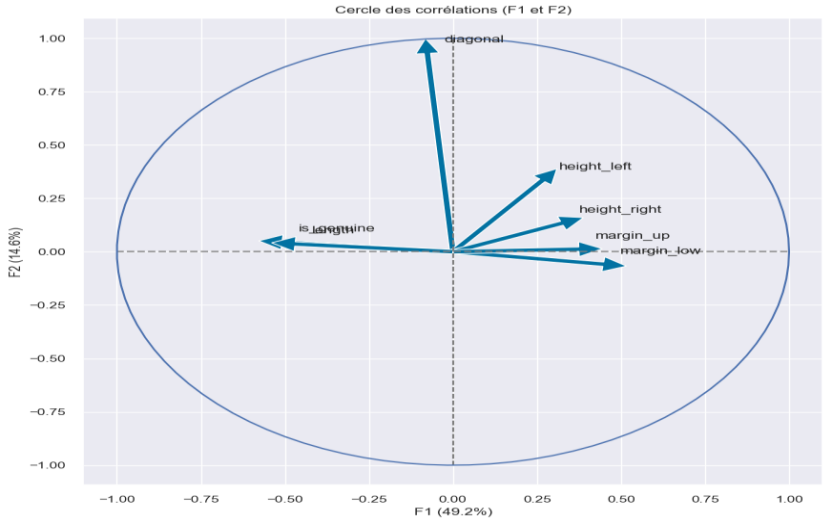
La première composante principale (F1) est fortement influencée par margin_low (0.45) et length (-0.47), indiquant que ces deux variables jouent un rôle clé dans la distinction des observations.

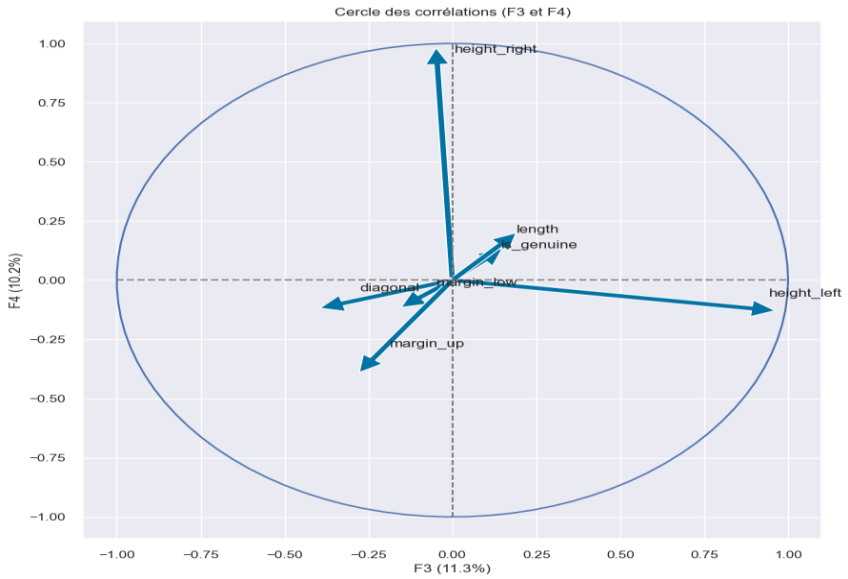
La deuxième composante principale (F2) est principalement liée à diagonal (0.93), ce qui reflète son importance indépendante des autres variables.

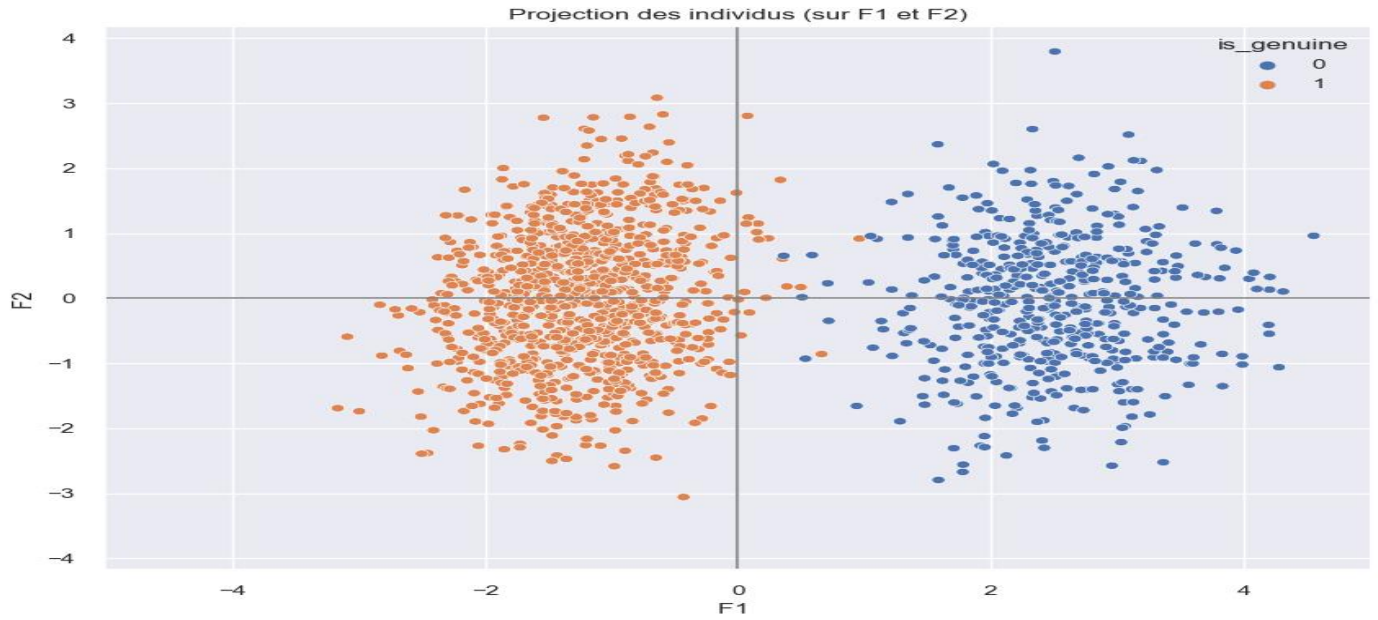
Les composantes suivantes (F3, F4, F5) capturent des variations plus spécifiques, comme l'opposition entre height_left et height_right sur F3.

3. Modélisations

1. Analyse en composantes principales: cercles de corrélation







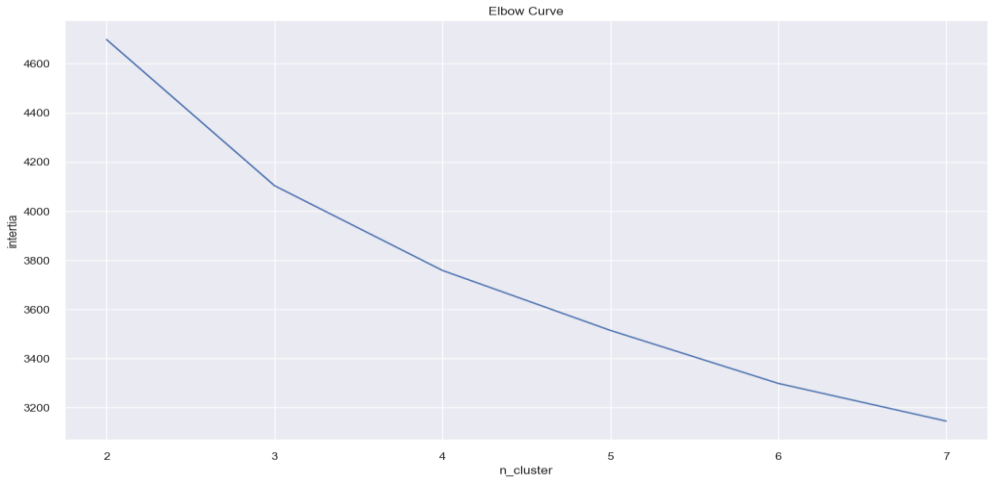
La projection des individus sur les deux premières composantes principales (F1 et F2) met en évidence une bonne séparation entre les vrais billets (orange) et les faux billets (bleu).

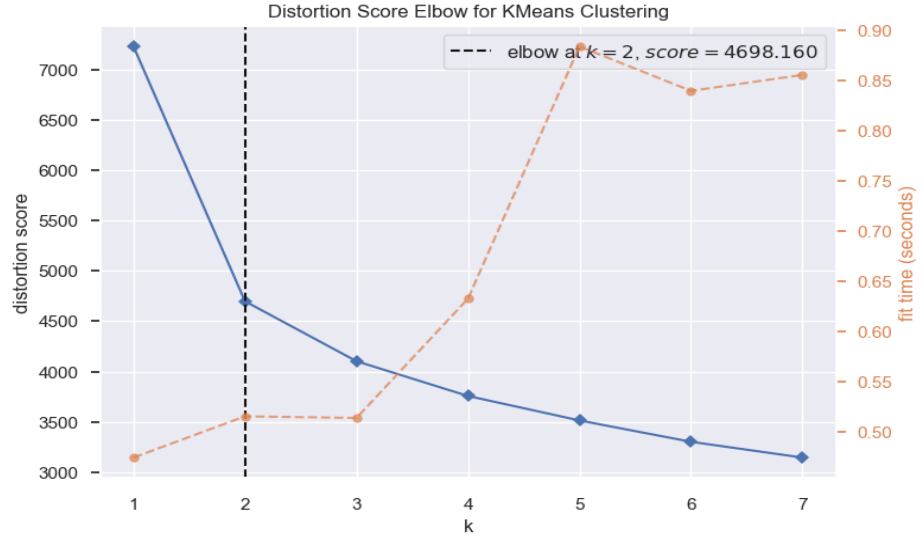
Cette séparation montre que l'ACP réussit à réduire la dimensionnalité tout en préservant les distinctions importantes dans les données.

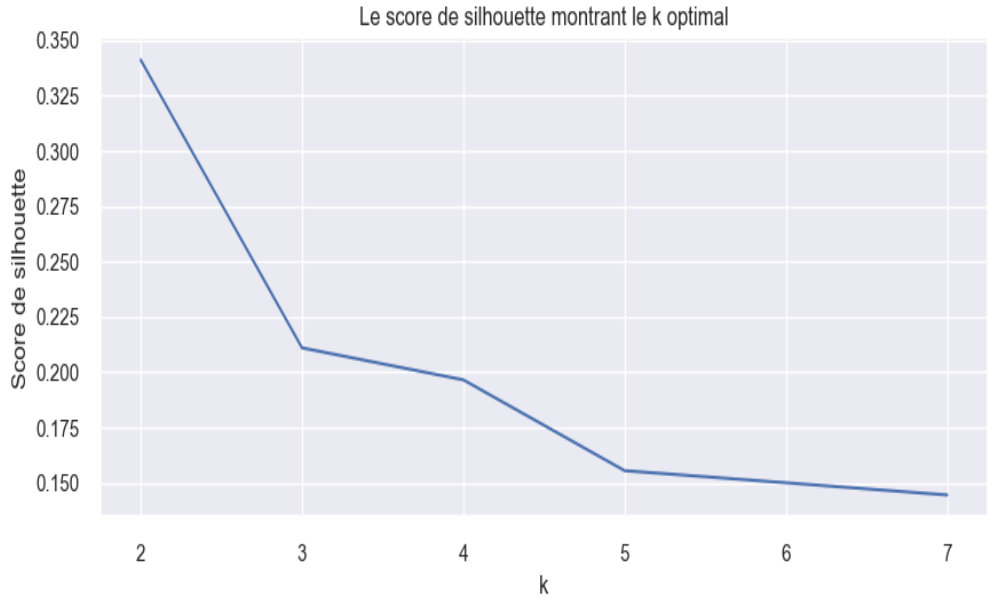
Les vrais billets sont principalement regroupés dans la zone positive de F1, tandis que les faux billets se trouvent dans la zone négative.

3. Modélisations

2. Clustering KMEANS: détermination du k optimal



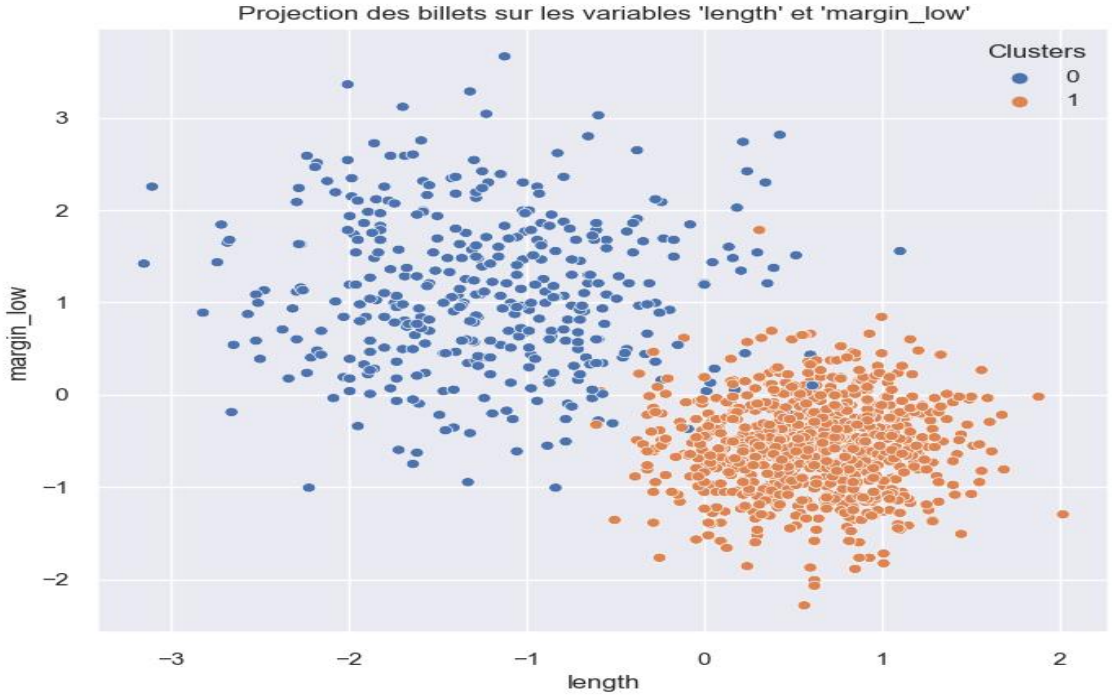




- Le choix du nombre de clusters (k) est déterminé par la méthode du coude et le score silhouette.
- k=2 est retenu comme valeur optimale car il minimise l'inertie tout en maximisant la qualité des clusters.

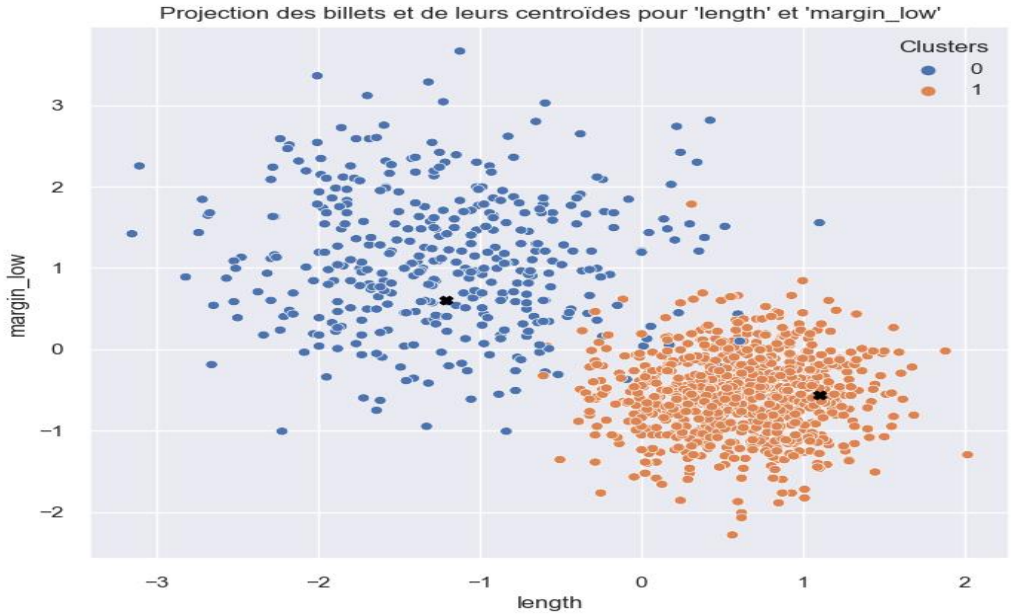
3. Modélisations

2. Clustering KMEANS: projection des individus



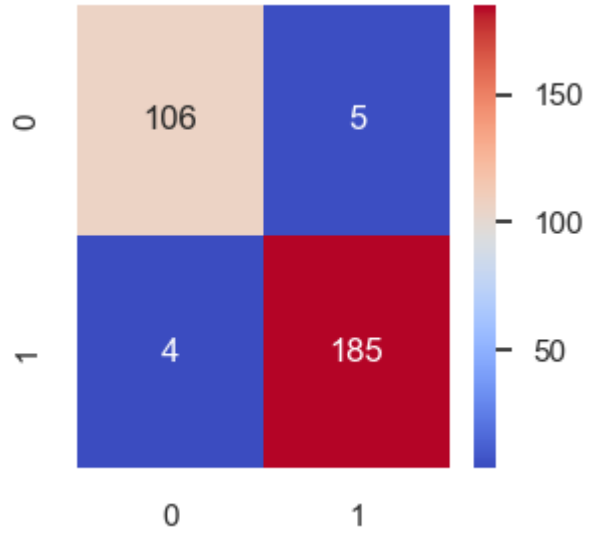

```
centroids.head()
```

	diagonal	height_left	height_right	margin_low	margin_up	length
Cluster_0	-0.195963	0.578520	0.726474	1.103768	0.847527	-1.212909
Cluster_1	0.114598	-0.272365	-0.350753	-0.558303	-0.425956	0.602171



- La projection montre les individus regroupés en deux clusters bien distincts sur les variables length et margin_low.
- Les centroïdes marquent les centres des clusters et résument leurs caractéristiques moyennes.

3. Modélisations
2.1. Clustering KMEANS: Évaluation du modèle K-means



Le modèle montre une excellente séparation entre les vrais et faux billets, avec seulement 9 erreurs (4 faux négatifs et 5 faux positifs).

```
accuracy_kmeans=accuracy_score(y_testK,y_predK)*100
print("Accuracy :",accuracy_kmeans)
```

Accuracy : 97.0

```
precision_kmeans=precision_score(y_testK,y_predK).round(4)*100
print("Précision :", precision_kmeans)
```

Précision : 97.37

```
recall_kmeans=recall_score(y_testK,y_predK).round(4)*100
print("Recall :",recall_kmeans)
```

Recall : 97.88

```
f1_score_kmeans=f1_score(y_testK,y_predK).round(4)*100
print("F1_score :",f1_score_kmeans)
```

F1_score : 97.63

3. Modélisations
3. Régression logistique

La **régression logistique** est un modèle statistique permettant d'étudier les relations entre un ensemble de variables prédictives (explicatives) Xi et une variable à prédire Y. C'est-à-dire qu'à partir des variables prédictives, elle va pouvoir prédire les valeurs prises par la variable catégorielle y à prédire.

Quelle différence y a-t-il entre la régression logistique et la régression linéaire ?

La régression logistique est utilisée lorsque la variable réponse est catégorique, binaire, comme oui/non, vrai/faux. C'est un algorithme de classification et elle ne nécessite pas la présence d'une relation linéaire entre les variables.

La régression linéaire, quant à elle, est utilisée lorsque la variable réponse est continue (comme les valeurs prises par margin_low).

3. Modélisations
3.1. Régression logistique

Paramétrage : le **Grid search** est une méthode d'optimisation qui va nous permettre de tester une série de paramètres (hyperparamètres) et de comparer les performances pour en déduire le meilleur paramétrage.
Nous allons choisir un score à optimiser et le grid search va tester chaque combinaison d'hyperparamètres et retenir celle qui va donner le meilleur score.
Ensuite nous pourrons lancer notre modèle avec ces meilleurs paramètres.

```
estimateur = LogisticRegression()

score='accuracy'
param_grid = {'solver' : ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'],
              'penalty' : ['l1', 'l2', 'elasticnet', None],
              'C':[0.001, 0.01, 0.1, 1, 10, 100, 1000]}

grid = GridSearchCV(estimateur,param_grid, cv=5,scoring=score,verbose=0)
grid.fit(X_train_log, y_train_log)
```

```
print("Meilleurs paramètres :",grid .best_params_)
print("Meilleur score (accuracy):",round(grid .best_score_,4))
```

```
Meilleurs paramètres : {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
Meilleur score (accuracy): 0.9917
```

```
modelRegLog=grid.best_estimator_
round(modelRegLog.score(X_test_log,y_test_log),4)*100
```

```
99.0
```

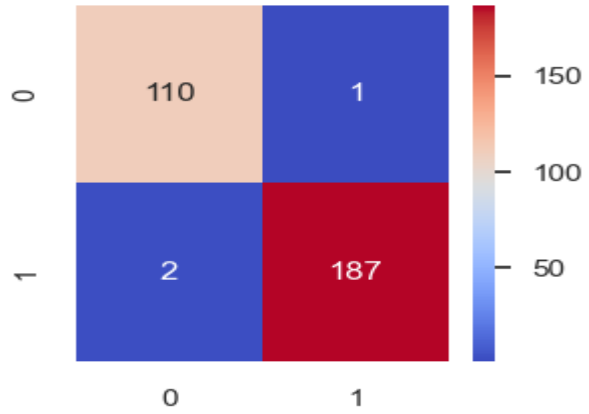
3. Modélisations
3.2. Régression logistique

Une fois que nous avons **instancié** et **entrainé** notre modèle, que nous avons **prédit y_pred à partir de X_test**, nous pouvons **tester la probabilité pour chaque billet de X_test, d'appartenir à la classe fausse (0) ou vraie (1).**
Ici pour le premier billet par exemple, il y a 99% de probabilité qu'il appartienne à la classe fausse et 0,1% à la classe vraie. Pour le second, 82% à la classe fausse et 0,18% à la classe vraie.

```
y_prob = grid.predict_proba(X_test_log).round(2)
y_prob
```

```
array([[0.99, 0.01],
       [0.82, 0.18],
       [0.01, 0.99],
       [0.02, 0.98],
```

On enchaîne sur la **matrice de confusion** qui nous donne de très bons résultats




```
accuracy_Reglog=accuracy_score(y_test_log,y_pred_log)*100
print("Accuracy :",accuracy_Reglog)
```

```
Accuracy : 99.0
```

```
precision_Reglog=precision_score(y_test_log,y_pred_log).round(4)*100
print("Précision :", precision_Reglog)
```

```
Précision : 99.47
```

```
recall_Reglog=recall_score(y_test_log,y_pred_log).round(4)*100
print("Recall :",recall_Reglog)
```

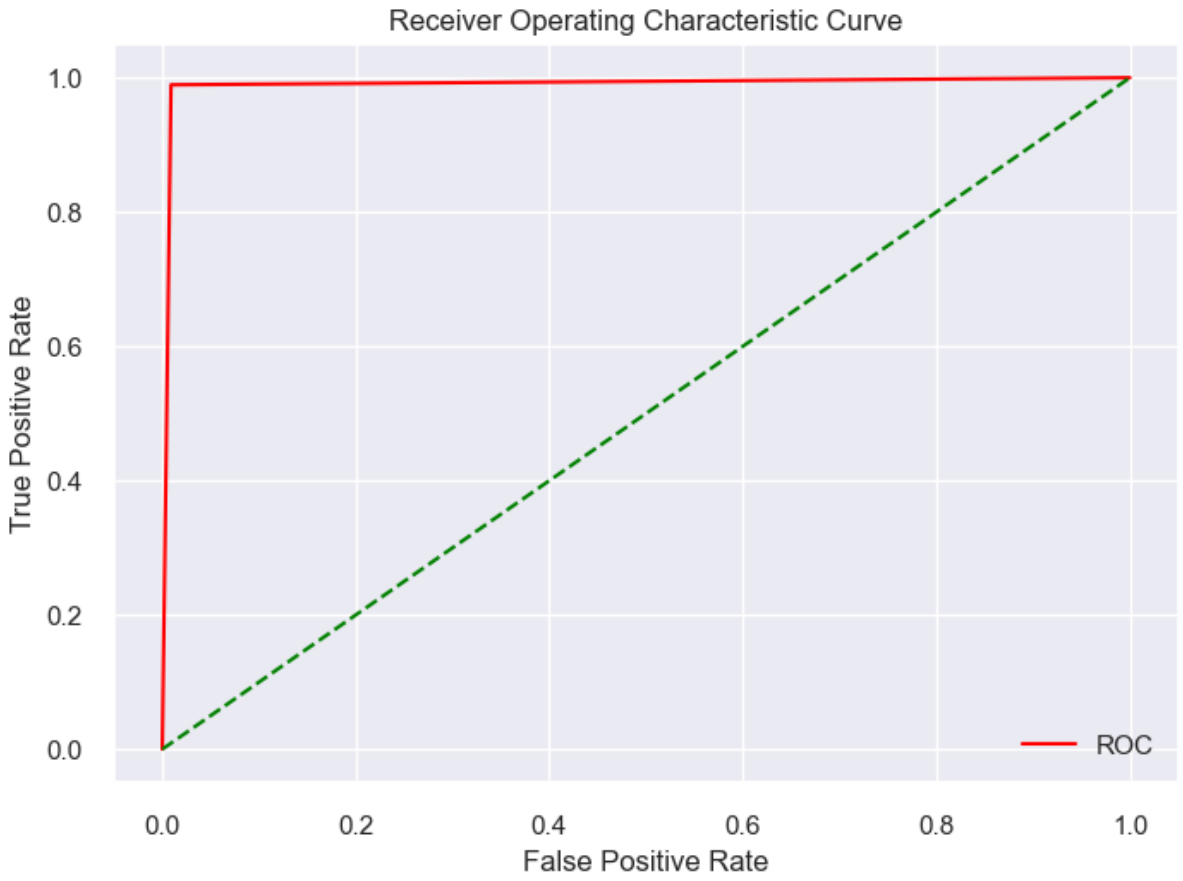
```
Recall : 98.94
```

```
f1_score_Reglog=f1_score(y_test_log,y_pred_log).round(4)*100
print("F1_score :",f1_score_Reglog)
```

```
F1_score : 99.2
```

3. Modélisations

3.3. Régression logistique: Courbe ROC et AUC




```
# Calculons la performance de notre modèle avec l'AUC ROC.
# L'AUC ROC va mesurer la performance globale de notre modèle, il correspond à l'air sous la courbe

false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test_log,y_pred_log)
RegLog_auc=auc(fper, tper).round(4)*100
print("L'AUC ROC de notre Regression Logistique est de :",RegLog_auc)
```

```
L'AUC ROC de notre Regression Logistique est de : 99.02
```

Avec une AUC de 99.02%, la régression logistique démontre une excellente performance, proche de la perfection.

=> Notre modèle est presque parfait.

La courbe ROC illustre la capacité de différenciation des classes par le modèle.

3. Modélisations
4. K-Nearest Neighbors (KNN)

KNN est un algorithme supervisé basé sur la proximité : il classe les observations en fonction de leurs voisins les plus proches dans l'espace des caractéristiques.

Le choix du paramètre k (nombre de voisins) influence directement les performances du modèle.

Grid search sur une cross validation
=> *définition du nombre de voisins*

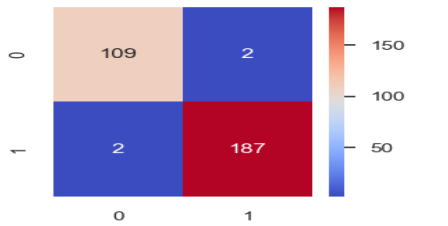
→ **Instanciation** du modèle avec le meilleur k

→ **Entrainement** du modèle sur nos données train

→ **Prédictions** y_pred à partir de X_test

↓

Matrice de confusion



← Calcul des scores de performance avec la boucle Kfold :

- Accuracy : 98,67%
- Precision : 98,94%
- Rappel : 98,94%

← AUC ROC de notre modèle :
AUC ROC : 98.57

Les performances semblent similaires aux autres modèles avec des erreurs limitées

4. Comparaison globale des modèles

Nous allons maintenant, après avoir testé différentes méthodes de classification, comparer les scores de performance obtenus.

	Kmeans	Régression Logistique	KNN
Accuracy	97.00	99.00	98.666667
Precison	97.37	99.47	98.940000
Recall	97.88	98.94	98.940000
F1_Score	97.63	99.20	98.940000
Auc_Roc	96.69	99.02	98.570000

L'ensemble des modèles donne de très bons résultats.

La régression logistique obtient les meilleurs scores, suivie par KNN et K-means.

Nous allons donc utiliser ce modèle pour notre application finale, c'est-à-dire pour prédire l'authenticité ou non de nouveaux billets.

5. Application finale

Prédiction sur de nouvelles données avec la régression logistique

APPLICATION DIRECTE SUR LE NOTEBOOK