ONCFM

Organisation nationale de lutte contre le faux-monnayage

ALGORITHME DE DÉTECTION DES FAUX BILLETS

ONCFM

Sommaire

-Contexte et demande
-Méthodologie
-Traitements et analyses réalisés
-Pistes explorées pour la construction de l'algorithme
-Modèle final retenu

Contexte et demande

- L'**Organisation nationale de lutte contre le faux-monnayage** : organisation publique ayant pour objectif de mettre en place des méthodes d'identification des contrefaçons des billets en euros
- Les **caractéristiques géométriques** de chaque billet sont mesurées et consignées par une machine
- Demande : construire un algorithme capable de définir si un billet est un vrai ou un faux à partir de ses caractéristiques géométriques
- **Objectif : identifier un maximum de faux billets au sein de la masse de billets analysés chaque jour**

ONCFM

Méthodologie

- **DONNEES** :
  - 1500 billets
  - six informations géométriques, en millimètres : length, height_left, height_right, margin_up, margin_low, diagonal
  - étiquettes : 1000 billets vrais, 500 billets faux

    ➔ Algorithmes de prédiction vrai / faux à partir des caractéristiques géométriques
- **TECHNOLOGIES** : utilisation du notebook Jupyter, du langage de programmation **Python** et de ses librairies : Numpy, Pandas, Matplotlib, Seaborn, Statsmodels, Scikit Learn, Scipy

Méthodologie

- **ANALYSE DESCRIPTIVE** DE L'ECHANTILLON
- **TRAITEMENT** (valeurs nulles)
- **MODELISATION**
  - REGRESSION LOGISTIQUE ➔ algorithme de classification supervisée
  - KMEANS ➔ algorithme de clustering
- **EVALUATION** DES MODELES (matrice de confusion)
- **SELECTION** D'UN MODELE

Analyse descriptives et traitements réalisés

Analyse descriptive de l'échantillon

Fonction « describe » de Python ➔

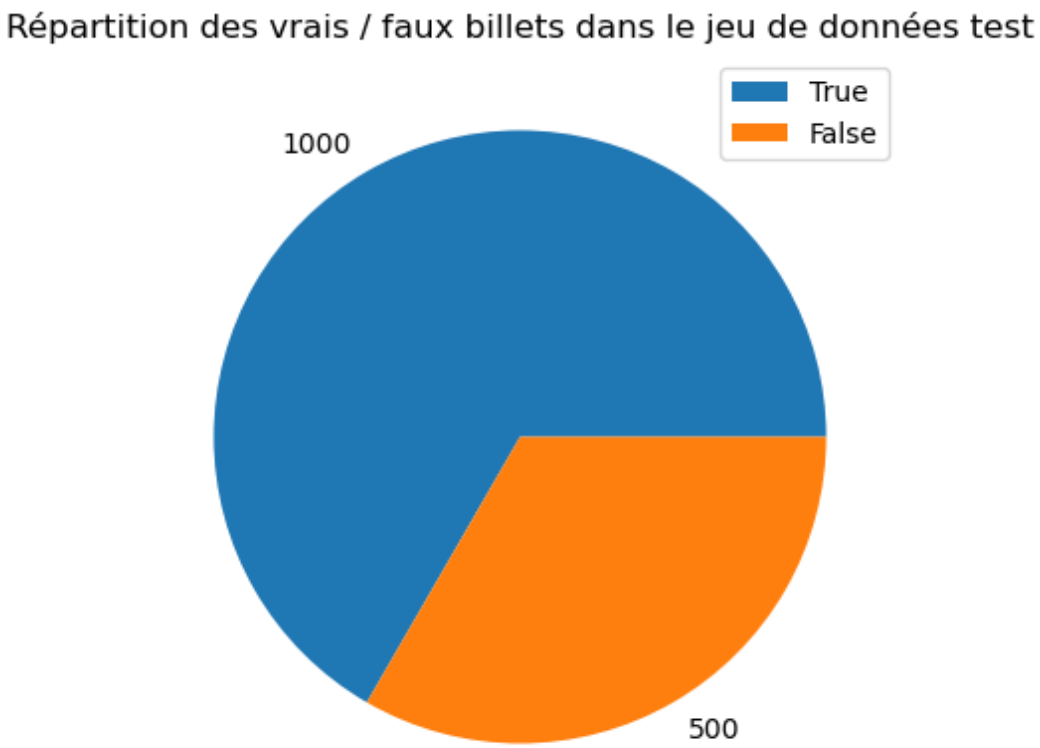
MESURES STATISTIQUES DESCRIPTIVES DE L'ECHANTILLON

	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length
count	1500	1500	1500	1500	**1463**	1500	1500
unique	2						
top	True						
freq	1000						
mean		171.95	104.02	103.92	4.48	3.15	112.67
std		0.305195	0.29	0.32	0.66	0.23	0.87
min		171.04	103.14	102.82	2.98	2.27	109.49
25%		171.75	103.82	103.71	4.01	2.99	112.03
50%		171.96	104.04	103.92	4.31	3.14	112.96
75%		172.17	104.23	104.15	4.87	3.31	113.34
max		173.01	104.88	104.95	6.90	3.91	114.44

ONCFM

Analyse descriptive 1500 billets : variable is genuine

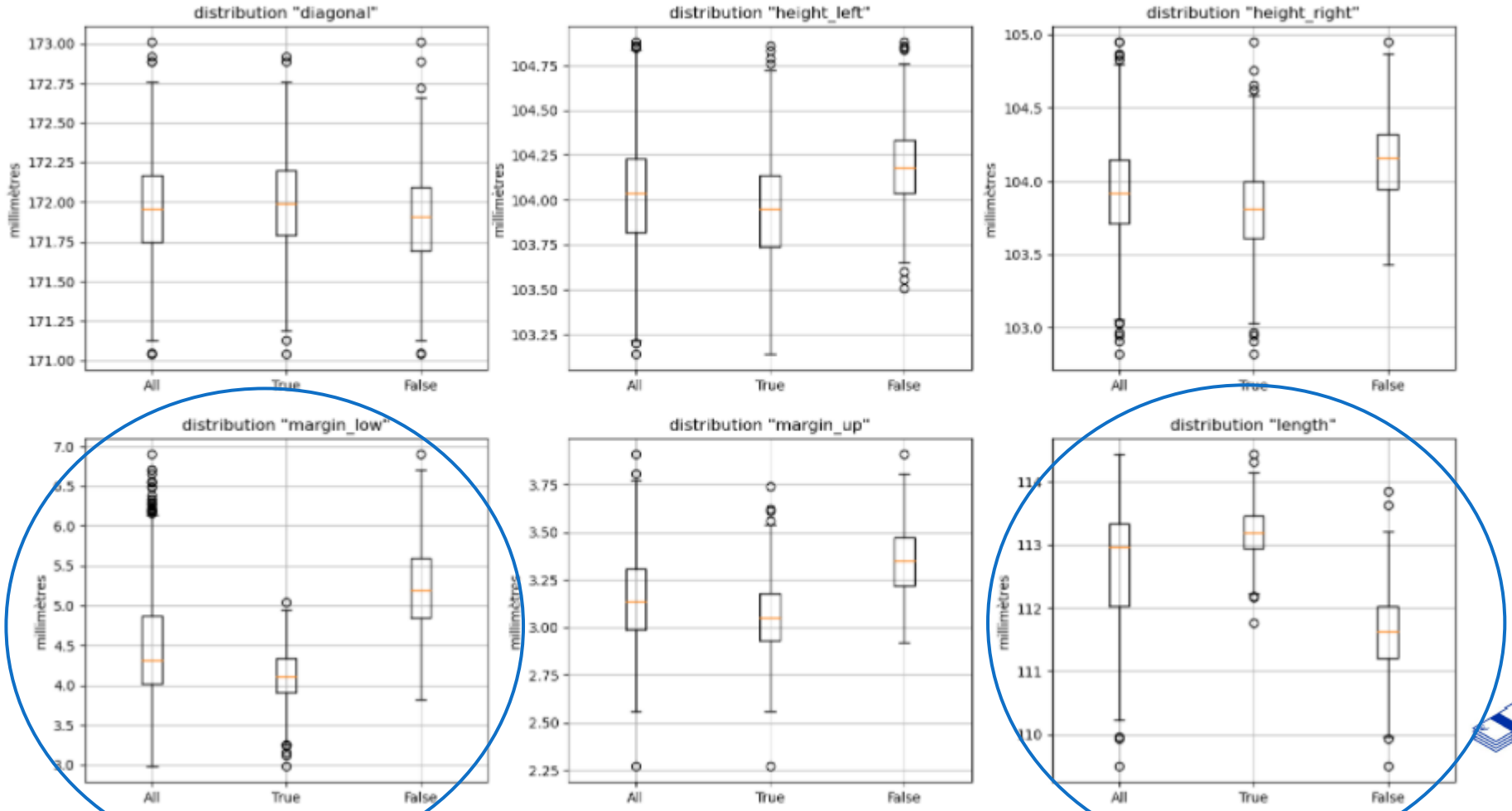
	is_genuine
count	1500
unique	2
top	True
freq	1000



ONCFM

Analyse descriptive 1500 billets : distribution des variables dimension

Note : les outliers sur les variables « length » et « margin low » sont tous catégorisés comme « faux billets »

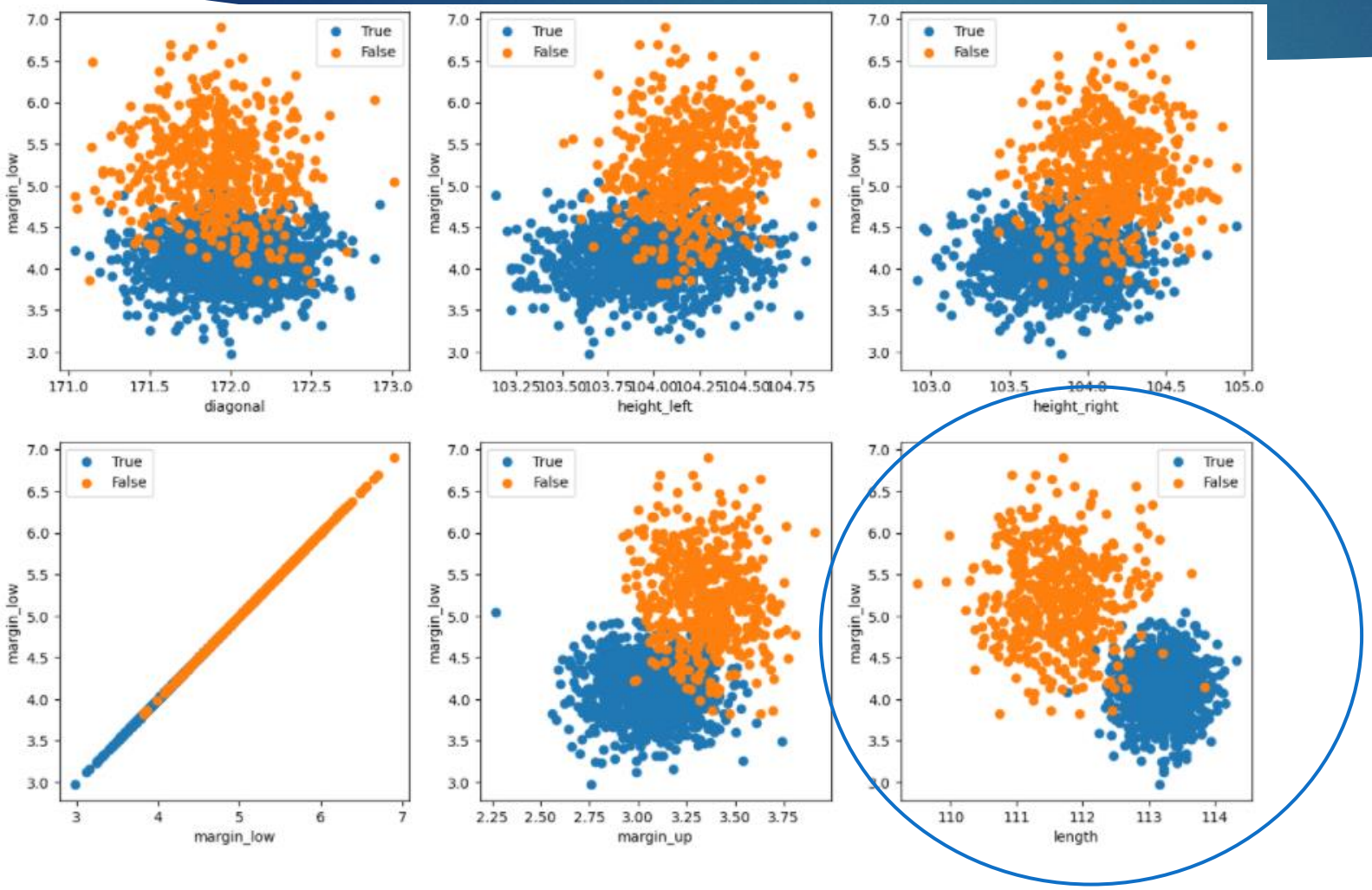


ONCFM

Traitements : recherche des doublons et valeurs nulles

- Recherche des doublons avec la fonction « duplicated » : **aucun doublon**
- La fonction « info» appliquée sur le dataframe confirme la présence de **valeurs nulles** sur la variable « margin low » (2,5% des valeurs)
  - Une approche possible pour remplacer ces valeurs nulles consiste à utiliser la régression linéaire, simple ou multiple.
  - Dans un premier temps, nous allons rechercher les corrélations linéaires existantes entre "margin_low" et les autres variables dans le but de réaliser une régression linéaire simple

Remplacement des valeurs nulles : recherche des corrélations linéaires



Coefficients de corrélation de Pearson	margin_low
diagonal	-0.024492
height_left	0.242279
height_right	1.000000
margin_low	0.391085
margin_up	0.307005
length	**-0.401751**

ONCFM

Remplacement des valeurs nulles : régression linéaire simple

- On réalise une regression linéraire simple en utilisant la variable "length", qui est la plus corrélée linéairement avec la variable "margin_low", d'après l'étude des coefficients de corrélation de Pearson
- Utilisation de l'objet regression linéaire de Statsmodels sur les lignes où margin low est non nulle :
  - Le **paramètre β1** (intercept) est environ égal à 62. Le paramètre **β2** est environ égal à -0,5
  - Les **p-values** sont inférieures à 5 %. À un niveau de test de 5 %, on considère que les paramètres sont significativement différents de 0

    ➔ *On conclut que la variable "length" est significative*
  - Le **coefficient de détermination** est de l'ordre de 0,44
  - La **RMSE** est de 0,49

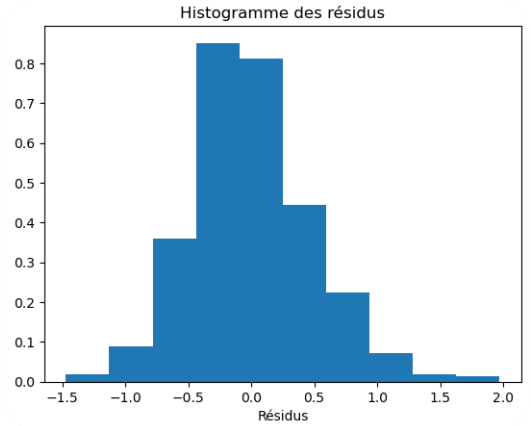
Remplacement des valeurs nulles : régression linéaire multiple

- On réalise une régression linéraire multiple en utilisant l'ensemble des variables de dimension
- Utilisation de l'objet régression linéaire de Statsmodels sur les lignes où margin low est non nulle :
  - On remarque que tous les paramètres sont significatifs, d'après les **p-values** obtenues suite aux tests de Student
  - Le **coefficient de détermination** est de l'ordre de 0,48 (meilleur que pour la régression linéaire simple)
  - La **RMSE** est de 0,48 (légèrement meilleur que pour la régression linéaire simple)

➔ à ce stade, on peut considérer que la régression linéaire multiple donne de meilleurs résultats que la régression linaire simple

Remplacement des valeurs nulles : évaluation du modèle

- La régression linéaire multiple obtient un meilleur score RMSE que **l'approche naïve** (remplacement des valeurs par la moyenne)
- Le calcul des coefficients **VIF**, qui sont tous proches de 1, indique qu'il n'y a pas de problème de colinéarité entre variables
- **L'histogramme des résidus à peu près centré et symétrique**
- En revanche les tests concernant **l'homoscédasticité** des résidus et leur **normalité** indiquent un rejet des hypothèse de constance des variances et de normalité de la distribution
  - ➔ *Au vu des résultats du coefficient de détermination, des résultats sur l'étude de l'homoscédasticité et du test de la normalité des résidus, on peut se poser la question de la pertinence du remplacement des valeurs par une régression linéaire.*
  - ➔ *Une autre approche possible serait de supprimer les individus pour lesquels les valeurs sont manquantes.*



ONCFM

Pistes explorées pour la construction du modèle

Régression logistique

Pistes explorées pour la construction de l'algorithme : régression logistique

- L'algorithme de **régression logistique** permet de **classifier** une variable binaire prenant ses valeurs dans {0,1} (ici genuine True or False) à l'aide de variables explicatives (ici les variables de dimensions)
- Etape 1 : création d'un **jeu d'entraînement** et un **jeu de test** à l'aide de la fonction **train_test_split de Scikit Learn**. On choisit de stratifier les jeux d'entraînement et de test
- Etape 2 : création d'un objet **régression logistique** à l'aide de la librairie **Statsmodels** à partir du jeu d'entraînement :
  - Les valeurs des p-value pour les paramètres « height right » « diagonal» « height left » sont supérieures à 0,05, on ne rejette donc pas l'hypothèse selon laquelle le paramètre est égal à 0. Ces paramètres sont donc considérés comme non significatifs et retirés du modèle.
  - Les paramètres pour «length » et « margin up » et « margin low» sont eux significativement différents de 0 et conservés dans le modèle.

Régression logistique : évaluation du modèle

- Etape 3 : Application du modèle sur le **jeu de test**
- *On obtient, pour chaque point, la probabilité qu'il soit positif, on décide de fixer un seuil à* ***> 0,5*** *pour déterminer qu'un point est considéré positif*
- Etape 4 : **Evaluation** du modèle

Matrice de confusion régression logistique *(générée par l'objet confusion_ matrix de scikit learn)*			
True negative	199	1	False positive
False Negative	2	98	True positive

- **Proportion de points correctement prédits**
  - Test accuracy = 0,99
- **Taux de vrais positifs**
  - Sensibilité = 0,98
- **Taux de vrais négatifs**
  - Spécificité = 0,995

ONCFM

Algorithme de régression logistique

- Etape 5 : définition d'une fonction contenant l'algorithme et générant un dataframe de résultats

```
def pred_regression_log (csv) :
    dataframe = pd.read_csv(csv)
    dataframe = dataframe.dropna()
    dataframe = dataframe.loc[:,["id","margin_low","margin_up","length"]]
    prob_faux_billet = reg_log.predict(dataframe)
    faux_billet = list(map(round,prob_faux_billet))
    dataframe["prob_faux_billet"] = prob_faux_billet
    dataframe["faux_billet?"] = faux_billet
    dataframe["faux_billet?"] =  dataframe["faux_billet?"].replace(to_replace=[1, 0], value = ["Oui", "Non"])
    return dataframe
```

pred_regression_log("billets_production.csv")

id	margin_low	margin_up	length	prob_faux_billet	faux_billet?
A_1	5.21	3.30	111.42	0.999999	Oui
A_2	6.00	3.31	112.09	1.000000	Oui
A_3	4.99	3.39	111.57	0.999995	Oui
A_4	4.44	3.03	113.20	0.000333	Non
A_5	3.77	3.16	113.33	0.000004	Non

ONCFM

Kmeans

Pistes explorées pour la construction de l'algorithme : kmeans

- L'algorithme de **kmeans** permet de créer des **clusters** à partir de variables d'étude
- Etape 1 : définition du nombre de **clusters** souhaités, ici **2** (vrais et faux billets)
- Etape 2 : **création** et **entraînement** d'un objet **kmeans** à l'aide de la librairie **Scikit Learn** :
- Etape 3 : récupération des **clusters attribués à chaque individu** et des coordonnées des **centroïdes**
- Etape 4 : **comparaison des étiquettes** vrais (0) faux (1) aux clusters attribués aux points par le kmeans
  - L'algorithme a arbitrairement attribué le numéro de cluster « 1 »

  au cluster « vrais billets » et « 0 » au « faux billets » ➔

  Pour palier cela, on inversera l'ordre

  des deux centroïdes calculés par l'objet Kmeans
  - *On note au passage qu'on a 4% des faux billets classés par le kmeans*

  dans le cluster "vrais billets" => false negative et 0.3% des vrais billets classés

  par le kmeans dans le cluster "faux billets" => false positive

Crosstab

clusters_km is_genuine	0	1	All
0	2	798	800
1	384	16	400
All	386	814	1200

ONCFM

Pistes explorées pour la construction de l'algorithme : kmeans

- Etape 5 : création d'une fonction permettant le **calcul de la distance euclidienne** entre un point donné et les clusters afin de déterminer le cluster le plus proche à l'aide de la fonction dist de la librairie math
- Etape 6 : Application de la fonction sur le **jeu de test**

```
def predict_kmeans(data):
    centroids, data = np.array(centroids_after_kmeans_get_centroides), np.array(data)
    distances = []
    for unit in data:
        for center in centroids:
            distances.append(dist(unit, center))
    distances = np.reshape(distances, [len(data),n_clust])
    closest_centroid = [np.argmin(dist) for dist in distances]
    return(closest_centroid)
```

```
predictions_knn = predict_kmeans(X_test)
```

ONCFM

Kmeans: évaluation du modèle

- Etape 7 : **Evaluation** du modèle

Matrice de confusion Kmeans *(générée par l'objet confusion_ matrix de scikit learn)*			
True negative	199	1	False positive
False Negative	3	97	True positive

- **Proportion de points correctement prédits**
  - Test accuracy = 0,986
- **Taux de vrais positifs**
  - Sensibilité = 0,97
- **Taux de vrais négatifs**
  - Spécificité = 0,995

Algorithme Kmeans

- Etape 8 : définition d'une fonction contenant l'algorithme et générant un dataframe de résultats

```
def pred_kmeans(csv) :
    dataframe = pd.read_csv(csv)
    dataframe = dataframe.dropna()
    data= dataframe.loc[:,["diagonal","height_left","height_right","margin_low","margin_up","length"]]
    faux_billet = predict_kmeans(data)
    dataframe["faux_billet?"] = faux_billet
    dataframe["faux_billet?"] =  dataframe["faux_billet?"].replace(to_replace=[1, 0], value = ["Oui", "Non"])
    first_column = dataframe.pop("id")
    dataframe.insert(0, 'id', first_column)
    return dataframe
```

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

ONCFM

Les résultats concernant l'**accuracy** et la **sensibilité** sont très légèrement supérieurs avec la régression logistique

Application finale : pred_regression_log()

ONCFM

MERCI