

# Présentation du projet 10 : Déetectez des faux billets avec Python



Préparé par :  
**MAGHLAZI Hanane**

# Projet 10 : Déetectez des faux billets avec Python

- **Introduction**
- **Objectif**
- **Analyse bi variée et multivariée des données :**
  - ✓ **Analyse bi variée et multivariée**
  - ✓ **Test ANOVA sur les variables pertinentes**
- **Gestion des valeurs manquantes par régression linéaire**
- **ACP**
- **Projection des données selon les clusters**
- **Division de l'ensemble des données : données d'entraînement et de test**
- **Construction de plusieurs modèles à partir de différents algorithmes de Machine Learning**
  - ✓ **KNN**
  - ✓ **Random Forest**
  - ✓ **Régression logistique**
  - ✓ **Kmeans**
- **Evaluation du modèle final**
  - ✓ **Matrices de confusion**
- **Evaluation d'un algorithme de classification qui retourne des scores**
  - ✓ **Courbe ROC**
  - ✓ **Precision Recall Curve**
- **Comparaison des performances des algorithmes**
- **Test de l'algorithme sélectionné sur le fichier billets\_production**
- **Prédiction sur un fichier fourni**
- **Discussion**

# **Introduction :**

- **l'Organisation nationale de lutte contre le faux-monnayage (ONCFM) a pour objectif de mettre en place des méthodes d'identification des contrefaçons des billets en euros .**
- **Elle souhaite mettre en place une modélisation qui serait capable d'identifier automatiquement les vrais des faux billets et ce à partir simplement de certaines dimensions du billet et des éléments qui le composent.**



# Objectif :

- Il faudrait construire un algorithme qui, à partir des caractéristiques géométriques d'un billet, serait capable de définir si ce dernier est un vrai ou un faux billet.



# Analyse des données

- Nous disposons actuellement de six informations géométriques sur un billet :
  - length : la longueur du billet (en mm) ;
  - height\_left : la hauteur du billet (mesurée sur le côté gauche, en mm) ;
  - height\_right : la hauteur du billet (mesurée sur le côté droit, en mm) ;
  - margin\_up : la marge entre le bord supérieur du billet et l'image de celui-ci (en mm) ;
  - margin\_low : la marge entre le bord inférieur du billet et l'image de celui-ci (en mm) ;
  - diagonal : la diagonale du billet (en mm).

# Analyse des données

```
[5]: 1 billets.head()
```

[5]:

	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

```
[6]: 1 billets.shape
```

[6]: (1500, 7)

# Analyse des données

```
1 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
```

# Analyse des données

```
1 billets.describe()
```

	diagonal	height_left	height_right	margin_low	margin_up	length
count	1500.000000	1500.000000	1500.000000	1463.000000	1500.000000	1500.000000
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

Il compte des billets true et false

# Analyse des données

	diagonal	height_left	height_right	length	margin_low	margin_up
is_genuine						
<b>False</b>	171.90116	104.19034	104.14362	111.63064	5.215935	3.35016
<b>True</b>	171.98708	103.94913	103.80865	113.20243	4.116097	3.05213

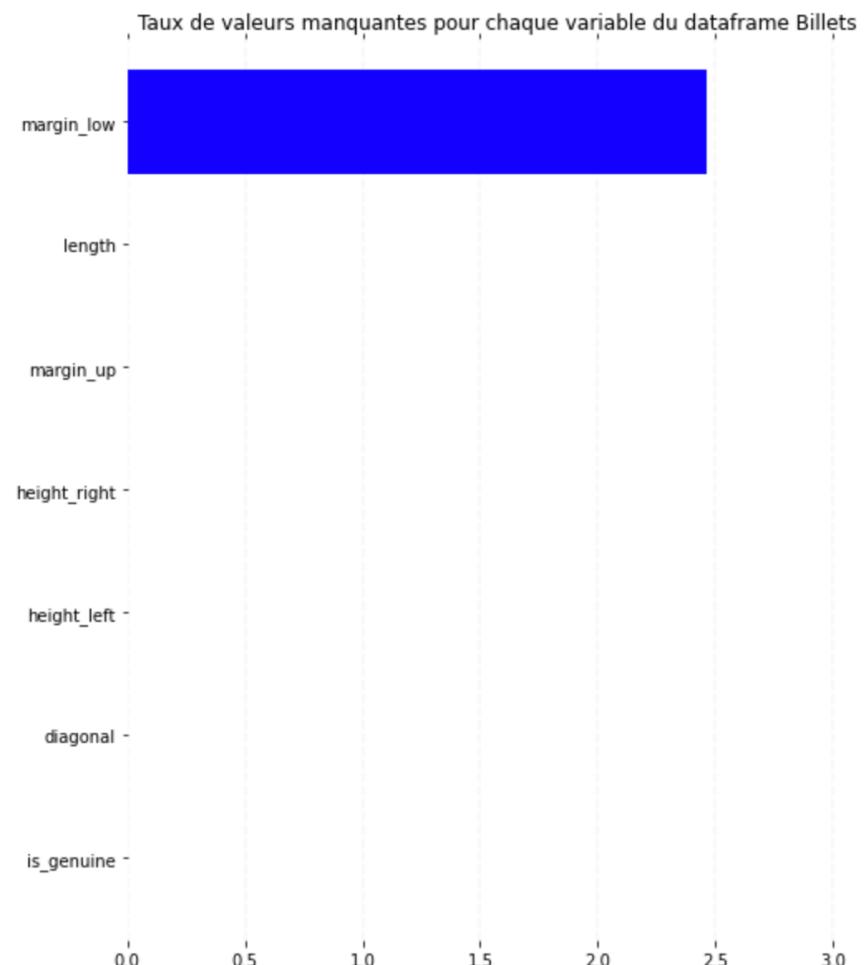
- Des moyennes assez similaires pour les deux classes, plus de différence pour les variables : length et margin\_low

# Analyse des données

```
: 1 billets.isnull().sum()
```

```
is_genuine      0  
diagonal       0  
height_left    0  
height_right   0  
margin_low     37  
margin_up      0  
length         0  
dtype: int64
```

37 valeurs nulles pour la colonne margin\_low



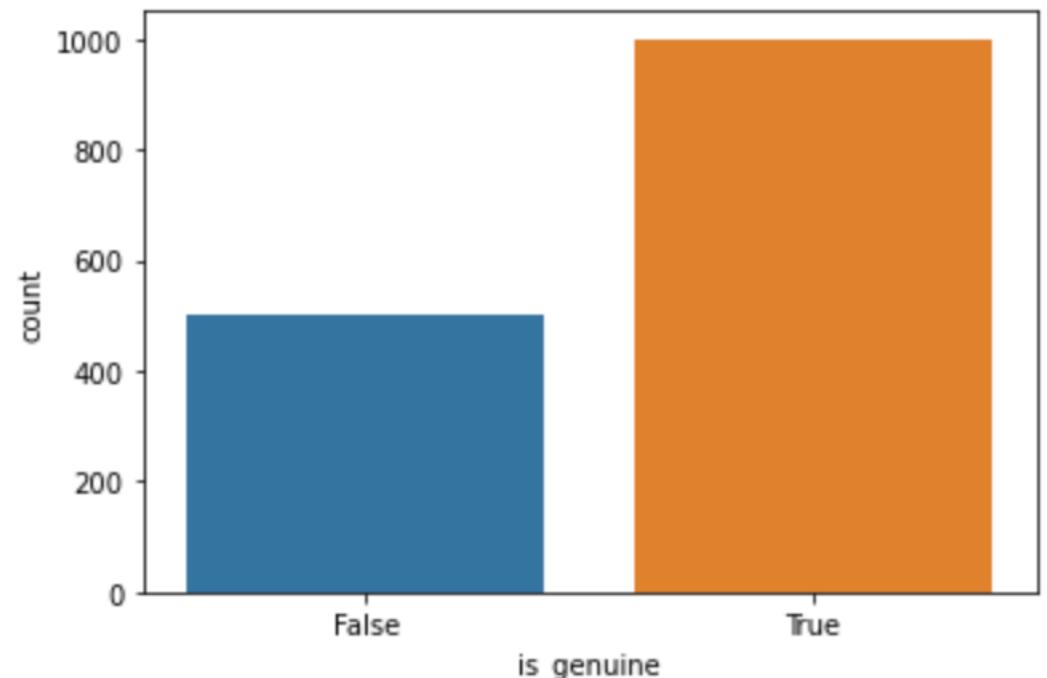
# Analyse des données

Le compte des billets true et false

```
: 1 billets['is_genuine'].value_counts()
```

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

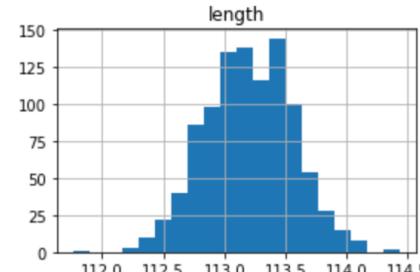
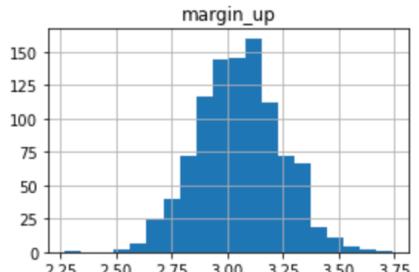
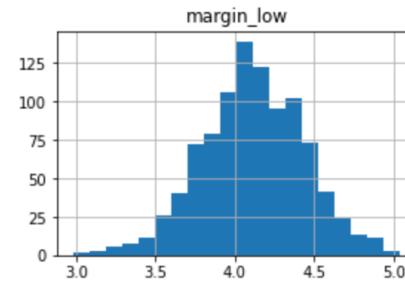
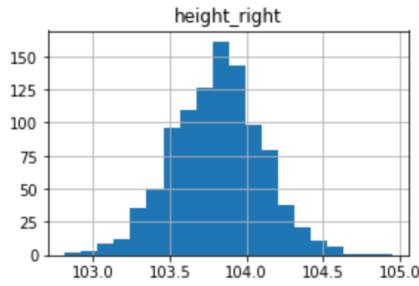
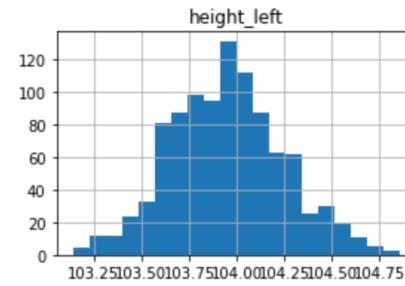
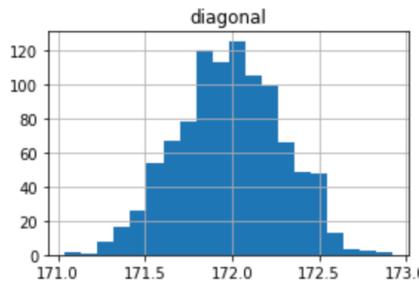
```
: 1 sns.countplot(x = 'is_genuine', data=billets);
```



# Analyse des données

Histogrammes des colonnes billets True

```
|: 1 #plot histogram
  2 billetstrue.hist(bins=20,figsize=(10,10))
  3 #plot showing
  4 plt.show()
```

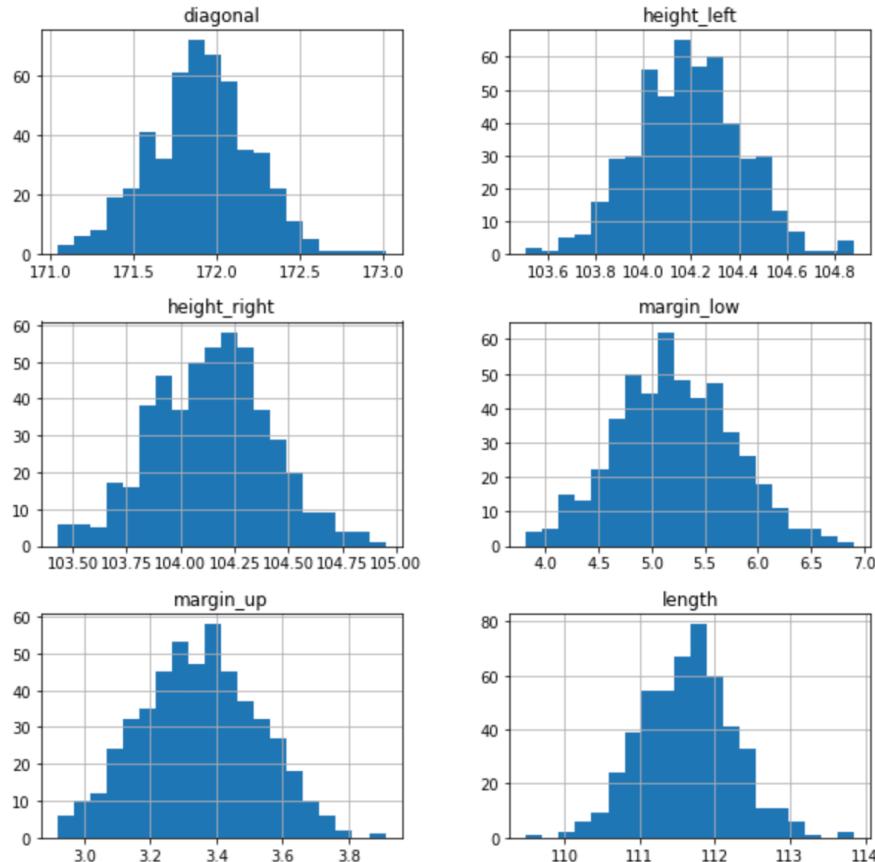


- Des distributions normaux.

# Analyse des données

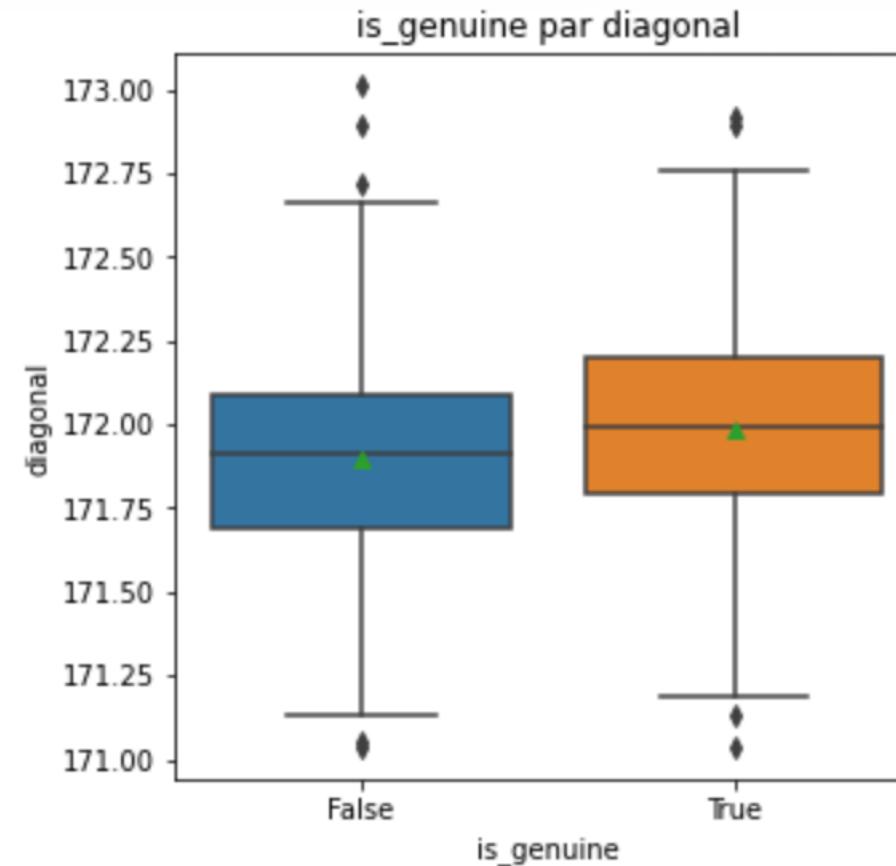
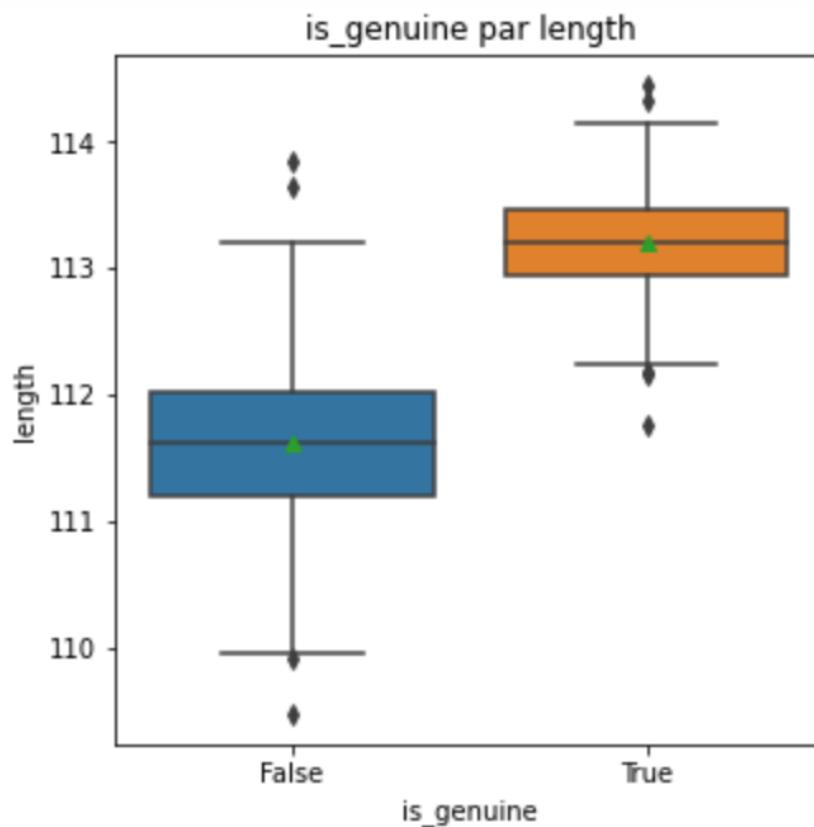
Histogrammes des colonnes billets False

```
| : 1 #plot histogram
| 2 billetsfalse.hist(bins=20,figsize=(10,10))
| 3 #plot showing
| 4 plt.show()
```



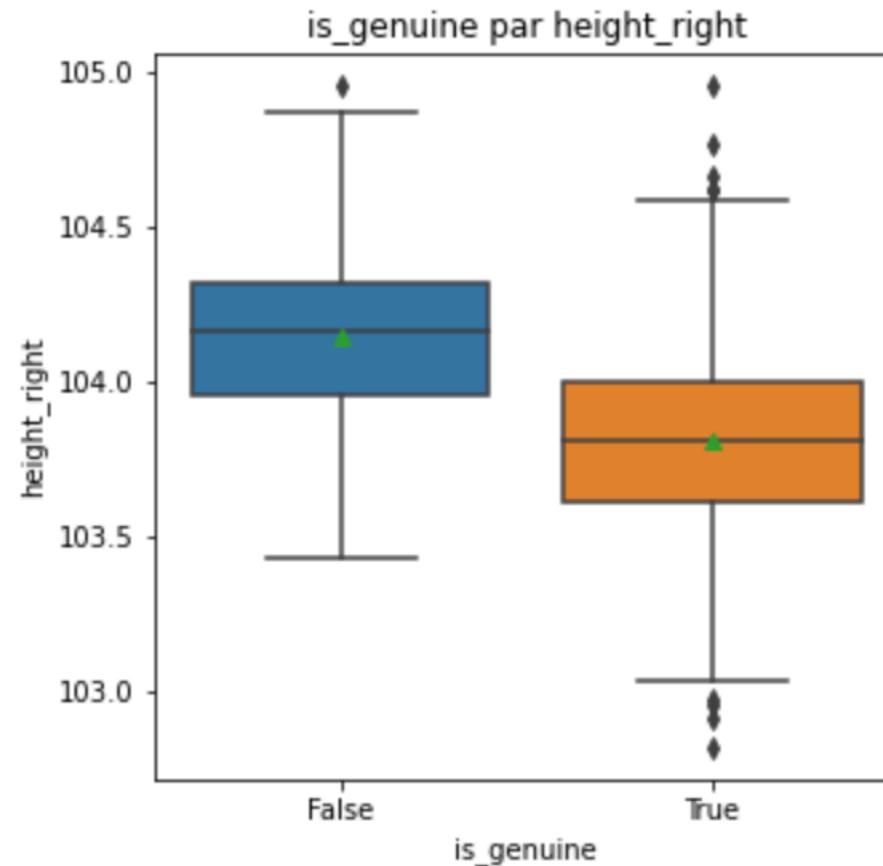
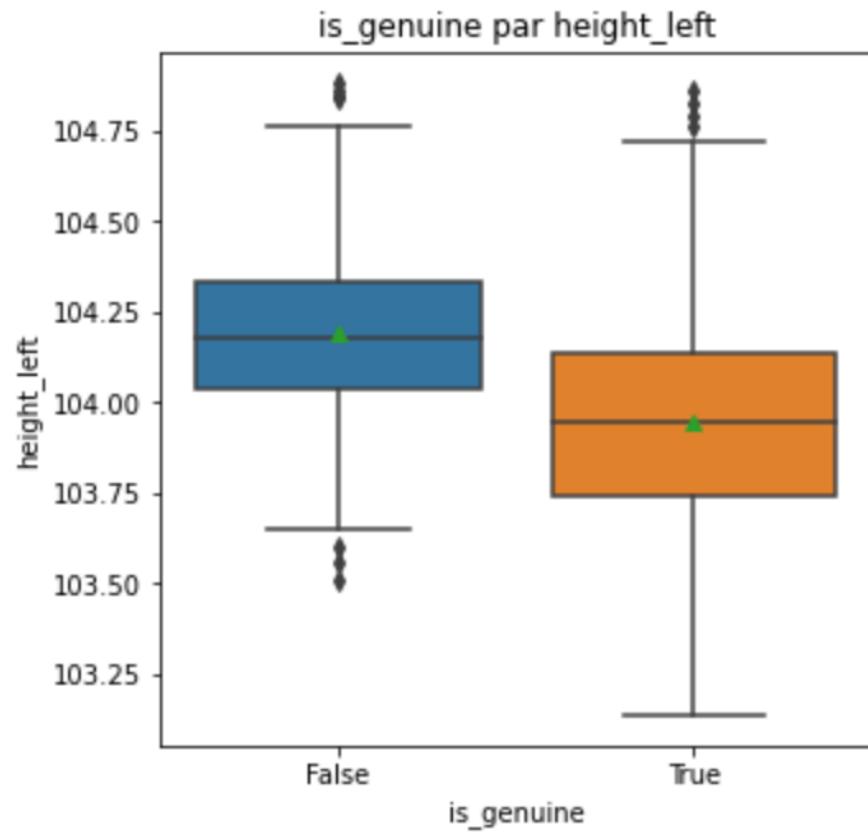
- Des distributions normaux.

# Analyse des données



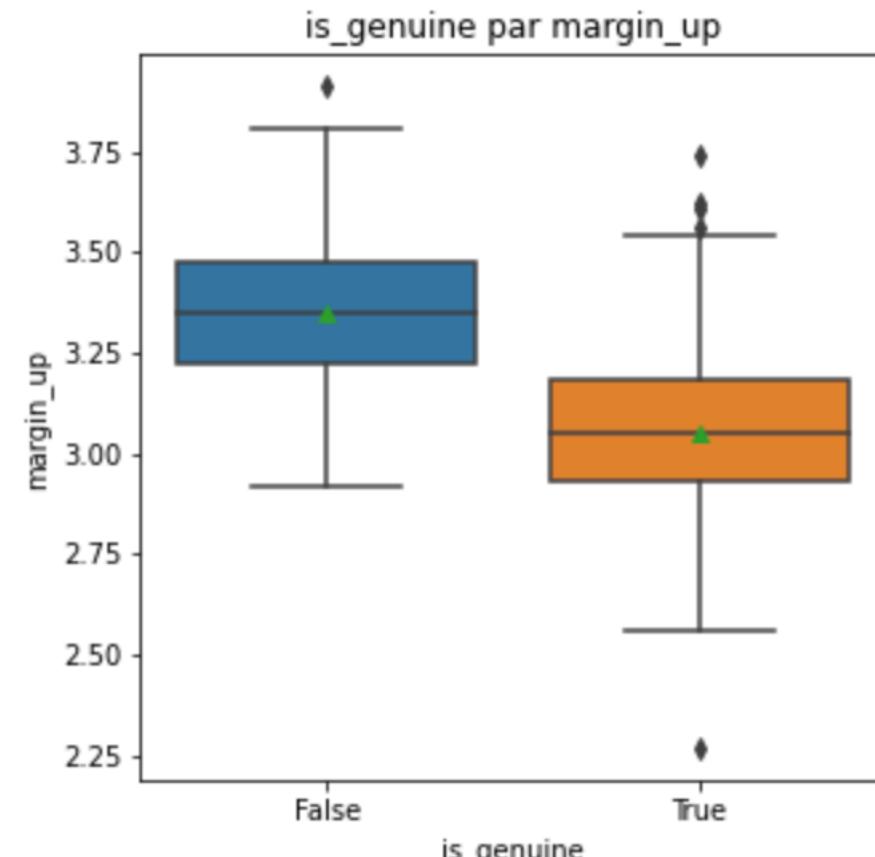
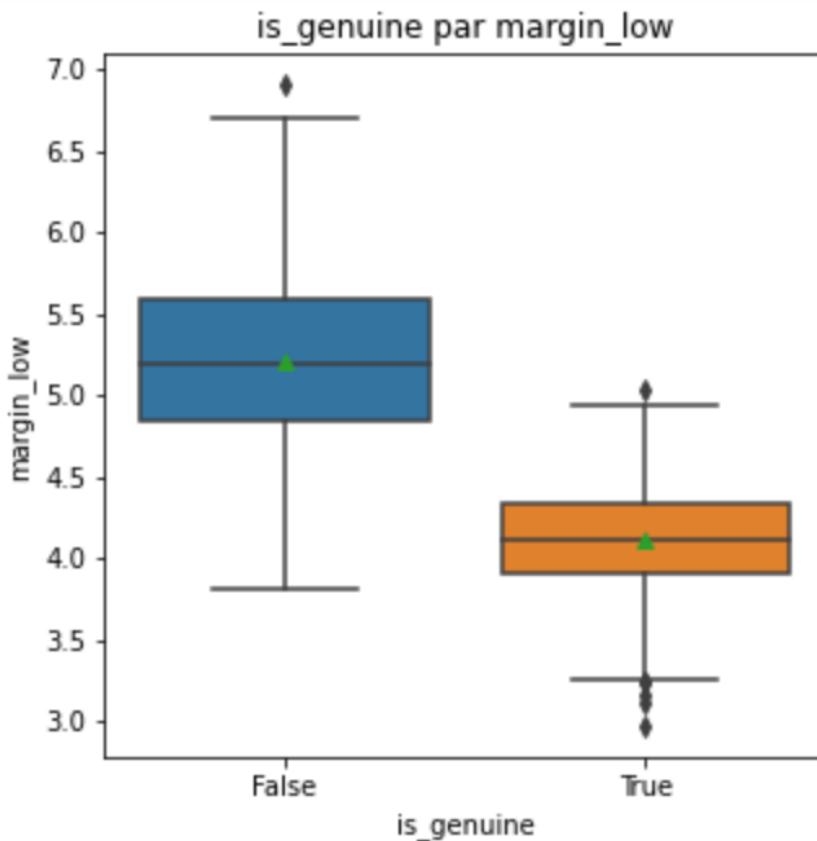
- Des boites presque similaires pour la variable diagonal pour les deux variétés.
- Les 2 variétés semblent différentes pour la variable length : la variable lenght de la catégorie False est plus dispersé que celle de la catégorie True
- L'étendue de la variable length de la catégorie False est plus importante .

# Analyse des données



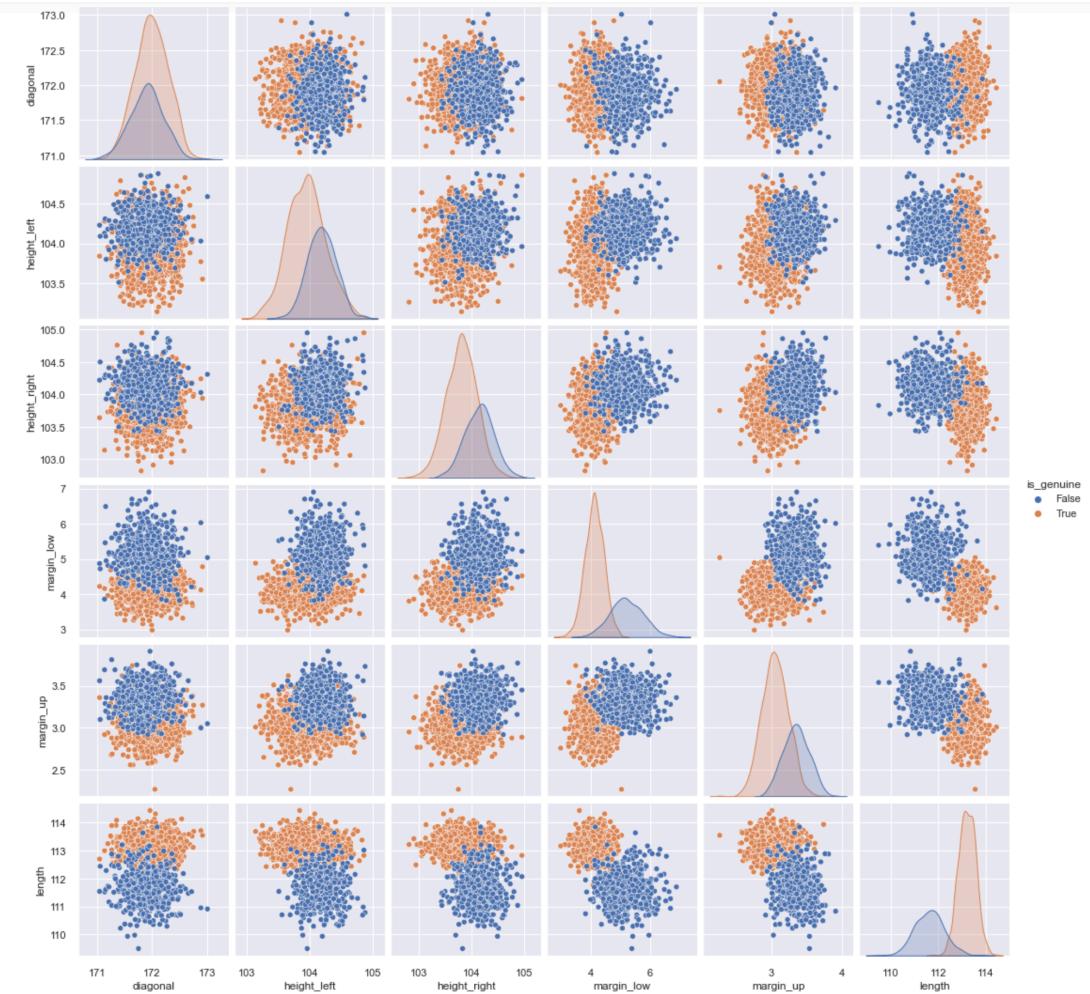
- Pour la variable height\_left : La variété True est plus dispersée et plus étendue.
- Pour la variable height\_right : Des boites assez similaires mais avec une médiane différente.

# Analyse des données



- Pour la variable margin\_low: La variété False est plus étendue et dispersée que True.
- Pour la variable margin\_up : Des boîtes assez similaires avec une différence de médiane.

# Analyse des données



- margin\_low et length sont les deux variables qui jouent un rôle dans la distinction des billets True et False . Ce sont les variables les plus déterminantes pour qu'un billet soit vrai ou faux .

# Test ANOVA sur les variables pertinentes

```
)> 1 anova_variete = smf.ols('length~is_genuine', data=billets).fit()
2 print(anova_variete.summary())
```

OLS Regression Results						
Dep. Variable:	length	R-squared:	0.721			
Model:	OLS	Adj. R-squared:	0.721			
Method:	Least Squares	F-statistic:	3877.			
Date:	Fri, 25 Mar 2022	Prob (F-statistic):	0.00			
Time:	23:19:01	Log-Likelihood:	-965.54			
No. Observations:	1500	AIC:	1935.			
Df Residuals:	1498	BIC:	1946.			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
Intercept	111.6306	0.021	5415.805	0.000	111.590	111.671
is_genuine[T.True]	1.5718	0.025	62.263	0.000	1.522	1.621
Omnibus:	40.929	Durbin-Watson:	2.022			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	92.631			
Skew:	0.074	Prob(JB):	7.68e-21			
Kurtosis:	4.208	Cond. No.	3.23			

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

- Le R au carré du test anova est de 0.72

# Test ANOVA sur les variables pertinentes

```
1 sm.stats.anova_lm(anova_variete, typ=2)
```

	sum_sq	df	F	PR(>F)
is_genuine	823.507935	1.0	3876.654138	0.0
Residual	318.216390	1498.0	Nan	Nan

- La p-valeur de ce test est à 0, est très petite et largement inférieure à 5 %. On rejette donc l'hypothèse H0 donc l'indépendance entre ces deux variables.

```
|: 1 # Test de variance  
2 billets.groupby("is_genuine")['length'].agg('var')
```

```
is_genuine  
False    0.378894  
True     0.129278  
Name: length, dtype: float64
```

- Au regard des résultats Les moyennes de chaque groupe ne sont pas toutes égales

```
020]: 1 from scipy.stats import bartlett  
2 bartlett(billets.length[billets.is_genuine == 0],  
3           billets.length[billets.is_genuine == 1])  
  
: BartlettResult(statistic=207.23111965352825, pvalue=5.520465521419425e-47)
```

- Les variances de chaque groupe ne sont pas toutes égales car p value < 5%

```
21]: 1 from scipy.stats import shapiro  
2 model = ols('length ~ is_genuine', data=billets).fit()  
3 shapiro(model.resid)
```

- Les résidus ne suivent pas une loi normale car p-value < 5%

```
ShapiroResult(statistic=0.99116450548172, pvalue=7.486640640763653e-08)
```

# Test ANOVA sur les variables pertinentes

```
e [23]: 1 anova_variete2 = smf.ols('margin_low-is_genuine', data=billets).fit()
2 print(anova_variete2.summary())
```

```
OLS Regression Results
=====
Dep. Variable: margin_low    R-squared:      0.613
Model:          OLS             Adj. R-squared:  0.613
Method:         Least Squares F-statistic:   2316.
Date:           Fri, 25 Mar 2022 Prob (F-statistic): 1.36e-303
Time:           23:19:08        Log-Likelihood: -781.24
No. Observations: 1463        AIC:            1566.
Df Residuals:   1461        BIC:            1577.
Df Model:       1
Covariance Type: nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	5.2159	0.019	280.120	0.000	5.179	5.252
is_genuine[T.True]	-1.0998	0.023	-48.120	0.000	-1.145	-1.055

```
=====
Omnibus:            21.882    Durbin-Watson:     2.031
Prob(Omnibus):      0.000    Jarque-Bera (JB): 37.654
Skew:               0.063    Prob(JB):        6.66e-09
Kurtosis:            3.776    Cond. No.:       3.21
=====
```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

- Le R au carré du test anova est de 0.61

# Test ANOVA sur les variables pertinentes

```
[24]: 1 # Test de Fisher  
2 sm.stats.anova_lm(anova_variete2, typ=2)  
  
]:  


|            | sum_sq     | df     | F           | PR(>F)        |
|------------|------------|--------|-------------|---------------|
| is_genuine | 395.000430 | 1.0    | 2315.553532 | 1.362034e-303 |
| Residual   | 249.225777 | 1461.0 | Nan         | Nan           |


```

- La p-valeur de ce test est à 0, est très petite et largement inférieure à 5 %. On rejette donc l'hypothèse H0 donc l'indépendance entre ces deux variables.

```
[]: 1 # Test de variance  
2 billets.groupby("is_genuine")['margin_low'].agg('var')  
  
is_genuine  
False    0.306397  
True     0.101840  
Name: margin_low, dtype: float64
```

- Au regard des résultats Les moyennes de chaque groupe ne sont pas toutes égales

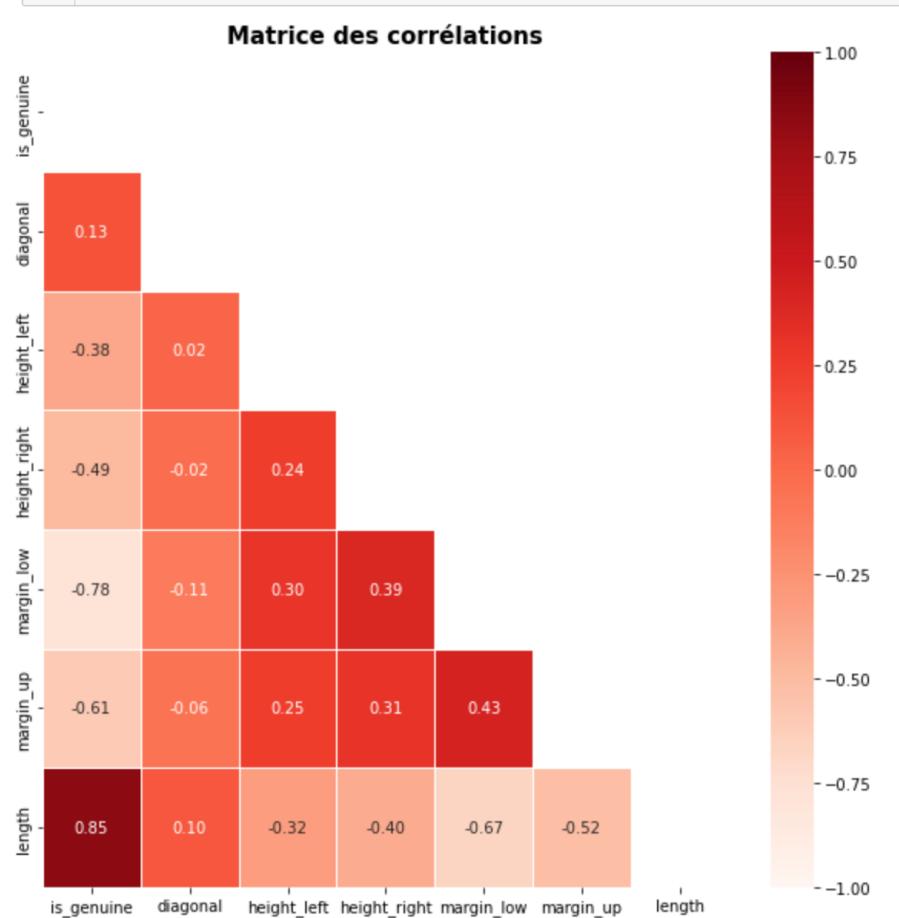
```
[22]: 1 from scipy.stats import bartlett  
2 bartlett(billets.margin_low[billets.is_genuine == 0],  
3           billets.margin_low[billets.is_genuine == 1])  
  
]: BartlettResult(statistic=216.88739037277128, pvalue=4.318616798051279e-49)
```

- Les variances de chaque groupe ne sont pas toutes égales car p-value < 5%

```
[23]: 1 from scipy.stats import shapiro  
2 model = ols('margin_low ~ is_genuine', data=billets).fit()  
3 shapiro(model.resid)  
  
ShapiroResult(statistic=0.993627667427063, pvalue=4.701406851381762e-06)
```

- Les résidus ne suivent pas une loi normale car p-value < 5%

# Gestion des valeurs manquantes par régression linéaire



- Les variables les plus corrélés sont: length et is\_genuine, margin\_low margin\_up

# Gestion des valeurs manquantes par régression linéaire

## Régression linéaire simple

```
[1]: 1 reg_simp = smf.ols('margin_up ~ margin_low', data=df_full).fit()
2 print(reg_simp.summary())
```

```
OLS Regression Results
=====
Dep. Variable: margin_up R-squared: 0.186
Model: OLS Adj. R-squared: 0.186
Method: Least Squares F-statistic: 334.5
Date: Fri, 25 Mar 2022 Prob (F-statistic): 1.92e-67
Time: 23:20:05 Log-Likelihood: 216.23
No. Observations: 1463 AIC: -428.5
Df Residuals: 1461 BIC: -417.9
Df Model: 1
Covariance Type: nonrobust
=====

      coef  std err      t      P>|t|      [0.025    0.975]
-----
Intercept  2.4780   0.037  66.402  0.000     2.405    2.551
margin_low 0.1505   0.008  18.288  0.000     0.134    0.167
=====
Omnibus: 1.749 Durbin-Watson: 1.771
Prob(Omnibus): 0.417 Jarque-Bera (JB): 1.665
Skew: 0.079 Prob(JB): 0.435
Kurtosis: 3.047 Cond. No. 32.5
=====
```

- R-squared: 0.18 , très faible par conséquent la qualité prédictive de ce modèle est faible

# Gestion des valeurs manquantes par régression linéaire

## Régression linéaire multiple

```
.]: 1 #On régresse margin_low en fonction des autres variables de l'échantillon
 2 reg_multi = smf.ols('margin_low~diagonal+height_left+height_right+margin_up+length'
 3 print(reg_multi.summary())
```

```
OLS Regression Results
=====
Dep. Variable: margin_low R-squared: 0.477
Model: OLS Adj. R-squared: 0.476
Method: Least Squares F-statistic: 266.1
Date: Fri, 25 Mar 2022 Prob (F-statistic): 2.60e-202
Time: 23:20:10 Log-Likelihood: -1001.3
No. Observations: 1463 AIC: 2015.
Df Residuals: 1457 BIC: 2046.
Df Model: 5
Covariance Type: nonrobust
=====

      coef  std err      t  P>|t|  [0.025  0.975]
-----
Intercept    22.9948   9.656   2.382  0.017   4.055  41.935
diagonal    -0.1111   0.041  -2.680  0.007  -0.192  -0.030
height_left   0.1841   0.045   4.113  0.000   0.096  0.272
height_right   0.2571   0.043   5.978  0.000   0.173  0.342
margin_up     0.2562   0.064   3.980  0.000   0.130  0.382
length       -0.4091   0.018 -22.627  0.000  -0.445  -0.374
=====
Omnibus: 73.627 Durbin-Watson: 1.893
Prob(Omnibus): 0.000 Jarque-Bera (JB): 95.862
Skew: 0.482 Prob(JB): 1.53e-21
Kurtosis: 3.801 Cond. No. 1.94e+05
=====
```

- Le R-squared est de l'ordre de 0.47 Ce n'est pas très élevé, mais ceci est logique au vu de la dispersion du nuage de points originel.

# Gestion des valeurs manquantes par régression linéaire

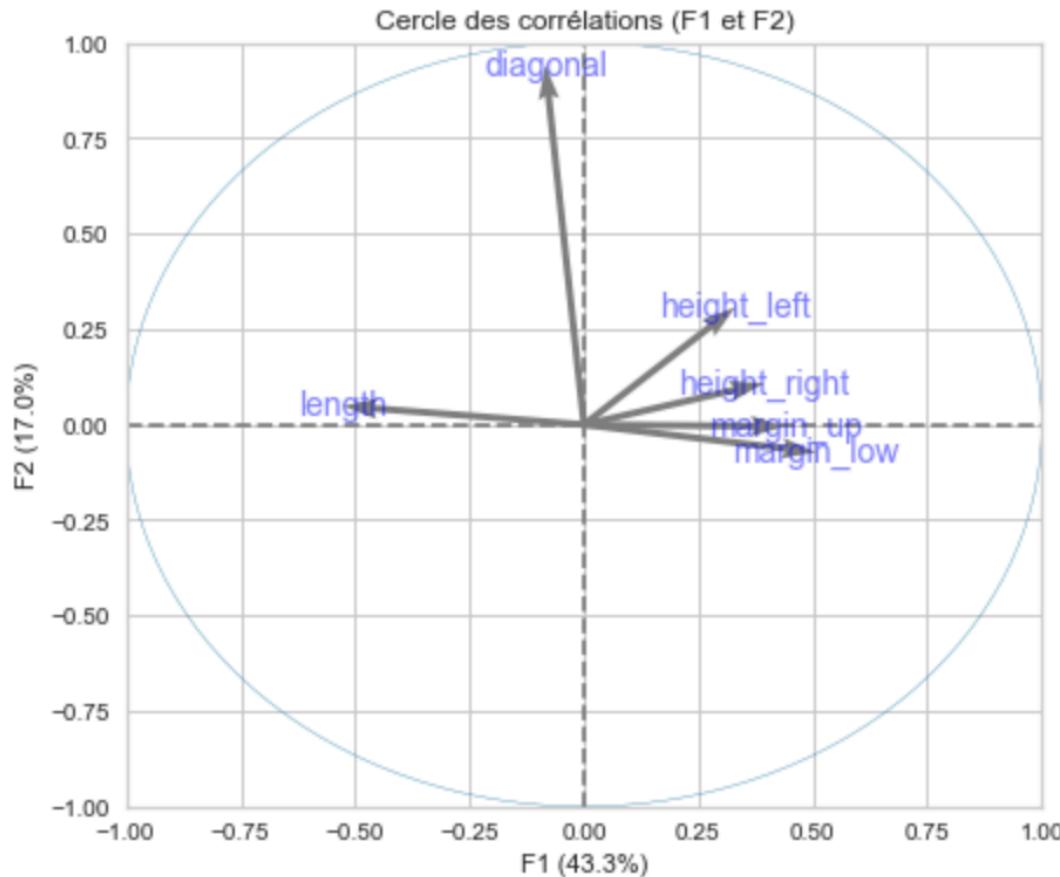
```
billets.loc[billets.margin_low.isna(), 'margin_low'] = reg_multi.predict(billets)
```

```
] : 1 billets.describe()
```

	diagonal	height_left	height_right	margin_low	margin_up	length
count	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000	1500.000000
mean	171.958440	104.029533	103.920307	4.483475	3.151473	112.67850
std	0.305195	0.299462	0.325627	0.659632	0.231813	0.87273
min	171.040000	103.140000	102.820000	2.980000	2.270000	109.49000
25%	171.750000	103.820000	103.710000	4.020000	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

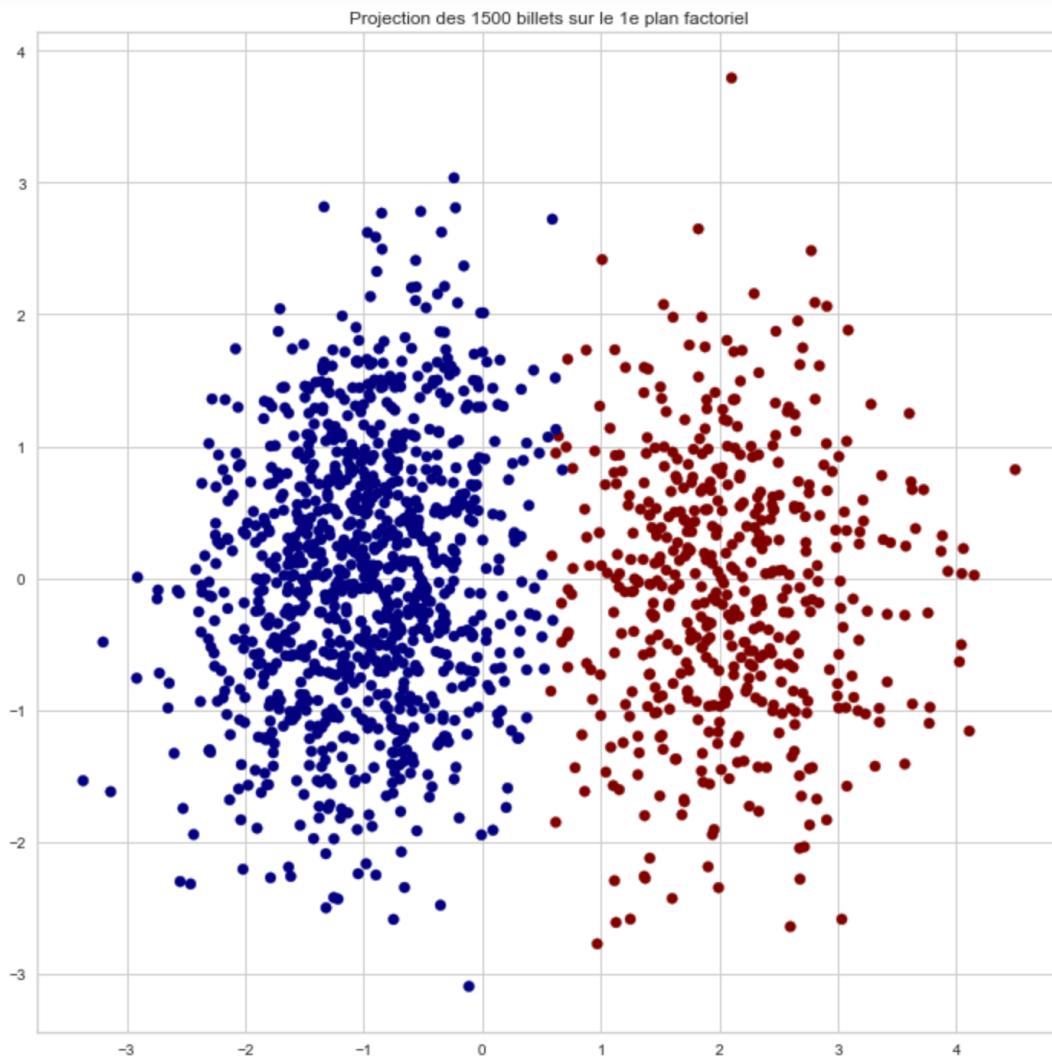
- Au vu des résultats comparé à la description avec NAN, des résultats similaires en terme de moyenne et de l'écart type(std).

# ACP



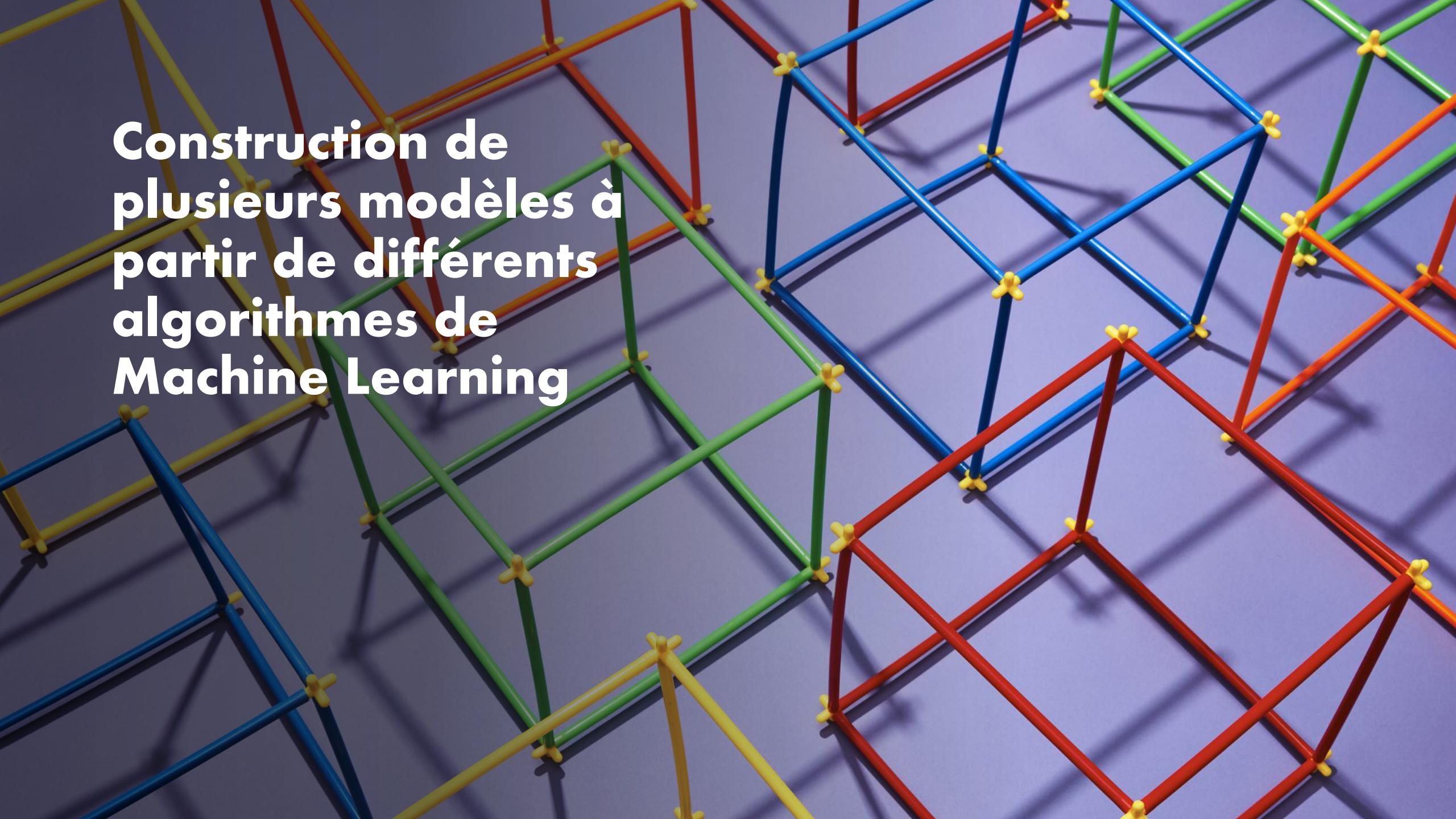
- Les variables height\_left, height\_right, margin\_low , margin\_up sont corrélés positivement aux variables synthétiques F1 (coeff corrélation entre 0,25 et 0,3)
- La variable length est corrélée négativement à F1 c'est-à-dire que, quand length croît, alors F1 décroît. Cela se traduit par un coefficient de corrélation proche de -1 (ici -0.50).
- Les variables les plus corrélées à F2 sont : diagonal

# Projection des données selon les clusters



- Des clusters bien distincts.

**Construction de  
plusieurs modèles à  
partir de différents  
algorithmes de  
Machine Learning**



# Division de l'ensemble des données : données d'entraînement et de test

```
1 # La variable prédictive : Target
2 y=billets.is_genuine.values
3 # Les features
4 x=billets.iloc[:,1:7].values

]: 1 from sklearn.model_selection import train_test_split
2
3 xtrain, xtest, ytrain, ytest = train_test_split(x, y, train_size=0.8,stratify=y, random_state=42)

]: 1 print("shape of original dataset :", billets.shape)
2 print("shape of input - training set", xtrain.shape)
3 print("shape of output - training set", ytrain.shape)
4 print("shape of input - testing set", xtest.shape)
5 print("shape of output - testing set", ytest.shape)

shape of original dataset : (1500, 7)
shape of input - training set (1200, 6)
shape of output - training set (1200,)
shape of input - testing set (300, 6)
shape of output - testing set (300,)
```

# KNN

**L'algorithme des K plus proches voisins est un algorithme de Machine Learning qui appartient à la classe des algorithmes d'apprentissage supervisé . Le principe de ce modèle consiste à choisir les k données les plus proches du point étudié afin d'en prédire sa valeur**

```
1 # Fonction qui va entraîner le modèle, prédire et évaluer l'ensemble des modèles
2 def evaluation(model):
3     model.fit(xtrain, ytrain)
4     y_pred = model.predict(xtest)
5     y_true = ytest
6     print(classification_report(y_true, y_pred))
7     print(confusion_matrix(y_true, y_pred))
```

```
1 # Entrainement du modèle avec 3 neighbors
2 model = neighbors.KNeighborsClassifier(n_neighbors=3)
```

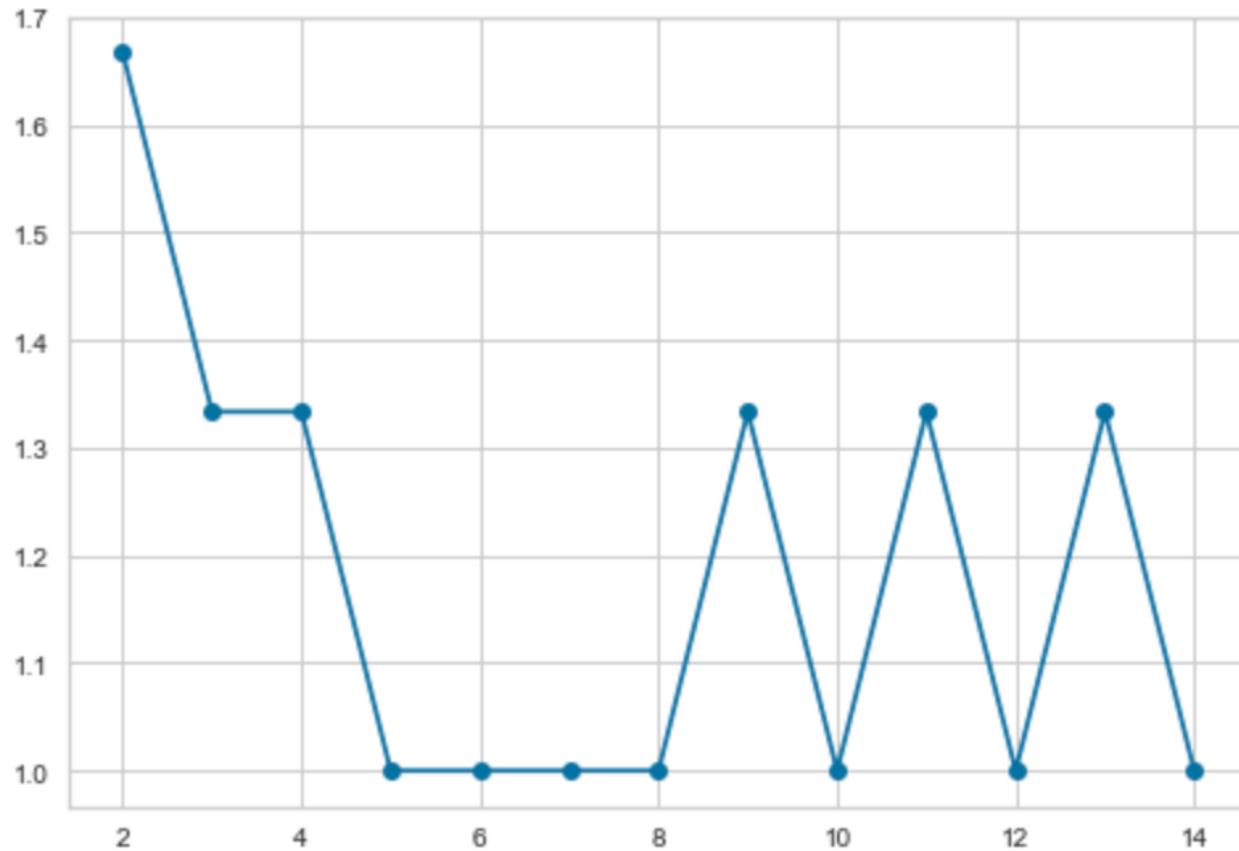
```
6]: 1 evaluation(model)
```

	precision	recall	f1-score	support
False	0.98	0.98	0.98	100
True	0.99	0.99	0.99	200
accuracy			0.99	300
macro avg	0.98	0.98	0.98	300
weighted avg	0.99	0.99	0.99	300

```
3 error = 1 - knn.score(xtest, ytest)
4 print('Erreur: %f' % error)
```

Erreur: 0.010000

# KNN



- Comme on peut le voir, le k-NN le plus performant est celui pour lequel  $k$  est entre 5 et 8 .
- On connaît donc notre classifieur final optimal

# KNN

```
Entrée [323]: 1 # Entrainement du modèle avec 5 neighbors
```

```
2
```

```
3 model = neighbors.KNeighborsClassifier(n_neighbors=5)
```

```
Entrée [324]: 1 evaluation(model)
```

	precision	recall	f1-score	support
False	0.99	0.98	0.98	100
True	0.99	0.99	0.99	200
accuracy			0.99	300
macro avg	0.99	0.99	0.99	300
weighted avg	0.99	0.99	0.99	300
	[[ 98  2]			
	[ 1 199]]			

# KNN

```
Entrée [317]: 1 model = neighbors.KNeighborsClassifier(n_neighbors=6)
```

```
Entrée [318]: 1 evaluation(model)
```

	precision	recall	f1-score	support
False	0.99	0.98	0.98	100
True	0.99	0.99	0.99	200
accuracy			0.99	300
macro avg	0.99	0.99	0.99	300
weighted avg	0.99	0.99	0.99	300
	[[ 98 2]			
	[ 1 199]]			

# KNN

```
[1]: model = neighbors.KNeighborsClassifier(n_neighbors=8)
```

```
[1]: evaluation(model)
```

	precision	recall	f1-score	support
False	0.99	0.98	0.98	100
True	0.99	0.99	0.99	200
accuracy			0.99	300
macro avg	0.99	0.99	0.99	300
weighted avg	0.99	0.99	0.99	300

```
[[ 98  2]
 [ 1 199]]
```

# Random Forest

- Le random forest ou forêt aléatoire est un algorithme supervisé, dans le domaine du machine learning. Il permet d'obtenir une prédiction, grâce à son système de forêt d'arbres décisionnels

```
1 #random forest
2 from sklearn.ensemble import RandomForestClassifier
3 model= RandomForestClassifier(n_estimators=100,random_state=1)
```

```
:8]: 1 evaluation(model)
```

	precision	recall	f1-score	support
False	0.99	0.98	0.98	100
True	0.99	0.99	0.99	200
accuracy			0.99	300
macro avg	0.99	0.99	0.99	300
weighted avg	0.99	0.99	0.99	300

```
[[ 98  2]
 [ 1 199]]
```

# Régression logistique

```
: 1 reg_log2 = smf.glm('is_genuine~diagonal+height_left+height_right+margin_low+margin_up+length',
2                      data=billets, family=sm.families.Binomial()).fit()
3 print(reg_log2.summary())
```

```
Generalized Linear Model Regression Results
=====
Dep. Variable: ['is_genuine[False]', 'is_genuine[True]'] No. Observations: 1500
Model: GLM Df Residuals: 1493
Model Family: Binomial Df Model: 6
Link Function: logit Scale: 1.0000
Method: IRLS Log-Likelihood: -42.342
Date: Fri, 25 Mar 2022 Deviance: 84.685
Time: 23:21:56 Pearson chi2: 2.65e+03
No. Iterations: 10
Covariance Type: nonrobust
=====
      coef    std err      z   P>|z|    [0.025    0.975]
Intercept  204.5582  241.768     0.846    0.398   -269.299   678.415
diagonal   -0.0680    1.091    -0.062    0.950    -2.207    2.071
height_left  1.7162    1.104     1.555    0.120    -0.447    3.880
height_right 2.2584    1.072     2.107    0.035    0.157    4.359
margin_low   5.7756    0.937     6.164    0.000    3.939    7.612
margin_up    10.1531   2.108     4.817    0.000    6.022   14.284
length     -5.9129    0.846    -6.991    0.000    -7.571   -4.255
=====
```

- Au vu des résultats je retire les variables moins significatifs avec la p-value supérieur à 0.05 à savoir : diagonal et height\_left

# Régression logistique

```
6]: 1 reg_log2 = smf.glm('is_genuine~height_right+margin_low+margin_up+length',data=billets, family=sm.families.Binomial)
2 print(reg_log2.summary())
```

## Generalized Linear Model Regression Results

```
=====
Dep. Variable: ['is_genuine[False]', 'is_genuine[True]'] No. Observations: 1500
Model: GLM Df Residuals: 1495
Model Family: Binomial Df Model: 4
Link Function: logit Scale: 1.0000
Method: IRLS Log-Likelihood: -43.586
Date: Fri, 25 Mar 2022 Deviance: 87.173
Time: 23:21:58 Pearson chi2: 3.24e+03
No. Iterations: 10
Covariance Type: nonrobust
=====
```

	coef	std err	z	P> z	[ 0.025	0.975]
Intercept	323.4452	139.532	2.318	0.020	49.968	596.923
height_right	2.7839	1.078	2.583	0.010	0.671	4.897
margin_low	6.0195	0.897	6.712	0.000	4.262	7.777
margin_up	10.2075	2.079	4.910	0.000	6.133	14.282
length	-5.9831	0.827	-7.237	0.000	-7.603	-4.363

# Régression logistique

```
1 from sklearn.linear_model import LogisticRegression  
2  
3 model=LogisticRegression()
```

```
1 evaluation(model)
```

	precision	recall	f1-score	support
False	0.99	0.98	0.98	100
True	0.99	0.99	0.99	200
accuracy			0.99	300
macro avg	0.99	0.99	0.99	300
weighted avg	0.99	0.99	0.99	300

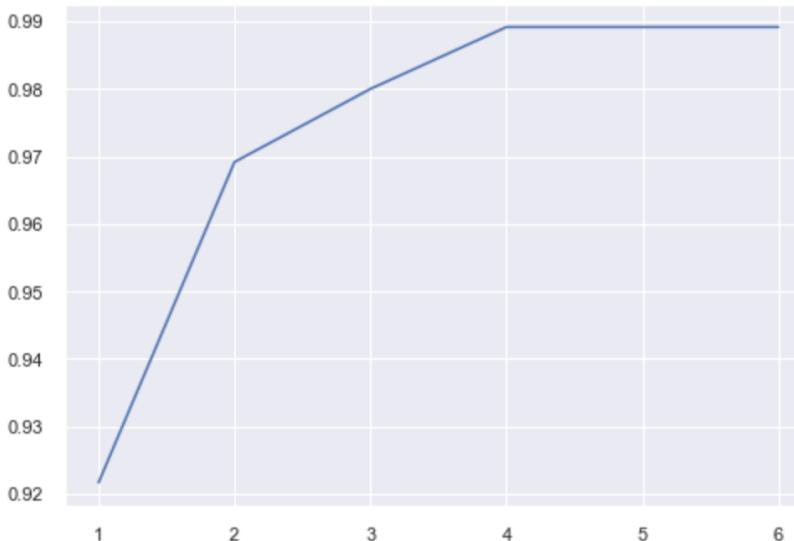
```
[ 98  2]  
[ 1 199]]
```

# Régression logistique par sélection de variables et validation croisée

```
9 #instanciation t calculs  
10 sel=RFECV(estimator=lr,cv=10,scoring='accuracy')  
11 sel.fit(xtrain,ytrain)
```

```
: RFECV(cv=10, estimator=LogisticRegression(), scoring='accuracy')
```

```
t[115]: [<matplotlib.lines.Line2D at 0x7fd369e27b50>]
```



```
: 1 evaluation(model)
```

	precision	recall	f1-score	support
False	0.99	0.98	0.98	100
True	0.99	0.99	0.99	200
accuracy			0.99	300
macro avg	0.99	0.99	0.99	300
weighted avg	0.99	0.99	0.99	300
[ 98 2 ]				
[ 1 199 ]				

```
rée [116]: 1 # D après l'outil 4 variables  
2 sel.n_features_
```

```
t[116]: 4
```

	predite	cluster
0	True	1
1	False	0
2	False	0
3	True	1
4	True	1
5	True	1
6	True	1
7	True	1
8	True	1
9	True	1
10	False	0
11	True	1
12	False	0
13	False	0
14	True	1
15	False	0
16	True	1
17	True	1
18	True	1
19	True	1
20	False	0
21	False	0
22	False	0
23	True	1
24	False	0
25	True	1
26	True	1
27	False	0

# Kmeans

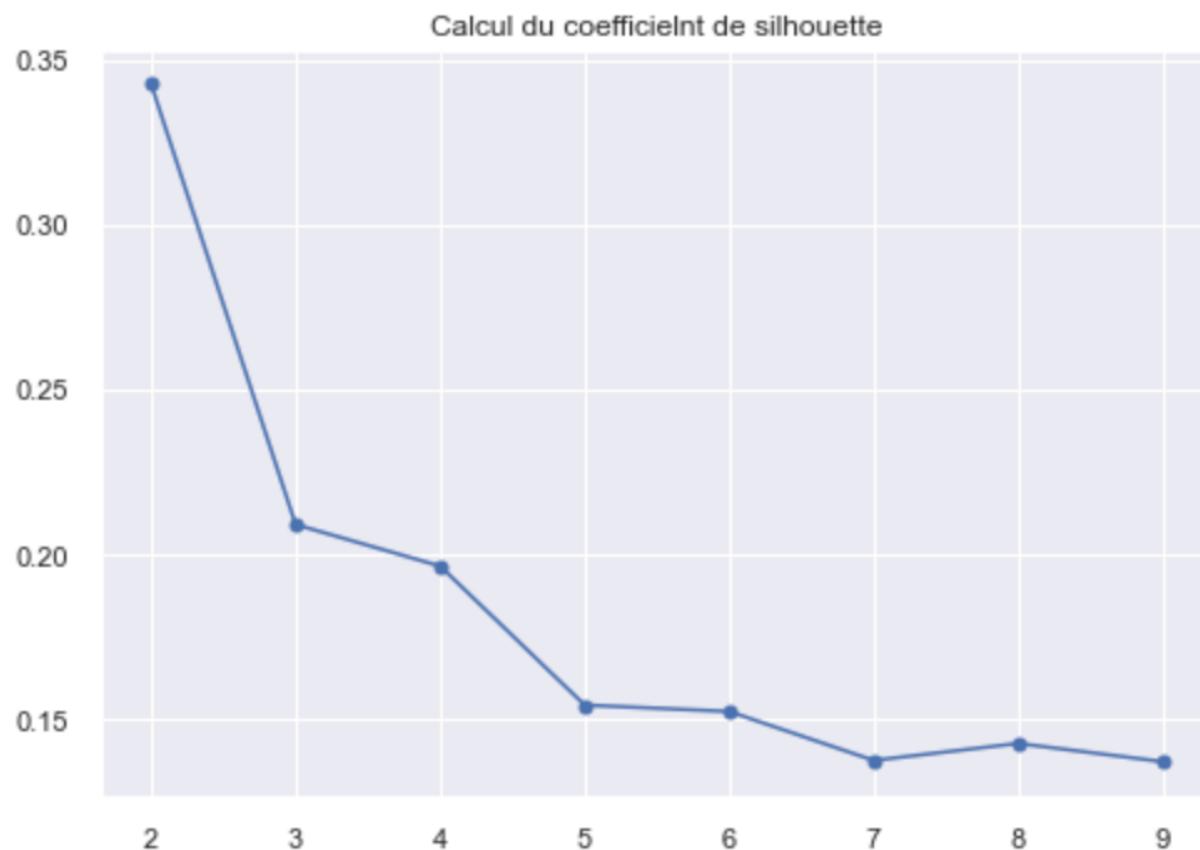
```
: 1 import sklearn.cluster as cluster
2 model = cluster.KMeans(n_clusters=2)
```

```
] : 1 evaluation(model)
```

	precision	recall	f1-score	support
False	0.99	0.95	0.97	100
True	0.98	0.99	0.99	200
accuracy			0.98	300
macro avg	0.98	0.97	0.98	300
weighted avg	0.98	0.98	0.98	300

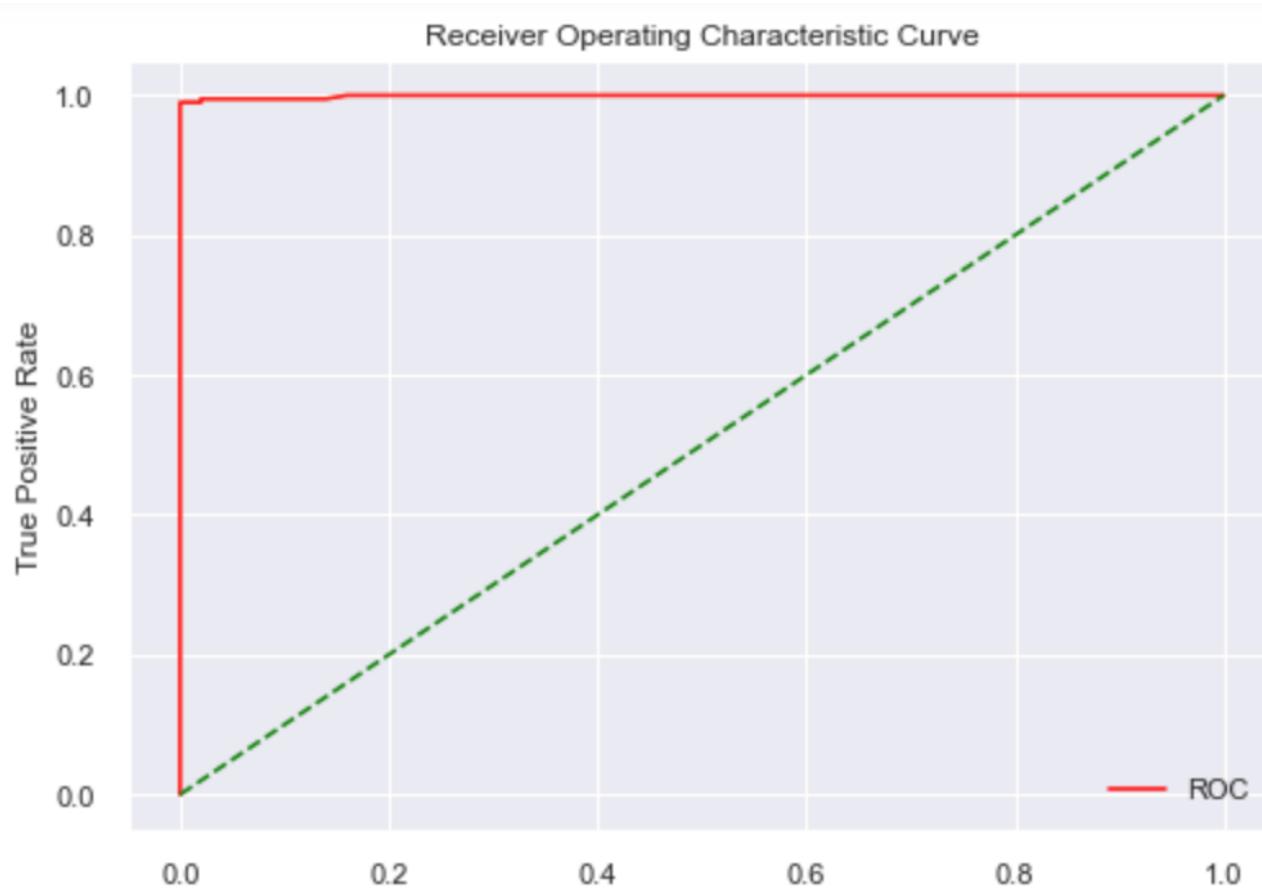
```
[ 95  5]
[  1 199]]
```

# Kmeans



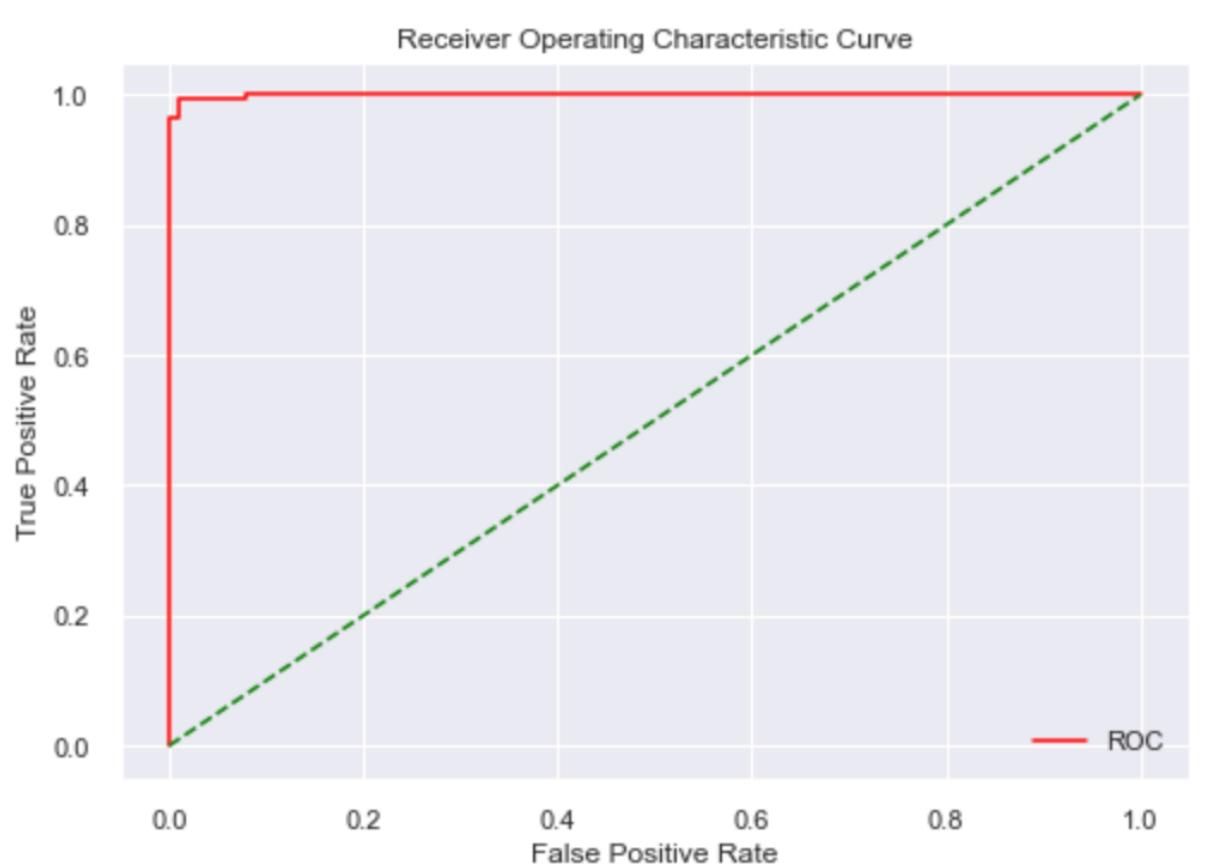
# Courbe ROC

Cette courbe est essentiellement une représentation graphique des performances de tout modèle de classification à tous les seuils de classification.



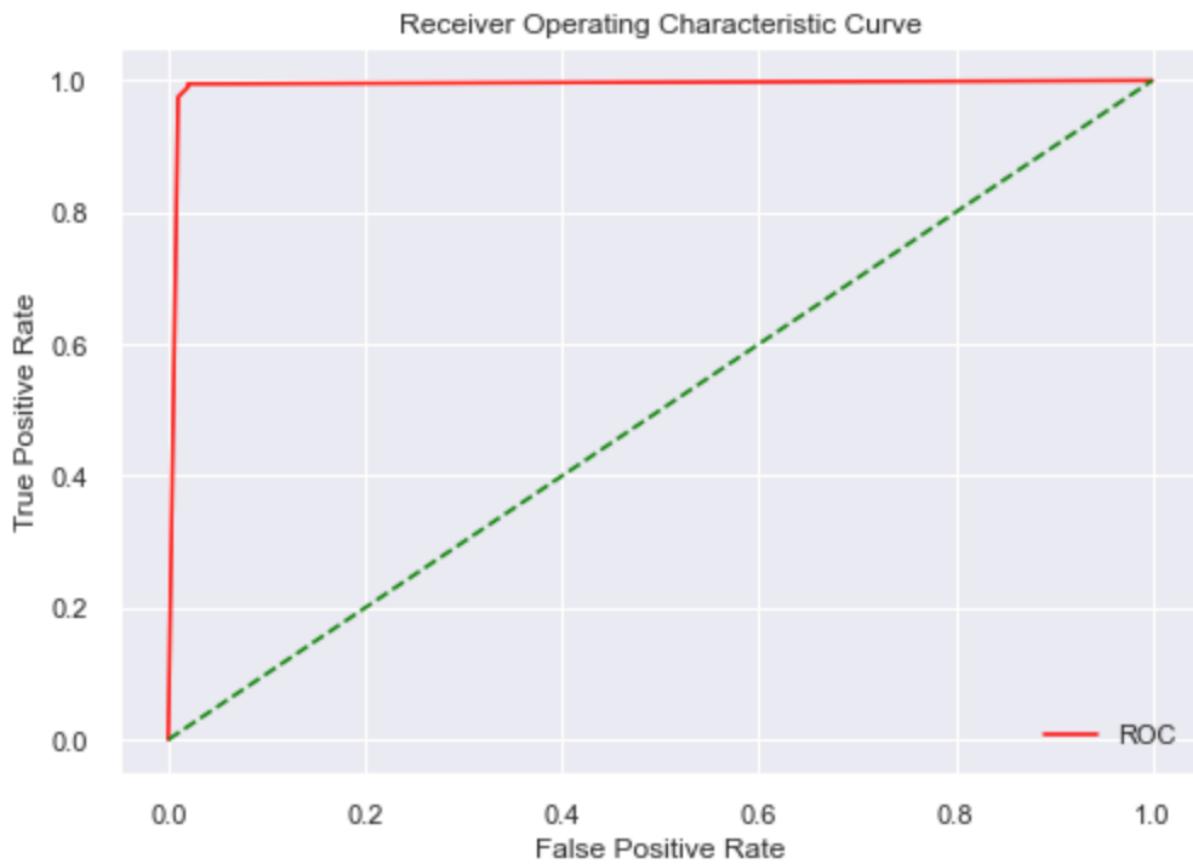
- Courbe ROC du random forest

# Courbe ROC



- Courbe ROC de la régression logistique

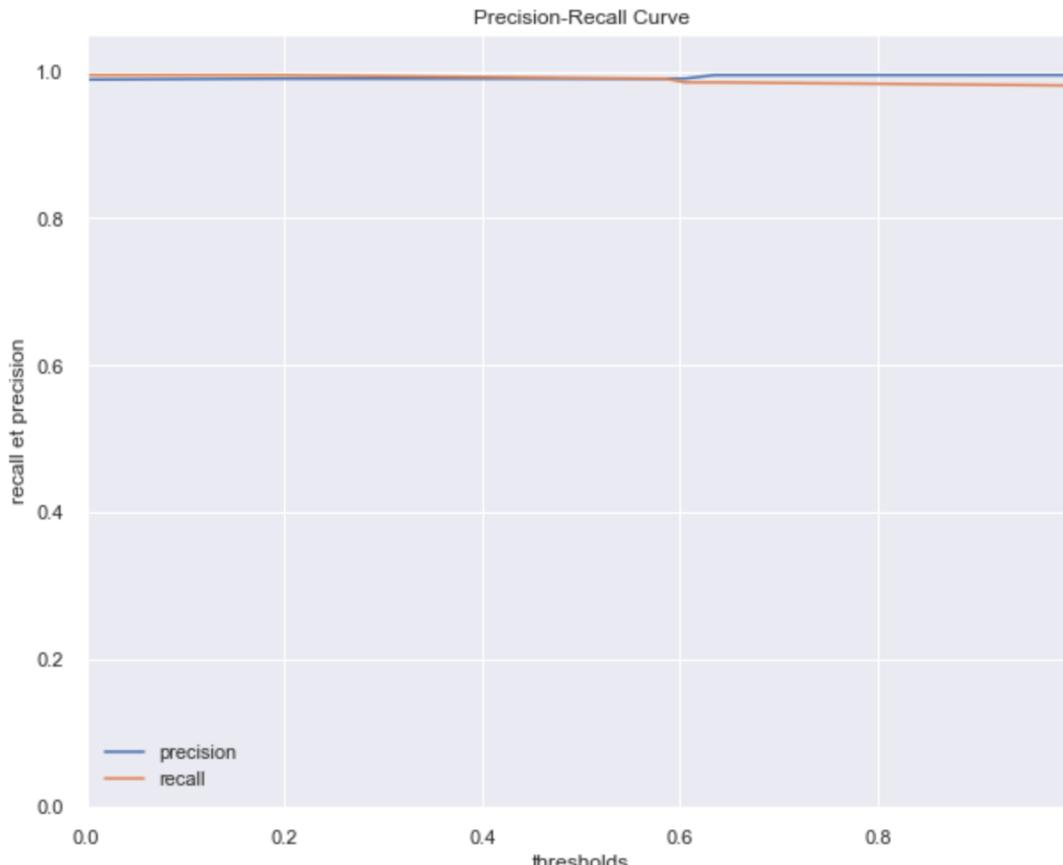
# Courbe ROC



- Courbe ROC du KNN

# Precision Recall Curve

- Cette courbe est une représentation directe de la précision et du rappel



```
1 y_pred=model_final(lr,xtest,thresholds=0.6)
9]: 1 from sklearn.metrics import f1_score
2 f1_score(ytest,y_pred)
0.9925187032418954
51]: 1 from sklearn.metrics import precision_score
2 precision_score(ytest,y_pred)
: 0.9900497512437811
Entrée [550]: 1 from sklearn.metrics import recall_score
2 recall_score(ytest,y_pred)
Out[550]: 0.995
```

# Comparaison des performances des algorithmes

## KNN

```
Entrée [317]: 1 model = neighbors.KNeighborsClassifier(n_neighbors=6)
```

```
Entrée [318]: 1 evaluation(model)
```

	precision	recall	f1-score	support
False	0.99	0.98	0.98	100
True	0.99	0.99	0.99	200
accuracy			0.99	300
macro avg	0.99	0.99	0.99	300
weighted avg	0.99	0.99	0.99	300

[[ 98 2]
[ 1 199]]

## Random Forest

```
:8]: 1 evaluation(model)
```

	precision	recall	f1-score	support
False	0.99	0.98	0.98	100
True	0.99	0.99	0.99	200
accuracy			0.99	300
macro avg	0.99	0.99	0.99	300
weighted avg	0.99	0.99	0.99	300

[[ 98 2]
[ 1 199]]

## Régression logistique

```
: 1 evaluation(model)
```

	precision	recall	f1-score	support
False	0.99	0.98	0.98	100
True	0.99	0.99	0.99	200
accuracy			0.99	300
macro avg	0.99	0.99	0.99	300
weighted avg	0.99	0.99	0.99	300

[ 98 2]
[ 1 199]]

## Kmeans

```
: ]: 1 evaluation(model)
```

	precision	recall	f1-score	support
False	0.99	0.95	0.97	100
True	0.98	0.99	0.99	200
accuracy			0.98	300
macro avg	0.98	0.97	0.98	300
weighted avg	0.98	0.98	0.98	300

[ 95 5]
[ 1 199]]

# Evaluation du modèle final

Matrice de confusion

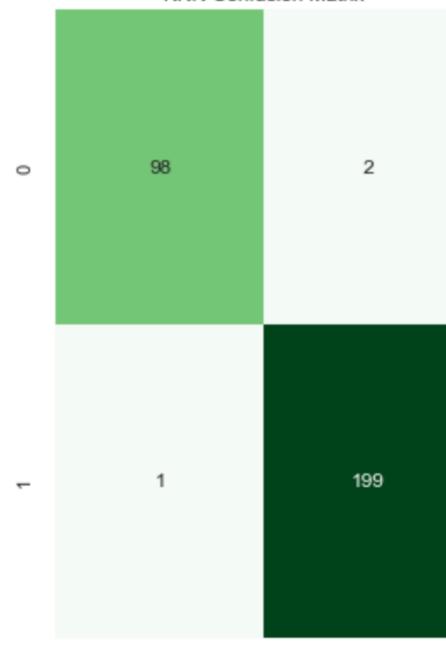
Logistic Regression Confusion Matrix : Sélection de variables RFE



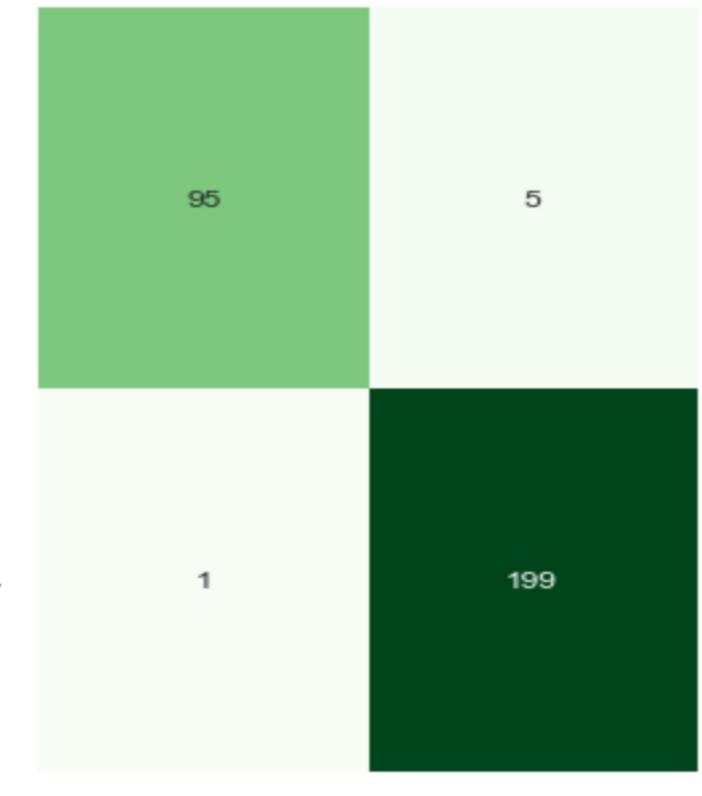
Random Forest Confusion Matrix



KNN Confusion Matrix



Kmeans Confusion Matrix



# Choix du modèle final

- Les performances des quatre modèles présentés sont excellentes et se rapprochent.
- Le K-means a donné un peu moins de bonnes performances.
- Le choix du modèle se porte sur la régression logistique qui donne les meilleures performances avec une precision = 0.99, recall= 0.99 et un f1\_score = 0.99

# Test de l'algorithme sélectionné sur le fichier billets\_production

```
1 # Fonction qui fait une prédiction sur un fichier donné en appliquant la régression logistique
2 # Elle crée les probabilités et crée une condition sur le seuil défini
3 # La colonne is_genuine est créée en se basant sur les critères prédefnis
4
5 def detecteur(df):
6     #df.loc[df['column name'] condition, 'new column name'] = 'value if condition is met'
7     df[['probabilite_Faux','probabilie_Vrai']] = sel.estimator.predict_proba(df.iloc[:,0:6])
8     df.loc[df['probabilite_Faux'] <= 0.6, 'is_genuine'] = 'True'
9     df.loc[df['probabilite_Faux'] > 0.6, 'is_genuine'] = 'False'
10    return df
```

```
[46]: 1 detecteur(billets_production)
```

```
:
```

	diagonal	height_left	height_right	margin_low	margin_up	length	probabilite_Faux	probabilie_Vrai	is_genuine
id									
A_1	171.76	104.01	103.54	5.21	3.30	111.42	0.995625	0.004375	False
A_2	171.87	104.17	104.13	6.00	3.31	112.09	0.999065	0.000935	False
A_3	172.00	104.58	104.29	4.99	3.39	111.57	0.998784	0.001216	False
A_4	172.49	104.55	104.34	4.44	3.03	113.20	0.080994	0.919006	True
A_5	171.65	103.63	103.56	3.77	3.16	113.33	0.000342	0.999658	True

# Prédiction sur un fichier fourni

# Discussion :



**FIN**

• **MERCI**