In [81]:

```
#Importation de la librairie
import pandas as pd
import numpy as np
from sklearn import decomposition
from sklearn import preprocessing
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
import matplotlib.pyplot as plt
import scipy.stats  as st
import seaborn as sns
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')
sns.set_style('dark')
from sklearn.metrics import f1_score, confusion_matrix, classification_report
from sklearn.model_selection import learning_curve
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn import metrics
import statsmodels.formula.api as smf
import statsmodels.api as sm
```

In [82]:

```
#Importation des données
Billete = pd.DataFrame(pd.read_csv('billets (1).csv',sep=';'))
```

In [83]:

```
Billete.head()
```

Out[83]:

	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

In [84]:

```
#Voir s'il existe des valeurs null
Billete.isnull().sum().sum()
```

Out[84]:

```
37
```

In [85]:

```
Billete.dtypes.value_counts()
```

Out[85]:

```
float64    6
bool       1
dtype: int64
```

In [86]:

```
#Description de nos variables
Billete.info()
#Une variable qualitative et 5 variables quantitatives
```

```
<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
```

In [87]:

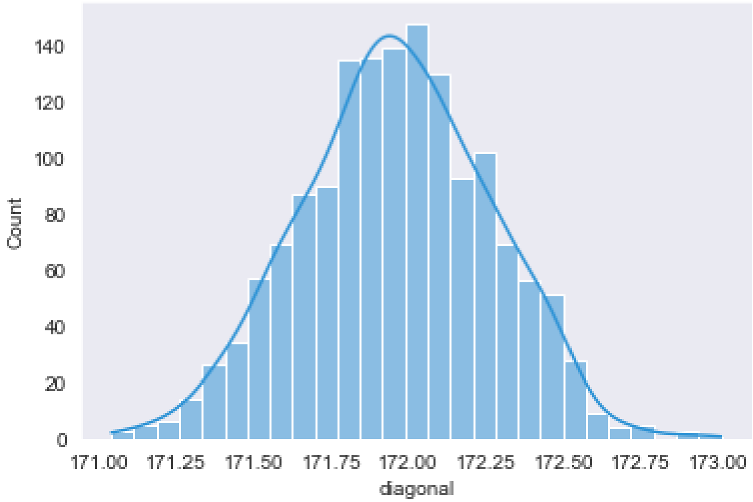
```
#Analyse de forme de notre valeur booléenne
Billete['is_genuine'].value_counts(normalize=True)
#Nous avons 67% de vrais billets et 33% de faux billets
```

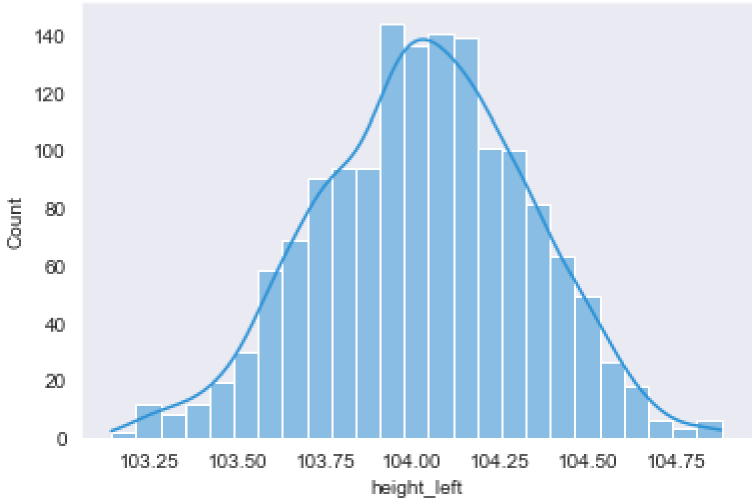
Out[87]:

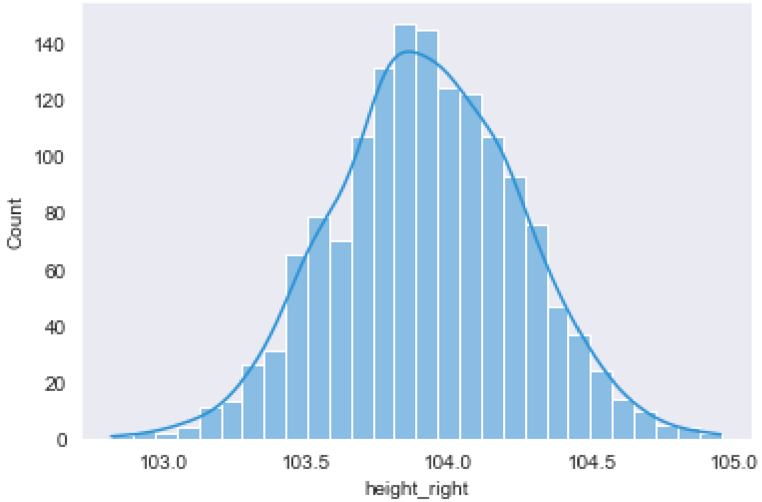
```
True     0.666667
False    0.333333
Name: is_genuine, dtype: float64
```

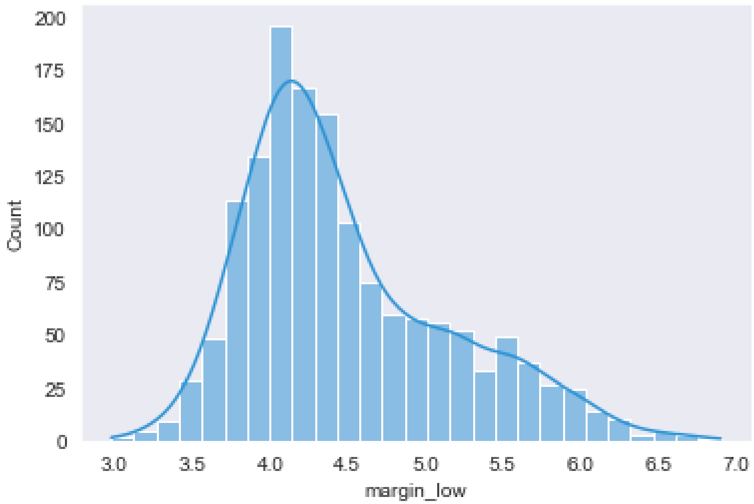
In [88]:

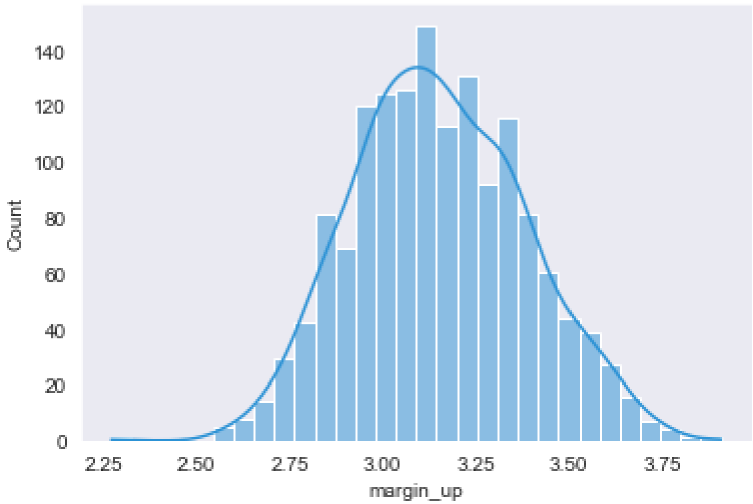
```
#Histogramme de nos différentes variables sauf la valeur booléenne
for col in Billete.select_dtypes('float'):
    plt.figure()
    sns.histplot(Billete[col], kde=True, color='#2C92D5')
```

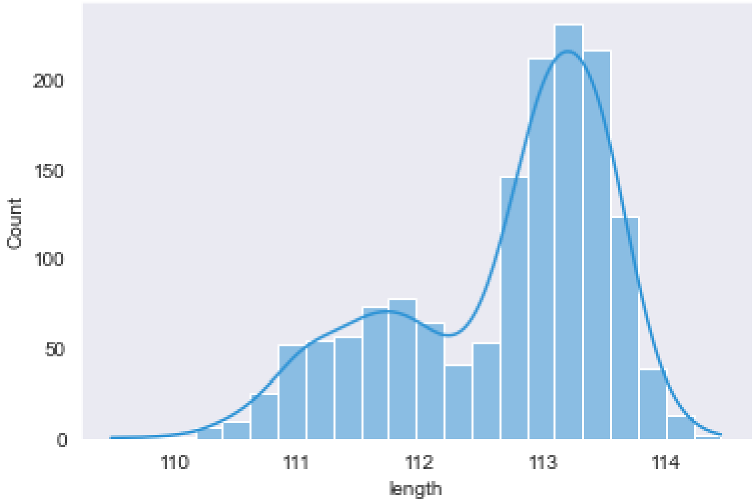



Count
height_left


Count
height_right


Count
margin_low


Count
margin_up

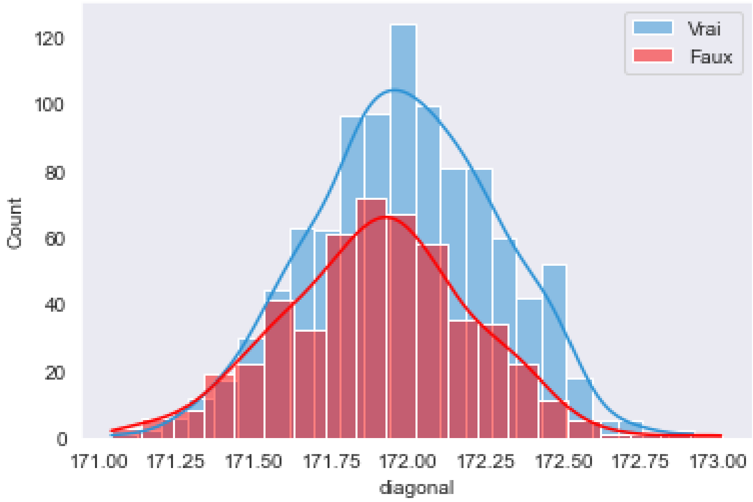


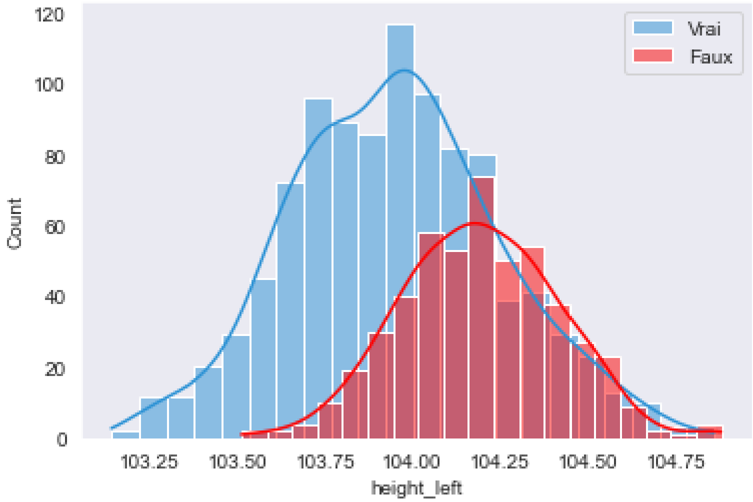
In [89]:

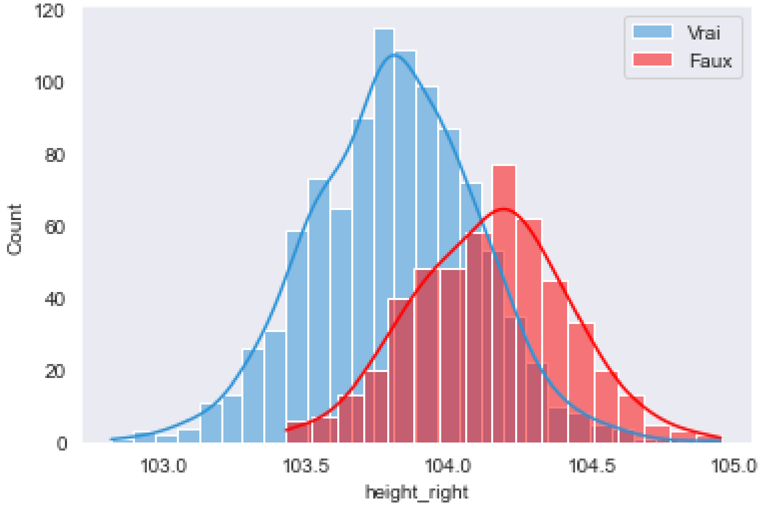
```
#Analyse de la relation entre notre variable qualitative et nos variables quantitatives
# Création de sous - échantillons True et False
df_true = Billete[Billete['is_genuine'] == True]
df_false = Billete[Billete['is_genuine'] == False]
```

In [90]:

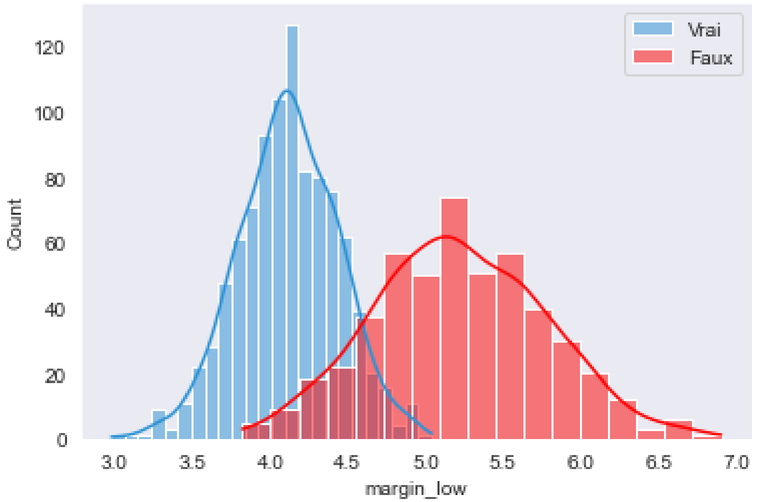
```
for col in Billete.select_dtypes('float'):
    plt.figure()
    sns.histplot(df_true[col], kde=True, label='Vrai', color='#2C92D5')
    sns.histplot(df_false[col], kde=True, label='Faux', color='red')
    plt.legend()
```


Vrai
Faux
Count
diagonal
0
20
40
60
80
100
120
171.00
171.25
171.50
171.75
172.00
172.25
172.50
172.75
173.00

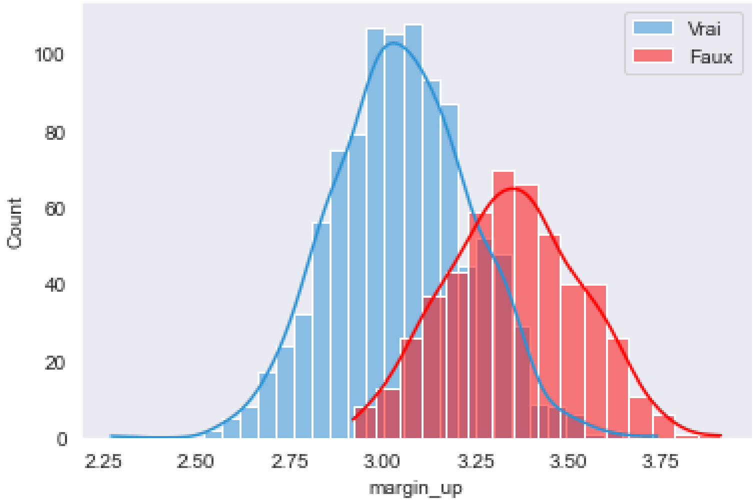

Vrai
Faux
Count
height_left
0
20
40
60
80
100
120
103.25
103.50
103.75
104.00
104.25
104.50
104.75

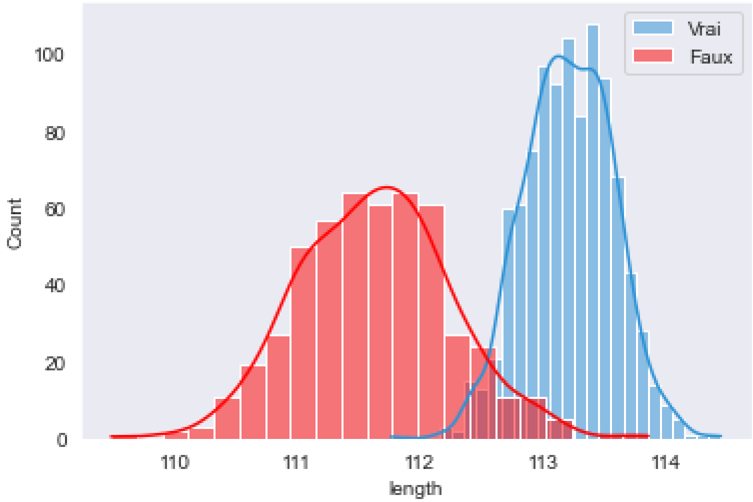


Vrai
Faux
Count
0
20
40
60
80
100
120
103.0
103.5
104.0
104.5
105.0
height_right



Vrai
Faux
Count
0
20
40
60
80
100
120
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
margin_low





In [91]:

```
#Test d'agostino
from scipy import stats
numeric_columns = Billete.select_dtypes(include = ['int32','float64']).columns
#numeric_columns = numeric_columns[1:]
for column in numeric_columns:
    print('________\n{}'.format(column))
    k2, p = stats.normaltest(Billete[column],
                             axis=0,
                             nan_policy = 'omit')
    alpha = 5e-2
```

```
    print("p = {:g}".format(p))
    if p < alpha:  # null hypothese: x a une distribution normale
        print("H0 est rejetée : {} on ne considère pas l'hypothèse de normalité".format(column))
    else:
        print("H0 ne peut être rejetée :{}, on considère l'hypothèse de normalité".format(column))
```

```
________
diagonal
p = 0.526269
H0 ne peut être rejetée :diagonal, on considère l'hypothèse de normalité
________
height_left
p = 0.0866879
H0 ne peut être rejetée :height_left, on considère l'hypothèse de normalité
________
height_right
p = 0.987581
H0 ne peut être rejetée :height_right, on considère l'hypothèse de normalité
________
margin_low
p = 1.27242e-31
H0 est rejetée : margin_low on ne considère pas l'hypothèse de normalité
________
margin_up
p = 0.00675383
H0 est rejetée : margin_up on ne considère pas l'hypothèse de normalité
________
length
p = 1.58771e-30
H0 est rejetée : length on ne considère pas l'hypothèse de normalité
```

In [92]:

```
#Test de kruskal
x = df_true['diagonal']
y = df_false['diagonal']
stats.kruskal(x, y)
#La p-value nous indique que la probabilité de rejeter l'hypothèse nulle alors qu'elle serait vraie est inférieure à 0.0005.
#Dans notre cas, on peut rejeter en toute confiance l'hypothèse nulle d'absence de différence significative entre nos variables.
```

Out[92]:

```
KruskalResult(statistic=24.961601644703098, pvalue=5.848354129462713e-07)
```

In [93]:

```
#Test de student
```

```
from scipy.stats import ttest_ind
```

In [94]:

```
def t_test(col):
    from scipy.stats import ttest_ind
    alpha = 0.05
    stat, p_val = ttest_ind(df_true.sample(df_false.shape[0])[col], df_false[col])
    if p_val < alpha:
        return 'Cette variable influe sur le statut du billet !!!'
    else:
        return 'Cette variable n\'a aucune influence sur le statut du billet !!!'
```

In [95]:

```
for col in Billete.select_dtypes('float'):
    print(f'{col :-<50} {t_test(col)}')
```

```
diagonal------------------------------------------ Cette variable influe sur le statut du billet !!!
height_left--------------------------------------- Cette variable influe sur le statut du billet !!!
height_right-------------------------------------- Cette variable influe sur le statut du billet !!!
margin_low---------------------------------------- Cette variable n'a aucune influence sur le statut du billet !!!
margin_up----------------------------------------- Cette variable influe sur le statut du billet !!!
length-------------------------------------------- Cette variable influe sur le statut du billet !!!
```

In [96]:

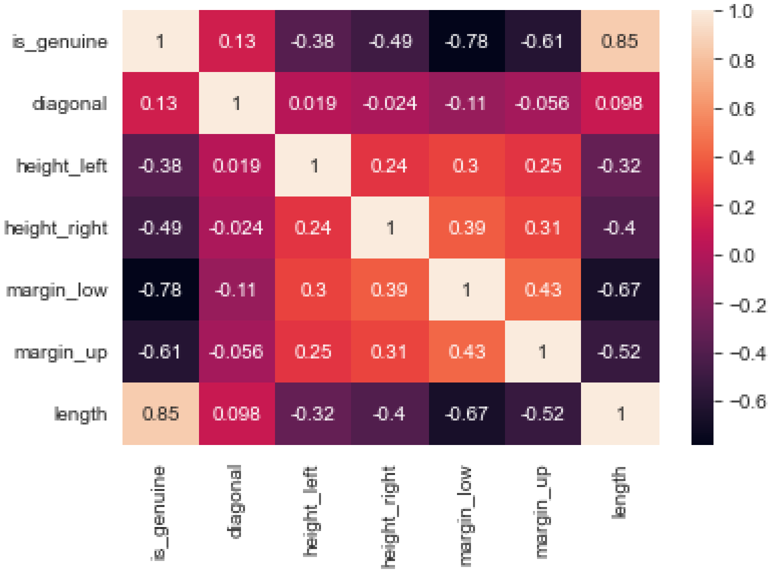
```
#Corrélation entre les différentes variables
Billete.corr()
```

Out[96]:

	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length
is_genuine	1.000000	0.132756	-0.379833	-0.485092	-0.783032	-0.606262	0.849285
diagonal	0.132756	1.000000	0.019472	-0.024492	-0.111534	-0.055649	0.097587
height_left	-0.379833	0.019472	1.000000	0.242279	0.302643	0.246522	-0.320863
height_right	-0.485092	-0.024492	0.242279	1.000000	0.391085	0.307005	-0.401751
margin_low	-0.783032	-0.111534	0.302643	0.391085	1.000000	0.431606	-0.666753
margin_up	-0.606262	-0.055649	0.246522	0.307005	0.431606	1.000000	-0.520575
length	0.849285	0.097587	-0.320863	-0.401751	-0.666753	-0.520575	1.000000

In [97]:

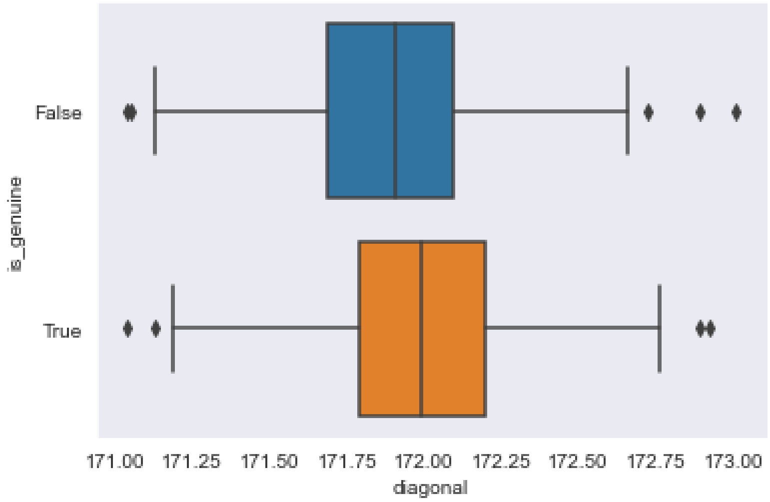
```
cor_B = Billete.corr()
sns.heatmap(cor_B, annot = True)

plt.show()
#Les variables margin_low et is_genuine sont faiblement corrélées
```



In [98]:

```
#Analyse descriptive des données
import  seaborn  as  sns
sns . boxplot ( data = Billete , orient  =  'h' ,  x = 'diagonal' ,  y = 'is_genuine' )
```

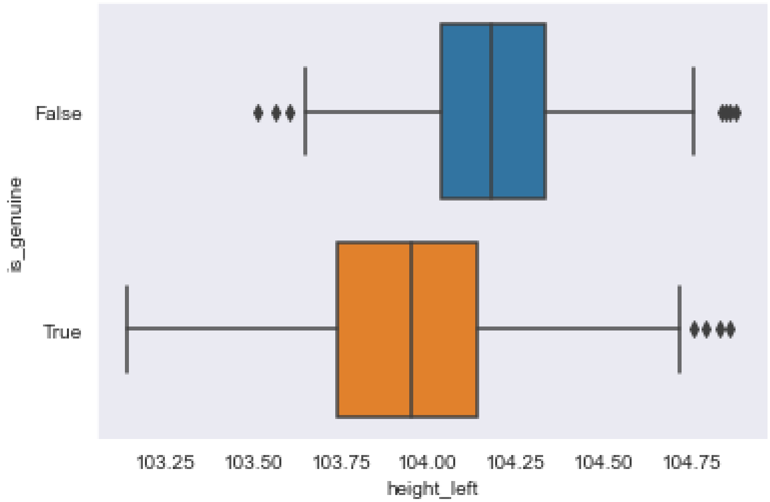
Out[98]: <AxesSubplot:xlabel='diagonal', ylabel='is_genuine'>



In [99]:

```
sns . boxplot ( data = Billete , orient  =  'h' ,  x = 'height_left' ,  y = 'is_genuine' )
```

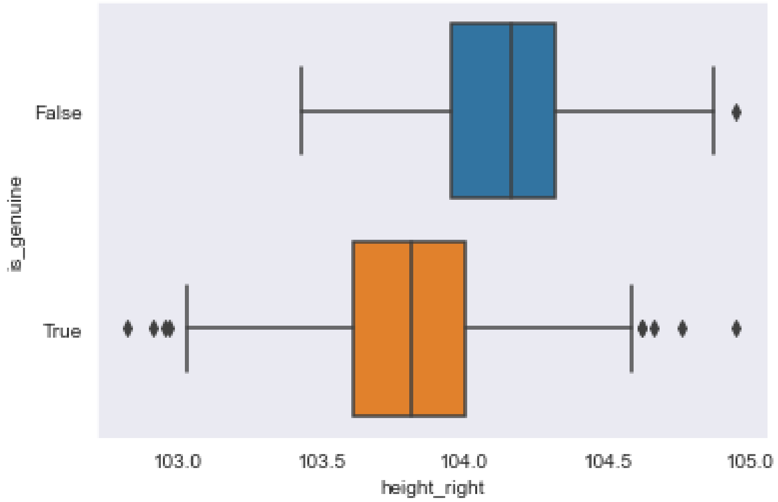
Out[99]: <AxesSubplot:xlabel='height_left', ylabel='is_genuine'>



In [100]:

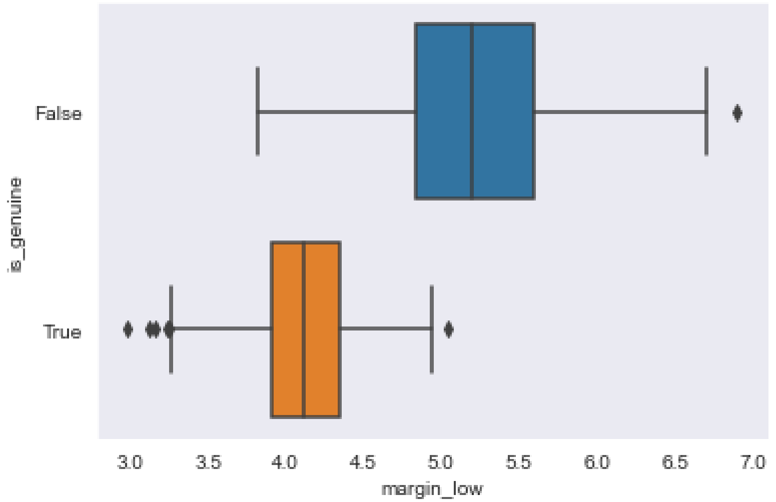
```
sns . boxplot ( data = Billete , orient  =  'h' ,  x = 'height_right' ,  y = 'is_genuine' )
```

Out[100]: <AxesSubplot:xlabel='height_right', ylabel='is_genuine'>



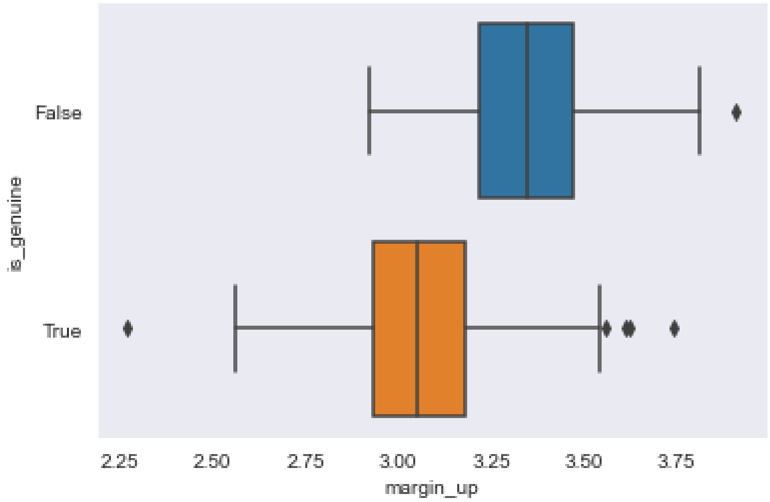
In [101]:
```
sns . boxplot ( data = Billete , orient  =  'h' ,  x = 'margin_low' ,  y = 'is_genuine' )
```

Out[101]:
```
<AxesSubplot:xlabel='margin_low', ylabel='is_genuine'>
```



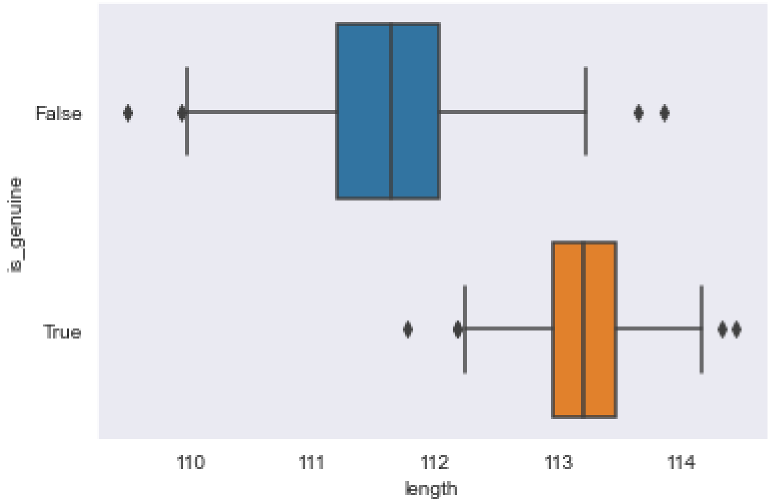
In [102]:
```
sns . boxplot ( data = Billete , orient  =  'h' ,  x = 'margin_up' ,  y = 'is_genuine' )
```

Out[102]:
```
<AxesSubplot:xlabel='margin_up', ylabel='is_genuine'>
```



In [103]:
```
sns . boxplot ( data = Billete , orient  =  'h' ,  x = 'length' ,  y = 'is_genuine' )
```

Out[103]:
```
<AxesSubplot:xlabel='length', ylabel='is_genuine'>
```



In [104]:
```
#On transforme la valeur booléenne en 0 (False) ou 1 (True)
Billete["is_genuine"] = Billete["is_genuine"].astype(int)
```

In [105]:
```
#On régresse margin_low en fonction des autres variables du dataframe

import statsmodels.formula.api as smf

#On fait ici une régression linéaire simple
reg = smf.ols('margin_low~is_genuine+margin_up+diagonal+length+height_right+height_left', data=Billete).fit()
print(reg.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             margin_low   R-squared:                       0.617
Model:                            OLS   Adj. R-squared:                  0.615
Method:                 Least Squares   F-statistic:                     390.7
Date:                Fri, 04 Feb 2022   Prob (F-statistic):          4.75e-299
Time:                        20:19:16   Log-Likelihood:                -774.14
No. Observations:                1463   AIC:                             1562.
Df Residuals:                    1456   BIC:                             1599.
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept        2.8668      8.316      0.345      0.730     -13.445      19.179
is_genuine      -1.1406      0.050    -23.028      0.000      -1.238      -1.043
margin_up       -0.2128      0.059     -3.621      0.000      -0.328      -0.098
diagonal        -0.0130      0.036     -0.364      0.716      -0.083       0.057
length          -0.0039      0.023     -0.166      0.868      -0.050       0.042
height_right     0.0267      0.038      0.701      0.484      -0.048       0.102
height_left      0.0283      0.039      0.727      0.468      -0.048       0.105
==============================================================================
Omnibus:                       21.975   Durbin-Watson:                   2.038
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               37.993
Skew:                           0.061   Prob(JB):                     5.62e-09
Kurtosis:                       3.780   Cond. No.                     1.95e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.95e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [106]:
```
#On constate ici que certains paramètres ne sont pas forcément pertinent car leur p-valeur n'est pas inférieure à 5 %
#De ce fait deux variables restent significatives: is_genuine et margin_up. On régresse margin_low en fonction de ces deux variabl
```

```
reg_mult = smf.ols('margin_low~is_genuine+margin_up', data=Billete).fit()
print(reg_mult.summary())
#Notre R2 est de 61.7%
```

```
                            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, 04 Feb 2022   Prob (F-statistic):         1.24e-304
Time:                        20:19:18   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
is_genuine    -1.1632      0.029    -40.477      0.000      -1.220      -1.107
margin_up     -0.2119      0.059     -3.612      0.000      -0.327      -0.097
==============================================================================
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
==============================================================================

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

In [107]:

```
#On cherche ensuite à faire une regression linéaire, en utilisant le modèle ci-dessus,
#On va donc pouvoir estimer les valeurs manquantes de margin_low

#On a ici l'index des valeurs manquantes
v_manq = Billete[Billete.isnull().any(axis=1)].index.tolist()
i=0
#On créé une boucle qui va déterminer le margin_low pour chaque ligne ayant une valeur manquante
for i in v_manq:
    prev = pd.DataFrame({'is_genuine': Billete.loc[[i], 'is_genuine'].item(),
                         'margin_up': Billete.loc[[i], 'margin_up'].item()}, index=[0])
```

```
    v_margin_low = reg_mult.predict(prev)
    Billete.loc[[i], 'margin_low'] = round(v_margin_low[0], 2)
```

In [108]:

```
#créé un dataframe d'apprentissage et un de test
from sklearn.model_selection import train_test_split
X_train, X_test = train_test_split(Billete, train_size=0.8)
```

In [109]:

```
#On garde uniquement les variables ayant des p-valeurs inférieurs à 5% :
import statsmodels.api as sm

#On rajoute .Binomial() pour faire comprendre que l'on souhaite faire une régression logistique
r_log = smf.glm('is_genuine~margin_low+margin_up+length+height_right',
                   data=X_train, family=sm.families.Binomial()).fit()
print(r_log.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:             is_genuine   No. Observations:                 1200
Model:                            GLM   Df Residuals:                     1195
Model Family:                Binomial   Df Model:                            4
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -28.765
Date:                Fri, 04 Feb 2022   Deviance:                       57.531
Time:                        20:19:22   Pearson chi2:                 2.42e+03
No. Iterations:                    10   Pseudo R-squ. (CS):             0.7076
Covariance Type:            nonrobust
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept     -145.1308    179.043     -0.811      0.418    -496.048     205.787
margin_low      -6.4534      1.124     -5.742      0.000      -8.656      -4.251
margin_up      -11.0142      2.573     -4.280      0.000     -16.058      -5.971
length           6.6058      1.115      5.925      0.000       4.421       8.791
height_right    -5.1353      1.727     -2.974      0.003      -8.519      -1.751
================================================================================
```

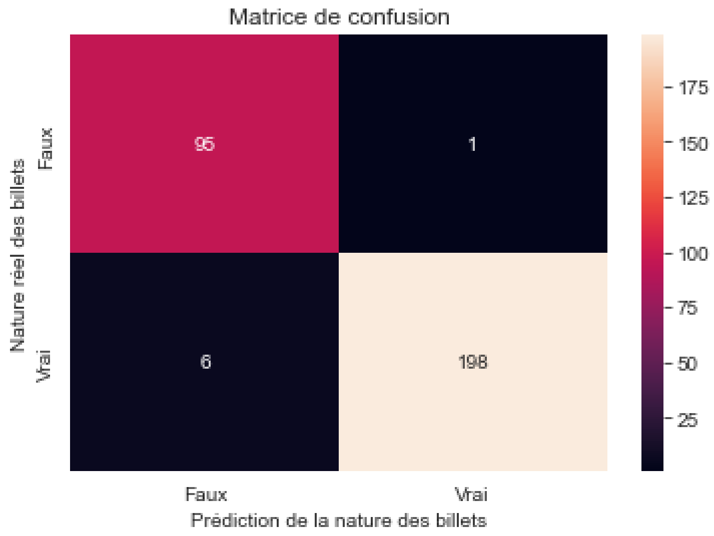
In [117]:

```
#Créer un algorithme qui prédit la nature des billets :

count=0
y_true = []
```

```
y_pred = []
#index des billets sous forme de liste
X_test_index_list = X_test.index.tolist()

#Pour chaque billet :
for i in X_test_index_list :
#On récupère les valeurs qui nous intéressent :
    a_prev = pd.DataFrame({'height_right': X_test.loc[[i], 'height_right'].item(),
                          'margin_low': X_test.loc[[i], 'margin_low'].item(),
                          'margin_up': X_test.loc[[i], 'margin_up'].item(),
                          'length': X_test.loc[[i], 'length'].item()}, index=[0])
#On prédit la valeur estimé et on récupère la valeur exact
    predict_billets = round(r_log.predict(a_prev).item())
    r_init = X_test.loc[[i], 'is_genuine'].item()
    y_true.append(r_init)
    y_pred.append(predict_billets)
```

In [118]:

```
#Afficher la matrice de confusion
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

mat_conf = confusion_matrix(y_true, y_pred)
ax= plt.subplot()
sns.heatmap(mat_conf, annot=True, fmt='g', ax=ax);

# labels, title and ticks
ax.set_xlabel('Prédiction de la nature des billets');ax.set_ylabel('Nature réel des billets');
ax.set_title('Matrice de confusion');
ax.xaxis.set_ticklabels(['Faux', 'Vrai']); ax.yaxis.set_ticklabels(['Faux', 'Vrai']);
```



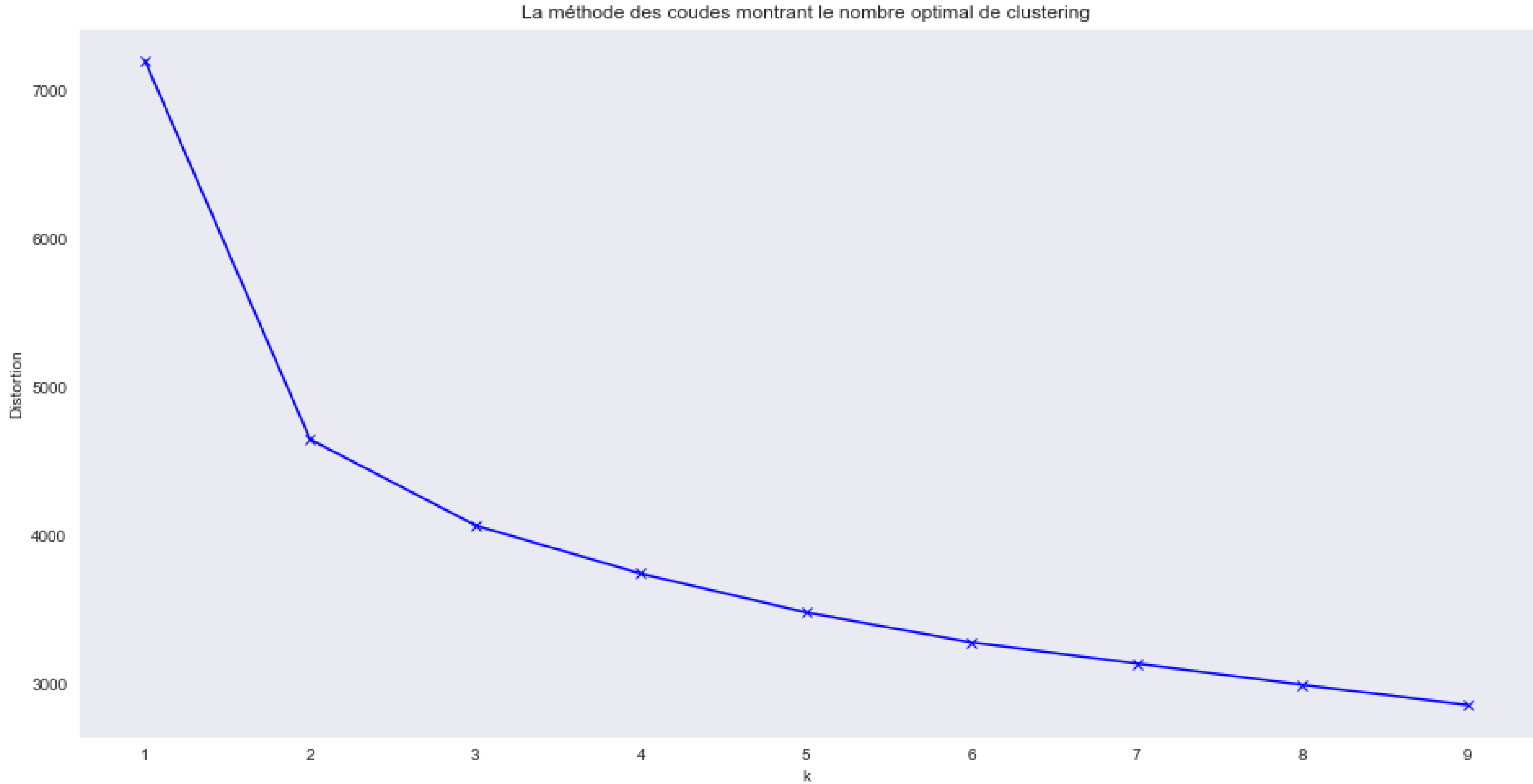
In [119]:

```
#On centre-réduit nos données de manière à avoir des variables exploitables et de même grandeur
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_train_scaled = pd.DataFrame(X_train_scaled, columns =['is_genuine', 'diagonal', 'height_left',
                                                         'height_right','margin_low','margin_up','length'])
X_test_scaled = sc.fit_transform(X_test)
X_test_scaled = pd.DataFrame(X_test_scaled, columns =['is_genuine', 'diagonal', 'height_left',
                                                         'height_right','margin_low','margin_up','length'])
```

In [120]:

```
from sklearn.cluster import KMeans

X_train_drop_scaled = X_train_scaled.drop('is_genuine', axis=1)

#"Méthode des coudes" pour déterminer le nombre de clusters pour les kmeans
distortions = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(X_train_drop_scaled)
    distortions.append(kmeanModel.inertia_)

plt.figure(figsize=(16,8))
```

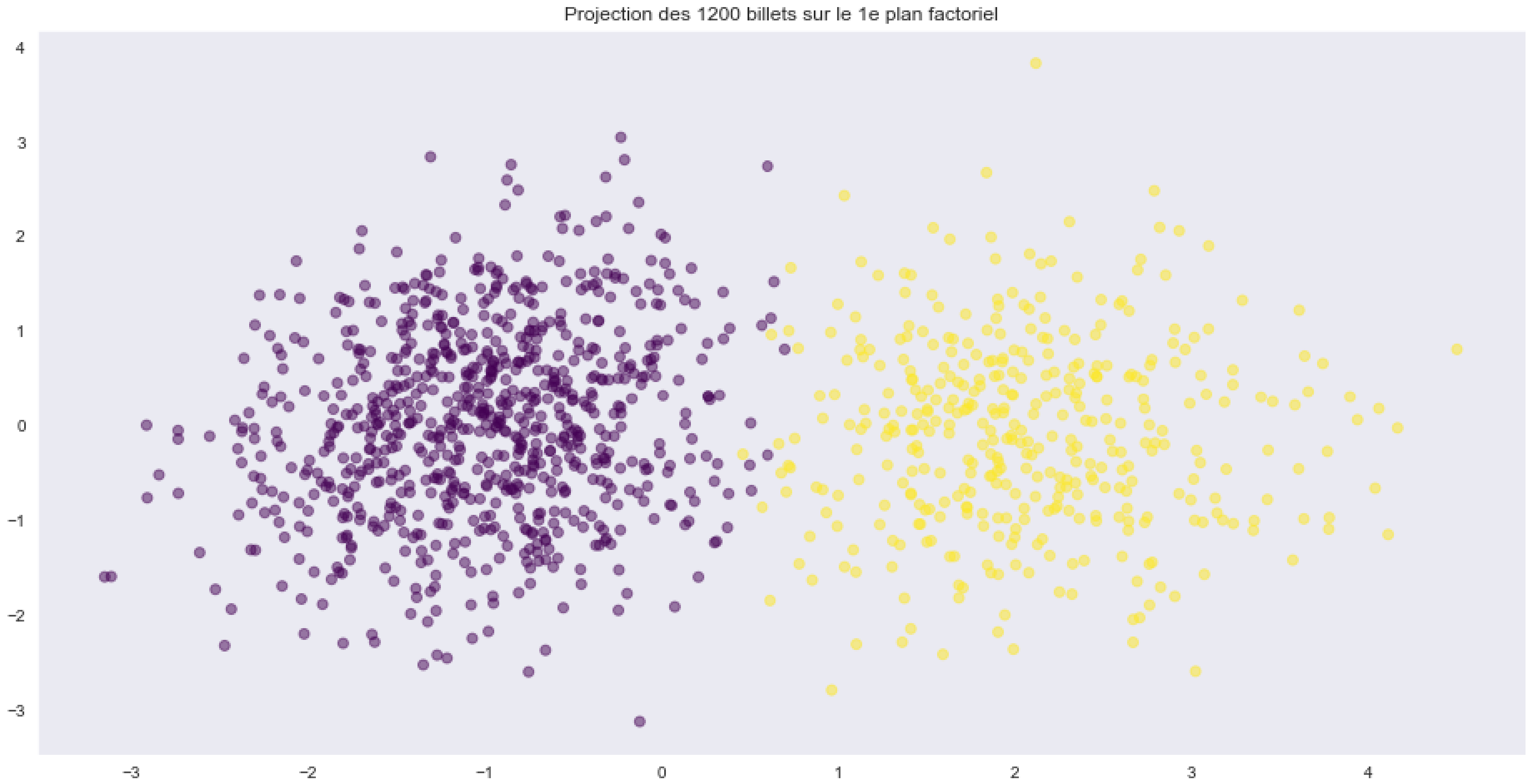
```
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('La méthode des coudes montrant le nombre optimal de clustering')
plt.show()

#Le nombre optimal de clusters semble être 2
```



In [121]:

```
#On va faire un clustering des billets, avec la méthode des k-means
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn import decomposition

# Nombre de clusters souhaités
n_clust = 2
# Clustering par K-means
km = KMeans(n_clusters=n_clust)
km.fit(X_train_drop_scaled)
# Récupération des clusters attribués à chaque pays
clusters = km.labels_
# Affichage du clustering par projection des pays sur le premier plan factoriel
pca = decomposition.PCA(n_components=2).fit(X_train_drop_scaled)
X_projected = pca.transform(X_train_drop_scaled)
# On récupère les centroïdes
centroids  = km.cluster_centers_
plt.figure(figsize=(16,8))
plt.scatter(X_projected[:, 0], X_projected[:, 1], c=clusters.astype(np.float), cmap = 'viridis', alpha=.5)
plt.title("Projection des {} billets sur le 1e plan factoriel".format(X_projected.shape[0]))
plt.show(block=False)
```



In [122]:

```
#On tente de prédire la nature des billets
import numpy as np

X_test_drop_scaled = X_test_scaled.drop('is_genuine', axis=1)

x_true = X_test.is_genuine.tolist()
x_pred = []
X_test_index_list = X_test_scaled.index.tolist()

for i in X_test_index_list:
    diff = centroids-X_test_drop_scaled.loc[i].tolist()
    #distance euclidienne
```

```
    dist = np.sqrt(np.sum(diff**2, axis=-1))
    if dist[0] >= dist[1]:
        x_pred.append(1)
    else :
        x_pred.append(0)
```

In [123]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

mat_conf_1 = confusion_matrix(x_true, x_pred)
ax= plt.subplot()
sns.heatmap(mat_conf_1, annot=True, fmt='g', ax=ax);

# labels, title and ticks
ax.set_xlabel('Prédiction de la nature des billets');ax.set_ylabel('Nature réel des billets');
ax.set_title('Matrice de confusion');
ax.xaxis.set_ticklabels(['Faux', 'Vrai']); ax.yaxis.set_ticklabels(['Faux', 'Vrai']);
```