ONCFM

Détecter des faux billets

OPENCLASSROOMS

Réalisé par Yu-Ting HUANG

Mission

- Mise en place d'un modèle capable d'identifier automatiquement les vrais des faux billets.

Plan

01 **Présentation des données**

02 **Analyse exploratoire des données (AED)**

03 **Traitement des valeurs manquantes**
- Régression linéaire

04 **Analyse des composants principaux (ACP)**

05 **Modèles de prediction**
- K-MEANS
- K-NN
- Régression logistique

06 **Conclusion**

07 **Test soutenance**

Présentation des données

Jeu de données

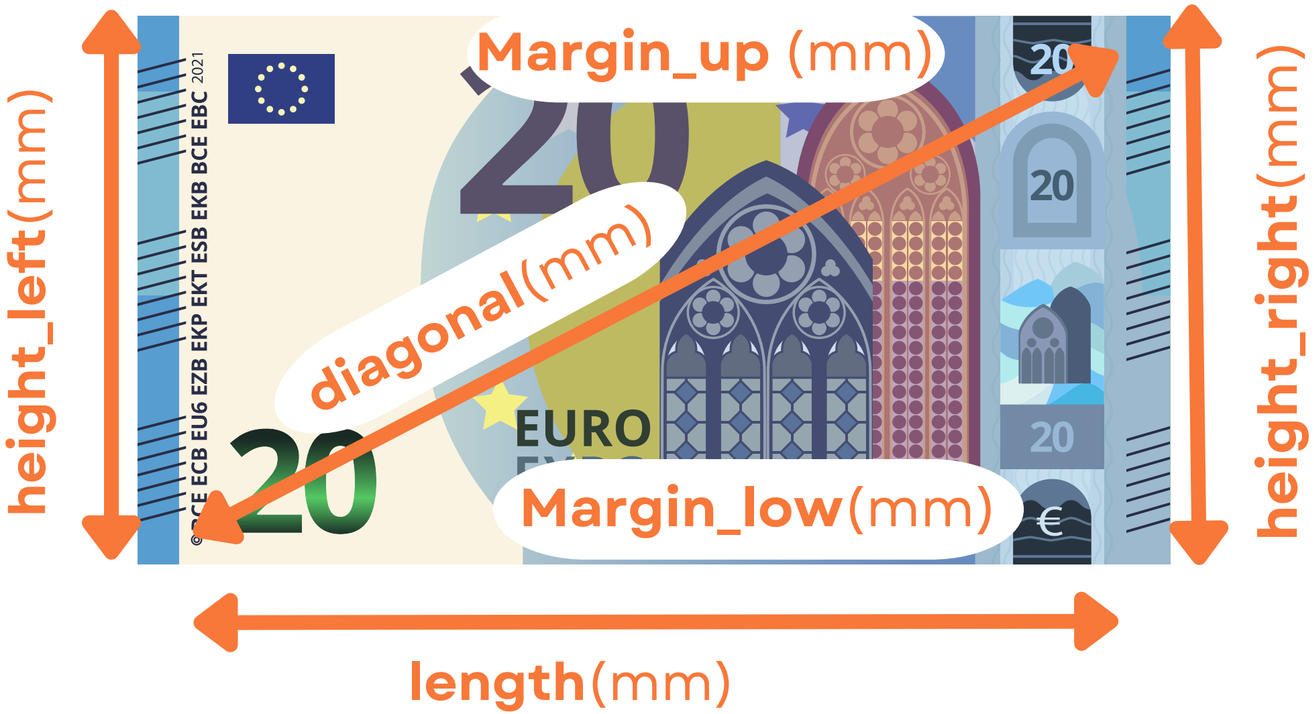
is_genuine : variable qualitative

70% vrais billets

30% faux billets

	count	mean	std	min	25%	50%	75%	max
is_genuine	1500.0	0.666667	0.471562	0.00	0.000	1.00	1.00	1.00
diagonal	1500.0	171.958440	0.305195	171.04	171.750	171.96	172.17	173.01
height_left	1500.0	104.029533	0.299462	103.14	103.820	104.04	104.23	104.88
height_right	1500.0	103.920307	0.325627	102.82	103.710	103.92	104.15	104.95
margin_low	1463.0	4.485967	0.663813	2.98	4.015	4.31	4.87	6.90
margin_up	1500.0	3.151473	0.231813	2.27	2.990	3.14	3.31	3.91
length	1500.0	112.678500	0.872730	109.49	112.030	112.96	113.34	114.44

1500 billets avec 7 variables




```
df_billets1.margin_low.isna().sum()
```

```
37
```

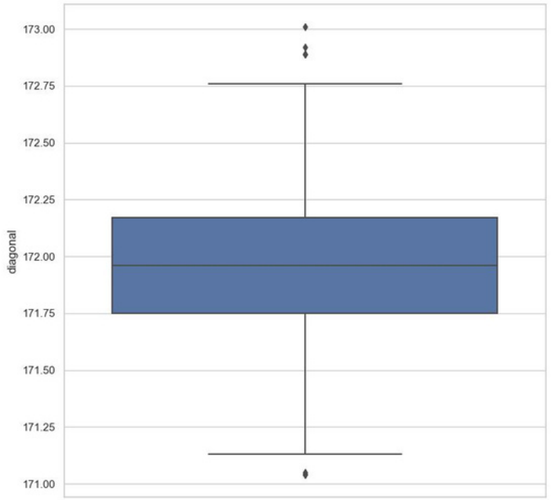
37 valeurs nulles dans **margin_low**

Analyse exploratoire des données (AED)

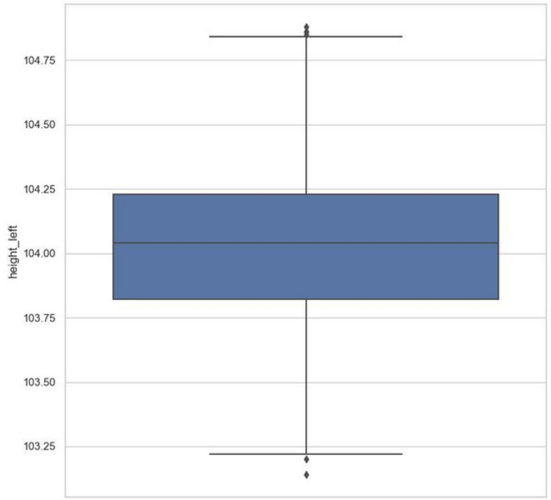
Univariée

Repartition des valeurs par variables

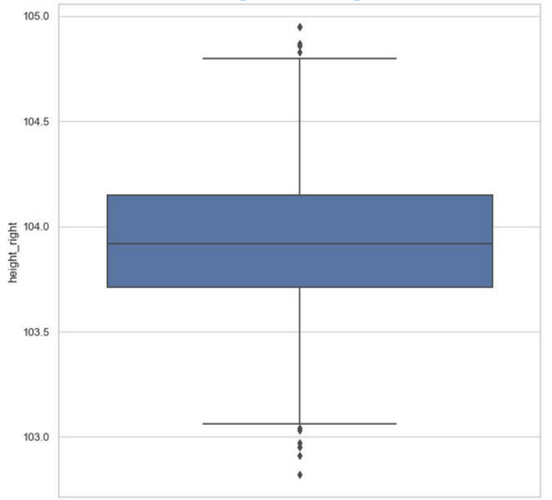
diagonal



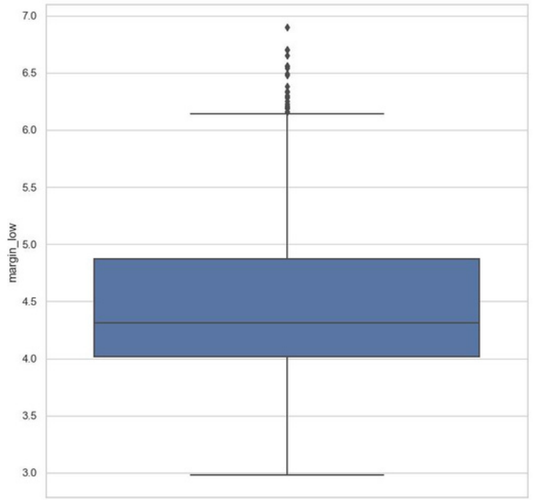
height_left



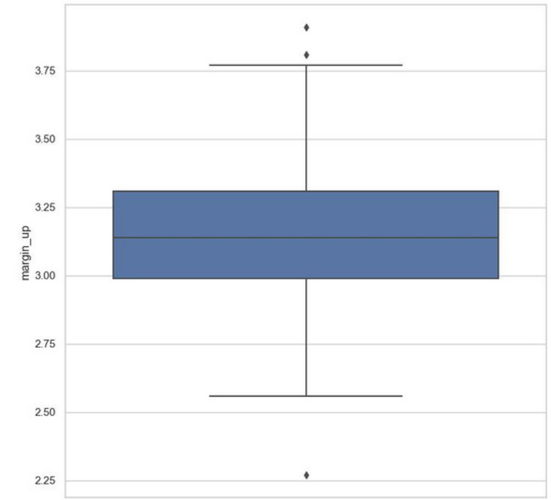
height_right



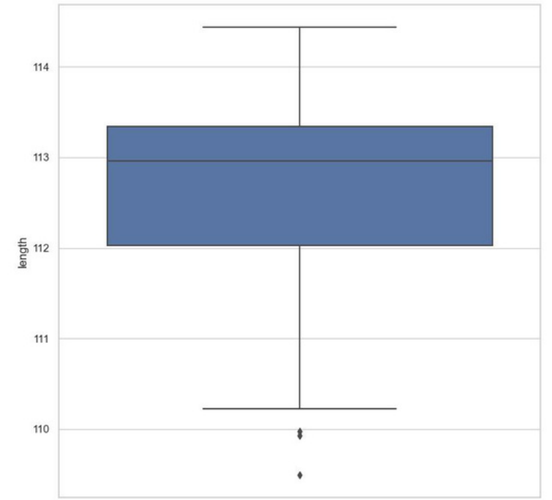
margin_low



margin_up

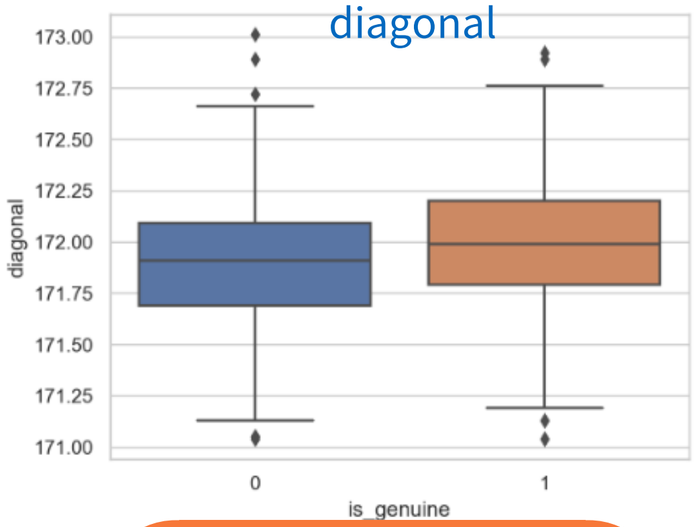


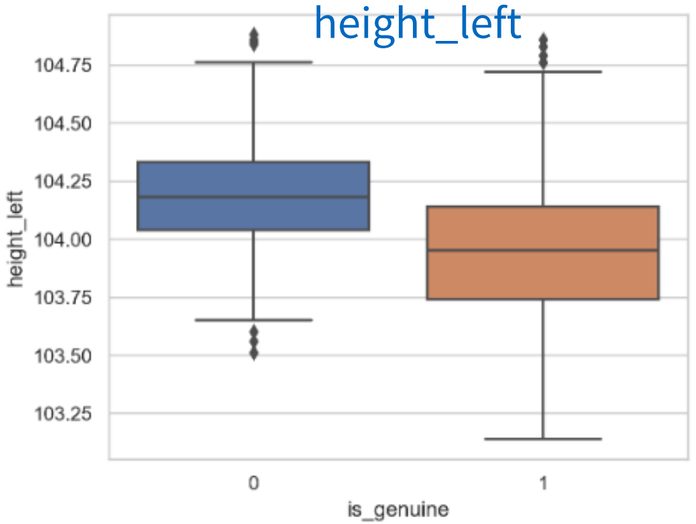
length

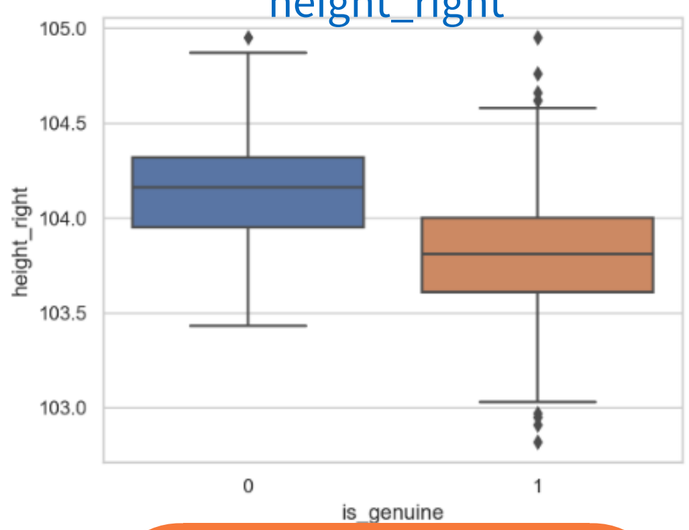


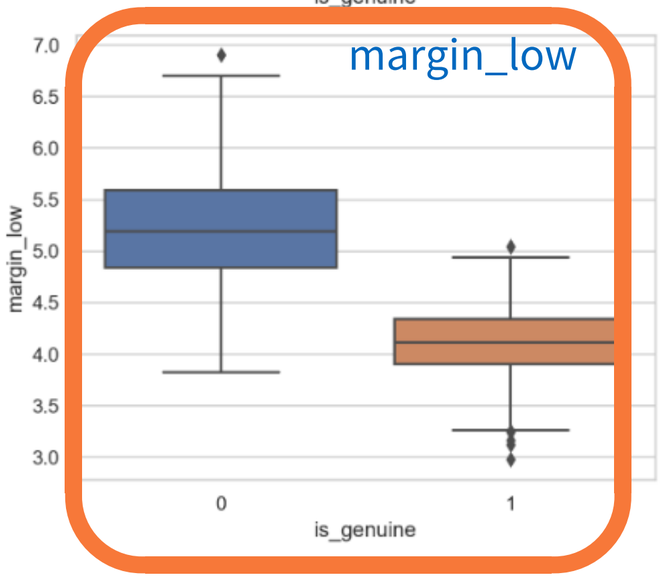
Bivariée - boxplot

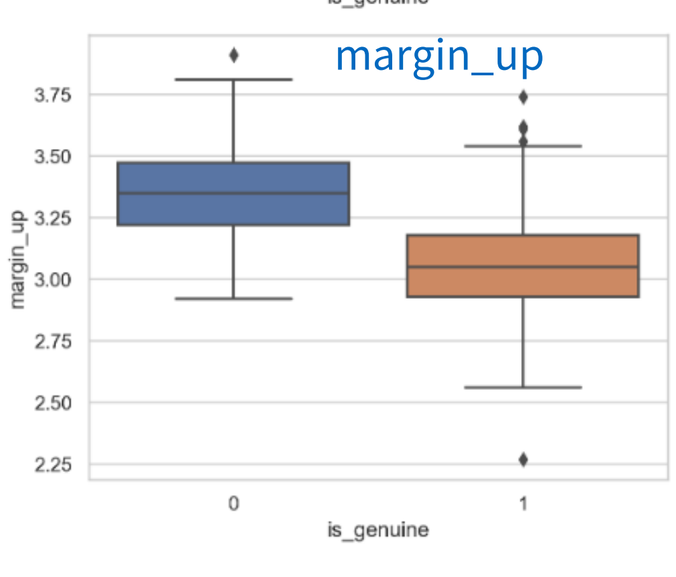
- 0 : Faux billets
- 1 : Vrai billets

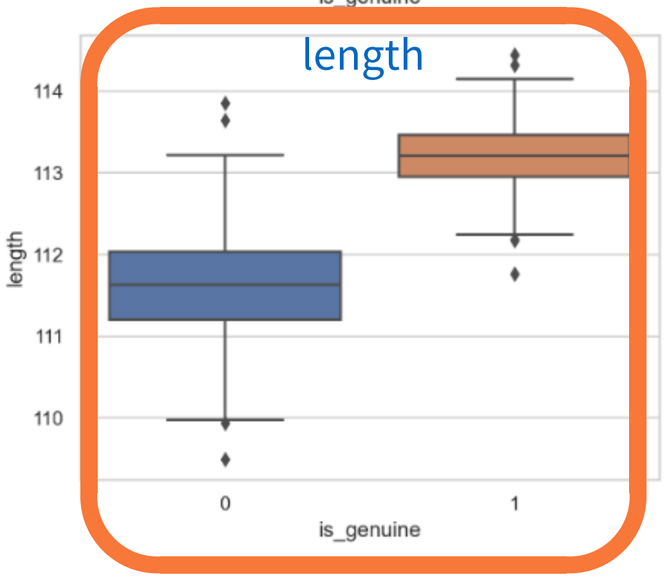




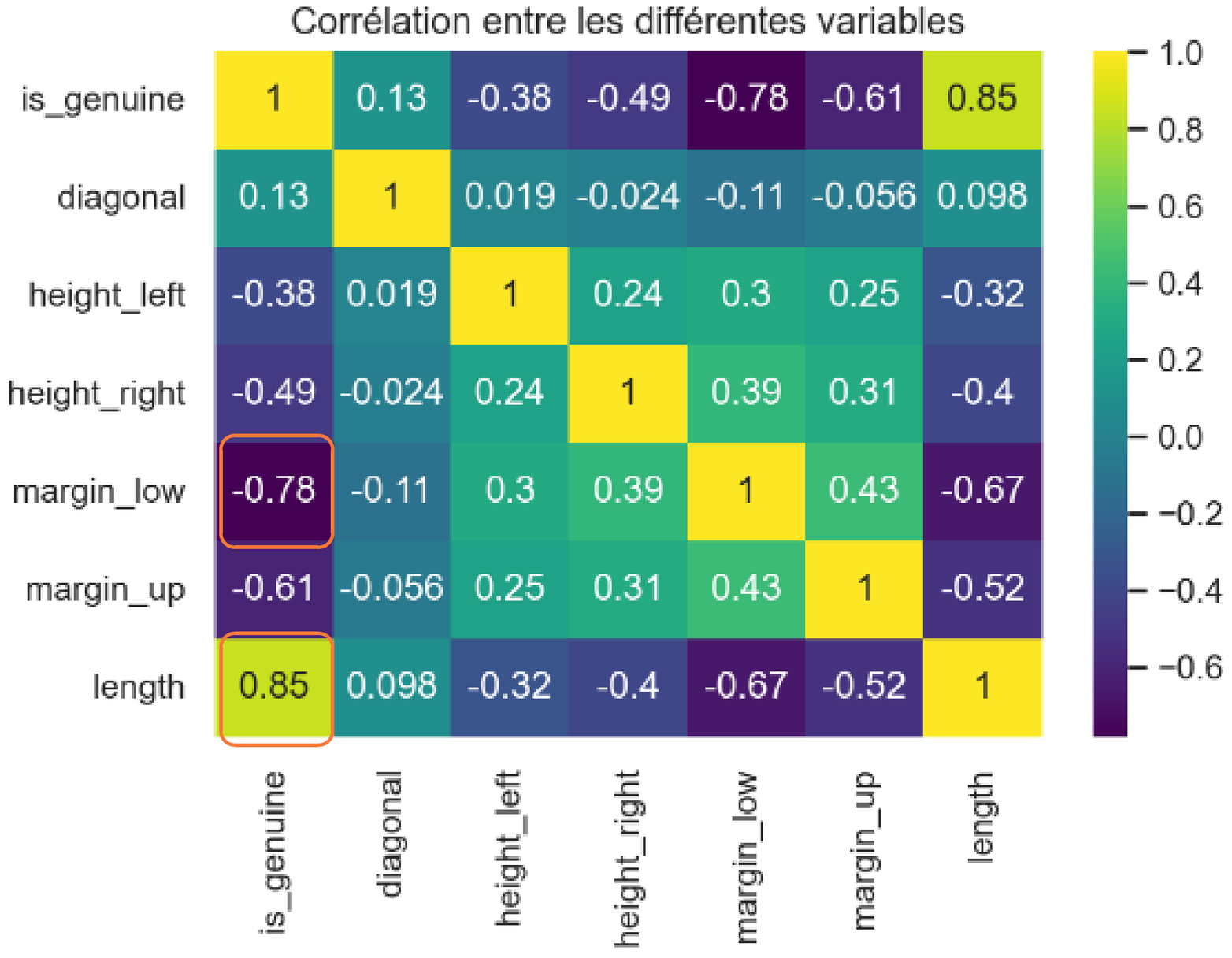




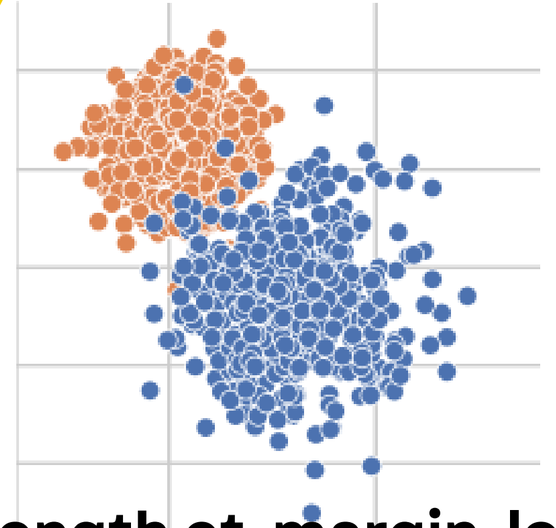




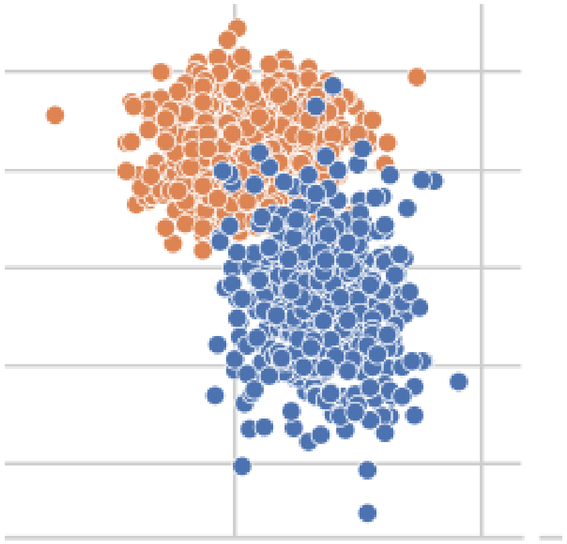
Bivariée - Matrice de corrélation



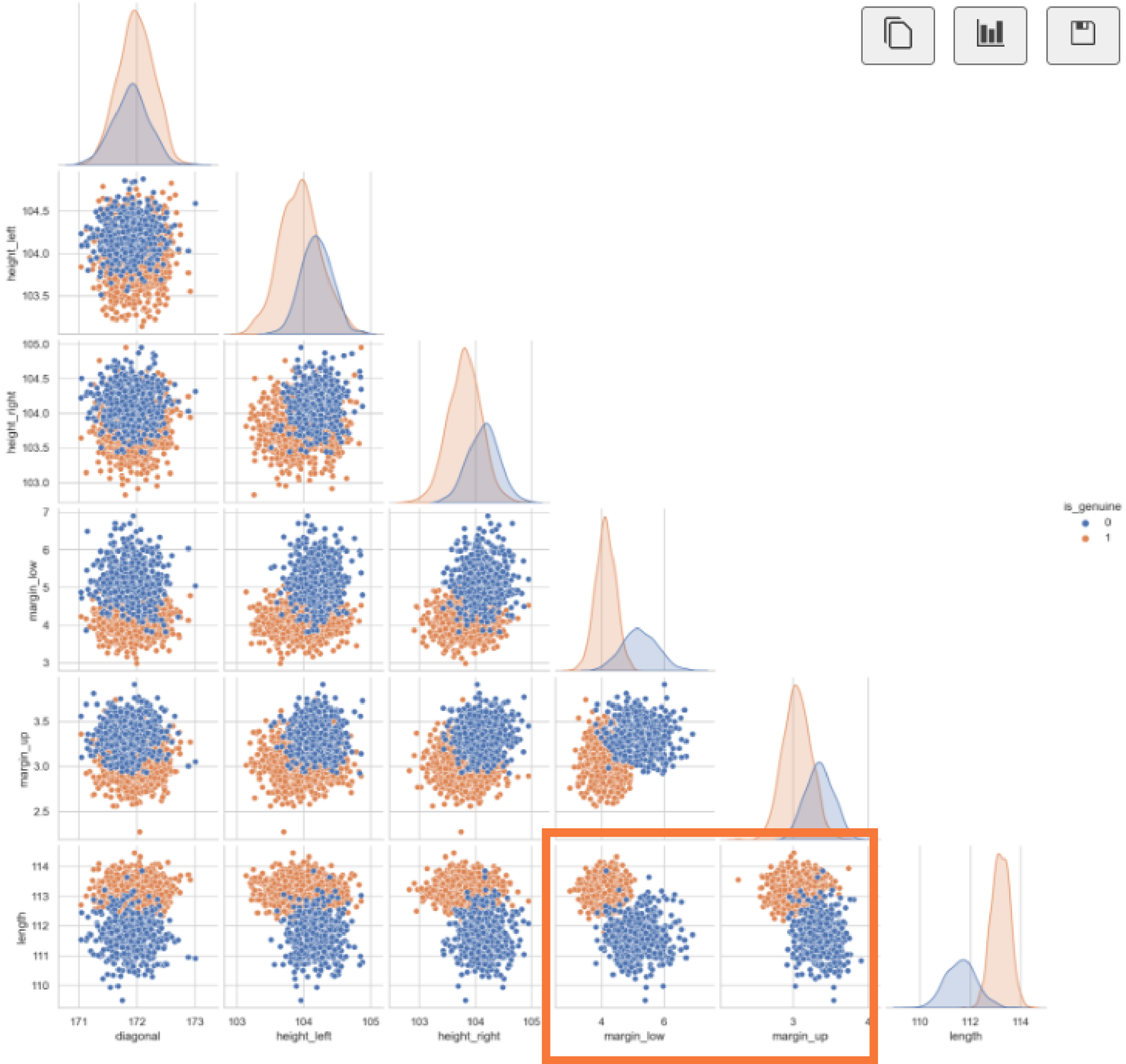
Bivariée -Pairplot



length et margin_low



length et margin_up



Traitement des valeurs manquantes

Imputation par régression linéaire multiple

Création du modèle initial → **Validation du modèle** → **Optimisation du modèle** → **Analyse des résultats** → **Application du modèle**

- Utilisation de toutes les features
- Sélection des variables X ayant une forte influence sur la variable de prédiction
- Analyse des résidus
- Application du modèle pour prédire les valeurs nulles dans 'margin_low

Création du modèle initial

```
# data prep
df = df_billets1.dropna(axis=0).reset_index(drop=True).copy()
X = df[["diagonal","height_left","height_right", "margin_up", "length"]]
y = df["margin_low"]


# Division des données en données de train et de test
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size =0.2, random_state =1)

# Training
# Premier modele avec totues les features
model_linreg_first = LinearRegression()
model_linreg_first =model_linreg_first.fit(X_train, y_train)
```

✓ 0.0s

Validation du modèle

Le modèle peut être amélioré.

```
# Evaluation
print("r2 score:", model_linreg_first.score(X_train, y_train))
print("intercept:", model_linreg_first.intercept_)
print("slope:", model_linreg_first.coef_) # regression line往下或往上
```

✓ 0.0s

```
r2 score: 0.4731708830096394
intercept: 24.34268556331221
slope: [-0.12199179  0.16999104  0.27531021  0.25363622 -0.40798163]
```

Optimisation du modèle

P < 0.05 : la variable est significative
P > 0.05 : la variable n'est pas significative

On utilise toutes les variables.

OLS Regression Results

Dep. Variable:	margin_low	R-squared:	0.473
Model:	OLS	Adj. R-squared:	0.471
Method:	Least Squares	F-statistic:	209.1
Date:	Tue, 09 May 2023	Prob (F-statistic):	3.55e-159
Time:	18:05:36	Log-Likelihood:	-818.37
No. Observations:	1170	AIC:	1649.
Df Residuals:	1164	BIC:	1679.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	24.3427	10.841	2.245	0.025	3.073	45.613
diagonal	-0.1220	0.047	-2.607	0.009	-0.214	-0.030
height_left	0.1700	0.051	3.339	0.001	0.070	0.270
height_right	0.2753	0.049	5.596	0.000	0.179	0.372
margin_up	0.2536	0.072	3.519	0.000	0.112	0.395
length	-0.4080	0.020	-20.219	0.000	-0.448	-0.368

Omnibus:	61.773	Durbin-Watson:	1.974
Prob(Omnibus):	0.000	Jarque-Bera (JB):	78.246
Skew:	0.506	Prob(JB):	1.02e-17
Kurtosis:	3.763	Cond. No.	1.92e+05

```python
def backward_elimination(data, target,significance_level = 0.05):
    features = data.columns.tolist()
    while(len(features)>0):
        features_with_constant = sm.add_constant(data[features])
        p_values = sm.OLS(target, features_with_constant).fit().pvalues[1:]
        max_p_value = p_values.max()
        if(max_p_value >= significance_level):
            excluded_feature = p_values.idxmax()
            features.remove(excluded_feature)
        else:
            break
    return features
```

✓ 0.0s

```python
backward_elimination(X_train, y_train)
```

✓ 0.0s

```
['diagonal', 'height_left', 'height_right', 'margin_up', 'length']
```

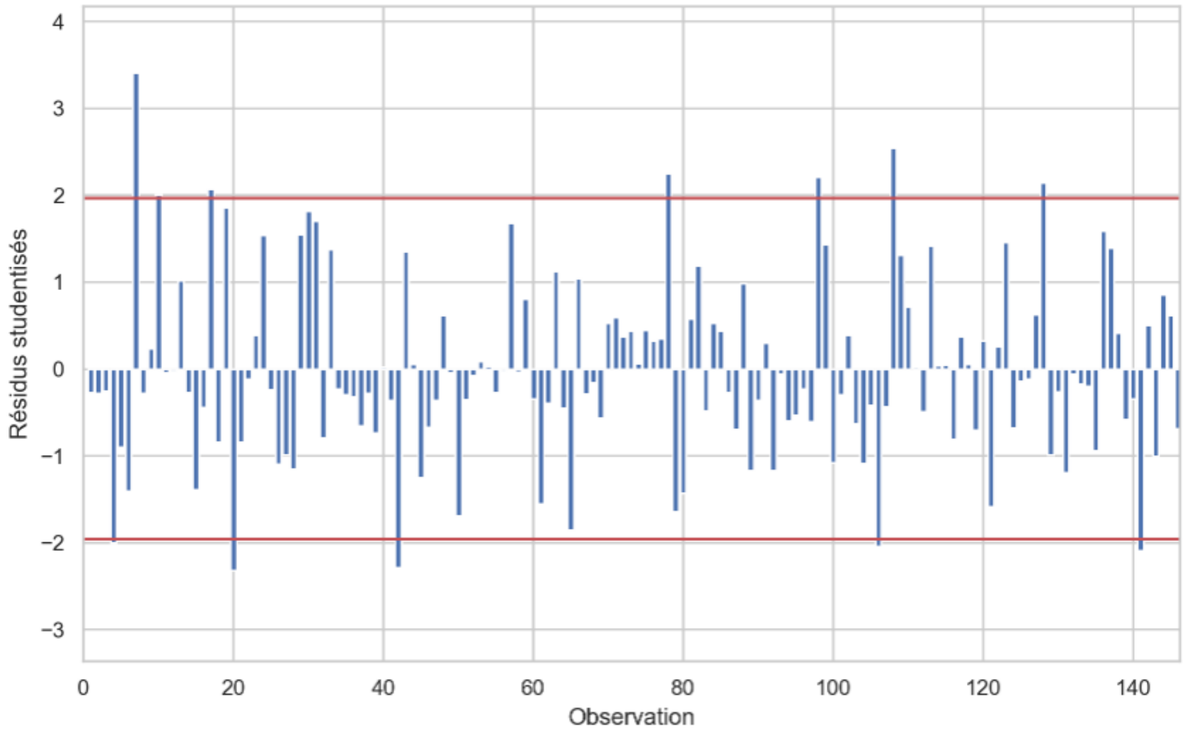
Le code de test indique que toutes les variables ont une influence importante sur la prédiction de la "margin low".

Analyse des résultats

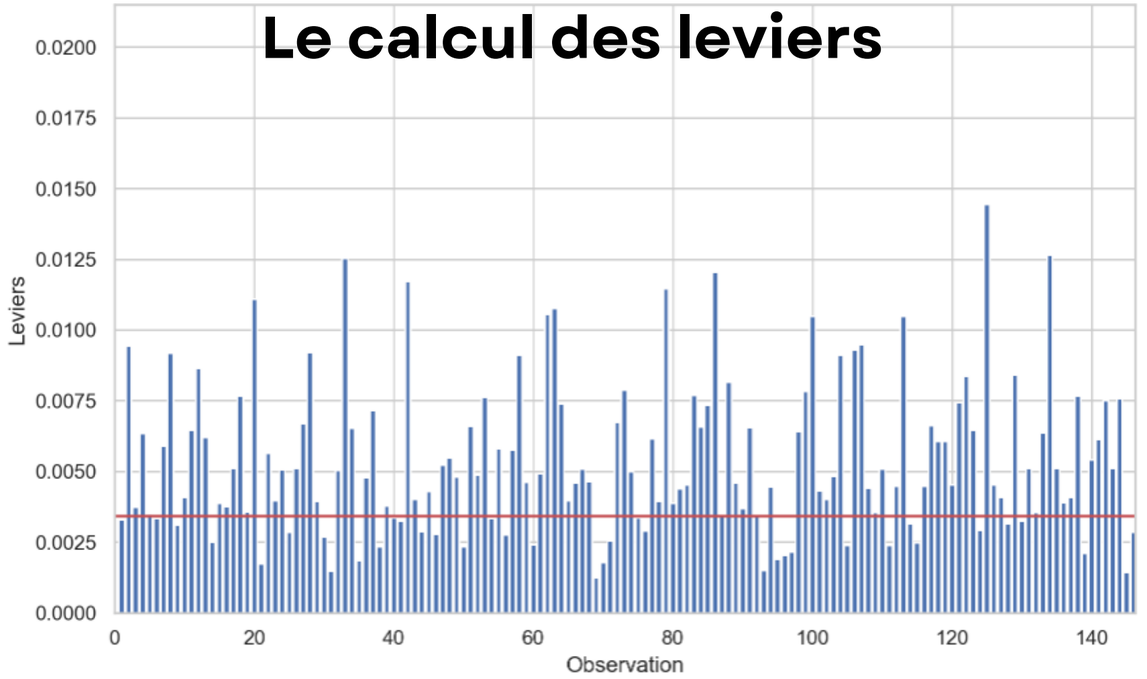
- Outliers
- Linéarité
- Normalité de la distribution des résidus
- Homocédasticité
- Non-autocorrélation
- Multicollinéarité

Outliers

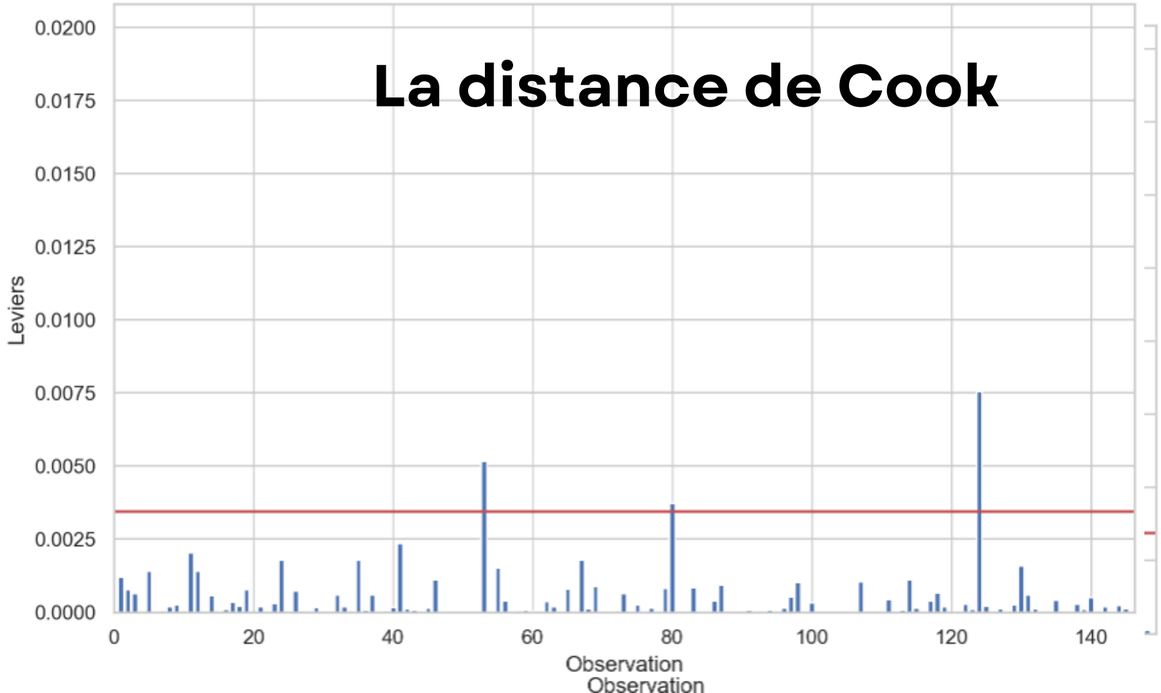
Les résidus studentisés



Le calcul des leviers



La distance de Cook



Linéarité

Rainbow test:

H0 : La relation entre les variables independants et la variable dependente est linéaire.

H1 : H0 est refusé.

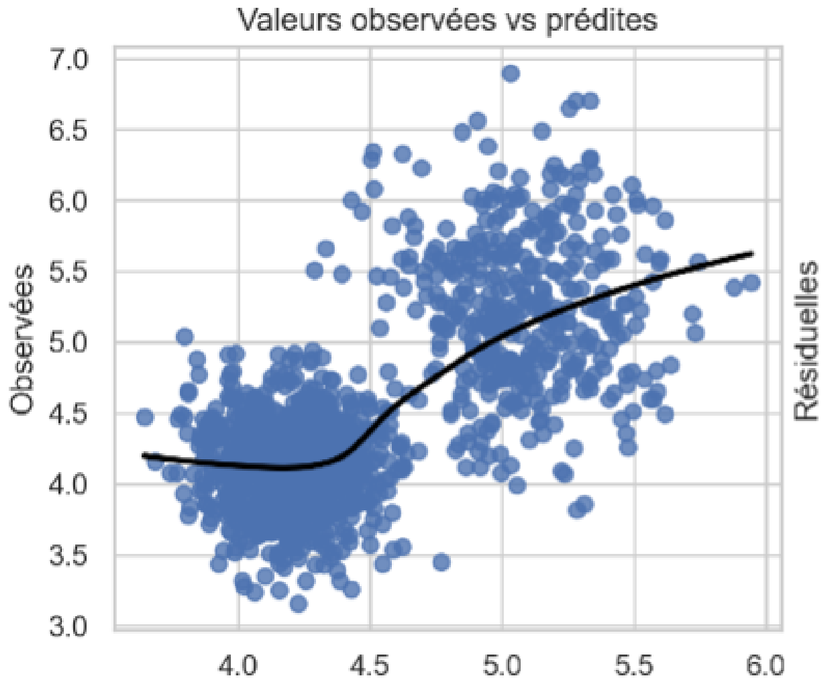
```
# Perform the Rainbow test
rainbow_statistic, p_value = linear_rainbow(reg_multiple_1)

print('Rainbow test statistic:', rainbow_statistic)
print('p-value:', p_value)
```

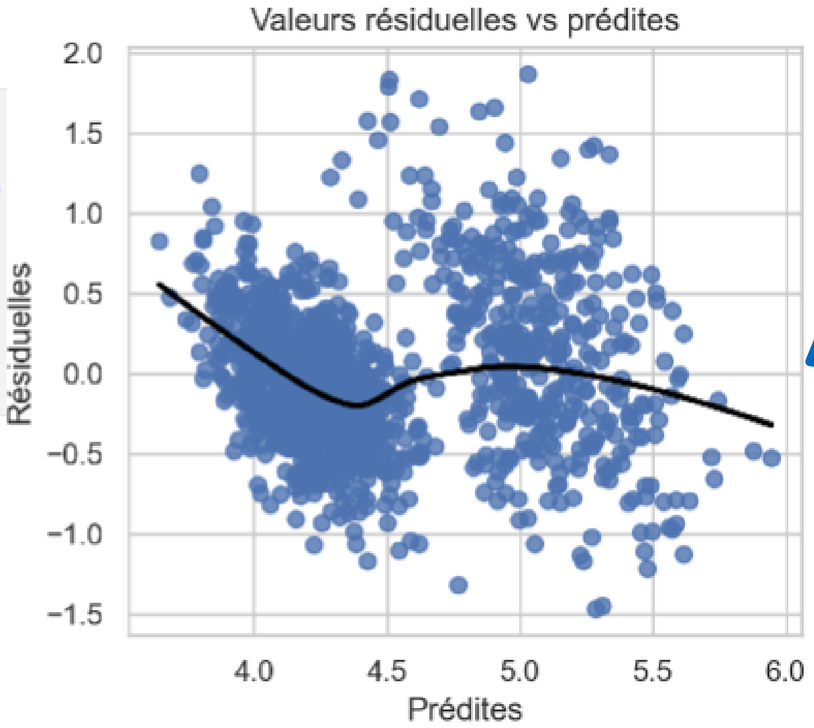
✓ 0.0s

```
Rainbow test statistic: 0.9800291720140607
p-value: 0.596140504173489
```

p-value > 0.05

linéaire





Pas linéaire

Normalité de la distribution des résidus

Omnibus test

H0 : les résidus sont distribués selon une loi Normale.

H1 : Refus de H0

La distribution des résidus n'est pas normale.

OLS Regression Results

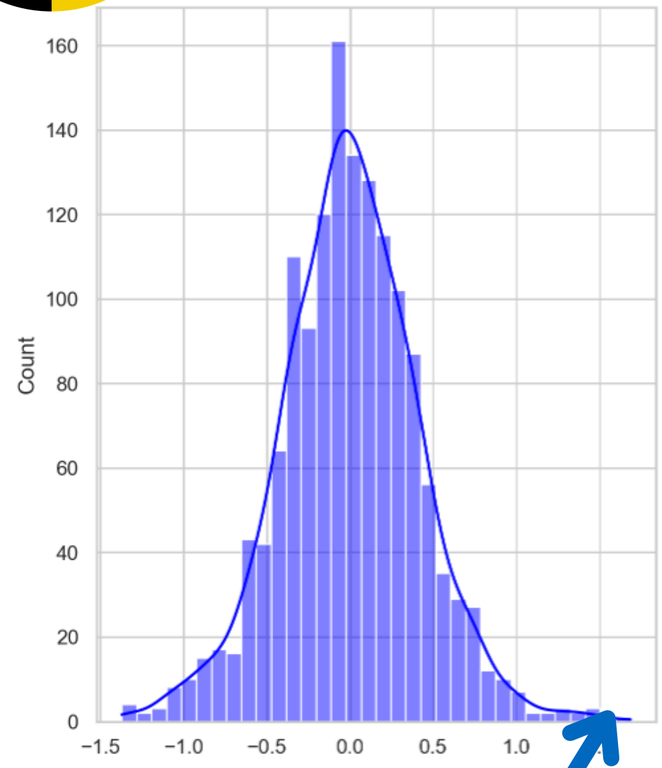
Dep. Variable:	margin_low	R-squared:	0.473
Model:	OLS	Adj. R-squared:	0.471
Method:	Least Squares	F-statistic:	209.1
Date:	Tue, 09 May 2023	Prob (F-statistic):	3.55e-159
Time:	18:05:36	Log-Likelihood:	-818.37
No. Observations:	1170	AIC:	1649.
Df Residuals:	1164	BIC:	1679.
Df Model:	5		
Covariance Type:	nonrobust		

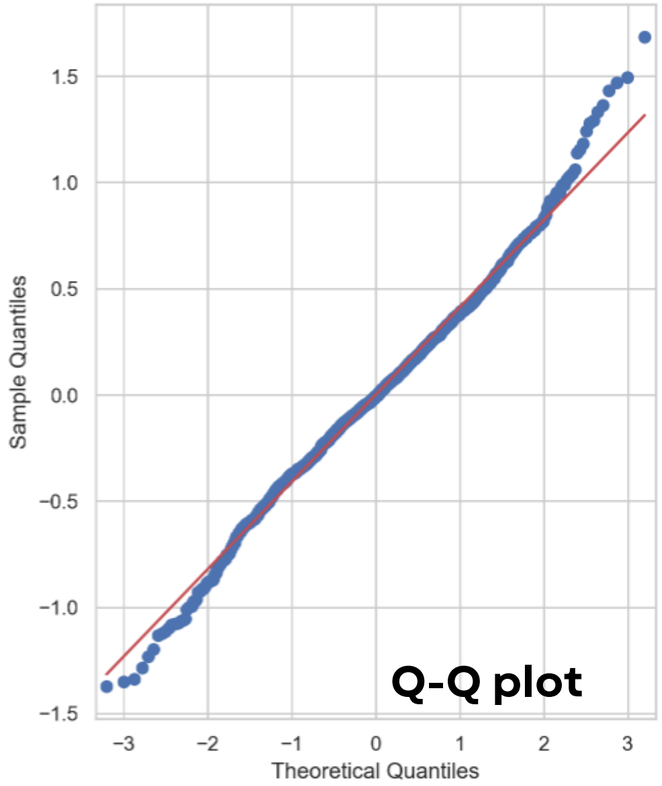
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	24.3427	10.841	2.245	0.025	3.073	45.613
diagonal	-0.1220	0.047	-2.607	0.009	-0.214	-0.030
height_left	0.1700	0.051	3.339	0.001	0.070	0.270
height_right	0.2753	0.049	5.596	0.000	0.179	0.372
margin_up	0.2536	0.072	3.519	0.000	0.112	0.395
length	-0.4080	0.020	-20.219	0.000	-0.448	-0.368

Omnibus:	61.773	Durbin-Watson:	1.974
Prob(Omnibus):	0.000	Jarque-Bera (JB):	78.246
Skew:	0.506	Prob(JB):	1.02e-17
Kurtosis:	3.763	Cond. No.	1.92e+05

(La valeur de *Prob(Omnibus)* doit être proche de 1)

Normalité de la distribution des résidus





Q-Q plot

La distribution des résidus semble être normale

Mais le test de normalité échoue.

```
shapiro_test = stats.shapiro(residus)
shapiro_test
```

✓ 0.0s

```
ShapiroResult(statistic=0.9837262630462646, pvalue=3.642248169732909e-10)
```

Homocédasticité

Pour verifier si la variance de residus est constante

```
# Homoscedasticite:
white_test = het_white(reg_multiple_1.resid, reg_multiple_1.model.exog )
labels_white_test = ["Test Statistic", "p-value", "F-Statistic", "F-Test p-value"]
print(dict(zip(labels_white_test, white_test)))

# Graphique:
plt.subplots(figsize=(8, 4))
plt.scatter(x=reg_multiple_1.resid.index, y=reg_multiple_1.resid, alpha=0.8)
plt.plot(np.repeat(0, len(reg_multiple_1.resid.index)+2), color="black", linestyle='--')
plt.title("Homoscédasticité")
```

✓ 0.3s

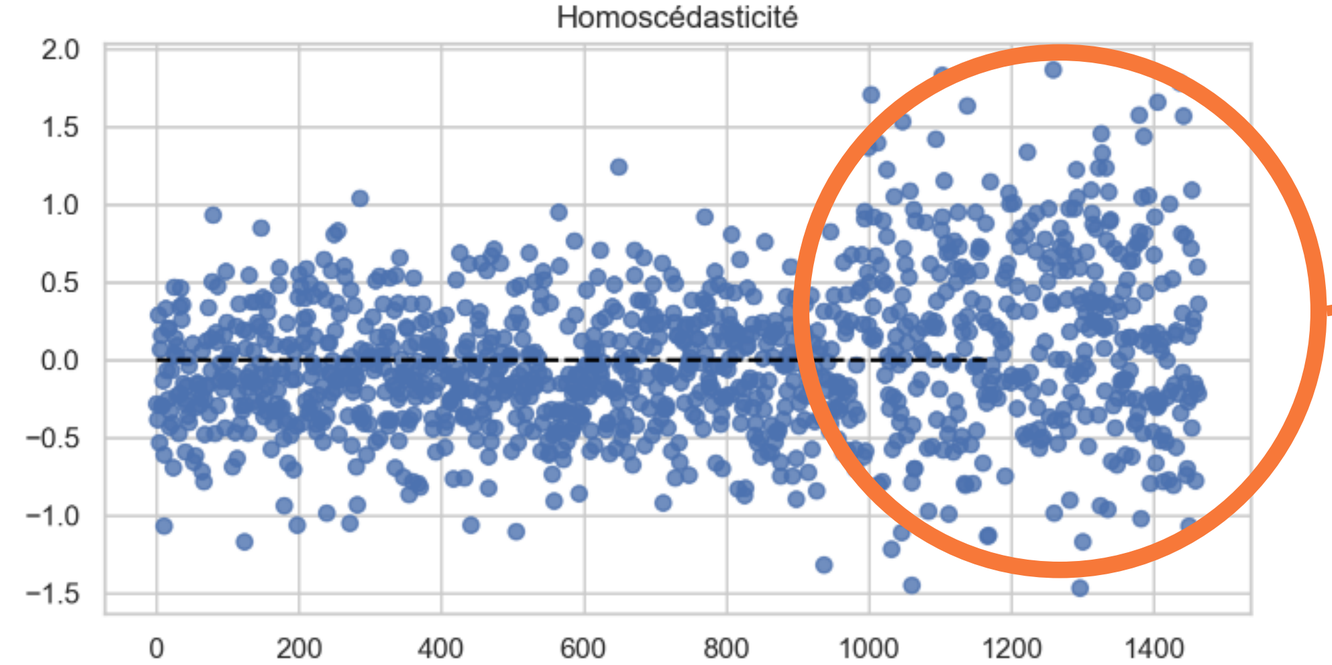
White Test:

H0 : Il y a homoscédasticité

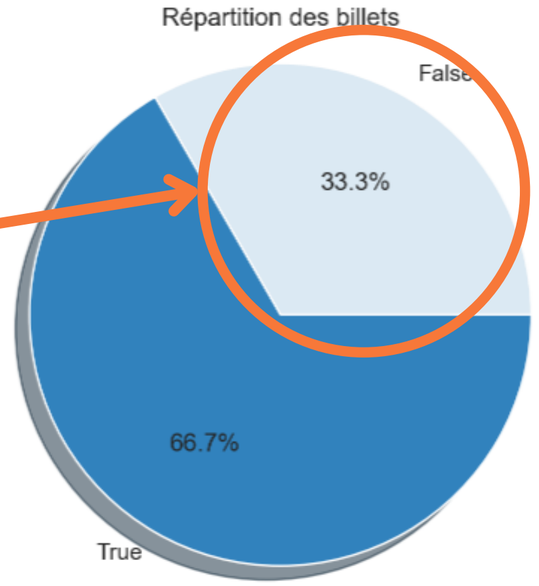
H1 : Il y a hétéroscédasticité

La P-valeur < 0.05 (5%)

```
{'Test Statistic': 98.54382081417977, 'p-value': 2.2931853488004334e-12, 'F-Statistic': 5.283783523537637, 'F-Test p-value': 5.796846582244855e-13}

Text(0.5, 1.0, 'Homoscédasticité')
```



is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length
0	500	500	500	500	500	500
1	1000	1000	1000	1000	1000	1000



Non-autocorrélation

La corrélation des erreurs (résidus) dans le temps

Valeurs DW proche de 2
= Pas d'autocorrélation

OLS Regression Results

Dep. Variable:	margin_low	R-squared:	0.473
Model:	OLS	Adj. R-squared:	0.471
Method:	Least Squares	F-statistic:	209.1
Date:	Tue, 09 May 2023	Prob (F-statistic):	3.55e-159
Time:	18:05:36	Log-Likelihood:	-818.37
No. Observations:	1170	AIC:	1649.
Df Residuals:	1164	BIC:	1679.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	24.3427	10.841	2.245	0.025	3.073	45.613
diagonal	-0.1220	0.047	-2.607	0.009	-0.214	-0.030
height_left	0.1700	0.051	3.339	0.001	0.070	0.270
height_right	0.2753	0.049	5.596	0.000	0.179	0.372
margin_up	0.2536	0.072	3.519	0.000	0.112	0.395
length	-0.4080	0.020	-20.219	0.000	-0.448	-0.368

Omnibus:	61.773	Durbin-Watson:	1.974
Prob(Omnibus):	0.000	Jarque-Bera (JB):	78.246
Skew:	0.506	Prob(JB):	1.02e-17
Kurtosis:	3.763	Cond. No.	1.92e+05

Multicollinéarité

```
# VIF (variance inflation factor)
variables = reg_multiple_1.model.exog # 要求在reg_multiple裡面直接拿取variable X
[variance_inflation_factor(variables, i) for i in np.arange(1,variables.shape[1])]

# endog = variable y
# exog = variable x
```

✓ 0.0s

```
[1.0149497057950003,
 1.1404387009379209,
 1.2528008354104854,
 1.3787230768857526,
 1.5415658682095223]
```

Durbin-Watson:	1.974
Jarque-Bera (JB):	78.246
Prob(JB):	1.02e-17
Cond. No.	1.92e+05

Cond. No. > 30 : il y **a** peut-être un problème de multicollinéarité.

VIF < 10 : il y a **pas** un problème de multicollinéarité.

Application du modèle

pour prédire les valeurs nulles dans margin_low

	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length
72	1	171.94	103.89	103.45	NaN	3.25	112.79
99	1	171.93	104.07	104.18	NaN	3.14	113.08
151	1	172.07	103.80	104.38	NaN	3.02	112.93
197	1	171.45	103.66	103.80	NaN	3.62	113.27
241	1	171.83	104.14	104.06	NaN	3.02	112.36

```
# Combler les valeurs dans les valeur manquant de margin low
tmp['margin_low'] = y_null
```

	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length
72	1	171.94	103.89	103.45	4.316697	3.25	112.79
99	1	171.93	104.07	104.18	4.403277	3.14	113.08
151	1	172.07	103.80	104.38	4.426124	3.02	112.93
197	1	171.45	103.66	103.80	4.331748	3.62	113.27
241	1	171.83	104.14	104.06	4.657649	3.02	112.36

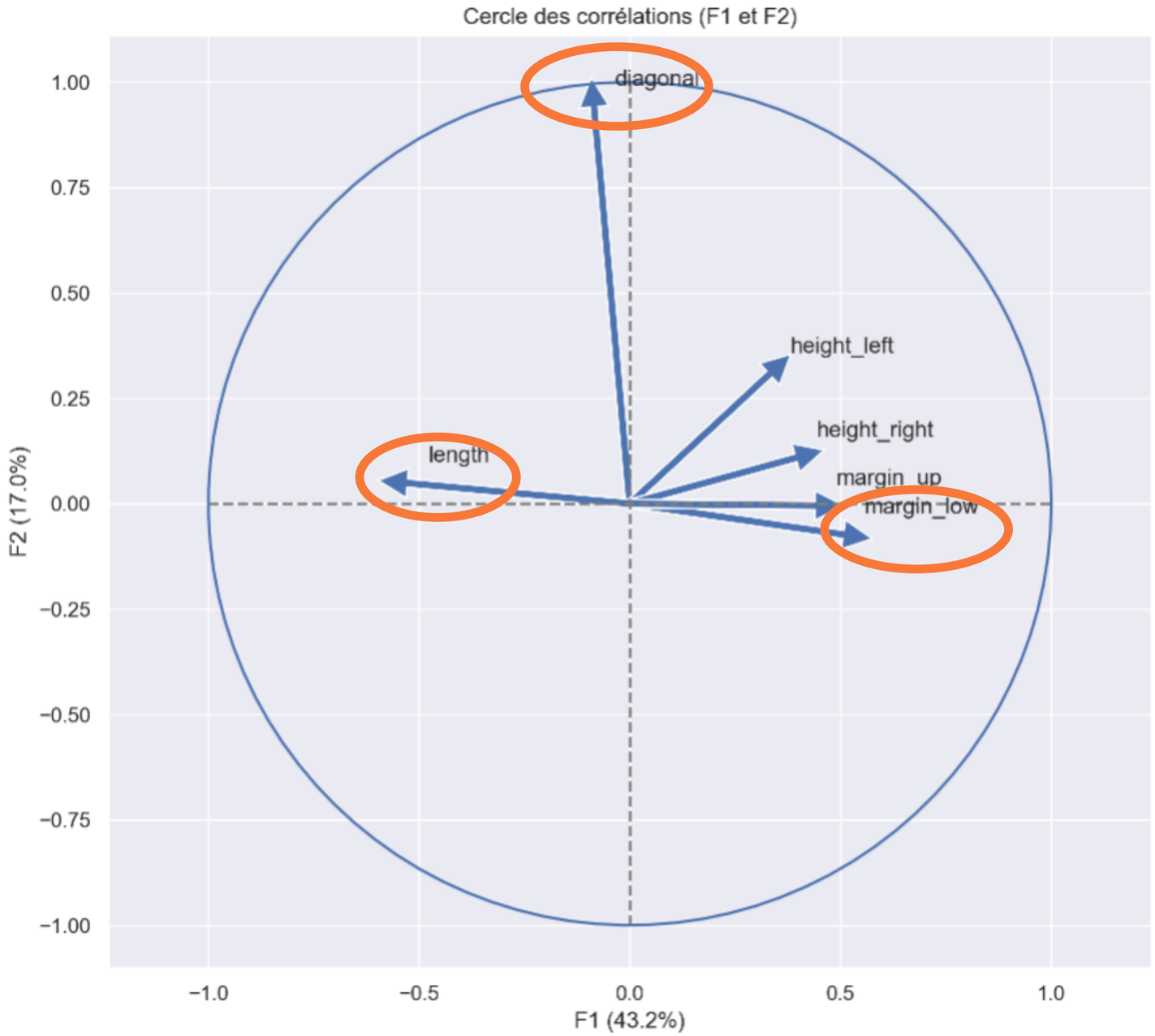
```
# Billets moyens selon l'authenticité:
print(df_new.groupby(["is_genuine"]).mean())
```

✓ 0.0s

```
              diagonal  height_left  height_right  margin_low  margin_up
is_genuine
0            171.90116    104.19034     104.14362    5.214151    3.35016
1            171.98708    103.94913     103.80865    4.118341    3.05213

                length
is_genuine
0            111.63064
1            113.20243
```

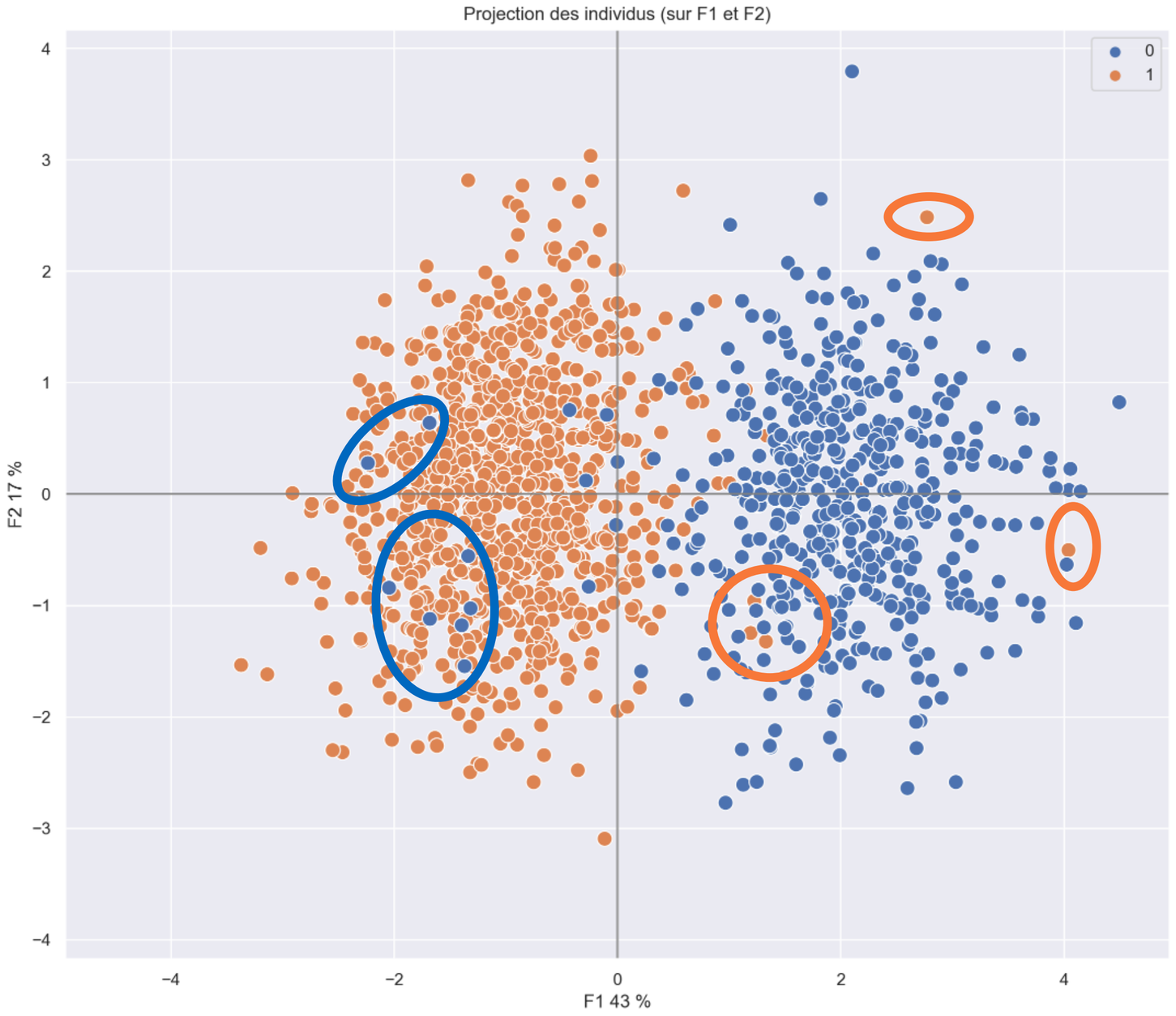
Analyse des composants principaux (ACP)

Les relations entre les variables



La variabilité des individus

- **0 : Faux billets**
- **1 : Vrai billets**



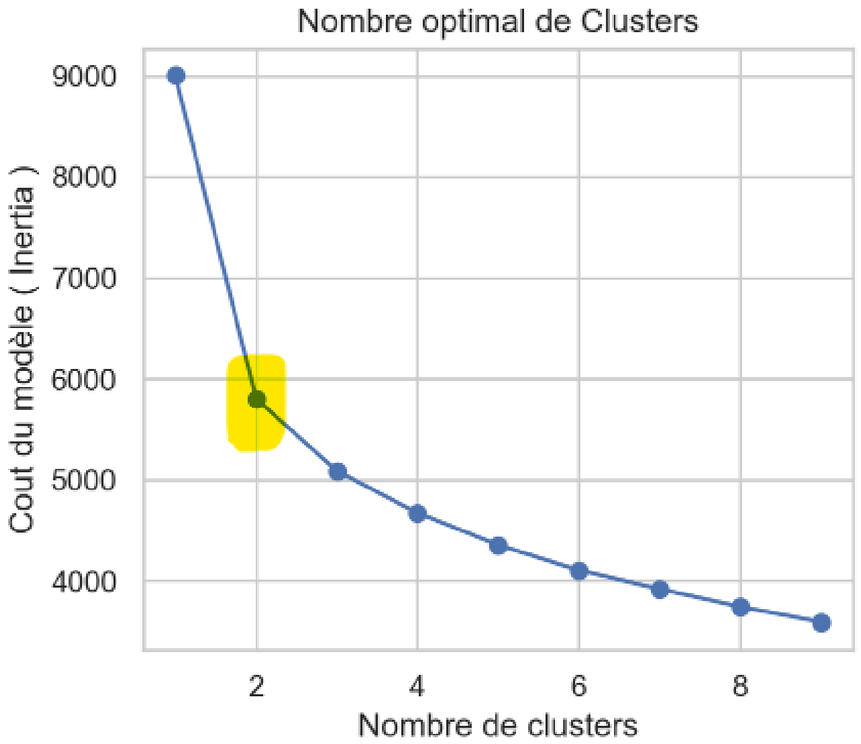
Modèles de prediction

- K-MEANS
- K-NN
- Régression logistique

Algorithme k-means

Non-supervisé

On utilise 2 clusters .




```
model_km = KMeans(n_clusters=2)
model_km.fit(X_train)
tmp_predicted_groups = model_km.predict(X_train)
```

```
tmp_comparison_df = pd.DataFrame()
tmp_comparison_df["actual"] = y_train
tmp_comparison_df["predicted_group"] = tmp_predicted_groups
tmp_comparison_df.head()
```

✓ 0.1s

	actual	predicted_group
36	1	0
680	1	0
1302	0	1
532	1	0
767	1	0

0 : Faux billets

1 : Vrai billets

```
# inverser les valeur tel que vrai billet=1 et faux billet=0
predicted_groups_km = 1-tmp_predicted_groups
```

✓ 0.2s

Mesures pour évaluer la performance du modèle de K-means

La somme des carrés intra-cluster (inertia)

```
print('Inertie totale :',model_km.inertia_)
```

✓ 0.2s

```
Inertie totale : 4634.708767789602
```

Difficile de juger si cette valeur est bonne ou mauvaise

Le coefficient de silhouette

```
silhouette_score(X_train,predicted_groups_km)
```

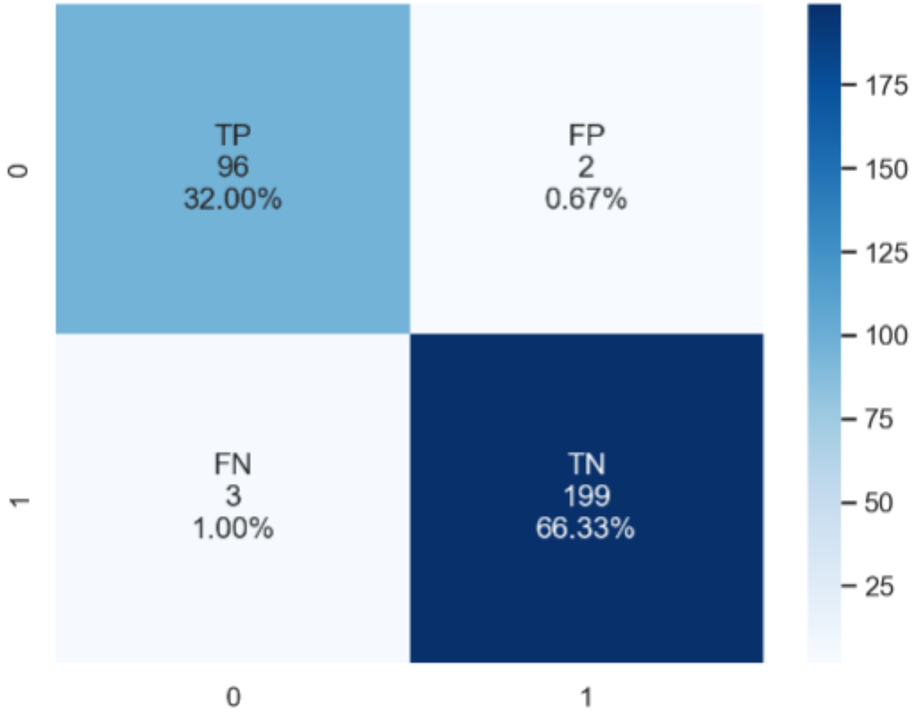
✓ 0.1s

```
0.3436995904916229
```

La valeur du coefficient de silhouette est entre -1 à 1:

- proche de 1 : la classification est dense et bien séparée
- proche de 0 : la classification qui pourrait être améliorée
- proche de -1 : une classification incorrecte

Matrice de confusion (k-means)



On vérifie la prédiction avec testing set

TP = True positive

FP = False positve

FN = False negative

TN = True negative

Sensibilité & Spécificité

```
tp = confu_matrix_test_km[0,0]
fp = confu_matrix_test_km[0,1]
fn = confu_matrix_test_km[1,0]
tn = confu_matrix_test_km[1,1]

sensibility = tp/(tp+fn)
specificity = tn/(tn+fp)

print("sensibility:", sensibility)
print("specificity:", specificity)
```

✓ 0.0s

```
sensibility: 0.9696969696969697
specificity: 0.9900497512437811
```

K-NN (k Nearest Neighbors)

Supervisé

Nombre de voisin optimal

```
knn_model = KNeighborsClassifier(n_neighbors=2,p=2,weights='distance',algorithm='auto')
knn_model.fit(X_train, y_train)  # n neightbors最好是基數(K值為基數最佳),我們加入weight加權
```

✓ 0.2s

```
KNeighborsClassifier(n_neighbors=2, weights='distance')
```

```
knn_model.score(X_test,y_test)
# 1表示係數吻合 perfect fitting
```

✓ 0.1s

```
0.9833333333333333
```

Accuracy avec 2 K

```
knn_model = KNeighborsClassifier(n_neighbors=5,p=2,weights='distance',algorithm='auto')
knn_model.fit(X_train, y_train)
```

✓ 0.0s

```
KNeighborsClassifier(weights='distance')
```

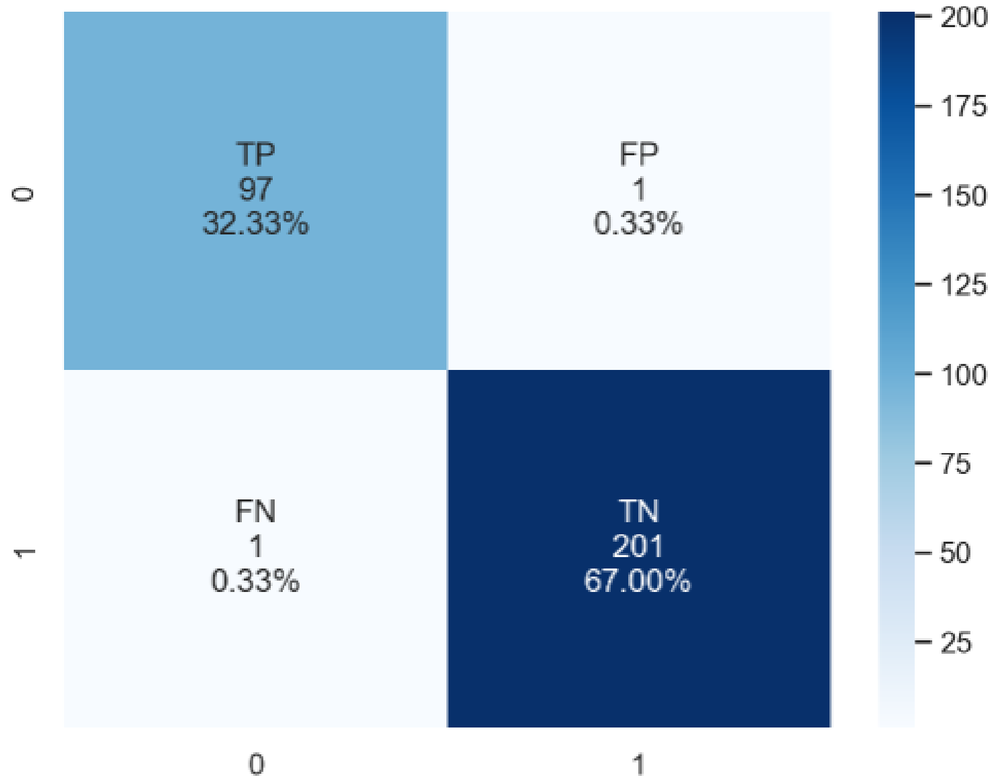
```
knn_model.score(X_test,y_test)
# 1表示係數吻合 perfect fitting
```

✓ 0.2s

```
0.9933333333333333
```

Accuracy avec 5 K

Matrice de confusion (K-NN)



On vérifie la prédiction avec testing set

Sensibilité & Spécificité

```
# sensibility (recall | rappel)
sensibility = recall_score(y_test, y_pred)
print('sensibility :', sensibility)
```

✓ 0.1s

```
sensibility : 0.995049504950495
```

```
# specificity
tp = confu_matrix_knn[0,0]
fp = confu_matrix_knn[0,1]
fn = confu_matrix_knn[1,0]
tn = confu_matrix_knn[1,1]

specificity = tn/(tn+fp)

print("specificity:", specificity)
```

✓ 0.1s

```
specificity: 0.995049504950495
```

Régression logistique

Supervisé

Generalized Linear Model Regression Results

Dep. Variable:	is_genuine	No. Observations:	1500
Model:	GLM	Df Residuals:	1493
Model Family:	Binomial	Df Model:	6
Link Function:	Logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-39.487
Date:	Mon, 08 May 2023	Deviance:	78.975
Time:	11:27:37	Pearson chi2:	2.80e+03
No. Iterations:	10	Pseudo R-squ. (CS):	0.7049
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	2.1988	0.376	5.842	0.000	1.461	2.937
diagonal	0.0818	0.347	0.236	0.814	-0.598	0.762
height_left	-0.3809	0.335	-1.138	0.255	-1.037	0.275
height_right	-0.9499	0.374	-2.538	0.011	-1.683	-0.216
margin_low	-3.9709	0.651	-6.101	0.000	-5.247	-2.695
margin_up	-2.3869	0.511	-4.668	0.000	-3.389	-1.385
length	5.2840	0.783	6.751	0.000	3.750	6.818

Generalized Linear Model Regression Results

Dep. Variable:	is_genuine	No. Observations:	1500
Model:	GLM	Df Residuals:	1495
Model Family:	Binomial	Df Model:	4
Link Function:	Logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-40.175
Date:	Mon, 08 May 2023	Deviance:	80.351
Time:	11:27:37	Pearson chi2:	3.42e+03
No. Iterations:	10	Pseudo R-squ. (CS):	0.7046
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	2.2088	0.373	5.920	0.000	1.478	2.940
height_right	-1.0904	0.365	-2.983	0.003	-1.807	-0.374
margin_low	-4.1595	0.635	-6.550	0.000	-5.404	-2.915
margin_up	-2.4117	0.509	-4.737	0.000	-3.410	-1.414
length	5.3737	0.775	6.932	0.000	3.854	6.893

P < 0,05 = la variable indépendante a un impact significatif sur la variable dépendante (is_genuine).

Les variables utilisées pour notre modèle :
height_right, margin_low, margin_up, length

Modélisation finale de regression logistique

```
# On entraine notre modèle
model_reglog_final = linear_model.LogisticRegression()
model_reglog_final.fit(X_train2,y_train2)
```

✓ 0.1s

```
LogisticRegression()
```

On compare les données de l'ensemble de test avec les données prédites par le modèle

```
# actual données
actual_y = y_test2 # predicted y的數據shape必須和predictor x 的大小一樣才能做
# prédiction sur les données de test.
predicted_y = model_reglog_final.predict(X_test2)
confu_matrix_reglog_final = confusion_matrix(actual_y,predicted_y)
```

```
predicted = model_reglog_final.predict(X_test2)

df_compare = pd.DataFrame()
df_compare["actual_value"] = actual_y
df_compare["predicted"] = predicted_y
```

```
df_compare
```

✓ 0.2s

	actual_value	predicted
0	1	1
1	1	1
2	0	0
3	1	1
4	0	0
...	...	...
295	0	0
296	0	0
297	0	0
298	1	1
299	1	1

300 rows × 2 columns

Mesures pour évaluer la performance du modèle de regression logistique

Accuracy (exactitude)

```
# Le taux de précision des résultats de test
model_reglog_final.score(X_test2, y_test2)
```

✓ 0.2s

```
0.9966666666666667
```

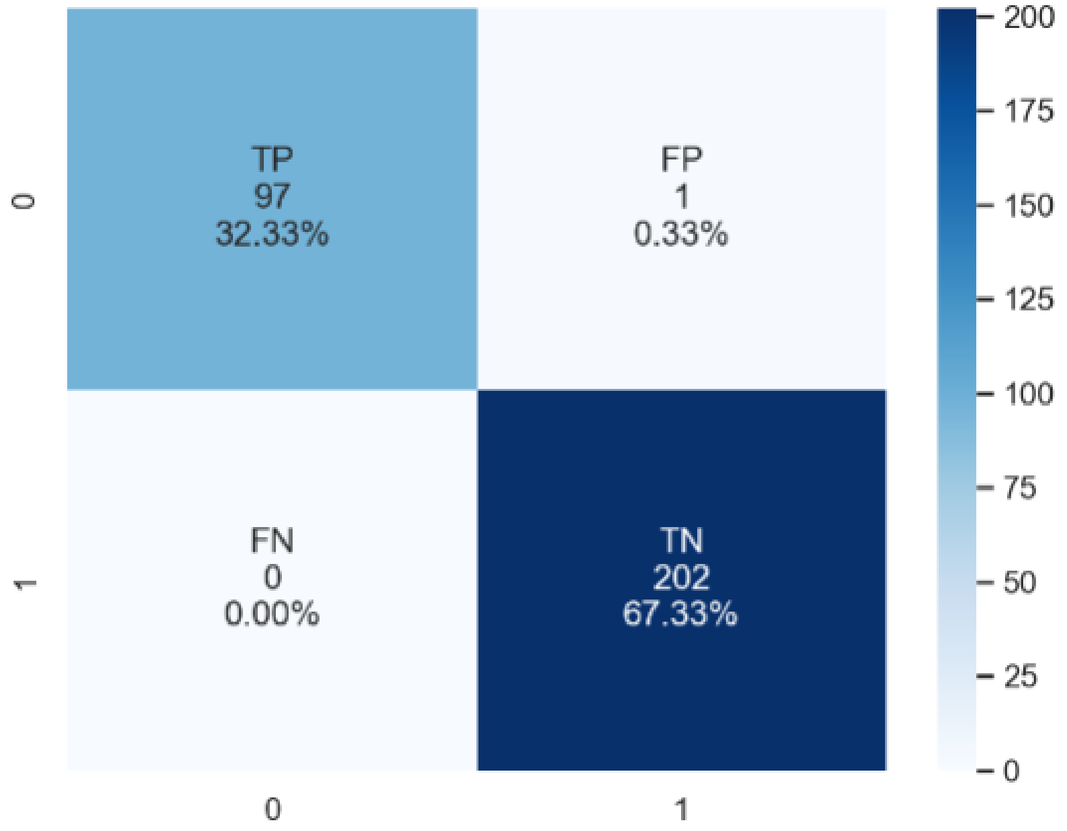
Sensibilité & Spécificité

```
sensibility = tp/(tp+fn)
specificity = tn/(tn+fp)

print("sensibility:", sensibility)
print("specificity:", specificity)
```

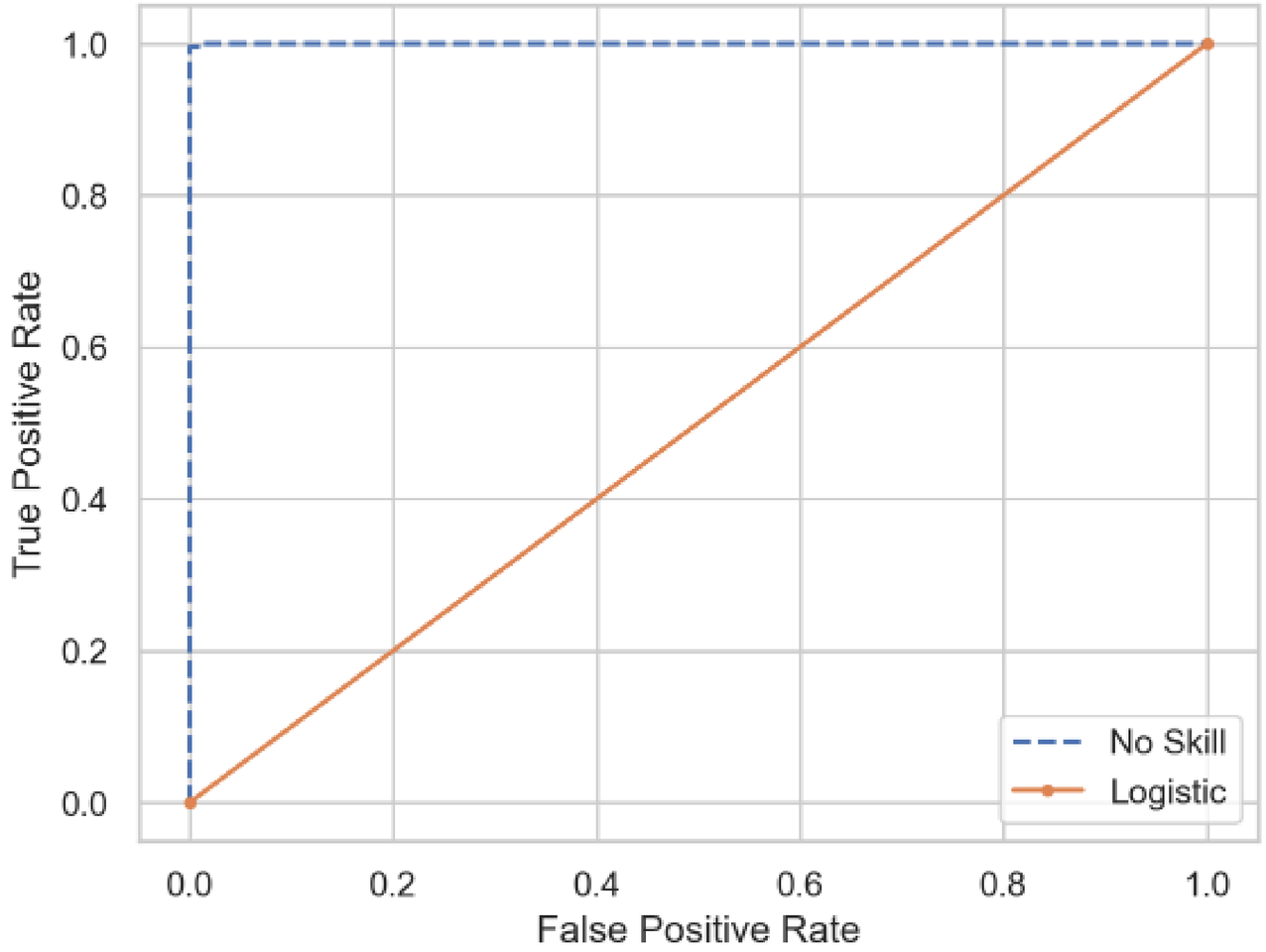
✓ 0.0s

```
sensibility: 1.0
specificity: 0.9950738916256158
```

Matrice de confusion



Mesures pour évaluer la performance du modèle de regression logistique



L'aire sous la courbe est quasiment égale à 1 :

- modèle presque parfait

La courbe ROC (Receiver Operating Characteristic)

Mesures pour évaluer la performance du modèle de regression logistique

Validation croisée (cross-validation)

```
# Créer le nouveau modèle de régression logistique
lr = LogisticRegression()

# Validation croisée avec 5 folds
scores = cross_val_score(lr, X_reglog_final_scaled, y, cv=5)

# Scores moyens de la validation croisée
print("Accuracy: %0.3f (+/- %0.3f)" % (scores.mean(), scores.std() * 2))
```

✓ 0.1s

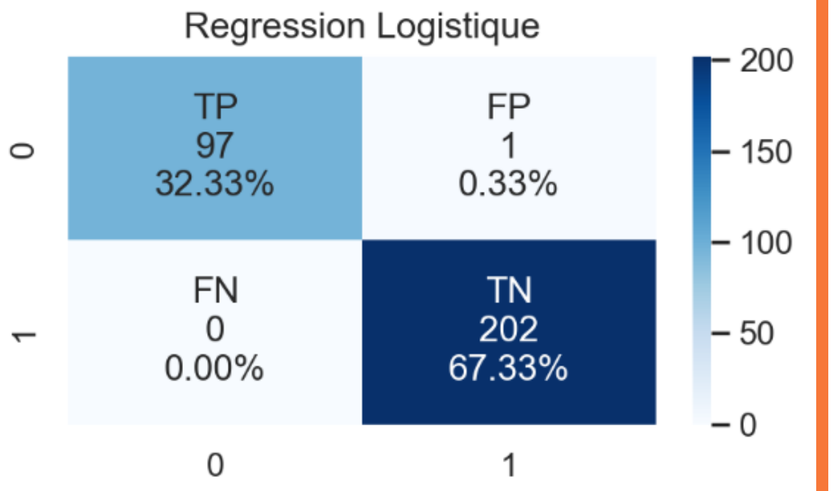
```
Accuracy: 0.991 (+/- 0.011)
```

Validation croisée avec 5 plies

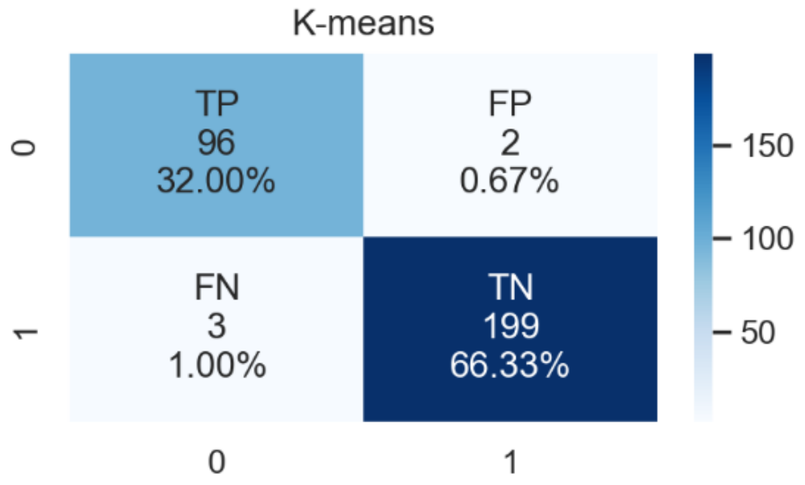
Accuracy: 0.991 = Une performance élevée avec une précision moyenne de 99,1 %.
(+/- 0.011) : Une plage de variation de +/- 0,011.

Conclusion

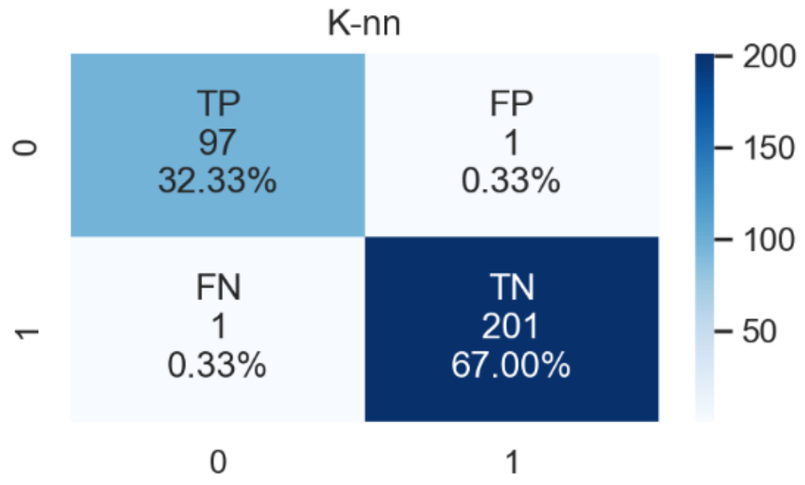
Comparaison des matrices de confusions pour les trois modèles vus



sensibility: 1.0
specificity: 0.9950738916256158



sensibility: 0.9696969696969697
specificity: 0.9900497512437811



sensibility : 0.995049504950495
specificity: 0.995049504950495

Test soutenance

Merci pour votre attention