Détection de faux billets

Descriptif de la mission

Analyse des données

Tour d’horizon

Visualisation des variables

Imputation des valeurs manquantes

Option 1 - Eliminer les valeurs manquantes

Option 2 - Régression linéaire simple

Option 3 - Régression linéaire multiple

Modélisation

Modèle 1 - Dummy Classifier

Modèle 2 - Régression logistique

Modèle 3 - K-means

Modèle 4 - K-Nearest Neighbors (KNN)

Conclusion

Comparaison des modèles

Remarques et pistes d’amélioration

Descriptif de la mission

Le client

L’organisation nationale de lutte contre le faux-monnayage (ONFCM) souhaite mettre en place des méthodes d’identification des contrefaçons de billets en euros.

La mission

Concevoir une modélisation qui serait capable d’identifier automatiquement les vrais des faux billets.

Analyse des données

Tour d'horizon des données

Présentation

```
bil.sample(5)
```

	is_genuine	diagonal	height_left	height_right	margin_low	margin_up	length
450	True	172.01	104.11	103.13	4.53	2.96	113.45
548	True	172.05	103.75	103.47	4.19	2.98	112.89
568	True	172.05	104.11	104.07	3.60	2.78	112.93
501	True	171.87	103.73	103.69	4.06	3.10	112.96
1143	False	171.90	104.25	104.64	4.46	3.07	110.86

```
bil.shape
```

```
(1500, 7)
```

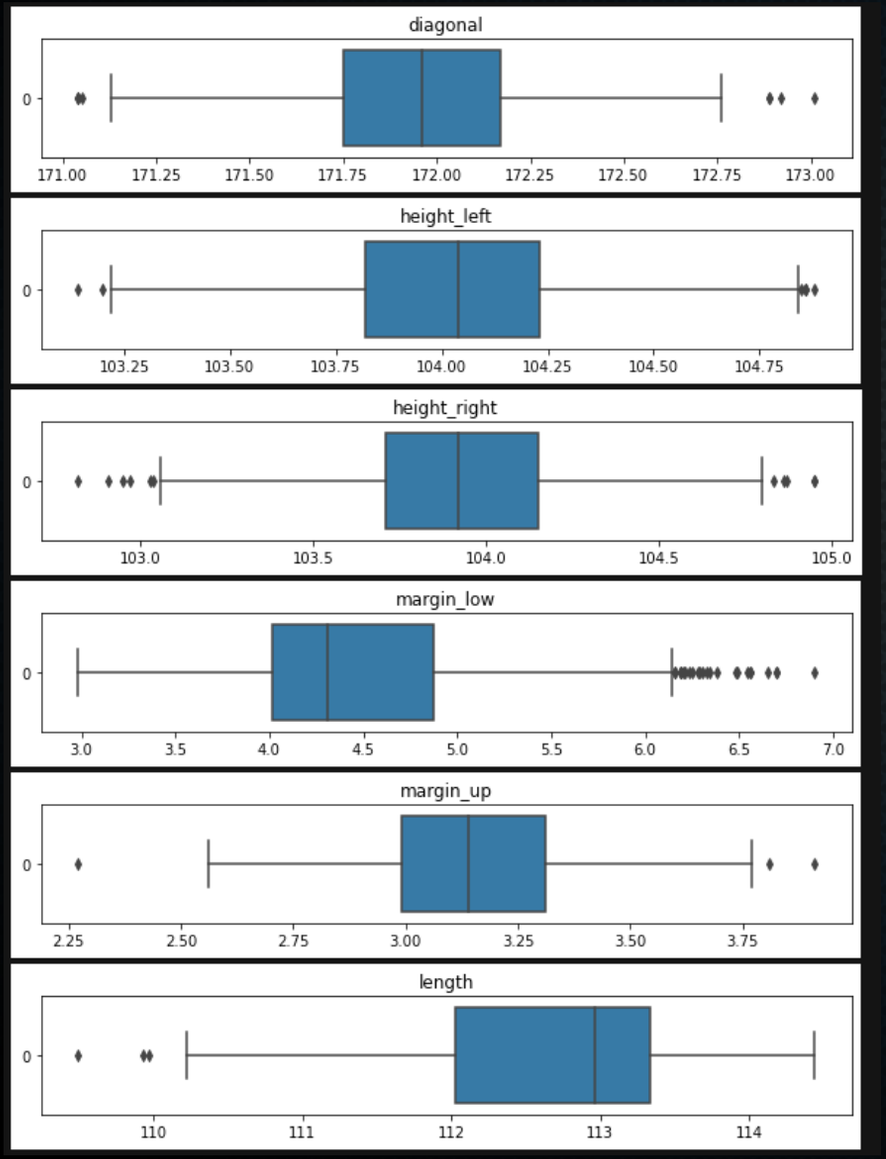
Valeurs manquantes

```
bil.isna().sum()
```

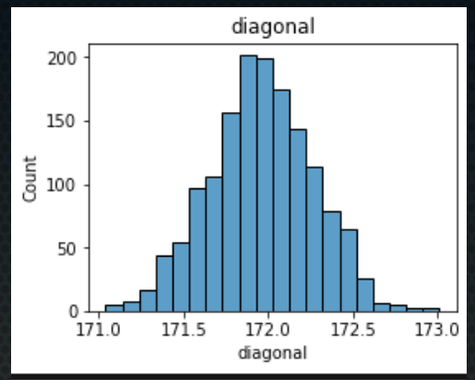
```
is_genuine      0
diagonal        0
height_left     0
height_right    0
margin_low     37
margin_up       0
length          0
dtype: int64
```

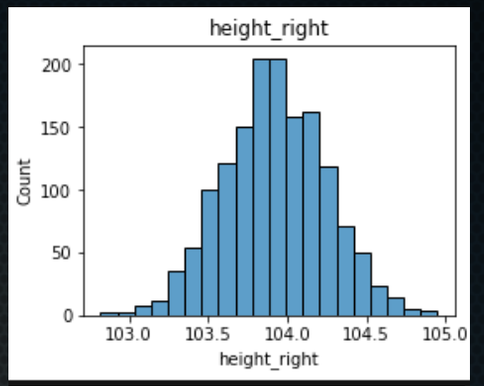
Visualisation des variables

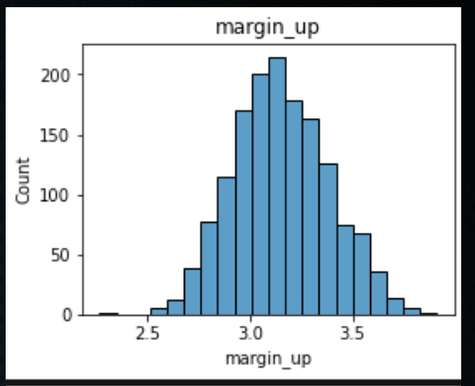
Boxplots

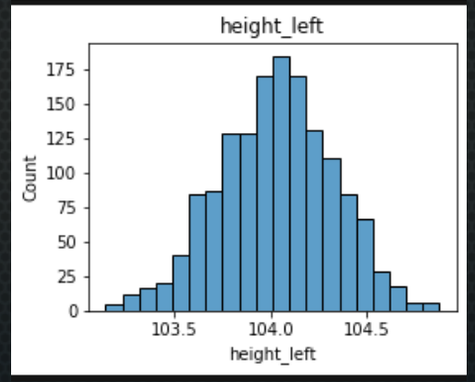


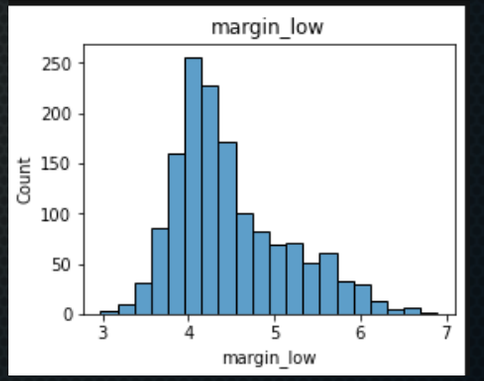
Distribution

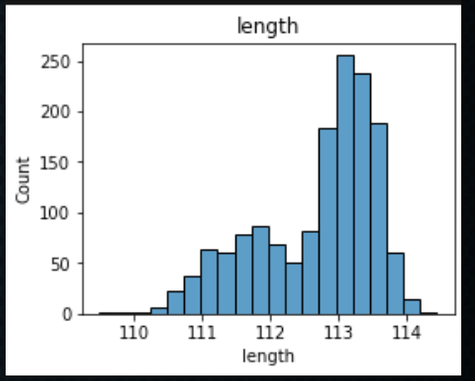






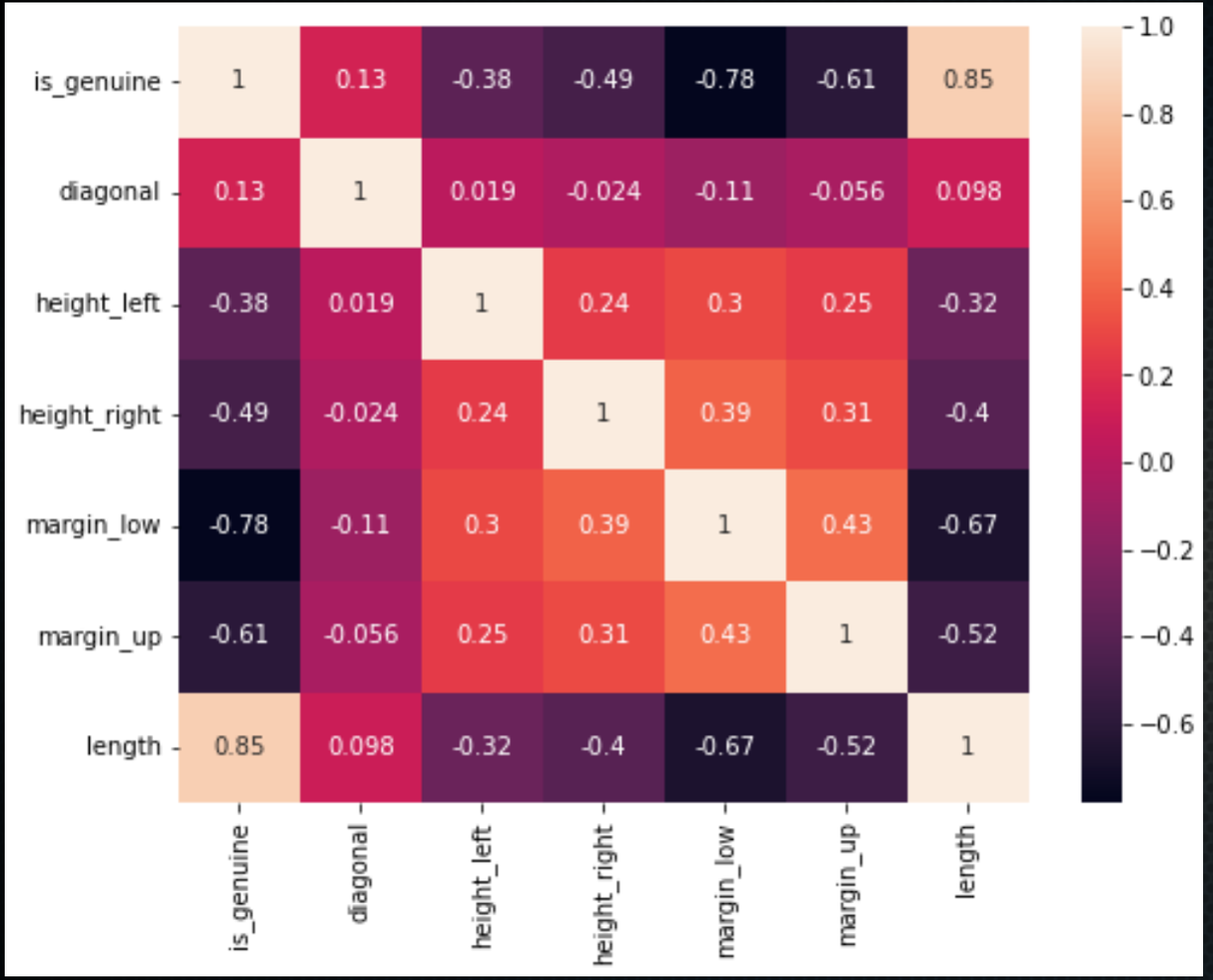






- **Pas de valeurs aberrantes**
- **Les distributions de ‘margin_low’ et de ‘length’ ne sont pas gaussiennes**

Matrice de corrélation

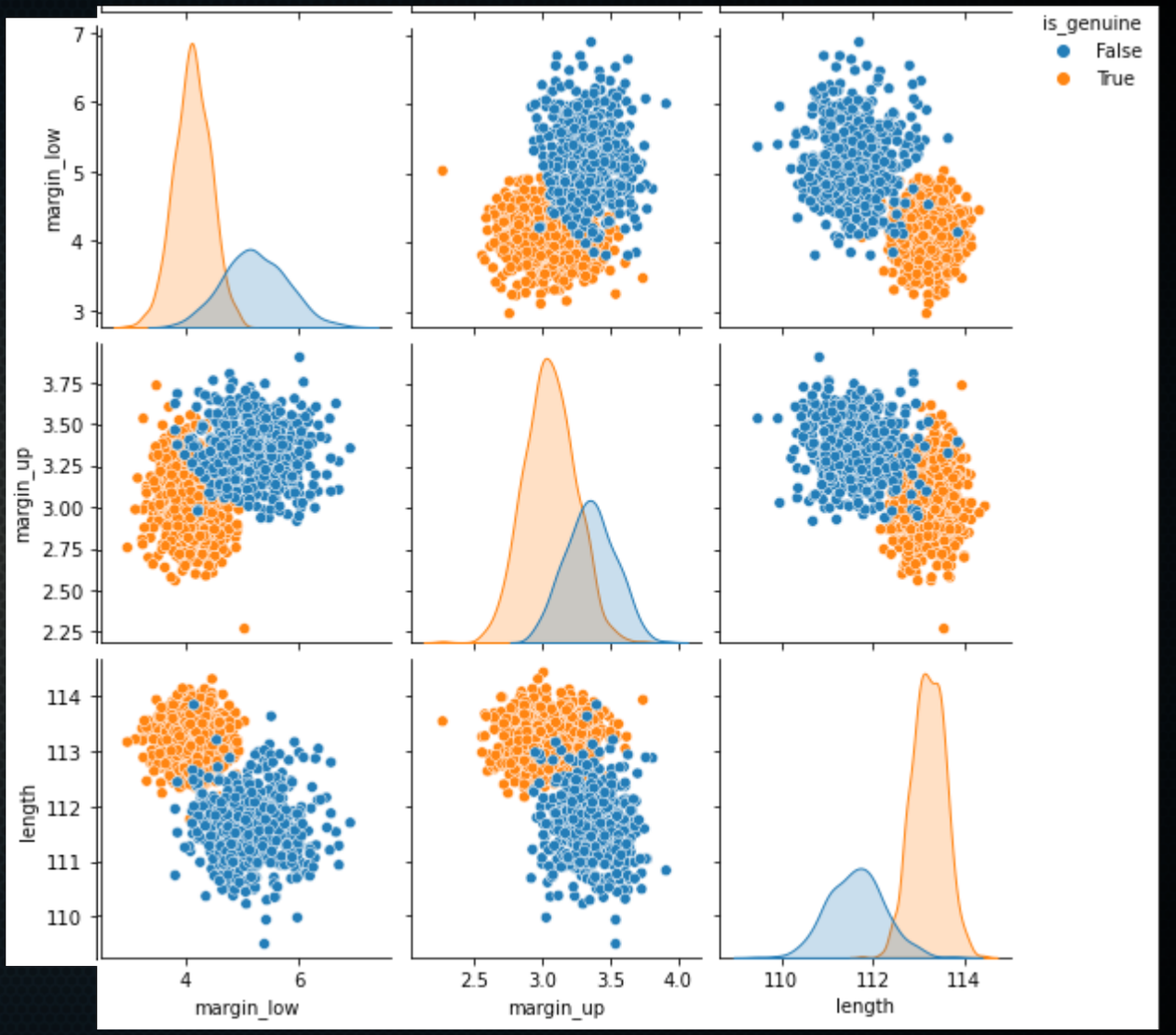


. **length** est la variable la plus corrélée à la variable cible

. **diagonal** est la moins corrélée

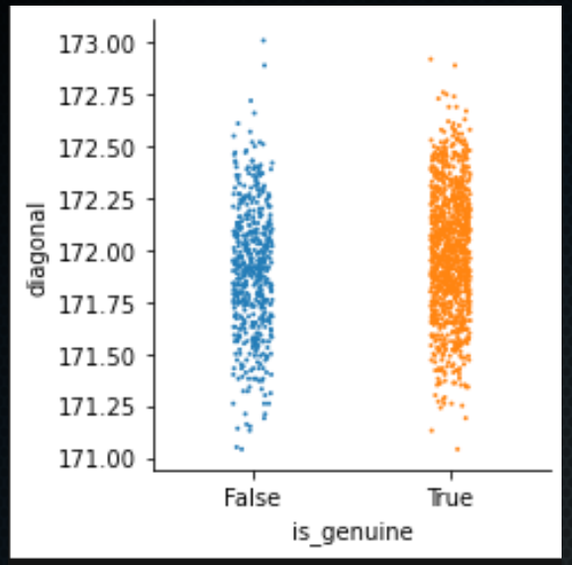
. pas de variables continues corrélées 2 à 2 à un niveau > à 0,8

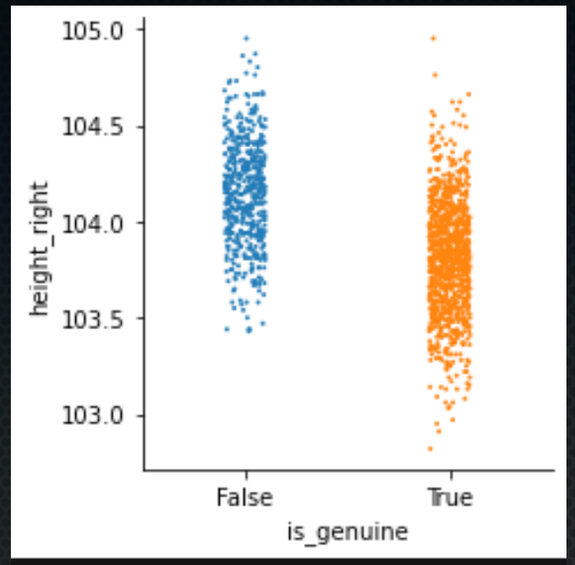
Pairplot (extrait)

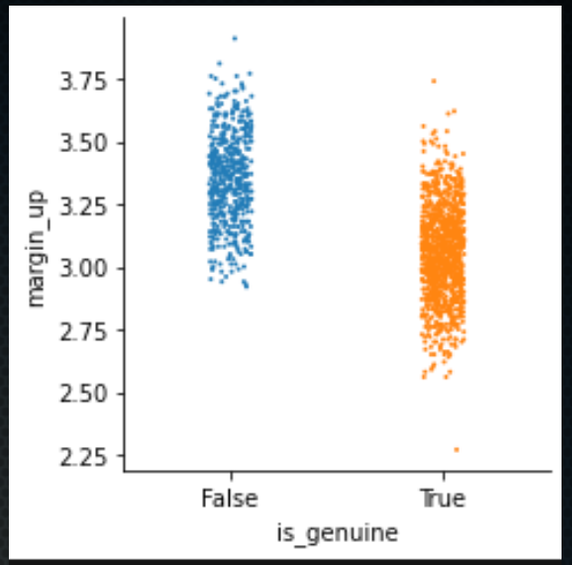


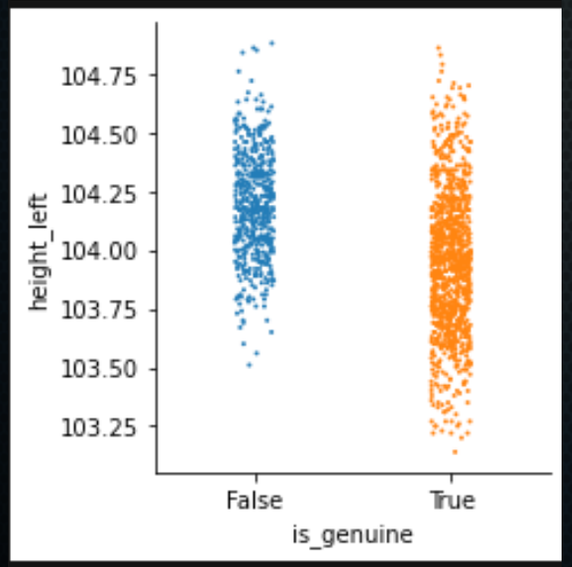
Les billets true et false sont clairement localisés

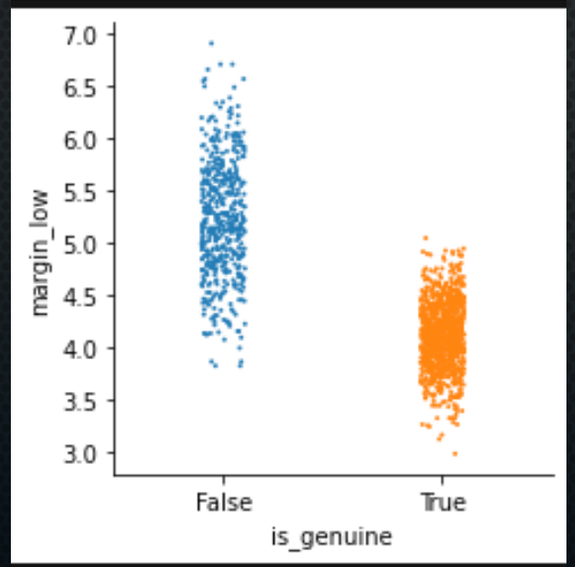
Catplots

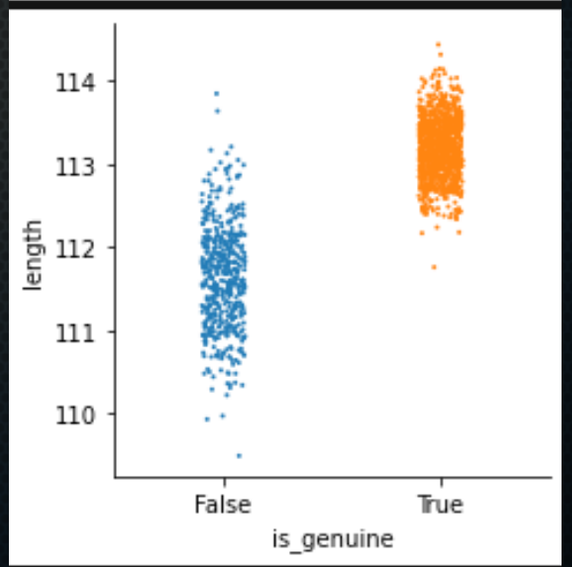












. La variable ‘length’ est la meilleure prédictrice des faux billets

. La variable ‘diagonal’ est la moins bonne

Imputation des valeurs manquantes

Option 1 - Eliminer les valeurs manquantes

Création d'un dataframe nettoyé des valeurs manquantes

```
bil_no_miss = bil.dropna()
bil_no_miss.shape
```

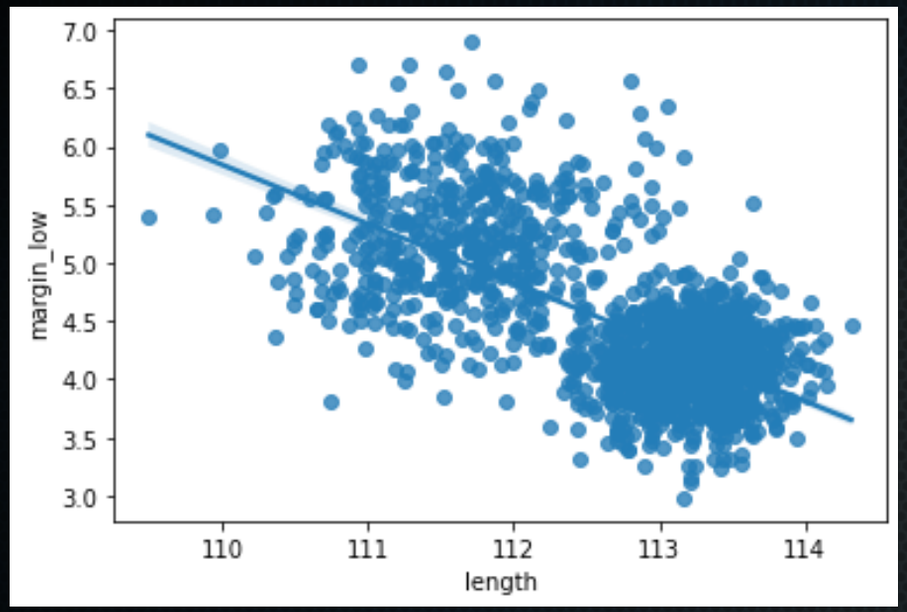
```
(1463, 7)
```

Création d'un fichier csv

```
bil_no_miss.to_csv('bill_dropna.csv', index=False)
```

Option 2 - Régression linéaire simple

Regplot



Score R² : 0,47

Code

```
X = bil_no_miss['length']
y = bil_no_miss['margin_low']
```

```
X.shape
```

```
(1463,)
```

```
y.shape
```

```
(1463,)
```

```
X = X.values.reshape(X.shape[0],1)
y = y.values.reshape(y.shape[0],1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

```
lrs = LinearRegression()
```

```
lrs.fit(X_train,y_train)
```

```
LinearRegression()
```

```
y_pred = lrs.predict(X_test)
```

```
r2_score(y_test, y_pred)
```

```
0.469322114307938
```

```
y_pred_train = lrs.predict(X_train)
```

```
r2_score(y_train, y_pred_train)
```

```
0.43877892959647924
```

Option 3 - Régression linéaire multiple

Vérification des conditions de la régression linéaire

Significativité des variables

```
import statsmodels.formula.api as smf
reg_multi = smf.ols('margin_low~diagonal+height_left+height_right+margin_up+length'
                    , data=bil_no_miss).fit()
print(reg_multi.summary())
```

```
==============================================================================
                 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
==============================================================================
```

Toutes les variables sont significatives

(p-value <0,05)

Colinéarité des variables (VIF)

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
variables = reg_multi.model.exog
[variance_inflation_factor(variables, i).round(2)
 for i in np.arange(1,variables.shape[1])]
```

```
[1.01, 1.14, 1.23, 1.4, 1.58]
```

Pas de problème de colinéarité

(tous les coefficients sont inférieurs à 10)

Modèle (pipeline), entraînement

```
li_reg_pipe = Pipeline([
    ('scale', StandardScaler()),
    ('model', LinearRegression())
])
```

```
li_reg_pipe.fit(X_train, y_train)
```

```
y_pred = li_reg_pipe.predict(X_test)
```

Score

```
tr_score = li_reg_pipe.score(X_train, y_train).round(4)
te_score = li_reg_pipe.score(X_test, y_test).round(4)
print(f"Score train: {tr_score} \nScore test: {te_score}")
```

```
Score train: 0.4732
Score test: 0.4943
```

Score R^2 : 0,49

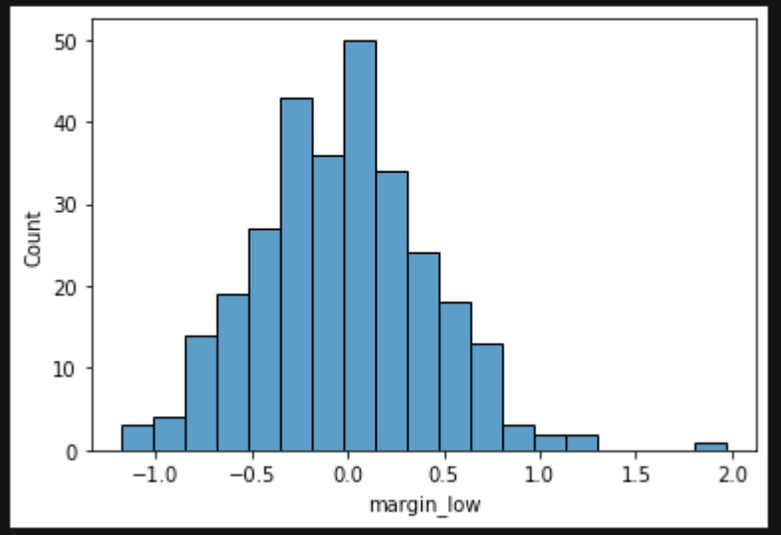
La régression linéaire multiple est l'option d'imputation retenue.

Validation du modèle : analyse des résidus

Normalité

```
sns.histplot(residual)
```

```
<AxesSubplot:xlabel='margin_low', ylabel='Count'>
```



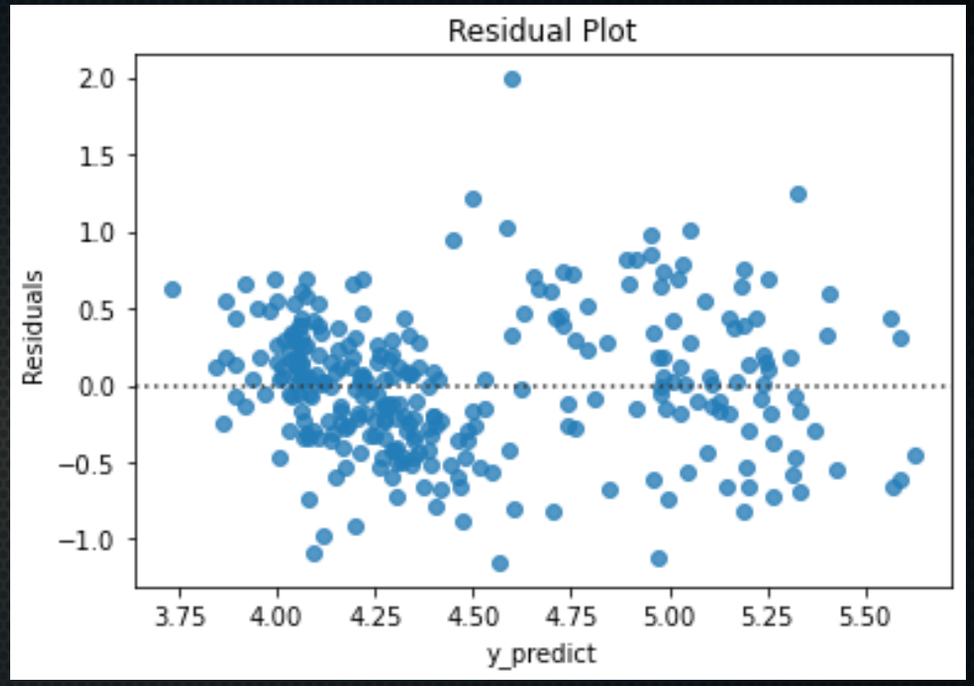

```
statistic, critical_values, levels = anderson(residual_flat_list)
print('Statistic:',statistic.round(3))
print('Critical value:',critical_values)
print('Niveaux:',levels)
```

```
Statistic: 0.275
Critical value: [0.568 0.647 0.777 0.906 1.078]
Niveaux: [15.  10.   5.   2.5  1. ]
```

Statistic < Critical value (quel que soit le niveau choisi):

on ne peut pas rejeter l'hypothèse de normalité.

Homoscédasticité



Variance constante des résidus

Pas de pattern notable

Modélisation

Modèle 1 - Dummy Classifier

X & y

```
X = bill.drop(columns='is_genuine', axis=0)
y = bill.is_genuine
```

Test train split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Fonction d'affichage des scores

```
def score(model):
    tr_score = model.score(X_train, y_train).round(4)
    te_score = model.score(X_test, y_test).round(4)
    print(f"Score train: {tr_score} \nScore test: {te_score}")
```

```
dc_model = Pipeline([
    ('scale', StandardScaler()),
    ('model', DummyClassifier(strategy='most_frequent'))
     ])
dc_model.fit(X_train, y_train)
```

```
Pipeline(steps=[('scale', StandardScaler()),
                ('model', DummyClassifier(strategy='most_frequent'))])
```

```
y_pred = dc_model.predict(X_test)
```

```
score(dc_model)
```

```
Score train: 0.6633
Score test: 0.68
```

```
mat = pd.DataFrame(confusion_matrix(y_test, y_pred))
mat
```

	0	1
0	0	96
1	0	204

Modèle 2 - Régression logistique

Vérification des conditions de la régression logistique

Significativité des variables

	coef	std err	z	P>|z|
Intercept	203.9841	241.233	0.846	0.398
diagonal	-0.0667	1.089	-0.061	0.951

*La variable **diagonal** n'est pas significative (p-value> 0,05).*
Elle doit être retirée.

	coef	std err	z	P>|z|
Intercept	192.7972	157.439	1.225	0.221
height_left	1.7266	1.102	1.566	0.117

*La variable **height_left** n'est pas significative (p-value> 0,05).*
Elle doit être retirée.

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	323.4569	139.414	2.320	0.020	50.211	596.702
height_right	2.7745	1.078	2.574	0.010	0.662	4.887
margin_low	5.9965	0.892	6.724	0.000	4.249	7.744
margin_up	10.1846	2.071	4.918	0.000	6.126	14.243
length	-5.9729	0.824	-7.252	0.000	-7.587	-4.359

Les 4 variables restantes sont significatives.

Colinéarité des variables (VIF)

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
variables = log_reg2.model.exog
[variance_inflation_factor(variables, i).round(2)
 for i in np.arange(1,variables.shape[1])]
```

```
[1.25, 1.91, 1.41, 2.11]
```

Pas de problème de colinéarité

(tous les coefficients sont inférieurs à 10)

Préparation des données

X & y

```
X = bill.drop(columns=['is_genuine', 'diagonal', 'height_left'], axis=0)
y = bill.is_genuine
```

Test train split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Optimisation par les hyper-paramètres

```
params={'model__C': np.logspace(-3,3,7),
        'model__penalty':['l1', 'l2'],
        'model__solver':['newton-cg', 'lbfgs', 'liblinear']
       }
```

```
grid = GridSearchCV(log_reg_pipe,
                    params,
                    cv=4,
                    return_train_score=True,
                    verbose=1)
grid.fit(X_train, y_train)
```

Entraînement et score (paramètres par défaut)

```
log_reg_pipe = Pipeline([
    ('scale', StandardScaler()),
    ('model', LogisticRegression())
    ])
```

```
log_reg_pipe.fit(X_train, y_train)
```

```
Pipeline(steps=[('scale', StandardScaler()), ('model', LogisticRegression())])
```

```
y_pred = log_reg_pipe.predict(X_test)
```

```
score(log_reg_pipe)
pd.DataFrame(confusion_matrix(y_test, y_pred))
```

```
Score train: 0.9925
Score test: 0.9867
```

	0	1
0	92	4
1	0	204

Entraînement et score (best_params)

```
log_reg_best = Pipeline([
    ('scale', StandardScaler()),
    ('model', LogisticRegression(C=1,solver='newton-cg',  penalty='l2'))
    ])
```

```
score(log_reg_best)
pd.DataFrame(confusion_matrix(y_test, y_pred))
```

```
Score train: 0.9925
Score test: 0.9867
```

	0	1
0	92	4
1	0	204

Les scores sont strictement identiques

avec ou sans optimisation par les hyper-paramètres.

Metrics

Matrice de confusion

	0	1
0	92	4
1	0	204

4 faux négatifs

Accuracy

```
accuracy_score(y_test, y_pred)
```

```
0.9867
```

Precision / Recall / F1 score

```
precision_score(y_test, y_pred).round(3)
```

```
0.981
```

```
recall_score(y_test, y_pred).round(3)
```

```
1.0
```

```
f1_score(y_test, y_pred).round(3)
```

```
0.99
```

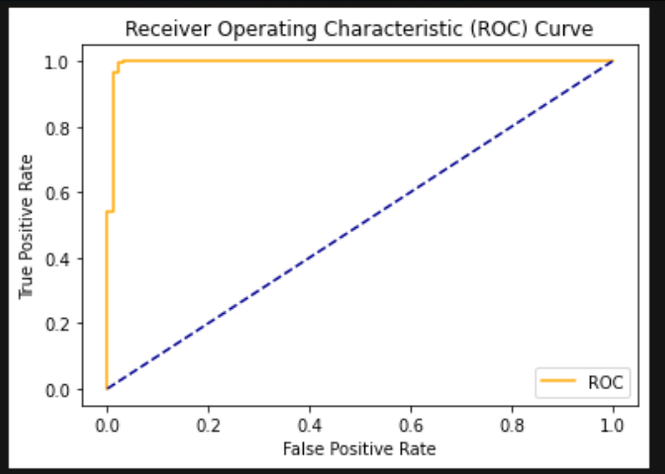
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

       False       1.00      0.96      0.98        96
        True       0.98      1.00      0.99       204

    accuracy                           0.99       300
```

Roc curve, AUC

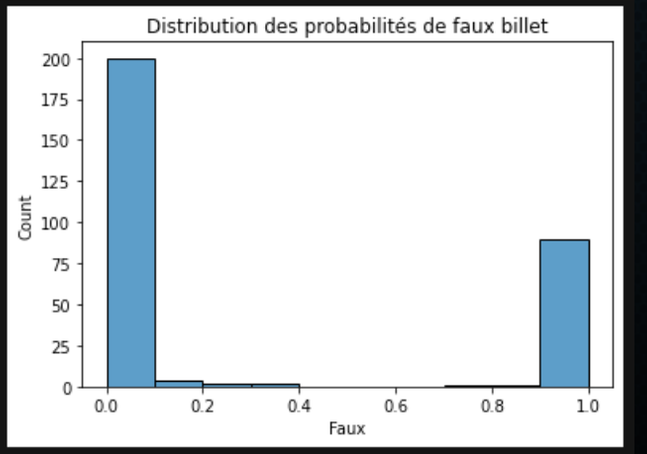



```
auc = roc_auc_score(y_test,y_pred_proba)
print('AUC: %.4f' % auc)
```

```
AUC: 0.9948
```

Probabilités

	Faux	Vrai
131	0.00	1.00
74	1.00	0.00
280	0.00	1.00
201	0.01	0.99
233	0.36	0.64
117	0.00	1.00
56	0.01	0.99
42	0.02	0.98
199	0.01	0.99
71	0.00	1.00



Modèle 3 - K-means

Modèle

```
km_model=KMeans(n_clusters=2, init='k-means++', random_state=103)
km_model.fit(X_init)
```

```
centroids  = km_model.cluster_centers_
centroids
```

```
array([[171.89849174, 104.19301653, 104.14485537,   5.23785755,
          3.3472314 , 111.5918595 ],
       [171.98699803, 103.95165354, 103.81333661,   4.12430421,
          3.0582185 , 113.19615157]])
```

Metrics

Matrice de confusion

	0	1
0	481	19
1	3	997

Accuracy

```
accuracy_score(bill['is_genuine'], bill['Labels K'])
```

```
0.9853
```

Precision / Recall / F1 score

```
print(classification_report(bill['is_genuine'], bill['Labels K']))
```

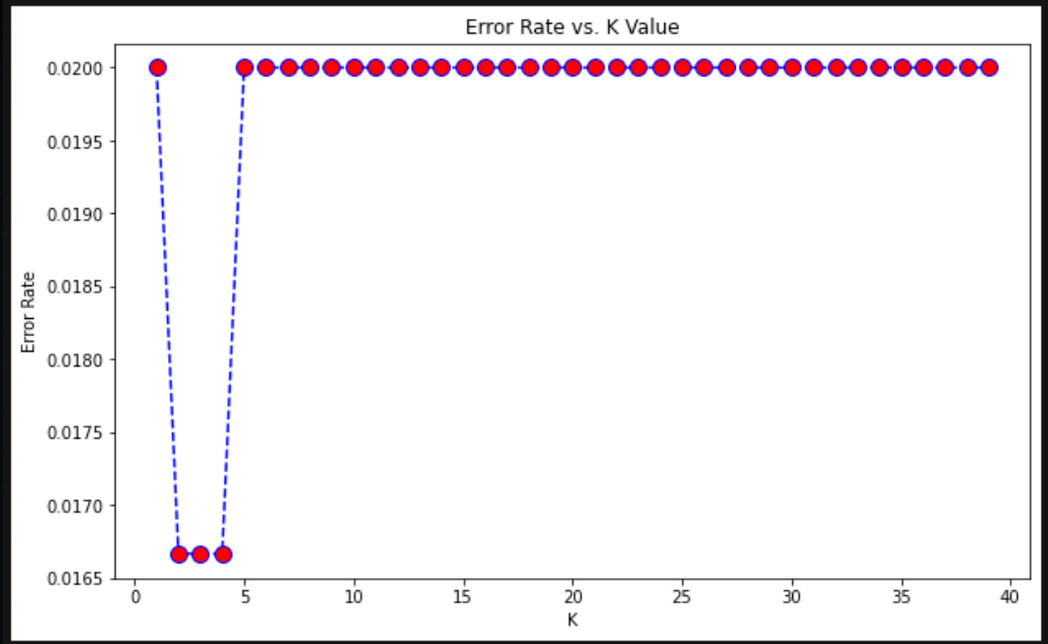
```
              precision    recall  f1-score   support

       False       0.99      0.96      0.98       500
        True       0.98      1.00      0.99      1000

    accuracy                           0.99      1500
```

Modèle 4 - K-Nearest Neighbors (KNN)

Modèle




```
knn = neighbors.KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)
```

Metrics

Matrice de confusion

	0	1
0	92	4
1	1	203

Accuracy

```
accuracy_score(y_test,pred)
```

```
0.9833
```

Precision / Recall / F1 score

	precision	recall	f1-score	support
False	0.99	0.96	0.97	96
True	0.98	1.00	0.99	204
accuracy			0.98	300

Conclusion

Comparaison des modèles

Régression logistique

	precision	recall	f1-score	support
False	1.00	0.96	0.98	96
True	0.98	1.00	0.99	204
accuracy			0.99	300

K-means

	precision	recall	f1-score	support
False	0.99	0.96	0.98	500
True	0.98	1.00	0.99	1000
accuracy			0.99	1500

K-NN

	precision	recall	f1-score	support
False	0.99	0.96	0.97	96
True	0.98	1.00	0.99	204
accuracy			0.98	300

Globalement, les scores des modèles étudiés sont plutôt proches.

La régression logistique est légèrement plus performante. C'est le modèle retenu.

Remarques et pistes d'amélioration

- Améliorer l'imputation des valeurs manquantes :
chercher à en déterminer le type (totalement aléatoires ou pas)

- Diminuer le nombre de faux négatifs :
4 faux billets sur 100 sont prédits comme vrais par le modèle. Déplacer le seuil de classification peut être souhaitable, quitte à augmenter le nombre de faux positifs (Precision/Recall tradeoff).

- Essayer d'autres modèles de classification :
Decision Tree, Random Forest, Gradient boost, ...

- Intégrer de nouvelles features dans les données