ONCFM

Détection de faux billets

Introduction

… la lutte contre le faux monnayage

L’Organisation nationale de lutte contre le faux monnayage ou ONCFM, est une organisation publique ayant pour objectif de mettre en place des méthodes d’identification de contrefaçons de billets en euros.

L'objectif de ce projet, sera de mettre en place un algorithme qui soit capable de différencier automatiquement les vrais des faux billets en fonction de leurs dimensions.

Sommaire

ONCFM

Exploration des données

Dimensions du fichier — 1500 entrées et 7 variables

Format des variables — 6 variables quantitatives et 1 variable qualitative

Lignes dupliquées — Aucune ligne dupliquée

Valeurs manquantes — 37 valeurs manquantes

ONCFM

La variable qualitative

... son utilité

"is_genuine"

... une variable booléenne

Le type de variable

"is_geniune" est une variable de type booléenne qui apporte une réponse binaire

le type de réponse

"is_geniune" indique si un billet est vrai ou faux avec 2 réponses [True , False]

ONCFM

Exploration des valeurs manquantes

... leur importance

Identification	Nombre
"margin_low"	37
Pourcentage (%)	Choix pour l'imputation
2.47 %	Régression linéaire

ONCFM

Régression linéaire

... imputation des valeurs manquantes

Régression linéaire

... principes de base

La régression linéaire permet d'expliquer une variable Y, en fonction d'une ou plusieurs variables X (régresseur ou covariable).

Cela suppose qu'il existe une relation linéaire entre les variables X et Y.

Tout modèle est une approximation de la réalité, aussi on ajoute un terme d’erreur qui sert à tenir compte du fait qu’aucune relation linéaire exacte ne lie X et Y.

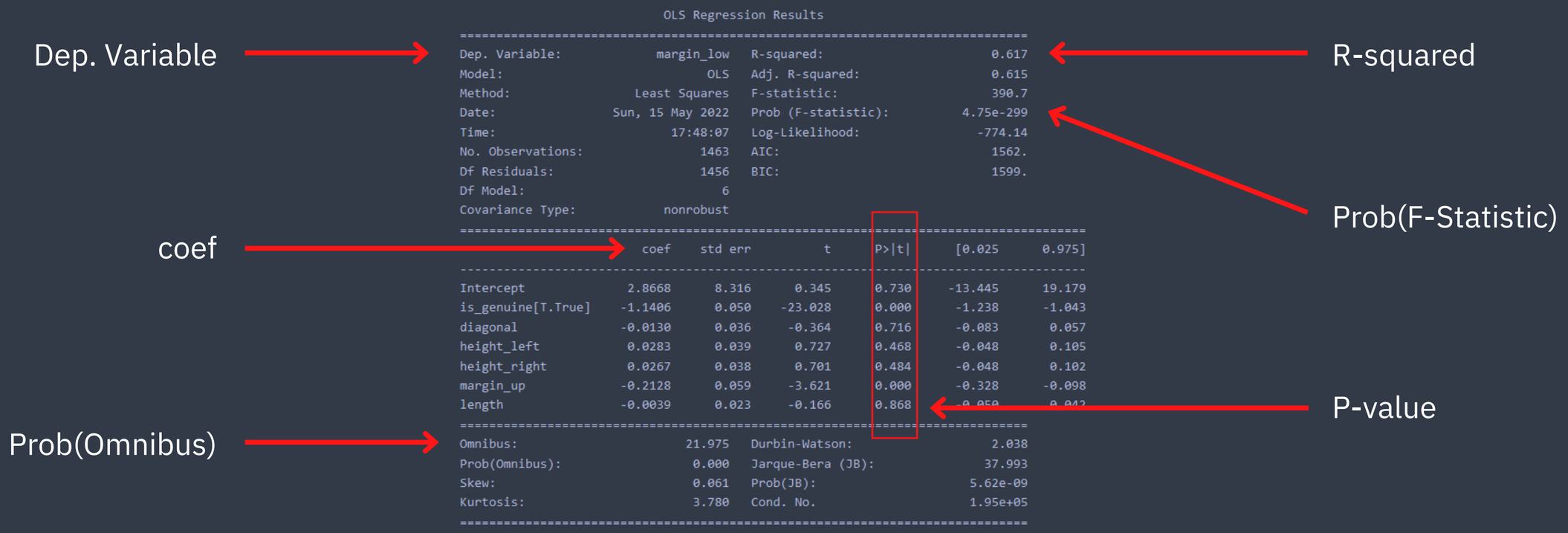
$$Y = \beta 1 + \beta 2X + \varepsilon$$

- **Y** est la variable à expliquer
- **X** est la variable explicative
- **β1** et **β2** sont des paramètres
- **ε** est le terme d'erreur

Régression linéaire

… construction du modèle vias statsmodels

```
# Instanciation
reg_multi = smf.ols('margin_low ~ is_genuine + diagonal + height_left + height_right + margin_up + length', data = df1).fit()

# Affichage statistiques
print(reg_multi.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:	Sun, 15 May 2022	Prob (F-statistic):	4.75e-299
Time:	17:48:07	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[T.True]	-1.1406	0.050	-23.028	0.000	-1.238	-1.043
diagonal	-0.0130	0.036	-0.364	0.716	-0.083	0.057
height_left	0.0283	0.039	0.727	0.468	-0.048	0.105
height_right	0.0267	0.038	0.701	0.484	-0.048	0.102
margin_up	-0.2128	0.059	-3.621	0.000	-0.328	-0.098
length	-0.0039	0.023	-0.166	0.868	-0.050	0.042

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

Dep. Variable

est la variable dépendante de notre modèle soit "**margin_low**"

R-squared

le coefficient de détermination indique la variation en pourcentage expliquée par les variables indépendantes, soit **61,7%** dans notre modèle. Ce coefficient varie de 0 à 1 et mesure la proximité des données avec la droite de régression ajustée.

coef

représente la relation entre la variable indépendante et la variable dépendante pour 1 unité de mesure tout en maintenant d'autres prédicteurs dans le modèle.

ex : Si `**length**` change d'une unité (1 mm), l'estimation de `**margin_low**` diminuera de 0,0039 mm en moyenne. ***NB** : Un coefficient négatif représente une association linéaire négative*

Prob(Omnibus)

Omnibus test est une des hypothèses d'OLS qui indique si les résidus sont distribués selon une loi normale.

Prob (F-statistic)

Prob(F-statistics) est un test utilisé pour évaluer le pouvoir explicatif d'un groupe de variables indépendantes sur la variation de la variable dépendante.

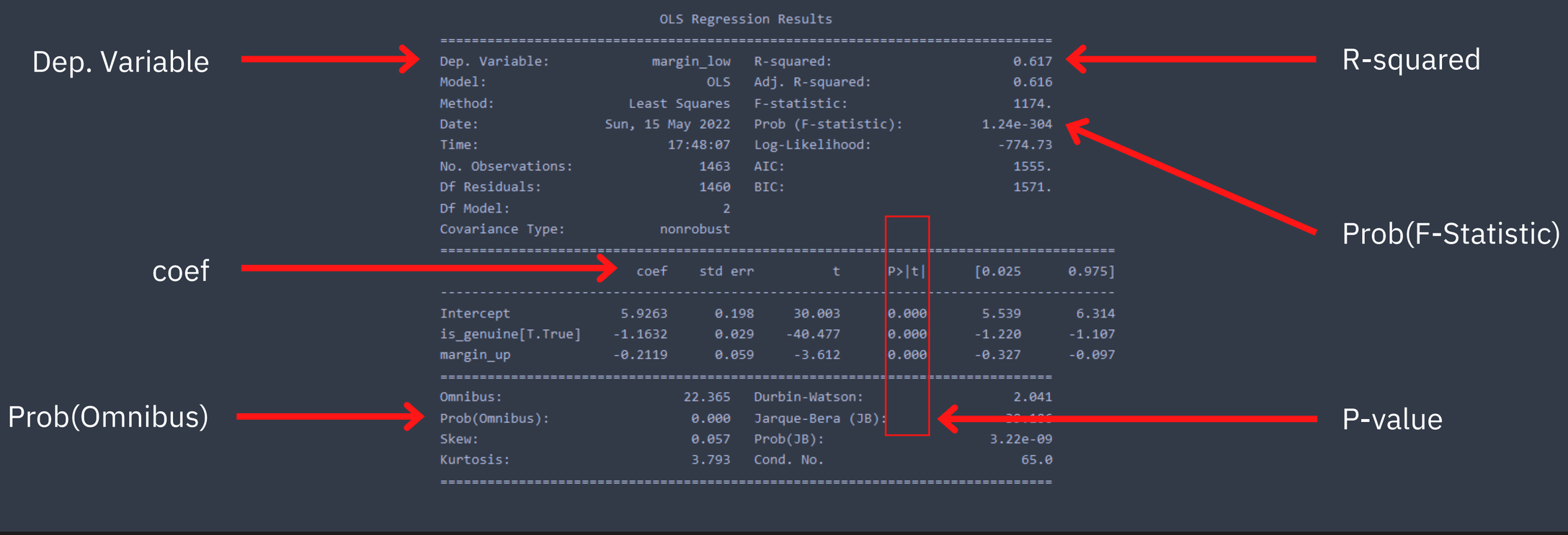
P-value

Evalue le niveau de significativité de chaque variable. Pour une significativité optimale, son seuil ne doit pas dépasser 5% au risque de laisser penser à un soucis de multi colinéarité. Dans notre cas plusieurs variables sont au delà de ce seuil. Nous réajusterons donc notre modèle en retirant ces variables.

Suppression des variables

… non significatives

```
# Instanciation
reg_multi = smf.ols('margin_low ~ is_genuine + margin_up , data = df1).fit()

# Affichage statistiques
print(reg_multi.summary())
```



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:	Sun, 15 May 2022	Prob (F-statistic):	1.24e-304
Time:	17:48:07	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[T.True]	-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

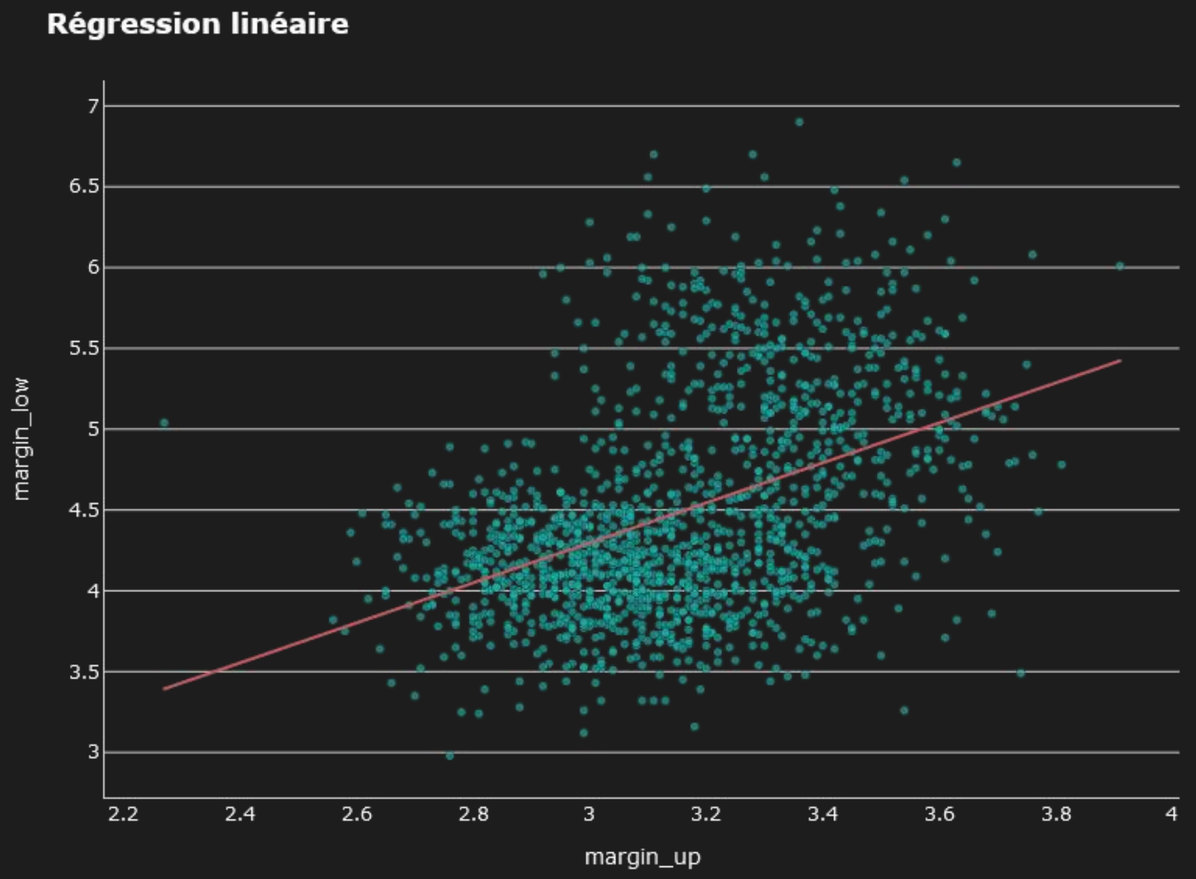
Omnibus:	22.365	Durbin-Watson:	2.041
Prob(Omnibus):	0.000	Jarque-Bera (JB):	[illegible]
Skew:	0.057	Prob(JB):	3.22e-09
Kurtosis:	3.793	Cond. No.	65.0

Modèle final

... représentation graphique

Tous les paramètres sont désormais significatifs.

R-squared = 0.617
Adj. R-squared = 0.616
Prob (F-statistic) = 1.24e - 304



Analyse des résultats

... hypothèses

Multicolinéarité

La **multicolinéarité** fait référence à une situation dans laquelle plus de deux variables explicatives dans un modèle de régression multiple sont fortement liées de manière linéaire.

Homocédasticité

On parle d'**homoscédasticité** lorsque la variance des erreurs de la régression est la même pour chaque observation. Le test de *Breusch-Pagan* permet de tester cette hypothèse

Normalité des résidus

Les tests de normalité permettent de vérifier si des données suivent une loi normale.
Omnibus test est une de ces hypothèses qui dans notre cas n'est pas satisfaite.
Nous créerons une matrice des résidus que nous représenterons par un histogramme.

Analyse des résultats

... multicolinéarité et homocédasticité

Multicolinéarité

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

```
[1.5938854494007753, 1.5938854494007746]
```

Les coefficients sont inférieurs à 10, il n'y a donc pas de problème de colinéarité.

Homocédasticité

```
_, pval, __, f_pval = statsmodels.stats.diagnostic.het_breuschpagan(reg_multi.resid, variables)
print('P-value test Breusch Pagan:', pval)
```

```
P-value test Breusch Pagan : 3.2033559115838186e-36
```

La P-valeur est inférieure à 5%, on rejette l'hypothèse selon laquelle les variances sont constantes.

Analyse des résultats

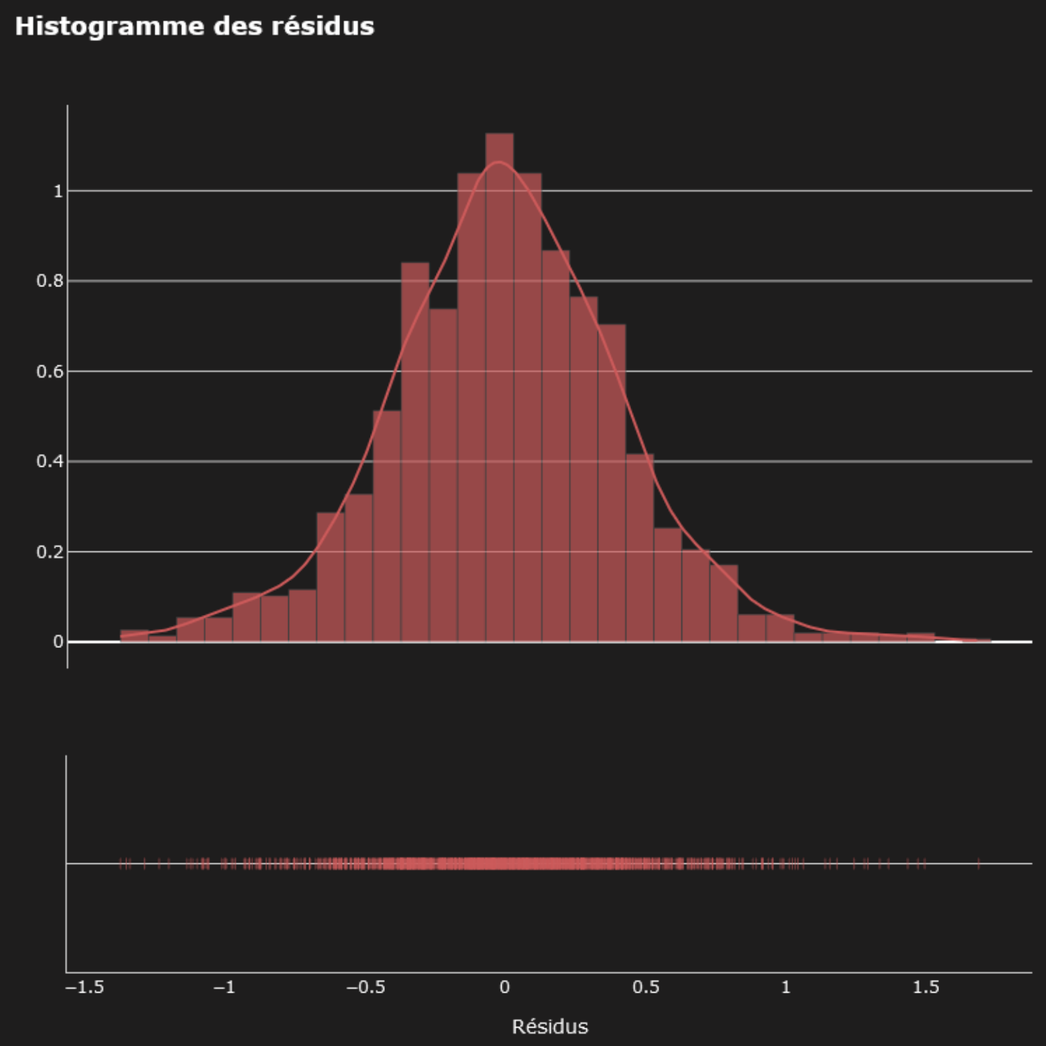
... résidus

```
# Création d'une matrice des résidus
residus = reg_multi.resid
```

Normalité des résidus

On appelle résidu la différence entre la valeur observée pour la variable à expliquer et son estimation.

Nous avons donc utiliser la fonction **.resid** de statsmodels afin de générer l'histogramme des résidus qui confirme la normalité de ces derniers.



Imputation des valeurs manquantes

... itératives

```
# Remplacement des valeurs manquantes de "margin_low" par les prédictions de la regression linéaire
data_complete.loc[data_complete["margin_low"].isnull(), "margin_low"] = reg_multi.predict(data_complete)

# Affichage des dimensions
print('Dimensions :',data_complete.shape)
# Identification du nombre de lignes en doublon
print('Nombre de lignes dupliquées :',data_complete.duplicated().sum())
# Nombre de valeurs manquantes
print('Nombre total de valeurs manquantes :', data_complete.isnull().sum().sum(),'\n')
```

```
Dimensions : (1500, 7)
Nombre de lignes dupliquées : 0
Nombre total de valeurs manquantes : 0
```

Analyse exploratoire

Répartition des billets

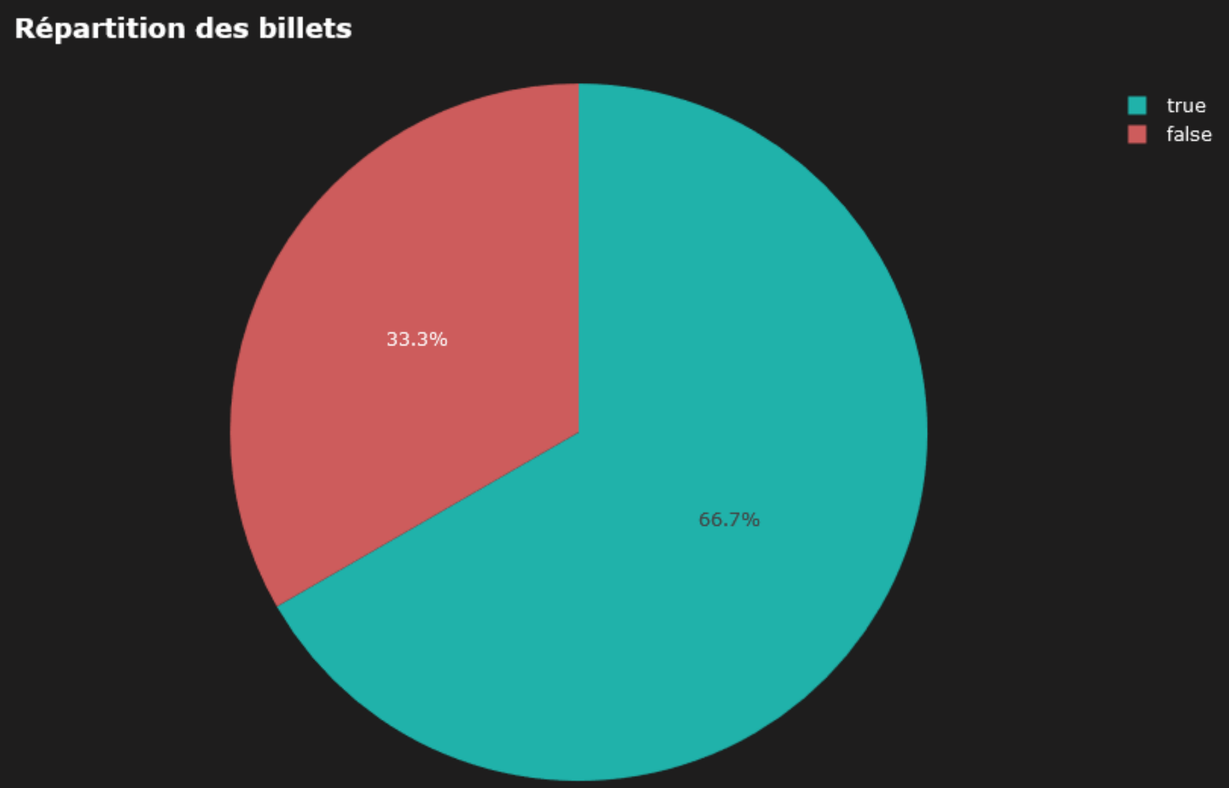
… leur nombre

```
print('Répartition du nombre des billets :', '\n',data_complete.is_genuine.value_counts())
```

```
Répartition du nombre des billets :
True     1000
False     500
```

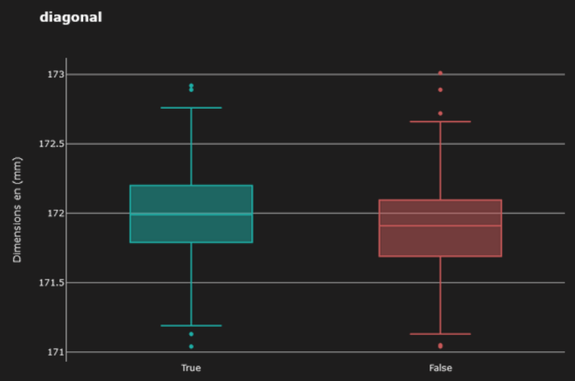
Le fichier contient 1500 billets dont :

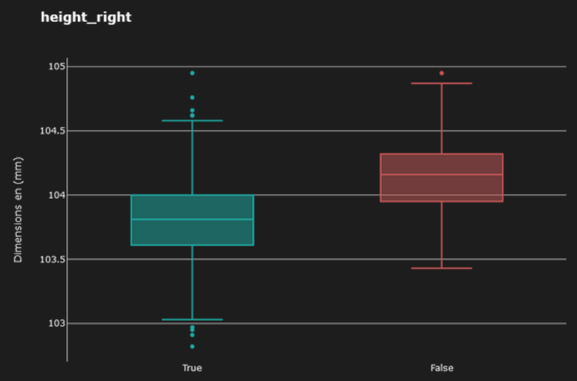
- 1000 vrais billets
- 500 faux billets

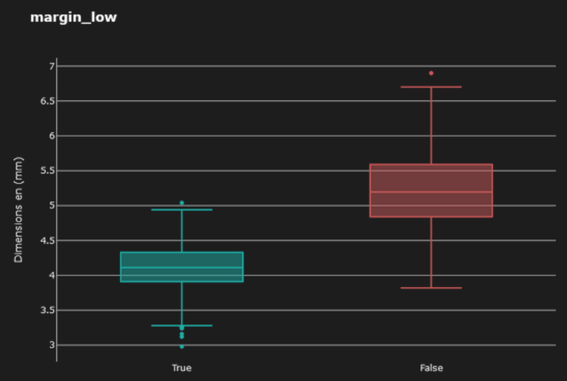


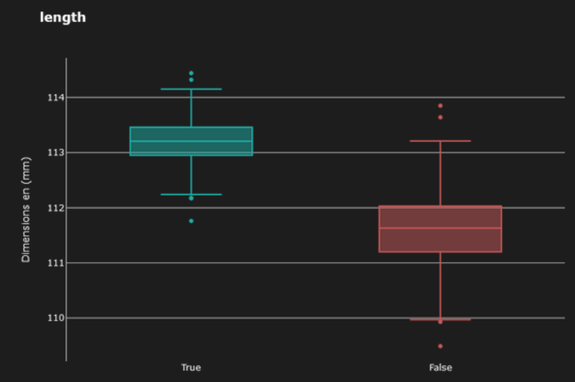
Répartition des billets

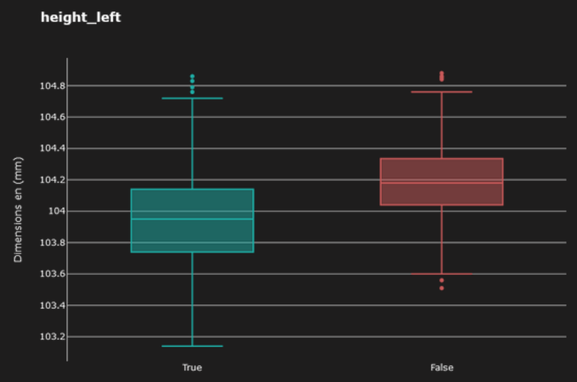
… leurs dimensions

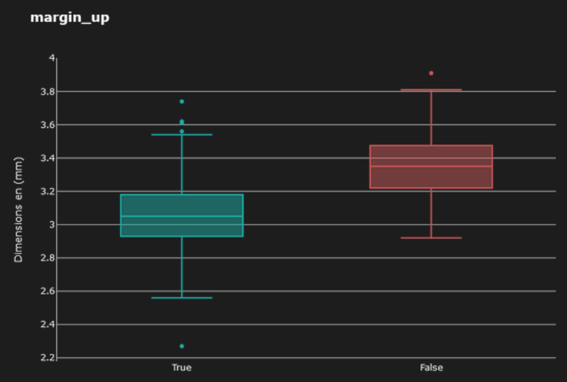


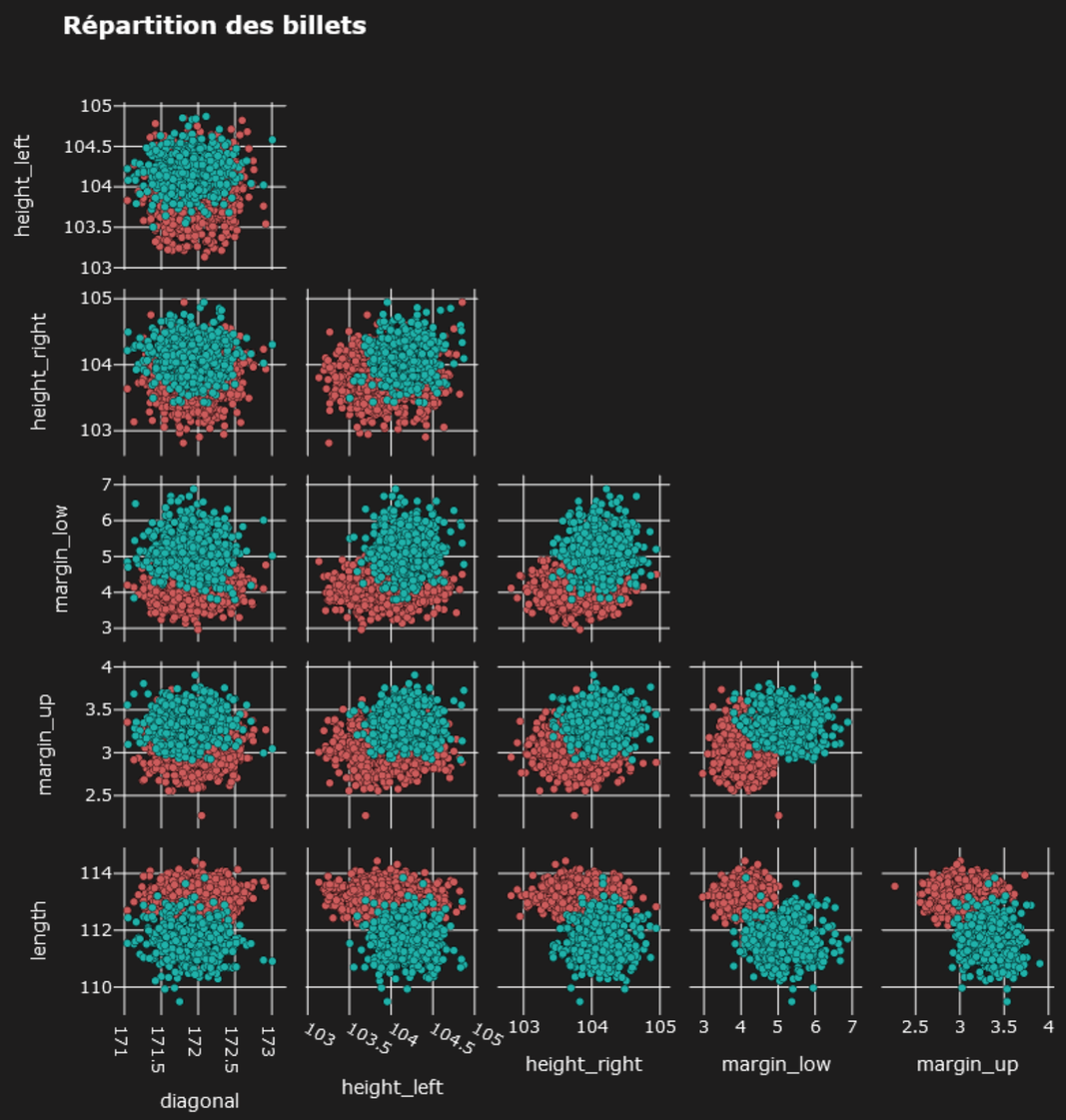












Répartition des billets

... conclusion

Dans notre échantillon, nous avons 2 fois plus de vrais que de faux billets.

De manière générales, les faux billets sont plus larges et plus court que les vrais.

Cela implique de plus grandes marges entre les bords des billets et les images mais aussi une forme certainement plus carré pour les faux billets.

ONCFM

Classification par K-Means

Classification K-Means

… principes de base

La méthode des k-means permet de regrouper les individus en un nombre de clusters distincts.

Cette méthode repose sur la minimisation de la somme des distances au carré entre chaque individu et le centroïde (le point central) de son cluster.

Classification K-Means

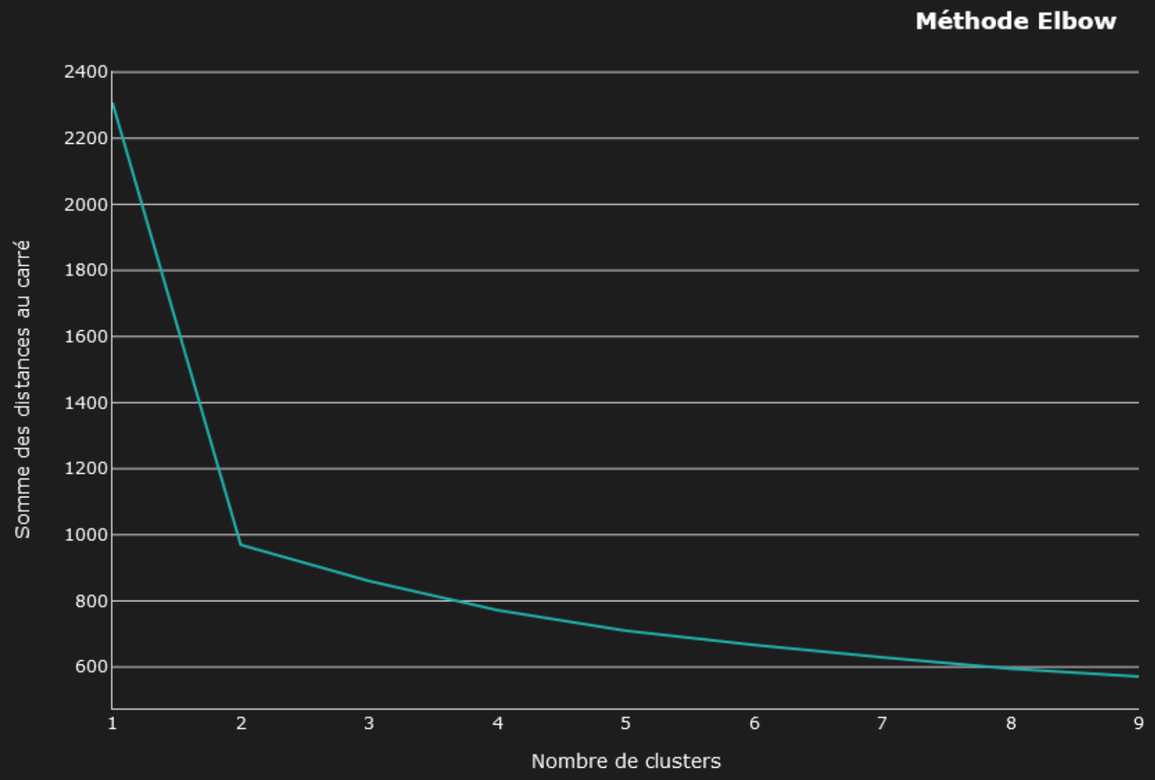
... la méthode du Elbow

La méthode Elbow consiste à tracer la variation expliquée en fonction du nombre de clusters.

Le coude de la courbe détermine le nombre de clusters optimal qui dans notre cas est de 2 .

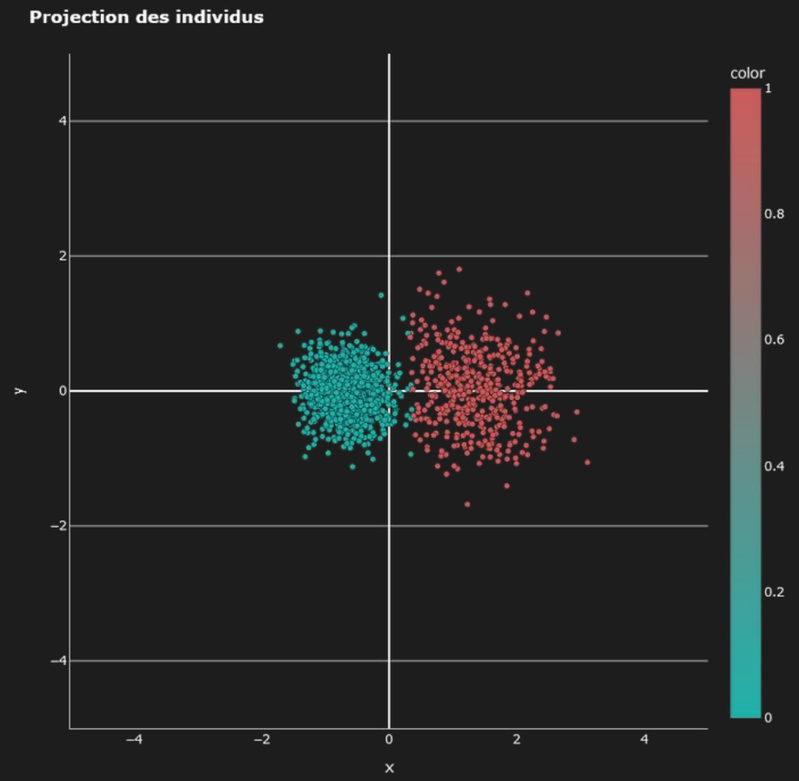
Les clusters sont répartis de la façon suivante :

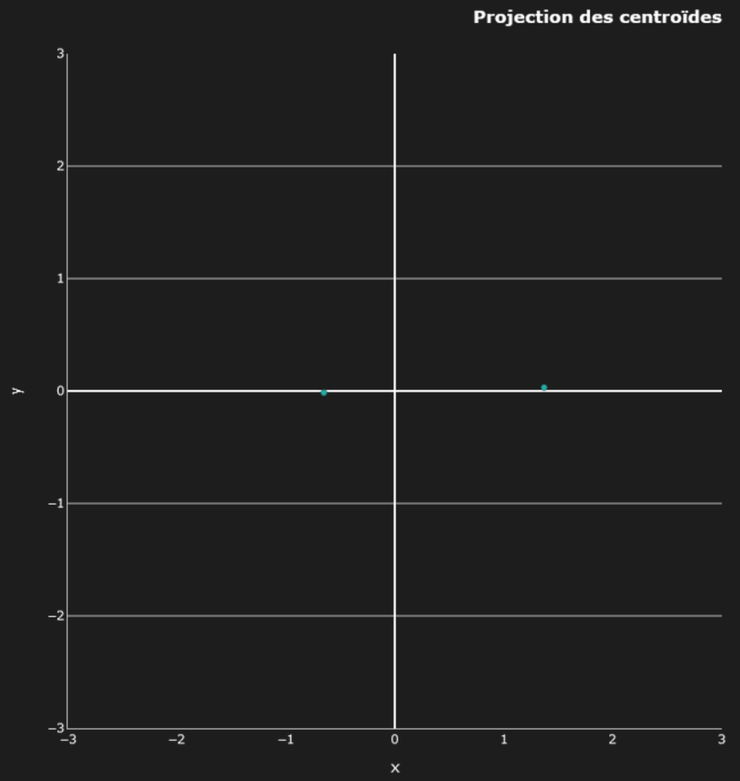
- Vrais billets = 0
- Faux billets = 1



Classification K-Means

... projection des individus et centroïdes

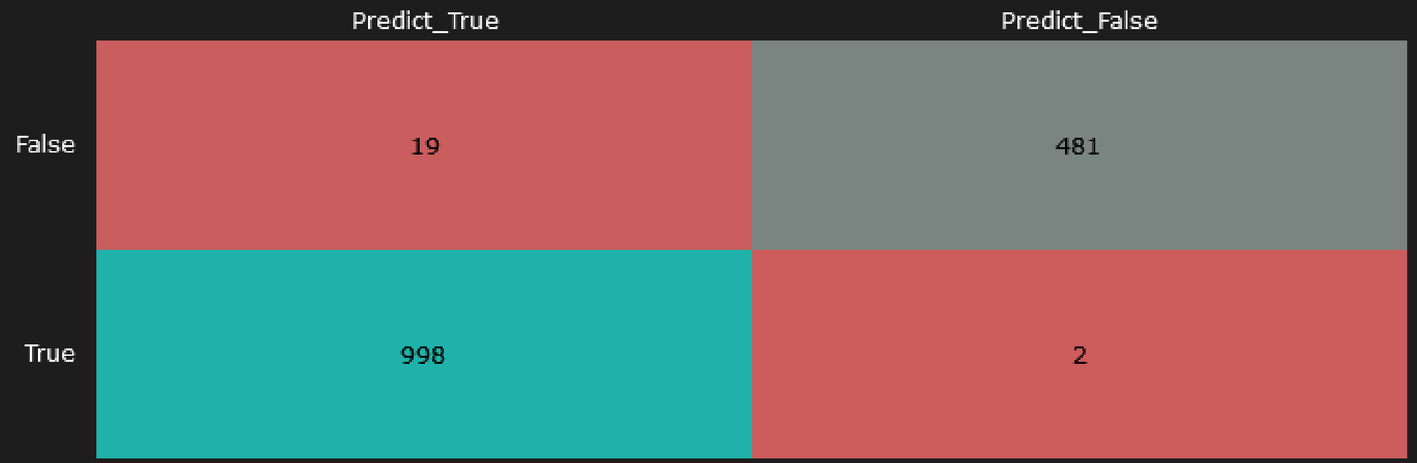




Classification K-Means

... matrice de confusion

Matrice de confusion




```python
# Test de classification
accuracy = metrics.accuracy_score(y_true, y_pred)
print("Accuracy :", accuracy)

# Test de précision
precision = metrics.precision_score(y_true, y_pred)
print("Precision :", precision)

# Test de rappel
recall = metrics.recall_score(y_true, y_pred)
print("Recall :", recall)

# Rapport de classification
print(classification_report(y_true, y_pred))
```

```
Accuracy : 0.986
Precision : 0.9958592132505176
Recall : 0.962

              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1000
           1       1.00      0.96      0.98       500

    accuracy                           0.99      1500
   macro avg       0.99      0.98      0.98      1500
weighted avg       0.99      0.99      0.99      1500
```

Classification K-Means

… application du modèle aux données de test

```
# Création copie DataFrame
data_test_km = data_test.copy()

# Clustering par K-means
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)

# Etiquettes de clusters
clusters_kmeans = kmeans.labels_

# Predictions sur des donnees de test
data_test_km["cluster_pred"] = kmeans.predict(data_test_km[["diagonal","height_left","height_right","margin_low","margin_up","length"]])

# Création colonne "statut_billet"
data_test_km["statut_billet"] = np.where(data_test_km["cluster_pred"]>=0.5,"Vrai billet","Faux Billet")

# Affichage DataFrame
print(data_test_km[["id","cluster_pred", "statut_billet"]])
```

```
     id       cluster_pred       statut_billet

0  A_1                   1         Vrai billet
1  A_2                   1         Vrai billet
2  A_3                   1         Vrai billet
3  A_4                   0        Faux Billet
4  A_5                   0        Faux Billet
```

Régression logistique

Régression logistique

… principes de base

La régression logistique est une méthode de classification supervisée qui permet de modéliser et de classifier une variable binaire en fonction de variables explicatives quantitatives

L'objectif de cette régression est de construire un modèle qui sera amener à prédire à partir de n'importe quel fichier, dans quelle catégorie chaque billet se situe.

Les clusters sont répartis de la façon suivante :

- Vrais billets = 0
- Faux billets = 1

Régression logistique

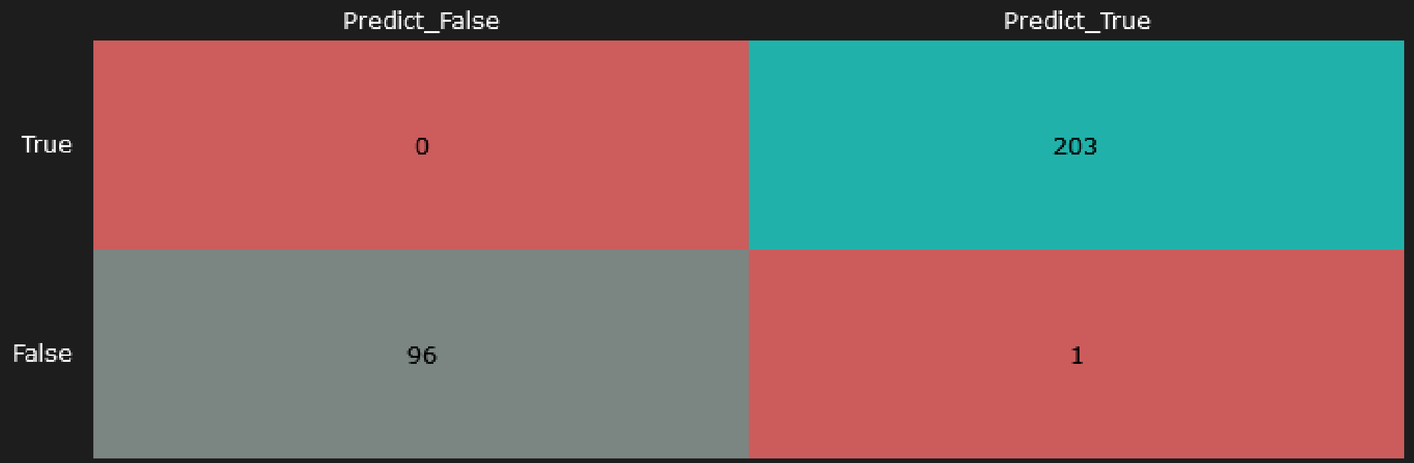
... construction du modèle via statsmodels

```
# Création copie DataFrame
data = data_complete.copy()

# Imputation de la variable 'is_genuine' en binaire
data['is_genuine'] = pd.get_dummies(data['is_genuine'],drop_first=True)

# Séparation des données
X = data.drop(columns=('is_genuine'))
y = data["is_genuine"]

# Séparation des données en Tarining Set et testing Set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Entrainement du modèle
model = LogisticRegression(random_state=0)
model.fit(X_train, y_train)

# Prédiction du modèle sur le Testing Set
y_pred = model.predict(X_test)
```

Régression logistique

... matrice de confusion

Matrice de confusion




```python
# Test de classification
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy :", accuracy)

# Test de précision
precision = metrics.precision_score(y_test, y_pred)
print("Precision :", precision)

# Test de rappel
recall = metrics.recall_score(y_test, y_pred)
print("Recall :", recall)

# Rapport de classification
print(classification_report(y_test, y_pred))
```

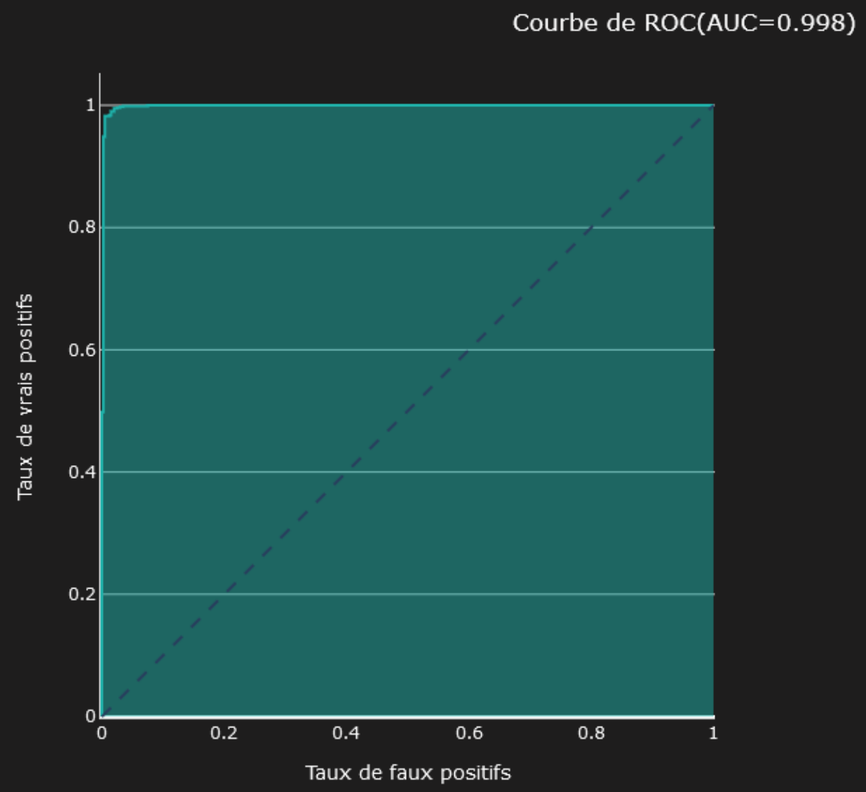
```
Accuracy : 0.9966666666666667
Precision : 0.9950980392156863
Recall : 1.0

              precision    recall  f1-score   support

           0       1.00      0.99      0.99        97
           1       1.00      1.00      1.00       203

    accuracy                           1.00       300
   macro avg       1.00      0.99      1.00       300
weighted avg       1.00      1.00      1.00       300
```

Régression logistique

... courbe de ROC

La courbe ROC (Receiver Operator Characteristic) est un graphique utilisé pour montrer la capacité de diagnostic des classificateurs binaires.

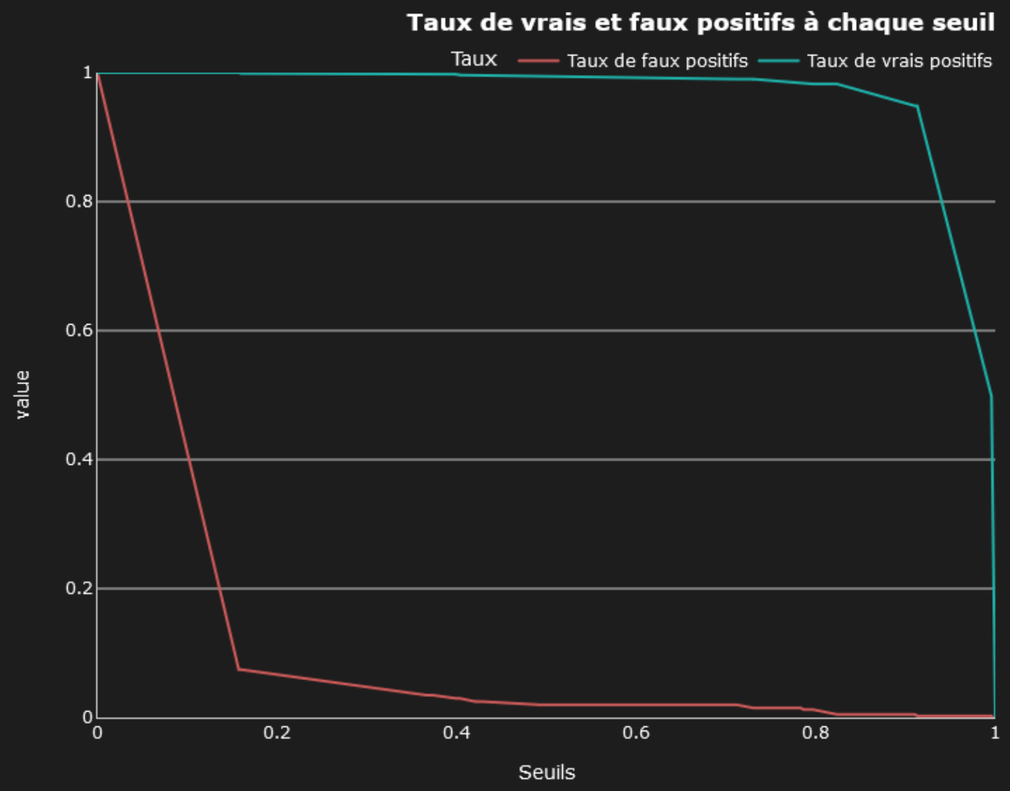
Dans notre modèle, la courbe nous montre la capacité de notre modèle à prédire les vrais et faux positifs.



Régression logistique

... taux de vrais/faux positifs

En complément de la courbe ROC (Receiver Operator Characteristic), ce graphique nous montre distinctement le taux d'erreur de prédiction des vrais et faux positif à chaque seuil.



Régression logistique

… application du modèle aux données de test

```
# Création copie DataFrame
data_sc = data_test.copy()

# Séparation des données
X = data_sc.drop(columns=('id'))

# Prédiction du modèle sur les données de test
data_sc['cluster_pred'] = model.predict(X)

# Calcul des probabilités d'affectation sur l'échantillon exemple
probas_new = model.predict_proba(X)

# Score de présence : Probalilité de chance que le billet soit VRAI (is_genuine = 0)
score_new = probas_new[:,0]

# Ajout et configuration de la colonne probabilité
data_sc["proba_true"] = score_new
data_sc["proba_true"] = (data_sc["proba_true"]*100).round(2)

# Affichage DataFrame
print(data_sc[["id","proba_true","cluster_pred"]])
```

	id	proba_true	cluster_pred	statut_billet
0	A_1	99.42	0	Vrai billet
1	A_2	99.90	0	Vrai billet
2	A_3	99.86	0	Vrai billet
3	A_4	8.88	1	Faux Billet
4	A_5	0.04	1	Faux Billet

MERCI

DE VOTRE ATTENTION