

# Projet Grands Risques, Valeurs Extrêmes

A l'attention du Professeur ROUSSELLE

Astrid BROUCAS

Benjamin LE GUELLEC

Sebastien HAAG

ING3 Actuariat

13/12/2024

# Table des matières

- Introduction
- Initialisation du projet
- Lecture du jeu de données
- Modélisation de la série X
- Modélisation de la série Y
- Modélisation de la dépendance des séries X et Y grâce à la théorie des copules
- Perte annuelle combinée de X et Y de période de retour 200 ans
- Conclusion
- Bibliographie

# Introduction

Ce projet a pour objectif de mettre en application les notions étudiées tout au long du cours « Grands risques, Valeurs extrêmes ». Il repose sur l'analyse de deux séries de données représentant des pertes mensuelles enregistrées sur deux portefeuilles distincts, X et Y.

Le but de ce travail est de modéliser séparément les séries X et Y en utilisant plusieurs approches, notamment la loi normale, la loi GEV ou encore la loi GPD, afin de sélectionner le modèle le plus adapté pour chacune des séries. Puis, nous modéliserons la dépendance entre les séries X et Y à l'aide de la théorie des copules. Nous attacherons également à calculer les pertes annuelles associées à une période de retour de 200 ans pour la série X ainsi que la perte annuelle combinée correspondant à la même période de retour.

Notre rapport sera structuré en trois grandes parties. Dans un premier temps, nous nous intéresserons à la modélisation de la série X, où nous présentons et comparons les différents modèles, avant de déterminer la perte annuelle de période de retour 200 ans. Ensuite, nous nous concentrerons sur la modélisation de la série Y en utilisant une démarche similaire à celle de la première partie. Enfin, nous nous attacherons à modéliser la dépendance entre les séries X et Y, où nous utilisons des copules pour simuler et analyser la perte annuelle combinée.

# Initialisation du projet

Pour la réalisation de ce projet, nous avons choisi d'utiliser le langage Python plutôt que le langage R. Ce choix s'explique par le fait que tous les membres de notre groupe étaient plus à l'aise avec Python qu'avec le logiciel R, ce qui nous a permis de travailler de manière efficace.

Afin de répondre aux exigences de ce projet, nous avons utilisé plusieurs modules Python. Voici les différents modules que nous avons exploité au cours de notre projet.

- PyExtremes : Ce module est conçu pour extraire et analyser les valeurs extrêmes d'une série temporelle, permettant d'estimer des quantiles ou des seuils critiques à partir de la théorie des valeurs extrêmes.
- SKExtremes : Bien que ce module nécessite une installation depuis GitHub, nous avons choisi de l'utiliser car il offre une interface simplifiée pour modéliser la loi GEV.
- Pandas : Ce module est indispensable pour la manipulation et l'analyse des données. Nous l'avons utilisé pour la gestion de notre jeu de données Excel.
- SciPy.stats : Ce module nous a permis d'accéder à un large éventail de fonctions statistiques nécessaires pour la modélisation et l'estimation des paramètres des différents modèles étudiés.
- Fitter : Cet outil nous a aidés à comparer et ajuster les distributions statistiques aux données.
- Seaborn : Ce module de visualisation graphique nous a permis de créer des représentations claires et esthétiques, facilitant ainsi l'interprétation de nos résultats.
- Copulalib : Ce module est utilisé pour modéliser et analyser la dépendance entre variables en ajustant des copules paramétriques (comme Gaussienne, Student, Clayton, Frank, Gumbel, ...)
- Sklearn : Cette bibliothèque Python est utilisée pour l'analyse de données, offrant des outils pour la classification, la régression, la réduction de dimensions, la validation croisée et le traitement des données.

```
In [62]: import numpy as np
from pyextremes import plot_mean_residual_life
import nbconvert
import pandas as pd
import scipy
import scipy.stats
import time
import scipy.stats as stats
from scipy.stats import genextreme
from scipy.optimize import minimize
import skextremes as ske
import matplotlib.pyplot as plt
%matplotlib inline
```

```

import seaborn as sns
from fitter import Fitter
import statsmodels.api as sm
from statsmodels.graphics.gofplots import ProbPlot
from scipy.stats import norm, kstest, multivariate_normal
from scipy.stats import genpareto
from copulalib.copulalib import Copula
from statsmodels.distributions.empirical_distribution import ECDF
from scipy.stats import rankdata
from sklearn.metrics import log_loss
import warnings

# Supprimer les warnings RuntimeWarning
warnings.filterwarnings("ignore", category=RuntimeWarning)

```

## Lecture du jeu de données

Après avoir importé ces modules, nous avons commencé par explorer notre jeu de données. Celui-ci, contenait deux séries d'observations mensuelles de pertes réalisées sur les portefeuilles X et Y. Nous avons commencé par lire notre base de données en utilisant la librairie Pandas et nous avons choisi d'afficher les premières lignes afin de bien comprendre la structure des données que nous allions manipuler.

```

In [63]: data = pd.read_excel('Base X Y.xlsx')
data_x = pd.DataFrame(data).drop(['Y'], axis = 1)
x_wo_date=data_x['X']
data_y = pd.DataFrame(data).drop(['X'], axis = 1)
y_wo_date=data_y['Y']
## Afficher les 5 premières lignes de la base de données
data.head(5)

```

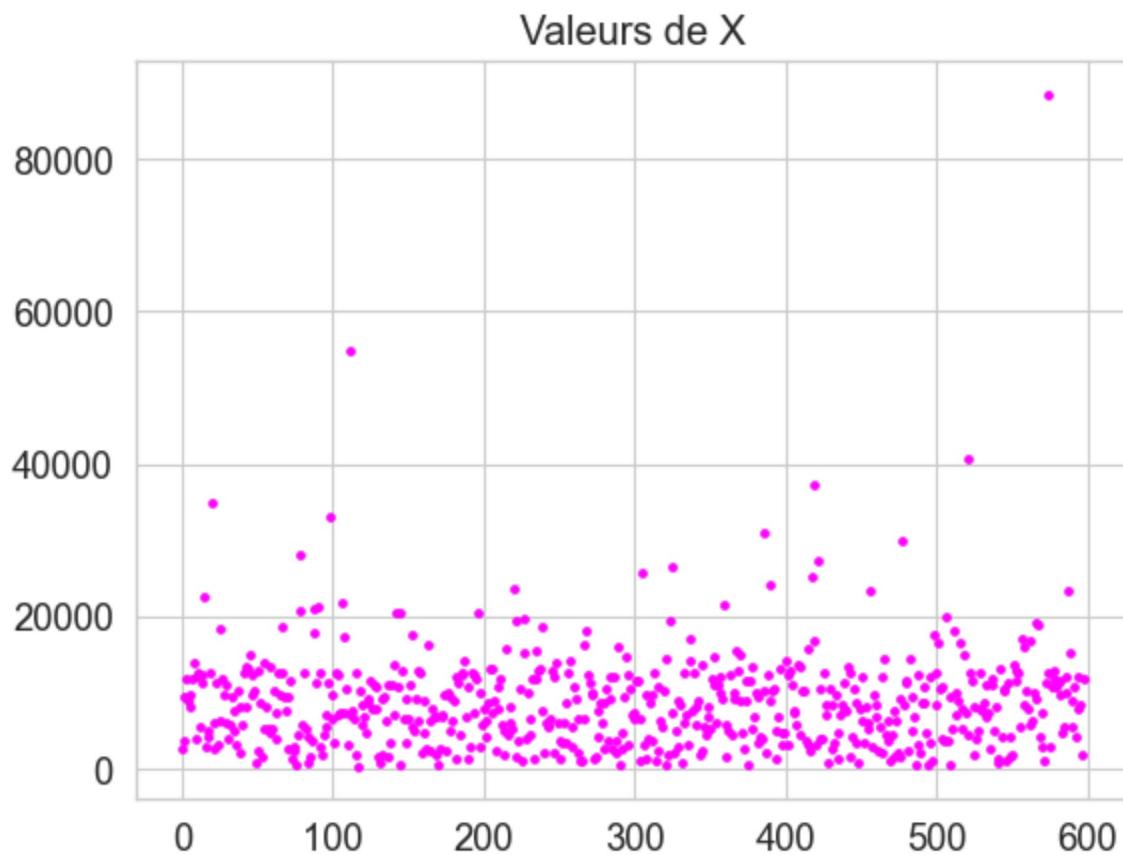
	<b>date</b>	<b>X</b>	<b>Y</b>
<b>0</b>	1974-03-31	2668.362506	2382.317298
<b>1</b>	1974-04-30	3637.960810	2401.965035
<b>2</b>	1974-05-31	9615.228633	2713.505324
<b>3</b>	1974-06-30	11921.839672	3156.486354
<b>4</b>	1974-07-31	9016.528952	2894.837030

## Visualisation des données pour la série X et pour la série Y

Ensuite, nous avons visualisé les séries X et Y de manière indépendante afin de mieux les appréhender.

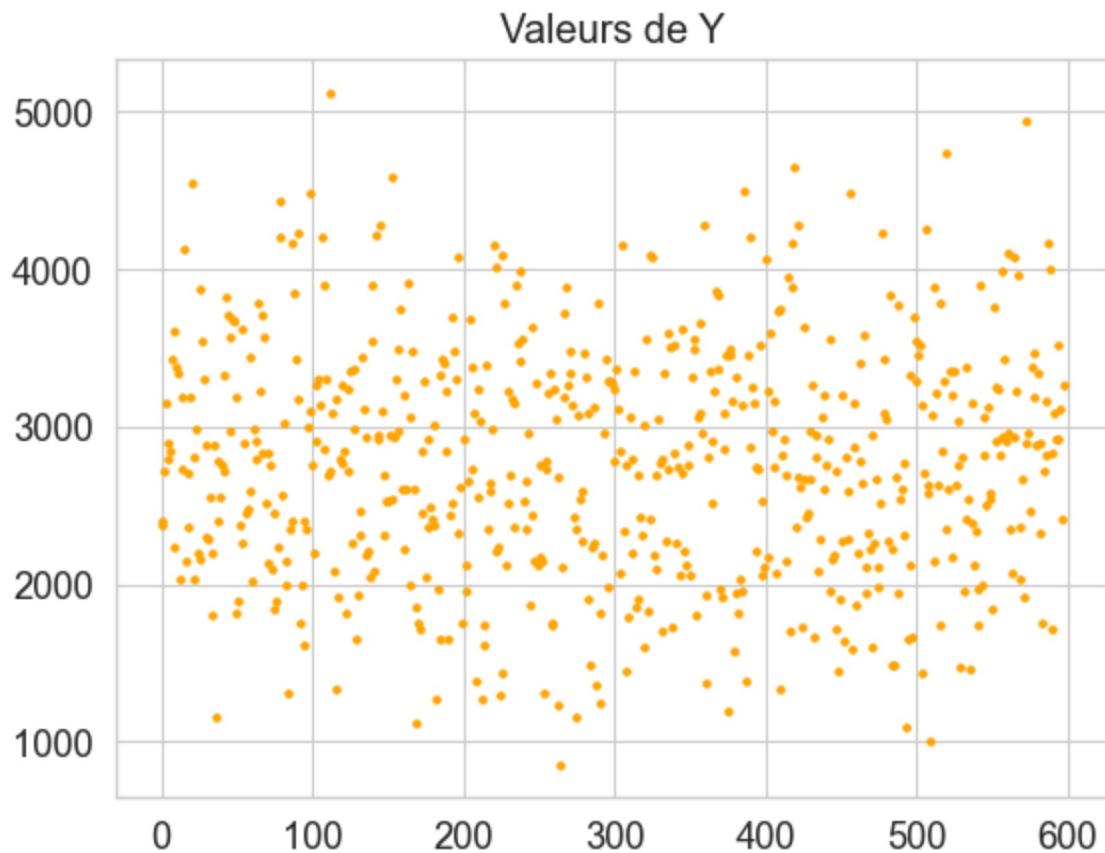
### Visualisation des données pour la série X

```
In [64]: plt.plot(range(len(x_wo_date)), x_wo_date, '.', color='magenta')
plt.title('Valeurs de X')
plt.show()
```



Visualisation des données pour la série Y

```
In [65]: plt.scatter(range(len(y_wo_date)), y_wo_date, marker='.', color='orange')
plt.title('Valeurs de Y')
plt.show()
```



## Analyse statistique rapide des données de la série X et de la série Y

Cette visualisation a été complétée par une analyse statistique rapide des deux séries, nous permettant ainsi de mieux comprendre les séries X et Y.

### Analyse rapide des données de la série X

```
In [66]: print("Analyse rapide des données de la série X")
print(f"taille de l'échantillon = {len(data)}")
print(f"moyenne = {np.mean(data['X'])}")
print(f"écart-type = {np.std(data['X'])}")
print(f"valeur minimale = {np.min(data['X'])}")
print(f"valeur maximale = {np.max(data['X'])}")
print(f"kurtosis = {stats.kurtosis(data['X'])}")
print(f"skewness = {stats.skew(data['X'])}")
```

```
Analyse rapide des données de la série X
taille de l'échantillon = 598
moyenne = 8836.30171997055
écart-type = 6854.673107261386
valeur minimale = 416.19587625048445
valeur maximale = 88523.0
kurtosis = 34.31058501208395
skewness = 3.910184513195929
```

### Analyse rapide des données de la série Y

```
In [67]: print("Analyse rapide des données de la série Y")
print(f"taille de l'échantillon = {len(data)}")
print(f"moyenne = {np.mean(data['Y'])}")
print(f"écart-type = {np.std(data['Y'])}")
print(f"valeur minimale = {np.min(data['Y'])}")
print(f"valeur maximale = {np.max(data['Y'])}")
print(f"kurtosis = {stats.kurtosis(data['Y'])}")
print(f"skewness = {stats.skew(data['Y'])}")
```

```
Analyse rapide des données de la série Y
taille de l'échantillon = 598
moyenne = 2772.6653105386995
écart-type = 749.1155088671603
valeur minimale = 856.1455866054935
valeur maximale = 5123.995393511563
kurtosis = -0.2755030336281865
skewness = 0.1123036080532401
```

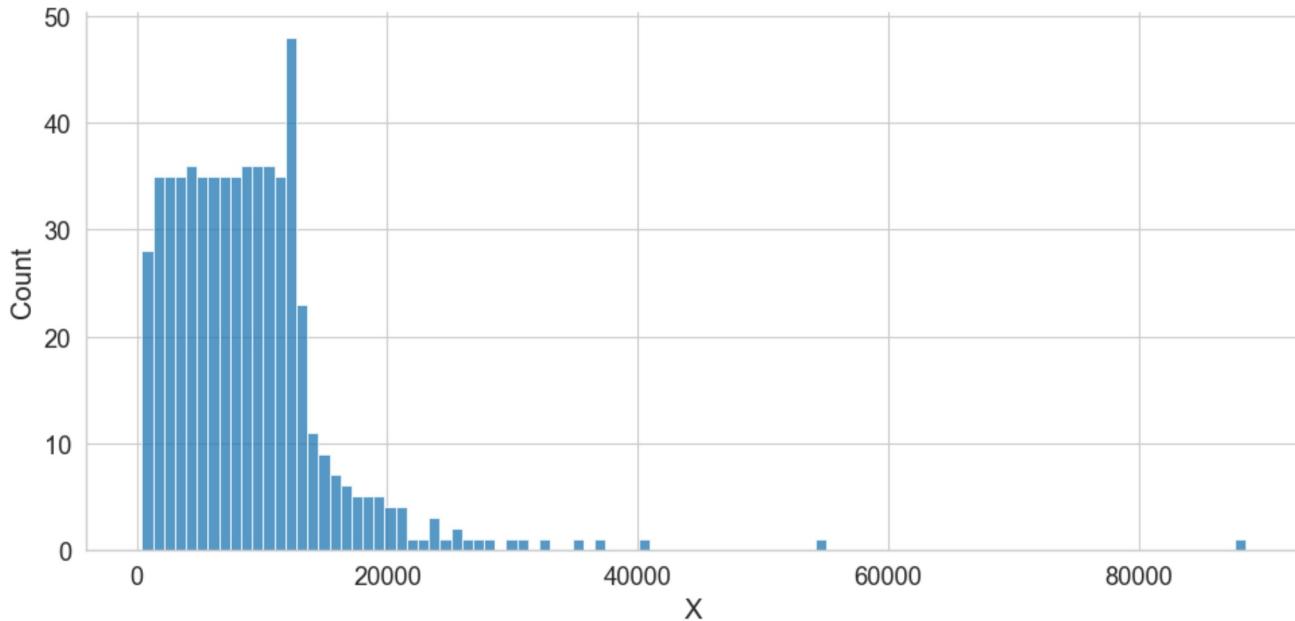
## Distribution de la série X et de la série Y

Ensuite, nous avons étudié les distributions de chaque série. Cette étape nous a permis de nous faire une première idée des lois qui pourraient convenir pour modéliser les séries X et Y.

### Distribution de la série X

```
In [68]: sns.set_style('whitegrid')
sns.set_context("paper", font_scale = 1.5)
sns.displot(data=data, x="X", kind="hist", bins = 100, aspect = 2)
```

```
Out[68]: <seaborn.axisgrid.FacetGrid at 0x1f3c42f4cd0>
```

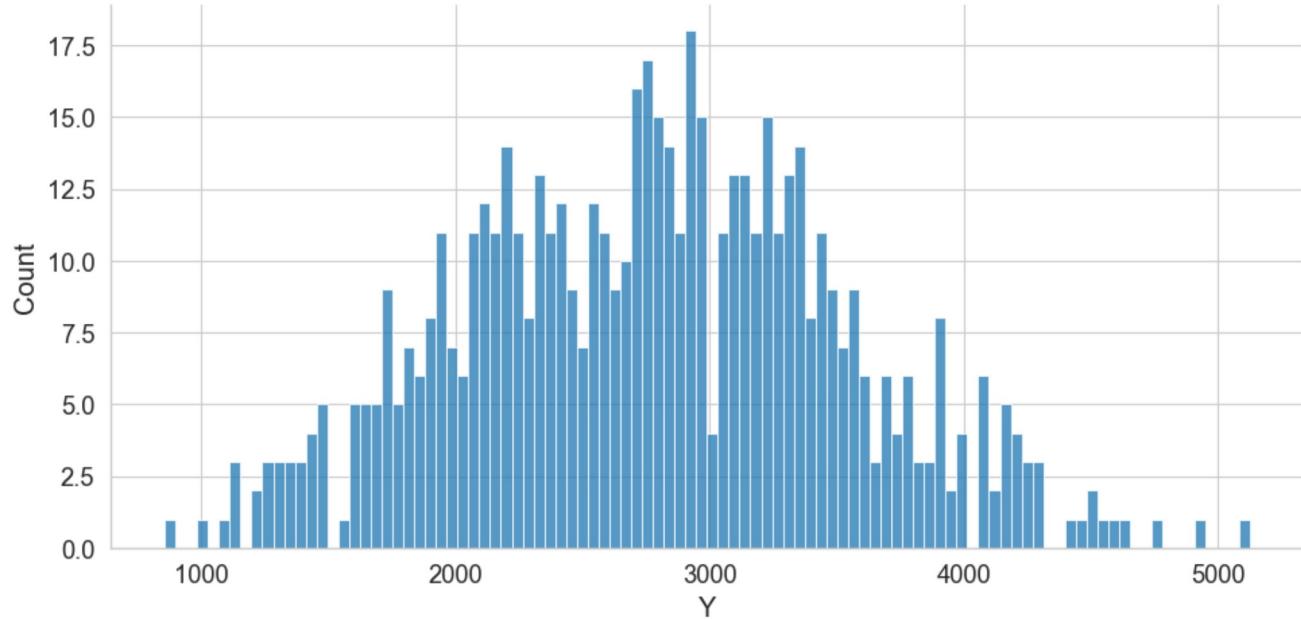


A première vue, la distribution de la série X ne ressemble pas à celle de la loi normale. On suppose donc qu'elle sera modélisée par une loi GEV ou GPD.

## Distribution de la série Y

```
In [69]: sns.set_style('whitegrid')
sns.set_context("paper", font_scale = 1.5)
sns.displot(data=data, x="Y", kind="hist", bins = 100, aspect = 2)
```

```
Out[69]: <seaborn.axisgrid.FacetGrid at 0x1f3c41a2d30>
```



A première vue, la série Y a une distribution similaire à celle de la loi normale.

## Estimation empirique du quantile 99.5% de la série X

Enfin, nous avons réalisé une estimation empirique du quantile à 99,5 % pour la série X.

```
In [70]: quantile_empirique = data_x['X'].quantile(0.995)
print(f"Le quantile 99.5% de la série X est : {quantile_empirique}")
```

```
Le quantile 99.5% de la série X est : 37399.492674568464
```

## Modélisation de la série X

### Modélisation de la série X par la loi Normale

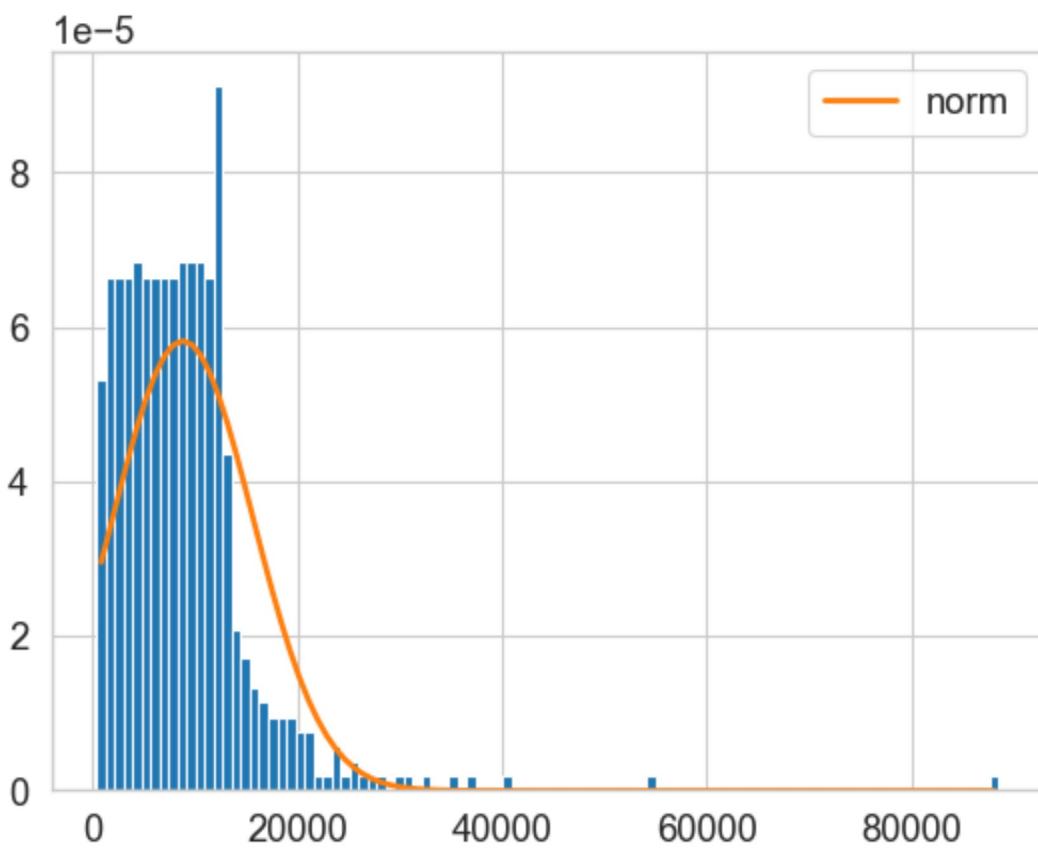
Nous allons commencer par modéliser la série X en utilisant la loi normale.

### Distribution de la série X modélisée par la loi Normale

```
In [71]: model = Fitter(data_x['X'], distributions=["norm"])
model.fit()
model.summary()
```

Out[71]:

	<b>sumsquare_error</b>	<b>aic</b>	<b>bic</b>	<b>kl_div</b>	<b>ks_statistic</b>	<b>ks_pvalue</b>
<b>norm</b>	8.534303e-09	6033.296684	6042.083865	inf	0.125104	1.287769e-08



#### Analyse des résultats obtenus :

- **sumsquare\_error :**

Somme des erreurs quadratiques entre la distribution ajustée (ici loi normale) et les données réelles. Ici la valeur est très faible, ce qui indique que la distribution normale correspond bien aux données.

- **aic (Akaike Information Criterion) :**

Critère utilisé pour évaluer la qualité de l'ajustement d'un modèle tout en pénalisant les modèles complexes. Plus la valeur de l'AIC est faible, meilleur est le compromis entre qualité d'ajustement et complexité.

- **bic (Bayesian Information Criterion) :**

Critère similaire à l'AIC, mais avec une pénalisation plus forte pour les modèles complexes. Plus le BIC est faible, meilleur est le modèle.

- **kl\_div (Kullback-Leibler Divergence) :**

Mesure de divergence entre la distribution ajustée et la distribution empirique des données. Elle

quantifie la "distance" entre deux distributions. Une divergence infinie (infinit) indique que la distribution ajustée ne capture pas correctement certains aspects des données.

- **ks-statistic (Kolmogorov-Smirnov Statistic) :**

Statistique de Kolmogorov-Smirnov, qui mesure la distance maximale entre la fonction de répartition empirique des données et celle de la distribution ajustée. Plus cette valeur est proche de 0, plus la distribution ajustée correspond bien aux données. Dans notre cas, cette valeur indique qu'il existe une certaine divergence entre la distribution normale et les données.

- **ks\_pvalue :**

P-valeur associée au test de Kolmogorov-Smirnov. Dans notre cas, la p-valeur est très faible, cela indique que les données ne correspondent pas parfaitement à la distribution normale.

Le graphe obtenu représente une comparaison entre l'histogramme empirique des données et la densité théorique d'une distribution normale ajustée. Nous pouvons bien voir que la loi normale n'est pas en adéquation avec les données de la série X étant donné que la courbe orange ne capture pas les valeurs extrêmes situées dans la queue à droite.

## Test de Kolmogorov-Smirnov

La modélisation que nous venons d'effectuer nous fournit la valeur de plusieurs paramètres afin d'estimer l'adéquation du modèle aux données. Parmi ces données nous pouvons retrouver les résultats du test de Kolmogorov-Smirnov (ks\_statistic et ks\_pvalue).

Ces deux données nous permettent de dire que la loi normale n'est pas adaptée pour modéliser la série X.

## Somme des moindres carrés

La somme des moindres carrés dans le cadre d'un test statistique mesure l'écart total entre les valeurs observées et les valeurs prédictes par un modèle, en sommant les carrés de ces écarts pour quantifier l'erreur globale d'ajustement.

Ici la valeur est très faible, ce qui indique que la distribution normale correspond assez bien aux données. Cependant, cette donnée n'est pas suffisante pour conclure l'adéquation de la loi normale pour la modélisation de la série X. Elle est même contradictoire avec le résultat du test de Kolmogorov-Smirnov.

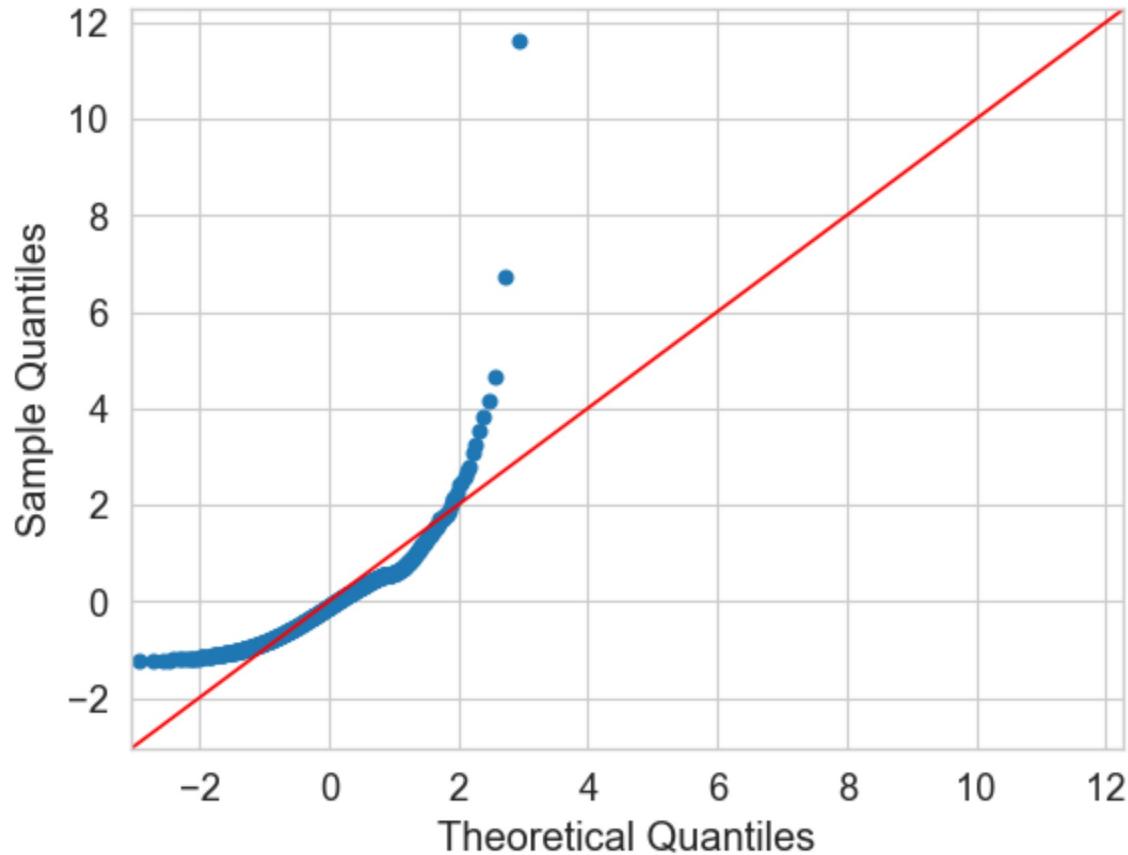
## Tracé du QQ-plot

Un QQ-plot (Quantile-Quantile plot) est un graphique qui permet de comparer les quantiles d'une distribution empirique (ceux du jeu de données) avec les quantiles d'une distribution théorique (ici la loi normale). Il est utilisé pour évaluer visuellement si les données suivent une distribution

particulière, comme une distribution normale, une loi GEV ou une loi GPD.

Sur le QQ-plot, si les points s'alignent sur la première bissectrice, cela indique que la distribution empirique correspond bien à la distribution théorique. Les déviations par rapport à cette ligne fournissent des informations importantes : des écarts dans les extrémités (queues) suggèrent que les données ne respectent pas bien les hypothèses de la distribution pour les valeurs extrêmes.

```
In [72]: sm.qqplot(x_wo_date, fit=True, line='45', dist=stats.norm)
plt.show()
```



On remarque que les points dévient de la ligne droite donc les données ne suivent pas la distribution normale.

### Tracé du PP-plot

Un PP-plot (Probability-Probability plot) est un graphique qui permet de comparer les probabilités cumulées empiriques d'un ensemble de données avec les probabilités cumulées théoriques prévues par un modèle de distribution (ici la loi normale).

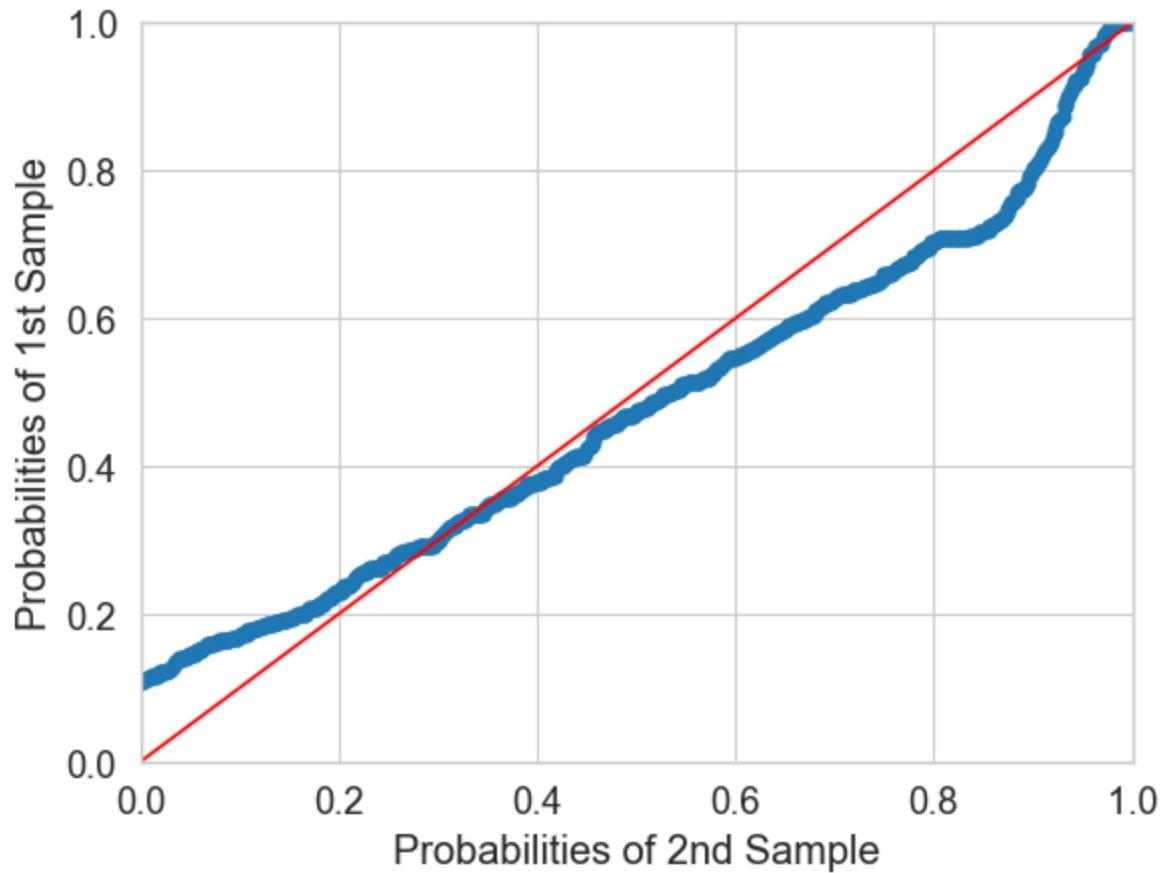
Sur ce graphique, l'axe des abscisses représente les probabilités théoriques (issues de la fonction de répartition cumulative de la distribution ajustée), tandis que l'axe des ordonnées représente les probabilités empiriques (celles du jeu de données).

Si les données suivent bien la distribution théorique, les points du PP-plot s'alignent sur la première

bissectrice. Les déviations par rapport à cette ligne révèlent les écarts entre les probabilités théoriques et empiriques, ce qui permet de détecter des problèmes d'ajustement du modèle, en particulier pour les queues de la distribution (événements extrêmes).

In [73]:

```
# Modéliser La Loi - Paramètres de la loi
dist = getattr(scipy.stats, "norm")
param = dist.fit(x_wo_date)
## moyenne
loc = param[-2]
## écart-type
scale = param[-1]
arg = param[:-2]
Y = np.random.normal(loc=loc, scale=scale, size=len(x_wo_date))
pp_x = sm.ProbPlot(x_wo_date, fit=True)
pp_y = sm.ProbPlot(Y, fit=True)
fig = pp_x.ppplot(line='45', other=pp_y)
plt.show()
```



### Analyse du graphe

1st sample = données réelles // 2nd sample = données simulées par la loi normale

On remarque que les points ne sont pas alignés sur la première bissectrice donc les données ne suivent pas une loi normale.

Estimation de la valeur quantile 99.5% de la série X

```
In [74]: mu, sigma = stats.norm.fit(x_wo_date)
quantile_norm = stats.norm.ppf(0.995, loc=mu, scale=sigma)
print(f"Le quantile 99.5% estimé avec la loi normale est : {quantile_norm}")
```

Le quantile 99.5% estimé avec la loi normale est : 26492.76957590302

## Modélisation de la série X par le modèle GEV

Le modèle GEV (Generalized Extreme Value) est une distribution statistique utilisée pour modéliser les valeurs extrêmes d'un ensemble de données, comme les maximums ou minimums observés sur des périodes fixes (il y a donc une dépendance en fonction de la durée de la période considérée). Il combine trois familles de distributions de valeurs extrêmes : Gumbel, Fréchet et Weibull, grâce à un paramètre de forme  $\xi$ .

$$\begin{aligned} - G_{\xi,\mu,\sigma}(t) &= e^{-\left[1+\xi\left(\frac{t-\mu}{\sigma}\right)\right]_+^{-1/\xi}} \text{ si } \xi \neq 0 \\ - G_{0,\mu,\sigma}(t) &= e^{-e^{-\left(\frac{t-\mu}{\sigma}\right)}} \end{aligned}$$

✓ Notation :  $[x]_+ = \max(0; x)$

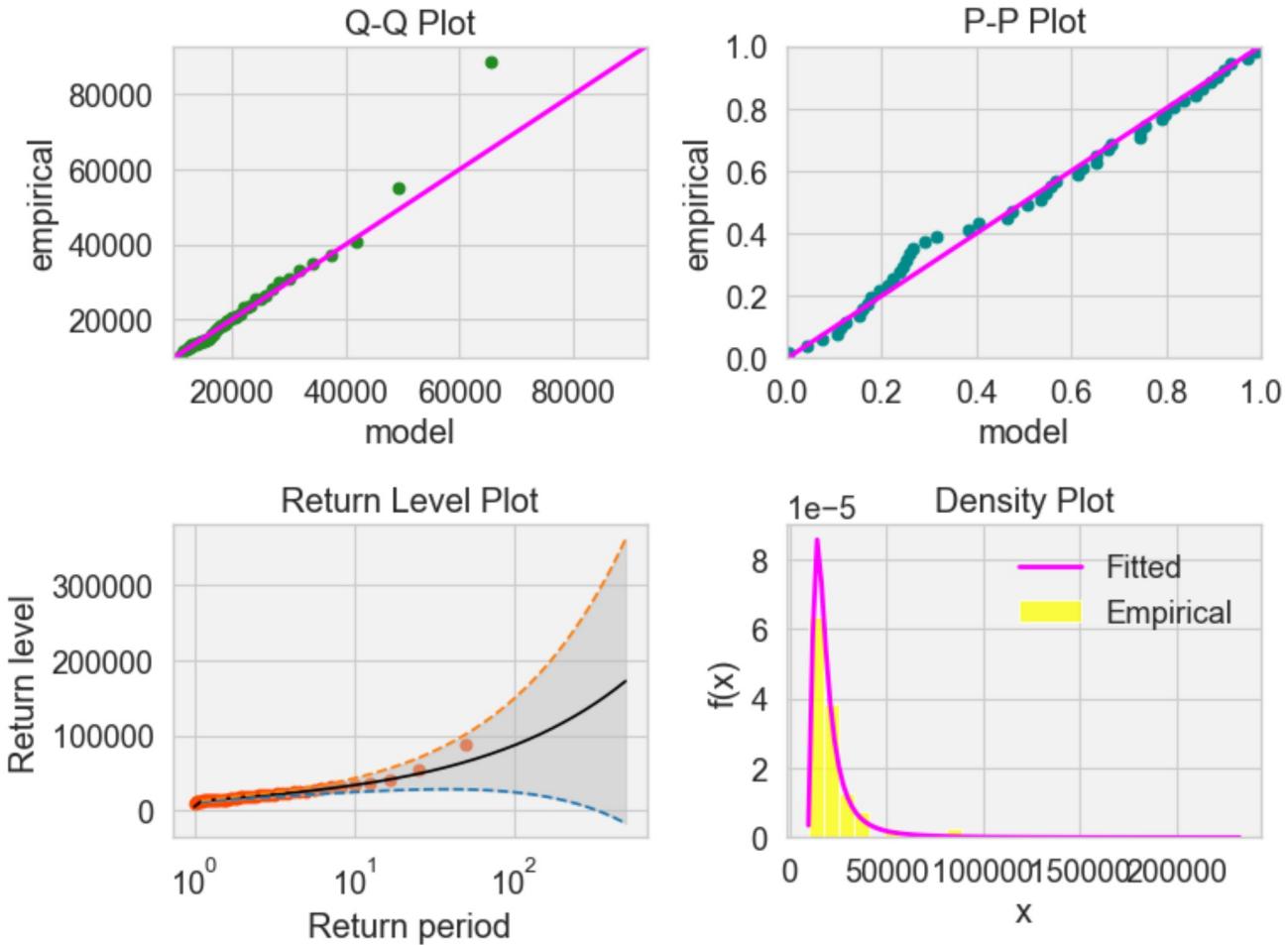
### 1. Modèle GEV en fonction des années

QQ-plot, PP-plot, Density plot, Return level plot

```
In [75]: ## regrouper les données de la série X par années
group_by_Year = data.date.dt.to_period("Y")
## sélectionner la date maximale par année
max_Year = data.groupby(group_by_Year).max()
Year = np.array(max_Year['X'])
## On utilise la méthode de vraisemblance pour estimer les paramètres avec un intervalle
model = ske.models.classic.GEV(Year, fit_method = 'mle', ci = 0.05, ci_method = 'delta')
## paramètre de queue (eta)
shape= model.params["shape"]
## mu
loc= model.params["location"]
## sigma
scale= model.params["scale"]
## Etude statistique (à comparer à celle effectuée au départ sur le jeu de données)
print("\nEtude statistique: \n nombre d'observations = {} \n moyenne = {} \n écart-type = {} \n indice de queue = {} \n min = {} \n max = {} \n kurt = {} \n".format(len(Year), np.mean(Year), np.std(Year), shape,
np.min(Year), np.max(Year), stats.kurtosis(Year), stats.skew(Year)))
model.plot_summary()
plt.show()
```

Etude statistique:

```
nombre d'observations = 50
moyenne = 21529.91351305563
ecart-type = 12892.223369252215
indice de queue = -0.44493782740769644
min = 10121.799600187425
max = 88523.0
kurtosis = 12.757113366235451
skewness = 3.2025969366362435
```



Valeur du quantile 99.5% avec la loi GEV par année

```
In [76]: quantile_GEV_Year = genextreme.ppf(0.995, c=shape, loc=loc, scale=scale)
print(f"Le quantile 99.5% estimé avec la loi GEV par année est : {quantile_GEV_Year}")
```

Le quantile 99.5% estimé avec la loi GEV par année est : 116298.01458543929

Test de Komogorov-Smirnov et somme des moindres carrés

```
In [77]: # Test de Kolmogorov-Smirnov
## Définir la fonction de répartition cumulée (CDF) de la Loi GEV ajustée
cdf_gev = lambda x: genextreme.cdf(x_wo_date, c=shape, loc=loc, scale=scale)

## Calculer la statistique et la p-valeur du test KS
ks_statistic, ks_pvalue = kstest(x_wo_date, cdf_gev)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")
```

```

# Calcul de la somme des moindres carrés
## Histogramme empirique des données et densité théorique ajustée
pdf_empirical, bins = np.histogram(x_wo_date, bins=20, density=True)
bin_centers = (bins[:-1] + bins[1:]) / 2 # Centres des bins
pdf_theoretical = genextreme.pdf(bin_centers, c=shape, loc=loc, scale=scale)

# Calcul de la somme des moindres carrés
sse = np.sum((pdf_empirical - pdf_theoretical) ** 2)
print(f"Somme des moindres carrés : {sse}")

```

Statistique KS : 0.9983277591973244  
P-valeur KS : 0.0  
Somme des moindres carrés : 1.5025615783766423e-08

## 2. Modèle GEV en fonction des semestres

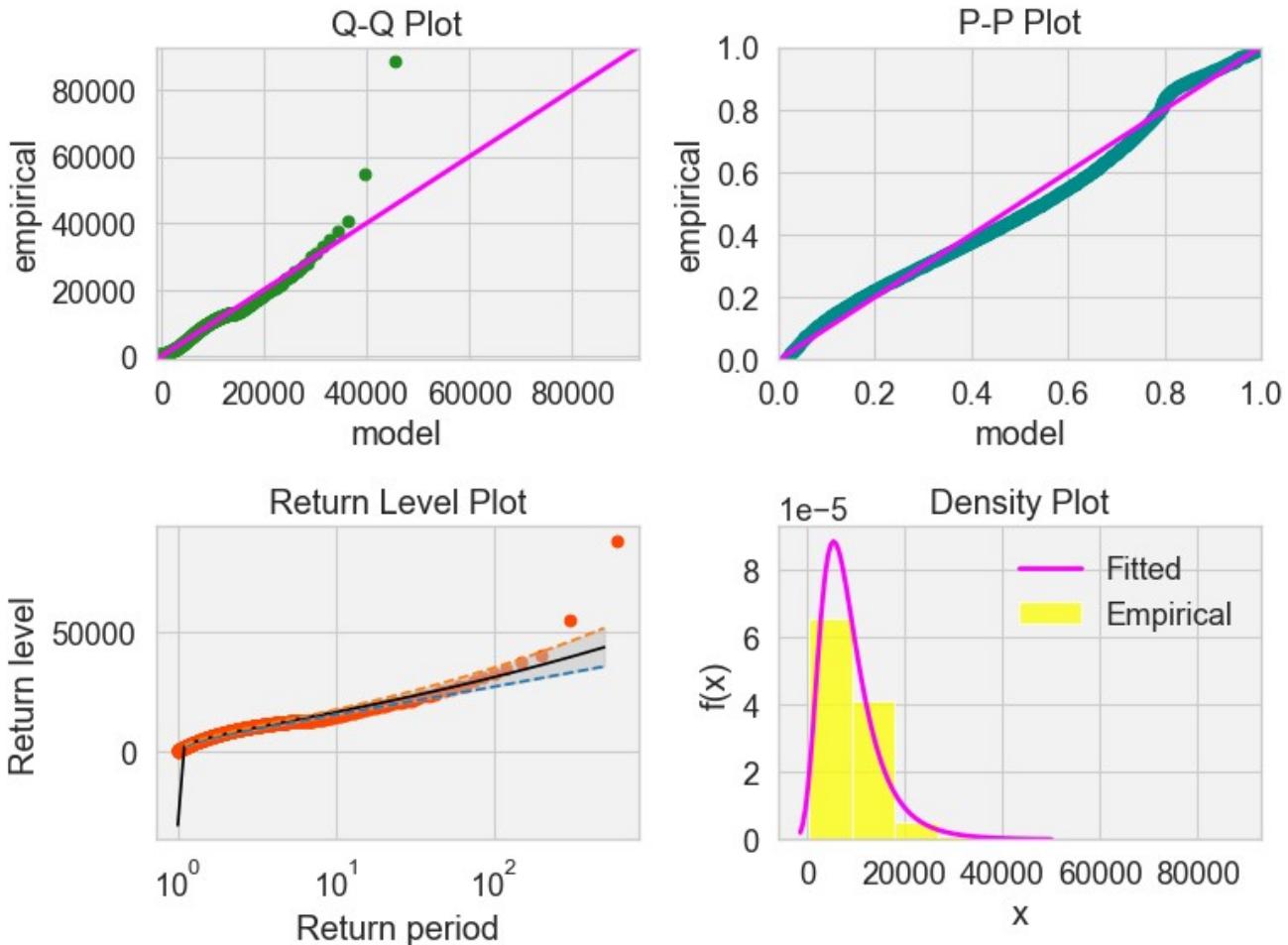
QQ-plot, PP-plot, Density plot, Return level plot

```

In [78]: ## regrouper les données de la série X par semestre
group_by_sem = data.date.dt.to_period("6M")
## sélectionner la date maximale par semestre
max_sem = data.groupby(group_by_sem).max()
Sem = np.array(max_sem['X'])
## On utilise la méthode de vraisemblance pour estimer les paramètres avec un intervalle
model = ske.models.classic.GEV(Sem, fit_method = 'mle', ci = 0.05, ci_method = 'delta')
## paramètre de queue (eta)
shape= model.params["shape"]
## mu
loc= model.params["location"]
## sigma
scale= model.params["scale"]
## Etude statistique (à comparer à celle effectuée au départ sur le jeu de données)
print("\nEtude statistique: \n nombre d'observations = {} \
\n moyenne = {} \n ecart-type = {} \n indice de queue = {} \n min = {} \n max = {} \n kurt \
    .format(len(Sem), np.mean(Sem), np.std(Sem), shape, \
        np.min(Sem), np.max(Sem), stats.kurtosis(Sem), stats.skew(Sem)))
model.plot_summary()
plt.show()

```

Etude statistique:  
nombre d'observations = 598  
moyenne = 8836.30171997055  
ecart-type = 6854.673107261387  
indice de queue = -0.1143616264456188  
min = 416.19587625048445  
max = 88523.0  
kurtosis = 34.310585012083926  
skewness = 3.9101845131959276



### Analyse des graphes obtenus

Sur le QQ-plot, les points suivent globalement la diagonale (ligne rose), ce qui indique un bon ajustement du modèle pour une grande partie des données. Cependant, on observe une légère déviation pour les valeurs extrêmes (à droite du graphe), ce qui suggère que le modèle pourrait sous-estimer ces valeurs.

Sur le PP-plot, la courbe suit assez bien la diagonale (ligne rose), ce qui montre que le modèle capture correctement les probabilités cumulées sur toute la plage des données. Cela confirme un ajustement globalement satisfaisant.

Sur le Return Level Plot, les points orange représentent les données empiriques, et les courbes correspondent aux modèles ajustés. Le modèle semble bien capturer les tendances générales des niveaux de retour pour les périodes courtes à moyennes. Pour les longues périodes de retour (droite du graphe), il y a une divergence entre les points empiriques et les modèles, ce qui peut indiquer une sous-estimation des événements extrêmes par le modèle.

Sur le density Plot, la courbe ajustée (rose) capture bien la forme générale de la distribution des données empiriques. Cependant, on observe une sous-représentation de la densité empirique dans les queues (parties les plus extrêmes), ce qui peut indiquer que le modèle ne capture pas parfaitement les événements rares ou extrêmes.

Valeur du quantile 99.5% avec la loi GEV par semestre

```
In [79]: quantile_GEV_Sem = genextreme.ppf(0.995, c=shape, loc=loc, scale=scale)
print(f"Le quantile 99.5% estimé avec la loi GEV par semestre est : {quantile_GEV_Sem}")
```

Le quantile 99.5% estimé avec la loi GEV par semestre est : 36429.71879172335

Test de Komogorov-Smirnov et somme des moindres carrés

```
In [80]: # Test de Kolmogorov-Smirnov
## Définir la fonction de répartition cumulée (CDF) de la loi GEV ajustée
cdf_gev = lambda x: genextreme.cdf(x_wo_date, c=shape, loc=loc, scale=scale)

## Calculer la statistique et la p-valeur du test KS
ks_statistic, ks_pvalue = kstest(x_wo_date, cdf_gev)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

# Calcul de la somme des moindres carrés
## Histogramme empirique des données et densité théorique ajustée
pdf_empirical, bins = np.histogram(x_wo_date, bins=20, density=True)
bin_centers = (bins[:-1] + bins[1:]) / 2 # Centres des bins
pdf_theoretical = genextreme.pdf(bin_centers, c=shape, loc=loc, scale=scale)

# Calcul de la somme des moindres carrés
sse = np.sum((pdf_empirical - pdf_theoretical) ** 2)
print(f"Somme des moindres carrés : {sse}")
```

Statistique KS : 0.9605112513037483

P-valeur KS : 0.0

Somme des moindres carrés : 7.877879703386002e-10

### 3. Modèle GEV en fonction des trimestres

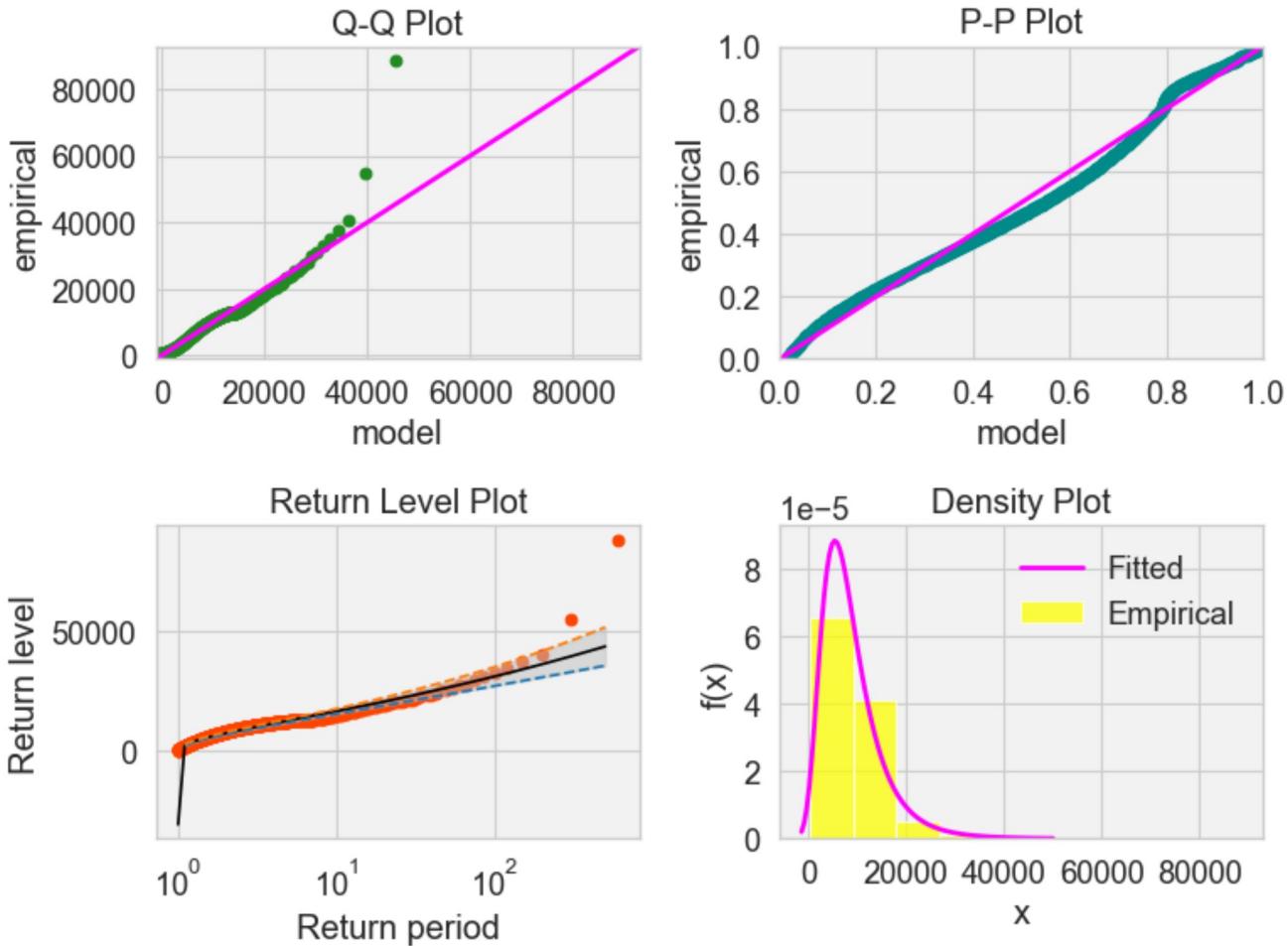
QQ-plot, PP-plot, Density plot, Return level plot

```
In [81]: ## regrouper les données de la série X par trimestre
group_by_tri = data.date.dt.to_period("3M")
## sélectionner la date maximale par semestre
max_tri = data.groupby(group_by_tri).max()
Tri = np.array(max_tri['X'])
## On utilise la méthode de vraisemblance pour estimer les paramètres avec un intervalle
model = ske.models.classic.GEV(Tri, fit_method = 'mle', ci = 0.05, ci_method = 'delta')
## paramètre de queue (eta)
shape= model.params["shape"]
## mu
loc= model.params["location"]
## sigma
scale= model.params["scale"]
## Etude statistique (à comparer à celle effectuée au départ sur le jeu de données)
print("\nEtude statistique: \n nombre d'observations = {} \
\n moyenne = {} \n ecart-type = {} \n indice de queue = {} \n min = {} \n max = {} \n kurt \
.format(len(Tri), np.mean(Tri), np.std(Tri), shape, \
np.min(Tri), np.max(Tri), stats.kurtosis(Tri), stats.skew(Tri)))
model.plot_summary()
```

```
plt.show()
```

Etude statistique:

```
nombre d'observations = 598
moyenne = 8836.30171997055
ecart-type = 6854.673107261387
indice de queue = -0.1143616264456188
min = 416.19587625048445
max = 88523.0
kurtosis = 34.310585012083926
skewness = 3.9101845131959276
```



Valeur du quantile 99.5% avec la loi GEV par trimestre

```
In [82]: quantile_GEV_Tri = genextreme.ppf(0.995, c=shape, loc=loc, scale=scale)
print(f"Le quantile 99.5% estimé avec la loi GEV par trimestre est : {quantile_GEV_Tri}")
```

Le quantile 99.5% estimé avec la loi GEV par trimestre est : 36429.71879172335

Test de Komogorov-Smirnov et somme des moindres carrés

```
In [83]: # Test de Kolmogorov-Smirnov
## Définir la fonction de répartition cumulée (CDF) de la Loi GEV ajustée
cdf_gev = lambda x: genextreme.cdf(x_wo_date, c=shape, loc=loc, scale=scale)

## Calculer la statistique et la p-valeur du test KS
ks_statistic, ks_pvalue = kstest(x_wo_date, cdf_gev)
```

```

print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

# Calcul de la somme des moindres carrés
## Histogramme empirique des données et densité théorique ajustée
pdf_empirical, bins = np.histogram(x_wo_date, bins=20, density=True)
bin_centers = (bins[:-1] + bins[1:]) / 2 # Centres des bins
pdf_theoretical = genextreme.pdf(bin_centers, c=shape, loc=loc, scale=scale)

# Calcul de la somme des moindres carrés
sse = np.sum((pdf_empirical - pdf_theoretical) ** 2)
print(f"Somme des moindres carrés : {sse}")

```

```

Statistique KS : 0.9605112513037483
P-valeur KS : 0.0
Somme des moindres carrés : 7.877879703386002e-10

```

## Modélisation de la série X par le modèle GPD

Le modèle GPD (Generalized Pareto Distribution) est une distribution utilisée pour modéliser les excès au-delà d'un seuil dans l'analyse des valeurs extrêmes. Il est particulièrement adapté pour décrire les queues des distributions, c'est-à-dire les événements rares et extrêmes.

$$\bar{H}_{\xi,\sigma}(y) = \begin{cases} \left(1 + \frac{\xi y}{\sigma}\right)^{-\frac{1}{\xi}} & \text{si } \xi \neq 0 \\ e^{-\frac{y}{\sigma}} & \text{sinon} \end{cases}$$

## Mean Excess Plot

```

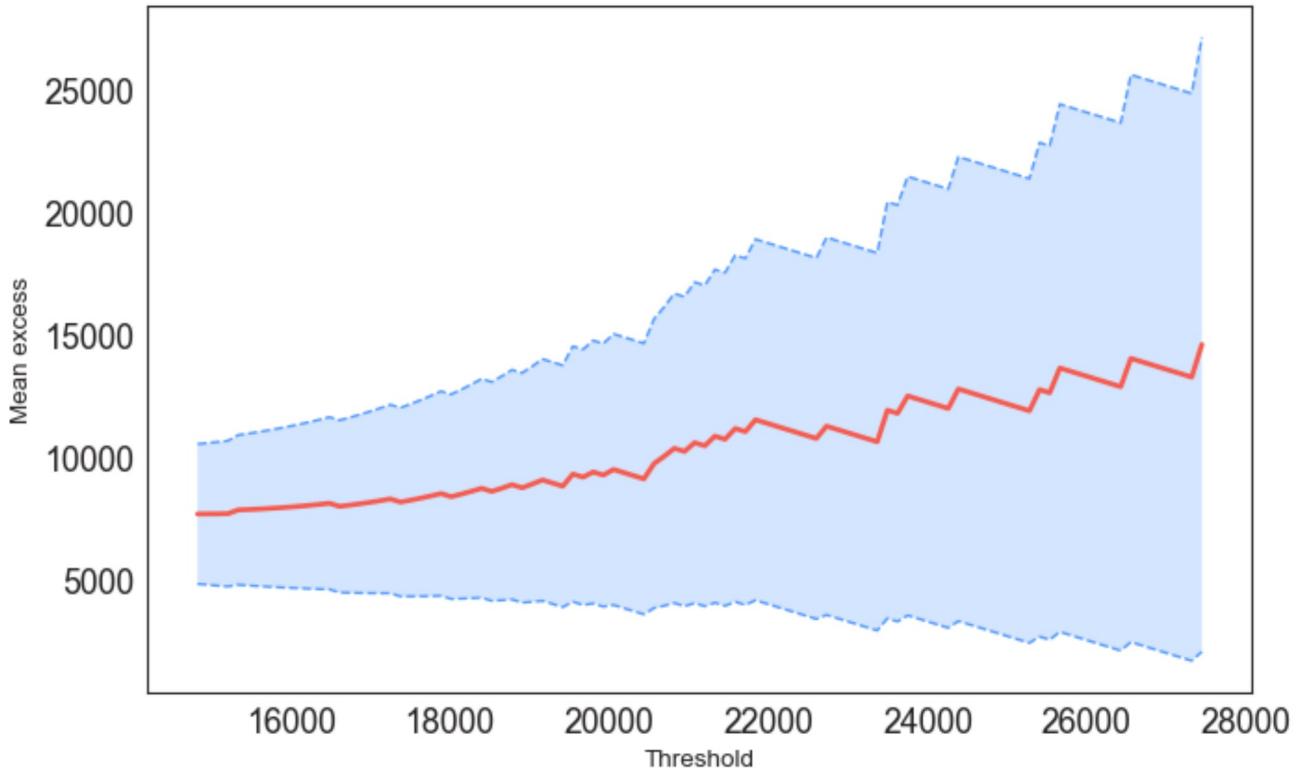
In [84]: ## Mean Excess Plot
plt.figure(figsize=(10, 6))
plot_mean_residual_life(x_wo_date)

```

```

Out[84]: <Axes: xlabel='Threshold', ylabel='Mean excess'>
<Figure size 1000x600 with 0 Axes>

```



Jusqu'à environ  $u=20\ 000$ , la moyenne conditionnelle (ligne rouge) reste relativement stable ou suit une tendance linéaire douce, avec peu de fluctuations importantes. C'est une caractéristique indiquant que les excès suivent probablement une GPD.

Après  $u=20\ 000$ , la moyenne conditionnelle devient de plus en plus instable et commence à augmenter de manière non linéaire, avec des écarts importants entre les bornes de confiance (zones bleues). Cela pourrait indiquer que le nombre de données au-delà de ces seuils devient trop faible, ce qui entraîne une incertitude plus grande.

Un bon seuil devrait se situer dans la région stable avant que l'instabilité ne commence. Ici,  $u \approx 20\ 000$  semble être un choix raisonnable. Il se situe à la transition entre la zone stable et le début des fluctuations, ce qui garantit une taille d'échantillon suffisante pour l'ajustement du modèle GPD.

## Distribution du modèle avec la loi GPD

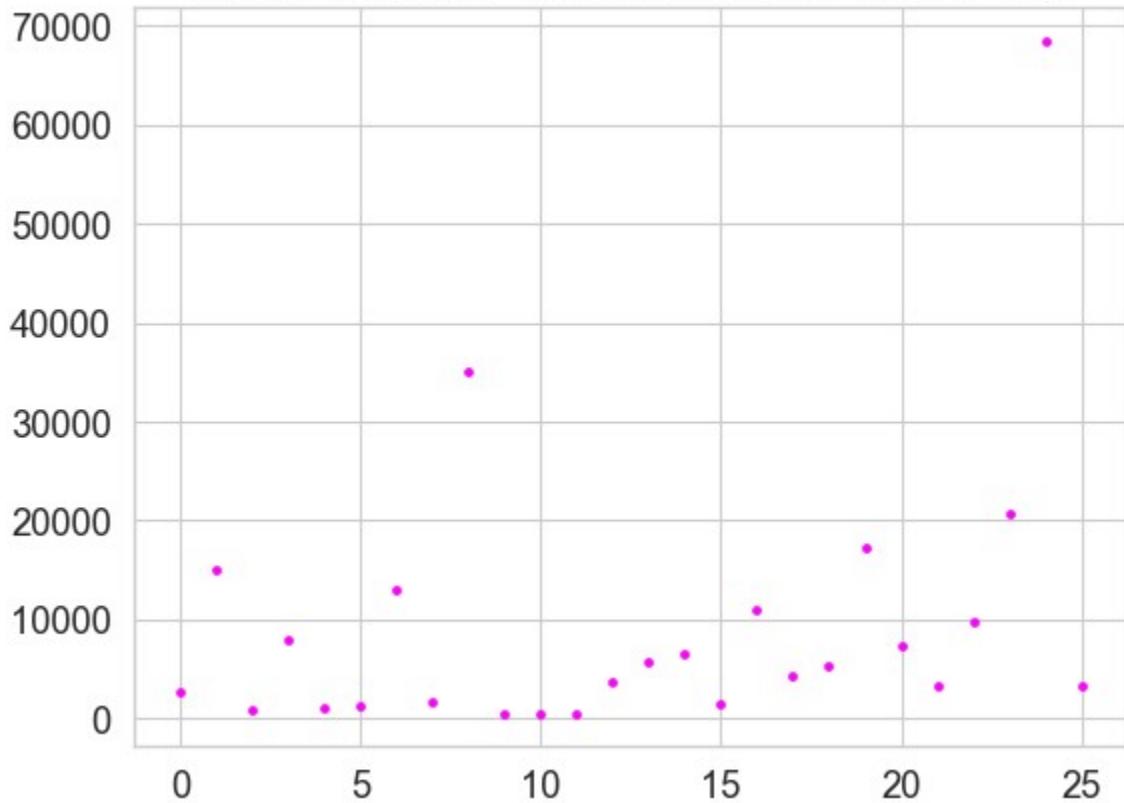
```
In [85]: # Choix du seuil
threshold = 20000
# Calcul des excès
excess = x_wo_date[x_wo_date > threshold] - threshold
if len(excess) == 0:
    raise ValueError(f"Aucune valeur dans la série X ne dépasse le seuil de {threshold}.")
# Tracé du graphe des valeurs de X prises en compte pour le modèle GPD
plt.plot(range(len(excess)), excess, '.', color='magenta')
plt.title('Valeurs de X prise en compte dans le modèle GPD')
plt.show()
# Ajustement du modèle GPD
shape, loc, scale = genpareto.fit(excess, loc=0)
print(f"Paramètre de forme (shape): {shape}")
```

```

print(f"Paramètre d'échelle (scale): {scale}")
# Histogramme des excès
plt.figure(figsize=(10, 6))
plt.hist(excess, bins=30, density=True, alpha=0.6, color='purple', label='Empirical Excess')
# Densité théorique ajustée
x_gpd = np.linspace(0, max(excess), 100)
pdf = genpareto.pdf(x_gpd, c=shape, loc=0, scale=scale)
plt.plot(x_gpd, pdf, 'orange', label='Fitted GPD')
plt.title(f"Modélisation GPD des excès (Seuil = {threshold})")
plt.xlabel("Excess over Threshold")
plt.ylabel("Density")
plt.legend()
plt.grid()
plt.show()

```

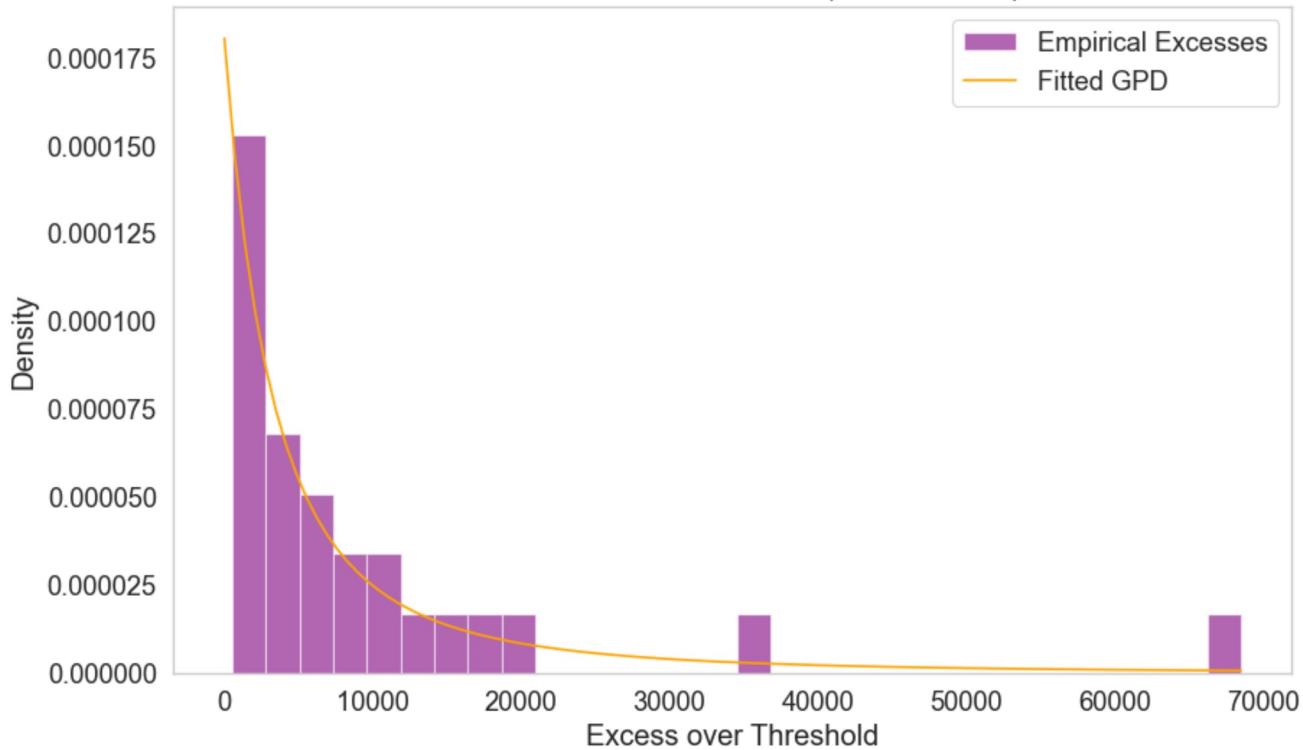
Valeurs de X prise en compte dans le modèle GPD



Paramètre de forme (shape): 0.693146944942664

Paramètre d'échelle (scale): 5542.4481857325545

### Modélisation GPD des excès (Seuil = 20000)

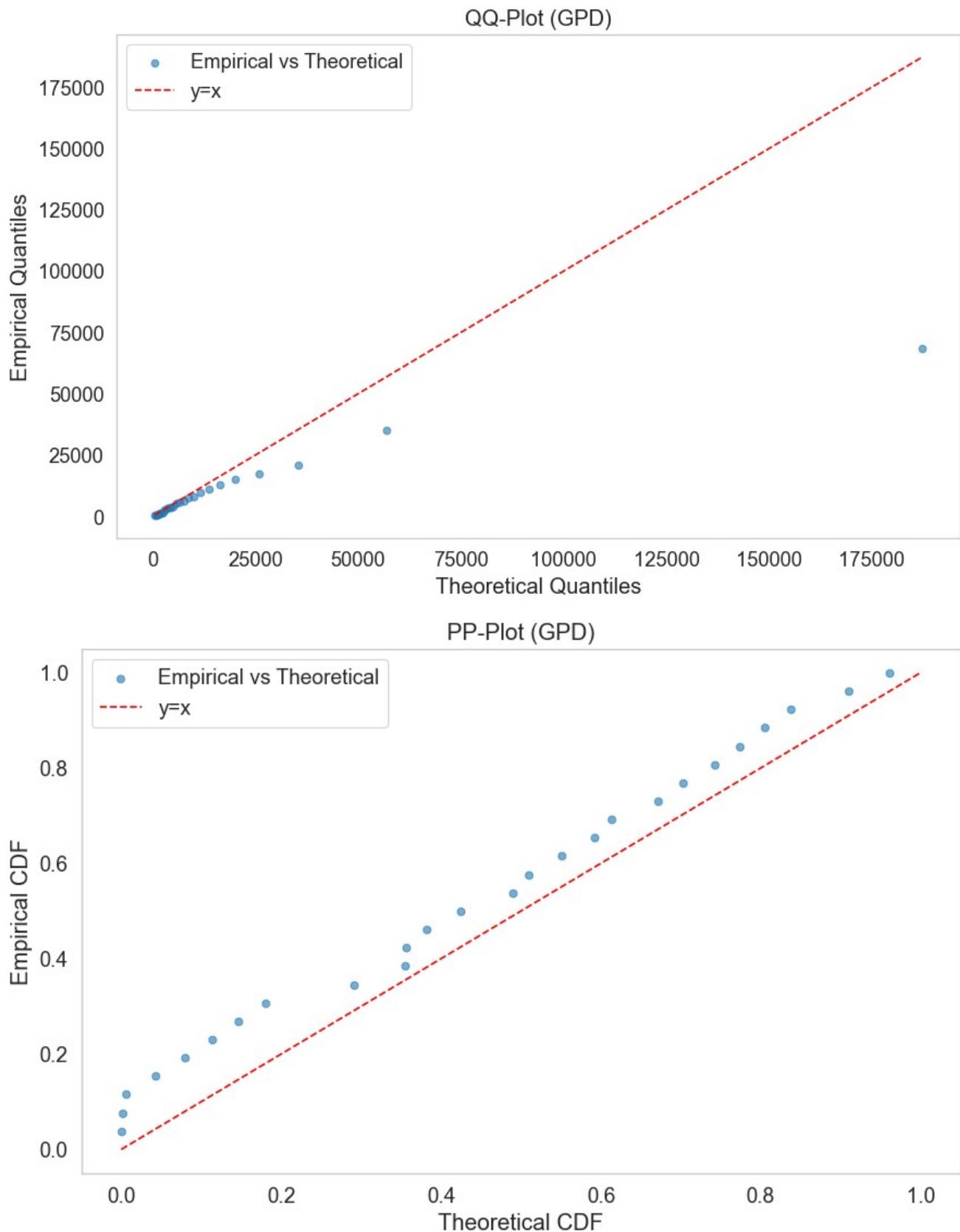


Tracé du QQ-plot et du PP-plot

In [86]:

```
# QQ-Plot
plt.figure(figsize=(10, 6))
sorted_data = np.sort(excess)
quantiles = genpareto.ppf(np.linspace(0.01, 0.99, len(sorted_data)), c=shape, loc=loc, scale=scale)
plt.scatter(quantiles, sorted_data, alpha=0.6, label="Empirical vs Theoretical")
plt.plot([min(quantiles), max(quantiles)], [min(quantiles), max(quantiles)], 'r--', label="y=x")
plt.title("QQ-Plot (GPD)")
plt.xlabel("Theoretical Quantiles")
plt.ylabel("Empirical Quantiles")
plt.legend()
plt.grid()
plt.show()

# PP-Plot
plt.figure(figsize=(10, 6))
empirical_cdf = np.arange(1, len(excess) + 1) / len(excess)
theoretical_cdf = genpareto.cdf(np.sort(excess), c=shape, loc=loc, scale=scale)
plt.scatter(theoretical_cdf, empirical_cdf, alpha=0.6, label="Empirical vs Theoretical")
plt.plot([0, 1], [0, 1], 'r--', label="y=x")
plt.title("PP-Plot (GPD)")
plt.xlabel("Theoretical CDF")
plt.ylabel("Empirical CDF")
plt.legend()
plt.grid()
plt.show()
```



Quantile 99.5%

In [87]:

```
# Calcul du quantile à 99.5%
quantile_GPD = genpareto.ppf(0.995, c=shape, loc=loc, scale=scale)
print(f"Quantile à 99.5% : {quantile_GPD}")
```

Quantile à 99.5% : 307196.29989544797

## Test de Kolmogorov-Smirnov et Somme des moindres carrés

In [88]:

```
# Test de Kolmogorov-Smirnov
ks_statistic, ks_pvalue = kstest(excess, lambda x: genpareto.cdf(x, c=shape, loc=loc, scale=scale))
print(f"KS-Statistic : {ks_statistic}")
print(f"KS P-Value : {ks_pvalue}")

# Somme des moindres carrés
pdf_empirical, bins = np.histogram(excess, bins=20, density=True)
bin_centers = (bins[:-1] + bins[1:]) / 2
pdf_theoretical = genpareto.pdf(bin_centers, c=shape, loc=loc, scale=scale)
sse = np.sum((pdf_empirical - pdf_theoretical) ** 2)
print(f"Somme des moindres carrés : {sse}")
```

KS-Statistic : 0.12662596994782008

KS P-Value : 0.7523119732931691

Somme des moindres carrés : 1.0355485869065409e-09

## Choix du modèle pour la série X

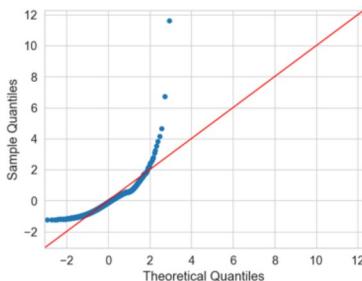


Figure : QQ-plot loi Normale série X

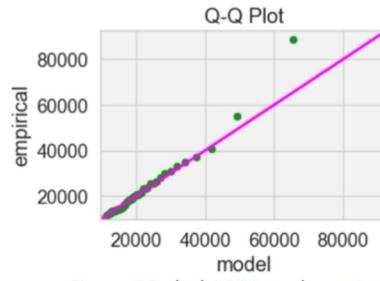


Figure : QQ-plot loi GEV pour les années pour la série X

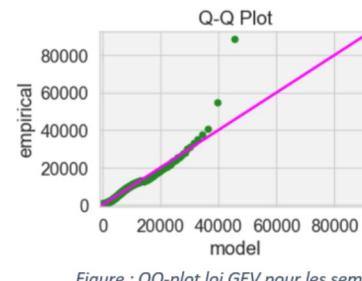


Figure : QQ-plot loi GEV pour les semestres pour la série X

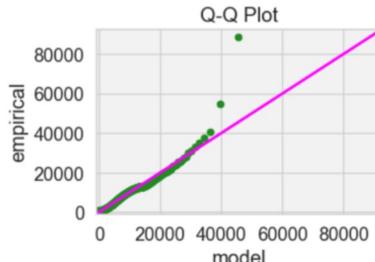


Figure : QQ-plot loi GEV pour les trimestres pour la série X

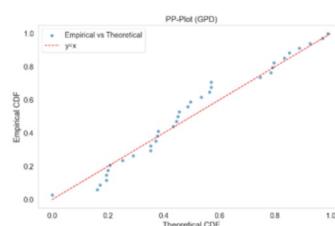


Figure : PP-Plot (GPD)

Quand on regarde ces QQ-plot, on remarque qu'il y a seulement celui de la loi GPD pour lequel les points sont assez bien alignés autour de la première bissectrice. Cela nous pousse à dire qu'il y a une bonne adéquation entre la série X et la loi GPD.

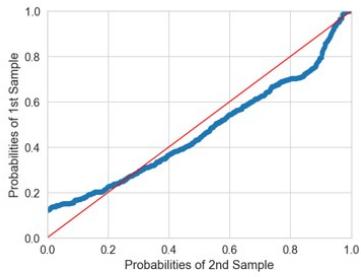


Figure : PP-plot loi Normale pour la série X

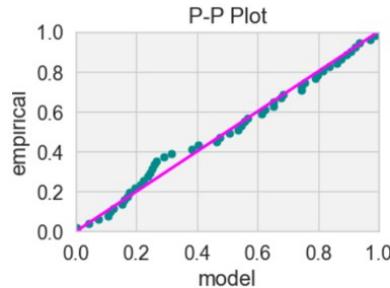


Figure : PP-plot loi GEV pour les années pour la série X

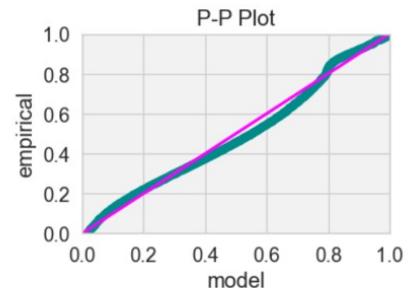


Figure : PP-plot pour les semestres pour la série X

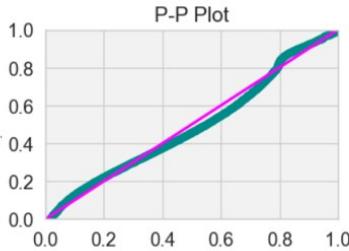


Figure : PP-plot pour les trimestres pour la série X

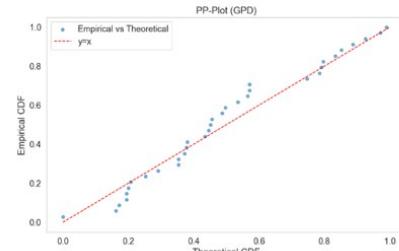


Figure : PP-plot loi GPD pour la série X

En observant ces PP-plot, on remarque que les PP-plots des lois GEV et de la loi GPD présentent des points alignés avec la première bissectrice. On en conclut donc que les lois GEV et la loi GPD pourraient être des bonnes lois pour modéliser la série X.

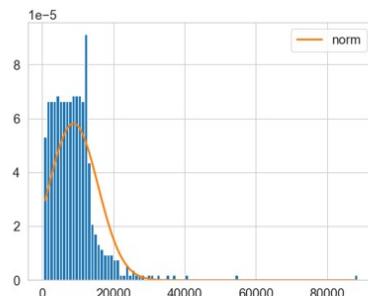


Figure : loi Normale pour la série X

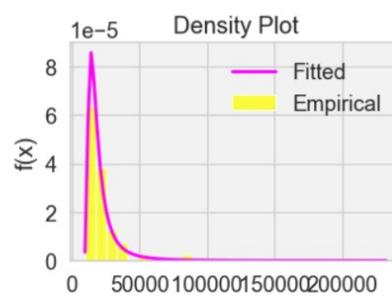


Figure : loi GEV pour les années pour la série X

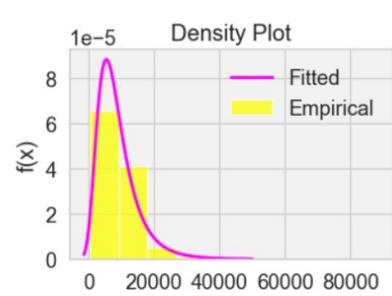


Figure : loi GEV pour les semestres pour la série X

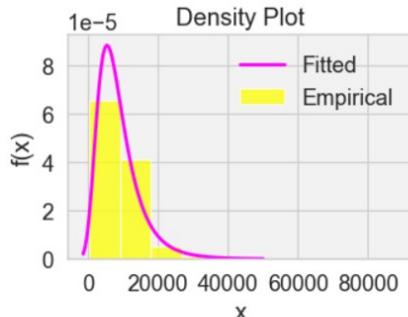


Figure : loi GEV pour les trimestres pour la série X

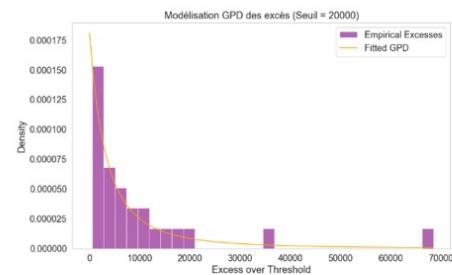


Figure : loi GPD pour la série X

La loi normale est clairement inappropriée pour modéliser X, car elle ne capture ni l'asymétrie ni les queues lourdes. La loi GEV (en particulier pour les périodes annuelles et semestrielles) est bien adaptée pour modéliser la série X dans son ensemble. Cependant, la loi GPD est plus adaptée pour modéliser les valeurs extrêmes de X au-delà du seuil de 20 000.

	Loi Normale	Loi GEV par année	Loi GEV par semestre	Loi GEV par trimestre	Loi GPD
Statistique KS	0.125	0.998	0.961	0.961	0.126
P-value	1.25 <sup>e</sup> .8	0.0	0.0	0.0	0.752
Somme des moindres carrés	8.53 <sup>e</sup> -9	1.50 <sup>e</sup> -8	7.88 <sup>e</sup> -10	7.88 <sup>e</sup> -10	1.04 <sup>e</sup> -9

Pour la loi normale, bien que la statistique KS soit faible, la p-valeur indique que la loi normale ne convient pas pour modéliser la série X.

Les statistiques KS et p-valeurs sont très défavorables pour toutes les versions de la loi GEV. Bien que la somme des moindres carrés est faible pour les versions semestrielles et trimestrielles, cela ne compense pas les mauvais résultats des tests KS.

La p-valeur de la loi GPD est significative, ce qui indique que la loi est statistiquement valide pour modéliser la série X.

La loi GPD est la plus adaptée pour modéliser la série X, car elle est la seule loi avec une p-valeur significative et des métriques globalement favorables. Elle capture les caractéristiques des valeurs extrêmes tout en respectant les hypothèses statistiques.

## Calcul de la perte annuelle de période de retour 200 ans pour la série X

Pour calculer la perte annuelle de X pour différents niveaux de retour, nous procédons en plusieurs étapes en utilisant la loi GPD (Generalized Pareto Distribution). Tout d'abord, les périodes de retour sont définies (par exemple, 25, 50, 100, 200, 500 ans). Chaque période de retour correspond à une probabilité d'occurrence calculée comme étant égale à  $p = 1 - 1/T$ , où T est la période de retour. Cette probabilité est utilisée pour déterminer les quantiles de la loi GPD, qui représentent les niveaux de retour associés à ces périodes.

Le calcul des niveaux de retour se fait avec la fonction `genpareto.ppf` de la bibliothèque `scipy.stats`. Cette fonction permet de calculer le quantile correspondant à une probabilité donnée, en utilisant les paramètres ajustés de la GPD. Une fois le quantile obtenu, le seuil est ajouté au résultat pour obtenir le niveau de retour réel.

In [125...]

```
# Définir les périodes de retour
return_periods = np.array([25, 50, 100, 200, 500])

# Calculer les niveaux de retour en utilisant genpareto
```

```

return_levels = []
for period in return_periods:
    prob = 1 - 1 / period
    level = threshold + genpareto.ppf(prob, c=shape, loc=0, scale=scale) # Fonction quantile
    return_levels.append(level)

# Afficher les niveaux de retour
print("Niveaux de retour (GPD)")
print("Période Niveau")
for period, level in zip(return_periods, return_levels):
    print(f'{period:4.0f} {level:10.2f}')

```

Niveaux de retour (GPD)

Période	Niveau
25	4865.35
50	4996.08
100	5111.91
200	5214.52
500	5332.42

## Modélisation de la série Y

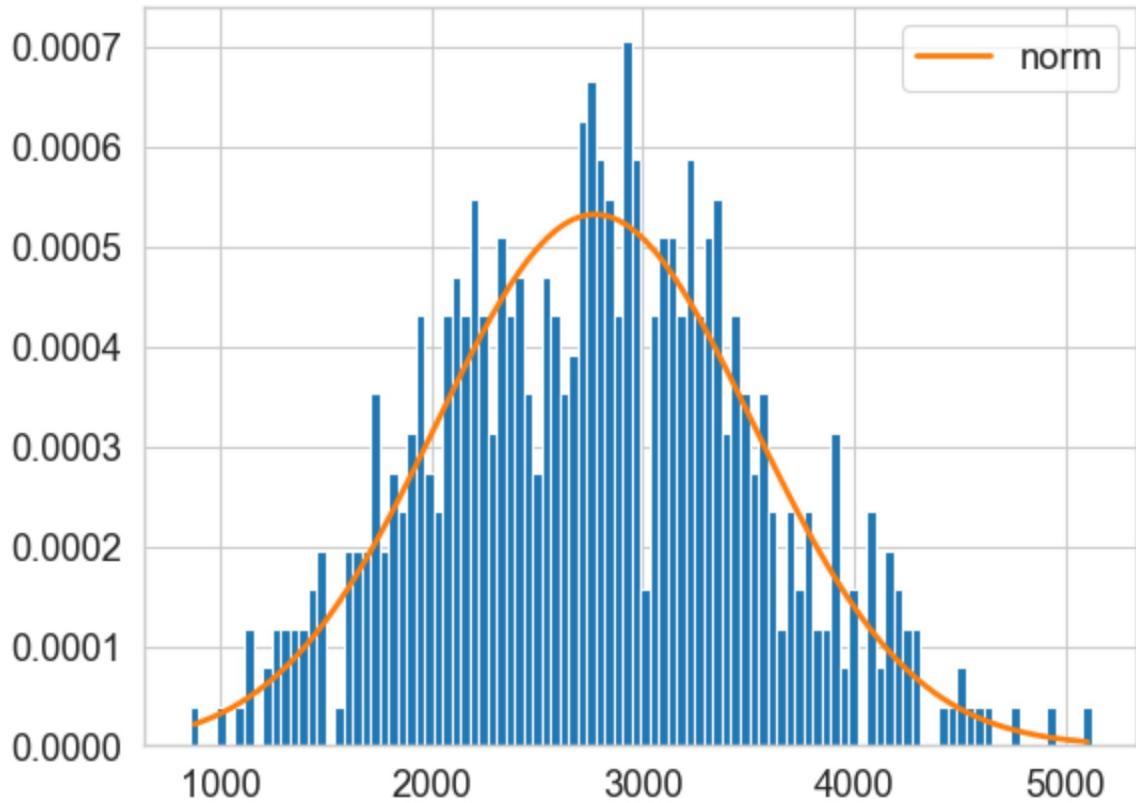
### Modélisation de la série Y par la loi Normale

Distribution de la série Y modélisée par la loi Normale

```
In [90]: model = Fitter(data_y['Y'], distributions=['norm'])
model.fit()
model.summary()
```

```
Out[90]:
```

	sumsquare_error	aic	bic	kl_div	ks_statistic	ks_pvalue
<b>norm</b>	7.078389e-07	1790.444481	1799.231663	inf	0.027375	0.750552



### Test de Kolmogorov-Smirnov

La modélisation que nous venons d'effectuer nous fournit la valeur de plusieurs paramètres afin d'estimer l'adéquation du modèle aux données. Parmi ces données nous pouvons retrouver les résultats du test de Kolmogorov-Smirnov (ks\_statistic et ks\_pvalue).

Ces deux données nous permettent de dire que la loi normale est adaptée pour modéliser la série Y.

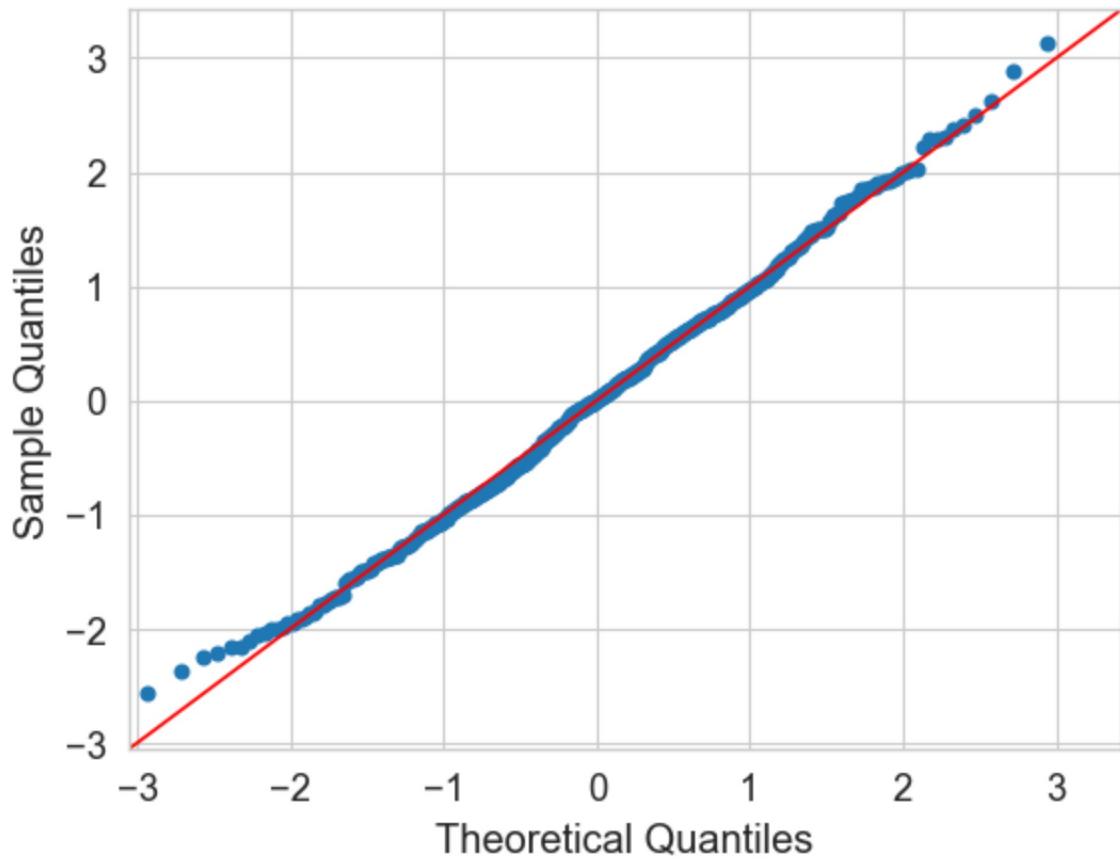
### Somme des moindres carrés

La somme des moindres carrés dans le cadre d'un test statistique mesure l'écart total entre les valeurs observées et les valeurs prédites par un modèle, en sommant les carrés de ces écarts pour quantifier l'erreur globale d'ajustement.

Ici la valeur est très faible, ce qui indique que la distribution normale correspond assez bien aux données. Cette donnée confirme donc le résultat du test de Kolmogorov-Smirnov.

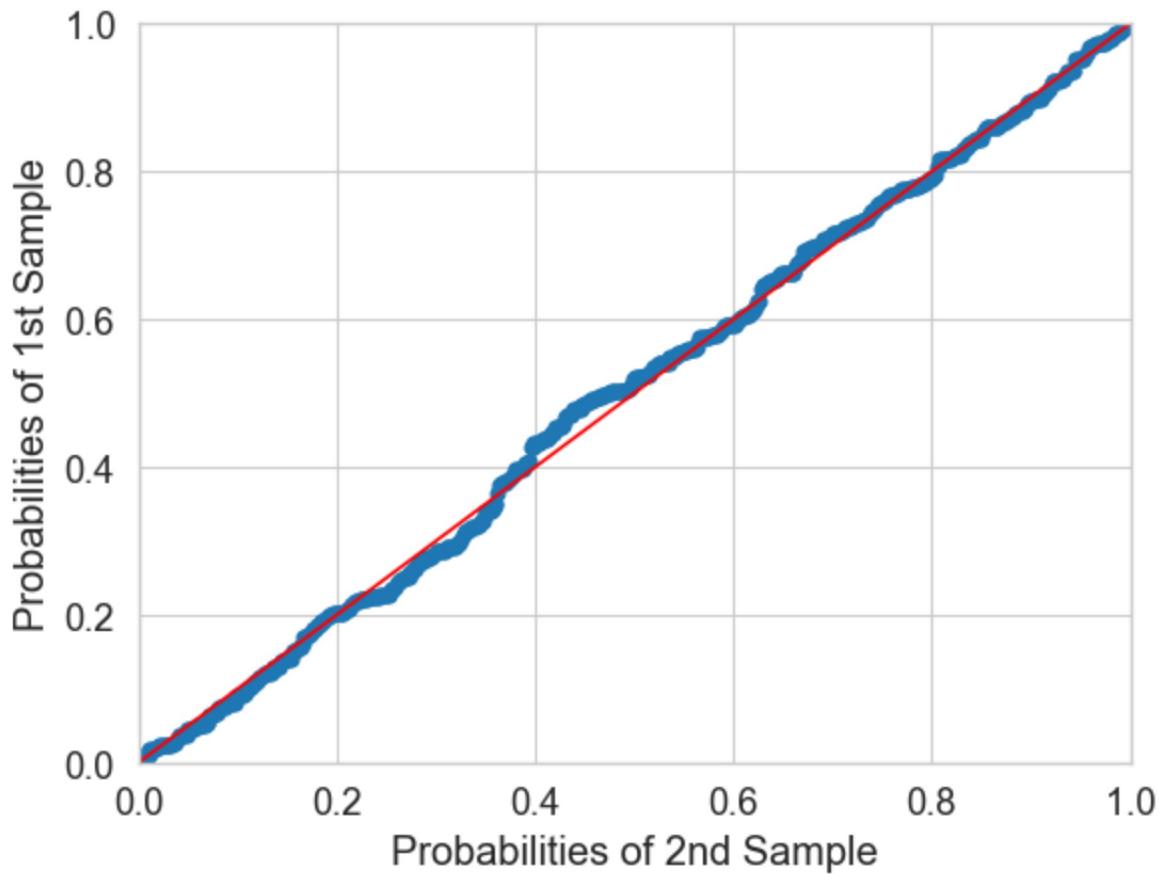
### Tracé du QQ-Plot

```
In [91]: sm.qqplot(y_wo_date, fit=True, line='45', dist=stats.norm)
plt.show()
```



Tracé du PP-plot

```
In [92]: # Modéliser la loi - Paramètres de la loi
dist = getattr(scipy.stats, "norm")
param = dist.fit(y_wo_date)
## moyenne
loc = param[-2]
## écart-type
scale = param[-1]
Y = np.random.normal(loc=loc, scale=scale, size=len(y_wo_date))
pp_x = sm.ProbPlot(y_wo_date, fit=True)
pp_y = sm.ProbPlot(Y, fit=True)
fig = pp_x.ppplot(line='45', other=pp_y)
plt.show()
```



Estimation de la valeur du quantile 99.5% de la série Y

```
In [93]: mu, sigma = stats.norm.fit(y_wo_date)
quantile_norm = stats.norm.ppf(0.995, loc=mu, scale=sigma)
print(f"Le quantile 99.5% estimé avec la loi normale est : {quantile_norm}")
```

Le quantile 99.5% estimé avec la loi normale est : 4702.258990021677

### Modélisation de la série Y par la loi GEV

1. Modèle GEV en fonction des années

QQ-plot, PP-plot, Density plot, Return level plot

```
In [94]: ## regrouper les données de la série Y par années
group_by_Year = data.date.dt.to_period("Y")
## sélectionner la date maximale par année
max_Year = data.groupby(group_by_Year).max()
Year = np.array(max_Year['Y'])
## On utilise la méthode de vraisemblance pour estimer les paramètres avec un intervalle
model = ske.models.classic.GEV(Year, fit_method = 'mle', ci = 0.05, ci_method = 'delta')
## paramètre de queue (eta)
shape= model.params["shape"]
## mu
loc= model.params["location"]
## sigma
scale= model.params["scale"]
```

```

## Etude statistique (à comparer à celle effectuée au départ sur le jeu de données)
print("\nEtude statistique: \n nombre d'observations = {} \
\n moyenne = {} \n ecart-type = {} \n indice de queue = {} \n min = {} \n max = {} \n kurt \
.format(len(Year), np.mean(Year), np.std(Year), shape, \
np.min(Year), np.max(Year), stats.kurtosis(Year), stats.skew(Year)))")
model.plot_summary()
plt.show()

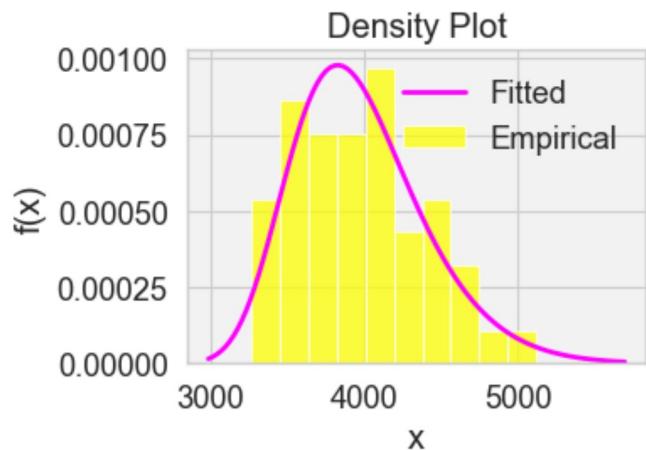
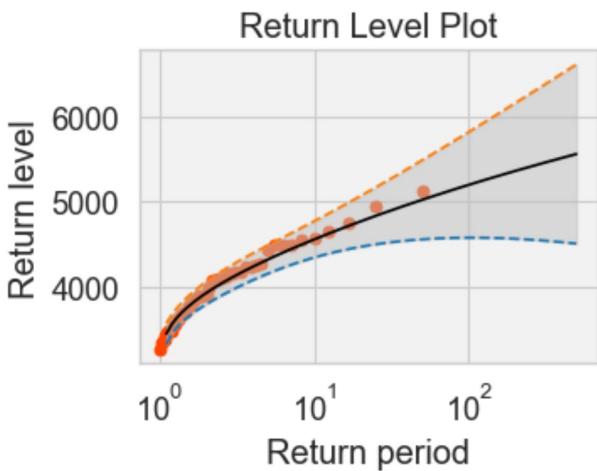
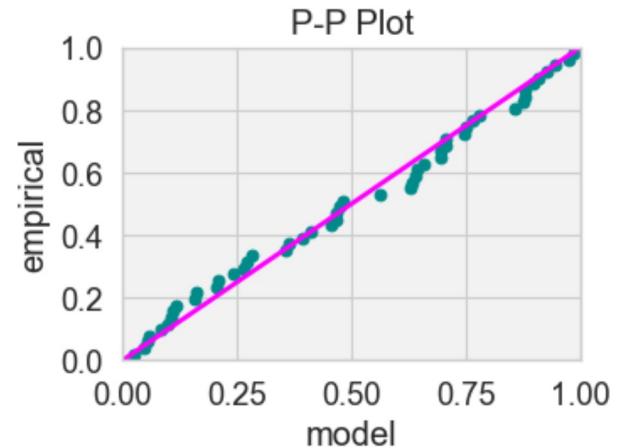
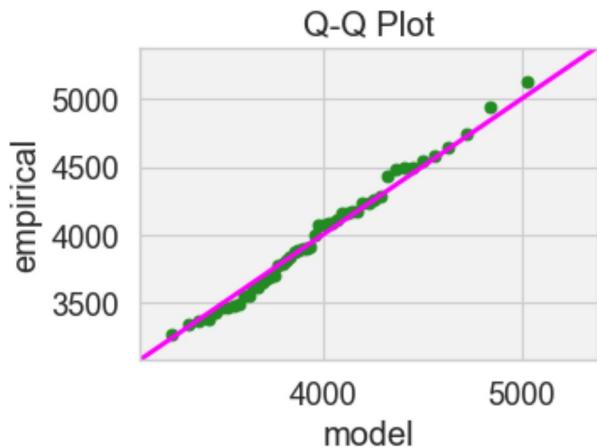
```

Etude statistique:

```

nombre d'observations = 50
moyenne = 3980.967156611694
ecart-type = 436.1461589542328
indice de queue = 0.096485103989056
min = 3266.7069717638474
max = 5123.995393511563
kurtosis = -0.3656455408853305
skewness = 0.47697439816213405

```



Valeur du quantile 99.5% avec la loi GEV par année

```
In [95]: quantile_GEV_Year = genextreme.ppf(0.995, c=shape, loc=loc, scale=scale)
print(f"Le quantile 99.5% estimé avec la loi GEV par année est : {quantile_GEV_Year}")
```

Le quantile 99.5% estimé avec la loi GEV par année est : 5359.48380672228

Test de Kolmogorov-Smirnov et somme des moindres carrés

```
In [96]: # Test de Kolmogorov-Smirnov
## Définir la fonction de répartition cumulée (CDF) de La Loi GEV ajustée
cdf_gev = lambda x: genextreme.cdf(y_wo_date, c=shape, loc=loc, scale=scale)

## Calculer la statistique et la p-valeur du test KS
ks_statistic, ks_pvalue = kstest(y_wo_date, cdf_gev)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

# Calcul de la somme des moindres carrés
## Histogramme empirique des données et densité théorique ajustée
pdf_empirical, bins = np.histogram(y_wo_date, bins=20, density=True)
bin_centers = (bins[:-1] + bins[1:]) / 2 # Centres des bins
pdf_theoretical = genextreme.pdf(bin_centers, c=shape, loc=loc, scale=scale)

# Calcul de la somme des moindres carrés
sse = np.sum((pdf_empirical - pdf_theoretical) ** 2)
print(f"Somme des moindres carrés : {sse}")
```

Statistique KS : 0.9983277590865622  
P-valeur KS : 0.0  
Somme des moindres carrés : 3.322268315767011e-06

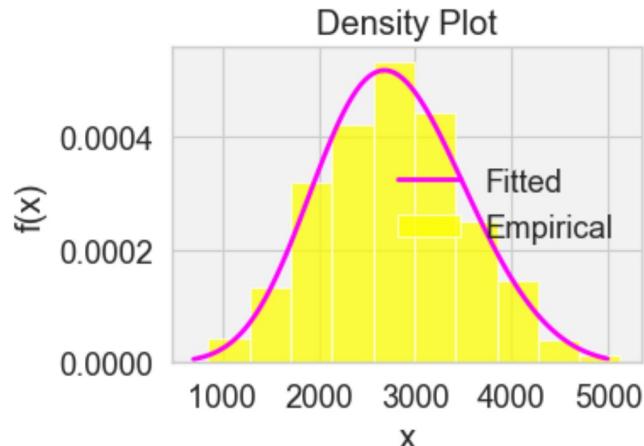
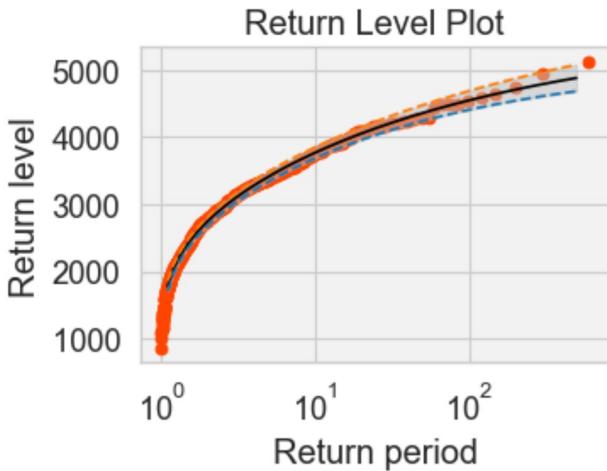
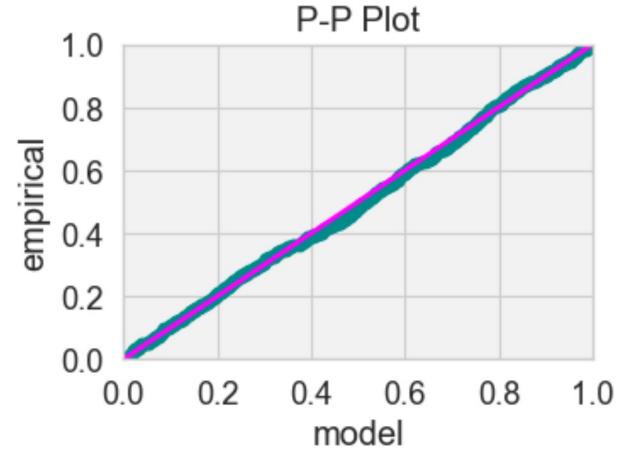
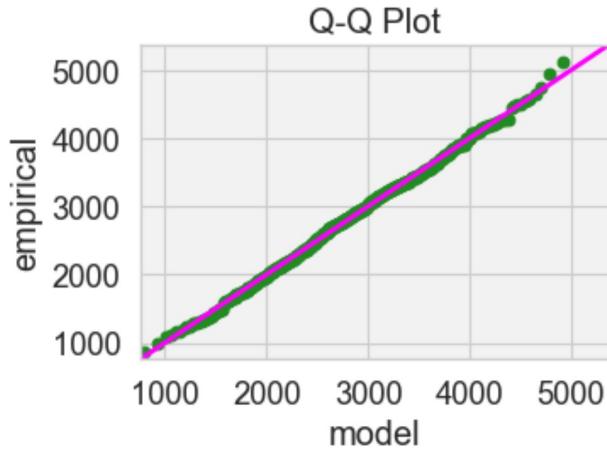
## 2. Modèle GEV en fonction des semestres

QQ-plot, PP-plot, Density plot, Return level plot

```
In [97]: ## regrouper les données de la série Y par semestre
group_by_sem = data.date.dt.to_period("6M")
## sélectionner la date maximale par semestre
max_sem = data.groupby(group_by_sem).max()
Sem = np.array(max_sem['Y'])
## On utilise la méthode de vraisemblance pour estimer les paramètres avec un intervalle
model = ske.models.classic.GEV(Sem, fit_method = 'mle', ci = 0.05, ci_method = 'delta')
## paramètre de queue (eta)
shape= model.params["shape"]
## mu
loc= model.params["location"]
## sigma
scale= model.params["scale"]
## Etude statistique (à comparer à celle effectuée au départ sur le jeu de données)
print("\nEtude statistique: \n nombre d'observations = {} \n
moyenne = {} \n ecart-type = {} \n indice de queue = {} \n min = {} \n max = {} \n kurt
.format(len(Sem), np.mean(Sem), np.std(Sem), shape,
        np.min(Sem), np.max(Sem), stats.kurtosis(Sem), stats.skew(Sem)))
model.plot_summary()
plt.show()
```

Etude statistique:

```
nombre d'observations = 598
moyenne = 2772.6653105386995
ecart-type = 749.1155088671602
indice de queue = 0.2345587481701928
min = 856.1455866054935
max = 5123.995393511563
kurtosis = -0.2755030336281856
skewness = 0.11230360805324
```



Valeur du quantile 99.5% avec la loi GEV par semestre

```
In [98]: quantile_GEV_Sem = genextreme.ppf(0.995, c=shape, loc=loc, scale=scale)
print(f"Le quantile 99.5% estimé avec la loi GEV par semestre est : {quantile_GEV_Sem}")
```

Le quantile 99.5% estimé avec la loi GEV par semestre est : 4706.779468154658

Test de Kolmogorov-Smirnov et somme des moindres carrés

```
In [99]: # Test de Kolmogorov-Smirnov
## Définir la fonction de répartition cumulée (CDF) de la Loi GEV ajustée
cdf_gev = lambda x: genextreme.cdf(y_wo_date, c=shape, loc=loc, scale=scale)

## Calculer la statistique et la p-valeur du test KS
ks_statistic, ks_pvalue = kstest(y_wo_date, cdf_gev)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")
```

```

# Calcul de la somme des moindres carrés
## Histogramme empirique des données et densité théorique ajustée
pdf_empirical, bins = np.histogram(y_wo_date, bins=20, density=True)
bin_centers = (bins[:-1] + bins[1:]) / 2 # Centres des bins
pdf_theoretical = genextreme.pdf(bin_centers, c=shape, loc=loc, scale=scale)

# Calcul de la somme des moindres carrés
sse = np.sum((pdf_empirical - pdf_theoretical) ** 2)
print(f"Somme des moindres carrés : {sse}")

```

Statistique KS : 0.9566753109410007  
P-valeur KS : 0.0  
Somme des moindres carrés : 3.872529996970231e-08

### 3. Modèle GEV en fonction des trimestres

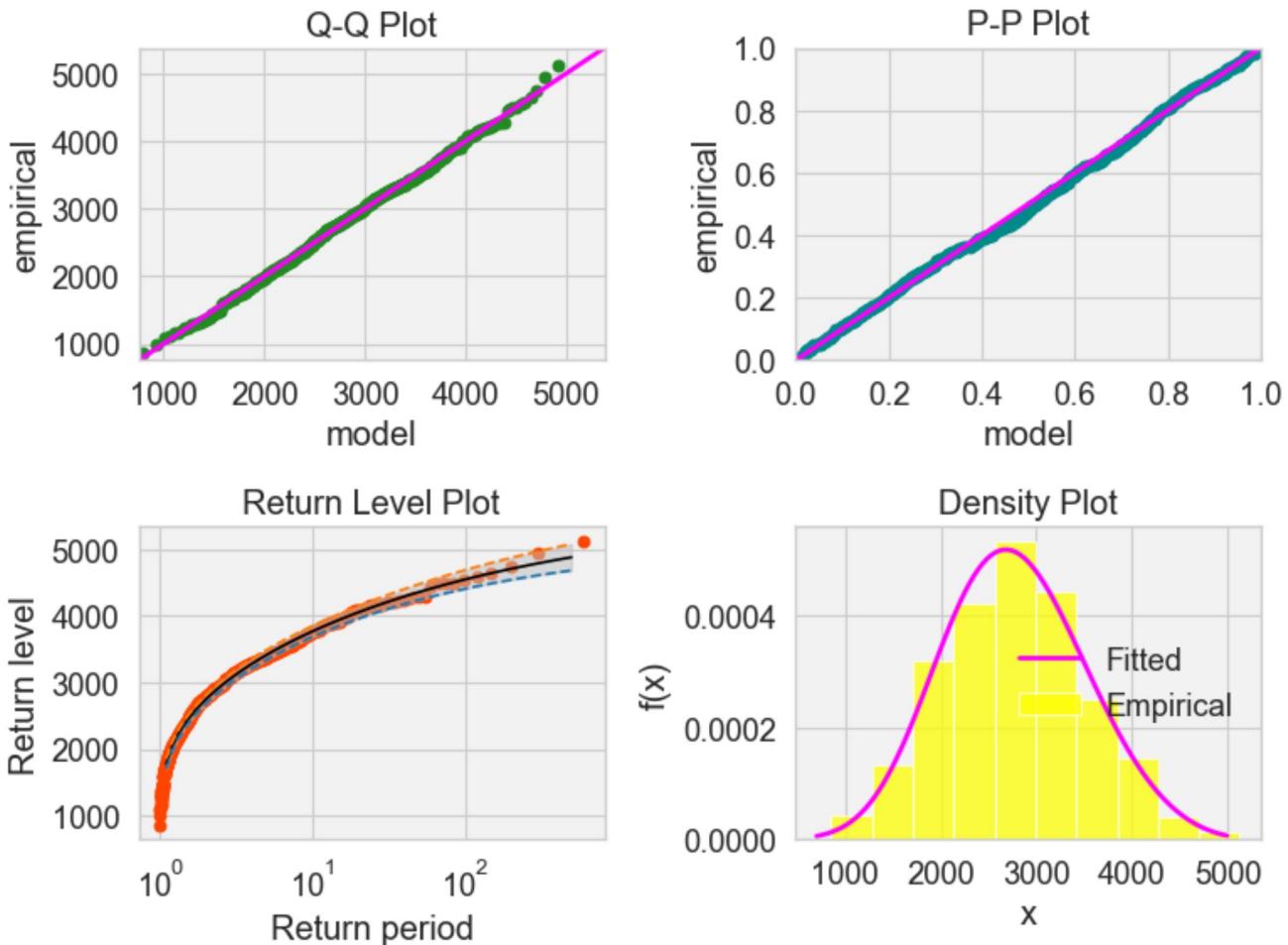
QQ-plot, PP-plot, Density plot, Return level plot

```

In [100...]: ## regrouper les données de la série Y par trimestre
group_by_tri = data.date.dt.to_period("3M")
## sélectionner la date maximale par semestre
max_tri = data.groupby(group_by_tri).max()
Tri = np.array(max_tri['Y'])
## On utilise la méthode de vraisemblance pour estimer les paramètres avec un intervalle
model = ske.models.classic.GEV(Tri, fit_method = 'mle', ci = 0.05, ci_method = 'delta')
## paramètre de queue (eta)
shape= model.params["shape"]
## mu
loc= model.params["location"]
## sigma
scale= model.params["scale"]
## Etude statistique (à comparer à celle effectuée au départ sur le jeu de données)
print("\nEtude statistique: \n nombre d'observations = {} \n moyenne = {} \n ecart-type = {} \n indice de queue = {} \n min = {} \n max = {} \n kurt .format(len(Tri), np.mean(Tri), np.std(Tri), shape, np.min(Tri), np.max(Tri), stats.kurtosis(Tri), stats.skew(Tri)))
model.plot_summary()
plt.show()

```

Etude statistique:  
nombre d'observations = 598  
moyenne = 2772.6653105386995  
ecart-type = 749.1155088671602  
indice de queue = 0.2345587481701928  
min = 856.1455866054935  
max = 5123.995393511563  
kurtosis = -0.2755030336281856  
skewness = 0.11230360805324



Valeur du quantile 99.5% avec la loi GEV par trimestre

```
In [101...]: quantile_GEV_Tri = genextreme.ppf(0.995, c=shape, loc=loc, scale=scale)
print(f"Le quantile 99.5% estimé avec la loi GEV par trimestre est : {quantile_GEV_Tri}")
```

Le quantile 99.5% estimé avec la loi GEV par trimestre est : 4706.779468154658

Test de Kolmogorov-Smirnov et somme des moindres carrés

```
In [102...]: # Test de Kolmogorov-Smirnov
## Définir la fonction de répartition cumulée (CDF) de la Loi GEV ajustée
cdf_gev = lambda x: genextreme.cdf(y_wo_date, c=shape, loc=loc, scale=scale)

## Calculer la statistique et la p-valeur du test KS
ks_statistic, ks_pvalue = kstest(y_wo_date, cdf_gev)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

# Calcul de la somme des moindres carrés
## Histogramme empirique des données et densité théorique ajustée
pdf_empirical, bins = np.histogram(y_wo_date, bins=20, density=True)
bin_centers = (bins[:-1] + bins[1:]) / 2 # Centres des bins
pdf_theoretical = genextreme.pdf(bin_centers, c=shape, loc=loc, scale=scale)

# Calcul de la somme des moindres carrés
sse = np.sum((pdf_empirical - pdf_theoretical) ** 2)
```

```
print(f"Somme des moindres carrés : {sse}")
```

Statistique KS : 0.9566753109410007

P-valeur KS : 0.0

Somme des moindres carrés : 3.872529996970231e-08

## Modélisation de la série Y par la loi GPD

### Mean Excess Plot

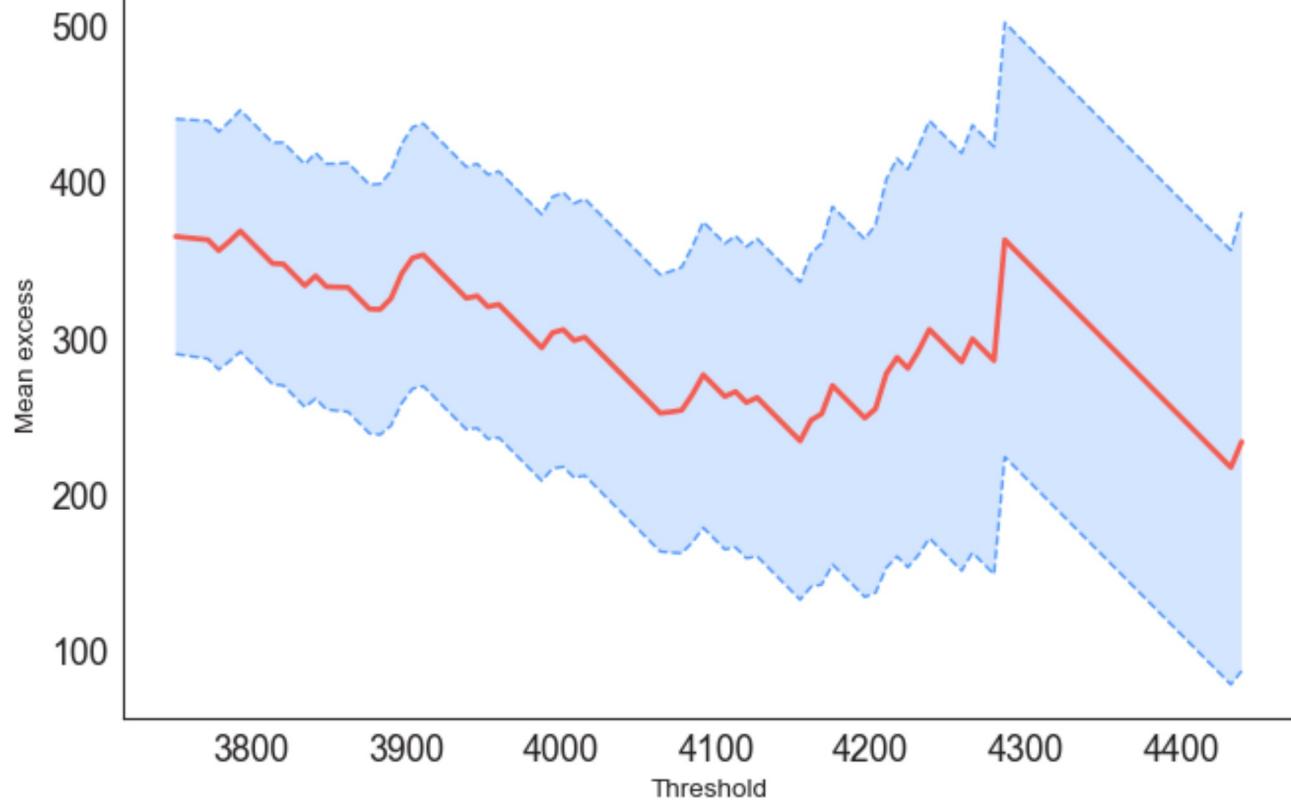
In [103...]

```
## Mean Excess Plot  
plt.figure(figsize=(10, 6))  
plot_mean_residual_life(y_wo_date)
```

Out[103...]

<Axes: xlabel='Threshold', ylabel='Mean excess'>

<Figure size 1000x600 with 0 Axes>



Pour des seuils entre 3800 et 4200, la courbe reste relativement stable et linéaire. Au-delà de 4300, la courbe devient moins stable, et l'incertitude augmente considérablement (zone bleue plus large).

Un seuil de 4000 semble être un bon choix, car la courbe rouge est relativement linéaire et stable autour de ce seuil. De plus, il reste suffisamment de données au-dessus du seuil pour ajuster correctement la loi GPD.

### Distribution du modèle avec la loi GPD

In [104...]

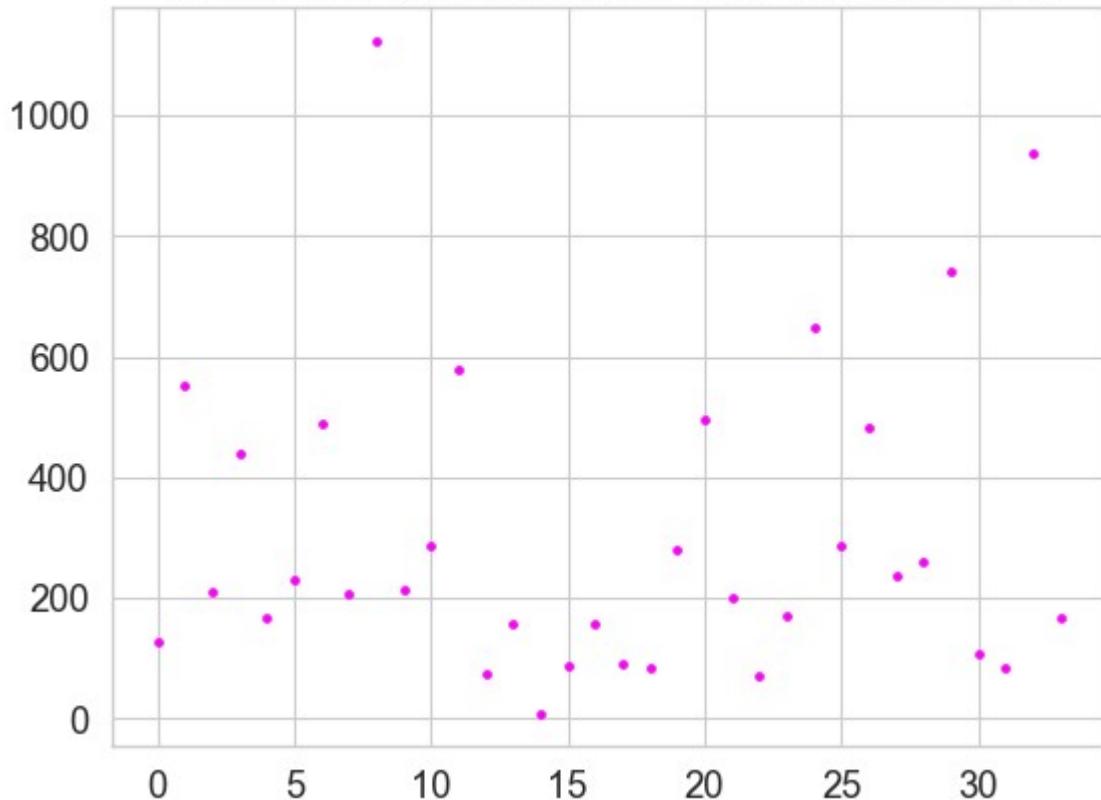
```
# Choix du seuil  
threshold = 4000
```

```

# Calcul des excès
excess = y_wo_date[y_wo_date > threshold] - threshold
if len(excess) == 0:
    raise ValueError(f"Aucune valeur dans la série Y ne dépasse le seuil de {threshold}.")
# Tracé du graphe des valeurs de X prises en compte pour le modèle GPD
plt.plot(range(len(excess)), excess, '.', color='magenta')
plt.title('Valeurs de Y prise en compte dans le modèle GPD')
plt.show()
# Ajustement du modèle GPD
shape, loc, scale = genpareto.fit(excess, loc=0)
print(f"Paramètre de forme (shape): {shape}")
print(f"Paramètre d'échelle (scale): {scale}")
# Histogramme des excès
plt.figure(figsize=(10, 6))
plt.hist(excess, bins=30, density=True, alpha=0.6, color='purple', label='Empirical Excess')
# Densité théorique ajustée
y_gpd = np.linspace(0, max(excess), 100)
pdf = genpareto.pdf(y_gpd, c=shape, loc=0, scale=scale)
plt.plot(y_gpd, pdf, 'orange', label='Fitted GPD')
plt.title(f"Modélisation GPD des excès (Seuil = {threshold})")
plt.xlabel("Excess over Threshold")
plt.ylabel("Density")
plt.legend()
plt.grid()
plt.show()

```

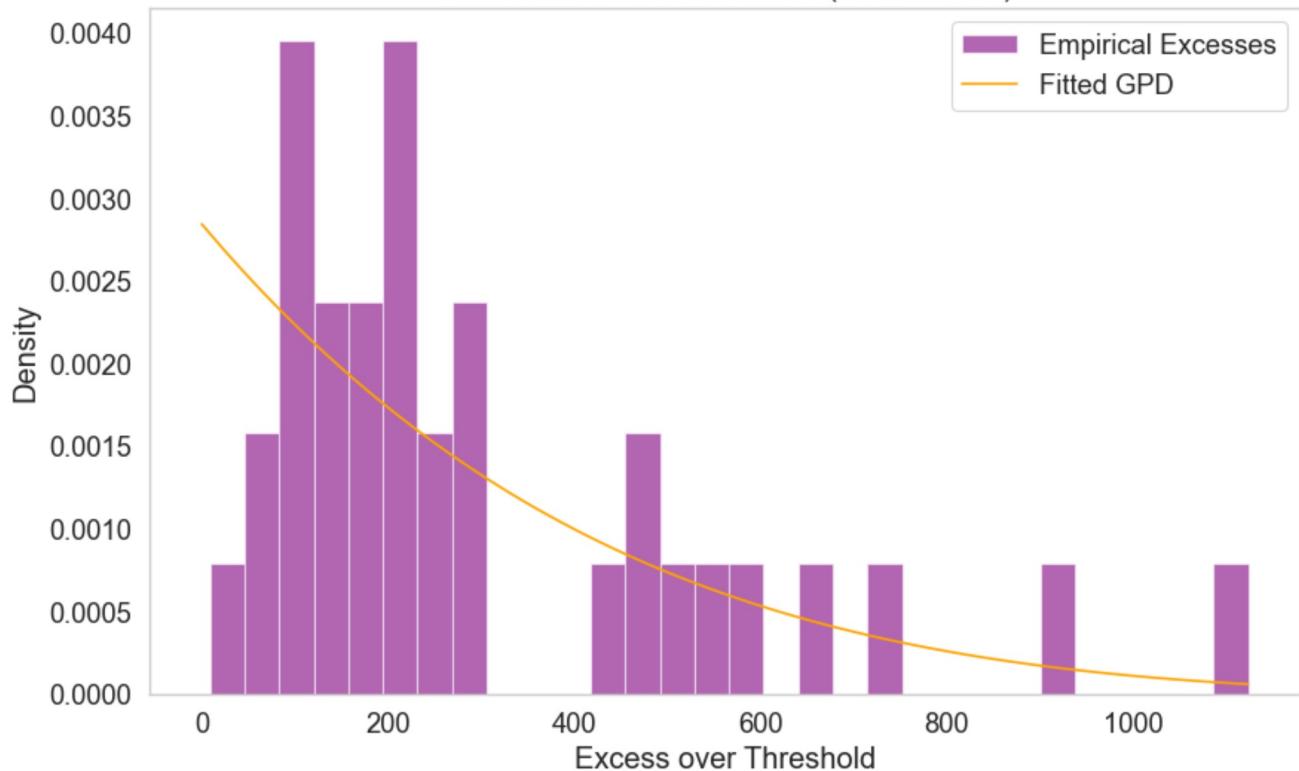
Valeurs de Y prise en compte dans le modèle GPD



Paramètre de forme (shape): -0.1747629925708966

Paramètre d'échelle (scale): 351.50214210148727

### Modélisation GPD des excès (Seuil = 4000)



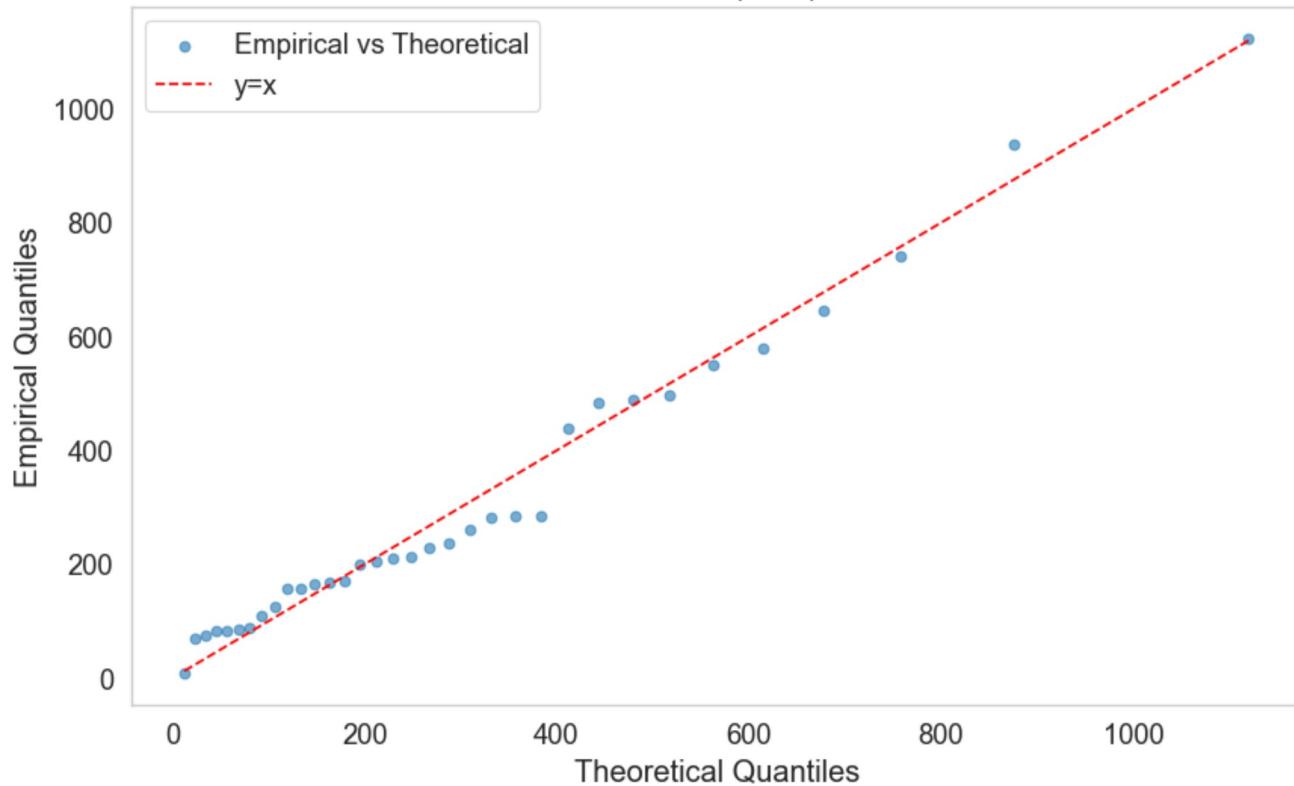
Tracé du QQ-plot et du PP-plot

In [105...]

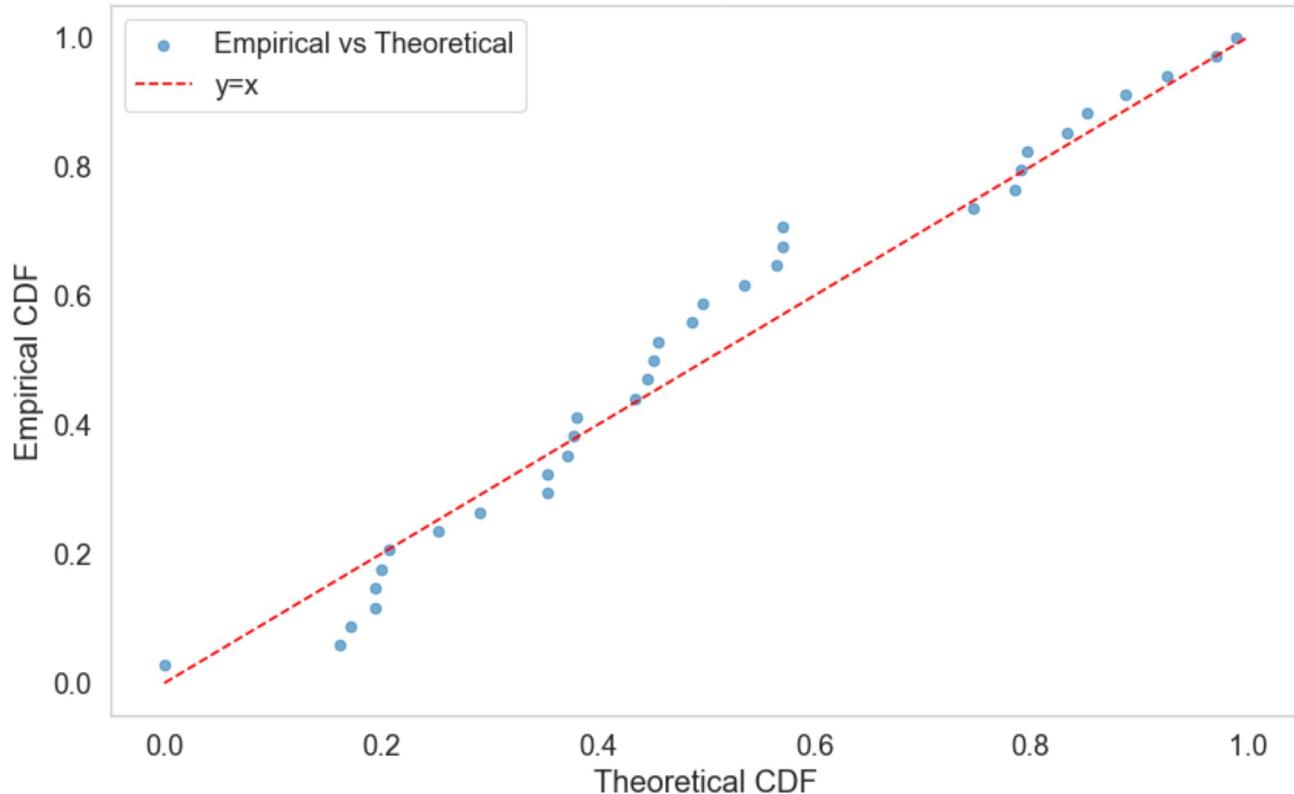
```
# QQ-Plot
plt.figure(figsize=(10, 6))
sorted_data = np.sort(excess)
quantiles = genpareto.ppf(np.linspace(0.01, 0.99, len(sorted_data)), c=shape, loc=loc, scale=scale)
plt.scatter(quantiles, sorted_data, alpha=0.6, label="Empirical vs Theoretical")
plt.plot([min(quantiles), max(quantiles)], [min(quantiles), max(quantiles)], 'r--', label="y=x")
plt.title("QQ-Plot (GPD)")
plt.xlabel("Theoretical Quantiles")
plt.ylabel("Empirical Quantiles")
plt.legend()
plt.grid()
plt.show()

# PP-Plot
plt.figure(figsize=(10, 6))
empirical_cdf = np.arange(1, len(excess) + 1) / len(excess)
theoretical_cdf = genpareto.cdf(np.sort(excess), c=shape, loc=loc, scale=scale)
plt.scatter(theoretical_cdf, empirical_cdf, alpha=0.6, label="Empirical vs Theoretical")
plt.plot([0, 1], [0, 1], 'r--', label="y=x")
plt.title("PP-Plot (GPD)")
plt.xlabel("Theoretical CDF")
plt.ylabel("Empirical CDF")
plt.legend()
plt.grid()
plt.show()
```

QQ-Plot (GPD)



PP-Plot (GPD)



Quantile 99.5%

In [106...]

```
# Calcul du quantile à 99.5%
quantile_GPD = genpareto.ppf(0.995, c=shape, loc=loc, scale=scale)
```

```
print(f"Quantile à 99.5% : {quantile_GPD}")
```

```
Quantile à 99.5% : 1223.1139271575537
```

## Test de Kolmogorov-Smirnov et Somme des moindres carrés

```
In [107...]
```

```
# Test de Kolmogorov-Smirnov
ks_statistic, ks_pvalue = kstest(excess, lambda x: genpareto.cdf(x, c=shape, loc=loc, scale=scale))
print(f"KS-Statistic : {ks_statistic}")
print(f"KS P-Value : {ks_pvalue}")

# Somme des moindres carrés
pdf_empirical, bins = np.histogram(excess, bins=20, density=True)
bin_centers = (bins[:-1] + bins[1:]) / 2
pdf_theoretical = genpareto.pdf(bin_centers, c=shape, loc=loc, scale=scale)
sse = np.sum((pdf_empirical - pdf_theoretical) ** 2)
print(f"Somme des moindres carrés : {sse}")
```

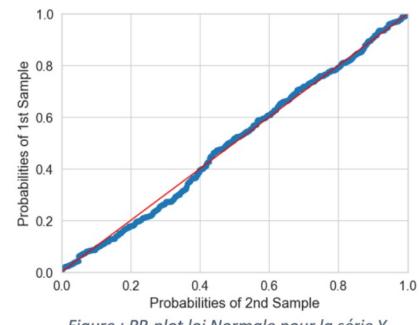
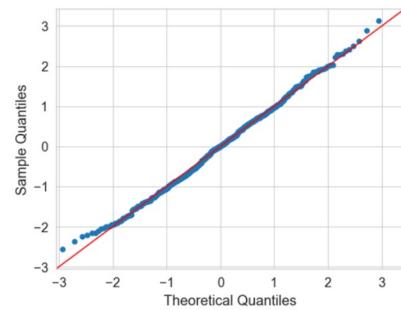
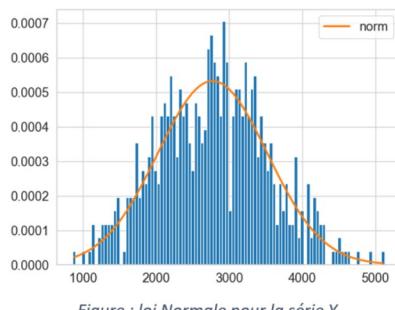
```
KS-Statistic : 0.13444683531772694
```

```
KS P-Value : 0.526704985918818
```

```
Somme des moindres carrés : 1.4160001646172754e-05
```

## Choix du modèle pour la série Y

Au regard des graphes obtenus précédemment, il est évidemment que les lois GEV et la loi GPD ne conviennent pas pour modéliser la série Y. La loi normale est la plus adaptée pour modéliser la série Y, ce qui confirme notre intuition.



Statistique KS	P-value	Somme des moindres carrés
0.027	0.751	7.08e-7

L'histogramme de la série Y (données empiriques) montre une forme symétrique, centrée autour d'une moyenne, qui correspond bien à une loi normale.

Le QQ-plot compare les quantiles théoriques de la loi normale avec les quantiles empiriques de Y. Les points suivent étroitement la ligne rouge (droite de référence), ce qui indique une correspondance entre les quantiles observés et ceux d'une loi normale.

Le PP-plot compare les probabilités cumulées empiriques et théoriques. Les points sont alignés sur la ligne rouge, ce qui montre que la probabilité cumulative empirique est en accord avec celle de la

loi normale.

La statistique KS est de 0.027, ce qui est très faible. Cela indique une faible distance maximale entre la distribution empirique et la distribution théorique (loi normale). Une faible statistique KS est un bon indicateur de la qualité de l'ajustement.

La p-valeur associée est de 0.7510.751, bien supérieure à 0.05. Cela signifie que l'hypothèse nulle ("la loi normale modélise correctement Y") n'est pas rejetée. Statistiquement, il est acceptable de modéliser Y avec une loi normale.

La somme des moindres carrés est de  $7.08 \times 10^{-7}$ , ce qui est très faible. Cela indique que la densité empirique et la densité ajustée par la loi normale sont presque identiques, renforçant l'idée que la loi normale est un bon choix.

## Modélisation de la dépendance des séries X et Y grâce à la théorie des copules

Une copule est une fonction mathématique qui permet de décomposer la distribution conjointe de plusieurs variables aléatoires en deux parties: leurs distributions marginales respectives et une fonction qui décrit leur dépendance, appelée copule. Cette approche sépare le caractère aléatoire propre à chaque variable (capturé par leurs distributions marginales) des relations de dépendance entre elles (modélisées par la copule). Ainsi, la copule représente uniquement les propriétés de corrélation entre les variables, indépendamment de leur comportement individuel.

Dans les parties précédentes, nous avons modéliser les séries X et Y indépendamment l'un de l'autre. Nous allons maintenant nous intéresser à la modélisation de leurs dépendances en utilisant les copules.

Il existe différentes copules adaptées à des types spécifiques de dépendance.

- La copule empirique est une estimation non paramétrique qui repose uniquement sur les données observées. Elle ne nécessite pas de forme spécifique pour modéliser la dépendance mais est limitée aux données disponibles, rendant l'extrapolation difficile.
- La copule gaussienne, quant à elle, repose sur une matrice de corrélation et suppose une dépendance symétrique, ce qui la rend adaptée aux situations où les queues des distributions n'ont pas d'importance particulière.
- La copule de Student est similaire mais intègre des queues lourdes, ce qui la rend plus adaptée aux phénomènes extrêmes.
- La copule de Frank est symétrique et modélise des dépendances modérées.
- La copule de Clayton est spécialisée pour capturer des dépendances dans les queues inférieures, comme dans les pertes financières cumulées.

- La copule de Joe capture des dépendances dans les queues supérieures, typiques des événements simultanément extrêmes.
- La copule de Gumbel est également axée sur les queues supérieures et est utilisée pour modéliser des phénomènes où des extrêmes supérieurs communs sont probables.
- Les versions dites "de survie" de ces copules (survie de Clayton, survie de Gumbel, survie de Joe) inversent leurs propriétés et se concentrent sur les queues opposées.

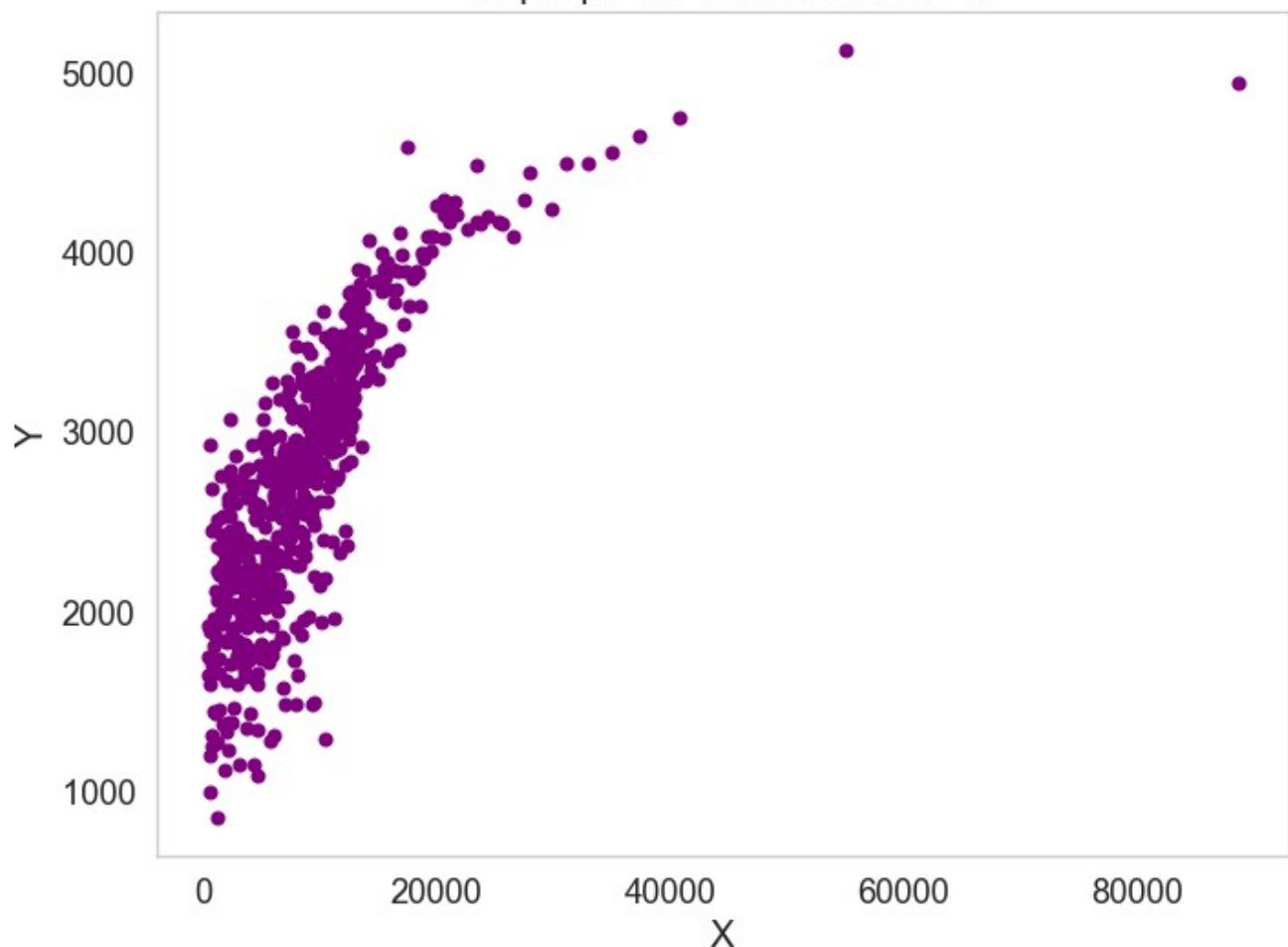
Dans cette partie, nous allons donc comparer la copule empirique aux copules théoriques.

## Justification de l'utilisation des copules

Nous allons avoir recours aux copules afin de modéliser la dépendance entre X et Y, qui n'a pas encore été prises en compte.

```
In [108...]  
plt.figure(figsize=(8, 6))  
plt.plot(x_wo_date, y_wo_date, 'o', c='purple')  
plt.title("Graphique de Y en fonction de X")  
plt.xlabel("X")  
plt.ylabel("Y")  
plt.grid()  
plt.show()
```

Graphique de Y en fonction de X



Ce graphe nous montre bien qu'il existe une dépendance entre X et Y.

Nous allons maintenant évaluer la dépendance entre X et Y grâce au coefficient de corrélation de Pearson, au tau de Kendall et au rho de Spearmann.

## Tests de dépendance

Coefficient de corrélation de Pearson

Le coefficient de corrélation de Pearson est une mesure classique de la dépendance linéaire entre deux variables. Il est défini comme le rapport entre la covariance des variables et le produit de leurs écarts-types.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2} \sqrt{\sum (y_i - \bar{y})^2}}$$

```
In [109... stats.pearsonr(x_wo_date,y_wo_date)
```

```
Out[109... PearsonRResult(statistic=0.76792789117707, pvalue=2.1300231509012158e-117)
```

Le coefficient  $r=0.7679$  indique une corrélation positive forte entre X et Y. Cela signifie que lorsque X augmente, Y tend également à augmenter.

La p-valeur est extrêmement faible ( $2.13 \times 10^{-117}$ ), ce qui signifie que la probabilité d'observer une telle corrélation par pur hasard est pratiquement nulle.

Cela confirme donc que la corrélation entre X et Y est significative et qu'elle doit être prise en compte.

### Tau de Kendall

Le tau de Kendall mesure la proportion de paires concordantes et discordantes dans les données, reflétant ainsi une relation globale de dépendance monotone.

$$\tau = \frac{(\text{nombre de paires concordantes}) - (\text{nombre de paires discordantes})}{\frac{1}{2} \cdot n \cdot (n - 1)}$$

```
In [110... stats.kendalltau(x_wo_date, y_wo_date)
```

```
Out[110... SignificanceResult(statistic=0.6518601928258907, pvalue=1.2514009281440972e-125)
```

Le coefficient  $\tau=0.6519$  indique une dépendance monotone croissante forte entre X et Y.

De plus, la p-valeur est extrêmement faible ( $1.25 \times 10^{-125}$ ), ce qui rejette l'hypothèse d'absence de dépendance monotone.

Cela confirme qu'il existe une dépendance significative entre X et Y.

Contrairement à la corrélation de Pearson,  $\tau$  est robuste face aux relations non linéaires et aux

données aberrantes.

## Rho de Spearmann

Le rho de Spearman, repose sur les rangs des données et évalue la corrélation des rangs, ce qui le rend également robuste face aux relations non linéaires.

$$r_s = \frac{\text{cov}(\text{rg}_X, \text{rg}_Y)}{\sigma_{\text{rg}_X} \sigma_{\text{rg}_Y}}$$

```
In [111... stats.spearmanr(x_wo_date,y_wo_date)
```

```
Out[111... SignificanceResult(statistic=0.8361501142262755, pvalue=1.3839741778154348e-157)
```

Le coefficient  $p=0.8362$  indique une forte dépendance monotone croissante entre X et Y.

La p-valeur ( $1.38 \times 10^{-157}$ ) est extrêmement faible, ce qui rejette l'hypothèse selon laquelle il n'existe aucune corrélation des rangs entre X et Y.

Cela montre qu'il existe une dépendance significative entre X et Y.

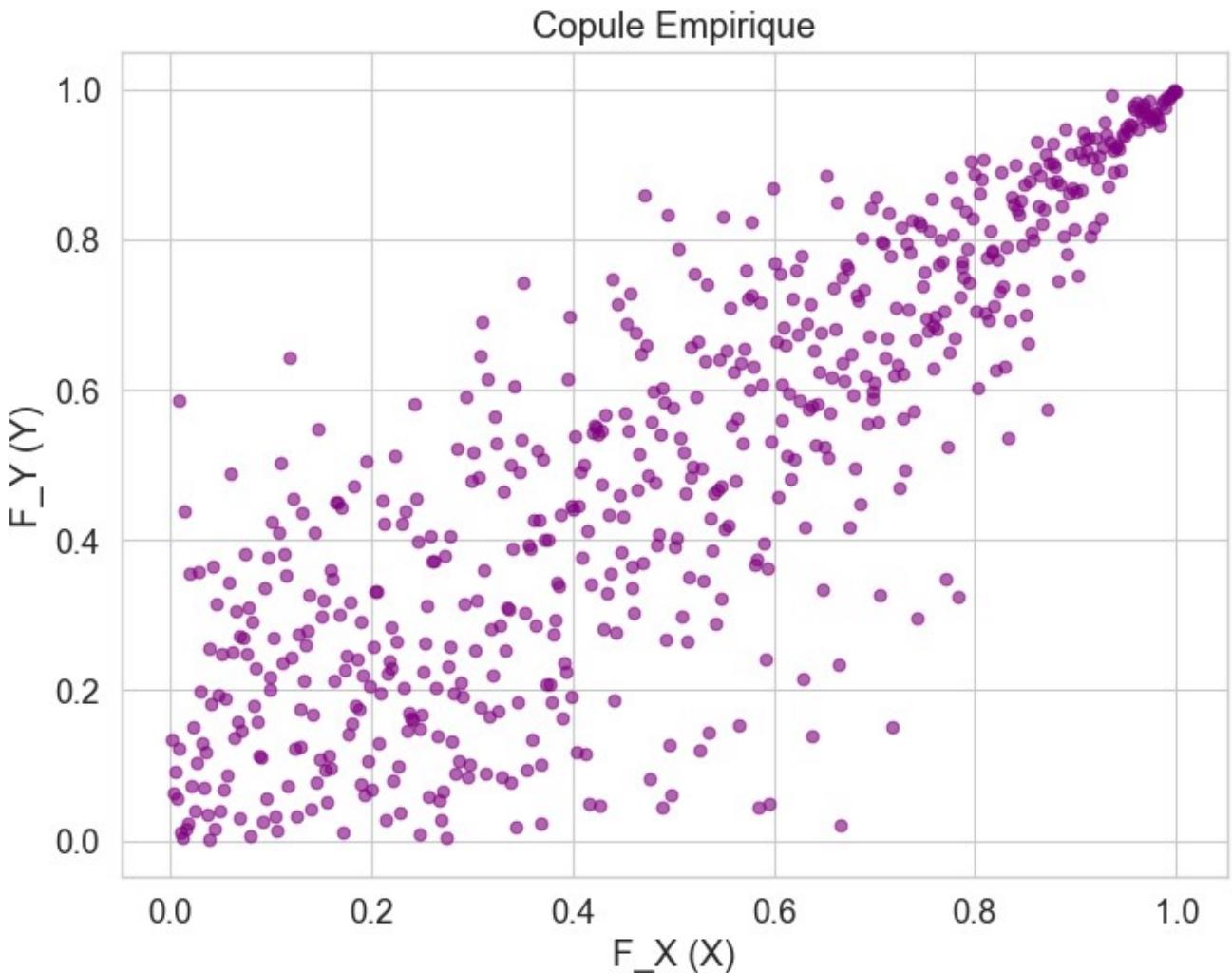
Contrairement à la corrélation de Pearson, le rho de Spearman est basé sur les rangs des données plutôt que sur leurs valeurs brutes. Il est donc robuste aux relations non linéaires et moins sensible aux valeurs aberrantes.

## Copule empirique

```
# Calcul des fonctions de répartition cumulées empiriques (ECDF)
F_X = ECDF(x_wo_date)
F_Y = ECDF(y_wo_date)

# Calcul des coordonnées de la copule empirique
copula_x = F_X(x_wo_date)
copula_y = F_Y(y_wo_date)

# Tracé de la copule empirique
plt.figure(figsize=(8, 6))
plt.scatter(copula_x, copula_y, alpha=0.6, color="purple", marker="o")
plt.title("Copule Empirique")
plt.xlabel("F_X (X)")
plt.ylabel("F_Y (Y)")
plt.grid(True)
plt.show()
```



Ce graphe de copule empirique met en évidence une relation monotone positive entre XX et YY, avec une forte dépendance dans les extrêmes (queues supérieures).

## Copules théoriques

Nous allons maintenant comparer la copule empirique aux neuf copules théoriques vues en cours.

Pour déterminer rigoureusement laquelle des copules théoriques est la plus adaptée pour modéliser la dépendance nette X et Y, nous avons dû utilisé des outils statistiques.

Tout d'abord, nous avons appliqué le test de Kolmogorov-Smirnov, en formulant l'hypothèse nulle ( $H_0$ ) selon laquelle «la copule empirique est correctement modélisée par la copule théorique». Ce test permet de mesurer la différence entre les deux distributions, et une p-valeur supérieure à 0.05 indique que l'hypothèse nulle ne peut pas être rejetée, validant ainsi la compatibilité de la copule théorique avec les données.

Ensuite, nous avons calculé le critère d'information d'Akaike (AIC) et le critère d'information bayésien (BIC). Ces critères mesurent la qualité de l'ajustement tout en pénalisant les modèles trop complexes : des valeurs plus faibles d'AIC et de BIC indiquent un modèle plus efficace et parcimonieux.

Le critère d'information d'Akaike s'écrit comme suit :

$$AIC = 2k - 2 \ln(L)$$

où  $k$  est le nombre de paramètres à estimer du modèle et  $L$  est le maximum de la [fonction de vraisemblance](#) du modèle.

$$BIC = -2 \ln(L) + k \cdot \ln(N)$$

avec  $L$  la [vraisemblance](#) du modèle estimée,  $N$  le nombre d'observations dans l'échantillon et  $k$  le nombre de [paramètres libres](#)

La copule théorique la plus adaptée à la modélisation de la copule empirique sera donc celle qui valide le test de Kolmogorov-Smirnov (p-valeur >0.05) et qui minimise les valeurs des critères AIC et BIC.

## Copule gaussienne

La copule gaussienne est la copule sous-jacente à une distribution normale multivariée.

**Définition 10** Soit  $\rho$  une matrice diagonale définie positive avec  $\text{diag } \rho = \mathbf{1}$  et  $\Phi_\rho$  la distribution normale multivariée standard de matrice de corrélation  $\rho$ . La copule Normale est alors définie de la façon suivante :

$$\mathbf{C}(u_1, \dots, u_n; \rho) = \Phi_\rho(\Phi^{-1}(u_1), \dots, \Phi^{-1}(u_n)) \quad (4.20)$$

**Théorème 7** La densité de la copule Normale est

$$c(u_1, \dots, u_n; \rho) = \frac{1}{|\rho|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} \varsigma^\top (\rho^{-1} - \mathbf{I}) \varsigma\right) \quad (4.21)$$

avec  $\varsigma_i = \Phi^{-1}(u_i)$  et  $\mathbf{I}$  la matrice identité de dimension  $(n \times n)$ .

```
In [113...]: # Calcul du tau de Spearman entre les séries x_wo_date et y_wo_date qui mesure la corrélation
tau = stats.spearmanr(x_wo_date, y_wo_date)[0]

# Définition de la copule gaussienne
rho = tau # On utilise le tau comme estimation de rho (corrélation)
mu_c = np.array([0, 0]) # Moyenne des marges normalisées (0 pour standard normal)
cov_c = np.array([[1, rho], [rho, 1]]) # Matrice de corrélation pour la copule gaussienne

# Définition de la distribution normale multivariée
dist_c = stats.multivariate_normal(mean=mu_c, cov=cov_c)

# Génération d'un échantillon simulé à partir de la copule gaussienne
sample_c = dist_c.rvs(size=1000) # 1000 échantillons simulés
X = sample_c[:, 0] # Première variable
Y = sample_c[:, 1] # Deuxième variable

# Calcul des fonctions de répartition cumulative marginales
# Cela transforme les données en uniformes [0, 1] pour construire la copule
u_x = stats.norm.cdf(X)
u_y = stats.norm.cdf(Y)

# Visualisation de la copule simulée
plt.figure(figsize=(10, 7))
plt.scatter(u_x, u_y, alpha=0.2, marker='o', color='purple', label='Copule Gaussienne')
plt.title("Copule Gaussienne Simulée")
plt.xlabel("u_x (CDF marginale de X)")
plt.ylabel("u_y (CDF marginale de Y)")
```

```

plt.legend()
plt.grid()
plt.show()

# --- Test de Kolmogorov-Smirnov (KS) ---
# Hypothèse nulle : la copule empirique suit la distribution simulée de la copule gaussienne

# Génération de la copule empirique pour comparaison
empirical_copula = np.column_stack((u_x, u_y))
gaussian_copula_simulated = dist_c.cdf(sample_c) # Simule la copule gaussienne théorique

# Test KS
ks_statistic, ks_pvalue = stats.kstest(
    gaussian_copula_simulated, lambda x: x # Comparaison avec l'uniforme empirique
)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

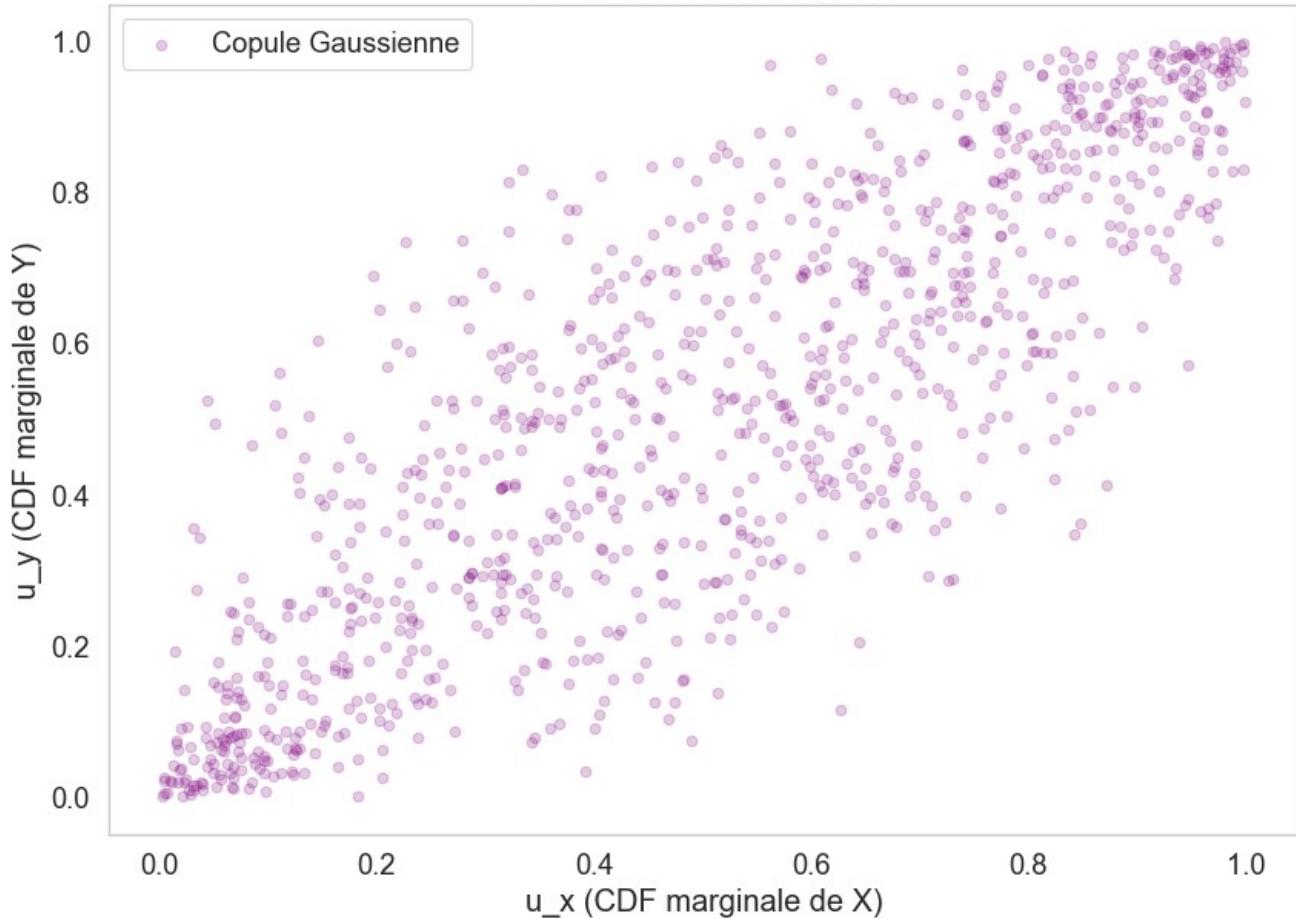
# --- Calcul des critères AIC et BIC ---
# Log-vraisemblance pour la copule gaussienne
log_likelihood = np.sum(dist_c.logpdf(sample_c)) # Somme des log-probabilités
n_params = 1 # Nombre de paramètres estimés (rho dans ce cas)
n_samples = len(sample_c) # Nombre d'échantillons (1000)

# Calcul de l'AIC et du BIC
aic = -2 * log_likelihood + 2 * n_params
bic = -2 * log_likelihood + n_params * np.log(n_samples)

print(f"AIC : {aic}")
print(f"BIC : {bic}")

```

### Copule Gaussienne Simulée



Statistique KS : 0.12646514924136526

P-valeur KS : 2.1175317667420172e-14

AIC : 4475.593947182966

BIC : 4480.501702461948

### Copule de Student

La copule de Student est la copule sous-jacente de la distribution multivariée de Student.

**Définition 12** Soit  $\rho$  une matrice diagonale définie positive avec  $\text{diag } \rho = \mathbf{1}$  et  $t_{\rho, \nu}$  la distribution de Student multivariée standard à  $\nu$  degrés de liberté et de matrice de corrélation  $\rho$ . La copule Student est alors définie de la façon suivante :

$$C(u_1, \dots, u_n; \rho, \nu) = t_{\rho, \nu}(t_{\nu}^{-1}(u_1), \dots, t_{\nu}^{-1}(u_n)) \quad (4.46)$$

**Théorème 9** La densité de la copule Student est

$$c(u_1, \dots, u_n; \rho, \nu) = |\rho|^{-\frac{1}{2}} \frac{\Gamma(\frac{\nu+n}{2}) [\Gamma(\frac{\nu}{2})]^n}{[\Gamma(\frac{\nu+1}{2})]^n \Gamma(\frac{\nu}{2})} \frac{(1 + \frac{1}{\nu} \varsigma^T \rho^{-1} \varsigma)^{-\frac{\nu+n}{2}}}{\prod_{i=1}^n \left(1 + \frac{\varsigma_i^2}{\nu}\right)^{-\frac{\nu+1}{2}}}$$

avec  $\varsigma_i = t_{\nu}^{-1}(u_i)$ .

In [114...]

```
# Calcul des corrélations de Spearman - Tau (corrélation de Spearman pour x_wo_date et y_wo_date)
tau = stats.spearmanr(x_wo_date, y_wo_date)[0]

# Rho (corrélation de Spearman pour X et Y)
rho = stats.spearmanr(x_wo_date, y_wo_date)[0]

# Paramètres pour une copule de Student
nu = 2 # Nombre de degrés de liberté de la distribution de Student
mu_c = np.array([0, 0]) # Moyenne vectorielle
cov_c = np.array([[1, rho], [rho, 1]]) # Matrice de covariance avec dépendance rho

# Définir une distribution multivariée de Student
dist_c = stats.multivariate_t(loc=mu_c, shape=cov_c, df=nu)

# Génération d'échantillons aléatoires à partir de la distribution de Student
sample_c = dist_c.rvs(size=1000) # Génération de 1000 échantillons
x = sample_c[:, 0] # Première dimension
y = sample_c[:, 1] # Deuxième dimension

# Transformation des échantillons en uniformes via la fonction CDF
# Les échantillons sont transformés pour correspondre à une copule de Student dans l'espace uniforme
u_x = stats.t.cdf(x, df=nu) # Transforme x en [0, 1]
u_y = stats.t.cdf(y, df=nu) # Transforme y en [0, 1]

# Visualisation de la copule théorique de Student
plt.figure(figsize=(10, 7))
plt.scatter(u_x, u_y, alpha=0.2, marker='o', color='purple')
plt.title("Copule de Student dans l'espace uniforme")
plt.xlabel("u_x")
plt.ylabel("u_y")
plt.grid(True)
plt.show()

# Test de Kolmogorov-Smirnov
# Hypothèse nulle : la copule théorique et les données sont équivalentes
ks_statistic, ks_pvalue = stats.kstest(
    np.column_stack((u_x, u_y)), 'uniform'
)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

# Calcul de la Log-vraisemblance pour AIC et BIC
# Log-vraisemblance de la copule
log_likelihood = dist_c.logpdf(np.column_stack((x, y))).sum()

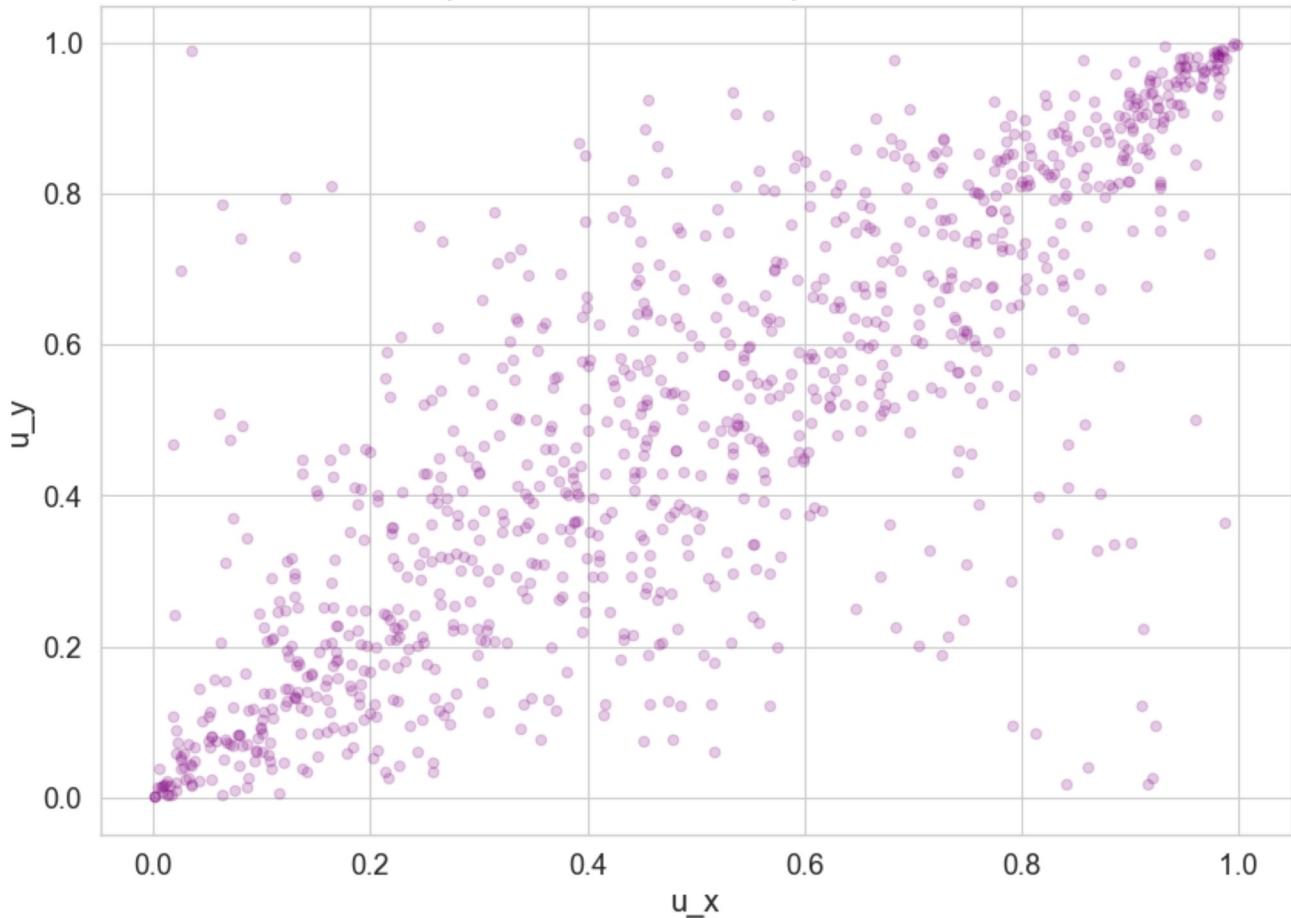
# Nombre de paramètres (1 pour rho, 1 pour nu)
n_params = 2
# Nombre d'échantillons
n_samples = len(x)

# Calcul de AIC
aic = -2 * log_likelihood + 2 * n_params

# Calcul de BIC
bic = -2 * log_likelihood + n_params * np.log(n_samples)
```

```
# Affichage des résultats
print(f"AIC : {aic}")
print(f"BIC : {bic}")
```

Copule de Student dans l'espace uniforme



Statistique KS : [0.02439798 0.01974471]

P-valeur KS : [0.5824151 0.82290318]

AIC : 6348.633392903294

BIC : 6358.448903461259

### Copule de Frank

La copule de Frank est une copule archimédienne, c'est-à-dire qu'elle admet une formule explicite.

$$C(u, v) = -\frac{1}{\theta} \log \left[ 1 + \frac{(\exp(-\theta u) - 1)(\exp(-\theta v) - 1)}{\exp(-\theta) - 1} \right] \quad \theta \in \mathbb{R} \setminus \{0\}$$

In [115...]

```
# Fonction de répartition cumulative de la copule de Frank
def frank_copula_cdf(u, v, theta):
    """
    Fonction de répartition cumulative de la copule de Frank.
    u, v : les données uniformes [0, 1].
    theta : paramètre de la copule de Frank.
    """
    if theta == 0:
```

```

        return u * v # Copule indépendante (cas limite)

    exp_theta = np.exp(-theta)
    num = (np.exp(-theta * u) - 1) * (np.exp(-theta * v) - 1)
    den = exp_theta - 1
    return -1 / theta * np.log(1 + num / den)

# Normaliser les séries en données uniformes [0, 1]
def normalize_to_uniform(series):
    """
    Transforme une série en données uniformes [0, 1] via la fonction ECDF.
    """
    ecdf = ECDF(series)
    return ecdf(series)

# Transformation des séries en uniformes
u_empirical = normalize_to_uniform(x_wo_date)
v_empirical = normalize_to_uniform(y_wo_date)

# Ajustement du paramètre theta
def frank_log_likelihood(params, u, v):
    """
    Log-vraisemblance de la copule de Frank.
    params : [theta].
    u, v : données uniformes [0, 1].
    """
    theta = params[0]
    cdf_values = frank_copula_cdf(u, v, theta)
    return -np.sum(np.log(cdf_values + 1e-9)) # Éviter les problèmes numériques

result = minimize(frank_log_likelihood, x0=[1], args=(u_empirical, v_empirical), bounds=[(0, 10)])
theta_hat = result.x[0]
print(f"Paramètre theta ajusté : {theta_hat}")

# Calculer la copule de Frank pour les séries transformées
copula_values = frank_copula_cdf(u_empirical, v_empirical, theta_hat)

# Tracer la copule simulée
plt.figure(figsize=(10, 7))
plt.scatter(u_empirical, v_empirical, alpha=0.5, color='purple', label="Copule de Frank simulée pour x_wo_date et y_wo_date")
plt.title("Copule de Frank simulée pour x_wo_date et y_wo_date")
plt.xlabel("u (x_wo_date)")
plt.ylabel("v (y_wo_date)")
plt.legend()
plt.grid(True)
plt.show()

# Test de Kolmogorov-Smirnov (KS)
# Comparer les copules simulée et empirique
ks_statistic, ks_pvalue = kstest(
    copula_values, lambda x: frank_copula_cdf(x, x, theta_hat)
)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

```

```

# Calcul des critères AIC et BIC
# Log-vraisemblance pour la copule ajustée
log_likelihood = -frank_log_likelihood([theta_hat], u_empirical, v_empirical)
n_params = 1 # Nombre de paramètres (theta)
n_samples = len(u_empirical)

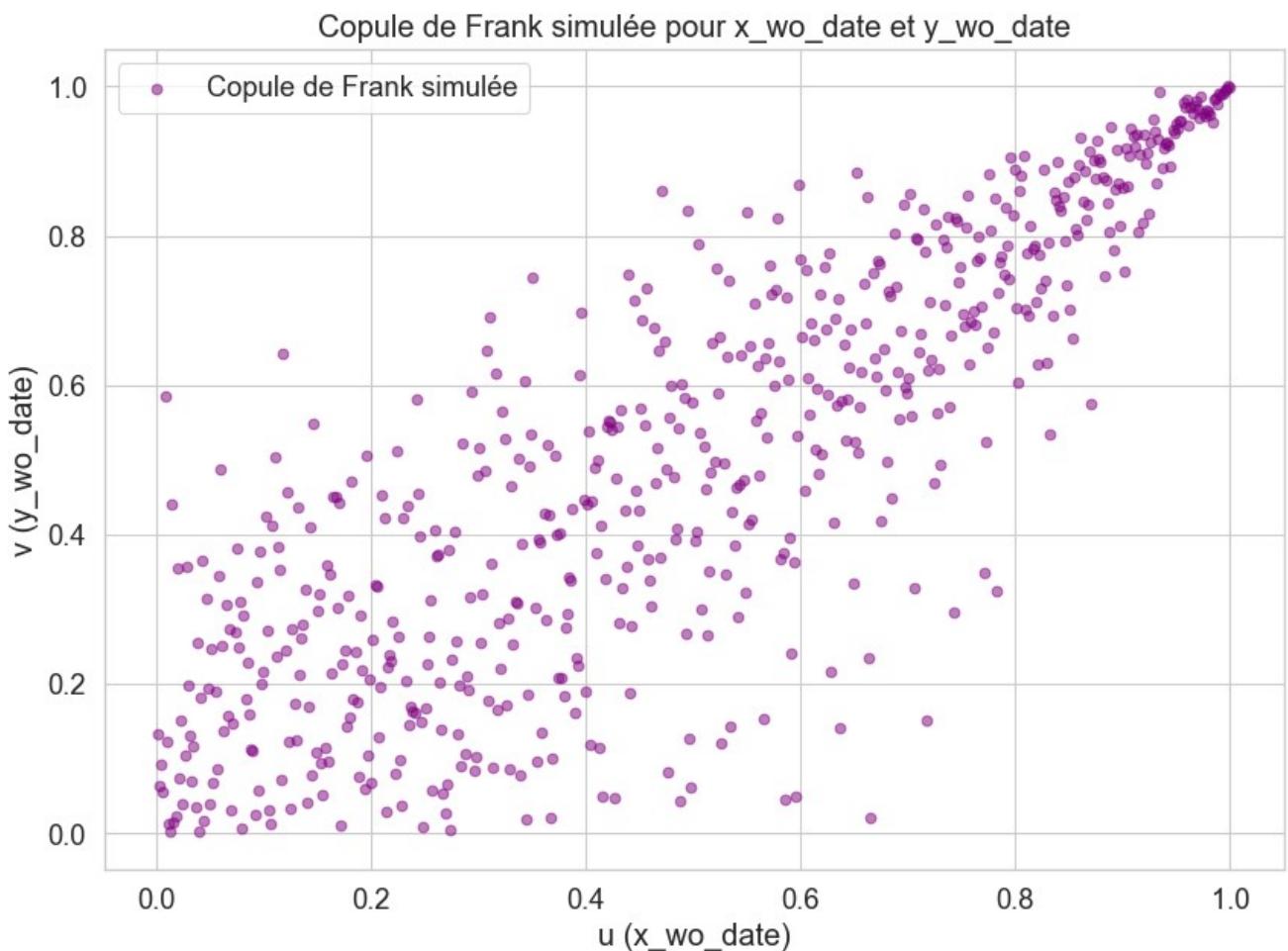
# Calcul de AIC
aic = -2 * log_likelihood + 2 * n_params

# Calcul de BIC
bic = -2 * log_likelihood + n_params * np.log(n_samples)

# Affichage des résultats
print(f"AIC : {aic}")
print(f"BIC : {bic}")

```

Paramètre theta ajusté : 10.0



Statistique KS : 0.1938829922779123  
P-valeur KS : 3.630947368525744e-20  
AIC : 1629.6473009278643  
BIC : 1634.0408916818149

## Copule de Clayton

La copule de Clayton est une copule archimédienne, c'est-à-dire qu'elle admet une formule explicite.

$$C(u, v) = [\max \{u^{-\theta} + v^{-\theta} - 1; 0\}]^{-1/\theta} \quad \theta \in [-1, \infty) \setminus \{0\}$$

In [116...]

```
# Fonction de répartition cumulative de la copule de Clayton
def clayton_copula_cdf(u, v, theta):
    """
    Fonction de répartition cumulative de la copule de Clayton.
    u, v : les données uniformes [0, 1].
    theta : paramètre de la copule de Clayton.
    """
    if theta == 0:
        return u * v # Copule indépendante (cas limite)
    return (u**(-theta) + v**(-theta) - 1)**(-1/theta)

# Normaliser les séries en données uniformes [0, 1]
def normalize_to_uniform(series):
    """
    Transforme une série en données uniformes [0, 1] via la fonction ECDF.
    """
    ecdf = ECDF(series)
    return ecdf(series)

# Transformation des séries en uniformes
u_empirical = normalize_to_uniform(x_wo_date)
v_empirical = normalize_to_uniform(y_wo_date)

# Ajustement du paramètre theta
def clayton_log_likelihood(params, u, v):
    """
    Log-vraisemblance de la copule de Clayton.
    params : [theta].
    u, v : données uniformes [0, 1].
    """
    theta = params[0]
    cdf_values = clayton_copula_cdf(u, v, theta)
    return -np.sum(np.log(cdf_values + 1e-9)) # Éviter les problèmes numériques

result = minimize(clayton_log_likelihood, x0=[1], args=(u_empirical, v_empirical), bounds=[(0, 10)])
theta_hat = result.x[0]
print(f"Paramètre theta ajusté : {theta_hat}")

# Calculer la copule de Clayton pour les séries transformées
copula_values = clayton_copula_cdf(u_empirical, v_empirical, theta_hat)

# Tracer la copule simulée
plt.figure(figsize=(10, 7))
plt.scatter(u_empirical, v_empirical, alpha=0.5, color='purple', label="Copule de Clayton")
plt.title("Copule de Clayton simulée pour x_wo_date et y_wo_date")
plt.xlabel("u (x_wo_date)")
plt.ylabel("v (y_wo_date)")
plt.legend()
plt.grid(True)
```

```

plt.show()

# Test de Kolmogorov-Smirnov (KS)
# Comparer les copules simulée et empirique
ks_statistic, ks_pvalue = kstest(
    copula_values, lambda x: clayton_copula_cdf(x, x, theta_hat)
)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

# Calcul des critères AIC et BIC
# Log-vraisemblance pour la copule ajustée
log_likelihood = -clayton_log_likelihood([theta_hat], u_empirical, v_empirical)
n_params = 1 # Nombre de paramètres (theta)
n_samples = len(u_empirical)

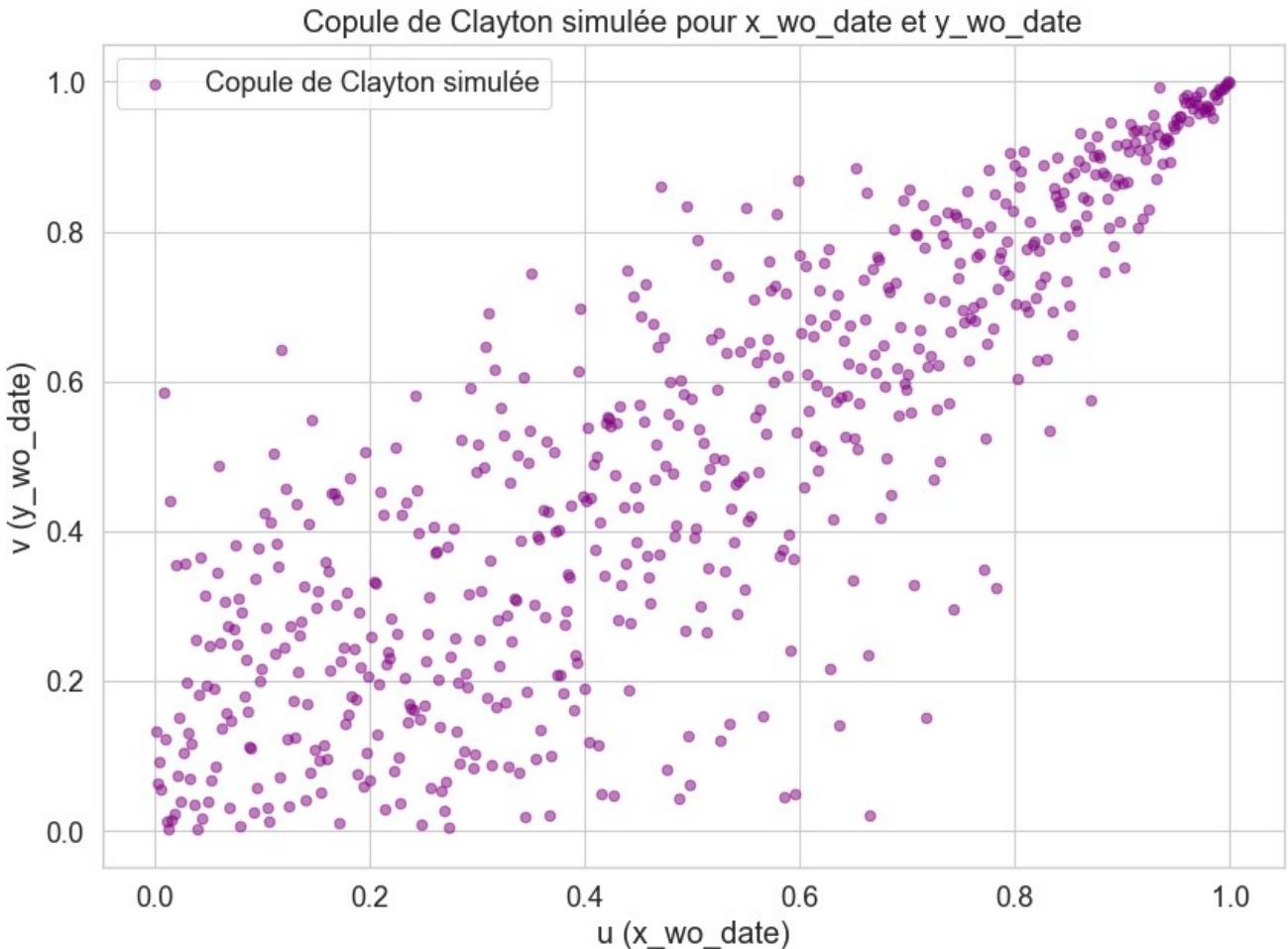
# Calcul de AIC
aic = -2 * log_likelihood + 2 * n_params

# Calcul de BIC
bic = -2 * log_likelihood + n_params * np.log(n_samples)

# Affichage des résultats
print(f"AIC : {aic}")
print(f"BIC : {bic}")

```

Paramètre theta ajusté : 10.0



```
Statistique KS : 0.13033659412752274
P-valeur KS : 2.5653397251438034e-09
AIC : 1501.6540412968984
BIC : 1506.047632050849
```

## Copule de Gumbel

La copule de Gumbel est une copule archimédienne, c'est-à-dire qu'elle admet une formule explicite.

$$C(u, v) = \exp\left[-\left((- \log(u))^{\theta} + (- \log(v))^{\theta}\right)^{1/\theta}\right] \quad \theta \in [1, \infty)$$

In [117...]

```
# Fonction de répartition cumulative de la copule de Gumbel
def gumbel_copula_cdf(u, v, theta):
    """
    Fonction de répartition cumulative de la copule de Gumbel.
    u, v : les données uniformes [0, 1].
    theta : paramètre de la copule de Gumbel.
    """
    if theta == 1:
        return u * v # Copule indépendante (cas limite)
    t1 = (-np.log(u))**theta
    t2 = (-np.log(v))**theta
    return np.exp(-((t1 + t2)**(1 / theta)))

# Normaliser les séries en données uniformes [0, 1]
def normalize_to_uniform(series):
    """
    Transforme une série en données uniformes [0, 1] via la fonction ECDF.
    """
    ecdf = ECDF(series)
    return ecdf(series)

# Transformation des séries en uniformes
u_empirical = normalize_to_uniform(x_wo_date)
v_empirical = normalize_to_uniform(y_wo_date)

# Ajustement du paramètre theta
def gumbel_log_likelihood(params, u, v):
    """
    Log-vraisemblance de la copule de Gumbel.
    params : [theta].
    u, v : données uniformes [0, 1].
    """
    theta = params[0]
    cdf_values = gumbel_copula_cdf(u, v, theta)
    return -np.sum(np.log(cdf_values + 1e-9)) # Éviter les problèmes numériques

result = minimize(gumbel_log_likelihood, x0=[1.5], args=(u_empirical, v_empirical), bounds
theta_hat = result.x[0]
```

```

print(f"Paramètre theta ajusté : {theta_hat}")

# Calculer la copule de Gumbel pour les séries transformées
copula_values = gumbel_copula_cdf(u_empirical, v_empirical, theta_hat)

# Tracer la copule simulée
plt.figure(figsize=(10, 7))
plt.scatter(u_empirical, v_empirical, alpha=0.5, color='purple', label="Copule de Gumbel simulée")
plt.title("Copule de Gumbel simulée pour x_wo_date et y_wo_date")
plt.xlabel("u (x_wo_date)")
plt.ylabel("v (y_wo_date)")
plt.legend()
plt.grid(True)
plt.show()

# Test de Kolmogorov-Smirnov (KS)
# Comparer les copules simulée et empirique
ks_statistic, ks_pvalue = kstest(
    copula_values, lambda x: gumbel_copula_cdf(x, x, theta_hat)
)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

# Calcul des critères AIC et BIC
# Log-vraisemblance pour la copule ajustée
log_likelihood = -gumbel_log_likelihood([theta_hat], u_empirical, v_empirical)
n_params = 1 # Nombre de paramètres (theta)
n_samples = len(u_empirical)

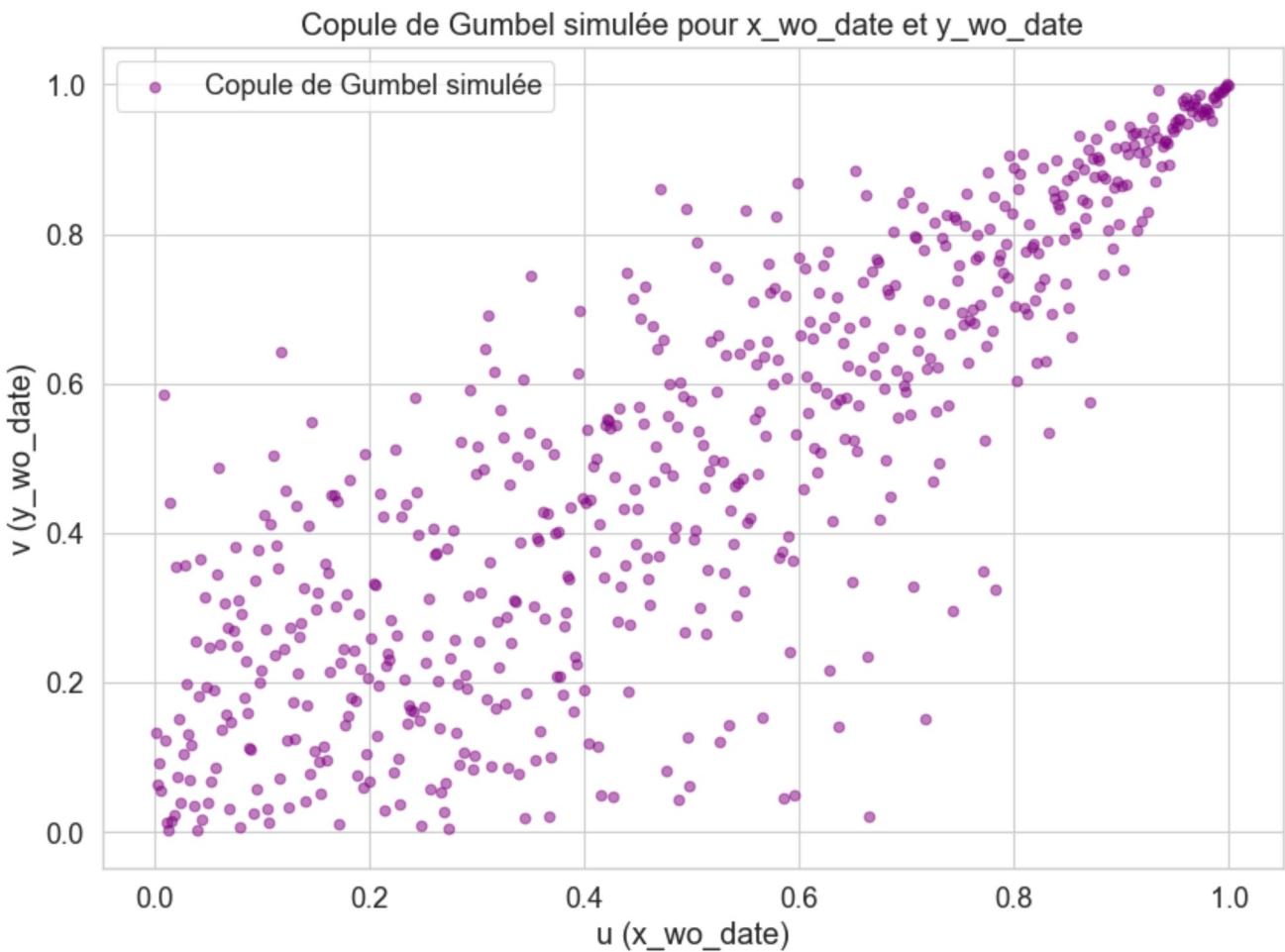
# Calcul de AIC
aic = -2 * log_likelihood + 2 * n_params

# Calcul de BIC
bic = -2 * log_likelihood + n_params * np.log(n_samples)

# Affichage des résultats
print(f"AIC : {aic}")
print(f"BIC : {bic}")

```

Paramètre theta ajusté : 10.0



Statistique KS : 0.1322885288202327  
 P-valeur KS : 1.3814509262040953e-09  
 AIC : 1494.4946059376784  
 BIC : 1498.888196691629

### Copule de Joe

La copule de Joe est une copule archimédienne, c'est-à-dire qu'elle admet une formule explicite.

$$C(u, v) = 1 - [(1 - u)^\theta + (1 - v)^\theta - (1 - u)^\theta(1 - v)^\theta]^{1/\theta} \quad \theta \in [1, \infty)$$

```
In [118]: # Fonction de répartition cumulative de la copule de Joe
def joe_copula_cdf(u, v, theta):
    """
    Fonction de répartition cumulative de la copule de Joe.
    u, v : les données uniformes [0, 1].
    theta : paramètre de la copule de Joe.
    """
    if theta == 1:
        return u * v # Copule indépendante (cas limite)
    term1 = (1 - (1 - u)**theta)
```

```

term2 = (1 - (1 - v)**theta)
return 1 - ((1 - term1 * term2)**(1 / theta))

# Normaliser les séries en données uniformes [0, 1]
def normalize_to_uniform(series):
    """
    Transforme une série en données uniformes [0, 1] via la fonction ECDF.
    """
    ecdf = ECDF(series)
    return ecdf(series)

# Transformation des séries en uniformes
u_empirical = normalize_to_uniform(x_wo_date)
v_empirical = normalize_to_uniform(y_wo_date)

# Ajustement du paramètre theta
def joe_log_likelihood(params, u, v):
    """
    Log-vraisemblance de la copule de Joe.
    params : [theta].
    u, v : données uniformes [0, 1].
    """
    theta = params[0]
    cdf_values = joe_copula_cdf(u, v, theta)
    return -np.sum(np.log(cdf_values + 1e-9)) # Éviter les problèmes numériques

result = minimize(joe_log_likelihood, x0=[1.5], args=(u_empirical, v_empirical), bounds=([(theta_hat = result.x[0]
print(f"Paramètre theta ajusté : {theta_hat}")

# Calculer la copule de Joe pour les séries transformées
copula_values = joe_copula_cdf(u_empirical, v_empirical, theta_hat)

# Tracer la copule simulée
plt.figure(figsize=(10, 7))
plt.scatter(u_empirical, v_empirical, alpha=0.5, color='purple', label="Copule de Joe simulée")
plt.title("Copule de Joe simulée pour x_wo_date et y_wo_date")
plt.xlabel("u (x_wo_date)")
plt.ylabel("v (y_wo_date)")
plt.legend()
plt.grid(True)
plt.show()

# Test de Kolmogorov-Smirnov (KS)
# Comparer les copules simulée et empirique
ks_statistic, ks_pvalue = kstest(
    copula_values, lambda x: joe_copula_cdf(x, x, theta_hat)
)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

# Calcul des critères AIC et BIC
# Log-vraisemblance pour la copule ajustée
log_likelihood = -joe_log_likelihood([theta_hat], u_empirical, v_empirical)

```

```

n_params = 1 # Nombre de paramètres (theta)
n_samples = len(u_empirical)

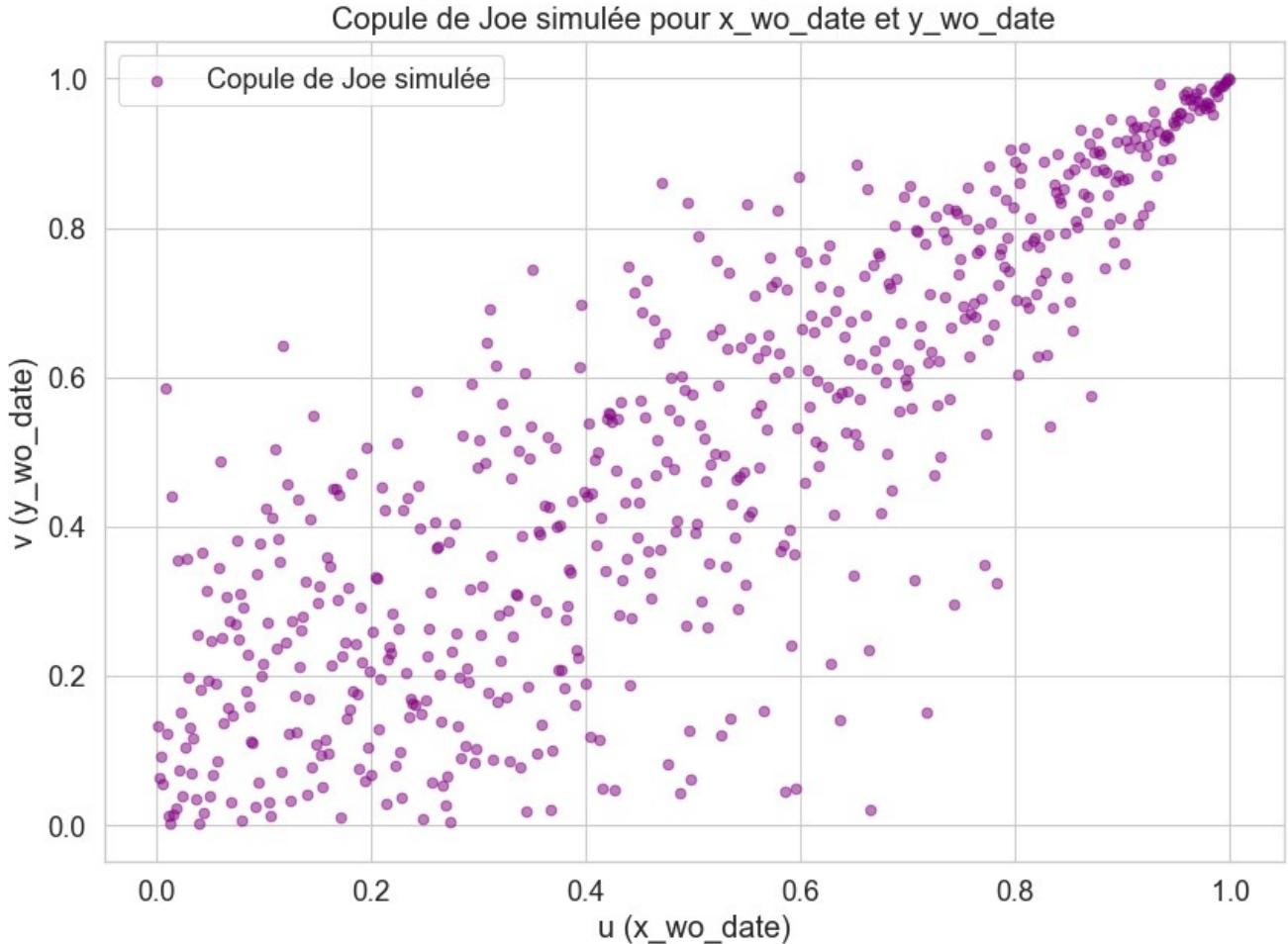
# Calcul de AIC
aic = -2 * log_likelihood + 2 * n_params

# Calcul de BIC
bic = -2 * log_likelihood + n_params * np.log(n_samples)

# Affichage des résultats
print(f"AIC : {aic}")
print(f"BIC : {bic}")

```

Paramètre theta ajusté : 10.0



Statistique KS : 0.17534354834257293

P-valeur KS : 1.4970783536956668e-16

AIC : 1579.7465672920011

BIC : 1584.1401580459517

### Copule de Survie Clayton

La copule de survie Clayton est une transformation de la copule de Clayton qui modélise la dépendance dans les queues supérieures des distributions. Elle est utile pour analyser des situations où les grandes valeurs des variables sont corrélées.

Pour deux variables  $U$  et  $V$  uniformes dans  $[0, 1]$ , la copule de survie Clayton est donnée par :

$$C^{\text{surv}}(u, v; \theta) = u + v - 1 + \left[ (u^{-\theta} + v^{-\theta} - 1)^{-1/\theta} \right]$$

Où :

- $\theta > 0$  est le paramètre de dépendance,
- $C^{\text{surv}}(u, v; \theta)$  modélise la dépendance dans les queues supérieures,

In [119...]

```
# Fonction de répartition cumulative de la copule de survie Clayton
def survival_clayton_cdf(u, v, theta):
    """
    Fonction de répartition cumulative de la copule de survie Clayton.
    u, v : les données uniformes [0, 1].
    theta : paramètre de la copule de survie Clayton.
    """
    if theta == 0:
        return u * v # Copule indépendante (cas limite)
    copula_value = (u**(-theta) + v**(-theta) - 1)**(-1/theta)
    return u + v - copula_value # Transformation survie

# Normaliser les séries en données uniformes [0, 1]
def normalize_to_uniform(series):
    """
    Transforme une série en données uniformes [0, 1] via la fonction ECDF.
    """
    ecdf = ECDF(series)
    return ecdf(series)

# Transformation des séries en uniformes
u_empirical = normalize_to_uniform(x_wo_date)
v_empirical = normalize_to_uniform(y_wo_date)

# Ajustement du paramètre theta
def survival_clayton_log_likelihood(params, u, v):
    """
    Log-vraisemblance de la copule de survie Clayton.
    params : [theta].
    u, v : données uniformes [0, 1].
    """
    theta = params[0]
    cdf_values = survival_clayton_cdf(u, v, theta)
    return -np.sum(np.log(cdf_values + 1e-9)) # Éviter les problèmes numériques

result = minimize(survival_clayton_log_likelihood, x0=[1], args=(u_empirical, v_empirical))
theta_hat = result.x[0]
print(f"Paramètre theta ajusté : {theta_hat}")

# Calculer la copule de survie Clayton pour les séries transformées
copula_values = survival_clayton_cdf(u_empirical, v_empirical, theta_hat)
```

```

# Tracer la copule simulée
plt.figure(figsize=(10, 7))
plt.scatter(u_empirical, v_empirical, alpha=0.5, color='purple', label="Copule de survie Clayton")
plt.title("Copule de survie Clayton simulée pour x_wo_date et y_wo_date")
plt.xlabel("u (x_wo_date)")
plt.ylabel("v (y_wo_date)")
plt.legend()
plt.grid(True)
plt.show()

# Test de Kolmogorov-Smirnov (KS)
# Comparer les copules simulée et empirique
ks_statistic, ks_pvalue = kstest(
    copula_values, lambda x: survival_clayton_copula_cdf(x, x, theta_hat)
)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

# Calcul des critères AIC et BIC
# Log-vraisemblance pour la copule ajustée
log_likelihood = -survival_clayton_log_likelihood([theta_hat], u_empirical, v_empirical)
n_params = 1 # Nombre de paramètres (theta)
n_samples = len(u_empirical)

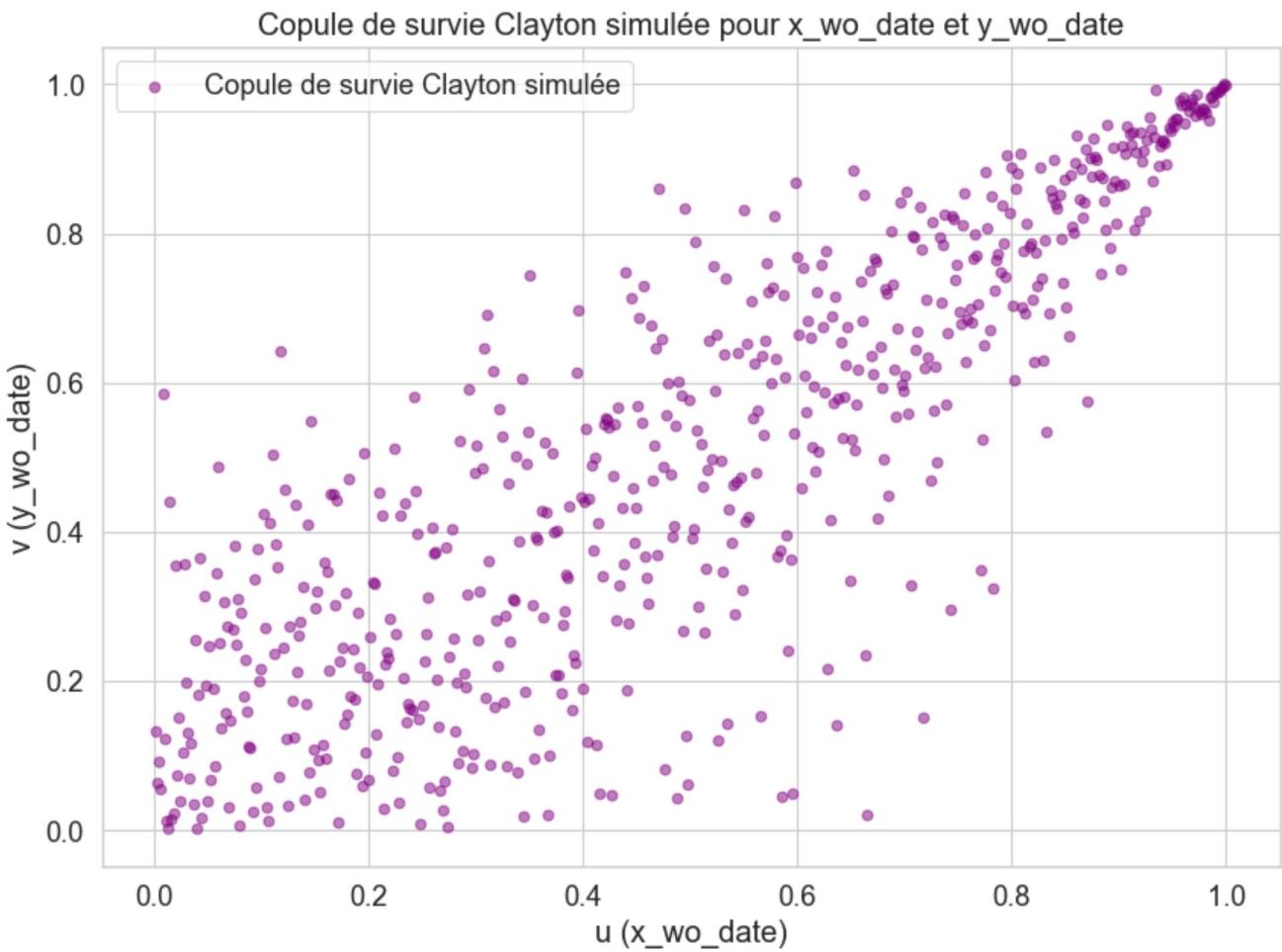
# Calcul de AIC
aic = -2 * log_likelihood + 2 * n_params

# Calcul de BIC
bic = -2 * log_likelihood + n_params * np.log(n_samples)

# Affichage des résultats
print(f"AIC : {aic}")
print(f"BIC : {bic}")

```

Paramètre theta ajusté : 0.1



Statistique KS : 0.46156520104967474

P-valeur KS : 4.035030119301034e-117

AIC : 639.7280219193341

BIC : 644.1216126732847

### Copule de Survie Gumbel

La copule de survie Gumbel est une transformation de la copule de Gumbel qui modélise la dépendance dans les queues inférieures des distributions. Elle est utilisée pour analyser les situations où les faibles valeurs des variables sont corrélées.

Pour deux variables  $U$  et  $V$  uniformes dans  $[0, 1]$ , la copule de survie Gumbel est donnée par :

$$C^{\text{surv}}(u, v; \theta) = u + v - 1 + \exp\left(-\left[(-\ln(1-u))^{\theta} + (-\ln(1-v))^{\theta}\right]^{1/\theta}\right)$$

Où :

- $\theta \geq 1$  est le paramètre de dépendance,
- $C^{\text{surv}}(u, v; \theta)$  modélise la dépendance dans les queues inférieures,

In [120...]

```
# Fonction de répartition cumulative de la copule de survie Gumbel
def survival_gumbel_copula_cdf(u, v, theta):
    ...
```

```

Fonction de répartition cumulative de la copule de survie Gumbel.
u, v : les données uniformes [0, 1].
theta : paramètre de la copule de survie Gumbel.
"""
if theta == 1:
    return u * v # Copule indépendante (cas limite)
t1 = (-np.log(1 - u))**theta
t2 = (-np.log(1 - v))**theta
copula_value = np.exp(-((t1 + t2)**(1 / theta)))
return u + v - 1 + copula_value # Transformation survie

# Normaliser les séries en données uniformes [0, 1]
def normalize_to_uniform(series):
"""
Transforme une série en données uniformes [0, 1] via la fonction ECDF.
"""
ecdf = ECDF(series)
return ecdf(series)

# Transformation des séries en uniformes
u_empirical = normalize_to_uniform(x_wo_date)
v_empirical = normalize_to_uniform(y_wo_date)

# Ajustement du paramètre theta
def survival_gumbel_log_likelihood(params, u, v):
"""
Log-vraisemblance de la copule de survie Gumbel.
params : [theta].
u, v : données uniformes [0, 1].
"""
theta = params[0]
cdf_values = survival_gumbel_copula_cdf(u, v, theta)
return -np.sum(np.log(cdf_values + 1e-9)) # Éviter les problèmes numériques

result = minimize(survival_gumbel_log_likelihood, x0=[1.5], args=(u_empirical, v_empirical))
theta_hat = result.x[0]
print(f"Paramètre theta ajusté : {theta_hat}")

# Calculer la copule de survie Gumbel pour les séries transformées
copula_values = survival_gumbel_copula_cdf(u_empirical, v_empirical, theta_hat)

# Tracer la copule simulée
plt.figure(figsize=(10, 7))
plt.scatter(u_empirical, v_empirical, alpha=0.5, color='purple', label="Copule de survie Gumbel simulée")
plt.title("Copule de survie Gumbel simulée pour x_wo_date et y_wo_date")
plt.xlabel("u (x_wo_date)")
plt.ylabel("v (y_wo_date)")
plt.legend()
plt.grid(True)
plt.show()

# Test de Kolmogorov-Smirnov (KS)
# Comparer les copules simulée et empirique
ks_statistic, ks_pvalue = kstest(

```

```

    copula_values, lambda x: survival_gumbel_copula_cdf(x, x, theta_hat)
)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")

# Calcul des critères AIC et BIC
# Log-vraisemblance pour la copule ajustée
log_likelihood = -survival_gumbel_log_likelihood([theta_hat], u_empirical, v_empirical)
n_params = 1 # Nombre de paramètres (theta)
n_samples = len(u_empirical)

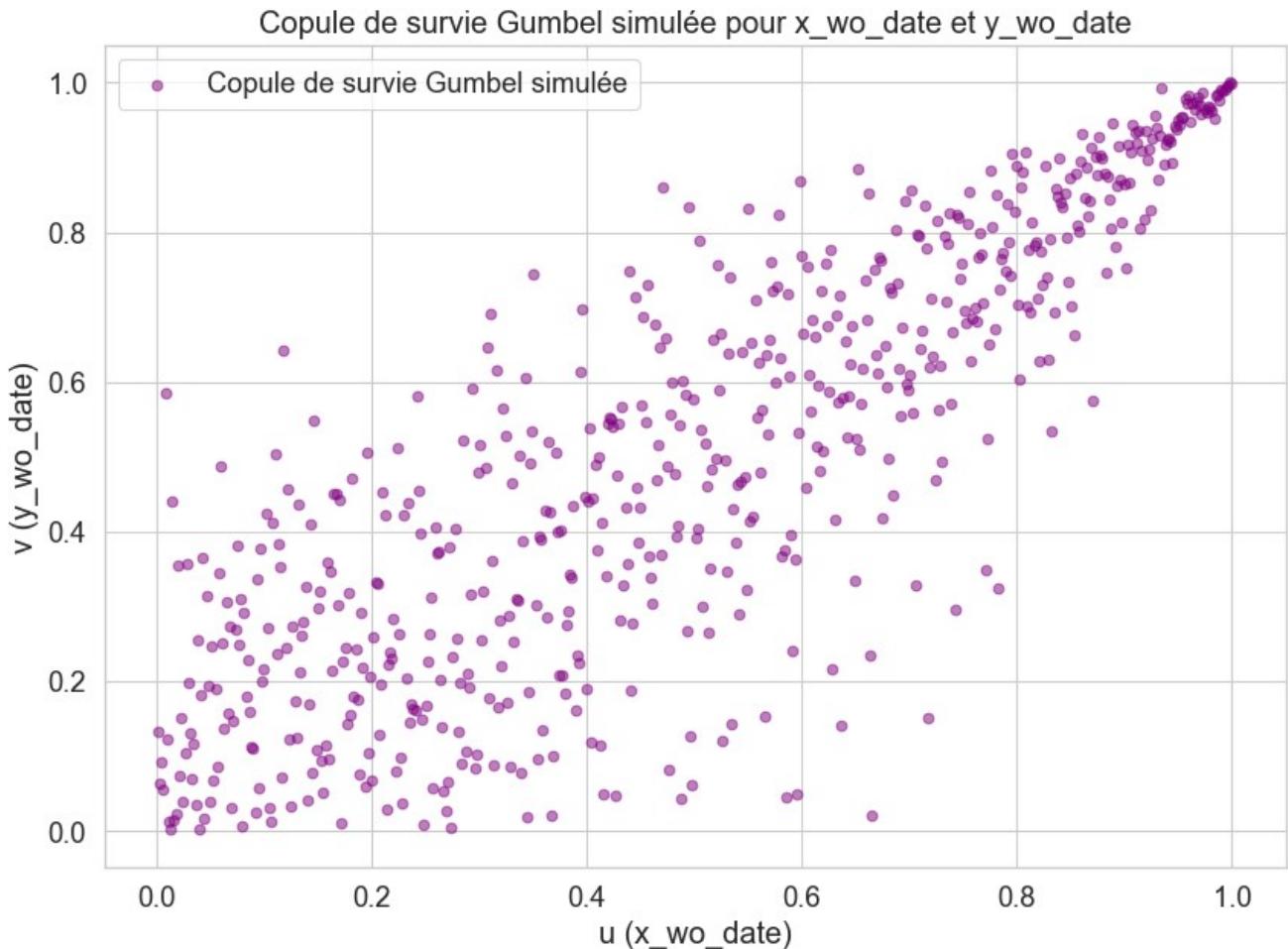
# Calcul de AIC
aic = -2 * log_likelihood + 2 * n_params

# Calcul de BIC
bic = -2 * log_likelihood + n_params * np.log(n_samples)

# Affichage des résultats
print(f"AIC : {aic}")
print(f"BIC : {bic}")

```

Paramètre theta ajusté : 10.0



Statistique KS : 0.12102558277037756

P-valeur KS : 4.322519355716673e-08

AIC : 1489.748359304766

BIC : 1494.1419500587165

## Copule de Survie Joe

La copule de survie Joe est une transformation de la copule de Joe qui modélise la dépendance dans les queues inférieures des distributions. Elle est particulièrement utile pour analyser les relations entre faibles valeurs corrélées des variables.

Pour deux variables  $U$  et  $V$  uniformes dans  $[0, 1]$ , la copule de survie Joe est donnée par :

$$C^{\text{surv}}(u, v; \theta) = u + v - 1 + 1 - (1 - (1 - u)^\theta \cdot (1 - v)^\theta)^{1/\theta}$$

Où :

- $\theta \geq 1$  est le paramètre de dépendance,
- $C^{\text{surv}}(u, v; \theta)$  modélise la dépendance dans les queues inférieures,

In [121...]

```
# Fonction de répartition cumulative de la copule de survie Joe
def survival_joe_copula_cdf(u, v, theta):
    """
    Fonction de répartition cumulative de la copule de survie Joe.
    u, v : les données uniformes [0, 1].
    theta : paramètre de la copule de survie Joe.
    """
    if theta == 1:
        return u * v # Cas limite d'indépendance
    term1 = (1 - u)**theta
    term2 = (1 - v)**theta
    copula_value = 1 - (1 - (1 - term1) * (1 - term2))**(1 / theta)
    return np.clip(u + v - 1 + copula_value, 1e-9, 1) # Transformation survie avec régularisation

# Normaliser les séries en données uniformes [0, 1]
def normalize_to_uniform(series):
    """
    Transforme une série en données uniformes [0, 1] via la fonction ECDF.
    """
    ecdf = ECDF(series)
    return np.clip(ecdf(series), 1e-9, 1 - 1e-9) # Régularisation pour éviter les extrêmes

# Transformation des séries en uniformes
u_empirical = normalize_to_uniform(x_wo_date)
v_empirical = normalize_to_uniform(y_wo_date)

# Ajustement du paramètre theta
def survival_joe_log_likelihood(params, u, v):
    """
    Log-vraisemblance de la copule de survie Joe.
    params : [theta].
    u, v : données uniformes [0, 1].
    """
    theta = params[0]
```

```

cdf_values = survival_joe_copula_cdf(u, v, theta)
return -np.sum(np.log(cdf_values + 1e-9)) # Éviter les problèmes numériques

result = minimize(
    survival_joe_log_likelihood,
    x0=[1.5], # Initialisation de theta
    args=(u_empirical, v_empirical),
    bounds=[(1.01, 10)] # Régularisation des bornes pour éviter theta = 1
)
theta_hat = result.x[0]
print(f"Paramètre theta ajusté : {theta_hat}")

# Calculer la copule de survie Joe pour les séries transformées
copula_values = survival_joe_copula_cdf(u_empirical, v_empirical, theta_hat)

# Vérification des valeurs de la copule
print(f"Copule values range: {copula_values.min()} to {copula_values.max()}")


# Tracer la copule simulée
plt.figure(figsize=(10, 7))
plt.scatter(u_empirical, v_empirical, alpha=0.5, color='purple', label="Copule de survie Joe")
plt.title("Copule de survie Joe simulée pour x_wo_date et y_wo_date")
plt.xlabel("u (x_wo_date)")
plt.ylabel("v (y_wo_date)")
plt.legend()
plt.grid(True)
plt.show()

# Test KS
ks_statistic, ks_pvalue = kstest(
    copula_values, lambda x: survival_joe_copula_cdf(x, x, theta_hat)
)
print(f"Statistique KS : {ks_statistic}")
print(f"P-valeur KS : {ks_pvalue}")


# Calcul des critères AIC et BIC
# Log-vraisemblance pour la copule ajustée
log_likelihood = -survival_joe_log_likelihood([theta_hat], u_empirical, v_empirical)
n_params = 1 # Nombre de paramètres (theta)
n_samples = len(u_empirical)

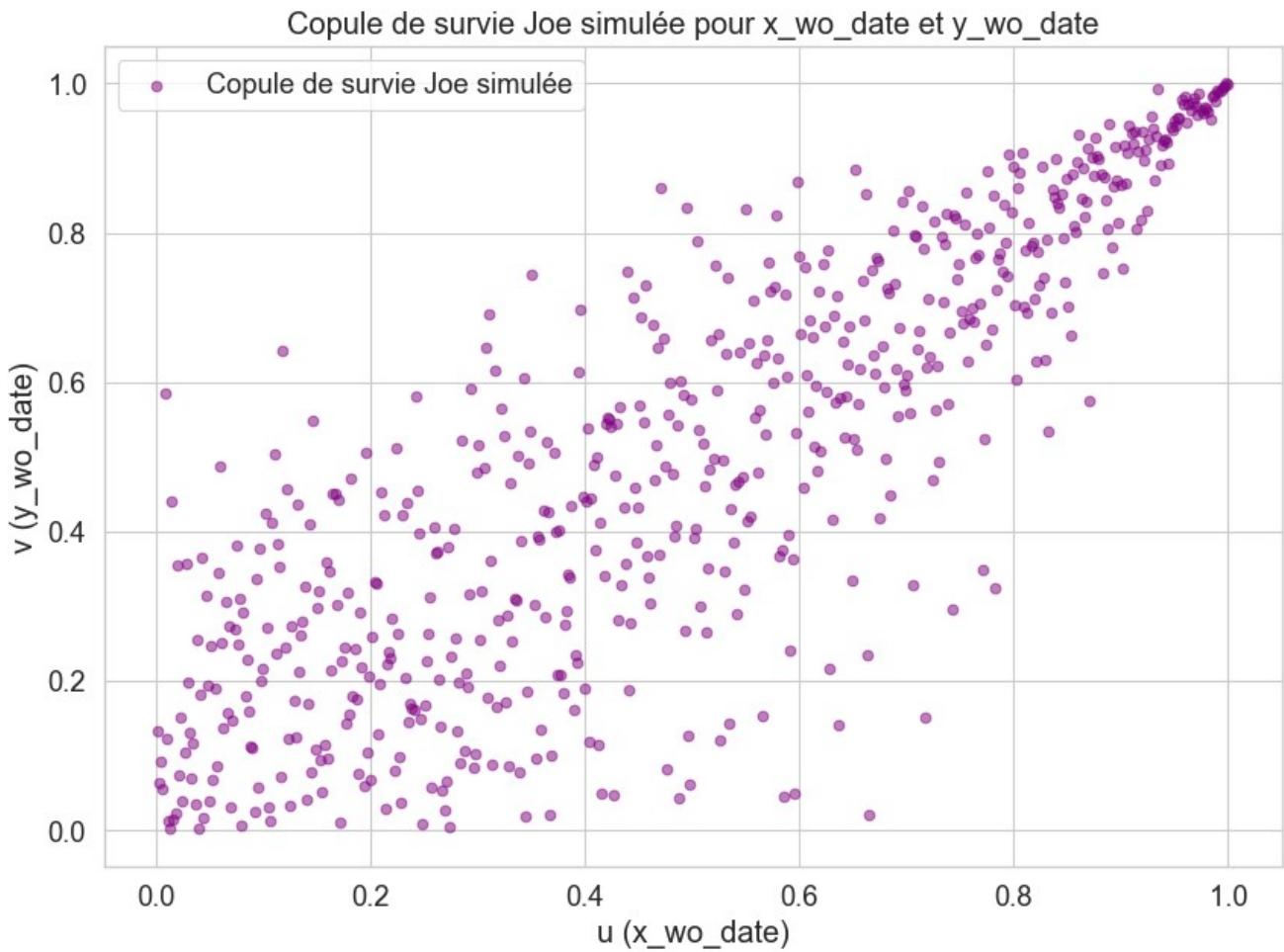
# Calcul de AIC
aic = -2 * log_likelihood + 2 * n_params

# Calcul de BIC
bic = -2 * log_likelihood + n_params * np.log(n_samples)

# Affichage des résultats
print(f"AIC : {aic}")
print(f"BIC : {bic}")

```

Paramètre theta ajusté : 10.0  
Copule values range: 1e-09 to 1.0



Statistique KS : 0.49163879498662205

P-valeur KS : 7.212354659178918e-134

AIC : 9379.336926616445

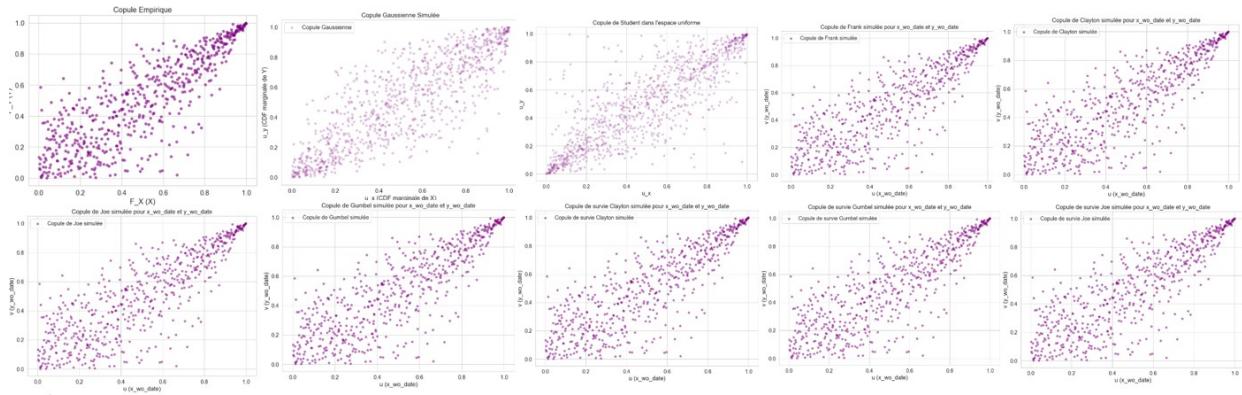
BIC : 9383.730517370395

## Choix de la meilleure copule pour modéliser la dépendance des séries X et Y

Voici les résultats obtenus pour les neufs copules théoriques étudiées précédemment.

	Copule Gaussienne	Copule de Student	Copule de Frank	Copule de Clayton	Copule de Gumbel	Copule de Joe	Copule de Survie Clayton	Copule de Survie Gumbel	Copule de Survie Joe
Statistique KS	0.123	0.023	0.194	0.1303	0.132	0.175	0.461	0.121	0.492
P-Valeur KS	9.70e-14	0.663	3.631	2.565e-9	1.381e-9	1.497e-16	4.035e-117	4.323e-08	7.212e-134
AIC	4477.66	6459.72	1629.65	1501.65	1494.49	1579.75	639.728	1489.75	9379.34
BIC	4482.57	6469.53	1634.04	1506.05	1498.88	1584.14	644.12	1494.14	9383.73

Voici les graphes des neufs copules théoriques et de la copule empirique.



La copule de Gumbel est la plus adaptée pour modéliser la dépendance entre X et Y, car elle offre le meilleur compromis entre les différents critères d'évaluation. Tout d'abord, ses valeurs d'AIC et de BIC sont parmi les plus faibles, ce qui indique un excellent ajustement sans complexité exagérée du modèle. Ensuite, la statistique KS montre une distance raisonnablement faible entre la copule théorique de Gumbel et la copule empirique, renforçant la validité de l'ajustement. Bien que la p-valeur KS soit faible, les résultats globaux restent cohérents avec une bonne qualité d'ajustement. Visuellement, la copule de Gumbel s'aligne étroitement avec la copule empirique, particulièrement dans les queues supérieures, ce qui reflète bien les caractéristiques spécifiques de dépendance dans les données. Par rapport aux autres copules, comme la copule de Student ou la copule de survie Clayton, elle évite les problèmes liés à des AIC/BIC trop élevés ou à des p-valeurs KS trop faibles. En conclusion, la copule de Gumbel capture efficacement la structure de dépendance entre X et Y.

## Calcul de la perte annuelle combinée de X et Y de période de retour 200 ans

Pour calculer la perte annuelle combinée pour X et Y avec une période de retour de 200 ans, nous allons procéder en plusieurs étapes. Tout d'abord, les distributions marginales de X et Y sont modélisées. La série X, représentant des valeurs extrêmes, est ajustée à une loi GPD (Generalized Pareto Distribution) au-delà d'un seuil défini 20 000. La série Y, qui suit la loi normale.

Ensuite, la dépendance entre X et Y est modélisée à l'aide de la copule de Gumbel, qui est particulièrement adaptée pour capturer une dépendance dans les queues supérieures. Le paramètre de dépendance theta de la copule est ajusté en fonction des données.

Pour déterminer les niveaux de retour individuels de X et Y pour une période de 200 ans, nous utilisons les propriétés de leurs lois marginales. Pour X, le niveau de retour est calculé directement à partir de la loi GPD en intégrant la probabilité associée à la période de retour. Pour Y, le niveau de retour est déterminé en utilisant la fonction de répartition inverse (quantile) de la loi normale.

Enfin, pour estimer la perte combinée, nous réalisons une simulation Monte Carlo. Des valeurs uniformes [0, 1] sont générées pour X et Y, puis transformées en fonction de la copule de Gumbel

pour introduire la dépendance. Ces valeurs sont ensuite retransformées dans leurs distributions marginales respectives à l'aide des fonctions quantiles inverses. Les pertes combinées sont calculées comme la somme des pertes simulées pour X et Y. La perte annuelle combinée pour une période de retour de 200 ans est obtenue en prenant le 99.5<sup>e</sup> percentile des pertes combinées simulées, correspondant à une probabilité d'occurrence de 1/200.

In [124...]

```
# Définir les paramètres des distributions marginales pour X et Y

# Paramètres de la loi GPD pour X
# Choix du seuil
threshold_x = 20000
# Calcul des excès
excess = x_wo_date[x_wo_date > threshold_x] - threshold_x
# Ajustement du modèle GPD
shape_x, loc, scale_x = genpareto.fit(excess, loc=0)
print(f"Paramètre de forme (shape): {shape_x}")
print(f"Paramètre d'échelle (scale): {scale_x}")

# Paramètres de la loi normale pour Y
dist = getattr(scipy.stats, "norm")
param = dist.fit(y_wo_date)
## moyenne
mean_y = param[-2]
## écart-type
std_y = param[-1]

#Définir la copule de Gumbel
def gumbel_copula_cdf(u, v, theta):
    """
    Fonction de répartition cumulative de la copule de Gumbel.
    u, v : les données uniformes [0, 1].
    theta : paramètre de la copule de Gumbel.
    """
    if theta == 1:
        return u * v # Copule indépendante (cas limite)
    t1 = (-np.log(u))**theta
    t2 = (-np.log(v))**theta
    return np.exp(-((t1 + t2)**(1 / theta)))

# Ajuster le paramètre theta pour la copule
theta = 10 # Paramètre de dépendance de la copule de Gumbel (déterminé précédemment)

# Calculer les niveaux de retour pour X et Y
# Période de retour
return_period = 200

# Niveau de retour pour X (via la loi GPD)
level_x = threshold_x + scale_x / shape_x * ((return_period * (1 - genpareto.cdf(threshold_x))

# Niveau de retour pour Y (via la loi normale)
prob_y = 1 - 1 / return_period # Probabilité correspondante pour la période de retour
level_y = norm.ppf(prob_y, loc=mean_y, scale=std_y)
```

```

# Calculer la perte annuelle combinée
# Simulation pour approximer la dépendance via la copule
n_simulations = 100000 # Nombre de simulations pour échantillonner
u = np.random.uniform(0, 1, n_simulations) # Uniforme [0, 1] pour X
v = np.random.uniform(0, 1, n_simulations) # Uniforme [0, 1] pour Y

# Appliquer la copule de Gumbel pour modéliser la dépendance
u_gumbel = np.exp(-((-np.log(u))**theta + (-np.log(v))**theta)**(1/theta)) # Transformation
v_gumbel = v # On garde v tel quel pour conserver la symétrie

# Transformation inverse pour obtenir des valeurs simulées de X et Y
x_simulated = threshold_x + genpareto.ppf(u_gumbel, shape_x, scale=scale_x)
y_simulated = norm.ppf(v_gumbel, loc=mean_y, scale=std_y)

# Calculer la perte combinée pour chaque simulation
combined_losses = x_simulated + y_simulated

# Calculer la perte annuelle combinée pour une période de retour de 200 ans
combined_loss_200 = np.percentile(combined_losses, 99.5) # 99.5% correspond à 1/200

# Afficher les résultats
print(f"Pour la période de retour {return_period} ans :")
print(f"Niveau de retour pour X : {level_x:.2f}")
print(f"Niveau de retour pour Y : {level_y:.2f}")
print(f"Perte annuelle combinée (X + Y) : {combined_loss_200:.2f}")

```

Paramètre de forme (shape): 0.693146944942664  
 Paramètre d'échelle (scale): 5542.4481857325545  
 Pour la période de retour 200 ans :  
 Niveau de retour pour X : 101872.45  
 Niveau de retour pour Y : 4702.26  
 Perte annuelle combinée (X + Y) : 66076.86

# Conclusion

Dans le cadre de ce projet, nous avons appliqué différentes techniques de modélisation afin d'analyser des séries de données. Tout d'abord, nous avons comparé plusieurs modèles de distribution, tels que la loi normale, le modèle GEV (Generalized Extreme Value) et le modèle GPD (Generalized Pareto Distribution), pour trouver le modèle qui modélise le mieux les séries de données. Par la suite, nous avons utilisé la théorie des copules pour modéliser la dépendance entre les deux séries, en testant différents types de copules afin d'identifier celles qui décrivent le mieux cette dépendance. Enfin, l'objectif ultime consistait à calculer la perte annuelle combinée pour les deux séries, permettant ainsi d'estimer les impacts des événements extrêmes conjointement.

Ce projet s'est avéré particulièrement enrichissant car il nous a permis de consolider notre compréhension des concepts fondamentaux abordés durant le cours de GRVE. Il nous a également donné l'opportunité de mettre en pratique des notions théoriques complexes, telles que la modélisation des extrêmes et la théorie des copules, dans un contexte appliquéd et concret.

# Bibliographie

- Cours de GRVE du Professeur ROUSSELLE
- [https://www.maths.univ-evry.fr/pages\\_perso/crepey/Credit/copula-gdrm%20Roncalli%20ENSAI.pdf](https://www.maths.univ-evry.fr/pages_perso/crepey/Credit/copula-gdrm%20Roncalli%20ENSAI.pdf)
- [https://perso.univ-lemans.fr/~apopier/enseignement/M2\\_risque\\_credit/slides\\_copule.pdf](https://perso.univ-lemans.fr/~apopier/enseignement/M2_risque_credit/slides_copule.pdf)
- [http://www.actuaries.org/ASTIN/Colloquia/Zurich/Cadoux\\_Loizeau.pdf](http://www.actuaries.org/ASTIN/Colloquia/Zurich/Cadoux_Loizeau.pdf)
- <https://essec.hal.science/file/index/docid/572559/filename/10009.pdf>