# Fiche de révision : CC1 d'Econométrie en Banque-Finance

## Importation des packages

Pandas est une librarie Python largement utilisée pour la manipulation et l'analyse de données. Elle permet de travailler avec des structures de données appelées DataFrames (tableaux avec lignes et colonnes) et Series (colonnes individuelles).

In [1]:
```
import pandas as pd
```

Statsmodels.api nous servira lorsque nous voudrons estimer un modèle économétrique type modèle linéaire avec la méthode des MCO

In [2]:
```
import statsmodels.api as sm
```

matplotlib.pyplot nous servira si on veut faire de graphique que ce soit pour de nuages de point ou pour les résidus de notre modèle

In [3]:
```
import matplotlib.pyplot as plt
```

numpy est une librairie qui sert à faire principalement de l'algèbre, on l'utilise souvent pour estimer un nmodèle linaire multiple

In [4]:
```
import numpy as np
```

Avec la ligne de code suivante on importe de la librairie statsmodels.stats.diagnostic les fonctions het_breuschpagan et het_white qui sont les tests d'hétéroscédasticité de Breusch Pagan et celui de white

In [5]:
```
from statsmodels.stats.diagnostic import het_breuschpagan, het_white
```

Avec la ligne de code suivante on importe de la librairie statsmodels.stats.diagnostic la fonction acorr_breusch_godfrey qui sont les tests d'autocorrelation de erreur de Breusch Godfrey

In [6]:
```
from statsmodels.stats.diagnostic import acorr_breusch_godfrey
```

## Créer un lien avec le dossier où se trouve notre base de donnée

On commence par spécifier un chemin `path` Attention à ce qu'il y ait bien des slash et non back-slash dans votre chemin d'accès. ajouter un "" à la fin du chemin.

Technique plus rapide : on peut doubler les back-slash

In [7]:
```
path = "C:\\Users\\issak\\Documents\\M1\\Econométrie en Banque-Finance\\data_2\\
```

# Question de cours type CC

Pour chaque code suivant il faut commenter

In [8]:
```
a = 1 #Declaration d'une variable num
print(a)
```

```
1
```

Nous créons une variable de type integer appelée a et elle contient l'entier 1. Nous utilisons ensuite la fonction préintégrée de python permettant de l'afficher. L'output de cette cellule de code sera donc d'imprimer 1

In [9]:
```
# 4
# nous supposons que df est un dataframe
df.head()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[9], line 3
      1 # 4
      2 # nous supposons que df est un dataframe
----> 3 df.head()

NameError: name 'df' is not defined
```

La fonction .head() de pandas, appliquée à un dataframe permet d'afficher l'en-tête du dataframe (les 5 premières lignes). Ici ça fonctionne pas car on a pas définit de dataframe

In [14]:
```
# 5
une_liste = [1,2,3,4]

for i in une_liste :
    print(i + 5)
```

```
6
7
8
9
```

Nous créons une variable appelée une_liste et contenant une liste de 4 integer. Puis nous utilisons une boucle for permettant d'afficher les éléments de cette liste +5 (6, 7, 8, 9) un à un.

In [15]:
```
# 6
for i in une_liste :
    if i > 2
        print(i)
```

```
  Cell In[15], line 3
    if i > 2
             ^
SyntaxError: expected ':'
```

Cette boucle semble permettre d'afficher les élements de la liste une_liste qui sont supérieurs à 2 strictement. Cela étant, il manque les ":" après l'instruction if. Un message d'erreur va donc apparaître.

In [16]:
```
# 6
for i in une_liste :
    if i > 2 :
        print(i)
```

```
3
4
```

In [17]:
```
# 7
# df est un dataframe avec 3 variable : "date", "X1" et "X2"
df["X1_gr"] = 100 * ( df["X1"]/df["X1"].shift(1) - 1 )
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[17], line 3
      1 # 7
      2 # df est un dataframe avec 3 variable : "date", "X1" et "X2"
----> 3 df["X1_gr"] = 100 * ( df["X1"]/df["X1"].shift(1) - 1 )

NameError: name 'df' is not defined
```

Cette cellule permet de créer une nouvelle variable dans le dataframe df. Cette variable est égale au taux de croissance de X1. La fonction .shift(1) de pandas permet de décaler d'une ligne (donc d'une période) les éléments de df["X1"]. Ici ça fonctionne pas car on a pas définit de dataframe

In [18]:
```
# 8
# df est un dataframe avec 3 variable : "date", "X1" et "X2"
df.plot(x = "date",y="X2", title="X2", legend=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[18], line 3
      1 # 8
      2 # df est un dataframe avec 3 variable : "date", "X1" et "X2"
----> 3 df.plot(x = "date",y="X2", title="X2", legend=True)

NameError: name 'df' is not defined
```

Cette cellule de code permet d'afficher le graphique ligne de X2 en ordonnée en fonction des dates en abscisses. Le graphique aura un titre "X2" et affichera une légende. Ici ça fonctionne pas car on a pas définit de dataframe

In [19]:
```
# 9
import statsmodels.api as sm


X = df["X1"].dropna()


y = df["X2"].dropna()

modele = sm.OLS(y,X)
```

```
resultats = modele.fit()

print(resultats.summary())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[19], line 5
      1 # 9
      2 import statsmodels.api as sm
----> 5 X = df["X1"].dropna()
      8 y = df["X2"].dropna()
     10 modele = sm.OLS(y,X)

NameError: name 'df' is not defined
```

Dans cette cellule, le package statsmodels.api est importé. C'est le package que nous utilisons pour mener des estimations et tests économétriques. Nous créons ensuite une matrice X (contenant la variable X1) et une autre matrice y (contenant la variable X2). Nous définissons ensuite un modèle MCO $(X_2 = \beta X_1 + \epsilon)$. On remarque que la constante a été oubliée. Nous estimons ensuite le modèle par les MCO et stockons les résultats dans une variable appelée resultats. Enfin, nous utilisons la fonction print pour afficher un résumé des résultats d'estimation MCO.Ici ça fonctionne pas car on a pas définit de dataframe

# Importation d'une base de donnée

On rappelle le chemin mais si on a bien excécuté la celulle avant il devrait pas y avoir de problème

In [20]:
```
path = "C:\\Users\\issak\\Documents\\M1\\Econométrie en Banque-Finance\\data_2\\
```

In [21]:
```
# Pour un document txt

df = pd.read_csv(path + "capm2.txt", sep=";")

# Pour un document csv

df_2 = pd.read_csv(path + "bigdata.csv", sep=";")

# Pour un document xlsx

df_3 = pd.read_excel(path + "bigdata.xlsx")
```

On créer une variable df (pour dataframe) et on utilise la fonction `read_csv` qu'on a appelé grâce à pandas sous la forme `pd.` pour lire la base de donnée capm2.txt. On rajoute également un séparateur pour bien voir les donnée et on a le résultat suivant

In [22]:
```
df.head()
```

Out[22]:

| | date | ford | sandp | ustb3m | oil |
|---|---|---|---|---|---|
| **0** | 2001-01-01 | 15.454666 | 1366.010010 | 4.84 | 28.700001 |
| **1** | 2001-02-01 | 15.411097 | 1239.939941 | 4.72 | 27.420000 |
| **2** | 2001-03-01 | 15.582887 | 1160.329956 | 4.18 | 26.400000 |
| **3** | 2001-04-01 | 16.336536 | 1249.459961 | 3.83 | 26.400000 |
| **4** | 2001-05-01 | 13.631827 | 1255.819946 | 3.54 | 28.370001 |

## Affichage d'un certain nombre d'observation par exemple $x = 10$

In [39]:
```
df.head(10)
```

Out[39]:

| | date | ford | sandp | ustb3m | oil | rsandp | rford | |
|---|---|---|---|---|---|---|---|---|
| **0** | 2001-01-01 | 15.454666 | 1366.010010 | 4.84 | 28.700001 | NaN | NaN | |
| **1** | 2001-02-01 | 15.411097 | 1239.939941 | 4.72 | 27.420000 | -9.229074 | -0.281915 | -4.45 |
| **2** | 2001-03-01 | 15.582887 | 1160.329956 | 4.18 | 26.400000 | -6.420471 | 1.114716 | -3.71 |
| **3** | 2001-04-01 | 16.336536 | 1249.459961 | 3.83 | 26.400000 | 7.681436 | 4.836389 | 0.00 |
| **4** | 2001-05-01 | 13.631827 | 1255.819946 | 3.54 | 28.370001 | 0.509019 | -16.556196 | 7.46 |
| **5** | 2001-06-01 | 13.743798 | 1224.380005 | 3.56 | 26.250000 | -2.503539 | 0.821394 | -7.47 |
| **6** | 2001-07-01 | 14.258834 | 1211.229980 | 3.44 | 26.250000 | -1.074015 | 3.747407 | 0.00 |
| **7** | 2001-08-01 | 11.123795 | 1133.579956 | 3.28 | 27.200001 | -6.410841 | -21.986644 | 3.61 |
| **8** | 2001-09-01 | 9.828797 | 1040.939941 | 2.30 | 23.430000 | -8.172341 | -11.641692 | -13.86 |
| **9** | 2001-10-01 | 9.092343 | 1059.780029 | 2.01 | 21.180000 | 1.809911 | -7.492819 | -9.60 |

## Connaitre la taille de la base de donnée sans l'ouvrir

In [41]:
```
df_3.shape
```

Out[41]:
```
(200000, 2)
```

In [42]:
```
conda install -c conda-forge jupyter_contrib_nbextensions
```

```
^C
```

```
Note: you may need to restart the kernel to use updated packages.
```

C'est très utile, ici la base de donnée est tellement lourde que ça aurait beaucoup de temps à Python de l'afficher.

# Création de variable taux de croissance

Pour calculer les différentes variables suivantes :

- $rsandp_t = 100 \times (sandp_t / sandp_{t-1} - 1)$
- $rford_t = 100 \times (ford_t / ford_{t-1} - 1)$
- $roil = 100 \times (oil_t / oil_{t-1} - 1)$
- $ersandp = rsandp - ustb3m$
- $erford = rford - ustb3m$

pour rappel, créer une nouvelle variable dans un dataframe à partir d'une variable déjà contenue par ce dataframe se fait comme suit (par exemple la différence entre deux var X_1 et X_2),

```
df[ " nom_de_la_nouvelle_var " ] = df[ " X_1 " ] - df[ " X_2 " ]
```

```
    pour avoir une variable décalée d'une période (en t-1, on
    utilise la fonction .shft(-1)) :
```

```
df[ "X" ].shift(1)
```

In [23]:
```
df["rsandp"] = 100 * ( df["sandp"]/df["sandp"].shift(1) - 1 )
df["rford"] = 100 * ( df["ford"]/df["ford"].shift(1) - 1 )


df["roil"] = 100 * ( df["oil"]/df["oil"].shift(1) - 1 )


df["ersandp"] = df["rsandp"] - df["ustb3m"]
df["erford"] = df["rford"] - df["ustb3m"]

df.head()
```

Out[23]:

| | date | ford | sandp | ustb3m | oil | rsandp | rford | |
|---|---|---|---|---|---|---|---|---|
| **0** | 2001-01-01 | 15.454666 | 1366.010010 | 4.84 | 28.700001 | NaN | NaN | N |
| **1** | 2001-02-01 | 15.411097 | 1239.939941 | 4.72 | 27.420000 | -9.229074 | -0.281915 | -4.459 |
| **2** | 2001-03-01 | 15.582887 | 1160.329956 | 4.18 | 26.400000 | -6.420471 | 1.114716 | -3.719 |
| **3** | 2001-04-01 | 16.336536 | 1249.459961 | 3.83 | 26.400000 | 7.681436 | 4.836389 | 0.000 |
| **4** | 2001-05-01 | 13.631827 | 1255.819946 | 3.54 | 28.370001 | 0.509019 | -16.556196 | 7.462 |

# Création d'une variable dichotomiques

Avec ce code on créer une variable dichotomiques qui prend soit la valeur 1 pour signifier la présence d'un phénomène soit la valeurs 0 pour signifier son absence.

In [24]:
```
l_crisis = []
date_crisis = ["2007-07-01","2007-08-01","2007-09-01","2007-10-01","2007-11-01",
for i in df.date :
    if i in date_crisis :
        l_crisis.append(1)
    else :
        l_crisis.append(0)
```

```
l_covid = []
date_covid = ["2020-01-01","2020-02-01","2020-03-01","2020-04-01","2020-05-01","
for i in df.date :
    if i in date_covid :
        l_covid.append(1)
    else :
        l_covid.append(0)

df["d_crisis"] = l_crisis
df["d_covid"] = l_covid

df
```

Out[24]:

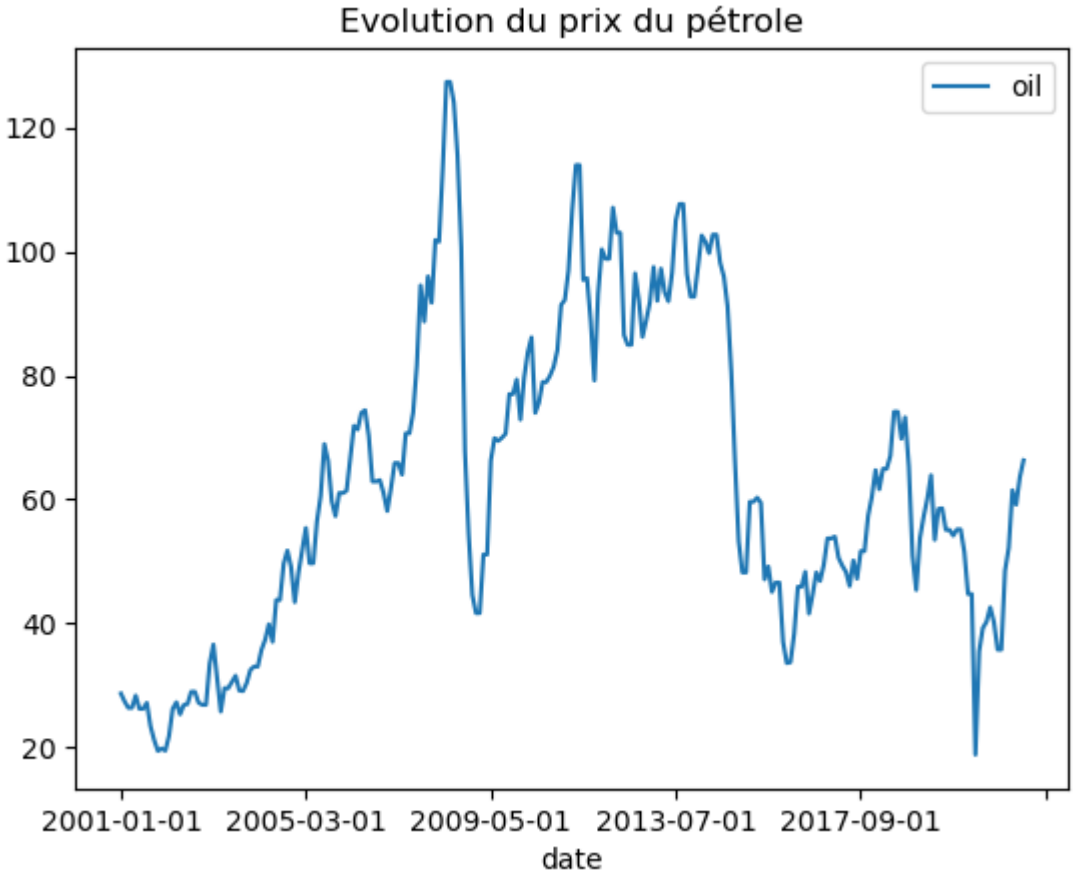
| | date | ford | sandp | ustb3m | oil | rsandp | rford | |
|---|---|---|---|---|---|---|---|---|
| **0** | 2001-01-01 | 15.454666 | 1366.010010 | 4.840 | 28.700001 | NaN | NaN | |
| **1** | 2001-02-01 | 15.411097 | 1239.939941 | 4.720 | 27.420000 | -9.229074 | -0.281915 | -4.4 |
| **2** | 2001-03-01 | 15.582887 | 1160.329956 | 4.180 | 26.400000 | -6.420471 | 1.114716 | -3.7 |
| **3** | 2001-04-01 | 16.336536 | 1249.459961 | 3.830 | 26.400000 | 7.681436 | 4.836389 | 0.0 |
| **4** | 2001-05-01 | 13.631827 | 1255.819946 | 3.540 | 28.370001 | 0.509019 | -16.556196 | 7.4 |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **240** | 2021-01-01 | 10.530000 | 3714.239990 | 0.048 | 52.200001 | -1.113666 | 19.795222 | 7.5 |
| **241** | 2021-02-01 | 11.700000 | 3811.149902 | 0.035 | 61.500000 | 2.609145 | 11.111111 | 17.8 |
| **242** | 2021-03-01 | 12.250000 | 3972.889893 | 0.013 | 59.160000 | 4.243863 | 4.700855 | -3.8 |
| **243** | 2021-04-01 | 11.540000 | 4181.169922 | 0.003 | 63.580002 | 5.242532 | -5.795918 | 7.4 |
| **244** | 2021-05-01 | 14.530000 | 4204.109863 | 0.008 | 66.320000 | 0.548649 | 25.909879 | 4.3 |

245 rows × 12 columns

# Création de graphique

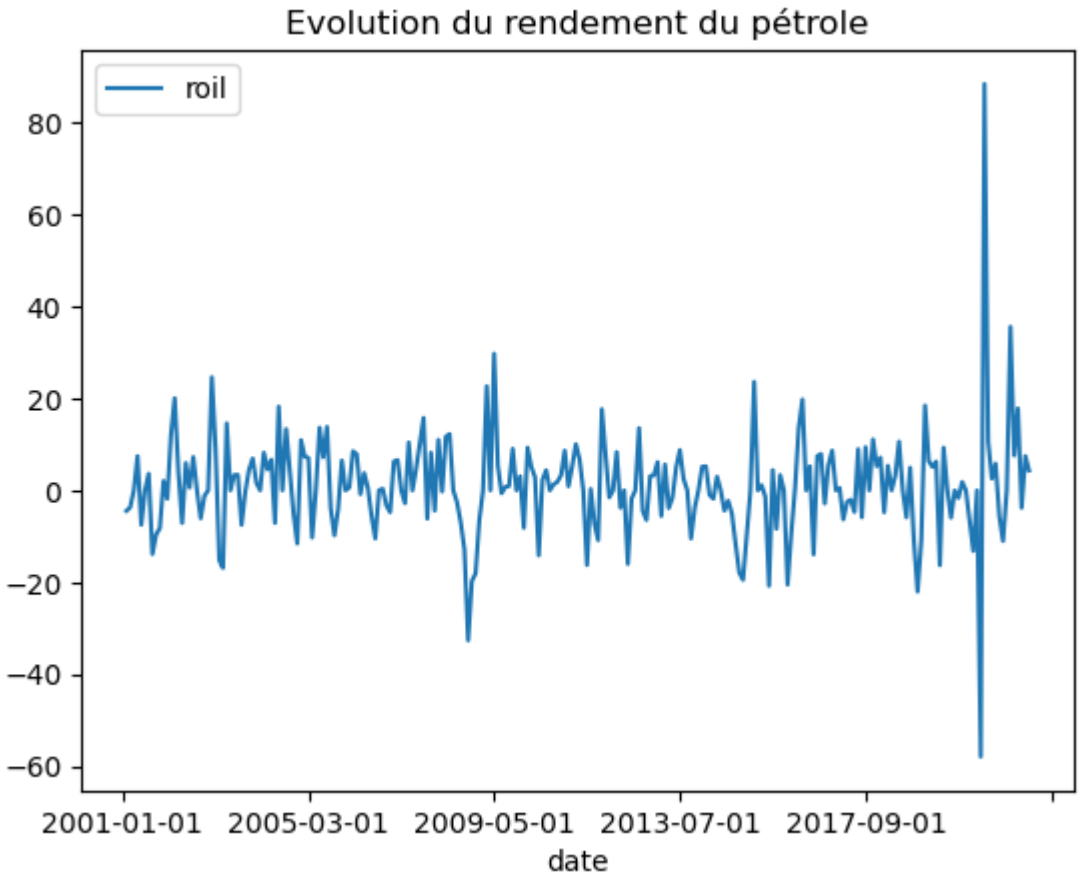
In [25]:
```
# en niveau
# ustb3m avec les options
df.plot(x = "date",y="oil", title="Evolution du prix du pétrole", legend=True)
```

Out[25]:
```
<Axes: title={'center': 'Evolution du prix du pétrole'}, xlabel='date'>
```



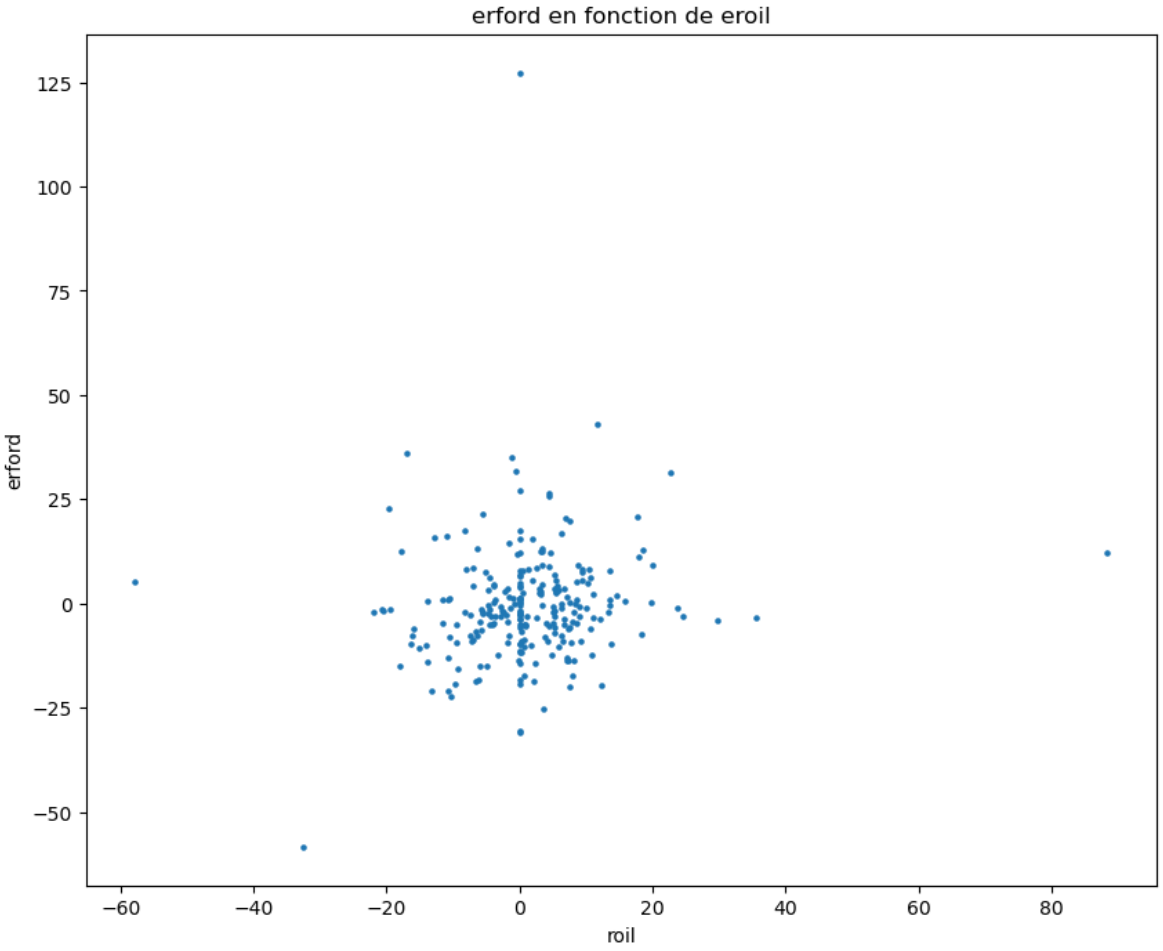

In [26]:
```
# en niveau
# ustb3m avec les options
df.plot(x = "date",y="roil", title="Evolution du rendement du pétrole", legend=T
```

Out[26]:
```
<Axes: title={'center': 'Evolution du rendement du pétrole'}, xlabel='date'>
```



In [27]:
```
# Nuage de points avec options
df.plot.scatter(x="roil",y="erford", title="erford en fonction de eroil", s=5, f
#OUTLIER
```

Out[27]:
```
<Axes: title={'center': 'erford en fonction de eroil'}, xlabel='roil', ylabel
='erford'>
```




Commentaires :

- on remarque que les prix du pétrole ont fortement variés sur la période.
- on peut remarquer la présence des crises (fin des années 2000 et covid) sont très visibles. Les variations du pétrole sont particulièrement touchées par la crise du covid (même si
- il n'y a pas de relation visible directe entre les rendements sur le marché du pétrole et le rendement de l'actif de Ford. Déjà, ce graphique peut laisser présager d'une faible dépendance de la variable expliquée au pétrole.

# Création de la matrice des coefficients de corrélation entre les variables

In [28]:
```
# rford simple
df[["erford","ersandp","roil","d_crisis","d_covid"]].corr()
# matrice de corrélation : a partir de 40 forte
# Pour les binaires interpreter juste le signe
```

Out[28]:

| | erford | ersandp | roil | d_crisis | d_covid |
|---|---|---|---|---|---|
| **erford** | 1.000000 | 0.548571 | 0.132862 | -0.093048 | -0.059061 |
| **ersandp** | 0.548571 | 1.000000 | 0.195503 | -0.114105 | -0.004164 |
| **roil** | 0.132862 | 0.195503 | 1.000000 | 0.063149 | 0.036325 |
| **d_crisis** | -0.093048 | -0.114105 | 0.063149 | 1.000000 | -0.025105 |
| **d_covid** | -0.059061 | -0.004164 | 0.036325 | -0.025105 | 1.000000 |

Commentaires :

- On regarde d'une part les corrélations linéaires entre les variables explicatives et la variable expliquée.
    - positif et fort pour le rendement espéré de SP500
    - positif mais plus faible pour le pétrole (consistent avec le commentaire sur le nuage de points)
    - négatif pour les deux dummy : normalement, on évite de regarder une corrélation linéaire entre une dummy et une variable mais là, nous pouvons voir qu'il semblerait y avoir une relation négative ce qui n'est pas inintéressant : pendant les périodes de crise et de covid, l'indice de ford semble avoir eu des taux de croissance négatifs.
- On peut ensuite regarder les coefficients entre les variables explicatives : l'idée ici est de se faire une idée quant à la potentielle présence de multicolinéarité. S'il en est, elle devrait être entre le rendement du SP500 et celui du prix du pétrole.

# Estimation du modèle
$$\mathbb{E}(R_i) - r = \alpha + \beta_1 \times (\mathbb{E}(R_m) - r) + \epsilon$$

In [29]:
```
X = df["ersandp"].dropna()
X = sm.add_constant(X)

y = df["erford"].dropna()

modele = sm.OLS(y,X)

resultats = modele.fit()

print(resultats.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 erford   R-squared:                       0.301
Model:                            OLS   Adj. R-squared:                  0.298
Method:                 Least Squares   F-statistic:                     104.2
Date:                Wed, 16 Oct 2024   Prob (F-statistic):           1.42e-20
Time:                        17:36:18   Log-Likelihood:                -951.59
No. Observations:                 244   AIC:                             1907.
Df Residuals:                     242   BIC:                             1914.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.7841      0.778      1.008      0.314      -0.748       2.316
ersandp        1.6718      0.164     10.207      0.000       1.349       1.994
==============================================================================
Omnibus:                      234.681   Durbin-Watson:                   2.214
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             9390.427
Skew:                           3.610   Prob(JB):                         0.00
Kurtosis:                      32.521   Cond. No.                         4.81
==============================================================================

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

Prob (F-statistic): 2.82e-11 < 0.05 le modèle est globalement significatif

R-squared: 0.301 : $X$ arrive à explique 30% de la variance de $Y$

ersandp : p value : 0.000 < 0.05 significatif au seuil de 5%

In [30]:
```
X = df[["ersandp","roil","d_crisis","d_covid"]].dropna()
X = sm.add_constant(X)

y = df["erford"].dropna()

modele2 = sm.OLS(y,X)

resultats2 = modele2.fit()

print(resultats2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 erford   R-squared:                       0.306
Model:                            OLS   Adj. R-squared:                  0.295
Method:                 Least Squares   F-statistic:                     26.37
Date:                Wed, 16 Oct 2024   Prob (F-statistic):           4.03e-18
Time:                        17:36:19   Log-Likelihood:                -950.67
No. Observations:                 244   AIC:                             1911.
Df Residuals:                     239   BIC:                             1929.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.9343      0.801      1.167      0.245      -0.643       2.512
ersandp        1.6397      0.169      9.710      0.000       1.307       1.972
roil           0.0405      0.070      0.580      0.562      -0.097       0.178
d_crisis      -3.2457      5.028     -0.646      0.519     -13.150       6.659
d_covid       -5.4344      4.979     -1.091      0.276     -15.244       4.375
==============================================================================
Omnibus:                      237.294   Durbin-Watson:                   2.237
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             9781.267
Skew:                           3.664   Prob(JB):                         0.00
Kurtosis:                      33.139   Cond. No.                         75.1
==============================================================================

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

Commentaires :

- Sur le modèle seul :
  - on remarque que seul la variable associée au rendement du marché est significative (ce à quoi on pouvait s'attendre pour `oil` toujours en nous basant sur le nuage de point). L'absence d'effet pour les crise est plus étonnant.
  - le modèle est globalement significatif
- En comparant avec le premier modèle :
  - $R^2$ du même ordre. Pour comparer on regarde le coefficient de détermination ajusté (on peut d'ailleurs remarquer que l'introduction de nouvelles variables dans le modèle a naturellement fait grimper le $R^2$ non ajusté).
  - les critères d'information AIC et BIC suggèrent aussi que le premier modèle est le meilleur (même s'ils sont très proche).
- Ainsi, si aucun coefficient aujouté n'est significatif et que le modèle n'est pas amélioré en termes de pouvoir explicatif, il n'y a pas de raison de le préserver. Regardons tout de même la présence d'autocorrélation / hétéroscédasticité dans le second modèle.

# Test de d'autocorrélation et hétéroscédasticité

In [31]:
```
# on récupère le résidu du deuxième modèle
```

```
residus2 = resultats2.resid
```

## Hétéroscédasticité

In [32]:
```
# hétéroscedasticité
# H0 : homoscédasticité vs HA : hétéroscédasticité

# Si la p-value est supérieure à 0,05 : On ne rejette pas H0, homoscédasticité
# Si la p-value est inférieure ou égale à 0,05 : On rejette H0, hétéroscédastici

stat1 = het_breuschpagan(residus2, X)
print(stat1[1]) # mettre [1] pour avoir la p-valeur des test
```

```
0.3594502177621543
```

0.3594502177621543 > 0.05 selon ce test nous sommes en présence d'homoscédastique

In [33]:
```
stat2 = het_white(residus2, X)

# H0 : homoscédasticité vs HA : hétéroscédasticité

# Si la p-value est supérieure à 0,05 : On ne rejette pas H0, homoscédasticité
# Si la p-value est inférieure ou égale à 0,05 : On rejette H0, hétéroscédastici


print(stat2[1]) # mettre [1] pour avoir la p-valeur des test
```

```
0.025161576899813128
```

0.025161576899813128 < 0.05 selon ce test nous sommes en présence d'hétéroscédasticité

Lorsque deux tests se contredisent, nous retenons le résultats le plus "pessimiste" : nous aurons donc présomption d'hétéroscédasticité dans notre modèle.

## Autocorrélation

In [34]:
```
stat3 = acorr_breusch_godfrey(resultats, nlags=10)

# Pour le test de breusch godfrey d'autocorrelation
# H0 : représente ce qu'on veut avoir donc
# H0 : non autocorrélation vs HA : Autocorrélation

# Si p-value ≤ 0.05 rejet de H0 DONC Autocorrélation
# Si p-value > 0.05 non rejet de H0 DONC non Autocorrélation

print(stat3[1])
```

```
0.0003779461753214761
```

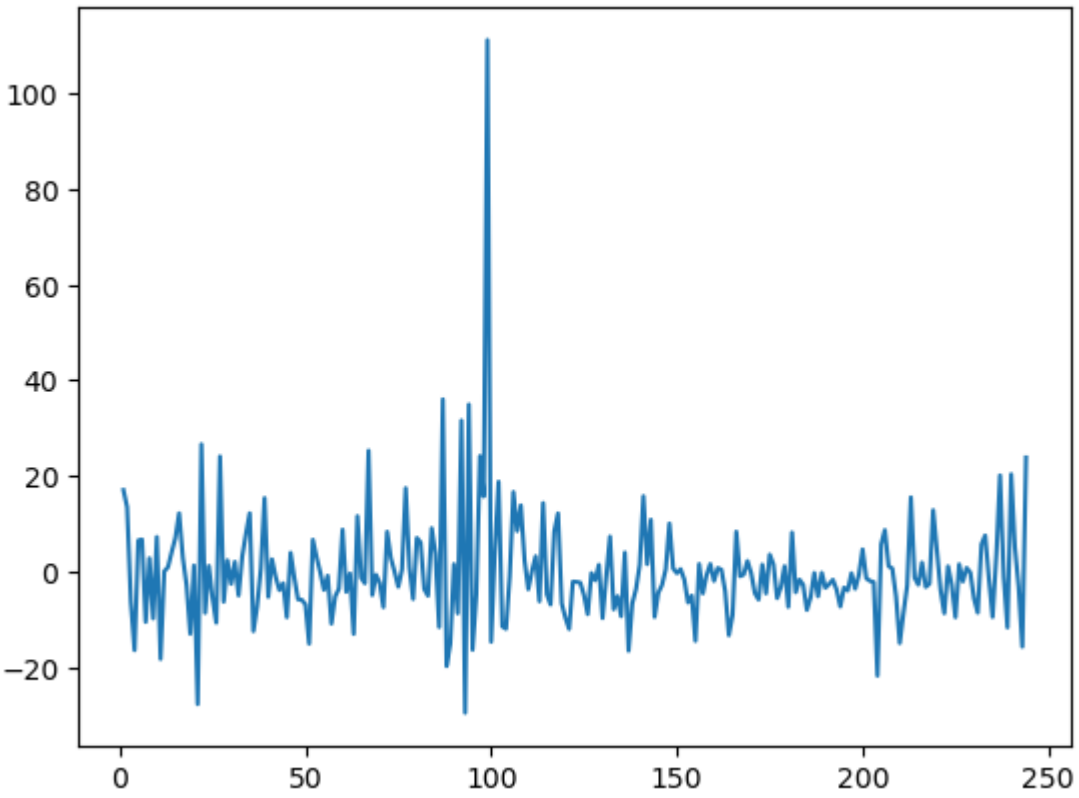
0.0003779461753214761 < 0.05 selon ce test nous sommes en présence d'autocorrélation

Conclusion : nous devons procéder à une correction de Newey-West

# Correction de Newey-West

In [35]:
```
plt.plot(residus2)
```

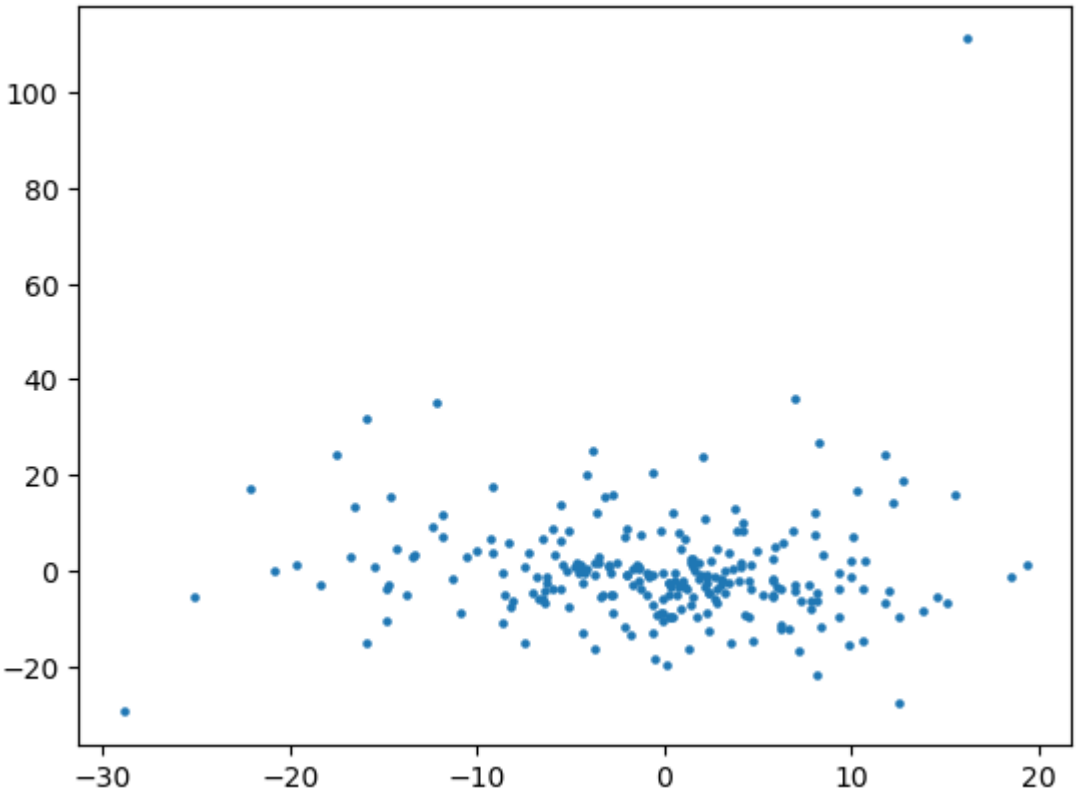
Out[35]:
```
[<matplotlib.lines.Line2D at 0x279dd5343d0>]
```




In [36]:
```
ford_chapeau = resultats2.predict()


plt.scatter(x=ford_chapeau, y=residus2, s=5 )
```

Out[36]:
```
<matplotlib.collections.PathCollection at 0x279dd5c1b10>
```




Commentaires :

- l'idée est de repérer ici quelles sont les variables dont on fait le graphique et de comprendre pourquoi.
- le premier graphique montre l'évolution dans le temps du résidu issu du deuxième modèle : on peut constater que la variance n'apparait pas constante. Nous avons donc raison d'avoir présomption d'hétéroscédasticité
- nou pouvons également constater qu'il y a des "cluster" / regroupements dans la volatilité du résidu : cela peut dénoter la présence d'autocorrélation
- le second graphique montre un nuage de points entre les valeurs prédites par le modèle pour la variable expliquée et le résidu. Une légère relation décroissante apparait entre les deux. Elle n'est cela étant pas assez forte pour que cela puisse confirmer ni infirmer ce que nous avons précédemment établi.

In [37]:
```
resultat_robust2 = resultats2.get_robustcov_results(cov_type='HAC', maxlags=10)
print(resultat_robust2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 erford   R-squared:                       0.306
Model:                            OLS   Adj. R-squared:                  0.295
Method:                 Least Squares   F-statistic:                     23.54
Date:                Wed, 16 Oct 2024   Prob (F-statistic):           2.01e-16
Time:                        17:36:28   Log-Likelihood:                -950.67
No. Observations:                 244   AIC:                             1911.
Df Residuals:                     239   BIC:                             1929.
Df Model:                           4
Covariance Type:                  HAC
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.9343      0.928      1.007      0.315      -0.894       2.762
ersandp        1.6397      0.248      6.623      0.000       1.152       2.127
roil           0.0405      0.047      0.867      0.387      -0.052       0.133
d_crisis      -3.2457      1.053     -3.081      0.002      -5.321      -1.170
d_covid       -5.4344      1.323     -4.108      0.000      -8.040      -2.828
==============================================================================
Omnibus:                      237.294   Durbin-Watson:                   2.237
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             9781.267
Skew:                           3.664   Prob(JB):                         0.00
Kurtosis:                      33.139   Cond. No.                         75.1
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity and autocorrelation robust (HAC) using
10 lags and without small sample correction
```

Commentaires :

- Cette fois, les variables dummy sont significatives
- La variable du pétrole reste non significative (même si la p-valeur qui lui est associée a diminué).
- Il semblerait que les moments de crise ont eu un impact négatif sur la rentabilité espérée de Ford.
- Ce modèle apparait donc comme intéressant dans la mesure où il n'est que très peu moins précis que le modèle initial.

Commentaires :

- On a créé un sous dataframe contenant les varibales explicatives et le résidu.
- Nous avons ensuite afficher la table de corrélation de ce dataframe.
- Nous nous intéressons donc à la dernière colonne qui établie la corrélation entre les variables explicatives et le résidu.
- Nous constatons que ces corrélations sont faibles (très proches de $0$). Cela vient conforter l'une des hypothèses du modèle MCO :
  - $X$ est non aléatoire : $E(\epsilon_t|X) = 0$
  - on rappelle que cette hypothèse assure la convergence de l'estimateur.

In [38]:
```
df_endo = X[list(X)[1::]]
df_endo["residus"] = residus2
```

```
df_endo.corr()
```

Out[38]:

| | ersandp | roil | d_crisis | d_covid | residus |
|---|---|---|---|---|---|
| **ersandp** | 1.000000e+00 | 1.955025e-01 | -1.141052e-01 | -4.163571e-03 | -1.008652e-16 |
| **roil** | 1.955025e-01 | 1.000000e+00 | 6.314855e-02 | 3.632537e-02 | 2.864035e-16 |
| **d_crisis** | -1.141052e-01 | 6.314855e-02 | 1.000000e+00 | -2.521008e-02 | 4.638211e-17 |
| **d_covid** | -4.163571e-03 | 3.632537e-02 | -2.521008e-02 | 1.000000e+00 | 3.305959e-17 |
| **residus** | -1.008652e-16 | 2.864035e-16 | 4.638211e-17 | 3.305959e-17 | 1.000000e+00 |

Commentaires :

- On a créé un sous dataframe contenant les varibales explicatives et le résidu.
- Nous avons ensuite afficher la table de corrélation de ce dataframe.
- Nous nous intéressons donc à la dernière colonne qui établie la corrélation entre les variables explicatives et le résidu.
- Nous constatons que ces corrélations sont faibles (très proches de $0$). Cela vient conforter l'une des hypothèses du modèle MCO :
  - $X$ est non aléatoire : $E(\epsilon_t|X) = 0$
  - on rappelle que cette hypothèse assure la convergence de l'estimateur.