Projet Série Temporelles

AHOUMENOU Onel, DEVIGNAC Vladimir, DURAND Arnaud

Packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#!pip install scikit-learn
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.api import qqplot
from statsmodels.tsa.arima.model import ARIMA
```

Introduction

Dans ce projet, nous allons illustrer sur un jeu de données concret les méthodes étudiées au cours. Nos données proviennent du site kaggle https://www.kaggle.com/datasets/aminesnoussi/air-pollution-dataset. Il traite de la pollution de l'air à Pekin entre le 01 Janvier 2010 et le 31 Décembre 2014.

L'objectif est donc de prédire la pollution futur à l'aide des données passées en suivant le plan suivant:

1. Pré-traitement des données
2. Gestion de la non-stationnarité
3. Identification du modèle probabiliste
4. Estimation des paramètres du modèle
5. Prédiction des valeurs futures
6. Evaluation de la précision de prédiction

Pré-traitement des données

Analyse de la base de données

```
# Importation des données

data = pd.read_csv("air_pollution.csv", index_col='date',
```

```
parse_dates=True)
data.head()
```

```
            pollution_today        dew       temp         press
wnd_spd  \
date

2010-01-02       145.958333  -8.500000  -5.125000  1024.750000
24.860000
2010-01-03        78.833333 -10.125000  -8.541667  1022.791667
70.937917
2010-01-04        31.333333 -20.875000 -11.500000  1029.291667
111.160833
2010-01-05        42.458333 -24.583333 -14.458333  1033.625000
56.920000
2010-01-06        56.416667 -23.708333 -12.541667  1033.750000
18.511667

                  snow  rain  pollution_yesterday
date
2010-01-02   0.708333   0.0            10.041667
2010-01-03  14.166667   0.0           145.958333
2010-01-04   0.000000   0.0            78.833333
2010-01-05   0.000000   0.0            31.333333
2010-01-06   0.000000   0.0            42.458333
```

Isolation de la variable cible "pollution_today"

```
pollution = data['pollution_today']

# Nombre de valeurs manquantes

pollution.isna().sum()
```

```
0
```

Il n'y a aucune valeur manquantes donc pas aucune valeur à estimer avant de commencer le traitement des données

```
#  Quelques statistiques utiles

pollution.describe()
```

```
count    1825.000000
mean       98.245080
std        76.807697
min         3.166667
25%        42.333333
50%        79.166667
75%       131.166667
```

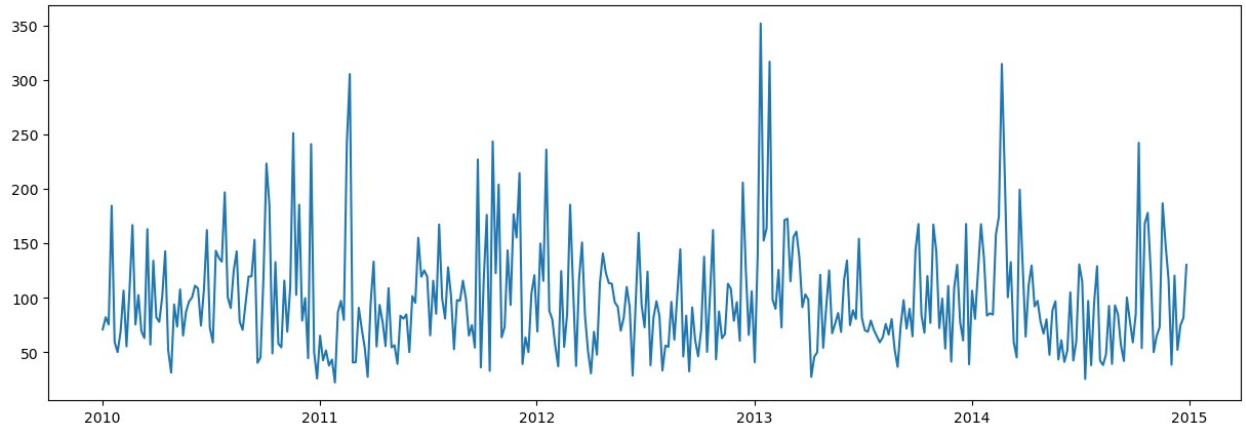
```
max       541.895833
Name: pollution_today, dtype: float64
```

Affichage de la série temporelle

Afin de faciliter nos analyses tout en gardant une quantité de données satisfaisante, nous allons travailler sur des moyennes de 5 jours

```
# Rééchantillonage sur 5 jours
y = pollution.resample('5D').mean()
x = y.index

plt.figure(figsize=(15, 5))
plt.plot(x,y)
```

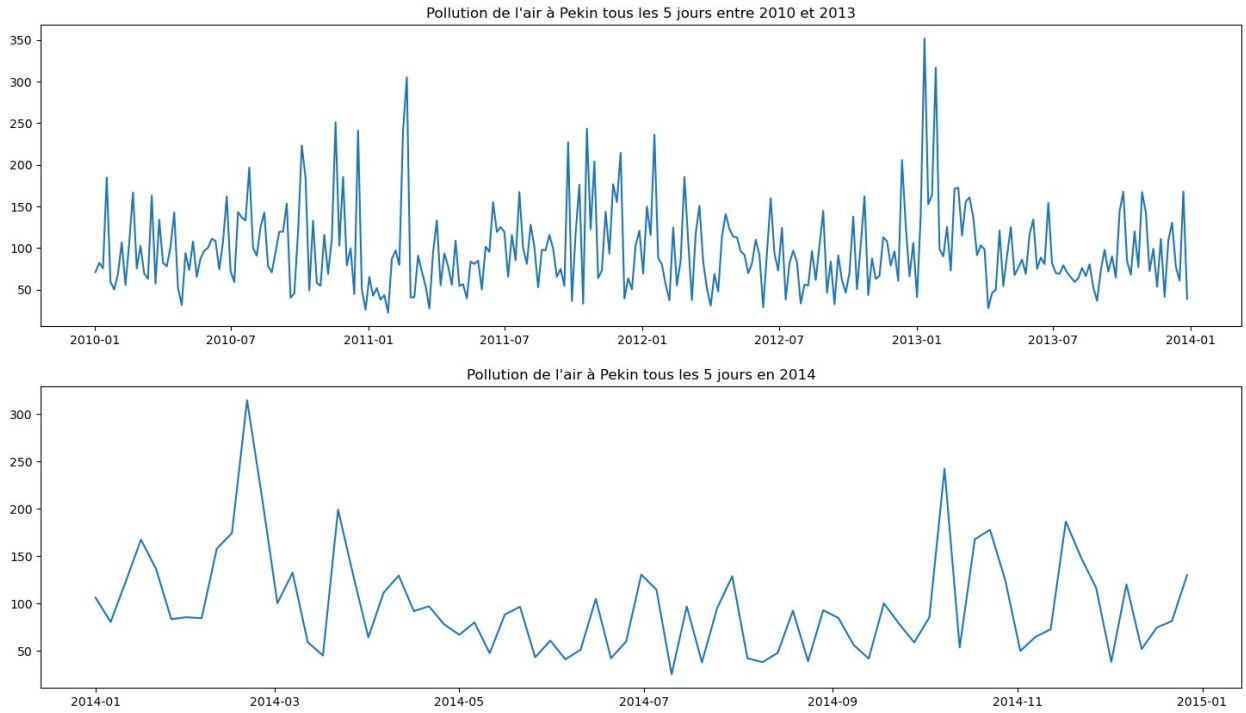
```
[<matplotlib.lines.Line2D at 0x1dba630e840>]
```



Nous allons entrainer le(s) futur(s) modèle(s) sur les données de 2010 à 2013 inclus et procéder à l'évalution du(des) dit(s) modèle(s) sur les données de 2014.

```
# Données d'apprentissage
X_train = y['2010':'2013'].index
y_train = np.array(y['2010':'2013'])

# Données de test
X_test = y['2014'].index
y_test = np.array(y['2014':])

plt.figure(figsize=(18, 10))
plt.subplot(211)
plt.plot(X_train, y_train)
plt.title("Pollution de l'air à Pekin tous les 5 jours entre 2010 et
2013")

plt.subplot(212)
plt.plot(X_test, y_test)
```

```
plt.title("Pollution de l'air à Pekin tous les 5 jours en 2014")
plt.show()
```

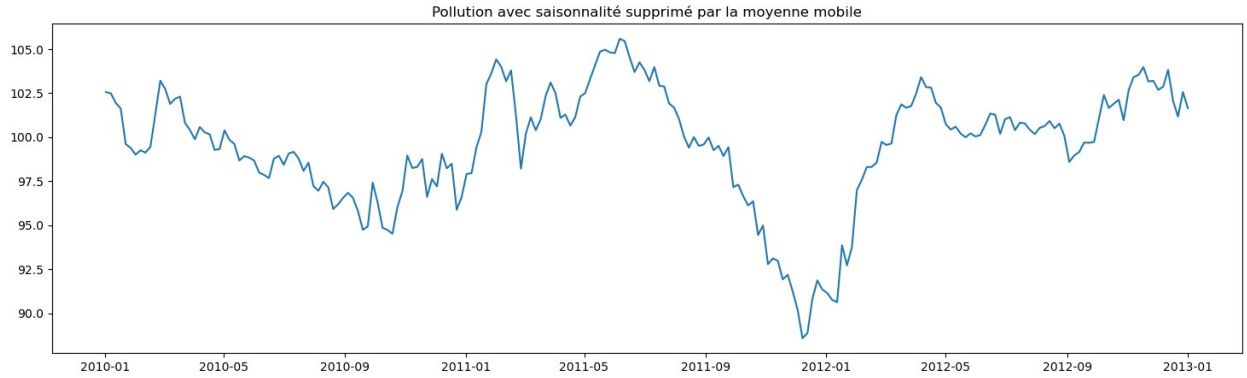


Gestion de la non-stationnarité

Estimation de la saisonnalité

Nous allons eliminer la saisonnalité à l'aide d'une moyenne mobile d'ordre 73 (5*73 = 365 pour 365 jours par an)

```
def moving_average(x, p):
    x.shape = len(x)
    return np.convolve(x, np.ones(p), "valid") / p

p = 73  # période de la moyenne mobile

y_train_sans_saison = moving_average(y_train, 73)
x_train_sans_saison = X_train[0:len(y_train_sans_saison)]

plt.figure(figsize=(18, 5))
plt.plot(x_train_sans_saison,y_train_sans_saison)
plt.title("Pollution avec saisonnalité supprimé par la moyenne
mobile")
plt.show()
```



Nous allons determiner les coefficients de saisonnalité centrés.

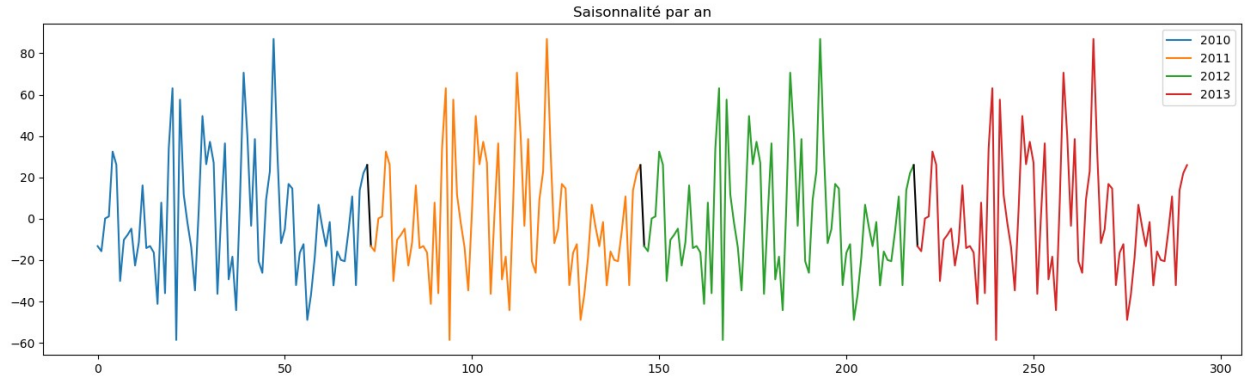
```
# On retire aux valeurs de bases les valeurs dénuées de saisonnalités

# Moyenne pour chaque mois des valeurs de saisonnalités
saisonnalite = [np.mean([serie_corrigee[j] for j in range(0,
len(y_train_sans_saison)) if j%p == i]) for i in range(p)]

# Vérification que la saisonnalité est nulle sur une année
moy = np.mean(saisonnalite)
for i in range(len(saisonnalite)):
    saisonnalite[i] = saisonnalite[i] - moy
sum(saisonnalite)
```

```
-4.618527782440651e-14
```

On a bien une saisonnalité annuelle nulle

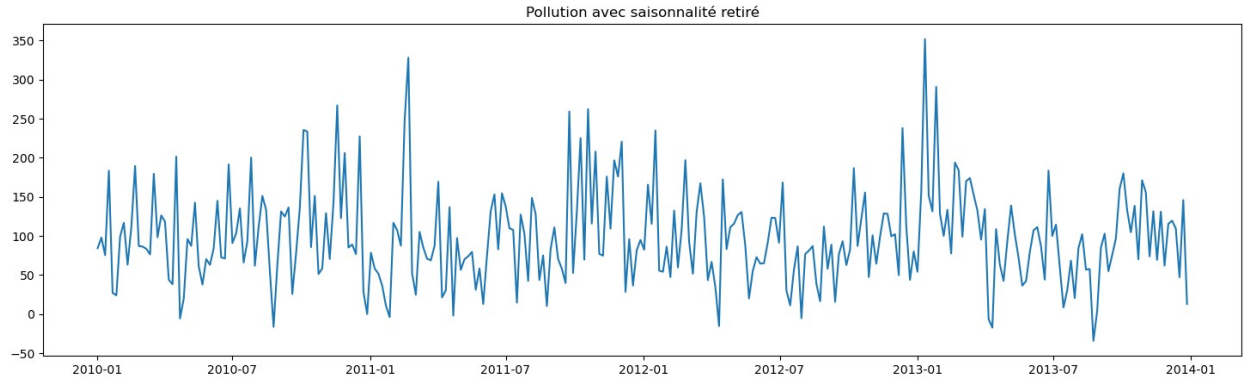
```
saison = []
rep_n = 0
while len(saison) != len(y_train):
    saison.append(saisonnalite[rep_n%p])
    rep_n += 1
df_saison = pd.DataFrame(saison, columns=["saison"])

plt.figure(figsize=(18,5))
plt.plot(df_saison["saison"][0:73],label="2010")
plt.plot(df_saison["saison"][72:74],c="black")

plt.plot(df_saison["saison"][73:146],label="2011")
plt.plot(df_saison["saison"][145:147],c="black")

plt.plot(df_saison["saison"][146:219],label="2012")
plt.plot(df_saison["saison"][218:220],c="black")

plt.plot(df_saison["saison"][219:292],label="2013")
plt.legend()
plt.title("Saisonnalité par an")
```

```
Text(0.5, 1.0, 'Saisonnalité par an')
```



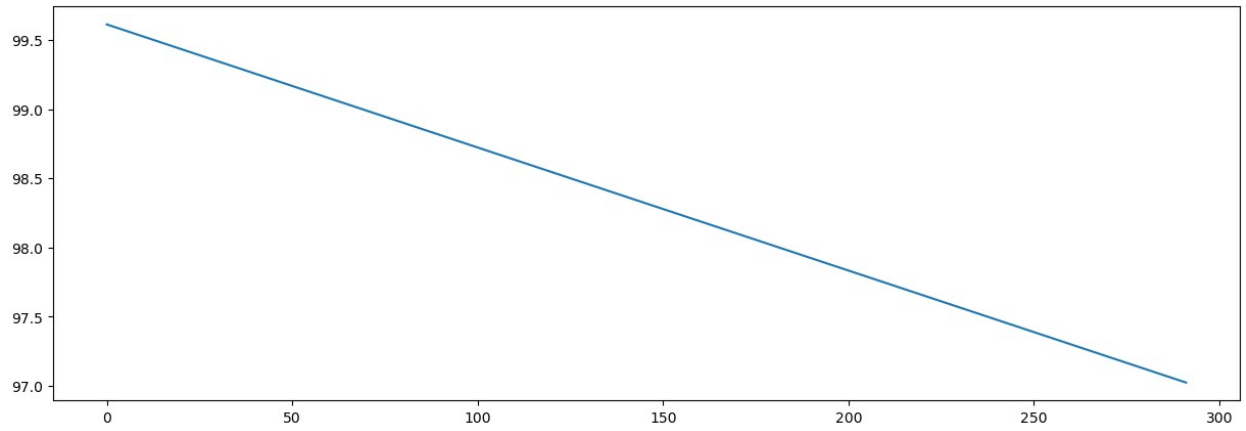
Estimation de la tendance

```
#On retire aux données de bases la saisonnalité
tendance = y_train - saison

df_tendance = pd.DataFrame(tendance, columns=["tendance"])
plt.figure(figsize=(18, 5))
plt.plot(X_train,df_tendance["tendance"])
plt.title("Pollution avec saisonnalité retiré")

Text(0.5, 1.0, 'Pollution avec saisonnalité retiré')
```



Modelisation de la tendance

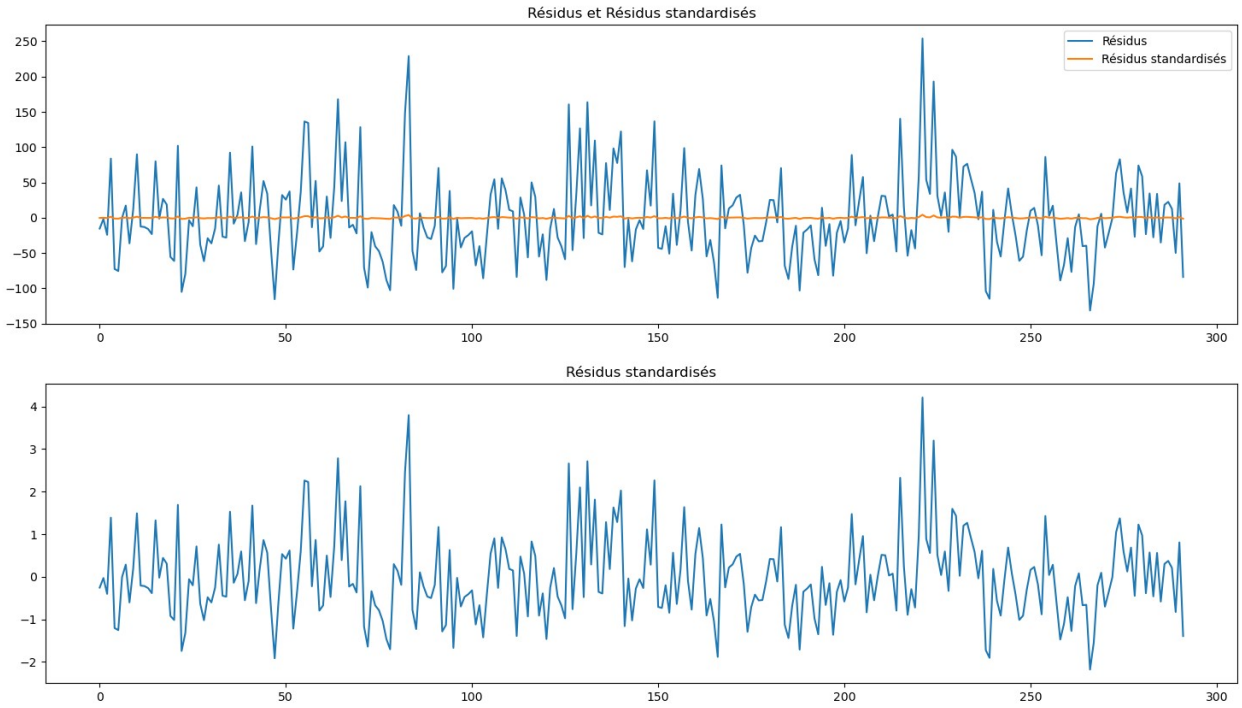
```
X_t= np.array(X['2010':'2013'])

modele_tendance = LinearRegression().fit(X_t,tendance)
fitted_values = modele_tendance.predict(X_t)
plt.figure(figsize=(15,5))
plt.plot(fitted_values)

[<matplotlib.lines.Line2D at 0x1dba781fb00>]
```



Calcul et standartisation des résidus

```
# On calcul les résidus en retirant des données la tendance et la saisonnalité
residus = y_train - saison - fitted_values
residus.shape = (len(residus), 1) #Même changement de dimension de précédemment

# Standartisation des résidus: on retire la moyenne et divise par la variance
residus_standards = StandardScaler().fit_transform(residus)

# Visualisation
plt.figure(figsize=(18,10))

plt.subplot(211)
plt.plot(residus,label="Résidus")
plt.plot(residus_standards,label="Résidus standardisés")
plt.legend()
plt.title("Résidus et Résidus standardisés")

#On affiche les résidus standardisés seuls afin que leur graphe soit plus visible
plt.subplot(212)
plt.plot(residus_standards)
plt.title("Résidus standardisés")
```

```
Text(0.5, 1.0, 'Résidus standardisés')
```



Test de stationnarité: test de Dickey-Fuller

L'hypothèse nulle de ce test est que la série n'est pas stationnaire.

On rentre les données et le type de régréssion: ici cct signifie que l'on a des constantes, une tendance linéaire et une composante quadratique

```
adfuller(y_train, regression="ctt")
```

```
(-8.087982825619694,
 2.0418627174046147e-10,
 2,
 289,
 {'1%': -4.411670279354562,
  '5%': -3.852979391736177,
  '10%': -3.565988982259978},
 2945.1802429717404)
```

Inteprétation des résulats :

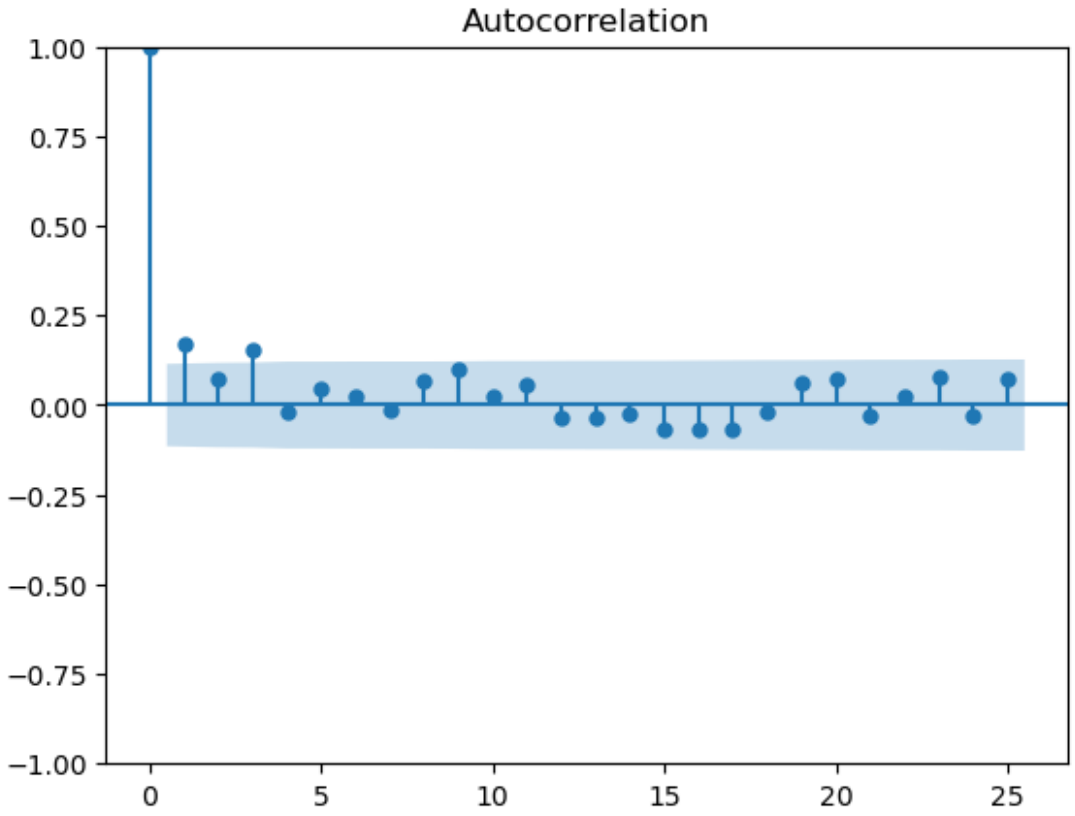
La variable nous intéréssant est celle de la deuxième ligne: c'est la P-valeur du test. Celle-ci étant pratiquement nulle, on rejete l'hypothèse de non stationnarité.

La série obtenue est donc bien stationnaire

Identification du modèle probabiliste

Analyse de l'auto correlation

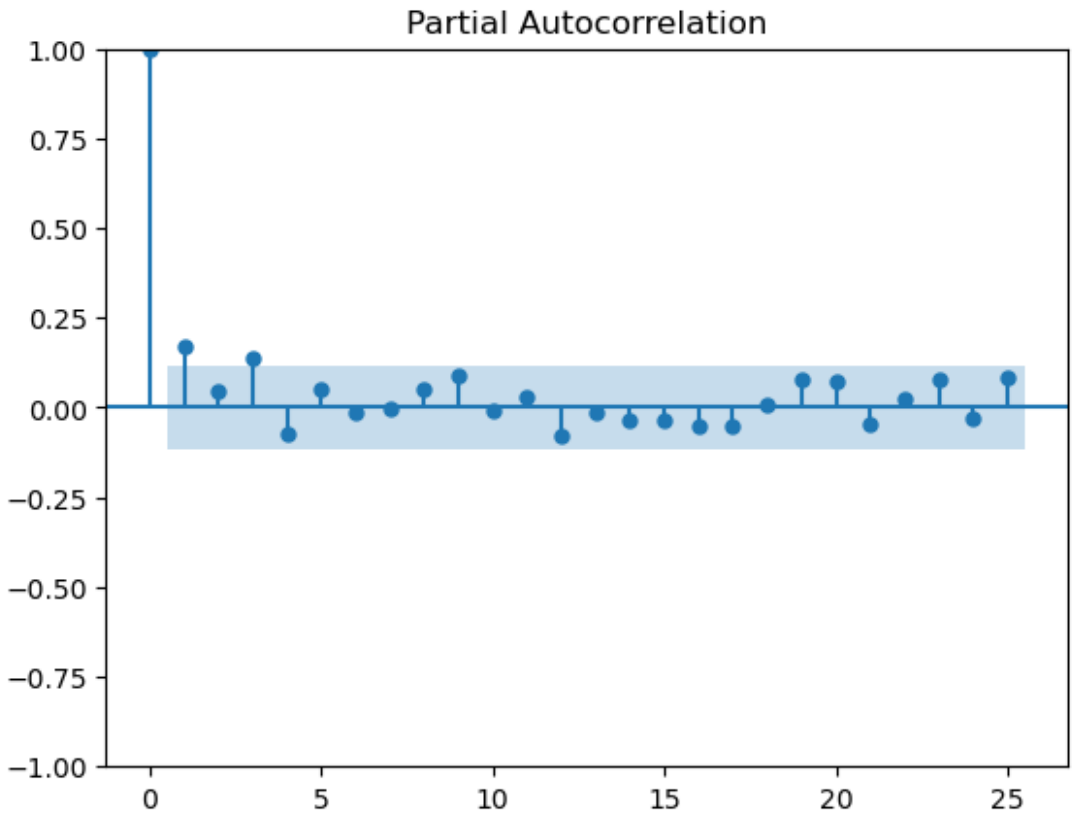
```
plot_acf(residus_standards)
plt.show()
```



On observe qu'un modèle MA(3) serait adapté à nos données

Analyse de l'auto correlation partielle

```
plot_pacf(residus_standards, method='ywm')
plt.show()
```



On observe qu'un modèle AR(3) serait adapté à nos données

Nous décidons donc de choisir pour modèle probabiliste le modèle ARMA(3,3).

Estimation des paramètres du modèle

```
modele_arma33 = ARIMA(endog=y_train, order=(3, 0, 3)).fit()
parametres = modele_arma33.params
parametres
```

```
array([ 9.82632686e+01, -2.70266709e-01,  2.93870302e-01, -
1.14151239e-01,
        4.02951502e-01, -2.31024178e-01,  2.15324171e-01,
2.44358930e+03])
```

```
modele_arma33.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                SARIMAX Results

==========================================================================
========
Dep. Variable:                           y   No. Observations:
292
Model:                  ARIMA(3, 0, 3)   Log Likelihood                -
```

```
1553.423
Date:                Tue, 17 Dec 2024   AIC
3122.847
Time:                        17:11:13   BIC
3152.261
Sample:                             0   HQIC
3134.629
                                - 292

Covariance Type:                   opg

==============================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
const         98.2633      4.810     20.429      0.000      88.836
107.691
ar.L1         -0.2703      0.729     -0.371      0.711      -1.700
1.159
ar.L2          0.2939      0.438      0.672      0.502      -0.564
1.152
ar.L3         -0.1142      0.516     -0.221      0.825      -1.126
0.898
ma.L1          0.4030      0.730      0.552      0.581      -1.027
1.833
ma.L2         -0.2310      0.493     -0.469      0.639      -1.197
0.735
ma.L3          0.2153      0.512      0.421      0.674      -0.788
1.218
sigma2      2443.5893    160.588     15.217      0.000    2128.843
2758.335
==============================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
244.00
Prob(Q):                              0.99   Prob(JB):
0.00
Heteroskedasticity (H):               0.83   Skew:
1.44
Prob(H) (two-sided):                  0.35   Kurtosis:
6.43
==============================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients
(complex-step).
"""
```

Prédiction des valeurs futures

```
modele_arma33.predict(start='2013-12-27',end='2014-12-27')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call
last)
File ~\anaconda3\Lib\site-packages\statsmodels\tsa\base\
tsa_model.py:249, in get_index_label_loc(key, index, row_labels)
    248 if not isinstance(key, (int, np.integer)):
--> 249     loc = row_labels.get_loc(key)
    250 else:

AttributeError: 'NoneType' object has no attribute 'get_loc'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call
last)
File ~\anaconda3\Lib\site-packages\statsmodels\tsa\base\
tsa_model.py:358, in get_prediction_index(start, end, nobs,
base_index, index, silent, index_none, index_generated, data)
    357 try:
--> 358     start, _, start_oos = get_index_label_loc(
    359         start, base_index, data.row_labels
    360     )
    361 except KeyError:

File ~\anaconda3\Lib\site-packages\statsmodels\tsa\base\
tsa_model.py:281, in get_index_label_loc(key, index, row_labels)
    280     except:
--> 281         raise e
    282 return loc, index, index_was_expanded

File ~\anaconda3\Lib\site-packages\statsmodels\tsa\base\
tsa_model.py:245, in get_index_label_loc(key, index, row_labels)
    244 try:
--> 245     loc, index, index_was_expanded = get_index_loc(key, index)
    246 except KeyError as e:

File ~\anaconda3\Lib\site-packages\statsmodels\tsa\base\
tsa_model.py:195, in get_index_loc(key, index)
    194 except (IndexError, ValueError) as e:
--> 195     raise KeyError(str(e))
    196 loc = key

KeyError: 'only integers, slices (`:`), ellipsis (`...`),
numpy.newaxis (`None`) and integer or boolean arrays are valid
indices'
```

```
During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call
last)
Cell In[907], line 1
----> 1 modele_arma33.predict(start='2013-12-27',end='2014-12-27')

File ~\anaconda3\Lib\site-packages\statsmodels\base\wrapper.py:113, in
make_wrapper.<locals>.wrapper(self, *args, **kwargs)
    111     obj = data.wrap_output(func(results, *args, **kwargs),
how[0], how[1:])
    112 elif how:
--> 113     obj = data.wrap_output(func(results, *args, **kwargs),
how)
    114 return obj

File ~\anaconda3\Lib\site-packages\statsmodels\tsa\statespace\
mlemodel.py:3486, in MLEResults.predict(self, start, end, dynamic,
information_set, signal_only, **kwargs)
   3421 r"""
   3422 In-sample prediction and out-of-sample forecasting
   3423
   (...)
   3483     including confidence intervals.
   3484 """
   3485 # Perform the prediction
-> 3486 prediction_results = self.get_prediction(
   3487     start, end, dynamic, information_set=information_set,
   3488     signal_only=signal_only, **kwargs)
   3489 return prediction_results.predicted_mean

File ~\anaconda3\Lib\site-packages\statsmodels\tsa\statespace\
mlemodel.py:3339, in MLEResults.get_prediction(self, start, end,
dynamic, information_set, signal_only, index, exog, extend_model,
extend_kwargs, **kwargs)
   3335     start = 0
   3337 # Handle start, end, dynamic
   3338 start, end, out_of_sample, prediction_index = (
-> 3339     self.model._get_prediction_index(start, end, index))
   3341 # Handle `dynamic`
   3342 if isinstance(dynamic, (str, dt.datetime, pd.Timestamp)):

File ~\anaconda3\Lib\site-packages\statsmodels\tsa\base\
tsa_model.py:836, in TimeSeriesModel._get_prediction_index(self,
start, end, index, silent)
    780 """
    781 Get the location of a specific key in an index or model row
labels
    782
   (...)
```

```
    833 since we have required them to be full indexes, there is no
ambiguity).
    834 """
    835 nobs = len(self.endog)
--> 836 return get_prediction_index(
    837     start,
    838     end,
    839     nobs,
    840     base_index=self._index,
    841     index=index,
    842     silent=silent,
    843     index_none=self._index_none,
    844     index_generated=self._index_generated,
    845     data=self.data,
    846 )

File ~\anaconda3\Lib\site-packages\statsmodels\tsa\base\
tsa_model.py:362, in get_prediction_index(start, end, nobs,
base_index, index, silent, index_none, index_generated, data)
    358     start, _, start_oos = get_index_label_loc(
    359         start, base_index, data.row_labels
    360     )
    361 except KeyError:
--> 362     raise KeyError(
    363         "The `start` argument could not be matched to a"
    364         " location related to the index of the data."
    365     )
    366 if end is None:
    367     end = max(start, len(base_index) - 1)

KeyError: 'The `start` argument could not be matched to a location
related to the index of the data.'
```

Evaluation de la précision de prédiction