In [1]:
```
# @hidden_cell
# The project token is an authorization token that is used to access
project resources like data sources, connections, and used by platfo
rm APIs.
from project_lib import Project
project = Project(project_id='ca8998af-60ec-4af7-a5ee-935daeff58ca',
project_access_token='p-1379f525698dc66ee90bcd6b8164145eb35ea2e7')
pc = project.project_context
```

In [2]:
```
# @hidden_cell
! pip install arch
```

```
Requirement already satisfied: arch in /opt/conda/envs/Python-3.7-
main/lib/python3.7/site-packages (4.15)
Requirement already satisfied: property-cached>=1.6.3 in /opt/cond
a/envs/Python-3.7-main/lib/python3.7/site-packages (from arch) (1.
6.4)
Requirement already satisfied: statsmodels>=0.9 in /opt/conda/envs
/Python-3.7-main/lib/python3.7/site-packages (from arch) (0.11.1)
Requirement already satisfied: scipy>=1.0.1 in /opt/conda/envs/Pyt
hon-3.7-main/lib/python3.7/site-packages (from arch) (1.5.0)
Requirement already satisfied: cython>=0.29.14 in /opt/conda/envs/
Python-3.7-main/lib/python3.7/site-packages (from arch) (0.29.21)
Requirement already satisfied: pandas>=0.23 in /opt/conda/envs/Pyt
hon-3.7-main/lib/python3.7/site-packages (from arch) (1.0.5)
Requirement already satisfied: numpy>=1.14 in /opt/conda/envs/Pyth
on-3.7-main/lib/python3.7/site-packages (from arch) (1.18.5)
Requirement already satisfied: patsy>=0.5 in /opt/conda/envs/Pytho
n-3.7-main/lib/python3.7/site-packages (from statsmodels>=0.9->arc
h) (0.5.1)
Requirement already satisfied: pytz>=2017.2 in /opt/conda/envs/Pyt
hon-3.7-main/lib/python3.7/site-packages (from pandas>=0.23->arch)
(2020.1)
Requirement already satisfied: python-dateutil>=2.6.1 in /opt/cond
a/envs/Python-3.7-main/lib/python3.7/site-packages (from pandas>=
0.23->arch) (2.8.1)
Requirement already satisfied: six in /opt/conda/envs/Python-3.7-m
ain/lib/python3.7/site-packages (from patsy>=0.5->statsmodels>=0.9
->arch) (1.15.0)
```

# Importation des Bilbiothèques

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
sns.set()
```

# Defintion des fonctions des graphes

In [116]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
def tsplot(y, ADF=True, lags=None,title=None, figsize=(15, 7), style
='bmh'):
    """
        Plot time series, its ACF and PACF, calculate Dickey-Fuller
test

        y - timeseries
        lags - how many lags to include in ACF, PACF calculation
    """
    if title == None:
        title = "Time Series Analysis Plots"
    else:
        title = str(title)
    if not isinstance(y, pd.Series):
        y = pd.Series(y)

    with plt.style.context(style):
        fig = plt.figure(figsize=figsize)
        layout = (2, 2)
        ts_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
        acf_ax = plt.subplot2grid(layout, (1, 0))
        pacf_ax = plt.subplot2grid(layout, (1, 1))

        y.plot(ax=ts_ax)
        if ADF == True:
            p_value = adfuller(y)[1]
            ts_ax.set_title(title + '\n Dickey-Fuller: p_value =
{0:.5f}'.format(p_value))
        else:
             ts_ax.set_title(title)
        plot_acf(y, lags=lags, ax=acf_ax)
        plot_pacf(y, lags=lags, ax=pacf_ax)
        plt.tight_layout()
```

In [128]:
```
from statsmodels.graphics.gofplots import qqplot
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
from seaborn import distplot
def tsplot_resid(y, lags=None,title=None, figsize=(15, 7), style='bm
h'):
    """
        Plot time series, its ACF and PACF, calculate Dickey-Fuller
test

        y - timeseries
        lags - how many lags to include in ACF, PACF calculation
    """
    if title == None:
        title = "Time Series Analysis Plots"
    else:
        title = str(title)
    if not isinstance(y, pd.Series):
        y = pd.Series(y)

    with plt.style.context(style):
        fig = plt.figure(figsize=figsize)
        layout = (3, 2)
        ts_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
        acf_ax = plt.subplot2grid(layout, (1, 0))
        pacf_ax = plt.subplot2grid(layout, (1, 1))
        qqplot_ax = plt.subplot2grid(layout, (2, 0))
        dist_ax = plt.subplot2grid(layout, (2, 1))

        y.plot(ax=ts_ax)
        ts_ax.set_title(title)
        plot_acf(y, lags=lags, ax=acf_ax)
        plot_pacf(y, lags=lags, ax=pacf_ax)
        qqplot(y,fit=True, line="45", ax=qqplot_ax)
        qqplot_ax.set_title("Normalitée")
        distplot(y, ax=dist_ax)
        dist_ax.set_title("Distribution")
        plt.tight_layout()
```

# Importation des données

In [75]:
```
consommation = project.get_file("Consommation d'électricité.xls")
df = pd.read_excel(consommation, sheet_name = "Algerie", index_col=
0)
df = df.transpose()
df.head()
```

Out[75]:

| Année | Consommation d'electricité par habitant |
|---|---|
| **1971** | 133.873489 |
| **1972** | 142.875928 |
| **1973** | 158.754155 |
| **1974** | 170.660458 |
| **1975** | 195.692277 |

## Traitement des données

In [35]:
```
df["Année"] = df.index
df.reset_index(inplace=True)
df["Année"] = pd.to_datetime(df["Année"], format="%Y")
df.drop(["index"], axis=1, inplace=True)
df.set_index("Année", inplace = True)
df.head()
```

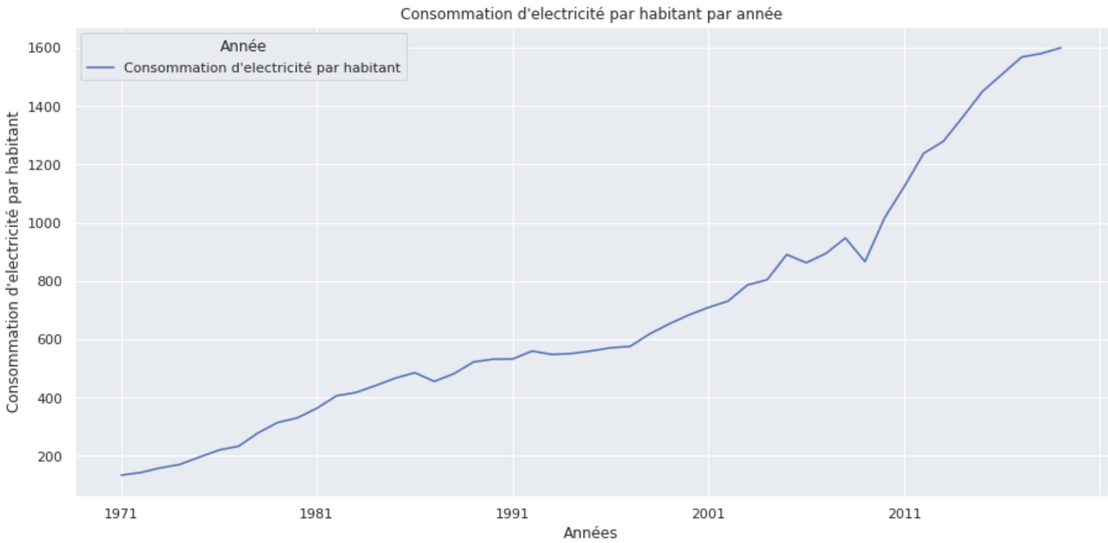
Out[35]:

| | Année | Consommation d'electricité par habitant |
|---|---|---|
| **Année** | | |
| **1971-01-01** | | 133.873489 |
| **1972-01-01** | | 142.875928 |
| **1973-01-01** | | 158.754155 |
| **1974-01-01** | | 170.660458 |
| **1975-01-01** | | 195.692277 |

## Graphique de la serie chronologique

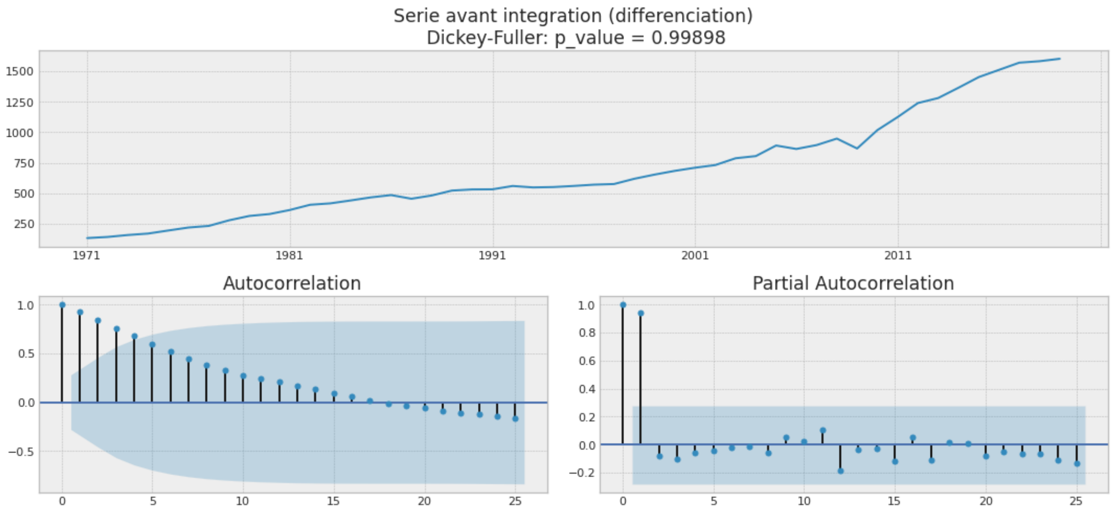
In [109]:
```
df.plot(figsize=(15,7))
plt.xlabel("Années")
plt.ylabel("Consommation d'electricité par habitant")
plt.title("Consommation d'electricité par habitant par année")
```

Out[109]:
```
Text(0.5, 1.0, "Consommation d'electricité par habitant par anné
e")
```




# Examination des données

In [119]:
```
tsplot(df["Consommation d'electricité par habitant"],lags=25, title
= "Serie avant integration (differenciation)")
```




- Après avoir appliqué le test de Dickey-Fuller Augmenté, la p_value = 0.99898 > 0.05, ce qui signifie que les données ne sont pas stationnaires.

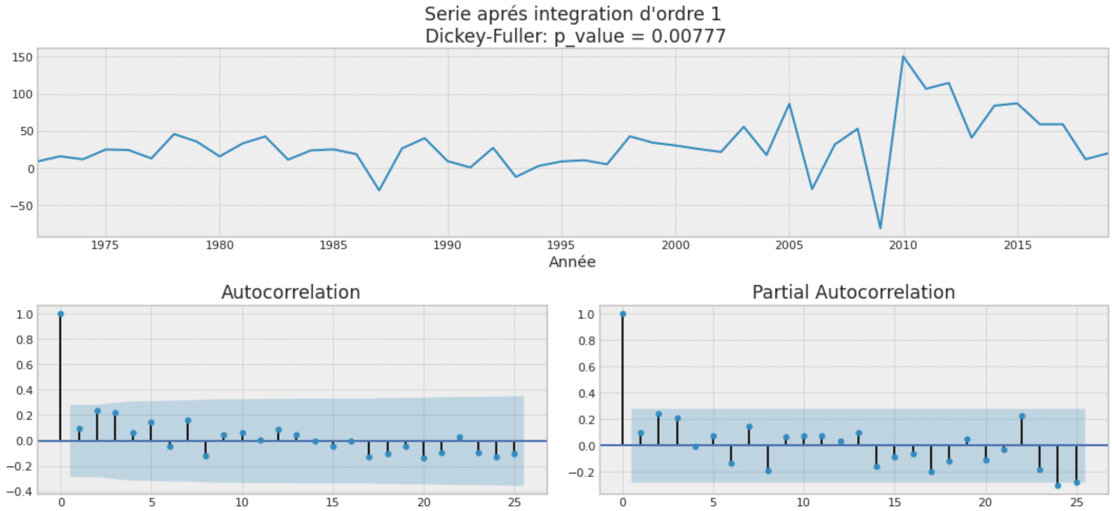
# Differenciation des données

In [13]:
```
dff = df.diff()
dff = dff.dropna()
dff.head()
```

Out[13]:

| Année | Consommation d'electricité par habitant |
|---|---|
| **Année** | |
| **1972-01-01** | 9.002438 |
| **1973-01-01** | 15.878228 |
| **1974-01-01** | 11.906302 |
| **1975-01-01** | 25.031819 |
| **1976-01-01** | 24.373496 |

In [120]:
```
tsplot(dff["Consommation d'electricité par habitant"],lags=25, title
="Serie aprés integration d'ordre 1")
```




- La nouvelle série après différenciation est une serie stationnaire selon la p_value du test ADF qui est egale a 0.007 < 0.05
- Aucun pique n'est significatif ni dans la FAC ni dans la FAP
- A première vue, un processus ARIMA(0,1,0) est le meilleure pour cette serie de données

In [16]:
```
from statsmodels.tsa.arima_model import ARIMA
def _get_best_model(TS):
    best_aic = np.inf
    best_order = None
    best_mdl = None
    pq_rng = range(5) # [0,1,2,3,4]
    d_rng = range(2) # [0,1]
    for i in pq_rng:
        for d in d_rng:
            for j in pq_rng:
                try:
                    tmp_mdl = ARIMA(TS, order=(i,d,j)).fit(
                    )
                    tmp_aic = tmp_mdl.aic
                    if tmp_aic < best_aic:
                        best_aic = tmp_aic
                        best_order = (i, d, j)
                        best_mdl = tmp_mdl
                except: continue
    print('aic: {:6.2f} | order: {}'.format(best_aic, best_order))
    return best_aic, best_order, best_mdl
res_tup = _get_best_model(df)
```

```
aic: 488.37 | order: (0, 1, 0)
```

# Choix du meilleure modèle

In [123]:
```
def get_best_model_2(TS):
    AIC = []
    order = []
    pq_rng = range(3) # [0,1,2,3,4]
    d_rng = range(1,2) # [0,1]
    for d in d_rng:
        for i in pq_rng:
            for j in pq_rng:
                try:
                    tmp_mdl = ARIMA(TS, order=(i,d,j)).fit()
                    AIC.append(tmp_mdl.aic)
                    order.append((i,d,j))
                except: continue
    tbl = {"Ordre":order, "AIC":AIC}
    rank = pd.DataFrame(tbl)
    rank.sort_values(by=["AIC"], ascending=True, inplace=True)
    rank.set_index("Ordre", inplace=True)
    return rank
```

In [124]:
```
get_best_model_2(df)
```

Out[124]:

| | AIC |
|---|---|
| **Ordre** | |
| **(0, 1, 0)** | 488.371799 |
| **(2, 1, 0)** | 489.413460 |
| **(1, 1, 2)** | 489.867997 |
| **(0, 1, 2)** | 489.879725 |
| **(1, 1, 0)** | 489.972328 |
| **(2, 1, 2)** | 490.062727 |
| **(0, 1, 1)** | 490.093880 |
| **(2, 1, 1)** | 490.167045 |
| **(1, 1, 1)** | 492.251340 |

- Selon le critère Akaike (AIC), le meilleur modele est un ARIMA(0,1,0) avec un AIC = 488.37
- Cela confirme notre première hypothèse

# Estimation du modele ARIMA(0,1,0)

In [125]:
```
modele = ARIMA(df, order=(0,1,0)).fit()
modele.summary()
```

Out[125]:

ARIMA Model Results

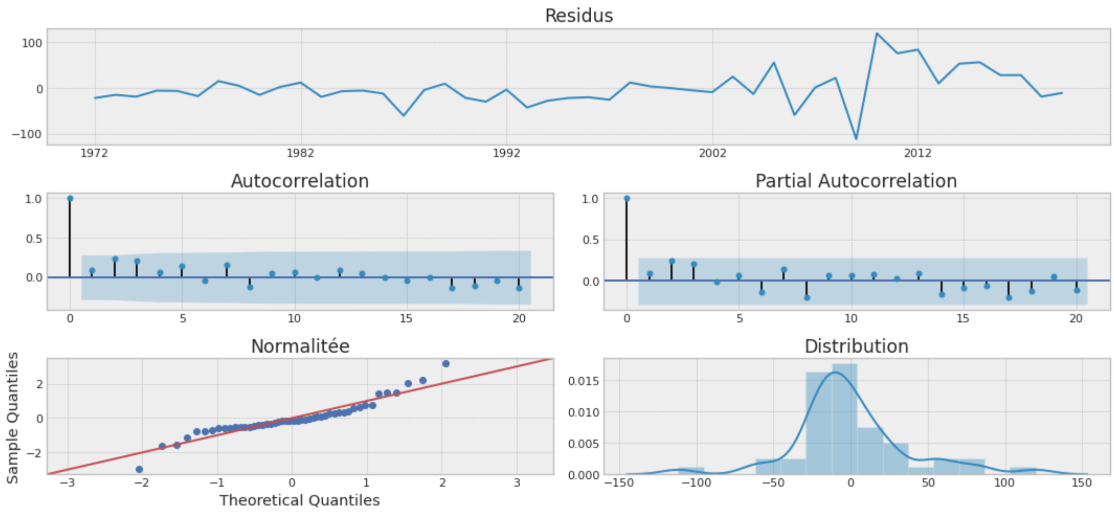
| | | | |
|---|---|---|---|
| **Dep. Variable:** | D.Consommation d'electricité par habitant | **No. Observations:** | 48 |
| **Model:** | ARIMA(0, 1, 0) | **Log Likelihood** | -242.186 |
| **Method:** | css | **S.D. of innovations** | 37.585 |
| **Date:** | Mon, 28 Dec 2020 | **AIC** | 488.372 |
| **Time:** | 22:08:33 | **BIC** | 492.114 |
| **Sample:** | 01-01-1972 | **HQIC** | 489.786 |
| | - 01-01-2019 | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | 30.5443 | 5.425 | 5.630 | 0.000 | 19.912 | 41.177 |

- La constante du modele est signifiactif

# Examination des Residus

In [129]:
```
tsplot_resid(modele.resid,lags=20, title="Residus")
```




- A première vue, les residus ont les propriétés d'un Bruit Blanc
- Il n'existe aucun pique significatif dans la FAC et la FAP

## Test ARCH d'Engle pour l'homoscedasticité des Erreurs

#### Ho : Les résidus au carré sont une séquence de bruit blanc - les résidus sont homoscédastiques.

#### H1 : Les résidus au carré n'ont pas pu être ajustés avec un modèle de régression linéaire et présentent une hétéroscédasticité.

In [130]:
```
from statsmodels.stats.diagnostic import het_arch
print()
print("La valeur critique :",het_arch(modele.resid)[2],",      la p_value :",het_arch(modele.resid)[3])
```

```
La valeur critique : 1.6320909879309025 ,      la p_value : 0.1505
1916477323907
```

- Étant donné que la p_value est de 0.1505 > 0.05, nous ne parvenons pas à rejeter l'hypothèse nulle, d'ou les résidus sont homoscédastiques
- Donc il n'existe pas d'effet ARCH sur les residus.

## Test de Ljung-Box pour Autocorrelation des Erreurs

#### H0: Les résidus sont distribués indépendamment.

#### HA: les résidus ne sont pas distribués indépendamment; ils présentent une corrélation en série.

In [134]:
```
from statsmodels.stats.diagnostic import acorr_ljungbox
LB = acorr_ljungbox(modele.resid,lags=[10])
print("statistique =", LB[0][0], "             p_value =",LB[1][0]
)
```

```
statistique = 10.181333984986784              p_value = 0.424730
9703532606
```

- Étant donné que la p_value est de 0,4247 > 0,05, nous ne parvenons pas à rejeter l'hypothèse nulle, les résidus sont donc distribués indépendamment,
- Ce qui signifie que les résidus ne sont pas autocorrélés

### Test de Jarque-Bera de Normalitée des Erreurs

**H0 : les données suivent une loi normale.**

**H1 : les données ne suivent pas une loi normale.**

In [137]:
```
from scipy.stats import jarque_bera
JB = jarque_bera(modele_garch.resid)
print("statistic=", JB[0],"            p_value=",JB[1] )
print()
```

```
statistic= 12.584370443790368            p_value= 0.00185071131000
53596
```

- Puisque la p_value = 0.001 < 0,05, nous rejetons l'hypothèse nulle. Ainsi, nous avons des preuves suffisantes pour dire que ces données ont une asymétrie et un kurtosis qui sont significativement différents d'une distribution normale.
- Donc les erreurs ne suivent pas une loi Normale

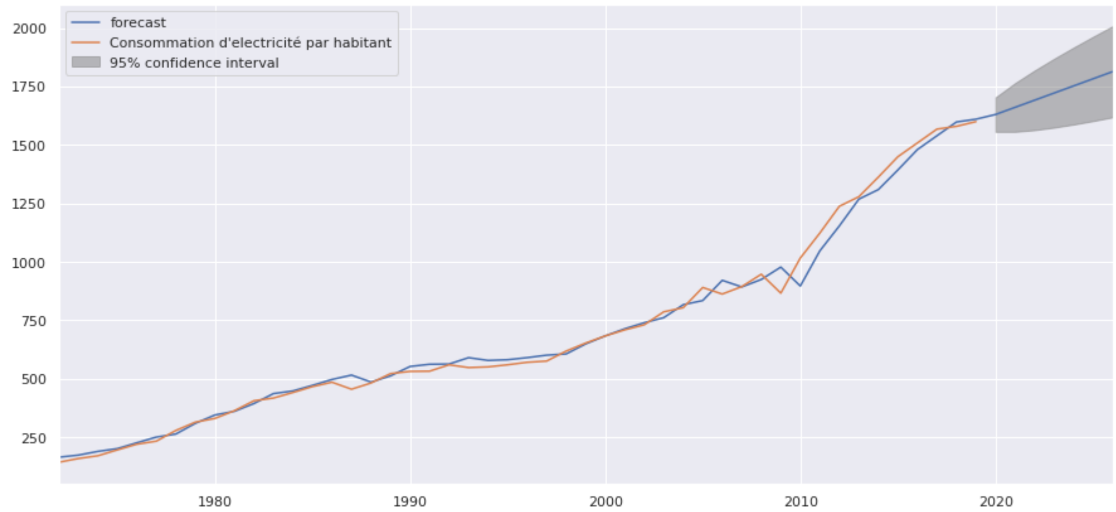
## Prevision pur les années 2020 - 2024

In [138]:
```
pred = modele.forecast(5)
années = [2020, 2021, 2022, 2023, 2024]
df_pred = pd.DataFrame({"Années":années, "Prevision":pred[0]})
df_pred.set_index("Années", inplace=True)
df_pred
```

Out[138]:

| | Prevision |
|---|---|
| **Années** | |
| **2020** | 1630.544302 |
| **2021** | 1661.088605 |
| **2022** | 1691.632907 |
| **2023** | 1722.177209 |
| **2024** | 1752.721512 |

In [139]:
```
fig, ax = plt.subplots(figsize=(15, 7))
fig = modele.plot_predict(1,55, ax=ax)
```




# ARCH / GARCH

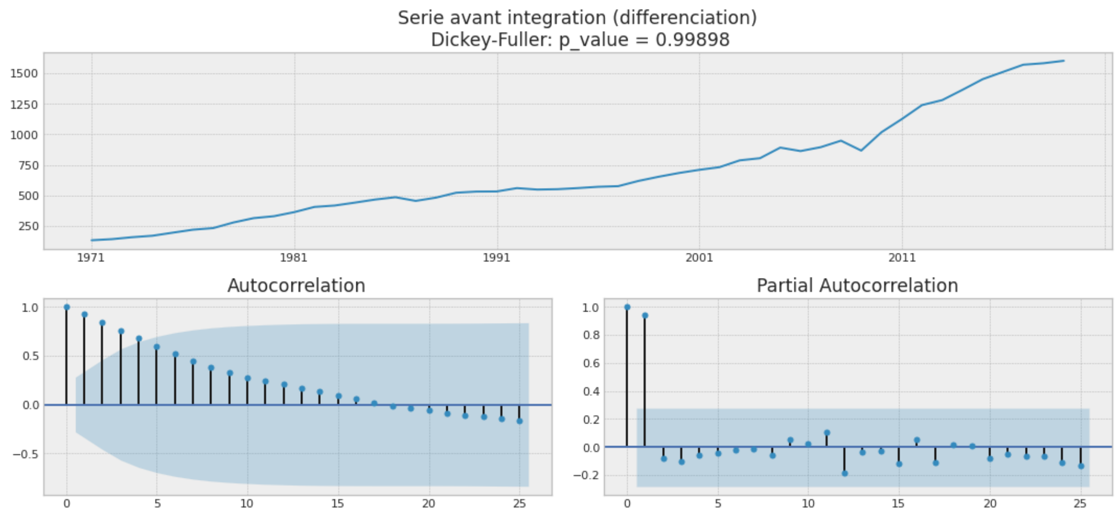
# Fractionnement des données en données d'entrainement et de test

In [7]:
```
size = int(len(df)*0.9)
df_train = df.iloc[:size]
df_test = df.iloc[size:]
df_train.tail()
```

Out[7]:

| Année | Consommation d'electricité par habitant |
|---|---|
| **Année** | |
| **2010-01-01** | 1016.636669 |
| **2011-01-01** | 1123.332731 |
| **2012-01-01** | 1237.966507 |
| **2013-01-01** | 1278.915343 |
| **2014-01-01** | 1362.871919 |

In [113]:
```
tsplot(df_train["Consommation d'electricité par habitant"],lags=25,
title = "Serie avant integration (differenciation)")
```

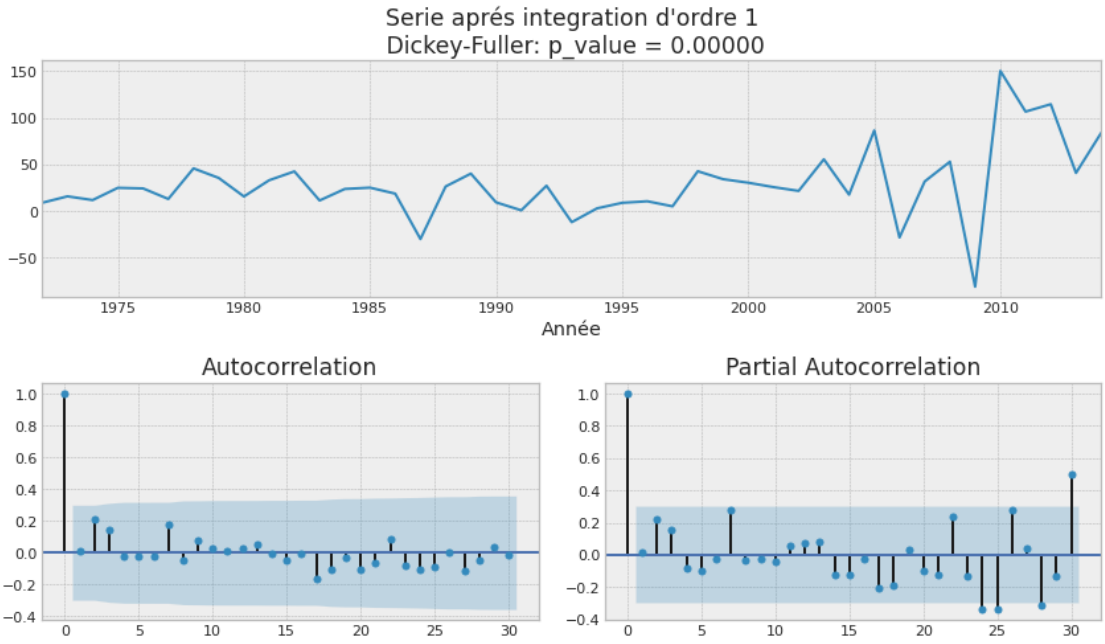



In [12]:
```
dif_train = df_train.diff()
dif_train = dif_train.dropna()
dif_train.head()
```

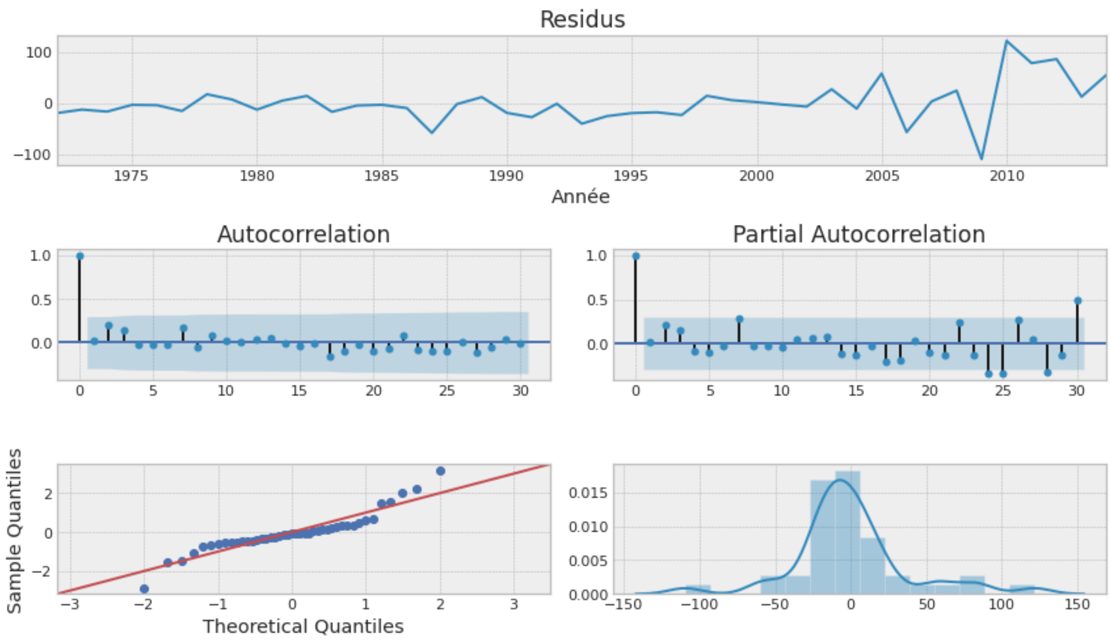
Out[12]:

| Année | Consommation d'electricité par habitant |
|---|---|
| **Année** | |
| **1972-01-01** | 9.002438 |
| **1973-01-01** | 15.878228 |
| **1974-01-01** | 11.906302 |
| **1975-01-01** | 25.031819 |
| **1976-01-01** | 24.373496 |

In [14]:
```
tsplot(dif_train["Consommation d'electricité par habitant"],lags=30,
title="Serie aprés integration d'ordre 1")
```



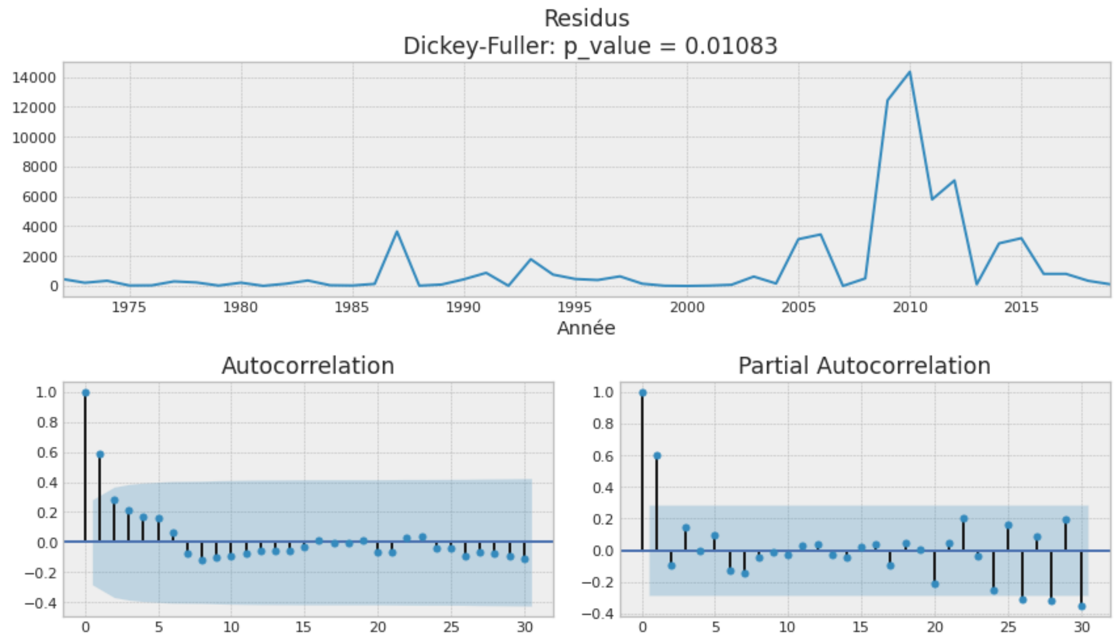

In [45]:
```
model = ARIMA(df_train, order=(0,1,0)).fit()
tsplot_resid(model.resid,lags=30, title="Residus")
```




In [ ]:

Elevation des residus au carré pour detecter la necessité d'un modéle ARCH pour les residus

In [50]:
```
residus_carré = modele.resid**2
tsplot(residus_carré,lags=30, title="Residus")
```




- Le carré des residus n'est pas un Bruit Blanc a cause de la présence de piques significatifs dans la FAC et la FAP
- Selon les deux graphs des autocorrelation, un modéle GARCH(1,1) est le meilleure modele pour expliquer la volatiliter dans les residus

# Estiomation d'un modele GARCH

In [51]:
```
from arch import arch_model
garch = arch_model(modele.resid,p=1, q=1)
modele_garch = garch.fit(update_freq=5)
modele_garch.summary()
```

```
Iteration:      5,   Func. Count:     28,   Neg. LLF: 231.17058190
25867
Iteration:     10,   Func. Count:     53,   Neg. LLF: 231.08722051
3383
Iteration:     15,   Func. Count:     78,   Neg. LLF: 231.07867542
736687
Iteration:     20,   Func. Count:    103,   Neg. LLF: 230.99550939
67152
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.9949613661152
            Iterations: 24
            Function evaluations: 122
            Gradient evaluations: 24
```

Out[51]:

Constant Mean - GARCH Model Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | None | **R-squared:** | -0.060 |
| **Mean Model:** | Constant Mean | **Adj. R-squared:** | -0.060 |
| **Vol Model:** | GARCH | **Log-Likelihood:** | -230.995 |
| **Distribution:** | Normal | **AIC:** | 469.990 |
| **Method:** | Maximum Likelihood | **BIC:** | 477.475 |
| | | **No. Observations:** | 48 |
| **Date:** | Mon, Dec 28 2020 | **Df Residuals:** | 44 |
| **Time:** | 20:11:23 | **Df Model:** | 4 |

Mean Model

| | coef | std err | t | P>\|t\| | 95.0% Conf. Int. |
|---|---|---|---|---|---|
| **mu** | -9.2138 | 4.516 | -2.040 | 4.132e-02 | [-18.065, -0.363] |

Volatility Model

| | coef | std err | t | P>\|t\| | 95.0% Conf. Int. |
|---|---|---|---|---|---|
| **omega** | 189.8474 | 117.529 | 1.615 | 0.106 | [-40.505,4.202e+02] |
| **alpha[1]** | 0.3799 | 0.156 | 2.433 | 1.497e-02 | [7.385e-02, 0.686] |
| **beta[1]** | 0.4789 | 0.185 | 2.593 | 9.503e-03 | [ 0.117, 0.841] |

Covariance estimator: robust

In [53]:
```
df_train2 = df_train**2
df_train2.head()
```

Out[53]:

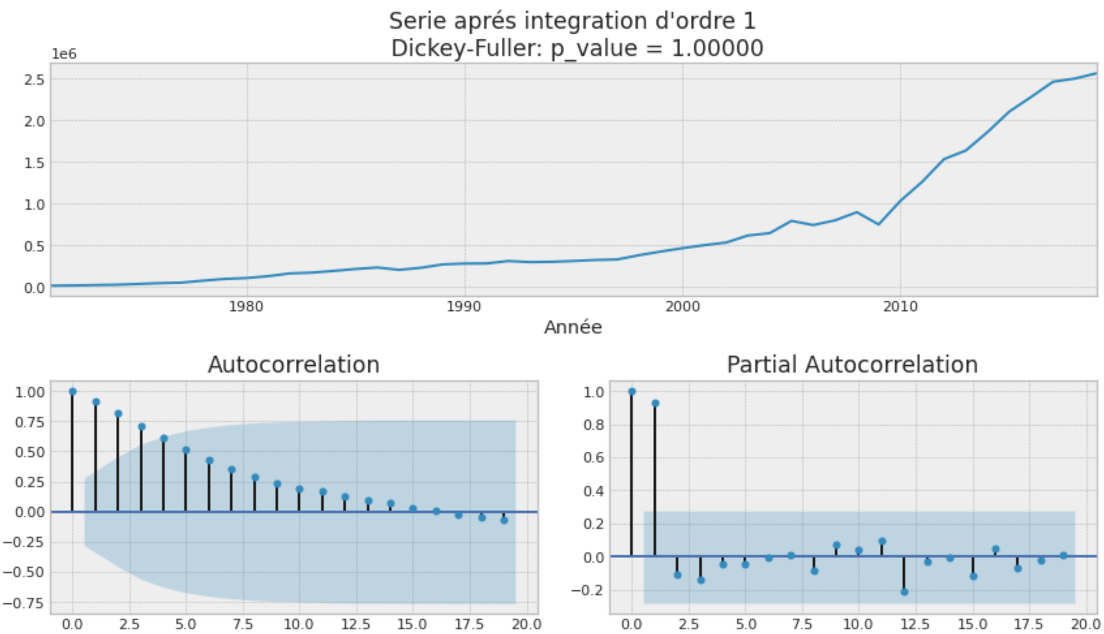
| | Consommation d'electricité par habitant |
|---|---|
| **Année** | |
| **1971-01-01** | 17922.111115 |
| **1972-01-01** | 20413.530678 |
| **1973-01-01** | 25202.881870 |
| **1974-01-01** | 29124.991867 |
| **1975-01-01** | 38295.467311 |

In [52]:
```
df2 = df**2
df2.head()
```

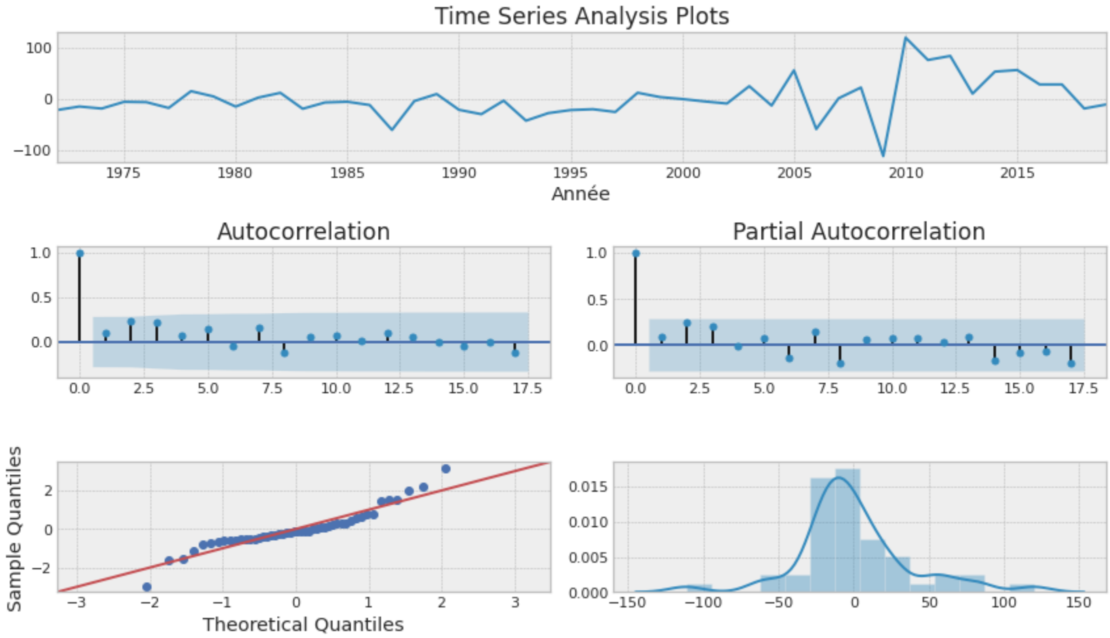
Out[52]:

| | Consommation d'electricité par habitant |
|---|---|
| **Année** | |
| **1971-01-01** | 17922.111115 |
| **1972-01-01** | 20413.530678 |
| **1973-01-01** | 25202.881870 |
| **1974-01-01** | 29124.991867 |
| **1975-01-01** | 38295.467311 |

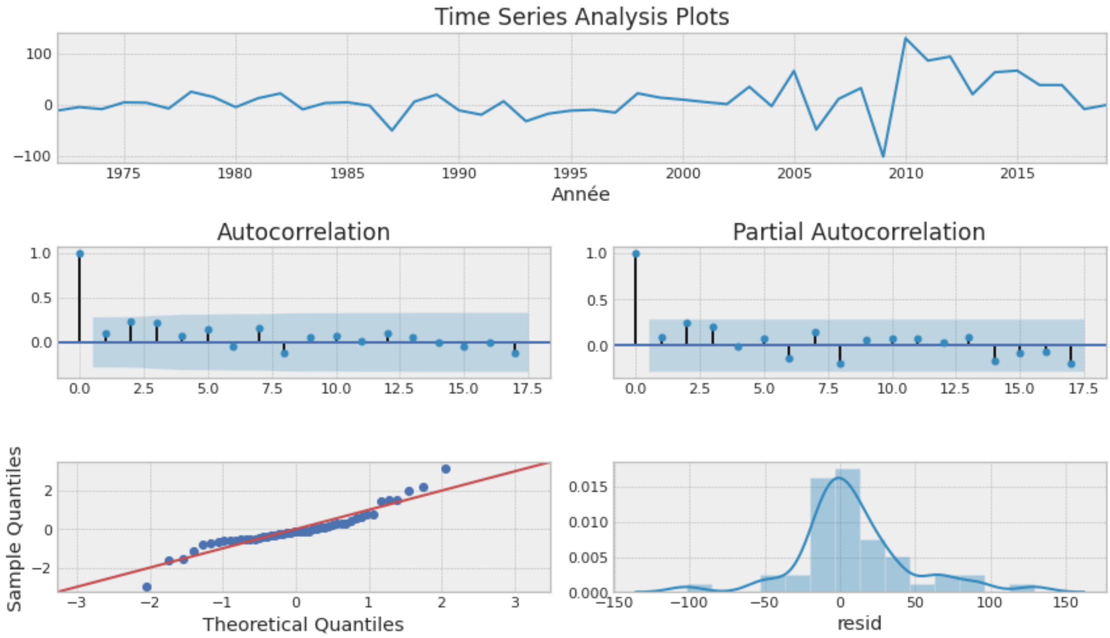
In [54]:
```
tsplot(df2["Consommation d'electricité par habitant"],lags=19, title
="Serie aprés integration d'ordre 1")
```



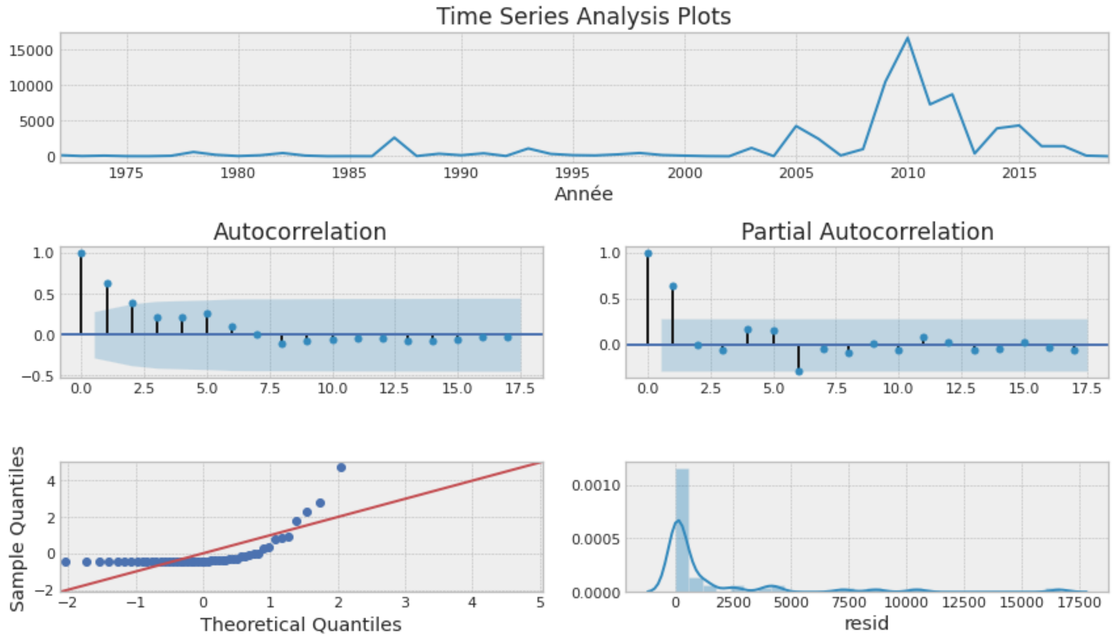
In [55]:
```
tsplot_resid(modele.resid)
```




In [56]:
```
tsplot_resid(modele_garch.resid)
```



In [57]:
```
garch2 = modele_garch.resid**2
tsplot_resid(garch2)
```




In [135]:
```
AIC = []
order = []
for p in range(1,11):
    for q in range(2):
        garch = arch_model(model.resid,p=p, q=q)
        modele_garch = garch.fit(update_freq=50)
        AIC.append(tmp_mdl.aic)
        order.append((p,q))

tbl = {"Ordre":order, "AIC":AIC}
df = pd.DataFrame(tbl)
df
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 229.40434518955567
            Iterations: 16
            Function evaluations: 67
            Gradient evaluations: 16

------------------------------------------------------------------
---------
NameError                                 Traceback (most recent c
all last)
<ipython-input-135-c903177ca88b> in <module>
      5         garch = arch_model(model.resid,p=p, q=q)
      6         modele_garch = garch.fit(update_freq=50)
----> 7         AIC.append(tmp_mdl.aic)
      8         order.append((p,q))
      9 

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

In [136]:
```
df.sort_values(by="AIC", ascending=True, inplace=True)
df.head(1)
```

Out[136]:

| | AIC |
|---|---|
| **Ordre** | |
| **(0, 1, 1)** | 479.129958 |

In [149]:
```
garch = arch_model(model.resid, vol='GARCH', p=1, q=1)
modele_garch = garch.fit(update_freq=50)
modele_garch.summary()
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 227.68691210273192
            Iterations: 23
            Function evaluations: 117
            Gradient evaluations: 23
```

Out[149]:

Constant Mean - GARCH Model Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | None | **R-squared:** | -0.013 |
| **Mean Model:** | Constant Mean | **Adj. R-squared:** | -0.013 |
| **Vol Model:** | GARCH | **Log-Likelihood:** | -227.687 |
| **Distribution:** | Normal | **AIC:** | 463.374 |
| **Method:** | Maximum Likelihood | **BIC:** | 470.774 |
| | | **No. Observations:** | 47 |
| **Date:** | Mon, Dec 28 2020 | **Df Residuals:** | 43 |
| **Time:** | 18:32:33 | **Df Model:** | 4 |

Mean Model

| | coef | std err | t | P>\|t\| | 95.0% Conf. Int. |
|---|---|---|---|---|---|
| **mu** | -6.0977 | 4.910 | -1.242 | 0.214 | [-15.721, 3.526] |

Volatility Model

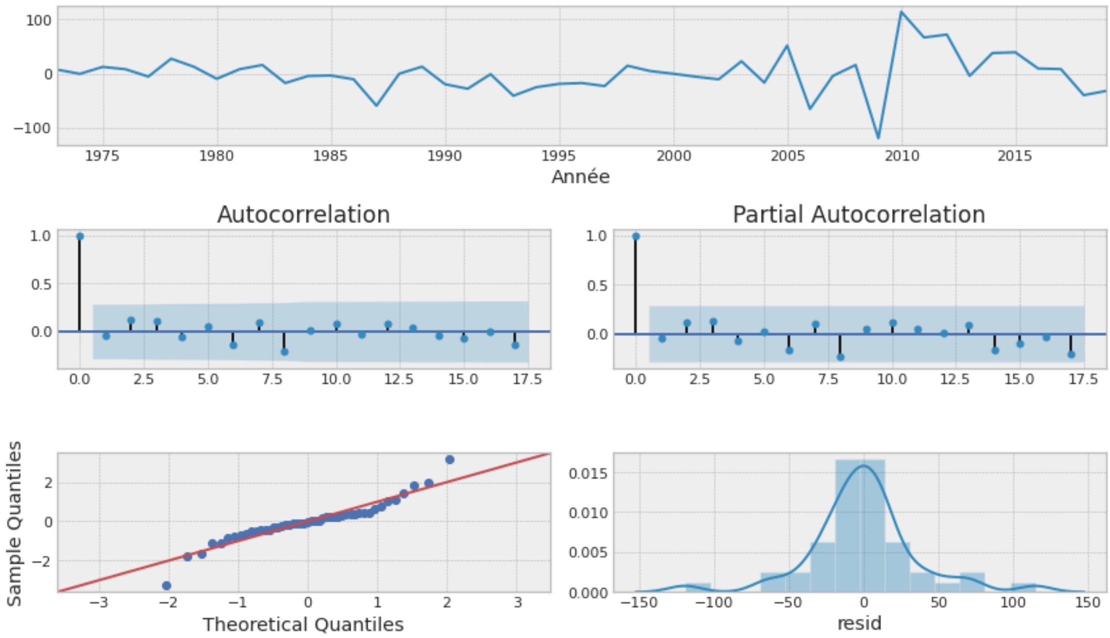
| | coef | std err | t | P>\|t\| | 95.0% Conf. Int. |
|---|---|---|---|---|---|
| **omega** | 176.5956 | 114.559 | 1.542 | 0.123 | [-47.935,4.011e+02] |
| **alpha[1]** | 0.2931 | 0.136 | 2.153 | 3.132e-02 | [2.628e-02, 0.560] |
| **beta[1]** | 0.5872 | 0.129 | 4.557 | 5.179e-06 | [ 0.335, 0.840] |

Covariance estimator: robust

In [138]:
```
print("La valeur critique :",het_arch(modele_garch.resid)[0],het_arch(modele_garch.resid)[2] ,",      la p_value :",het_arch(modele_garch.resid)[1],het_arch(modele_garch.resid)[3])
```

```
La valeur critique : 14.583755980836452 1.6915307273492852 ,
la p_value : 0.1479902669312523 0.13637259886661823
```

In [143]:
```
tsplot_resid(modele_garch.resid)
```



In [59]:
```
AIC = []
order = []
p_val = []
for p in range(1,11):
    for q in range(2):
        garch = arch_model(modele.resid, vol='GARCH', p=p, q=q)
        modele_garch = garch.fit(update_freq=50)
        AIC.append(modele_garch.aic)
        order.append((p,q))
        p_val.append(het_arch(modele_garch.resid)[3])
tbl = {"Ordre":order,"P_val":p_val, "AIC":AIC}
df = pd.DataFrame(tbl)
df.sort_values(by="P_val", ascending=True, inplace=True)
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 232.9765195734206
            Iterations: 24
            Function evaluations: 97
            Gradient evaluations: 24
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.9949613661152
            Iterations: 24
            Function evaluations: 122
            Gradient evaluations: 24
Optimization terminated successfully    (Exit mode 0)
            Current function value: 231.46459894576708
            Iterations: 26
            Function evaluations: 132
            Gradient evaluations: 26
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.99496121007604
            Iterations: 27
            Function evaluations: 166
            Gradient evaluations: 27
Optimization terminated successfully    (Exit mode 0)
            Current function value: 231.12671119026473
            Iterations: 29
            Function evaluations: 178
            Gradient evaluations: 29
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.9464724923102
            Iterations: 28
            Function evaluations: 203
            Gradient evaluations: 28
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.00531736435568
            Iterations: 34
            Function evaluations: 245
            Gradient evaluations: 34
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.0053176639155
            Iterations: 35
            Function evaluations: 288
            Gradient evaluations: 35
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.0053174436408
            Iterations: 31
            Function evaluations: 251
            Gradient evaluations: 31
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.00531733630223
            Iterations: 31
            Function evaluations: 283
            Gradient evaluations: 31
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.00531732748857
            Iterations: 36
            Function evaluations: 330
            Gradient evaluations: 36
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.00531735829298
            Iterations: 29
            Function evaluations: 293
            Gradient evaluations: 29
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.0053174873644
            Iterations: 34
            Function evaluations: 345
            Gradient evaluations: 34
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.00531735025632
            Iterations: 30
            Function evaluations: 335
            Gradient evaluations: 30
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.00531768885725
            Iterations: 35
            Function evaluations: 390
            Gradient evaluations: 35
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.00531760298708
            Iterations: 30
            Function evaluations: 365
            Gradient evaluations: 30
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.00531740228558
            Iterations: 33
            Function evaluations: 401
            Gradient evaluations: 33
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.00531780911427
            Iterations: 30
            Function evaluations: 395
            Gradient evaluations: 30
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.00531735473038
            Iterations: 31
            Function evaluations: 409
            Gradient evaluations: 31
Optimization terminated successfully    (Exit mode 0)
            Current function value: 230.00531741232135
            Iterations: 28
            Function evaluations: 397
            Gradient evaluations: 28
```

In [146]:
```
df.sort_values(by="AIC", ascending=True, inplace=True)
df
```

Out[146]:

| | Ordre | P_val | AIC |
|---|---|---|---|
| **4** | (3, 0) | 0.192056 | 460.602458 |
| **6** | (4, 0) | 0.192272 | 462.601188 |
| **5** | (3, 1) | 0.192069 | 462.602458 |
| **1** | (1, 1) | 0.136373 | 463.373824 |
| **7** | (4, 1) | 0.192263 | 464.601188 |
| **8** | (5, 0) | 0.192271 | 464.601188 |
| **0** | (1, 0) | 0.114131 | 464.808690 |
| **3** | (2, 1) | 0.136374 | 465.373825 |
| **10** | (6, 0) | 0.192248 | 466.601188 |
| **9** | (5, 1) | 0.191536 | 466.602406 |
| **2** | (2, 0) | 0.100641 | 466.703509 |
| **12** | (7, 0) | 0.192249 | 468.601188 |
| **11** | (6, 1) | 0.191530 | 468.602407 |
| **14** | (8, 0) | 0.192258 | 470.601188 |
| **13** | (7, 1) | 0.191534 | 470.602407 |
| **16** | (9, 0) | 0.197059 | 472.482455 |
| **15** | (8, 1) | 0.191533 | 472.602408 |
| **17** | (9, 1) | 0.197059 | 474.482455 |
| **18** | (10, 0) | 0.197064 | 474.482455 |
| **19** | (10, 1) | 0.197039 | 476.482455 |

In [ ]: