In [59]:
```
import yfinance as yf
from statsmodels.tsa.stattools import adfuller
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import ruptures as rpt


plt.style.use('seaborn-darkgrid')
from IPython.display import set_matplotlib_formats
%matplotlib inline
set_matplotlib_formats('svg')
```

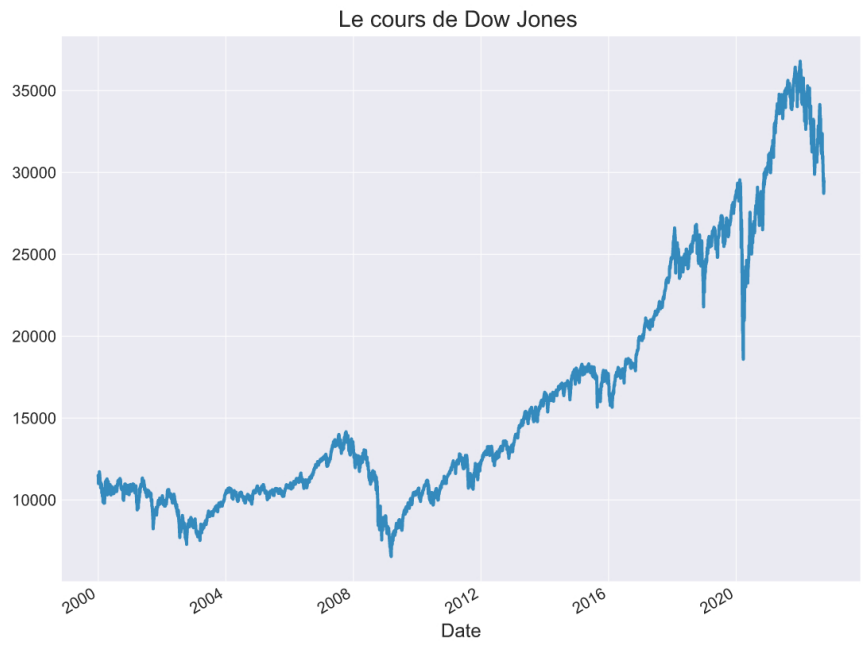
```
C:\Users\HPELIT~1\AppData\Local\Temp/ipykernel_18696/744802615.py:12: DeprecationWarning: `set_matplotlib_formats` is deprecate
d since IPython 7.23, directly use `matplotlib_inline.backend_inline.set_matplotlib_formats()`
  set_matplotlib_formats('svg')
```

In [60]:
```
def simulate_GARCH(observations, omega, alpha, beta = 0):
    np.random.seed(4)
    # Initialize the parameters
    white_noise = np.random.normal(size = observations)
    resid = np.zeros_like(white_noise)
    variance = np.zeros_like(white_noise)

    for t in range(1, observations):
        # Simulate the variance (sigma squared)
        variance[t] = omega + alpha * resid[t-1]**2 + beta * variance[t-1]
        # Simulate the residuals
        resid[t] = np.sqrt(variance[t]) * white_noise[t]

    return resid, variance
```

In [61]:
```
data = yf.download('^DJI','2000-01-01','2022-10-04').Close
data.plot(figsize=(9,7))
plt.title("Le cours de Dow Jones ")
```

```
[**********100%**********]  1 of 1 completed
```

Out[61]:
```
Text(0.5, 1.0, 'Le cours de Dow Jones ')
```



Trouver les regimes qui sont homogénes

In [62]:
```
dwj_list = np.array(data.tolist())
```

In [63]:
```
n_breaks = 3
```

In [64]:
```
#del breaks
model = rpt.Pelt(model="rbf").fit(dwj_list)
model.fit(dwj_list)
breaks = model.predict(pen=np.log(len(data)))
```

In [65]:
```
#del breaks_rpt
breaks_rpt = []
for i in breaks:
```

```
    breaks_rpt.append(data.index[i-1])
breaks_rpt = pd.to_datetime(breaks_rpt)
#breaks_rpt
```

In [66]:
```
fig, ax = plt.subplots(1, 1, figsize=(9, 5))
#fig1, ax1 = plt.subplots(1, 1, figsize=(9, 5))

plt.plot(data, label='data')
plt.title('S&P500')
print_legend = True
for i in breaks_rpt:
    if print_legend:
        plt.axvline(i, color='red',linestyle='dashed', label='breaks')
        print_legend = False
    else:
        plt.axvline(i, color='red',linestyle='dashed')

#rend = spf.pct_change().dropna()
#ax1.plot(rend)

#plt.xticks(pd.date_range(data.index.min(), data.index.max(), freq='YS'))
plt.legend()
#plt.grid()
plt.show()
```

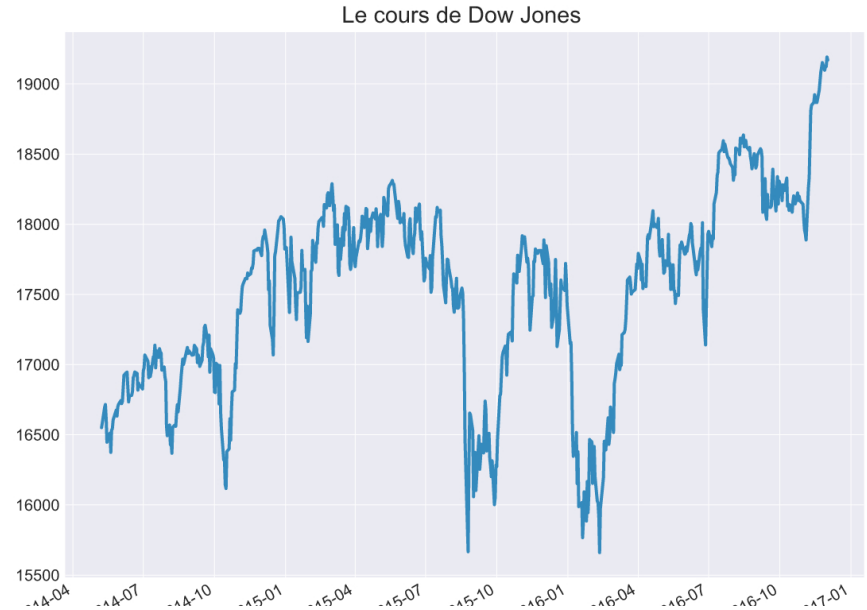
In [67]:
```
breaks_rpt
```

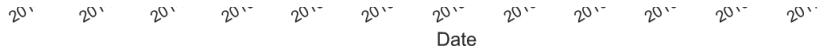
Out[67]:
```
DatetimeIndex(['2002-06-21', '2003-10-06', '2006-09-27', '2008-09-30',
               '2009-08-21', '2010-10-22', '2012-01-17', '2013-03-07',
               '2014-05-08', '2016-12-05', '2017-10-12', '2019-02-13',
               '2019-10-30', '2020-03-04', '2020-05-29', '2020-11-10',
               '2021-03-23', '2022-05-02', '2022-10-03'],
              dtype='datetime64[ns]', freq=None)
```

In [68]:
```
dowj = yf.download('^DJI','2014-05-08','2016-12-05').Close
dowj.plot(figsize=(9,7))
plt.title("Le cours de Dow Jones ")
```

```
[*********************100%***********************]  1 of 1 completed
```

Out[68]:
```
Text(0.5, 1.0, 'Le cours de Dow Jones ')
```





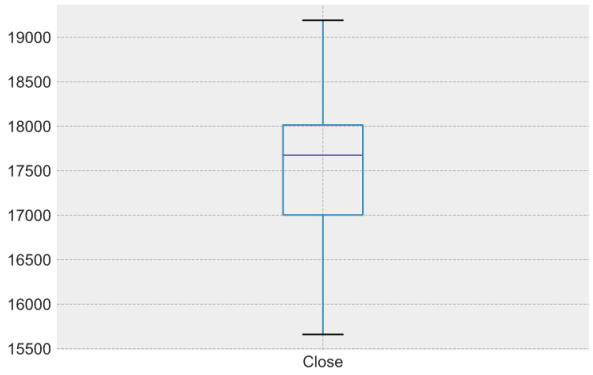
In [103]:
```
pd.DataFrame(dowj.describe()).T
```

Out[103]:

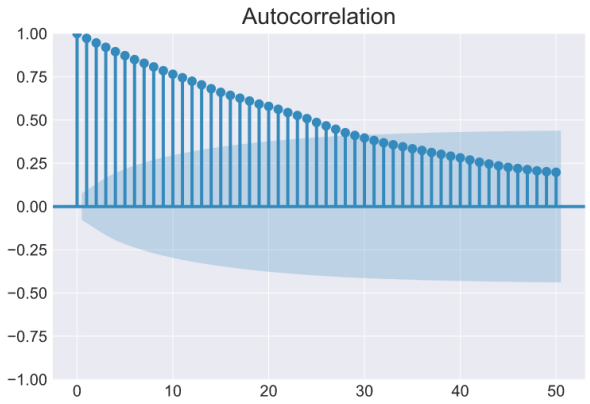
	count	mean	std	min	25%	50%	75%	max
Close	650.0	17520.362554	696.744703	15660.179688	17002.353027	17673.919922	18014.946777	19191.929688

In [105]:
```
pd.DataFrame(dowj).boxplot()
```

Out[105]: <AxesSubplot:>



In [69]:
```
from pandas.plotting import autocorrelation_plot
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf

# autocorrelation_plot(dowj)
# plt.show()
plot_acf(dowj,lags=50,alpha=0.05)
plt.show()
# print('PACF :')
# plot_pacf(dowj, alpha =0.05, lags=40)
# plt.show()
```



In [70]:
```
X = dowj
split1 = round(len(X) / 2)
X1, X2 = X[0:split1], X[split1:]
mean1, mean2 = X1.mean(), X2.mean()
var1, var2 = X1.var(), X2.var()
print('mean1=' +str(mean1)+
'   mean2= '+str(mean2))
print('variance1=' +str(var1)+
'   variance2= '+str(var2))
```

```
mean1=17447.72942908654   mean2= 17592.99567908654
variance1=296181.73525435146   variance2= 665639.2337059662
```

In [71]:
```
result = adfuller(dowj)
print('ADF Test Statistic: %.2f' % result[0])
print('5%% Critical Value: %.2f' % result[4]['5%'])
print('p-value: %.2f' % result[1])
```

```
ADF Test Statistic: -2.33
5% Critical Value: -2.87
p-value: 0.16
```

H0 : A des racines unitaires et appartient à des séries non stationnaires.

H1 : Il n'y a pas de racine unitaire et appartient à une séquence stationnaire.

Conclusion : p_value sup a 0.05 donc l'hypothèse H0 est vraie et la serie n'est pas stationnaire.

In [72]:
```
from scipy.stats import shapiro
```

In [72]:
```
from scipy.stats import shapiro
print("Test de Shapiro")
#shapiro(Fited_model.resid)
shapiro(dowj)
```

```
Test de Shapiro
```

Out[72]:
```
ShapiroResult(statistic=0.9767274260520935, pvalue=1.1967192570239149e-08)
```

la valeur p est de est inférieur à l'alpha (0,05), alors nous rejetons l'hypothèse nulle, c'est-à-dire que nous avons suffisamment de preuves pour affirmer que l'échantillon ne provient pas d'une distribution normale.

In [73]:
```
from statsmodels.stats.stattools import jarque_bera

#g = np.array(dowj)
# Using statsmodels.jarque_bera() method
gfg = jarque_bera(dowj)

print(gfg)
```

```
(17.38529685144711, 0.00016781499162088988, -0.34316611394269153, 2.586639714120484)
```

(Test statistics ,pvalue, skewness , kurtosis)

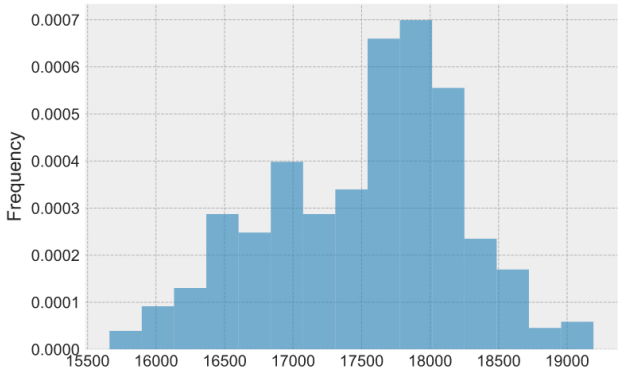
H0 : les données suivent une loi normale.

H1 : les données ne suivent pas une loi normale.

Conclusion : Pour le test de Jarque et Bera, on a trouvé une probabilité inferieure a 0.05, alors on peut déduire que la serie ne suit pas le comportement d'une normale.

In [74]:
```
plt.style.use("bmh")
ax.grid(False)
dowj.plot(kind = "hist", density = True, alpha = 0.65, bins = 15)
```

Out[74]:
```
<AxesSubplot:ylabel='Frequency'>
```



Prediction

In [75]:
```
from statsmodels.tsa.arima.model import ARIMA
import pmdarima
import arch


def split(data ,size):
    train_data = data[:int(np.ceil(len(data)*size))]
    test_data = data[len(train_data):]
    return train_data, test_data
# ------------------------------------------------------------
train_data1, test_data1 = split(dowj, 0.75)
# ------------------------------------------------------------

arima_model_fitted1 = pmdarima.auto_arima(train_data1)
arima_residuals1 = arima_model_fitted1.arima_res_.resid

# fit a GARCH(1,1) model on the residuals of the ARIMA model
garch1 = arch.arch_model(arima_residuals1, p=1, q=1);
garch_fitted1 = garch1.fit();

# Use ARIMA to predict mu
Model = ARIMA(train_data1, order=arima_model_fitted1.order)
Fited_model1 =  Model.fit()
arima_predict1 = pd.DataFrame(columns=[])
arima_predict1['predicted'] = Fited_model1.forecast(steps=len(test_data1))  #Fited_model.predict(start =len(sp_return_trn) ,end=l
arima_predict1['Date'] = test_data1.index # np.arange(0,60)
arima_predict1= arima_predict1.set_index('Date')

# Use GARCH to predict the residual
sim_resid1, sim_variance1= simulate_GARCH(observations =len(test_data1) ,omega = garch_fitted1.params[1], alpha = garch_fitted1.p
```

. . .

In [76]:

In [76]:
```
Fited_model1.predict()
```

Out[76]:
```
Date
2014-05-08        0.000000
2014-05-09    16550.970703
2014-05-12    16583.339844
2014-05-13    16695.470703
2014-05-14    16715.439453
                  ...
2016-04-08    17541.960938
2016-04-11    17576.960938
2016-04-12    17556.410156
2016-04-13    17721.250000
2016-04-14    17908.279297
Name: predicted_mean, Length: 488, dtype: float64
```

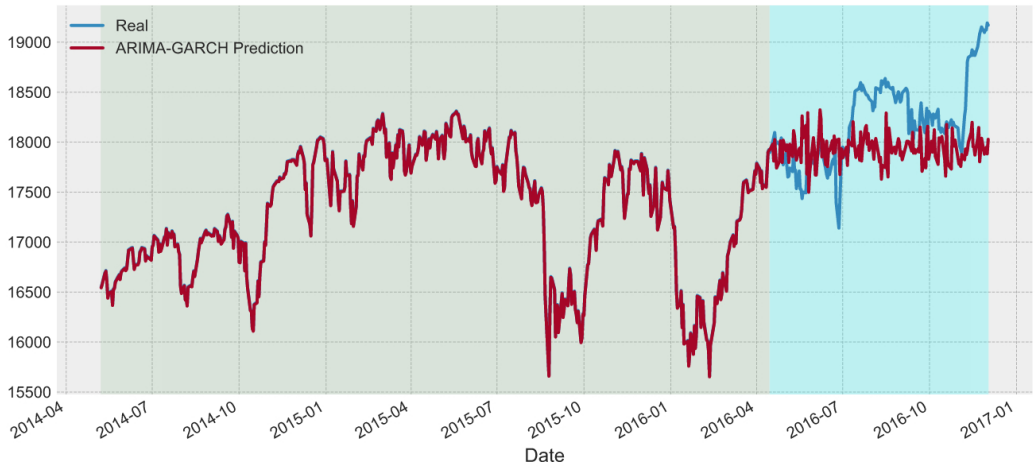
In [77]:
```
len(sim_resid1)
len(arima_predict1.predicted)
len(garch_fitted1.resid) + len(sim_resid1)
len(dowj)
```

Out[77]:
```
650
```

In [78]:
```
df1 = pd.DataFrame()
df1['Real'] = dowj
df1['ARIMA-GARCH Prediction'] = pd.concat([pd.Series(garch_fitted1.resid+Fited_model1.predict()), arima_predict1.predicted + sim_
```

In [79]:
```
df1.plot(figsize=(11,5))
plt.axvspan(train_data1.index[0], train_data1.index[-1], alpha=0.2, color='darkseagreen')
plt.axvspan(test_data1.index[0], test_data1.index[-1], alpha=0.2, color='cyan')
```

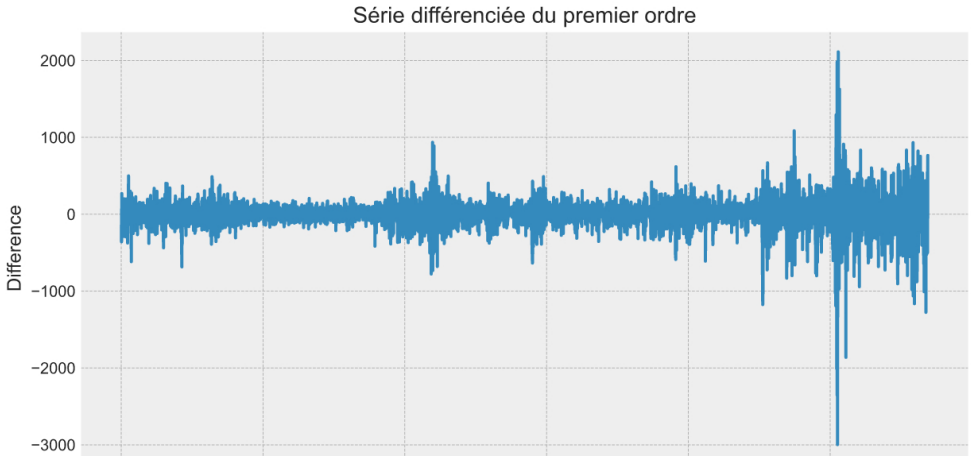
Out[79]:
```
<matplotlib.patches.Polygon at 0x1ca0e2d9b70>
```

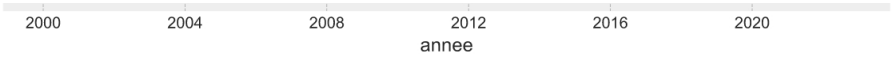


In [80]:
```
from sklearn.metrics import mean_squared_error
rmse1 = mean_squared_error(df1['ARIMA-GARCH Prediction'].loc['2016-04-15':], dowj.loc['2016-04-15':], squared=False)
rmse1
```

Out[80]:
```
501.33158954846874
```

In [81]:
```
dow_diff = pd.DataFrame()
dow_diff['Difference'] = data.diff().dropna()

# Plot the Change
plt.figure(figsize=(10, 5))
plt.plot(dow_diff['Difference'])
plt.title('Série différenciée du premier ordre', fontsize=14)
plt.xlabel('annee', fontsize=12)
plt.ylabel('Difference', fontsize=12)
plt.show()
```





In [82]:
```
result = adfuller(dowj.diff().dropna())
print(result)
print('ADF Test Statistic: %.2f' % result[0])
print('5%% Critical Value: %.2f' % result[4]['5%'])
print('p-value: %.2f' % result[1])
```

```
(-26.226977579632806, 0.0, 0, 648, {'1%': -3.4404817800778034, '5%': -2.866010569916275, '10%': -2.569150763698369}, 8053.8646854642775)
ADF Test Statistic: -26.23
5% Critical Value: -2.87
p-value: 0.00
```

H0 : A des racines unitaires et appartient à des séries non stationnaires.

H1 : Il n'y a pas de racine unitaire et appartient à une séquence stationnaire.

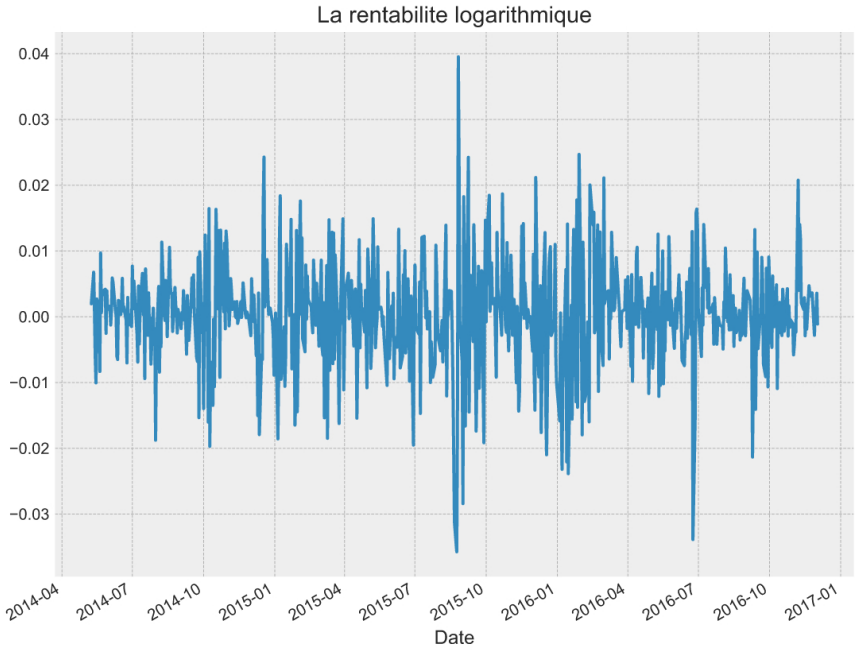
Conclusion: p_value inferieur a 0.05 donc l'hypothèse H1 est vraie et la serie est stationnaire.

Etude de Rentabilité

In [83]:
```
rent_log= dowj.pct_change().dropna()
```

In [84]:
```
rent_log.plot(figsize=(9,7))
plt.title("La rentabilite logarithmique")
```

Out[84]:
```
Text(0.5, 1.0, 'La rentabilite logarithmique')
```



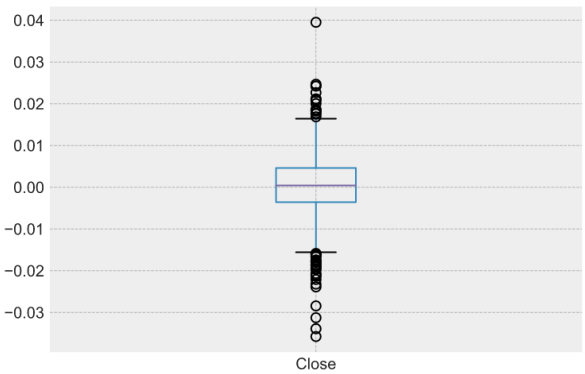
In [106]:
```
pd.DataFrame(rent_log.describe()).T
```

Out[106]:

	count	mean	std	min	25%	50%	75%	max
Close	649.0	0.000262	0.008476	-0.035748	-0.003549	0.00044	0.0046	0.039516

In [107]:
```
pd.DataFrame(rent_log).boxplot()
```

Out[107]:
```
<AxesSubplot:>
```



In [85]:
```
result = adfuller(rent_log)
print(result)
print('ADF Test Statistic: %.2f' % result[0])
print('5%% Critical Value: %.2f' % result[4]['5%'])
print('p-value: %.2f' % result[1])
```

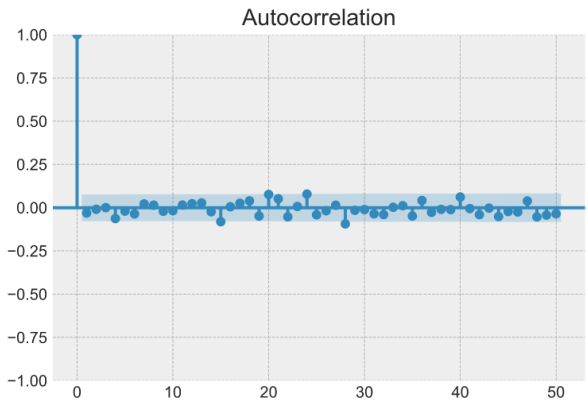
```
(-26.21805202162574, 0.0, 0, 648, {'1%': -3.4404817800778034, '5%': -2.866010569916275, '10%': -2.569150763698369}, -4192.96725
0250338)
ADF Test Statistic: -26.22
5% Critical Value: -2.87
p-value: 0.00
```

H0 : A des racines unitaires et appartient à des séries non stationnaires.

H1 : Il n'y a pas de racine unitaire et appartient à une séquence stationnaire.

Conclusion : p_value inferieur a 0.05 donc l'hypothèse H1 est vraie et la serie est stationnaire.

In [86]:
```
from pandas.plotting import autocorrelation_plot
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf , plot_pacf

plot_acf(rent_log,lags=50,alpha=0.05)
plt.show()
```



Nous constatons que le coefficient d'autocorrélation décroit rapidement vers Zéro, On peut considérer que la série a fluctué autour de l'axe zéro qui est une caractéristique typique d'une série stationnaire

In [87]:
```
X = rent_log
split1 = round(len(X) / 2)
X1, X2 = X[0:split1], X[split1:]
mean1, mean2 = X1.mean(), X2.mean()
var1, var2 = X1.var(), X2.var()
print('mean1=' +str(mean1)+
'   mean2= '+str(mean2))
print('variance1=' +str(var1)+
'   variance2= '+str(var2))
```

```
mean1=0.00010809188632999789   mean2= 0.0004160621074486964
variance1=5.4460558283046336e-05   variance2= 8.934058636797217e-05
```

In [88]:
```
from scipy.stats import shapiro
print("Test de Shapiro")
#shapiro(Fited_model.resid)
shapiro(rent_log)
```

```
Test de Shapiro
```

Out[88]:
```
ShapiroResult(statistic=0.9732717275619507, pvalue=1.6719958750854857e-09)
```

la valeur p est de est inférieur à l'alpha (0,05), alors nous rejetons l'hypothèse nulle, c'est-à-dire que nous avons suffisamment de preuves pour affirmer que l'échantillon ne provient pas d'une distribution normale.

In [89]:
```
from statsmodels.stats.stattools import jarque_bera

#g = np.array(dowj)
# Using statsmodels.jarque_bera() method
gfg = jarque_bera(rent_log)

print(gfg)
```

```
(102.94333983796754, 4.4272880571255853e-23, -0.21149858167376795, 4.904708621978709)
```

(Test statistics ,pvalue, skewness , kurtosis)

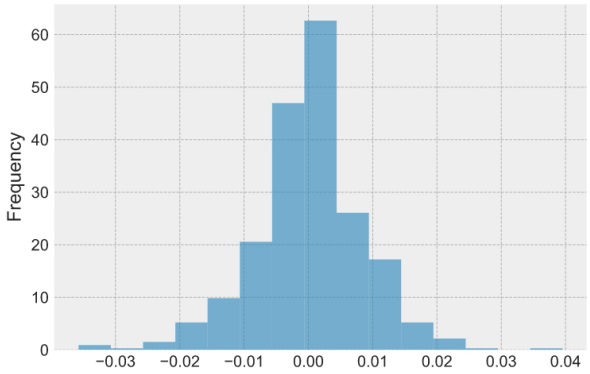
H0 : les données suivent une loi normale.

H1 : les données ne suivent pas une loi normale.

Conclusion : Pour le test de Jarque et Bera, on a trouvé que la probabilité pvalue est superieure a 0.05, alors on peut déduire que les valeurs de rentabilités sont normales.

In [90]:
```
plt.style.use("bmh")
ax.grid(False)
rent_log.plot(kind = "hist", density = True, alpha = 0.65, bins = 15)
```

Out[90]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [91]:
```
from statsmodels.tsa.arima.model import ARIMA
import pmdarima
import arch

# ---------------------------------------------------------
train_data, test_data = split(rent_log, 0.75)
# ---------------------------------------------------------


arima_model_fitted = pmdarima.auto_arima(train_data)
arima_residuals = arima_model_fitted.arima_res_.resid

# fit a GARCH(1,1) model on the residuals of the ARIMA model
garch = arch.arch_model(arima_residuals, p=1, q=1);
garch_fitted = garch.fit();

# Use ARIMA to predict mu
Model = ARIMA(train_data, order=arima_model_fitted.order)
Fited_model =  Model.fit()
arima_predict = pd.DataFrame(columns=[])
arima_predict['predicted'] = Fited_model.forecast(steps=len(test_data))  #Fited_model.predict(start =len(sp_return_trn) ,end=len(
arima_predict['Date'] = test_data.index # np.arange(0,60)
arima_predict= arima_predict.set_index('Date')

# Use GARCH to predict the residual
sim_resid, sim_variance= simulate_GARCH(observations =len(test_data) ,omega = garch_fitted.params[1], alpha = garch_fitted.params
```

. . .

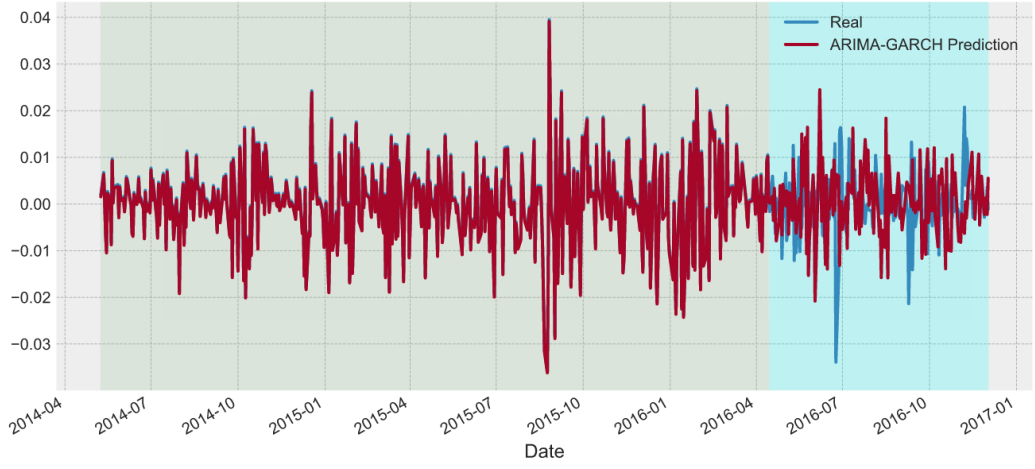
In [92]:
```
df = pd.DataFrame()
df['Real'] = rent_log
df['ARIMA-GARCH Prediction'] = pd.concat([pd.Series(garch_fitted.resid + Fited_model.predict()), arima_predict.predicted + sim_re
```

In [93]:
```
test_data.index[0]
```

Out[93]:
```
Timestamp('2016-04-15 00:00:00')
```

In [94]:
```
df.plot(figsize=(11,5))
plt.axvspan(train_data.index[0], train_data.index[-1], alpha=0.2, color='darkseagreen')
plt.axvspan(test_data.index[0], test_data.index[-1], alpha=0.2, color='cyan')
```

Out[94]:
```
<matplotlib.patches.Polygon at 0x1ca0dbfa050>
```



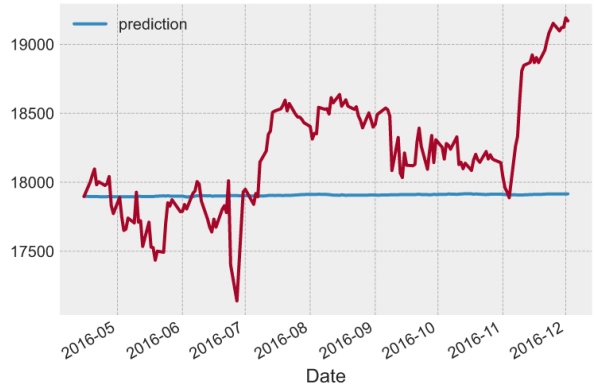
In [95]:
```
def convert_return_to_price(start_point , prediction):
    predicted_price = pd.DataFrame(index = prediction.index)
    predicted_price['prediction'] = start_point * (1 +  (prediction/100)).cumprod()

    return predicted_price
```

In [96]:
```
dowj.loc['2016-04-15']
#test_data.index
df['ARIMA-GARCH Prediction'].loc['2016-04-15']
len(test_data)
```

Out[96]: 162

In [97]:
```
convert_return_to_price(dowj.loc['2016-04-15'],df['ARIMA-GARCH Prediction'].loc['2016-04-15':]).plot()
dowj.loc['2016-04-15':].plot()
```

Out[97]: <AxesSubplot:xlabel='Date'>



In [98]:
```
rmse = mean_squared_error(df['ARIMA-GARCH Prediction'].loc['2016-04-15':], dowj.loc['2016-04-15':], squared=False)
rmse
```

Out[98]: 18205.28224692338