Étude de cas : Analysis and Forecasting of House Price Indices

Projet de François CREPIN et d'Adrien RUGGIERO

Importation des librairies utiles

Dans cette première sous partie nous regrouperons tous les modules dont nous allons avoir besoin lors de notre projet.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import autocorrelation_plot
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from pandas.plotting import register_matplotlib_converters
from datetime import datetime
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_predict
```

In [2]:
```
plt.rcParams['figure.figsize'] = [10, 5]
```

Création du parser pour les indexs du dataset qui sont des dates

In [3]:
```
def parser(x):
    return datetime.strptime(x, '%d-%b-%y')
```

Importation du dataset

In [4]:
```
df = pd.read_csv('data/house_index_canberra.csv', names=['date','index'], parse_dat
df.drop(df.tail(1).index,inplace=True)
#df.columns = ['Date','Index']
```

In [5]:
```
df.head()
```

Out[5]:

date	index
1990-01-31	1.00763
1990-02-28	1.01469
1990-03-31	1.02241
1990-04-30	1.03062
1990-05-31	1.03903

In [6]:
```
df.index
```

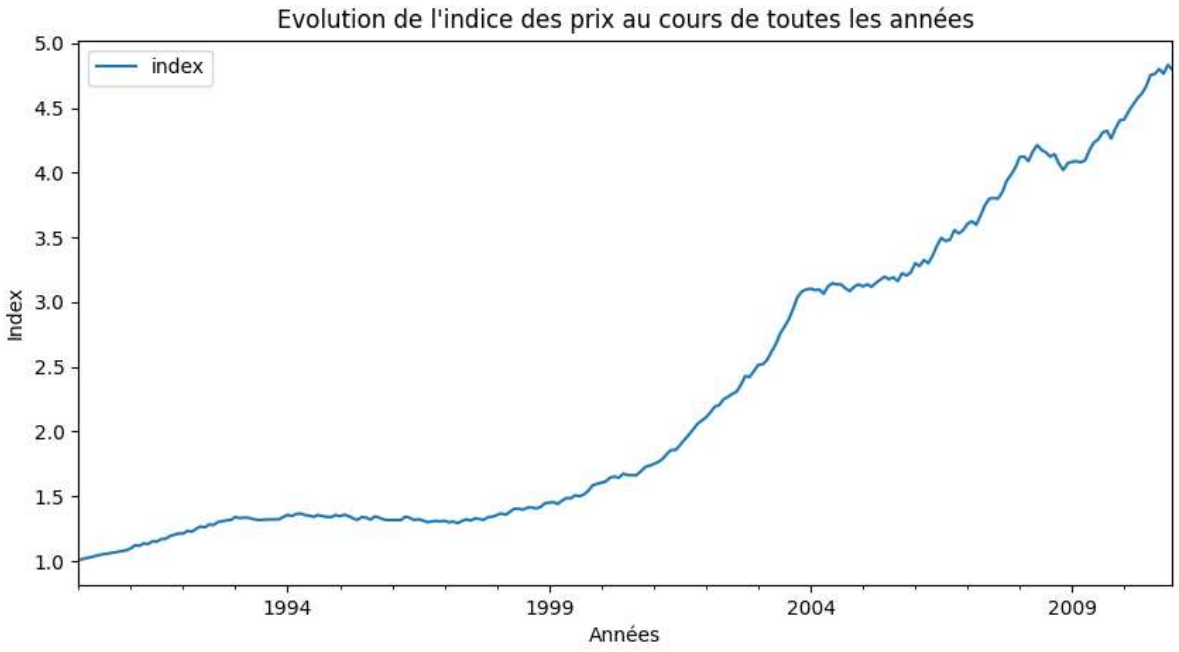
Out[6]:
```
DatetimeIndex(['1990-01-31', '1990-02-28', '1990-03-31', '1990-04-30',
               '1990-05-31', '1990-06-30', '1990-07-31', '1990-08-31',
               '1990-09-30', '1990-10-31',
               ...
               '2010-03-31', '2010-04-30', '2010-05-31', '2010-06-30',
               '2010-07-31', '2010-08-31', '2010-09-30', '2010-10-31',
               '2010-11-30', '2010-12-31'],
              dtype='datetime64[ns]', name='date', length=252, freq=None)
```

Exploration des données

Visualisation des données

In [7]:
```
df.plot()
plt.xlabel("Années")
plt.ylabel("Index")
plt.title("Evolution de l'indice des prix au cours de toutes les années")
plt.legend(['index'],loc = 'best')
```

Out[7]:
```
<matplotlib.legend.Legend at 0x7f8dc073d610>
```



Statistiques globales

In [8]:
```
#On créer un dataset juste pour faire de l'exploration statistiques
df_stats = df.copy()
df_stats.insert(1,'Annee' ,value = df_stats.index.year)  #On ajoute une colonne qui
df_stats.head()
```

Out[8]:

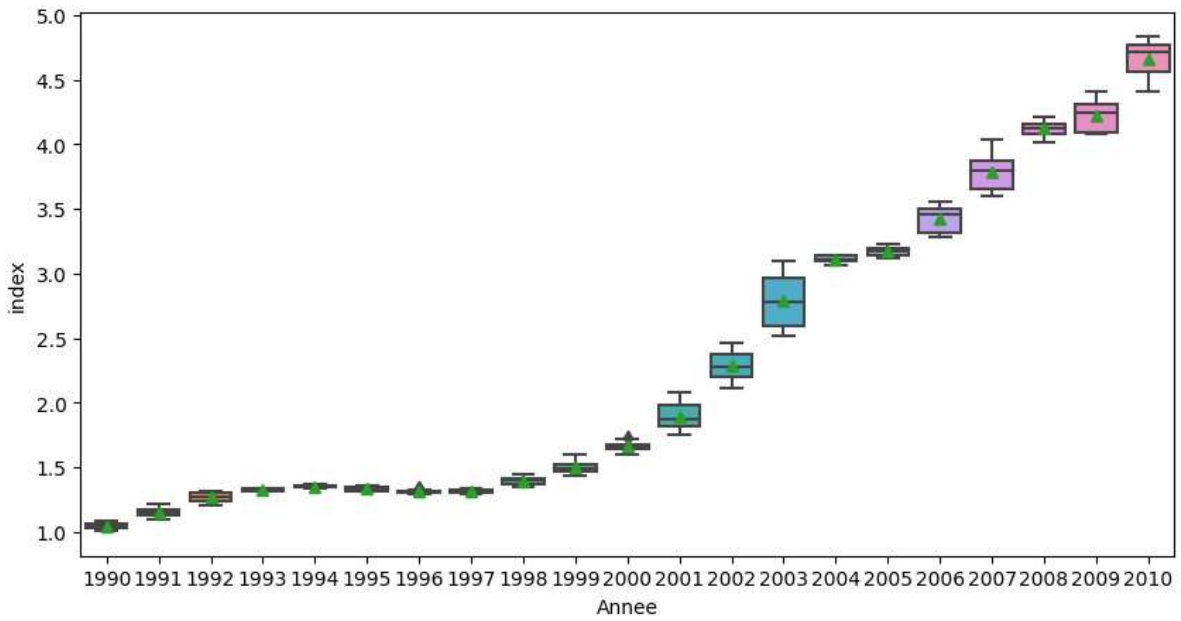
date	index	Annee
1990-01-31	1.00763	1990
1990-02-28	1.01469	1990
1990-03-31	1.02241	1990
1990-04-30	1.03062	1990
1990-05-31	1.03903	1990

Affichage par année

On affiche les informations essentielles à travers des boîtes à moustaches. De plus elles seront visuellement plus agréables (couleurs et formes) grâce à la bibliothèque 'seaborn'.

In [9]:
```
sns.boxplot(x = 'Annee', y = 'index', data = df_stats, showmeans = True)
```

Out[9]:
```
<AxesSubplot:xlabel='Annee', ylabel='index'>
```



Remarque : On peut ainsi voir les gros changements qu'il y a eu année par année

Affichage par trimestre

In [10]:
```
#On ajoute une colonne "Trimestre", de la meme manière que pour "Annee"
df_stats.insert(2,"Trimestre", value = df_stats.index.quarter)
```

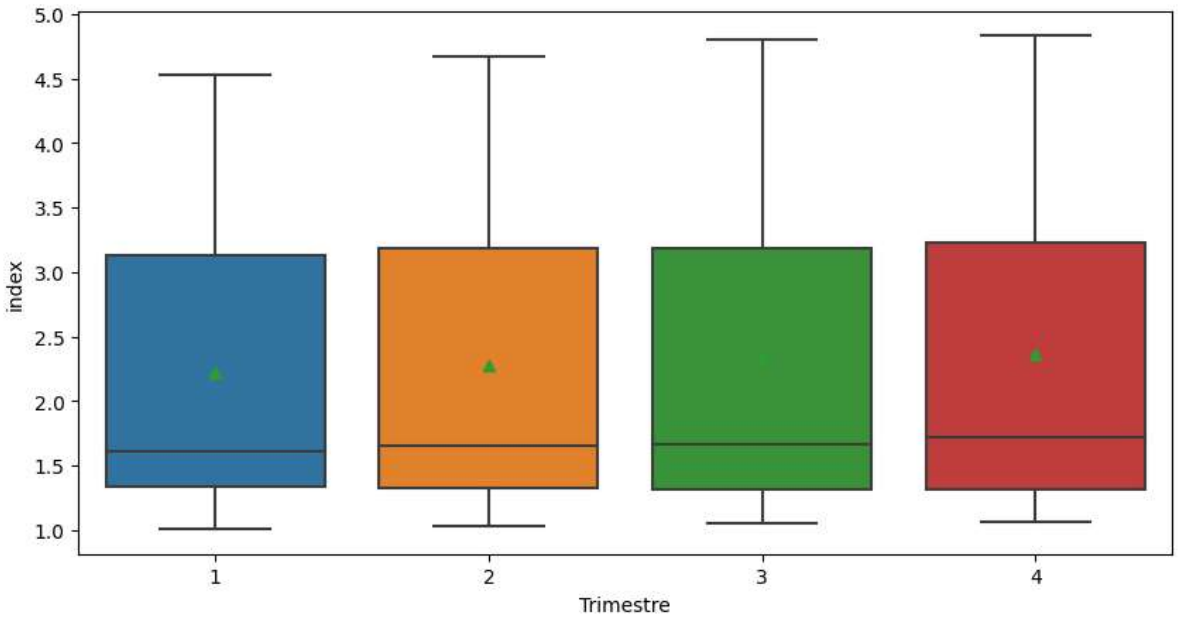
In [11]:
```
df_stats.head(12)
```

Out[11]:

date	index	Annee	Trimestre
1990-01-31	1.00763	1990	1
1990-02-28	1.01469	1990	1
1990-03-31	1.02241	1990	1
1990-04-30	1.03062	1990	2
1990-05-31	1.03903	1990	2
1990-06-30	1.04729	1990	2
1990-07-31	1.05276	1990	3
1990-08-31	1.05833	1990	3
1990-09-30	1.06390	1990	3
1990-10-31	1.06977	1990	4
1990-11-30	1.07594	1990	4
1990-12-31	1.08253	1990	4

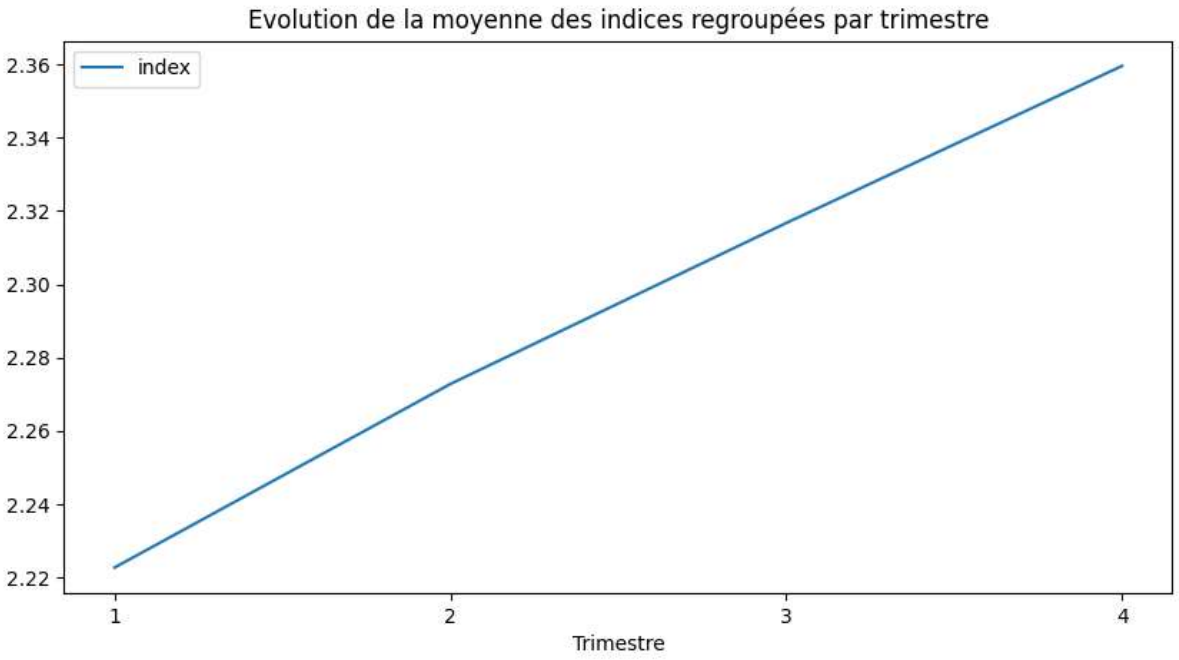
In [12]:
```
sns.boxplot(x = 'Trimestre', y = 'index', data = df_stats, showmeans = True)
```

Out[12]: <AxesSubplot:xlabel='Trimestre', ylabel='index'>



In [13]:
```
data_trimestre = df_stats[['index','Trimestre']].groupby("Trimestre").mean()
data_trimestre.plot()
plt.xticks([1,2,3,4])
plt.title("Evolution de la moyenne des indices regroupées par trimestre")
```

Out[13]: Text(0.5, 1.0, 'Evolution de la moyenne des indices regroupées par trimestre')

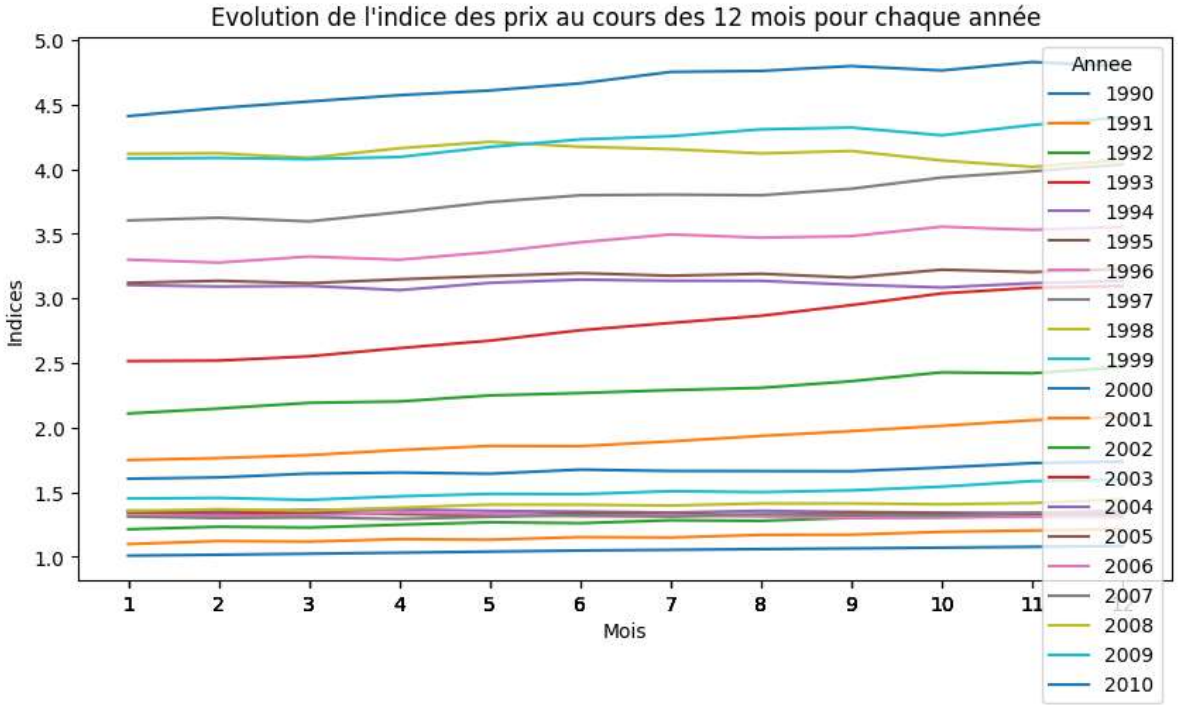


Remarque : On voit que les moyennes des indices par trimestre sont plus élevés pour les deux derniers trimestres. Plus généralement, l'indice augmente au à peu près linéairement du trimestre 1 au trimestre 4 sur une année.

Nous allons maintenant voir la tendance de l'indice de prix au cours des 12 mois pour **chaque** année.

In [14]:
```
pivot = pd.pivot_table(df_stats,values='index', index=df_stats.index.month,columns=
pivot.plot(legend=True)
plt.xlabel("Mois")
plt.ylabel("Indices")
plt.xticks(np.linspace(1,12, dtype=int))
plt.title("Evolution de l'indice des prix au cours des 12 mois pour chaque année")
```

Out[14]:
```
Text(0.5, 1.0, "Evolution de l'indice des prix au cours des 12 mois pour chaque an
née")
```



On voit bien que, généralement, au sein d'une année, l'indice de prix a tendance à augmenter au fil des mois de l'année.

Affichage des données avec la méthode des fenêtres glissantes

In [15]:
```
rolling_mean = df.rolling(window = 12).mean()
rolling_std = df.rolling(window = 12).std()
print(rolling_mean[12:])
print(rolling_std[12:])
```

```
               index
date
1991-01-31  1.054529
1991-02-28  1.063397
1991-03-31  1.071222
1991-04-30  1.079982
1991-05-31  1.087653
...              ...
2010-08-31  4.509383
2010-09-30  4.549086
2010-10-31  4.590942
2010-11-30  4.631597
2010-12-31  4.664368

[240 rows x 1 columns]
               index
date
1991-01-31  0.024998
1991-02-28  0.028244
1991-03-31  0.028858
1991-04-30  0.031265
1991-05-31  0.031596
...              ...
2010-08-31  0.167135
2010-09-30  0.175313
2010-10-31  0.160179
2010-11-30  0.153423
2010-12-31  0.142018

[240 rows x 1 columns]
```

On extrait quelques prorpiétés de la série temporelle : Stationarité

Le test de Dickey Fuller augmenté (test ADF) est un test statistique commun utilisé pour tester si une série temporelle donnée est stationnaire ou non. C'est l'un des tests statistiques les plus couramment utilisés lorsqu'il s'agit d'analyser la stationnarité d'une série.

In [16]:
```
result = adfuller(df['index'])

print('Statistiques ADF : {}'.format(result[0]))
print('p-value : {}'.format(result[1]))
print('Valeurs Critiques :')
for key, value in result[4].items():
    print('\t{}: {}'.format(key, value))
```

```
Statistiques ADF : 1.3416966708882263
p-value : 0.9968324926277587
Valeurs Critiques :
        1%: -3.458010773719797
        5%: -2.8737103617125186
        10%: -2.5732559963936206
```

In [17]:
```
# Logarithme de la variable dépendante : réduire le taux d'augmentation de la moyer
df_log = np.log(df)
```

In [18]:
```
def get_stationarity(timeseries, title):

    result = adfuller(timeseries['index'])
    print('Statistiques ADF : {}'.format(result[0]))
    print('p-value : {}'.format(result[1]))
    print('Valeurs Critiques :')
    for key, value in result[4].items():
        print('\t{}: {}'.format(key, value))
```

In [19]:
```
rolling_mean = df_log.rolling(window=12).mean()
df_log_minus_mean = df_log - rolling_mean
df_log_minus_mean.dropna(inplace=True)
get_stationarity(df_log_minus_mean, 'Moyenne mobile et eccart type avec origine = 1
```

```
Statistiques ADF : -2.042192784985336
p-value : 0.2683847721477889
Valeurs Critiques :
        1%: -3.459105583381277
        5%: -2.8741898504150574
        10%: -2.5735117958412097
```

Interprétations : D'après les propriétés des tests "ADF", on peut déduire que la série temporelle n'est pas stationnaire car la p-value est plus grande que 0.05 (elle vaut 0.27 environ)

In [20]:
```
rolling_mean_exp_decay = df_log.ewm(halflife=12, min_periods=0, adjust=True).mean()
df_log_exp_decay = df_log - rolling_mean_exp_decay
df_log_exp_decay.dropna(inplace=True)
get_stationarity(df_log_exp_decay, 'décroissance exponentielle')

df_log_shift = df_log - df_log.shift()
df_log_shift.dropna(inplace=True)
get_stationarity(df_log_shift ,'log - log.shift')
```

```
Statistiques ADF : -1.765172266931151
p-value : 0.39788457332375154
Valeurs Critiques :
        1%: -3.457105309726321
        5%: -2.873313676101283
        10%: -2.5730443824681606
Statistiques ADF : -4.621664564677448
p-value : 0.00011779996355354567
Valeurs Critiques :
        1%: -3.457105309726321
        5%: -2.873313676101283
        10%: -2.5730443824681606
```

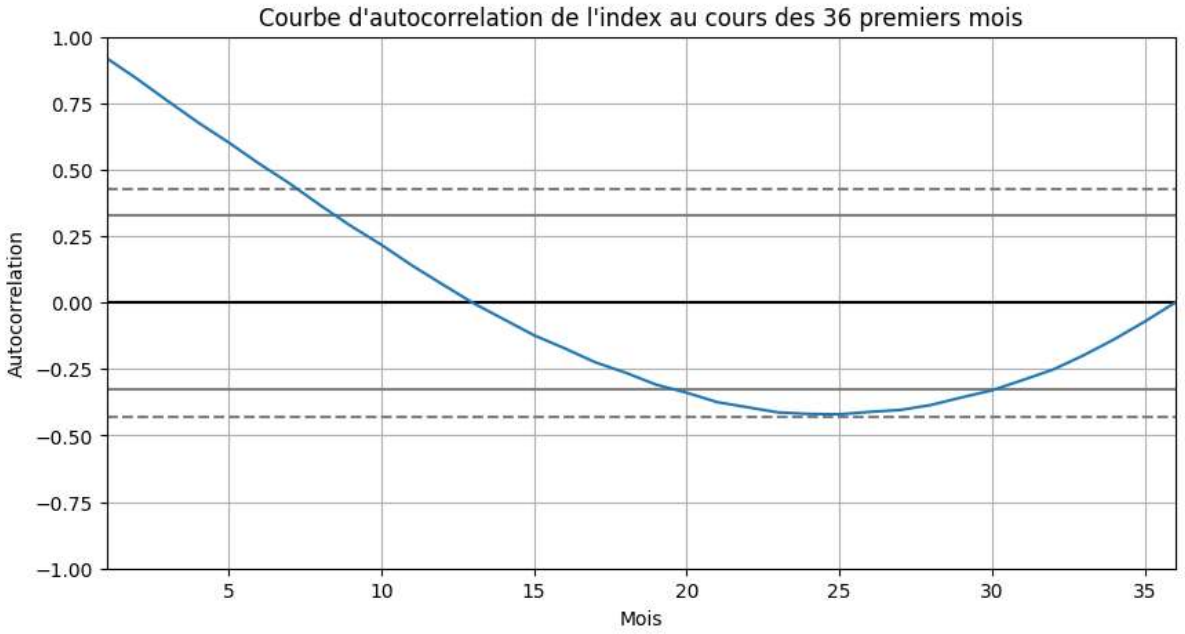
Méthode ARIMA

Le modèle ARIMA (AutoRegressive Integrated Moving Average) est un modèle de prévision des séries temporelles qui incorpore des mesures d'autocorrélation pour modéliser les structures temporelles au sein des données des séries temporelles afin de prédire les valeurs futures.

Trois facteurs définissent le modèle ARIMA, il est défini comme ARIMA(p,d,q) où p, d et q désignent respectivement le nombre d'observations passées à prendre en compte pour l'autorégression, le nombre de fois où les observations brutes sont différenciées et la taille de la fenêtre de la moyenne mobile.

In [21]:
```
#On cherche le paramètre p
autocorrelation_plot(df['index'].iloc[0:36])
plt.xlabel("Mois")
plt.ylabel("Autocorrelation")
plt.title("Courbe d'autocorrelation de l'index au cours des 36 premiers mois")
```

Out[21]:
```
Text(0.5, 1.0, "Courbe d'autocorrelation de l'index au cours des 36 premiers moi
s")
```



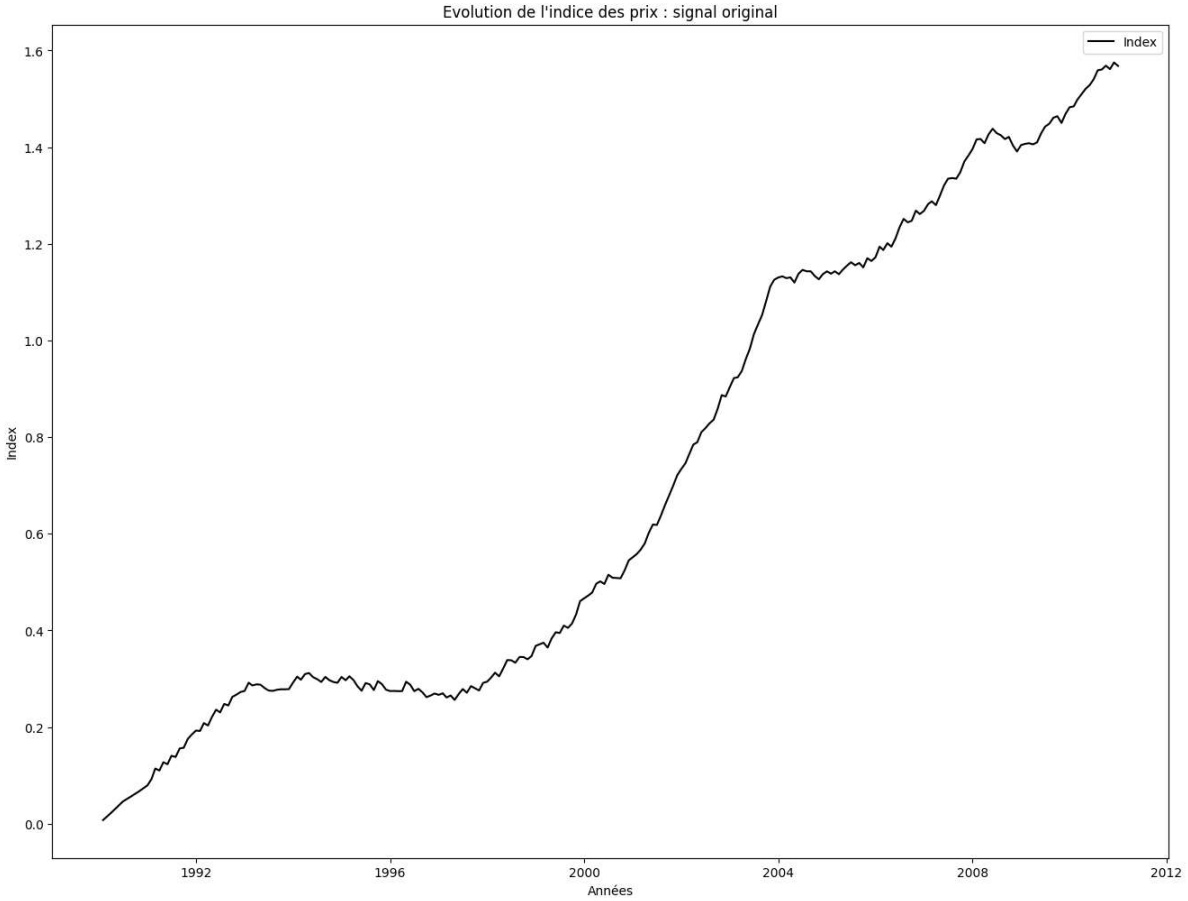
La courbe rentre dans la zone délimitée par les droites horizontales en pointillés au niveau du Mois 4. Ainsi, pour nos paramètres (p,d,q), la valeur p prend la valeur 4.

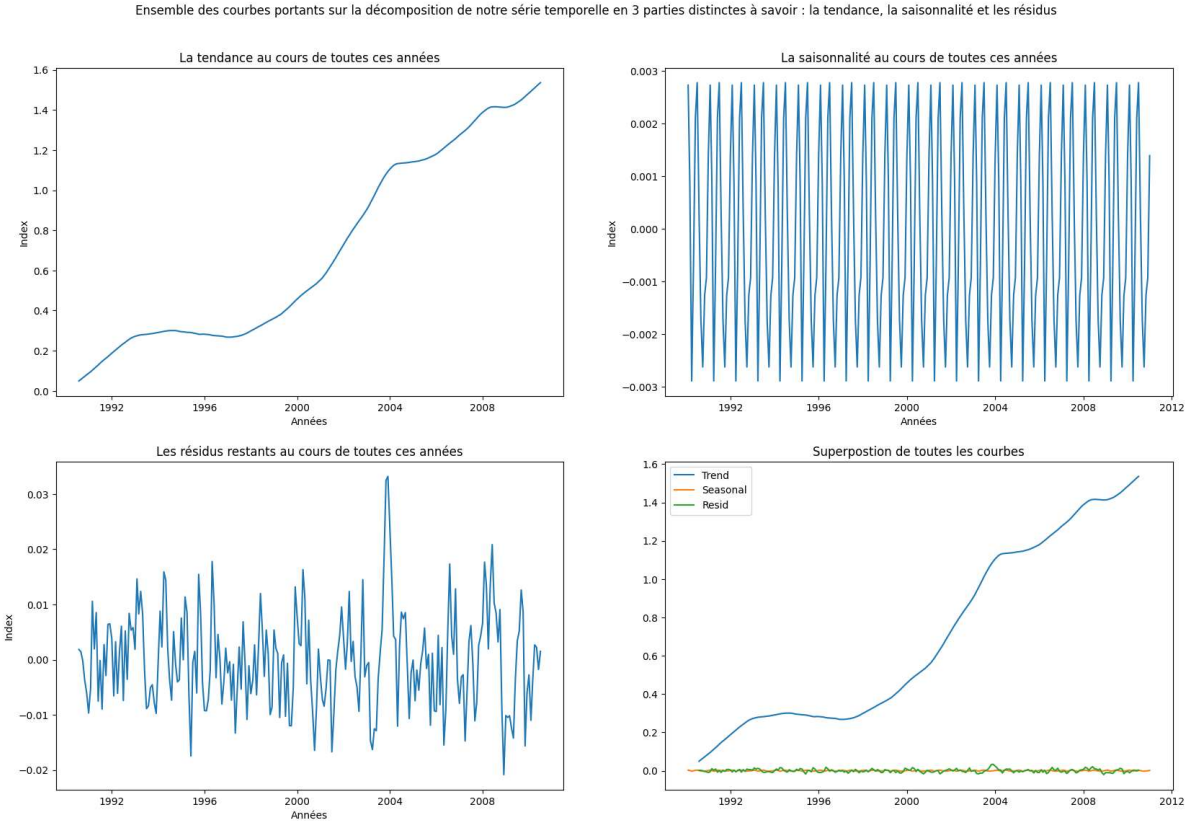
In [22]:
```
decomposition = seasonal_decompose(df_log)
seasonal = decomposition.seasonal
trend = decomposition.trend
resid = decomposition.resid
```

In [ ]:
```
```

In [23]:
```
plt.figure(1 , figsize =(16,12))
plt.plot(df_log.index, df_log, 'k')
plt.legend(['Index'])
plt.xlabel("Années")
plt.ylabel("Index")
plt.title("Evolution de l'indice des prix : signal original")

plt.figure(2 , figsize =(20,13))
plt.suptitle("\n\nEnsemble des courbes portants sur la décomposition de notre série
```

```
plt.subplot(2, 2, 1)
plt.plot(df_log.index,trend)
plt.xlabel("Années")
plt.ylabel("Index")
plt.title("La tendance au cours de toutes ces années")

plt.subplot(2, 2, 2)
plt.plot(df_log.index,seasonal)
plt.xlabel("Années")
plt.ylabel("Index")
plt.title("La saisonnalité au cours de toutes ces années")

plt.subplot(2, 2, 3)
plt.plot(df_log.index,resid) #ce qui reste après avoir éliminé les tendances et la
plt.xlabel("Années")
plt.ylabel("Index")
plt.title("Les résidus restants au cours de toutes ces années")

plt.subplot(2, 2, 4)
plt.plot(df_log.index,trend)
plt.plot(df_log.index,seasonal)
plt.plot(df_log.index,resid)
plt.title("Superpostion de toutes les courbes")
plt.legend(['Trend', 'Seasonal', 'Resid'],loc='best')
plt.show()
```





Remarque : L'indice des prix n'a jamais connu de grosses décroissance. La plus forte baisse depuis les 1990 semble être en 2007-2008, lors de la crise des "sub-primes".

In [24]:
```
model = ARIMA(df, order=(4,1,1))
results = model.fit()
results.summary()
```

```
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: Valu
eWarning: No frequency information was provided, so inferred frequency M will be u
sed.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: Valu
eWarning: No frequency information was provided, so inferred frequency M will be u
sed.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471: Valu
eWarning: No frequency information was provided, so inferred frequency M will be u
sed.
  self._init_dates(dates, freq)
/opt/conda/lib/python3.7/site-packages/statsmodels/base/model.py:606: ConvergenceW
arning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  ConvergenceWarning)
```

Out[24]:

SARIMAX Results

Dep. Variable:	index	**No. Observations:**	252
Model:	ARIMA(4, 1, 1)	**Log Likelihood**	553.719
Date:	Tue, 04 Oct 2022	**AIC**	-1095.439
Time:	14:50:07	**BIC**	-1074.286
Sample:	01-31-1990	**HQIC**	-1086.926
	- 12-31-2010		
Covariance Type:	opg		

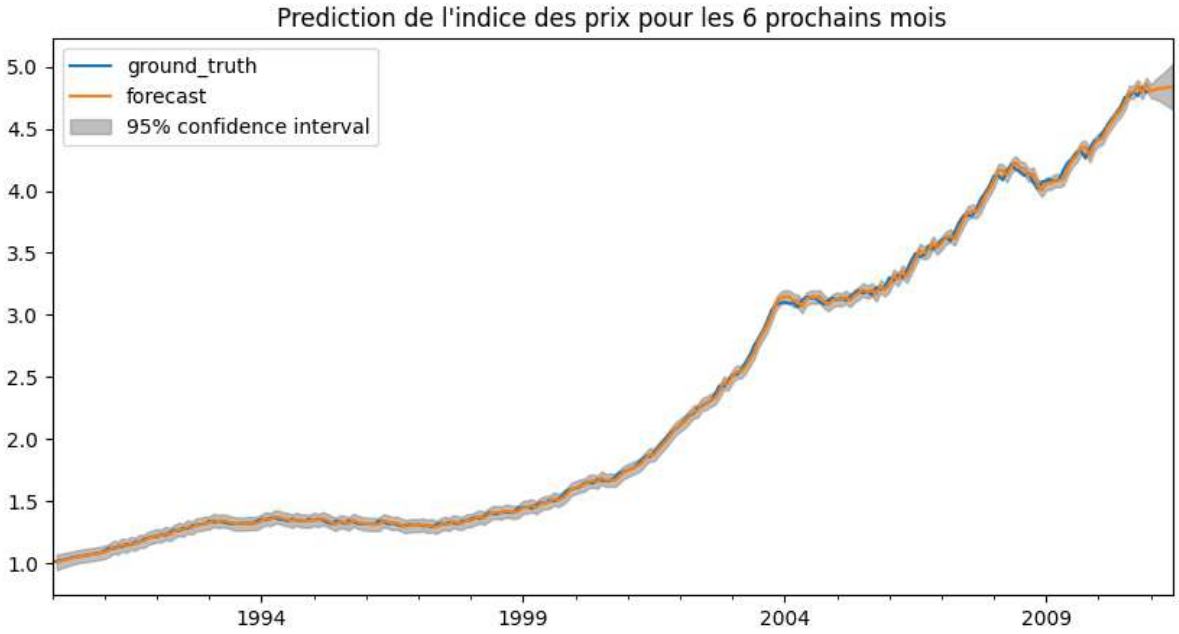
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.6305	0.138	4.567	0.000	0.360	0.901
ar.L2	-0.0058	0.058	-0.101	0.919	-0.119	0.107
ar.L3	0.2004	0.060	3.366	0.001	0.084	0.317
ar.L4	0.0739	0.077	0.964	0.335	-0.076	0.224
ma.L1	-0.5046	0.134	-3.767	0.000	-0.767	-0.242
sigma2	0.0007	5.66e-05	12.509	0.000	0.001	0.001

Ljung-Box (L1) (Q):	0.07	**Jarque-Bera (JB):**	14.66
Prob(Q):	0.80	**Prob(JB):**	0.00
Heteroskedasticity (H):	12.57	**Skew:**	-0.36
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.94

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

In [25]:
```
df.columns = ["ground_truth"]
```

In [26]:
```
fig, ax = plt.subplots()
df.index = pd.date_range(start='1990', end='2011', freq='M')
ax = df.loc['1990':].plot(ax=ax)
fig = plot_predict(results,1, 257,plot_insample=False, ax = ax) #On fait une predic
plt.title("Prediction de l'indice des prix pour les 6 prochains mois")
plt.show()
```



Recherche du meilleur model

On utilise une librairie qui recherche les meilleurs hyperparamètres pour ARIMA

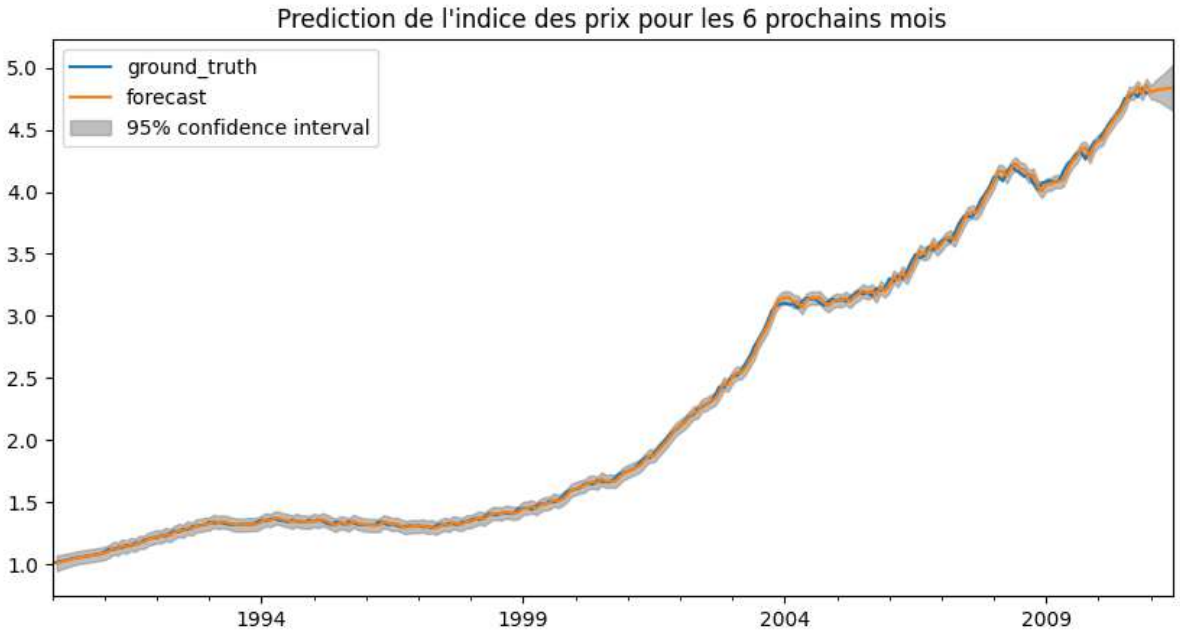
In [27]:
```
import pmdarima as pm
model_auto = pm.auto_arima(df.values, start_p=1, start_q=1,
                      test='adf',       # use adftest to find optimal 'd'
                      max_p=12, max_q=3, # maximum p and q
                      m=1,              # frequency of series
                      d=None,           # let model determine 'd'
                      seasonal=True,   # No Seasonality
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=-1098.089, Time=0.31 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=-1069.339, Time=0.05 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=-1079.163, Time=0.11 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=-1077.243, Time=0.07 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=-1009.331, Time=0.03 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=-1087.556, Time=0.48 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=-1096.942, Time=0.40 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=-1076.418, Time=0.13 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=-1080.691, Time=0.16 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=-1094.101, Time=0.47 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=-1095.209, Time=0.16 sec

Best model:  ARIMA(1,1,1)(0,0,0)[0] intercept
Total fit time: 2.424 seconds
```

In [28]:
```
model_opti = ARIMA(df, order=(1,1,1))
results_opti = model.fit()
```

```
/opt/conda/lib/python3.7/site-packages/statsmodels/base/model.py:606: ConvergenceW
arning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  ConvergenceWarning)
```

In [29]:
```
fig, ax = plt.subplots()
df.index = pd.date_range(start='1990', end='2011', freq='M')
ax = df.loc['1990':].plot(ax=ax)
fig = plot_predict(results,1, 257,plot_insample=False, ax = ax) #On fait une predic
plt.title("Prediction de l'indice des prix pour les 6 prochains mois")
plt.show()
```



Estimation du prix

L'étude des chiffres de l'IPH dans le temps est un bon moyen pour les acheteurs, les vendeurs et les investisseurs de juger si la valeur des maisons dans différentes parties du pays est en hausse ou en baisse. Cela peut aider les investisseurs à déterminer la probabilité que leur investissement dans une maison individuelle soit rentabilisé par un bénéfice solide. Elle peut aider les vendeurs à calculer s'ils doivent mettre leur maison en vente à un prix plus ou moins élevé. Et il peut aider les acheteurs à déterminer si les prix sont en hausse dans les régions où ils aimeraient le plus vivre.

Ainsi, les prédictions montrent que l'indice va problement évoluer entre -1% et 5% avec une confiance de 95%. La prediction pure prévoit une augmentation de l'indice des prix d'environ 2% **au cours des 6 mois suivants.** Cela veut qu'une maison qui coûterait 100 000€ au mois m = 0, pourrait, avec un intervalle de confiance de 95%, se trouver dans la fourchette de prix [99 000;105 000]€. On peut donc être assez sur que le prix des propriétés vont augmenter.