# souvenir-sales-time-series-analysis

May 4, 2023

## 1 Souvenir Sales - Time Series Analysis

The following is a statistical analysis on a monthly time series which collects data about the sales of a souvenir shop in Australia in the period between 1987 and 1992.

The analysis will roughly follow the Box-Jenkins method and will focus on reaching stationarity for the time series, estimating a SARIMA model and predicting future values.

### 1.1 Data exploration

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller, kpss
from scipy.stats import boxcox
from scipy.special import inv_boxcox
import pmdarima as pm
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
tseries = pd.read_csv("data/monthly_sales_queensland.csv", header = 0,␣
↪parse_dates = ["date"], index_col = 0)
```

Now that we have imported the time series, let's have a first look at its values.

[3]:
```
tseries
```

[3]:
```
               sales
date
1987-01-01   1664.81
1987-02-01   2397.53
1987-03-01   2840.71
1987-04-01   3547.29
1987-05-01   3752.96
…              …
1992-08-01  19888.61
1992-09-01  23933.38
```

```
1992-10-01  25391.35
1992-11-01  36024.80
1992-12-01  80721.71

[72 rows x 1 columns]
```

[4]:
```
tseries.index.min(), tseries.index.max()
```

[4]:
```
(Timestamp('1987-01-01 00:00:00'), Timestamp('1992-12-01 00:00:00'))
```

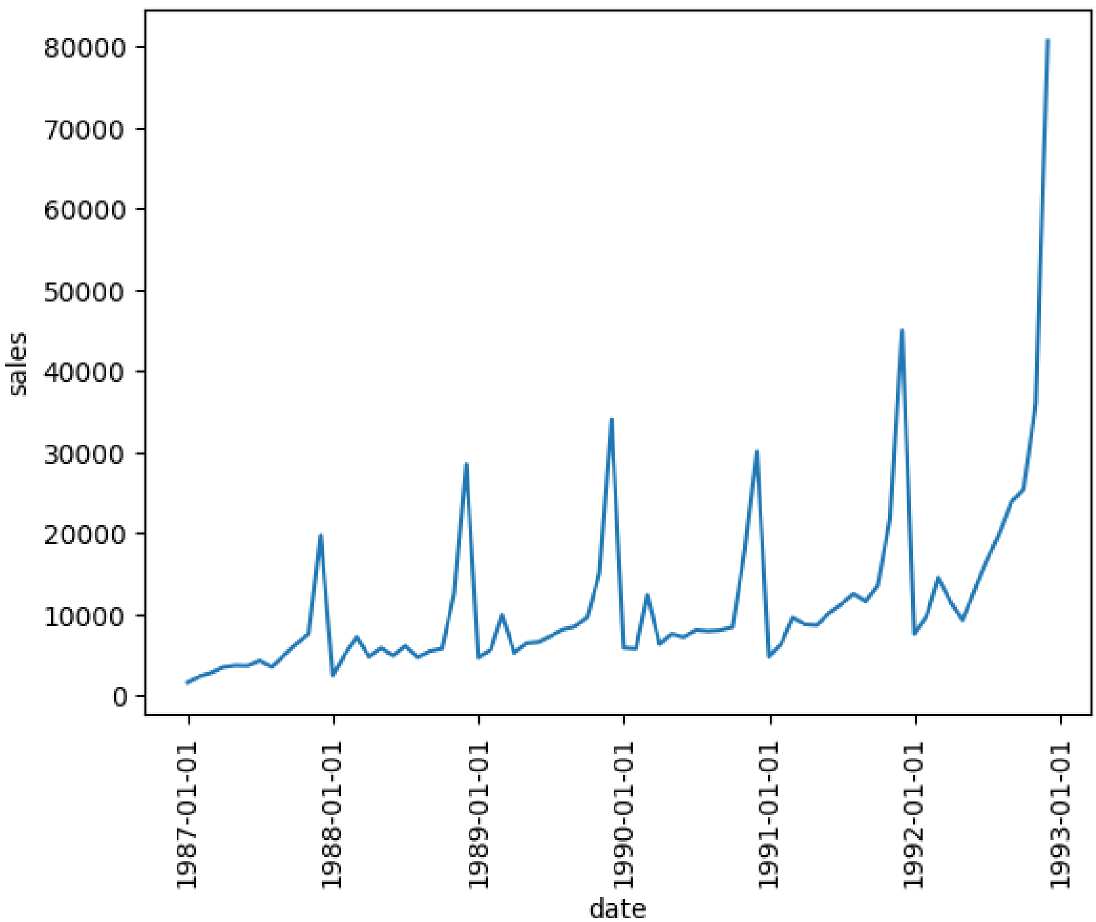
[5]:
```
tseries.index.max() - tseries.index.min()
```

[5]:
```
Timedelta('2161 days 00:00:00')
```

### 1.2 Check for non-stationarity

Let's make a plot and see what we can say about it.

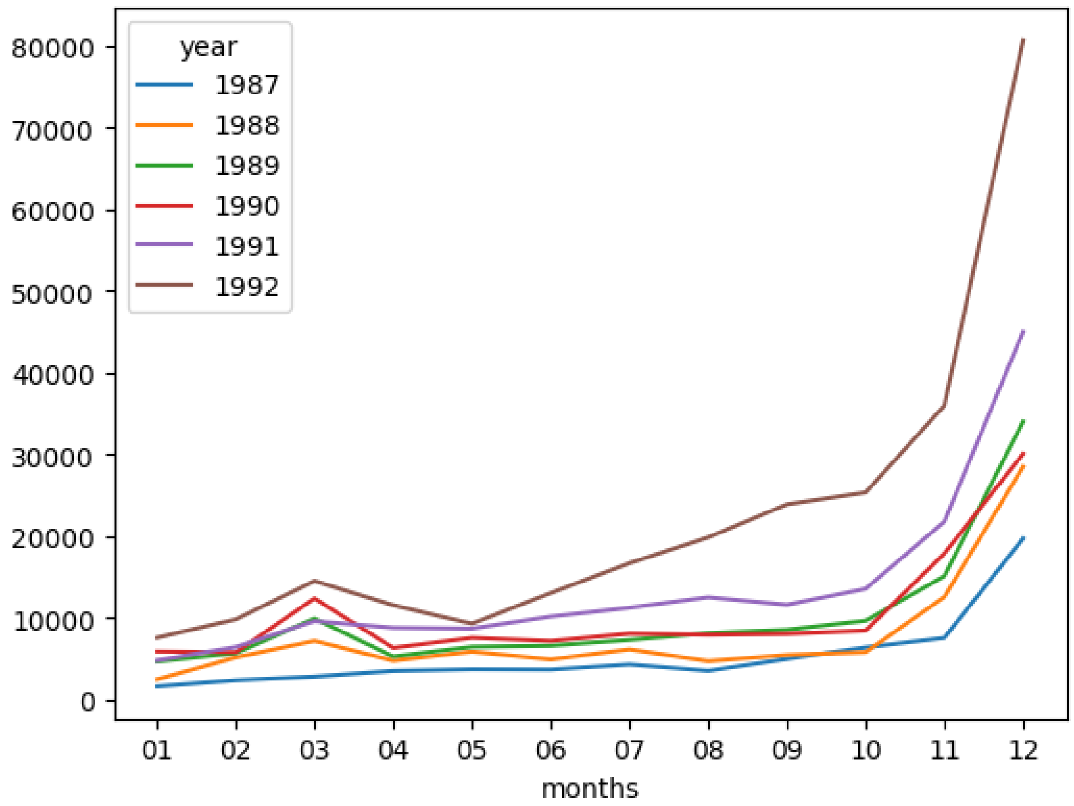
[6]:
```
fig, ax = plt.subplots()
ax.plot(tseries["sales"])
ax.set_xlabel("date")
ax.set_ylabel("sales")
ax.set_xticks(ax.get_xticks()[1::1])
plt.xticks(rotation = 90)
plt.show()
```




From the plot of the series, we can already grasp that it is not stationary, as the expected value is not constant over time and neither is variance, which tends to increase. In particular, we can speculate the presence of an upward trend and a seasonal effect, which is comprehensible, considering the turistical vocation of the shop.

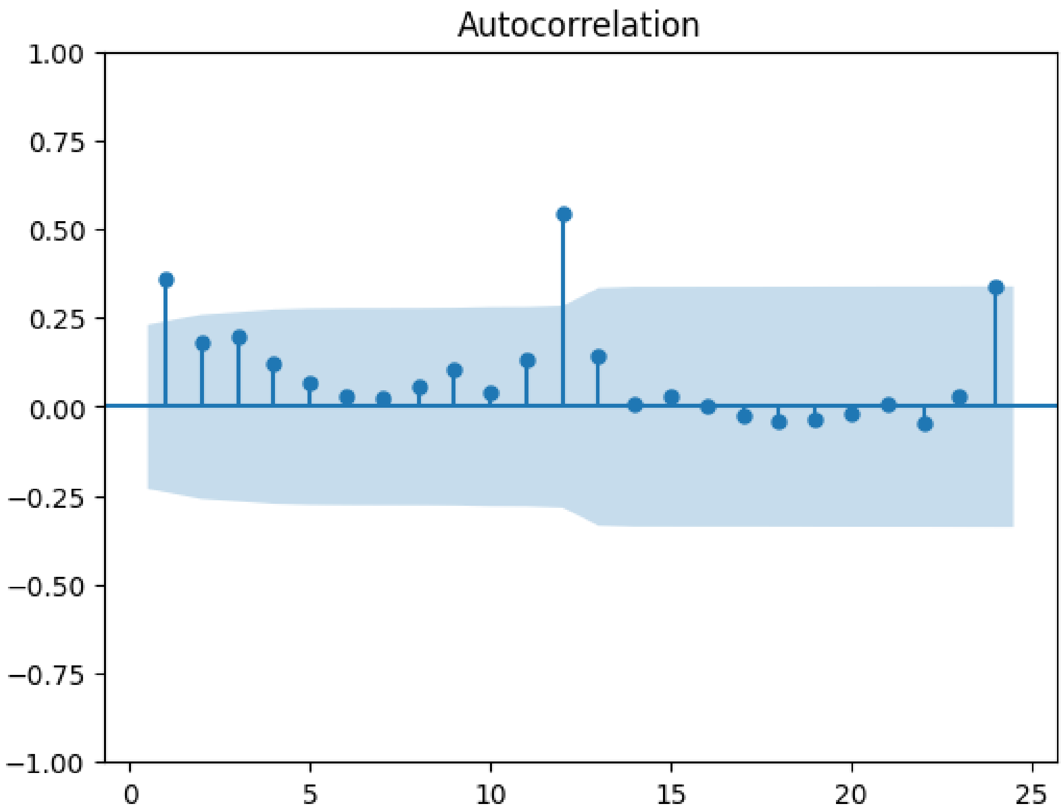
[7]:
```
df_years = tseries.copy(deep = True)
df_years.reset_index(inplace = True)
df_years["year"] = pd.to_datetime(df_years["date"]).dt.year
df_years["date"] = pd.to_datetime(df_years["date"]).dt.strftime("%m")
unstacked = df_years.set_index(["year", "date"])["sales"].unstack(-2)
unstacked.plot(xlabel = "months", xticks = pd.Series(range(0,12)))
```

[7]:
```
<Axes: xlabel='months'>
```

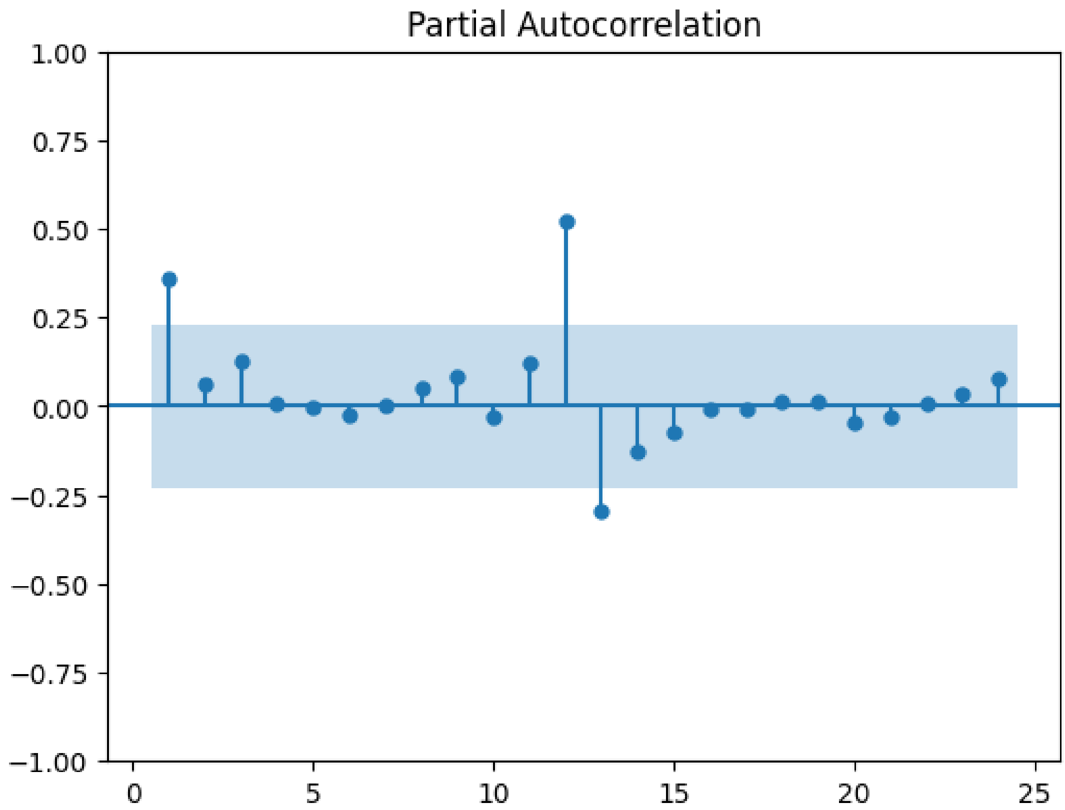



To better appreciate it, this plot shows the data for each year separately. The values of the series are clearly higher for the latest years and there is a recurring peak in the months of March and August, followed by a valley in October.

[8]:
```
plot_acf(tseries["sales"], lags = 24, zero = False);
```




[9]:
```
plot_pacf(tseries["sales"], lags = 24, zero = False, method = "ywm");
```




Lastly, these are the *global* and *partial autocorrelation functions* for the series. The slow decay for the ACF suggests, once again, the existence of a trend, while the spikes at lag 12 indicate a probable seasonality.

To formalize our guesses, let's resort to two statistical test: - The **Augmented Dickey-Fuller test** tests the null hypothesis of the presence of a unit root in our time series - The **KPSS test** tests the null hypothesis that our data is stationary

[10]:
```
adfuller(tseries["sales"])
```

[10]:
```
(2.4588875832659216,
 0.9990353814133549,
 12,
 59,
 {'1%': -3.5463945337644063,
  '5%': -2.911939409384601,
  '10%': -2.5936515282964665},
 1152.601255771056)
```

[11]:
```
kpss(tseries["sales"])
```

[11]: (0.9887885592096786,
 0.01,
 3,
 {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739})

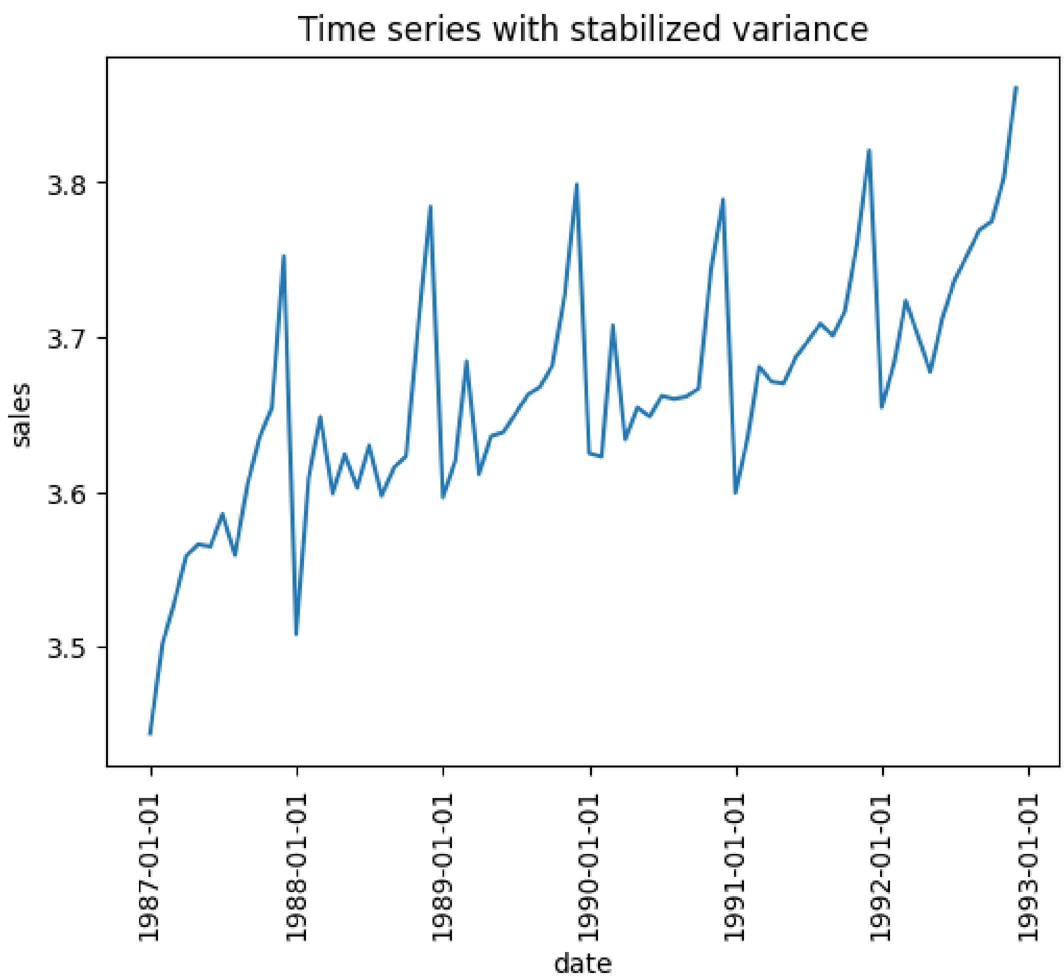
We were expecting to reject H0 for **KPSS** and not be able to reject it for **ADF** and that's exactly what happened, looking at the p-values.

### 1.3 Reach stationarity

In order to obtain stationarity in our time series, we need to perform a series of operations: we are going to stabilize the variance through the *Box-Cox transformation* and then apply differencing to treat trend and seasonality.

[12]:
```
tseries_novar = tseries.copy(deep = True)
bc = boxcox(tseries["sales"])[0]
lmbda = boxcox(tseries["sales"])[1]
tseries_novar["sales"] = bc
```

[13]:
```
fig, ax = plt.subplots()
ax.plot(tseries_novar["sales"])
ax.set_title("Time series with stabilized variance")
ax.set_xlabel("date")
ax.set_ylabel("sales")
ax.set_xticks(ax.get_xticks()[1::1])
plt.xticks(rotation = 90)
plt.show()
```




[14]:
```
tseries_diff = tseries_novar.copy(deep = True)
tseries_diff["sales"] = tseries_diff["sales"].diff(periods = 1)
tseries_diff = tseries_diff.iloc[1:]
tseries_diff
```

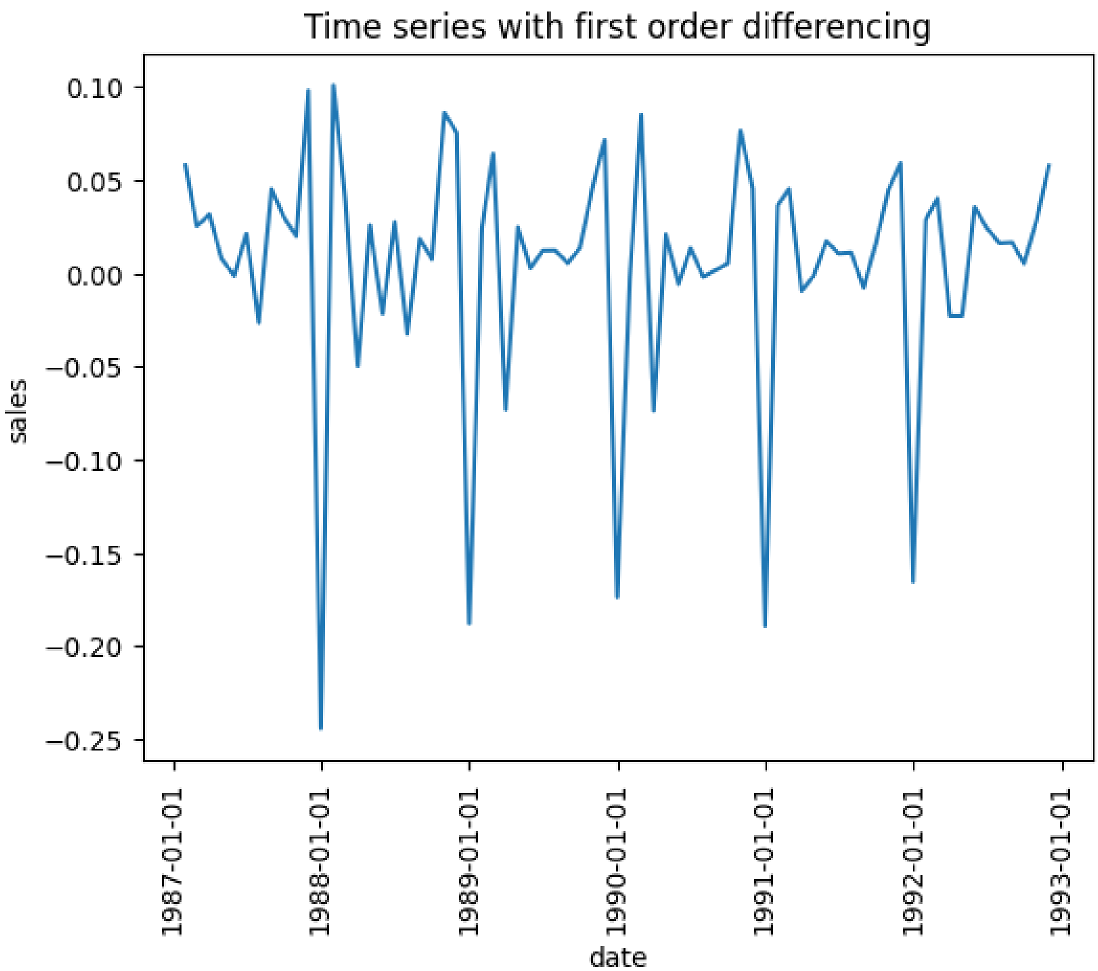
[14]:

| | sales |
|---|---|
| date | |
| 1987-02-01 | 0.057891 |
| 1987-03-01 | 0.025229 |
| 1987-04-01 | 0.031510 |
| 1987-05-01 | 0.007729 |
| 1987-06-01 | -0.001396 |
| … | … |
| 1992-08-01 | 0.016046 |
| 1992-09-01 | 0.016463 |
| 1992-10-01 | 0.005105 |

```
1992-11-01  0.028748
1992-12-01  0.057713

[71 rows x 1 columns]
```

[15]:
```
fig, ax = plt.subplots()
ax.plot(tseries_diff["sales"])
ax.set_title("Time series with first order differencing")
ax.set_xlabel("date")
ax.set_ylabel("sales")
ax.set_xticks(ax.get_xticks()[1::1])
plt.xticks(rotation = 90)
plt.show()
```



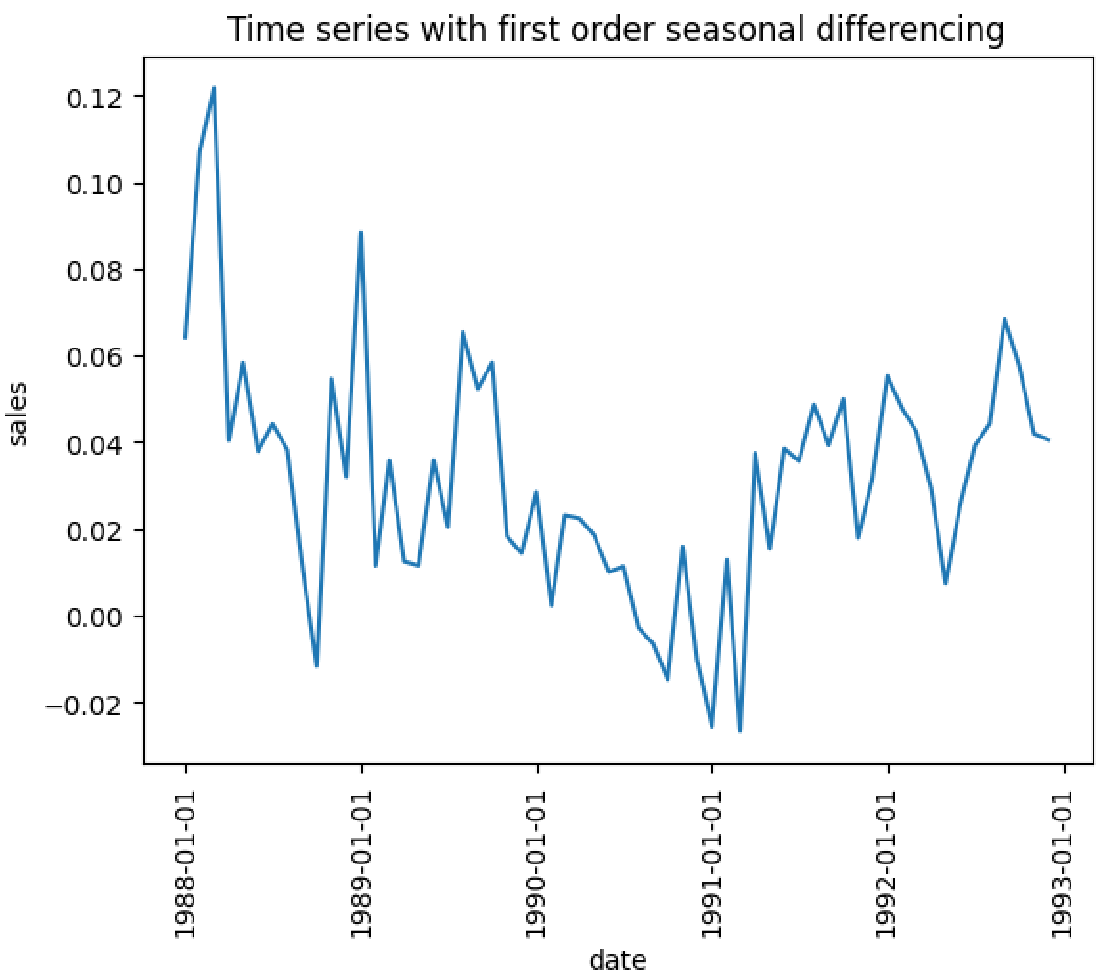

The first order differencing removes trend, but leaves seasonality.

[16]:
```python
tseries_diff_s = tseries_novar.copy(deep = True)
tseries_diff_s["sales"] = tseries_diff_s["sales"].diff(periods = 12)
tseries_diff_s = tseries_diff_s.iloc[12:]
tseries_diff_s
```

[16]:
```
               sales
date
1988-01-01  0.064202
1988-02-01  0.107133
1988-03-01  0.121736
1988-04-01  0.040426
1988-05-01  0.058380
1988-06-01  0.037900
1988-07-01  0.044232
1988-08-01  0.038239
1988-09-01  0.011337
1988-10-01 -0.011622
1988-11-01  0.054617
1988-12-01  0.032061
1989-01-01  0.088451
1989-02-01  0.011526
1989-03-01  0.035867
1989-04-01  0.012516
1989-05-01  0.011527
1989-06-01  0.035912
1989-07-01  0.020505
1989-08-01  0.065412
1989-09-01  0.052333
1989-10-01  0.058421
1989-11-01  0.018319
1989-12-01  0.014419
1990-01-01  0.028504
1990-02-01  0.002387
1990-03-01  0.023147
1990-04-01  0.022415
1990-05-01  0.018570
1990-06-01  0.010074
1990-07-01  0.011416
1990-08-01 -0.002764
1990-09-01 -0.006471
1990-10-01 -0.014722
1990-11-01  0.015944
1990-12-01 -0.009979
1991-01-01 -0.025552
1991-02-01  0.012920
1991-03-01 -0.026660
1991-04-01  0.037513
```

```
1991-05-01  0.015415
1991-06-01  0.038552
1991-07-01  0.035647
1991-08-01  0.048681
1991-09-01  0.039313
1991-10-01  0.049992
1991-11-01  0.017970
1991-12-01  0.031579
1992-01-01  0.055320
1992-02-01  0.047692
1992-03-01  0.042533
1992-04-01  0.029227
1992-05-01  0.007447
1992-06-01  0.025719
1992-07-01  0.039269
1992-08-01  0.044277
1992-09-01  0.068504
1992-10-01  0.057723
1992-11-01  0.041973
1992-12-01  0.040557
```

[17]:
```
fig, ax = plt.subplots()
ax.plot(tseries_diff_s["sales"])
ax.set_title("Time series with first order seasonal differencing")
ax.set_xlabel("date")
ax.set_ylabel("sales")
ax.set_xticks(ax.get_xticks()[1::1])
plt.xticks(rotation = 90)
plt.show()
```




The seasonal differencing removes seasonality, but leaves trend.

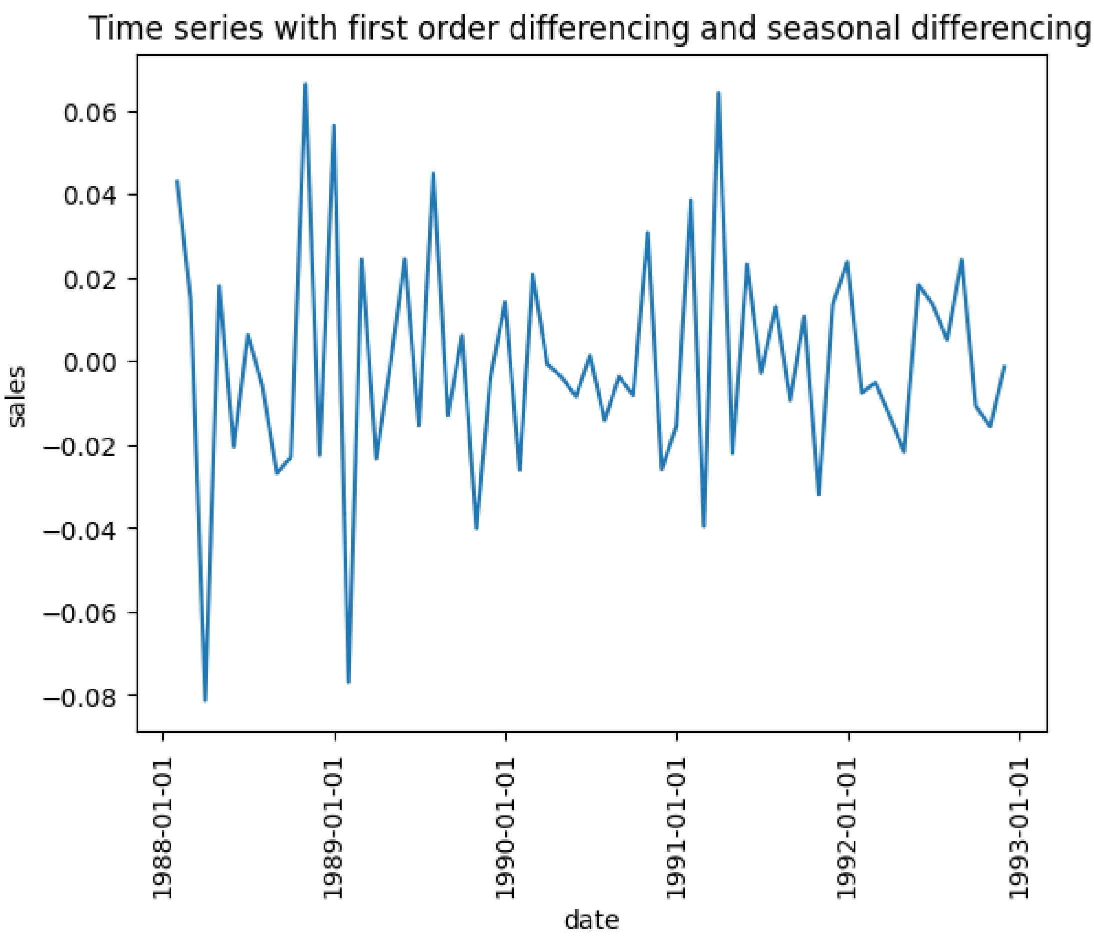
[18]:
```
tseries_diff_final = tseries_novar.copy(deep = True)
tseries_diff_final["sales"] = tseries_diff_final["sales"].diff(periods = 1).
 ↪diff(periods = 12)
tseries_diff_final = tseries_diff_final.iloc[13:]
tseries_diff_final
```

[18]:
```
               sales
date
1988-02-01  0.042931
1988-03-01  0.014603
1988-04-01 -0.081310
1988-05-01  0.017954
1988-06-01 -0.020480
1988-07-01  0.006332
1988-08-01 -0.005994
```

```
1988-09-01 -0.026901
1988-10-01 -0.022960
1988-11-01  0.066239
1988-12-01 -0.022556
1989-01-01  0.056390
1989-02-01 -0.076924
1989-03-01  0.024341
1989-04-01 -0.023351
1989-05-01 -0.000989
1989-06-01  0.024385
1989-07-01 -0.015407
1989-08-01  0.044907
1989-09-01 -0.013079
1989-10-01  0.006088
1989-11-01 -0.040102
1989-12-01 -0.003900
1990-01-01  0.014085
1990-02-01 -0.026117
1990-03-01  0.020760
1990-04-01 -0.000732
1990-05-01 -0.003845
1990-06-01 -0.008495
1990-07-01  0.001341
1990-08-01 -0.014180
1990-09-01 -0.003707
1990-10-01 -0.008252
1990-11-01  0.030666
1990-12-01 -0.025923
1991-01-01 -0.015573
1991-02-01  0.038472
1991-03-01 -0.039579
1991-04-01  0.064173
1991-05-01 -0.022099
1991-06-01  0.023137
1991-07-01 -0.002905
1991-08-01  0.013035
1991-09-01 -0.009368
1991-10-01  0.010679
1991-11-01 -0.032023
1991-12-01  0.013609
1992-01-01  0.023742
1992-02-01 -0.007628
1992-03-01 -0.005159
1992-04-01 -0.013307
1992-05-01 -0.021780
1992-06-01  0.018272
1992-07-01  0.013551
```

```
1992-08-01  0.005007
1992-09-01  0.024228
1992-10-01 -0.010782
1992-11-01 -0.015750
1992-12-01 -0.001416
```

[19]:
```
fig, ax = plt.subplots()
ax.plot(tseries_diff_final["sales"])
ax.set_title("Time series with first order differencing and seasonal␣
 ↪differencing")
ax.set_xlabel("date")
ax.set_ylabel("sales")
ax.set_xticks(ax.get_xticks()[1::1])
plt.xticks(rotation = 90)
plt.show()
```




The new series should be stationary now. Let's check with our two tests.

[20]:
```
adfuller(tseries_diff_final["sales"])
```

[20]:
```
(-5.903726178997968,
 2.73961678960973e-07,
 3,
 55,
 {'1%': -3.5552728880540942,
  '5%': -2.9157312396694217,
  '10%': -2.5956695041322315},
 -234.49533536833405)
```

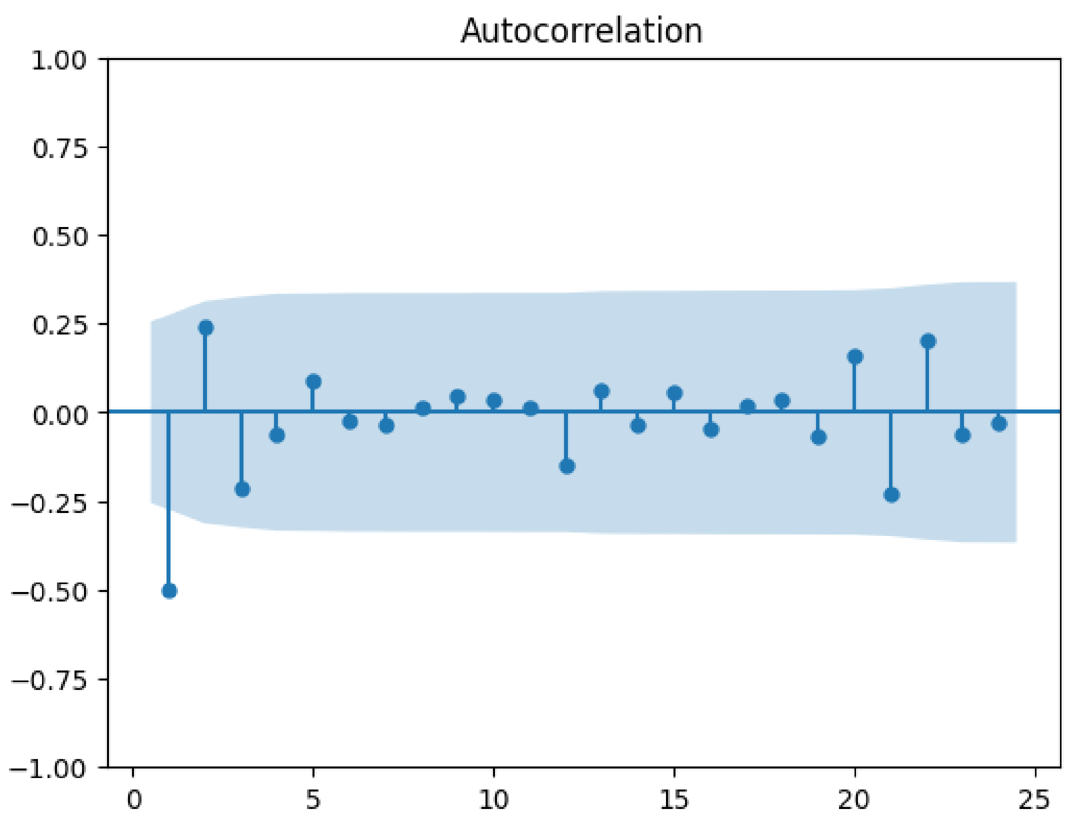
[21]:
```
kpss(tseries_diff_final["sales"])
```

[21]:
```
(0.02613487681632163,
 0.1,
 0,
 {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739})
```

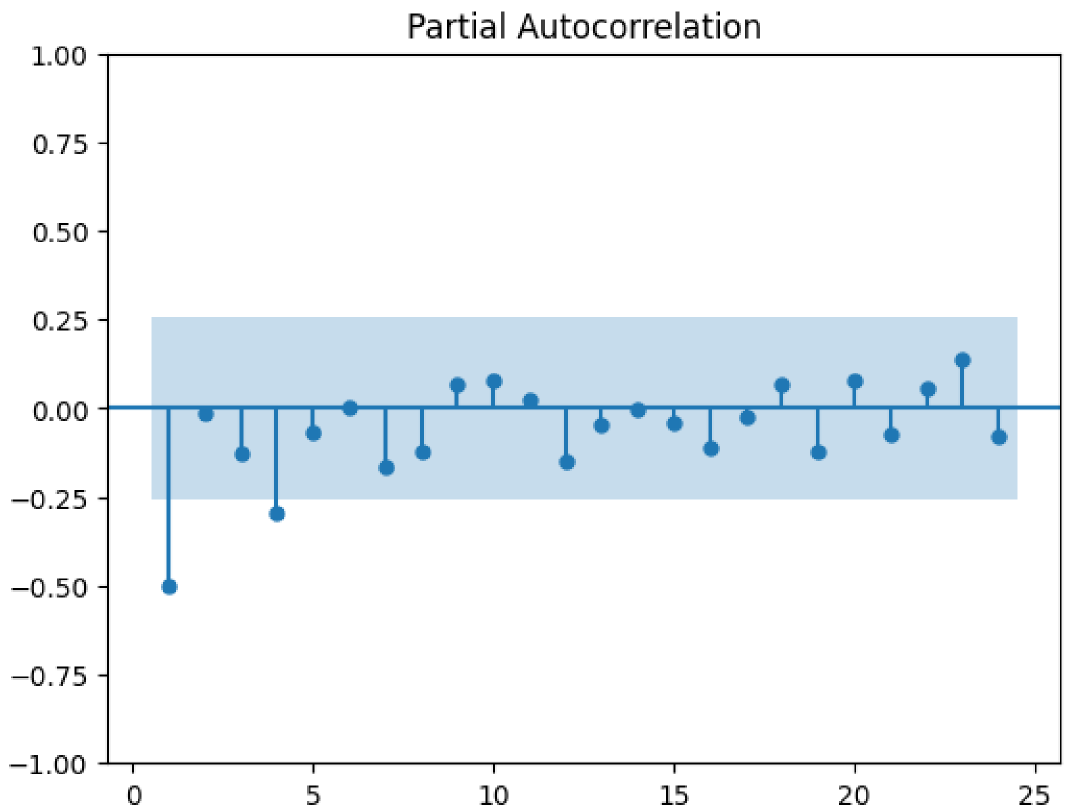
Our conclusions on the null hypothesis have now switched, as we expected.

Let's look at the ACF and PACF.

[22]:
```
plot_acf(tseries_diff_final["sales"], lags = 24, zero = False);
```



[23]:
```
plot_pacf(tseries_diff_final["sales"], lags = 24, zero = False, method = "ywm");
```




At this point, we should be able to estimate the values for the parameters of our ARIMA model by looking at these two plots and the spikes in them. However, the most reliable way to actually determine the parameters is using an objective procedure, for example a stepwise-like, and let a computer do it for us by choosing among many ARIMA models the "best" one, in terms of optimizing a certain indicator.

[24]:
```
fit = pm.auto_arima(tseries_novar, start_p = 1, start_q = 1, max_p = 3, max_q =␣
 ↪3, m = 12,
                             start_P = 0, seasonal = True, d = 1, D = 1, trace␣
 ↪= True,
                             error_action = "ignore",
                             suppress_warnings = True,
                             stepwise = True)
fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=-269.383, Time=1.02 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=-251.964, Time=0.04 sec
```

```
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=-269.841, Time=0.68 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=-269.740, Time=0.42 sec
 ARIMA(1,1,0)(0,1,0)[12]             : AIC=-267.544, Time=0.07 sec
 ARIMA(1,1,0)(2,1,0)[12]             : AIC=-268.583, Time=0.43 sec
 ARIMA(1,1,0)(1,1,1)[12]             : AIC=-269.956, Time=0.56 sec
 ARIMA(1,1,0)(0,1,1)[12]             : AIC=-271.244, Time=0.22 sec
 ARIMA(1,1,0)(0,1,2)[12]             : AIC=-269.821, Time=0.67 sec
 ARIMA(1,1,0)(1,1,2)[12]             : AIC=-267.700, Time=1.65 sec
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=-252.302, Time=0.38 sec
 ARIMA(2,1,0)(0,1,1)[12]             : AIC=-269.369, Time=0.81 sec
 ARIMA(2,1,1)(0,1,1)[12]             : AIC=-267.667, Time=1.18 sec
 ARIMA(1,1,0)(0,1,1)[12] intercept   : AIC=-269.622, Time=0.45 sec

Best model:  ARIMA(1,1,0)(0,1,1)[12]
Total fit time: 8.612 seconds
```

[24]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                      SARIMAX Results
================================================================================
============
Dep. Variable:                                     y   No. Observations:
72
Model:             SARIMAX(1, 1, 0)x(0, 1, [1], 12)   Log Likelihood
138.622
Date:                               Thu, 04 May 2023   AIC
-271.244
Time:                                       21:22:09   BIC
-265.011
Sample:                                   01-01-1987   HQIC
-268.811
                                        - 12-01-1992
Covariance Type:                                 opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.5521      0.124     -4.442      0.000      -0.796      -0.308
ma.S.L12      -0.4516      0.181     -2.493      0.013      -0.807      -0.097
sigma2         0.0005      0.000      4.347      0.000       0.000       0.001
================================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
0.02
Prob(Q):                              0.99   Prob(JB):
0.99
Heteroskedasticity (H):               0.30   Skew:
-0.04
```

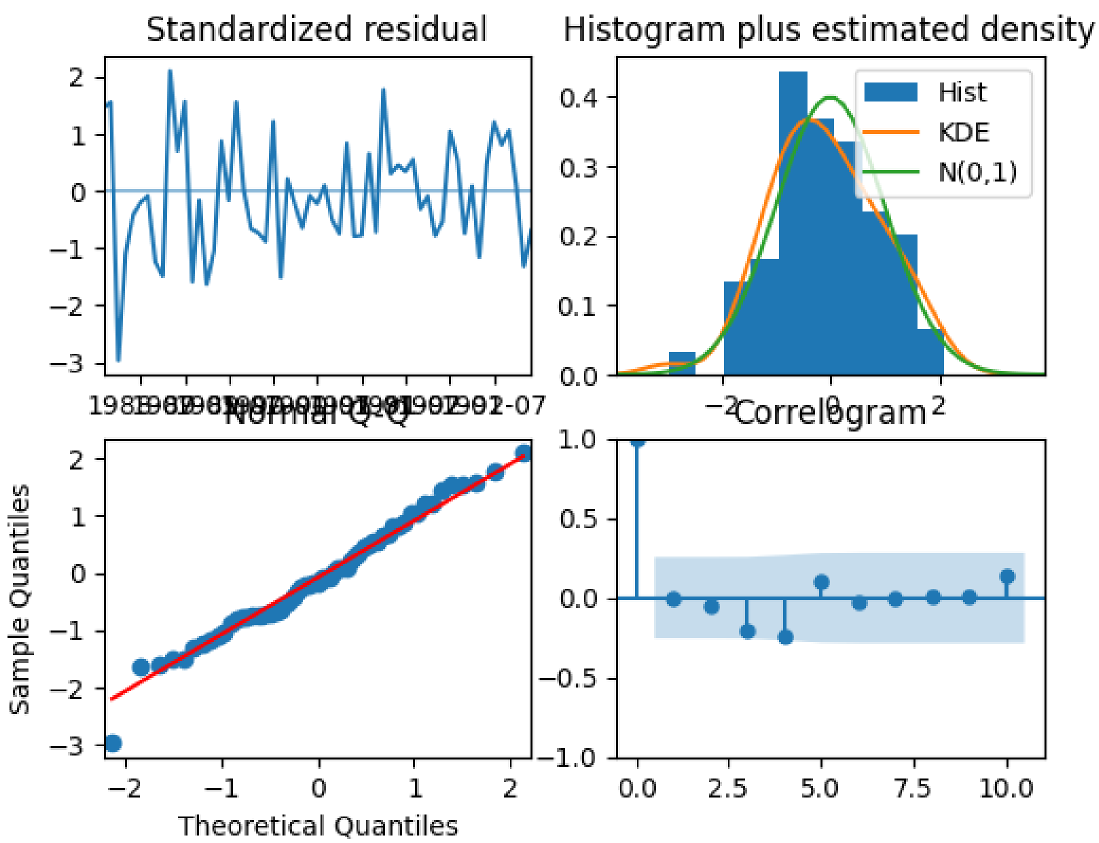
```
Prob(H) (two-sided):                  0.01   Kurtosis:
2.95
================================================================================
===

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

The chosen model appears to be *ARIMA(1,1,0)(0,1,1)[12]*. Along with it, we have also got a number of diagnostic tools: we can see that the parameters are significantly different than 0, while none of the residuals is significant.

Let's look at other diagnostic measures through some plots.

[25]:
```
fit.plot_diagnostics();
```




The residuals roughly follow a normal distribution, as deduced from the histogram and the Q-Q plot, but they seem to follow a pattern in their time series, which is not good for our model.

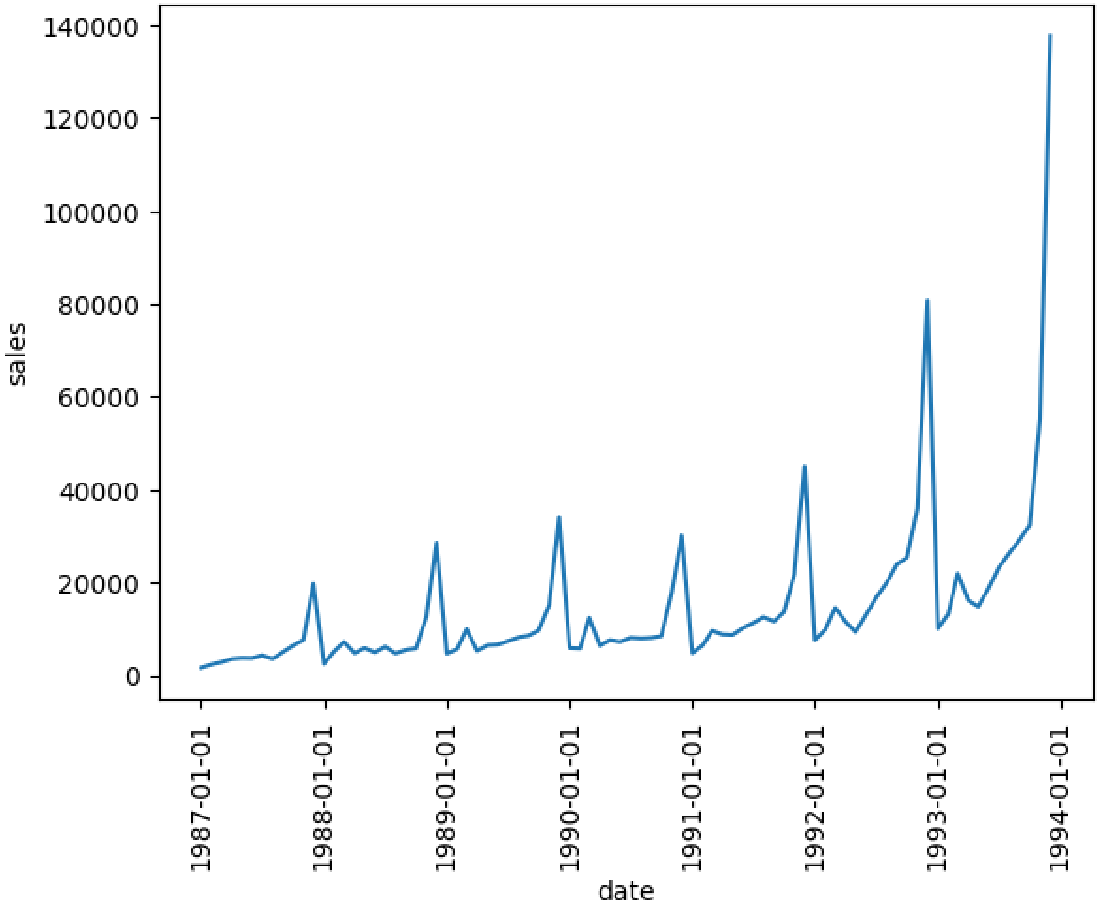
For the last step, let's try to forecast some future values, in particular 12 more observations, and plot the result.

[26]:
```
forecast = fit.predict(n_periods = 12)
df_forecast = pd.DataFrame({"sales": forecast.values})
df_forecast["sales"] = inv_boxcox(df_forecast["sales"], lmbda)
df_forecast.set_index(forecast.index, inplace = True)
df_forecast.index.name = "date"
tseries_forecast = pd.concat([tseries, df_forecast])
tseries_forecast
```

[26]:
```
                    sales
date
1987-01-01    1664.810000
1987-02-01    2397.530000
1987-03-01    2840.710000
1987-04-01    3547.290000
1987-05-01    3752.960000
…                     …
1993-08-01   26288.574310
1993-09-01   29161.143853
1993-10-01   32410.080979
1993-11-01   55069.265169
1993-12-01  137764.601875

[84 rows x 1 columns]
```

[27]:
```
fig, ax = plt.subplots()
ax.plot(tseries_forecast["sales"])
ax.set_xlabel("date")
ax.set_ylabel("sales")
ax.set_xticks(ax.get_xticks()[1::1])
plt.xticks(rotation = 90)
plt.show()
```


140000
120000
100000
80000
60000
40000
20000
0
sales
1987-01-01
1988-01-01
1989-01-01
1990-01-01
1991-01-01
1992-01-01
1993-01-01
1994-01-01
date