03_vector_autoregressive_model

September 29, 2021

1 How to use the VAR model for macro fundamentals forecasts

The vector autoregressive VAR(p) model extends the AR(p) model to k series by creating a system of k equations where each contains p lagged values of all k series. The coefficients on the own lags provide information about the dynamics of the series itself, whereas the cross-variable coefficients offer some insight into the interactions across the series.

1.1 Imports and Settings

[1]:
```
%matplotlib inline

import os
import sys
import warnings
from datetime import date
import pandas as pd
import pandas_datareader.data as web
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.transforms as mtransforms
import seaborn as sns

import statsmodels.api as sm
import statsmodels.tsa.api as smt
from statsmodels.tsa.api import VAR, VARMAX
from statsmodels.tsa.stattools import acf, q_stat, adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from scipy.stats import probplot, moment
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

[2]:
```
%matplotlib inline
warnings.filterwarnings('ignore')
sns.set(style='darkgrid', context='notebook', color_codes=True)
```

1.2 Helper Functions

1.2.1 Correlogram Plot

[3]:
```
def plot_correlogram(x, lags=None, title=None):
    lags = min(10, int(len(x)/5)) if lags is None else lags
    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 8))
    x.plot(ax=axes[0][0])
    q_p = np.max(q_stat(acf(x, nlags=lags), len(x))[1])
    stats = f'Q-Stat: {np.max(q_p):>8.2f}\nADF: {adfuller(x)[1]:>11.2f}'
    axes[0][0].text(x=.02, y=.85, s=stats, transform=axes[0][0].transAxes)
    probplot(x, plot=axes[0][1])
    mean, var, skew, kurtosis = moment(x, moment=[1, 2, 3, 4])
    s = f'Mean: {mean:>12.2f}\nSD: {np.sqrt(var):>16.2f}\nSkew: {skew:12.
↪2f}\nKurtosis:{kurtosis:9.2f}'
    axes[0][1].text(x=.02, y=.75, s=s, transform=axes[0][1].transAxes)
    plot_acf(x=x, lags=lags, zero=False, ax=axes[1][0])
    plot_pacf(x, lags=lags, zero=False, ax=axes[1][1])
    axes[1][0].set_xlabel('Lag')
    axes[1][1].set_xlabel('Lag')
    fig.suptitle(title, fontsize=20)
    fig.tight_layout()
    fig.subplots_adjust(top=.9)
```

1.2.2 Unit Root Test

[4]:
```
def test_unit_root(df):
    return df.apply(lambda x: f'{pd.Series(adfuller(x)).iloc[1]:.2%}').
↪to_frame('p-value')
```

1.3 Load Data

We will extend the univariate example of a single time series of monthly data on industrial production and add a monthly time series on consumer sentiment, both provided by the Federal Reserve's data service. We will use the familiar pandas-datareader library to retrieve data from 1970 through 2017:

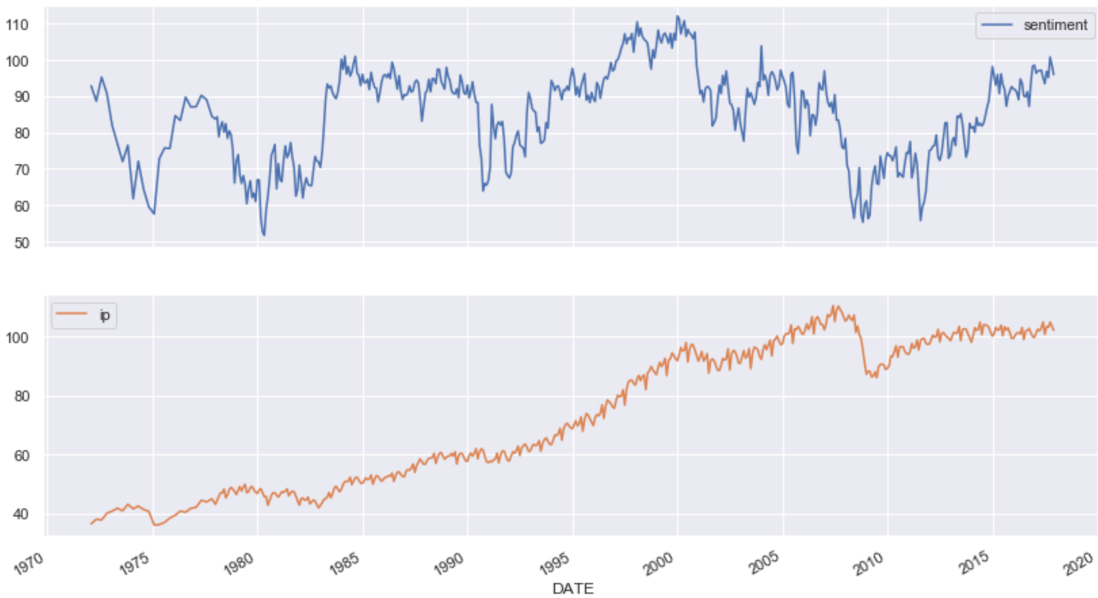
[5]:
```
sent = 'UMCSENT'
df = web.DataReader(['UMCSENT', 'IPGMFN'], 'fred', '1970', '2017-12').dropna()
df.columns = ['sentiment', 'ip']
```

[6]:
```
df.info()
```

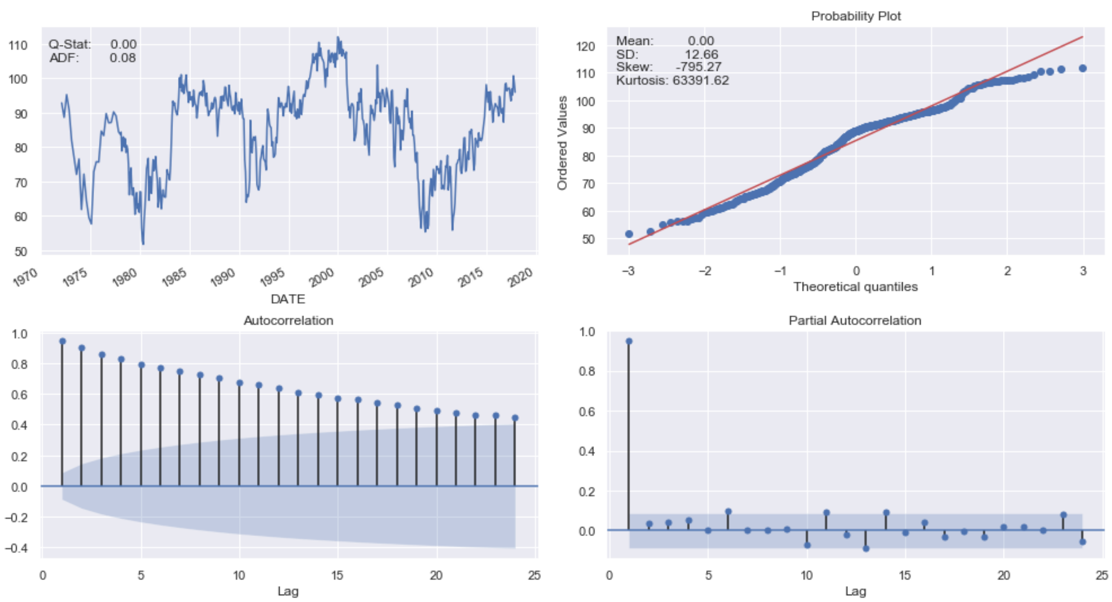
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 504 entries, 1972-02-01 to 2017-12-01
Data columns (total 2 columns):
sentiment    504 non-null float64
ip           504 non-null float64
```

```
dtypes: float64(2)
memory usage: 11.8 KB
```

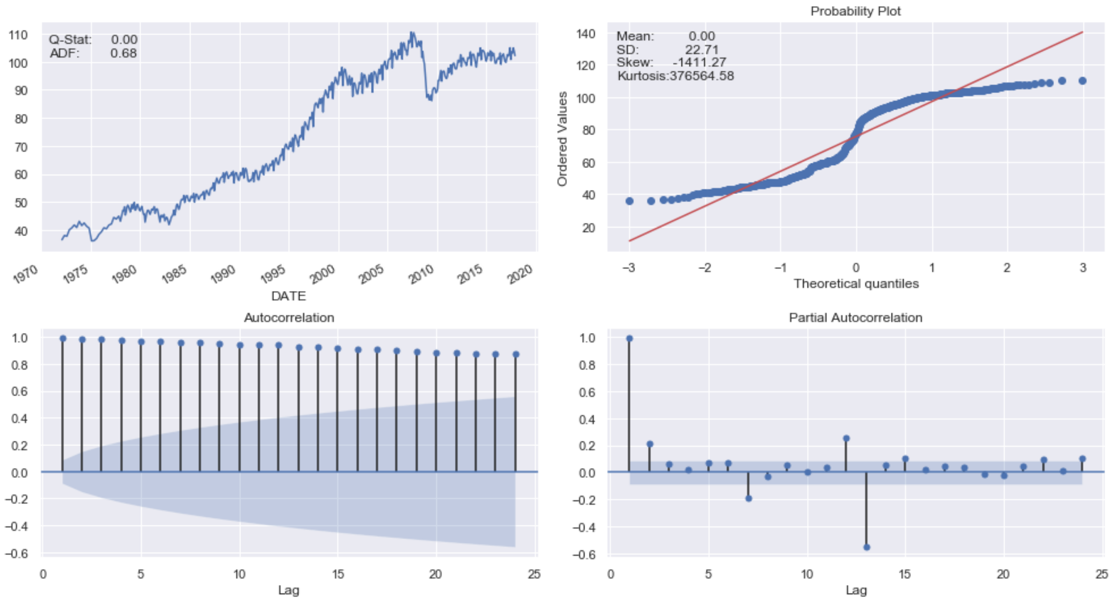
[7]:
```python
df.plot(subplots=True, figsize=(14,8));
```



[8]:
```python
plot_correlogram(df.sentiment, lags=24)
```



```
[9]: plot_correlogram(df.ip, lags=24)
```



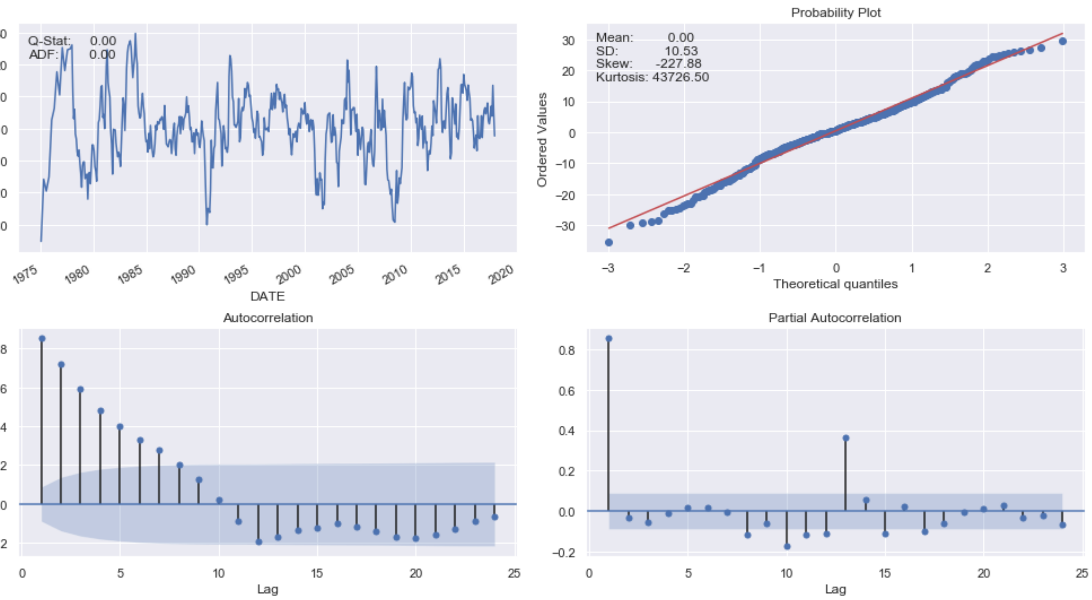
1.4 Stationarity Transform

Log-transforming the industrial production series and seasonal differencing using lag 12 of both series yields stationary results:

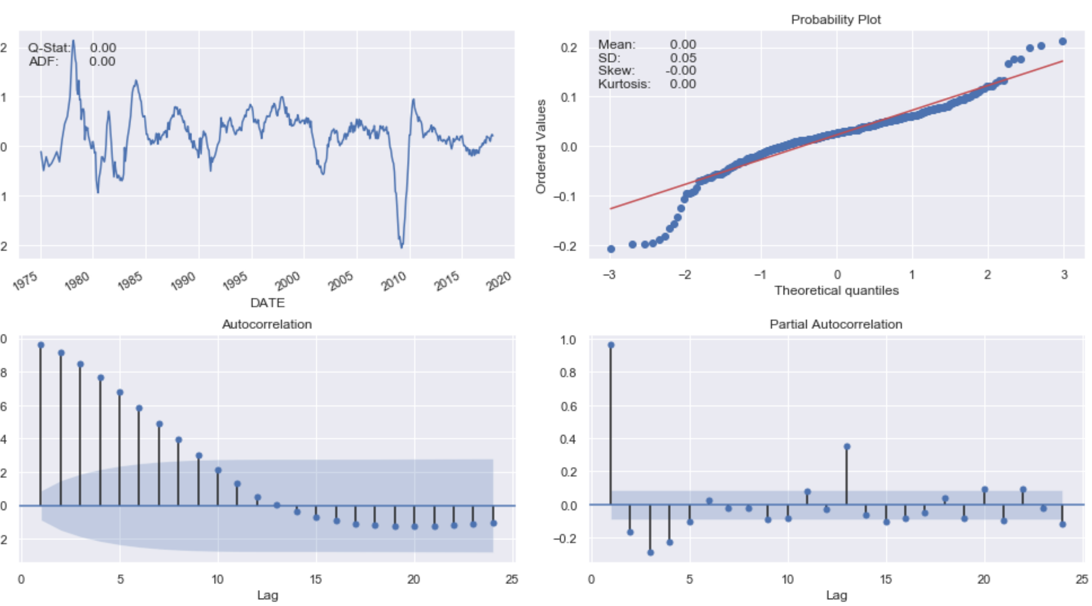
```
[10]: df_transformed = pd.DataFrame({'ip': np.log(df.ip).diff(12),
                                     'sentiment': df.sentiment.diff(12)}).dropna()
```

1.5 Inspect Correlograms

```
[11]: plot_correlogram(df_transformed.sentiment, lags=24)
```



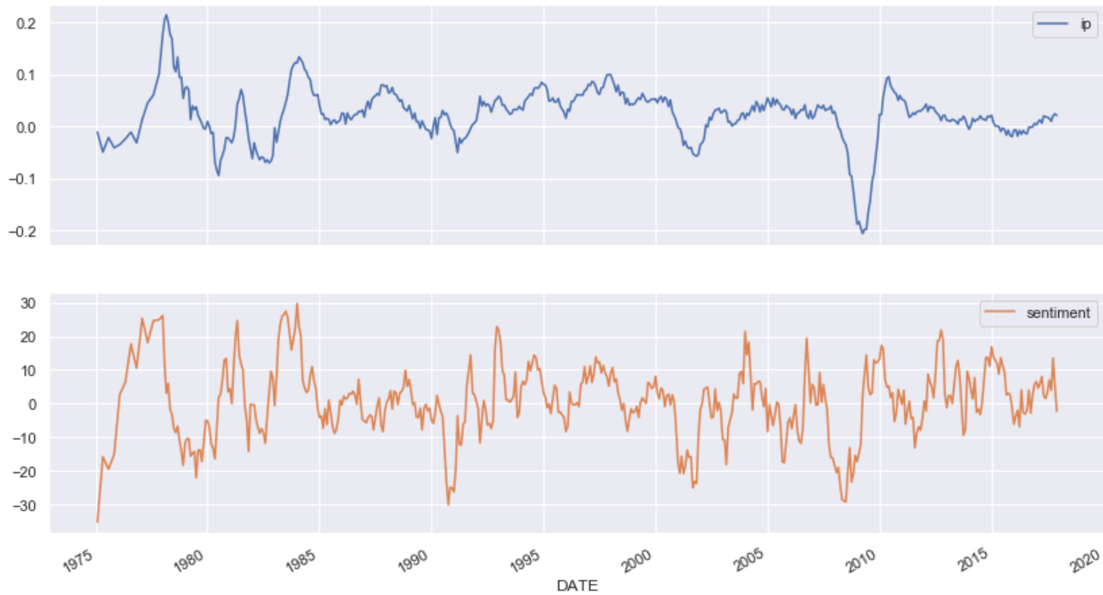

```
[12]: plot_correlogram(df_transformed.ip, lags=24)
```




```
[13]: test_unit_root(df_transformed)
```

```
[13]:            p-value
      ip           0.03%
      sentiment    0.00%
```

```
[14]: df_transformed.plot(subplots=True, figsize=(14,8));
```



1.6 VAR Model

To limit the size of the output, we will just estimate a VAR(1) model using the statsmodels VARMAX implementation (which allows for optional exogenous variables) with a constant trend using the first 480 observations. The output contains the coefficients for both time series equations.

```
[31]: model = VARMAX(df_transformed.iloc[:468], order=(1,1), trend='c').
       ↪fit(maxiter=1000)
      print(model.summary())
```

```
                           Statespace Model Results
===============================================================================
Dep. Variable:     ['ip', 'sentiment']   No. Observations:                  468
Model:                      VARMA(1,1)   Log Likelihood                 -71.824
                           + intercept   AIC                            169.647
Date:                 Tue, 05 Feb 2019   BIC                            223.578
Time:                         09:54:39   HQIC                           190.869
Sample:                              0
                                 - 468
Covariance Type:                   opg
================================================================================
===
Ljung-Box (Q):               128.08, 161.44   Jarque-Bera (JB):         130.30,
16.85
Prob(Q):                         0.00, 0.00   Prob(JB):                   0.00,
```

```
0.00
Heteroskedasticity (H):         0.48, 1.10   Skew:                         0.20,
0.21
Prob(H) (two-sided):            0.00, 0.55   Kurtosis:                     5.56,
3.83
                              Results for equation ip
================================================================================
===
                      coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
---
const               0.0015      0.001      2.418      0.016       0.000
0.003
L1.ip               0.9282      0.010     93.695      0.000       0.909
0.948
L1.sentiment        0.0006   6.02e-05     10.110      0.000       0.000
0.001
L1.e(ip)            0.0121      0.037      0.325      0.745      -0.061
0.085
L1.e(sentiment)    -0.0001      0.000     -0.867      0.386      -0.000
0.000
                           Results for equation sentiment
================================================================================
===
                      coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
---
const               0.3877      0.279      1.391      0.164      -0.159
0.934
L1.ip             -14.6492      5.444     -2.691      0.007     -25.320
-3.979
L1.sentiment        0.8805      0.023     37.684      0.000       0.835
0.926
L1.e(ip)           39.0203     18.828      2.072      0.038       2.119
75.922
L1.e(sentiment)     0.0507      0.052      0.979      0.327      -0.051
0.152
                                Error covariance matrix
================================================================================
=========
                            coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
---------
sqrt.var.ip               0.0129      0.000     40.347      0.000       0.012
0.014
```

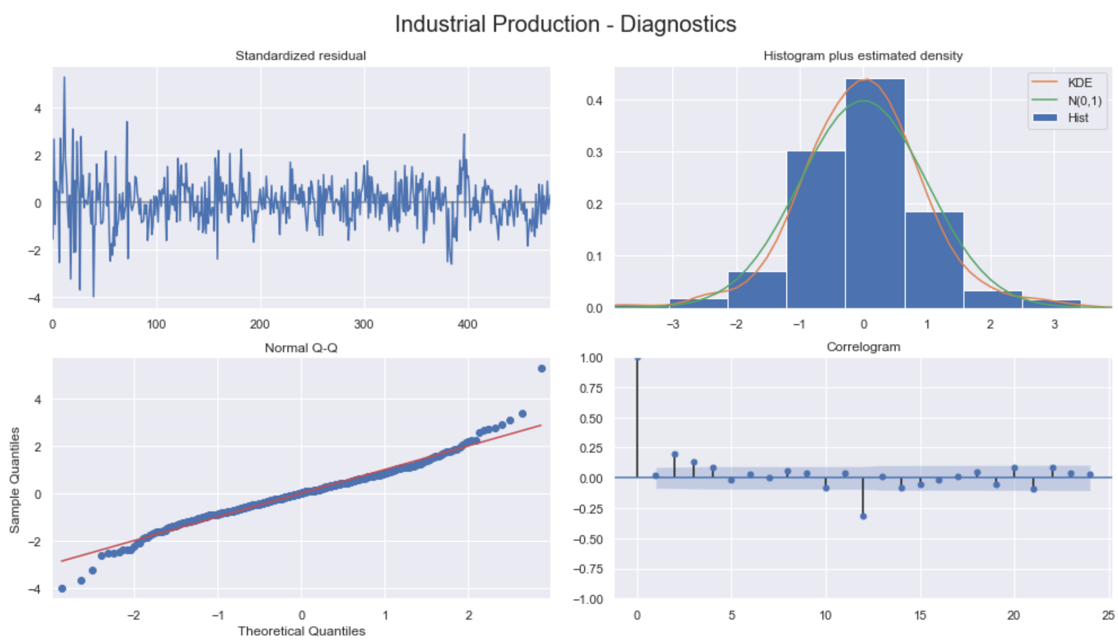
```
sqrt.cov.ip.sentiment     0.0436      0.232      0.188      0.851      -0.411
0.498
sqrt.var.sentiment        5.2755      0.149     35.506      0.000       4.984
5.567
================================================================================
=========

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

1.6.1 Plot Diagnostics

`statsmodels` provides diagnostic plots to check whether the residuals meet the white noise assumptions, which are not exactly met in this simple case:

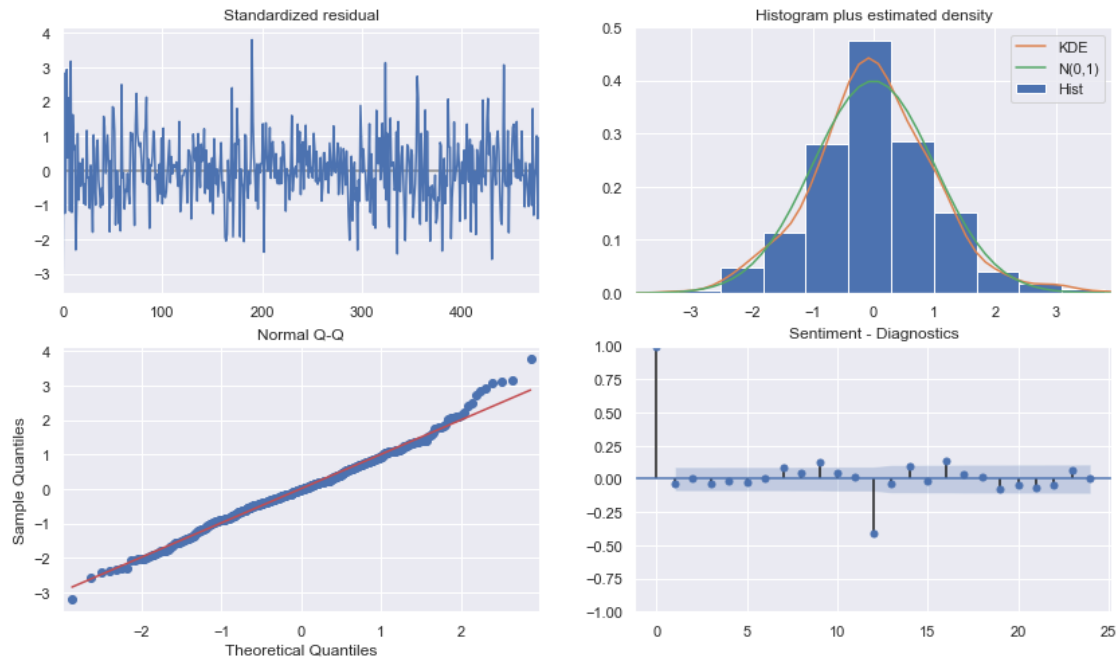
Industrial Production

[17]:
```
model.plot_diagnostics(variable=0, figsize=(14,8), lags=24)
plt.gcf().suptitle('Industrial Production - Diagnostics', fontsize=20)
plt.tight_layout()
plt.subplots_adjust(top=.9);
```



Sentiment

[18]:
```
model.plot_diagnostics(variable=1, figsize=(14,8), lags=24)
plt.title('Sentiment - Diagnostics');
```



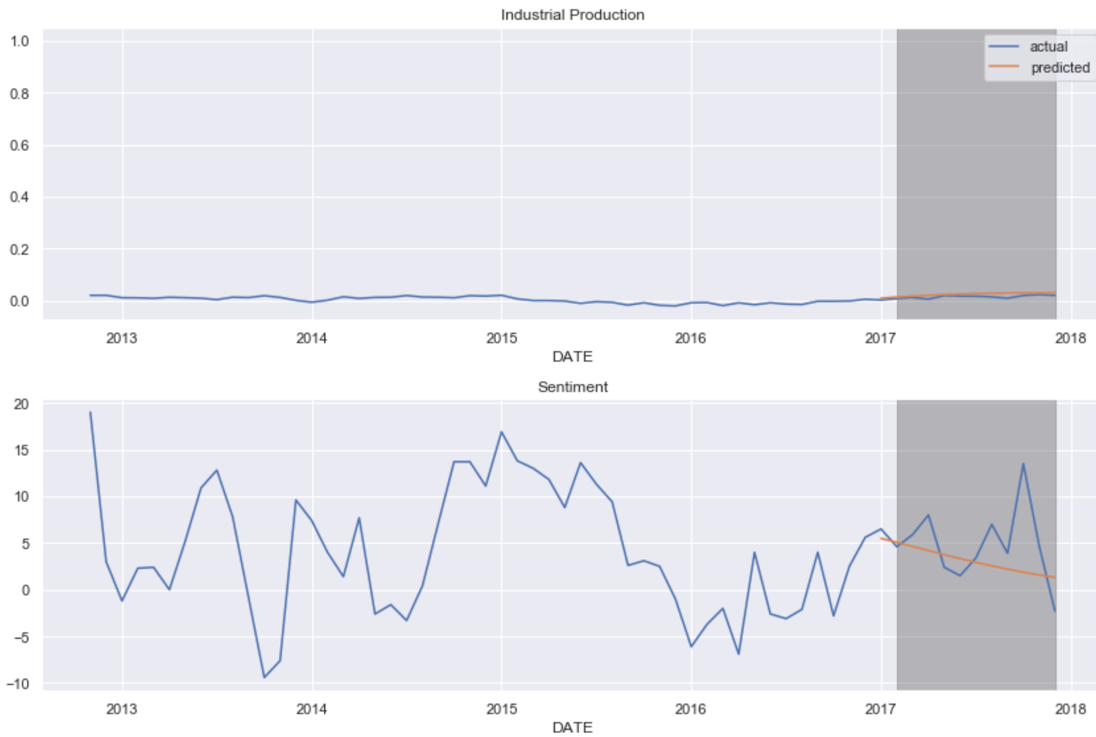
1.6.2 Impulse-Response Function

[19]:
```
median_change = df_transformed.diff().quantile(.5).tolist()
model.impulse_responses(steps=12, impulse=median_change).plot.
↪bar(subplots=True);
```

1.6.3 Generate Predictions

Out-of-sample predictions can be generated as follows:

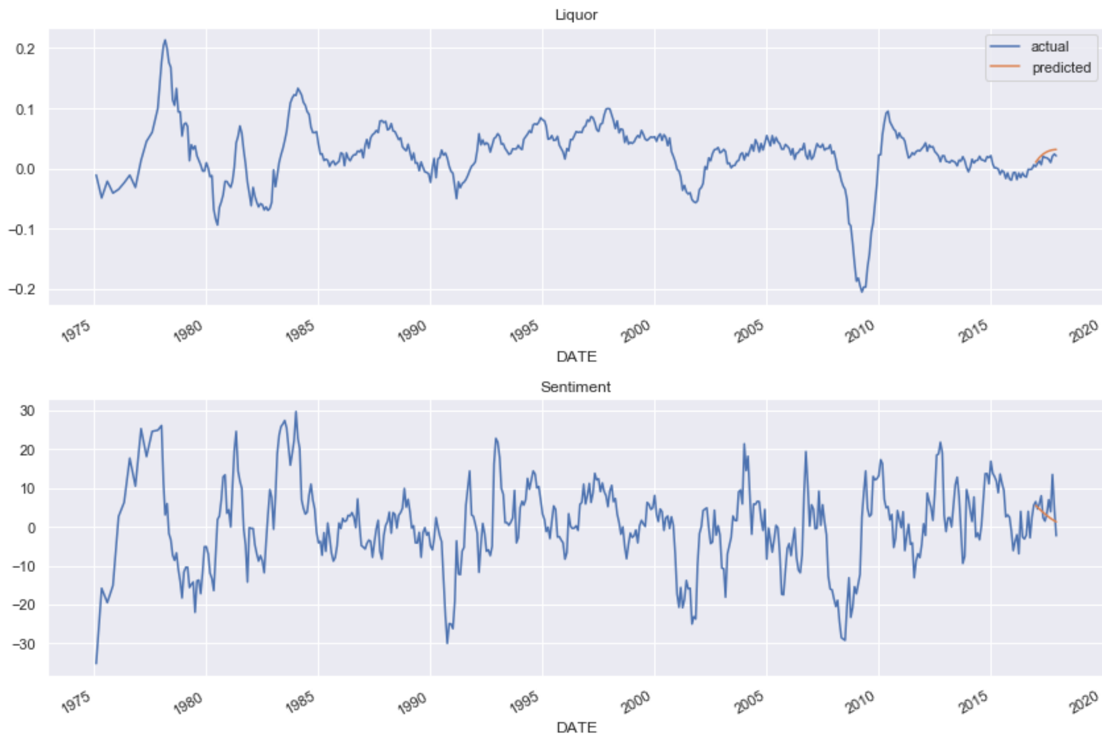
[20]:
```
start = 430
preds = model.predict(start=480, end=len(df_transformed)-1)
preds.index = df_transformed.index[480:]

fig, axes = plt.subplots(nrows=2, figsize=(12, 8))

df_transformed.ip.iloc[start:].plot(ax=axes[0], label='actual',␣
↪title='Industrial Production')
preds.ip.plot(label='predicted', ax=axes[0])
trans = mtransforms.blended_transform_factory(axes[0].transData, axes[0].
↪transAxes)
axes[0].legend()
axes[0].fill_between(x=df_transformed.index[481:], y1=0, y2=1, transform=trans,␣
↪color='grey', alpha=.5)

trans = mtransforms.blended_transform_factory(axes[0].transData, axes[1].
↪transAxes)
df_transformed.sentiment.iloc[start:].plot(ax=axes[1], label='actual',␣
↪title='Sentiment')
preds.sentiment.plot(label='predicted', ax=axes[1])
axes[1].fill_between(x=df_transformed.index[481:], y1=0, y2=1, transform=trans,␣
↪color='grey', alpha=.5)
fig.tight_layout();
```



1.6.4 Out-of-sample forecasts

A visualization of actual and predicted values shows how the prediction lags the actual values and does not capture non-linear out-of-sample patterns well:

[32]:
```
forecast = model.forecast(steps=24)

fig, axes = plt.subplots(nrows=2, figsize=(12, 8))

df_transformed.ip.plot(ax=axes[0], label='actual', title='Liquor')
preds.ip.plot(label='predicted', ax=axes[0])
axes[0].legend()

df_transformed.sentiment.plot(ax=axes[1], label='actual', title='Sentiment')
preds.sentiment.plot(label='predicted', ax=axes[1])
axes[1]
fig.tight_layout();
```



[35]:
```
mean_absolute_error(forecast, df_transformed.iloc[468:])
```

[35]: 1.9520062876193942

[ ]: