04_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]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
%matplotlib inline

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

from statsmodels.tsa.api import VARMAX
from statsmodels.tsa.stattools import acf, q_stat, adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.preprocessing import minmax_scale
from scipy.stats import probplot, moment
from sklearn.metrics import mean_absolute_error
```

[3]:
```
sns.set(style='whitegrid',
        context='notebook',
        color_codes=True)
```

1.2 Helper Functions

1.2.1 Correlogram Plot

[4]:
```
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], title='Time Series')
    x.rolling(21).mean().plot(ax=axes[0][0], c='k', lw=1)
    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=14)
    sns.despine()
    fig.tight_layout()
    fig.subplots_adjust(top=.9)
```

1.2.2 Unit Root Test

[5]:
```
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:

[6]:
```
sent = 'UMCSENT'
df = web.DataReader(['UMCSENT', 'IPGMFN'], 'fred', '1970', '2019-12').dropna()
df.columns = ['sentiment', 'ip']
```

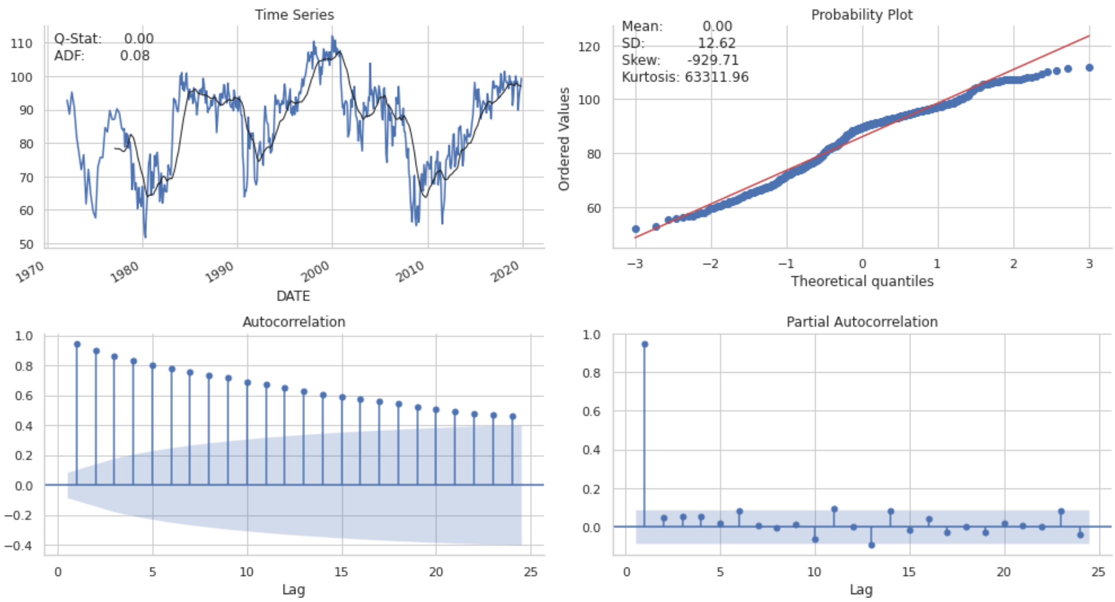
[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 528 entries, 1972-02-01 to 2019-12-01
Data columns (total 2 columns):
```

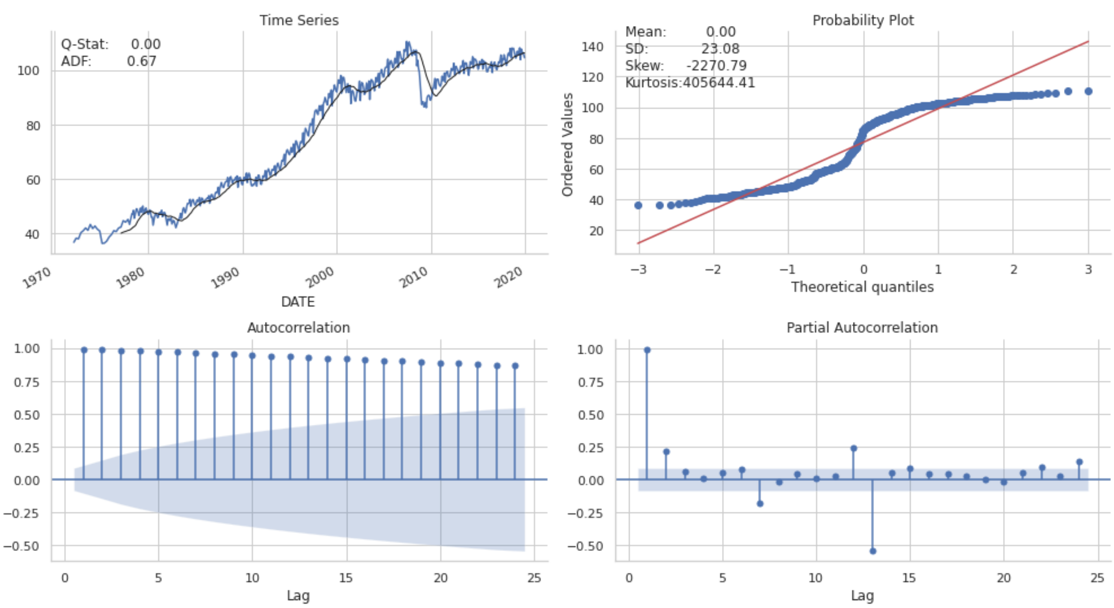
```
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   sentiment  528 non-null    float64
 1   ip         528 non-null    float64
dtypes: float64(2)
memory usage: 12.4 KB
```

[8]:
```
df.plot(subplots=True, figsize=(14,8), rot=0)
sns.despine()
plt.tight_layout();
```

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




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



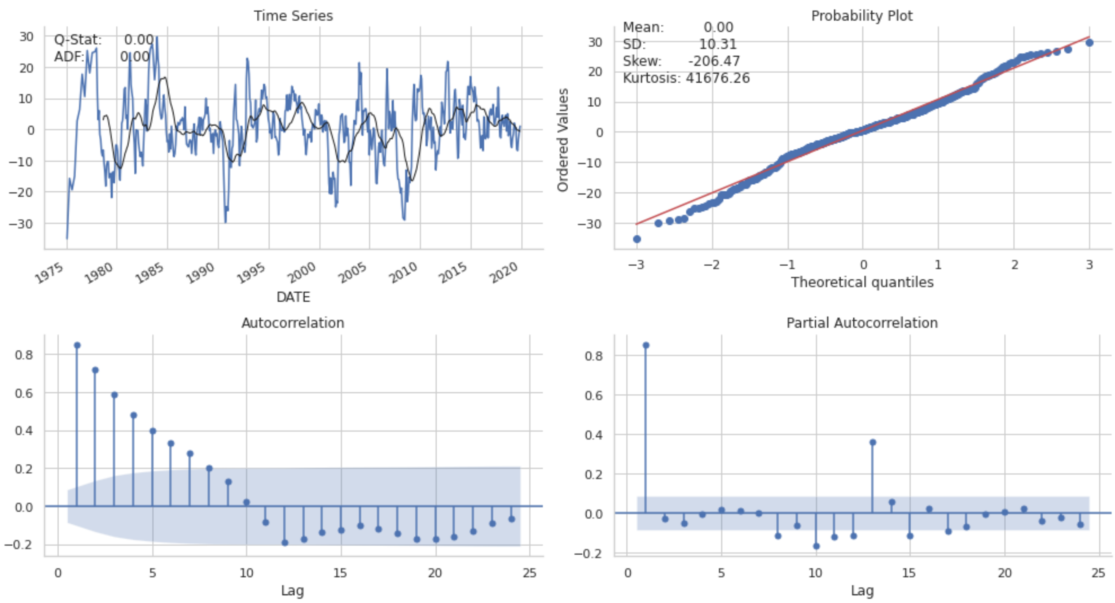
1.4 Stationarity Transform

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

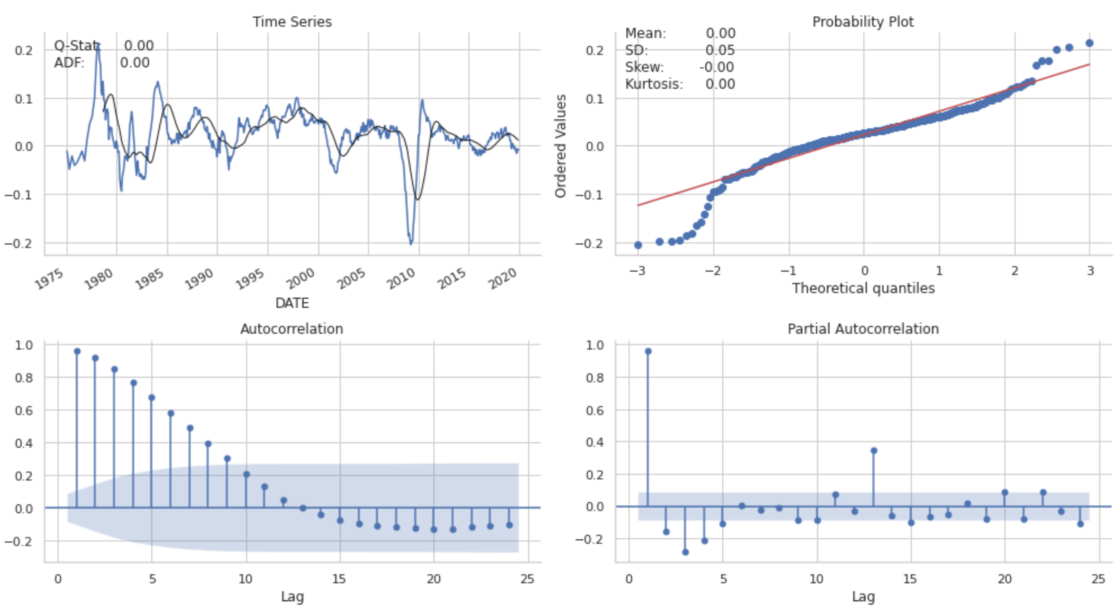
```
[11]: df_transformed = pd.DataFrame({'ip': np.log(df.ip).diff(12),
                                    'sentiment': df.sentiment.diff(12)}).dropna()
```

1.5 Inspect Correlograms

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



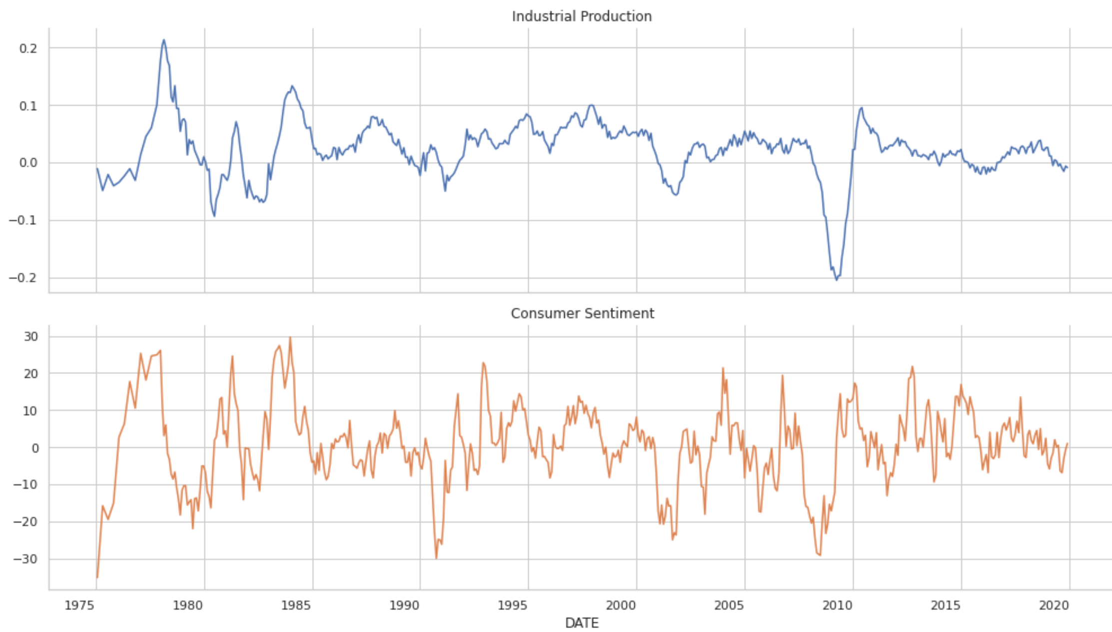

```
[13]: plot_correlogram(df_transformed.ip, lags=24)
```



[14]:
```
test_unit_root(df_transformed)
```

[14]:
```
          p-value
ip          0.02%
sentiment   0.00%
```

[15]:
```
df_transformed.plot(subplots=True, figsize=(14, 8),
                    title=['Industrial Production', 'Consumer Sentiment'],
                    legend=False,
                    rot=0)
sns.despine()
plt.tight_layout()
```



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 through 2017. The output contains the coefficients for both time series equations.

[16]:
```
df_transformed = df_transformed.apply(minmax_scale)
```

[17]:
```
model = VARMAX(df_transformed.loc[:'2017'], order=(1,1), trend='c').
 ↪fit(maxiter=1000)
```

```
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.8/site-
packages/statsmodels/tsa/statespace/varmax.py:161: EstimationWarning: Estimation
```

```
of VARMA(p,q) models is not generically robust, due especially to identification
issues.
  warn('Estimation of VARMA(p,q) models is not generically robust,'
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.8/site-
packages/statsmodels/tsa/base/tsa_model.py:581: ValueWarning: A date index has
been provided, but it has no associated frequency information and so will be
ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
```

[18]:
```
print(model.summary())
```

```
                           Statespace Model Results
==============================================================================
Dep. Variable:     ['ip', 'sentiment']   No. Observations:                  492
Model:                       VARMA(1,1)   Log Likelihood                1560.335
                            + intercept   AIC                          -3094.669
Date:                  Thu, 15 Apr 2021   BIC                          -3040.089
Time:                          15:41:56   HQIC                         -3073.237
Sample:                               0
                                  - 492
Covariance Type:                    opg
==============================================================================
===
Ljung-Box (L1) (Q):             0.15, 0.29   Jarque-Bera (JB):        151.67,
16.67
Prob(Q):                        0.70, 0.59   Prob(JB):                   0.00,
0.00
Heteroskedasticity (H):         0.46, 1.03   Skew:                       0.19,
0.21
Prob(H) (two-sided):            0.00, 0.86   Kurtosis:                   5.69,
3.80
                            Results for equation ip
==============================================================================
===
                      coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
---
intercept          -0.0115      0.006     -1.794      0.073      -0.024
0.001
L1.ip               0.9270      0.010     97.414      0.000       0.908
0.946
L1.sentiment        0.0940      0.009     10.478      0.000       0.076
0.112
L1.e(ip)            0.0065      0.036      0.182      0.856      -0.064
0.077
L1.e(sentiment)    -0.0157      0.018     -0.864      0.388      -0.051
0.020
```

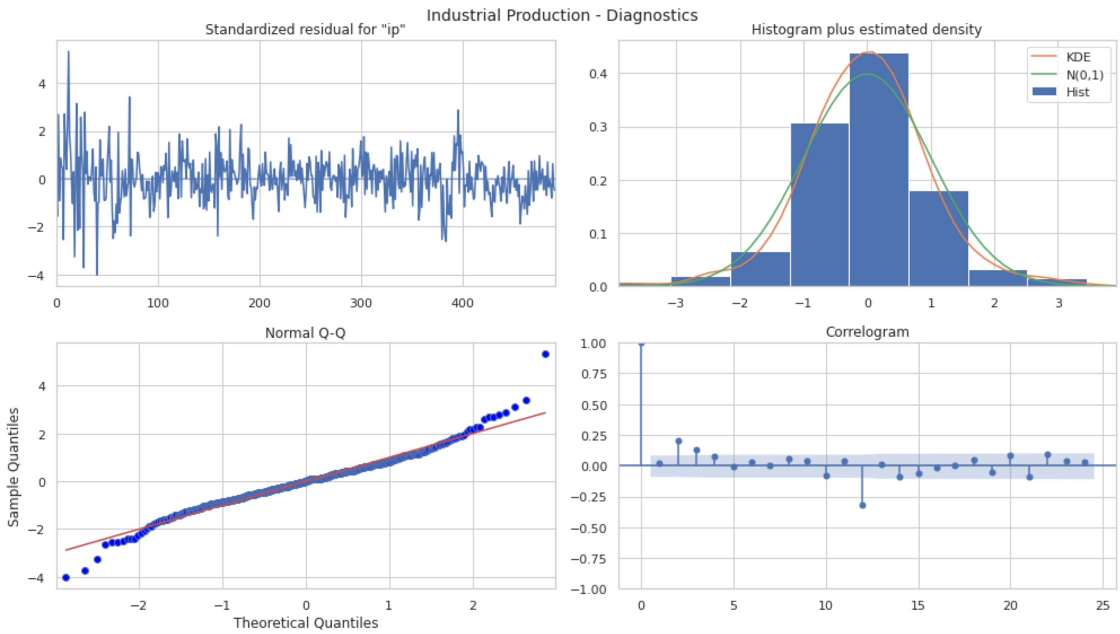
```
                           Results for equation sentiment
================================================================================
===
                      coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
---
intercept           0.1189      0.019      6.169      0.000       0.081
0.157
L1.ip              -0.0996      0.034     -2.955      0.003      -0.166
-0.034
L1.sentiment        0.8825      0.023     39.025      0.000       0.838
0.927
L1.e(ip)            0.2983      0.120      2.496      0.013       0.064
0.532
L1.e(sentiment)     0.0190      0.051      0.373      0.709      -0.081
0.119
                                 Error covariance matrix
================================================================================
=========
                            coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
---------
sqrt.var.ip                 0.0302      0.001     41.980      0.000       0.029
0.032
sqrt.cov.ip.sentiment       0.0004      0.003      0.111      0.912      -0.006
0.007
sqrt.var.sentiment          0.0809      0.002     36.292      0.000       0.076
0.085
================================================================================
=========

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:

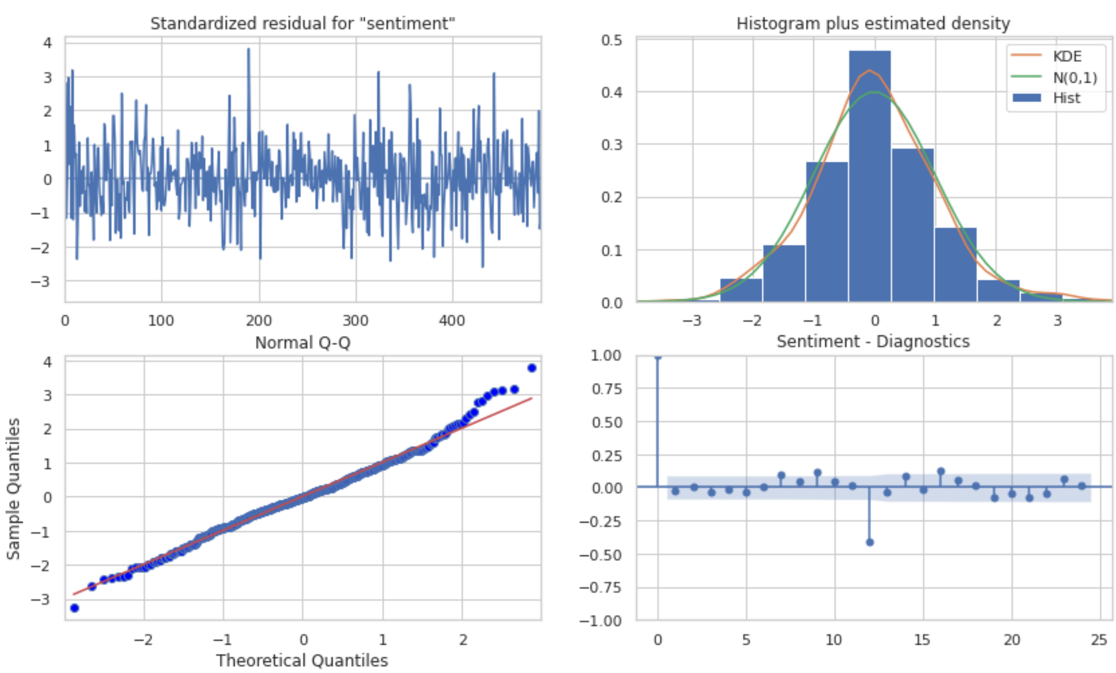
Industrial Production

[19]:
```
model.plot_diagnostics(variable=0, figsize=(14,8), lags=24)
plt.gcf().suptitle('Industrial Production - Diagnostics', fontsize=14)
plt.tight_layout()
plt.subplots_adjust(top=.93);
```



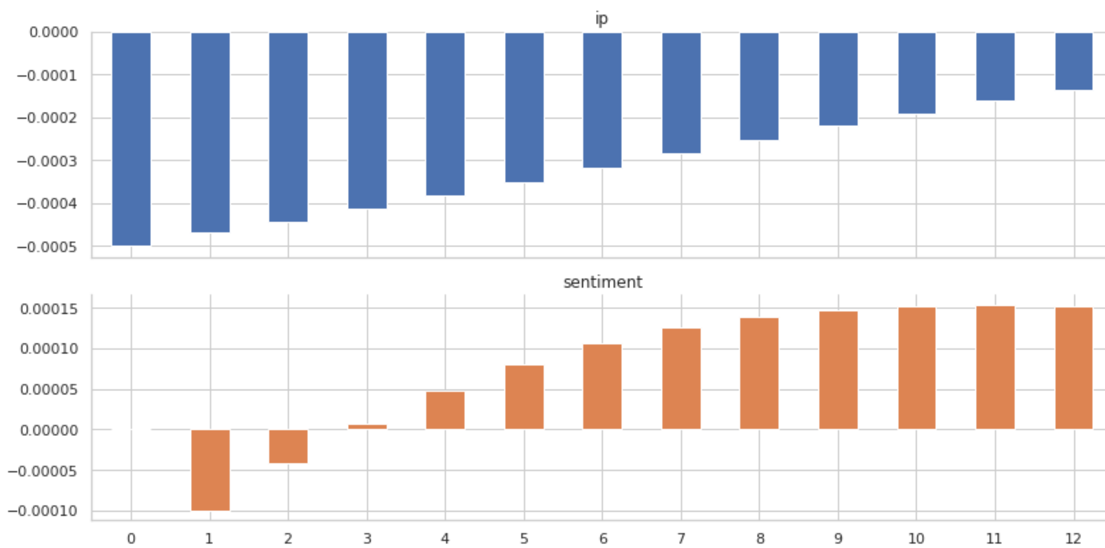
Sentiment

[20]:
```
model.plot_diagnostics(variable=1, figsize=(14,8), lags=24)
plt.title('Sentiment - Diagnostics');
```



1.6.2 Impulse-Response Function

[21]:
```
median_change = df_transformed.diff().quantile(.5).tolist()
model.impulse_responses(steps=12,
                        impulse=median_change).plot.bar(subplots=True,
                                                        figsize=(12, 6),
                                                        rot=0,
                                                        legend=False)
sns.despine()
plt.tight_layout()
```



1.6.3 Generate Predictions

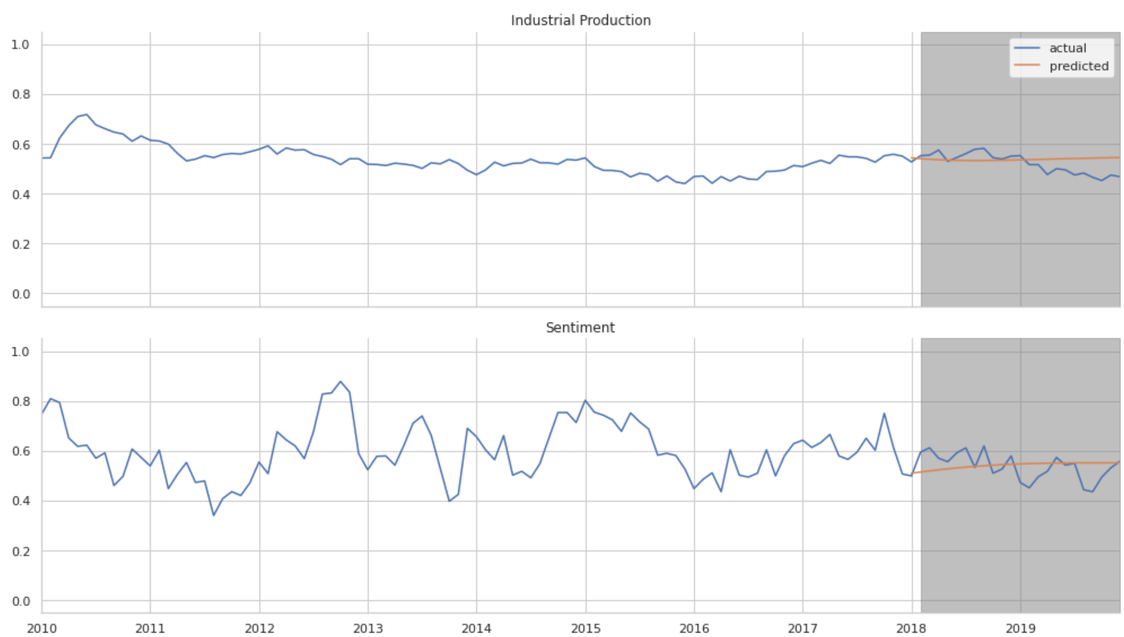
Out-of-sample predictions can be generated as follows:

[22]:
```
n =len(df_transformed)
start = n-24

preds = model.predict(start=start+1, end=n)
```

```
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.8/site-
packages/statsmodels/tsa/base/tsa_model.py:376: ValueWarning: No supported index
is available. Prediction results will be given with an integer index beginning
at `start`.
  warnings.warn('No supported index is available.'
```

[23]:
```
preds.index = df_transformed.index[start:]

fig, axes = plt.subplots(nrows=2, figsize=(14, 8), sharex=True)
```

```
df_transformed.ip.loc['2010':].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[start+1:], y1=0, y2=1,␣
 ↪transform=trans, color='grey', alpha=.5)

trans = mtransforms.blended_transform_factory(axes[0].transData, axes[1].
 ↪transAxes)
df_transformed.sentiment.loc['2010':].plot(ax=axes[1], label='actual',␣
 ↪title='Sentiment')
preds.sentiment.plot(label='predicted', ax=axes[1])
axes[1].fill_between(x=df_transformed.index[start+1:], y1=0, y2=1,␣
 ↪transform=trans, color='grey', alpha=.5)
axes[1].set_xlabel('')
sns.despine()
fig.tight_layout();
```



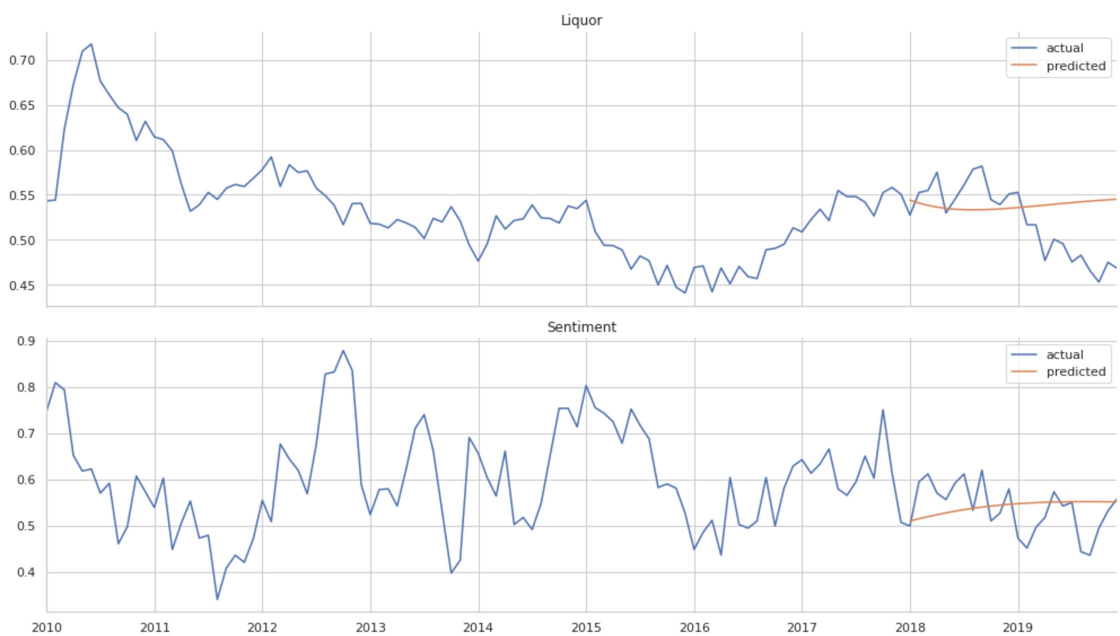
1.6.4 Out-of-sample forecasts

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:

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

```
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.8/site-
packages/statsmodels/tsa/base/tsa_model.py:376: ValueWarning: No supported index
is available. Prediction results will be given with an integer index beginning
at `start`.
  warnings.warn('No supported index is available.'
```

[25]:
```
fig, axes = plt.subplots(nrows=2, figsize=(14, 8), sharex=True)

df_transformed['2010':].ip.plot(ax=axes[0], label='actual', title='Liquor')
preds.ip.plot(label='predicted', ax=axes[0])
axes[0].legend()

df_transformed['2010':].sentiment.plot(ax=axes[1], label='actual',␣
 ↪title='Sentiment')
preds.sentiment.plot(label='predicted', ax=axes[1])
axes[1].legend()
axes[1].set_xlabel('')
sns.despine()
fig.tight_layout();
```



[26]:
```
mean_absolute_error(forecast, df_transformed.iloc[492:])
```

[26]: 0.042856543992030566