Understand ARIMA and tune P, D, Q

Python notebook using data from multiple data sources · 13,191 views · 1y ago

Introduction

ARIMA is one of the most classic time series forecasting models. During the modeling process, we mainly want to find 3 parameters. Auto-regression(AR) term, namly the lags of previous value; Integral(I) term for non-stationary differencing and Moving Average(MA) for error term.

I'm a newbie in this field. Found many online tutorials used grid search technique(auto.arima in R). Meanwhile I also found many hypothesis test to validate the time series, i.e. see if it's stationary, looking at ACF and PACF to suggest a AR term etc...

Facebook has a package called prophet, which is quite complex and consider many things automaticlly. But out of curiosity, I want to understand what's the reasoning behind the model. ARIMA is definitely a good starting point.

My goal for this notebook:

1. Understand ARIMA, SARIMA, ARIMAX
2. Walkthrough the necessary tests that ARIMA needs to statisfy
3. Find a set of reasonable parameters base on a statistic tests and visualizations

Notebook Outline:

- ARIMA introduction
- Decompose the ts
- Stationarize the data
- Interpret ACF and PACF
- Determine p, d, q
- Adding seasonality: S-ARMIA
- Adding holiday factors to be SARIMA-X

A few things on my TODO list:

- mulitple seasonality
- outlier detection

In [1]:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
%matplotlib inline
import matplotlib.pyplot as plt  # Matlab-style plotting
import seaborn as sns
import statsmodels.api as sm

color = sns.color_palette()
sns.set_style('darkgrid')
```

In [2]:

```
from subprocess import check_output
```

```
print(check_output(['ls', '../input']).decode('utf-8'))
```

```
demand-forecasting-kernels-only
holiday
```

In [3]:

```
train = pd.read_csv('../input/demand-forecasting-kernels-only/train.csv')
train['date'] = pd.to_datetime(train['date'], format="%Y-%m-%d")

train.head()
```

Out[3]:

	date	store	item	sales
0	2013-01-01	1	1	13
1	2013-01-02	1	1	11
2	2013-01-03	1	1	14
3	2013-01-04	1	1	13
4	2013-01-05	1	1	10

Forecast modeling - ARIMA

we want to start with some basic/classic model like armia. Here is a list of online tutorials that helps me get started:
http://www.statsmodels.org/dev/examples/notebooks/generated/tsa_arma_0.html
http://www.seanabu.com/2016/03/22/time-series-seasonal-ARIMA-model-in-python/
http://barnesanalytics.com/basics-of-arima-models-with-statsmodels-in-python

ARIMA model includes the AR term, the I term, and the MA term. Let's actually start with the I term, as it is the easiest to explain. The I term is a full difference. That is today's value minus yesterday's value. That's it.
The way that I like to think of the AR term is that it is a partial difference. The coefficient on the AR term will tell you the percent of a difference you need to take.
MA
A moving average term in a time series model is a past error (multiplied by a coefficient).The label "moving average" is is somewhat misleading because the weights $1,-\theta_1,-\theta_2,\ldots,-\theta_q$, which multiply the a's, need not total unity nor need that be positive.
$X_t=\varepsilon_t+\theta_1\varepsilon_{t-1}+\cdots+\theta_q\varepsilon_{t-q}$ as akin to a weighted moving average of the ε terms,

In [4]:

```
# per 1 store, 1 item
train_df = train[train['store']==1]
train_df = train_df[train['item']==1]
# train_df = train_df.set_index('date')
train_df['year'] = train['date'].dt.year
train_df['month'] = train['date'].dt.month
train_df['day'] = train['date'].dt.dayofyear
train_df['weekday'] = train['date'].dt.weekday

train_df.head()
```

Out[4]:

	date	store	item	sales	year	month	day	weekday
0	2013-01-01	1	1	13	2013	1	1	1
1	2013-01-02	1	1	11	2013	1	2	2
2	2013-01-03	1	1	14	2013	1	3	3
3	2013-01-04	1	1	13	2013	1	4	4

4	2013-01-05	1	1	10	2013	1	5	5

Decompose the time series

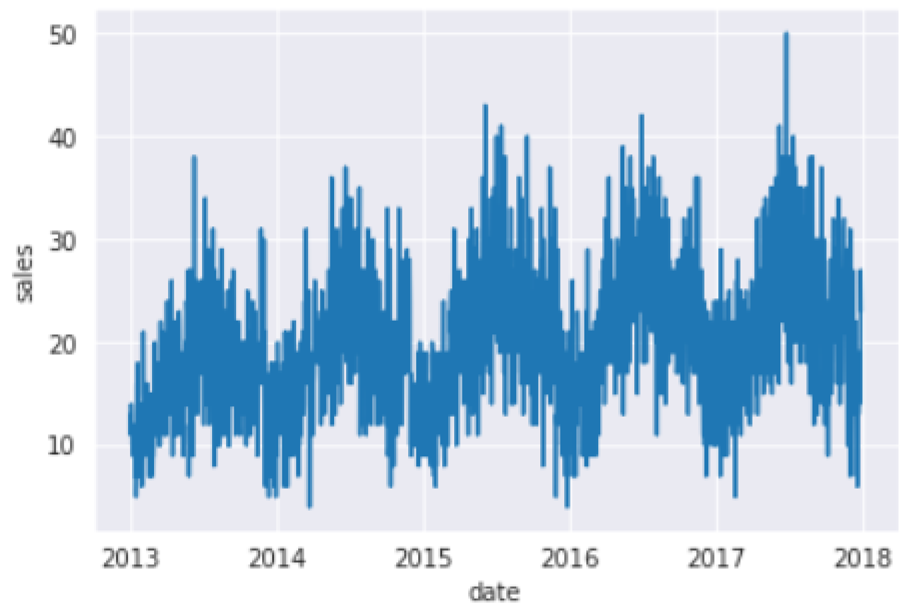
To start with, we want to decompose the data to seperate the seasonality, trend and residual. Since we have 5 years of sales data. We would expect there's a yearly or weekly pattern. Let's use a function in statsmodels to help us find it.

In [5]:

```
sns.lineplot(x="date", y="sales",legend = 'full' , data=train_df)
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbc42d7320>
```

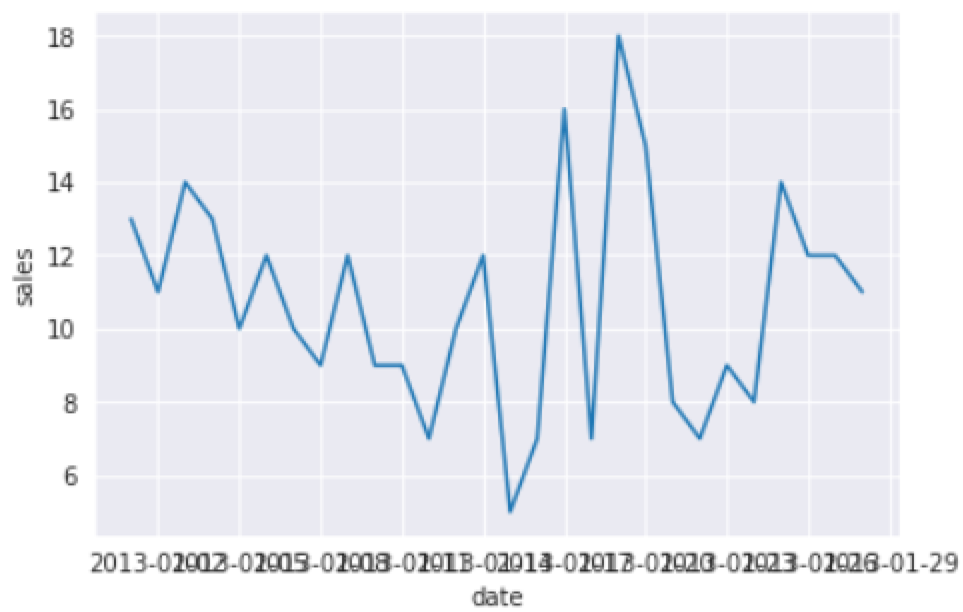


In [6]:

```
sns.lineplot(x="date", y="sales",legend = 'full' , data=train_df[:28])
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbc49a12e8>
```

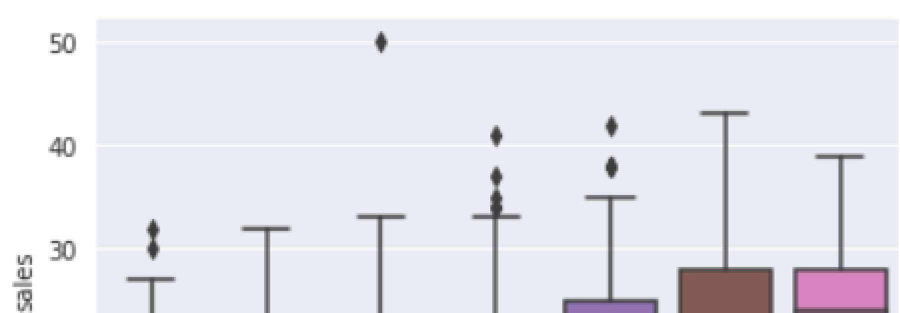


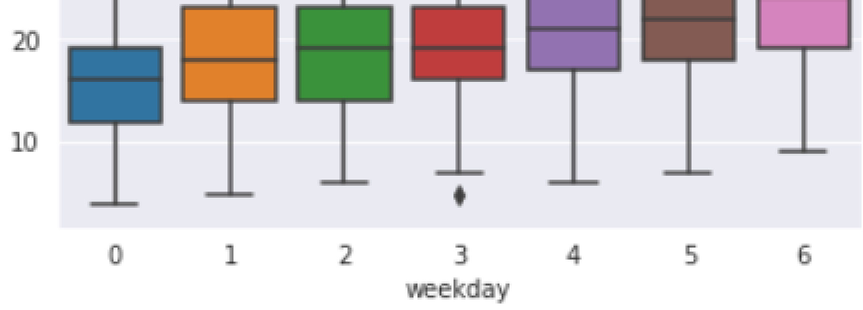
In [7]:

```
sns.boxplot(x="weekday", y="sales", data=train_df)
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbc491bdd8>
```





Monday=0, Sunday=6.
Here we can find the weekends(5,6) has a larger sales, weekdays(0-4) are smaller. There's a few outliers on Monday, Wed.

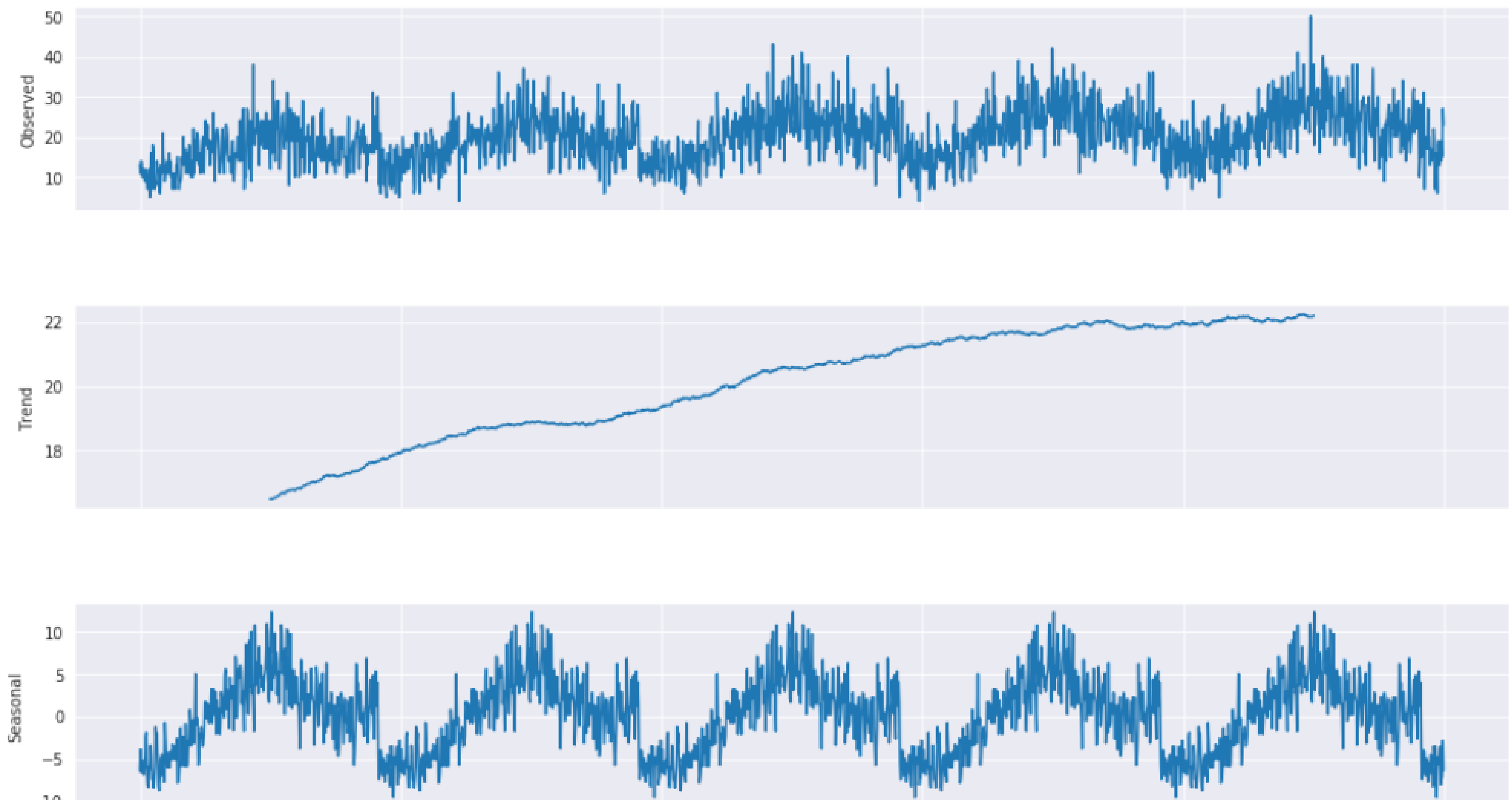
In [8]:
```
train_df = train_df.set_index('date')
train_df['sales'] = train_df['sales'].astype(float)

train_df.head()
```

Out[8]:

	store	item	sales	year	month	day	weekday
date							
2013-01-01	1	1	13.0	2013	1	1	1
2013-01-02	1	1	11.0	2013	1	2	2
2013-01-03	1	1	14.0	2013	1	3	3
2013-01-04	1	1	13.0	2013	1	4	4
2013-01-05	1	1	10.0	2013	1	5	5

In [9]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(train_df['sales'], model='additive', freq=365)

fig = plt.figure()
fig = result.plot()
fig.set_size_inches(15, 12)
```

```
<Figure size 432x288 with 0 Axes>
```

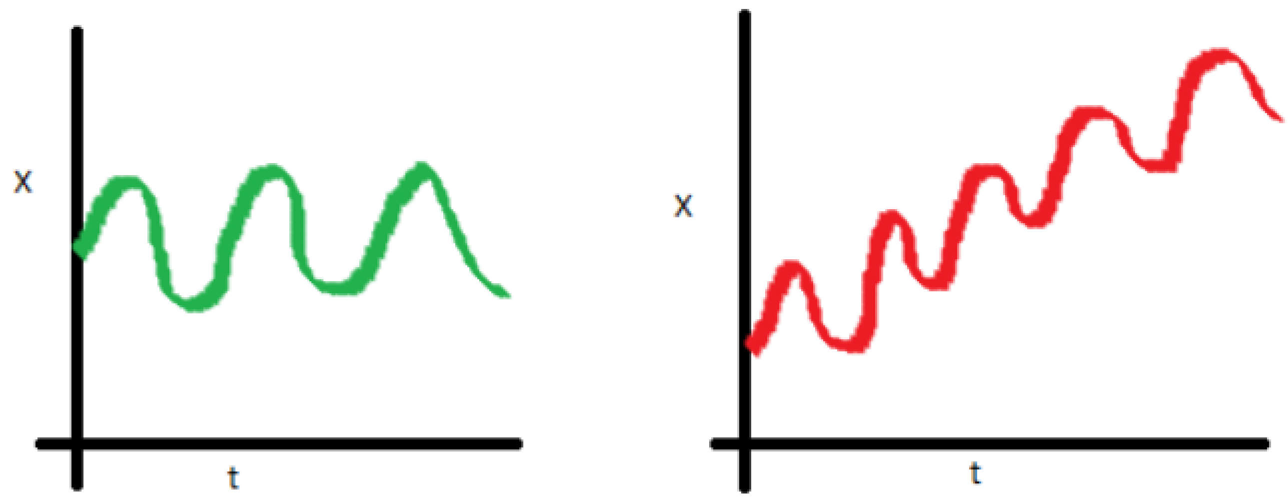


Playing with a few frequency, the yearly pattern is very obvious. and also we can see a upwards trend. Which means this data is not stationary.

Stationarize the data:

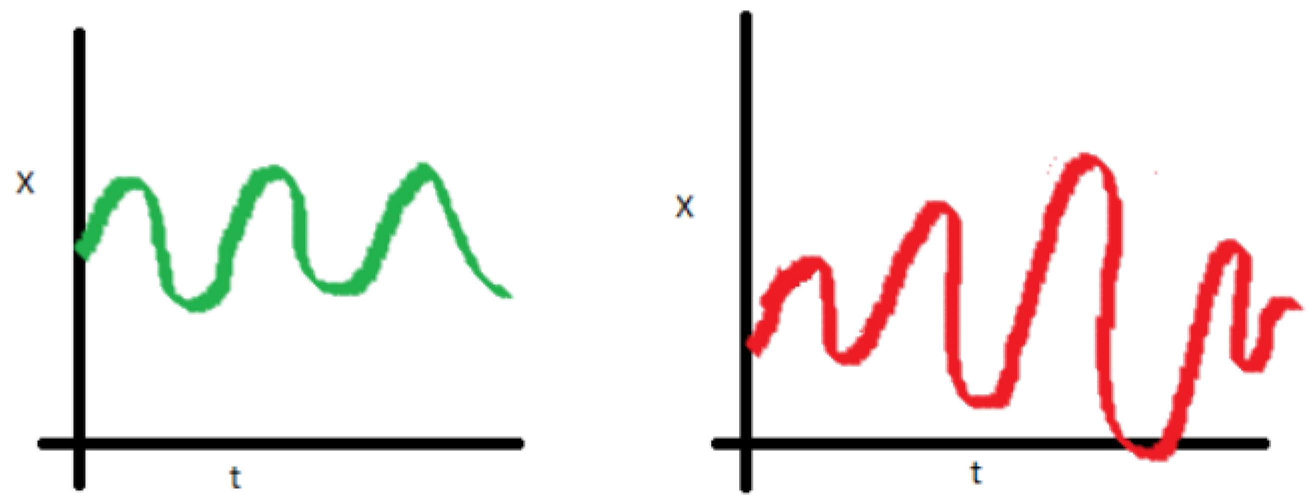
What does it mean for data to be stationary?

The mean of the series should not be a function of time. The red graph below is not stationary because the mean increases over time.



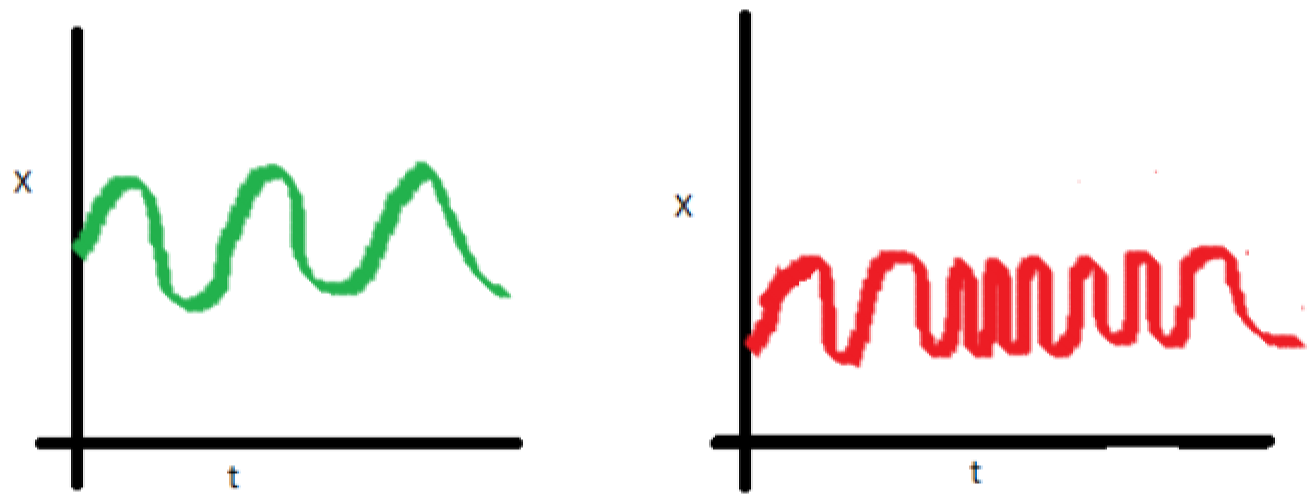
Stationary series Non-Stationary series

The variance of the series should not be a function of time. This property is known as homoscedasticity. Notice in the red graph the varying spread of data over time.



Stationary series Non-Stationary series

Finally, the covariance of the i th term and the (i + m) th term should not be a function of time. In the following graph, you will notice the spread becomes closer as the time increases. Hence, the covariance is not constant with time for the 'red series'.



Stationary series Non-Stationary series

Why is this important? When running a linear regression the assumption is that all of the observations are all independent of each

Why is this important? When running a linear regression the assumption is that all of the observations are all independent of each other. In a time series, however, we know that observations are time dependent. It turns out that a lot of nice results that hold for independent random variables (law of large numbers and central limit theorem to name a couple) hold for stationary random variables. So by making the data stationary, we can actually apply regression techniques to this time dependent variable.

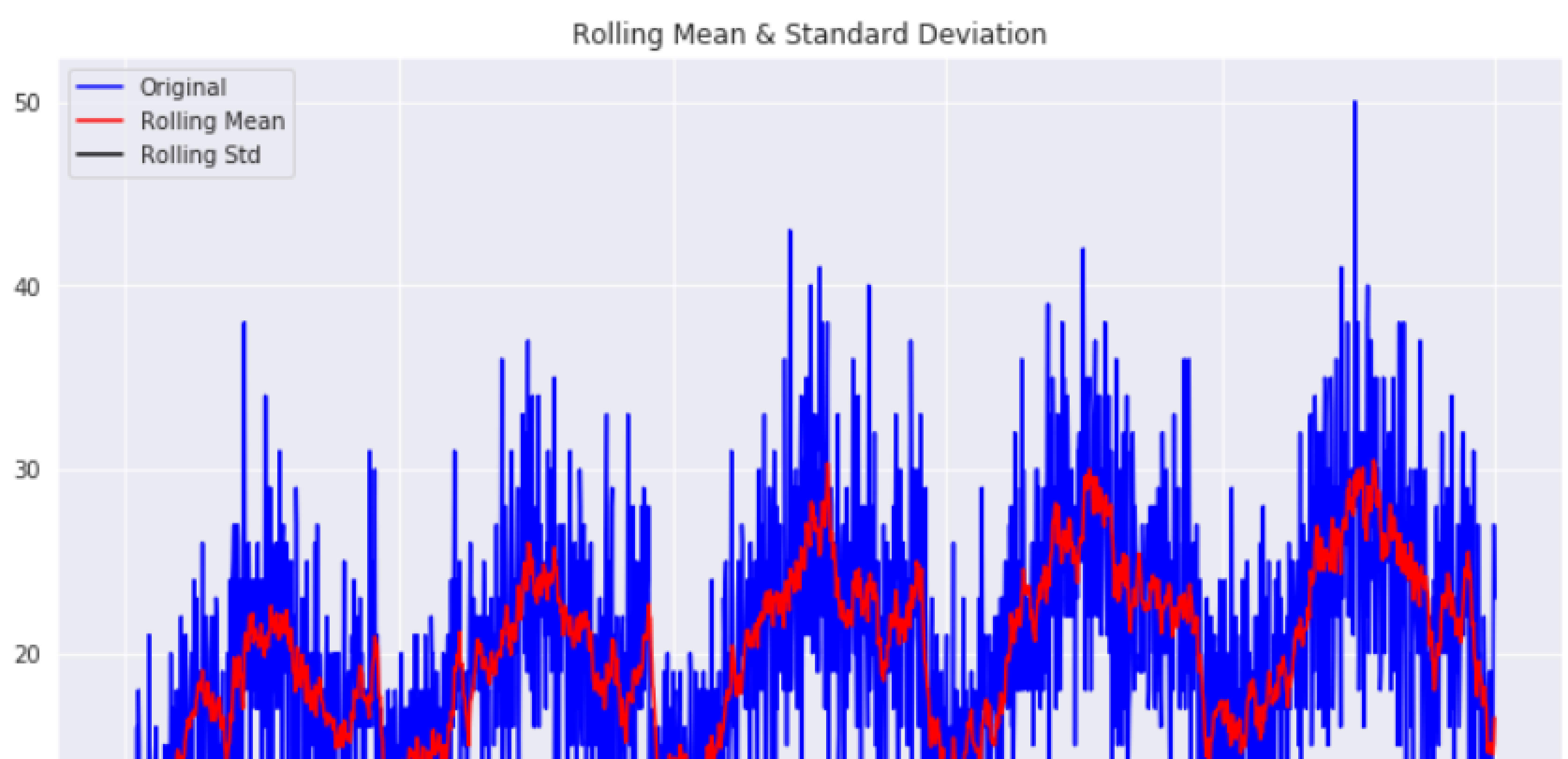
There are two ways you can check the stationarity of a time series. The first is by looking at the data. By visualizing the data it should be easy to identify a changing mean or variation in the data. For a more accurate assessment there is the Dickey-Fuller test. I won't go into the specifics of this test, but if the 'Test Statistic' is greater than the 'Critical Value' than the time series is stationary. Below is code that will help you visualize the time series and test for stationarity.

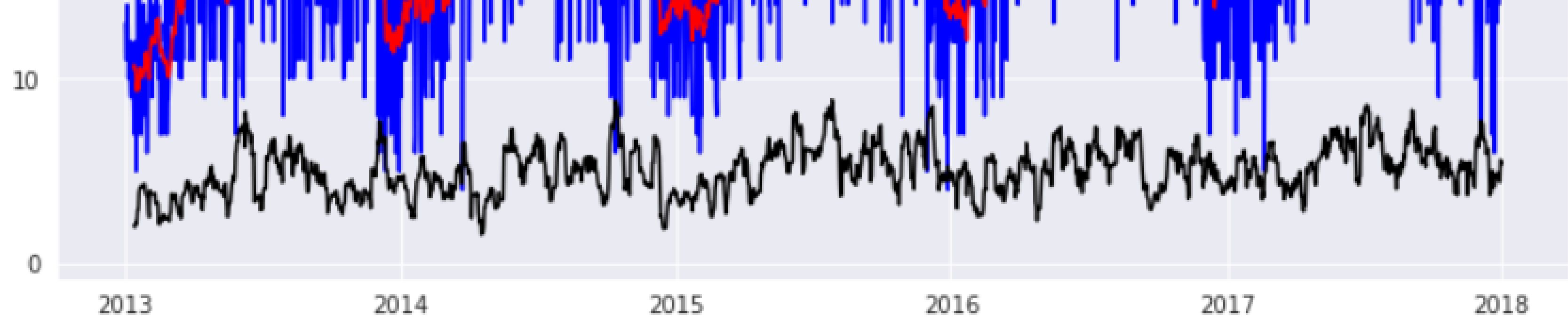
In [10]:

```
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries, window = 12, cutoff = 0.01):

    #Determing rolling statistics
    rolmean = timeseries.rolling(window).mean()
    rolstd = timeseries.rolling(window).std()

    #Plot rolling statistics:
    fig = plt.figure(figsize=(12, 8))
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show()

    #Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC', maxlag = 20 )
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of
Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    pvalue = dftest[1]
    if pvalue < cutoff:
        print('p-value = %.4f. The series is likely stationary.' % pvalue)
    else:
        print('p-value = %.4f. The series is likely non-stationary.' % pvalue)

    print(dfoutput)
```

In [11]:

```
test_stationarity(train_df['sales'])
```



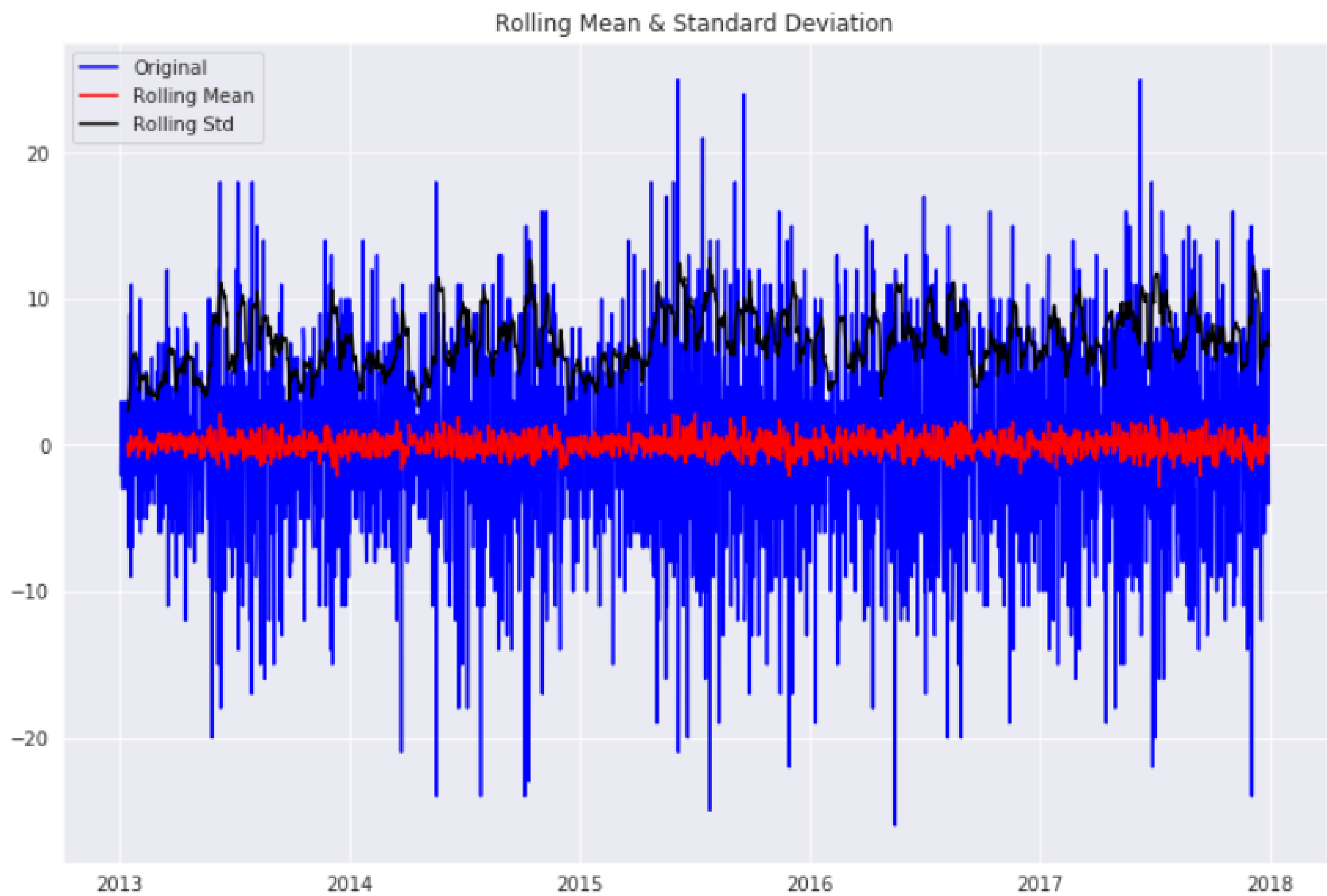


```
Results of Dickey-Fuller Test:
p-value = 0.0361. The series is likely non-stationary.
Test Statistic                   -2.987278
p-value                           0.036100
#Lags Used                       20.000000
Number of Observations Used    1805.000000
Critical Value (1%)              -3.433978
Critical Value (5%)              -2.863143
Critical Value (10%)             -2.567623
dtype: float64
```

the smaller p-value, the more likely it's stationary. Here our p-value is 0.036. It's actually not bad, if we use a 5% Critical Value(CV), this series would be considered stationary. But as we just visually found an upward trend, we want to be more strict, we use 1% CV. To get a stationary data, there's many techiniques. We can use log, differencing etc...

In [12]:

```
first_diff = train_df.sales - train_df.sales.shift(1)
first_diff = first_diff.dropna(inplace = False)
test_stationarity(first_diff, window = 12)
```




```
Results of Dickey-Fuller Test:
p-value = 0.0000. The series is likely stationary.
Test Statistic                -1.520810e+01
p-value                        5.705031e-28
#Lags Used                     2.000000e+01
Number of Observations Used    1.804000e+03
Critical Value (1%)           -3.433980e+00
```

```
Critical Value (5%)          -2.863143e+00
Critical Value (10%)         -2.567624e+00
dtype: float64
```

After differencing, the p-value is extremely small. Thus this series is very likely to be stationary.

ACF and PACF

The partial autocorrelation at lag k is the correlation that results after removing the effect of any correlations due to the terms at shorter lags.

Autoregression Intuition

Consider a time series that was generated by an autoregression (AR) process with a lag of k.

We know that the ACF describes the autocorrelation between an observation and another observation at a prior time step that includes direct and indirect dependence information.

This means we would expect the ACF for the AR(k) time series to be strong to a lag of k and the inertia of that relationship would carry on to subsequent lag values, trailing off at some point as the effect was weakened.

We know that the PACF only describes the direct relationship between an observation and its lag. This would suggest that there would be no correlation for lag values beyond k.

This is exactly the expectation of the ACF and PACF plots for an AR(k) process.

Moving Average Intuition

Consider a time series that was generated by a moving average (MA) process with a lag of k.

Remember that the moving average process is an autoregression model of the time series of residual errors from prior predictions. Another way to think about the moving average model is that it corrects future forecasts based on errors made on recent forecasts.

We would expect the ACF for the MA(k) process to show a strong correlation with recent values up to the lag of k, then a sharp decline to low or no correlation. By definition, this is how the process was generated.

For the PACF, we would expect the plot to show a strong relationship to the lag and a trailing off of correlation from the lag onwards.

Again, this is exactly the expectation of the ACF and PACF plots for an MA(k) process.

Summary

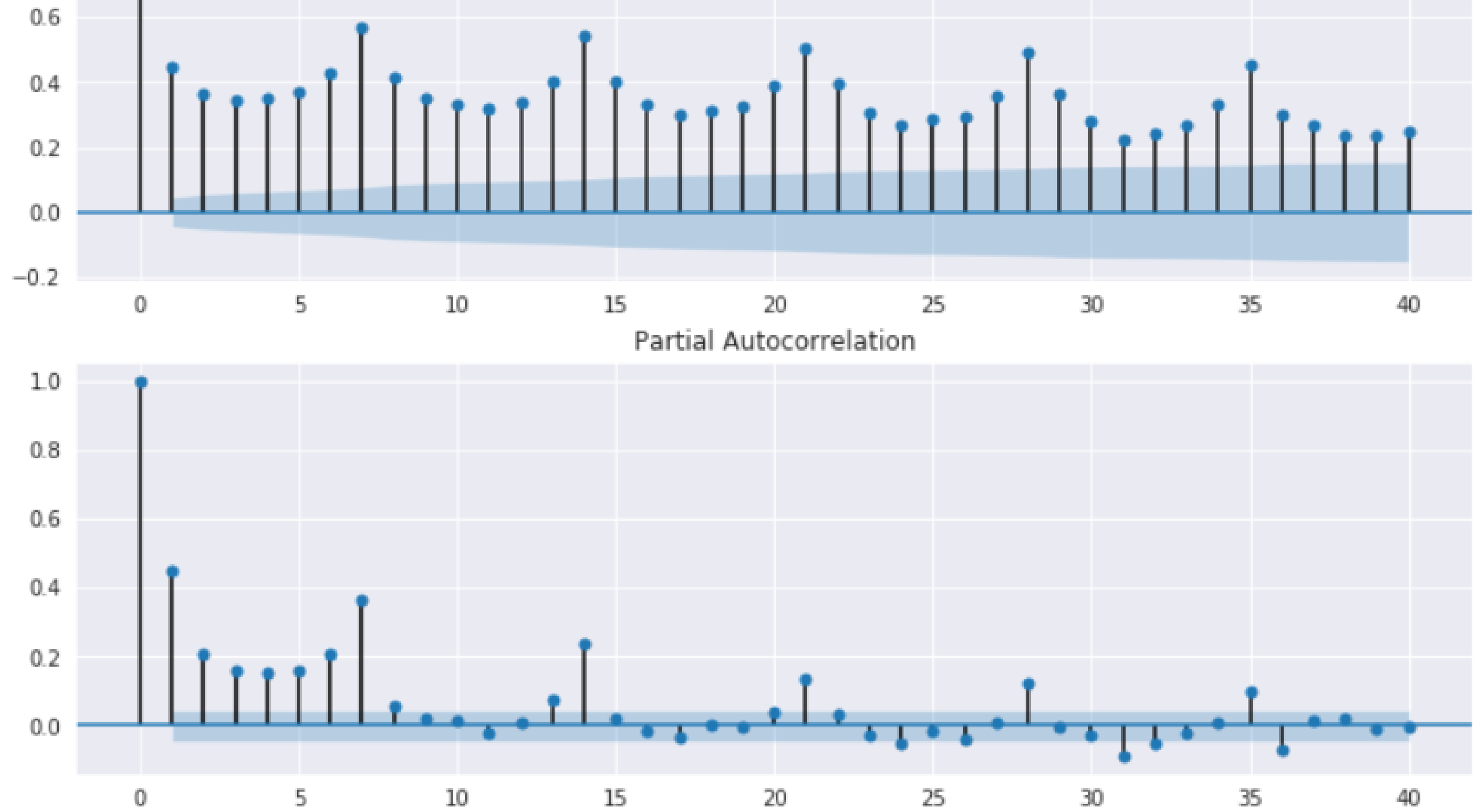
From the autocorrelation plot we can tell whether or not we need to add MA terms. From the partial autocorrelation plot we know we need to add AR terms.

References:

https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/

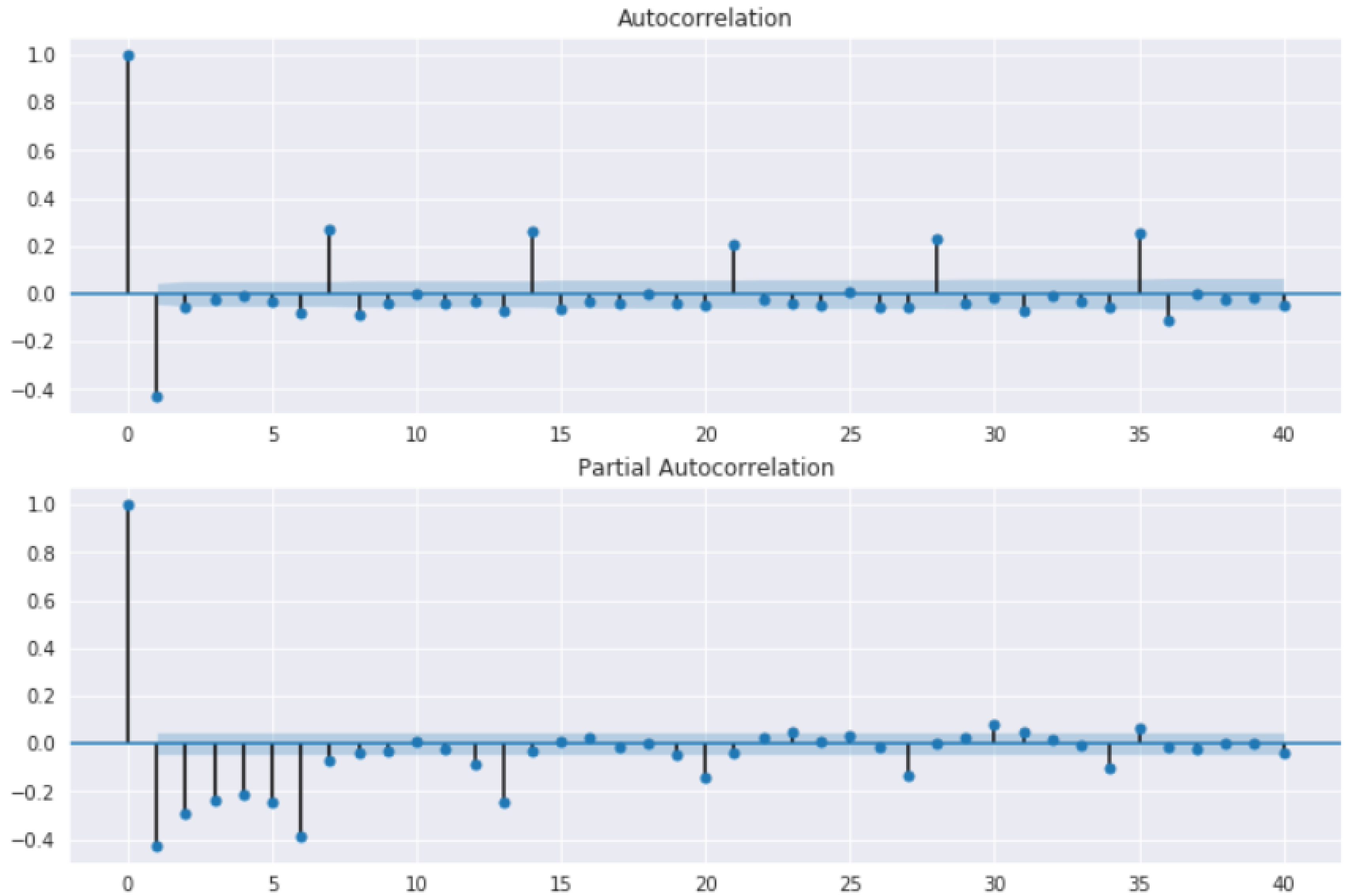
In [13]:

```
import statsmodels.api as sm

fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(train_df.sales, lags=40, ax=ax1) #
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(train_df.sales, lags=40, ax=ax2)# , lags=40
```



In [14]:

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(first_diff, lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(first_diff, lags=40, ax=ax2)

# Here we can see the acf and pacf both has a recurring pattern every 7 periods. Indicating a wee
kly pattern exists.
# Any time you see a regular pattern like that in one of these plots, you should suspect that the
re is some sort of
# significant seasonal thing going on. Then we should start to consider SARIMA to take seasonalit
y into accuont
```



Because the autocorrelation of the differenced series is negative at lag 7, 14, 21 etc.. (every week), I should an SMA term to the model.

Build the model

How to determin p, d, q

It's easy to determin I. In our case, we see the first order differencing make the ts stationary. **I = 1**.

AR model might be investigated first with lag length selected from the PACF or via empirical investigation. In our case, it's clearly that within 6 lags the AR is significant. Which means, we can use **AR = 6**

To avoid the potential for incorrectly specifying the MA order (in the case where the MA is first tried then the MA order is being set to 0), it may often make sense to extend the lag observed from the last significant term in the PACF.

What is interesting is that when the AR model is appropriately specified, the the residuals from this model can be used to directly observe the uncorrelated error. This residual can be used to further investigate alternative MA and ARMA model specifications directly by regression.

Assuming an AR(s) model were computed, then I would suggest that the next step in identification is to estimate an MA model with s-1 lags in the uncorrelated errors derived from the regression. The parsimonious MA specification might be considered and this might be compared with a more parsimonious AR specification. Then ARMA models might also be analysed.

Reference:

https://www.researchgate.net/post/How_does_one_determine_the_values_for_ARp_and_MAq
https://stats.stackexchange.com/questions/281666/how-does-acf-pacf-identify-the-order-of-ma-and-ar-terms/281726#281726
https://stats.stackexchange.com/questions/134487/analyse-acf-and-pacf-plots?rq=1

In [15]:

```
arima_mod6 = sm.tsa.ARIMA(train_df.sales, (6,1,0)).fit(disp=False)
print(arima_mod6.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.sales   No. Observations:                 1825
Model:                 ARIMA(6, 1, 0)   Log Likelihood               -5597.668
Method:                       css-mle   S.D. of innovations              5.195
Date:                Mon, 20 Aug 2018   AIC                          11211.335
Time:                        05:28:29   BIC                          11255.410
Sample:                    01-02-2013   HQIC                         11227.594
                         - 12-31-2017
=================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------
const             0.0039      0.025      0.152      0.879      -0.046       0.054
ar.L1.D.sales    -0.8174      0.022    -37.921      0.000      -0.860      -0.775
ar.L2.D.sales    -0.7497      0.026    -28.728      0.000      -0.801      -0.699
ar.L3.D.sales    -0.6900      0.028    -24.665      0.000      -0.745      -0.635
ar.L4.D.sales    -0.6138      0.028    -21.950      0.000      -0.669      -0.559
ar.L5.D.sales    -0.5247      0.026    -20.132      0.000      -0.576      -0.474
ar.L6.D.sales    -0.3892      0.022    -18.064      0.000      -0.431      -0.347
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            0.6842            -0.8982j            1.1292           -0.1464
AR.2            0.6842            +0.8982j            1.1292            0.1464
AR.3           -1.0869            -0.5171j            1.2037           -0.4293
AR.4           -1.0869            +0.5171j            1.2037            0.4293
AR.5           -0.2714            -1.1477j            1.1794           -0.2870
AR.6           -0.2714            +1.1477j            1.1794            0.2870
-----------------------------------------------------------------------------
```

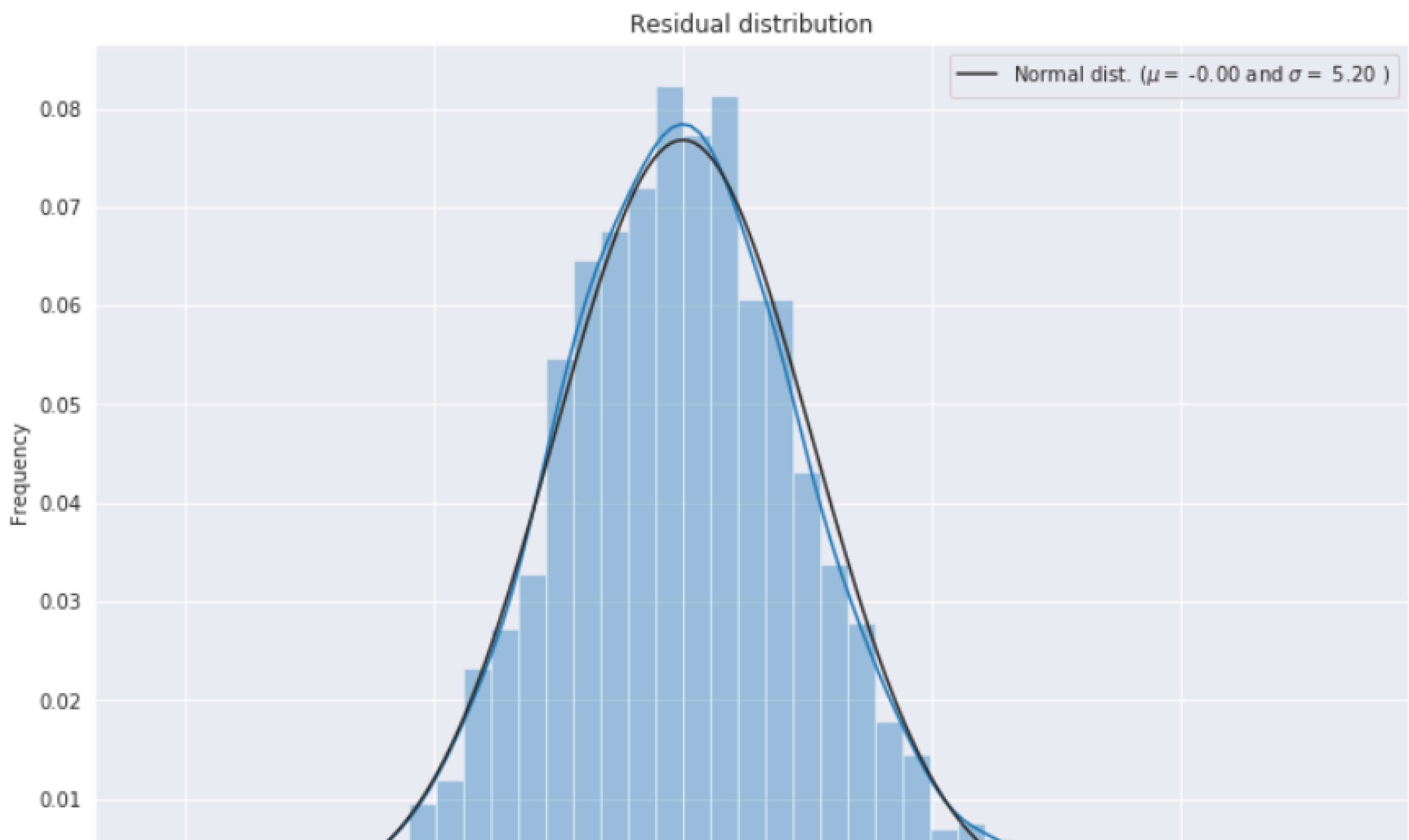
Analyze the result

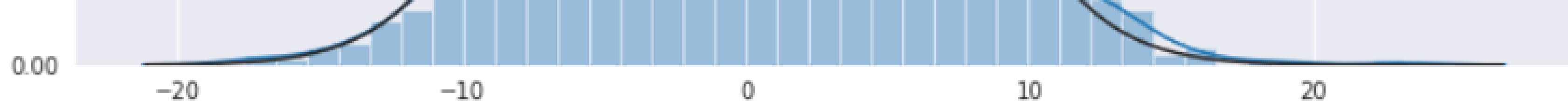
To see how our first model perform, we can plot the residual distribution. See if it's normal dist. And the ACF and PACF. For a good model, we want to see the residual is normal distribution. And ACF, PACF has not significant terms.

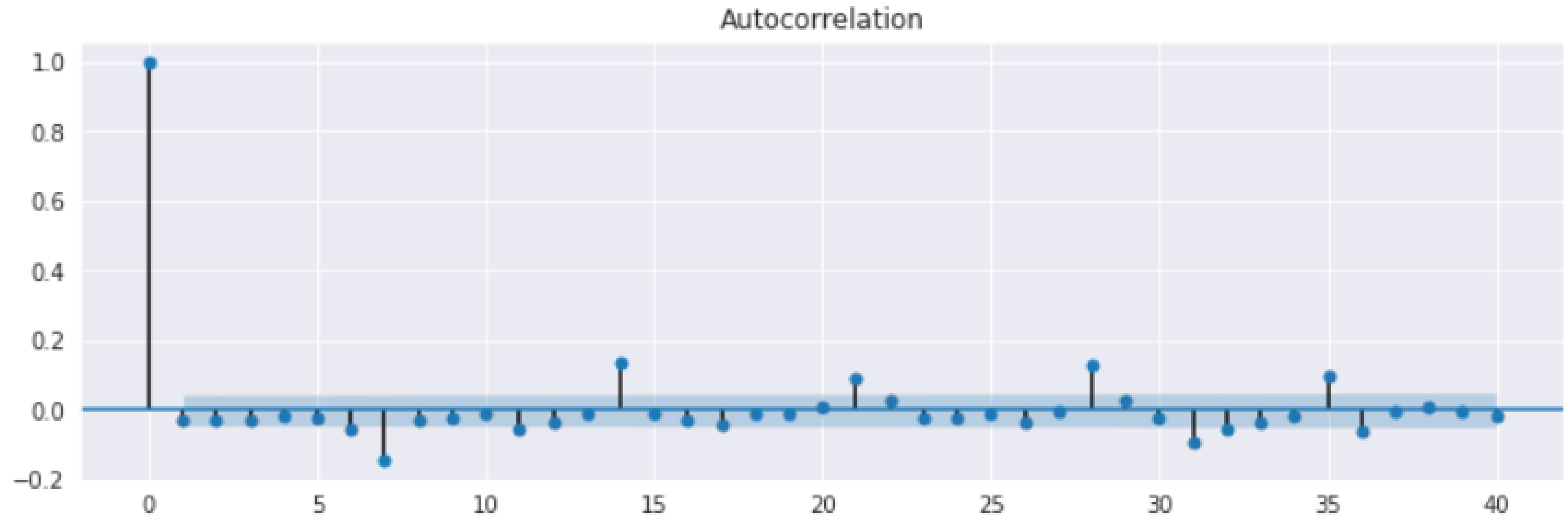
In [16]:

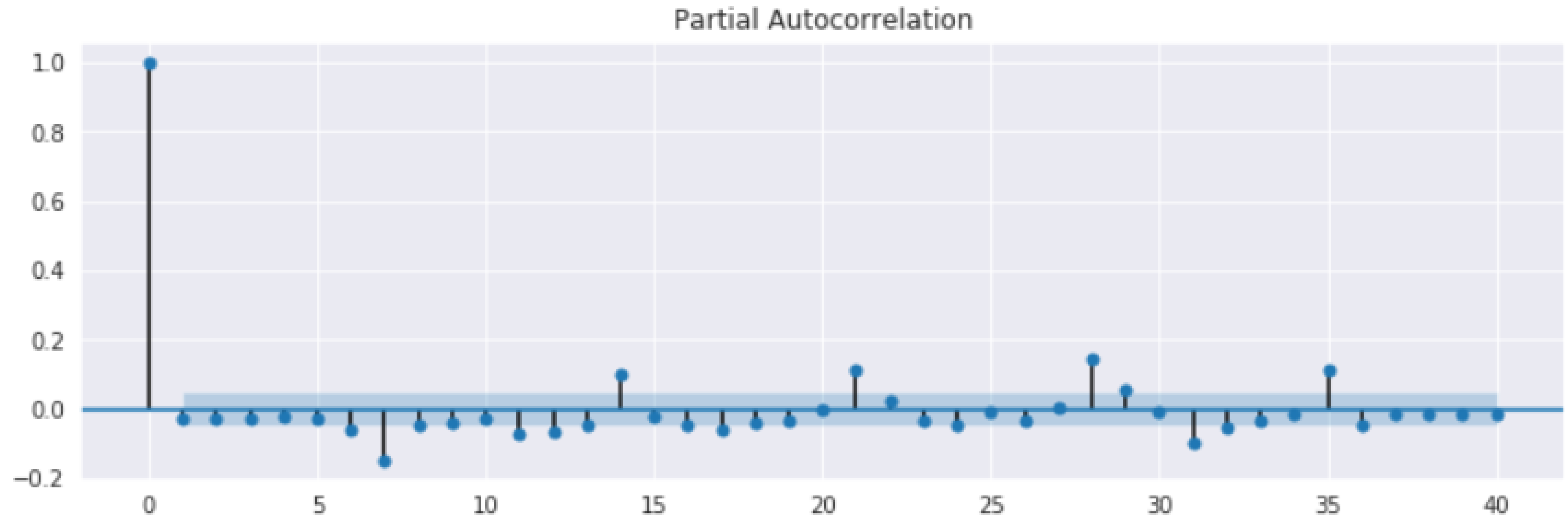
```
from scipy import stats
from scipy.stats import normaltest

resid = arima_mod6.resid
print(normaltest(resid))
# returns a 2-tuple of the chi-squared statistic, and the associated p-value. the p-value is very small, meaning
# the residual is not a normal distribution

fig = plt.figure(figsize=(12,8))
ax0 = fig.add_subplot(111)

sns.distplot(resid ,fit = stats.norm, ax = ax0) # need to import scipy.stats

# Get the fitted parameters used by the function
(mu, sigma) = stats.norm.fit(resid)

#Now plot the distribution using
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], loc='best')
plt.ylabel('Frequency')
plt.title('Residual distribution')


# ACF and PACF
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(arima_mod6.resid, lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(arima_mod6.resid, lags=40, ax=ax2)
```

```
NormaltestResult(statistic=16.42638861347859, pvalue=0.0002710535089055376)
```









Although the graph looks very like a normal distribution. But it failed the test. Also we see a recurring correlation exists in both ACF and PACF. So we need to deal with seasonality.

Consider seasonality affect by SARIMA

https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_sarimax_stata.html
https://barnesanalytics.com/sarima-models-using-statsmodels-in-python

In [17]:

```
sarima_mod6 = sm.tsa.statespace.SARIMAX(train_df.sales, trend='n', order=(6,1,0)).fit()
print(sarima_mod6.summary())
```

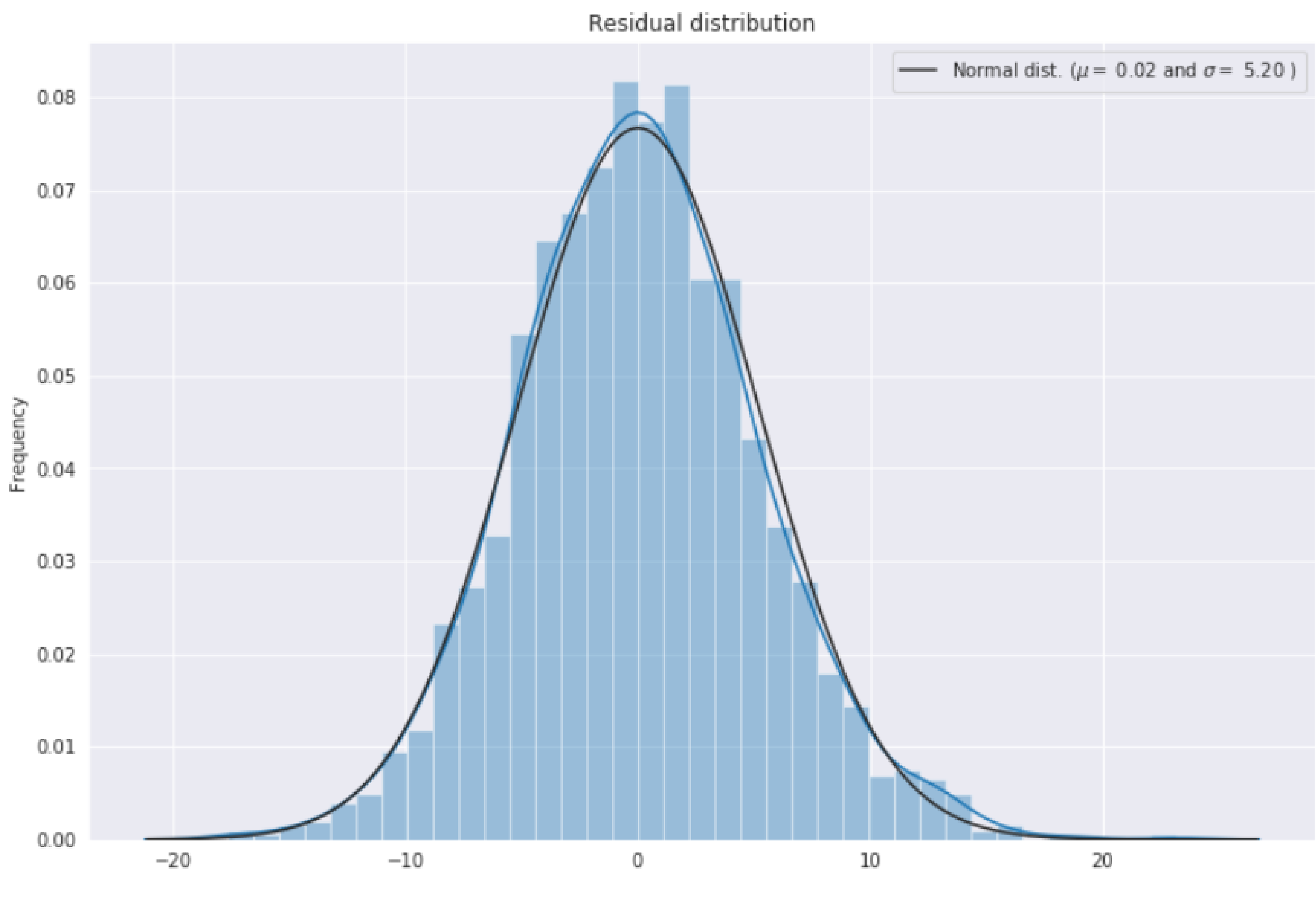
```
                           Statespace Model Results
==============================================================================
Dep. Variable:                  sales   No. Observations:                 1826
Model:               SARIMAX(6, 1, 0)   Log Likelihood               -5597.679
Date:                Mon, 20 Aug 2018   AIC                          11209.359
Time:                        05:28:37   BIC                          11247.928
Sample:                    01-01-2013   HQIC                         11223.586
                         - 12-31-2017
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.8174      0.021    -39.063      0.000      -0.858      -0.776
ar.L2         -0.7497      0.025    -30.480      0.000      -0.798      -0.702
ar.L3         -0.6900      0.026    -26.686      0.000      -0.741      -0.639
ar.L4         -0.6138      0.027    -22.743      0.000      -0.667      -0.561
ar.L5         -0.5247      0.025    -21.199      0.000      -0.573      -0.476
ar.L6         -0.3892      0.021    -18.819      0.000      -0.430      -0.349
sigma2        26.9896      0.817     33.037      0.000      25.388      28.591
===================================================================================
```

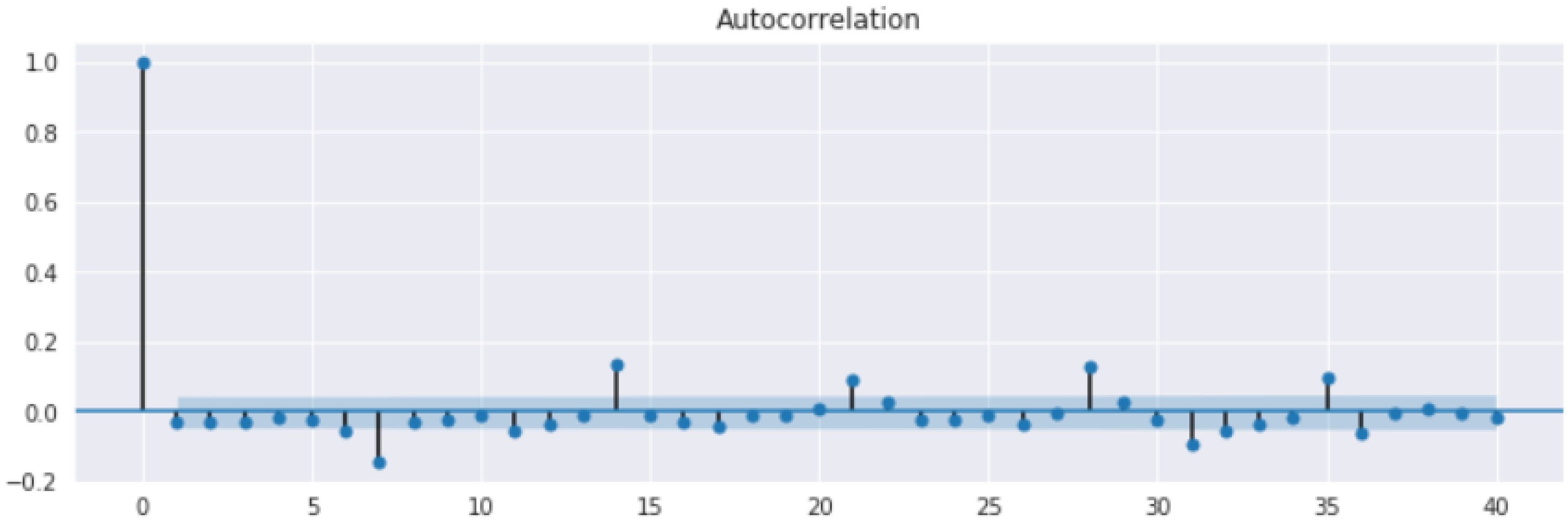
```
Ljung-Box (Q):                      205.88   Jarque-Bera (JB):               19.53
Prob(Q):                              0.00   Prob(JB):                        0.00
Heteroskedasticity (H):               1.41   Skew:                            0.15
Prob(H) (two-sided):                  0.00   Kurtosis:                        3.40
===================================================================================

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

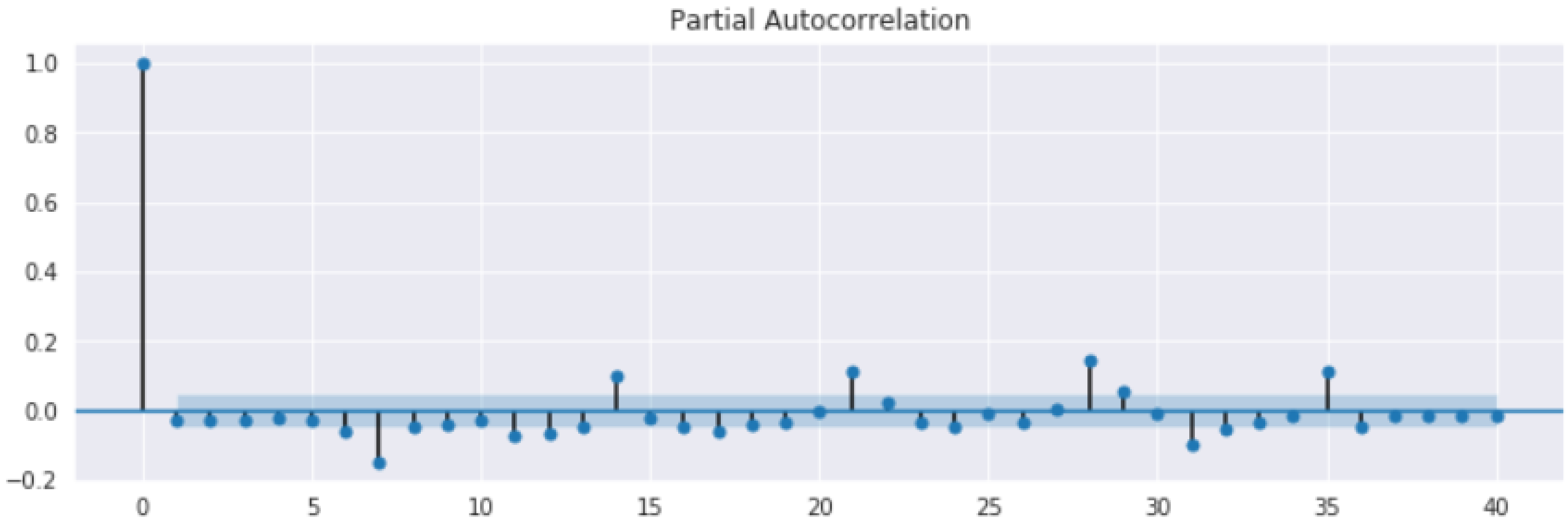
In [18]:

```python
resid = sarima_mod6.resid
print(normaltest(resid))

fig = plt.figure(figsize=(12,8))
ax0 = fig.add_subplot(111)

sns.distplot(resid ,fit = stats.norm, ax = ax0) # need to import scipy.stats

# Get the fitted parameters used by the function
(mu, sigma) = stats.norm.fit(resid)

#Now plot the distribution using
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)], loc='best')
plt.ylabel('Frequency')
plt.title('Residual distribution')


# ACF and PACF
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(arima_mod6.resid, lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(arima_mod6.resid, lags=40, ax=ax2)
```

```
NormaltestResult(statistic=16.74269015239875, pvalue=0.00023140408881811444)
```







Make prediction and evaluation

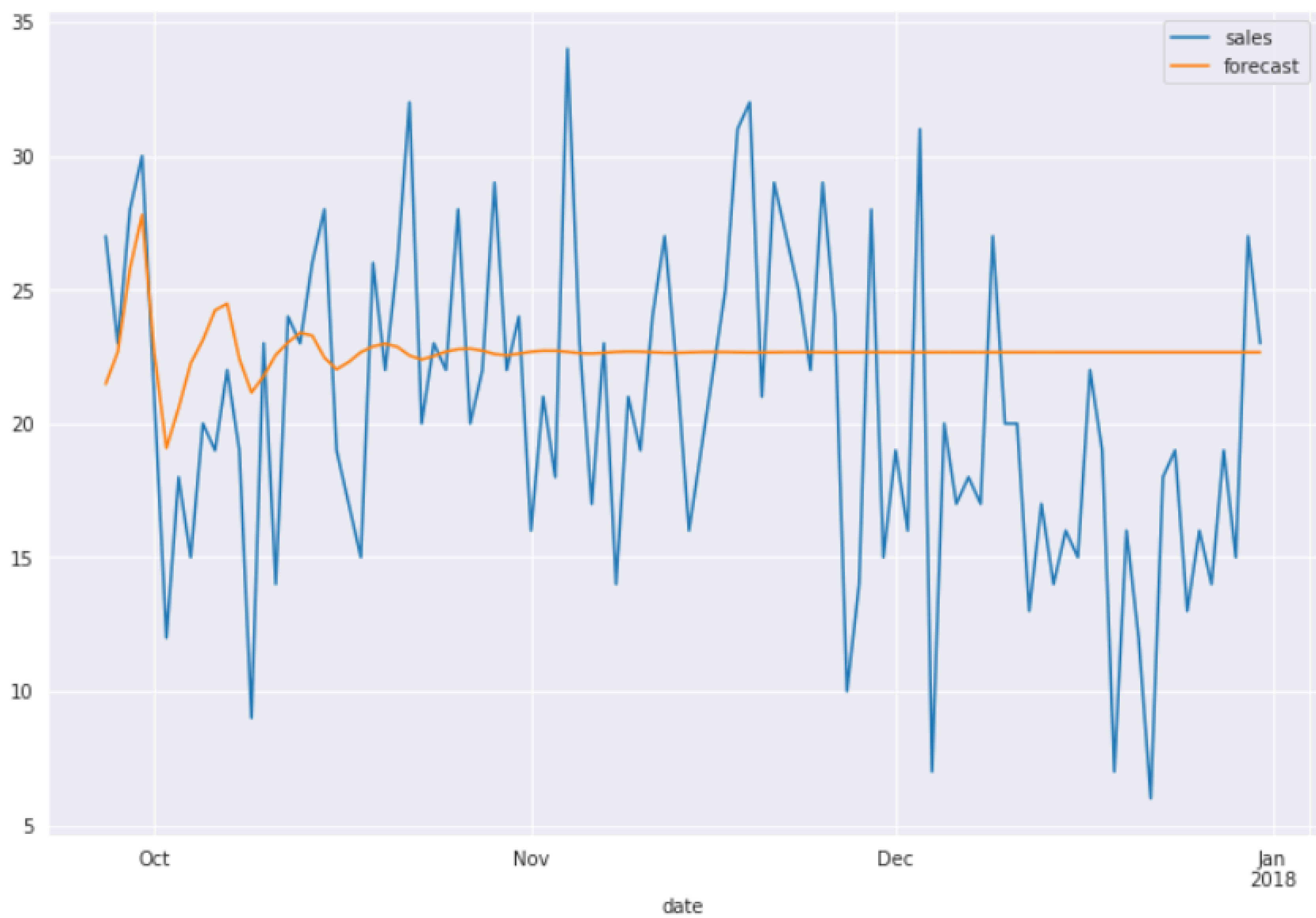
Take the last 30 days in training set as validation data

In [19]:

```
start_index = 1730
end_index = 1826
train_df['forecast'] = sarima_mod6.predict(start = start_index, end= end_index, dynamic= True)
train_df[start_index:end_index][['sales', 'forecast']].plot(figsize=(12, 8))
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbac14ce80>
```



In [20]:

```
def smape_kun(y_true, y_pred):
    mape = np.mean(abs((y_true-y_pred)/y_true))*100
    smape = np.mean((np.abs(y_pred - y_true) * 200/ (np.abs(y_pred) + np.abs(y_true))).fillna
(0))
    print('MAPE: %.2f %% \nSMAPE: %.2f'% (mape,smape), "%")
```

In [21]:

```
smape_kun(train_df[1730:1825]['sales'],train_df[1730:1825]['forecast'])
```

```
MAPE: 33.01 %
SMAPE: 25.07 %
```

SARIMAX: adding external variables

In [22]:

```
# per 1 store, 1 item
storeid = 1
itemid = 1
train_df = train[train['store']==storeid]
train_df = train_df[train_df['item']==itemid]

# train_df = train_df.set_index('date')
train_df['year'] = train_df['date'].dt.year - 2012
train_df['month'] = train_df['date'].dt.month
train_df['day'] = train_df['date'].dt.dayofyear
train_df['weekday'] = train_df['date'].dt.weekday

start_index = 1730
end_index = 1826

# train_df.head()
```

In [23]:

```
holiday = pd.read_csv('../input/holiday/USholidays.csv',header=None, names = ['date', 'holida
y'])
holiday['date'] = pd.to_datetime(holiday['date'], yearfirst = True, format = '%y/%m/%d')
holiday.head()
```

Out[23]:

	date	holiday
0	2012-01-02	New Year Day
1	2012-01-16	Martin Luther King Jr. Day
2	2012-02-20	Presidents Day (Washingtons Birthday)
3	2012-05-28	Memorial Day
4	2012-07-04	Independence Day

In [24]:

```
train_df = train_df.merge(holiday, how='left', on='date')
train_df['holiday_bool'] = pd.notnull(train_df['holiday']).astype(int)
train_df = pd.get_dummies(train_df, columns = ['month','holiday','weekday'] , prefix = ['mont
h','holiday','weekday'])
# train_df.head()
```

```
# train_df.shape
# train_df.columns
```

In [25]:

```
ext_var_list = ['date','year', 'day', 'holiday_bool',
       'month_1', 'month_2', 'month_3', 'month_4', 'month_5', 'month_6',
       'month_7', 'month_8', 'month_9', 'month_10', 'month_11', 'month_12',
       'holiday_Christmas Day', 'holiday_Columbus Day',
       'holiday_Independence Day', 'holiday_Labor Day',
       'holiday_Martin Luther King Jr. Day', 'holiday_Memorial Day',
       'holiday_New Year Day', 'holiday_Presidents Day (Washingtons Birthday)',
       'holiday_Thanksgiving Day', 'holiday_Veterans Day', 'weekday_0',
       'weekday_1', 'weekday_2', 'weekday_3', 'weekday_4', 'weekday_5',
       'weekday_6']
```

In [26]:

```
exog_data = train_df[ext_var_list]
exog_data = exog_data.set_index('date')
exog_data.head()
```

Out[26]:

	year	day	holiday_bool	month_1	month_2	month_3	month_4	month_5	month_6	month_7	month_8	month_9	month
date													
2013-01-01	1	1	1	1	0	0	0	0	0	0	0	0	0
2013-01-02	1	2	0	1	0	0	0	0	0	0	0	0	0
2013-01-03	1	3	0	1	0	0	0	0	0	0	0	0	0
2013-01-04	1	4	0	1	0	0	0	0	0	0	0	0	0
2013-01-05	1	5	0	1	0	0	0	0	0	0	0	0	0

In [27]:

```
train_df = train_df.set_index('date')
# train_df = train_df.reset_index()
train_df.head()
```

Out[27]:

	store	item	sales	year	day	holiday_bool	month_1	month_2	month_3	month_4	month_5	month_6	month_7	month
date														
2013-01-01	1	1	13	1	1	1	1	0	0	0	0	0	0	0
2013-01-02	1	1	11	1	2	0	1	0	0	0	0	0	0	0
2013-01-03	1	1	14	1	3	0	1	0	0	0	0	0	0	0
2013-01-04	1	1	13	1	4	0	1	0	0	0	0	0	0	0
2013-01-05	1	1	10	1	5	0	1	0	0	0	0	0	0	0

In [28]:

```
start_index = '2017-10-01'
end_index = '2017-12-31'
# exog_data.head()
```

In [29]:

```
%%time
sarimax_mod6 = sm.tsa.statespace.SARIMAX(endog = train_df.sales[:start_index],
                                        exog = exog_data[:start_index],
                                        trend='n', order=(6,1,0), seasonal_order=(0,1,1,7)).fit
()
print(sarimax_mod6.summary())
```

```
/opt/conda/lib/python3.6/site-packages/statsmodels/base/model.py:496: ConvergenceWarning: Maxim
um Likelihood optimization failed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)

                                 Statespace Model Results
==========================================================================================
Dep. Variable:                            sales   No. Observations:                 1735
Model:             SARIMAX(6, 1, 0)x(0, 1, 1, 7)   Log Likelihood               -5133.376
Date:                          Mon, 20 Aug 2018   AIC                          10346.751
Time:                                  05:32:22   BIC                          10565.102
Sample:                              01-01-2013   HQIC                         10427.503
                                   - 10-01-2017
Covariance Type:                            opg
==============================================================================================
==================
                                                   coef    std err          z      P>|z|
[0.025      0.975]
---------------------------------------------------------------------------------------------
-------------------
year                                          -432.7281   1667.189     -0.260      0.795    -37
00.359    2834.903
day                                             -1.1815      4.565     -0.259      0.796      -
10.129       7.766
holiday_bool                                    -0.3558      5.285     -0.067      0.946      -
10.714      10.003
month_1                                         -5.3239   2783.622     -0.002      0.998    -54
61.122    5450.475
month_2                                         -3.9888   2783.646     -0.001      0.999    -54
59.834    5451.856
month_3                                         -2.3152   2783.698     -0.001      0.999    -54
58.264    5453.634
month_4                                          1.0331   2783.629      0.000      1.000    -54
54.779    5456.845
month_5                                          1.4194   2783.634      0.001      1.000    -54
54.403    5457.242
month_6                                          3.4937   2783.604      0.001      0.999    -54
52.270    5459.258
month_7                                          4.7113   2783.611      0.002      0.999    -54
51.067    5460.489
month_8                                          2.1281   2783.630      0.001      0.999    -54
53.687    5457.943
```

```
month_9                                       2.4363   2783.625      0.001      0.999   -54
53.369    5458.242
month_10                                      0.1956   2783.631   7.03e-05      1.000   -54
55.621    5456.012
month_11                                      2.4600   2783.634      0.001      0.999   -54
53.362    5458.282
month_12                                     -6.2498   2783.647     -0.002      0.998   -54
62.097    5449.597
holiday_Christmas Day                         0.6370      5.678      0.112      0.911     -
10.493      11.766
holiday_Columbus Day                         -2.6386      5.709     -0.462      0.644     -
13.828       8.551
holiday_Independence Day                     -0.2743      5.518     -0.050      0.960     -
11.088      10.540
holiday_Labor Day                             1.1197      5.401      0.207      0.836
-9.466      11.705
holiday_Martin Luther King Jr. Day           -0.3839      5.845     -0.066      0.948     -
11.840      11.072
holiday_Memorial Day                          0.9443      5.451      0.173      0.862
-9.740      11.628
holiday_New Year Day                          0.6313      6.174      0.102      0.919     -
11.470      12.732
holiday_Presidents Day (Washingtons Birthday)    -0.7298      5.759     -0.127      0.899     -
12.017      10.557
holiday_Thanksgiving Day                     -1.1924      6.248     -0.191      0.849     -
13.438      11.053
holiday_Veterans Day                          1.5310      5.832      0.263      0.793
-9.900      12.962
weekday_0                                 -2.354e-05   9.41e+04   -2.5e-10      1.000   -1.
84e+05    1.84e+05
weekday_1                                 -6.254e-07   3.13e+04     -2e-11      1.000   -6.
13e+04    6.13e+04
weekday_2                                  9.363e-06   5.44e+04   1.72e-10      1.000   -1.
07e+05    1.07e+05
weekday_3                                 -7.188e-06   7.46e+04  -9.64e-11      1.000   -1.
46e+05    1.46e+05
weekday_4                                 -2.849e-06   4.53e+04  -6.29e-11      1.000   -8.
87e+04    8.87e+04
weekday_5                                  1.011e-05   7.73e+04   1.31e-10      1.000   -1.
52e+05    1.52e+05
weekday_6                                   4.54e-06   6.29e+04   7.21e-11      1.000   -1.
23e+05    1.23e+05
ar.L1                                        -0.8514      0.024    -35.827      0.000
-0.898      -0.805
ar.L2                                        -0.7492      0.030    -24.571      0.000
-0.809      -0.689
ar.L3                                        -0.6235      0.033    -18.632      0.000
-0.689      -0.558
ar.L4                                        -0.4776      0.033    -14.452      0.000
-0.542      -0.413
ar.L5                                        -0.3143      0.030    -10.314      0.000
-0.374      -0.255
ar.L6                                        -0.1813      0.023     -7.829      0.000
-0.227      -0.136
ma.S.L7                                      -0.9991      0.052    -19.310      0.000
-1.100      -0.898
sigma2                                       21.8632      1.224     17.857      0.000
19.464      24.263
===================================================================================
Ljung-Box (Q):                      103.73   Jarque-Bera (JB):                37.67
Prob(Q):                              0.00   Prob(JB):                         0.00
```

```
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               1.32   Skew:                             0.21
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.59
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 5.5e+17. Standard err
ors may be unstable.
CPU times: user 4min 28s, sys: 10min 25s, total: 14min 54s
Wall time: 3min 43s
```

These model coefficients are not very reliable as most of them are not significant. This would imply a high collinearity between the data.

In [30]:

```
start_index = '2017-10-01'
end_index = '2017-12-30'
end_index1 = '2017-12-31'
```

In [31]:

```
sarimax_mod6.forecast(steps = 121,exog = exog_data[start_index:end_index])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-31-2794689cebf7> in <module>()
----> 1 sarimax_mod6.forecast(steps = 121,exog = exog_data[start_index:end_index])

/opt/conda/lib/python3.6/site-packages/statsmodels/base/wrapper.py in wrapper(self, *args, **kw
args)
     93             obj = data.wrap_output(func(results, *args, **kwargs), how[0], how[1:])
     94         elif how:
---> 95             obj = data.wrap_output(func(results, *args, **kwargs), how)
     96         return obj
     97 

/opt/conda/lib/python3.6/site-packages/statsmodels/tsa/statespace/mlemodel.py in forecast(self,
steps, **kwargs)
   2394         else:
   2395             end = steps
-> 2396         return self.predict(start=self.nobs, end=end, **kwargs)
   2397 
   2398     def simulate(self, nsimulations, measurement_shocks=None,

/opt/conda/lib/python3.6/site-packages/statsmodels/tsa/statespace/mlemodel.py in predict(self,
 start, end, dynamic, **kwargs)
   2367         """
   2368         # Perform the prediction
-> 2369         prediction_results = self.get_prediction(start, end, dynamic, **kwargs)
   2370         return prediction_results.predicted_mean
   2371 

/opt/conda/lib/python3.6/site-packages/statsmodels/tsa/statespace/sarimax.py in get_prediction
(self, start, end, dynamic, exog, **kwargs)
   1901                                  ' appropriate shape. Required %s, got %s.'
   1902                                  % (str(required_exog_shape),
-> 1903                                     str(exog.shape)))
   1904                 exog = np.c_[self.model.data.orig_exog.T, exog.T].T
```

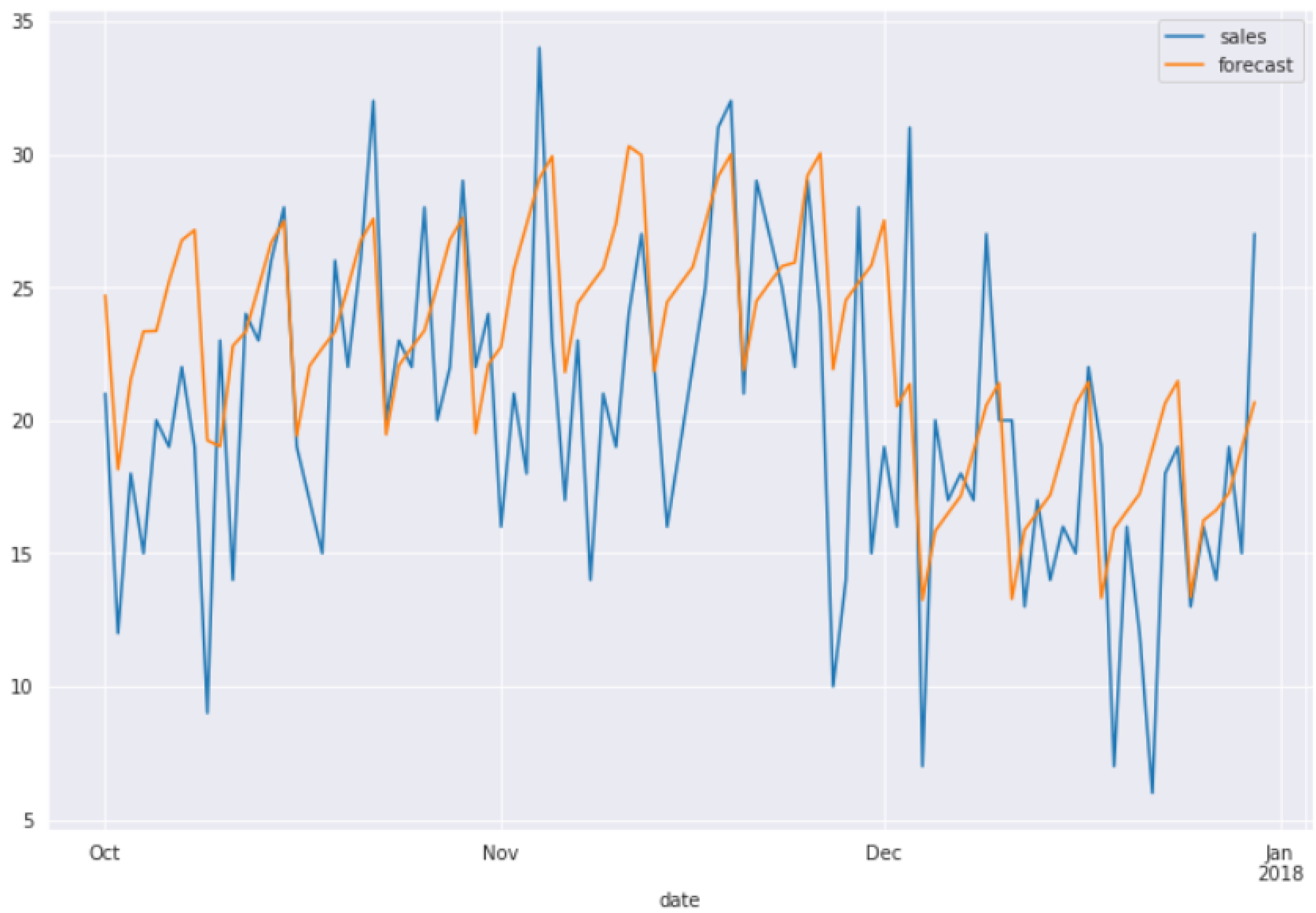
```
1905

ValueError: Provided exogenous values are not of the appropriate shape. Required (121, 32), got (91, 32).
```

In [32]:

```
train_df['forecast'] = sarimax_mod6.predict(start = pd.to_datetime(start_index), end= pd.to_datetime(end_index1),
                                            exog = exog_data[start_index:end_index],
                                            dynamic= True)

train_df[start_index:end_index][['sales', 'forecast']].plot(figsize=(12, 8))
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbbc476d400>
```



In [33]:

```
smape_kun(train_df[start_index:end_index]['sales'],train_df[start_index:end_index]['forecast'])
```

```
MAPE: 27.19 %
SMAPE: 21.93 %
```

Some last words:

ARIMA makes much more sense to me now. ACF and PACF are useful to determine the p, d, q. And each test is indeed helping me to justify whether I'm getting a better model or worse one.

Pros:

- Intepretability: Each coefficient means a specific thing
- ts key elements understanding: the concept of lags, and error lag terms are very unique, ARIMA gave a comprehensive cover on them. So even in the future I want to try some other regression model. I would add the lag terms and consider the error term.

Cons:

- Inefficiency: ARIMA needs to be run on each time series, since we have 500 store/item combinations, it needs to run 500 times. Every time we want to forecast the future, say on Jan 2, 2018, we want to forecast next 90 days. We need to re-run ARIMA.