Import Library

In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt, rcParams, style
style.use('seaborn-darkgrid')
import seaborn as sns
sns.set_style('darkgrid')
from plotly import express as px, graph_objects as go

from statsmodels.tsa.deterministic import DeterministicProcess, CalendarFourier
from statsmodels.graphics.tsaplots import plot_pacf
from sklearn.preprocessing import RobustScaler, StandardScaler, Normalizer, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor, BaggingRegressor

import gc
gc.enable()
from warnings import filterwarnings, simplefilter
filterwarnings('ignore')
simplefilter('ignore')
```

In [2]:
```
rcParams['figure.figsize'] = (12, 9) # Konfigurasi jendela figure
```

Fetching dataset

In [3]:
```
train = pd.read_csv('D:\\old data\\Download Folder\\store-sales-time-series-forecasting\\t
                    parse_dates = ['date'], infer_datetime_format = True,
                    dtype = {'store_nbr' : 'category',
                             'family' : 'category'},
                    usecols = ['date', 'store_nbr', 'family', 'sales'])
train['date'] = train.date.dt.to_period('D')
train = train.set_index(['date', 'store_nbr', 'family']).sort_index()
train
```

Out[3]:

			sales
date	**store_nbr**	**family**	
2013-01-01	**1**	**AUTOMOTIVE**	0.000
		BABY CARE	0.000
		BEAUTY	0.000
		BEVERAGES	0.000
		BOOKS	0.000
...	**...**	**...**	...
2017-08-15	**9**	**POULTRY**	438.133
		PREPARED FOODS	154.553
		PRODUCE	2419.729
		SCHOOL AND OFFICE SUPPLIES	121.000

			sales
date	**store_nbr**	**family**	
		SEAFOOD	16.000

3000888 rows × 1 columns

In [4]:
```
test = pd.read_csv('D:\\old data\\Download Folder\\store-sales-time-series-forecasting\\te
                   parse_dates = ['date'], infer_datetime_format = True)
test['date'] = test.date.dt.to_period('D')
test = test.set_index(['date', 'store_nbr', 'family']).sort_values('id')
test
```

Out[4]:

			id	onpromotion
date	**store_nbr**	**family**		
2017-08-16	**1**	**AUTOMOTIVE**	3000888	0
		BABY CARE	3000889	0
		BEAUTY	3000890	2
		BEVERAGES	3000891	20
		BOOKS	3000892	0
...	**...**	**...**	...	...
2017-08-31	**9**	**POULTRY**	3029395	1
		PREPARED FOODS	3029396	0
		PRODUCE	3029397	1
		SCHOOL AND OFFICE SUPPLIES	3029398	9
		SEAFOOD	3029399	0

28512 rows × 2 columns

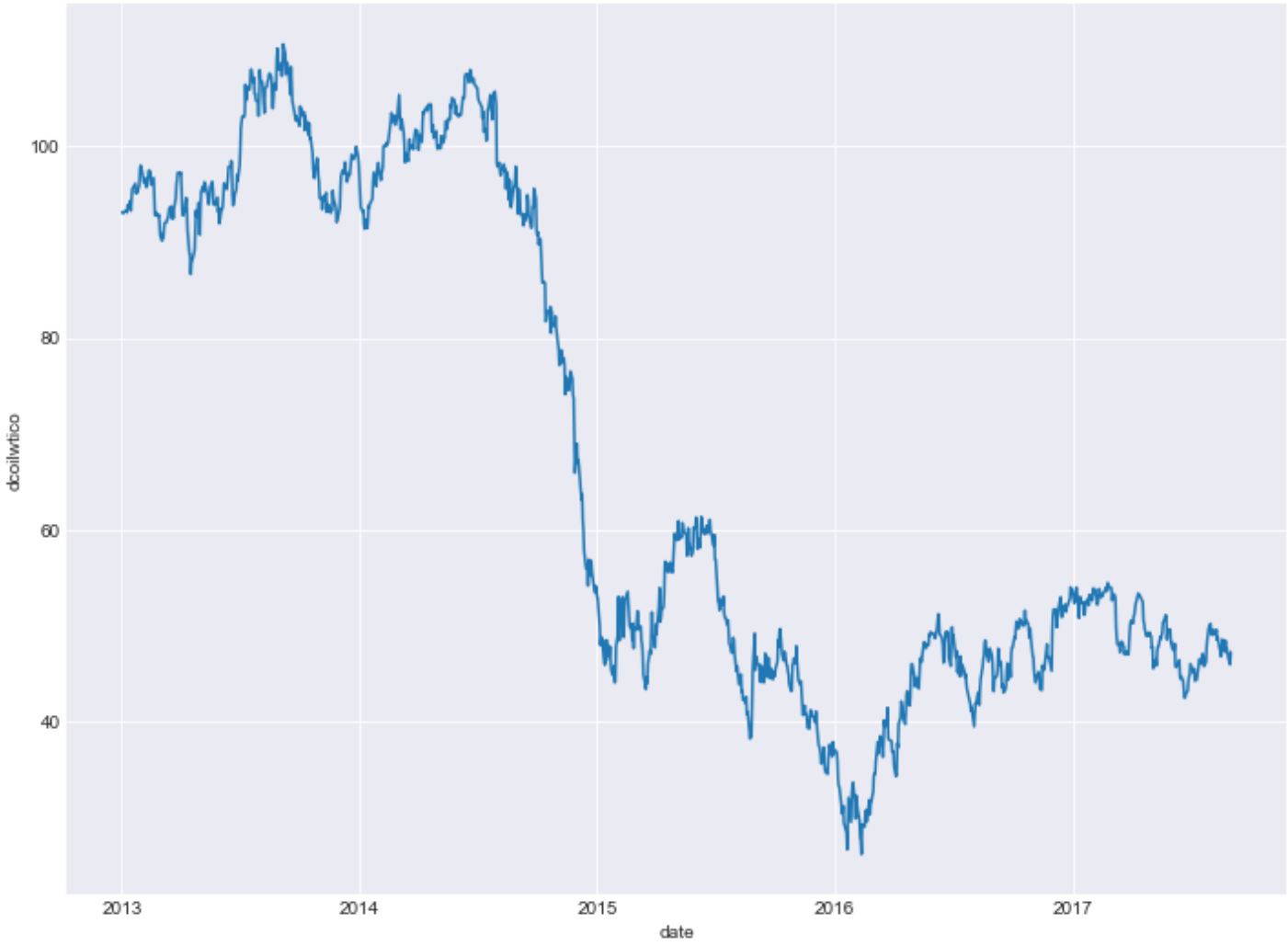
Calendar Engineering

In [5]:
```
calendar = pd.DataFrame(index = pd.date_range('2013-01-01', '2017-08-31')).to_period('D')
oil = pd.read_csv('D:\\old data\\Download Folder\\store-sales-time-series-forecasting\\oil
                  parse_dates = ['date'], infer_datetime_format = True,
                  index_col = 'date').to_period('D')
oil['avg_oil'] = oil['dcoilwtico'].rolling(7).mean()
calendar = calendar.join(oil.avg_oil)
calendar['avg_oil'].fillna(method = 'ffill', inplace = True)
calendar.dropna(inplace = True)
```

We make date in calendar from beginning of train until last date of test.

We also concatenate calendar with oil price.

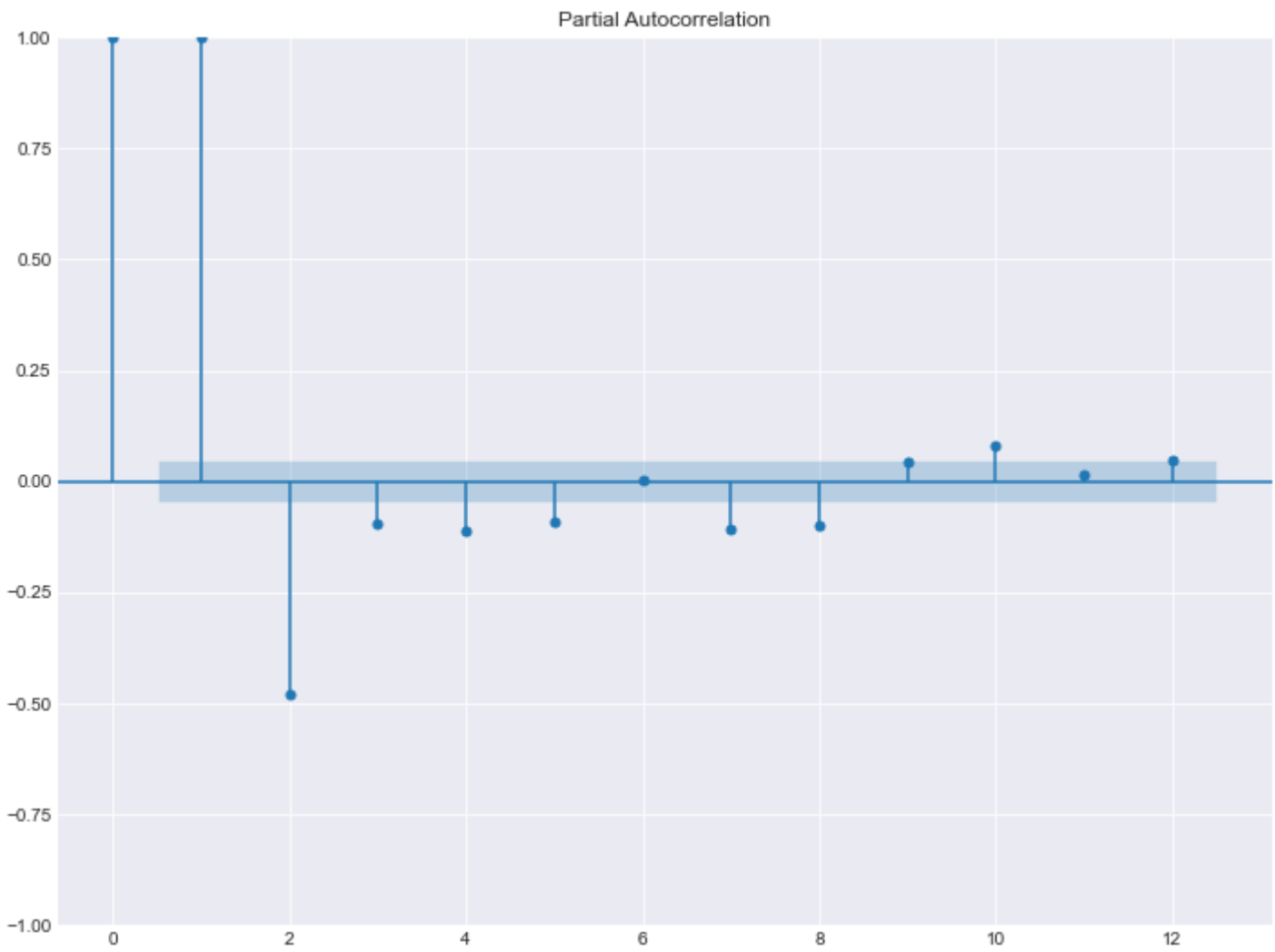
In [6]:
```
# Plotting oil price
_ = sns.lineplot(data = oil.dcoilwtico.to_timestamp())
```



You can see that oil price is only high at 2013 to 2014, however in 2015 it's starting to go down.

So, because we only predict 16 data points we will only need the training data from at least 2015

In [7]:
```
_ = plot_pacf(calendar.avg_oil, lags = 12) # Lagplot oil price (Feature Engineering)
```



You can see that max value for making a lags is up to 5, but you can take whatever you want.

I'm taking 3 lags of oil

Adding lags

In [8]:
```
n_lags = 3
for l in range(1, n_lags + 1) :
    calendar[f'oil_lags{l}'] = calendar.avg_oil.shift(l)
calendar.dropna(inplace = True)
calendar
```

Out[8]:

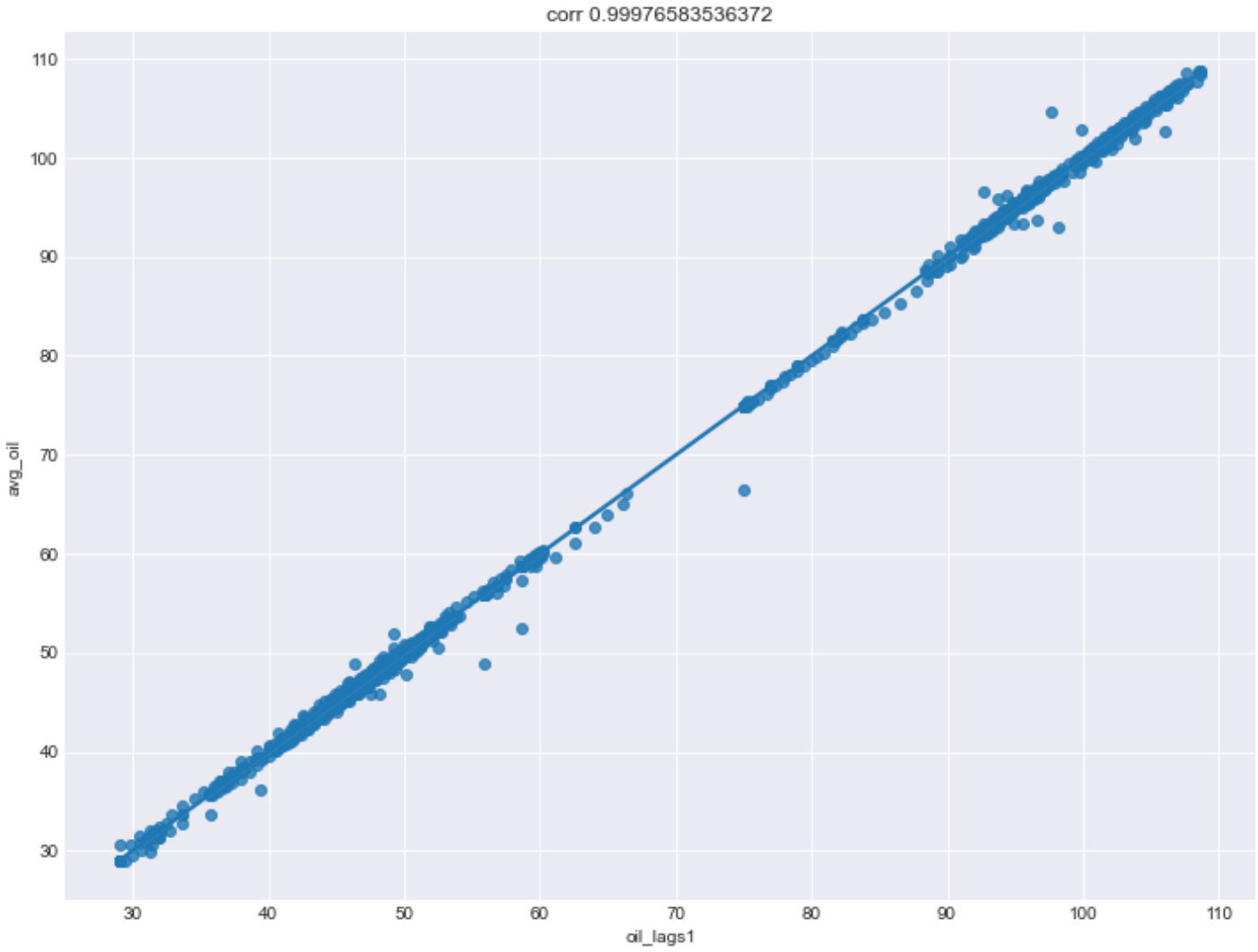
	avg_oil	oil_lags1	oil_lags2	oil_lags3
2013-01-13	93.284286	93.284286	93.284286	93.218571
2013-01-14	93.470000	93.284286	93.284286	93.284286
2013-01-15	93.490000	93.470000	93.284286	93.284286
2013-01-16	93.644286	93.490000	93.470000	93.284286
2013-01-17	93.970000	93.644286	93.490000	93.470000
...	...	...	...	...
2017-08-27	47.720000	47.720000	47.720000	47.598571
2017-08-28	47.624286	47.720000	47.720000	47.720000

	avg_oil	oil_lags1	oil_lags2	oil_lags3
2017-08-29	47.320000	47.624286	47.720000	47.720000
2017-08-30	47.115714	47.320000	47.624286	47.720000
2017-08-31	47.060000	47.115714	47.320000	47.624286

1692 rows × 4 columns

Correlation plot

In [9]:
```
lag = 'oil_lags1'
plt.figure()
sns.regplot(x = calendar[lag], y = calendar.avg_oil)
plt.title(f'corr {calendar.avg_oil.corr(calendar[lag])}')
plt.show()
```



Fetching holiday dataset

In [10]:
```
hol = pd.read_csv('D:\\old data\\Download Folder\\store-sales-time-series-forecasting\\hol
                  parse_dates = ['date'], infer_datetime_format = True,
                  index_col = 'date').to_period('D')
hol = hol[hol.locale == 'National'] # I'm only taking National holiday so there's no false
```

```
hol = hol.groupby(hol.index).first()  # Removing duplicated holiday at the same date
hol
```

Out[10]:

date	type	locale	locale_name	description	transferred
2012-08-10	Holiday	National	Ecuador	Primer Grito de Independencia	False
2012-10-09	Holiday	National	Ecuador	Independencia de Guayaquil	True
2012-10-12	Transfer	National	Ecuador	Traslado Independencia de Guayaquil	False
2012-11-02	Holiday	National	Ecuador	Dia de Difuntos	False
2012-11-03	Holiday	National	Ecuador	Independencia de Cuenca	False
...	...	...	...	...	...
2017-12-22	Additional	National	Ecuador	Navidad-3	False
2017-12-23	Additional	National	Ecuador	Navidad-2	False
2017-12-24	Additional	National	Ecuador	Navidad-1	False
2017-12-25	Holiday	National	Ecuador	Navidad	False
2017-12-26	Additional	National	Ecuador	Navidad+1	False

168 rows × 5 columns

Feature Engineering for holiday

In [11]:

```
calendar = calendar.join(hol)  # Joining calendar with holiday dataset
calendar['dofw'] = calendar.index.dayofweek # Weekly day
calendar['wd'] = 1
calendar.loc[calendar.dofw > 4, 'wd'] = 0 # If it's saturday or sunday then it's not Week
calendar.loc[calendar.type == 'Work Day', 'wd'] = 1 # If it's Work Day event then it's a
calendar.loc[calendar.type == 'Transfer', 'wd'] = 0 # If it's Transfer event then it's no
calendar.loc[calendar.type == 'Bridge', 'wd'] = 0 # If it's Bridge event then it's not a
calendar.loc[(calendar.type == 'Holiday') & (calendar.transferred == False), 'wd'] = 0 #
calendar.loc[(calendar.type == 'Holiday') & (calendar.transferred == True), 'wd'] = 1 # I
calendar = pd.get_dummies(calendar, columns = ['dofw'], drop_first = True) # One-hot enco
calendar = pd.get_dummies(calendar, columns = ['type']) # One-hot encoding for type holid
calendar.drop(['locale', 'locale_name', 'description', 'transferred'], axis = 1, inplace =
calendar
```

Out[11]:

	avg_oil	oil_lags1	oil_lags2	oil_lags3	wd	dofw_1	dofw_2	dofw_3	dofw_4	dofw_5	dofw_6	type_Ac
2013-01-13	93.284286	93.284286	93.284286	93.218571	0	0	0	0	0	0	1	
2013-01-14	93.470000	93.284286	93.284286	93.284286	1	0	0	0	0	0	0	
2013-01-15	93.490000	93.470000	93.284286	93.284286	1	1	0	0	0	0	0	
2013-01-16	93.644286	93.490000	93.470000	93.284286	1	0	1	0	0	0	0	
2013-01-17	93.970000	93.644286	93.490000	93.470000	1	0	0	1	0	0	0	

	avg_oil	oil_lags1	oil_lags2	oil_lags3	wd	dofw_1	dofw_2	dofw_3	dofw_4	dofw_5	dofw_6	type_Ad
...	...	...	...	...	...	...	...	...	...	...	...	
2017-08-27	47.720000	47.720000	47.720000	47.598571	0	0	0	0	0	0	1	
2017-08-28	47.624286	47.720000	47.720000	47.720000	1	0	0	0	0	0	0	
2017-08-29	47.320000	47.624286	47.720000	47.720000	1	1	0	0	0	0	0	
2017-08-30	47.115714	47.320000	47.624286	47.720000	1	0	1	0	0	0	0	
2017-08-31	47.060000	47.115714	47.320000	47.624286	1	0	0	1	0	0	0	

1692 rows × 17 columns

In [12]:
```
# calendar['wd_lag1'] = calendar.wd.shift(1)
# calendar['wd_fore1'] = calendar.wd.shift(-1).fillna(0)
# calendar.dropna(inplace = True)
# calendar
```

Out[12]:

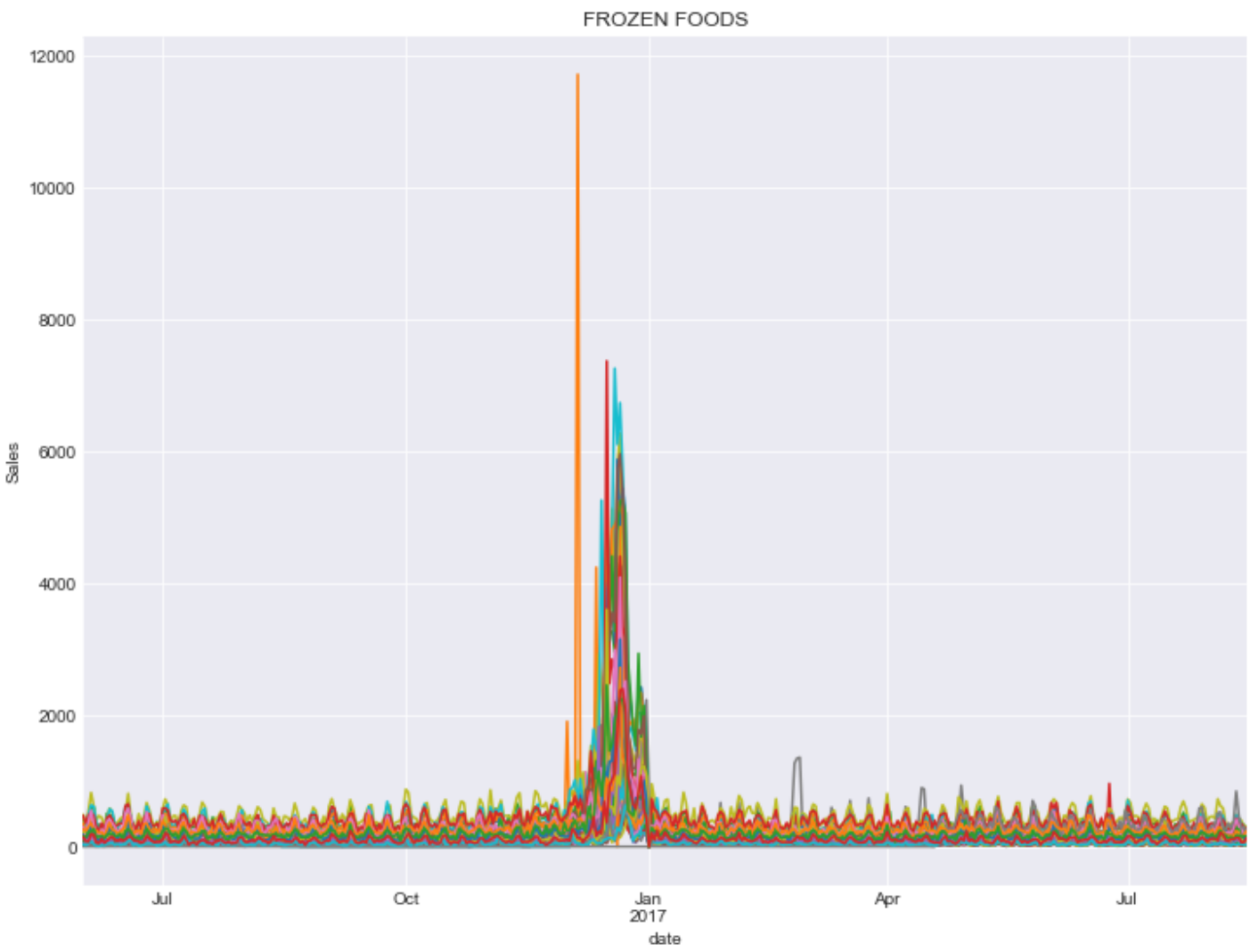
	avg_oil	oil_lags1	oil_lags2	oil_lags3	wd	dofw_1	dofw_2	dofw_3	dofw_4	dofw_5	dofw_6	type_Ad
2013-01-14	93.470000	93.284286	93.284286	93.284286	1	0	0	0	0	0	0	
2013-01-15	93.490000	93.470000	93.284286	93.284286	1	1	0	0	0	0	0	
2013-01-16	93.644286	93.490000	93.470000	93.284286	1	0	1	0	0	0	0	
2013-01-17	93.970000	93.644286	93.490000	93.470000	1	0	0	1	0	0	0	
2013-01-18	94.331429	93.970000	93.644286	93.490000	1	0	0	0	1	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	
2017-08-27	47.720000	47.720000	47.720000	47.598571	0	0	0	0	0	0	1	
2017-08-28	47.624286	47.720000	47.720000	47.720000	1	0	0	0	0	0	0	
2017-08-29	47.320000	47.624286	47.720000	47.720000	1	1	0	0	0	0	0	
2017-08-30	47.115714	47.320000	47.624286	47.720000	1	0	1	0	0	0	0	
2017-08-31	47.060000	47.115714	47.320000	47.624286	1	0	0	1	0	0	0	

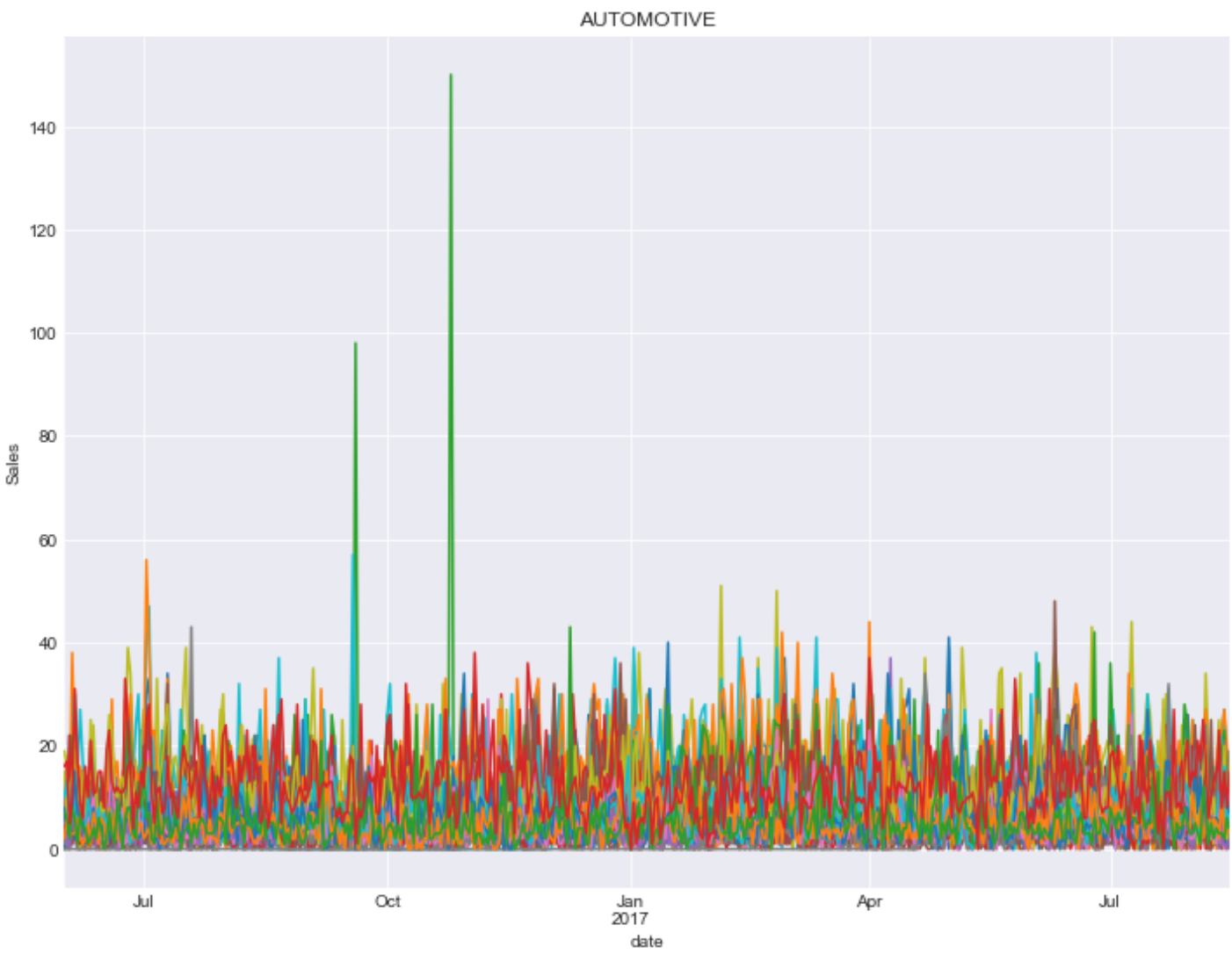
1691 rows × 19 columns

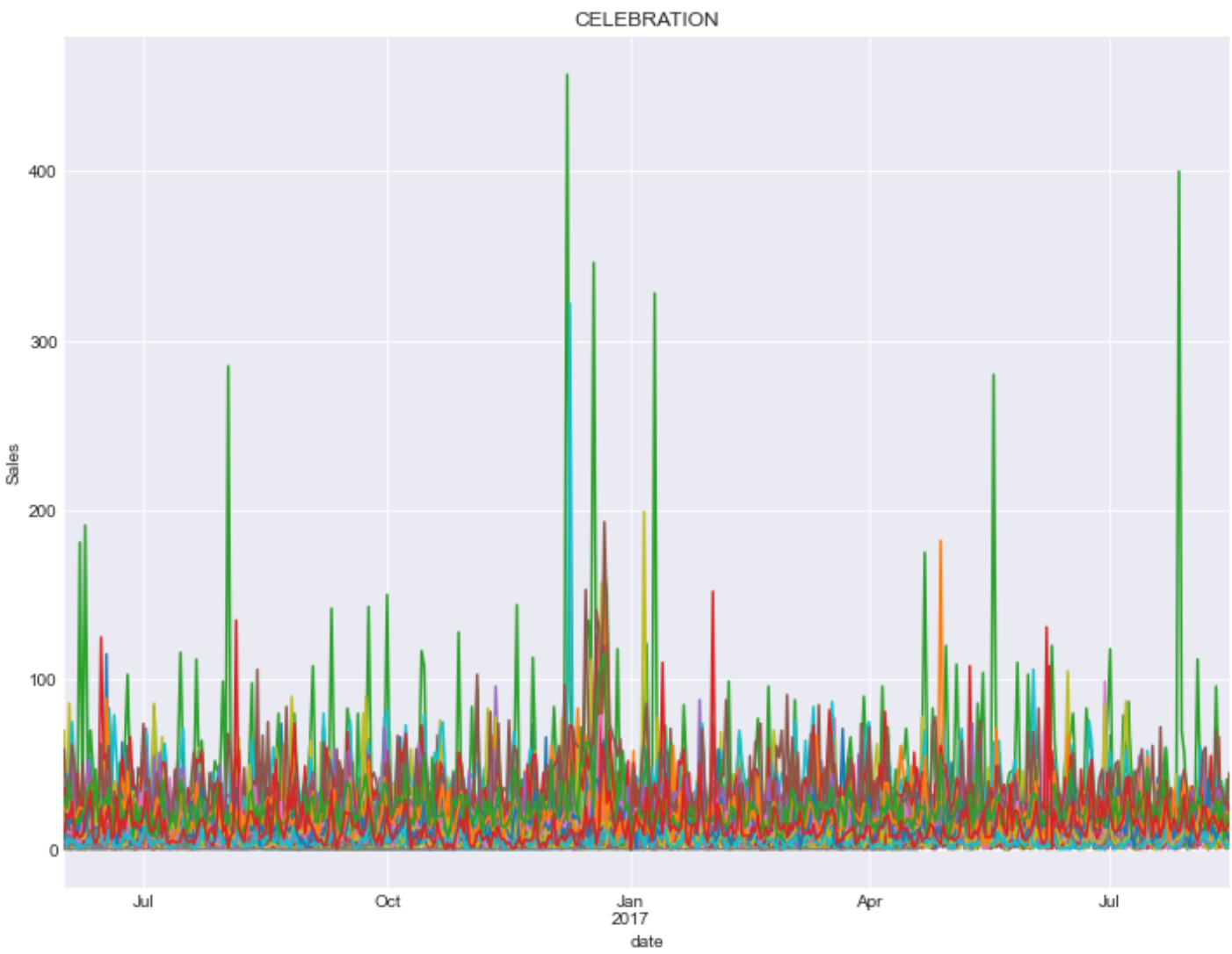
Dependent Variable Viz

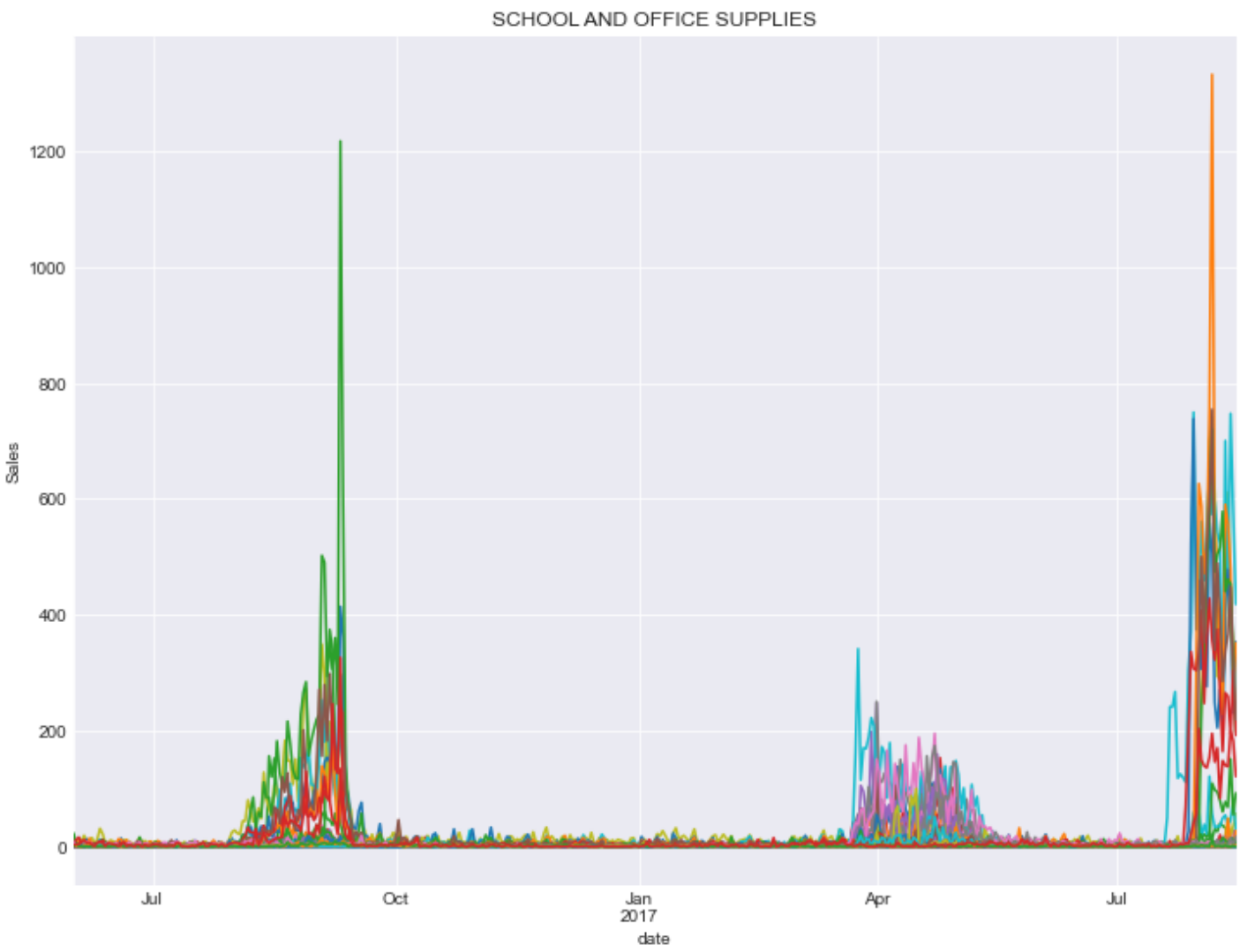
In [13]:

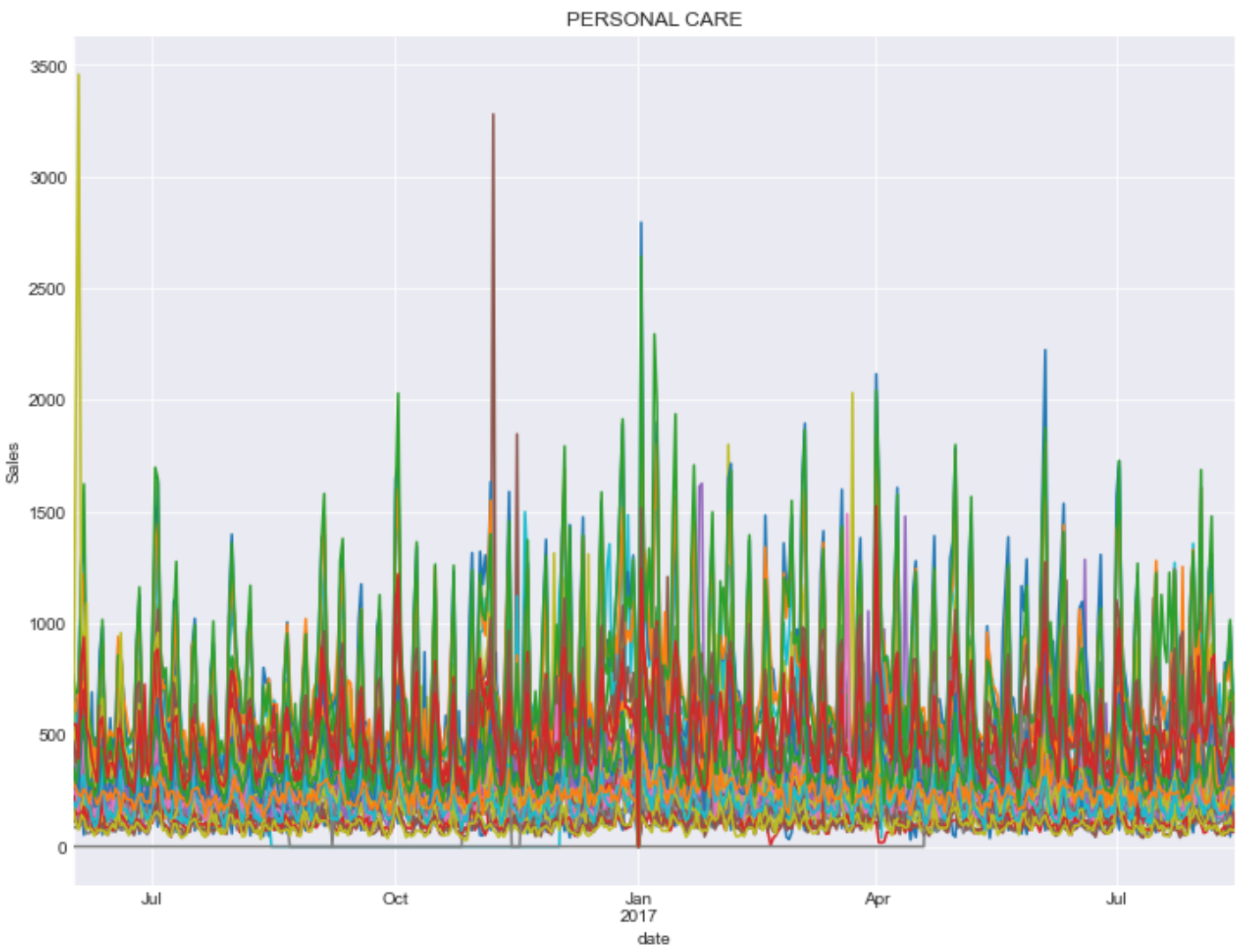
```
y = train.unstack(['store_nbr', 'family']).loc['2016-06':'2017']
family = {c[2] for c in train.index}
for f in family :
    ax = y.loc(axis = 1)['sales', :, f].plot(legend = None)
    ax.set_title(f)
    ax.set_ylabel("Sales")
```

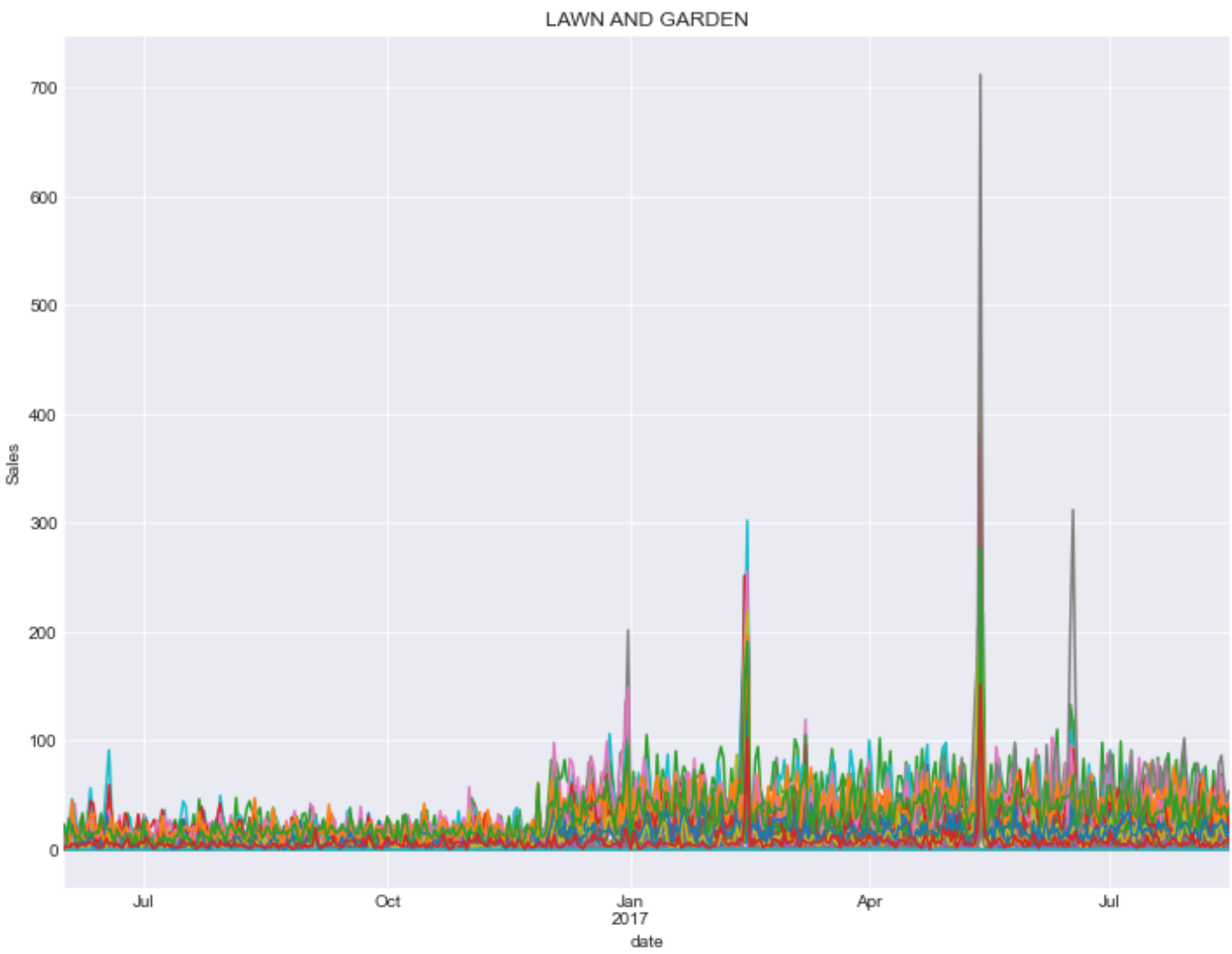

FROZEN FOODS
12000
10000
8000
6000
4000
2000
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

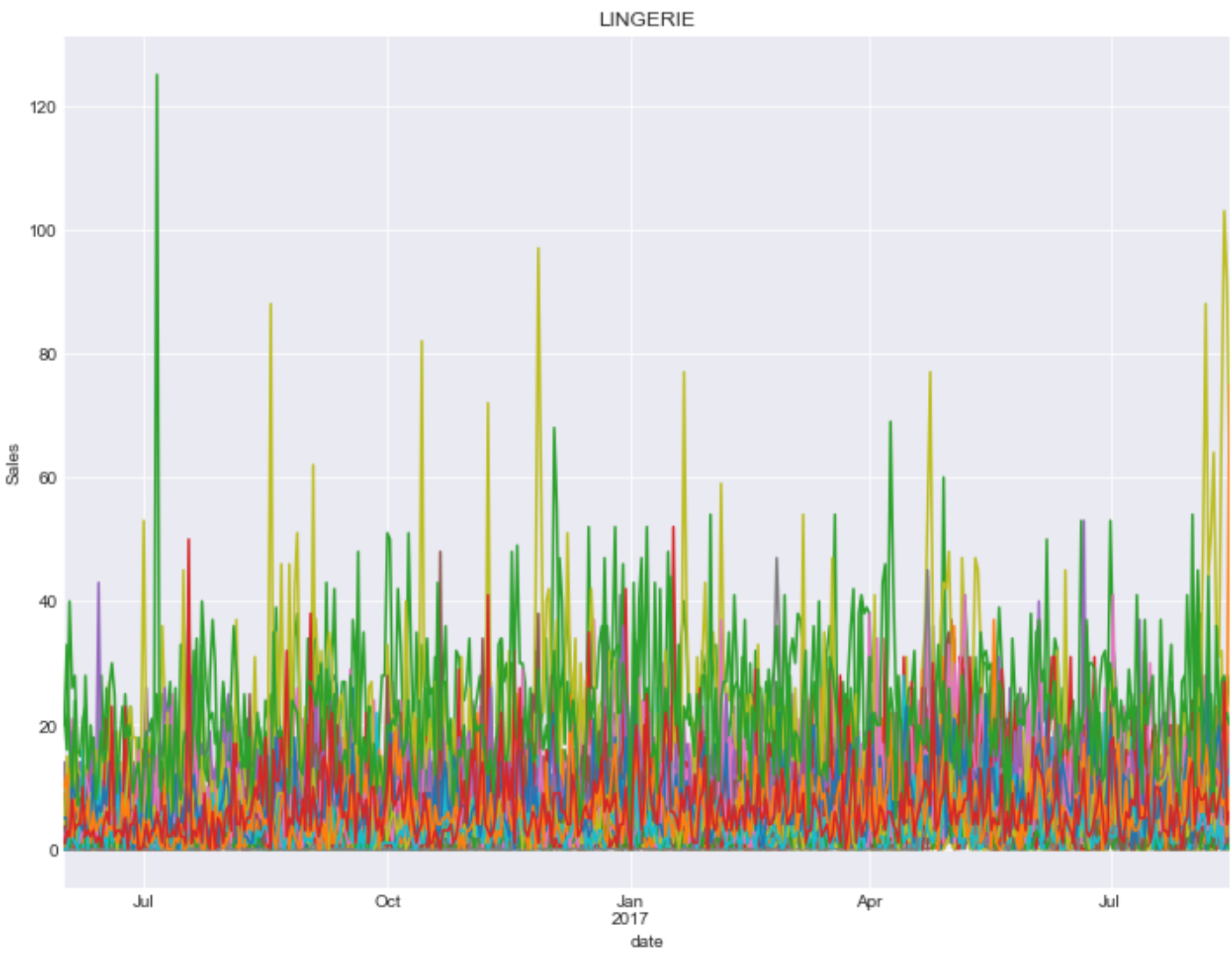

AUTOMOTIVE
Sales
140
120
100
80
60
40
20
0
Jul
Oct
Jan
2017
Apr
Jul
date

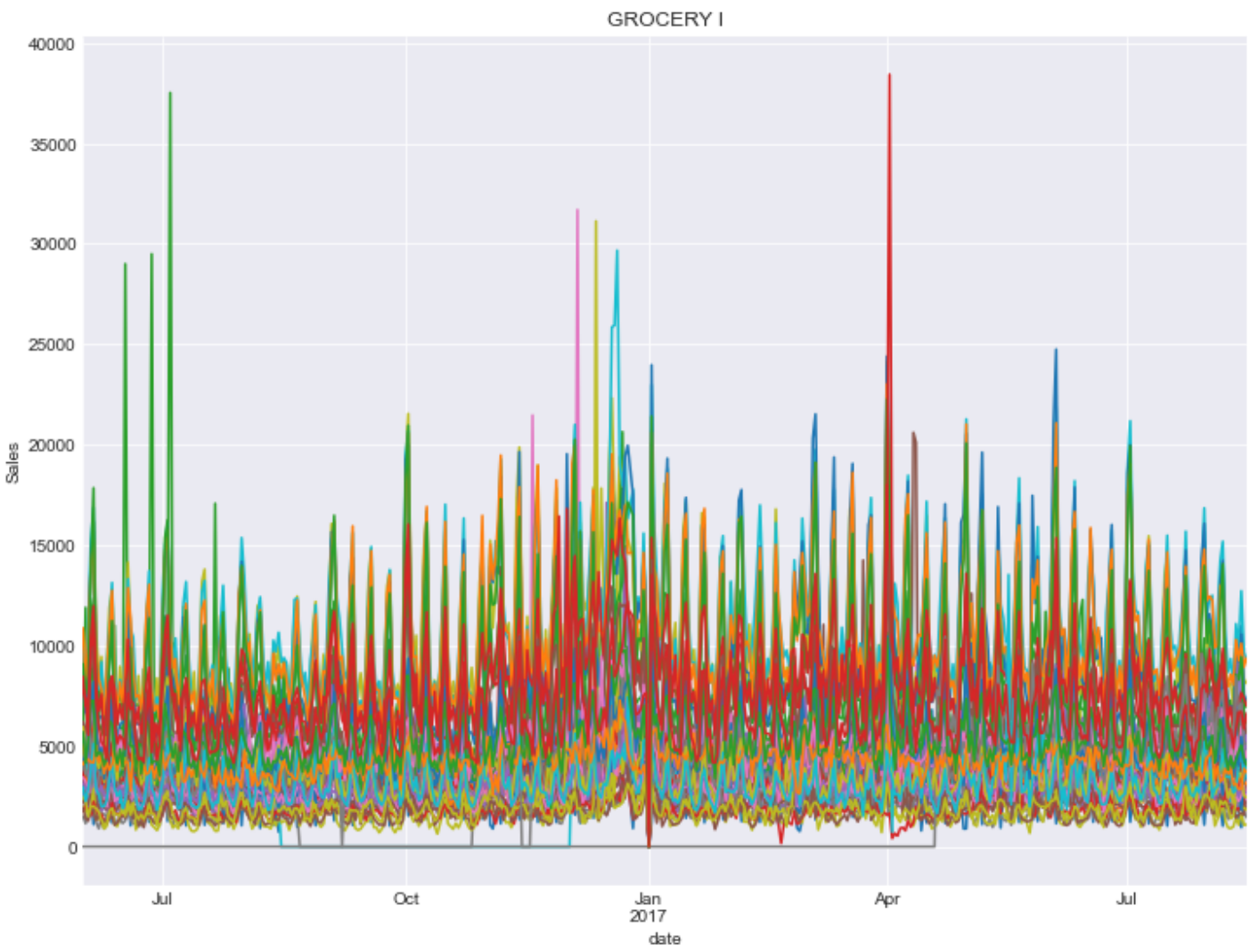

CELEBRATION
Sales
400
300
200
100
0
Jul
Oct
Jan
2017
Apr
Jul
date

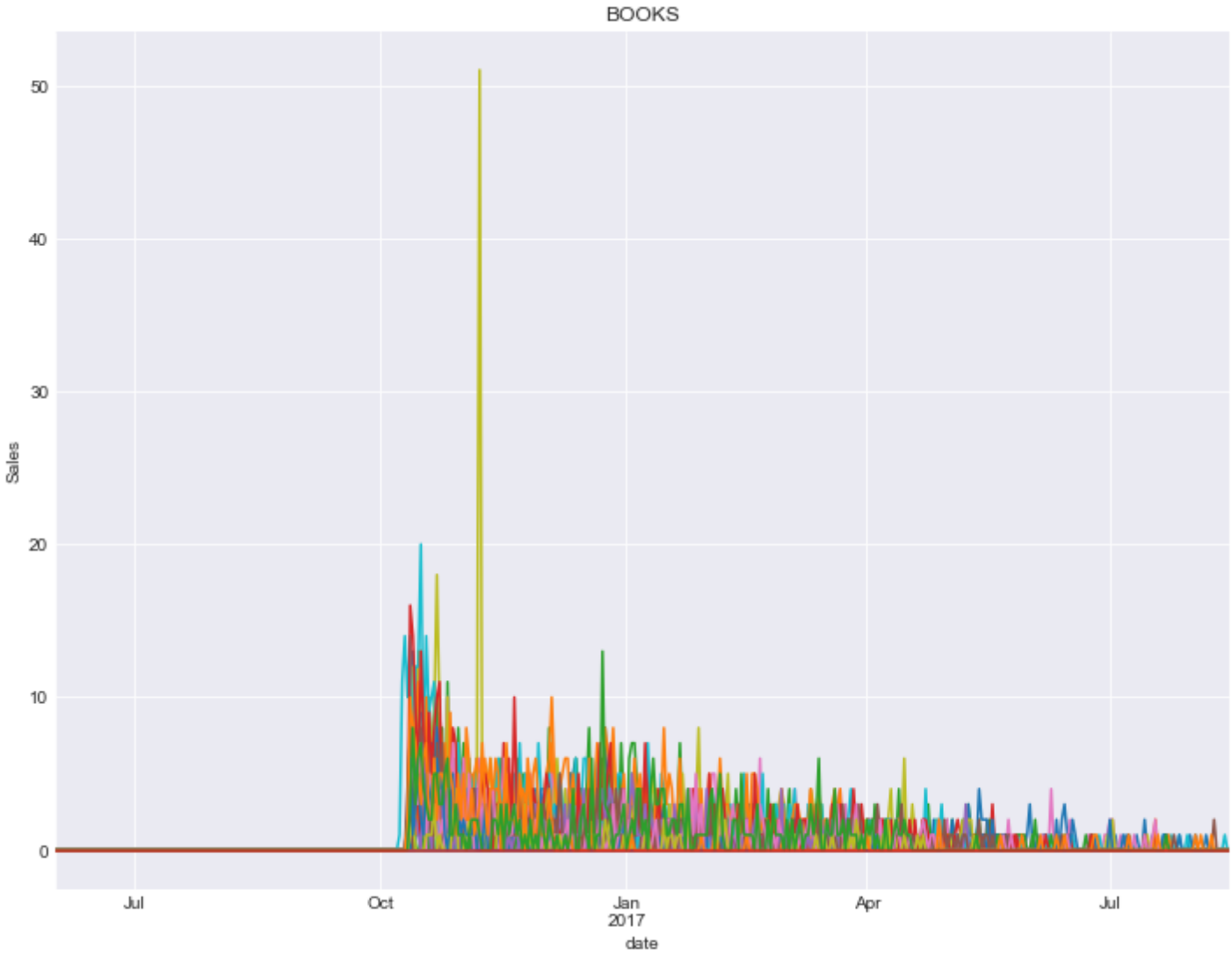

SCHOOL AND OFFICE SUPPLIES
Sales
1200
1000
800
600
400
200
0
Jul
Oct
Jan
2017
Apr
Jul
date

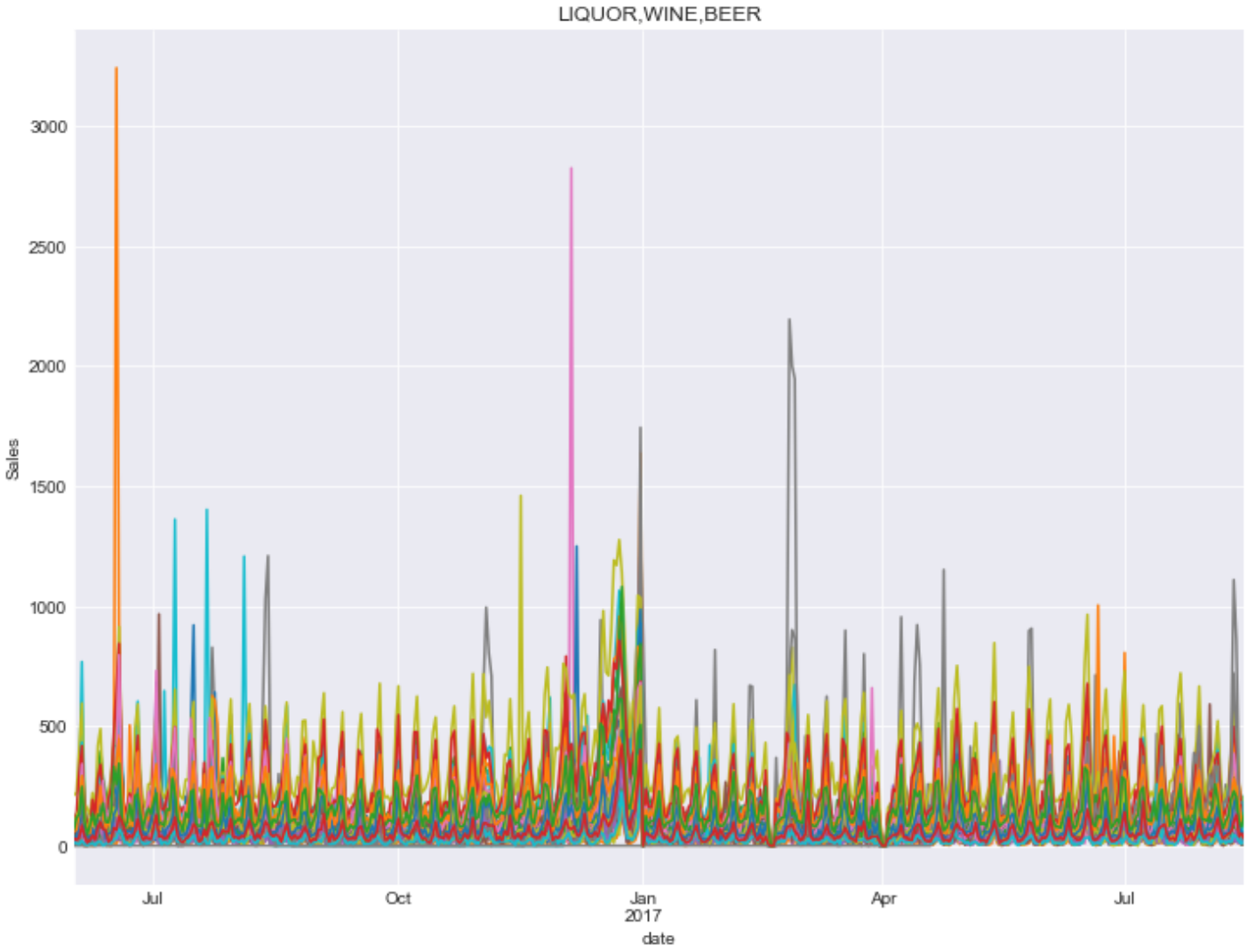

PERSONAL CARE
3500
3000
2500
2000
1500
1000
500
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

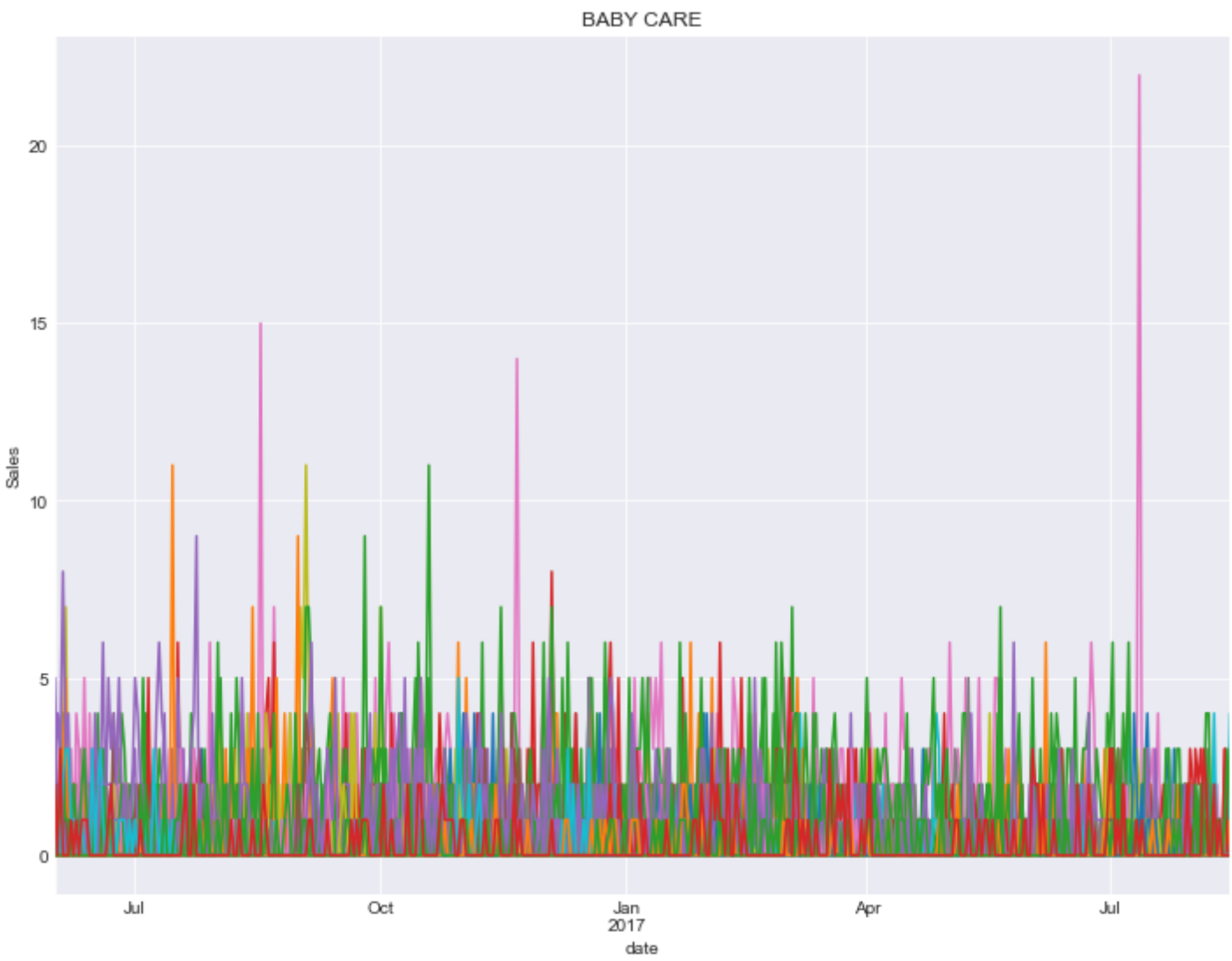

LAWN AND GARDEN
Sales
700
600
500
400
300
200
100
0
Jul
Oct
Jan
2017
Apr
Jul
date

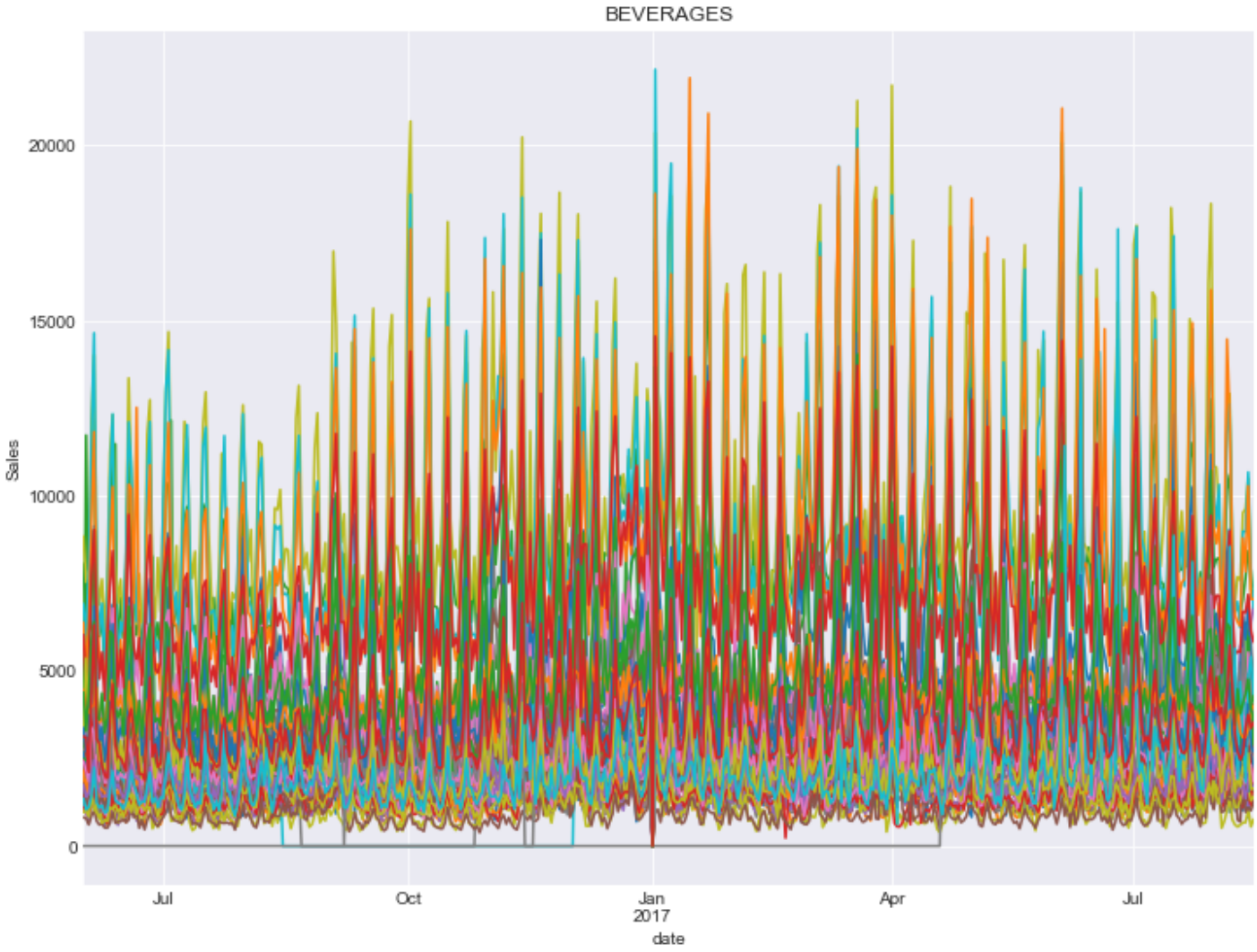

LINGERIE
120
100
80
60
40
20
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

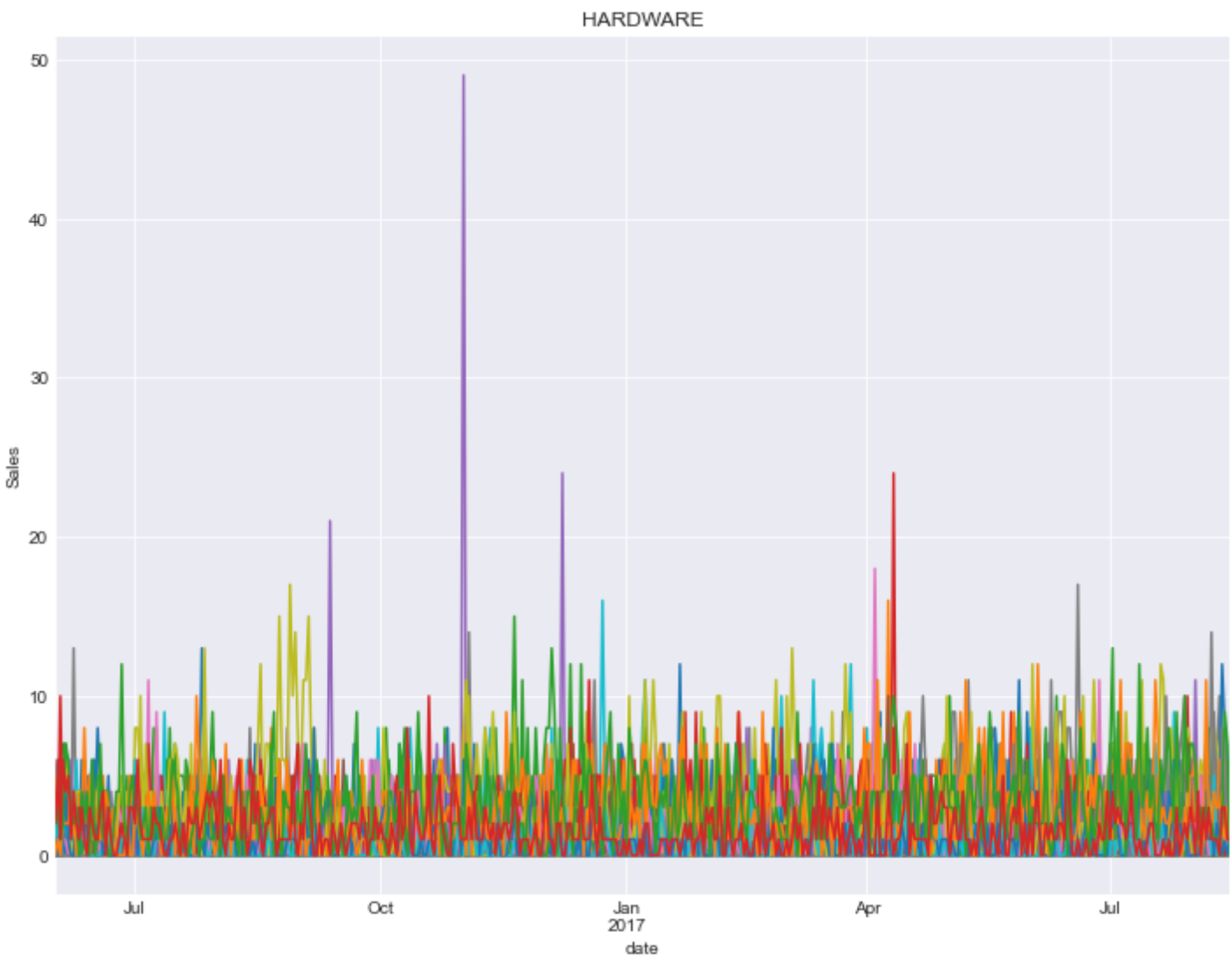

GROCERY I
40000
35000
30000
25000
20000
15000
10000
5000
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

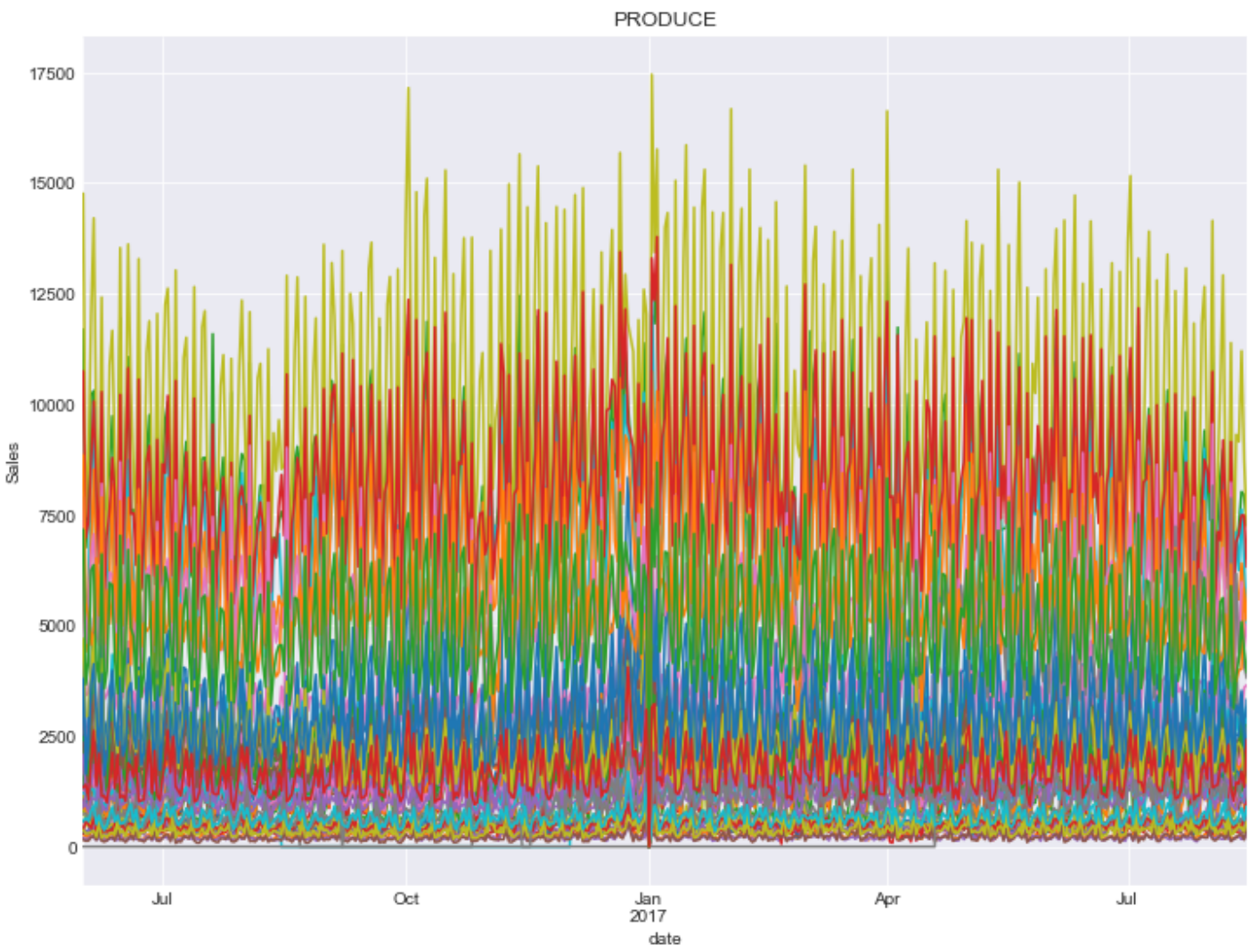

BOOKS
Sales
50
40
30
20
10
0
Jul
Oct
Jan
2017
Apr
Jul
date

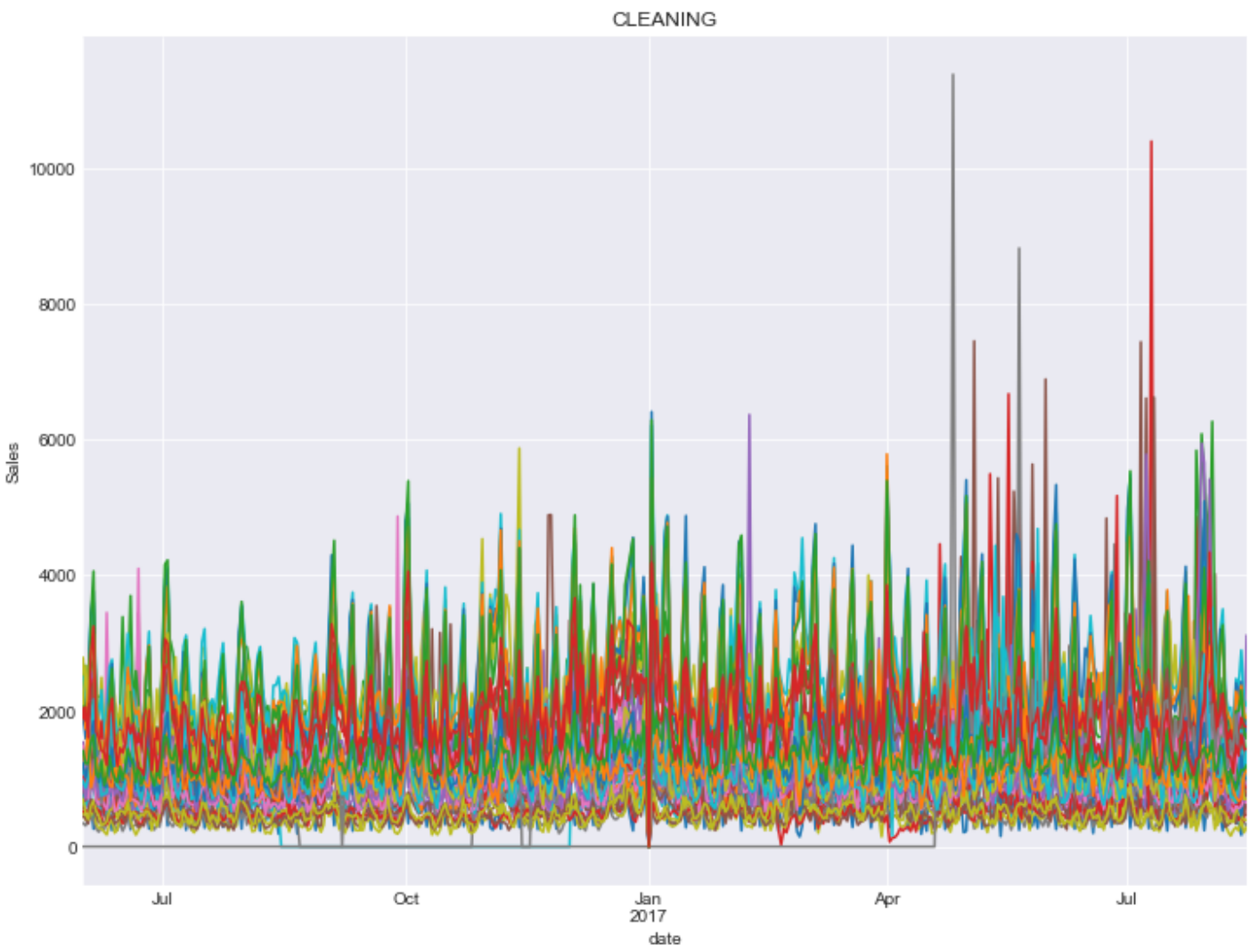

LIQUOR,WINE,BEER
3000
2500
2000
1500
1000
500
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

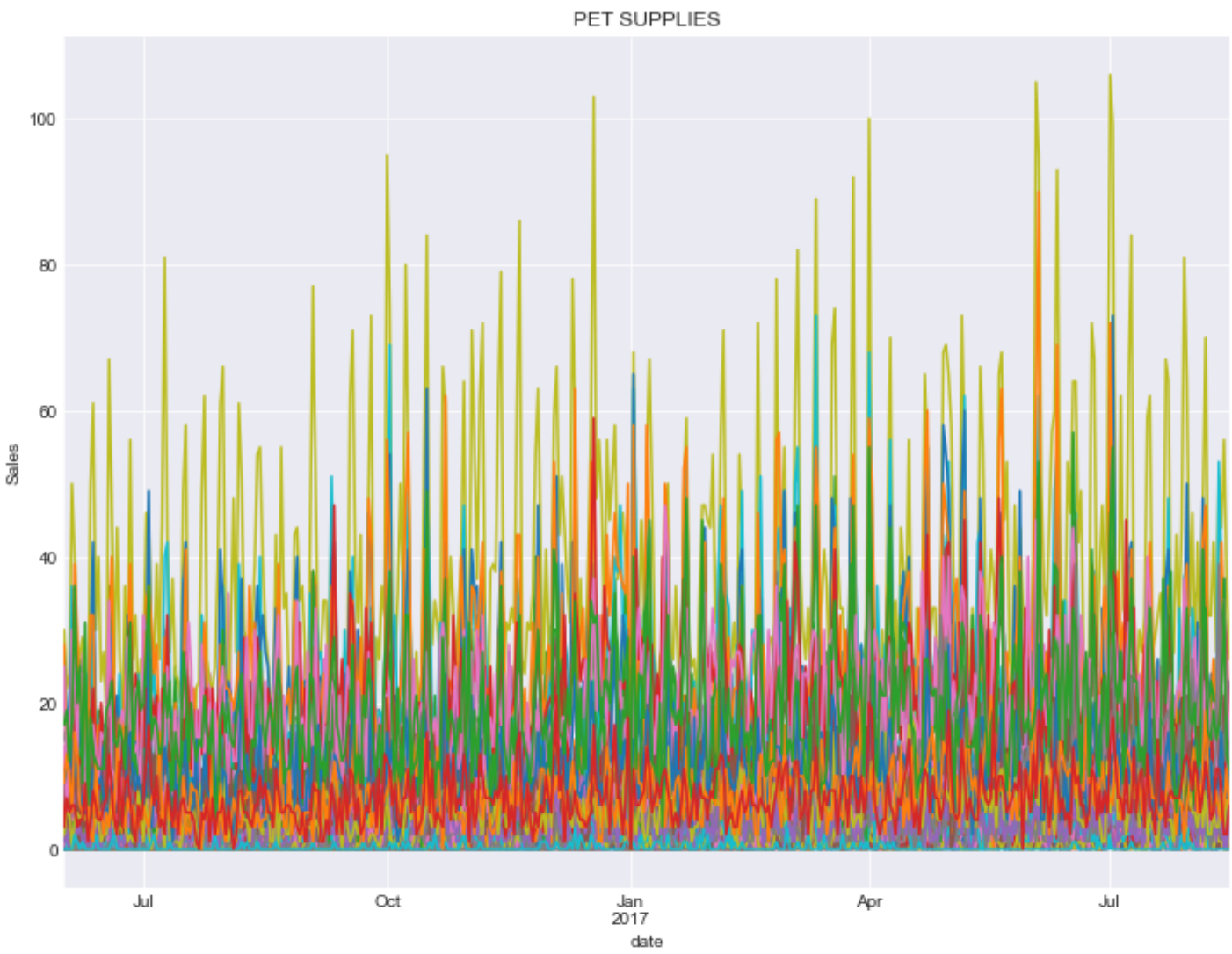

BABY CARE
Sales
20
15
10
5
0
Jul
Oct
Jan
2017
Apr
Jul
date

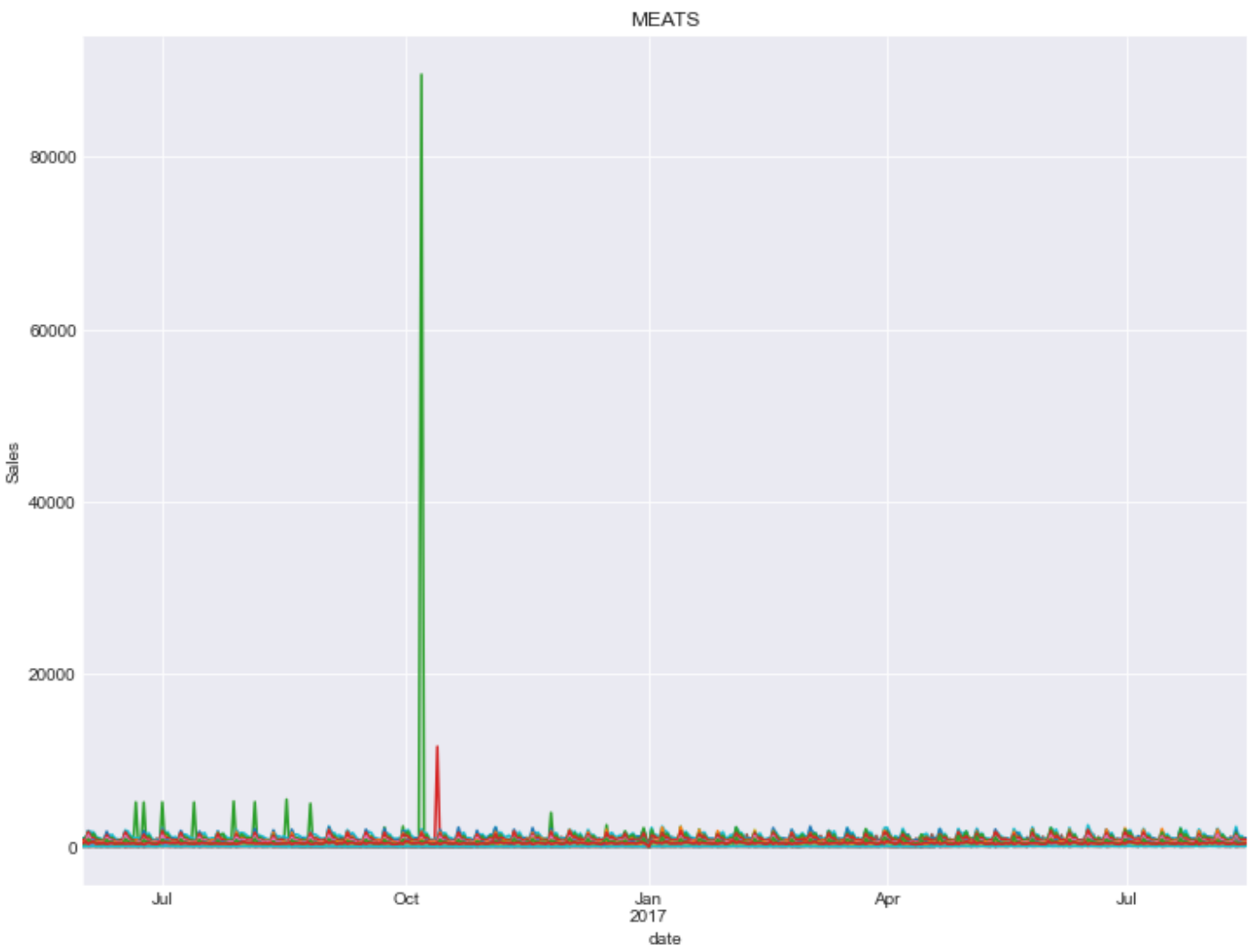

BEVERAGES
20000
15000
10000
5000
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

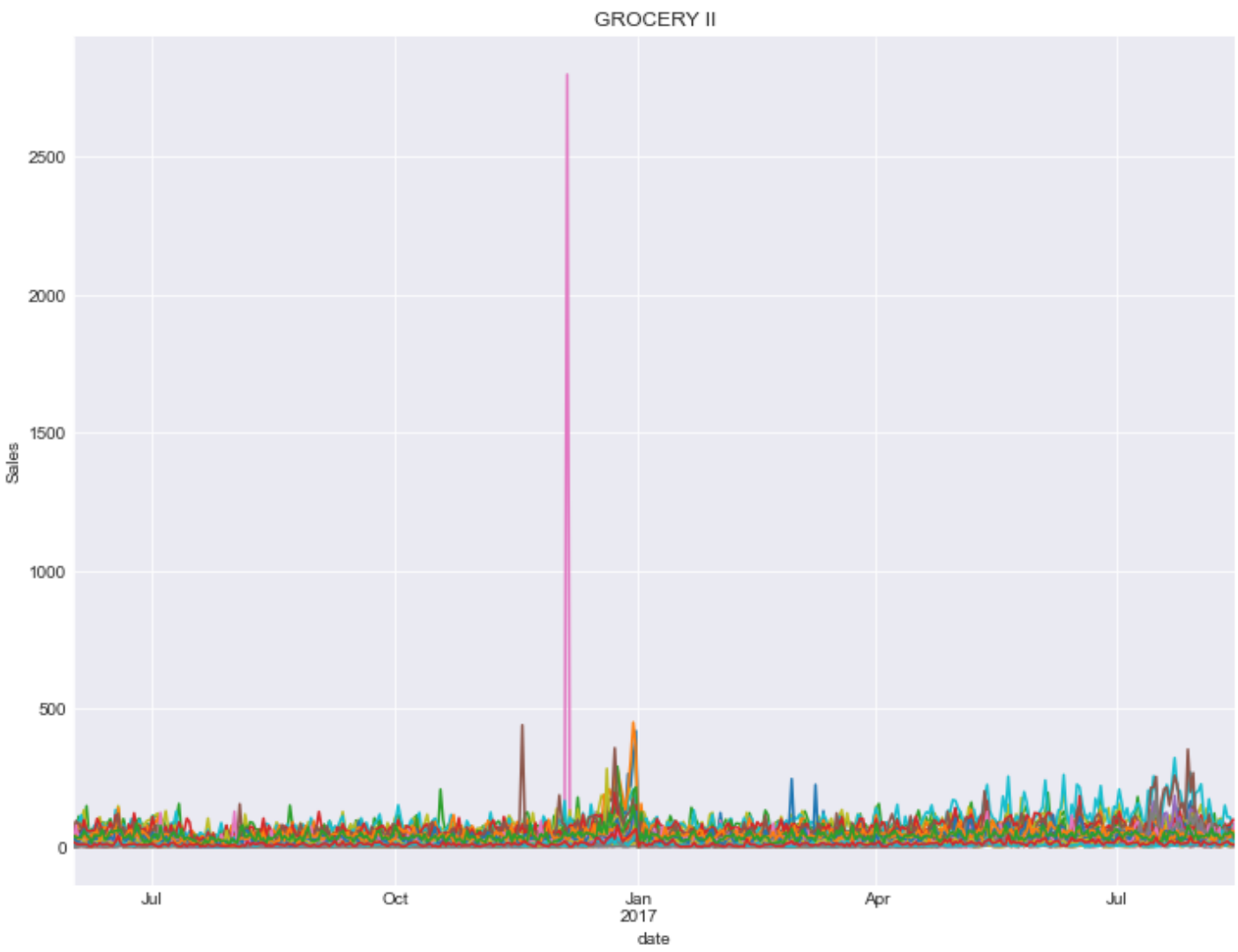

HARDWARE
50
40
30
20
10
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

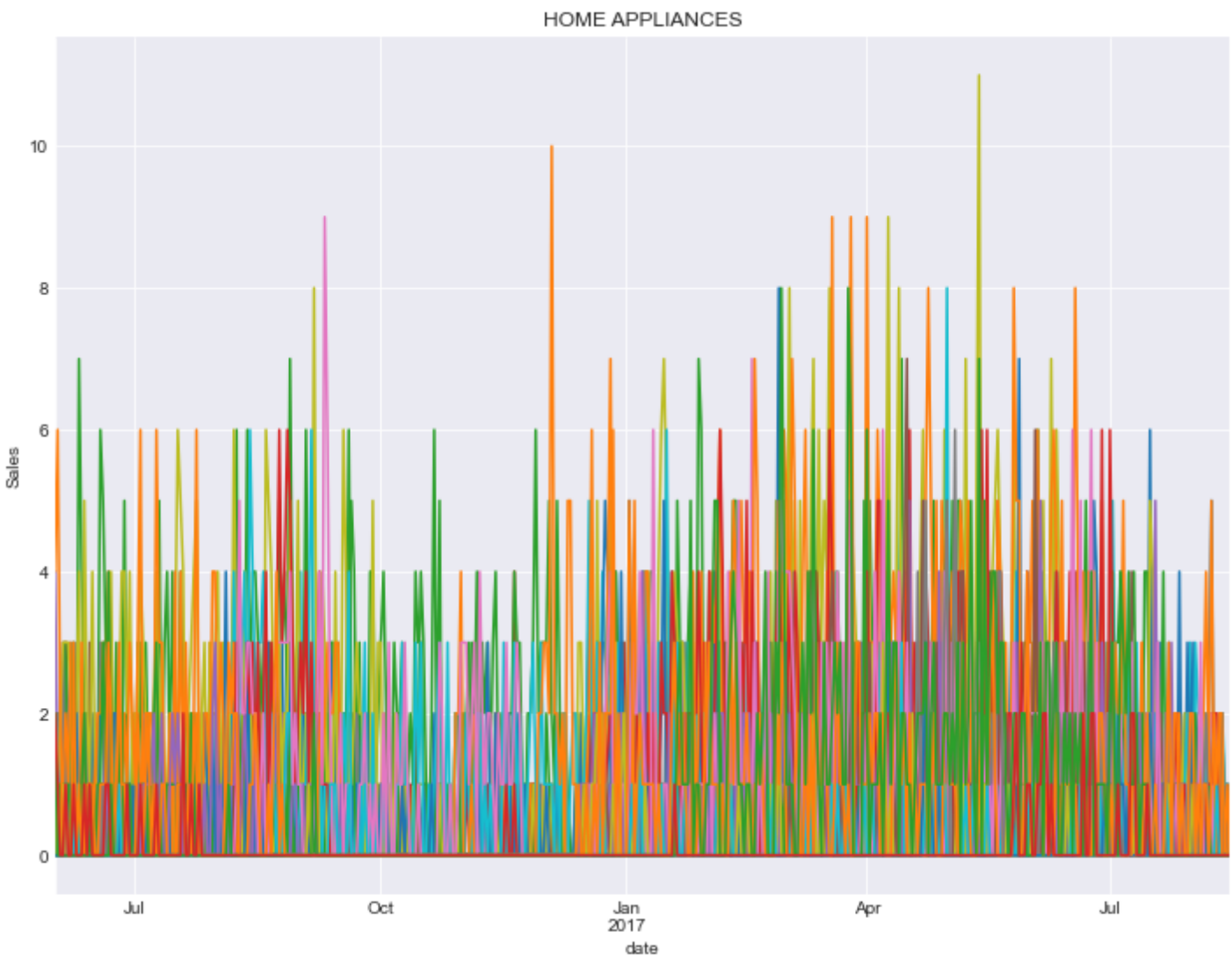

PRODUCE
17500
15000
12500
10000
7500
5000
2500
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

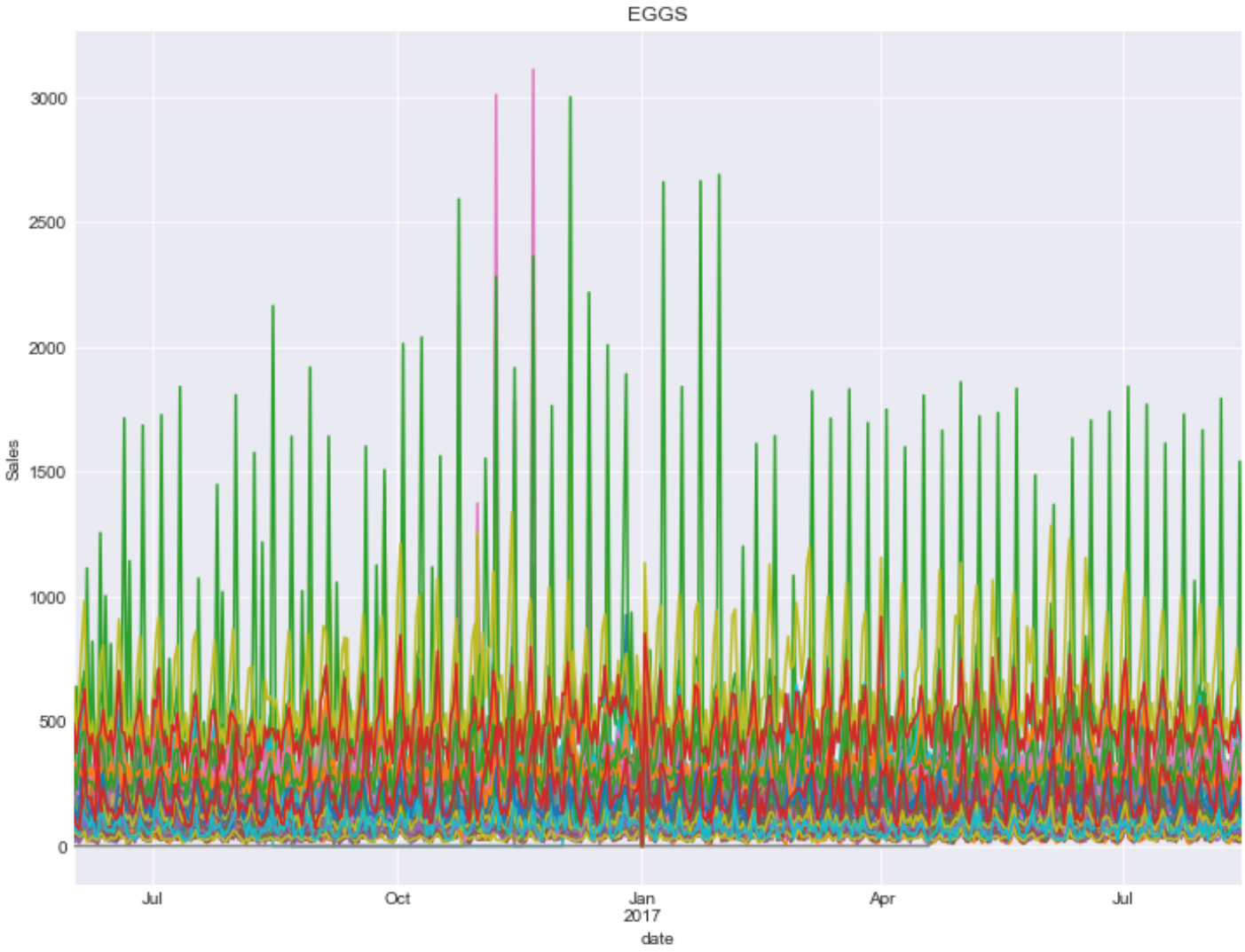

CLEANING
10000
8000
6000
4000
2000
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

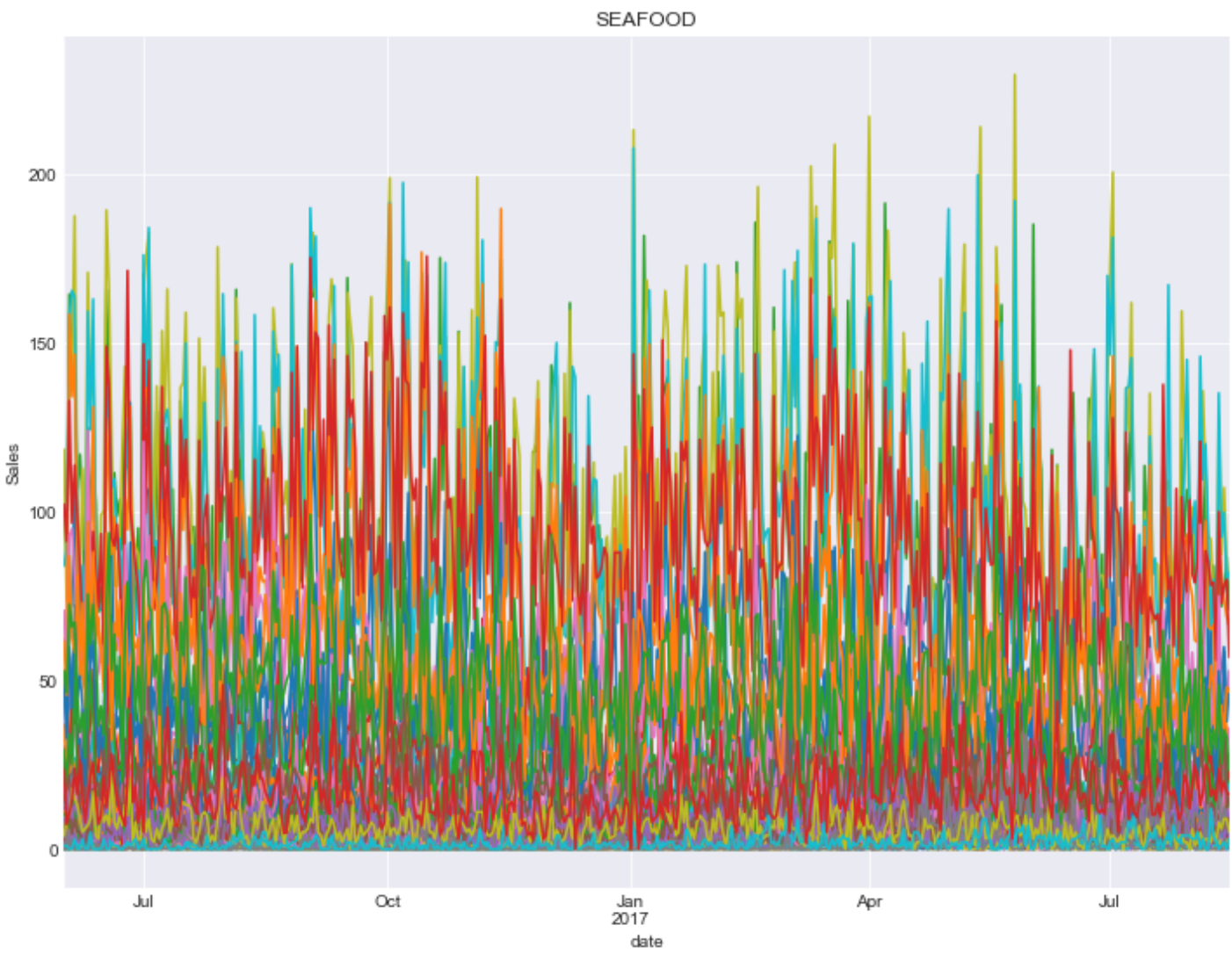

PET SUPPLIES
Sales
100
80
60
40
20
0
Jul
Oct
Jan
2017
Apr
Jul
date

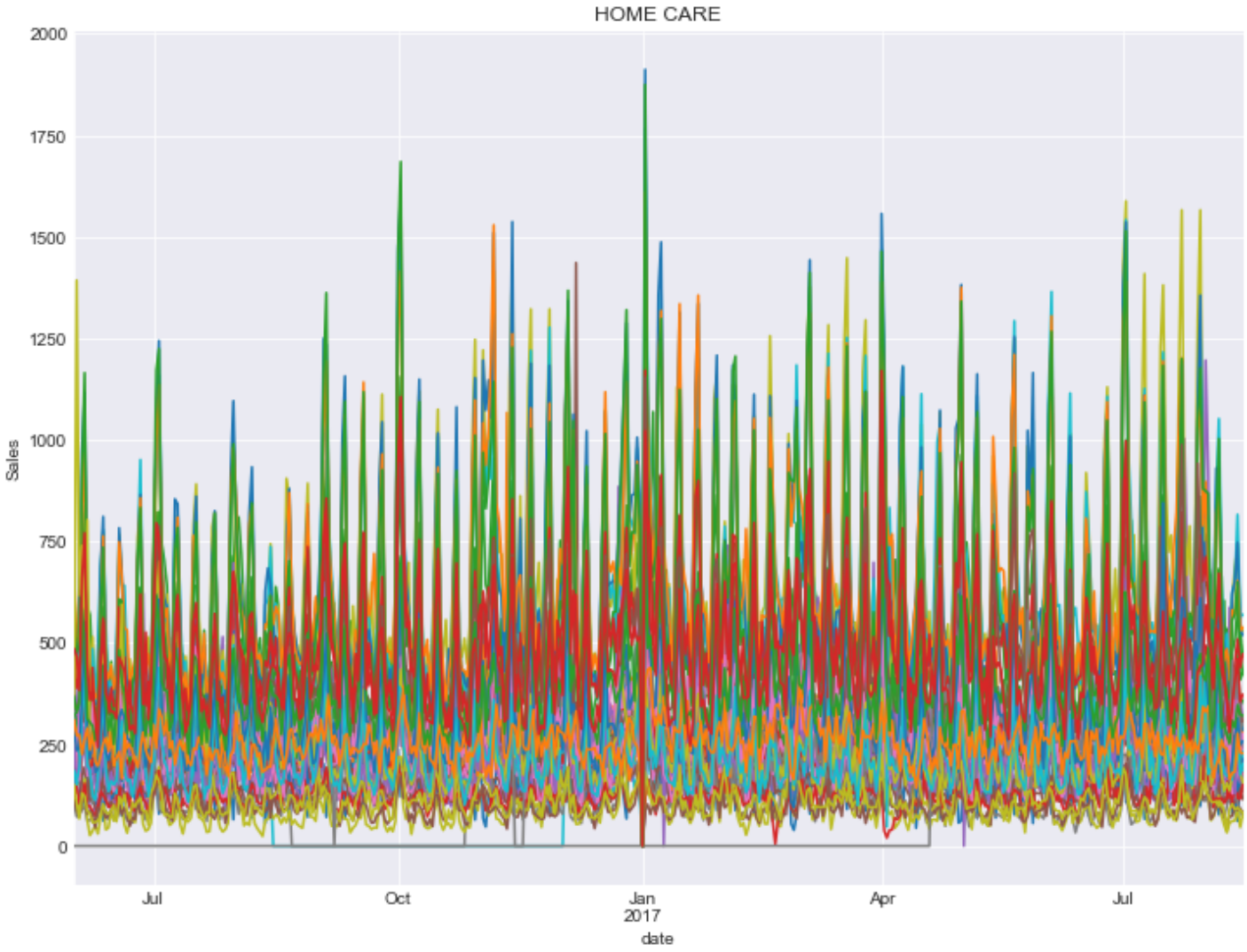

MEATS
80000
60000
40000
20000
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

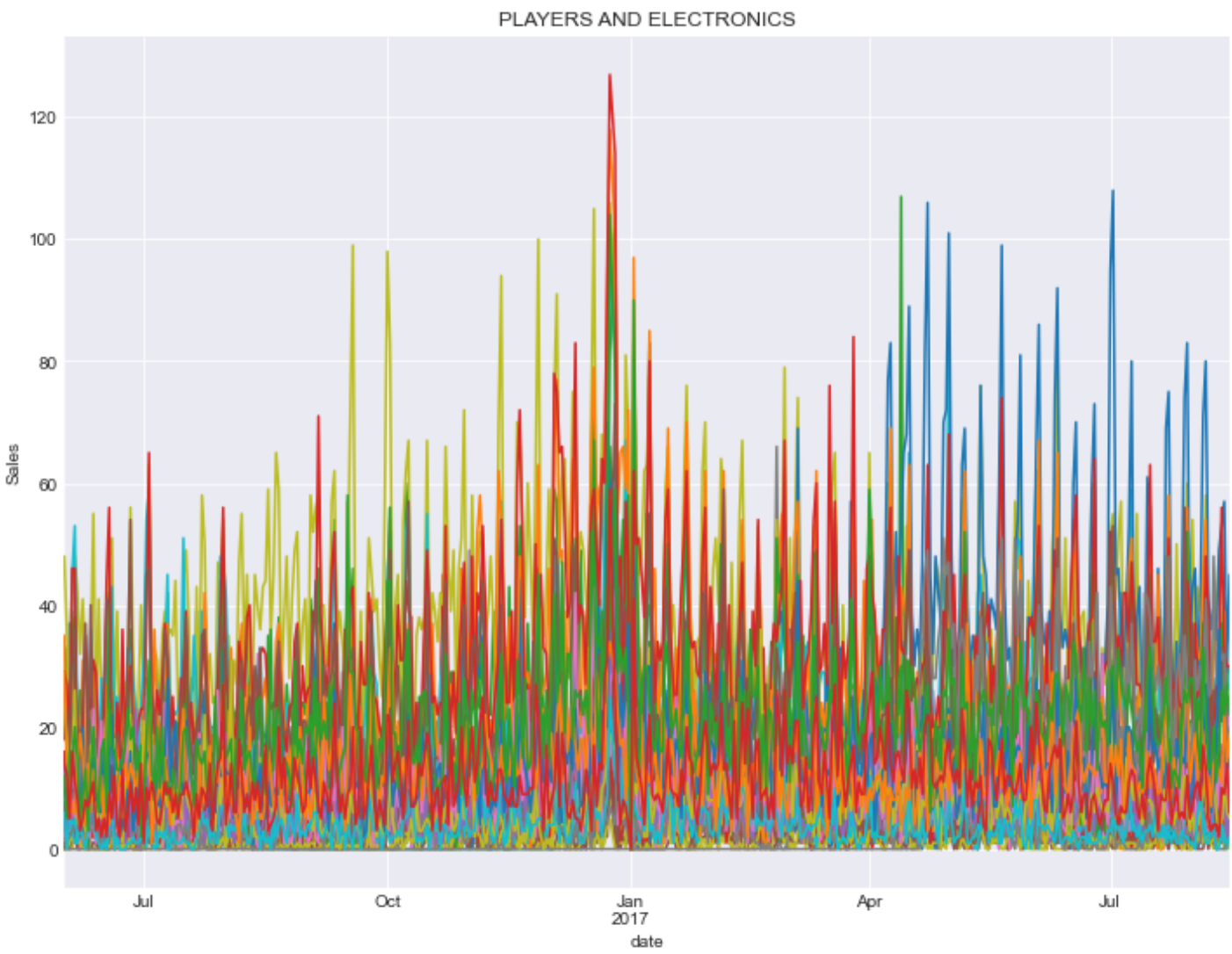

GROCERY II
2500
2000
1500
1000
500
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

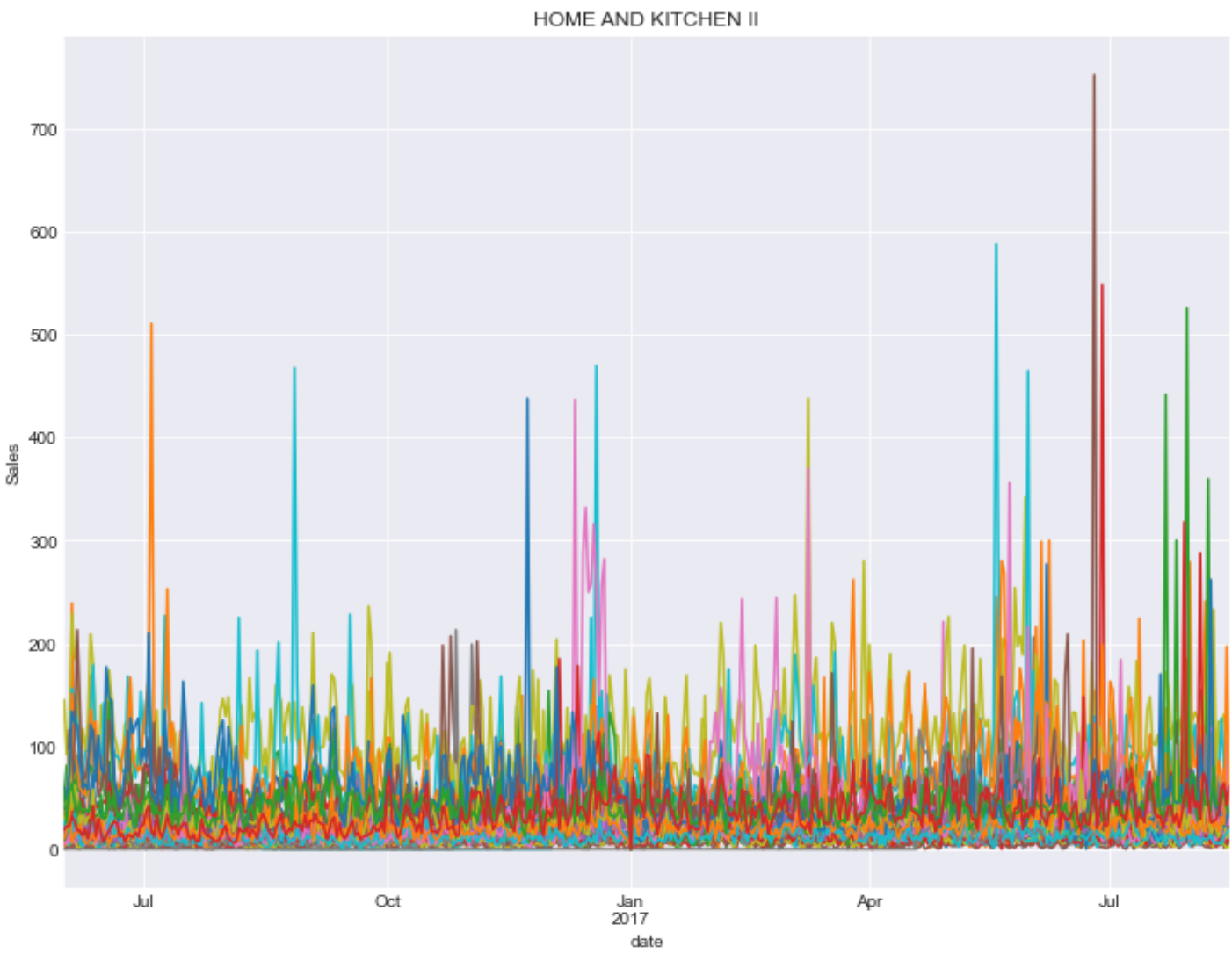

HOME APPLIANCES
Sales
10
8
6
4
2
0
Jul
Oct
Jan
2017
Apr
Jul
date

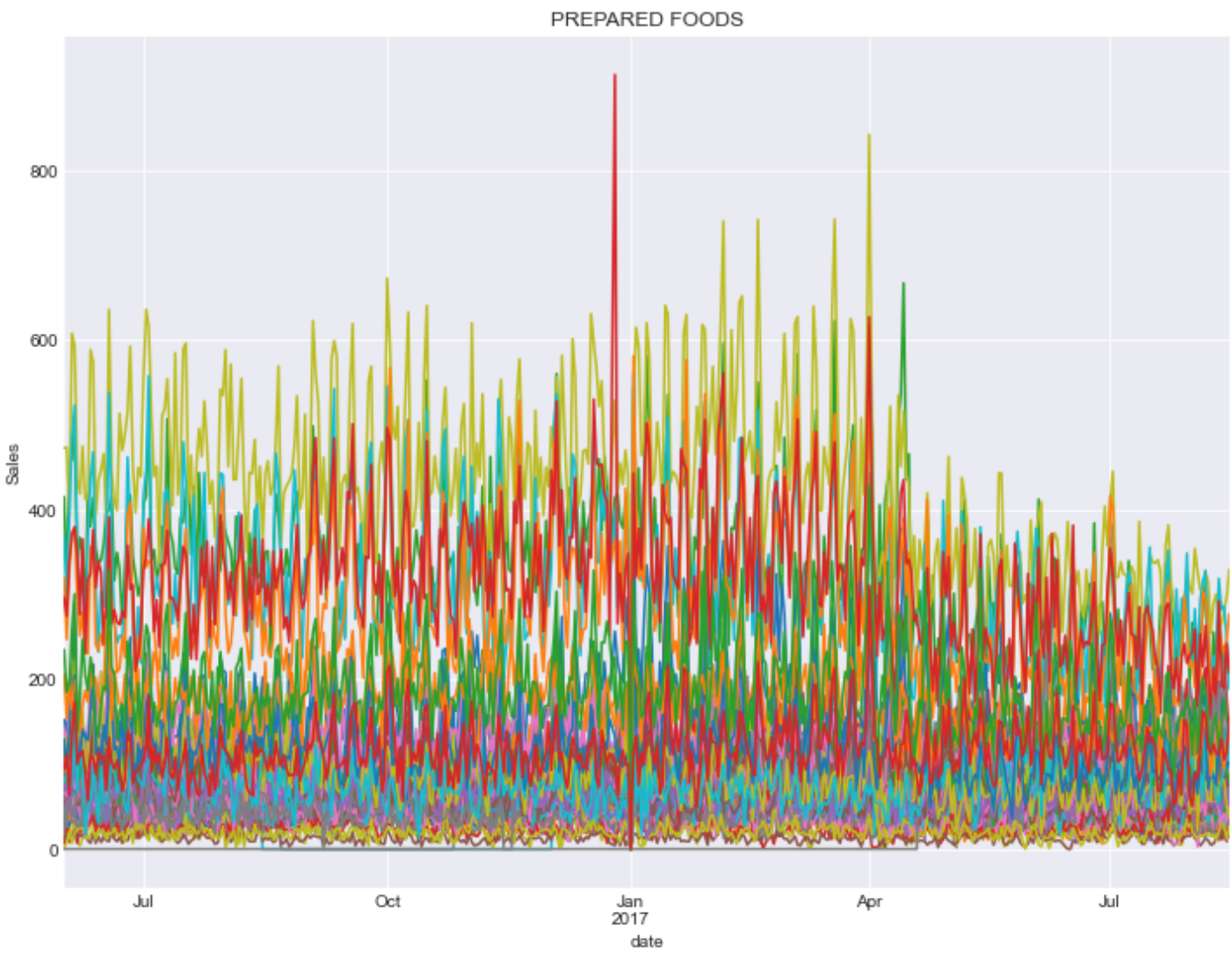

EGGS
3000
2500
2000
1500
1000
500
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

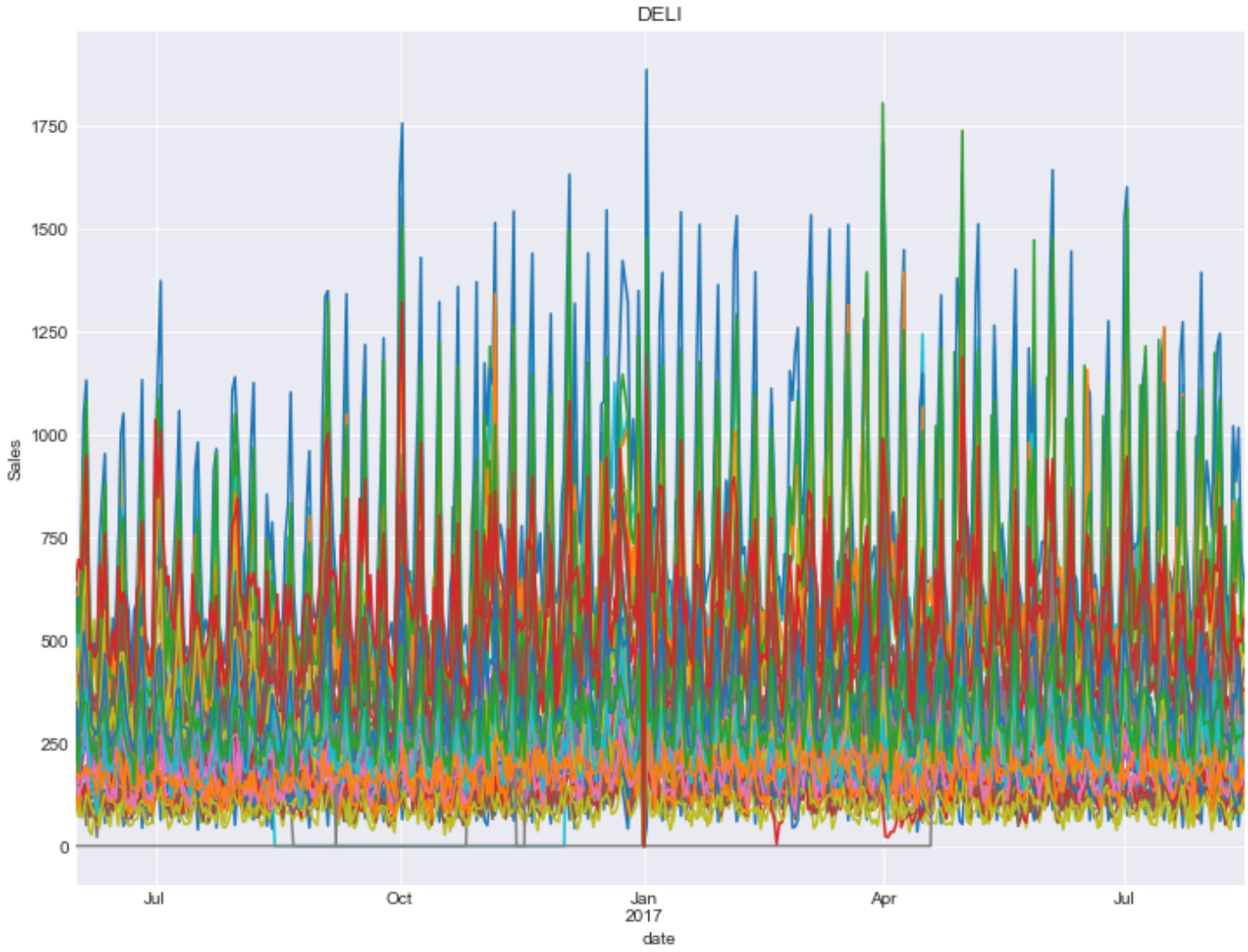

SEAFOOD
Sales
200
150
100
50
0
Jul
Oct
Jan
2017
Apr
Jul
date

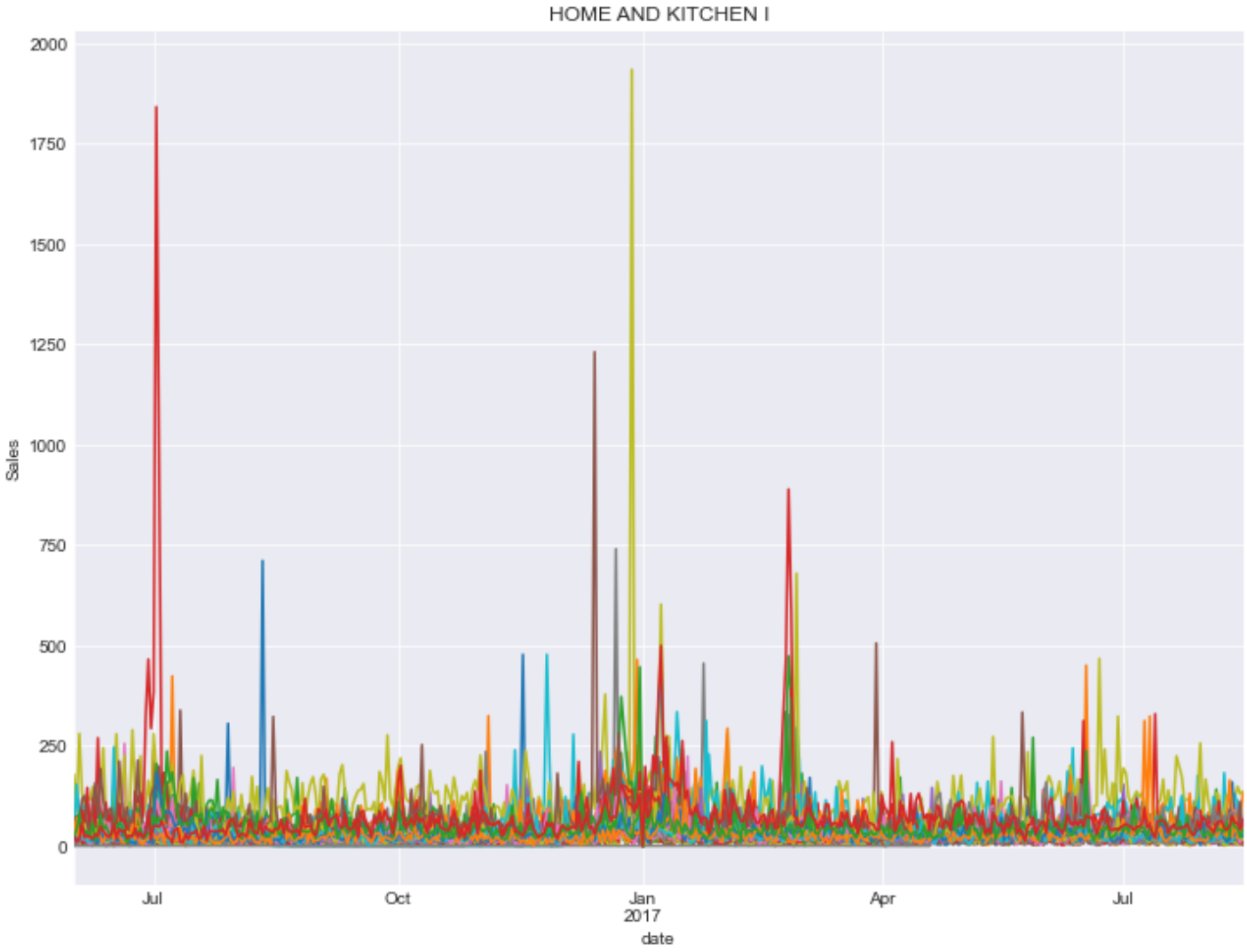

HOME CARE
2000
1750
1500
1250
1000
750
500
250
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

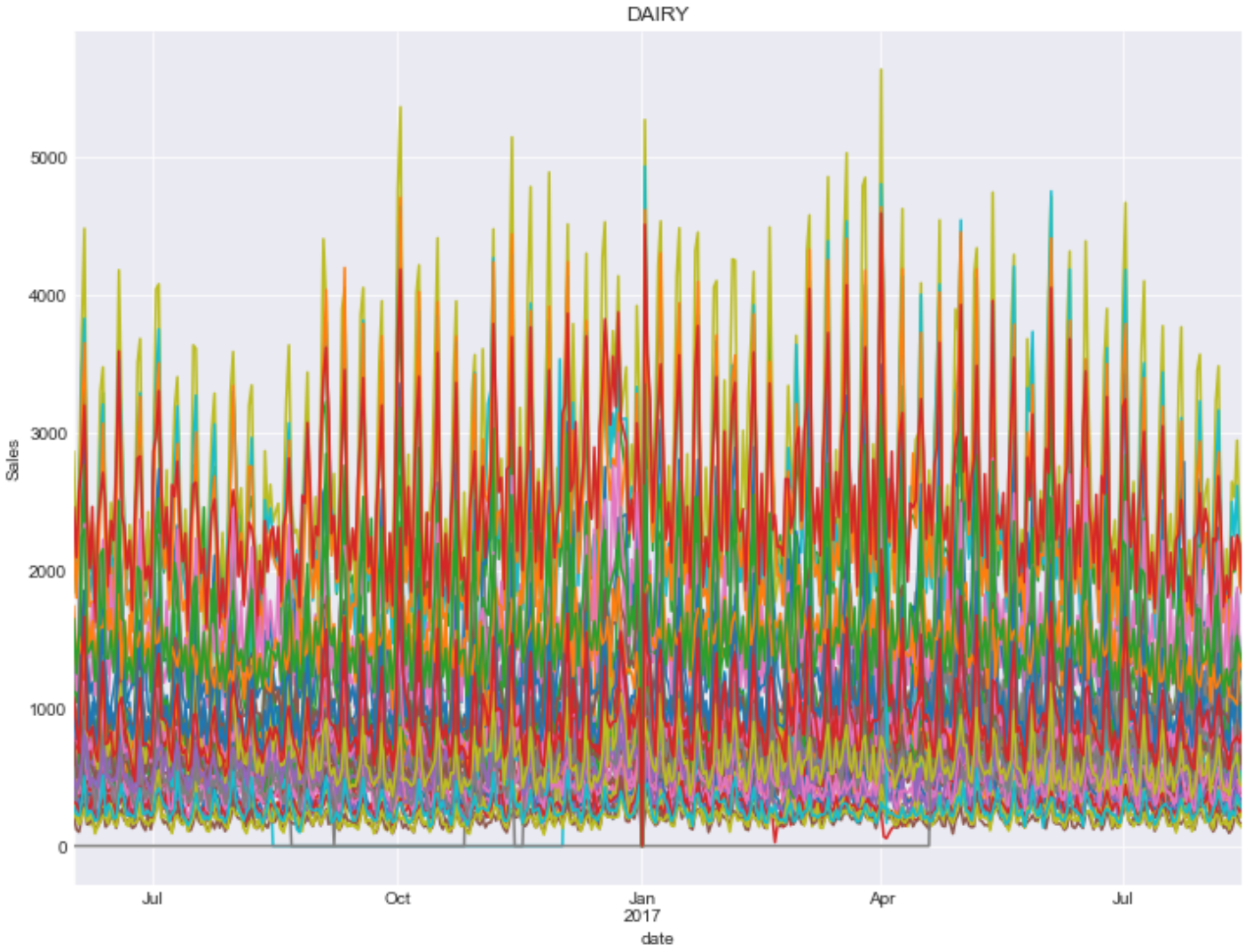

PLAYERS AND ELECTRONICS
120
100
80
60
40
20
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

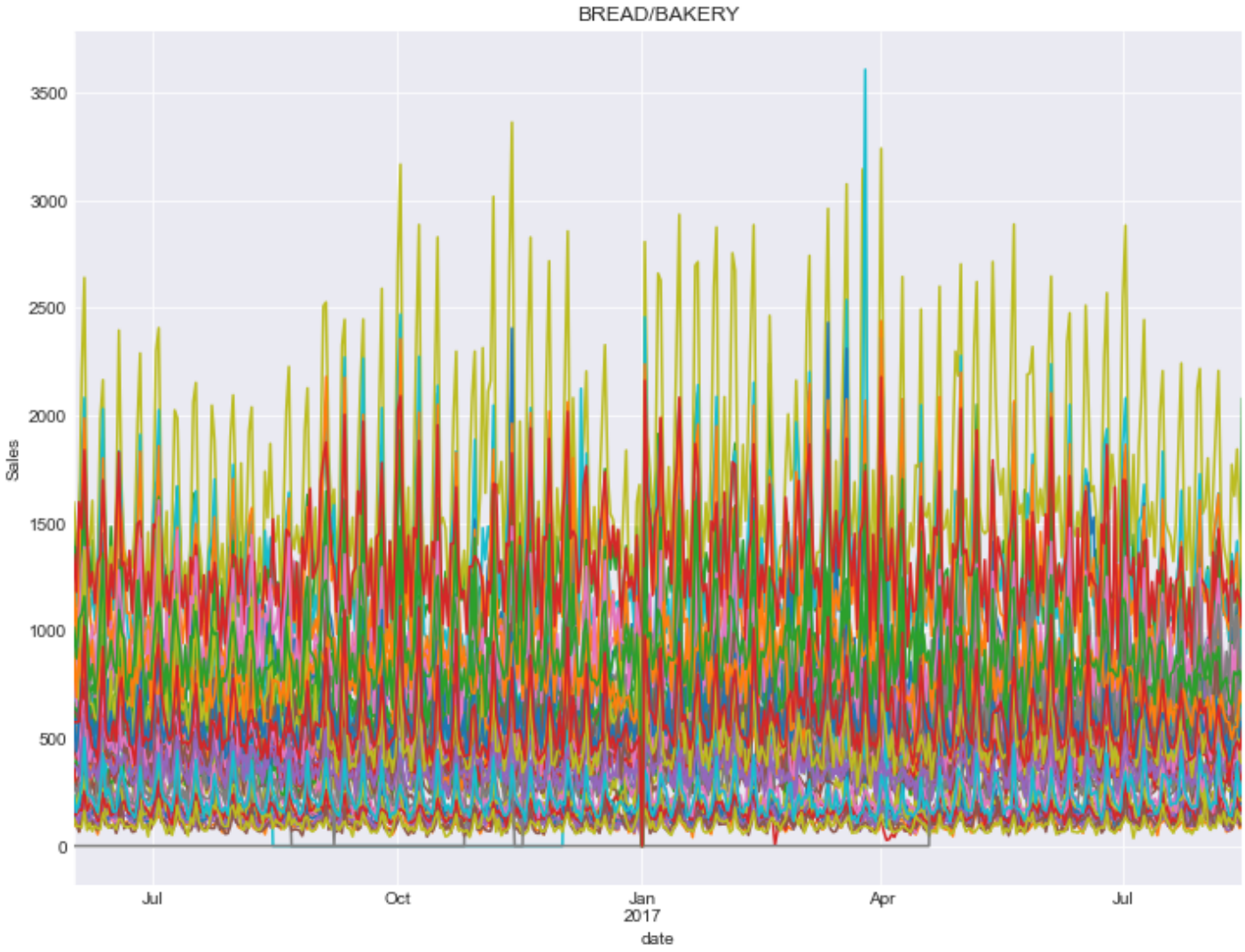

HOME AND KITCHEN II
Sales
700
600
500
400
300
200
100
0
Jul
Oct
Jan
2017
Apr
Jul
date

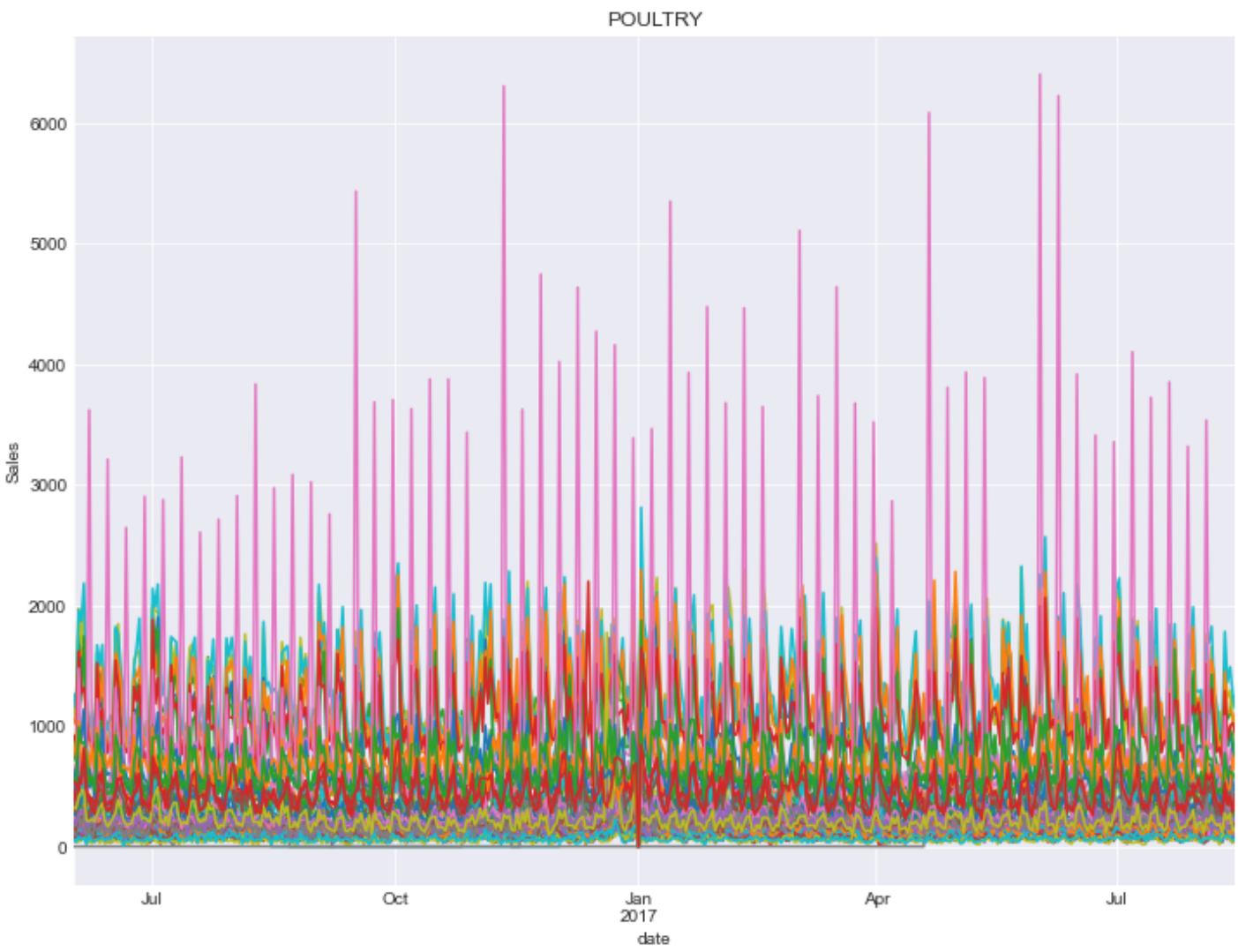

PREPARED FOODS
800
600
400
200
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date

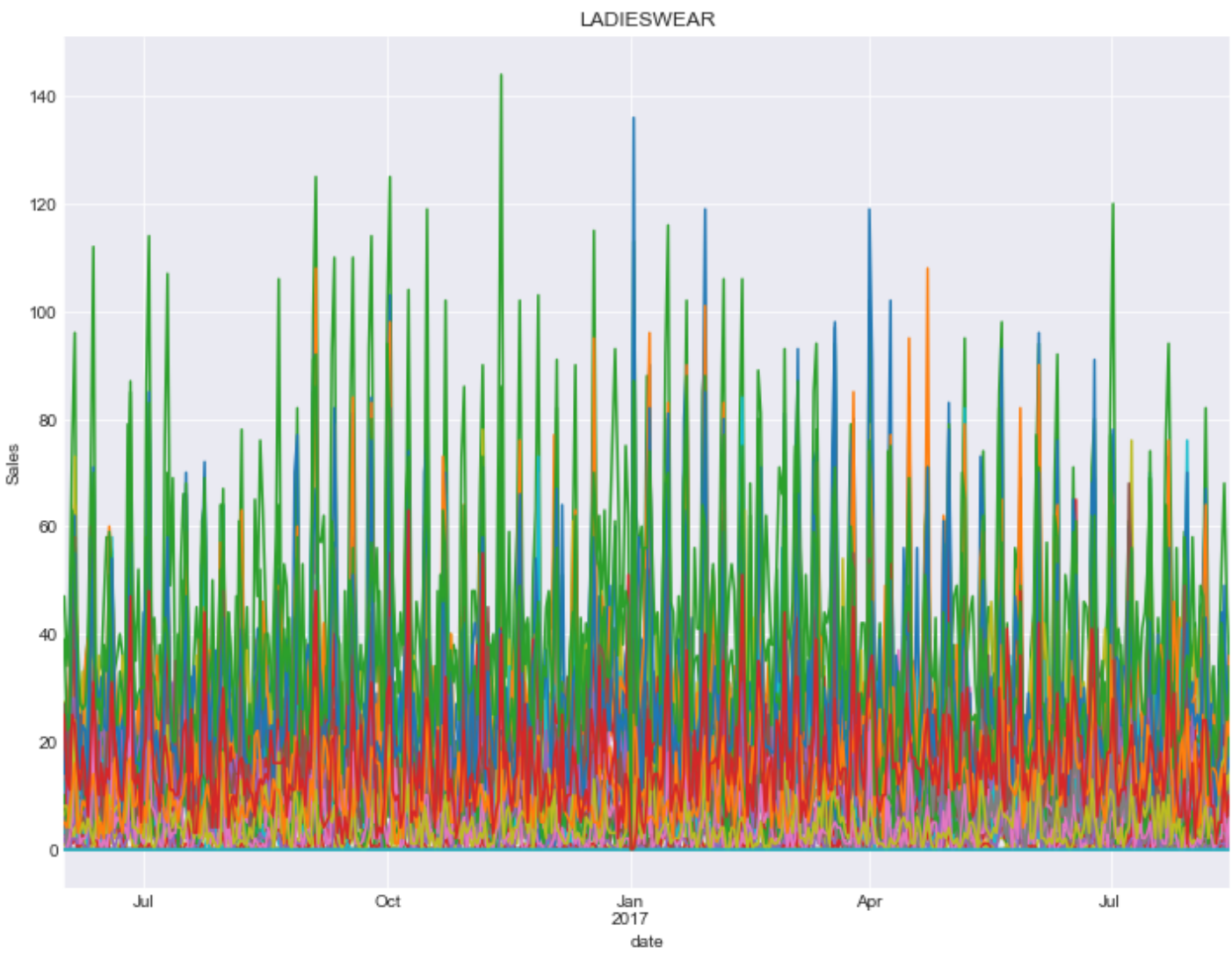

DELI
Sales
1750
1500
1250
1000
750
500
250
0
Jul
Oct
Jan
2017
Apr
Jul
date

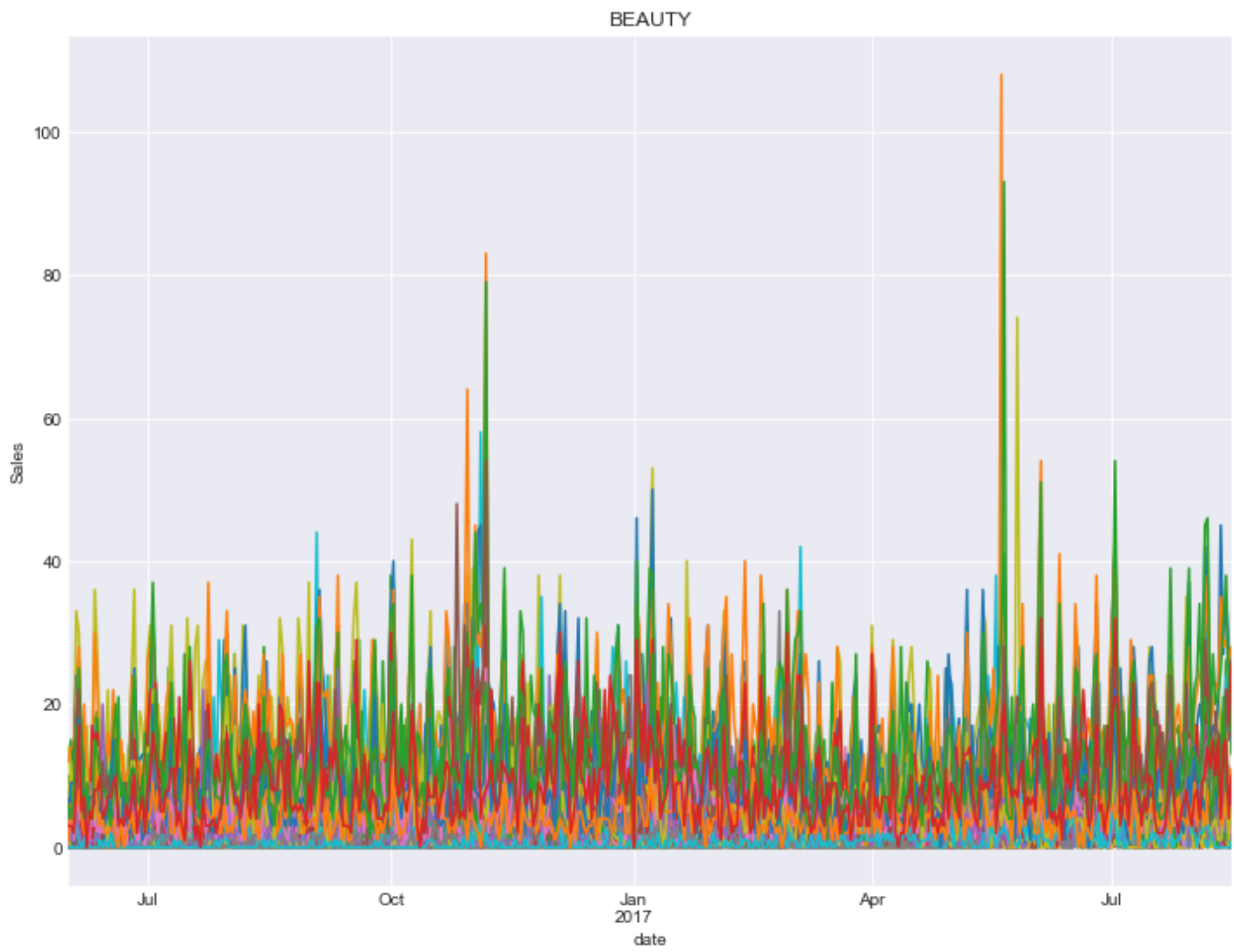

HOME AND KITCHEN I
2000
1750
1500
1250
1000
750
500
250
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date


DAIRY
5000
4000
3000
2000
1000
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date


BREAD/BAKERY
3500
3000
2500
2000
1500
1000
500
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date


POULTRY
6000
5000
4000
3000
2000
1000
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date


LADIESWEAR
140
120
100
80
60
40
20
0
Sales
Jul
Oct
Jan
2017
Apr
Jul
date



Graphs above are the visualization of each product

In [14]:
```
sdate = '2017-04-30' # Start and end of training date
edate = '2017-08-15'
```

In [15]:
```
school_season = [] # Feature for school fluctuations
for i, r in calendar.iterrows() :
    if i.month in [4, 5, 8, 9] :
        school_season.append(1)
    else :
        school_season.append(0)
calendar['school_season'] = school_season
calendar
```

Out[15]:

	avg_oil	oil_lags1	oil_lags2	oil_lags3	wd	dofw_1	dofw_2	dofw_3	dofw_4	dofw_5	dofw_6	type_Ac
2013-01-14	93.470000	93.284286	93.284286	93.284286	1	0	0	0	0	0	0	
2013-01-15	93.490000	93.470000	93.284286	93.284286	1	1	0	0	0	0	0	
2013-01-16	93.644286	93.490000	93.470000	93.284286	1	0	1	0	0	0	0	
2013-01-17	93.970000	93.644286	93.490000	93.470000	1	0	0	1	0	0	0	

	avg_oil	oil_lags1	oil_lags2	oil_lags3	wd	dofw_1	dofw_2	dofw_3	dofw_4	dofw_5	dofw_6	type_Ac
2013-01-18	94.331429	93.970000	93.644286	93.490000	1	0	0	0	1	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...
2017-08-27	47.720000	47.720000	47.720000	47.598571	0	0	0	0	0	0	1	
2017-08-28	47.624286	47.720000	47.720000	47.720000	1	0	0	0	0	0	0	
2017-08-29	47.320000	47.624286	47.720000	47.720000	1	1	0	0	0	0	0	
2017-08-30	47.115714	47.320000	47.624286	47.720000	1	0	1	0	0	0	0	
2017-08-31	47.060000	47.115714	47.320000	47.624286	1	0	0	1	0	0	0	

1691 rows × 20 columns

DeterministicProcess

In [16]:
```
y = train.unstack(['store_nbr', 'family']).loc[sdate:edate]
fourier = CalendarFourier(freq = 'W', order = 4)
dp = DeterministicProcess(index = y.index,
                          order = 1,
                          seasonal = False,
                          constant = False,
                          additional_terms = [fourier],
                          drop = True)
x = dp.in_sample()
x = x.join(calendar)
x
```

Out[16]:

date	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
2017-04-30	1.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	49.358571
2017-05-01	2.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	49.154286
2017-05-02	3.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	48.870000
2017-05-03	4.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	48.711429
2017-05-04	5.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	48.187143
...	...	...	...	...	...	...	...	...
2017-08-11	104.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	49.140000

date	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
2017-08-12	105.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	49.140000
2017-08-13	106.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	49.140000
2017-08-14	107.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	48.934286
2017-08-15	108.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	48.648571

108 rows × 27 columns

In [17]:
```
print(y.isna().sum().sum())
display(y)
```

0

store_nbr

family	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	DAIRY
date									
2017-04-30	3.0	0.0	0.0	995.0	1.0	139.50700	2.0	208.0	315.0
2017-05-01	0.0	0.0	2.0	825.0	0.0	116.33900	2.0	227.0	326.0
2017-05-02	2.0	0.0	2.0	3179.0	0.0	447.23800	20.0	1061.0	897.0
2017-05-03	5.0	0.0	6.0	2479.0	1.0	434.02900	22.0	1117.0	927.0
2017-05-04	3.0	0.0	1.0	2454.0	0.0	438.21400	15.0	956.0	755.0
...	...	...	...	...	...	...	...	...	...
2017-08-11	1.0	0.0	1.0	1006.0	0.0	145.60700	4.0	341.0	343.0
2017-08-12	6.0	0.0	3.0	1659.0	0.0	243.22000	3.0	351.0	526.0
2017-08-13	1.0	0.0	1.0	803.0	0.0	136.67900	1.0	169.0	266.0
2017-08-14	1.0	0.0	6.0	2201.0	0.0	346.03800	4.0	571.0	699.0
2017-08-15	4.0	0.0	4.0	1942.0	0.0	329.54102	21.0	703.0	602.0

108 rows × 1782 columns

In [18]:
```
xtest = dp.out_of_sample(steps = 16) # 16 because we are predicting next 16 days
xtest = xtest.join(calendar)
xtest
```

Out[18]:

	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
2017-08-16	109.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	48.281429
2017-08-17	110.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.995714
2017-08-18	111.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	47.852857
2017-08-19	112.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	47.852857
2017-08-20	113.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	47.852857
2017-08-21	114.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	47.688571
2017-08-22	115.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	47.522857
2017-08-23	116.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	47.645714
2017-08-24	117.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.598571
2017-08-25	118.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	47.720000
2017-08-26	119.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	47.720000
2017-08-27	120.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	47.720000
2017-08-28	121.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	47.624286
2017-08-29	122.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	47.320000
2017-08-30	123.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	47.115714
2017-08-31	124.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.060000

16 rows × 27 columns

In [19]:
```
def make_lags(x, lags = 1) : #Fungsi untuk membuat fitur lags
    lags = lags
    x_ = x.copy()
    for i in range(lags) :
        lag = x_.shift(i + 1)
        x = pd.concat([x, lag], axis = 1)
    return x
```

Using LinearRegression to make a generalized line (It's usually called blending.)

In [20]:

```
from joblib import Parallel, delayed
from tqdm.auto import tqdm
from sklearn.metrics import mean_squared_log_error as msle
from sklearn.model_selection import TimeSeriesSplit
from sklearn.svm import SVR
from sklearn.multioutput import MultiOutputRegressor

lnr = LinearRegression(fit_intercept = True, n_jobs = -1, normalize = True)
lnr.fit(x, y)

yfit_lnr = pd.DataFrame(lnr.predict(x), index = x.index, columns = y.columns).clip(0.)
ypred_lnr = pd.DataFrame(lnr.predict(xtest), index = xtest.index, columns = y.columns).cli

svr = MultiOutputRegressor(SVR(C = 0.2, kernel = 'rbf'), n_jobs = -1)
svr.fit(x, y)

yfit_svr = pd.DataFrame(svr.predict(x), index = x.index, columns = y.columns).clip(0.)
ypred_svr = pd.DataFrame(svr.predict(xtest), index = xtest.index, columns = y.columns).cli

yfit_mean = pd.DataFrame(np.mean([yfit_svr.values, yfit_lnr.values], axis = 0), index = x.
ypred_mean = pd.DataFrame(np.mean([ypred_lnr.values, ypred_svr.values], axis = 0), index =

y_ = y.stack(['store_nbr', 'family'])
y_['lnr'] = yfit_lnr.stack(['store_nbr', 'family'])['sales']
y_['svr'] = yfit_svr.stack(['store_nbr', 'family'])['sales']
y_['mean'] = yfit_mean.stack(['store_nbr', 'family'])['sales']

print('='*50, 'Linear Regression', '='*50)
print(y_.groupby('family').apply(lambda r : np.sqrt(msle(r['sales'], r['lnr']))))
print('LNR RMSLE :', np.sqrt(msle(y, yfit_lnr)))
print('='*50, 'SVR', '='*50)
print(y_.groupby('family').apply(lambda r : np.sqrt(msle(r['sales'], r['svr']))))
print('SVR RMSLE :', np.sqrt(msle(y, yfit_svr)))
print('='*50, 'Mean', '='*50)
print(y_.groupby('family').apply(lambda r : np.sqrt(msle(r['sales'], r['mean']))))
print('Mean RMSLE :', np.sqrt(msle(y, yfit_mean)))
```

```
================================================== Linear Regression ====================
============================
family
AUTOMOTIVE                    0.467574
BABY CARE                     0.251807
BEAUTY                        0.465461
BEVERAGES                     0.164721
BOOKS                         0.122513
BREAD/BAKERY                  0.131096
CELEBRATION                   0.507705
CLEANING                      0.334493
DAIRY                         0.120512
DELI                          0.150241
EGGS                          0.282907
FROZEN FOODS                  0.242533
GROCERY I                     0.145968
GROCERY II                    0.545454
HARDWARE                      0.487417
HOME AND KITCHEN I            0.457091
HOME AND KITCHEN II           0.451102
HOME APPLIANCES               0.359912
HOME CARE                     0.187623
LADIESWEAR                    0.444551
```

```
LAWN AND GARDEN               0.394160
LINGERIE                      0.582688
LIQUOR,WINE,BEER              0.547617
MAGAZINES                     0.456816
MEATS                         0.161546
PERSONAL CARE                 0.207893
PET SUPPLIES                  0.419474
PLAYERS AND ELECTRONICS       0.420885
POULTRY                       0.169622
PREPARED FOODS                0.243171
PRODUCE                       0.106124
SCHOOL AND OFFICE SUPPLIES    0.978071
SEAFOOD                       0.454080
dtype: float64
LNR RMSLE : 0.3939950841717154
================================================== SVR ==================================
===============
family
AUTOMOTIVE                    0.538003
BABY CARE                     0.276040
BEAUTY                        0.548494
BEVERAGES                     0.287354
BOOKS                         0.148125
BREAD/BAKERY                  0.247434
CELEBRATION                   0.583788
CLEANING                      0.342808
DAIRY                         0.257000
DELI                          0.259029
EGGS                          0.404391
FROZEN FOODS                  0.397954
GROCERY I                     0.249870
GROCERY II                    0.594411
HARDWARE                      0.537447
HOME AND KITCHEN I            0.512488
HOME AND KITCHEN II           0.468337
HOME APPLIANCES               0.407768
HOME CARE                     0.306198
LADIESWEAR                    0.542889
LAWN AND GARDEN               0.521845
LINGERIE                      0.636773
LIQUOR,WINE,BEER              0.691029
MAGAZINES                     0.525053
MEATS                         0.313577
PERSONAL CARE                 0.327348
PET SUPPLIES                  0.489559
PLAYERS AND ELECTRONICS       0.495011
POULTRY                       0.308538
PREPARED FOODS                0.312284
PRODUCE                       0.288331
SCHOOL AND OFFICE SUPPLIES    0.945396
SEAFOOD                       0.538674
dtype: float64
SVR RMSLE : 0.46279589943989
================================================== Mean =================================
================
family
AUTOMOTIVE                    0.482208
BABY CARE                     0.246344
BEAUTY                        0.477568
BEVERAGES                     0.191608
BOOKS                         0.125442
BREAD/BAKERY                  0.163472
CELEBRATION                   0.513064
CLEANING                      0.272229
DAIRY                         0.159272
DELI                          0.180674
```

```
EGGS                            0.292826
FROZEN FOODS                    0.274777
GROCERY I                       0.171275
GROCERY II                      0.523404
HARDWARE                        0.488310
HOME AND KITCHEN I              0.448673
HOME AND KITCHEN II             0.412161
HOME APPLIANCES                 0.358596
HOME CARE                       0.215621
LADIESWEAR                      0.447354
LAWN AND GARDEN                 0.417942
LINGERIE                        0.579607
LIQUOR,WINE,BEER                0.489847
MAGAZINES                       0.468171
MEATS                           0.202253
PERSONAL CARE                   0.234314
PET SUPPLIES                    0.435392
PLAYERS AND ELECTRONICS         0.436626
POULTRY                         0.205004
PREPARED FOODS                  0.260602
PRODUCE                         0.162398
SCHOOL AND OFFICE SUPPLIES      0.753955
SEAFOOD                         0.467708
dtype: float64
Mean RMSLE : 0.3819594017570757
```

In [21]:

```
from sklearn.metrics import mean_absolute_error as mae

print('='*50, 'Linear Regression', '='*50)
print(y_.groupby('family').apply(lambda r : mae(r['sales'], r['lnr'])))
print('LNR RMSLE :', mae(y, yfit_lnr))
print('='*50, 'SVR', '='*50)
print(y_.groupby('family').apply(lambda r : mae(r['sales'], r['svr'])))
print('SVR RMSLE :', mae(y, yfit_svr))
print('='*50, 'Mean', '='*50)
print(y_.groupby('family').apply(lambda r : mae(r['sales'], r['mean'])))
print('Mean RMSLE :', mae(y, yfit_mean))
```

```
================================================== Linear Regression ====================
==============================
family
AUTOMOTIVE                      2.442080
BABY CARE                       0.216345
BEAUTY                          1.968669
BEVERAGES                     387.940862
BOOKS                           0.061457
BREAD/BAKERY                   48.151179
CELEBRATION                     4.428800
CLEANING                      233.191742
DAIRY                          76.291747
DELI                           34.817572
EGGS                           28.729909
FROZEN FOODS                   19.250407
GROCERY I                     480.934806
GROCERY II                      8.560527
HARDWARE                        0.934495
HOME AND KITCHEN I             10.195658
HOME AND KITCHEN II             9.677513
HOME APPLIANCES                 0.460105
HOME CARE                      39.301723
LADIESWEAR                      3.341730
LAWN AND GARDEN                 4.186430
LINGERIE                        2.944640
LIQUOR,WINE,BEER               21.053310
MAGAZINES                       2.053009
```

```
MEATS                          40.098808
PERSONAL CARE                  47.373139
PET SUPPLIES                    2.405904
PLAYERS AND ELECTRONICS         2.914275
POULTRY                        42.034273
PREPARED FOODS                 12.318155
PRODUCE                       161.693724
SCHOOL AND OFFICE SUPPLIES      7.537295
SEAFOOD                         4.106225
dtype: float64
LNR RMSLE : 52.77625798557621
================================================ SVR ================================
===============
family
AUTOMOTIVE                      3.038803
BABY CARE                       0.213634
BEAUTY                          2.589691
BEVERAGES                     807.458009
BOOKS                           0.078355
BREAD/BAKERY                  101.145436
CELEBRATION                     5.584398
CLEANING                      319.913726
DAIRY                         180.692332
DELI                           68.286457
EGGS                           56.842326
FROZEN FOODS                   38.264472
GROCERY I                     920.832951
GROCERY II                     10.857967
HARDWARE                        1.031666
HOME AND KITCHEN I             12.069607
HOME AND KITCHEN II            10.982188
HOME APPLIANCES                 0.495151
HOME CARE                      72.678242
LADIESWEAR                      5.104468
LAWN AND GARDEN                 6.987729
LINGERIE                        3.431237
LIQUOR,WINE,BEER               44.262642
MAGAZINES                       2.614445
MEATS                          95.632221
PERSONAL CARE                  84.484467
PET SUPPLIES                    3.281105
PLAYERS AND ELECTRONICS         4.082930
POULTRY                        96.222824
PREPARED FOODS                 18.383223
PRODUCE                       513.937437
SCHOOL AND OFFICE SUPPLIES     11.533356
SEAFOOD                         6.531768
dtype: float64
SVR RMSLE : 106.34985652463914
================================================ Mean ===============================
================
family
AUTOMOTIVE                      2.599489
BABY CARE                       0.213773
BEAUTY                          2.133365
BEVERAGES                     526.464207
BOOKS                           0.069840
BREAD/BAKERY                   66.053145
CELEBRATION                     4.680432
CLEANING                      252.507720
DAIRY                         112.042779
DELI                           46.121385
EGGS                           38.004720
FROZEN FOODS                   25.164598
GROCERY I                     623.650687
GROCERY II                      9.063140
```

```
HARDWARE                        0.959691
HOME AND KITCHEN I             10.407447
HOME AND KITCHEN II             9.644035
HOME APPLIANCES                 0.470557
HOME CARE                      49.905125
LADIESWEAR                      3.872454
LAWN AND GARDEN                 5.189201
LINGERIE                        3.061907
LIQUOR,WINE,BEER               28.235661
MAGAZINES                       2.217959
MEATS                          60.070083
PERSONAL CARE                  58.694391
PET SUPPLIES                    2.667893
PLAYERS AND ELECTRONICS         3.278149
POULTRY                        60.596267
PREPARED FOODS                 14.204071
PRODUCE                       293.829495
SCHOOL AND OFFICE SUPPLIES      8.618245
SEAFOOD                         4.855631
dtype: float64
Mean RMSLE : 70.59234983126385
```

As you can see, with RMSLE, the best model is the averaging of linear regression and SVR.

But, in MAE, Linear Regression has the least loss than Mean. What does it mean?

Because in RMSLE we are applying log, that means higher the value, the lower the deviation.

Let me show you

In [22]:
```
true_low = [2]
pred_low = [4]

print('RMSLE for low value :', np.sqrt(msle(true_low, pred_low)))
print('MAE for low value :', mae(true_low, pred_low))

true_high = [255]
pred_high = [269]

print('RMSLE for high value :', np.sqrt(msle(true_high, pred_high)))
print('MAE for high value :', mae(true_high, pred_high))
```

```
RMSLE for low value : 0.5108256237659905
MAE for low value : 2.0
RMSLE for high value : 0.053244514518812736
MAE for high value : 14.0
```

As you can see, RMSLE will have higher "tolerance" for higher value meanwhile Mean Absolute Error will go as usual.

Therefore, I will take the Linear Regression's result instead because it has lower MAE and MAE is reliable because it's robust to outlier

I'm not gonna use validation data because the data we have is not much and because we are using linear-based algorithm so only using training would be fine.

In [23]:
```
display(x, xtest)
```

	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
date								

date	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
2017-04-30	1.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	49.358571
2017-05-01	2.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	49.154286
2017-05-02	3.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	48.870000
2017-05-03	4.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	48.711429
2017-05-04	5.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	48.187143
...	...	...	...	...	...	...	...	...
2017-08-11	104.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	49.140000
2017-08-12	105.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	49.140000
2017-08-13	106.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	49.140000
2017-08-14	107.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	48.934286
2017-08-15	108.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	48.648571

108 rows × 27 columns

date	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
2017-08-16	109.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	48.281429
2017-08-17	110.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.995714
2017-08-18	111.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	47.852857
2017-08-19	112.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	47.852857
2017-08-20	113.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	47.852857
2017-08-21	114.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	47.688571
2017-08-22	115.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	47.522857
2017-08-23	116.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	47.645714
2017-08-24	117.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.598571

	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
2017-08-25	118.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	47.720000
2017-08-26	119.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	47.720000
2017-08-27	120.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	47.720000
2017-08-28	121.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	47.624286
2017-08-29	122.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	47.320000
2017-08-30	123.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	47.115714
2017-08-31	124.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.060000

16 rows × 27 columns

In [24]:
```
ypred_svr
```

Out[24]:

store_nbr

family	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	DA
2017-08-16	4.167616	0.0	3.106738	2325.255078	0.099660	373.675216	12.960532	703.347053	745.640
2017-08-17	4.170530	0.0	3.108458	2325.244128	0.099596	373.659826	12.955996	703.339209	745.623
2017-08-18	4.174150	0.0	3.109134	2325.238307	0.099890	373.648625	12.950268	703.335088	745.612
2017-08-19	4.176423	0.0	3.110713	2325.228046	0.099546	373.635221	12.942318	703.325831	745.597
2017-08-20	4.175686	0.0	3.110323	2325.216809	0.099480	373.620677	12.934499	703.317980	745.582
2017-08-21	4.178321	0.0	3.113246	2325.210775	0.099840	373.611547	12.931209	703.313647	745.572
2017-08-22	4.180108	0.0	3.113897	2325.203328	0.099689	373.600088	12.926298	703.308889	745.559
2017-08-23	4.181297	0.0	3.114472	2325.195087	0.099533	373.591268	12.920400	703.303147	745.550
2017-08-24	4.182594	0.0	3.114758	2325.185122	0.099522	373.579132	12.916589	703.295256	745.537
2017-08-25	4.184427	0.0	3.113910	2325.179816	0.099819	373.570921	12.911496	703.290533	745.527

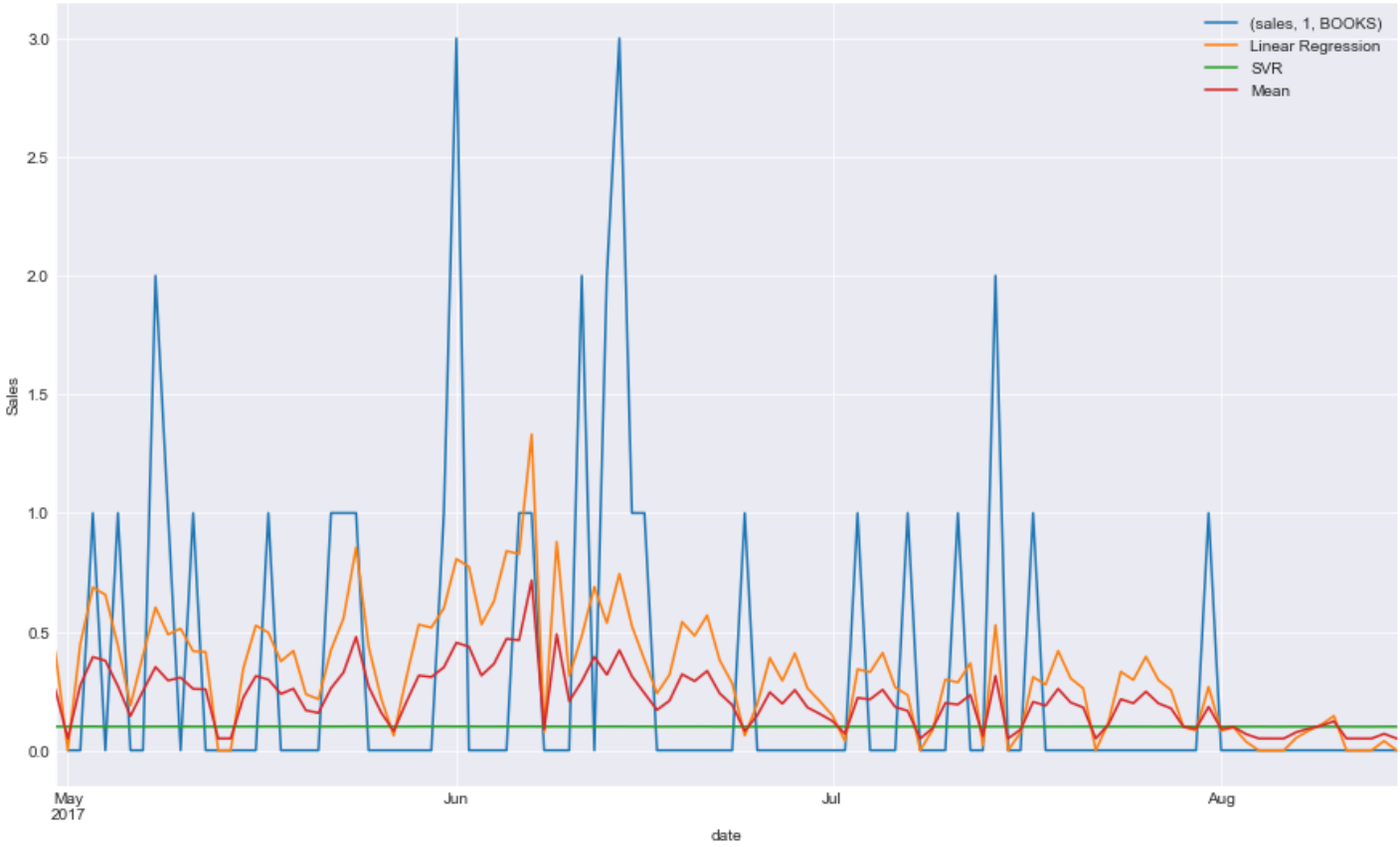
store_nbr

family	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	DA
2017-08-26	4.185025	0.0	3.114221	2325.170420	0.099505	373.560708	12.904458	703.281207	745.516
2017-08-27	4.183004	0.0	3.113137	2325.160444	0.099431	373.549571	12.897875	703.273881	745.503
2017-08-28	4.184209	0.0	3.115297	2325.155401	0.099781	373.543456	12.895611	703.270102	745.496
2017-08-29	4.184780	0.0	3.115669	2325.149278	0.099630	373.535338	12.892119	703.266483	745.487
2017-08-30	4.184837	0.0	3.116258	2325.142555	0.099455	373.529888	12.887944	703.262419	745.482
2017-08-31	4.185366	0.0	3.116459	2325.134617	0.099471	373.521577	12.885936	703.256447	745.472

16 rows × 1782 columns

In [41]:

```
fam = 'BOOKS'
nbr = '1'
plt.rcParams['figure.figsize'] = (15, 9)
plt.figure()
y.loc(axis = 1)['sales', nbr, fam].plot()
yfit_lnr.loc(axis = 1)['sales', nbr, fam].plot(label = 'Linear Regression')
yfit_svr.loc(axis = 1)['sales', nbr, fam].plot(label = 'SVR')
yfit_mean.loc(axis = 1)['sales', nbr, fam].plot(label = 'Mean')
# y.mean(axis = 1).plot()
# yfit_lnr.median(axis = 1).plot(label = 'Linear Regression')
# yfit_svr.median(axis = 1).plot(label = 'SVR')
# yfit_mean.mean(axis = 1).plot(label = 'Mean')
plt.ylabel("Sales")
plt.legend()
plt.show()
```



You can concat linear regression's prediction with the training data, this is called blending.

In [26]:
```
ymean = yfit_lnr.append(ypred_lnr)
school = ymean.loc(axis = 1)['sales', :, 'SCHOOL AND OFFICE SUPPLIES']
ymean = ymean.join(school.shift(1), rsuffix = 'lag1') # I'm also adding school lag for it
# x = x.loc['2017-05-01':]
```

In [27]:
```
ymean.loc['2017-08-16':]
```

Out[27]:

store_nbr

family	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	DA
2017-08-16	3.703143	0.0	5.961231	2175.355695	0.062954	378.512360	11.840255	782.615017	756.305
2017-08-17	3.265076	0.0	5.391272	1808.988688	0.000000	325.991217	15.919410	637.899994	631.115
2017-08-18	7.346108	0.0	4.713946	2222.180801	0.000000	362.877846	20.578767	732.449983	721.036
2017-08-19	5.505018	0.0	5.420240	2101.679045	0.000000	331.970197	7.913550	550.062696	673.922
2017-08-20	1.969054	0.0	3.584260	783.858009	0.000000	122.621511	0.214962	185.298778	258.219
2017-08-21	3.964694	0.0	5.560288	2083.247431	0.000000	352.036950	12.819171	626.875612	669.740

store_nbr

family	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	DA
2017-08-22	4.102674	0.0	5.240299	2137.604724	0.000000	336.103079	13.508832	719.433833	634.796
2017-08-23	3.924562	0.0	6.007869	2152.555395	0.000000	376.762706	11.006355	764.623216	754.871
2017-08-24	3.260815	0.0	5.043776	1820.391159	0.000000	320.368187	15.918683	643.227689	628.717
2017-08-25	7.924292	0.0	4.292528	2214.964855	0.000000	355.396813	20.653618	723.783654	698.523
2017-08-26	5.631009	0.0	5.193243	2098.854910	0.000000	324.983190	8.225574	553.732290	658.837
2017-08-27	2.255153	0.0	3.586564	759.147229	0.000000	116.668523	0.324867	178.178812	237.490
2017-08-28	4.035039	0.0	5.669174	2063.811288	0.000000	348.423139	12.393976	617.137530	661.740
2017-08-29	3.971870	0.0	5.392265	2119.649899	0.000000	331.258484	13.652275	721.828970	624.716
2017-08-30	3.777936	0.0	6.476885	2108.142533	0.000000	371.732994	11.626553	768.369196	735.212
2017-08-31	3.536644	0.0	6.124531	1836.284366	0.000000	336.179255	14.822816	579.204554	602.215

16 rows × 1836 columns

In [28]:
```
x = x.join(ymean)  # Concating linear result
xtest = xtest.join(ymean)
display(x, xtest)
```

	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
date								
2017-04-30	1.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	49.358571
2017-05-01	2.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	49.154286
2017-05-02	3.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	48.870000
2017-05-03	4.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	48.711429

date	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
2017-05-04	5.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	48.187143
...	...	...	...	...	...	...	...	...
2017-08-11	104.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	49.140000
2017-08-12	105.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	49.140000
2017-08-13	106.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	49.140000
2017-08-14	107.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	48.934286
2017-08-15	108.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	48.648571

108 rows × 1863 columns

date	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
2017-08-16	109.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	48.281429
2017-08-17	110.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.995714
2017-08-18	111.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	47.852857
2017-08-19	112.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	47.852857
2017-08-20	113.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	47.852857
2017-08-21	114.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	47.688571
2017-08-22	115.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	47.522857
2017-08-23	116.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	47.645714
2017-08-24	117.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.598571
2017-08-25	118.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	47.720000

	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
2017-08-26	119.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	47.720000
2017-08-27	120.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	47.720000
2017-08-28	121.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	47.624286
2017-08-29	122.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	47.320000
2017-08-30	123.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	47.115714
2017-08-31	124.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.060000

16 rows × 1863 columns

In [29]:
```
y = y.loc['2017-05-01':]
y
```

Out[29]:

store_nbr

family	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	DAIRY
date									
2017-05-01	0.0	0.0	2.0	825.0	0.0	116.33900	2.0	227.0	326.0
2017-05-02	2.0	0.0	2.0	3179.0	0.0	447.23800	20.0	1061.0	897.0
2017-05-03	5.0	0.0	6.0	2479.0	1.0	434.02900	22.0	1117.0	927.0
2017-05-04	3.0	0.0	1.0	2454.0	0.0	438.21400	15.0	956.0	755.0
2017-05-05	12.0	0.0	0.0	2243.0	1.0	398.96500	15.0	829.0	882.0
...	...	...	...	...	...	...	...	...	...
2017-08-11	1.0	0.0	1.0	1006.0	0.0	145.60700	4.0	341.0	343.0
2017-08-12	6.0	0.0	3.0	1659.0	0.0	243.22000	3.0	351.0	526.0
2017-08-13	1.0	0.0	1.0	803.0	0.0	136.67900	1.0	169.0	266.0

store_nbr

family	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	DAIRY
date									
2017-08-14	1.0	0.0	6.0	2201.0	0.0	346.03800	4.0	571.0	699.0
2017-08-15	4.0	0.0	4.0	1942.0	0.0	329.54102	21.0	703.0	602.0

107 rows × 1782 columns

In [30]:
```
print(y.isna().sum().sum())
print(x.isna().sum().sum())
```

```
0
54
```

In [31]:
```
x.dropna(inplace = True)
display(x, xtest)
```

	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
date								
2017-05-01	2.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	49.154286
2017-05-02	3.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	48.870000
2017-05-03	4.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	48.711429
2017-05-04	5.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	48.187143
2017-05-05	6.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	47.760000
...	...	...	...	...	...	...	...	...
2017-08-11	104.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	49.140000
2017-08-12	105.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	49.140000
2017-08-13	106.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	49.140000
2017-08-14	107.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	48.934286

	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
date								
2017-08-15	108.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	48.648571

107 rows × 1863 columns

	trend	sin(1,freq=W-SUN)	cos(1,freq=W-SUN)	sin(2,freq=W-SUN)	cos(2,freq=W-SUN)	sin(3,freq=W-SUN)	cos(3,freq=W-SUN)	avg_oil
2017-08-16	109.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	48.281429
2017-08-17	110.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.995714
2017-08-18	111.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	47.852857
2017-08-19	112.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	47.852857
2017-08-20	113.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	47.852857
2017-08-21	114.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	47.688571
2017-08-22	115.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	47.522857
2017-08-23	116.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	47.645714
2017-08-24	117.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.598571
2017-08-25	118.0	-0.433884	-0.900969	0.781831	0.623490	-0.974928	-0.222521	47.720000
2017-08-26	119.0	-0.974928	-0.222521	0.433884	-0.900969	0.781831	0.623490	47.720000
2017-08-27	120.0	-0.781831	0.623490	-0.974928	-0.222521	-0.433884	-0.900969	47.720000
2017-08-28	121.0	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	47.624286
2017-08-29	122.0	0.781831	0.623490	0.974928	-0.222521	0.433884	-0.900969	47.320000
2017-08-30	123.0	0.974928	-0.222521	-0.433884	-0.900969	-0.781831	0.623490	47.115714
2017-08-31	124.0	0.433884	-0.900969	-0.781831	0.623490	0.974928	-0.222521	47.060000

16 rows × 1863 columns

This is the model I use, as I said I'm taking it from BIZEN and modifying it.

Model Creation

In [32]:

```
from joblib import Parallel, delayed
import warnings

# Import necessary library
from sklearn.linear_model import Ridge, LinearRegression, ElasticNet
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import VotingRegressor

# SEED for reproducible result
SEED = 5

class CustomRegressor():

    def __init__(self, n_jobs=-1, verbose=0):

        self.n_jobs = n_jobs
        self.verbose = verbose

        self.estimators_ = None

    def _estimator_(self, X, y):

        warnings.simplefilter(action='ignore', category=FutureWarning)

        if y.name[2] == 'SCHOOL AND OFFICE SUPPLIES': # Because SCHOOL AND OFFICE SUPPLIES
            r1 = ExtraTreesRegressor(n_estimators = 225, n_jobs=-1, random_state=SEED)
            r2 = RandomForestRegressor(n_estimators = 225, n_jobs=-1, random_state=SEED)
            b1 = BaggingRegressor(base_estimator=r1,
                                  n_estimators=10,
                                  n_jobs=-1,
                                  random_state=SEED)
            b2 = BaggingRegressor(base_estimator=r2,
                                  n_estimators=10,
                                  n_jobs=-1,
                                  random_state=SEED)
            model = VotingRegressor([('et', b1), ('rf', b2)]) # Averaging the result
        else:
            ridge = Ridge(fit_intercept=True, solver='auto', alpha=0.75, normalize=True, r
            svr = SVR(C = 0.2, kernel = 'rbf')

            model = VotingRegressor([('ridge', ridge), ('svr', svr)]) # Averaging result
        model.fit(X, y)

        return model

    def fit(self, X, y):
        from tqdm.auto import tqdm


        if self.verbose == 0 :
            self.estimators_ = Parallel(n_jobs=self.n_jobs,
                                  verbose=0,
                                  )(delayed(self._estimator_)(X, y.iloc[:, i]) for i in ra
```

```
        else :
            print('Fit Progress')
            self.estimators_ = Parallel(n_jobs=self.n_jobs,
                                        verbose=0,
                                        )(delayed(self._estimator_)(X, y.iloc[:, i]) for i in tq
        return

    def predict(self, X):
        from tqdm.auto import tqdm
        if self.verbose == 0 :
            y_pred = Parallel(n_jobs=self.n_jobs,
                              verbose=0)(delayed(e.predict)(X) for e in self.estimators_)
        else :
            print('Predict Progress')
            y_pred = Parallel(n_jobs=self.n_jobs,
                              verbose=0)(delayed(e.predict)(X) for e in tqdm(self.estimato

        return np.stack(y_pred, axis=1)
```

In [33]:

```
%%time

model = CustomRegressor(n_jobs=-1, verbose=1)
model.fit(x, y)
y_pred = pd.DataFrame(model.predict(x), index=x.index, columns=y.columns)
```

```
Fit Progress

Predict Progress

Wall time: 18min 2s
```

In [34]:

```
display(y_pred)
print(y_pred.isna().sum().sum())
```

store_nbr

family	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	D
date									
2017-05-01	1.710101	0.0	2.172708	1581.195785	0.051013	245.130208	6.680767	464.544456	536.58
2017-05-02	2.859073	0.0	2.637962	2667.907725	0.147906	405.471354	17.418823	860.060848	805.29
2017-05-03	3.633820	0.0	3.738809	2405.576439	0.429805	410.554142	15.717973	834.248104	826.77
2017-05-04	3.319663	0.0	3.253936	2205.132578	0.359016	380.146680	17.593851	740.946102	754.10
2017-05-05	5.532839	0.0	3.069640	2387.331187	0.268401	397.334794	19.466068	771.567047	798.30
...	...	...	...	...	...	...	...	...	
2017-08-11	2.508385	0.0	1.909387	1619.202564	0.008018	258.154745	8.487920	496.823838	533.99

store_nbr

family	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	DA
date									
2017-08-12	4.156669	0.0	3.048395	2113.597009	-0.085538	316.117429	9.004954	547.350765	649.66
2017-08-13	3.199676	0.0	2.582250	1617.152427	-0.021156	253.826917	6.727258	470.198773	513.76
2017-08-14	4.161053	0.0	4.089362	2236.624695	0.112755	364.030801	13.341007	663.462510	716.78
2017-08-15	4.314411	0.0	3.863173	2240.804914	0.079994	354.789842	13.974925	702.143599	688.35

107 rows × 1782 columns

0

Evaluation

In [35]:

```
from sklearn.metrics import mean_squared_log_error
y_pred = y_pred.stack(['store_nbr', 'family']).clip(0.)
y_ = y.stack(['store_nbr', 'family']).clip(0.)

y_['pred'] = y_pred.values
print(y_.groupby('family').apply(lambda r : np.sqrt(np.sqrt(mean_squared_log_error(r['sale
# Looking at error
print('RMSLE : ', np.sqrt(np.sqrt(msle(y_['sales'], y_['pred']))))
```

```
family
AUTOMOTIVE                    0.680103
BABY CARE                     0.483875
BEAUTY                        0.675622
BEVERAGES                     0.428625
BOOKS                         0.343947
BREAD/BAKERY                  0.396854
CELEBRATION                   0.704345
CLEANING                      0.507188
DAIRY                         0.392309
DELI                          0.417531
EGGS                          0.531208
FROZEN FOODS                  0.515245
GROCERY I                     0.405405
GROCERY II                    0.706123
HARDWARE                      0.683425
HOME AND KITCHEN I            0.655306
HOME AND KITCHEN II           0.629114
HOME APPLIANCES               0.587010
HOME CARE                     0.452552
LADIESWEAR                    0.654179
LAWN AND GARDEN               0.635128
LINGERIE                      0.747751
LIQUOR,WINE,BEER              0.686544
MAGAZINES                     0.670218
MEATS                         0.443427
```

```
PERSONAL CARE                 0.473083
PET SUPPLIES                  0.644720
PLAYERS AND ELECTRONICS       0.645568
POULTRY                       0.445695
PREPARED FOODS                0.501518
PRODUCE                       0.399338
SCHOOL AND OFFICE SUPPLIES    0.668054
SEAFOOD                       0.671963
dtype: float64
RMSLE :  0.5940810072021273
```

All seems good.

In [36]:
```
y_pred.isna().sum()
```

Out[36]:
```
sales    0
dtype: int64
```

In [37]:
```
ypred = pd.DataFrame(model.predict(xtest), index = xtest.index, columns = y.columns).clip
ypred
```

```
Predict Progress
```

Out[37]:

store_nbr

family	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	DA
2017-08-16	4.087050	0.0	4.541333	2263.676462	0.083917	377.234540	13.086333	734.304642	753.389
2017-08-17	3.925971	0.0	4.255894	2068.168667	0.032209	347.717506	15.188844	665.062301	683.236
2017-08-18	5.984456	0.0	3.922917	2287.214546	0.018095	369.736630	17.492370	711.274759	737.068
2017-08-19	5.024207	0.0	3.885471	2228.968159	0.000000	353.569865	10.932725	627.682617	714.592
2017-08-20	3.054923	0.0	2.977994	1568.510928	0.000000	250.975497	6.724591	439.236029	508.874
2017-08-21	4.101424	0.0	4.369636	2210.981086	0.072566	362.772352	13.593018	662.618027	713.493
2017-08-22	4.348088	0.0	4.068288	2218.522386	0.022500	353.377613	14.091944	699.076808	685.494
2017-08-23	4.255025	0.0	4.572167	2253.468552	0.015387	377.939947	12.862961	727.022002	756.618
2017-08-24	3.915132	0.0	4.157769	2078.024517	0.040735	343.750922	14.936246	669.832736	682.076
2017-08-25	6.241452	0.0	3.752043	2289.810858	0.000000	367.493687	17.596576	709.073236	726.731
2017-08-26	4.911870	0.0	3.848903	2233.820667	0.000000	348.822733	10.923239	635.451337	705.589
2017-08-27	3.105618	0.0	2.985311	1554.685065	0.000000	247.272055	6.767610	438.120497	498.722

store_nbr

family	AUTOMOTIVE	BABY CARE	BEAUTY	BEVERAGES	BOOKS	BREAD/BAKERY	CELEBRATION	CLEANING	DA
2017-08-28	4.147133	0.0	4.452798	2197.652654	0.043968	359.883580	13.319269	658.730316	708.072
2017-08-29	4.341364	0.0	4.129430	2207.393630	0.024032	350.426322	14.198767	699.233593	679.113
2017-08-30	4.257734	0.0	4.737399	2226.511472	0.007824	374.774134	13.168789	725.111580	744.030
2017-08-31	4.401758	0.0	5.471875	2162.910664	0.000000	378.558941	15.627504	662.622662	710.884

16 rows × 1782 columns

In [38]:
```
ypred = ypred.stack(['store_nbr', 'family'])
ypred
```

Out[38]:

			sales
	store_nbr	family	
2017-08-16	**1**	**AUTOMOTIVE**	4.087050
		BABY CARE	0.000000
		BEAUTY	4.541333
		BEVERAGES	2263.676462
		BOOKS	0.083917
...	**...**	**...**	...
2017-08-31	**9**	**POULTRY**	335.475052
		PREPARED FOODS	114.244708
		PRODUCE	1232.600247
		SCHOOL AND OFFICE SUPPLIES	136.152000
		SEAFOOD	8.912933

28512 rows × 1 columns

Submission

In [39]:
```
sub = pd.read_csv('D:\\old data\\Download Folder\\store-sales-time-series-forecasting\\sar
sub['sales'] = ypred.values
sub.to_csv('D:\\old data\\Download Folder\\store-sales-time-series-forecasting\\submissior
sub
```

Out[39]:

	id	sales

	id	sales
0	3000888	4.087050
1	3000889	0.000000
2	3000890	4.541333
3	3000891	2263.676462
4	3000892	0.083917
...	...	...
28507	3029395	335.475052
28508	3029396	114.244708
28509	3029397	1232.600247
28510	3029398	136.152000
28511	3029399	8.912933

28512 rows × 2 columns

Thank you!!!