Tatsiana_Sokalava}_code

July 29, 2021

1 Tatsiana Sokalava

1.1 Time Series Forecasting

[1]:
```
import numpy as np
import pandas as pd
```

[2]:
```
import requests
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

[3]:
```
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/00352/
↪Online%20Retail.xlsx"
resp = requests.get(url)
with open('test.xlsx','wb') as output:
    output.write(resp.content)
```

[4]:
```
df=pd.read_excel('test.xlsx')
```

[5]:
```
df.head()
```

[5]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6

          InvoiceDate  UnitPrice  CustomerID         Country
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
```

[6]:
```
# The dataframe contains 541,909 observations and 8 features. There are missing␣
↪values for Description and CustomerID.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
```

[7]:
```
#Let's shift InvoiceDate as our index and convert County column to a␣
↪categorical variable
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate']).dt.date
```

[8]:
```
df1=df.set_index('InvoiceDate')
```

[9]:
```
df1['Country']=df1['Country'].astype('category')
df1['Country'].value_counts().head(10)
```

[9]:
```
United Kingdom    495478
Germany             9495
France              8557
EIRE                8196
Spain               2533
Netherlands         2371
Belgium             2069
Switzerland         2002
Portugal            1519
Australia           1259
Name: Country, dtype: int64
```

[10]:
```
df1['InvoiceNo']=df1['InvoiceNo'].astype(str)
```

[11]:
```
# It is evident that there are missing values in CustomerID field.
# Also, this dataset may need to be adjusted for non-positive observations of␣
↪UnitPrice and Quantities.
df1.describe()
```

[11]:
```
             Quantity      UnitPrice     CustomerID
count  541909.000000  541909.000000  406829.000000
mean        9.552250       4.611114   15287.690570
std       218.081158      96.759853    1713.600303
min    -80995.000000  -11062.060000   12346.000000
25%         1.000000       1.250000   13953.000000
50%         3.000000       2.080000   15152.000000
75%        10.000000       4.130000   16791.000000
max     80995.000000   38970.000000   18287.000000
```

[12]:
```
df1.head()
```

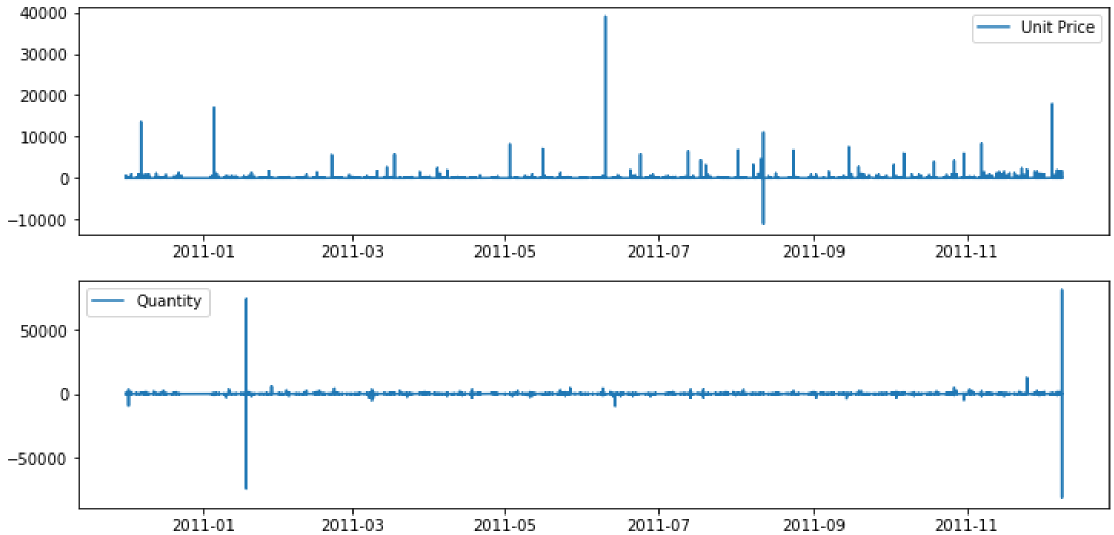
[12]:
```
            InvoiceNo StockCode                          Description  \
InvoiceDate
2010-12-01     536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER
2010-12-01     536365     71053                  WHITE METAL LANTERN
2010-12-01     536365    84406B       CREAM CUPID HEARTS COAT HANGER
2010-12-01     536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE
2010-12-01     536365    84029E       RED WOOLLY HOTTIE WHITE HEART.

             Quantity  UnitPrice  CustomerID         Country
InvoiceDate
2010-12-01          6       2.55     17850.0  United Kingdom
2010-12-01          6       3.39     17850.0  United Kingdom
2010-12-01          8       2.75     17850.0  United Kingdom
2010-12-01          6       3.39     17850.0  United Kingdom
2010-12-01          6       3.39     17850.0  United Kingdom
```

[13]:
```
df1.tail()
```

[13]:
```
            InvoiceNo StockCode                      Description  Quantity  \
InvoiceDate
2011-12-09     581587     22613      PACK OF 20 SPACEBOY NAPKINS        12
2011-12-09     581587     22899     CHILDREN'S APRON DOLLY GIRL          6
2011-12-09     581587     23254    CHILDRENS CUTLERY DOLLY GIRL          4
2011-12-09     581587     23255  CHILDRENS CUTLERY CIRCUS PARADE         4
2011-12-09     581587     22138    BAKING SET 9 PIECE RETROSPOT          3

             UnitPrice  CustomerID Country
InvoiceDate
2011-12-09        0.85     12680.0  France
2011-12-09        2.10     12680.0  France
2011-12-09        4.15     12680.0  France
2011-12-09        4.15     12680.0  France
2011-12-09        4.95     12680.0  France
```

[14]:
```
# Plot UnitPrice and Quantity
fig, ax =plt.subplots(2, figsize=(12,6))
ax[0].plot(df1['UnitPrice'], label = 'Unit Price')
ax[0].legend()
ax[1].plot(df1['Quantity'], label = 'Quantity')
ax[1].legend()
#In the second graph, there are two symmetrical spikes, which needs to be␣
↪further investigated
```

[14]: <matplotlib.legend.Legend at 0x7fa43d380190>



[15]:
```
# Check how many records have zero or less unit price
len(df1[df1.UnitPrice<=0])
```

[15]: 2517

[16]:
```
df1[df1.UnitPrice<=0].Description.isna().sum()
```

[16]: 1454

[17]:
```
df1[df1.UnitPrice<=0].Description.value_counts()
```

[17]:
```
check                                   159
?                                        47
damages                                  45
damaged                                  43
found                                    25
                                       …
HEART OF WICKER LARGE                     1
```

```
?sold as sets?                          1
CERAMIC HEART FAIRY CAKE MONEY BANK     1
MINI CAKE STAND  HANGING STRAWBERY      1
WATERING CAN PINK BUNNY                 1
Name: Description, Length: 377, dtype: int64
```

[18]:
```
# Remove it from the dataset
df1=df1[df1.UnitPrice>0]
df1.head()
```

[18]:
```
            InvoiceNo StockCode                          Description  \
InvoiceDate
2010-12-01     536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER
2010-12-01     536365     71053                  WHITE METAL LANTERN
2010-12-01     536365    84406B       CREAM CUPID HEARTS COAT HANGER
2010-12-01     536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE
2010-12-01     536365    84029E       RED WOOLLY HOTTIE WHITE HEART.

             Quantity  UnitPrice  CustomerID         Country
InvoiceDate
2010-12-01          6       2.55     17850.0  United Kingdom
2010-12-01          6       3.39     17850.0  United Kingdom
2010-12-01          8       2.75     17850.0  United Kingdom
2010-12-01          6       3.39     17850.0  United Kingdom
2010-12-01          6       3.39     17850.0  United Kingdom
```

[19]:
```
# Verify what negative quantities are attributed to:
neg=df1[df1.Quantity<0].sort_values(by='Quantity')
neg.head(5)
#It is evident that two spikes are the cancelled invoices 581484 and 23166.
```

[19]:
```
            InvoiceNo StockCode                          Description  \
InvoiceDate
2011-12-09    C581484     23843          PAPER CRAFT , LITTLE BIRDIE
2011-01-18    C541433     23166       MEDIUM CERAMIC TOP STORAGE JAR
2010-12-02    C536757     84347  ROTATING SILVER ANGELS T-LIGHT HLDR
2011-04-18    C550456     21108   FAIRY CAKE FLANNEL ASSORTED COLOUR
2011-04-18    C550456     21175          GIN + TONIC DIET METAL SIGN

             Quantity  UnitPrice  CustomerID         Country
InvoiceDate
2011-12-09     -80995       2.08     16446.0  United Kingdom
2011-01-18     -74215       1.04     12346.0  United Kingdom
2010-12-02      -9360       0.03     15838.0  United Kingdom
2011-04-18      -3114       2.10     15749.0  United Kingdom
2011-04-18      -2000       1.85     15749.0  United Kingdom
```

[20]:
```
#Since top 5 represent 61% of cancellation quantity, let's look at the top 5␣
↪records to confirm our assumption of symmetry.
neg.head(5).Quantity.sum()/neg.Quantity.sum()
```

[20]:
```
0.6113108576451685
```

[21]:
```
# Inspect the records
[df1[df1.StockCode==x].sort_values(by='Quantity') for x in neg.head(5).
↪StockCode.values]
```

[21]:
```
[            InvoiceNo StockCode                  Description  Quantity  \
 InvoiceDate
 2011-12-09    C581484     23843  PAPER CRAFT , LITTLE BIRDIE    -80995
 2011-12-09     581483     23843  PAPER CRAFT , LITTLE BIRDIE     80995

              UnitPrice  CustomerID         Country
 InvoiceDate
 2011-12-09        2.08     16446.0  United Kingdom
 2011-12-09        2.08     16446.0  United Kingdom  ,
             InvoiceNo StockCode                     Description  Quantity  \
 InvoiceDate
 2011-01-18    C541433     23166  MEDIUM CERAMIC TOP STORAGE JAR    -74215
 2011-06-20    C557508     23166  MEDIUM CERAMIC TOP STORAGE JAR      -240
 2011-08-04    C562375     23166  MEDIUM CERAMIC TOP STORAGE JAR       -12
 2011-10-12    C570867     23166  MEDIUM CERAMIC TOP STORAGE JAR       -12
 2011-05-24    C554527     23166  MEDIUM CERAMIC TOP STORAGE JAR        -9
 …                …       …                              …       …
 2011-05-12     552882     23166  MEDIUM CERAMIC TOP STORAGE JAR        96
 2011-07-24     561051     23166  MEDIUM CERAMIC TOP STORAGE JAR       144
 2011-05-18     553607     23166  MEDIUM CERAMIC TOP STORAGE JAR       240
 2011-07-31     561901     23166  MEDIUM CERAMIC TOP STORAGE JAR       288
 2011-01-18     541431     23166  MEDIUM CERAMIC TOP STORAGE JAR     74215

              UnitPrice  CustomerID         Country
 InvoiceDate
 2011-01-18        1.04     12346.0  United Kingdom
 2011-06-20        1.04     16684.0  United Kingdom
 2011-08-04        1.25     14911.0            EIRE
 2011-10-12        1.25     12607.0             USA
 2011-05-24        1.04     15251.0  United Kingdom
 …                 …         …           …
 2011-05-12        1.04     14646.0     Netherlands
 2011-07-24        1.04     16684.0  United Kingdom
 2011-05-18        1.04     16684.0  United Kingdom
 2011-07-31        1.25     14156.0            EIRE
 2011-01-18        1.04     12346.0  United Kingdom
```

```
[260 rows x 7 columns],
            InvoiceNo StockCode                          Description  \
InvoiceDate
2010-12-02    C536757     84347  ROTATING SILVER ANGELS T-LIGHT HLDR
2010-12-06    C537251     84347  ROTATING SILVER ANGELS T-LIGHT HLDR
2011-01-05    C540164     84347  ROTATING SILVER ANGELS T-LIGHT HLDR
2011-10-24    C572473     84347  ROTATING SILVER ANGELS T-LIGHT HLDR
2011-02-28     545217     84347  ROTATING SILVER ANGELS T-LIGHT HLDR
…               …       …                                   …
2010-12-10     538191     84347  ROTATING SILVER ANGELS T-LIGHT HLDR
2010-12-02     536784     84347  ROTATING SILVER ANGELS T-LIGHT HLDR
2010-12-15     538998     84347  ROTATING SILVER ANGELS T-LIGHT HLDR
2011-10-24     572325     84347  ROTATING SILVER ANGELS T-LIGHT HLDR
2011-11-22     577822     84347  ROTATING SILVER ANGELS T-LIGHT HLDR

             Quantity  UnitPrice  CustomerID         Country
InvoiceDate
2010-12-02      -9360       0.03     15838.0  United Kingdom
2010-12-06         -9       2.55         NaN  United Kingdom
2011-01-05         -6       2.55     14911.0            EIRE
2011-10-24         -1       2.55     18188.0  United Kingdom
2011-02-28          1       4.96         NaN  United Kingdom
…               …        …         …             …
2010-12-10        240       1.88     15061.0  United Kingdom
2010-12-02        240       1.88     15061.0  United Kingdom
2010-12-15        480       1.88     15061.0  United Kingdom
2011-10-24        600       1.74     14607.0  United Kingdom
2011-11-22        600       1.74     14607.0  United Kingdom

[475 rows x 7 columns],
            InvoiceNo StockCode                         Description  Quantity
\
InvoiceDate
2011-04-18    C550456     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR     -3114
2011-11-22    C577832     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR       -18
2011-12-01    C580131     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR       -18
2011-07-28     561658     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR         1
2011-06-20     557502     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR         1
…               …       …                                  …       …
2011-11-23     578125     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR        90
2011-11-14     576180     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR       180
2011-11-09     575296     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR       540
2011-01-11     540815     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR      3114
2011-04-18     550461     21108  FAIRY CAKE FLANNEL ASSORTED COLOUR      3114

             UnitPrice  CustomerID         Country
InvoiceDate
```

```
2011-04-18        2.10     15749.0  United Kingdom
2011-11-22        0.79     18274.0  United Kingdom
2011-12-01        2.10     17576.0  United Kingdom
2011-07-28        2.55     12743.0     Unspecified
2011-06-20        4.96         NaN  United Kingdom
…                 …           …          …
2011-11-23        0.79     17511.0  United Kingdom
2011-11-14        0.79     13694.0  United Kingdom
2011-11-09        0.79     16041.0  United Kingdom
2011-01-11        2.10     15749.0  United Kingdom
2011-04-18        2.10     15749.0  United Kingdom

[270 rows x 7 columns],
            InvoiceNo StockCode                 Description  Quantity  \
InvoiceDate
2011-04-18    C550456     21175  GIN + TONIC DIET METAL SIGN     -2000
2011-10-10    C570290     21175  GIN + TONIC DIET METAL SIGN       -12
2011-06-13    C556647     21175  GIN + TONIC DIET METAL SIGN       -12
2011-05-27    C554870     21175  GIN + TONIC DIET METAL SIGN        -3
2010-12-10    C538350     21175  GIN + TONIC DIET METAL SIGN        -1
…               …        …                             …       …
2011-07-14     560080     21175  GIN + TONIC DIET METAL SIGN       192
2011-09-20     567458     21175  GIN + TONIC DIET METAL SIGN       192
2011-11-21     577747     21175  GIN + TONIC DIET METAL SIGN       240
2011-01-11     540815     21175  GIN + TONIC DIET METAL SIGN      2000
2011-04-18     550461     21175  GIN + TONIC DIET METAL SIGN      2000

             UnitPrice  CustomerID         Country
InvoiceDate
2011-04-18        1.85     15749.0  United Kingdom
2011-10-10        2.55     14665.0  United Kingdom
2011-06-13        2.55     13012.0  United Kingdom
2011-05-27        2.55     15078.0  United Kingdom
2010-12-10        1.85     13798.0  United Kingdom
…                 …           …          …
2011-07-14        2.08     17450.0  United Kingdom
2011-09-20        2.66     17450.0  United Kingdom
2011-11-21        2.67     17450.0  United Kingdom
2011-01-11        1.85     15749.0  United Kingdom
2011-04-18        1.69     15749.0  United Kingdom

[825 rows x 7 columns]]
```

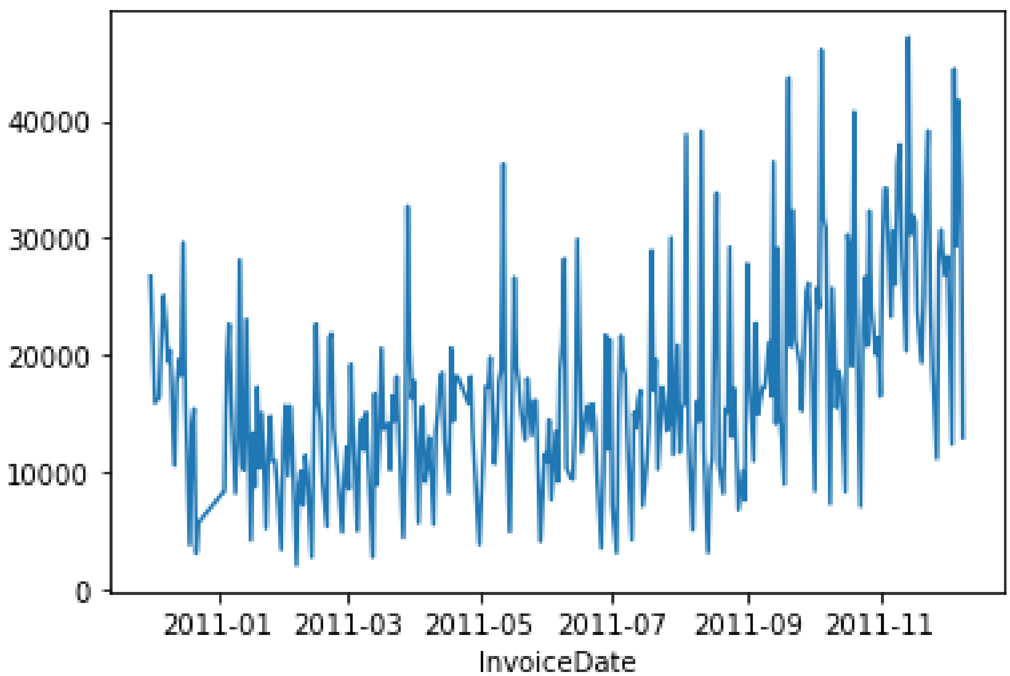
[22]:
```
#By looking at top 5, it is reasonable to assume that the majority of cancelled
↪transactions happened on the same day.
# The exception is item 84347, which doesn't have a reasonable explanation
↪without questioning the business users.
```

```
# For now, we will not be removing cancelled transaction, since many are voided␣
↪out duaring the aggregation process by day.
# However, we will define a function and remove transactions that have a␣
↪negative quantity, since it seems mostly due to the damages and write offs.
```

[23]:
```
def remove_writeoffs(df):
    qnegat=[x for x in df.InvoiceNo if x.startswith('C')]
    return df[(df.Quantity>0) | (df.InvoiceNo.isin(qnegat))].
↪sort_values(by='Quantity')
df1=remove_writeoffs(df1)
```

[24]:
```
# View daily sales volume by day
df2=df1.groupby(['InvoiceDate'])['Quantity'].sum()
df2.plot()

# The plot doesn't look stationary. While we take a note of it, we will coninue␣
↪to find our top three items.
```

[24]: <AxesSubplot:xlabel='InvoiceDate'>



[25]:
```
# Check how many quantity of products have been sold online from each country
a = df1['Quantity'].groupby(df1['Country']).agg('sum').sort_values(ascending =␣
↪False)[0:].head(10)
print(a)
```

```
Country
United Kingdom    4399357
Netherlands        199552
EIRE               142363
Germany            117446
France             110479
Australia           83345
Sweden              35637
Switzerland         30324
Spain               26813
Japan               25218
Name: Quantity, dtype: int64
```

[26]:
```
# Since the dates range from 12/01/2010-12/09/2011, it looks like there are␣
↪missing data for some days
len(df2.index)
```

[26]: 305

[27]:
```
# Define a function to reinstate dates
import datetime
def zero_sales(df):
    idx = pd.date_range(df.index.min(),datetime.date(2011,12,9))
    return df.reindex(idx, fill_value=0)
```

[28]:
```
df2=zero_sales(df2)
```

[29]:
```
# Review which days of the week we have no sales reported
pd.DataFrame(df2.index[df2.values==0])[0].dt.day_name().value_counts()
```

[29]:
```
Saturday     53
Monday        6
Friday        4
Sunday        3
Thursday      1
Wednesday     1
Tuesday       1
Name: 0, dtype: int64
```

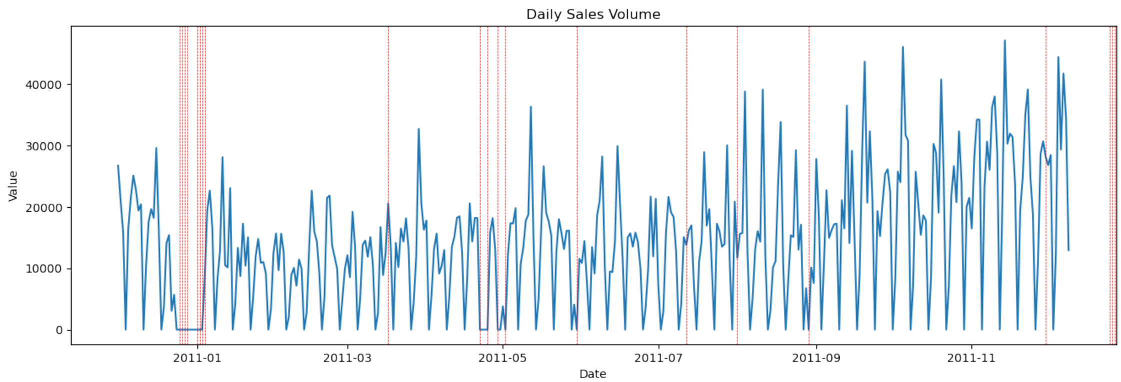
[30]:
```
#!pip install holidays
```

[31]:
```
from datetime import date
import holidays
```

[32]:
```
uk_holidays=pd.Series(holidays.UnitedKingdom(years= [2010,2011] ).keys())
```

[33]:
```
# Plot data with holidays
from matplotlib import pylab
```

```
def plot_df(data, x, y, title="", xlabel='Date', ylabel='Value', dpi=100):
    plt.figure(figsize=(16,5), dpi=dpi)
    plt.plot(x, y, color='tab:blue')
    plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)
    holidays = uk_holidays[uk_holidays.between(data.index.min(), datetime.
 ↪date(2011, 12, 31))]
    [pylab.axvline(_x, linewidth=0.5, color='r', ls='--') for _x in holidays]
    plt.show()

plot_df(df2, x=df2.index, y=df2.values, title='Daily Sales Volume')
```



[34]:
```
# Define a week between 2011-11-27 and 2011-12-3 and identify the top 3 per␣
 ↪objective
week=df1[(df1.index>=datetime.date(2011,11,27) ) & (df1.index<=datetime.
 ↪date(2011,12,3))]
top3=week[['StockCode','Quantity']].groupby('StockCode').sum().
 ↪sort_values(by='Quantity', ascending=False).head(3)
top3
```

[34]:
```
           Quantity
StockCode
23084          4588
22197          3195
23582          1851
```

[35]:
```
# Since the dataset has various descriptions per SKU, let's review the␣
 ↪Descriptions:
df1[df1['StockCode']==top3.index[0]]['Description'].unique()
```

[35]:
```
array(['RABBIT NIGHT LIGHT'], dtype=object)
```

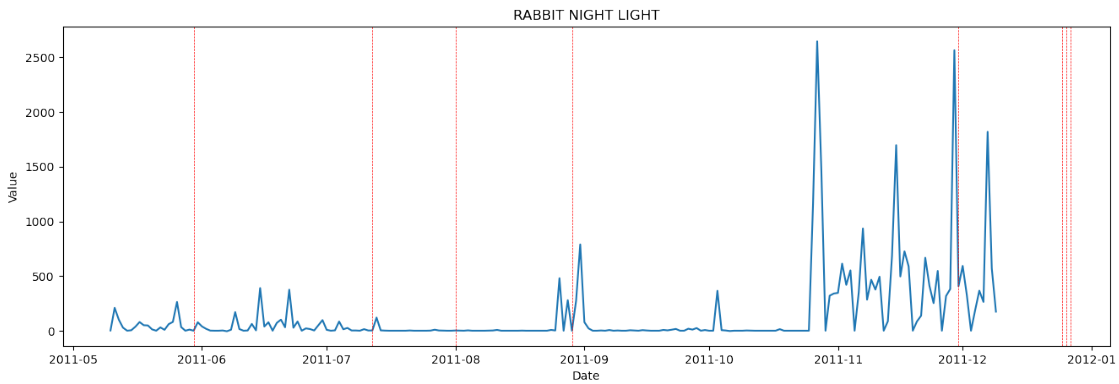
[36]:
```
df1[df1['StockCode']==top3.index[1]]['Description'].unique()
```

[36]: array(['POPCORN HOLDER', 'SMALL POPCORN HOLDER'], dtype=object)

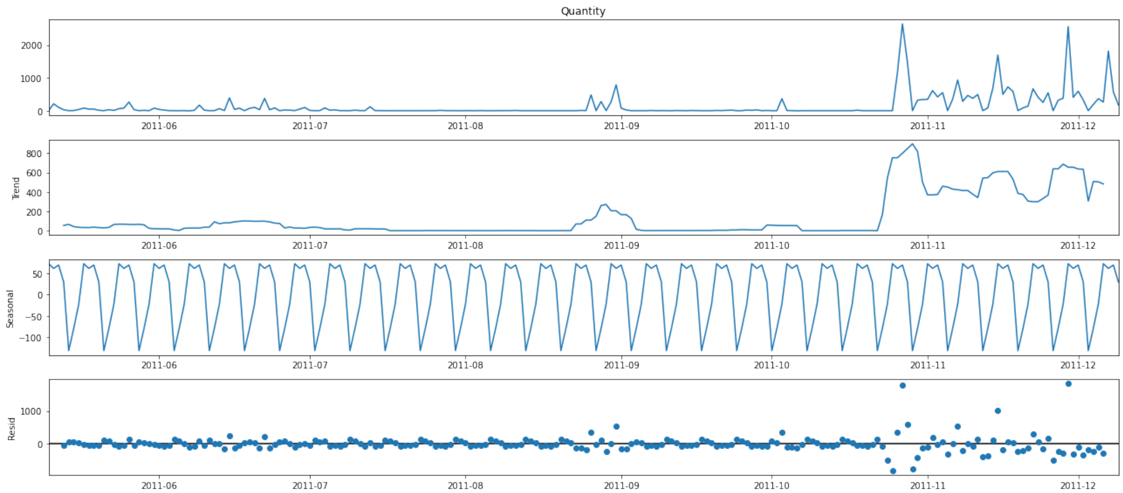
[37]:
```
df1[df1['StockCode']==top3.index[2]]['Description'].unique()
```

[37]: array(['VINTAGE DOILY JUMBO BAG RED '], dtype=object)

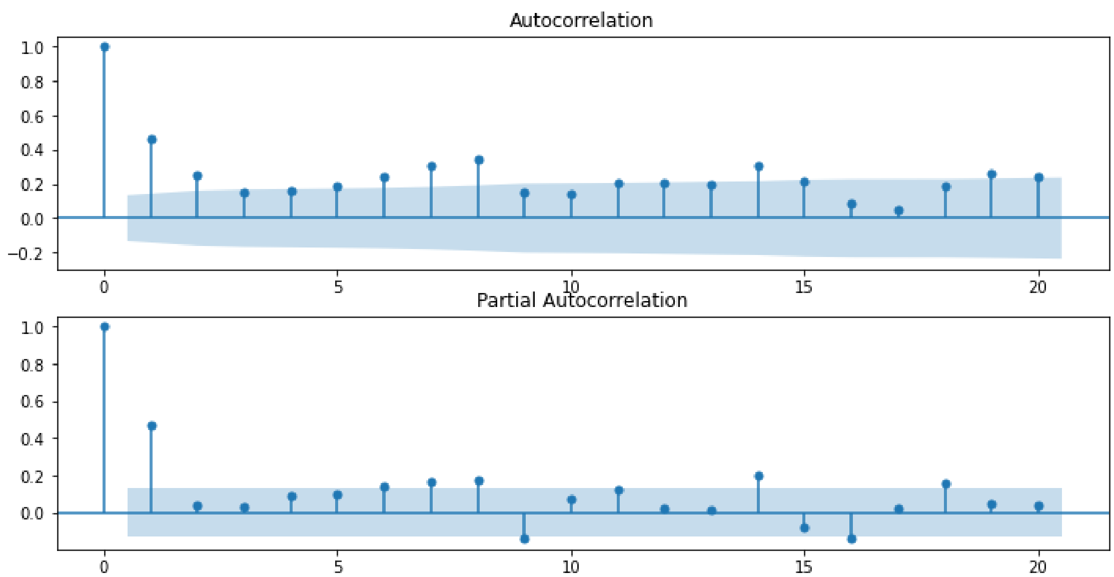
[38]:
```
# Plot the sales quantity timeseries of each item
rabl=df1[df1.StockCode==top3.index[0]].sort_values(by='InvoiceDate')
rabl2=zero_sales(rabl.groupby(['InvoiceDate'])['Quantity'].sum())
plot_df(rabl2, x=rabl2.index, y=rabl2.values, title="RABBIT NIGHT LIGHT",␣
↪xlabel='Date', ylabel='Value', dpi=100)
```



[39]:
```
# By looking at the RABBIT NIGHT LIGHT, we can infer that there is an␣
↪increasing sales trend towards the end of the year.
#There is a somewhat seasonality in the observed data. Also, the data may not␣
↪be stationary.
import statsmodels.api as sm
decomposition = sm.tsa.seasonal_decompose(rabl2, model='additive')
from pylab import rcParams
rcParams['figure.figsize'] = 18, 8
fig = decomposition.plot()
plt.show()
```

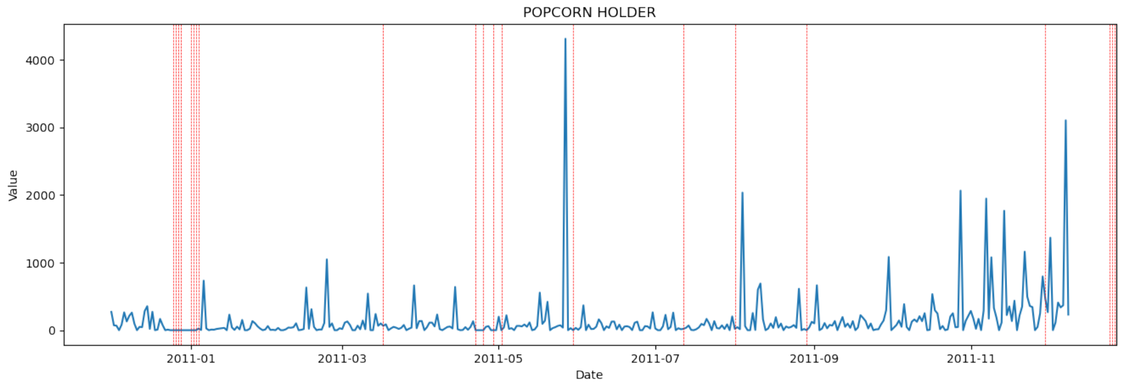


[40]:
```python
# Review autocorrelation function plots
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
fig, ax = plt.subplots(2, figsize=(12,6))
ax[0]=plot_acf(rabl2, ax=ax[0], lags=20)
ax[1]=plot_pacf(rabl2, ax=ax[1], lags=20)
```

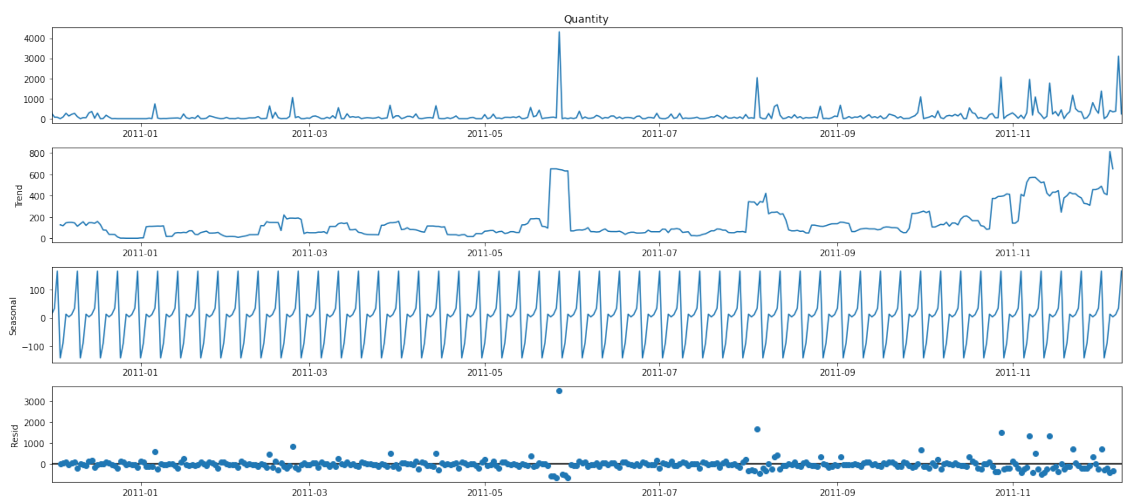


[41]:
```python
# Plot the Popcorn Holder data
poph=df1[df1.StockCode==top3.index[1]].sort_values(by='InvoiceDate')
poph2=zero_sales(poph.groupby(['InvoiceDate'])['Quantity'].sum())
```

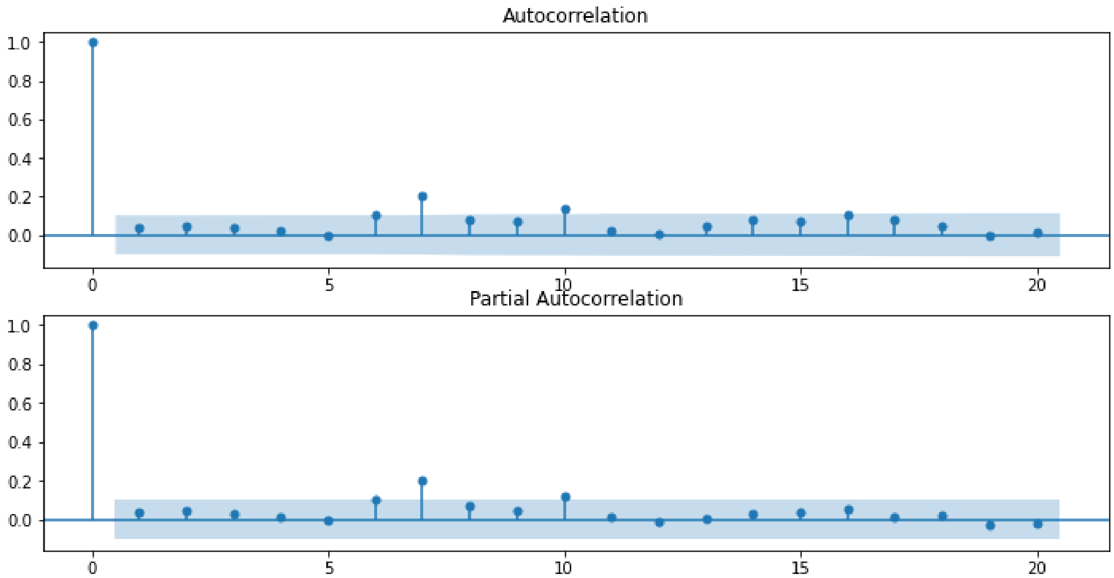
```
plot_df(poph2, x=poph2.index, y=poph2.values, title="POPCORN HOLDER",␣
↪xlabel='Date', ylabel='Value', dpi=100)
```



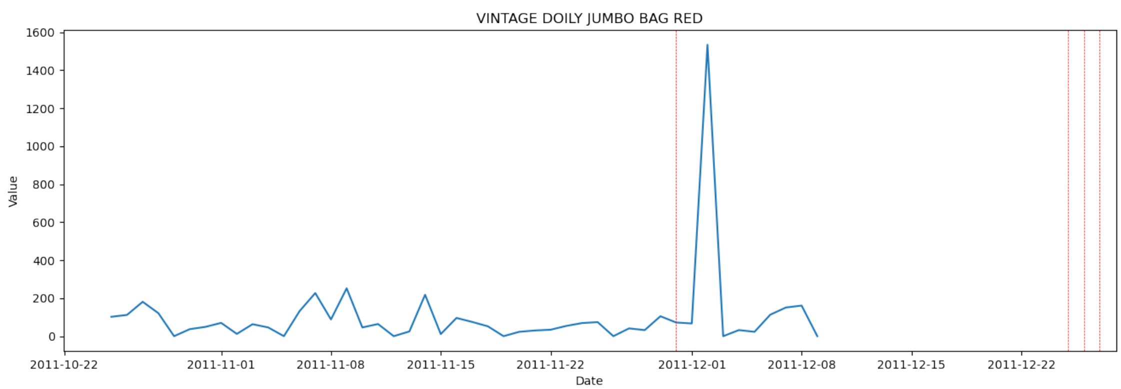
[42]:
```
# By looking at it, we can also infer that there is an unusual spike around␣
↪June and an increasing sales variation towards the end of the year.
import statsmodels.api as sm
from pylab import rcParams
decomposition = sm.tsa.seasonal_decompose(poph2, model='additive')
rcParams['figure.figsize'] = 18, 8
fig = decomposition.plot()
plt.show()
```



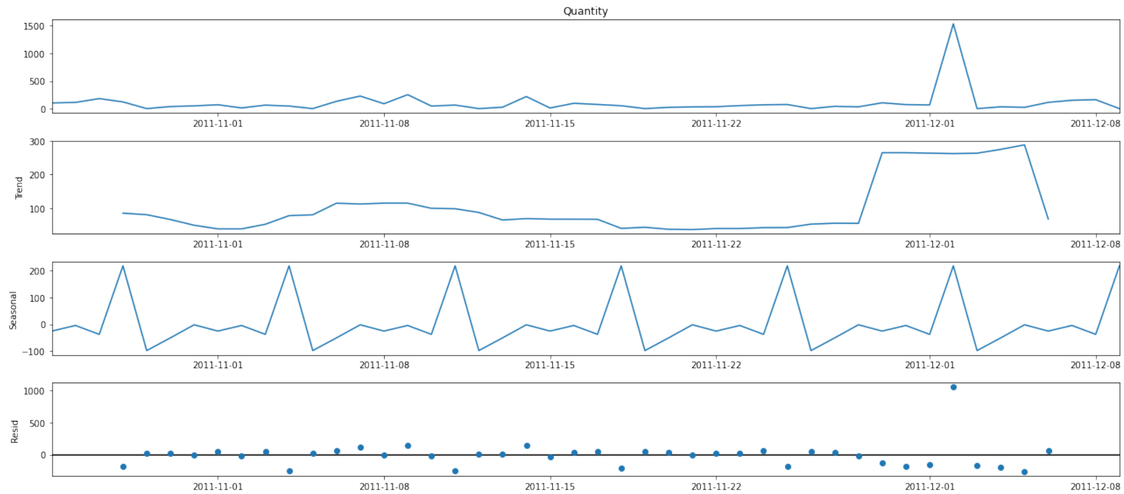
[43]:
```
fig, ax = plt.subplots(2, figsize=(12,6))
ax[0]=plot_acf(poph2, ax=ax[0], lags=20)
ax[1]=plot_pacf(poph2, ax=ax[1], lags=20)
```



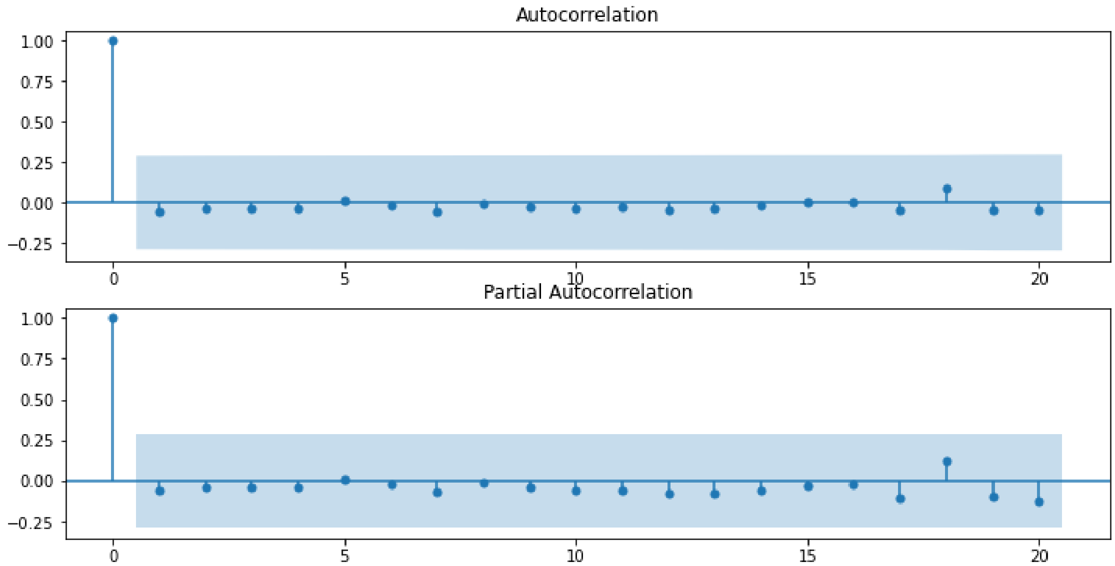
[44]:
```
# Plot sales of the VINTAGE DOILY JUMBO BAG RED
vinb=df1[df1.StockCode==top3.index[2]].sort_values(by='InvoiceDate')
vinb2=zero_sales(vinb.groupby(['InvoiceDate'])['Quantity'].sum())
plot_df(vinb2, x=vinb2.index, y=vinb2.values, title="VINTAGE DOILY JUMBO BAG␣
↪RED", xlabel='Date', ylabel='Value', dpi=100)
```



[45]:
```
# By looking at the VINTAGE DOILY JUMBO BAG RED, we can also infer that there␣
↪is an increasing trend towards the end of the year.
decomposition = sm.tsa.seasonal_decompose(vinb2, model='additive')
rcParams['figure.figsize'] = 18, 8
fig = decomposition.plot()
plt.show()
```



[46]:
```
fig, ax = plt.subplots(2, figsize=(12,6))
ax[0]=plot_acf(vinb2, ax=ax[0], lags=20)
ax[1]=plot_pacf(vinb2, ax=ax[1], lags=20)
#The data is stationary and appears to have daily and weekly seasonality alog with the increasing trend.
```



[47]:
```
#It is evident that three items have a different history of sales: Over a year for POPCORN HOLDER, less then a year
#for RABBIT NIGHT LIGHT, and a month and a half for VINTAGE DOILY JUMBO BAG RED.
```

2 Models

[48]:
```
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot

# Begin with modeling the POPCORN HOLDER data

poph2_train=poph2[poph2.index<'2011-11-27']
poph2_test=poph2[poph2.index>='2011-11-27']

#rabl2_train=rabl2[rabl2.index<'2011-11-27']
#rabl2_test=rabl2[rabl2.index>='2011-11-27']

#vinb2_train=vinb2[vinb2.index<'2011-11-27']
#vinb2_test=vinb2[vinb2.index>='2011-11-27']
```

[49]:
```
train_dates, test_dates = poph2_train.index, poph2_test.index
train_data, test_data = poph2_train.values, poph2_test.values

#train_dates, test_dates = rabl2_train.index, rabl2_test.index
#train_data, test_data = rabl2_train.values, rabl2_test.values

#train_dates, test_dates = vinb2_train.index, vinb2_test.index
#train_data, test_data = vinb2_train.values, vinb2_test.values
```

[50]:
```
# Define a dataframe to view the performance of fitted models
perform=pd.DataFrame()
perform.index.name='Models Popcorn Holder'
perform['RMSE']=None
perform['Parameters']=None
```

[51]:
```
# Define a function for RMSE
from sklearn.metrics import mean_squared_error
def rmse(actual, predicted):
    rmse=np.sqrt(mean_squared_error(actual,predicted))
    return rmse
```

[52]:
```
# Define a plot function for actual vs predicted:
import plotly.graph_objects as go
def plot_actual_predicted(actual, predicted, model_name):
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=test_dates, y=actual, name = "Expected", line =␣
↪dict(color = '#17BECF'), opacity = 0.8))
    fig.add_trace(go.Scatter(x=test_dates, y=predicted, name = model_name, line␣
↪= dict(color = '#CF1717'), opacity = 0.8))
    fig.show()
```

2.1 Model 1: Moving Average

[53]:
```
# Begin selecting the best window size

X=poph2.values
for a in range(1,13):
    window = a
    history = [X[i] for i in range(window)]
    test = [X[i] for i in range(window, (len(X)-14))]
    predictions = list()
    # walk forward over time steps in test
    for t in range(len(test)):
        length = len(history)
        preds = np.mean([history[i] for i in range(length-window,length)])
        obs = test[t]
        predictions.append(preds)
        history.append(obs)
    error = rmse(test, predictions)
    print ('RMSE: %.3f' % round(error), 'Window size: %.3f' % a)
```

```
RMSE: 476.000 Window size: 1.000
RMSE: 409.000 Window size: 2.000
RMSE: 389.000 Window size: 3.000
RMSE: 376.000 Window size: 4.000
RMSE: 370.000 Window size: 5.000
RMSE: 364.000 Window size: 6.000
RMSE: 352.000 Window size: 7.000
RMSE: 350.000 Window size: 8.000
RMSE: 349.000 Window size: 9.000
RMSE: 346.000 Window size: 10.000
RMSE: 346.000 Window size: 11.000
RMSE: 346.000 Window size: 12.000
```

[54]:
```
# Apply a window size of 10 and predict
window = 10
predict=pd.DataFrame(X)[0].rolling(window).mean()[-13:]
observ=pd.DataFrame(X)[0][-13:]
error_ma = rmse(observ, predict)
print('RMSE error: %.3f' % error_ma, 'Window size: %.3f' % window)
```

```
RMSE error: 745.817 Window size: 10.000
```

[55]:
```
plot_actual_predicted(observ, predict, "Moving Average Predictions")
```

[56]:
```
perform.loc['Moving Average', ('RMSE','Parameters')]= round(error_ma,0),␣
 ↪'Window size: 10'
perform
```

[56]:

	RMSE	Parameters
Models Popcorn Holder		
Moving Average	746	Window size: 10

2.2 Model 2: Exponential Smoothing

[57]:
```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error
```

[58]:
```
#!pip install pmdarima
```

[59]:
```
#from statsmodels.tsa.statespace.sarimax import SARIMAX
#from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
```

[60]:
```
mod_ex=ExponentialSmoothing(np.array(train_data), seasonal_periods=52,␣
↪trend='add', seasonal='add')
```

[61]:
```
fit=mod_ex.fit()
pred_es=fit.forecast(13)
```

[62]:
```
error_es = rmse(test_data, pred_es)
print('RMSE: %.3f' % error_es)
```

```
RMSE: 781.167
```

[63]:
```
perform.loc['Exponential Smoothing', ('RMSE','Parameters')]= round(error_es),␣
↪'Seasonal_periods =52'
perform
```

[63]:

	RMSE	Parameters
Models Popcorn Holder		
Moving Average	746	Window size: 10
Exponential Smoothing	781	Seasonal_periods =52

[64]:
```
plot_actual_predicted(test_data, pred_es, "Exponential Smoothing Prediction")
```

2.3 Model 3: Arima

[65]:
```
#!pip install pmdarima
from pmdarima.arima import auto_arima
from statsmodels.tsa.stattools import adfuller
```

[66]:
```
# Let's use auto_arima. We know our data shows seasonality and needs␣
↪differencing.
stepwise_model = auto_arima(train_data, start_p=1, start_q=1, max_p=3, max_q=3,␣
↪m=12, start_P=0, d=1, D=1, seasonal=True,
```

```
                                trace=True, error_action='ignore',␣
↪suppress_warnings=True, stepwise=True)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=inf, Time=0.54 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=5531.538, Time=0.02 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=5334.731, Time=0.15 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=inf, Time=0.28 sec
 ARIMA(1,1,0)(0,1,0)[12]             : AIC=5445.079, Time=0.03 sec
 ARIMA(1,1,0)(2,1,0)[12]             : AIC=5291.955, Time=0.32 sec
 ARIMA(1,1,0)(2,1,1)[12]             : AIC=inf, Time=0.94 sec
 ARIMA(1,1,0)(1,1,1)[12]             : AIC=inf, Time=0.36 sec
 ARIMA(0,1,0)(2,1,0)[12]             : AIC=5387.740, Time=0.28 sec
 ARIMA(2,1,0)(2,1,0)[12]             : AIC=5256.161, Time=0.44 sec
 ARIMA(2,1,0)(1,1,0)[12]             : AIC=5300.342, Time=0.21 sec
 ARIMA(2,1,0)(2,1,1)[12]             : AIC=inf, Time=1.17 sec
 ARIMA(2,1,0)(1,1,1)[12]             : AIC=inf, Time=0.53 sec
 ARIMA(3,1,0)(2,1,0)[12]             : AIC=5235.364, Time=0.48 sec
 ARIMA(3,1,0)(1,1,0)[12]             : AIC=5282.255, Time=0.27 sec
 ARIMA(3,1,0)(2,1,1)[12]             : AIC=inf, Time=1.38 sec
 ARIMA(3,1,0)(1,1,1)[12]             : AIC=inf, Time=0.63 sec
 ARIMA(3,1,1)(2,1,0)[12]             : AIC=inf, Time=1.28 sec
 ARIMA(2,1,1)(2,1,0)[12]             : AIC=inf, Time=1.25 sec
 ARIMA(3,1,0)(2,1,0)[12] intercept   : AIC=5237.368, Time=0.97 sec

Best model:  ARIMA(3,1,0)(2,1,0)[12]
Total fit time: 11.532 seconds
```

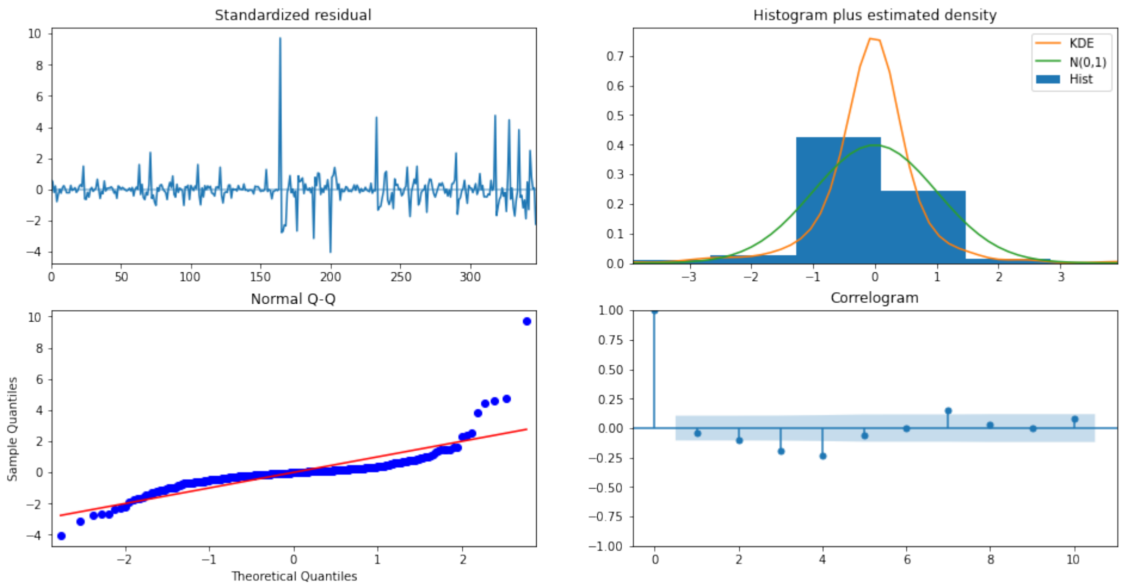
[67]:
```
stepwise_model.fit(train_data)
preds = stepwise_model.predict(n_periods=13)
preds
```

[67]:
```
array([  88.78128757,  118.38078031,   25.54930964,  140.30679137,
        585.01505017,   41.36413624, 1147.21044358,  323.36986013,
        156.51130324,   79.41973066,  139.3972984 ,  543.33842197,
          3.84710817])
```

[68]:
```
error_ar=rmse(test_data, preds)

print('RMSE: %.3f' % error_ar)
```

```
RMSE: 909.169
```

[69]:
```
stepwise_model.plot_diagnostics(figsize=(16, 8))
plt.show()
```



[70]:
```
plot_actual_predicted(test_data, preds, 'Arima Predictions')
```

[71]:
```
perform.loc['Arima', ('RMSE','Parameters')]=round(error_ar), stepwise_model
perform
```

[71]:

	RMSE	Parameters
Models Popcorn Holder		
Moving Average	746	Window size: 10
Exponential Smoothing	781	Seasonal_periods =52
Arima	909	ARIMA(3,1,0)(2,1,0)[12]

2.4 Model 4: XGBoost

[72]:
```
#!pip install xgboost
```

[73]:
```
import xgboost as xgb
from xgboost import DMatrix
from pandas import concat
from numpy import asarray
from xgboost import XGBRegressor
```

[74]:
```
# Define functions to apply XGBoost model:
data=poph2.values

# Transform a time series dataset into a supervised learning dataset
def series_to_supervised(data, n_in, n_out=1, dropnan=True):
    df = pd.DataFrame(data)
    cols = list()
```

```
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
    # forecast out
    for i in range(0, n_out):
        cols.append(df.shift(-i))
    # aggregate
    agg = concat(cols, axis=1)
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg.values

# Split a dataset into train/test sets
def train_test_split(data, n_test):
    return data[:-n_test, :], data[-n_test:, :]

# Fit an xgboost model and make a one step prediction
def xgboost_forecast(train, testX):
    train = asarray(train)
    # split into input and output columns
    trainX, trainY = train[:, :-1], train[:, -1]
    # fit model
    model = XGBRegressor(objective='reg:squarederror', n_estimators=1000,␣
 ↪max_depth=5,
 min_child_weight=1)
    model.fit(trainX, trainY)
    preds = model.predict(asarray([testX]))
    return preds[0]

# Validation
def walk_forward_test(data, n_test):
    predictions = list()
    train, test = train_test_split(data, n_test)
    # add history with training dataset
    history = [x for x in train]
    # step over each time-step in the test set
    for i in range(len(test)):
        # split test row into input and output columns
        testX, testY = test[i, :-1], test[i, -1]
        # make a prediction
        preds = xgboost_forecast(history, testX)
        predictions.append(preds)
        # add actual observation for the next loop to history
        history.append(test[i])
    # estimate error
    error_xgboost = np.sqrt(mean_squared_error(test[:,-1], predictions))
```

```
    return error_xgboost, test[:, -1], predictions
```

[75]:
```
# Transform the time series data into supervised learning
data = series_to_supervised(data, n_in=17)
# Evaluate
error_xgboost, y, preds = walk_forward_test(data, n_test=13)
print('RMSE: %.3f' % error_xgboost)
# Plot expected vs preducted
plot_actual_predicted(y, preds, "XGBoost Predictions")
```

RMSE: 714.233

[76]:
```
perform.loc['XGBoost', ('RMSE','Parameters')]=round(error_xgboost),␣
↪"n_estimators=1000, max_depth=5,min_child_weight=1"
perform
```

[76]:

Models Popcorn Holder	RMSE	Parameters
Moving Average	746	Window size: 10
Exponential Smoothing	781	Seasonal_periods =52
Arima	909	ARIMA(3,1,0)(2,1,0)[12]
XGBoost	714	n_estimators=1000, max_depth=5,min_child_weight=1

2.5 Model 5: FB Prophet

[77]:
```
#!pip install fbprophet
```

[78]:
```
from fbprophet import Prophet
```

[79]:
```
# Prepare data
df_train, df_test  =poph2_train.reset_index(), poph2_test.reset_index()
#df_train, df_test  =rabl2_train.reset_index(), rabl2_test.reset_index()
#df_train, df_test  =vinb2_train.reset_index(), rabl2_test.reset_index()
df_train.columns, df_test.columns = ['ds','y'], ['ds','y']
```

[80]:
```
# Add holidays

holi = pd.DataFrame(list(holidays.UnitedKingdom(years= [2010,2011]).items()))
holi.columns=['ds','holiday']
```

[81]:
```
# Define a model
m=Prophet(holidays=holi, holidays_prior_scale=0.05) #weekly_seasonality=True,␣
↪daily_seasonality=True, yearly_seasonality=False)

m.add_seasonality(name='weekly', period=7, fourier_order=3, prior_scale=5)
m.add_seasonality(name='daily', period=8, fourier_order=3, prior_scale=0.1)
m.add_seasonality(name='yearly', period=6, fourier_order=3, prior_scale=0.1)
```

[81]: <fbprophet.forecaster.Prophet at 0x7fa441cde5e0>

[82]:
```
m.fit(df_train)
```

```
INFO:fbprophet:Found custom seasonality named 'yearly', disabling built-in
'yearly' seasonality.
INFO:fbprophet:Found custom seasonality named 'weekly', disabling built-in
'weekly' seasonality.
INFO:fbprophet:Found custom seasonality named 'daily', disabling built-in
'daily' seasonality.
```

[82]: <fbprophet.forecaster.Prophet at 0x7fa441cde5e0>

[83]:
```
future=m.make_future_dataframe(periods=13, freq='D')
```

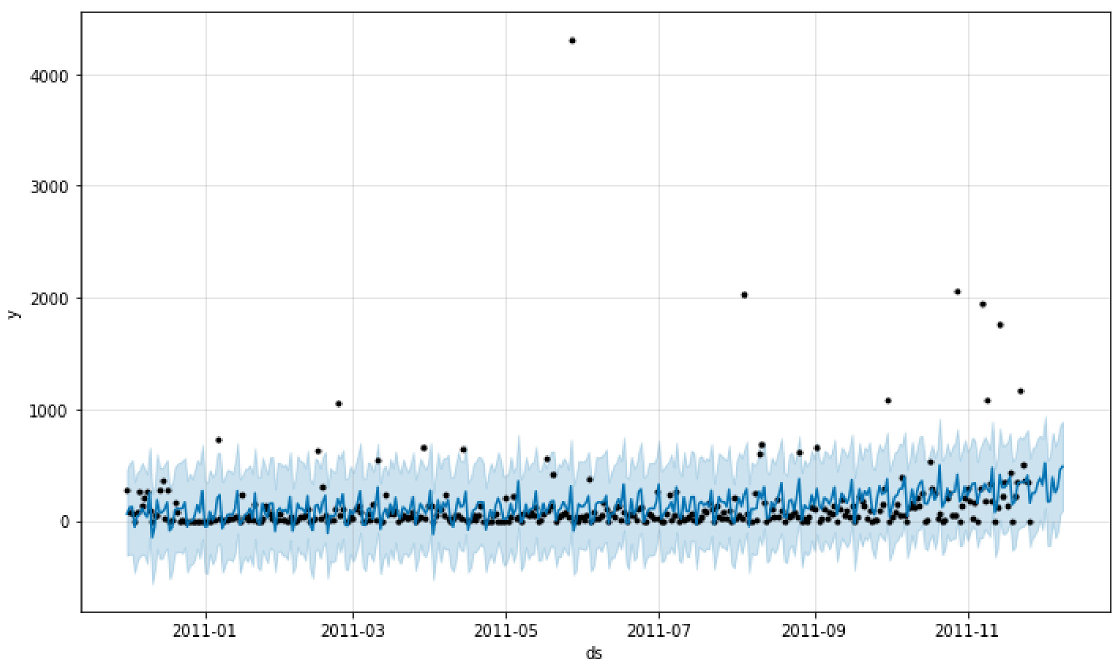
[84]:
```
prophet_pred=m.predict(future)
prophet_pred.tail()
```

[84]:
```
             ds       trend  yhat_lower  yhat_upper  trend_lower  trend_upper  \
369  2011-12-05  318.564071    -7.675460  785.822680   318.383531   318.741389
370  2011-12-06  320.122080 -150.624089  672.610332   319.889838   320.344679
371  2011-12-07  321.680090 -106.786993  730.466544   321.391054   321.955181
372  2011-12-08  323.238099   56.800815  855.213451   322.874798   323.591524
373  2011-12-09  324.796109   91.167244  888.424066   324.362337   325.205168

     Battle of the Boyne [Northern Ireland]  \
369                                     0.0
370                                     0.0
371                                     0.0
372                                     0.0
373                                     0.0

     Battle of the Boyne [Northern Ireland]_lower  \
369                                           0.0
370                                           0.0
371                                           0.0
372                                           0.0
373                                           0.0

     Battle of the Boyne [Northern Ireland]_upper  Boxing Day  …  \
369                                           0.0         0.0  …
370                                           0.0         0.0  …
371                                           0.0         0.0  …
372                                           0.0         0.0  …
373                                           0.0         0.0  …

         weekly  weekly_lower  weekly_upper     yearly  yearly_lower  \
369   26.054554     26.054554     26.054554  27.467111     27.467111
```

```
370   -0.161679     -0.161679     -0.161679  -9.837723     -9.837723
371   12.286547     12.286547     12.286547 -30.887382    -30.887382
372   34.192456     34.192456     34.192456  28.023700     28.023700
373  149.040787    149.040787    149.040787   8.596196      8.596196

     yearly_upper  multiplicative_terms  multiplicative_terms_lower  \
369     27.467111                   0.0                         0.0
370     -9.837723                   0.0                         0.0
371    -30.887382                   0.0                         0.0
372     28.023700                   0.0                         0.0
373      8.596196                   0.0                         0.0

     multiplicative_terms_upper        yhat
369                         0.0  389.003404
370                         0.0  250.120927
371                         0.0  299.609560
372                         0.0  457.182332
373                         0.0  485.263119

[5 rows x 79 columns]
```

[85]:

```python
import matplotlib.pyplot as plt
m.plot(prophet_pred)
plt.show()
```



[86]:
```
plot_actual_predicted(poph2_test.values, prophet_pred[prophet_pred.
 ↪ds>='2011-11-27'].yhat, "Prophet Predictions")
```

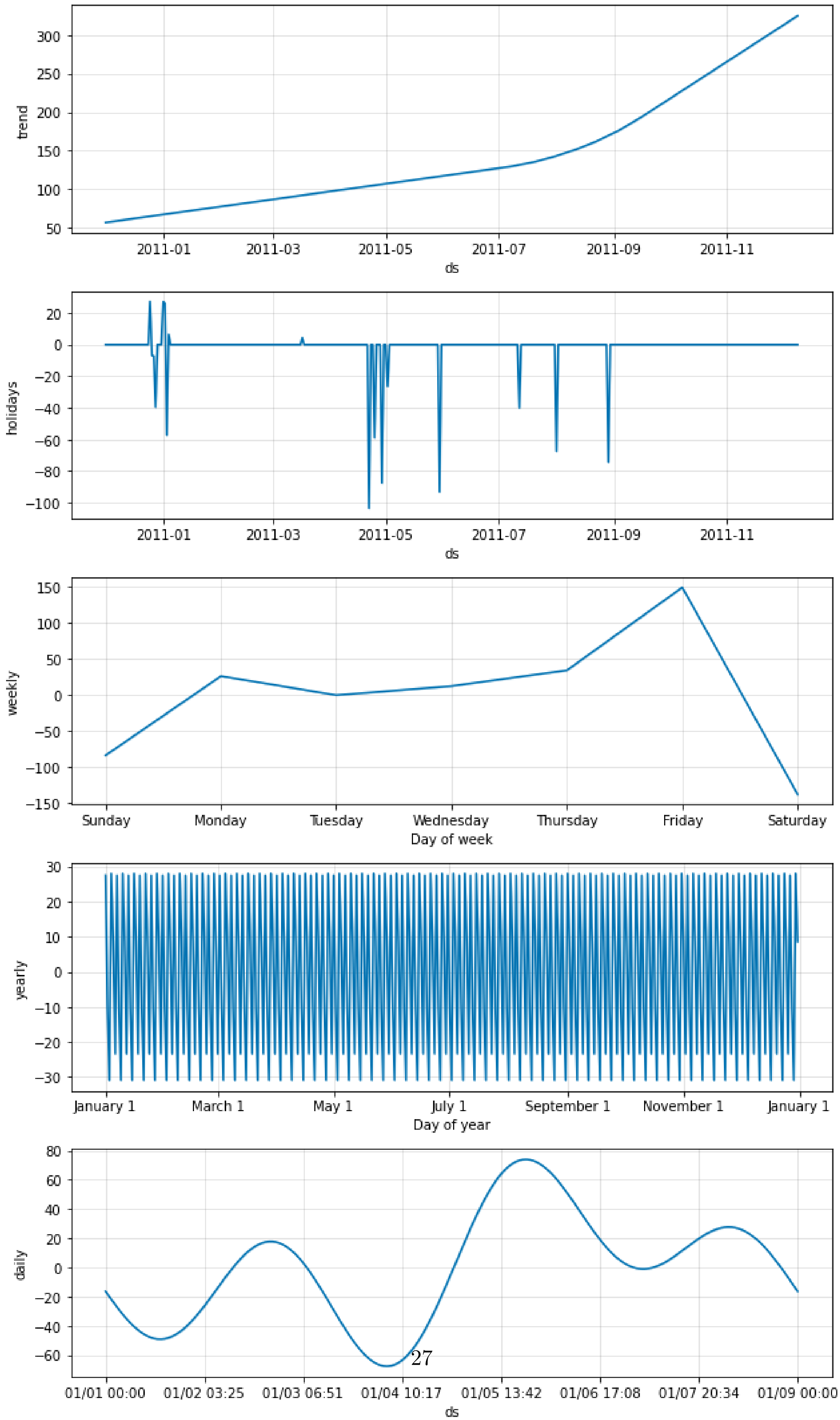
[87]:
```
preds_prophet=prophet_pred[prophet_pred.ds>='2011-11-27']['yhat'].values
```

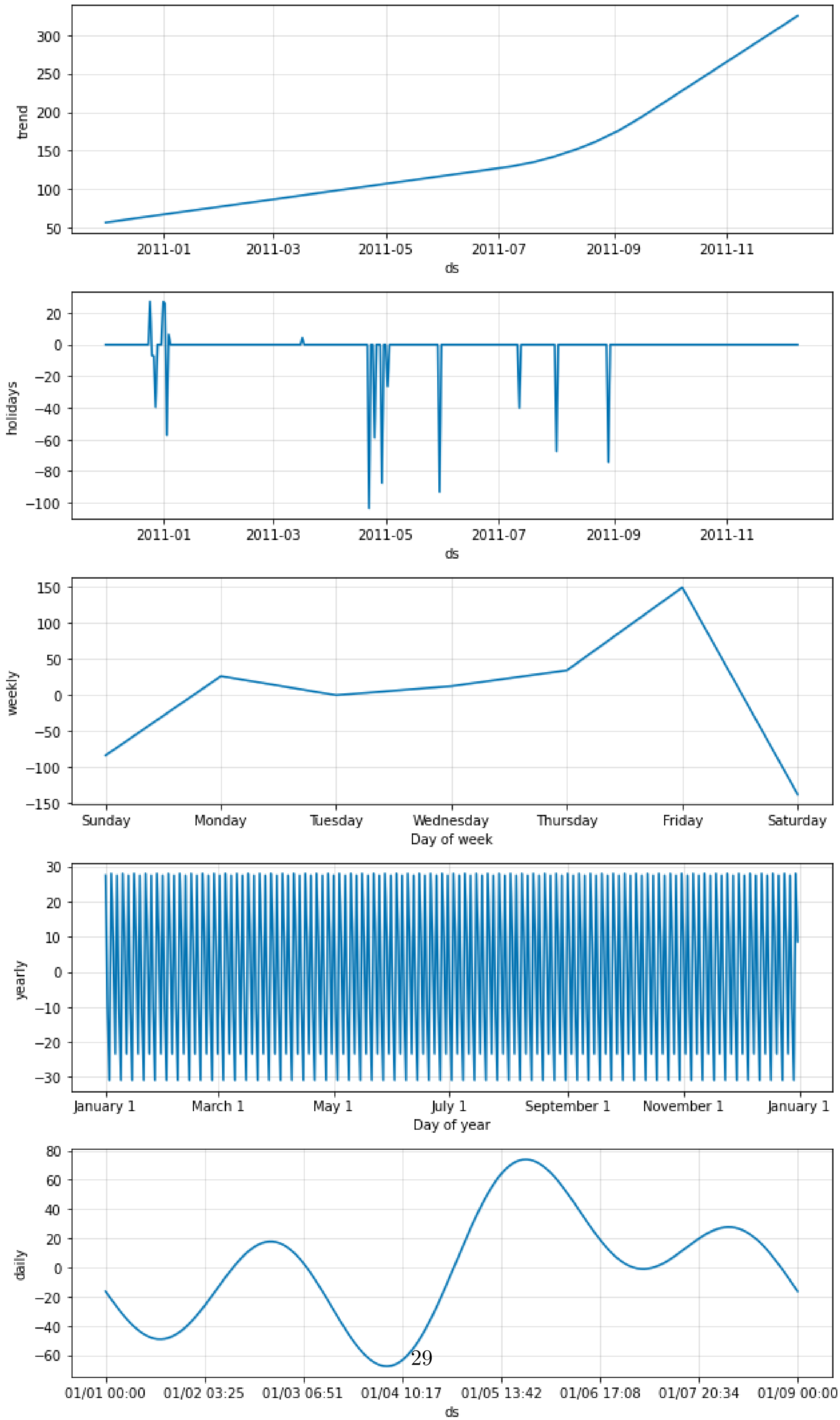
[88]:
```
error_prophet=np.sqrt(mean_squared_error(poph2_test.values, preds_prophet))

print('RMSE: %.3f' % error_prophet)
```

```
RMSE: 790.290
```

[89]:
```
m.plot_components(prophet_pred)
```

[89]:


trend
300
250
200
150
100
50
2011-01
2011-03
2011-05
2011-07
2011-09
2011-11
ds
holidays
20
0
−20
−40
−60
−80
−100
2011-01
2011-03
2011-05
2011-07
2011-09
2011-11
ds
weekly
150
100
50
0
−50
−100
−150
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week
yearly
30
20
10
0
−10
−20
−30
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year
daily
80
60
40
20
0
−20
−40
−60
01/01 00:00
01/02 03:25
01/03 06:51
01/04 10:17
01/05 13:42
01/06 17:08
01/07 20:34
01/09 00:00
ds


trend
ds
2011-01
2011-03
2011-05
2011-07
2011-09
2011-11
holidays
ds
weekly
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week
yearly
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year
daily
01/01 00:00
01/02 03:25
01/03 06:51
01/04 10:17
01/05 13:42
01/06 17:08
01/07 20:34
01/09 00:00
ds

[90]:
```
perform.loc['Prophet', ('RMSE','Parameters')]=round(error_prophet), "Holidays,␣
↪Seasonality: W, D, Y"
perform
```

[90]:
```
                         RMSE                                    Parameters
Models Popcorn Holder
Moving Average            746                               Window size: 10
Exponential Smoothing     781                           Seasonal_periods =52
Arima                     909                    ARIMA(3,1,0)(2,1,0)[12]
XGBoost                   714  n_estimators=1000, max_depth=5,min_child_weight=1
Prophet                   790                    Holidays, Seasonality: W, D, Y
```

2.6 Model 6: LSTM

[91]:
```
# Now, let's fit LSTM. Please note, no parameter tuning was performed and it is␣
↪fit in its simplified form.
# A model is required to learn from the series of past observations to predict␣
↪the next value in the sequence.
```

[92]:
```
#!pip install tensorflow
#!pip install keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.preprocessing.sequence import TimeseriesGenerator
from sklearn.preprocessing import MinMaxScaler

scaler=MinMaxScaler()
```

[93]:
```
# Prepare data
train_data, test_data = pd.DataFrame(train_data), pd.DataFrame(test_data)
```

[94]:
```
# Normalize the dataset
scaler.fit(train_data)
scaled_train_data=scaler.transform(train_data)
scaled_test_data=scaler.transform(test_data)
```

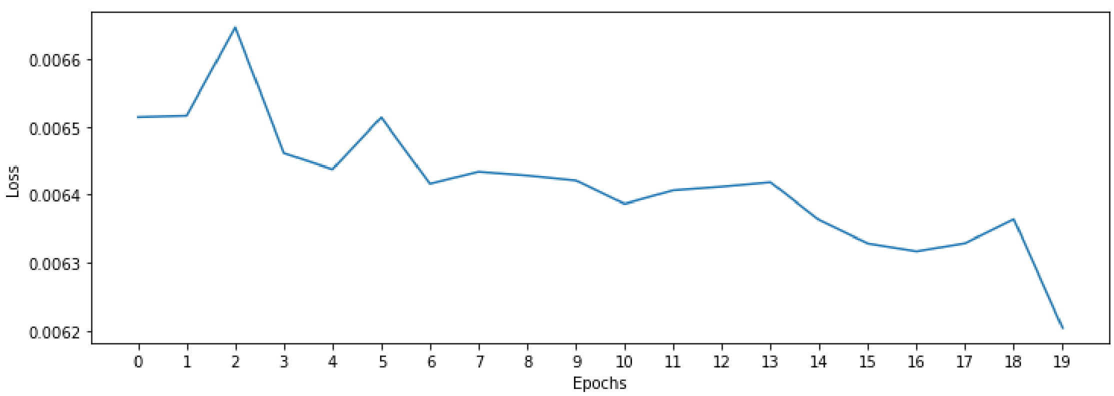
[95]:
```
# Build a model
n_input = 16
n_features = 1
generator = TimeseriesGenerator(scaled_train_data, scaled_train_data, length =␣
↪n_input, batch_size=1)
lstm_model = Sequential()
lstm_model.add(LSTM(100, activation='relu', input_shape=(n_input, n_features)))
lstm_model.add(Dense(1))
```

```
lstm_model.compile(optimizer='adam', loss='mse')
lstm_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100)               40800
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
_________________________________________________________________
```

[96]:
```
# Fit a model
lstm_model.fit(generator, epochs=20, verbose=0)

losses_lstm=lstm_model.history.history['loss']
plt.figure(figsize=(12,4))
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.xticks(np.arange(0,21,1))
plt.plot(range(len(losses_lstm)), losses_lstm)
```

[96]: [<matplotlib.lines.Line2D at 0x7fa456c9e850>]



[97]:
```
# Predict and inverse scaling
lstm_predictions_scaled=list()
batch = scaled_train_data[-n_input:]
current_batch = batch.reshape((1, n_input, n_features))
```

```python
for i in range(len(test_data)):
    lstm_pred = lstm_model.predict(current_batch)[0]
    lstm_predictions_scaled.append(lstm_pred)
    current_batch = np.append(current_batch[:, 1:, :], [[lstm_pred]], axis=1)

lstm_predictions=scaler.inverse_transform(lstm_predictions_scaled)
```

[98]:
```python
# Calculate error
error_lstm=np.sqrt(mean_squared_error(test_data.values, lstm_predictions))
error_lstm
```

[98]: 904.9477093089544

[99]:
```python
plot_actual_predicted(test_data[0], pd.DataFrame(lstm_predictions)[0], "LSTM␣
↪Predicted")
```

[100]:
```python
perform.loc['LSTM', ('RMSE','Parameters')]=round(error_lstm), "LSTM: 100, act =␣
↪'relu', input_shape: 16x1"
perform
```

[100]:

Models Popcorn Holder	RMSE	Parameters
Moving Average	746	Window size: 10
Exponential Smoothing	781	Seasonal_periods =52
Arima	909	ARIMA(3,1,0)(2,1,0)[12]
XGBoost	714	n_estimators=1000, max_depth=5,min_child_weight=1
Prophet	790	Holidays, Seasonality: W, D, Y
LSTM	905	LSTM: 100, act = 'relu', input_shape: 16x1

[101]:
```python
#By looking at the results so far, it is evident that seasonality doesn't have␣
↪strong predicive capacity. The models we applied so far do not fit well.
#My assumption is that lagged, differenciated values and rolling mean will␣
↪positevly affect model's performance.
#Let's test it out with Random Forrest by adding these features in the model's␣
↪input.
```

2.7 Model 7: Random Forest

[102]:
```python
# Prepare the data by differencing, adding 4 lags and rolling mean
df_forecasting=pd.DataFrame(poph2)
df_forecasting = df_forecasting.diff()
df_forecasting.columns=['Values']
for i in range(4,0,-1):
    df_forecasting['t-'+str(i)] = df_forecasting['Values'].shift(i)
df_forecasting=df_forecasting.dropna()
df_forecasting['Values_Rolling'] = df_forecasting['Values'].rolling(window =␣
↪16).mean()
```

```
df_forecasting= df_forecasting.dropna()
df_forecasting
```

[102]:

```
            Values     t-4     t-3     t-2     t-1  Values_Rolling
2010-12-21   -92.0   253.0  -272.0     8.0   159.0         -0.3750
2010-12-22   -75.0  -272.0     8.0   159.0   -92.0        -16.5000
2010-12-23     8.0     8.0   159.0   -92.0   -75.0         -7.5625
2010-12-24    -8.0   159.0   -92.0   -75.0     8.0        -13.2500
2010-12-25     0.0   -92.0   -75.0     8.0    -8.0        -16.2500
…              …       …       …       …       …               …
2011-12-05   296.0  -203.0  1103.0 -1369.0   113.0         25.5625
2011-12-06   -72.0  1103.0 -1369.0   113.0   296.0          7.4375
2011-12-07    31.0 -1369.0   113.0   296.0   -72.0          1.1875
2011-12-08  2738.0   113.0   296.0   -72.0    31.0        121.5000
2011-12-09 -2876.0   296.0   -72.0    31.0  2738.0        -16.6250

[354 rows x 6 columns]
```

[103]:

```
from sklearn.ensemble import RandomForestRegressor
from random import seed
x=df_forecasting.iloc[:,1:]
y=df_forecasting.iloc[:,0]
x_train, x_valid = x.loc[x.index < '2011-11-27'], x.loc[x.index >= '2011-11-27']
y_train, y_valid = y.loc[y.index < '2011-11-27'], y.loc[y.index >= '2011-11-27']
mdl = RandomForestRegressor(n_estimators=100)
np.random.seed(55)
mdl.fit(x_train, y_train)
pred=mdl.predict(x_valid)
pred=pd.Series(pred, index=y_valid.index)
```

[104]:

```
error_rf_dif=np.sqrt(mean_squared_error(y_valid, pred))
print('RMSE_dif: %.3f' % error_rf_dif)
```

```
RMSE_dif: 359.950
```

[105]:

```
plot_actual_predicted(y_valid, pred, "Random Forest Fit")
```

[106]:

```
# Inverse differencing and plot predicted values
converted=pd.DataFrame()
last_obs=train_data.iloc[-1][0]
converted['Conv']=np.r_[last_obs, pred[0:]].cumsum()[1:]
```

[107]:

```
error_rf=np.sqrt(mean_squared_error(test_data, converted))
print('RMSE: %.3f' % error_rf)
```

```
RMSE: 261.448
```

[108]:
```
plot_actual_predicted(test_data[0], converted['Conv'], "Random Forest␣
↪Predictions")
```

[109]:
```
perform.loc['RF', ('RMSE','Parameters')]=round(error_rf), "Differenced: 1, Lags:
↪ 4 , RM Window: 16"
perform
```

[109]:

	RMSE	Parameters
Models Popcorn Holder		
Moving Average	746	Window size: 10
Exponential Smoothing	781	Seasonal_periods =52
Arima	909	ARIMA(3,1,0)(2,1,0)[12]
XGBoost	714	n_estimators=1000, max_depth=5,min_child_weight=1
Prophet	790	Holidays, Seasonality: W, D, Y
LSTM	905	LSTM: 100, act = 'relu', input_shape: 16x1
RF	261	Differenced: 1, Lags: 4 , RM Window: 16

[110]:
```
#Random forrest seems to have the best fit by far. We will apply that model to␣
↪the remaining two SKU's as our first choice.
```

[111]:
```
# Therefore, the predicted order quantity for the 7 days from 11/27/2011 - 12/3/
↪2011 Sun - Sat
PO_poph2=converted[0:7].sum()[0]
print('Predicted Sales Quantity Total 11/27-12/3: %.3f' % PO_poph2)
```

```
Predicted Sales Quantity Total 11/27-12/3: 3001.460
```

[112]:
```
print('Actual Sales Quantity Total 11/27-12/3: %.3f' % test_data[0:7].sum()[0])
```

```
Actual Sales Quantity Total 11/27-12/3: 3195.000
```

[113]:
```
# Define a function to fit RF

def RF_fit(data,lags, window):
    df_forecasting=pd.DataFrame(data)
    df_forecasting = df_forecasting.diff()
    df_forecasting.columns=['Values']
    for i in range(lags,0,-1):
        df_forecasting['t-'+str(i)] = df_forecasting['Values'].shift(i)
    df_forecasting=df_forecasting.dropna()
    df_forecasting['Values_Rolling'] = df_forecasting['Values'].rolling(window).
↪mean()
    df_forecasting= df_forecasting.dropna()
    x=df_forecasting.iloc[:,1:]
    y=df_forecasting.iloc[:,0]
    x_train, x_valid = x.loc[x.index < '2011-11-27'], x.loc[x.index >=␣
↪'2011-11-27']
```

```
    y_train, y_valid = y.loc[y.index < '2011-11-27'], y.loc[y.index >=␣
↪'2011-11-27']
    mdl = rf=RandomForestRegressor(n_estimators=100)
    np.random.seed(55)
    mdl.fit(x_train, y_train)
    pred=mdl.predict(x_valid)
    pred=pd.Series(pred, index=y_valid.index)
    error_rf_dif=np.sqrt(mean_squared_error(y_valid, pred))
    print('RMSE_dif: %.3f' % error_rf_dif)
    test_data=pd.DataFrame(data[data.index>='2011-11-27'].values)
    train_data=pd.DataFrame(data[data.index<'2011-11-27'].values)
    last_obs=train_data.iloc[-1][0]
    #Let's inverse differencing and plot predicted values
    converted=pd.DataFrame()
    converted['Conv']=np.r_[last_obs, pred[0:]].cumsum()[1:]
    error_rf=np.sqrt(mean_squared_error(test_data, converted))
    print('RMSE: %.3f' % error_rf)
    plot_actual_predicted(test_data[0], converted['Conv'], "Random Forest␣
↪Predictions")
    #Therefore, the predicted order quantity for the 7 days from 11/27/2011 -␣
↪12/3/2011 Sun - Sat
    purchase_order=converted[0:7].sum()[0]
    print('Predicted Sales Quantity Total 11/27-12/3: %.3f' % purchase_order)
    print('Actual Sales Quantity Total 11/27-12/3: %.3f' % test_data[0:7].
↪sum()[0])
    return purchase_order
```

2.8 Predicting Quantities: Popcorn Holder

[114]:
```
# Let's test it and ensure it works
PO_poph2=RF_fit(poph2, 4, 16)
```

```
RMSE_dif: 359.950
RMSE: 261.448

Predicted Sales Quantity Total 11/27-12/3: 3001.460
Actual Sales Quantity Total 11/27-12/3: 3195.000
```

[115]:
```
# While we can regress the quantities over counties and dates to come up with␣
↪country coefficients, I used a naive approach of picking the last month of␣
↪sales data and applying factors for each country
```

[116]:
```
# Define a function for it
def predict_orders (data, predicted_order):
    data_train=data[(data.index<datetime.date(2011,11,27)) & (data.
↪index>=(datetime.date(2011,10,26)))]
    data_test=data[(data.index>=datetime.date(2011,11,27)) & (data.
↪index<=datetime.date(2011,12,3))]
```

```
    pred_countries=pd.DataFrame(data_train.groupby('Country')['Quantity'].sum().
↪sort_values(ascending=False))
    actual_countries_data=pd.DataFrame(data_test.groupby('Country')['Quantity'].
↪sum().sort_values(ascending=False))
    countries_data=pd.merge(pred_countries, actual_countries_data,␣
↪on=['Country'])
    countries_data['Ratio']=round(countries_data.Quantity_x/countries_data.
↪Quantity_x.sum(),2)
    countries_data['Predictions']=round(countries_data.Ratio*predicted_order)
    countries_data.rename(columns = {'Quantity_y':'Actual'}, inplace = True)
    countries_data.rename(columns = {'Quantity_x':'Historical'}, inplace = True)
    return (countries_data) #.head(15))
```

[117]:
```
predict_orders (poph, PO_poph2).head()
```

[117]:

Country	Historical	Actual	Ratio	Predictions
United Kingdom	12783	3083	0.98	2941.0
Italy	100	0	0.01	30.0
EIRE	92	12	0.01	30.0
France	54	0	0.00	0.0
Belgium	36	0	0.00	0.0

2.9 Predicting quantities: Vintage Doily Jumbo Bag Red

[118]:
```
# Fit RF model
PO_vinb2=RF_fit(vinb2, 4, 1)
```

```
RMSE_dif: 521.429
RMSE: 362.303

Predicted Sales Quantity Total 11/27-12/3: 602.360
Actual Sales Quantity Total 11/27-12/3: 1851.000
```

[119]:
```
predict_orders(vinb, PO_vinb2).head()
```

[119]:

Country	Historical	Actual	Ratio	Predictions
United Kingdom	2125	1830	0.94	566.0
Portugal	40	0	0.02	12.0
France	35	10	0.02	12.0
Germany	20	10	0.01	6.0
Finland	20	0	0.01	6.0

2.10 Predicting Quantities: Rabbit Night Light

[120]:
```
# Fit the Random Forest to Rabbit Night Light data
```

[121]:
```
PO_rabl2=RF_fit(rabl2, 4, 1)
```

```
RMSE_dif: 485.418
RMSE: 337.100

Predicted Sales Quantity Total 11/27-12/3: 3906.450
Actual Sales Quantity Total 11/27-12/3: 4588.000
```

[122]:
```
predict_orders(rabl, PO_rabl2).head()
```

[122]:
```
                Historical  Actual  Ratio  Predictions
Country
United Kingdom        8784    1985   0.51       1992.0
Netherlands           2616       0   0.15        586.0
France                2326     383   0.14        547.0
Australia             1632       0   0.10        391.0
Japan                 1080    2040   0.06        234.0
```

[123]:
```
# Japan's order is obviously underpredicted
```

[124]:
```
rabl[rabl.Country=="Japan"] #It was unusual order for that country
```

[124]:
```
            InvoiceNo StockCode        Description  Quantity  UnitPrice  \
InvoiceDate
2011-06-22     557670     23084  RABBIT NIGHT LIGHT       288       1.79
2011-10-26     572869     23084  RABBIT NIGHT LIGHT       960       1.79
2011-11-17     576923     23084  RABBIT NIGHT LIGHT       120       1.79
2011-11-29     579498     23084  RABBIT NIGHT LIGHT      2040       1.79
2011-12-06    C580832     23084  RABBIT NIGHT LIGHT        -7       1.79

             CustomerID Country
InvoiceDate
2011-06-22      12798.0   Japan
2011-10-26      12798.0   Japan
2011-11-17      12753.0   Japan
2011-11-29      12798.0   Japan
2011-12-06      12753.0   Japan
```

[125]:
```
# Add predicted orders to top3 dataframe
orders={'Rabbit_Night_Light': round(PO_rabl2), 'Popcorn_Holder':␣
 ↪round(PO_poph2), 'Vintage_Doily_Jumbo_Bag': round(PO_vinb2)}
```

[126]:
```
top3['Description']=[x for x in pd.DataFrame(orders.items())[0]]
```

[127]:
```
top3['Predicted_Order_ML']=[x for x in pd.DataFrame(orders.items())[1]]
```

[128]:
```
top3
```

[128]:
```
           Quantity             Description  Predicted_Order_ML
StockCode
23084          4588       Rabbit_Night_Light                3906
22197          3195           Popcorn_Holder                3001
23582          1851  Vintage_Doily_Jumbo_Bag                 602
```

[129]:
```
# A workable solution to address understock is to add a safety stock:
# Safety stock ss = std(delivery_lead_time) * importance_factor
# Since we don't have that data, we can substitute:
# ss = std(daily_quantity) * (days_of_prediction) * (model_confidence_factor)
# Therefore, our prediction could be the following:
# Predicted_Order = Predicted_Order_ML + ss
```

[130]:
```
model_confidence_factor=[0.3, 0.1, 0.9]
stdevs=[np.std(rabl2.values)*6,np.std(poph2)*6,np.std(vinb2)*6]
```

[131]:
```
ss = [round(a * b) for a, b in zip(model_confidence_factor, stdevs)]
print(ss)
```

```
[645, 224, 1201]
```

[132]:
```
top3.insert(2, "Safety_Stock", ss)
```

[133]:
```
top3['Final_Order']=top3.Safety_Stock+top3.Predicted_Order_ML
```

[134]:
```
top3
```

[134]:
```
           Quantity             Description  Safety_Stock  \
StockCode
23084          4588       Rabbit_Night_Light           645
22197          3195           Popcorn_Holder           224
23582          1851  Vintage_Doily_Jumbo_Bag          1201

           Predicted_Order_ML  Final_Order
StockCode
23084                    3906         4551
22197                    3001         3225
23582                     602         1803
```

[135]:
```
# Aggregate the results
rabbit_light=predict_orders(rabl, top3.Final_Order.values[0])
rabbit_light.head()
```

[135]:
```
                Historical  Actual  Ratio  Predictions
Country
United Kingdom        8784    1985   0.51       2321.0
```

```
Netherlands           2616       0   0.15        683.0
France                2326     383   0.14        637.0
Australia             1632       0   0.10        455.0
Japan                 1080    2040   0.06        273.0
```

[136]:
```
popcorn_holder=predict_orders(poph, top3.Final_Order.values[1])
popcorn_holder.head()
```

[136]:
```
                Historical  Actual  Ratio  Predictions
Country
United Kingdom       12783    3083   0.98       3160.0
Italy                  100       0   0.01         32.0
EIRE                    92      12   0.01         32.0
France                  54       0   0.00          0.0
Belgium                 36       0   0.00          0.0
```

[137]:
```
vintage_bag=predict_orders(vinb, top3.Final_Order.values[2])
vintage_bag.head()
```

[137]:
```
                Historical  Actual  Ratio  Predictions
Country
United Kingdom        2125    1830   0.94       1695.0
Portugal                40       0   0.02         36.0
France                  35      10   0.02         36.0
Germany                 20      10   0.01         18.0
Finland                 20       0   0.01         18.0
```

[138]:
```
step1=pd.merge(rabbit_light, popcorn_holder, on = 'Country', how='outer')
```

[139]:
```
final_data=pd.merge(step1, vintage_bag, on = 'Country', how='outer')
```

[140]:
```
final_data.columns=['Hist_rabbbit_light', 'Act_rabbit_light', 'R_rabbit_light',␣
 ↪'Preds_rabbit_light', 'Hist_popcorn_holder', 'Act_popcorn_holder',␣
 ↪'R_popcorn_holder', \
                    'Preds_popcorn_holder', 'Hist_vintage_bag',␣
 ↪'Act_vintage_bag', 'R_vintage_bag', 'Preds_vintage_bag']
```

[141]:
```
final_data
```

[141]:
```
                      Hist_rabbbit_light  Act_rabbit_light  R_rabbit_light  \
Country
United Kingdom                      8784              1985            0.51
Netherlands                         2616                 0            0.15
France                              2326               383            0.14
Australia                           1632                 0            0.10
Japan                               1080              2040            0.06
Germany                              192                72            0.01
Belgium                              108                 0            0.01
```

Finland	96	48	0.01
Sweden	84	0	0.00
EIRE	48	0	0.00
Iceland	48	0	0.00
Italy	48	0	0.00
Denmark	24	12	0.00
Portugal	18	48	0.00
Unspecified	12	0	0.00
Norway	12	0	0.00
Switzerland	12	0	0.00
Spain	6	0	0.00
Lithuania	0	0	0.00
USA	0	0	0.00
Austria	0	0	0.00
Bahrain	0	0	0.00
United Arab Emirates	0	0	0.00
Brazil	0	0	0.00
Canada	0	0	0.00
Channel Islands	0	0	0.00
Cyprus	0	0	0.00
Czech Republic	0	0	0.00
European Community	0	0	0.00
Lebanon	0	0	0.00
Singapore	0	0	0.00
Saudi Arabia	0	0	0.00
Greece	0	0	0.00
Hong Kong	0	0	0.00
RSA	0	0	0.00
Poland	0	0	0.00
Malta	0	0	0.00
Israel	0	0	0.00

	Preds_rabbit_light	Hist_popcorn_holder	\
Country			
United Kingdom	2321.0	12783	
Netherlands	683.0	0	
France	637.0	54	
Australia	455.0	0	
Japan	273.0	0	
Germany	46.0	0	
Belgium	46.0	36	
Finland	46.0	0	
Sweden	0.0	0	
EIRE	0.0	92	
Iceland	0.0	0	
Italy	0.0	100	
Denmark	0.0	0	

```
Portugal                              0.0                0
Unspecified                           0.0                0
Norway                                0.0                0
Switzerland                           0.0                0
Spain                                 0.0               36
Lithuania                             0.0                0
USA                                   0.0                0
Austria                               0.0                0
Bahrain                               0.0                0
United Arab Emirates                  0.0                0
Brazil                                0.0                0
Canada                                0.0                0
Channel Islands                       0.0                0
Cyprus                                0.0                0
Czech Republic                        0.0                0
European Community                    0.0                0
Lebanon                               0.0                0
Singapore                             0.0                0
Saudi Arabia                          0.0                0
Greece                                0.0                0
Hong Kong                             0.0                0
RSA                                   0.0                0
Poland                                0.0                0
Malta                                 0.0                0
Israel                                0.0                0

                      Act_popcorn_holder  R_popcorn_holder  \
Country
United Kingdom                      3083              0.98
Netherlands                            0              0.00
France                                 0              0.00
Australia                              0              0.00
Japan                                  0              0.00
Germany                                0              0.00
Belgium                                0              0.00
Finland                                0              0.00
Sweden                                 0              0.00
EIRE                                  12              0.01
Iceland                                0              0.00
Italy                                  0              0.01
Denmark                                0              0.00
Portugal                               0              0.00
Unspecified                            0              0.00
Norway                               100              0.00
Switzerland                            0              0.00
Spain                                  0              0.00
Lithuania                              0              0.00
```

USA	0	0.00
Austria	0	0.00
Bahrain	0	0.00
United Arab Emirates	0	0.00
Brazil	0	0.00
Canada	0	0.00
Channel Islands	0	0.00
Cyprus	0	0.00
Czech Republic	0	0.00
European Community	0	0.00
Lebanon	0	0.00
Singapore	0	0.00
Saudi Arabia	0	0.00
Greece	0	0.00
Hong Kong	0	0.00
RSA	0	0.00
Poland	0	0.00
Malta	0	0.00
Israel	0	0.00

	Preds_popcorn_holder	Hist_vintage_bag	Act_vintage_bag	\
Country				
United Kingdom	3160.0	2125	1830	
Netherlands	0.0	1	0	
France	0.0	35	10	
Australia	0.0	0	0	
Japan	0.0	0	0	
Germany	0.0	20	10	
Belgium	0.0	0	0	
Finland	0.0	20	0	
Sweden	0.0	0	0	
EIRE	32.0	0	0	
Iceland	0.0	0	0	
Italy	32.0	10	0	
Denmark	0.0	0	0	
Portugal	0.0	40	0	
Unspecified	0.0	0	0	
Norway	0.0	0	0	
Switzerland	0.0	0	0	
Spain	0.0	0	1	
Lithuania	0.0	0	0	
USA	0.0	0	0	
Austria	0.0	0	0	
Bahrain	0.0	0	0	
United Arab Emirates	0.0	0	0	
Brazil	0.0	0	0	
Canada	0.0	0	0	

Channel Islands	0.0	10	0
Cyprus	0.0	0	0
Czech Republic	0.0	0	0
European Community	0.0	0	0
Lebanon	0.0	0	0
Singapore	0.0	0	0
Saudi Arabia	0.0	0	0
Greece	0.0	0	0
Hong Kong	0.0	0	0
RSA	0.0	0	0
Poland	0.0	0	0
Malta	0.0	0	0
Israel	0.0	0	0

	R_vintage_bag	Preds_vintage_bag
Country		
United Kingdom	0.94	1695.0
Netherlands	0.00	0.0
France	0.02	36.0
Australia	0.00	0.0
Japan	0.00	0.0
Germany	0.01	18.0
Belgium	0.00	0.0
Finland	0.01	18.0
Sweden	0.00	0.0
EIRE	0.00	0.0
Iceland	0.00	0.0
Italy	0.00	0.0
Denmark	0.00	0.0
Portugal	0.02	36.0
Unspecified	0.00	0.0
Norway	0.00	0.0
Switzerland	0.00	0.0
Spain	0.00	0.0
Lithuania	0.00	0.0
USA	0.00	0.0
Austria	0.00	0.0
Bahrain	0.00	0.0
United Arab Emirates	0.00	0.0
Brazil	0.00	0.0
Canada	0.00	0.0
Channel Islands	0.00	0.0
Cyprus	0.00	0.0
Czech Republic	0.00	0.0
European Community	0.00	0.0
Lebanon	0.00	0.0
Singapore	0.00	0.0

```
Saudi Arabia                    0.00            0.0
Greece                          0.00            0.0
Hong Kong                       0.00            0.0
RSA                             0.00            0.0
Poland                          0.00            0.0
Malta                           0.00            0.0
Israel                          0.00            0.0
```

[142]:

```
final_data.to_csv(r'Tatsiana_Sokalava}_result.csv', index = True, header=True)
```

3 Thank you!