Machine Learning Solution

Business Problem

- To forecast sales data for each unique "Region Department Facility" combination
- To find the best featurization/transformation technique based on the model performance
- To find the best model for each unique "Region Department Facility" combination

Note

- "unique_df" folder - contains good data which has passed our thresholds
- "log_df" folder - contains Log Transformed Data
- "final_df" folder - contains Differencing/Lag Transformed Data
- "plots" folder - contains timeseries plots of all the good data stored in our unique_df folder, they are stored in .png format so you can look at it normally
- "predictions" folder - contains the predictions of our LSTM model, read the "LSTM.ipynb" for better understanding
- "results_pickle" folder - contains results of all our models which are stored in pickle format, read "Results.ipynb" for their use

In [1]:
```
import pandas as pd
import itertools
import pylab as pl
import os
import seaborn as sns
import scipy.stats as stats
from sklearn.neighbors import KernelDensity
from statsmodels.graphics.gofplots import qqplot
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import numpy as np
from matplotlib import pyplot as plt
import matplotlib
from datetime import datetime
%matplotlib inline
import warnings
import os
import pickle
import re
import statsmodels.api as sm
from datetime import datetime
from matplotlib.ticker import MultipleLocator, FormatStrFormatter, AutoMinorLocator
from matplotlib import dates as mdates
import cv2
from IPython.display import display
from PIL import Image
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.holtwinters import ExponentialSmoothing, SimpleExpSmoothing
from prettytable import PrettyTable
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('RDF_OLD_CASTLE_AGGREGATED_INPUT_FILE.csv', encoding="ISO-8859-1")

print('number of rows and cols in our dataset: ', df.shape)
```

```
number of rows and cols in our dataset:  (15603, 6)
```

In [3]:
```
df.head()
```

Out[3]:

	Region	Division_Name	Facility_Name	Year	Month	Actual_Sales
0	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	1	1590866
1	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	2	1903852
2	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	3	373065
3	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	4	98
4	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	7	-10572

In [4]:
```
def combine(year, month):
    return(str(year)+'/'+str(month))
```

In [5]:
```
new_dates = df.apply(lambda x: combine(x['Year'], x['Month']), axis=1)
```

In [6]:
```
df['Dates'] = new_dates
df.head()
```

Out[6]:

	Region	Division_Name	Facility_Name	Year	Month	Actual_Sales	Dates
0	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	1	1590866	2015/1
1	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	2	1903852	2015/2
2	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	3	373065	2015/3
3	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	4	98	2015/4
4	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	7	-10572	2015/7

In [7]:
```
df = df.set_index('Dates')
```

In [8]:
```
df.head(20)
```

Out[8]:

	Region	Division_Name	Facility_Name	Year	Month	Actual_Sales
Dates						
2015/1	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	1	1590866
2015/2	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	2	1903852
2015/3	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	3	373065
2015/4	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	4	98
2015/7	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	7	-10572
2015/1	APG ATLANTA	Big River Industries	Livingston AL - HISTORICAL	2015	1	1240326
2015/2	APG ATLANTA	Big River Industries	Livingston AL - HISTORICAL	2015	2	1068318
2015/3	APG ATLANTA	Big River Industries	Livingston AL - HISTORICAL	2015	3	296140
2015/6	APG ATLANTA	Oldcastle Bonsal American	Auburn Hills MI - HISTORICAL	2015	6	-3193
2015/4	APG ATLANTA	Oldcastle Bonsal American	Auburndale F - HISTORICAL	2015	4	-302
2015/5	APG ATLANTA	Oldcastle Bonsal American	Auburndale F - HISTORICAL	2015	5	-162
2015/2	APG ATLANTA	Oldcastle Bonsal American	Conley GA - HISTORICAL	2015	2	-6958
2015/2	APG ATLANTA	Oldcastle Bonsal American	Cresson TX - HISTORICAL	2015	2	-8107
2015/3	APG ATLANTA	Oldcastle Bonsal American	Fredonia PA-Plant - HISTORICAL	2015	3	-299
2015/2	APG ATLANTA	Oldcastle Bonsal American	Midlothian TX - HISTORICAL	2015	2	-3160
2015/2	APG ATLANTA	Oldcastle Bonsal American	Pompano FL - HISTORICAL	2015	2	-3400
2015/6	APG ATLANTA	Oldcastle Bonsal American	Pompano FL - HISTORICAL	2015	6	277
2015/2	APG ATLANTA	Oldcastle Bonsal American	Tampa FL - HISTORICAL	2015	2	-154
2015/4	APG ATLANTA	Oldcastle Bonsal American	Tampa FL - HISTORICAL	2015	4	-406
2017/3	APG ATLANTA	Oldcastle Retail CMP999820	EZ Mix	2017	3	58061

In [9]:
```
df.sort_index()
df.sort_values(by=['Region', 'Division_Name','Facility_Name'], ascending=[True, True, True])
```

Out[9]:

	Region	Division_Name	Facility_Name	Year	Month	Actual_Sales
Dates						
2015/1	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	1	1590866
2015/2	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	2	1903852
2015/3	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	3	373065
2015/4	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	4	98
2015/7	APG ATLANTA	Big River Industries	Erwinville LA - HISTORICAL	2015	7	-10572
...	...	...	...	...	...	...
2019/7	West	Superlite	West Phoenix N. 42nd Ave, AZ	2019	7	5666431
2019/8	West	Superlite	West Phoenix N. 42nd Ave, AZ	2019	8	6430412
2019/9	West	Superlite	West Phoenix N. 42nd Ave, AZ	2019	9	5454475
2019/10	West	Superlite	West Phoenix N. 42nd Ave, AZ	2019	10	6745807
2019/11	West	Superlite	West Phoenix N. 42nd Ave, AZ	2019	11	2877254

15603 rows × 6 columns

In [10]:
```
df.index
```

Out[10]:
```
Index(['2015/1', '2015/2', '2015/3', '2015/4', '2015/7', '2015/1', '2015/2',
       '2015/3', '2015/6', '2015/4',
       ...
       '2019/2', '2019/3', '2019/4', '2019/5', '2019/6', '2019/7', '2019/8',
       '2019/9', '2019/10', '2019/11'],
      dtype='object', name='Dates', length=15603)
```

Converting Dates to proper date time format

In [11]:
```
df.index = pd.to_datetime(df.index)
```

In [12]:
```
df.index
```

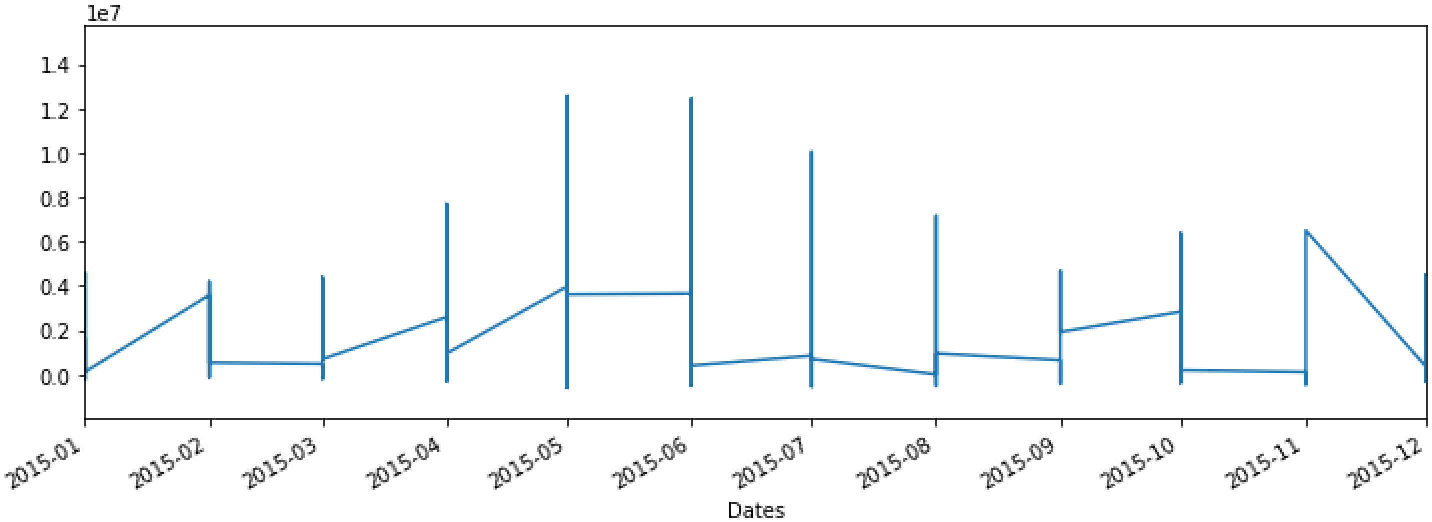
Out[12]:
```
DatetimeIndex(['2015-01-01', '2015-02-01', '2015-03-01', '2015-04-01',
               '2015-07-01', '2015-01-01', '2015-02-01', '2015-03-01',
               '2015-06-01', '2015-04-01',
               ...
               '2019-02-01', '2019-03-01', '2019-04-01', '2019-05-01',
               '2019-06-01', '2019-07-01', '2019-08-01', '2019-09-01',
               '2019-10-01', '2019-11-01'],
              dtype='datetime64[ns]', name='Dates', length=15603, freq=None)
```

In [13]:
```
len(df["Facility_Name"].unique())
```

Out[13]:
```
469
```

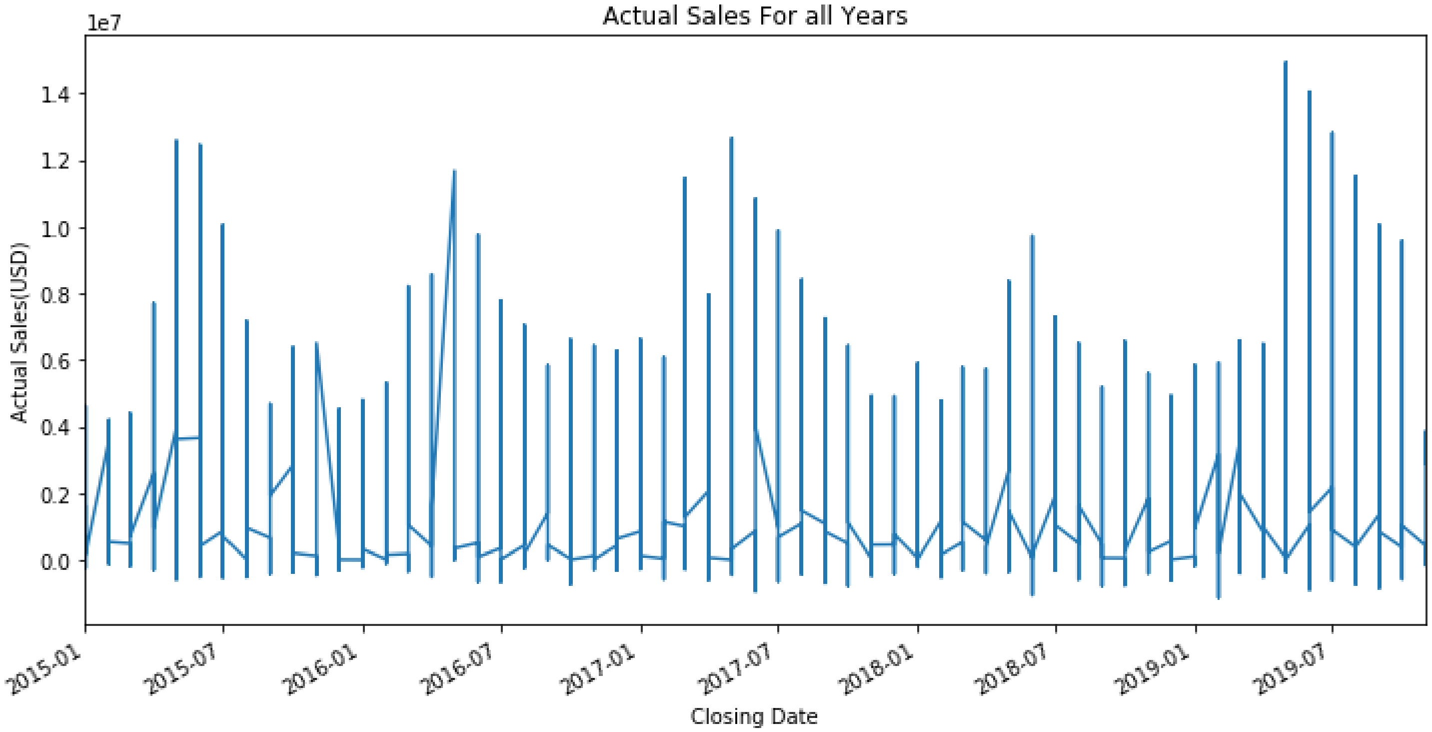
Incorrect Plots

In [15]:
```
# Dates are separated by a comma:
df['Actual_Sales'].plot(figsize=(12,4),xlim=['2015/01','2015/12']);
```

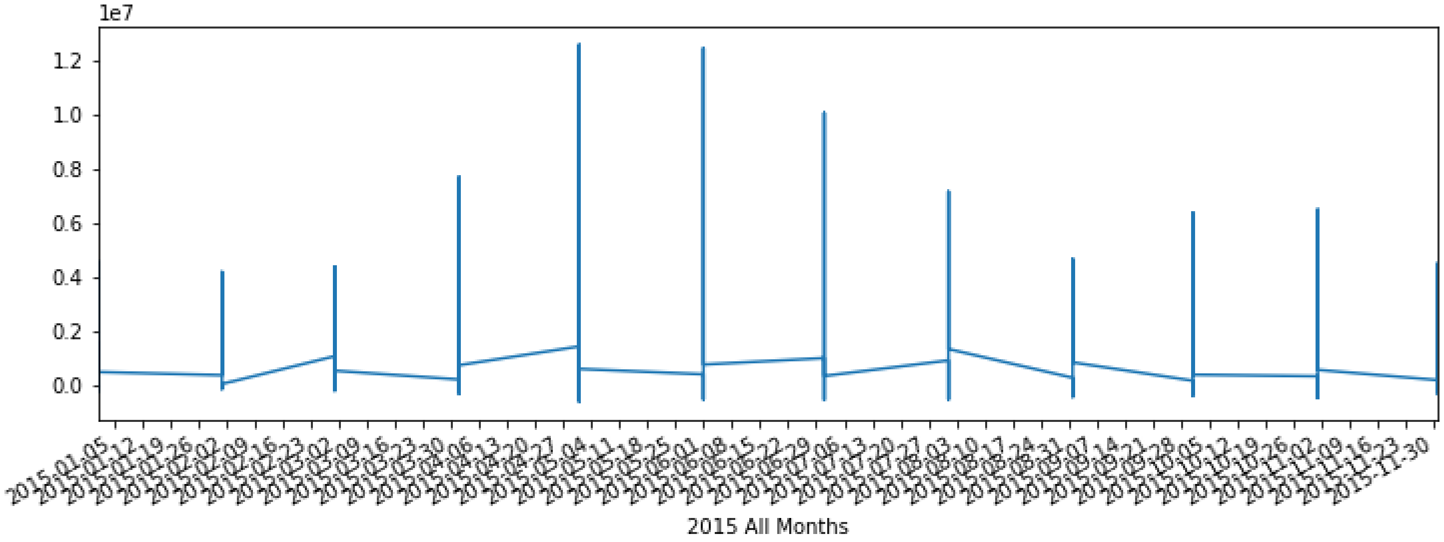


See the above plot. it doesnot gives valueable information because its takes a combination of dates from 2015 to 2017 using any Division and facility values

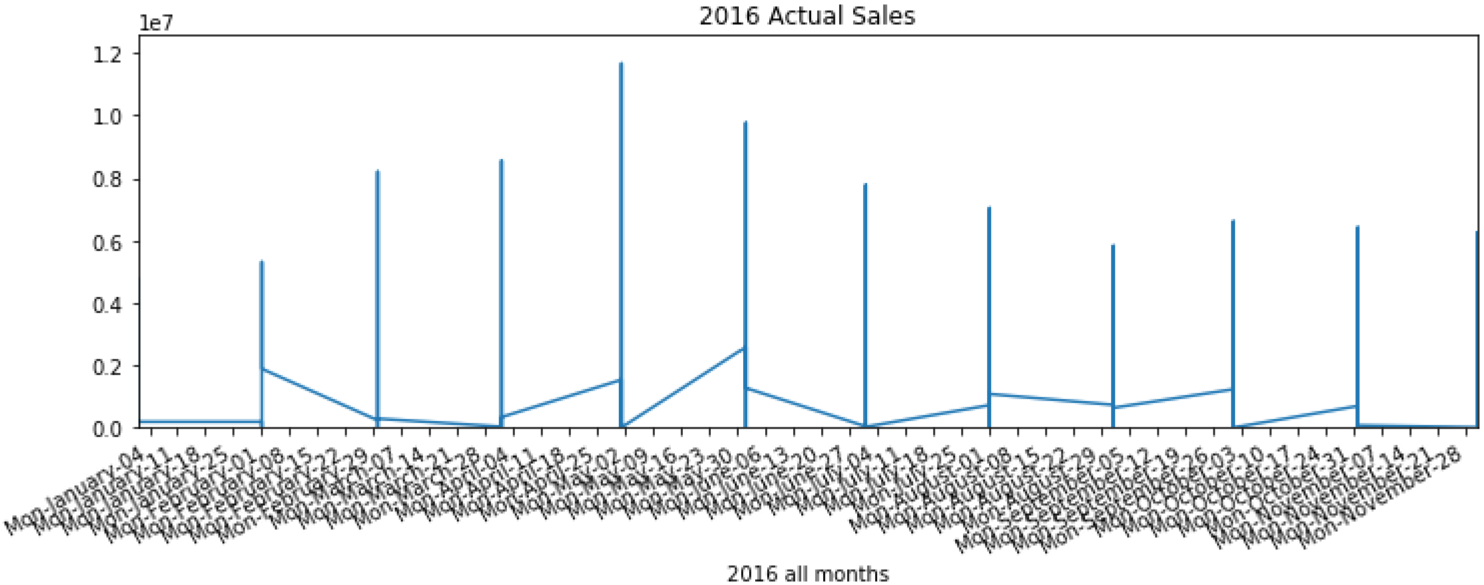
In [16]:
```
title='Actual Sales For all Years'
ylabel='Actual Sales(USD)'
xlabel='Closing Date'

ax = df['Actual_Sales'].plot(figsize=(12,6),title=title)
ax.autoscale(axis='x',tight=True)
ax.set(xlabel=xlabel, ylabel=ylabel);
```



In [17]:
```
# CREATE OUR AXIS OBJECT
from matplotlib import dates
ax = df['Actual_Sales']['2015/01':'2015/12'].plot(figsize=(12,4))

ax.set(xlabel='2015 All Months')

ax.xaxis.set_major_locator(dates.WeekdayLocator(byweekday=0))
```

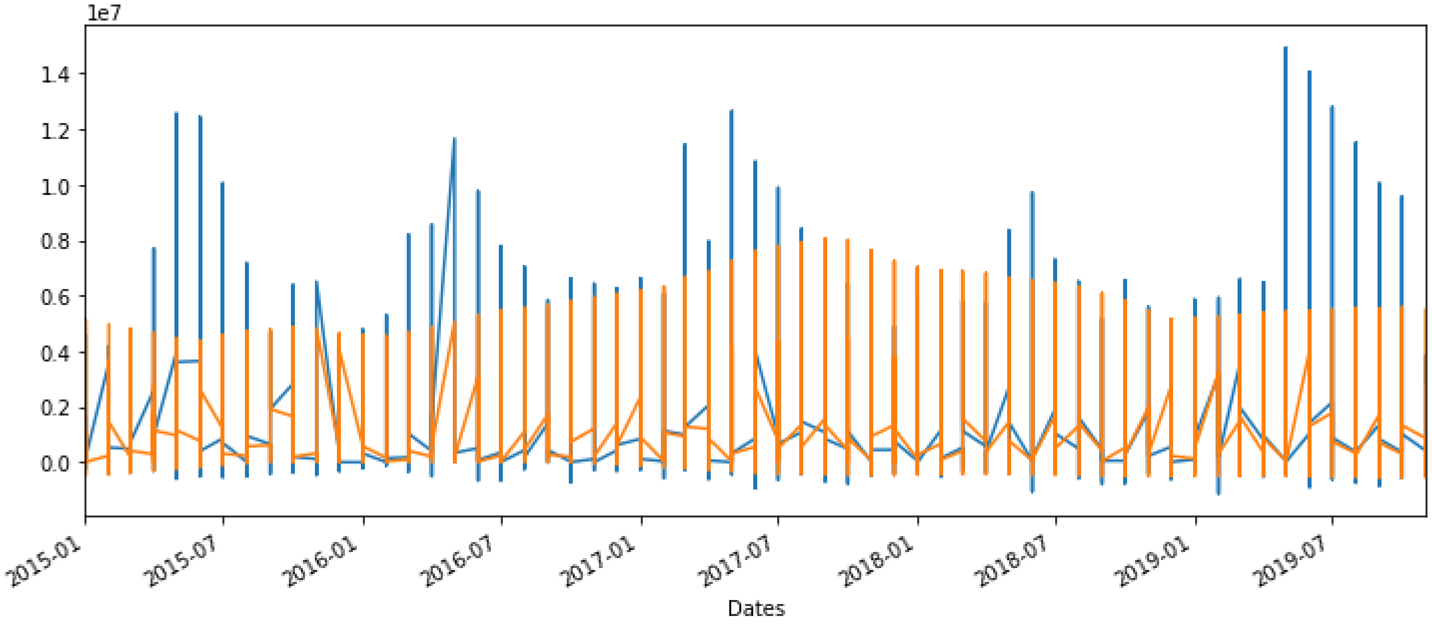


In [18]:
```
ax = df['Actual_Sales']['2016/01':'2016/12'].plot(ylim=[0,12570305],title='2016 Actual Sales',figsize=(12,4))
ax.set(xlabel='2016 all months')

ax.xaxis.set_major_locator(dates.WeekdayLocator(byweekday=0))
ax.xaxis.set_major_formatter(dates.DateFormatter("%a-%B-%d"))
```



Increase the fig size if you want to see detailed labels or reduce the data points

In [19]:
```
#Rolling Means
df['Actual_Sales'].plot(figsize=(12,5)).autoscale(axis='x',tight=True)
df["Actual_Sales"].rolling(30).mean().plot();
```



Observations

- Above are the plots which shows you the incorrect method of plotting of our data
- where we haven't taken any unique combinations from our original dataset and hence are not able to interpret anything from the plots

Applying ML Solution

In [22]:
```
def preprocessing(text):
    text = re.sub('/', '-', text)

    return text
```

In [23]:
```
# substiuting '/' with '-' in the dataset values
df['Region'] = df.Region.apply(lambda text: preprocessing(text))
df['Division_Name'] = df.Division_Name.apply(lambda text: preprocessing(text))
df['Facility_Name'] = df.Facility_Name.apply(lambda text: preprocessing(text))
```

In [27]:
```
print('Number of unique Regions: ', len(df.Region.unique()))
df.Region.unique()
```

```
Number of unique Regions:  8
```

Out[27]:
```
array(['APG ATLANTA', 'CLOSED', 'Canada', 'Central', 'East',
       'Lawn and Garden', 'National', 'West'], dtype=object)
```

In [25]:
```
print('Number of unique Division Names: ', len(df.Division_Name.unique()))
print('\nDivision Names: ', df.Division_Name.unique())
```

```
Number of unique Division Names:  31

Division Names:  ['Big River Industries' 'Oldcastle Bonsal American'
 'Oldcastle Retail CMP999820' 'H B Fuller' 'Merchants Metals'
 'Oldcastle Glen Gery Brick' 'Pavement Maintenance Division'
 'Abbotsford Concrete' 'Expocrete' 'Permacon' 'Ash Grove MPC' 'Jewell'
 'Northfield' 'Adams Products' 'Anchor' 'Georgia Masonry Supply'
 'OldcastleCoastal' 'L&G Central' 'L&G Northeast' 'L&G Southeast' 'AMTC'
 'Anchor Wall Systems' 'MoistureShield' 'National Strategic Accounts'
 'Oldcastle Sakerete Billing' 'Techniseal' 'Westile' 'Amcor' 'CPM'
 'Sierra' 'Superlite']
```

In [26]:
```
print('Number of unique Facilities: ', len(df.Facility_Name.unique()))
f_names = df.Facility_Name.unique()
print('5 Facility names: \n', f_names[:5])
```

```
Number of unique Facilities:  469
5 Facility names:
 ['Erwinville LA - HISTORICAL' 'Livingston AL - HISTORICAL'
 'Auburn Hills MI  - HISTORICAL' 'Auburndale F - HISTORICAL'
 'Conley GA - HISTORICAL']
```

Now we'll check all the unique combinations and store it in "unique_df" folder

In [28]:
```
# creating unique combinatios of all the regions, division and facilities
# checking if each combination is there in the dataset or not
# storing the pickle files

start = datetime.now()
r_names = df.Region.unique()  # region names
d_names = df.Division_Name.unique()  # department names
f_names = df.Facility_Name.unique() # facility names

# combinations = []
not_present = 0
for r in r_names:
    for d in d_names:
        for f in f_names:
            name = r + '_' + d + '_' + f
            new_df = df[(df.Region == r) & (df.Division_Name == d) & (df.Facility_Name == f)]
            if len(new_df) == 0:
#                 print('{0} not in df'.format(name))
                not_present += 1
            else:
                dir = os.path.join('unique_df'+'/'+name)
                if not os.path.exists(dir):
                    os.mkdir(dir)
                    new_df.to_pickle(dir+'/'+name+'.pkl')
print('Number of combinations not there in df: ', not_present)
print('Time taken: ', datetime.now() - start)
```

```
Number of combinations not there in df:  115836
Time taken:  0:16:55.862643
```

In [29]:
```
# Let's see the number of unique combinations we have

ls_dir = os.listdir("./unique_df/")
print('Number of unique combinations we have in our unique_df folder: ', len(ls_dir))
```

```
Number of unique combinations we have in our unique_df folder:  476
```

Thresholding by checking if our unique combination's data is there in 2019 and has min 2 years of data

In [32]:
```
def check_2019(dir):
    '''
        This function checks if data is there in year 2019 from Jan to Nov, if not we remove it
    '''
    print(check_2019.__doc__)
    ls_dir = os.listdir(dir)
    print('\nTotal files in the folder: ', len(ls_dir))
    not_2019 = []
    count = 0
    for file_n in ls_dir:
        location = dir + file_n + '/' + file_n + '.pkl'
        data = pd.read_pickle(location)
        # checking if there is data in 2019 or not in the dataframe
        if len(data['2019-01-01':'2019-11-01']) != 11:
            not_2019.append(file_n)
            dir_rem = dir + file_n
            os.remove(location)  # removing the pkl file from it's folder
            os.rmdir(dir_rem) # removing the folder cuz it's empty now
            count += 1
    print('Number of unique combinations where data is not there in 2019: ', count)

    return not_2019
```

In [33]:
```
start = datetime.now()
closed_data_names = check_2019("./unique_df/")
print('\nTime taken: ', datetime.now() - start)
```

```
        This function checks if data is there in year 2019 from Jan to Nov, if not we remove it


Total files in the folder:  476
Number of unique combinations where data is not there in 2019:  265

Time taken:  0:00:02.487827
```

In [36]:
```
# now let's check how many unique combinations we have after cleaning useless data

ls_dir = os.listdir("./unique_df/")
print('Number of good files left after our 2019 threshold is: ', len(ls_dir))
```

```
Number of good files left after our 2019 threshold is:  211
```

In [14]:
```
def get_files_from_dir(dir):
    '''
        This function checks if files are atleast having 2 years of data
    '''
    print(get_files_from_dir.__doc__)
    ls = os.listdir(dir)
    print('\nTotal files in folder: ', len(ls))
    pairs = []
    good_data = []
    good_data_names = []
    locations = []
    bad_data_names = []
    count = 0
    for file_n in ls:
        count += 1
        if dir == './unique_df/':
            location = dir + file_n + '/' + file_n + '.pkl'
        else:
            location = dir + file_n
        a = pd.read_pickle(location)
        size = a.shape[0]
        # keeping a threshold size of 24
        if (size > 24):
            good_data.append(a)
            good_data_names.append(file_n)
            locations.append(location)
            pairs.append((file_n, location))
        else:
            bad_data_names.append(file_n)

    return good_data, good_data_names, locations, pairs, bad_data_names
```

In [15]:
```
start = datetime.now()
good_data_unique, good_names_unique, good_paths_unique, good_pairs_unique, bad_names_unique = \
get_files_from_dir("./unique_df/")
print('Time taken: ', datetime.now() - start)
```

```
        This function checks if files are atleast having 2 years of data


Total files in folder:  211
Time taken:  0:00:02.174525
```

In [5]:
```
print('Number of files which do not have min 2 years of data: ', len(bad_names_unique))
```

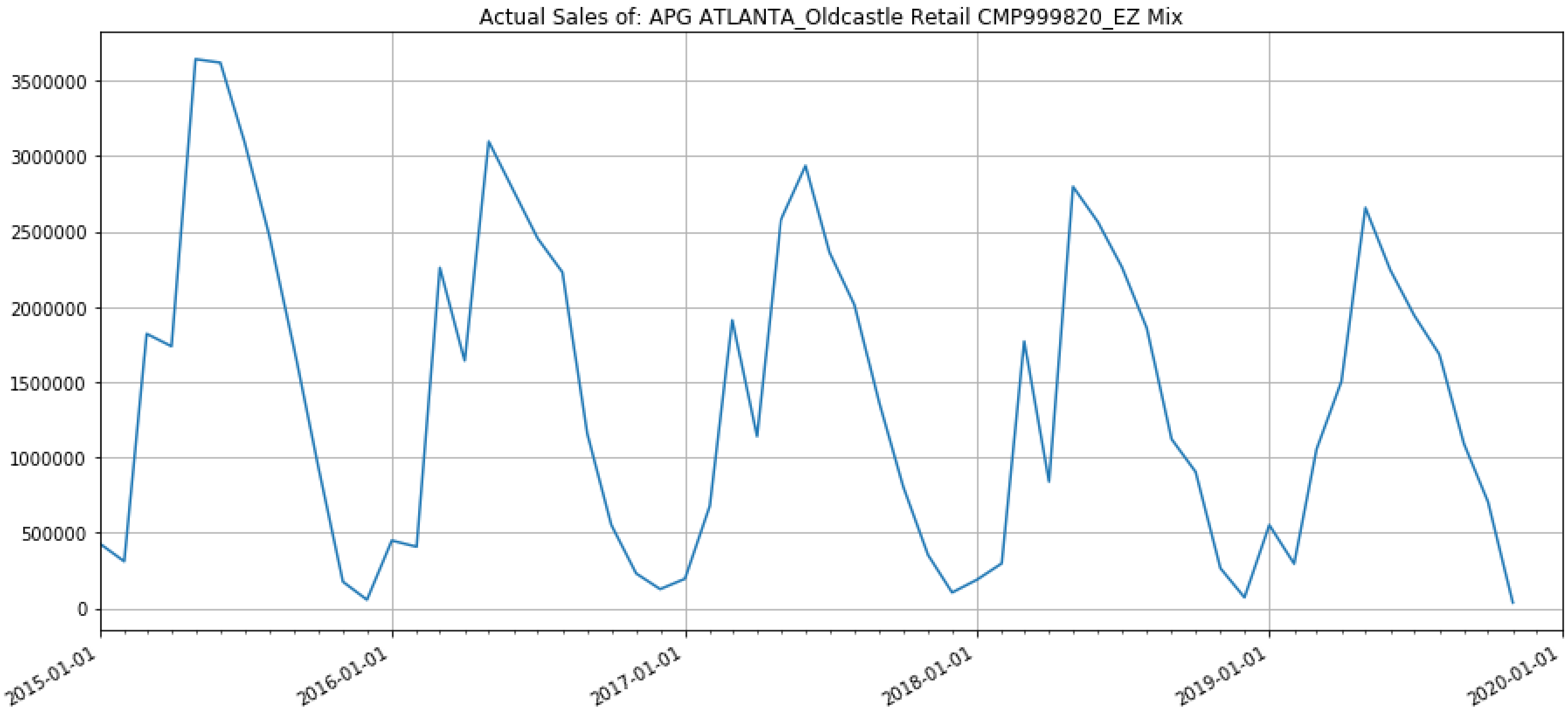
```
Number of files which do not have min 2 years of data:  2
```

In [6]:
```
print('Number of files which are useful: ', len(good_names_unique))
```

```
Number of files which are useful:  209
```

Let's see one such unique combination after thresholding data

In [35]:
```
# single test plot
months = mdates.MonthLocator()  # every month
years = mdates.YearLocator()   # every year
years_fmt = mdates.DateFormatter('%Y-%m-%d')

df_plot = pd.read_pickle(good_paths_unique[1]) # using good_paths which we got from get_files_from_dir()
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15,7))  # create figure & 1 axis
# ax.plot('Dates', 'Actual_Sales', data=df)
ax.plot(df_plot['Actual_Sales'])
ax.set(xlabel='')
ax.xaxis.set_major_locator(years)
ax.xaxis.set_major_formatter(years_fmt)
ax.xaxis.set_minor_locator(months)

datemin = np.datetime64(df.index[0], 'Y')
datemax = np.datetime64(df.index[-1], 'Y') + np.timedelta64(1, 'Y')
ax.set_xlim(datemin, datemax)

ax.format_xdata = mdates.DateFormatter('%Y-%m-%d')
ax.format_ydata = lambda x: '$%1.2f' % x  # format the price.
ax.grid(True)

fig.autofmt_xdate()
plt.title('Actual Sales of: {0}'.format(good_names_unique[0]))
plt.show()
```



Observations

- This is how a plot of a unique region, division and facility looks like with enough data
- It has enough data for future forecasting

Now saving all the plots of all unique combinations in "Plots" folder for future reference

In [15]:
```
def save_plots(data_pairs):
    '''
        input pairs in this function and save each unique data's plot image in it's respective
        folder
    '''
    months = mdates.MonthLocator()  # every month
    years = mdates.YearLocator()   # every year
    years_fmt = mdates.DateFormatter('%Y-%m-%d')

    for idx in range(len(data_pairs)):
        pairs = data_pairs[idx]
        label = pairs[0]
        path = pairs[1]
        plot_fig = pd.read_pickle(path)
        fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15,7))  # create figure & 1 axis
        # ax.plot('Dates', 'Actual_Sales', data=df)
        ax.plot(plot_fig['Actual_Sales'])
        ax.set(xlabel='')
        ax.xaxis.set_major_locator(years)
        ax.xaxis.set_major_formatter(years_fmt)
        ax.xaxis.set_minor_locator(months)

        datemin = np.datetime64(df.index[0], 'Y')
        datemax = np.datetime64(df.index[-1], 'Y') + np.timedelta64(1, 'Y')
        ax.set_xlim(datemin, datemax)

        ax.format_xdata = mdates.DateFormatter('%Y-%m-%d')
        ax.format_ydata = lambda x: '$%1.2f' % x  # format the price.
        ax.grid(True)

        fig.autofmt_xdate()
        plt.title('Actual Sales')
#         plt.show()

        imglabel=label+".png"
        final_path=("./plots"+"/"+imglabel)
        fig.savefig(final_path, bbox_inches='tight')   # save the figure to file
        plt.close(fig)
```

In [28]:
```
# each good unique combination's data plot is stored in the plots folder
start = datetime.now()
save_plots(pairs)
print('Time taken: ', datetime.now() - start)
```

```
Time taken:  0:02:53.272155
```

Plotting Functions

In [20]:
```
def get_plots(region, division_name, facility_name):
    '''
        Shows the plot of any input unique combination
    '''
    print(get_plots.__doc__)
    file_name="./Plots/"+region+"_"+division_name+"_"+facility_name+".png"
    print('\nFile name: ', file_name)
    display(Image.open(file_name))
```

In [15]:
```
def analysis_plots(name, path=None, data=None):
    '''
        Returns KDE, QQ and Autocorrelation plots of the input data
    '''
    print(analysis_plots.__doc__)
    plt.rcParams["figure.figsize"] = (8,5)
    if path == None:
        df_ana_t = data
    else:
        df_ana_t = pd.read_pickle(path)
        df_ana_t.fillna(1, inplace=True)

    plt.figure(1)
    sscaler = StandardScaler()
    np.random.seed(0)
    x = np.random.randn(100)
    vals = df_ana_t.Actual_Sales.values
    vals = vals.reshape(-1, 1)
    sscaled_vals = sscaler.fit_transform(vals)
    sns.distplot(sscaled_vals, hist=False, label='KDE')
    sns.distplot(x, hist=False, label='Gaussian')
    plt.title('KDE of: {0}'.format(name))
    plt.ylabel('Probability Density')
    plt.legend()

    plt.figure(1, 3)
    qqplot(df_ana_t.Actual_Sales, line='s')
    plt.title('Q-Q Plot of: {0}'.format(name))

    plt.figure(1, 4)
    sm.graphics.tsa.plot_acf(df_ana_t.Actual_Sales)
    plt.title('Autocorrelation of: {0}'.format(name))
    plt.show()
```

Testing Stationarity function

In [16]:
```
# test stationary

def test_stationarity(timeseries):
    '''
      this function tests stationarity of data using Dickey Fuller test
    '''
    print(test_stationarity.__doc__)
    #Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()

    #Plot rolling statistics
    plt.rcParams["figure.figsize"] = (20,8)
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

Analysing one of the unique combinations

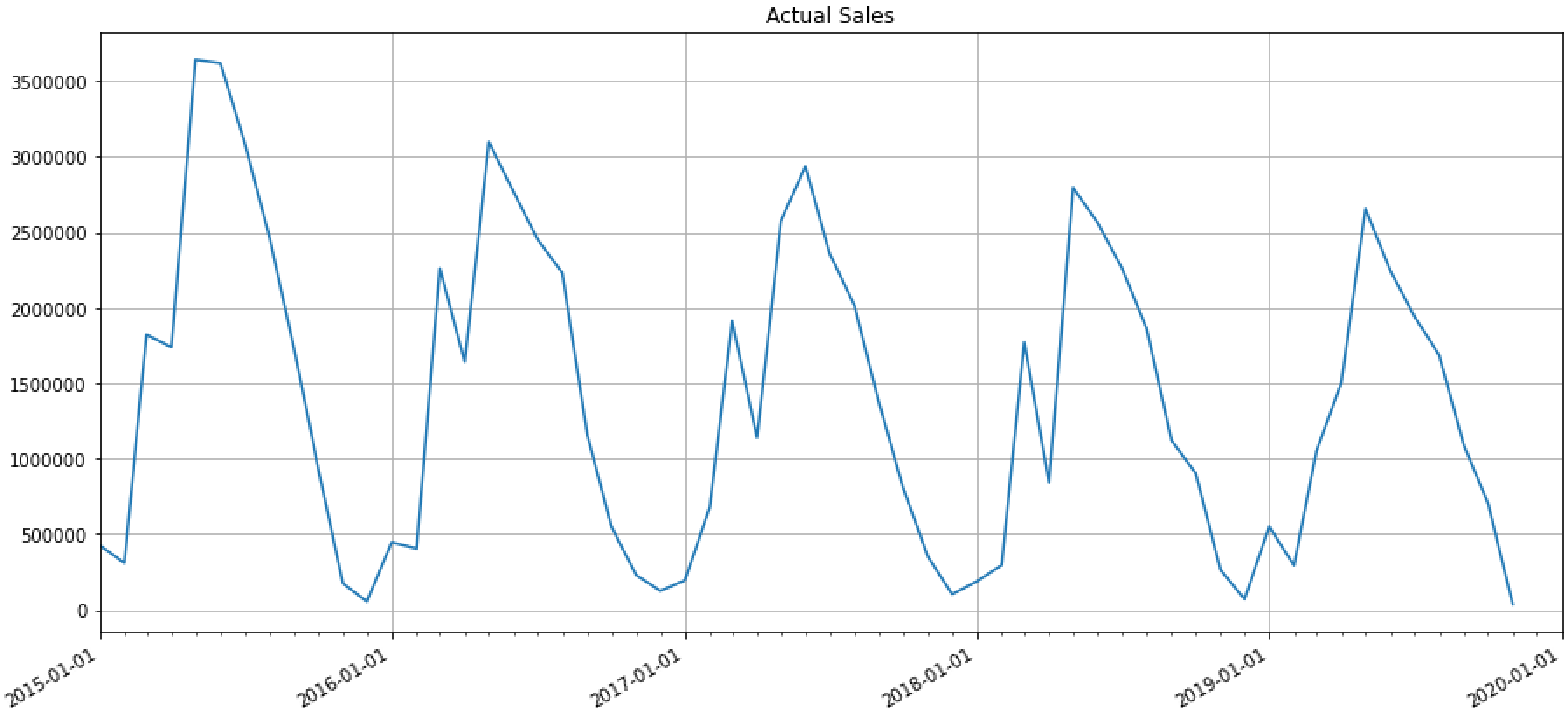
Note: The data analysis done below is before stationarity tests

In [18]:
```
good_names_unique[1]
```

Out[18]:
```
'Canada_Expocrete_Acheson'
```

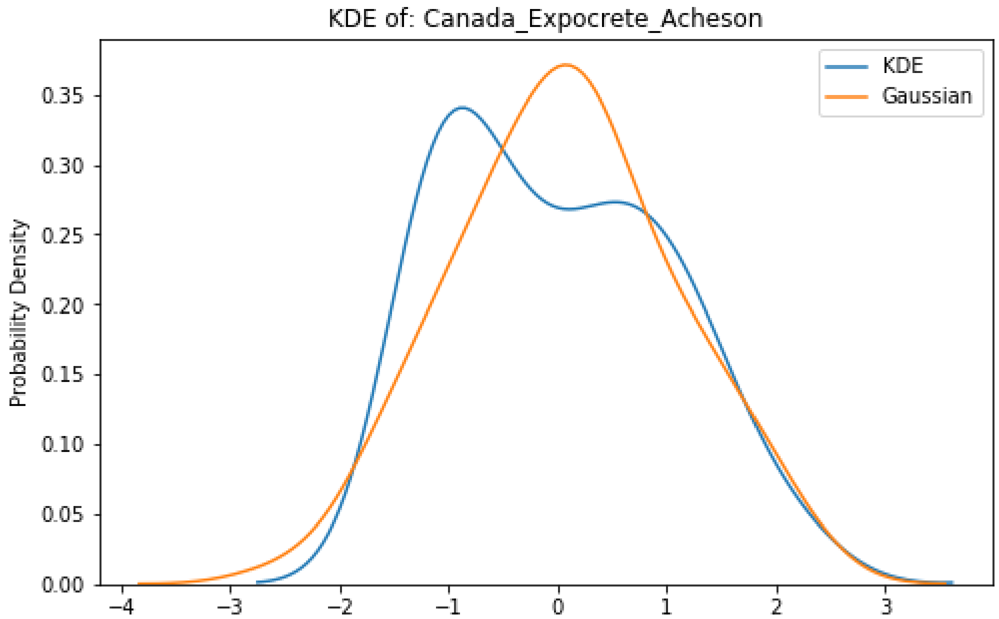
```
In [100]: get_plots("Canada", "Expocrete", "Acheson")
```

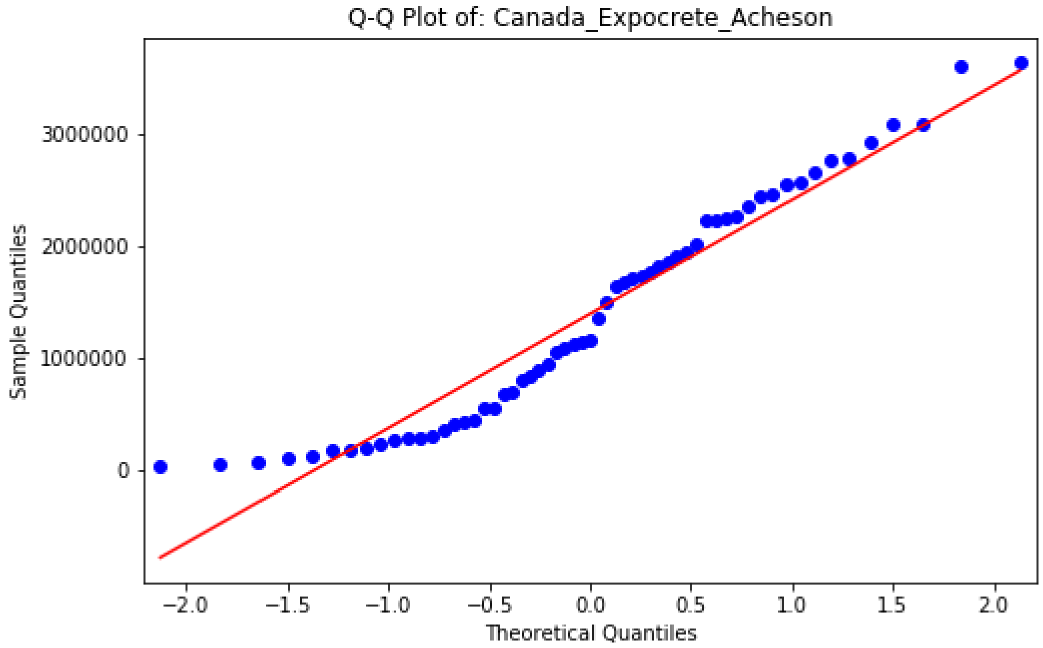
```
            Shows the plot of any input unique combination


File name:  ./Plots/Canada_Expocrete_Acheson.png
```

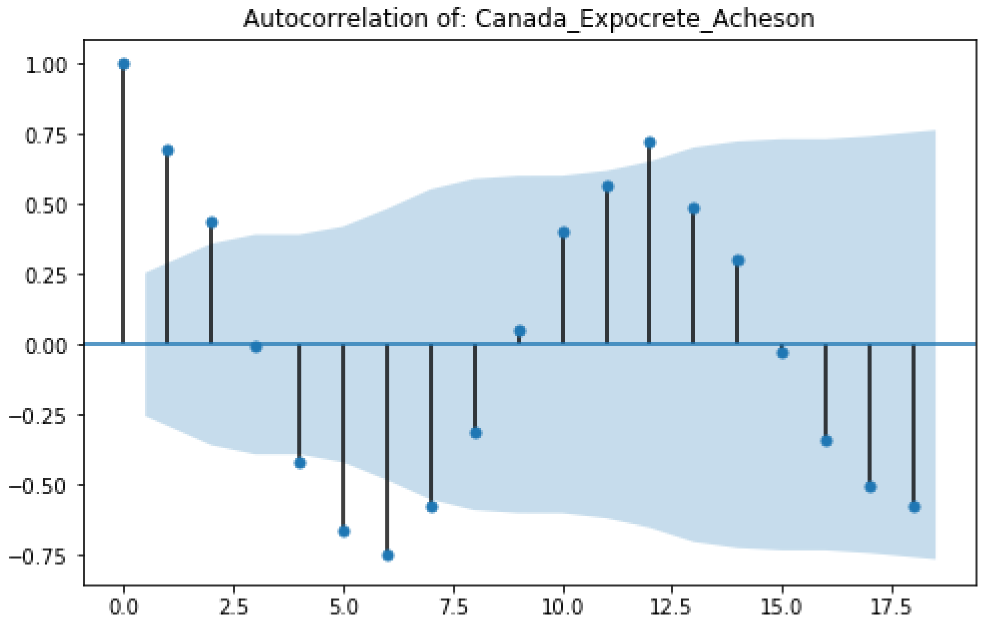


In [176]:
```
# this plot is of a unique data on which stationarity tests have not been done yet
start = datetime.now()
analysis_plots(good_names_unique[1], good_paths_unique[1])
print('Time taken: ', datetime.now() - start)
```

```
Returns KDE, QQ and Autocorrelation plots of the input data
```








```
Time taken:  0:00:00.540144
```

Observations

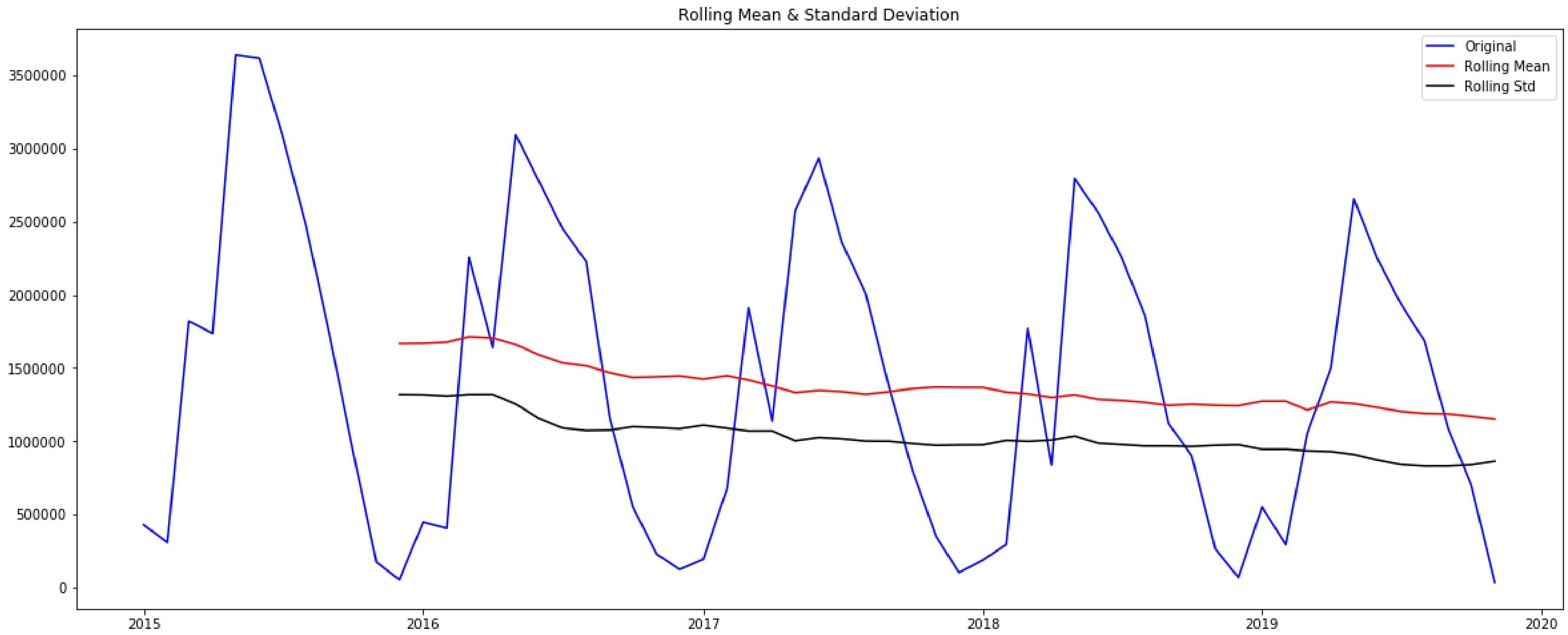
- KDE and Q-Q plot shows signs of stationarity and gaussian distribution of our data
- Autocorrelation(ACF) plot also shows good correlation of data till lag 3, where the values are above more confidence intervals and shows positive correlation

Now, let's test stationarity of the combination above

We are using Dickey Fuller test to test the stationarity of the model

In [20]:
```
df_1 = good_data_unique[1]
test_stationarity(df_1['Actual_Sales'].resample('MS').mean().fillna("1").astype(float))
```

```
this function tests stationarity of data using Dickey Fuller test
```




```
Results of Dickey-Fuller Test:
Test Statistic                 -1.404210
p-value                         0.580256
#Lags Used                     11.000000
Number of Observations Used    47.000000
Critical Value (1%)            -3.577848
Critical Value (5%)            -2.925338
Critical Value (10%)           -2.600774
dtype: float64
```

Observations

- we can see that the data is not stationary
- p-value is above the threshold of 0.005
- we need to convert this data into stationary
- but first let's run a smoothing model and check the performance metric before making it stationary

Note that below the model is run on data without making it stationary

Running Holt-Winters Model/Triple Exponential Smoothing on the above data

In [32]:
```
def mape(y_true, y_pred):
    '''
        Mean Absolute Percentage Error
    '''
    if len(y_true) != len(y_pred):
        print('true and predicted values are not of same length')

    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

In [70]:
```
print('Working on this data first before making it stationary: ', good_names_unique[1])
```

```
Working on this data first before making it stationary:  Canada_Expocrete_Acheson
```

In [19]:
```
eda_df = pd.read_pickle(good_paths_unique[1])
eda_df.head()
```

Out[19]:

Dates	Region	Division_Name	Facility_Name	Year	Month	Actual_Sales
2015-01-01	Canada	Expocrete	Acheson	2015	1	429019
2015-02-01	Canada	Expocrete	Acheson	2015	2	309218
2015-03-01	Canada	Expocrete	Acheson	2015	3	1820819
2015-04-01	Canada	Expocrete	Acheson	2015	4	1738015
2015-05-01	Canada	Expocrete	Acheson	2015	5	3641473

In [20]:
```
print('Shape of our test data: ', eda_df.shape)
```

```
Shape of our test data:  (59, 6)
```

In [21]:
```
# train test split

X_train_eda = eda_df[:'2018-11-01']
X_test_eda = eda_df['2018-11-01':]

print('Shape of train: ', X_train_eda.shape)
print('Shape of test: ', X_test_eda.shape)
```

```
Shape of train:  (47, 6)
Shape of test:  (13, 6)
```

In [22]:
```
print('Values in our Train Data: \n\n', X_train_eda.Actual_Sales)
```

```
Values in our Train Data:

 Dates
2015-01-01     429019
2015-02-01     309218
2015-03-01    1820819
2015-04-01    1738015
2015-05-01    3641473
2015-06-01    3618173
2015-07-01    3095153
2015-08-01    2473930
2015-09-01    1723349
2015-10-01     942322
2015-11-01     174417
2015-12-01      54765
2016-01-01     447548
2016-02-01     406344
2016-03-01    2258727
2016-04-01    1640186
2016-05-01    3095481
2016-06-01    2769057
2016-07-01    2454221
2016-08-01    2228865
2016-09-01    1156924
2016-10-01     551436
2016-11-01     229591
2016-12-01     125317
2017-01-01     194226
2017-02-01     677514
2017-03-01    1911401
2017-04-01    1138480
2017-05-01    2573980
2017-06-01    2934701
2017-07-01    2359913
2017-08-01    2013289
2017-09-01    1362999
2017-10-01     804670
2017-11-01     351737
2017-12-01     103020
2018-01-01     187874
2018-02-01     295922
```

```
2018-03-01    1770734
2018-04-01     838061
2018-05-01    2795981
2018-06-01    2562034
2018-07-01    2263102
2018-08-01    1859135
2018-09-01    1120055
2018-10-01     902617
2018-11-01     263819
Name: Actual_Sales, dtype: int64
```

In [23]:
```
print('Values in our Test Data: \n\n', X_test_eda.Actual_Sales)
```

```
Values in our Test Data:

 Dates
2018-11-01     263819
2018-12-01      70129
2019-01-01     550647
2019-02-01     293627
2019-03-01    1052946
2019-04-01    1502043
2019-05-01    2656542
2019-06-01    2244117
2019-07-01    1942107
2019-08-01    1686924
2019-09-01    1089160
2019-10-01     704241
2019-11-01      35807
Name: Actual_Sales, dtype: int64
```

In [24]:
```
start = len(X_train_eda)
end = (len(X_train_eda) + len(X_test_eda)) - 1
```

Running Model

In [26]:
```
model_eda = ExponentialSmoothing(X_train_eda.Actual_Sales, seasonal_periods=12, trend='add', seasonal='add')
results_eda = model_eda.fit()

print(results_eda.summary().tables[1])
```

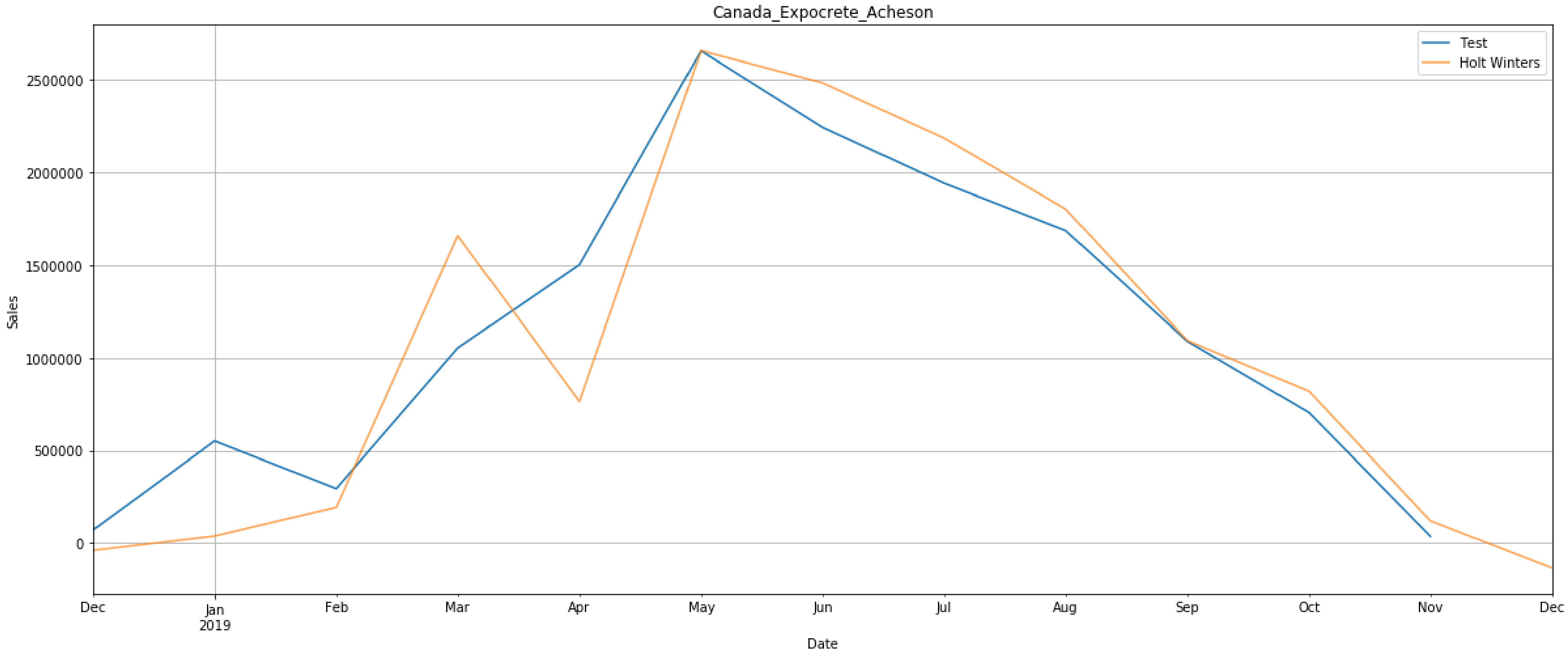
```
================================================================================
                           coeff                 code              optimized
--------------------------------------------------------------------------------
smoothing_level              0.2105263                alpha                 True
smoothing_slope              0.0526316                 beta                 True
smoothing_seasonal           0.7894737                gamma                 True
initial_level               3.1467e+05                  l.0                 True
initial_slope                 0.000000                  b.0                 True
initial_seasons.0           1.1435e+05                  s.0                 True
initial_seasons.1           -5448.7500                  s.1                 True
initial_seasons.2           1.5062e+06                  s.2                 True
initial_seasons.3           1.4233e+06                  s.3                 True
initial_seasons.4           3.3268e+06                  s.4                 True
initial_seasons.5           3.3035e+06                  s.5                 True
initial_seasons.6           2.7805e+06                  s.6                 True
initial_seasons.7           2.1593e+06                  s.7                 True
initial_seasons.8           1.4087e+06                  s.8                 True
initial_seasons.9           6.2766e+05                  s.9                 True
initial_seasons.10         -1.4025e+05                 s.10                 True
initial_seasons.11          -2.599e+05                 s.11                 True
--------------------------------------------------------------------------------
```

In [27]:
```
y_pred_eda = results_eda.predict(start=start, end=end)

y_pred_eda
```

Out[27]:
```
2018-12-01   -3.976361e+04
2019-01-01    3.690713e+04
2019-02-01    1.926455e+05
2019-03-01    1.659179e+06
2019-04-01    7.637956e+05
2019-05-01    2.657711e+06
2019-06-01    2.484884e+06
2019-07-01    2.185215e+06
2019-08-01    1.802918e+06
2019-09-01    1.092824e+06
2019-10-01    8.194760e+05
2019-11-01    1.191370e+05
2019-12-01   -1.353163e+05
Freq: MS, dtype: float64
```

Plotting our forecasts

In [28]:
```
plt.rcParams["figure.figsize"] = (20,8)
ax = eda_df.Actual_Sales['2018-12-01':].plot(label='Test')
y_pred_eda.plot(ax=ax, label='Holt Winters', alpha=.7)
ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.title(good_names_unique[1])
plt.legend()
plt.grid()
plt.show()
```



Observations

- The prediction plot looks slightly close to our observed value

Evaluating Performance Metric

In [31]:
```
# first let's evaluate our MAPE
y_truth = X_test_eda.Actual_Sales
mape_val = mape(y_truth, y_pred_eda)

print('MAPE: ', round(mape_val, 2))
```

```
        Mean Absolute Percentage Error

MAPE:  110.32
```

Observations

- as we can see that the MAPE score before converting the data to sationary or gaussian is 110.32
- which is a very bad performance by our model
- let's test our model performance after converting the data to stationary

Now, we'll make this data stationary and then check our model performance

First we'll perform 1st order differencing/lag to make our data stationary

In [43]:
```
df_stat = pd.read_pickle(good_paths_unique[1])
df_stat['Actual_Sales'] = df_stat['Actual_Sales'] - df_stat['Actual_Sales'].shift(1)
df_stat.fillna(1, inplace=True)

df_stat.head()
```

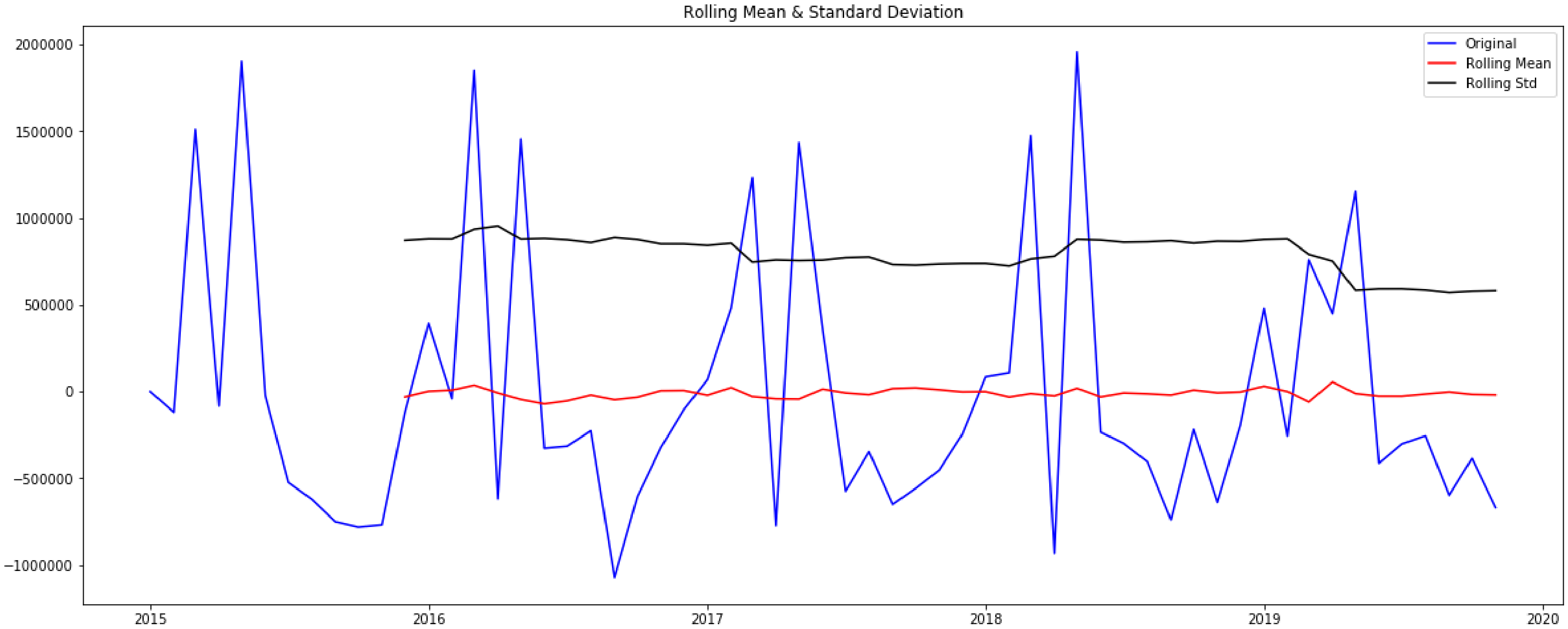
Out[43]:

	Region	Division_Name	Facility_Name	Year	Month	Actual_Sales
Dates						
2015-01-01	Canada	Expocrete	Acheson	2015	1	1.0
2015-02-01	Canada	Expocrete	Acheson	2015	2	-119801.0
2015-03-01	Canada	Expocrete	Acheson	2015	3	1511601.0
2015-04-01	Canada	Expocrete	Acheson	2015	4	-82804.0
2015-05-01	Canada	Expocrete	Acheson	2015	5	1903458.0

Testing the stationarity of data after differencing

In [44]:
```
test_stationarity(df_stat['Actual_Sales'].resample('MS').mean().fillna("1").astype(float))
```

```
this function tests stationarity of data using Dickey Fuller test
```



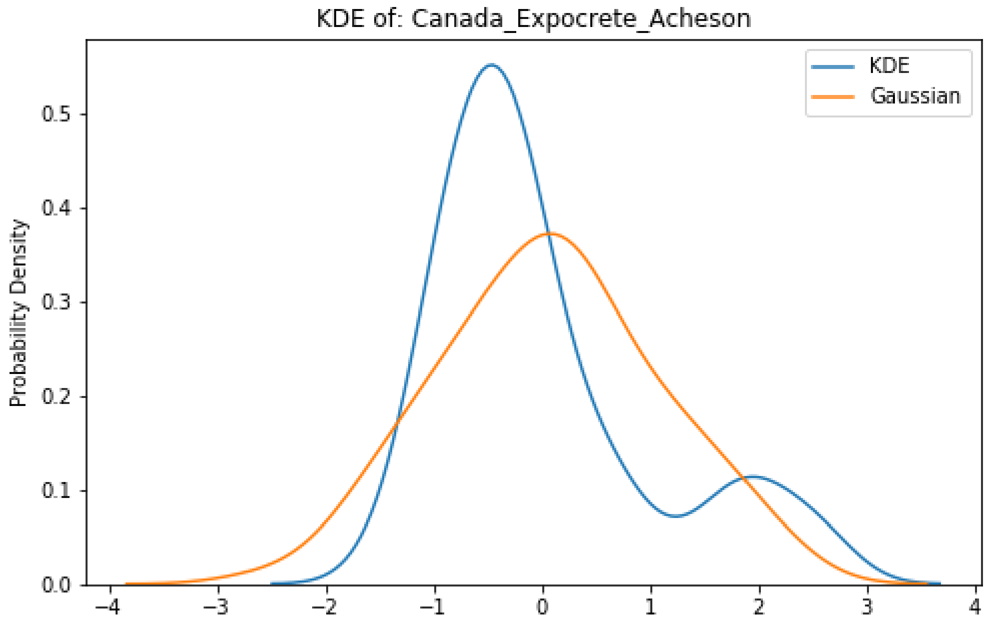

```
Results of Dickey-Fuller Test:
Test Statistic                 -8.685063e+00
p-value                         4.184969e-14
#Lags Used                      1.000000e+01
Number of Observations Used     4.800000e+01
Critical Value (1%)            -3.574589e+00
Critical Value (5%)            -2.923954e+00
Critical Value (10%)           -2.600039e+00
dtype: float64
```

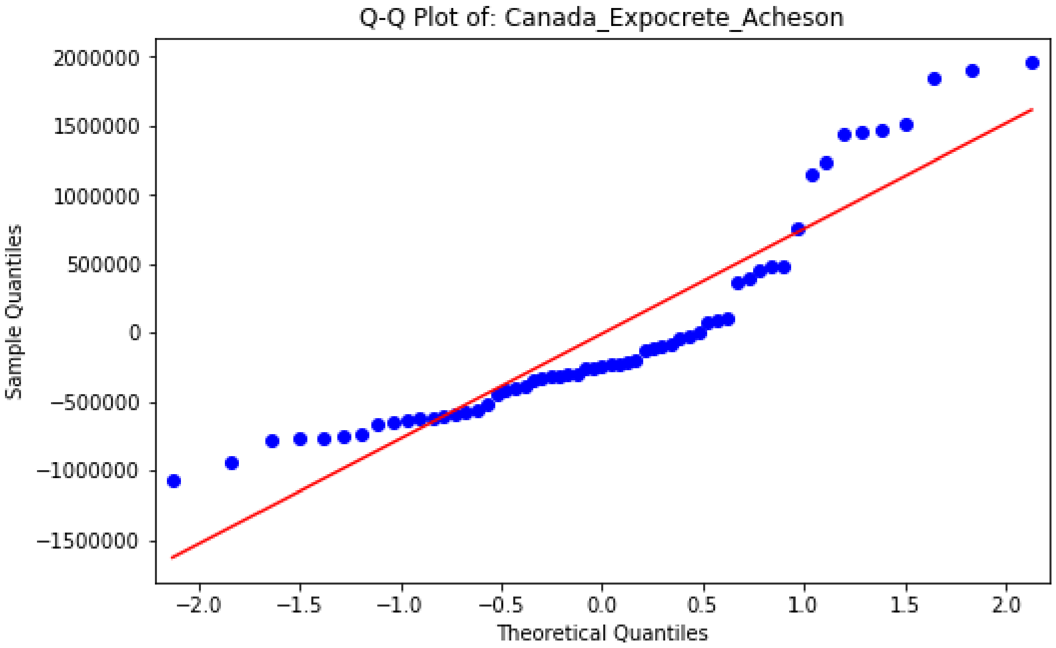
Observations

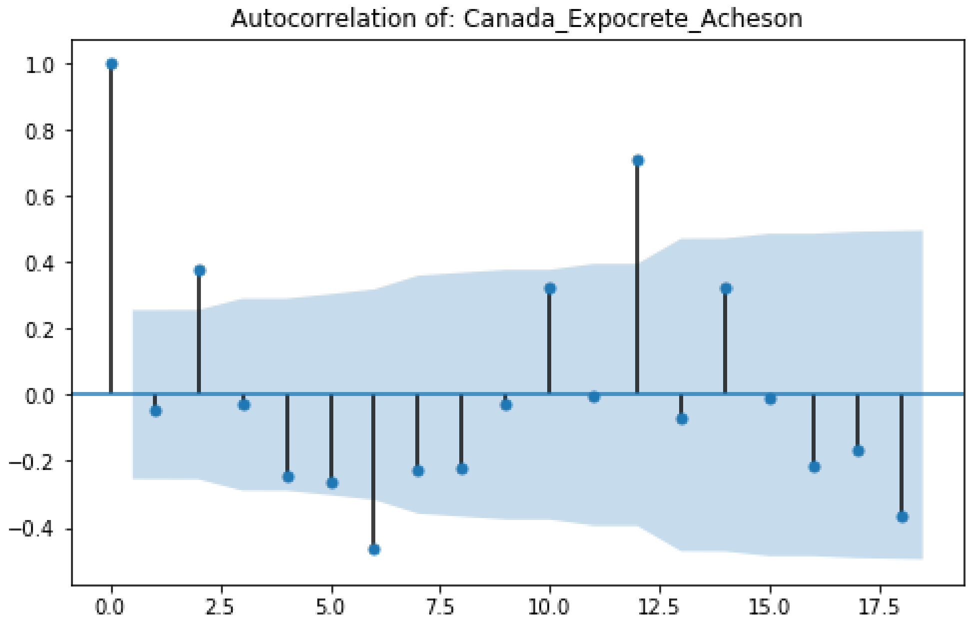
- as we can see above after the stationarity test our p-value has reduced way below the threshold of 0.005
- we can also see the graph and observe the rolling mean and std does not contain any trend and is constant on multiple time steps of the data
- this concludes that our data gets stationary after performing 1st order differencing

In [177]:
```
start = datetime.now()
analysis_plots(good_names_unique[1], data=df_stat)
print('Time taken: ', datetime.now() - start)
```

```
Returns KDE, QQ and Autocorrelation plots of the input data
```








```
Time taken:  0:00:00.523282
```

Observations

- based on our plots we can observe that our Q-Q plot shows that our data is not that normalized or gaussian
- ACF(Autocorrelation) plot shows that our from our 1st lag or with just 1st order differencing we have almost no correlation with our time series data, as it is slightly negative
- but let's test our model performance on this data and observe

Running Model on 1st order differenced data

In [179]:

```
X_train_eda_d = df_stat[:'2018-11-01']
X_test_eda_d = df_stat['2018-11-01':]

print('Shape of train: ', X_train_eda_d.shape)
print('Shape of test: ', X_test_eda_d.shape)
```

```
Shape of train:  (47, 6)
Shape of test:  (13, 6)
```

In [182]:

```
start_d = len(X_train_eda_d)
end_d = (len(X_train_eda_d) + len(X_test_eda_d)) - 1
```

In [180]:
```
model_eda_d = ExponentialSmoothing(X_train_eda_d.Actual_Sales, seasonal_periods=12, trend='add',
                                   seasonal='add')
results_eda_d = model_eda_d.fit()

print(results_eda_d.summary().tables[1])
```

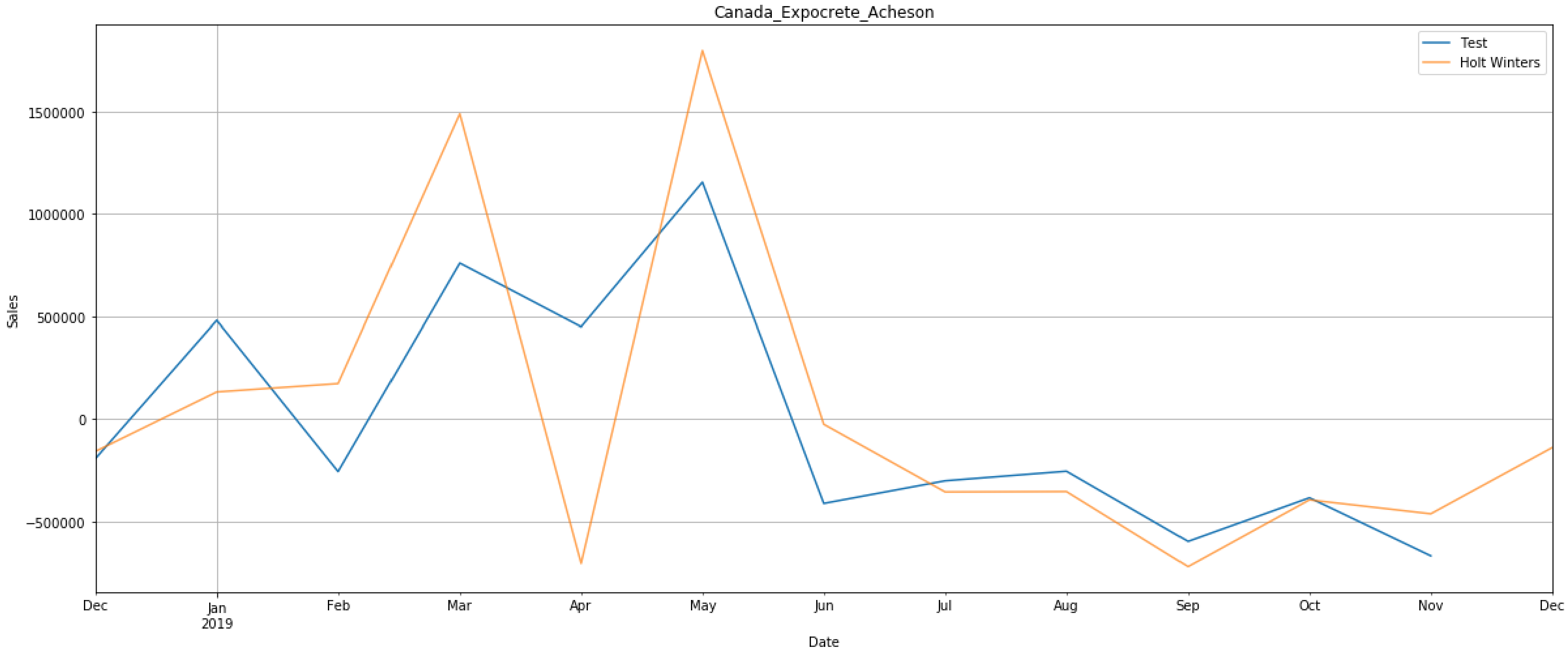
```
================================================================================
                          coeff                 code              optimized
--------------------------------------------------------------------------------
smoothing_level               0.0526316                alpha                True
smoothing_slope               0.0526316                 beta                True
smoothing_seasonal            0.4736842                gamma                True
initial_level                1.3664e+05                  l.0                True
initial_slope                 3088.9236                  b.0                True
initial_seasons.0           -1.3664e+05                  s.0                True
initial_seasons.1           -2.5644e+05                  s.1                True
initial_seasons.2             1.375e+06                  s.2                True
initial_seasons.3           -2.1944e+05                  s.3                True
initial_seasons.4            1.7668e+06                  s.4                True
initial_seasons.5           -1.5994e+05                  s.5                True
initial_seasons.6           -6.5966e+05                  s.6                True
initial_seasons.7           -7.5786e+05                  s.7                True
initial_seasons.8           -8.8722e+05                  s.8                True
initial_seasons.9           -9.1766e+05                  s.9                True
initial_seasons.10          -9.0454e+05                 s.10                True
initial_seasons.11          -2.5629e+05                 s.11                True
--------------------------------------------------------------------------------
```

In [186]:
```
y_pred_eda_d = results_eda_d.predict(start=start_d, end=end_d)

y_pred_eda_d
```

Out[186]:
```
2018-12-01   -1.583356e+05
2019-01-01    1.318928e+05
2019-02-01    1.727794e+05
2019-03-01    1.488185e+06
2019-04-01   -7.053108e+05
2019-05-01    1.796999e+06
2019-06-01   -2.598318e+04
2019-07-01   -3.562523e+05
2019-08-01   -3.545858e+05
2019-09-01   -7.210369e+05
2019-10-01   -3.951379e+05
2019-11-01   -4.631685e+05
2019-12-01   -1.401061e+05
Freq: MS, dtype: float64
```

Plotting our forecasts with 1st order differenced data

In [187]:
```
plt.rcParams["figure.figsize"] = (20,8)
ax = df_stat.Actual_Sales['2018-12-01':].plot(label='Test')
y_pred_eda_d.plot(ax=ax, label='Holt Winters', alpha=.7)
ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.title(good_names_unique[1])
plt.legend()
plt.grid()
plt.show()
```



Evaluating Performance Metric

In [188]:
```
# first let's evaluate our MAPE
y_truth = X_test_eda_d.Actual_Sales
mape_val = mape(y_truth, y_pred_eda_d)

print('MAPE: ', round(mape_val, 2))
```

```
        Mean Absolute Percentage Error

MAPE:  148.35
```

Observations

- this shows that we should not judge our stationarity of our models just by looking at the p-value
- our model is shown stationary after 1st order differencing but the performance metric says otherwise
- MAPE value after 1st order differencing is 148.35 which is worse than the normal data
- let's try using log transform on our data and then check if it turns out to be gaussian and stationary
- then we'll again observe our model performance on that data

Performing Log Transform on our data

In [19]:
```
df_log = pd.read_pickle(good_paths_unique[1])
df_log.head()
```

Out[19]:

	Region	Division_Name	Facility_Name	Year	Month	Actual_Sales
Dates						
2015-01-01	Canada	Expocrete	Acheson	2015	1	429019
2015-02-01	Canada	Expocrete	Acheson	2015	2	309218
2015-03-01	Canada	Expocrete	Acheson	2015	3	1820819
2015-04-01	Canada	Expocrete	Acheson	2015	4	1738015
2015-05-01	Canada	Expocrete	Acheson	2015	5	3641473

In [28]:
```
# log transformation
df_log.Actual_Sales = np.log(df_log.Actual_Sales)
print('Length: ', len(df_log.Actual_Sales))
df_log.Actual_Sales[:5]
```

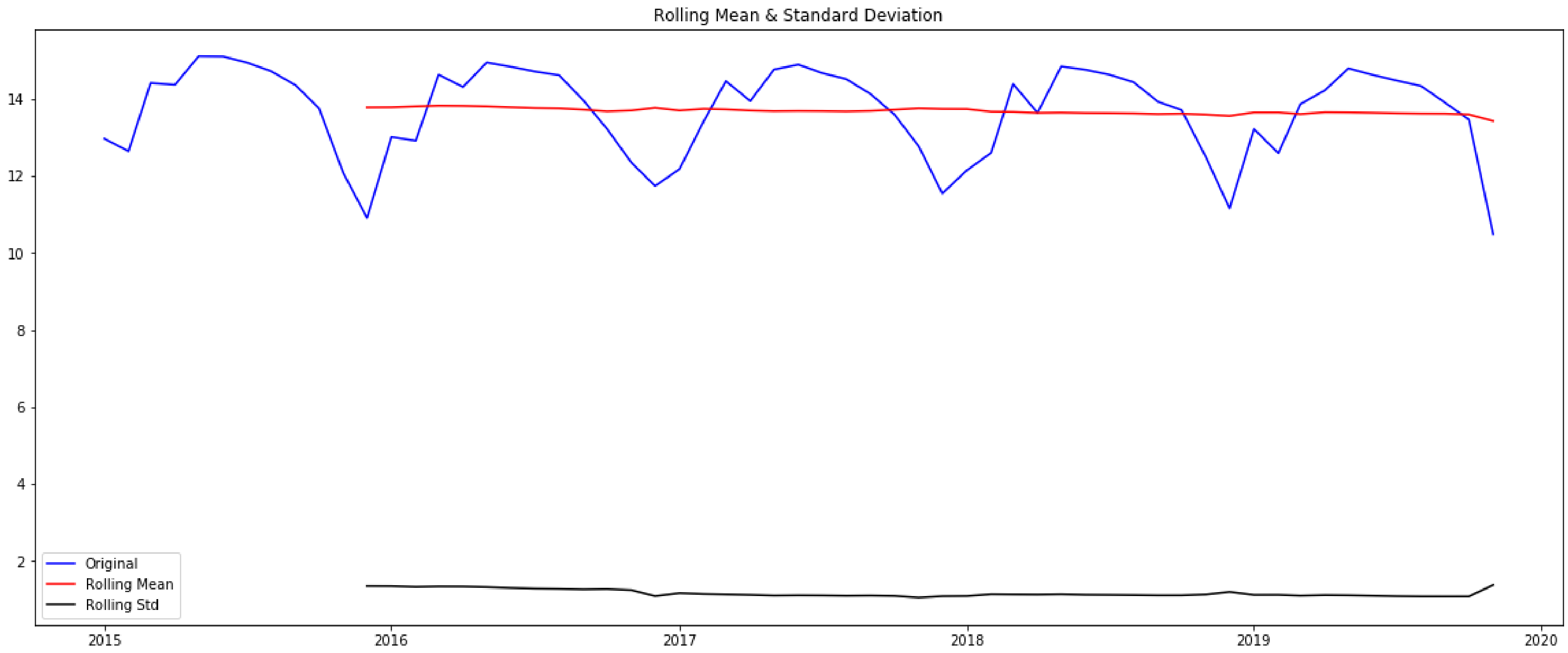
```
Length:  59
```

Out[28]:
```
Dates
2015-01-01    12.969256
2015-02-01    12.641802
2015-03-01    14.414797
2015-04-01    14.368254
2015-05-01    15.107899
Name: Actual_Sales, dtype: float64
```

Testing Stationarity of log transformed data

In [29]:
```
test_stationarity(df_log.Actual_Sales.resample('MS').mean().fillna("1").astype(float))
```

```
        this function tests stationarity of data using Dickey Fuller test
```



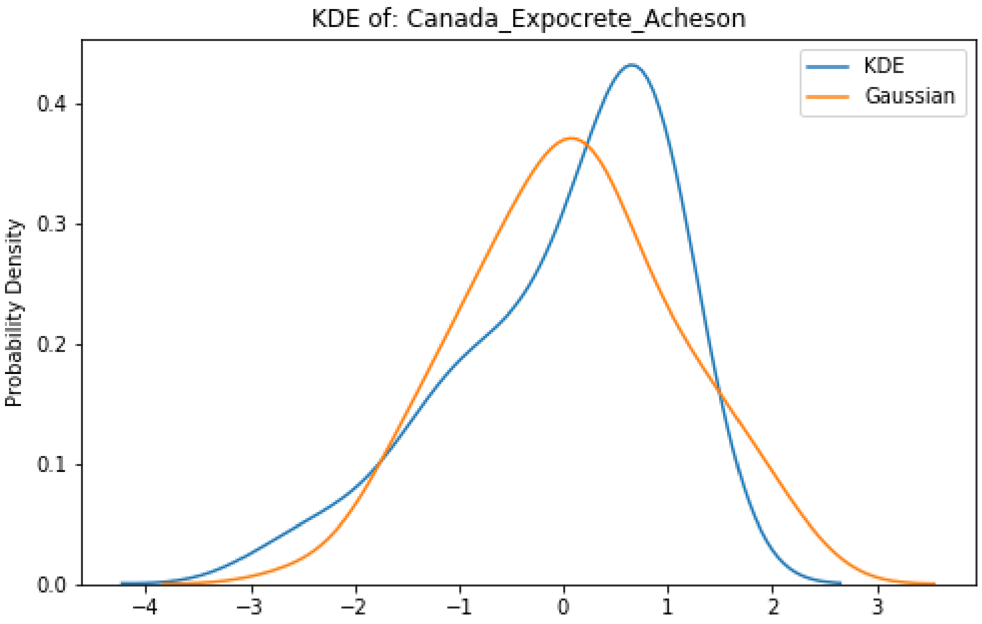

```
Results of Dickey-Fuller Test:
Test Statistic                -0.456766
p-value                        0.900181
#Lags Used                    11.000000
Number of Observations Used   47.000000
Critical Value (1%)           -3.577848
Critical Value (5%)           -2.925338
Critical Value (10%)          -2.600774
dtype: float64
```

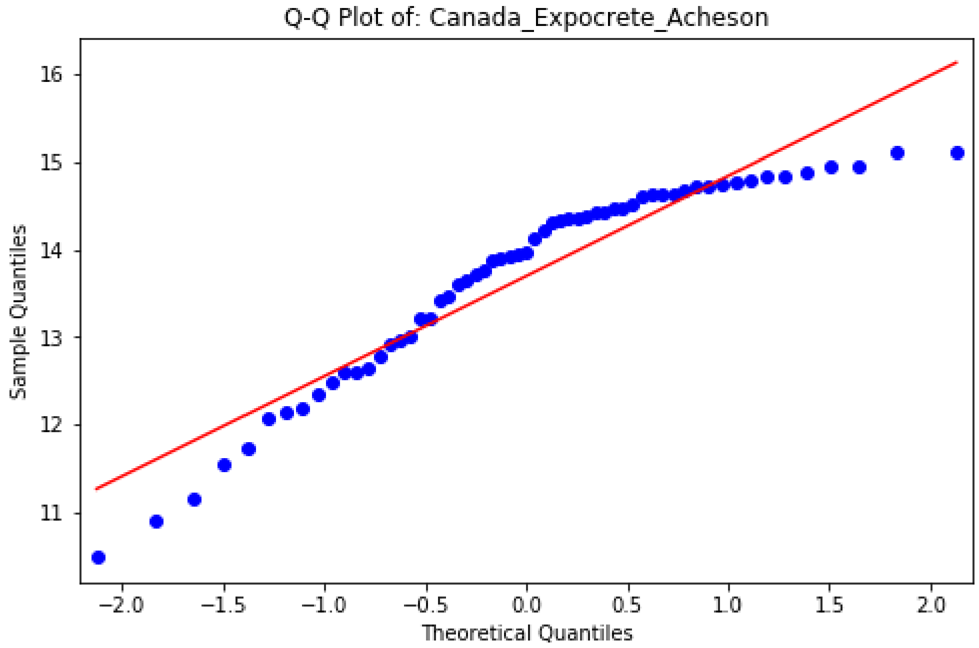
Obseravations

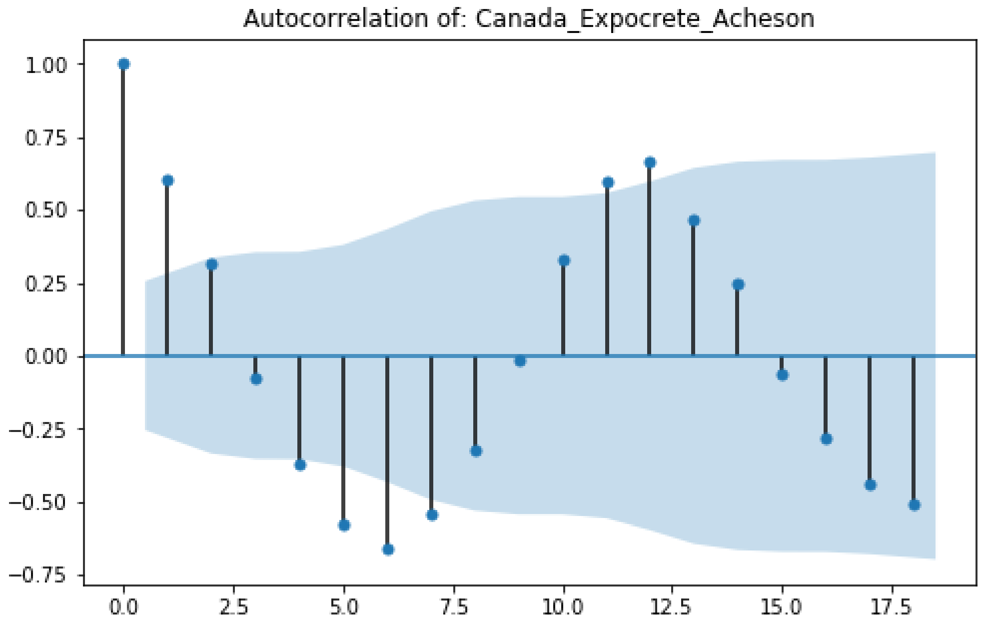
- The data after log transform consists of seasonality and minor downward trend
- Based on the p-value of the Dickey Fuller test we can see that it is way above the threshold of 0.005
- let's see some other plots and check if it has turned gaussian or not

In [30]:
```
start = datetime.now()
analysis_plots(good_names_unique[1], data=df_log)
print('Time taken: ', datetime.now() - start)
```

```
Returns KDE, QQ and Autocorrelation plots of the input data
```








```
Time taken:  0:00:00.540112
```

Observations

- Based on the above plots we can see it is close to a gaussian plot
- Q-Q plot is slightly better than the 1st order differencing plot
- Auto Correlation plot looks better at multiple lags from our time series, it shows positive correlation till lag 3, till lag 2 the data is definitely correlated as it has passed the 95% confidence interval
- Let's check our Model performance and compare the results

Running Model

In [31]:
```
X_train_log = df_log[:'2018-11-01']
X_test_log = df_log['2018-11-01':]

print('Shape of train: ', X_train_log.shape)
print('Shape of test: ', X_test_log.shape)
```

```
Shape of train:  (47, 6)
Shape of test:  (13, 6)
```

In [32]:
```
start_log = len(X_train_log)
end_log = (len(X_train_log) + len(X_test_log)) - 1
```

In [33]:
```
model_log = ExponentialSmoothing(X_train_log.Actual_Sales, seasonal_periods=12, trend='add',
                                 seasonal='add')
results_log = model_log.fit()

print(results_log.summary().tables[1])
```

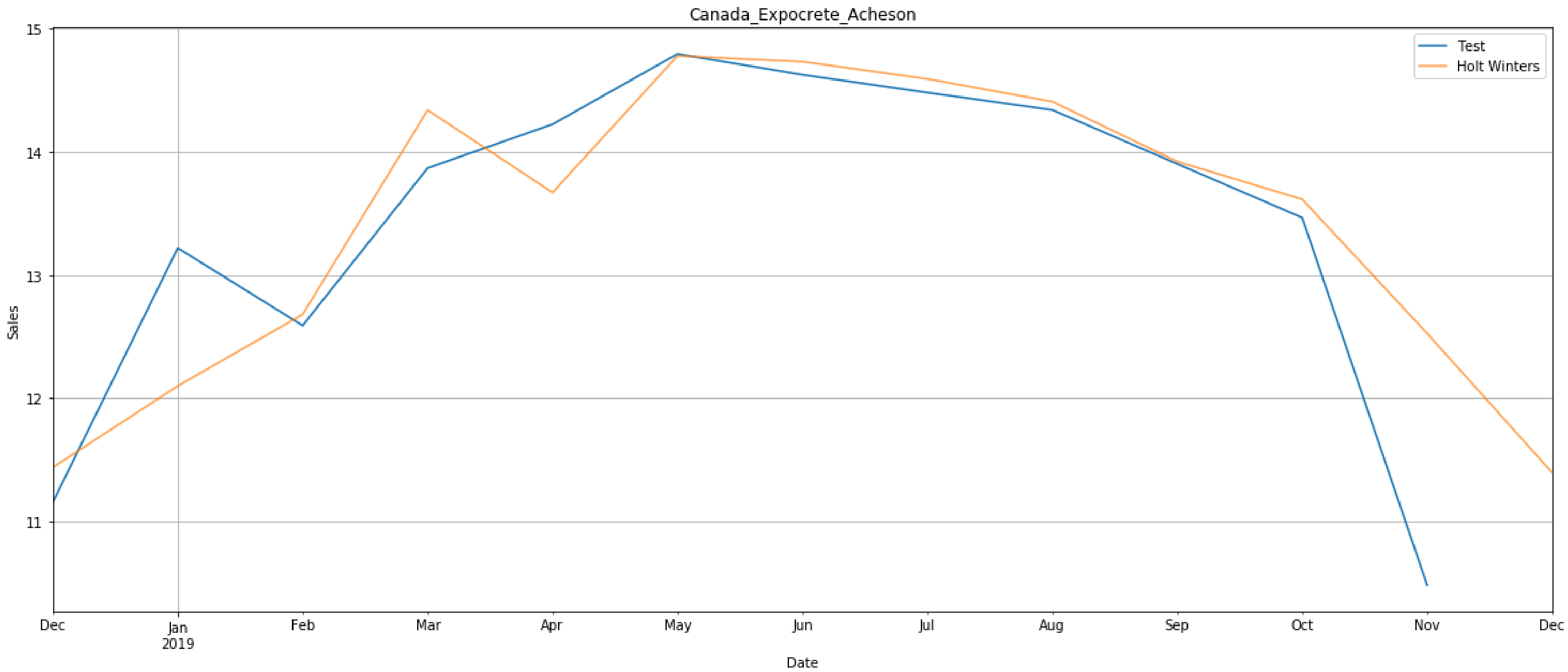
```
================================================================================
                           coeff                 code              optimized
--------------------------------------------------------------------------------
smoothing_level          0.0526313                alpha                  True
smoothing_slope          0.0526357                 beta                  True
smoothing_seasonal       0.7368421                gamma                  True
initial_level            12.575277                  l.0                  True
initial_slope             0.000000                  b.0                  True
initial_seasons.0        0.3939812                  s.0                  True
initial_seasons.1        0.0665278                  s.1                  True
initial_seasons.2        1.8395224                  s.2                  True
initial_seasons.3        1.7929788                  s.3                  True
initial_seasons.4        2.5326234                  s.4                  True
initial_seasons.5        2.5262043                  s.5                  True
initial_seasons.6        2.3700726                  s.6                  True
initial_seasons.7        2.1460435                  s.7                  True
initial_seasons.8        1.7845046                  s.8                  True
initial_seasons.9        1.1808268                  s.9                  True
initial_seasons.10      -0.5060694                 s.10                  True
initial_seasons.11      -1.6644663                 s.11                  True
--------------------------------------------------------------------------------
```

In [34]:
```
y_pred_log = results_log.predict(start=start_log, end=end_log)

y_pred_log
```

Out[34]:
```
2018-12-01    11.440212
2019-01-01    12.099906
2019-02-01    12.680688
2019-03-01    14.337346
2019-04-01    13.668931
2019-05-01    14.777022
2019-06-01    14.730055
2019-07-01    14.590079
2019-08-01    14.406106
2019-09-01    13.919719
2019-10-01    13.614019
2019-11-01    12.526450
2019-12-01    11.398202
Freq: MS, dtype: float64
```

Plotting our Forecasts of log transformed data

In [35]:
```
plt.rcParams["figure.figsize"] = (20,8)
ax = df_log.Actual_Sales['2018-12-01':].plot(label='Test')
y_pred_log.plot(ax=ax, label='Holt Winters', alpha=.7)
ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.title(good_names_unique[1])
plt.legend()
plt.grid()
plt.show()
```



Observations

- Our predicted plot is close to our original values
- Let's check our performance metric

Evaluating Performance Metric

In [38]:
```
y_truth = X_test_log.Actual_Sales
mape_val = mape(y_truth, y_pred_log)

print('MAPE: ', round(mape_val, 2))
```

```
        Mean Absolute Percentage Error

MAPE:  4.76
```

Observations

- As we can see through our valdation score, MAPE is 4.76 which is the best yet after transformation of data
- This is a good example of not relying totally on p-value score of any stationarity tests
- It is always better to look at our plots and check validation score for checking the quality of data
- So we will first run all our models on log transformed data and then later on run it on differenced/lag transformed data as well just for experimental purposes and check the value difference between both the transformations
- We wil check how differently both transformations have affected our Model performance

Log Transforming all our good data and storing it in "log_df" folder

In [19]:
```
len(good_paths_unique)
```

Out[19]:
```
209
```

In [22]:
```
def log_transform(names, paths):
    '''
        This function log transforms data and moves the data to log_df folder
    '''
    print(log_transform.__doc__)
    count = 0
    for i in range(len(names)):
        df_log = pd.read_pickle(paths[i])
        df_log.Actual_Sales = np.log(df_log.Actual_Sales)
        df_log.to_pickle("./log_df/"+names[i]+'.pkl')
        count += 1
    print('Total files transformed: ', count)
```

In [23]:
```
start = datetime.now()
log_transform(good_names_unique, good_paths_unique)
print('\nTime taken: ', datetime.now() - start)
```

```
        This function log transforms data and moves the data to log_df folder

Total files transformed:  209

Time taken:  0:00:01.580256
```

Let's verify the same data from our "log_df" folder

In [17]:
```
start = datetime.now()
good_data_log, good_names_log, good_paths_log, good_pairs_log, bad_names_log = \
get_files_from_dir("./log_df/")
print('Time taken: ', datetime.now() - start)
```

```
        This function checks if files are atleast having 2 years of data


Total files in folder:  208
Time taken:  0:00:01.570463
```

In [18]:
```
good_names_log[1]
```

Out[18]: 'Canada_Expocrete_Acheson.pkl'

In [49]:
```
# for i in range(len(good_paths_log)):
#     if good_names_log[i] == 'East_Anchor_Manasquan NJ.pkl':
#         print(i)
```

In [29]:
```
# first let's see if there is any data which is less than 24 months
print('Number of files in log_df folder which has data less than 24 months: ', len(bad_names_log))
```

```
Number of files in log_df folder which has data less than 24 months:  0
```

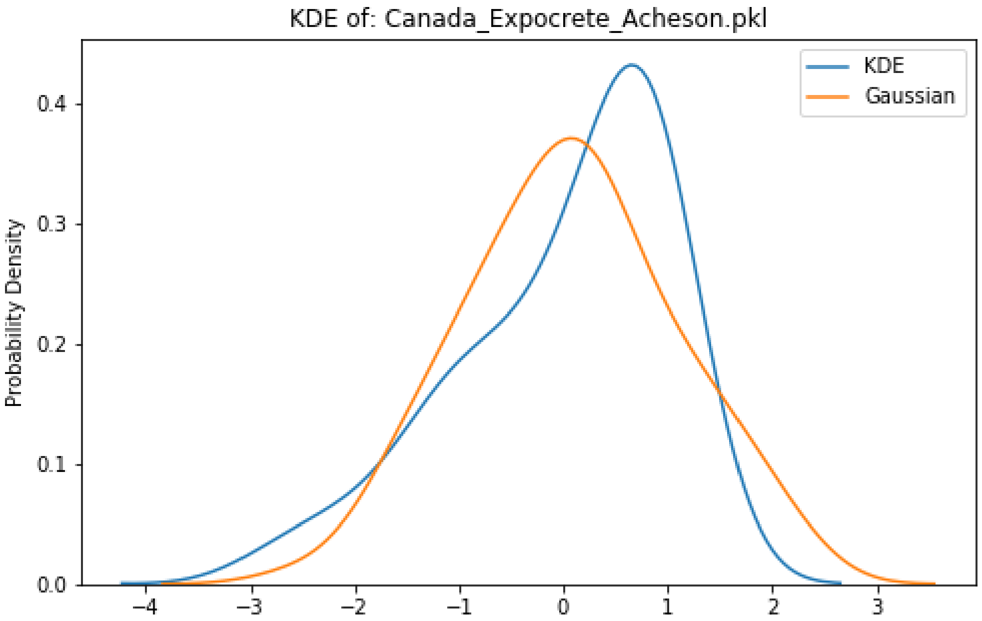
In [19]:
```
df_verify = pd.read_pickle(good_paths_log[1])
df_verify.head()
```

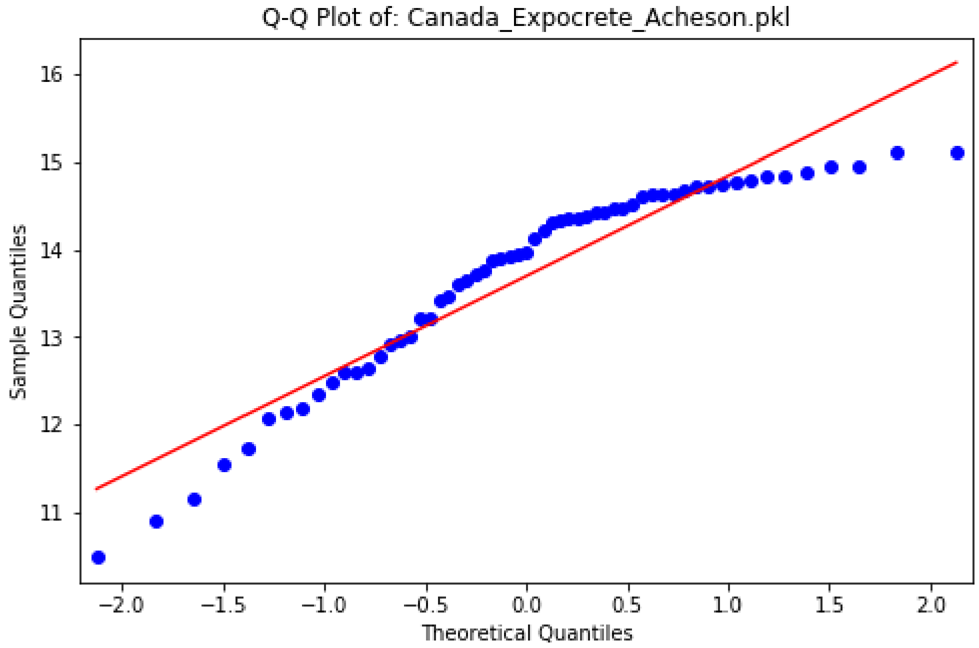
Out[19]:

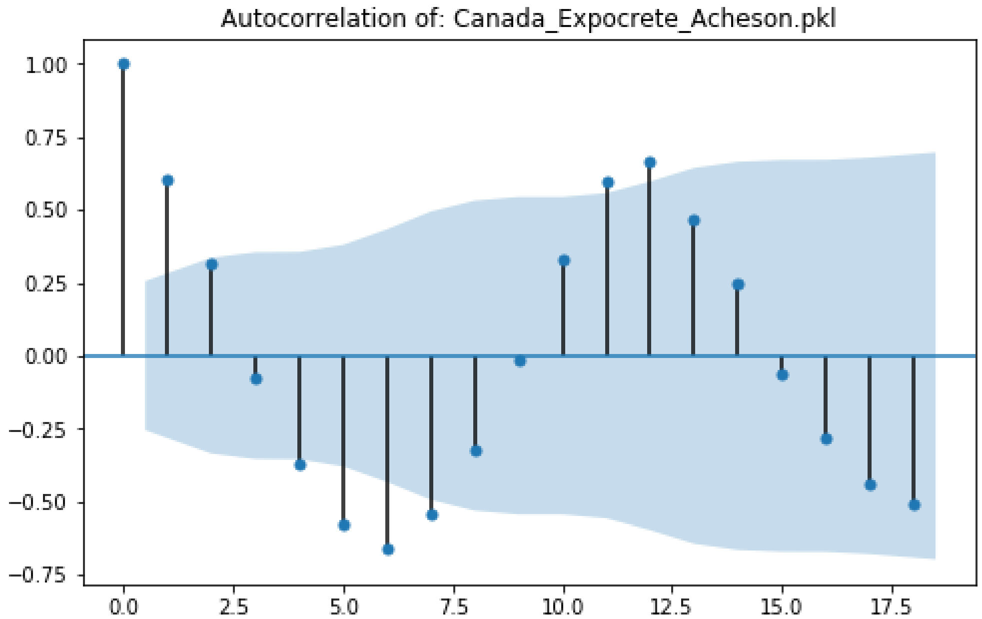
	Region	Division_Name	Facility_Name	Year	Month	Actual_Sales
Dates						
2015-01-01	Canada	Expocrete	Acheson	2015	1	12.969256
2015-02-01	Canada	Expocrete	Acheson	2015	2	12.641802
2015-03-01	Canada	Expocrete	Acheson	2015	3	14.414797
2015-04-01	Canada	Expocrete	Acheson	2015	4	14.368254
2015-05-01	Canada	Expocrete	Acheson	2015	5	15.107899

In [20]:
```
start = datetime.now()
analysis_plots(good_names_log[1], path=good_paths_log[1])
print('Time taken: ', datetime.now() - start)
```

```
Returns KDE, QQ and Autocorrelation plots of the input data
```








```
Time taken:  0:00:00.615034
```

Observations

- As we can see after verification, all our data is log transformed and stored in the log_df folder for future reference

Baseline Models on Log Transformed Data

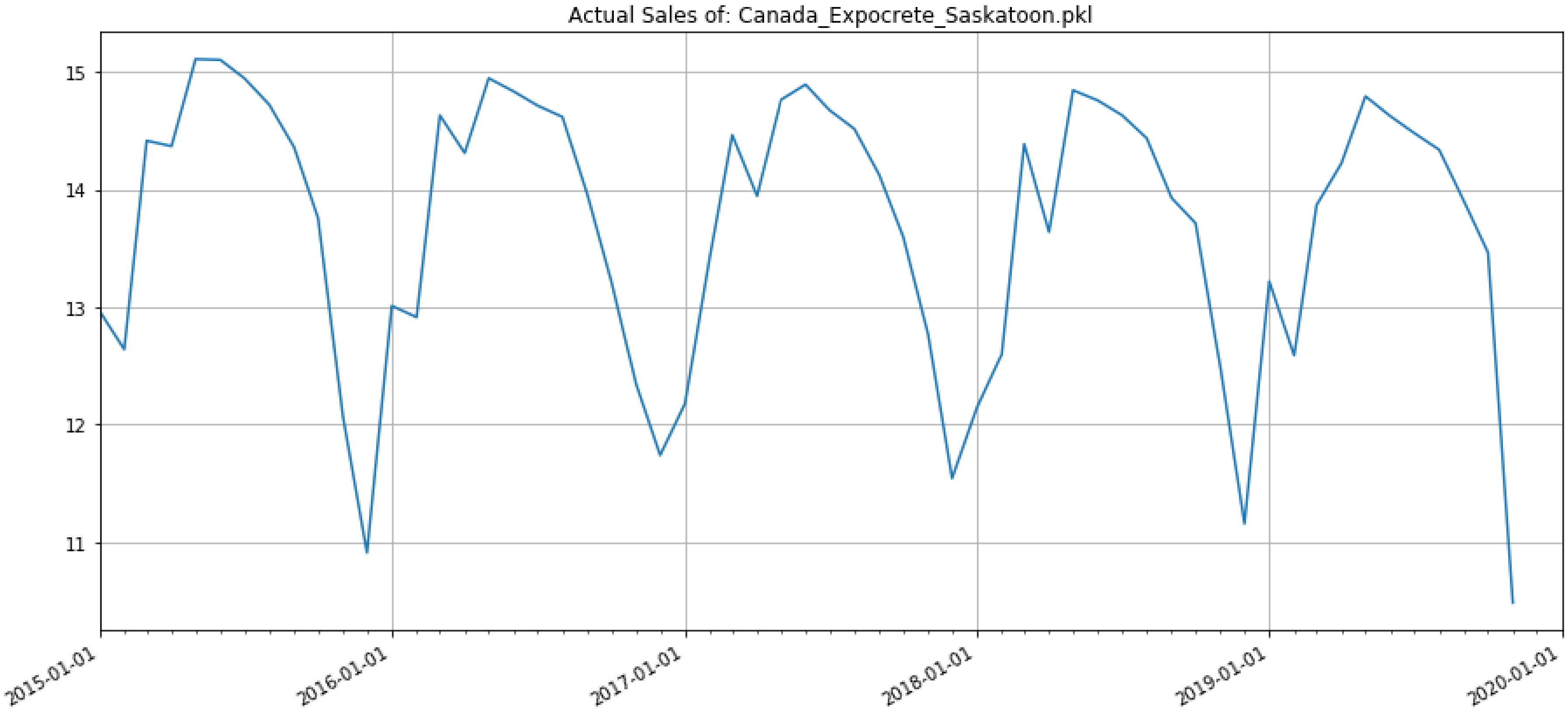
First, testing SARIMAX Model on our log transformed data. similar to what we did above with Holt-Winters Model

In [21]:
```
base_df = pd.read_pickle(good_paths_log[1])
base_df.dropna(inplace=True)
base_df.head()
```

Out[21]:

	Region	Division_Name	Facility_Name	Year	Month	Actual_Sales
Dates						
2015-01-01	Canada	Expocrete	Acheson	2015	1	12.969256
2015-02-01	Canada	Expocrete	Acheson	2015	2	12.641802
2015-03-01	Canada	Expocrete	Acheson	2015	3	14.414797
2015-04-01	Canada	Expocrete	Acheson	2015	4	14.368254
2015-05-01	Canada	Expocrete	Acheson	2015	5	15.107899

In [23]:
```
months = mdates.MonthLocator()  # every month
years = mdates.YearLocator()   # every year
years_fmt = mdates.DateFormatter('%Y-%m-%d')

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15,7))  # create figure & 1 axis
# ax.plot('Dates', 'Actual_Sales', data=df)
ax.plot(base_df['Actual_Sales'])
ax.set(xlabel='')
ax.xaxis.set_major_locator(years)
ax.xaxis.set_major_formatter(years_fmt)
ax.xaxis.set_minor_locator(months)

datemin = np.datetime64(df.index[0], 'Y')
datemax = np.datetime64(df.index[-1], 'Y') + np.timedelta64(1, 'Y')
ax.set_xlim(datemin, datemax)

ax.format_xdata = mdates.DateFormatter('%Y-%m-%d')
ax.format_ydata = lambda x: '$%1.2f' % x  # format the price.
ax.grid(True)

fig.autofmt_xdate()
plt.title('Actual Sales of: {0}'.format(good_names_log[5]))
plt.show()
```

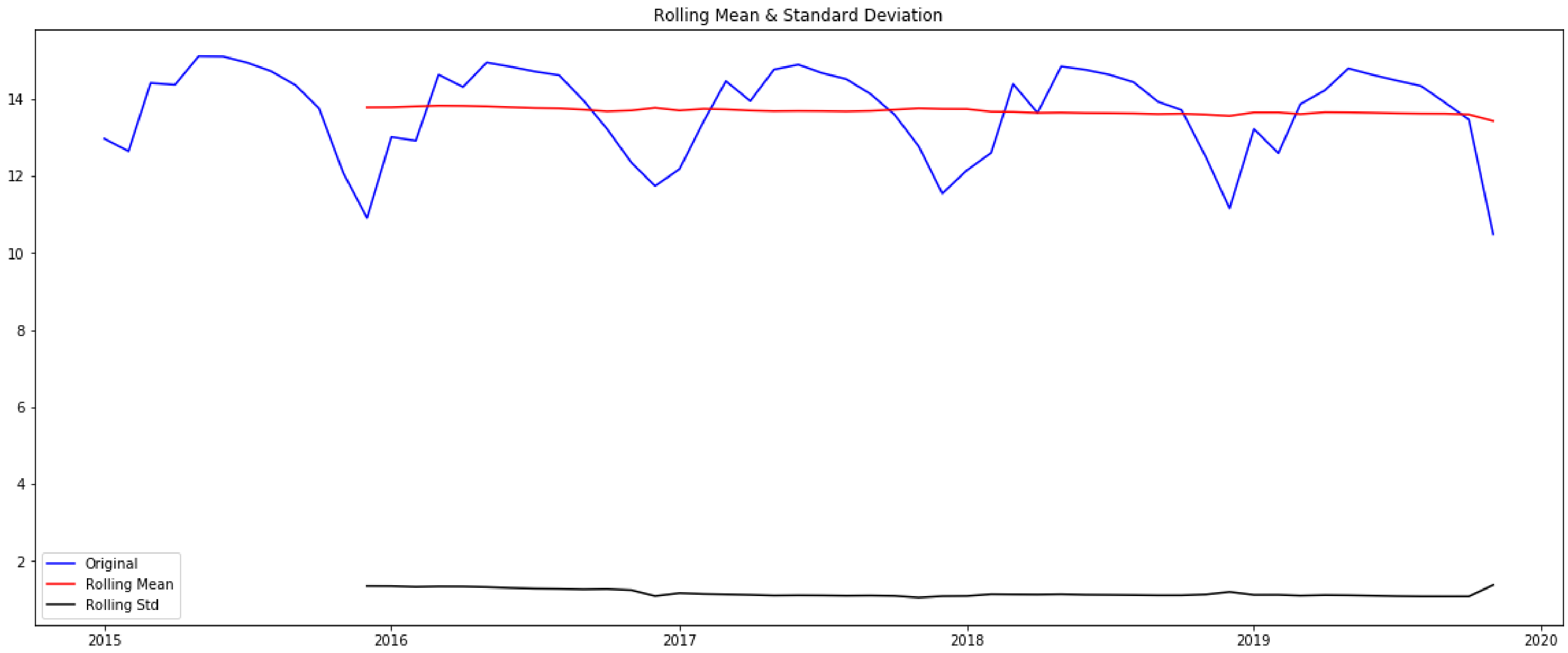


Observations

- This particular data shows good seasonality and slightly downward trend

In [24]:
```
# let's check stationarity for our test data to verify everything we have done till now works or not
test_stationarity(base_df['Actual_Sales'].resample('MS').mean().fillna("1").astype(float))
```

```
        this function tests stationarity of data using Dickey Fuller test
```




```
Results of Dickey-Fuller Test:
Test Statistic                 -0.456766
p-value                         0.900181
#Lags Used                     11.000000
Number of Observations Used    47.000000
Critical Value (1%)            -3.577848
Critical Value (5%)            -2.925338
Critical Value (10%)           -2.600774
dtype: float64
```

In [22]:
```
X_train_base = base_df[:'2018-11-01']
X_test_base = base_df['2018-11-01':]

print('Shape of train: ', X_train_base.shape)
print('Shape of test: ', X_test_base.shape)
```

```
Shape of train:  (47, 6)
Shape of test:  (13, 6)
```

In [23]:
```
print('Test data: ', X_test_base.Actual_Sales)
```

```
Test data:  Dates
2018-11-01    12.483019
2018-12-01    11.158092
2019-01-01    13.218849
2019-02-01    12.590066
2019-03-01    13.867103
2019-04-01    14.222337
2019-05-01    14.792536
2019-06-01    14.623823
2019-07-01    14.479284
2019-08-01    14.338417
2019-09-01    13.900917
2019-10-01    13.464876
2019-11-01    10.485899
Name: Actual_Sales, dtype: float64
```

In [24]:
```
start_base = len(X_train_base)
end_base = (len(X_train_base) + len(X_test_base)) - 1
print(start_base, end_base)
```

```
47 59
```

Testing SARIMAX Model on log transformed data

In [25]:
```
# 12 in seasonal_pdq is just yearly becuase our seasonal data is yearly
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

print('These are all the possible combinations individually\n')
print(pdq)
print()
print(seasonal_pdq)
print('\n')

print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
These are all the possible combinations individually

[(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)]

[(0, 0, 0, 12), (0, 0, 1, 12), (0, 1, 0, 12), (0, 1, 1, 12), (1, 0, 0, 12), (1, 0, 1, 12), (1, 1, 0, 12), (1, 1, 1, 1
2)]


Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

Gridsearch for our SARIMAX hyperparameters

In [26]:
```
# hyperparameter tuning

# params = []
# params_seasonal = []
aic_vals_base = []
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(X_train_base.Actual_Sales,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)

            results = mod.fit()

            print('SARIMA{}x{} - AIC:{}'.format(param, param_seasonal, results.aic))
#             params.append(param)
#             params_seasonal.append(param_seasonal)
            aic_vals_base.append((param, param_seasonal, results.aic))
        except:
            continue
```

```
SARIMA(0, 0, 0)x(0, 0, 0, 12) - AIC:374.19500186934823
SARIMA(0, 0, 0)x(0, 0, 1, 12) - AIC:1587.4640978146697
SARIMA(0, 0, 0)x(0, 1, 0, 12) - AIC:25.78715928715626
SARIMA(0, 0, 0)x(1, 0, 0, 12) - AIC:25.76945764514025
SARIMA(0, 0, 0)x(1, 0, 1, 12) - AIC:25.850797872434935
SARIMA(0, 0, 0)x(1, 1, 0, 12) - AIC:17.290527618738533
SARIMA(0, 0, 1)x(0, 0, 0, 12) - AIC:313.70907616041484
SARIMA(0, 0, 1)x(0, 0, 1, 12) - AIC:2523.4747167998157
SARIMA(0, 0, 1)x(0, 1, 0, 12) - AIC:27.165394350199797
SARIMA(0, 0, 1)x(1, 0, 0, 12) - AIC:27.11939852870235
SARIMA(0, 0, 1)x(1, 0, 1, 12) - AIC:25.736619673925265
SARIMA(0, 0, 1)x(1, 1, 0, 12) - AIC:19.26916925519024
SARIMA(0, 1, 0)x(0, 0, 0, 12) - AIC:115.20281651735976
SARIMA(0, 1, 0)x(0, 0, 1, 12) - AIC:1321.5172472404495
SARIMA(0, 1, 0)x(0, 1, 0, 12) - AIC:51.390918167183194
SARIMA(0, 1, 0)x(1, 0, 0, 12) - AIC:46.887521406856536
SARIMA(0, 1, 0)x(1, 0, 1, 12) - AIC:1192.1809325526756
SARIMA(0, 1, 0)x(1, 1, 0, 12) - AIC:29.79890328963231
SARIMA(0, 1, 1)x(0, 0, 0, 12) - AIC:111.05155197904658
SARIMA(0, 1, 1)x(0, 0, 1, 12) - AIC:1435.853561396154
SARIMA(0, 1, 1)x(0, 1, 0, 12) - AIC:29.368588733095372
SARIMA(0, 1, 1)x(1, 0, 0, 12) - AIC:27.034717312133782
SARIMA(0, 1, 1)x(1, 0, 1, 12) - AIC:1397.0116142780191
SARIMA(0, 1, 1)x(1, 1, 0, 12) - AIC:20.816782085260733
SARIMA(1, 0, 0)x(0, 0, 0, 12) - AIC:118.7772309615505
SARIMA(1, 0, 0)x(0, 0, 1, 12) - AIC:1740.8279246307234
SARIMA(1, 0, 0)x(0, 1, 0, 12) - AIC:27.481556817491686
SARIMA(1, 0, 0)x(1, 0, 0, 12) - AIC:48.63710751735803
SARIMA(1, 0, 0)x(1, 0, 1, 12) - AIC:49.519359288506145
SARIMA(1, 0, 0)x(1, 1, 0, 12) - AIC:12.280878243119739
SARIMA(1, 0, 1)x(0, 0, 0, 12) - AIC:118.98840378023093
SARIMA(1, 0, 1)x(0, 0, 1, 12) - AIC:1546.552024623027
SARIMA(1, 0, 1)x(0, 1, 0, 12) - AIC:29.073058786222404
SARIMA(1, 0, 1)x(1, 0, 0, 12) - AIC:28.594856030818065
SARIMA(1, 0, 1)x(1, 0, 1, 12) - AIC:28.067194697296898
SARIMA(1, 0, 1)x(1, 1, 0, 12) - AIC:11.979156723849863
SARIMA(1, 1, 0)x(0, 0, 0, 12) - AIC:117.07507932615226
SARIMA(1, 1, 0)x(0, 0, 1, 12) - AIC:1278.9697116306563
SARIMA(1, 1, 0)x(0, 1, 0, 12) - AIC:40.35582924685278
SARIMA(1, 1, 0)x(1, 0, 0, 12) - AIC:36.13909619993472
SARIMA(1, 1, 0)x(1, 0, 1, 12) - AIC:1238.8842964734536
```

```
SARIMA(1, 1, 0)x(1, 1, 0, 12) - AIC:12.744834896972609
SARIMA(1, 1, 1)x(0, 0, 0, 12) - AIC:112.91379419143362
SARIMA(1, 1, 1)x(0, 0, 1, 12) - AIC:1446.5864645835009
SARIMA(1, 1, 1)x(0, 1, 0, 12) - AIC:30.884607559637743
SARIMA(1, 1, 1)x(1, 0, 0, 12) - AIC:28.626522681285632
SARIMA(1, 1, 1)x(1, 0, 1, 12) - AIC:1407.781240619415
SARIMA(1, 1, 1)x(1, 1, 0, 12) - AIC:8.501581927903816
```

In [27]:
```
# print(len(params), len(params_seasonal), len(aic_vals))
print('Length of params: {0} and an example {1}'.format(len(aic_vals_base), aic_vals_base[0]))

clean_aic_vals_base = [(i, j, k) for i, j, k in aic_vals_base if not np.isnan(k)]

print('After removing nan values, length:', len(clean_aic_vals_base))

scores_base = [k for i, j, k in clean_aic_vals_base]
idx = np.argmin(scores_base)
print('Best params for SARIMAX are: {0}x{1} and score: {2}'.format(clean_aic_vals_base[idx][0],
                                                               clean_aic_vals_base[idx][1],
                                                               clean_aic_vals_base[idx][2]))
```

```
Length of params: 48 and an example ((0, 0, 0), (0, 0, 0, 12), 374.19500186934823)
After removing nan values, length: 48
Best params for SARIMAX are: (1, 1, 1)x(1, 1, 0, 12) and score: 8.501581927903816
```

Training our Best Model

In [28]:
```
model = sm.tsa.statespace.SARIMAX(X_train_base.Actual_Sales,
                                order=clean_aic_vals_base[idx][0],
                                seasonal_order=clean_aic_vals_base[idx][1],
                                enforce_stationarity=False,
                                enforce_invertibility=False)

results_final = model.fit()

print(results_final.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2451      0.352      0.697      0.486      -0.444       0.935
ma.L1         -1.0000   1007.897     -0.001      0.999   -1976.442    1974.442
ar.S.L12      -0.3057      0.106     -2.876      0.004      -0.514      -0.097
sigma2         0.0540     54.457      0.001      0.999    -106.680     106.788
==============================================================================
```

In [29]:
```
pred_base = results_final.get_prediction(start=start_base, end=end_base,
                              dynamic=False)
pred_ci_base = pred_base.conf_int()
```

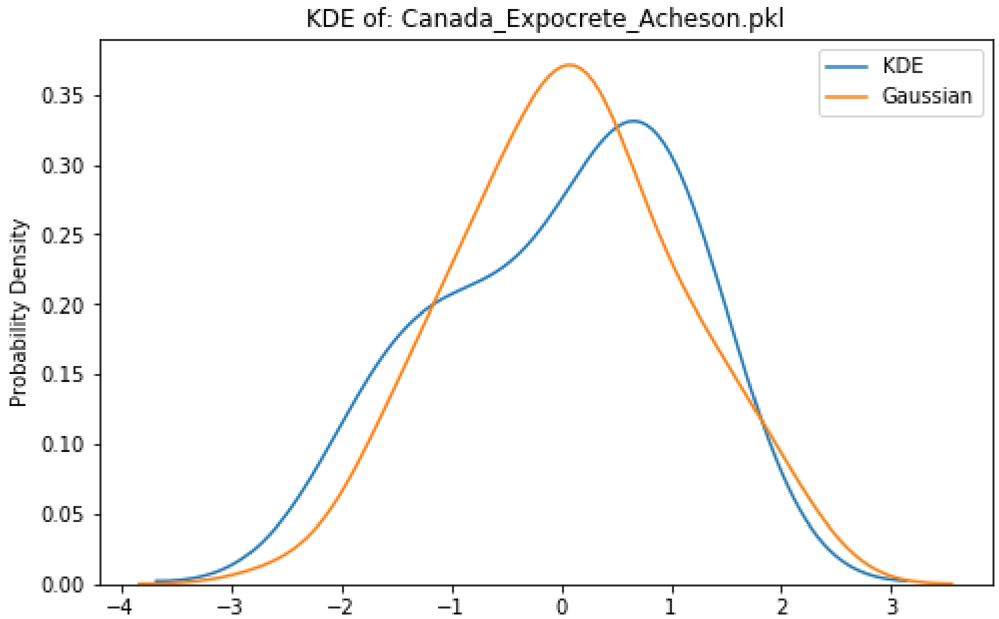
In [30]:
```
preds_base = pred_base.predicted_mean
preds_base.index = X_test_base.index
preds_base
```

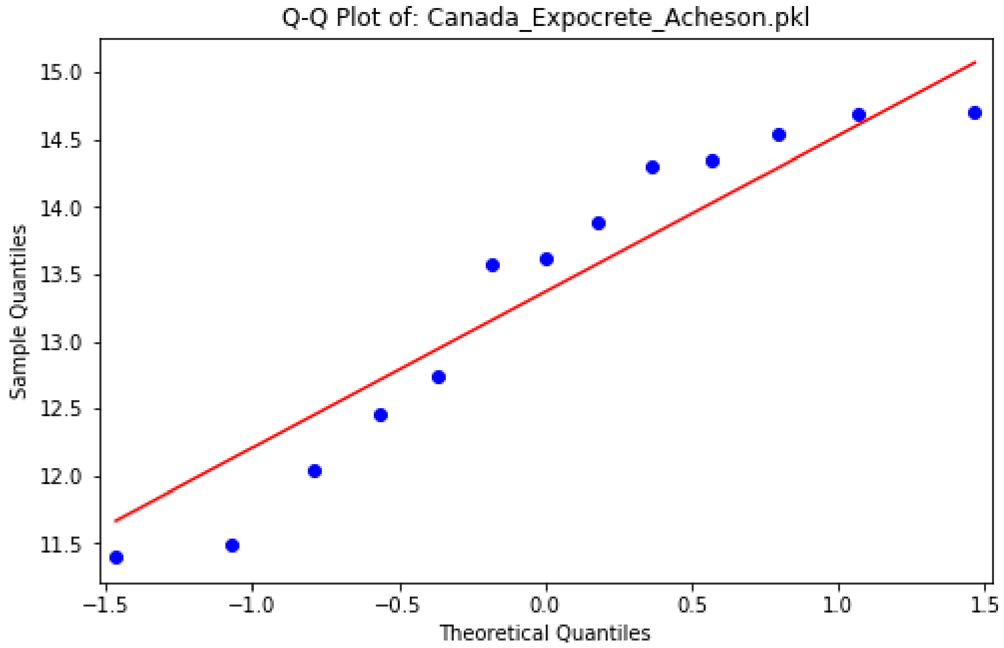
Out[30]:
```
Dates
2018-11-01    11.482904
2018-12-01    12.043224
2019-01-01    12.742869
2019-02-01    14.302615
2019-03-01    13.624979
2019-04-01    14.710912
2019-05-01    14.690347
2019-06-01    14.537573
2019-07-01    14.352495
2019-08-01    13.881423
2019-09-01    13.570459
2019-10-01    12.463467
2019-11-01    11.393698
dtype: float64
```

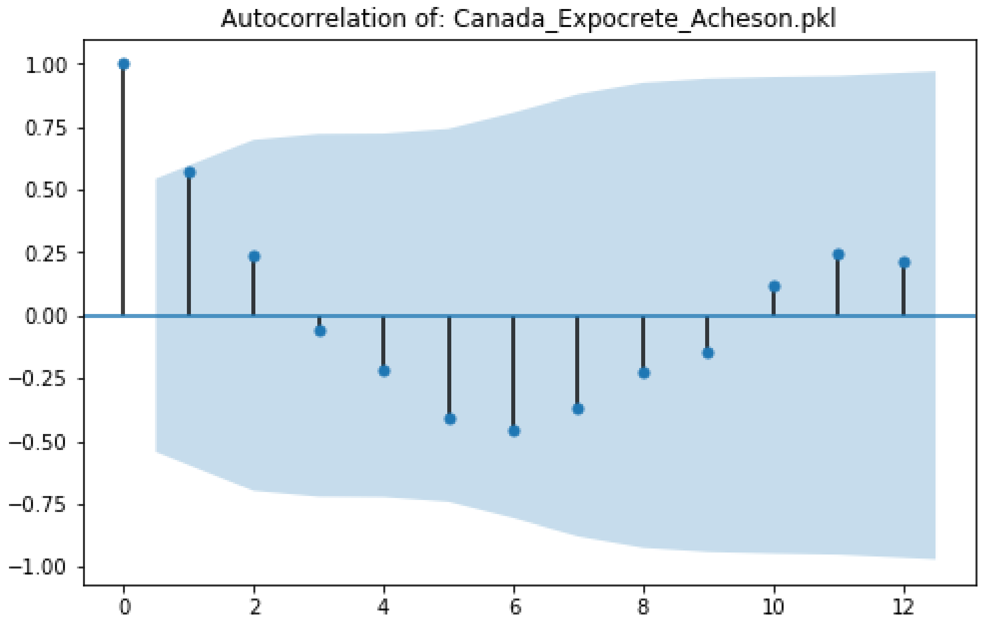
In [38]:
```
# converting our predicted values from series to a dataframe for out plot analysis
pb = preds_base.to_frame(name='Actual_Sales')
```

In [37]:
```
start = datetime.now()
analysis_plots(good_names_log[1], data=pb)
print('Time taken: ', datetime.now() - start)
```

```
Returns KDE, QQ and Autocorrelation plots of the input data
```







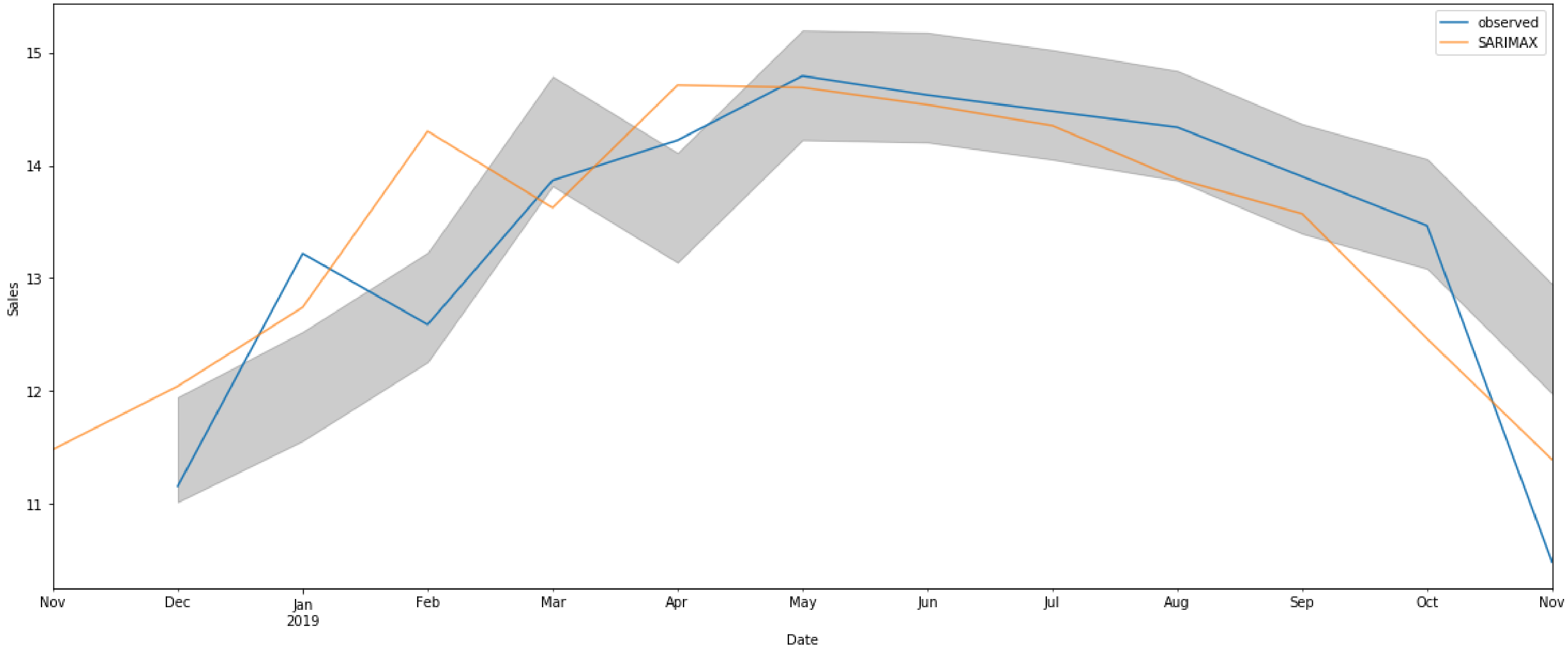

```
Time taken:  0:00:00.485280
```

Observations

- We can see that out predicted values are gaussian and looks similar to the original distribution
- we can also the ACF plot where it shows a positive correlation for lag 1 as given to our SARIMAX model while training
- Q-Q plot doesn't show a good result, there might be multiple reasons like less data etc

Plotting our SARIMAX forecasts

In [39]:
```
plt.rcParams["figure.figsize"] = (20,8)
ax = X_test_base.Actual_Sales['2018-12-01':].plot(label='observed')
preds_base.plot(ax=ax, label='SARIMAX', alpha=.7)

ax.fill_between(pred_ci_base.index,
                pred_ci_base.iloc[:, 0],
                pred_ci_base.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.legend()

plt.show()
```



Observations

- Our predicted plot looks close to our original values
- Let's check our Performance metric

Evaluating Performance Metric

In [40]:
```
y_truth = X_test_base.Actual_Sales
mape_val = mape(y_truth, preds_base)

print('MAPE: ', round(mape_val, 2))
```

```
MAPE:  4.78
```

Observations

- As we can see MAPE Score of our SARIMAX Model is 4.78 and it is almost same as our Holt-Winters Model score
- there is only a 0.02 difference between our Holt-Winters and SARIMAX Model performance

We will run 3 Models - SARIMAX, Holt-Winters & Exponential Weighted Avg Models on all the log-transformed data

Sarimax Model

In [128]:
```
def sarimax_on_all(names, paths, pdq_ls, seasonal_pdq_ls):
    '''
        Running SARIMAX model with hyperparameter tuning on all files,
        this captures seasonality as well as trend
    '''
    print(sarimax_on_all.__doc__)
    sari_di = {} # dictionary of sarimax model, contains name and mape score
    count = 1
    for i in range(len(names)):
        df_sari = pd.read_pickle(paths[i])
        df_sari.dropna(inplace=True)
        train = df_sari[:'2018-11-01']
        test = df_sari['2018-11-01':]
        start = len(train)
        end = (len(train) + len(test)) - 1

        # now hyperparameter tuning
        aic_vals = []
        for param in pdq_ls:
            for param_seasonal in seasonal_pdq_ls:
                try:
                    mod = sm.tsa.statespace.SARIMAX(train.Actual_Sales,
                                                    order=param,
                                                    seasonal_order=param_seasonal,
                                                    enforce_stationarity=False,
                                                    enforce_invertibility=False)
                    results = mod.fit()
                    aic_vals.append((param, param_seasonal, results.aic))
                except:
                    continue

        # selecting best params
        clean_aic_vals = [(i, j, k) for i, j, k in aic_vals if not np.isnan(k)]
        scores = [k for i, j, k in clean_aic_vals]
        idx_min = np.argmin(scores)

        # traning our model with best params
        model = sm.tsa.statespace.SARIMAX(train.Actual_Sales,
                                          order=clean_aic_vals[idx_min][0],
                                          seasonal_order=clean_aic_vals[idx_min][1],
                                          enforce_stationarity=False,
```

```
                                            enforce_invertibility=False)

        results_final = model.fit()

        # predicting on test data
        pred = results_final.get_prediction(start=start, end=end, dynamic=False)
        preds = pred.predicted_mean
        preds.index = test.index

        # calculating MAPE score
        y_pred_sari = pred.predicted_mean
        y_true = test.Actual_Sales
        mape_val = mape(y_true, y_pred_sari)
        sari_di[names[i]] = mape_val

        print('{0} Files---done!'.format(count))
        count += 1

    return sari_di
```

In [129]:
```
start = datetime.now()
sarimax_di_log = sarimax_on_all(good_names_log, good_paths_log, pdq, seasonal_pdq)
print('\nTime taken: ', datetime.now() - start)
```

```
running SARIMAX model with hyperparameter tuning on all files,
this captures seasonality as well as trend
```

```
1 Files---done!
2 Files---done!
3 Files---done!
4 Files---done!
5 Files---done!
6 Files---done!
7 Files---done!
8 Files---done!
9 Files---done!
10 Files---done!
11 Files---done!
12 Files---done!
13 Files---done!
14 Files---done!
15 Files---done!
16 Files---done!
17 Files---done!
18 Files---done!
19 Files---done!
20 Files---done!
21 Files---done!
22 Files---done!
23 Files---done!
24 Files---done!
25 Files---done!
26 Files---done!
27 Files---done!
28 Files---done!
29 Files---done!
30 Files---done!
31 Files---done!
32 Files---done!
33 Files---done!
34 Files---done!
35 Files---done!
36 Files---done!
37 Files---done!
38 Files---done!
```

```
39 Files---done!
40 Files---done!
41 Files---done!
42 Files---done!
43 Files---done!
44 Files---done!
45 Files---done!
46 Files---done!
47 Files---done!
48 Files---done!
49 Files---done!
50 Files---done!
51 Files---done!
52 Files---done!
53 Files---done!
54 Files---done!
55 Files---done!
56 Files---done!
57 Files---done!
58 Files---done!
59 Files---done!
60 Files---done!
61 Files---done!
62 Files---done!
63 Files---done!
64 Files---done!
65 Files---done!
66 Files---done!
67 Files---done!
68 Files---done!
69 Files---done!
70 Files---done!
71 Files---done!
72 Files---done!
73 Files---done!
74 Files---done!
75 Files---done!
76 Files---done!
77 Files---done!
78 Files---done!
79 Files---done!
80 Files---done!
```

```
81 Files---done!
82 Files---done!
83 Files---done!
84 Files---done!
85 Files---done!
86 Files---done!
87 Files---done!
88 Files---done!
89 Files---done!
90 Files---done!
91 Files---done!
92 Files---done!
93 Files---done!
94 Files---done!
95 Files---done!
96 Files---done!
97 Files---done!
98 Files---done!
99 Files---done!
100 Files---done!
101 Files---done!
102 Files---done!
103 Files---done!
104 Files---done!
105 Files---done!
106 Files---done!
107 Files---done!
108 Files---done!
109 Files---done!
110 Files---done!
111 Files---done!
112 Files---done!
113 Files---done!
114 Files---done!
115 Files---done!
116 Files---done!
117 Files---done!
118 Files---done!
119 Files---done!
120 Files---done!
121 Files---done!
122 Files---done!
```

```
123 Files---done!
124 Files---done!
125 Files---done!
126 Files---done!
127 Files---done!
128 Files---done!
129 Files---done!
130 Files---done!
131 Files---done!
132 Files---done!
133 Files---done!
134 Files---done!
135 Files---done!
136 Files---done!
137 Files---done!
138 Files---done!
139 Files---done!
140 Files---done!
141 Files---done!
142 Files---done!
143 Files---done!
144 Files---done!
145 Files---done!
146 Files---done!
147 Files---done!
148 Files---done!
149 Files---done!
150 Files---done!
151 Files---done!
152 Files---done!
153 Files---done!
154 Files---done!
155 Files---done!
156 Files---done!
157 Files---done!
158 Files---done!
159 Files---done!
160 Files---done!
161 Files---done!
162 Files---done!
163 Files---done!
164 Files---done!
```

```
165 Files---done!
166 Files---done!
167 Files---done!
168 Files---done!
169 Files---done!
170 Files---done!
171 Files---done!
172 Files---done!
173 Files---done!
174 Files---done!
175 Files---done!
176 Files---done!
177 Files---done!
178 Files---done!
179 Files---done!
180 Files---done!
181 Files---done!
182 Files---done!
183 Files---done!
184 Files---done!
185 Files---done!
186 Files---done!
187 Files---done!
188 Files---done!
189 Files---done!
190 Files---done!
191 Files---done!
192 Files---done!
193 Files---done!
194 Files---done!
195 Files---done!
196 Files---done!
197 Files---done!
198 Files---done!
199 Files---done!
200 Files---done!
201 Files---done!
202 Files---done!
203 Files---done!
204 Files---done!
205 Files---done!
206 Files---done!
```

```
207 Files---done!
208 Files---done!

Time taken:  0:22:06.370669
```

In [130]:
```
sarimax_di_log
```

Out[130]:
```
{'APG ATLANTA_Oldcastle Retail CMP999820_EZ Mix.pkl': 8.96283726359432,
 'Canada_Expocrete_Acheson.pkl': 4.780319247573839,
 'Canada_Expocrete_Balzac.pkl': 3.4407645751395926,
 'Canada_Expocrete_Edmonton.pkl': 2.0069848095460996,
 'Canada_Expocrete_Richmond.pkl': 2.2510574146376503,
 'Canada_Expocrete_Saskatoon.pkl': 23.978136345240234,
 'Canada_Expocrete_Winnipeg.pkl': 2.771846968958864,
 'Canada_Permacon_Milton ON.pkl': 3.1143712156669685,
 'Canada_Permacon_Montreal QC.pkl': 5.987995378002994,
 'Canada_Permacon_Woodstock Ontario.pkl': 31.57087622881542,
 'Central_Ash Grove MPC_Fort Smith, AR.pkl': 1.7200142119421151,
 'Central_Ash Grove MPC_Fremont, NE.pkl': 2.1619898321908715,
 'Central_Ash Grove MPC_Harrisonville, MO.pkl': 2.145230315024334,
 'Central_Ash Grove MPC_Jackson, MS.pkl': 5.758110520341434,
 'Central_Ash Grove MPC_Memphis, TN.pkl': 2.142208238832877,
 'Central_Ash Grove MPC_Muskogee, OK.pkl': 1428502.6091180083,
 'Central_Ash Grove MPC_North Little Rock, AR.pkl': 1.7636632678711037,
 'Central_Ash Grove MPC_Oklahoma City, OK.pkl': 2.3337264272021265,
 'Central_Jewell_Austin TX (S).pkl': 2.4024752099118385,
 'Central_Jewell_Brittmoore.pkl': 1.4690198955625173,
 'Central_Jewell_Dallas TX (S).pkl': 1.6165941069070864,
 'Central_Jewell_Frisco TX (S).pkl': 1.6977089084460126,
 'Central_Jewell_Houston TX - West Hardy (M).pkl': 4.347201910258736,
 'Central_Jewell_Houston TX-N Garden.pkl': 1.119374819325691,
 'Central_Jewell_Hurst TX-SAK.pkl': 1.0970172250316084,
 'Central_Jewell_IBC TX-SAK.pkl': 1.1233755845612257,
 'Central_Jewell_Katy TX-SAK.pkl': 1.2783954764055896,
 'Central_Jewell_Keller TX (S).pkl': 3.1237529195635103,
 'Central_Jewell_Marble Falls (SAK).pkl': 9.840629019254923,
 'Central_Jewell_Rosenberg TX.pkl': 1.6877824011778002,
 'Central_Jewell_Waco TX.pkl': 1.8715323422050374,
 'Central_Northfield_Bridgeport MI.pkl': 12.838514778837592,
 'Central_Northfield_Cincinnati OH-SAK.pkl': 2.5654736893205836,
 'Central_Northfield_Forest View IL.pkl': 3.333865894660769,
 'Central_Northfield_Franklin Park IL-SAK.pkl': 1.8044690250927906,
 'Central_Northfield_Indianapolis IN.pkl': 2.1108654104162192,
 'Central_Northfield_Miller Materials KC Plant.pkl': 2.375126052464595,
 'Central_Northfield_Miller MaterialsBonner Springs.pkl': 2.364994207397047,
 'Central_Northfield_Morris IL.pkl': 1.8791494138406049,
 'Central_Northfield_Mundelein IL.pkl': 2.52853624686265,
 'Central_Northfield_Shakopee.pkl': 3.774556240407539,
```

```
'Central_Northfield_Sheffield OH.pkl': 2.9455682290521294,
'Central_Northfield_West Des Moines IA.pkl': 1.76222736544945,
'Central_Northfield_Wisconsin Distribution Yard.pkl': 5.026637171612148,
'East_Adams Products_Anderson SC.pkl': 2.4099302523058768,
'East_Adams Products_Asheville NC.pkl': 1.7484653189151471,
'East_Adams Products_Charlotte NC Plant.pkl': 1.5275797498703982,
'East_Adams Products_Charlotte NC.pkl': 1.6102193993292868,
'East_Adams Products_Clarksville TN.pkl': 2.5021750078035905,
'East_Adams Products_Colfax NC.pkl': 1.6163756588764184,
'East_Adams Products_Cowpens SC.pkl': 2.012271708431551,
'East_Adams Products_Dunn NC.pkl': 1.3268299896965687,
'East_Adams Products_Durham NC.pkl': 2.476633726826292,
'East_Adams Products_Fayetteville NC.pkl': 2.484978878583234,
'East_Adams Products_Franklin NC.pkl': 1.6380172082668696,
'East_Adams Products_Franklin TN-SAK.pkl': 1.5596145700950743,
'East_Adams Products_Goldsboro NC.pkl': 3.483098679340521,
'East_Adams Products_Greensboro NC.pkl': 1.7716797048850454,
'East_Adams Products_Greenville NC.pkl': 2.032057099367161,
'East_Adams Products_Greenville SC.pkl': 1.7302507257120676,
'East_Adams Products_Hickory NC.pkl': 1.9147514227889315,
'East_Adams Products_HollyHill SC.pkl': 3.156665007903027,
'East_Adams Products_Inman SC.pkl': 1.445467651322382,
'East_Adams Products_Jacksonville NC.pkl': 1.8067917419240858,
'East_Adams Products_Lilesville NC-SAK.pkl': 1.1621183982638061,
'East_Adams Products_Morehead NC.pkl': 2.1814449291817506,
'East_Adams Products_Morrisville NC.pkl': 1.144515380739426,
'East_Adams Products_Myrtle Beach SC.pkl': 1.6344985331069335,
'East_Adams Products_Nashville TN.pkl': 6.733539945824357,
'East_Adams Products_Rockwood TN.pkl': 1.7216049201703874,
'East_Adams Products_Rocky Mount NC.pkl': 1.8877476598566825,
'East_Adams Products_Stallings NC.pkl': 3.43707480319481,
'East_Adams Products_Wilmington NC.pkl': 2.141829587703977,
'East_Adams Products_Youngsville NC.pkl': 1.5249180695848426,
'East_Anchor_Anchor South Region.pkl': 12.118185922622166,
'East_Anchor_Batavia NY-SAK.pkl': 2.2634615398061277,
'East_Anchor_Brick NJ.pkl': 2.3750561121718725,
'East_Anchor_Bristol PA-SAK.pkl': 1.740160486955105,
'East_Anchor_Calverton NY-SAK.pkl': 1.526431727477066,
'East_Anchor_Canaan CT-SAK.pkl': 1.7042385043810762,
'East_Anchor_Cranston RI.pkl': 1.9577233115208845,
'East_Anchor_Crofton MD.pkl': 2.018957314975493,
'East_Anchor_East Petersburg PA.pkl': 1.7359735313599751,
```

'East_Anchor_Easton PA.pkl': 2.90451171597774,
'East_Anchor_Emigsville PA.pkl': 3.6789270976870942,
'East_Anchor_Farmingdale NJ.pkl': 2.6225277700465663,
'East_Anchor_Fishers NY.pkl': 2.250411049827676,
'East_Anchor_Fredonia PA-SAK.pkl': 1.9504286725240456,
'East_Anchor_Holbrook MA.pkl': 2.322498510925735,
'East_Anchor_Keene NH.pkl': 3.5693742306937963,
'East_Anchor_Lebanon NH.pkl': 2.0484114934655686,
'East_Anchor_Lyndhurst NJ.pkl': 4.12521756340611,
'East_Anchor_Milford VA-SAK.pkl': 1.4395243691124235,
'East_Anchor_Oxford MA-SAK.pkl': 1.961896612648129,
'East_Anchor_White Marsh MD-SAK.pkl': 1.2975448261797935,
'East_Anchor_Winchester VA.pkl': 1.9998864120232385,
'East_Georgia Masonry Supply_Cartersville GA BLOCK.pkl': 12.09063913535963,
'East_Georgia Masonry Supply_Conley GA-SAK.pkl': 1.1835414783151035,
'East_Georgia Masonry Supply_Florence AL.pkl': 2.1929006869647085,
'East_Georgia Masonry Supply_Jasper AL.pkl': 33.0756473887974,
'East_Georgia Masonry Supply_Jonesboro GA.pkl': 1.4721056452115162,
'East_Georgia Masonry Supply_Lawrenceville GA DIST.pkl': 1.4514417360059708,
'East_Georgia Masonry Supply_Lawrenceville GA MANF.pkl': 56.01818469566572,
'East_Georgia Masonry Supply_Macon GA.pkl': 1.5679352519727328,
'East_Georgia Masonry Supply_Montgomery AL.pkl': 7.994239974601809,
'East_Georgia Masonry Supply_Pelham AL.pkl': 2.9415115266223326,
'East_Georgia Masonry Supply_Tyrone GA.pkl': 1.8657147167337809,
'East_OldcastleCoastal_Auburndale FL-SAK.pkl': 1.1370160867292645,
'East_OldcastleCoastal_Defuniak.pkl': 20.80845071416081,
'East_OldcastleCoastal_Fort Myers FL.pkl': 1.8028990314332825,
'East_OldcastleCoastal_Fort Pierce FL.pkl': 1.7402464558278394,
'East_OldcastleCoastal_Gainesville FL.pkl': 2.8809792194582338,
'East_OldcastleCoastal_Gulfport MS.pkl': 3.5325530761332575,
'East_OldcastleCoastal_Haines City FL Hardscapes.pkl': 1.0249652896084576,
'East_OldcastleCoastal_Jacksonville FL-SAK.pkl': 1.329983848861221,
'East_OldcastleCoastal_Jacksonville FL.pkl': 1.3425075345659694,
'East_OldcastleCoastal_Lehigh Acres FL.pkl': 1.875982722989825,
'East_OldcastleCoastal_Longwood FL.pkl': 1.6742951908030606,
'East_OldcastleCoastal_Orlando FL.pkl': 0.9178946332401242,
'East_OldcastleCoastal_Pensacola FL-SAK.pkl': 1.6811230159720507,
'East_OldcastleCoastal_Pompano FL Hardscapes.pkl': 1.3518633375080726,
'East_OldcastleCoastal_Pompano FL-SAK.pkl': 0.889967962385461,
'East_OldcastleCoastal_Sarasota FL.pkl': 1.12169778898559,
'East_OldcastleCoastal_Tampa FL - Anderson Rd.pkl': 1.1127140759599328,
'East_OldcastleCoastal_Tampa FL - Busch Blvd.pkl': 1.4957146712762897,

```
'East_OldcastleCoastal_Tampa FL-SAK.pkl': 1.2028954252737134,
'East_OldcastleCoastal_Theodore AL.pkl': 3.3113983446729724,
'East_OldcastleCoastal_West Palm Beach FL.pkl': 0.87039908092292,
'East_OldcastleCoastal_Zephyrhills FL.pkl': 1.3301865798876182,
'Lawn and Garden_L&G Central_Aliceville AL.pkl': 12.466553469596974,
'Lawn and Garden_L&G Central_Amherst Junction WI.pkl': 4.788420469104281,
'Lawn and Garden_L&G Central_Bridgeport MI.pkl': 14.935567239463673,
'Lawn and Garden_L&G Central_Cleveland, TX.pkl': 3.158422129842594,
'Lawn and Garden_L&G Central_Dallas TX.pkl': 2.440578993948743,
'Lawn and Garden_L&G Central_Del Valle, TX.pkl': 3.9043363894103913,
'Lawn and Garden_L&G Central_Harrah OK.pkl': 4.621391758828142,
'Lawn and Garden_L&G Central_Hope AR.pkl': 3.3752320538041043,
'Lawn and Garden_L&G Central_Livingston TX.pkl': 1.958513737043142,
'Lawn and Garden_L&G Central_Marseilles IL.pkl': 3.0331430616909,
'Lawn and Garden_L&G Central_Miami OK.pkl': 4.828655418263714,
'Lawn and Garden_L&G Central_Paola KS.pkl': 4.683758679781783,
'Lawn and Garden_L&G Central_Powderly TX.pkl': 3.721468206595412,
'Lawn and Garden_L&G Central_Sauget IL.pkl': 3.2171214460909723,
'Lawn and Garden_L&G Central_Tulia TX.pkl': 7.655064280557737,
'Lawn and Garden_L&G Central_Tylertown MS.pkl': 2.589973563848342,
'Lawn and Garden_L&G Central_Waterloo IN.pkl': 6.716018393257836,
'Lawn and Garden_L&G Northeast_Berlin NY.pkl': 8.620563418283233,
'Lawn and Garden_L&G Northeast_Carey OH.pkl': 5.394179973072117,
'Lawn and Garden_L&G Northeast_Castlewood VA.pkl': 2.5014954147456505,
'Lawn and Garden_L&G Northeast_Chatsworth GA.pkl': 3.0471306730560372,
'Lawn and Garden_L&G Northeast_Historical-Co-Packer.pkl': 87.34732014216664,
'Lawn and Garden_L&G Northeast_Hooksett NH.pkl': 4.798211834761299,
'Lawn and Garden_L&G Northeast_Lee MA.pkl': 2.409644272715557,
'Lawn and Garden_L&G Northeast_Manchester NY.pkl': 16.473710149995174,
'Lawn and Garden_L&G Northeast_Mount Hope NJ.pkl': 4.321626097870144,
'Lawn and Garden_L&G Northeast_Poland Spring ME.pkl': 5.320320415299232,
'Lawn and Garden_L&G Northeast_Quakertown PA.pkl': 4.800649753317508,
'Lawn and Garden_L&G Northeast_Thomasville PA.pkl': 2.0536036872658983,
'Lawn and Garden_L&G Northeast_Wyoming RI.pkl': 5.651111943365099,
'Lawn and Garden_L&G Southeast_Aberdeen NC.pkl': 2.6766107910362456,
'Lawn and Garden_L&G Southeast_Bostwick FL.pkl': 3.0293355195787917,
'Lawn and Garden_L&G Southeast_Cross City FL.pkl': 2.221612086000611,
'Lawn and Garden_L&G Southeast_Davenport FL.pkl': 2.821098984158969,
'Lawn and Garden_L&G Southeast_Fort Green FL.pkl': 4.215655373630957,
'Lawn and Garden_L&G Southeast_Gaffney SC.pkl': 3.476275284097664,
'Lawn and Garden_L&G Southeast_Louisburg NC.pkl': 3.022017310652945,
'Lawn and Garden_L&G Southeast_Moore Haven FL.pkl': 2.1039789718999318,
```

```
'Lawn and Garden_L&G Southeast_Pageland SC.pkl': 2.199579768094937,
'Lawn and Garden_L&G Southeast_Shady Dale GA.pkl': 2.5802948056053547,
'Lawn and Garden_L&G Southeast_Walterboro SC.pkl': 2.7899658836484806,
'National_AMTC_Saint Paul MN.pkl': 4.395712770902298,
'National_Anchor Wall Systems_Minnetonka MN.pkl': 2.2286262274647193,
'National_MoistureShield_Lowell AR.pkl': 9.623386424489015,
'National_MoistureShield_Springdale AR.pkl': 2.939848407229587,
'National_Oldcastle Sakerete Billing_Easy Mix.pkl': 4.977366769776699,
'National_Oldcastle Sakerete Billing_Roberts Concrete.pkl': 6.347222071147277,
'National_Oldcastle Sakerete Billing_US Mix.pkl': 1.7879589594556002,
'National_Techniseal_Ash Grove Memphis.pkl': 7.906050184268601,
'National_Techniseal_Ash Grove Nebraska.pkl': 5.9530311668236555,
'National_Techniseal_Candiac.pkl': 4.342724077636547,
'National_Techniseal_Coastal Tampa.pkl': 14.27102205862256,
'National_Techniseal_CPM Portland.pkl': 11.566425514139379,
'National_Techniseal_Ectra.pkl': 20.596614747452517,
'National_Techniseal_Eurl Leps Mehat.pkl': 13.75303833976532,
'National_Techniseal_EZ Mix.pkl': 5.948904482091848,
'National_Techniseal_Handy Concrete.pkl': 7.197153969239752,
'National_Techniseal_PTB  Compaktuna.pkl': 7.299917161595297,
'National_Techniseal_Ras.pkl': 8.248179271272397,
'National_Techniseal_Techmix.pkl': 3.4248718455920106,
'National_Westile_Westile Roofing Products.pkl': 1.5614617637754675,
'West_Amcor_North Salt Lake UT.pkl': 2.5473586840633002,
'West_CPM_Frederickson, WA.pkl': 5.045699469430224,
'West_CPM_Kent, WA.pkl': 1.4465102559961787,
'West_CPM_Northstar Consignment.pkl': 12.317972856708549,
'West_CPM_Pasco WA.pkl': 3.7220832605999883,
'West_CPM_Portland, OR.pkl': 1.3589922981922657,
'West_CPM_Spokane, WA.pkl': 2.023427876671506,
'West_Sierra_Fontana CA.pkl': 1.5775000700835775,
'West_Sierra_Reno NV.pkl': 2.0819223862516245,
'West_Sierra_San Carlos CA.pkl': 6.1647164907965335,
'West_Sierra_Stockton CA.pkl': 1.2164440508316785,
'West_Superlite_Gilbert, AZ.pkl': 1.8337850220912197,
'West_Superlite_Integra Product.pkl': 5.565757333247085,
'West_Superlite_Lone Butte.pkl': 1.6445992780422056,
'West_Superlite_North Las Vegas.pkl': 1.647633630484574,
'West_Superlite_Superlite - Western 19th Ave.pkl': 1.9484778458068401,
'West_Superlite_Tucson, AZ - Gardner Ln.pkl': 1.3314505403197452,
'West_Superlite_West Phoenix N. 42nd Ave, AZ.pkl': 0.9529261214407598}
```

Holt-Winters Model

In [125]:
```
def hw_on_all(names, paths):
    '''
        Running Holt winters model/triple exponential on all combinations
    '''
    print(hw_on_all.__doc__)
    hw_di = {} # dictionary of holt winters model, contains name and mape score
    for i in range(len(names)):
#         print(names[i])
        df_hw = pd.read_pickle(paths[i])
        df_hw.dropna(inplace=True)
        train = df_hw[:'2018-11-01']
        test = df_hw['2018-11-01':]
        start = len(train)
        end = (len(train) + len(test)) - 1

        model_hw = ExponentialSmoothing(train.Actual_Sales, seasonal_periods=4,
                                        trend='add', seasonal='add')
        results_hw = model_hw.fit()
        y_pred_hw = results_hw.predict(start=start, end=end)

        y_true = test.Actual_Sales
        mape_val = mape(y_true, y_pred_hw)
        hw_di[names[i]] = mape_val

    return hw_di
```

In [126]:
```
start = datetime.now()
holt_win_di_log = hw_on_all(good_names_log, good_paths_log)
print('\nTime taken: ', datetime.now() - start)
```

```
        Running Holt winters model/triple exponential on all combinations


Time taken:  0:00:35.446975
```

In [121]:
```
holt_win_di_log
```

Out[121]:
```
{'APG ATLANTA_Oldcastle Retail CMP999820_EZ Mix.pkl': 7.569700933414738,
 'Canada_Expocrete_Acheson.pkl': 10.417335694819618,
 'Canada_Expocrete_Balzac.pkl': 11.643074344536618,
 'Canada_Expocrete_Edmonton.pkl': 2.01343883801723,
 'Canada_Expocrete_Richmond.pkl': 4.0305308906600965,
 'Canada_Expocrete_Saskatoon.pkl': 17.582360166470664,
 'Canada_Expocrete_Winnipeg.pkl': 4.290771819256081,
 'Canada_Permacon_Milton ON.pkl': 4.750073472409732,
 'Canada_Permacon_Montreal QC.pkl': 7.490347684939843,
 'Canada_Permacon_Woodstock Ontario.pkl': 61.79481811558352,
 'Central_Ash Grove MPC_Fort Smith, AR.pkl': 1.7712972388519235,
 'Central_Ash Grove MPC_Fremont, NE.pkl': 3.2600634442219847,
 'Central_Ash Grove MPC_Harrisonville, MO.pkl': 2.555598298705693,
 'Central_Ash Grove MPC_Jackson, MS.pkl': 4.409853427680009,
 'Central_Ash Grove MPC_Memphis, TN.pkl': 3.379470013916531,
 'Central_Ash Grove MPC_Muskogee, OK.pkl': 3.2520415153195046,
 'Central_Ash Grove MPC_North Little Rock, AR.pkl': 2.1028480304674484,
 'Central_Ash Grove MPC_Oklahoma City, OK.pkl': 2.1796319886108995,
 'Central_Jewell_Austin TX (S).pkl': 3.03558786453521,
 'Central_Jewell_Brittmoore.pkl': 1.7627846830856848,
 'Central_Jewell_Dallas TX (S).pkl': 1.4942922732088524,
 'Central_Jewell_Frisco TX (S).pkl': 1.5244961632034038,
 'Central_Jewell_Houston TX - West Hardy (M).pkl': 1.9374792475285956,
 'Central_Jewell_Houston TX-N Garden.pkl': 1.841098515196849,
 'Central_Jewell_Hurst TX-SAK.pkl': 1.0934353971450328,
 'Central_Jewell_IBC TX-SAK.pkl': 1.8412633951039683,
 'Central_Jewell_Katy TX-SAK.pkl': 1.4651095003575625,
 'Central_Jewell_Keller TX (S).pkl': 2.755405657626958,
 'Central_Jewell_Marble Falls (SAK).pkl': 1.7430768036880449,
 'Central_Jewell_Rosenberg TX.pkl': 7.075726461970786,
 'Central_Jewell_Waco TX.pkl': 4.743303718957051,
 'Central_Northfield_Bridgeport MI.pkl': 13.437006425989416,
 'Central_Northfield_Cincinnati OH-SAK.pkl': 4.488478133671128,
 'Central_Northfield_Forest View IL.pkl': 4.380139393755977,
 'Central_Northfield_Franklin Park IL-SAK.pkl': 4.158346432775368,
 'Central_Northfield_Indianapolis IN.pkl': 5.272019069246607,
 'Central_Northfield_Miller Materials KC Plant.pkl': 3.1468951870738717,
 'Central_Northfield_Miller MaterialsBonner Springs.pkl': 7.054505190154632,
 'Central_Northfield_Morris IL.pkl': 2.063633215499743,
 'Central_Northfield_Mundelein IL.pkl': 3.3270467678672957,
 'Central_Northfield_Shakopee.pkl': 5.649269697337009,
```

```
'Central_Northfield_Sheffield OH.pkl': 7.040342681083025,
'Central_Northfield_West Des Moines IA.pkl': 3.6669501389123433,
'Central_Northfield_Wisconsin Distribution Yard.pkl': 7.171793480042514,
'East_Adams Products_Anderson SC.pkl': 2.37130578687455,
'East_Adams Products_Asheville NC.pkl': 1.8283176401750392,
'East_Adams Products_Charlotte NC Plant.pkl': 1.3129057306717744,
'East_Adams Products_Charlotte NC.pkl': 1.5307780946680247,
'East_Adams Products_Clarksville TN.pkl': 4.600408407215135,
'East_Adams Products_Colfax NC.pkl': 1.6888477185840793,
'East_Adams Products_Cowpens SC.pkl': 4.517406777461142,
'East_Adams Products_Dunn NC.pkl': 1.3489234772747594,
'East_Adams Products_Durham NC.pkl': 2.270447393674892,
'East_Adams Products_Fayetteville NC.pkl': 2.1720839553934104,
'East_Adams Products_Franklin NC.pkl': 2.0270765713481853,
'East_Adams Products_Franklin TN-SAK.pkl': 2.361338544227272,
'East_Adams Products_Goldsboro NC.pkl': 2.9580383630242655,
'East_Adams Products_Greensboro NC.pkl': 4.949171552621415,
'East_Adams Products_Greenville NC.pkl': 1.9107543567407916,
'East_Adams Products_Greenville SC.pkl': 2.2545654811741906,
'East_Adams Products_Hickory NC.pkl': 1.869105904298283,
'East_Adams Products_HollyHill SC.pkl': 2.5328176459551583,
'East_Adams Products_Inman SC.pkl': 1.6497061959350727,
'East_Adams Products_Jacksonville NC.pkl': 1.5551525554287609,
'East_Adams Products_Lilesville NC-SAK.pkl': 1.6710617015871967,
'East_Adams Products_Morehead NC.pkl': 1.9680980452980044,
'East_Adams Products_Morrisville NC.pkl': 1.328643250124349,
'East_Adams Products_Myrtle Beach SC.pkl': 1.6719008096884596,
'East_Adams Products_Nashville TN.pkl': 3.7584217433837366,
'East_Adams Products_Rockwood TN.pkl': 4.365753236289578,
'East_Adams Products_Rocky Mount NC.pkl': 2.438560141362724,
'East_Adams Products_Stallings NC.pkl': 5.7150531070111175,
'East_Adams Products_Wilmington NC.pkl': 2.190817422200473,
'East_Adams Products_Youngsville NC.pkl': 8.387649970926068,
'East_Anchor_Anchor South Region.pkl': 10.627055963255403,
'East_Anchor_Batavia NY-SAK.pkl': 4.880268987904104,
'East_Anchor_Brick NJ.pkl': 2.8992771212107358,
'East_Anchor_Bristol PA-SAK.pkl': 2.535455189123602,
'East_Anchor_Calverton NY-SAK.pkl': 3.4892126257938654,
'East_Anchor_Canaan CT-SAK.pkl': 4.018033490344178,
'East_Anchor_Cranston RI.pkl': 6.319237567854417,
'East_Anchor_Crofton MD.pkl': 4.040789560280717,
'East_Anchor_East Petersburg PA.pkl': 2.1434876738183872,
```

```
'East_Anchor_Easton PA.pkl': 5.4753852955030355,
'East_Anchor_Emigsville PA.pkl': 1.9516969684221261,
'East_Anchor_Farmingdale NJ.pkl': 4.628576938724663,
'East_Anchor_Fishers NY.pkl': 5.668879018927879,
'East_Anchor_Fredonia PA-SAK.pkl': 5.731031000457236,
'East_Anchor_Holbrook MA.pkl': 12.267787497189143,
'East_Anchor_Keene NH.pkl': 6.906232963403207,
'East_Anchor_Lebanon NH.pkl': 4.682695522989181,
'East_Anchor_Lyndhurst NJ.pkl': 4.745937498656871,
'East_Anchor_Milford VA-SAK.pkl': 3.1046163984324417,
'East_Anchor_Oxford MA-SAK.pkl': 3.4134871351093983,
'East_Anchor_White Marsh MD-SAK.pkl': 1.8242782350492588,
'East_Anchor_Winchester VA.pkl': 3.3786428385942284,
'East_Georgia Masonry Supply_Cartersville GA BLOCK.pkl': 1.6632653478895272,
'East_Georgia Masonry Supply_Conley GA-SAK.pkl': 2.9006046494886273,
'East_Georgia Masonry Supply_Florence AL.pkl': 4.847248876984937,
'East_Georgia Masonry Supply_Jasper AL.pkl': 12.744742498393794,
'East_Georgia Masonry Supply_Jonesboro GA.pkl': 4.093016299364296,
'East_Georgia Masonry Supply_Lawrenceville GA DIST.pkl': 1.9459911738776932,
'East_Georgia Masonry Supply_Lawrenceville GA MANF.pkl': 1.9557955824552165,
'East_Georgia Masonry Supply_Macon GA.pkl': 1.6016106557044003,
'East_Georgia Masonry Supply_Montgomery AL.pkl': 2.5975466569648225,
'East_Georgia Masonry Supply_Pelham AL.pkl': 3.9874691297823914,
'East_Georgia Masonry Supply_Tyrone GA.pkl': 2.1867766025915794,
'East_OldcastleCoastal_Auburndale FL-SAK.pkl': 1.1325562087456933,
'East_OldcastleCoastal_Defuniak.pkl': 1.4722701636516666,
'East_OldcastleCoastal_Fort Myers FL.pkl': 2.160975375142724,
'East_OldcastleCoastal_Fort Pierce FL.pkl': 2.073992108995173,
'East_OldcastleCoastal_Gainesville FL.pkl': 2.9390819873451197,
'East_OldcastleCoastal_Gulfport MS.pkl': 2.514954511572039,
'East_OldcastleCoastal_Haines City FL Hardscapes.pkl': 0.8986031602249684,
'East_OldcastleCoastal_Jacksonville FL-SAK.pkl': 1.4358086104227996,
'East_OldcastleCoastal_Jacksonville FL.pkl': 2.121754347064379,
'East_OldcastleCoastal_Lehigh Acres FL.pkl': 1.8807363276478386,
'East_OldcastleCoastal_Longwood FL.pkl': 1.583274098424035,
'East_OldcastleCoastal_Orlando FL.pkl': 1.2777448266895957,
'East_OldcastleCoastal_Pensacola FL-SAK.pkl': 1.9900001715960132,
'East_OldcastleCoastal_Pompano FL Hardscapes.pkl': 1.7722776166817202,
'East_OldcastleCoastal_Pompano FL-SAK.pkl': 0.8977985761411008,
'East_OldcastleCoastal_Sarasota FL.pkl': 1.1759395515018332,
'East_OldcastleCoastal_Tampa FL - Anderson Rd.pkl': 1.1431075127925443,
'East_OldcastleCoastal_Tampa FL - Busch Blvd.pkl': 1.5722917011265454,
```

'East_OldcastleCoastal_Tampa FL-SAK.pkl': 1.0574612892254291,
'East_OldcastleCoastal_Theodore AL.pkl': 2.53978653956751,
'East_OldcastleCoastal_West Palm Beach FL.pkl': 0.8398642010953772,
'East_OldcastleCoastal_Zephyrhills FL.pkl': 1.9532258789811439,
'Lawn and Garden_L&G Central_Aliceville AL.pkl': 8.753490270533995,
'Lawn and Garden_L&G Central_Amherst Junction WI.pkl': 10.390819942965477,
'Lawn and Garden_L&G Central_Bridgeport MI.pkl': 10.168362116874745,
'Lawn and Garden_L&G Central_Cleveland, TX.pkl': 5.515334415921876,
'Lawn and Garden_L&G Central_Dallas TX.pkl': 6.937480409825722,
'Lawn and Garden_L&G Central_Del Valle, TX.pkl': 6.1224867883067065,
'Lawn and Garden_L&G Central_Harrah OK.pkl': 10.43788523903745,
'Lawn and Garden_L&G Central_Hope AR.pkl': 7.916963909621017,
'Lawn and Garden_L&G Central_Livingston TX.pkl': 4.961961185292129,
'Lawn and Garden_L&G Central_Marseilles IL.pkl': 4.492234490676647,
'Lawn and Garden_L&G Central_Miami OK.pkl': 7.174095151053329,
'Lawn and Garden_L&G Central_Paola KS.pkl': 11.827378737570033,
'Lawn and Garden_L&G Central_Powderly TX.pkl': 6.760614158824893,
'Lawn and Garden_L&G Central_Sauget IL.pkl': 7.135480146750141,
'Lawn and Garden_L&G Central_Tulia TX.pkl': 10.890861995769438,
'Lawn and Garden_L&G Central_Tylertown MS.pkl': 7.128151417693653,
'Lawn and Garden_L&G Central_Waterloo IN.pkl': 11.004742041874477,
'Lawn and Garden_L&G Northeast_Berlin NY.pkl': 34.9090288909396,
'Lawn and Garden_L&G Northeast_Carey OH.pkl': 4.590105048817018,
'Lawn and Garden_L&G Northeast_Castlewood VA.pkl': 3.2492066958522408,
'Lawn and Garden_L&G Northeast_Chatsworth GA.pkl': 4.726957514884648,
'Lawn and Garden_L&G Northeast_Historical-Co-Packer.pkl': 33.808728345573705,
'Lawn and Garden_L&G Northeast_Hooksett NH.pkl': 15.449942380741799,
'Lawn and Garden_L&G Northeast_Lee MA.pkl': 3.094451692859155,
'Lawn and Garden_L&G Northeast_Manchester NY.pkl': 14.859468727082008,
'Lawn and Garden_L&G Northeast_Mount Hope NJ.pkl': 10.20725298769233,
'Lawn and Garden_L&G Northeast_Poland Spring ME.pkl': 7.455291145336254,
'Lawn and Garden_L&G Northeast_Quakertown PA.pkl': 9.402383873310184,
'Lawn and Garden_L&G Northeast_Thomasville PA.pkl': 4.5621290869704305,
'Lawn and Garden_L&G Northeast_Wyoming RI.pkl': 15.127404146769877,
'Lawn and Garden_L&G Southeast_Aberdeen NC.pkl': 7.244119122217405,
'Lawn and Garden_L&G Southeast_Bostwick FL.pkl': 3.684336390081061,
'Lawn and Garden_L&G Southeast_Cross City FL.pkl': 3.7181236257311574,
'Lawn and Garden_L&G Southeast_Davenport FL.pkl': 2.26618778904053,
'Lawn and Garden_L&G Southeast_Fort Green FL.pkl': 5.119311103698004,
'Lawn and Garden_L&G Southeast_Gaffney SC.pkl': 9.859660826237073,
'Lawn and Garden_L&G Southeast_Louisburg NC.pkl': 9.34129488640508,
'Lawn and Garden_L&G Southeast_Moore Haven FL.pkl': 2.4023844395709393,

```
'Lawn and Garden_L&G Southeast_Pageland SC.pkl': 6.311932681224743,
'Lawn and Garden_L&G Southeast_Shady Dale GA.pkl': 6.903741333429929,
'Lawn and Garden_L&G Southeast_Walterboro SC.pkl': 4.487511809989975,
'National_AMTC_Saint Paul MN.pkl': 6.414574394001398,
'National_Anchor Wall Systems_Minnetonka MN.pkl': 3.1435243216281226,
'National_MoistureShield_Lowell AR.pkl': 11.18786606939449,
'National_MoistureShield_Springdale AR.pkl': 1.6958945034402368,
'National_Oldcastle Sakerete Billing_Easy Mix.pkl': 6.485183551534218,
'National_Oldcastle Sakerete Billing_Roberts Concrete.pkl': 7.08897731323762,
'National_Oldcastle Sakerete Billing_US Mix.pkl': 3.5303271830194745,
'National_Techniseal_Ash Grove Memphis.pkl': 9.1896786967675,
'National_Techniseal_Ash Grove Nebraska.pkl': 7.260311244991982,
'National_Techniseal_Candiac.pkl': 7.476254973618689,
'National_Techniseal_Coastal Tampa.pkl': 11.547826651905341,
'National_Techniseal_CPM Portland.pkl': 12.968872523840355,
'National_Techniseal_Ectra.pkl': 21.7567403821444,
'National_Techniseal_Eurl Leps Mehat.pkl': 12.526520782787173,
'National_Techniseal_EZ Mix.pkl': 5.352325034693823,
'National_Techniseal_Handy Concrete.pkl': 6.644441738761421,
'National_Techniseal_PTB  Compaktuna.pkl': 6.423383676002453,
'National_Techniseal_Ras.pkl': 12.7178412616186,
'National_Techniseal_Techmix.pkl': 4.410453302883164,
'National_Westile_Westile Roofing Products.pkl': 2.4908408619524427,
'West_Amcor_North Salt Lake UT.pkl': 3.6952936399143566,
'West_CPM_Frederickson, WA.pkl': 4.1144604354368886,
'West_CPM_Kent, WA.pkl': 2.5149164416706156,
'West_CPM_Northstar Consignment.pkl': 9.643984651740567,
'West_CPM_Pasco WA.pkl': 4.413578119128289,
'West_CPM_Portland, OR.pkl': 2.0109214896073606,
'West_CPM_Spokane, WA.pkl': 3.3323889236498636,
'West_Sierra_Fontana CA.pkl': 1.3884545708867593,
'West_Sierra_Reno NV.pkl': 2.920226452995267,
'West_Sierra_San Carlos CA.pkl': 7.327622372347285,
'West_Sierra_Stockton CA.pkl': 1.7416726655054295,
'West_Superlite_Gilbert, AZ.pkl': 1.9633409348424256,
'West_Superlite_Integra Product.pkl': 4.915487940254884,
'West_Superlite_Lone Butte.pkl': 1.778490176808602,
'West_Superlite_North Las Vegas.pkl': 1.8971961878985748,
'West_Superlite_Superlite - Western 19th Ave.pkl': 1.4595649039429057,
'West_Superlite_Tucson, AZ - Gardner Ln.pkl': 1.2345407775132053,
'West_Superlite_West Phoenix N. 42nd Ave, AZ.pkl': 0.9413365631006284}
```

Exponential Weighted Avg

In [43]:
```
def se_on_all(names, paths):
    '''
        Running exponential weighted avg/Simple Exponential on all files
    '''
    print(se_on_all.__doc__)
    se_di = {} # dcitionary of Simple Exponential Model, contains names and mape scores
    for i in range(len(names)):
        df_se = pd.read_pickle(paths[i])
        df_se.dropna(inplace=True)
        train = df_se[:'2018-11-01']
        test = df_se['2018-11-01':]
        start = len(train)
        end = (len(train) + len(test)) - 1

        model_se = SimpleExpSmoothing(train.Actual_Sales)
        results_se = model_se.fit()

        y_pred_se = results_se.predict(start=start, end=end)

        y_true = test.Actual_Sales
        mape_val = mape(y_true, y_pred_se)
        se_di[names[i]] = mape_val

    return se_di
```

In [44]:
```
start = datetime.now()
simple_avg_di_log = se_on_all(good_names_log, good_paths_log)
print('\nTime taken: ', datetime.now() - start)
```

```
        Running exponential weighted avg/Simple Exponential on all files


Time taken:  0:00:03.273342
```

In [45]:
```
simple_avg_di_log
```

Out[45]:
```
{'APG ATLANTA_Oldcastle Retail CMP999820_EZ Mix.pkl': 5.314153841659752,
 'Canada_Expocrete_Acheson.pkl': 10.309094091998833,
 'Canada_Expocrete_Balzac.pkl': 6.056849820659617,
 'Canada_Expocrete_Edmonton.pkl': 1.991395027713599,
 'Canada_Expocrete_Richmond.pkl': 4.262636595196109,
 'Canada_Expocrete_Saskatoon.pkl': 14.16207765915776,
 'Canada_Expocrete_Winnipeg.pkl': 3.0519994481363515,
 'Canada_Permacon_Milton ON.pkl': 5.104273005024259,
 'Canada_Permacon_Montreal QC.pkl': 9.247445456355791,
 'Canada_Permacon_Woodstock Ontario.pkl': 54.18431101795753,
 'Central_Ash Grove MPC_Fort Smith, AR.pkl': 1.725902619068026,
 'Central_Ash Grove MPC_Fremont, NE.pkl': 3.319843233705012,
 'Central_Ash Grove MPC_Harrisonville, MO.pkl': 2.7682056972564135,
 'Central_Ash Grove MPC_Jackson, MS.pkl': 3.6403674669256945,
 'Central_Ash Grove MPC_Memphis, TN.pkl': 2.5834140878747083,
 'Central_Ash Grove MPC_Muskogee, OK.pkl': 3.0170361254325018,
 'Central_Ash Grove MPC_North Little Rock, AR.pkl': 1.997168492100671,
 'Central_Ash Grove MPC_Oklahoma City, OK.pkl': 2.2570817006541883,
 'Central_Jewell_Austin TX (S).pkl': 2.271974899746169,
 'Central_Jewell_Brittmoore.pkl': 1.6232686581619602,
 'Central_Jewell_Dallas TX (S).pkl': 1.3194466999526306,
 'Central_Jewell_Frisco TX (S).pkl': 1.6786760590303211,
 'Central_Jewell_Houston TX - West Hardy (M).pkl': 2.134202890883004,
 'Central_Jewell_Houston TX-N Garden.pkl': 1.1189297595001129,
 'Central_Jewell_Hurst TX-SAK.pkl': 1.2273927860137441,
 'Central_Jewell_IBC TX-SAK.pkl': 1.6457726101390684,
 'Central_Jewell_Katy TX-SAK.pkl': 1.3727863639010331,
 'Central_Jewell_Keller TX (S).pkl': 2.9018750519661682,
 'Central_Jewell_Marble Falls (SAK).pkl': 1.3737712779143458,
 'Central_Jewell_Rosenberg TX.pkl': 5.260692902778648,
 'Central_Jewell_Waco TX.pkl': 3.958013660315841,
 'Central_Northfield_Bridgeport MI.pkl': 8.81175009138693,
 'Central_Northfield_Cincinnati OH-SAK.pkl': 5.016930342565777,
 'Central_Northfield_Forest View IL.pkl': 2.9781433346719446,
 'Central_Northfield_Franklin Park IL-SAK.pkl': 2.6252390515774646,
 'Central_Northfield_Indianapolis IN.pkl': 5.567748364677107,
 'Central_Northfield_Miller Materials KC Plant.pkl': 2.8696051520569683,
 'Central_Northfield_Miller MaterialsBonner Springs.pkl': 6.073984590537707,
 'Central_Northfield_Morris IL.pkl': 2.3043394306625364,
 'Central_Northfield_Mundelein IL.pkl': 3.633595631495158,
 'Central_Northfield_Shakopee.pkl': 5.847210300276865,
```

'Central_Northfield_Sheffield OH.pkl': 7.883884527690789,
'Central_Northfield_West Des Moines IA.pkl': 3.8588450138043324,
'Central_Northfield_Wisconsin Distribution Yard.pkl': 6.929730684520109,
'East_Adams Products_Anderson SC.pkl': 2.404075520195797,
'East_Adams Products_Asheville NC.pkl': 1.9803617189587488,
'East_Adams Products_Charlotte NC Plant.pkl': 1.4001523509071192,
'East_Adams Products_Charlotte NC.pkl': 1.5106347975759606,
'East_Adams Products_Clarksville TN.pkl': 2.151528955516315,
'East_Adams Products_Colfax NC.pkl': 1.9286959230388614,
'East_Adams Products_Cowpens SC.pkl': 3.8291987176991995,
'East_Adams Products_Dunn NC.pkl': 1.2981720655395443,
'East_Adams Products_Durham NC.pkl': 2.496763463219302,
'East_Adams Products_Fayetteville NC.pkl': 2.2592385636137724,
'East_Adams Products_Franklin NC.pkl': 2.450961591108351,
'East_Adams Products_Franklin TN-SAK.pkl': 2.688397853878146,
'East_Adams Products_Goldsboro NC.pkl': 2.8366480485695256,
'East_Adams Products_Greensboro NC.pkl': 4.3060826615200805,
'East_Adams Products_Greenville NC.pkl': 2.0698232464691677,
'East_Adams Products_Greenville SC.pkl': 2.0010745259795732,
'East_Adams Products_Hickory NC.pkl': 2.138448323531481,
'East_Adams Products_HollyHill SC.pkl': 1.8725214272494781,
'East_Adams Products_Inman SC.pkl': 1.4294861346718903,
'East_Adams Products_Jacksonville NC.pkl': 1.6078590037345675,
'East_Adams Products_Lilesville NC-SAK.pkl': 2.027046468268612,
'East_Adams Products_Morehead NC.pkl': 2.1434853142963015,
'East_Adams Products_Morrisville NC.pkl': 1.465242981475816,
'East_Adams Products_Myrtle Beach SC.pkl': 1.6451772589051727,
'East_Adams Products_Nashville TN.pkl': 4.938543822690803,
'East_Adams Products_Rockwood TN.pkl': 4.269983009520037,
'East_Adams Products_Rocky Mount NC.pkl': 2.3689534673968535,
'East_Adams Products_Stallings NC.pkl': 3.3389383572697917,
'East_Adams Products_Wilmington NC.pkl': 2.127484306623306,
'East_Adams Products_Youngsville NC.pkl': 2.6750671189976702,
'East_Anchor_Anchor South Region.pkl': 10.566363383559315,
'East_Anchor_Batavia NY-SAK.pkl': 4.262071875279329,
'East_Anchor_Brick NJ.pkl': 2.1191627888122255,
'East_Anchor_Bristol PA-SAK.pkl': 2.463566783057888,
'East_Anchor_Calverton NY-SAK.pkl': 2.882436014426028,
'East_Anchor_Canaan CT-SAK.pkl': 3.2366088743633328,
'East_Anchor_Cranston RI.pkl': 3.993232200680699,
'East_Anchor_Crofton MD.pkl': 4.686986853259641,
'East_Anchor_East Petersburg PA.pkl': 2.2448028512477944,

```
'East_Anchor_Easton PA.pkl': 5.568131241473304,
'East_Anchor_Emigsville PA.pkl': 2.1399096397493027,
'East_Anchor_Farmingdale NJ.pkl': 5.064793166005751,
'East_Anchor_Fishers NY.pkl': 5.332250684169275,
'East_Anchor_Fredonia PA-SAK.pkl': 3.9404913785486384,
'East_Anchor_Holbrook MA.pkl': 6.482598907816336,
'East_Anchor_Keene NH.pkl': 6.209489679963351,
'East_Anchor_Lebanon NH.pkl': 4.932427089289941,
'East_Anchor_Lyndhurst NJ.pkl': 3.6034294175061916,
'East_Anchor_Milford VA-SAK.pkl': 2.5277133859594443,
'East_Anchor_Oxford MA-SAK.pkl': 3.1180598276762987,
'East_Anchor_White Marsh MD-SAK.pkl': 2.0807001059712045,
'East_Anchor_Winchester VA.pkl': 3.277412528236707,
'East_Georgia Masonry Supply_Cartersville GA BLOCK.pkl': 1.7067345324117762,
'East_Georgia Masonry Supply_Conley GA-SAK.pkl': 2.7429101222696533,
'East_Georgia Masonry Supply_Florence AL.pkl': 2.4072822945103862,
'East_Georgia Masonry Supply_Jasper AL.pkl': 16.535419051051807,
'East_Georgia Masonry Supply_Jonesboro GA.pkl': 3.9618285056833353,
'East_Georgia Masonry Supply_Lawrenceville GA DIST.pkl': 1.8942127690531534,
'East_Georgia Masonry Supply_Lawrenceville GA MANF.pkl': 1.9218196171019577,
'East_Georgia Masonry Supply_Macon GA.pkl': 1.5929725820638119,
'East_Georgia Masonry Supply_Montgomery AL.pkl': 2.364615618216656,
'East_Georgia Masonry Supply_Pelham AL.pkl': 2.1756783021046355,
'East_Georgia Masonry Supply_Tyrone GA.pkl': 1.8396925335020071,
'East_OldcastleCoastal_Auburndale FL-SAK.pkl': 1.1380858281738875,
'East_OldcastleCoastal_Defuniak.pkl': 2.092576346803387,
'East_OldcastleCoastal_Fort Myers FL.pkl': 1.7955257556573896,
'East_OldcastleCoastal_Fort Pierce FL.pkl': 2.053790138535034,
'East_OldcastleCoastal_Gainesville FL.pkl': 2.7971291666571565,
'East_OldcastleCoastal_Gulfport MS.pkl': 3.2757066190799664,
'East_OldcastleCoastal_Haines City FL Hardscapes.pkl': 1.2109336275587022,
'East_OldcastleCoastal_Jacksonville FL-SAK.pkl': 1.6399735079683124,
'East_OldcastleCoastal_Jacksonville FL.pkl': 1.9171871075265179,
'East_OldcastleCoastal_Lehigh Acres FL.pkl': 1.880717800516973,
'East_OldcastleCoastal_Longwood FL.pkl': 1.4317259390917412,
'East_OldcastleCoastal_Orlando FL.pkl': 0.920956080744084,
'East_OldcastleCoastal_Pensacola FL-SAK.pkl': 2.41503815809707,
'East_OldcastleCoastal_Pompano FL Hardscapes.pkl': 1.3551980360494942,
'East_OldcastleCoastal_Pompano FL-SAK.pkl': 0.9029012077431496,
'East_OldcastleCoastal_Sarasota FL.pkl': 1.1305641847480947,
'East_OldcastleCoastal_Tampa FL - Anderson Rd.pkl': 1.1217280083216656,
'East_OldcastleCoastal_Tampa FL - Busch Blvd.pkl': 1.3061543654248913,
```

'East_OldcastleCoastal_Tampa FL-SAK.pkl': 1.1826045063921966,
'East_OldcastleCoastal_Theodore AL.pkl': 3.061739628410632,
'East_OldcastleCoastal_West Palm Beach FL.pkl': 0.8783607573089005,
'East_OldcastleCoastal_Zephyrhills FL.pkl': 1.787556483427466,
'Lawn and Garden_L&G Central_Aliceville AL.pkl': 8.588007393377156,
'Lawn and Garden_L&G Central_Amherst Junction WI.pkl': 9.250666370401788,
'Lawn and Garden_L&G Central_Bridgeport MI.pkl': 10.691629614153637,
'Lawn and Garden_L&G Central_Cleveland, TX.pkl': 5.409952251237716,
'Lawn and Garden_L&G Central_Dallas TX.pkl': 6.421269018171921,
'Lawn and Garden_L&G Central_Del Valle, TX.pkl': 5.73835080276837,
'Lawn and Garden_L&G Central_Harrah OK.pkl': 9.764769585309232,
'Lawn and Garden_L&G Central_Hope AR.pkl': 7.88155818615846,
'Lawn and Garden_L&G Central_Livingston TX.pkl': 4.719396338904277,
'Lawn and Garden_L&G Central_Marseilles IL.pkl': 3.6890224520223884,
'Lawn and Garden_L&G Central_Miami OK.pkl': 6.9361354422960115,
'Lawn and Garden_L&G Central_Paola KS.pkl': 11.238533420128054,
'Lawn and Garden_L&G Central_Powderly TX.pkl': 7.691435453227473,
'Lawn and Garden_L&G Central_Sauget IL.pkl': 6.681648914090256,
'Lawn and Garden_L&G Central_Tulia TX.pkl': 9.592493534458198,
'Lawn and Garden_L&G Central_Tylertown MS.pkl': 7.295774513879699,
'Lawn and Garden_L&G Central_Waterloo IN.pkl': 9.996425502195393,
'Lawn and Garden_L&G Northeast_Berlin NY.pkl': 28.89621745478501,
'Lawn and Garden_L&G Northeast_Carey OH.pkl': 4.496259023014833,
'Lawn and Garden_L&G Northeast_Castlewood VA.pkl': 3.008648530019278,
'Lawn and Garden_L&G Northeast_Chatsworth GA.pkl': 4.746034019113628,
'Lawn and Garden_L&G Northeast_Historical-Co-Packer.pkl': 28.196582315854137,
'Lawn and Garden_L&G Northeast_Hooksett NH.pkl': 12.665604483890375,
'Lawn and Garden_L&G Northeast_Lee MA.pkl': 2.8509538368380474,
'Lawn and Garden_L&G Northeast_Manchester NY.pkl': 14.146939991044821,
'Lawn and Garden_L&G Northeast_Mount Hope NJ.pkl': 12.368740846373344,
'Lawn and Garden_L&G Northeast_Poland Spring ME.pkl': 7.520751153769599,
'Lawn and Garden_L&G Northeast_Quakertown PA.pkl': 8.82512509445268,
'Lawn and Garden_L&G Northeast_Thomasville PA.pkl': 4.648828415295732,
'Lawn and Garden_L&G Northeast_Wyoming RI.pkl': 15.338762769103417,
'Lawn and Garden_L&G Southeast_Aberdeen NC.pkl': 6.748457355038361,
'Lawn and Garden_L&G Southeast_Bostwick FL.pkl': 3.2541934573662625,
'Lawn and Garden_L&G Southeast_Cross City FL.pkl': 2.976583413444688,
'Lawn and Garden_L&G Southeast_Davenport FL.pkl': 2.240506443862894,
'Lawn and Garden_L&G Southeast_Fort Green FL.pkl': 5.491789731702979,
'Lawn and Garden_L&G Southeast_Gaffney SC.pkl': 9.356386040244171,
'Lawn and Garden_L&G Southeast_Louisburg NC.pkl': 8.793627170217317,
'Lawn and Garden_L&G Southeast_Moore Haven FL.pkl': 2.114129961207233,

```
'Lawn and Garden_L&G Southeast_Pageland SC.pkl': 5.671329030489307,
'Lawn and Garden_L&G Southeast_Shady Dale GA.pkl': 6.60926458824032,
'Lawn and Garden_L&G Southeast_Walterboro SC.pkl': 3.58163878417245,
'National_AMTC_Saint Paul MN.pkl': 3.932416202302284,
'National_Anchor Wall Systems_Minnetonka MN.pkl': 3.2762939264907094,
'National_MoistureShield_Lowell AR.pkl': 5.131726485447699,
'National_MoistureShield_Springdale AR.pkl': 1.5572749110180906,
'National_Oldcastle Sakerete Billing_Easy Mix.pkl': 6.943698126623167,
'National_Oldcastle Sakerete Billing_Roberts Concrete.pkl': 7.03706589544625,
'National_Oldcastle Sakerete Billing_US Mix.pkl': 3.6921001870026116,
'National_Techniseal_Ash Grove Memphis.pkl': 5.222541129353486,
'National_Techniseal_Ash Grove Nebraska.pkl': 6.607108428415858,
'National_Techniseal_Candiac.pkl': 8.065686508675206,
'National_Techniseal_Coastal Tampa.pkl': 10.975614109963058,
'National_Techniseal_CPM Portland.pkl': 7.586512832211051,
'National_Techniseal_Ectra.pkl': 20.761889426573056,
'National_Techniseal_Eurl Leps Mehat.pkl': 10.84836291285565,
'National_Techniseal_EZ Mix.pkl': 4.951231733470895,
'National_Techniseal_Handy Concrete.pkl': 5.550416720015288,
'National_Techniseal_PTB  Compaktuna.pkl': 8.593376354640755,
'National_Techniseal_Ras.pkl': 12.055398034810263,
'National_Techniseal_Techmix.pkl': 4.172024530475322,
'National_Westile_Westile Roofing Products.pkl': 2.002653355927265,
'West_Amcor_North Salt Lake UT.pkl': 3.796928478803517,
'West_CPM_Frederickson, WA.pkl': 5.665443077101006,
'West_CPM_Kent, WA.pkl': 2.712883321331006,
'West_CPM_Northstar Consignment.pkl': 7.94673612576838,
'West_CPM_Pasco WA.pkl': 3.56483360422325,
'West_CPM_Portland, OR.pkl': 2.311203919063117,
'West_CPM_Spokane, WA.pkl': 3.723356591327165,
'West_Sierra_Fontana CA.pkl': 1.5931214785328576,
'West_Sierra_Reno NV.pkl': 3.1807752615095204,
'West_Sierra_San Carlos CA.pkl': 6.183617351068042,
'West_Sierra_Stockton CA.pkl': 1.9623982511617417,
'West_Superlite_Gilbert, AZ.pkl': 1.851831421088695,
'West_Superlite_Integra Product.pkl': 5.250760333891583,
'West_Superlite_Lone Butte.pkl': 1.7103344243279122,
'West_Superlite_North Las Vegas.pkl': 1.6522148142790454,
'West_Superlite_Superlite - Western 19th Ave.pkl': 1.425847769833303,
'West_Superlite_Tucson, AZ - Gardner Ln.pkl': 1.3309117239361563,
'West_Superlite_West Phoenix N. 42nd Ave, AZ.pkl': 0.8849324247526559}
```

In [46]:
```
# # storing our dictionary into a pickle file for future reference

# with open('SARIMAX_di_log.pkl', 'wb') as f:
#     pickle.dump(sarimax_di_log, f)


# with open('holt_winters_di_log.pkl', 'wb') as f:
#     pickle.dump(holt_win_di_log, f)

# with open('Exp_weight_avg_di_log.pkl', 'wb') as f:
#     pickle.dump(simple_avg_di_log, f)
```

Now we'll perform Differencing/Lag Transformation on our original data & use Dickey Fuller Test to check stationarity of our data

Now we'll check stationarity of all unique combinations and if they are then move it to "final_df" folder

Note that the below function only checksthe stationarity and does not perform any differencing yet

In [66]:
```
def check_move_stationary(names, locs):
    '''
        this functions checks stationarity of multiple files base on the p_value of Dickey Fuller test
        and moves them to "final_df" folder
    '''
    print(check_move_stationary.__doc__)
    non_stationary_names = []
    non_stationary_path = []
    for i in range(len(names)):
        path = locs[i]
        df_check = pd.read_pickle(path)

        timeseries = df_check['Actual_Sales'].resample('MS').mean().fillna("1").astype(float)

        rolmean = timeseries.rolling(12).mean()
        rolstd = timeseries.rolling(12).std()

        dftest = adfuller(timeseries, autolag='AIC')
        dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used',
                                                  'Number of Observations Used'])

        # keeping the p-value threshold as 0.05, if > threshold then it is not stationary
        if(dftest[1] > 0.05):
            non_stationary_path.append(path)
            non_stationary_names.append(names[i])
        else:
            # if it is good then storing the data into the final_df folder which contains stationary data
            df_check.to_pickle("./final_df/"+names[i]+'.pkl')


    return non_stationary_names, non_stationary_path
```

In [67]:
```
# here we are passing our unique_df files which are good to use

start = datetime.now()
non_stat_names, non_stationary_data = check_move_stationary(good_names_unique, good_paths_unique)
print('Time taken: ', datetime.now() - start)
```

```
        this functions checks stationarity of multiple files and moves them to "final_df" folder

Time taken:  0:00:04.784683
```

In [68]:
```
print('Total Number of data which is stationary from good_data: ', len(good_names_unique)
      - len(non_stationary_data))
print('Total Number of Good Data: ', len(good_names_unique))
print('Total Number of Non Stationary data: ', len(non_stationary_data))
```

```
Total Number of data which is stationary from good_data:  87
Total Number of Good Data:  209
Total Number of Non Stationary data:  122
```

- As we can see there are 122 files which are not stationary and are not fit for modelling
- 87 files which are stationary have been moved to "final_df" folder

In [70]:
```
print('One such Non Stationary Data: ', non_stat_names[0])
```

```
One such Non Stationary Data:  Canada_Expocrete_Acheson
```

In [71]:
```
# # storing this non_stationary list into a pickle file
# with open('non_stationary_paths.pkl', 'wb') as f:
#     pickle.dump(non_stationary_data, f)
```

In [72]:
```
# with open('non_stationary_names.pkl', 'wb') as f:
#     pickle.dump(non_stat_names, f)
```

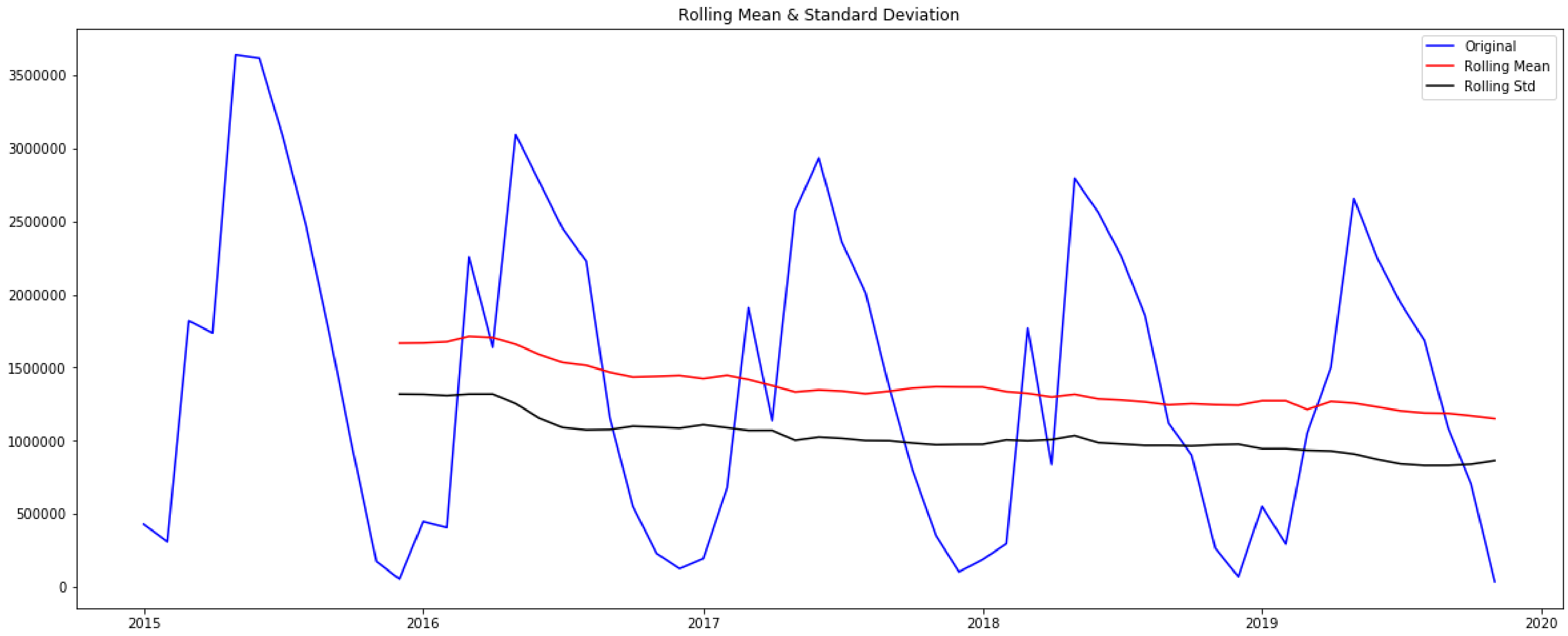
In [73]:
```
with open('non_stationary_names.pkl', 'rb') as f:
    non_stat_names = pickle.load(f)
```

In [74]:
```
# opening the pickle file
with open('non_stationary_paths.pkl', 'rb') as f:
    non_stat_paths = pickle.load(f)
```

Now, let's verify the stationarity of one such file which is non stationary

In [78]:
```
df_verify = pd.read_pickle(non_stat_paths[0])
test_stationarity(df_verify['Actual_Sales'].resample('MS').mean().fillna("1").astype(float))
```

```
this function tests stationarity of data using Dickey Fuller test
```

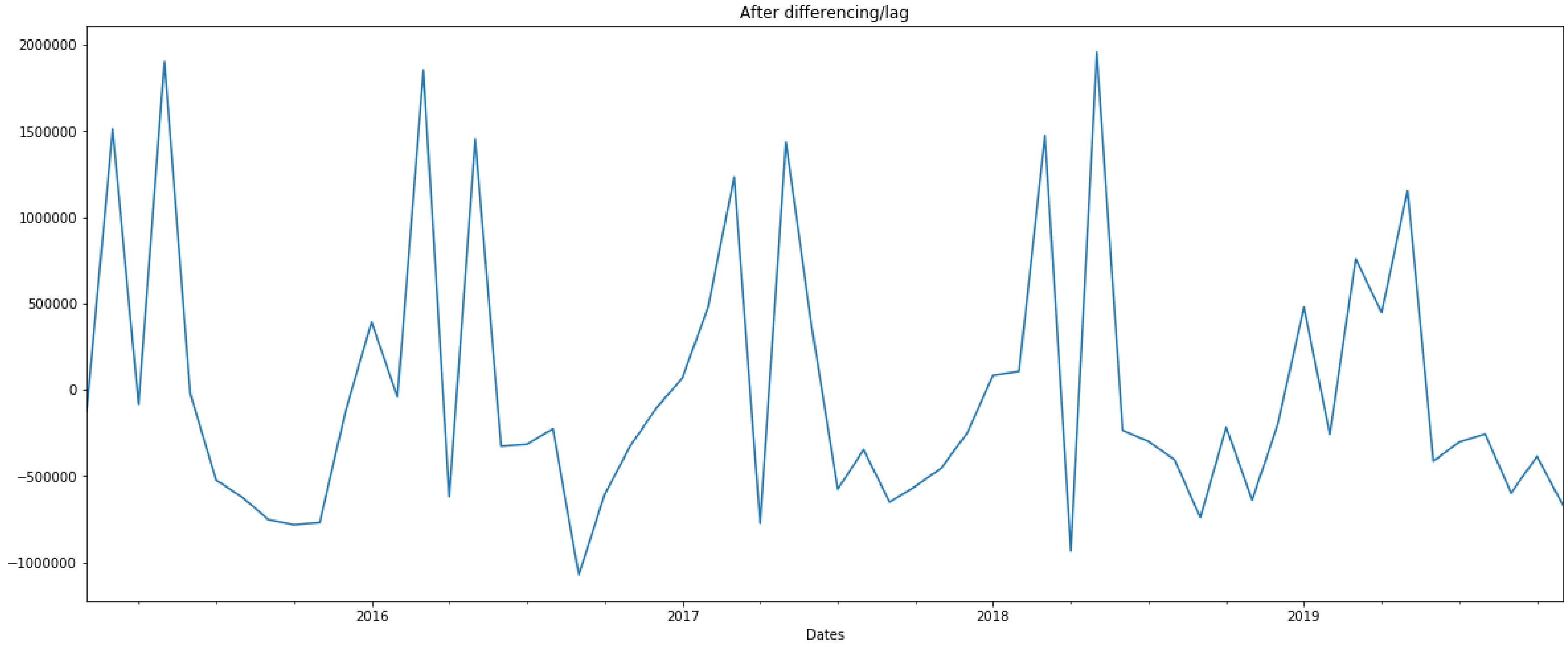



```
Results of Dickey-Fuller Test:
Test Statistic                 -1.404210
p-value                         0.580256
#Lags Used                     11.000000
Number of Observations Used    47.000000
Critical Value (1%)            -3.577848
Critical Value (5%)            -2.925338
Critical Value (10%)           -2.600774
dtype: float64
```

- As we can see above our stationarity check works fine and indeed the above data is non-stationary
- we can see the rolling mean and std is down trending
- the p-value is 0.5 which is greater than 0.05 our threshold

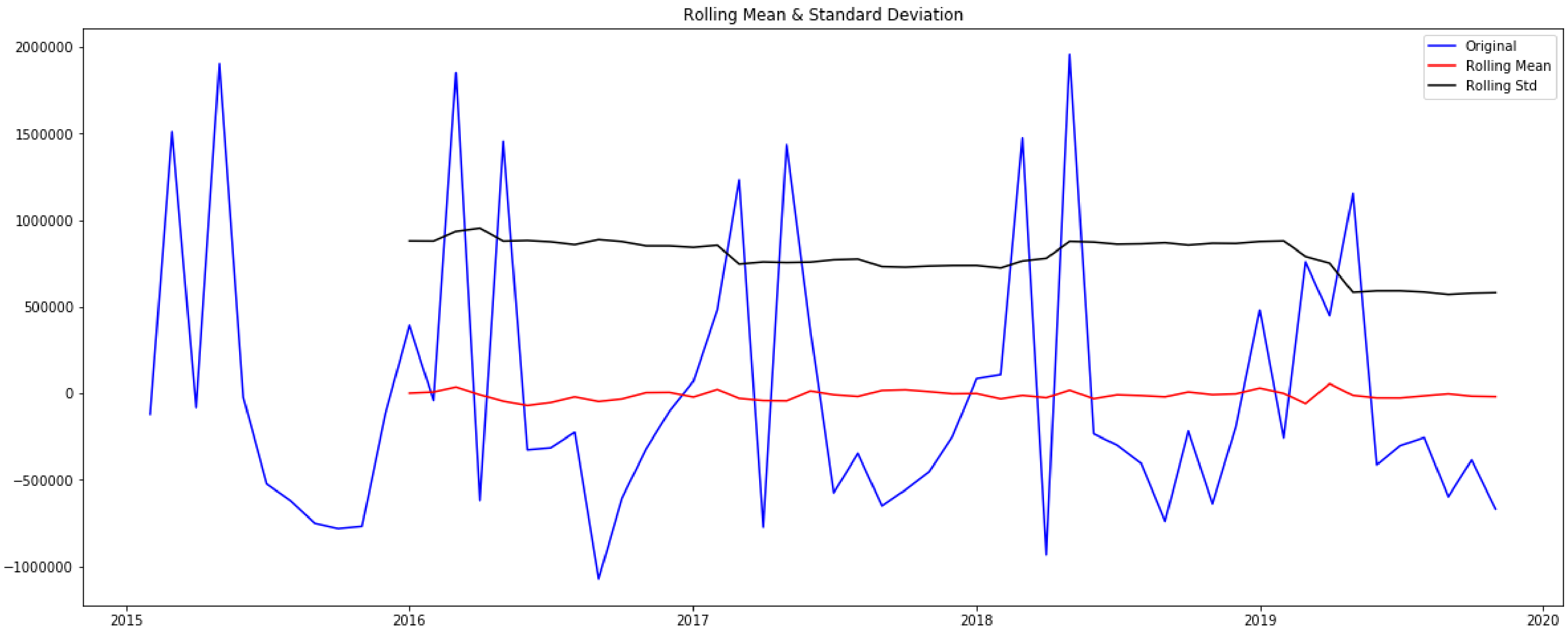
Now, testing 1st order differencing/lag and then again we'll check it's stationarity

In [81]:
```
# differencing on this data to make it stationary
plt.rcParams["figure.figsize"] = (20,8)
plt.title('After differencing/lag')
dff = pd.read_pickle(non_stat_paths[0])
dff['Actual_Sales'] = dff['Actual_Sales'] - dff['Actual_Sales'].shift(1)

dff['Actual_Sales'].dropna().plot()
dff['Actual_Sales'].dropna(inplace=True)
```



In [80]:
```
test_stationarity(dff['Actual_Sales'].resample('MS').mean().fillna("1").astype(float))
```

```
this function tests stationarity of data using Dickey Fuller test
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -3.738250
p-value                          0.003606
#Lags Used                      11.000000
Number of Observations Used     46.000000
Critical Value (1%)             -3.581258
Critical Value (5%)             -2.926785
Critical Value (10%)            -2.601541
dtype: float64
```

Observations

- Now from differencing above we cans see that our data has become stationary
- we can see that the p-value has become 0.003 and way less than 0.05 threshold of p-value
- we can also see the rolling mean and std has also become very constant from the dickey-fuller test

Now we'll convert the rest 122 non stationary data into stationary and move to "final_df" folder

Note that this function below performs differencing, checks stationarity again and sends them to "final_df" folder

In [205]:
```
def convert_to_stationary(names, paths=None, data=None):
    '''
        this function performs differencing/lag to multiple data and moves it to our
        "final_df" folder
    '''
    print(convert_to_stationary.__doc__)
    still_not_stat_names = []
    still_not_stat_data = []
    count = 0
    for i in range(len(names)):
        if data != None and paths == None:
            df_convert = data[i]
        else:
            df_convert = pd.read_pickle(paths[i])

        df_convert['Actual_Sales'] = df_convert['Actual_Sales'] - df_convert['Actual_Sales'].shift(1)
        df_convert['Actual_Sales'].dropna(inplace=True)
        timeseries = df_convert['Actual_Sales'].resample('MS').mean().fillna("1").astype(float)

        dftest = adfuller(timeseries, autolag='AIC')
        dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used',
                                                 'Number of Observations Used'])

        if(dftest[1] > 0.05):
            still_not_stat_data.append(df_convert)
            still_not_stat_names.append(names[i])
        else:
            count += 1
            df_convert.to_pickle("./final_df/"+names[i]+'.pkl')


    if paths != None:
        print('Total number of files: ', len(paths))
    else:
        print('Total number of files: ', len(data))

    print('Number of files stationary: ', count)

    return still_not_stat_names, still_not_stat_data
```

In [180]:
```
start = datetime.now()
one_non_stat_names, one_non_stat_data = convert_to_stationary(non_stat_names, paths=non_stat_paths,
                                                               data=None)
print('Time taken: ', datetime.now() - start)
```

```
        this function performs differencing/lag to multiple data and moves it to our
        "final_df" folder

Total number of files:  122
Number of files stationary:  108
Time taken:  0:00:02.402124
```

In [181]:
```
print('Number of data which are still non stationary even after 1st order differencing: ',
      len(one_non_stat_names))
```

```
Number of data which are still non stationary even after 1st order differencing:  14
```

Doing 2nd order differencing on 14 files which are still non stationary

In [182]:
```
start = datetime.now()
sec_non_stat_names, sec_non_stat_data = convert_to_stationary(one_non_stat_names, paths=None,
                                                               data=one_non_stat_data)
print('Time taken: ', datetime.now() - start)
```

```
        this function performs differencing/lag to multiple data and moves it to our
        "final_df" folder

Total number of files:  14
Number of files stationary:  12
Time taken:  0:00:00.134416
```

Observations

- as we can see we still have 2 files which are non stationary even after 2nd order differencing

Doing 3rd order differencing on 2 files which are still non stationary

In [204]:
```
start = datetime.now()
third_non_stat_names, third_non_stat_data = convert_to_stationary(sec_non_stat_names, paths=None,
                                                                   data=sec_non_stat_data)
print('Time taken: ', datetime.now() - start)
```

```
        this function performs differencing/lag to multiple data and moves it to our
        "final_df" folder

Total number of files:  2
Number of files stationary:  2
Time taken:  0:00:00.017363
```

In [206]:
```
ls_dir_f = os.listdir("./final_df/")
print('Number of files in our final_df folder: ', len(ls_dir_f))
```

```
Number of files in our final_df folder:  209
```

Finally, now since we have converted all our data into stationary, let's verify for the one last time

In [207]:
```
# now we'll check the stationarity from the final_df where all our stationarized data is stored
def check_non_stationary(names, paths):
    '''
        this function checks the stationarity of multiple files and returns those
        which are not stationary
    '''
    print(check_non_stationary.__doc__)
    non_stati_names = []
    non_stati_path = []
    for i in range(len(names)):
        df_check = pd.read_pickle(paths[i])

        timeseries = df_check['Actual_Sales'].resample('MS').mean().fillna("1").astype(float)

        dftest = adfuller(timeseries, autolag='AIC')
        dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used',
                                                  'Number of Observations Used'])

        # keeping the p-value threshold as 0.05, if > threshold then it is not stationary
        if(dftest[1] > 0.05):
            non_stati_path.append(paths[i])
            non_stati_names.append(names[i])

    return non_stati_names, non_stati_path
```

In [17]:
```
start = datetime.now()
good_df, good_names, good_paths, good_pairs, bad_names = get_files_from_dir("./final_df/")
print('Time taken: ', datetime.now() - start)
```

```
        This function checks if files are atleast having 2 years of data


Total files in folder:  207
Time taken:  0:00:01.483541
```

In [216]:
```
start = datetime.now()
non_stat_names_final, non_stationary_data_final = check_non_stationary(good_names, good_paths)
print('Time taken: ', datetime.now() - start)
```

```
        this funnction checks the stationarity of multiple files and returns those
        which are not stationary

Time taken:  0:00:03.654014
```

In [217]:
```
print('Number of files still non stationary: ', len(non_stat_names_final))
```

```
Number of files still non stationary:  0
```

Observations

- As we can see all our useful data has become stationary now and stored in the "final_df" folder

Baseline Models (Only for a single unique combination)

We will run multiple multiple models on one of the unique combination which has been made stationary using differencing and compare each model's performance

In [47]:
```
print('We\'ll test our models with this data first: ', good_names[1])
```

```
We'll test our models with this data first:  Canada_Expocrete_Acheson.pkl
```

In [219]:
```
test_df = pd.read_pickle(good_paths[1])
test_df.fillna(1, inplace=True)
test_df.head()
```

Out[219]:

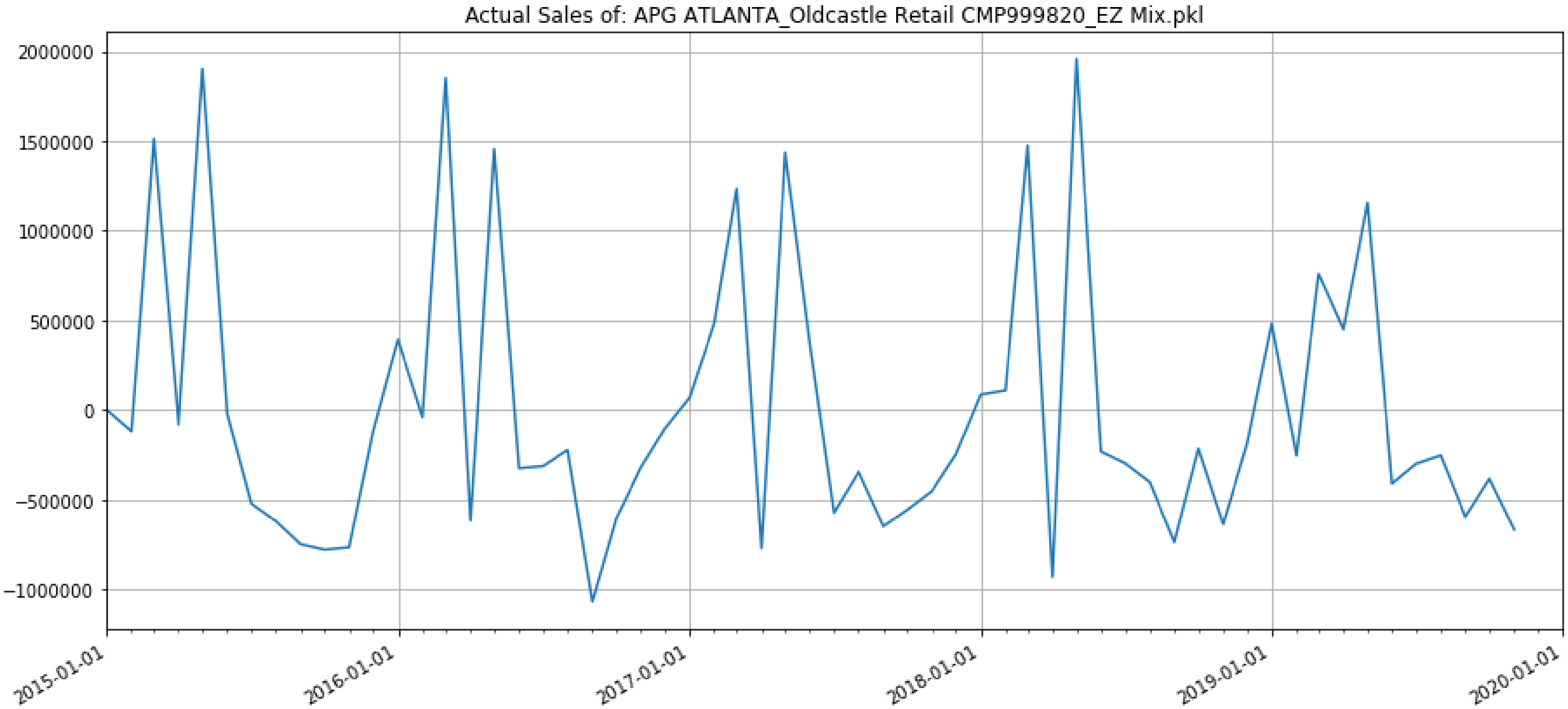
	Region	Division_Name	Facility_Name	Year	Month	Actual_Sales
Dates						
2015-01-01	Canada	Expocrete	Acheson	2015	1	1.0
2015-02-01	Canada	Expocrete	Acheson	2015	2	-119801.0
2015-03-01	Canada	Expocrete	Acheson	2015	3	1511601.0
2015-04-01	Canada	Expocrete	Acheson	2015	4	-82804.0
2015-05-01	Canada	Expocrete	Acheson	2015	5	1903458.0

In [220]:
```
print('Shape of our test data: ', test_df.shape)
```

```
Shape of our test data:  (59, 6)
```

In [221]:

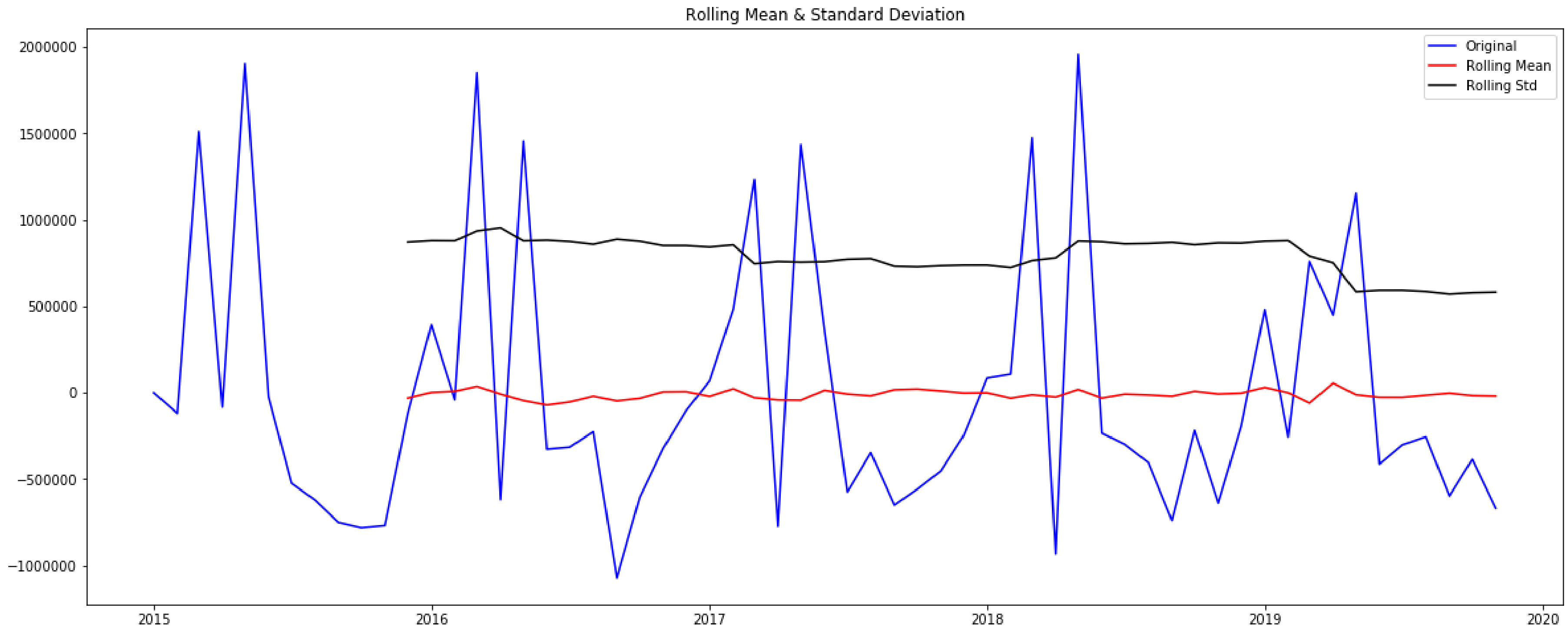
```
months = mdates.MonthLocator()  # every month
years = mdates.YearLocator()   # every year
years_fmt = mdates.DateFormatter('%Y-%m-%d')

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15,7))  # create figure & 1 axis
# ax.plot('Dates', 'Actual_Sales', data=df)
ax.plot(test_df['Actual_Sales'])
ax.set(xlabel='')
ax.xaxis.set_major_locator(years)
ax.xaxis.set_major_formatter(years_fmt)
ax.xaxis.set_minor_locator(months)

datemin = np.datetime64(df.index[0], 'Y')
datemax = np.datetime64(df.index[-1], 'Y') + np.timedelta64(1, 'Y')
ax.set_xlim(datemin, datemax)

ax.format_xdata = mdates.DateFormatter('%Y-%m-%d')
ax.format_ydata = lambda x: '$%1.2f' % x  # format the price.
ax.grid(True)

fig.autofmt_xdate()
plt.title('Actual Sales of: {0}'.format(good_names[0]))
plt.show()
```



- we can see there is some loss in our data after for 2015-Jan and 2017-Feb
- but we still have enough data to train our models

In [222]:
```
# let's check stationarity for our test data to verify everything we have done till now works or not
test_stationarity(test_df['Actual_Sales'].resample('MS').mean().fillna("1").astype(float))
```

```
this function tests stationarity of data using Dickey Fuller test
```




```
Results of Dickey-Fuller Test:
Test Statistic                -8.685063e+00
p-value                        4.184969e-14
#Lags Used                     1.000000e+01
Number of Observations Used    4.800000e+01
Critical Value (1%)           -3.574589e+00
Critical Value (5%)           -2.923954e+00
Critical Value (10%)          -2.600039e+00
dtype: float64
```

Observations

- As we can see our test data is perfectly stationary
- so our conversion of data to stationary works perfectly

Tyring Various Models on our test combination

In [224]:
```
X_train = test_df[:'2018-11-01']
X_test = test_df['2018-11-01':]

print('Shape of train: ', X_train.shape)
print('Shape of test: ', X_test.shape)
```

```
Shape of train:  (47, 6)
Shape of test:  (13, 6)
```

In [225]:
```
X_train.Actual_Sales
```

Out[225]:
```
Dates
2015-01-01          1.0
2015-02-01    -119801.0
2015-03-01    1511601.0
2015-04-01     -82804.0
2015-05-01    1903458.0
2015-06-01     -23300.0
2015-07-01    -523020.0
2015-08-01    -621223.0
2015-09-01    -750581.0
2015-10-01    -781027.0
2015-11-01    -767905.0
2015-12-01    -119652.0
2016-01-01     392783.0
2016-02-01     -41204.0
2016-03-01    1852383.0
2016-04-01    -618541.0
2016-05-01    1455295.0
2016-06-01    -326424.0
2016-07-01    -314836.0
2016-08-01    -225356.0
2016-09-01   -1071941.0
2016-10-01    -605488.0
2016-11-01    -321845.0
2016-12-01    -104274.0
2017-01-01      68909.0
2017-02-01     483288.0
2017-03-01    1233887.0
2017-04-01    -772921.0
2017-05-01    1435500.0
2017-06-01     360721.0
2017-07-01    -574788.0
2017-08-01    -346624.0
2017-09-01    -650290.0
2017-10-01    -558329.0
2017-11-01    -452933.0
2017-12-01    -248717.0
2018-01-01      84854.0
2018-02-01     108048.0
2018-03-01    1474812.0
2018-04-01    -932673.0
```

```
2018-05-01    1957920.0
2018-06-01    -233947.0
2018-07-01    -298932.0
2018-08-01    -403967.0
2018-09-01    -739080.0
2018-10-01    -217438.0
2018-11-01    -638798.0
Name: Actual_Sales, dtype: float64
```

In [226]:
```
X_test.Actual_Sales
```

Out[226]:
```
Dates
2018-11-01    -638798.0
2018-12-01    -193690.0
2019-01-01     480518.0
2019-02-01    -257020.0
2019-03-01     759319.0
2019-04-01     449097.0
2019-05-01    1154499.0
2019-06-01    -412425.0
2019-07-01    -302010.0
2019-08-01    -255183.0
2019-09-01    -597764.0
2019-10-01    -384919.0
2019-11-01    -668434.0
Name: Actual_Sales, dtype: float64
```

SARIMAX model

In [223]:
```
# 12 in seasonal_pdq is just yearly becuase our seasonal data is yearly
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

print('These are all the possible combinations individually\n')
print(pdq)
print()
print(seasonal_pdq)
print('\n')

print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
These are all the possible combinations individually

[(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)]

[(0, 0, 0, 12), (0, 0, 1, 12), (0, 1, 0, 12), (0, 1, 1, 12), (1, 0, 0, 12), (1, 0, 1, 12), (1, 1, 0, 12), (1, 1, 1, 1
2)]


Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

In [227]:
```
# hyperparameter tuning

# params = []
# params_seasonal = []
aic_vals = []
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(X_train.Actual_Sales,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)

            results = mod.fit()

            print('SARIMA{}x{} - AIC:{}'.format(param, param_seasonal, results.aic))
#             params.append(param)
#             params_seasonal.append(param_seasonal)
            aic_vals.append((param, param_seasonal, results.aic))
        except:
            continue
```

```
SARIMA(0, 0, 0)x(0, 0, 0, 12) - AIC:1384.7507541500506
SARIMA(0, 0, 0)x(0, 0, 1, 12) - AIC:8523977.470113501
SARIMA(0, 0, 0)x(0, 1, 0, 12) - AIC:965.6985138324096
SARIMA(0, 0, 0)x(1, 0, 0, 12) - AIC:993.1214747620172
SARIMA(0, 0, 0)x(1, 0, 1, 12) - AIC:8377665.594070783
SARIMA(0, 0, 0)x(1, 1, 0, 12) - AIC:647.0548176836143
SARIMA(0, 0, 1)x(0, 0, 0, 12) - AIC:1357.4991113952256
SARIMA(0, 0, 1)x(0, 0, 1, 12) - AIC:8509536.027780563
SARIMA(0, 0, 1)x(0, 1, 0, 12) - AIC:933.0695097479751
SARIMA(0, 0, 1)x(1, 0, 0, 12) - AIC:1026.6006350309585
SARIMA(0, 0, 1)x(1, 0, 1, 12) - AIC:8509394.469328035
SARIMA(0, 0, 1)x(1, 1, 0, 12) - AIC:642.4997696486863
SARIMA(0, 1, 0)x(0, 0, 0, 12) - AIC:1390.1484655356646
SARIMA(0, 1, 0)x(0, 0, 1, 12) - AIC:13824239.291712431
SARIMA(0, 1, 0)x(0, 1, 0, 12) - AIC:970.9385440319501
SARIMA(0, 1, 0)x(1, 0, 0, 12) - AIC:999.2237196963825
SARIMA(0, 1, 0)x(1, 0, 1, 12) - AIC:13561480.545502469
SARIMA(0, 1, 0)x(1, 1, 0, 12) - AIC:638.3426739334968
SARIMA(0, 1, 1)x(0, 0, 0, 12) - AIC:1326.797083903331
SARIMA(0, 1, 1)x(0, 0, 1, 12) - AIC:nan
SARIMA(0, 1, 1)x(0, 1, 0, 12) - AIC:909.8038300580222
SARIMA(0, 1, 1)x(1, 0, 0, 12) - AIC:1009.3639738105944
SARIMA(0, 1, 1)x(1, 0, 1, 12) - AIC:10511702.185160987
SARIMA(0, 1, 1)x(1, 1, 0, 12) - AIC:623.61753146276
SARIMA(1, 0, 0)x(0, 0, 0, 12) - AIC:1386.4279434045525
SARIMA(1, 0, 0)x(0, 0, 1, 12) - AIC:8245641.22168544
SARIMA(1, 0, 0)x(0, 1, 0, 12) - AIC:962.8294379925003
SARIMA(1, 0, 0)x(1, 0, 0, 12) - AIC:998.4346015294489
SARIMA(1, 0, 0)x(1, 0, 1, 12) - AIC:8245489.986336725
SARIMA(1, 0, 0)x(1, 1, 0, 12) - AIC:619.9767027124781
SARIMA(1, 0, 1)x(0, 0, 0, 12) - AIC:1357.845920475084
SARIMA(1, 0, 1)x(0, 0, 1, 12) - AIC:8231209.6558365505
SARIMA(1, 0, 1)x(0, 1, 0, 12) - AIC:934.6395810292064
SARIMA(1, 0, 1)x(1, 0, 0, 12) - AIC:999.2450179203947
SARIMA(1, 0, 1)x(1, 0, 1, 12) - AIC:8231062.9002917325
SARIMA(1, 0, 1)x(1, 1, 0, 12) - AIC:619.1722468038142
SARIMA(1, 1, 0)x(0, 0, 0, 12) - AIC:1359.4440141833122
SARIMA(1, 1, 0)x(0, 0, 1, 12) - AIC:8598825.90885154
SARIMA(1, 1, 0)x(0, 1, 0, 12) - AIC:955.8006298809224
SARIMA(1, 1, 0)x(1, 0, 0, 12) - AIC:975.6166977821912
SARIMA(1, 1, 0)x(1, 0, 1, 12) - AIC:7735385.836235457
```

```
SARIMA(1, 1, 0)x(1, 1, 0, 12) - AIC:604.3974493897485
SARIMA(1, 1, 1)x(0, 0, 0, 12) - AIC:1327.5048583351565
SARIMA(1, 1, 1)x(0, 0, 1, 12) - AIC:6171718.978726447
SARIMA(1, 1, 1)x(0, 1, 0, 12) - AIC:908.6669851781537
SARIMA(1, 1, 1)x(1, 0, 0, 12) - AIC:971.3068850032347
SARIMA(1, 1, 1)x(1, 0, 1, 12) - AIC:5325008.499038684
SARIMA(1, 1, 1)x(1, 1, 0, 12) - AIC:594.4265669396103
```

In [228]:

```
# print(len(params), len(params_seasonal), len(aic_vals))
print('Length of params: {0} and an example {1}'.format(len(aic_vals), aic_vals[0]))

clean_aic_vals = [(i, j, k) for i, j, k in aic_vals if not np.isnan(k)]

print('After removing nan values, length:', len(clean_aic_vals))

scores = [k for i, j, k in clean_aic_vals]
idx = np.argmin(scores)
print('Best params for SARIMAX are: {0}x{1} and score: {2}'.format(clean_aic_vals[idx][0],
                                                                clean_aic_vals[idx][1],
                                                                clean_aic_vals[idx][2]))
```

```
Length of params: 48 and an example ((0, 0, 0), (0, 0, 0, 12), 1384.7507541500506)
After removing nan values, length: 47
Best params for SARIMAX are: (1, 1, 1)x(1, 1, 0, 12) and score: 594.4265669396103
```

In [28]:

```
start = len(X_train)
end = (len(X_train) + len(X_test)) - 1
print(start, end)
```

```
47 59
```

In [230]:
```
model = sm.tsa.statespace.SARIMAX(X_train.Actual_Sales,
                                order=clean_aic_vals[idx][0],
                                seasonal_order=clean_aic_vals[idx][1],
                                enforce_stationarity=False,
                                enforce_invertibility=False)

results_final = model.fit()

print(results_final.summary().tables[1])
```

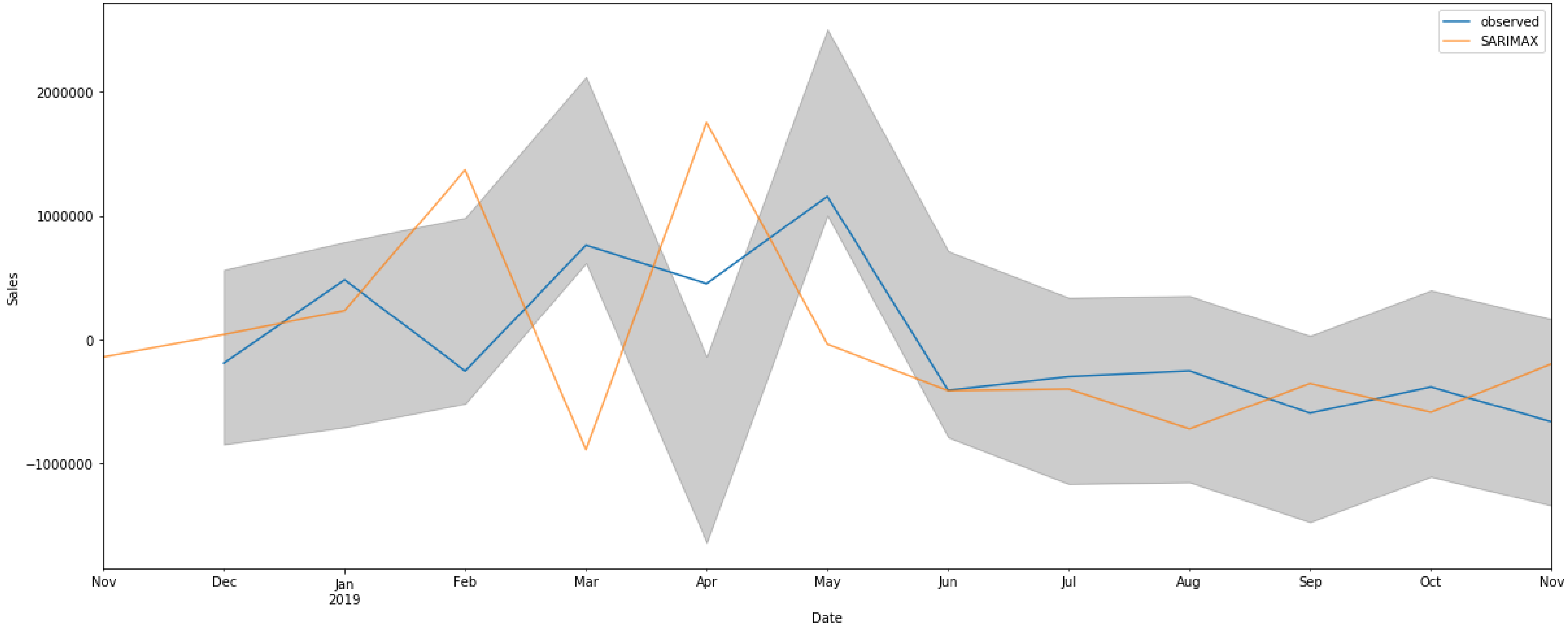
```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3335      0.601     -0.555      0.579      -1.511       0.843
ma.L1         -1.0222      0.061    -16.663      0.000      -1.142      -0.902
ar.S.L12      -0.3561      0.502     -0.709      0.478      -1.341       0.629
sigma2      1.294e+11   1.88e-12   6.88e+22      0.000    1.29e+11    1.29e+11
==============================================================================
```

In [231]:
```
pred = results_final.get_prediction(start=start, end=end,
                              dynamic=False)
pred_ci = pred.conf_int()
```

In [232]:
```
preds = pred.predicted_mean
preds.index = X_test.index
preds
```

Out[232]:
```
Dates
2018-11-01   -1.439272e+05
2018-12-01    3.717359e+04
2019-01-01    2.314659e+05
2019-02-01    1.368219e+06
2019-03-01   -8.930556e+05
2019-04-01    1.753452e+06
2019-05-01   -4.025303e+04
2019-06-01   -4.153421e+05
2019-07-01   -4.016891e+05
2019-08-01   -7.256192e+05
2019-09-01   -3.569714e+05
2019-10-01   -5.907679e+05
2019-11-01   -1.993917e+05
dtype: float64
```

In [233]:
```
plt.rcParams["figure.figsize"] = (20,8)
ax = X_test.Actual_Sales['2018-12-01':].plot(label='observed')
preds.plot(ax=ax, label='SARIMAX', alpha=.7)

ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)

ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.legend()

plt.show()
```



Observations

- As we can see there is a huge error in out predicted plot for March 2019
- This will certainly effect the performance metric

In [234]:
```
y_forecasted = pred.predicted_mean
y_truth = X_test.Actual_Sales
```

In [235]:
```
print(len(y_forecasted), len(y_truth))
```

```
13 13
```

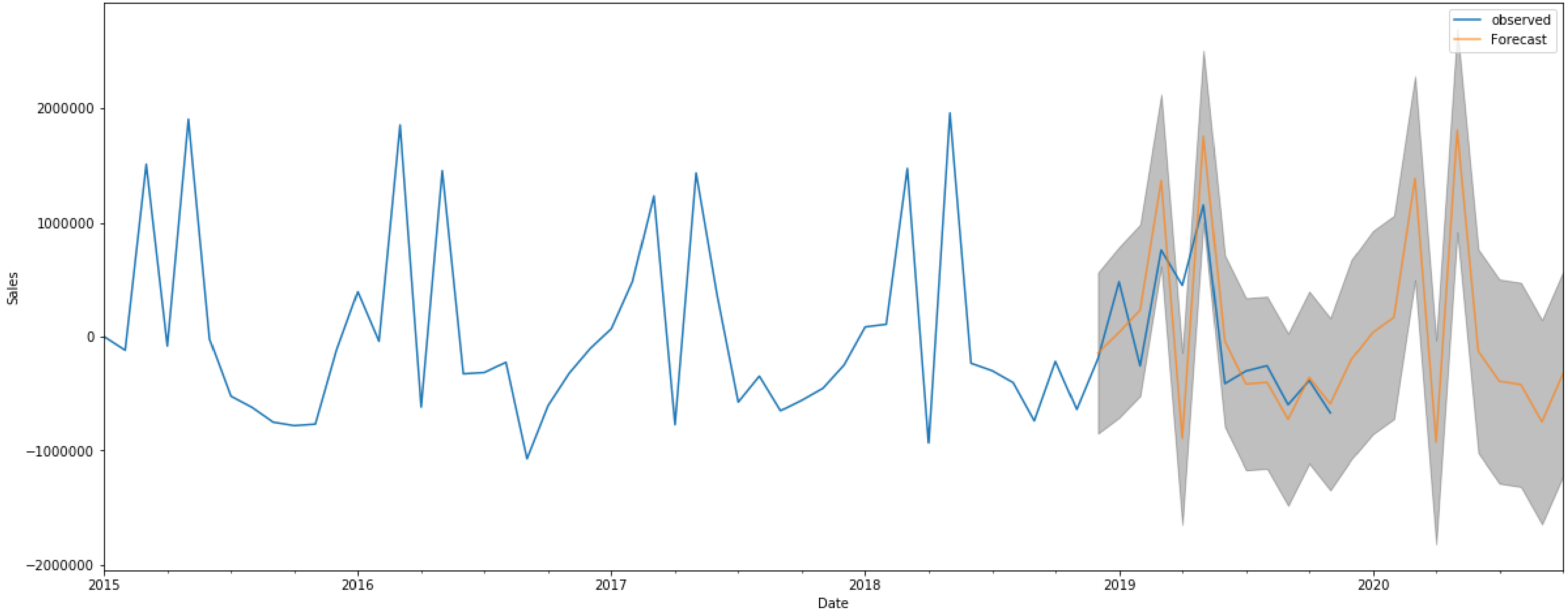
In [236]:
```
mape_val = mape(y_truth, y_forecasted)

print('Mean Absoulte percentage error: ', round(mape_val, 2))
```

```
Mean Absoulte percentage error:  144.18
```

Observations

- MAPE for our SARIMAX model on differenced data is 144.18
- This performance is way worse as compared to our Log Transformed model data

In [69]:
```
pred_uc = results.get_forecast(steps=23)
pred_ci = pred_uc.conf_int()
ax = test_df.Actual_Sales.plot(label='observed', figsize=(20, 8))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast', alpha=.7)
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.legend()
plt.show()
```



Observations

- Doing some future forecasting as well and predicting values for 2020
- Certianly our future forecasts look similar to our data at past time steps
- This shows that model is at least learning something

In [70]:
```
print('Predicted values from 2018-12-01')
y_forecasted = pred.predicted_mean
y_forecasted.head(12)
```

```
Predicted values from 2018-12-01
```

Out[70]:
```
2018-12-01   -2.933890e+05
2019-01-01    4.018200e+04
2019-02-01    6.337600e+04
2019-03-01    1.430140e+06
2019-04-01   -9.773450e+05
2019-05-01    1.913248e+06
2019-06-01   -2.786190e+05
2019-07-01   -3.436040e+05
2019-08-01   -4.486390e+05
2019-09-01   -7.837520e+05
2019-10-01   -2.621100e+05
2019-11-01   -6.834700e+05
Freq: MS, dtype: float64
```

Observations

- Above is the predicted values from 2018-12-01 till 2019-11-01

In [71]:
```
print('Actual Values from 2019-01-01')
y_truth.head(12)
```

```
Actual Values from 2019-01-01
```

Out[71]:
```
Dates
2018-11-01    -638798.0
2018-12-01    -193690.0
2019-01-01     480518.0
2019-02-01    -257020.0
2019-03-01     759319.0
2019-04-01     449097.0
2019-05-01    1154499.0
2019-06-01    -412425.0
2019-07-01    -302010.0
2019-08-01    -255183.0
2019-09-01    -597764.0
2019-10-01    -384919.0
Name: Actual_Sales, dtype: float64
```

Observations

- Below we have our future forecasts from 2019-12-01 till 2020-10-01

In [72]:
```
forecast = pred_uc.predicted_mean
forecast['2019-12-01':]
```

Out[72]:
```
2019-12-01   -1.993917e+05
2020-01-01    3.599987e+04
2020-02-01    1.693682e+05
2020-03-01    1.388023e+06
2020-04-01   -9.253139e+05
2020-05-01    1.808107e+06
2020-06-01   -1.273743e+05
2020-07-01   -3.920428e+05
2020-08-01   -4.206517e+05
2020-09-01   -7.485638e+05
2020-10-01   -3.254384e+05
Freq: MS, dtype: float64
```

Below we will follow the same methodology for the Holt-Winters Model and Exponenetial Weighted Avg Model

Now, Holt - Winters Model

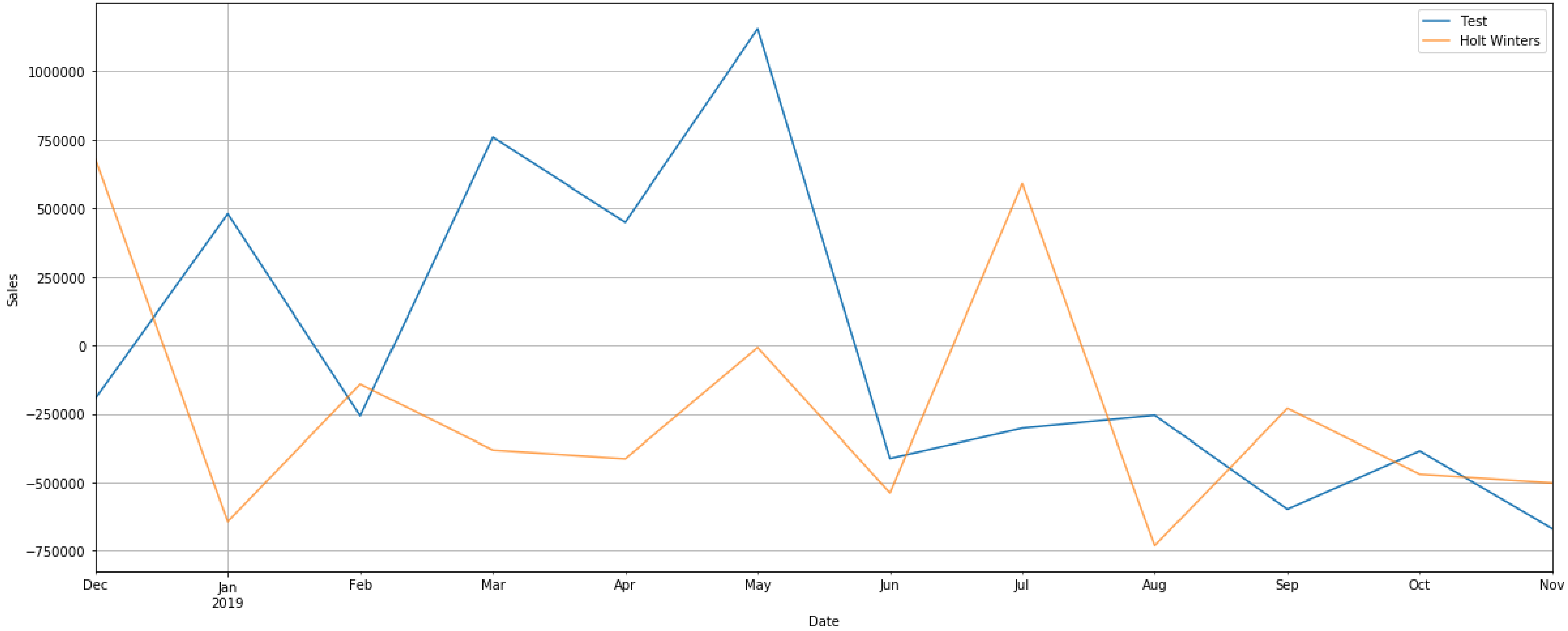
In [337]:
```
model_hw = ExponentialSmoothing(X_train.Actual_Sales, seasonal_periods=7, trend='add', seasonal='add')
results_hw = model_hw.fit()

print(results_hw.summary().tables[1])
```

```
================================================================================
                           coeff                 code              optimized
--------------------------------------------------------------------------------
smoothing_level              0.1052632                alpha                 True
smoothing_slope              0.1052632                 beta                 True
smoothing_seasonal           0.3684211                gamma                 True
initial_level               2.1622e+05                  l.0                 True
initial_slope                 0.000000                  b.0                 True
initial_seasons.0          -2.1622e+05                  s.0                 True
initial_seasons.1          -3.3602e+05                  s.1                 True
initial_seasons.2           1.2954e+06                  s.2                 True
initial_seasons.3          -2.9902e+05                  s.3                 True
initial_seasons.4           1.6872e+06                  s.4                 True
initial_seasons.5          -2.3952e+05                  s.5                 True
initial_seasons.6          -7.3924e+05                  s.6                 True
--------------------------------------------------------------------------------
```

In [338]:
```
y_pred_hw = results_hw.predict(start=pd.to_datetime('2018-12-01'), end=pd.to_datetime('2019-11-01'))

y_pred_hw
```

Out[338]:
```
2018-12-01    678553.756730
2019-01-01   -642849.616222
2019-02-01   -141734.132740
2019-03-01   -381965.983041
2019-04-01   -413600.781009
2019-05-01     -7543.660151
2019-06-01   -538107.502311
2019-07-01    590581.665564
2019-08-01   -730821.707388
2019-09-01   -229706.223906
2019-10-01   -469938.074207
2019-11-01   -501572.872175
Freq: MS, dtype: float64
```

In [339]:
```
plt.rcParams["figure.figsize"] = (20,8)
ax = test_df.Actual_Sales['2018-12-01':].plot(label='Test')
y_pred_hw.plot(ax=ax, label='Holt Winters', alpha=.7)

ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.legend()
plt.grid()
plt.show()
```



Observations

- As we can see the predicitve plot above that Holt-Winters model seems to perform more poorly than our SARIMAX model
- Let's check the performance metric on our Test Data

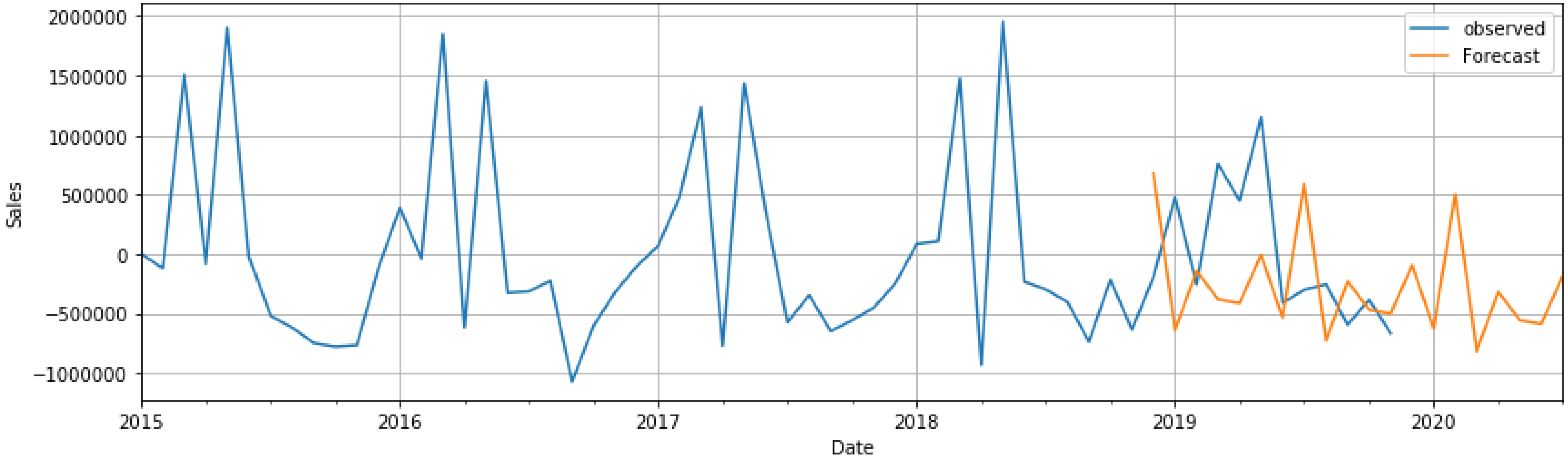
In [340]:
```
mape_val = mape(y_truth, y_pred_hw)

print('Mean Absoulte percentage error: ', round(mape_val, 2))
```

```
Mean Absoulte percentage error:  149.42
```

Observations

- MAPE for our Holt-Winters model on differenced data is 149.42
- This performance is also poor to as compared to our Log Transformed model data
- As expected from the plot above it is worse than the SARIMAX Model performance as well

In [341]:
```
y_forecast_hw = results_hw.forecast(steps=20)
ax = test_df.Actual_Sales.plot(label='observed', figsize=(14, 4))
y_forecast_hw.plot(ax=ax, label='Forecast')

ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.legend()
plt.grid()
plt.show()
```



Observations

- Our model seems to be not learning anything and is not really forecasting well enough into our future

In [342]:
```
y_forecast_hw['2019-12-01':]
```

Out[342]:
```
2019-12-01    -95515.751317
2020-01-01   -626079.593476
2020-02-01    502609.574398
2020-03-01   -818793.798553
2020-04-01   -317678.315072
2020-05-01   -557910.165373
2020-06-01   -589544.963340
2020-07-01   -183487.842483
Freq: MS, dtype: float64
```

Observations

- Above is our future forecasts which seems to be showing a negative sales data, which might not be accurate based on the model performance

Exponential Weighted Average

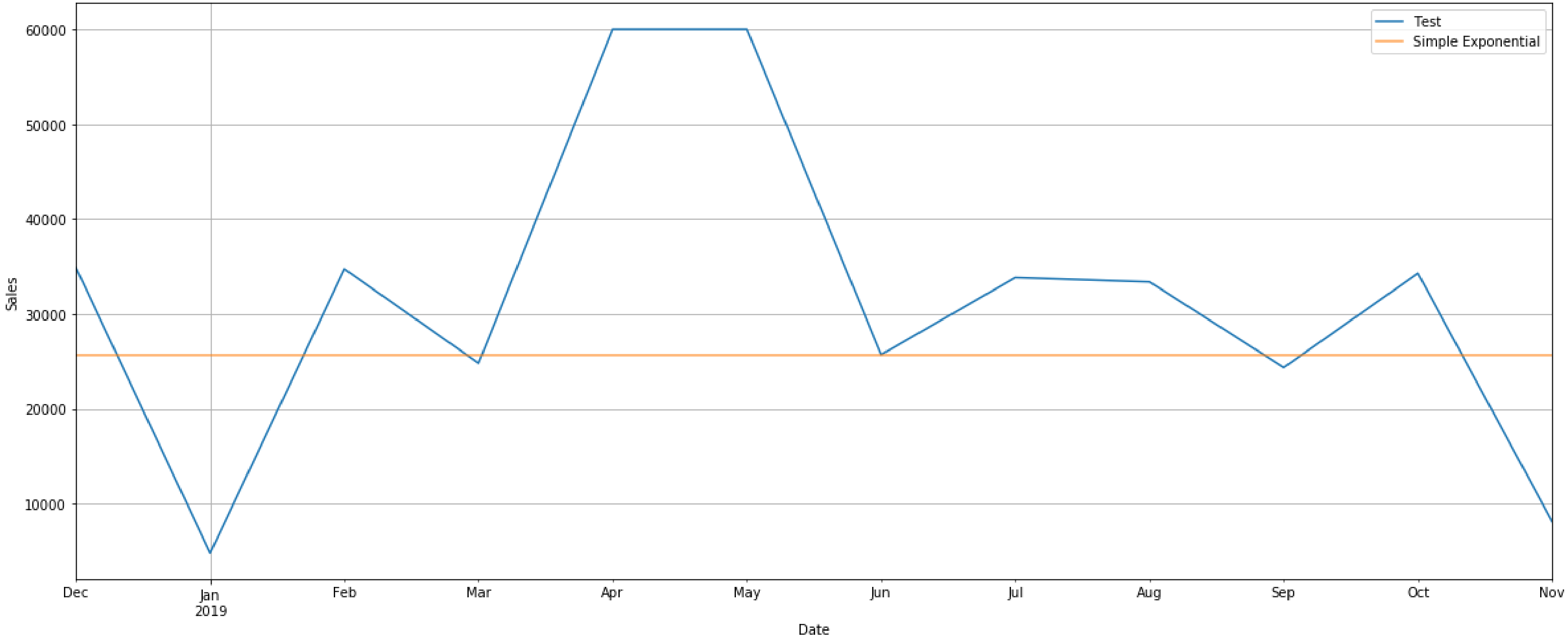
In [159]:
```
model_se = SimpleExpSmoothing(train.Actual_Sales)
results_se = model_se.fit()

print(results_se.summary().tables[1])
```

```
==============================================================================
                       coeff                 code              optimized
------------------------------------------------------------------------------
smoothing_level           0.4410054                alpha                 True
initial_level             43422.796                  1.0                 True
------------------------------------------------------------------------------
```

In [160]:
```
y_pred_se = results_se.predict(start=pd.to_datetime('2018-12-01'), end=pd.to_datetime('2019-11-01'))

y_pred_se
```

Out[160]:
```
2018-12-01    25674.100224
2019-01-01    25674.100224
2019-02-01    25674.100224
2019-03-01    25674.100224
2019-04-01    25674.100224
2019-05-01    25674.100224
2019-06-01    25674.100224
2019-07-01    25674.100224
2019-08-01    25674.100224
2019-09-01    25674.100224
2019-10-01    25674.100224
2019-11-01    25674.100224
Freq: MS, dtype: float64
```

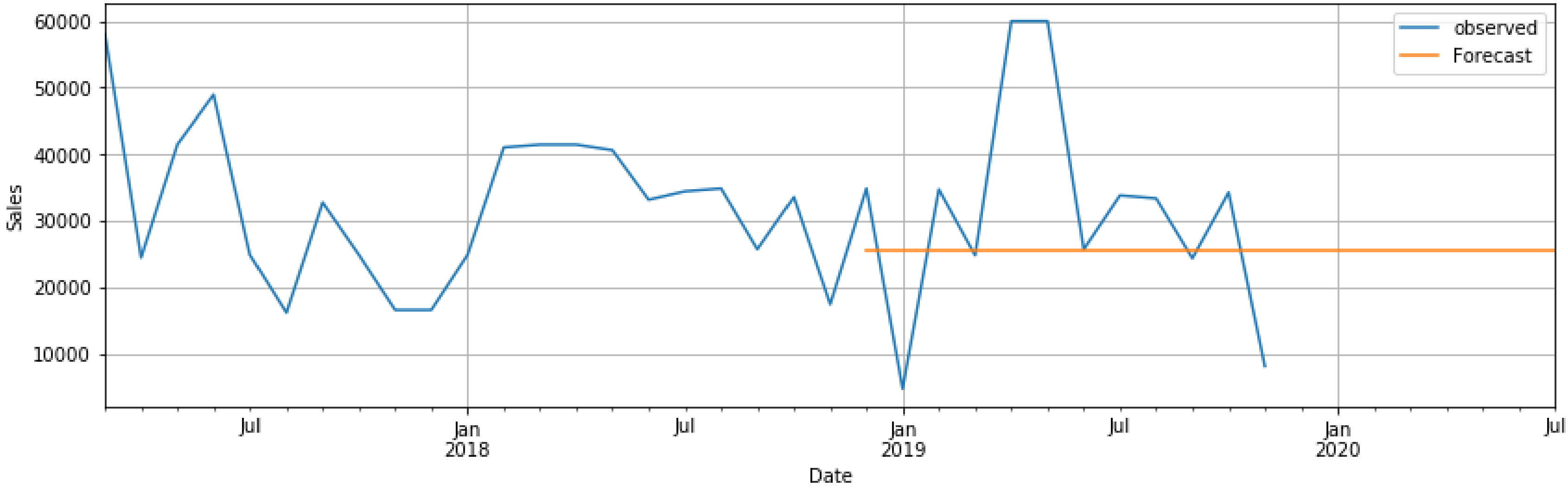
In [161]:
```
plt.rcParams["figure.figsize"] = (20,8)
ax = test_df.Actual_Sales['2018-12-01':].plot(label='Test')
y_pred_se.plot(ax=ax, label='Simple Exponential', alpha=.7)

ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.legend()
plt.grid()
plt.show()
```



In [162]:
```
mape_val = mape(y_truth, y_pred_se)

print('Mean Absoulte percentage error: ', round(mape_val, 2))
```

```
Mean Absoulte percentage error:  74.98
```

In [164]:
```
y_forecast_se = results_se.forecast(steps=20)
ax = test_df.Actual_Sales.plot(label='observed', figsize=(14, 4))
y_forecast_se.plot(ax=ax, label='Forecast')

ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.legend()
plt.grid()
plt.show()
```



In [165]:
```
y_forecast_se['2019-12-01':]
```

Out[165]:
```
2019-12-01    25674.100224
2020-01-01    25674.100224
2020-02-01    25674.100224
2020-03-01    25674.100224
2020-04-01    25674.100224
2020-05-01    25674.100224
2020-06-01    25674.100224
2020-07-01    25674.100224
Freq: MS, dtype: float64
```

Holt's Model

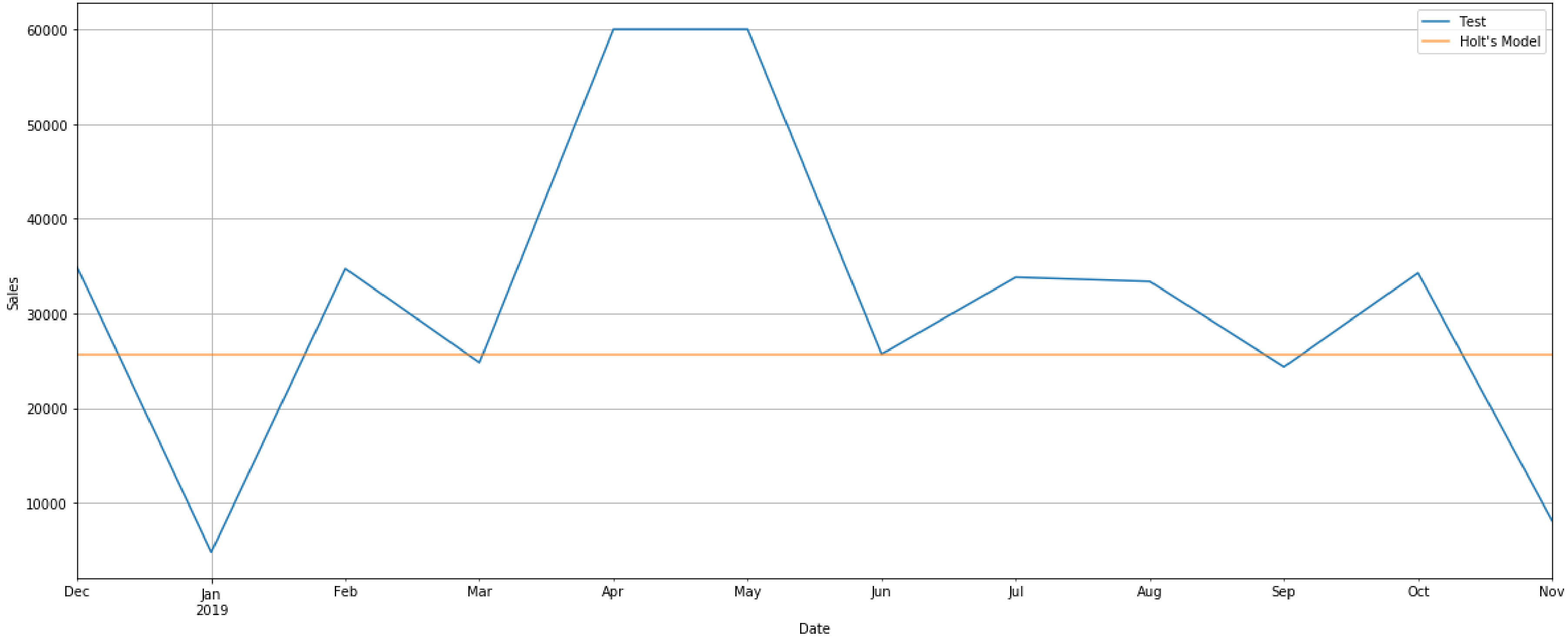
In [166]:
```
model_de = ExponentialSmoothing(train.Actual_Sales, trend='add')
results_de = model_de.fit()

print(results_de.summary().tables[1])
```

```
==============================================================================
                       coeff                 code              optimized
------------------------------------------------------------------------------
smoothing_level          0.4410870                alpha                 True
smoothing_slope           0.000000                 beta                 True
initial_level            43427.766                  l.0                 True
initial_slope             0.000000                  b.0                 True
------------------------------------------------------------------------------
```

In [167]:
```
y_pred_de = results_de.predict(start=pd.to_datetime('2018-12-01'), end=pd.to_datetime('2019-11-01'))

y_pred_de
```

Out[167]:
```
2018-12-01    25672.74393
2019-01-01    25672.74393
2019-02-01    25672.74393
2019-03-01    25672.74393
2019-04-01    25672.74393
2019-05-01    25672.74393
2019-06-01    25672.74393
2019-07-01    25672.74393
2019-08-01    25672.74393
2019-09-01    25672.74393
2019-10-01    25672.74393
2019-11-01    25672.74393
Freq: MS, dtype: float64
```

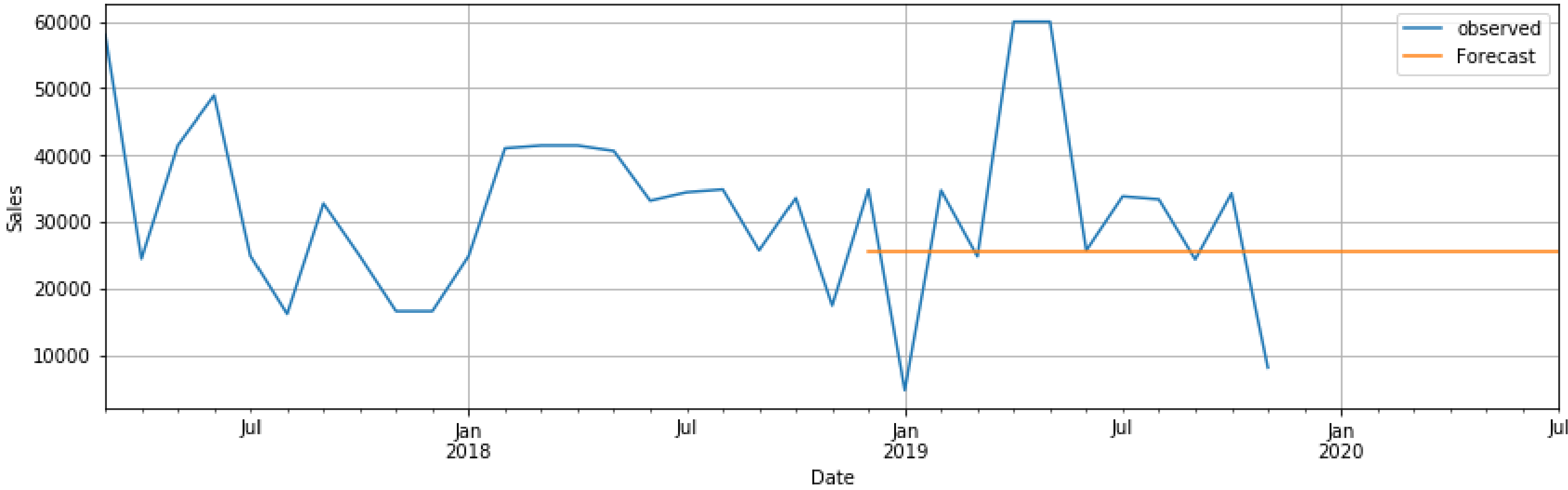
In [169]:
```
plt.rcParams["figure.figsize"] = (20,8)
ax = test_df.Actual_Sales['2018-12-01':].plot(label='Test')
y_pred_de.plot(ax=ax, label='Holt\'s Model', alpha=.7)

ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.legend()
plt.grid()
plt.show()
```



In [170]:
```
mape_val = mape(y_truth, y_pred_de)

print('Mean Absoulte percentage error: ', round(mape_val, 2))
```

```
Mean Absoulte percentage error:  74.98
```

In [172]:
```
y_forecast_de = results_de.forecast(steps=20)
ax = test_df.Actual_Sales.plot(label='observed', figsize=(14, 4))
y_forecast_de.plot(ax=ax, label='Forecast')

ax.set_xlabel('Date')
ax.set_ylabel('Sales')
plt.legend()
plt.grid()
plt.show()
```



In [173]:
```
y_forecast_de['2019-12-01':]
```

Out[173]:
```
2019-12-01    25672.74393
2020-01-01    25672.74393
2020-02-01    25672.74393
2020-03-01    25672.74393
2020-04-01    25672.74393
2020-05-01    25672.74393
2020-06-01    25672.74393
2020-07-01    25672.74393
Freq: MS, dtype: float64
```

Running models on all data

Running our best models which is SARIMAX, Holt-Winters and Exponential Weighted Avg Model on the Differencing/Lag Transformed data

Now running Holt Winters Model on all files

In [128]:
```
def hw_on_all(names, paths):
    '''
        Running Holt winters model/triple exponential on all combinations
    '''
    hw_di = {} # dictionary of holt winters model, contains name and mape score
    for i in range(len(names)):
#           print(names[i])
        df_hw = pd.read_pickle(paths[i])
        df_hw.fillna(1, inplace=True)
        train = df_hw[:'2018-11-01']
        test = df_hw['2018-11-01':]
        start = len(train)
        end = (len(train) + len(test)) - 1

        model_hw = ExponentialSmoothing(train.Actual_Sales, seasonal_periods=7,
                                        trend='add', seasonal='add')
        results_hw = model_hw.fit()
        y_pred_hw = results_hw.predict(start=start, end=end)

        y_true = test.Actual_Sales
        mape_val = mape(y_true, y_pred_hw)
        hw_di[names[i]] = mape_val

    return hw_di
```

In [240]:
```
def se_on_all(names, paths):
    '''
        Running exponential weighted avg/Simple Exponential on all files
    '''
    print(se_on_all.__doc__)
    se_di = {} # dcitionary of Simple Exponential Model, contains names and mape scores
    for i in range(len(names)):
        df_se = pd.read_pickle(paths[i])
        df_se.fillna(1, inplace=True)
        train = df_se[:'2018-11-01']
        test = df_se['2018-11-01':]
        start = len(train)
        end = (len(train) + len(test)) - 1

        model_se = SimpleExpSmoothing(train.Actual_Sales)
        results_se = model_se.fit()

        y_pred_se = results_se.predict(start=start, end=end)

        y_true = test.Actual_Sales
        mape_val = mape(y_true, y_pred_se)
        se_di[names[i]] = mape_val

    return se_di
```

In [215]:
```
def sarimax_on_all(names, paths, pdq_ls, seasonal_pdq_ls):
    '''
        running SARIMAX model with hyperparameter tuning on all files,
        this captures seasonality as well as trend
    '''
    print(sarimax_on_all.__doc__)
    sari_di = {} # dictionary of sarimax model, contains name and mape score
    count = 1
    for i in range(len(names)):
        df_sari = pd.read_pickle(paths[i])
        df_sari.fillna(1, inplace=True)
        train = df_sari[:'2018-11-01']
        test = df_sari['2018-11-01':]
        start = len(train)
        end = (len(train) + len(test)) - 1

        # now hyperparameter tuning
        aic_vals = []
        for param in pdq_ls:
            for param_seasonal in seasonal_pdq_ls:
                try:
                    mod = sm.tsa.statespace.SARIMAX(train.Actual_Sales,
                                                    order=param,
                                                    seasonal_order=param_seasonal,
                                                    enforce_stationarity=False,
                                                    enforce_invertibility=False)
                    results = mod.fit()
                    aic_vals.append((param, param_seasonal, results.aic))
                except:
                    continue

        # selecting best params
        clean_aic_vals = [(i, j, k) for i, j, k in aic_vals if not np.isnan(k)]
        scores = [k for i, j, k in clean_aic_vals]
        idx_min = np.argmin(scores)

        # traning our model with best params
        model = sm.tsa.statespace.SARIMAX(train.Actual_Sales,
                                          order=clean_aic_vals[idx_min][0],
                                          seasonal_order=clean_aic_vals[idx_min][1],
                                          enforce_stationarity=False,
```

```
                                            enforce_invertibility=False)

        results_final = model.fit()

        # predicting on test data
        pred = results_final.get_prediction(start=start, end=end, dynamic=False)
        preds = pred.predicted_mean
        preds.index = test.index

        # calculating MAPE score
        y_pred_sari = pred.predicted_mean
        y_true = test.Actual_Sales
        mape_val = mape(y_true, y_pred_sari)
        sari_di[names[i]] = mape_val

        print('{0} Files---done!'.format(count))
        count += 1

    return sari_di
```

Running Exponential Weighted Average Model

In [201]:

```
start = datetime.now()
good_df_f, good_names_f, good_paths_f, good_pairs_f, bad_names_f = get_files_from_dir("./final_df/")
print('Time taken: ', datetime.now() - start)
```

```
        This function checks if files are atleast having 2 years of data


Total files in folder:  207
Time taken:  0:00:00.455826
```

In [241]:
```
start = datetime.now()
simple_avg_di = se_on_all(good_names_f, good_paths_f)
print('\nTime taken: ', datetime.now() - start)
```

```
        Running exponential weighted avg/Simple Exponential on all files


Time taken:  0:00:02.164519
```

In [242]:
```
simple_avg_di
```

Out[242]:
```
{'APG ATLANTA_Oldcastle Retail CMP999820_EZ Mix.pkl': 72.84604355583394,
 'Canada_Expocrete_Acheson.pkl': 100.00015038173412,
 'Canada_Expocrete_Balzac.pkl': 100.0,
 'Canada_Expocrete_Edmonton.pkl': 99.99961050732165,
 'Canada_Expocrete_Richmond.pkl': 99.70302996454639,
 'Canada_Expocrete_Saskatoon.pkl': 100.0,
 'Canada_Expocrete_Winnipeg.pkl': 43.89612282241068,
 'Canada_Permacon_Milton ON.pkl': 93.40841190029032,
 'Canada_Permacon_Montreal QC.pkl': 100.0,
 'Canada_Permacon_Woodstock Ontario.pkl': 100.02342022191189,
 'Central_Ash Grove MPC_Fort Smith, AR.pkl': 24.964054484779936,
 'Central_Ash Grove MPC_Fremont, NE.pkl': 375.3285620132314,
 'Central_Ash Grove MPC_Harrisonville, MO.pkl': 103.39289457617993,
 'Central_Ash Grove MPC_Jackson, MS.pkl': 100.0,
 'Central_Ash Grove MPC_Memphis, TN.pkl': 34.112887949418514,
 'Central_Ash Grove MPC_Muskogee, OK.pkl': 44.655810370855384,
 'Central_Ash Grove MPC_North Little Rock, AR.pkl': 100.0,
 'Central_Ash Grove MPC_Oklahoma City, OK.pkl': 30.967048593388903,
 'Central_Jewell_Austin TX (S).pkl': 39.122600320586734,
 'Central_Jewell_Brittmoore.pkl': 20.50468017595813,
 'Central_Jewell_Dallas TX (S).pkl': 22.532908024065136,
 'Central_Jewell_Frisco TX (S).pkl': 26.138389192554907,
 'Central_Jewell_Houston TX - West Hardy (M).pkl': 100.0,
 'Central_Jewell_Houston TX-N Garden.pkl': 109.12115556137559,
 'Central_Jewell_Hurst TX-SAK.pkl': 22.564799663956045,
 'Central_Jewell_IBC TX-SAK.pkl': 18.92472736995882,
 'Central_Jewell_Katy TX-SAK.pkl': 98.95650509436827,
 'Central_Jewell_Keller TX (S).pkl': 39.55753822180291,
 'Central_Jewell_Marble Falls (SAK).pkl': 18.877120860764258,
 'Central_Jewell_Rosenberg TX.pkl': 100.0,
 'Central_Jewell_Waco TX.pkl': 51.404496177086266,
 'Central_Northfield_Bridgeport MI.pkl': 100.0,
 'Central_Northfield_Cincinnati OH-SAK.pkl': 46.44461891755831,
 'Central_Northfield_Forest View IL.pkl': 72.66966250560128,
 'Central_Northfield_Franklin Park IL-SAK.pkl': 34.27596051615035,
 'Central_Northfield_Indianapolis IN.pkl': 98.8563557991144,
 'Central_Northfield_Miller Materials KC Plant.pkl': 100.0,
 'Central_Northfield_Miller MaterialsBonner Springs.pkl': 100.0,
 'Central_Northfield_Morris IL.pkl': 95.94530048466756,
 'Central_Northfield_Mundelein IL.pkl': 100.0,
 'Central_Northfield_Shakopee.pkl': 100.00101846491572,
```

```
'Central_Northfield_Sheffield OH.pkl': 66.13677575070186,
'Central_Northfield_West Des Moines IA.pkl': 105.11927771413707,
'Central_Northfield_Wisconsin Distribution Yard.pkl': 100.00971371072625,
'East_Adams Products_Anderson SC.pkl': 97.66752109805478,
'East_Adams Products_Asheville NC.pkl': 46024.075075701665,
'East_Adams Products_Charlotte NC Plant.pkl': 23.051880813287294,
'East_Adams Products_Charlotte NC.pkl': 22.623397839886074,
'East_Adams Products_Clarksville TN.pkl': 27.493073325177765,
'East_Adams Products_Colfax NC.pkl': 95.27303263055127,
'East_Adams Products_Cowpens SC.pkl': 105.79780584075186,
'East_Adams Products_Durham NC.pkl': 29.153153304793477,
'East_Adams Products_Fayetteville NC.pkl': 28.902889399218584,
'East_Adams Products_Franklin NC.pkl': 99.78916055722374,
'East_Adams Products_Franklin TN-SAK.pkl': 104.65966977615729,
'East_Adams Products_Goldsboro NC.pkl': 32.963347102458904,
'East_Adams Products_Greensboro NC.pkl': 100.0,
'East_Adams Products_Greenville NC.pkl': 26.478633268274216,
'East_Adams Products_Greenville SC.pkl': 26.811974551504104,
'East_Adams Products_Hickory NC.pkl': 98.4878234726042,
'East_Adams Products_HollyHill SC.pkl': 29.129899980530155,
'East_Adams Products_Inman SC.pkl': 22.516409893721427,
'East_Adams Products_Jacksonville NC.pkl': 23.608215549474473,
'East_Adams Products_Lilesville NC-SAK.pkl': 30.617372305344258,
'East_Adams Products_Morehead NC.pkl': 24.415645624522824,
'East_Adams Products_Morrisville NC.pkl': 101.01064206061858,
'East_Adams Products_Myrtle Beach SC.pkl': 105.27048102574177,
'East_Adams Products_Nashville TN.pkl': 71.41849770961046,
'East_Adams Products_Rockwood TN.pkl': 100.0,
'East_Adams Products_Rocky Mount NC.pkl': 99.78616158011658,
'East_Adams Products_Stallings NC.pkl': 41.22157258599648,
'East_Adams Products_Wilmington NC.pkl': 133.22146025996042,
'East_Adams Products_Youngsville NC.pkl': 100.0,
'East_Anchor_Anchor South Region.pkl': 100.0,
'East_Anchor_Batavia NY-SAK.pkl': 99.43762798077866,
'East_Anchor_Brick NJ.pkl': 98.98800226999953,
'East_Anchor_Bristol PA-SAK.pkl': 100.0,
'East_Anchor_Calverton NY-SAK.pkl': 132.60729825085,
'East_Anchor_Canaan CT-SAK.pkl': 130.3276522308542,
'East_Anchor_Cranston RI.pkl': 110.31586627556342,
'East_Anchor_Crofton MD.pkl': 98.76073853199384,
'East_Anchor_East Petersburg PA.pkl': 97.95066521262498,
'East_Anchor_Easton PA.pkl': 103.02557848459645,
```

```
'East_Anchor_Emigsville PA.pkl': 23.26121356304503,
'East_Anchor_Farmingdale NJ.pkl': 108.99737740741773,
'East_Anchor_Fishers NY.pkl': 98.25499259873138,
'East_Anchor_Fredonia PA-SAK.pkl': 106.0009230561212,
'East_Anchor_Holbrook MA.pkl': 57.05498362698478,
'East_Anchor_Keene NH.pkl': 99.87845231249534,
'East_Anchor_Lebanon NH.pkl': 99.45607789528246,
'East_Anchor_Lyndhurst NJ.pkl': 100.0,
'East_Anchor_Manasquan NJ.pkl': 100.0,
'East_Anchor_Milford VA-SAK.pkl': 100.0,
'East_Anchor_Oxford MA-SAK.pkl': 43.31308732540788,
'East_Anchor_White Marsh MD-SAK.pkl': 113.69752232431726,
'East_Anchor_Winchester VA.pkl': 40.5830018073335,
'East_Georgia Masonry Supply_Cartersville GA BLOCK.pkl': 24.42777555327033,
'East_Georgia Masonry Supply_Conley GA-SAK.pkl': 35.882705736283505,
'East_Georgia Masonry Supply_Florence AL.pkl': 35.40394523797486,
'East_Georgia Masonry Supply_Jasper AL.pkl': 99.99999769676664,
'East_Georgia Masonry Supply_Jonesboro GA.pkl': 100.22939459511963,
'East_Georgia Masonry Supply_Lawrenceville GA DIST.pkl': 28.562445133260866,
'East_Georgia Masonry Supply_Lawrenceville GA MANF.pkl': 25.788291832201804,
'East_Georgia Masonry Supply_Macon GA.pkl': 23.651414988441168,
'East_Georgia Masonry Supply_Montgomery AL.pkl': 101.77825290565627,
'East_Georgia Masonry Supply_Pelham AL.pkl': 117.44292478912202,
'East_Georgia Masonry Supply_Tyrone GA.pkl': 26.153977550897373,
'East_OldcastleCoastal_Auburndale FL-SAK.pkl': 103.2979706188381,
'East_OldcastleCoastal_Defuniak.pkl': 29.38528818748783,
'East_OldcastleCoastal_Fort Myers FL.pkl': 23.69317236423318,
'East_OldcastleCoastal_Fort Pierce FL.pkl': 102.21474299217333,
'East_OldcastleCoastal_Gainesville FL.pkl': 98.41049590049002,
'East_OldcastleCoastal_Gulfport MS.pkl': 33.508206114237964,
'East_OldcastleCoastal_Haines City FL Hardscapes.pkl': 101.13980746499386,
'East_OldcastleCoastal_Jacksonville FL-SAK.pkl': 24.036934879263242,
'East_OldcastleCoastal_Jacksonville FL.pkl': 100.99649969572322,
'East_OldcastleCoastal_Lehigh Acres FL.pkl': 100.0,
'East_OldcastleCoastal_Longwood FL.pkl': 22.362843151522654,
'East_OldcastleCoastal_Orlando FL.pkl': 16.030745604980574,
'East_OldcastleCoastal_Pensacola FL-SAK.pkl': 30.157874206044205,
'East_OldcastleCoastal_Pompano FL Hardscapes.pkl': 123.44227303740647,
'East_OldcastleCoastal_Pompano FL-SAK.pkl': 16.139547536913916,
'East_OldcastleCoastal_Sarasota FL.pkl': 102.58195051779452,
'East_OldcastleCoastal_Tampa FL - Anderson Rd.pkl': 16.50049849914757,
'East_OldcastleCoastal_Tampa FL - Busch Blvd.pkl': 99.98818208408984,
```

```
'East_OldcastleCoastal_Tampa FL-SAK.pkl': 18.91384015251571,
'East_OldcastleCoastal_Theodore AL.pkl': 101.45701426717851,
'East_OldcastleCoastal_West Palm Beach FL.pkl': 15.364590368115765,
'East_OldcastleCoastal_Zephyrhills FL.pkl': 99.9997093542163,
'Lawn and Garden_L&G Central_Aliceville AL.pkl': 81.48293940687725,
'Lawn and Garden_L&G Central_Amherst Junction WI.pkl': 66.46036321324605,
'Lawn and Garden_L&G Central_Bridgeport MI.pkl': 101.26604638448387,
'Lawn and Garden_L&G Central_Cleveland, TX.pkl': 100.09364758704248,
'Lawn and Garden_L&G Central_Dallas TX.pkl': 100.0,
'Lawn and Garden_L&G Central_Del Valle, TX.pkl': 48.82725952146983,
'Lawn and Garden_L&G Central_Harrah OK.pkl': 75.83808666944728,
'Lawn and Garden_L&G Central_Hope AR.pkl': 60.6092021439195,
'Lawn and Garden_L&G Central_Livingston TX.pkl': 99.99985725118438,
'Lawn and Garden_L&G Central_Marseilles IL.pkl': 102.76240541185133,
'Lawn and Garden_L&G Central_Miami OK.pkl': 100.00032626047191,
'Lawn and Garden_L&G Central_Paola KS.pkl': 100.0,
'Lawn and Garden_L&G Central_Powderly TX.pkl': 58.601122873873365,
'Lawn and Garden_L&G Central_Sauget IL.pkl': 67.45553693490703,
'Lawn and Garden_L&G Central_Tulia TX.pkl': 56.187079247518014,
'Lawn and Garden_L&G Central_Tylertown MS.pkl': 98.89571328472938,
'Lawn and Garden_L&G Central_Waterloo IN.pkl': 117.0918107010605,
'Lawn and Garden_L&G Northeast_Berlin NY.pkl': 100.0,
'Lawn and Garden_L&G Northeast_Carey OH.pkl': 44.91671398412932,
'Lawn and Garden_L&G Northeast_Castlewood VA.pkl': 100.0,
'Lawn and Garden_L&G Northeast_Chatsworth GA.pkl': 101.65545185175095,
'Lawn and Garden_L&G Northeast_Historical-Co-Packer.pkl': 90.44361201481476,
'Lawn and Garden_L&G Northeast_Hooksett NH.pkl': 102.83932702984586,
'Lawn and Garden_L&G Northeast_Lee MA.pkl': 45.15681460086673,
'Lawn and Garden_L&G Northeast_Manchester NY.pkl': 100.0,
'Lawn and Garden_L&G Northeast_Mount Hope NJ.pkl': 100.00012738607981,
'Lawn and Garden_L&G Northeast_Poland Spring ME.pkl': 99.99967132056334,
'Lawn and Garden_L&G Northeast_Quakertown PA.pkl': 99.99400806464868,
'Lawn and Garden_L&G Northeast_Thomasville PA.pkl': 99.99749514233368,
'Lawn and Garden_L&G Northeast_Wyoming RI.pkl': 100.0,
'Lawn and Garden_L&G Southeast_Aberdeen NC.pkl': 101.44943459423624,
'Lawn and Garden_L&G Southeast_Bostwick FL.pkl': 100.0,
'Lawn and Garden_L&G Southeast_Cross City FL.pkl': 100.0,
'Lawn and Garden_L&G Southeast_Davenport FL.pkl': 106.20359447503871,
'Lawn and Garden_L&G Southeast_Fort Green FL.pkl': 99.99971010381621,
'Lawn and Garden_L&G Southeast_Gaffney SC.pkl': 100.0,
'Lawn and Garden_L&G Southeast_Louisburg NC.pkl': 100.0,
'Lawn and Garden_L&G Southeast_Moore Haven FL.pkl': 98.78970345565796,
```

```
'Lawn and Garden_L&G Southeast_Pageland SC.pkl': 100.0,
'Lawn and Garden_L&G Southeast_Shady Dale GA.pkl': 100.0,
'Lawn and Garden_L&G Southeast_Walterboro SC.pkl': 100.0,
'National_AMTC_Saint Paul MN.pkl': 61.1385961066768,
'National_Anchor Wall Systems_Minnetonka MN.pkl': 100.20350697742518,
'National_MoistureShield_Springdale AR.pkl': 100.0,
'National_Oldcastle Sakerete Billing_Easy Mix.pkl': 54.20980915958224,
'National_Oldcastle Sakerete Billing_Roberts Concrete.pkl': 66.48305732124939,
'National_Oldcastle Sakerete Billing_US Mix.pkl': 43.4784736681168,
'National_Techniseal_Ash Grove Memphis.pkl': 76.20696143586312,
'National_Techniseal_Ash Grove Nebraska.pkl': 98.96178464379707,
'National_Techniseal_Candiac.pkl': 99.16234750777629,
'National_Techniseal_Coastal Tampa.pkl': 332.05536802598135,
'National_Techniseal_CPM Portland.pkl': 194.89456578973926,
'National_Techniseal_Ectra.pkl': 2050.63071310443,
'National_Techniseal_Eurl Leps Mehat.pkl': 384.5658082019748,
'National_Techniseal_EZ Mix.pkl': 63.82025978162893,
'National_Techniseal_Handy Concrete.pkl': 108.05363490478841,
'National_Techniseal_PTB  Compaktuna.pkl': 161.90722667357292,
'National_Techniseal_Ras.pkl': 248.10574171961682,
'National_Techniseal_Techmix.pkl': 52.47062612771786,
'National_Westile_Westile Roofing Products.pkl': 31.146207878631127,
'West_Amcor_North Salt Lake UT.pkl': 98.22454557179093,
'West_CPM_Frederickson, WA.pkl': 2320.2647702933104,
'West_CPM_Kent, WA.pkl': 102.49553751118567,
'West_CPM_Northstar Consignment.pkl': 312.2972920876084,
'West_CPM_Pasco WA.pkl': 116.11403323452119,
'West_CPM_Portland, OR.pkl': 101.56253206273279,
'West_CPM_Spokane, WA.pkl': 100.0,
'West_Sierra_Fontana CA.pkl': 92.45907005595387,
'West_Sierra_Reno NV.pkl': 96.93968916927118,
'West_Sierra_San Carlos CA.pkl': 615.7309246060131,
'West_Sierra_Stockton CA.pkl': 98.29301536894833,
'West_Superlite_Gilbert, AZ.pkl': 98.7051097053735,
'West_Superlite_Integra Product.pkl': 100.0,
'West_Superlite_Lone Butte.pkl': 99.99993342530978,
'West_Superlite_North Las Vegas.pkl': 100.0,
'West_Superlite_Superlite - Western 19th Ave.pkl': 20.59434988628568,
'West_Superlite_Tucson, AZ - Gardner Ln.pkl': 101.83952541132857,
'West_Superlite_West Phoenix N. 42nd Ave, AZ.pkl': 99.78844048344257}
```

In [243]:
```
count = 0
for name, score in simple_avg_di.items():
    if np.isnan(score):
        count +=1
print('Number of nan scores: ', count)
```

```
Number of nan scores:  0
```

Running SARIMAX Model

First creating our unique pdq and pdqs values for hyperparameter tuning

In [202]:
```
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

print('These are all the possible combinations individually\n')
print(pdq)
print()
print(seasonal_pdq)
print('\n')

print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))
```

```
These are all the possible combinations individually

[(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)]

[(0, 0, 0, 12), (0, 0, 1, 12), (0, 1, 0, 12), (0, 1, 1, 12), (1, 0, 0, 12), (1, 0, 1, 12), (1, 1, 0, 12), (1, 1, 1, 1
2)]


Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

Running our model

In [216]:
```
start = datetime.now()
sarimax_di = sarimax_on_all(good_names_f, good_paths_f, pdq, seasonal_pdq)
print('\nTime taken: ', datetime.now() - start)
```

```
running SARIMAX model with hyperparameter tuning on all files,
this captures seasonality as well as trend
```

```
1 Files---done!
2 Files---done!
3 Files---done!
4 Files---done!
5 Files---done!
6 Files---done!
7 Files---done!
8 Files---done!
9 Files---done!
10 Files---done!
11 Files---done!
12 Files---done!
13 Files---done!
14 Files---done!
15 Files---done!
16 Files---done!
17 Files---done!
18 Files---done!
19 Files---done!
20 Files---done!
21 Files---done!
22 Files---done!
23 Files---done!
24 Files---done!
25 Files---done!
26 Files---done!
27 Files---done!
28 Files---done!
29 Files---done!
30 Files---done!
31 Files---done!
32 Files---done!
33 Files---done!
34 Files---done!
35 Files---done!
36 Files---done!
37 Files---done!
38 Files---done!
```

```
39 Files---done!
40 Files---done!
41 Files---done!
42 Files---done!
43 Files---done!
44 Files---done!
45 Files---done!
46 Files---done!
47 Files---done!
48 Files---done!
49 Files---done!
50 Files---done!
51 Files---done!
52 Files---done!
53 Files---done!
54 Files---done!
55 Files---done!
56 Files---done!
57 Files---done!
58 Files---done!
59 Files---done!
60 Files---done!
61 Files---done!
62 Files---done!
63 Files---done!
64 Files---done!
65 Files---done!
66 Files---done!
67 Files---done!
68 Files---done!
69 Files---done!
70 Files---done!
71 Files---done!
72 Files---done!
73 Files---done!
74 Files---done!
75 Files---done!
76 Files---done!
77 Files---done!
78 Files---done!
79 Files---done!
80 Files---done!
```

```
81 Files---done!
82 Files---done!
83 Files---done!
84 Files---done!
85 Files---done!
86 Files---done!
87 Files---done!
88 Files---done!
89 Files---done!
90 Files---done!
91 Files---done!
92 Files---done!
93 Files---done!
94 Files---done!
95 Files---done!
96 Files---done!
97 Files---done!
98 Files---done!
99 Files---done!
100 Files---done!
101 Files---done!
102 Files---done!
103 Files---done!
104 Files---done!
105 Files---done!
106 Files---done!
107 Files---done!
108 Files---done!
109 Files---done!
110 Files---done!
111 Files---done!
112 Files---done!
113 Files---done!
114 Files---done!
115 Files---done!
116 Files---done!
117 Files---done!
118 Files---done!
119 Files---done!
120 Files---done!
121 Files---done!
122 Files---done!
```

```
123 Files---done!
124 Files---done!
125 Files---done!
126 Files---done!
127 Files---done!
128 Files---done!
129 Files---done!
130 Files---done!
131 Files---done!
132 Files---done!
133 Files---done!
134 Files---done!
135 Files---done!
136 Files---done!
137 Files---done!
138 Files---done!
139 Files---done!
140 Files---done!
141 Files---done!
142 Files---done!
143 Files---done!
144 Files---done!
145 Files---done!
146 Files---done!
147 Files---done!
148 Files---done!
149 Files---done!
150 Files---done!
151 Files---done!
152 Files---done!
153 Files---done!
154 Files---done!
155 Files---done!
156 Files---done!
157 Files---done!
158 Files---done!
159 Files---done!
160 Files---done!
161 Files---done!
162 Files---done!
163 Files---done!
164 Files---done!
```

```
165 Files---done!
166 Files---done!
167 Files---done!
168 Files---done!
169 Files---done!
170 Files---done!
171 Files---done!
172 Files---done!
173 Files---done!
174 Files---done!
175 Files---done!
176 Files---done!
177 Files---done!
178 Files---done!
179 Files---done!
180 Files---done!
181 Files---done!
182 Files---done!
183 Files---done!
184 Files---done!
185 Files---done!
186 Files---done!
187 Files---done!
188 Files---done!
189 Files---done!
190 Files---done!
191 Files---done!
192 Files---done!
193 Files---done!
194 Files---done!
195 Files---done!
196 Files---done!
197 Files---done!
198 Files---done!
199 Files---done!
200 Files---done!
201 Files---done!
202 Files---done!
203 Files---done!
204 Files---done!
205 Files---done!
206 Files---done!
```

```
207 Files---done!

Time taken:  0:11:06.951300
```

In [217]:
```
sarimax_di
```

Out[217]:
```
{'APG ATLANTA_Oldcastle Retail CMP999820_EZ Mix.pkl': 109.93651808174201,
 'Canada_Expocrete_Acheson.pkl': 144.1819844240368,
 'Canada_Expocrete_Balzac.pkl': 314.4469205225893,
 'Canada_Expocrete_Edmonton.pkl': 190.0685387771617,
 'Canada_Expocrete_Richmond.pkl': 78.73791935303535,
 'Canada_Expocrete_Saskatoon.pkl': 1752.5107487497276,
 'Canada_Expocrete_Winnipeg.pkl': 64.36110142058789,
 'Canada_Permacon_Milton ON.pkl': 135.99874662365866,
 'Canada_Permacon_Montreal QC.pkl': 365.09789424532033,
 'Canada_Permacon_Woodstock Ontario.pkl': 2978.696746868404,
 'Central_Ash Grove MPC_Fort Smith, AR.pkl': 34.93662725820034,
 'Central_Ash Grove MPC_Fremont, NE.pkl': 5490.793360197135,
 'Central_Ash Grove MPC_Harrisonville, MO.pkl': 665.1067401357551,
 'Central_Ash Grove MPC_Jackson, MS.pkl': 487.1146736297773,
 'Central_Ash Grove MPC_Memphis, TN.pkl': 31.489738052788713,
 'Central_Ash Grove MPC_Muskogee, OK.pkl': 120.9819317061914,
 'Central_Ash Grove MPC_North Little Rock, AR.pkl': 326.73342908532516,
 'Central_Ash Grove MPC_Oklahoma City, OK.pkl': 28.17529034034666,
 'Central_Jewell_Austin TX (S).pkl': 62.22174538080162,
 'Central_Jewell_Brittmoore.pkl': 20.572382257322673,
 'Central_Jewell_Dallas TX (S).pkl': 27.74688734625076,
 'Central_Jewell_Frisco TX (S).pkl': 38.45670532925165,
 'Central_Jewell_Houston TX - West Hardy (M).pkl': 238.60771326598527,
 'Central_Jewell_Houston TX-N Garden.pkl': 224.6094852123716,
 'Central_Jewell_Hurst TX-SAK.pkl': 18.93714980288836,
 'Central_Jewell_IBC TX-SAK.pkl': 16.83630728688726,
 'Central_Jewell_Katy TX-SAK.pkl': 226.8643069551233,
 'Central_Jewell_Keller TX (S).pkl': 37.70293051078848,
 'Central_Jewell_Marble Falls (SAK).pkl': 35.17659949392116,
 'Central_Jewell_Rosenberg TX.pkl': 90.34990714422648,
 'Central_Jewell_Waco TX.pkl': 34.398706464701064,
 'Central_Northfield_Bridgeport MI.pkl': 296.41184502514625,
 'Central_Northfield_Cincinnati OH-SAK.pkl': 28.806010790797572,
 'Central_Northfield_Forest View IL.pkl': 48.3880667448228,
 'Central_Northfield_Franklin Park IL-SAK.pkl': 24.26233643465519,
 'Central_Northfield_Indianapolis IN.pkl': 186.85360891556036,
 'Central_Northfield_Miller Materials KC Plant.pkl': 1852.3584196736415,
 'Central_Northfield_Miller MaterialsBonner Springs.pkl': 288.3036870517219,
 'Central_Northfield_Morris IL.pkl': 399.36261863262206,
 'Central_Northfield_Mundelein IL.pkl': 301.7877190212824,
 'Central_Northfield_Shakopee.pkl': 791.7684101326377,
```

```
'Central_Northfield_Sheffield OH.pkl': 43.94680397260296,
'Central_Northfield_West Des Moines IA.pkl': 233.94899082753216,
'Central_Northfield_Wisconsin Distribution Yard.pkl': 180.21500160086003,
'East_Adams Products_Anderson SC.pkl': 123.50968658676713,
'East_Adams Products_Asheville NC.pkl': 1519361.7756017018,
'East_Adams Products_Charlotte NC Plant.pkl': 15.228867759029537,
'East_Adams Products_Charlotte NC.pkl': 18.720787188078162,
'East_Adams Products_Clarksville TN.pkl': 29.806924095785796,
'East_Adams Products_Colfax NC.pkl': 258.39376832820176,
'East_Adams Products_Cowpens SC.pkl': 135.34628342409215,
'East_Adams Products_Durham NC.pkl': 21.338261357634916,
'East_Adams Products_Fayetteville NC.pkl': 28.4594175407765,
'East_Adams Products_Franklin NC.pkl': 105.16317063129432,
'East_Adams Products_Franklin TN-SAK.pkl': 99.83737915895513,
'East_Adams Products_Goldsboro NC.pkl': 40.11943142080961,
'East_Adams Products_Greensboro NC.pkl': 162.19527163462536,
'East_Adams Products_Greenville NC.pkl': 23.423877805855614,
'East_Adams Products_Greenville SC.pkl': 20.55964967266661,
'East_Adams Products_Hickory NC.pkl': 76.97633302110971,
'East_Adams Products_HollyHill SC.pkl': 71.50527832429052,
'East_Adams Products_Inman SC.pkl': 20.789483313356378,
'East_Adams Products_Jacksonville NC.pkl': 28.640810058485545,
'East_Adams Products_Lilesville NC-SAK.pkl': 17.3312647336415,
'East_Adams Products_Morehead NC.pkl': 24.52477055741376,
'East_Adams Products_Morrisville NC.pkl': 277.52477724685957,
'East_Adams Products_Myrtle Beach SC.pkl': 361.12792718682937,
'East_Adams Products_Nashville TN.pkl': 203.05459239863123,
'East_Adams Products_Rockwood TN.pkl': 219.00234008682213,
'East_Adams Products_Rocky Mount NC.pkl': 286.1018036398052,
'East_Adams Products_Stallings NC.pkl': 59.80522709985047,
'East_Adams Products_Wilmington NC.pkl': 163.17490632745654,
'East_Adams Products_Youngsville NC.pkl': 426.1551746598653,
'East_Anchor_Anchor South Region.pkl': 696.5559424038474,
'East_Anchor_Batavia NY-SAK.pkl': 84.97900543344686,
'East_Anchor_Brick NJ.pkl': 140.92883733844937,
'East_Anchor_Bristol PA-SAK.pkl': 651.5173795223087,
'East_Anchor_Calverton NY-SAK.pkl': 1246.736846702456,
'East_Anchor_Canaan CT-SAK.pkl': 595.3836701450228,
'East_Anchor_Cranston RI.pkl': 302.1323594057968,
'East_Anchor_Crofton MD.pkl': 190.45344679479012,
'East_Anchor_East Petersburg PA.pkl': 125.54244018070062,
'East_Anchor_Easton PA.pkl': 88.42018081882307,
```

```
'East_Anchor_Emigsville PA.pkl': 40.959834481179975,
'East_Anchor_Farmingdale NJ.pkl': 312.09448779684124,
'East_Anchor_Fishers NY.pkl': 456.43037549584216,
'East_Anchor_Fredonia PA-SAK.pkl': 243.73913957650853,
'East_Anchor_Holbrook MA.pkl': 30.133875276995475,
'East_Anchor_Keene NH.pkl': 106.55773090325704,
'East_Anchor_Lebanon NH.pkl': 184.93448390456248,
'East_Anchor_Lyndhurst NJ.pkl': 181.94263762671287,
'East_Anchor_Manasquan NJ.pkl': 152.41737217526062,
'East_Anchor_Milford VA-SAK.pkl': 427.4408719143663,
'East_Anchor_Oxford MA-SAK.pkl': 28.056116917134617,
'East_Anchor_White Marsh MD-SAK.pkl': 546.8939204225785,
'East_Anchor_Winchester VA.pkl': 26.20801396576699,
'East_Georgia Masonry Supply_Cartersville GA BLOCK.pkl': 43.749577581929756,
'East_Georgia Masonry Supply_Conley GA-SAK.pkl': 16.525061158020062,
'East_Georgia Masonry Supply_Florence AL.pkl': 42.602222384094965,
'East_Georgia Masonry Supply_Jasper AL.pkl': 124.6710558833372,
'East_Georgia Masonry Supply_Jonesboro GA.pkl': 131.09621603455776,
'East_Georgia Masonry Supply_Lawrenceville GA DIST.pkl': 26.702419522567922,
'East_Georgia Masonry Supply_Lawrenceville GA MANF.pkl': 559.1191100813787,
'East_Georgia Masonry Supply_Macon GA.pkl': 35.695176874354395,
'East_Georgia Masonry Supply_Montgomery AL.pkl': 365.0648207426065,
'East_Georgia Masonry Supply_Pelham AL.pkl': 248.85039555502874,
'East_Georgia Masonry Supply_Tyrone GA.pkl': 32.126783244793266,
'East_OldcastleCoastal_Auburndale FL-SAK.pkl': 235.60803157116217,
'East_OldcastleCoastal_Defuniak.pkl': 28.456889370423117,
'East_OldcastleCoastal_Fort Myers FL.pkl': 26.716341053845298,
'East_OldcastleCoastal_Fort Pierce FL.pkl': 421.77852747982075,
'East_OldcastleCoastal_Gainesville FL.pkl': 188.53315311175908,
'East_OldcastleCoastal_Gulfport MS.pkl': 30.256561535624538,
'East_OldcastleCoastal_Haines City FL Hardscapes.pkl': 154.6333866823817,
'East_OldcastleCoastal_Jacksonville FL-SAK.pkl': 20.172252697148974,
'East_OldcastleCoastal_Jacksonville FL.pkl': 181.4423857607238,
'East_OldcastleCoastal_Lehigh Acres FL.pkl': 680.8676322668522,
'East_OldcastleCoastal_Longwood FL.pkl': 25.09451190102463,
'East_OldcastleCoastal_Orlando FL.pkl': 19.498957025453365,
'East_OldcastleCoastal_Pensacola FL-SAK.pkl': 25.06098445435501,
'East_OldcastleCoastal_Pompano FL Hardscapes.pkl': 377.9130518287157,
'East_OldcastleCoastal_Pompano FL-SAK.pkl': 14.807996915502923,
'East_OldcastleCoastal_Sarasota FL.pkl': 130.45484306998566,
'East_OldcastleCoastal_Tampa FL - Anderson Rd.pkl': 18.191524222227606,
'East_OldcastleCoastal_Tampa FL - Busch Blvd.pkl': 128.76463510423432,
```

```
'East_OldcastleCoastal_Tampa FL-SAK.pkl': 18.95846940815152,
'East_OldcastleCoastal_Theodore AL.pkl': 101.26796017028342,
'East_OldcastleCoastal_West Palm Beach FL.pkl': 15.686036144581298,
'East_OldcastleCoastal_Zephyrhills FL.pkl': 294.1994389128125,
'Lawn and Garden_L&G Central_Aliceville AL.pkl': 258.85622096007035,
'Lawn and Garden_L&G Central_Amherst Junction WI.pkl': 86.74340565806801,
'Lawn and Garden_L&G Central_Bridgeport MI.pkl': 109.07558466148825,
'Lawn and Garden_L&G Central_Cleveland, TX.pkl': 328.39100502936765,
'Lawn and Garden_L&G Central_Dallas TX.pkl': 180.27523532270862,
'Lawn and Garden_L&G Central_Del Valle, TX.pkl': 64.47215216443875,
'Lawn and Garden_L&G Central_Harrah OK.pkl': 101.60556537918264,
'Lawn and Garden_L&G Central_Hope AR.pkl': 55.813905859673405,
'Lawn and Garden_L&G Central_Livingston TX.pkl': 181.1337943554691,
'Lawn and Garden_L&G Central_Marseilles IL.pkl': 253.13594384988605,
'Lawn and Garden_L&G Central_Miami OK.pkl': 249.55055234151536,
'Lawn and Garden_L&G Central_Paola KS.pkl': 86.99028254916958,
'Lawn and Garden_L&G Central_Powderly TX.pkl': 55.54774802006895,
'Lawn and Garden_L&G Central_Sauget IL.pkl': 71.411722304266,
'Lawn and Garden_L&G Central_Tulia TX.pkl': 113.2528973624713,
'Lawn and Garden_L&G Central_Tylertown MS.pkl': 391.54988092539196,
'Lawn and Garden_L&G Central_Waterloo IN.pkl': 106.21189409063123,
'Lawn and Garden_L&G Northeast_Berlin NY.pkl': 209.13748158342506,
'Lawn and Garden_L&G Northeast_Carey OH.pkl': 65.0233317977504,
'Lawn and Garden_L&G Northeast_Castlewood VA.pkl': 477.54801116551147,
'Lawn and Garden_L&G Northeast_Chatsworth GA.pkl': 629.9788923510592,
'Lawn and Garden_L&G Northeast_Historical-Co-Packer.pkl': 320.86058031940183,
'Lawn and Garden_L&G Northeast_Hooksett NH.pkl': 96.29499216764816,
'Lawn and Garden_L&G Northeast_Lee MA.pkl': 40.93788741700914,
'Lawn and Garden_L&G Northeast_Manchester NY.pkl': 188.38213444287794,
'Lawn and Garden_L&G Northeast_Mount Hope NJ.pkl': 365.6636103802521,
'Lawn and Garden_L&G Northeast_Poland Spring ME.pkl': 1007.0138788130141,
'Lawn and Garden_L&G Northeast_Quakertown PA.pkl': 1340.3596057372827,
'Lawn and Garden_L&G Northeast_Thomasville PA.pkl': 537.6813086170334,
'Lawn and Garden_L&G Northeast_Wyoming RI.pkl': 132.70875668308858,
'Lawn and Garden_L&G Southeast_Aberdeen NC.pkl': 147.62204398474705,
'Lawn and Garden_L&G Southeast_Bostwick FL.pkl': 235.7339844959026,
'Lawn and Garden_L&G Southeast_Cross City FL.pkl': 344.5168705168408,
'Lawn and Garden_L&G Southeast_Davenport FL.pkl': 371.06451732598435,
'Lawn and Garden_L&G Southeast_Fort Green FL.pkl': 384.24124446115206,
'Lawn and Garden_L&G Southeast_Gaffney SC.pkl': 231.0573665550691,
'Lawn and Garden_L&G Southeast_Louisburg NC.pkl': 281.7384981143307,
'Lawn and Garden_L&G Southeast_Moore Haven FL.pkl': 188.8851186640662,
```

```
'Lawn and Garden_L&G Southeast_Pageland SC.pkl': 212.49538650820548,
'Lawn and Garden_L&G Southeast_Shady Dale GA.pkl': 129.18786307804277,
'Lawn and Garden_L&G Southeast_Walterboro SC.pkl': 283.6102323075451,
'National_AMTC_Saint Paul MN.pkl': 53.889306039220685,
'National_Anchor Wall Systems_Minnetonka MN.pkl': 221.77370918887868,
'National_MoistureShield_Springdale AR.pkl': 1331.0266386365786,
'National_Oldcastle Sakerete Billing_Easy Mix.pkl': 46.75687619120704,
'National_Oldcastle Sakerete Billing_Roberts Concrete.pkl': 34.06017110587195,
'National_Oldcastle Sakerete Billing_US Mix.pkl': 27.03247068672397,
'National_Techniseal_Ash Grove Memphis.pkl': 82.51313808334127,
'National_Techniseal_Ash Grove Nebraska.pkl': 144.7259897365299,
'National_Techniseal_Candiac.pkl': 70.78811544362111,
'National_Techniseal_Coastal Tampa.pkl': 436.47682830441033,
'National_Techniseal_CPM Portland.pkl': 315.5664213797403,
'National_Techniseal_Ectra.pkl': 964.6157699569834,
'National_Techniseal_Eurl Leps Mehat.pkl': 432.21238965092584,
'National_Techniseal_EZ Mix.pkl': 64.2103480817769,
'National_Techniseal_Handy Concrete.pkl': 86.82551019933835,
'National_Techniseal_PTB  Compaktuna.pkl': 134.98578449324862,
'National_Techniseal_Ras.pkl': 299.7250696436543,
'National_Techniseal_Techmix.pkl': 65.65661056599173,
'National_Westile_Westile Roofing Products.pkl': 14.768028882350945,
'West_Amcor_North Salt Lake UT.pkl': 144.1859542822866,
'West_CPM_Frederickson, WA.pkl': 233.76454558151337,
'West_CPM_Kent, WA.pkl': 180.84382732162126,
'West_CPM_Northstar Consignment.pkl': 498.7928818978574,
'West_CPM_Pasco WA.pkl': 296.08568482544206,
'West_CPM_Portland, OR.pkl': 160.64056897300534,
'West_CPM_Spokane, WA.pkl': 220.85401125470776,
'West_Sierra_Fontana CA.pkl': 155.62329210177543,
'West_Sierra_Reno NV.pkl': 254.12740348845935,
'West_Sierra_San Carlos CA.pkl': 1409.0150239320972,
'West_Sierra_Stockton CA.pkl': 149.84809551428677,
'West_Superlite_Gilbert, AZ.pkl': 138.83493772437595,
'West_Superlite_Integra Product.pkl': 225.69884968155742,
'West_Superlite_Lone Butte.pkl': 124.53492085477818,
'West_Superlite_North Las Vegas.pkl': 345.46684718220865,
'West_Superlite_Superlite - Western 19th Ave.pkl': 26.167161700921515,
'West_Superlite_Tucson, AZ - Gardner Ln.pkl': 387.2227251811518,
'West_Superlite_West Phoenix N. 42nd Ave, AZ.pkl': 235.19426061571008}
```

In [212]:
```
count = 0
for name, score in sarimax_di.items():
    if np.isnan(score):
        count +=1
print('Number of nan scores: ', count)
```

```
Number of nan scores:  0
```

Running Holt-Winters Model

In [130]:
```
start = datetime.now()
holt_winters_di = hw_on_all(good_names_f, good_paths_f)
print('\nTime taken: ', datetime.now() - start)
```

```
Time taken:  0:00:42.880684
```

In [211]:
```
count = 0
for name, score in holt_winters_di.items():
    if np.isnan(score):
        count +=1
print('Number of nan scores: ', count)
```

```
Number of nan scores:  0
```

In [132]:
```
holt_winters_di
```

Out[132]:
```
{'APG ATLANTA_Oldcastle Retail CMP999820_EZ Mix.pkl': 93.85905539093197,
 'Canada_Expocrete_Acheson.pkl': 119.35046429665404,
 'Canada_Expocrete_Balzac.pkl': 313.99468484705125,
 'Canada_Expocrete_Edmonton.pkl': 165.96320651592487,
 'Canada_Expocrete_Richmond.pkl': 121.01455149755584,
 'Canada_Expocrete_Saskatoon.pkl': 2128.1807780172153,
 'Canada_Expocrete_Winnipeg.pkl': 58.56260330095482,
 'Canada_Permacon_Milton ON.pkl': 174.39261769667988,
 'Canada_Permacon_Montreal QC.pkl': 1916.2283949193302,
 'Canada_Permacon_Woodstock Ontario.pkl': 16457.96499144497,
 'Central_Ash Grove MPC_Fort Smith, AR.pkl': 30.480140616536055,
 'Central_Ash Grove MPC_Fremont, NE.pkl': 2088.639020443927,
 'Central_Ash Grove MPC_Harrisonville, MO.pkl': 312.25521035758493,
 'Central_Ash Grove MPC_Jackson, MS.pkl': 767.0060635479747,
 'Central_Ash Grove MPC_Memphis, TN.pkl': 38.11704457953739,
 'Central_Ash Grove MPC_Muskogee, OK.pkl': 40.39735375922514,
 'Central_Ash Grove MPC_North Little Rock, AR.pkl': 254.0187091720863,
 'Central_Ash Grove MPC_Oklahoma City, OK.pkl': 34.50526607415018,
 'Central_Jewell_Austin TX (S).pkl': 57.90071760993032,
 'Central_Jewell_Brittmoore.pkl': 24.152318059317633,
 'Central_Jewell_Dallas TX (S).pkl': 22.802692341913477,
 'Central_Jewell_Frisco TX (S).pkl': 48.889117458106455,
 'Central_Jewell_Houston TX - West Hardy (M).pkl': 133.27738931839508,
 'Central_Jewell_Houston TX-N Garden.pkl': 287.56557022329355,
 'Central_Jewell_Hurst TX-SAK.pkl': 23.074740588603753,
 'Central_Jewell_IBC TX-SAK.pkl': 21.662673752716085,
 'Central_Jewell_Katy TX-SAK.pkl': 166.976777551913,
 'Central_Jewell_Keller TX (S).pkl': 43.54026007177998,
 'Central_Jewell_Marble Falls (SAK).pkl': 19.86620399068628,
 'Central_Jewell_Rosenberg TX.pkl': 120.71640933009846,
 'Central_Jewell_Waco TX.pkl': 67.97911493424482,
 'Central_Northfield_Bridgeport MI.pkl': 144.79932652666352,
 'Central_Northfield_Cincinnati OH-SAK.pkl': 53.620161646561314,
 'Central_Northfield_Forest View IL.pkl': 98.6449579673137,
 'Central_Northfield_Franklin Park IL-SAK.pkl': 44.017142060612954,
 'Central_Northfield_Indianapolis IN.pkl': 205.84305295843936,
 'Central_Northfield_Miller Materials KC Plant.pkl': 889.6720248221985,
 'Central_Northfield_Miller MaterialsBonner Springs.pkl': 304.3149242546115,
 'Central_Northfield_Morris IL.pkl': 198.55782283281422,
 'Central_Northfield_Mundelein IL.pkl': 265.868026510729,
 'Central_Northfield_Shakopee.pkl': 1173.2034782964495,
```

'Central_Northfield_Sheffield OH.pkl': 197.02101146128112,
'Central_Northfield_West Des Moines IA.pkl': 250.8705388023314,
'Central_Northfield_Wisconsin Distribution Yard.pkl': 120.92145140560382,
'East_Adams Products_Anderson SC.pkl': 157.17058985119345,
'East_Adams Products_Asheville NC.pkl': 929371.1017800604,
'East_Adams Products_Charlotte NC Plant.pkl': 24.386922251734713,
'East_Adams Products_Charlotte NC.pkl': 40.8658235863145,
'East_Adams Products_Clarksville TN.pkl': 35.06549376381989,
'East_Adams Products_Colfax NC.pkl': 185.8316229770227,
'East_Adams Products_Cowpens SC.pkl': 341.1915305658555,
'East_Adams Products_Durham NC.pkl': 29.551684990033973,
'East_Adams Products_Fayetteville NC.pkl': 30.242153580627186,
'East_Adams Products_Franklin NC.pkl': 82.73159949183567,
'East_Adams Products_Franklin TN-SAK.pkl': 113.28091588438059,
'East_Adams Products_Goldsboro NC.pkl': 38.394166193838814,
'East_Adams Products_Greensboro NC.pkl': 102.63913103535451,
'East_Adams Products_Greenville NC.pkl': 33.20707113909144,
'East_Adams Products_Greenville SC.pkl': 26.689758674122654,
'East_Adams Products_Hickory NC.pkl': 128.4071138890912,
'East_Adams Products_HollyHill SC.pkl': 51.2501541341706,
'East_Adams Products_Inman SC.pkl': 26.423964274197388,
'East_Adams Products_Jacksonville NC.pkl': 30.01868147636283,
'East_Adams Products_Lilesville NC-SAK.pkl': 32.816285291463366,
'East_Adams Products_Morehead NC.pkl': 28.71139025191065,
'East_Adams Products_Morrisville NC.pkl': 166.95275350022973,
'East_Adams Products_Myrtle Beach SC.pkl': 315.27191347220725,
'East_Adams Products_Nashville TN.pkl': 62.07611470208617,
'East_Adams Products_Rockwood TN.pkl': 250.20217404940448,
'East_Adams Products_Rocky Mount NC.pkl': 166.58916918644536,
'East_Adams Products_Stallings NC.pkl': 46.457416343771115,
'East_Adams Products_Wilmington NC.pkl': 139.87478526110314,
'East_Adams Products_Youngsville NC.pkl': 156.5916033229734,
'East_Anchor_Anchor South Region.pkl': 349.2803679873529,
'East_Anchor_Batavia NY-SAK.pkl': 101.54565700740625,
'East_Anchor_Brick NJ.pkl': 120.3361547967182,
'East_Anchor_Bristol PA-SAK.pkl': 174.56070763947764,
'East_Anchor_Calverton NY-SAK.pkl': 1764.3453353538303,
'East_Anchor_Canaan CT-SAK.pkl': 1326.4049449874615,
'East_Anchor_Cranston RI.pkl': 247.9355804416351,
'East_Anchor_Crofton MD.pkl': 162.84038170845213,
'East_Anchor_East Petersburg PA.pkl': 164.28008453208167,
'East_Anchor_Easton PA.pkl': 192.50185139584096,

'East_Anchor_Emigsville PA.pkl': 21.26391699668475,
'East_Anchor_Farmingdale NJ.pkl': 245.39018785432023,
'East_Anchor_Fishers NY.pkl': 509.27061541638903,
'East_Anchor_Fredonia PA-SAK.pkl': 111.74572657058907,
'East_Anchor_Holbrook MA.pkl': 120.29121915145873,
'East_Anchor_Keene NH.pkl': 292.2746142583223,
'East_Anchor_Lebanon NH.pkl': 294.12364894315965,
'East_Anchor_Lyndhurst NJ.pkl': 95.52719731882337,
'East_Anchor_Manasquan NJ.pkl': 112.37901369610763,
'East_Anchor_Milford VA-SAK.pkl': 213.30557641856575,
'East_Anchor_Oxford MA-SAK.pkl': 43.93857236370797,
'East_Anchor_White Marsh MD-SAK.pkl': 117.70281312627797,
'East_Anchor_Winchester VA.pkl': 43.007005681154894,
'East_Georgia Masonry Supply_Cartersville GA BLOCK.pkl': 32.599129401942456,
'East_Georgia Masonry Supply_Conley GA-SAK.pkl': 36.662422425815706,
'East_Georgia Masonry Supply_Florence AL.pkl': 92.57256145494782,
'East_Georgia Masonry Supply_Jasper AL.pkl': 98.55454068225514,
'East_Georgia Masonry Supply_Jonesboro GA.pkl': 146.96316480270502,
'East_Georgia Masonry Supply_Lawrenceville GA DIST.pkl': 31.08752485023047,
'East_Georgia Masonry Supply_Lawrenceville GA MANF.pkl': 29.008173508193703,
'East_Georgia Masonry Supply_Macon GA.pkl': 29.666375024014414,
'East_Georgia Masonry Supply_Montgomery AL.pkl': 111.16559563410706,
'East_Georgia Masonry Supply_Pelham AL.pkl': 450.4285189415662,
'East_Georgia Masonry Supply_Tyrone GA.pkl': 27.309693723579752,
'East_OldcastleCoastal_Auburndale FL-SAK.pkl': 155.51986593930448,
'East_OldcastleCoastal_Defuniak.pkl': 27.215327068415945,
'East_OldcastleCoastal_Fort Myers FL.pkl': 27.432246997304365,
'East_OldcastleCoastal_Fort Pierce FL.pkl': 133.47280757070118,
'East_OldcastleCoastal_Gainesville FL.pkl': 171.1090170806272,
'East_OldcastleCoastal_Gulfport MS.pkl': 39.982007469093176,
'East_OldcastleCoastal_Haines City FL Hardscapes.pkl': 171.83609216126627,
'East_OldcastleCoastal_Jacksonville FL-SAK.pkl': 25.385785139398497,
'East_OldcastleCoastal_Jacksonville FL.pkl': 151.09512614044337,
'East_OldcastleCoastal_Lehigh Acres FL.pkl': 341.3947154838352,
'East_OldcastleCoastal_Longwood FL.pkl': 24.245645695692524,
'East_OldcastleCoastal_Orlando FL.pkl': 19.908929621681327,
'East_OldcastleCoastal_Pensacola FL-SAK.pkl': 29.44011510614079,
'East_OldcastleCoastal_Pompano FL Hardscapes.pkl': 342.7733995401319,
'East_OldcastleCoastal_Pompano FL-SAK.pkl': 17.176012826942156,
'East_OldcastleCoastal_Sarasota FL.pkl': 127.75519519704194,
'East_OldcastleCoastal_Tampa FL - Anderson Rd.pkl': 21.814463308779356,
'East_OldcastleCoastal_Tampa FL - Busch Blvd.pkl': 122.16310863959241,

'East_OldcastleCoastal_Tampa FL-SAK.pkl': 19.948774952702294,
'East_OldcastleCoastal_Theodore AL.pkl': 111.41694498091192,
'East_OldcastleCoastal_West Palm Beach FL.pkl': 18.051980719751683,
'East_OldcastleCoastal_Zephyrhills FL.pkl': 156.82798586993366,
'Lawn and Garden_L&G Central_Aliceville AL.pkl': 171.66963854572285,
'Lawn and Garden_L&G Central_Amherst Junction WI.pkl': 353.04497456405164,
'Lawn and Garden_L&G Central_Bridgeport MI.pkl': 220.5823294747309,
'Lawn and Garden_L&G Central_Cleveland, TX.pkl': 169.14582386165912,
'Lawn and Garden_L&G Central_Dallas TX.pkl': 173.7342648346988,
'Lawn and Garden_L&G Central_Del Valle, TX.pkl': 48.59199659094988,
'Lawn and Garden_L&G Central_Harrah OK.pkl': 253.50950511443813,
'Lawn and Garden_L&G Central_Hope AR.pkl': 225.97348601833883,
'Lawn and Garden_L&G Central_Livingston TX.pkl': 113.66065850825989,
'Lawn and Garden_L&G Central_Marseilles IL.pkl': 143.8753829814954,
'Lawn and Garden_L&G Central_Miami OK.pkl': 524.9899848933846,
'Lawn and Garden_L&G Central_Paola KS.pkl': 336.91450072347453,
'Lawn and Garden_L&G Central_Powderly TX.pkl': 202.83406741791688,
'Lawn and Garden_L&G Central_Sauget IL.pkl': 259.14244196187326,
'Lawn and Garden_L&G Central_Tulia TX.pkl': 98.27702155781417,
'Lawn and Garden_L&G Central_Tylertown MS.pkl': 351.9529740257661,
'Lawn and Garden_L&G Central_Waterloo IN.pkl': 384.1217783198407,
'Lawn and Garden_L&G Northeast_Berlin NY.pkl': 280.6761895596563,
'Lawn and Garden_L&G Northeast_Carey OH.pkl': 66.13133007124841,
'Lawn and Garden_L&G Northeast_Castlewood VA.pkl': 383.2663521572013,
'Lawn and Garden_L&G Northeast_Chatsworth GA.pkl': 261.334730254544,
'Lawn and Garden_L&G Northeast_Historical-Co-Packer.pkl': 3386.042406929486,
'Lawn and Garden_L&G Northeast_Hooksett NH.pkl': 560.9469672981429,
'Lawn and Garden_L&G Northeast_Lee MA.pkl': 47.50731098965178,
'Lawn and Garden_L&G Northeast_Manchester NY.pkl': 354.6640254902569,
'Lawn and Garden_L&G Northeast_Mount Hope NJ.pkl': 220.07138112600333,
'Lawn and Garden_L&G Northeast_Poland Spring ME.pkl': 654.6661957033944,
'Lawn and Garden_L&G Northeast_Quakertown PA.pkl': 3935.608766411267,
'Lawn and Garden_L&G Northeast_Thomasville PA.pkl': 645.7208255718726,
'Lawn and Garden_L&G Northeast_Wyoming RI.pkl': 671.0708031230572,
'Lawn and Garden_L&G Southeast_Aberdeen NC.pkl': 134.10380984400663,
'Lawn and Garden_L&G Southeast_Bostwick FL.pkl': 87.42880215688001,
'Lawn and Garden_L&G Southeast_Cross City FL.pkl': 193.9466444095789,
'Lawn and Garden_L&G Southeast_Davenport FL.pkl': 185.545562456409,
'Lawn and Garden_L&G Southeast_Fort Green FL.pkl': 145.16503395572272,
'Lawn and Garden_L&G Southeast_Gaffney SC.pkl': 317.8666066446932,
'Lawn and Garden_L&G Southeast_Louisburg NC.pkl': 327.6702524038856,
'Lawn and Garden_L&G Southeast_Moore Haven FL.pkl': 80.01529384140485,

```
'Lawn and Garden_L&G Southeast_Pageland SC.pkl': 266.2069633407753,
'Lawn and Garden_L&G Southeast_Shady Dale GA.pkl': 127.37094080613547,
'Lawn and Garden_L&G Southeast_Walterboro SC.pkl': 281.7974702308582,
'National_AMTC_Saint Paul MN.pkl': 45.62618922538426,
'National_Anchor Wall Systems_Minnetonka MN.pkl': 153.0084708500616,
'National_MoistureShield_Springdale AR.pkl': 481.68878533093897,
'National_Oldcastle Sakerete Billing_Easy Mix.pkl': 50.10932559249597,
'National_Oldcastle Sakerete Billing_Roberts Concrete.pkl': 72.66272463872366,
'National_Oldcastle Sakerete Billing_US Mix.pkl': 44.06943887538193,
'National_Techniseal_Ash Grove Memphis.pkl': 121.23281074082442,
'National_Techniseal_Ash Grove Nebraska.pkl': 212.92917052718389,
'National_Techniseal_Candiac.pkl': 159.39187999672924,
'National_Techniseal_Coastal Tampa.pkl': 324.75692324050607,
'National_Techniseal_CPM Portland.pkl': 283.74206702120574,
'National_Techniseal_Ectra.pkl': 1581.252481801131,
'National_Techniseal_Eurl Leps Mehat.pkl': 262.7640934630769,
'National_Techniseal_EZ Mix.pkl': 50.955654735940705,
'National_Techniseal_Handy Concrete.pkl': 102.80998981562044,
'National_Techniseal_PTB  Compaktuna.pkl': 116.22389149982415,
'National_Techniseal_Ras.pkl': 164.20411923180336,
'National_Techniseal_Techmix.pkl': 158.0011427901422,
'National_Westile_Westile Roofing Products.pkl': 33.09242372597726,
'West_Amcor_North Salt Lake UT.pkl': 281.38541700231764,
'West_CPM_Frederickson, WA.pkl': 2454.1892895323094,
'West_CPM_Kent, WA.pkl': 174.58930421310927,
'West_CPM_Northstar Consignment.pkl': 622.5127604638815,
'West_CPM_Pasco WA.pkl': 398.67058413772327,
'West_CPM_Portland, OR.pkl': 126.10867504121191,
'West_CPM_Spokane, WA.pkl': 160.7340095584321,
'West_Sierra_Fontana CA.pkl': 132.2442668089304,
'West_Sierra_Reno NV.pkl': 110.09994841406343,
'West_Sierra_San Carlos CA.pkl': 938.5229513505541,
'West_Sierra_Stockton CA.pkl': 170.00226901387143,
'West_Superlite_Gilbert, AZ.pkl': 85.08110075416826,
'West_Superlite_Integra Product.pkl': 182.45439197509407,
'West_Superlite_Lone Butte.pkl': 90.34668813870861,
'West_Superlite_North Las Vegas.pkl': 241.91400535922907,
'West_Superlite_Superlite - Western 19th Ave.pkl': 32.47282575570401,
'West_Superlite_Tucson, AZ - Gardner Ln.pkl': 241.61665984248532,
'West_Superlite_West Phoenix N. 42nd Ave, AZ.pkl': 209.73199323047896}
```

Storing all our model dictionaries for future use

In [244]:
```
# # storing our results into a pickle file

# with open('Holts-Winters_di.pkl', 'wb') as f:
#     pickle.dump(holt_winters_di, f)

# with open('SARIMAX_di.pkl', 'wb') as f:
#     pickle.dump(sarimax_di, f)

# with open('Exponential_weighted_avg_di.pkl', 'wb') as f:
#     pickle.dump(simple_avg_di, f)
```

Observations

- By just comparing the dictionaries of our Log Transformed and Lag/Differencing transformed Models we can see that the latter is performing more poorly as compared to the first one
- We can certainly conclude that Log Transformation is the right choice for our data
- Another important observation is that we cannot truly rely on the p-value of our stationarity tests and have to try multiple transformations methods for evaluating the model performance

Important Questions Answered

- Why does log transformation work ?
  - The log transformation can be used to make highly skewed distributions less skewed. This can be valuable both for making patterns in the data more interpretable and for helping to meet the assumptions of inferential statistics.
  - So inturn it helps our Model learn better
- Why doesn't Differencing/Lag Transformation work ?
  - The reason is pretty simple, while differencing we are just looking at the perfectly correlated lag point
  - Now, this doesn't truly transform our data or help with skewness of data
  - It only removes the trend and stationarity from the data so that our Models are able to learn them
  - Models like SARIMAX and Holt-Winters capture trend and seasonaltiy of our data
  - Another main reason is that, we are judging our data stationarity based on the p-value of our stationarity test such as Dickey-Fuller test instead of looking at it's graphs and concluding the stationarity
- What can we learn from our Model performances ?
  - We can first compare our model performances for each data and select the best model
  - Now since we have a best model for each "Region Department Facility" combination we can always use the same best model for the specific combinations
  - Looking the best model we will also know if the data generated by the unique combination comprises of any trend or seasonality, cyclic etc for ex: if the best model is SARIMAX, we know that this model captures Trend and Seasonality so surely the product of the unqiue combination is generating sales based on seasons and is affected by various trends
  - This will help us to do better Market Research on such product of the unique combinations
  - It will save us time, we don't keep testing models on the same unique combinations again and again, we will now know which model works best for a specific unique combination and will directly use it for that data in the coming future
- What can we do about data loss?
  - Due to data loss we are keeping thresholds to find our good unique combinations of data
  - We can do past forecasting just like how we do future forecasting using our best model for our specific unqiue combination
  - This will help us in restoring data
  - But, we must make proper research and try to find the reason for data loss, it can be due to closing of the company or maybe that particular unqiue combination didn't even start making products in the past
  - If the reason is just sheer mishap of data then we can apply past forecasting just like future forecasting to restore our valuable data

Note

- LSTM Models have been run in the LSTM.ipynb Notebook, you can check the model performance and code there
- Results.ipynb Notebook contains all the comparisons of our model performances and the final conclusion