slides_sales_forecasting_spark_vf

August 8, 2022

0.1 # Sales Forecasting

0.1.1 Group 16: MIS 381N Project, Summer 2022

Group Members: Prathmesh Savale, Juwon Lee, Bolun Zhang, Vivek Dhullipala

- **Problem statement**: Given the daily sales across stores and product families, forecast the future daily sales at store-product level
- **What do we want to answer**: How can the retail industry utilize machine learning for budgeting, inventory management or financial planning
- **Data set**: Store Sales - Time Series Forecasting

0.2 Importing required libraries

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
%matplotlib inline

import statsmodels as sm
import statsmodels.api as sm_api

from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.stats.diagnostic import acorr_ljungbox
from statsmodels.tsa.seasonal import seasonal_decompose

from sklearn.metrics import mean_absolute_percentage_error,␣
↪mean_absolute_error, mean_squared_error

import contextlib
import warnings

import copy
try:
    from google.colab import drive
    drive.mount('/content/drive')
```

```
except:
    pass

from datetime import date, timedelta
import datetime

from tslearn.clustering import TimeSeriesKMeans
# try:
#   !pip install pystan~=2.14
#   !pip install fbprophet
# except:
#   pass
```

[2]:
```
# filter simple warnings
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

[3]:
```
import os
os.getcwd()
```

[3]:
```
'/home/praths/notebooks/MIS/group_project/MIS_group_project'
```

0.3 ## Reading Datasets

- reading train, test, oil, stores, transactions and holiday data

[4]:
```
train = pd.read_csv('/home/praths/notebooks/MIS/group_project/MIS_group_project/
 ↪train.csv')
test = pd.read_csv('/home/praths/notebooks/MIS/group_project/MIS_group_project/
 ↪test.csv')
oil = pd.read_csv('/home/praths/notebooks/MIS/group_project/MIS_group_project/
 ↪oil.csv')
stores = pd.read_csv('/home/praths/notebooks/MIS/group_project/
 ↪MIS_group_project/stores.csv')
transactions = pd.read_csv('/home/praths/notebooks/MIS/group_project/
 ↪MIS_group_project/transactions.csv')
holiday_events = pd.read_csv('/home/praths/notebooks/MIS/group_project/
 ↪MIS_group_project/holidays_events.csv')
```

0.4 ## Data preprocessing

- checking datasets that can be used for analysis and modelling
- checking NULL values across all datasets
- treating NULL values
- Joining train and test datasets with

```
[5]: train_backup = copy.deepcopy(train)
     test_backup = copy.deepcopy(test)
```

```
[6]: print(train.shape)
     train.head(3)
```

```
(3000888, 6)

[6]:    id        date  store_nbr      family  sales  onpromotion
     0   0  2013-01-01          1  AUTOMOTIVE    0.0            0
     1   1  2013-01-01          1   BABY CARE    0.0            0
     2   2  2013-01-01          1      BEAUTY    0.0            0
```

```
[7]: print(test.shape)
     test.head(3)
```

```
(28512, 5)

[7]:        id        date  store_nbr      family  onpromotion
     0  3000888  2017-08-16          1  AUTOMOTIVE            0
     1  3000889  2017-08-16          1   BABY CARE            0
     2  3000890  2017-08-16          1      BEAUTY            2
```

```
[8]: # train data is present for 4 years from 2013-2017 and test data is present for␣
      ↪15 days
     min(train['date']), max(train['date']), min(test['date']), max(test['date'])
```

```
[8]: ('2013-01-01', '2017-08-15', '2017-08-16', '2017-08-31')
```

```
[9]: # There are 54 different sotes and 33 product families in the train dataset
     train['family'].value_counts().shape, train['store_nbr'].value_counts().shape
```

```
[9]: ((33,), (54,))
```

```
[10]: train['family'].unique(), train['store_nbr'].unique()
```

```
[10]: (array(['AUTOMOTIVE', 'BABY CARE', 'BEAUTY', 'BEVERAGES', 'BOOKS',
              'BREAD/BAKERY', 'CELEBRATION', 'CLEANING', 'DAIRY', 'DELI', 'EGGS',
              'FROZEN FOODS', 'GROCERY I', 'GROCERY II', 'HARDWARE',
              'HOME AND KITCHEN I', 'HOME AND KITCHEN II', 'HOME APPLIANCES',
              'HOME CARE', 'LADIESWEAR', 'LAWN AND GARDEN', 'LINGERIE',
              'LIQUOR,WINE,BEER', 'MAGAZINES', 'MEATS', 'PERSONAL CARE',
              'PET SUPPLIES', 'PLAYERS AND ELECTRONICS', 'POULTRY',
              'PREPARED FOODS', 'PRODUCE', 'SCHOOL AND OFFICE SUPPLIES',
              'SEAFOOD'], dtype=object),
       array([ 1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,  2, 20, 21, 22, 23, 24,
              25, 26, 27, 28, 29,  3, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,  4,
       40, 41, 42, 43, 44, 45, 46, 47, 48, 49,  5, 50, 51, 52, 53, 54,  6,
        7,  8,  9]))
```

Adding holiday and oil price information to the train data

[11]:
```
# adding holiday flag if there is a holiday present on that day
holiday_events['holiday_present'] = 1
holiday_events.rename(columns={'type' : 'holiday_type'}, inplace=True)
holiday_events_df = holiday_events[['date','holiday_present','holiday_type']]

# removing id column as it will not be used in predictions
train_id_dropped = train.drop('id', axis=1)
train_df = pd.merge(train_id_dropped, holiday_events, on='date', how='left')
train_df.loc[train_df['holiday_present'].isna(), 'holiday_type'] = 'not_holiday'
train_df.loc[train_df['holiday_present'].isna(), 'holiday_present'] = 0
```

[12]:
```
# merging oil data with train dataset
oil = oil.fillna(method='bfill')
train_holiday_oil_df = pd.merge(train_df, oil, on='date', how='left')
train_holiday_oil_df['dcoilwtico'].fillna(method='bfill', inplace=True)
```

[13]:
```
train_holiday_oil_df.head(5)
```

[13]:
```
         date  store_nbr      family  sales  onpromotion holiday_type  \
0  2013-01-01          1  AUTOMOTIVE    0.0            0      Holiday
1  2013-01-01          1   BABY CARE    0.0            0      Holiday
2  2013-01-01          1      BEAUTY    0.0            0      Holiday
3  2013-01-01          1   BEVERAGES    0.0            0      Holiday
4  2013-01-01          1       BOOKS    0.0            0      Holiday

     locale locale_name         description transferred  holiday_present  \
0  National     Ecuador  Primer dia del ano       False              1.0
1  National     Ecuador  Primer dia del ano       False              1.0
2  National     Ecuador  Primer dia del ano       False              1.0
3  National     Ecuador  Primer dia del ano       False              1.0
4  National     Ecuador  Primer dia del ano       False              1.0

   dcoilwtico
0       93.14
1       93.14
2       93.14
3       93.14
4       93.14
```

checking NULL values

```
[14]: # locale_name, description, transferred holiday etc. are ignored as they are␣
       ↪not considered for our analysis
      train_holiday_oil_df.isnull().sum()
```

```
[14]: date                     0
      store_nbr                0
      family                   0
      sales                    0
      onpromotion              0
      holiday_type             0
      locale             2551824
      locale_name        2551824
      description        2551824
      transferred        2551824
      holiday_present          0
      dcoilwtico               0
      dtype: int64
```

```
[15]: # subset specific columns and ready data for EDA
      train_holiday_oil_df = train_holiday_oil_df[['date', 'store_nbr', 'family',␣
       ↪'sales', 'onpromotion', \
                                                   'holiday_type', 'holiday_present',␣
       ↪'dcoilwtico']]

      train_holiday_oil_df.head(5)
```

```
[15]:         date  store_nbr      family  sales  onpromotion holiday_type  \
      0  2013-01-01          1  AUTOMOTIVE    0.0            0      Holiday
      1  2013-01-01          1   BABY CARE    0.0            0      Holiday
      2  2013-01-01          1      BEAUTY    0.0            0      Holiday
      3  2013-01-01          1   BEVERAGES    0.0            0      Holiday
      4  2013-01-01          1       BOOKS    0.0            0      Holiday

         holiday_present  dcoilwtico
      0              1.0       93.14
      1              1.0       93.14
      2              1.0       93.14
      3              1.0       93.14
      4              1.0       93.14
```

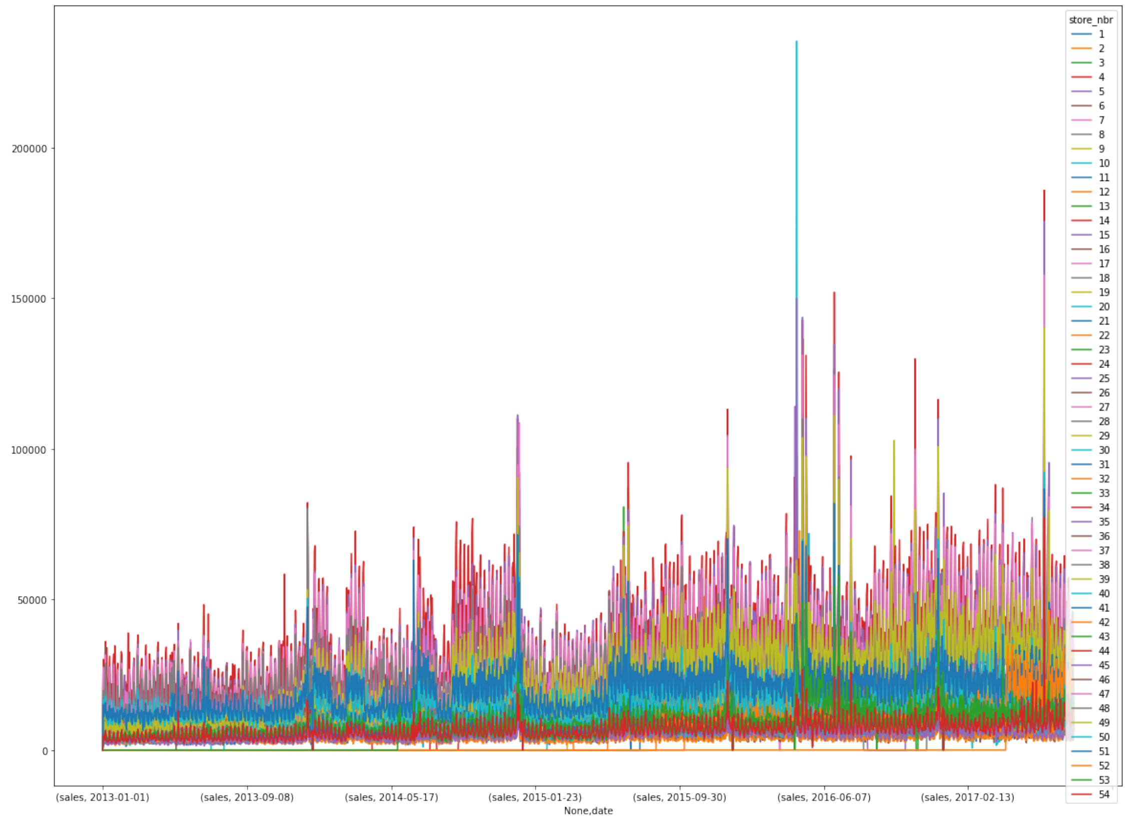
0.5 ## Exploratory data analysis

- Checking time series trends across stores and product families
- Impact of oil prices on sales
- Impact of promotions and holidays on sales

Time series for Stores

[16]:
```
# sales across stores seem to have similar trends at a daily level with␣
↪occasional spikes
train_holiday_oil_df[['sales', 'store_nbr', 'date']].
↪groupby(['store_nbr','date']).agg({'sales':'sum'})\
.unstack().transpose().plot(figsize=(20,15))
```

[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2cabc8efd0>
```



Checking if certain stores have similar time series shapes using dynamic time warping

[17]:
```
#aggregating data at store-date level
stores_ts_sales = train_holiday_oil_df[['sales', 'store_nbr', 'date']].
↪groupby(['store_nbr','date']).agg({'sales':'sum'})\
.unstack()
```

[18]:
```
%%time
ts_clust_fit_stores = TimeSeriesKMeans(n_clusters=3, metric="dtw",
                          max_iter=10, random_state=0).fit(stores_ts_sales)
```

```
CPU times: user 2min 59s, sys: 1min 11s, total: 4min 11s
Wall time: 2min 19s
```

[19]:
```
%%time
predicted_ts_clusters = ts_clust_fit_stores.predict(stores_ts_sales)
```

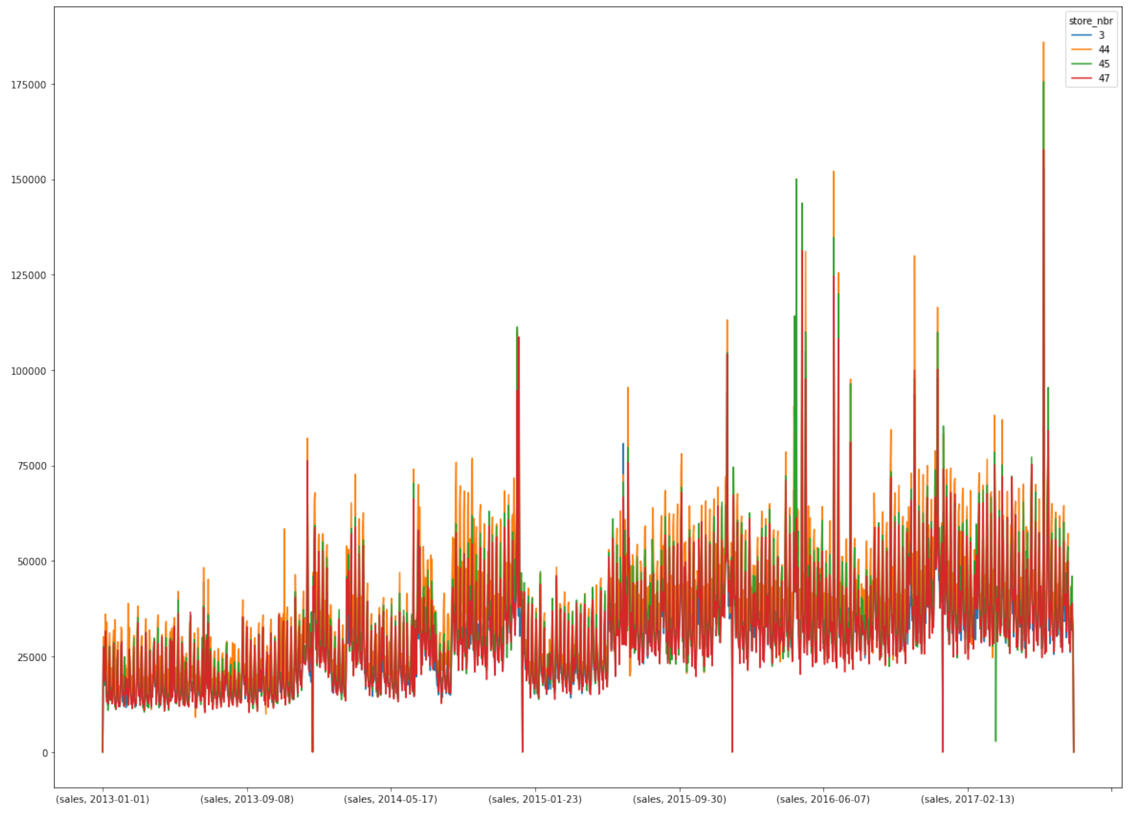
```
CPU times: user 7.08 s, sys: 0 ns, total: 7.08 s
Wall time: 7.07 s
```

[20]:
```
clusters_df_stores = pd.Series(predicted_ts_clusters, index=stores_ts_sales.
 ↪index, name='cluster')
cluster_stores_ts = stores_ts_sales.merge(clusters_df_stores.to_frame(),␣
 ↪left_index=True, right_index=True)
cluster_stores_ts['cluster'].value_counts()
```

[20]:
```
1    47
0     4
2     3
Name: cluster, dtype: int64
```

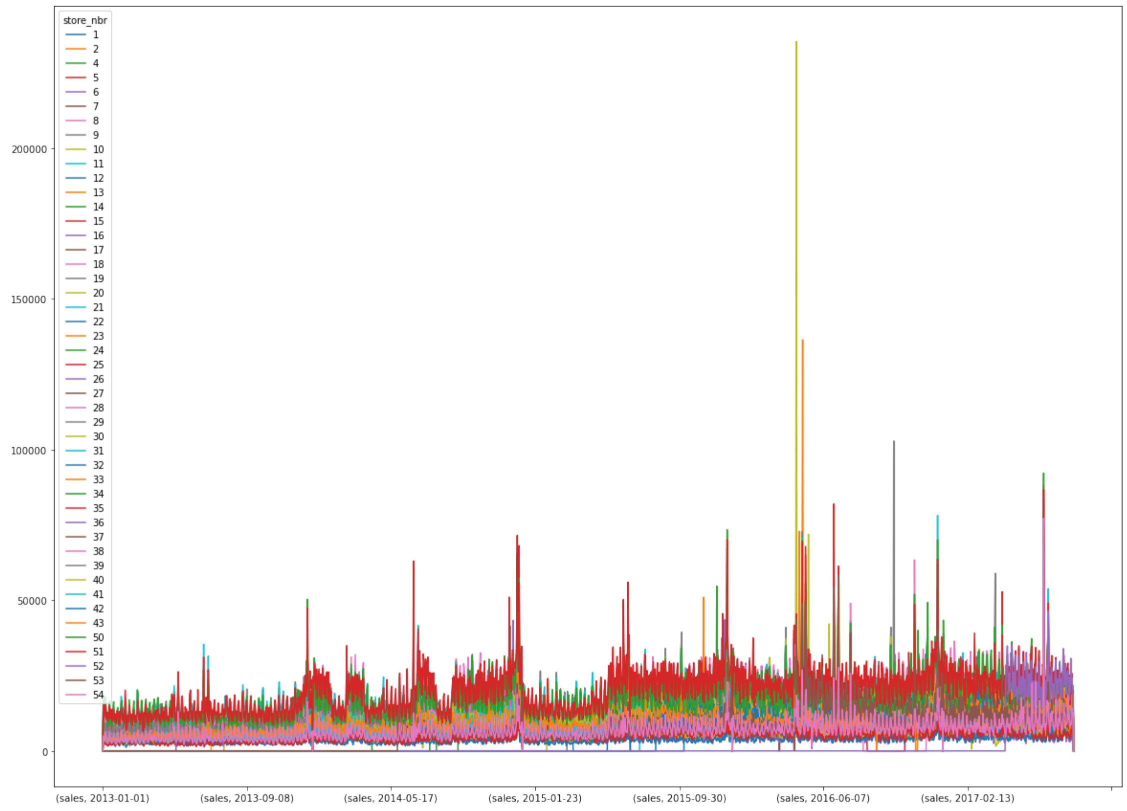
[21]:
```
cluster_stores_ts[cluster_stores_ts['cluster'] == 0].transpose().
 ↪plot(figsize=(20,15))
```

[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2c9d9a3a20>
```



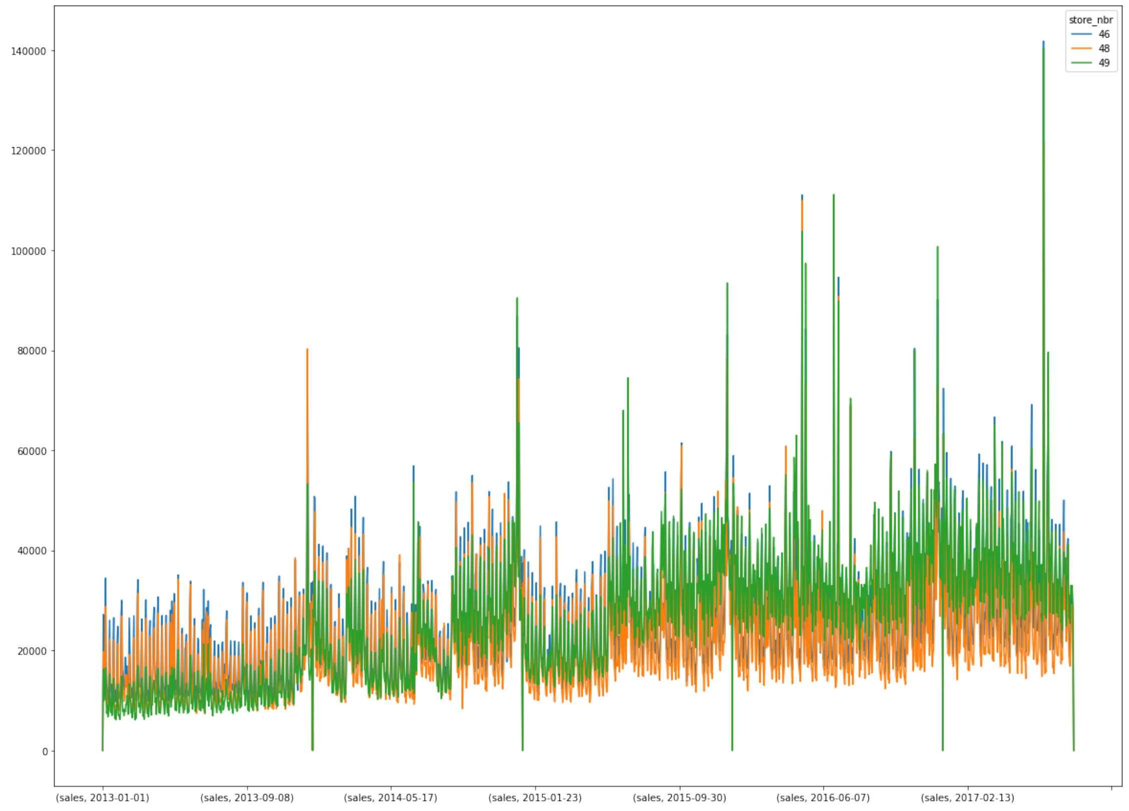
```
[22]: cluster_stores_ts[cluster_stores_ts['cluster'] == 1].transpose().
        ↪plot(figsize=(20,15))
```

```
[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c9d979ba8>
```



```
[23]: cluster_stores_ts[cluster_stores_ts['cluster'] == 2].transpose().
        ↪plot(figsize=(20,15))
```

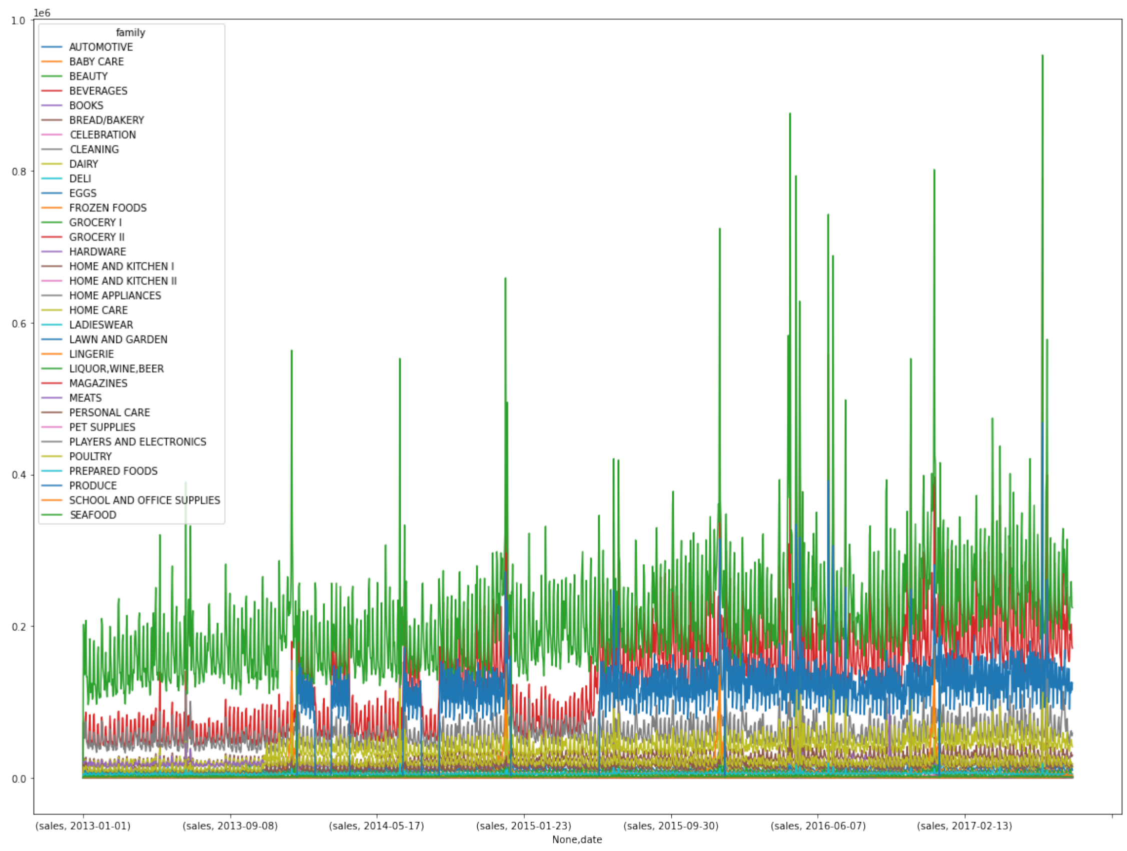
```
[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c8ea30e10>
```



Time series for product families

```
[24]: # aggregating data at product-date level
      train_holiday_oil_df[['sales', 'family', 'date']].groupby(['family','date']).
       ↪agg({'sales':'sum'})\
      .unstack().transpose().plot(figsize=(20,15))
```

```
[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c8e8a67b8>
```



```
[25]: family_ts_sales = train_holiday_oil_df[['sales', 'family', 'date']].
       ↪groupby(['family','date']).agg({'sales':'sum'})\
      .unstack()
```

```
[26]: %%time
      ts_clust_fit_family = TimeSeriesKMeans(n_clusters=3, metric="dtw",
                                   max_iter=10, random_state=0).fit(family_ts_sales)
```

```
CPU times: user 2min 11s, sys: 1min 18s, total: 3min 29s
Wall time: 1min 32s
```

```
[27]: %%time
      predicted_ts_clusters = ts_clust_fit_family.predict(family_ts_sales)
```

```
CPU times: user 4.36 s, sys: 97 µs, total: 4.36 s
Wall time: 4.36 s
```

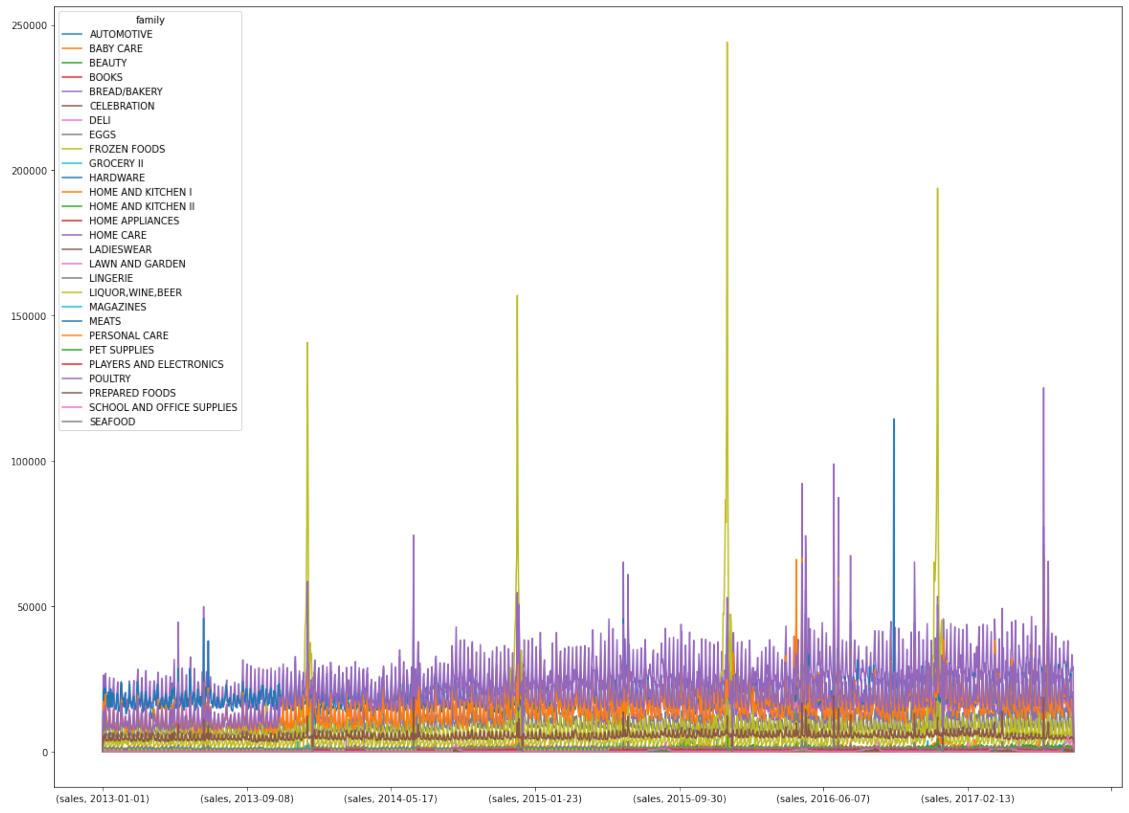
```
[28]: clusters_df_family = pd.Series(predicted_ts_clusters, index=family_ts_sales.
       ↪index, name='cluster')
      cluster_family_ts = family_ts_sales.merge(clusters_df_family.to_frame(),␣
       ↪left_index=True, right_index=True)
```

```
[29]: cluster_family_ts['cluster'].value_counts()
```

```
[29]: 0    28
      2     4
      1     1
      Name: cluster, dtype: int64
```

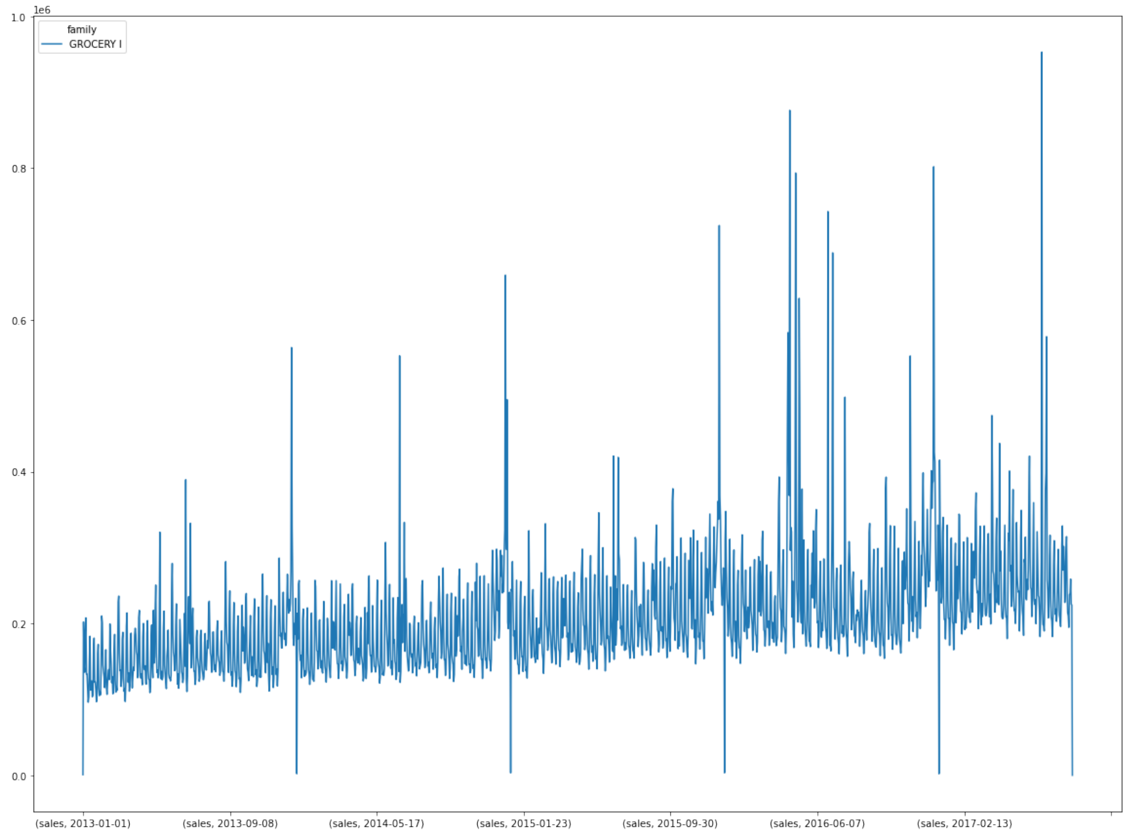
```
[30]: cluster_family_ts[cluster_family_ts['cluster'] == 0].transpose().
       ↪plot(figsize=(20,15))
```

```
[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c8e3499e8>
```



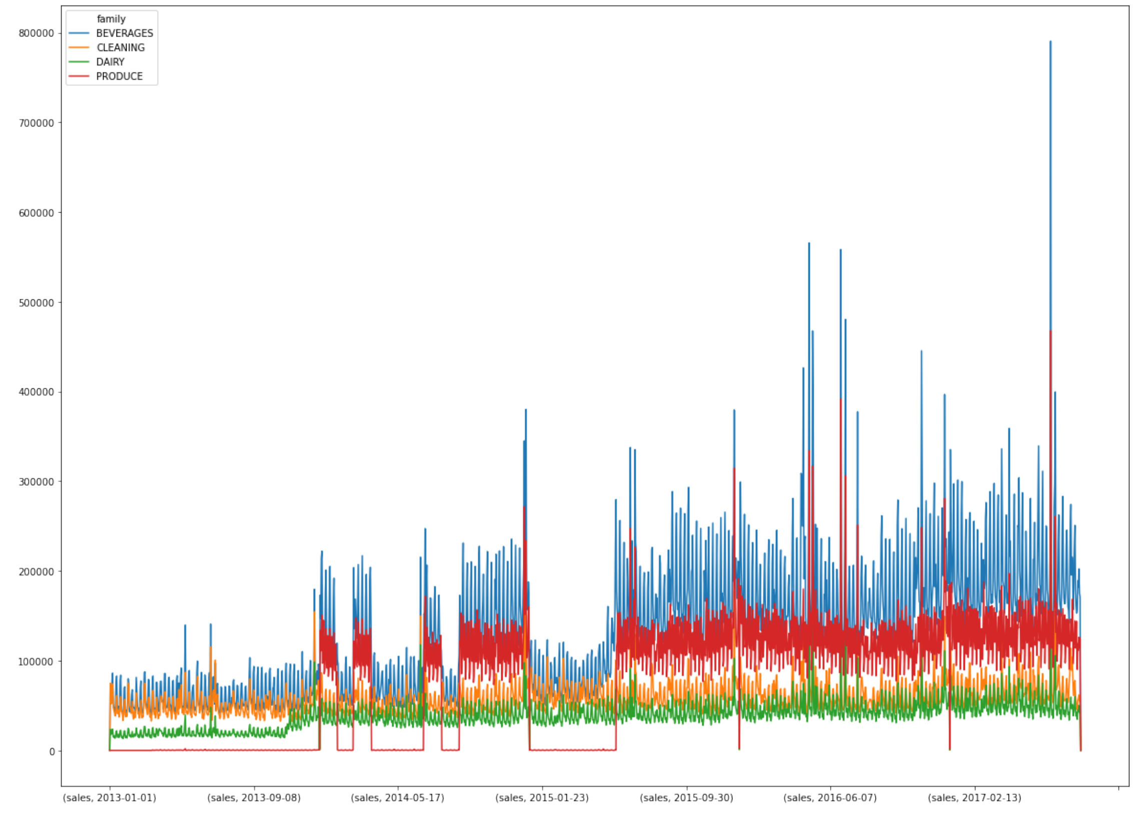
```
[31]: cluster_family_ts[cluster_family_ts['cluster'] == 1].transpose().
       ↪plot(figsize=(20,15))
```

```
[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c8e5b3588>
```



```
[32]: cluster_family_ts[cluster_family_ts['cluster'] == 2].transpose().
       ↪plot(figsize=(20,15))
```

```
[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c8e747320>
```

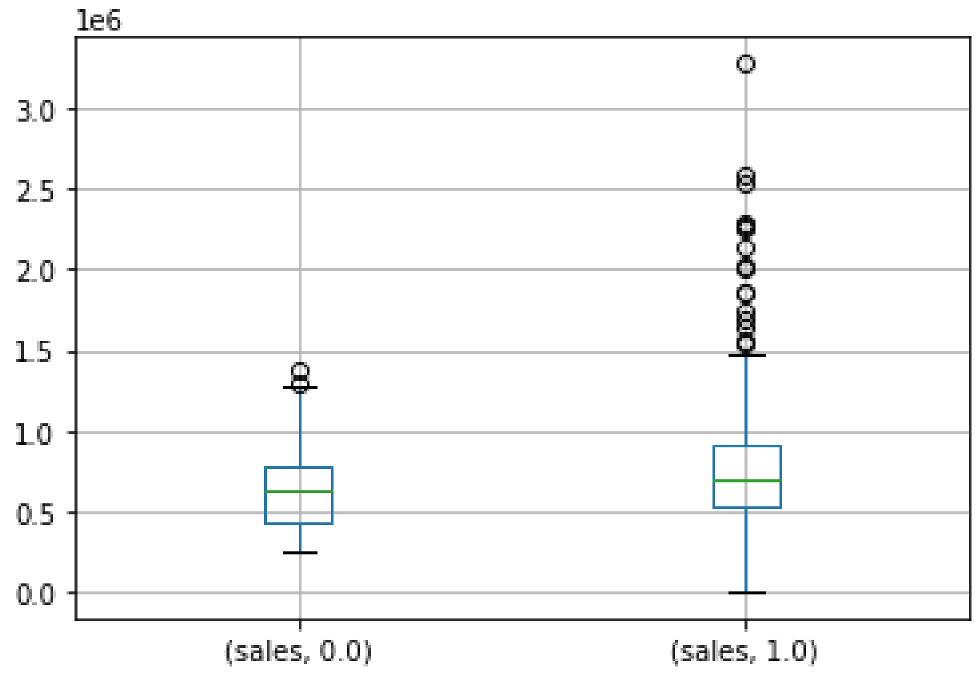


Checking impact of external factors (oil, promotions and holidays) on overall sales

[33]:
```
train_holiday_oil_grouped_by_date  = train_holiday_oil_df.groupby('date')\
[['sales','dcoilwtico','holiday_present','onpromotion']]\
.agg({'sales':'sum', 'dcoilwtico':'mean','onpromotion':'mean','holiday_present':
 ↪'max'})
```

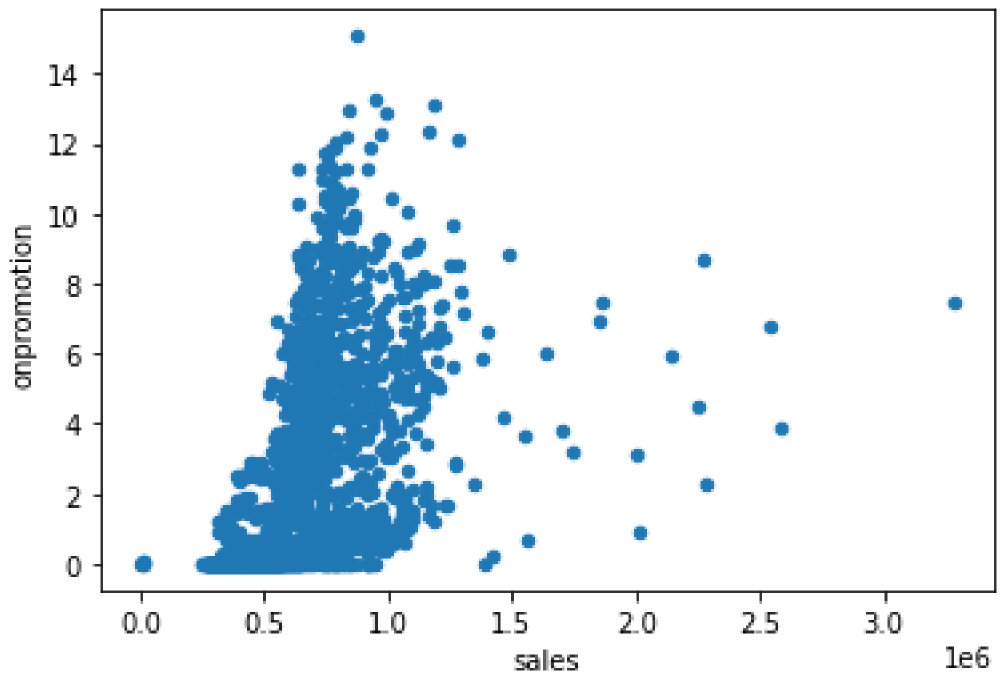
[34]:
```
train_holiday_oil_grouped_by_date[['sales', 'holiday_present']].
 ↪pivot(columns=['holiday_present']).boxplot()
```

[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2c8e5e7748>
```



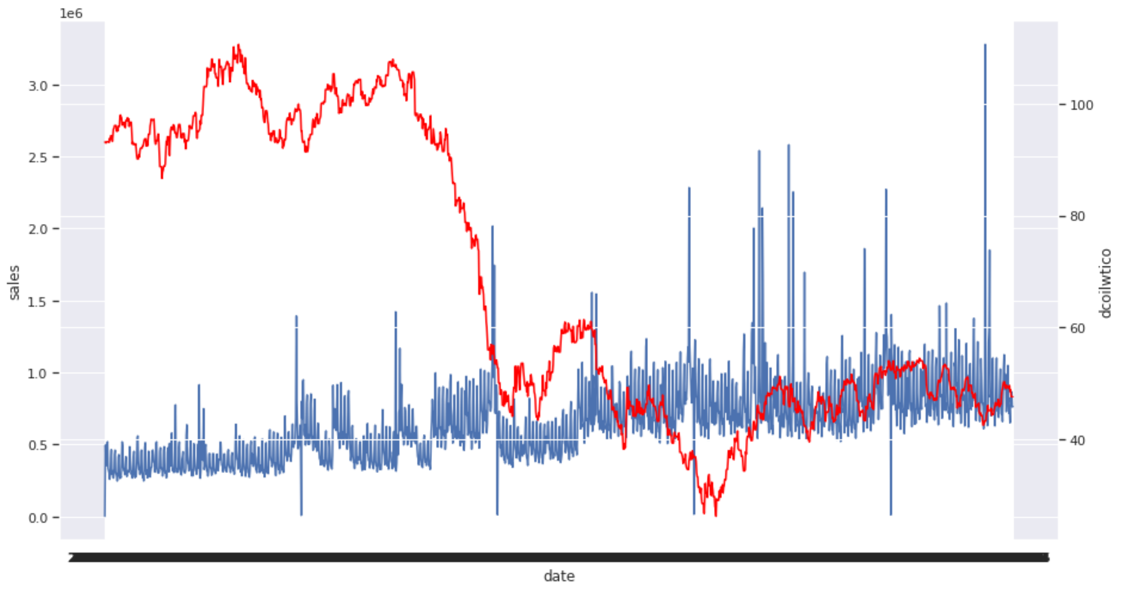
```
[35]: train_holiday_oil_grouped_by_date[['sales', 'onpromotion']].
       ↪sort_values(by='sales').plot.scatter(x='sales', y='onpromotion')
```

```
[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c8ea3cf98>
```



Holidays and promotions have impact on sales and they will be considered while building the model

[36]:
```
# plotting this takes too much time, skip this cell
sns.set(rc = {'figure.figsize':(15,8)})
fig, ax = plt.subplots()
ax= sns.lineplot(x=train_holiday_oil_grouped_by_date.index, y='sales',␣
 ↪data=train_holiday_oil_grouped_by_date)
# adding secondary axis for oil
ax1 = ax.twinx()
ax1 = sns.lineplot(x=train_holiday_oil_grouped_by_date.index, y='dcoilwtico',␣
 ↪data=train_holiday_oil_grouped_by_date, color='red', ci=None)
# strong negative correlation between sales and oil prices
```



There is a strong negative correlation between sales and oil prices

0.6 ### STL decomposition and running the forecast at overall level

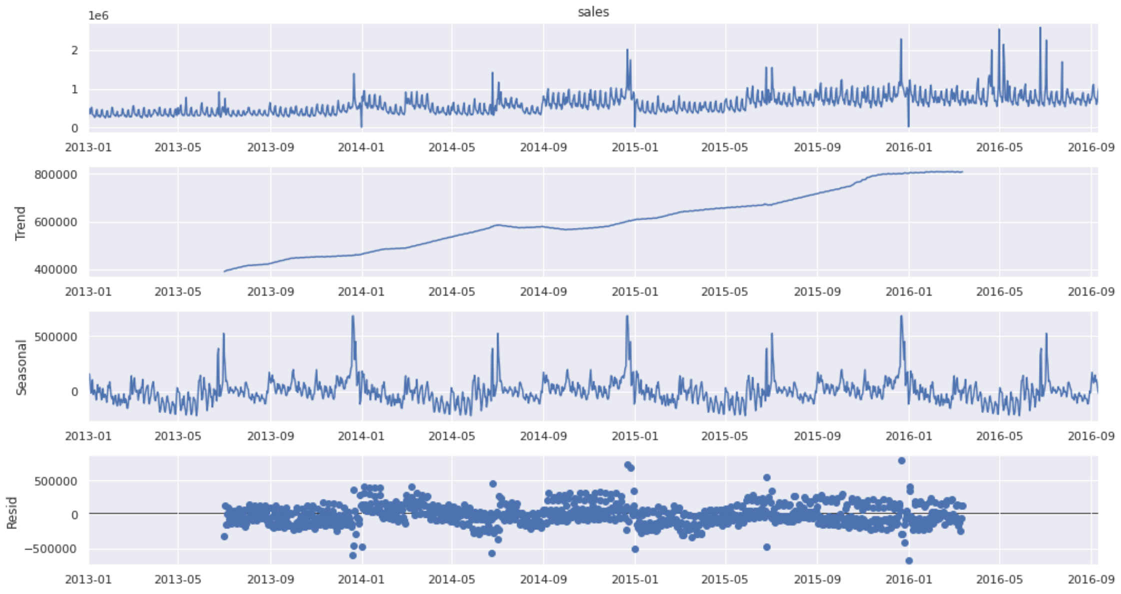
- Checking if simple seasonal decompostion can give good results at overall level

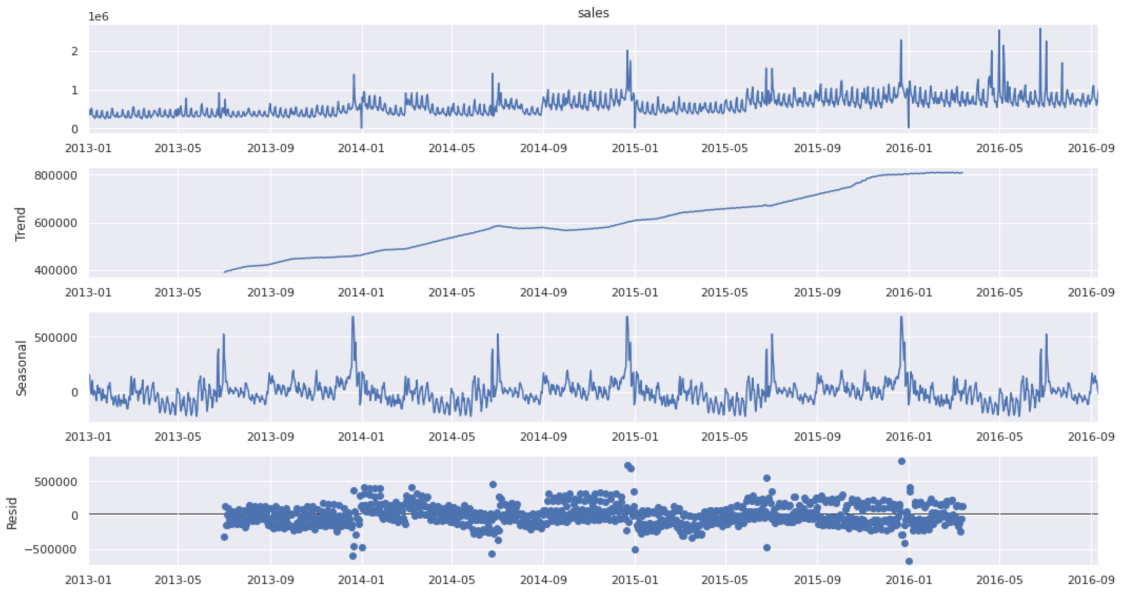
[37]:
```
# making a 80:20 split for train and validation
train_validation_cutoff = int(train_holiday_oil_grouped_by_date.shape[0]*0.8)
train_holiday_oil_grouped_by_date.index = train_holiday_oil_grouped_by_date.
 ↪index.map(lambda x:datetime.datetime.strptime(x,'%Y-%m-%d'))
train = train_holiday_oil_grouped_by_date.iloc[:train_validation_cutoff]
validation = train_holiday_oil_grouped_by_date.iloc[train_validation_cutoff:]
```

[38]:
```
# converting dataframe to time series and decomposing to trend, seasonal and␣
↪residue
train_ts = pd.Series(train['sales'], index=train.index)
tsdecomposed = seasonal_decompose(train_ts, model='additive', freq=365)
```

[39]:
```
tsdecomposed.plot()
```

[39]:





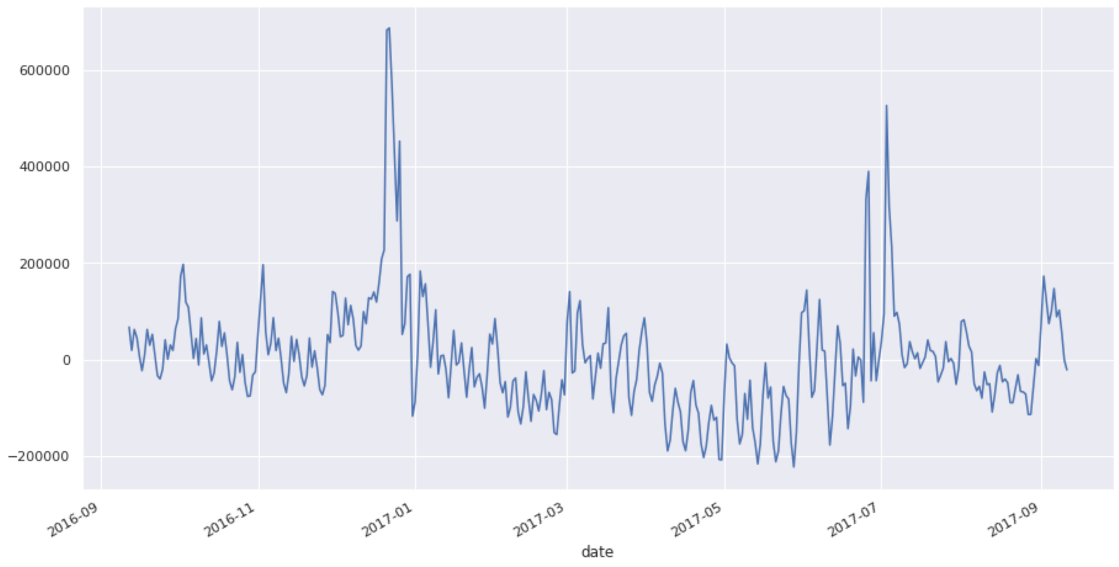
Decomposing trend, seasonal and irregular component

```
[40]: trend = tsdecomposed.trend
      seasonality = tsdecomposed.seasonal
      trend_filled_forward = trend.ffill().bfill()

      residual = train_ts - trend_filled_forward - seasonality
```

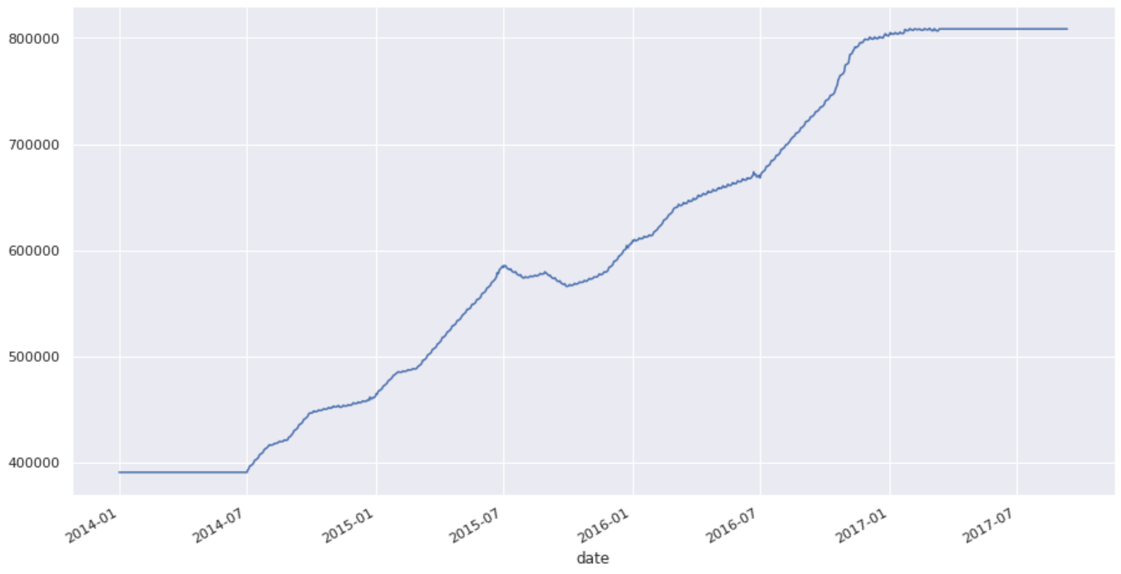
```
[41]: # forecasting seasonality
      # get seasonality factors for past 1 year and propogate the seasonality
      seasonality_forward = seasonality[seasonality.index > max(train.index) -␣
       ↪timedelta(365)]
      # create timestamp index from past 1 year to date
      seasonality_forward.index = seasonality_forward.index + timedelta(365)
      seasonality_forward.plot()
```

```
[41]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c6572e630>
```



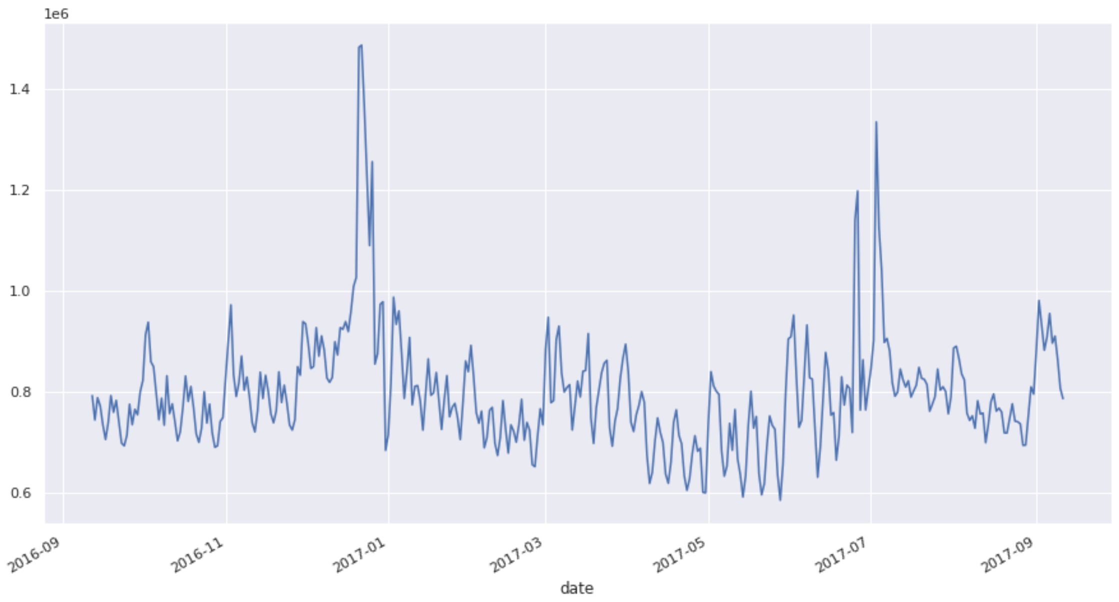
```
[42]: # forecasting trend
      # propogate the trend (this is calculated using centered mean - so initial and␣
       ↪end values are absent)
      trend_filled_forward = trend.ffill().bfill()
      # create timestamp index from past 1 year to date
      trend_filled_forward.index = trend_filled_forward.index + timedelta(365)
      trend_filled_forward.plot()
```

```
[42]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c6572e898>
```



[43]:
```
# add propogated trend and forecast
forecast = seasonality_forward + trend_filled_forward
forecast.plot()
```

[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2c656c0208>
```



[44]:
```
mae_cal = validation.merge(pd.Series(forecast , name='sales'), how='inner',␣
 ↪left_index=True, right_index=True, suffixes=('_actual', '_forecast'))
```

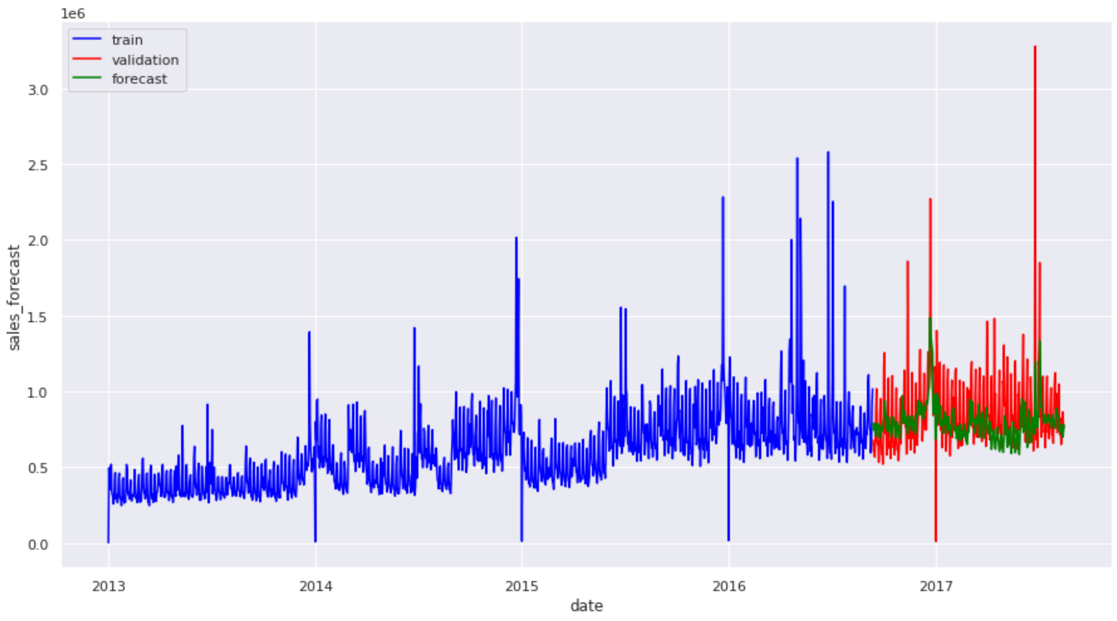
```
[45]: mean_absolute_error(mae_cal['sales_actual'], mae_cal['sales_forecast'])
```

```
[45]: 171513.40709485166
```

```
[46]: %%time
      sns.set(rc = {'figure.figsize':(15,8)})
      ax4 = plt.subplots()
      sns.lineplot(y=train['sales'], x=train.index, color='blue', label='train')
      sns.lineplot(y=mae_cal['sales_actual'], x=mae_cal['sales_actual'].index,␣
       ↪color='red', label='validation')
      sns.lineplot(y=mae_cal['sales_forecast'], x=mae_cal['sales_forecast'].index,␣
       ↪color='green', label = 'forecast')
```

```
CPU times: user 555 ms, sys: 12.1 ms, total: 567 ms
Wall time: 559 ms
```

```
[46]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2c656acd68>
```



This looks like a decent forecast, it can be improved by including the residual/irregular component

0.6.1 Forecasting residue - Overall level

Calculating the impact of holidays on on residue, as anything that is not seasonal will be captured in the residue

[47]:
```
external_merged = residual.to_frame().merge(train, left_index=True,␣
↪right_index=True)

# regressing the holiday variable against sales to calculate betas for effects
X = np.array(external_merged[['holiday_present', 'onpromotion']])
X = sm_api.add_constant(X)
Y = np.array(external_merged[0])
ols = sm.regression.linear_model.OLS(X, Y)
ols_fit = ols.fit()
beta_val = ols_fit.params[0][1]
```

[48]:
```
# scaling residue based on the holiday variables and removing its effect
residual = external_merged.apply(lambda x: x[0] - x[0]*beta_val  \
                                 if((x['holiday_present'] == 1) or␣
↪(x['onpromotion'] == 1)) else x[0] ,axis=1)
```

[49]:
```
#ARIMA 1,1,2 model for residue as that gives the better results
history = [x for x in residual.dropna()]
predictions = list()

model = sm_api.tsa.ARIMA(history, order=(1,1,2))
model_fit = model.fit()
yhat = model_fit.forecast(len(validation))


residue_adjusted_series = pd.Series(yhat[0], index=validation.index)

predictions = residue_adjusted_series - np.mean(residue_adjusted_series)
```

[50]:
```
sales_pred_series = pd.Series(np.array(predictions).ravel(),␣
↪name='sales_prediction', index=validation.index)
external_merged_post = sales_pred_series.to_frame().merge(validation,␣
↪left_index=True, right_index=True)
```

[51]:
```
# scaling residue based on the holiday variables and adding back its effect
predictions_corrected = external_merged_post.apply(lambda x:␣
↪x['sales_prediction'] + x['sales_prediction'] * beta_val \
                                                  if((x['holiday_present'] ==␣
↪1) or (x['onpromotion'] == 1)) else x['sales_prediction'] ,axis=1)
```

[52]:
```
# adding seasonality, trend and adjusted residue to get the forecast
forecast1 = seasonality_forward + trend_filled_forward + pd.Series(np.
↪array(predictions_corrected).ravel(), index=validation.index)
```

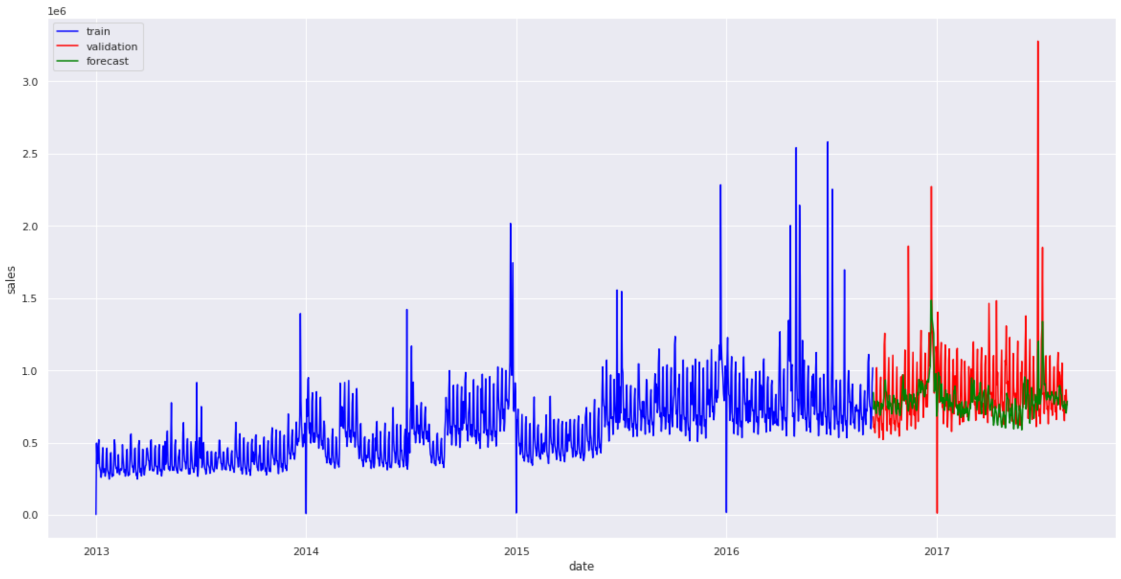
[53]:

```
mae_cal2 = validation.merge(pd.Series(forecast1.dropna(), name='sales'),␣
 ↪how='inner', left_index=True, right_index=True, suffixes=('_actual',␣
 ↪'_forecast'))
mean_absolute_error(mae_cal2['sales_actual'], mae_cal2['sales_forecast']) # mae␣
 ↪has reduced
```

[53]:
```
171144.87610258584
```

[54]:
```
ax4 = plt.subplots(figsize=(20,10))
sns.lineplot(y=train['sales'], x=train.index, color='blue', label='train')
sns.lineplot(y=validation['sales'], x=validation.index, color='red',␣
 ↪label='validation')
sns.lineplot(y=forecast1, x=forecast1.index, color='green', label = 'forecast')
```

[54]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2c65812080>
```



This looks like a better forecast and the reduction in MAE proves it

1 Spark execution for across stores and products

[55]:
```
# https://www.pauldesalvo.com/how-to-install-spark-on-google-colab/

# !sudo apt update
# !apt-get install openjdk-8-jdk-headless -qq > /dev/null
# !wget -q https://dlcdn.apache.org/spark/spark-3.0.3/spark-3.0.3-bin-hadoop3.2.
 ↪tgz
# !tar xf /content/spark-3.0.3-bin-hadoop3.2.tgz
# !pip install -q findspark
# !pip install pyspark

# import os
# os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
# os.environ["SPARK_HOME"] = "/content/spark-3.0.3-bin-hadoop3.2"

# !pip3 install pyarrow

import findspark
findspark.init()
findspark.find()

from pyspark.sql import DataFrame, SparkSession
from typing import List
import pyspark.sql.types as T
import pyspark.sql.functions as F

from pyspark.sql.functions import col, struct, next_day, date_sub,␣
 ↪collect_list, array, lit, explode, min, max, first, last
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import pandas_udf, PandasUDFType, sum, max, col,␣
 ↪concat, lit
import sys
import os
from fbprophet import Prophet

import statsmodels.tsa.api as sm
import numpy as np
import pandas as pd
from pandas.core.indexes.datetimes import date_range
```

```
Importing plotly failed. Interactive plots will not work.
```

[56]:
```
spark = SparkSession.builder.appName("sales_forecasting_psavale").
 ↪master("local[*]").getOrCreate()
sc = spark.sparkContext
sc.setSystemProperty("spark.dynamicAllocation.enabled", "true")
sc.setSystemProperty('spark.shuffle.service.enabled', "true")
sc.setSystemProperty("spark.speculation", "true")
sc.setSystemProperty("spark.serializer", "org.apache.spark.serializer.
 ↪KryoSerializer")
sc.setSystemProperty("spark.driver.memory", "2G")
sc.setSystemProperty("spark.executor.memory", "8G")
sc.setSystemProperty("spark.executor.cores", "2")
sc.setSystemProperty("spark.executor.instances", "20")
sc.setSystemProperty("spark.sql.shuffle.partitions", "400")
```

```
/home/praths/anaconda3/envs/py36_gpu/lib/python3.6/site-
packages/pyspark/context.py:238: FutureWarning: Python 3.6 support is deprecated
in Spark 3.2.
  FutureWarning
```

[57]:
```
spark
```

[57]:
```
<pyspark.sql.session.SparkSession at 0x7f2c6162a048>
```

[58]:
```
spark.sparkContext.getConf().get("spark.serializer")
```

[58]:
```
'org.apache.spark.serializer.KryoSerializer'
```

1.1 ### Train test stratified split

- This is required because we want equal division of stores and products across test ant train

[59]:
```
family=list(set(train_holiday_oil_df['family']))
store_nbr=list(set(train_holiday_oil_df['store_nbr']))
```

[60]:
```
train_validation_df = train_holiday_oil_df[['date', 'store_nbr',␣
↪'family','sales', 'onpromotion', 'holiday_present']]
```

[61]:
```
# 80:20 stratified split is defined here
def train_test_split(data):
    size=int(len(data)*0.8)
    x_train =data.drop(columns=['sales']).iloc[0:size]
    x_test = data.drop(columns=['sales']).iloc[size:]
    y_train=data['sales'].iloc[0:size]
    y_test=data['sales'].iloc[size:]
    return x_train, x_test,y_train,y_test
```

[62]:
```
# %%time
# This takes a long time for execution so pick split data from local
try:

    X_train= pd.read_pickle("/home/praths/notebooks/MIS/group_project/
↪MIS_group_project/X_train.pkl")
    X_test= pd.read_pickle("/home/praths/notebooks/MIS/group_project/
↪MIS_group_project/X_test.pkl")
    Y_train= pd.read_pickle("/home/praths/notebooks/MIS/group_project/
↪MIS_group_project/Y_train.pkl")
    Y_test= pd.read_pickle("/home/praths/notebooks/MIS/group_project/
↪MIS_group_project/Y_test.pkl")
```

```
except:
  # this takes a long time to compute
  # loop each family and store number split the data into train and test data
    X_train=[]
    X_test=[]
    Y_train=[]
    Y_test=[]
    for i in range(0,len(family)):
        for j in range(0, len(store_nbr)):
            df = train_validation_df[(train_validation_df['family']==family[i])␣
↪& (train_validation_df['store_nbr']==store_nbr[j])]
            x_train, x_test,y_train,y_test=train_test_split(df)
            X_train.append(x_train)
            X_test.append(x_test)
            Y_train.append(y_train)
            Y_test.append(y_test)
    X_train=pd.concat(X_train)
    Y_train=pd.DataFrame(pd.concat(Y_train))
    X_test=pd.concat(X_test)
    Y_test=pd.DataFrame(pd.concat(Y_test))

    X_train.to_pickle("/home/praths/notebooks/MIS/group_project/
↪MIS_group_project/X_train.pkl")
    X_test.to_pickle("/home/praths/notebooks/MIS/group_project/
↪MIS_group_project/X_test.pkl")
    Y_train.to_pickle("/home/praths/notebooks/MIS/group_project/
↪MIS_group_project/Y_train.pkl")
    Y_test.to_pickle("/home/praths/notebooks/MIS/group_project/
↪MIS_group_project/Y_test.pkl")
```

[63]:
```
train = pd.concat([X_train, Y_train], axis=1)
test = pd.concat([X_test, Y_test], axis=1)
```

[64]:
```
train_spark = train[['date', 'store_nbr', 'family', 'sales', 'onpromotion',␣
↪'holiday_present']]
train_spark['forecast'] = 0.0
train_spark['mae'] = 0.0
```

[65]:
```
test_spark  = test[['date', 'store_nbr', 'family', 'sales', 'onpromotion',␣
↪'holiday_present']]
test_spark['forecast'] = 0.0
test_spark['mae'] = 0.0
```

[66]:
```
# define schema for udf operations
schema = StructType([
        StructField('date', StringType(), True),
        StructField('store_nbr', IntegerType(), True),
        StructField('family', StringType(), True),
        StructField('sales', DoubleType(), True),
        StructField('onpromotion', IntegerType(), True),
        StructField('holiday_present', DoubleType(), True),
        StructField('forecast', DoubleType(), True),
        StructField('mae', DoubleType(), True)
    ])
```

```
[67]: train_spark_df = spark.createDataFrame(train_spark, schema=schema)
      train_spark_df = train_spark_df.withColumn("date", F.
       ↪to_date(col("date"),"yyyy-MM-dd").alias("date"))
```

```
[68]: test_spark_df = spark.createDataFrame(test_spark, schema=schema)
      test_spark_df = test_spark_df.withColumn("date", F.
       ↪to_date(col("date"),"yyyy-MM-dd").alias("date"))
```

```
[69]: cutoff = train_spark_df.select(max(col('date'))).collect()[0][0]
      df = (train_spark_df.union(test_spark_df)).sort(col('date'))
```

STL decomposition

```
[70]: @pandas_udf(schema, PandasUDFType.GROUPED_MAP)
      def fit_stl_pandas_udf(df):
          def stl_udf(df,cutoff):

              df['date'] = pd.to_datetime(df['date'])
              ts_train = (df.query('date <= @cutoff'))
              ts_test = (df.query('date > @cutoff'))

              ts_train.set_index('date',inplace = True)
              time_series_data = ts_train['sales']

              ts_test.set_index('date',inplace = True)
              ts_test_backup = copy.deepcopy(ts_test)
              ts_test = ts_test.drop('sales', axis=1)


              train_ts = pd.Series(ts_train['sales'], index=ts_train.index)
              tsdecomposed = seasonal_decompose(train_ts, model='additive',␣
       ↪period=365)

              # decompose ts
              trend = tsdecomposed.trend
              seasonality = tsdecomposed.seasonal
```

```
        #ffill trend
        trend_filled_forward = trend.ffill().bfill()
        residual = train_ts - trend_filled_forward - seasonality

        # forecasting seasonality
        # get seasonality factors for past 1 year and propogate the seasonality
        seasonality_forward = seasonality[seasonality.index > train_ts.index.
↪max() - timedelta(ts_test.shape[0])]


        # create timestamp index from past 1 year to date
        seasonality_forward.index = seasonality_forward.index +␣
↪timedelta(ts_test.shape[0])

        # forecasting trend
        # propogate the trend (this is calculated using centered mean - so␣
↪initial and end values are absent)
        trend_filled_forward = trend.ffill().bfill()
        # create timestamp index from past 1 year to date
        trend_filled_forward.index = trend_filled_forward.index +␣
↪timedelta(ts_test.shape[0])


        forecast = seasonality_forward + trend_filled_forward


        try:
            preds = ts_test_backup.merge(pd.Series(forecast, name='sales'),␣
↪how='left', left_index=True, right_index=True, suffixes=('_actual',␣
↪'_forecast'))
            preds = preds.dropna()
            mae = mean_absolute_error(np.array(preds['sales_actual']), np.
↪array(preds['sales_forecast']))
            return pd.DataFrame({'date': preds.index.astype(str), 'store_nbr':␣
↪preds.store_nbr,\
                                 'family': preds.family, 'sales': preds.
↪sales_actual,\
                                 'onpromotion': preds.onpromotion,␣
↪'holiday_present': preds.holiday_present,\
                                 'forecast':preds.sales_forecast, 'mae': mae})
        except:
            return pd.DataFrame({'date': preds.index.astype(str), 'store_nbr':␣
↪preds.store_nbr,\
                                 'family': preds.family, 'sales': preds.
↪sales_actual,\
                                 'onpromotion': preds.onpromotion,␣
↪'holiday_present': preds.holiday_present,\
                                 'forecast':9999, 'mae': 9999})
    return stl_udf(df, cutoff)
```

[71]:
```
%%time
# stl extension
stl_fit_df = df\
.groupBy(["store_nbr", "family"])\
.apply(fit_stl_pandas_udf).cache()
forecasted_stl_fit_df = stl_fit_df.groupBy(['store_nbr', 'family']).avg("mae")\
.withColumnRenamed("avg(mae)","avg_mae")
forecasted_stl_fit_df.select(F.mean('avg_mae')).collect()[0][0]
```

```
CPU times: user 107 ms, sys: 17 ms, total: 124 ms
Wall time: 21.3 s
```

[71]:
```
139.25674140723902
```

STL decomposition with adjustment for external factors and residue

[76]:
```
@pandas_udf(schema, PandasUDFType.GROUPED_MAP)
def fit_stl_arima_pandas_w_residue_udf(df):
    def stl_arima_w_residue_udf(df,cutoff):

        df['date'] = pd.to_datetime(df['date'])
        ts_train = (df.query('date <= @cutoff'))
        ts_test = (df.query('date > @cutoff'))

        ts_train.set_index('date',inplace = True)
        time_series_data = ts_train['sales']

        ts_test.set_index('date',inplace = True)
        ts_test_backup = copy.deepcopy(ts_test)
        ts_test = ts_test.drop('sales', axis=1)


        train_ts = pd.Series(ts_train['sales'], index=ts_train.index)
        tsdecomposed = seasonal_decompose(train_ts, model='additive',␣
↪period=365)

        # decompose ts
        trend = tsdecomposed.trend
        seasonality = tsdecomposed.seasonal

        #ffill trend
        trend_filled_forward = trend.ffill().bfill()
        residual = train_ts - trend_filled_forward - seasonality

        # forecasting seasonality
        # get seasonality factors for past 1 year and propogate the seasonality
        seasonality_forward = seasonality[seasonality.index > train_ts.index.
↪max() - timedelta(ts_test.shape[0])]


        # create timestamp index from past 1 year to date
        seasonality_forward.index = seasonality_forward.index +␣
↪timedelta(ts_test.shape[0])

        # forecasting trend
        # propogate the trend (this is calculated using centered mean - so␣
↪initial and end values are absent)
        trend_filled_forward = trend.ffill().bfill()
        # create timestamp index from past 1 year to date
        trend_filled_forward.index = trend_filled_forward.index +␣
↪timedelta(ts_test.shape[0])


        # adding seasonality and trend
        forecast = seasonality_forward + trend_filled_forward

        external_merged = residual.to_frame().merge(ts_train, left_index=True,␣
↪right_index=True)

        #forecasting residue with ARIMA 000
        # regressing the holiday and promotion variable against sales to␣
↪calculate betas for effect
        X_ext_factor = np.array(external_merged[['holiday_present',␣
↪'onpromotion']])
        X_ext_factor = sm_api.add_constant(X_ext_factor)
        Y_ext_factor = np.array(external_merged[0])
        ols_ext_factor = sm_api.regression.linear_model.OLS(X_ext_factor,␣
↪Y_ext_factor)
        ols_ext_factor_fit = ols_ext_factor.fit()
        beta_ext_factor = ols_ext_factor_fit.params[0][1]


        #remove effect of external factors
        residual = external_merged.apply(lambda x: x[0] - x[0]*beta_ext_factor \
                                         if ((x['holiday_present'] == 1) or␣
↪(x['onpromotion'] == 1)) else x[0], axis=1)

        history = [x for x in residual.dropna()]
        try:
            model = sm_api.tsa.ARIMA(history, order=(0,0,0))
            model_fit = model.fit()
            yhat = model_fit.forecast(len(ts_test))


            residue_adjusted_series = pd.Series(yhat[0], index=ts_test.index)
            residue_adjusted_series = residue_adjusted_series - np.
↪mean(residue_adjusted_series)

            #add effect of external factors
            external_merged_post = residue_adjusted_series.
↪to_frame(name='sales_prediction').merge(ts_test, left_index=True,␣
↪right_index=True)


            predictions_corrected = external_merged_post.apply(lambda x:␣
↪x['sales_prediction'] + \
                                        x['sales_prediction']*beta_ext_factor \
                                             if ((x['holiday_present'] == 1) or␣
↪(x['onpromotion'] == 1))\
                                                                  else␣
↪x['sales_prediction'], axis=1)

            forecast1 = forecast + predictions_corrected


            preds = ts_test_backup.merge(pd.Series(forecast1, name='sales'),␣
↪how='left', left_index=True, right_index=True, suffixes=('_actual',␣
↪'_forecast'))
            preds = preds.dropna()
            mae = mean_absolute_error(np.array(preds['sales_actual']), np.
↪array(preds['sales_forecast']))
            return pd.DataFrame({'date': preds.index.astype(str), 'store_nbr':␣
↪preds.store_nbr,\
                                'family': preds.family, 'sales': preds.
↪sales_actual,\
                                'onpromotion': preds.onpromotion,␣
↪'holiday_present': preds.holiday_present,\
                                'forecast':preds.sales_forecast, 'mae': mae})
        except:
            return pd.DataFrame({'date': ts_test_backup.index.astype(str),␣
↪'store_nbr': ts_test_backup.store_nbr,\
                                'family': ts_test_backup.family, 'sales':␣
↪ts_test_backup.sales,\
                                'onpromotion': ts_test_backup.onpromotion,␣
↪'holiday_present': ts_test_backup.holiday_present,\
                                'forecast':9999, 'mae':9999})
    return stl_arima_w_residue_udf(df, cutoff)
```

```
[77]: %%time
      # arima with residue and external regressor adjustment
      stl_arima_fit_w_residue_df = df\
      .groupBy(["store_nbr", "family"])\
      .apply(fit_stl_arima_pandas_w_residue_udf).cache()
      forecasted_stl_arima_w_residue_df = stl_arima_fit_w_residue_df.
       ↪groupBy(['store_nbr', 'family']).avg("mae")\
      .withColumnRenamed("avg(mae)","avg_mae")
      forecasted_stl_arima_w_residue_df.select(F.mean('avg_mae')).collect()[0][0]
```

```
CPU times: user 159 ms, sys: 78.8 ms, total: 238 ms
Wall time: 42.4 s
```

```
[77]: 135.52448726673828
```

STL with residue adjustments gives better results compared to only STL, which is also something that we saw at an overall level

Prophet

```
[74]: @pandas_udf(schema, PandasUDFType.GROUPED_MAP)
      def fit_prophet_auto_arima_udf(df):
          def prophet_auto_arima_udf(df,cutoff):

              df['date'] = pd.to_datetime(df['date'])
              ts_train = (df.query('date <= @cutoff')
                          .rename(columns={'date': 'ds', 'sales': 'y'})
                           .sort_values('ds')

                          )
              ts_train = ts_train[['ds', 'y']]
              ts_test = (df.query('date > @cutoff'))
              ts_test_backup = copy.deepcopy(ts_test)
              ts_test = ts_test[['date', 'sales']]
              ts_test = (ts_test.rename(columns={'date': 'ds', 'sales': 'y'})
                         .sort_values('ds')
                          .assign(ds=lambda x: pd.to_datetime(x["ds"]))
                          .drop('y', axis=1))

              try:
                  m = Prophet(yearly_seasonality=True,
                              weekly_seasonality=False,
                              daily_seasonality=False)
                  m.fit(ts_train, iter=20)


                  ts_hat = (m.predict(ts_test)[["ds", "yhat"]]
                            .assign(ds=lambda x: pd.to_datetime(x["ds"]))
                            ).merge(ts_test, on=["ds"], how="left")

                  preds = ts_test_backup.merge(ts_hat, how='inner', left_on='date',␣
      ↪right_on='ds', suffixes=('_actual', '_forecast'))
                  preds = preds.dropna()

                  mae = mean_absolute_error(np.array(preds['sales']), np.
      ↪array(preds['yhat']))
                  return pd.DataFrame({'date': preds.index.astype(str), 'store_nbr':␣
      ↪preds.store_nbr,\
                                       'family': preds.family, 'sales': preds.sales,\
                                       'onpromotion': preds.onpromotion,␣
      ↪'holiday_present': preds.holiday_present,\
                                       'forecast':preds['yhat'], 'mae': mae})
              except:
                  return pd.DataFrame({'date': ts_test_backup.index.astype(str),␣
      ↪'store_nbr': ts_test_backup.store_nbr,\
                                       'family': ts_test_backup.family, 'sales':␣
      ↪ts_test_backup.sales,\
                                       'onpromotion': ts_test_backup.onpromotion,␣
      ↪'holiday_present': ts_test_backup.holiday_present,\
                                       'forecast':9999, 'mae': 9999})
          return prophet_auto_arima_udf(df, cutoff)
```

[75]:
```
%%time
# prophet
prophet_auto_arima_df = df\
.groupBy(["store_nbr", "family"])\
.apply(fit_prophet_auto_arima_udf).cache()
forecasted_prophet_auto_arima_df = prophet_auto_arima_df.groupBy(['store_nbr',␣
 ↪'family']).avg("mae")\
.withColumnRenamed("avg(mae)","avg_mae")
forecasted_prophet_auto_arima_df.select(F.mean('avg_mae')).collect()[0][0]
```

```
CPU times: user 169 ms, sys: 47.1 ms, total: 216 ms
Wall time: 6min 20s
```

[75]:
```
148.4759609038448
```

2 Conclusion

- STL and STL with residue adjustment give good results and they have simpler implementation and better explainibility
- Auto-ML techniques like prophet take more time for execution, cost compute resources and are black box making the results difficult to interpret

2.1 # Next Steps

Improvements can be made to the models to increase accuracy. Following are some of the changes we can incorporate: 1. Running grid search to choose best ARIMA(PDQ) values for modeling the residue 2. Grid search can also be used to select the best modelling technique for residue viz.- simple exponential smoothening, holts winter etc. 3. Deep Neural networks with memory like RNN or LSTMs can be used for better predict time series 4. Prophet can be tuned and external regressors like holidays and promotions can be introduced in it to improve accuracy