Table of contents

Introduction

Kernel for the demand forecasting (https://www.kaggle.com/c/demand-forecasting-kernels-only) Kaggle competition.

Answer some of the questions posed:

- What's the best way to deal with seasonality?
- Should stores be modeled separately, or can you pool them together?
- Does deep learning work better than ARIMA?
- Can either beat xgboost?

Preparation

Dependencies

In [1]:

```
import pandas as pd
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('fivethirtyeight')
sns.set()
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import statsmodels.api as sm
import xgboost as xgb
import lightgbm as lgb
from sklearn.model_selection import train_test_split

import warnings
# import the_module_that_warns

warnings.filterwarnings("ignore")

from fbprophet import Prophet


## for Deep-learing:
import keras
from keras.layers import Dense
from keras.models import Sequential
from keras.utils import to_categorical
from keras.optimizers import SGD,Adadelta,Adam,RMSprop
from keras.callbacks import EarlyStopping
from keras.utils import np_utils
import itertools
from keras.layers import LSTM
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers import Dropout
```

```
/opt/conda/lib/python3.6/site-packages/statsmodels/compat/pandas.p
y:56: FutureWarning:

The pandas.core.datetools module is deprecated and will be removed
in a future version. Please use the pandas.tseries module instead.

Using TensorFlow backend.
```

Load the datasets

In [2]:

```
# Input data files are available in the "../input/" directory.
# First let us load the datasets into different Dataframes
def load_data(datapath):
    data = pd.read_csv(datapath)
   # Dimensions
    print('Shape:', data.shape)
    # Set of features we have are: date, store, and item
    display(data.sample(10))
    return data


train_df = load_data('../input/demand-forecasting-kernels-only/train.csv')
test_df = load_data('../input/demand-forecasting-kernels-only/test.csv')
sample_df = load_data('../input/demand-forecasting-kernels-only/sample_submissio
n.csv')
```

```
Shape: (913000, 4)
```

	date	store	item	sales
527821	2013-04-18	10	29	67
131445	2017-12-05	2	8	72
754918	2015-02-20	4	42	29
329449	2015-02-09	1	19	26
389892	2015-08-13	4	22	100
419943	2017-11-25	10	23	37
668129	2017-06-28	6	37	28
657709	2013-12-16	1	37	13
526902	2015-10-12	9	29	60
435884	2016-07-20	9	24	98

```
Shape: (45000, 4)
```

	id	date	store	item
16643	16643	2018-03-25	5	19
16579	16579	2018-01-20	5	19
595	595	2018-02-25	7	1
17817	17817	2018-03-29	8	20
15099	15099	2018-03-11	8	17
25534	25534	2018-03-06	4	29
41844	41844	2018-03-26	5	47
26423	26423	2018-02-23	4	30
35984	35984	2018-03-16	10	40
41863	41863	2018-01-14	6	47

```
Shape: (45000, 2)
```

	id	sales
15487	15487	52
16684	16684	52
19001	19001	52
36374	36374	52
10529	10529	52
23016	23016	52
13132	13132	52
13547	13547	52
12174	12174	52
13472	13472	52

Time series data exploration

(This portion was forked (https://www.kaggle.com/danofer/getting-started-with-time-series-features).)

The goal of this kernel is data exploration of a time-series sales data of store items.
The tools `pandas`, `matplotlib` and, `plotly` are used for slicing & dicing the data and visualizations.

Distribution of sales

Now let us understand how the sales varies across all the items in all the stores

In [3]:

```
# Sales distribution across the train data
def sales_dist(data):
    """
        Sales_dist used for Checing Sales Distribution.
        data :  contain data frame which contain sales data
    """
    sales_df = data.copy(deep=True)
    sales_df['sales_bins'] = pd.cut(sales_df.sales, [0, 50, 100, 150, 200, 250])
    print('Max sale:', sales_df.sales.max())
    print('Min sale:', sales_df.sales.min())
    print('Avg sale:', sales_df.sales.mean())
    print()
    return sales_df

sales_df = sales_dist(train_df)

# Total number of data points
total_points = pd.value_counts(sales_df.sales_bins).sum()
print('Sales bucket v/s Total percentage:')
display(pd.value_counts(sales_df.sales_bins).apply(lambda s: (s/total_points)*10
0))
```

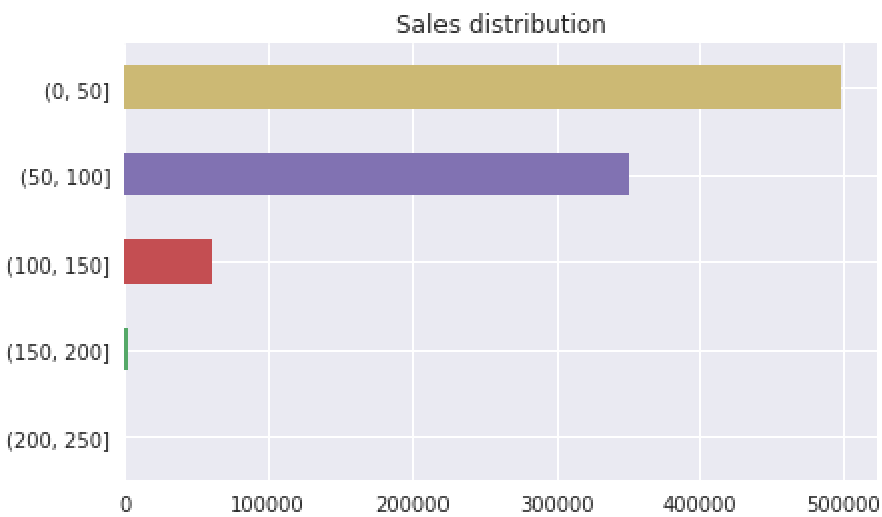
```
Max sale: 231
Min sale: 0
Avg sale: 52.250286966046005

Sales bucket v/s Total percentage:


(0, 50]       54.591407
(50, 100]     38.388322
(100, 150]     6.709974
(150, 200]     0.308544
(200, 250]     0.001752
Name: sales_bins, dtype: float64
```

In [4]:

```
# Let us visualize the same
sales_count = pd.value_counts(sales_df.sales_bins)
sales_count.sort_values(ascending=True).plot(kind='barh', title='Sales distribut
ion', );
# sns.countplot(sales_count)
```



As we can see, almost 92% of sales are less than 100. Max, min and average sales are 231, 0 and 52.25 respectively.
So any prediction model has to deal with the skewness in the data appropriately.

How does sales vary across stores

Let us get a overview of sales distribution in the whole data.

In [5]:

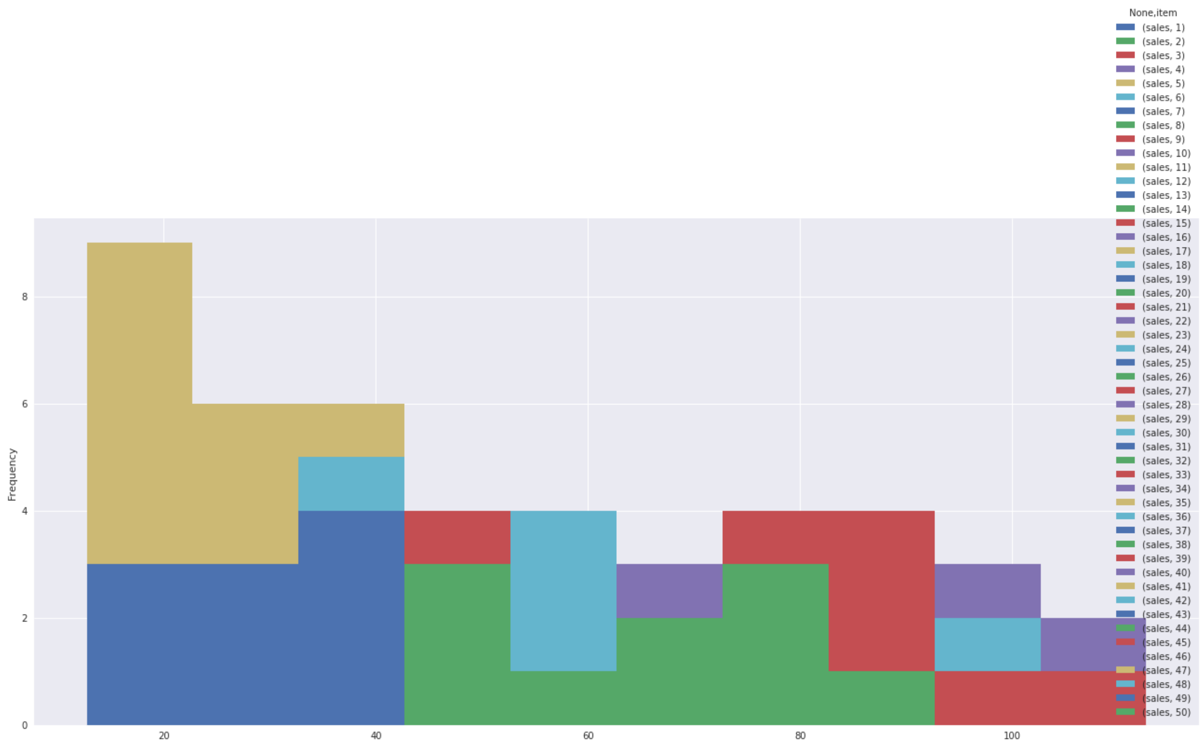
```
# Let us understand the sales data distribution across the stores
def sales_data_understanding(data):
    store_df = data.copy()
    plt.figure(figsize=(20,10))
    sales_pivoted_df = pd.pivot_table(store_df, index='store', values=['sales',
'date'], columns='item', aggfunc=np.mean)
    sales_pivoted_df.plot(kind="hist",figsize=(20,10))
    # Pivoted dataframe
    display(sales_pivoted_df)
    return (store_df,sales_pivoted_df)

store_df,sales_pivoted_df = sales_data_understanding(train_df)
```

	sales						
item	1	2	3	4	5	6	7
store							
1	19.971522	53.148959	33.208105	19.956188	16.612815	53.060789	52.78368
2	28.173604	75.316539	46.992333	28.234940	23.540526	74.945235	75.05859
3	25.070099	66.804491	41.771084	25.116101	20.857612	67.007119	66.64786
4	22.938664	61.715225	38.548193	23.086528	19.525192	61.270537	61.62541
5	16.739321	44.488499	27.835706	16.776561	14.086528	44.564622	44.53559
6	16.717963	44.533954	27.811062	16.754107	13.893209	44.503834	44.59912
7	15.159365	40.717963	25.531216	15.358160	12.733844	40.703724	40.70974
8	26.983571	71.656627	45.076123	26.948521	22.427711	71.958379	71.73055
9	23.325849	61.792442	38.535049	23.150055	19.272180	61.412377	61.81215
10	24.736035	65.566813	41.113363	24.721249	20.637459	65.612267	65.80777

10 rows × 50 columns

```
<matplotlib.figure.Figure at 0x7f7c6954cd30>
```



This pivoted dataframe has average sales per each store per each item.
Let use this dataframe and produce some interesting visualizations!

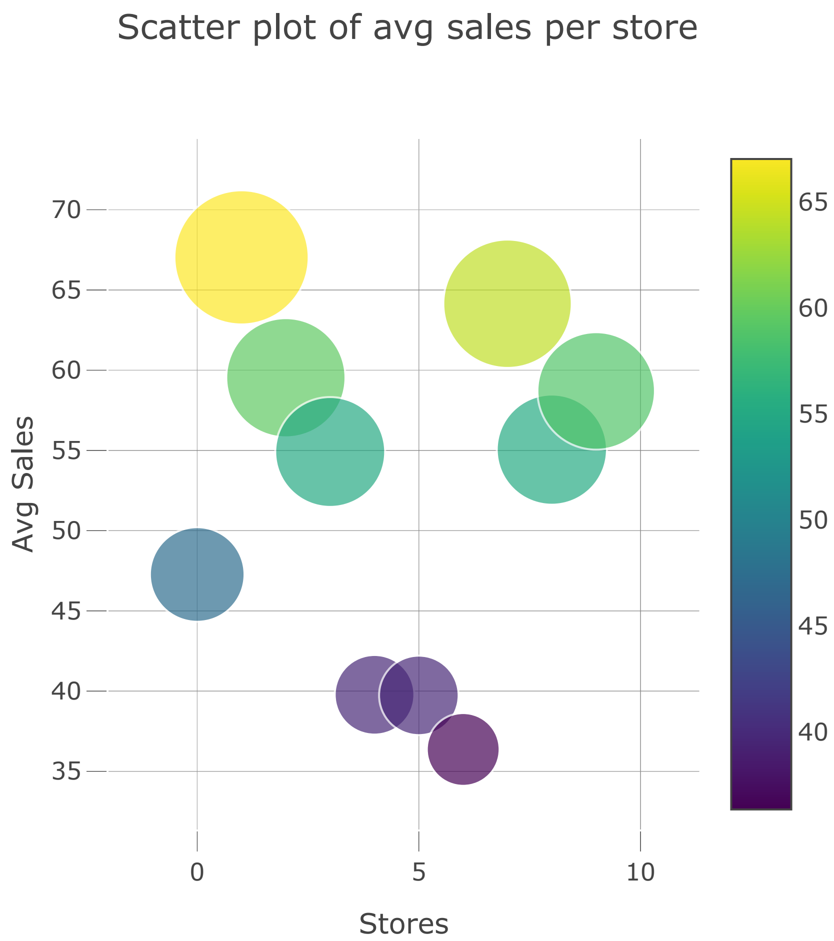
In [6]:

```
# Let us calculate the average sales of all the items by each store
sales_across_store_df = sales_pivoted_df.copy()
sales_across_store_df['avg_sale'] = sales_across_store_df.apply(lambda r: r.mean
(), axis=1)
```

In [7]:

```
# Scatter plot of average sales per store
sales_store_data = go.Scatter(
    y = sales_across_store_df.avg_sale.values,
    mode='markers',
    marker=dict(
        size = sales_across_store_df.avg_sale.values,
        color = sales_across_store_df.avg_sale.values,
        colorscale='Viridis',
        showscale=True
    ),
    text = sales_across_store_df.index.values
)
data = [sales_store_data]

sales_store_layout = go.Layout(
    autosize= True,
    title= 'Scatter plot of avg sales per store',
    hovermode= 'closest',
    xaxis= dict(
        title= 'Stores',
        ticklen= 10,
        zeroline= False,
        gridwidth= 1,
    ),
    yaxis=dict(
        title= 'Avg Sales',
        ticklen= 10,
        zeroline= False,
        gridwidth= 1,
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=sales_store_layout)
py.iplot(fig,filename='scatter_sales_store')
```



Export to plot.ly »

From the visualization, it is clear that the stores with ID 2 and 8 have higher average sales than the remaining stores and is a clear indication that they are doing good money!

Whereas store with ID 7 has very poor performance in terms of average sales.

How does sales vary across items

In [8]:

```
def sales_insight(sales_pivoted_df):
    # Let us calculate the average sales of each of the item across all the stores
    sales_across_item_df = sales_pivoted_df.copy()
    # Aggregate the sales per item and add it as a new row in the same dataframe
    sales_across_item_df.loc[11] = sales_across_item_df.apply(lambda r: r.mean
(), axis=0)
    # Note the 11th index row, which is the average sale of each of the item acros
s all the stores
    #display(sales_across_item_df.loc[11:])
    avg_sales_per_item_across_stores_df = pd.DataFrame(data=[[i+1,a] for i,a in
enumerate(sales_across_item_df.loc[11:].values[0])], columns=['item', 'avg_sale'
])
    # And finally, sort by avg sale
    avg_sales_per_item_across_stores_df.sort_values(by='avg_sale', ascending=Fal
se, inplace=True)
    # Display the top 10 rows
    display(avg_sales_per_item_across_stores_df.head())
    return (sales_across_item_df,avg_sales_per_item_across_stores_df)

sales_across_item_df,avg_sales_per_item_across_stores_df = sales_insight(sales_p
ivoted_df)
```

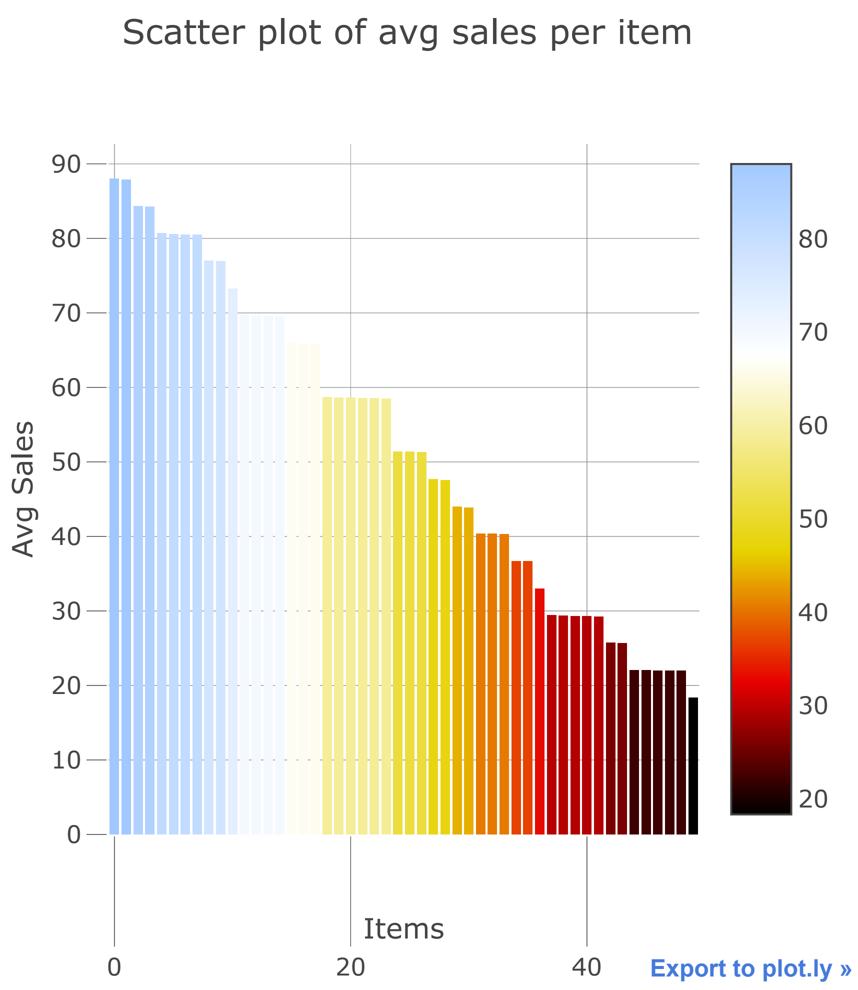
	item	avg_sale
14	15	88.030778
27	28	87.881325
12	13	84.316594
17	18	84.275794
24	25	80.686418

Great! Let us visualize these average sales per item!

In [9]:

```
avg_sales_per_item_across_stores_sorted = avg_sales_per_item_across_stores_df.av
g_sale.values
# Scatter plot of average sales per item
sales_item_data = go.Bar(
    x=[i for i in range(0, 50)],
    y=avg_sales_per_item_across_stores_sorted,
    marker=dict(
        color=avg_sales_per_item_across_stores_sorted,
        colorscale='Blackbody',
        showscale=True
    ),
    text = avg_sales_per_item_across_stores_df.item.values
)
data = [sales_item_data]

sales_item_layout = go.Layout(
    autosize= True,
    title= 'Scatter plot of avg sales per item',
    hovermode= 'closest',
    xaxis= dict(
        title= 'Items',
        ticklen= 55,
        zeroline= False,
        gridwidth= 1,
    ),
    yaxis=dict(
        title= 'Avg Sales',
        ticklen= 10,
        zeroline= False,
        gridwidth= 1,
    ),
    showlegend= False
)
fig = go.Figure(data=data, layout=sales_item_layout)
py.iplot(fig,filename='scatter_sales_item')
```



Amazing! The sales is uniformly distributed across all the items.
Top items with highest average sale are 15, 28, 13, 18 and with least average sales are 5, 1, 41 and so on.

Time-series visualization of the sales

Let us see how sales of a given item in a given store varies in a span of 5 years.

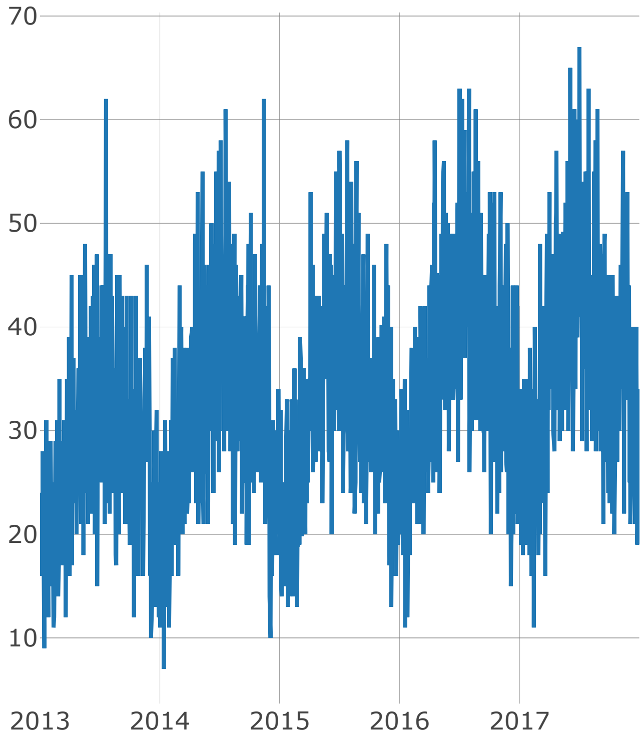
In [10]:

```
def Time_visualization(data):
    store_item_df = data.copy()
    # First, let us filterout the required data
    store_id = 10   # Some store
    item_id = 40    # Some item
    print('Before filter:', store_item_df.shape)
    store_item_df = store_item_df[store_item_df.store == store_id]
    store_item_df = store_item_df[store_item_df.item == item_id]
    print('After filter:', store_item_df.shape)
    #display(store_item_df.head())

    # Let us plot this now
    store_item_ts_data = [go.Scatter(
        x=store_item_df.date,
        y=store_item_df.sales)]
    py.iplot(store_item_ts_data)
    return store_item_df

store_item_df = Time_visualization(train_df)
```

```
Before filter: (913000, 4)
After filter: (1826, 4)
```



Export to plot.ly »

Woww! Clearly there is a pattern here! Feel free to play around with different store and item IDs. Almost all the items and store combination has this pattern!

The sales go high in June, July and August months. The sales will be lowest in December, January and February months. That's something!!

Let us make it more interesting. What if we aggregate the sales on a montly basis and compare different items and stores.
This should help us understand how different item sales behave at a high level.

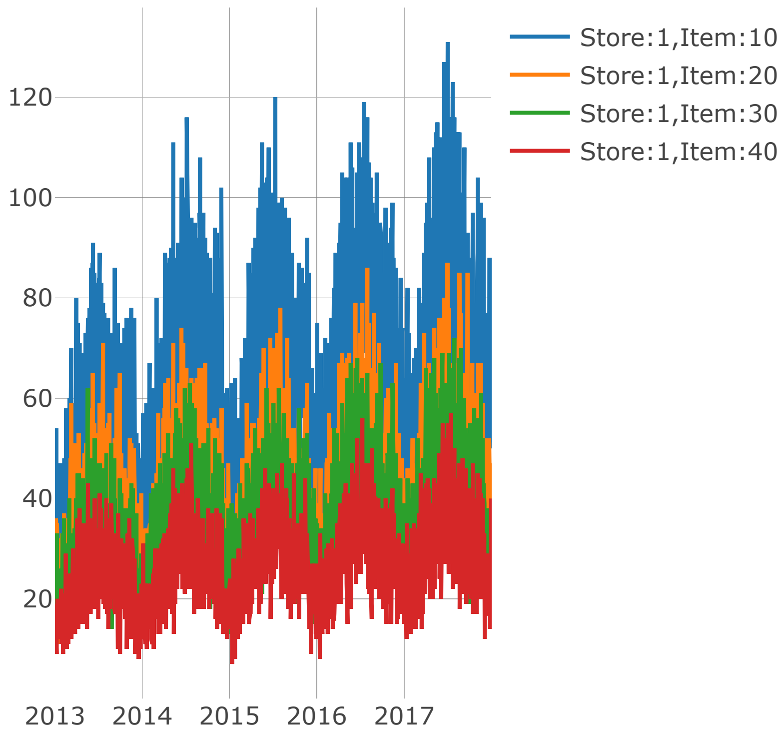
In [11]:

```
def sales_monthly(data):
    multi_store_item_df = data.copy()
    # First, let us filterout the required data
    store_ids = [1, 1, 1, 1]   # Some stores
    item_ids = [10, 20, 30, 40]    # Some items
    print('Before filter:', multi_store_item_df.shape)
    multi_store_item_df = multi_store_item_df[multi_store_item_df.store.isin(sto
re_ids)]
    multi_store_item_df = multi_store_item_df[multi_store_item_df.item.isin(item
_ids)]
    print('After filter:', multi_store_item_df.shape)
    #display(multi_store_item_df)
    # TODO Monthly avg sales

    # Let us plot this now
    multi_store_item_ts_data = []
    for st,it in zip(store_ids, item_ids):
        flt = multi_store_item_df[multi_store_item_df.store == st]
        flt = flt[flt.item == it]
        multi_store_item_ts_data.append(go.Scatter(x=flt.date, y=flt.sales, name
= "Store:" + str(st) + ",Item:" + str(it)))
    py.iplot(multi_store_item_ts_data)
    return (multi_store_item_df)

multi_store_item_df = sales_monthly(train_df)
```

```
Before filter: (913000, 4)
After filter: (7304, 4)
```



Export to plot.ly »

Interesting!!
Though the pattern remains same across different stores and items combinations, the **actual sale value consitently varies with the same scale**.

As we can see in the visualization, item 10 has consistently highest sales through out the span of 5 years!
This is an interesting behaviour that can be seen across almost all the items.

ARIMA

ARIMA is Autoregressive Integrated Moving Average Model, which is a component of SARIMAX, i.e. Seasonal ARIMA with eXogenous regressors.

(sources: 1 (https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/), 2 (https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-arima-in-python-3), 3 (http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases))

http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases (http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases)

LIGHTGBM

In [12]:

```
def split_data(train_data,test_data):
    train_data['date'] = pd.to_datetime(train_data['date'])
    test_data['date'] = pd.to_datetime(test_data['date'])

    train_data['month'] = train_data['date'].dt.month
    train_data['day'] = train_data['date'].dt.dayofweek
    train_data['year'] = train_data['date'].dt.year

    test_data['month'] = test_data['date'].dt.month
    test_data['day'] = test_data['date'].dt.dayofweek
    test_data['year'] = test_data['date'].dt.year

    col = [i for i in test_data.columns if i not in ['date','id']]
    y = 'sales'
    train_x, test_x, train_y, test_y = train_test_split(train_data[col],train_da
ta[y], test_size=0.2, random_state=2018)
    return (train_x, test_x, train_y, test_y,col)

train_x, test_x, train_y, test_y,col = split_data(train_df,test_df)
```

In [13]:

```
train_x.shape, test_x.shape
```

Out[13]:

```
((730400, 5), (182600, 5))
```

In [14]:

```
# reshape input to be [samples, time steps, features]
train_x = np.array(train_x).reshape(train_x.shape[0], 1, train_x.shape[1])
test_x = np.array(test_x).reshape(test_x.shape[0], 1, test_x.shape[1])
train_x.shape,test_x.shape
```

Out[14]:

```
((730400, 1, 5), (182600, 1, 5))
```

In [15]:

```
_optimiser = ['Adam','Nadam','RMSprop']
model = Sequential()
model.add(LSTM(144, batch_input_shape=(32, 1, 5), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer=_optimiser[0])
model.summary()
model.fit(train_x,train_y, batch_size=32,epochs=5)
submission = pd.read_csv("../input/demand-forecasting-kernels-only/sample_submis
sion.csv")
submission['sales'] = model.predict(test_x)
submission.to_csv("submission_Adam")
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (32, 144)                 86400
_________________________________________________________________
dense_1 (Dense)              (32, 1)                   145
=================================================================
Total params: 86,545
Trainable params: 86,545
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
730400/730400 [==============================] - 121s 165us/step -
loss: 890.5505
Epoch 2/5
730400/730400 [==============================] - 117s 160us/step -
loss: 830.1061
Epoch 3/5
730400/730400 [==============================] - 116s 159us/step -
loss: 830.0956
Epoch 4/5
730400/730400 [==============================] - 115s 157us/step -
loss: 830.1040
Epoch 5/5
730400/730400 [==============================] - 114s 157us/step -
loss: 830.1174
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-253b2f67d2e3> in <module>()
      7 model.fit(train_x,train_y, batch_size=32,epochs=5)
      8 submission = pd.read_csv("../input/demand-forecasting-kernels-only/sample_submission.csv")
----> 9 submission['sales'] = model.predict(test_x)
     10 submission.to_csv("submission_Adam")

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/models.py in predict(self, x, batch_size, verbose, steps)
   1023             self.build()
   1024         return self.model.predict(x, batch_size=batch_size, verbose=verbose,
-> 1025                                   steps=steps)
   1026 
   1027     def predict_on_batch(self, x):

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/engine/training.py in predict(self, x, batch_size, verbose, steps)
   1823                                  'divided by the batch size. Found: ' +
   1824                                  str(x[0].shape[0]) + ' samples. '
-> 1825                                  'Batch size: ' + str(batch_size) + '.')
   1826 
   1827         # Prepare inputs, delegate logic to `_predict_loop`.

ValueError: In a stateful network, you should only pass inputs with a number of samples that can be divided by the batch size. Found: 182600 samples. Batch size: 32.
```

In [16]:

```
_optimiser = ['Adam','Nadam','RMSprop']
model = Sequential()
model.add(LSTM(144, batch_input_shape=(32, 1, 5), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer=_optimiser[1])
model.summary()
model.fit(train_x,train_y, batch_size=40,epochs=5)
submission = pd.read_csv("../input/demand-forecasting-kernels-only/sample_submis
sion.csv")
submission['sales'] = model.predict(test_x, batch_size=32,verbose = 1)
submission.to_csv("submission_Nadam")
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (32, 144)                 86400
_________________________________________________________________
dense_2 (Dense)              (32, 1)                   145
=================================================================
Total params: 86,545
Trainable params: 86,545
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-f7745119629e> in <module>()
      5 model.compile(loss='mean_squared_error', optimizer=_optimiser[1])
      6 model.summary()
----> 7 model.fit(train_x,train_y, batch_size=40,epochs=5)
      8 submission = pd.read_csv("../input/demand-forecasting-kernels-only/sample_submission.csv")
      9 submission['sales'] = model.predict(test_x, batch_size=32,verbose = 1)

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/models.py in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, **kwargs)
    961                               initial_epoch=initial_epoch,
    962                               steps_per_epoch=steps_per_epoch,
--> 963                               validation_steps=validation_steps)
    964 
    965     def evaluate(self, x=None, y=None,

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/engine/training.py in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, **kwargs)
   1703                               initial_epoch=initial_epoch,
   1704                               steps_per_epoch=steps_per_epoch,
-> 1705                               validation_steps=validation_steps)
   1706 
   1707     def evaluate(self, x=None, y=None,

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/engine/training.py in _fit_loop(self, f, ins, out_labels, batch_size, epochs, verbose, callbacks, val_f, val_ins, shuffle, callback_metrics, initial_epoch, steps_per_epoch, validation_steps)
```

```
   1233                             ins_batch[i] = ins_batch[i].toarray
()
   1234
-> 1235                         outs = f(ins_batch)
   1236                         if not isinstance(outs, list):
   1237                             outs = [outs]

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/
backend/tensorflow_backend.py in __call__(self, inputs)
   2477         session = get_session()
   2478         updated = session.run(fetches=fetches, feed_dict=fe
ed_dict,
-> 2479                                   **self.session_kwargs)
   2480         return updated[:len(self.outputs)]
   2481

/opt/conda/lib/python3.6/site-packages/tensorflow/python/client/ses
sion.py in run(self, fetches, feed_dict, options, run_metadata)
    906     try:
    907       result = self._run(None, fetches, feed_dict, options_
ptr,
--> 908                          run_metadata_ptr)
    909       if run_metadata:
    910         proto_data = tf_session.TF_GetBuffer(run_metadata_p
tr)

/opt/conda/lib/python3.6/site-packages/tensorflow/python/client/ses
sion.py in _run(self, handle, fetches, feed_dict, options, run_meta
data)
   1117                                 'which has shape %r' %
   1118                                 (np_val.shape, subfeed_t.name,
-> 1119                                  str(subfeed_t.get_shape())))
   1120           if not self.graph.is_feedable(subfeed_t):
   1121             raise ValueError('Tensor %s may not be fed.' %
 subfeed_t)

ValueError: Cannot feed value of shape (40, 1, 5) for Tensor 'lstm_
2_input:0', which has shape '(32, 1, 5)'
```

In [17]:

```
_optimiser = ['Adam','Nadam','RMSprop']
model = Sequential()
model.add(LSTM(144, batch_input_shape=(32, 1, 5), stateful=True))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer=_optimiser[2])
model.summary()
model.fit(train_x,train_y, batch_size=40,epochs=5)
submission = pd.read_csv("../input/demand-forecasting-kernels-only/sample_submis
sion.csv")
submission['sales'] = model.predict(test_x, batch_size=32,verbose = 1)
submission.to_csv("submission_RMS")
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_3 (LSTM)                (32, 144)                 86400     
_________________________________________________________________
dense_3 (Dense)              (32, 1)                   145       
=================================================================
Total params: 86,545
Trainable params: 86,545
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-e422ead273e9> in <module>()
      5 model.compile(loss='mean_squared_error', optimizer=_optimiser[2])
      6 model.summary()
----> 7 model.fit(train_x,train_y, batch_size=40,epochs=5)
      8 submission = pd.read_csv("../input/demand-forecasting-kernels-only/sample_submission.csv")
      9 submission['sales'] = model.predict(test_x, batch_size=32,verbose = 1)

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/models.py in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, **kwargs)
    961                               initial_epoch=initial_epoch,
    962                               steps_per_epoch=steps_per_epoch,
--> 963                               validation_steps=validation_steps)
    964 
    965     def evaluate(self, x=None, y=None,

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/engine/training.py in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, **kwargs)
   1703                               initial_epoch=initial_epoch,
   1704                               steps_per_epoch=steps_per_epoch,
-> 1705                               validation_steps=validation_steps)
   1706 
   1707     def evaluate(self, x=None, y=None,

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/engine/training.py in _fit_loop(self, f, ins, out_labels, batch_size, epochs, verbose, callbacks, val_f, val_ins, shuffle, callback_metrics, initial_epoch, steps_per_epoch, validation_steps)
```

```
   1233                             ins_batch[i] = ins_batch[i].toarray
()
   1234
-> 1235                         outs = f(ins_batch)
   1236                         if not isinstance(outs, list):
   1237                             outs = [outs]

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/
backend/tensorflow_backend.py in __call__(self, inputs)
   2477         session = get_session()
   2478         updated = session.run(fetches=fetches, feed_dict=fe
ed_dict,
-> 2479                                 **self.session_kwargs)
   2480         return updated[:len(self.outputs)]
   2481

/opt/conda/lib/python3.6/site-packages/tensorflow/python/client/ses
sion.py in run(self, fetches, feed_dict, options, run_metadata)
    906     try:
    907       result = self._run(None, fetches, feed_dict, options_
ptr,
--> 908                          run_metadata_ptr)
    909       if run_metadata:
    910         proto_data = tf_session.TF_GetBuffer(run_metadata_p
tr)

/opt/conda/lib/python3.6/site-packages/tensorflow/python/client/ses
sion.py in _run(self, handle, fetches, feed_dict, options, run_meta
data)
   1117                                 'which has shape %r' %
   1118                                 (np_val.shape, subfeed_t.name,
-> 1119                                  str(subfeed_t.get_shape())))
   1120           if not self.graph.is_feedable(subfeed_t):
   1121             raise ValueError('Tensor %s may not be fed.' %
 subfeed_t)

ValueError: Cannot feed value of shape (40, 1, 5) for Tensor 'lstm_
3_input:0', which has shape '(32, 1, 5)'
```

In [18]:

```
# from bayes_opt import BayesianOptimization
# def bayes_parameter_opt_lgb(X, y, init_round=15, opt_round=25, n_folds=5, random
_seed=6, n_estimators=10000, learning_rate=0.02, output_process=False):
#     # prepare data
#     train_data = lgb.Dataset(data=X, label=y)
#     # parameters
#     def lgb_eval(num_leaves, feature_fraction, bagging_fraction, max_depth, lamb
da_l1, lambda_l2, min_split_gain, min_child_weight):
#         params = {'application':'regression_l1','num_iterations': n_estimators,
 'learning_rate':learning_rate, 'early_stopping_round':100, 'metric':'auc'}
#         params["num_leaves"] = int(round(num_leaves))
#         params['feature_fraction'] = max(min(feature_fraction, 1), 0)
#         params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
#         params['max_depth'] = int(round(max_depth))
#         params['lambda_l1'] = max(lambda_l1, 0)
#         params['lambda_l2'] = max(lambda_l2, 0)
#         params['min_split_gain'] = min_split_gain
#         params['min_child_weight'] = min_child_weight
#         cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed,
 stratified=True, verbose_eval =200, metrics=['auc'])
#         return max(cv_result['auc-mean'])
#     # range
#     lgbBO = BayesianOptimization(lgb_eval, {'num_leaves': (24, 45),
#                                              'feature_fraction': (0.1, 0.9),
#                                              'bagging_fraction': (0.8, 1),
#                                              'max_depth': (5, 8.99),
#                                              'lambda_l1': (0, 5),
#                                              'lambda_l2': (0, 3),
#                                              'min_split_gain': (0.001, 0.1),
#                                              'min_child_weight': (5, 50)}, random
_state=0)
#     # optimize
#     lgbBO.maximize(init_points=init_round, n_iter=opt_round)

#     # output optimization process
#     if output_process==True: lgbBO.points_to_csv("bayes_opt_result.csv")

#     # return best parameters
#     return lgbBO.res['max']['max_params']

# opt_params = bayes_parameter_opt_lgb(train_x, train_y, init_round=5, opt_round=1
0, n_folds=3, random_seed=6, n_estimators=100, learning_rate=0.02)
```

In [19]:

```
# opt_params
```

In [20]:

```
sample_df['sales'] = model.predict(test_x)
sample_df.to_csv('lgb_bayasian_param.csv', index=False)
sample_df['sales'].head()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent ca
ll last)
<ipython-input-20-1481181b7ae0> in <module>()
----> 1 sample_df['sales'] = model.predict(test_x)
      2 sample_df.to_csv('lgb_bayasian_param.csv', index=False)
      3 sample_df['sales'].head()

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/
models.py in predict(self, x, batch_size, verbose, steps)
   1023             self.build()
   1024         return self.model.predict(x, batch_size=batch_size,
verbose=verbose,
-> 1025                                   steps=steps)
   1026 
   1027     def predict_on_batch(self, x):

/opt/conda/lib/python3.6/site-packages/Keras-2.1.5-py3.6.egg/keras/
engine/training.py in predict(self, x, batch_size, verbose, steps)
   1823                                  'divided by the batch siz
e. Found: ' +
   1824                                  str(x[0].shape[0]) + ' sam
ples. '
-> 1825                                  'Batch size: ' + str(batch
_size) + '.')
   1826 
   1827         # Prepare inputs, delegate logic to `_predict_loop
`.

ValueError: In a stateful network, you should only pass inputs with
a number of samples that can be divided by the batch size. Found: 1
82600 samples. Batch size: 32.
```

In [21]:

```
def average(df1):
    avg  = df1
    df2 = pd.read_csv("../input/private/sub_val-0.132358565029612.csv")
    avg['sales'] = (df1["sales"]*0.3 + df2["sales"]*0.7)
    return avg

avg = average(sample_df)
avg.to_csv("Submission.csv", index=False)
```