Table of Contents

1. Data Objective

The objective of the M5 forecasting competition is to advance the theory and practice of forecasting by identifying the method(s) that provide the most accurate point forecasts for each of the 42,840 time series of the competition. I addition, to elicit information to estimate the uncertainty distribution of the realized values of these series as precisely as possible.

To that end, the participants of M5 are asked to provide 28 days ahead point forecasts (PFs) for all the series of the competition, as well as the corresponding median and 50%, 67%, 95%, and 99% prediction intervals (PIs). The M5 differs from the previous four ones in five important ways, some of them suggested by the discussants of the M4 competition, as follows:

- First, it uses grouped unit sales data, starting at the product-store level and being aggregated to that of product departments, product categories, stores, and three geographical areas: the States of California (CA), Texas (TX), and Wisconsin (WI).
- Second, besides the time series data, it includes explanatory variables such as sell prices, promotions, days of the week, and special events (e.g. Super Bowl, Valentine's Day, and Orthodox Easter) that typically affect unit sales and could improve forecasting accuracy.
- Third, in addition to point forecasts, it assesses the distribution of uncertainty, as the participants are asked to provide information on nine indicative quantiles.

- Fourth, instead of having a single competition to estimate both the point forecasts and the uncertainty distribution, there will be two parallel tracks using the same dataset, the first requiring 28 days ahead point forecasts and the second 28 days ahead probabilistic forecasts for the median and four prediction intervals (50%, 67%, 95%, and 99%).
- Fifth, for the first time it focuses on series that display intermittency, i.e., sporadic demand including zeros.

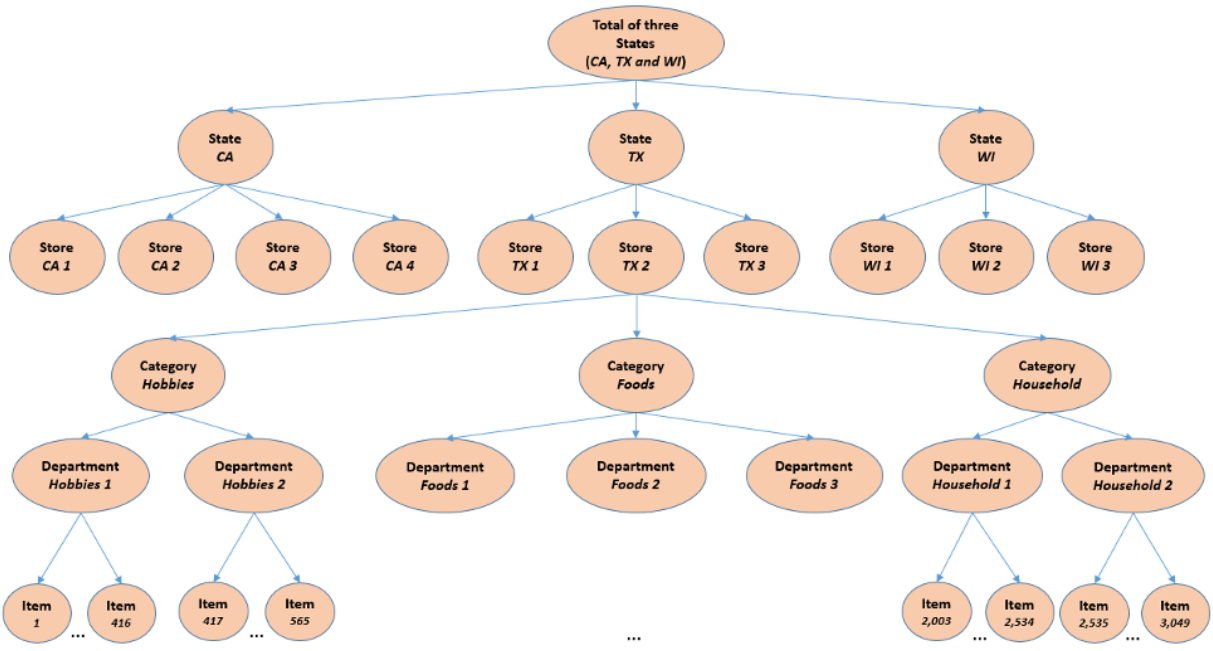
1.1. Dates and hosting

The M5 will start on March 2, 2020 and finish on June 30 of the same year. The competition will be run using the Kaggle platform. Thus, we expect many submissions from all types of forecasters including data scientists, statisticians, and practitioners, expanding the field of forecasting and eventually integrating its various approaches for improving accuracy and uncertainty estimation.

The competition will be divided into two separate Kaggle competitions, using the same dataset, with the first (M5 Forecasting Competition – Accuracy) requiring 28 days ahead point forecasts and the second (M5 Forecasting Competition – Uncertainty) 28 days ahead probabilistic forecasts for the corresponding median and four prediction intervals (50%, 67%, 95%, and 99%).

1.2. The dataset

The M5 dataset, generously made available by Walmart, involves the unit sales of various products sold in the USA, organized in the form of grouped time series. More specifically, the dataset involves the unit sales of 3,049 products, classified in 3 product categories (Hobbies, Foods, and Household) and 7 product departments, in which the above-mentioned categories are disaggregated. The products are sold across ten stores, located in three States (CA, TX, and WI). In this respect, the bottom-level of the hierarchy, i.e., product-store unit sales can be mapped across either product categories or geographical regions



The historical data range from 2011-01-29 to 2016-06-19. Thus, the products have a (maximum) selling history of 1,941 days / 5.4 years (test data of h=28 days not included).

The M5 dataset consists of the following four (4) files:

File 1: “calendar.csv”

Contains information about the dates the products are sold.

- date: The date in a “y-m-d” format.
- wm_yr_wk: The id of the week the date belongs to.
- weekday: The type of the day (Saturday, Sunday, …, Friday).
- wday: The id of the weekday, starting from Saturday.
- month: The month of the date.
- year: The year of the date.
- event_name_1: If the date includes an event, the name of this event.
- event_type_1: If the date includes an event, the type of this event.
- event_name_2: If the date includes a second event, the name of this event.
- event_type_2: If the date includes a second event, the type of this event.
- snap_CA, snap_TX, and snap_WI: A binary variable (0 or 1) indicating whether the stores of CA, TX or WI allow SNAP purchases on the examined date. 1 indicates that SNAP purchases are allowed.

File 2: “sell_prices.csv”

Contains information about the price of the products sold per store and date.

- store_id: The id of the store where the product is sold.
- item_id: The id of the product.
- wm_yr_wk: The id of the week.
- sell_price: The price of the product for the given week/store. The price is provided per week (average across seven days). If not available, this means that the product was not sold during the examined week. Note that although prices are constant at weekly basis, they may change through time (both training and test set).

File 3: “sales_train_valid.csv”

Contains the historical daily unit sales data per product and store.

- item_id: The id of the product.
- dept_id: The id of the department the product belongs to.
- cat_id: The id of the category the product belongs to.
- store_id: The id of the store where the product is sold.
- state_id: The State where the store is located.
- d_1, d_2, …, d_i, … d_1941: The number of units sold at day i, starting from 2011-01-29.

File 4: “sales_train_evaluation.csv”

Contains the historical daily unit sales data per product and store.

- item_id: The id of the product.
- dept_id: The id of the department the product belongs to.
- cat_id: The id of the category the product belongs to.
- store_id: The id of the store where the product is sold.
- state_id: The State where the store is located.
- d_1, d_2, …, d_i, … d_1969: The number of units sold at day i, starting from 2011-01-29.

2. Fetch the data

2.1. Import libraries

In [1]:
```
import os, gc

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error

import lightgbm as lgb

import plotly_express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_rows', 500)
for dirname, _, filenames in os.walk('data/m5-forecasting-accuracy/'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

executed in 5.72s, finished 12:05:53 2020-07-17

```
data/m5-forecasting-accuracy/calendar.csv
data/m5-forecasting-accuracy/sales_train_evaluation.csv
data/m5-forecasting-accuracy/sales_train_validation.csv
data/m5-forecasting-accuracy/sample_submission.csv
data/m5-forecasting-accuracy/sell_prices.csv
```

2.2. Setting working directory and loading datasets

In [2]:
```
# working path
path = 'data/m5-forecasting-accuracy/'
```
executed in 13ms, finished 12:05:53 2020-07-17

In [3]:
```
# loading datasets
calendar = pd.read_csv(path + 'calendar.csv')
sellPrice = pd.read_csv(path + 'sell_prices.csv')
sales = pd.read_csv(path + 'sales_train_evaluation.csv')
sampleSubmission = pd.read_csv(path + 'sample_submission.csv')
```
executed in 6.37s, finished 12:06:00 2020-07-17

In [4]:
```
print('Calendar dataset has {} rows and {} columns'.format(calendar.shape[0], ca
print('Sell Price dataset has {} rows and {} columns'.format(sellPrice.shape[0],
print('Sales dataset has {} rows and {} columns'.format(sales.shape[0], sales.sh
print('Sample Submission dataset has {} rows and {} columns'.format(sampleSubmis
```
executed in 13ms, finished 12:06:00 2020-07-17

```
Calendar dataset has 1969 rows and 14 columns
Sell Price dataset has 6841121 rows and 4 columns
Sales dataset has 30490 rows and 1947 columns
Sample Submission dataset has 60980 rows and 29 columns
```

In [5]:
```
# Function using to get number of rows and data types
def infoData(df):
    columns = []
    values = []
    for feature in df.columns:
        columns.append(feature)
        values.append(len(df[feature].unique()))
    percent_missing = df.isnull().sum() * 100 / len(df)
    data = {'Unique_Value': values,'Percent_Missing': percent_missing, 'Data_Typ
    return pd.DataFrame(data=data)
```
executed in 15ms, finished 12:06:00 2020-07-17

In [6]:
```
# Downcast in order to save memory
def reduce_mem_usage(df, verbose=True):
    start_mem = df.memory_usage().sum() / 1024**2
    cols = df.dtypes.index.tolist()
    types = df.dtypes.values.tolist()
    for i,t in enumerate(types):
        if 'int' in str(t):
            if df[cols[i]].min() > np.iinfo(np.int8).min and df[cols[i]].max() <
                df[cols[i]] = df[cols[i]].astype(np.int8)
            elif df[cols[i]].min() > np.iinfo(np.int16).min and df[cols[i]].max(
                df[cols[i]] = df[cols[i]].astype(np.int16)
            elif df[cols[i]].min() > np.iinfo(np.int32).min and df[cols[i]].max(
                df[cols[i]] = df[cols[i]].astype(np.int32)
            else:
                df[cols[i]] = df[cols[i]].astype(np.int64)
        elif 'float' in str(t):
            if df[cols[i]].min() > np.finfo(np.float16).min and df[cols[i]].max(
                df[cols[i]] = df[cols[i]].astype(np.float16)
            elif df[cols[i]].min() > np.finfo(np.float32).min and df[cols[i]].ma
                df[cols[i]] = df[cols[i]].astype(np.float32)
            else:
                df[cols[i]] = df[cols[i]].astype(np.float64)
        elif t == np.object:
            if cols[i] == 'date':
                df[cols[i]] = pd.to_datetime(df[cols[i]], format='%Y-%m-%d')
    end_mem = df.memory_usage().sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.f
    return df
```

executed in 13ms, finished 12:06:00 2020-07-17

In [7]:
```
# reducing memory of datasets
calendar = reduce_mem_usage(calendar)
sellPrice = reduce_mem_usage(sellPrice)
sales = reduce_mem_usage(sales)
```

executed in 1m 32.1s, finished 12:07:32 2020-07-17

```
Mem. usage decreased to  0.12 Mb (41.9% reduction)
Mem. usage decreased to 130.48 Mb (37.5% reduction)
Mem. usage decreased to 96.13 Mb (78.8% reduction)
```

In [8]:
```
calendar.head()
```
executed in 29ms, finished 12:07:32 2020-07-17

Out[8]:

	date	wm_yr_wk	weekday	wday	month	year	d	event_name_1	event_type_1	event_nar
0	2011-01-29	11101	Saturday	1	1	2011	d_1	NaN	NaN	
1	2011-01-30	11101	Sunday	2	1	2011	d_2	NaN	NaN	
2	2011-01-31	11101	Monday	3	1	2011	d_3	NaN	NaN	
3	2011-02-01	11101	Tuesday	4	2	2011	d_4	NaN	NaN	
4	2011-02-02	11101	Wednesday	5	2	2011	d_5	NaN	NaN	

In [9]:
```
sellPrice.head()
```
executed in 12ms, finished 12:07:32 2020-07-17

Out[9]:

	store_id	item_id	wm_yr_wk	sell_price
0	CA_1	HOBBIES_1_001	11325	9.578125
1	CA_1	HOBBIES_1_001	11326	9.578125
2	CA_1	HOBBIES_1_001	11327	8.257812
3	CA_1	HOBBIES_1_001	11328	8.257812
4	CA_1	HOBBIES_1_001	11329	8.257812

In [10]:
```
sales.head()
```
executed in 231ms, finished 12:07:32 2020-07-17

Out[10]:

	id	item_id	dept_id	cat_id	store_id	state_id	d_1
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0

5 rows × 1947 columns

Validation data is now from 1914 to 1941. And test data is from 1942 to 1969

In [11]:
```
# adding more days to use as test datasets
for d in range(1942, 1970):
    col = 'd_' + str(d)
    sales[col] = 0
    sales[col] = sales[col].astype(np.int16)
```
executed in 121ms, finished 12:07:32 2020-07-17

In [12]:
```
sales.head()
```
executed in 153ms, finished 12:07:32 2020-07-17

Out[12]:

	id	item_id	dept_id	cat_id	store_id	state_id	d_1
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0

5 rows × 1975 columns

3. Melting the data

In this case what the melt function is doing is that it is converting the sales dataframe which is in wide format to a long format. I have kept the id variables as id, item_id, dept_id, cat_id, store_id and state_id. They have in total 30490 unique values when compunded together. Now the total number of days for which we have the data is 1969 days. Therefore the melted dataframe will be having 30490x1969 i.e. 60034810 rows

In [13]:
```
identifierVariables = ['id', 'item_id', 'dept_id', 'cat_id', 'store_id', 'state_
variableName = 'd'
measuredVariables = 'sales'

data = pd.melt(sales,
               id_vars = identifierVariables,
               var_name = variableName,
               value_name = measuredVariables).drop_duplicates()

del identifierVariables, variableName, measuredVariables
gc.collect()
print('Dataset has {} rows and {} columns'.format(data.shape[0], data.shape[1]))
```

executed in 50.3s, finished 12:08:23 2020-07-17

```
Dataset has 60034810 rows and 8 columns
```

In [14]:
```
data.head()
```

executed in 14ms, finished 12:08:23 2020-07-17

Out[14]:

	id	item_id	dept_id	cat_id	store_id	state_id	d
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	d_1
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	d_1
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	d_1
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	d_1
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	d_1

Combine price data from prices dataframe and days data from calendar dataset.

In [15]:
```
data = data.merge(calendar, how='left', on='d')
data = data.merge(sellPrice, on=['store_id', 'item_id', 'wm_yr_wk'], how='left')

del calendar, sellPrice
gc.collect()
print('Dataset has {} rows and {} columns'.format(data.shape[0], data.shape[1]))
```

executed in 55.1s, finished 12:09:18 2020-07-17

```
Dataset has 60034810 rows and 22 columns
```

In [16]:
```
data.head()
```

executed in 60ms, finished 12:09:18 2020-07-17

Out[16]:

	id	item_id	dept_id	cat_id	store_id	state_id	d
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	d_1
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	d_1
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	d_1
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	d_1
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	d_1

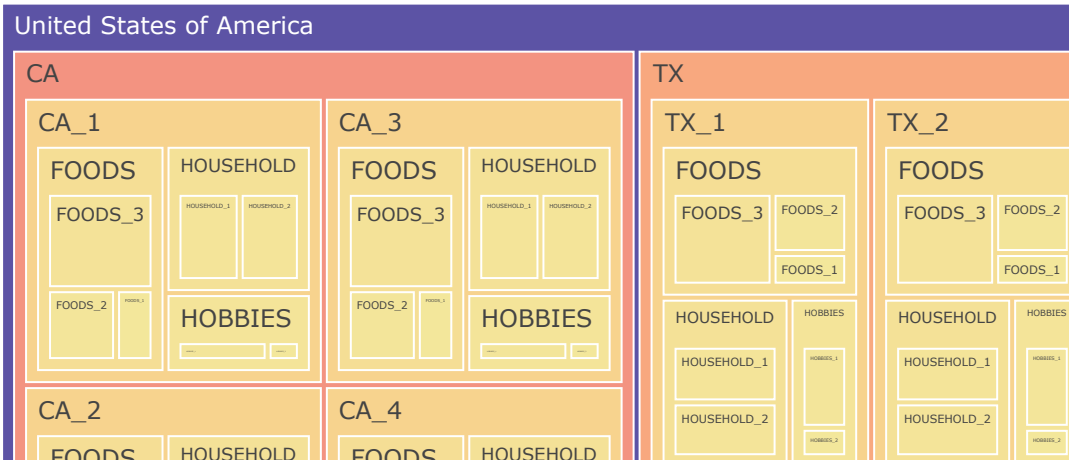
4. Exploratory Data Analysis

4.1 The Dataset

In [17]:
```
group = sales.groupby(['state_id','store_id','cat_id','dept_id'],as_index=False)
group['USA'] = 'United States of America'
group.rename(columns={'state_id':'State','store_id':'Store','cat_id':'Category',
             inplace=True)
fig = px.treemap(group, path=['USA', 'State', 'Store', 'Category', 'Department']
                 color='Count',
                 color_continuous_scale= px.colors.sequential.Sunset,
                 title='Walmart: Distribution of items')
fig.update_layout(template='seaborn')
fig.show()
```

executed in 1.77s, finished 12:09:20 2020-07-17

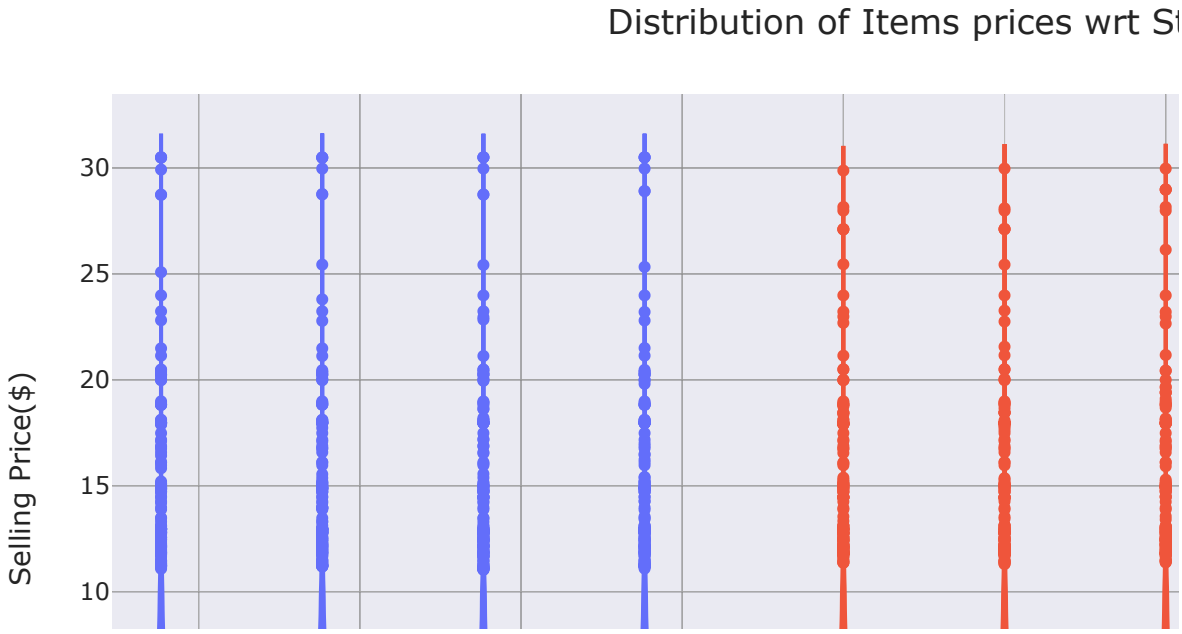
Walmart: Distribution of item



4.2 Item Prices

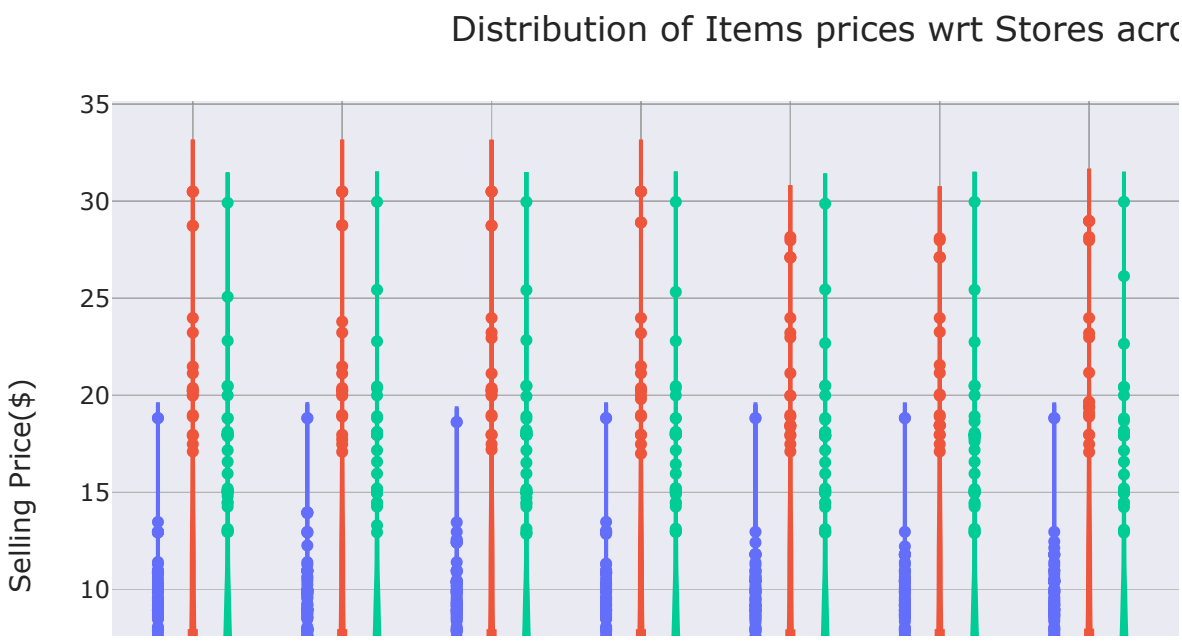
In [18]:
```
group_price_store = data.groupby(['state_id','store_id','item_id'], as_index=Fal
fig = px.violin(group_price_store, x='store_id', color='state_id', y='sell_price
fig.update_xaxes(title_text='Store')
fig.update_yaxes(title_text='Selling Price($)')
fig.update_layout(template='seaborn', title='Distribution of Items prices wrt St
fig.show()
```

executed in 32.9s, finished 12:09:52 2020-07-17



In [19]:
```
group_price_cat = data.groupby(['store_id', 'cat_id', 'item_id'],as_index=False)
fig = px.violin(group_price_cat, x='store_id', color='cat_id', y='sell_price', b
fig.update_xaxes(title_text='Store')
fig.update_yaxes(title_text='Selling Price($)')
fig.update_layout(template='seaborn',title='Distribution of Items prices wrt Sto
                  legend_title_text='Category')
fig.show()
```

executed in 8.20s, finished 12:10:01 2020-07-17

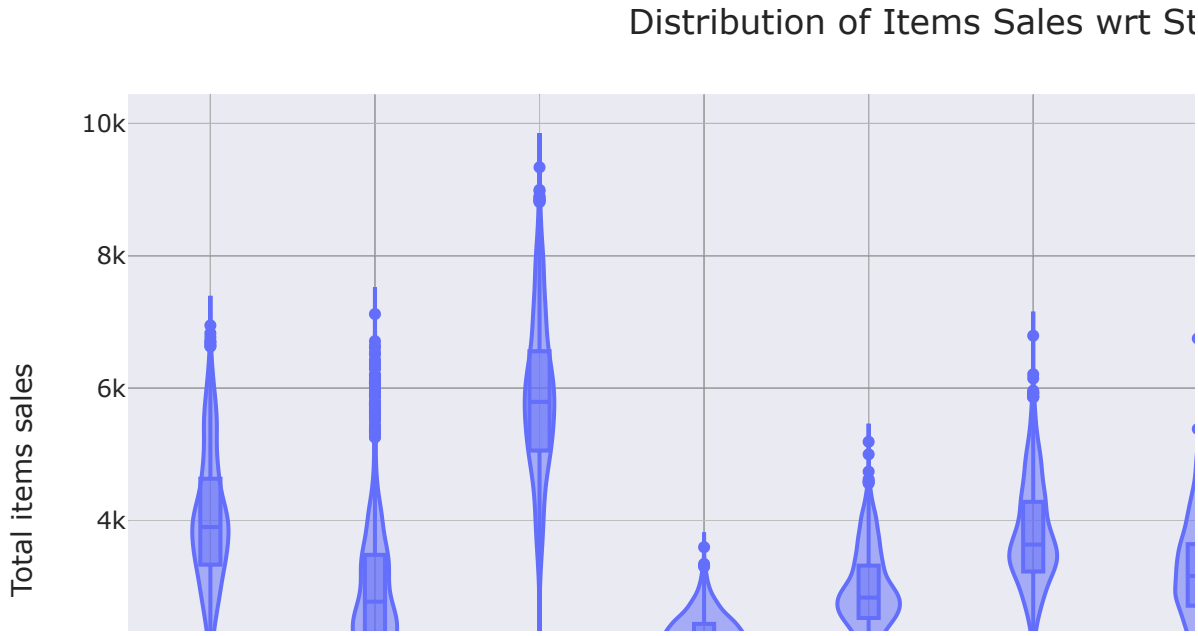


As can be seen from the plot above, food category items are quite cheap as compared with hobbies and household items. Hobbies and household items have almost the same price range.

4.3 Items Sales

In [20]:
```
group = data.groupby(['year','date', 'state_id', 'store_id'], as_index=False)['s
fig = px.violin(group, x='store_id', y='sales', box=True)
fig.update_xaxes(title_text='Store')
fig.update_yaxes(title_text='Total items sales')
fig.update_layout(template='seaborn', title='Distribution of Items Sales wrt Sto
fig.show()
```

executed in 6.93s, finished 12:10:08 2020-07-17



- California: CA_3 has sold the most number of items while, CA_4 has sold the least number of items.
- Texas: TX_2 and TX_3 have sold the maximum number of items. TX_1 has sold the least number of items.
- Wisconsin: WI_2 has sold the maximum number of items while, WI_3 has sold the least number of items.
- USA: CA_3 has sold the most number of items while, CA_4 has sold the least number of items.

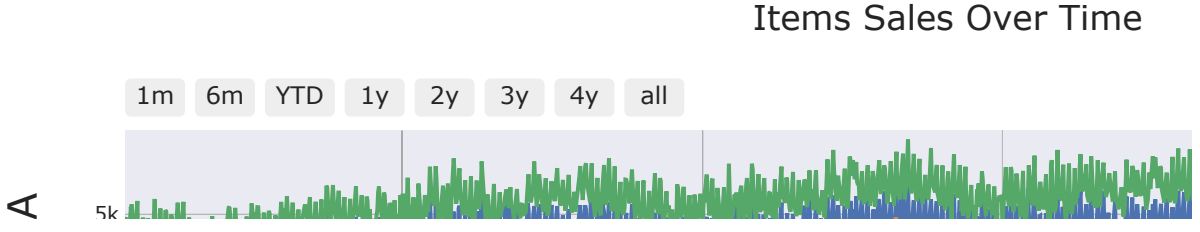
In [21]:

```
fig = go.Figure()
title = 'Items Sales Over Time'
years = group.year.unique().tolist()
buttons = []
y=3
for state in group.state_id.unique().tolist():
    group_state = group[group['state_id']==state]
    for store in group_state.store_id.unique().tolist():
        group_state_store = group_state[group_state['store_id']==store]
        fig.add_trace(go.Scatter(name=store, x=group_state_store['date'], y=grou
                                 yaxis='y'+str(y) if y!=1 else 'y'))
    y-=1

fig.update_layout(
        xaxis=dict(
        #autorange=True,
        range = ['2011-01-29','2016-05-22'],
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(count=2,
                     label="2y",
                     step="year",
                     stepmode="backward"),
                dict(count=3,
                     label="3y",
                     step="year",
                     stepmode="backward"),
                dict(count=4,
                     label="4y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            autorange=True,
        ),
        type="date"
    ),
    yaxis=dict(
```

```
        anchor="x",
        autorange=True,
        domain=[0, 0.33],
        mirror=True,
        showline=True,
        side="left",
        tickfont={"size":10},
        tickmode="auto",
        ticks="",
        title='WI',
        titlefont={"size":20},
        type="linear",
        zeroline=False
    ),
    yaxis2=dict(
        anchor="x",
        autorange=True,
        domain=[0.33, 0.66],
        mirror=True,
        showline=True,
        side="left",
        tickfont={"size":10},
        tickmode="auto",
        ticks="",
        title = 'TX',
        titlefont={"size":20},
        type="linear",
        zeroline=False
    ),
    yaxis3=dict(
        anchor="x",
        autorange=True,
        domain=[0.66, 1],
        mirror=True,
        showline=True,
        side="left",
        tickfont={"size":10},
        tickmode="auto",
        ticks='',
        title="CA",
        titlefont={"size":20},
        type="linear",
        zeroline=False
    )
    )
fig.update_layout(template='seaborn', title=title)
fig.show()
```

executed in 522ms, finished 12:10:08 2020-07-17



4.4 State wise Analysis

Let's see sales and revenue of all the stores individually across all the three states: California, Texas & Wisconsin. Plotting total three plots for each store: CA_1, CA_2, CA_3, CA_4, TX_1, TX_2, TX_3, WI_1, WI_2 & WI_3. Details about the plots are as follows:

- First plot shows the daily sales of a store, plotting the values separately for SNAP days. Also, SNAP promotes food purchase, and food sales as well to check if it really affects the food sales.
- Second plot shows the daily revenue of a store with separate plotting for SNAP days.
- Third is a heatmap to show daily sales. It's plotted in such a way that it becomes easier to see day wise values.

What is SNAP?
The United States federal government provides a nutrition assistance benefit called the Supplement Nutrition Assistance Program (SNAP). SNAP provides low income families and individuals with an Electronic Benefits Transfer debit card to purchase food products. In many states, the monetary benefits are dispersed to people across 10 days of the month and on each of these days 1/10 of the people will receive the benefit on their card. More information about the SNAP program can be found [here.](https://www.fns.usda.gov/snap/supplemental-nutrition-assistance-program)

For the heatmaps, the data is till 16th week of 2016 and datetime.weekofyear of function is returning 1,2 & 3 january of 2016 in 53rd week. Plotly's heatmap is connecting the data gap between the 16th and 53rd week. Still figuring out on how to remove this gap.

4.4 State wise Analysis

In [22]:
```
data['revenue'] = data['sales'] * data['sell_price'].astype(np.float32)
```

executed in 246ms, finished 12:10:08 2020-07-17

In [23]:
```
def introduce_nulls(df):
    idx = pd.date_range(df.date.dt.date.min(), df.date.dt.date.max())
    df = df.set_index('date')
    df = df.reindex(idx)
    df.reset_index(inplace=True)
    df.rename(columns={'index':'date'},inplace=True)
    return df

def plot_metric(df,state,store,metric):
    store_sales = df[(df['state_id']==state)&(df['store_id']==store)&(df['date']
    food_sales = store_sales[store_sales['cat_id']=='FOODS']
    store_sales = store_sales.groupby(['date','snap_'+state],as_index=False)['sa
    snap_sales = store_sales[store_sales['snap_'+state]==1]
    non_snap_sales = store_sales[store_sales['snap_'+state]==0]
    food_sales = food_sales.groupby(['date','snap_'+state],as_index=False)['sale
    snap_foods = food_sales[food_sales['snap_'+state]==1]
    non_snap_foods = food_sales[food_sales['snap_'+state]==0]
    non_snap_sales = introduce_nulls(non_snap_sales)
    snap_sales = introduce_nulls(snap_sales)
    non_snap_foods = introduce_nulls(non_snap_foods)
    snap_foods = introduce_nulls(snap_foods)
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=non_snap_sales['date'],y=non_snap_sales[metric],
                             name='Total '+metric+'(Non-SNAP)'))
    fig.add_trace(go.Scatter(x=snap_sales['date'],y=snap_sales[metric],
                             name='Total '+metric+'(SNAP)'))
    fig.add_trace(go.Scatter(x=non_snap_foods['date'],y=non_snap_foods[metric],
                             name='Food '+metric+'(Non-SNAP)'))
    fig.add_trace(go.Scatter(x=snap_foods['date'],y=snap_foods[metric],
                             name='Food '+metric+'(SNAP)'))
    fig.update_yaxes(title_text='Total items sold' if metric=='sold' else 'Total
    fig.update_layout(template='seaborn',title=store)
    fig.update_layout(
        xaxis=dict(
        #autorange=True,
        range = ['2011-01-29','2016-05-22'],
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(count=2,
                     label="2y",
```

```
                    step="year",
                    stepmode="backward"),
                dict(count=3,
                    label="3y",
                    step="year",
                    stepmode="backward"),
                dict(count=4,
                    label="4y",
                    step="year",
                    stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            autorange=True,
        ),
        type="date"
    ))
    return fig
```

executed in 25ms, finished 12:10:08 2020-07-17

In [24]:

```
cal_data = group.copy()
cal_data = cal_data[cal_data.date <= '22-05-2016']
cal_data['week'] = cal_data.date.dt.weekofyear
cal_data['day_name'] = cal_data.date.dt.day_name()
```

executed in 29ms, finished 12:10:08 2020-07-17

In [25]:

```
def calmap(cal_data, state, store, scale):
    cal_data = cal_data[(cal_data['state_id']==state)&(cal_data['store_id']==sto
    years = cal_data.year.unique().tolist()
    fig = make_subplots(rows=len(years),cols=1,shared_xaxes=True,vertical_spacin
    r=1
    for year in years:
        data = cal_data[cal_data['year']==year]
        data = introduce_nulls(data)
        fig.add_trace(go.Heatmap(
            z=data.sales,
            x=data.week,
            y=data.day_name,
            hovertext=data.date.dt.date,
            coloraxis = "coloraxis",name=year,
        ),r,1)
        fig.update_yaxes(title_text=year,tickfont=dict(size=5),row = r,col = 1)
        r+=1
    fig.update_xaxes(range=[1,53],tickfont=dict(size=10), nticks=53)
    fig.update_layout(coloraxis = {'colorscale':scale})
    fig.update_layout(template='seaborn', title=store)
    return fig
```

executed in 14ms, finished 12:10:08 2020-07-17

California

Store Navigator

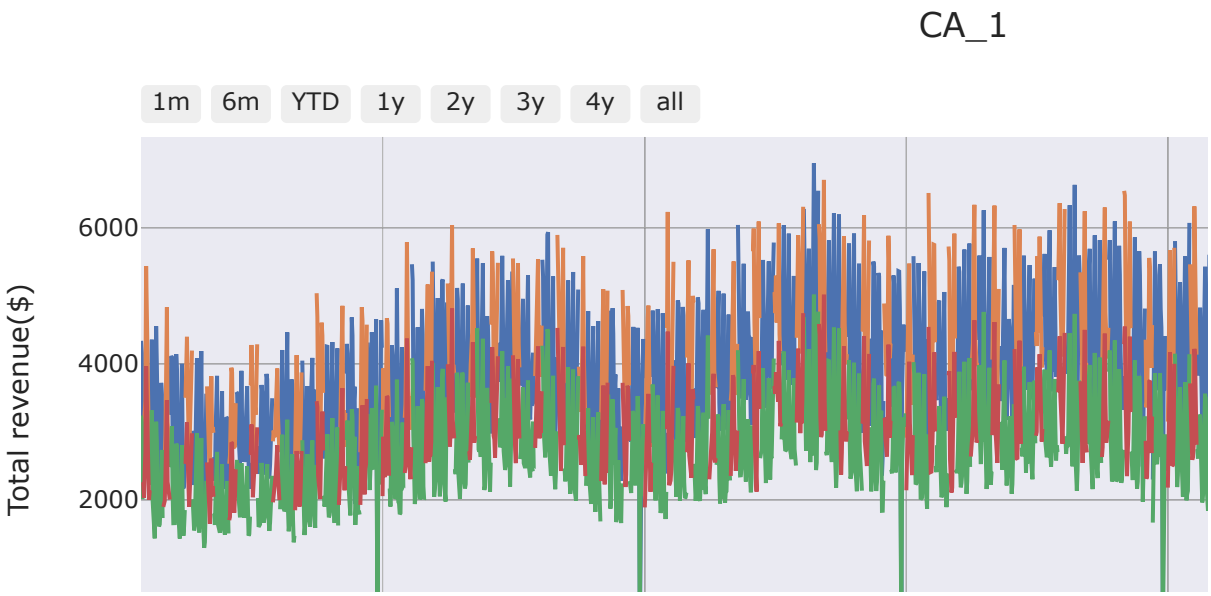
CA_1	CA_2	CA_3	CA_4

CA_1

Go to the Store Navigator

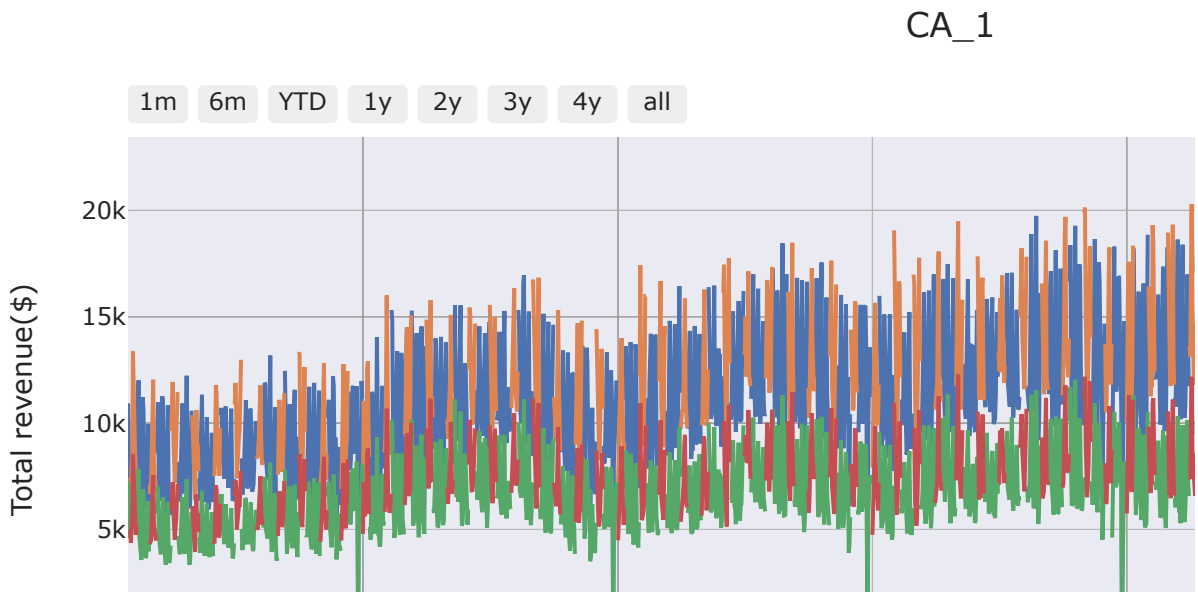
In [26]:
```
fig = plot_metric(data,'CA','CA_1','sales')
fig.show()
```

executed in 7.44s, finished 12:10:16 2020-07-17



In [27]:
```
fig = plot_metric(data,'CA','CA_1','revenue')
fig.show()
```

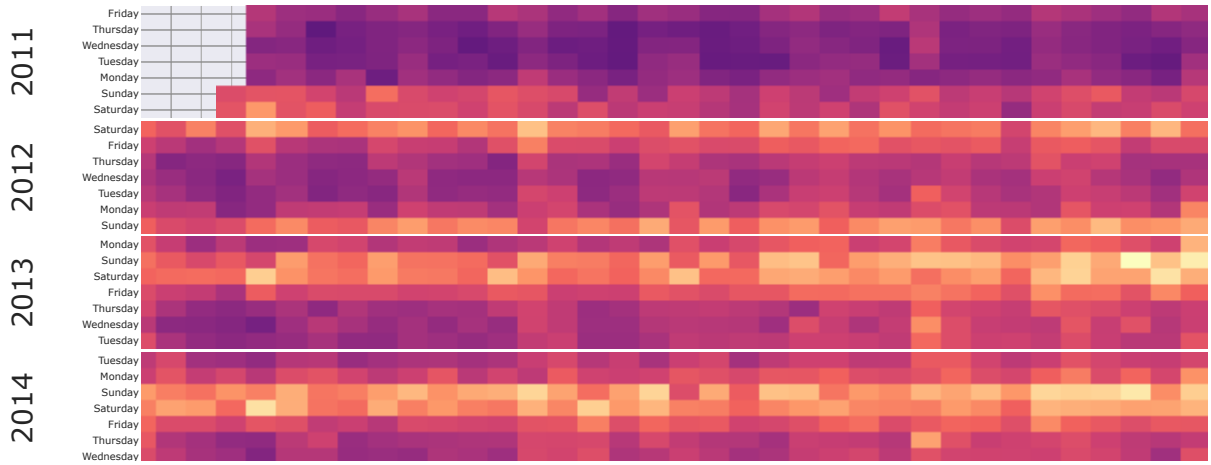
executed in 6.74s, finished 12:10:23 2020-07-17



In [28]:
```
fig = calmap(cal_data, 'CA', 'CA_1', 'magma')
fig.show()
```

executed in 138ms, finished 12:10:23 2020-07-17

CA_1

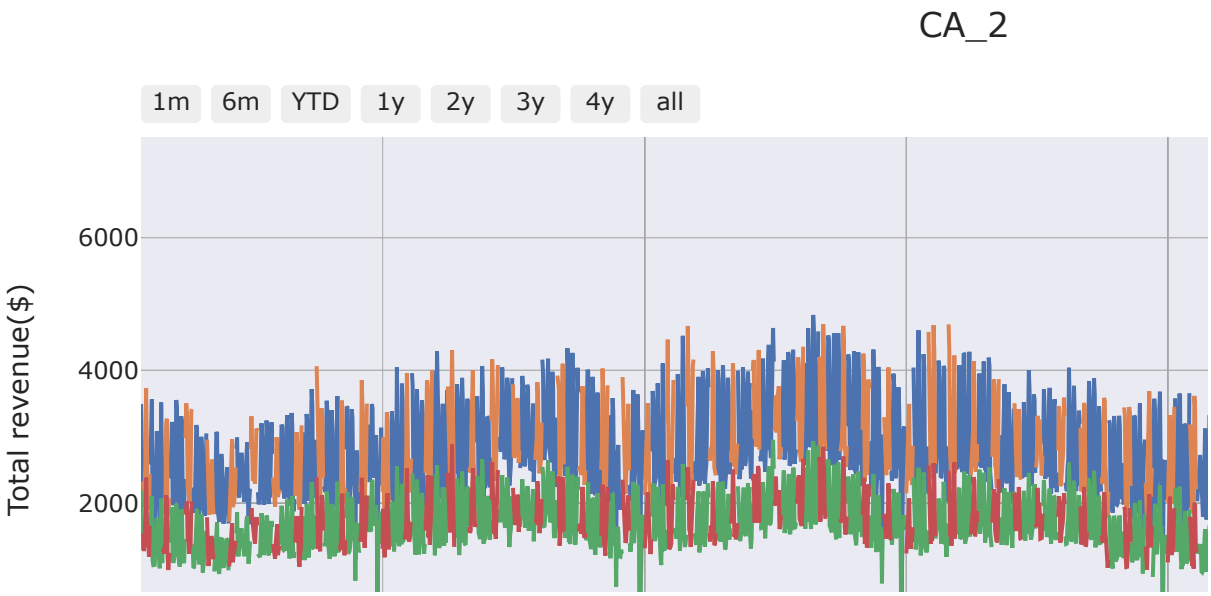


CA_2

Go to the Store Navigator

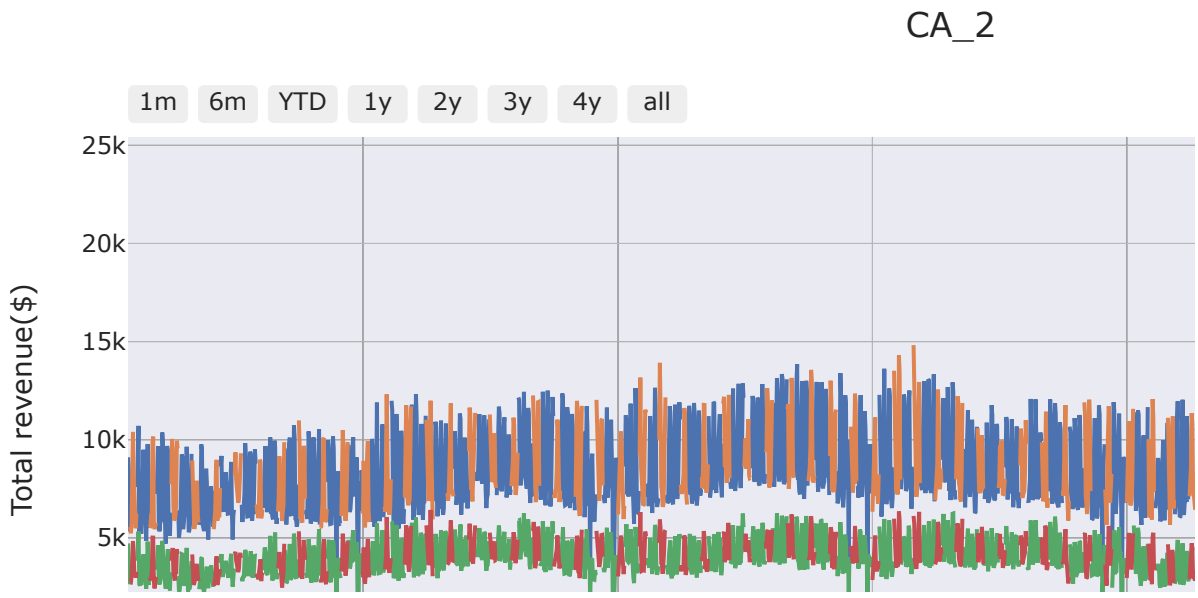
In [29]:
```
fig = plot_metric(data, 'CA', 'CA_2', 'sales')
fig.show()
```

executed in 6.85s, finished 12:10:30 2020-07-17



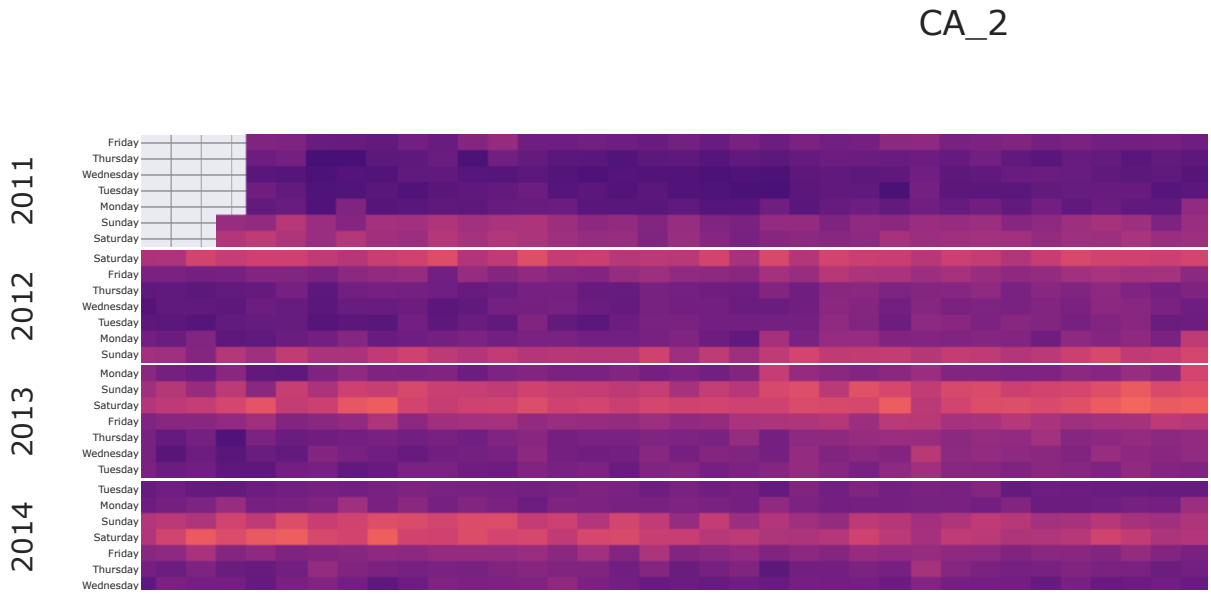
In [30]:
```
fig = plot_metric(data, 'CA', 'CA_2', 'revenue')
fig.show()
```

executed in 6.75s, finished 12:10:36 2020-07-17



In [31]:
```
fig = calmap(cal_data, 'CA', 'CA_2', 'magma')
fig.show()
```

executed in 121ms, finished 12:10:36 2020-07-17

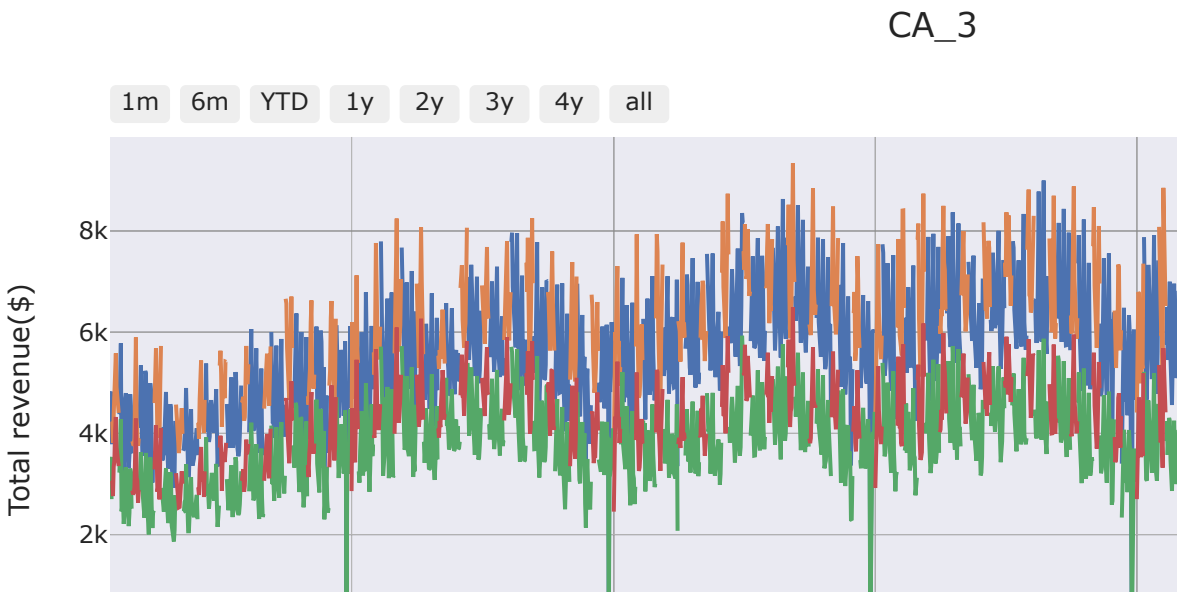


CA_3

Go to the Store Navigator

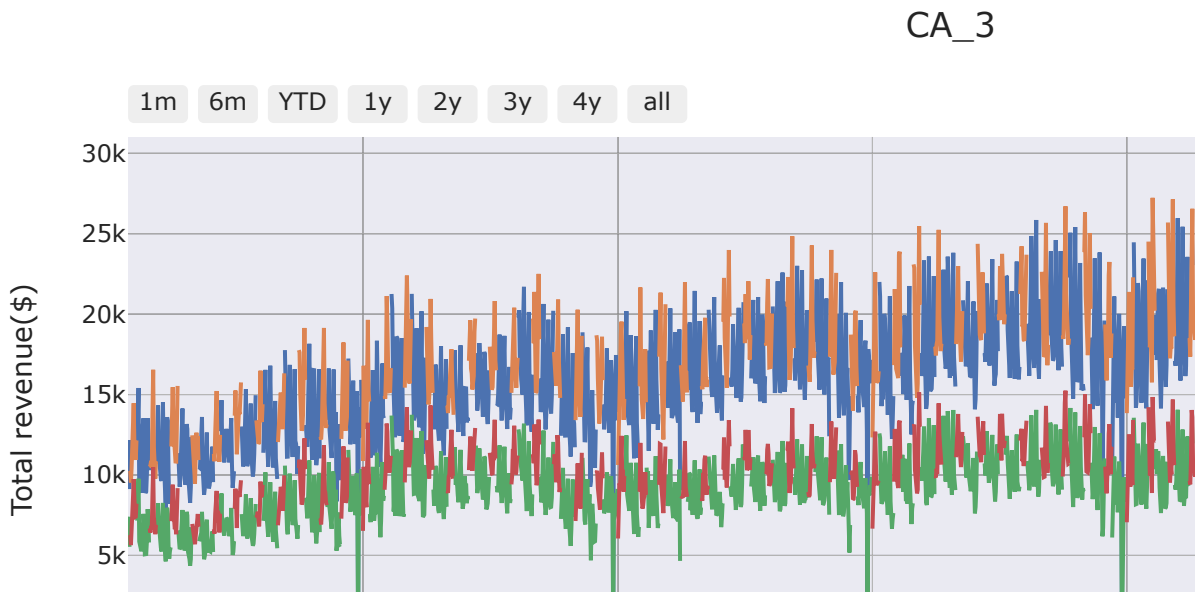
In [32]:
```
fig = plot_metric(data,'CA','CA_3','sales')
fig.show()
```

executed in 6.88s, finished 12:10:43 2020-07-17



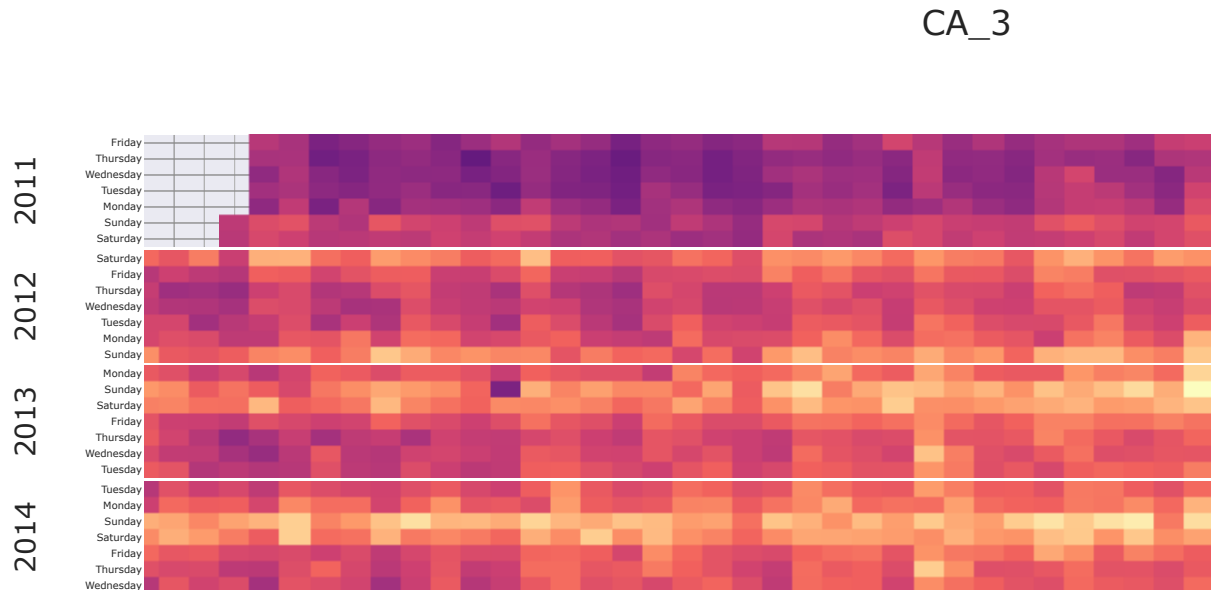
In [33]:
```
fig = plot_metric(data,'CA','CA_3','revenue')
fig.show()
```

executed in 6.76s, finished 12:10:50 2020-07-17



In [34]:
```
fig = calmap(cal_data, 'CA', 'CA_3', 'magma')
fig.show()
```

executed in 122ms, finished 12:10:50 2020-07-17

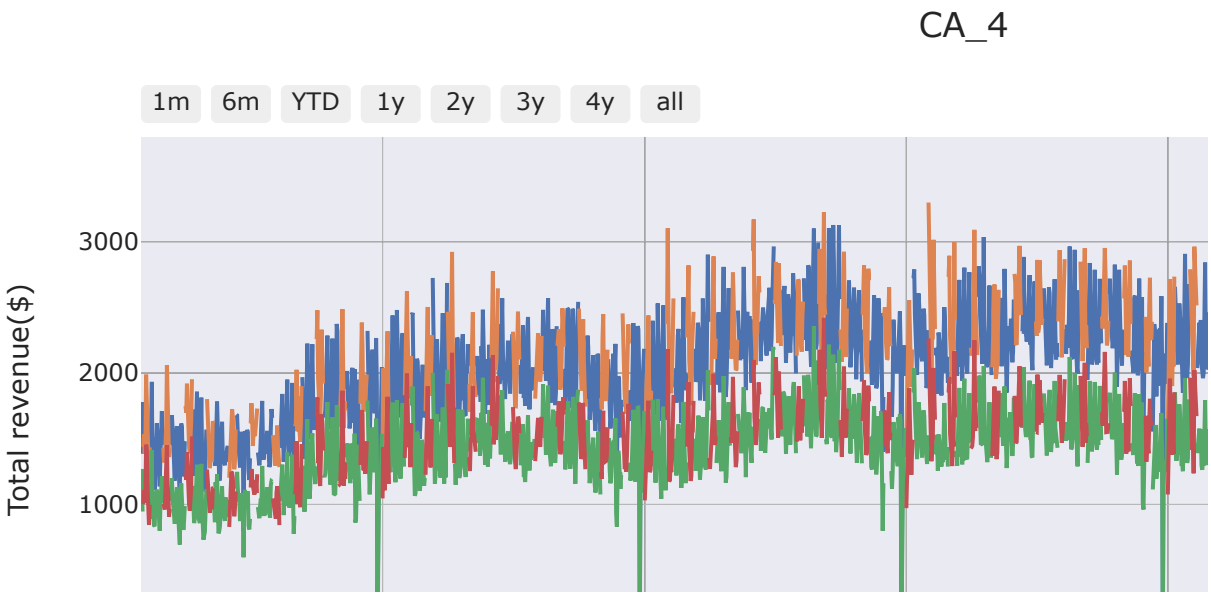


CA_4

Go to the Store Navigator

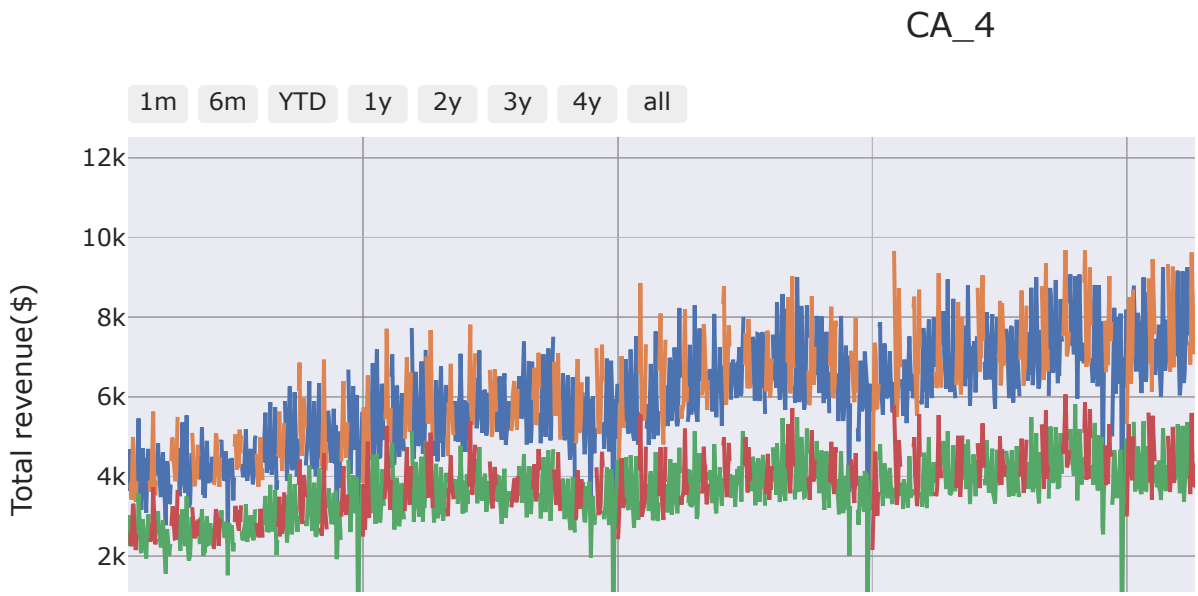
In [35]:
```
fig = plot_metric(data, 'CA', 'CA_4', 'sales')
fig.show()
```

executed in 6.79s, finished 12:10:57 2020-07-17



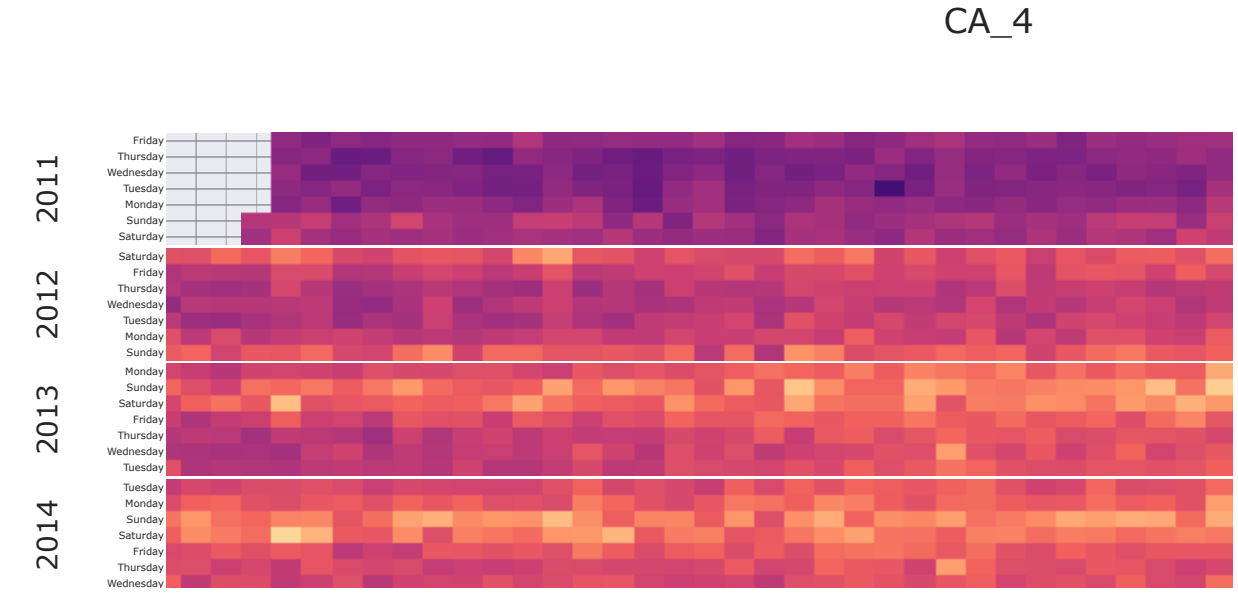
In [36]:
```
fig = plot_metric(data,'CA','CA_4','revenue')
fig.show()
```

executed in 6.84s, finished 12:11:04 2020-07-17



In [37]:
```
fig = calmap(cal_data, 'CA', 'CA_4', 'magma')
fig.show()
```

executed in 123ms, finished 12:11:04 2020-07-17



Texas

Store Navigator

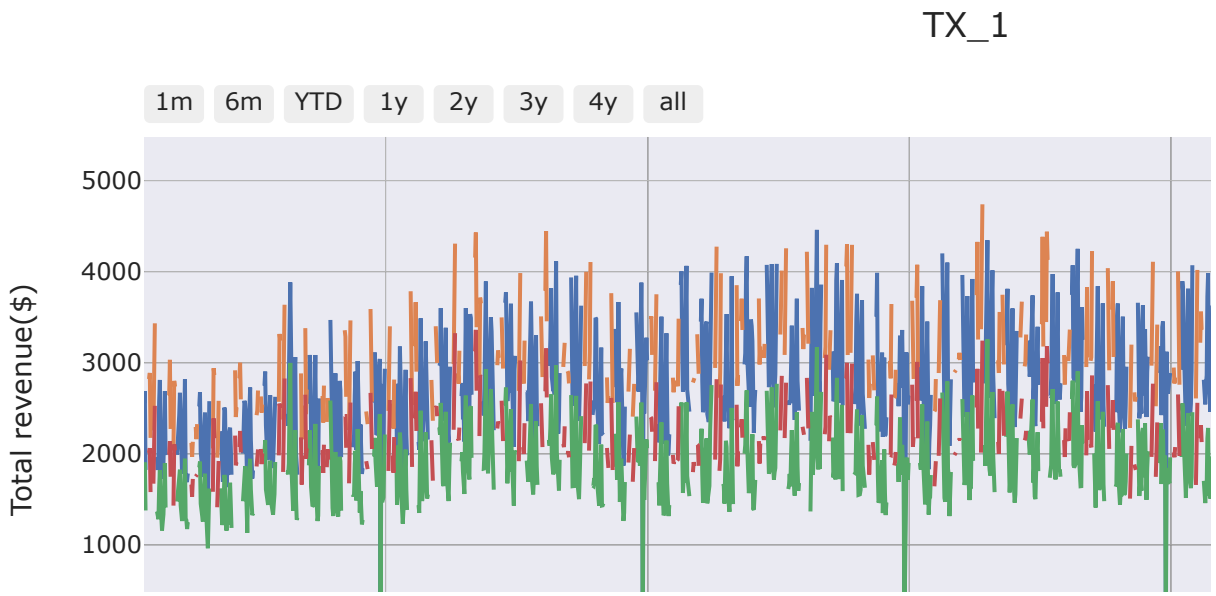
TX_1	TX_2	TX_3

TX_1

Go to the Store Navigator

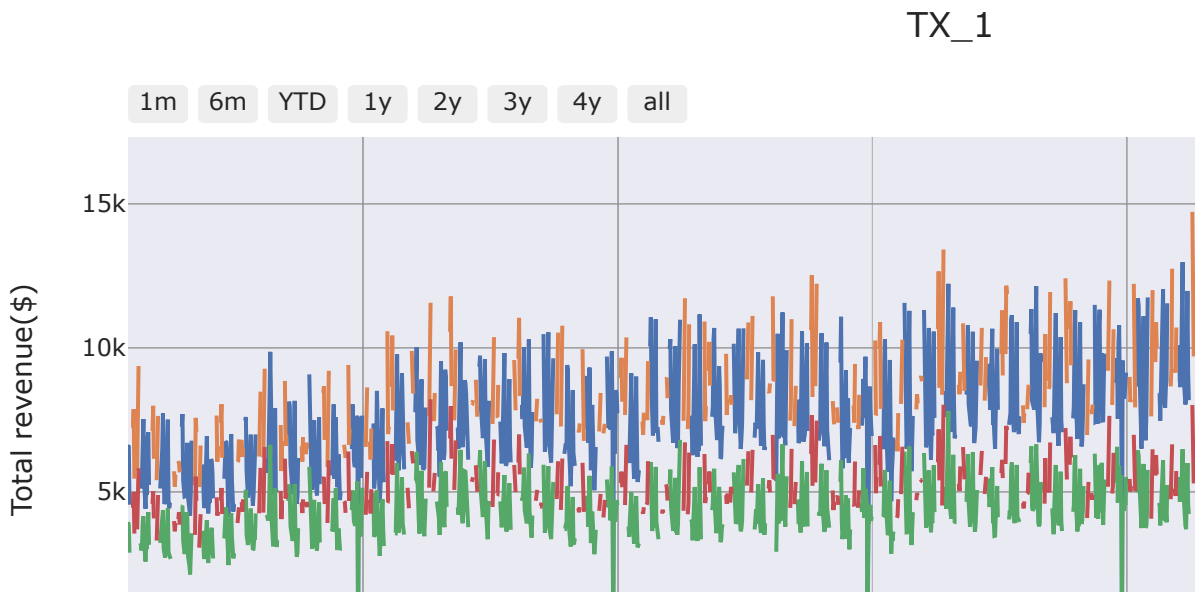
In [38]:
```
fig = plot_metric(data,'TX','TX_1','sales')
fig.show()
```

executed in 7.00s, finished 12:11:11 2020-07-17



In [39]:
```
fig = plot_metric(data,'TX','TX_1','revenue')
fig.show()
```

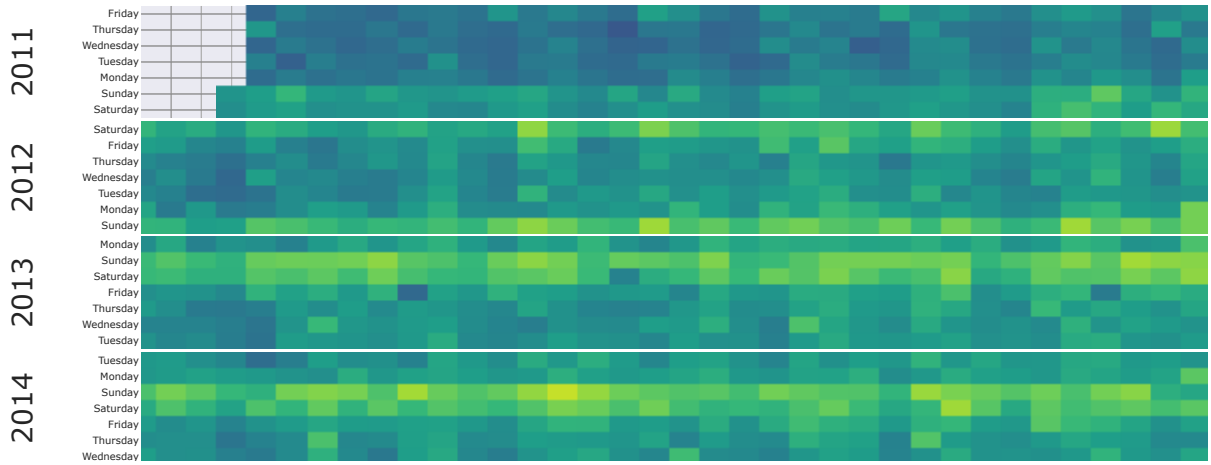
executed in 6.76s, finished 12:11:18 2020-07-17



In [40]:
```
fig = calmap(cal_data, 'TX', 'TX_1', 'viridis')
fig.show()
```

executed in 122ms, finished 12:11:18 2020-07-17

TX_1

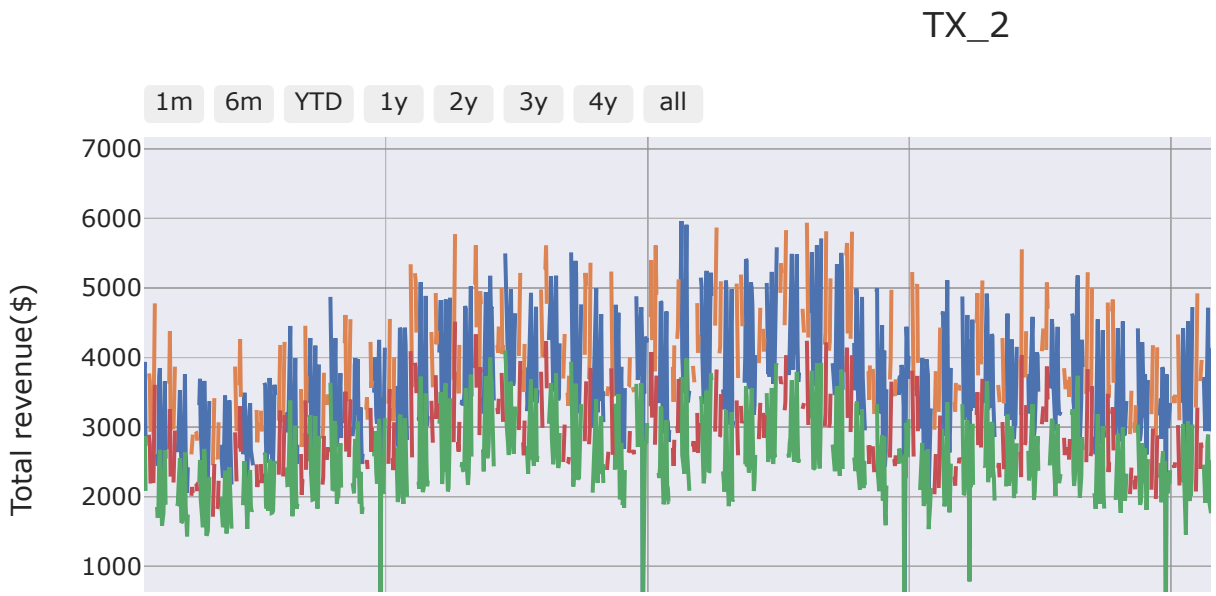


TX_2

Go to the Store Navigator

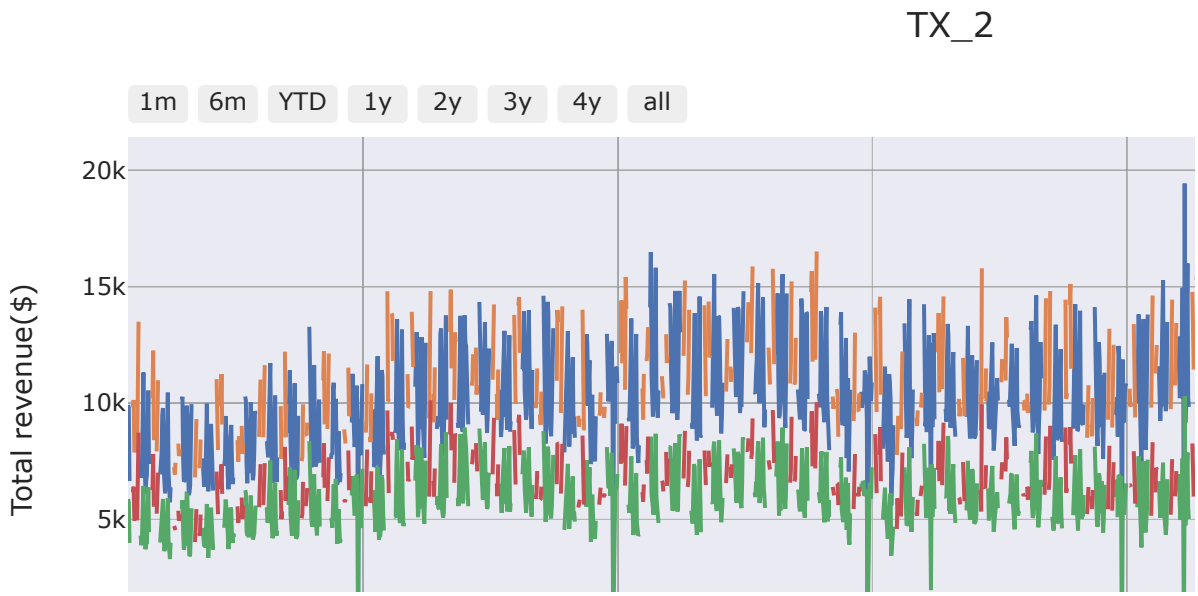
In [41]:
```
fig = plot_metric(data,'TX','TX_2','sales')
fig.show()
```

executed in 6.95s, finished 12:11:25 2020-07-17



In [42]:
```
fig = plot_metric(data,'TX','TX_2','revenue')
fig.show()
```

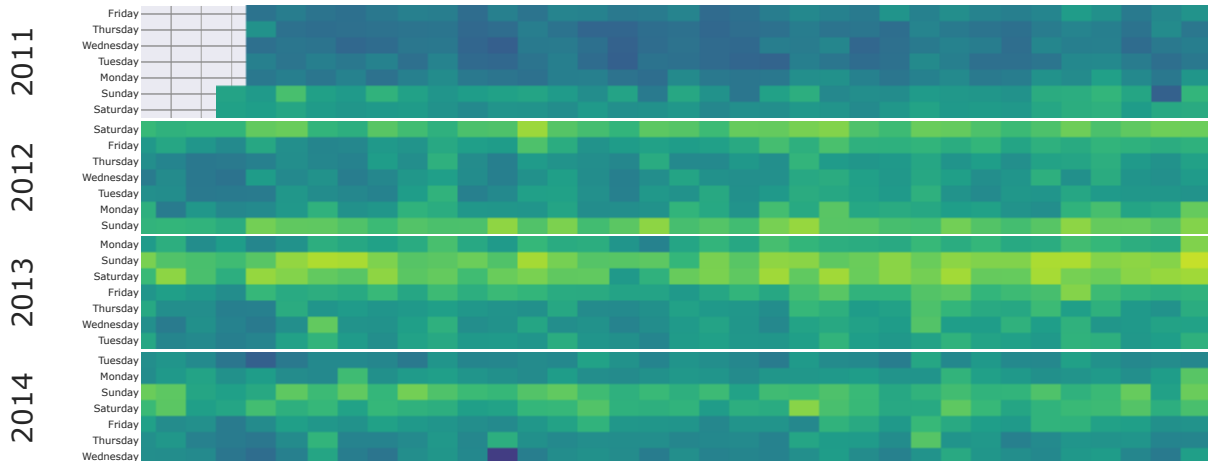
executed in 6.79s, finished 12:11:32 2020-07-17



In [43]:
```
fig = calmap(cal_data, 'TX', 'TX_2', 'viridis')
fig.show()
```

executed in 123ms, finished 12:11:32 2020-07-17

TX_2

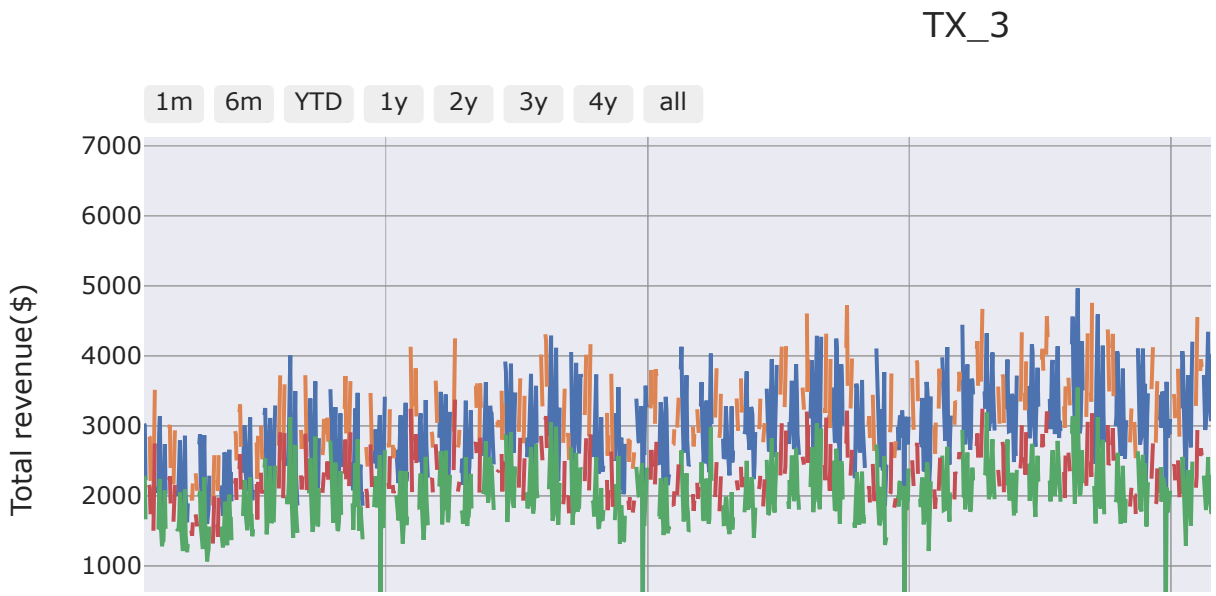


TX_3

Go to the Store Navigator

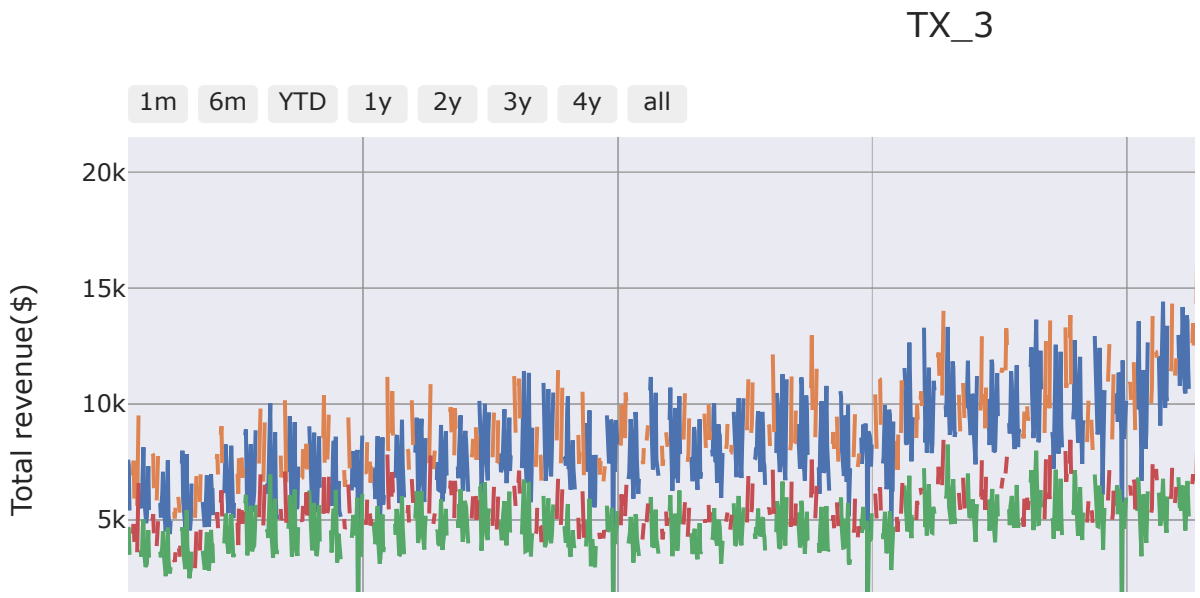
In [44]:
```
fig = plot_metric(data,'TX','TX_3','sales')
fig.show()
```

executed in 6.84s, finished 12:11:39 2020-07-17



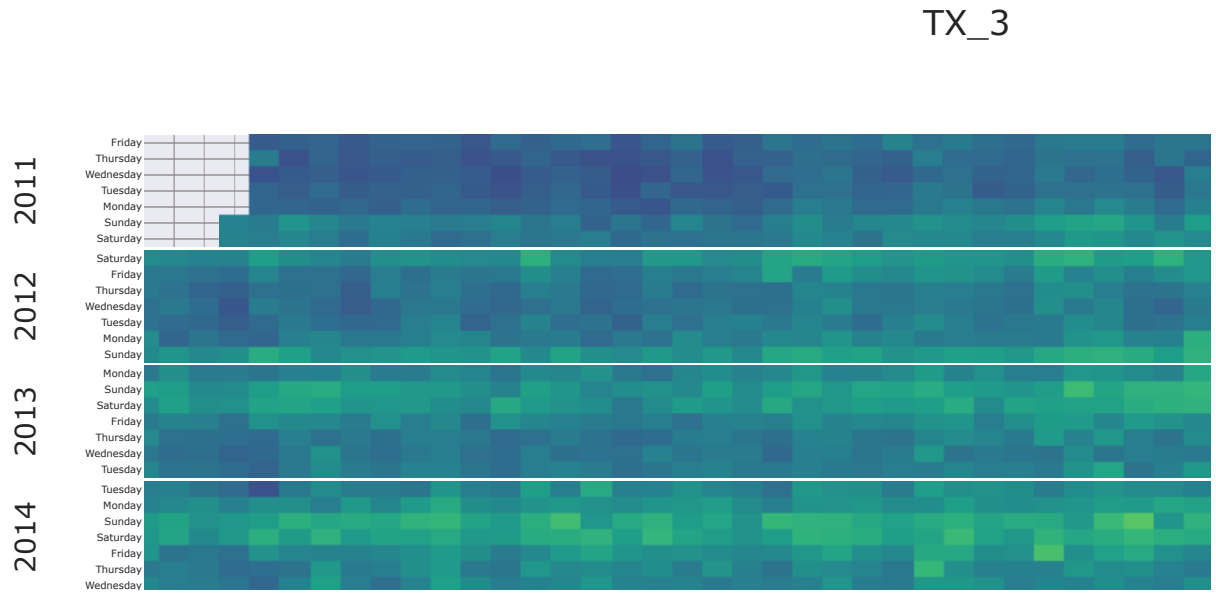
In [45]:
```
fig = plot_metric(data,'TX','TX_3','revenue')
fig.show()
```

executed in 6.79s, finished 12:11:45 2020-07-17



In [46]:
```
fig = calmap(cal_data, 'TX', 'TX_3', 'viridis')
fig.show()
```

executed in 122ms, finished 12:11:46 2020-07-17



Wisconsin

Store Navigator

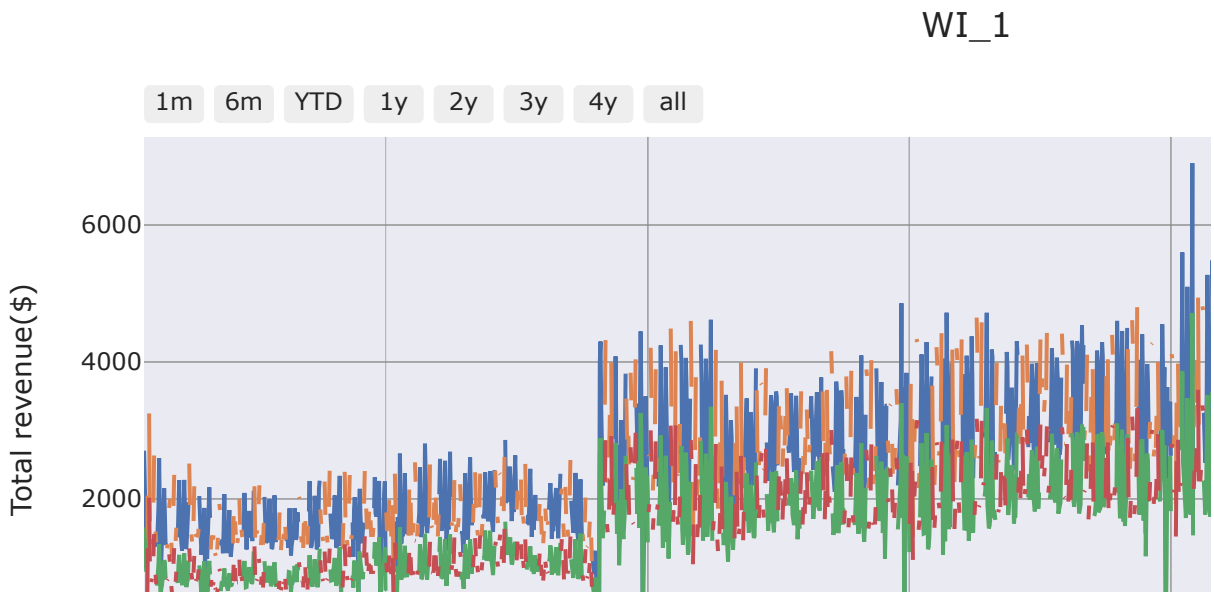
WI_1	WI_2	WI_3

WI_1

Go to the Store Navigator

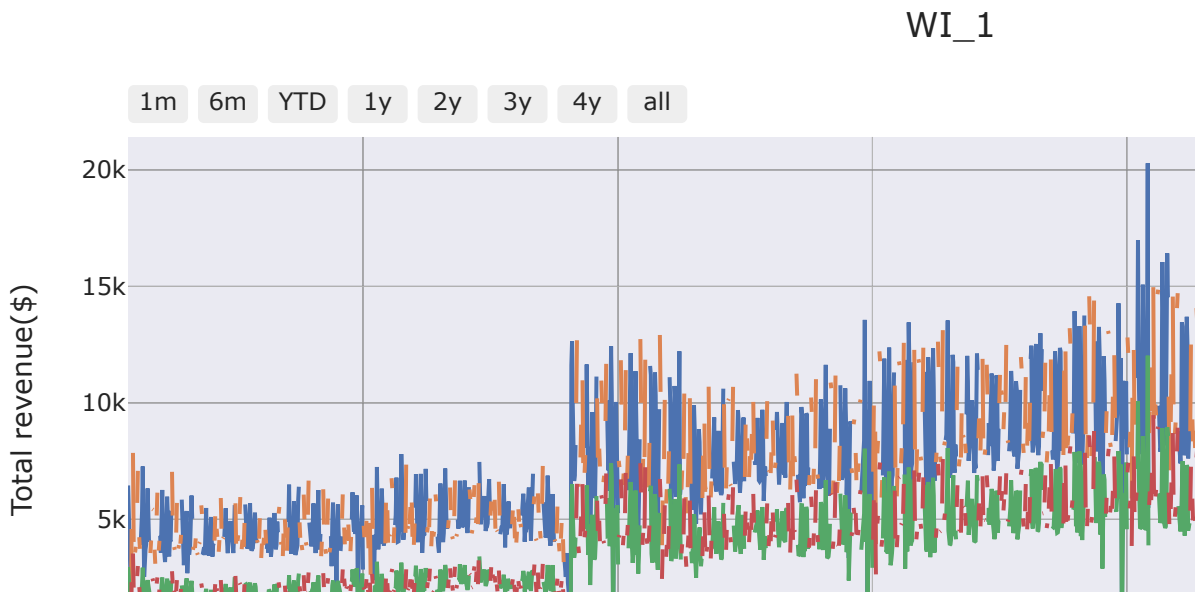
In [47]:
```
fig = plot_metric(data,'WI','WI_1','sales')
fig.show()
```

executed in 6.86s, finished 12:11:52 2020-07-17



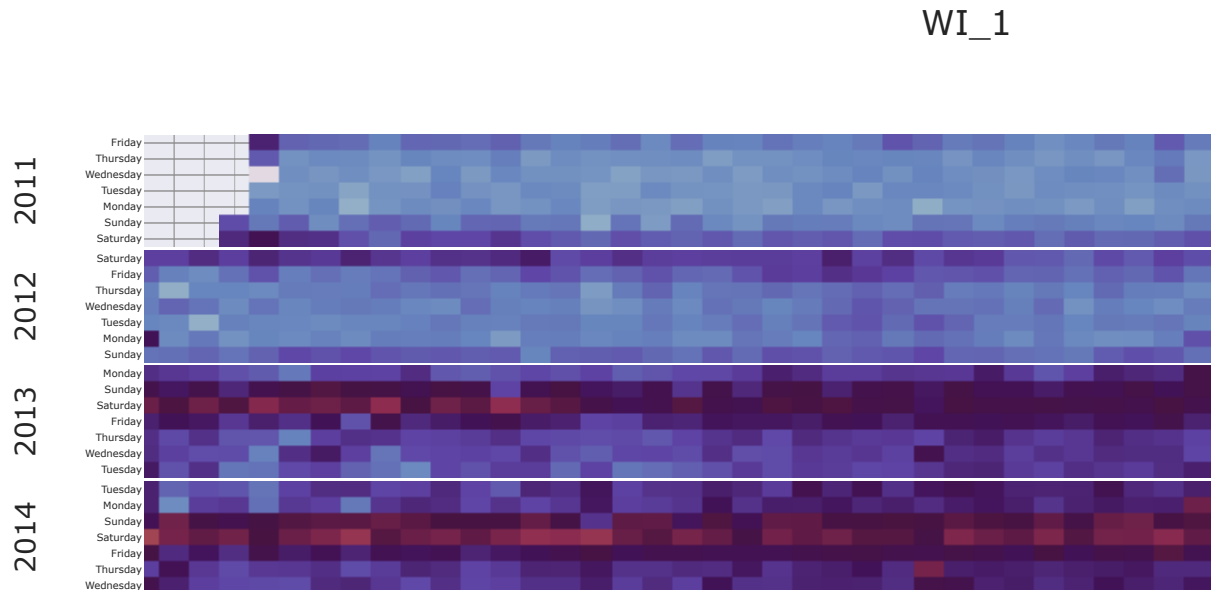
In [48]:
```
fig = plot_metric(data,'WI','WI_1','revenue')
fig.show()
```

executed in 6.82s, finished 12:11:59 2020-07-17



In [49]:
```
fig = calmap(cal_data, 'WI', 'WI_1', 'twilight')
fig.show()
```

executed in 123ms, finished 12:11:59 2020-07-17

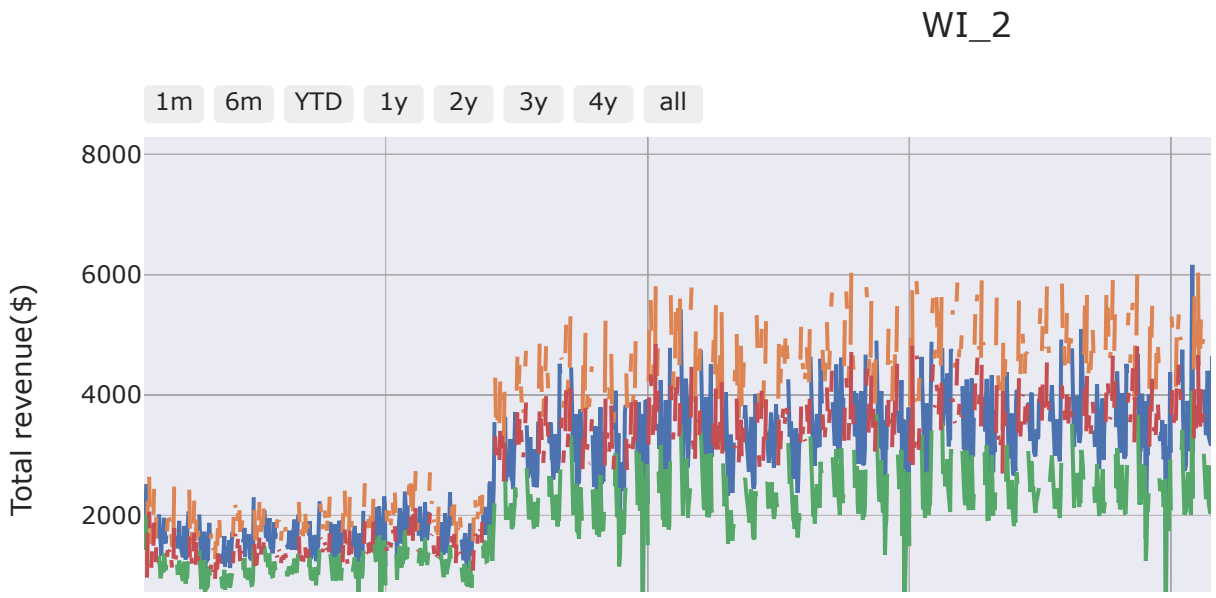


WI_2

Go to the Store Navigator

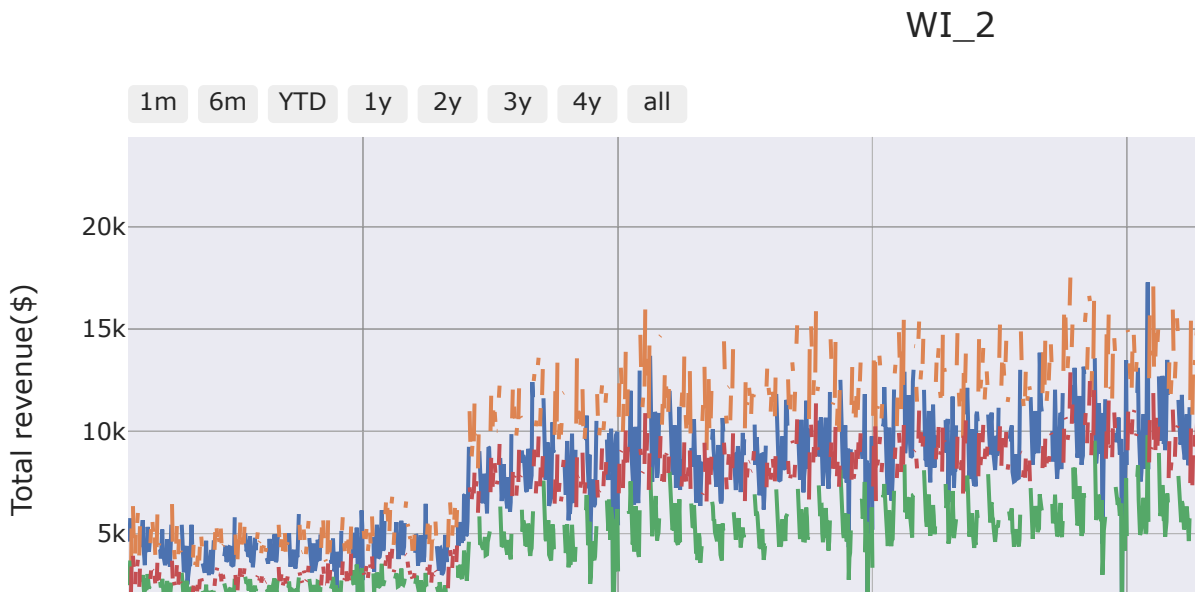
In [50]:
```
fig = plot_metric(data,'WI','WI_2','sales')
fig.show()
```

executed in 6.89s, finished 12:12:06 2020-07-17



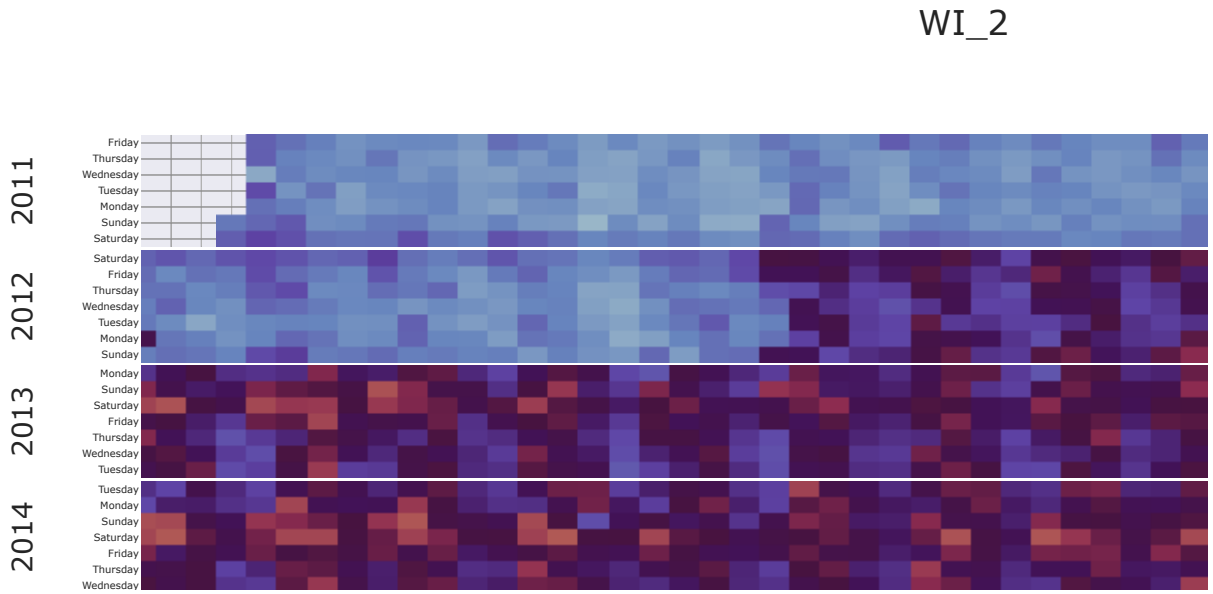
In [51]:
```
fig = plot_metric(data, 'WI','WI_2','revenue')
fig.show()
```

executed in 6.77s, finished 12:12:13 2020-07-17



In [52]:
```
fig = calmap(cal_data, 'WI', 'WI_2', 'twilight')
fig.show()
```

executed in 123ms, finished 12:12:13 2020-07-17

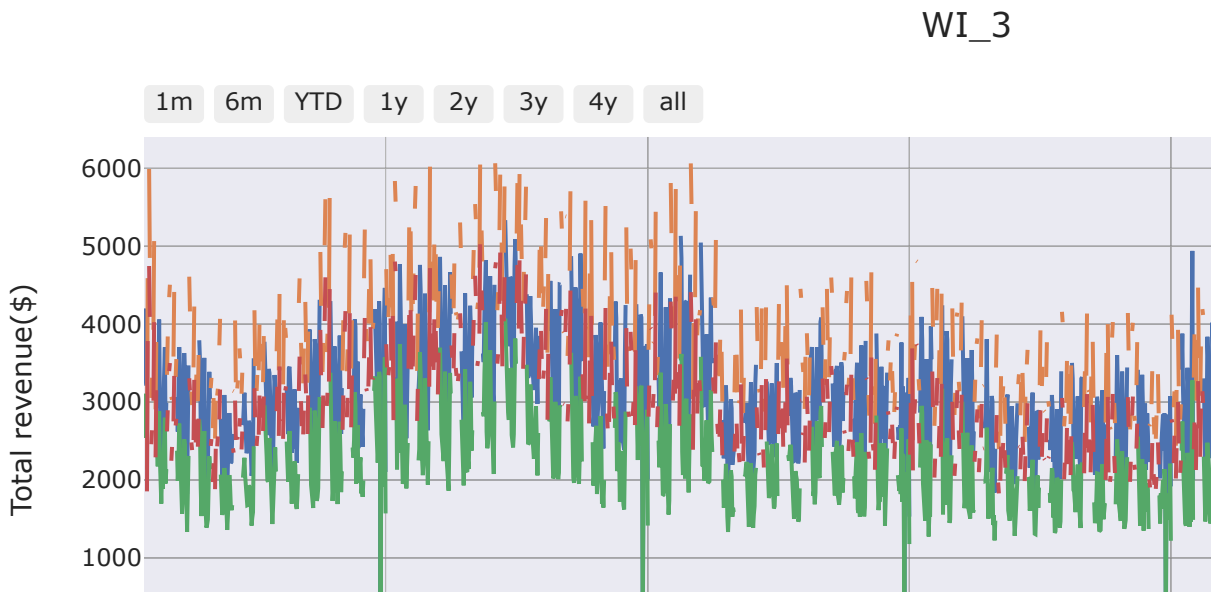


WI_3

Go to the Store Navigator

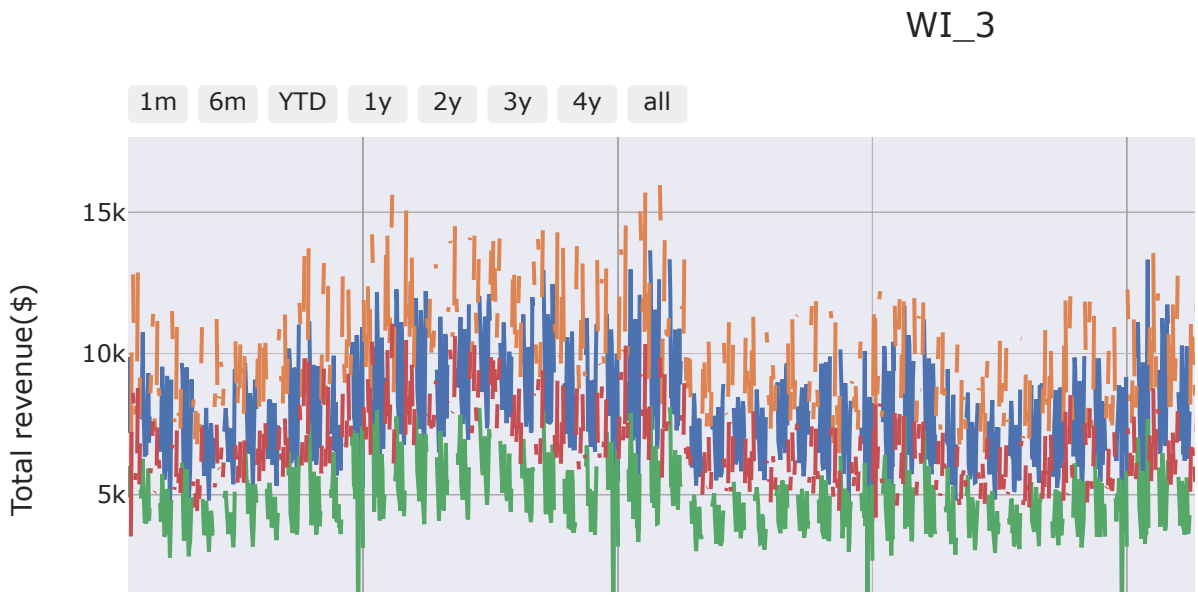
In [53]:
```
fig = plot_metric(data,'WI','WI_3','sales')
fig.show()
```

executed in 6.86s, finished 12:12:20 2020-07-17



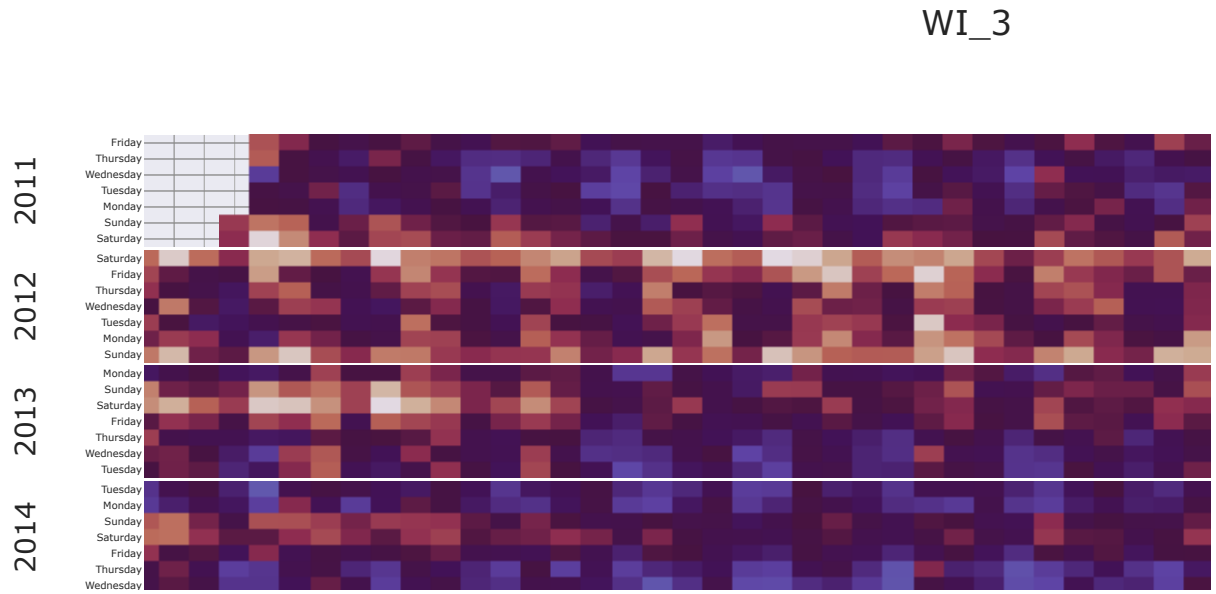
In [54]:
```
fig = plot_metric(data,'WI','WI_3','revenue')
fig.show()
```

executed in 7.04s, finished 12:12:27 2020-07-17



In [55]:
```
fig = calmap(cal_data, 'WI', 'WI_3', 'twilight')
fig.show()
```

executed in 123ms, finished 12:12:27 2020-07-17



In [56]:
```
del group, group_price_cat, group_price_store, group_state, group_state_store, c
gc.collect();
```

executed in 73ms, finished 12:12:27 2020-07-17

5. Feature Engineering

The goal of feature engineering is to provide strong and ideally simple relationships between new input features and the output feature for the supervised learning algorithm to model.

Topics

5.1 Label Encoding

1. Remove unwanted data to create space in RAM for further processing.
2. Label Encode categorical features.(I had converted already converted categorical variable to category type. So, I can simply use their codes instead of using LableEncoder)
3. Remove date as its features are already present.

In [57]:
```
nanFeatures = ['event_name_1', 'event_name_2', 'event_type_1', 'event_type_2']
for feature in nanFeatures:
    data[feature].fillna('MISSING', inplace=True)

del nanFeatures
gc.collect()
data
```

executed in 8.66s, finished 12:12:36 2020-07-17

Out[57]:

	id	item_id	dept_id	cat_id	store_id	state_
0	HOBBIES_1_001_CA_1_evaluation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	C
1	HOBBIES_1_002_CA_1_evaluation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	C
2	HOBBIES_1_003_CA_1_evaluation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	C
3	HOBBIES_1_004_CA_1_evaluation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	C
4	HOBBIES_1_005_CA_1_evaluation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	C
...	...	...	...	...	...	
60034805	FOODS_3_823_WI_3_evaluation	FOODS_3_823	FOODS_3	FOODS	WI_3	W
60034806	FOODS_3_824_WI_3_evaluation	FOODS_3_824	FOODS_3	FOODS	WI_3	W
60034807	FOODS_3_825_WI_3_evaluation	FOODS_3_825	FOODS_3	FOODS	WI_3	W
60034808	FOODS_3_826_WI_3_evaluation	FOODS_3_826	FOODS_3	FOODS	WI_3	W
60034809	FOODS_3_827_WI_3_evaluation	FOODS_3_827	FOODS_3	FOODS	WI_3	W

60034810 rows × 23 columns

In [58]:
```
categorical_features = data.columns.drop([
    'sales', 'date', 'sell_price', 'wm_yr_wk', 'snap_CA', 'snap_TX', 'snap_WI',

le = LabelEncoder()
for feature in categorical_features:
    data[feature] = le.fit_transform(data[feature])

del categorical_features
gc.collect()
data
```

executed in 1m 37.8s, finished 12:14:14 2020-07-17

Out[58]:

	id	item_id	dept_id	cat_id	store_id	state_id	d	sal
0	HOBBIES_1_001_CA_1_evaluation	1437	3	1	0	0	d_1	
1	HOBBIES_1_002_CA_1_evaluation	1438	3	1	0	0	d_1	
2	HOBBIES_1_003_CA_1_evaluation	1439	3	1	0	0	d_1	
3	HOBBIES_1_004_CA_1_evaluation	1440	3	1	0	0	d_1	
4	HOBBIES_1_005_CA_1_evaluation	1441	3	1	0	0	d_1	
...	...	...	...	...	...	...	...	
60034805	FOODS_3_823_WI_3_evaluation	1432	2	0	9	2	d_1969	
60034806	FOODS_3_824_WI_3_evaluation	1433	2	0	9	2	d_1969	
60034807	FOODS_3_825_WI_3_evaluation	1434	2	0	9	2	d_1969	
60034808	FOODS_3_826_WI_3_evaluation	1435	2	0	9	2	d_1969	
60034809	FOODS_3_827_WI_3_evaluation	1436	2	0	9	2	d_1969	

60034810 rows × 23 columns

5.2 Introduce Lags

Go back to topics

Lag features are the classical way that time series forecasting problems are transformed into supervised learning problems.

Introduce lags to the the target variable `sales` . The maximum lag I have introduced is 30 days. It's purely upto you how many lags you want to introduce.

In [59]:
```
data['lag_t28'] = data.groupby(['id'])['sales'].transform(lambda x: x.shift(28))
data['lag_t29'] = data.groupby(['id'])['sales'].transform(lambda x: x.shift(29))
data['lag_t30'] = data.groupby(['id'])['sales'].transform(lambda x: x.shift(30))
```

executed in 3m 14s, finished 12:17:28 2020-07-17

5.3 Mean and Median Encoding

Go back to topics

From a mathematical point of view, mean encoding represents a probability of your target variable, conditional on each value of the feature. In a way, it embodies the target variable in its encoded value. I have calculated mean encodings on the basis of following logical features I could think of:-

- item
- state
- store
- category
- department
- category & department
- store & item
- category & item
- department & item
- state & store
- state, store and category
- store, category and department

In [60]:
```
data['iteam_sold_avg_mean'] = data.groupby('item_id')['sales'].transform('mean')
data['state_sold_avg_mean'] = data.groupby('state_id')['sales'].transform('mean
data['store_sold_avg_mean'] = data.groupby('store_id')['sales'].transform('mean
data['cat_sold_avg_mean'] = data.groupby('cat_id')['sales'].transform('mean').as
data['dept_sold_avg_mean'] = data.groupby('dept_id')['sales'].transform('mean')
data['cat_dept_sold_avg_mean'] = data.groupby(['cat_id', 'dept_id'])['sales'].tr
data['store_item_sold_avg_mean'] = data.groupby(['store_id', 'item_id'])['sales
data['cat_item_sold_avg_mean'] = data.groupby(['cat_id', 'item_id'])['sales'].tr
data['dept_item_sold_avg_mean'] = data.groupby(['dept_id', 'item_id'])['sales']
data['state_store_sold_avg_mean'] = data.groupby(['state_id', 'store_id'])['sale
data['state_store_cat_sold_avg_mean'] = data.groupby(['state_id', 'store_id', 'c
data['store_cat_dept_sold_avg_mean'] = data.groupby(['store_id', 'cat_id', 'dept
```

executed in 48.0s, finished 12:18:16 2020-07-17

In [61]:
```
data['iteam_sold_avg_median'] = data.groupby('item_id')['sales'].transform('medi
data['state_sold_avg_median'] = data.groupby('state_id')['sales'].transform('mec
data['store_sold_avg_median'] = data.groupby('store_id')['sales'].transform('mec
data['cat_sold_avg_median'] = data.groupby('cat_id')['sales'].transform('median'
data['dept_sold_avg_median'] = data.groupby('dept_id')['sales'].transform('media
data['cat_dept_sold_avg_median'] = data.groupby(['cat_id', 'dept_id'])['sales'].
data['store_item_sold_avg_median'] = data.groupby(['store_id', 'item_id'])['sale
data['cat_item_sold_avg_median'] = data.groupby(['cat_id', 'item_id'])['sales'].
data['dept_item_sold_avg_median'] = data.groupby(['dept_id', 'item_id'])['sales'
data['state_store_sold_avg_median'] = data.groupby(['state_id', 'store_id'])['sa
data['state_store_cat_sold_avg_median'] = data.groupby(['state_id', 'store_id',
data['store_cat_dept_sold_avg_median'] = data.groupby(['store_id', 'cat_id', 'de
```

executed in 58.5s, finished 12:19:15 2020-07-17

5.4 Rolling Window Stats

Go back to topics

In [62]:
```
data['rolling_mean_t7'] = data.groupby(['id'])['sales'].transform(lambda x: x.sh
data['rolling_std_t7'] = data.groupby(['id'])['sales'].transform(lambda x: x.shi
data['rolling_std_t30'] = data.groupby(['id'])['sales'].transform(lambda x: x.sh
data['rolling_mean_t30'] = data.groupby(['id'])['sales'].transform(lambda x: x.s
data['rolling_mean_t90'] = data.groupby(['id'])['sales'].transform(lambda x: x.s
data['rolling_mean_t180'] = data.groupby(['id'])['sales'].transform(lambda x: x.
```

executed in 7m 29s, finished 12:26:44 2020-07-17

5.5 Transform Price

Go back to topics

In [63]:
```
data['lag_price'] = data.groupby(['id'])['sell_price'].transform(lambda x: x.shi
data['price_change'] = (data['lag_price'] - data['sell_price']) / (data['lag_pri
data['price_std'] = (data['sell_price'] - data['sell_price'].min()) / (1 + data[
data['price_std'] = data['price_std'].astype(np.float16)
data.drop(['lag_price'], inplace = True, axis = 1)
```

executed in 1m 47.1s, finished 12:28:31 2020-07-17

5.6 Adding daytime

Go back to topics

In [64]:
```
data.drop(['weekday', 'wday', 'month', 'year', 'wm_yr_wk'], inplace=True, axis=1
```

executed in 23.9s, finished 12:28:55 2020-07-17

In [65]:
```
data['date'] = pd.to_datetime(data['date'])
data['year'] = (data['date'].dt.year - 2010).astype(np.int16)
data['quarter'] = data['date'].dt.quarter.astype(np.int16)
data['month'] = data['date'].dt.month.astype(np.int16)
data['week'] = data['date'].dt.week.astype(np.int16)
data['day'] = data['date'].dt.day.astype(np.int16)
data['dayofweek'] = data['date'].dt.dayofweek.astype(np.int16)
data['d'] = data['d'].apply(lambda x: x.split('_')[1]).astype(np.int16)
```

executed in 30.7s, finished 12:29:26 2020-07-17

5.7 Saving the data

Go back to topics

In [66]:
```
data = reduce_mem_usage(data)
```

executed in 35.9s, finished 12:30:02 2020-07-17

```
Mem. usage decreased to 7042.20 Mb (22.6% reduction)
```

In [67]:
```
data = data[data['d'] >= 180]
data.to_pickle(path + "cleaned_data.pkl")
data
```

executed in 1m 1.08s, finished 12:31:03 2020-07-17

Out[67]:

	id	item_id	dept_id	cat_id	store_id	state_id	d	sales
5457710	HOBBIES_1_001_CA_1_evaluation	1437	3	1	0	0	180	0
5457711	HOBBIES_1_002_CA_1_evaluation	1438	3	1	0	0	180	0
5457712	HOBBIES_1_003_CA_1_evaluation	1439	3	1	0	0	180	0
5457713	HOBBIES_1_004_CA_1_evaluation	1440	3	1	0	0	180	0
5457714	HOBBIES_1_005_CA_1_evaluation	1441	3	1	0	0	180	0
...	...	...	...	...	...	...	...	...
60034805	FOODS_3_823_WI_3_evaluation	1432	2	0	9	2	1969	0
60034806	FOODS_3_824_WI_3_evaluation	1433	2	0	9	2	1969	0
60034807	FOODS_3_825_WI_3_evaluation	1434	2	0	9	2	1969	0
60034808	FOODS_3_826_WI_3_evaluation	1435	2	0	9	2	1969	0
60034809	FOODS_3_827_WI_3_evaluation	1436	2	0	9	2	1969	0

54577100 rows × 59 columns

6. Modelling and Prediction

In [68]:
```
# Original
params = {
        "metric": "rmse",
        "objective": "poisson",
        "alpha": 0.1,
        "lambda": 0.1,
        "seed": 42,
        "num_leaves": 100,
        "learning_rate": 0.075,
        "bagging_fraction": 0.75,
        "bagging_freq": 2,
        "colsample_bytree": 0.75,
    }

fit_params = {
        "num_boost_round": 2000,
        "early_stopping_rounds": 200,
        "verbose_eval": 100}
```

executed in 27ms, finished 12:31:03 2020-07-17

In [69]:
```
features = data.columns.drop(['id', 'date', 'd', 'sales','revenue'])
```

executed in 29ms, finished 12:31:03 2020-07-17

In [70]:
```
valid = data[(data['d'] >= 1914) & (data['d'] < 1942)][['id', 'd', 'sales']]
test = data[data['d'] >= 1942][['id', 'd', 'sales']]
eval_preds = test['sales']
valid_preds = valid['sales']
```

executed in 696ms, finished 12:31:04 2020-07-17

In [71]:
```
# Training individual store

stores = data['store_id'].unique().tolist()
for store in stores:
    print("STORE TRAINING: ", store+1)
    df = data[data['store_id'] == store]

    #Split the data
    X_train, y_train = df[df['d'] < 1914].drop('sales', axis=1), df[df['d'] < 19
    X_valid, y_valid = df[(df['d'] >= 1914) & (df['d'] < 1942)].drop('sales', ax
    X_test = df[df['d'] >= 1942].drop('sales', axis=1)

    train_set = lgb.Dataset(X_train[features], y_train)
    valid_set = lgb.Dataset(X_valid[features], y_valid)

    #Train and validate
    model = lgb.train(params, train_set, valid_sets=[train_set, valid_set], **fi
    valid_rmse = np.sqrt(mean_squared_error(model.predict(X_valid[features]), y_
    eval_preds[X_test[features].index] = model.predict(X_test[features])

    del model, X_train, y_train, X_valid, y_valid
    gc.collect()

    print("\n")
```

executed in 35m 32s, finished 13:06:35 2020-07-17

```
[500]   training's rmse: 1.33146        valid_1's rmse: 1.44784
[600]   training's rmse: 1.31836        valid_1's rmse: 1.44713
[700]   training's rmse: 1.30877        valid_1's rmse: 1.44696
[800]   training's rmse: 1.29803        valid_1's rmse: 1.44693
[900]   training's rmse: 1.28892        valid_1's rmse: 1.44673
[1000]  training's rmse: 1.27956        valid_1's rmse: 1.4461
[1100]  training's rmse: 1.27224        valid_1's rmse: 1.44637
Early stopping, best iteration is:
[973]   training's rmse: 1.28203        valid_1's rmse: 1.44595


STORE TRAINING:  5
Training until validation scores don't improve for 200 rounds
[100]   training's rmse: 2.00006        valid_1's rmse: 1.83158
[200]   training's rmse: 1.93425        valid_1's rmse: 1.82825
[300]   training's rmse: 1.88499        valid_1's rmse: 1.82948
[400]   training's rmse: 1.8423 valid_1's rmse: 1.82871
Early stopping, best iteration is:
[216]   training's rmse: 1.92496        valid_1's rmse: 1.82723
```

In [72]:
```
validation = sales[['id'] + ['d_' + str(i) for i in range(1914, 1942)]]
validation['id'] = pd.read_csv(path + 'sales_train_validation.csv').id
validation.columns = ['id'] + ['F' + str(i + 1) for i in range(28)]
```

executed in 2.78s, finished 13:06:38 2020-07-17

In [73]:
```
test['sales'] = eval_preds
predictions = test[['id', 'd', 'sales']]
predictions = pd.pivot(predictions, index='id', columns='d', values='sales').res
predictions.columns = ['id'] + ['F' + str(i + 1) for i in range(28)]
evaluation = sampleSubmission[['id']].merge(predictions, on = 'id')
```

executed in 543ms, finished 13:06:38 2020-07-17

In [74]:
```
# Prepare the submission
submit = pd.concat([validation, evaluation]).reset_index(drop=True)
submit.to_csv('submission.csv', index=False)
submit[30490:]
```

executed in 1.74s, finished 13:06:40 2020-07-17

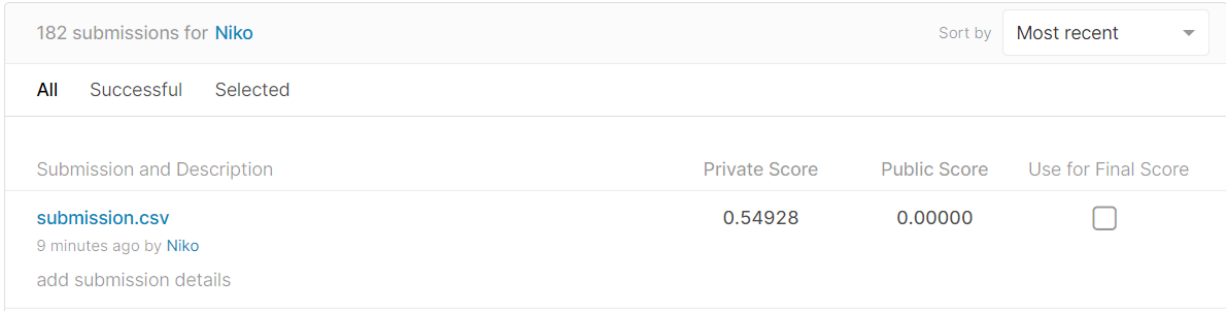
Out[74]:

	id	F1	F2	F3	F4	F5	F6
30490	HOBBIES_1_001_CA_1_evaluation	0.781391	0.688036	0.641017	0.678752	0.763680	1.030321
30491	HOBBIES_1_002_CA_1_evaluation	0.224454	0.248943	0.218462	0.213915	0.245684	0.287393
30492	HOBBIES_1_003_CA_1_evaluation	0.416459	0.391007	0.408197	0.431325	0.464422	0.633954
30493	HOBBIES_1_004_CA_1_evaluation	1.738800	1.387244	1.420946	1.506771	1.867614	2.242170
30494	HOBBIES_1_005_CA_1_evaluation	1.184605	0.983508	1.026241	1.115308	1.207863	1.364240
...	...	...	...	...	...	...	...
60975	FOODS_3_823_WI_3_evaluation	0.394008	0.370248	0.360984	0.416617	0.569052	0.559997
60976	FOODS_3_824_WI_3_evaluation	0.303700	0.364939	0.395116	0.422882	0.467844	0.468962
60977	FOODS_3_825_WI_3_evaluation	0.656078	0.615287	0.621199	0.627837	0.690633	0.934827
60978	FOODS_3_826_WI_3_evaluation	0.840524	0.857839	0.762839	0.804806	0.967293	1.110600
60979	FOODS_3_827_WI_3_evaluation	1.026137	0.957193	0.853730	0.756914	0.876564	0.881965

30490 rows × 29 columns

7. Conclusion

With one model LightGBM and those features above, we can get 0.54928 private score and top 12 on M5 forecasting accuracy competition of Kaggle leader board.



Future Work

- Focus more in feature engineering part
- Hyperparameter tunning
- Ensembling models Lasso, Ridge and LightGBM (certainly better score)
- Deploying categorical embedding
- Deploying LSTM model

Reference

- EDA reference

https://www.kaggle.com/anshuls235/time-series-forecasting-eda-fe-modelling (https://www.kaggle.com/anshuls235/time-series-forecasting-eda-fe-modelling)

In [ ]: