Business Problem

Description

Source: https://www.kaggle.com/c/favorita-grocery-sales-forecasting

Data: https://www.kaggle.com/c/favorita-grocery-sales-forecasting/data

Problem Statement: Need to build a model that more accurately forecasts product sales. In this ca
grocery store chain called 'Corporacion Favorita'. This is one case of demand forecasting, also, wit
and over stocking can be a major concern. So, with this case study we are trying to implement ma
items sold by Corporacion Favorita.

Sources Referred:

1. https://pdfs.semanticscholar.org/74b7/dd4a1bb435699c076dbcad9e826107c06c41.pdf
2. https://arxiv.org/pdf/1803.04037.pdf
3. http://web.stanford.edu/class/archive/cs/cs221/cs221.1192/2018/restricted/posters/jzhac

Business Objectives and Constraints:

1. No low-latency requirement.
2. Errors can be costly, as stock outs or over stocking are not good in terms of business.
3. Interpretability is not important.

Machine Learning Problem Formulation

Data

Data Overview

Source: https://www.kaggle.com/c/favorita-grocery-sales-forecasting/data

We are provided with 6 files(excluding test file), we need to make prediction at item-store level.

train.csv:

1. Training data, which includes the target unit_sales by date, store_nbr, and item_nbr and a uni

2. The target unit_sales can be integer (e.g., a bag of chips) or float (e.g., 1.5 kg of cheese).
3. Negative values of unit_sales represent returns of that particular item.
4. The onpromotion column tells whether that item_nbr was on promotion for a specified date a
5. Approximately 16% of the onpromotion values in this file are NaN.
6. NOTE: The training data does not include rows for items that had zero unit_sales for a store/ to whether or not the item was in stock for the store on the date. Also, there are a small num aren't seen in the test data.

stores.csv:

1. Store metadata, including city, state, type, and cluster.
2. cluster is a grouping of similar stores.

items.csv:

1. Item metadata, including family, class, and perishable.
2. NOTE: Items marked as perishable have a score weight of 1.25; otherwise, the weight is 1.0.

transactions.csv:

1. The count of sales transactions for each date, store_nbr combination. Only included for the t

oil.csv:

1. Daily oil price. Includes values during both the train and test data timeframe. (Ecuador is an health is highly vulnerable to shocks in oil prices.)

holidays_events.csv:

1. Holidays and Events, with metadata
2. NOTE: Pay special attention to the transferred column. A holiday that is transferred officially another date by the government. A transferred day is more like a normal day than a holiday. look for the corresponding row where type is Transfer. For example, the holiday Independenc 10-09 to 2012-10-12, which means it was celebrated on 2012-10-12. Days that are type Bridg (e.g., to extend the break across a long weekend). These are frequently made up by the type scheduled for work (e.g., Saturday) that is meant to payback the Bridge.
3. Additional holidays are days added a regular calendar holiday, for example, as typically happ a holiday).

Type of Machine Learning Problem

We are here trying to forecast unit sales of item per store, i.e, we are looking at a time series data at an item-stor

Performance Metric

Source: https://www.kaggle.com/c/favorita-grocery-sales-forecasting/overview/evaluation

Normalized Weighted Root Mean Squared Logarithmic Error (NWRMSLE), calculated as follows:

$$NWRMSLE = \sqrt{\frac{\sum i=1^{n} wi(\ln(\hat{y}i+1) - \ln(yi}{\sum i=1^{n} wi}}$$

where for row i, (\hat{y}i) is the predicted unitsales of an item and (yi) is the actual unitsales; n is tl

Getting Enviornment Set

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://www-us.apache.org/dist//spark/spark-2.4.5/spark-2.4.5-bin-hadoop2.7.tgz
!tar -xf spark-2.4.5-bin-hadoop2.7.tgz
!pip install -q findspark
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.4.5-bin-hadoop2.7"
import findspark
findspark.init()
findspark.find()
import pyspark
findspark.find()
```

```
'/content/spark-2.4.5-bin-hadoop2.7'
```

Major imports

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
from datetime import datetime
from pyspark.sql.functions import col, udf
from pyspark.sql.types import DateType, IntegerType, StringType, FloatType
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnin
  import pandas.util.testing as tm
```

Starting spark session

```
conf = pyspark.SparkConf().setAppName('CaseStudy1').setMaster('local')
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession(sc)
```

EXPLORATORY DATA ANALYSIS

We are provided with multiple files, let's just read them all in one go, we will see what we have in ea
by one.

```
train_df = spark.read.format("csv").option("header", "true").load('/content/drive/My Drive
test_df = spark.read.format("csv").option("header", "true").load('/content/drive/My Drive/
transactions_df = spark.read.format("csv").option("header", "true").load('/content/drive/M
stores_df = spark.read.format("csv").option("header", "true").load('/content/drive/My Driv
oil_df = spark.read.format("csv").option("header", "true").load('/content/drive/My Drive/G
items_df = spark.read.format("csv").option("header", "true").load('/content/drive/My Drive
holidays_events_df = spark.read.format("csv").option("header", "true").load('/content/driv
```

train data

If we try to print top rows this is what we get

```
train_df.show()
```

```
+---+----------+---------+--------+----------+-----------+
| id|      date|store_nbr|item_nbr|unit_sales|onpromotion|
+---+----------+---------+--------+----------+-----------+
|  0|2013-01-01|       25|  103665|       7.0|       null|
|  1|2013-01-01|       25|  105574|       1.0|       null|
|  2|2013-01-01|       25|  105575|       2.0|       null|
|  3|2013-01-01|       25|  108079|       1.0|       null|
|  4|2013-01-01|       25|  108701|       1.0|       null|
|  5|2013-01-01|       25|  108786|       3.0|       null|
|  6|2013-01-01|       25|  108797|       1.0|       null|
|  7|2013-01-01|       25|  108952|       1.0|       null|
|  8|2013-01-01|       25|  111397|      13.0|       null|
|  9|2013-01-01|       25|  114790|       3.0|       null|
| 10|2013-01-01|       25|  114800|       1.0|       null|
| 11|2013-01-01|       25|  115267|       1.0|       null|
| 12|2013-01-01|       25|  115611|       1.0|       null|
| 13|2013-01-01|       25|  115693|       1.0|       null|
| 14|2013-01-01|       25|  115720|       5.0|       null|
| 15|2013-01-01|       25|  115850|       1.0|       null|
| 16|2013-01-01|       25|  115891|       6.0|       null|
| 17|2013-01-01|       25|  115892|      10.0|       null|
| 18|2013-01-01|       25|  115894|       5.0|       null|
| 19|2013-01-01|       25|  119024|       1.0|       null|
+---+----------+---------+--------+----------+-----------+
only showing top 20 rows
```

Let us first see what info is already provided to us, the important one:

> Negative values of unit_sales represent returns of that particular item

> Approximately 16% of the onpromotion values in this file are NaN.

> The training data does not include rows for items that had zero unit_sales for a store/dat
> no information as to whether or not the item was in stock for the store on the date, and w
> way to handle that situation. Also, there are a small number of items seen in the training
> test data.

```
train_df.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- date: string (nullable = true)
 |-- store_nbr: string (nullable = true)
 |-- item_nbr: string (nullable = true)
 |-- unit_sales: string (nullable = true)
 |-- onpromotion: string (nullable = true)
```

As we can see all the dtypes for all the columns are string, let us just convert them as to what they

```
train_df = train_df.withColumn('id', col('id').cast(IntegerType())).withColumn('date', col
```

```
train_df.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- date: date (nullable = true)
 |-- store_nbr: integer (nullable = true)
 |-- item_nbr: integer (nullable = true)
 |-- unit_sales: float (nullable = true)
 |-- onpromotion: string (nullable = true)
```

```
print('Total number of rows in train: {}'.format(train_df.count()))
```

```
Total number of rows in train: 125497040
```

```
#Creating a table view so that we can use SQL queries to collect data and shows results...
train_df.createOrReplaceTempView('data')
train_df.cache
```

```
<bound method DataFrame.cache of DataFrame[id: int, date: date, store_nbr: int, item_
```

Let us see how many null values we have for onpromotion data

```
spark.sql('select onpromotion, count(1) as counts from data group by onpromotion').show()
```

```
+-----------+--------+
|onpromotion|  counts|
+-----------+--------+
|      False|96028767|
|       null|21657651|
|       True| 7810622|
+-----------+--------+
```

```
#Digging deep into it
spark.sql('select EXTRACT(YEAR FROM date) as year, onpromotion, count(1) from data group b
```

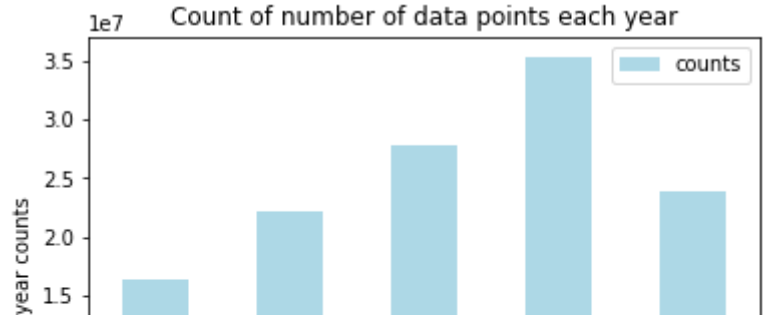
```
+----+-----------+--------+
|year|onpromotion|count(1)|
+----+-----------+--------+
|2013|       null|16322662|
|2014|       True|  459114|
|2014|      False|16477499|
|2014|       null| 5334989|
|2015|      False|26777369|
|2015|       True| 1087275|
|2016|      False|31715287|
|2016|       True| 3514584|
|2017|       True| 2749649|
|2017|      False|21058612|
+----+-----------+--------+
```

We can see that the null values are for year 2013 and 2014, from 2015 we don't see any null values were not tracked for these parts with missing values for on promotion. Atleast, we know about the is good....

Let us start by plotting frequencies of data point on yearly/monthly/daily basis

```
year_data = spark.sql('select EXTRACT(YEAR FROM date) as year, count(1) as counts from dat
```

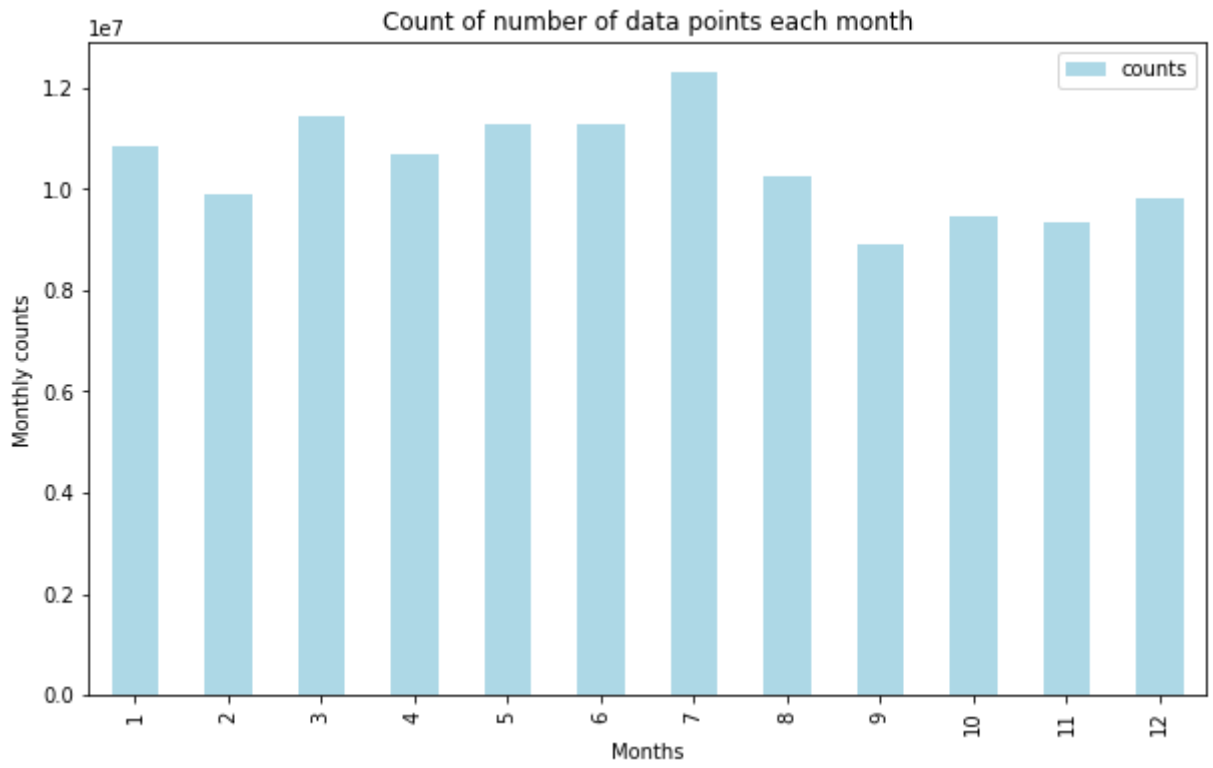
```
x_labels = year_data['year'].values
fig = year_data[['counts']].plot(kind='bar', facecolor='lightblue')
fig.set_xticklabels(x_labels)
fig.set_title('Count of number of data points each year')
fig.set_xlabel('year')
fig.set_ylabel('year counts')
plt.show()
```



This shows an increasing trend in terms of dates, which indicates rising sales and/or rise in the un only till 2017, but in general an upward trend

```
monthly_data = spark.sql('select EXTRACT(MONTH FROM date) as year, count(1) as counts from
```

```
x_labels = monthly_data['year'].values
fig = monthly_data[['counts']].plot(kind='bar', facecolor='lightblue', figsize=(10, 6))
fig.set_xticklabels(x_labels)
fig.set_title('Count of number of data points each month')
fig.set_xlabel('Months')
fig.set_ylabel('Monthly counts')
plt.show()
```

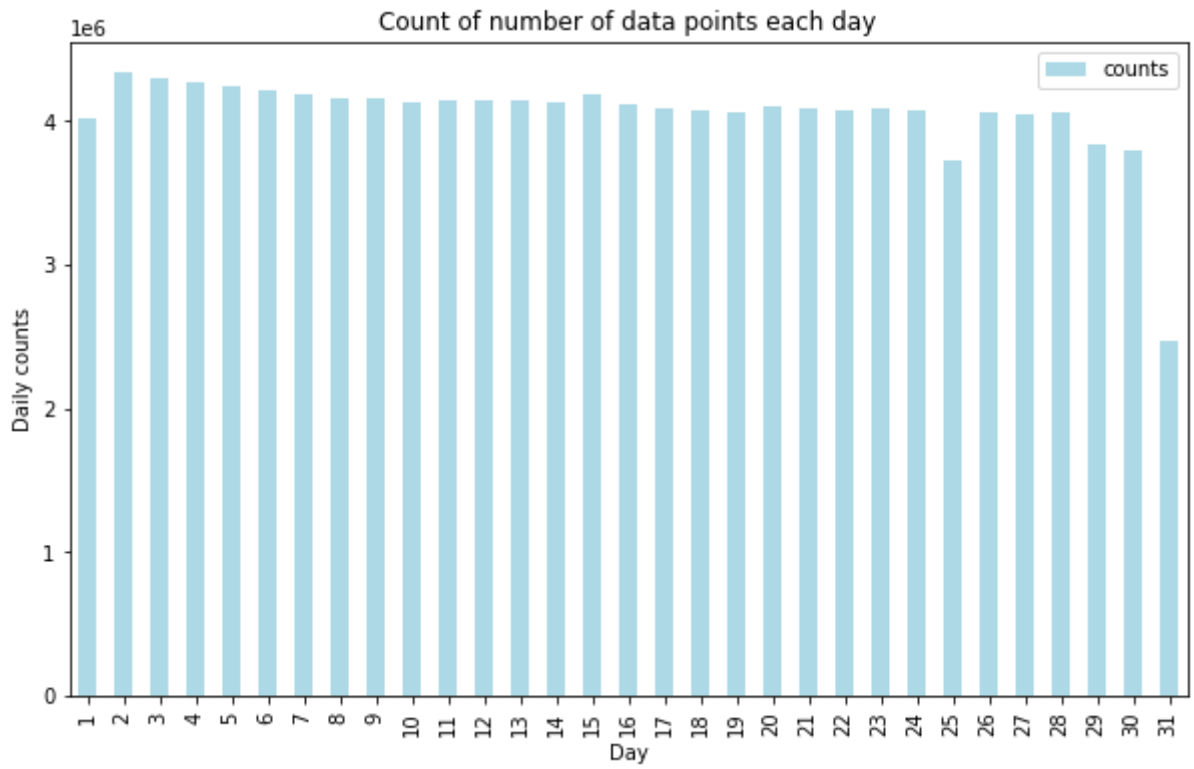


There is not much to tell from this plot, uniformity is what it shows....

```
daily_data = spark.sql('select EXTRACT(DAY FROM date) as year, count(1) as counts from dat
```

```
x_labels = daily_data['year'].values
fig = daily_data[['counts']].plot(kind='bar', facecolor='lightblue', figsize=(10, 6))
```

```
fig.set_xticklabels(x_labels)
fig.set_title('Count of number of data points each day')
fig.set_xlabel('Day')
fig.set_ylabel('Daily counts')
plt.show()
```



Uniformly distributed among days, for 31st we see less observation, but then not all month have 3

```
#Counting the total number of items
spark.sql('select COUNT(DISTINCT item_nbr) FROM data').show()
```

```
+------------------------+
|count(DISTINCT item_nbr)|
+------------------------+
|                    4036|
+------------------------+
```

We have a universe of 4036 items

```
#Total stores
spark.sql('select COUNT(DISTINCT store_nbr) FROM data').show()
```

```
+-------------------------+
|count(DISTINCT store_nbr)|
+-------------------------+
|                       54|
+-------------------------+
```

Total number of stores selling these items are : 54

```
spark.sql('select EXTRACT(YEAR FROM date) as year, COUNT(DISTINCT item_nbr) as items FROM
```

```
+----+-----+
|year|items|
+----+-----+
|2013| 1977|
|2014| 2885|
|2015| 3445|
|2016| 3886|
|2017| 4018|
+----+-----+
```

Total number of items in universe is: 4036, we can see that number of items they are selling increa
total of 4018 items and for 2016 we have 3886, there are new items launched in 2017 which may h
already stated that we have unseen items, we need to deal with such data points during training or
items that may be discontinued, we need to see if we can somehow figure out those items and rer
required.....

```
spark.sql('select EXTRACT(YEAR FROM date) as year, COUNT(DISTINCT store_nbr) as items FROM
```

```
+----+-----+
|year|items|
+----+-----+
|2013|   47|
|2014|   48|
|2015|   53|
|2016|   53|
|2017|   54|
+----+-----+
```

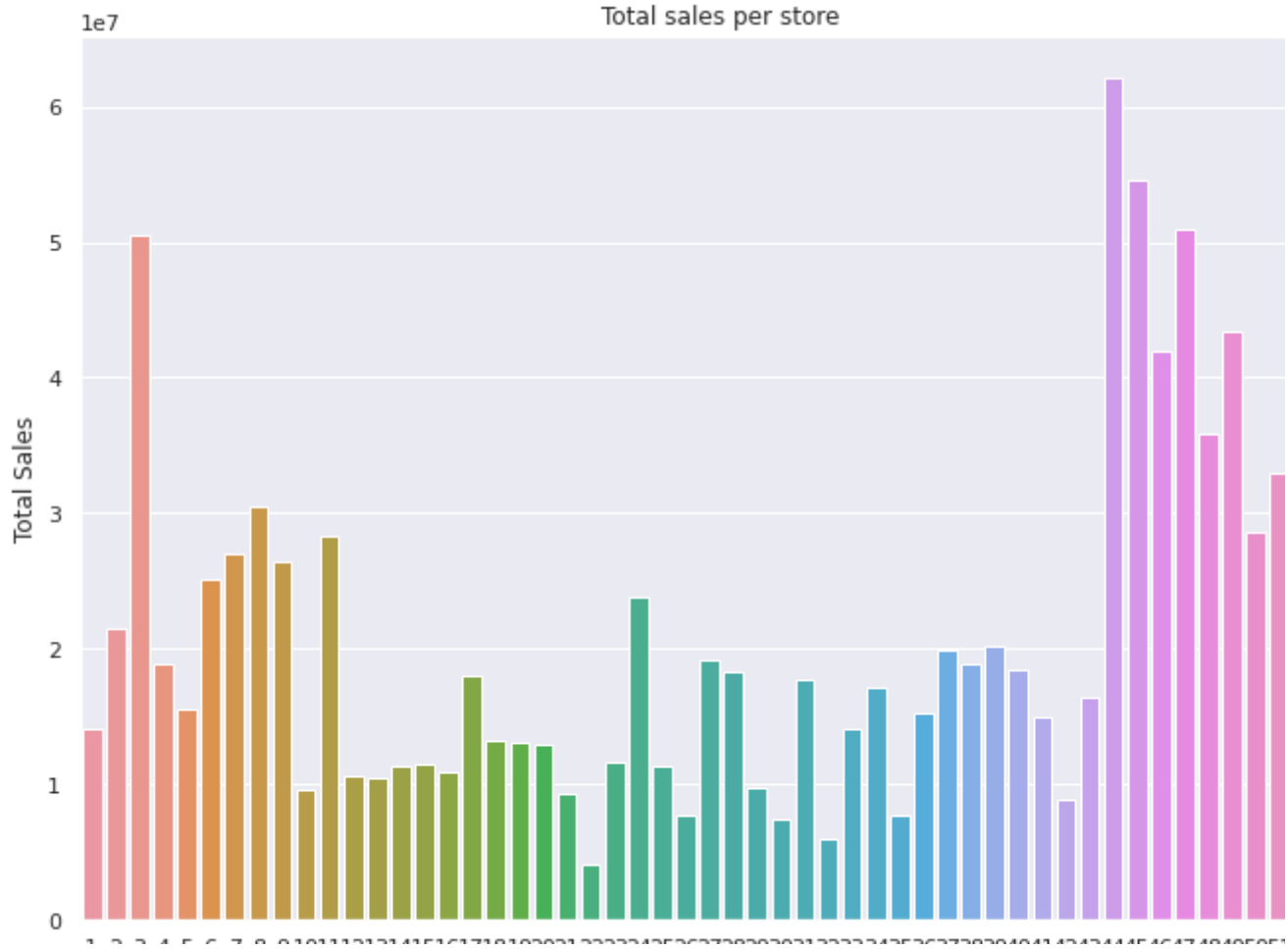
From above we can say that 47 stores were there in 2013, 1 new store opened in 2014, 5 new oper

Total Sales By Store

```
store_unit_sales = spark.sql('select store_nbr, SUM(unit_sales) as total_sales FROM data W
```

```
import seaborn as sns
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.barplot(store_unit_sales['store_nbr'].values, store_unit_sales['total_sales'].values)
plt.title('Total sales per store')
plt.xlabel('Stores')
plt.ylabel('Total Sales')
```

```
Text(0, 0.5, 'Total Sales')
```



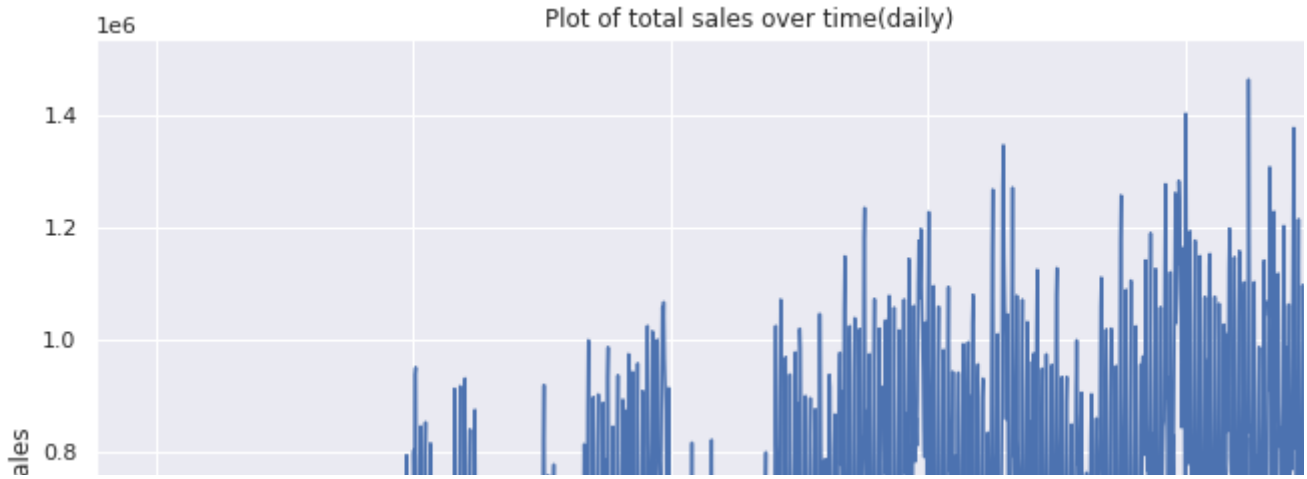
We can say Store number 3,44,45,46,47,48,49 corresponds to high volume of sales

Let us try to plot unit sales and see what kind of signal we get

```
sales_date_agg = spark.sql('select date, SUM(unit_sales) as total_sales FROM data WHERE un
```

```
sns.lineplot(x=sales_date_agg['date'], y=sales_date_agg['total_sales'])
plt.title('Plot of total sales over time(daily)')
```

```
Text(0.5, 1.0, 'Plot of total sales over time(daily)')
```



We can see there is an upward trend, year by year, obviously as the number of items will increase t into it and plot for one particular year and see how the plot looks like on a daily basis, let us consic

```
import datetime
start_index = sales_date_agg[sales_date_agg['date'] == datetime.date(2015, 1, 1)].index[0]
end_index = sales_date_agg[sales_date_agg['date'] == datetime.date(2015, 12, 31)].index[0]

year_2015_df = sales_date_agg.loc[start_index : end_index]
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.lineplot(x=year_2015_df['date'], y=year_2015_df['total_sales'])
plt.title('Plot of total sales over time(daily) for year 2015')
```

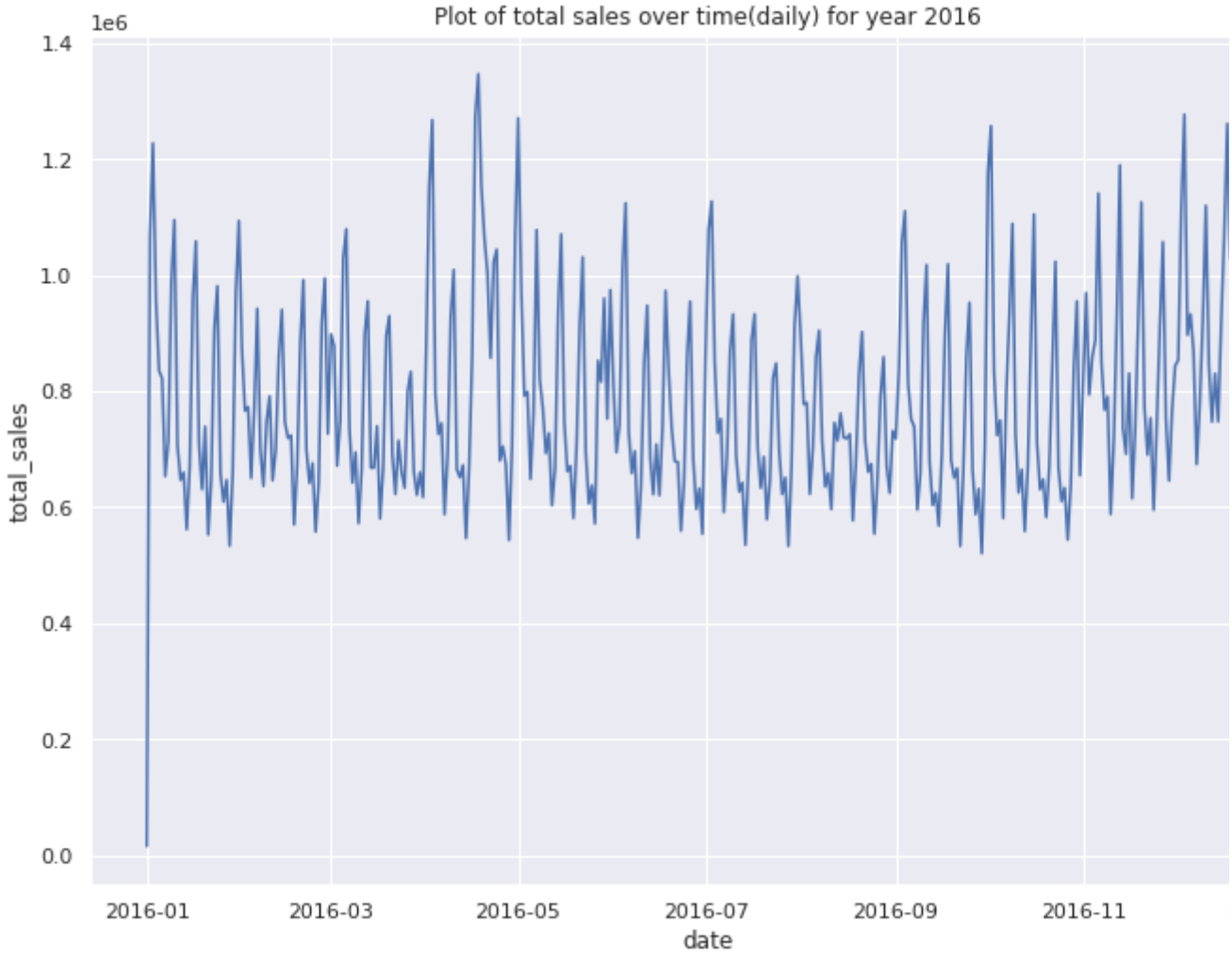
```
Text(0.5, 1.0, 'Plot of total sales over time(daily) for year 2015')
```

A little upward trend with sales for 2015

```
start_index = sales_date_agg[sales_date_agg['date'] == datetime.date(2016, 1, 1)].index[0]
end_index = sales_date_agg[sales_date_agg['date'] == datetime.date(2016, 12, 31)].index[0]

year_2016_df = sales_date_agg.loc[start_index : end_index]
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.lineplot(x=year_2016_df['date'], y=year_2016_df['total_sales'])
plt.title('Plot of total sales over time(daily) for year 2016')
```

```
Text(0.5, 1.0, 'Plot of total sales over time(daily) for year 2016')
```



2016 seems to be a stable year in terms of sales, but we need to check more, as selecting right am up with best results....

From the plots seen for 2015 and 2016, we can say that the data in general is uniform in terms of s general, this may vary for item as different items will show different signals, as some items may ha

What if we plot total sales on weekly level and monthly level....

```
sales_date_agg['week'] = sales_date_agg['date'].apply(lambda x : x.isocalendar()[1])
```

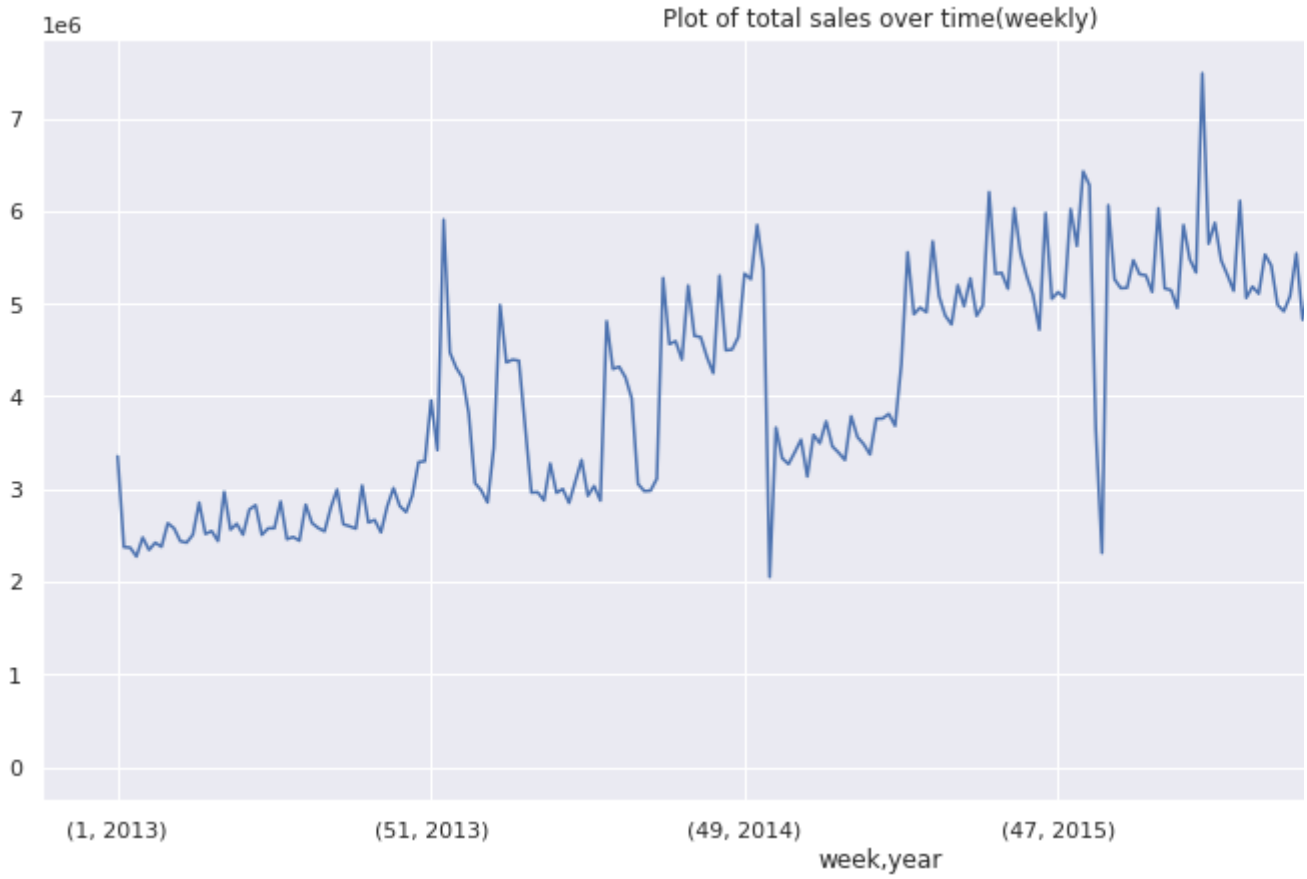
```
sales_date_agg['month'] = sales_date_agg['date'].apply(lambda x : x.month)
sales_date_agg['year'] = sales_date_agg['date'].apply(lambda x : x.year)
```

```
sales_date_agg.head()
```

	date	total_sales	week	month	year
0	2013-01-01	2511.619000	1	1	2013
1	2013-01-02	496095.418000	1	1	2013
2	2013-01-03	361487.311047	1	1	2013
3	2013-01-04	354472.193056	1	1	2013
4	2013-01-05	477357.121073	1	1	2013

```
fig, ax = plt.subplots(figsize=(15,7))
sales_date_agg.groupby(['week','year'], sort = False).sum()['total_sales'].plot(ax=ax)
plt.title('Plot of total sales over time(weekly)')
```

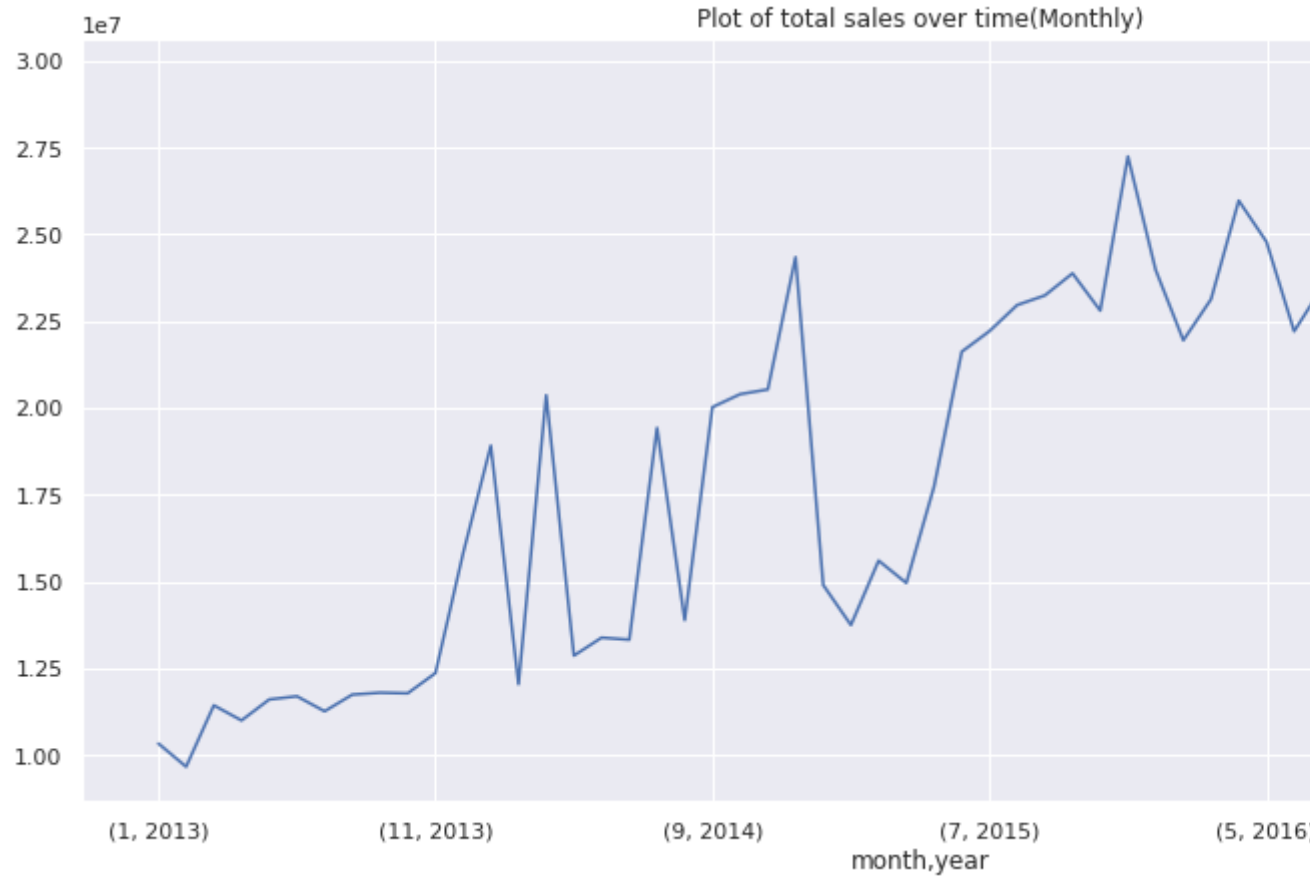
```
Text(0.5, 1.0, 'Plot of total sales over time(weekly)')
```



This doesn't add much value to our analysis....

```
fig, ax = plt.subplots(figsize=(15,7))
sales_date_agg.groupby(['month','year'], sort = False).sum()['total_sales'].plot(ax=ax)
plt.title('Plot of total sales over time(Monthly)')
```

```
Text(0.5, 1.0, 'Plot of total sales over time(Monthly)')
```



This doesn't add much value to our analysis....

```
def plot_line_plot(x, y, txt):
  sns.set(rc={'figure.figsize':(11.7,8.27)})
  sns.lineplot(x =x, y = y)
  plt.title(txt)
  plt.show()
```

Let us see Sales of a particular store over time

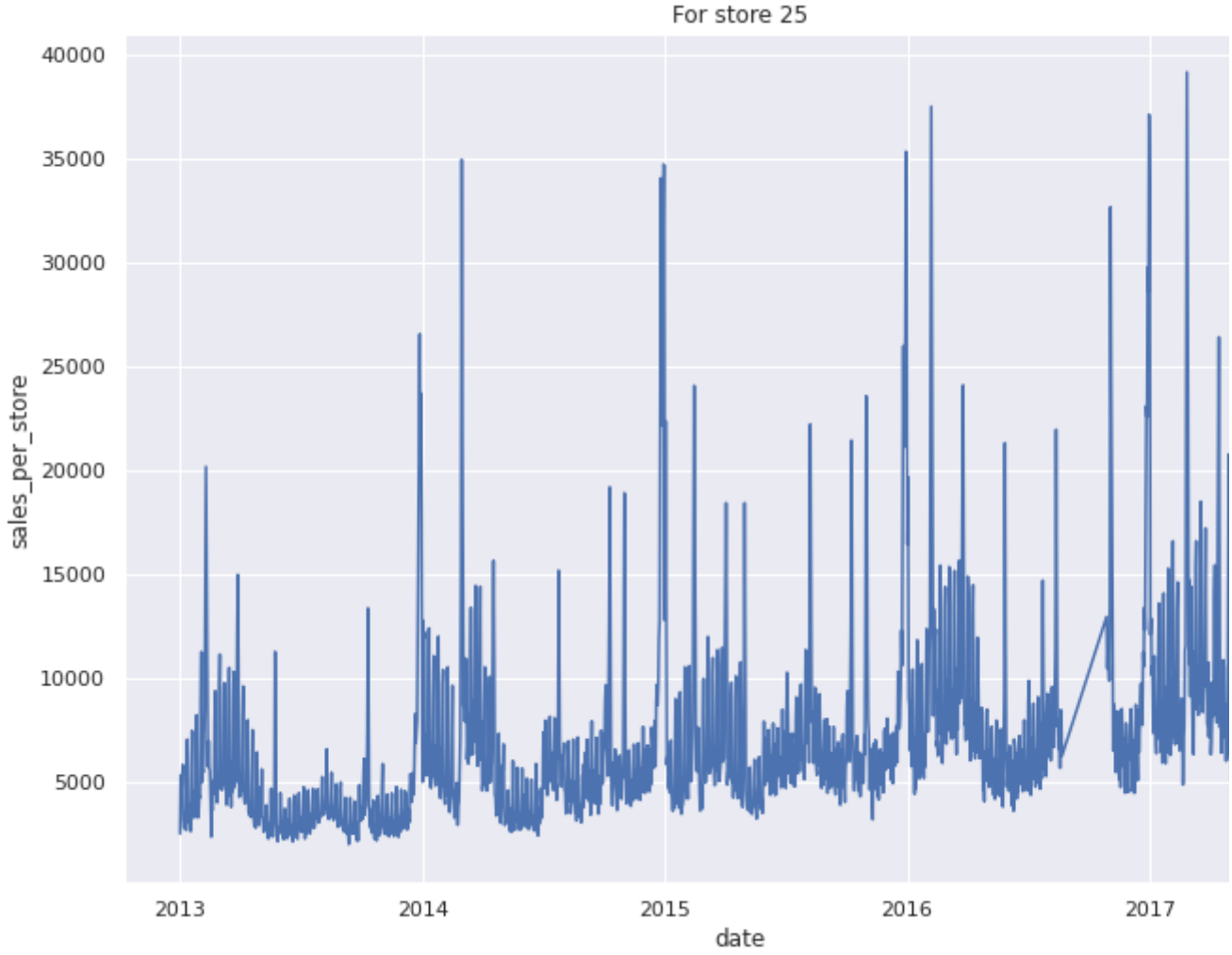
```
date_store_level_data = spark.sql('select date, store_nbr, SUM(unit_sales) as sales_per_st
```

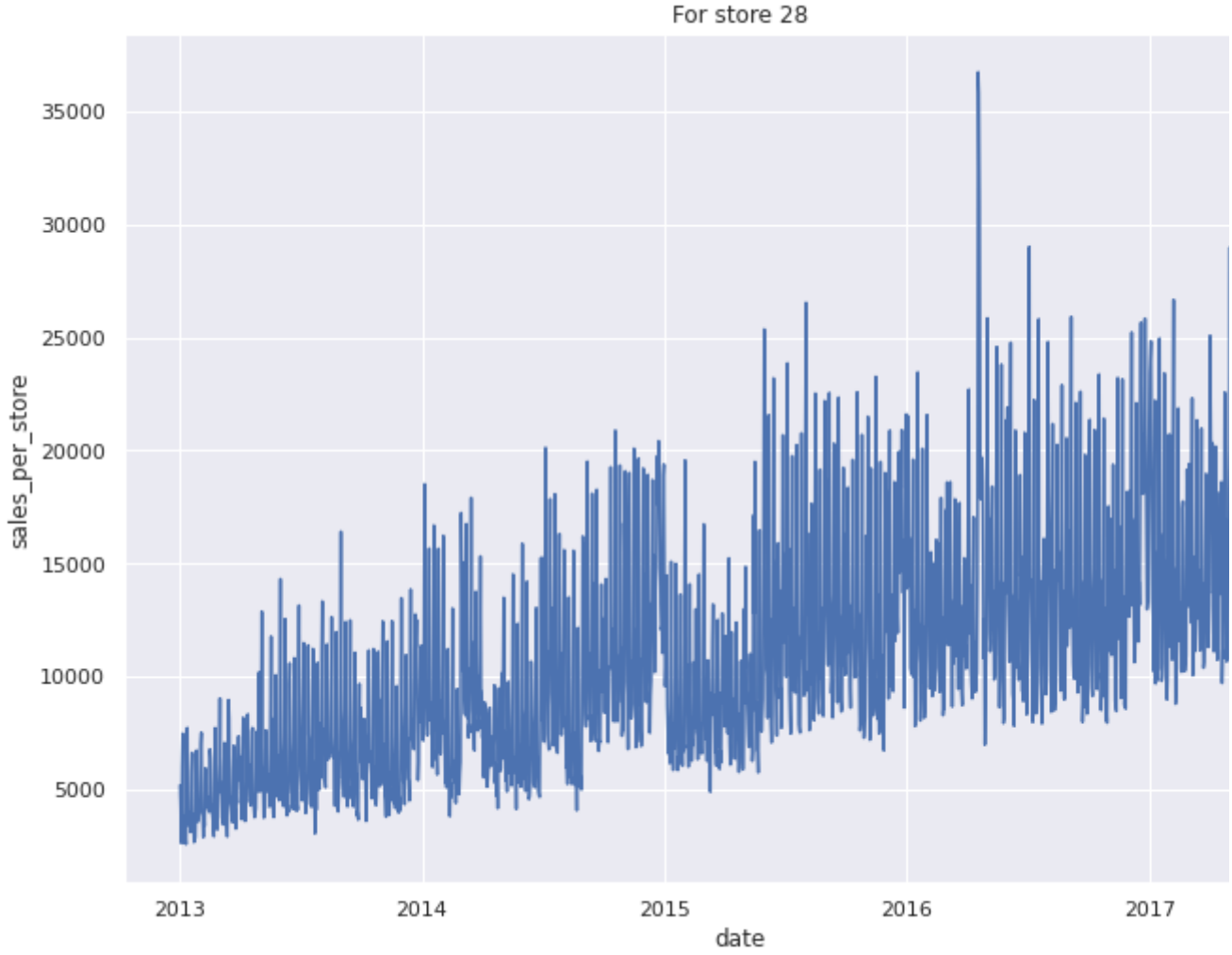
```
date_store_level_data.head()
```

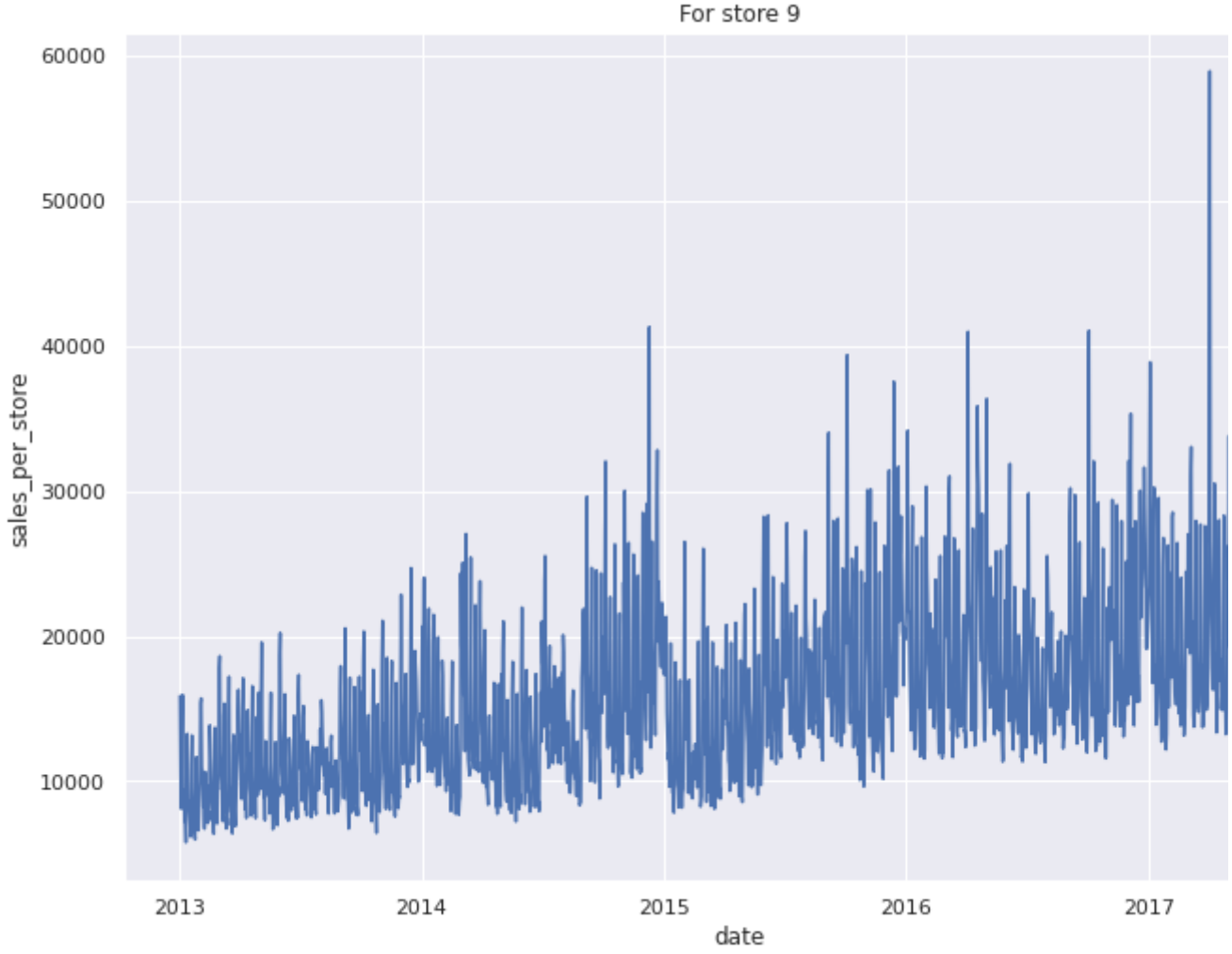
	date	store_nbr	sales_per_store
0	2013-01-01	25	2511.619000
1	2013-01-02	28	5176.473998
2	2013-01-02	9	15867.484008
3	2013-01-02	25	5316.224002
4	2013-01-02	38	9198.507002

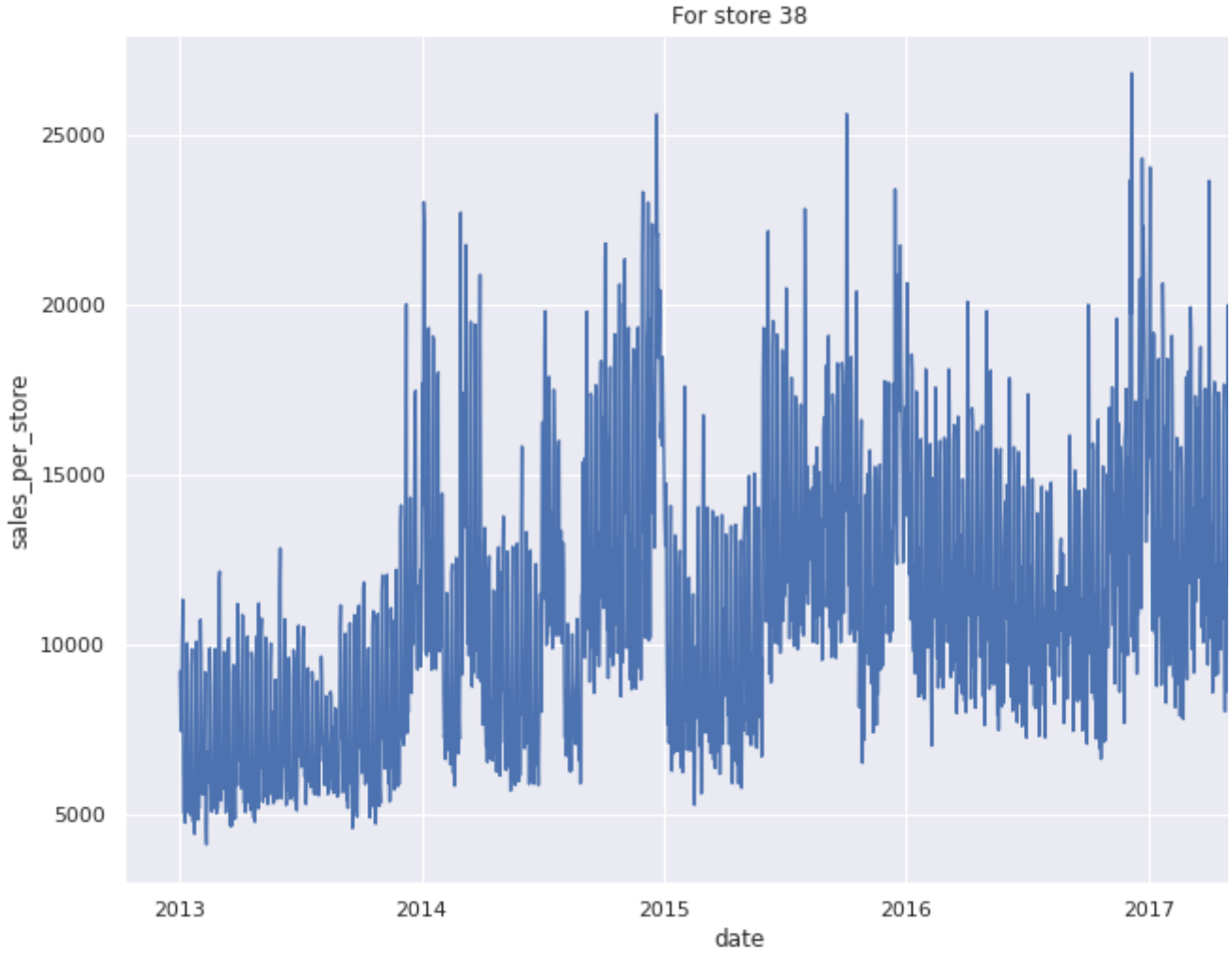
```
store_list = list(date_store_level_data['store_nbr'].unique())
```

```
for i in range(10):
  plot_line_plot(date_store_level_data[date_store_level_data['store_nbr'] == store_list[i]
```

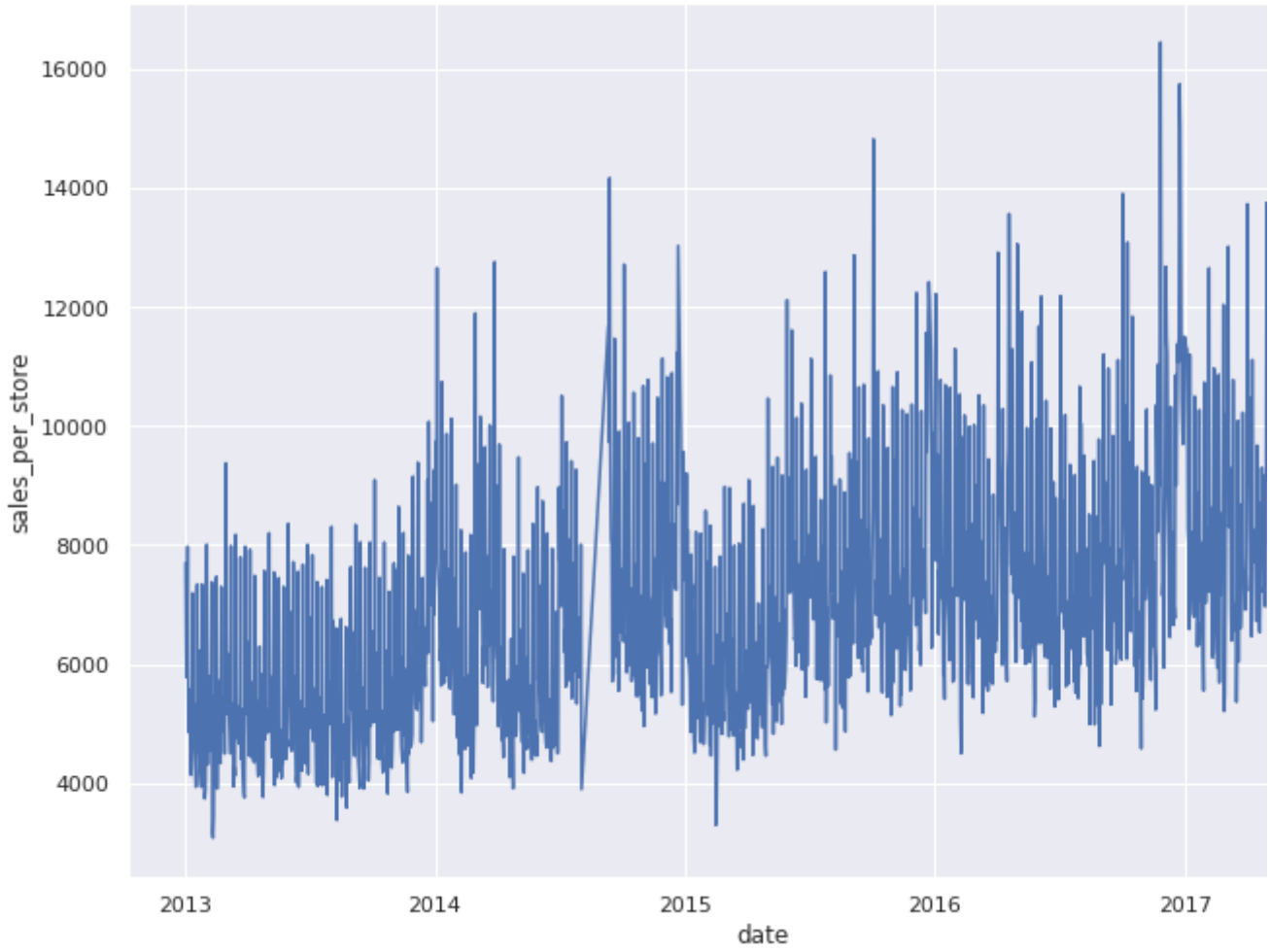

For store 25
sales_per_store
40000
35000
30000
25000
20000
15000
10000
5000
2013
2014
2015
2016
2017
date


For store 28
sales_per_store
35000
30000
25000
20000
15000
10000
5000
2013
2014
2015
2016
2017
date

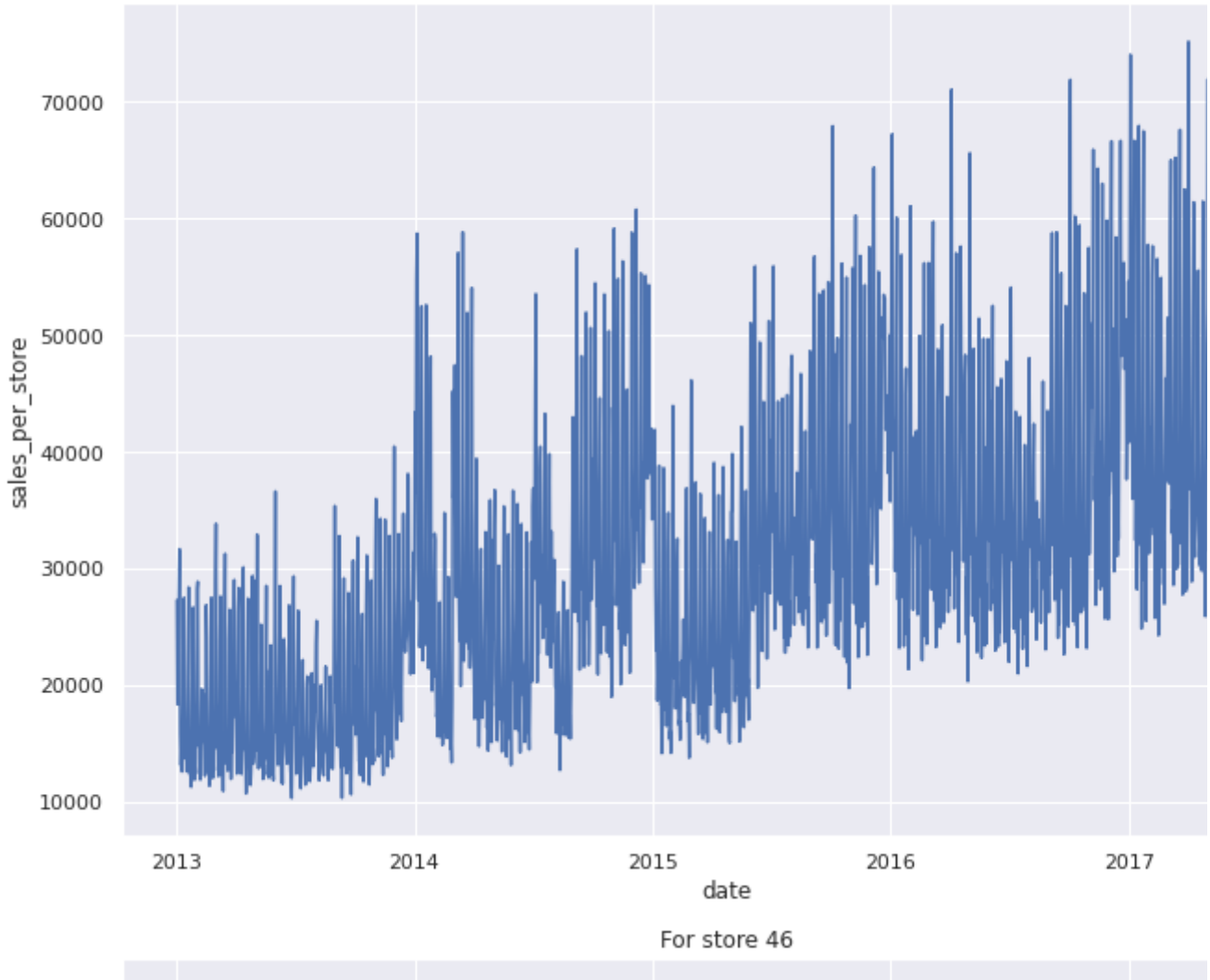

For store 9
60000
50000
40000
30000
20000
10000
sales_per_store
2013
2014
2015
2016
2017
date


For store 38
25000
20000
15000
10000
5000
sales_per_store
2013
2014
2015
2016
2017
date

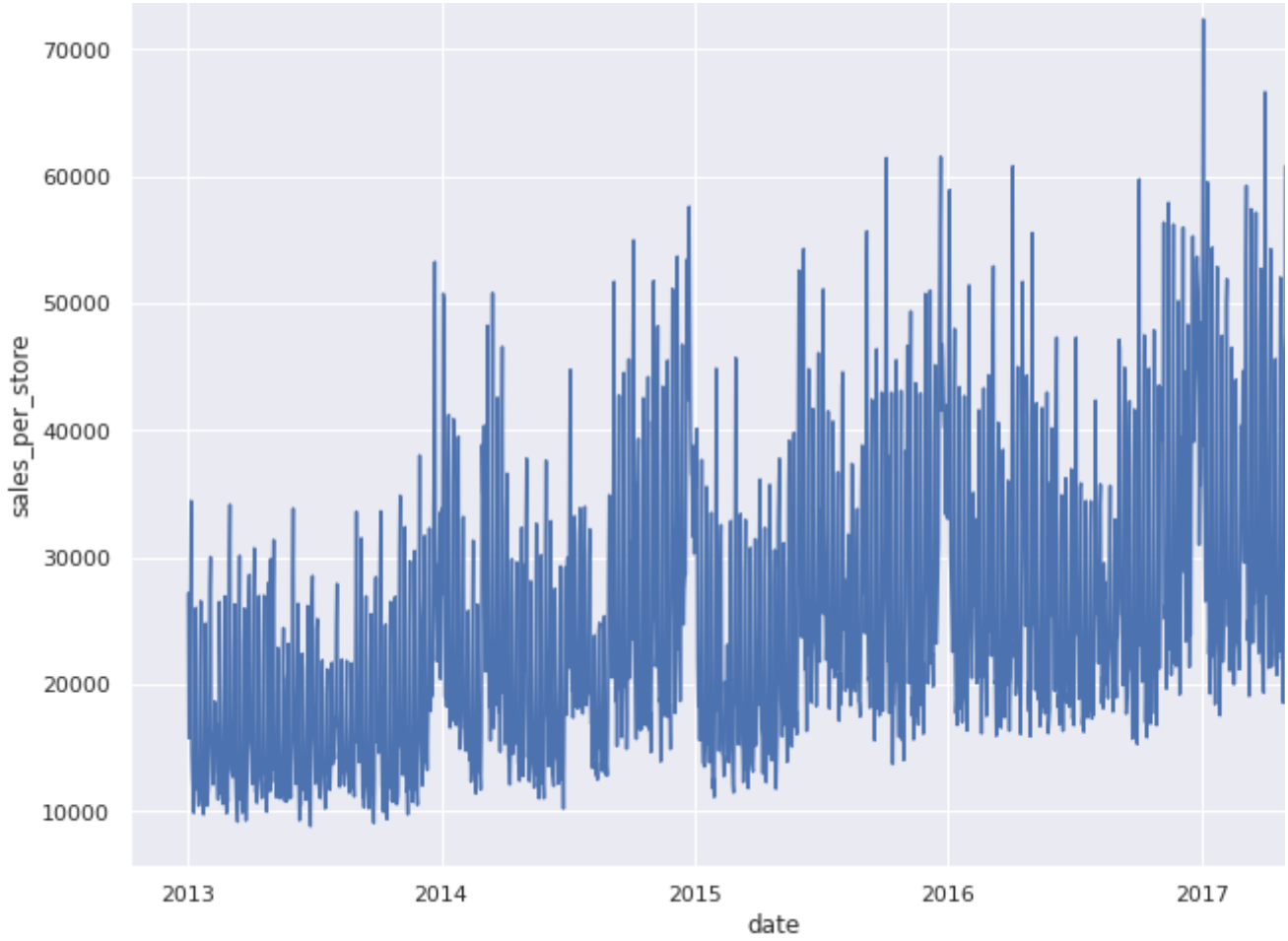
For store 14



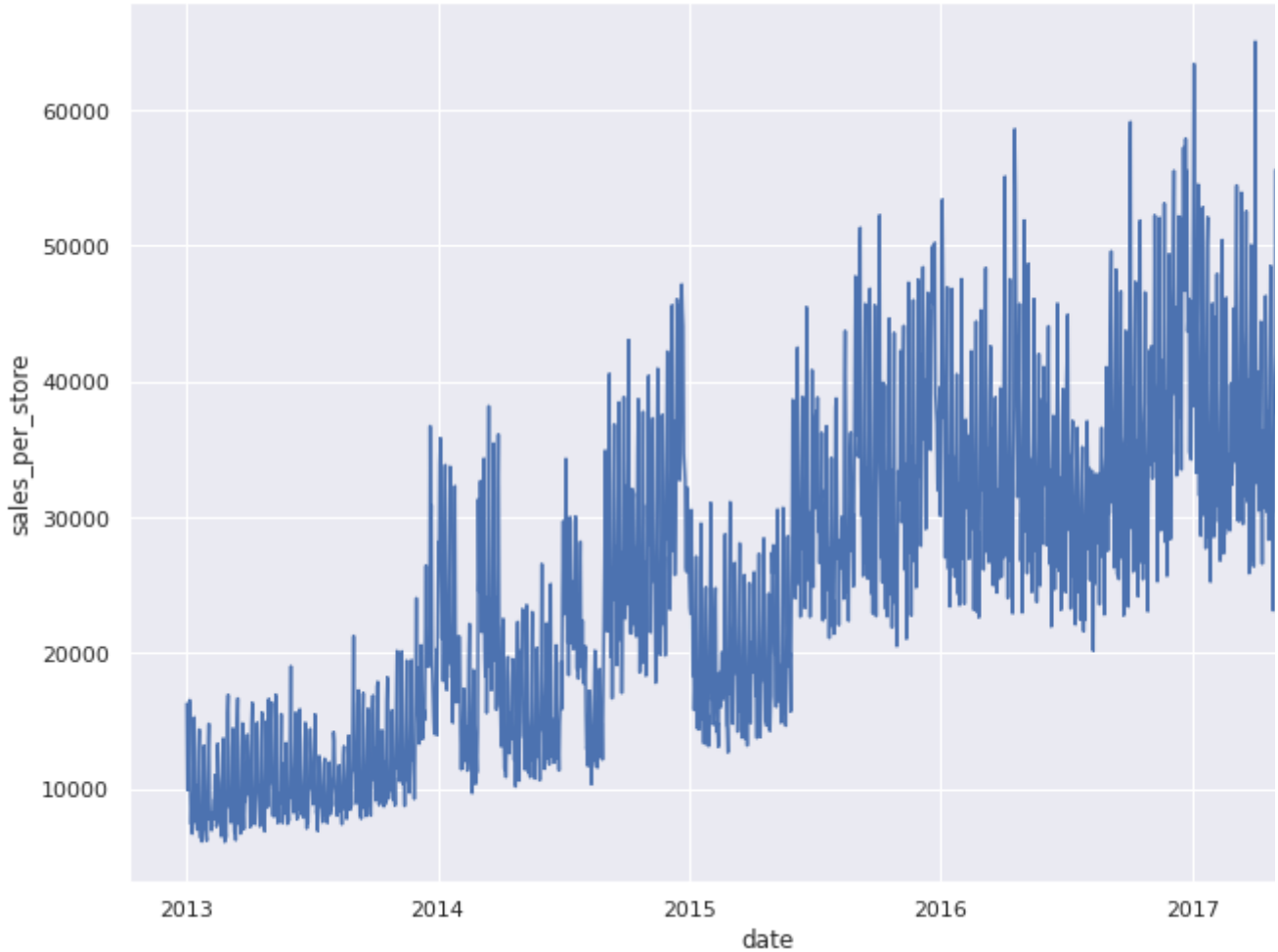
For store 47



For store 46



For store 49



For store 37

We plotted for 10 stores randomly to check the sale of a particular store over time, we did see som be due to the facts that new items are being sold in that store and so the overall sales have increas of the stores....

Sales of an item over time

```
date_item_level_sales = spark.sql('select date, item_nbr, SUM(unit_sales) as sales_per_ite
date_item_level_count = spark.sql('select date, item_nbr, COUNT(1) as sales_per_item FROM
```

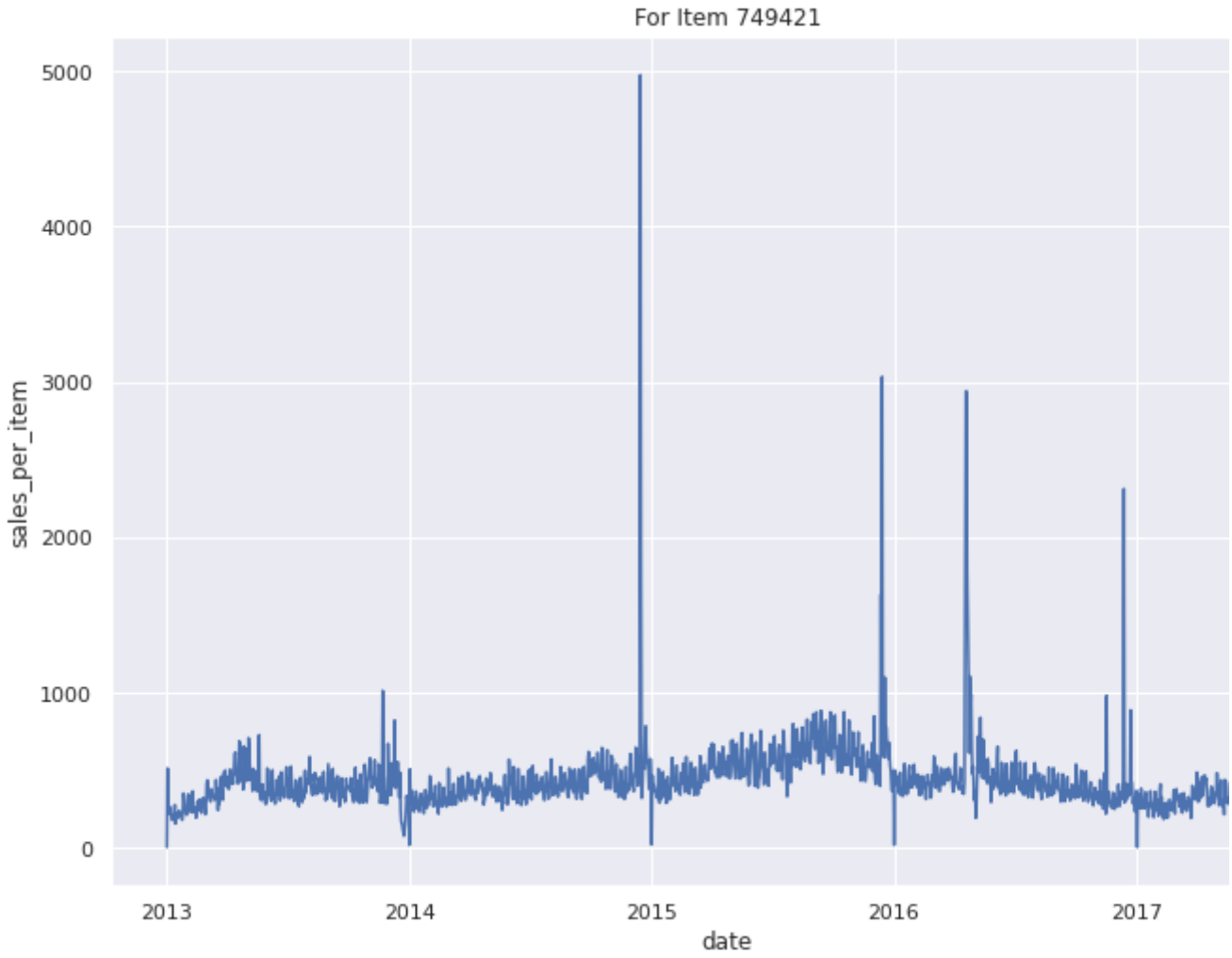
```
date_item_level_sales.head()
```

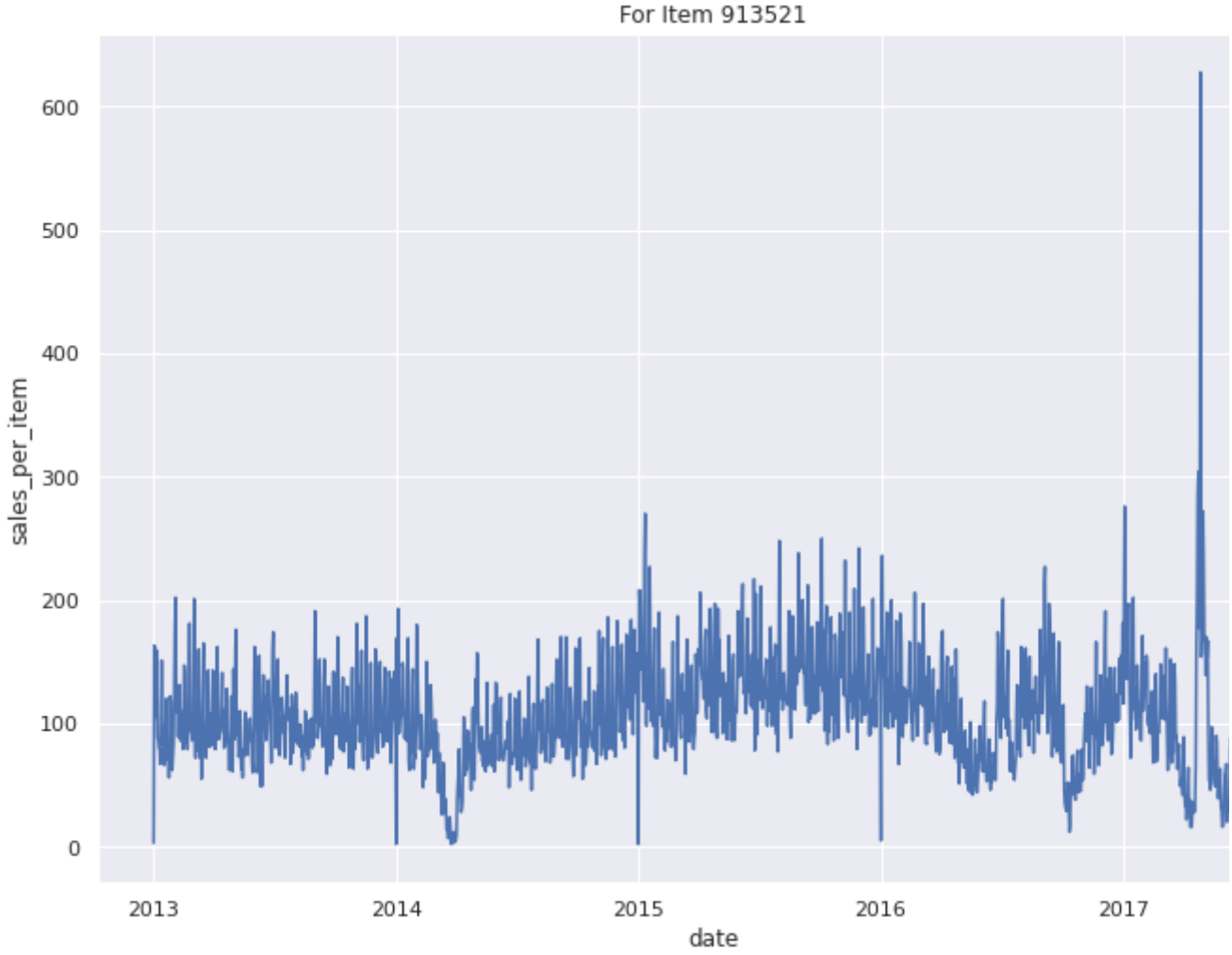
	date	item_nbr	sales_per_item
0	2013-01-01	749421	7.0
1	2013-01-01	913521	3.0
2	2013-01-01	587069	1.0
3	2013-01-01	636846	1.0
4	2013-01-01	802832	2.0

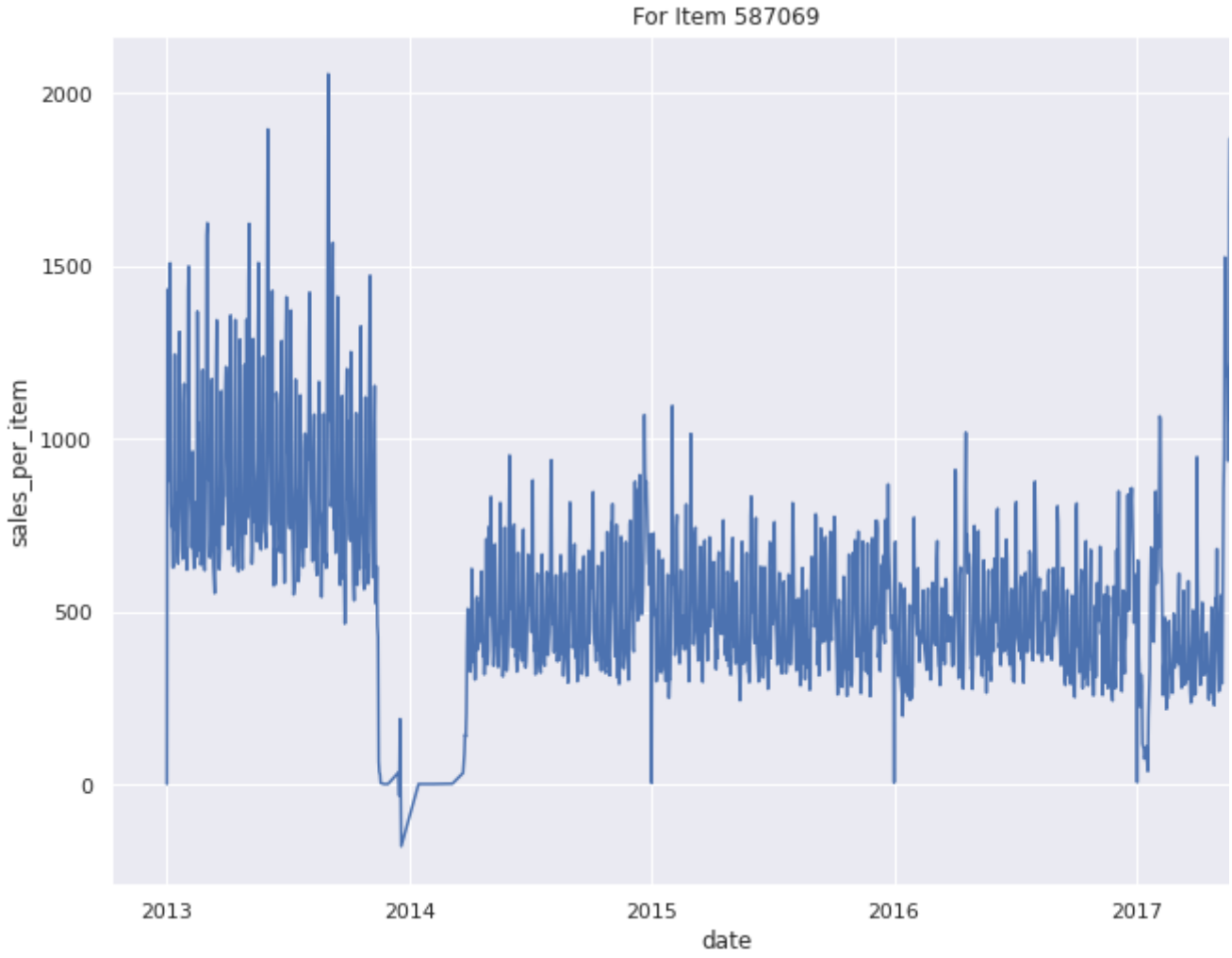

```
item_list = list(date_item_level_sales['item_nbr'].unique())
```

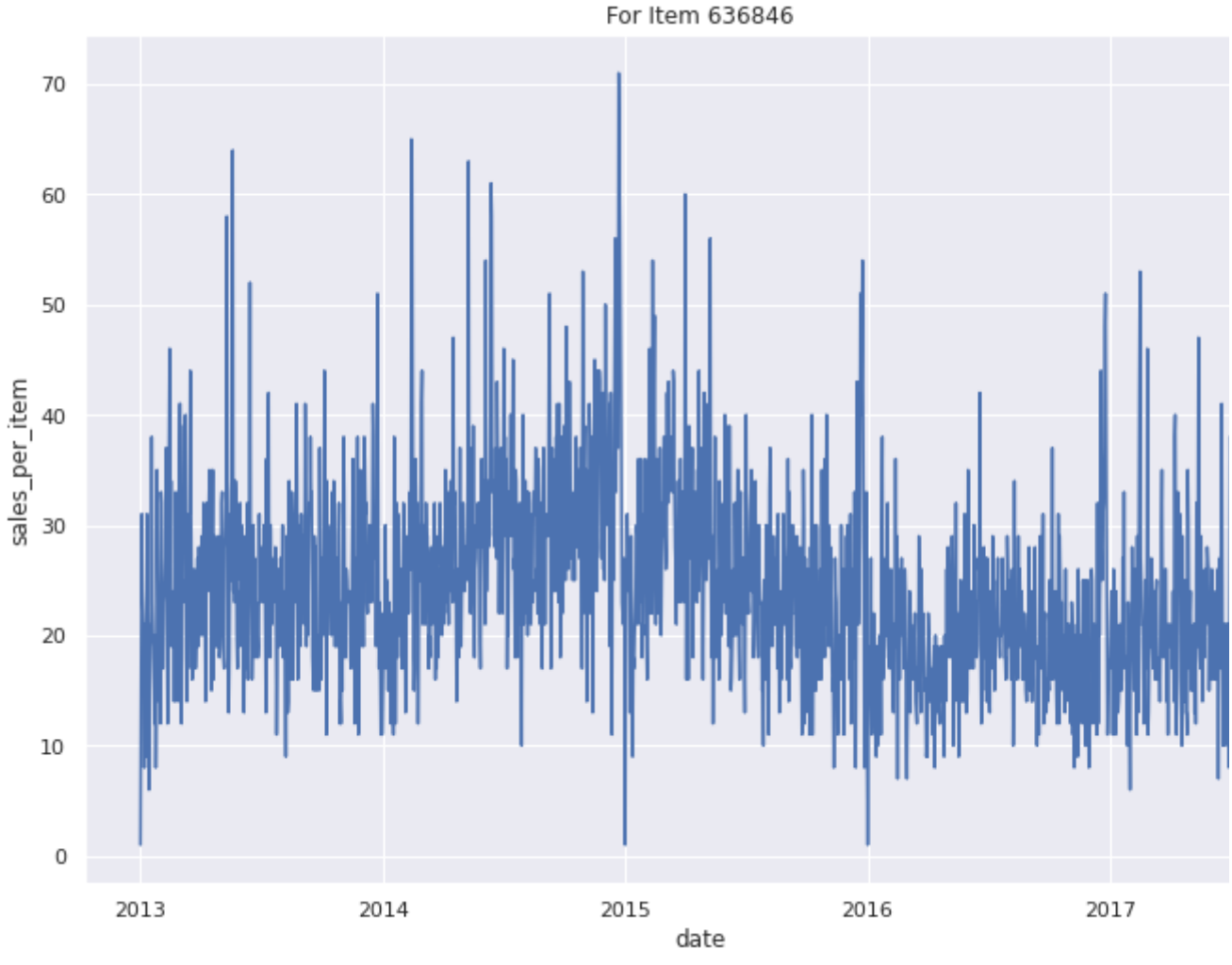


```
for i in range(25):
  plot_line_plot(date_item_level_sales[date_item_level_sales['item_nbr'] == item_list[i]][
```

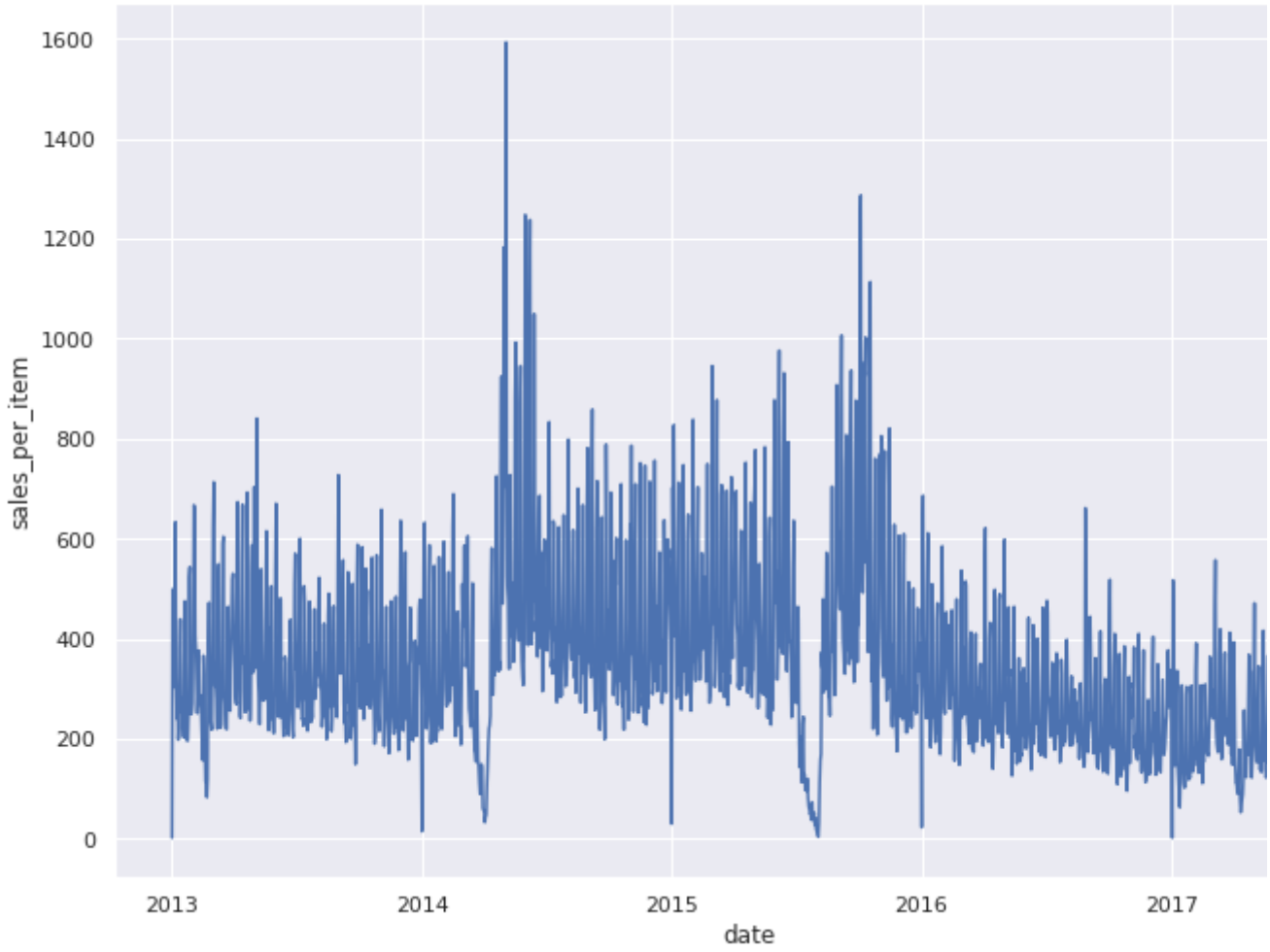

For Item 749421
sales_per_item
0
1000
2000
3000
4000
5000
2013
2014
2015
2016
2017
date


For Item 913521
sales_per_item
0
100
200
300
400
500
600
2013
2014
2015
2016
2017
date

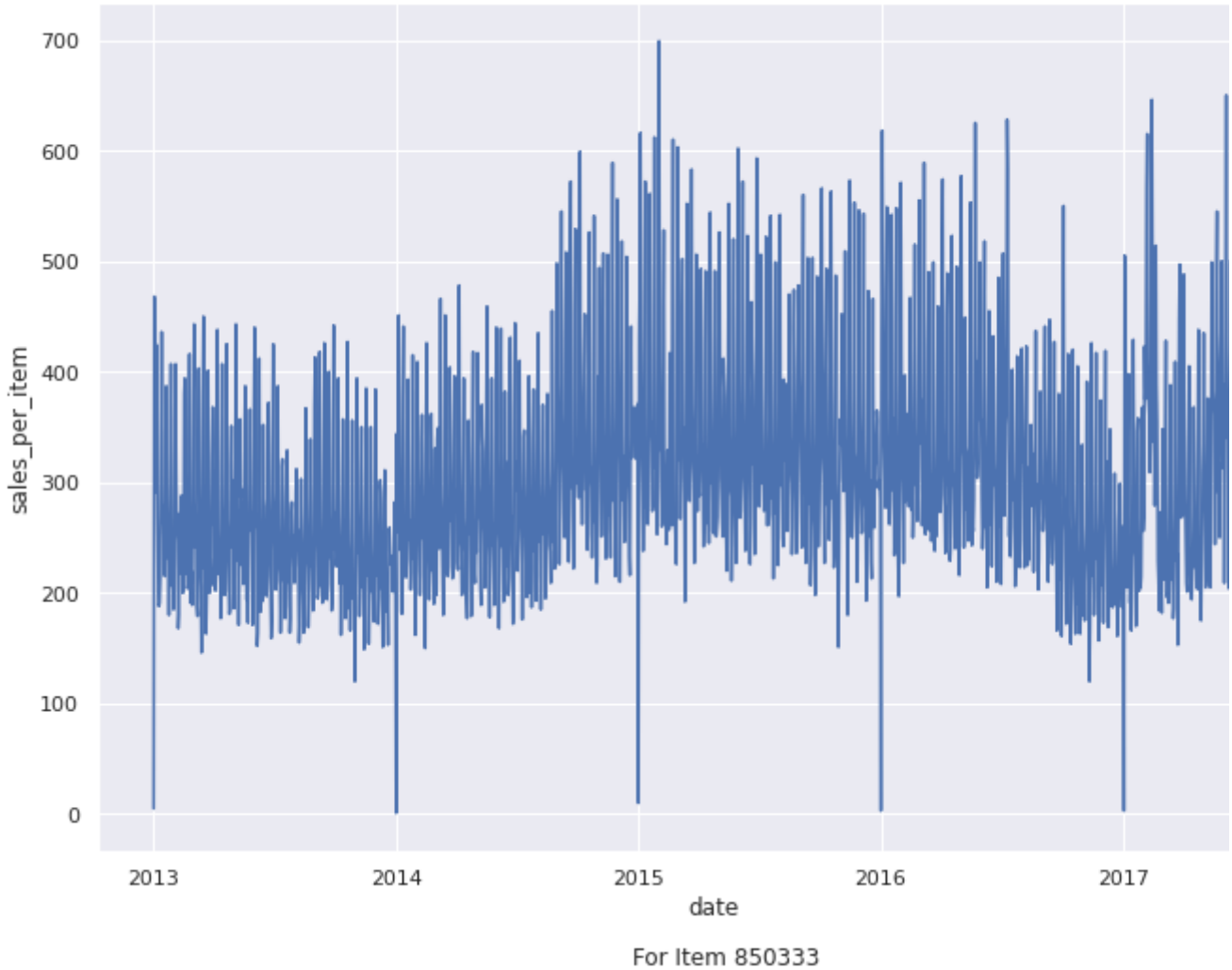




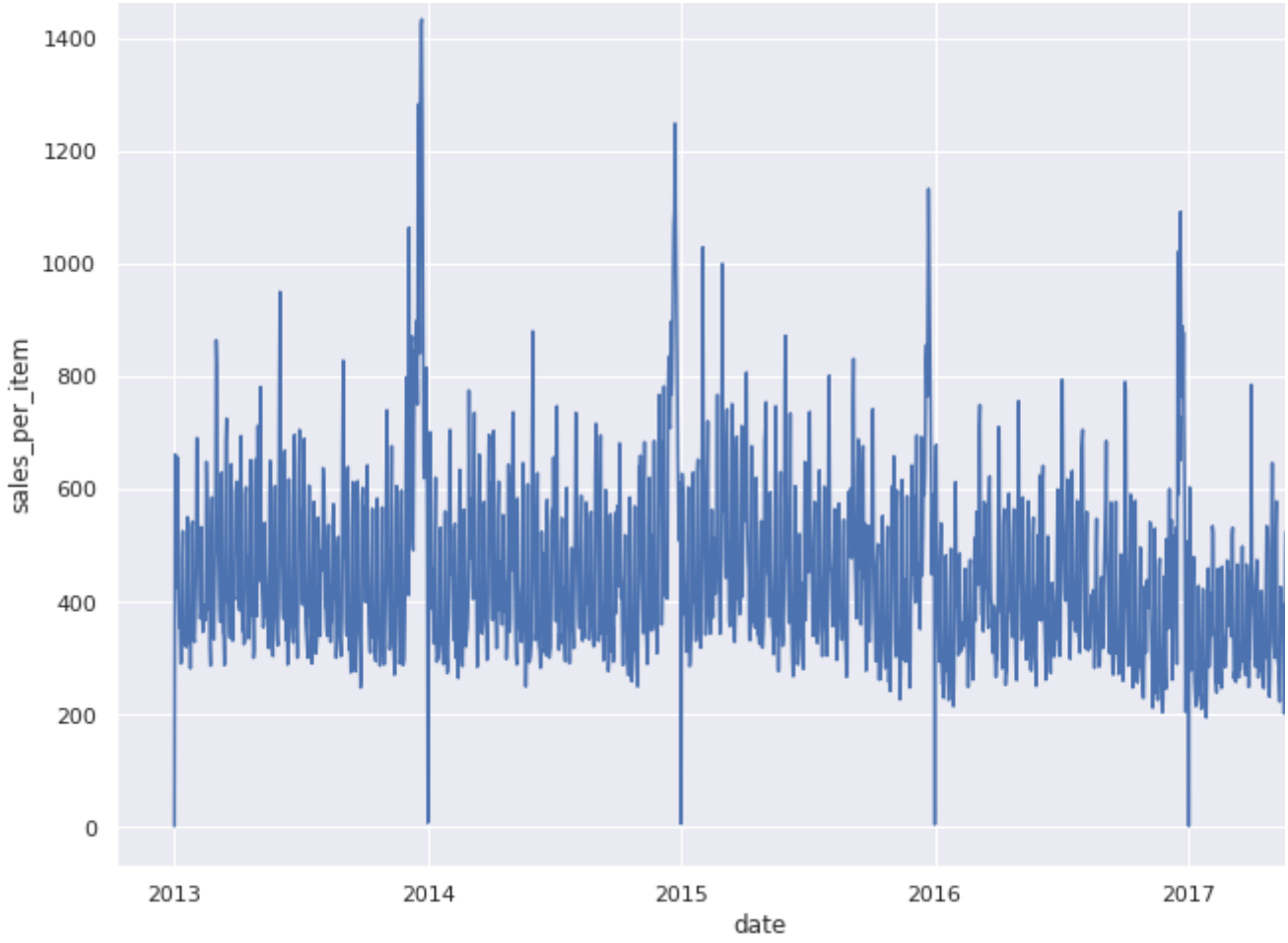
For Item 802832



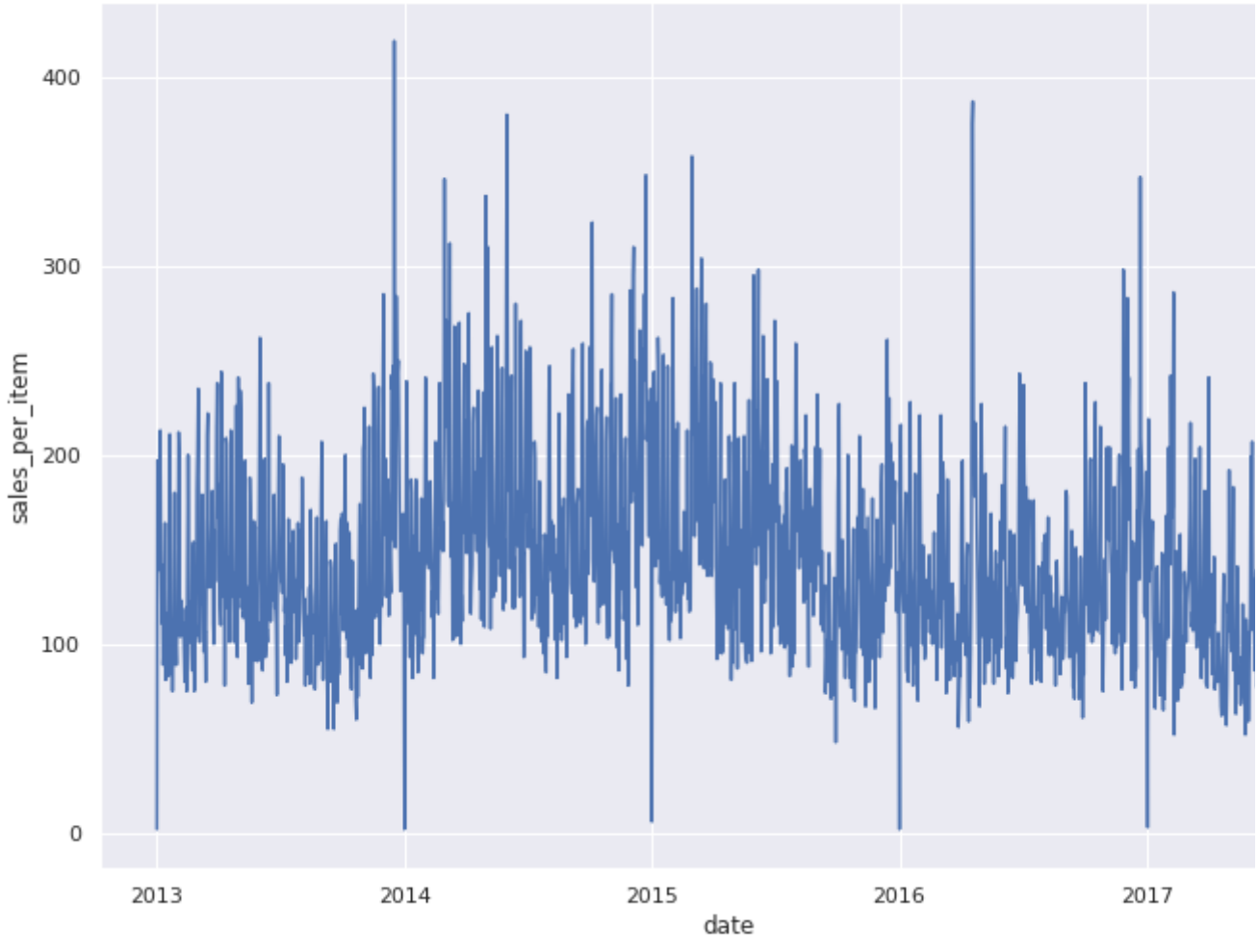
For Item 838407



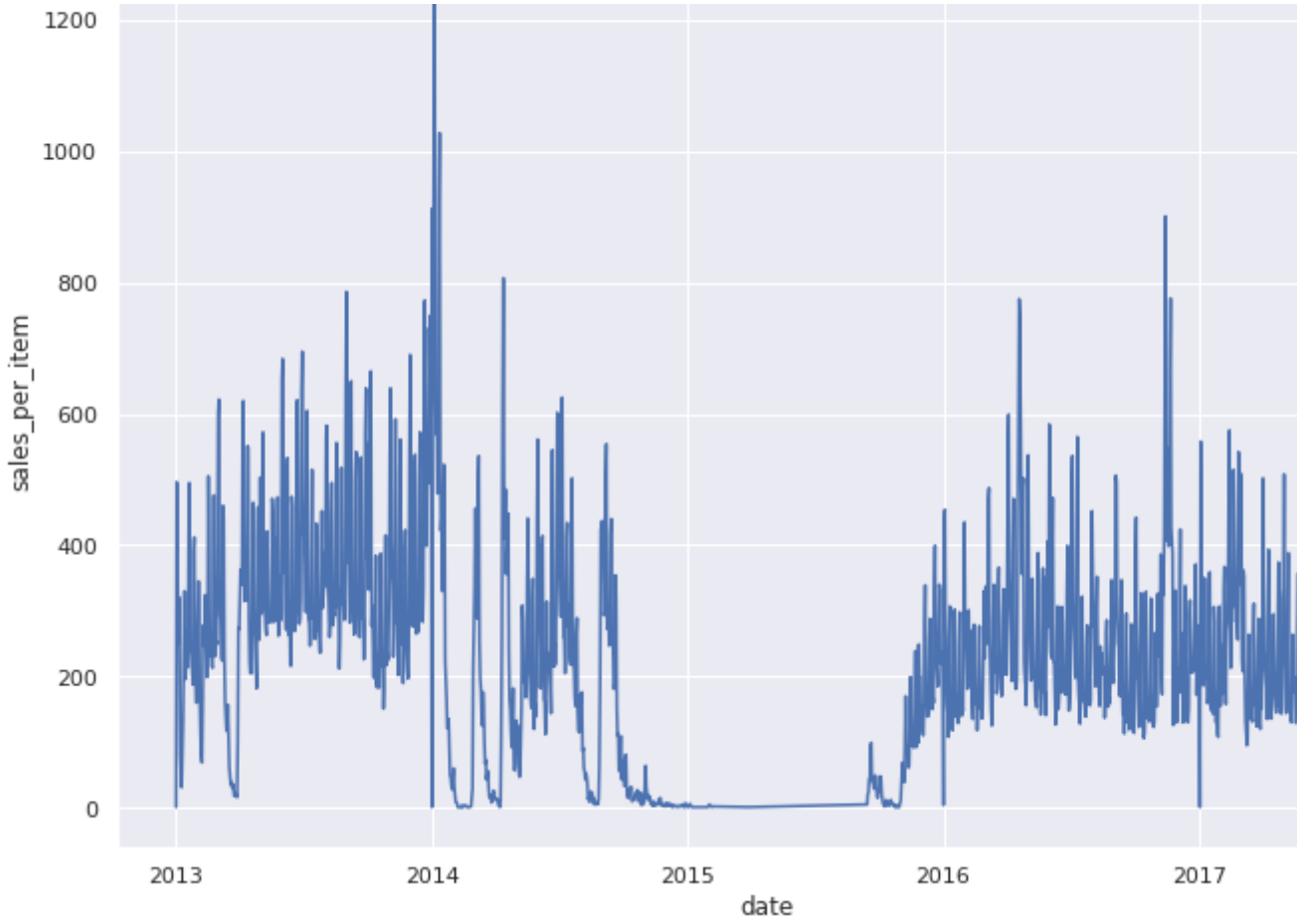
For Item 850333



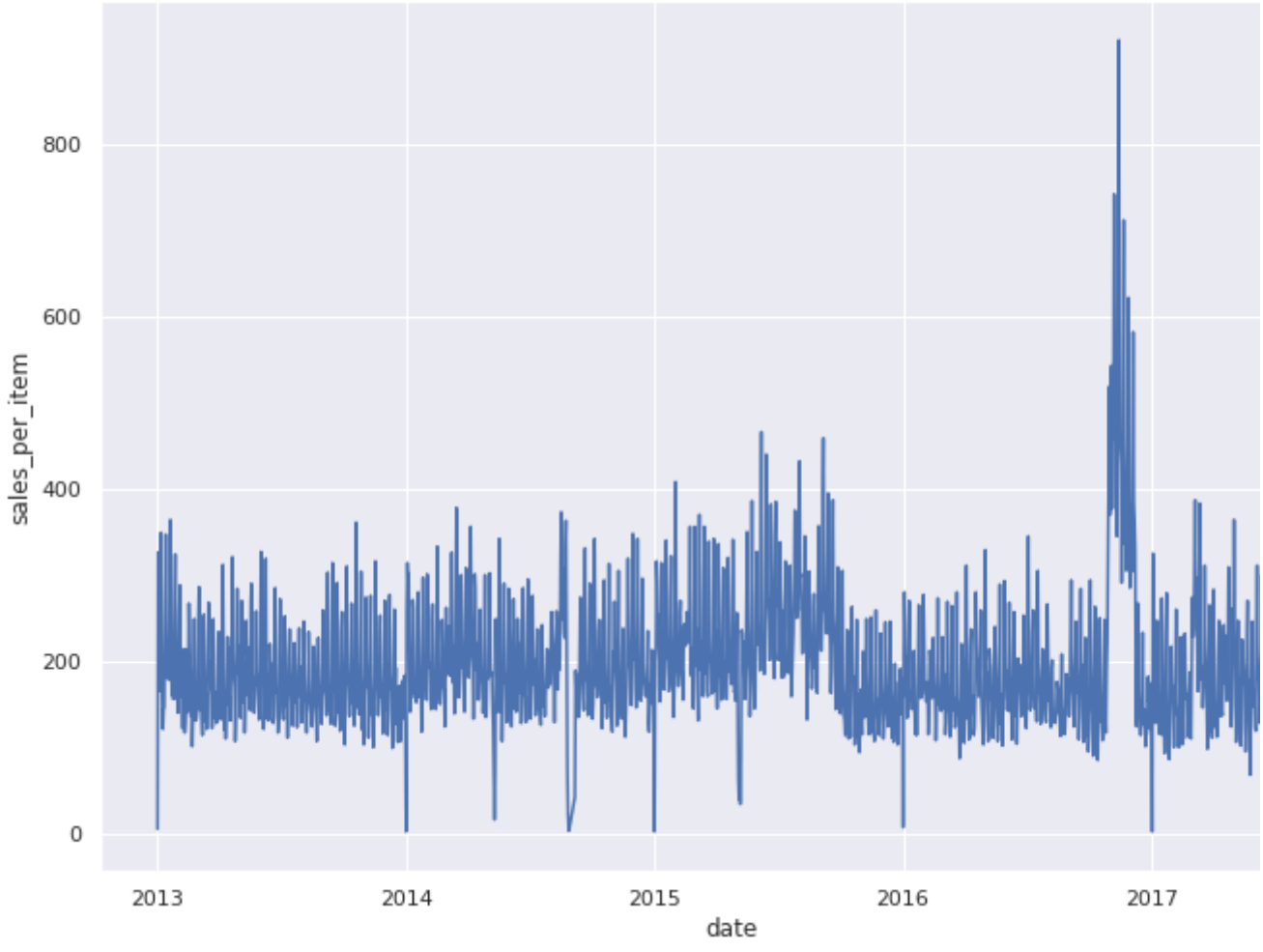
For Item 991329



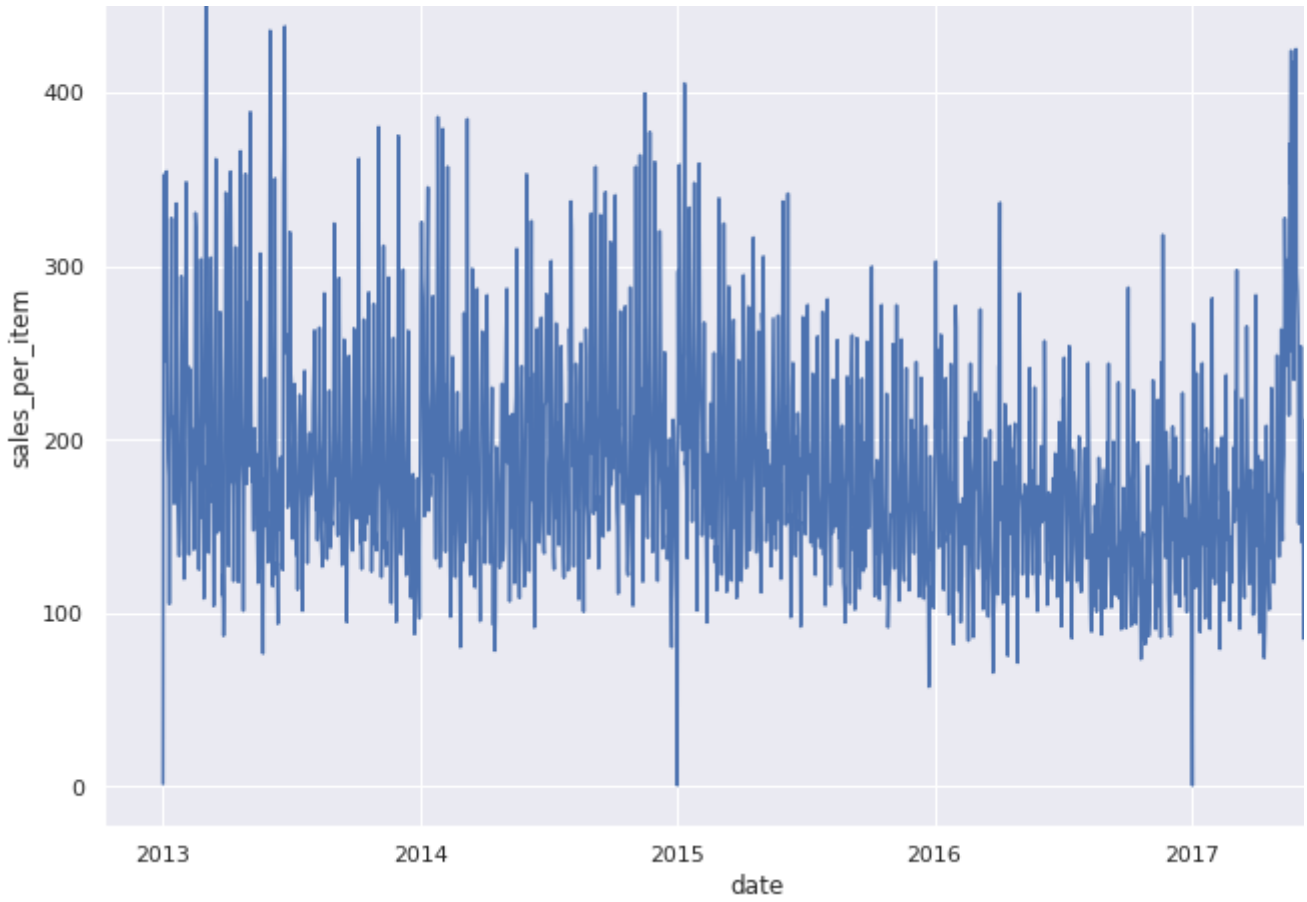
For Item 1102268

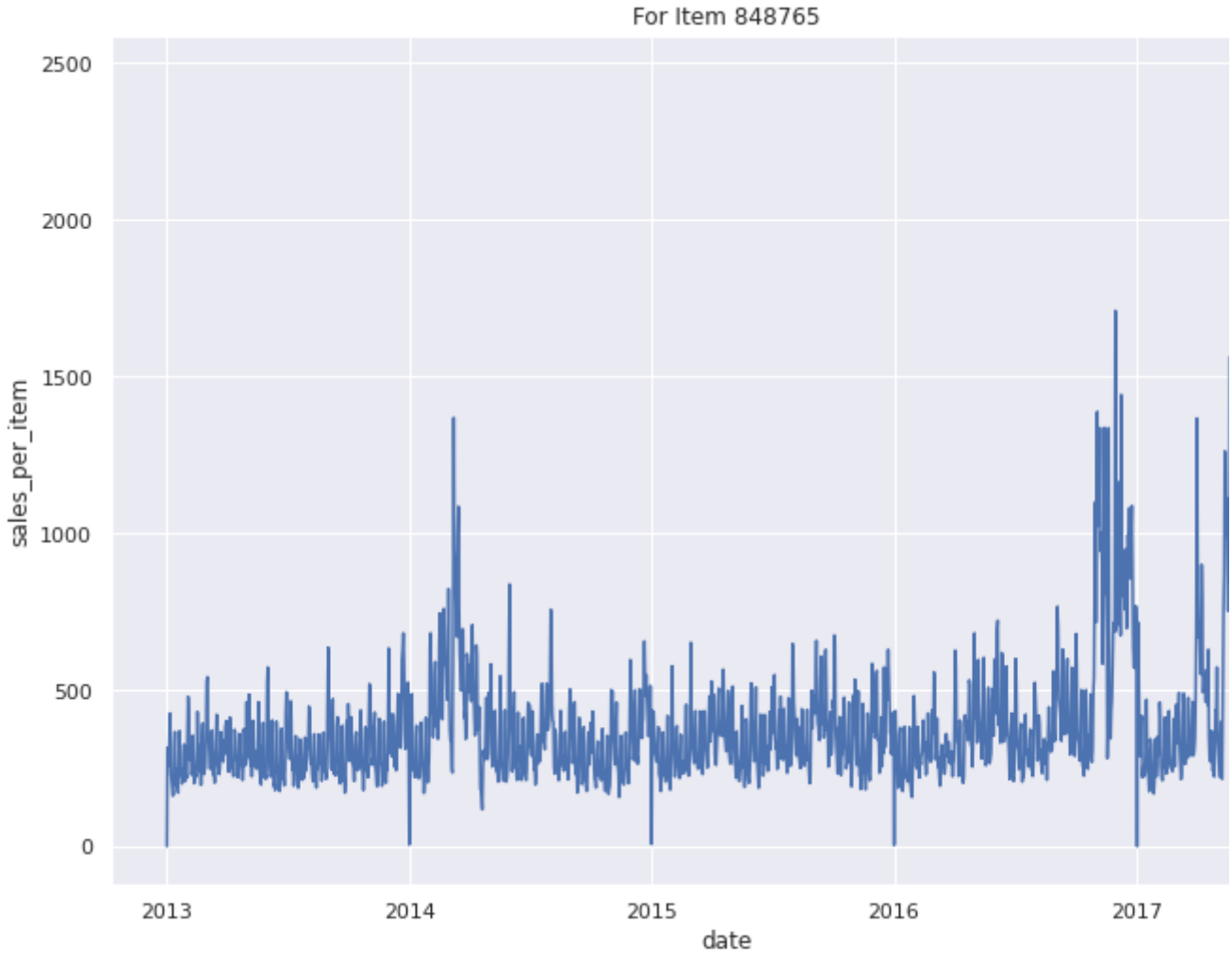


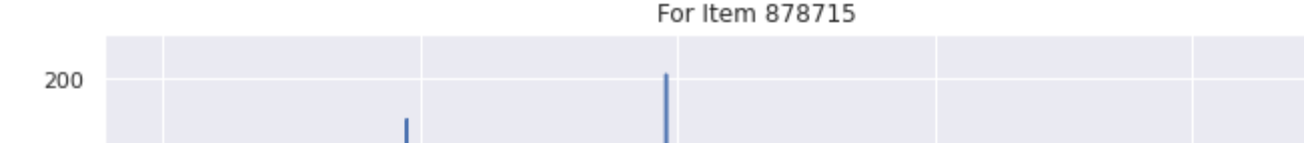
For Item 459762

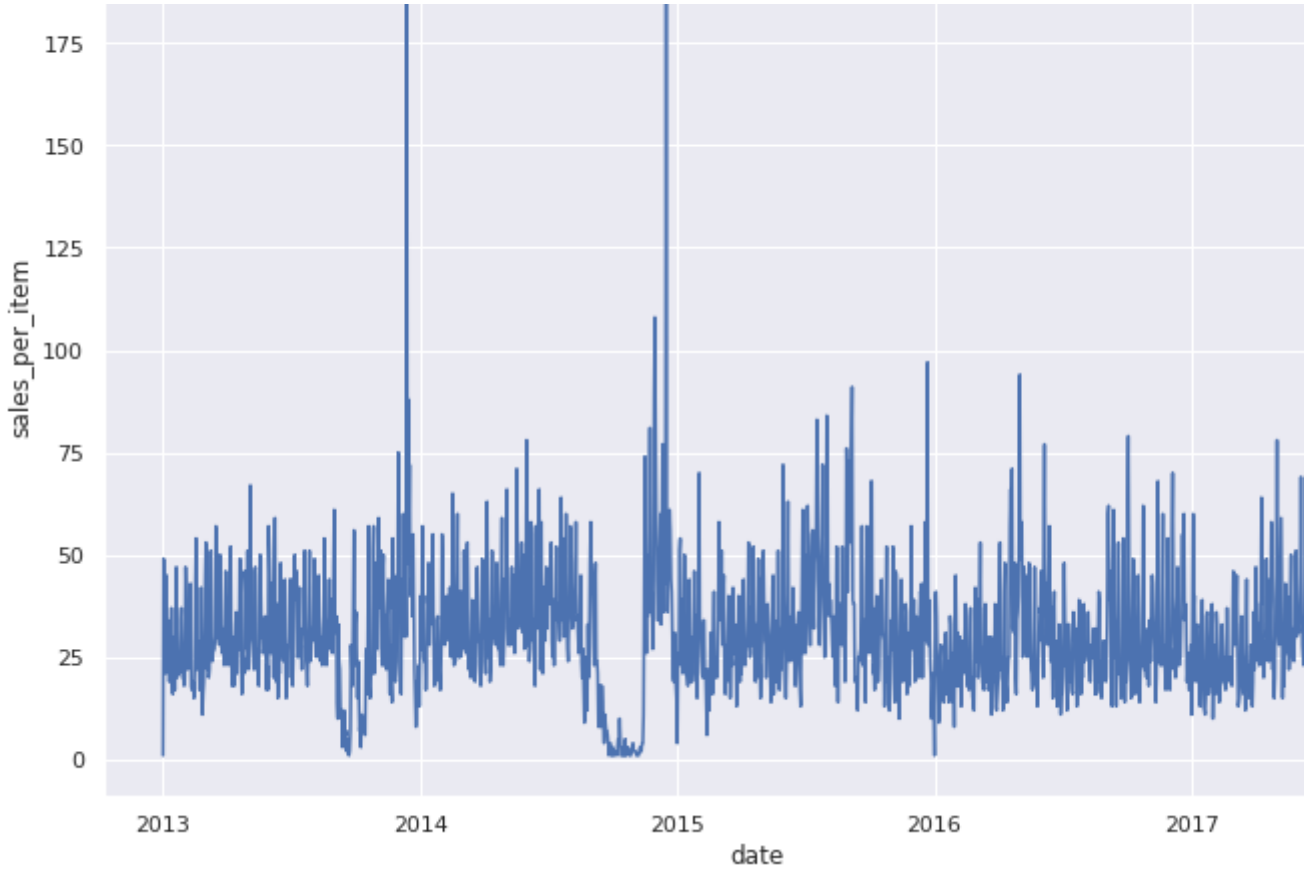


For Item 583868

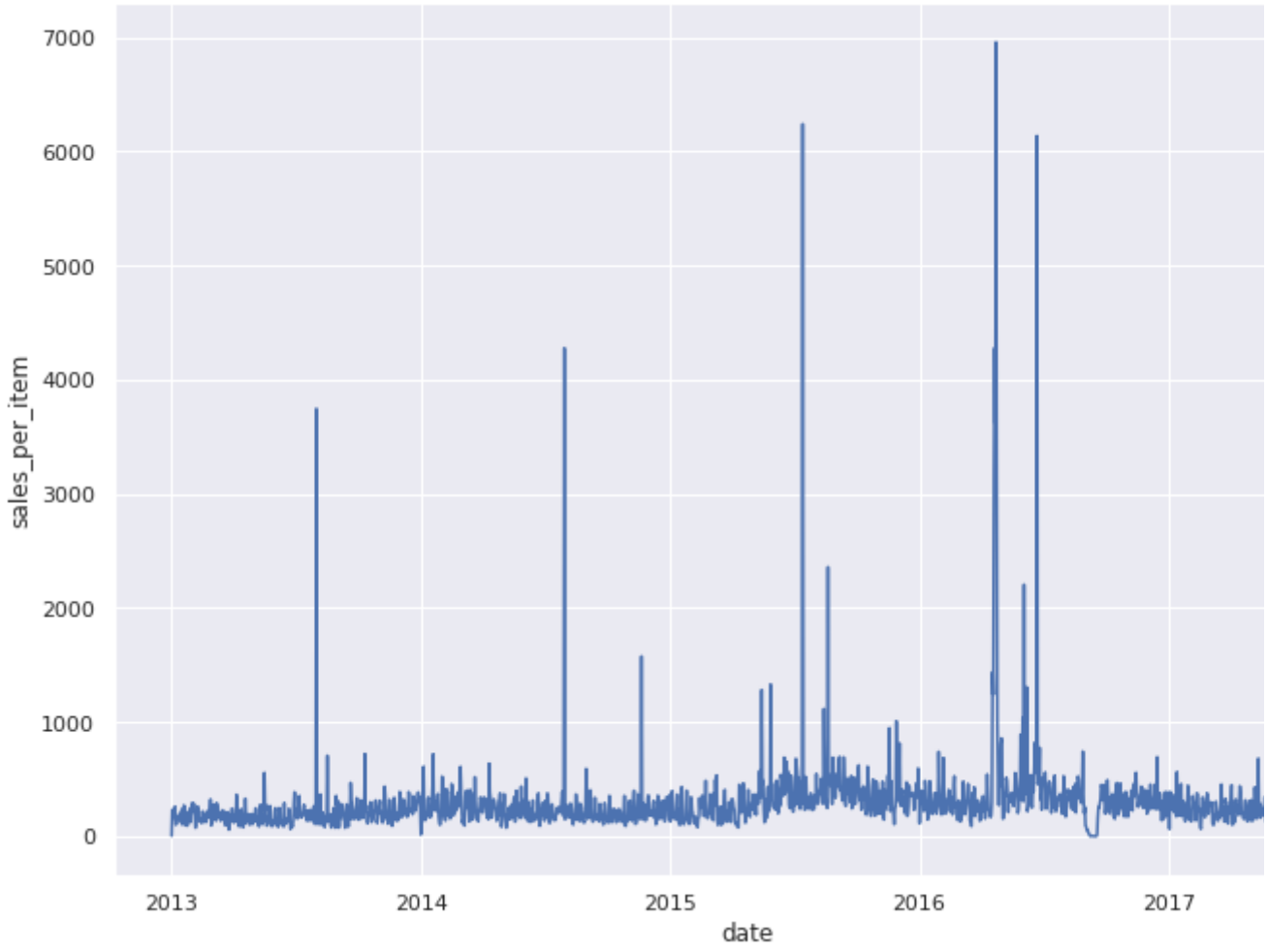

400
300
200
100
0
sales_per_item
2013
2014
2015
2016
2017
date


For Item 848765
2500
2000
1500
1000
500
0
sales_per_item
2013
2014
2015
2016
2017
date

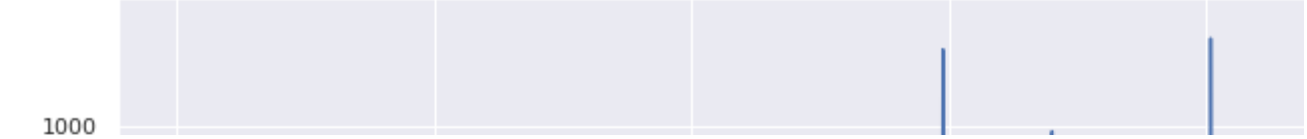

For Item 878715
200

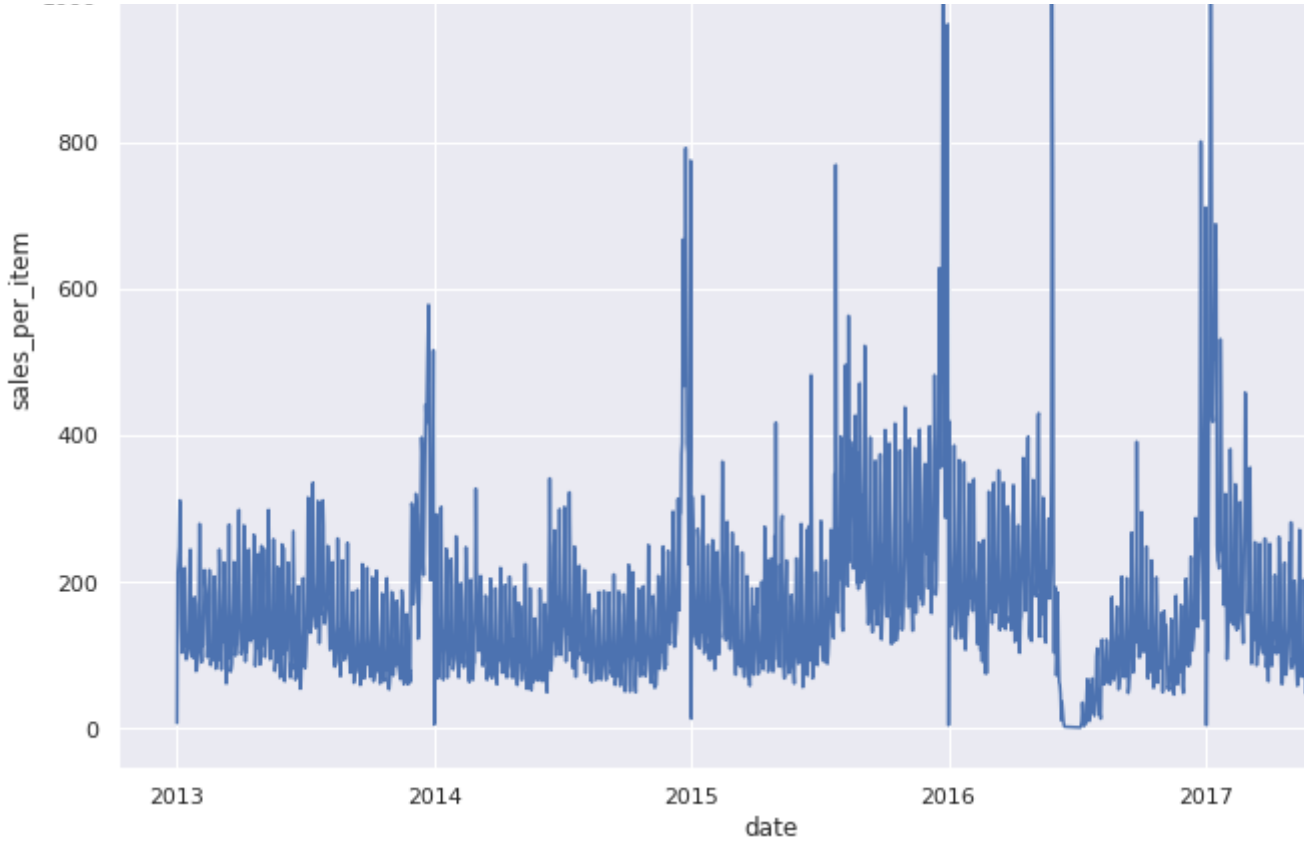


For Item 906979

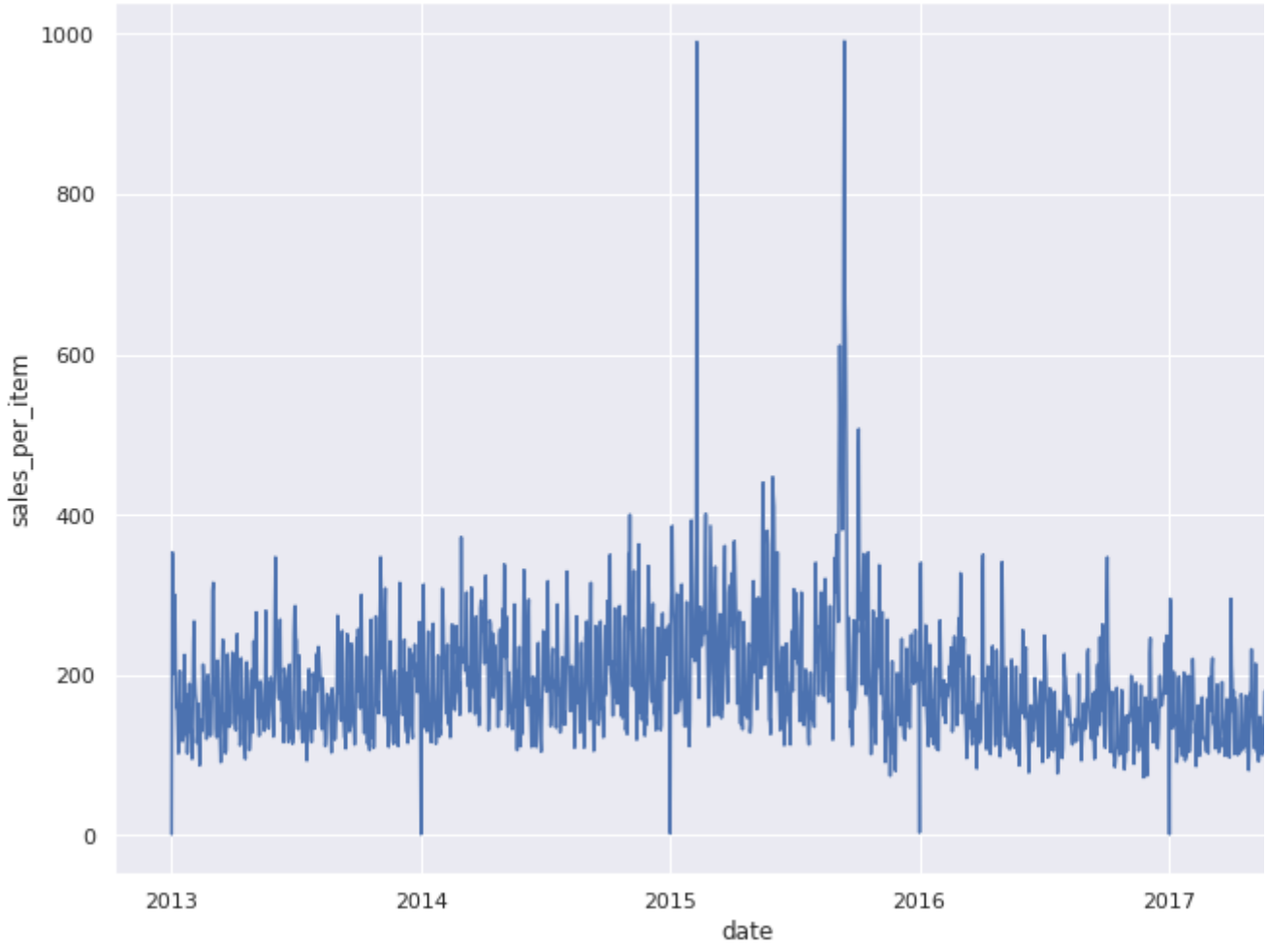


For Item 1047717

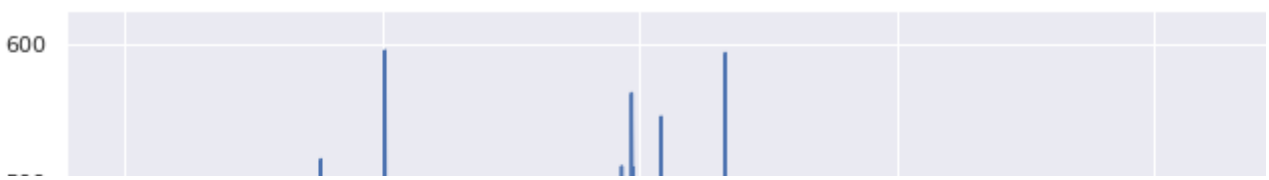


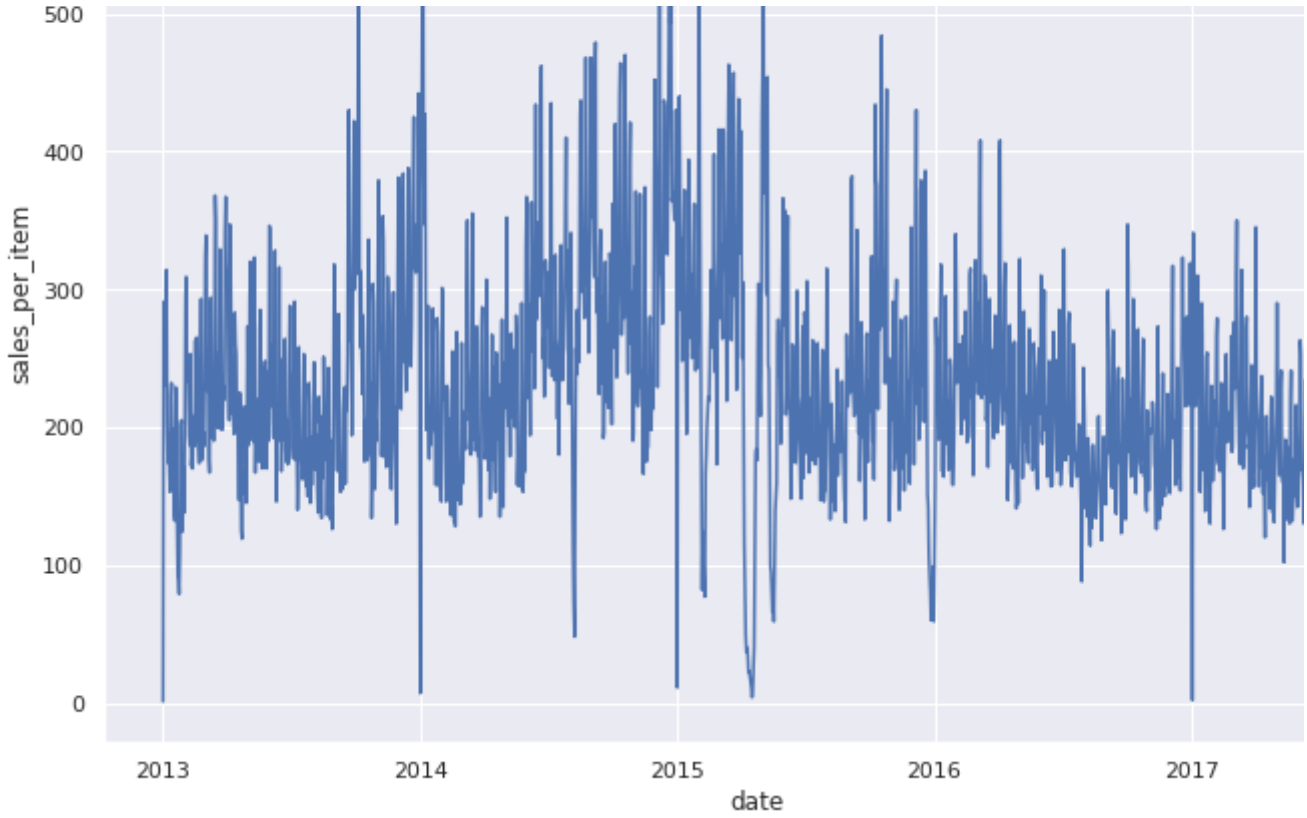


For Item 214862

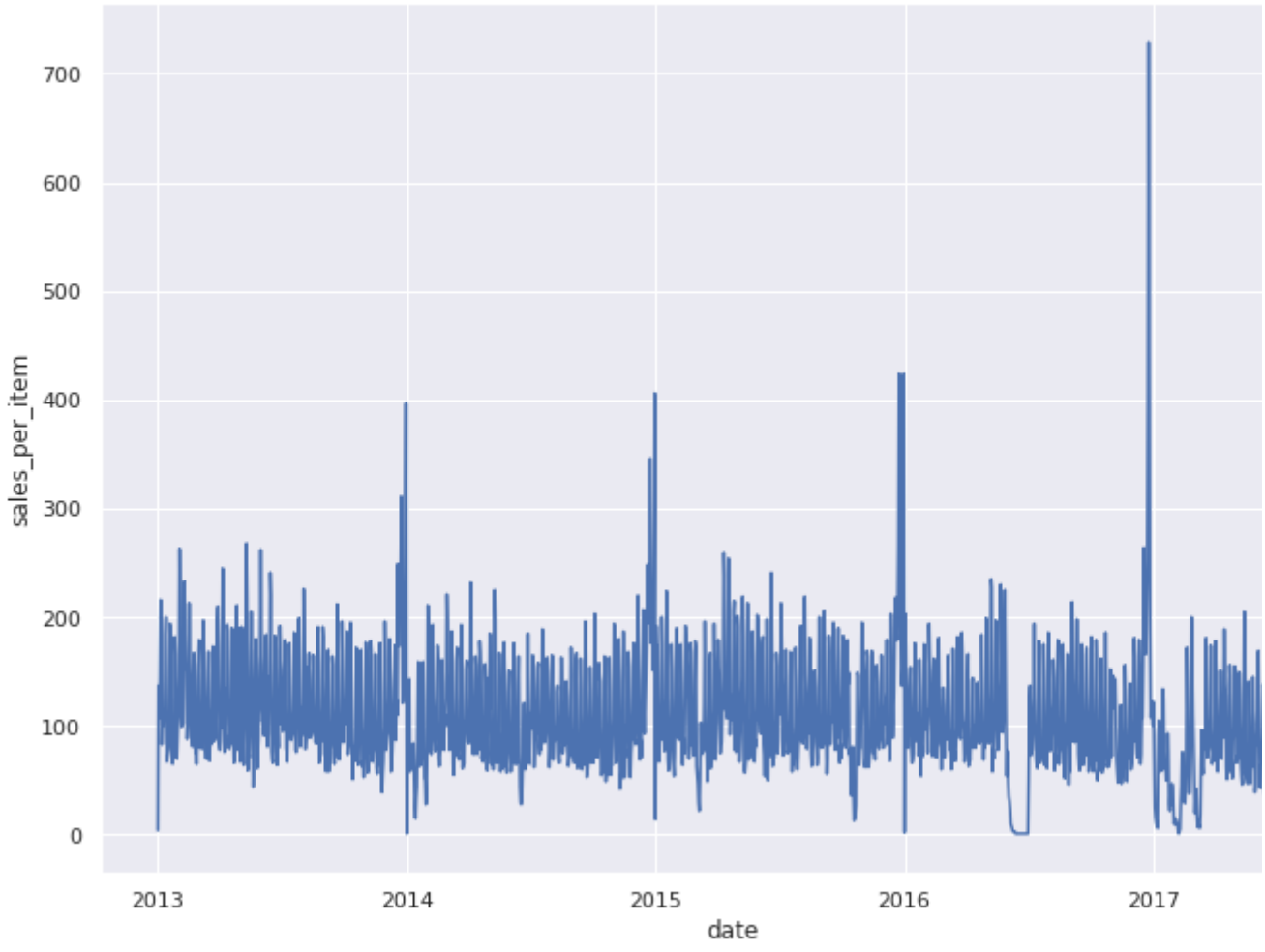


For Item 407769

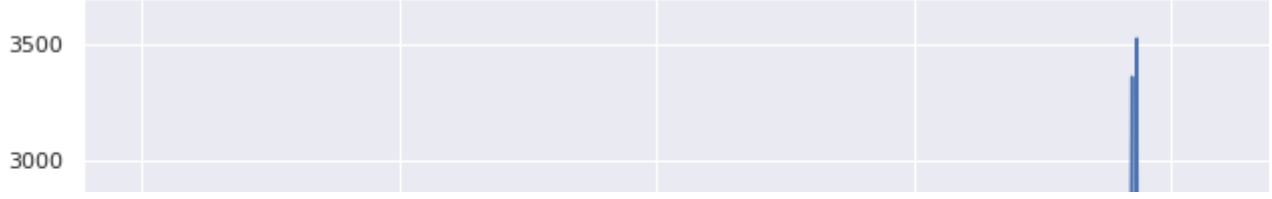


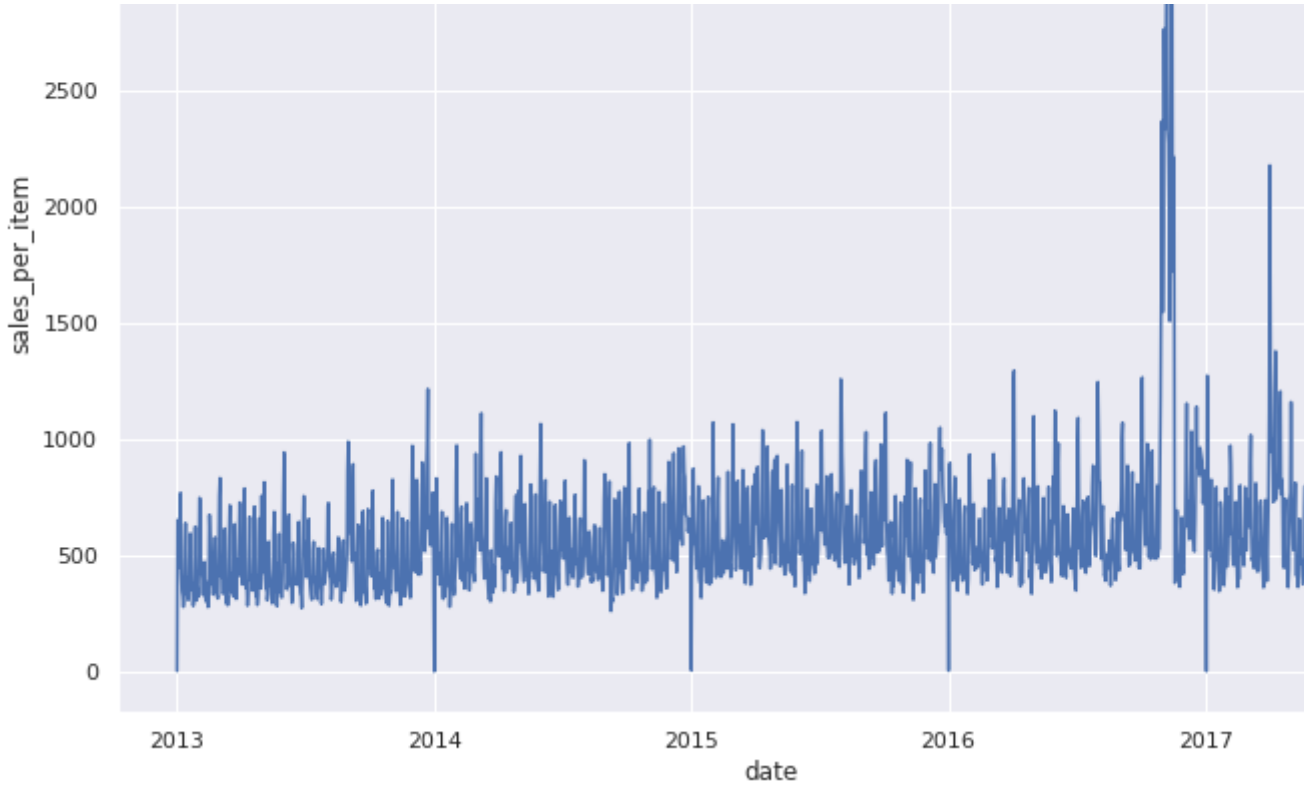


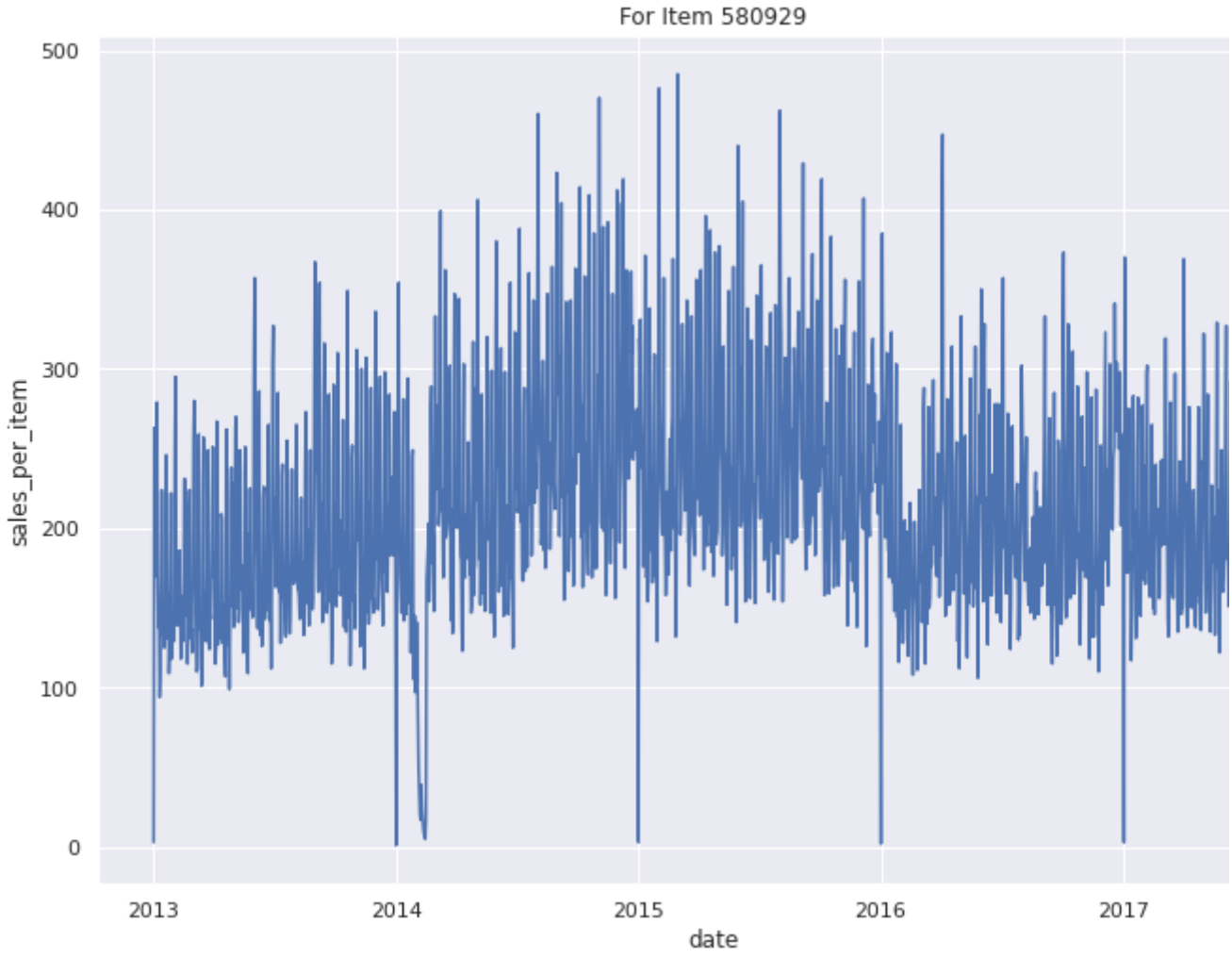
For Item 1047705



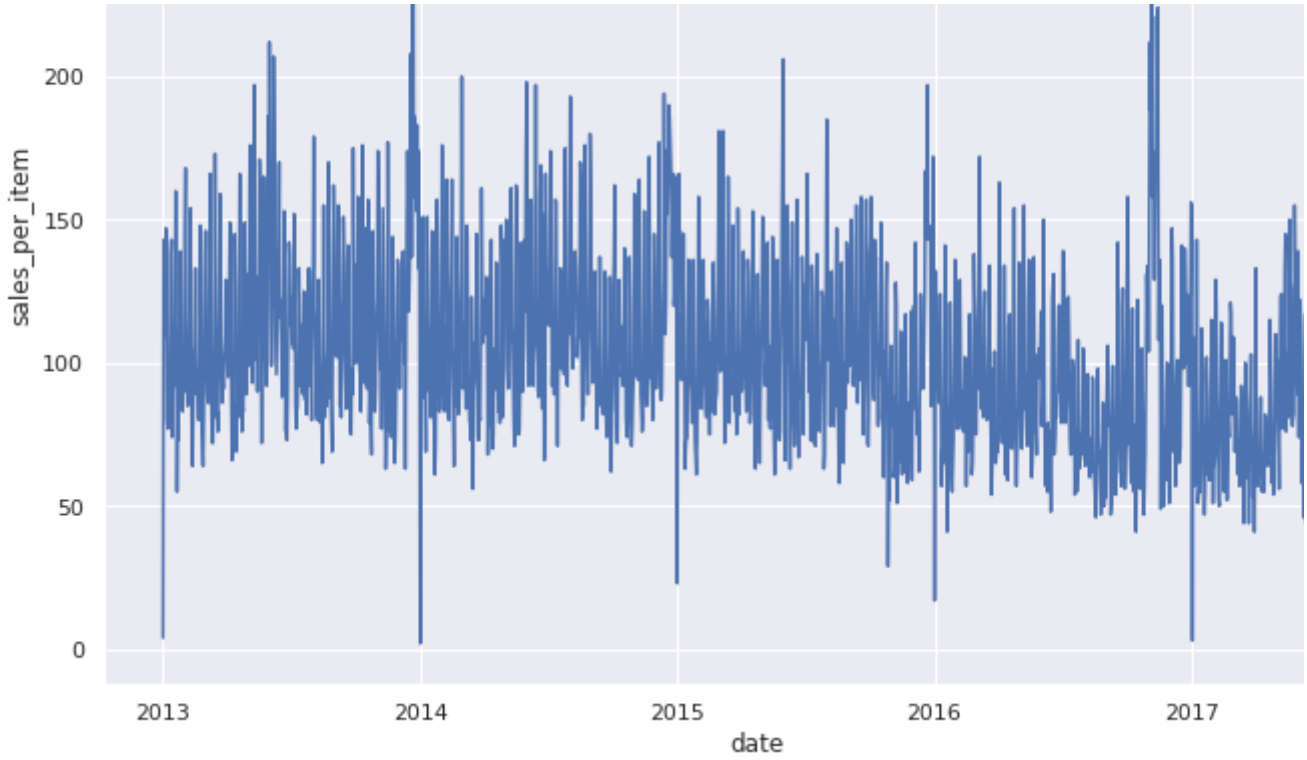
For Item 114790

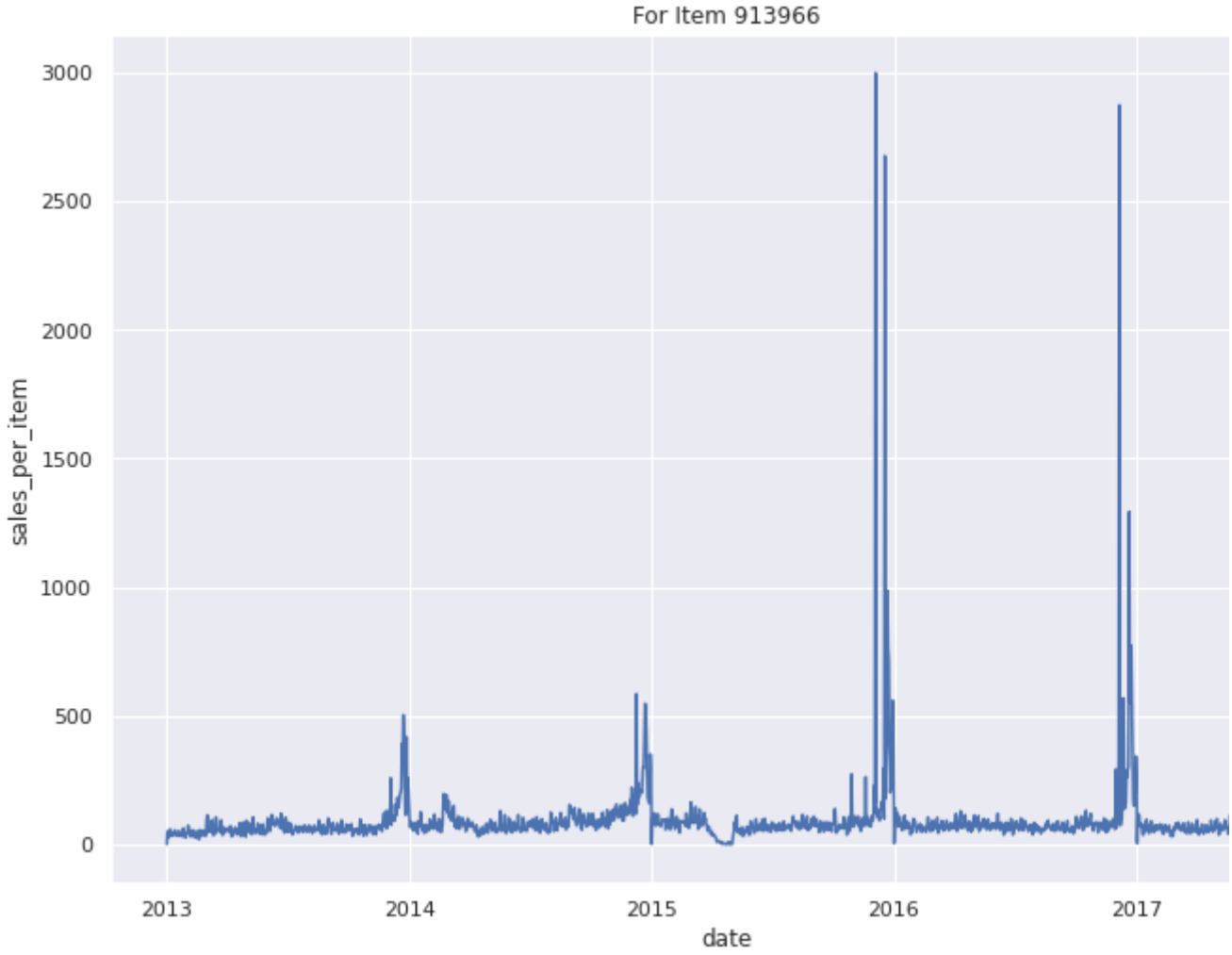


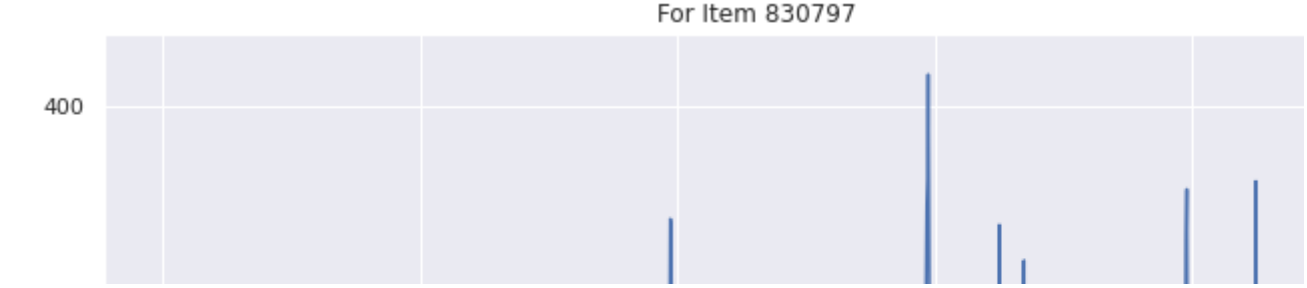

sales_per_item
2500
2000
1500
1000
500
0
2013
2014
2015
2016
2017
date

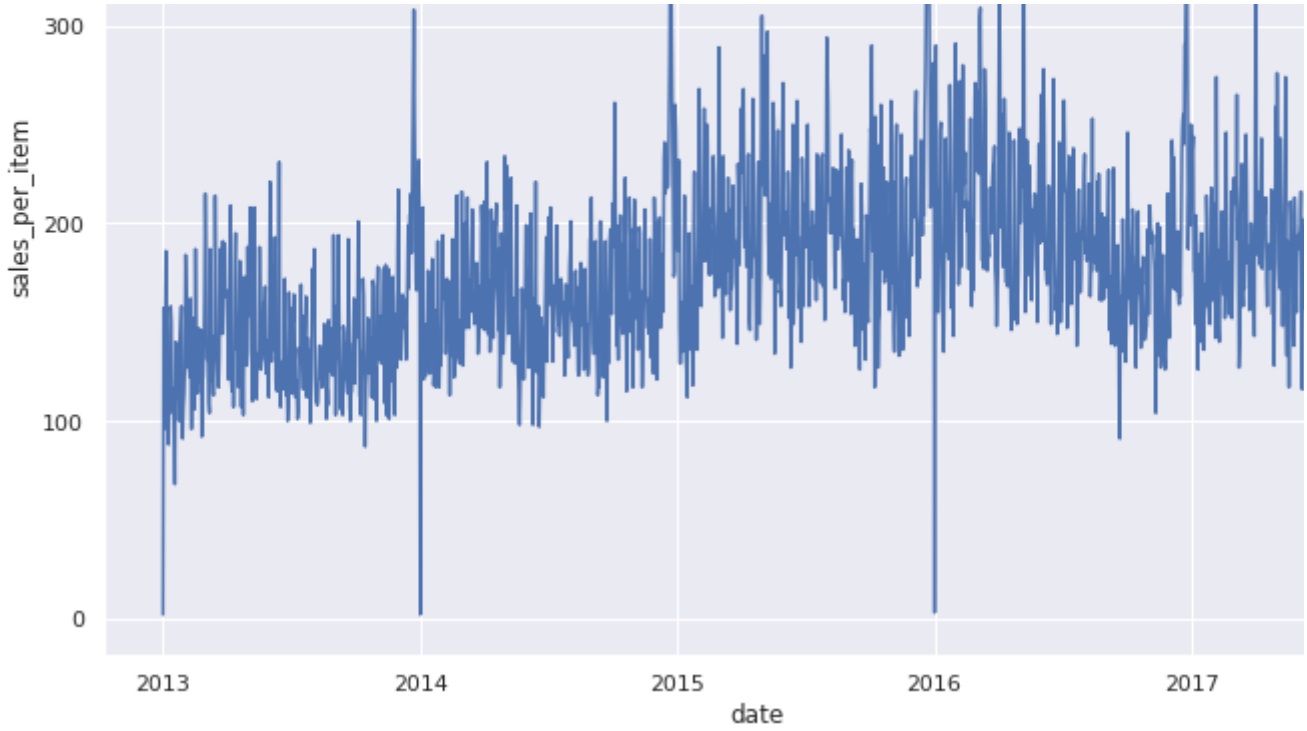

For Item 580929
sales_per_item
500
400
300
200
100
0
2013
2014
2015
2016
2017
date


For Item 671079
300
250

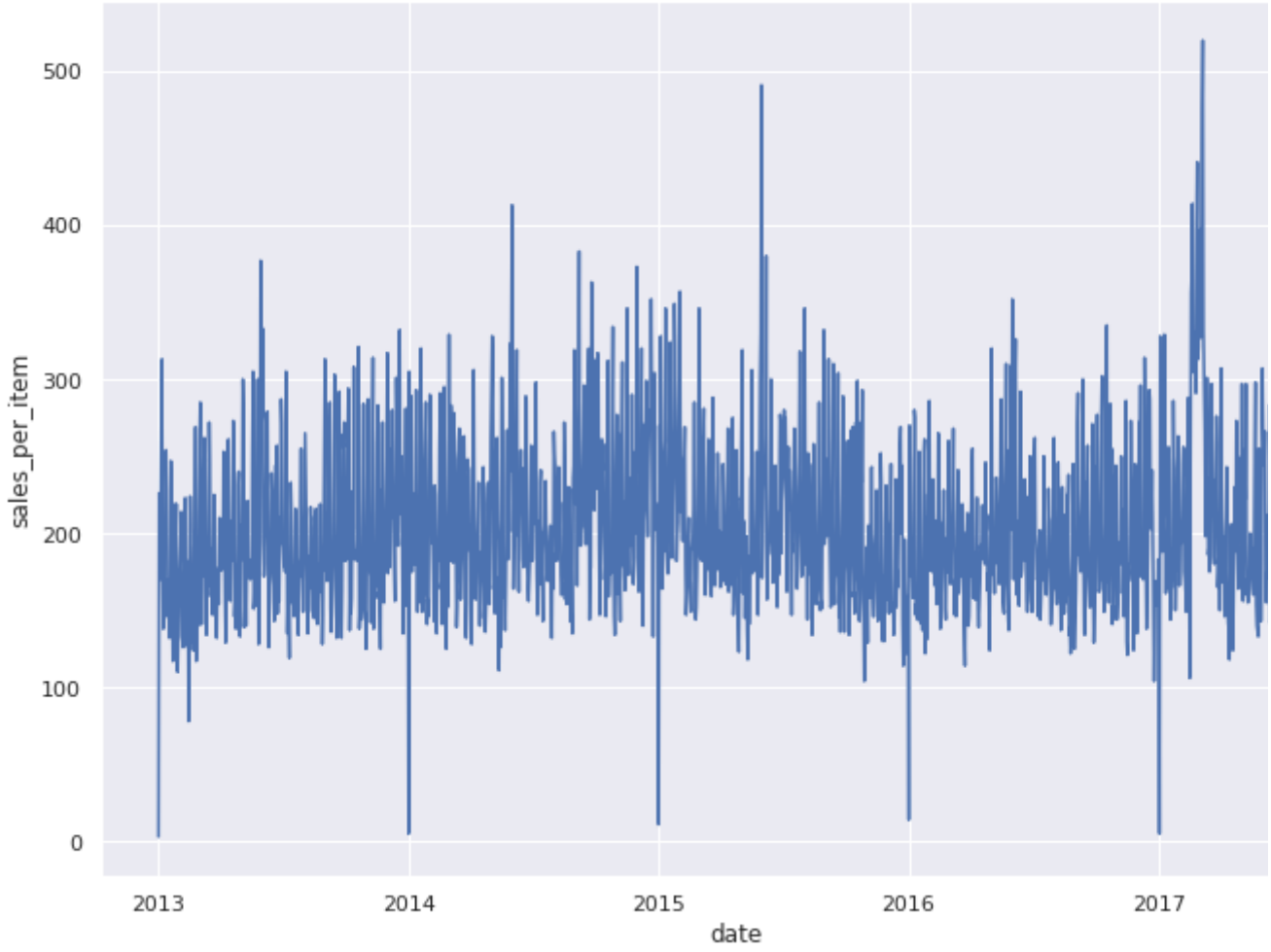

sales_per_item
200
150
100
50
0
2013
2014
2015
2016
2017
date


For Item 913966
sales_per_item
3000
2500
2000
1500
1000
500
0
2013
2014
2015
2016
2017
date

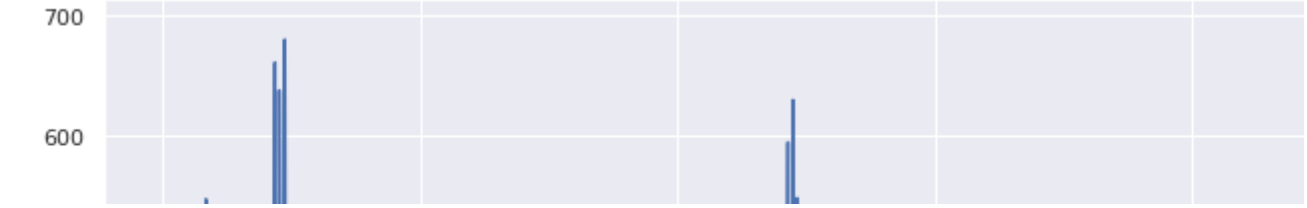

For Item 830797
400



For Item 153239



For Item 258268



So, Tried plotting 25 time series which is a very small sample if we talk about the whole populatior were that non of these items showed a trend, also, items did showed some cyclic behaviour and s almost 0 for some part of the year and few items had mixed signal, but most of the items had goo

Let's see the most popular items across the chain

```
date_item_level_count.head()
```

	date	item_nbr	sales_per_item
0	2013-01-01	749421	1
1	2013-01-01	913521	1
2	2013-01-01	587069	1
3	2013-01-01	636846	1
4	2013-01-01	802832	1

```
date_item_level_count.groupby('item_nbr').sum().sort_values('sales_per_item', ascending =
```

item_nbr	sales_per_item
502331	83475
314384	83450

These are the top items with maximum total sales....

265559	83047

Store Data

```
stores_df.printSchema()
```

```
root
 |-- store_nbr: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- type: string (nullable = true)
 |-- cluster: string (nullable = true)
```

850333	81097

```
stores_df.show()
```

```
+---------+-------------+--------------------+----+-------+
|store_nbr|         city|               state|type|cluster|
+---------+-------------+--------------------+----+-------+
|        1|        Quito|           Pichincha|   D|     13|
|        2|        Quito|           Pichincha|   D|     13|
|        3|        Quito|           Pichincha|   D|      8|
|        4|        Quito|           Pichincha|   D|      9|
|        5|Santo Domingo|Santo Domingo de ...|   D|      4|
|        6|        Quito|           Pichincha|   D|     13|
|        7|        Quito|           Pichincha|   D|      8|
|        8|        Quito|           Pichincha|   D|      8|
|        9|        Quito|           Pichincha|   B|      6|
|       10|        Quito|           Pichincha|   C|     15|
|       11|      Cayambe|           Pichincha|   B|      6|
|       12|    Latacunga|            Cotopaxi|   C|     15|
|       13|    Latacunga|            Cotopaxi|   C|     15|
|       14|     Riobamba|          Chimborazo|   C|      7|
|       15|       Ibarra|            Imbabura|   C|     15|
|       16|Santo Domingo|Santo Domingo de ...|   C|      3|
|       17|        Quito|           Pichincha|   C|     12|
|       18|        Quito|           Pichincha|   B|     16|
|       19|     Guaranda|             Bolivar|   C|     15|
|       20|        Quito|           Pichincha|   B|      6|
+---------+-------------+--------------------+----+-------+
only showing top 20 rows
```

```
stores_df.dtypes
```

```
     [('store_nbr', 'string'),
      ('city', 'string'),
      ('state', 'string'),
      ('type', 'string'),
```

```
stores_df = stores_df.withColumn('store_nbr', col('store_nbr').cast(IntegerType())).withCo
```

```
stores_df.dtypes
```

```
[('store_nbr', 'int'),
 ('city', 'string'),
 ('state', 'string'),
 ('type', 'string'),
 ('cluster', 'int')]
```

```
stores_df.createOrReplaceTempView('store_data')
stores_df.cache
```

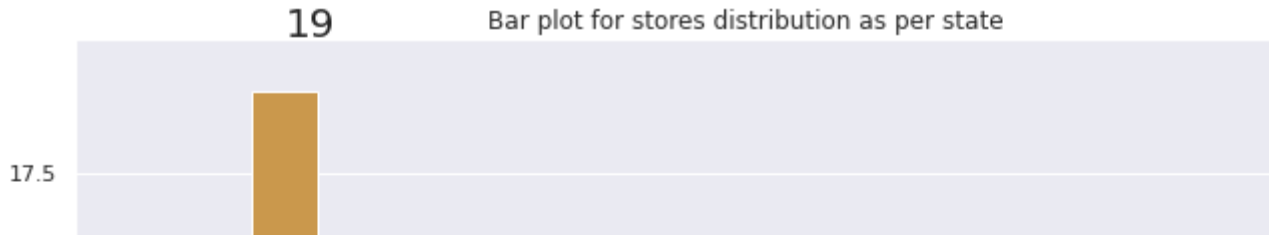
```
<bound method DataFrame.cache of DataFrame[store_nbr: int, city: string, state: strin
```

```
store_type = spark.sql('select type, count(1) count FROM store_data GROUP BY type').toPand
store_city = spark.sql('select city, count(1) count FROM store_data GROUP BY city').toPand
store_state = spark.sql('select state, count(1) count FROM store_data GROUP BY state').toP
store_cluster = spark.sql('select cluster, count(1) count FROM store_data GROUP BY cluster
```

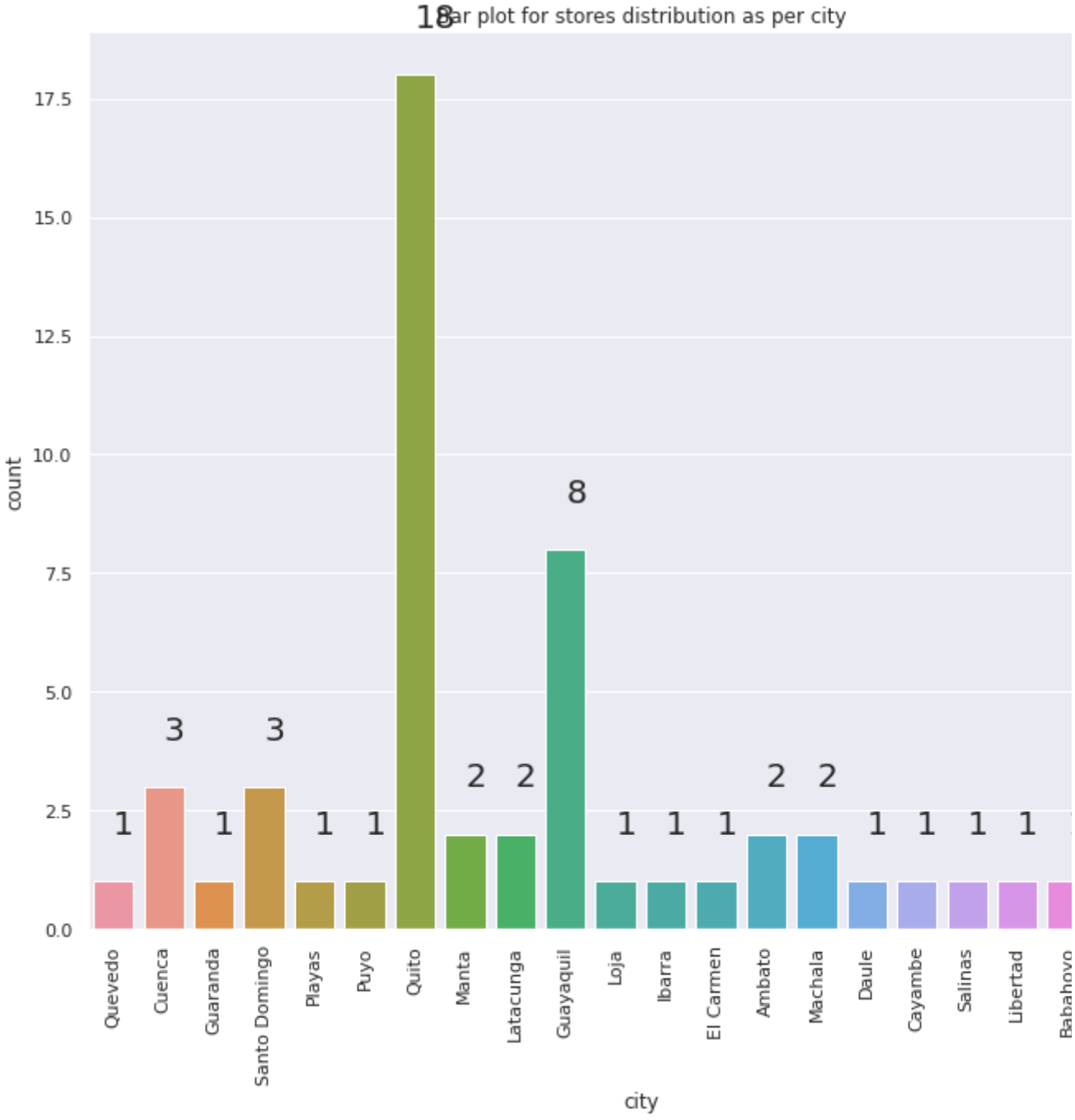
```
def bar_plot(x, y, txt):
  sns.set(rc={'figure.figsize':(12,10)})
  sns.barplot(x, y)
  #Ref: https://stackoverflow.com/questions/30228069/how-to-display-the-value-of-the-bar-o
  for index,data in enumerate(y):
    plt.text(x=index , y =data+1 , s=f"{data}" , fontdict=dict(fontsize=20))
  plt.title(txt)
  plt.xticks(rotation=90)
  plt.show()
```

```
bar_plot(store_type['type'], store_type['count'], 'Bar plot for stores distribution as per
```

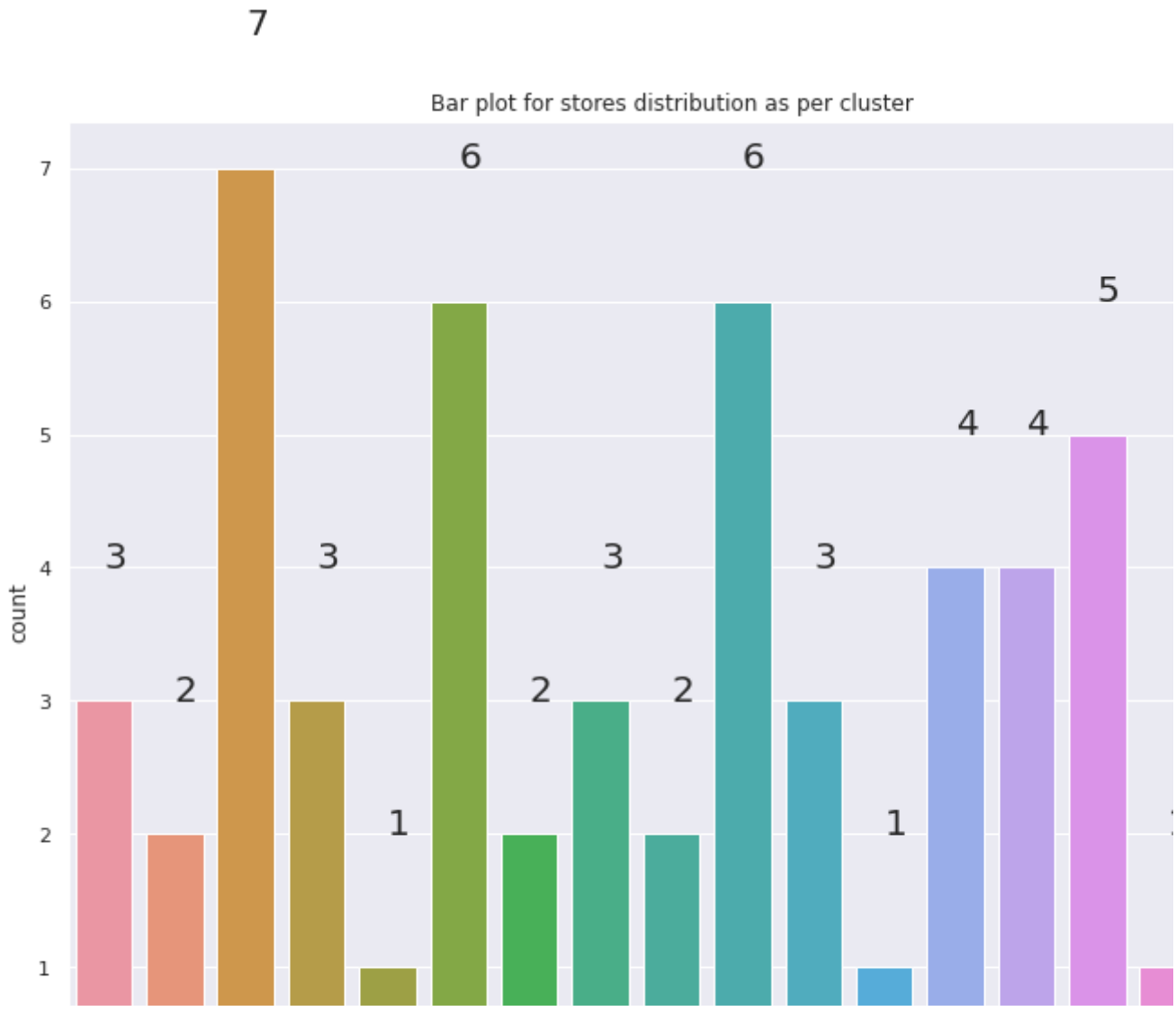

```
bar_plot(store_state['state'], store_state['count'], 'Bar plot for stores distribution as
```




```
bar_plot(store_city['city'], store_city['count'], 'Bar plot for stores distribution as per
```




```
bar_plot(store_cluster['cluster'], store_cluster['count'], 'Bar plot for stores distributi
```



From all of the plots for stores we can say that:

1) Among the types, we see that D and C are the most frequent, with A and B having similar mediu stores.

2) The cities fall into four groups, with most of them having only a single store. Six cities have 2 or are each in a group of their own with 8 and 18 stores, respectively.

3) The city distribution is reflected in the state distribution as well, with “Pichincha” having 19 store

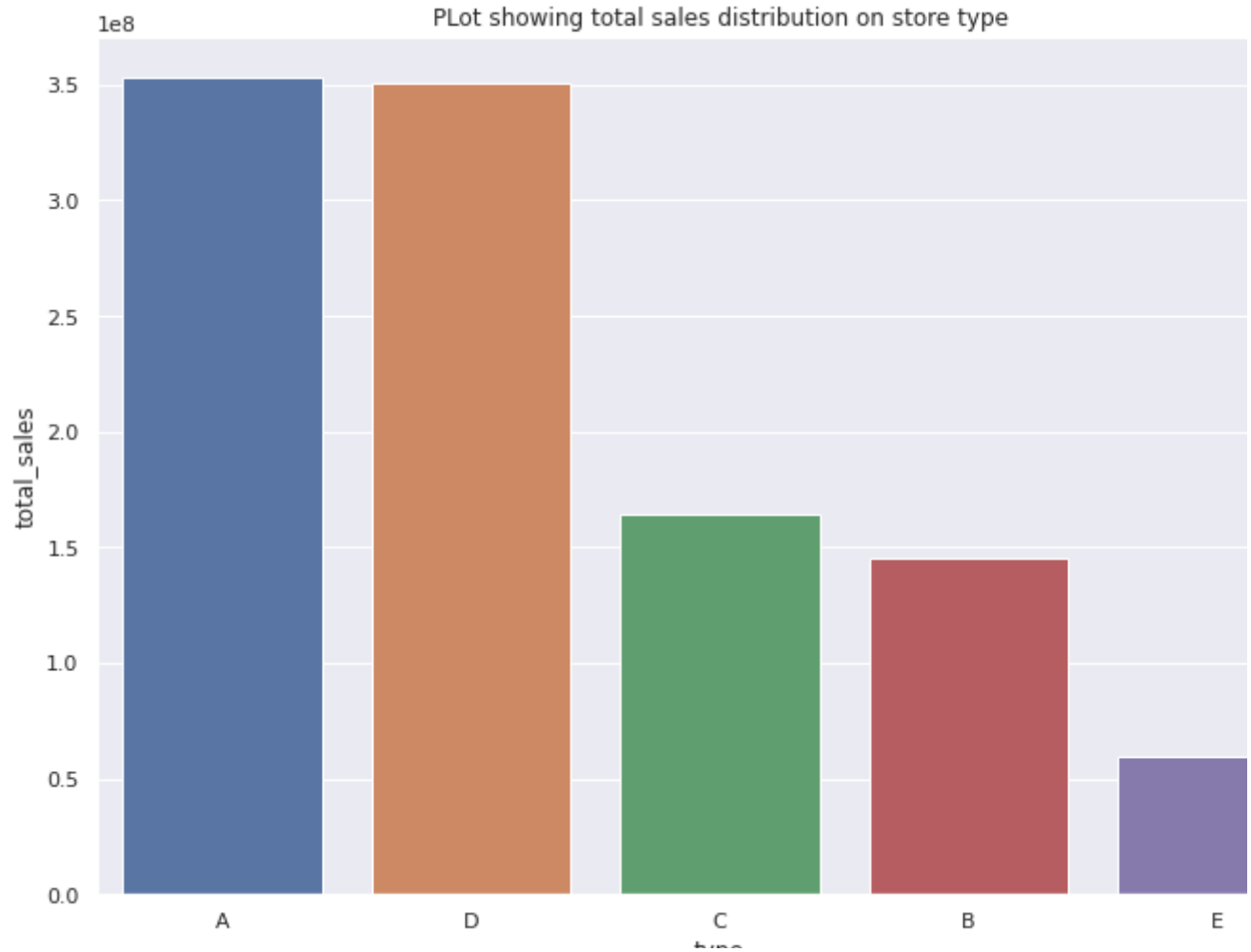
4) The cluster feature shows a range from 1 to 7.

Total sales by type

```
type_sales = spark.sql('''SELECT s.type, SUM(d.unit_sales) AS total_sales FROM data d LEFT
                          GROUP BY s.type ORDER BY total_sales DESC''').toPandas()


sns.barplot(type_sales['type'], type_sales['total_sales'])
plt.title('PLot showing total sales distribution on store type')
```

```
Text(0.5, 1.0, 'PLot showing total sales distribution on store type')
```



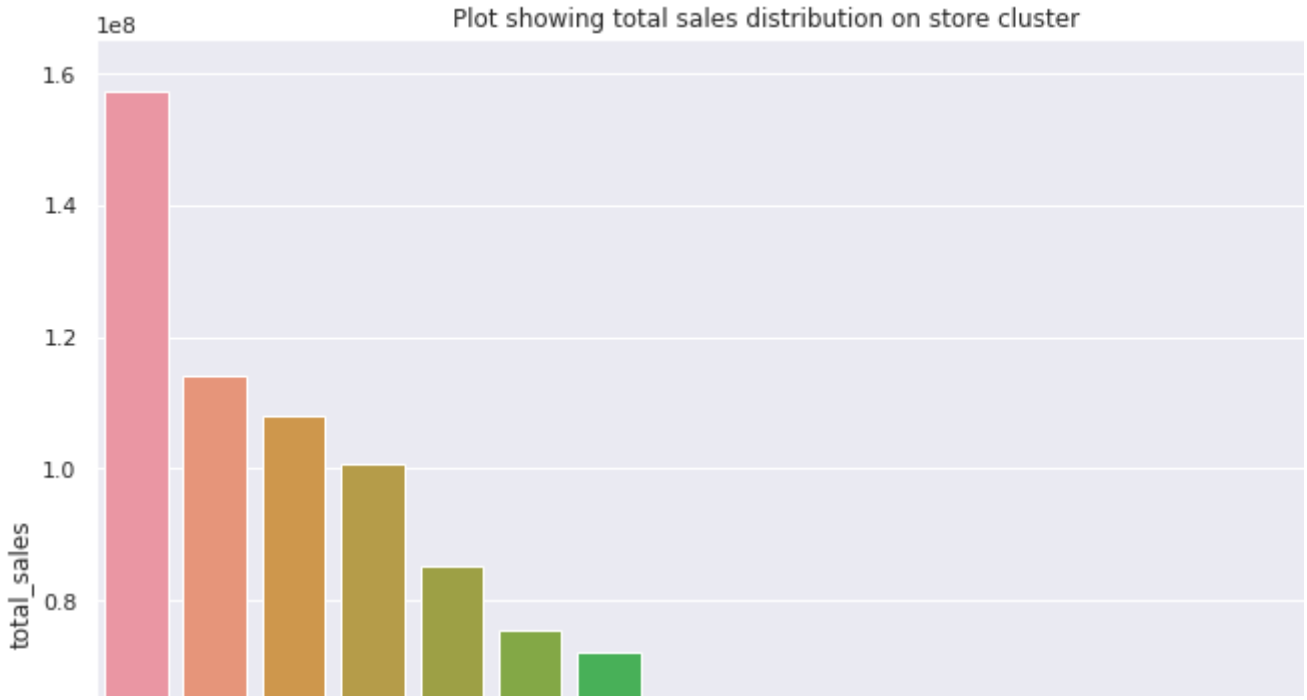
Again type A and D corresponds to the maximum number of sales.

Total sales by cluster

```
cluster_sales = spark.sql('select s.cluster, SUM(d.unit_sales) as total_sales from data d

sns.barplot(cluster_sales['cluster'], cluster_sales['total_sales'], order=cluster_sales.so
plt.title('Plot showing total sales distribution on store cluster')
```

```
Text(0.5, 1.0, 'Plot showing total sales distribution on store cluster')
```



Total Sales by State

```
state_sales = spark.sql('select s.state, SUM(d.unit_sales) as total_sales from data d LEFT
```

```
sns.barplot(state_sales['state'], state_sales['total_sales'])
plt.title('Plot showing total sales distribution on store state location')
plt.xticks(rotation=90)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 <a list of 16 Text major ticklabel objects>)
```

Pichincha corresponds to maimum number of sales

Total sales by city

```
city_sales = spark.sql('select s.city, SUM(d.unit_sales) as total_sales from data d LEFT J
```

```
sns.barplot(city_sales['city'], city_sales['total_sales'])
plt.title('Plot showing total sales distribution on store cities location')
plt.xticks(rotation=90)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21]), <a list of 22 Text major ticklabel objects>)
```

Most of the sales are seen in Quito and then in Guayaquil....

Item Data

```
items_df.printSchema()
```

```
root
 |-- item_nbr: string (nullable = true)
 |-- family: string (nullable = true)
 |-- class: string (nullable = true)
 |-- perishable: string (nullable = true)
```

```
items_df.show()
```

```
+--------+------------+-----+----------+
|item_nbr|      family|class|perishable|
+--------+------------+-----+----------+
|   96995|   GROCERY I| 1093|         0|
|   99197|   GROCERY I| 1067|         0|
|  103501|    CLEANING| 3008|         0|
|  103520|   GROCERY I| 1028|         0|
```

```
items_df.dtypes
```

```
[('item_nbr', 'string'),
 ('family', 'string'),
 ('class', 'string'),
 ('perishable', 'string')]
```

```
|  105857|   GROCERY I| 1092|         0|
```

```
items_df = items_df.withColumn('item_nbr', col('item_nbr').cast(IntegerType())).withColumn
```

```
|  108634|   GROCERY I| 1075|         0|
```

```
items_df.dtypes
```

```
[('item_nbr', 'int'),
 ('family', 'string'),
 ('class', 'int'),
 ('perishable', 'int')]
```

```
items_df.createOrReplaceTempView('items_data')
items_df.cache
```

```
<bound method DataFrame.cache of DataFrame[item_nbr: int, family: string, class: int,
```

```
spark.sql('select COUNT(DISTINCT item_nbr) as total_items from items_data').show()
```

```
+-----------+
|total_items|
+-----------+
|       4100|
+-----------+
```

Total items in the file is 4100, in train file we saw total items to be 4036, 64 may be the items in the train.

```
spark.sql('select COUNT(DISTINCT family) as all_family from items_data').show()
```

```
+----------+
|all_family|
+----------+
|        33|
+----------+
```

We have total 33 family of items

```
spark.sql('select COUNT(DISTINCT class) as all_class from items_data').show()
```

```
+---------+
|all_class|
+---------+
|      337|
+---------+
```

Total class of the items is 337

```
spark.sql('select perishable, COUNT(1) as count from items_data GROUP BY perishable').show
```

```
+----------+-----+
|perishable|count|
+----------+-----+
|         1|  986|
|         0| 3114|
+----------+-----+
```

```
spark.sql('SELECT perishable, count, ROUND(count/SUM(count) OVER (), 2) as percent FROM (s
```

```
+----------+-----+-------+
|perishable|count|percent|
+----------+-----+-------+
|         1|  986|   0.24|
|         0| 3114|   0.76|
+----------+-----+-------+
```

24 % of the items stored in the store are perishable....

Let's look at items by Item Family

```
spark.sql('select family, count(Distinct item_nbr) as item_count from items_data GROUP BY
```

```
+--------------------+----------+
|              family|item_count|
+--------------------+----------+
|           GROCERY I|      1334|
|           BEVERAGES|       613|
|            CLEANING|       446|
|             PRODUCE|       306|
|              DAIRY|       242|
|       PERSONAL CARE|       153|
```

Let's plot this

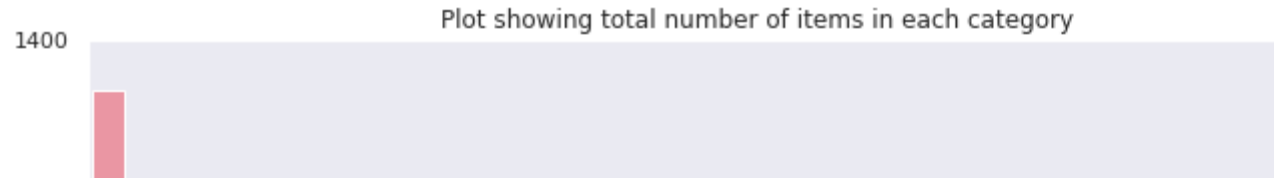
```
|              DELI|        91|
```

```
itm_family = spark.sql('select family, count(Distinct item_nbr) as item_count from items_d
```

```
|   LIQUOR,WINE,BEER|        75|
```

```
sns.barplot(itm_family['family'], itm_family['item_count'])
plt.title('Plot showing total number of items in each category')
plt.xticks(rotation=90)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]),
 <a list of 33 Text major ticklabel objects>)
```



As expected Grocery, Beverages, cleaning, prodice, dairy contains most of the items....

Let see how many class of items have not more than 2 items

```
item_class_count = spark.sql('select class, count(Distinct item_nbr) as item_count from it
```

```
item_class_count[item_class_count['item_count'] <=  2]['class'].nunique()
```

```
118
```

In total we have 337 classes, out of these 118 Classes have items less than or equal to 2.

Checking relationship between classes and family

```
class_family_item_group = spark.sql('select class, family, count(DISTINCT item_nbr) as ite
```

```
class_family_item_group[class_family_item_group['family'] == 'GROCERY I']
```

	class	family	item_count
1	1079	GROCERY I	1
5	1058	GROCERY I	16
11	1054	GROCERY I	2
13	1076	GROCERY I	23
16	1024	GROCERY I	4
...	...	...	...
302	1045	GROCERY I	24
306	1096	GROCERY I	18
309	1029	GROCERY I	1
319	1067	GROCERY I	2
328	1027	GROCERY I	2

67 rows × 3 columns

Have checked for various families, and now it is known that classes are the sub group of families, with other data present will tell us of the predictive importance of this data.

Transaction Data

```
transactions_df.printSchema()
```

```
root
 |-- date: string (nullable = true)
 |-- store_nbr: string (nullable = true)
 |-- transactions: string (nullable = true)
```

```
transactions_df.show()
```

```
+----------+---------+------------+
|      date|store_nbr|transactions|
+----------+---------+------------+
|2013-01-01|       25|         770|
|2013-01-02|        1|        2111|
|2013-01-02|        2|        2358|
|2013-01-02|        3|        3487|
|2013-01-02|        4|        1922|
|2013-01-02|        5|        1903|
|2013-01-02|        6|        2143|
|2013-01-02|        7|        1874|
|2013-01-02|        8|        3250|
|2013-01-02|        9|        2940|
|2013-01-02|       10|        1293|
|2013-01-02|       11|        3547|
|2013-01-02|       12|        1362|
|2013-01-02|       13|        1102|
|2013-01-02|       14|        2002|
|2013-01-02|       15|        1622|
|2013-01-02|       16|        1167|
|2013-01-02|       17|        1580|
|2013-01-02|       18|        1635|
|2013-01-02|       19|        1369|
+----------+---------+------------+
only showing top 20 rows
```

```
transactions_df.count()
```

```
83488
```

```
transactions_df.dtypes
```

```
[('date', 'string'), ('store_nbr', 'string'), ('transactions', 'string')]
```

```
transactions_df = transactions_df.withColumn('date', col('date').cast(DateType())).withCol
```

```
transactions_df.dtypes
```

```
[('date', 'date'), ('store_nbr', 'int'), ('transactions', 'int')]
```

```
transactions_df.createOrReplaceTempView('transactions_data')
transactions_df.cache
```

```
<bound method DataFrame.cache of DataFrame[date: date, store_nbr: int, transactions:
```

Let us start by plotting transactions signals to see how they look

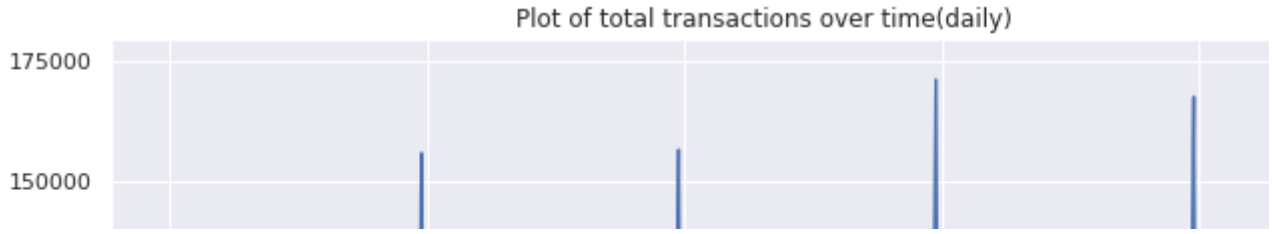
```
transaction_date_agg = spark.sql('select date, SUM(transactions) as total_transaction FROM
```

```
transaction_date_agg.head()
```

	date	total_transaction
0	2013-01-01	770
1	2013-01-02	93215
2	2013-01-03	78504
3	2013-01-04	78494
4	2013-01-05	93573

```
sns.lineplot(x=transaction_date_agg['date'], y=transaction_date_agg['total_transaction'])
plt.title('Plot of total transactions over time(daily)')
```

```
Text(0.5, 1.0, 'Plot of total transactions over time(daily)')
```



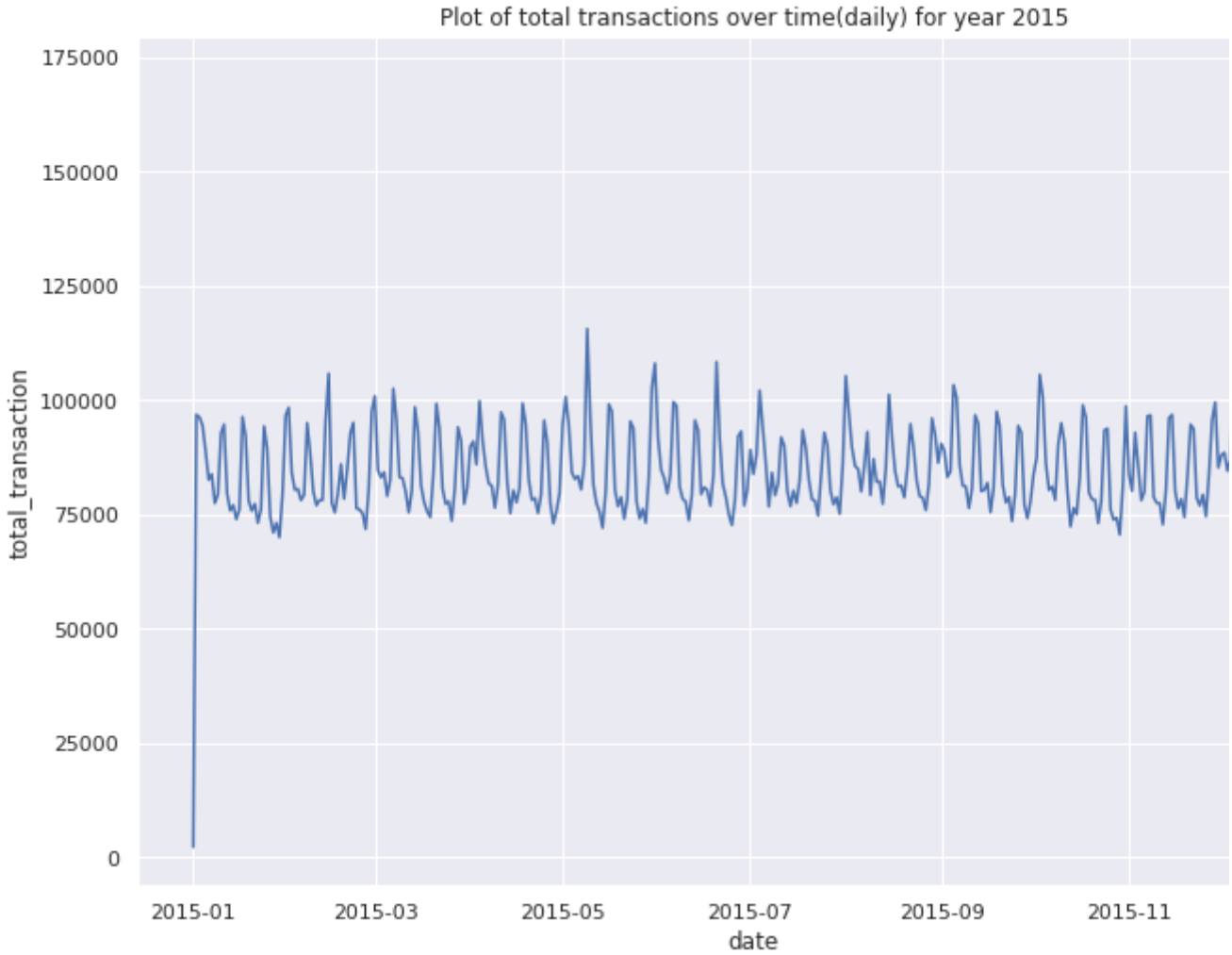
We find that there is a strong spike before Christmas, with corresponding drops when the stores a
Overall, the sales appear to be stable throughout this time range.

Plotting for year 2015

```
import datetime
start_index = transaction_date_agg[transaction_date_agg['date'] == datetime.date(2015, 1,
end_index = transaction_date_agg[transaction_date_agg['date'] == datetime.date(2015, 12, 3

year_2015_df = transaction_date_agg.loc[start_index : end_index]
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.lineplot(x=year_2015_df['date'], y=year_2015_df['total_transaction'])
plt.title('Plot of total transactions over time(daily) for year 2015')
```

```
Text(0.5, 1.0, 'Plot of total transactions over time(daily) for year 2015')
```



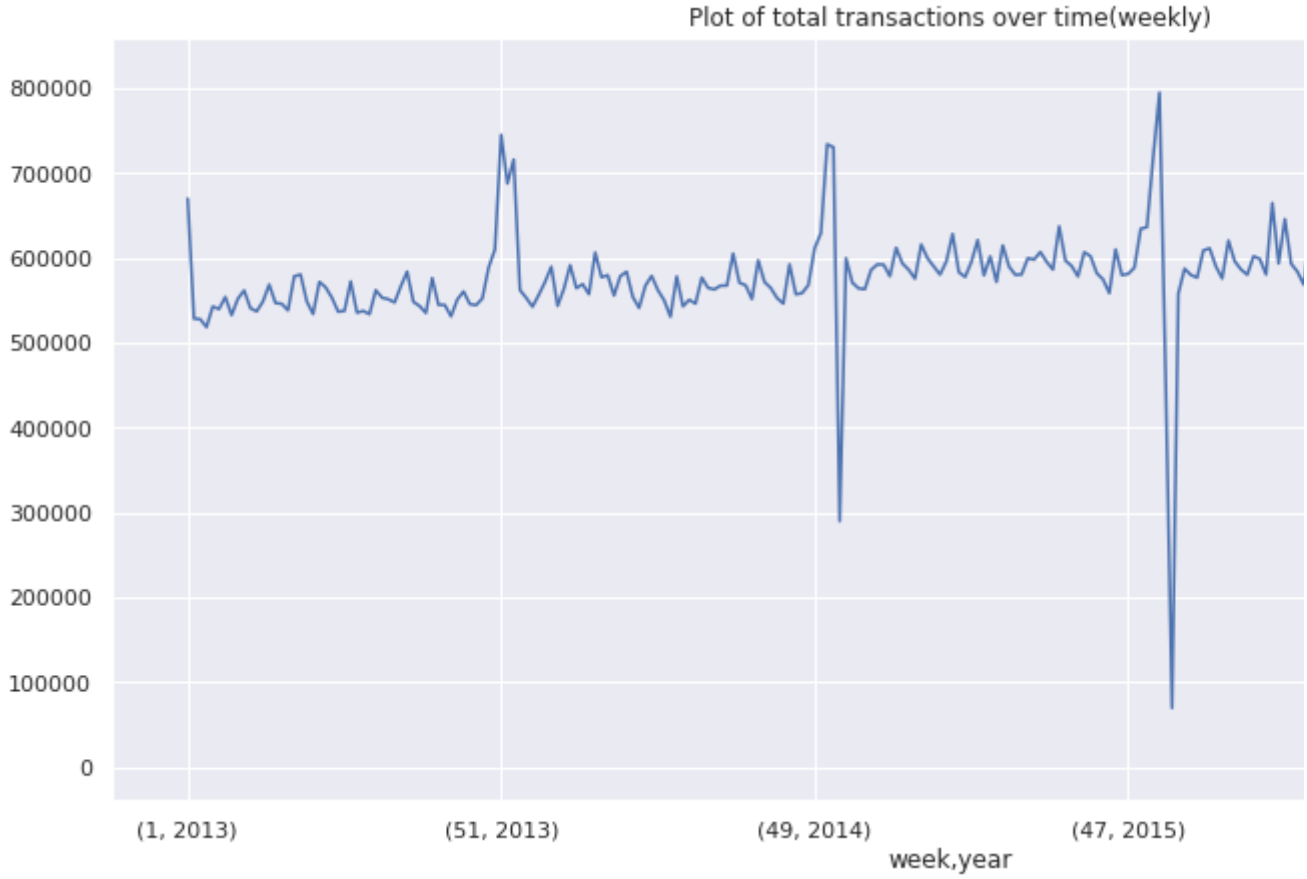
When we tried plotting for year 2015, we saw the saw observation as see when plotted yearly...

Let us aggragate date weekly and monthly and see how our plot looks like...

```
transaction_date_agg['week'] = transaction_date_agg['date'].apply(lambda x : x.isocalendar
transaction_date_agg['month'] = transaction_date_agg['date'].apply(lambda x : x.month)
transaction_date_agg['year'] = transaction_date_agg['date'].apply(lambda x : x.year)
```

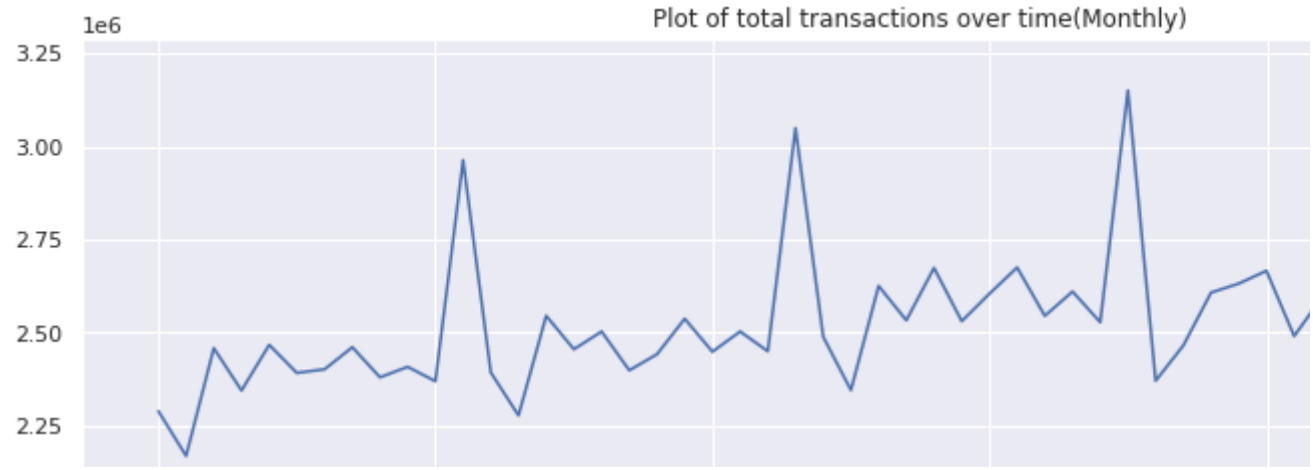
```
fig, ax = plt.subplots(figsize=(15,7))
transaction_date_agg.groupby(['week','year'], sort = False).sum()['total_transaction'].plo
plt.title('Plot of total transactions over time(weekly)')
```

```
Text(0.5, 1.0, 'Plot of total transactions over time(weekly)')
```




```
fig, ax = plt.subplots(figsize=(15,7))
transaction_date_agg.groupby(['month','year'], sort = False).sum()['total_transaction'].pl
plt.title('Plot of total transactions over time(Monthly)')
```

```
Text(0.5, 1.0, 'Plot of total transactions over time(Monthly)')
```



These plots doesn't make much sense to us....

Let us check the number of items purchased per transaction over the period

```
spark.sql('''SELECT transactions_data.date,  total_sales/total_transaction as items_per_tr
FROM
(select date, SUM(transactions) as total_transaction FROM transactions_data GROUP BY date
INNER JOIN
(select date, SUM(unit_sales) as total_sales FROM data WHERE unit_sales > 0 GROUP BY date
ON transactions_data.date=data.date''').show()
```

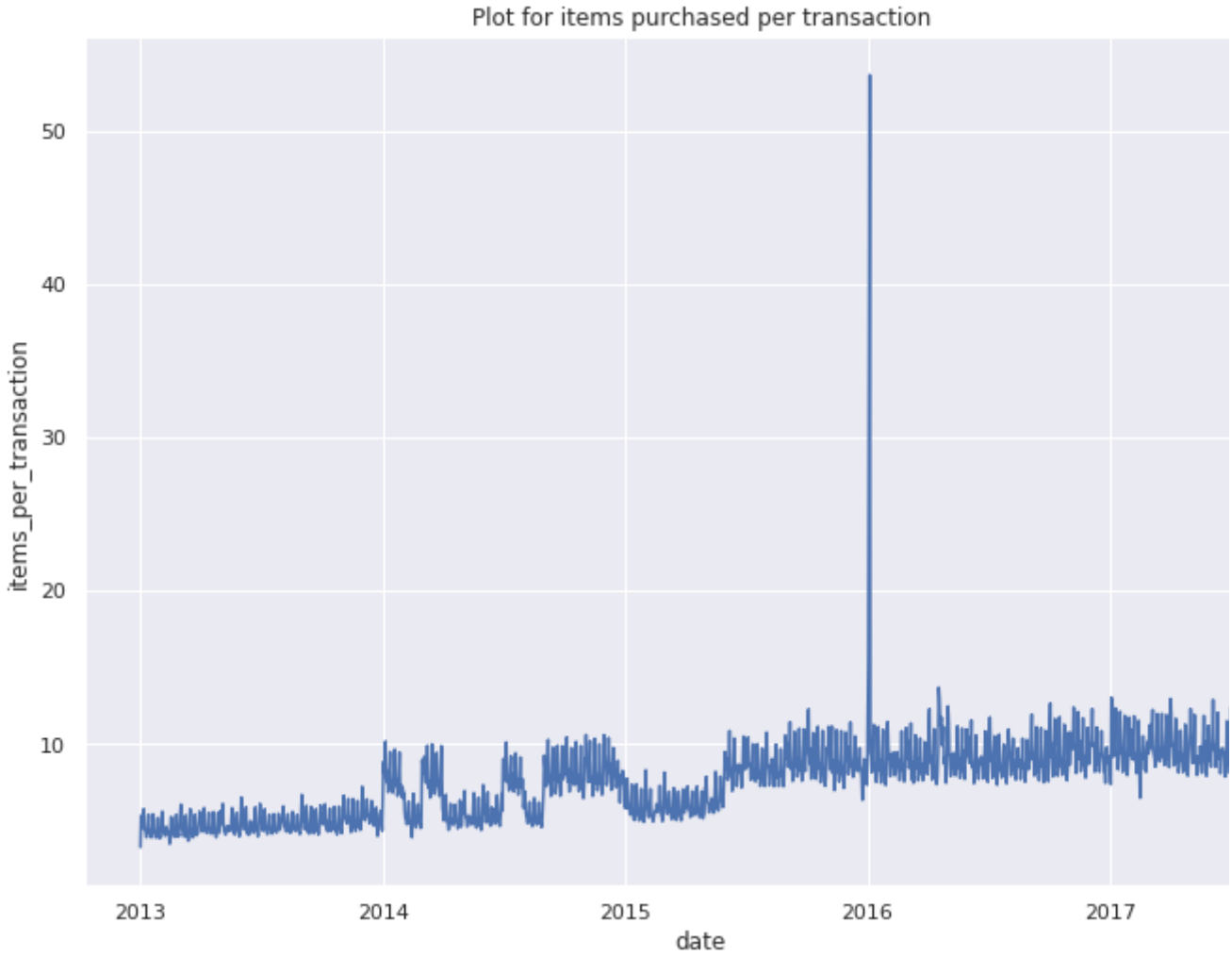
```
+----------+---------------------+
|      date|items_per_transaction|
+----------+---------------------+
|2013-01-01|   3.2618428574754046|
|2013-01-02|    5.322055656274635|
|2013-01-03|    4.604699264327097|
|2013-01-04|    4.515914503734317|
|2013-01-05|    5.101440811699174|
|2013-01-06|   5.7447758334714205|
|2013-01-07|    4.446258463242334|
|2013-01-08|    4.401656107473159|
|2013-01-09|    4.204051757270201|
|2013-01-10|   3.9020080897907308|
|2013-01-11|    4.122575066501422|
|2013-01-12|    4.715922068776316|
|2013-01-13|   5.3836181497208795|
|2013-01-14|   4.1905642571400055|
|2013-01-15|    4.205858662957437|
|2013-01-16|    4.392895157657377|
|2013-01-17|   3.8839387852835032|
|2013-01-18|    4.149216559139545|
|2013-01-19|    4.883430655618284|
|2013-01-20|    5.406747682204343|
+----------+---------------------+
only showing top 20 rows
```

```
items_per_transaction_df = spark.sql('''SELECT transactions_data.date,  total_sales/total_
FROM
(select date, SUM(transactions) as total_transaction FROM transactions_data GROUP BY date
```

```
INNER JOIN
(select date, SUM(unit_sales) as total_sales FROM data WHERE unit_sales > 0 GROUP BY date
ON transactions_data.date=data.date''').toPandas()


sns.lineplot(x=items_per_transaction_df['date'], y=items_per_transaction_df['items_per_tra
plt.title('Plot for items purchased per transaction')
```

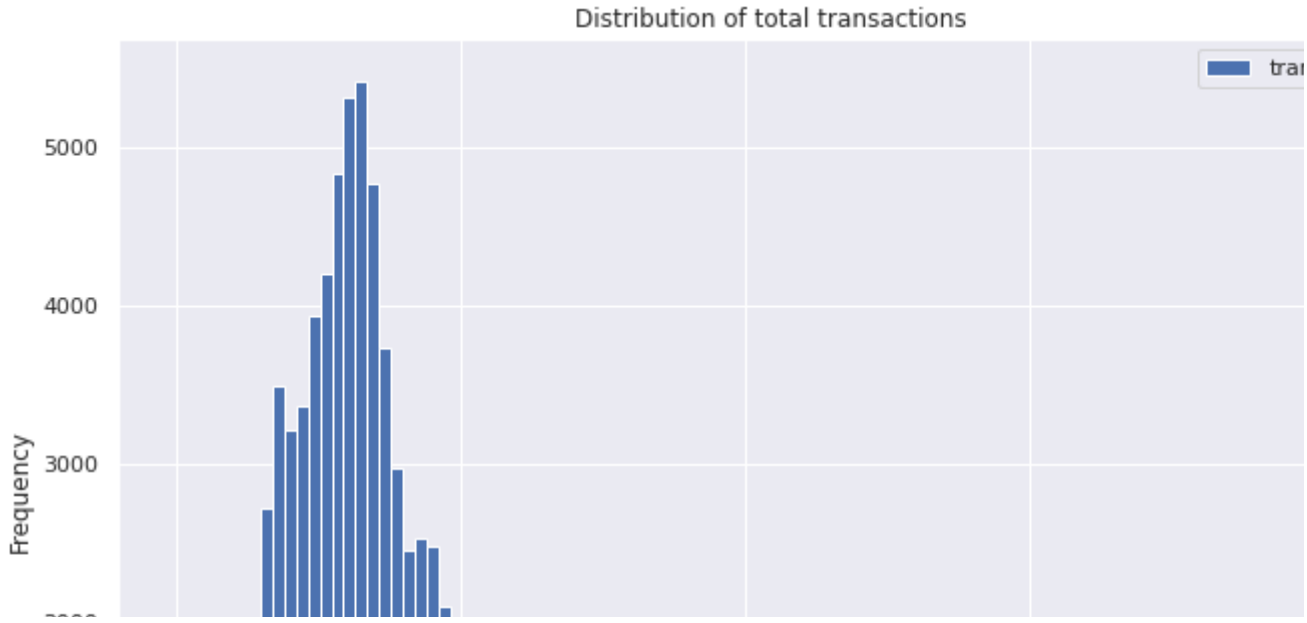
```
Text(0.5, 1.0, 'Plot for items purchased per transaction')
```



Number of items purchased per transaction shows a small upward trend, which can be due to the time, also, we see a big spike that could be an outlier point, but overall seems to be a good signal..

```
spark.sql('select transactions from transactions_data').toPandas().plot.hist(bins = 100)
plt.title('Distribution of total transactions')
```

```
Text(0.5, 1.0, 'Distribution of total transactions')
```



This is a very informative plot as it tells us that the total number of transactions is approximately F transactions for particular items are also Poisson distributed.

Oil Data

```
oil_df.printSchema()
```

```
root
 |-- date: string (nullable = true)
 |-- dcoilwtico: string (nullable = true)
```

```
oil_df.show()
```

```
+----------+----------+
|      date|dcoilwtico|
+----------+----------+
|2013-01-01|      null|
|2013-01-02|     93.14|
|2013-01-03|     92.97|
```

```
oil_df.dtypes
```

```
[('date', 'string'), ('dcoilwtico', 'string')]
```

```
|2013-01-10|     93.81|
```

```
oil_df = oil_df.withColumn('date', col('date').cast(DateType())).withColumn('dcoilwtico',
```

```
|2013-01-15|     93.26|
```

```
oil_df.dtypes
```

```
[('date', 'date'), ('dcoilwtico', 'float')]
```

```
|2013-01-22|     96.09|
```

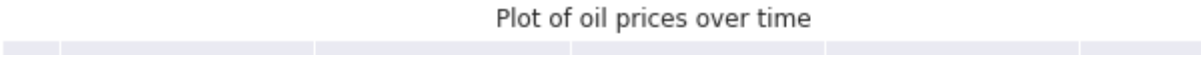
```
oil_df.createOrReplaceTempView('oil_data')
oil_df.cache
```

```
<bound method DataFrame.cache of DataFrame[date: date, dcoilwtico: float]>
only showing top 20 rows
```

```
oil_df = spark.sql('select date, dcoilwtico from oil_data').toPandas()


sns.lineplot(x=oil_df['date'], y=oil_df['dcoilwtico'])
plt.title('Plot of oil prices over time')
```

```
Text(0.5, 1.0, 'Plot of oil prices over time')
```



We see that oil prices suffered a collapse towards the end of 2014 and have not recovered. In fact same level as they were in the beginning of 2015. As a result of this we may see a significant shift the unit sales data, this is not readily apparent. Although sales do appear to drop off in the early pa to add oil price drop doesn't seems to have any impact on the sales, as it was seen from the sales dropping on the sales, so we can say that this feature or data is of no importance to us and will no

Holiday Data

```
holidays_events_df.printSchema()
```

```
root
 |-- date: string (nullable = true)
 |-- type: string (nullable = true)
 |-- locale: string (nullable = true)
 |-- locale_name: string (nullable = true)
 |-- description: string (nullable = true)
 |-- transferred: string (nullable = true)
```

```
holidays_events_df.show()
```

```
+----------+-------+--------+-------------+--------------------+-----------+
|      date|   type|  locale|  locale_name|         description|transferred|
+----------+-------+--------+-------------+--------------------+-----------+
|2012-03-02|Holiday|   Local|        Manta|  Fundacion de Manta|      False|
|2012-04-01|Holiday|Regional|     Cotopaxi|Provincializacion...|      False|
|2012-04-12|Holiday|   Local|       Cuenca| Fundacion de Cuenca|      False|
|2012-04-14|Holiday|   Local|     Libertad|Cantonizacion de ...|      False|
|2012-04-21|Holiday|   Local|     Riobamba|Cantonizacion de ...|      False|
|2012-05-12|Holiday|   Local|         Puyo|Cantonizacion del...|      False|
|2012-06-23|Holiday|   Local|     Guaranda|Cantonizacion de ...|      False|
|2012-06-25|Holiday|Regional|     Imbabura|Provincializacion...|      False|
|2012-06-25|Holiday|   Local|    Latacunga|Cantonizacion de ...|      False|
|2012-06-25|Holiday|   Local|      Machala|Fundacion de Machala|      False|
|2012-07-03|Holiday|   Local|Santo Domingo|Fundacion de Sant...|      False|
|2012-07-03|Holiday|   Local|    El Carmen|Cantonizacion de ...|      False|
|2012-07-23|Holiday|   Local|      Cayambe|Cantonizacion de ...|      False|
|2012-08-05|Holiday|   Local|   Esmeraldas|Fundacion de Esme...|      False|
|2012-08-10|Holiday|National|      Ecuador|Primer Grito de I...|      False|
|2012-08-15|Holiday|   Local|     Riobamba|Fundacion de Riob...|      False|
|2012-08-24|Holiday|   Local|       Ambato| Fundacion de Ambato|      False|
|2012-09-28|Holiday|   Local|       Ibarra| Fundacion de Ibarra|      False|
|2012-10-07|Holiday|   Local|      Quevedo|Cantonizacion de ...|      False|
|2012-10-09|Holiday|National|      Ecuador|Independencia de ...|       True|
+----------+-------+--------+-------------+--------------------+-----------+
only showing top 20 rows
```

```
holidays_events_df= holidays_events_df.withColumn('date', col('date').cast(DateType()))
```

```
holidays_events_df.dtypes
```

```
[('date', 'date'),
 ('type', 'string'),
 ('locale', 'string'),
 ('locale_name', 'string'),
 ('description', 'string'),
 ('transferred', 'string')]
```

```
holidays_events_df.createOrReplaceTempView('holidays_data')
holidays_events_df.cache
```

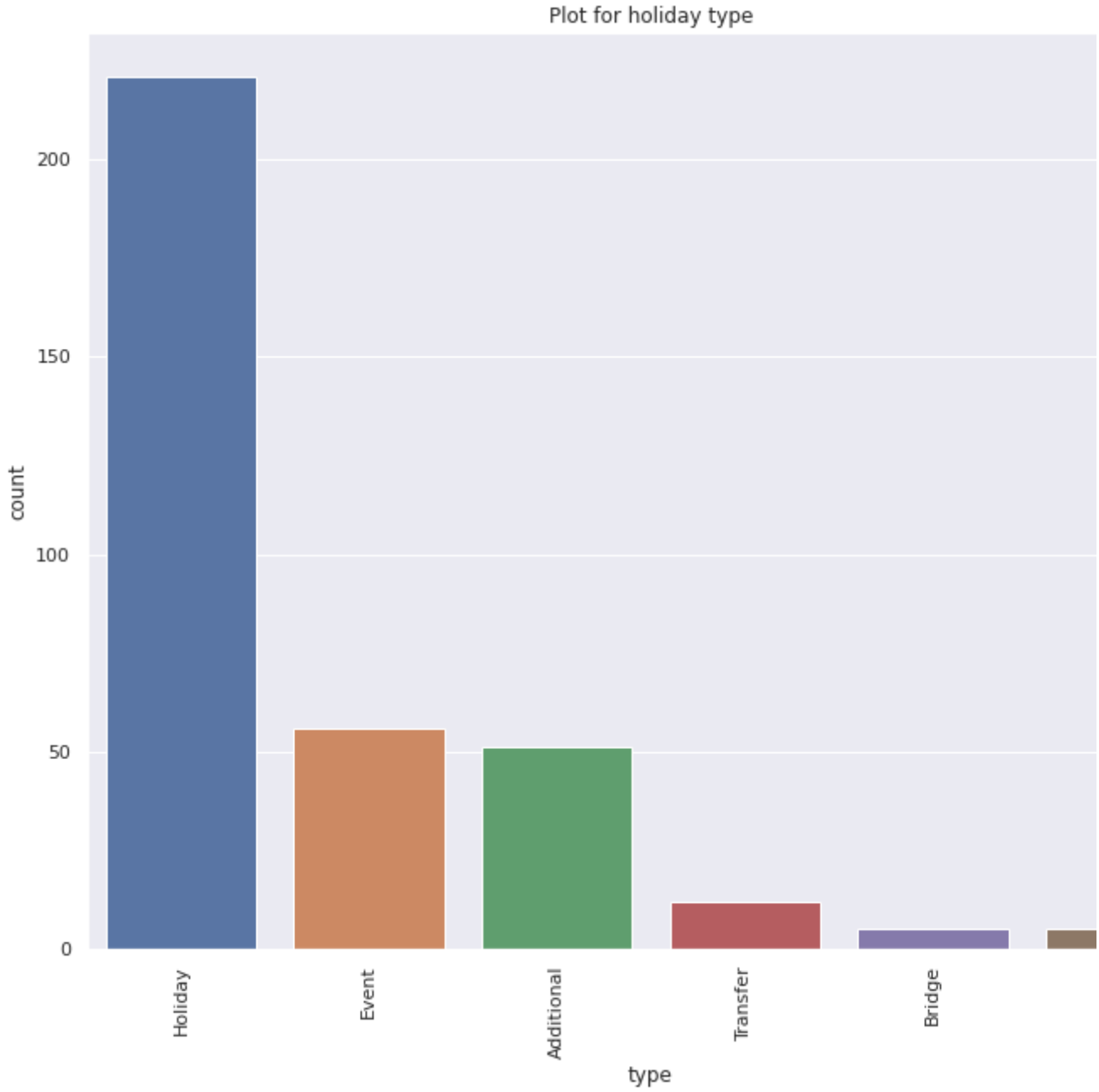
```
<bound method DataFrame.cache of DataFrame[date: date, type: string, locale: string,
```

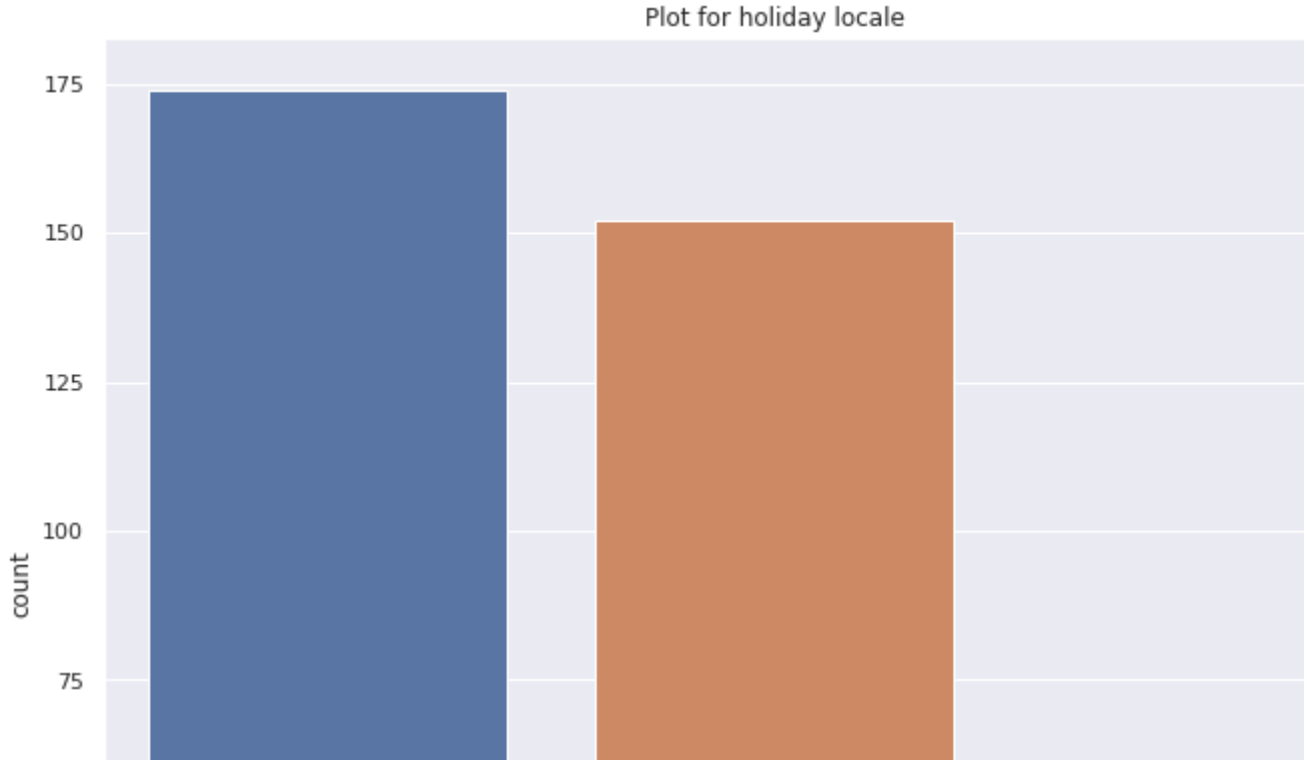
Plotting the count for all of the columns to see if we get any major info out of these

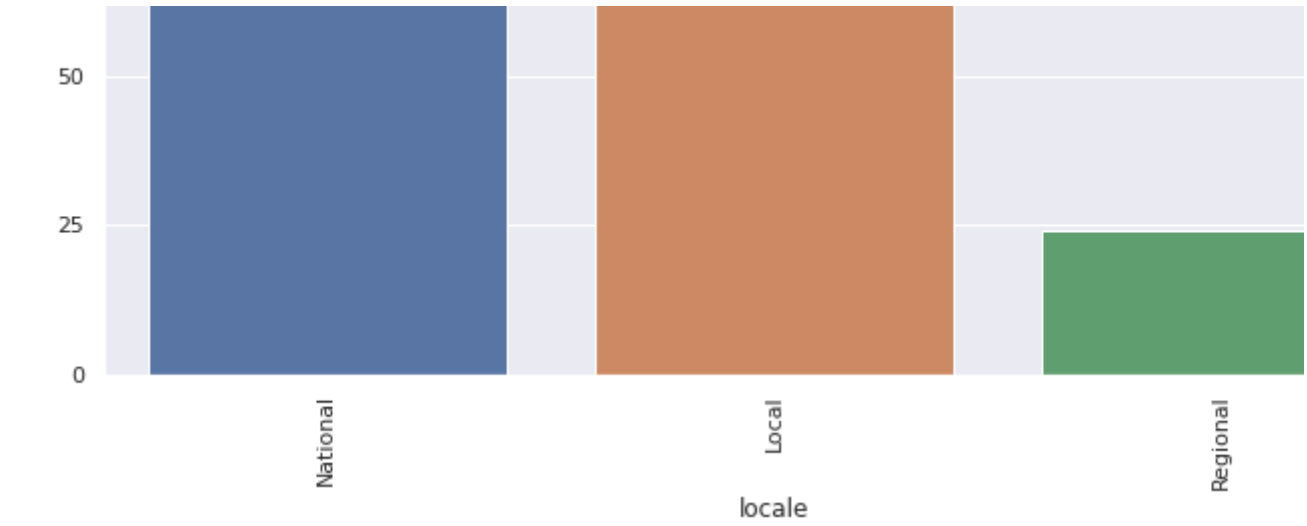
```
def holiday_plot(x, y, txt):
  sns.set(rc={'figure.figsize':(12,10)})
  sns.barplot(x, y)
  plt.title(txt)
  plt.xticks(rotation=90)
  plt.show()
```

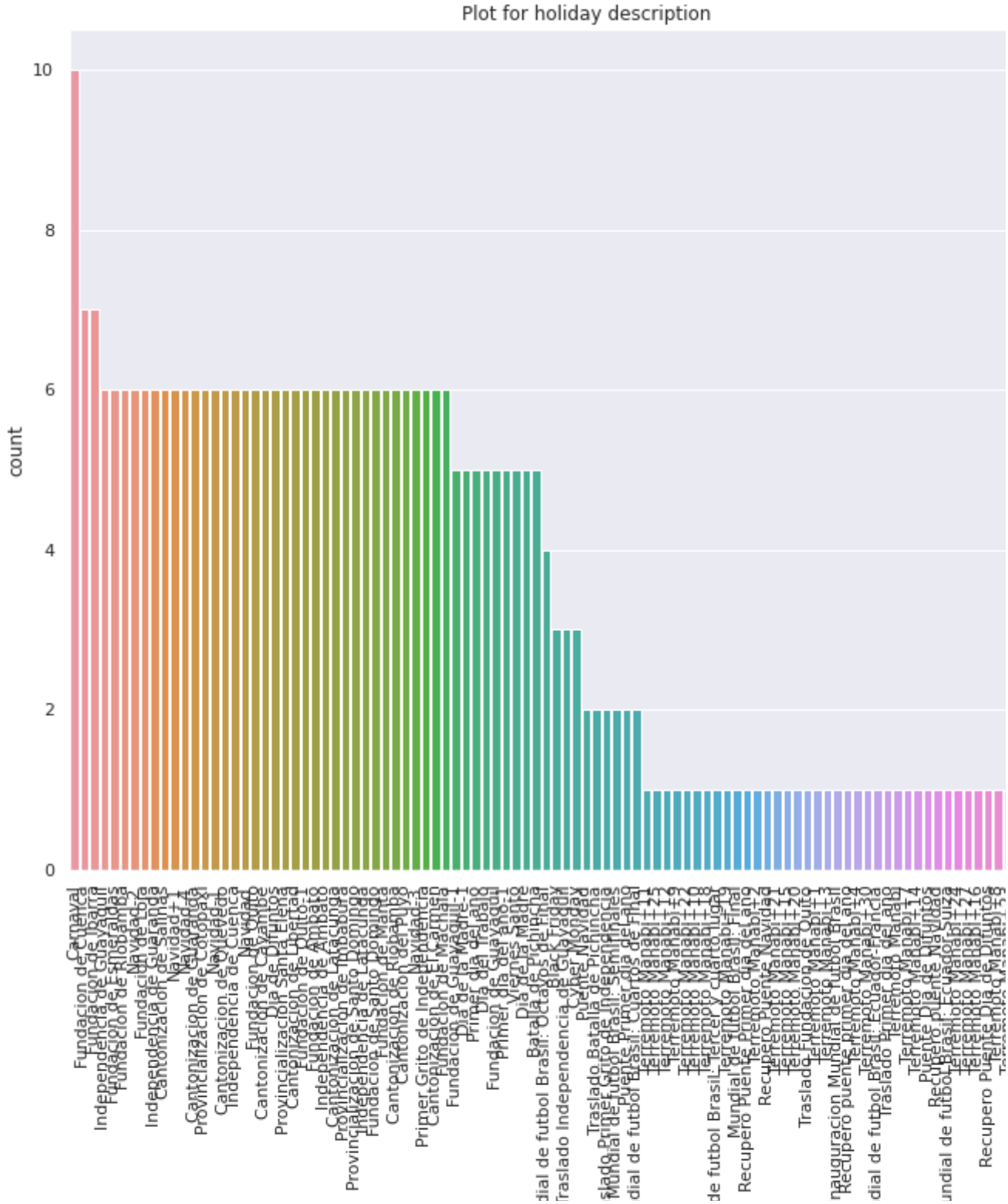
```
holidays_type_df = spark.sql('select type, COUNT(1) as count FROM holidays_data GROUP BY t
holidays_local_df = spark.sql('select locale, COUNT(1) as count FROM holidays_data GROUP B
holidays_desc_df = spark.sql('select description, COUNT(1) as count FROM holidays_data GRO
holidays_transf_df = spark.sql('select transferred, COUNT(1) as count FROM holidays_data G
holidays_locnam_df = spark.sql('select locale_name, COUNT(1) as count FROM holidays_data G
```

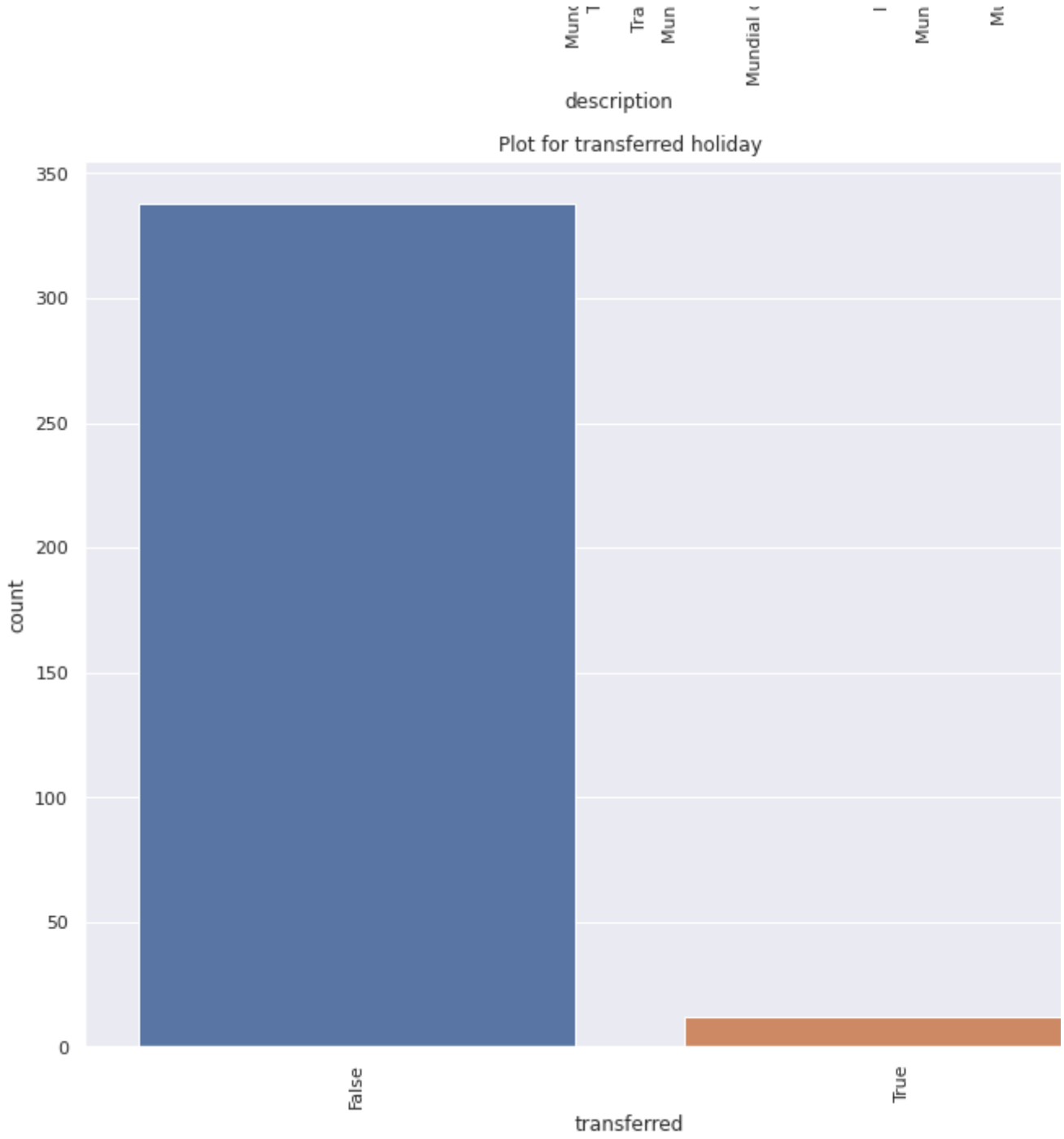
```
holiday_plot(holidays_type_df['type'], holidays_type_df['count'], 'Plot for holiday type')
holiday_plot(holidays_local_df['locale'], holidays_local_df['count'], 'Plot for holiday lo
holiday_plot(holidays_desc_df['description'], holidays_desc_df['count'], 'Plot for holiday
holiday_plot(holidays_transf_df['transferred'], holidays_transf_df['count'], 'Plot for tra
holiday_plot(holidays_locnam_df['locale_name'], holidays_locnam_df['count'], 'Plot for hol
```


Plot for holiday type
count
200
150
100
50
0
Holiday
Event
Additional
Transfer
Bridge
type

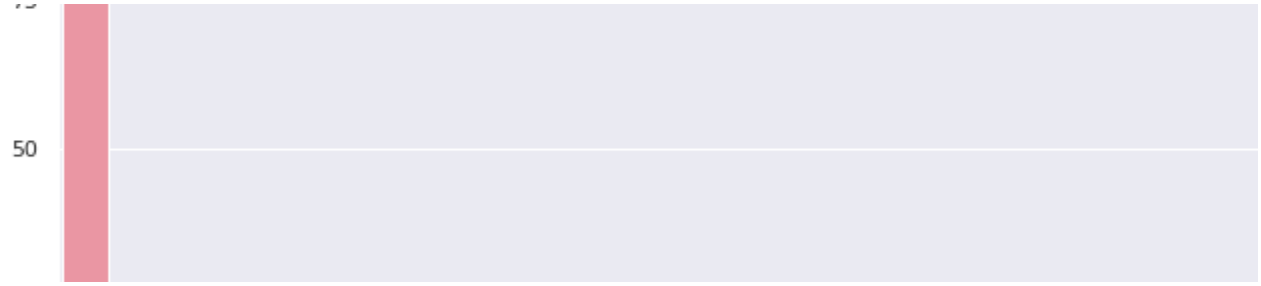

Plot for holiday locale
count
175
150
125
100
75


50
25
0
National
Local
Regional
locale


Plot for holiday description
count
10
8
6
4
2
0


description
Plot for transferred holiday
count
transferred
False
True
0
50
100
150
200
250
300
350


Plot for holiday locale_name
count
75
100
125
150
175



Major Takeaway:

1. Most of the holiday types are national and locale.
2. Carnavals have the highest number of holidays
3. There are few transferred holidays.

Let's dig more on this transferred holiday type

```
spark.sql('select transferred, type, count(1) as count FROM holidays_data GROUP BY transfe
```

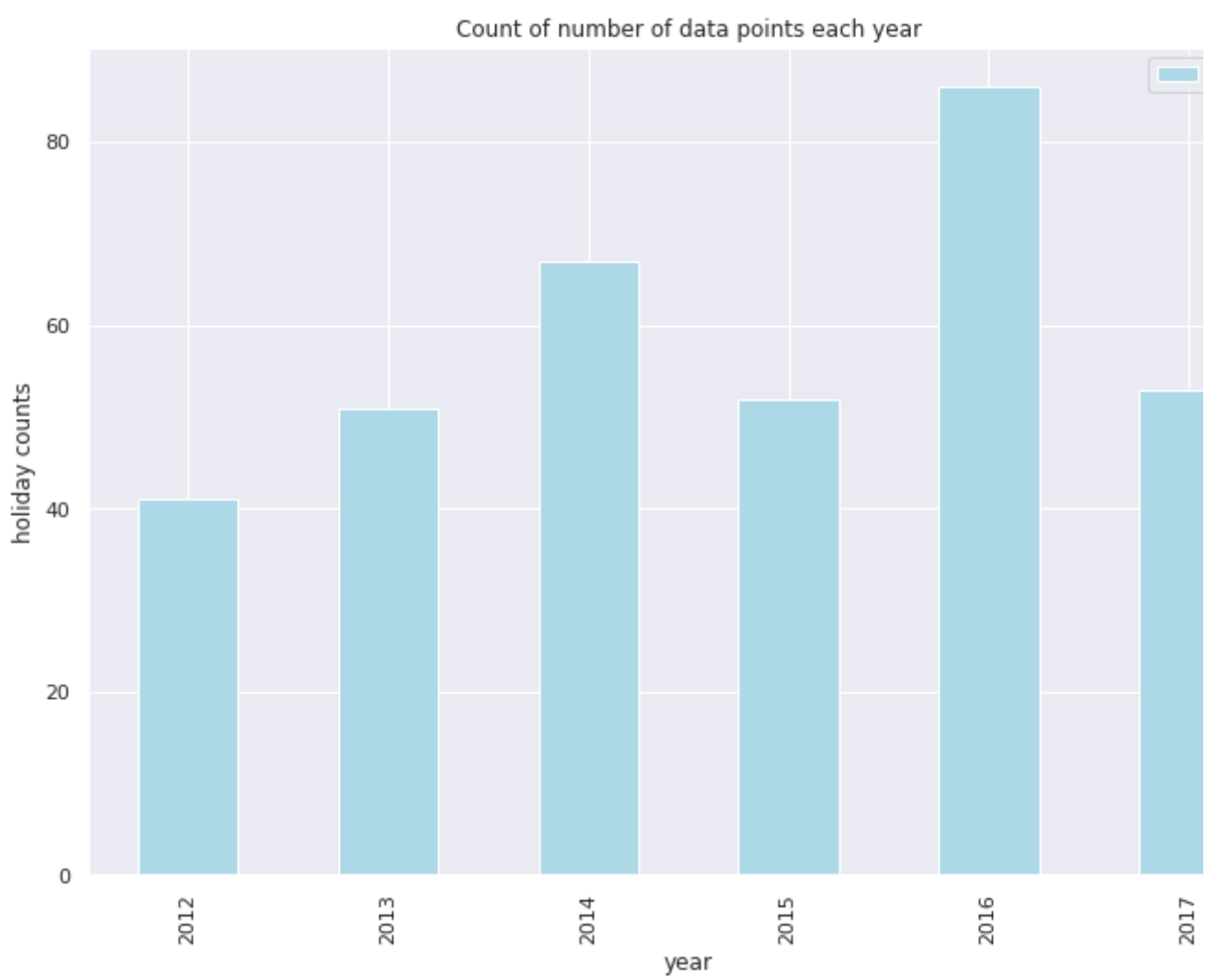
```
+-----------+----------+-----+
|transferred|      type|count|
+-----------+----------+-----+
|      False|    Bridge|    5|
|      False|  Work Day|    5|
|       True|   Holiday|   12|
|      False|  Transfer|   12|
|      False|Additional|   51|
|      False|     Event|   56|
|      False|   Holiday|  209|
+-----------+----------+-----+
```

This transferred features works a little different.It is not directly related to type == Transfer, but is a
the transfer. This can be seen when we group by transferred and type. What this means, is that a tr
data. First on its original date, but with a transferred == TRUE flag. This means that on this day the
day. Instead, the moved holiday has a type == Transfer and a transferred == FALSE on the new date

Let us see the number of holidays on yearly basis

```
holiday_count_yearly = spark.sql('SELECT EXTRACT(YEAR FROM date) as year, count(1) as coun
```

```
x_labels = holiday_count_yearly['year'].values
fig = holiday_count_yearly[['count']].plot(kind='bar', facecolor='lightblue')
fig.set_xticklabels(x_labels)
fig.set_title('Count of number of data points each year')
fig.set_xlabel('year')
fig.set_ylabel('holiday counts')
plt.show()
```



2014 and 2016 have a more number of holidays/events compared to other years, let's see why...

```
holiday_monthly_df = spark.sql('SELECT EXTRACT(MONTH FROM date) as month, EXTRACT(YEAR FRO
```

```
plt.plot(holiday_monthly_df[holiday_monthly_df['year'] == 2013]['month'].values, holiday_m
plt.plot(holiday_monthly_df[holiday_monthly_df['year'] == 2014]['month'].values, holiday_m
plt.plot(holiday_monthly_df[holiday_monthly_df['year'] == 2015]['month'].values, holiday_m
plt.plot(holiday_monthly_df[holiday_monthly_df['year'] == 2016]['month'].values, holiday_m
plt.plot(holiday_monthly_df[holiday_monthly_df['year'] == 2017]['month'].values, holiday_m
plt.legend(['2013', '2014', '2015', '2016', '2017'], loc='upper left')
plt.title("Holidays by year and month")
plt.xlabel('Month')
plt.ylabel('Count')
plt.show()
```

From this we see that in 2016 for the month of 5 and in 2014 for month 7, there are high number o
obviously because of Christmas and year end holidays. Let's see what we have for the period of 20

```
spark.sql('SELECT date, type, description FROM holidays_data WHERE date BETWEEN \'2014-07-
```

	date	type	description
0	2014-07-01	Event	Mundial de futbol Brasil: Octavos de Final
1	2014-07-03	Holiday	Cantonizacion de El Carmen
2	2014-07-03	Holiday	Fundacion de Santo Domingo
3	2014-07-04	Event	Mundial de futbol Brasil: Cuartos de Final
4	2014-07-05	Event	Mundial de futbol Brasil: Cuartos de Final
5	2014-07-08	Event	Mundial de futbol Brasil: Semifinales
6	2014-07-09	Event	Mundial de futbol Brasil: Semifinales
7	2014-07-12	Event	Mundial de futbol Brasil: Tercer y cuarto lugar
8	2014-07-13	Event	Mundial de futbol Brasil: Final
9	2014-07-23	Holiday	Cantonizacion de Cayambe
10	2014-07-24	Additional	Fundacion de Guayaquil-1
11	2014-07-25	Holiday	Fundacion de Guayaquil

So, for 2014, it was the world cup, but then these are not the real holidays, these are the events.

```
spark.sql('SELECT date, type, description FROM holidays_data WHERE date BETWEEN \'2016-05-
```

	date	type	description
0	2016-05-01	Holiday	Dia del Trabajo
1	2016-05-01	Event	Terremoto Manabi+15
2	2016-05-02	Event	Terremoto Manabi+16
3	2016-05-03	Event	Terremoto Manabi+17
4	2016-05-04	Event	Terremoto Manabi+18
5	2016-05-05	Event	Terremoto Manabi+19
6	2016-05-06	Event	Terremoto Manabi+20
7	2016-05-07	Additional	Dia de la Madre-1
8	2016-05-07	Event	Terremoto Manabi+21
9	2016-05-08	Event	Terremoto Manabi+22
10	2016-05-08	Event	Dia de la Madre
11	2016-05-09	Event	Terremoto Manabi+23
12	2016-05-10	Event	Terremoto Manabi+24
13	2016-05-11	Event	Terremoto Manabi+25
14	2016-05-12	Holiday	Cantonizacion del Puyo
15	2016-05-12	Event	Terremoto Manabi+26
16	2016-05-13	Event	Terremoto Manabi+27
17	2016-05-14	Event	Terremoto Manabi+28
18	2016-05-15	Event	Terremoto Manabi+29
19	2016-05-16	Event	Terremoto Manabi+30
20	2016-05-24	Holiday	Batalla de Pichincha

This is when the earthquake occurred, sales may also increase at this time, as people tend to stoc need. This kind of an event is an external factor that shifts our time series from the actual behavio this.

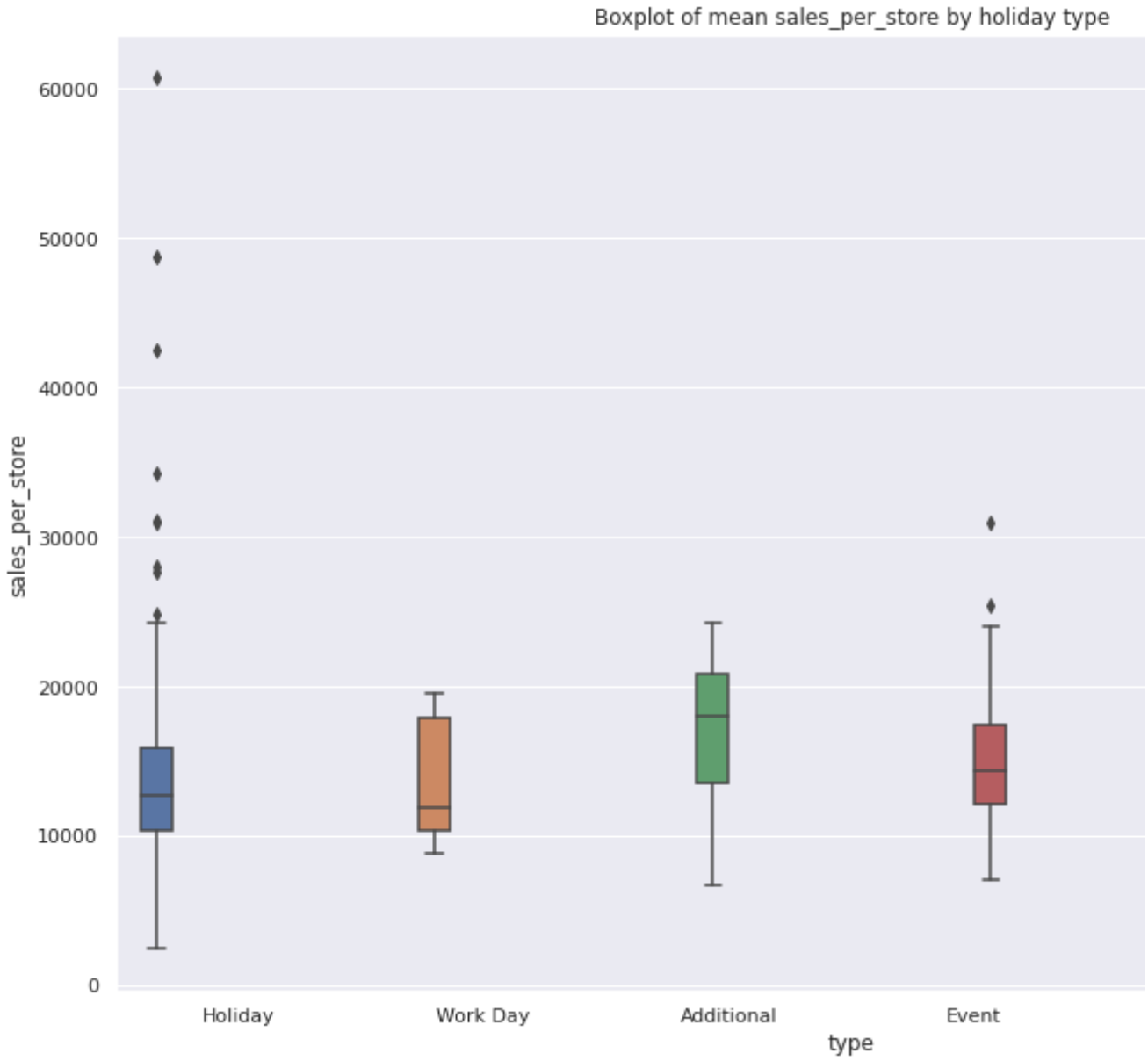
Finally let us see the mean sales per store bases on the holiday type

```
sales_per_store_type = spark.sql('''SELECT date, type, sum(total_sales)/ count(store_nbr)
   (SELECT date, store_nbr, type, sum(unit_sales) as total_sales FROM
     (SELECT a.date, a.store_nbr, a.unit_sales, b.type FROM (
      (SELECT date, store_nbr, unit_sales FROM data) a INNER JOIN
        (SELECT date, type FROM holidays_data) b ON a.date = b.date)) GROUP BY date, store
          GROUP BY date, type ORDER BY date''').toPandas()


sns.set(rc={'figure.figsize':(15,10)})
sns.boxplot(x=sales_per_store_type['type'], y = sales_per_store_type['sales_per_store'], h
```

```
sns.boxplot(x=sales_per_store_type[ type ], y = sales_per_store_type[ sales_per_store ], h
plt.title('Boxplot of mean sales_per_store by holiday type')
```

```
Text(0.5, 1.0, 'Boxplot of mean sales_per_store by holiday type')
```



There is some indication of differences between groups if we talk about mean sales per store with from the box plots...

Conclusions from EDA

Major things that we saw after the analysis:

1. The missing values seen for the onpromotion field are for the initial days for which promotio
2. There is an upward trend in year by year sales, but this is because of the increase in number year, our signal seems close to stationary.
3. There are total 4036 items in the train data and overall item in universe is 4100, so there are i
4. For item-store combination data is not available for all the days, or simply, not all items are s
5. For new items, added recently, we may not have sufficient data points to train on.

6. Total number of stores are 54, out of which few stores geneartes maximum number of sales number of stores and sales as well.
7. Quito itself gives more than 70% of sales.
8. If we look at item level, we have total 33 familes of items, 337 classes and 24% of items are
9. Most of the items fall under common classes, such as Grocery, Beverages, Cleaning, Produc families.
10. If we try to plot signal for items, we see some items having seasonal effects and cyclic effec sample was too small to make a conclusion on trend.
11. When looked at transaction level, it was found that there is a strong spike before Christmas, presumably closed during the holidays. Overall, the sales appear to be stable throughout this
12. One important point that was seen was that the total number of transactions is approximate
13. Oil price drop doesn't seems to have any impact on the sales, as it was seen from the sales p dropping on the sales, so we can say that this feature or data is of no importance to us and
14. We saw high number of holiday events for 2014 and 2016, because of Football WorldCup an
15. When compared mean sales per store based on holiday type, some indication of differences outlier points were seen.

Further Work

Now the major question is how to start with the work and what are the factors that needs to be co make sure what our goal is, we want to forecast the sales per item at store level. Also, we have so work with, but this number will increase if we take every time series at item-store level.

One way to start is by generating results at chain level and then we can simply calculate factors fo results with stores to give the final results.

Or, we can start by simply training the model at item-store level.It will be really important to see ho on which the item is sell or simply saying we may need to to normalize the sales before feeding th say an item is sold at 5 stores in some date and 10 stores in some other day, so instead of feedin the effect by getting sales at a chain store level by simply multiplying by the number of stores whe

Now, we can start our analysis by a simple univariate model, with normalised sales data being use move this further, for which we may need to create lot of new features. Some may be binary featur but we need to make sure we don't overfit in the whole process. All, this is part of the experimenta with the final models and final results.....