#Getting Enviornment Set

In [ ]:

```
!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.t
!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()
```

Out[1]:

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

Major imports

In [ ]:

```
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 numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import datetime
from datetime import date
import calendar
import math
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: Fut
ureWarning: pandas.util.testing is deprecated. Use the functions in the publ
ic API at pandas.testing instead.
  import pandas.util.testing as tm
```

Starting Spark Session

In [ ]:

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

Loading Data

In [ ]:

```
train_df = spark.read.format("csv").option("header", "true").load('/content/drive/My D
test_df = spark.read.format("csv").option("header", "true").load('/content/drive/My Dr
transactions_df = spark.read.format("csv").option("header", "true").load('/content/dri
stores_df = spark.read.format("csv").option("header", "true").load('/content/drive/My I
oil_df = spark.read.format("csv").option("header", "true").load('/content/drive/My Dri
items_df = spark.read.format("csv").option("header", "true").load('/content/drive/My D
holidays_events_df = spark.read.format("csv").option("header", "true").load('/content/
```

Creating Views for writing SQL queries....

In [ ]:

```
train_df = train_df.withColumn('id', col('id').cast(IntegerType())).withColumn('date',
stores_df = stores_df.withColumn('store_nbr', col('store_nbr').cast(IntegerType())).wi
items_df = items_df.withColumn('item_nbr', col('item_nbr').cast(IntegerType())).withCo
transactions_df = transactions_df.withColumn('date', col('date').cast(DateType())).wit
holidays_events_df= holidays_events_df.withColumn('date', col('date').cast(DateType())
test_df = test_df.withColumn('date', col('date').cast(DateType())).withColumn('store_n
```

In [ ]:

```
train_df.createOrReplaceTempView('data')
train_df.cache

stores_df.createOrReplaceTempView('store_data')
stores_df.cache

items_df.createOrReplaceTempView('items_data')
items_df.cache

transactions_df.createOrReplaceTempView('transactions_data')
transactions_df.cache

holidays_events_df.createOrReplaceTempView('holidays_data')
holidays_events_df.cache

test_df.createOrReplaceTempView('test_view')
test_df.cache
```

Out[6]:

```
<bound method DataFrame.cache of DataFrame[id: string, date: date, store_nb
r: int, item_nbr: int, onpromotion: string]>
```

#Data Collection

Before building features and stacking data from other views, 2 things we are considering:

1. From the EDA we did saw that oil prices didn't have any impact on the sales, so oil prices will be skipped from our final data that we will build for modelling purpose
2. The onpromotion field was for the starting days, we can say that at that point they might not be tracking promotions, we have 2 option either, we can give a value of 2(which will give us info that promotions were not tracked), since we have a lot of data, and we are considering data at day level, considering data with data points only for 2017.

3. Eliminating negative values, making them zero and chaning onpromotion field to 0 and 1, where 0 is not on promotion and 1 is on promotion
4. Normaizing unit sales with log(x + 1), as for real-valued input, log(x + 1) is accurate, also for x so small that 1 + x == 1 in floating-point accuracy.

In [ ]:

```
train_df = spark.sql('''SELECT date, store_nbr, item_nbr, CASE WHEN unit_sales < 0 THEI
                        CASE WHEN onpromotion = \'True\' THEN 1 ELSE 0 END as onpromot:
```

In [ ]:

```
train_df.cache
```

Out[8]:

```
<bound method DataFrame.cache of DataFrame[date: date, store_nbr: int, item_
nbr: int, unit_sales: double, onpromotion: int]>
```

In [ ]:

```
train_df.show()
```

```
+----------+---------+--------+------------------+-----------+
|      date|store_nbr|item_nbr|        unit_sales|onpromotion|
+----------+---------+--------+------------------+-----------+
|2017-01-01|       25|   99197|0.6931471805599453|          0|
|2017-01-01|       25|  103665|2.0794415416798357|          0|
|2017-01-01|       25|  105574|0.6931471805599453|          0|
|2017-01-01|       25|  105857|1.6094379124341003|          0|
|2017-01-01|       25|  106716|1.0986122886681098|          0|
|2017-01-01|       25|  108698|1.0986122886681098|          0|
|2017-01-01|       25|  108786|0.6931471805599453|          0|
|2017-01-01|       25|  108797|0.6931471805599453|          0|
|2017-01-01|       25|  108862|0.6931471805599453|          0|
|2017-01-01|       25|  108952|1.0986122886681098|          0|
|2017-01-01|       25|  114790|1.0986122886681098|          0|
|2017-01-01|       25|  114800|1.9459101490553132|          0|
|2017-01-01|       25|  115267|1.0986122886681098|          0|
|2017-01-01|       25|  115693|0.6931471805599453|          1|
|2017-01-01|       25|  115720|1.9459101490553132|          0|
|2017-01-01|       25|  115850|0.6931471805599453|          0|
|2017-01-01|       25|  115891|2.5649493574615367|          0|
|2017-01-01|       25|  115892|1.3862943611198906|          0|
|2017-01-01|       25|  115893|1.3862943611198906|          0|
|2017-01-01|       25|  115894|2.0794415416798357|          0|
+----------+---------+--------+------------------+-----------+
only showing top 20 rows
```

In [ ]:

```
#final_df.coalesce(1).write.option("header","true").option("sep",",").mode("overwrite",
```

In [ ]:

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

```
Total number of rows in train: 23808261
```

In [ ]:

```
#final_df.createOrReplaceTempView('final_data')
#final_df.cache
```

Out[76]:

```
<bound method DataFrame.cache of DataFrame[date: date, store_nbr: int, item_
nbr: int, unit_sales: double, onpromotion: int]>
```

Since we are conisdering data daily, and we want to convert our time series problem to a supervised learning problem, we will be using the Windowing method to do so, what we do here is we unstack time from rows to column with each column representing a timestep

One important update that we need to consider here is that, we don't have information about stock outs, so for store-item level, we will be seeing null values when windowing data, we are going fill all null with 0 for me, we may smooth it later, but this can give us some information on the stock out/not on shelf, but again this is an assumption and we are going forward with this for now.

Data collection using train file a item-store level

Using Sales Data

In [ ]:

```
train_sales_df = train_df.groupby('store_nbr', 'item_nbr').pivot('date').sum('unit_sal
```

In [ ]:

```
spark.conf.set("spark.sql.execution.arrow.enabled", "true")
spark.conf.set("spark.sql.crossJoin.enabled", "true")
```

In [ ]:

```
print(train_sales_df.toPandas().shape)
```

```
(167515, 229)
```

This 167515 is the total number of item-store combination we have in our sales data, we may train our models with the same number of data points if we continue with this approach of stacking information from all our files....

In [ ]:

```
train_sales_df.cache
```

Out[15]:

```
<bound method DataFrame.cache of DataFrame[store_nbr: int, item_nbr: int, 20
17-01-01: double, 2017-01-02: double, 2017-01-03: double, 2017-01-04: doubl
e, 2017-01-05: double, 2017-01-06: double, 2017-01-07: double, 2017-01-08: d
ouble, 2017-01-09: double, 2017-01-10: double, 2017-01-11: double, 2017-01-1
2: double, 2017-01-13: double, 2017-01-14: double, 2017-01-15: double, 2017-
01-16: double, 2017-01-17: double, 2017-01-18: double, 2017-01-19: double, 2
017-01-20: double, 2017-01-21: double, 2017-01-22: double, 2017-01-23: doubl
e, 2017-01-24: double, 2017-01-25: double, 2017-01-26: double, 2017-01-27: d
ouble, 2017-01-28: double, 2017-01-29: double, 2017-01-30: double, 2017-01-3
1: double, 2017-02-01: double, 2017-02-02: double, 2017-02-03: double, 2017-
02-04: double, 2017-02-05: double, 2017-02-06: double, 2017-02-07: double, 2
017-02-08: double, 2017-02-09: double, 2017-02-10: double, 2017-02-11: doubl
e, 2017-02-12: double, 2017-02-13: double, 2017-02-14: double, 2017-02-15: d
ouble, 2017-02-16: double, 2017-02-17: double, 2017-02-18: double, 2017-02-1
9: double, 2017-02-20: double, 2017-02-21: double, 2017-02-22: double, 2017-
02-23: double, 2017-02-24: double, 2017-02-25: double, 2017-02-26: double, 2
017-02-27: double, 2017-02-28: double, 2017-03-01: double, 2017-03-02: doubl
e, 2017-03-03: double, 2017-03-04: double, 2017-03-05: double, 2017-03-06: d
ouble, 2017-03-07: double, 2017-03-08: double, 2017-03-09: double, 2017-03-1
0: double, 2017-03-11: double, 2017-03-12: double, 2017-03-13: double, 2017-
03-14: double, 2017-03-15: double, 2017-03-16: double, 2017-03-17: double, 2
017-03-18: double, 2017-03-19: double, 2017-03-20: double, 2017-03-21: doubl
e, 2017-03-22: double, 2017-03-23: double, 2017-03-24: double, 2017-03-25: d
ouble, 2017-03-26: double, 2017-03-27: double, 2017-03-28: double, 2017-03-2
9: double, 2017-03-30: double, 2017-03-31: double, 2017-04-01: double, 2017-
04-02: double, 2017-04-03: double, 2017-04-04: double, 2017-04-05: double, 2
017-04-06: double, 2017-04-07: double, 2017-04-08: double, 2017-04-09: doubl
e, 2017-04-10: double, 2017-04-11: double, 2017-04-12: double, 2017-04-13: d
ouble, 2017-04-14: double, 2017-04-15: double, 2017-04-16: double, 2017-04-1
7: double, 2017-04-18: double, 2017-04-19: double, 2017-04-20: double, 2017-
04-21: double, 2017-04-22: double, 2017-04-23: double, 2017-04-24: double, 2
017-04-25: double, 2017-04-26: double, 2017-04-27: double, 2017-04-28: doubl
e, 2017-04-29: double, 2017-04-30: double, 2017-05-01: double, 2017-05-02: d
ouble, 2017-05-03: double, 2017-05-04: double, 2017-05-05: double, 2017-05-0
6: double, 2017-05-07: double, 2017-05-08: double, 2017-05-09: double, 2017-
05-10: double, 2017-05-11: double, 2017-05-12: double, 2017-05-13: double, 2
017-05-14: double, 2017-05-15: double, 2017-05-16: double, 2017-05-17: doubl
e, 2017-05-18: double, 2017-05-19: double, 2017-05-20: double, 2017-05-21: d
ouble, 2017-05-22: double, 2017-05-23: double, 2017-05-24: double, 2017-05-2
5: double, 2017-05-26: double, 2017-05-27: double, 2017-05-28: double, 2017-
05-29: double, 2017-05-30: double, 2017-05-31: double, 2017-06-01: double, 2
017-06-02: double, 2017-06-03: double, 2017-06-04: double, 2017-06-05: doubl
e, 2017-06-06: double, 2017-06-07: double, 2017-06-08: double, 2017-06-09: d
ouble, 2017-06-10: double, 2017-06-11: double, 2017-06-12: double, 2017-06-1
3: double, 2017-06-14: double, 2017-06-15: double, 2017-06-16: double, 2017-
06-17: double, 2017-06-18: double, 2017-06-19: double, 2017-06-20: double, 2
017-06-21: double, 2017-06-22: double, 2017-06-23: double, 2017-06-24: doubl
e, 2017-06-25: double, 2017-06-26: double, 2017-06-27: double, 2017-06-28: d
ouble, 2017-06-29: double, 2017-06-30: double, 2017-07-01: double, 2017-07-0
2: double, 2017-07-03: double, 2017-07-04: double, 2017-07-05: double, 2017-
07-06: double, 2017-07-07: double, 2017-07-08: double, 2017-07-09: double, 2
017-07-10: double, 2017-07-11: double, 2017-07-12: double, 2017-07-13: doubl
e, 2017-07-14: double, 2017-07-15: double, 2017-07-16: double, 2017-07-17: d
ouble, 2017-07-18: double, 2017-07-19: double, 2017-07-20: double, 2017-07-2
1: double, 2017-07-22: double, 2017-07-23: double, 2017-07-24: double, 2017-
```

```
07-25: double, 2017-07-26: double, 2017-07-27: double, 2017-07-28: double, 2
017-07-29: double, 2017-07-30: double, 2017-07-31: double, 2017-08-01: doubl
e, 2017-08-02: double, 2017-08-03: double, 2017-08-04: double, 2017-08-05: d
ouble, 2017-08-06: double, 2017-08-07: double, 2017-08-08: double, 2017-08-0
9: double, 2017-08-10: double, 2017-08-11: double, 2017-08-12: double, 2017-
08-13: double, 2017-08-14: double, 2017-08-15: double]>
```

Using promo data

Now test files does have promotion details, so adding that information to create the promo df.

In [ ]:

```
train_promo_df = train_df.groupby('store_nbr', 'item_nbr').pivot('date').sum('onpromot
```

In [ ]:

```
print(train_promo_df.toPandas().shape)
```

```
(167515, 229)
```

In [ ]:

```
train_promo_df.cache
```

Out[19]:

```
<bound method DataFrame.cache of DataFrame[store_nbr: int, item_nbr: int, 20
17-01-01: bigint, 2017-01-02: bigint, 2017-01-03: bigint, 2017-01-04: bigin
t, 2017-01-05: bigint, 2017-01-06: bigint, 2017-01-07: bigint, 2017-01-08: b
igint, 2017-01-09: bigint, 2017-01-10: bigint, 2017-01-11: bigint, 2017-01-1
2: bigint, 2017-01-13: bigint, 2017-01-14: bigint, 2017-01-15: bigint, 2017-
01-16: bigint, 2017-01-17: bigint, 2017-01-18: bigint, 2017-01-19: bigint, 2
017-01-20: bigint, 2017-01-21: bigint, 2017-01-22: bigint, 2017-01-23: bigin
t, 2017-01-24: bigint, 2017-01-25: bigint, 2017-01-26: bigint, 2017-01-27: b
igint, 2017-01-28: bigint, 2017-01-29: bigint, 2017-01-30: bigint, 2017-01-3
1: bigint, 2017-02-01: bigint, 2017-02-02: bigint, 2017-02-03: bigint, 2017-
02-04: bigint, 2017-02-05: bigint, 2017-02-06: bigint, 2017-02-07: bigint, 2
017-02-08: bigint, 2017-02-09: bigint, 2017-02-10: bigint, 2017-02-11: bigin
t, 2017-02-12: bigint, 2017-02-13: bigint, 2017-02-14: bigint, 2017-02-15: b
igint, 2017-02-16: bigint, 2017-02-17: bigint, 2017-02-18: bigint, 2017-02-1
9: bigint, 2017-02-20: bigint, 2017-02-21: bigint, 2017-02-22: bigint, 2017-
02-23: bigint, 2017-02-24: bigint, 2017-02-25: bigint, 2017-02-26: bigint, 2
017-02-27: bigint, 2017-02-28: bigint, 2017-03-01: bigint, 2017-03-02: bigin
t, 2017-03-03: bigint, 2017-03-04: bigint, 2017-03-05: bigint, 2017-03-06: b
igint, 2017-03-07: bigint, 2017-03-08: bigint, 2017-03-09: bigint, 2017-03-1
0: bigint, 2017-03-11: bigint, 2017-03-12: bigint, 2017-03-13: bigint, 2017-
03-14: bigint, 2017-03-15: bigint, 2017-03-16: bigint, 2017-03-17: bigint, 2
017-03-18: bigint, 2017-03-19: bigint, 2017-03-20: bigint, 2017-03-21: bigin
t, 2017-03-22: bigint, 2017-03-23: bigint, 2017-03-24: bigint, 2017-03-25: b
igint, 2017-03-26: bigint, 2017-03-27: bigint, 2017-03-28: bigint, 2017-03-2
9: bigint, 2017-03-30: bigint, 2017-03-31: bigint, 2017-04-01: bigint, 2017-
04-02: bigint, 2017-04-03: bigint, 2017-04-04: bigint, 2017-04-05: bigint, 2
017-04-06: bigint, 2017-04-07: bigint, 2017-04-08: bigint, 2017-04-09: bigin
t, 2017-04-10: bigint, 2017-04-11: bigint, 2017-04-12: bigint, 2017-04-13: b
igint, 2017-04-14: bigint, 2017-04-15: bigint, 2017-04-16: bigint, 2017-04-1
7: bigint, 2017-04-18: bigint, 2017-04-19: bigint, 2017-04-20: bigint, 2017-
04-21: bigint, 2017-04-22: bigint, 2017-04-23: bigint, 2017-04-24: bigint, 2
017-04-25: bigint, 2017-04-26: bigint, 2017-04-27: bigint, 2017-04-28: bigin
t, 2017-04-29: bigint, 2017-04-30: bigint, 2017-05-01: bigint, 2017-05-02: b
igint, 2017-05-03: bigint, 2017-05-04: bigint, 2017-05-05: bigint, 2017-05-0
6: bigint, 2017-05-07: bigint, 2017-05-08: bigint, 2017-05-09: bigint, 2017-
05-10: bigint, 2017-05-11: bigint, 2017-05-12: bigint, 2017-05-13: bigint, 2
017-05-14: bigint, 2017-05-15: bigint, 2017-05-16: bigint, 2017-05-17: bigin
t, 2017-05-18: bigint, 2017-05-19: bigint, 2017-05-20: bigint, 2017-05-21: b
igint, 2017-05-22: bigint, 2017-05-23: bigint, 2017-05-24: bigint, 2017-05-2
5: bigint, 2017-05-26: bigint, 2017-05-27: bigint, 2017-05-28: bigint, 2017-
05-29: bigint, 2017-05-30: bigint, 2017-05-31: bigint, 2017-06-01: bigint, 2
017-06-02: bigint, 2017-06-03: bigint, 2017-06-04: bigint, 2017-06-05: bigin
t, 2017-06-06: bigint, 2017-06-07: bigint, 2017-06-08: bigint, 2017-06-09: b
igint, 2017-06-10: bigint, 2017-06-11: bigint, 2017-06-12: bigint, 2017-06-1
3: bigint, 2017-06-14: bigint, 2017-06-15: bigint, 2017-06-16: bigint, 2017-
06-17: bigint, 2017-06-18: bigint, 2017-06-19: bigint, 2017-06-20: bigint, 2
017-06-21: bigint, 2017-06-22: bigint, 2017-06-23: bigint, 2017-06-24: bigin
t, 2017-06-25: bigint, 2017-06-26: bigint, 2017-06-27: bigint, 2017-06-28: b
igint, 2017-06-29: bigint, 2017-06-30: bigint, 2017-07-01: bigint, 2017-07-0
2: bigint, 2017-07-03: bigint, 2017-07-04: bigint, 2017-07-05: bigint, 2017-
07-06: bigint, 2017-07-07: bigint, 2017-07-08: bigint, 2017-07-09: bigint, 2
017-07-10: bigint, 2017-07-11: bigint, 2017-07-12: bigint, 2017-07-13: bigin
t, 2017-07-14: bigint, 2017-07-15: bigint, 2017-07-16: bigint, 2017-07-17: b
igint, 2017-07-18: bigint, 2017-07-19: bigint, 2017-07-20: bigint, 2017-07-2
1: bigint, 2017-07-22: bigint, 2017-07-23: bigint, 2017-07-24: bigint, 2017-
```

```
07-25: bigint, 2017-07-26: bigint, 2017-07-27: bigint, 2017-07-28: bigint, 2
017-07-29: bigint, 2017-07-30: bigint, 2017-07-31: bigint, 2017-08-01: bigin
t, 2017-08-02: bigint, 2017-08-03: bigint, 2017-08-04: bigint, 2017-08-05: b
igint, 2017-08-06: bigint, 2017-08-07: bigint, 2017-08-08: bigint, 2017-08-0
9: bigint, 2017-08-10: bigint, 2017-08-11: bigint, 2017-08-12: bigint, 2017-
08-13: bigint, 2017-08-14: bigint, 2017-08-15: bigint]>
```

In [ ]:

```
test_df = spark.sql('SELECT date, store_nbr, item_nbr, CASE WHEN onpromotion = \'True\
```

In [ ]:

```
test_promo_df = test_df.groupby('store_nbr', 'item_nbr').pivot('date').sum('onpromotio
```

In [ ]:

```
print(test_promo_df.toPandas().shape)
```

```
(210654, 18)
```

test_promo has more number of rows than that of train, but we will stack only the rows that are in train...

In [ ]:

```
test_promo_df.cache
```

Out[24]:

```
<bound method DataFrame.cache of DataFrame[store_nbr: int, item_nbr: int, 20
17-08-16: bigint, 2017-08-17: bigint, 2017-08-18: bigint, 2017-08-19: bigin
t, 2017-08-20: bigint, 2017-08-21: bigint, 2017-08-22: bigint, 2017-08-23: b
igint, 2017-08-24: bigint, 2017-08-25: bigint, 2017-08-26: bigint, 2017-08-2
7: bigint, 2017-08-28: bigint, 2017-08-29: bigint, 2017-08-30: bigint, 2017-
08-31: bigint]>
```

In [ ]:

```
#creating the list of items part of test_promo
train_promo_store_item_lst = [(i[0], i[1]) for i in train_promo_df.select('store_nbr',
```

In [ ]:

```
len(train_promo_store_item_lst)
```

Out[26]:

```
167515
```

In [ ]:

```
train_promo_store_item_lst_str = [",".join([str(x) for x in item]) for item in train_pi
```

In [ ]:

```
import pyspark.sql.functions as f
```

In [ ]:

```
#filtering rows from the test_promo
test_promo_df = test_promo_df.withColumn("combined_id", f.concat(f.col("store_nbr"), f
    .where(f.col("combined_id").isin(train_promo_store_item_lst_str))
```

In [ ]:

```
test_promo_df = test_promo_df.drop('combined_id')
```

In [ ]:

```
train_promo_df = train_promo_df.toPandas()
test_promo_df = test_promo_df.toPandas()
```

In [ ]:

```
final_promo_df = train_promo_df.merge(test_promo_df, on = ['store_nbr', 'item_nbr'], h
```

In [ ]:

```
final_promo_df = train_promo_df.join(test_promo_df, on=['store_nbr', 'item_nbr'], how=
```

In [ ]:

```
train_sales_df = train_sales_df.toPandas()
```

In [ ]:

```
print('Shape of sales df is {}'.format(train_sales_df.shape))
print('Shape of promo df is {}'.format(final_promo_df.shape))
```

```
Shape of sales df is (167515, 229)
Shape of promo df is (167515, 245)
```

So we have created df such that each row correspond to sales and promotion details for item-store level. We will gather this detail at store level and item level separately in next few cells...

In [ ]:

```
train_sales_df.head()
```

Out[3]:

	store_nbr	item_nbr	2017-01-01	2017-01-02	2017-01-03	2017-01-04	2017-01-05	2017-01-06	2017-01-07	201
0	1	96995	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0(
1	1	99197	0.0	0.000000	1.386294	0.693147	0.693147	0.693147	1.098612	0.0(
2	1	103520	0.0	0.693147	1.098612	0.000000	1.098612	1.386294	0.693147	0.0(
3	1	103665	0.0	0.000000	0.000000	1.386294	1.098612	1.098612	0.693147	1.0(
4	1	105574	0.0	0.000000	1.791759	2.564949	2.302585	1.945910	1.609438	1.0(

5 rows × 229 columns

In [ ]:

```
1 final_promo_df.head()
```

Out[4]:

	store_nbr	item_nbr	2017-01-01	2017-01-02	2017-01-03	2017-01-04	2017-01-05	2017-01-06	2017-01-07	2017-01-08	2017-01-09	2017-01-10	20 01
0	1	96995	0	0	0	0	0	0	0	0	0	0	
1	1	99197	0	0	0	0	0	0	0	0	0	0	
2	1	103520	0	0	0	0	0	0	0	0	0	0	
3	1	103665	0	0	0	0	0	0	0	0	0	0	
4	1	105574	0	0	1	0	0	1	0	0	0	0	

5 rows × 245 columns

Data Collection using item file

In [ ]:

```
1 items_df = items_df.toPandas()
```

In [ ]:

```
1 items_df.head()
```

Out[87]:

	item_nbr	family	class	perishable
0	96995	GROCERY I	1093	0
1	99197	GROCERY I	1067	0
2	103501	CLEANING	3008	0
3	103520	GROCERY I	1028	0
4	103665	BREAD/BAKERY	2712	1

Adding item info to our sales df to form data points at item level....

In [ ]:

```
1 items_sales_df = train_sales_df.drop('store_nbr', axis =1).groupby('item_nbr').sum().r
```

In [ ]:

```
1 items_promo_df = final_promo_df.drop('store_nbr', axis =1).groupby('item_nbr').sum().r
```

In [ ]:

```
items_promo_df.head()
```

Out[11]:

	item_nbr	2017-01-01	2017-01-02	2017-01-03	2017-01-04	2017-01-05	2017-01-06	2017-01-07	2017-01-08	2017-01-09	2017-01-10	2017-01-11	2017-01-12
0	96995	0	0	0	0	0	0	0	0	0	0	0	0
1	99197	0	0	0	0	0	0	0	0	0	0	0	0
2	103501	0	0	0	0	0	0	0	0	0	0	0	0
3	103520	0	0	0	0	0	0	0	0	0	0	0	0
4	103665	0	0	0	0	0	0	0	13	0	0	0	0

5 rows × 244 columns

In [ ]:

```
items_sales_df.head()
```

Out[12]:

	item_nbr	2017-01-01	2017-01-02	2017-01-03	2017-01-04	2017-01-05	2017-01-06	2017-01-07	2017
0	96995	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
1	99197	0.693147	17.422746	16.604036	20.569303	16.203025	16.278613	14.775909	17.317
2	103501	0.000000	55.868320	54.627085	42.810313	39.555298	35.717635	47.208504	47.542
3	103520	0.000000	38.875486	35.822995	34.979211	42.252967	51.397412	49.505990	33.846
4	103665	2.079442	56.225402	40.233610	46.138063	38.100507	49.690810	54.725492	54.286

5 rows × 228 columns

In [ ]:

```
items_sales_df.shape, items_promo_df.shape
```

Out[14]:

```
((4018, 228), (4018, 244))
```

Data collection using store file

In [ ]:

```
stores_df = stores_df.toPandas()
```

In [ ]:

```
stores_df.columns
```

Out[96]:

```
Index(['store_nbr', 'state', 'type', 'cluster'], dtype='object')
```

One thing we saw from EDA was that city & state add same sort of information in terms of total sales and we can use only one of these for our model

In [ ]:

```
stores_df = stores_df.drop('city', axis = 1)
```

In [ ]:

```
stores_df.head()
```

Out[97]:

	store_nbr	state	type	cluster
0	1	Pichincha	D	13
1	2	Pichincha	D	13
2	3	Pichincha	D	8
3	4	Pichincha	D	9
4	5	Santo Domingo de los Tsachilas	D	4

In [ ]:

```
store_sales_df = train_sales_df.drop('item_nbr', axis =1).groupby('store_nbr').sum().r
```

In [ ]:

```
store_promo_df = final_promo_df.drop('item_nbr', axis =1).groupby('store_nbr').sum().r
```

In [ ]:

```
store_sales_df.shape, store_promo_df.shape
```

Out[17]:

```
((54, 228), (54, 244))
```

In [ ]:

```
store_sales_df.head()
```

Out[18]:

	store_nbr	2017-01-01	2017-01-02	2017-01-03	2017-01-04	2017-01-05	2017-01-06	2017-0
0	1	0.0	1909.598470	3734.738494	3721.581691	3381.289871	3405.684219	3267.28
1	2	0.0	4917.580925	4215.896120	4179.835998	3633.582513	4091.651346	4416.10
2	3	0.0	7122.511109	6226.395166	6144.813040	5641.845924	6099.497740	6790.19
3	4	0.0	4729.934125	4120.414355	3914.114557	3479.424707	3761.194571	4200.49
4	5	0.0	3488.186848	3315.595652	3313.125877	2781.415432	3173.998698	3355.08

5 rows × 228 columns

In [ ]:

```
store_promo_df.head()
```

Out[19]:

	store_nbr	2017-01-01	2017-01-02	2017-01-03	2017-01-04	2017-01-05	2017-01-06	2017-01-07	2017-01-08	2017-01-09	2017-01-10	2017-01-11	2017-01-12
0	1	0	128	197	479	152	358	156	134	166	168	381	159
1	2	0	237	199	508	168	348	192	194	163	176	394	171
2	3	0	246	230	555	184	404	224	193	198	211	421	203
3	4	0	221	216	486	157	357	201	199	166	170	370	174
4	5	0	239	248	482	181	324	210	210	182	215	383	166

5 rows × 244 columns

In [ ]:

```
items_sales_df.to_csv('/content/drive/My Drive/Grocery/items_sales.csv', index = False
items_promo_df.to_csv('/content/drive/My Drive/Grocery/items_promo.csv', index = False
store_sales_df.to_csv('/content/drive/My Drive/Grocery/store_sales.csv', index = False
store_promo_df.to_csv('/content/drive/My Drive/Grocery/store_promo.csv', index = False
```

So with this we have created 6 dataframes which have sales and onpromo information at item-store level, item level, store level...

Important thing that we saw that we had 200k data points(roughly) on test file, and 167k on train, so do have some data points on test for which we don't have any information on train, also, during baseline model creation the same information was seen, so this is where class information comes handy. We have 4018 items belonging to 337 classes, let us gather same inforamtion that we fetched at store-class

level, also, if we don't have a data point where we don't have any information on store-class, we will use class information for such data points, and for remaining, zero prediction as nothing much can be done if we don't have any information with us.

Data collection at store-class level

In [ ]:

```
store_class_sales_df = train_sales_df
store_class_sales_df['class'] = items_promo_df['class'].values
store_class_sales_df= store_class_sales_df.drop('item_nbr', axis = 1)
```

In [ ]:

```
store_class_sales_df = store_class_sales_df.groupby(['class', 'store_nbr']).sum().rese
```

In [ ]:

```
store_class_promo_df = final_promo_df
store_class_promo_df['class'] = items_promo_df['class'].values
store_class_promo_df = store_class_promo_df.drop('item_nbr', axis = 1)
```

In [ ]:

```
store_class_promo_df = store_class_promo_df.groupby(['class', 'store_nbr']).sum().rese
```

In [ ]:

```
store_class_promo_df.head()
```

Out[126]:

	class	store_nbr	2017-01-01	2017-01-02	2017-01-03	2017-01-04	2017-01-05	2017-01-06	2017-01-07	2017-01-08	2017-01-09	2017-01-10	2017-01-11
0	1002	1	0	0	0	0	0	0	0	0	0	0	0
1	1002	2	0	0	0	0	0	0	0	0	0	0	0
2	1002	3	0	0	0	0	0	0	0	0	0	0	0
3	1002	4	0	0	0	0	0	0	0	0	0	0	0
4	1002	5	0	0	0	0	0	0	0	0	0	1	0

5 rows × 245 columns

In [ ]:

```
store_class_sales_df.head()
```

Out[127]:

	class	store_nbr	2017-01-01	2017-01-02	2017-01-03	2017-01-04	2017-01-05	2017-01-06	2017-01-07	
0	1002	1	0.0	6.291569	11.901285	9.939627	12.817576	10.961278	13.708549	
1	1002	2	0.0	27.836761	21.942946	23.265525	20.405583	23.207544	32.629193	3
2	1002	3	0.0	42.484074	29.286804	35.991684	29.124900	31.628492	36.412191	3
3	1002	4	0.0	28.353452	21.278199	22.805993	20.207757	19.822911	24.720218	3
4	1002	5	0.0	19.157935	15.744315	14.909440	12.177673	12.765460	12.306750	1

5 rows × 229 columns

In [ ]:

```
store_class_sales_df.shape
```

Out[22]:

```
(15826, 229)
```

In [ ]:

```
store_class_sales_df.to_csv('/content/drive/My Drive/Grocery/store_class_sales.csv', i
store_class_promo_df.to_csv('/content/drive/My Drive/Grocery/store_class_promo.csv', i
```

Data collection using transaction data

Using the transaction file, only information that I can thought of was the number of items per transaction, which can give us some information, not sure now, how will be using this information, but preparing data using this....

This df is at store level

In [ ]:

```
items_per_transaction = spark.sql('''SELECT transaction_view.store_nbr, transaction_vi
             FROM
             (SELECT store_nbr, date , SUM(transactions) AS total_transaction FROM tra
             GROUP BY store_nbr, date ORDER BY store_nbr, date) AS transaction_view
             INNER JOIN
             (SELECT store_nbr, date , SUM(unit_sales) AS total_sales FROM data WHERE
             AND unit_sales > 0 GROUP BY store_nbr, date ORDER BY store_nbr, date
             ) AS sales_view
             ON transaction_view.store_nbr = sales_view.store_nbr AND transaction_view
          ''')
```

In [ ]:

```
items_per_transaction.show()
```

```
+---------+----------+---------------------+
|store_nbr|      date|items_per_transaction|
+---------+----------+---------------------+
|        1|2017-01-02|    10.89927518148293|
|        1|2017-01-03|    7.673864851875148|
|        1|2017-01-04|    8.586256167065754|
|        1|2017-01-05|    7.077579414771945|
|        1|2017-01-06|    7.201415950817795|
|        1|2017-01-07|    9.036027625511906|
|        1|2017-01-08|    11.01076522496225|
|        1|2017-01-09|    7.960172965873999|
|        1|2017-01-10|    6.963213621073039|
|        1|2017-01-11|    7.575320663429174|
|        1|2017-01-12|    6.419577613287557|
|        1|2017-01-13|   7.3566062226118385|
|        1|2017-01-14|    9.440297620845552|
|        1|2017-01-15|    11.14722532877487|
|        1|2017-01-16|    6.384192097940164|
|        1|2017-01-17|    7.384185392101425|
|        1|2017-01-18|    7.631446812136615|
|        1|2017-01-19|    6.473136167208941|
|        1|2017-01-20|    6.964179811972696|
|        1|2017-01-21|    9.353179002717567|
+---------+----------+---------------------+
only showing top 20 rows
```

In [ ]:

```
items_per_transaction.cache
```

Out[24]:

```
<bound method DataFrame.cache of DataFrame[store_nbr: int, date: date, items
_per_transaction: double]>
```

In [ ]:

```
items_per_transaction = items_per_transaction.groupby('store_nbr').pivot('date').sum('
```

In [ ]:

```
items_per_transaction = items_per_transaction.toPandas()
```

In [ ]:

```
items_per_transaction = items_per_transaction.sort_values('store_nbr').reset_index(dro
```

In [ ]:

```
items_per_transaction.head()
```

Out[37]:

	store_nbr	2017-01-01	2017-01-02	2017-01-03	2017-01-04	2017-01-05	2017-01-06	2017-01-07	2017-01-0
0	1	0.0	10.899275	7.673865	8.586256	7.077579	7.201416	9.036028	11.01076
1	2	0.0	11.458034	9.492144	9.438200	8.255399	8.825503	8.898593	11.48377
2	3	0.0	15.984335	14.377846	14.265220	11.626882	12.636792	13.991900	15.28914
3	4	0.0	13.903885	11.589372	11.222042	9.222110	10.097864	10.429400	13.52783
4	5	0.0	10.072835	8.579036	9.088492	7.200205	8.416129	8.800967	11.34000

5 rows × 228 columns

In [ ]:

```
items_per_transaction.shape
```

Out[32]:

```
(54, 228)
```

In [ ]:

```
items_per_transaction.to_csv('/content/drive/My Drive/Grocery/items_per_transaction.csv
```

Data Collection using holiday file

Holiday file does have some information, but one thing that striked was the mean sale per store on a holiday day, so what it means is we can find the mean sale per store on a holiday and find the factor that we can multiply with the final result once we have predictions ready, this is one way in which we can actually use information from this file..... Simplest way is to create vectors from categorical data with information of normal day and holiday day, or create a binary feature if holiday 1 if not 0, then the model will train with this information, but for now collecting data based on the logic of mean sales per store.....

In [ ]:

```
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
```

In [ ]:

```
sales_per_store_holiday_type = spark.sql('''SELECT date, type, sum(total_sales)/ count
         (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, CASE WHEN unit_sales < 0 THEN 0 ELSE LOG(unit_sal
              (SELECT date, type FROM holidays_data WHERE date >= \'2017-01-01\' AND d
                GROUP BY date, store_nbr, type ORDER BY date)
                  GROUP BY date, type ORDER BY date
              ''')
```

In [ ]:

```
sales_per_store_holiday_type = sales_per_store_holiday_type.groupby('type').pivot('dat
```

In [ ]:

```
sales_per_store_holiday_type.head()
```

Out[14]:

	type	2017-01-01	2017-01-02	2017-02-27	2017-02-28	2017-03-02	2017-04-01	20
0	Event	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	
1	Holiday	2507.591043	0.000000	3104.925873	3559.93742	3422.307517	4469.905208	30
2	Transfer	0.000000	4299.217933	0.000000	0.00000	0.000000	0.000000	
3	Additional	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	

In [ ]:

```
sales_per_store_holiday_type.to_csv('/content/drive/My Drive/Grocery/sales_per_store_h
```

In [ ]:

```
train_sales_df = pd.read_csv('/content/drive/My Drive/Grocery/train_sales.csv')
final_promo_df = pd.read_csv('/content/drive/My Drive/Grocery/final_promo.csv')
items_sales_df = pd.read_csv('/content/drive/My Drive/Grocery/items_sales.csv')
items_promo_df = pd.read_csv('/content/drive/My Drive/Grocery/items_promo.csv')
store_sales_df = pd.read_csv('/content/drive/My Drive/Grocery/store_sales.csv')
store_promo_df = pd.read_csv('/content/drive/My Drive/Grocery/store_promo.csv')
store_class_sales_df = pd.read_csv('/content/drive/My Drive/Grocery/store_class_sales.
store_class_promo_df = pd.read_csv('/content/drive/My Drive/Grocery/store_class_promo.
items_per_transaction = pd.read_csv('/content/drive/My Drive/Grocery/items_per_transac
test_df = pd.read_csv('/content/drive/My Drive/Grocery/test.csv')
items_df = pd.read_csv('/content/drive/My Drive/Grocery/items.csv')
```

In [ ]:

```
test_df['date'] = pd.to_datetime(test_df['date'])
```

#Baseline Model

Let us start with building simplest of model as our baseline model. So, we are doing here is building a model that takes item-store-date information and forecasts result based on 16 week moving average, if we don't have item-store information in test, it uses sales from class-storeand if we don't have even class-store information on our data collection df, it gives results based on the classes that item belong to.

In [ ]:

```
def generate_baseline_forecast(df, forecast_date):
  '''
  This function takes the df and the forecast_date, calculates the step size
  and based on that generates average with a 16 datapoints window
  '''
  arry = df.values.reshape(-1,1)[-15:]
  step = ( datetime.datetime.date(forecast_date) - datetime.date(2017, 8, 15)).days
  for i in range(step):
    avg = np.mean(arry)
    arry = np.append(arry, avg)
    arry = arry[1:]

  return avg
```

In [ ]:

```
def moving_average(store, item, forecast_date):
  '''
  This function checks for the level which will be used for moving window average
  '''
  df = item_store_sales_df[(item_store_sales_df['store_nbr'] == store) & (item_store_s

  if df.shape[0] == 0:
    return 0
  else:
    return np.expm1(generate_baseline_forecast(df, forecast_date))
```

In [ ]:

```
%%time
test_df['unit_sales'] = test_df.apply(lambda x: moving_average(x.store_nbr, x.item_nbr
```

```
CPU times: user 1h 13min 42s, sys: 2.24 s, total: 1h 13min 45s
Wall time: 1h 13min 45s
```

In [ ]:

```
test_df.head()
```

Out[8]:

	id	date	store_nbr	item_nbr	onpromotion	unit_sales
0	125497040	2017-08-16	1	96995	False	0.366349
1	125497041	2017-08-16	1	99197	False	0.212509
2	125497042	2017-08-16	1	103501	False	0.000000
3	125497043	2017-08-16	1	103520	False	0.873407
4	125497044	2017-08-16	1	103665	False	1.915161

In [ ]:

```
test_df[test_df.isnull().any(axis = 1)]['item_nbr'].unique()
```

Out[9]:

```
array([], dtype=int64)
```

In [ ]:

```
#Creating a submission file for kaggle submission to get a score based on the baseline
test_df[['id', 'unit_sales']].to_csv('baseline_16_submission.csv', index = False)
```

After making submission from the results generated on test using the baseline model, score was .59249 on private, this score we will consider as a base score for other models, we need to create models that will improve the score than this.....

#Data Preparation

imports

In [5]:

```
import pandas as pd
import numpy as np
import datetime
from datetime import date
from datetime import timedelta
import calendar
import math
import time
from tqdm import tqdm
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBRegressor
import xgboost as xgb
from joblib import dump, load
from sklearn import preprocessing
import category_encoders as ce
```

Loading files

In [6]:

```
item_store_sales_df = pd.read_csv('/content/drive/My Drive/Grocery/train_sales.csv')
item_store_promo_df = pd.read_csv('/content/drive/My Drive/Grocery/final_promo.csv')
#items_sales_df = pd.read_csv('/home/jupyter/final_files/items_sales.csv')
#items_promo_df = pd.read_csv('/home/jupyter/final_files/items_promo.csv')
#store_sales_df = pd.read_csv('/home/jupyter/final_files/store_sales.csv')
#store_promo_df = pd.read_csv('/home/jupyter/final_files/store_promo.csv')
#store_class_sales_df = pd.read_csv('/home/jupyter/final_files/store_class_sales.csv')
#store_class_promo_df = pd.read_csv('/home/jupyter/final_files/store_class_promo.csv')
#items_per_transaction_df = pd.read_csv('/home/jupyter/final_files/items_per_transacti
test_df = pd.read_csv('/content/drive/My Drive/Grocery/test.csv')
items_df = pd.read_csv('/content/drive/My Drive/Grocery/items.csv')
stores_df = pd.read_csv('/content/drive/My Drive/Grocery/stores.csv')
```

The data collected at item, store, store-class level was not used as it was adding noise to all the models, and using them score got impacted, even transaction data was not used for final modelling.

Approcah to solve the problem:

1. First point that we need to understand is how our test file looks like, so if we take dates in columns we will see that for an item-store combination, we need to make prediction for date from 16th Aug till 31st Aug, so we can say that we need to make prediction from t+1 till t+16 time stamp using the date we have.....
2. Let us collect X_i such that the next 16 dates can be thought of as our Y_i, so we can use historical data to generate X_i & next 16 intervals as Y_i(unit_sales), train our model using this and make predictions on test.
3. How to use this, let say X_i is a matrix of [m x n] & Y_i is a matrix of [m x 16], for each Y_i or the step that Y_i represents, train using X_i, i.e, at each iteration we will use X_i[m x n] and Y_i[m x 1] and make prediction using the same, so, for 1st set of Y_i, we will use X_i and get step1 forecast, for 2nd set of Y_i, we will again use the same X_i and get step2 forecast, and so on till 16th step. Our resultant prediction will also be a vector of [m x 16], where m is the number of data points(item-store level), and 16 are the date ranging from 16th Aug till 31st Aug.

Generating df for categorical features

In [7]:

```
stores_df.head()
```

Out[7]:

	store_nbr	city	state	type	cluster
0	1	Quito	Pichincha	D	13
1	2	Quito	Pichincha	D	13
2	3	Quito	Pichincha	D	8
3	4	Quito	Pichincha	D	9
4	5	Santo Domingo	Santo Domingo de los Tsachilas	D	4

In [9]:

```
class_family_df = pd.DataFrame(item_store_sales_df['item_nbr']).merge(items_df[['item_
class_family_df['class'] = class_family_df['class'].astype('str')
class_family_df['item_nbr'] = class_family_df['item_nbr'].astype('str')
class_family_df.head()
```

Out[9]:

	item_nbr	class	family	perishable
0	96995	1093	GROCERY I	0
1	99197	1067	GROCERY I	0
2	103520	1028	GROCERY I	0
3	103665	2712	BREAD/BAKERY	1
4	105574	1045	GROCERY I	0

In [10]:

```
store_detail_df = pd.DataFrame(item_store_sales_df['store_nbr']).merge(stores_df[['sto
store_detail_df['store_nbr'] = store_detail_df['store_nbr'].astype('str')
store_detail_df['cluster'] = store_detail_df['cluster'].astype('str')
store_detail_df.head()
```

Out[10]:

	store_nbr	state	city	type	cluster
0	1	Pichincha	Quito	D	13
1	1	Pichincha	Quito	D	13
2	1	Pichincha	Quito	D	13
3	1	Pichincha	Quito	D	13
4	1	Pichincha	Quito	D	13

Y_train:

Y_cv:

Y_test:

Helper Functions

In [11]:

```
def cat_encoding(cat_data, category):
  '''
  This function takes a df and the category and generate
  binary encoded vectors for the same
  '''
  encoder = ce.BinaryEncoder()
  return encoder.fit_transform(cat_data[category]).values
```

In [12]:

```
#Generating binary encoded vector for categories part of item table
class_array = cat_encoding(class_family_df, 'class')
family_array = cat_encoding(class_family_df, 'family')
item_array = cat_encoding(class_family_df, 'item_nbr')
```

In [13]:

```
print(class_array.shape, family_array.shape, item_array.shape)
```

```
(167515, 10) (167515, 7) (167515, 13)
```

In [14]:

```
store_detail_df.head()
```

Out[14]:

	store_nbr	state	city	type	cluster
0	1	Pichincha	Quito	D	13
1	1	Pichincha	Quito	D	13
2	1	Pichincha	Quito	D	13
3	1	Pichincha	Quito	D	13
4	1	Pichincha	Quito	D	13

In [15]:

```
# Generating binary encoded vectors for category part of store table
store_array = cat_encoding(store_detail_df, 'store_nbr')
store_state_array = cat_encoding(store_detail_df, 'state')
store_city_array = cat_encoding(store_detail_df, 'city')
store_type_array = cat_encoding(store_detail_df, 'type')
store_cluster_array = cat_encoding(store_detail_df, 'cluster')
```

In [16]:

```
print(store_array.shape, store_state_array.shape, store_city_array.shape, store_type_ar
```

```
(167515, 7) (167515, 5) (167515, 6) (167515, 4) (167515, 6)
```

In [17]:

```
store_array
```

Out[17]:

```
array([[0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 1],
       ...,
       [0, 1, 1, ..., 1, 1, 0],
       [0, 1, 1, ..., 1, 1, 0],
       [0, 1, 1, ..., 1, 1, 0]])
```

In [18]:

```
def get_data(data, dt_end, days, period, freq='D'):
  '''
  This function gives us the selected columns based on a range of dates passed.
  '''
  return data[[str(col)[0:10] for col in pd.date_range(dt_end - datetime.timedelta(day
```

In [19]:

```
def average(data):
  '''
  Here we are calculating simple average
  '''
  return np.mean(data, axis = 1)
```

In [20]:

```
def weighted_moving_average(data):
  '''
  This function computes weighted moving average,
  higher weights are given to recent observations.
  '''
  data = data.values
  weight_len = data.shape[1]
  denom = (weight_len *(weight_len + 1))/2
  weights = [i+1/denom for i in range(weight_len)]
  data = average(data * weights)
  return data
```

In [21]:

```
#This was excluded from the final features, as this was of no use to the models.
def expo_smoothing(data_row, alpha = 0.7):
  '''
  This function gives us the exponential smoothing compoenent of our time series.
  '''
  values = [data_row[0]]
  for i in range(len(data_row)):
    values.append(alpha * data_row[i] + (1 - alpha) * values[i - 1])
  return values
```

In [22]:

```
#These functions were excluded from the model, here we are calculating
#triple exponential smoothing, also known as Holt's Winter technique.
#Ref: https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc435.htm
def trend_component(data_row, season_len):
  sum = 0
  for i in range(season_len):
    sum += (data_row[i + season_len] - data_row[i])/season_len
  return sum/season_len

def seasonal_components(data_row, season_len):
  n_seasons = int(len(data_row)/season_len) #Total number of seasons in our series
  #next we find the average value of each season, let say if we have 70 data points wi
  # have total 10 season, so, for each of these seasons we find the average value
  average = [sum(data_row[i*season_len:(i*season_len) + season_len])/season_len for i i
  #print(average)
  #The computed average will be subtracted from the appropriate season and we will take

  dict_season = {i: sum([data_row[season_len * j + i] - average[j] for j in range(n_sea
  #print(dict_season)
  return dict_season

def triple_expo_smoothing(data_row, season_len = 7, alpha = 0.7 , beta = 0.4, gamma = (
  result = []
  trend = trend_component(data_row, season_len) #Initial Trend
  seasonal_component = seasonal_components(data_row, season_len) #Initial Seasonal comp
  #print(seasonal_component)
  for i in range(len(data_row)):
    if i == 0:
      smooth = data_row[0]
      result.append(data_row[0])
      continue
    value = data_row[i]
    pre_smooth, smooth = smooth, alpha*(value - seasonal_component[i % season_len]) +
    trend = beta * (smooth - pre_smooth) + (1- beta) * trend #Trend Smoothing
    seasonal_component[i % season_len] = gamma * (value - smooth) + (1 - gamma) * seas
    result.append(smooth + trend + seasonal_component[i % season_len])
  return result
```

In [23]:

```
def feature_engg_sales(data, end_date, prefix):
  '''
  This function generates feature dictionary for train, cv, test
  Features generated are:
  moving average, weighted moving average, standard deviation observed,
  moving average of DOW, weighted moving average of DOW, having total sales day,
  last sales day in n days, first sales day in n days
  '''
  days_list = [3, 7, 16, 30, 60, 120] # These are the list of days used for extracting
  #feature_dict = {}
  feature_dict = {'{}_average_{}_days'.format(prefix, days): average(get_data(data, en
  feature_dict.update({'{}_WMA_{}_days'.format(prefix, days): weighted_moving_average(
  #feature_dict.update({'{}_average_diff_{}_days'.format(prefix, days) : get_data(data,
  #feature_dict.update({'{}_max_{}_days'.format(prefix, days) : get_data(data, end_date
  feature_dict.update({'{}_std_{}_days'.format(prefix, days) : get_data(data, end_date
  feature_dict.update({'{}_6avgdow_{}_days'.format(prefix, day) : get_data(data, end_d
  feature_dict.update({'{}_20avgdow_{}_days'.format(prefix, day) : get_data(data, end_
  feature_dict.update({'{}_6WMAdow_{}_days'.format(prefix, day) : weighted_moving_aver
  feature_dict.update({'{}_20WMAdow_{}_days'.format(prefix, day) : weighted_moving_ave
  feature_dict.update({'{}_has_sale_day_{}'.format(prefix, days) : (get_data(data, end
  feature_dict.update({'{}_last_has_sale_day_{}'.format(prefix, days) : days - ((get_d
  feature_dict.update({'{}_first_has_sale_day_{}'.format(prefix, days) : ((get_data(da


  #feature_dict.update({'{}_lastday'.format(prefix) : get_data(data, end_date, 1, 1).v
  #feature_dict.update({'{}_day_{}'.format(prefix, day) : get_data(data, end_date, day

  #exponential smoothing: smoothing 16 days data point with a smoothing factor of 0.7
  #df = get_data(data, end_date, 16, 16)
  #expo_arry = np.array([expo_smoothing(df.iloc[i])[1:] for i in range(df.shape[0])])
  #feature_dict.update({'expo_smooth_{}'.format(col_num): expo_arry[:, col_num] for co

  #Triple Exponential Smoothing(Holt's Winter)
  #df = get_data(data, end_date, 35, 35)
  #holt_winter_arry = np.array([triple_expo_smoothing(df.iloc[i]) for i in range(df.sh
  #feature_dict.update({'holt_winter_{}'.format(col_num): holt_winter_arry[:, col_num]

  return feature_dict
```

In [24]:

```
def feature_engg_promo(data, class_array, store_array, end_date, prefix):
    '''
    This function uses promo information and categorical array to create features
    features created are---
    promo: total_promo, future promo information, promo days in 15 days, last promo in
    categorical: class, item, store, family, city, state, clsuter, type
    '''
    days_list = [16, 30, 60, 120]
    feature_dict = {'{}_totalpromo_{}_days'.format(prefix, days) : get_data(data, end_
    feature_dict.update({'{}_totalpromoafter_{}_days'.format(prefix, days) : get_data(
   # if prefix in ['item', 'store_class']:
   #     feature_dict.update({'{}_maxnopromo_{}_days'.format(prefix, days) : get_data(
   #     feature_dict.update({'{}_maxnopromoafter_{}_days'.format(prefix, days) : get_
    feature_dict.update({'{}_promo_{}_day'.format(prefix, abs(day - 1)): get_data(data
    feature_dict.update({'promo_day_in_15_days' : (get_data(data, end_date + timedelta
    feature_dict.update({'last_promo_day_in_15_days' : 15 - ((get_data(data, end_date 
    feature_dict.update({'firt_promo_day_in_15_days' : ((get_data(data, end_date + tim
    feature_dict.update({'class_{}'.format(i+1) : class_array[:, i] for i in range(cla
    feature_dict.update({'item_{}'.format(i+1) : item_array[:, i] for i in range(item_
    feature_dict.update({'store_{}'.format(i+1) : store_array[:, i] for i in range(sto
    feature_dict.update({'family_{}'.format(i+1) : family_array[:, i] for i in range(f
    feature_dict.update({'city_{}'.format(i+1) : store_city_array[:, i] for i in range
    feature_dict.update({'state_{}'.format(i+1) : store_state_array[:, i] for i in ran
    feature_dict.update({'cluster_{}'.format(i+1) : store_cluster_array[:, i] for i in
    feature_dict.update({'type_{}'.format(i+1) : store_type_array[:, i] for i in range
    feature_dict.update({'perishable' : class_family_df['perishable'].values})
    #feature_dict.update({'class_{}'.format(i + 1) : class_vector.toarray()[:, i] for 
    #feature_dict.update({'{}_promo_{}_day'.format(prefix, day - 1): get_data(data, en

    return feature_dict
```

Preparing Train Data

In [25]:

```
#To create training points we will take multiple intervals and will concat all the inf
x_lst, y_lst = [], []
num_of_intervals = 8
dates = [date(2017, 5, 31) + timedelta(days=7 * interval) for interval in range(num_of_
for train_date in tqdm(dates):
  train_dict = feature_engg_sales(item_store_sales_df, train_date,'item_store')
  x_lst.append(pd.DataFrame(train_dict, index = [i for i in range(len(list(train_dict.
  y_lst.append(item_store_sales_df[[str(col)[0:10] for col in pd.date_range(train_date

train_item_store_x = pd.concat(x_lst, axis=0)
train_y = np.concatenate(y_lst, axis=0)
del x_lst, y_lst
print(train_item_store_x.shape, train_y.shape)
```

```
100%|██████████| 8/8 [00:18<00:00,  2.34s/it]

(1340120, 64) (1340120, 16)
```

In [26]:

```
train_item_store_x.head()
```

Out[26]:

	item_store_average_3_days	item_store_average_7_days	item_store_average_16_days	item_sto
0	0.231049	0.297063	0.129965	
1	0.597253	0.610952	0.585266	
2	0.000000	0.824046	0.728402	
3	0.366204	0.709973	0.939201	
4	1.059351	1.403121	1.648731	

In [27]:

```
x_lst = []
num_of_intervals = 8
dates = [date(2017, 5, 31) + timedelta(days=7 * interval) for interval in range(num_of_
for train_date in tqdm(dates):
  train_dict = feature_engg_promo(item_store_promo_df, class_array, store_array, train_
  x_lst.append(pd.DataFrame(train_dict, index = [i for i in range(len(list(train_dict.

train_item_store_x1 = pd.concat(x_lst, axis=0)
del x_lst
print(train_item_store_x1.shape)
```

```
100%|██████████| 8/8 [00:02<00:00,  2.74it/s]

(1340120, 85)
```

In [28]:

```
train_item_store_x1.head()
```

Out[28]:

	item_store_totalpromo_16_days	item_store_totalpromo_30_days	item_store_totalpromo_60_day
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	0	2

5 rows × 85 columns

In [29]:

```
train_x = train_item_store_x.reset_index(drop = True).merge(train_item_store_x1.reset_
```

In [30]:

```
[train_x[col].update((train_x[col] - train_x[col].min()) / (train_x[col].max() - train_
```

Out[30]:

```
[None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None.
```

In [31]:

```
train_x.head()
```

Out[31]:

	item_store_average_3_days	item_store_average_7_days	item_store_average_16_days	item_sto
0	0.029921	0.043809	0.019646	
1	0.077346	0.090100	0.088472	
2	0.000000	0.121526	0.110109	
3	0.047424	0.104703	0.141975	
4	0.137189	0.206924	0.249231	

5 rows × 149 columns

In [32]:

```
print('Shape of train_x and corresponding train_y is {} & {}'.format(train_x.shape, tr
```

```
Shape of train_x and corresponding train_y is (1340120, 149) & (1340120, 16)
```

Preparing CV Data

In [34]:

```
#Generating sales features
cv_date = date(2017, 7, 26)
cv_dict = feature_engg_sales(item_store_sales_df, cv_date, 'item_store')
cv_item_store_x = pd.DataFrame(cv_dict, index = [i for i in range(len(list(cv_dict.valu
cv_item_store_x.shape
```

Out[34]:

(167515, 64)

In [35]:

```
cv_item_store_x.head()
```

Out[35]:

	item_store_average_3_days	item_store_average_7_days	item_store_average_16_days	item_sto
0	0.000000	0.354987	0.155307	
1	0.000000	0.610952	0.664548	
2	1.059351	0.850092	0.804148	
3	1.229626	0.881969	0.902465	
4	1.866141	1.892588	1.820677	

In [36]:

```
#Generating promo and categorical features
cv_dict = feature_engg_promo(item_store_promo_df, class_array, store_array, cv_date, '
cv_item_store_x1 = pd.DataFrame(cv_dict, index = [i for i in range(len(list(cv_dict.va
cv_item_store_x1.shape
```

Out[36]:

(167515, 85)

In [37]:

```
cv_item_store_x1.head()
```

Out[37]:

	item_store_totalpromo_16_days	item_store_totalpromo_30_days	item_store_totalpromo_60_day
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	0	0	

5 rows × 85 columns

In [38]:

```
#Merging all the data points
cv_x = cv_item_store_x.reset_index(drop = True).merge(cv_item_store_x1.reset_index(dro
```

In [39]:

```
[cv_x[col].update((cv_x[col] - cv_x[col].min()) / (cv_x[col].max() - cv_x[col].min()))
```

Out[39]:

```
[None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
```

In [40]:

```
cv_x.head()
```

Out[40]:

	item_store_average_3_days	item_store_average_7_days	item_store_average_16_days	item_sto
0	0.000000	0.054309	0.024038	
1	0.000000	0.093470	0.102859	
2	0.154317	0.130056	0.124467	
3	0.179121	0.134933	0.139684	
4	0.271843	0.289547	0.281806	

5 rows × 149 columns

In [41]:

```
print('Shape of train_x and corresponding train_y is {}'.format(cv_x.shape))
```

```
Shape of train_x and corresponding train_y is (167515, 149)
```

In [42]:

```
#Generating y_i for cv
cv_y = item_store_sales_df[[str(col)[0:10] for col in pd.date_range(cv_date, periods =
```

Preparing test data

In [43]:

```
#gathering sales featres
test_date = date(2017, 8, 16)
test_dict = feature_engg_sales(item_store_sales_df, test_date, 'item_store')
test_item_store_x = pd.DataFrame(test_dict, index = [i for i in range(len(list(test_di
test_item_store_x.shape
```

Out[43]:

```
(167515, 64)
```

In [44]:

```
test_item_store_x.head()
```

Out[44]:

	item_store_average_3_days	item_store_average_7_days	item_store_average_16_days	item_sto
0	0.000000	0.099021	0.361296	
1	0.000000	0.156945	0.180648	
2	0.231049	0.495105	0.631845	
3	0.462098	0.980990	1.071718	
4	0.998577	1.560437	1.663453	

In [45]:

```
test_dict = feature_engg_promo(item_store_promo_df, class_array, store_array, test_dat
test_item_store_x1 = pd.DataFrame(test_dict, index = [i for i in range(len(list(test_d
test_item_store_x1.shape
```

Out[45]:

```
(167515, 85)
```

In [46]:

```
test_item_store_x1.shape
```

Out[46]:

```
(167515, 85)
```

In [47]:

```
test_x = test_item_store_x.reset_index(drop = True).merge(test_item_store_x1.reset_ind
```

In [48]:

```
[test_x[col].update((test_x[col] - test_x[col].min()) / (test_x[col].max() - test_x[co
```

Out[48]:

```
[None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
```

In [49]:

```
test_x.shape
```

Out[49]:

```
(167515, 149)
```

In [50]:

```
print('Shape of train_x and corresponding train_y is {}'.format(test_x.shape))
```

```
Shape of train_x and corresponding train_y is (167515, 149)
```

In [51]:

```
print(train_x.shape, train_y.shape)
print(cv_x.shape, cv_y.shape)
print(test_x.shape)
```

```
(1340120, 149) (1340120, 16)
(167515, 149) (167515, 16)
(167515, 149)
```

#Modelling

Linear Regression

We have 16 steps to predict and we have collected our y such that it is a vector of Mx16, so we will train x for each of these y and based on the result for every y we will generate the forecast

In [ ]:

```
test_pred = []
for i in range(train_y.shape[1]):
    print('step{}'.format(i+1))
    lr = LinearRegression()
    lr.fit(train_x, train_y[: , i])
    test_pred.append(lr.predict(test_x))
```

```
step1
step2
step3
step4
step5
step6
step7
step8
step9
step10
step11
step12
step13
step14
step15
step16
```

In [ ]:

```
#Creating prediction df
y_test = np.array(test_pred).transpose()
pred_df = pd.DataFrame(y_test, columns=pd.date_range("2017-08-16", periods=16))
```

In [ ]:

```
pred_df.head()
```

Out[40]:

	2017-08-16	2017-08-17	2017-08-18	2017-08-19	2017-08-20	2017-08-21	2017-08-22	2017-08-23	2017-08-24	2
0	0.210249	0.223341	0.289090	0.270891	0.210922	0.250148	0.215341	0.262398	0.232133	0
1	0.317271	0.300324	0.379285	0.318358	0.118592	0.314419	0.322006	0.400568	0.322340	0
2	0.714128	0.769529	0.864926	0.703121	0.287471	0.621062	0.693032	0.759970	0.796433	0
3	1.016627	0.975421	1.213480	1.213211	0.711763	0.852536	0.844865	0.994915	0.971457	1
4	1.903333	1.759549	1.925219	1.665465	1.139938	1.764955	1.755045	1.879388	1.731188	1

In [ ]:

```
item_store_sales_df['store_nbr'] = pd.to_numeric(item_store_sales_df['store_nbr'])
items_df['class'] = pd.to_numeric(items_df['class'])
```

In [ ]:

```
#Melting down the predicted values based on dates
pred_df = item_store_sales_df[['item_nbr', 'store_nbr']].merge(pred_df, left_index=Tru
pred_df = pred_df.melt(id_vars=['item_nbr', 'store_nbr'], var_name='date', value_name=
pred_df = pred_df.merge(items_df[['item_nbr', 'class']], how = 'left', on = 'item_nbr'
pred_df['unit_sales'] = pred_df['unit_sales'].apply(lambda x : np.expm1(x))
```

In [ ]:

```
pred_df.head()
```

Out[43]:

	item_nbr	store_nbr	date	unit_sales	class
0	96995	1	2017-08-16	0.233986	1093
1	99197	1	2017-08-16	0.373375	1067
2	103520	1	2017-08-16	1.042405	1028
3	103665	1	2017-08-16	1.763855	2712
4	105574	1	2017-08-16	5.708217	1045

In [ ]:

```
#Reading test_file
test_df = pd.read_csv('test.csv')
test_df['date'] = pd.to_datetime(test_df['date'])
```

In [ ]:

```
#Merging with the predicted values
test_df = test_df.merge(pred_df[['item_nbr', 'store_nbr', 'date', 'unit_sales']], on =
test_df['unit_sales'] = test_df['unit_sales'].clip(lower = 0)
#Filling null values with 0
test_df = test_df.fillna(0)
#Making submission file
test_df[['id', 'unit_sales']].to_csv('lr_submission.csv', index = False)
```

In [ ]:

```
test_df[['id', 'unit_sales']].head()
```

Out[46]:

	id	unit_sales
0	125497040	0.233986
1	125497041	0.373375
2	125497042	0.000000
3	125497043	1.042405
4	125497044	1.763855

In [ ]:

```
del test_df, pred_df
```

Since we know that perishable items have more weights in our scoring method as compared to non perishable, hence we are creating a weight vector with perishable items having a weight of 1.25 and others having a weight of 1, this will be used by XGBoost to give more efforts with items with higher weights.

In [52]:

```
train_weights = pd.concat([pd.DataFrame(item_store_sales_df['item_nbr']).merge(items_d
cv_weights = pd.DataFrame(item_store_sales_df['item_nbr']).merge(items_df[['item_nbr',
```

In [ ]:

```
train_weights.shape, cv_weights.shape
```

Out[49]:

```
((1340120,), (167515,))
```

In [53]:

```
train_weights.head()
```

Out[53]:

```
0    1.00
1    1.00
2    1.00
3    1.25
4    1.00
Name: perishable, dtype: float64
```

In [54]:

```
cv_weights.head()
```

Out[54]:

```
0    1.00
1    1.00
2    1.00
3    1.25
4    1.00
Name: perishable, dtype: float64
```

XGBoost without tuned parameters

In [ ]:

```
test_pred = []
for i in range(train_y.shape[1]):
    print('step{}'.format(i+1))
    start_time = time.time()
    xg = XGBRegressor()
    xg.fit(train_x, train_y[: , i], sample_weight = train_weights.values)
    test_pred.append(xg.predict(test_x))
    print('done in {}'.format(time.time() - start_time))
```

```
step1
done in 331.6978657245636
step2
done in 343.8066828250885
step3
done in 342.00068831443787
step4
done in 340.25216579437256
step5
done in 340.5731554031372
step6
done in 339.94398260116577
step7
done in 340.4451413154602
step8
done in 341.4083557128906
step9
done in 340.8268074989319
step10
done in 341.2782769203186
step11
done in 340.4570393562317
step12
done in 340.13201689720154
step13
done in 340.543958902359
step14
done in 339.9647686481476
step15
done in 339.2215938568115
step16
done in 341.0175998210907
```

In [55]:

```
#Generating prediction df
y_test = np.array(test_pred).transpose()
pred_df = pd.DataFrame(y_test, columns=pd.date_range("2017-08-16", periods=16))
pred_df.head(10)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-55-488b21904765> in <module>()
----> 1 y_test = np.array(test_pred).transpose()
      2 pred_df = pd.DataFrame(y_test, columns=pd.date_range("2017-08-16", p
eriods=16))
      3 pred_df.head(10)

NameError: name 'test_pred' is not defined
```

In [ ]:

```
item_store_sales_df['store_nbr'] = pd.to_numeric(item_store_sales_df['store_nbr'])
items_df['class'] = pd.to_numeric(items_df['class'])
```

In [ ]:

```
#Melting based on dates and adding other columns
pred_df = item_store_sales_df[['item_nbr', 'store_nbr']].merge(pred_df, left_index=Tru
pred_df = pred_df.melt(id_vars=['item_nbr', 'store_nbr'], var_name='date', value_name=
pred_df = pred_df.merge(items_df[['item_nbr', 'class']], how = 'left', on = 'item_nbr'
pred_df['unit_sales'] = pred_df['unit_sales'].apply(lambda x : np.expm1(x))
pred_df.head()
```

Out[43]:

	item_nbr	store_nbr	date	unit_sales	class
0	96995	1	2017-08-16	0.268535	1093
1	99197	1	2017-08-16	0.302717	1067
2	103520	1	2017-08-16	1.264865	1028
3	103665	1	2017-08-16	2.325738	2712
4	105574	1	2017-08-16	6.268309	1045

In [ ]:

```
#Loading test file
test_df = pd.read_csv('test.csv')
test_df['date'] = pd.to_datetime(test_df['date'])
```

In [ ]:

```
#Merging with predicted results and saving submission file
test_df = test_df.merge(pred_df[['item_nbr', 'store_nbr', 'date', 'unit_sales']], on =
test_df['unit_sales'] = test_df['unit_sales'].clip(lower = 0)
test_df = test_df.fillna(0)
test_df[['id', 'unit_sales']].to_csv('xg_submission.csv', index = False)
```

In [ ]:

```
test_df[['id', 'unit_sales']].head(10)
```

Out[46]:

	id	unit_sales
0	125497040	0.268535
1	125497041	0.302717
2	125497042	0.000000
3	125497043	1.264865
4	125497044	2.325738
5	125497045	6.268309
6	125497046	13.093959
7	125497047	0.000000
8	125497048	0.668626
9	125497049	0.324347

In [ ]:

```
del test_df, pred_df
```

hyperparameter tuning using RandomizedSearchCv

In [ ]:

```
def random_search(x, y, x_cv, y_cv):
    '''
    This function is called during each step and it returns best parameter and best es
    '''
    params = {'max_depth' : [2, 4, 6, 8, 10],
          'learning_rate' : [0.1, 0.2, 0.3],
          'n_estimators' : [5, 10, 50, 100]
        }
    clf = XGBRegressor(objective = 'reg:squarederror', eval_metric = 'rmse')
    rv = RandomizedSearchCV(clf, param_distributions = params, n_iter = 8, scoring = '
    rv.fit(x, y)
    return rv.best_estimator_, rv.best_params_
```

In [ ]:

```
#Tuning parameter and saving the best estimator and parameters
test_pred = []
for i in range(train_y.shape[1]):
    print('step{}'.format(i+1))
    xg, best_params = random_search(train_x, train_y[:, i], cv_x, cv_y[:, i])
    dump(xg, 'clf_step_{}.joblib'.format(i+1))
    dump(best_params, 'para_step_{}.joblib'.format(i+1))
```

```
step1
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 11.7min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 31.0min finished

step2
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 45.8min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 95.6min finished

step3
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:  4.8min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 85.3min finished

step4
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 16.8min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 80.1min finished

step5
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 38.2min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 69.5min finished

step6
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 70.3min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 106.7min finished

step7
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 64.5min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 104.6min finished

step8
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
```

```
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 16.6min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 44.7min finished

step9
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 21.1min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 37.6min finished

step10
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 42.8min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 88.6min finished

step11
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:  5.7min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 38.4min finished

step12
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 14.1min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 85.5min finished

step13
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 17.2min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 37.4min finished

step14
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 15.6min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 66.6min finished

step15
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 32.9min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 75.6min finished

step16
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:  9.1min
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 53.9min finished
```

Using XGBoost with tuned parameter to generate final results

In [ ]:

```
def load_param(step):
    '''
    This function loads the best parameters used by XGBoost
    '''
    best_param = load('/home/jupyter/final_para/para_step_{}.joblib'.format(step))
    print(best_param)
    params = {}
    params['objective'] = 'reg:squarederror'
    params['eval_metric'] = 'rmse'
    params['eta'] = 0.02
    #params['n_estimators'] = best_param['n_estimators']
    params['max_depth'] = best_param['max_depth']
    params['learning_rate'] = best_param['learning_rate']

    return params
```

In [ ]:

```
import xgboost as xgb
test_pred = []
dtest = xgb.DMatrix(test_x)
for i in range(train_y.shape[1]):
    param = load_param(i + 1)
    #print(param)
    print('step{}'.format(i+1))
    dtrain = xgb.DMatrix(train_x, label = train_y[:, i], weight = train_weights)
    dval = xgb.DMatrix(cv_x, label = cv_y[:, i], weight = cv_weights)

    watchlist = [(dtrain, 'train'), (dval, 'val')]
    model = xgb.train(param, dtrain, 500, watchlist, early_stopping_rounds = 20, verbo

    test_pred.append(model.predict(dtest))
```

```
{'n_estimators': 50, 'max_depth': 8, 'learning_rate': 0.1}
step1
[0]     train-rmse:1.09407      val-rmse:1.05591
Multiple eval metrics have been passed: 'val-rmse' will be used for early
stopping.

Will train until val-rmse hasn't improved in 20 rounds.
[10]    train-rmse:0.64464      val-rmse:0.61657
[20]    train-rmse:0.55929      val-rmse:0.54722
[30]    train-rmse:0.54389      val-rmse:0.54058
[40]    train-rmse:0.53921      val-rmse:0.54024
[50]    train-rmse:0.53623      val-rmse:0.53984
[60]    train-rmse:0.53388      val-rmse:0.53917
[70]    train-rmse:0.53215      val-rmse:0.53901
[80]    train-rmse:0.53080      val-rmse:0.53890
[90]    train-rmse:0.52960      val-rmse:0.53882
[100]   train-rmse:0.52858      val-rmse:0.53873
[110]   train-rmse:0.52758      val-rmse:0.53867
[120]   train-rmse:0.52655      val-rmse:0.53866
```

In [ ]:

```
#Creating prediction df
y_test = np.array(test_pred).transpose()
pred_df = pd.DataFrame(y_test, columns=pd.date_range("2017-08-16", periods=16))
pred_df.head()
```

Out[52]:

	2017-08-16	2017-08-17	2017-08-18	2017-08-19	2017-08-20	2017-08-21	2017-08-22	2017-08-23	2017-08-24	2
0	0.211937	0.176730	0.259274	0.234703	0.137470	0.206698	0.206255	0.219694	0.199748	0
1	0.318681	0.320026	0.323715	0.375148	0.145478	0.303089	0.305589	0.333401	0.275312	0
2	0.790881	0.766193	0.904757	0.819249	0.243183	0.683963	0.748915	0.821975	0.812995	0
3	1.180265	1.016764	1.299401	1.288223	0.580249	0.952050	1.048379	1.133404	0.982418	1
4	2.059678	1.869376	1.993411	1.624396	0.901049	1.838199	1.895689	2.003506	1.824216	2

In [ ]:

```
item_store_sales_df['store_nbr'] = pd.to_numeric(item_store_sales_df['store_nbr'])
items_df['class'] = pd.to_numeric(items_df['class'])
```

In [ ]:

```
#melting the predicted result based on dates
pred_df = item_store_sales_df[['item_nbr', 'store_nbr']].merge(pred_df, left_index=Tru
pred_df = pred_df.melt(id_vars=['item_nbr', 'store_nbr'], var_name='date', value_name=
pred_df = pred_df.merge(items_df[['item_nbr', 'class']], how = 'left', on = 'item_nbr'
pred_df['unit_sales'] = pred_df['unit_sales'].apply(lambda x : np.expm1(x))
pred_df.head()
```

Out[54]:

	item_nbr	store_nbr	date	unit_sales	class
0	96995	1	2017-08-16	0.236070	1093
1	99197	1	2017-08-16	0.375312	1067
2	103520	1	2017-08-16	1.205339	1028
3	103665	1	2017-08-16	2.255238	2712
4	105574	1	2017-08-16	6.843444	1045

In [ ]:

```
test_df = pd.read_csv('test.csv')
test_df['date'] = pd.to_datetime(test_df['date'])
```

In [ ]:

```
#Results merged with test file and submission file created
test_df = test_df.merge(pred_df[['item_nbr', 'store_nbr', 'date', 'unit_sales']], on =
test_df['unit_sales'] = test_df['unit_sales'].clip(lower = 0)
test_df = test_df.fillna(0)
test_df[['id', 'unit_sales']].to_csv('xg_submission2.csv', index = False)
```

In [ ]:

```
test_df[['id', 'unit_sales']].head()
```

Out[57]:

	id	unit_sales
0	125497040	0.236070
1	125497041	0.375312
2	125497042	0.000000
3	125497043	1.205339
4	125497044	2.255238

#Conclusions

In [56]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "Private Score", "Rank"]

x.add_row(["Baseline - 16days MA", .59249, 1197])
x.add_row(["Linear Regression", .53398, 728])
x.add_row(["XGBoost Regressor", .52293, 334])
x.add_row(["XGBoost with tuned parameters", .52026, 103])

print(x)
```

```
+-------------------------------+---------------+------+
|             Model             | Private Score | Rank |
+-------------------------------+---------------+------+
|      Baseline - 16days MA     |    0.59249    | 1197 |
|       Linear Regression       |    0.53398    | 728  |
|       XGBoost Regressor       |    0.52293    | 334  |
| XGBoost with tuned parameters |    0.52026    | 103  |
+-------------------------------+---------------+------+
```

#Future Scope:

1. Information from transaction file, holiday file and oil file is still not explored in the model.
2. Sales/promo features are used at item-store level, but information at item/ store/ item-class level, may give us better results, although tried using them but the model results were not great, may be if these features used differently can give better result.
3. We can use LSTM to make prediction and see if our results improve further, but for that we need to come up with the correct architecture. For now, XGBoost is doing a good job, so sticking with it.