Coraline Assignment

Choose any problem on a link below that can represent your skill in Data science by following the CRISP-DM process. https://www.kaggle.com/c/m5-forecasting-accuracy

What we are looking to evaluate:

1. Business Understanding and Problem Formulation
2. Data Understanding / EDA process
3. Modeling: Model / Feature selection
4. Model Evaluation
5. Business Impact / Insights

Project Overview

In this project, I will use modern machine learning models to forecast the sales of product in Walmart.

Section 1: Business Understanding and Problem Formulation

In Walmart, the world's largest company by revenue, Sales Forecasting is a crucial area in the field of Business Management. It helps the store retailers to maintain their Stocks according to the demand they are expecting, thus maximizing the profit and minimizing the loss of time and stock. Forecasting can drive sales by processing just-in-time orders efficiently.

The data covers stores in three US states (California, Texas, and Wisconsin) and includes item level, department, product categories, and store details. In addition, it has explanatory variables such as price, promotions, day of the week, and special events. Together, this robust dataset can be used to improve forecasting accuracy.

Thus, I addressed the objective to 2 main following questions:

- What are the main features influencing the sales of the product in Walmart?

- Is different months and weekdays affect the sales of the product?

Section 2: Data Understanding / EDA process

In [1]:

```
#import neccessary package for analysis and model
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
import seaborn as sns
import math
import warnings
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing, metrics
from sklearn.model_selection import KFold, train_test_split, GridSearchCV, cross_val_score
import datetime
import xgboost as xgb
import gc
import calendar
from sklearn.cluster import KMeans
import multiprocessing as mp
from scipy.sparse import csr_matrix,hstack
import tensorflow as tf
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import mean_squared_error
from tqdm import tqdm
import pickle
import time
from itertools import cycle

color_pal=plt.rcParams['axes.prop_cycle'].by_key()['color']
color_cycle=cycle(plt.rcParams['axes.prop_cycle'].by_key()['color'])

encoder = preprocessing.LabelEncoder()

warnings.filterwarnings("ignore")

%matplotlib inline

pd.set_option('display.max_columns', 100)
```

In [2]:

```python
# Credit: The reduce_mem_usage function has been taken from
# https://gist.github.com/tkazusa/4d9e26d403c73755edc6b77b5b053a43

def reduce_mem_usage(df):
    """ iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object and str(col_type)!= 'category':
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

    return df
```

Gather

In [3]:

```
#Gather the data from csv file

sales = pd.read_csv('data/sales_train_validation.csv')
calendar = pd.read_csv('data/calendar.csv')
prices = pd.read_csv('data/sell_prices.csv')
```

In [4]:

```
print("Shape of calender.csv",calendar.shape)
print("Shape of sell_price.csv",prices.shape)
print("Shape of sales train validation.csv",sales.shape)
```

```
Shape of calender.csv (1969, 14)
Shape of sell_price.csv (6841121, 4)
Shape of sales train validation.csv (30490, 1919)
```

In [5]:

```
prices.head()
```

Out[5]:

	store_id	item_id	wm_yr_wk	sell_price
0	CA_1	HOBBIES_1_001	11325	9.58
1	CA_1	HOBBIES_1_001	11326	9.58
2	CA_1	HOBBIES_1_001	11327	8.26
3	CA_1	HOBBIES_1_001	11328	8.26
4	CA_1	HOBBIES_1_001	11329	8.26

Calendar dataset

In [6]:

```
calendar.head()
```

Out[6]:

	date	wm_yr_wk	weekday	wday	month	year	d	event_name_1	event_type_1	event_name_2	event_type_
0	2011-01-29	11101	Saturday	1	1	2011	d_1	NaN	NaN	NaN	Na
1	2011-01-30	11101	Sunday	2	1	2011	d_2	NaN	NaN	NaN	Na
2	2011-01-31	11101	Monday	3	1	2011	d_3	NaN	NaN	NaN	Na
3	2011-02-01	11101	Tuesday	4	2	2011	d_4	NaN	NaN	NaN	Na
4	2011-02-02	11101	Wednesday	5	2	2011	d_5	NaN	NaN	NaN	Na

In [7]:

```
print('event_name_1 types for products:\n')
print(calendar.event_name_1.value_counts())

print('\nevent_type_1 types for products:\n')
print(calendar.event_type_1.value_counts())

print('\nevent_name_2 types for products:\n')
print(calendar.event_name_2.value_counts())

print('\nevent_type_2 types for products:\n')
print(calendar.event_type_2.value_counts())
```

```
event_name_1 types for products:

SuperBowl              6
Pesach End             6
Ramadan starts         6
ValentinesDay          6
NBAFinalsEnd           6
NBAFinalsStart         6
MemorialDay            6
Mother's day           6
Purim End              6
StPatricksDay          6
LentWeek2              6
LentStart              6
PresidentsDay          6
MartinLutherKingDay    5
OrthodoxChristmas      5
EidAlAdha              5
NewYear                5
Chanukah End           5
Christmas              5
Thanksgiving           5
VeteransDay            5
IndependenceDay        5
Halloween              5
ColumbusDay            5
LaborDay               5
Eid al-Fitr            5
Cinco De Mayo          5
OrthodoxEaster         5
Easter                 5
Father's day           4
Name: event_name_1, dtype: int64

event_type_1 types for products:

Religious    55
National     52
Cultural     37
Sporting     18
Name: event_type_1, dtype: int64

event_name_2 types for products:

Father's day      2
Easter            1
Cinco De Mayo     1
OrthodoxEaster    1
Name: event_name_2, dtype: int64

event_type_2 types for products:

Cultural     4
Religious    1
Name: event_type_2, dtype: int64
```

The Price and Calendar datasets will be used to merge with the sales dataset which will help is find holiday event, weekly and annual trends.

Sales dataset

In [8]:

```
sales.head()
```

Out[8]:

	id	item_id	dept_id	cat_id	store_id	state_id	d_1	d_2	d_3	d_4	d_5
0	HOBBIES_1_001_CA_1_validation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0
1	HOBBIES_1_002_CA_1_validation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0
2	HOBBIES_1_003_CA_1_validation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0
3	HOBBIES_1_004_CA_1_validation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0
4	HOBBIES_1_005_CA_1_validation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0	0	0	0	0

5 rows × 1919 columns

In [9]:

```
sales.shape
```

Out[9]:

```
(30490, 1919)
```

In [10]:

```
sales.columns
```

Out[10]:

```
Index(['id', 'item_id', 'dept_id', 'cat_id', 'store_id', 'state_id', 'd_1',
       'd_2', 'd_3', 'd_4',
       ...
       'd_1904', 'd_1905', 'd_1906', 'd_1907', 'd_1908', 'd_1909', 'd_1910',
       'd_1911', 'd_1912', 'd_1913'],
      dtype='object', length=1919)
```

In [11]:

```
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30490 entries, 0 to 30489
Columns: 1919 entries, id to d_1913
dtypes: int64(1913), object(6)
memory usage: 446.4+ MB
```

In [12]:

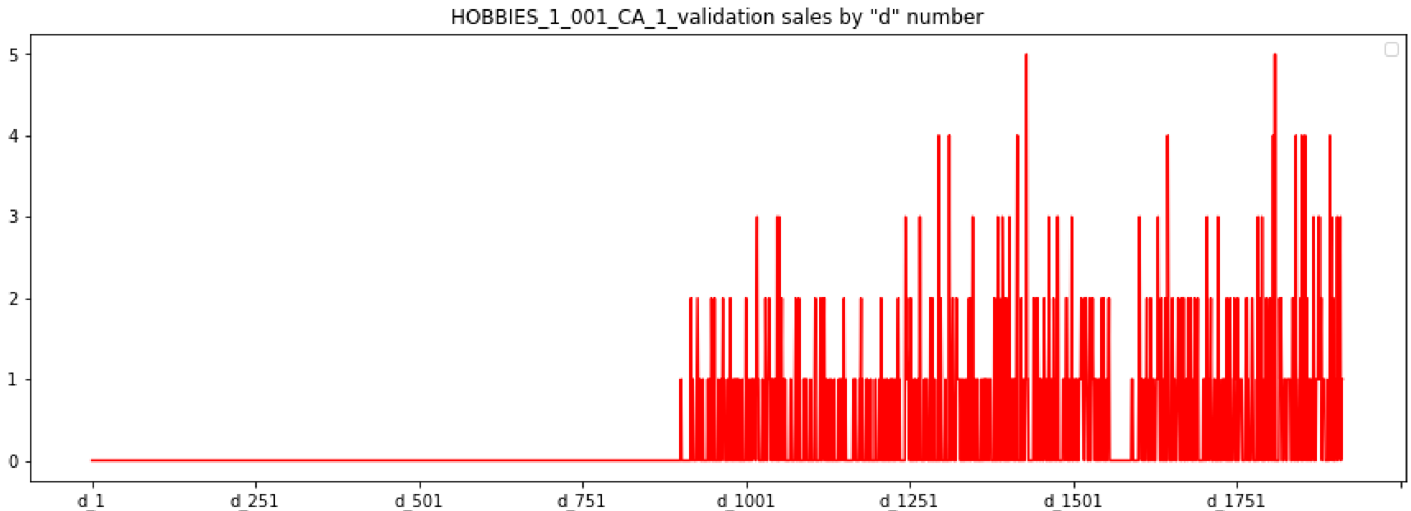
```
sales[sales.columns[:]].describe().T
```

Out[12]:

	count	mean	std	min	25%	50%	75%	max
d_1	30490.0	1.070220	5.126689	0.0	0.0	0.0	0.0	360.0
d_2	30490.0	1.041292	5.365468	0.0	0.0	0.0	0.0	436.0
d_3	30490.0	0.780026	3.667454	0.0	0.0	0.0	0.0	207.0
d_4	30490.0	0.833454	4.415141	0.0	0.0	0.0	0.0	323.0
d_5	30490.0	0.627944	3.379344	0.0	0.0	0.0	0.0	296.0
...	...	...	...	...	...	...	...	...
d_1909	30490.0	1.159167	2.876026	0.0	0.0	0.0	1.0	88.0
d_1910	30490.0	1.149000	2.950364	0.0	0.0	0.0	1.0	77.0
d_1911	30490.0	1.328862	3.358012	0.0	0.0	0.0	1.0	141.0
d_1912	30490.0	1.605838	4.089422	0.0	0.0	0.0	2.0	171.0
d_1913	30490.0	1.633158	3.812248	0.0	0.0	0.0	2.0	130.0

1913 rows × 8 columns

In [13]:
```
#Let's visual on a single product sales

d_cols = [c for c in sales.columns if 'd_' in c]

sales.loc[sales['id'] == 'HOBBIES_1_001_CA_1_validation'].set_index('id')[d_cols].T.plot(f
         title='HOBBIES_1_001_CA_1_validation sales by "d" number',
         color='red')

plt.legend('')
plt.show()
```



Data Preprocessing

Clean

In [14]:
```
print("For prices dataset")
print("Duplicated values: {}" .format(prices.duplicated().sum()))
print("Missing values: {}" .format(prices.isnull().sum()))
print("---------------------------")

print("For calendar dataset")
print("Duplicated values: {}" .format(calendar.duplicated().sum()))
print("Missing values: {}" .format(calendar.isnull().sum()))
print("---------------------------")

print("For prices dataset")
print("Duplicated values: {}" .format(prices.duplicated().sum()))
print("Missing values: {}" .format(prices.isnull().sum()))
print("---------------------------")
```

```
For prices dataset
Duplicated values: 0
Missing values: store_id      0
item_id       0
wm_yr_wk      0
sell_price    0
dtype: int64
---------------------------
For calendar dataset
Duplicated values: 0
Missing values: date              0
wm_yr_wk          0
weekday           0
wday              0
month             0
year              0
```

```
d                    0
event_name_1      1807
event_type_1      1807
event_name_2      1964
event_type_2      1964
snap_CA              0
snap_TX              0
snap_WI              0
dtype: int64
---------------------------
For prices dataset
Duplicated values: 0
Missing values: store_id      0
item_id       0
wm_yr_wk      0
sell_price    0
dtype: int64
---------------------------
```

There are rows contain NaN in event_type_1, event_type_2, event_name_1, and event_name_2

In [15]:
```
calendar = calendar.fillna("no_event")
```

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

Out[16]:

	date	wm_yr_wk	weekday	wday	month	year	d	event_name_1	event_type_1	event_name_2	event_type_
0	2011-01-29	11101	Saturday	1	1	2011	d_1	no_event	no_event	no_event	no_ever
1	2011-01-30	11101	Sunday	2	1	2011	d_2	no_event	no_event	no_event	no_ever
2	2011-01-31	11101	Monday	3	1	2011	d_3	no_event	no_event	no_event	no_ever
3	2011-02-01	11101	Tuesday	4	2	2011	d_4	no_event	no_event	no_event	no_ever
4	2011-02-02	11101	Wednesday	5	2	2011	d_5	no_event	no_event	no_event	no_ever

In [17]:
```
sales = reduce_mem_usage(sales)
```

```
Memory usage of dataframe is 446.40 MB
Memory usage after optimization is: 95.16 MB
Decreased by 78.7%
```

In [18]:
```
calendar = reduce_mem_usage(calendar)
```

```
Memory usage of dataframe is 0.21 MB
Memory usage after optimization is: 0.19 MB
Decreased by 8.7%
```

In [19]:
```
prices = reduce_mem_usage(prices)
```

```
Memory usage of dataframe is 208.77 MB
Memory usage after optimization is: 45.76 MB
Decreased by 78.1%
```

Merging to single dataframe df

In [20]:

```
d_colnames = []
for i in range(1, 1914):
    d_colnames.append('d_'+str(i))

sales_new = pd.melt(sales, id_vars=['id', 'item_id', 'dept_id', 'cat_id', 'store_id', 'sta
        value_vars=d_colnames, var_name='day_number', value_name='units_sold')
```

In [21]:

```
sales_new
```

Out[21]:

	id	item_id	dept_id	cat_id	store_id	state_id	day_number	units_
0	HOBBIES_1_001_CA_1_validation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	d_1	
1	HOBBIES_1_002_CA_1_validation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	d_1	
2	HOBBIES_1_003_CA_1_validation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	d_1	
3	HOBBIES_1_004_CA_1_validation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	d_1	
4	HOBBIES_1_005_CA_1_validation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	d_1	
...	...	...	...	...	...	...	...	
58327365	FOODS_3_823_WI_3_validation	FOODS_3_823	FOODS_3	FOODS	WI_3	WI	d_1913	
58327366	FOODS_3_824_WI_3_validation	FOODS_3_824	FOODS_3	FOODS	WI_3	WI	d_1913	
58327367	FOODS_3_825_WI_3_validation	FOODS_3_825	FOODS_3	FOODS	WI_3	WI	d_1913	
58327368	FOODS_3_826_WI_3_validation	FOODS_3_826	FOODS_3	FOODS	WI_3	WI	d_1913	
58327369	FOODS_3_827_WI_3_validation	FOODS_3_827	FOODS_3	FOODS	WI_3	WI	d_1913	

58327370 rows × 8 columns

In [22]:

```
#taking only last 28 data days of test

# d_colnames = []
# for i in range(1914,1942):
#     d_colnames.append("d_"+str(i))

# sales_2_new = pd.melt(sales_2, id_vars=['id', 'item_id', 'dept_id', 'cat_id', 'store_id
#         value_vars=d_colnames, var_name='day_number', value_name='units_sold')
```

In [23]:

```
# for i in range(1942,1970):
#     sales_2['d_'+str(i)]=0
```

In [24]:

```
#Create futures sales data

# d_colnames = []
# for i in range(1942,1970):
#     d_colnames.append("d_"+str(i))

# sales_2_future = pd.melt(sales_2, id_vars=['id', 'item_id', 'dept_id', 'cat_id', 'store_
#         value_vars=d_colnames, var_name='day_number', value_name='units_sold')
```

In [25]:

```
#Merge all These 3 datasets to get final df train

data = sales_new.merge(calendar,left_on='day_number', right_on='d', how='left')
data = data.merge(prices, on=['store_id', 'item_id', 'wm_yr_wk'], how='left')
```

```
data.drop(['day_number','weekday','d',], inplace = True, axis = 1)
data['sell_price'].fillna(0, inplace = True)
data = reduce_mem_usage(data)
# data.to_csv('data/final_data.csv',index=False)
```

```
Memory usage of dataframe is 1948.29 MB
Memory usage after optimization is: 1948.29 MB
Decreased by 0.0%
```

In [26]:

```
#Merge all These 3 datasets to get final df test

# data_test = sales_2_new.merge(calendar,left_on='day_number', right_on='d', how='left')
# data_test = data_test.merge(prices, on=['store_id', 'item_id', 'wm_yr_wk'], how='left')
# data_test.drop(['day_number','weekday','d',], inplace = True, axis = 1)
# data_test['sell_price'].fillna(0, inplace = True)
# data_test = reduce_mem_usage(data_test)
# data_test.to_csv('data/final_data_test.csv',index=False)
```

In [27]:

```
#Merge all These 3 datasets to get final df future

# data_future = sales_2_future.merge(calendar,left_on='day_number', right_on='d', how='le
# data_future = data_future.merge(prices, on=['store_id', 'item_id', 'wm_yr_wk'], how='le
# data_future.drop(['day_number','weekday','d',], inplace = True, axis = 1)
# data_future['sell_price'].fillna(0, inplace = True)
# data_future = reduce_mem_usage(data_future)
# data_future.to_csv('data/final_data_future.csv',index=False)
```

In [28]:

```
#Load saved dataframe
# data = pd.read_csv('data/final_data.csv')
# test = pd.read_csv('data/final_data_test.csv')
# future = pd.read_csv('data/final_data_future.csv')
```

In [29]:

```
%%time
data = reduce_mem_usage(data)
# test = reduce_mem_usage(test)
# future = reduce_mem_usage(future)
```

```
Memory usage of dataframe is 1948.29 MB
Memory usage after optimization is: 1948.29 MB
Decreased by 0.0%
Wall time: 2.2 s
```

In [30]:

```
data
```

Out[30]:

	id	item_id	dept_id	cat_id	store_id	state_id	units_sold	date
0	HOBBIES_1_001_CA_1_validation	HOBBIES_1_001	HOBBIES_1	HOBBIES	CA_1	CA	0	2011-01-29
1	HOBBIES_1_002_CA_1_validation	HOBBIES_1_002	HOBBIES_1	HOBBIES	CA_1	CA	0	2011-01-29
2	HOBBIES_1_003_CA_1_validation	HOBBIES_1_003	HOBBIES_1	HOBBIES	CA_1	CA	0	2011-01-29
3	HOBBIES_1_004_CA_1_validation	HOBBIES_1_004	HOBBIES_1	HOBBIES	CA_1	CA	0	2011-01-29
4	HOBBIES_1_005_CA_1_validation	HOBBIES_1_005	HOBBIES_1	HOBBIES	CA_1	CA	0	2011-01-29

	id	item_id	dept_id	cat_id	store_id	state_id	units_sold	date
...	...	...	...	...	...	...	...	...
58327365	FOODS_3_823_WI_3_validation	FOODS_3_823	FOODS_3	FOODS	WI_3	WI	1	2016-04-24
58327366	FOODS_3_824_WI_3_validation	FOODS_3_824	FOODS_3	FOODS	WI_3	WI	0	2016-04-24
58327367	FOODS_3_825_WI_3_validation	FOODS_3_825	FOODS_3	FOODS	WI_3	WI	0	2016-04-24
58327368	FOODS_3_826_WI_3_validation	FOODS_3_826	FOODS_3	FOODS	WI_3	WI	3	2016-04-24
58327369	FOODS_3_827_WI_3_validation	FOODS_3_827	FOODS_3	FOODS	WI_3	WI	0	2016-04-24

58327370 rows × 20 columns

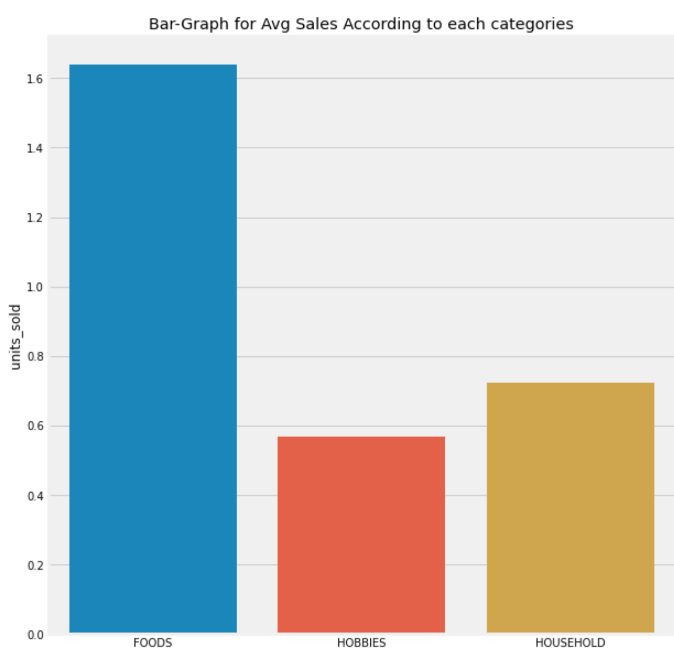
In [ ]:

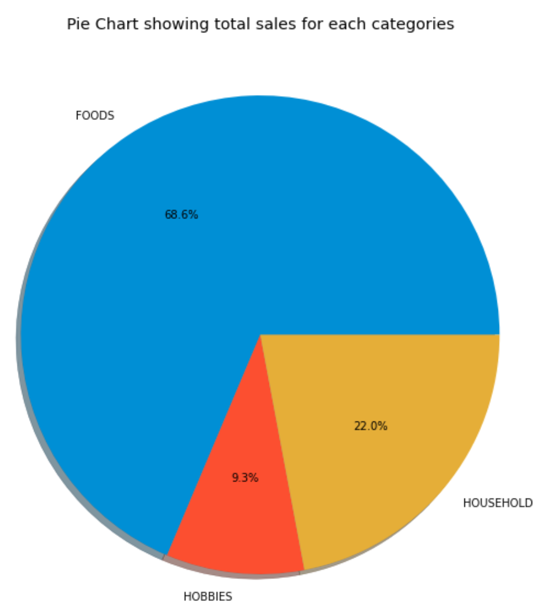
Exploratory Data Analysis

Visual on catrgorical data

In [310…

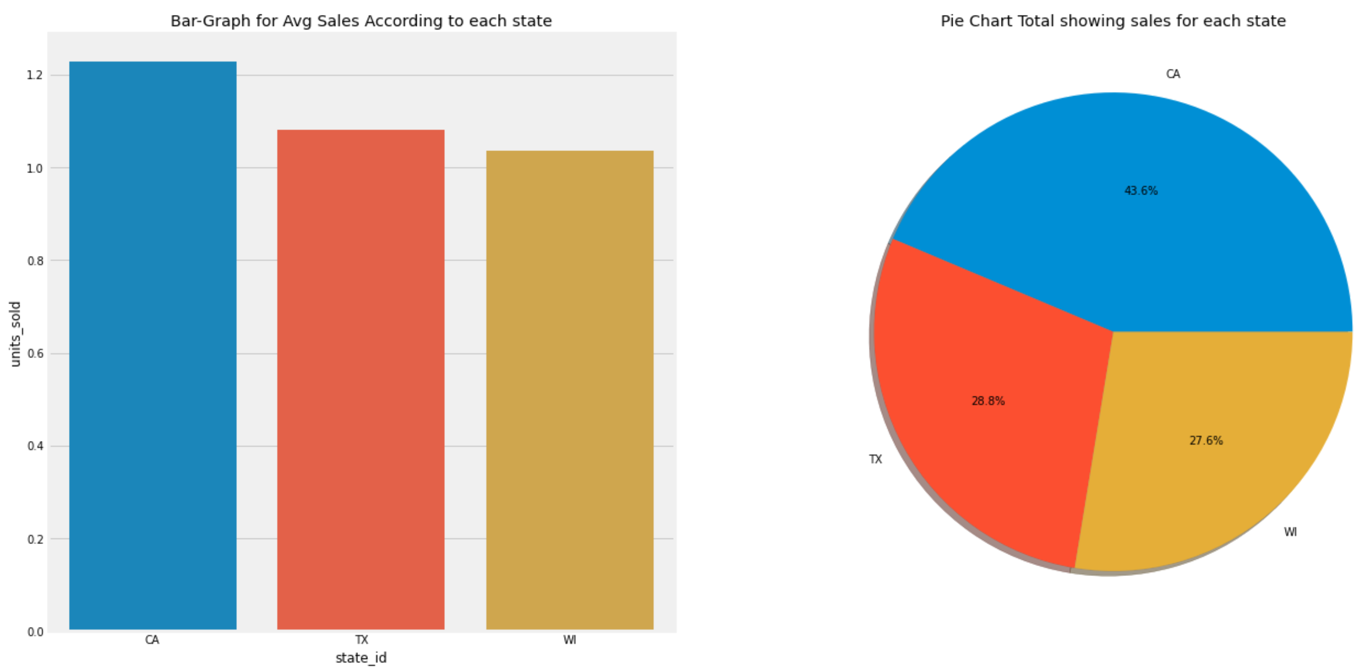
```
df=data.groupby('cat_id').mean()
df.reset_index(level=0,inplace=True)
plt.figure(figsize=(20,10))
plt.subplot(121)
sns.barplot(x='cat_id',y='units_sold',data=df)
plt.title("Bar-Graph for Avg Sales According to each categories")

plt.subplot(122)
df=data.groupby('cat_id').sum()
df.reset_index(level=0,inplace=True)
df['perc']=df['units_sold']/sum(df['units_sold'].values)*100
plt.pie(df['perc'].values,labels=df['cat_id'].values,shadow=True,autopct='%1.1f%%')
plt.title("Pie Chart showing total sales for each categories")
plt.show()
```





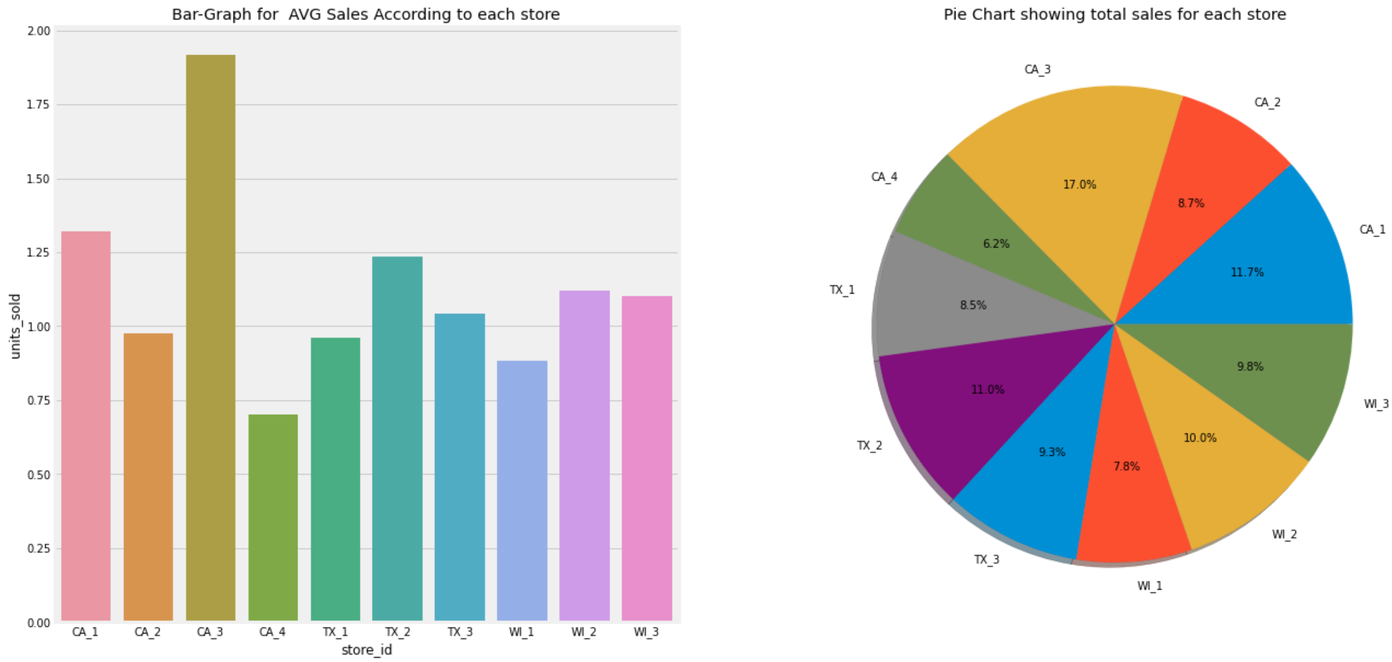
In [311…

```
#Total Sales for each State(Barplot+Pie Chart)
#https://matplotlib.org/3.1.1/gallery/pie_and_polar_charts/pie_features.html
df=data.groupby('state_id').mean()
df.reset_index(level=0,inplace=True)
plt.figure(figsize=(20,10))
plt.subplot(121)
sns.barplot(x='state_id',y='units_sold',data=df)
plt.title("Bar-Graph for Avg Sales According to each state")

df=data.groupby('state_id').sum()
df.reset_index(level=0,inplace=True)
df['perc']=df['units_sold']/sum(df['units_sold'].values)*100
plt.subplot(122)
plt.pie(df['perc'].values,labels=df['state_id'].values,shadow=True,autopct='%1.1f%%')
plt.title("Pie Chart Total showing sales for each state")
plt.show()
```



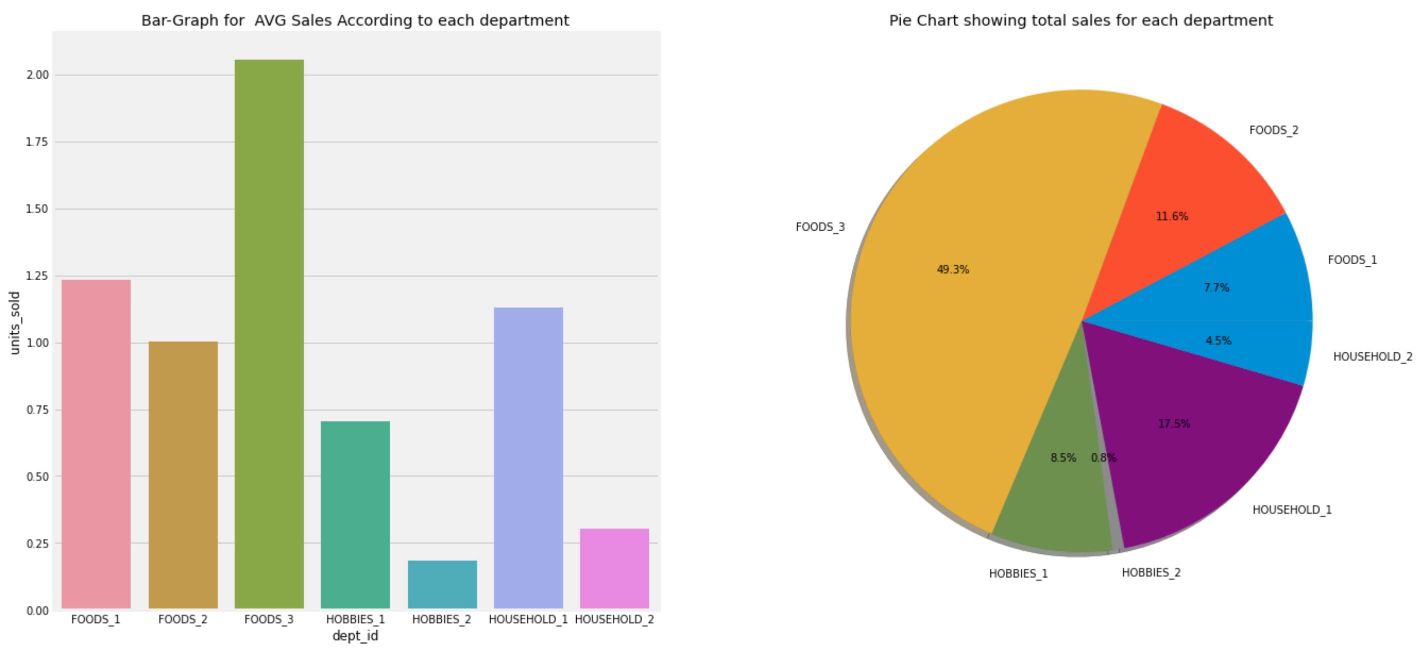
In [312…

```
#Total Sales for each store(Bargraph+pieChart)
#https://matplotlib.org/3.1.1/gallery/pie_and_polar_charts/pie_features.html
df=data.groupby('store_id').mean()
df.reset_index(level=0,inplace=True)
plt.figure(figsize=(20,10))
plt.subplot(121)
sns.barplot(x='store_id',y='units_sold',data=df)
plt.title("Bar-Graph for  AVG Sales According to each store")
df=data.groupby('store_id').sum()
df.reset_index(level=0,inplace=True)
df['perc']=df['units_sold']/sum(df['units_sold'].values)*100
plt.subplot(122)
plt.pie(df['perc'].values,labels=df['store_id'].values,shadow=True,autopct='%1.1f%%')
plt.title("Pie Chart showing total sales for each store")

plt.show()
```



In [313…

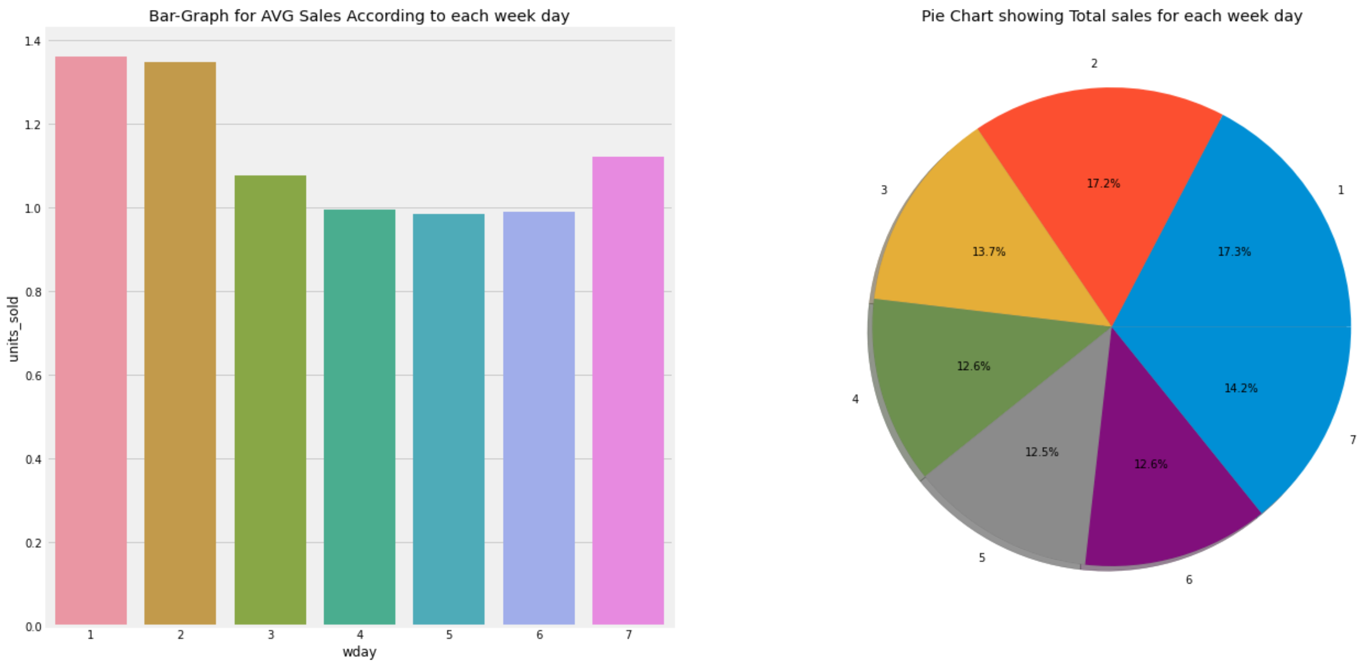
```
#Total Sales for each store(Bargraph+pieChart)
#https://matplotlib.org/3.1.1/gallery/pie_and_polar_charts/pie_features.html
df=data.groupby('dept_id').mean()
df.reset_index(level=0,inplace=True)
plt.figure(figsize=(20,10))
plt.subplot(121)
sns.barplot(x='dept_id',y='units_sold',data=df)
plt.title("Bar-Graph for  AVG Sales According to each department")
df=data.groupby('dept_id').sum()
df.reset_index(level=0,inplace=True)
df['perc']=df['units_sold']/sum(df['units_sold'].values)*100
plt.subplot(122)
plt.pie(df['perc'].values,labels=df['dept_id'].values,shadow=True,autopct='%1.1f%%')
plt.title("Pie Chart showing total sales for each department")

plt.show()
```



In [314…

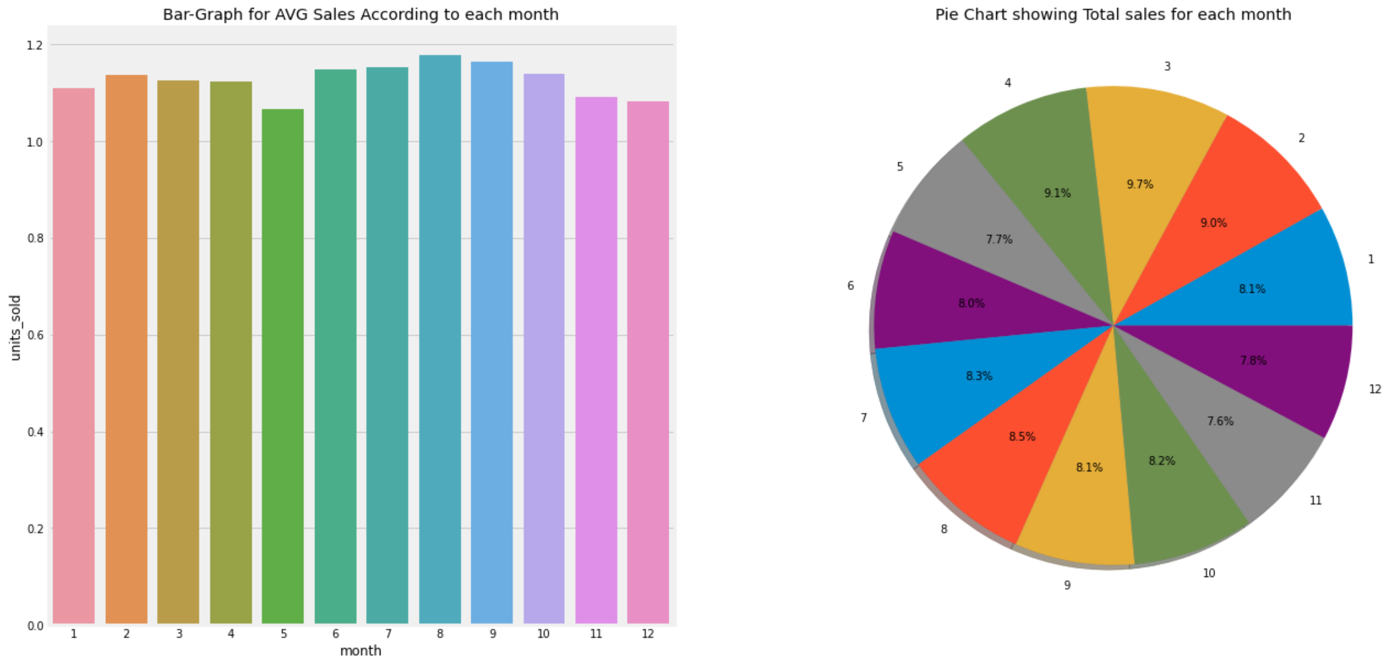
```
#Total Sales for each day of week(Barplot+PieChart)
#https://matplotlib.org/3.1.1/gallery/pie_and_polar_charts/pie_features.html
df=data.groupby('wday').mean()
df.reset_index(level=0,inplace=True)
```

```
plt.figure(figsize=(20,10))
plt.subplot(121)
sns.barplot(x='wday',y='units_sold',data=df)
plt.title("Bar-Graph for AVG Sales According to each week day")

df=data.groupby('wday').sum()
df.reset_index(level=0,inplace=True)
df['perc']=df['units_sold']/sum(df['units_sold'].values)*100
plt.subplot(122)
plt.pie(df['perc'].values,labels=df['wday'].values,shadow=True,autopct='%1.1f%%')
plt.title("Pie Chart showing Total sales for each week day")

plt.show()
```



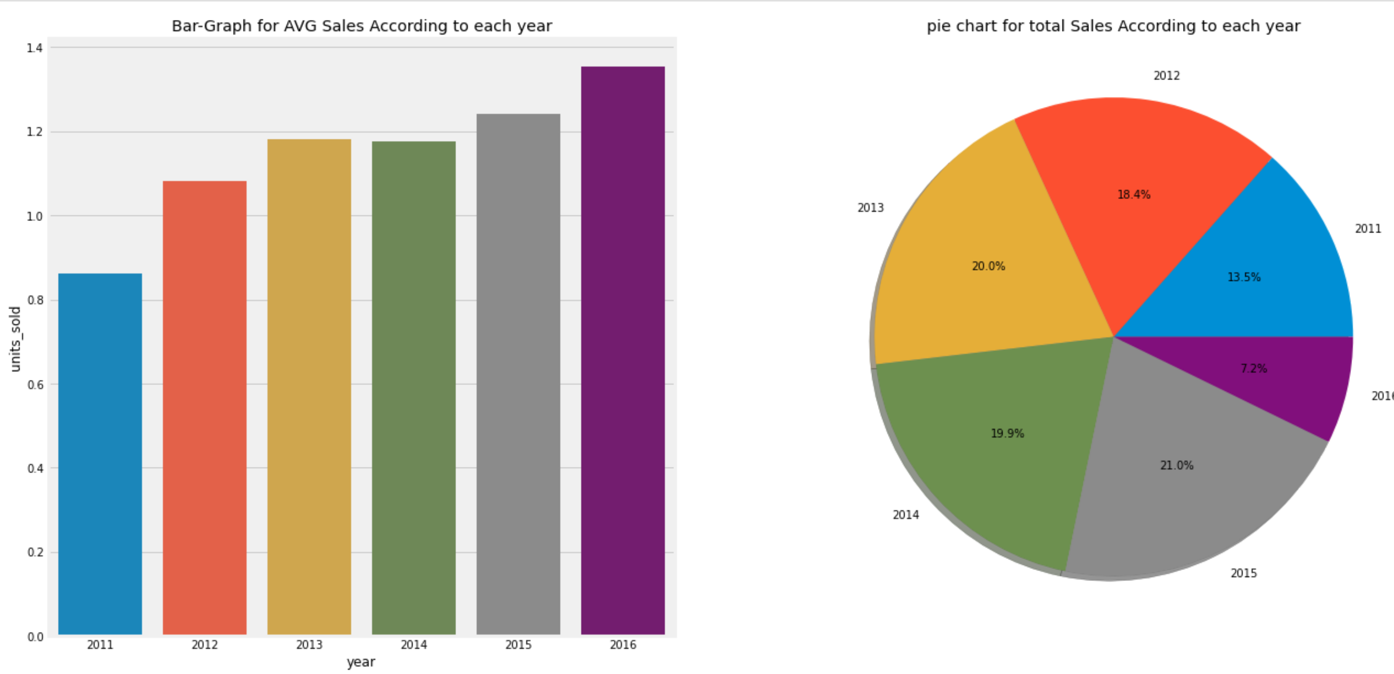
In [315…

```
#Total Sales for each month(Barplot+pie chart)
#https://matplotlib.org/3.1.1/gallery/pie_and_polar_charts/pie_features.html
df=data.groupby('month').mean()
df.reset_index(level=0,inplace=True)
plt.figure(figsize=(20,10))
plt.subplot(121)
sns.barplot(x='month',y='units_sold',data=df)
plt.title("Bar-Graph for AVG Sales According to each month")

df=data.groupby('month').sum()
df.reset_index(level=0,inplace=True)
df['perc']=df['units_sold']/sum(df['units_sold'].values)*100
plt.subplot(122)
plt.pie(df['perc'].values,labels=df['month'].values,shadow=True,autopct='%1.1f%%')
plt.title("Pie Chart showing Total sales for each month")

plt.show()
```



In [316…

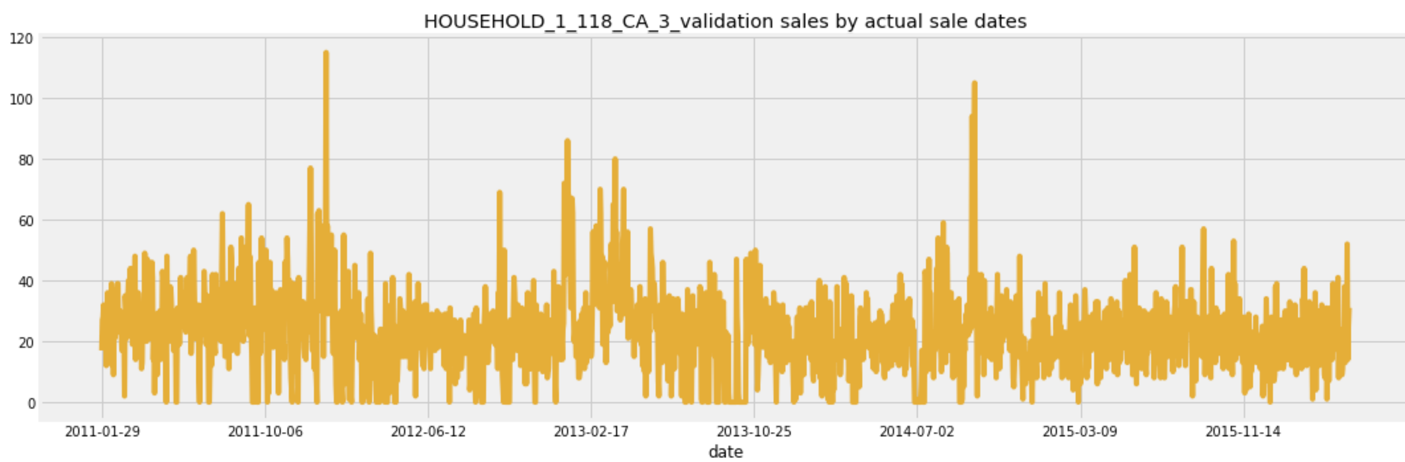
```python
#Total Sales for each year(bar plot+pie chart)
#https://matplotlib.org/3.1.1/gallery/pie_and_polar_charts/pie_features.html

df=data.groupby('year').mean()
df.reset_index(level=0,inplace=True)
plt.figure(figsize=(20,10))
plt.subplot(121)

sns.barplot(x='year',y='units_sold',data=df)
plt.title("Bar-Graph for AVG Sales According to each year")

plt.subplot(122)
df=data.groupby('year').sum()
df.reset_index(level=0,inplace=True)
df['year_avg']=df['units_sold']/sum(df['units_sold'].values)*100
plt.pie(df['year_avg'].values,labels=df['year'].values,shadow=True,autopct='%1.1f%%')

plt.title("pie chart for total Sales According to each year")
plt.show()
```



Visual on time series data

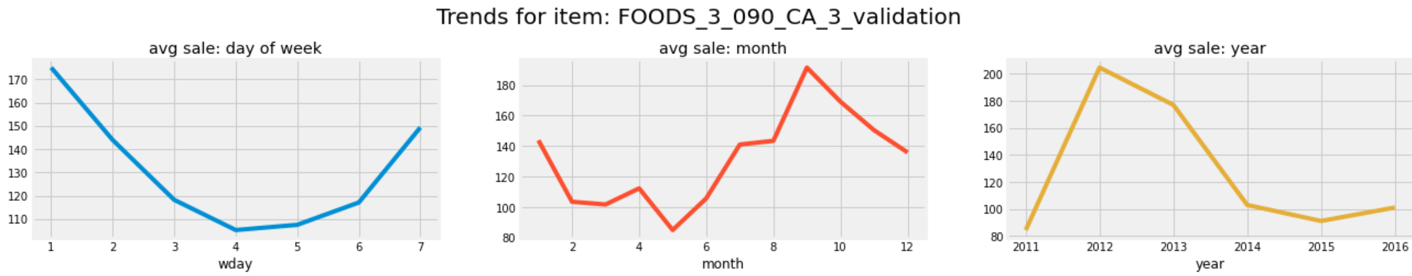
In [317…

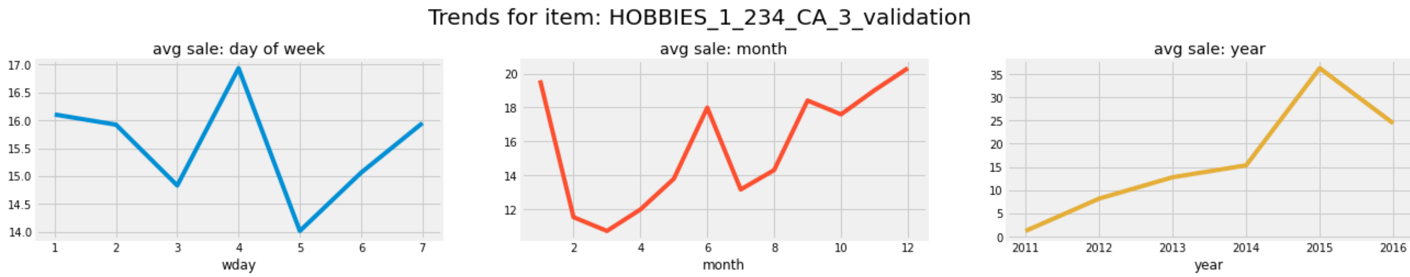
```
ids = ['FOODS_3_090_CA_3_validation', 'HOBBIES_1_234_CA_3_validation', 'HOUSEHOLD_1_118_CA
examples = []

# General function to extract specific item Id values from sales_train_evaluation dataset
for i in range(len(ids)):
    # Merge calendar date with sales_train_evaluation date
    examples.append(sales.loc[sales['id']== ids[i]][d_cols].T) # Fetch a specific Id and t
    examples[i] = examples[i].rename(columns={examples[i].columns.values[0] : ids[i]}) # R
    examples[i] = examples[i].reset_index().rename(columns={'index':'d'}) # Reset index so
    examples[i] = examples[i].merge(calendar) # Merge two dataframe on the bases of column

graph_ids = [2]
fig, axs = plt.subplots(len(graph_ids),1,figsize=(15,10))
if len(graph_ids) != 1 : axs=axs.flatten()

ax_id=0
for graph_id in graph_ids:
    # Set date as the index of dataframe to be plotted properly and only fetch specific Id
    examples[graph_id].set_index('date')[ids[graph_id]] \
    .plot(figsize=(15,5), color=next(color_cycle), title= f'{ids[graph_id]} sales by actua

    ax_id+=1
plt.tight_layout()
plt.show()
```

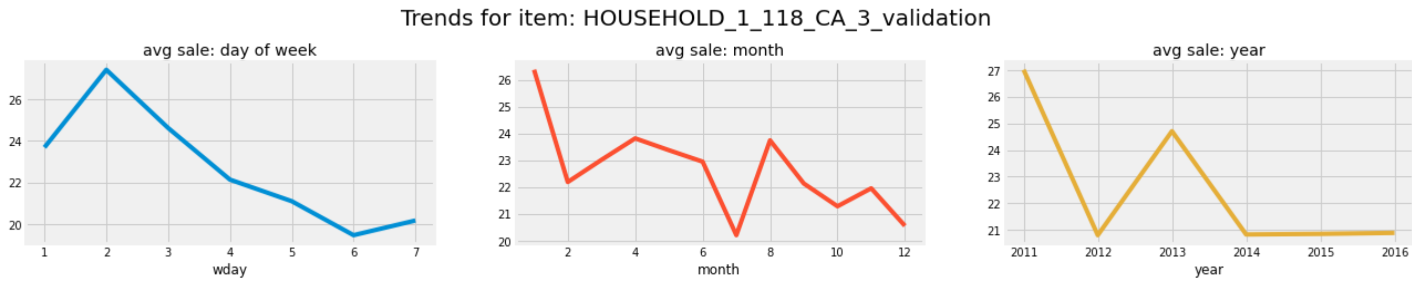


In [318…

```
for i in [0, 1, 2]:
    fig, (ax1, ax2, ax3) = plt.subplots(1,3,figsize=(20,3))
    # Groupby wday colum and do mean or count etc. to values of other columns & then, extr
    examples[i].groupby('wday').mean()[ids[i]] \
    .plot(kind='line', title='avg sale: day of week', color=color_pal[0], ax=ax1)

    examples[i].groupby('month').mean()[ids[i]] \
    .plot(kind='line', title='avg sale: month', color=color_pal[1], ax=ax2)

    examples[i].groupby('year').mean()[ids[i]] \
    .plot(kind='line', title='avg sale: year', color=color_pal[2], ax=ax3)

    fig.suptitle(f'Trends for item: {ids[i]}', size=20, y=1.1)
```

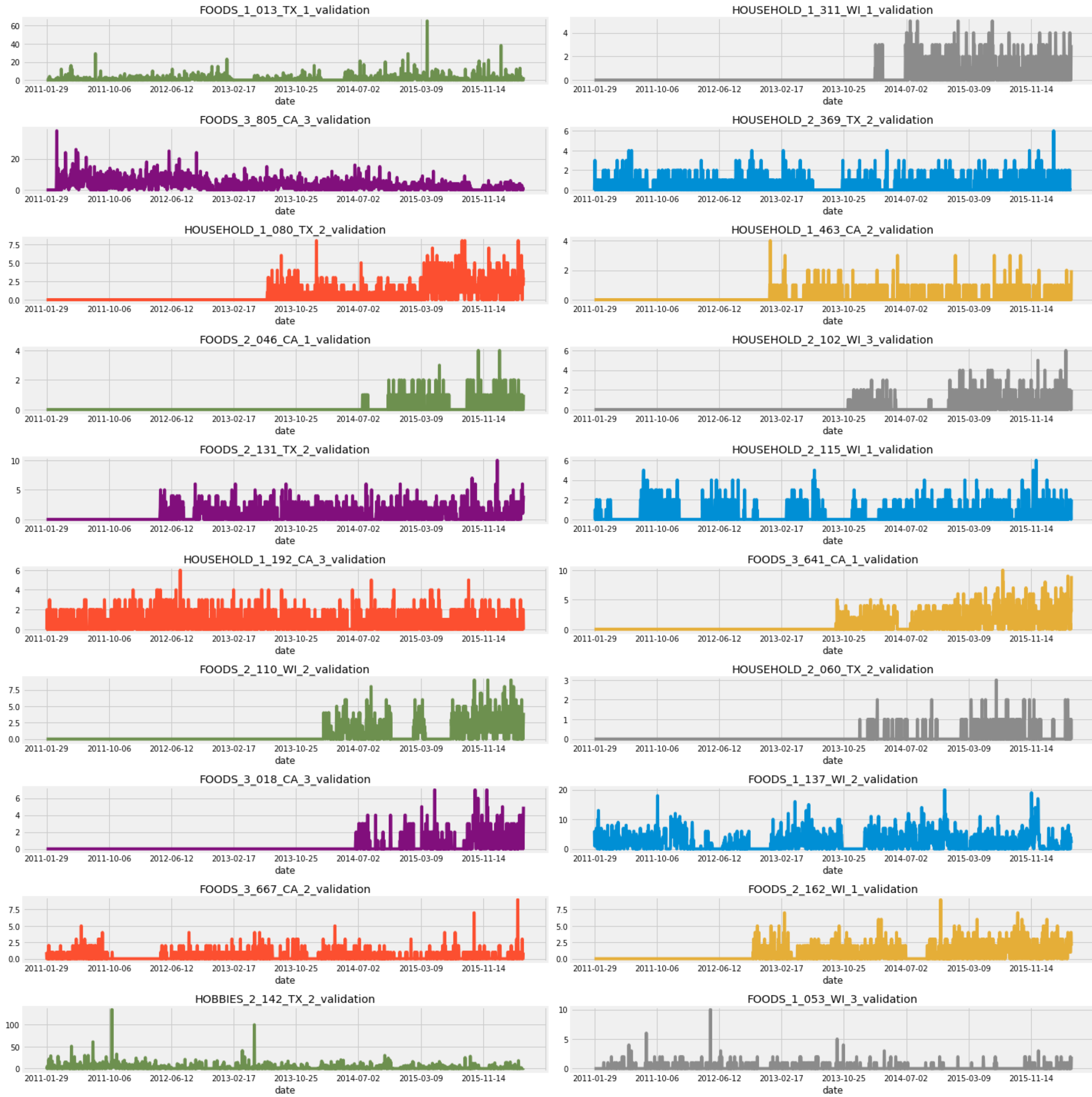






In [319…

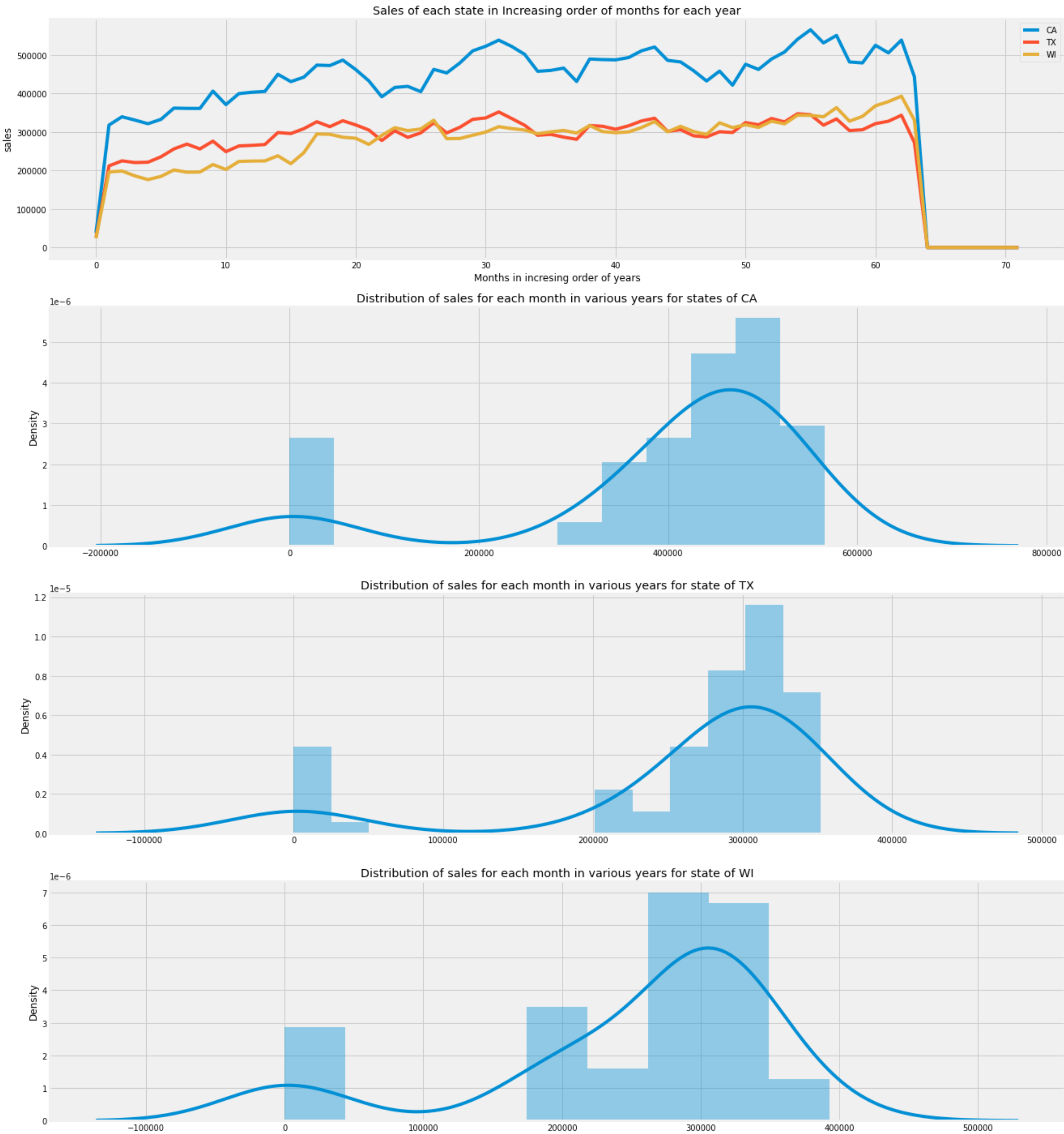
```
twenty_examples = sales.sample(20, random_state=529) # Sample random 20 samples from the s
list_of_sampled_id = list(twenty_examples['id']) # Extract the list of sampled id of items
twenty_examples = twenty_examples.set_index('id')[d_cols].T # Set the index of dataframe t
twenty_examples = twenty_examples.reset_index().rename(columns={'index':'d'})
twenty_examples = twenty_examples.merge(calendar)
twenty_examples = twenty_examples.set_index('date')[list_of_sampled_id]


fig, axs = plt.subplots(10,2,figsize=(20,20))
axs = axs.flatten()

axs_id=0
for item in twenty_examples.columns:
    twenty_examples[item].plot(title=item, color=next(color_cycle), ax=axs[axs_id])
    axs_id+=1
plt.tight_layout()
plt.show()
```



In [320…

```
df=data.groupby(['year','month','state_id']).sum()
df.reset_index(level=[0,1,2],inplace=True)
plt.figure(figsize=(20,25))
plt.subplot(411)
plt.plot(df[df['state_id']=="CA"]['units_sold'].values,label="CA")
plt.plot(df[df['state_id']=="TX"]['units_sold'].values,label="TX")
plt.plot(df[df['state_id']=="WI"]['units_sold'].values,label="WI")
plt.xlabel('Months in incresing order of years')
plt.ylabel('sales')
plt.title("Sales of each state in Increasing order of months for each year")
plt.legend()

plt.subplot(412)
sns.distplot(df[df['state_id']=="CA"]['units_sold'].values)
plt.title("Distribution of sales for each month in various years for states of CA")

plt.subplot(413)
sns.distplot(df[df['state_id']=="TX"]['units_sold'].values)
plt.title("Distribution of sales for each month in various years for state of TX")
```

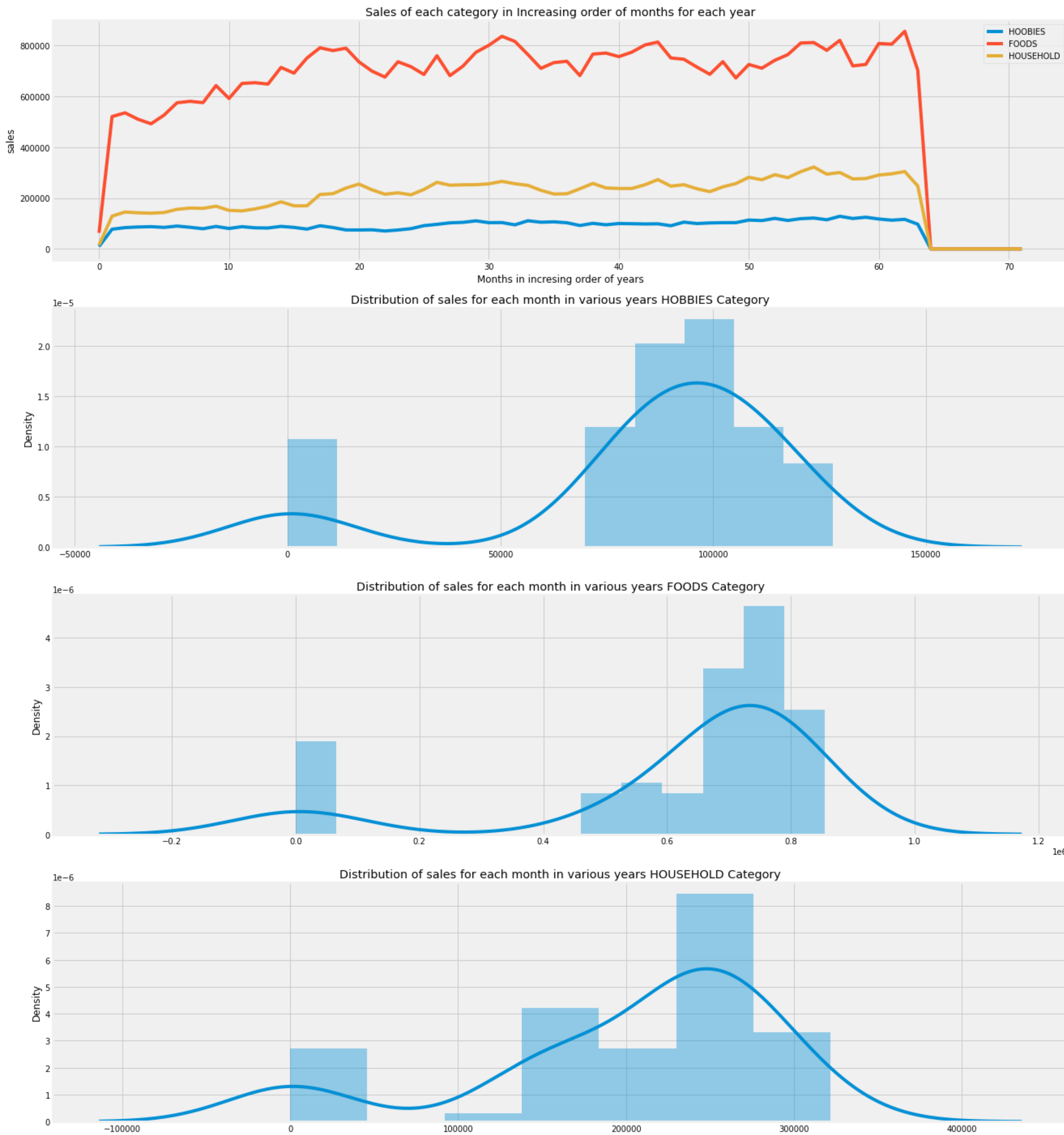
```
plt.subplot(414)
sns.distplot(df[df['state_id']=="WI"]['units_sold'].values)
plt.title("Distribution of sales for each month in various years for state of WI")

plt.show()
```



In [323…

```
df=data.groupby(['year','month','cat_id']).sum()
df.reset_index(level=[0,1,2],inplace=True)
plt.figure(figsize=(20,25))
plt.subplot(411)
plt.plot(df[df['cat_id']=="HOBBIES"]['units_sold'].values,label="HOOBIES")
plt.plot(df[df['cat_id']=="FOODS"]['units_sold'].values,label="FOODS")
plt.plot(df[df['cat_id']=="HOUSEHOLD"]['units_sold'].values,label="HOUSEHOLD")
plt.xlabel('Months in incresing order of years')
plt.ylabel('sales')
plt.title("Sales of each category in Increasing order of months for each year")
```

```
plt.legend()
plt.subplot(412)
sns.distplot(df[df['cat_id']=="HOBBIES"]['units_sold'].values)
plt.title("Distribution of sales for each month in various years HOBBIES Category")


plt.subplot(413)
sns.distplot(df[df['cat_id']=="FOODS"]['units_sold'].values)
plt.title("Distribution of sales for each month in various years FOODS Category")


plt.subplot(414)
sns.distplot(df[df['cat_id']=="HOUSEHOLD"]['units_sold'].values)
plt.title("Distribution of sales for each month in various years HOUSEHOLD Category")

plt.show()
```



Section 3: Model / Feature selection

Very Unfortunately, there are some problems running in the model training process due to lag of memory in my computing resources, I decided to reduce the data to around one year from the latest date.

In [31]:

```
#Dropping the column

for col in range(1,1548):

    sales.drop(['d_'+str(col)], inplace = True, axis = 1)

#Reduce the category to food only
# sales = sales.loc[sales['cat_id']=='FOODS' ]

d_colnames = []
for i in range(1548, 1914):
    d_colnames.append('d_'+str(i))

sales_new = pd.melt(sales, id_vars=['id', 'item_id', 'dept_id', 'cat_id', 'store_id', 'sta
        value_vars=d_colnames, var_name='day_number', value_name='units_sold')


#Merge all These 3 datasets to get final df train
data = sales_new.merge(calendar,left_on='day_number', right_on='d', how='left')
data = data.merge(prices, on=['store_id', 'item_id', 'wm_yr_wk'], how='left')
data.drop(['day_number','weekday','d',], inplace = True, axis = 1)
data['sell_price'].fillna(0, inplace = True)
data = reduce_mem_usage(data)
```

```
Memory usage of dataframe is 373.89 MB
Memory usage after optimization is: 373.89 MB
Decreased by 0.0%
```

Feature Engineering

Dealing with caterogical data

In [33]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11159340 entries, 0 to 11159339
Data columns (total 20 columns):
 #   Column        Dtype   
---  ------        -----   
 0   id            category
 1   item_id       category
 2   dept_id       category
 3   cat_id        category
 4   store_id      category
 5   state_id      category
 6   units_sold    int16   
 7   date          category
 8   wm_yr_wk      int16   
 9   wday          int8    
 10  month         int8    
 11  year          int16   
 12  event_name_1  category
 13  event_type_1  category
 14  event_name_2  category
 15  event_type_2  category
 16  snap_CA       int8    
```

```
 17   snap_TX        int8
 18   snap_WI        int8
 19   sell_price     float16
dtypes: category(11), float16(1), int16(3), int8(5)
memory usage: 373.9 MB
```

In [34]:
```
def prep_categorical(df):
    """This function is for prepare catergorical data in dataframe"""
    categorical_col = list(data.dtypes[data.dtypes=='category'].index)
    categorical_col.remove('id')
    categorical_col.remove('date')
    categorical_col

    for feature in categorical_col:

        encoder = preprocessing.LabelEncoder()
        df[feature] = encoder.fit_transform(df[feature])

    return df
```

In [35]:
```
%%time
data = prep_categorical(data)
```

```
Wall time: 16.3 s
```

In this analysis, Features that we could generate in this section will be:

Units sold related feature:

- Rolling mean 7 days on Unit_sold
- Rolling mean 30 days on Unit_sold
- Rolling mean 90 days on Unit_sold
- Rolling mean 180 days on Unit_sold
- Rolling std 7 days on Unit_sold
- Rolling std 7 days on Unit_sold
- Lagged Unit sold from 7 – 30 days

Price related feature:

- Price change in 7 days
- Price change in 30 days
- Rolling std 7 days on Price
- Rolling std 30 days on Price

Time related feature:

- Week numbers
- Days
- Quarter number
- Seasonal

In [36]:
```
def get_season(x):
    """This function is used to get season in US according to various months"""

    if x in [12,1,2]:
        return 0         #"Winter"
```

```
    elif x in [3,4,5]:
        return 1        #"Spring"
    elif x in [6,7,8]:
        return 2        #"Summer"
    else:
        return 3        #"Autumn"

def month_start(x):
    """This is used to check if day is begining of month"""

    day=calendar.datetime.date.fromisoformat(x).day

    return 1 if day==1 else 0

def month_end(x):
    """This is used to check if day is end of month"""

    day=calendar.datetime.date.fromisoformat(x).day
    month=calendar.datetime.date.fromisoformat(x).month
    year=calendar.datetime.date.fromisoformat(x).year
    leap_yr=(year%4==0) # to check if it is a leap year
    val=(day==31 and month==1) or (day==29 if leap_yr else day==28) or (day==31 and month=
        (day==31 and month==5) or (day==30 and month==6) or (day==31 and month==7) or (day
        (day==30 and month==9) or (day==31 and month==10) or (day==30 and month==11) or (c

    return 1 if val else 0

def quater_begin(x):
    """This is used to check if day is begining of quater"""

    day=calendar.datetime.date.fromisoformat(x).day
    month=calendar.datetime.date.fromisoformat(x).month

    return 1 if (day==1 and (month in [1,4,7,9])) else 0

def quater_end(x):
    """This is used to check if day is end of quater"""

    day=calendar.datetime.date.fromisoformat(x).day
    month=calendar.datetime.date.fromisoformat(x).month
    if (day==31 and month==3) or (day==30 and month==6) or (day==30 and month==9) or (day=
        return 1
    else:
        return 0

def year_start(x):
    """This is used to check if day is begining of year"""

    day=calendar.datetime.date.fromisoformat(x).day
    month=calendar.datetime.date.fromisoformat(x).month

    return 1 if (day==1 and month==1) else 0


def year_end(x):
    """This is used to check if day is end of year"""

    day=calendar.datetime.date.fromisoformat(x).day
    month=calendar.datetime.date.fromisoformat(x).month

    return 1 if (day==31 and month==12) else 0

def add_lagging_feature(data, lag_d =30):
    """This function is used to get lagging units sold in day"""

    for i in range(7,lag_d+1):
```

```
        data['lag_t' + str(i)] = data.groupby(['id'])['units_sold'].transform(lambda x: x.

    return data
```

In [37]:

```
# Function to do feature engineering.
def feature_engineering(data):

    # units_sold features.
    data = add_lagging_feature(data,30)
    data['rolling_mean_t7'] = data.groupby(['id'])['units_sold'].transform(lambda x: x.shi
    data['rolling_std_t7'] = data.groupby(['id'])['units_sold'].transform(lambda x: x.shif
    data['rolling_mean_t30'] = data.groupby(['id'])['units_sold'].transform(lambda x: x.sh
    data['rolling_mean_t90'] = data.groupby(['id'])['units_sold'].transform(lambda x: x.sh
    data['rolling_mean_t180'] = data.groupby(['id'])['units_sold'].transform(lambda x: x.s
    data['rolling_std_t30'] = data.groupby(['id'])['units_sold'].transform(lambda x: x.shi
    data = reduce_mem_usage(data)

    # Price features
    data['lag_price_t7'] = data.groupby(['id'])['sell_price'].transform(lambda x: x.shift
    data['price_change_t7'] = (data['lag_price_t7'] - data['sell_price']) / (data['lag_pri
    data['rolling_price_max_t30'] = data.groupby(['id'])['sell_price'].transform(lambda x:
    data['price_change_t30'] = (data['rolling_price_max_t30'] - data['sell_price']) / (dat
    data['rolling_price_std_t7'] = data.groupby(['id'])['sell_price'].transform(lambda x:
    data['rolling_price_std_t30'] = data.groupby(['id'])['sell_price'].transform(lambda x:
    data.drop(['rolling_price_max_t30', 'lag_price_t7'], inplace = True, axis = 1)
    data = reduce_mem_usage(data)

    # Time features
    data['date'] = pd.to_datetime(data['date'])
    data['week'] = data['date'].dt.week
    data['day'] = data['date'].dt.day
    data['day'] = data['date'].dt.quarter
    data['season'] = data['month'].apply(lambda x:get_season(x))

#     data = reduce_mem_usage(data)


    return data
```

In [38]:

```
%%time
data = feature_engineering(data)
gc.collect()
```

```
Memory usage of dataframe is 3204.67 MB
Memory usage after optimization is: 1012.34 MB
Decreased by 68.4%
Memory usage of dataframe is 1289.05 MB
Memory usage after optimization is: 1225.19 MB
Decreased by 5.0%
Wall time: 4min 19s
```

Out[38]:

```
18
```

In [39]:

```
data
```

Out[39]:

	id	item_id	dept_id	cat_id	store_id	state_id	units_sold	date	wm_yr_wk
0	HOBBIES_1_001_CA_1_validation	1437	3	1	0	0	2	2015-04-25	11513

	id	item_id	dept_id	cat_id	store_id	state_id	units_sold	date	wm_yr_wk
1	HOBBIES_1_002_CA_1_validation	1438	3	1	0	0	0	2015-04-25	11513
2	HOBBIES_1_003_CA_1_validation	1439	3	1	0	0	0	2015-04-25	11513
3	HOBBIES_1_004_CA_1_validation	1440	3	1	0	0	2	2015-04-25	11513
4	HOBBIES_1_005_CA_1_validation	1441	3	1	0	0	2	2015-04-25	11513
...	...	...	...	...	...	...	...	...	...
11159335	FOODS_3_823_WI_3_validation	1432	2	0	9	2	1	2016-04-24	11613
11159336	FOODS_3_824_WI_3_validation	1433	2	0	9	2	0	2016-04-24	11613
11159337	FOODS_3_825_WI_3_validation	1434	2	0	9	2	0	2016-04-24	11613
11159338	FOODS_3_826_WI_3_validation	1435	2	0	9	2	3	2016-04-24	11613
11159339	FOODS_3_827_WI_3_validation	1436	2	0	9	2	0	2016-04-24	11613

11159340 rows × 57 columns

In [40]:

```
data.replace([np.inf, -np.inf], np.nan, inplace=True)
```

In [41]:

```
data.isna().sum()
```

Out[41]:

```
id                     0
item_id                0
dept_id                0
cat_id                 0
store_id               0
state_id               0
units_sold             0
date                   0
wm_yr_wk               0
wday                   0
month                  0
year                   0
event_name_1           0
event_type_1           0
event_name_2           0
event_type_2           0
snap_CA                0
snap_TX                0
snap_WI                0
sell_price             0
lag_t7            213430
lag_t8            243920
lag_t9            274410
lag_t10           304900
lag_t11           335390
lag_t12           365880
lag_t13           396370
```

```
lag_t14                   426860
lag_t15                   457350
lag_t16                   487840
lag_t17                   518330
lag_t18                   548820
lag_t19                   579310
lag_t20                   609800
lag_t21                   640290
lag_t22                   670780
lag_t23                   701270
lag_t24                   731760
lag_t25                   762250
lag_t26                   792740
lag_t27                   823230
lag_t28                   853720
lag_t29                   884210
lag_t30                   914700
rolling_mean_t7          1036660
rolling_std_t7           1036660
rolling_mean_t30         1737930
rolling_mean_t90         3567330
rolling_mean_t180        6311430
rolling_std_t30          1737930
price_change_t7           245882
price_change_t30          935749
rolling_price_std_t7      182940
rolling_price_std_t30     884210
week                           0
day                            0
season                         0
dtype: int64
```

In [42]:

```
data.dropna(inplace=True)
```

In [43]:

```
data
```

Out[43]:

	id	item_id	dept_id	cat_id	store_id	state_id	units_sold	date	wm_yr_wk
6311430	HOBBIES_1_001_CA_1_validation	1437	3	1	0	0	0	2015-11-18	11542
6311431	HOBBIES_1_002_CA_1_validation	1438	3	1	0	0	1	2015-11-18	11542
6311432	HOBBIES_1_003_CA_1_validation	1439	3	1	0	0	0	2015-11-18	11542
6311433	HOBBIES_1_004_CA_1_validation	1440	3	1	0	0	1	2015-11-18	11542
6311434	HOBBIES_1_005_CA_1_validation	1441	3	1	0	0	0	2015-11-18	11542
...	...	...	...	...	...	...	...	...	...
11159335	FOODS_3_823_WI_3_validation	1432	2	0	9	2	1	2016-04-24	11613
11159336	FOODS_3_824_WI_3_validation	1433	2	0	9	2	0	2016-04-24	11613
11159337	FOODS_3_825_WI_3_validation	1434	2	0	9	2	0	2016-04-24	11613

	id	item_id	dept_id	cat_id	store_id	state_id	units_sold	date	wm_yr_wk
11159338	FOODS_3_826_WI_3_validation	1435	2	0	9	2	3	2016-04-24	11613
11159339	FOODS_3_827_WI_3_validation	1436	2	0	9	2	0	2016-04-24	11613

4846129 rows × 57 columns

Section 4: Model Evaluation

In [44]:
```
import time
import xgboost as xgb
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
```

In [45]:
```
# Separating X and y

x = data.drop(['id','units_sold','date'], axis=1)
y = data.units_sold

# Splitting into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

Model1: XGBoosting Model

In [46]:
```
model_xgb = xgb.XGBRegressor(seed=42, n_jobs=-1)

model_xgb.fit(x_train, y_train)
```

Out[46]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=-1,
             num_parallel_tree=1, predictor='auto', random_state=42,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=42,
             subsample=1, tree_method='exact', validate_parameters=1,
             verbosity=None)
```

In [47]:
```
y_pred = model_xgb.predict(x_test)
```

In [48]:
```
scores_train = cross_val_score(model_xgb, x_train, y_train, cv=3, n_jobs=6 ,scoring='r2').
scores_test = cross_val_score(model_xgb, x_test, y_test, cv=3, n_jobs=6 ,scoring='r2').mea
```

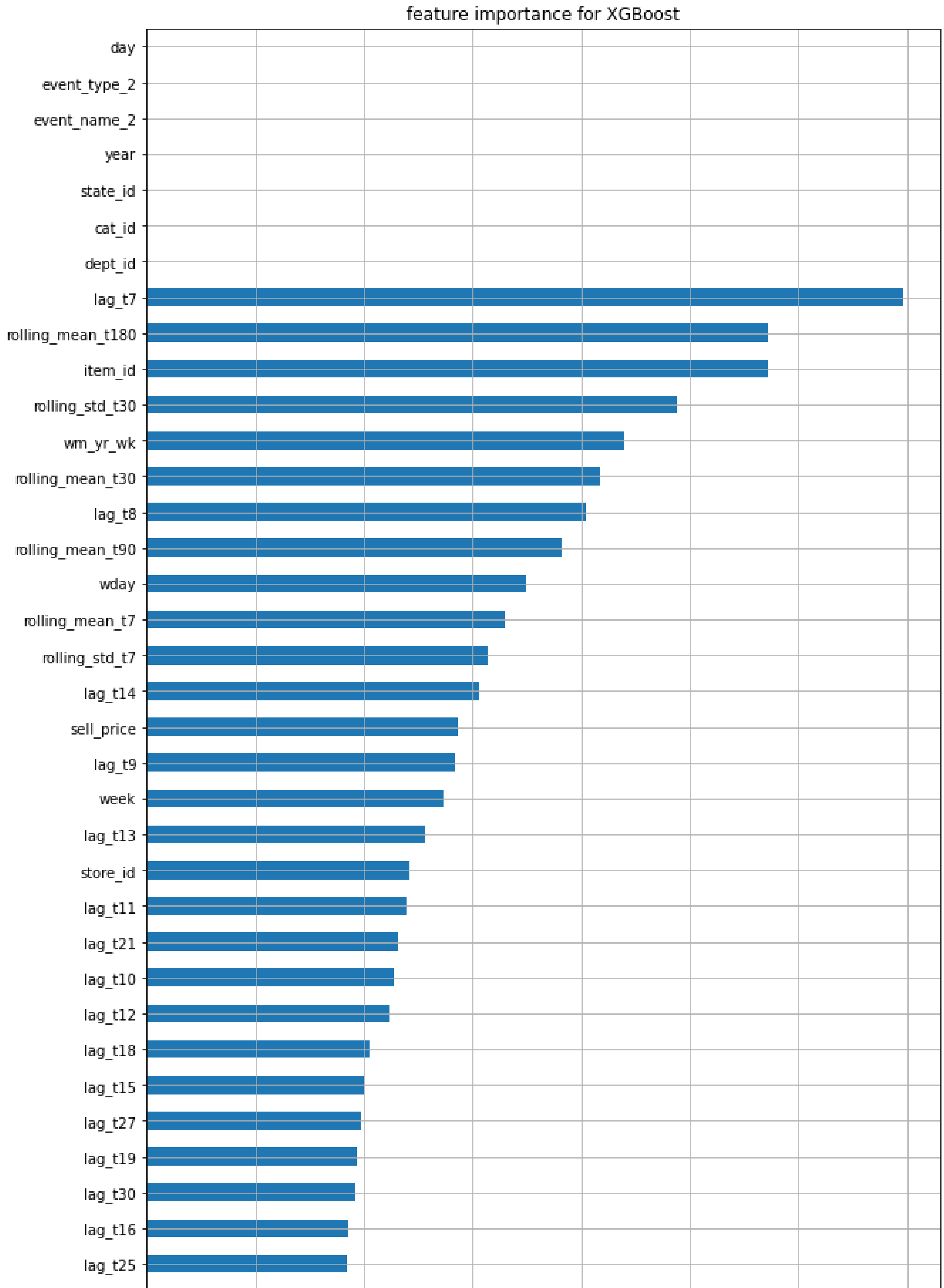
In [49]:
```
print("Training accuracy:%.4f" % scores_train.mean())
print("Test accuracy:%.4f" % scores_test.mean())
```

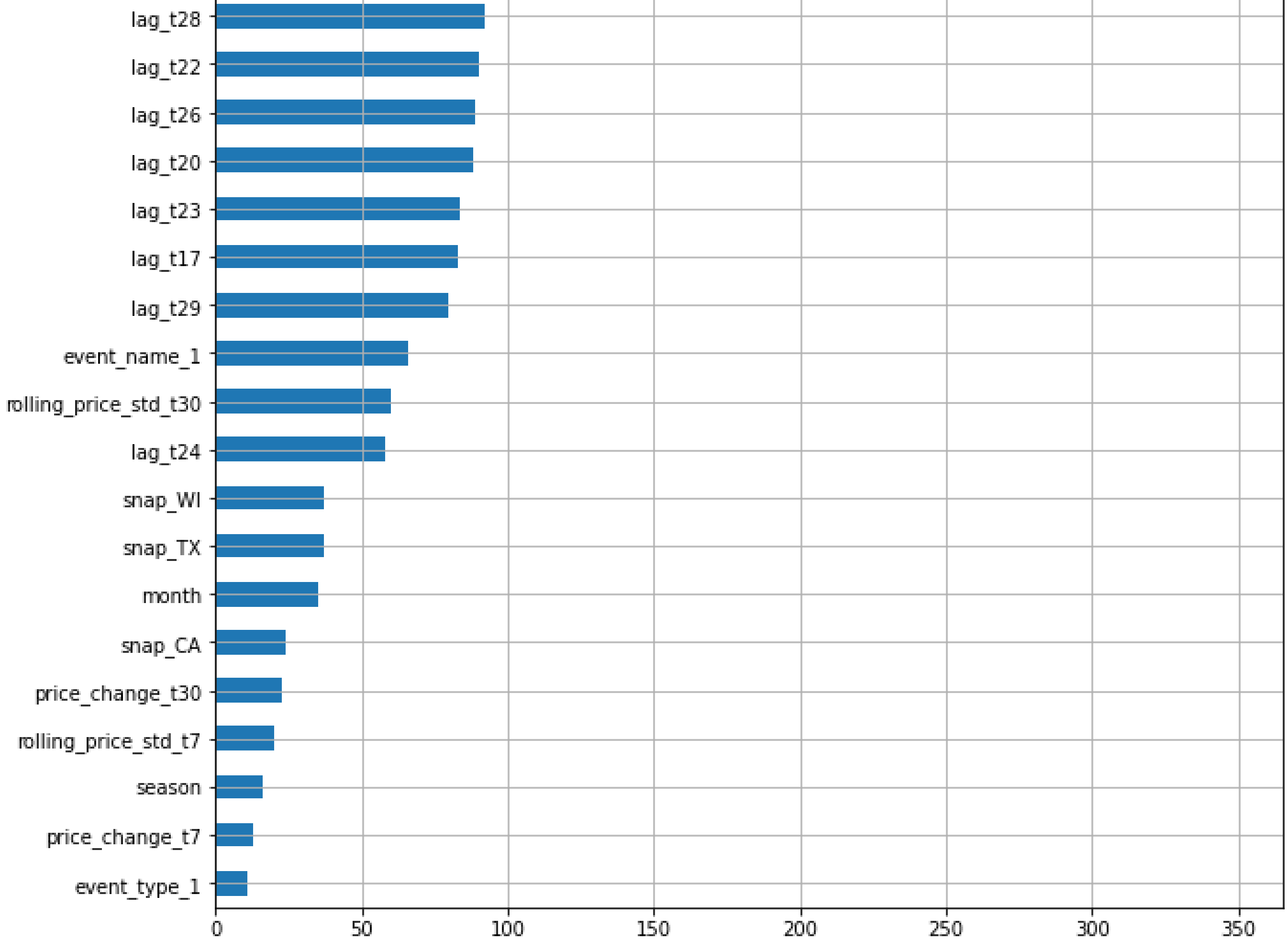
```
Training accuracy:0.6519
Test accuracy:0.6379
```

In [51]:
```
filename = 'data/xgb_model.pkl'
pickle.dump(model_xgb, open(filename, 'wb'))
```

In [52]:

```
plt.figure(figsize=(10,25))

feature_importance_xgb = pd.Series(model_xgb.get_booster().get_score(importance_type='weig
feature_importance_xgb.plot.barh()
plt.title('feature importance for XGBoost')
plt.grid()
```





Model2: Random Forest Model

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor

model_rf = RandomForestRegressor(n_estimators=100, n_jobs=-1)
model_rf.fit(x, y)
```

In [60]:

```
scores_train = cross_val_score(model_rf, x_train, y_train, cv=3, n_jobs=6 ,scoring='r2').r
scores_test = cross_val_score(model_rf, x_test, y_test, cv=3, n_jobs=6 ,scoring='r2').mear
```

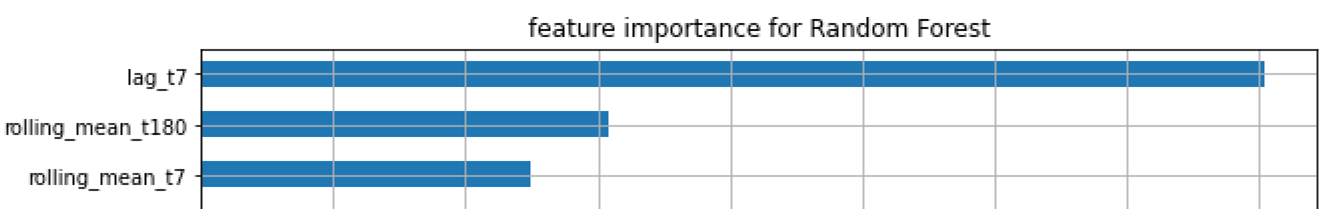
In [61]:

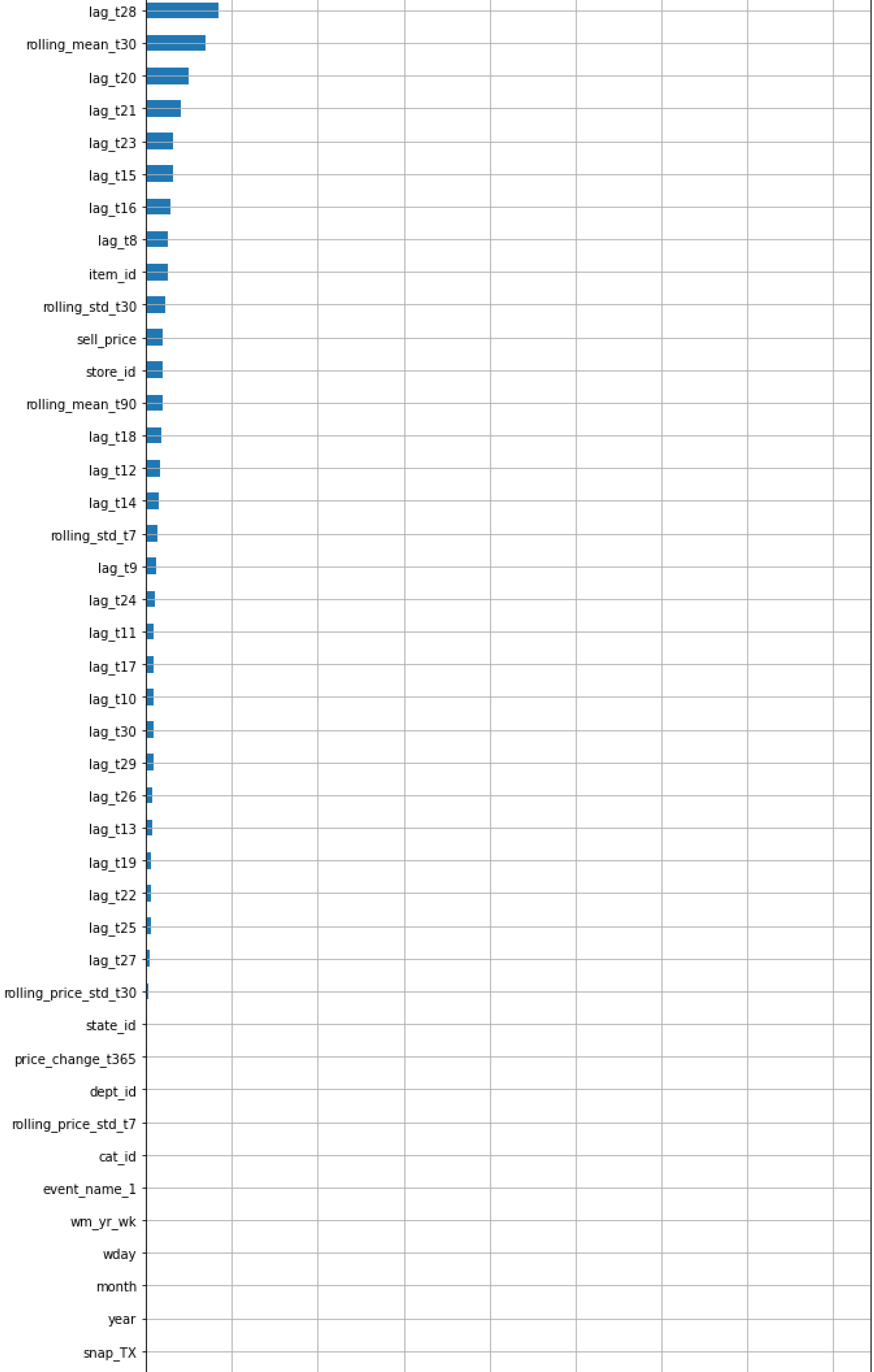
```
print("Training accuracy:%.4f" % scores_train.mean())
print("Test accuracy:%.4f" % scores_test.mean())
```

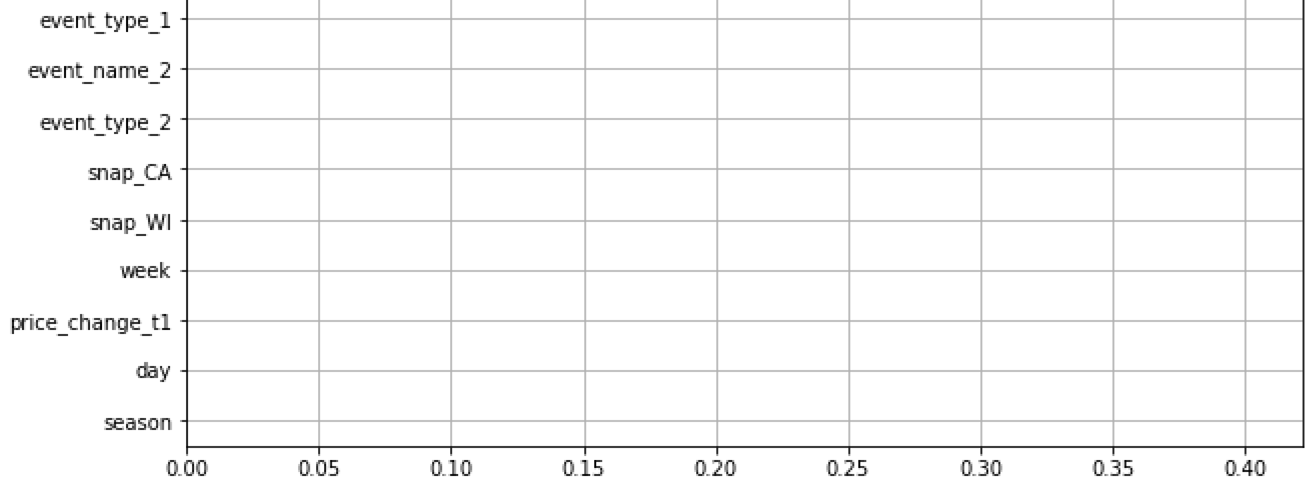
```
Training accuracy:0.7120
Test accuracy:0.6842
```

In [62]:

```
plt.figure(figsize=(10,25))

feature_importance_rf = pd.Series(model_rf.estimators_[0].feature_importances_, index = x_
feature_importance_rf.plot.barh()
plt.title('feature importance for Random Forest')
plt.grid()
```




lag_t28
rolling_mean_t30
lag_t20
lag_t21
lag_t23
lag_t15
lag_t16
lag_t8
item_id
rolling_std_t30
sell_price
store_id
rolling_mean_t90
lag_t18
lag_t12
lag_t14
rolling_std_t7
lag_t9
lag_t24
lag_t11
lag_t17
lag_t10
lag_t30
lag_t29
lag_t26
lag_t13
lag_t19
lag_t22
lag_t25
lag_t27
rolling_price_std_t30
state_id
price_change_t365
dept_id
rolling_price_std_t7
cat_id
event_name_1
wm_yr_wk
wday
month
year
snap_TX



From both of the model, i will go in with Random Forest since the initial accuracy (R2) is better than XGBoost

Model Refining

To improve the model performance, I decided to use GridsearchCV to tuning for the best parameter on Random Forest model

In [64]:

```
param_grid={'max_features': ['auto', 'sqrt'],
            'max_depth' : [5,10,15,20],
            'min_samples_split': [2, 10, 20],
            'min_samples_leaf': [2, 10,15, 20],
            }
gcv = GridSearchCV(RandomForestRegressor(random_state=12,n_jobs=-1), param_grid)
gcv.fit(x_train, y_train)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
C:\Users\PHONGS~1\AppData\Local\Temp/ipykernel_20916/3192033912.py in <module>
      5                 }
      6 gcv = GridSearchCV(RandomForestRegressor(random_state=12,n_jobs=-1), param_grid)
----> 7 gcv.fit(x_train, y_train)

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64 
     65             # extra_args > 0

~\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py in fit(self, X, y, group
s, **fit_params)
    839                 return results
    840 
--> 841             self._run_search(evaluate_candidates)
    842 
    843             # multimetric is determined here because in the case of a callable

~\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py in _run_search(self, eval
uate_candidates)
   1294     def _run_search(self, evaluate_candidates):
   1295         """Search all candidates in param_grid"""
-> 1296         evaluate_candidates(ParameterGrid(self.param_grid))
   1297 
   1298 

~\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py in evaluate_candidates(ca
ndidate_params, cv, more_results)
```

```
    807                                     (split_idx, (train, test)) in product(
    808                                     enumerate(candidate_params),
--> 809                                     enumerate(cv.split(X, y, groups))))
    810 
    811                 if len(out) < 1:

~\Anaconda3\lib\site-packages\joblib\parallel.py in __call__(self, iterable)
   1042                 self._iterating = self._original_iterator is not None
   1043 
-> 1044             while self.dispatch_one_batch(iterator):
   1045                 pass
   1046 

~\Anaconda3\lib\site-packages\joblib\parallel.py in dispatch_one_batch(self, iterator)
    857                 return False
    858             else:
--> 859                 self._dispatch(tasks)
    860                 return True
    861 

~\Anaconda3\lib\site-packages\joblib\parallel.py in _dispatch(self, batch)
    775         with self._lock:
    776             job_idx = len(self._jobs)
--> 777             job = self._backend.apply_async(batch, callback=cb)
    778             # A job can complete so quickly than its callback is
    779             # called before we get here, causing self._jobs to

~\Anaconda3\lib\site-packages\joblib\_parallel_backends.py in apply_async(self, func, call
back)
    206     def apply_async(self, func, callback=None):
    207         """Schedule a func to be run"""
--> 208         result = ImmediateResult(func)
    209         if callback:
    210             callback(result)

~\Anaconda3\lib\site-packages\joblib\_parallel_backends.py in __init__(self, batch)
    570         # Don't delay the application, to avoid keeping the input
    571         # arguments in memory
--> 572         self.results = batch()
    573 
    574     def get(self):

~\Anaconda3\lib\site-packages\joblib\parallel.py in __call__(self)
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
    262             return [func(*args, **kwargs)
--> 263                     for func, args, kwargs in self.items]
    264 
    265     def __reduce__(self):

~\Anaconda3\lib\site-packages\joblib\parallel.py in <listcomp>(.0)
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
    262             return [func(*args, **kwargs)
--> 263                     for func, args, kwargs in self.items]
    264 
    265     def __reduce__(self):

~\Anaconda3\lib\site-packages\sklearn\utils\fixes.py in __call__(self, *args, **kwargs)
    220     def __call__(self, *args, **kwargs):
    221         with config_context(**self.config):
--> 222             return self.function(*args, **kwargs)

~\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py in _fit_and_score(est
imator, X, y, scorer, train, test, verbose, parameters, fit_params, return_train_score, re
turn_parameters, return_n_test_samples, return_times, return_estimator, split_progress, ca
ndidate_progress, error_score)
    596             estimator.fit(X_train, **fit_params)
```

```
    597             else:
--> 598                 estimator.fit(X_train, y_train, **fit_params)
    599 
    600         except Exception as e:

~\Anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in fit(self, X, y, sample_weigh
t)
    391                     verbose=self.verbose, class_weight=self.class_weight,
    392                     n_samples_bootstrap=n_samples_bootstrap)
--> 393                 for i, t in enumerate(trees))
    394 
    395             # Collect newly grown trees

~\Anaconda3\lib\site-packages\joblib\parallel.py in __call__(self, iterable)
   1052 
   1053             with self._backend.retrieval_context():
-> 1054                 self.retrieve()
   1055             # Make sure that we get a last message telling us we are done
   1056             elapsed_time = time.time() - self._start_time

~\Anaconda3\lib\site-packages\joblib\parallel.py in retrieve(self)
    931             try:
    932                 if getattr(self._backend, 'supports_timeout', False):
--> 933                     self._output.extend(job.get(timeout=self.timeout))
    934                 else:
    935                     self._output.extend(job.get())

~\Anaconda3\lib\site-packages\joblib\_parallel_backends.py in wrap_future_result(future, t
imeout)
    540         AsyncResults.get from multiprocessing."""
    541         try:
--> 542             return future.result(timeout=timeout)
    543         except CfTimeoutError as e:
    544             raise TimeoutError from e

~\Anaconda3\lib\concurrent\futures\_base.py in result(self, timeout)
    425                 return self.__get_result()
    426 
--> 427             self._condition.wait(timeout)
    428 
    429             if self._state in [CANCELLED, CANCELLED_AND_NOTIFIED]:

~\Anaconda3\lib\threading.py in wait(self, timeout)
    294         try:    # restore state no matter what (e.g., KeyboardInterrupt)
    295             if timeout is None:
--> 296                 waiter.acquire()
    297                 gotit = True
    298             else:

KeyboardInterrupt: 
```

In [ ]:

```
gcv.best_params_
```

In [ ]:

```
param_grid={'max_features': ['auto', 'sqrt'],
                'max_depth' : [20, 50 ,100],
                'min_samples_split': [2, 10],
                'min_samples_leaf': [2, 10],
                }
gcv = GridSearchCV(RandomForestRegressor(random_state=12,n_jobs=-1), param_grid)
gcv.fit(x_train, y_train)
```

In [ ]:

```
gcv.best_params_
```

In [133…

```
model_rf2 = RandomForestRegressor(n_estimators=100,max_depth=20,max_features='auto',min_sa
model_rf2.fit(x_train, y_train)
```

Out[133…

```
RandomForestRegressor(max_depth=20, min_samples_leaf=2, n_jobs=-1,
                      oob_score=True)
```

In [134…

```
scores_train = cross_val_score(model_rf2, x_train, y_train, cv=10, n_jobs=-1 ,scoring='r2'
scores_test = cross_val_score(model_rf2, x_test, y_test, cv=10, n_jobs=-1 ,scoring='r2').r
```

In [135…

```
print("Training accuracy:%.4f" % scores_train.mean())
print("Test accuracy:%.4f" % scores_test.mean())
```

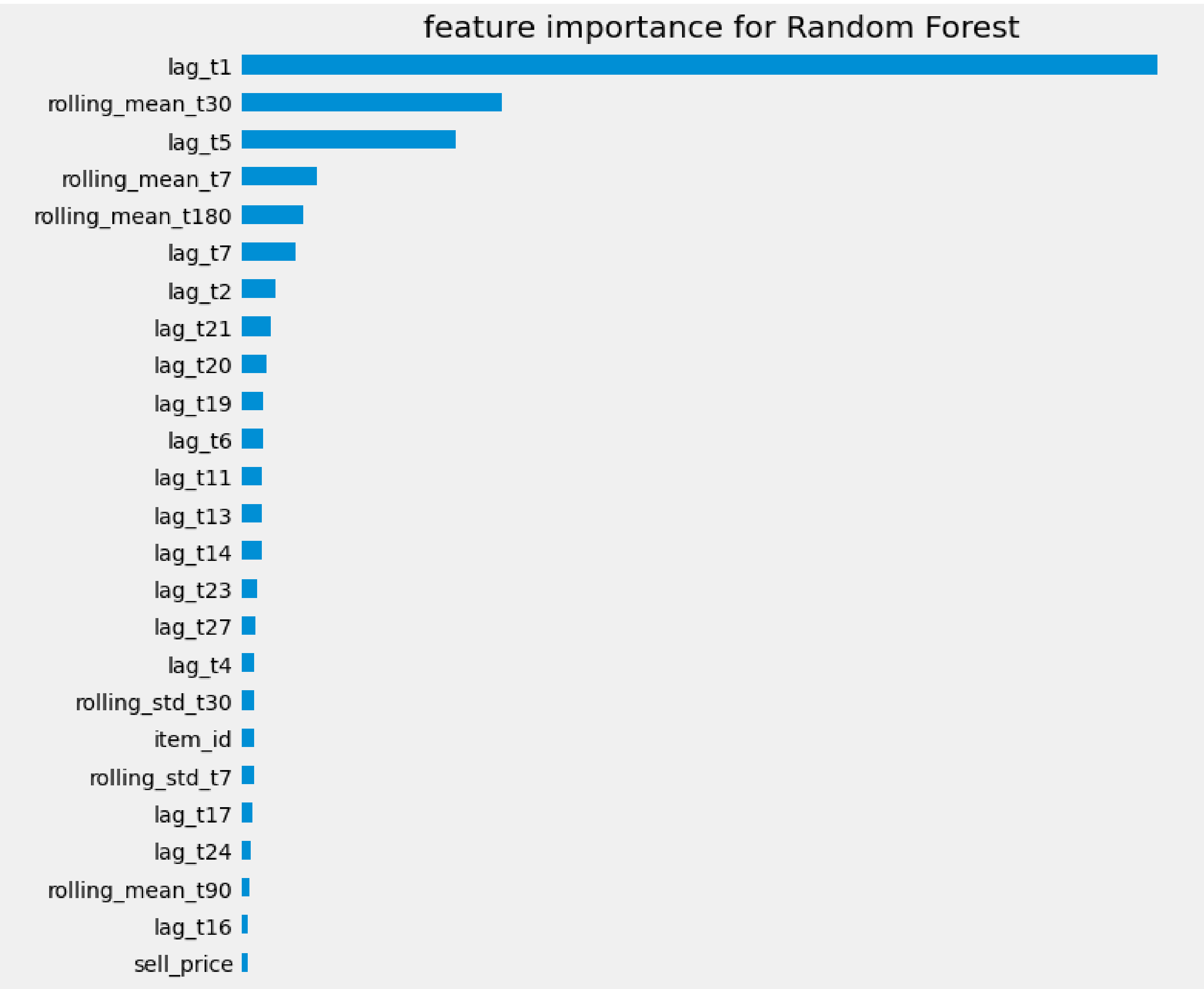
```
Training accuracy:0.7434
Test accuracy:0.7119
```

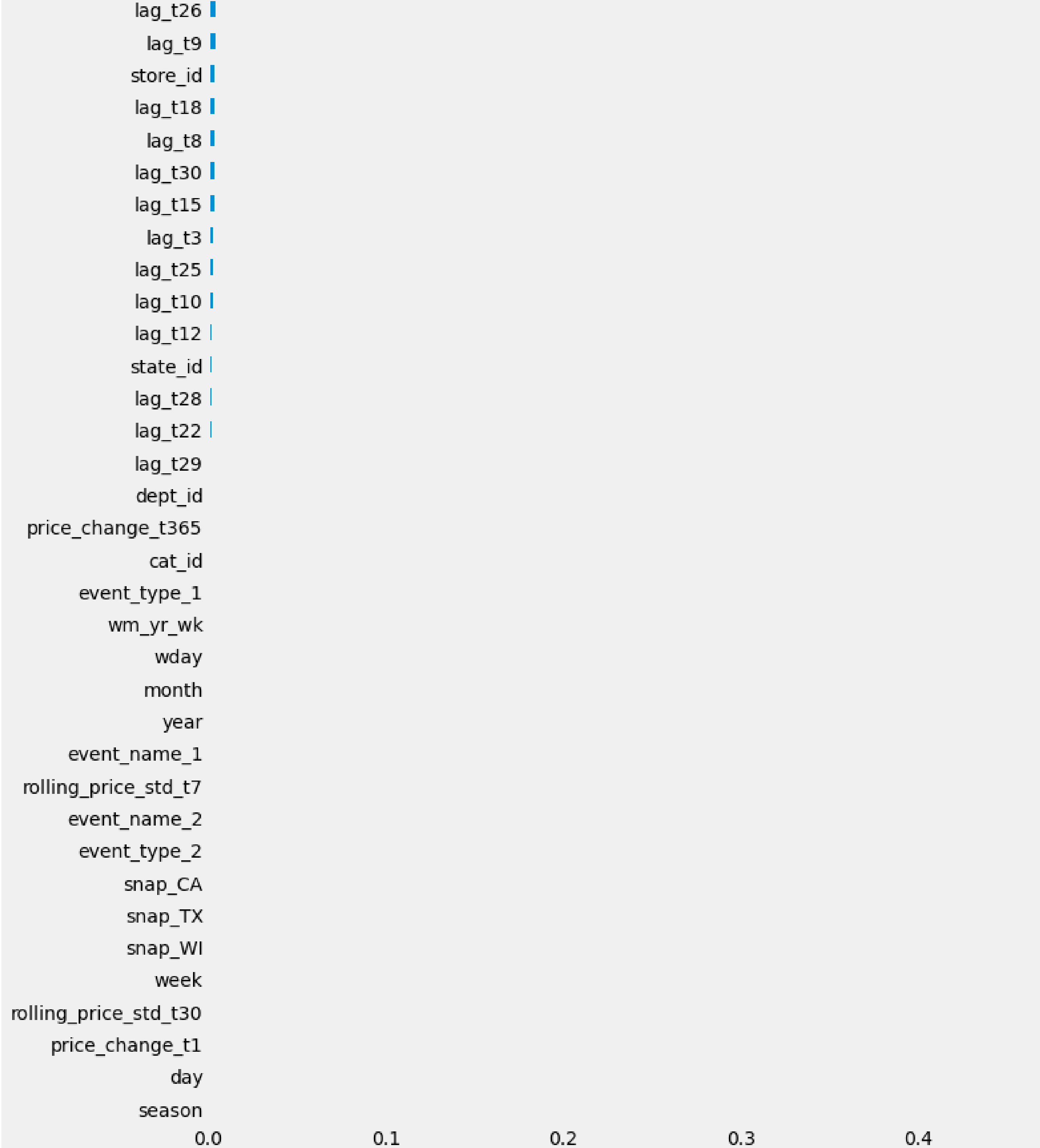
In [137…

```
y_pred_rf2 = model_rf2.predict(x_test)
```

In [136…

```
plt.figure(figsize=(10,25))

feature_importance_rf = pd.Series(model_rf2.estimators_[0].feature_importances_, index = x
feature_importance_rf.plot.barh()
plt.title('feature importance for Random Forest')
plt.grid()
```

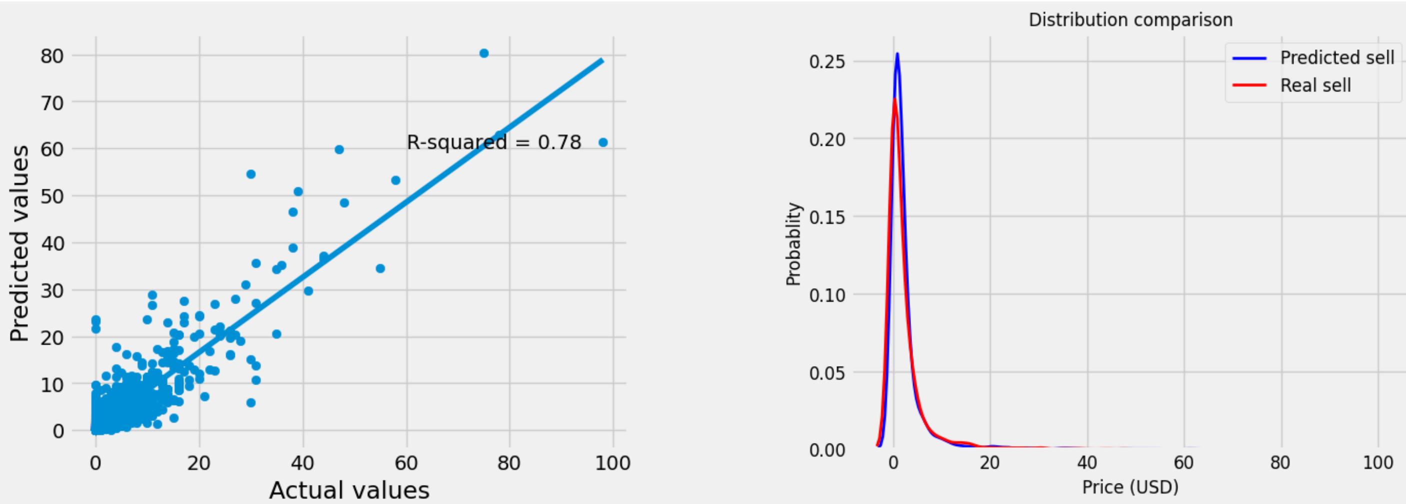




In [147…

```
fig = plt.figure(figsize =(15, 5),dpi=100)
fig.subplots_adjust(hspace=0.4, wspace=0.4)
ax = plt.axes(aspect = 'equal')

plt.subplot(121)
y_pred_out = y_pred_rf2
r_squared = (r2_score(y_test, y_pred_out))
plt.scatter(y_test,y_pred_out)
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
plt.plot(np.unique(y_test), np.poly1d(np.polyfit(y_test,y_pred_out, 1))(np.unique(y_test))
plt.text(60, 60 ,'R-squared = %0.2f' % r_squared)


plt.subplot(122)
```

```
sns.distplot(y_pred_out, hist=False,
             kde_kws={'color': 'b', 'lw': 2, 'label': 'Predicted price'})
sns.distplot(y_test, hist=False,
             kde_kws={'color': 'r', 'lw': 2, 'label': 'Real price'})
plt.title('Distribution comparison', fontsize=12)
plt.ylabel('Probablity', fontsize=12)
plt.xlabel('Price (USD)', fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.legend(['Predicted sell', 'Real sell'], prop={"size":12})


plt.show()
```



Section 5: Business Impact / Insights

To answer my business questions is:

What are the main features influencing the sales of product in Walmart?

To summarize, from the prediction result we could see that the feature importances of the food product sales is:

- 1) lag_t7 (40%)
- 2) rolling_mean_t180 (16%)
- 3) rolling_mean_t7 (12.5%)
- 4) lag_t28 (4%)
- 5) rolling_mean_t30 (3%)

This could imply that the most importance features would fall in the time series categories such as amount of product sold last 7, 28 days, moving average of unit sell 7, 30, 180 days. From features that not related to time, item_id sell_price and store_id come to account that Walmart should focus on.

This model will benefits in multiple ways, for example:

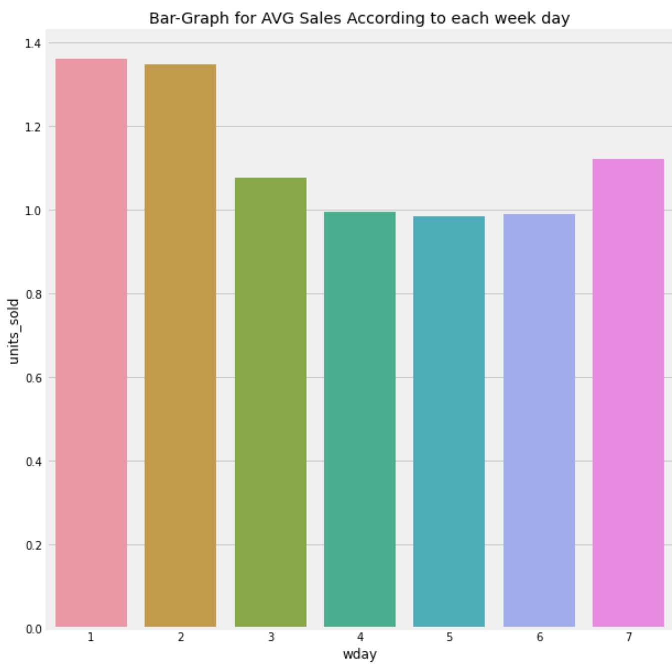
- Stocking - it will helps the store retailers to maintain there Stocks according to demand they are expecting
- Creating a promotion for time to time, weekly, event, etc.

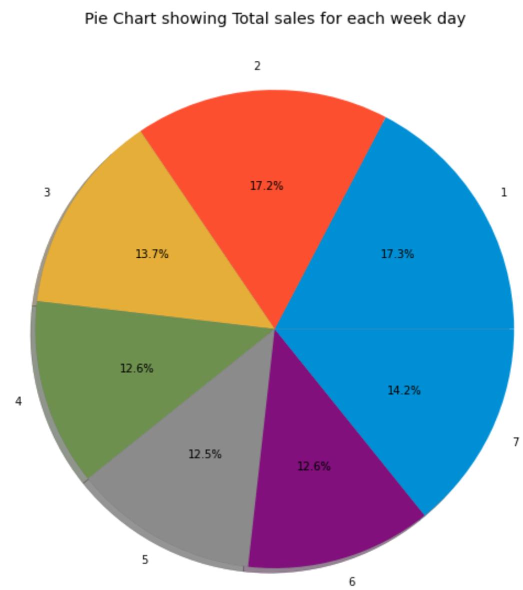
Is different month and week day affect the sales of the product?

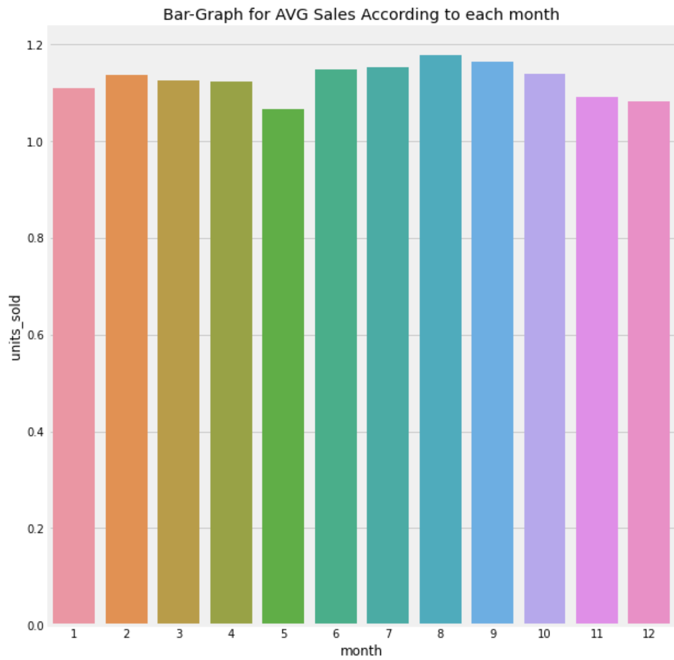
In [324…

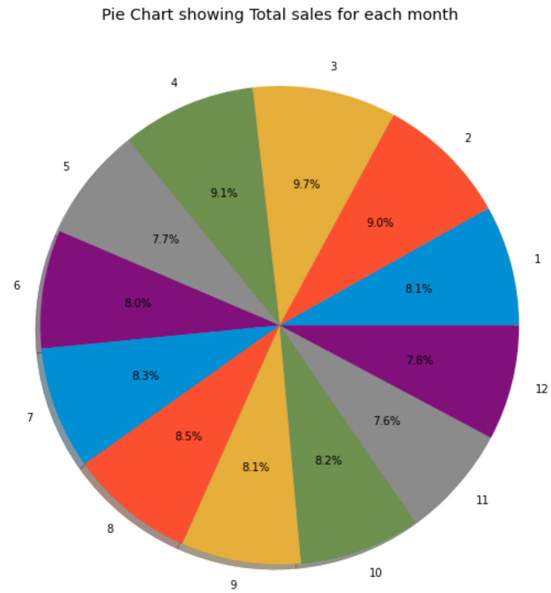
```
df=data.groupby('wday').mean()
df.reset_index(level=0,inplace=True)
plt.figure(figsize=(20,10))
plt.subplot(121)
```

```
sns.barplot(x='wday',y='units_sold',data=df)
plt.title("Bar-Graph for AVG Sales According to each week day")

df=data.groupby('wday').sum()
df.reset_index(level=0,inplace=True)
df['perc']=df['units_sold']/sum(df['units_sold'].values)*100
plt.subplot(122)
plt.pie(df['perc'].values,labels=df['wday'].values,shadow=True,autopct='%1.1f%%')
plt.title("Pie Chart showing Total sales for each week day")

df=data.groupby('month').mean()
df.reset_index(level=0,inplace=True)
plt.figure(figsize=(20,10))
plt.subplot(121)
sns.barplot(x='month',y='units_sold',data=df)
plt.title("Bar-Graph for AVG Sales According to each month")

df=data.groupby('month').sum()
df.reset_index(level=0,inplace=True)
df['perc']=df['units_sold']/sum(df['units_sold'].values)*100
plt.subplot(122)
plt.pie(df['perc'].values,labels=df['month'].values,shadow=True,autopct='%1.1f%%')
plt.title("Pie Chart showing Total sales for each month")

plt.show()
```









According to the chart, For weekdays, we can see that on the weekend especially Saturday(1), Sunday(2) are the days that have higher average sales than others. In the time dimension, we could imply that if Walmart wants to contribute the higher product sales on weekdays, they could use the campaign strategies to influence the customer to come and buy the product on weekdays. For the month chart as well, the customer tend to buy a product in Q3 more than other quarters.

Future Improvement Work

To improve the model performance, we could

- Getting more data, since, very unfortunately I have to remove 4 years of historical data due to memory lacking, this could lead to huge amount of error
- Reduce the overfitting by better tunning on hyperparameters
- Try other model such as random forest or deep learning to improve the accuracy
- Clustering technique to get more insightful information from the data.

In [ ]:

```

```

In [1]:

```
!pip install -U notebook-as-pdf
```

```
Collecting notebook-as-pdf
  Downloading notebook_as_pdf-0.5.0-py3-none-any.whl (6.5 kB)
Collecting pyppeteer
  Downloading pyppeteer-1.0.2-py3-none-any.whl (83 kB)
Requirement already satisfied: nbconvert in c:\users\phongsapon\anaconda3\lib\site-package
s (from notebook-as-pdf) (6.1.0)
Collecting PyPDF2
  Downloading PyPDF2-1.26.0.tar.gz (77 kB)
Requirement already satisfied: traitlets>=5.0 in c:\users\phongsapon\anaconda3\lib\site-pa
ckages (from nbconvert->notebook-as-pdf) (5.1.0)
Requirement already satisfied: testpath in c:\users\phongsapon\anaconda3\lib\site-packages
(from nbconvert->notebook-as-pdf) (0.5.0)
Requirement already satisfied: mistune<2,>=0.8.1 in c:\users\phongsapon\anaconda3\lib\site
-packages (from nbconvert->notebook-as-pdf) (0.8.4)
Requirement already satisfied: pandocfilters>=1.4.1 in c:\users\phongsapon\anaconda3\lib\s
ite-packages (from nbconvert->notebook-as-pdf) (1.4.3)
Requirement already satisfied: jupyter-core in c:\users\phongsapon\anaconda3\lib\site-pack
```

```
ages (from nbconvert->notebook-as-pdf) (4.8.1)
Requirement already satisfied: pygments>=2.4.1 in c:\users\phongsapon\anaconda3\lib\site-p
ackages (from nbconvert->notebook-as-pdf) (2.10.0)
Requirement already satisfied: nbclient<0.6.0,>=0.5.0 in c:\users\phongsapon\anaconda3\lib
\site-packages (from nbconvert->notebook-as-pdf) (0.5.3)
Requirement already satisfied: defusedxml in c:\users\phongsapon\anaconda3\lib\site-packag
es (from nbconvert->notebook-as-pdf) (0.7.1)
Requirement already satisfied: bleach in c:\users\phongsapon\anaconda3\lib\site-packages
(from nbconvert->notebook-as-pdf) (4.0.0)
Requirement already satisfied: nbformat>=4.4 in c:\users\phongsapon\anaconda3\lib\site-pac
kages (from nbconvert->notebook-as-pdf) (5.1.3)
Requirement already satisfied: entrypoints>=0.2.2 in c:\users\phongsapon\anaconda3\lib\sit
e-packages (from nbconvert->notebook-as-pdf) (0.3)
Requirement already satisfied: jinja2>=2.4 in c:\users\phongsapon\anaconda3\lib\site-packa
ges (from nbconvert->notebook-as-pdf) (3.0.1)
Requirement already satisfied: jupyterlab-pygments in c:\users\phongsapon\anaconda3\lib\si
te-packages (from nbconvert->notebook-as-pdf) (0.1.2)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\phongsapon\anaconda3\lib\site-p
ackages (from jinja2>=2.4->nbconvert->notebook-as-pdf) (2.0.1)
Requirement already satisfied: async-generator in c:\users\phongsapon\anaconda3\lib\site-p
ackages (from nbclient<0.6.0,>=0.5.0->nbconvert->notebook-as-pdf) (1.10)
Requirement already satisfied: nest-asyncio in c:\users\phongsapon\anaconda3\lib\site-pack
ages (from nbclient<0.6.0,>=0.5.0->nbconvert->notebook-as-pdf) (1.5.1)
Requirement already satisfied: jupyter-client>=6.1.5 in c:\users\phongsapon\anaconda3\lib
\site-packages (from nbclient<0.6.0,>=0.5.0->nbconvert->notebook-as-pdf) (7.0.1)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\phongsapon\anaconda3\lib\s
ite-packages (from jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert->notebook-as-p
df) (2.8.2)
Requirement already satisfied: tornado>=4.1 in c:\users\phongsapon\anaconda3\lib\site-pack
ages (from jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert->notebook-as-pdf) (6.
1)
Requirement already satisfied: pyzmq>=13 in c:\users\phongsapon\anaconda3\lib\site-package
s (from jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert->notebook-as-pdf) (22.2.
1)
Requirement already satisfied: pywin32>=1.0 in c:\users\phongsapon\appdata\roaming\python
\python37\site-packages (from jupyter-core->nbconvert->notebook-as-pdf) (301)
Requirement already satisfied: ipython-genutils in c:\users\phongsapon\anaconda3\lib\site-
packages (from nbformat>=4.4->nbconvert->notebook-as-pdf) (0.2.0)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in c:\users\phongsapon\anaconda3\li
b\site-packages (from nbformat>=4.4->nbconvert->notebook-as-pdf) (3.2.0)
Requirement already satisfied: attrs>=17.4.0 in c:\users\phongsapon\anaconda3\lib\site-pac
kages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert->notebook-as-pdf) (21.2.0)
Requirement already satisfied: six>=1.11.0 in c:\users\phongsapon\anaconda3\lib\site-packa
ges (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert->notebook-as-pdf) (1.16.0)
Requirement already satisfied: setuptools in c:\users\phongsapon\anaconda3\lib\site-packag
es (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert->notebook-as-pdf) (58.0.4)
Requirement already satisfied: pyrsistent>=0.14.0 in c:\users\phongsapon\anaconda3\lib\sit
e-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert->notebook-as-pdf) (0.1
7.3)
Requirement already satisfied: importlib-metadata in c:\users\phongsapon\anaconda3\lib\sit
e-packages (from jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert->notebook-as-pdf) (4.8.
1)
Requirement already satisfied: webencodings in c:\users\phongsapon\anaconda3\lib\site-pack
ages (from bleach->nbconvert->notebook-as-pdf) (0.5.1)
Requirement already satisfied: packaging in c:\users\phongsapon\anaconda3\lib\site-package
s (from bleach->nbconvert->notebook-as-pdf) (21.0)
Requirement already satisfied: typing-extensions>=3.6.4 in c:\users\phongsapon\anaconda3\l
ib\site-packages (from importlib-metadata->jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconve
rt->notebook-as-pdf) (3.10.0.2)
Requirement already satisfied: zipp>=0.5 in c:\users\phongsapon\anaconda3\lib\site-package
s (from importlib-metadata->jsonschema!=2.5.0,>=2.4->nbformat>=4.4->nbconvert->notebook-as
-pdf) (3.6.0)
Requirement already satisfied: pyparsing>=2.0.2 in c:\users\phongsapon\anaconda3\lib\site-
packages (from packaging->bleach->nbconvert->notebook-as-pdf) (2.4.7)
Requirement already satisfied: urllib3<2.0.0,>=1.25.8 in c:\users\phongsapon\anaconda3\lib
\site-packages (from pyppeteer->notebook-as-pdf) (1.26.7)
```

```
Collecting websockets<11.0,>=10.0
  Downloading websockets-10.1-cp37-cp37m-win_amd64.whl (97 kB)
Requirement already satisfied: tqdm<5.0.0,>=4.42.1 in c:\users\phongsapon\anaconda3\lib\si
te-packages (from pyppeteer->notebook-as-pdf) (4.62.3)
Collecting pyee<9.0.0,>=8.1.0
  Downloading pyee-8.2.2-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: certifi>=2021 in c:\users\phongsapon\anaconda3\lib\site-pac
kages (from pyppeteer->notebook-as-pdf) (2021.10.8)
Requirement already satisfied: appdirs<2.0.0,>=1.4.3 in c:\users\phongsapon\anaconda3\lib
\site-packages (from pyppeteer->notebook-as-pdf) (1.4.4)
Requirement already satisfied: colorama in c:\users\phongsapon\anaconda3\lib\site-packages
(from tqdm<5.0.0,>=4.42.1->pyppeteer->notebook-as-pdf) (0.4.4)
Building wheels for collected packages: PyPDF2
  Building wheel for PyPDF2 (setup.py): started
  Building wheel for PyPDF2 (setup.py): finished with status 'done'
  Created wheel for PyPDF2: filename=PyPDF2-1.26.0-py3-none-any.whl size=61101 sha256=02b5
452257043a0af55abb1c785436ca64f159395cc73acede737e66eb18667e
  Stored in directory: c:\users\phongsapon\appdata\local\pip\cache\wheels\80\1a\24\648467a
de3a77ed20f35cfd2badd32134e96dd25ca811e64b3
Successfully built PyPDF2
Installing collected packages: websockets, pyee, pyppeteer, PyPDF2, notebook-as-pdf
Successfully installed PyPDF2-1.26.0 notebook-as-pdf-0.5.0 pyee-8.2.2 pyppeteer-1.0.2 webs
ockets-10.1

WARNING: Ignoring invalid distribution -umpy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -cipy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -umpy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -cipy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -umpy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -cipy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -umpy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -cipy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -umpy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -cipy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -umpy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -cipy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -umpy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -cipy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -umpy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -cipy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -umpy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -cipy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -umpy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -cipy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
WARNING: Ignoring invalid distribution -umpy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
```

```
WARNING: Ignoring invalid distribution -cipy (c:\users\phongsapon\anaconda3\lib\site-packa
ges)
```

In [ ]:

```

```