TASK #1: UNDERSTAND THE PROBLEM STATEMENT/GOAL

- This dataset contains weekly sales from 99 departments belonging to 45 different stores.
- Our aim is to forecast weekly sales from a particular department.
- The objective of this case study is to forecast weekly retail store sales based on historical data.
- The data contains holidays and promotional markdowns offered by various stores and several departments throughout the year.
- Markdowns are crucial to promote sales especially before key events such as Super Bowl, Christmas and Thanksgiving.
- Developing accurate model will enable make informed decisions and make recommendations to improve business processes in the future.
- The data consists of three sheets:
    - Stores
    - Features
    - Sales
- Data Source : [https://www.kaggle.com/manjeetsingh/retaildataset (https://www.kaggle.com/manjeetsingh/retaildataset)](https://www.kaggle.com/manjeetsingh/retaildataset)

TASK #2: IMPORT DATASET AND LIBRARIES

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import zipfile
```

In [2]:
```
# import the csv files using pandas
feature = pd.read_csv('Features_data_set.csv')
sales = pd.read_csv('sales_data_set.csv')
stores = pd.read_csv('stores_data_set.csv')
```

In [3]:
```
# Let's explore the 3 dataframes
# "stores" dataframe contains information related to the 45 stores such as type and size of store.

stores
```

Out[3]:

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875
5	6	A	202505
6	7	B	70713
7	8	A	155078
8	9	B	125833
9	10	B	126512
10	11	A	207499
11	12	B	112238
12	13	A	219622
13	14	A	200898
14	15	B	123737
15	16	B	57197
16	17	B	93188
17	18	B	120653
18	19	A	203819
19	20	A	203742
20	21	B	140167
21	22	B	119557
22	23	B	114533

	Store	Type	Size
23	24	A	203819
24	25	B	128107
25	26	A	152513
26	27	A	204184
27	28	A	206302
28	29	B	93638
29	30	C	42988
30	31	A	203750
31	32	A	203007
32	33	A	39690
33	34	A	158114
34	35	B	103681
35	36	A	39910
36	37	C	39910
37	38	C	39690
38	39	A	184109
39	40	A	155083
40	41	A	196321
41	42	C	39690
42	43	C	41062
43	44	C	39910
44	45	B	118221

In [4]:
```
# Let's explore the "feature" dataframe
# Features dataframe contains additional data related to the store, department, and regional activity for the gi
# Store: store number
# Date: week
# Temperature: average temperature in the region
# Fuel_Price: cost of fuel in the region
# MarkDown1-5: anonymized data related to promotional markdowns.
# CPI: consumer price index
# Unemployment: unemployment rate
# IsHoliday: whether the week is a special holiday week or not

feature
```

Out[4]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemploym
0	1	05/02/2010	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.
1	1	12/02/2010	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.
2	1	19/02/2010	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.
3	1	26/02/2010	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.
4	1	05/03/2010	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.
...	...	...	...	...	...	...	...	...	...	...	
8185	45	28/06/2013	76.05	3.639	4842.29	975.03	3.00	2449.97	3169.69	NaN	N
8186	45	05/07/2013	77.50	3.614	9090.48	2268.58	582.74	5797.47	1514.93	NaN	N
8187	45	12/07/2013	79.37	3.614	3789.94	1827.31	85.72	744.84	2150.36	NaN	N
8188	45	19/07/2013	82.84	3.737	2961.49	1047.07	204.19	363.00	1059.46	NaN	N
8189	45	26/07/2013	76.06	3.804	212.02	851.73	2.06	10.88	1864.57	NaN	N

8190 rows × 12 columns

In [5]:
```
# Let's explore the "sales" dataframe
# "Sales" dataframe contains historical sales data, which covers 2010-02-05 to 2012-11-01.
# Store: store number
# Dept: department number
# Date: the week
# Weekly_Sales: sales for the given department in the given store
# IsHoliday: whether the week is a special holiday week

sales
```

Out[5]:

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	05/02/2010	24924.50	False
1	1	1	12/02/2010	46039.49	True
2	1	1	19/02/2010	41595.55	False
3	1	1	26/02/2010	19403.54	False
4	1	1	05/03/2010	21827.90	False
...	...	...	...	...	...
421565	45	98	28/09/2012	508.37	False
421566	45	98	05/10/2012	628.10	False
421567	45	98	12/10/2012	1061.02	False
421568	45	98	19/10/2012	760.01	False
421569	45	98	26/10/2012	1076.80	False

421570 rows × 5 columns

TASK #3: EXPLORE INDIVIDUAL DATASET

MINI CHALLENGE

- Use info and describe to individually explore the 3 dataframes
- What is the maximum fuel price? and maximum unemployment numbers?

- What is the average size of the stores?

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [6]:
```
# Change the datatype of 'date' column

feature['Date'] = pd.to_datetime(feature['Date'])
sales['Date'] = pd.to_datetime(sales['Date'])
```

In [7]:
```
feature
```

Out[7]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment
0	1	2010-05-02	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106
1	1	2010-12-02	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106
4	1	2010-05-03	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106
...	...	...	...	...	...	...	...	...	...	...	...
8185	45	2013-06-28	76.05	3.639	4842.29	975.03	3.00	2449.97	3169.69	NaN	NaN
8186	45	2013-05-07	77.50	3.614	9090.48	2268.58	582.74	5797.47	1514.93	NaN	NaN
8187	45	2013-12-07	79.37	3.614	3789.94	1827.31	85.72	744.84	2150.36	NaN	NaN
8188	45	2013-07-19	82.84	3.737	2961.49	1047.07	204.19	363.00	1059.46	NaN	NaN
8189	45	2013-07-26	76.06	3.804	212.02	851.73	2.06	10.88	1864.57	NaN	NaN

8190 rows × 12 columns

In [8]:
```
sales
```

Out[8]:

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-05-02	24924.50	False
1	1	1	2010-12-02	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-05-03	21827.90	False
...	...	...	...	...	...
421565	45	98	2012-09-28	508.37	False
421566	45	98	2012-05-10	628.10	False
421567	45	98	2012-12-10	1061.02	False
421568	45	98	2012-10-19	760.01	False
421569	45	98	2012-10-26	1076.80	False

421570 rows × 5 columns

TASK #4: MERGE DATASET INTO ONE DATAFRAME

In [9]:
```
sales.head()
```

Out[9]:

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-05-02	24924.50	False
1	1	1	2010-12-02	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-05-03	21827.90	False

In [10]:
```
feature.head()
```

Out[10]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsH
0	1	2010-05-02	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	
1	1	2010-12-02	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	
4	1	2010-05-03	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	

In [11]:
```
df = pd.merge(sales, feature, on = ['Store','Date','IsHoliday'])
```

In [12]:
```
df
```

Out[12]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkD
0	1	1	2010-05-02	24924.50	False	42.31	2.572	NaN	NaN	NaN	NaN	
1	1	2	2010-05-02	50605.27	False	42.31	2.572	NaN	NaN	NaN	NaN	
2	1	3	2010-05-02	13740.12	False	42.31	2.572	NaN	NaN	NaN	NaN	
3	1	4	2010-05-02	39954.04	False	42.31	2.572	NaN	NaN	NaN	NaN	
4	1	5	2010-05-02	32229.38	False	42.31	2.572	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	
421565	45	93	2012-10-26	2487.80	False	58.85	3.882	4018.91	58.08	100.0	211.94	8
421566	45	94	2012-10-26	5203.31	False	58.85	3.882	4018.91	58.08	100.0	211.94	8
421567	45	95	2012-10-26	56017.47	False	58.85	3.882	4018.91	58.08	100.0	211.94	8
421568	45	97	2012-10-26	6817.48	False	58.85	3.882	4018.91	58.08	100.0	211.94	8
421569	45	98	2012-10-26	1076.80	False	58.85	3.882	4018.91	58.08	100.0	211.94	8

421570 rows × 14 columns

In [13]:
```
df.head()
```

Out[13]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5
0	1	1	2010-05-02	24924.50	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN
1	1	2	2010-05-02	50605.27	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN
2	1	3	2010-05-02	13740.12	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN
3	1	4	2010-05-02	39954.04	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN
4	1	5	2010-05-02	32229.38	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN

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

Out[14]:

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

In [15]:
```
df = pd.merge(df, stores, on = ['Store'], how = 'left')
```

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

Out[16]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5
0	1	1	2010-05-02	24924.50	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN
1	1	2	2010-05-02	50605.27	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN
2	1	3	2010-05-02	13740.12	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN
3	1	4	2010-05-02	39954.04	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN
4	1	5	2010-05-02	32229.38	False	42.31	2.572	NaN	NaN	NaN	NaN	NaN

In [17]:
```
x = '2010-05-02'
str(x).split('-')
```

Out[17]:
```
['2010', '05', '02']
```

MINI CHALLENGE

- Define a function to extract the month information from the dataframe column "Date"
- Apply the function to the entire column "Date" in the merged dataframe "df" and write the output in a column entitled "month"

In [ ]:

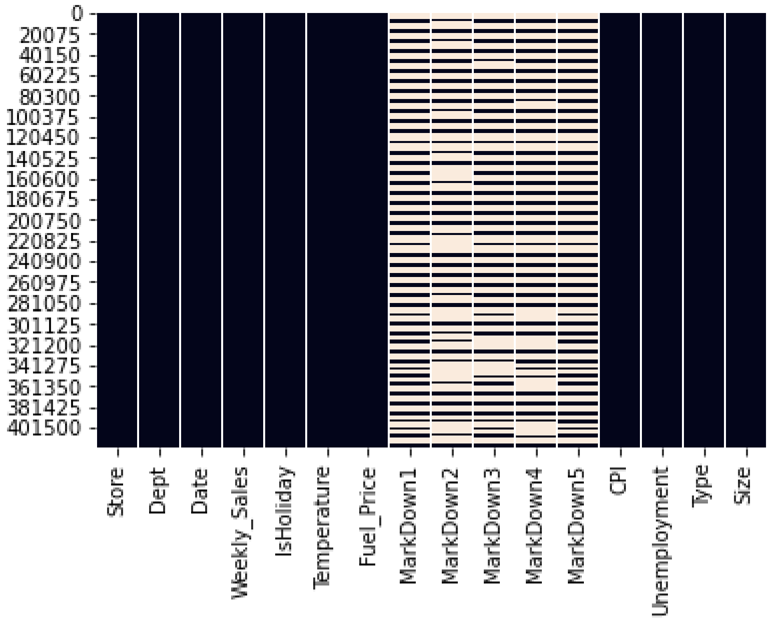
In [ ]:

In [ ]:

TASK #5: EXPLORE MERGED DATASET

In [18]:
```
sns.heatmap(df.isnull(), cbar = False)
```

Out[18]: <AxesSubplot:>



In [19]:
```
# check the number of non-null values in the dataframe
df.isnull().sum()
```

Out[19]:
```
Store                0
Dept                 0
Date                 0
Weekly_Sales         0
IsHoliday            0
Temperature          0
Fuel_Price           0
MarkDown1       270889
MarkDown2       310322
MarkDown3       284479
MarkDown4       286603
MarkDown5       270138
CPI                  0
Unemployment         0
Type                 0
Size                 0
dtype: int64
```

In [20]:
```
# Fill up NaN elements with zeros
df = df.fillna(0)
```

In [21]:
```
df
```

Out[21]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDo
0	1	1	2010-05-02	24924.50	False	42.31	2.572	0.00	0.00	0.0	0.00	
1	1	2	2010-05-02	50605.27	False	42.31	2.572	0.00	0.00	0.0	0.00	
2	1	3	2010-05-02	13740.12	False	42.31	2.572	0.00	0.00	0.0	0.00	
3	1	4	2010-05-02	39954.04	False	42.31	2.572	0.00	0.00	0.0	0.00	
4	1	5	2010-05-02	32229.38	False	42.31	2.572	0.00	0.00	0.0	0.00	
...	...	...	...	...	...	...	...	...	...	...	...	
421565	45	93	2012-10-26	2487.80	False	58.85	3.882	4018.91	58.08	100.0	211.94	8
421566	45	94	2012-10-26	5203.31	False	58.85	3.882	4018.91	58.08	100.0	211.94	8
421567	45	95	2012-10-26	56017.47	False	58.85	3.882	4018.91	58.08	100.0	211.94	8
421568	45	97	2012-10-26	6817.48	False	58.85	3.882	4018.91	58.08	100.0	211.94	8
421569	45	98	2012-10-26	1076.80	False	58.85	3.882	4018.91	58.08	100.0	211.94	8

421570 rows × 16 columns

In [22]:
```
# Statistical summary of the combined dataframe
df.describe()
```

Out[22]:

	Store	Dept	Weekly_Sales	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	Marl
count	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570
mean	22.200546	44.260317	15981.258123	60.090059	3.361027	2590.074819	879.974298	468.087665	1083
std	12.785297	30.492054	22711.183519	18.447931	0.458515	6052.385934	5084.538801	5528.873453	3894
min	1.000000	1.000000	-4988.940000	-2.060000	2.472000	0.000000	-265.760000	-29.100000	0
25%	11.000000	18.000000	2079.650000	46.680000	2.933000	0.000000	0.000000	0.000000	0
50%	22.000000	37.000000	7612.030000	62.090000	3.452000	0.000000	0.000000	0.000000	0
75%	33.000000	74.000000	20205.852500	74.280000	3.738000	2809.050000	2.200000	4.540000	425
max	45.000000	99.000000	693099.360000	100.140000	4.468000	88646.760000	104519.540000	141630.610000	67474

In [23]:
```
# check the number of duplicated entries in the dataframe
df.duplicated().sum()
```

Out[23]: 0

In [24]:
```
df['Type'].value_counts()
```

Out[24]:
```
A    215478
B    163495
C     42597
Name: Type, dtype: int64
```

MINI CHALLENGE

- Replace the "IsHoliday" with ones and zeros instead of True and False (characters with numbers)

In [ ]:

In [ ]:

TASK #6: PERFORM EXPLORATORY DATA ANALYSIS

In [25]:
```
# Create pivot tables to understand the relationship in the data

result = pd.pivot_table(df, values = 'Weekly_Sales', columns = ['Type'], index = ['Date', 'Store', 'Dept'],
                    aggfunc= np.mean)
```

In [26]:
```
result
```

Out[26]:

		Type	A	B	C
Date	**Store**	**Dept**			
		1	20094.19	NaN	NaN
		2	45829.02	NaN	NaN
2010-01-10	**1**	**3**	9775.17	NaN	NaN
		4	34912.45	NaN	NaN
		5	23381.38	NaN	NaN
...	**...**	**...**	...	...	...
		93	NaN	2644.24	NaN
		94	NaN	4041.28	NaN
2012-12-10	**45**	**95**	NaN	49334.77	NaN
		97	NaN	6463.32	NaN
		98	NaN	1061.02	NaN

421570 rows × 3 columns

In [27]:
```
result.describe()
# It can be seen that Type A stores have much higher sales than Type B and Type C
```

Out[27]:

Type	A	B	C
count	215478.000000	163495.000000	42597.000000
mean	20099.568043	12237.075977	9519.532538
std	26423.457227	17203.668989	15985.351612
min	-4988.940000	-3924.000000	-379.000000
25%	3315.090000	1927.055000	131.990000
50%	10105.170000	6187.870000	1149.670000
75%	26357.180000	15353.740000	12695.010000
max	474330.100000	693099.360000	112152.350000

In [28]:
```

result_md = pd.pivot_table(df, values = ['MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDown5'], columns =
                    aggfunc={'MarkDown1' : np.mean,'MarkDown2' : np.mean, 'MarkDown3' : np.mean, 'MarkDown4' : r
```

In [29]:
```
result_md
```

Out[29]:

			MarkDown1		MarkDown2		MarkDown3		MarkDown4		MarkDown5	
		IsHoliday	**False**	**True**	**False**	**True**	**False**	**True**	**False**	**True**	**False**	**True**
Date	**Store**	**Dept**										
2010-01-10	**1**	**1**	0.00	NaN	0.0	NaN	0.00	NaN	0.00	NaN	0.00	NaN
		2	0.00	NaN	0.0	NaN	0.00	NaN	0.00	NaN	0.00	NaN
		3	0.00	NaN	0.0	NaN	0.00	NaN	0.00	NaN	0.00	NaN
		4	0.00	NaN	0.0	NaN	0.00	NaN	0.00	NaN	0.00	NaN
		5	0.00	NaN	0.0	NaN	0.00	NaN	0.00	NaN	0.00	NaN
...	**...**	**...**	...	...	...	...	...	...	...	...	...	...
2012-12-10	**45**	**93**	1956.28	NaN	0.0	NaN	7.89	NaN	599.32	NaN	3990.54	NaN
		94	1956.28	NaN	0.0	NaN	7.89	NaN	599.32	NaN	3990.54	NaN
		95	1956.28	NaN	0.0	NaN	7.89	NaN	599.32	NaN	3990.54	NaN
		97	1956.28	NaN	0.0	NaN	7.89	NaN	599.32	NaN	3990.54	NaN
		98	1956.28	NaN	0.0	NaN	7.89	NaN	599.32	NaN	3990.54	NaN

421570 rows × 10 columns

In [30]:
```
result_md.sum()
```

Out[30]:
```
           IsHoliday
MarkDown1  False        1.017371e+09
           True         7.452684e+07
MarkDown2  False        2.310619e+08
           True         1.399088e+08
MarkDown3  False        2.460332e+07
           True         1.727284e+08
MarkDown4  False        4.196331e+08
           True         3.698298e+07
MarkDown5  False        6.585670e+08
           True         4.240793e+07
dtype: float64
```

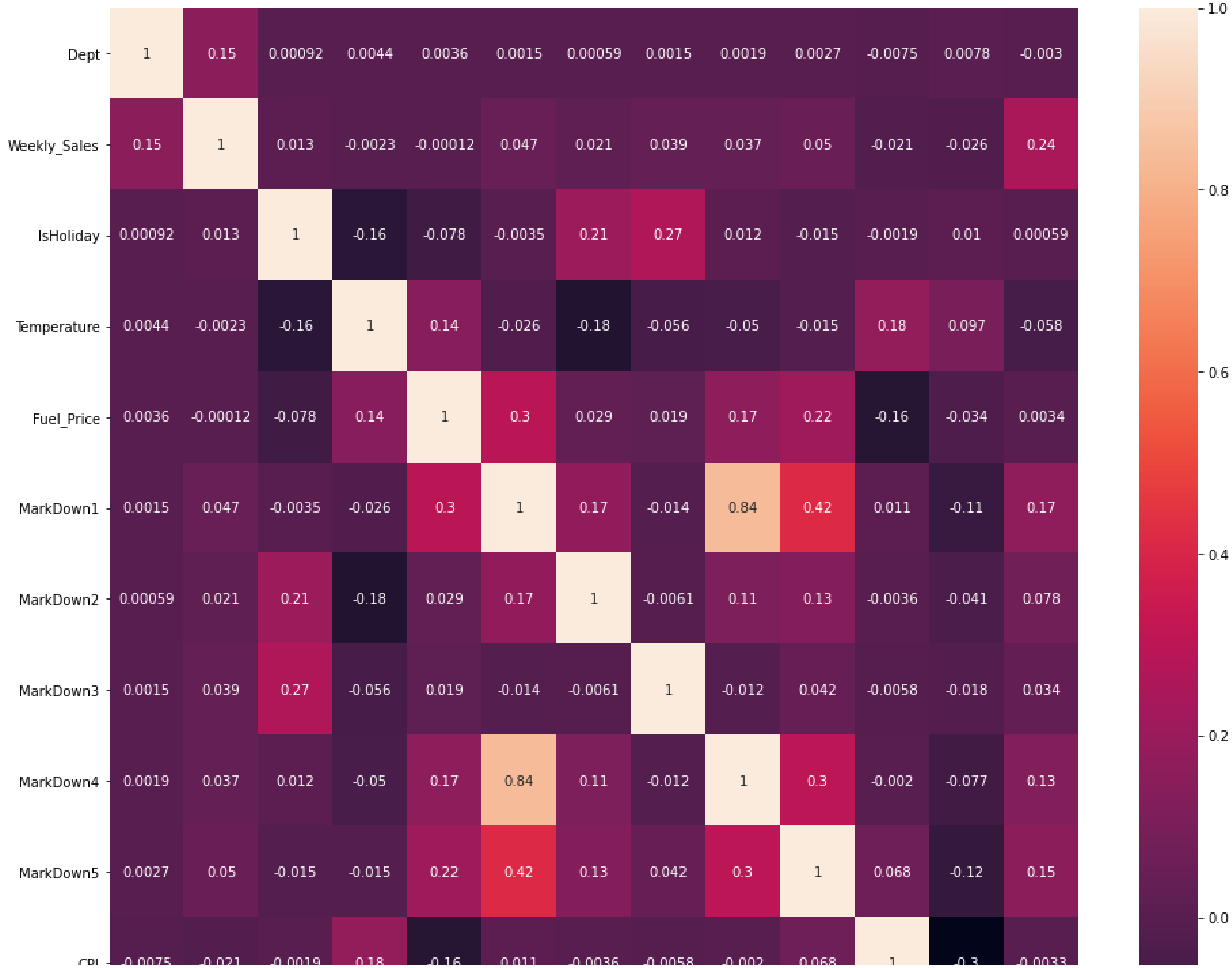
In [31]:
```
result_md.describe()
# we can conclude that MarkDown2 and MarkDown3 have higher volume on holidays compared to that of regular days
# while other MarkDowns don't show significant changes relating to holiday.
```

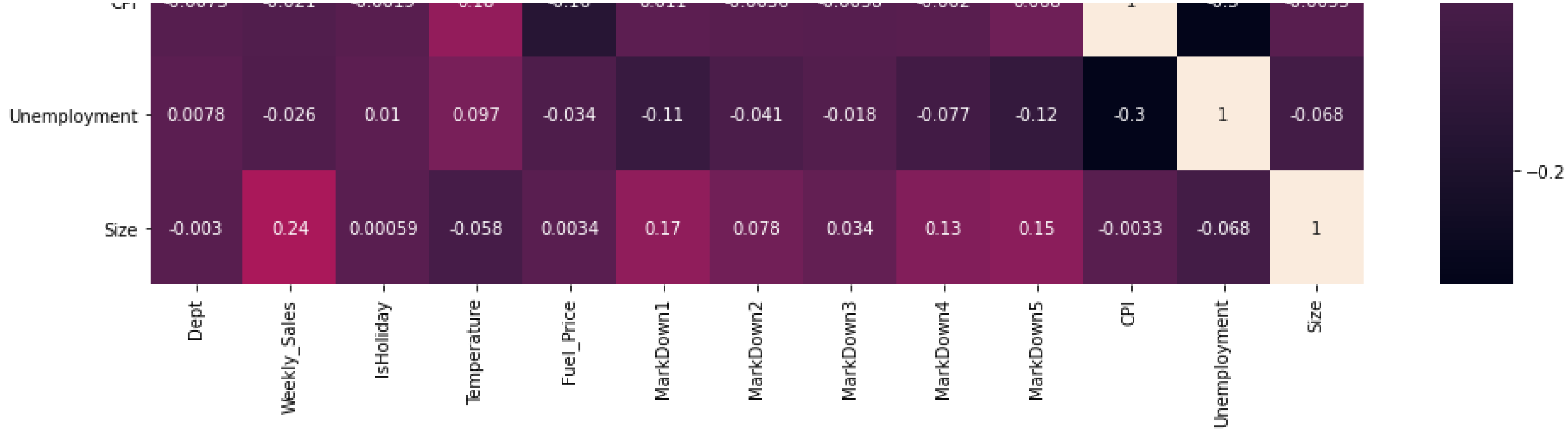
Out[31]:

	MarkDown1		MarkDown2		MarkDown3		MarkDown4		
IsHoliday	**False**	**True**	**False**	**True**	**False**	**True**	**False**	**True**	
count	391909.000000	29661.000000	391909.000000	29661.000000	391909.000000	29661.000000	391909.000000	29661.000000	39190
mean	2595.936803	2512.620778	589.580546	4716.929394	62.778142	5823.417900	1070.741151	1246.855336	168
std	6123.402037	5020.047408	2984.163111	15295.329993	630.704594	19959.302249	3921.553070	3513.998030	431
min	0.000000	0.000000	-265.760000	-9.980000	-29.100000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	2826.570000	2463.160000	0.500000	65.000000	3.840000	66.080000	442.390000	319.190000	218
max	88646.760000	36778.650000	45971.430000	104519.540000	25959.980000	141630.610000	67474.850000	29483.810000	10851

In [32]:
```
corr_matrix = df.drop(columns = ['Store']).corr()
```

In [33]:
```
plt.figure(figsize = (16,16))
sns.heatmap(corr_matrix, annot = True)
plt.show()
```





TASK #7: PERFORM DATA VISUALIZATION

In [34]:
```
df
```

Out[34]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDo
0	1	1	2010-05-02	24924.50	False	42.31	2.572	0.00	0.00	0.0	0.00	
1	1	2	2010-05-02	50605.27	False	42.31	2.572	0.00	0.00	0.0	0.00	
2	1	3	2010-05-02	13740.12	False	42.31	2.572	0.00	0.00	0.0	0.00	
3	1	4	2010-05-02	39954.04	False	42.31	2.572	0.00	0.00	0.0	0.00	
4	1	5	2010-05-02	32229.38	False	42.31	2.572	0.00	0.00	0.0	0.00	
...	...	...	...	...	...	...	...	...	...	...	...	
421565	45	93	2012-10-26	2487.80	False	58.85	3.882	4018.91	58.08	100.0	211.94	85
421566	45	94	2012-10-26	5203.31	False	58.85	3.882	4018.91	58.08	100.0	211.94	85
421567	45	95	2012-10-26	56017.47	False	58.85	3.882	4018.91	58.08	100.0	211.94	85
421568	45	97	2012-10-26	6817.48	False	58.85	3.882	4018.91	58.08	100.0	211.94	85
421569	45	98	2012-10-26	1076.80	False	58.85	3.882	4018.91	58.08	100.0	211.94	85

421570 rows × 16 columns

In [35]:
```
df.hist(bins = 30, figsize = (20,20), color = 'r')
```

...

In [36]:
```
# visualizing the relationship using pairplots
# there is a relationship between markdown #1 and Markdown #4
# holiday and sales
# Weekly sales and markdown #3
sns.pairplot(df[["Weekly_Sales","IsHoliday","MarkDown1","MarkDown2","MarkDown3","MarkDown4","MarkDown5","Type","
```

...

In [37]:
```
df_type = df.groupby('Type').mean()
```

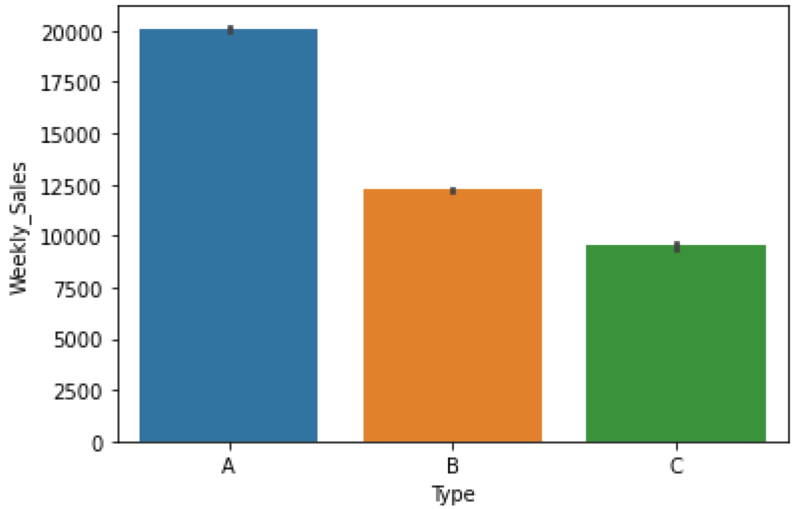
In [38]:
```
df_type
```

Out[38]:

Type	Store	Dept	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	Mark
A	21.736419	44.622156	20099.568043	0.070471	60.531945	3.343999	3102.403194	1083.216159	549.644930	1325.891281	2147.
B	18.450417	43.112273	12237.075977	0.070412	57.562951	3.382523	2553.465968	827.500452	481.215226	1043.927675	1324.
C	38.942015	46.836350	9519.532538	0.069582	67.554266	3.364654	138.960203	53.274338	5.142226	5.603993	505.

In [39]:
```
sns.barplot(x = df['Type'], y = df['Weekly_Sales'], data = df)
```

Out[39]: <AxesSubplot:xlabel='Type', ylabel='Weekly_Sales'>



In [40]:

```
# df_dept = df.drop(columns = ['Store','Type','IsHoliday','Temperature','Fuel_Price','CPI','Unemployment','Size'
df_dept = df.groupby('Dept').mean()
df_dept
```

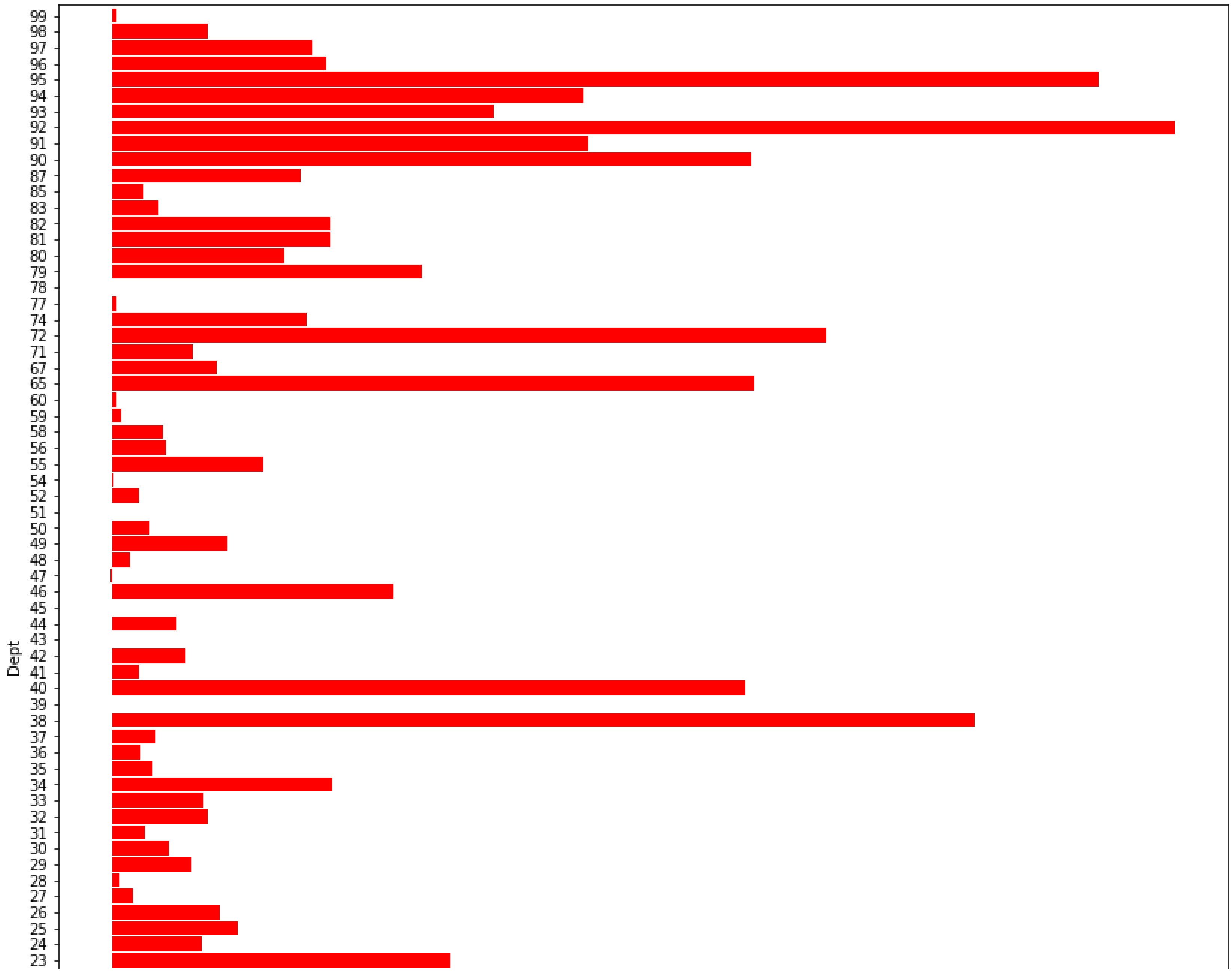
Out[40]:

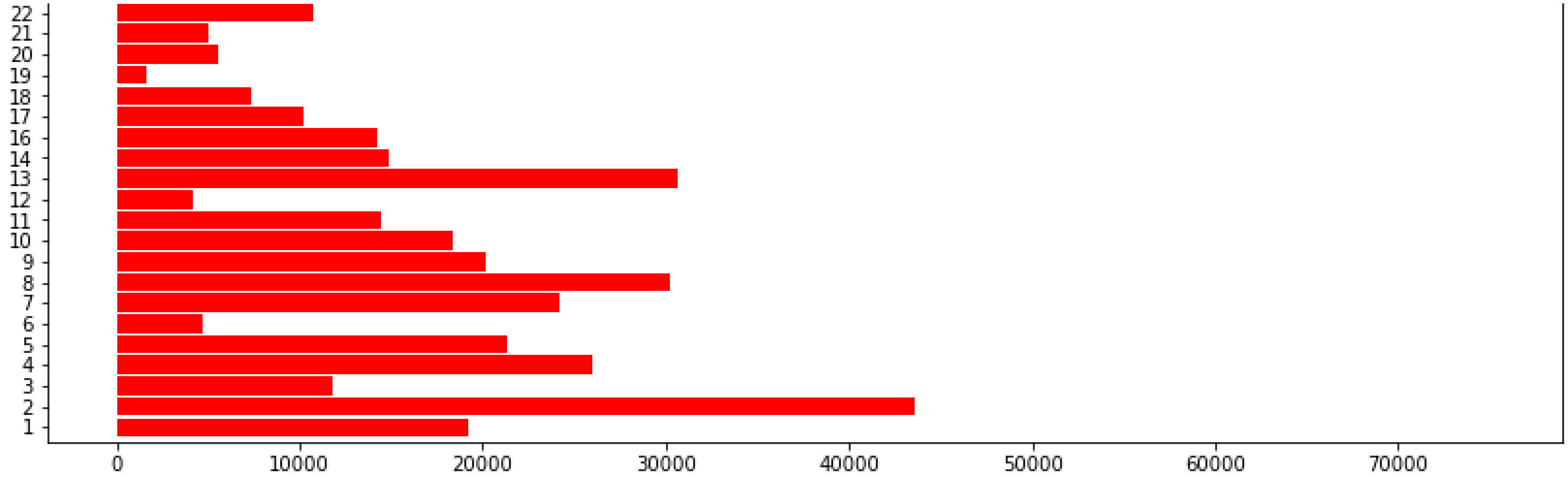
Dept	Store	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	
1	23.000000	19213.485088	0.069930	60.663782	3.358607	2429.019322	818.872810	429.184037	1008.870435	1581.806813	17
2	23.000000	43607.020113	0.069930	60.663782	3.358607	2429.019322	818.872810	429.184037	1008.870435	1581.806813	17
3	23.000000	11793.698516	0.069930	60.663782	3.358607	2429.019322	818.872810	429.184037	1008.870435	1581.806813	17
4	23.000000	25974.630238	0.069930	60.663782	3.358607	2429.019322	818.872810	429.184037	1008.870435	1581.806813	17
5	22.757366	21365.583515	0.069797	60.559367	3.365397	2462.697233	830.226332	435.134596	1022.858240	1603.738276	17
...	...	...	...	...	...	...	...	...	...	...	
95	23.000000	69824.423080	0.069930	60.663782	3.358607	2429.019322	818.872810	429.184037	1008.870435	1581.806813	17
96	23.258138	15210.942761	0.069839	61.539285	3.359920	2362.845647	820.762363	397.214137	999.452087	1660.599345	17
97	23.357439	14255.576919	0.069767	60.490781	3.362418	2463.638764	833.096524	432.439341	1025.957821	1591.276367	17
98	24.173920	6824.694889	0.071967	60.115942	3.372656	2569.994716	882.483088	467.655716	1074.883525	1678.390840	16
99	21.438515	415.487065	0.110209	62.813596	3.592702	7741.403376	2164.573063	1734.841903	3897.476369	4526.868643	17

81 rows × 13 columns

In [41]:
```
fig = plt.figure(figsize = (14,16))
df_dept['Weekly_Sales'].plot(kind = 'barh', color = 'r', width = 0.9)
```

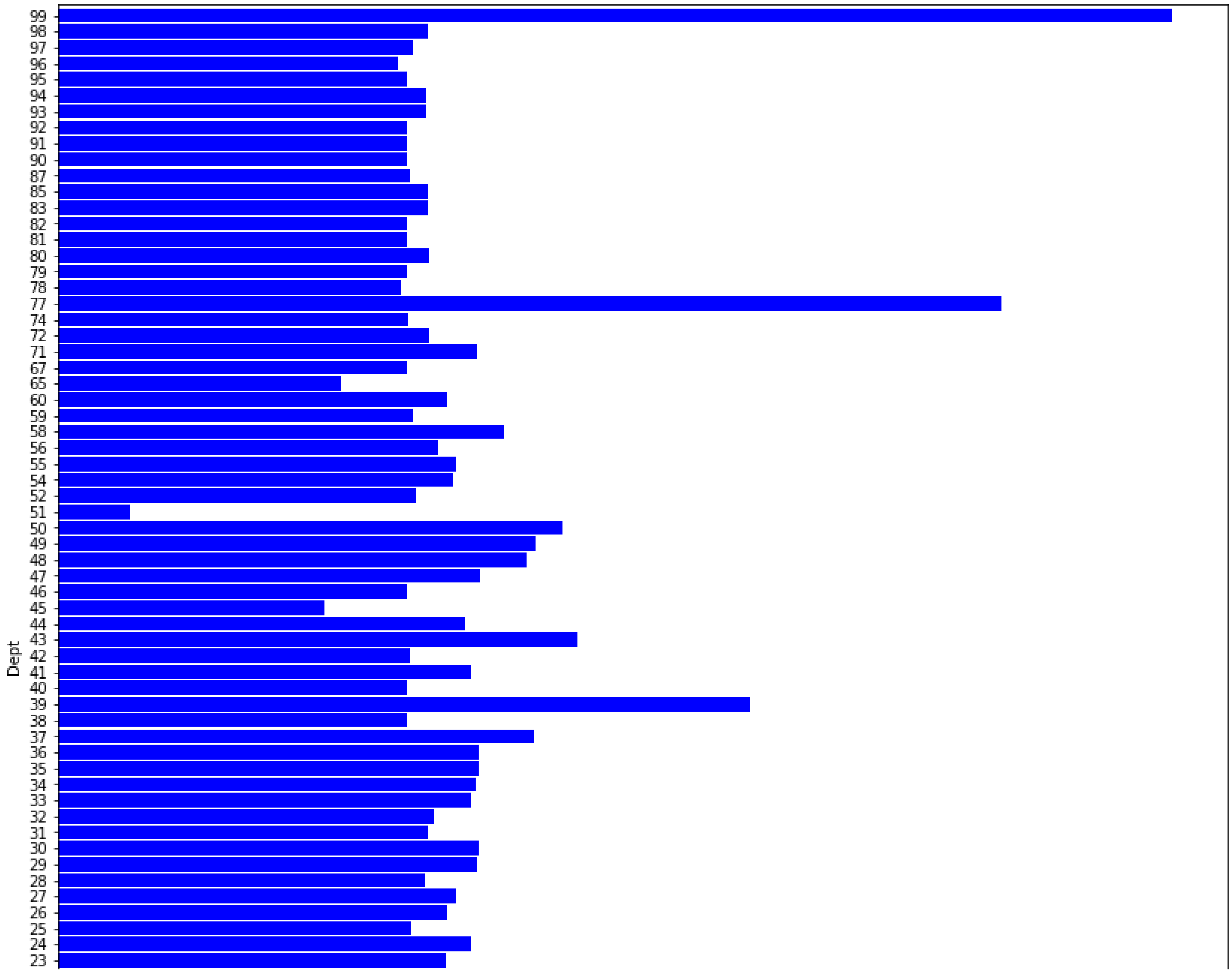
Out[41]: <AxesSubplot:ylabel='Dept'>

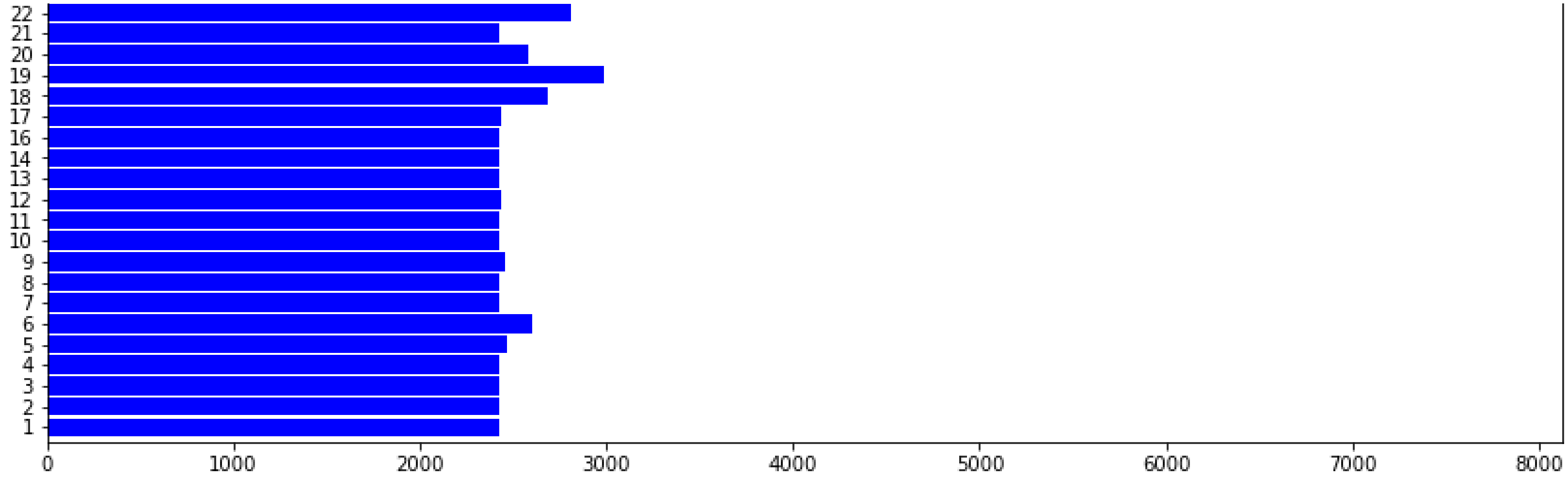



22
21
20
19
18
17
16
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
10000
20000
30000
40000
50000
60000
70000

In [42]:
```
fig = plt.figure(figsize = (14,16))
df_dept['MarkDown1'].plot(kind = 'barh', color = 'blue', width = 0.9)
```

Out[42]: <AxesSubplot:ylabel='Dept'>

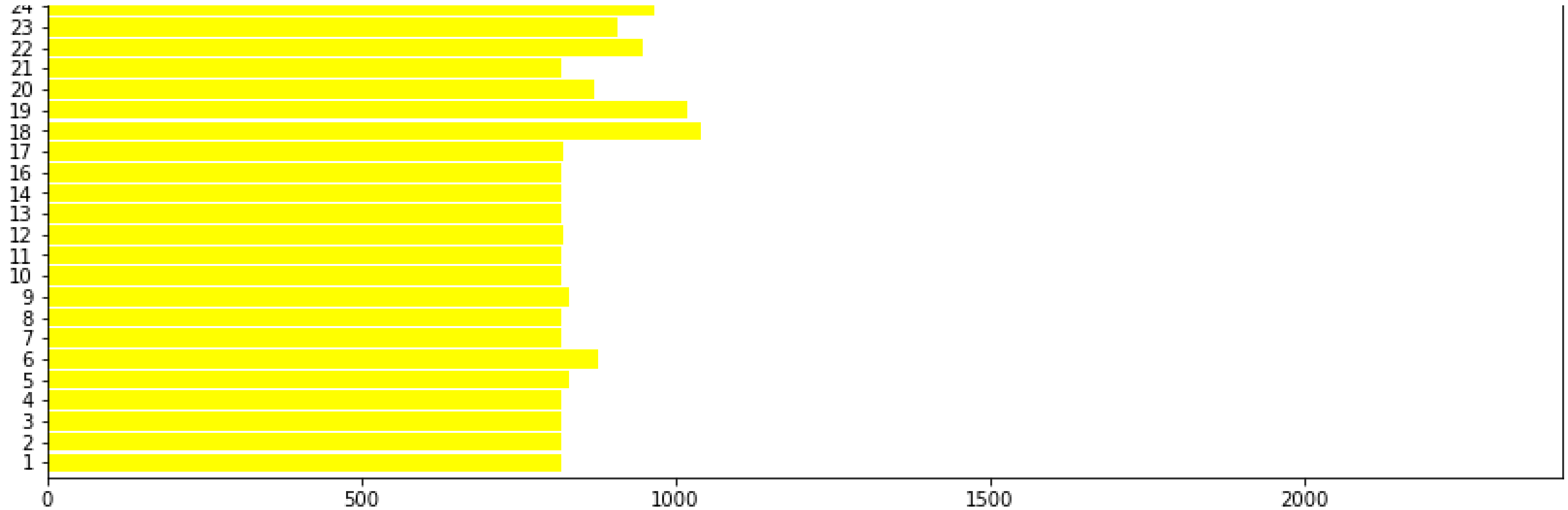



22
21
20
19
18
17
16
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
1000
2000
3000
4000
5000
6000
7000
8000

In [43]:
```
fig = plt.figure(figsize = (14,16))

df_dept['MarkDown2'].plot(kind = 'barh', color = 'yellow', width = 0.9)
```

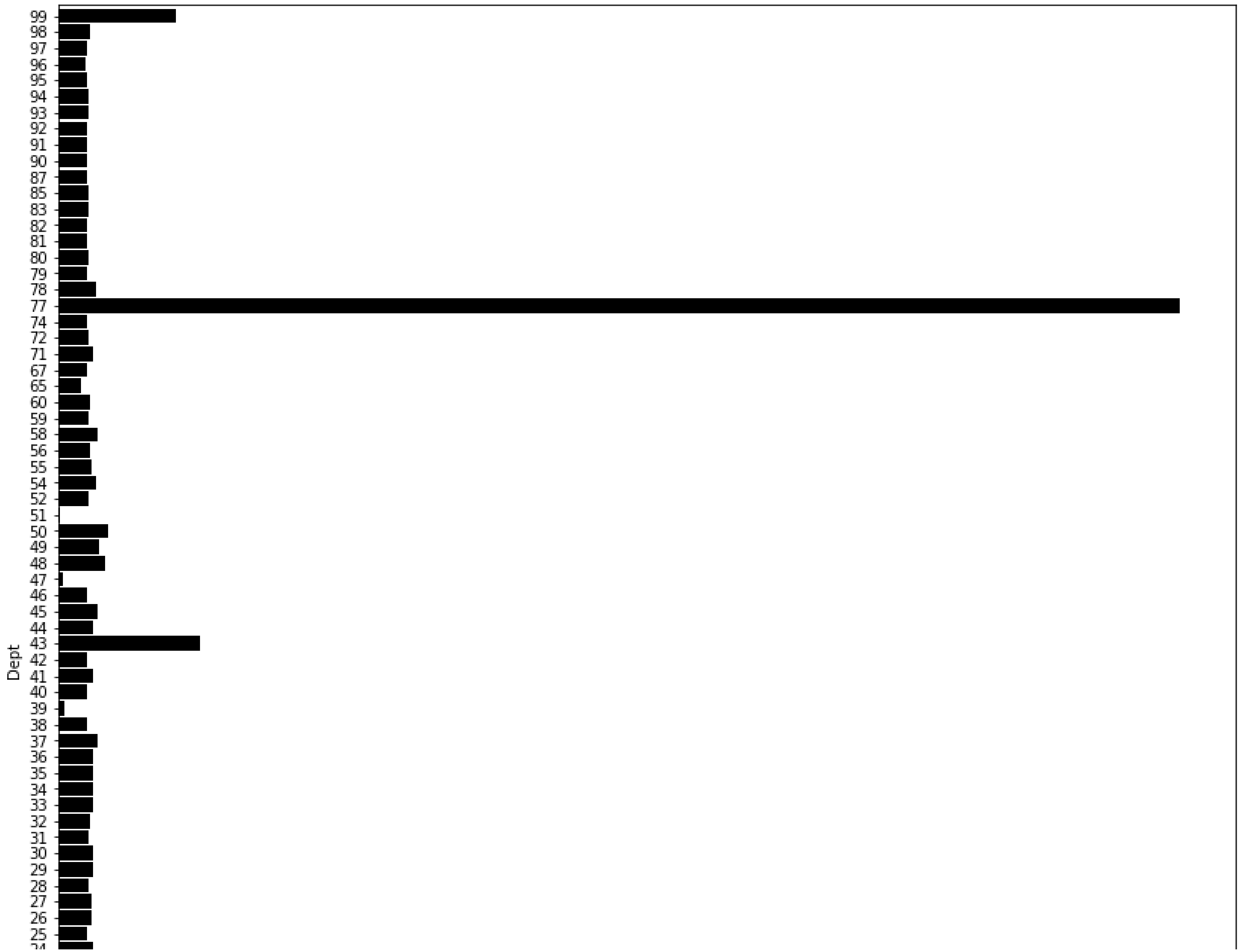
Out[43]: <AxesSubplot:ylabel='Dept'>

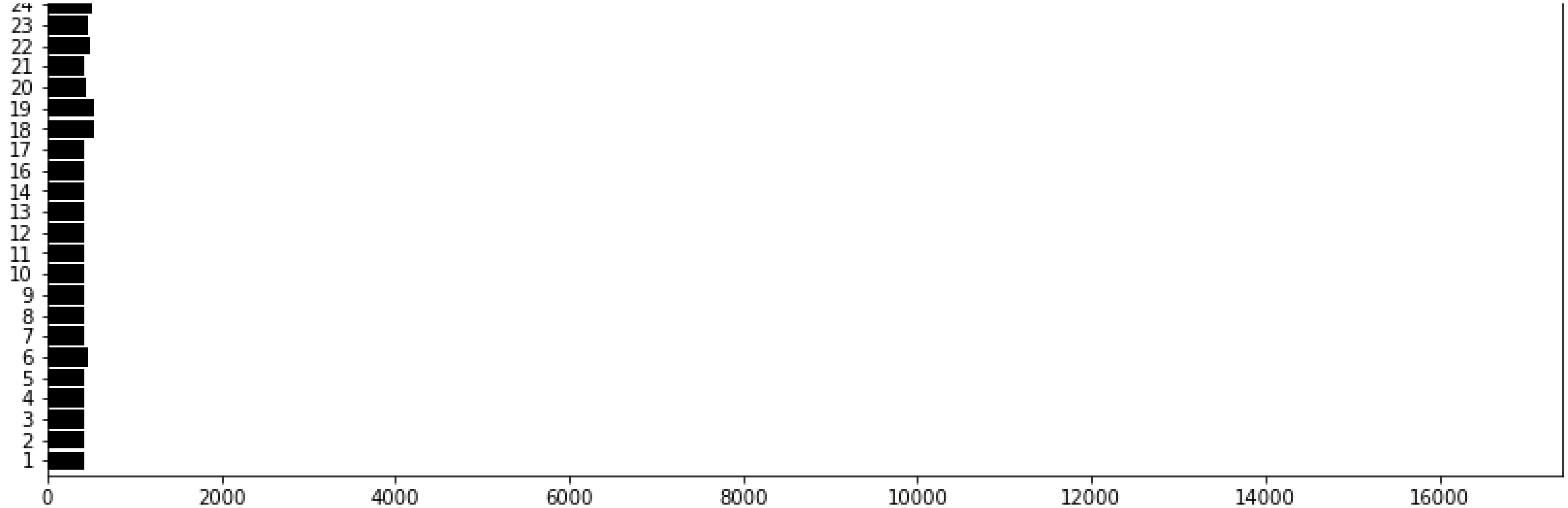

24
23
22
21
20
19
18
17
16
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
500
1000
1500
2000

In [44]:
```
fig = plt.figure(figsize = (14,16))

df_dept['MarkDown3'].plot(kind = 'barh', color = 'black', width = 0.9)
```

Out[44]: <AxesSubplot:ylabel='Dept'>

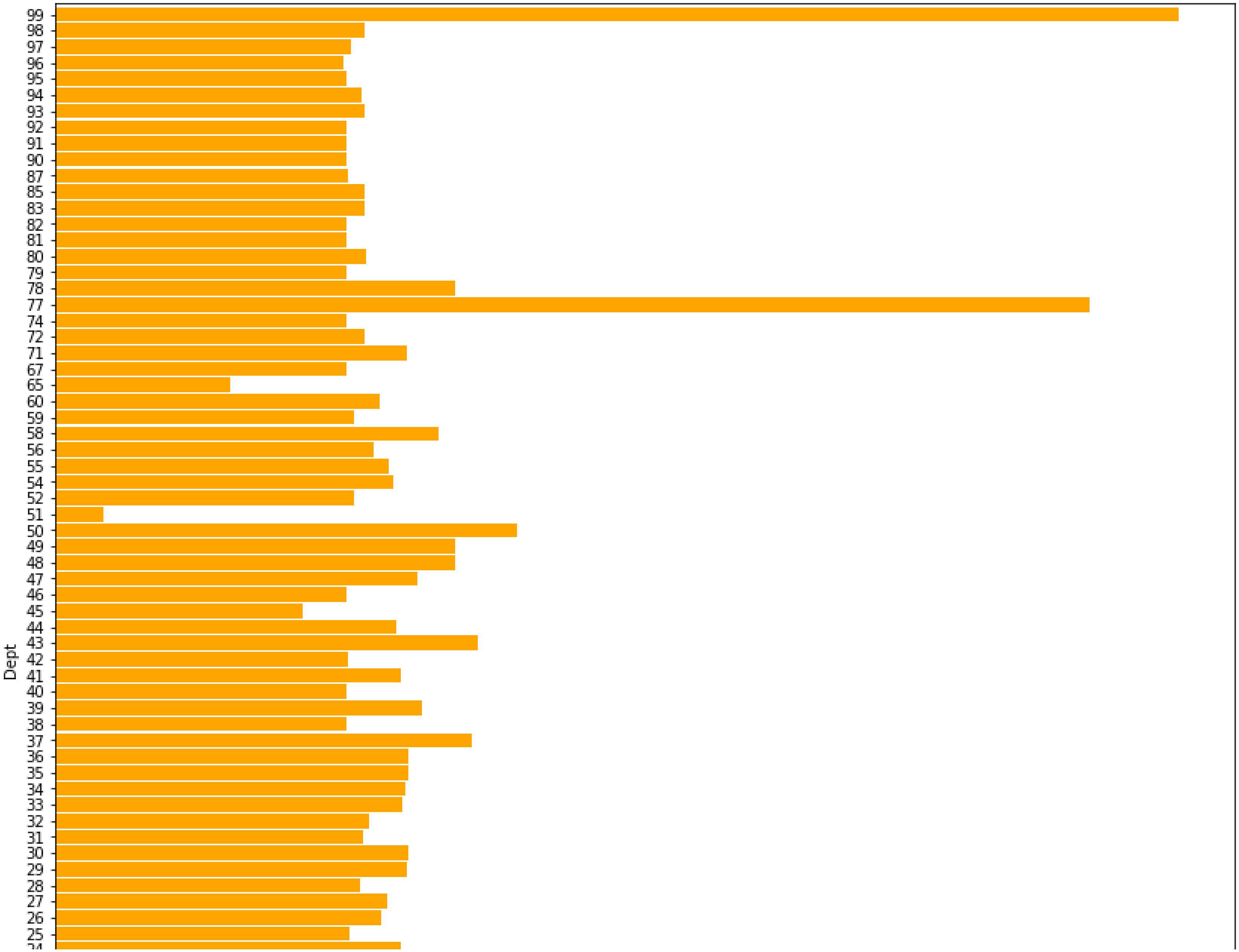


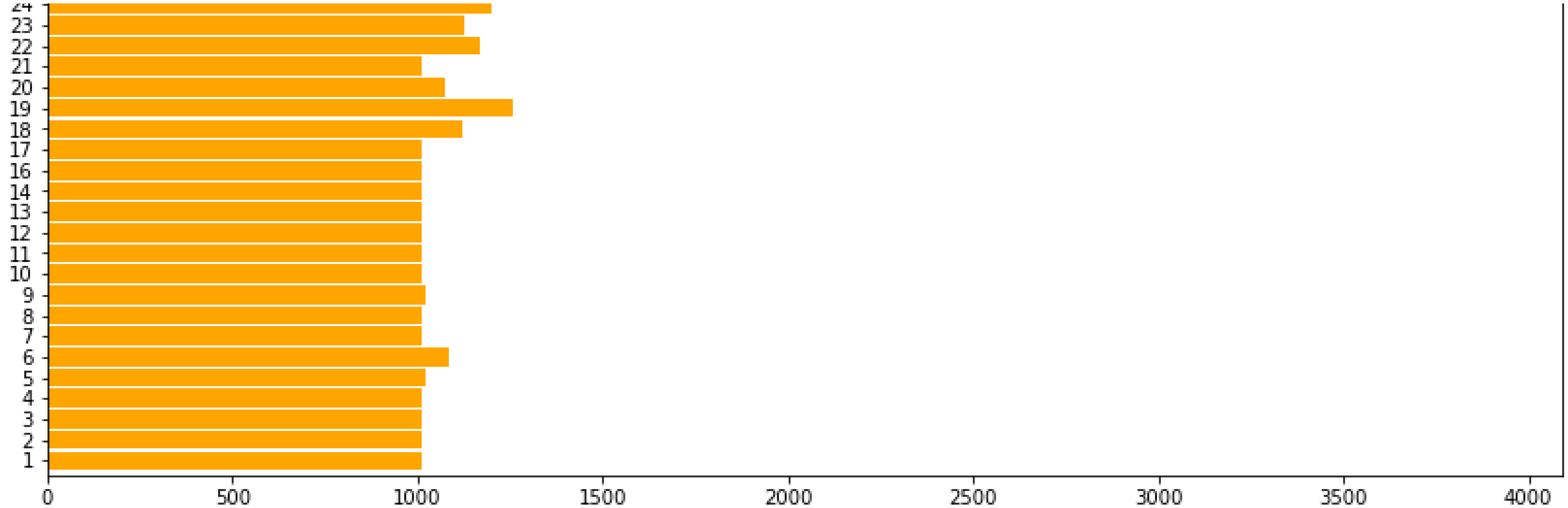

24
23
22
21
20
19
18
17
16
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
2000
4000
6000
8000
10000
12000
14000
16000

In [45]:
```
fig = plt.figure(figsize = (14,16))

df_dept['MarkDown4'].plot(kind = 'barh', color = 'orange', width = 0.9)
```

Out[45]: <AxesSubplot:ylabel='Dept'>

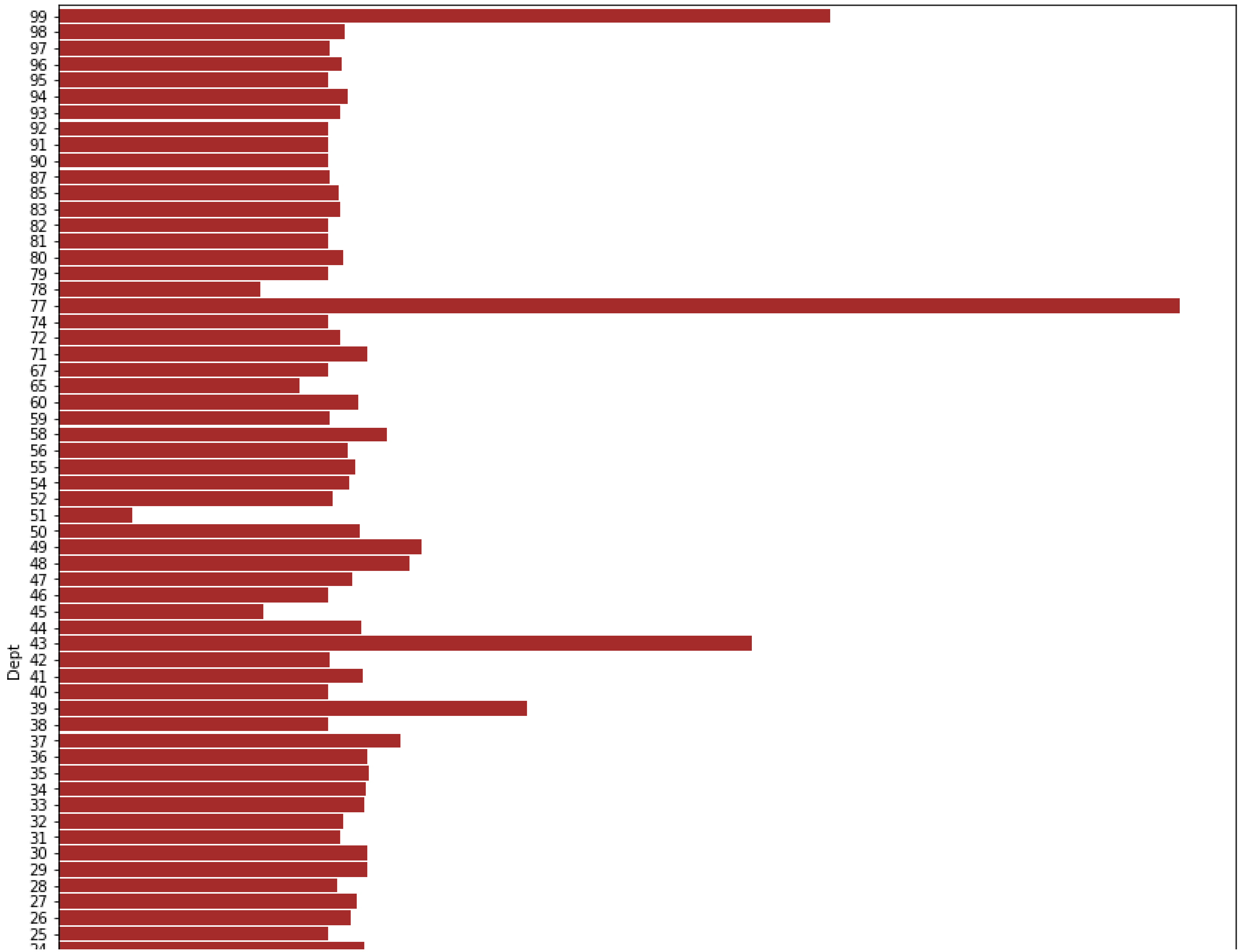


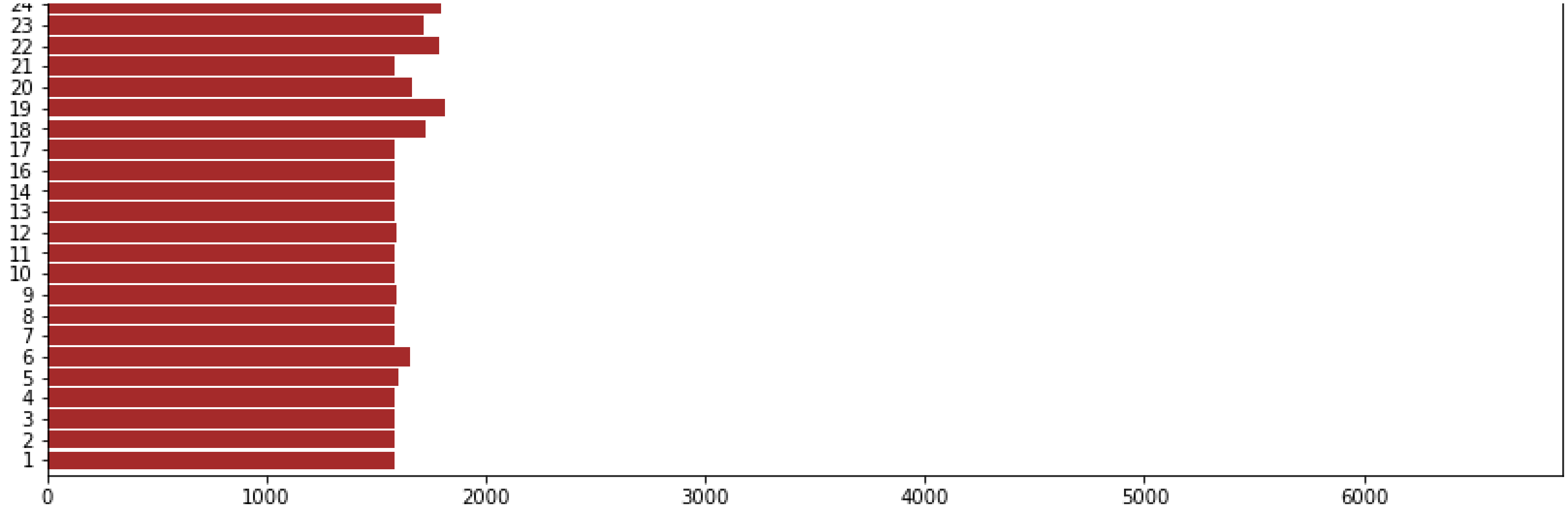

24
23
22
21
20
19
18
17
16
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
500
1000
1500
2000
2500
3000
3500
4000

In [46]:
```
fig = plt.figure(figsize = (14,16))

df_dept['MarkDown5'].plot(kind = 'barh', color = 'brown', width = 0.9)
```

Out[46]: <AxesSubplot:ylabel='Dept'>





- We can conclude that departments that have poor weekly sales have been assigned high number of markdowns. Let's explore this in more details
- Example: check out store 77 and 99

In [47]:
```
# Sort by weekly sales
df_dept_sale = df_dept.sort_values(by = ['Weekly_Sales'], ascending = True)
df_dept_sale['Weekly_Sales'][:30]
```

Out[47]:
```
Dept
47      -7.682554
43       1.193333
78       7.296638
39      11.123750
51      21.931729
45      23.211586
54     108.305985
77     328.961800
60     347.370229
99     415.487065
28     618.085116
59     694.463564
48    1344.893576
27    1583.437727
19    1654.815030
52    1928.356252
41    1965.559998
36    2022.571061
85    2264.359407
31    2339.440287
50    2658.897010
35    2921.044946
37    3111.076193
83    3383.349838
58    3702.907419
56    3833.706211
30    4118.197208
12    4175.397021
44    4651.729658
6     4747.856188
Name: Weekly_Sales, dtype: float64
```

TASK #8: PREPARE THE DATA BEFORE TRAINING

In [48]:
```
# Drop the date
df_target = df['Weekly_Sales']
df_final = df.drop(columns = ['Weekly_Sales', 'Date'])
```

In [49]:
```
df_final = pd.get_dummies(df_final, columns = ['Type', 'Store', 'Dept'], drop_first = True)
```

In [50]:
```
df_final.shape
```

Out[50]:
```
(421570, 137)
```

In [51]:
```
df_target.shape
```

Out[51]:
```
(421570,)
```

In [52]:
```
df_final
```

Out[52]:

	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	.
0	False	42.31	2.572	0.00	0.00	0.0	0.00	0.00	211.096358	8.106	.
1	False	42.31	2.572	0.00	0.00	0.0	0.00	0.00	211.096358	8.106	.
2	False	42.31	2.572	0.00	0.00	0.0	0.00	0.00	211.096358	8.106	.
3	False	42.31	2.572	0.00	0.00	0.0	0.00	0.00	211.096358	8.106	.
4	False	42.31	2.572	0.00	0.00	0.0	0.00	0.00	211.096358	8.106	.
...	...	...	...	...	...	...	...	...	...	...	.
421565	False	58.85	3.882	4018.91	58.08	100.0	211.94	858.33	192.308899	8.667	.
421566	False	58.85	3.882	4018.91	58.08	100.0	211.94	858.33	192.308899	8.667	.
421567	False	58.85	3.882	4018.91	58.08	100.0	211.94	858.33	192.308899	8.667	.
421568	False	58.85	3.882	4018.91	58.08	100.0	211.94	858.33	192.308899	8.667	.
421569	False	58.85	3.882	4018.91	58.08	100.0	211.94	858.33	192.308899	8.667	.

421570 rows × 137 columns

In [53]:
```
X = np.array(df_final).astype('float32')
y = np.array(df_target).astype('float32')
```

In [54]:
```
# reshaping the array from (421570,) to (421570, 1)
y = y.reshape(-1,1)
y.shape
```

Out[54]:
```
(421570, 1)
```

In [55]:
```
# scaling the data before feeding the model
# from sklearn.preprocessing import StandardScaler, MinMaxScaler

# scaler_x = StandardScaler()
# X = scaler_x.fit_transform(X)

# scaler_y = StandardScaler()
# y = scaler_y.fit_transform(y)
```

In [56]:
```
# spliting the data in to test and train sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15)
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size = 0.5)
```

In [57]:
```
X_train
```

Out[57]:
```
array([[ 0.   , 91.05 ,  3.575, ...,  0.   ,  0.   ,  0.   ],
       [ 0.   , 76.91 ,  2.784, ...,  0.   ,  0.   ,  0.   ],
       [ 0.   , 39.   ,  3.751, ...,  0.   ,  0.   ,  0.   ],
       ...,
       [ 0.   , 85.8  ,  3.554, ...,  0.   ,  0.   ,  0.   ],
       [ 0.   , 74.36 ,  3.827, ...,  0.   ,  0.   ,  0.   ],
       [ 0.   , 81.47 ,  3.523, ...,  0.   ,  0.   ,  0.   ]],
      dtype=float32)
```

TASK #9: TRAIN XGBOOST REGRESSOR IN LOCAL MODE

In [58]:
```
!pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-1.3.3-py3-none-manylinux2010_x86_64.whl (157.5 MB)
     |████████████████████████████████| 157.5 MB 24 kB/s s eta 0:00:01
Requirement already satisfied: numpy in /home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (fro
m xgboost) (1.19.5)
Requirement already satisfied: scipy in /home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (fro
m xgboost) (1.5.3)
Installing collected packages: xgboost
Successfully installed xgboost-1.3.3
```

In [59]:
```
# Train an XGBoost regressor model

import xgboost as xgb


model = xgb.XGBRegressor(objective ='reg:squarederror', learning_rate = 0.1, max_depth = 5, n_estimators = 100)

model.fit(X_train, y_train)
```

Out[59]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.1, max_delta_step=0, max_depth=5,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=2, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [60]:
```
# predict the score of the trained model using the testing dataset

result = model.score(X_test, y_test)

print("Accuracy : {}".format(result))
```

```
Accuracy : 0.8192406043997631
```

In [61]:
```
# make predictions on the test data

y_predict = model.predict(X_test)
```

In [62]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt
k = X_test.shape[1]
n = len(X_test)
RMSE = float(format(np.sqrt(mean_squared_error(y_test, y_predict)),'.3f'))
MSE = mean_squared_error(y_test, y_predict)
MAE = mean_absolute_error(y_test, y_predict)
r2 = r2_score(y_test, y_predict)
adj_r2 = 1-(1-r2)*(n-1)/(n-k-1)

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =', adj_r2)
```

```
RMSE = 9779.869
MSE = 95645850.0
MAE = 6435.3916
R2 = 0.8192406043997631
Adjusted R2 = 0.8184539450224686
```

MINI CHALLENGE

- Retrain the model with less 'max_depth'
- Comment on the results

In [ ]:
```

```

TASK #10: TRAIN XGBOOST USING SAGEMAKER

In [63]:
```
# Convert the array into dataframe in a way that target variable is set as the first column and followed by feat
# This is because sagemaker built-in algorithm expects the data in this format.

train_data = pd.DataFrame({'Target': y_train[:,0]})
for i in range(X_train.shape[1]):
    train_data[i] = X_train[:,i]
```

In [64]:
```
train_data.head()
```

Out[64]:

	Target	0	1	2	3	4	5	6	7	8	...	127	128	129	130	1
0	83.400002	0.0	91.050003	3.575	0.000000	0.000000	0.0	0.000000	0.000000	215.013443	...	0.0	0.0	0.0	0.0	(
1	19221.000000	0.0	76.910004	2.784	0.000000	0.000000	0.0	0.000000	0.000000	136.436691	...	1.0	0.0	0.0	0.0	(
2	22466.269531	0.0	39.000000	3.751	10045.030273	7913.379883	0.0	8695.830078	3361.360107	141.300781	...	0.0	0.0	0.0	0.0	(
3	11735.540039	0.0	80.889999	3.786	0.000000	0.000000	0.0	0.000000	0.000000	207.311981	...	0.0	0.0	0.0	0.0	(
4	1358.140015	0.0	85.730003	2.664	0.000000	0.000000	0.0	0.000000	0.000000	210.361755	...	0.0	0.0	0.0	0.0	(

5 rows × 138 columns

In [65]:
```
val_data = pd.DataFrame({'Target':y_val[:,0]})
for i in range(X_val.shape[1]):
    val_data[i] = X_val[:,i]
```

In [66]:
```
val_data.head()
```

Out[66]:

	Target	0	1	2	3	4	5	6	7	8	...	127	128	129
0	70.000000	0.0	74.690002	2.860	0.000000	0.000000	0.000000	0.000000	0.000000	132.724838	...	0.0	0.0	0.0
1	83.339996	0.0	67.790001	3.524	0.000000	0.000000	0.000000	0.000000	0.000000	206.673309	...	0.0	0.0	0.0
2	5162.040039	1.0	28.139999	2.771	0.000000	0.000000	0.000000	0.000000	0.000000	131.586609	...	0.0	0.0	0.0
3	898.780029	0.0	50.820000	3.583	0.000000	0.000000	0.000000	0.000000	0.000000	210.117065	...	0.0	0.0	0.0
4	73200.062500	0.0	54.439999	3.157	5107.290039	32305.300781	144.660004	530.549988	6004.189941	223.192307	...	0.0	0.0	0.0

5 rows × 138 columns

In [70]:
```
val_data.shape
```

Out[70]:
```
(31618, 138)
```

In [71]:
```
# save train_data and validation_data as csv files.

train_data.to_csv('train.csv', header = False, index = False)
val_data.to_csv('validation.csv', header = False, index = False)
```

In [72]:
```
# Boto3 is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python
# Boto3 allows Python developer to write software that makes use of services like Amazon S3 and Amazon EC2

import sagemaker
import boto3
from sagemaker import Session

# Let's create a Sagemaker session
sagemaker_session = sagemaker.Session()
bucket = Session().default_bucket()
prefix = 'XGBoost-Regressor'
key = 'XGBoost-Regressor'
#Roles give learning and hosting access to the data
#This is specified while opening the sagemakers instance in "Create an IAM role"
role = sagemaker.get_execution_role()
```

In [73]:
```
print(role)
```

```
arn:aws:iam::542063182511:role/service-role/AmazonSageMaker-ExecutionRole-20191104T033920
```

In [74]:
```
# read the data from csv file and then upload the data to s3 bucket
import os
with open('train.csv','rb') as f:
    # The following code uploads the data into S3 bucket to be accessed later for training
    boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train', key)).upload_fileobj(f)

# Let's print out the training data location in s3
s3_train_data = 's3://{}/{}/train/{}'.format(bucket, prefix, key)
print('uploaded training data location: {}'.format(s3_train_data))
```

```
uploaded training data location: s3://sagemaker-us-east-2-542063182511/XGBoost-Regressor/train/XGBoost-Regress
or
```

In [75]:
```
# read the data from csv file and then upload the data to s3 bucket

with open('validation.csv','rb') as f:
    # The following code uploads the data into S3 bucket to be accessed later for training

    boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'validation', key)).upload_fileobj
# Let's print out the validation data location in s3
s3_validation_data = 's3://{}/{}/validation/{}'.format(bucket, prefix, key)
print('uploaded validation data location: {}'.format(s3_validation_data))
```

```
uploaded validation data location: s3://sagemaker-us-east-2-542063182511/XGBoost-Regressor/validation/XGBoost-
Regressor
```

In [76]:
```
# creates output placeholder in S3 bucket to store the output

output_location = 's3://{}/{}/output'.format(bucket, prefix)
print('training artifacts will be uploaded to: {}'.format(output_location))
```

```
training artifacts will be uploaded to: s3://sagemaker-us-east-2-542063182511/XGBoost-Regressor/output
```

In [77]:
```
# This code is used to get the training container of sagemaker built-in algorithms
# all we have to do is to specify the name of the algorithm, that we want to use

# Let's obtain a reference to the XGBoost container image
# Note that all regression models are named estimators
# You don't have to specify (hardcode) the region, get_image_uri will get the current region name using boto3.Se

from sagemaker.amazon.amazon_estimator import get_image_uri

container = get_image_uri(boto3.Session().region_name, 'xgboost','0.90-2') # Latest version of XGboost
```

```
The method get_image_uri has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html (https://sagemaker.readthedocs.io/en/stable/v2.html) f
or details.
```

In [78]:
```
# Specify the type of instance that we would like to use for training
# output path and sagemaker session into the Estimator.
# We can also specify how many instances we would like to use for training

# Recall that XGBoost works by combining an ensemble of weak models to generate accurate/robust results.
# The weak models are randomized to avoid overfitting

# num_round: The number of rounds to run the training.


# Alpha: L1 regularization term on weights. Increasing this value makes models more conservative.

# colsample_by_tree: fraction of features that will be used to train each tree.

# eta: Step size shrinkage used in updates to prevent overfitting.
# After each boosting step, eta parameter shrinks the feature weights to make the boosting process more conserva


Xgboost_regressor1 = sagemaker.estimator.Estimator(container,
                                       role,
                                       train_instance_count = 1,
                                       train_instance_type = 'ml.m5.2xlarge',
                                       output_path = output_location,
                                       sagemaker_session = sagemaker_session)

#We can tune the hyper-parameters to improve the performance of the model

Xgboost_regressor1.set_hyperparameters(max_depth = 10,
                           objective = 'reg:linear',
                           colsample_bytree = 0.3,
                           alpha = 10,
                           eta = 0.1,
                           num_round = 100
                           )
```

```
train_instance_count has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html (https://sagemaker.readthedocs.io/en/stable/v2.html)
for details.
train_instance_type has been renamed in sagemaker>=2.
```

```
See: https://sagemaker.readthedocs.io/en/stable/v2.html (https://sagemaker.readthedocs.io/en/stable/v2.html)
for details.
```

In [79]:

```
# Creating "train", "validation" channels to feed in the model
# Source: https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-algo-docker-registry-paths.html

train_input = sagemaker.session.s3_input(s3_data = s3_train_data, content_type='csv',s3_data_type = 'S3Prefix')
valid_input = sagemaker.session.s3_input(s3_data = s3_validation_data, content_type='csv',s3_data_type = 'S3Pref


data_channels = {'train': train_input,'validation': valid_input}


Xgboost_regressor1.fit(data_channels)
```

```
The class sagemaker.session.s3_input has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html (https://sagemaker.readthedocs.io/en/stable/v2.html)
for details.
The class sagemaker.session.s3_input has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html (https://sagemaker.readthedocs.io/en/stable/v2.html)
for details.
```

TASK #11: DEPLOY THE MODEL TO MAKE PREDICTIONS

In [104]:
```
# Deploy the model to perform inference

Xgboost_regressor = Xgboost_regressor1.deploy(initial_instance_count = 1, instance_type = 'ml.m5.2xlarge')
```

```
-------------!
```

In [105]:
```
'''
Content type over-rides the data that will be passed to the deployed model, since the deployed model expects dat
in text/csv format, we specify this as content -type.

Serializer accepts a single argument, the input data, and returns a sequence of bytes in the specified content
type

Reference: https://sagemaker.readthedocs.io/en/stable/predictors.html
'''
from sagemaker.predictor import csv_serializer, json_deserializer


Xgboost_regressor.serializer = csv_serializer
```

In [106]:
```
X_test.shape()
```

Out[106]:
```
array([0.0000000e+00, 8.3940002e+01, 3.5940001e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       2.1849796e+02, 6.2969999e+00, 1.5507800e+05, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 1.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       1.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
       0.0000000e+00], dtype=float32)
```

In [107]:
```
# making prediction

predictions1 = Xgboost_regressor.predict(X_test[0:10000])
```

```
The csv_serializer has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html (https://sagemaker.readthedocs.io/en/stable/v2.html) f
or details.
```

In [108]:
```
predictions2 = Xgboost_regressor.predict(X_test[10000:20000])
```

```
The csv_serializer has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html (https://sagemaker.readthedocs.io/en/stable/v2.html) f
or details.
```

In [109]:
```
predictions3 = Xgboost_regressor.predict(X_test[20000:30000])
```

```
The csv_serializer has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html (https://sagemaker.readthedocs.io/en/stable/v2.html) f
or details.
```

In [110]:
```
predictions4 = Xgboost_regressor.predict(X_test[30000:31618])
```

```
The csv_serializer has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html (https://sagemaker.readthedocs.io/en/stable/v2.html) f
or details.
```

In [111]:
```
predictions4
```

Out[111]:
```
b'1999.1929931640625,3078.161865234375,4414.736328125,11633.8515625,-463.63836669921875,11362.0048828125,415
78.640625,36161.68359375,602.8242797851562,41789.234375,12402.9580078125,554.1234741210938,3380.978515625,36
330.34375,16914.1875,28990.048828125,4246.6123046875,17728.220703125,27032.330078125,13092.6416015625,32869.
5546875,6483.7578125,25817.921875,4401.28857421875,28091.416015625,7997.95703125,21320.953125,16551.859375,1
2514.013671875,32213.34765625,8867.73046875,4201.767578125,16387.240234375,5738.65283203125,2404.47436523437
5,2604.037841796875,44783.96875,7274.0634765625,22599.21875,8157.22607421875,38943.390625,8834.017578125,261
63.22265625,8505.078125,28946.072265625,27573.875,872.2776489257812,2361.410888671875,13053.7080078125,4590.
64404296875,12168.9501953125,11649.18359375,8906.0009765625,6870.8544921875,49789.0859375,8500.6875,3929.152
587890625,8186.830078125,19732.755859375,5798.0888671875,19480.546875,10384.32421875,12705.80859375,33913.12
5,10727.3212890625,9039.345703125,48937.2421875,16747.0703125,11274.0400390625,20981.955078125,3885.46728515
625,8479.6396484375,18074.9921875,5983.62646484375,16441.66015625,-338.8611145019531,4756.0,36873.7265625,88
03.7958984375,8204.9814453125,8077.28173828125,38030.77734375,5977.6865234375,20757.244140625,22230.4765625,
32084.669921875,7444.47509765625,24784.24609375,4610.44384765625,7933.7421875,18390.2734375,11670.2578125,26
677.16796875,5578.78955078125,6049.763671875,16846.861328125,14495.8623046875,15375.416015625,11267.75683593
75,6694.017578125,23236.736328125,11168.44921875,3593.521240234375,4954.36181640625,5783.1259765625,7499.485
3515625,11491.8759765625,16478.16015625,6035.986328125,4161.17431640625,5256.97265625,-989.1417236328125,347
24.91796875,53325.35546875,10156.076171875,780.858642578125,5501.49365234375,5307.56396484375,67498.9609375,
5271.1181640625,3935.5927734375,4135.5537109375,42683.50390625,8447.21875,7916.408203125,5443.00341796875,39
944.5703125,4229.0009765625,3565.88623046875,8700.818359375,33300.4453125,14466.7763671875,23672.849609375,6
```

In [112]:
```
# custom code to convert the values in bytes format to array

def bytes_2_array(x):

    # makes entire prediction as string and splits based on ','
    l = str(x).split(',')

    # Since the first element contains unwanted characters like (b,',') we remove them
    l[0] = l[0][2:]
    #same-thing as above remove the unwanted last character (')
    l[-1] = l[-1][:-1]

    # iterating through the list of strings and converting them into float type
    for i in range(len(l)):
        l[i] = float(l[i])

    # converting the list into array
    l = np.array(l).astype('float32')

    # reshape one-dimensional array to two-dimensional array
    return l.reshape(-1,1)
```

In [113]:
```
predicted_values_1 = bytes_2_array(predictions1)
```

In [114]:
```
predicted_values_1.shape
```

Out[114]:
```
(10000, 1)
```

In [115]:
```
predicted_values_2 = bytes_2_array(predictions2)
predicted_values_2.shape
```

Out[115]:
```
(10000, 1)
```

In [116]:
```
predicted_values_3 = bytes_2_array(predictions3)
predicted_values_3.shape
```

Out[116]:
```
(10000, 1)
```

In [117]:
```
predicted_values_4 = bytes_2_array(predictions4)
predicted_values_4.shape
```

Out[117]:
```
(1618, 1)
```

In [118]:
```
predicted_values = np.concatenate((predicted_values_1, predicted_values_2, predicted_values_3, predicted_values_
```

In [119]:
```
predicted_values.shape
```

Out[119]:
```
(31618, 1)
```

In [120]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt
k = X_test.shape[1]
n = len(X_test)
RMSE = float(format(np.sqrt(mean_squared_error(y_test, predicted_values)),'.3f'))
MSE = mean_squared_error(y_test, predicted_values)
MAE = mean_absolute_error(y_test, predicted_values)
r2 = r2_score(y_test, predicted_values)
adj_r2 = 1-(1-r2)*(n-1)/(n-k-1)

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =', adj_r2)
```

```
RMSE = 7492.593
MSE = 56138950.0
MAE = 4353.634
R2 = 0.8939039998714412
Adjusted R2 = 0.8934422733143379
```

In [121]:
```
# Delete the end-point

Xgboost_regressor.delete_endpoint()
```

TASK #12: PERFORM HYPERPARAMETERS OPTIMIZATION

See Slides for detailed steps

TASK #13: TRAIN THE MODEL WITH BEST PARAMETERS

In [190]:
```
# We have pass in the container, the type of instance that we would like to use for training
# output path and sagemaker session into the Estimator.
# We can also specify how many instances we would like to use for training

Xgboost_regressor = sagemaker.estimator.Estimator(container,
                                       role,
                                       train_instance_count=1,
                                       train_instance_type='ml.m4.xlarge',
                                       output_path=output_location,
                                       sagemaker_session=sagemaker_session)

# We can tune the hyper-parameters to improve the performance of the model
Xgboost_regressor.set_hyperparameters(max_depth=25,
                           objective='reg:linear',
                           colsample_bytree = 0.3913546819101119,
                           alpha = 1.0994354985124635,
                           eta = 0.23848185159806115,
                           num_round = 237
                           )
```

In [191]:
```
train_input = sagemaker.session.s3_input(s3_data = s3_train_data, content_type='csv',s3_data_type = 'S3Prefix')
valid_input = sagemaker.session.s3_input(s3_data = s3_validation_data, content_type='csv',s3_data_type = 'S3Pref
data_channels = {'train': train_input,'validation': valid_input}
Xgboost_regressor.fit(data_channels)
```

```
2020-05-22 07:36:14 Starting - Starting the training job...
2020-05-22 07:36:16 Starting - Launching requested ML instances.........
2020-05-22 07:38:00 Starting - Preparing the instances for training......
2020-05-22 07:38:55 Downloading - Downloading input data......
2020-05-22 07:40:05 Training - Downloading the training image...
2020-05-22 07:40:25 Training - Training image download completed. Training in progress.INFO:sagemaker-contai
ners:Imported framework sagemaker_xgboost_container.training
INFO:sagemaker-containers:Failed to parse hyperparameter objective value reg:linear to Json.
Returning the value itself
INFO:sagemaker-containers:No GPUs detected (normal if no gpus installed)
INFO:sagemaker_xgboost_container.training:Running XGBoost Sagemaker in algorithm mode
INFO:root:Determined delimiter of CSV input is ','
INFO:root:Determined delimiter of CSV input is ','
INFO:root:Determined delimiter of CSV input is ','
[07:40:30] 358334x138 matrix with 49450092 entries loaded from /opt/ml/input/data/train?format=csv&label_col
umn=0&delimiter=,
INFO:root:Determined delimiter of CSV input is ','
[07:40:30] 31618x138 matrix with 4363284 entries loaded from /opt/ml/input/data/validation?format=csv&label_
column=0&delimiter=,
```

In [192]:
```
# Deploying the model to perform inference

Xgboost_regressor = Xgboost_regressor.deploy(initial_instance_count = 1,
                                          instance_type = 'ml.m4.xlarge')
```

```
---------------!
```

In [194]:
```
from sagemaker.predictor import csv_serializer, json_deserializer

# Xgboost_regressor.content_type = 'text/csv'
Xgboost_regressor.serializer = csv_serializer
```

In [ ]:
```
# Try to make inference with the entire testing dataset (Crashes!)
predictions = Xgboost_regressor.predict(X_test)
predicted_values = bytes_2_array(predictions)
```

In [196]:
```
predictions1 = Xgboost_regressor.predict(X_test[0:10000])
```

In [197]:
```
predicted_values_1 = bytes_2_array(predictions1)
predicted_values_1.shape
```

Out[197]: (10000, 1)

In [198]:
```
predictions2 = Xgboost_regressor.predict(X_test[10000:20000])
predicted_values_2 = bytes_2_array(predictions2)
predicted_values_2.shape
```

Out[198]: (10000, 1)

In [199]:
```
predictions3 = Xgboost_regressor.predict(X_test[20000:30000])
predicted_values_3 = bytes_2_array(predictions3)
predicted_values_3.shape
```

Out[199]: (10000, 1)

In [200]:
```
predictions4 = Xgboost_regressor.predict(X_test[30000:31618])
predicted_values_4 = bytes_2_array(predictions4)
predicted_values_4.shape
```

Out[200]: (1618, 1)

In [201]:
```
predicted_values = np.concatenate((predicted_values_1, predicted_values_2, predicted_values_3, predicted_values_
```

In [202]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt
k = X_test.shape[1]
n = len(X_test)
RMSE = float(format(np.sqrt(mean_squared_error(y_test, predicted_values)),'.3f'))
MSE = mean_squared_error(y_test, predicted_values)
MAE = mean_absolute_error(y_test, predicted_values)
r2 = r2_score(y_test, predicted_values)
adj_r2 = 1-(1-r2)*(n-1)/(n-k-1)

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =', adj_r2)
```

```
RMSE = 4266.012
MSE = 18198860.0
MAE = 1811.6404
R2 = 0.9638345632190437
Adjusted R2 = 0.9636760184661681
```

In [203]:
```
# Delete the end-point

Xgboost_regressor.delete_endpoint()
```

In [ ]:

In [ ]:

In [ ]: