Walmart_Sales_Forecasting

February 12, 2025

[428]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib as mpl
import math
from datetime import datetime
from datetime import timedelta

from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import make_pipeline, Pipeline

from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.tsa.arima_model import ARIMA

#!pip install pmdarima
from pmdarima.utils import decomposed_plot
from pmdarima.arima import decompose
from pmdarima import auto_arima

from statsmodels.tsa.holtwinters import ExponentialSmoothing

import warnings
warnings.filterwarnings("ignore")
```

[3]:
```
df_store = pd.read_csv("C:/Users/aryan/Downloads/archive (1)/stores.csv")
```

[5]:
```
df_features = pd.read_csv("C:/Users/aryan/Downloads/archive (1)/features.csv")
```

[7]:
```
df_train = pd.read_csv("C:/Users/aryan/Downloads/archive (1)/train.csv")
```

[9]:
```
df_store.head()
```

[9]:

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392

```
3      4    A  205863
4      5    B   34875
```

[11]:
```
df_features.head()
```

[11]:
```
   Store        Date  Temperature  Fuel_Price  MarkDown1  MarkDown2  \
0      1  2010-02-05        42.31       2.572        NaN        NaN
1      1  2010-02-12        38.51       2.548        NaN        NaN
2      1  2010-02-19        39.93       2.514        NaN        NaN
3      1  2010-02-26        46.63       2.561        NaN        NaN
4      1  2010-03-05        46.50       2.625        NaN        NaN

   MarkDown3  MarkDown4  MarkDown5         CPI  Unemployment  IsHoliday
0        NaN        NaN        NaN  211.096358         8.106      False
1        NaN        NaN        NaN  211.242170         8.106       True
2        NaN        NaN        NaN  211.289143         8.106      False
3        NaN        NaN        NaN  211.319643         8.106      False
4        NaN        NaN        NaN  211.350143         8.106      False
```

[13]:
```
df_train.head()
```

[13]:
```
   Store  Dept        Date  Weekly_Sales  IsHoliday
0      1     1  2010-02-05      24924.50      False
1      1     1  2010-02-12      46039.49       True
2      1     1  2010-02-19      41595.55      False
3      1     1  2010-02-26      19403.54      False
4      1     1  2010-03-05      21827.90      False
```

[15]:
```
df = df_train.merge(df_features, on = ['Store', 'Date'], how = 'inner').
 ↪merge(df_store, on = ['Store'], how = 'inner')
```

[17]:
```
df.head()
```

[17]:
```
   Store  Dept        Date  Weekly_Sales  IsHoliday_x  Temperature  \
0      1     1  2010-02-05      24924.50        False        42.31
1      1     2  2010-02-05      50605.27        False        42.31
2      1     3  2010-02-05      13740.12        False        42.31
3      1     4  2010-02-05      39954.04        False        42.31
4      1     5  2010-02-05      32229.38        False        42.31

   Fuel_Price  MarkDown1  MarkDown2  MarkDown3  MarkDown4  MarkDown5  \
0       2.572        NaN        NaN        NaN        NaN        NaN
1       2.572        NaN        NaN        NaN        NaN        NaN
2       2.572        NaN        NaN        NaN        NaN        NaN
3       2.572        NaN        NaN        NaN        NaN        NaN
4       2.572        NaN        NaN        NaN        NaN        NaN
```

```
          CPI  Unemployment  IsHoliday_y Type    Size
0  211.096358         8.106        False    A  151315
1  211.096358         8.106        False    A  151315
2  211.096358         8.106        False    A  151315
3  211.096358         8.106        False    A  151315
4  211.096358         8.106        False    A  151315
```

[19]:
```
df.drop(['IsHoliday_y'], axis = 1, inplace = True)
```

[21]:
```
df.rename(columns = {"IsHoliday_x":"IsHoliday"}, inplace = True)
```

[23]:
```
df.head()
```

[23]:
```
   Store  Dept        Date  Weekly_Sales  IsHoliday  Temperature  Fuel_Price  \
0      1     1  2010-02-05      24924.50      False        42.31       2.572
1      1     2  2010-02-05      50605.27      False        42.31       2.572
2      1     3  2010-02-05      13740.12      False        42.31       2.572
3      1     4  2010-02-05      39954.04      False        42.31       2.572
4      1     5  2010-02-05      32229.38      False        42.31       2.572

   MarkDown1  MarkDown2  MarkDown3  MarkDown4  MarkDown5         CPI  \
0        NaN        NaN        NaN        NaN        NaN  211.096358
1        NaN        NaN        NaN        NaN        NaN  211.096358
2        NaN        NaN        NaN        NaN        NaN  211.096358
3        NaN        NaN        NaN        NaN        NaN  211.096358
4        NaN        NaN        NaN        NaN        NaN  211.096358

   Unemployment Type    Size
0         8.106    A  151315
1         8.106    A  151315
2         8.106    A  151315
3         8.106    A  151315
4         8.106    A  151315
```

[25]:
```
df.shape
```

[25]: (421570, 16)

[27]:
```
df['Store'].nunique()
```

[27]: 45

[29]:
```
df['Dept'].nunique()
```

[29]: 81

There are 45 different stores and 81 different departments.

[32]:
```python
store_dept_table = pd.pivot_table(df, index='Store', columns='Dept',
                                  values='Weekly_Sales', aggfunc=np.mean)
display(store_dept_table)
```

```
C:\Users\aryan\AppData\Local\Temp\ipykernel_23232\2681735725.py:1:
FutureWarning: The provided callable <function mean at 0x000001AB201463E0> is
currently using DataFrameGroupBy.mean. In a future version of pandas, the
provided callable will be used directly. To keep current behavior pass the
string "mean" instead.
  store_dept_table = pd.pivot_table(df, index='Store', columns='Dept',
```

```
Dept              1              2             3             4             5   \
Store
1      22513.322937   46102.090420  13150.478042  36964.154476  24257.941119
2      30777.980769   65912.922517  17476.563357  45607.666573  30555.315315
3       7328.621049   16841.775664   5509.300769   8434.186503  11695.366573
4      36979.940070   93639.315385  19012.491678  56603.400140  45668.406783
5       9774.553077   12317.953287   4101.085175   9860.806783   6699.202238
6      23867.553776   50269.437273  16806.638811  34187.366503  34465.307622
7       9542.801259   22603.690769   8633.536923  14950.518601  13860.350490
8      14789.827343   35729.821748  10683.305105  21089.309301  19838.849231
9      11846.558252   24969.477413   7497.356783  17165.947762  19282.746014
10     39925.138951  109795.291469  32086.181469  48579.826364  58373.460280
11     18860.911958   57114.326224  17628.778671  28837.744545  36663.363916
12     17330.087622   74494.846224  17535.251678  26673.788182  27756.204615
13     47020.455455   76339.960000  26116.623706  42563.275455  56786.934755
14     30611.783357   77704.857972  19418.273986  52936.323287  33468.325035
15     13845.747832   26317.410769  10470.811958  13082.172448  16465.706993
16     11352.479371   23549.144965   7635.427273  14748.078112  13494.538671
17     22801.609161   42231.844406  19278.955035  23961.357273  27082.325594
18     21988.356224   63665.139510  16392.980490  26775.207203  22933.954965
19     21504.029161   50841.072937  18414.224476  31365.545315  28759.223846
20     40545.473217   78251.249930  15490.971259  51456.376643  41647.786503
21     14950.049231   47780.599161  14607.126923  19354.728042  16090.874545
22     21493.271119   53361.851888  13150.979510  32104.132378  23187.335105
23     33186.460559   70522.580140  19912.564755  27324.303077  36895.869021
24     18859.023357   40797.169301  11825.589021  29245.357552  29178.058811
25     20145.897483   36871.310559  11788.130979  20351.455455  12422.996434
26     19402.762937   27398.030979   7357.400769  24498.113846  17589.532587
27     30437.976224   79001.049161  20226.734615  43596.933916  28059.038252
28     20180.453986   57751.274336  12562.223287  27980.817203  28221.618392
29     15504.699580   25181.662727   7995.955804  14326.216224  12931.821259
30      9788.376643   12974.464476    739.981888  13216.100909    405.565944
31     17356.652448   58512.131538  10616.675944  34848.899231  18715.630769
32     22852.639510   50323.497343  15472.540140  28137.154965  20748.371888
33      2379.086573    7471.425105    283.950140   6107.616014    112.728310
34     19947.573077   34916.225874   8377.376434  19791.509021  21633.900559
35     17082.647902   45578.456224  14308.382797  19495.631119  24858.433706
```

36	2239.227413	13416.025664	381.324266	9873.505105	314.753982
37	11024.235874	16511.446224	1297.862028	17614.013636	1137.631189
38	6923.538531	10986.172657	498.700210	10669.501329	397.418322
39	21925.021189	67338.429371	20569.701608	44807.515105	24043.436783
40	18794.578811	26702.705175	6489.030350	24386.750559	17689.671678
41	23205.259930	48349.828951	17021.833357	30538.574895	25513.943776
42	10375.148392	15976.902448	814.451189	14885.264755	1052.296783
43	7549.109021	20722.851469	999.648881	18227.382168	575.417326
44	8049.992308	9377.273007	571.016713	7403.959580	960.670490
45	17745.916014	35800.912448	9508.014965	24229.873147	16107.063077

Dept	6	7	8	9	10 \
Store					
1	4801.780140	24566.487413	35718.257622	28062.052238	31033.386364
2	6808.382517	40477.837063	58707.369441	34375.864476	38845.854476
3	2012.411818	10044.341608	8310.254196	9062.007692	10871.944126
4	8241.777692	50728.151399	62949.723776	34437.170979	37269.667413
5	1191.057622	6124.484336	13735.709441	7919.805944	9783.395385
6	7225.566643	34526.870420	47577.719790	48271.060140	47436.477902
7	6329.928811	10925.757063	13970.619371	29722.736084	21136.560280
8	3395.425455	20268.743776	26438.524336	11792.661678	20666.433776
9	2806.416364	13826.694336	21424.470699	13196.569720	12810.480350
10	10556.550769	58964.715664	86739.846643	64436.722517	48108.063497
11	5925.281678	34844.108462	34415.449580	19056.162168	23449.992727
12	6741.174895	34242.449161	42229.665035	19553.030490	17975.211119
13	7886.826993	59896.738601	36238.867972	41236.445175	29431.879231
14	7016.829790	53256.150280	53425.359860	22025.603497	20165.667133
15	4244.143776	22267.220070	20416.967273	15954.692937	11524.856294
16	5146.038951	11544.310140	14676.778322	28990.377343	12681.776643
17	5944.435245	19474.770559	20110.270839	27293.658042	14165.000000
18	5664.913077	33152.347203	32036.582098	18589.371259	16754.599860
19	5948.962867	33882.926853	42613.662937	30645.018112	27622.457762
20	8210.745734	49394.699231	76445.061259	38243.623916	41826.467552
21	3988.656294	24456.825664	18238.059790	16387.963636	14695.978881
22	5236.811329	29068.621608	37236.347692	23452.908881	19438.354266
23	7393.499650	43624.067413	36710.240909	50178.361748	31155.170559
24	4911.185804	28788.329441	49171.841748	23246.748322	27175.089231
25	3760.045035	17971.439580	29858.353636	14636.113636	20202.701469
26	4656.670490	16287.658531	28694.950909	16556.330769	10172.815734
27	7730.729091	43272.914965	42181.469580	29315.697133	36757.327413
28	5016.258671	29228.446923	33375.575524	17930.710070	21083.404825
29	3289.884965	16854.082238	20680.465944	11370.866364	9400.183077
30	27.303937	379.771958	11733.993776	76.845352	196.116923
31	3489.809441	21012.438531	25277.976713	10815.516713	19911.584406
32	4589.748392	25375.036993	24681.349580	20739.684685	22887.257483
33	11.996538	392.912867	3679.792168	42.788348	80.301189
34	3419.062028	18055.491608	27165.013147	17224.253497	16957.163566
35	7256.417133	30267.589790	18416.401678	15657.032937	14818.443706

36	26.291579	414.428322	3417.640420	102.211739	175.052308
37	46.313630	824.978392	16151.397902	151.233803	387.644685
38	37.014855	413.539021	9485.399441	77.520350	365.364895
39	4911.540420	40020.492867	36130.641608	19396.117692	14919.373916
40	4003.068601	18898.214336	33971.532238	19065.436294	19612.629301
41	5267.832098	33711.105734	33729.081678	32743.470140	15194.223706
42	3.333333	721.913846	18238.584196	135.524056	404.596014
43	37.843246	516.772867	13185.211678	147.694196	507.426713
44	34.648722	531.034895	4963.966224	99.817273	153.792657
45	3554.222657	23757.771538	34050.409580	15485.885804	14245.086993

Dept	…	90	91	92	93	\
Store	…					
1	…	82427.547832	64238.943427	135458.969510	71699.182378	
2	…	97611.537133	80610.380350	164840.230979	70581.977063	
3	…	1540.049161	318.685594	7568.280210	NaN	
4	…	89248.965524	66535.407203	159365.107902	67815.163007	
5	…	3059.520000	1457.221678	7759.205594	NaN	
6	…	53715.366084	45270.405175	99024.796503	41359.651189	
7	…	13858.405874	10263.880000	26530.890559	1328.178252	
8	…	39333.566154	31530.560909	60465.630000	27515.635315	
9	…	2981.249510	869.273287	14123.063147	21.240000	
10	…	14291.869790	12703.554406	50450.731958	1420.418462	
11	…	48995.984196	42030.370699	77392.741608	32623.853706	
12	…	11060.175455	6779.841469	24682.599161	562.897203	
13	…	115592.108042	81272.990979	162034.099301	50024.937203	
14	…	107174.743986	91406.434615	182527.956014	62088.622937	
15	…	5345.240420	3414.740909	18262.376853	422.878252	
16	…	6922.744685	3331.204965	20446.967832	997.032281	
17	…	31293.306224	12033.678951	53043.348741	3646.955664	
18	…	18481.394266	14124.482517	50079.623636	2113.300147	
19	…	67545.406434	54692.797413	113720.212937	37087.937063	
20	…	95858.587343	78493.190140	164633.741538	52818.583706	
21	…	10983.598741	6735.454126	21915.114965	663.384126	
22	…	21413.411608	21405.250629	51603.339091	2531.663986	
23	…	20814.992168	19604.867692	59604.574615	2111.610780	
24	…	72650.442867	52435.498252	121882.073916	37876.836853	
25	…	11932.596503	7767.272098	38854.460699	777.747483	
26	…	57016.589231	39434.281259	84988.311818	25615.331469	
27	…	96374.536573	66687.096573	146518.141399	54910.693776	
28	…	65285.952098	57575.601119	98486.960350	47923.508671	
29	…	10950.327972	4691.213217	25166.714266	1190.882098	
30	…	34622.986224	31576.583986	53256.041399	22409.698392	
31	…	86167.265804	70232.133566	127010.118601	57876.205664	
32	…	61639.637133	48368.756154	100122.929021	30732.226923	
33	…	24899.923147	9862.862867	34227.662867	25648.054266	
34	…	44338.936783	34018.102238	67782.520909	29590.111329	
35	…	12960.450210	7919.080140	33776.032028	1528.451469	

36	…	35474.191958	11097.875874	44539.564476	26103.315664
37	…	44144.428112	30870.677063	59440.577133	21599.851049
38	…	34765.576783	25404.860420	45314.434825	18868.919091
39	…	78649.534685	60386.286014	110126.209580	39684.510000
40	…	61258.202867	43256.156853	96475.753287	27532.751189
41	…	70852.021818	52714.928462	115827.664056	35415.340000
42	…	53384.897902	42913.221259	83497.778671	32852.632308
43	…	63668.895594	34808.442168	83646.160909	36196.693217
44	…	31182.601818	18169.510070	39619.563287	11029.915734
45	…	23674.035245	16641.927343	48125.897762	2728.627133

Dept	94	95	96	97	98	\
Store						
1	63180.568182	120772.062168	33251.831639	35207.348811	11827.770769	
2	70018.672517	143588.751888	34319.063846	40697.204056	14035.400839	
3	656.294444	15745.528252	3934.540000	343.437357	30.570833	
4	68159.106573	147236.473706	38346.573077	39339.238951	15009.249371	
5	411.431486	19340.693986	5985.671119	667.070315	29.976087	
6	41701.693497	89208.786294	30450.542238	20637.667063	9728.100629	
7	699.332522	34208.097273	1123.383217	4374.927902	260.886596	
8	25442.578042	62951.463706	16.986667	16978.366503	6880.466434	
9	599.112568	29575.050769	3596.107762	372.655556	27.930000	
10	393.833168	73344.654685	11079.676643	5323.506503	198.179091	
11	37474.038531	77487.279091	21685.298811	16596.197552	9570.351469	
12	355.264000	43405.853357	6.441176	2394.894755	747.609860	
13	75522.874406	136844.834056	9165.079930	27556.759231	14980.825385	
14	64541.165664	144446.932517	5.193846	25684.497762	17768.013706	
15	272.906250	27291.017133	2784.158881	2071.211888	273.504884	
16	673.280928	27385.769231	126.934126	2116.696993	42.618571	
17	855.782273	50614.958462	819.416458	7798.283427	169.379120	
18	4880.242248	57668.251748	0.481333	5350.500432	881.150853	
19	37643.786434	97240.503566	15860.814825	20370.269720	12884.229091	
20	63148.334965	150613.955385	15.266875	25836.062238	19284.377343	
21	537.663333	40379.295175	2.000000	3260.404685	111.680672	
22	857.190894	57868.571119	6.243000	4582.594755	177.560576	
23	374.898804	54199.088322	13168.146713	6149.684755	100.585083	
24	51850.045105	93927.992098	13623.074615	18597.824126	9878.970140	
25	2607.109754	43991.147692	-1.270000	2706.628252	665.919779	
26	42544.202028	70236.827622	18596.331888	14830.084825	8025.948601	
27	69638.930420	119519.410909	20806.990909	21268.805734	11524.137832	
28	36164.364615	96322.113846	26288.955734	23828.861329	10673.133077	
29	263.083012	30980.395594	11.800000	2131.676783	139.677971	
30	24522.622587	45456.508322	19163.112028	13172.531119	3207.034685	
31	68732.141818	106696.019231	30335.294266	31144.978112	10101.886713	
32	48650.040979	84695.234196	2308.411818	17160.310000	7939.262378	
33	29002.624476	27022.949161	9371.822168	5375.769510	7340.692168	
34	37428.096923	69245.187972	19154.212308	17570.577483	7775.998182	
35	200.270435	43286.536993	10.788333	3738.292517	68.284831	

36	47372.151119	39735.688741	15683.341818	6469.273636	9009.943776
37	33656.648112	51410.551119	20375.380769	13960.701399	5286.761119
38	21331.411259	41793.649021	11981.676643	9902.368182	4783.086713
39	59830.190280	103036.757133	27089.158601	23993.406853	9767.295734
40	38210.900699	66572.881259	15309.077972	17131.033497	8178.371049
41	47218.529161	88666.468392	2883.492238	19789.219231	9371.531608
42	35724.612098	61205.272308	15183.474196	17495.198811	6540.721259
43	50769.708322	72883.223287	25058.369371	19349.989930	9594.867483
44	23812.046993	31100.185175	2834.139580	6636.467413	3466.077063
45	3690.272090	52896.166643	2.970000	6466.961888	561.239037

Dept	99
Store	
1	306.091081
2	475.896905
3	NaN
4	623.182381
5	NaN
6	388.636750
7	15.000000
8	298.153714
9	NaN
10	NaN
11	520.938125
12	29.880000
13	732.604651
14	635.556047
15	29.880000
16	59.760000
17	2.290000
18	12.560000
19	440.374878
20	796.153864
21	29.880000
22	27.150000
23	29.880000
24	413.774211
25	NaN
26	221.950278
27	562.980000
28	316.605610
29	29.880000
30	-0.641818
31	218.742203
32	379.147250
33	0.022000
34	347.144324
35	NaN

```
36        0.020000
37       15.000000
38       25.000000
39      334.869756
40      167.374167
41      443.736512
42             NaN
43       26.250000
44        3.505000
45             NaN

[45 rows x 81 columns]
```

[34]:
```python
df.groupby(['Store', 'Dept'])['Weekly_Sales'].mean().reset_index()
```

[34]:
```
      Store  Dept  Weekly_Sales
0         1     1  22513.322937
1         1     2  46102.090420
2         1     3  13150.478042
3         1     4  36964.154476
4         1     5  24257.941119
…       …     …           …
3326     45    94   3690.272090
3327     45    95  52896.166643
3328     45    96      2.970000
3329     45    97   6466.961888
3330     45    98    561.239037

[3331 rows x 3 columns]
```

[36]:
```python
df.loc[df['Weekly_Sales']<=0]
```

[36]:
```
        Store  Dept        Date  Weekly_Sales  IsHoliday  Temperature  \
188         1    47  2010-02-19       -863.00      False        39.93
406         1    47  2010-03-12       -698.00      False        57.79
2549        1    47  2010-10-08        -58.00      False        63.93
3632        1    54  2011-01-21        -50.00      False        44.04
4132        1    47  2011-03-11          0.00      False        53.56
…       …     …         …            …         …           …
420066     45    49  2012-05-25         -4.97      False        67.21
420403     45    49  2012-06-29        -34.00      False        75.22
420736     45    49  2012-08-03         -1.91      False        76.58
421007     45    54  2012-08-31          0.00      False        75.09
421142     45    49  2012-09-14         -6.83      False        67.87

        Fuel_Price  MarkDown1  MarkDown2  MarkDown3  MarkDown4  MarkDown5  \
188          2.514        NaN        NaN        NaN        NaN        NaN
406          2.667        NaN        NaN        NaN        NaN        NaN
```

```
2549          2.633        NaN        NaN        NaN        NaN        NaN
3632          3.016        NaN        NaN        NaN        NaN        NaN
4132          3.459        NaN        NaN        NaN        NaN        NaN
…             …          …        …        …        …        …
420066        3.798    5370.39        NaN     361.22    1287.62    2461.81
420403        3.506    3291.36     425.60        NaN     314.88    2255.34
420736        3.654   24853.05      39.56      17.96   11142.69    2768.32
421007        3.867   23641.30       6.00      92.93    6988.31    3992.13
421142        3.948   11407.95        NaN       4.30    3421.72    5268.92

               CPI  Unemployment Type    Size
188     211.289143         8.106    A  151315
406     211.380643         8.106    A  151315
2549    211.746754         7.838    A  151315
3632    211.827234         7.742    A  151315
4132    214.111056         7.742    A  151315
…             …             …   …     …
420066  191.002810         8.567    B  118221
420403  191.099246         8.567    B  118221
420736  191.164090         8.684    B  118221
421007  191.461281         8.684    B  118221
421142  191.699850         8.684    B  118221

[1358 rows x 16 columns]
```

There are total of 421570 rows in which 1358 rows are either zero or negative. This makes 0.3% of the rows negative/zero. We can drop these rows to get the dataframe with correct values.

[39]:
```
df = df.loc[df['Weekly_Sales']>0]
```

[41]:
```
df
```

[41]:
```
        Store  Dept        Date  Weekly_Sales  IsHoliday  Temperature  \
0           1     1  2010-02-05      24924.50      False        42.31
1           1     2  2010-02-05      50605.27      False        42.31
2           1     3  2010-02-05      13740.12      False        42.31
3           1     4  2010-02-05      39954.04      False        42.31
4           1     5  2010-02-05      32229.38      False        42.31
…         …    …          …           …        …           …
421565     45    93  2012-10-26       2487.80      False        58.85
421566     45    94  2012-10-26       5203.31      False        58.85
421567     45    95  2012-10-26      56017.47      False        58.85
421568     45    97  2012-10-26       6817.48      False        58.85
421569     45    98  2012-10-26       1076.80      False        58.85

        Fuel_Price  MarkDown1  MarkDown2  MarkDown3  MarkDown4  MarkDown5  \
0            2.572        NaN        NaN        NaN        NaN        NaN
1            2.572        NaN        NaN        NaN        NaN        NaN
```

```
2           2.572       NaN       NaN       NaN       NaN       NaN
3           2.572       NaN       NaN       NaN       NaN       NaN
4           2.572       NaN       NaN       NaN       NaN       NaN
…           …         …       …       …       …       …
421565      3.882     4018.91     58.08     100.0    211.94    858.33
421566      3.882     4018.91     58.08     100.0    211.94    858.33
421567      3.882     4018.91     58.08     100.0    211.94    858.33
421568      3.882     4018.91     58.08     100.0    211.94    858.33
421569      3.882     4018.91     58.08     100.0    211.94    858.33

               CPI  Unemployment Type    Size
0       211.096358         8.106    A  151315
1       211.096358         8.106    A  151315
2       211.096358         8.106    A  151315
3       211.096358         8.106    A  151315
4       211.096358         8.106    A  151315
…              …            …  …     …
421565  192.308899         8.667    B  118221
421566  192.308899         8.667    B  118221
421567  192.308899         8.667    B  118221
421568  192.308899         8.667    B  118221
421569  192.308899         8.667    B  118221

[420212 rows x 16 columns]
```

Now we will look at the Holidays in the dataset.

Now that we have the correct dataset, we can start working on it.

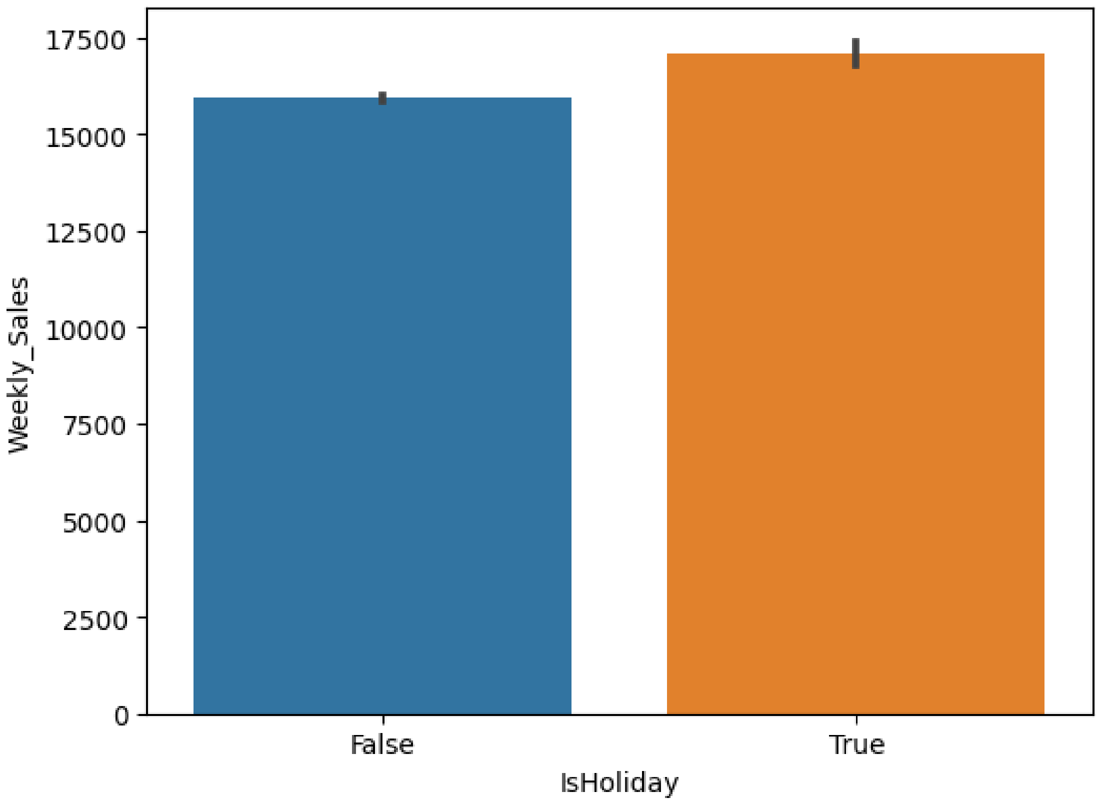
[45]:
```
pd.concat([df['Date'].head(5), df['Date'].tail(5)])
```

[45]:
```
0         2010-02-05
1         2010-02-05
2         2010-02-05
3         2010-02-05
4         2010-02-05
421565    2012-10-26
421566    2012-10-26
421567    2012-10-26
421568    2012-10-26
421569    2012-10-26
Name: Date, dtype: object
```

The date starts from 5th Feb 2010 to 26th October 2012

[48]:
```
sns.barplot(x = 'IsHoliday', y = 'Weekly_Sales', data = df)
```

[48]:
```
<Axes: xlabel='IsHoliday', ylabel='Weekly_Sales'>
```



[50]:
```
df_holiday = df.loc[df['IsHoliday']==True]
df_holiday['Date'].unique()
```

[50]:
```
array(['2010-02-12', '2010-09-10', '2010-11-26', '2010-12-31',
       '2011-02-11', '2011-09-09', '2011-11-25', '2011-12-30',
       '2012-02-10', '2012-09-07'], dtype=object)
```

[52]:
```
df_not_holiday = df.loc[df['IsHoliday']==False]
df_not_holiday['Date'].nunique()
```

[52]:
```
133
```

Here we have used unique() for Holidays as there are few dates, and nunique() for not holidays as there are many dates and we dont want the array, we just want the number of dates.

All holidays are not in the data. There are 4 holiday values such as;

Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12, 8-Feb-13

Labor Day: 10-Sep-10, 9-Sep-11, 7-Sep-12, 6-Sep-13

Thanksgiving: 26-Nov-10, 25-Nov-11, 23-Nov-12, 29-Nov-13

Christmas: 31-Dec-10, 30-Dec-11, 28-Dec-12, 27-Dec-13

[56]:
```
#Taking all the Super Bowl Dates
df['Super_Bowl'] = df['Date'].isin(['2010-02-12', '2011-02-11', '2012-02-12'])
```

[58]:
```
#Taking all the Labor Days Dates
df['Labor_Day'] = df['Date'].isin(['2010-09-10', '2011-09-10', '2012-09-07'])
```

[60]:
```
#Taking all the Thanksgiving Dates
df['Thanksgiving'] = df['Date'].isin(['2010-11-26', '2011-11-25', '2012-11-23'])
```

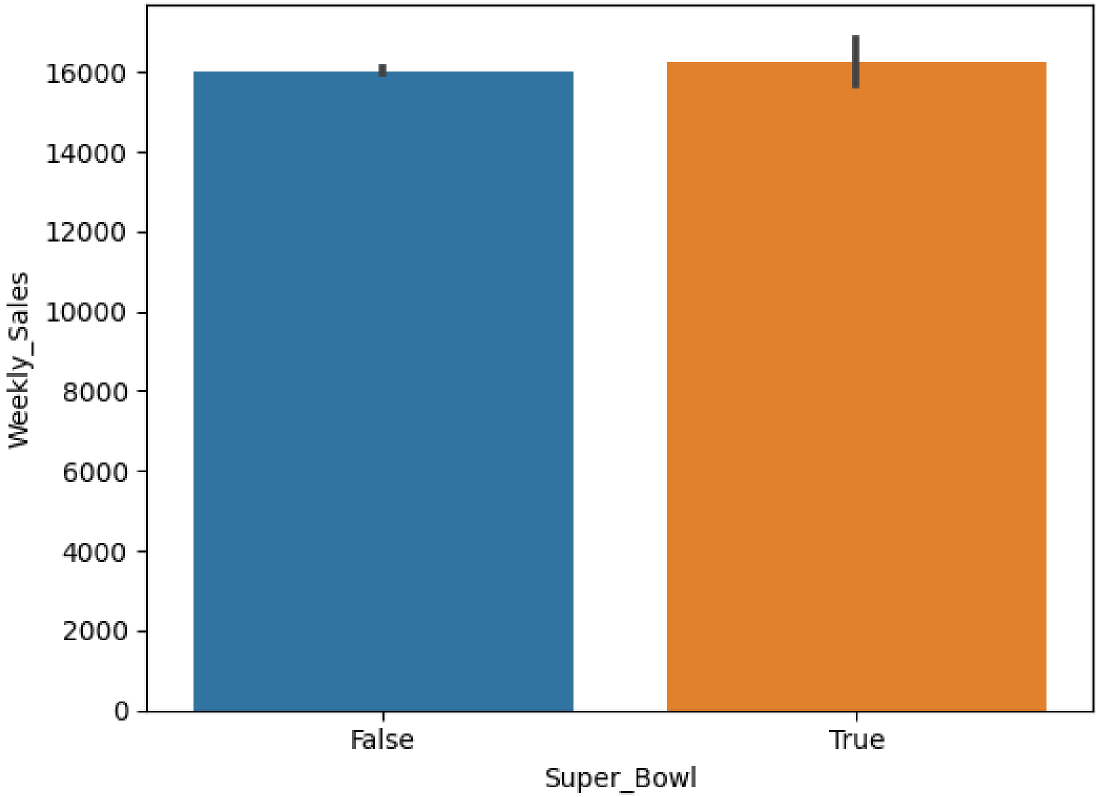
[62]:
```
#Taking all the Christmas Dates
df['Christmas'] = df['Date'].isin(['2010-12-31', '2011-12-30', '2012-12-28'])
```

[64]:
```
df.head(5)
```

[64]:
```
   Store  Dept        Date  Weekly_Sales  IsHoliday  Temperature  Fuel_Price  \
0      1     1  2010-02-05      24924.50      False        42.31       2.572
1      1     2  2010-02-05      50605.27      False        42.31       2.572
2      1     3  2010-02-05      13740.12      False        42.31       2.572
3      1     4  2010-02-05      39954.04      False        42.31       2.572
4      1     5  2010-02-05      32229.38      False        42.31       2.572

   MarkDown1  MarkDown2  MarkDown3  MarkDown4  MarkDown5         CPI  \
0        NaN        NaN        NaN        NaN        NaN  211.096358
1        NaN        NaN        NaN        NaN        NaN  211.096358
2        NaN        NaN        NaN        NaN        NaN  211.096358
3        NaN        NaN        NaN        NaN        NaN  211.096358
4        NaN        NaN        NaN        NaN        NaN  211.096358

   Unemployment Type    Size  Super_Bowl  Labor_Day  Thanksgiving  Christmas
0         8.106    A  151315       False      False         False      False
1         8.106    A  151315       False      False         False      False
2         8.106    A  151315       False      False         False      False
3         8.106    A  151315       False      False         False      False
4         8.106    A  151315       False      False         False      False
```

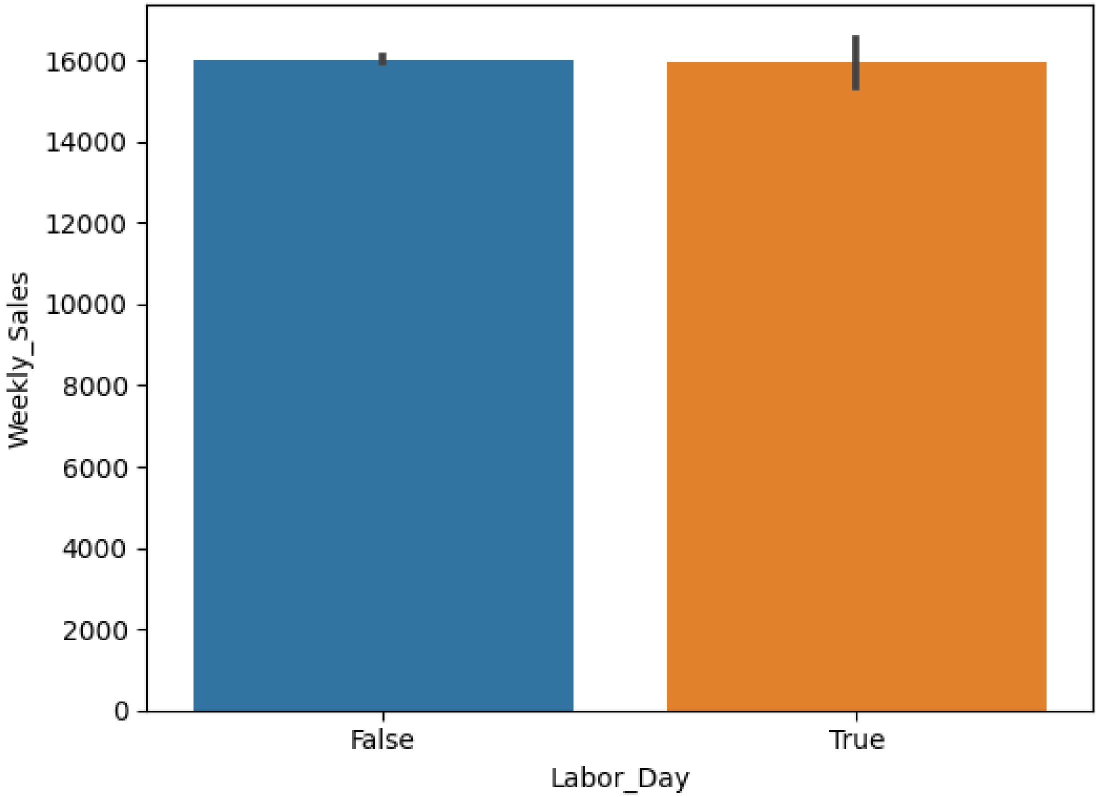
[66]:
```
sns.barplot(x = 'Super_Bowl', y = 'Weekly_Sales', data = df)
```

[66]: <Axes: xlabel='Super_Bowl', ylabel='Weekly_Sales'>



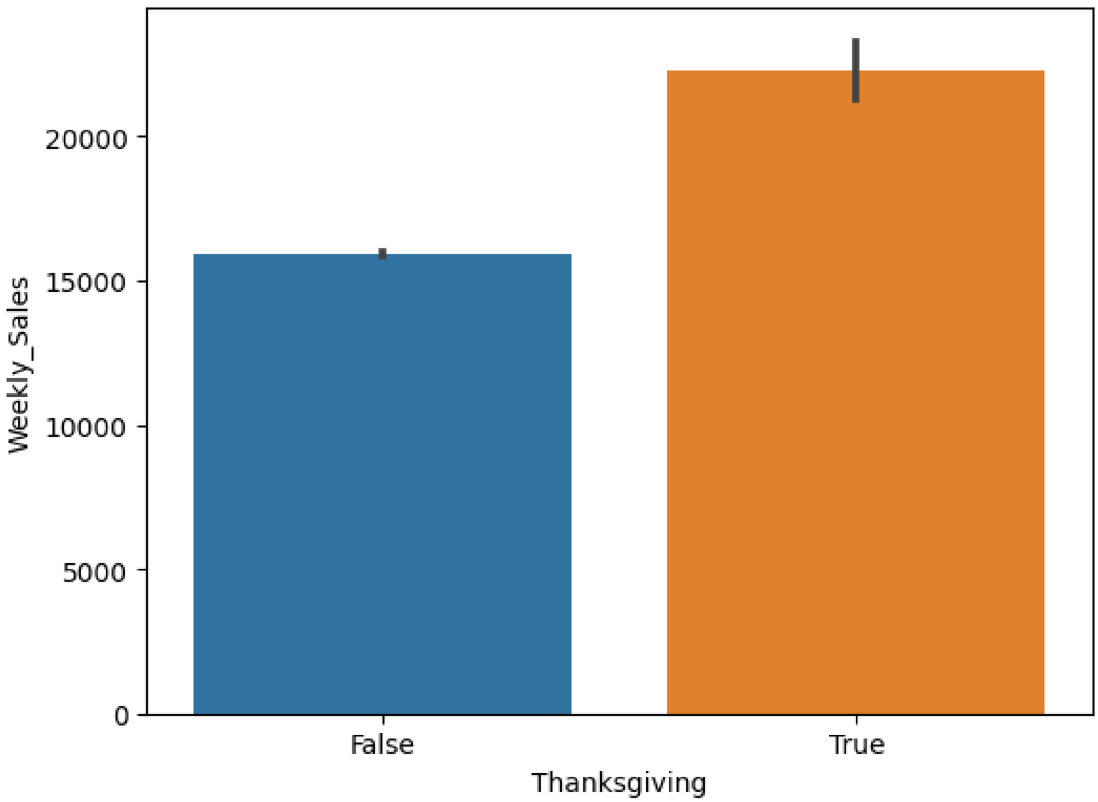
[68]:
```
sns.barplot(x = 'Labor_Day', y = 'Weekly_Sales', data = df)
```

[68]: <Axes: xlabel='Labor_Day', ylabel='Weekly_Sales'>



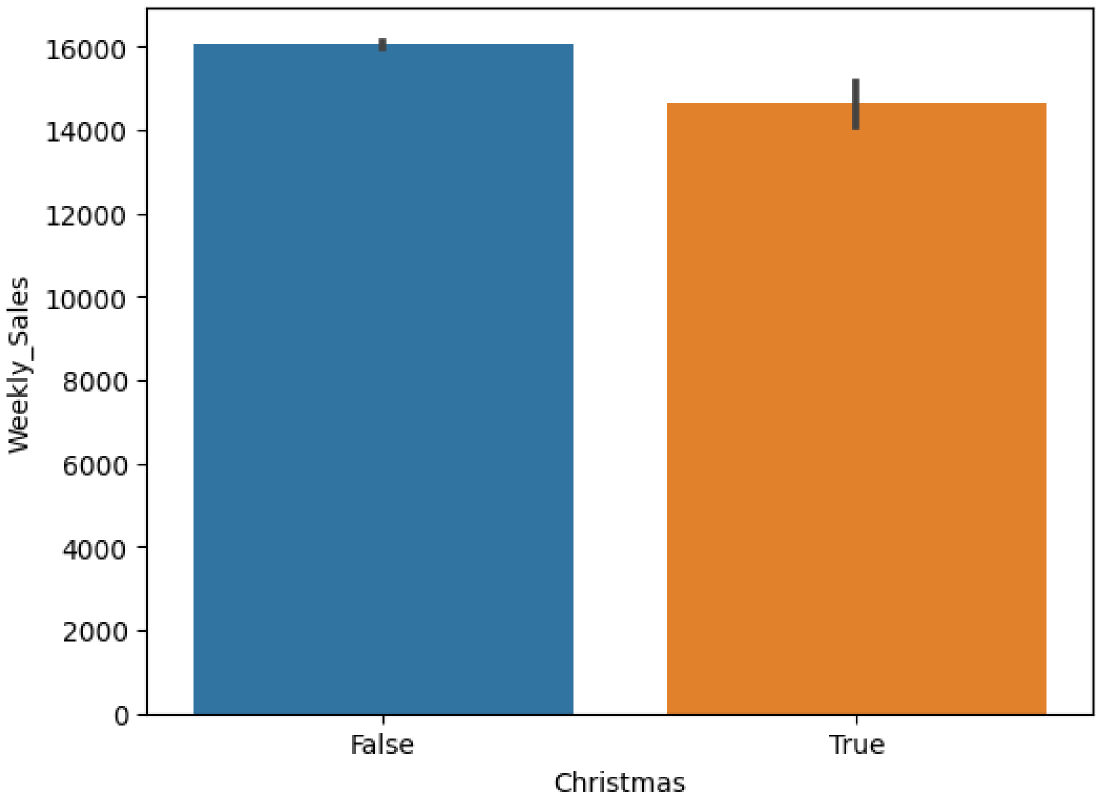
[70]:
```
sns.barplot(x = 'Thanksgiving', y = 'Weekly_Sales', data = df)
```

[70]: <Axes: xlabel='Thanksgiving', ylabel='Weekly_Sales'>



[72]:
```
sns.barplot(x = 'Christmas', y = 'Weekly_Sales', data = df)
```

[72]: <Axes: xlabel='Christmas', ylabel='Weekly_Sales'>



Here we see that the sales on Christmas and Labor Day does not increase the average sales. Super Bowl and Thanksgiving have an increase in the average sales. Thanksgiving has an increase because of the Black Friday sales. People buy christmas gift a week or two in advance, during the Thanksgiving sale.

[75]:
```
df.groupby(['Super_Bowl', 'Type'])['Weekly_Sales'].mean()
```

[75]:
```
Super_Bowl  Type
False       A       20144.507137
            B       12289.697797
            C        9540.026119
True        A       20401.250063
            B       12350.174708
            C       10239.943409
Name: Weekly_Sales, dtype: float64
```

[77]:
```
df.groupby(['Labor_Day', 'Type'])['Weekly_Sales'].mean()
```

[77]:
```
Labor_Day  Type
False      A       20149.353858
           B       12293.264327
           C        9542.417249
```

```
True        A        20060.598111
            B        12098.648882
            C        10045.474040
Name: Weekly_Sales, dtype: float64
```

[79]:
```
df.groupby(['Thanksgiving', 'Type'])['Weekly_Sales'].mean()
```

[79]:
```
Thanksgiving  Type
False         A       20044.007801
              B       12197.717405
              C        9547.377807
True          A       27397.776346
              B       18733.973971
              C        9696.566616
Name: Weekly_Sales, dtype: float64
```

[81]:
```
df.groupby(['Christmas', 'Type'])['Weekly_Sales'].mean()
```

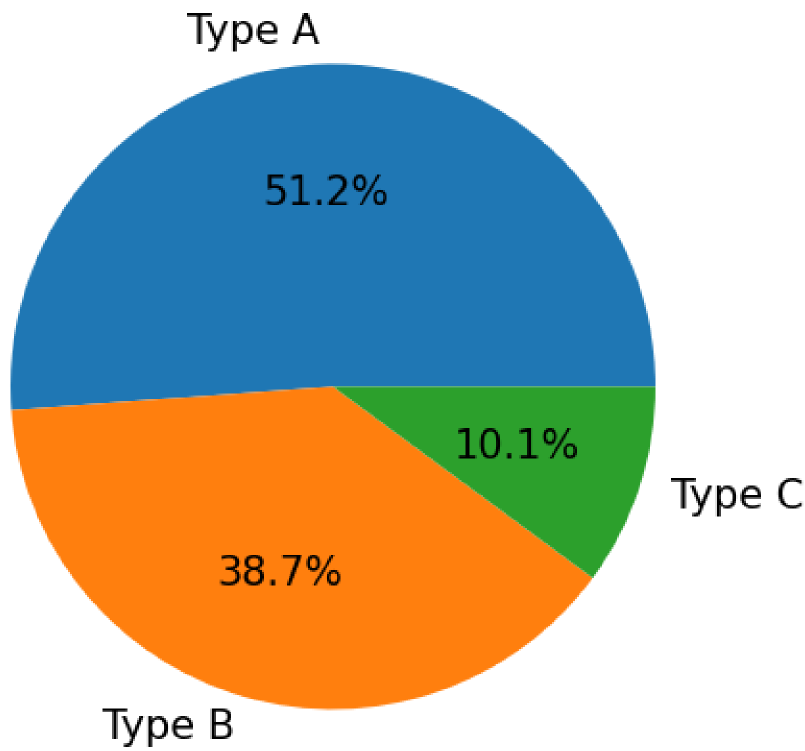
[81]:
```
Christmas  Type
False      A       20174.350209
           B       12301.986116
           C        9570.951973
True       A       18310.167535
           B       11488.988057
           C        8031.520607
Name: Weekly_Sales, dtype: float64
```

[83]:
```
store_counts = df['Type'].value_counts()  # Count occurrences of each store type
total_stores = store_counts.sum()  # Total number of stores

# Calculate percentages
store_percentages = (store_counts / total_stores) * 100

print(store_percentages)
```

```
Type
A    51.155369
B    38.739255
C    10.105375
Name: count, dtype: float64
```

[85]:
```
my_data = store_percentages  #percentages
my_labels = 'Type A','Type B', 'Type C' # labels
plt.pie(my_data,labels=my_labels,autopct='%1.1f%%', textprops={'fontsize': 15})␣
  ↪#plot pie type and bigger the labels
plt.axis('equal')
mpl.rcParams.update({'font.size': 20}) #bigger percentage labels
```

```
plt.show()
```



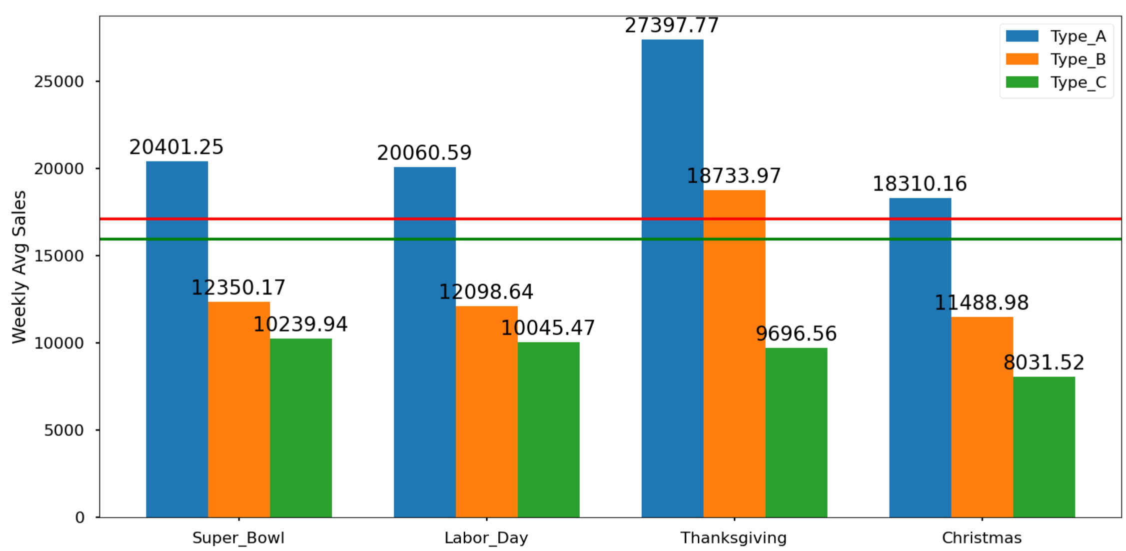
More than half of the stores belongs to Type A.

[88]:
```
df.groupby('IsHoliday')['Weekly_Sales'].mean()
```

[88]:
```
IsHoliday
False    15952.816352
True     17094.300918
Name: Weekly_Sales, dtype: float64
```

[96]:
```
# Plotting avg wekkly sales according to holidays by types
plt.style.use('seaborn-v0_8-poster')
labels = ['Super_Bowl', 'Labor_Day', 'Thanksgiving', 'Christmas']
A_means = [20401.25, 20060.59, 27397.77, 18310.16]
B_means = [12350.17, 12098.64, 18733.97, 11488.98]
C_means = [10239.94,10045.47,9696.56,8031.52]

x = np.arange(len(labels))  # the label locations
width = 0.25  # the width of the bars

fig, ax = plt.subplots(figsize=(16, 8))
rects1 = ax.bar(x - width, A_means, width, label='Type_A')
```

```python
rects2 = ax.bar(x , B_means, width, label='Type_B')
rects3 = ax.bar(x + width, C_means, width, label='Type_C')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Weekly Avg Sales')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()


def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')


autolabel(rects1)
autolabel(rects2)
autolabel(rects3)

plt.axhline(y=17094.30,color='r') # holidays avg
plt.axhline(y=15952.81,color='green') # not-holiday avg

fig.tight_layout()

plt.show()
```



The highest sales were in Thanksgiving. The highest sales among all the stores were from the Type A stores.

[99]:
```
df.sort_values(by='Weekly_Sales',ascending=False).head(5)
```

[99]:
```
        Store  Dept        Date  Weekly_Sales  IsHoliday  Temperature  \
90645      10    72  2010-11-26     693099.36       True        55.33
337053     35    72  2011-11-25     649770.18       True        47.88
94393      10    72  2011-11-25     630999.19       True        60.68
333594     35    72  2010-11-26     627962.93       True        46.67
131088     14    72  2010-11-26     474330.10       True        46.15

        Fuel_Price  MarkDown1  MarkDown2  MarkDown3  MarkDown4  MarkDown5  \
90645        3.162        NaN        NaN        NaN        NaN        NaN
337053       3.492    1333.24        NaN   58563.24      20.97    6386.86
94393        3.760     174.72      329.0  141630.61      79.00    1009.98
333594       3.039        NaN        NaN        NaN        NaN        NaN
131088       3.039        NaN        NaN        NaN        NaN        NaN

               CPI  Unemployment Type    Size  Super_Bowl  Labor_Day  \
90645   126.669267         9.003    B  126512       False      False
337053  140.421786         8.745    B  103681       False      False
94393   129.836400         7.874    B  126512       False      False
333594  136.689571         8.763    B  103681       False      False
131088  182.783277         8.724    A  200898       False      False

        Thanksgiving  Christmas
90645           True      False
337053          True      False
94393           True      False
333594          True      False
131088          True      False
```

All the top 5 sales were from Thannksgiving holidays.

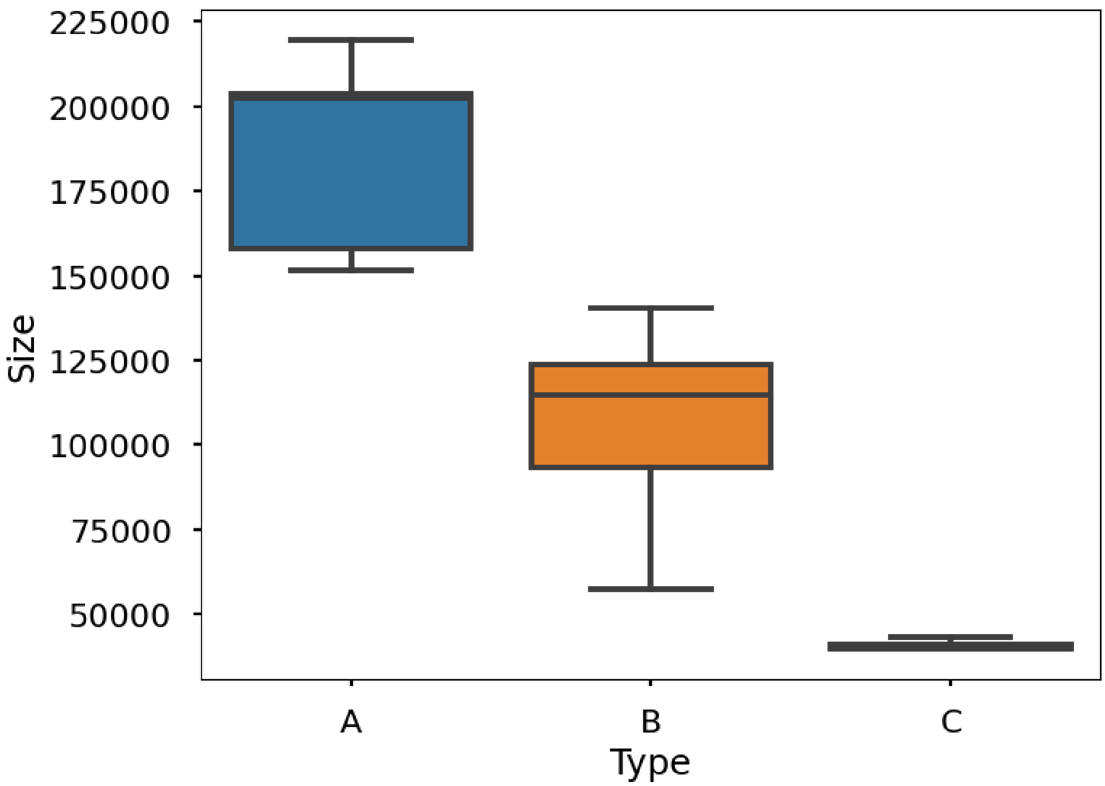
SIZE_TYPE RELATION

[104]:
```
df_store.groupby('Type').describe()['Size'].round(2)
```

[104]:
```
      count       mean       std      min        25%       50%       75%  \
Type
A      22.0  177247.73  49392.62  39690.0  155840.75  202406.0  203819.0
B      17.0  101190.71  32371.14  34875.0   93188.00  114533.0  123737.0
C       6.0   40541.67   1304.15  39690.0   39745.00   39910.0   40774.0

           max
```

```
Type
A     219622.0
B     140167.0
C      42988.0
```

[108]:
```python
plt.figure(figsize=(8,6)) # To see the type-size relation
fig = sns.boxplot(x='Type', y='Size', data=df, showfliers=False)
```



Here we can see that, higher the number of stores, higher is the sales. The smallest size of Type A store, the Type B store begins and same for the Type C store. The smallest size of Type B store, the Type C store starts.

MARKDOWN SALES

[112]:
```python
df.isna().sum()
```

[112]:
```
Store                    0
Dept                     0
Date                     0
Weekly_Sales             0
IsHoliday                0
Temperature              0
Fuel_Price               0
```

```
MarkDown1        270031
MarkDown2        309308
MarkDown3        283561
MarkDown4        285694
MarkDown5        269283
CPI                   0
Unemployment          0
Type                  0
Size                  0
Super_Bowl            0
Labor_Day             0
Thanksgiving          0
Christmas             0
dtype: int64
```

[114]:
```
df = df.fillna(0)
```

[116]:
```
df.isna().sum()
```

[116]:
```
Store           0
Dept            0
Date            0
Weekly_Sales    0
IsHoliday       0
Temperature     0
Fuel_Price      0
MarkDown1       0
MarkDown2       0
MarkDown3       0
MarkDown4       0
MarkDown5       0
CPI             0
Unemployment    0
Type            0
Size            0
Super_Bowl      0
Labor_Day       0
Thanksgiving    0
Christmas       0
dtype: int64
```

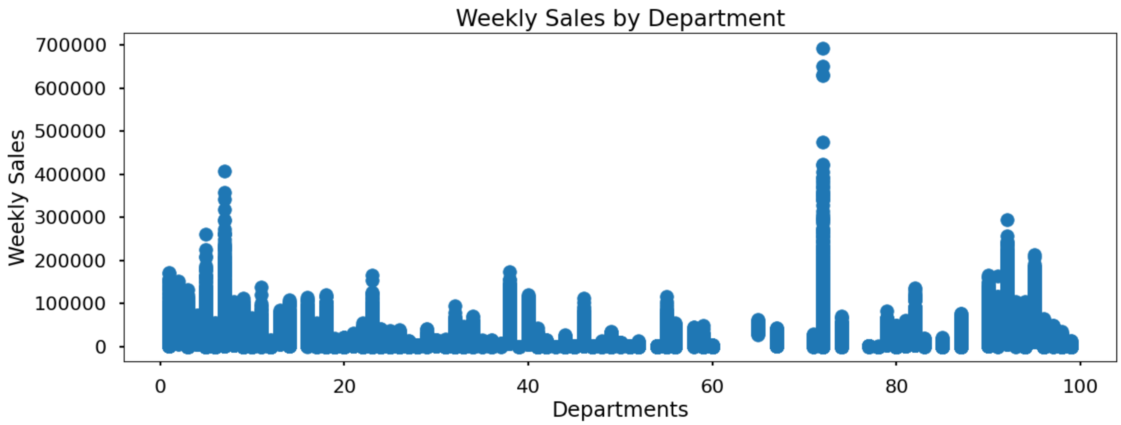
[118]:
```
df.describe()
```

[118]:
```
               Store           Dept   Weekly_Sales    Temperature  \
count  420212.000000  420212.000000  420212.000000  420212.000000
mean       22.195611      44.241309   16033.114591      60.090599
std        12.787236      30.508819   22729.492116      18.447857
```

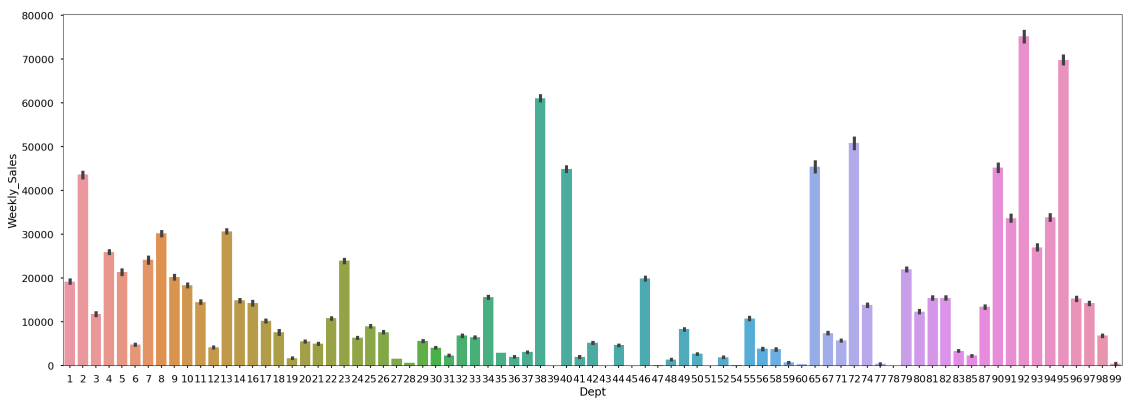
```
min         1.000000       1.000000       0.010000      -2.060000
25%        11.000000      18.000000    2120.130000      46.680000
50%        22.000000      37.000000    7661.700000      62.090000
75%        33.000000      74.000000   20271.265000      74.280000
max        45.000000      99.000000  693099.360000     100.140000

          Fuel_Price      MarkDown1      MarkDown2      MarkDown3  \
count  420212.000000  420212.000000  420212.000000  420212.000000
mean        3.360890    2590.323565     878.905242     468.845949
std         0.458519    6053.415601    5076.928566    5534.069859
min         2.472000       0.000000    -265.760000     -29.100000
25%         2.933000       0.000000       0.000000       0.000000
50%         3.452000       0.000000       0.000000       0.000000
75%         3.738000    2809.050000       2.400000       4.540000
max         4.468000   88646.760000  104519.540000  141630.610000

           MarkDown4      MarkDown5            CPI   Unemployment  \
count  420212.000000  420212.000000  420212.000000  420212.000000
mean     1083.534361    1662.805002     171.212496       7.960000
std      3896.068938    4206.209357      39.162445       1.863879
min         0.000000       0.000000     126.064000       3.879000
25%         0.000000       0.000000     132.022667       6.891000
50%         0.000000       0.000000     182.350989       7.866000
75%       425.290000    2168.040000     212.445487       8.567000
max     67474.850000  108519.280000     227.232807      14.313000

                Size
count  420212.000000
mean   136749.732787
std     60993.084568
min     34875.000000
25%     93638.000000
50%    140167.000000
75%    202505.000000
max    219622.000000
```

[120]:

```python
x = df['Dept']
y = df['Weekly_Sales']
plt.figure(figsize=(15,5))
plt.title('Weekly Sales by Department')
plt.xlabel('Departments')
plt.ylabel('Weekly Sales')
plt.scatter(x,y)
plt.show()
```

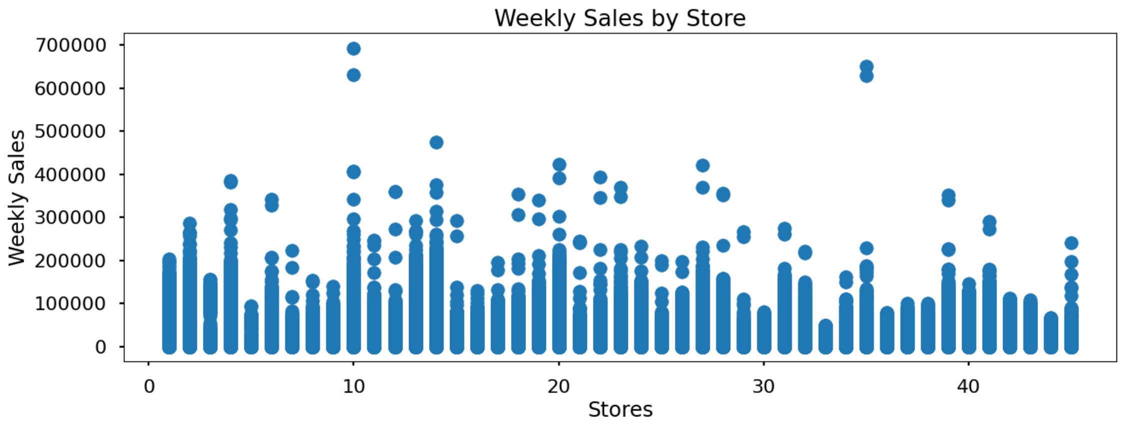


[122]:
```
plt.figure(figsize=(30,10))
fig = sns.barplot(x='Dept', y='Weekly_Sales', data=df)
```

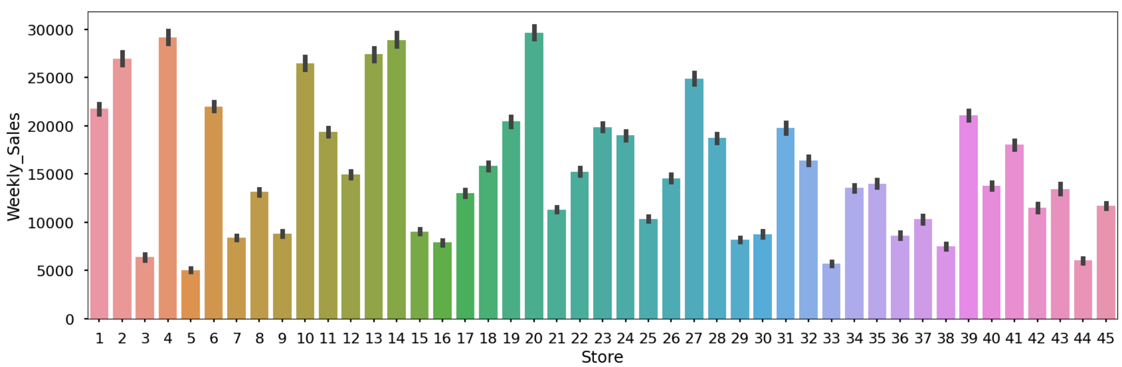


From graph 1, we see that one department between 60-80 has the higher sales values. This might be a seasonal department as when we check the graph 2 (average sales), we see that department 92 has higher weekly sales.

[125]:
```
x = df['Store']
y = df['Weekly_Sales']
plt.figure(figsize=(15,5))
plt.title('Weekly Sales by Store')
plt.xlabel('Stores')
plt.ylabel('Weekly Sales')
plt.scatter(x,y)
plt.show()
```



[127]:
```
plt.figure(figsize=(20,6))
fig = sns.barplot(x='Store', y='Weekly_Sales', data=df)
```



In the above two graphs, from graph 1, we see that store 10 has the higher value sales and from graph 2 we see that store 4 and store 20 have higher average sales. Store 20, 4 have best sales followed by store 14.

[134]:
```
df["Date"] = pd.to_datetime(df["Date"])  # convert to datetime
df['week'] = df['Date'].dt.isocalendar().week  # get ISO week
df['month'] = df['Date'].dt.month  # get month
df['year'] = df['Date'].dt.year  # get year
```

[136]:
```
df
```

[136]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	\
0	1	1	2010-02-05	24924.50	False	42.31	
1	1	2	2010-02-05	50605.27	False	42.31	
2	1	3	2010-02-05	13740.12	False	42.31	
3	1	4	2010-02-05	39954.04	False	42.31	

```
4            1      5 2010-02-05      32229.38      False        42.31
…         …    …          …          …           …           …
421565      45     93 2012-10-26       2487.80      False        58.85
421566      45     94 2012-10-26       5203.31      False        58.85
421567      45     95 2012-10-26      56017.47      False        58.85
421568      45     97 2012-10-26       6817.48      False        58.85
421569      45     98 2012-10-26       1076.80      False        58.85

        Fuel_Price  MarkDown1  MarkDown2  MarkDown3  …  Unemployment  Type  \
0            2.572       0.00       0.00        0.0  …         8.106     A
1            2.572       0.00       0.00        0.0  …         8.106     A
2            2.572       0.00       0.00        0.0  …         8.106     A
3            2.572       0.00       0.00        0.0  …         8.106     A
4            2.572       0.00       0.00        0.0  …         8.106     A
…              …         …          …          …  …            …    …
421565       3.882    4018.91      58.08      100.0  …         8.667     B
421566       3.882    4018.91      58.08      100.0  …         8.667     B
421567       3.882    4018.91      58.08      100.0  …         8.667     B
421568       3.882    4018.91      58.08      100.0  …         8.667     B
421569       3.882    4018.91      58.08      100.0  …         8.667     B

          Size  Super_Bowl Labor_Day  Thanksgiving  Christmas  week  month  \
0       151315       False     False         False      False     5      2
1       151315       False     False         False      False     5      2
2       151315       False     False         False      False     5      2
3       151315       False     False         False      False     5      2
4       151315       False     False         False      False     5      2
…          …          …        …            …          …    …     …
421565  118221       False     False         False      False    43     10
421566  118221       False     False         False      False    43     10
421567  118221       False     False         False      False    43     10
421568  118221       False     False         False      False    43     10
421569  118221       False     False         False      False    43     10

        year
0       2010
1       2010
2       2010
3       2010
4       2010
…      …
421565  2012
421566  2012
421567  2012
421568  2012
421569  2012
```

```
[420212 rows x 23 columns]
```

[138]:
```
df.groupby('month')['Weekly_Sales'].mean()
```

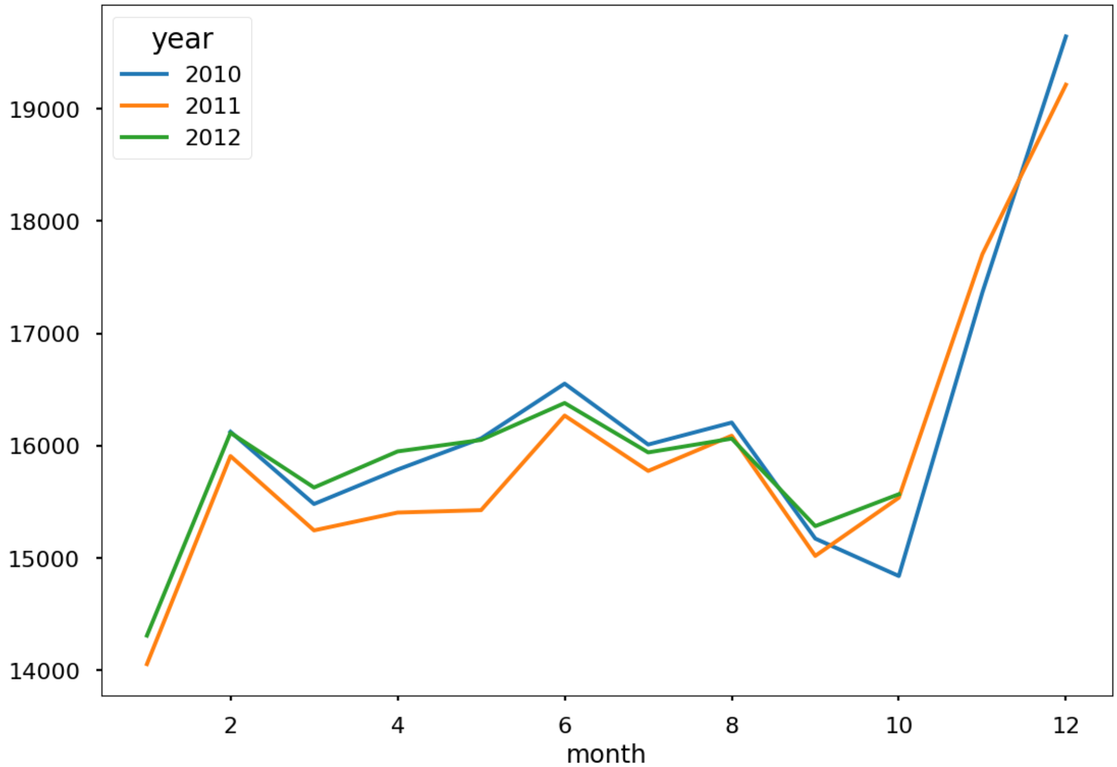
[138]:
```
month
1     14182.239153
2     16048.701191
3     15464.817698
4     15696.435193
5     15845.556200
6     16397.605478
7     15905.472425
8     16113.800069
9     15147.216063
10    15279.182119
11    17534.964277
12    19425.798603
Name: Weekly_Sales, dtype: float64
```

[140]:
```
df.groupby('year')['Weekly_Sales'].mean()
```

[140]:
```
year
2010    16318.648285
2011    16007.797985
2012    15748.265005
Name: Weekly_Sales, dtype: float64
```

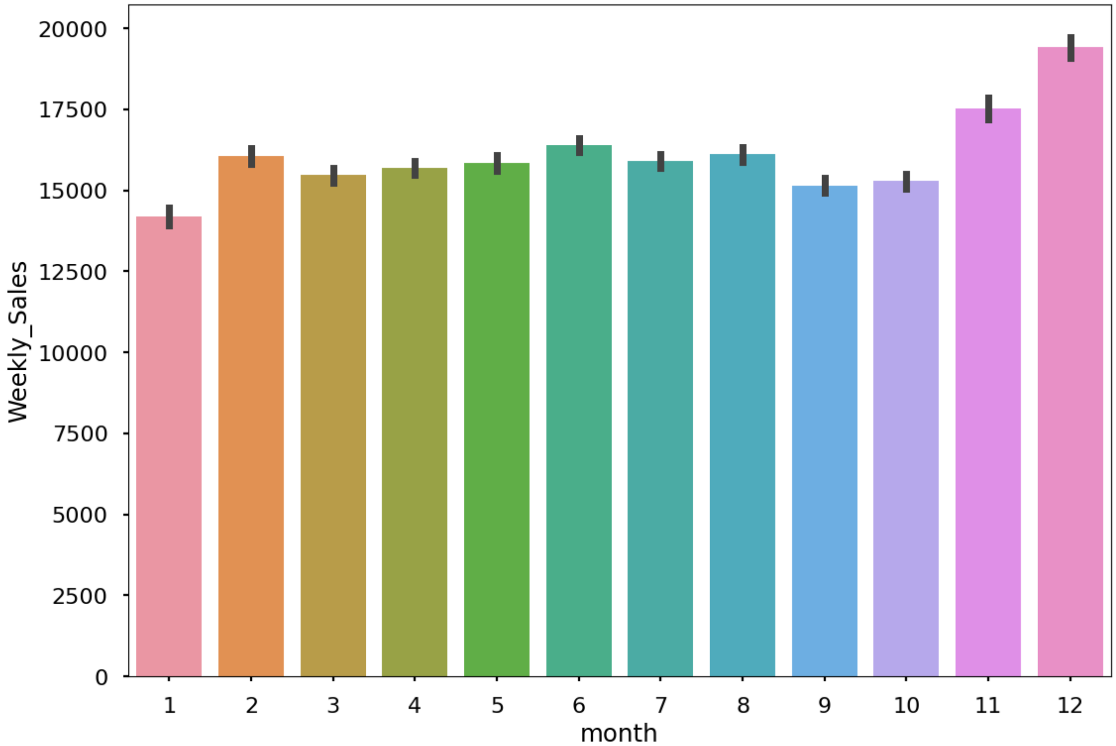
[142]:
```
monthly_sales = pd.pivot_table(df, values = 'Weekly_Sales', columns = 'year',␣
 ↪index = 'month')
monthly_sales.plot()
```

[142]:
```
<Axes: xlabel='month'>
```



In general, 2011 has lesser sales than 2012. Every year, November and December had the highest sale. Despite of 2012 has no last two months sales, it's mean is near to 2010. Most probably, it will take the first place if we get 2012 results and add them.

[145]:
```
fig = sns.barplot(x='month', y='Weekly_Sales', data=df)
```



When we look at the graph above, the best sales are in December and November, as expected. The highest values are belongs to Thankgiving holiday but when we take average it is obvious that December has the best value.

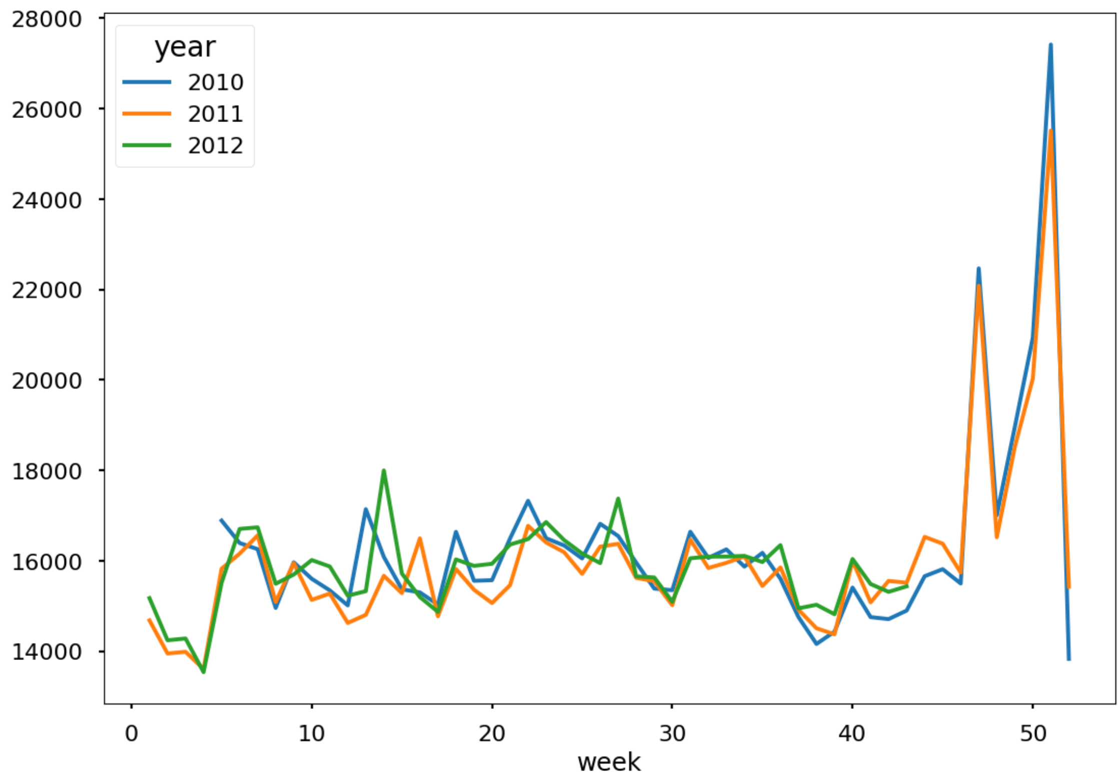
[148]:
```
df.groupby('week')['Weekly_Sales'].mean().sort_values(ascending=False).head()
```

[148]:
```
week
51    26454.164116
47    22269.601768
50    20478.421134
49    18731.794840
22    16856.650245
Name: Weekly_Sales, dtype: float64
```

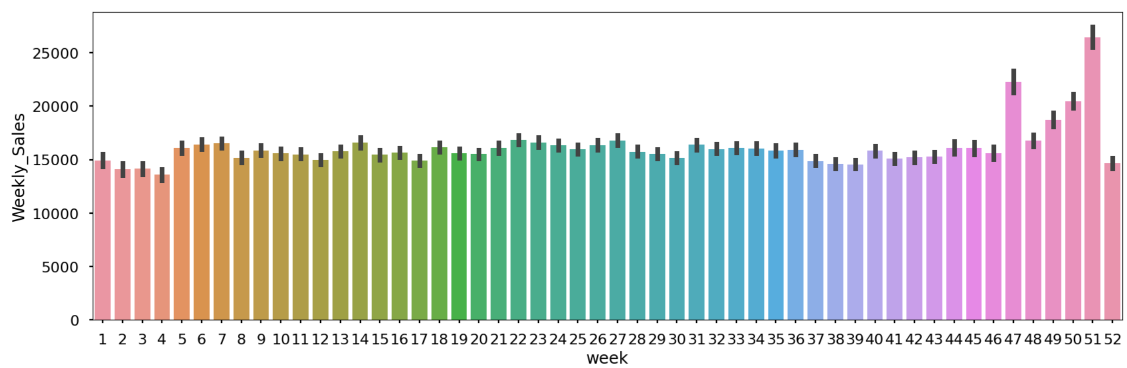
Top 5 sales averages by weekly belongs to 1-2 weeks before Christmas, Thanksgiving, Black Friday and end of May, when the schools are closed.

[151]:
```
weekly_sales = pd.pivot_table(df, values = "Weekly_Sales", columns = "year",␣
 ↪index = "week")
weekly_sales.plot()
```

[151]:
```
<Axes: xlabel='week'>
```




```
[153]: plt.figure(figsize=(20,6))
       fig = sns.barplot(x='week', y='Weekly_Sales', data=df)
```

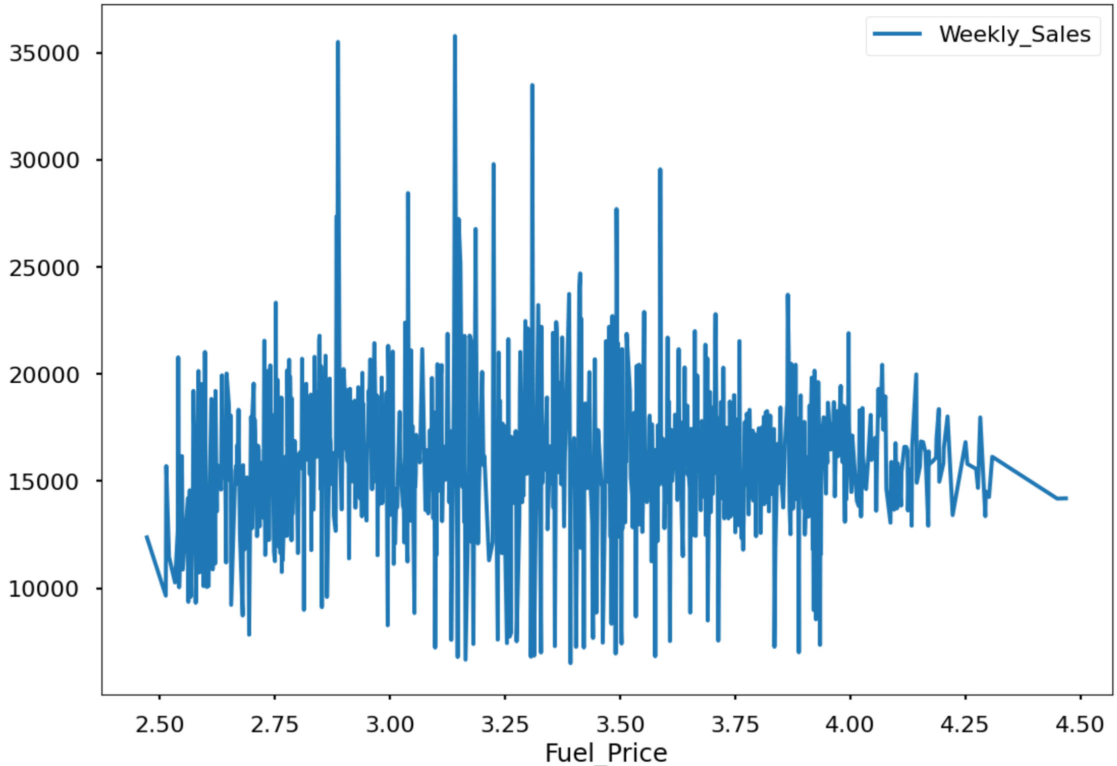


The best sales were in week 51 and week 47. These are the weeks before christmas and Thanksgiving/Black Friday.

Fuel Price, CPI , Unemployment , Temperature Effects

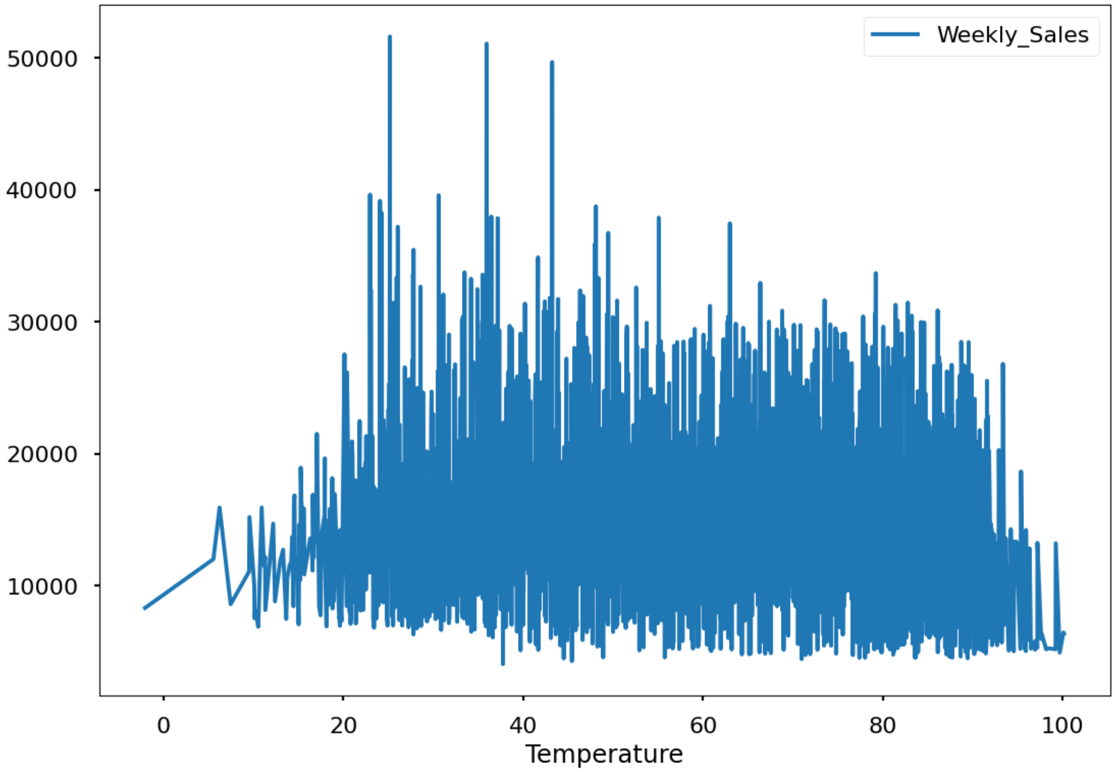
```
[158]: fuel_price = pd.pivot_table(df, values = "Weekly_Sales", index= "Fuel_Price")
       fuel_price.plot()
```

[158]: <Axes: xlabel='Fuel_Price'>



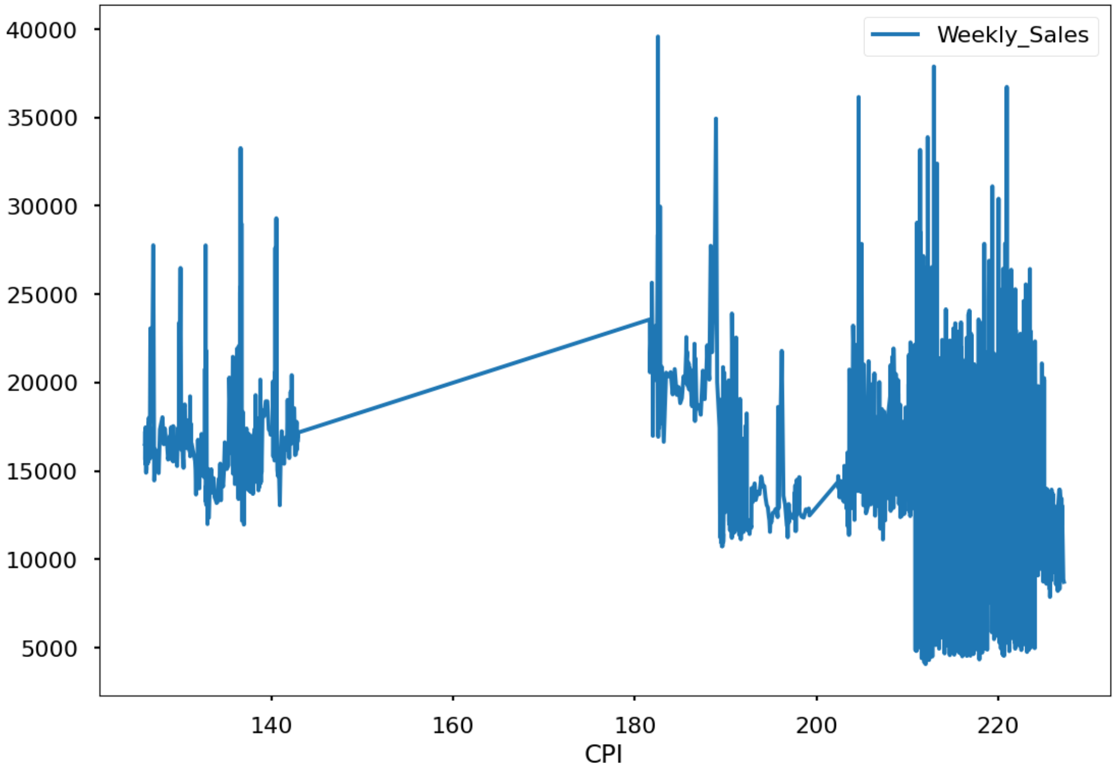
[160]:
```
temp = pd.pivot_table(df, values = "Weekly_Sales", index= "Temperature")
temp.plot()
```

[160]: <Axes: xlabel='Temperature'>



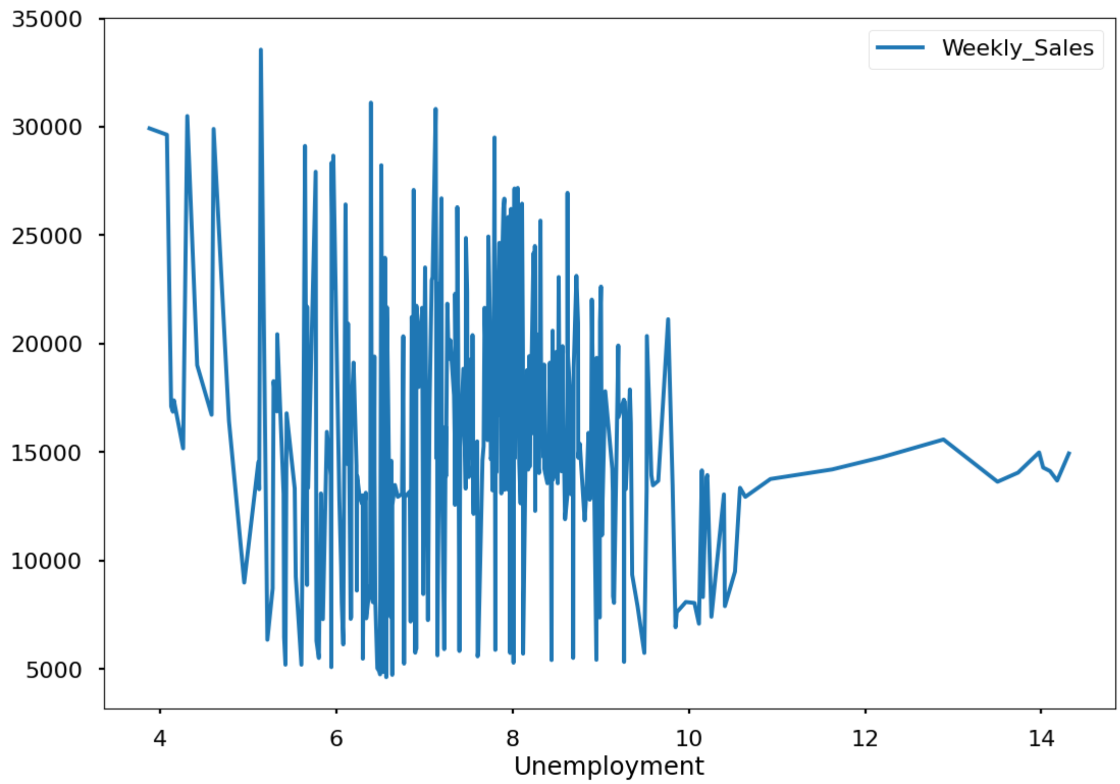
[162]:
```
CPI = pd.pivot_table(df, values = "Weekly_Sales", index= "CPI")
CPI.plot()
```

[162]: <Axes: xlabel='CPI'>



[164]:
```
unemployment = pd.pivot_table(df, values = "Weekly_Sales", index=␣
 ↪"Unemployment")
unemployment.plot()
```

[164]: <Axes: xlabel='Unemployment'>



From graphs, it is seen that there are no significant patterns between CPI, temperature, unemployment rate, fuel price vs weekly sales. There is no data for CPI between 140-180 also.

[167]:
```
df.to_csv('clean_data.csv')
```

- There are 45 stores and 81 department in data. Departments are not same in all stores. Although department 72 has higher weekly sales values, on average department 92 is the best. It shows us, some departments has higher values as seasonal like Thanksgiving. It is consistant when we look at the top 5 sales in data, all of them belongs to 72th department at Thanksgiving holiday time.
- Although stores 10 and 35 have higher weekly sales values sometimes, in general average store 20 and store 4 are on the first and second rank. It means that some areas has higher seasonal sales.
- Stores has 3 types as A, B and C according to their sizes. Almost half of the stores are bigger than 150000 and categorized as A. According to type, sales of the stores are changing.
- As expected, holiday average sales are higher than normal dates.
- Christmas holiday introduces as the last days of the year. But people generally shop at 51th week. So, when we look at the total sales of holidays, Thankgiving has higher sales between them which was assigned by Walmart.
- Year 2010 has higher sales than 2011 and 2012. But, November and December sales are not in the data for 2012. Even without highest sale months, 2012 is not significantly less than 2010, so after adding last two months, it can be first.
- It is obviously seen that week 51 and 47 have higher values and 50-48 weeks follow them.

Interestingly, 5th top sales belongs to 22th week of the year. This results show that Christmas, Thankgiving and Black Friday are very important than other weeks for sales and 5th important time is 22th week of the year and it is end of the May, when schools are closed. Most probably, people are preparing for holiday at the end of the May.

- January sales are significantly less than other months. This is the result of November and December high sales. After two high sales month, people prefer to pay less on January.
- CPI, temperature, unemployment rate and fuel price have no pattern on weekly sales.

Generally, Rondom Forest Regressor gives good results when we tune it well. So, to find simple baseline model, I will use RandomForestRegressor in this notebook.

[179]:
```
pd.options.display.max_columns=100 # to see columns
```

[181]:
```
df = pd.read_csv('C:/Users/aryan/Desktop/clean_data.csv')
```

[183]:
```
df.drop(columns=['Unnamed: 0'],inplace=True)
```

[185]:
```
df['Date'] = pd.to_datetime(df['Date']) # changing datetime to divide if needs
```

[187]:
```
df
```

[187]:
```
        Store  Dept       Date  Weekly_Sales  IsHoliday  Temperature  \
0           1     1 2010-02-05      24924.50      False        42.31
1           1     2 2010-02-05      50605.27      False        42.31
2           1     3 2010-02-05      13740.12      False        42.31
3           1     4 2010-02-05      39954.04      False        42.31
4           1     5 2010-02-05      32229.38      False        42.31
…         …    …        …           …          …            …
420207     45    93 2012-10-26       2487.80      False        58.85
420208     45    94 2012-10-26       5203.31      False        58.85
420209     45    95 2012-10-26      56017.47      False        58.85
420210     45    97 2012-10-26       6817.48      False        58.85
420211     45    98 2012-10-26       1076.80      False        58.85

        Fuel_Price  MarkDown1  MarkDown2  MarkDown3  MarkDown4  MarkDown5  \
0            2.572       0.00       0.00        0.0       0.00       0.00
1            2.572       0.00       0.00        0.0       0.00       0.00
2            2.572       0.00       0.00        0.0       0.00       0.00
3            2.572       0.00       0.00        0.0       0.00       0.00
4            2.572       0.00       0.00        0.0       0.00       0.00
…              …          …          …          …          …          …
420207       3.882    4018.91      58.08      100.0     211.94     858.33
420208       3.882    4018.91      58.08      100.0     211.94     858.33
420209       3.882    4018.91      58.08      100.0     211.94     858.33
420210       3.882    4018.91      58.08      100.0     211.94     858.33
420211       3.882    4018.91      58.08      100.0     211.94     858.33

               CPI  Unemployment Type    Size  Super_Bowl  Labor_Day  \
```

```
0       211.096358        8.106    A  151315       False      False
1       211.096358        8.106    A  151315       False      False
2       211.096358        8.106    A  151315       False      False
3       211.096358        8.106    A  151315       False      False
4       211.096358        8.106    A  151315       False      False
…            …          …   …      …          …      …
420207  192.308899        8.667    B  118221       False      False
420208  192.308899        8.667    B  118221       False      False
420209  192.308899        8.667    B  118221       False      False
420210  192.308899        8.667    B  118221       False      False
420211  192.308899        8.667    B  118221       False      False

        Thanksgiving  Christmas  week  month  year
0              False      False     5      2  2010
1              False      False     5      2  2010
2              False      False     5      2  2010
3              False      False     5      2  2010
4              False      False     5      2  2010
…              …          …   …      …   …
420207         False      False    43     10  2012
420208         False      False    43     10  2012
420209         False      False    43     10  2012
420210         False      False    43     10  2012
420211         False      False    43     10  2012

[420212 rows x 23 columns]
```

For preprocessing our data, I will change holidays boolean values to 0-1 and replace type of the stores from A, B, C to 1, 2, 3.

[190]:
```
df_encoded = df.copy()
```

[192]:
```
type_group = {'A':1, 'B': 2, 'C': 3}  # changing A,B,C to 1-2-3
df_encoded['Type'] = df_encoded['Type'].replace(type_group)
```

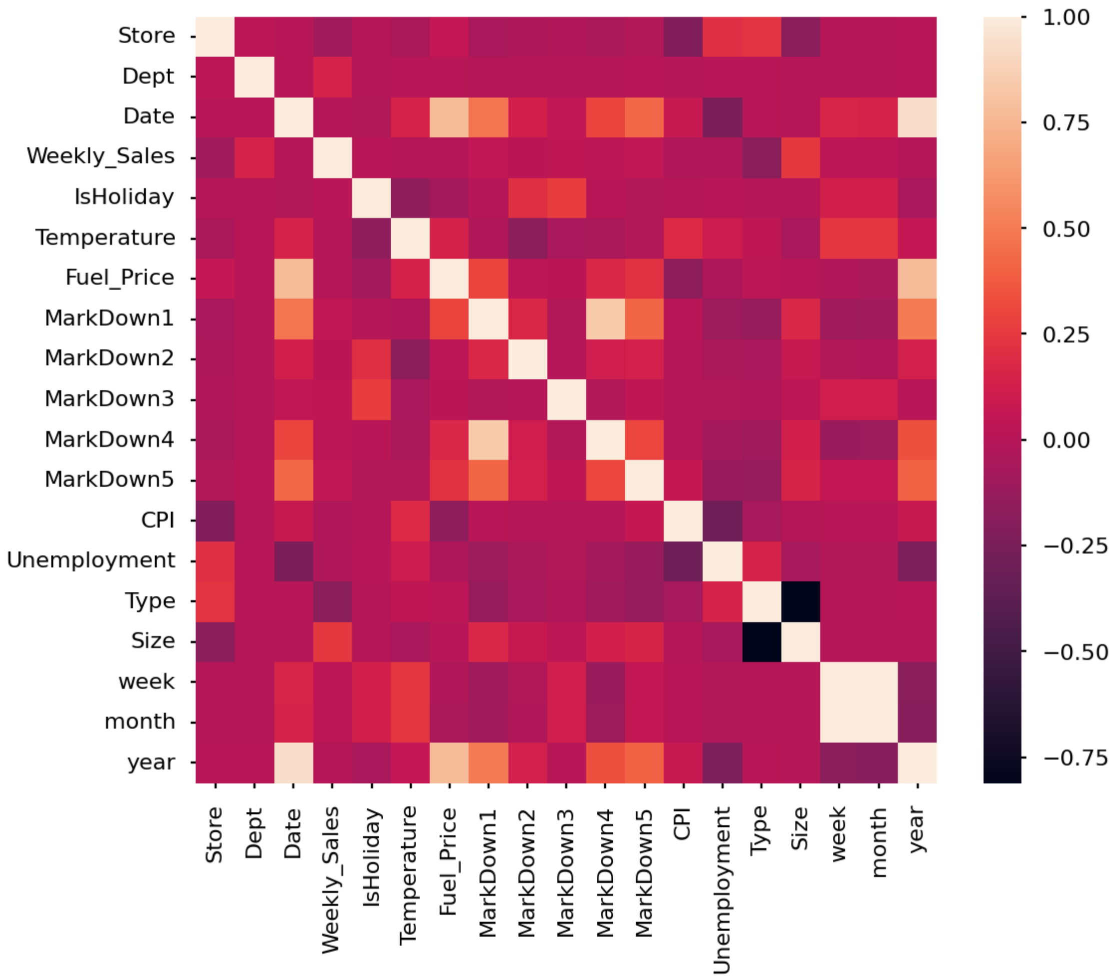
[194]:
```
df_encoded['Super_Bowl'] = df_encoded['Super_Bowl'].astype(bool).astype(int) #␣
 ↪changing T,F to 0-1
df_encoded['Labor_Day'] = df_encoded['Labor_Day'].astype(bool).astype(int) #␣
 ↪changing T,F to 0-1
df_encoded['Thanksgiving'] = df_encoded['Thanksgiving'].astype(bool).
 ↪astype(int) # changing T,F to 0-1
df_encoded['Christmas'] = df_encoded['Christmas'].astype(bool).astype(int) #␣
 ↪changing T,F to 0-1
df_encoded['IsHoliday'] = df_encoded['IsHoliday'].astype(bool).astype(int) #␣
 ↪changing T,F to 0-1
```

[196]:
```
df_new = df_encoded.copy() # taking the copy of encoded df to keep it original
```

Firstly, I will drop the holiday columns (Christmas, Thanksgiving, Labor_Day and Super_Bowl) from my data and try without them. To keep my encoded data safe, I assigned my dataframe to new one and I will use for this.

[199]:
```
drop_column = ['Super_Bowl', 'Labor_Day', 'Thanksgiving', 'Christmas']
df_new.drop(drop_column, axis = 1, inplace = True)
```

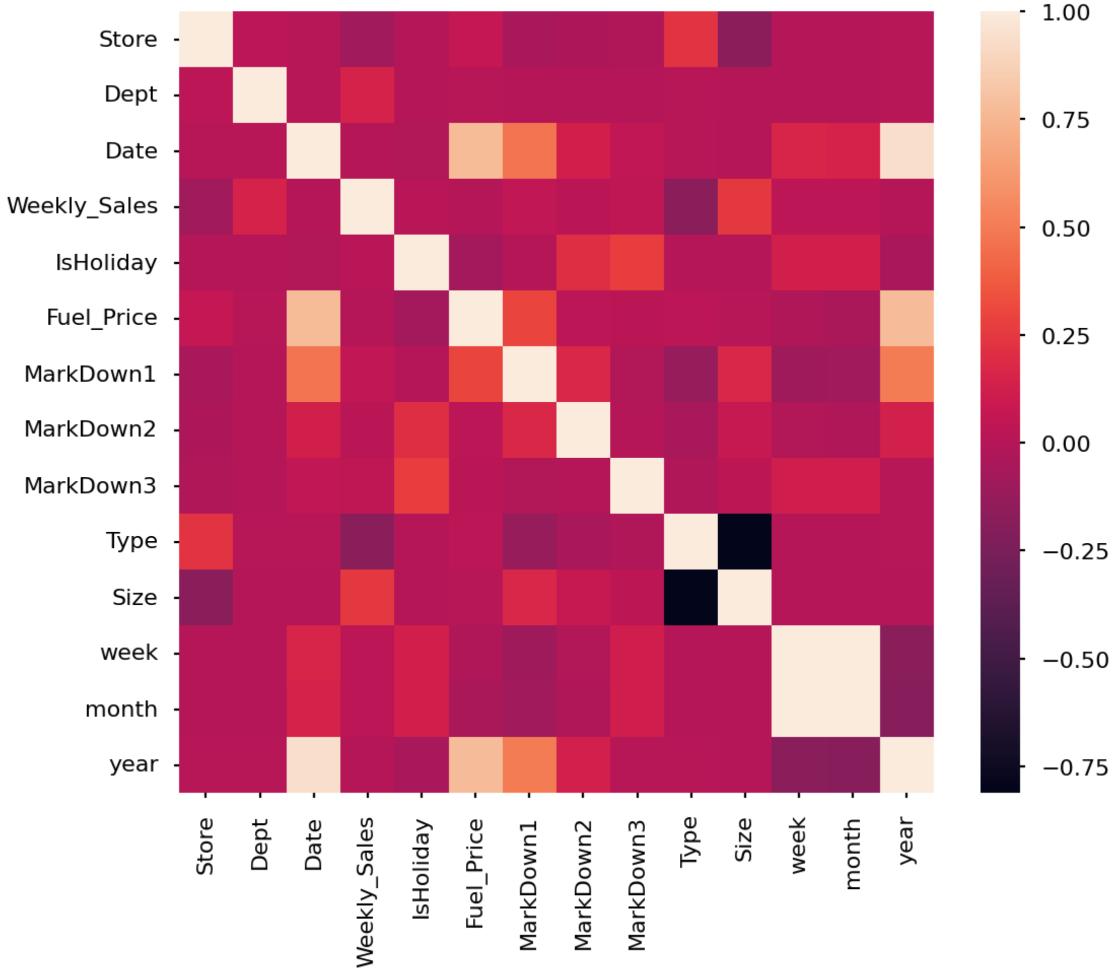
[201]:
```
plt.figure(figsize = (12,10))
sns.heatmap(df_new.corr())
plt.show()
```



Temperature, unemployment, CPI have no significant effect on weekly sales, so I will drop them. Also, Markdown 4 and 5 highly correlated with Markdown 1. So, I will drop them also. It can create multicollinearity problem, maybe. So, first I will try without them.

[204]:
```
drop_column = ['Temperature','MarkDown4','MarkDown5','CPI','Unemployment']
df_new.drop(drop_column, axis=1, inplace=True) # dropping columns
```

[208]:
```
plt.figure(figsize = (12,10))
sns.heatmap(df_new.corr())
plt.show()
```



Size and type are highly correlated with weekly sales. Also, department and store are correlated with sales.

[213]:
```
df_new = df_new.sort_values(by = 'Date', ascending = True)
```

[215]:
```
df_new
```

[215]:
```
        Store  Dept       Date  Weekly_Sales  IsHoliday  Fuel_Price  \
0           1     1 2010-02-05      24924.50          0       2.572
329781     35     3 2010-02-05      14612.19          0       2.784
329782     35     4 2010-02-05      26323.15          0       2.784
329783     35     5 2010-02-05      36414.63          0       2.784
329784     35     6 2010-02-05      11437.81          0       2.784
```

```
…          …   …          …           …          …            …
329722     34    14  2012-10-26       8930.71          0        3.514
329723     34    16  2012-10-26       4841.81          0        3.514
329724     34    17  2012-10-26       7035.13          0        3.514
329726     34    20  2012-10-26       2124.60          0        3.514
420211     45    98  2012-10-26       1076.80          0        3.882

        MarkDown1  MarkDown2  MarkDown3  Type    Size  week  month  year
0            0.00       0.00        0.0     1  151315     5      2  2010
329781       0.00       0.00        0.0     2  103681     5      2  2010
329782       0.00       0.00        0.0     2  103681     5      2  2010
329783       0.00       0.00        0.0     2  103681     5      2  2010
329784       0.00       0.00        0.0     2  103681     5      2  2010
…             …        …        …   …     …    …    …   …
329722    1151.88      68.01        3.0     1  158114    43     10  2012
329723    1151.88      68.01        3.0     1  158114    43     10  2012
329724    1151.88      68.01        3.0     1  158114    43     10  2012
329726    1151.88      68.01        3.0     1  158114    43     10  2012
420211    4018.91      58.08      100.0     2  118221    43     10  2012

[420212 rows x 14 columns]
```

CREATING TRAIN-TEST SPLITS

[220]:
```
train_data = df_new[:int(0.7*(len(df_new)))] # taking train part
test_data = df_new[int(0.7*(len(df_new))):] # taking test part

target = 'Weekly_Sales'
used_cols = [c for c in df_new.columns.to_list() if c not in [target]]

X_train = train_data[used_cols]
X_test = train_data[used_cols]
Y_train = train_data[target]
Y_test = train_data[target]
```

[222]:
```
X = df_new[used_cols]
```

[224]:
```
X_train = X_train.drop(['Date'], axis = 1)
X_test = X_test.drop(['Date'], axis = 1)
```

[230]:
```
#Our metric is not calculated as default from ready models. It is weighed error␣
↪so, I will use function below to calculate it.
def wmae_test(test, pred): # WMAE for test
    weights = X_test['IsHoliday'].apply(lambda is_holiday:5 if is_holiday else␣
↪1)
    error = np.sum(weights * np.abs(test - pred), axis=0) / np.sum(weights)
    return error
```

RANDOM FOREST REGRESSOR

[247]:
```
#To tune the regressor, I choose regressor parameters manually. I changed the␣
↪parameters each time and try to find the best result.

rf = RandomForestRegressor(n_estimators=50, random_state=42, n_jobs=-1,␣
↪max_depth=35,
                           max_features = 'sqrt',min_samples_split = 10)

from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()

#making pipe tp use scaler and regressor together
pipe = make_pipeline(scaler,rf)

pipe.fit(X_train, Y_train)

# predictions on train set
Y_pred = pipe.predict(X_train)

# predictions on test set
Y_pred_test = pipe.predict(X_test)
```

[249]:
```
wmae_test(Y_test, y_pred_test)
```

[249]: 4701.466772365683

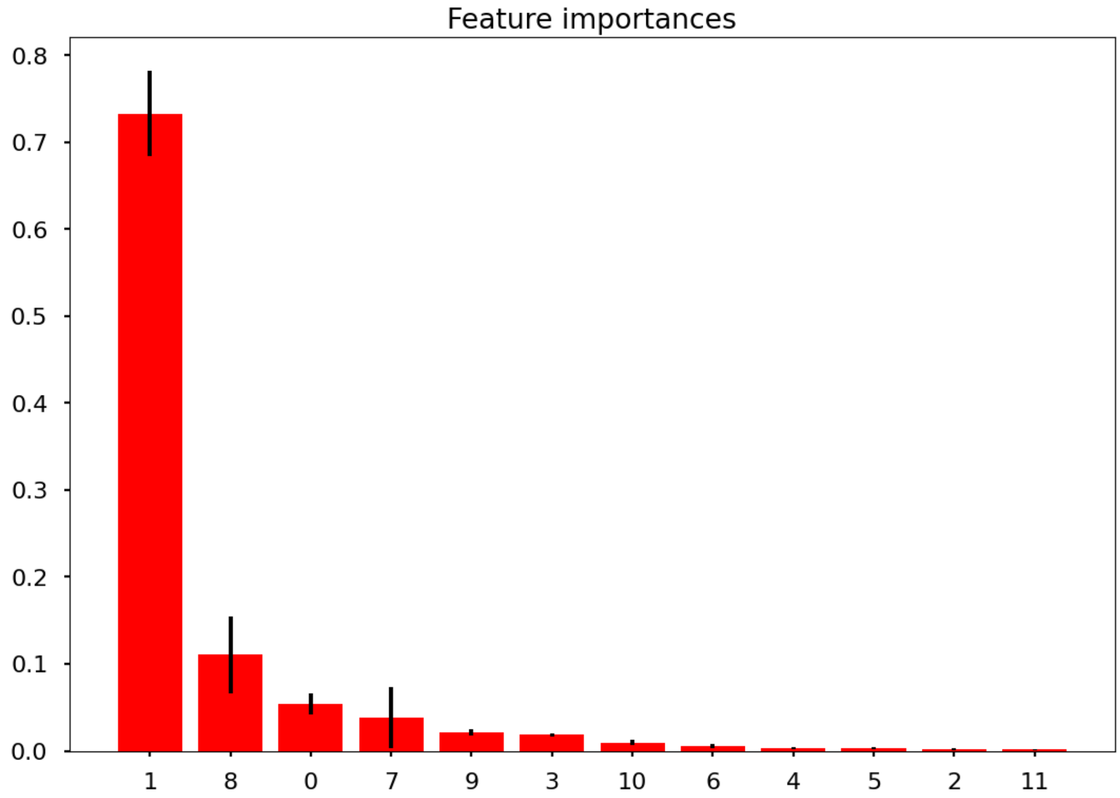
For the first trial, my weighted error is around 4701.

[252]:
```
X = X.drop(['Date'], axis=1)
```

[254]:
```
importances = rf.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Printing the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plotting the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
```

```
plt.xlim([-1, X.shape[1]])
plt.show()
```

```
Feature ranking:
1. feature 1 (0.732822)
2. feature 8 (0.110390)
3. feature 0 (0.054027)
4. feature 7 (0.038210)
5. feature 9 (0.021277)
6. feature 3 (0.018402)
7. feature 10 (0.009446)
8. feature 6 (0.005523)
9. feature 4 (0.003413)
10. feature 5 (0.002776)
11. feature 2 (0.002246)
12. feature 11 (0.001467)
```



[256]:
```
#Dropping month as it is least important feature

X1_train = X_train.drop(['month'], axis=1) # dropping month
X1_test = X_test.drop(['month'], axis=1)
```

[260]:
```
#Running model again without 'month'

rf = RandomForestRegressor(n_estimators=50, random_state=42, n_jobs=-1,␣
↪max_depth=35,
                           max_features = 'sqrt',min_samples_split = 10)

scaler=RobustScaler()
pipe = make_pipeline(scaler,rf)

pipe.fit(X1_train, Y_train)

# predictions on train set
Y_pred = pipe.predict(X1_train)

# predictions on test set
Y_pred_test = pipe.predict(X1_test)
```

[262]:
```
wmae_test(Y_test, Y_pred_test)
```

[262]: 4341.767849846648

Better result than before

[265]:
```
# Now, I want to make sure that my model will learn from the columns which I␣
↪dropped or not. So, I will apply my model to whole encoded data again.

# splitting train-test to whole dataset
train_data_enc = df_encoded[:int(0.7*(len(df_encoded)))]
test_data_enc = df_encoded[int(0.7*(len(df_encoded))):]

target = "Weekly_Sales"
used_cols1 = [c for c in df_encoded.columns.to_list() if c not in [target]] #␣
↪all columns except price

X_train_enc = train_data_enc[used_cols1]
X_test_enc = test_data_enc[used_cols1]
Y_train_enc = train_data_enc[target]
Y_test_enc = test_data_enc[target]
```

[267]:
```
X_enc = df_encoded[used_cols1]
```

[269]:
```
X_enc = X_enc.drop(['Date'], axis=1)
```

[271]:
```
X_train_enc = X_train_enc.drop(['Date'], axis=1) # dropping date from train and␣
↪test
X_test_enc= X_test_enc.drop(['Date'], axis=1)
```

[273]:
```
rf = RandomForestRegressor(n_estimators=50, random_state=42, n_jobs=-1,␣
↪max_depth=35,
                           max_features = 'sqrt',min_samples_split = 10)

scaler=RobustScaler()
pipe = make_pipeline(scaler,rf)

pipe.fit(X_train_enc, Y_train_enc)

# predictions on train set
Y_pred_enc = pipe.predict(X_train_enc)

# predictions on test set
Y_pred_test_enc = pipe.predict(X_test_enc)
```

[275]:
```
wmae_test(Y_test_enc, Y_pred_test_enc)
```

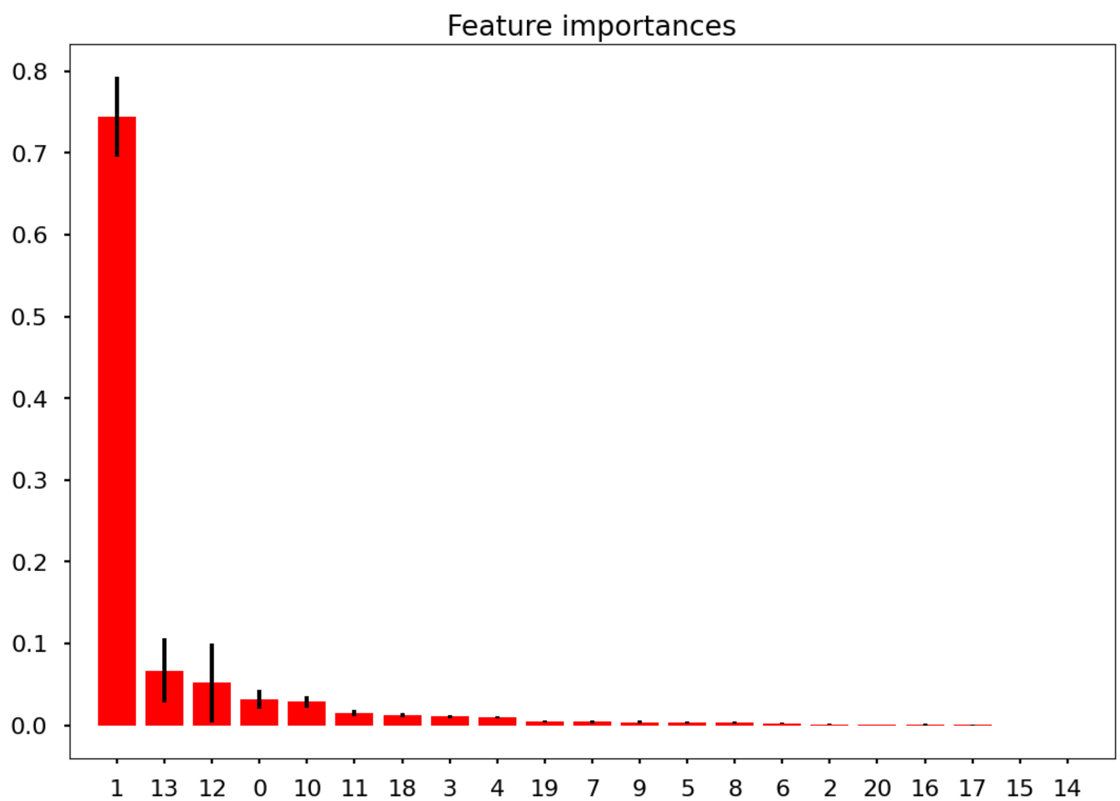
[275]: 2527.8371453807604

We found better results for whole data, it means our model can learn from columns which I dropped before.

[278]:
```
importances = rf.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Printing the feature ranking
print("Feature ranking:")

for f in range(X_enc.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plotting the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X_enc.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X_enc.shape[1]), indices)
plt.xlim([-1, X_enc.shape[1]])
plt.show()
```

```
Feature ranking:
1. feature 1 (0.744048)
2. feature 13 (0.067102)
3. feature 12 (0.052065)
4. feature 0 (0.031637)
5. feature 10 (0.028733)
```

```
6. feature 11 (0.015035)
7. feature 18 (0.012780)
8. feature 3 (0.011008)
9. feature 4 (0.009966)
10. feature 19 (0.005065)
11. feature 7 (0.004577)
12. feature 9 (0.003981)
13. feature 5 (0.003667)
14. feature 8 (0.003639)
15. feature 6 (0.002623)
16. feature 2 (0.001102)
17. feature 20 (0.001028)
18. feature 16 (0.000970)
19. feature 17 (0.000606)
20. feature 15 (0.000193)
21. feature 14 (0.000174)
```



[282]:

```
df_encoded_new = df_encoded.copy() # taking copy of encoded data to keep it␣
↪without change.
df_encoded_new.drop(drop_column, axis=1, inplace=True)
```

[284]:
```
#train-test splitting
train_data_enc_new = df_encoded_new[:int(0.7*(len(df_encoded_new)))]
test_data_enc_new = df_encoded_new[int(0.7*(len(df_encoded_new))):]

target = "Weekly_Sales"
used_cols2 = [c for c in df_encoded_new.columns.to_list() if c not in [target]]␣
↪# all columns except price

X_train_enc1 = train_data_enc_new[used_cols2]
X_test_enc1 = test_data_enc_new[used_cols2]
Y_train_enc1 = train_data_enc_new[target]
Y_test_enc1 = test_data_enc_new[target]

#droping date from train-test
X_train_enc1 = X_train_enc1.drop(['Date'], axis=1)
X_test_enc1= X_test_enc1.drop(['Date'], axis=1)
```

[286]:
```
rf = RandomForestRegressor(n_estimators=50, random_state=42, n_jobs=-1,␣
↪max_depth=40,
                           max_features = 'log2',min_samples_split = 10)

scaler=RobustScaler()
pipe = make_pipeline(scaler,rf)

pipe.fit(X_train_enc1, Y_train_enc1)

# predictions on train set
Y_pred_enc = pipe.predict(X_train_enc1)

# predictions on test set
Y_pred_test_enc = pipe.predict(X_test_enc1)
```

[290]:
```
pipe.score(X_test_enc1,Y_test_enc1)
```

[290]: 0.7301163967714206

[294]:
```
wmae_test(Y_test_enc1, Y_pred_test_enc)
```

[294]: 2000.840362122697

Best results with doing feature selection from whole encoded dataset.

[297]:
```
#With the same dateset before, I try to model again without month column.
df_encoded_new1 = df_encoded.copy()
df_encoded_new1.drop(drop_column, axis=1, inplace=True)
```

[299]:
```
df_encoded_new1 = df_encoded_new1.drop(['Date'], axis=1)
```

[301]:
```
df_encoded_new1 = df_encoded_new1.drop(['month'], axis=1)
```

[303]:
```
#train-test split
train_data_enc_new1 = df_encoded_new1[:int(0.7*(len(df_encoded_new1)))]
test_data_enc_new1 = df_encoded_new1[int(0.7*(len(df_encoded_new1))):]

target = "Weekly_Sales"
used_cols3 = [c for c in df_encoded_new1.columns.to_list() if c not in␣
 ↪[target]] # all columns except price

X_train_enc2 = train_data_enc_new1[used_cols3]
X_test_enc2 = test_data_enc_new1[used_cols3]
Y_train_enc2 = train_data_enc_new1[target]
Y_test_enc2 = test_data_enc_new1[target]
```

[305]:
```
#modeling part
pipe = make_pipeline(scaler,rf)

pipe.fit(X_train_enc2, Y_train_enc2)

# predictions on train set
Y_pred_enc = pipe.predict(X_train_enc2)

# predictions on test set
Y_pred_test_enc = pipe.predict(X_test_enc2)
```

[307]:
```
pipe.score(X_test_enc2, Y_test_enc2)
```

[307]: 0.7151872056567885

[309]:
```
wmae_test(Y_test_enc2, Y_pred_test_enc)
```

[309]: 2054.609836290659

Not better results than before.

[312]:
```
df_results = pd.DataFrame(columns=["Model", "Info",'WMAE']) # result df for␣
 ↪showing results together
```

[316]:
```
# writing results to df
df_results = pd.concat([df_results, pd.DataFrame([{
     "Model": 'RandomForestRegressor',
      "Info": 'w/out divided holiday columns',
       'WMAE': 4701
}])], ignore_index=True)
```

[320]:
```
df_results = pd.concat([df_results, pd.DataFrame([{
     "Model": 'RandomForestRegressor' ,
```

```
       "Info": 'w/out month column' ,
        'WMAE' : 4341}])], ignore_index=True)
df_results = pd.concat([df_results, pd.DataFrame([{
      "Model": 'RandomForestRegressor' ,
       "Info": 'whole data' ,
        'WMAE' : 2527}])], ignore_index=True)
df_results = pd.concat([df_results, pd.DataFrame([{
      "Model": 'RandomForestRegressor' ,
       "Info": 'whole data with feature selection' ,
        'WMAE' : 2000}])], ignore_index=True)
df_results = pd.concat([df_results, pd.DataFrame([{
      "Model": 'RandomForestRegressor' ,
       "Info": 'whole data with feature selection w/out month' ,
        'WMAE' : 2044}])], ignore_index=True)
```

[322]:
```
df_results
```

[322]:

	Model	Info	WMAE
0	RandomForestRegressor	w/out divided holiday columns	4701
1	RandomForestRegressor	w/out month column	4341
2	RandomForestRegressor	whole data	2527
3	RandomForestRegressor	whole data with feature selection	2000
4	RandomForestRegressor	whole data with feature selection w/out month	2044

TIME SERIES MODEL

[325]:
```
df.head()
```

[325]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	\
0	1	1	2010-02-05	24924.50	False	42.31	2.572	
1	1	2	2010-02-05	50605.27	False	42.31	2.572	
2	1	3	2010-02-05	13740.12	False	42.31	2.572	
3	1	4	2010-02-05	39954.04	False	42.31	2.572	
4	1	5	2010-02-05	32229.38	False	42.31	2.572	

	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	\
0	0.0	0.0	0.0	0.0	0.0	211.096358	
1	0.0	0.0	0.0	0.0	0.0	211.096358	
2	0.0	0.0	0.0	0.0	0.0	211.096358	
3	0.0	0.0	0.0	0.0	0.0	211.096358	
4	0.0	0.0	0.0	0.0	0.0	211.096358	

	Unemployment	Type	Size	Super_Bowl	Labor_Day	Thanksgiving	Christmas	\
0	8.106	A	151315	False	False	False	False	
1	8.106	A	151315	False	False	False	False	
2	8.106	A	151315	False	False	False	False	
3	8.106	A	151315	False	False	False	False	
4	8.106	A	151315	False	False	False	False	

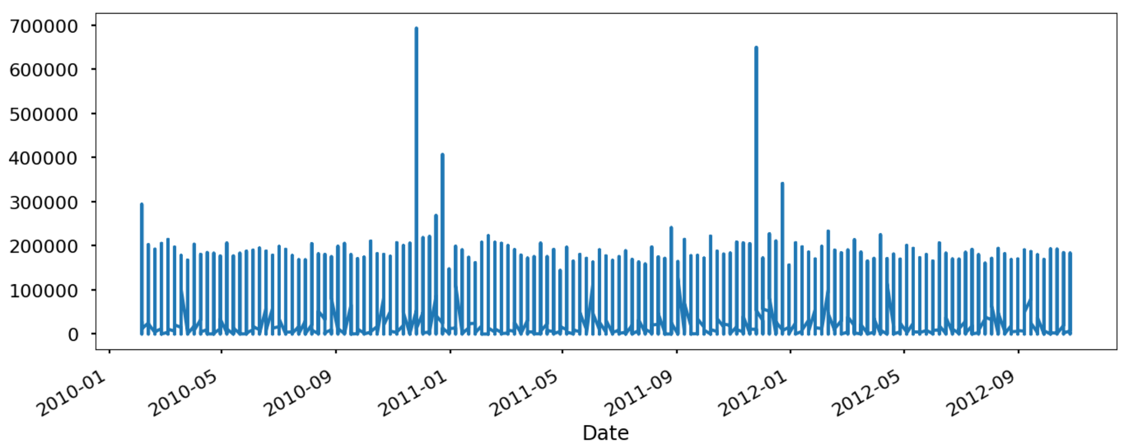
```
   week  month  year
0     5      2  2010
1     5      2  2010
2     5      2  2010
3     5      2  2010
4     5      2  2010
```

[327]:
```
df['Date'] = pd.to_datetime(df['Date'])
```

[329]:
```
df.set_index('Date', inplace=True) #seting date as index
```

PLOTTING SALES

[332]:
```
plt.figure(figsize=(16,6))
df['Weekly_Sales'].plot()
plt.show()
```



In this data, there are lots of same data values. So, I will collect them together as weekly.

[347]:
```
df_week = df.resample('W').mean() #resample data as weekly
```

[337]:
```
print(df.dtypes)
```

```
Store              int64
Dept               int64
Weekly_Sales     float64
IsHoliday           bool
Temperature      float64
Fuel_Price       float64
MarkDown1        float64
MarkDown2        float64
MarkDown3        float64
```

```
MarkDown4        float64
MarkDown5        float64
CPI              float64
Unemployment     float64
Type              object
Size               int64
Super_Bowl          bool
Labor_Day           bool
Thanksgiving        bool
Christmas           bool
week               int64
month              int64
year               int64
dtype: object
```

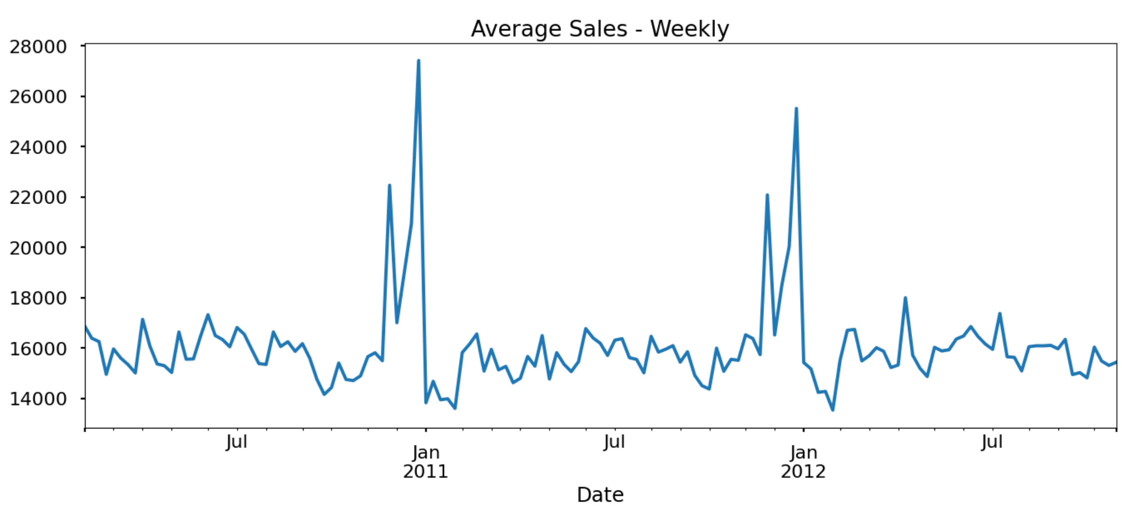
[339]:
```
df_week = df.select_dtypes(include=['number']).resample('W').mean()
```

[341]:
```
df = df.apply(pd.to_numeric, errors='coerce')
df_week = df.resample('W').mean()
```

[343]:
```
for col in df.columns:
    if df[col].dtype == 'object':
        print(f"Column: {col}, Unique Values: {df[col].unique()[:10]}")
```

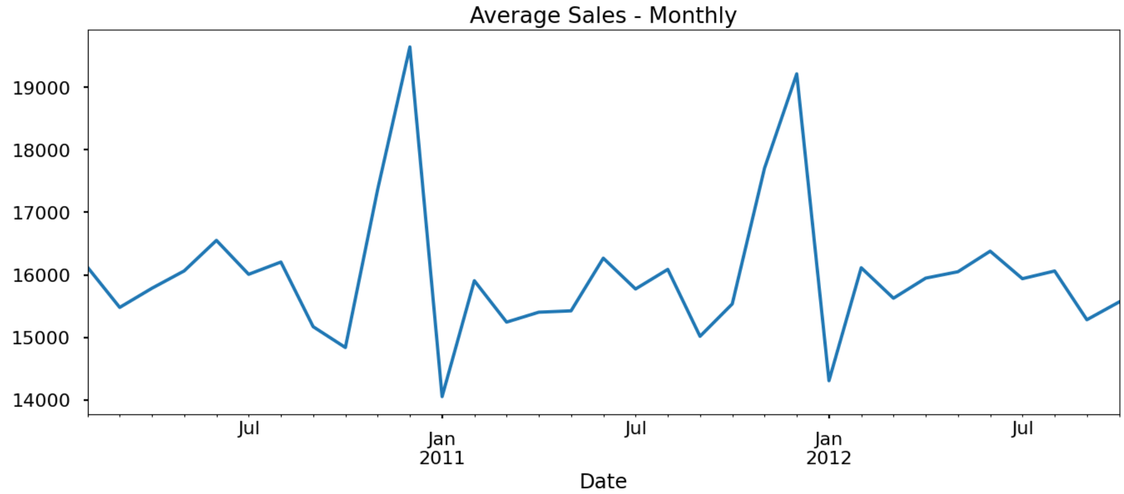
[345]:
```
df.replace({'AAAAA': np.nan, 'BBBBB': np.nan, 'CCCCC': np.nan}, inplace=True)
```

[349]:
```
plt.figure(figsize=(16,6))
df_week['Weekly_Sales'].plot()
plt.title('Average Sales - Weekly')
plt.show()
```



[351]:
```
df_month = df.resample('MS').mean() # resampling as monthly
```

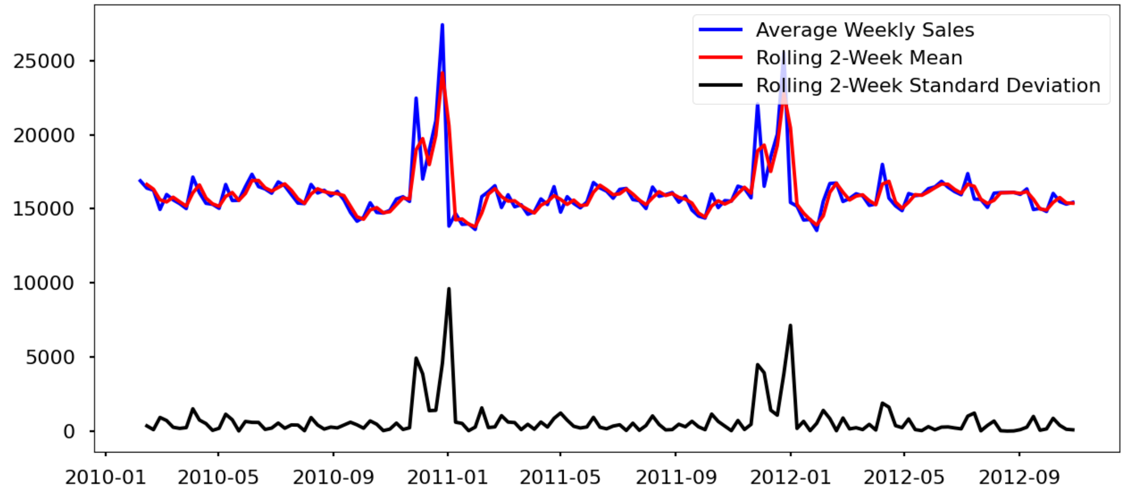
[353]:
```
plt.figure(figsize=(16,6))
df_month['Weekly_Sales'].plot()
plt.title('Average Sales - Monthly')
plt.show()
```



When I turned data to monthly, I realized that I lost some patterns in weekly data. So, I will continue with weekly resampled data.

[356]:
```
# finding 2-weeks rolling mean and std
roll_mean = df_week['Weekly_Sales'].rolling(window=2, center=False).mean()
roll_std = df_week['Weekly_Sales'].rolling(window=2, center=False).std()
```

[358]:
```
fig, ax = plt.subplots(figsize=(13, 6))
ax.plot(df_week['Weekly_Sales'], color='blue',label='Average Weekly Sales')
ax.plot(roll_mean, color='red', label='Rolling 2-Week Mean')
ax.plot(roll_std, color='black', label='Rolling 2-Week Standard Deviation')
ax.legend()
fig.tight_layout()
```



Adfuller Test to Make Sure

[365]:
```
adfuller(df_week['Weekly_Sales'])
```

[365]:
```
(-5.927107223737566,
 2.4290492082043256e-07,
 4,
 138,
 {'1%': -3.47864788917503,
  '5%': -2.882721765644168,
  '10%': -2.578065326612056},
 2261.596421168073)
```

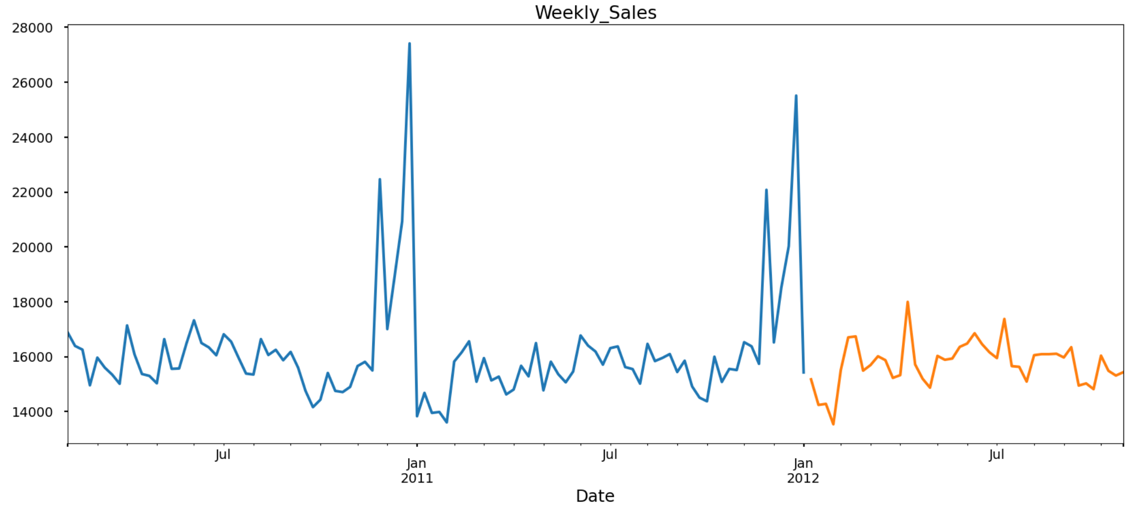
[367]:
```
train_data = df_week[:int(0.7*(len(df_week)))]
test_data = df_week[int(0.7*(len(df_week))):]

print('Train:', train_data.shape)
print('Test:', test_data.shape)
```

```
Train: (100, 22)
Test: (43, 22)
```

[369]:
```
target = "Weekly_Sales"
used_cols = [c for c in df_week.columns.to_list() if c not in [target]] # all␣
 ↪columns except price

# assigning train-test X-y values

X_train = train_data[used_cols]
X_test = test_data[used_cols]
Y_train = train_data[target]
```

```
Y_test = test_data[target]
```

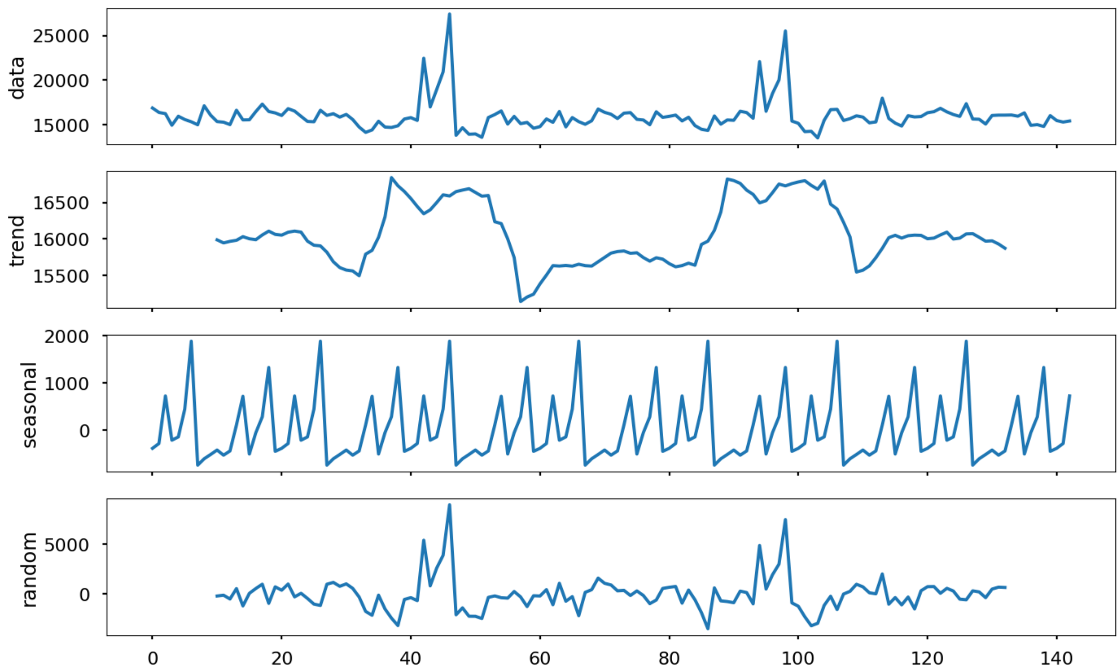
[371]:
```
train_data['Weekly_Sales'].plot(figsize=(20,8), title= 'Weekly_Sales',␣
↪fontsize=14)
test_data['Weekly_Sales'].plot(figsize=(20,8), title= 'Weekly_Sales',␣
↪fontsize=14)
plt.show()
```



Blue line represents my train data, orange is test data.

[380]:
```
decomposed = decompose(df_week['Weekly_Sales'].values, 'additive', m=20)␣
↪#decomposing of weekly data
```

[382]:
```
decomposed_plot(decomposed, figure_kwargs={'figsize': (16, 10)})
plt.show()
```

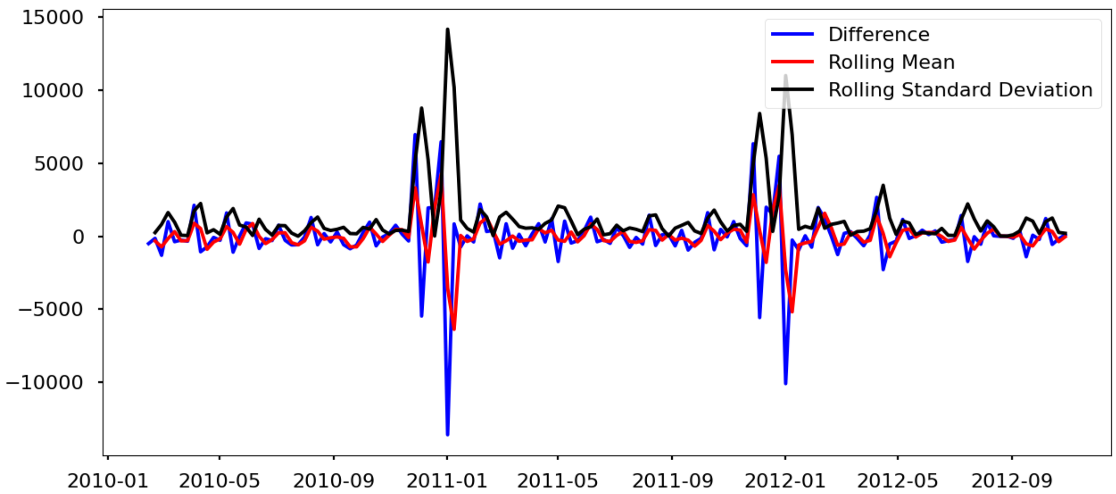


From the graphs above, every 20 step seasonality converges to beginning point. This helps me to tune my model.

[385]:
```
# I will try to make my data more stationary. To do this, I will try model with␣
↪differenced, logged and shifted data.

df_week_diff = df_week['Weekly_Sales'].diff().dropna() #creating difference␣
↪values
```

[387]:
```
# taking mean and std of differenced data
diff_roll_mean = df_week_diff.rolling(window=2, center=False).mean()
diff_roll_std = df_week_diff.rolling(window=2, center=False).std()
```

[389]:
```
fig, ax = plt.subplots(figsize=(13, 6))
ax.plot(df_week_diff, color='blue',label='Difference')
ax.plot(diff_roll_mean, color='red', label='Rolling Mean')
ax.plot(diff_roll_std, color='black', label='Rolling Standard Deviation')
ax.legend()
fig.tight_layout()
```

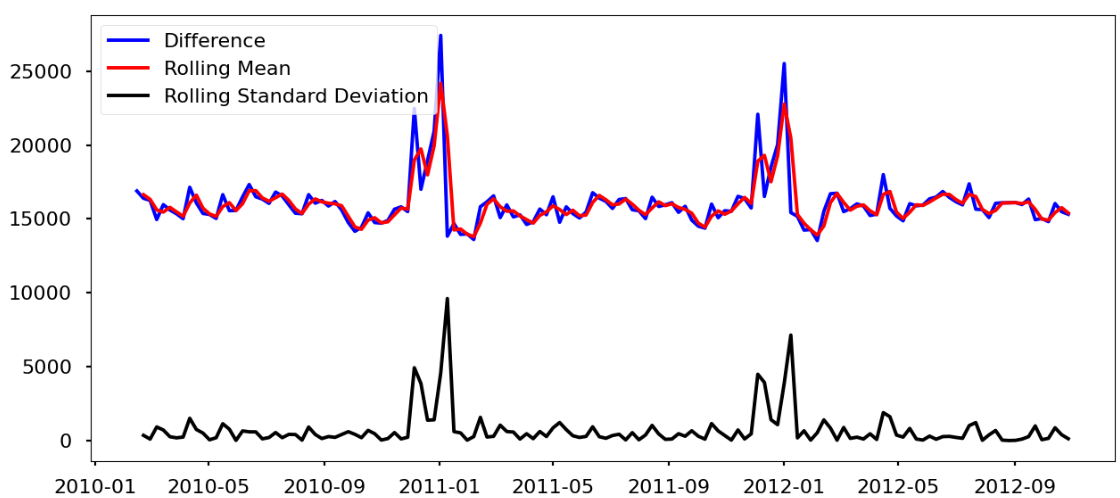


2. SHIFT

[392]:
```
df_week_lag = df_week['Weekly_Sales'].shift().dropna() #shifting the data
```

[394]:
```
lag_roll_mean = df_week_lag.rolling(window=2, center=False).mean()
lag_roll_std = df_week_lag.rolling(window=2, center=False).std()
```

[396]:
```
fig, ax = plt.subplots(figsize=(13, 6))
ax.plot(df_week_lag, color='blue',label='Difference')
ax.plot(lag_roll_mean, color='red', label='Rolling Mean')
ax.plot(lag_roll_std, color='black', label='Rolling Standard Deviation')
ax.legend()
fig.tight_layout()
```

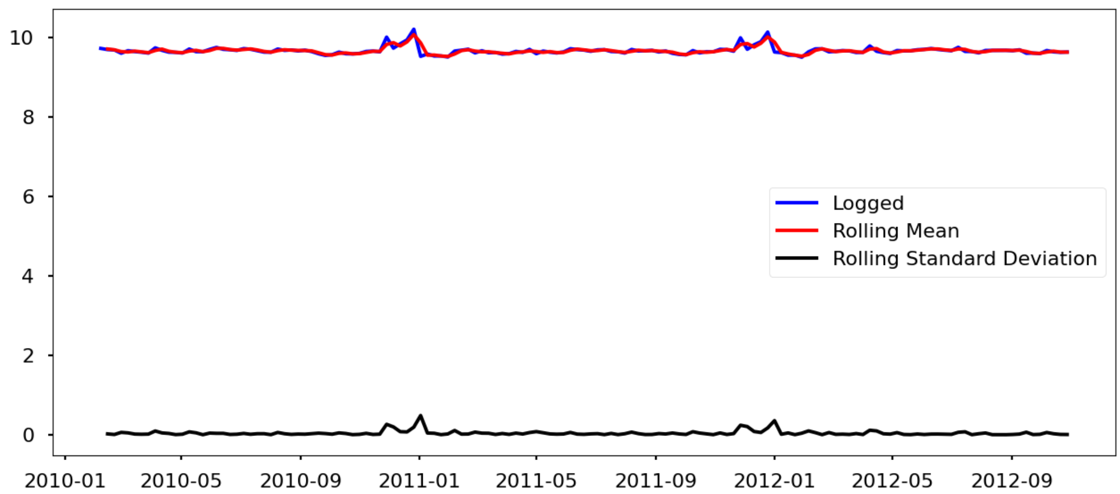


3. LOG

```
[399]: logged_week = np.log1p(df_week['Weekly_Sales']).dropna() #taking log of data
```

```
[401]: log_roll_mean = logged_week.rolling(window=2, center=False).mean()
       log_roll_std = logged_week.rolling(window=2, center=False).std()
```

```
[403]: fig, ax = plt.subplots(figsize=(13, 6))
       ax.plot(logged_week, color='blue',label='Logged')
       ax.plot(log_roll_mean, color='red', label='Rolling Mean')
       ax.plot(log_roll_std, color='black', label='Rolling Standard Deviation')
       ax.legend()
       fig.tight_layout()
```



Auto-ARIMA MODEL

I tried my data without any changes, then tried with shifting, taking log and difference version of data. Differenced data gave best results. So, I decided to take difference and use this data.

```
[407]: train_data_diff = df_week_diff [:int(0.7*(len(df_week_diff )))]
       test_data_diff = df_week_diff [int(0.7*(len(df_week_diff ))):]
```

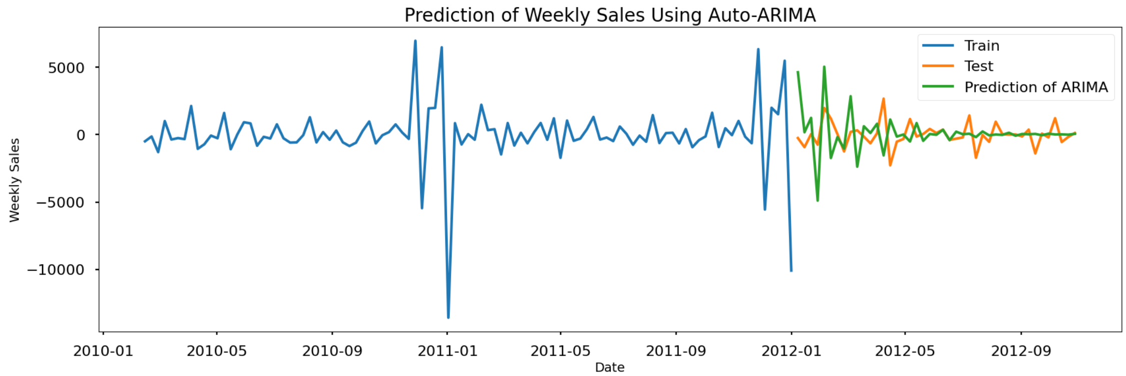
```
[409]: # train_data = train_data['Weekly_Sales']
       # test_data = test_data['Weekly_Sales']

       model_auto_arima = auto_arima(train_data_diff, trace=True,start_p=0, start_q=0,␣
        ↪start_P=0, start_Q=0,
                         max_p=20, max_q=20, max_P=20, max_Q=20,␣
        ↪seasonal=True,maxiter=200,
                         information_criterion='aic',stepwise=False,␣
        ↪suppress_warnings=True, D=1, max_D=10,
```

```
                     error_action='ignore',approximation = False)
model_auto_arima.fit(train_data_diff)
```

```
ARIMA(0,0,0)(0,0,0)[1] intercept   : AIC=1826.858, Time=0.06 sec
ARIMA(0,0,1)(0,0,0)[1] intercept   : AIC=1793.619, Time=0.07 sec
ARIMA(0,0,2)(0,0,0)[1] intercept   : AIC=1795.532, Time=0.21 sec
ARIMA(0,0,3)(0,0,0)[1] intercept   : AIC=inf, Time=0.25 sec
ARIMA(0,0,4)(0,0,0)[1] intercept   : AIC=inf, Time=0.42 sec
ARIMA(0,0,5)(0,0,0)[1] intercept   : AIC=inf, Time=0.26 sec
ARIMA(1,0,0)(0,0,0)[1] intercept   : AIC=1804.051, Time=0.02 sec
ARIMA(1,0,1)(0,0,0)[1] intercept   : AIC=inf, Time=0.16 sec
ARIMA(1,0,2)(0,0,0)[1] intercept   : AIC=1794.966, Time=0.14 sec
ARIMA(1,0,3)(0,0,0)[1] intercept   : AIC=inf, Time=0.27 sec
ARIMA(1,0,4)(0,0,0)[1] intercept   : AIC=inf, Time=0.35 sec
ARIMA(2,0,0)(0,0,0)[1] intercept   : AIC=1801.215, Time=0.03 sec
ARIMA(2,0,1)(0,0,0)[1] intercept   : AIC=inf, Time=0.25 sec
ARIMA(2,0,2)(0,0,0)[1] intercept   : AIC=inf, Time=0.27 sec
ARIMA(2,0,3)(0,0,0)[1] intercept   : AIC=inf, Time=0.55 sec
ARIMA(3,0,0)(0,0,0)[1] intercept   : AIC=1791.045, Time=0.06 sec
ARIMA(3,0,1)(0,0,0)[1] intercept   : AIC=1787.198, Time=0.11 sec
ARIMA(3,0,2)(0,0,0)[1] intercept   : AIC=1782.922, Time=0.10 sec
ARIMA(4,0,0)(0,0,0)[1] intercept   : AIC=1785.231, Time=0.06 sec
ARIMA(4,0,1)(0,0,0)[1] intercept   : AIC=1786.221, Time=0.18 sec
ARIMA(5,0,0)(0,0,0)[1] intercept   : AIC=1784.877, Time=0.07 sec

Best model:  ARIMA(3,0,2)(0,0,0)[1] intercept
Total fit time: 3.963 seconds
```

[409]:
```
ARIMA(maxiter=200, order=(3, 0, 2), scoring_args={},
      seasonal_order=(0, 0, 0, 1), suppress_warnings=True)
```

[432]:
```
Y_pred = model_auto_arima.predict(n_periods=len(test_data_diff))
Y_pred = pd.DataFrame(Y_pred,index = test_data.index,columns=['Prediction'])
plt.figure(figsize=(20,6))
plt.title('Prediction of Weekly Sales Using Auto-ARIMA', fontsize=20)
plt.plot(train_data_diff, label='Train')
plt.plot(test_data_diff, label='Test')
plt.plot(Y_pred, label='Prediction of ARIMA')
plt.legend(loc='best')
plt.xlabel('Date', fontsize=14)
plt.ylabel('Weekly Sales', fontsize=14)
plt.show()
```



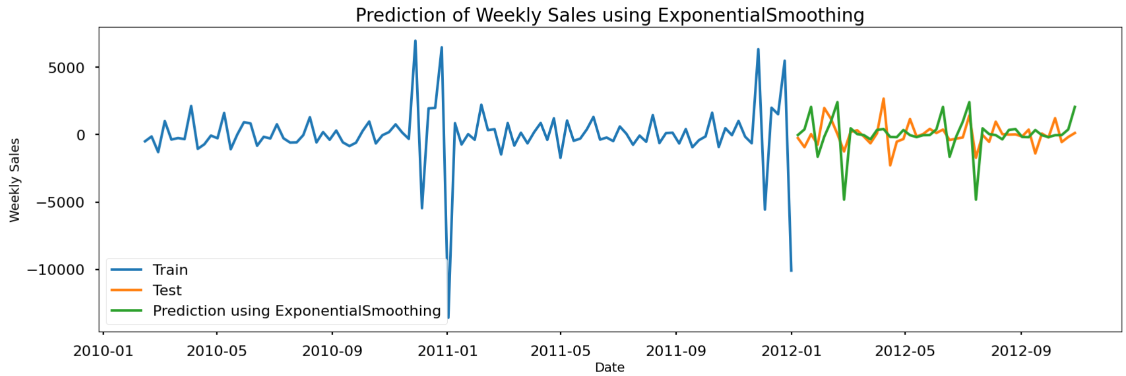
I do not like the pattern of predictions so I decided to try another model.

ExponentialSmoothing

I checked suitable Holt-Winters models according tp my data. Exponential Smooting are used when data has trend, and it flattens the trend. The damped trend method adds a damping parameter so, the trend converges to a constant value in the future.

My difference data has some minus and zero values, so I used additive seasonal and trend instead of multiplicative. Seasonal periods are chosen from the decomposed graphs above. For tuning the model with iterations take too much time so, I changed and tried model for different parameters and found the best parameters and fitted them to model.

[437]:
```
model_holt_winters = ExponentialSmoothing(train_data_diff, seasonal_periods=20,␣
 ↪seasonal='additive',
                                           trend='additive',damped=True).fit()␣
 ↪#Taking additive trend and seasonality.
Y_pred = model_holt_winters.forecast(len(test_data_diff))# Predict the test data

#Visualize train, test and predicted data.
plt.figure(figsize=(20,6))
plt.title('Prediction of Weekly Sales using ExponentialSmoothing', fontsize=20)
plt.plot(train_data_diff, label='Train')
plt.plot(test_data_diff, label='Test')
plt.plot(Y_pred, label='Prediction using ExponentialSmoothing')
plt.legend(loc='best')
plt.xlabel('Date', fontsize=14)
plt.ylabel('Weekly Sales', fontsize=14)
plt.show()
```



[439]:
```
wmae_test(test_data_diff, Y_pred)
```

[439]: 840.681060966696

At the end, I found best results for my data with Exponential Smoothing Model.

My best result for this project is 840. According to sales amounts this value is roughly around 5% error. If we can take our average sales and take percentage of 840 errors, it gives 5% roughly.

[ ]:
```

```