FourthBrain

EDA with Walmart Sales Data

Business Objectives

Walmart, the retail giant that operates a chain of hypermarkets, wants to understand their weekly sales data, especially the impact from holidays and or big events on the weekly sales data; specifically, Super Bowl, Labor Day, Thanksgiving, and Christmas. In addition, Walmart wants to consider the effect from different macroeconomic/external factors.

Learning Objectives

At the end of this session, you will know how to

1. Manipulate data of different types using `pandas`
2. Visualize data with `matplotlib` and `seaborn` to extract insights
3. Perform feature engineering
4. Build a pipeline to preprocess data and fit a simple model using `sklearn`

Note: if you see code that's unfamiliar to you, look up for the documentation, and try to understand what it does.

Data Overview

- Original sales data were collected from 45 stores across the United States; yet for this session, you will first inspect data from three stores and later focus on just store 1.
- Each store is of certain type and size, and there are multiple departments in a store.
- The dataset has a temporal component, we ignore this mostly in this session and will discuss time series related techniques later in the cohort.

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all" # allow multiple outputs in
import warnings
warnings.filterwarnings("ignore")
```

Task I: Load Data

Built on top of `numpy`, `pandas` is one of the most widely used tools in machine learning. Its rich features are used for exploring, cleaning, visualizing, and transforming data. We need to import the library to access all of its features.

In [2]:
```
import pandas as pd
```

Use `pd.read_csv` to read `train_comb.csv` that contains weekly sales, metadata, and macroeconomic features from three stores into a `pd.DataFrame`.

In [3]:
```
filepath = '../dat/train_comb.csv'
data = pd.read_csv(filepath)
```

Verify that the data is loaded correctly by running `data.head(3)` to see the first few row (AVOID printing out the entire DataFrame, i.e., `data` or `print(data)`; it might be trivial for small dataset but it can crash your kernel when the dataset is big and slow down the initial data exploration process).

In [4]:
```
print(data.shape)
print(data.columns)
data.head(3)
data.MarkDown1.value_counts()
```

```
(30990, 16)
Index(['Store', 'Dept', 'Date', 'Weekly_Sales', 'IsHoliday', 'Temperatur
e',
       'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4',
       'MarkDown5', 'CPI', 'Unemployment', 'Type', 'Size'],
      dtype='object')
```

Out[4]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2
0	1	1	2010-02-05	24924.50	False	42.31	2.572	NaN	NaN
1	1	1	2010-02-12	46039.49	True	38.51	2.548	NaN	NaN
2	1	1	2010-02-19	41595.55	False	39.93	2.514	NaN	NaN

Out[4]:

```
1282.42     75
9264.48     75
686.24      75
5924.71     75
16404.54    74
            ..
12218.76    70
1164.46     70
10165.22    70
7011.68     70
3965.73     69
Name: MarkDown1, Length: 153, dtype: int64
```

❓ Question 1:

Look at the output to get an idea of what each column is and then write a few sentences describing what you notice about the data. You can also use `data.sample(3)` to draw random samples from the data (hints: number of rows and columns, any missing values? data types of the elements? date ranges of the data collected? etc.).

In [5]:
```
data.sample(3)
data.Store.value_counts()
```

Out[5]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDo
21955	13	11	2010-04-09	18898.52	False	38.97	2.877	NaN	N
3918	1	29	2011-10-07	4011.50	False	69.31	3.285	NaN	N
21108	13	5	2010-06-25	45776.21	False	71.83	2.820	NaN	N

Out[5]:
```
13    10474
4     10272
1     10244
Name: Store, dtype: int64
```

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
Store              0
Dept               0
Date               0
Weekly_Sales       0
IsHoliday          0
Temperature        0
Fuel_Price         0
MarkDown1      19928
MarkDown2      21730
MarkDown3      20211
MarkDown4      20000
MarkDown5      19928
CPI                0
Unemployment       0
Type               0
Size               0
dtype: int64
```

In [7]:
```
data.describe()
```

Out[7]:

	Store	Dept	Weekly_Sales	Temperature	Fuel_Price	MarkDown1	Mar
count	30990.000000	30990.000000	30990.000000	30990.000000	30990.000000	11062.000000	9260
mean	6.050145	44.513746	26087.914360	61.278170	3.240187	9542.600315	4586
std	5.113624	29.835120	32376.766734	17.113568	0.412234	8561.277370	12377
min	1.000000	1.000000	-898.000000	16.940000	2.514000	410.310000	-10
25%	1.000000	20.000000	4539.080000	47.960000	2.837000	4539.940000	67
50%	4.000000	38.000000	12941.920000	63.930000	3.294000	7146.900000	193
75%	13.000000	72.000000	35645.547500	76.800000	3.610000	11075.380000	3579
max	13.000000	99.000000	385051.040000	91.650000	3.907000	53423.290000	89121

In [8]:
```
data.dtypes
```

Out[8]:
```
Store             int64
Dept              int64
Date             object
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
dtype: object
```

Question 1

Number of rows: 30,990 Number of columns: 16 Data types: Described in cell above -> Integer, boolean, float, date Descriptive: `data.describe()` shows descriptive characteristics This data representatives weekly sales in three Walmart stores during holidays There are plently of null values in the MarkDown# columns which might post problems and can be resolved through some data cleansing. Ideas: Check the impact of particular holidays, temp, fuel price, correlation with economic indicators.

Acceptable responses include the number of rows and columns in the dataset, the data types of the elements, how many NaNs there are (and perhaps which columns and/or rows tend to have them), the range of values in each column or other descriptive statistics, some commentary on what this data represents, any initial concerns about how you think we should model this data, or any other commentary you would like to add.

Use `.shape` to inspect the size of the data: sample size and number of features.

In [9]:
```
data.shape
```

Out[9]:
```
(30990, 16)
```

Expected Output

For the following task, we focus on Store `1` only,

In [10]:
```
data_store1 = data[data.Store == 1]
print(data_store1.shape)
data_store1.head()
```

```
(10244, 16)
```

Out[10]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2
0	1	1	2010-02-05	24924.50	False	42.31	2.572	NaN	NaN
1	1	1	2010-02-12	46039.49	True	38.51	2.548	NaN	NaN
2	1	1	2010-02-19	41595.55	False	39.93	2.514	NaN	NaN
3	1	1	2010-02-26	19403.54	False	46.63	2.561	NaN	NaN
4	1	1	2010-03-05	21827.90	False	46.50	2.625	NaN	NaN

Retrieve the data from department 9 (a random choice) at store 1:

In [11]:
```
data_store1_dept9 = data_store1[data_store1.Dept == 9]
```

Verify the result using `.head()` , `.shape` .

In [12]:
```
data_store1_dept9.head()
data_store1_dept9.shape
```

Out[12]:

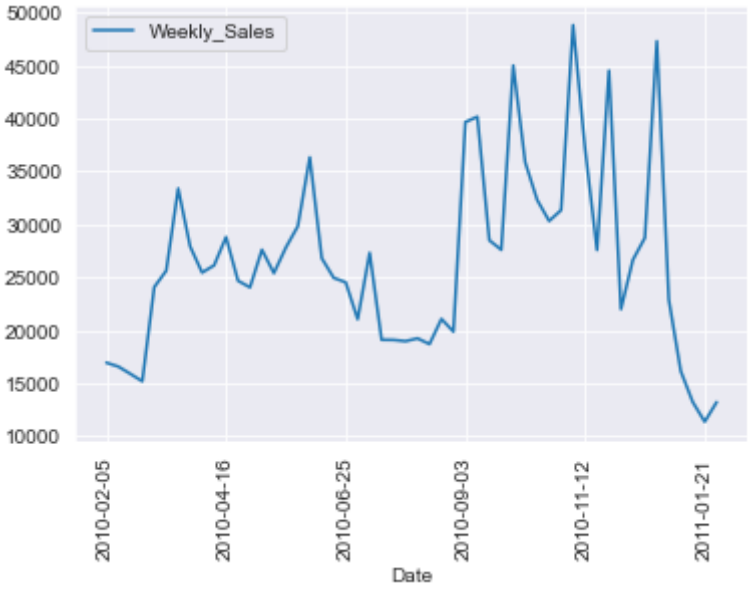
	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDow
1144	1	9	2010-02-05	16930.99	False	42.31	2.572	NaN	Na
1145	1	9	2010-02-12	16562.49	True	38.51	2.548	NaN	Na
1146	1	9	2010-02-19	15880.85	False	39.93	2.514	NaN	Na
1147	1	9	2010-02-26	15175.52	False	46.63	2.561	NaN	Na
1148	1	9	2010-03-05	24064.70	False	46.50	2.625	NaN	Na

Out[12]: (143, 16)

Expected Output

Visualize one full year of sales. The data came with dates sorted, but we can make sure of it and then visualize the first 52 data points.

In [13]:
```
data_store1_dept9 = data_store1_dept9.sort_values('Date')
data_store1_dept9[['Date', 'Weekly_Sales']].iloc[:52]\
    .set_index('Date').plot(rot=90);
```



❓ Question 2:

Do you have any hypotheses about the holidays' impact on the sales?

Question 2

> There's some evidence that holidays do impact sales figures as witnessed by the spikes such as Christmas and Thanksgiving.

For the purpose of this notebook, we focus on the sales data from Store 1 in DataFrame `df` and is saved in `train_store1.csv` . Let's read in the data.

In [14]:
```
df = pd.read_csv("../dat/train-store1.csv")
```

Extract week, month, and year information from the raw `Date` column to better manipulate the weekly data later. Pandas comes with powerful features to make this step easy. Reference: tutorial (https://pandas.pydata.org/docs/getting_started/intro_tutorials/09_timeseries.html).

First, use `.dtypes` to check the datatype of the `Date` column. What's the difference between `df[['Date']]` and `df['Date']` ?.

In [15]:
```
df['Date'] = pd.to_datetime(df.Date)
df['year'] = df.Date.dt.year
df['month'] = df.Date.dt.month
df['week'] = df.Date.dt.week
df.head()
```

Out[15]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Temperature	Fuel_Price	MarkDown1	MarkDown2
0	1	1	2010-02-05	24924.50	False	42.31	2.572	NaN	NaN
1	1	1	2010-02-12	46039.49	True	38.51	2.548	NaN	NaN
2	1	1	2010-02-19	41595.55	False	39.93	2.514	NaN	NaN
3	1	1	2010-02-26	19403.54	False	46.63	2.561	NaN	NaN
4	1	1	2010-03-05	21827.90	False	46.50	2.625	NaN	NaN

Question 2b

> `df['Date']` returns pd.Series whereas `df[['Date']]` returns pd.DataFrame.

In [15]:
```
```

Expected Output

In [16]:
```
df.Date=pd.to_datetime(df.Date)
```

Verify that the `Date` column's datatype has changed as expected:

In [17]:
```
df[['Date']].dtypes
```

Out[17]:
```
Date    datetime64[ns]
dtype: object
```

In [18]:
```
df['week'] = df.Date.dt.week
df['month'] = df.Date.dt.month
df['year'] = df.Date.dt.year
```

Verify that now there are 19 columns in `df` :

In [19]:
```
df.shape
```

Out[19]:
```
(10244, 19)
```

? Question 3:

Last step before we look deeper into the features is to split the data set into training and testing datasets. Discuss: why do we want to perform EDA only on the training data, not the entire dataset? Shouldn't it be the more the better?

Question 3

> You always split into train/test subsets as this is the only way to check whether the bias/variance problem occurs. Additionally, splitting the data in the very beginning is a good practice that helps avoiding one of the data leakage reasons (leaky validation strategies).

The answer should mention data leakage, and / or overfitting

Split the data into training (80%) and test dataset (20%). Use function `train_test_split` from `scikit-learn` (a popular library for machine learning in Python), and set `random_state` to be 42 for reproducibility (this is not the best way to do train-test-split due to the temporal nature of

the data, however, we will ignore it for now).

In [20]:
```
from sklearn.model_selection import train_test_split
```

In [21]:
```
df_train, df_test =  train_test_split(df, test_size=0.2, random_state=42)
```

In [22]:
```
print('Original set  ---> ',df.shape,
      '\nTraining set  ---> ',df_train.shape,
      '\nTesting set   ---> ', df_test.shape)
```

```
Original set  --->  (10244, 19)
Training set  --->  (8195, 19)
Testing set   --->  (2049, 19)
```

Expected Output

Task II: Target, Features, and Distributions

We inspect the datatype of column `Date` ; now find datatypes for all columns in `df_train` using `.dtypes` :

In [23]:
```
df_train.dtypes
```

Out[23]:
```
Store                    int64
Dept                     int64
Date            datetime64[ns]
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
year                     int64
month                    int64
week                     int64
dtype: object
```

Expected Output

Summary statistics provide you with a general understanding of the data. Use method

`.describe()` . By default it reports statistics mean, max, min, quantiles for numerical features and counts, unique, mode for categorical features.

In [24]:
```
pd.options.display.float_format = "{:,.2f}".format
df_train.describe()
```

Out[24]:

	Store	Dept	Weekly_Sales	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDo
count	8,195.00	8,195.00	8,195.00	8,195.00	8,195.00	2,931.00	2,424.00	2,87
mean	1.00	44.65	21,865.28	68.19	3.22	8,045.43	2,961.55	1,23
std	0.00	29.95	27,970.00	14.16	0.43	6,484.49	8,032.30	7,83
min	1.00	1.00	-863.00	35.40	2.51	410.31	0.50	
25%	1.00	20.00	3,502.09	57.79	2.76	4,039.39	40.48	
50%	1.00	38.00	10,357.32	69.64	3.29	6,154.14	137.86	3
75%	1.00	72.00	31,647.36	80.48	3.59	10,121.97	1,569.00	10
max	1.00	99.00	203,670.47	91.65	3.91	34,577.06	46,011.38	55,80

Expected Output

? Question 4:

Inspect the output, what are some of your observations?

Question 4

> Weekly sales changes heavily. Somehow we see that there are negative weekly sales. Fuel price is really good back in the day. Markdown# fields are full of NaNs.

Are there any missing values? Use `.isna()` and `.sum()` to show the number of missing values from each column.

In [25]:
```
df_train.isnull().sum()
```

Out[25]:
```
Store             0
Dept              0
Date              0
Weekly_Sales      0
IsHoliday         0
Temperature       0
Fuel_Price        0
MarkDown1      5264
MarkDown2      5771
MarkDown3      5317
MarkDown4      5264
MarkDown5      5264
CPI               0
Unemployment      0
Type              0
Size              0
year              0
month             0
week              0
dtype: int64
```

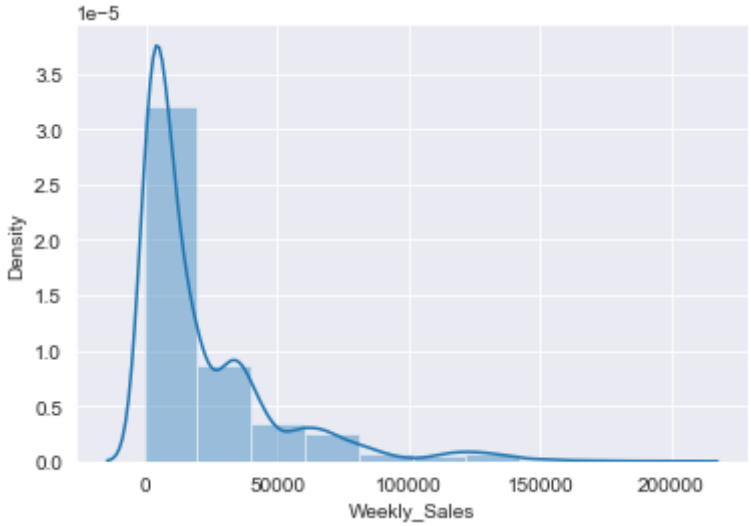
Expected Output

What do you think the target variable is in this problem? Assign the column name to `target` for later use.

In [26]:
```
target = 'Weekly_Sales'
```

Visualize the distribution of target variable using `distplot()` from library `seaborn` (Why seaborn? Check out a comparison between Matplotlib and Seaborn here (https://analyticsindiamag.com/comparing-python-data-visualization-tools-matplotlib-vs-seaborn/)). Anything here you observe but the output from `.describe` does not make obvious? Does it follow a normal distribution?

In [27]:
```
import seaborn as sns
sns.distplot(df_train[target],bins=10);
```



Notice that there exists nonpositive weekly sales. How many of rows are there that the weekly sales are negative or 0?

In [28]:
```
(df_train[target] <= 0).sum() # Expected Output: 13
```

Out[28]:
```
13
```

What percentage is the negative and zero sales?

In [29]:
```
print(f'Percentage of negative and zero sales is {100 * (df_train[target] <
```

```
Percentage of negative and zero sales is 0.1586%.
```

Expected Output

After communicating your findings, the stakeholders confirm that you can remove these data

entries for now and they are launching an investigation by analysts and data engineers.

Now remove them from the training dataset.

In [30]:
```
mask = df_train[target] > 0
df_train = df_train[mask]
df_train.shape # Expected Output: (8182, 19)
```

Out[30]:
```
(8182, 19)
```

Let's move on to features.

Though almost all the come through as numerical, should they all be treated as numerical features? Let's inspect the number of unique values:

In [31]:
```
[(col, df[col].nunique())for col in df_train.columns]
```

Out[31]:
```
[('Store', 1),
 ('Dept', 77),
 ('Date', 143),
 ('Weekly_Sales', 10042),
 ('IsHoliday', 2),
 ('Temperature', 143),
 ('Fuel_Price', 137),
 ('MarkDown1', 51),
 ('MarkDown2', 41),
 ('MarkDown3', 49),
 ('MarkDown4', 51),
 ('MarkDown5', 51),
 ('CPI', 143),
 ('Unemployment', 12),
 ('Type', 1),
 ('Size', 1),
 ('year', 3),
 ('month', 12),
 ('week', 52)]
```

`Temperature` , `CPI` , `Unemployment` , `Fuel_Price` are continuous. Those tie to the second business objective. Let us put these four into a list and store it in `external_factors` . From earlier, we noticed that `MarkDownx` columns contain some missing values, we will treat them in a later task.

In [32]:
```
external_factors = ['Temperature','CPI','Unemployment', 'Fuel_Price']
```

Visualize Temperature in a box plot, what do you think is the advantage of a box plot over a histogram? You can use `pd.DataFrame.boxplot()` , set the figure size as (6, 4), and turn off the grid.

? Question 5:

Visualize Temperature in a box plot, what do you think the advantage of a box plot over histogram?

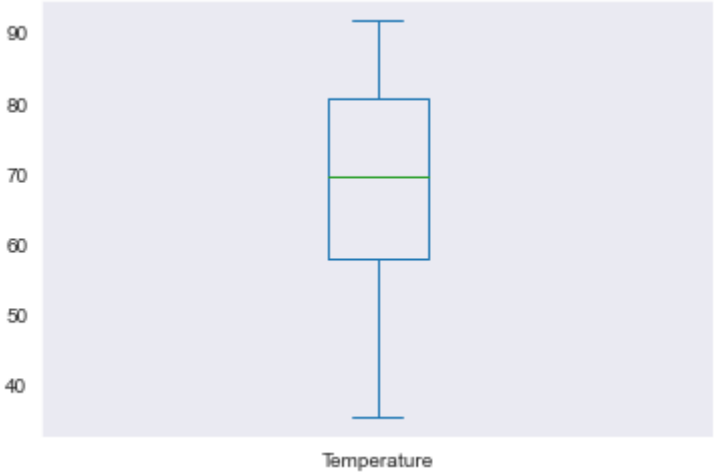
HINT: You can use `pd.DataFrame.boxplot()` , set the figure size as (6, 4), and turn off the grid.

Question 5

> Box plots show the summary of the data (quartiles, interquartile ranges, outliers) whereas histograms show data distribution. You can also use violin plots to get both at the same chart.

In [33]:
```
df_train[['Temperature']].plot.box(figsize=(6,4), grid=False)
```

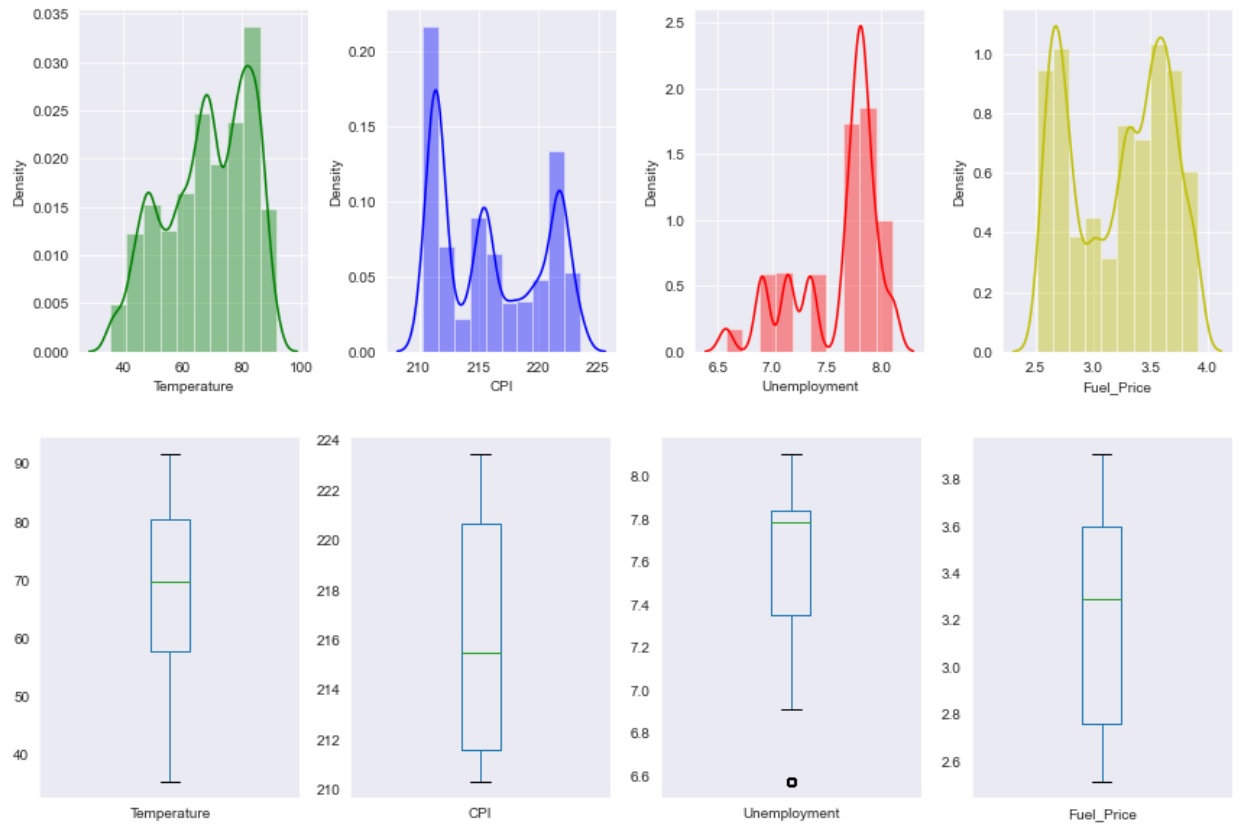
Out[33]: <AxesSubplot:>



Let's visualize all four numerical features in both density plot and box plot. Note any observations.

In [34]:
```
import matplotlib.pyplot as plt
print('\033[1mNumeric Features Distribution'.center(100))

figsize = (12, 4)

n=len(external_factors)
colors = ['g', 'b', 'r', 'y', 'k']

# histogram
plt.figure(figsize=figsize)
for i in range(len(external_factors)):
    plt.subplot(1,n,i+1)
    sns.distplot(df_train[external_factors[i]],
                 bins=10,
                 color = colors[i])
plt.tight_layout();

# boxplot
plt.figure(figsize=figsize)
for i in range(len(external_factors)):
    plt.subplot(1,n,i+1)
    df_train.boxplot(external_factors[i], grid=False)
plt.tight_layout();
```

Numeric Features Distribution



We will investigate the impacts of the external factors later. Now let's scan through the other features.

`Store` , `Type` , and `Size` each has only one unique value, offering no information, we can safely ignore them.

We extracted `year` , `month` , and `week` from `Date` , thus `Date` is redundant; but it is easy to find the date range in the training dataset using `Date` :

In [35]:
```
df_train['Date'].min(), df_train['Date'].max() # Expected Output: (Timestam
```

Out[35]:
```
(Timestamp('2010-02-05 00:00:00'), Timestamp('2012-10-26 00:00:00'))
```

Our training data ranges from 5th of February 2010 to 26th of October 2012.

It makes more sense to treat `year` , `month` , `week` as categorical, more accurately ordinal; and the boolean feature `IsHoliday` can be considered as categorical, so can `Dept` . Let's put these column names into a list `categoricalFeatures` .

In [36]:
```
categoricalFeatures = ['year','month','week','IsHoliday', 'Dept']
```

For the categorical features, we are more interested in the frequency of each value, use `pd.Series.value_counts` to see how many rows where `IsHoliday` is true and false respectively (Data imbalance is the norm).

In [37]:
```
df_train['IsHoliday'].value_counts()
```

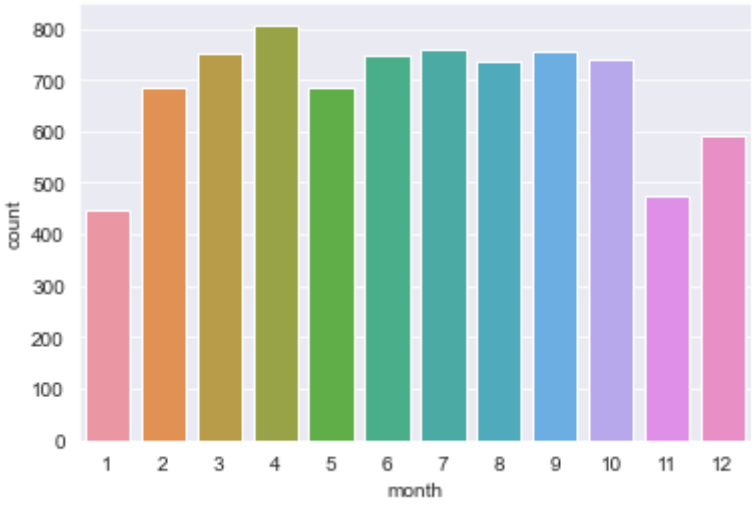
Out[37]:
```
False    7586
True      596
Name: IsHoliday, dtype: int64
```

Expected Output

Visualize the distribution of `month` ; use `sns.countplot()` .

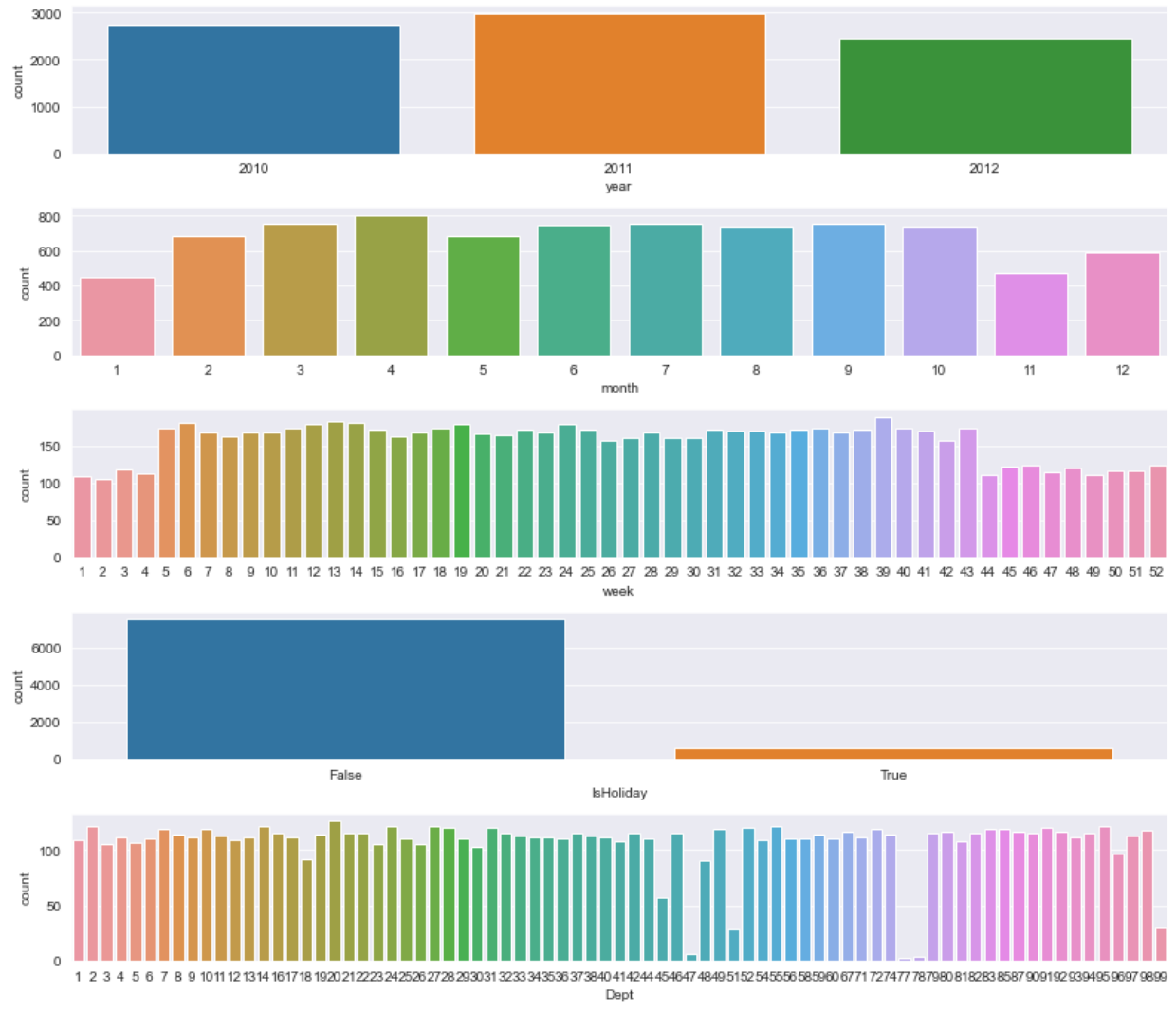
In [38]:
```
sns.countplot(df_train['month'])
```

Out[38]:
```
<AxesSubplot:xlabel='month', ylabel='count'>
```



In [39]:
```
#Visualising the categorical features

print('\033[1mVisualising Categorical Features:'.center(100))

plt.figure(figsize=(12,12))

for i in range(len(categoricalFeatures)):
    plt.subplot(6,1,i+1)
    sns.countplot(df_train[categoricalFeatures[i]])
plt.tight_layout();
```

Visualising Categorical Features:



❓ Question 6:

Discuss with your pair programming partner: There is less data in 2012 than the previous two years, did the sale drop from previous years? Does it affect what we see in the plots for month and week? Does the plot below clarify it to some degree?

In [40]:
```
plt.figure(figsize=(12, 6))
sns.lineplot(data=df_train, x="week", y="Weekly_Sales",  style='year');
```

Question 6

Even though we have less data for 2012, it does not affect the sales assumptions because as shown in the graph above, 2012 sales follows the same path as in previous years. This could be the case because we are missing some end parts of 2012 data as well.

Task III: Impact from Holidays

The first business objective is to understand the impact of holidays on weekly sales.

There is a flag provided for us: `IsHoliday` , let's calculate the average weekly sales for holiday weeks and non-holiday weeks, respectively. For this, we will use `.groupBy` and `.mean()` . Are holiday sales higher?

In [41]:
```
df_train.groupby('IsHoliday').mean()[[target]]
```

Out[41]:

IsHoliday	Weekly_Sales
False	21,756.05
True	23,737.05

Expected Output

But we would like to understand it at more granular level, remember Simpson's paradox (https://en.wikipedia.org/wiki/Simpson's_paradox)? To save some time, date mapping are identified for the training data

- Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12
- Labor Day: 10-Sep-10, 9-Sep-11, 7-Sep-12
- Thanksgiving: 26-Nov-10, 25-Nov-11
- Christmas: 31-Dec-10, 30-Dec-11

We create a flag for each holiday to help you analyze weekly sale by each holiday type

In [42]:
```
superbowl_mask = df_train['Date'].isin(['2010-02-12', '2011-02-11', '2012-0
laborday_mask = df_train['Date'].isin(['2010-09-10', '2011-09-09','2012-09-
thanksgiving_mask = df_train['Date'].isin(['2010-11-26', '2011-11-25'])
christmas_mask = df_train['Date'].isin(['2010-12-31', '2011-12-30'])
```

In [43]:
```
df_train['superbowl'] = superbowl_mask
df_train['laborday'] = laborday_mask
df_train['thanksgiving'] =thanksgiving_mask
df_train['christmas'] = christmas_mask
```

Run the next cell to see 1) how many weekly sales fell on Christmas (does it make sense? what did we not account for?) 2) what is the average weekly sales stratified by whether it is Christmas week or not?

In [44]:
```
df_train.groupby(['christmas'])\
        .agg(count = ('christmas', 'size'),
             avg_weekly_sales= ('Weekly_Sales','mean'))
```

Out[44]:

christmas	count	avg_weekly_sales
False	8057	21,921.06
True	125	20,565.56

Perform the same for the other three holidays:

In [45]:
```
holidays = ['superbowl', 'laborday', 'thanksgiving', 'christmas']
for holiday in holidays:
    summary_stats = df_train.groupby([holiday])\
            .agg(count = (holiday, 'size'),
                 avg_weekly_sales= ('Weekly_Sales','mean'))
    print(summary_stats)
    print()
```

```
           count  avg_weekly_sales
superbowl
False       8001         21,845.80
True         181         24,311.98

          count  avg_weekly_sales
laborday
False      8007         21,884.35
True        175         22,632.78

              count  avg_weekly_sales
thanksgiving
False          8067         21,813.97
True            115         27,959.84

           count  avg_weekly_sales
christmas
False       8057         21,921.06
True         125         20,565.56
```

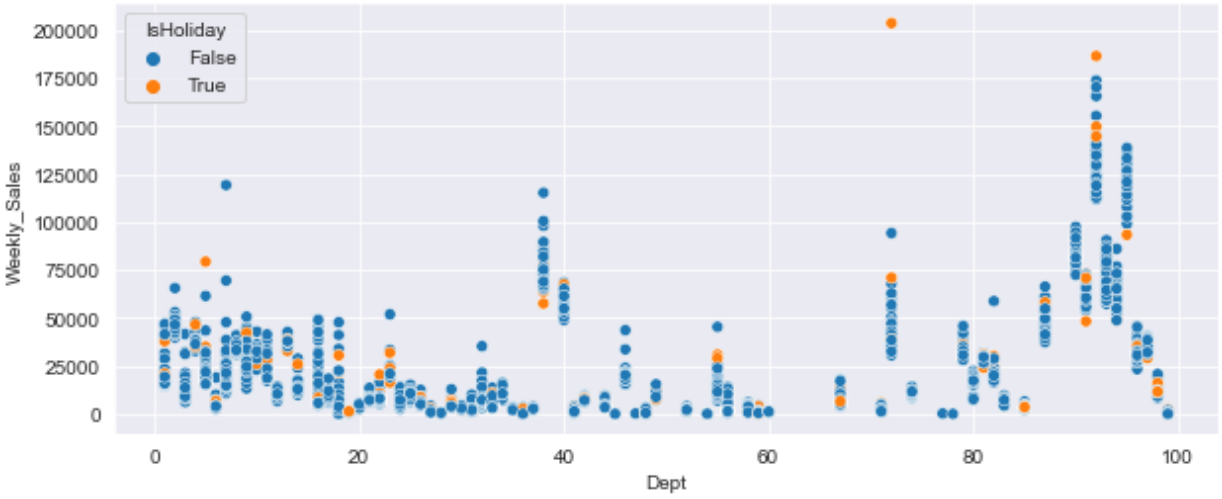
Expected Output

Without hypothesis testing and by only eyeballing, it seems like Super Bowl and Thanksgiving has a positive impact on the weekly sales for Store 1 in this training dataset. Discuss with your teammate, are you surprised that during Christmas, sales at Walmart do not go up? Holiday effect, if causal, happened most during Thanksgiving weeks, is this something you expected?

If causal, happened most during Thanksgiving weeks, is this something you expected?

We have been ignoring `Dept`, let's take a look at the plot below showing the weekly sales by department in 2011.

In [46]:
```
plt.figure(figsize=(10,4))
sns.scatterplot(data=df_train[df_train.year==2011], x = 'Dept', y= target,
```



Dept 72 has a very unusual high weekly sales during the holiday week, but we will need more data to understand if this is data issue, outlier, or special event.

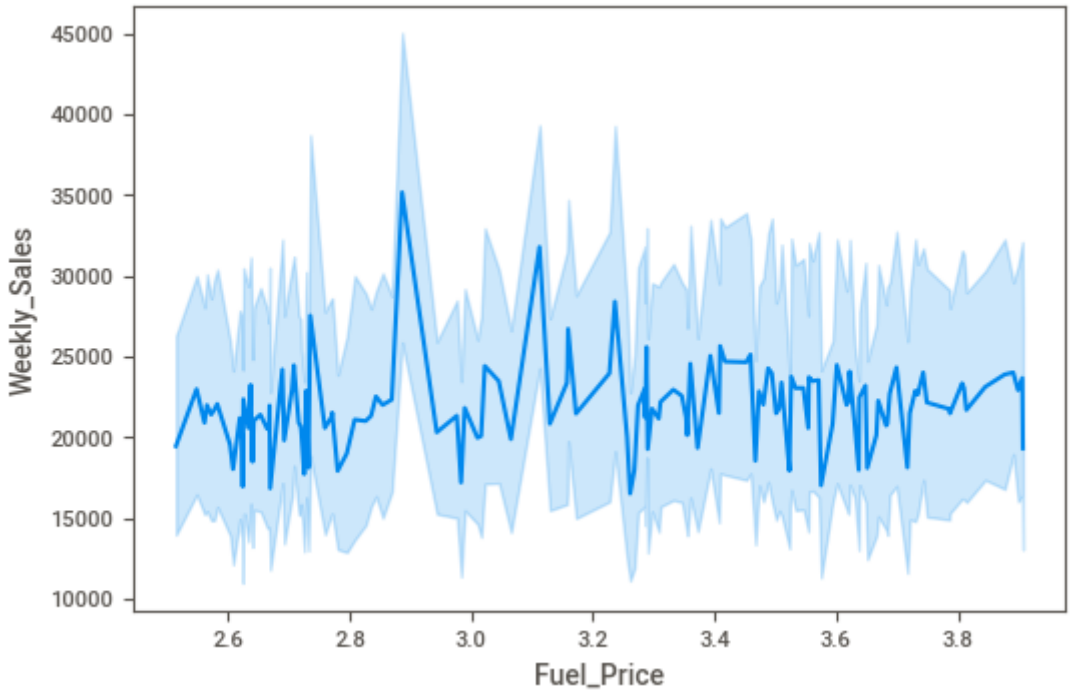
Task IV: Visualize Relationship between Macroeconomic & External Factors and Sales

In [127]:
```
sns.lineplot(data=df_train, x="Fuel_Price", y="Weekly_Sales");
df[["Fuel_Price","Weekly_Sales"]].corr()
```

Out[127]: <AxesSubplot:xlabel='Fuel_Price', ylabel='Weekly_Sales'>

Out[127]:

	Fuel_Price	Weekly_Sales
Fuel_Price	1.00	0.01
Weekly_Sales	0.01	1.00

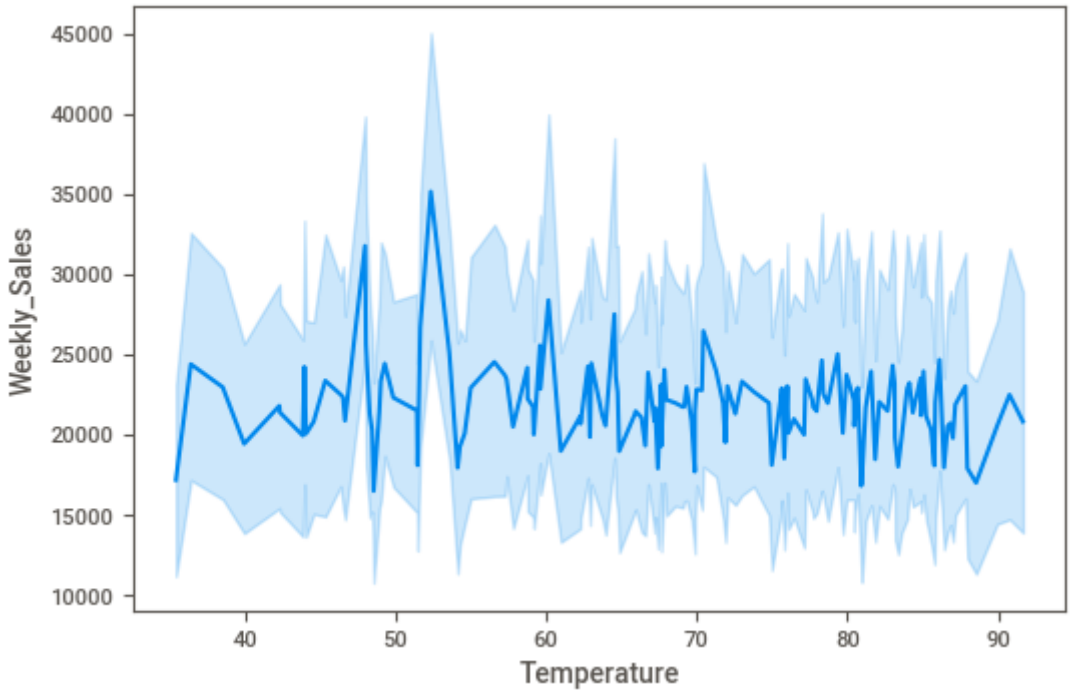


In [128]:
```
sns.lineplot(data=df_train, x="Temperature", y="Weekly_Sales");
df[["Temperature","Weekly_Sales"]].corr()
```

Out[128]:
```
<AxesSubplot:xlabel='Temperature', ylabel='Weekly_Sales'>
```

Out[128]:

	Temperature	Weekly_Sales
Temperature	1.00	-0.01
Weekly_Sales	-0.01	1.00

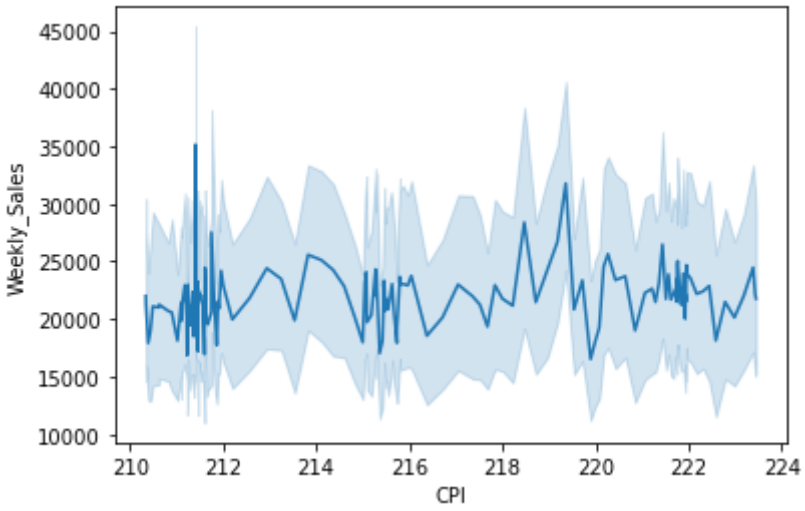


In [82]:
```
sns.lineplot(data=df_train, x="CPI", y="Weekly_Sales");
df[["CPI","Weekly_Sales"]].corr()
```

Out[82]: <AxesSubplot:xlabel='CPI', ylabel='Weekly_Sales'>

Out[82]:

	CPI	Weekly_Sales
CPI	1.00	0.02
Weekly_Sales	0.02	1.00

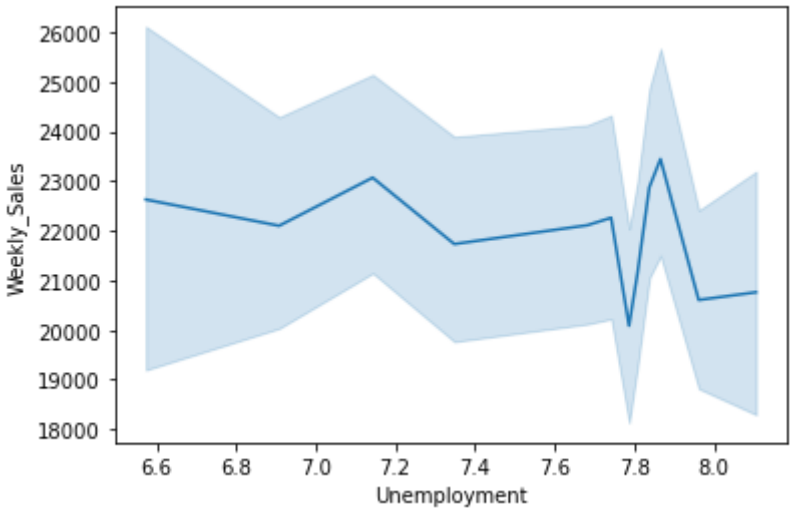


In [83]:
```
sns.lineplot(data=df_train, x="Unemployment", y="Weekly_Sales");
df[["Unemployment","Weekly_Sales"]].corr()
```

Out[83]: <AxesSubplot:xlabel='Unemployment', ylabel='Weekly_Sales'>

Out[83]:

	Unemployment	Weekly_Sales
Unemployment	1.00	-0.01
Weekly_Sales	-0.01	1.00



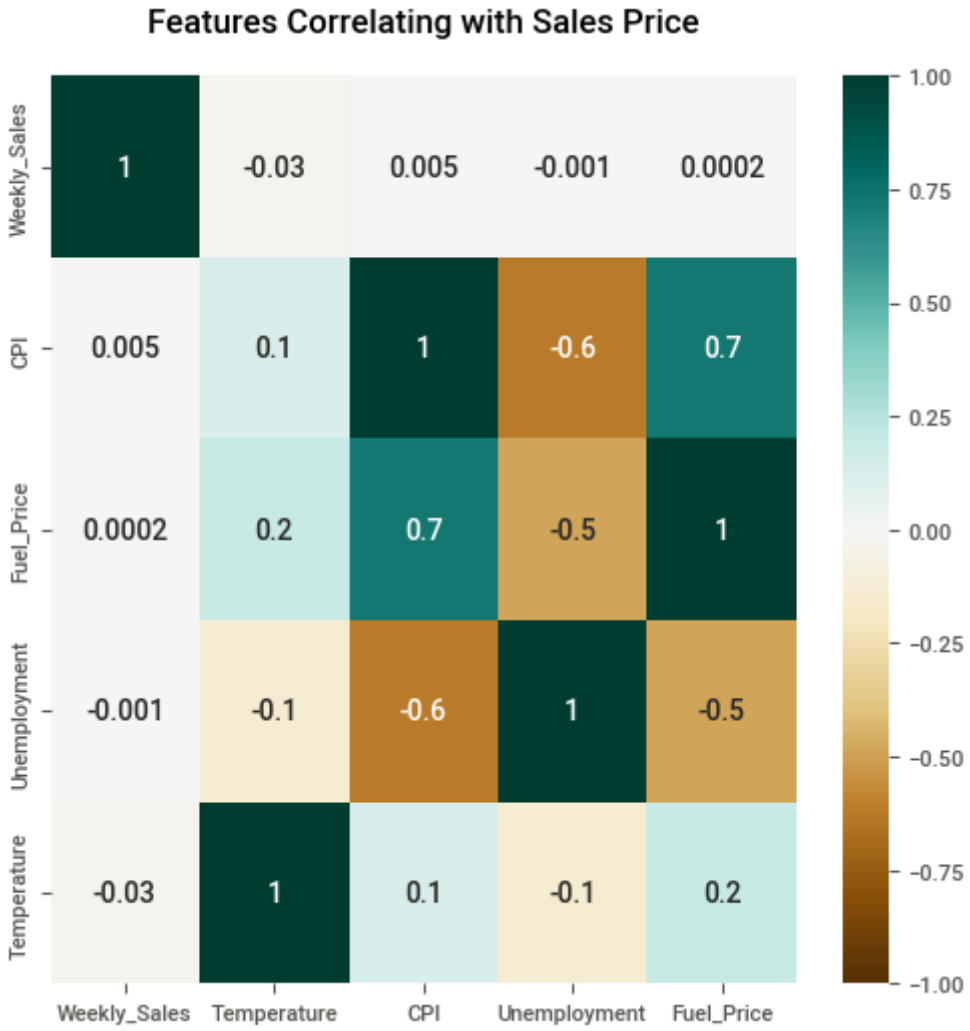
By eyeballing, do you find strong evidence that those are correlated with Walmart's weekly sales? Do you think `lineplot` is an appropriate plot for this?

Question

> The `lineplot` does not show much correlation. We do not like using lineplots so use `.corr` method. No pearson correlations here.

Lastly, we calculate the spearman correlations among target and external factors and verify that there is no strong linear correlation between the target variable and these features.

In [129]:
```
plt.figure(figsize=(6, 6))
df_train_reduced = df_train[[target] + external_factors]
corr = df_train_reduced.corr(method='spearman')
heatmap = sns.heatmap(corr.sort_values(by=target, ascending=False),
                      vmin=-1, vmax=1, annot=True, fmt='.1g', cmap='BrBG')
heatmap.set_title('Features Correlating with Sales Price', fontdict={'fonts
```



Task V: Feature Engineering

"*Feature Engineering encapsulates various data engineering techniques such as selecting relevant features, handling missing data, encoding the data, and normalizing it. It is one of the most crucial tasks and plays a major role in determining the outcome of a model.*" Ref (https://www.analyticsvidhya.com/blog/2021/10/a-beginners-guide-to-feature-engineering-everything-you-need-to-know/).

One part of feature engineering is to create new features from given data, like `thanksgiving` column earlier was derived from `Date`. Common techniques for tabular data include to add summary statistics of the numerical features such as mean and standard deviation, to create new features from the interaction of multiple features, etc. In this task, however, we will work on handling missing data, normalizing numerical features, and encoding categorical features.

First, missing data. Missing value treatment is crucial, yet not trivial. Take a read on Tackling Missing Value in Dataset (https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/) for detailed explanation. Features with nulls or wrong values (e.g., negative fuel price) needs to be imputed or removed.

- Do you want to keep the features with missing value? Discuss the trade offs
- If answer to the first question is yes, then how do you want to impute them? Discuss the trade offs

From ealier steps, we observed that only the markdown columns contain missing values, yet we do not have more information on what those values are for.

In [85]:
```
df_train.columns[df_train.isna().sum() != 0]
```

Out[85]:
```
Index(['MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4', 'MarkDown5'],
dtype='object')
```

For each column, find out the percentage of the data is missing

In [87]:
```
md_cols = ['MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4', 'MarkDown5']
for col in ['MarkDown'+str(i) for i in range(1,6)]:
    perc_missing =  df_train[col].isna().sum() / len(df_train)
    print (f'{col}: {perc_missing:.0%} is missing')
```

```
MarkDown1: 64% is missing
MarkDown2: 70% is missing
MarkDown3: 65% is missing
MarkDown4: 64% is missing
MarkDown5: 64% is missing
```

Expected Output

The majority of the markdown fields are missing. This is where, again, we need to communicate with the stakeholders to understand what the data measure, how the data was collected and then determine our strategy from there. Since we want to understand the impacts of `MarkDownx` on weekly sales, we will keep the features and impute the missing values. We have learned that there are tradeoffs with how we treat missing values and that our choice of imputation can be significantly impacted by extreme values and the amount of the missing data. We choose to impute with the median here to mitigate these negative impacts. Use `.fillna()` to impute the missing values.

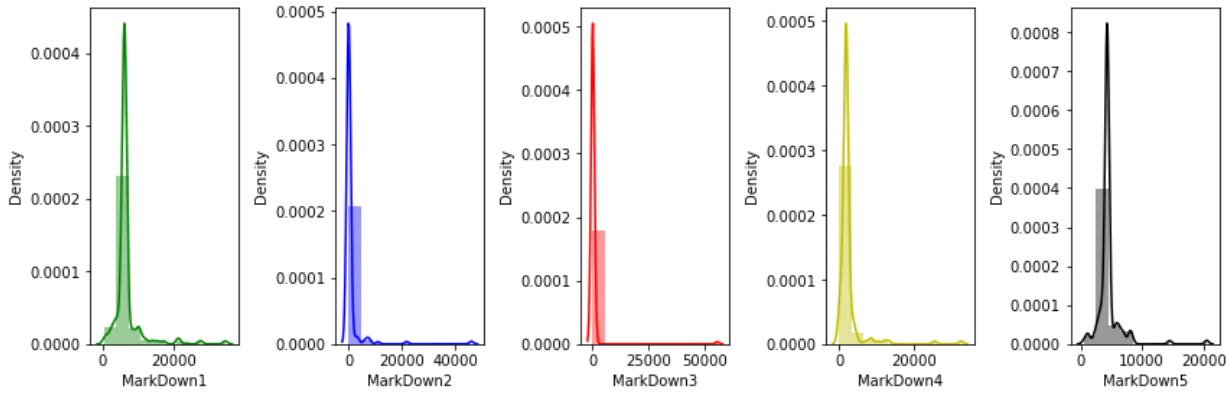
In [91]:
```
for col in ['MarkDown'+str(i) for i in range(1,6)]:
    df_train[col].fillna(df_train[col].median(), inplace=True)
```

In [92]:
```
(df_train.isna().sum() != 0).sum() # sanity check: 0.
```

Out[92]: 0

Visualize the distributions for those markdown fields after imputations, are they normal?

In [93]:
```
plt.figure(figsize=figsize)
for i in range(len(md_cols)):
    plt.subplot(1,len(md_cols),i+1)
    sns.distplot(df_train[md_cols[i]],
                 hist_kws=dict(linewidth=2),
                 bins=10,
                 color = colors[i])
plt.tight_layout();
```



Note that missing values are different from outliers. Outliers, on the other hand, are feature values that are rare in nature. They can unncessarily skew the data and causes problem for modeling. Outlier treatment involves removing or imputing such values. One popular approach to identify outliers is IQR; that is, data points that lie 1.5 times of IQR above Q3 (third quartile) and below Q1 (first quartile) are outliers. Take a read on Detecting and Treating Outliers (https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/). We will leave it as an optional exercise for you to identify outliers using IQR, and replace the outliers with the median.

Now let's see how we normalize the data. For numerical features it means scaling the features to be of similar range. This step is crucial for machine learning algorithms that calculate distances between data (e.g., read The Importance of Feature Scaling (https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html).

For this task, of the external features, let's keep Temperature since it is the most linearly correlated with the target variable, though very weak and negative (feature selection). In addition, we include one markdown field. Since neither seems to follow normal distributions, it is safer to use `MinMaxScaler` from `sklearn.preprocessing` to transform features by scaling each feature

to a given range (See discussion on Normalization vs Standardization (https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/))

In [94]:
```
from sklearn.preprocessing import MinMaxScaler

numericalFeatures = ['Temperature', 'MarkDown1']
df_train_num = df_train[numericalFeatures]
```

In [95]:
```
df_train_num.describe() # Check the summary statistics
```

Out[95]:

	Temperature	MarkDown1
count	8,182.00	8,182.00
mean	68.19	6,828.42
std	14.16	3,981.30
min	35.40	410.31
25%	57.79	6,154.14
50%	69.64	6,154.14
75%	80.48	6,154.14
max	91.65	34,577.06

Instantiate a MinMaxScaler and fit using `df_train_num` :

In [98]:
```
scaler = MinMaxScaler()
scaler.fit(df_train_num)
```

Out[98]:
```
MinMaxScaler()
```

Now transform training data `df_train_num` and store the resulting nparray in `train_norm` :

In [99]:
```
train_norm = scaler.transform(df_train_num)
```

Verify that both columns now have minimum 0 and maximum 1.

In [100]:
```
pd.DataFrame(train_norm, columns=df_train_num.columns).describe()
```

Out[100]:

	Temperature	MarkDown1
count	8,182.00	8,182.00
mean	0.58	0.19
std	0.25	0.12
min	0.00	0.00
25%	0.40	0.17
50%	0.61	0.17
75%	0.80	0.17
max	1.00	1.00

In [59]:
```
# Expected Output:
```

Out[59]:

	Temperature	MarkDown1
count	8,182.00	8,182.00
mean	0.58	0.19
std	0.25	0.12
min	0.00	0.00
25%	0.40	0.17
50%	0.61	0.17
75%	0.80	0.17
max	1.00	1.00

Question

> Yes it scales from 0 to 1 as expected.

Let's turn to categorical fatures. So far most, if not all Python packages for modeling do not accept strings as input; thus encoding the categorical value to numerical value is a necessary step. Here, let's apply [one-hot encoding (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html)](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) on `Dept` and `IsHoliday`:

In [101]:
```
from sklearn.preprocessing import OneHotEncoder
categoricalFeatures = ['Dept', 'IsHoliday']
df_train_cat = df_train[categoricalFeatures]
ohe = OneHotEncoder(handle_unknown='ignore',sparse = False).fit(df_train_ca
```

Transform the categorical features using one hote encoding `ohe` .

In [103]:
```
train_ohe = ohe.transform(df_train_cat)
```

In [104]:
```
train_ohe.shape, df_train_cat.shape # Expected Output: ((8182, 79), (8182,
```

Out[104]:
```
((8182, 79), (8182, 2))
```

The number of columns explodes from 2 to 79.

Lastly we merge the processed numerical features with the processed categorical features using `hstack` in `numpy` :

In [105]:
```
import numpy as np
X_train = np.hstack([train_norm, train_ohe])
```

In [106]:
```
X_train.shape # sanity check: (8182, 81)
```

Out[106]:
```
(8182, 81)
```

What about the test data? Yes you need to apply the same treatments. We spare some copy + paste + edit and see how this can be done when we introduce `pipeline` next.

Task VI: Pipeline

Even with less than 20 features in our dataset, there are many many possibilities that you can preprocessing the data. There is no one-fits-all approach; often you will find yourself experimenting with many combinations to achieve better modelling performance: Should I apply normalization or standardization? Do I remove the outliers or should I impute them? Do I impute the missing values with median or mean or 0? Answers to many of these questions is "It depends." (Have you heard Graduate Student Descent (https://sciencedryad.wordpress.com/2014/01/25/grad-student-descent/)?) That means trial-and-error and it is not efficient to produce a notebook each time when you need to try something slightly different. You will get lost quickly. Pipeline is one useful tool.

Not only does Pipeline help streamline the process, keep the code modular, but also reduces the possibility of introducing errors/bugs. In this task, we build the pipeline following the strategies used in the last task, run a simple linear regression model, and print out the model's performance. Note there is minimal code required for you to implement, the key is to understand each step.

To avoid confusion, let's read the data again directly from `train-store1.csv` .

In [107]:
```
df = pd.read_csv('../dat/train-store1.csv')
```

In [108]:
```
df.shape
```

Out[108]:
```
(10244, 16)
```

Separating the target `y` from the features `X` :

In [109]:
```
X, y = df.drop(columns=target), df[target]
```

Import `Pipeline` from submodule `sklearn.pipeline`

In [110]:
```
from sklearn.pipeline import Pipeline
```

Now we build a transformer for numerical features following two steps: impute the missing values with the feature median (use `SimpleImputer`), followed by normalization (use `MinMaxScaler`)

In [111]:
```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
numeric_features = ['CPI', 'MarkDown1']
numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", MinMaxScaler())
])
```

For categorical features, we apply one hot encoding `OneHotEncoder` (there are many other options; see Scikit-learn documentation (https://scikit-learn.org/stable/modules/preprocessing.html#encoding-categorical-features)):

In [112]:
```
categorical_features = ['Dept', 'IsHoliday']
categorical_transformer = OneHotEncoder(handle_unknown='ignore')
```

Piece the `numeric_transformer` and `categorical_transformer` using `ColumnTransformer` :

In [113]:
```
from sklearn.compose import ColumnTransformer

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)
```

Lastly, let's append the regression model to preprocessing pipeline to complete a full prediction pipeline.

In [114]:
```
from sklearn.linear_model import LinearRegression

model = Pipeline(
    steps=[("preprocessor", preprocessor), ("model", LinearRegression())]
)
```

The pipepline has been built! The rest is to

- split the data into training and testing sets
- apply the pipeline to the training data
- obtain the prediction performance on testing data

In [115]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

Let's run the prediction!

In [116]:
```
model.fit(X_train, y_train)
```

Out[116]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imput
er',
                                                                   Simple
Imputer(strategy='median')),
                                                                  ('scale
r',
                                                                   MinMax
Scaler())]),
                                                  ['CPI', 'MarkDown1']),
                                                 ('cat',
                                                  OneHotEncoder(handle_un
known='ignore'),
                                                  ['Dept', 'IsHolida
y'])])),
                ('model', LinearRegression())])
```

In [117]:
```
print("model score: %.3f" % model.score(X_test, y_test))
```

```
model score: 0.949
```

Optional: Discuss what type of Feature Selection (https://scikit-learn.org/stable/modules/feature_selection.html#feature-selection) strategy you would use to select the features.

Automating EDA

In this exercise, you have learned the manual way to perform EDA. Doing EDA manually has the benefits of customization, but is also highly repetitive. For this reason, a lot of EDA can easily be automated! In automating our EDA, we can get to know our data more quickly and spend more time on feature engineering and modeling. Let's check out a library called SweetViz (https://github.com/fbdesignpro/sweetviz) to see how we can automate EDA!

In [118]:
```
import sweetviz as sv

orig_data_report = sv.analyze(df)
orig_data_report.show_notebook()
```

```
                                             |          | [  0%]   00:00 ->
(? left)
```

1. Click on a feature to tab to explore the feature in more detail.
2. Notice that `SweetViz` calculates the descriptive stats for each feature, along with its missing and duplicate value stats.
3. Notice that `SweetViz` helps to detect numerical vs categorical datatypes.
4. Click on the `ASSOCIATIONS` tab to explore associations/correlations!

Prefer a browswer experience?

In [119]:
```
orig_data_report.show_html('orig_data_report.html', open_browser=True)
```

```
Report orig_data_report.html was generated! NOTEBOOK/COLAB USERS: the web
browser MAY not pop up, regardless, the report IS saved in your notebook/
colab files.
```

Now let's have a look at a comparison report of our train and test datasets!

In [120]:
```
compare_report = sv.compare([X_train, 'Train'], [X_test, 'Test'])
compare_report.show_notebook()
```

```
                                                |          | [   0%]   00:00 ->
(? left)
```

Assignment Questions

Write down three lessons learned about exploratory data analysis, feature engineering, sales data modeling, or time-series data modeling.

- SciKit-Learn Pipelines! This is something new to me, and I enjoyed learning about it with my pairing partner. I am going to try it in a real project as soon as I am done with this assignment :)
- Creating a test subset BEFORE even starting the analysis. I am usually doing it a bit later, but I see why it is a good practice.
- SimpleImputer. I did not know this.

Write down three lessons not yet learned about exploratory data analysis, feature engineering, sales data modeling, or time-series data modeling.

- I would like to learn how to use pipelines more effectively and the techniques shown in this class.
- I want to see how I can take the lessons from this class and apply to my work where I look at financial market data.
- I would love to see how I can automate all these processes!

Write down three advantages to using Automated EDA tools such as SweetViz, compared to a manual approach?

- Speed. You do not have to implement everything from scratch
- Less error-prone. No need to implement everything from scratch means fewer errors.
- Completeness. Even though the analysis is very rudimentary, you may be sure that you won't forget any feature.

Write down three disadvantages/drawbacks to using automated EDA tools such as SweetViz, compared to a manual approach?

- High-dimensional data. With tons of features, you cannot really use it.
- Basics only. SweetViz is limited to basic data digging only, so it will not replace the in-depth analysis.
- Data understanding level. Implementing EDA yourself gives more in-depth insight than just looking at charts.

Note

- EDA, like other parts of machine learning, is an iterative process, NOT linear.
- This analysis is far from being comprehensive; rather it is a starting point.
- There does not exist one "standard" way to perform EDA. You should always keep business objectives in mind and perform analysis as seen fit. It is one of those skills that grows with lots of practices.

References

1. Original dataset is from kaggle: wallmart sales forecast datasets (https://www.kaggle.com/datasets/iamprateek/wallmart-sales-forecast-datasets)
2. Notebook: craking the walmart sales forecasting challenge (https://www.kaggle.com/code/fernandol/cracking-the-walmart-sales-forecasting-challenge)