1: Importing the Relevant Libraries

In [2]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression # as baseline model

import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')
```

2. Data Observation

2.1 Importing train and test data

In [3]:
```
train_df = pd.read_csv('TRAIN.csv')

train_df.head()
```

Out[3]:

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales
0	T1000001	1	S1	L3	R1	2018-01-01	1	Yes	9	7011.84
1	T1000002	253	S4	L2	R1	2018-01-01	1	Yes	60	51789.12
2	T1000003	252	S3	L2	R1	2018-01-01	1	Yes	42	36868.20
3	T1000004	251	S2	L3	R1	2018-01-01	1	Yes	23	19715.16
4	T1000005	250	S2	L3	R4	2018-01-01	1	Yes	62	45614.52

In [4]:
```
test_df = pd.read_csv('TEST_FINAL.csv')

test_df.head()
```

Out[4]:

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount
0	T1188341	171	S4	L2	R3	2019-06-01	0	No
1	T1188342	172	S1	L1	R1	2019-06-01	0	No
2	T1188343	173	S4	L2	R1	2019-06-01	0	No
3	T1188344	174	S1	L1	R4	2019-06-01	0	No
4	T1188345	170	S1	L1	R2	2019-06-01	0	No

2.2 Observing rows(or datapoints)

In [5]:
```
train_df.shape, test_df.shape
```

Out[5]:
```
((188340, 10), (22265, 8))
```

- Train data contains 17 months of data and Test contains 2 months of data.

In [6]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188340 entries, 0 to 188339
Data columns (total 10 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   ID             188340 non-null  object
 1   Store_id       188340 non-null  int64
 2   Store_Type     188340 non-null  object
 3   Location_Type  188340 non-null  object
 4   Region_Code    188340 non-null  object
 5   Date           188340 non-null  object
 6   Holiday        188340 non-null  int64
 7   Discount       188340 non-null  object
 8   #Order         188340 non-null  int64
 9   Sales          188340 non-null  float64
dtypes: float64(1), int64(3), object(6)
memory usage: 14.4+ MB
```

In [7]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22265 entries, 0 to 22264
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   ID             22265 non-null  object
 1   Store_id       22265 non-null  int64
 2   Store_Type     22265 non-null  object
 3   Location_Type  22265 non-null  object
 4   Region_Code    22265 non-null  object
 5   Date           22265 non-null  object
 6   Holiday        22265 non-null  int64
 7   Discount       22265 non-null  object
dtypes: int64(2), object(6)
memory usage: 1.4+ MB
```

- There are no null values in train and test data.

2.3 Observing Columns(or features)

- Out of all the columns(features) in train data, ID is should not be used, Sales is the feature that we have to predict, so test dataset does not contain Sales.
- There is a feature named #Order, which is highly correlated with Sales, and the Test data doesn't have this feature, and I believe it is not useful because we can't know the Orders of test data, and also observing its collinearity with Sales feature I have ignored this feature.

Store_id : categorical feature.

In [8]:
```
print(train_df['Store_Type'].value_counts())
train_df.groupby('Store_Type')['Sales'].mean()
```

```
S1    88752
S4    45924
S2    28896
S3    24768
Name: Store_Type, dtype: int64
```

Out[8]:
```
Store_Type
S1    37676.511694
S2    27530.828222
S3    47063.068209
S4    59945.685926
Name: Sales, dtype: float64
```

- more no. of stores are type S1, followed by S4
- for S5 the average sales are high, and for S2 the average sales are low

Location_Type : Categorical feature

In [9]:
```
print(train_df['Location_Type'].value_counts())
train_df.groupby('Location_Type')['Sales'].mean()
```

```
L1    85140
L2    48504
L3    29928
L5    13932
L4    10836
Name: Location_Type, dtype: int64
```

Out[9]:
```
Location_Type
L1    41453.597889
L2    59231.480373
L3    33072.257756
L4    29067.414313
L5    25187.787261
Name: Sales, dtype: float64
```

- more no. of stores are type L1, followed by L2
- for L2 the average sales are high, and for L5 the average sales are low

Region_Code : Categorical feature

In [10]:
```
print(train_df['Region_Code'].value_counts())
train_df.groupby('Region_Code')['Sales'].mean()
```

```
R1    63984
R2    54180
R3    44376
R4    25800
Name: Region_Code, dtype: int64
```

Out[10]:
```
Region_Code
R1    46765.488405
R2    40054.847344
R3    42144.517063
R4    39743.434249
Name: Sales, dtype: float64
```

- more no. of stores are type R1, followed by R2
- the average sales of all the region_codes is significantly same.

Holiday : Categorical feature

In [11]:
```
print(train_df['Holiday'].value_counts())
train_df.groupby('Holiday')['Sales'].mean()
```

```
0    163520
1     24820
Name: Holiday, dtype: int64
```

Out[11]:
```
Holiday
0    43897.288998
1    35451.878930
Name: Sales, dtype: float64
```

- the sales are high in the days of non-holidays.

- Discount : Categorical feature

In [12]:
```
print(train_df['Discount'].value_counts())
train_df.groupby('Discount')['Sales'].mean()
```

```
No     104051
Yes     84289
Name: Discount, dtype: int64
```

Out[12]:
```
Discount
No     37403.679678
Yes    49426.497620
Name: Sales, dtype: float64
```

- the sales are high when there is discount.

#Observations from Categorical features :

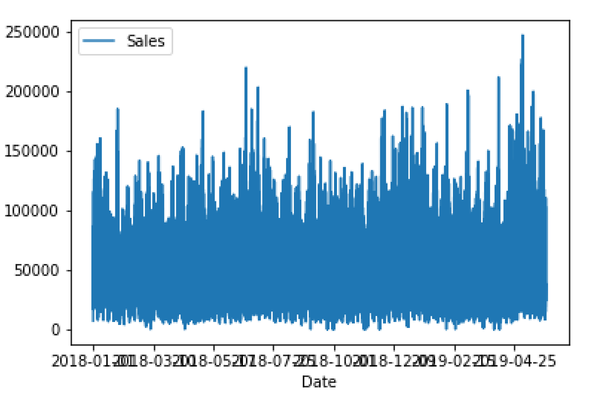
- All the above features() are categorical, because they don't have continous values.
- All these features are useful because they are distinct from one other and each distinct value in a particular categorical feature are different, each feature add value in taking a decision.(None of them are highly correlated).
- I believe tree based algorithms work well for this problem, because almost all of the features are categorical which involves in conditional prediction rather than complex math calculation.

Date : String

- not useful as categorical feature, ther are many dates.
- we might take the timestramp as numerical value, but the value increses as date goes to future, and tree based algoriths doesn't predict better results when the test value is grater than max *of train value(same for min* values).
- So, we have to create new features out of date and test their importance in modelling.

In [13]:
```
train_df.plot(x='Date', y='Sales')
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2758bb0d0f0>
```



- we will see later how the date feature is useful in featurization part.

3. Featurization

3.1 Converting categorical features to numerical.

- Store_Type, Location_Type, Region_Code, and Discount are passed to Location Encoder function of sklearn preprocessing module that will return the numerical labels.
- I have also tried One-hot-encoding, but got same results.

3.2 numerical features

- there isn't much to do with numberical features in our dataset, becuase the are not continues values, they are also like lables, the Sales and #Order features are continous but those are not used for training.

- At this point I have split the training data into training and cross validation and I have used linear regression(simple regression model), and I have applied the model on total training data and final test data, and submitted the results to hackathon, and I have made it my base model.
- I have n't normalized or standardized the data because there are no continous or pure useful numerical features. and If I decide to use tree based algorithms there is no need of normalization or standardization.
- after this attemt I have worked on featurization and modelling parallely.

3.3 creating new features from Date Feature

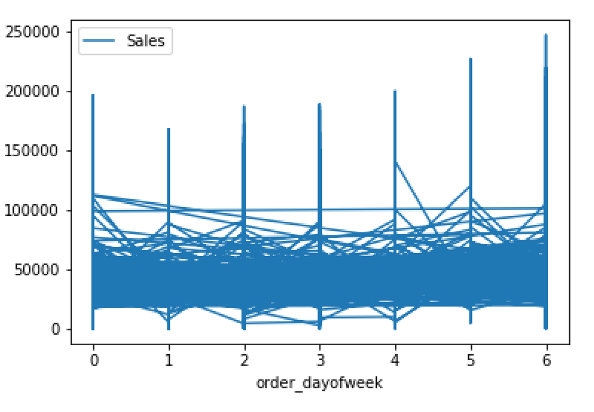
- let's create features like day, day in week, month, year and check the importance.

In [14]:
```
train_order_date = train_df['Date']
train_df['Date'] = pd.to_datetime(train_df['Date'])

train_df['order_year'] = train_df['Date'].dt.year
train_df['order_month'] = train_df['Date'].dt.month
train_df['order_week'] = train_df['Date'].dt.week
train_df['order_day'] = train_df['Date'].dt.day
train_df['order_dayofweek'] = train_df['Date'].dt.dayofweek
```

In [15]:
```
train_df.plot(x='order_dayofweek', y='Sales')
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2758bb536d8>
```



- out of all, the dayofweek values seemed to be useful, I have tried with all the other features but didn't added much value.
- I will use this feature transform if a different way, like all the values form 0-4 as 0 and 5-6 as 1, calling is week_end.
- I have read this from Analytics_vidhya blog, that if the no.of decisions are less the decision tree will make decisions faster and the chance of overfitting also less.

3.4 Replacing zeroes in the Sales column

- replacing the zeroes in sales column with mean values of sales improved the metric(very small improvement), I have also tried like all the zero value data points as test set and the remaining data as train_data and predict better values instead of zeroes, but replacing zeroes with the predicted values also doesn't improve the metric.

4 Building Model

- I ahve decided do use tree based algoriths beacuse all the fatures that are going to be trained are categorical(and many of them have less categories, less chance of overfit) so the tree based models make good decisions.
- So, starting from Decision Tree regressor with basic hyper parameter tuning, I have used Random Forest regressor, and Xgboost regressor.'
- Modelling went strainght forward, I have know basic working of tree based models, so I have used the models, tuned the important parameters for the above mentioned models.
- finally with some parameter tuning, and required featurization I have got decent score.

5 Evaluation

- I have split the train data into X_train and X_test(as cross validation data), and I have calculated the msle*1000 for every predicted set of values with Y_test values. This helped me understand the correctness of model, based on that I decide to submit my results in hackathon.
- I also used to compare my final test_prediction(between main total_data and final_test data) with my old_submitted values, to know whether there is significant difference or not, if not I didn't submit the solution, I will work again.