In [ ]:
```python
# Importing libraries
```

In [55]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import dates
from datetime import datetime
import sklearn
import seaborn as sns
```

In [ ]:
```python
## Importing Data
```

In [3]:
```python
data=pd.read_excel("E:\data_excel.xlsx")
```

In [4]:
```python
data
```

Out[4]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	2010-05-02 00:00:00	1643690.90	0	42.31	2.572	211.096358	8.106
1	1	2010-12-02 00:00:00	1641957.44	1	38.51	2.548	211.242170	8.106
2	1	19-02-2010	1611968.17	0	39.93	2.514	211.289143	8.106
3	1	26-02-2010	1409727.59	0	46.63	2.561	211.319643	8.106
4	1	2010-05-03 00:00:00	1554806.68	0	46.50	2.625	211.350143	8.106
...	...	...	...	...	...	...	...	...
6430	45	28-09-2012	713173.95	0	64.88	3.997	192.013558	8.684
6431	45	2012-05-10 00:00:00	733455.07	0	64.89	3.985	192.170412	8.667
6432	45	2012-12-10 00:00:00	734464.36	0	54.47	4.000	192.327265	8.667
6433	45	19-10-2012	718125.53	0	56.47	3.969	192.330854	8.667
6434	45	26-10-2012	760281.43	0	58.85	3.882	192.308899	8.667

6435 rows × 8 columns

Changing dates into days by using library datetime

In [5]:
```python
data['Date'] =  pd.to_datetime(data['Date'])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Date          6435 non-null   datetime64[ns]
 2   Weekly_Sales  6435 non-null   float64
 3   Holiday_Flag  6435 non-null   int64
 4   Temperature   6435 non-null   float64
 5   Fuel_Price    6435 non-null   float64
 6   CPI           6435 non-null   float64
 7   Unemployment  6435 non-null   float64
dtypes: datetime64[ns](1), float64(5), int64(2)
memory usage: 402.3 KB
```

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

Out[6]:
```
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64
```

In [7]:
```python
data["Day"]= pd.DatetimeIndex(data['Date']).day
data['Month'] = pd.DatetimeIndex(data['Date']).month
data['Year'] = pd.DatetimeIndex(data['Date']).year
data
```

Out[7]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Day	Month	Year
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106	2	5	2010
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106	2	12	2010
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106	19	2	2010
3	1	2010-02-26	1409727.59	0	46.63	2.561	211.319643	8.106	26	2	2010
4	1	2010-05-03	1554806.68	0	46.50	2.625	211.350143	8.106	3	5	2010
...	...	...	...	...	...	...	...	...	...	...	...
6430	45	2012-09-28	713173.95	0	64.88	3.997	192.013558	8.684	28	9	2012
6431	45	2012-05-10	733455.07	0	64.89	3.985	192.170412	8.667	10	5	2012
6432	45	2012-12-10	734464.36	0	54.47	4.000	192.327265	8.667	10	12	2012
6433	45	2012-10-19	718125.53	0	56.47	3.969	192.330854	8.667	19	10	2012
6434	45	2012-10-26	760281.43	0	58.85	3.882	192.308899	8.667	26	10	2012

6435 rows × 11 columns

In [9]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Date          6435 non-null   datetime64[ns]
 2   Weekly_Sales  6435 non-null   float64
 3   Holiday_Flag  6435 non-null   int64
 4   Temperature   6435 non-null   float64
 5   Fuel_Price    6435 non-null   float64
 6   CPI           6435 non-null   float64
 7   Unemployment  6435 non-null   float64
 8   Day           6435 non-null   int64
 9   Month         6435 non-null   int64
 10  Year          6435 non-null   int64
dtypes: datetime64[ns](1), float64(5), int64(5)
memory usage: 553.1 KB
```

Finding the Maximum Sales Of store

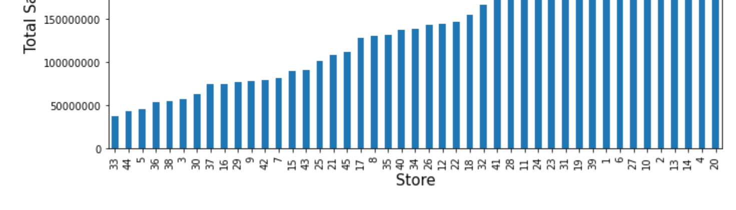
In [11]:
```python
total_sales=data.groupby(data['Store']).sum()['Weekly_Sales'].max()
```

In [12]:
```python
total_sales
```

Out[12]:
```
301397792.46
```

In [13]:
```python
total_sales= data.groupby('Store')['Weekly_Sales'].sum().sort_values()
total_sales_array = np.array(total_sales)
```

In [45]:
```python
plt.figure(figsize=(11,5))
plt.xticks(rotation=0)
plt.ticklabel_format(useOffset=False, style='plain', axis='y')
plt.title('Total sales for each store')
plt.xlabel('Store',fontsize=15)
plt.ylabel('Total Sales',fontsize=15)
total_sales.plot(kind='bar')
plt.show()
```



Maximum Standard Deviation of Store

In [16]:
```python
max_std = data.groupby('Store')['Weekly_Sales'].std().sort_values(ascending=False)
```

In [48]:
```python
max_std.max()
```

Out[48]:
```
317569.9494755081
```

In [49]:
```python
max_std.head(5)
```

Out[49]:
```
Store
14    317569.949476
10    302262.062504
20    275900.562742
4     266201.442297
13    265506.995776
Name: Weekly_Sales, dtype: float64
```

Finding Out The Coefficient Of Mean To Standard Deviation

In [18]:
```python
mean_1=data.groupby('Store')['Weekly_Sales'].mean().sort_values(ascending =True)
```

In [19]:
```python
mean_1.head(7)
```

Out[19]:
```
Store
33    259861.692028
44    302748.866014
5     318011.810490
36    373511.992797
38    385731.653287
3     402704.441049
30    438579.616224
Name: Weekly_Sales, dtype: float64
```

In [20]:
```python
coef = pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std() / data.groupby('Store')['Weekly_Sales'].mean())
coef = coef.rename(columns={'Weekly_Sales':'Coefficient of mean to standard deviation'})
coef_max = coef.sort_values(by='Coefficient of mean to standard deviation',ascending=False)
```

In [21]:
```python
coef_max.head(7)
```

Out[21]:

Store	Coefficient of mean to standard deviation
35	0.229681
7	0.197305
15	0.193384
29	0.183742
23	0.179721
21	0.170292
45	0.165613

Quarterly Growth Rate in Q3'2012

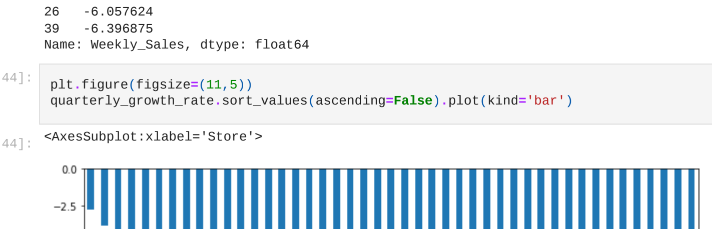
In [22]:
```python
quarter_2_sales = data[(data['Date'] >= '2012-04-01') & (data['Date'] <= '2012-06-30')].groupby('Store')['Weekly_Sales'].sum()
quarter_3_sales= data[(data['Date'] >= '2012-07-01') & (data['Date'] <= '2012-09-30')].groupby('Store')['Weekly_Sales'].sum()
```

In [23]:
```python
quarterly_growth_rate = ((quarter_3_sales - quarter_2_sales )/quarter_2_sales)*100
quarterly_growth_rate.sort_values(ascending=False).head()
```

Out[23]:
```
Store
16   -2.789294
7    -3.824738
35   -4.663086
26   -6.057624
39   -6.396875
Name: Weekly_Sales, dtype: float64
```

In [44]:
```python
plt.figure(figsize=(11,5))
quarterly_growth_rate.sort_values(ascending=False).plot(kind='bar')
```

Out[44]:
```
<AxesSubplot:xlabel='Store'>
```



Find out holidays which have higher sales than the mean sales in non-holiday season for all stores together

In [25]:
```python
Super_Bowl =['12-2-2010', '11-2-2011', '10-2-2012']
Labour_Day =  ['10-9-2010', '9-9-2011', '7-9-2012']
Thanksgiving =  ['26-11-2010', '25-11-2011', '23-11-2012']
Christmas = ['31-12-2010', '30-12-2011', '28-12-2012']
```

In [26]:
```python
Super_Bowl_Sales = (pd.DataFrame(data.loc[data.Date.isin(Super_Bowl)]))['Weekly_Sales'].mean()
Labour_Day_Sales = (pd.DataFrame(data.loc[data.Date.isin(Labour_Day)]))['Weekly_Sales'].mean()
Thanksgiving_Sales = (pd.DataFrame(data.loc[data.Date.isin(Thanksgiving)]))['Weekly_Sales'].mean()
Christmas_Sales = (pd.DataFrame(data.loc[data.Date.isin(Christmas)]))['Weekly_Sales'].mean()
Super_Bowl_Sales,Labour_Day_Sales,Thanksgiving_Sales,Christmas_Sales
```

Out[26]:
```
(1079127.9877037033, 1042427.2939259257, 1471273.427777778, 960833.1115555551)
```

In [31]:
```python
Mean_Sales = {'Super_Bowl_Sales' : Super_Bowl_Sales,
              'Labour_Day_Sales': Labour_Day_Sales,
              'Thanksgiving_Sales':Thanksgiving_Sales,
              'Christmas_Sales': Christmas_Sales,
              'Non_Holiday_Sales': Non_Holiday_Sales}
Mean_Sales
```

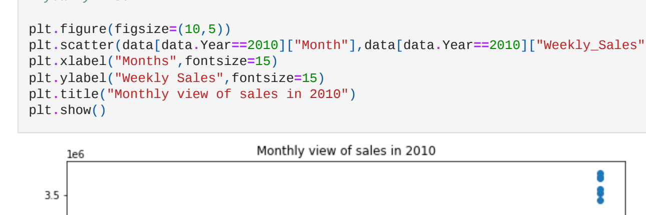
Out[31]:
```
{'Super_Bowl_Sales': 1079127.9877037033,
 'Labour_Day_Sales': 1042427.2939259257,
 'Thanksgiving_Sales': 1471273.427777778,
 'Christmas_Sales': 960833.1115555551,
 'Non_Holiday_Sales': 1041256.3802088564}
```

In [28]:
```python
Non_Holiday_Sales = data[data['Holiday_Flag'] == 0 ]['Weekly_Sales'].mean()
Non_Holiday_Sales
```

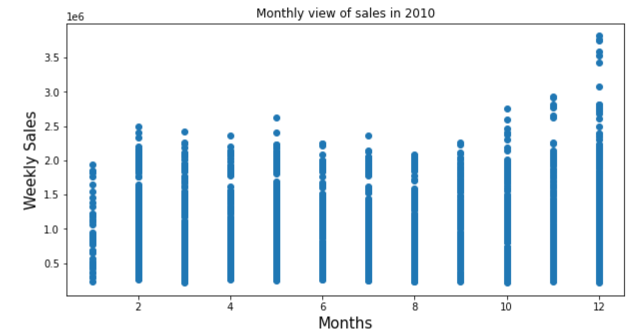
Out[28]:
```
1041256.3802088564
```

Providing a monthly and semester view of sales in units and give insights

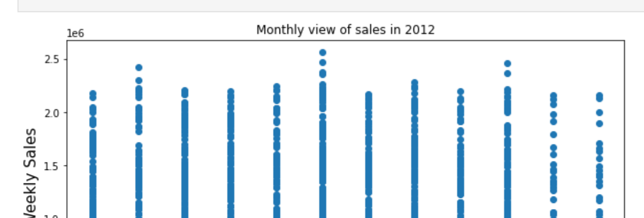
In [43]:
```python
# yearly wise

plt.figure(figsize=(10,5))
plt.scatter(data[data.Year==2010]["Month"],data[data.Year==2010]["Weekly_Sales"])
plt.xlabel("Months",fontsize=15)
plt.ylabel("Weekly Sales",fontsize=15)
plt.title("Monthly view of sales in 2010")
plt.show()
```



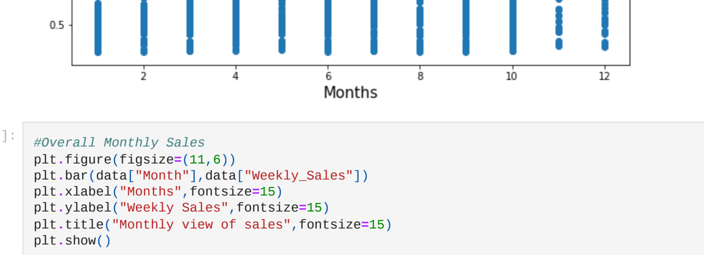
In [40]:
```python
plt.figure(figsize=(10,5))
plt.scatter(data[data.Year==2012]["Month"],data[data.Year==2012]["Weekly_Sales"])
plt.xlabel("Months",fontsize=15)
plt.ylabel("Weekly Sales",fontsize=15)
plt.title("Monthly view of sales in 2011",fontsize=17)
plt.show()
```



In [42]:
```python
plt.figure(figsize=(10,5))
plt.scatter(data[data.Year==2012]["Month"],data[data.Year==2012]["Weekly_Sales"])
plt.xlabel("Months",fontsize=15)
plt.ylabel("Weekly Sales",fontsize=15)
plt.title("Monthly view of sales in 2012")
plt.show()
```

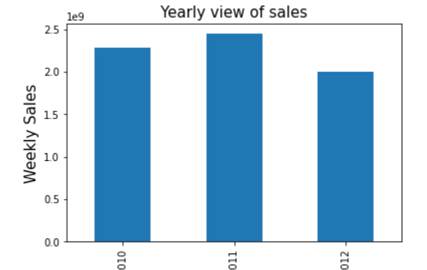


In [52]:
```python
#Overall Monthly Sales
plt.figure(figsize=(11,6))
plt.bar(data["Month"],data["Weekly_Sales"])
plt.xlabel("Months",fontsize=15)
plt.ylabel("Weekly Sales",fontsize=15)
plt.title("Monthly view of sales",fontsize=15)
plt.show()
```



In [53]:
```python
#Yearly Sales

plt.figure(figsize=(11,6))
data.groupby("Year")[["Weekly_Sales"]].sum().plot(kind='bar',legend=False)
plt.xlabel("Years",fontsize=15)
plt.ylabel("Weekly Sales",fontsize=15)
plt.title("Yearly view of sales",fontsize=15)
plt.show()
```

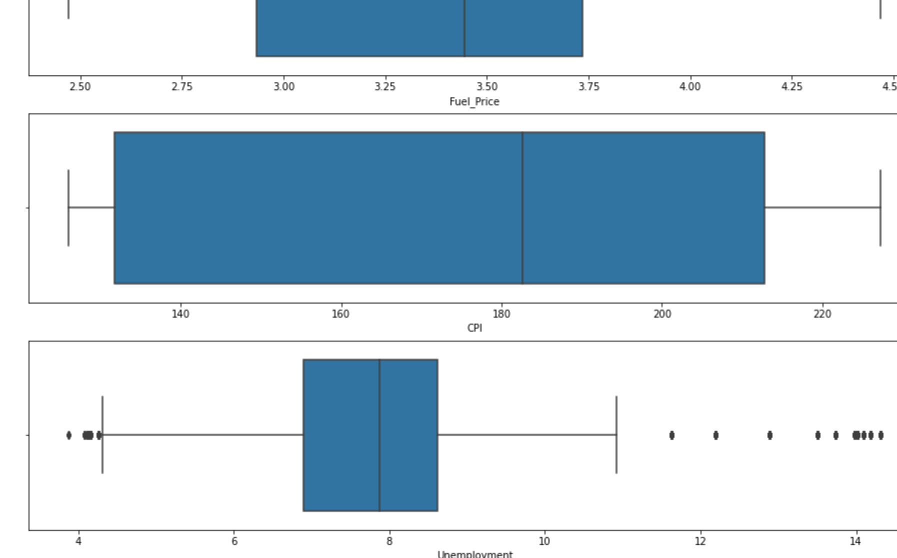
```
<Figure size 792x432 with 0 Axes>
```



Build prediction models to forecast demand

In [ ]:
```python
#Detecting outliers :
```

In [59]:
```python
fig, axis = plt.subplots(4,figsize=(16,16))
X = data[['Temperature','Fuel_Price','CPI','Unemployment']]
for i,column in enumerate(X):
    sns.boxplot(data[column],ax=axis[i])

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

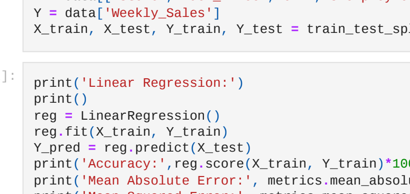


In [ ]:
```python
### Linear Regression:
```

In [61]:
```python
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression
X = data[['Store','Fuel_Price','CPI','Unemployment','Day','Month','Year']]
Y = data['Weekly_Sales']
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2)
```

In [62]:
```python
print('Linear Regression:')
print()
reg = LinearRegression()
reg.fit(X_train, Y_train)
Y_pred = reg.predict(X_test)
print('Accuracy:',reg.score(X_train, Y_train)*100)
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, Y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, Y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test, Y_pred)))
sns.scatterplot(Y_pred, Y_test)


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

```
Linear Regression:

Accuracy: 14.134167455727887
Mean Absolute Error: 431012.4729576318
Mean Squared Error: 269942356340.15155
Root Mean Squared Error: 519559.7716722798
```



In [ ]:
```python
# Random Forest Regressor
```

In [63]:
```python
from sklearn.ensemble import RandomForestRegressor
print('Random Forest Regressor:')
print()
rfr = RandomForestRegressor()
rfr.fit(X_train,Y_train)
Y_pred = rfr.predict(X_test)
print('Accuracy:',rfr.score(X_test, Y_test)*100)
print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, Y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, Y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test, Y_pred)))
sns.scatterplot(Y_pred, Y_test)

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

```
Random Forest Regressor:

Accuracy: 96.95885470685413
Mean Absolute Error: 56811.21409564879
Mean Squared Error: 9732693015.633503
Root Mean Squared Error: 98654.41204342309
```


In [ ]:
```python
###Here, Linear Regression is not an appropriate model to use which is clear from it's low accuracy.
##However, Random Forest Regression gives accuracy of over 95% , so, it is the best model to forecast demand.
```