jupyter Walmart_Project_kk Last Checkpoint: 13 minutes ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

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

[2]:
```python
data_sale = pd.read_csv('Walmart_Store_sales.csv')
data_sale.head() #check first 5
```

[2]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	05-02-2010	1643690.90	0	42.31	2.572	211.096358	8.106
1	1	12-02-2010	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	05-03-2010	1554806.68	0	46.50	2.625	211.350143	8.106

[3]:
```python
data_sale.shape
```

[3]: (6435, 8)

[4]:
```python
data_sale.info() #check for missing values
```

```
<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   object
 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: float64(5), int64(2), object(1)
memory usage: 402.3+ KB
```

[5]:
```python
data_sale.isna().sum() #check for the sum of missing values
```

[5]:
```
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64
```

[6]:
```python
from datetime import datetime #as date is treated as an object Dtype, we need to convert that into datetime format
data_sale['Date'] = pd.to_datetime(data_sale['Date'],format='mixed')
```

[7]:
```python
data_sale.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
```

1. Which store has maximum sales?

[9]:
```python
high_sales = data_sale.groupby('Store')['Weekly_Sales'].sum().round().sort_values(ascending=0)
high_sales
```

[9]:
```
Store
20    301397792.0
4     299543953.0
14    288999911.0
13    286517704.0
2     275382441.0
10    271617714.0
27    253855917.0
6     223756131.0
1     222402809.0
39    207445542.0
19    206634862.0
31    199613906.0
23    198750618.0
24    194016021.0
11    193962787.0
28    189263681.0
41    181341935.0
```

[10]:
```python
pd.DataFrame(high_sales).head(3) #seems like 20th store is best(maximum sales)
```

[10]:

	Weekly_Sales
Store	
20	301397792.0
4	299543953.0
14	288999911.0

[10]:
```
pd.DataFrame(high_sales).head(3) #seems like 20th store is best(maximum sales)
```

[10]:

Store	Weekly_Sales
20	301397792.0
4	299543953.0
14	288999911.0

[11]:
```
pd.DataFrame(high_sales).tail(3) #seems like 33rd is lowest(minimum sales)
```

[11]:

Store	Weekly_Sales
5	45475689.0
44	43293088.0
33	37160222.0

2. Which store has maximum standard deviation

[13]:
```
high_std = data_sale.groupby('Store')['Weekly_Sales'].std().round(3).sort_values(ascending=0)
high_std
```

[13]:
```
Store
14    317569.949
10    302262.063
20    275900.563
4     266201.442
13    265506.996
23    249788.038
27    239930.136
2     237683.695
39    217466.455
6     212525.856
35    211243.458
19    191722.639
41    187907.163
28    181758.968
18    176641.511
24    167745.678
11    165833.888
22    161251.351
1     155980.768
12    139166.872
32    138017.252
45    130168.527
21    128752.813
31    125855.943
15    120538.652
40    119002.113
25    112976.789
7     112585.469
17    112162.936
26    110431.288
8     106280.830
34    104630.165
29     99120.137
16     85769.680
9      69028.667
36     60725.174
```

[14]:
```
pd.DataFrame(high_std).head(2) #Store - 14 has a maximum standard deviation
```

[14]:

Store	Weekly_Sales
14	317569.949
10	302262.063

[15]:
```
store14 = data_sale[data_sale.Store == 14].Weekly_Sales
store14
```

[15]:
```
1859    2623469.95
1860    1704218.84
1861    2204556.70
1862    2095591.63
1863    2237544.75
           ...
1997    1522512.20
1998    1687592.16
1999    1639585.61
2000    1590274.72
2001    1704357.62
Name: Weekly_Sales, Length: 143, dtype: float64
```

[16]:
```
mean_to_stddev = store14.std()/store14.mean()*100

mean_to_stddev.round(3) #Mean to Standard Deviation = 15.714%
```

[16]:
```
15.714
```

3. Which store/s has good quarterly growth rate in Q3'2012

[18]:
```
q2_onlySales = data_sale[(data_sale['Date']>='2012-04-01')&(data_sale['Date']<='2012-06-30')].groupby('Store')['Weekly_Sales'].sum().round()
pd.DataFrame(q2_onlySales).head(6)
```

[18]:

Store	Weekly_Sales
1	21036966.0
2	25085124.0
3	5562668.0
4	28384185.0
5	4427262.0
6	20728970.0

[19]:
```
q3_onlySales = data_sale[(data_sale['Date']>='2012-07-01')&(data_sale['Date']<='2012-09-30')].groupby('Store')['Weekly_Sales'].sum().round()
pd.DataFrame(q3_onlySales).head(6)
```

[19]:

Store	Weekly_Sales
1	18633210.0
2	22396868.0
3	4966496.0
4	25652119.0
5	3880622.0
6	18341221.0

[20]:
```
pd.DataFrame({'Q2_Sales':q2_onlySales,'Q3_Sales':q3_onlySales,'Difference':q3_onlySales-q2_onlySales,
              'Growth_Rate':(q3_onlySales-q2_onlySales)/q3_onlySales*100}).sort_values(by=['Growth_Rate'],ascending=0).head(3)
```

[20]:

Store	Q2_Sales	Q3_Sales	Difference	Growth_Rate
16	6626133.0	6441311.0	-184822.0	-2.869323
7	7613594.0	7322394.0	-291200.0	-3.976841
35	10753571.0	10252123.0	-501448.0	-4.891163

[21]:
```
pd.DataFrame({'Q2_Sales':q2_onlySales,'Q3_Sales':q3_onlySales,'Difference':q3_onlySales-q2_onlySales,'Growth_Rate':(q3_onlySales-q2_onlySales)/q3_onlySal
```

[21]:

Store	Q2_Sales	Q3_Sales	Difference	Growth_Rate
29	7034493.0	6127862.0	-906631.0	-14.795225
45	10278900.0	8851242.0	-1427658.0	-16.129465
14	24427769.0	20140430.0	-4287339.0	-21.287227

4. Some holidays have a negative impact on sales. Find out holidays which have higher sales than the mean sales in non-holiday season for all stores together.

[23]:
```
Super_Bowl  = ['12-2-10','11-2-11','10-2-12', '8-2-13']
Labour_Day = ['10-9-10', '9-9-11', '7-9-12', '6-9-13']
ThanksGiving = ['26-11-10', '25-11-11', '23-11-12', '29-11-13']
Christmas = ['31-12-10', '30-12-11', '28-12-12', '27-12-13']
```

[24]:
```
Super_Bowl_sales = data_sale.loc[data_sale.Date.isin(Super_Bowl)]['Weekly_Sales'].mean().round(2)
Labour_Day_sales = data_sale.loc[data_sale.Date.isin(Labour_Day)]['Weekly_Sales'].mean().round(2)
Thanksgiving_sales = data_sale.loc[data_sale.Date.isin(ThanksGiving)]['Weekly_Sales'].mean().round(2)
Christmas_sales = data_sale.loc[data_sale.Date.isin(Christmas)]['Weekly_Sales'].mean().round(2)
```

```
C:\Users\Asus\AppData\Local\Temp\ipykernel_18952\3188927236.py:1: FutureWarning: The behavior of 'isin' with dtype=datetime64[ns] and castable values
(e.g. strings) is deprecated. In a future version, these will not be considered matching by isin. Explicitly cast to the appropriate dtype before callin
g isin instead.
  Super_Bowl_sales = data_sale.loc[data_sale.Date.isin(Super_Bowl)]['Weekly_Sales'].mean().round(2)
C:\Users\Asus\AppData\Local\Temp\ipykernel_18952\3188927236.py:2: FutureWarning: The behavior of 'isin' with dtype=datetime64[ns] and castable values
(e.g. strings) is deprecated. In a future version, these will not be considered matching by isin. Explicitly cast to the appropriate dtype before callin
g isin instead.
  Labour_Day_sales = data_sale.loc[data_sale.Date.isin(Labour_Day)]['Weekly_Sales'].mean().round(2)
C:\Users\Asus\AppData\Local\Temp\ipykernel_18952\3188927236.py:3: FutureWarning: The behavior of 'isin' with dtype=datetime64[ns] and castable values
(e.g. strings) is deprecated. In a future version, these will not be considered matching by isin. Explicitly cast to the appropriate dtype before callin
g isin instead.
  Thanksgiving_sales = data_sale.loc[data_sale.Date.isin(ThanksGiving)]['Weekly_Sales'].mean().round(2)
C:\Users\Asus\AppData\Local\Temp\ipykernel_18952\3188927236.py:4: FutureWarning: The behavior of 'isin' with dtype=datetime64[ns] and castable values
(e.g. strings) is deprecated. In a future version, these will not be considered matching by isin. Explicitly cast to the appropriate dtype before callin
g isin instead.
  Christmas_sales = data_sale.loc[data_sale.Date.isin(Christmas)]['Weekly_Sales'].mean().round(2)
```

[25]:
```
Super_Bowl_sales, Labour_Day_sales, Thanksgiving_sales, Christmas_sales
```

[25]: (1079127.99, 1042427.29, 1471273.43, 960833.11)

[26]:
```
no_holiday_sales =  data_sale[(data_sale['Holiday_Flag']==0)]['Weekly_Sales'].mean().round(2)
no_holiday_sales
```

[26]: 1041256.38

[27]:
```python
Together = pd.DataFrame([{'Super_Bowl_sales':Super_Bowl_sales,'Labour_Day_sales':Labour_Day_sales,
                          'ThanksGiving_sales':Thanksgiving_sales,'Christmas_sales':Christmas_sales}])

Together #Thanksgiving has the highest sales
```

[27]:

	Super_Bowl_sales	Labour_Day_sales	ThanksGiving_sales	Christmas_sales
0	1079127.99	1042427.29	1471273.43	960833.11

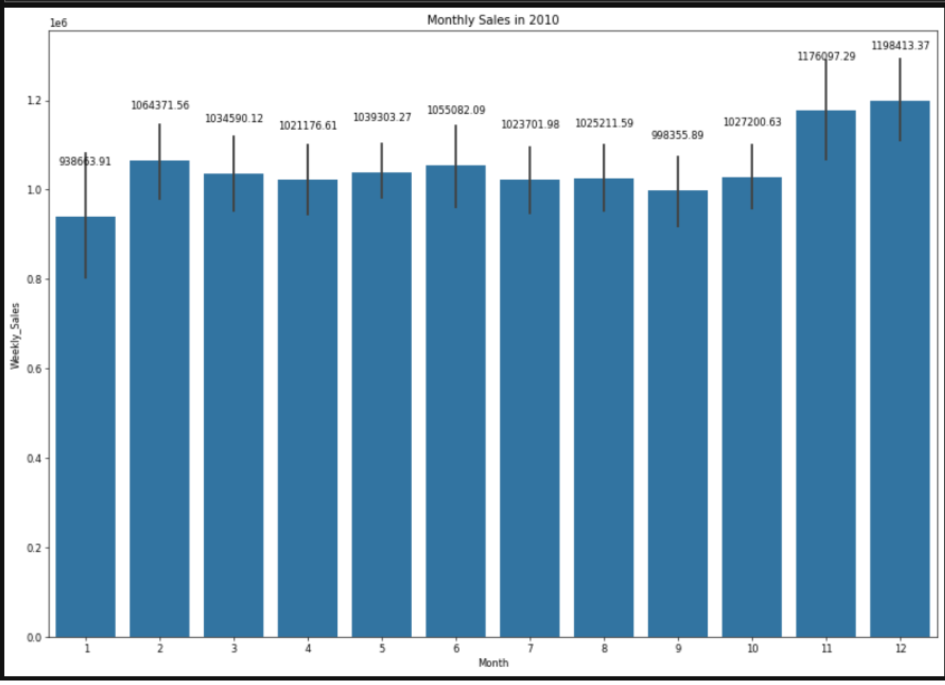
5. Provide a monthly and semester view of sales in units and give insights

[46]:
```python
data_sale['Day'] = pd.DatetimeIndex(data_sale['Date']).day
data_sale['Month'] = pd.DatetimeIndex(data_sale['Date']).month
data_sale['Year'] = pd.DatetimeIndex(data_sale['Date']).year
data_sale.head()
```

[46]:

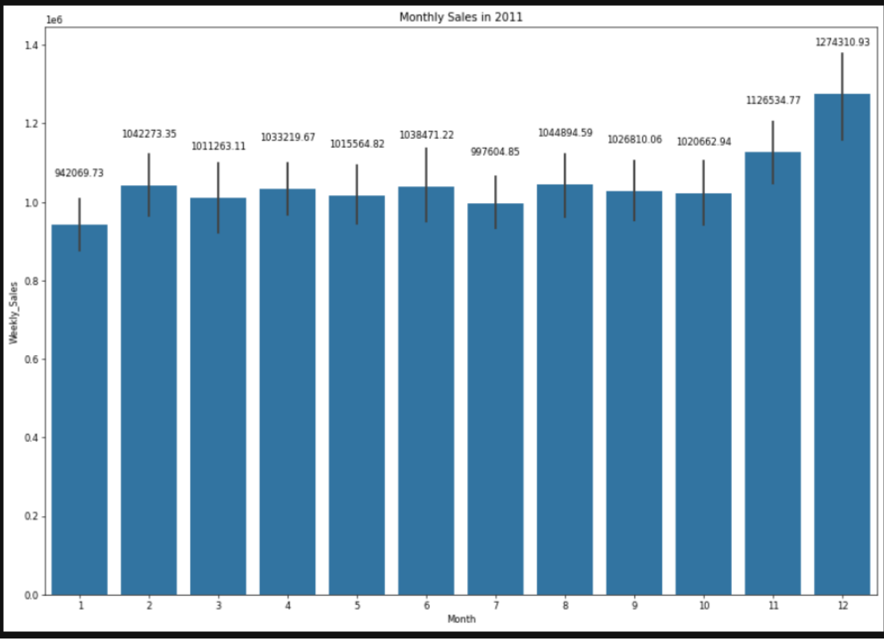
	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Day	Month	Year
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106	2	5	2010
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106	2	12	2010
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106	19	2	2010
3	1	2010-02-26	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

[30]:
```python
plt.figure(figsize=(16,11),dpi=60)
one_graph = sns.barplot(data=data_sale,x=data_sale[data_sale.Year==2010]['Month'],y=data_sale[data_sale.Year==2010]['Weekly_Sales'])
one_graph.set(title = 'Monthly Sales in 2010')

for i in one_graph.patches:
    one_graph.annotate(format(i.get_height(),'.2f'),(i.get_x()+i.get_width()/2.,i.get_height()),
                       ha='center',va='center',xytext=(0,55),textcoords='offset points')
```



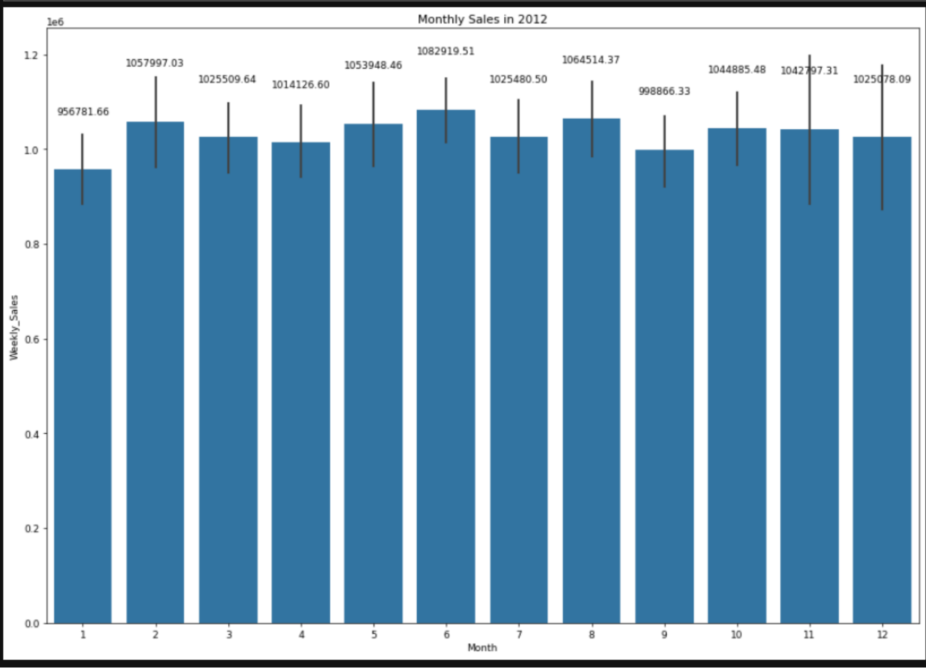
[31]:

```python
plt.figure(figsize=(16,11),dpi=60)
two_graph = sns.barplot(data=data_sale,x=data_sale[data_sale.Year==2011]['Month'],y=data_sale[data_sale.Year==2011]['Weekly_Sales'])
two_graph.set(title='Monthly Sales in 2011')
for i in two_graph.patches:
    two_graph.annotate(format(i.get_height(),'.2f'),(i.get_x()+i.get_width()/2.,i.get_height()),
                       ha='center',va='center',xytext=(0,55),textcoords='offset points')
```



[32]:

```python
plt.figure(figsize=(16,11),dpi=65)
three_graph = sns.barplot(data=data_sale,x=data_sale[data_sale.Year==2012]['Month'],y=data_sale[data_sale.Year==2012]['Weekly_Sales'])
three_graph.set(title='Monthly Sales in 2012')
for i in three_graph.patches:
    three_graph.annotate(format(i.get_height(),'.2f'),(i.get_x()+i.get_width()/2.,i.get_height()),
                       ha='center',va='center',xytext=(0,60),textcoords='offset points')
```



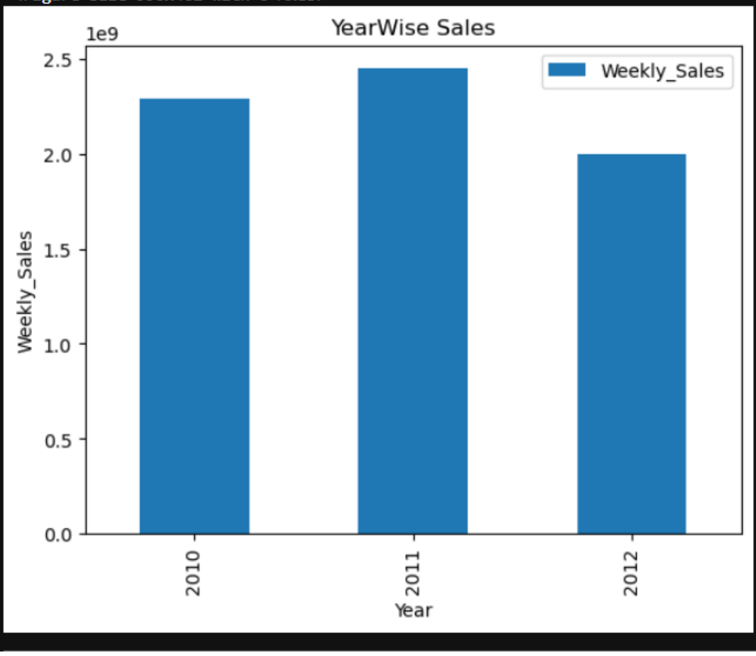
[33]:
```
plt.figure(figsize=(13,6),dpi=62)
plt.bar(data_sale['Month'],data_sale['Weekly_Sales'])
plt.xlabel('Months')
plt.ylabel('Weekly_Sales')
plt.title('Monthly Sales Data')
```

[33]: Text(0.5, 1.0, 'Monthly Sales Data')

[104]:
```
plt.figure(figsize=(15,7),dpi=66)
data_sale.groupby('Year')[['Weekly_Sales']].sum().plot(kind='bar',legend=True)
plt.xlabel('Year')
plt.ylabel('Weekly_Sales')
plt.title('YearWise Sales')
```

[104]: Text(0.5, 1.0, 'YearWise Sales')

<Figure size 990x462 with 0 Axes>

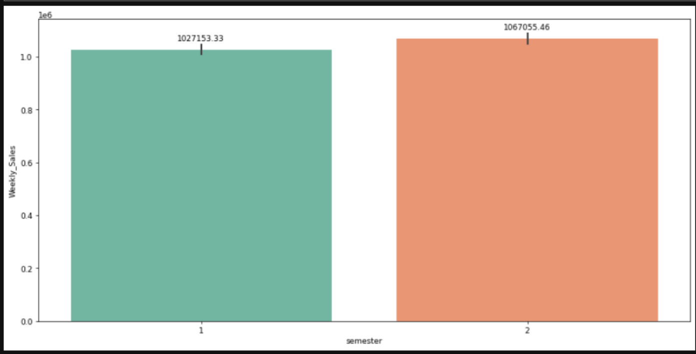


[56]:
```
#Semesterwise Sales
data_sale['semester'] = np.where(data_sale['Month']< 7, 1, 2)
data_sale.head()
```

[56]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Day	Month	Year	semester
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106	2	5	2010	1
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106	2	12	2010	2
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106	19	2	2010	1
3	1	2010-02-26	1409727.59	0	46.63	2.561	211.319643	8.106	26	2	2010	1
4	1	2010-05-03	1554806.68	0	46.50	2.625	211.350143	8.106	3	5	2010	1

```
[140]: plt.figure(figsize=(15,7),dpi=65)
sem = sns.barplot(data = data_sale,x='semester',y='Weekly_Sales', hue='semester', dodge=False, palette='Set2', legend=0)
for i in sem.patches:
    sem.annotate(format(i.get_height(),'.2f'),(i.get_x()+i.get_width()/2.,i.get_height()),
                 ha='center',va='center',xytext=(0,15),textcoords='offset points')
```




```
[120]: #Define independent and dependent variable
# Select features and target
x=data_sale[['Store','Fuel_Price','CPI','Unemployment','Day','Month','Year']]
y=data_sale['Weekly_Sales']
```

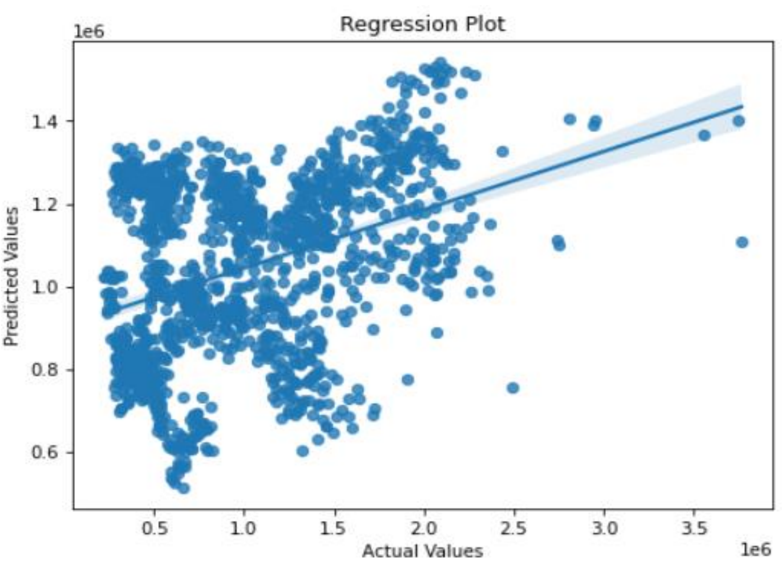
```
[142]: from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

```
[144]: from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

```
[150]: from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import r2_score
from IPython.display import display, Markdown #Added this for bold formatting.

display(Markdown('**Linear Regression**'))

lr = LinearRegression()
lr.fit(x_train, y_train)
lr_y_pred = lr.predict(x_test)



print('Train Accuracy Score:', lr.score(x_train, y_train).round(5)*100, '%')
print('Test Accuracy Score:', r2_score(y_test, lr_y_pred).round(5)*100, '%')

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, lr_y_pred).round(3))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, lr_y_pred).round(3))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, lr_y_pred)).round(3))

# Fixed plotting code
plt.figure(figsize=(7,5), dpi=75)
sns.regplot(x=y_test, y=lr_y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Regression Plot')
plt.show()
```

Linear Regression

```
Train Accuracy Score: 14.757000000000001 %
Test Accuracy Score: 13.276 %
Mean Absolute Error: 427687.491
Mean Squared Error: 271416877089.924
Root Mean Squared Error: 520976.849
```

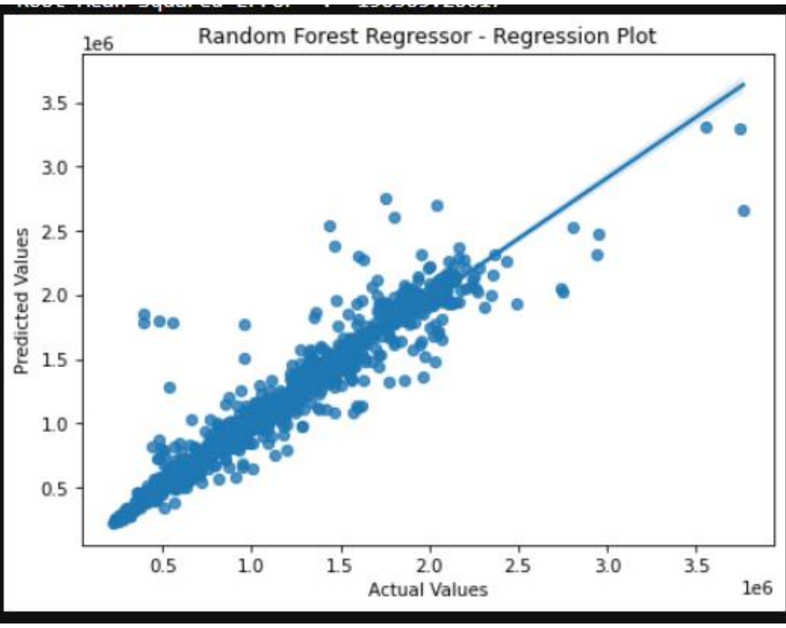


[162]:

```python
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(x_train, y_train)

# Predict and evaluate
rfr_y_pred = rfr.predict(x_test)

R2_rfr = r2_score(y_test, rfr_y_pred)
mae = metrics.mean_absolute_error(y_test, rfr_y_pred)
mse = metrics.mean_squared_error(y_test, rfr_y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, rfr_y_pred))

display(Markdown('**Random Forest Regressor**'))

print('Accuracy                 : ', R2_rfr.round(5)*100, '%')
print('Mean Absolute Error      : ', mae.round(5))
print('Mean Squared Error       : ', mse.round(5))
print('Root Mean Squared Error  : ', rmse.round(5))

# Fixed plotting code
plt.figure(figsize=(7,5), dpi=75)
sns.regplot(x=y_test, y=rfr_y_pred)  # Pass x and y as named arguments
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Random Forest Regressor - Regression Plot')
plt.show()
```

Random Forest Regressor

```
Accuracy                 :  91.984 %
Mean Absolute Error      :  80730.16203
Mean Squared Error       :  25085899508.46507
Root Mean Squared Error  :  158385.28817
```



[164]:
```
from sklearn.model_selection import cross_val_score
```

[166]:
```
# Linear Regression

lr_scores = cross_val_score(lr, x_train,y_train, cv=10, scoring='r2')
print(lr_scores)
print("Mean Score:", lr_scores.mean()*100,'%')
```

```
[0.1407938  0.22136063 0.13030993 0.16570556 0.16420734 0.14010222
 0.1386756  0.08596318 0.05685548 0.17510001]
Mean Score: 14.190737457064781 %
```

[168]:
```
# Random Forest Regression

rfr_scores = cross_val_score(rfr, x_train,y_train, cv=10, scoring='r2')
print(rfr_scores)
print("Mean Score:", rfr_scores.mean()*100,'%')
```

```
[0.96303951 0.96080219 0.95201032 0.9630272  0.9661305  0.95157184
 0.94738629 0.92837591 0.94018771 0.95321581]
Mean Score: 95.25747294706642 %
```

[ ]: