```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from PIL import Image

walmart = Image.open("/Users/saileshkumarm/Downloads/walmart-
icon.webp")
walmart
```

shutterstock.com · 2425802673

```python
data =
pd.read_csv("/Users/saileshkumarm/Downloads/Walmart_Store_sales.csv")

data.head()

   Store        Date  Weekly_Sales  Holiday_Flag  Temperature
Fuel_Price  \
0      1  05-02-2010    1643690.90             0        42.31
2.572
1      1  12-02-2010    1641957.44             1        38.51
2.548
2      1  19-02-2010    1611968.17             0        39.93
2.514
3      1  26-02-2010    1409727.59             0        46.63
2.561
4      1  05-03-2010    1554806.68             0        46.50
2.625
```

```
          CPI  Unemployment
0  211.096358         8.106
1  211.242170         8.106
2  211.289143         8.106
3  211.319643         8.106
4  211.350143         8.106
```

```
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   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
```

```
data.isna().sum()
```

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

```
from datetime import datetime
data['Date']=pd.to_datetime(data['Date'],format="mixed")
```

```
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
```

```
data.head()
```

```
   Store       Date  Weekly_Sales  Holiday_Flag  Temperature
Fuel_Price  \
0      1 2010-05-02    1643690.90             0        42.31
2.572
1      1 2010-12-02    1641957.44             1        38.51
2.548
2      1 2010-02-19    1611968.17             0        39.93
2.514
3      1 2010-02-26    1409727.59             0        46.63
2.561
4      1 2010-05-03    1554806.68             0        46.50
2.625

          CPI  Unemployment
0  211.096358         8.106
1  211.242170         8.106
2  211.289143         8.106
3  211.319643         8.106
4  211.350143         8.106
```

```
Highest_sales = data.groupby('Store')
['Weekly_Sales'].sum().round(1).sort_values(ascending = 0)

print(Highest_sales)
```

```
Store
20    301397792.5
4     299543953.4
14    288999911.3
13    286517703.8
2     275382441.0
10    271617713.9
27    253855916.9
6     223756130.6
1     222402808.8
39    207445542.5
19    206634862.1
31    199613905.5
23    198750617.8
24    194016021.3
```

```
11    193962786.8
28    189263680.6
41    181341934.9
32    166819246.2
18    155114734.2
22    147075648.6
12    144287230.2
26    143416393.8
34    138249763.0
40    137870309.8
35    131520672.1
8     129951181.1
17    127782138.8
45    112395341.4
21    108117878.9
25    101061179.2
43     90565435.4
15     89133683.9
7      81598275.1
42     79565752.4
9      77789219.0
29     77141554.3
16     74252425.4
37     74202740.3
30     62716885.1
3      57586735.1
38     55159626.4
36     53412215.0
5      45475688.9
44     43293087.8
33     37160222.0
Name: Weekly_Sales, dtype: float64

highest_sales = pd.DataFrame(Highest_sales)
highest_sales.head()

       Weekly_Sales
Store
20      301397792.5
4       299543953.4
14      288999911.3
13      286517703.8
2       275382441.0

Highest_std = data.groupby('Store')
['Weekly_Sales'].std().round().sort_values(ascending = 0)

Highest_std = pd.DataFrame(Highest_std)

Highest_std
```

	Weekly_Sales
Store	
14	317570.0
10	302262.0
20	275901.0
4	266201.0
13	265507.0
23	249788.0
27	239930.0
2	237684.0
39	217466.0
6	212526.0
35	211243.0
19	191723.0
41	187907.0
28	181759.0
18	176642.0
24	167746.0
11	165834.0
22	161251.0
1	155981.0
12	139167.0
32	138017.0
45	130169.0
21	128753.0
31	125856.0
15	120539.0
40	119002.0
25	112977.0
7	112585.0
17	112163.0
26	110431.0
8	106281.0
34	104630.0
29	99120.0
16	85770.0
9	69029.0
36	60725.0
42	50263.0
3	46320.0
38	42768.0
43	40598.0
5	37738.0
44	24763.0
33	24133.0
30	22810.0
37	21837.0

```
store14_data = data[data.Store == 14].Weekly_Sales
```

```
store14_data
```

```
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
```

```
cv_store14 = (store14_data.std()/store14_data.mean())*100
```

```
cv_store14.round(2)
```

```
15.71
```

```
Q2_sales = data[(data['Date'] >= '2012-04-01')& (data['Date'] <=
'2012-06-30')].groupby('Store')['Weekly_Sales'].sum().round()
Q3_sales = data[(data['Date'] >= '2012-07-01')& (data['Date'] <=
'2012-09-30')].groupby('Store')['Weekly_Sales'].sum().round()
```

```
Q2_sales
```

```
Store
1     21036966.0
2     25085124.0
3      5562668.0
4     28384185.0
5      4427262.0
6     20728970.0
7      7613594.0
8     11934276.0
9      7431320.0
10    23598434.0
11    17879096.0
12    13193365.0
13    26803226.0
14    24427769.0
15     7867952.0
16     6626133.0
17    12918892.0
18    13834706.0
19    18315279.0
20    27550181.0
21     9226280.0
22    13329065.0
```

```
23    18283425.0
24    17768192.0
25     9247467.0
26    13218290.0
27    22593641.0
28    16986000.0
29     7034493.0
30     5786335.0
31    18249155.0
32    15415236.0
33     3512138.0
34    12858028.0
35    10753571.0
36     4090379.0
37     6859778.0
38     5732363.0
39    20191586.0
40    12849747.0
41    17560036.0
42     7608247.0
43     8239793.0
44     4322555.0
45    10278900.0
Name: Weekly_Sales, dtype: float64
```

```
Q3_sales
```

```
Store
1     18633210.0
2     22396868.0
3      4966496.0
4     25652119.0
5      3880622.0
6     18341221.0
7      7322394.0
8     10873860.0
9      6528240.0
10    21169356.0
11    16094363.0
12    11777508.0
13    24319994.0
14    20140430.0
15     6909374.0
16     6441311.0
17    11533998.0
18    12507522.0
19    16644341.0
20    24665938.0
21     8403508.0
22    11818544.0
```

```
23    17103654.0
24    16126000.0
25     8309440.0
26    12417575.0
27    20191238.0
28    15055660.0
29     6127862.0
30     5181974.0
31    16454328.0
32    14142165.0
33     3177072.0
34    11476259.0
35    10252123.0
36     3578124.0
37     6250524.0
38     5129298.0
39    18899955.0
40    11647661.0
41    16373588.0
42     6830840.0
43     7376726.0
44     4020486.0
45     8851242.0
Name: Weekly_Sales, dtype: float64
```

```
qoq_growth=pd.DataFrame({'Q2_sales':Q2_sales,'Q3_sales':Q3_sales,'Sale_diff':(Q3_sales-Q2_sales),'%Sale_Growth':((Q3_sales-Q2_sales)/(Q3_sales))*100}).sort_values(by=['%Sale_Growth'],ascending=0)

qoq_growth
```

Store	Q2_sales	Q3_sales	Sale_diff	%Sale_Growth
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
26	13218290.0	12417575.0	-800715.0	-6.448240
39	20191586.0	18899955.0	-1291631.0	-6.834043
23	18283425.0	17103654.0	-1179771.0	-6.897772
41	17560036.0	16373588.0	-1186448.0	-7.246109
44	4322555.0	4020486.0	-302069.0	-7.513246
32	15415236.0	14142165.0	-1273071.0	-9.001953
37	6859778.0	6250524.0	-609254.0	-9.747247
8	11934276.0	10873860.0	-1060416.0	-9.751974
21	9226280.0	8403508.0	-822772.0	-9.790816
19	18315279.0	16644341.0	-1670938.0	-10.039076
24	17768192.0	16126000.0	-1642192.0	-10.183505
13	26803226.0	24319994.0	-2483232.0	-10.210660
40	12849747.0	11647661.0	-1202086.0	-10.320407
33	3512138.0	3177072.0	-335066.0	-10.546377

```
18     13834706.0  12507522.0 -1327184.0    -10.611087
4      28384185.0  25652119.0 -2732066.0    -10.650450
31     18249155.0  16454328.0 -1794827.0    -10.907933
11     17879096.0  16094363.0 -1784733.0    -11.089181
25      9247467.0   8309440.0  -938027.0    -11.288691
42      7608247.0   6830840.0  -777407.0    -11.380840
10     23598434.0  21169356.0 -2429078.0    -11.474501
30      5786335.0   5181974.0  -604361.0    -11.662756
20     27550181.0  24665938.0 -2884243.0    -11.693222
43      8239793.0   7376726.0  -863067.0    -11.699865
38      5732363.0   5129298.0  -603065.0    -11.757262
27     22593641.0  20191238.0 -2402403.0    -11.898245
2      25085124.0  22396868.0 -2688256.0    -12.002821
3       5562668.0   4966496.0  -596172.0    -12.003876
17     12918892.0  11533998.0 -1384894.0    -12.007059
12     13193365.0  11777508.0 -1415857.0    -12.021703
34     12858028.0  11476259.0 -1381769.0    -12.040239
22     13329065.0  11818544.0 -1510521.0    -12.780940
28     16986000.0  15055660.0 -1930340.0    -12.821358
1      21036966.0  18633210.0 -2403756.0    -12.900386
6      20728970.0  18341221.0 -2387749.0    -13.018484
9       7431320.0   6528240.0  -903080.0    -13.833437
15      7867952.0   6909374.0  -958578.0    -13.873587
5       4427262.0   3880622.0  -546640.0    -14.086402
36      4090379.0   3578124.0  -512255.0    -14.316301
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
```

```
#Holiday Events
Super_Bowl= ['12-2-2010', '11-2-2011', '10-2-2012','8-2-2013']
Labour_Day= ['10-9-2010', '9-9-2011', '7-9-2012', '6-9-2013']
Thanksgiving = ['26-11-2010', '25-11-2011', '23-11-2012', '29-11-
2013']
Christmas = ['31-12-2010', '30-12-2011', '28-12-2012', '27-12-2013']

SB_sales = data.loc[data.Date.isin(Super_Bowl)]
['Weekly_Sales'].mean().round()
LB_sales = data.loc[data.Date.isin(Labour_Day)]
['Weekly_Sales'].mean().round()
TG_sales = data.loc[data.Date.isin(Thanksgiving)]
['Weekly_Sales'].mean().round()
CH_sales = data.loc[data.Date.isin(Christmas)]
['Weekly_Sales'].mean().round()


non_holiday_sales = data[data['Holiday_Flag']==0]
['Weekly_Sales'].mean().round(2)
```

```
diff_holiday = pd.DataFrame([{'Super_Bowl_sales':SB_sales,
                             'Labour_Day_sales':LB_sales,
                             'Thanksgiving_sales': TG_sales,
                             'Christmas_sales': CH_sales,
                             'Non_holiday_sales':non_holiday_sales}])

s1_data = data[data['Store'] == 1]
s1_data2 = s1_data.sort_values(by='Date')

s1_data2

     Store       Date  Weekly_Sales  Holiday_Flag  Temperature
Fuel_Price  \
34       1 2010-01-10    1453329.50             0        71.89
2.603
8        1 2010-02-04    1594968.28             0        62.27
2.719
21       1 2010-02-07    1492418.14             0        80.91
2.669
2        1 2010-02-19    1611968.17             0        39.93
2.514
3        1 2010-02-26    1409727.59             0        46.63
2.561
..     ...        ...           ...           ...          ...
...
131      1 2012-10-08    1592409.97             0        85.05
3.494
141      1 2012-10-19    1508068.77             0        67.97
3.594
142      1 2012-10-26    1493659.74             0        69.16
3.506
118      1 2012-11-05    1611096.05             0        73.77
3.688
140      1 2012-12-10    1573072.81             0        62.99
3.601

            CPI  Unemployment  Days
34   211.671989         7.838     1
8    210.820450         7.808    26
21   211.223533         7.787    29
2    211.289143         8.106    41
3    211.319643         8.106    48
..          ...           ...   ...
131  221.958433         6.908  1003
141  223.425723         6.573  1014
142  223.444251         6.573  1021
118  221.725663         7.143  1031
140  223.381296         6.573  1066

[143 rows x 9 columns]
```

```
s1_data2.head()
```

```
    Store       Date  Weekly_Sales  Holiday_Flag  Temperature  
Fuel_Price  \
34      1 2010-01-10    1453329.50             0        71.89  
2.603   
8       1 2010-02-04    1594968.28             0        62.27  
2.719   
21      1 2010-02-07    1492418.14             0        80.91  
2.669   
2       1 2010-02-19    1611968.17             0        39.93  
2.514   
3       1 2010-02-26    1409727.59             0        46.63  
2.561   

           CPI  Unemployment  Days  
34  211.671989         7.838     1  
8   210.820450         7.808    26  
21  211.223533         7.787    29  
2   211.289143         8.106    41  
3   211.319643         8.106    48  
```

```
s1_data2.info()
```

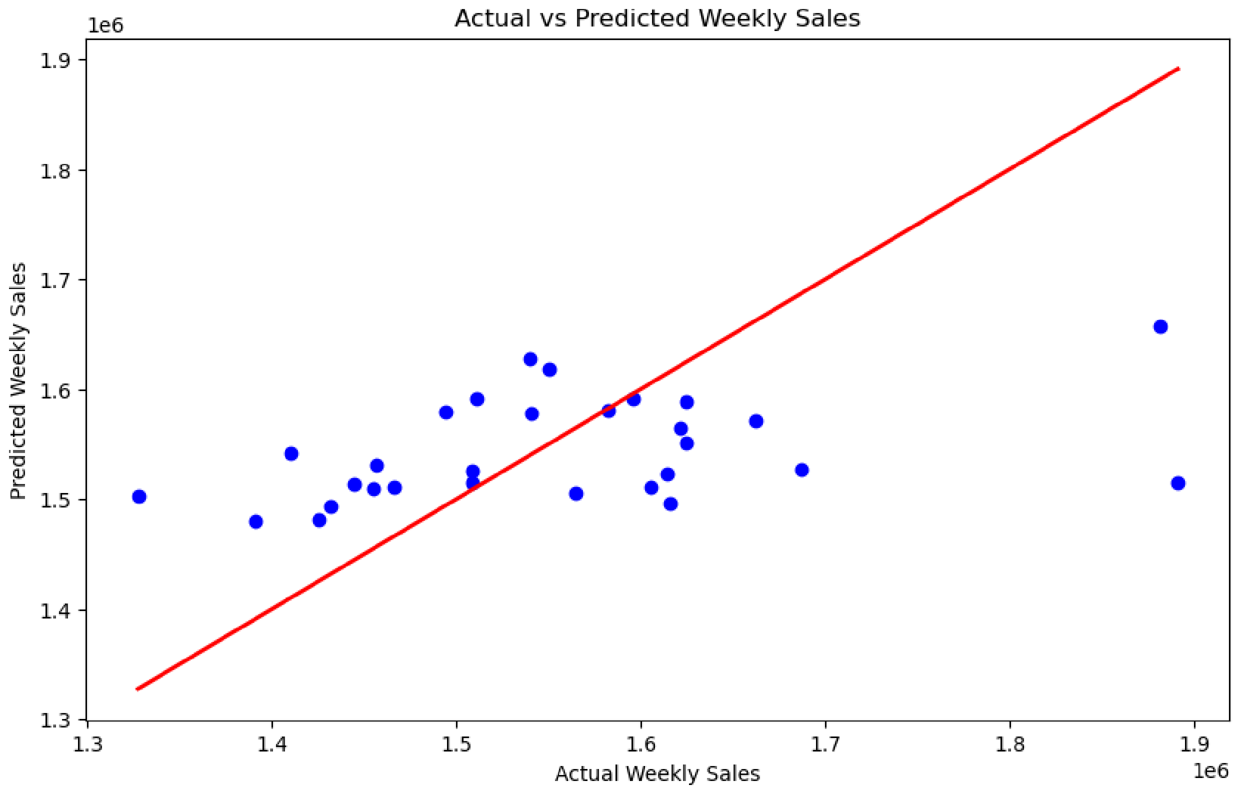
```
<class 'pandas.core.frame.DataFrame'>
Index: 143 entries, 34 to 140
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   Store         143 non-null    int64         
 1   Date          143 non-null    datetime64[ns]
 2   Weekly_Sales  143 non-null    float64       
 3   Holiday_Flag  143 non-null    int64         
 4   Temperature   143 non-null    float64       
 5   Fuel_Price    143 non-null    float64       
 6   CPI           143 non-null    float64       
 7   Unemployment  143 non-null    float64       
dtypes: datetime64[ns](1), float64(5), int64(2)
memory usage: 10.1 KB
```

```
s1_data2['Days'] = (s1_data2['Date']-s1_data2['Date'].min())

s1_data2['Days'] = (s1_data2['Date']-s1_data2['Date'].min()).dt.days
+1

X= s1_data2[['Days','Fuel_Price','CPI','Unemployment']]
Y= s1_data2['Weekly_Sales']

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 , random_state=42)
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
lin_reg = LinearRegression()
lin_reg.fit(X_train,Y_train)
Y_pred = lin_reg.predict(X_test)

## Using Linear Regression to predict Weekly_Sales

# Step 3: Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)

# Step 4: Build and Train the Linear Regression Model
model = LinearRegression()
model.fit(X_train, Y_train)

# Predict the sales
Y_pred = model.predict(X_test)

# Step 5: Evaluate the Model
mae = mean_absolute_error(Y_test, Y_pred)
mse = mean_squared_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)

coefficients = pd.DataFrame(lin_reg.coef_, X.columns,
columns=['Coefficient'])


print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R²): {r2}")

print(coefficients)

# Step 6: Visualize the Predictions
plt.figure(figsize=(10, 6))
plt.scatter(Y_test, Y_pred, color='blue')
plt.plot([Y_test.min(), Y_test.max()], [Y_test.min(), Y_test.max()],
color='red', linewidth=2)
plt.xlabel('Actual Weekly Sales')
plt.ylabel('Predicted Weekly Sales')
plt.title('Actual vs Predicted Weekly Sales')
plt.show()
```

```
Mean Absolute Error (MAE): 87081.11672443505
Mean Squared Error (MSE): 12971929290.524311
R-squared (R²): 0.17512372936665344
                Coefficient
Days              90.714196
```

```
Fuel_Price     -72100.810686
CPI             16199.580709
Unemployment   121002.335286
```




```
#Interpretation:

#CPI (Consumer Price Index):
##Higher CPI tends to be associated with higher sales.
##Inflation (reflected in higher CPI) could lead to increased consumer
spending.

#Unemployment Rate:
##Higher unemployment correlates with lower sales.
##Rising unemployment may reduce consumer spending.

#Fuel Price:
##Higher fuel prices reduce consumer spending.
##Consumers cut back on discretionary purchases when fuel costs rise.

#Days (Time Trend):
##The Days coefficient indicates growth or decline in sales over time.

  Cell In[120], line 4
    Higher CPI tends to be associated with higher sales.
           ^
```

```
SyntaxError: invalid syntax


#For each model, we'll compute key metrics like Mean Absolute Error
(MAE),
#Mean Squared Error (MSE), and R² score,and then determine which model
performs best.

#Implementing the Additional Models
from sklearn.linear_model import Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score


# Ridge Regression
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, Y_train)
Y_pred_ridge = ridge_model.predict(X_test)

# Lasso Regression
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, Y_train)
Y_pred_lasso = lasso_model.predict(X_test)

# Random Forest Regression
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, Y_train)
Y_pred_rf = rf_model.predict(X_test)

# Support Vector Regression
svr_model = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)
svr_model.fit(X_train, Y_train)
Y_pred_svr = svr_model.predict(X_test)

lin_reg = LinearRegression()
lin_reg.fit(X_train, Y_train)
Y_pred_lr = lin_reg.predict(X_test)

# Ridge Regression
ridge_reg = Ridge(alpha=1.0)
ridge_reg.fit(X_train, Y_train)
Y_pred_ridge = ridge_reg.predict(X_test)

# Lasso Regression
lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X_train, Y_train)
Y_pred_lasso = lasso_reg.predict(X_test)

# Random Forest Regression
```

```python
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train, Y_train)
Y_pred_rf = rf_reg.predict(X_test)

# Support Vector Regression
svr_reg = SVR(kernel='rbf', C=1.0, epsilon=0.1)
svr_reg.fit(X_train, Y_train)
Y_pred_svr = svr_reg.predict(X_test)

# Evaluate each model
models = {
    "Linear Regression": Y_pred_lr,
    "Ridge Regression": Y_pred_ridge,
    "Lasso Regression": Y_pred_lasso,
    "Random Forest Regression": Y_pred_rf,
    "Support Vector Regression": Y_pred_svr
}


#Evaluating the Models
def evaluate_model(Y_test, Y_pred, model_name):
    mae = mean_absolute_error(Y_test, Y_pred)
    mse = mean_squared_error(Y_test, Y_pred)
    r2 = r2_score(Y_test, Y_pred)

    print(f"Model: {model_name}")
    print(f"Mean Absolute Error: {mae:.4f}")
    print(f"Mean Squared Error: {mse:.4f}")
    print(f"R² Score: {r2:.4f}")
    print("-" * 30)
    return mae, mse, r2

# Evaluate each model
models = {
    "Linear Regression": Y_pred_lr,
    "Ridge Regression": Y_pred_ridge,
    "Lasso Regression": Y_pred_lasso,
    "Random Forest Regression": Y_pred_rf,
    "Support Vector Regression": Y_pred_svr
}

results = {}
for model_name, Y_pred in models.items():
    results[model_name] = evaluate_model(Y_test, Y_pred, model_name)
```

```
Model: Linear Regression
Mean Absolute Error: 87081.1167
Mean Squared Error: 12971929290.5243
R² Score: 0.1751
```

```
------------------------------
Model: Ridge Regression
Mean Absolute Error: 87026.0602
Mean Squared Error: 13077433533.7946
R² Score: 0.1684
------------------------------
Model: Lasso Regression
Mean Absolute Error: 87081.0856
Mean Squared Error: 12971939689.8148
R² Score: 0.1751
------------------------------
Model: Random Forest Regression
Mean Absolute Error: 84489.2595
Mean Squared Error: 9616810015.5455
R² Score: 0.3885
------------------------------
Model: Support Vector Regression
Mean Absolute Error: 95684.8615
Mean Squared Error: 16178760481.2167
R² Score: -0.0288
------------------------------

#High R² of Random Forrest suggest that the variables explain a
significant portion of
#the sales variance, leading to the hypothesis that these variables
impact sales.

'''
Insights from These Metrics:
*MAE gives you an average error in the same units as your dependent
variable.
*MSE penalizes larger errors more than smaller ones, making it useful
if you want to focus on large deviations.
*RMSE is like MSE but gives you a result in the same units as your
target variable, making it more interpretable.
*R² shows how well your model explains the variance of the data. A
higher R² indicates a better fit.
'''
```