SALES PREDICTION USING PYTHON

This Jupyter Notebook was created by [Dajah Vincent (https://www.linkedin.com/in/dajahvincent/)](https://www.linkedin.com/in/dajahvincent/)

Importing Data Manipulation and Visualization Libraries

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

In [2]:
```
#Reading the dataset from the CSV format

sales = pd.read_csv("Advertising.csv")
```

In [3]:
```
#Viewing the first 5 rows and last 5 rows from the dataset

sales
```

Out[3]:

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9
...	...	...	...	...	...
195	196	38.2	3.7	13.8	7.6
196	197	94.2	4.9	8.1	9.7
197	198	177.0	9.3	6.4	12.8
198	199	283.6	42.0	66.2	25.5
199	200	232.1	8.6	8.7	13.4

200 rows × 5 columns

In [4]:
```
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  200 non-null    int64
 1   TV          200 non-null    float64
 2   Radio       200 non-null    float64
 3   Newspaper   200 non-null    float64
 4   Sales       200 non-null    float64
dtypes: float64(4), int64(1)
memory usage: 7.9 KB
```

Exploratory Data Analysis - EDA

In [5]:
```
#Viewing the datasets columns with numerical values along with its basic statistical attributes

sales.describe().round(2)
```

Out[5]:

	Unnamed: 0	TV	Radio	Newspaper	Sales
count	200.00	200.00	200.00	200.00	200.00
mean	100.50	147.04	23.26	30.55	14.02
std	57.88	85.85	14.85	21.78	5.22
min	1.00	0.70	0.00	0.30	1.60
25%	50.75	74.38	9.98	12.75	10.38
50%	100.50	149.75	22.90	25.75	12.90
75%	150.25	218.82	36.52	45.10	17.40
max	200.00	296.40	49.60	114.00	27.00

In [6]:
```
# The datasets columns
sales.columns
```

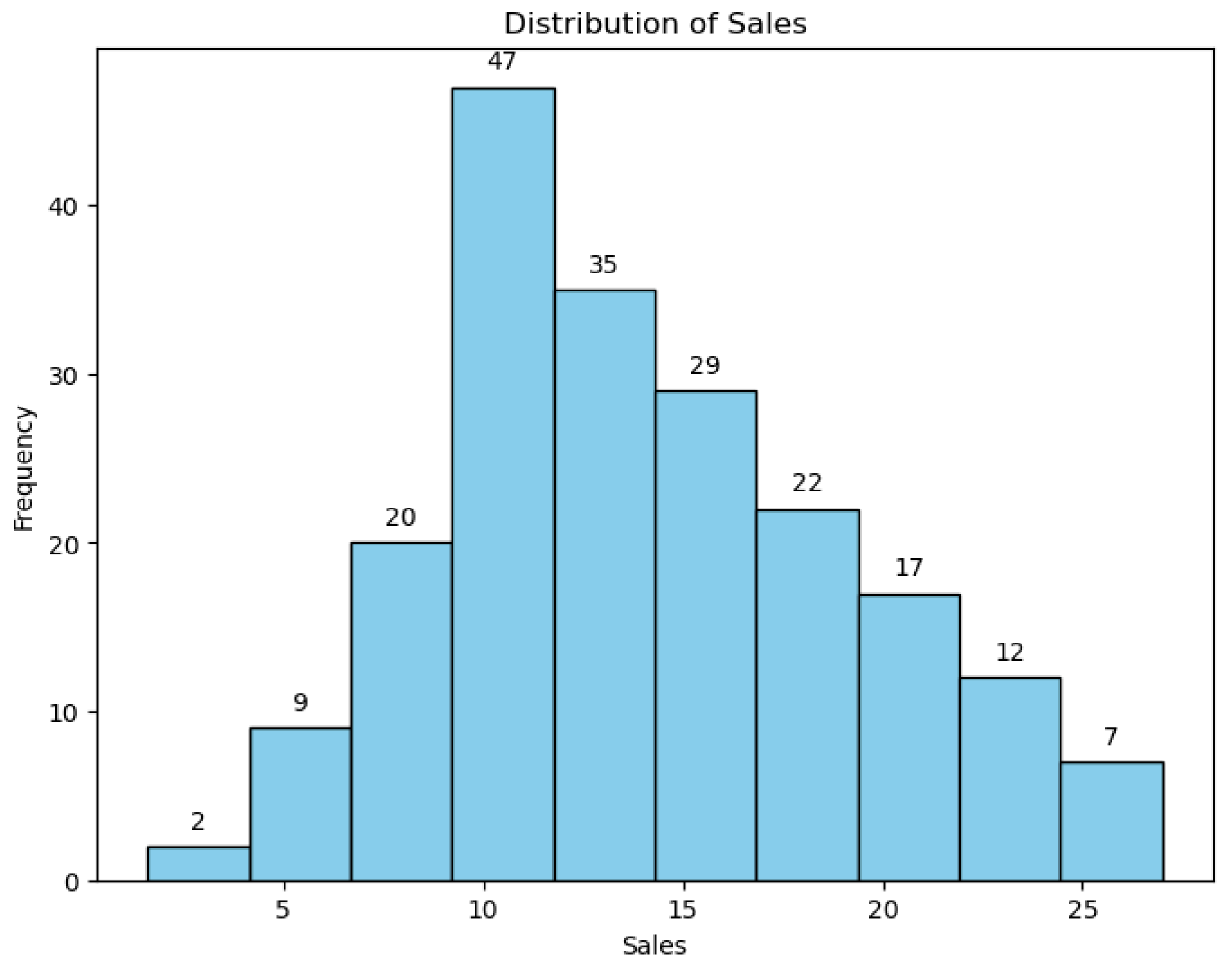
Out[6]:
```
Index(['Unnamed: 0', 'TV', 'Radio', 'Newspaper', 'Sales'], dtype='object')
```

In [7]:
```
#Checking if there are columns that contain null or empty values

sales.isna().sum()
```

Out[7]:
```
Unnamed: 0    0
TV            0
Radio         0
Newspaper     0
Sales         0
dtype: int64
```

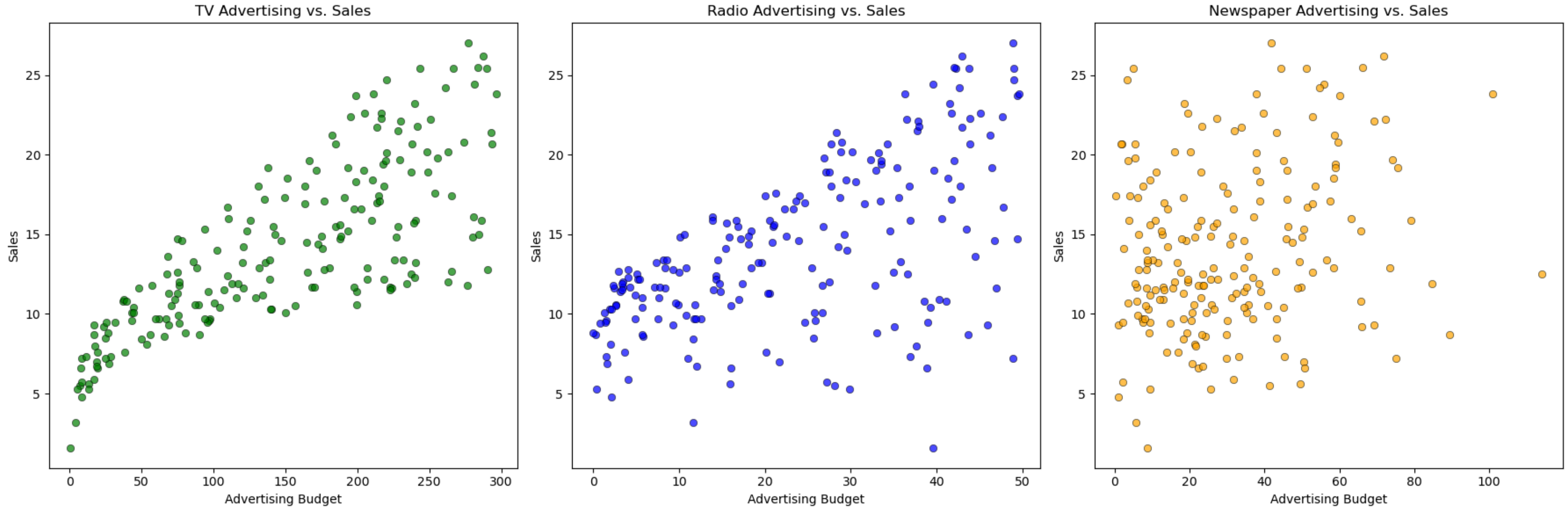
In [8]:

```
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have a pandas DataFrame called 'sales' with a 'Sales' column
plt.figure(figsize=(8, 6))  # Set the figure size (optional)

# Create the histogram
counts, bins, patches = plt.hist(sales['Sales'], bins=10, color='skyblue', edgecolor='black')

# Calculate bin centers
bin_centers = 0.5 * np.diff(bins) + bins[:-1]

# Add count labels to the bars
for count, bin_center in zip(counts, bin_centers):
    plt.annotate(str(int(count)), xy=(bin_center, count), xytext=(0, 5), textcoords='offset points', ha='center', va='bottom


plt.xlabel('Sales')
plt.ylabel('Frequency')
plt.title('Distribution of Sales')

plt.show()  # Display the plot
```

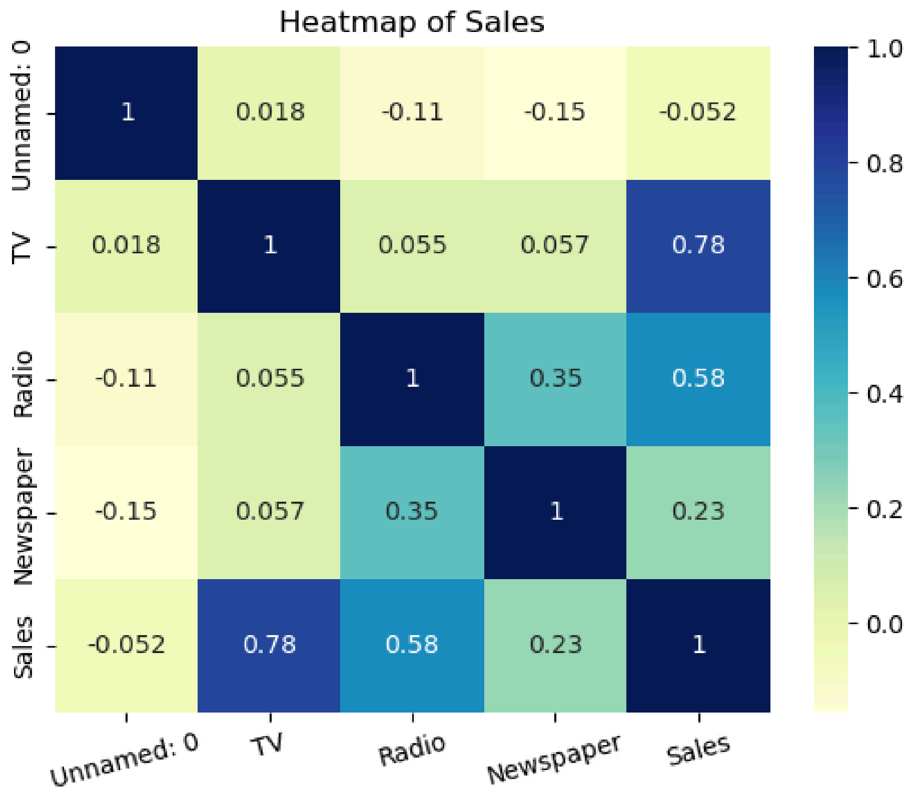


Distribution of Sales
Frequency
Sales
0
10
20
30
40
5
10
15
20
25
2
9
20
47
35
29
22
17
12
7

In [9]:
```python
# Create a 1x3 grid of subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Scatter plot for TV data (subplot 1)
sns.scatterplot(data=sales, x='TV', y='Sales', color='green', edgecolor='black', alpha=0.7, ax=axes[0])
axes[0].set_title('TV Advertising vs. Sales')

# Scatter plot for Radio data (subplot 2)
sns.scatterplot(data=sales, x='Radio', y='Sales', color='blue', edgecolor='black', alpha=0.7, ax=axes[1])
axes[1].set_title('Radio Advertising vs. Sales')

# Scatter plot for Newspaper data (subplot 3)
sns.scatterplot(data=sales, x='Newspaper', y='Sales', color='orange', edgecolor='black', alpha=0.7, ax=axes[2])
axes[2].set_title('Newspaper Advertising vs. Sales')

# Set common labels for all subplots
for ax in axes:
    ax.set_xlabel('Advertising Budget')
    ax.set_ylabel('Sales')

plt.tight_layout()  # Adjust subplot spacing

plt.show()  # Display the subplots
```

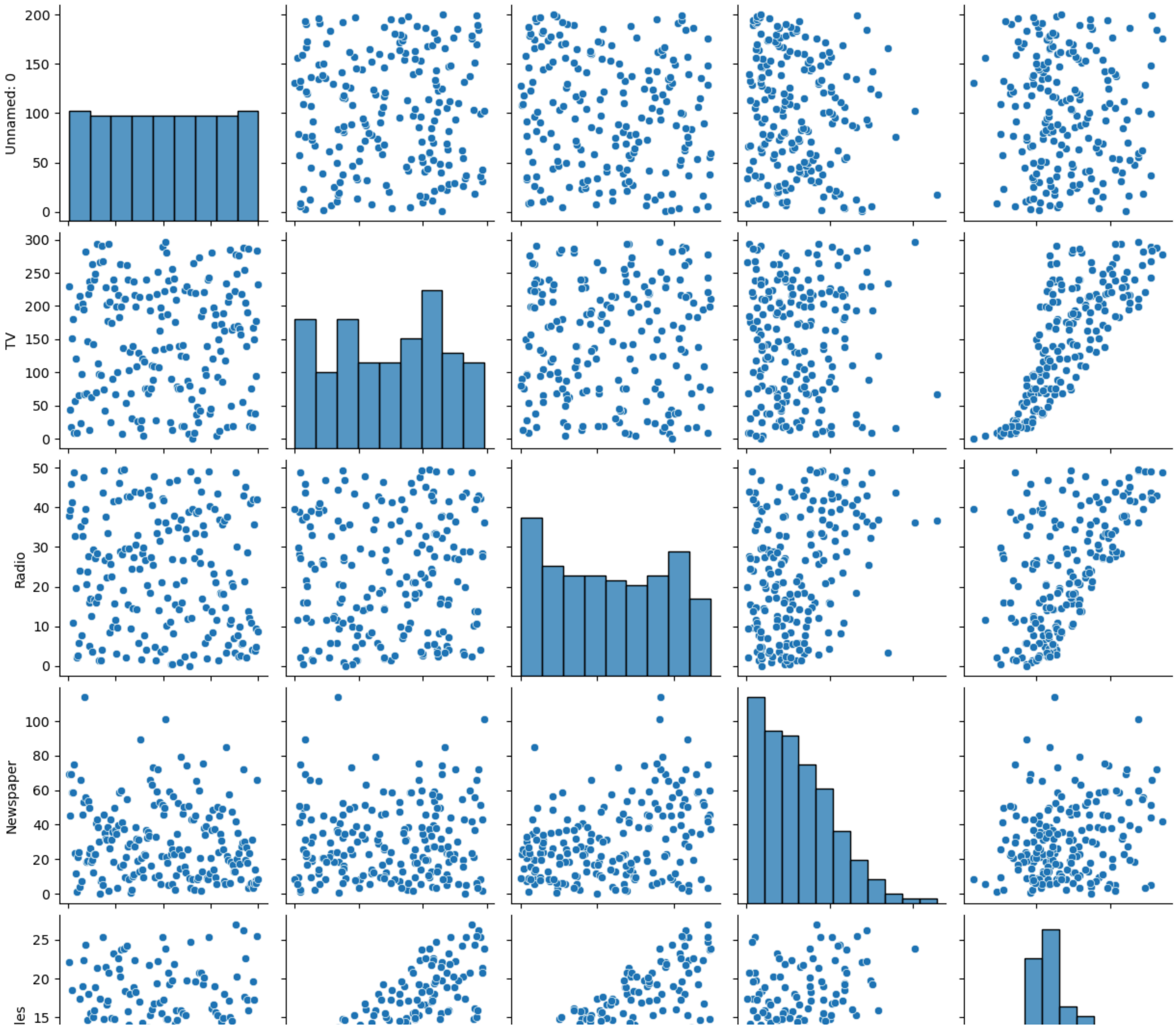


In [10]:
```
# Create the heatmap
sns.heatmap(sales.corr(),  annot=True, cmap='YlGnBu')

# Customize the plot (optional)
plt.xticks(rotation=15)
plt.title("Heatmap of Sales")

# Display the plot
plt.show()
```

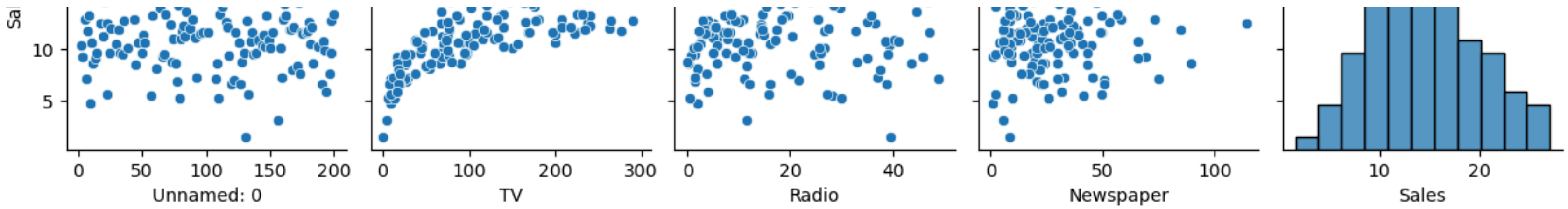


In [11]:
```
# Create a pairplot
sns.pairplot(sales)

# Show the plot

plt.show()
```

```
C:\Users\DajahV01\AppData\Local\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has cha
nged to tight
  self._figure.tight_layout(*args, **kwargs)
```



Unnamed: 0
TV
Radio
Newspaper
les



Feature Engineering

In [12]:
```python
sales['Total Advert'] = sales['TV'] + sales['Radio'] + sales['Newspaper']
sales
```

Out[12]:

	Unnamed: 0	TV	Radio	Newspaper	Sales	Total Advert
0	1	230.1	37.8	69.2	22.1	337.1
1	2	44.5	39.3	45.1	10.4	128.9
2	3	17.2	45.9	69.3	9.3	132.4
3	4	151.5	41.3	58.5	18.5	251.3
4	5	180.8	10.8	58.4	12.9	250.0
...	...	...	...	...	...	...
195	196	38.2	3.7	13.8	7.6	55.7
196	197	94.2	4.9	8.1	9.7	107.2
197	198	177.0	9.3	6.4	12.8	192.7
198	199	283.6	42.0	66.2	25.5	391.8
199	200	232.1	8.6	8.7	13.4	249.4

200 rows × 6 columns

Machine Learning Model Training & Evaluation

In [13]:
```python
# Importing the libraries needed for machine learning models

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from math import sqrt
from tabulate import tabulate
```

In [14]:
```python
# Assigning the features and prediction columns to the X and y columns
X = sales.drop(columns='Sales')
y = sales['Sales']
```

In [15]:
```python
# Splitting the the sales dataset 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)

# Defining the models needed for the predictions
models = {
    'Linear Regression': LinearRegression(),
    'SVR': SVR(),
    'Gradient Boosting': GradientBoostingRegressor(),
    'Lasso': Lasso(alpha=1.0),
    'Ridge': Ridge(alpha=1.0)
}

# Models training and evaluation metrics
def train_and_evaluate(models, X_train, X_test, y_train, y_test):
    results = []
    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        mse = mean_squared_error(y_test, y_pred)
        rmse = sqrt(mse)
        mae = mean_absolute_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)
        results.append([name, mse, rmse, mae, r2])
    return results
```

Different Models Performance

In [20]:
```python
# Train and evaluate each model
results = train_and_evaluate(models, X_train, X_test, y_train, y_test)

# Create a tabular representation of results
table = tabulate(results, headers=['The Models', 'MSE', 'RMSE', 'MAE', 'R2'], tablefmt='grid')
print(table)
```

```
+-------------------+---------+----------+----------+----------+
| The Models        |     MSE |     RMSE |      MAE |       R2 |
+===================+=========+==========+==========+==========+
| Linear Regression | 3.21097 | 1.79192  | 1.46495  | 0.89827  |
+-------------------+---------+----------+----------+----------+
| SVR               | 5.16663 | 2.27302  | 1.74559  | 0.83631  |
+-------------------+---------+----------+----------+----------+
| Gradient Boosting | 0.9682  | 0.983972 | 0.776019 | 0.969325 |
+-------------------+---------+----------+----------+----------+
| Lasso             | 3.25628 | 1.80452  | 1.48934  | 0.896834 |
+-------------------+---------+----------+----------+----------+
| Ridge             | 3.19902 | 1.78858  | 1.46507  | 0.898649 |
+-------------------+---------+----------+----------+----------+
```

From the table above, It can be seen that the Gradient Boosting model did a better prediction with a much lower Mean Squared Error and Mean Absolute Error, However it also has a higher percentage of R-squared at about 97.0% which indicate the variability observed in the target variable is explained by the model and also indicates a better fit model as vmore of the data's variance are accounted for by the model

In [21]:
```python
#Importing random library
import random

random.seed(42)
random_samples = sales[['TV','Radio','Newspaper']].sample(10)

random_samples
```

Out[21]:

	TV	Radio	Newspaper
32	97.2	1.5	30.0
93	250.9	36.5	72.3
76	27.5	1.6	20.7
61	261.3	42.7	54.7
153	171.3	39.7	37.7
115	75.1	35.0	52.7
140	73.4	17.0	12.9
94	107.4	14.0	10.9
42	293.6	27.7	1.8
123	123.1	34.6	12.4

In [22]:
```
random_samples['Total Advert'] = random_samples['TV'] + random_samples['Radio'] + random_samples['Newspaper']
random_samples
```

Out[22]:

	TV	Radio	Newspaper	Total Advert
32	97.2	1.5	30.0	128.7
93	250.9	36.5	72.3	359.7
76	27.5	1.6	20.7	49.8
61	261.3	42.7	54.7	358.7
153	171.3	39.7	37.7	248.7
115	75.1	35.0	52.7	162.8
140	73.4	17.0	12.9	103.3
94	107.4	14.0	10.9	132.3
42	293.6	27.7	1.8	323.1
123	123.1	34.6	12.4	170.1

Evaluating Predicted Sales Against Actual Sales - Different Models Performance

In [30]:

```
# Select 10 random samples from the dataset
random_samples = sales.sample(n=10, random_state=np.random.RandomState())

# Prepare the data for prediction (excluding the 'Sales' column)
X_random_samples = random_samples.drop(columns='Sales')

# Actual sales values
actual_sales = random_samples['Sales'].values

# Make predictions with each model and store them in a list
predictions_table = []
for name, model in models.items():
    predicted_sales = model.predict(X_random_samples)
    for i in range(len(predicted_sales)):
        variance = actual_sales[i] - predicted_sales[i]
        predictions_table.append([name, i+1, actual_sales[i], predicted_sales[i], variance])

# Create a tabular representation of model, sample number, actual vs predicted values, and variance
headers = ['Model', 'Sample No.', 'Actual Sales', 'Predicted Sales', 'Variance']
table = tabulate(predictions_table, headers=headers, tablefmt='grid')

# Display the table
print("Table of the Models Used and their Performance:")
print(table)
```

Table of the Models Used and their Performance:

Model	Sample No.	Actual Sales	Predicted Sales	Variance
Linear Regression	1	6.7	6.25448	0.445516
Linear Regression	2	14.5	14.4779	0.0221217
Linear Regression	3	12.2	13.9004	-1.70044
Linear Regression	4	17	17.1824	-0.182408
Linear Regression	5	8.8	10.475	-1.67501
Linear Regression	6	9.5	9.03843	0.461573
Linear Regression	7	10.4	12.4931	-2.09306
Linear Regression	8	14.7	15.7799	-1.07991
Linear Regression	9	7.2	6.29077	0.909232
Linear Regression	10	9.6	7.65784	1.94216
SVR	1	6.7	7.89283	-1.19283
SVR	2	14.5	15.0218	-0.521838
SVR	3	12.2	14.7807	-2.5807
SVR	4	17	16.8549	0.145105
SVR	5	8.8	8.55771	0.242288
SVR	6	9.5	8.20474	1.29526
SVR	7	10.4	10.6449	-0.2449
SVR	8	14.7	13.1217	1.57833
SVR	9	7.2	7.57459	-0.374586
SVR	10	9.6	10.5751	-0.975139
Gradient Boosting	1	6.7	6.78632	-0.0863159
Gradient Boosting	2	14.5	14.8166	-0.316593

Gradient Boosting	3	12.2	12.3519	-0.151924
Gradient Boosting	4	17	16.9582	0.0418414
Gradient Boosting	5	8.8	8.88596	-0.0859593
Gradient Boosting	6	9.5	8.31065	1.18935
Gradient Boosting	7	10.4	10.5965	-0.196451
Gradient Boosting	8	14.7	14.5895	0.110548
Gradient Boosting	9	7.2	7.26327	-0.0632747
Gradient Boosting	10	9.6	9.44433	0.155672
Lasso	1	6.7	6.26676	0.433236
Lasso	2	14.5	14.5442	-0.0441551
Lasso	3	12.2	13.9872	-1.78722
Lasso	4	17	17.1107	-0.110708
Lasso	5	8.8	10.3585	-1.55846
Lasso	6	9.5	8.91	0.589997
Lasso	7	10.4	12.4514	-2.05139
Lasso	8	14.7	15.6624	-0.962369
Lasso	9	7.2	6.3551	0.844904
Lasso	10	9.6	7.81319	1.78681
Ridge	1	6.7	6.22629	0.473713
Ridge	2	14.5	14.4732	0.026836
Ridge	3	12.2	13.9001	-1.70012
Ridge	4	17	17.1931	-0.193127
Ridge	5	8.8	10.456	-1.65595
Ridge	6	9.5	9.02048	0.479522

```
| Ridge             |             7 |           10.4 |         12.4727  | -2.07266   |
+-------------------+---------------+----------------+------------------+------------+
| Ridge             |             8 |           14.7 |         15.7595  | -1.05954   |
+-------------------+---------------+----------------+------------------+------------+
| Ridge             |             9 |            7.2 |          6.26562 |  0.934376  |
+-------------------+---------------+----------------+------------------+------------+
| Ridge             |            10 |            9.6 |          7.64974 |  1.95026   |
+-------------------+---------------+----------------+------------------+------------+
```

Summation of Variances to Determine Best Performing Model

In [31]:
```
# Make predictions with each model and calculate the variance
model_variances = {}
for name, model in models.items():
    predicted_sales = model.predict(X_random_samples)
    variance = actual_sales - predicted_sales
    model_variances[name] = np.sum(variance)

# Create a tabular representation of model and sum of variances
variance_table = [[name, np.sum(var)] for name, var in model_variances.items()]
headers = ['Models', 'Sum of Variance']
table = tabulate(variance_table, headers=headers, tablefmt='grid')

# Display the table
print("Table of Sum of Variance per Model:")
print(table)
```

```
Table of Sum of Variance per Model:
+-------------------+-------------------+
| Models            |   Sum of Variance |
+===================+===================+
| Linear Regression |         -2.95022  |
+-------------------+-------------------+
| SVR               |         -2.62902  |
+-------------------+-------------------+
| Gradient Boosting |          0.596899 |
+-------------------+-------------------+
| Lasso             |         -2.85935  |
+-------------------+-------------------+
| Ridge             |         -2.8167   |
+-------------------+-------------------+
```

Consistently, Gradient Boosting model has the least sum of variances which makes it the best performing model in this project.