Name- Kore Pratiksha Jayant

Oasis Infobyte (Data Science)

Task-5 SALES PREDICTION USING PYTHON

INTRODUCTION- In this notebook, we prefer to employ machine learning for predicting sales based on advertising expenditure, providing valuable insights for businesses optimizing marketing strategies.

Importing necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
```

Loding Dataset

In [2]:
```
data = pd.read_csv('C:\\Users\\stati\\OneDrive\\Desktop\\Advertising.csv')
```

EDA (Exploratory Data Analysis)

In [3]:
```
# To check first few rows of the dataframe
(data.head())
```

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

In [4]:
```
# To get information about the dataframe
(data.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
```

In [5]:
```
# Displaying a random sample of 5 rows
data.sample(5)
```

Out[5]:

	Unnamed: 0	TV	Radio	Newspaper	Sales
112	113	175.7	15.4	2.4	14.1
107	108	90.4	0.3	23.2	8.7
80	81	76.4	26.7	22.3	11.8
164	165	117.2	14.7	5.4	11.9
3	4	151.5	41.3	58.5	18.5

In [6]:
```
# Dropping the "Unnamed: 0" column
data = data.drop(columns="Unnamed: 0", axis=1)
```

In [7]:
```
# Displaying another random sample of 2 rows
data.sample(2)
```

Out[7]:

	TV	Radio	Newspaper	Sales
188	286.0	13.9	3.7	15.9
164	117.2	14.7	5.4	11.9

In [8]:
```
# Duplicate Code - Remove redundant data.info() calls
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

In [9]:
```
data.duplicated().sum()
```

Out[9]:
```
0
```

In [10]:
```
data.shape
```

Out[10]:
```
(200, 4)
```

In [11]:
```
data.isnull().sum()
```

Out[11]:
```
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

In [12]:
```
data.head()
```

Out[12]:

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

In [13]:
```
data.describe()
```

Out[13]:

	TV	Radio	Newspaper	Sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	14.022500
std	85.854236	14.846809	21.778621	5.217457
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	10.375000
50%	149.750000	22.900000	25.750000	12.900000
75%	218.825000	36.525000	45.100000	17.400000
max	296.400000	49.600000	114.000000	27.000000

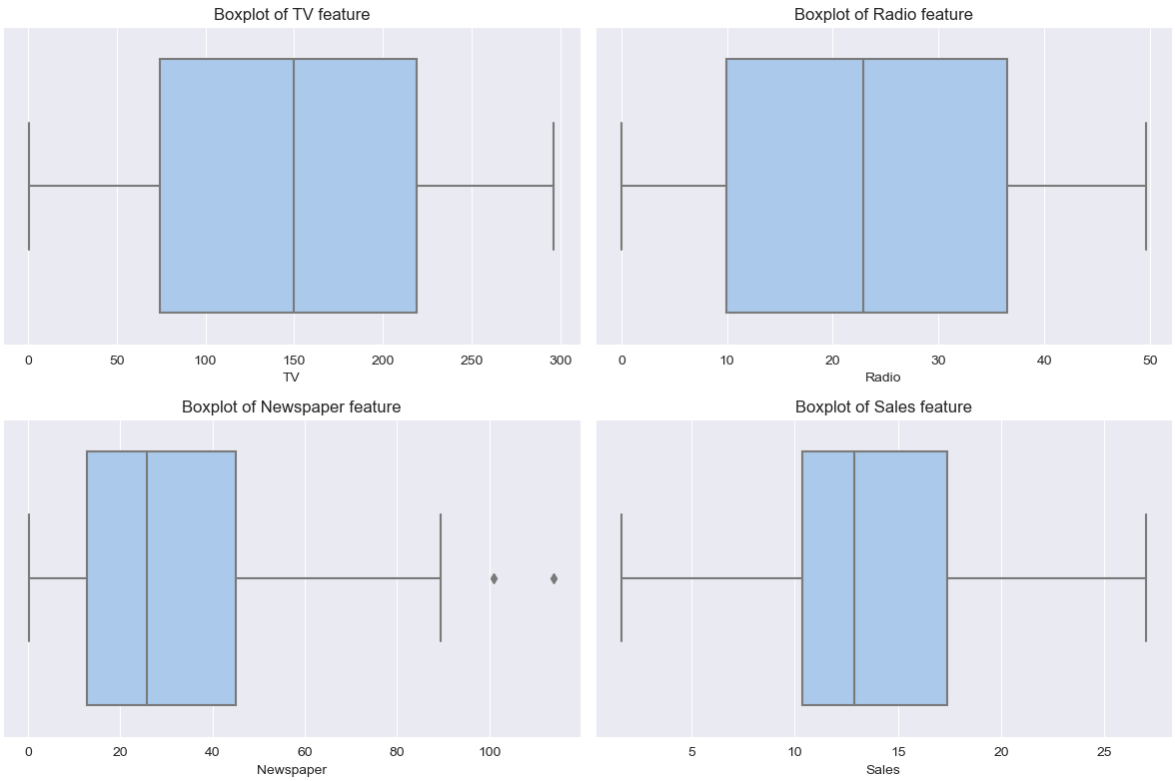
Visualization

In [14]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
```

BOXPLOT

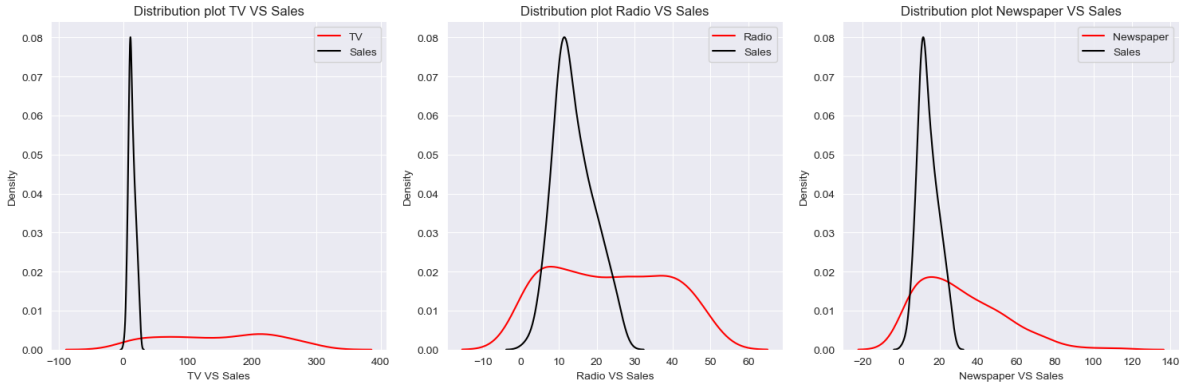
In [15]:
```
# Creating boxplots for each feature

plt.figure(figsize=(12, 8))
columns = {0: "TV", 1: "Radio", 2: "Newspaper", 3: "Sales"}

for plot, col_name in columns.items():
```

```
    plt.subplot(2, 2, plot + 1)
    sns.boxplot(x=data[col_name], palette="pastel")
    plt.xlabel(col_name)
    plt.title("Boxplot of {} feature".format(col_name))

plt.tight_layout()
plt.show()
```



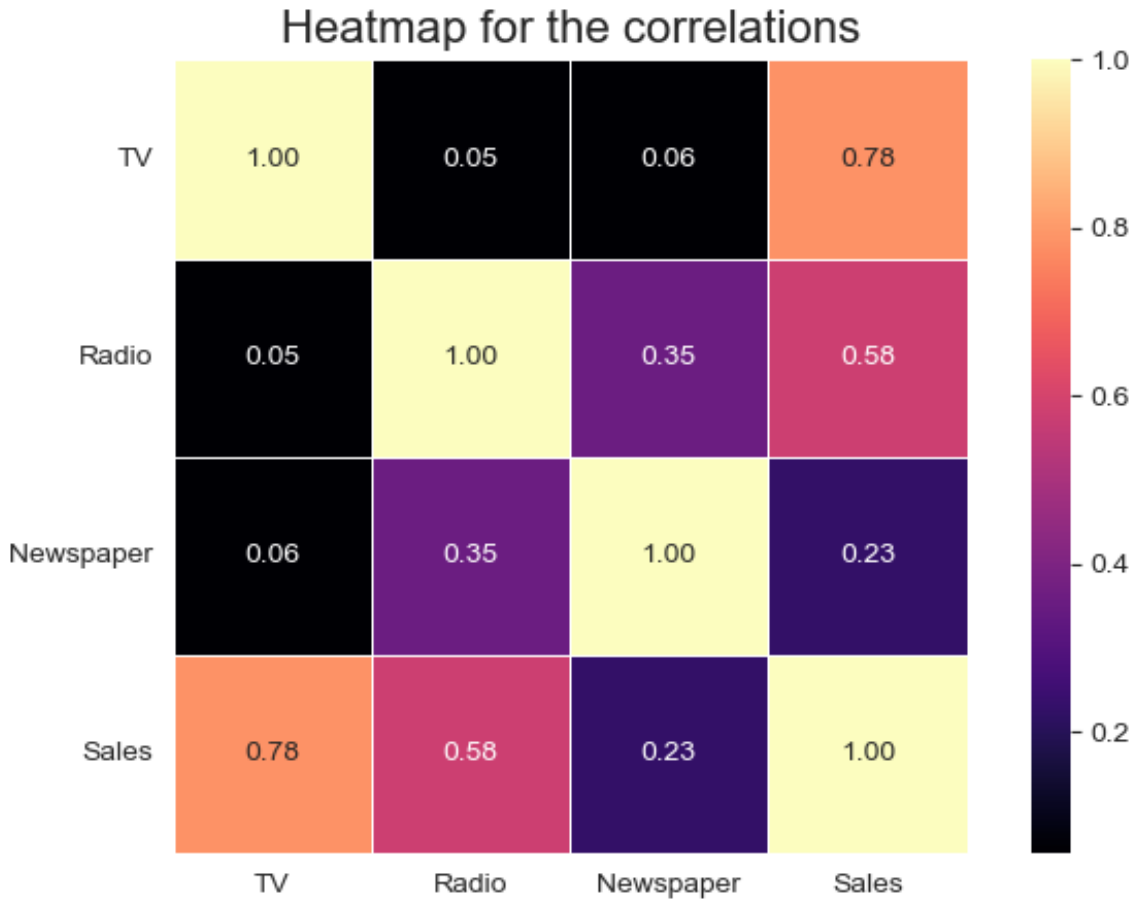
DISTRIBUTION PLOT

In [16]:
```
# Creating distribution plots for each feature

plt.figure(figsize=(15, 5))
columns = ["TV", "Radio", "Newspaper"]

for i, col_name in enumerate(columns, 1):
    plt.subplot(1, 3, i)
    sns.distplot(data[col_name], hist=False, label=col_name, color="red")
    sns.distplot(data["Sales"], hist=False, label="Sales", color="black")
    plt.xlabel("{} VS Sales".format(col_name))
    plt.title("Distribution plot {} VS Sales".format(col_name))
    plt.legend()

plt.tight_layout()
plt.show()
```



HEATMAP

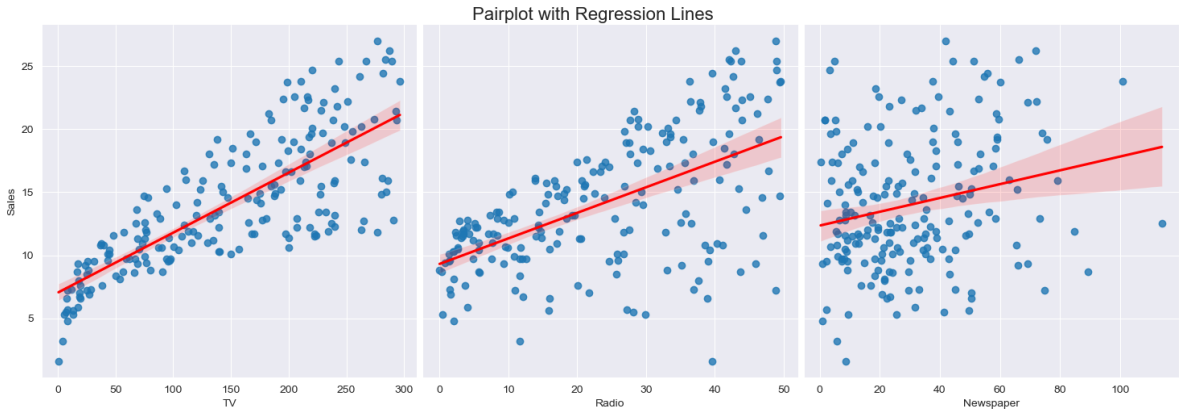
In [17]:
```
# Creating a heatmap for correlations
plt.figure(figsize=(8,5))

# Customize the heatmap
heatmap = sns.heatmap(data.corr(), annot=True, cmap="magma"
                      , linewidths=0.5, fmt=".2f", square=True)

# Set the title
plt.title("Heatmap for the correlations", fontsize=16)

# Rotate y-axis labels for better readability
heatmap.set_yticklabels(heatmap.get_yticklabels(), rotation=0)

plt.show()
```



PAIRPLOT

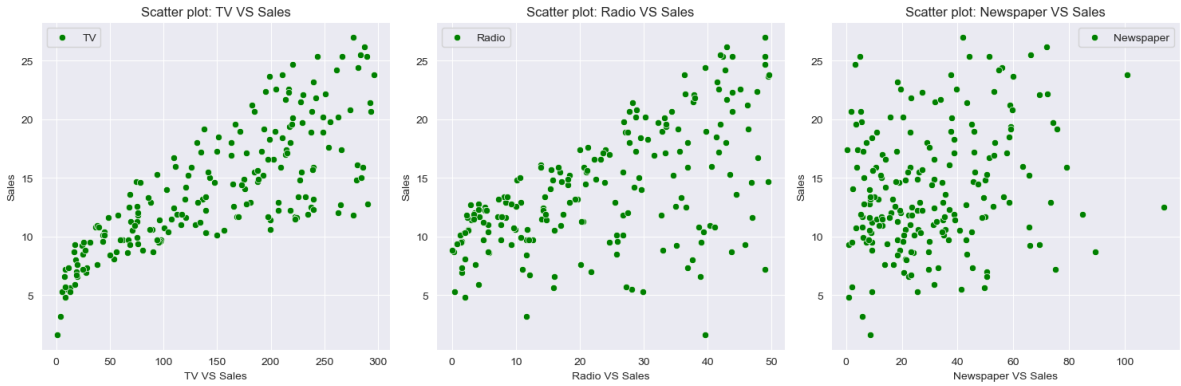
In [18]:
```
# Creating pairplots with regression lines
plt.figure(figsize=(12, 6))

# Use pairplot with regression lines, custom color, and size
sns.pairplot(data=data, x_vars=["TV", "Radio", "Newspaper"],
             y_vars="Sales", kind="reg",
             height=5, plot_kws={'line_kws':{'color':'red'}})

# Set title
plt.suptitle("Pairplot with Regression Lines", y=1.02, fontsize=16)

plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



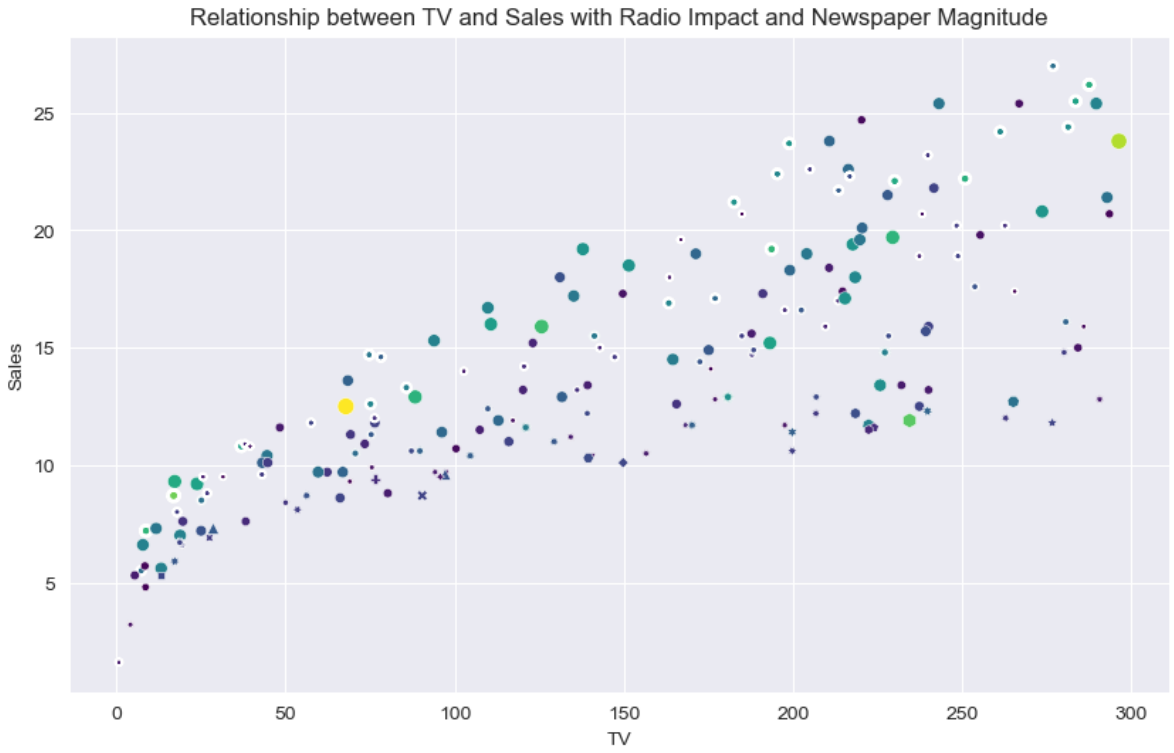
SCATTERED PLOT

In [19]:
```
# Creating scatter plots for each feature
plt.figure(figsize=(15, 5))
columns = {0: "TV", 1: "Radio", 2: "Newspaper"}


for plot, col_name in columns.items():
    plt.subplot(1, 3, plot + 1)
    sns.scatterplot(x=data[col_name], y=data["Sales"],
                    label=col_name, color="green")
    plt.xlabel("{} VS Sales".format(col_name))
    plt.title("Scatter plot: {} VS Sales".format(col_name))
    plt.legend()

plt.tight_layout()
plt.show()
```



ADVANCED SCATTERED PLOT

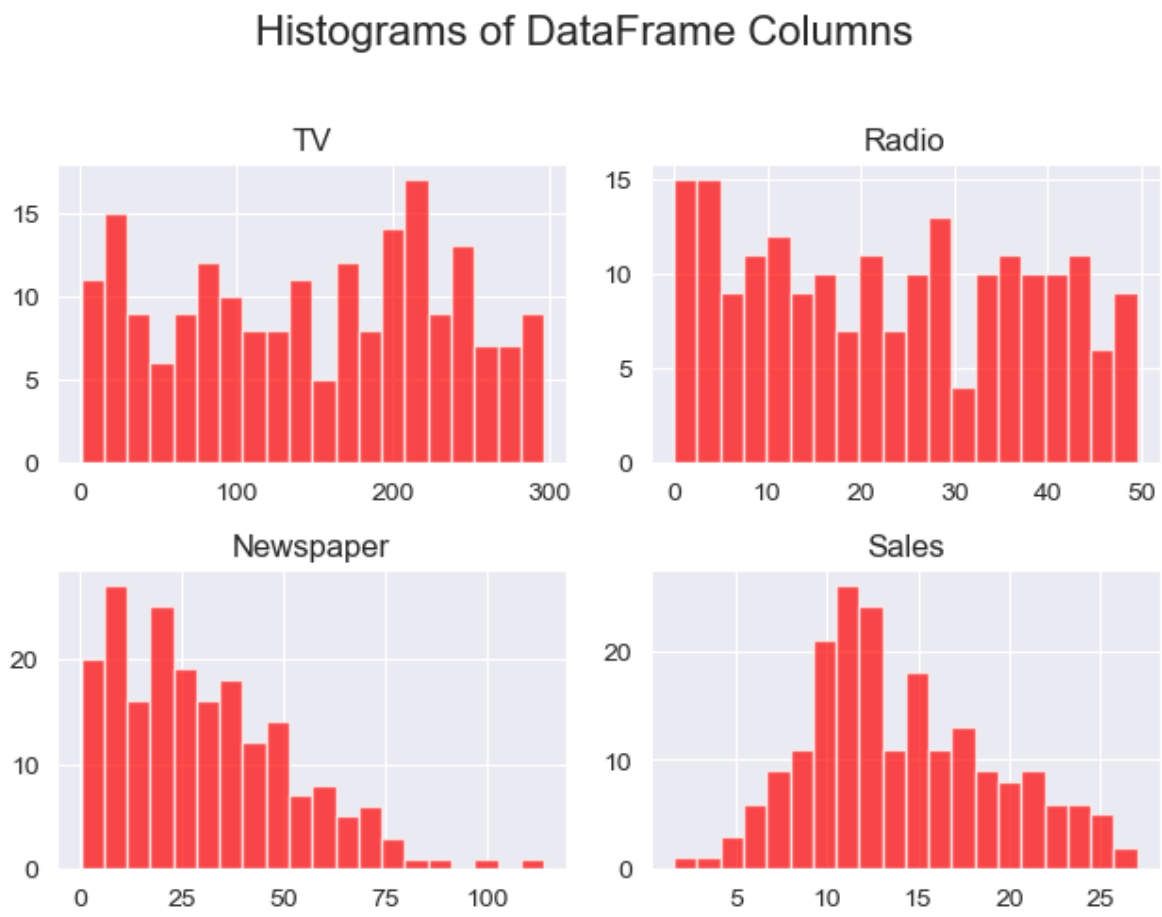
In [20]:
```
# Creating a scatter plot with additional styling
plt.figure(figsize=(10, 6))

# Use scatterplot with style, size, and color
sns.scatterplot(data=data, x="TV", y="Sales", style="Radio",
                size="Newspaper", hue="Newspaper",
                palette="viridis", legend=False)

# Set labels and title
plt.xlabel("TV")
plt.ylabel("Sales")
plt.title("Relationship between TV and Sales with Radio Impact and Newspaper Mag

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



HISTOGRAM PLOT

In [21]:
```
# Creating histograms for all columns in the DataFrame
plt.figure(figsize=(5, 20))

# Plot histograms for all columns in the DataFrame
data.hist(bins=20, color="red", edgecolor="white", alpha=0.7)

# Set title and labels
plt.suptitle("Histograms of DataFrame Columns", y=1.02, fontsize=16)
plt.tight_layout()
plt.show()
```

```
<Figure size 500x2000 with 0 Axes>
```



Data Processing

LABEL ENCODER

In [22]:
```
X = data.drop(columns="Sales",axis=1)
```

In [23]:
```
Y = data["Sales"]
```

In [24]:
```
from sklearn.preprocessing import MinMaxScaler,LabelEncoder,StandardScaler

le = LabelEncoder()
y= le.fit_transform(Y)
```

In [25]:
```
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)
```

MINMAX SCALER

In [26]:
```
scaler = MinMaxScaler()
scaler.fit(X_train,Y_train)
```

Out[26]:
```
▾ MinMaxScaler
MinMaxScaler()
```

In [27]:
```
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Model Training

In [28]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
```

In [29]:
```
models = [
    ("Linear Regression", LinearRegression()),
    ("Ridge Regression", Ridge()),
    ("Lasso Regression", Lasso())
]

# Looping through each model and training it
for name, model in models:
    model.fit(X_train_scaled, Y_train)
```

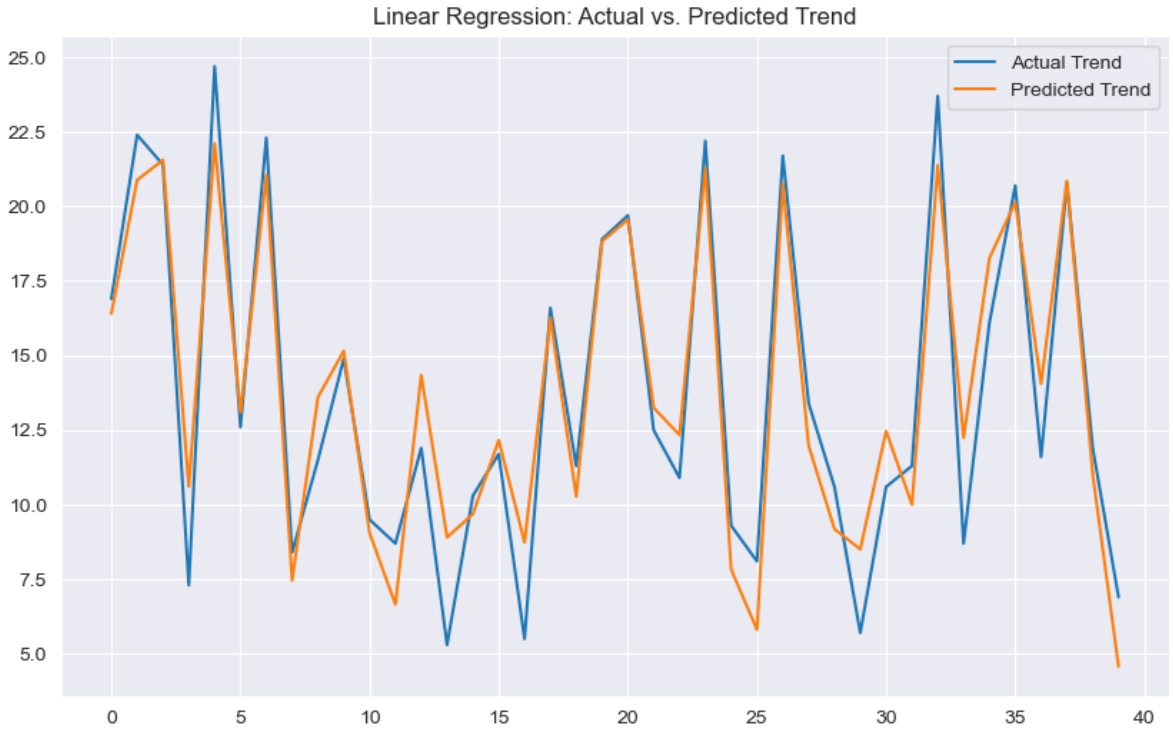
Model Evaluation

LINE PLOT

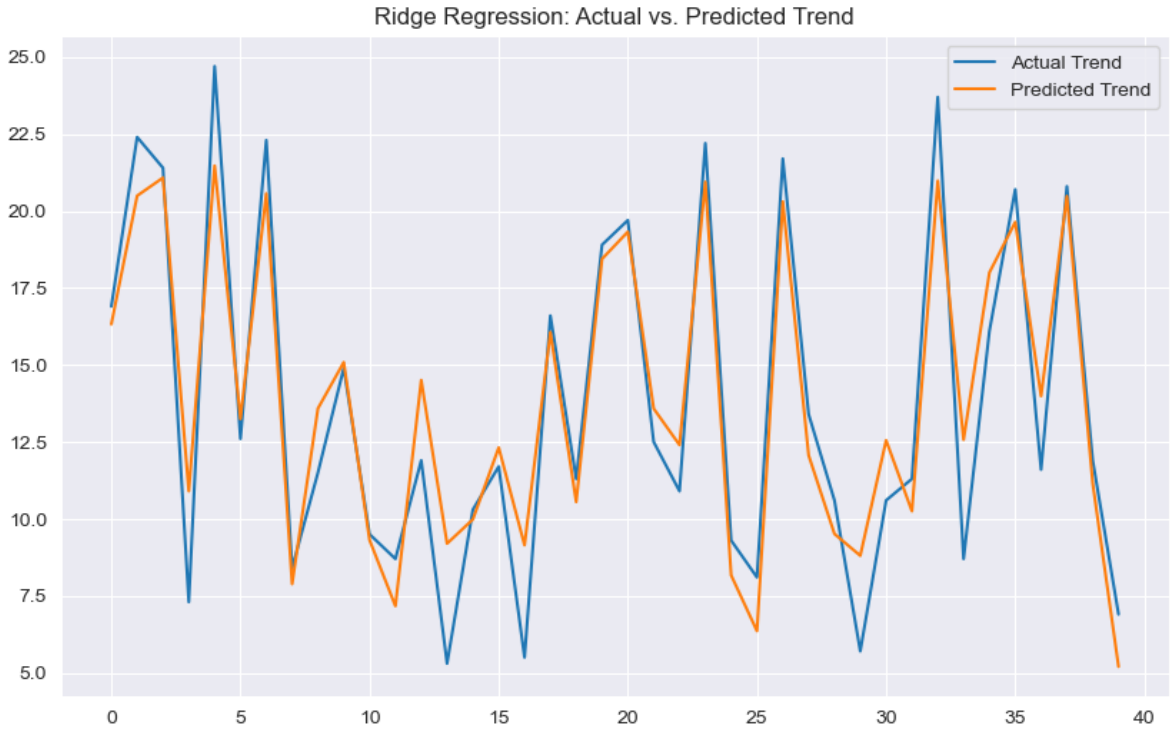
In [30]:
```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
```

In [31]:
```
# Looping through each model for evaluation
for name, model in models:
    Y_pred = model.predict(X_test_scaled)

    # Evaluate the model
    mse = mean_squared_error(Y_test, Y_pred)
    r2 = r2_score(Y_test, Y_pred)
    cv_scores = cross_val_score(
        model, X_train_scaled, Y_train, cv=5, scoring='r2')

    # Print evaluation metrics
    print("Model: {}".format(name))
    print("Mean Squared Error: {}".format(mse))
    print("R2 Score: {}".format(r2))
    print("Cross-Validation R2: {}".format(cv_scores.mean()))

    # Plot actual vs. predicted trend
    plt.figure(figsize=(10, 6))
    plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
    plt.plot(np.arange(len(Y_test)), Y_pred, label='Predicted Trend')
    plt.title(f'{name}: Actual vs. Predicted Trend')
    plt.legend()
    plt.show()
```

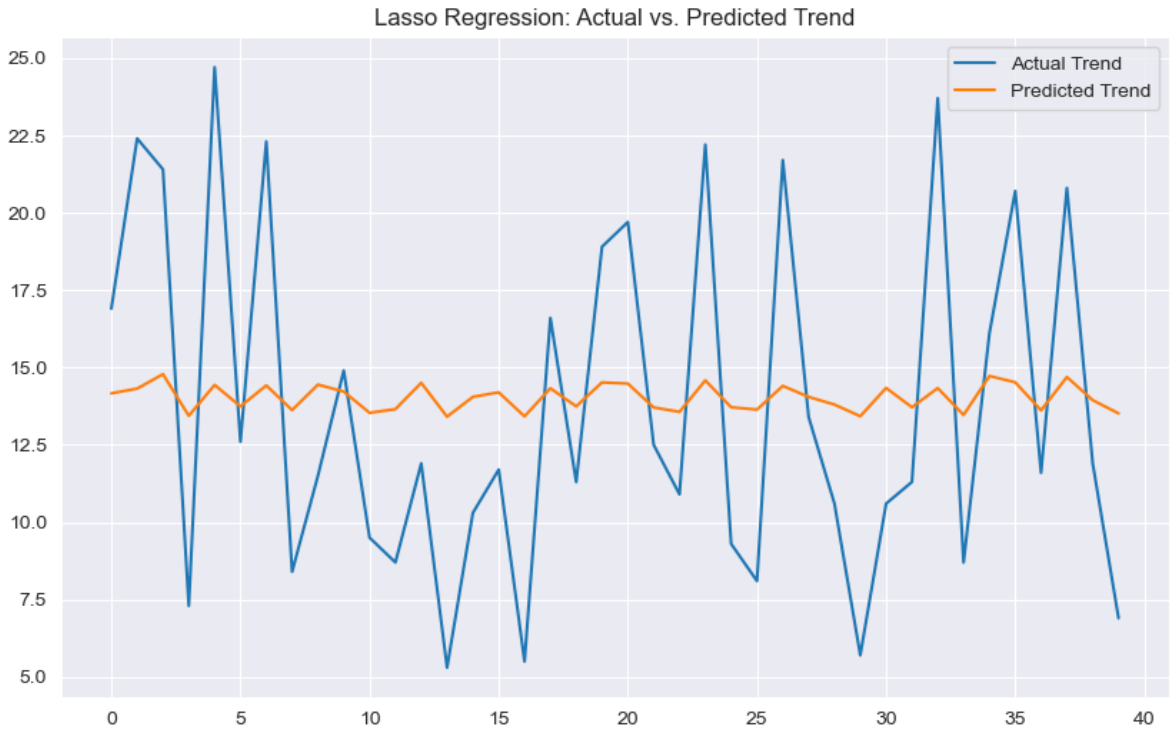
Model: Linear Regression
Mean Squared Error: 3.1740973539761046
R2 Score: 0.899438024100912
Cross-Validation R2: 0.8594884313276511



Model: Ridge Regression
Mean Squared Error: 3.500676810829309
R2 Score: 0.889091310750085
Cross-Validation R2: 0.8546283131951761



Model: Lasso Regression
Mean Squared Error: 27.833508953208344
R2 Score: 0.11817680921684193
Cross-Validation R2: -0.07480543323377124



From the model training results, it's clear that Linear Regression and Ridge Regression work well for our data. However, Lasso Regression shows higher prediction errors (MSE) and a less accurate fit (lower R2 Score).

Conclusion

After checking different models for predicting sales from advertising data, we found:

Linear Regression:

Good performance

Low prediction errors (MSE: 3.17)

Accurate fit (R2 Score: 0.90)

Consistent results in cross-validation (R2: 0.86)

Ridge Regression:

Performs well

Slightly higher prediction errors than Linear Regression (MSE: 3.50)

Good fit (R2 Score: 0.89)

Reasonable cross-validation results (R2: 0.85)

Lasso Regression:

Poor performance

Higher prediction errors (MSE: 27.83)

Less accurate fit (R2 Score: 0.12)

Negative cross-validation results (R2: -0.07), indicating unsuitability for the dataset.

In summary, both Linear Regression and Ridge Regression work well, but Linear Regression is slightly better due to lower errors and better accuracy. The choice between them depends on specific needs. For predicting sales in this advertising dataset, we recommend using the Linear Regression model.

Model Testing

In [32]:
```
# Train the Linear Regression model
Lr = LinearRegression().fit(X_train, Y_train)

# Input values for prediction
new_data = pd.DataFrame({"TV": [float(input("Enter the TV value: "))],
                         "Radio": [float(input("Enter the Radio value: "))],
                         "Newspaper":
                         [float(input("Enter the Newspaper value: "))]})

# Predict Sales
new_pred = Lr.predict(scaler.transform(new_data))

# Display the result
print("-------------------------------------")
print("Predicted Sales:", abs(new_pred))
```

```
Enter the TV value: 25
Enter the Radio value: 65
Enter the Newspaper value: 86
-------------------------------------
Predicted Sales: [3.23303234]
```

Tested on Following Details:

Enter the TV value: 25

Enter the Radio value: 65

Enter the Newspaper value: 86

END