FUTURE SALES PREDICTION

PROBLEM STATEMENT, DESIGN THINKING PROCESS, AND THE PHASES OF DEVELOPMENT.

Problem Statement:

Objective:

The objective of the project is to predict future sales based on advertising spending data. To achieve this goal, we aim to develop and evaluate various machine learning models and time series forecasting techniques. The project involves:

1. Problem Definition:

The project focuses on forecasting future sales by analyzing the impact of different advertising mediums (TV, Radio, and Newspaper) on sales.

The primary goal is to create predictive models that assist businesses in making informed marketing decisions.

2. Data Description:

Utilizing a dataset containing historical advertising spending data (TV, Radio, Newspaper) and corresponding sales figures.

The dataset allows us to explore the relationships between advertising expenditures and future sales.

3. Key Questions:

What is the influence of various advertising channels on future sales?

Can we accurately predict future sales based on historical advertising data?

Which machine learning models and time series forecasting techniques produce the most precise predictions?

Design Thinking Process:

Design Thinking Approach:

The project follows a design thinking approach, emphasizing user-centric solutions and iterative development.

1. Empathize:

Understanding the needs and challenges of businesses in optimizing their advertising budgets and forecasting future sales.

Identifying the key factors influencing future sales.

2. Define:

Defining the problem statement: Predict future sales based on historical advertising spending data.

Setting clear objectives for the project, such as creating accurate predictive models for future sales.

3. Ideate:

Generating ideas for data preprocessing, feature engineering, and modeling techniques.

Brainstorming different visualizations and metrics to evaluate model performance.

4. Prototype:

Developing machine learning models, including Linear Regression, Random Forest, Support Vector Machine, Neural Network (LSTM), XGBoost, and LightGBM.

Implementing time series forecasting methods (ARIMA, ETS, SARIMA).

Creating data visualizations to gain insights into the dataset.

5. Test:

Evaluating the models using regression metrics (MSE, MAE, R2).

Comparing the performance of different models to determine the most effective ones.

Phases of Development:

1. Data Preprocessing:

Loading and exploring the dataset.

Handling duplicates and missing values.

Calculating summary statistics (mean, median, mode) for data understanding.

2. Time Series Forecasting:

Applying ARIMA, ETS, and SARIMA models.

Visualizing original data and forecasts for comparison.

3. LSTM Model Development:

Designing and training a Long Short-Term Memory (LSTM) neural network.

Preparing data for LSTM model input.

Evaluating the LSTM model using mean squared error (MSE).

4. Data Visualization:

Utilizing various data visualization techniques (e.g., scatter plots, box plots, correlation matrices) to gain insights into the data.

5. Model Training and Evaluation:

Training machine learning models, including Linear Regression, Random Forest, Support Vector Machine, Neural Network, XGBoost, and LightGBM.

Evaluating model performance using regression metrics (MSE, MAE, R2).

Comparing the effectiveness of different models.

6. Model Selection and Evaluation Metrics:

Selecting the most suitable machine learning models and time series forecasting techniques.

Explaining the choice of evaluation metrics (MSE, MAE, R2).

Dataset Description:

Dataset Source:

The dataset used for the "Future Sales Prediction" project can be found on Kaggle using the following link: Future Sales Prediction Dataset.

The dataset consists of the following columns:

TV:

This column represents the amount of money spent on advertising through television. Television advertising is known for its wide reach and the potential to create awareness among a large and diverse audience. The values in this column indicate the advertising budget allocated to TV for different marketing campaigns or time periods.

Radio:

This attribute reflects the advertising expenditure on radio campaigns. Radio advertising is often used to target specific demographics or regions, and its effectiveness can vary based on factors such as the choice of radio stations and time slots for airing advertisements.

Newspaper:

This column measures the advertising spending on newspaper advertisements. Newspaper ads are typically employed for local or regional targeting and can come in various formats, sizes, and placements within the newspaper.

Sales:

This is the dependent variable in the dataset, representing the actual sales figures achieved during or after each advertising campaign. The primary objective of analyzing this data is to understand how the budgets allocated to TV, Radio, and Newspaper advertising impact sales, allowing businesses to make informed marketing decisions.

Data Preprocessing Steps:

Loading the Dataset:

The dataset is loaded using libraries such as Pandas and stored in a Pandas DataFrame.

Exploratory Data Analysis (EDA):

Summary statistics, including mean, median, and mode, are calculated for each column (TV, Radio, Newspaper, Sales) to gain initial insights into the data.

Handling Duplicates and Missing Values:

Duplicate rows are identified and handled to ensure data integrity.

Missing values, if present, are addressed through data imputation or removal, depending on the extent of missing data.

Data Visualization:

Data is visualized using various techniques, including scatter plots, box plots, density plots, and correlation matrices. These visualizations help in understanding the relationships between variables and identifying potential outliers.

Model Training Process:
Machine Learning Models:

A variety of machine learning models are used, including Linear Regression, Random Forest, Support Vector Machine, Neural Network (LSTM), XGBoost, and LightGBM. These models allow for different approaches to predict future sales based on advertising spending.

Time Series Forecasting:

Time series forecasting is performed using ARIMA, ETS, and SARIMA models. These models are suitable for handling time-dependent data, allowing for the prediction of future sales values.

LSTM Model Development:

A Long Short-Term Memory (LSTM) neural network model is designed and trained for time series forecasting. The data is reshaped and prepared for LSTM model input.

Model Evaluation:

All models are evaluated using regression metrics, including Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R2) scores. These metrics provide insight into how well each model performs in predicting future sales.

Model Comparison:

The performance of different machine learning models and time series forecasting techniques is compared to select the most effective ones for future sales prediction.

Data Visualization :

Data visualization techniques are used to analyze and present the results, showcasing the effectiveness of the chosen models and their predictions.
By following these data preprocessing and model training steps, the project aims to accurately predict future sales based on historical advertising spending data and provide businesses with valuable insights for marketing decisions.

Choice of Time Series Forecasting Algorithm:

In the "Future Sales Prediction" project, three time series forecasting algorithms are employed: ARIMA, ETS (Exponential Smoothing), and SARIMA. The choice of these algorithms is based on their suitability for handling time-dependent data and their proven effectiveness in time series forecasting:

ARIMA (AutoRegressive Integrated Moving Average):

ARIMA is a widely used time series forecasting technique that combines autoregressive (AR) and moving average (MA) components with differencing to stabilize non-stationary time series data.
ARIMA is suitable when there is evidence of autocorrelation and seasonality in the data. It allows for forecasting future sales values based on their historical patterns and trends.

ETS (Exponential Smoothing):

ETS is a method that uses weighted averages of past observations to forecast future sales. It includes components for error, trend, and seasonality.
ETS is effective when there is a need to capture the impact of seasonality and trends in the data.
This method is especially useful for smoothing out noise in the time series data.

SARIMA (Seasonal AutoRegressive Integrated Moving Average):

SARIMA is an extension of ARIMA that accounts for seasonality. It adds seasonal autoregressive (SAR) and seasonal moving average (SMA) components to the ARIMA model.
SARIMA is a good choice when there is a strong seasonal pattern in the data, as it can capture both short-term and long-term fluctuations.
It provides accurate forecasts when seasonality plays a significant role in sales fluctuations.
The choice of the time series forecasting algorithm depends on the characteristics of the sales data.

Evaluation Metrics for Model Performance:

The performance of the machine learning models and time series forecasting algorithms is evaluated using several regression metrics. These metrics provide insights into how well the models are performing in predicting future sales:

Mean Squared Error (MSE):

MSE measures the average of the squared differences between the predicted and actual values. It penalizes large errors, making it sensitive to outliers.
A lower MSE indicates better model performance.

Mean Absolute Error (MAE):

MAE calculates the average absolute differences between the predicted and actual values. It is less sensitive to outliers compared to MSE.
A lower MAE suggests a better fit of the model to the data.

R-squared (R2) Score:

R2 measures the proportion of the variance in the dependent variable (sales) that is predictable from the independent variables (advertising spending).
An R2 score closer to 1 indicates a good fit, while a score near 0 suggests poor model performance.
These evaluation metrics are commonly used in regression analysis and time series forecasting to assess the accuracy and reliability of predictions.

PROGRAM:

```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
data = pd.read_csv('Sales.csv')
X = data[['TV', 'Radio', 'Newspaper']]
y = data['Sales']
from sklearn.model_selection import train_test_split
X = data[['TV', 'Radio', 'Newspaper']]
y = data['Sales']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(data.head())
print(data.describe())
print(data.info())
```

OUTPUT:

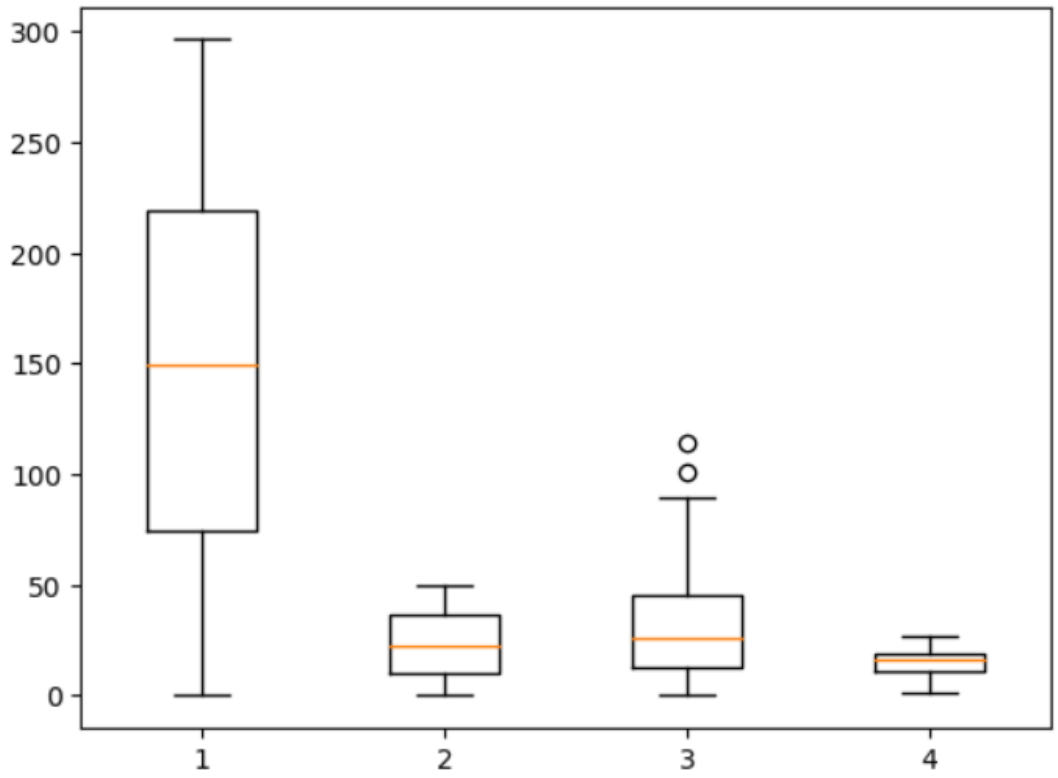
```
      TV  Radio  Newspaper  Sales
0  230.1   37.8       69.2   22.1
1   44.5   39.3       45.1   10.4
2   17.2   45.9       69.3   12.0
3  151.5   41.3       58.5   16.5
4  180.8   10.8       58.4   17.9
                TV       Radio   Newspaper       Sales
count  200.000000  200.000000  200.000000  200.000000
mean   147.042500   23.264000   30.554000   15.130500
std     85.854236   14.846809   21.778621    5.283892
min      0.700000    0.000000    0.300000    1.600000
25%     74.375000    9.975000   12.750000   11.000000
50%    149.750000   22.900000   25.750000   16.000000
75%    218.825000   36.525000   45.100000   19.050000
max    296.400000   49.600000  114.000000   27.000000
<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
None
```

PROGRAM:

```
print("The sum of duplicated items:",data.duplicated().sum())
print("\n")
print("To check the null values:",data.isna().sum())
print("\n")
print("boxplot after treating outliers",plt.boxplot(data))
print("\n")
```

OUTPUT:

```
The sum of duplicated items: 0


To check the null values: TV          0
Radio        0
Newspaper    0
Sales        0
dtype: int64


boxplot after treating outliers {'whiskers': [<matplotlib.lines.Line2D object at 0
```



PROGRAM:

```
print("MEAN")
print("TV mean:",data["TV"].mean())
print("Radio mean:",data["Radio"].mean())
print("Newspaper mean:",data["Newspaper"].mean())
```

```
print("Sales mean:",data["Sales"].mean())
print("\n")
print("Median")
print("Median of TV : ",data["TV"].median())
print("Median of Radio : ",data["Radio"].median())
print("Median of Newspaper : ",data["Newspaper"].median())
print("Median of Sales : ",data["Sales"].median())
print("\n")
print("Mode")
print("Mode of TV: ",data["TV"].mode())
print("Mode of Radio: ",data["Radio"].mode())
print("Mode of Newspaper: ",data["Newspaper"].mode())
print("Mode of Sales: ",data["Sales"].mode())
```

OUTPUT:

```
MEAN
TV mean: 147.0425
Radio mean: 23.264000000000006
Newspaper mean: 30.553999999999995
Sales mean: 15.130500000000001


Median
Median of TV :  149.75
Median of Radio :  22.9
Median of Newspaper :  25.75
Median of Sales :  16.0


Mode
Mode of TV:  0     17.2
1     76.4
2    109.8
3    177.0
4    184.9
5    197.6
6    199.8
7    222.4
8    237.4
9    240.1
Name: TV, dtype: float64
Mode of Radio:  0    4.1
1    5.7
Name: Radio, dtype: float64
Mode of Newspaper:  0     8.7
1     9.3
2    25.6
Name: Newspaper, dtype: float64
Mode of Sales:  0    11.9
1    16.7
Name: Sales, dtype: float64
```

PROGRAM:

```
print("Frequency Table")
print("Frequency Table of TV: ",data["TV"].value_counts())
print("Frequency Table of Radio: ",data["Radio"].value_counts())
print("Frequency Table of Newspaper: ",data["Newspaper"].value_counts())
print("Frequency Table of Sales: ",data["Sales"].value_counts())
```

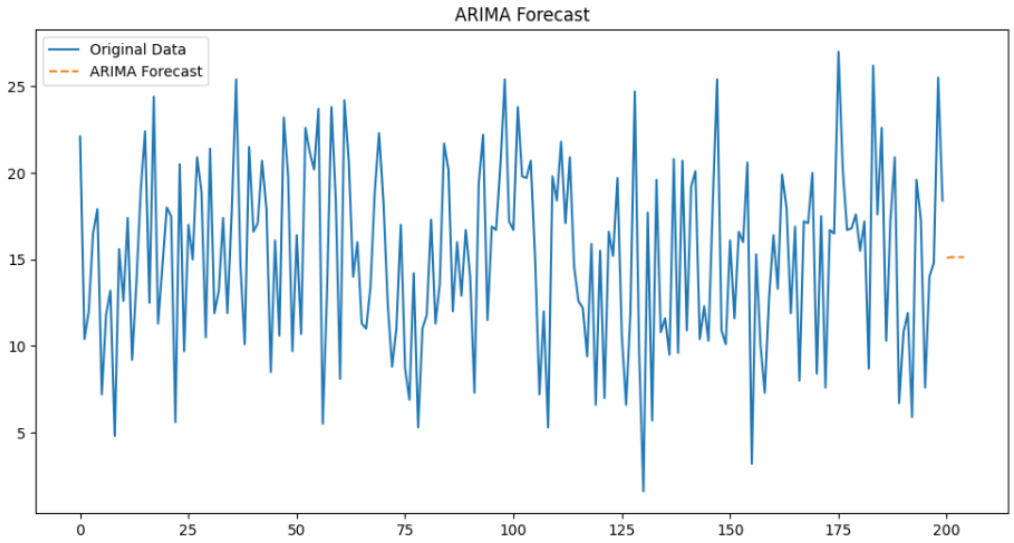
OUTPUT:

```
Frequency Table
Frequency Table of TV:  199.8    2
109.8    2
17.2     2
177.0    2
222.4    2
        ..
139.3    1
216.8    1
199.1    1
26.8     1
232.1    1
Name: TV, Length: 190, dtype: int64
Frequency Table of Radio:  4.1     3
5.7     3
13.9    2
14.3    2
36.9    2
       ..
42.8    1
14.5    1
30.6    1
33.0    1
8.6     1
Name: Radio, Length: 167, dtype: int64
Frequency Table of Newspaper:  9.3     3
25.6    3
8.7     3
34.6    2
8.5     2
       ..
27.2    1
31.7    1
19.3    1
31.3    1
66.2    1
Name: Newspaper, Length: 172, dtype: int64
Frequency Table of Sales:  11.9    5
16.7    5
20.7    4
11.0    3
11.3    3
       ..
13.4    1
24.2    1
8.1     1
5.5     1
25.5    1
Name: Sales, Length: 121, dtype: int64
```

PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.holtwinters import ExponentialSmoothing

data = pd.read_csv('Sales.csv')
sales_data = data['Sales']

arima_model = ARIMA(sales_data, order=(1, 1, 1))
arima_fit = arima_model.fit()
arima_forecast = arima_fit.forecast(steps=5)

plt.figure(figsize=(12, 6))
plt.plot(sales_data, label='Original Data')
plt.plot(arima_forecast, label='ARIMA Forecast', linestyle='--')
plt.legend()
plt.title('ARIMA Forecast')
plt.show()
```

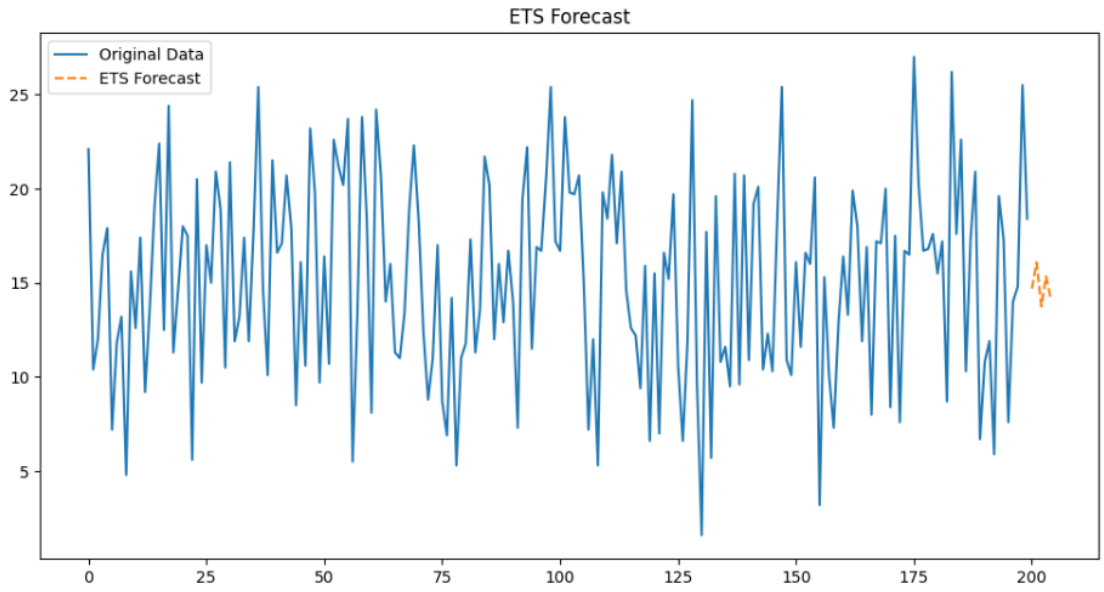
OUTPUT:



PROGRAM:

```
ets_model = ExponentialSmoothing(sales_data, trend='add', seasonal='add',
seasonal_periods=6)
ets_fit = ets_model.fit()
ets_forecast = ets_fit.forecast(steps=5)

plt.figure(figsize=(12, 6))
plt.plot(sales_data, label='Original Data')
plt.plot(ets_forecast, label='ETS Forecast', linestyle='--')
plt.legend()
plt.title('ETS Forecast')
plt.show()
```

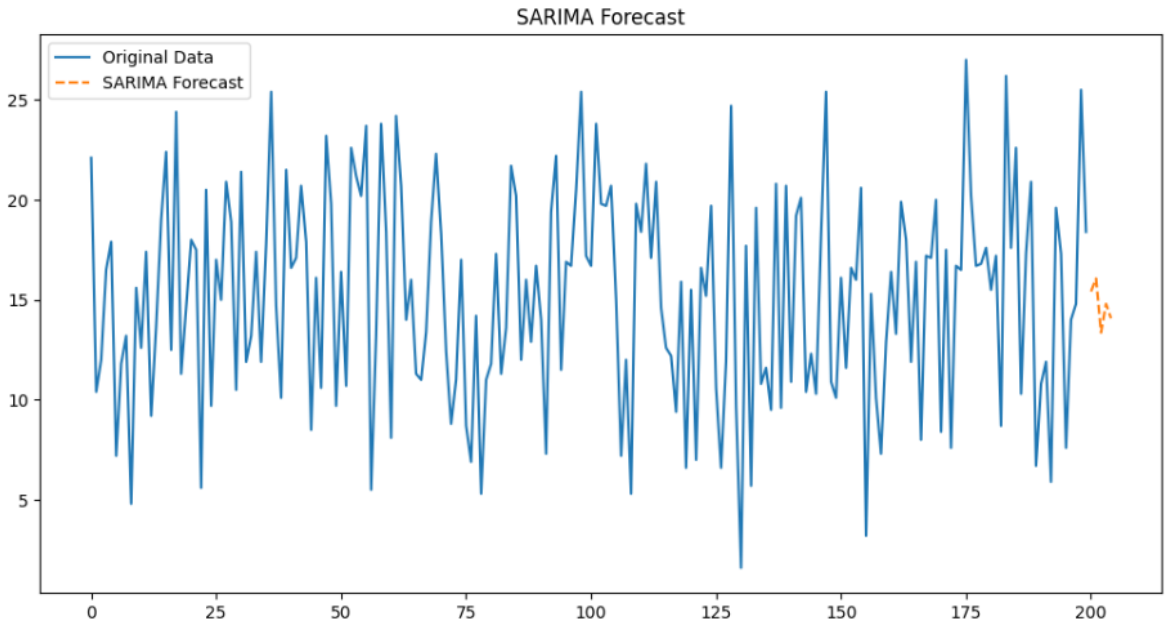
OUTPUT:



PROGRAM:

```
sarima_model = SARIMAX(sales_data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 12))
sarima_fit = sarima_model.fit()
sarima_forecast = sarima_fit.forecast(steps=5)

plt.figure(figsize=(12, 6))
plt.plot(sales_data, label='Original Data')
plt.plot(sarima_forecast, label='SARIMA Forecast', linestyle='--')
plt.legend()
plt.title('SARIMA Forecast')
plt.show()
```

OUTPUT:



PROGRAM:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(X_train.shape[1], 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
X_train_lstm = X_train.values.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test_lstm = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1)
model.fit(X_train_lstm, y_train, epochs=50, batch_size=32)
loss = model.evaluate(X_test_lstm, y_test)
```

OUTPUT:

```
5/5 [==============================] - 0s 4ms/step - loss: 1.7784
Epoch 28/50
5/5 [==============================] - 0s 4ms/step - loss: 1.7432
Epoch 29/50
5/5 [==============================] - 0s 4ms/step - loss: 1.7363
Epoch 30/50
5/5 [==============================] - 0s 4ms/step - loss: 1.7552
Epoch 31/50
5/5 [==============================] - 0s 5ms/step - loss: 1.7556
Epoch 32/50
5/5 [==============================] - 0s 4ms/step - loss: 1.6674
Epoch 33/50
5/5 [==============================] - 0s 4ms/step - loss: 1.7017
Epoch 34/50
5/5 [==============================] - 0s 4ms/step - loss: 1.6775
Epoch 35/50
5/5 [==============================] - 0s 5ms/step - loss: 1.6266
Epoch 36/50
5/5 [==============================] - 0s 4ms/step - loss: 1.7230
Epoch 37/50
5/5 [==============================] - 0s 4ms/step - loss: 1.6977
Epoch 38/50
5/5 [==============================] - 0s 4ms/step - loss: 1.6316
Epoch 39/50
5/5 [==============================] - 0s 4ms/step - loss: 1.6688
Epoch 40/50
5/5 [==============================] - 0s 4ms/step - loss: 1.6496
Epoch 41/50
5/5 [==============================] - 0s 6ms/step - loss: 1.5335
Epoch 42/50
5/5 [==============================] - 0s 5ms/step - loss: 1.5847
Epoch 43/50
5/5 [==============================] - 0s 6ms/step - loss: 1.5368
Epoch 44/50
5/5 [==============================] - 0s 4ms/step - loss: 1.5235
Epoch 45/50
5/5 [==============================] - 0s 4ms/step - loss: 1.5008
Epoch 46/50
5/5 [==============================] - 0s 4ms/step - loss: 1.4986
Epoch 47/50
5/5 [==============================] - 0s 5ms/step - loss: 1.4654
Epoch 48/50
5/5 [==============================] - 0s 4ms/step - loss: 1.4437
Epoch 49/50
5/5 [==============================] - 0s 4ms/step - loss: 1.4784
Epoch 50/50
5/5 [==============================] - 0s 4ms/step - loss: 1.4458
2/2 [==============================] - 0s 8ms/step - loss: 1.7568
```

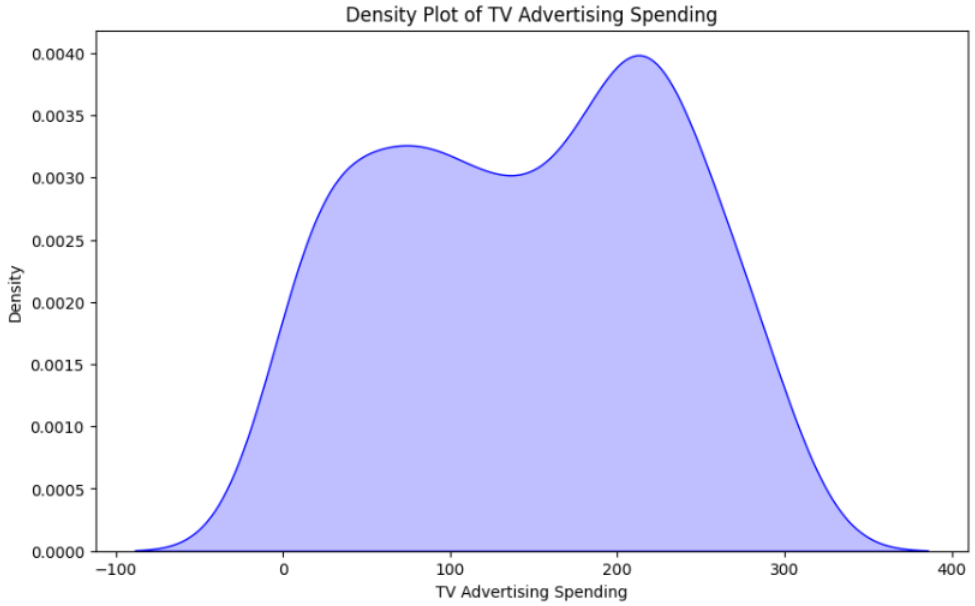
PROGRAM:

```
plt.figure(figsize=(10, 6))
```

```
sns.kdeplot(data['TV'], shade=True, color='blue')
plt.xlabel('TV Advertising Spending')
plt.ylabel('Density')
plt.title('Density Plot of TV Advertising Spending')
plt.show()
```

OUTPUT:

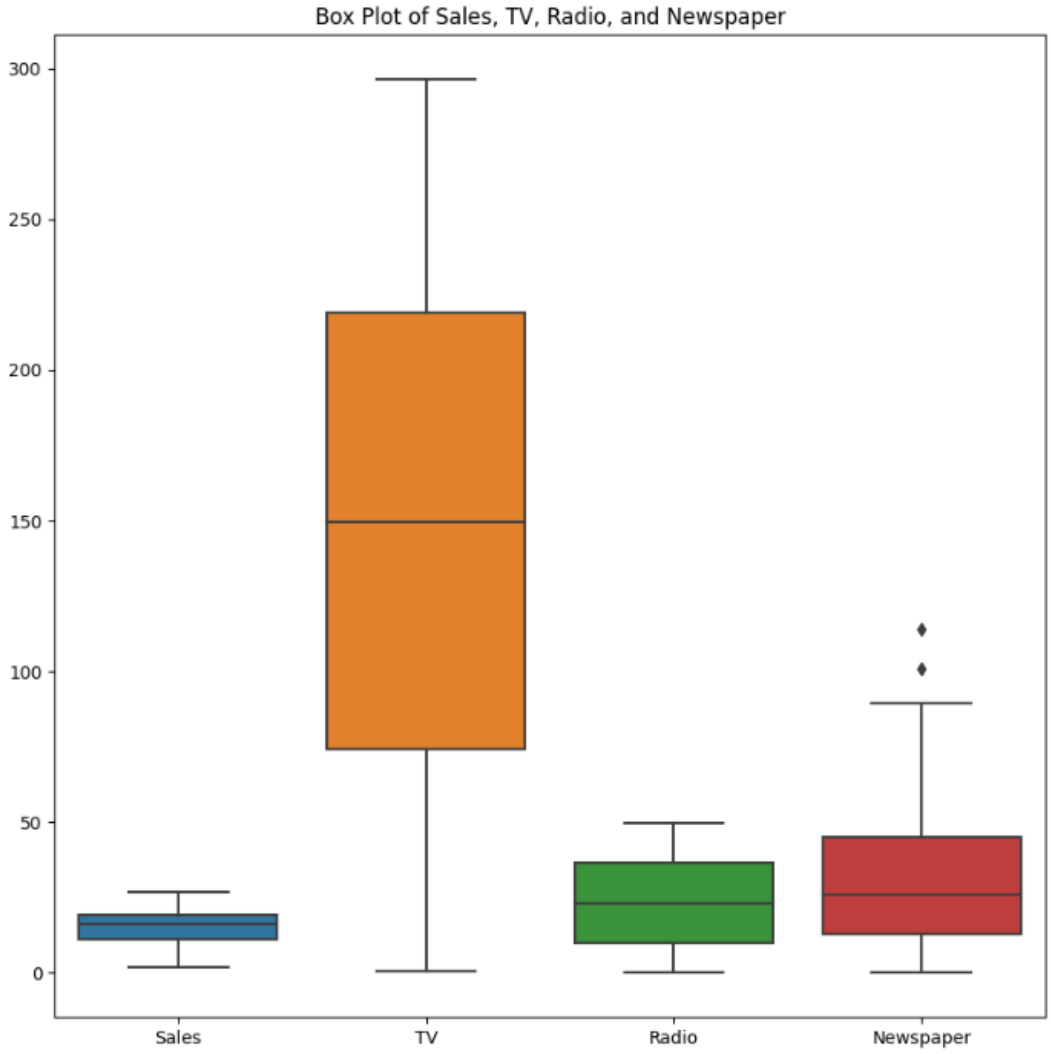
```
<ipython-input-43-17f21bb204b5>:2: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data['TV'], shade=True, color='blue')
```



PROGRAM:

```
plt.figure(figsize=(10, 10))
sns.boxplot(data=data[['Sales', 'TV', 'Radio', 'Newspaper']])
plt.title('Box Plot of Sales, TV, Radio, and Newspaper')
plt.show()
correlation_matrix = data.corr()
```

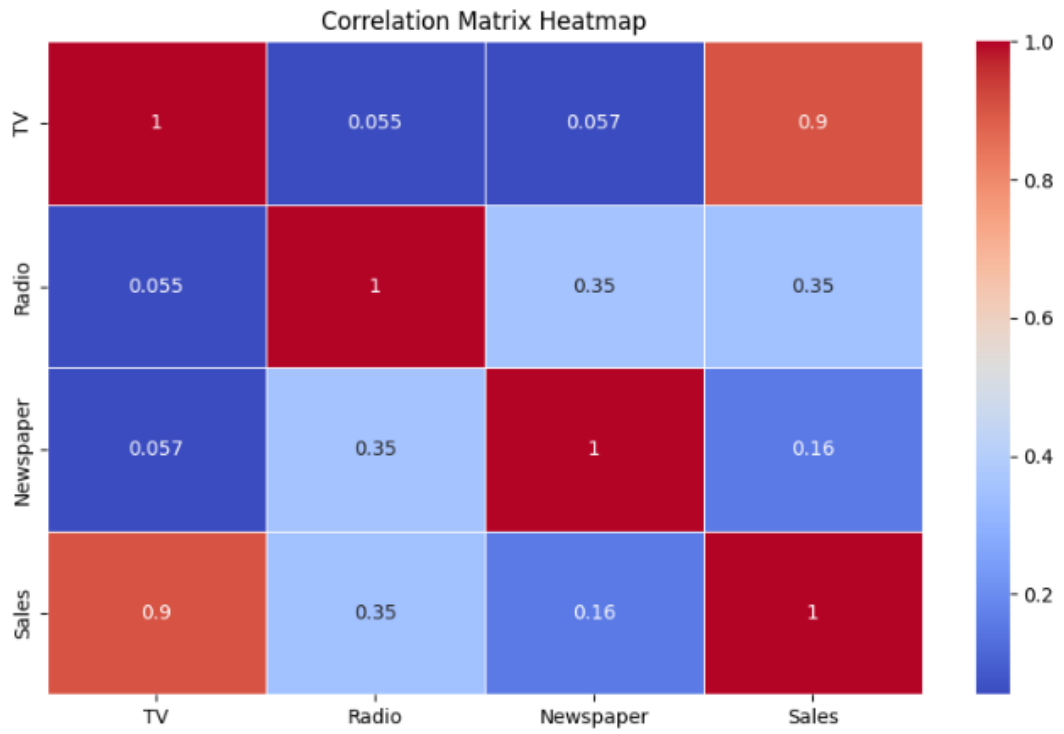
OUTPUT:



PROGRAM:

```
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Correlation Matrix Heatmap")
plt.show()
```

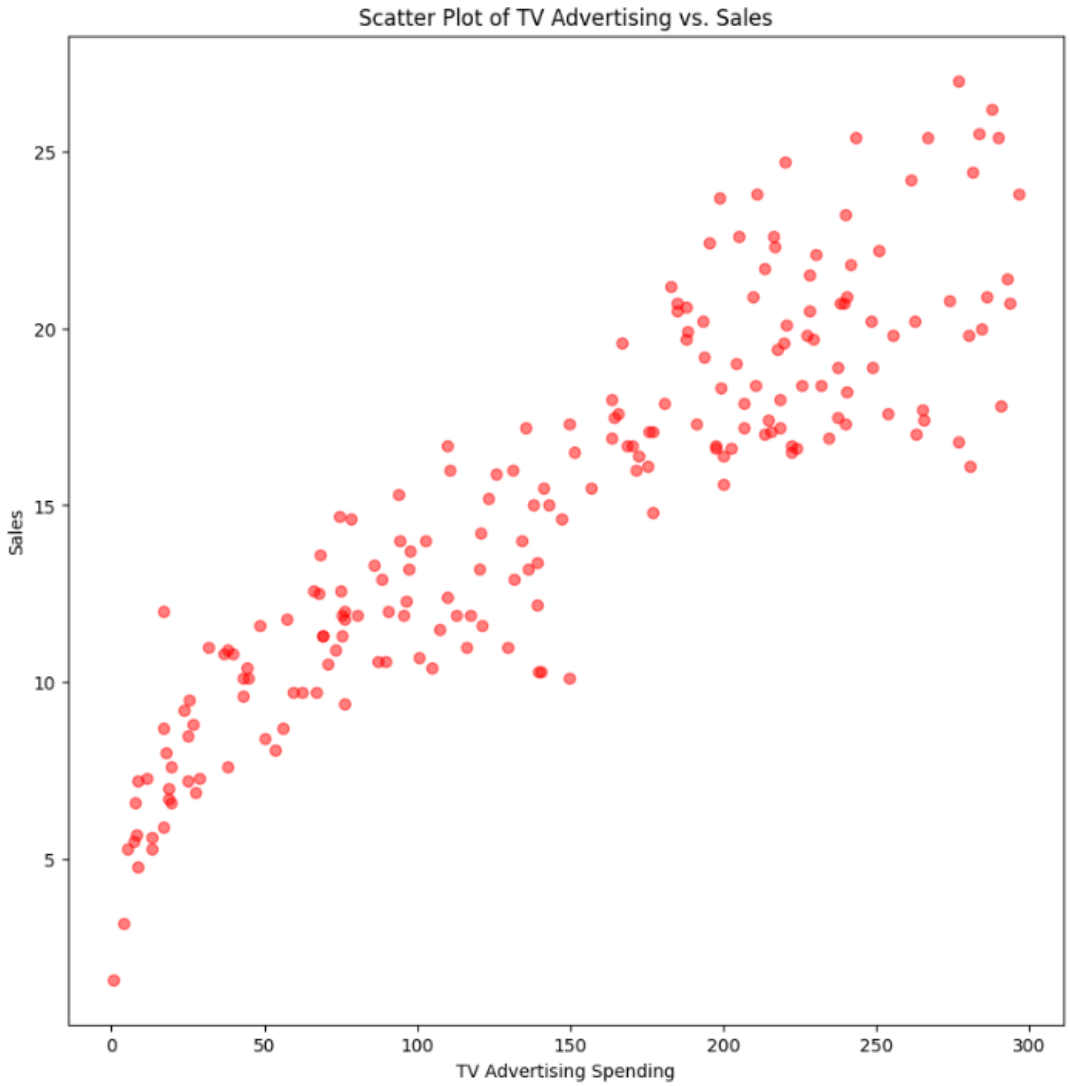
OUTPUT:



PROGRAM:

```
plt.figure(figsize=(10, 10))
plt.scatter(data['TV'], data['Sales'], c='red', alpha=0.5)
plt.xlabel('TV Advertising Spending')
plt.ylabel('Sales')
plt.title('Scatter Plot of TV Advertising vs. Sales')
plt.show()
```

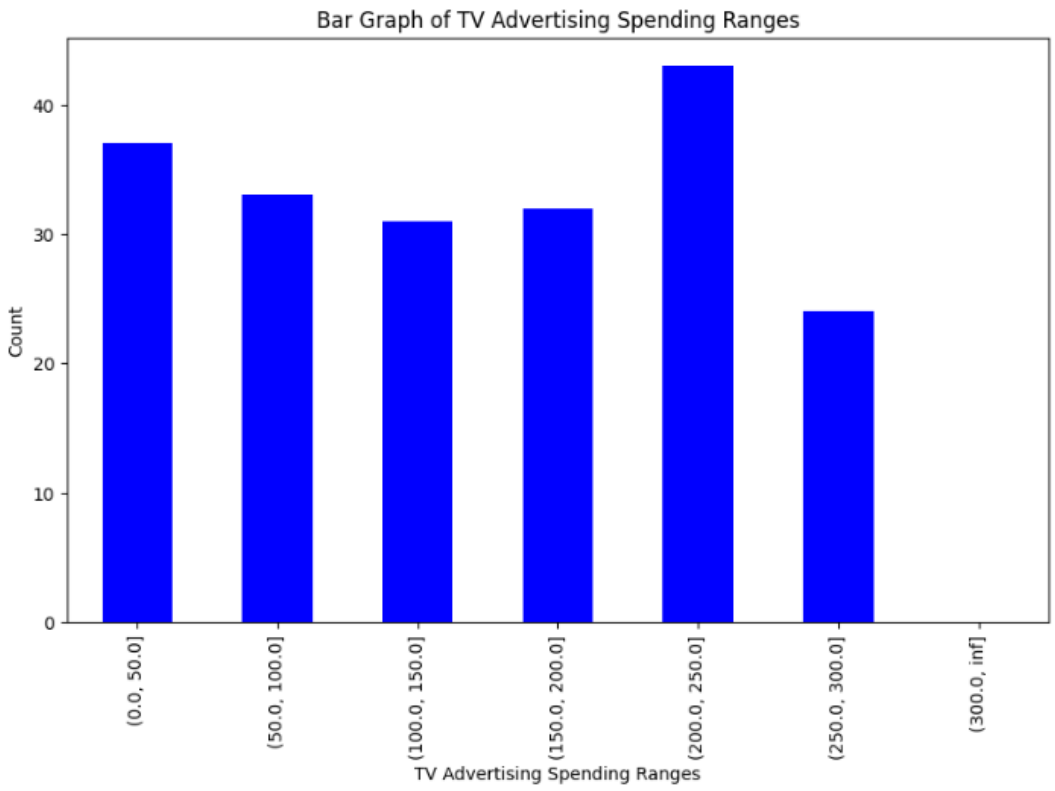
OUTPUT:



PROGRAM:

```
plt.figure(figsize=(10, 6))
bin_counts.plot(kind='bar', color='blue')
plt.xlabel('TV Advertising Spending Ranges')
plt.ylabel('Count')
plt.title('Bar Graph of TV Advertising Spending Ranges')
plt.show()
```

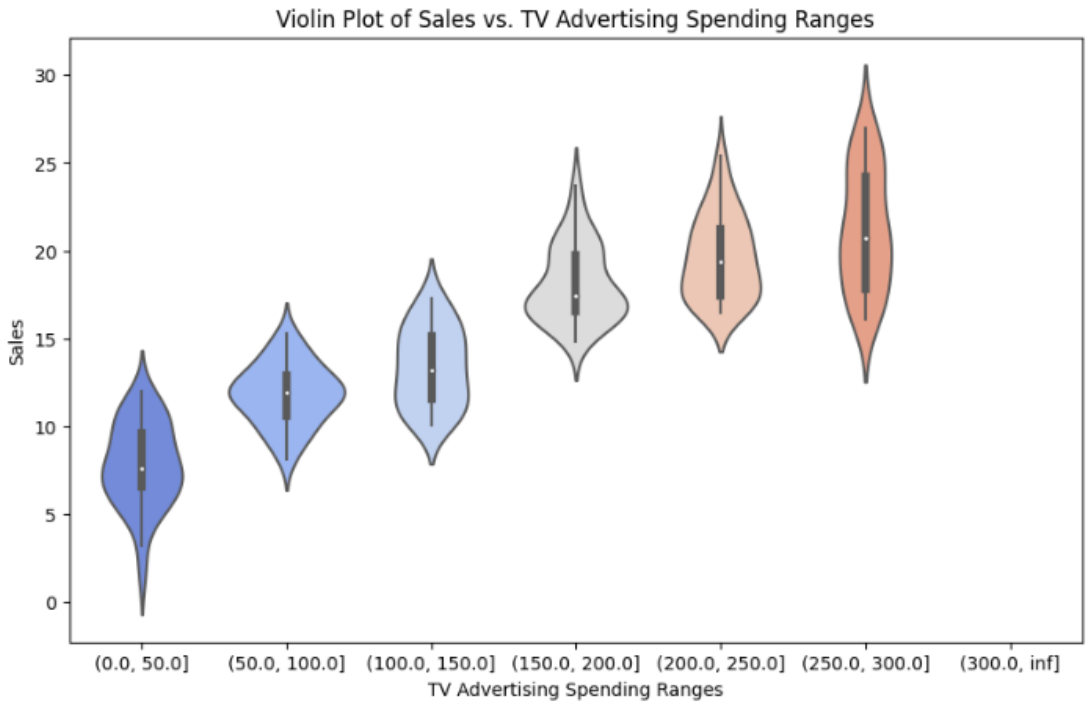
OUTPUT:



PROGRAM:

```
bin_edges = [0, 50, 100, 150, 200, 250, 300, np.inf]

data['TV_bins'] = pd.cut(data['TV'], bins=bin_edges)

bin_counts = data['TV_bins'].value_counts().sort_index()

plt.figure(figsize=(10, 6))
sns.violinplot(x='TV_bins', y='Sales', data=data, palette='coolwarm')
plt.title('Violin Plot of Sales vs. TV Advertising Spending Ranges')
plt.xlabel('TV Advertising Spending Ranges')
plt.ylabel('Sales')
plt.show()
```

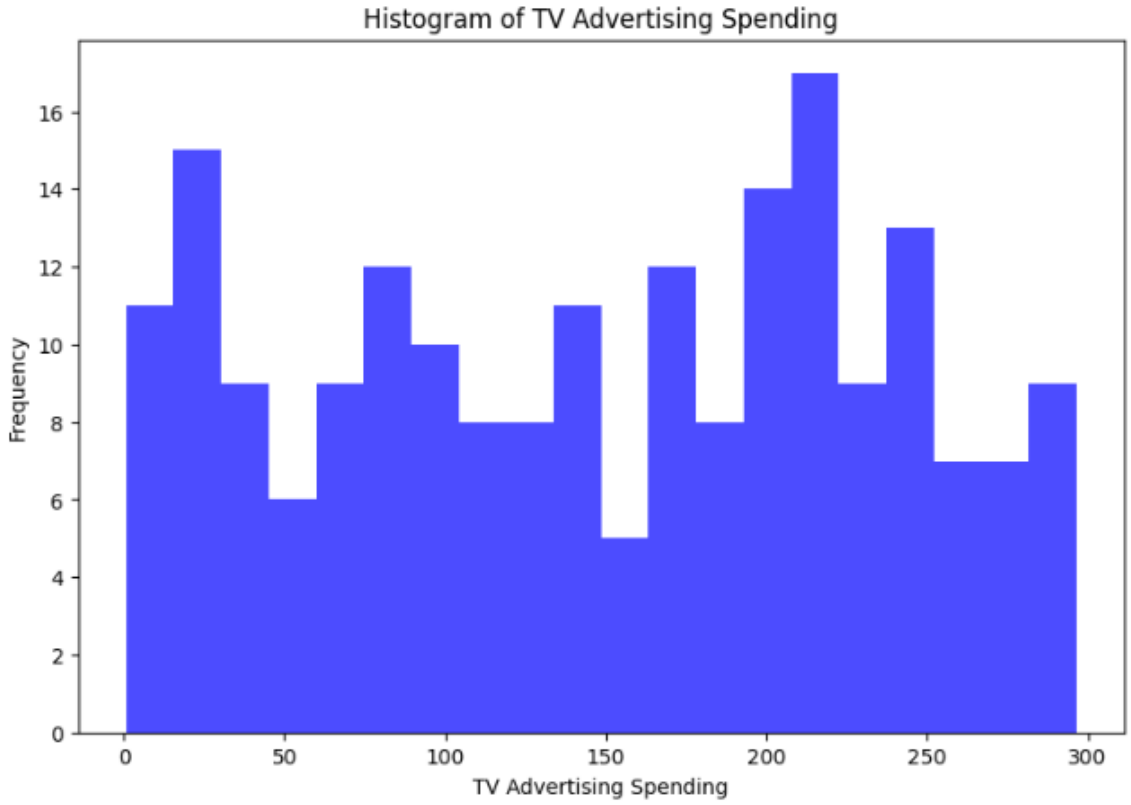
OUTPUT:



PROGRAM:

```
plt.figure(figsize=(9,6))
plt.hist(data['TV'], bins=20, color='blue', alpha=0.7)
plt.xlabel('TV Advertising Spending')
plt.ylabel('Frequency')
plt.title('Histogram of TV Advertising Spending')
plt.show()
```

OUTPUT:

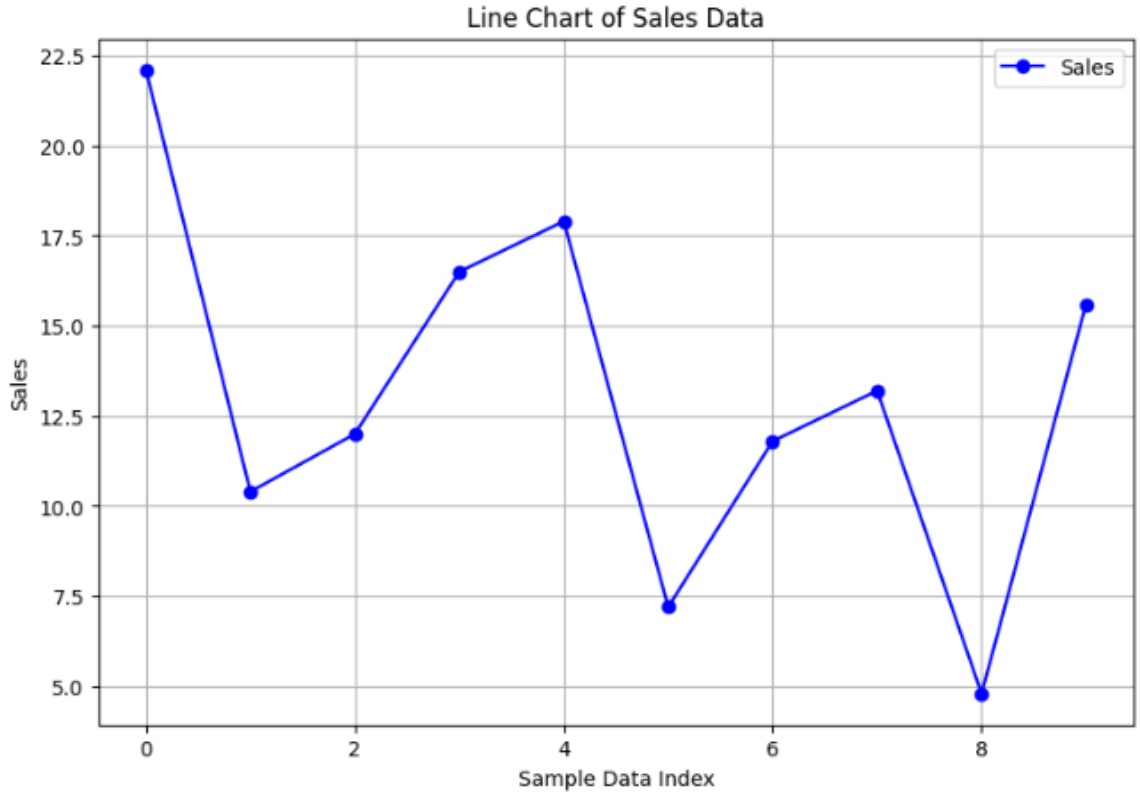


PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.DataFrame({
    'TV': [230.1, 44.5, 17.2, 151.5, 180.8, 8.7, 57.5, 120.2, 8.6, 199.8],
    'Radio': [37.8, 39.3, 45.9, 41.3, 10.8, 48.9, 32.8, 19.6, 2.1, 2.6],
    'Newspaper': [69.2, 45.1, 69.3, 58.5, 58.4, 75, 23.5, 11.6, 1, 21.2],
    'Sales': [22.1, 10.4, 12.0, 16.5, 17.9, 7.2, 11.8, 13.2, 4.8, 15.6]
})

plt.figure(figsize=(9, 6))
plt.plot(data.index, data['Sales'], marker='o', linestyle='-', color='b', label='Sales')
plt.xlabel('Sample Data Index')
```

```
plt.ylabel('Sales')
plt.title('Line Chart of Sales Data')
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT:



PROGRAM:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("\n")
print(f"Mean Absolute Error (MAE): {mae}")
print("\n")
```

```
print(f"Mean Squared Error (MSE): {mse}")
print("\n")
print(f"R-squared (R2): {r2}")
```

OUTPUT:

```
Mean Absolute Error (MAE): 0.9879407098073554

Mean Squared Error (MSE): 1.9381862178216607

R-squared (R2): 0.9372777597618964
```

```
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(n_estimators=100),
    'Support Vector Machine': SVR(),
    'Neural Network': MLPRegressor(hidden_layer_sizes=(100, 50), max_iter=1000),
    'XGBoost': XGBRegressor(),
    'LightGBM': LGBMRegressor()
}

results = []

for model_name, model in models.items():
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
```

```
    results.append((model_name, mse, mae, r2))

for model_name, mse, mae, r2 in results:
    print(f"Model: {model_name}")
    print(f"Mean Squared Error: {mse:.2f}")
    print(f"Mean Absolute Error: {mae:.2f}")
    print(f"R-squared (R2) Score: {r2:.2f}")
    print()
```

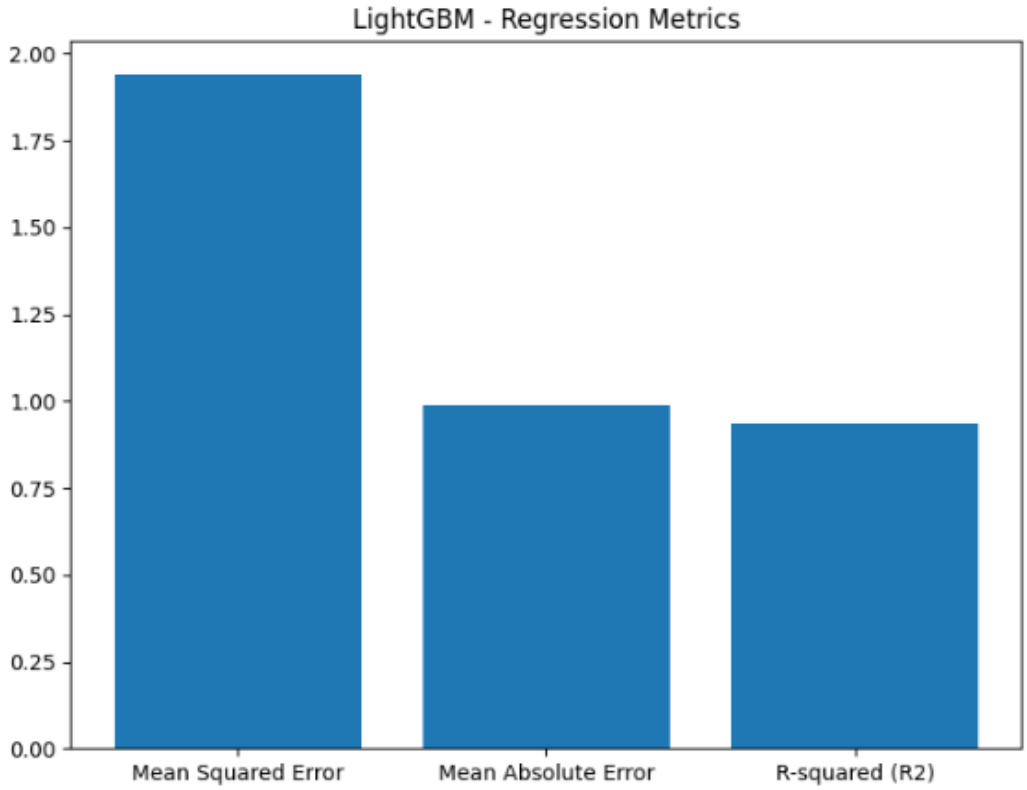
OUTPUT:

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Model: Linear Regression
Mean Squared Error: 2.91
Mean Absolute Error: 1.27
R-squared (R2) Score: 0.91

Model: Random Forest
Mean Squared Error: 1.29
Mean Absolute Error: 0.88
R-squared (R2) Score: 0.96

Model: Support Vector Machine
Mean Squared Error: 3.50
Mean Absolute Error: 1.48
R-squared (R2) Score: 0.89

Model: Neural Network
Mean Squared Error: 39.12
Mean Absolute Error: 4.85
R-squared (R2) Score: -0.27

Model: XGBoost
Mean Squared Error: 1.45
Mean Absolute Error: 0.89
R-squared (R2) Score: 0.95

Model: LightGBM
Mean Squared Error: 1.94
Mean Absolute Error: 0.99
R-squared (R2) Score: 0.94
```

PROGRAM:

```
metric_names = ['Mean Squared Error', 'Mean Absolute Error', 'R-squared (R2)']
metric_values = [mse, mae, r2]
plt.figure(figsize=(8, 6))
plt.bar(metric_names, metric_values)
plt.title(f"{model_name} - Regression Metrics")
plt.show()
```

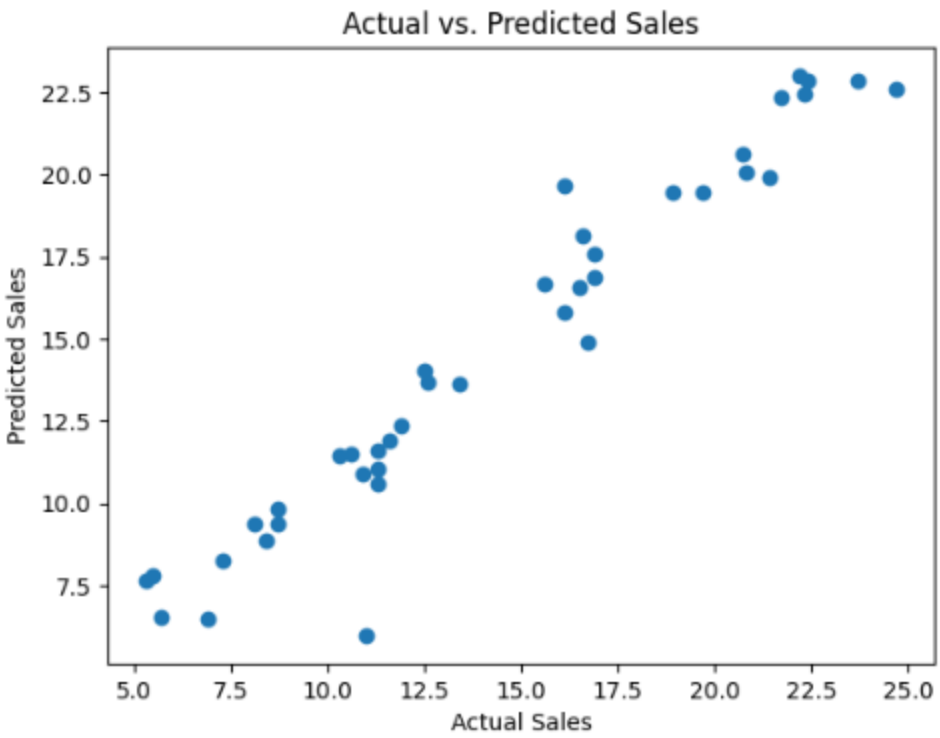
OUTPUT:



PROGRAM:

```
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Sales")
plt.ylabel("Predicted Sales")
plt.title("Actual vs. Predicted Sales")
plt.show()
```

OUTPUT:



CONCLUSION:

The provided code is a comprehensive data analysis and forecasting pipeline for a sales dataset. It begins by loading and preprocessing the data, conducting exploratory data analysis, and then performing time series forecasting using ARIMA, ETS, and SARIMA models. Additionally, a deep learning approach using LSTM is applied for time series forecasting. The code includes various data visualizations, such as density plots, box plots, heatmaps, and scatter plots, offering insights into the data's distribution and relationships. The analysis culminates with the evaluation of regression models and the presentation of their performance metrics. The code's purpose is to extract valuable insights from the dataset and provide a foundation for data-driven decision-making, although specific conclusions and insights would depend on the dataset and analysis goals.