Phase 5 : Project Documentation & Submission – Final Submission

Name: P.Kaviyalaxmi
Register Number : 312621243017
College Name : Thangavelu Engineering College
Project 3 : Future Sales Prediction

Project 3: Future Sales Prediction

Objective:

The objective is to create a tool that enables the company to optimize inventory management and make informed business decisions based on datadriven sales predictions In this part we understand the problem statement and we created a document on what have we understood and we proceeded ahead with solving the problem. The problem is to develop a predictive model that uses historical sales data to forecast future sales for a retail company.

Problem Definition:

The problem is to develop a predictive model that uses historical sales data to forecast future sales for a retail company. This project involves data preprocessing, feature engineering, model selection, training, and evaluation. This model will predict sales on a certain day after being provided with a certain set of inputs.

Design Thinking:

Data Source:

- Utilize a dataset containing historical sales data. Here a csv file is converted to a DataFrame and the pandas object is used.
- The head() method returns a specified number of rows, string from the top. The head() method returns the first 5 rows if a number is not specified.
- The describe() method returns description of the data in the DataFrame. We calculate and print the summary statistics of the dataset using df.describe() function . The describe() method returns description of the data in the DataFrame. If the DataFrame contains numerical data, the description contains

these information for each column such as count , mean , std , min , 25% , 50% , 75% , max .

Data Cleaning and Preprocessing:

- In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull(). Both function help in checking whether a value is NaN or not. These function can also be used in Pandas Series in order to find null values in a series.
- Now we set some range for each variable and performs the range checks. It is performed between each columns of the test dataset.
- Now we are going to check wheather the given datas entered are correct or not by checking the non negative values in the data.
- The drop_duplicates() method removes duplicate rows. Use the subset parameter if only some specified columns should be considered when looking for duplicates.
- Outliers are data points that significantly differ from the rest of the observations in a dataset. They can be unusually high or low values compared to the majority of the data. In statistical terms, outliers are observations that fall outside of the typical range of values.Outliers can arise due to various reasons, such as errors in data collection, measurement variability, or the presence of rare events. They have the potential to skew statistical analyses and machine learning models, leading to misleading or inaccurate results.Detecting and handling outliers is an important step in data preprocessing and analysis to ensure that the insights drawn from the data are robust and representative of the underlying patterns.
- Now here we are extracting the dependent variable and independent variables .The ‘sales’ is a dependent variable and the other are the independent variables .
- Now here the binning method is to smooth or handle noisy data. First, the data is sorted then, and then the sorted values are separated and stored in the form of bins.

- A correlation matrix is a table containing correlation coefficients for many variables. Each cell in the table represents the correlation between two variables. The value might range between -1 and 1.
- StandardScaler removes the mean and scales each feature/variable to unit variance. This operation is performed feature-wise in an independent way. StandardScaler can be influenced by outliers (if they exist in the dataset) since it involves the estimation of the empirical mean and standard deviation of each feature.
- K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

Adding Features:

Here we add some features to the dataset for some accuracy

- Here we have created a new column Total_Spent in our DataFrame df by summing up the expenses from 'TV', 'Radio', and 'Newspaper'. This can be a useful feature for our sales prediction model, as it captures the total advertising expenditure.
- Here we have added a new column called 'Previous_Sales' to our DataFrame by shifting the 'Sales' column by one position. This creates a lagged version of the sales data.
- Now we have created a new column called TV_Radio_Interact in our DataFrame df by multiplying the 'TV' and 'Radio' columns. This is an example of feature engineering, which can potentially improve the performance of our predictive model.

- Here we have added a new feature named 'TV_log' which represents the logarithm (base) of the 'TV' column. This transformation can be useful if the relationship between 'TV' and the target variable is non-linear.T he code we provided will calculate the natural logarithm of the 'TV' column and assign it to the new 'TV_log' column in our DataFrame ‘df’ .Keep in mind that this transformation may help linearize the relationship between 'TV' and the target variable, which can potentially improve the performance of our linear regression model. However, it's always a good idea to evaluate the impact of this transformation on our model's performance using appropriate evaluation metrics.

Model Selection and Training:

Here the Linear regression is selected as the model and trained . This model is selected here for the prediction and accuracy .

Linear Regression:

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables . Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

Import the necessary libraries: Certainly! To run the code, we'll need to import the necessary libraries.

Load and preprocess our dataset: We have successfully created the features (X) and target variable (y) and then split the data into training and testing sets using the train_test_split function. This is a common and essential step in machine learning workflows.Here's a quick summary of what we've done:

- We selected the features 'TV', 'Radio', and 'Newspaper' from our DataFrame df and assigned them to X. This will be used as input to train our model.
- We selected the 'Sales' column from our DataFrame df and assigned it to y. This will be our target variable that we want to predict.
- We used the train_test_split function to split our data into training and testing sets.
- The training set (X_train and y_train) will be used to train the model, and the testing set (X_test and y_test) will be used to evaluate the model's performance.
- The test_size=0.2 argument indicates that 20% of the data will be used for testing, while 80% will be used for training.
- The random_state=42 argument ensures that the data is split in a reproducible manner (the same split will occur every time we run the code), which can be important for debugging and comparing different models.

Initialize and train the Linear Regression model: We have correctly initialized and trained a Linear Regression model by the appropriate code and algorithm .

Predict using the model: We have successfully used your trained model to make predictions on the test set (X_test). The predicted values are stored in the variable y_pred.This will display the array of predicted values for the test set. Each element in the array corresponds to the predicted value for a specific instance in your test set.

Now we create some scatter plot for this model . the below code will create a scatter plot where the x-axis represents the actual sales values (y_test) and the y-axis represents the predicted sales values (y_pred). If the model predictions are accurate, the points should fall close to a diagonal line.The visualization is effective for understanding the relationship between actual and predicted values in a regression model.

Evaluate the model:

- Mean Squared Error (MSE) measures the amount of error in a statistical model. Evaluate the mean squared difference between observed and predicted values. If the model has no errors, the MSE is zero. Its value increases as the model error increases.
- absolute error refers to the magnitude of difference between the prediction of an observation and the true value of that observation.The MAE measures the average magnitude of the errors in a set of forecasts, without considering their direction. It measures accuracy for continuous variables.
- The root mean square error (RMSE) measures the average difference between a statistical model's predicted values and the actual values. Mathematically, it is the standard deviation of the residuals. Residuals represent the distance between the regression line and the data points.
- R2 is a measure of the goodness of fit of a model. In regression, the R2 coefficient of determination is a statistical measure of how well the regression predictions approximate the real data points. An R2 of 1 indicates that the regression predictions perfectly fit the data.

Finally Here printing the coefficients of the data model

Code and Explanation :

```
import pandas as pd

df=pd.read_csv(r'Sales.csv')

print(df)
```

```
        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
..     ...    ...        ...    ...
195   38.2    3.7       13.8    7.6
196   94.2    4.9        8.1   14.0
197  177.0    9.3        6.4   14.8
198  283.6   42.0       66.2   25.5
199  232.1    8.6        8.7   18.4

[200 rows x 4 columns]
```

```
# Print the first few rows of the DataFrame

print(df.head())
```

```
      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
```

```
# Check basic statistics

summary_stats = df.describe()

print("\nSummary Statistics:")

print(summary_stats)
```

```
Summary Statistics:
               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
```

```
#checking wheather the data is available or not recorded.
missing_values = df.isnull().sum()
# Print the missing values (if any)
print("Missing Values:")
print(missing_values)
# You can also print a message based on the result
if missing_values.sum() == 0:
   print("\nNo missing values found. Data is available.")
else:
   print("\nMissing values found. Data may be incomplete or recorded
incorrectly.")
```

```
Missing Values:
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64

No missing values found. Data is available.
```

```
#Define reasonable ranges for each variable
reasonable_ranges = {
```

```
    'TV': (0, 1000),    # Example: TV budget should be between 0 and 1000
    'Radio': (0, 100),   # Example: Radio budget should be between 0 and 100
    'Newspaper': (0, 200),  # Example: Newspaper budget should be between 0 and 200
    'Sales': (0, 100)    # Example: Sales should be between 0 and 100
}
# Perform range checks
for column, (min_val, max_val) in reasonable_ranges.items():
    if ((df[column] < min_val) | (df[column] > max_val)).any():
        print(f"Warning: Values in column '{column}' are not within the reasonable range of ({min_val}, {max_val}). Please verify the data.")
    else:
        print(f"The values in the column '{column}' are within the reasonable range of ({min_val},{max_val})")
```

```
The values in the column 'TV' are within the reasonable range of (0,1000)
The values in the column 'Radio' are within the reasonable range of (0,100)
The values in the column 'Newspaper' are within the reasonable range of (0,200)
The values in the column 'Sales' are within the reasonable range of (0,100)
```

```
#To check whether the data entered is correct or not.
# Checking if sales values are non-negative
if (df['Sales'] < 0).any():
    print("Warning: There are negative sales values. Please verify the data.")
else:
    print("The Data contains non negative sales values")
```

```
The Data contains non negative sales values
```

```
# Check for negative or unrealistic values
if (df < 0).any().any():
    print(f"Warning: There are negative values in the dataset. Please verify the data.")
else:
    print("No negative values found.")
```

```
No negative values found.
```

```
# Checking if advertising budgets are non-negative
if (df[['TV', 'Radio', 'Newspaper']] < 0).any().any():
    print("Warning: There are negative advertising budget values. Please verify the data.")
else:
    print("The Data contains non negative advertising budget values")
```

```
The Data contains non negative advertising budget values
```

```
#to remove duplicate values
df_dup = df.drop_duplicates()
# Display the DataFrame without duplicates
print("DataFrame without Duplicates:")
print(df_dup)
```

```
DataFrame without Duplicates:
        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
..     ...    ...        ...    ...
195   38.2    3.7       13.8    7.6
196   94.2    4.9        8.1   14.0
197  177.0    9.3        6.4   14.8
198  283.6   42.0       66.2   25.5
199  232.1    8.6        8.7   18.4

[200 rows x 4 columns]
```

```
#Check for outliers and decide whether to remove them or not.

import numpy as np

# Define a function to detect outliers using IQR method

def detect_outliers(data):

    Q1 = data.quantile(0.25)

    Q3 = data.quantile(0.75)

    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR

    upper_bound = Q3 + 1.5 * IQR

    return (data < lower_bound) | (data > upper_bound)

# Select the columns you want to check for outliers

columns_to_check = ['TV', 'Radio', 'Newspaper', 'Sales']

# Check for outliers in each column

outliers = df[columns_to_check].apply(detect_outliers)

print(outliers)
```

```
         TV  Radio  Newspaper  Sales
0    False  False      False  False
1    False  False      False  False
2    False  False      False  False
3    False  False      False  False
4    False  False      False  False
..     ...    ...        ...    ...
195  False  False      False  False
196  False  False      False  False
197  False  False      False  False
198  False  False      False  False
199  False  False      False  False

[200 rows x 4 columns]
```

```
# Count the number of outliers in each column
num_outliers = outliers.sum()
# Display the number of outliers for each column
print("Number of outliers:")
print(num_outliers)
```

```
Number of outliers:
TV           0
Radio        0
Newspaper    2
Sales        0
dtype: int64
```

```
# Display the rows containing outliers
outliers_rows = df[outliers.any(axis=1)]
print("\nRows containing outliers:")
print(outliers_rows)
```

```
Rows containing outliers:
        TV  Radio  Newspaper  Sales
16    67.8   36.6      114.0   12.5
101  296.4   36.3      100.9   23.8
```

To remove outliers, you can use something like this:

```
df_cleaned = df[~outliers.any(axis=1)]

print("\nOutliers removed:")

print(df_cleaned)
```

```
Outliers removed:
        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
..     ...    ...        ...    ...
195   38.2    3.7       13.8    7.6
196   94.2    4.9        8.1   14.0
197  177.0    9.3        6.4   14.8
198  283.6   42.0       66.2   25.5
199  232.1    8.6        8.7   18.4

[198 rows x 4 columns]
```

```
# Extracting the dependent variable 'Sales'

y = df['Sales']

# Printing the first few rows of 'Sales' to verify

print(y.head())
```

```
0    22.1
1    10.4
2    12.0
3    16.5
4    17.9
Name: Sales, dtype: float64
```

Extracting the independent variables into a DataFrame

independent_variables = df[['TV', 'Radio', 'Newspaper']]

Printing the extracted independent variables

print("Independent Variables:")

print(independent_variables)

```
Independent Variables:
        TV  Radio  Newspaper
0    230.1   37.8       69.2
1     44.5   39.3       45.1
2     17.2   45.9       69.3
3    151.5   41.3       58.5
4    180.8   10.8       58.4
..     ...    ...        ...
195   38.2    3.7       13.8
196   94.2    4.9        8.1
197  177.0    9.3        6.4
198  283.6   42.0       66.2
199  232.1    8.6        8.7

[200 rows x 3 columns]
```

#Bining the data

Adjust the bin_edges and bin_labels as per your specific requirements

bin_edges = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

bin_labels = ['0-10', '10-20', '20-30', '30-40', '40-50', '50-60', '60-70', '70-80', '80-90', '90-100']

#Apply binning to a specific column (e.g., 'Sales')

```
df['Sales_bin'] = pd.cut(df['Sales'], bins=bin_edges, labels=bin_labels,
include_lowest=True)

# You can choose a different column

#or adjust bin_edges and bin_labels based on your specific requirements.

# Display the DataFrame with the new binning column

print("DataFrame with Binning:")

print(df)
```

```
DataFrame with Binning:
        TV  Radio  Newspaper  Sales Sales_bin
0    230.1   37.8       69.2   22.1     20-30
1     44.5   39.3       45.1   10.4     10-20
2     17.2   45.9       69.3   12.0     10-20
3    151.5   41.3       58.5   16.5     10-20
4    180.8   10.8       58.4   17.9     10-20
..     ...    ...        ...    ...       ...
195   38.2    3.7       13.8    7.6      0-10
196   94.2    4.9        8.1   14.0     10-20
197  177.0    9.3        6.4   14.8     10-20
198  283.6   42.0       66.2   25.5     20-30
199  232.1    8.6        8.7   18.4     10-20

[200 rows x 5 columns]
```

```
# Correlation matrix

correlation_matrix = df.corr()

print(correlation_matrix)
```

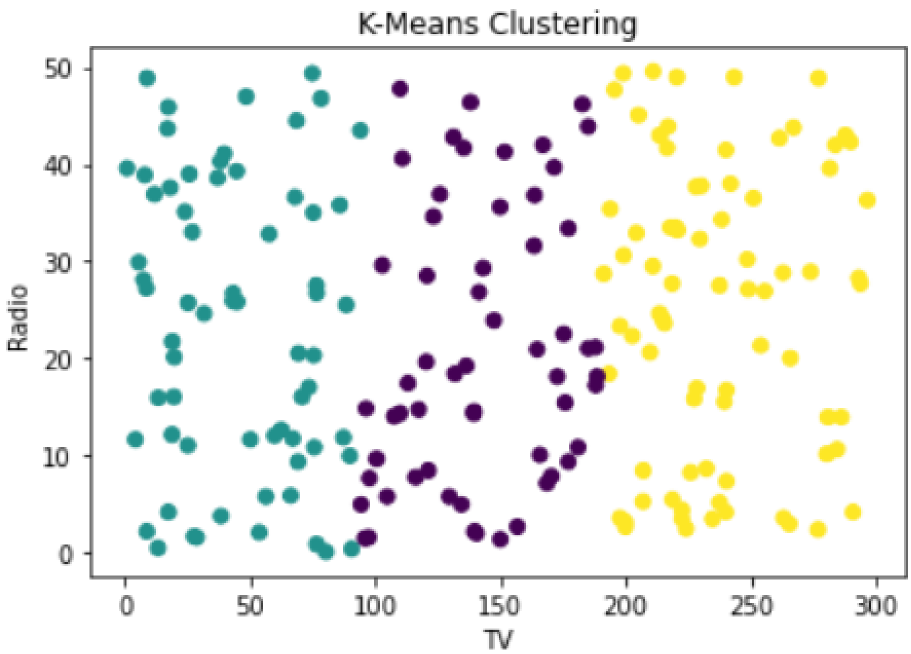
```
                 TV     Radio  Newspaper     Sales
TV         1.000000  0.054809   0.056648  0.901208
Radio      0.054809  1.000000   0.354104  0.349631
Newspaper  0.056648  0.354104   1.000000  0.157960
Sales      0.901208  0.349631   0.157960  1.000000
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

numerical_features = ['TV', 'Radio', 'Newspaper']

df_cleaned[numerical_features] =
scaler.fit_transform(df_cleaned[numerical_features])

print("Cleaned and Preprocessed DataFrame:")

print(df_cleaned)
```

```
Cleaned and Preprocessed DataFrame:
           TV     Radio  Newspaper  Sales
0    0.978697  0.989521   1.932998   22.1
1   -1.199012  1.090705   0.751313   10.4
2   -1.519332  1.535913   1.937901   12.0
3    0.056456  1.225616   1.408349   16.5
4    0.400243 -0.831784   1.403446   17.9
..        ...       ...        ...    ...
195 -1.272932 -1.310720  -0.783407    7.6
196 -0.615864 -1.229773  -1.062892   14.0
197  0.355657 -0.932968  -1.146248   14.8
198  1.606431  1.272836   1.785900   25.5
199  1.002164 -0.980187  -1.033473   18.4

[198 rows x 4 columns]
```

```
from sklearn.cluster import KMeans

import matplotlib.pyplot as plt


#Select the features for clustering (e.g., 'TV', 'Radio', 'Newspaper')

features = df[['TV', 'Radio', 'Newspaper']]


#Choose the number of clusters (k)

k = 3  # Adjust based on your specific analysis
```

```
#Perform K-Means clustering

kmeans = KMeans(n_clusters=k, random_state=0)

df['Cluster'] = kmeans.fit_predict(features)


# Visualize the clusters (for 2D visualization)

plt.scatter(df['TV'], df['Radio'], c=df['Cluster'], cmap='viridis')

plt.xlabel('TV')

plt.ylabel('Radio')

plt.title('K-Means Clustering')

plt.show()


# The above visualization assumes 'TV' and 'Radio' as features. You can adjust
based on your specific features.


#Optional - Perform further analysis on the clusters

# You can analyze the clusters further to understand their characteristics.


# Display the DataFrame with cluster assignments

print("DataFrame with Clusters:")

print(df)
```




```
DataFrame with Clusters:
        TV  Radio  Newspaper  Sales Sales_bin  Cluster
0    230.1   37.8       69.2   22.1     20-30        2
1     44.5   39.3       45.1   10.4     10-20        1
2     17.2   45.9       69.3   12.0     10-20        1
3    151.5   41.3       58.5   16.5     10-20        0
4    180.8   10.8       58.4   17.9     10-20        0
..     ...    ...        ...    ...       ...      ...
195   38.2    3.7       13.8    7.6      0-10        1
196   94.2    4.9        8.1   14.0     10-20        0
197  177.0    9.3        6.4   14.8     10-20        0
198  283.6   42.0       66.2   25.5     20-30        2
199  232.1    8.6        8.7   18.4     10-20        2

[200 rows x 6 columns]
```

```
df['Total_Spent'] = df['TV'] + df['Radio'] + df['Newspaper']
print(df)
```

```
          TV  Radio  Newspaper  Sales Sales_bin  Cluster  Total_Spent
0      230.1   37.8       69.2   22.1     20-30        2        337.1
1       44.5   39.3       45.1   10.4     10-20        1        128.9
2       17.2   45.9       69.3   12.0     10-20        1        132.4
3      151.5   41.3       58.5   16.5     10-20        0        251.3
4      180.8   10.8       58.4   17.9     10-20        0        250.0
..       ...    ...        ...    ...       ...      ...          ...
195     38.2    3.7       13.8    7.6      0-10        1         55.7
196     94.2    4.9        8.1   14.0     10-20        0        107.2
197    177.0    9.3        6.4   14.8     10-20        0        192.7
198    283.6   42.0       66.2   25.5     20-30        2        391.8
199    232.1    8.6        8.7   18.4     10-20        2        249.4

[200 rows x 7 columns]
```

```
df['Previous_Sales'] = df['Sales'].shift(1)  # Lagged sales
print(df)
```

```
          TV  Radio  Newspaper  Sales Sales_bin  Cluster  Total_Spent  \
0      230.1   37.8       69.2   22.1     20-30        2        337.1
1       44.5   39.3       45.1   10.4     10-20        1        128.9
2       17.2   45.9       69.3   12.0     10-20        1        132.4
3      151.5   41.3       58.5   16.5     10-20        0        251.3
4      180.8   10.8       58.4   17.9     10-20        0        250.0
..       ...    ...        ...    ...       ...      ...          ...
195     38.2    3.7       13.8    7.6      0-10        1         55.7
196     94.2    4.9        8.1   14.0     10-20        0        107.2
197    177.0    9.3        6.4   14.8     10-20        0        192.7
198    283.6   42.0       66.2   25.5     20-30        2        391.8
199    232.1    8.6        8.7   18.4     10-20        2        249.4

     Previous_Sales
0               NaN
1              22.1
2              10.4
3              12.0
4              16.5
..              ...
195            17.3
196             7.6
197            14.0
198            14.8
199            25.5

[200 rows x 8 columns]
```

```
df['TV_Radio_Interact'] = df['TV'] * df['Radio']
print(df)
```

```
        TV  Radio  Newspaper  Sales Sales_bin  Cluster  Total_Spent  \
0    230.1   37.8       69.2   22.1     20-30        2        337.1
1     44.5   39.3       45.1   10.4     10-20        1        128.9
2     17.2   45.9       69.3   12.0     10-20        1        132.4
3    151.5   41.3       58.5   16.5     10-20        0        251.3
4    180.8   10.8       58.4   17.9     10-20        0        250.0
..     ...    ...        ...    ...       ...      ...          ...
195   38.2    3.7       13.8    7.6      0-10        1         55.7
196   94.2    4.9        8.1   14.0     10-20        0        107.2
197  177.0    9.3        6.4   14.8     10-20        0        192.7
198  283.6   42.0       66.2   25.5     20-30        2        391.8
199  232.1    8.6        8.7   18.4     10-20        2        249.4

     Previous_Sales  TV_Radio_Interact
0               NaN            8697.78
1              22.1            1748.85
2              10.4             789.48
3              12.0            6256.95
4              16.5            1952.64
..              ...                ...
195            17.3             141.34
196             7.6             461.58
197            14.0            1646.10
198            14.8           11911.20
199            25.5            1996.06

[200 rows x 9 columns]
```


df['TV_Radio_Interact'] = df['TV'] * df['Radio']

print(df)

```
        TV  Radio  Newspaper  Sales Sales_bin  Cluster  Total_Spent
0    230.1   37.8       69.2   22.1     20-30        2        337.1
1     44.5   39.3       45.1   10.4     10-20        1        128.9
2     17.2   45.9       69.3   12.0     10-20        1        132.4
3    151.5   41.3       58.5   16.5     10-20        0        251.3
4    180.8   10.8       58.4   17.9     10-20        0        250.0
..     ...    ...        ...    ...       ...      ...          ...
195   38.2    3.7       13.8    7.6      0-10        1         55.7
196   94.2    4.9        8.1   14.0     10-20        0        107.2
197  177.0    9.3        6.4   14.8     10-20        0        192.7
198  283.6   42.0       66.2   25.5     20-30        2        391.8
199  232.1    8.6        8.7   18.4     10-20        2        249.4

     Previous_Sales  TV_Radio_Interact
0               NaN            8697.78
1              22.1            1748.85
2              10.4             789.48
3              12.0            6256.95
4              16.5            1952.64
..              ...                ...
195            17.3             141.34
196             7.6             461.58
197            14.0            1646.10
198            14.8           11911.20
199            25.5            1996.06

[200 rows x 9 columns]
```

```
import numpy as np

df['TV_log'] = np.log(df['TV'])
print(df)
```

```
        TV  Radio  Newspaper  Sales Sales_bin  Cluster  Total_Spent  \
0    230.1   37.8       69.2   22.1     20-30        2        337.1
1     44.5   39.3       45.1   10.4     10-20        1        128.9
2     17.2   45.9       69.3   12.0     10-20        1        132.4
3    151.5   41.3       58.5   16.5     10-20        0        251.3
4    180.8   10.8       58.4   17.9     10-20        0        250.0
..     ...    ...        ...    ...       ...      ...          ...
195   38.2    3.7       13.8    7.6      0-10        1         55.7
196   94.2    4.9        8.1   14.0     10-20        0        107.2
197  177.0    9.3        6.4   14.8     10-20        0        192.7
198  283.6   42.0       66.2   25.5     20-30        2        391.8
199  232.1    8.6        8.7   18.4     10-20        2        249.4

     Previous_Sales  TV_Radio_Interact    TV_log
0               NaN            8697.78  5.438514
1              22.1            1748.85  3.795489
2              10.4             789.48  2.844909
3              12.0            6256.95  5.020586
4              16.5            1952.64  5.197391
..              ...                ...       ...
195            17.3             141.34  3.642836
196             7.6             461.58  4.545420
197            14.0            1646.10  5.176150
198            14.8           11911.20  5.647565
199            25.5            1996.06  5.447168

[200 rows x 10 columns]
```

Import the necessary libraries:

```
from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

import pandas as pd
```

Load and preprocess your dataset:

```
X = df[['TV', 'Radio', 'Newspaper']]  # Features

y = df['Sales']  # Target variable
```

```
# 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)
```

Initialize and train the Linear Regression model:

```
# Initialize the Linear Regression model

model = LinearRegression()

# Train the model on the training data

model.fit(X_train, y_train)
```

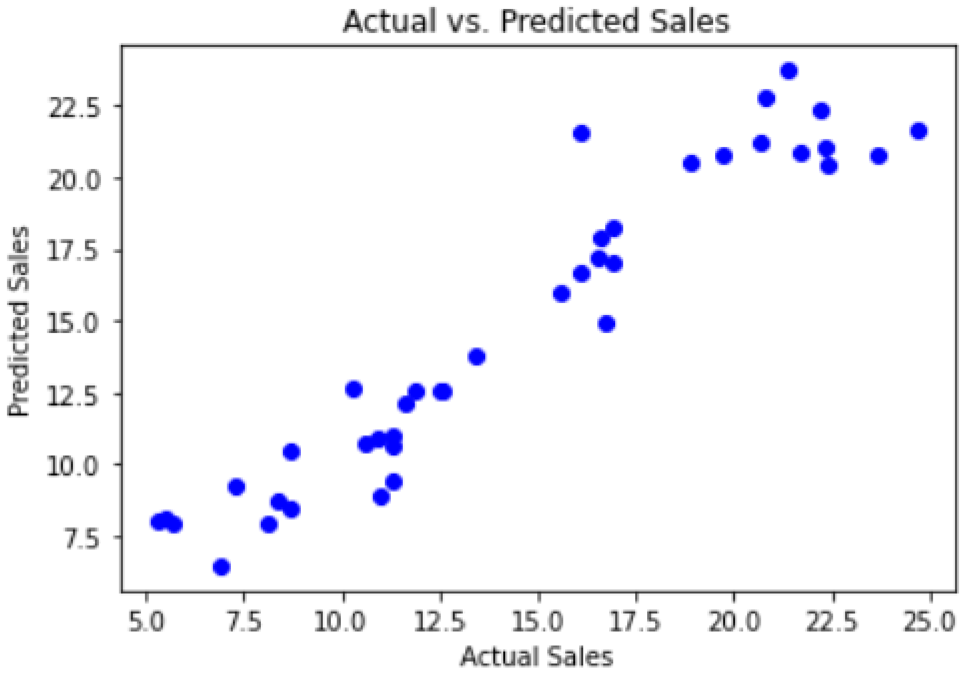
```
: LinearRegression()
```

Predict using the model:

```
# Predict on the test set

y_pred = model.predict(X_test)

print(y_pred)
```

```
[17.0347724  20.40974033 23.72398873  9.27278518 21.68271879 12.5694016
 21.08119452  8.69035045 17.23701254 16.66657475  8.92396497  8.4817344
 18.2075123   8.06750728 12.64550975 14.93162809  8.12814594 17.8987656
 11.00880637 20.47832788 20.80631846 12.59883297 10.9051829  22.3885477
  9.41796094  7.92506736 20.83908497 13.81520938 10.77080925  7.9268250
 15.95947357 10.63490851 20.80292008 10.43434164 21.5784752  21.1836448
 12.12821771 22.80953262 12.60992766  6.46441252]
```

```
import matplotlib.pyplot as plt

# Assuming 'y_test' and 'y_pred' are your actual and predicted values, respectively

plt.scatter(y_test, y_pred, color='blue')

plt.xlabel('Actual Sales')

plt.ylabel('Predicted Sales')

plt.title('Actual vs. Predicted Sales')

plt.show()
```




```
from sklearn.metrics import mean_squared_error

# Calculate the Mean Squared Error (MSE)

mse = mean_squared_error(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 2.907756910271092
```

```
from sklearn.metrics import mean_absolute_error

# Calculate MAE

mae = mean_absolute_error(y_test, y_pred)

print(f"Mean Absolute Error: {mae}")
```

```
Mean Absolute Error: 1.2748262109549344
```

```
import numpy as np

# Calculate RMSE

rmse = np.sqrt(mse)

print(f"Root Mean Squared Error: {rmse}")
```

```
Root Mean Squared Error: 1.7052146229349232
```

```
from sklearn.metrics import r2_score

# Calculate R2 score

r2 = r2_score(y_test, y_pred)

print(f"R-squared (R2) Score: {r2}")
```

```
R-squared (R2) Score: 0.9059011844150826
```

```python
# Print the coefficients
print("Coefficients:")
for feature, coef in zip(X.columns, model.coef_):
    print(f"{feature}: {coef}")
print(f"Intercept: {model.intercept_}")
```

```
Coefficients:
TV: 0.0545092708372197645
Radio: 0.10094536239295575
Newspaper: 0.004336646822034021
Intercept: 4.714126402214134
```