Phase 4: Development Part 2

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

Project 3: Future Sales Prediction

In the previous Phase 3 we have completed the process of uploading the dataset into dataframe and completed the preprocessing .

Now In this project we are going to implement the process of Model selection , training and evaluation of the model.

Code and Explanation :

Utilize a dataset containing historical sales data. Here a csv file is converted to a DataFrame and the pandas object is used.

```
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]
```

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.

```
df['Total_Spent'] = df['TV'] + df['Radio'] + df['Newspaper']

print(df)
```

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

[200 rows x 5 columns]
```

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.

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

```
        TV  Radio  Newspaper  Sales  Total_Spent  Previous_Sales
0    230.1   37.8       69.2   22.1        337.1             NaN
1     44.5   39.3       45.1   10.4        128.9            22.1
2     17.2   45.9       69.3   12.0        132.4            10.4
3    151.5   41.3       58.5   16.5        251.3            12.0
4    180.8   10.8       58.4   17.9        250.0            16.5
..     ...    ...        ...    ...          ...             ...
195   38.2    3.7       13.8    7.6         55.7            17.3
196   94.2    4.9        8.1   14.0        107.2             7.6
197  177.0    9.3        6.4   14.8        192.7            14.0
198  283.6   42.0       66.2   25.5        391.8            14.8
199  232.1    8.6        8.7   18.4        249.4            25.5

[200 rows x 6 columns]
```

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.

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

```
        TV  Radio  Newspaper  Sales  Total_Spent  Previous_Sales
0    230.1   37.8       69.2   22.1        337.1             NaN
1     44.5   39.3       45.1   10.4        128.9            22.1
2     17.2   45.9       69.3   12.0        132.4            10.4
3    151.5   41.3       58.5   16.5        251.3            12.0
4    180.8   10.8       58.4   17.9        250.0            16.5
..     ...    ...        ...    ...          ...             ...
195   38.2    3.7       13.8    7.6         55.7            17.3
196   94.2    4.9        8.1   14.0        107.2             7.6
197  177.0    9.3        6.4   14.8        192.7            14.0
198  283.6   42.0       66.2   25.5        391.8            14.8
199  232.1    8.6        8.7   18.4        249.4            25.5

     TV_Radio_Interact
0              8697.78
1              1748.85
2               789.48
3              6256.95
4              1952.64
..                 ...
195             141.34
196             461.58
197            1646.10
198           11911.20
199            1996.06

[200 rows x 7 columns]
```

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 nonlinear.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.

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

```
      TV  Radio  Newspaper  Sales  Total_Spent  Previous_Sales
0    230.1   37.8       69.2   22.1        337.1             NaN
1     44.5   39.3       45.1   10.4        128.9            22.1
2     17.2   45.9       69.3   12.0        132.4            10.4
3    151.5   41.3       58.5   16.5        251.3            12.0
4    180.8   10.8       58.4   17.9        250.0            16.5
..     ...    ...        ...    ...          ...             ...
195   38.2    3.7       13.8    7.6         55.7            17.3
196   94.2    4.9        8.1   14.0        107.2             7.6
197  177.0    9.3        6.4   14.8        192.7            14.0
198  283.6   42.0       66.2   25.5        391.8            14.8
199  232.1    8.6        8.7   18.4        249.4            25.5

     TV_Radio_Interact    TV_log
0              8697.78  5.438514
1              1748.85  3.795489
2               789.48  2.844909
3              6256.95  5.020586
4              1952.64  5.197391
..                 ...       ...
195             141.34  3.642836
196             461.58  4.545420
197            1646.10  5.176150
198           11911.20  5.647565
199            1996.06  5.447168

[200 rows x 8 columns]
```

Now we selecting the linear regression as the model

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.

```
from sklearn.linear_model import LinearRegression from
sklearn.model_selection import train_test_split import
pandas as pd
```

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.

```
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:

We have correctly initialized and trained a 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:

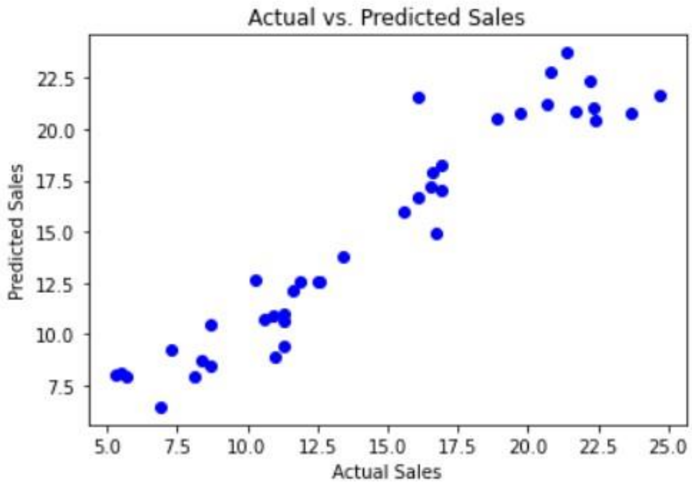
We have successfully used your trained model to make predictions on the test set (X_test). The predicted values are stored in the variable y_pred.This will display the array of predicted values for the test set. Each element in the array corresponds to the predicted value for a specific instance in your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

```
# Predict on the test set y_pred
= model.predict(X_test)
print(y_pred)
```

```
[17.0347724  20.40974033 23.72398873  9.27278518 21.68271879 12.56940161
 21.08119452  8.69035045 17.23701254 16.66657475  8.92396497  8.4817344
 18.2075123   8.06750728 12.64550975 14.93162809  8.12814594 17.89876565
 11.00880637 20.47832788 20.80631846 12.59883297 10.9051829  22.38854775
  9.41796094  7.92506736 20.83908497 13.81520938 10.77080925  7.92682509
 15.95947357 10.63490851 20.80292008 10.43434164 21.5784752  21.18364487
 12.12821771 22.80953262 12.60992766  6.46441252]
```

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.

```
import matplotlib.pyplot as plt
# Assuming 'y_test' and 'y_pred' are your actual and predicted values
plt.scatter(y_test, y_pred, color='blue') plt.xlabel('Actual Sales')
plt.ylabel('Predicted Sales') plt.title('Actual vs. Predicted Sales')
plt.show()
```



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.

```
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: 2.9077569102710923
```

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.

```
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
```

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.

```
import numpy as np #
Calculate RMSE rmse
= np.sqrt(mse)
print(f"Root Mean
Squared Error:
{rmse}")
```

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

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.

```
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
```

Here printing the coefficients of the data model

# 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: 0.054509270837219764
Radio: 0.10094536239295575
Newspaper: 0.004336646822034021
Intercept: 4.714126402214134
```