Submitted By: Syed Muhmmad Mustafa

Registration No: 22108130

Submitted to: Dr. Tehreem Qasim

Assignment #01

Course: Data Mining

Department: Computer Science – BS (AI)

Date: 17-03-2024

Shaheed Zulfikar Ali Bhutto Institute of Science and Technology

Islamabad

Report

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Importing all the necessary libraries:

1. Numpy is used for handling numerical arrays efficiently. In Linear Regression numpy arrays are often used to represent features (X) and target vectors (y)
2. Pandas is used for data manipulation and preprocessing. In Linear Regression it used to loading and preprocessing datasets in python
3. Matplotlib is used for data visualization.
4. Train_test_split is a function from the sklearn library, which is used to split the dataset into two sets, one being the trained set and the other being the testing set
5. LinearRegression is a class that is used to train the model on the training data by finding the coefficients that minimize the residual sum of squares between the observed and predict methods
6. The metrics class is used to import its methods such as mean_absolute_error, mean_squared_error, root_mean_ squared_error and the r2_score.
    a. Mean_absolute_error is used to evaluate the accuracy of the linear regression model, by measuring the absolute difference between the predicted and actual values
    b. Mean_ squared_error is also used to evaluate the accuracy of the linear regression_model but it factors in larger errors and is sensitive to outliers, because it squared the the errors before averaging them
    c. R2score basically checks how well you data fits into the Linear Regression model. If R2 is 1 then the data fits perfectly well into the Linear Regression line, if it is 0 then it does not. If it is in between then it partially fits in

Code:

```
df = pd.read_csv('TV_Marketing.csv')
```

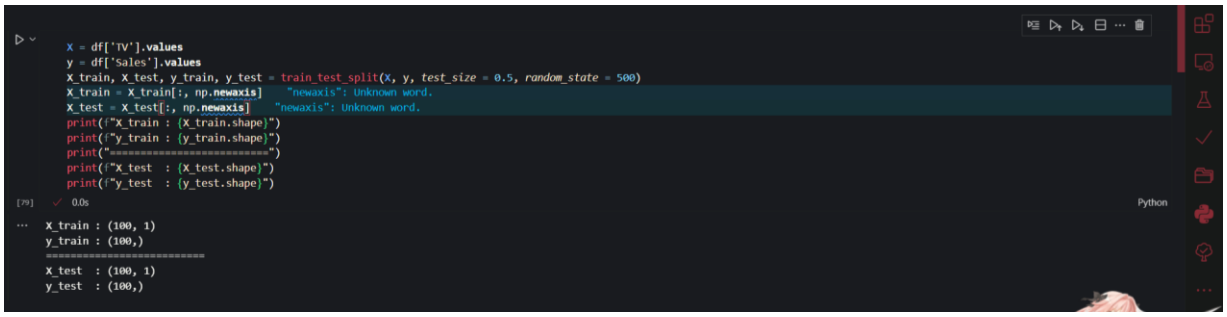
Loading the dataset into an object named df, via pandas

Code:

```
X = df['TV'].values
y = df['Sales'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.9,
random_state = 20)
X_train = X_train[:, np.newaxis]
X_test = X_test[:, np.newaxis]
print(f"X_train : {X_train.shape}")
print(f"y_train : {y_train.shape}")
print("==========================")
print(f"X_test  : {X_test.shape}")
print(f"y_test  : {y_test.shape}")
```

1. X = df[‘TV’].values extracts the column from the df object and stores it into ‘X
2. y = df[‘Sales’].values extracts the column from the df object and stores it into ‘y’
3. The data is split into training and testing sets using the train_test_split function from scikit-learn.
    a. It takes the features (X) and the target variable (y) as inputs and splits them into training and testing sets.
    b. The test_size=0.9 parameter specifies that 90% of the data will be used for testing, and 10% will be used for training.
    c. The random_state=500 parameter ensures reproducibility by fixing the random seed to 500.
4. The training features and testing features, matrix X_train and matrix X_test, both are reshaped to a two-dimensional array using NumPy's newaxis function.
    a. It's necessary because scikit-learn's linear regression model expects the feature matrix to have two dimensions: one for the number of samples and one for the number of features
5. The output is basically printed

Output:



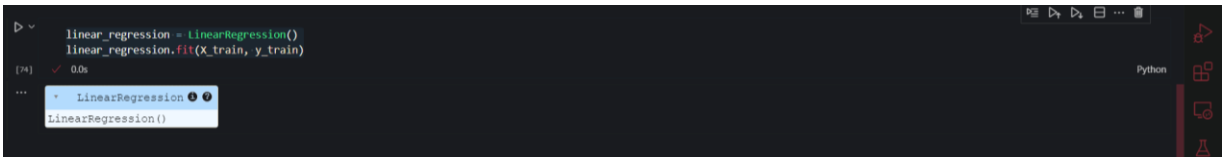
Code:

```
linear_regression = LinearRegression()
linear_regression.fit(X_train, y_train)
```

An instance of the LinearRegression class from scikit-learn, is created

This object will be used later in the code perform linear regression on our data.

Output:



Code:

```
print(f"linear_regression intercept : {linear_regression.intercept_}")
print(f"linear_regression coefficient : {linear_regression.coef_}")
```

The intercept and the coefficients of the Linear Regression model are printed out, using the methods intercept_ and coef_

1. The intercept is the value of the dependent variable (target) when all independent variables (features) are set to zero.
2. Coefficients represent the change in the target variable for a one-unit change in the corresponding feature, assuming all other features remain constant

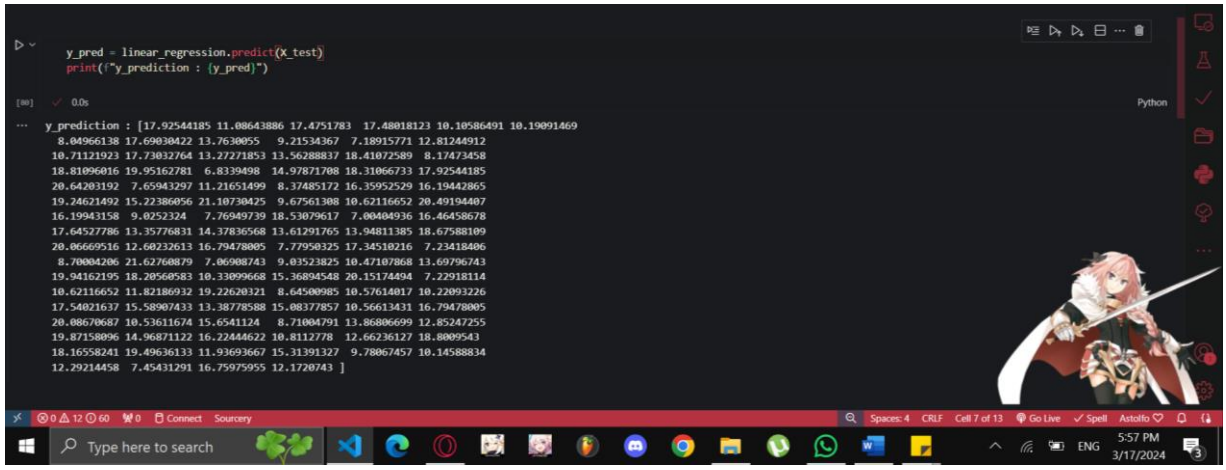
Code:

```
y_pred = linear_regression.predict(X_test)
print(f"y_prediction : {y_pred}")
```

Y_pred is a numpy array that stores the predicted values from the dataset, using the predict method from the Linear Regression class, via the X_test

1. The predict method takes the testing feature matrix X_test as input and returns the predicted target values.
2. After executing this line, y_pred will contain the predicted target values corresponding to the testing data.

Output:



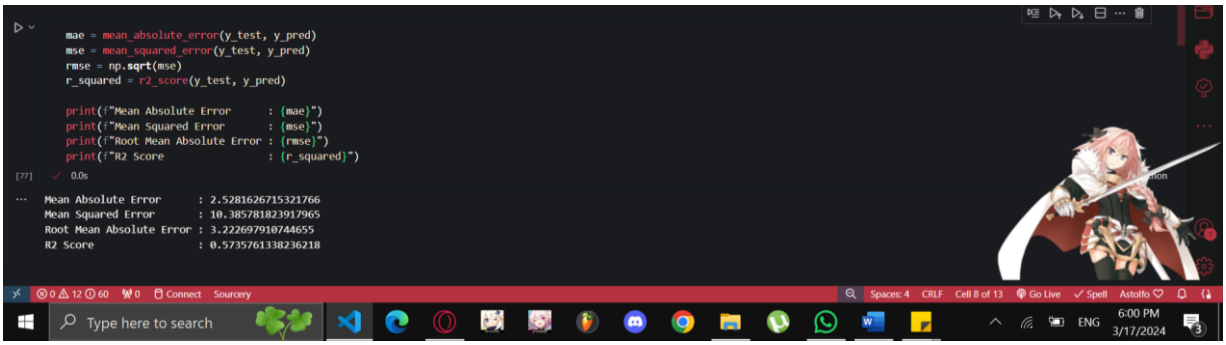
Code:

```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r_squared = r2_score(y_test, y_pred)

print(f"Mean Absolute Error      : {mae}")
print(f"Mean Squared Error       : {mse}")
print(f"Root Mean Absolute Error : {rmse}")
print(f"R2 Score                 : {r_squared}")
```

The values of the mean_absoulte_error, mean_squared_error, root_mean_squared_error and the r2_score are calculated the by using the metrics class

Output:



Code:

```
X_line = np.arange(min(y_test), max(y_test), 1)

plt.scatter(y_test, y_pred, c = 'black', alpha = 0.7, label = 'Data Points')
plt.plot(X_line, X_line, color = 'red', linestyle = '-', linewidth = 2, label =
'LinearRegression')

plt.xlabel('(Actual) y_test')
plt.ylabel('(Predicted) y_pred')
plt.grid(True)
plt.legend()
plt.show()
```

The above code is used to plot the scatter plot graph and the linear regression line.

Output: