



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 is 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 predicted 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 errors before averaging them
  - c. R2score basically checks how well your 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:

```
x = df['TV'].values
y = df['Sales'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5, random_state = 500)
X_train = X_train[:, np.newaxis]
X_test = X_test[:, np.newaxis]
print("X_train : (X_train.shape)")
print("y_train : (y_train.shape)")
print("X_test : (X_test.shape)")
print("y_test : (y_test.shape)")

[79] ✓ 0.0s

... X_train : (100, 1)
    y_train : (100,)
    X_test  : (100, 1)
    y_test  : (100,)
```

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:

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

[74] ✓ 0.0s

... * LinearRegression
    LinearRegression()
```

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

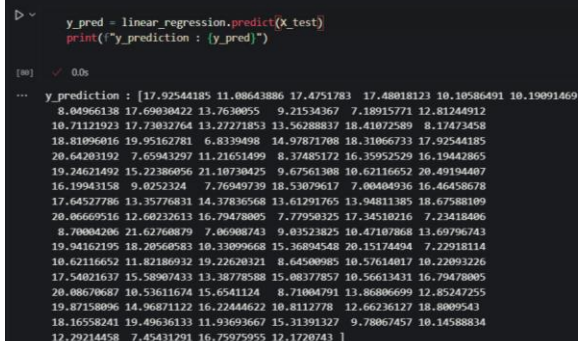
Code:

```
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:



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

[90] ✓ 0.0s

... y\_prediction : [17.92544185 11.08643886 17.4751783 17.48018123 10.10586491 10.19091469  
8.04966138 17.69030422 13.7630055 9.21534367 7.18915771 12.81244912  
10.71121923 17.73032764 13.27271853 13.56288837 18.41072589 8.17473458  
18.81096016 19.95162781 6.8339498 14.97871708 18.31066733 17.92544185  
20.64203192 7.65943297 11.21651499 8.37485172 16.35952529 16.19442865  
19.74621492 15.22386056 21.10730425 9.67561308 10.62116652 20.49194407  
16.19943158 9.0252324 7.76949739 18.53079617 7.00404936 16.46458678  
17.64527786 13.35776831 14.37836568 13.61291765 13.94811385 18.67588109  
20.06669516 12.60232613 16.79478005 7.77950325 17.34510216 7.23418406  
8.70004206 21.62760879 7.06908743 9.03523825 10.47107868 13.60796743  
19.94162195 18.20560583 10.33099668 15.36804548 20.15174004 7.22918114  
10.62116652 11.42186932 19.22020321 8.64500985 10.57614017 10.22093226  
17.54021637 15.58007433 13.38778588 15.08377857 10.56613431 16.79478005  
20.08670687 10.53611674 15.6541124 8.71004791 13.86806699 12.85247255  
19.87158096 14.96871122 16.22444622 10.8112778 12.66236127 18.8009543  
18.16558241 19.49636133 11.93693667 15.31391327 9.78067457 10.14588834  
12.29214458 7.45431291 16.75975955 12.1720743 ]

### 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\_absolute\_error, mean\_squared\_error, root\_mean\_squared\_error and the r2\_score are calculated by using the metrics class

### Output:

```
> \
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}")

[77] ✓ 0.0s
... Mean Absolute Error      : 2.5281626715321766
Mean Squared Error       : 10.385781823917965
Root Mean Absolute Error : 3.222697910744655
R2 Score                     : 0.5735761338236218
```

## 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:

