Linear Regression from scratch in Python

Importing the libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

# Step 1: Load the diabetes dataset
diabetes = load_diabetes()
diabetes_df = pd.DataFrame(data=diabetes.data, columns=diabetes.feature_names)
diabetes_df['target'] = diabetes.target

diabetes_df.head()
```

	age	sex	bmi	bp	s1	s2	s3	s4
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592

```
# splitting the data into X and y
X = diabetes_df.drop('target', axis=1)
y = diabetes_df['target']
```

Shuffling the Data

```
# shuffling the data
def shuffle(X, y):
    np.random.seed(67)
    randomize = np.arange(len(X))
    np.random.shuffle(randomize)
    X = X.iloc[randomize]
    y = y.iloc[randomize]
    return X, y

shuffle(X, y)
```

```
(          age       sex       bmi        bp        s1        s2        s3  \
 295 -0.052738  0.050680  0.039062 -0.040099 -0.005697 -0.012900  0.011824
 114  0.023546 -0.044642  0.110198  0.063187  0.013567 -0.032942 -0.024993
 197  0.048974  0.050680  0.003494  0.070073 -0.008449  0.013404 -0.054446
 255  0.001751 -0.044642 -0.065486 -0.005671 -0.007073 -0.019476  0.041277
 429 -0.041840 -0.044642 -0.033151 -0.022885  0.046589  0.041587  0.056003
 ..        ...       ...       ...       ...       ...       ...       ...
 393 -0.074533 -0.044642 -0.046085 -0.043542 -0.029088 -0.023234  0.015505
 7    0.063504  0.050680 -0.001895  0.066630  0.090620  0.108914  0.022869
 202  0.081666  0.050680  0.001339  0.035644  0.126395  0.091065  0.019187
 309 -0.009147  0.050680  0.001339 -0.002228  0.079612  0.070084  0.033914
 323  0.070769  0.050680 -0.007284  0.049415  0.060349 -0.004445 -0.054446

           s4        s5        s6
 295 -0.039493  0.016305  0.003064
 114  0.020655  0.099240  0.023775
 197  0.034309  0.013316  0.036201
 255 -0.039493 -0.003304  0.007207
 429 -0.024733 -0.025952 -0.038357
 ..        ...       ...       ...
 393 -0.039493 -0.039810 -0.021788
 7    0.017703 -0.035817  0.003064
 202  0.034309  0.084495 -0.030072
 309 -0.002592  0.026714  0.081764
 323  0.108111  0.129019  0.056912

 [442 rows x 10 columns], 295     85.0
 114    258.0
 197    129.0
 255    153.0
 429     94.0
```

```
         ...
393      69.0
7        63.0
202     196.0
309     142.0
323     248.0
Name: target, Length: 442, dtype: float64)
```

```
# make sure that the labels and features are still matching after shuffling the data.
print(pd.DataFrame(X.head()))
print(pd.DataFrame(y.head()))
```

```
        age       sex       bmi        bp        s1        s2        s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
2  0.085299  0.050680  0.044451 -0.005671 -0.045599 -0.034194 -0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142

         s4        s5        s6
0 -0.002592  0.019908 -0.017646
1 -0.039493 -0.068330 -0.092204
2 -0.002592  0.002864 -0.025930
3  0.034309  0.022692 -0.009362
4 -0.002592 -0.031991 -0.046641
   target
0   151.0
1    75.0
2   141.0
3   206.0
4   135.0
```

Splitting the data into Train ,dev and test

```
# Step 2: Split the dataset into train, dev, and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
X_dev, X_test, y_dev, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=42)
```

Implementing Univariate LinearRegression using Gradient descent

```
# Implement univariate linear regression using gradient descent
class UnivariateLinearRegression:
    # constructor to initialize learning rate and number of iterations
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations

    # fit method to train the model on input features and target values
    def fit(self, X, y):
        # initialize bias term to 0
        self.bias = 0
        # initialize weights to 0
        self.weights = np.zeros(X.shape[1])
        # number of samples
        N = len(y)

        # gradient descent optimization loop
        for i in range(self.num_iterations):
            # compute predicted target values
            y_pred = np.dot(X, self.weights) + self.bias
            # compute gradient of mean squared error with respect to weights
            dw = (1/N) * np.dot(X.T, (y_pred - y))
            # compute gradient of mean squared error with respect to bias
            db = (1/N) * np.sum(y_pred - y)
            # update weights and bias
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    # predict method to return predicted target values for a given input
    def predict(self, X):
        return np.dot(X, self.weights) + self.bias
```

Choosing the best feature

```
# choosing the best feature for the model
def bestFeature(X_train, y_train, X_dev, y_dev):
    # initialize the best feature and its corresponding RMSE
    best_feature = None
    # initialize the best RMSE to infinity
    best_rmse = float('inf')

    # loop over all features
    for feature in X_train.columns:
        # train the model on the current feature
        model = UnivariateLinearRegression()
        model.fit(X_train[feature].values.reshape(-1, 1), y_train)
        # compute the RMSE on the dev set
        y_pred = model.predict(X_dev[feature].values.reshape(-1, 1))
        rmse = np.sqrt(np.mean((y_pred - y_dev) ** 2))
        mse = np.mean((y_pred - y_dev) ** 2)
        # print each feature's MSE and RMSE
        print("RMSE and MSE for feature {}: \t{}, \t{}".format(feature, rmse, mse))
        # get the best feature
        if rmse < best_rmse:
            best_rmse = rmse
            best_feature = feature

    return best_feature, best_rmse


best_feature, best_score = bestFeature(X_train, y_train, X_dev, y_dev)
best_feature, best_score
```

```
RMSE and MSE for feature age:   71.74923996744872,      5147.95343590654
RMSE and MSE for feature sex:   71.79702426438132,      5154.81269322016
RMSE and MSE for feature bmi:   71.1546394665598,       5062.982717616109
RMSE and MSE for feature bp:    71.42481282185534,      5101.503886637071
RMSE and MSE for feature s1:    71.6871522034209,       5139.0477910364325
RMSE and MSE for feature s2:    71.72058653971676,      5143.842533601001
RMSE and MSE for feature s3:    71.48528448361391,      5110.1458977032125
RMSE and MSE for feature s4:    71.39989187488817,      5097.9445597457225
RMSE and MSE for feature s5:    71.14831660995208,      5062.082956429983
RMSE and MSE for feature s6:    71.43676326889366,      5103.211146335953
('s5', 71.14831660995208)
```

Defining the cost functions

```
# Mean Squared Error function
def mse_score(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

# Root Mean Squared Error function
def rmse_score(y_true, y_pred):
    return np.sqrt(np.mean((y_true - y_pred) ** 2))

# R^2 Score function
def r2_score(y_true, y_pred):
    return 1 - np.sum((y_true - y_pred) ** 2) / np.sum((y_true - np.mean(y_true)) ** 2)
```

Testing the model on the test data

```
#Test the model using the test set
# initialize the model
best_model = UnivariateLinearRegression()
# fit the model on the best feature
best_model.fit(X_train[best_feature].values.reshape(-1, 1), y_train)
# predict the target values on the test set
y_pred = best_model.predict(X_test[best_feature].values.reshape(-1, 1))

# compute the MSE on the test set
mse = mse_score(y_test, y_pred)
# compute the RMSE on the test set
rmse = rmse_score(y_test, y_pred)

print("RMSE on the test set: ", rmse)
print("MSE on the test set: ", mse)
print("r^2 score", r2_score(y_test, y_pred))
```

```
RMSE on the test set:  74.9462975460578
MSE on the test set:  5616.94751586223
r^2 score 0.010748736949064863
```

0s completed at 11:25 AM