Regression metrics

Time limit for program runtime	1 second
Memory limit	64 Mb
Input	standard input or input.txt
Output	standard output or output.txt

Implement regression metrics using only the `NumPy` and `math` libraries. Feel free to choose whatever implementation approach you prefer (create a separate function for each metric or place all of them in one single function).

Functions take two datasets as input: the true values of the target feature and the predicted ones. You need to calculate three regression metrics: MAE, MSE, and RMSE. Return the result of the calculated metrics that evaluate the differences between true and predicted values of the target feature in the given format.

Input format

The first input line contains space-separated true values of the target feature, and the second input line contains space-separated predicted values of the target feature.

Output format

Your code must output each metric from a new line in the following format: "metric name: metric value rounded to two decimal places". Metric values should be rounded according to the logic implemented by the round function. Metrics should be output in the following order: MSE, MAE, and RMSE.

Sample

Input	Output

`1 5 2 9 2 3`
`1.5 5.2 2.3 8.7 2.8 3.3`
	`MSE: 0.20`
	`MAE: 0.40`
	`RMSE: 0.45`

Notes

You can't use the sklearn library.

R^2

Time limit for program runtime	**1 second**
Memory limit	**64 Mb**
Input	**standard input or input.txt**
Output	**standard output or output.txt**

Besides the metrics from the previous task, there's one more metric that can help evaluate the quality of a regression model: the coefficient of determination (R^2). It shows the proportion of variance in your target feature that can be explained by the model.

For calculations, use the formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(\bar{y} - y_i)^2},$$

where $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$ — is the mean true value.

Calculate this metric using only the `NumPy` and `math` libraries.

The function takes two datasets as input: the true values of the target feature and the predicted ones. Note that the output depends on the order of datasets: the mean is calculated for true values.

Input format

The first input line contains space-separated true values of the target feature, and the second input line contains space-separated predicted values of the target feature.

Output format

Your code must output the metric value in the following format: "R2: metric value rounded to two decimal places". Metric values should be rounded according to the logic implemented by the round function.

Sample

Input	Output
`1 5 2 9 2 3` `1.5 5.2 2.3 8.7 2.8 3.3`	`R2: 0.97`

Linear regression: analytical solution

Time limit for program runtime	**1 second**
Memory limit	**64 Mb**
Input	**standard input or input.txt**
Output	**standard output or output.txt**

Implement the analytical solution of Linear regression using the method of least squares. You can't use any ready-made methods.

Write the function `analytical_solution(X: np.ndarray, y: np.ndarray, fit_intercept: bool = True) -> np.ndarray`, where:

- X is a NumPy matrix of $\ell \times \mathrm{d}$ object features.
- y is a NumPy vector of true labels of $\ell \times 1$ objects (target).
- `fit_intercept` indicates whether to calculate the intercept for this linear regression model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered). By default, the named parameter must be `fit_intercept=True`.

To calculate the analytical solution of linear regression, use the formula:

$$\mathrm{w} = (\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}\mathrm{y}$$

The function returns trained model weights in the format of a d-sized `np.ndarray`, where d is the number of features in the training dataset. If `fit_intercept=True`, the length of the weight array will be $\mathrm{d} + 1$, where the first element of the array will be the intercept coefficient.

Sample

Input	Output

`import numpy as np`
 	
`X = np.array([[0.7, 1], [0.3, 2]])`
`y = np.array([0.7, 0.3])`
`print(analytical_solution(X, y, fit_intercept=False))`
	`[1.00000000e+00 1.73093862`

Notes

The file submitted to the checking system must only contain the function described in the task and, possibly, helper functions. Make sure to also import the libraries that you've used.

From sklearn: Ridge and MSE

Time limit for program runtime	**1 second**
Memory limit	**128 Mb**
Input	**standard input or input.txt**
Output	**standard output or output.txt**

Implement a machine learning model (Linear regression with L2 regularization) and use the MSE metric to evaluate its performance. To do this, use the ready-made methods from the sklearn library: `sklearn.linear_model.Ridge` and `sklearn.metrics.mean_squared_error`.

Write the function `evaluation(X_train: np.ndarray, y_train: np.ndarray, X_test: np.ndarray, y_test: np.ndarray, fit_intercept=True,alpha=1.0) -> float`, where:

- X_train and y_train are NumPy arrays of the features and target variable of the training dataset.
- X_test and y_test are NumPy arrays of the features and target variable of the test dataset.
- `fit_intercept` indicates whether to calculate the intercept for this linear regression model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered). By default, the keyword parameter must be `fit_intercept=True`. This parameter must be specified when initializing the Ridge object.
- α: the regularization coefficient of the ridge regression. Set the default value of the keyword argument in the Ridge object to `alpha=1.0`.

The function must contain a call with the given parameters, the Ridge model training on the training dataset, and the MSE metric value calculated on the true and predicted values of the target feature of the test dataset.

The function returns the MSE metric value for true and predicted values of the target feature on the test dataset.

Sample

Input

```
import numpy as np
X_train = np.array([[0.7, 1], [0.3, 2]])
y_train = np.array([5.7, 7.3])
X_test = np.array([[4, 1.5], [8, 3]])
y_test = np.array([10., 17.])

print(evaluation(X_train, y_train, X_test, y_test, alpha=1.0, fit_intercept=False
```

Notes

The file submitted to the testing system must only contain the function described in the task and, possibly, helper functions. Make sure to also import the libraries that you've used.