**Data leakage in Normalization**

Data leakage refers to a situation in a machine learning project where information from outside the training dataset is used to create the model. This can lead to overly optimistic performance estimates during training and validation, because the model has essentially been given access to data it wouldn't have in a real-world scenario, leading to poor performance on unseen data.

In the context of feature normalization, data leakage can occur if you normalize your entire dataset (including both training and test data) together, before splitting it into training and test sets. The correct procedure is to split your data first and then normalize the training and test data separately.

### How to Avoid Data Leakage During Normalization

1. \*\*Split Your Data First:\*\* Before any preprocessing, including normalization, split your dataset into training, validation, and test sets. This ensures that the preprocessing of the training data does not influence the preprocessing of the test data and vice versa.

2. \*\*Fit the Scaler on Training Data Only:\*\* When normalizing (or standardizing) your features, you should fit your scaler (or normalization parameters) only on the training data. This means calculating the mean and standard deviation (for standard scaling) or the minimum and maximum values (for min-max normalization) using only the training data.

3. \*\*Apply the Same Transformation to Test Data:\*\* After fitting the scaler on the training data, use the same parameters (mean, standard deviation, min, max, etc.) to transform the test data. This mimics the real-world scenario where the model is applied to new, unseen data, ensuring that the test data is scaled based on the distribution of the training data only.

### Example in Python

Assuming you are using `scikit-learn` for normalization, here is how you might properly normalize your data to avoid data leakage:

```python

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

# Assume X is your feature matrix and y are your labels

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a StandardScaler object

scaler = StandardScaler()

# Fit on the training data

scaler.fit(X\_train)

# Transform the training data

X\_train\_scaled = scaler.transform(X\_train)

# Transform the test data using the same scaler

X\_test\_scaled = scaler.transform(X\_test)

```

By following these steps, you can prevent data leakage during the normalization process, ensuring that your model's evaluation metrics accurately reflect its performance on truly unseen data.