6.08: Data Leakage in Machine Learning

In [ ]:
```
# Training(Classroom), Validation(Mock Test), Test(JEE Examp)
# Data leakage - using cheat in mock test
```

Data leakage occurs when information that would not be available at prediction time is used to train the model. This makes validation scores unrealistically high and the model will fail in real deployment.

Dataset Recap

In [ ]:
```
# Lecture 6.6
```

Columns include:

1. IDs (`order_id, customer_id, product_id`)
2. Dates (`order_date, ship_date`)
3. Order details (`category, price, quantity, payment_type, city`)
4. Free-text (`review_text`)
5. Target = `returned` (whether order was returned)

Define Problem Statement (prediction moment)

- Assume we want to predict at order time whether an order will be returned.
- This means at prediction, we know `order_date, product, price, customer info` , but we don't know `ship_date` , `review_text` , or future `returns` .

Temporal problems

Prediction or forecasting tasks such as:

- Returns
- Delivery delays
- Sales forecasting

These must respect chronology.

Non-temporal problems

Classification or segmentation tasks such as:

- Customer clustering
- Product categorization

For these, time order is irrelevant.

Types of Leakage

(a) Future Info leakage

Incorrect: Using `ship_date` to predict return.
At prediction time, the order is placed but not yet shipped, so `ship_date` leaks future information.

Incorrect: Using `review_text`.
Reviews are left after receiving the product, so they contain post-outcome information.

Correct: Drop `ship_date` and `review_text` when training the model.

(b) Train–test contamination

Incorrect: Applying a scaler on the entire dataset before splitting.
For example, fitting a StandardScaler on the whole dataset means the scaler has already seen the distribution of the test data.

```
# Fit scaler + Transform(Encode) on FULL dataset (train + test info leaks in)
scaler = StandardScaler()
X[['price','quantity']] = scaler.fit_transform(X[['price','quantity']])
```

Correct: Split the dataset first, then fit the scaler only on the training set. Apply the same scaler to the test set.

(c) Temporal leakage

Incorrect: Random splitting may place orders from December 2024 in the training set while placing January 2024 orders in the test set.

This allows the model to see information from the future, such as seasonal patterns.

```
# Random split (temporal leakage)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Correct: Use a time-based split on `order_date`.

For example, training set = January–October 2024, test set = November–December 2024.

(d) Target leakage

- Target data for test dataset is leaked in many ways. one such way is - Target Encoding
- Instead of one-hot encoding a categorical variable (like product_id),
  we replace each category with some function of the target — usually the mean return rate for that category.

```
# wrong way
mean_return_rate = df.groupby('product_id')['returned'].mean()
df['product_te'] = df['product_id'].map(mean_return_rate)
```

- Suppose product P50 has 10 orders, 7 of which were returned. Every order for P50 gets encoded as 0.7.
- **Solution** : For each order, compute mean return rate of that category using only past orders (order_date < current).

Perfect checklist for the ecommerce dataset

- Drop leakage-prone features first

```
df = df.drop(columns=['order_id','ship_date','review_text'])
```

- Time-based split (simulate real-world scenario)

```
df = df.sort_values('order_date')
split_point = int(len(df)*0.8)
X_train, X_test = df.iloc[:split_point].drop(columns=['returned']),
df.iloc[split_point:].drop(columns=['returned'])
y_train, y_test = df.iloc[:split_point]['returned'], df.iloc[split_point:]['returned']
```

- Scale numeric features using ONLY train stats

```
scaler = StandardScaler()
X_train[['price','quantity']] = scaler.fit_transform(X_train[['price','quantity']])
X_test[['price','quantity']] = scaler.transform(X_test[['price','quantity']])
```

- Encode categorical features using ONLY train categories

```
ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)

X_train_cat = pd.DataFrame(
    ohe.fit_transform(X_train[['category','payment_type','city']]),
    index=X_train.index
)
X_test_cat = pd.DataFrame(
    ohe.transform(X_test[['category','payment_type','city']]),
    index=X_test.index
)
```

- Replace categorical columns with encoded ones

- Train model