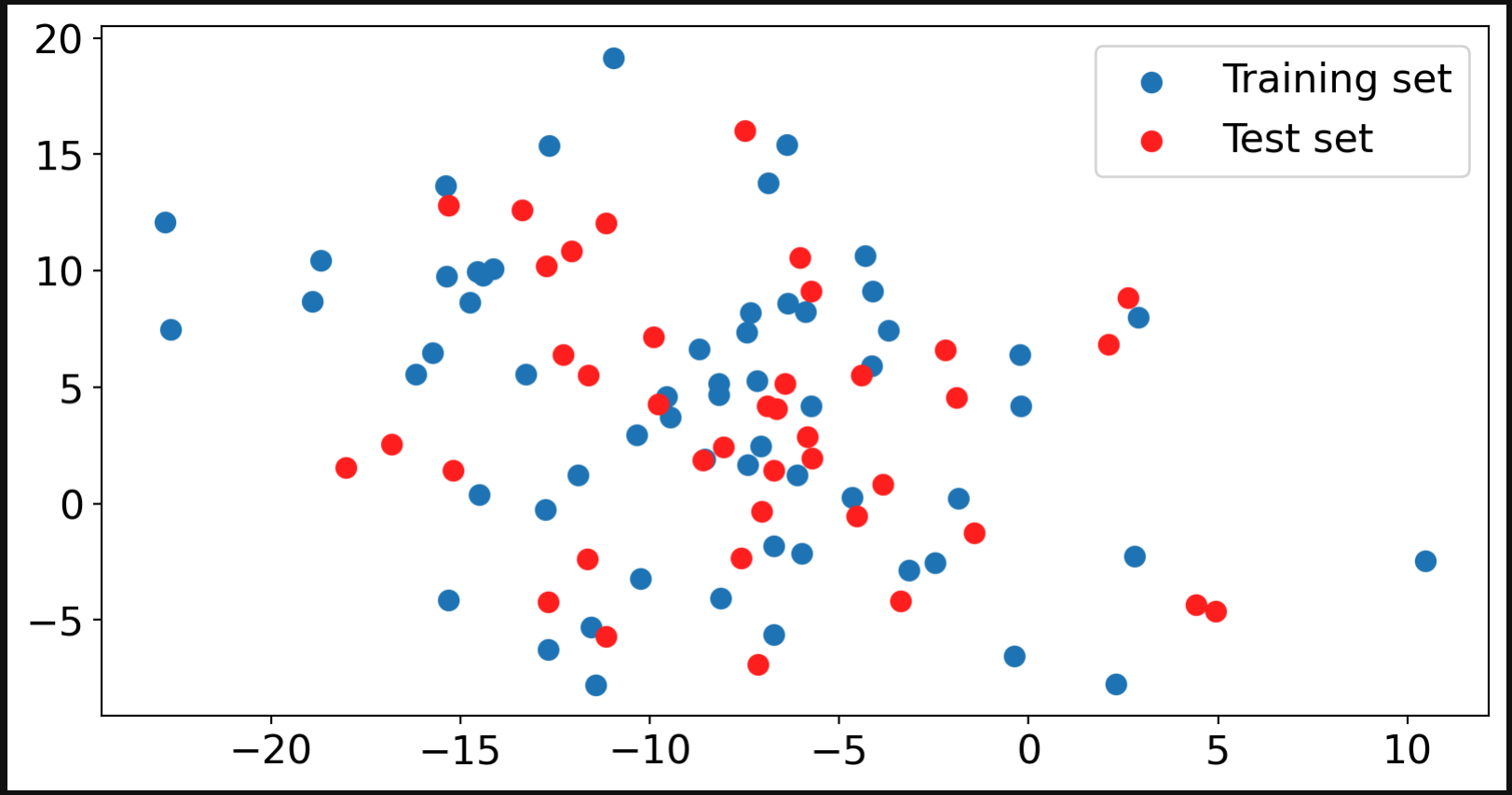
Lecture 6: Column transformer and text features

Firas Moosvi (Slides adapted from Varada Kolhatkar)

Recap: Preprocessing mistakes

Data

```
X, y = make_blobs(n_samples=100, centers=3, random_state=12, cluster_std=5) # make synthetic data
X_train_toy, X_test_toy, y_train_toy, y_test_toy = train_test_split(
    X, y, random_state=5, test_size=0.4) # split it into training and test sets
# Visualize the training data
plt.scatter(X_train_toy[:, 0], X_train_toy[:, 1], label="Training set", s=60)
plt.scatter(
    X_test_toy[:, 0], X_test_toy[:, 1], color=mglearn.cm2(1), label="Test set", s=60
)
plt.legend(loc="upper right")
```



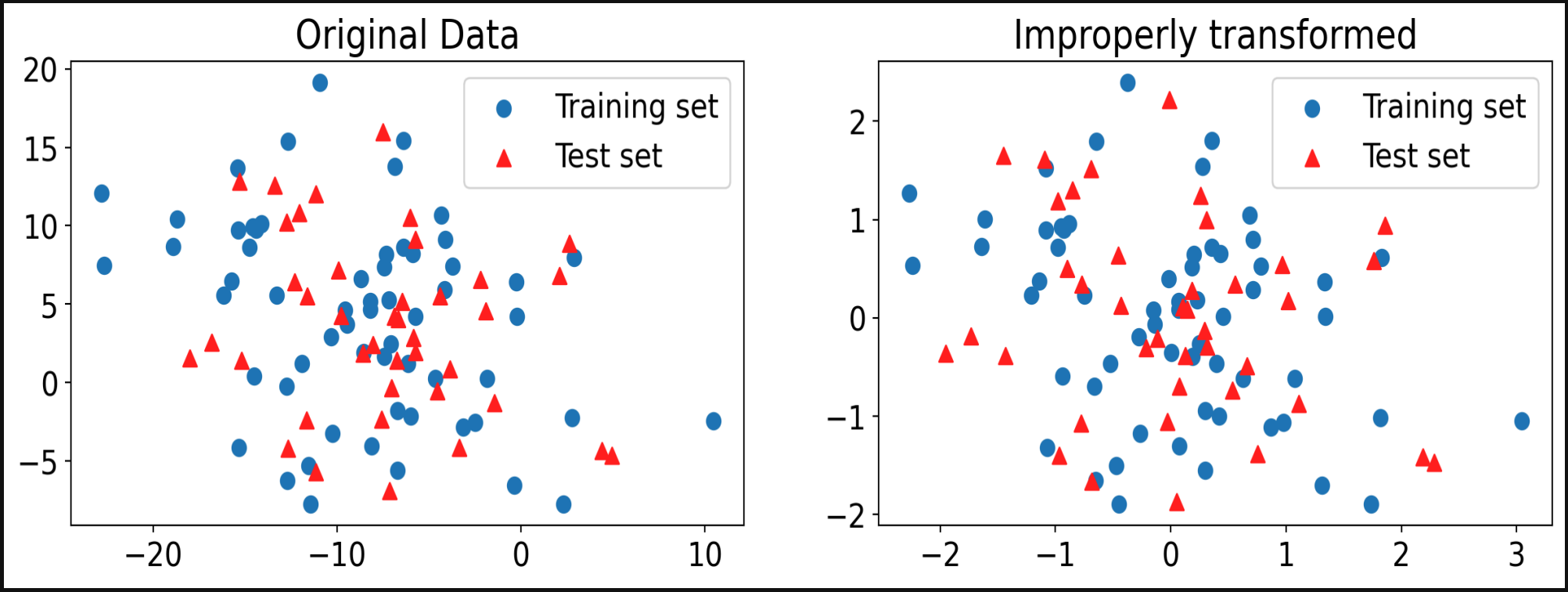
❌ Bad methodology 1

- What's wrong with scaling data separately?

```
scaler = StandardScaler() # Creating a scalert object
scaler.fit(X_train_toy) # Calling fit on the training data
train_scaled = scaler.transform(
    X_train_toy
)  # Transforming the training data using the scaler fit on training data

scaler = StandardScaler()  # Creating a separate object for scaling test data
scaler.fit(X_test_toy)  # Calling fit on the test data
test_scaled = scaler.transform(
    X_test_toy
)  # Transforming the test data using the scaler fit on test data

knn = KNeighborsClassifier()
knn.fit(train_scaled, y_train_toy)
print(f"Training score: {knn.score(train_scaled, y_train_toy):.2f}")
print(f"Test score: {knn.score(test_scaled, y_test_toy):.2f}") # misleading scores
```

```
Training score: 0.63
Test score: 0.60
```

Scaling train and test data separately



❌ Bad methodology 2

- What's wrong with the approach below?

```
# join the train and test sets back together
XX = np.vstack((X_train_toy, X_test_toy))

scaler = StandardScaler()
scaler.fit(XX)
XX_scaled = scaler.transform(XX)

XX_train = XX_scaled[:X_train_toy.shape[0]]
XX_test = XX_scaled[X_train_toy.shape[0]:]

knn = KNeighborsClassifier()
knn.fit(XX_train, y_train_toy)
print(f"Training score: {knn.score(XX_train, y_train_toy):.2f}")  # Misleading score
print(f"Test score: {knn.score(XX_test, y_test_toy):.2f}")  # Misleading score
```

```
Training score: 0.63
Test score: 0.55
```

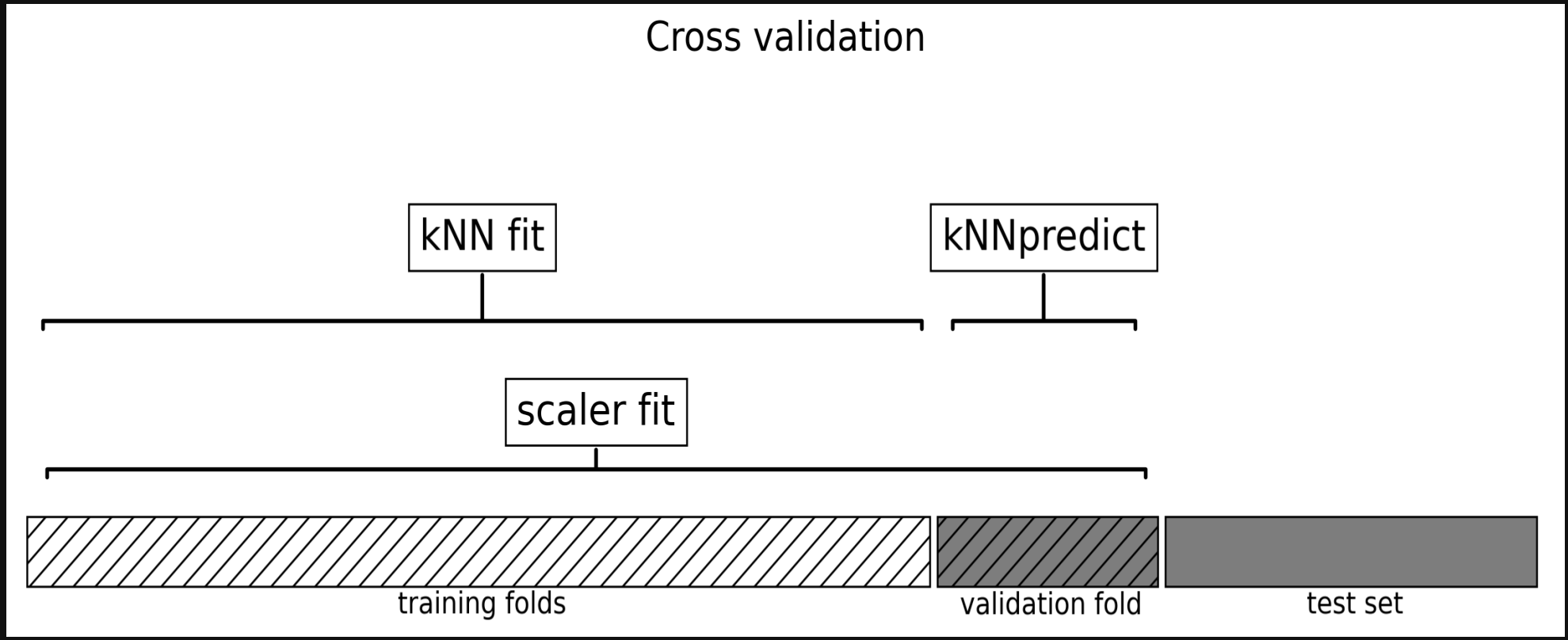
❌ Bad methodology 3

- What's wrong with the approach below?

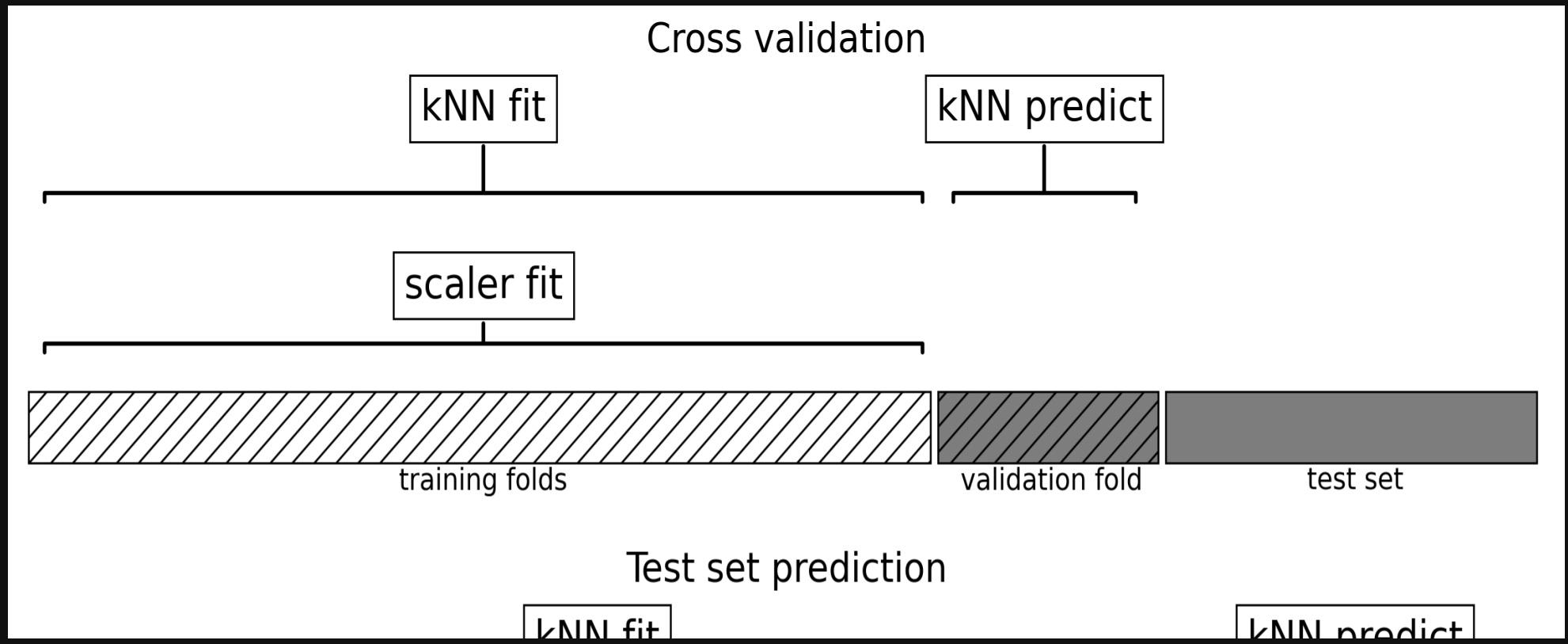
```
knn = KNeighborsClassifier()

scaler = StandardScaler()
scaler.fit(X_train_toy)
X_train_scaled = scaler.transform(X_train_toy)
X_test_scaled = scaler.transform(X_test_toy)
cross_val_score(knn, X_train_scaled, y_train_toy)
```

```
array([0.25      , 0.5       , 0.58333333, 0.58333333, 0.41666667])
```

Improper preprocessing



Proper preprocessing



Common scenarios of data leakage ⛔

- **Fitting preprocessing on validation or test data:** The preprocessing "learns" from data it shouldn't see, giving your model unfair information.
- **Fitting preprocessing on the full dataset (train + validation/test):** Even a small peek at validation/test data contaminates the training process.
- **Fitting preprocessing before cross-validation:** The preprocessing step has already "seen" all the data, so each validation fold is no longer truly unseen.

Avoiding data leakage ✅

- Ensure the **training process remains completely independent of the validation/test data**.
- Fit preprocessing steps only on the training data (e.g., imputer learns medians from training set).
- Apply the same preprocessing steps to both training and validation/test data.
- Use the validation set only for hyperparameter tuning. It must not contribute to model training.
- Use `sklearn` **pipelines** to automatically handle preprocessing and modeling without breaking the golden rule.

Recap: sklearn Pipelines

- Pipeline is a way to chain multiple steps (e.g., preprocessing + model fitting) into a single workflow.
- If you build a pipeline with preprocessing + model, then during cross-validation:
  - Each preprocessing step fits and transforms only on the training fold.
  - The same preprocessing steps are then applied (transform only) to the validation fold.
  - Finally, the model is trained on the preprocessed training data and evaluated on the preprocessed validation data.

Recap: sklearn Pipelines

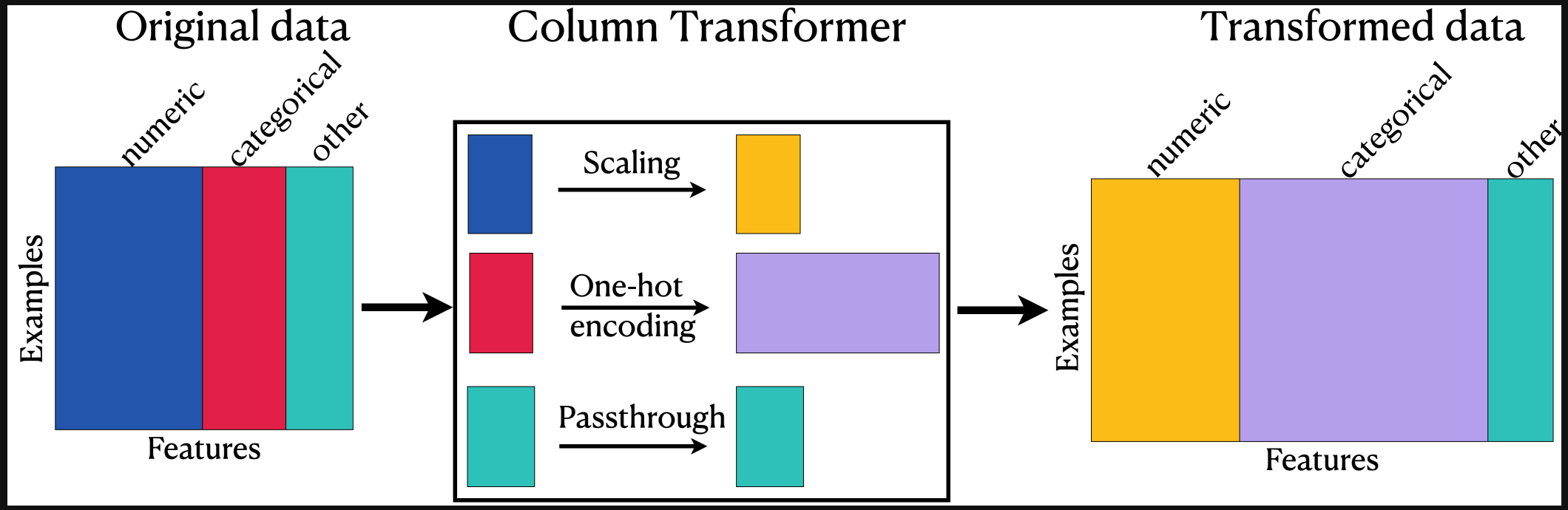
- Simplify the code and improves readability.
- Reduce the risk of data leakage by ensuring proper transformation of the training and test sets.
- Automatically apply transformations in sequence.
- **Example:**
  - Chaining a `StandardScaler` with a `KNeighborsClassifier` model.

```
from sklearn.pipeline import make_pipeline

pipe_knn = make_pipeline(StandardScaler(), KNeighborsClassifier())

# Correct way to do cross validation without breaking the golden rule.
cross_val_score(pipe_knn, X_train_toy, y_train_toy)
```

```
array([0.25      , 0.5       , 0.5       , 0.58333333, 0.41666667])
```

sklearn's ColumnTransformer

- Use ColumnTransformer to build all our transformations together into one object



- Use a column transformer with sklearn pipelines.

(iClicker) Exercise 6.1

Select all of the following statements which are TRUE.

- a. You could carry out cross-validation by passing a `ColumnTransformer` object to `cross_validate`.
- b. After applying column transformer, the order of the columns in the transformed data has to be the same as the order of the columns in the original data.
- c. After applying a column transformer, the transformed data is always going to be of different shape than the original data.
- d. When you call `fit_transform` on a `ColumnTransformer` object, you get a numpy ndarray.

More preprocessing

Remarks on preprocessing

- There is no one-size-fits-all solution in data preprocessing, and decisions often involve a degree of subjectivity.
  - Exploratory data analysis and domain knowledge inform these decisions
- Always consider the specific goals of your project when deciding how to encode features.

Alternative methods for scaling

- StandardScaler
  - Good choice when the column follows a normal distribution or a distribution somewhat like a normal distribution.
- MinMaxScaler: Transform each feature to a desired range. Appropriate when
  - Good choice for features such as human age, where there is a fixed range of values and the feature is uniformly distributed across the range
- Normalizer: Works on rows rather than columns. Normalize examples individually to unit norm.
  - Good choice for frequency-type data
- Log scaling
  - Good choice for features following power law distribution (e.g., counts of followers, salary or income)
- ...

Ordinal encoding vs. One-hot encoding

- Ordinal Encoding: Encodes categorical features as an integer array.
- One-hot Encoding: Creates binary columns for each category's presence.
- Sometimes how we encode a specific feature depends upon the context.

Ordinal encoding vs. One-hot encoding

- Consider **weather** feature and its four categories: Sunny (☀️), Cloudy (🌥️), Rainy (⛈️), Snowy (❄️)
- Which encoding would you use in each of the following scenarios?
  - **Predicting traffic volume**
  - **Predicting severity of weather-related road incidents**

Ordinal encoding vs. One-hot encoding

- Consider **weather** feature and its four categories: Sunny (☀️), Cloudy (🌥️), Rainy (⛈️), Snowy (❄️)
- **Predicting traffic volume:** Using one-hot encoding would make sense here because the impact of different weather conditions on traffic volume does not necessarily follow a clear order and different weather conditions could have very distinct effects.
- **Predicting severity of weather-related road incidents:** An ordinal encoding might be more appropriate if you define your weather categories from least to most severe as this could correlate directly with the likelihood or severity of incidents.

How to deal with unseen categories?

`handle_unknown = "ignore"` of `OneHotEncoder`

- Reasonable use: When unseen categories are less likely to impact the model's prediction accuracy (e.g., product categories in e-commerce), and you prefer to avoid breaking the model.
- Not-so-reasonable use: When unseen categories could provide critical new information that could significantly alter predictions (e.g., in medical diagnostics), ignoring them could result in a poor or dangerous outcome.

`drop="if_binary"` argument of OneHotEncoder

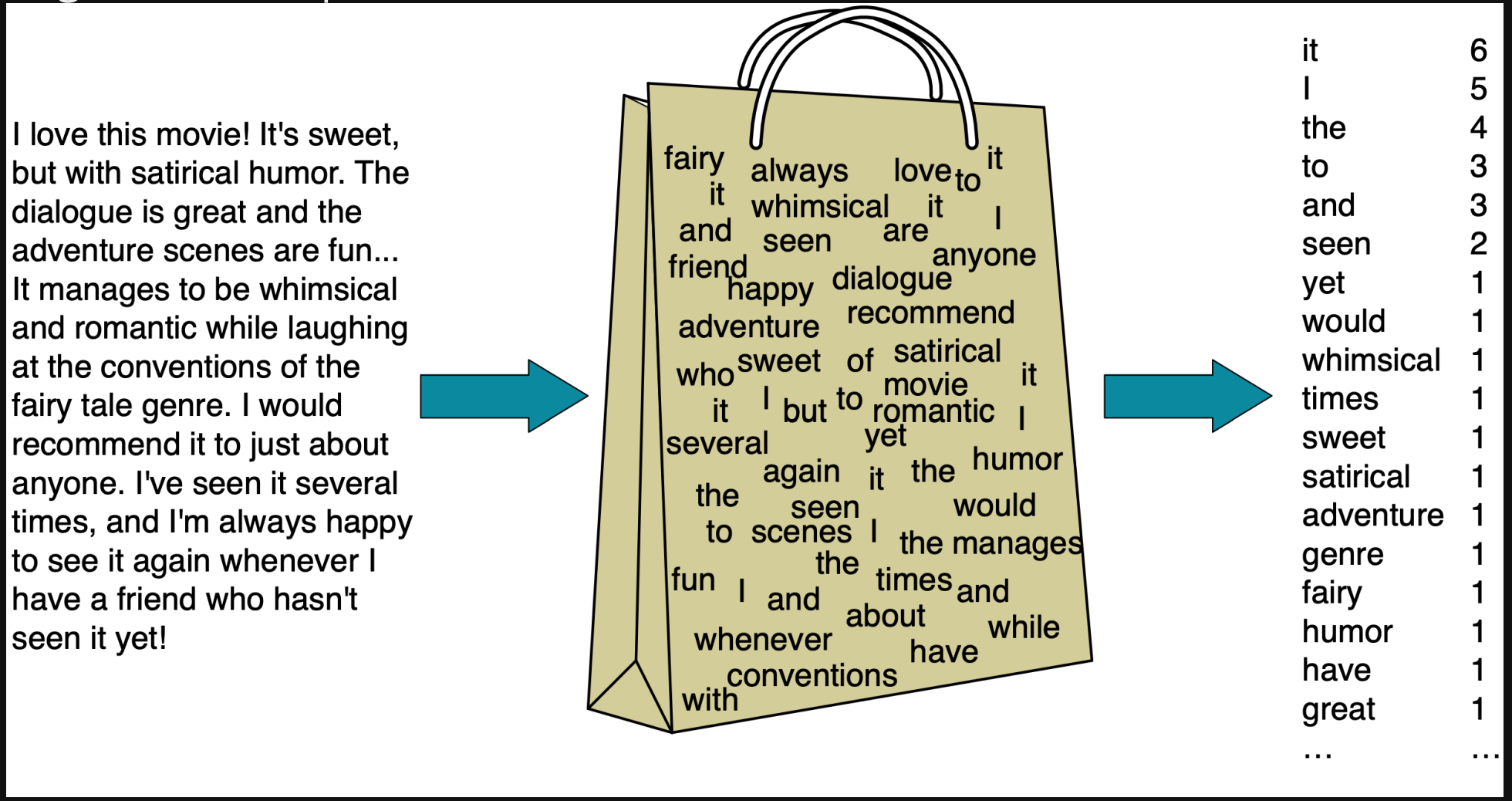
- drop='if_binary' argument in OneHotEncoder:
- Reduces redundancy by dropping one of the columns if the feature is binary.

Categorical variables with too many categories

- Strategies for categorical variables with too many categories:
  - Dimensionality reduction techniques
  - Bucketing categories into ‘others’
  - Clustering or grouping categories manually
  - Only considering top-N categories
  - ...

Dealing with text features

- Preprocessing text to fit into machine learning models using text vectorization.
- Bag of words representation



`sklearn CountVectorizer`

- Use `scikit-learn`'s `CountVectorizer` to encode text data
- `CountVectorizer`: Transforms text into a matrix of token counts
- Important parameters:
  - `max_features`: Control the number of features used in the model
  - `max_df`, `min_df`: Control document frequency thresholds
  - `ngram_range`: Defines the range of n-grams to be extracted
  - `stop_words`: Enables the removal of common words that are typically uninformative in most applications, such as "and", "the", etc.

Incorporating text features in a machine learning pipeline

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 from sklearn.svm import SVC
3 from sklearn.pipeline import make_pipeline
4
5 text_pipeline = make_pipeline(
6     CountVectorizer(),
7     SVC()
8 )
```

(iClicker) Exercise 6.2

Select all of the following statements which are TRUE.

- a. `handle_unknown="ignore"` would treat all unknown categories equally.
- b. As you increase the value for `max_features` hyperparameter of `CountVectorizer` the training score is likely to go up.
- c. Suppose you are encoding text data using `CountVectorizer`. If you encounter a word in the validation or the test split that's not available in the training data, we'll get an error.
- d. In the code below, inside `cross_validate`, each fold might have slightly different number of features (columns) in the fold.

```
1 pipe = (CountVectorizer(), SVC())
2 cross_validate(pipe, X_train, y_train)
```