

# Lecture 6: Column transformer and text features

Varada Kolhatkar

# Announcements

- Lecture recordings for the first two weeks have been made available.
- Where to find slides?
  - <https://kvarada.github.io/cpsc330-slides/lecture.html>
- HW3 is due next week Tuesday, Oct 1st, 11:59 pm.
  - You can work in pairs for this assignment.

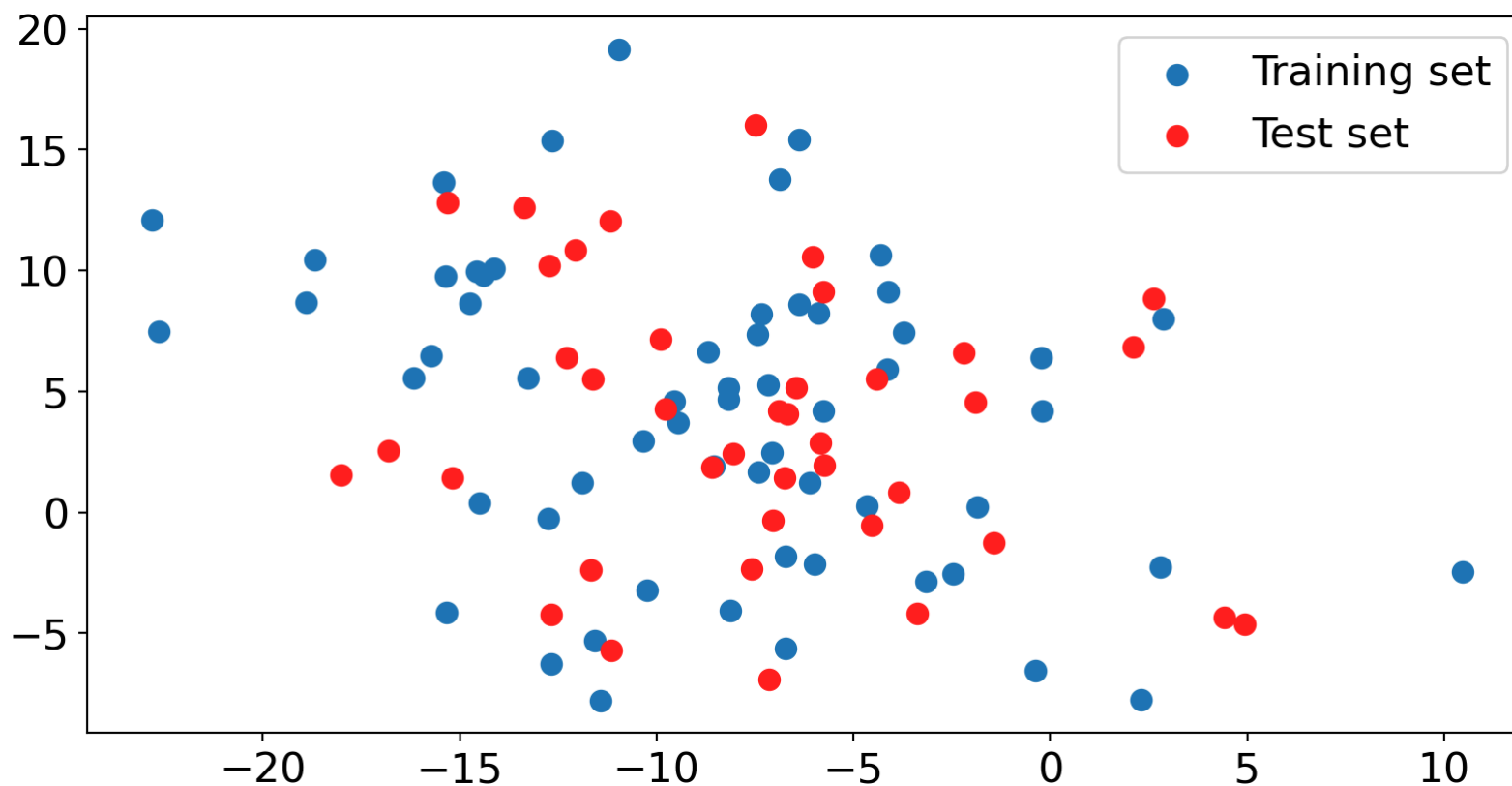
# Recap: Preprocessing mistakes

# Data

```

1 X, y = make_blobs(n_samples=100, centers=3, random_state=12, cluster_std=5) # make synthetic data
2 X_train_toy, X_test_toy, y_train_toy, y_test_toy = train_test_split(
3     X, y, random_state=5, test_size=0.4) # split it into training and test sets
4 # Visualize the training data
5 plt.scatter(X_train_toy[:, 0], X_train_toy[:, 1], label="Training set", s=60)
6 plt.scatter(
7     X_test_toy[:, 0], X_test_toy[:, 1], color=mplotlib.cm2(1), label="Test set", s=60
8 )
9 plt.legend(loc="upper right")

```



# Bad methodology 1

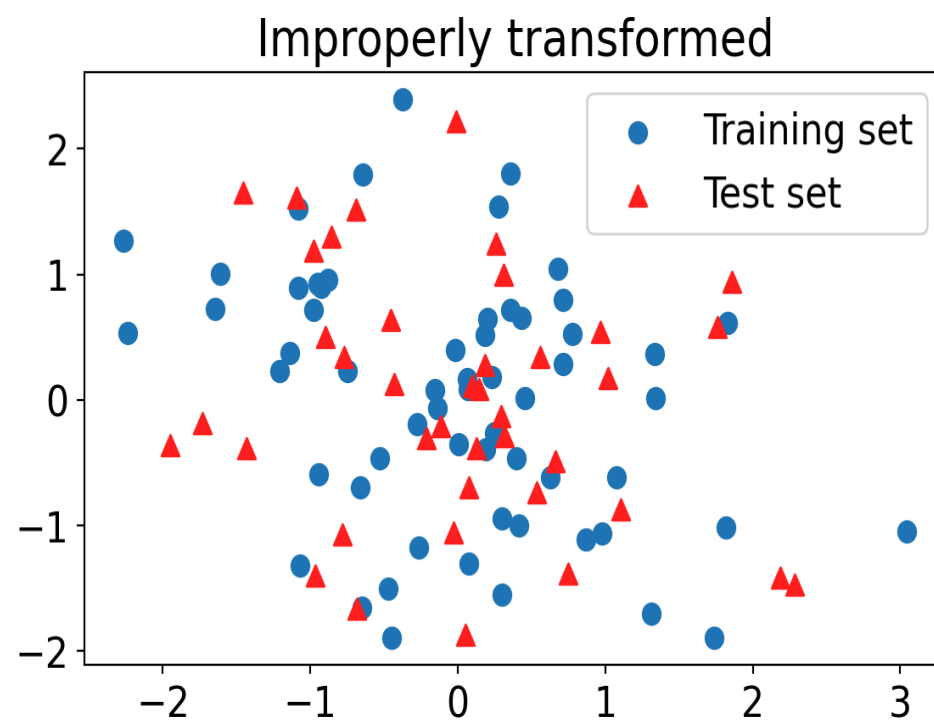
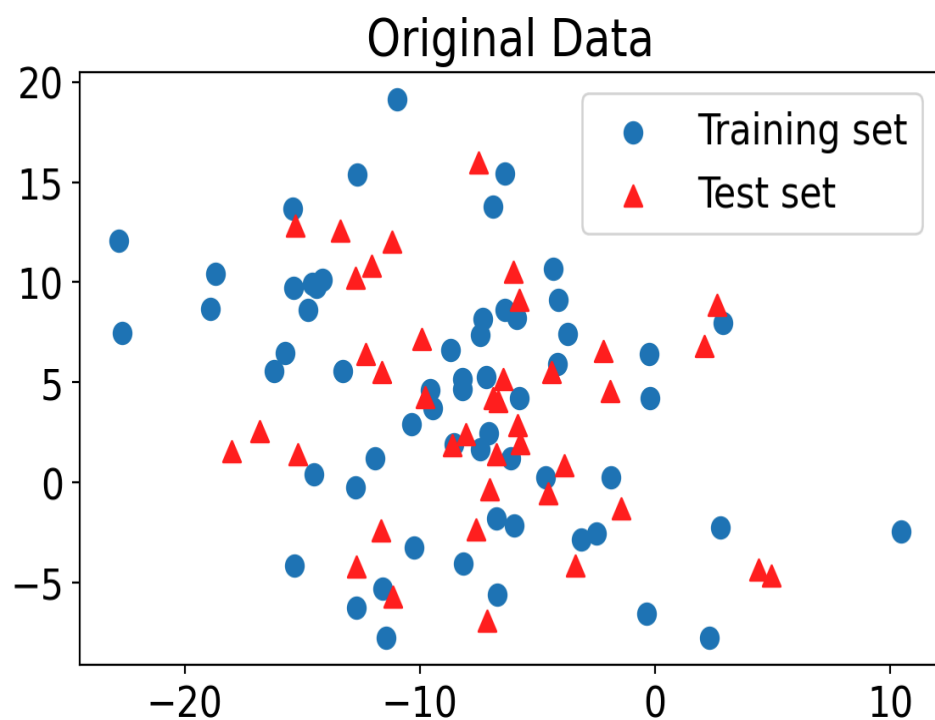
- What's wrong with scaling data separately?

```
1 scaler = StandardScaler() # Creating a scaler object
2 scaler.fit(X_train_toy) # Calling fit on the training data
3 train_scaled = scaler.transform(
4     X_train_toy
5 ) # Transforming the training data using the scaler fit on training data
6
7 scaler = StandardScaler() # Creating a separate object for scaling test data
8 scaler.fit(X_test_toy) # Calling fit on the test data
9 test_scaled = scaler.transform(
10     X_test_toy
11 ) # Transforming the test data using the scaler fit on test data
12
13 knn = KNeighborsClassifier()
14 knn.fit(train_scaled, y_train_toy)
15 print(f"Training score: {knn.score(train_scaled, y_train_toy):.2f}")
16 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 scaling the data together

```
1 # join the train and test sets back together
2 XX = np.vstack((X_train_toy, X_test_toy))
3
4 scaler = StandardScaler()
5 scaler.fit(XX)
6 XX_scaled = scaler.transform(XX)
7
8 XX_train = XX_scaled[:X_train_toy.shape[0]]
9 XX_test = XX_scaled[X_train_toy.shape[0]:]
10
11 knn = KNeighborsClassifier()
12 knn.fit(XX_train, y_train_toy)
13 print(f"Training score: {knn.score(XX_train, y_train_toy):.2f}") # Misleading score
14 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 here?

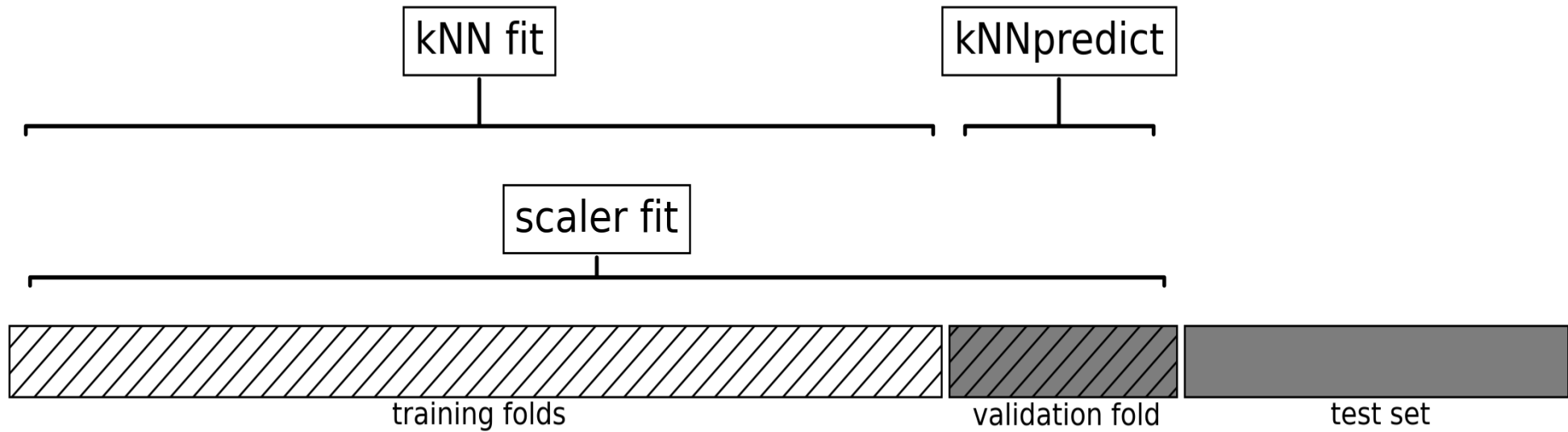
```
1 knn = KNeighborsClassifier()  
2  
3 scaler = StandardScaler()  
4 scaler.fit(X_train_toy)  
5 X_train_scaled = scaler.transform(X_train_toy)  
6 X_test_scaled = scaler.transform(X_test_toy)  
7 cross_val_score(knn, X_train_scaled, y_train_toy)
```

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



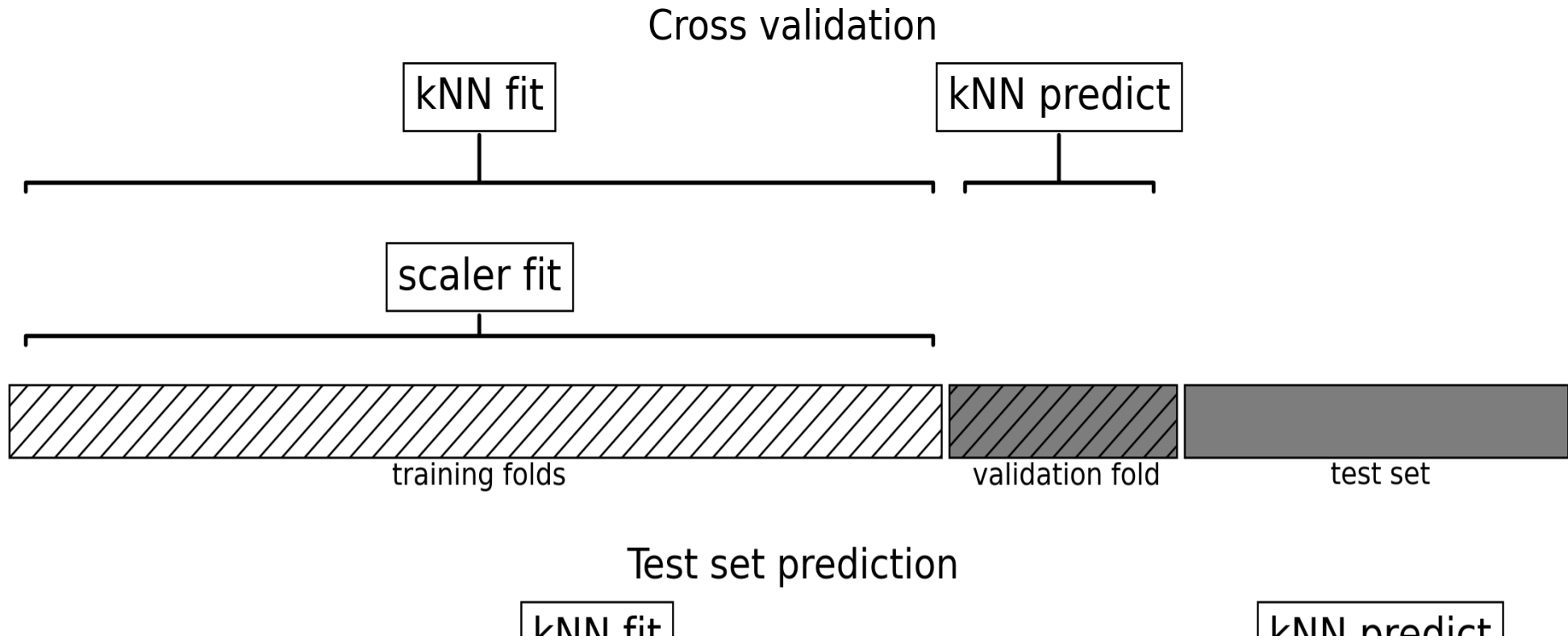
# Improper preprocessing

Cross validation





# Proper preprocessing





# Recap: **sklearn** Pipelines

- Pipeline is a way to chain multiple steps (e.g., preprocessing + model fitting) into a single workflow.
- 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.

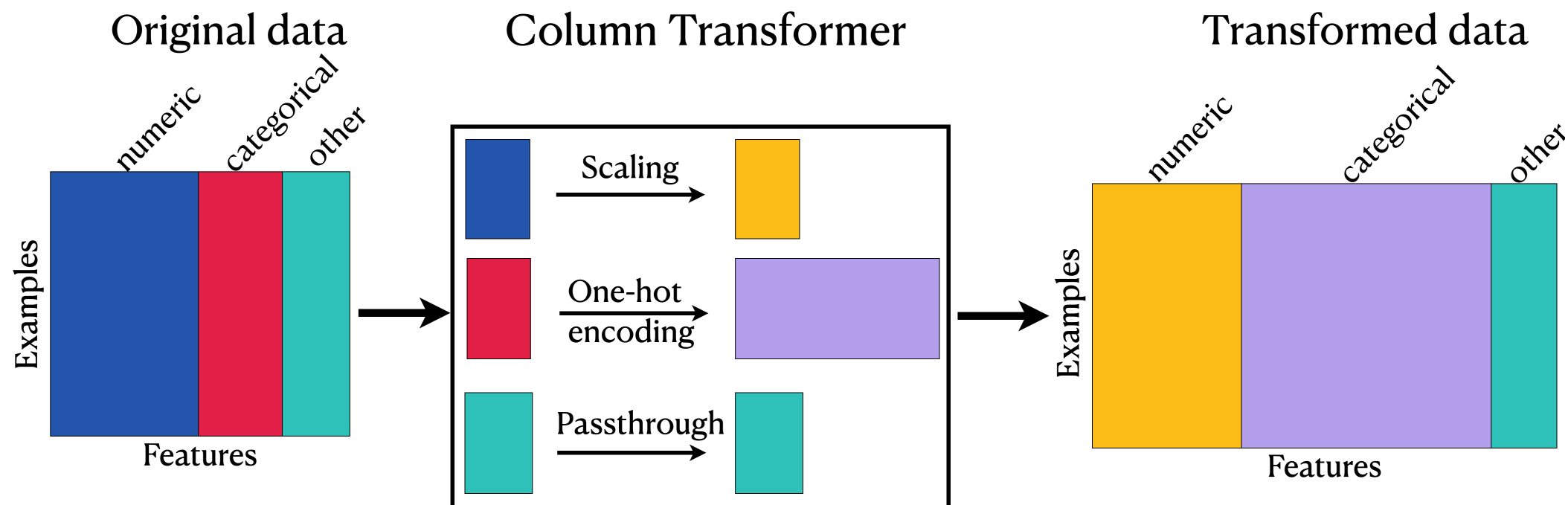
```
1 from sklearn.pipeline import make_pipeline
2
3 pipe_knn = make_pipeline(StandardScaler(), KNeighborsClassifier())
4
5 # Correct way to do cross validation without breaking the golden rule.
6 cross_val_score(pipe_knn, X_train_toy, y_train_toy)
```

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

# Class demo

# 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

iClicker cloud join link: <https://join.iclicker.com/VYFJ>

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 such as ratings per movies (power law distribution; a few movies have lots of ratings but most movies have very few ratings)
- ...

# 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.

# handle\_unknown = "ignore" of OneHotEncoder

- Use `handle_unknown='ignore'` with `OneHotEncoder` to safely ignore unseen categories during transform.
- In each of the following scenarios, identify whether it's a reasonable strategy or not.
  - Example 1: Suppose you are building a model to predict customer behavior (e.g., purchase likelihood) based on features like `location`, `device_type`, and `product_category`. During training, you have observed a set of categories for `product_category`, but in the future, new product categories might be added.
  - Example 2: You're building a model to predict disease diagnosis based on symptoms, where each symptom is categorized (e.g., fever, headache, nausea).

# 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.

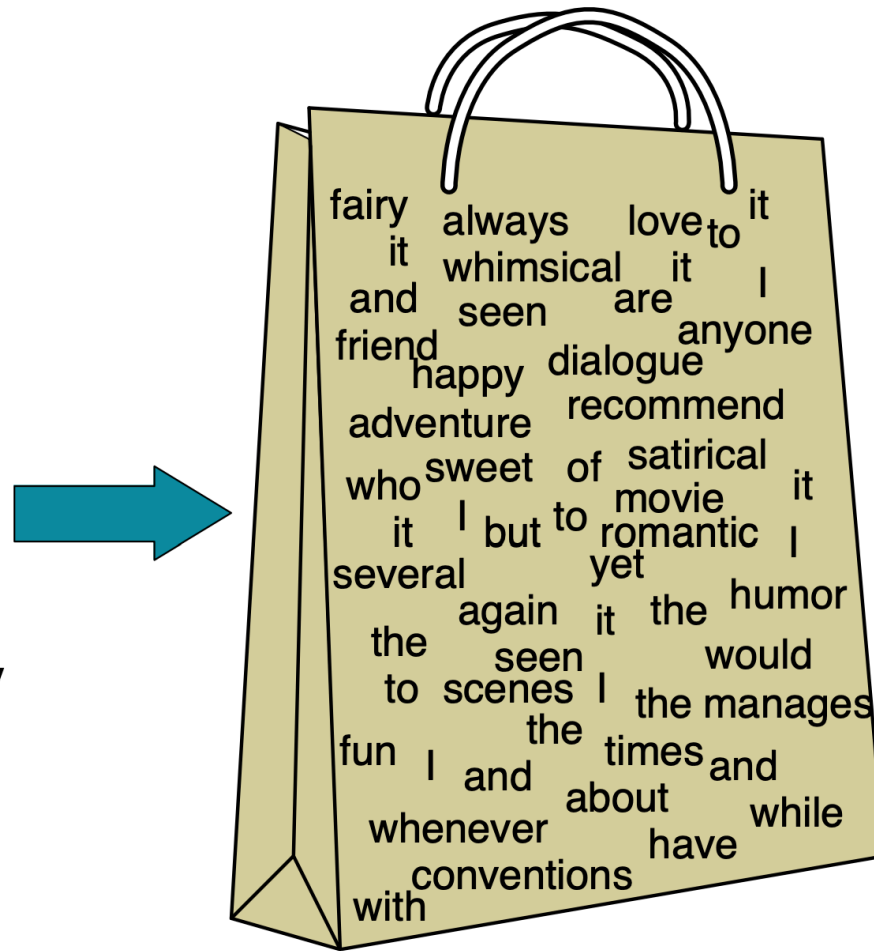
# Categorical variables with too many categories

- 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

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...



# 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 )
```

# Class demo

# (iClicker) Exercise 6.2

iClicker cloud join link: <https://join.iclicker.com/VYFJ>

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)
```