**Feature Engineering**

Feature engineering is the process of transforming raw data into meaningful features that improve the performance of machine learning models. In other words, it is a process of extracting useful features from raw data using math, statistics and domain knowledge.

Here are some common **feature engineering techniques** with examples:

### 1. Missing Value Imputation or ****Handling Missing Values****

Missing data can negatively impact model performance. Common techniques include:

* **Imputation**: Replace missing values with mean, median, mode, or a constant.
  + Example: Replace missing Age in a dataset with the median age.

df['Age'].fillna(df['Age'].median(), inplace=True)

* **Indicator Variable**: Add a binary column indicating whether the value was missing.
  + Example: Create a new column Age\_Missing:

df['Age\_Missing'] = df['Age'].isnull().astype(int)

### 2. Encoding Categorical Variables

Machine learning models require numerical inputs, so categorical variables must be encoded.

* **One-Hot Encoding**: Convert categories into binary columns.
  + Example: Convert Color (Blue, Red) into two binary columns.

|  |  |
| --- | --- |
| **Blue** | **Red** |
| 0 | 1 |
| 1 | 0 |
| 0 | 1 |

pd.get\_dummies(df, columns=['Color'])

* **Label Encoding**: Assign a unique integer to each category (useful for ordinal data).
  + Example: Convert Size (Small, Medium, Large) to 0, 1, 2.

from sklearn.preprocessing import LabelEncoder

df['Size'] = LabelEncoder().fit\_transform(df['Size'])

**4. Binning (Discretization)**

* **Description**: Converting numerical/ continuous features into categorical/ discrete bins.
* **Example**:
* Convert Age into groups (0-18, 19-35, 36-60, 60+).

bins = [0, 18, 35, 60, 100]

labels = ['Child', 'Young', 'Adult', 'Senior']

df['Age\_Group'] = pd.cut(df['Age'], bins=bins, labels=labels)

**5. Feature Scaling (Normalization/Standardization)**

Models like SVM, KNN, and neural networks perform better with scaled features.

* **Standardization (Z-score Normalization)**: Adjusting the scale of features to a standard range Scale to mean=0, std=1.
  + Example: Scale Income column.

from sklearn.preprocessing import StandardScaler

df['Income'] = StandardScaler().fit\_transform(df[['Income']])

* **Min-Max Normalization**: Scale to a range (e.g., [0, 1]).
  + Example: Normalize Age between 0 and 1.

from sklearn.preprocessing import MinMaxScaler

df['Age'] = MinMaxScaler().fit\_transform(df[['Age']])

### 6. ****Log/Polynomial Transformations****

Handle skewed data or non-linear relationships.

* **Log Transform**: Reduce skewness in data.
  + Example: Transform Income to log scale.

df['Log\_Income'] = np.log1p(df['Income'])

* **Polynomial Features**: Create interaction terms.
  + Example: Generate Age² and Age × Income.

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include\_bias=False)

df\_poly = poly.fit\_transform(df[['Age', 'Income']])

**8. Date/Time Feature Extraction**

Extract useful information from timestamps.

* Example: Extract Day\_of\_Week, Month, Hour from a Timestamp column.

df['Day\_of\_Week'] = df['Timestamp'].dt.dayofweek

df['Month'] = df['Timestamp'].dt.month

**9. Count/Frequency Encoding**

* **Description**: Replacing categories with their count/frequency.
* **Example**:
* df['City\_count'] = df['City'].map(df['City'].value\_counts())

### 10. Text Feature Extraction

Convert text data into numerical features.

* **Bag-of-Words (CountVectorizer)**:

from sklearn.feature\_extraction.text import CountVectorizer

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(df['Text'])

* **TF-IDF (Term Frequency-Inverse Document Frequency)**:

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer()

X = tfidf.fit\_transform(df['Text'])

**11. Interaction Features**

Combine features to capture relationships.

* Example: Multiply Num\_Rooms × Room\_Size to create Total\_Area.

df['Total\_Area'] = df['Num\_Rooms'] \* df['Room\_Size']

**12. Target Encoding**

Replace categories with the mean of the target variable (useful for high-cardinality categorical features).

* Example: Encode City based on average Salary.

city\_means = df.groupby('City')['Salary'].mean().to\_dict()

df['City\_Encoded'] = df['City'].map(city\_means)

**13. Dimensionality Reduction**

Reduce feature space while preserving information.

* **PCA (Principal Component Analysis)**:

from sklearn.decomposition import PCA

pca = PCA(n\_components=3)

X\_pca = pca.fit\_transform(X\_scaled)

**14. Feature Split**

Feature split refers to the process of dividing or separating features in machine learning and data science contexts. Here are the key aspects:

## Types of Feature Splits

1. **Train-Test Split**
   * Dividing data into training and testing sets (typically 70-80% train, 20-30% test)
   * Essential for evaluating model performance
2. **Train-Validation-Test Split**
   * Three-way split (e.g., 60% train, 20% validation, 20% test)
   * Validation set used for hyperparameter tuning
3. **Feature Group Splitting**
   * Separating features by type (numerical, categorical, text)
   * Different preprocessing for different feature types
4. **Time-Based Splitting**
   * For temporal data, splitting by time periods
   * Older data for training, newer for testing

## Why Perform Feature Splits?

* Prevent data leakage
* Evaluate model generalization
* Optimize model performance
* Handle different feature types appropriately

# **Feature Splitting with Practical Examples**

## **1. Basic Train-Test Split**

Most fundamental split for model evaluation

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

from sklearn.datasets import load\_iris

# Load sample data

data = load\_iris()

X, y = data.data, data.target

# Split into 70% train, 30% test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y,

test\_size=0.3,

random\_state=42,

stratify=y # Preserves class distribution

)

print(f"Train size: {len(X\_train)} samples")

print(f"Test size: {len(X\_test)} samples")

## **2. Temporal Split (Time Series)**

For time-dependent data where order matters

import pandas as pd

from sklearn.model\_selection import TimeSeriesSplit

# Create time series data

dates = pd.date\_range(start='2023-01-01', periods=100)

values = np.random.randn(100)

df = pd.DataFrame({'date': dates, 'value': values})

# Time-based splitting

train\_size = int(0.8 \* len(df))

train = df.iloc[:train\_size]

test = df.iloc[train\_size:]

# Or using TimeSeriesSplit

tscv = TimeSeriesSplit(n\_splits=5)

for train\_idx, test\_idx in tscv.split(df):

train\_fold = df.iloc[train\_idx]

test\_fold = df.iloc[test\_idx]

## **3. Feature-Type Splitting**

Separating different feature types for specialized preprocessing

import pandas as pd

# Sample dataframe with mixed features

data = {

'age': [25, 30, 35],

'income': [50000, 60000, 70000],

'gender': ['M', 'F', 'M'],

'review': ['Great!', 'Okay', 'Bad']

}

df = pd.DataFrame(data)

# Split by feature type

numerical = df[['age', 'income']]

categorical = df[['gender']]

text = df[['review']]

print("Numerical features:\n", numerical.head())

print("\nCategorical features:\n", categorical.head())

print("\nText features:\n", text.head())

## **4. Group-Based Splitting**

Splitting while keeping groups together (e.g., patient records)

from sklearn.model\_selection import GroupShuffleSplit

# Sample data with groups

X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])

y = np.array([0, 1, 0, 1])

groups = np.array(['patient1', 'patient1', 'patient2', 'patient2'])

gss = GroupShuffleSplit(n\_splits=1, test\_size=0.5, random\_state=42)

for train\_idx, test\_idx in gss.split(X, y, groups):

X\_train, X\_test = X[train\_idx], X[test\_idx]

y\_train, y\_test = y[train\_idx], y[test\_idx]

print("Train groups:", groups[train\_idx])

print("Test groups:", groups[test\_idx])

## **5. Stratified Splitting**

Maintaining class distribution in splits

from sklearn.model\_selection import StratifiedShuffleSplit

# Imbalanced dataset

X = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])

y = np.array([0, 0, 0, 1, 1, 1]) # 50-50 split

sss = StratifiedShuffleSplit(n\_splits=1, test\_size=0.33, random\_state=42)

for train\_idx, test\_idx in sss.split(X, y):

X\_train, X\_test = X[train\_idx], X[test\_idx]

y\_train, y\_test = y[train\_idx], y[test\_idx]

print("Class distribution in train:", np.bincount(y\_train))

print("Class distribution in test:", np.bincount(y\_test))

## **6. Nested Cross-Validation**

For hyperparameter tuning without data leakage

from sklearn.model\_selection import cross\_val\_score, KFold

from sklearn.linear\_model import LogisticRegression

# Inner CV (parameter tuning)

inner\_cv = KFold(n\_splits=3, shuffle=True, random\_state=42)

# Outer CV (performance evaluation)

outer\_cv = KFold(n\_splits=5, shuffle=True, random\_state=42)

model = LogisticRegression(solver='liblinear', penalty='l2')

param\_grid = {'C': [0.1, 1, 10]}

# Nested CV

nested\_score = cross\_val\_score(

GridSearchCV(model, param\_grid, cv=inner\_cv),

X=X,

y=y,

cv=outer\_cv

)

print("Nested CV scores:", nested\_score)

## **When to Use Each Split Type**

|  |  |  |
| --- | --- | --- |
| **Split Type** | **Best Use Case** | **Key Advantage** |
| **Train-Test** | Simple model evaluation | Fast and simple |
| **Time Series** | Temporal data | Preserves time order |
| **Feature-Type** | Mixed data types | Specialized preprocessing |
| **Group-Based** | Correlated samples | Keeps groups intact |
| **Stratified** | Imbalanced classes | Maintains distribution |
| **Nested CV** | Small datasets | Avoids optimistic bias |

**Pro Tip:** Always ensure your splitting strategy matches your data structure and problem requirements to avoid data leakage and biased evaluation.

### ****Summary Table****

|  |  |  |
| --- | --- | --- |
| **Technique** | **When to Use** | **Example** |
| Missing Value Imputation | When data has NaN values | Fill Age with median |
| One-Hot Encoding | Nominal categorical data | Convert Color to binary columns |
| Standardization | Features with different scales | Scale Income to mean=0, std=1 |
| Binning | Convert continuous to categorical | Group Age into ranges |
| Log Transform | Right-skewed data | Apply log(Income) |
| Feature Interaction | Capture multiplicative effects | Num\_Rooms × Room\_Size = Total\_Area |
| Target Encoding | High-cardinality categorical features | Encode City by mean Salary |

Feature engineering is crucial for model performance. The right techniques depend on:

- Data type (numerical, categorical, text).

- Problem context (regression, classification).

- Domain knowledge.

### Commonly Used Datasets for Feature Engineering Practice:

1. **Titanic Dataset (Kaggle)**
   * **Use Case**: Classification (Survival prediction)
   * **Features**: Age, Sex, Fare, Cabin, Embarked, etc.
   * **URL**: <https://www.kaggle.com/c/titanic/data>
2. **House Prices - Advanced Regression Techniques (Kaggle)**
   * **Use Case**: Regression (Predict house prices)
   * **Features**: Lot Area, Year Built, Neighborhood, SalePrice, etc.
   * **URL**: <https://www.kaggle.com/c/house-prices-advanced-regression-techniques>
3. **Adult Income Dataset (UCI ML Repository)**
   * **Use Case**: Classification (Predict income >50K)
   * **Features**: Age, Education, Occupation, Hours per week, etc.
   * **URL**: <https://archive.ics.uci.edu/ml/datasets/adult>
4. **Loan Prediction Dataset (Analytics Vidhya)**
   * **Use Case**: Classification (Loan approval)
   * **Features**: Gender, Education, Applicant Income, LoanAmount, etc.
   * **URL**: <https://datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii/>
5. **NYC Taxi Trip Duration (Kaggle)**
   * **Use Case**: Regression (Predict trip duration)
   * **Features**: Pickup datetime, passenger count, pickup/dropoff coordinates.
   * **URL**: <https://www.kaggle.com/c/nyc-taxi-trip-duration>

### ****1. Tabular Data (Structured Data)****

Used for **feature engineering** tasks (missing values, encoding, scaling, etc.):

#### **General Purpose:**

1. **Titanic Dataset**
   * **Use Case**: Binary classification (survival prediction).
   * **Features**: Age, Fare, Sex, Cabin (missing values), Embarked (categorical).
   * **Link**: [Kaggle Titanic](https://www.kaggle.com/c/titanic" \t "_blank)
2. **Boston Housing Dataset**
   * **Use Case**: Regression (house price prediction).
   * **Features**: CRIM (skewed), RM, LSTAT, etc.
   * **Link**: [Scikit-learn](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html" \t "_blank) (Note: Removed in newer versions due to ethical concerns; alternatives: *fetch\_california\_housing*).
3. **California Housing Dataset**
   * **Use Case**: Regression (median house value prediction).
   * **Features**: Median income, house age, skewed distributions.
   * **Link**: [Scikit-learn](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_california_housing.html" \t "_blank)
4. **Wine Quality Dataset**
   * **Use Case**: Classification/Regression (wine quality prediction).
   * **Features**: Alcohol, pH, sulphates (numerical + categorical).
   * **Link**: [UCI](https://archive.ics.uci.edu/ml/datasets/wine+quality)
5. **Credit Card Fraud Detection**
   * **Use Case**: Imbalanced classification (fraud detection).
   * **Features**: Transaction amounts (right-skewed), PCA-transformed variables.
   * **Link**: [Kaggle](https://www.kaggle.com/mlg-ulb/creditcardfraud" \t "_blank)

#### **Time-Series Data:**

1. **Air Passengers Dataset**
   * **Use Case**: Time-series forecasting (log transformations for skewness).
   * **Features**: Monthly passengers (right-skewed).
   * **Link**: [Kaggle](https://www.kaggle.com/datasets/rakannimer/air-passengers" \t "_blank)
2. **NASDAQ Stock Data**
   * **Use Case**: Financial feature engineering (log returns, volatility).
   * **Link**: [Yahoo Finance](https://finance.yahoo.com/)

### ****2. Image Data (Computer Vision)****

Used for **image feature engineering** (histogram equalization, normalization, etc.):

1. **MNIST**
   * **Use Case**: Digit recognition (pixel normalization).
   * **Link**: [Kaggle](https://www.kaggle.com/c/digit-recognizer" \t "_blank)
2. **CIFAR-10/100**
   * **Use Case**: Object classification (RGB pixel distributions).
   * **Link**: [Keras Datasets](https://keras.io/api/datasets/cifar10/" \t "_blank)
3. **ImageNet**
   * **Use Case**: Large-scale image classification (feature extraction).
   * **Link**: [ImageNet](http://www.image-net.org/)
4. **CelebA**
   * **Use Case**: Facial attribute recognition (histogram adjustments).
   * **Link**: [Kaggle](https://www.kaggle.com/datasets/jessicali9530/celeba-dataset" \t "_blank)

### ****3. Text Data (NLP)****

Used for **text feature engineering** (TF-IDF, word embeddings, etc.):

1. **IMDB Reviews**
   * **Use Case**: Sentiment analysis (BoW/TF-IDF).
   * **Link**: [Keras Datasets](https://keras.io/api/datasets/imdb/" \t "_blank)
2. **20 Newsgroups**
   * **Use Case**: Text classification (TF-IDF, embeddings).
   * **Link**: [Scikit-learn](https://scikit-learn.org/stable/datasets/real_world.html" \l "the-20-newsgroups-text-dataset" \t "_blank)
3. **Amazon Product Reviews**
   * **Use Case**: Sentiment/rating prediction (text + numerical features).
   * **Link**: [Kaggle](https://www.kaggle.com/datasets/kritanjalijain/amazon-reviews" \t "_blank)

### ****4. Geospatial Data****

1. **NYC Taxi Trip Duration**
   * **Use Case**: Regression (feature engineering on coordinates).
   * **Link**: [Kaggle](https://www.kaggle.com/c/nyc-taxi-trip-duration" \t "_blank)

### ****5. Audio Data****

1. **UrbanSound8K**
   * **Use Case**: Audio classification (MFCC feature extraction).
   * **Link**: [Kaggle](https://www.kaggle.com/datasets/chrisfilo/urbansound8k" \t "_blank)

### ****Summary Table****

|  |  |  |
| --- | --- | --- |
| **Dataset** | **Domain** | **Feature Engineering Tasks** |
| Titanic | Tabular | Missing values, one-hot encoding |
| California Housing | Tabular | Log transforms, scaling |
| Credit Card Fraud | Tabular | Handling imbalance, PCA features |
| MNIST | Image | Pixel normalization, flattening |
| IMDB Reviews | Text | TF-IDF, tokenization |
| UrbanSound8K | Audio | Spectrogram extraction |

### ****Where to Find These Datasets?****

* **Kaggle**: <https://www.kaggle.com/datasets>
* **UCI ML Repo**: <https://archive.ics.uci.edu/ml/>
* **Scikit-learn/Keras**: Built-in datasets.
* **Google Dataset Search**: <https://datasetsearch.research.google.com/>