ENCODING

Encoding

- Encoding in a DataFrame typically refers to converting categorical data (non-numeric values) into numerical format so that machine learning algorithms can process it.

- Categorical data classifies information into distinct groups or categories, lacking a specific numerical value. It refers to a form of information that can be stored and identified based on their names or labels.

- There are several methods for encoding categorical data, including:

  Label Encoding

  One-Hot Encoding

Label Encoding

- It is a technique that is used to convert categorical columns into numerical ones so that they can be fitted by machine learning models which only take numerical data. It is an important pre-processing step in a machine-learning project

Original Data

Team	Points
A	25
A	12
B	15
B	14
B	19
B	23
C	25
C	29

Label Encoded Data

Team	Points
0	25
0	12
1	15
1	14
1	19
1	23
2	25
2	29

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Example DataFrame
data = {'Color': ['Red', 'Green', 'Blue', 'Green', 'Red']}
df = pd.DataFrame(data)

# Initialize LabelEncoder
le = LabelEncoder()

# Apply label encoding to the 'Color' column
df['Color_Encoded'] = le.fit_transform(df['Color'])

print(df)
```

```
  Color  Color_Encoded
0   Red              2
1  Green             1
2  Blue              0
3  Green             1
4   Red              2
```

One-Hot Encoding

- It is a method for converting categorical variables into a binary format.
- It creates new binary columns (0s and 1s) for each category in the original variable.
- Each category in the original column is represented as a separate column, where a value of 1 indicates the presence of that category, and 0 indicates its absence.

Fruit	Categorical value of fruit	Price
apple	1	5
mango	2	10
apple	1	15
orange	3	20

Fruit_apple	Fruit_mango	Fruit_orange	price
1	0	0	5
0	1	0	10
1	0	0	15
0	0	1	20

To perform **One-Hot Encoding** on the column, you can use the pd. get_dummies() function from pandas

```
import pandas as pd

#create DataFrame
df = pd.DataFrame({'team': ['A', 'A', 'B', 'B', 'B', 'B', 'C', 'C'],
                   'points': [25, 12, 15, 14, 19, 23, 25, 29]})

#view DataFrame
print("Original DataFrame:")
print(df)

# Perform One-Hot Encoding on the 'team' column
df_one_hot = pd.get_dummies(df, columns=['team'])

#view updated DataFrame
print("\nUpdated DataFrame after One-Hot Encoding:")
print(df_one_hot)
```

	team	points
0	A	25
1	A	12
2	B	15
3	B	14
4	B	19
5	B	23
6	C	25
7	C	29

	points	team_A	team_B	team_C
0	25	1	0	0
1	12	1	0	0
2	15	0	1	0
3	14	0	1	0
4	19	0	1	0
5	23	0	1	0
6	25	0	0	1
7	29	0	0	1

Feature engineering

- Feature engineering is a crucial step in the data preprocessing pipeline that involves creating new features and transforming existing ones to improve model performance.

- It can also involve reducing the number of features, which is known as **dimensionality reduction**.

- To create new features from existing features in a DataFrame, you can apply mathematical operations, aggregations, or even combine different columns.

1. Mathematical Operations on Feature

You can create new features by performing arithmetic operations on existing features.

Example:

- Sum of two features
- Product of two features
- Difference between two features
- Ratio of two features

2. Combining Categorical Features

You can combine two categorical features into one by concatenating them or encoding them numerically.

Reduce dimensionality

- To reduce the dimensionality of the dataset using **Principal Component Analysis (PCA)**, we need to follow these steps:

1. **Create the dataset** with at least 3 features (as you've requested).
2. **Preprocess the data** (e.g., encode categorical features and standardize numeric features if necessary).
3. **Apply PCA** to reduce the data to a lower-dimensional space (e.g., 2 principal components).
4. **Display the transformed dataset**.

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Step 1: Create a dataset with 3 features
np.random.seed(42)  # For reproducibility

# Generate random data for 100 rows
data = {
    'feature1': np.random.rand(100),  # Random float values between 0 and 1
    'feature2': np.random.randint(1, 100, size=100),  # Random integers between 1 and 100
    'feature3': np.random.choice(['A', 'B', 'C'], size=100)  # Random categorical data
}

df = pd.DataFrame(data)
```

```
# Step 2: Preprocess the data
# Convert the categorical feature 'feature3' into numerical
values using Label Encoding
df['feature3'] = df['feature3'].map({'A': 0, 'B': 1, 'C': 2})

# Step 3: Standardize the numeric features before applying PCA
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)  # This scales the data to
have mean=0 and variance=1

# Step 4: Apply PCA to reduce the dataset to 2 principal
components
pca = PCA(n_components=2)  # Reduce to 2 dimensions
principal_components = pca.fit_transform(df_scaled)

# Step 5: Create a DataFrame with the 2 principal components
df_pca = pd.DataFrame(data=principal_components,
columns=['PC1', 'PC2'])

# Display the transformed dataset with reduced dimensions
print(df_pca.head())
```

Explanation:

1. **feature1**: Random float values between 0 and 1.
2. **feature2**: Random integers between 1 and 100.
3. **feature3**: Random categorical data chosen from the values 'A', 'B', or 'C'

	feature1	feature2	feature3
0	0.374540	52	B
1	0.950714	93	C
2	0.731994	15	B
3	0.598658	44	B
4	0.156019	31	C

	PC1	PC2
0	0.114041	-0.019530
1	1.018095	-0.277535
2	0.832225	0.019850
3	0.585945	-0.200324
4	-0.495595	0.265045