

UNIT - 2

DATA TRANSFORMATION AND RESHAPING

Techniques for data transformation

Data Transformation :- Data is the most important asset for organizations worldwide. The majority of data is raw, which makes it challenging to work with as it is difficult to understand this data directly. Therefore, it is essential to convert this data into a format that is more usable and understandable. **Data Transformation** is a technique used to transform raw data into a more appropriate format that enables efficient data mining and model building.

Normalization and Scaling

Normalization and scaling adjust the values of numerical data to a common scale.

1. Normalization (Min-Max Scaling)

What it does: Adjusts data values to fit within a specified range, usually 0 to 1.


Example:

- Original data: [10, 20, 30]
- Normalized data: [0, 0.5, 1]

Why Normalize Data?

- **Consistency:** When features have different ranges, one feature might dominate others. Normalization ensures all features contribute equally.
- **Model Performance:** Many machine learning algorithms, such as gradient descent-based models, perform better when the data is normalized, as it helps in faster convergence.
- **Comparability:** Normalization makes it easier to compare different datasets or features, especially when they originate from different sources.

python

 Copy code


```
import pandas as pd
import numpy as np

# Example dataset
df = pd.DataFrame({
    'value': [50, 100, 150, 200, 250]
})

# Applying Min-Max Normalization
df['normalized'] = (df['value'] - df['value'].min()) / (df['value'].max() - df['value'].min())

print(df)
```

OUTPUT:

 Copy code

	value	normalized
0	50	0.00
1	100	0.25
2	150	0.50
3	200	0.75
4	250	1.00

Applications of Normalization:

- **Preprocessing for Machine Learning:** Many machine learning algorithms, like k-nearest neighbors, neural networks, and support vector machines, benefit from normalized data as it improves performance and convergence.
- **Data Visualization:** Normalized data is easier to visualize, especially when comparing multiple features with different scales.

2. Standardization (Z-Score Scaling)

Standardization, also known as Z-score scaling, is a data transformation technique that adjusts the features of your data so that they have a mean of 0 and a standard deviation of 1. This technique is especially useful when your data has different units or scales, as it centers the data and scales it in a way that makes it easier to compare and analyze.

Why Use Standardization?

- **Comparability:** Ensures that features with different units or scales are on a level playing field.
- **Model Performance:** Improves the performance of machine learning models, particularly those that are sensitive to the scale of input data, like linear regression, logistic regression, and SVMs.
- **Interpretability:** Makes it easier to interpret the coefficients in models, as all features will be on the same scale.

How Z-Score Scaling Works

The Z-score of a data point indicates how many standard deviations that point is from the mean of the data. The formula for Z-score scaling is:

$$Z = \frac{X - \mu}{\sigma}$$

Where:

- X is the original value.
- μ is the mean of the data.
- σ is the standard deviation of the data.


Standardization (Z-Score Scaling)

What it does: Adjusts data so it has a mean of 0 and a standard deviation of 1.

Example:

- Original data: [10, 20, 30]
- Standardized data: [-1.22, 0, 1.22] (approx)

python

 Copy code

```
import pandas as pd
import numpy as np


# Example dataset
df = pd.DataFrame({
    'value': [10, 20, 30, 40, 50]
})

# Calculate the mean and standard deviation
mean_value = df['value'].mean()
std_dev_value = df['value'].std()

# Apply Z-Score Scaling
df['standardized'] = (df['value'] - mean_value) / std_dev_value

print(df)
```

OUTPUT:

 Copy code

```
value  standardized
0      10      -1.264911
1      20      -0.632456
2      30       0.000000
3      40       0.632456
4      50       1.264911
```

Applications of Standardization:

- **Machine Learning:** Used in algorithms like k-nearest neighbors, principal component analysis (PCA), and linear regression to improve performance.
- **Statistical Analysis:** Z-scores are commonly used in statistics to understand how far a data point is from the mean relative to the standard deviation.

Encoding

Encoding is a data transformation technique used to convert categorical data into a numerical format that can be used by machine learning models. Since most machine learning algorithms require numerical input, encoding is crucial for preparing categorical features for analysis.

Why Use Encoding?

- **Model Compatibility:** Most machine learning algorithms require numerical input, so categorical data must be converted.
- **Improved Performance:** Proper encoding can lead to better model performance by accurately representing categorical information.
- **Avoids Misinterpretation:** One-hot encoding avoids the problem of giving ordinal significance to categories, which might mislead the model.

Types of Encoding

➤ Label Encoding:

- Converts each category in a feature to a unique integer.
- Useful when the categorical data has an inherent order (ordinal data).

➤ One-Hot Encoding:

- Converts each category into a new binary column (0 or 1).
- Useful when there is no ordinal relationship between categories (nominal data).
- Prevents the model from assuming a natural order between categories.

Encoding:

Encoding transforms categorical data into numerical format.

1. One-Hot Encoding

What it does: Creates binary columns for each category.

Example:

- Original data: ["Red", "Green", "Blue"]
- One-hot encoded data: [[1, 0, 0], [0, 1, 0], [0, 0, 1]]

```
import pandas as pd

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

# Apply One-Hot Encoding
df_encoded = pd.get_dummies(df, columns=['Color'])

print(df_encoded)
```

OUTPUT:

	Color_Blue	Color_Green	Color_Red
0	0	0	1
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1


2. Label Encoding

What it does: Assigns a unique number to each category.

Example:

- Original data: ["Red", "Green", "Blue"]
- Label encoded data: [0, 1, 2]

python

 Copy code

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


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

# Apply Label Encoding
label_encoder = LabelEncoder()
df['Color_encoded'] = label_encoder.fit_transform(df['Color'])

print(df)
```

OUTPUT:

mathematica

 Copy code

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

When to Use Which Encoding?

- **Label Encoding:** Best suited for ordinal data where the categories have a meaningful order (e.g., "low", "medium", "high").
- **One-Hot Encoding:** Preferred for nominal data where the categories do not have any intrinsic order (e.g., "cat", "dog", "bird").

Reshaping Data

Reshaping data is a crucial technique in data wrangling, especially when dealing with datasets that need to be reorganized for analysis or visualization. Pivot tables are one of the most powerful tools for reshaping data, allowing you to summarize and aggregate data across multiple dimensions.

What is a Pivot Table

A pivot table is a data processing tool used to summarize, aggregate, and reorganize data in a table format. It allows you to:

- Change the structure of data from rows to columns (or vice versa).
- Aggregate data using functions like sum, mean, or count.
- Facilitate easy comparisons between different categories or groups.

Key Terms:

- **Long Format:** Each row represents an observation, and there may be multiple rows for each group or category.
- **Wide Format:** Each group/category is represented in a single row, with multiple columns representing the values of different observations.

Reshaping Data

Reshaping changes the structure of your data for easier analysis.

1. Pivot Tables

What it does: Summarizes data by key values.

Example:


- **Original data:**

Date	Category	Value
2021-01	A	10
2021-01	B	20
2021-02	A	30
2021-02	B	40

Pivot table:

Date	A	B
2021-01	10	20
2021-02	30	40

python

 Copy code

```
import pandas as pd
df = pd.DataFrame({
    'Date': ['2021-01', '2021-01', '2021-02', '2021-02'],
    'Category': ['A', 'B', 'A', 'B'],
    'Value': [10, 20, 30, 40]
})
pivot_df = df.pivot(index='Date', columns='Category', values='Value')
```

2. Stack

What it does: Converts columns into rows.

Example:

- **Original data:**

Date	A	B
2021-01	10	20
2021-02	30	40

- **Stacked data:**

Date	Category	Value
2021-01	A	10
2021-01	B	20
2021-02	A	30
2021-02	B	40

```
stacked_df = pivot_df.stack().reset_index(name='Value')
```

3. Unstack

What it does: Converts rows back into columns, reversing the stack operation.

Example:


- **Stacked data:**

Date	Category	Value
2021-01	A	10
2021-01	B	20
2021-02	A	30
2021-02	B	40

- **Un stacked :**

Date	A	B
2021-01	10	20
2021-02	30	40

python

 Copy code

```
unstacked_df = stacked_df.set_index(['Date', 'Category']).unstack()
```

These techniques help to prepare and organize data for analysis, making it easier to work with.

Extracting Features from Text

Extracting features from text is like picking out important parts of a sentence or a document that can help a computer understand and analyze the text. These "features" are pieces of information that tell the computer what the text is about or how it's structured.

Example Sentence:

"The quick brown fox jumps over the lazy dog."

How We Extract Features:

1. Counting Words (Bag of Words):

- **What You Do:** Count how often each word appears.
- **Example:** The words in the sentence are "The," "quick," "brown," "fox," "jumps," "over," "lazy," and "dog." Each word appears once.
- **Feature:** Word counts: {"The": 1, "quick": 1, "brown": 1, "fox": 1, "jumps": 1, "over": 1, "lazy": 1, "dog": 1}

2. Finding Important Words (TF-IDF):

- **What You Do:** Check how common or rare each word is in many texts, and give more importance to rare words.
- **Example:** If "fox" is rare in other texts but appears here, it's more important.
- **Feature:** Words like "fox" might get a higher score because they're not common in most texts.

3. Understanding Word Meanings (Word Embeddings):

- **What You Do:** Group words that have similar meanings.
- **Example:** If another sentence had "fast brown fox," the words "quick" and "fast" would be recognized as similar.
- **Feature:** The word "quick" might be grouped with "fast" because they have similar meanings.

4. Looking at Word Pairs (N-grams):

- **What You Do:** Look at pairs or groups of words that appear together.
- **Example:** "quick brown" and "lazy dog" are two-word pairs.
- **Feature:** Pairs like "quick brown" and "lazy dog" help capture how words are used together.

5. Tagging Parts of Speech (POS Tagging):

- **What You Do:** Identify the role of each word in the sentence (like noun, verb, adjective).
- **Example:** “The” is a determiner, “fox” is a noun, “jumps” is a verb.
- **Feature:** Tags: {“The”: determiner, “quick”: adjective, “fox”: noun, “jumps”: verb, “over”: preposition, “lazy”: adjective, “dog”: noun}

6. Finding Names and Dates (Named Entity Recognition):

- **What You Do:** Identify specific things like names of people, places, or dates.
- **Example:** If the sentence had “John” instead of “fox,” it would recognize “John” as a person’s name.
- **Feature:** If “John” were in the text, it would be tagged as a person.

7. Understanding Feelings (Sentiment Analysis):

- **What You Do:** Determine if the text is positive, negative, or neutral.
- **Example:** If the sentence was “The fox happily jumps,” it might be seen as positive.
- **Feature:** Sentiment might be neutral because there’s no strong emotion in the sentence.

8. Identifying Main Topics (Topic Modeling):

- **What You Do:** Find out what the text is mainly about.
- **Example:** If there were many sentences about animals, the topic might be “animals.”
- **Feature:** The topic could be “animals” based on words like “fox” and “dog.”

Summary:

When we extract features from the sentence “The quick brown fox jumps over the lazy dog,” we’re picking out details like word counts, the roles of words, word pairs, and overall topics. These details help the computer understand and analyze the text, just like how you might take notes to remember important parts of a story.

Extracting features from categorical data

Extracting features from categorical data involves turning categories or labels into a form that can be used for analysis or machine learning. Categorical data consists of distinct groups or categories, such as colors, types of animals, or product names.

Example:

Suppose you have data about different types of fruits:

Fruit	Color	Type
Apple	Red	Fruit
Banana	Yellow	Fruit
Carrot	Orange	Vegetable
Spinach	Green	Vegetable

How to Extract Features from Categorical Data:

1. One-Hot Encoding:

- **What It Is:** Create a new column for each category and mark the presence of each category with a 1 or 0.
- **Example:** For the "Fruit" column, you create new columns for each type (Apple, Banana, Carrot, Spinach). Each row will have a 1 in the column corresponding to its fruit and 0s elsewhere.

Result:

Apple	Banana	Carrot	Spinach
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

```
import pandas as pd

# Sample categorical data: Fruits
data = {'fruit': ['apple', 'banana', 'orange', 'apple', 'banana', 'orange']}

# Convert to DataFrame
df = pd.DataFrame(data)

# Apply One-Hot Encoding
encoded_df = pd.get_dummies(df)

print("Original Data:\n", df)
print("\nOne-Hot Encoded Data:\n", encoded_df)
```

Output:

Original Data:

	fruit
0	apple
1	banana
2	orange
3	apple
4	banana
5	orange

One-Hot Encoded Data:

	fruit_apple	fruit_banana	fruit_orange
0	1	0	0
1	0	1	0
2	0	0	1
3	1	0	0
4	0	1	0
5	0	0	1


2. Label Encoding:

- **What It Is:** Assign each category a unique number.
- **Example:** Assign numbers to colors: Red = 1, Yellow = 2, Orange = 3, Green = 4.

Result:

Fruit	Color	Label Encoded
Apple	Red	1
Banana	Yellow	2
Carrot	Orange	3
Spinach	Green	4

python

 Copy code

```
from sklearn.preprocessing import LabelEncoder

# Sample categorical data: Fruits
fruits = ['apple', 'banana', 'orange', 'apple', 'banana', 'orange']


# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the data
encoded_fruits = label_encoder.fit_transform(fruits)

print("Original Data:", fruits)
print("Label Encoded Data:", encoded_fruits)
```

Output:

less

 Copy code

```
Original Data: ['apple', 'banana', 'orange', 'apple', 'banana', 'orange']
```

```
Label Encoded Data: [0 1 2 0 1 2]
```

Summary

- **Label Encoding** gives each category a unique number. It's simple but doesn't imply any relationship between categories.
- **One-Hot Encoding** represents each category as a binary vector with a 1 for the category present and 0s for others. It's more detailed and works well when there's no inherent order among the categories.

Using these methods, we transform categorical data (like fruit names) into numerical features that machine learning models can use.

Reshaping and Pivoting

There are a number of basic operations for rearranging tabular data. These are alternatively referred to as reshape or pivot operations.

Reshaping with Hierarchical Indexing

Hierarchical indexing provides a consistent way to rearrange data in a Data Frame. There are two primary actions:

stack : This “rotates” or pivots from the columns in the data to the rows.

unstack : This pivots from the rows into the columns I’ll illustrate these operations through a series of examples.

Consider a small Data- Frame with string arrays as row and column indexes:

```
In[120]:data=pd.DataFrame(np.arange(6).reshape((2, 3)),
.....: index=pd.Index(['Ohio', 'Colorado'], name='state'),
.....: columns=pd.Index(['one', 'two', 'three'],
.....: name='number'))
```

```
In [121]: data
```

Out[121]:

Number	one	two	three
state			
Ohio	0	1	2
Colorado	3	4	5

- **Using the stack method on this data pivots the columns into the rows, producing a Series:**

```
In [122]: result = data.stack()
```

```
In [123]: result
```

Out[123]:

state	number	
Ohio	one	0
	two	1
	three	2
Colorado	one	3
	two	4
	three	5

- **From a hierarchically indexed Series, you can rearrange the data back into a Data- Frame with `unstack`:**

```
In [124]: result.unstack()
```

```
Out[124]:
```

Number	one	two	three
State			
Ohio	0	1	2
Colorado	3	4	5

- **By default the innermost level is unstacked (same with `stack`). You can unstack a different level by passing a level number or name:**

```
In [125]: result.unstack(0)
```

```
Out[125]:
```

state	Ohio	Colorado
number		
one	0	3
two	1	4
three	2	5

```
In [126]: result.unstack('state')
```

```
Out[126]:
```

State	Ohio	Colorado
number		
one	0	3
two	1	4
three	2	5

- **Unstacking might introduce missing data if all of the values in the level aren't found in each of the subgroups:**

```
In [127]: s1 = pd.Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])
```

```
In [128]: s2 = pd.Series([4, 5, 6], index=['c', 'd', 'e'])
```

```
In [129]: data2 = pd.concat([s1, s2], keys=['one', 'two'])
```

```
In [130]: data2
```

```
Out[130]:
```

One	a	0
	b	1
	c	2
	d	3
two	c	4
	d	5
	e	6

```
dtype: int64
```

```
In [131]: data2.unstack()
```

```
Out[131]:
```

	a	b	c	d	e
one	0.0	1.0	2.0	3.0	NaN
two	NaN	NaN	4.0	5.0	6.0

- **Stacking filters out missing data by default, so the operation is more easily invertible:**

```
In [132]: data2.unstack()
```

```
Out[132]:
```

	a	b	c	d	e
one	0.0	1.0	2.0	3.0	NaN
two	NaN	NaN	4.0	5.0	6.0

```
In [133]: data2.unstack().stack()
```

```
Out[133]:
```

One	a	0.0
	b	1.0
	c	2.0
	d	3.0
two	c	4.0
	d	5.0
	e	6.0

```
dtype: float64
```

Pivoting “Long” to “Wide” Format

A common way to store multiple time series in databases and CSV is in so-called *long* or *stacked* format.


Let’s load some example data and do a small amount of time series wrangling and other data cleaning:

Pivoting is a common data manipulation technique used to transform data from a “long” format (where each row represents a single observation) to a “wide” format (where each unique value in a column becomes a new column). This is particularly useful when you want to create a summary of the data.

- **Example:**

Let's say you have a dataset containing sales data for different products over several months. The data is in a long format like this:

python

 Copy code

```
import pandas as pd

# Creating a sample dataset
data = {
    'Product': ['A', 'A', 'A', 'B', 'B', 'B'],
    'Month': ['Jan', 'Feb', 'Mar', 'Jan', 'Feb', 'Mar'],
    'Sales': [100, 150, 200, 130, 170, 160]
}

df = pd.DataFrame(data)
print(df)
```

Output (Long Format):

	Product	Month	Sales
0	A	Jan	100
1	A	Feb	150
2	A	Mar	200
3	B	Jan	130
4	B	Feb	170
5	B	Mar	160

Here, each row represents a single observation for a product in a specific month.

Pivoting to Wide Format

To transform this data into a wide format where each month becomes a separate column, you can use the `pivot()` function in pandas.

```
python                                                                    Copy code

# Pivoting the DataFrame to wide format
df_wide = df.pivot(index='Product', columns='Month', values='Sales')
print(df_wide)
```

Output (Wide Format):

CSSCopy code

Month	Jan	Feb	Mar
Product			
A	100	150	200
B	130	170	160