**UNIT-2-PYTHON FOR MACHINE LEARNING**

**INTRODUCTION TO PANDAS DATA STRUCTURES**

Pandas is a popular Python library widely used for data manipulation and analysis. It provides powerful data structures and tools for working with structured data. The two primary data structures in Pandas are:

* Series
* Dataframes
* Panels

**Series**

A Series in Pandas is a one-dimensional labeled array that can hold data of any type, such as integers, floats, strings or Python objects. It is similar to a one-dimensional NumPy array but with additional indexing capabilities, making it more versatile and powerful for data manipulation and analysis.

Key features of a Series include:

1. Indexing:

- Each element in a Series has a corresponding index label, which can be explicitly defined or automatically generated. The index provides a unique identifier for each element, allowing for efficient access, manipulation and alignment of data.

2. Flexibility:

- A Series can hold data of any type, including numerical data, strings, datetime objects and more. This flexibility allows for the representation of diverse types of data within a single Series object.

3. Vectorized Operations:

- Series support vectorized operations, enabling efficient element-wise computations and transformations without the need for explicit looping over elements. This makes it easy to perform arithmetic operations, boolean comparisons and mathematical functions on entire Series at once.

4. Alignment:

- When performing operations between multiple Series objects, Pandas automatically aligns the data based on the index labels, ensuring that computations are performed element-wise on corresponding elements. This alignment feature simplifies data manipulation and prevents errors due to mismatched indices.

5. Handling Missing Values:

- Series can handle missing or undefined values, represented as NaN (Not a Number). Pandas provides methods for detecting, removing and replacing missing values, enabling robust handling of incomplete data.

6. Custom Indexing:

- Index labels in a Series can be customized to provide meaningful identifiers for each element. This allows for intuitive and efficient data access, particularly when working with labeled or hierarchical data.

Example:

import pandas as pd

# Creating a Series from a list

data = [1, 2, 3, 4, 5]

s = pd.Series(data)

print(s)

Output:

0 1

1 2

2 3

3 4

4 5

**Data frames**

A DataFrame in Pandas is a two-dimensional labeled data structure, resembling a table or spreadsheet with rows and columns. It is a powerful tool for data manipulation and analysis, allowing users to handle structured, semi-structured and even unstructured data efficiently.

Key features of a DataFrame include:

1. Tabular Structure:

- A DataFrame represents tabular data organized in rows and columns, similar to a relational database table or an Excel spreadsheet. Each row corresponds to a separate observation or record, while each column represents a distinct variable or attribute.

2. Indexing and Labeling:

- Like Series, DataFrames have both row and column index labels, enabling fast and flexible data access, slicing and manipulation. Index labels can be explicitly defined or automatically generated by Pandas.

3. Column-Wise and Row-Wise Operations:

- DataFrames support column-wise and row-wise operations, allowing users to perform calculations, transformations and filtering on entire columns or rows simultaneously. This facilitates efficient data manipulation and analysis.

4. Heterogeneous Data Types:

- DataFrames can store data of different types (e.g., integers, floats, strings, datetime objects) within the same structure. This flexibility accommodates diverse types of data commonly encountered in real-world datasets.

5. Missing Data Handling:

- DataFrames provide robust support for handling missing or undefined values (NaN), allowing users to detect, remove or replace missing values easily. This ensures data integrity and prevents errors during analysis.

6. Alignment and Broadcasting:

- Similar to Series, DataFrames support automatic alignment of data based on index labels when performing operations between multiple DataFrames or Series objects. This alignment simplifies data manipulation and ensures consistency across different datasets.

7. Merging and Joining:

- DataFrames offer powerful methods for merging, joining and concatenating multiple datasets based on common keys or indices. This facilitates data integration and enables the combination of information from different sources.

8. Grouping and Aggregation:

- DataFrames support grouping and aggregation operations, allowing users to group data based on one or more variables and compute summary statistics (e.g., mean, sum, count) for each group. This facilitates exploratory data analysis and reporting.

Example:

import pandas as pd

# Creating a DataFrame from a dictionary

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'Age': [25, 30, 35, 40],

'Salary': [50000, 60000, 70000, 80000]}

df = pd.DataFrame(data)

print(df)

Output:

Name Age Salary

0 Alice 25 50000

1 Bob 30 60000

2 Charlie 35 70000

3 David 40 80000

In this example, a DataFrame is created from a dictionary `data`, where keys represent column names and values represent column data. Each column in the DataFrame corresponds to a key-value pair in the dictionary and each row represents a separate observation. The data types of the columns are inferred from the input data.

**Panels:**

In older versions of Pandas, the `Panel` data structure was available, which represented three-dimensional data. However, as of Pandas version 1.0.0, the `Panel` has been deprecated due to its limited use and complexity compared to the more versatile `DataFrame` data structure. A `Panel` could be thought of as a three-dimensional analog of a `DataFrame`, where data is organized into items, major axes and minor axes:

- Items: The first dimension, similar to columns in a `DataFrame`.

- Major Axis: The second dimension, similar to rows in a `DataFrame`.

- Minor Axis: The third dimension.

While `Panel` offered functionality for working with three-dimensional data, it was less intuitive and less frequently used compared to `DataFrame`. Therefore, in the interest of simplifying the library and focusing on more commonly used data structures, the decision was made to deprecate `Panel` in favor of using `DataFrame` with hierarchical indexing (MultiIndex) or reshaping techniques to represent three-dimensional data.

For example, instead of using a `Panel`, you can represent three-dimensional data using a `DataFrame` with a MultiIndex:

Example:

import pandas as pd

# Create a DataFrame with MultiIndex

index = pd.MultiIndex.from\_product([['Item1', 'Item2'], ['A', 'B']], names=['Item', 'Subitem'])

columns = ['X', 'Y', 'Z']

data = [[[1, 2, 3], [4, 5, 6]],

[[7, 8, 9], [10, 11, 12]]]

df = pd.DataFrame(data, index=index, columns=columns)

print(df)

Output:

Item Subitem

X Y Z

Item1 A 1 2 3

B 4 5 6

Item2 A 7 8 9

B 10 11 12

In this example, the `DataFrame` `df` has a MultiIndex consisting of two levels (`Item` and `Subitem`), representing the three-dimensional structure of the data. Each cell in the `DataFrame` corresponds to a value in the three-dimensional space. By using `DataFrame` with hierarchical indexing or reshaping techniques, you can achieve similar functionality to `Panel` while leveraging the more commonly used and versatile `DataFrame` data structure.

**FUNCTION APPLICATION AND MAPPING**

Certainly! In Pandas, the `apply()`, `map()` and `applymap()` functions are essential tools for transforming data within DataFrames and Series. Each function serves a distinct purpose and offers flexibility for different types of operations:

1. apply():

- The `apply()` function is primarily used to apply a function along an axis of a DataFrame or Series. It allows you to perform custom operations on rows or columns of a DataFrame or on elements of a Series.

- When applied to a DataFrame:

- By default, `apply()` operates column-wise, where you pass a function that works on each column independently.

- You can also specify `axis=1` to apply the function row-wise.

- When applied to a Series:

- It applies the function element-wise to each element in the Series.

- Example with DataFrame:

import pandas as pd

# Create a DataFrame

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

# Define a function to square each value

def square(x):

return x\*2

# Apply the function element-wise to each column

result = df.apply(square)

print(result)

2. map():

- The `map()` function is used to substitute each value in a Series with another value. It works element-wise on a Series and is often used to map values from one set to another, such as replacing categories with labels.

- It takes a dictionary, a function or another Series as input, where each value in the Series is replaced according to the provided mapping.

Example

import pandas as pd

# Create a Series

s = pd.Series(['A', 'B', 'C', 'D'])

# Define a mapping dictionary

mapping = {'A': 1, 'B': 2, 'C': 3, 'D': 4}

# Map values using the mapping dictionary

result = s.map(mapping)

print(result)

3. applymap():

- The `applymap()` function is specifically designed for element-wise operations on a DataFrame. It applies a function to every element of the DataFrame.

- Unlike `apply()`, which works on columns or rows, `applymap()` applies the function to each element independently, making it ideal for element-wise transformations.

- Example:

import pandas as pd

# Create a DataFrame

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

# Define a function to square each value

def square(x):

return pow(x,2)

# Apply the function element-wise to each element of the DataFrame

result = df.applymap(square)

print(result)

These functions provide powerful tools for data manipulation and transformation in Pandas, allowing users to perform a wide range of operations efficiently on both DataFrames and Series. Whether you need to apply a function row-wise, column-wise or element-wise, Pandas has a function to suit every purpose.

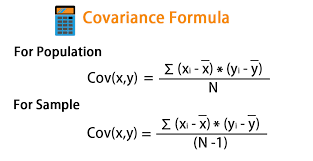
**COVARIANCE AND CORRELATION**

Correlation and covariance are two statistical measures used to describe the relationship between two variables in a dataset, particularly in the context of multivariate data analysis. While both measures assess the degree to which two variables change together, they differ in their interpretation and scale.

**1. Covariance:**

- Covariance measures the degree to which two variables vary together. It indicates the direction of the linear relationship between the variables (positive, negative or zero) and the magnitude of their joint variability.

- Mathematically, covariance between two variables X and Y is calculated as the average of the product of the deviations of each variable from their respective means:



Where,

x̄ = sample mean of x

ȳ = sample mean of y

x\_i and y\_i = the values of x and y for ith record in the sample.

N = is the no of records in the sample

- Covariance can take any value, positive, negative or zero, depending on the direction and strength of the relationship between the variables.

- However, the magnitude of covariance depends on the scale of the variables, making it difficult to compare covariances across different datasets.

**2. Correlation:**

Correlation is a standardized measure of the strength and direction of the linear relationship between two variables. It is derived from covariance and ranges between -1 and 1. Unlike covariance, which only indicates the direction of the relationship, correlation provides a standardized measure.

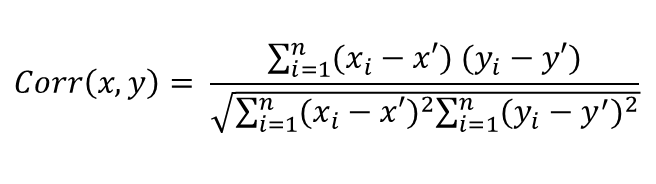
**Positive Correlation (close to +1)**: As one variable increases, the other variable also tends to increase.

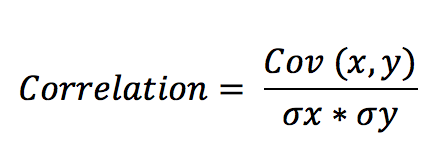
**Negative Correlation (close to -1)**: As one variable increases, the other variable tends to decrease.

**Zero Correlation:** There is no linear relationship between the variables.

The correlation coefficient ρ\rhoρ (rho) for variables X and Y is defined as:

* It show whether and how strongly pairs of variables are related to each other.
* Correlation takes values between -1 to +1, wherein values close to +1 represents strong positive correlation and values close to -1 represents strong negative correlation.
* In this variable are indirectly related to each other.
* It gives the direction and strength of relationship between variables.

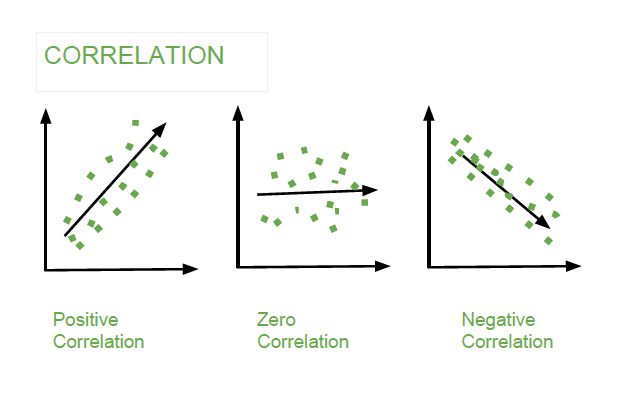




*Where,*

* Correlation = sample correlation between X and Y
* Cov(X,Y) = sample covariance between X and Y
* = sample standard deviation of X
* = sample standard deviation of Y

​ **Example –**



**Pearson’s correlation of coefficient**

Pearson’s correlation coefficient is a measure of the strength of a linear association between two variables and is denoted by r. Basically, a Pearson’s correlation attempts to draw a line of best fit through two variables' data. The Pearson correlation coefficient, r, indicates how far away all these data points are to this line of best fit.

In Pearson’s correlation coefficient, variables can be measured in entirely different units. For example, we can correlate the height of a person with his weight. It is designed in such a way that unit of measurement can’t affect the study of covariation.

Pearson’s correlation coefficient(r) is a unitless measure of correlation and doesn’t change in the effect of origin or scale shift measurement.

It doesn’t take into consideration whether a variable has been classified as a dependent or independent variable. It treats all variables equally. We might want to find out whether basketball performance is correlated to a person’s height. But if we determine whether a person’s height was determined by their basketball performance (which makes no sense), the result will be the same.

where (xi, yi) are the variables

Pearson’s correlation coefficient formula

where xi, yi, are the variables and xbar, ybar, are the mean, respectively

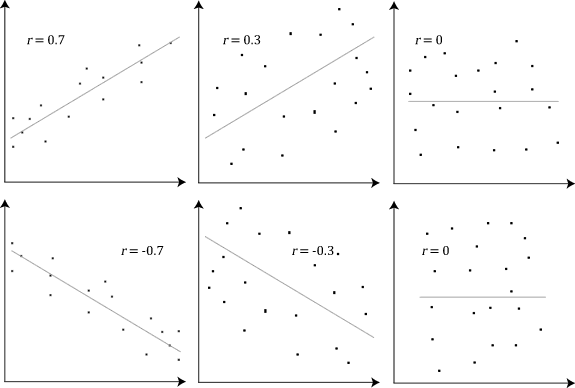
Properties:

The range of r is between [-1,1].

The computation of r is independent of the change of origin and scale of measurement.

r = 1 (perfectly positive correlation), r =-1 (perfectly negative correction)

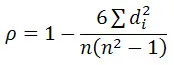
r = 0 (no correlation)



r with a linear relationship plot

**2. Spearman’s correlation of coefficient**

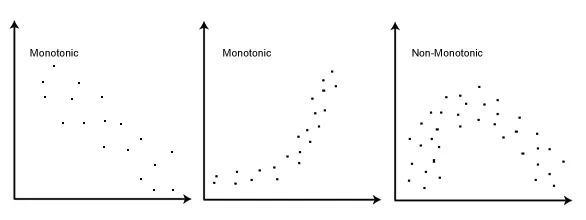
Spearman’s correlation coefficient is a non-parametric measure of the strength and direction of association that exists between two variables measured on at least an ordinal scale. The symbol rs or ρ denotes it. Such as:We may like to find out the correlation between ranks given by two Judges to candidates in an interview, marks secured by a group of students in five subjects, etc.



Spearman’s correlation coefficient formula

where n = total number of observations, di = (xi-Yi) where xi and yi are the observations

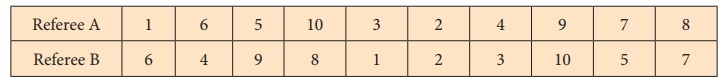
Spearman’s correlation determines the monotonic relationship's strength and direction between two variables rather than the strength and direction of the linear relationship between your two variables, which is what Pearson’s correlation determines.



ρ with monotonic plot

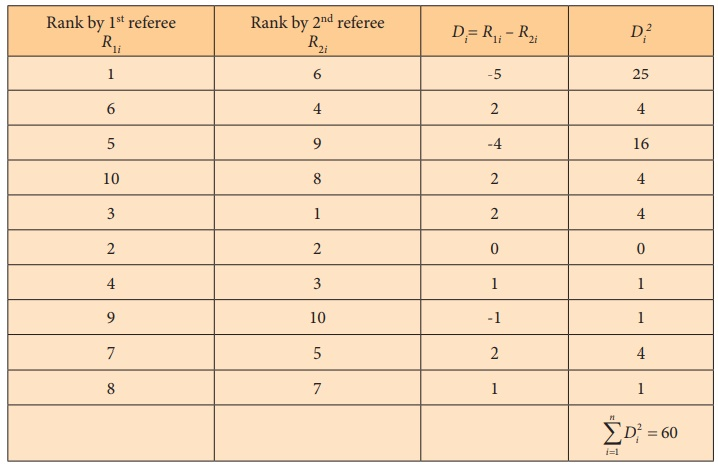
**Example:**

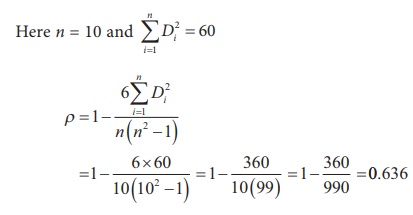
Two referees in a flower beauty competition rank the 10 types of flowers as follows:



Use the rank correlation coefficient and find out what degree of agreement is between the referees.

**Solution:**

****



**Interpretation***:*Degree of agreement between the referees ‘A’ and ‘B’ is 0.636 and they have  “strongagreement” in evaluating the competitors.

**HANDLING MISSING DATA**

Handling missing data is crucial in machine learning to ensure accurate model training and reliable predictions. Here are some common techniques to handle missing data in machine learning:

1. Deletion:

- Listwise Deletion (Complete Case Analysis): Involves removing entire rows with missing values. While simple, it can lead to loss of valuable information, especially if the missing data is not random.

2. Imputation:

- Mean/Median/Mode Imputation: Replace missing values with the mean, median or mode of the feature. This is a simple approach but may not preserve the distribution of the data.

- Predictive Imputation: Use machine learning algorithms (e.g., KNN, decision trees) to predict missing values based on other features. This method can capture complex relationships but may introduce bias if the model is overfitted.

- Interpolation: Estimate missing values by interpolating between neighbouring values. This is particularly useful for time series data or ordered data.

3. Extension Methods:

- Forward Fill (ffill) / Backward Fill (bfill): Propagate the last known value forward or backward to fill missing values in sequential data.

- Linear Regression: Use linear regression to predict missing values based on other features in the dataset. This method assumes a linear relationship between variables.

4. Algorithm-Specific Handling:

- Some machine learning algorithms can handle missing data inherently. For example, tree-based algorithms (e.g., decision trees, random forests) can handle missing values by treating them as a separate category or by using surrogate splits.

5. Data Augmentation:

- Generate synthetic data to replace missing values. This can be done through techniques like bootstrapping, multiple imputation or generative adversarial networks (GANs).

6. Feature Engineering:

- Create new features to encode missing information. For example, you can add binary indicators to flag missing values in specific features.

7. Domain-Specific Handling:

- Consider domain-specific knowledge to inform missing data handling. For instance, in medical data, missing values may carry specific clinical meanings that need to be addressed appropriately.

When choosing a technique for handling missing data, it's essential to consider the nature of the data, the underlying mechanism of missing and the potential impact on the downstream analysis or model performance. Additionally, it's good practice to assess the effectiveness of different techniques through cross-validation or other validation strategies.

**READING AND WRITING DATA INTO CSV FILES**

A CSV (Comma-Separated Values) file is a plain text file format used to store tabular data. It consists of rows and columns, where each row represents a record and each column represents a field or attribute. CSV files are commonly used for storing and exchanging structured data between different software systems, databases and spread sheet applications.

In a CSV file, each field in a row is separated by a delimiter, typically a comma (,), although other delimiters like semicolons (;) or tabs (\t) are also sometimes used. The first row of a CSV file often contains column headers, which describe the contents of each column.

Advantages of CSV Files:

Simplicity: CSV files have a simple structure, making them easy to create, read and edit.

Compatibility: CSV is universally supported by various software and platforms, ensuring seamless data exchange.

Flexibility: CSV files can handle tabular data of variable size and structure, supporting different data types.

Efficiency: CSV files have a compact size, making them efficient for storing and transferring data.

Interoperability: CSV facilitates integration between different systems and databases.

Ease of Use: CSV files are user-friendly and can be manipulated using basic tools.

Portability: CSV files are platform-independent and can be transferred across different operating systems.

Open Standard: CSV is an open standard, promoting transparency and accessibility in data management.

Reading and writing data to CSV files is a common task in data manipulation and analysis. Pandas provides simple and powerful functions for reading and writing CSV files.

Reading Data from CSV Files:

To read data from a CSV file into a DataFrame, you can use the `pd.read\_csv()` function:

import pandas as pd

# Read data from a CSV file into a DataFrame

df = pd.read\_csv('data.csv')

By default, `read\_csv()` assumes that the first row of the CSV file contains column names. You can specify additional parameters such as `header=None` if the file doesn't have a header row or `index\_col` to set one of the columns as the index.

Writing Data to CSV Files:

To write data from a DataFrame to a CSV file, you can use the `to\_csv()` method:

# Write data from a DataFrame to a CSV file

df.to\_csv('output.csv', index=False)

The `index=False` parameter specifies that the DataFrame index should not be written to the CSV file. You can set it to `True` if you want to include the index.

Both `read\_csv()` and `to\_csv()` functions support a variety of parameters to customize the behavior according to your needs. Some common parameters include:

- Delimiter (`sep`): Specifies the character used to separate fields in the CSV file. By default, it's a comma (`,`).

- Encoding (`encoding`): Specifies the encoding of the CSV file. Common encodings are `'utf-8'`, `'latin1'` and `'cp1252'`.

- Handling Missing Values (`na\_values`, `na\_rep`): Specifies how missing values are represented in the CSV file and how they should be handled when reading.

- Date Parsing (`parse\_dates`): Specifies columns to be parsed as dates.

- Chunksize (`chunksize`): Specifies the number of rows to read or write at a time. Useful for processing large files in chunks.

Example:

# Read only the first 10 rows of a CSV file

df\_chunk = pd.read\_csv('data.csv', nrows=10)

# Write data to a CSV file with a custom delimiter and encoding

df\_chunk.to\_csv('output.csv', sep='|', encoding='utf-8', index=False)

**DATA PREPARATION-MERGING AND REMOVING DATA**

In machine learning, data preparation, including merging and removing data, is crucial for ensuring that the dataset is suitable for training models effectively. Here's how these processes are applied in the context of machine learning:

1. Merging Data:

- Merging data involves combining multiple datasets to create a comprehensive dataset for model training.

- This is often done when the information needed for training a model is spread across multiple sources or tables.

- Merging can be based on a common key or index present in both datasets.

- For example, if you have one dataset containing customer information and another containing purchase history, you might merge them based on a common customer ID to create a dataset for customer segmentation or churn prediction.

2. Removing Data:

- Removing data involves eliminating irrelevant, redundant or noisy observations or features from the dataset.

- In machine learning, irrelevant or redundant features can negatively impact model performance and increase computational complexity.

- Removing data can also involve handling missing values by imputation or deletion, depending on the impact of missing data on the analysis.

- Additionally, removing outliers, which are data points significantly different from the rest of the dataset, can improve model accuracy and generalization.

- Care must be taken when removing data to avoid unintentional bias or loss of important information.

**DATA TRANSFORMATION – REMOVING DUPLICATES, MAPPING**

In machine learning, data transformation and removing duplicates are essential steps in preparing datasets for analysis and model training. Here's a brief overview of these processes:

1. Data Transformation:

- Data transformation involves converting the raw data into a format that is suitable for analysis and model training.

- Common data transformation techniques include:

- Scaling: Standardizing or normalizing numerical features to ensure that they have similar scales and distributions.

- Encoding: Converting categorical variables into numerical representations suitable for modeling (e.g., one-hot encoding, label encoding).

- Feature engineering: Creating new features from existing ones or transforming features to better capture patterns in the data.

- Dimensionality reduction: Reducing the number of features in the dataset while preserving important information using techniques like Principal Component Analysis (PCA) or feature selection methods.

- Data transformation aims to improve the quality of the dataset, reduce noise and enhance the performance of machine learning models.

2. Removing Duplicates:

- Removing duplicates involves identifying and eliminating duplicate observations from the dataset.

- Duplicates can arise due to data entry errors, data merging or other reasons and they can adversely affect model training and analysis.

- Removing duplicates ensures that each observation in the dataset is unique and avoids biasing the model towards repeated instances.

- Duplicate removal can be performed based on specific columns or the entire dataset, depending on the requirements of the analysis.

- After removing duplicates, it's important to check for any remaining inconsistencies or errors in the dataset to ensure data quality.

Data transformation and removing duplicates are important preprocessing steps in machine learning that help prepare datasets for analysis and model training. These steps contribute to improving the quality, consistency and reliability of the data, ultimately leading to more accurate and robust machine learning models.

3. Mapping

In machine learning, mapping refers to the process of transforming input data into a different representation that is more suitable for the learning algorithm or the problem at hand. Mapping can involve various techniques, each serving a different purpose in the data preprocessing pipeline. Here are some common types of mapping in machine learning:

1. Feature Mapping:

- Feature mapping involves transforming the original features of the dataset into a new set of features that better represent the underlying relationships in the data.

- This can include creating new features through feature engineering, such as polynomial features, interactions or transformations of existing features.

- Feature mapping aims to capture nonlinear relationships and improve the performance of machine learning models by providing them with richer input representations.

2. Label Mapping:

- Label mapping is the process of transforming categorical labels or target variables into numerical representations that can be processed by machine learning algorithms.

- This may involve techniques such as one-hot encoding, label encoding or ordinal encoding, depending on the nature of the labels and the requirements of the model.

- Label mapping ensures that the target variable is in a format that is compatible with the chosen machine learning algorithm and objective function.

3. Data Mapping:

- Data mapping refers to transforming the input data from its original format into a representation that is suitable for the specific requirements of the learning algorithm.

- This could involve preprocessing steps such as scaling, normalization or standardization to ensure that the input data has a consistent scale and distribution.

- Data mapping aims to improve the convergence and stability of machine learning algorithms by preparing the input data in a way that facilitates effective learning.

4. Output Mapping:

- Output mapping involves transforming the predictions or output of a machine learning model into a format that is interpretable or actionable for the intended application.

- This may include post-processing steps such as thresholding, scaling or converting the predicted probabilities into discrete class labels.

- Output mapping ensures that the output of the model is in a format that can be easily understood or utilized by stakeholders or downstream systems.