import pandas as pd

import numpy as np

df=pd.read\_csv('data1 (1).csv')

df

df.dtypes

df.index

df.info()

df.shape

df.size

df.describe()

df.head()

df.tail()

df.tail(5)

df.notnull()

df.isnull()

df.columns

df.columns.values

df.iloc[5]

df.iloc[:,:]

df.iloc[:,3]

df.iloc[:8,:2]

df.iloc[1,1]

df.iloc[3:5,0:2]

df.iloc[5]

print(df.columns.tolist())

df['Sepal Length']

df['Sepal Width']

df['Petal Length']

df['Petal Width']

df['Species']

from sklearn import preprocessing

min\_max\_scaler = preprocessing.MinMaxScaler()

x=df.iloc[:,:4]

x\_scaled = min\_max\_scaler.fit\_transform(x)

df\_normalized = pd.DataFrame(x\_scaled)

df\_normalized

from sklearn import preprocessing

label\_encoder = preprocessing.LabelEncoder()

df['Species'] = label\_encoder.fit\_transform(df['Species'])

print(df['Species'].unique())

df['Sepal Length'] = df['Sepal Length'].astype(float)

df['Species'] = df['Species'].astype('category')

print(df.dtypes)

1. Importing Libraries

import pandas as pd

import numpy as np

import pandas as pd: Imports the pandas library, which is used for data manipulation and analysis.

import numpy as np: Imports the numpy library, which is used for numerical operations, particularly arrays and matrices.

2. Loading the Dataset

df = pd.read\_csv('C:/Users/SHRAVANI/Downloads/data1 (1).csv')

pd.read\_csv('...'): Reads a CSV file from the specified file path ('C:/Users/SHRAVANI/Downloads/data1 (1).csv') and loads it into a pandas DataFrame df.

3. Display the DataFrame

df

This line simply displays the df DataFrame. It's often used interactively in Jupyter Notebooks, where it shows the first few rows of the dataset.

4. Checking Data Types of Columns

df.dtypes

df.dtypes: Displays the data types of each column in the DataFrame (df). This helps in identifying whether the columns are correctly typed (e.g., numeric, categorical, etc.).

5. Getting DataFrame Index

df.index

df.index: Returns the index of the DataFrame, which indicates the row labels (default is a range of integers starting from 0).

6. Getting DataFrame Information

df.info()

df.info(): Provides a concise summary of the DataFrame, including the column names, non-null counts, data types, and memory usage.

7. Getting Shape, Size, and Summary Statistics

df.shape

df.size

df.describe()

df.shape: Returns a tuple representing the dimensions of the DataFrame (rows, columns).

df.size: Returns the total number of elements in the DataFrame (i.e., rows \* columns).

df.describe(): Provides summary statistics of the numeric columns in the DataFrame, such as count, mean, standard deviation, min, max, and quartiles.

8. Viewing Top and Bottom Rows

df.head()

df.tail()

df.tail(5)

df.head(): Displays the first 5 rows of the DataFrame.

df.tail(): Displays the last 5 rows of the DataFrame.

df.tail(5): Displays the last 5 rows explicitly, though df.tail() by default also shows the last 5 rows.

9. Checking for Missing Values

df.notnull()

df.isnull()

df.notnull(): Returns a DataFrame of the same shape as df, with True for non-null values and False for null values.

df.isnull(): Returns a DataFrame of the same shape as df, with True for null values and False for non-null values.

10. Inspecting Column Names

df.columns

df.columns.values

df.columns: Displays the column names in the DataFrame.

df.columns.values: Returns the column names as a numpy array.

11. Indexing and Slicing Data

df.iloc[5]

df.iloc[:,:]

df.iloc[:,3]

df.iloc[:8,:2]

df.iloc[1,1]

df.iloc[3:5,0:2]

df.iloc[5]

df.iloc[5]: Returns the entire 6th row (index starts from 0)

df.iloc[:,:]: Returns all rows and all columns (the entire DataFrame).

df.iloc[:,3]: Returns all rows from the 4th column (index 3).

df.iloc[:8,:2]: Returns rows from 0 to 7 (the first 8 rows) and columns from 0 to 1 (the first two columns).

df.iloc[1,1]:the value in the second row and second column.

df.iloc[3:5,0:2]: Returns rows from 3 to 4 and columns from 0 to 1 (subset of rows and columns).

12. Print Column Names as a List

print(df.columns.tolist())

df.columns.tolist(): Converts the column names to a list and prints it. This is useful to see all column names in a simple list format.

13. Accessing Specific Columns

df['Sepal Length']

df['Sepal Width']

df['Petal Length']

df['Petal Width']

df['Species']

Each of these lines accesses a specific column of the DataFrame. These columns are likely numerical or categorical columns, which will be used later for processing or analysis.

14. Min-Max Normalization

from sklearn import preprocessing

min\_max\_scaler = preprocessing.MinMaxScaler()

x = df.iloc[:,:4]

x\_scaled = min\_max\_scaler.fit\_transform(x)

df\_normalized = pd.DataFrame(x\_scaled)

**from sklearn import preprocessing**

This imports the preprocessing module from the **scikit-learn** library, which contains functions for scaling, encoding, normalizing, etc.

**min\_max\_scaler = preprocessing.MinMaxScaler()**

This creates a **MinMaxScaler** object.

* It does not change any data yet — just initializes the scaler.
* You can now use this object to **fit** and **transform** your data.

x = df.iloc[:,:4]: Selects the first 4 columns (likely features like Sepal Length, Sepal Width, etc.).

**x\_scaled = min\_max\_scaler.fit\_transform(x)**

* x is your original dataset (typically a NumPy array or a DataFrame).
* fit\_transform() is a two-in-one function that:
  1. **fit(x)** – Learns the min and max values from each feature/column in x.
  2. **transform(x)** – Applies the min-max scaling formula:

x\_scaled is the **normalized array** where all values are now scaled between 0 and 1.

df\_normalized = pd.DataFrame(x\_scaled): Converts the scaled data back into a pandas DataFrame df\_normalized.

15. Label Encoding for Categorical Data

label\_encoder = preprocessing.LabelEncoder()

df['Species'] = label\_encoder.fit\_transform(df['Species'])

print(df['Species'].unique())

label\_encoder = preprocessing.LabelEncoder(): Creates a label encoder object to convert categorical labels into numeric values.

**df['Species'] = label\_encoder.fit\_transform(df['Species'])**

* fit\_transform() does two things:
  1. **fit()**: Finds all unique classes in the column (e.g., 'setosa', 'versicolor', 'virginica')
  2. **transform()**: Maps each class to a unique integer:
     + 'setosa' → 0
     + 'versicolor' → 1
     + 'virginica' → 2
* Replaces the original string values in the 'Species' column with their corresponding numeric labels.

**print(df['Species'].unique())**

Displays the **unique encoded labels** in the updated 'Species' column.

16. Data Type Conversion

df['Sepal Length'] = df['Sepal Length'].astype(float)

df['Species'] = df['Species'].astype('category')

print(df.dtypes)

df['Sepal Length'] = df['Sepal Length'].astype(float): Converts the Sepal Length column to float type (though it’s likely already float64).

df['Species'] = df['Species'].astype('category'): Converts the Species column to category type, which is memory-efficient for categorical data.

print(df.dtypes): Prints the data types of all columns in the DataFrame after conversion, confirming the types.

Summary of Operations:

You loaded a dataset into a pandas DataFrame.

Explored the DataFrame using various pandas functions to understand its structure.

Normalized numeric columns using Min-Max Scaling.

Applied label encoding to the Species column to convert categorical values to numeric.

Ensured the correct data types for specific columns using astype().

1. A dataset is a collection of records, similar to a relational database table.
2. Pandas is an open-source Python package that provides high-performance, easy-to-use

data structures and data analysis tools for the labeled data in Python programming

language.

1. What can you do with Pandas?

. Indexing, manipulating, renaming, sorting, merging data frame

1. Instance: A single row of data is called an instance.
2. What can you do with NumPy?

1. Basic array operations: add, multiply, slice, flatten, reshape, index arrays

1. Feature: A single column of data is called a feature.
2. Training Dataset: A dataset that we feed into our machine learning algorithm to train

our model.

1. Testing Dataset: A dataset that we use to validate the accuracy of our model but is not

used to train the model. It may be called the validation dataset.

1. Data Represented in a Table:

Data should be arranged in a two-dimensional space made up of rows and columns.

1. Q: What is data wrangling and why is it important?  
   A: Data wrangling is the process of cleaning, structuring, and enriching raw data into a usable format for analysis. It's important because real-world data is often messy or incomplete.
2. isna() - function is also used to get the count of missing values of column and row wise count
3. of missing values
4. Q: What is the difference between head(), tail(), shape, and describe()?  
   A:

head(): Shows the first 5 rows.

tail(): Shows the last 5 rows.

shape: Returns a tuple (rows, columns).

describe(): Provides summary statistics for numeric columns.

1. Q: Why do we use MinMaxScaler? What does it do?  
   A: It scales data to a fixed range [0, 1]. Useful for algorithms like KNN or SVM where feature scale affects performance.
2. Q: What is label encoding? When is it used?  
   A: Label encoding converts categorical text labels into numbers.

The number of categories is small,When there are just a few unique categories, label encoding is simple and efficient.

1. Q: What is the difference between astype('category') and label encoding?  
   A:
   * astype('category'): Changes data type for memory efficiency and analysis.
   * LabelEncoder: Converts categories to integers for modeling.
2. Q: What does df.dtypes show?  
   A: It shows the data type of each column in the DataFrame.
3. Q: How do you check for missing values?  
   A: Using df.isnull().sum() or df.isnull().any().
4. Q: What is the use of df.info() vs df.describe()?  
   A:
   * info(): Shows data types, non-null counts, and memory usage.
   * describe(): Gives summary statistics (mean, std, etc.) of numeric columns.
5. Q: How do you access specific rows/columns using iloc?  
   A:
   * df.iloc[1, 1]: Value at 2nd row and 2nd column.
   * df.iloc[:, 0:2]: All rows, first two columns.
6. Q: What is the difference between iloc and loc?  
   A:
   * iloc: Access by integer position.
   * loc: Access by label/index.
7. Q: Why is normalization important?  
   A: Normalization is important because it scales all features to a similar range, which helps models learn faster and more accurately, especially in algorithms that rely on distance or gradient calculations.
8. Q: What is the difference between normalization and standardization?  
   A:
   * Normalization: Scales to [0, 1].
   * Standardization: Transforms to mean = 0, std = 1.
9. Q: What happens if we don't encode categorical data?  
   A: Most ML algorithms cannot handle non-numeric data, leading to errors.
10. Q: How do you handle missing values?  
    A: Use methods like fillna() Used to **fill missing values** or dropna() to remove rows.
11. Q: Why convert Sepal Length to float?  
    A: To ensure it's treated as a numerical feature for analysis and modeling.
12. Q: What does astype('category') do?  
    A: Converts a column to categorical data type, reducing memory and enabling category-based operations.
13. Q: How do you convert a string column to numeric if needed?  
    A: Using pd.to\_numeric(column) or astype(float).
14. Q: Which dataset are you using and where is it from?  
    A: I used the Iris dataset downloaded from Kaggle.  
    *(Example URL:* [*https://www.kaggle.com/datasets/uciml/iris*](https://www.kaggle.com/datasets/uciml/iris)*)*
15. Q: What are the features and target in your dataset?  
    A:
    * Features: Sepal Length, Sepal Width, Petal Length, Petal Width
    * Target: Species
16. Q: What are the next steps after this preprocessing?  
    A: Splitting the data, training ML models, evaluating performance, and deploying if needed.

1. What is data preprocessing and why is it important?  
Data preprocessing is the process of cleaning, transforming, and organizing raw data to make it suitable for analysis or machine learning. It improves data quality and model performance.

2. What is the Iris dataset?  
The Iris dataset is a famous multivariate dataset containing 150 samples of iris flowers, with 4 features: sepal length, sepal width, petal length, and petal width, classified into 3 species.

3. What are the different types of features in the Iris dataset?  
There are 4 numerical features (float) and 1 categorical target feature (species).

4. What is the difference between head(), tail(), and describe()?

head(): Shows the first 5 rows.

tail(): Shows the last 5 rows.

describe(): Gives summary stats like mean, std, min, max, etc.

5. What are missing values? How do you handle them?  
Missing values are null or NaN entries in the dataset. We can handle them by:

**Deletion**:

Remove rows (dropna()) or columns with too many missing values.

**Imputation**:

Fill missing values using **mean**, **median**, or **mode** with fillna().

**Advanced Methods**:

Use models like **KNN imputation**, **regression**, or **iterative imputation** for more accurate filling.

6. How do you load a CSV file using Pandas?

import pandas as pd

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

7. How can you check for null values in a dataset?

df.isnull().sum()

8. What does df.info() show?  
It displays column names, data types, non-null counts, and memory usage.

9. How do you normalize data using MinMaxScaler?

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaled\_data = scaler.fit\_transform(df[['sepal\_length', 'sepal\_width']])

10. How do you convert categorical data into numerical?  
Using label encoding or one-hot encoding:

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df['species'] = le.fit\_transform(df['species'])

11. How do you plot a histogram in Seaborn?

import seaborn as sns

sns.histplot(df['sepal\_length'])

12. What does a pair plot tell us?  
A pair plot shows scatter plots between all pairs of features and histograms on the diagonal, helping visualize relationships and clusters.

13. Why is data visualization important in preprocessing?  
It helps identify outliers, missing values, feature distributions, and correlations between variables.

15. Why do we use encoding for categorical variables?  
We use encoding because **machine learning models require numerical input**, and **categorical variables are in text form** (e.g., "Male", "Female", "Yes", "No").

16. What is label encoding vs. one-hot encoding?

Label encoding assigns an integer to each category.

One-hot encoding creates binary columns for each category.

17. What happens if data is not normalized?  
Features with large values may dominate, causing poor model performance and convergence issues.

18. If your dataset has 30% missing values in one column, what would you do?  
Depending on context:

Drop the column if it's not important.

Impute missing values using mean/median if the column is crucial.

19. When would you use imputation vs. deletion?

Imputation: When data loss is unacceptable.

Deletion: When nulls are few or in non-critical columns.

20. What preprocessing steps are essential before ML modeling?

Handling nulls

Encoding categoricals

Scaling/normalizing

Removing outliers

Feature selection

1. Explain DataFrame with Suitable Example

A DataFrame is a two-dimensional, labeled data structure in Pandas. It is similar to a table in a database or an Excel spreadsheet.

Example:

import pandas as pd

data = {

'Name': ['Raj', 'Anita', 'John'],

'Age': [21, 22, 23],

'Marks': [85, 90, 88]

}

df = pd.DataFrame(data)

print(df)

Output:

Name Age Marks

0 Raj 21 85

1 Anita 22 90

2 John 23 88

2. What is the Limitation of the Label Encoding Method?

Label Encoding assigns a unique integer to each category in a column.

Limitation:

It creates an ordinal relationship between categories where none exists.  
For example:

red = 0, green = 1, blue = 2

A machine learning model might think green > red, which is not true.

Use One-Hot Encoding if categories are nominal (no order).

3. What is the Need for Data Normalization?

Normalization scales all feature values to a fixed range, usually [0, 1].

Purpose:

To treat all features equally

It makes training **faster and more accurate**.

To speed up convergence of ML algorithms

To improve accuracy by avoiding dominance of large-scale features

Example:

Features like "age" (0–100) and "salary" (10,000–1,00,000) have different scales, which may bias the model if not normalized.

4. What are the Different Techniques for Handling Missing Data?

Here are common techniques:

1. Remove missing values

df.dropna()

Removes rows/columns with NaNs.

2. Imputation with mean/median/mode

df['column'].fillna(df['column'].mean(), inplace=True)

Fills missing values with statistical estimates.

3. Forward/Backward Fill

df.fillna(method='ffill') # Forward fill

df.fillna(method='bfill') # Backward fill

4. Using ML models for imputation

Advanced method using KNN, regression, etc., to predict missing values.