

**Mecha**tronics

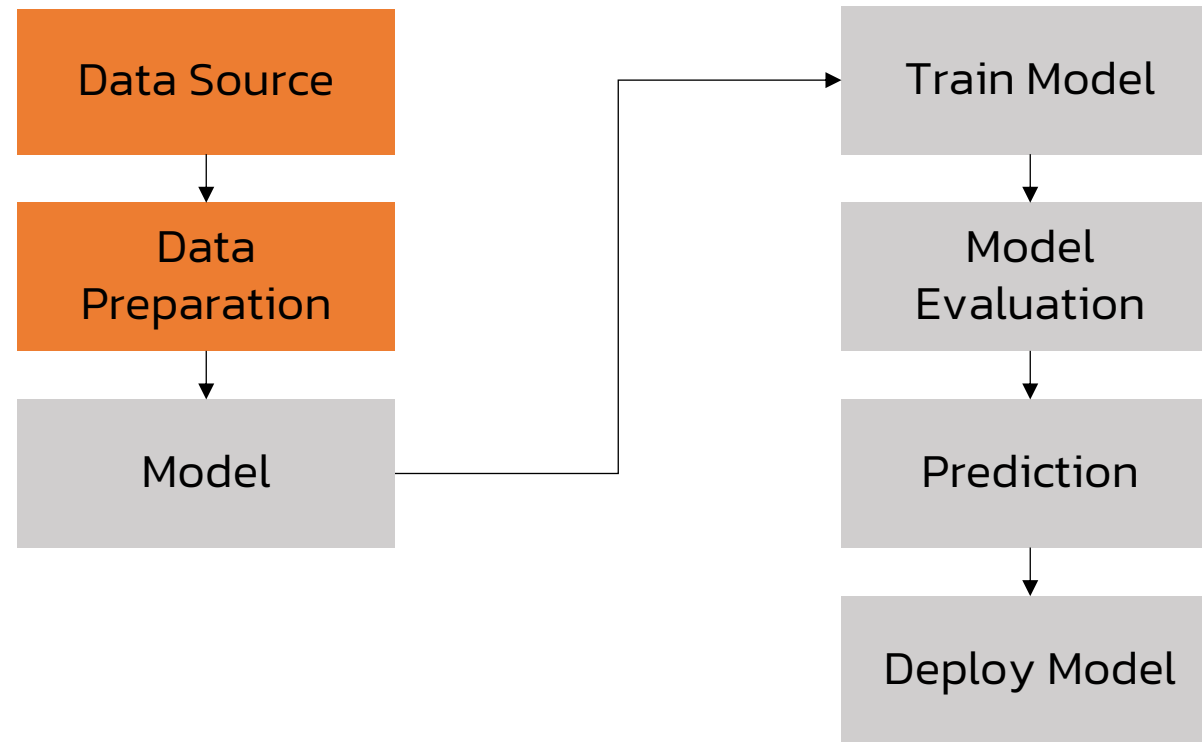
# Day 2

## Preparing Dataset for Machine Learning

# Outline

- Understanding Data
- Data Cleaning techniques
- Data Encoding and Transformation
- Splitting Data for model training
- Data Exploration and Visualization

# Processes for Training ML model




# (1) Understanding Data

- Library Scikit-learn
- Data for Machine Learning
  - Supervised Learning
  - Unsupervised Learning
- Data sources
  - Defined data
  - Imported data

# Scikit-learn

- It is Machine learning library
- Usually called "SKlearn"
- Developed by Google's team
- Using with Data Science's Libraries like : NumPy, Pandas, Matplotlib etc.

# Scikit-learn

[Install](#) [User Guide](#) [API](#) [Examples](#) [Community](#) [More](#) 1.6.0 (stable)

## scikit-learn

Machine Learning in Python

[Getting Started](#) [Release Highlights for 1.6](#)

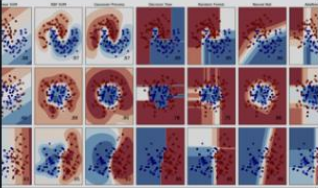
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

### Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.

**Algorithms:** [Gradient boosting](#), [nearest neighbors](#), [random forest](#), [logistic regression](#), and [more...](#)



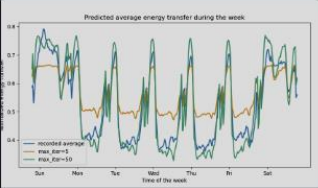
Examples

### Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, stock prices.

**Algorithms:** [Gradient boosting](#), [nearest neighbors](#), [random forest](#), [ridge](#), and [more...](#)



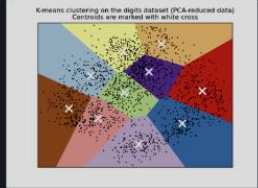
Examples

### Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, grouping experiment outcomes.

**Algorithms:** [k-Means](#), [HDBSCAN](#), [hierarchical clustering](#), and [more...](#)



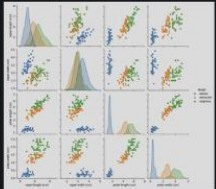
Examples

### Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, increased efficiency.

**Algorithms:** [PCA](#), [feature selection](#), [non-negative matrix factorization](#), and [more...](#)



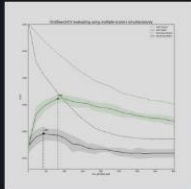
Examples

### Model selection

Comparing, validating and choosing parameters and models.

**Applications:** Improved accuracy via parameter tuning.

**Algorithms:** [Grid search](#), [cross validation](#), [metrics](#), and [more...](#)



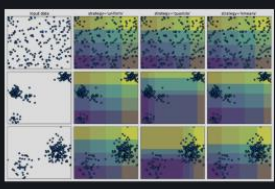
Examples

### Preprocessing

Feature extraction and normalization.

**Applications:** Transforming input data such as text for use with machine learning algorithms.

**Algorithms:** [Preprocessing](#), [feature extraction](#), and [more...](#)



Examples

<https://scikit-learn.org/stable/>

# How to use Sklearn for train ML model

```
"""
# Import necessary libraries
import ____
import ____
from sklearn.____ import ClassName
from sklearn.____ import FunctionName

# Defined Data
X = ____
y = ____
data = ____
data = pd.read_csv(____)

# Splitting or Encoding or Scaling data
encoder = LabelEncoder()
y = encoder.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=____, random_state=____)
sacler = ____()
X_train = sacler.fit_transform(X_train)
X_test = sacler.transform(X_test)

# Create and train the model
model = ModelName()
model.fit(X,y)

# Make predictions
y_pred = model.predict(X)

# Evaluate the model
score = model.score(X,y)

# Plot the results (Optional)
"""
```

# How to use Sklearn for train ML model : Regression (Example)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Generate sample data
np.random.seed(0)
X = np.random.rand(100, 1) * 10
y = 2 * X.squeeze() + 1 + np.random.randn(100)

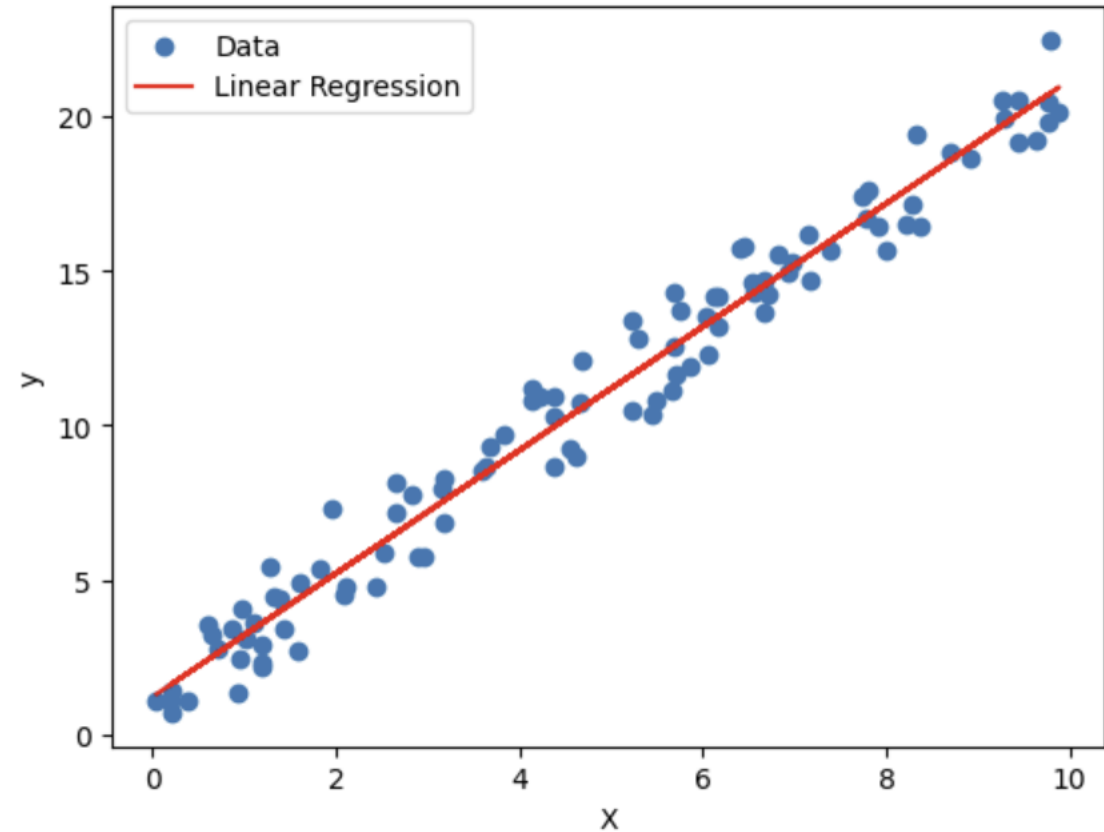
# Create and train the model
model = LinearRegression()
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)

# Evaluate the model
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'R-squared: {r2:.2f}')

# Plot the results
plt.scatter(X, y, label='Data')
plt.plot(X, y_pred, color='red', label='Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

Mean Squared Error: 0.99  
R-squared: 0.97



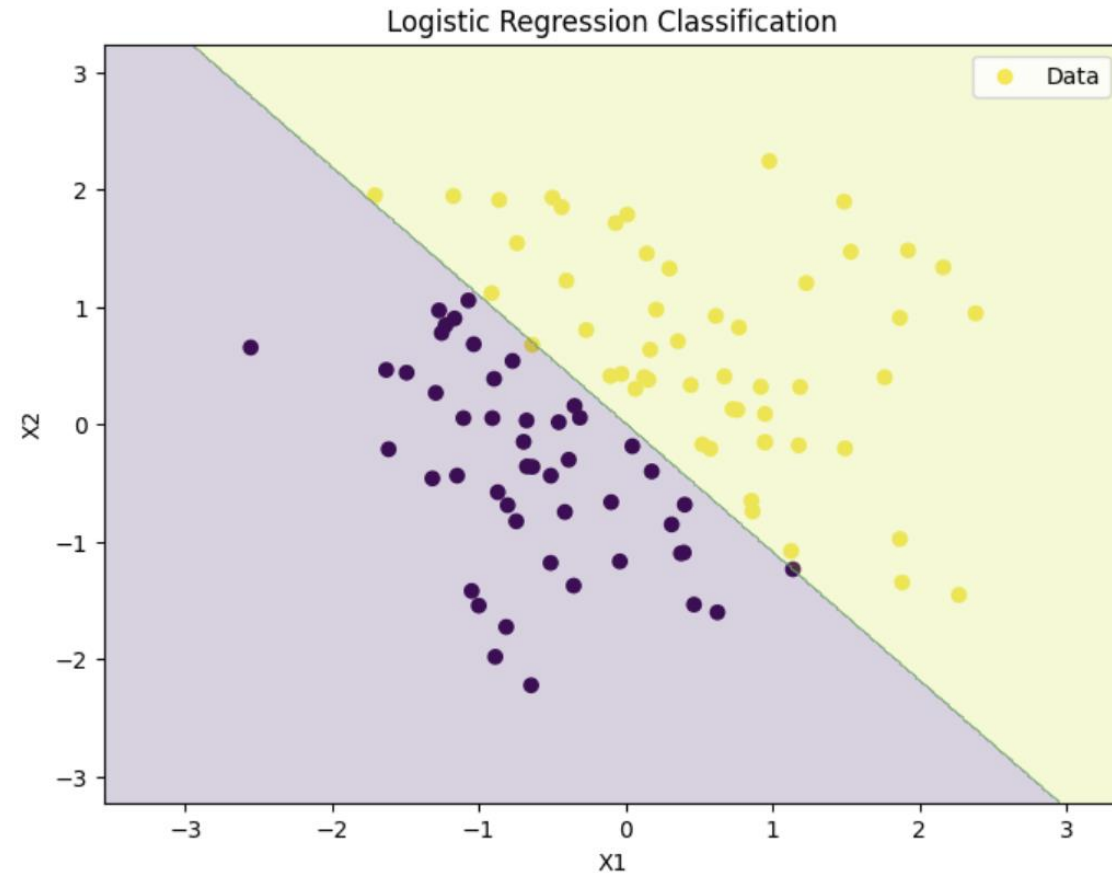


# How to use Sklearn for train ML model : Classification (Example)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
# Generate sample data
np.random.seed(0)
X = np.random.randn(100, 2)
y = np.array([1 if x[0] + x[1] > 0 else 0 for x in X])

# Create and train the model
model = LogisticRegression()
model.fit(X, y)
# Make predictions
y_pred = model.predict(X)
# Evaluate the model
accuracy = accuracy_score(y, y_pred)
conf_matrix = confusion_matrix(y, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(f'Confusion Matrix:\n{conf_matrix}')
# Plot the results
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', label='Data')
# Create a meshgrid for plotting the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
# Plot the decision boundary
plt.contourf(xx, yy, Z, alpha=0.2)
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.title('Logistic Regression Classification')
plt.show()
```

Accuracy: 0.98  
Confusion Matrix:  
[[47 1]  
 [ 1 51]]



# Data for Supervised Learning

## (1) Regression

**Goal:** Predict a continuous numerical value.

**Data Format:**

- **Features (X):** Independent variables, often numerical.
- **Target (y):** Continuous numerical value.

**Example: Predicting house prices**

- **Features:** Square footage, number of bedrooms, location, etc.
- **Target:** House price in dollars.

# Data for Supervised Learning

## Regression : Predicting house prices

```
import pandas as pd

# Sample data (replace with actual data)
data = {
    'Square_Footage': [1500, 1800, 2200, 1200, 2500],
    'Bedrooms': [3, 4, 5, 2, 3],
    'Bathrooms': [2, 2.5, 3, 1.5, 3],
    'Location': ['Suburb', 'City Center', 'Suburb', 'Rural', 'City Center'],
    'House_Price': [250000, 320000, 450000, 180000, 500000]
}

# Create a pandas DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print(df)
```

	Square_Footage	Bedrooms	Bathrooms	Location	House_Price
0	1500	3	2.0	Suburb	250000
1	1800	4	2.5	City Center	320000
2	2200	5	3.0	Suburb	450000
3	1200	2	1.5	Rural	180000
4	2500	3	3.0	City Center	500000

# Data for Supervised Learning

## (2) Classification

**Goal:** Predict a categorical label or class.

**Data Format:**

- **Features (X):** Independent variables, can be numerical, categorical, or a mix.
- **Target (y):** Categorical label (e.g., "spam" or "not spam," "dog" or "cat").

**Example:** Spam detection

- **Features:** Words in the email, sender's address, etc.
- **Target:** "spam" or "not spam."

# Data for Supervised Learning

## (2) Classification : Spam detection

```
import pandas as pd
# Sample data (replace with actual data)
data = {
    'Email_Text': [
        "Urgent! You've won a free vacation!",
        "Meeting scheduled for tomorrow at 2 PM.",
        "Free Viagra! No prescription needed.",
        "Project deadline is approaching.",
        "Enlarge your..."
    ],
    'Sender_Address': [
        "unknown@spam.com",
        "boss@company.com",
        "generic_sender@email.com",
        "colleague@company.com",
        "suspicious_sender@mail.net"
    ],
    'Target': [
        "spam",
        "not spam",
        "spam",
        "not spam",
        "spam"
    ]
}
# Create a pandas DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print(df)
```

	Email_Text	Sender_Address \	Target
0	Urgent! You've won a free vacation!	<a href="mailto:unknown@spam.com">unknown@spam.com</a>	spam
1	Meeting scheduled for tomorrow at 2 PM.	<a href="mailto:boss@company.com">boss@company.com</a>	not spam
2	Free Viagra! No prescription needed.	<a href="mailto:generic_sender@email.com">generic_sender@email.com</a>	spam
3	Project deadline is approaching.	<a href="mailto:colleague@company.com">colleague@company.com</a>	not spam
4	Enlarge your...	<a href="mailto:suspicious_sender@mail.net">suspicious_sender@mail.net</a>	spam

# Data for Supervised Learning

- Classification and Regression

Feature	Regression	Classification
Target Variable	Continuous (e.g., price, temperature)	Categorical (e.g., class labels, categories)
Goal	Predict a numerical value	Predict a class or category
Examples	Predicting stock prices, forecasting weather	Image recognition, spam detection

# Data for Unsupervised Learning

The data used for training lacks explicit labels or target values. The algorithms are tasked with identifying patterns, structures, and relationships within the data itself.

## Unlabeled Data

The data points do not have predefined categories or classes associated with them.

## Focus on Patterns and Structures

The goal is to discover underlying patterns, clusters, or anomalies within the data.

## Exploratory in Nature

Unsupervised learning is often used to explore data and gain insights that might not be apparent through manual analysis.

# Data for Unsupervised Learning

## Example

A retail company wants to understand its customer base better. They have a dataset of customer information, including:

- Age
- Gender
- Location
- Purchase history (products bought, frequency, amount spent)
- Browsing behavior (website visits, time spent on site, products viewed)

Customer_ID	Age	Gender	Location	Products_Bought	Purchase_Frequency	Amount_Spent	Website_Visits	Time_Spent_on_Site	Products_Viewed
1	25	Female	City A	['Shoes', 'T-shirt']	2	150	5	30	['Shoes', 'Jeans', 'T-shirt']
2	38	Male	City B	['Laptop', 'Phone']	1	1200	2	15	['Laptop', 'Phone', 'Tablet']
3	42	Female	City A	['Dress', 'Bag']	3	300	8	60	['Dress', 'Bag', 'Shoes']
4	20	Male	City C	['Headphones', 'Game']	4	80	10	45	['Headphones', 'Game', 'Controller']
5	55	Female	City B	['Jewelry', 'Watch']	1	500	1	5	['Jewelry', 'Watch', 'Bracelet']



# Data for Unsupervised Learning

```
import pandas as pd
import numpy as np
# Sample data
data = {
    'Customer_ID': [1, 2, 3, 4, 5],
    'Age': [25, 38, 42, 20, 55],
    'Gender': ['Female', 'Male', 'Female', 'Male', 'Female'],
    'Location': ['City A', 'City B', 'City A', 'City C', 'City B'],
    'Products_Bought': [
        ['Shoes', 'T-shirt'],
        ['Laptop', 'Phone'],
        ['Dress', 'Bag'],
        ['Headphones', 'Game'],
        ['Jewelry', 'Watch']
    ],
    'Purchase_Frequency': [2, 1, 3, 4, 1],
    'Amount_Spent': [150, 1200, 300, 80, 500],
    'Website_Visits': [5, 2, 8, 10, 1],
    'Time_Spent_on_Site': [30, 15, 60, 45, 5],
    'Products_Viewed': [
        ['Shoes', 'Jeans', 'T-shirt'],
        ['Laptop', 'Phone', 'Tablet'],
        ['Dress', 'Bag', 'Shoes'],
        ['Headphones', 'Game', 'Controller'],
        ['Jewelry', 'Watch', 'Bracelet']
    ]
}
# Create a pandas DataFrame
df = pd.DataFrame(data)
# Display the DataFrame
print(df)
```

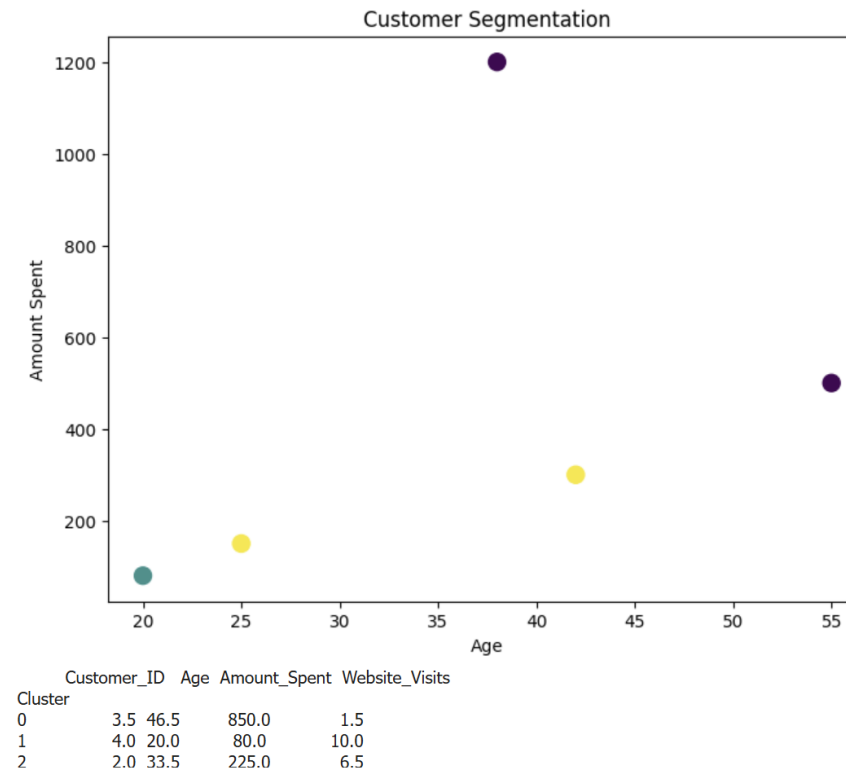
	Customer_ID	Age	Gender	Location	Products_Bought	Purchase_Frequency \
0	1	25	Female	City A	[Shoes, T-shirt]	2
1	2	38	Male	City B	[Laptop, Phone]	1
2	3	42	Female	City A	[Dress, Bag]	3
3	4	20	Male	City C	[Headphones, Game]	4
4	5	55	Female	City B	[Jewelry, Watch]	1

	Amount_Spent	Website_Visits	Time_Spent_on_Site \
0	150	5	30
1	1200	2	15
2	300	8	60
3	80	10	45
4	500	1	5

	Products_Viewed
0	[Shoes, Jeans, T-shirt]
1	[Laptop, Phone, Tablet]
2	[Dress, Bag, Shoes]
3	[Headphones, Game, Controller]
4	[Jewelry, Watch, Bracelet]

# Data for Unsupervised Learning

```
import pandas as pd
import numpy as np
# Sample data
data = {
    'Customer_ID': [1, 2, 3, 4, 5],
    'Age': [25, 38, 42, 20, 55],
    'Gender': ['Female', 'Male', 'Female', 'Male', 'Female'],
    'Location': ['City A', 'City B', 'City A', 'City C', 'City B'],
    'Products_Bought': [
        ['Shoes', 'T-shirt'],
        ['Laptop', 'Phone'],
        ['Dress', 'Bag'],
        ['Headphones', 'Game'],
        ['Jewelry', 'Watch']
    ],
    'Purchase_Frequency': [2, 1, 3, 4, 1],
    'Amount_Spent': [150, 1200, 300, 80, 500],
    'Website_Visits': [5, 2, 8, 10, 1],
    'Time_Spent_on_Site': [30, 15, 60, 45, 5],
    'Products_Viewed': [
        ['Shoes', 'Jeans', 'T-shirt'],
        ['Laptop', 'Phone', 'Tablet'],
        ['Dress', 'Bag', 'Shoes'],
        ['Headphones', 'Game', 'Controller'],
        ['Jewelry', 'Watch', 'Bracelet']
    ]
}
# Create a pandas DataFrame
df = pd.DataFrame(data)
# Display the DataFrame
print(df)
```



# Defined data

## DataFrame

It is a two-dimensional, size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). It's a fundamental data structure in the Python library Pandas, designed for efficient handling and analysis of tabular data.

```
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 22],
        'City': ['New York', 'London', 'Paris']}

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

	Name	Age	City
0	Alice	25	New York
1	Bob	30	London
2	Charlie	22	Paris

# Read data from csv and excel

- csv file

```
pd.read_csv(filepath_or_buffer, sep=',', delimiter=None, header='infer', names=None, index_col=None, usecols=None,
            squeeze=False, dtype=None, engine=None, converters=None, true_values=None, false_values=None,
            skiprows=None, nrows=None, na_values=None, keep_default_na=True, na_filter=True,
            verbose=False, skip_blank_lines=True, comment=None, encoding=None, quoting=csv.QUOTE_MINIMAL,
            quotechar='"', decimal='.', lineterminator=None, thousands=None,
            comment=None, converters=None, dtype=None, true_values=None, false_values=None,
            nrows=None, chunksize=None, iterator=False, compression='infer',
            thousands=None, decimal='.', decimal=',',
            error_bad_lines=True, warn_bad_lines=True, on_bad_lines='warn',
            skipinitialspace=False, engine='c',
            dtype_support='extension', low_memory=True, memory_map=False, float_precision=None)
```

filepath_or_buffer	Path to the CSV file.
sep	Delimiter to use. Default is ','.
header	Row number(s) to use as the column names. Defaults to 0.
Names	List of column names to use if the file doesn't have a header row.
index_col	Column to use as the row index.
usecols	List of column names or indices to read.
nrows	Number of rows to read from the beginning of the file.
skiprows	Number of rows to skip from the beginning of the file.
na_values	Values to consider as missing values.
dtype	Data types for each column.
encoding	Encoding of the file.

# Read data from csv and excel

- csv file

```
import pandas as pd

# Read data from a CSV file
df = pd.read_csv('my_data.csv')

# Read specific columns
df = pd.read_csv('my_data.csv', usecols=['column1', 'column2'])

# Read only the first 5 rows
df = pd.read_csv('my_data.csv', nrows=5)
```

# Read data from csv and excel

- excel file

```
pd.read_excel(file, sheet_name=0, header=0, names=None, index_col=None, usecols=None, squeeze=False, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skiprows=None, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, parse_dates=False, date_parser=None, thousands=None, decimal='.', converters=None, dtype=None, true_values=None, false_values=None, skipfooter=0, convert_float=True, ...)
```

`pd.read_excel('file.xlsx')`

This reads the data from the first sheet of the Excel file named 'your\_file.xlsx' and stores it in a DataFrame named df.

`sheet_name`

This parameter allows you to specify the sheet you want to read. You can provide the sheet name as a string or its index (0 for the first sheet, 1 for the second, etc.).

`nrows`

This parameter specifies the number of rows to read from the beginning of the sheet.

`usecols`

This parameter allows you to select specific columns to read. You can provide a list of column names or indices.

`header`

This parameter specifies the row number to use as the column header. The default is 0 (the first row).

`na_values`

This parameter specifies the values that should be considered as missing values (e.g., 'N/A', '', 'None').

# Read data from csv and excel

- excel file

```
import pandas as pd

# Read data from the first sheet of an Excel file
df = pd.read_excel('your_file.xlsx')

# Read data from a specific sheet (by name)
df = pd.read_excel('your_file.xlsx', sheet_name='Sheet2')

# Read data from a specific sheet (by index)
df = pd.read_excel('your_file.xlsx', sheet_name=1)

# Read only the first 5 rows
df = pd.read_excel('your_file.xlsx', nrows=5)

# Read specific columns
df = pd.read_excel('your_file.xlsx', usecols=['Column1', 'Column2'])

# Specify the header row (if not the first row)
df = pd.read_excel('your_file.xlsx', header=1)

# Handle missing values
df = pd.read_excel('your_file.xlsx', na_values=['N/A', ''])
```

## (2) Data Cleaning techniques

### Missing values

It refers to gaps or empty spaces within the data.

It occurs when:

1. Data Collection Issues
  - Surveys with unanswered questions
  - Equipment malfunctions leading to missing sensor readings.
  - Data entry errors.
2. Data Privacy
  - Sensitive information intentionally omitted.
3. Data Transformation
  - Calculations resulting in undefined values (e.g., division by zero).

machine	Temperature (°C)	Pressure (psi)	Machine Fault
A	100		N
A	200	50	N
	250		N
B	100	50	N
B	200		Y
C		50	Y
C		50	N



## (2) Data Cleaning techniques

### Missing values

machine	Temperature (°C)	Pressure (psi)	Machine Fault
A	100		N
A	200	50	N
	250		N
B	100	50	N
B	200		Y
C		50	Y
C		50	N

machine	Temperature (°C)	Pressure (psi)	Machine Fault
A	100	NaN	N
A	200	50	N
NaN	250	NaN	N
B	100	50	N
B	200	NaN	Y
C	NaN	50	Y
C	NaN	50	N

NaN = Not a number

# Handling Missing value using Pandas

Method	Purposes
<code>isnull().any()</code>	A method used to check if any value in a DataFrame or Series is missing (represented by NaN).
<code>isnull().sum()</code>	A method used to count the number of missing (NaN) values in each column of a DataFrame.
<code>dropna()</code>	A method used to remove rows or columns containing missing values (NaN) from a DataFrame.
<code>fillna()</code>	A method used to replace missing values (NaN) in a DataFrame or Series with specified values or methods.

# Handling Missing value using Pandas

```
import pandas as pd
import numpy as np

df = pd.read_csv('missing-values.csv')

#display(df)
print(df.isnull().any())
print()
print(df.isnull().sum())

# Check some column for example :
# print(df['volume'].isnull().any())
# print(df[['volume', 'weight']].isnull().sum())

df.dropna(inplace=True) #inplace=True is replace object in same
position
display(df)
```

```
car      False
color    False
volume    True
weight    True
co2       False
dtype: bool
```

```
car      0
color    0
volume    3
weight    2
co2      0
dtype: int64
```

	car	color	volume	weight	co2
0	Honda Civic	Red	1600.0	1252.0	94
5	Ford Focus	Blue	2000.0	1328.0	105
7	Benz E-Class	White	2100.0	1605.0	115
8	Ford Fiesta	Red	1500.0	1112.0	98
9	Volvo XC70	Silver	2000.0	1746.0	117

# Handling Missing value using Pandas

```
# example fillna() with mean
import pandas as pd
import numpy as np

df = pd.read_csv('missing-values.csv')

#display(df)
v_mean = df['volume'].mean()
df['volume'].fillna(v_mean, inplace=True)

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

display(df)
```

car	color	volume	weight	co2
Honda Civic	Red	1600	1252	94
Suzuki Swift	White	1300		101
Mazda 3	Black		1280	104
Benz CLA	Black		1465	102
Mini Cooper	Red	1500		105
Ford Focus	Blue	2000	1328	105
Benz C-Class	Silver		1365	99
Benz E-Class	White	2100	1605	115
Ford Fiesta	Red	1500	1112	98
Volvo XC70	Silver	2000	1746	117

	car	color	volume	weight	co2
0	Honda Civic	Red	1600.000000	1252.000	94
1	Suzuki Swift	White	1300.000000	1394.125	101
2	Mazda 3	Black	1714.285714	1280.000	104
3	Benz CLA	Black	1714.285714	1465.000	102
4	Mini Cooper	Red	1500.000000	1394.125	105
5	Ford Focus	Blue	2000.000000	1328.000	105
6	Benz C-Class	Silver	1714.285714	1365.000	99
7	Benz E-Class	White	2100.000000	1605.000	115
8	Ford Fiesta	Red	1500.000000	1112.000	98
9	Volvo XC70	Silver	2000.000000	1746.000	117

# Handling Missing value using Pandas

```
# example fillna()

import pandas as pd
import numpy as np

df = pd.read_csv('missing-values.csv')

# display(df)

# ffill = fill value forward
df['volume'].fillna(method='ffill', inplace=True) #method='pad'

# ffill = fill value backward
df['weight'].fillna(method='bfill', inplace=True) #bfill()

display(df)
```

car	color	volume	weight	co2
Honda Civic	Red	1600	1252	94
Suzuki Swift	White	1300		101
Mazda 3	Black		1280	104
Benz CLA	Black		1465	102
Mini Cooper	Red	1500		105
Ford Focus	Blue	2000	1328	105
Benz C-Class	Silver		1365	99
Benz E-Class	White	2100	1605	115
Ford Fiesta	Red	1500	1112	98
Volvo XC70	Silver	2000	1746	117

	car	color	volume	weight	co2
0	Honda Civic	Red	1600.0	1252.0	94
1	Suzuki Swift	White	1300.0	1280.0	101
2	Mazda 3	Black	1300.0	1280.0	104
3	Benz CLA	Black	1300.0	1465.0	102
4	Mini Cooper	Red	1500.0	1328.0	105
5	Ford Focus	Blue	2000.0	1328.0	105
6	Benz C-Class	Silver	2000.0	1365.0	99
7	Benz E-Class	White	2100.0	1605.0	115
8	Ford Fiesta	Red	1500.0	1112.0	98
9	Volvo XC70	Silver	2000.0	1746.0	117

# (3) Data Encoding and Transformation

## Data Encoding

In machine learning, data encoding is the process of transforming categorical data into a numerical format that can be understood and processed by machine learning algorithms.

# Data Encoding

## Label Encoding

Assigns a unique integer to each category.

e.g.          Color: Red -> 1, Blue -> 2, Green -> 3

## One-Hot Encoding

Creates a new binary column for each unique category within a categorical feature. A value of 1 is assigned to the column corresponding to the category present in that row, and 0 to the other columns.

e.g.          Color: Red -> Color\_Red : 1, Color\_Blue : 0, Color\_Green : 0  
              Color: Blue -> Color\_Red : 0, Color\_Blue : 1, Color\_Green : 0

## Ordinal Encoding

Similar to label encoding, but used when there's an inherent order or ranking among the categories.

## Target Encoding (Mean Encoding)

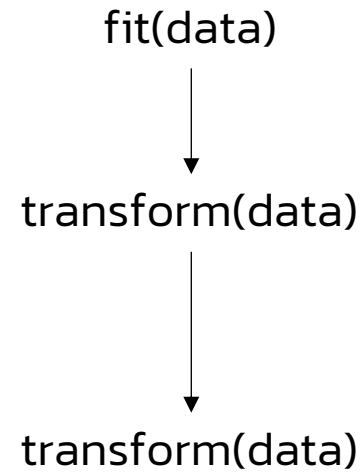
Replaces each category with the mean value of the target variable for that category.

# Data Encoding

## Class LabelEncoder

- Module: Sklearn.preprocessing

Method 1: using fit() -> transform()





# Data Encoding

```
# Import the LabelEncoder class from scikit-learn
from sklearn.preprocessing import LabelEncoder
# Create a list of fruits (categorical data)
data = ['Apple', 'Mango', 'Grape', 'Durian', 'Mango', 'Grape']
# Create a LabelEncoder object
encoder = LabelEncoder()
# Fit the encoder to the data, learning the unique labels
encoder.fit(data)
# Transform the data into numerical labels
enc_data1 = encoder.transform(data)
print(enc_data1)
#[0 3 2 1 3 2]
# If you want to transform new data (from the same set), you can directly use the transform() method
enc_data2 = encoder.transform(['Mango', 'Durian', 'Durian'])
print(enc_data2)
#[3 1 1]
enc_data3 = encoder.transform(['Grape', 'Mango', 'Mango', 'Durian'])
print(enc_data3)
#[2 3 3 1]

# enc_data4 = encoder.transform(['mango', 'Mangosteen', 'Melon']) # This line is Error
# An error occurs because:
# 'mango' has a different case (lowercase) than the data it was fitted on
# 'Mangosteen' and 'Melon' are not in the data that was fitted
```

# Data Encoding

Method 2 : using `fit_transform()` – combined fit and transform together

`fit_transform(data)`



`transform(data)`



`transform(data)`

# Data Encoding

```
from sklearn.preprocessing import LabelEncoder

data = ['Apple', 'Mango', 'Grape', 'Durian', 'Mango', 'Grape']
encoder = LabelEncoder()

enc_data1 = encoder.fit_transform(data)
print(enc_data1)  # Output: [0 3 2 1 3 2]

enc_data2 = encoder.transform(['Mango', 'Durian', 'Durian'])
print(enc_data2)  # Output: [3 1 1]

enc_data3 = encoder.transform(['Apple', 'Grape', 'Durian', 'Grape'])
print(enc_data3)  # Output: [0 2 1 2]
```

# Data Encoding

Use `LabelEncoder` to convert categorical data into numerical representations and back.

The `inverse_transform` method can be used to convert numerical labels back to their original categorical values.

```
from sklearn.preprocessing import LabelEncoder

data = ['Apple', 'Mango', 'Grape', 'Durian', 'Mango', 'Grape']
encoder = LabelEncoder()

enc_data = encoder.fit_transform(data)
print(enc_data)

# Transform Numeric to String
inv = encoder.inverse_transform(enc_data)
print(inv)

print(encoder.inverse_transform([1, 1, 0, 2]))
```

# Data Encoding

## Encode and transform with excel or csv file

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

df = pd.read_excel('test-label-encoder.xlsx')
display(df)

enc_weather = LabelEncoder()
enc_weather.fit(df['Weather'])
df['Weather'] = enc_weather.transform(df['Weather'])

enc_timeofweek = LabelEncoder().fit(df['TimeOfWeek'])
df['TimeOfWeek'] = enc_timeofweek.transform(df['TimeOfWeek'])

df['TimeOfDay'] = LabelEncoder().fit_transform(df['TimeOfDay'])

display(df)
```

	Weather	TimeOfWeek	TimeOfDay	TrafficJam
0	Clear	Workday	Morning	Yes
1	Clear	Workday	Evening	Yes
2	Clear	Weekend	Lunch	No
3	Rainy	Workday	Morning	Yes
4	Rainy	Workday	Lunch	Yes
5	Rainy	Workday	Evening	Yes
6	Rainy	Weekend	Lunch	No
7	Snowy	Workday	Morning	Yes
8	Snowy	Workday	Evening	Yes
9	Snowy	Weekend	Lunch	No

	Weather	TimeOfWeek	TimeOfDay	TrafficJam
0	0	1	2	Yes
1	0	1	0	Yes
2	0	0	1	No
3	1	1	2	Yes
4	1	1	1	Yes
5	1	1	0	Yes
6	1	0	1	No
7	2	1	2	Yes
8	2	1	0	Yes
9	2	0	1	No

# Data Encoding

Encode and transform with excel or csv file in specific column

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
df = pd.read_excel('test-label-encoder.xlsx')
encoder = LabelEncoder()
# Transform every columns
df = df.apply(encoder.fit_transform)
# Transform a specific column
"""
df['Weather'] = df['Weather'].apply(encoder.fit_transform)
"""
# Transform many columns
"""
cols = ['Weather', 'TimeOfWeek', 'TimeOfDay']
df[cols] = df[cols].apply(encoder.fit_transform)
"""
# Another ways
"""
df.iloc[:, 0:3] = df.iloc[:, 0:3].apply(encoder.fit_transform)
"""
print(df)
```

	Weather	TimeOfWeek	TimeOfDay	TrafficJam
0	0	1	2	1
1	0	1	0	1
2	0	0	1	0
3	1	1	2	1
4	1	1	1	1
5	1	1	0	1
6	1	0	1	0
7	2	1	2	1
8	2	1	0	1
9	2	0	1	0

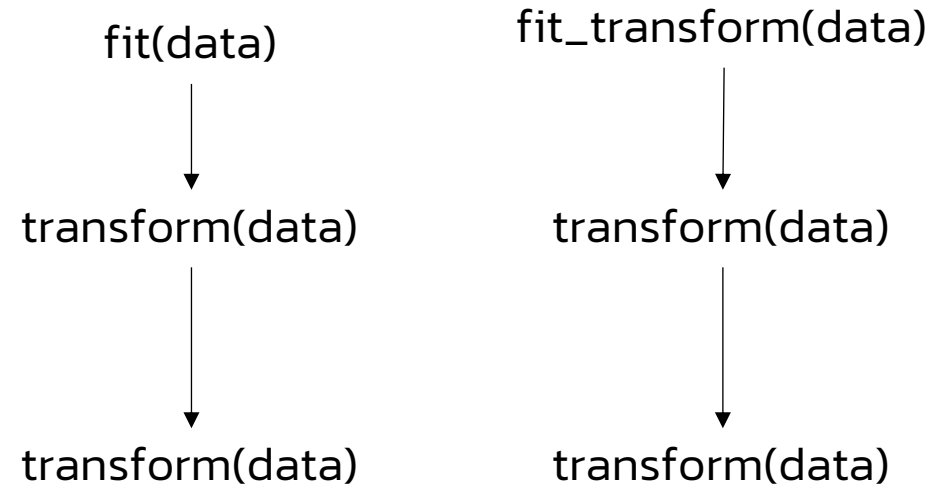
# Data Scaling

It is a crucial step in data preprocessing for many machine learning algorithms. It involves transforming the features of your dataset so that they fall within a specific range or distribution.

## Standard scaler

$$Z = \frac{x - \mu}{s}$$

Z = scaled data  
x = original data  
 $\mu$  = mean  
s = standard deviation



# Data Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
data = [[2, 2000],  
        [3.5, 5500],  
        [1.75, 2150],  
        [3.14, 4130],  
        [2.25, 2564]]
```

```
scaler = StandardScaler()  
scaler.fit(data)  
data2 = scaler.transform(data)  
print(data2)  
print()
```

```
data_test = [[1.23, 3210]]  
print(scaler.transform(data_test))
```

```
[[-0.78178395 -0.9416242 ]  
 [ 1.43919317  1.65585743]  
 [-1.1519468  -0.83030356]  
 [ 0.90615866  0.63912891]  
 [-0.41162109 -0.52305859]]
```

```
[[-1.92188553 -0.04363769]]
```



# Data Scaling

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

scaler = StandardScaler()

df = pd.read_csv('test-standard-scaler.csv')
display(df)

#fit() and transform()
scaler.fit(df[['volume', 'weight']])
df[['volume', 'weight']] = scaler.transform(df[['volume', 'weight']])
display(df)

data = [[1.9, 1000]]
print(scaler.transform(data))
print(scaler.transform([[2.3, 1234]]))
```

	car	color	volume	weight	co2
0	Honda Civic	Red	1.6	1252	94
1	Suzuki Swift	White	1.3	990	101
2	Mazda 3	Black	2.2	1280	104
3	Benz CLA	Black	1.5	1465	102
4	Mini Cooper	Red	1.5	1140	105
5	Ford Focus	Blue	2.0	1328	105
6	Benz C-Class	Silver	2.1	1365	99
7	Benz E-Class	White	2.1	1605	115
8	Ford Fiesta	Red	1.5	1112	98
9	Volvo XC70	Silver	2.0	1746	117

	car	color	volume	weight	co2
0	Honda Civic	Red	-0.576166	-0.349752	94
1	Suzuki Swift	White	-1.536443	-1.550735	101
2	Mazda 3	Black	1.344387	-0.221403	104
3	Benz CLA	Black	-0.896258	0.626620	102
4	Mini Cooper	Red	-0.896258	-0.863150	105
5	Ford Focus	Blue	0.704203	-0.001375	105
6	Benz C-Class	Silver	1.024295	0.168229	99
7	Benz E-Class	White	1.024295	1.268367	115
8	Ford Fiesta	Red	-0.896258	-0.991499	98
9	Volvo XC70	Silver	0.704203	1.914698	117

```
[[ 0.38411064 -1.50489638]]
[[ 1.66447944 -0.43226235]]
```

# Data Scaling

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

scaler = StandardScaler()

df = pd.read_csv('test-standard-scaler.csv')
display(df)

# fit_transform()
df[['volume', 'weight']] = scaler.fit_transform(df[['volume',
'weight']])
display(df)

data = [[1.9, 1000]]
print(scaler.transform(data))
print(scaler.transform([[2.3, 1234]]))
```

	car	color	volume	weight	co2
0	Honda Civic	Red	1.6	1252	94
1	Suzuki Swift	White	1.3	990	101
2	Mazda 3	Black	2.2	1280	104
3	Benz CLA	Black	1.5	1465	102
4	Mini Cooper	Red	1.5	1140	105
5	Ford Focus	Blue	2.0	1328	105
6	Benz C-Class	Silver	2.1	1365	99
7	Benz E-Class	White	2.1	1605	115
8	Ford Fiesta	Red	1.5	1112	98
9	Volvo XC70	Silver	2.0	1746	117

	car	color	volume	weight	co2
0	Honda Civic	Red	-0.576166	-0.349752	94
1	Suzuki Swift	White	-1.536443	-1.550735	101
2	Mazda 3	Black	1.344387	-0.221403	104
3	Benz CLA	Black	-0.896258	0.626620	102
4	Mini Cooper	Red	-0.896258	-0.863150	105
5	Ford Focus	Blue	0.704203	-0.001375	105
6	Benz C-Class	Silver	1.024295	0.168229	99
7	Benz E-Class	White	1.024295	1.268367	115
8	Ford Fiesta	Red	-0.896258	-0.991499	98
9	Volvo XC70	Silver	0.704203	1.914698	117

```
[[ 0.38411064 -1.50489638]]
[[ 1.66447944 -0.43226235]]
```

# (4) Data Splitting

## 1. Training Set

This is the primary portion of the data used to train the machine learning model. The model learns patterns and relationships within this set to make predictions.

## 2. Validation Set

This set is used to tune the model's hyperparameters (e.g., the number of trees in a random forest, the learning rate in a neural network). It helps to prevent overfitting to the training data.

## 3. Test Set

This set is used to evaluate the final performance of the trained model on unseen data. It provides an unbiased estimate of the model's generalization ability.

## (4) Data Splitting

X1	X2	X3	...	Y	
					Training set = 80%
					Test set = 20%

## (4) Data Splitting

X1	X2	X3	...	Y	
					Training set = 80%
					Validation set = 10%
					Test set = 10%

## (4) Data Splitting

Feature	Training Set – Test Set	Training Set – Validation Set – Test Set
Number of Sets	2	3
Hyperparameter Tuning	Not explicitly considered	Validation set used for tuning
Model Evaluation	Direct evaluation on the test set	More refined evaluation, separating hyperparameter tuning from final testing
Overfitting Risk	Higher risk of overfitting due to lack of dedicated validation	Lower risk of overfitting

## (4) Data Splitting

```
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=__, random_state=__)
```

<code>X</code>	Your feature data (independent variables).
<code>y</code>	Your target variable (dependent variable).
<code>test_size</code>	The proportion of the data to include in the test set, you can use <code>train_size</code> instead, (default: 0.25, meaning 25% of the data goes to the test set).
<code>train_size</code>	The proportion of the data to include in the training set (default: 0.75, meaning 75% of the data goes to the training set ).
<code>random_state</code>	A seed for the random number generator. This ensures that the same split is obtained every time the code is run with the same <code>random_state</code> .

# (4) Data Splitting

```
from sklearn.model_selection import train_test_split
import pandas as pd
```

```
df = pd.read_csv('test-standard-scaler.csv')
x = df.iloc[:, 2:4] #df[['volume', 'weight']]
y = df['co2']
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25 random_state=0)
```

```
display(x_train)
display(y_train)
print()
display(x_test)
display(y_test)
```

	volume	weight	
9	2.0	1746	x_train
1	1.3	990	
6	2.1	1365	
7	2.1	1605	
3	1.5	1465	
0	1.6	1252	
5	2.0	1328	
9	117		y_train
1	101		
6	99		
7	115		
3	102		
0	94		
5	105		
Name: co2, dtype: int64			

	volume	weight	
2	2.2	1280	x_test
8	1.5	1112	
4	1.5	1140	
2	104		y_test
8	98		
4	105		
Name: co2, dtype: int64			



# (4) Data Splitting

```
from sklearn.model_selection import train_test_split
import pandas as pd
```

```
df = pd.read_csv('test-standard-scaler.csv')
x = df.iloc[:, 2:4] #df[['volume', 'weight']]
y = df['co2']

x_train, _, _, _ = train_test_split(x, y, random_state=0)
display(x_train)
print('random_state = 0')

x_train, _, _, _ = train_test_split(x, y, random_state=10)
display(x_train)
print('random_state = 10')

x_train, _, _, _ = train_test_split(x, y, random_state=100)
display(x_train)
print('random_state = 100')
```

	volume	weight
9	2.0	1746
1	1.3	990
6	2.1	1365
7	2.1	1605
3	1.5	1465
0	1.6	1252
5	2.0	1328
random_state = 0		

	volume	weight
6	2.1	1365
3	1.5	1465
1	1.3	990
0	1.6	1252
7	2.1	1605
4	1.5	1140
9	2.0	1746
random_state = 10		

	volume	weight
5	2.0	1328
4	1.5	1140
2	2.2	1280
0	1.6	1252
3	1.5	1465
9	2.0	1746
8	1.5	1112
random_state = 100		

# Example: Data preparation

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import matplotlib.pyplot as plt

# 1. Generate Sample Data with Missing Values and Categorical Features
np.random.seed(42) # For reproducibility
data = {
    'Feature1': np.random.randn(100),
    'Feature2': np.random.randint(1, 10, 100),
    'Feature3': np.random.choice(['A', 'B', 'C'], 100),
    'Target': np.random.randint(0, 2, 100) # Binary target
}

# Introduce some missing values
data['Feature1'][np.random.choice(100, 10, replace=False)] = np.nan

# Create DataFrame
df = pd.DataFrame(data)

# 2. Data Cleaning (Handling Missing Values)
# Replace missing values with the mean of the column
df['Feature1'].fillna(df['Feature1'].mean(), inplace=True)
```

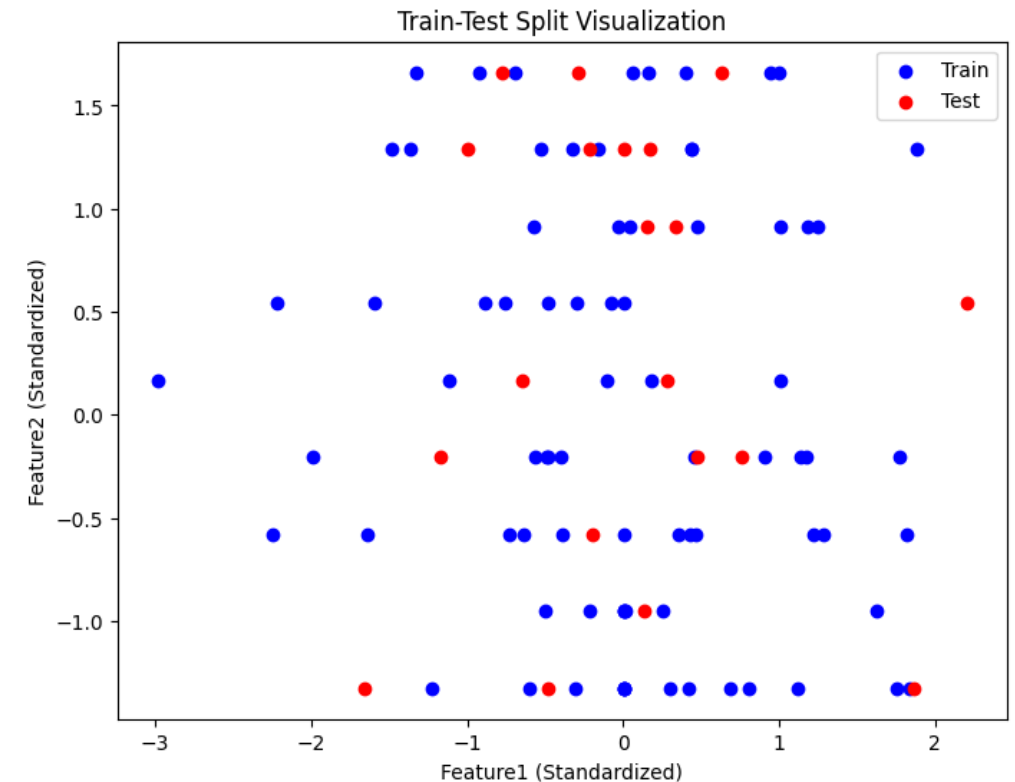
# Example: Data preparation

```
# 3. Data Encoding (Encode Categorical Feature)
le = LabelEncoder()
df['Feature3_Encoded'] = le.fit_transform(df['Feature3'])

# 4. Data Splitting
X = df[['Feature1', 'Feature2', 'Feature3_Encoded']]
y = df['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 5. Data Scaling (Standardize Features)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 6. Plot Data (Scatter Plot with Different Colors for Train/Test Sets)
plt.figure(figsize=(8, 6))
plt.scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], c='blue',
label='Train')
plt.scatter(X_test_scaled[:, 0], X_test_scaled[:, 1], c='red', label='Test')
plt.xlabel('Feature1 (Standardized)')
plt.ylabel('Feature2 (Standardized)')
plt.legend()
plt.title('Train-Test Split Visualization')
plt.show()
```



# Assignment

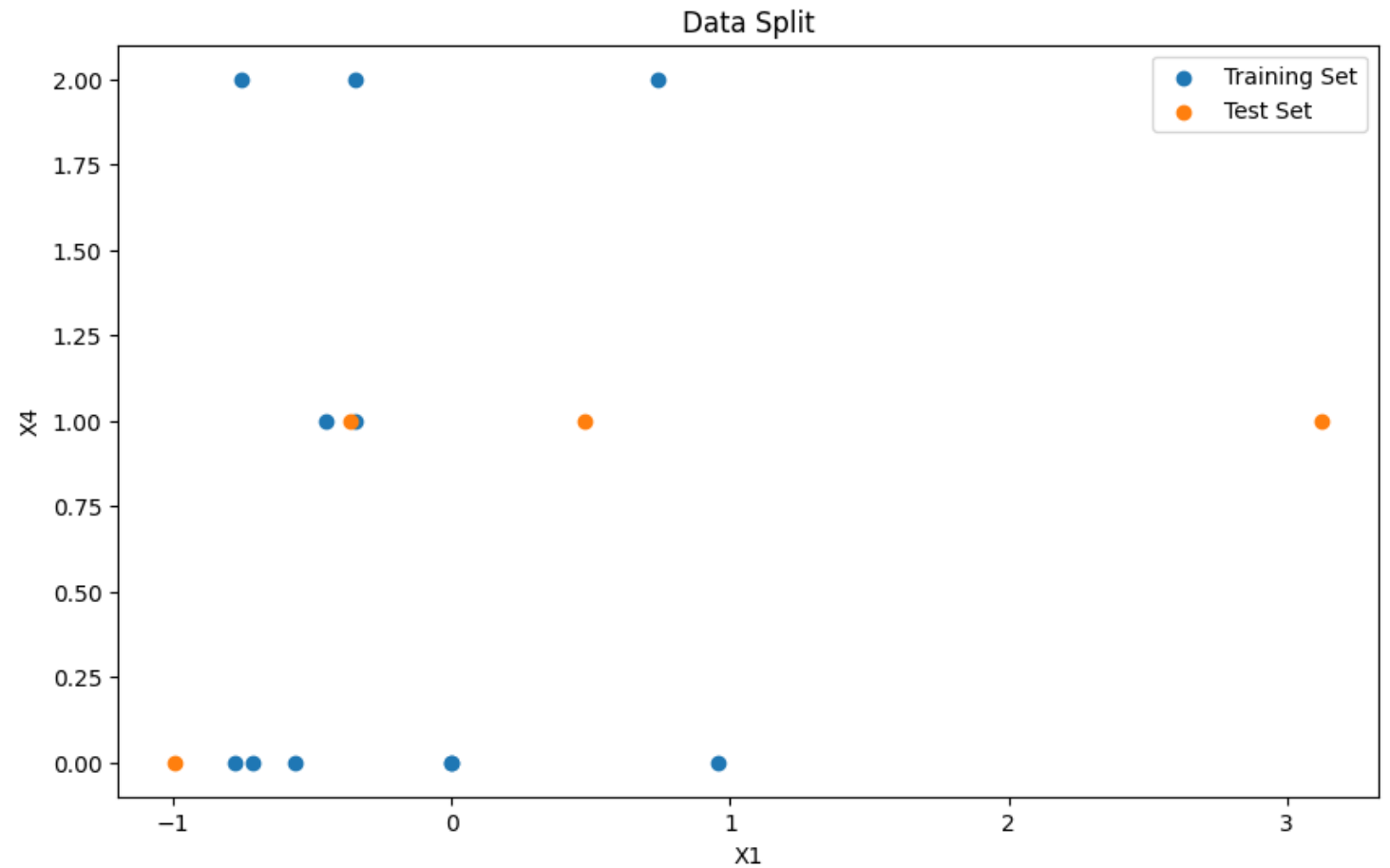
From raw data below:

1. Replace missing value with mean
2. Encode-transform and scale data
3. Split data with training set and test set = (75:25)
4. Show your table data and data points of X1 and X4 (training set and test set with difference color)

Machine	X1	X2	X3	X4	Fails (Y/N)
A	1000	2	10.5	AA	N
A		1	25.25	AA	N
A	2300	5	30.0	AA	Y
A	4000	5	90.001	DD	N
A	2101	3	0.005	DD	N
B	3000	6	20.201	AA	N
B	3500	2	10.10	BB	N
B	4000			BB	N
C		7	40	AA	N
C	3900		50.5	BB	N
C	2000	3	30.3	AA	N
C	20000	15	3.333	BB	Y
D	10000	8	7.6667	AA	Y
D	7800	9	0.125	BB	Y
D	9000	12	1.50	DD	Y

# Assignment

	X1	X2	X3	X4_encoded
0	-0.993314	-1.074172	-0.529913	0
1	0.000000	-1.342715	0.104514	0
2	-0.711657	-0.268543	0.308821	0
3	-0.343337	-0.268543	2.889584	2
4	-0.754772	-0.805629	-0.981324	2
5	-0.559996	0.000000	-0.112654	0
6	-0.451667	-1.074172	-0.547118	1
7	-0.343337	0.000000	0.000000	1
8	0.000000	0.268543	0.738941	0
9	-0.365003	0.000000	1.190567	1
10	-0.776655	-0.805629	0.321725	0
11	3.123203	2.416888	-0.838180	1
12	0.956615	0.537086	-0.651779	0
13	0.479966	0.805629	-0.976163	1
14	0.739956	1.611258	-0.917021	2





INSTITUTE OF  
**ENGINEERING**

**Mechatronics** Modern Automotive