Machine learning

↓

Brach of AI. where computers learn patterns from historical data & make predictions / decision without being explicitly programmed.

* Supervised learning =
  ↳ Models learns from labeled training data. (Input - output paired data)

Examples = Email spam Detection.

Step 1 = Data collection

Step 2 = Training

Step 3 = Testing

Algorithm

classification
↓
To predict discrete (categorical) values
↓
output Type
↓
Yes/No, 0/1, Pass/fail
↓
Examples
↓
prediction of
↓
email ⇒ spam
Result = pass/fail
Images = Animal/Bird.

Regression.
↓
To predict continuous values.
↓
output Types.
45.6 (temp)
85.3 (marks)
150000 (prices)
↓
Examples
↓
prediction of
↓
house price
Salary, temperature
sales revenue.

* Unsupervised learning -
Models learns from unlabeled data.
+
only Input.

Examples: Customer Segmentation.
Patient Grouping.

Clusting clustering
↓
Group similar data

Association
↓
find relationships.

Algorithms = K-means clustering
Hierarchical clustering
Apriori Algorithm
PCA.

* Reinforcement learning -
Models learns by trial & error using rewards & penalties =

Agent → learner
Environment → World
Rewards → Feedback.

* ML Pipeline : Complete workflow
Raw data → deployed Models.

1] Data collection :
From → Databases. APIs. Csv/Excel files.
Sensors. Web scraping.

2] Data cleaning -
Raw Data
↳ Missing values. Duplicates. outliers,
Incorrect entries.

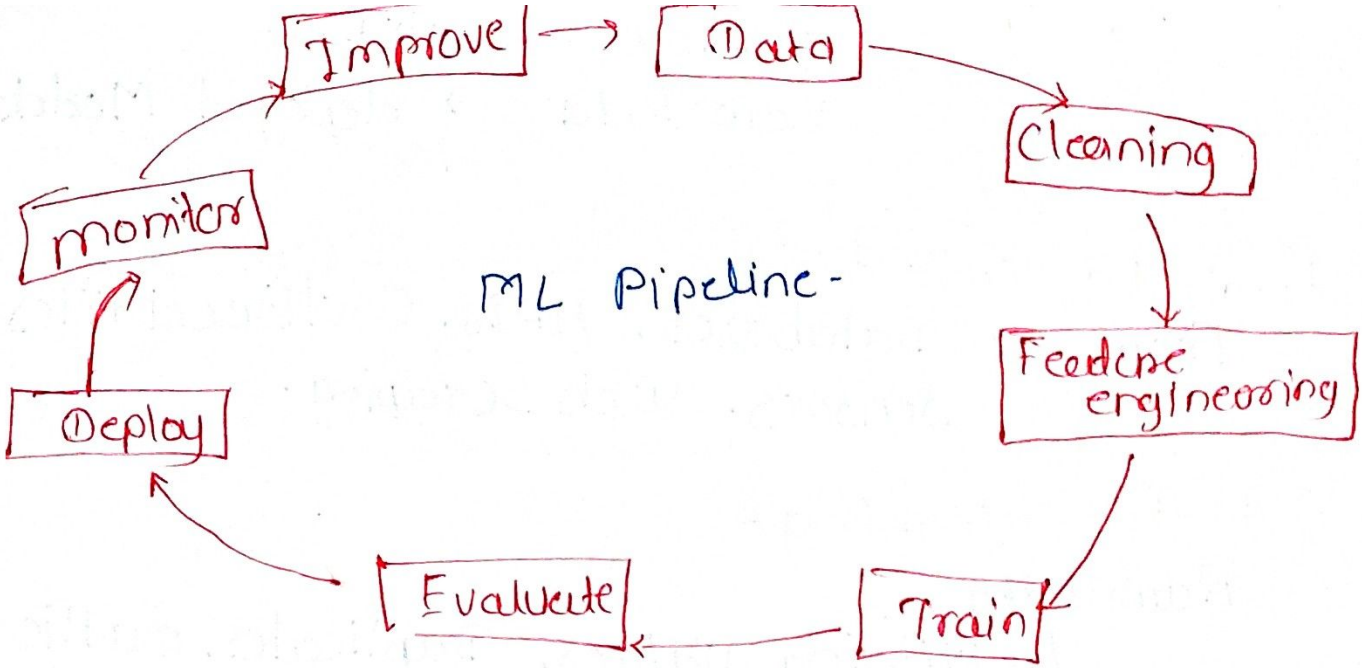
Steps →
↳ Handles Null values. outliers detection
Remove duplicates. fix inconsistent formats.

3] Feature Engineering.
Convert raw data into useful features.
Ex → data → Month. year.
Female / male → 0/1

4] Model training -
Training Set → 70-80%.
Testing Set → 30-20%.

5] Model Evaluation ⇒
↳ checking performance using metrics.

7] Deployment →
Make models available for real-world use.



Numpy Vs list

1] Size = Numpy ds take up less space.

2] Performance = They have a need for speed & are faster than lists.

3] Functionality = Scipy & Numpy have optimized functions such as liner algebra operations built in.

* Numpy = python libary used for numerical computing-

Provides → i) Fast Mathematical operations
ii) Multi-dimensional arrays
iii) linear Algebra functions
iv) Statistical operation.

Numpy Array	Python list
i) All elements of an array are of same data types.	list can have elements of different data types.
ii) Stored in continuous memory location.	Data not store in continuous memory location.
iii) Arrays are immutable.	Lists are mutable.
iv) Arrays support element wise operations.	list do not support element wise operation
v) Numpy array takes up less space in memory.	More space in memory.

* Numpy array -
  i) Main object of ~~arr~~ numpy
  ii) It also called ndarray.
  iii) Faster, Memory efficient.
  iv) Support mathematical operations directly.

* Use.

```
import Numpy as np.

arr = np.array([10,20,30, 40])
print(array)
arr2 = np.array([[10,20],[30,40]) #2D array
```

Array properties =

```
print (arr.ndim)   # Number of dimensions
print (arr.shape)  # shape of array
print (arr.size)   # Total elements
print (arr.dtype)  # Data type.
```

Numpy array

```
a=np.array([1,2,3])
b=np.array([4,5,6])
print(a+b)
# [5,7,9]
```

python list:

```
list1=[1,2,3]
list2=[4,5,6
print(list1+list2)
# [1,2,3,4,5,6]
```

Index & slicing:

```
Print(arr[0]) # first element
prin(arr[-1]) # last element
print(arr[1:4] # slicing.
```

Special Arrays:

```
np.Zeros((2,3))
np.onece((3,3))
np.eyes(3)  ⟶ # identity array [[1,0,0],[0,1,0],[0,0,1]]
np.arrang(0,10,2) → [0,2,4,6,8]
np.linespace(0,1,5) → [0. 0.25 0.5 0.75 1.]
```

Reshaping:

```
a= np.arrange(6)
print(a.reshape(2,3)
```

Arithmatic operation on numpy arrays.

Note ⇒ size of both are should be same.

```
arr1 = np.array([2,1])
arr2 = np.array([3,4])
```

1] Addition :

```
print(arr1+arr2)
```

→ ([5 5])

2] Addition :

```
print(arr2-arr1)
```

⇒ ([1, 3])

3] Multiplication :

```
print(arr1*arr2)
```

[6, 4]

4] Matrix Multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x & y \\ z & w \end{bmatrix}$$

$$M.M = \begin{bmatrix} a*x + b*z & a*y + b*w \\ c*x + d*z & c*y + d*w \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$$= \begin{bmatrix} 5+14 & 6+16 \\ 15+28 & 18+32 \end{bmatrix}$$

$$= \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

5] Division ⇒

```
print(arr2/arr1)
```

⇒ [1.5, 4]

6] Floor Division

```
print(arr2//arr1)
```

⇒ ([1 4])

7] Exponsion

```
print(arr2**arr1)
```

$= [3^2, 4^1]$

$= [27, 4]$

Matrix multiplication methods

i) result = np.dot(arr1, arr2)

ii) result = arr1 @ arr2

iii) result = np.matmul(arr1, arr2)

Axis concept:

In numpy, the axis concept tells us along which direction an operation is performed.

eg. arr = np.array([[1,2,3],[4,5,6]])

1] Axis	Direction.
axis = 0	column wise
axis = 1	Row wise.

e.g. np.sum(arr, axis = 0)

⇒ [1+4, 2+5, 3+6] → column wise.

= [5, 7, 9]

np.sum(arr, axis = 1)

[1+2+3, 4+5, 6] → Row wise.

= [6, 15]

Broadcasting:

arr1 : [1, 2, 3] 1x3 → 1 Row, 3 colum.

arr1 : [4, 5, 6] 3x1 → 3 Row, 1 column.

→ Issue.

1	2	3
1	2	3
1	2	3

+

4	4	4
5	5	5
6	6	6

$$= \begin{bmatrix} 5 & 6 & 7 \\ 6 & 7 & 8 \\ 7 & 8 & 9 \end{bmatrix}$$

|| Pandas ||

↓

data analysis & data manipulation:

Some important methods:

```
import pandas as pd
data = {"Name": ['A', 'B', 'C', 'D', 'E'],
        "Marks": [80, 75, 95, 90, 76]}

df = pd.DataFrame(data)
```

1] head() ⇒ Returns First n rows (default 5)

→ df.head(3) ⇒ 3 rows.

2] tail ⇒ Return last n rows (default 5)

→ df.tail(3) ⇒ last 3 rows.

3] info() ⇒ Gives summary of dataframe (columns, data type null values)

→ df.info.

4] describe() ⇒ Gives statistical summary (mean, std, min, max)

df.describe()

5] ~~Returns the~~ shape() ⇒ Returns number of rows & columns.

df.shape() ⇒ o/p = (5, 2)

6] columns() ⇒ shows column names

df.columns

7] dtypes => shows data types of columns.

8] df.dtypes.

8] isnull → checking missing values.

df.isnull()

df.isnull().sum() ⟹ To count null values.

9] df.drop() ⇒ Drops row / column

df.drop('Marks', axis=1) = col. column

df.drop('Marks', axis=0) ⇒ Row.

10] fillna() ⇒ Fills missing values

df['Marks'].fillna(0)

11] sort_values() ⇒ sort data frame.

df.sort_values(by='Marks', ascending=False)

12] groupby() ⇒ Groups data based on column.

df.groupby('Name')['Marks'].mean()

13] value_counts() ⇒ counts frequency of values

df['Marks'].value_counts()

14] loc[] ⇒ select data using labels.

⇒ df.loc[0]

15] iloc[] ⇒ select data using index position

df.iloc[0:2]

Data Visualization =

Representing data in graphical form

→ Matplotlib → Basic plotting library.
→ Seaborn → statistical visualization =

MatPlotlib =

use of cond^n ⇒ i) Basic plotting
(Ideal) ii) custom visualization
iii) Research papers
iv) Interview coding rounds.
v) Foundation for Seaborn =

Types =

```
import matplotlib.pyplot as plt
x = [1,2,3,4] , y = [10,20,30,40]
plt.plot(x,y)          # plot
plt.title(simple line plot)    # labels
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.show()             # show.
```

1] Line plot = → used for trends over time.

```
plt.plot(x,y)
plt.show()
```

2] Bar chart = → used for category comparison

```
plt.bar(x,y)
plt.show()
```

3] Histogram = Used for frequency distribution-

```
import numpy as np
data = np.random.randn(1000)
plt.hist(data, bins=30)
plt.show()
```

5] Scatter plot = Used to show relationship between two variables-

```
plt.scatter(x,y)
plt.show()
```

6] Pie chart =

```
sizes = [40, 30, 20, 10]
labels = ['A', 'B', 'C', 'D']
plt.pie(sizes, labels=labels,
        autopct='%1.1f%%')
plt.show()
```

* Figure, Axes ⇒

```
fig, axe = plt.subplots()
axe.plot(x,y)
plt.show()
```

⇒ Figure = Entire canvas
Axes = Actual plot Area.

* Multiple plots =

```
fig, axe = plt.subplots(1,2)
axe[0].plot(x,y)
axe[1].bar(['A','B','C'], [5,7,3])
plt.show().
```