Numpy

Numerical Python (NUMPY) is a Powerful Python library for numerical computing, especially when working with large arrays and matrices of numbers.

→ import numpy as np

ar = np.array([1,3,5,7,9]) one dimension
ar
//array([1,3,5,7,9])

ar1 = np.array([[1,3,5,7,9],[2,4,6,8,10]]) two dimension (like matrix)

ar2 = np.array([[[1,2],[3,4]],[[5,6],[7,8]]]) 3d array

Eg :-
ar = np.array([1,3,5,7,9])
ar
//array([1,3,5,7,9])
ar.ndim
//1

→ to change the dimension

ar = np.array([1,3,5,7,9], ndmin=3)
ar
//array([[[1,3,5,7,9]]])
ar.ndim
//3

Numpy Attributes

np.dtype → datatype
np.shape → give the sape (no of rows, no of colums)
np.itemsize → How much memory the item is consuderng
np.ndim → dimension
np.nbytes → total memory consume by all the elements

Example,

```
import numpy as np
ar1 = np.array([1,3,5,7,9])
ar2 = np.array([[1,2,3],[4,5,6]])
ar3 = np.array([[[1,1],[2,2]],[[3,3],[6,4]]])
ar4 = np.array([1,2,3,4], ndmin=4)

ar1.ndim
// 1
ar3.ndim
// 3
ar3.shape
// (2,1,2)
ar4.shape
// (1,1,1,4)
ar1.dtype
// dtype('int32')
ar1.itemsize
// 4
ar1.nbytes
// 20
ar5 = np.array([1.1, 2.2, 3.3])
ar5.dtype
// dtype('float64')
```

Creating an Array

np.zeros(3) → create array with zeros.
np.ones(3) → create array with ones.
np.empty(3) → create an array but elements may be anything (garbage value)
np.eye(3) → creating an identity matrix (square matrix) 3x3, 4x4 ...
np.diag(3) → // creats a square matrix
np.arrange(start, end, step) → Same as range function of Python
np.linspace(0,10, num=5) → // creats an array with 'num=' elements
(start, end)

```
import numpy as np
ar = np.zeros(3)
ar
// array([0., 0., 0.])
ar.dtype
// dtype('float64')

ar = np.zeros((3,4))
ar
// array([[0., 0., 0., 0.],
          [0., 0., 0., 0.],
          [0., 0., 0., 0.]])
```

```
ar = np.ones(3)
ar
// array([1., 1., 1.])
ar = np.empty((3,5))
ar
// array([[1.40526e-311, 3.55711058e-312, 0.00000e+001
        0.00000e+000, 0.00000e+000],
       [1.50008929e-248, 4.31185390e-076, 9.00641896e+252,
        1.29316688e+224, 1.05118426e-156],
       [9.08242391e+271, 9.28675431e-223, 1.06346660e+153,
        3.45551369e-175, 6.81076626e-310]])

ar = np.eye(4)
ar
// array([[1., 0., 0., 0.],          → identity matrix
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])

ar = np.eye((3,5))
ar
// array([[1., 0., 0., 0., 0.],      → It is not identity matrix
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.]])
```

```
ar = np.diag([2,4,5,6])          → ( ) also use
ar
// array([[2, 0, 0, 0],
          [0, 4, 0, 0],
          [0, 0, 5, 0],
          [0, 0, 0, 6]])

ar = np.arange(0,10)
ar
// array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

ar = np.arange(0,10,2)
ar
// array([0, 2, 4, 6, 8])

ar = np.arange(-10, 0, 2)
ar
// array([-10, -8, -6, -4, -2])

ar = np.linspace(0, 10, 5)
ar
// array([0, 2.5, 5, 7.5, 10])

// It will create an array with 5 elements
0   2.5   5   7.5   10
  d    d    d    difference

end - start / num - 1 = 10/4 = 2.5
```

```
ar = np.linspace(5,12,6)
ar
// array([5. , 6.4, 7.8, 9.2, 10.6, 12. ])
```

Array Indexing, Slicing

```
import numpy as np

ar1 = np.array([2,4,6,8,10,12,14,16,18,20])
ar2 = np.array([[2,4,6,8],[1,3,5,7],[11,22,33,44],[10,20,30,40]])

ar1[4]
//10
ar1[-4]
//14
ar2[2,3]
//44
ar2[-4,-3]
//-4
```

Slicing

```
arr[start, end : steps]

ar1[2:9]
ar
// array([6,8,10,12,14,16,18])

ar1[0:10:2]
ar1
// array([2,6,10,14,18])

ar1[1:8:3]
ar1
// array([4,10,16])
```

2-d

```
ar2[0]
// array([2,4,6,8])

ar2[0:2]
// array([[2, 4, 6, 8],
          [1, 3, 5, 7]])

ar2[0:3:2]
// array([[2, 4, 6, 8],
          [11,22,33,44]])

arr2[:, 0:2]
// array([[2, 4],
          [1, 3],
          [11,22],
          [10,20]])
```

```
ar2[0:2, 2:4]
// array([[6,8],
          [5,7]])

ar2[0:4:2, 1:4,2]
// array([[3,7],
          [20,40]])
```

Numpy Reshape

→ np.reshape()

1D to 2D
1D to 3D
2D to 3D

```
import numpy as np
ar = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
ar.shape
//(12,)
ar.reshape(3,4)
// array([[1,2,3,4],
          [5,6,7,8],
          [9,10,11,12]])

ar = ar.reshape(4,3)
ar
//array([[1,2,3],
         [4,5,6],
         [7,8,9],
         [10,11,12]])
```

```
ar = ar.reshape(3,2,2)
ar
//array([[[1,2],
           [3,4]],

          [[5,6],
           [7,8]],

          [[9,10],
           [11,12]]])
```

ar.flatten()

This function is used to convert any array into 1D.

```
ar.flatten()
//array([1,2,3,4,5,6,7,8,9,10,11,12])
```

Numpy Datatypes

b - byte
B - unsigned byte
i - integer
I - Unsigned integer
f - Float
F - Complex Float

m - timedelta
M - Datetime
O - Object
S - String
U - Unicode String
? - boolean

-128 to 127

```
import numpy as np
ar = np.array([1,2,3,4], 'b')
ar.dtype
// dtype('int8')

ar = np.array([1,2,3,127], 'b')
ar
// array([1, 2, 3, 127], dtype=int8)
ar = np.array([1,2,3,129], 'b')
ar
// array([1, 2, 3, -127], dtype=int8)
```

Unsigned

Only allow Positive numbers
Range (0-255) → only

```
ar = np.array([1,2,3,4], 'B')
ar.dtype
// dtype('uint8')

ar = np.array([1,2,3,256], 'B')
ar
// array([1, 2, 3, 0], dtype=uint8)
```

```
ar = np.array([1,2,3,4], 'i')
ar.dtype
// dtype('int32')

ar = np.array([1,2,3,4], 'I')
ar.dtype
// dtype('uint32')

ar = np.array([1,2,3,4], 'f')
ar.dtype
// dtype('float32')

ar = np.array([1,2,3,4], 'F')
ar.dtype
// dtype('complex64')
ar
// array([1.+0.j, 2.+0.j, 3.+0.j, 4.+0.j], dtype=complex64)

ar = np.array([1,2,3,4], 'O')
ar.dtype
// dtype('O')
ar
// array([1, 2, 3, 4], dtype=object)
```

```
ar=np.array([1,2,3,4], 's')
ar.dtype
// dtype('S1')
ar
// array([b'1', b'2', b'3', b'4'], dtype='S1')

ar=np.array(['a','b','c','d'], 'U')
ar.dtype
// dtype('<U1')
ar
// array(['a', 'b', 'c', 'd'], dtype='<U1')

ar=np.array([1,2,3,0], 'bool')
ar.dtype
// dtype('bool')
ar
// array([True, True, True, False])
```

→ Data Ingestion

* It is the process of gathering, managing, and utilizing data efficiently
* Plays a foundational step in the data processing pipeline.
* It involves the careful extraction, transfer, or loading of raw data from diverse external sources into a centralized system or storage infrastructure, where it awaits further processing and analysis.

Key steps in the ingestion process

1) Data collection - Gather raw data from various sources
Data Transformation - clean, normalize, and enrich the data
Data Loading - move the transformed data to target data

Type	Description	Use cases
* Batch	Data is collected and processed at scheduled intervals	Daily reports, Payroll backups
* Real-time	Data is ingested as it's generated, enabling instant processing	Fraud detection, live dashboards

* micro batching — A hybrid approach - small batches processed frequently — IoT data, semi live, analytics

The Data ingestion workflow

* Data source identification - Identify and register the data source. understand the data format, structure and access method
* Data extraction : Extract data from identified source using connectors, APIs, or other methods, ensure the data is collected efficiently and securely.
* Data Staging
* Data validation
* Data Transformation
* Data loading
* Data monitoring

→ Batch Processing

Data collection: Data from various sources (databases, logs, files) is collected

Characterising
latency, Data size, Processing, scheduling, Fault tolerance, Cost

Example of Batch Processing

1) Payroll Systems
2) Bank Statement Generation
3) E-commerce Order Processing
4) Credit Card Billing

Adv

Efficiency at Scale
Automated scheduling
Error handling and Retry Logic
Resource optimization
Reduced manual intervention
Consistency and standardization

Limitation

Delayed Results
Resource Spikes
Complex Debugging
Limited Flexibility
Data Staleness

Architecture

Data Sources
(Databases, Logs, CSV)
↓
Data Ingestion
(Scheduled Jobs)
↓
Storage Layer
(HDFS, S3, etc)
↓
Batch Processing
(Spark, Hadoop)
↓
Processed Data
(DW, OBS, Reports)
↓
BI Tool Reports
(Tableau, Excel)

Common Batch Processing Tool

Apache Spark
Apache Hadoop
Talend
AWS Glue

Stream Processing

Data Ingestion - Data is continuously ingested from sources like Kafka, Sensors, Logs, Assets.

Output delivery - Result are pushed immediately to databases, dashboards, alerting systems, or APIs.

Characteristics

Latency, Data size, Processing, Use case, Fault Tolerance, Cost.

Example

1) Fraud detection in Banking
2) Real-Time Ride matching (uber, ola)
3) Real-Time Analytics for E-commerce
4) Social Media Feeds and Notifications.
5) Log Monitoring and Alerting
6) Online Gaming Leaderboards.
7) Smart Traffic Systems

Architecture

Continuous Data Sources

↓

Data Ingestion

↓

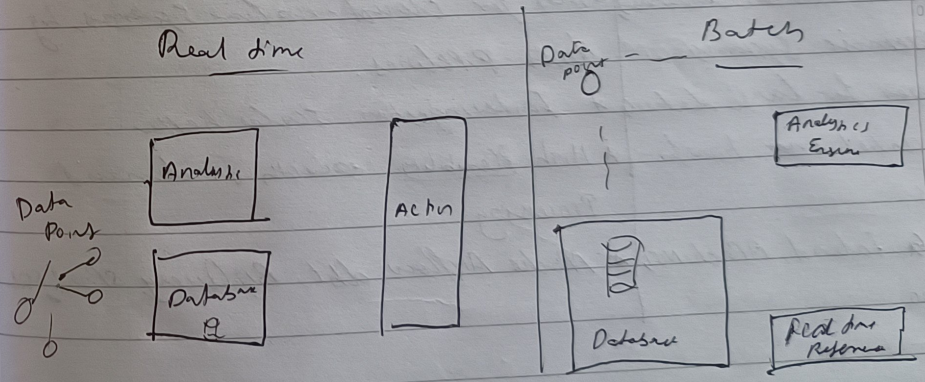
Data Processing Engine

↓ ↓

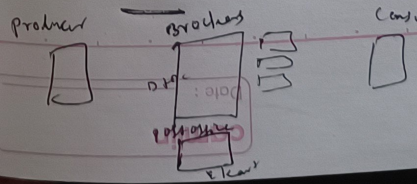
Alerting System (Slack, Email) | Real time Dashboards (

↓

Downstream DB (MySQL, DW)



Kafka (Support Stream Processing)



Kafka Architecture

Producer

Consumer

Brokers

TOPIC

Partitions

ETL (Extract → Transform → Load)	ELT (Extract → Load → Transform)
* Data is extracted, transformed (cleaned, enriched), then loaded into the target system	* Data is extracted and directly loaded into the target system where transformation happens.
* Transforms happens in a staging or ETL tool	* Transform happens in the target system (typically a modern data warehouse like BigQuery, Redshift, Snowflake.
* Best for traditional databases, on-Premise systems	* Best for cloud-native systems, big data Pipelines.
* Slower for large datasets	* Faster ingestion, scales better
* High control over transformations	* High flexibility with on-demand Processing
* Informatica, Talend, Apache NiFi, pentaho	* Apache Airflow, dbt, BigQuery, Spark SQL

ETL

[Source system]
↓ Extract
[Staging Area or ETL Tool]
↓ Transform (clean, bin, enrich)
[Data warehouse or Database] + Load

ELT

[Source systems]
↓ Extract
[cloud Data warehouse] + load
↓ Transform (using SQL/scripts on warehouse)
[Final structured Data]
↓
To store structured data

ETL use cases

* Banking and Finance
* Healthcare
* Retail POS systems
* Insurance

ELT use case

* Streaming Analytics/IoT
* E-commerce
* Social Media Analytics
* Real-Time Fraud Detection

SQL

Database

It is an organized collection of structured data
Eg: folder in a computer

-> CREATE DATABASE databasename;
-> SHOW DATBASES;
-> USE databasename;
-> CREATE TABLE Products (Product_id INT, Product_name VARCHAR(50), Price FLOAT);
-> SHOW TABLES;
-> INSERT INTO Products (Product_id, Product_name, Price)
VALUES (1, "Iphone 13 Pro", 120000),
(2, 'Iphone 13', 60000);
-> SELECT * FROM Products;
-> SELECT Product_name FROM Products;
-> SELECT Product_name, Price FROM Products;

Modify

-> ALTER TABLE Products RENAME TO Products_new;
-> ALTER TABLE Product_new RENAME COLUMN Price TO Price_new;
-> SELECT * FROM Products_new;

Adding a column

-> ALTER TABLE Products_new ADD brand VARCHAR(40);
-> Brand is null now, so, update it.
-> UPDATE Product_new
SET brand = 'APPLE'
WHERE Product_id = ;
-> DROP TABLE Product_new;
-> DROP DATABASE databasename;

CREATE DATBASE Mobiles;
USE Mobiles;
CREATE TABLE Products (Product_id INT, Product_name VARCHAR (50), Price FLOAT, brand VARCHAR(50));
INSERT INTO Products VALUES ( ), ( ), ( )

(Total 7 rows)

-> SELECT * FROM Products;
-> SELECT Product_name, brand FROM Products;
-> SELECT * FROM Products WHERE Price > 50000;
-> SELECT Product_name FROM Products WHERE brand = 'APPLE';
-> SELECT Product_name, Price FROM Products
WHERE price BETWEEN 150000 AND 200000

-> SELECT brand FROM Products WHERE brand LIKE 'A%';

-> SELECT brand FROM Products WHERE brand LIKE '%E' (last letter)

-> SELECT brand FROM Product WHERE brand LIKE '_P%' (second letter)

-> " " " WHERE brand LIKE '%P%'

-> SELECT * FROM Products WHERE price LIKE '1%';

-> SELECT * FROM Products WHERE brand IN ('Redmi', 'Asus', 'Vivo');

-> SELECT brand FROM Products;

to find unique values,

-> SELECT DISTINCT(brand) FROM Products;

AND, OR, NOT

-> SELECT * FROM Products WHERE Price > 100000 AND Product_id < 10;

-> SELECT * FROM Products WHERE Price > 100000 OR Product_id < 10;

-> SELECT * FROM Products WHERE Product_id < 10;

-> " " " WHERE NOT Product_id < 10;

Aggregate function

Count	MIN
SUM	MAX
AVG	

-> SELECT COUNT(*) FROM Products;

-> SELECT SUM(Price) FROM Products;

-> SELECT AVG(Price) FROM Products;

-> SELECT MIN(Price) FROM Products;

-> SELECT MAX(Price) FROM Products;

ORDER BY

-> SELECT * FROM Products ORDER BY brand Asc;

-> " " " Desc;

GROUP BY

-> SELECT brand FROM Products GROUP BY brand;

-> SELECT brand, COUNT(*) FROM Products GROUP BY brand;

-> SELECT brand, COUNT(*) FROM Products GROUP BY brand HAVING brand = "Apple"; (Don't use where)

LIMIT

-> SELECT * FROM Products LIMIT 10;

-> SELECT * FROM Products LIMIT 10 OFFSET 5

Joins

Product_id	Product_name	Price	brand
1			
2			
3			
4			
5			

Table 2

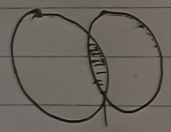
Specifications

→ Foreign key

ID	RAM	ROM
1	6	128
1	8	128
2	16	256
2	8	128
6	6	64
7	4	64
7	8	128

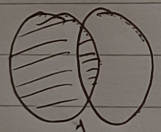
Types of Join

Inner Join



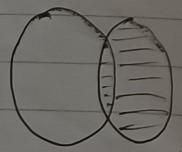
Inner Join

Left Join



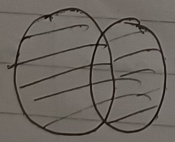
Left Joint

Right Join



Right Joint

Full Joint



camlin

Date :

→ SELECT Products.Product_id, Products.Product_name, Specifications.RAM, Specifications.ROM FROM Products JOIN Specifications ON Products.Product_id = Specifications.ID;

→ LEFT JOIN, RIGHT JOIN, FULL JOIN

Full Join →

→ SELECT Products.Product_id, Products.Product_name, Specifications.RAM, Specifications.ROM FROM Products LEFT JOIN Specifications ON Product.Product_id = Specification.ID

UNION ALL

SELECT Products.Product_id, Products.Product_name, Specifications.RAM, Specifications.ROM FROM Products RIGHT JOIN Specifications ON Product.Product_id = Specification.ID;

a) Get RAM and ROM details of Galaxy A34

Specifications Products

→ SELECT RAM, ROM FROM Specifications WHERE ID IN (SELECT Product_id FROM WHERE Product_name = 'Galaxy A34');

First run subquery then Outer Query

camlin

Date :