Q. What are the different file types used in big data for data processing. Describe and consume each one of them.

1. CSV (Comma-Separated Values)

- **Description**: A plain text format where data is separated by commas.
- **Cons:** No support for complex structures like nested data or metadata.
- **Usage**:
  - Easy to read and write.
  - Commonly used for small to medium datasets.
  - Supported by most tools like Excel, Python, and SQL.
- **Consumption**:
  - Use libraries like Pandas (Python) or Spark to read and process.
  - Not efficient for large datasets due to lack of compression and schema support.
- **Example:**

```
Name, Age, City

Alice, 25, New York

Bob, 30, London
```

2. JSON (JavaScript Object Notation)

- **Description**: A lightweight, human-readable format for storing and exchanging data in key-value pairs.
- **Cons:** Can be inefficient for very large datasets due to size.
- **Usage**:
  - Ideal for semi-structured data.
  - Commonly used in web APIs and NoSQL databases.
- **Consumption**:
  - Use libraries like `json` in Python or Spark's JSON reader.
  - Slower to parse compared to binary formats.

- **Example:**

  ```
  {
    "Name": "Alice",
    "Age": 25,
    "City": "New York"
  }
  ```

3. Parquet

- **Description**: A columnar storage format optimized for big data processing.
- **Cons:** Not human-readable and requires specialized tools.
- **Usage**:
  - Highly efficient for querying large datasets.
  - Supports compression and schema evolution.
  - Commonly used in Hadoop and Spark ecosystems.
- **Consumption**:
  - Use tools like Apache Spark, Apache Hive, or Pandas to read and write.
  - Faster for analytical queries due to columnar storage.
- **Example:** `Stores data column-wise for faster read access.`

4. ORC (Optimized Row Columnar)

- **Description**: Another columnar storage format designed for Hadoop workloads.
- **Cons:** Limited support outside the Hadoop ecosystem.
- **Usage**:
  - Provides high compression and fast query performance.
  - Supports ACID transactions in Hive.
- **Consumption**:
  - Use Hive, Spark, or Presto to process ORC files.
  - Ideal for large-scale data warehousing.
- **Example:** `Combines row groups and columnar data storage.`

5. Avro

- **Description**: A row-based binary format with schema support.
- **Cons:** Requires schema compatibility between writer and reader.
- **Usage**:
  - Great for serialization and deserialization.
  - Supports schema evolution (adding/removing fields).
  - Commonly used in Kafka and Hadoop ecosystems.
- **Consumption**:

    - Use libraries like Apache Avro or Spark to read and write.
    - Efficient for data storage and transfer.
- **Example:** `Stores binary data but uses JSON for metadata.`

6. SequenceFile

- **Description**: A flat file format for storing binary key-value pairs.
- **Cons:** Requires Hadoop tools to read/write.
- **Usage**:
    - Used in Hadoop for intermediate data storage.
    - Supports compression and splitting.
- **Consumption**:
    - Use Hadoop MapReduce or Spark to process.
    - Not human-readable.
- **Example:** `Stores data in binary format like [key1, value1].`

7. XML (eXtensible Markup Language)

- **Description**: A markup language for storing structured data in a human-readable format.
- **Cons:** Verbose, leading to large file sizes.
- **Usage**:
    - Commonly used in web services and legacy systems.
    - Verbose and less efficient compared to JSON.
- **Consumption**:
    - Use libraries like `xml.etree.ElementTree` in Python or Spark's XML reader.
    - Not ideal for big data due to large file sizes.
- **Example:**

```
<Person>

  <Name>Alice</Name>

  <Age>25</Age>

  <City>New York</City>

</Person>
```

8. TSV (Tab-Separated Values)

- **Description**: Similar to CSV but uses tabs as delimiters.
- **Cons:** If the data itself contains tabs, it can cause parsing issues, as tabs are used as the delimiter.

- **Usage**:
  - Used when data contains commas.
  - Simple and easy to parse.
- **Consumption**:
  - Use tools like Pandas or Spark to process.
  - Similar limitations to CSV for big data.
- **Example:** `Data export/import for text-heavy datasets.`

9. HDF5 (Hierarchical Data Format)

- **Description**: A file format for storing large and complex scientific data.
- **Cons:** Requires special libraries to process.
- **Usage**:
  - Used in scientific computing and machine learning.
  - Supports hierarchical data organization.
- **Consumption**:
  - Use libraries like `h5py` in Python or specialized tools for scientific data.
- **Example:** `Stores arrays, tables, and metadata efficiently.`

10. Feather

- **Description**: A lightweight, fast binary format for data frames.
- **Cons:** Not suitable for extremely large datasets.
- **Usage**:
  - Designed for high-speed data exchange between Python and R.
  - Not as efficient as Parquet for long-term storage.
- **Consumption**:
  - Use libraries like Pandas or Arrow to read and write.
- **Example:** `Efficient sharing of data between Python and R applications.`

11. RCFile (Record Columnar File)

- **Description**: A hybrid row-columnar format for Hadoop.
- **Cons:** Older format, less efficient than Parquet or ORC.
- **Usage**:
  - Balances row and column storage for efficient querying.
  - Used in Hive for big data processing.
- **Consumption**:
  - Use Hive or Spark to process.
- **Example:** `Stores data similar to Parquet but less advanced.`

12. Delta Lake

- **Description**: An open-source storage layer for big data that adds ACID transactions to Parquet.
- **Cons**: Requires integration with tools like Spark.
- **Usage**:
  - Used for data lakes to ensure reliability and performance.
  - Supports time travel (querying older versions of data).
- **Consumption**:
  - Use Delta Lake libraries with Spark or Databricks.
- **Example:** `Building a reliable data lake on platforms like Databricks.`

NOTE:

- **Row-Based Formats**: CSV, JSON, Avro, XML, TSV, SequenceFile.
- **Column-Based Formats**: Parquet, ORC, RCFile, Feather, Delta Lake.