Prepare sample data :

In local directory and copy in to hdfs directory

row1 Family1:col1 Family1:col2 Family2:col1

row2 Family1:col1 Family1:col2 Family2:col1

row3 Family1:col1 Family1:col2 Family2:col1

Hadoop fs -mkdir data

Hadoop fs -put data.tsv data

Create Hbase table

hbase shell>

create 'my_table', 'Family1', 'Family2'

- $ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv
  -Dimporttsv.columns="HBASE_ROW_KEY,Family1:col1,Family1:col2,Family2:col1"
  my_table /user/training/data/inputfile.tsv

rerequisites:

1. **HBase** and **Hadoop** installed.
2. **HBase shell** for querying data directly.
3. Optionally, **Apache Hive** can be used for SQL-like queries on top of HBase.

Scenario

Let's assume we have a sales dataset with the following columns:

- `transaction_id` (Row Key)
- `product_id` (Product ID)
- `product_name` (Product Name)
- `region` (Region where the sale happened)
- `sales_amount` (Amount of Sale)
- `sale_date` (Date of Transaction)

Steps for HBase Big Data Analytics Without Spark:

Step 1: Create and Load Sample Data into HBase

1.1 Create Sales Table in HBase

In HBase shell, we first need to create a table to store the sales data.

```
bash
Copy
hbase shell
create 'sales', 'details'
```

This creates a table called `sales` with a column family `details`.

1.2 Load Sample Sales Data into HBase

Now, let's load some sample sales data into this table using the `put` command in the HBase shell.

```
bash
Copy
put 'sales', '1', 'details:product_id', '1001'
put 'sales', '1', 'details:product_name', 'Laptop'
put 'sales', '1', 'details:region', 'North'
put 'sales', '1', 'details:sales_amount', '1200'
put 'sales', '1', 'details:sale_date', '2025-01-15'

put 'sales', '2', 'details:product_id', '1002'
put 'sales', '2', 'details:product_name', 'Smartphone'
put 'sales', '2', 'details:region', 'South'
put 'sales', '2', 'details:sales_amount', '800'
put 'sales', '2', 'details:sale_date', '2025-01-16'

put 'sales', '3', 'details:product_id', '1003'
put 'sales', '3', 'details:product_name', 'Tablet'
put 'sales', '3', 'details:region', 'East'
put 'sales', '3', 'details:sales_amount', '500'
put 'sales', '3', 'details:sale_date', '2025-01-17'
```

```
put 'sales', '4', 'details:product_id', '1001'
put 'sales', '4', 'details:product_name', 'Laptop'
put 'sales', '4', 'details:region', 'West'
put 'sales', '4', 'details:sales_amount', '1100'
put 'sales', '4', 'details:sale_date', '2025-01-17'
```

This inserts four records, each representing a sale with product, region, and sales amount information.

Step 2: Querying Data with HBase Shell

You can retrieve or query data directly from HBase using the HBase shell.

2.1 Retrieve a Single Record

To retrieve data for a specific row, use the `get` command:

```
get 'sales', '1'
```

This will return the sale data for the record with row key `1`.

2.2 Scan the Entire Table

To scan all the rows in the `sales` table, use the `scan` command:

```
scan 'sales'
```

This will display all the records from the `sales` table.

2.3 Scan with Filters

You can also apply filters to your queries. For example, to get all records where `product_name` is `Laptop`, use:

```
scan 'sales', {FILTER => "SingleColumnValueFilter('details',
'product_name', =, 'binary:Laptop')"}
```

This scans for all sales where the `product_name` is `Laptop`.

Step 3: Perform Basic Analytics with HBase Shell

While HBase shell does not support advanced aggregation or SQL-like operations, you can perform simple analysis directly within the shell or use external tools like **MapReduce** for more complex processing.

3.1 Total Sales Amount Using HBase Shell

You can scan the data and manually calculate the total sales amount using HBase shell.

1. First, scan the table to get the sales amount for each transaction.

```
scan 'sales'
```

2. Manually sum up the `sales_amount` values from the results.

For more complex aggregations, use **MapReduce** to process the data.

Step 4: Perform Aggregation with HBase and MapReduce

For more complex analytics such as aggregations (e.g., total sales by region or average sales by product), you would typically use **Hadoop MapReduce**. Let's walk through the process of calculating the **total sales by region** using HBase and MapReduce.

4.1 Write a MapReduce Job to Aggregate Data by Region

You can write a **MapReduce** job in Java to scan the `sales` table and compute the total sales by region.

Mapper:

- The mapper reads data from HBase (using the HBase API) and extracts the `region` and `sales_amount`.
- It emits a key-value pair where the key is the region and the value is the sales amount.

```
java
Copy
public class SalesByRegionMapper extends TableMapper<Text, DoubleWritable>
{
    public void map(ImmutableBytesWritable row, Result value, Context
context) throws IOException, InterruptedException {
        String region =
Bytes.toString(value.getValue(Bytes.toBytes("details"),
Bytes.toBytes("region")));
        String salesAmountStr =
Bytes.toString(value.getValue(Bytes.toBytes("details"),
Bytes.toBytes("sales_amount")));

        if (region != null && salesAmountStr != null) {
            double salesAmount = Double.parseDouble(salesAmountStr);
            context.write(new Text(region), new
DoubleWritable(salesAmount));
        }
    }
}
```

Reducer:

- The reducer will receive the region as the key and a list of sales amounts as values.
- It will aggregate the sales amounts by region.

```
java
Copy
public class SalesByRegionReducer extends TableReducer<Text,
DoubleWritable, ImmutableBytesWritable> {
    public void reduce(Text key, Iterable<DoubleWritable> values, Context
context) throws IOException, InterruptedException {
        double totalSales = 0;
        for (DoubleWritable value : values) {
            totalSales += value.get();
        }

        // Output the total sales by region (you can store this in HBase or
HDFS)
        context.write(null, new Text(key.toString() + ": " + totalSales));
    }
}
```

Driver:

- The driver class will configure the MapReduce job to use HBase as the input and output source.

```
java
Copy
public class SalesByRegionJob {
    public static void main(String[] args) throws Exception {
        Configuration conf = HBaseConfiguration.create();
        Job job = Job.getInstance(conf, "Sales By Region");

        job.setJarByClass(SalesByRegionJob.class);
        job.setMapperClass(SalesByRegionMapper.class);
        job.setReducerClass(SalesByRegionReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);

        Scan scan = new Scan();
        TableMapReduceUtil.initTableMapperJob("sales", scan,
SalesByRegionMapper.class, Text.class, DoubleWritable.class, job);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

This MapReduce job will scan the HBase `sales` table, process the data, and compute the total sales per region.

4.2 Run the MapReduce Job

Once you have written the code, compile it and run the job using the `hadoop jar` command:

```
hadoop jar SalesByRegionJob.jar SalesByRegionJob
```

The output will display the total sales by region, which you can then use for reporting or further analysis.

Step 5: Using Hive for SQL-like Queries (Optional)

To make querying easier, you can use **Apache Hive** to run SQL-like queries on HBase data. This can be done using the **HBase-Hive integration**.

1. **Create a Hive Table for HBase**:

```
CREATE EXTERNAL TABLE sales (
    product_id STRING,
    product_name STRING,
    region STRING,
    sales_amount DOUBLE,
    sale_date STRING
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" =
":key,details:product_id,details:product_name,details:region,details:sales_
amount,details:sale_date")
TBLPROPERTIES ("hbase.table.name" = "sales");
```

2. **Query Data Using Hive SQL**:

```
SELECT region, SUM(sales_amount) AS total_sales FROM sales GROUP BY region;
```

This query will compute the total sales per region using SQL.