✅ Step-by-Step Guide for `@CsvFileSource`

✅ 1. Java Test Code

```
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvFileSource;

import static org.junit.jupiter.api.Assertions.assertTrue;

public class CsvFileTest {

    @ParameterizedTest
    @CsvFileSource(resources = "/data.csv", numLinesToSkip = 1)
    void testFromFile(String name, int age) {
        assertTrue(age > 0);
    }
}
```

✅ 2. CSV File Setup

📁 **File location:**

`src/test/resources/data.csv`

> This is the standard location for test resources in a Maven/Gradle project.

✅ 3. Contents of `data.csv`

```
name,age
Alice,30
Bob,25
Charlie,40
"Doe, John",28
```

- The **first line** is a **header**, so it's skipped using `numLinesToSkip = 1`.
- The last row uses **quotes** because the name contains a comma.

✅ 4. Test Iterations

name	age
Alice	30
Bob	25
Charlie	40
Doe, John	28

Each row becomes a test case.

🧠 Summary

- Use `@CsvFileSource` to keep test data external and maintainable.
- Place the `.csv` file under `src/test/resources`.
- Use `numLinesToSkip = 1` to ignore headers.
- Enclose values in quotes if they contain commas.

Would you like a ready-to-use `data.csv` example you can download?

Using `@CsvFileSource` in JUnit 5 allows you to **externalize test data** into a `.csv` file instead of hardcoding it inside your test methods. This makes your test code **cleaner, more maintainable, and easier to update**, especially when working with **large datasets** or data shared across multiple tests.

✅ Benefits of Using `@CsvFileSource`

📌 1. Separation of Concerns

- Keeps test logic (in Java) separate from test data (in CSV).
- Makes tests easier to read and maintain.

```
@ParameterizedTest
@CsvFileSource(resources = "/users.csv", numLinesToSkip = 1)
void testUserData(String username, int age) {
    assertTrue(age > 0);
}
```

CSV (`users.csv`):

```
username,age
alice,30
bob,25
charlie,40
```

📌 2. Easier Data Updates

- You can update the test data in the CSV without touching the test code.
- Non-developers or testers can update data without needing Java knowledge.

📌 3. Reusability

- Same CSV file can be reused across multiple test methods or classes.

📌 4. Scalability

- Much more readable when testing 50+ cases.
- Instead of writing 50 inline parameters, just add rows to the CSV.

📌 5. Cleaner Code

```
// Good for small sets, but becomes messy:
@CsvSource({
    "alice, 30",
    "bob, 25"
})

// Clean and scalable:
@CsvFileSource(resources = "/users.csv", numLinesToSkip = 1)
```

💡 Real-World Example

Imagine testing a registration form with various usernames and ages. Instead of bloating your test file with inline data:

Inline version (less maintainable):

```
@CsvSource({
    "user1, 20", "user2, 25", "user3, 30"
})
```

External file version (more maintainable):

```
@CsvFileSource(resources = "/registration_data.csv", numLinesToSkip = 1)
```

`registration_data.csv`:

csv
CopyEdit

```
username,age
user1,20
user2,25
user3,30
```