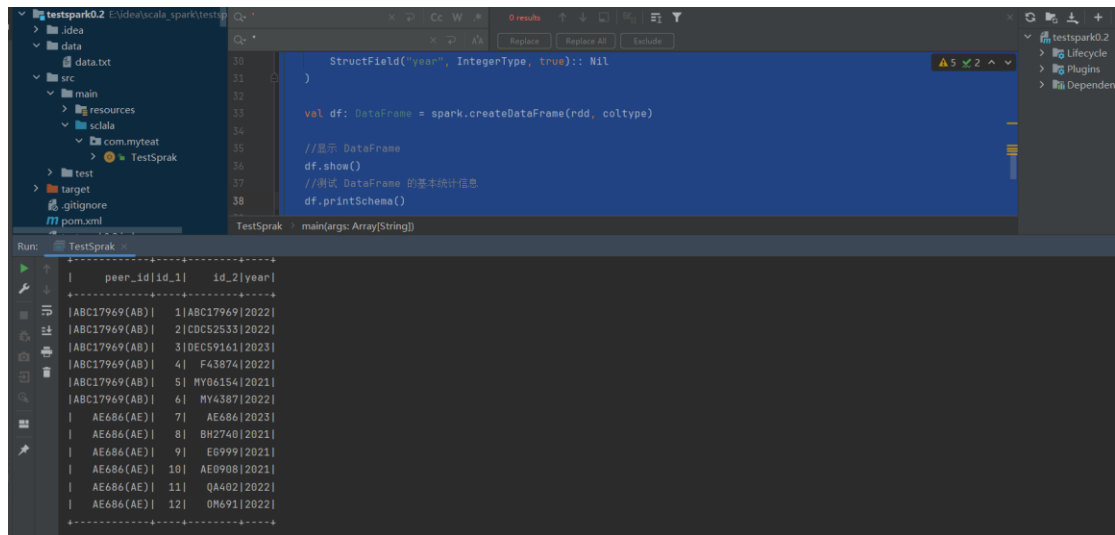


RDD to DataFrame

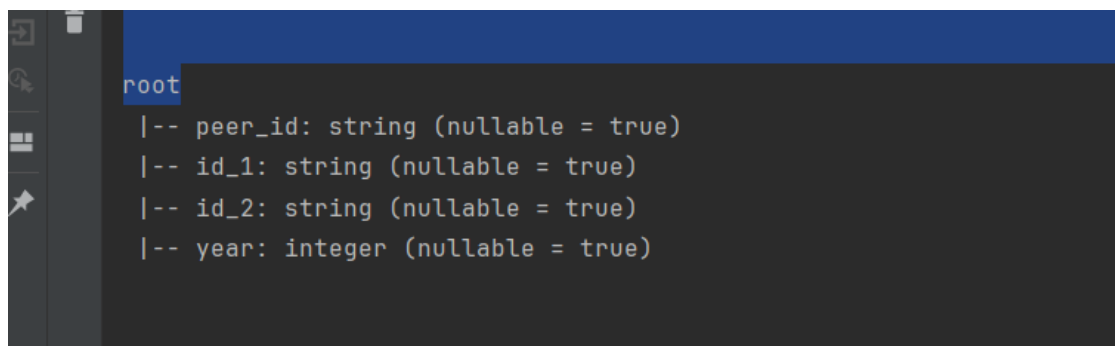


```
30      StructField("year", IntegerType, true):: Nil
31    )
32
33    val df: DataFrame = spark.createDataFrame(rdd, coltype)
34
35    //显示 DataFrame
36    df.show()
37    //测试 DataFrame 的基本统计信息
38    df.printSchema()
```

TestSprak main(args: Array[String])

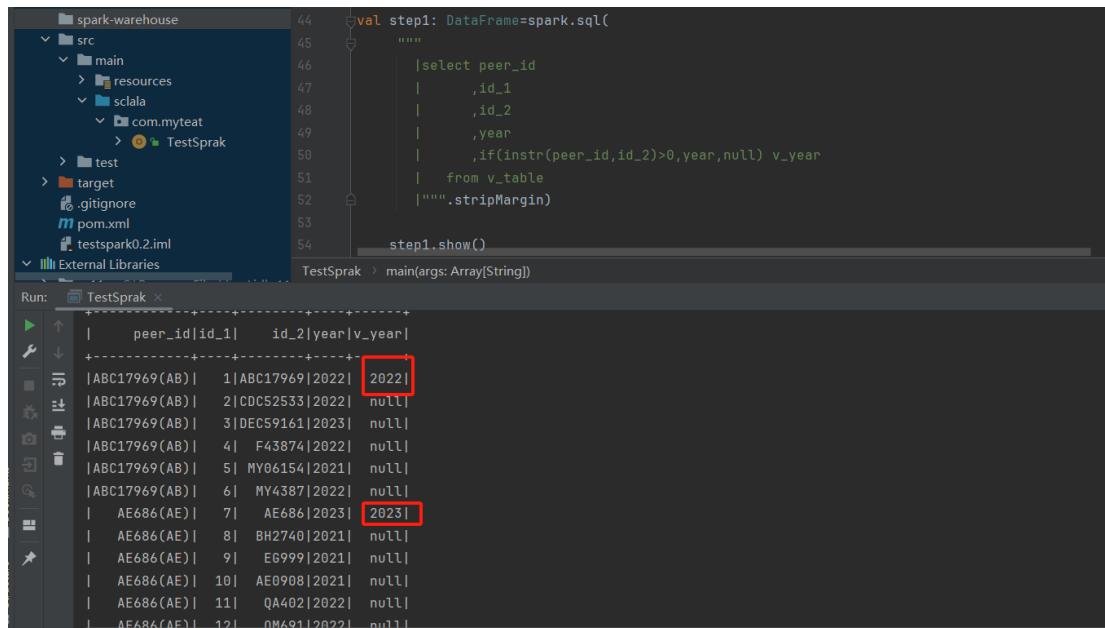
Run: TestSprak

```
-----+-----+-----+
| peer_id|id_1| id_2|year|
-----+-----+-----+
|ABC17969(AB)| 1|ABC17969|2022|
|ABC17969(AB)| 2|C0C52533|2022|
|ABC17969(AB)| 3|DEC59161|2023|
|ABC17969(AB)| 4| F43874|2022|
|ABC17969(AB)| 5| MY06154|2021|
|ABC17969(AB)| 6| MY4387|2022|
| AE686(AE)| 7| AE686|2023|
| AE686(AE)| 8| BH2740|2021|
| AE686(AE)| 9| EG9991|2021|
| AE686(AE)|10| AE0908|2021|
| AE686(AE)|11| QA402|2022|
| AE686(AE)|12| DM691|2022|
-----+-----+-----+
```



```
root
 |-- peer_id: string (nullable = true)
 |-- id_1: string (nullable = true)
 |-- id_2: string (nullable = true)
 |-- year: integer (nullable = true)
```

Step1、 For each peer_id, get the year when peer_id contains id_2, for example for 'ABC17969(AB)' year is 2022.



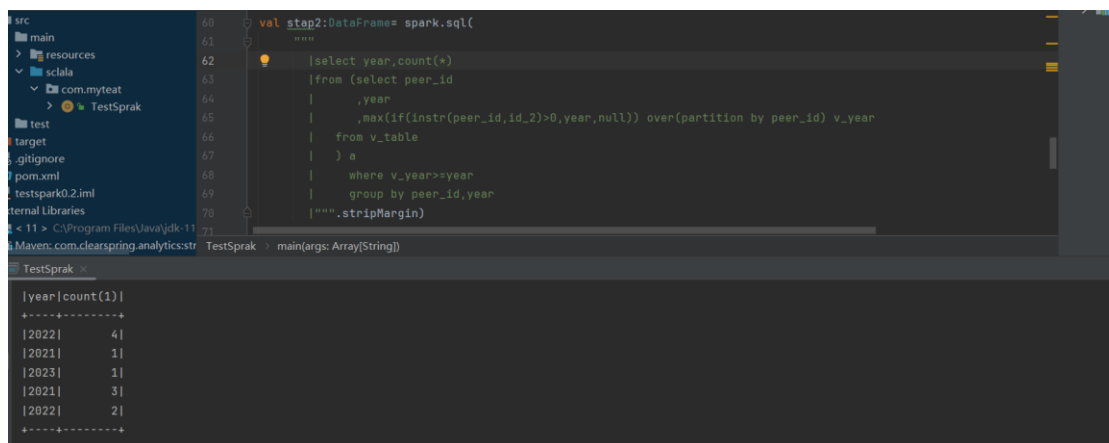
The screenshot shows a Scala IDE with a project named 'spark-warehouse'. The code defines a Spark SQL query that selects peer_id, id_1, id_2, year, and v_year. The query filters for peer_id containing id_2 and returns the year. The results are displayed in a table with 12 rows. The first row for peer_id 'ABC17969(AB)' shows year 2022, and the seventh row for peer_id 'AE686(AE)' shows year 2023. Both '2022' and '2023' are highlighted with red boxes.

```
val step1: DataFrame=spark.sql(
  """
  |select peer_id
  |      ,id_1
  |      ,id_2
  |      ,year
  |      ,if(instr(peer_id,id_2)>0,year,null) v_year
  |    from v_table
  |""".stripMargin)

step1.show()
```

peer_id	id_1	id_2	year	v_year
ABC17969(AB)	1	ABC17969	2022	2022
ABC17969(AB)	2	CDC52533	2022	null
ABC17969(AB)	3	DEC59161	2023	null
ABC17969(AB)	4	F43874	2022	null
ABC17969(AB)	5	MY06154	2021	null
ABC17969(AB)	6	MY4387	2022	null
AE686(AE)	7	AE686	2023	2023
AE686(AE)	8	BH2740	2021	null
AE686(AE)	9	E6999	2021	null
AE686(AE)	10	AE0908	2021	null
AE686(AE)	11	QA402	2022	null
AE686(AE)	12	OM691	2022	null

Step2、 Given a size number, for example
3. For each peer_id count the number of
each year (which is smaller or equal than
the year in step1).



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The code editor contains a Spark SQL query that counts the number of years for each peer_id. The output of the query is displayed in a console window at the bottom.

```
val step2:DataFrame= spark.sql(
  """
  |select year,count(*)
  |from (select peer_id
  |      ,year
  |      ,max(if(instr(peer_id,id_2)>0,year,null)) over(partition by peer_id) v_year
  |      from v_table
  |      ) a
  |   where v_year==year
  |   group by peer_id,year
  |   """
  .stripMargin)

TestSprak > main(args: Array[String])
```

year	count(1)
2022	4
2021	1
2023	1
2021	3
2022	2

Step3、 Order the value in step 2 by year and
check if the count number of the first year is
bigger or equal than the given size number.
If yes, just return the year. If not, plus the
count number from the biggest year to next
year until the count number is bigger or
equal than the given number. For example,
for 'AE686(AE)', the year is 2023, and count
are

The screenshot shows an IDE with a Scala file named `TestSprak.scala`. The code defines a Spark SQL query that selects `peer_id`, `year`, and `v_count` from a table, grouped by `peer_id` and `year`. The query is executed using `spark.sql(sql.stripMargin)`. The output of the query is displayed in the console, showing a table with columns `peer_id`, `year`, `v_count`, and `v_sum`.

```
79 from (
80   select peer_id, year, count(*) v_count
81   from (select peer_id
82         , year
83         , max(if(instr(peer_id, id_2) > 0, year, null)) over(partition by peer_id) v_year
84        from v_table
85       ) a
86   where v_year = year
87   group by peer_id, year
88  ) aa
89  ) bb where 1=1 and v_count+v_sum >= 5 order by peer_id, year desc
90  """, "3")
91 val step3: DataFrame = spark.sql(sql.stripMargin);
92 step3.show()
93
94
```

Run: TestSprak (1) x

peer_id	year	v_count	(v_count + v_sum)
ABC17969 (AB)	2022	4	5
AE686 (AE)	2023	1	3
AE686 (AE)	2022	2	5

ScalaSparkApi

The screenshot shows an IDE with a Scala file named `ScalaSparkDataFrameApi.scala`. The code defines a Spark DataFrame API query that selects `peer_id`, `year`, and `v_count` from a table, grouped by `peer_id` and `year`. The query is executed using `ScalaSparkDataFrameApi.sqlApi`. The output of the query is displayed in the console, showing a table with columns `peer_id`, `year`, `v_count`, and `v_sum`.

```
76 val frame = ScalaSparkDataFrameApi.sqlApi("data/data.txt", 3)
77 frame.foreach(println)
78
79 val frame2 = ScalaSparkDataFrameApi.sqlApi("data/data2.txt", 3)
80 frame2.foreach(println)
81
82
83
84
85
86
```

Run: ScalaSparkDataFrameApi x

Process finished with exit code 0

[ABC17969(AB), 2022, 4, 5]
[AE686(AE), 2023, 1, 3]
[AE686(AE), 2022, 2, 5]
[ABC17969(AB), 2022, 4, 5]
[AE686(AE), 2023, 1, 3]
[AE686(AE), 2022, 2, 5]