Step 1：对于每个 peer_id，获取 peer_id 包含 id_2 时的年份

代码片段：

```
def getYearByPeer(df: DataFrame): DataFrame = {
    df.withColumn("contains_id_2", expr("instr(peer_id, id_2) > 0"))
      .filter(expr("contains_id_2 = true"))
      .withColumnRenamed("year", "min_year")
  }
```

结果截图：

```
+------------+----+--------+--------+-------------+
|     peer_id|id_1|    id_2|min_year|contains_id_2|
+------------+----+--------+--------+-------------+
|ABC17969(AB)|   1|ABC17969|    2022|         true|
|   AE686(AE)|   7|   AE686|    2023|         true|
+------------+----+--------+--------+-------------+
```

Step 2：给定一个大小数，例如 3。对于每个 peer_id，计算每个年份(小于或等于 step1 中的年份)的个数

代码片段:

```
def getCountsByYear(df: DataFrame, yearByPeer: DataFrame): DataFrame = {
    df.join(yearByPeer, Seq("peer_id"), "inner")
      .filter(expr("year <= min_year"))
      .groupBy("peer_id", "year")
      .count()
  }
```

结果截图：

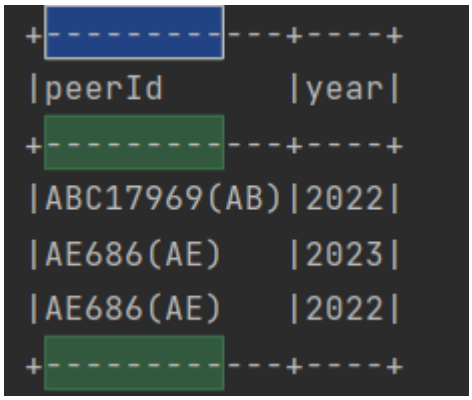
```
+------------+----+-----+
|     peer_id|year|count|
+------------+----+-----+
|   AE686(AE)|2022|    2|
|ABC17969(AB)|2021|    1|
|   AE686(AE)|2023|    1|
|ABC17969(AB)|2022|    4|
|   AE686(AE)|2021|    3|
+------------+----+-----+
```

Step 3：将步骤 2 中的值按年份排序，并检查第一年的计数数是否大于或等于给定的大小数。如果是，只需返回年份。如果不是，则加上从最大年份到下一年的计数数，直到计数数大于或等于给定的数

代码片段：

```
def calculateOutput(countsByYear: DataFrame, givenSize: Int): DataFrame = {
    val result = countsByYear.sort(desc("year"))
      .groupBy("peer_id").agg(collect_list(struct("year", "count")).as("data"))

    result.flatMap { row =>
      val peer_id = row.getString(0)
      val dataList = row.getAs[Seq[Row]](1)

      var cumulativeCount = 0
      var outputList = List.empty[Result]
      var stop = false

      for (data <- dataList if !stop) {
        val year = data.getInt(0)
        val count = data.getLong(1).toInt
        cumulativeCount += count
        outputList = Result(peer_id, year) :: outputList
        if (cumulativeCount >= givenSize) {
          stop = true
        }
      }

      outputList.reverse
    }.toDF()
  }
```

结果截图：(给定大小值为 3)



peerId	year
ABC17969(AB)	2022
AE686(AE)	2023
AE686(AE)	2022

完整代码：

```
package test.demo
import org.apache.spark.sql.{DataFrame, Row, SparkSession}
import org.apache.spark.sql.functions._

object PeerYearCounter2 {

  // 定义结果的 case class
  case class Result(peerId: String, year: Int)

  // 创建 SparkSession
  val spark = SparkSession.builder()
    .appName("Peer Year Counter")
    .master("local[*]")
    .getOrCreate()

  import spark.implicits._

  def main(args: Array[String]): Unit = {


    // 示例数据
    val data = Seq(
      ("ABC17969(AB)", "1", "ABC17969", 2022),
      ("ABC17969(AB)", "2", "CDC52533", 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", "EG999", 2021),
      ("AE686(AE)", "10", "AE0908", 2021),
      ("AE686(AE)", "11", "QA402", 2022),
      ("AE686(AE)", "12", "OM691", 2022)
    )

    // 将数据转换为 DataFrame
    val df = data.toDF("peer_id", "id_1", "id_2", "year")

    // 步骤 1: 获取每个 peer_id 包含 id_2 时的年份
    val yearByPeer = getYearByPeer(df)
    yearByPeer.show()
```

```
    // 步骤 2: 统计每个 peer_id 在每个年份的计数
    val countsByYear = getCountsByYear(df, yearByPeer)
    countsByYear.show()

    // 步骤 3: 将步骤 2 中的值按年份排序，并检查第一年的计数数是否大于或等于给定的大小数。如果是，只需返回年份。如果不是，则加上从最大年份到下一年的计数数，直到计数数大于或等于给定的数
    val output = calculateOutput(countsByYear, 3)
    // 显示结果
    output.show(false)

    // 停止 SparkSession
    spark.stop()
  }

  // 步骤 1: 获取每个 peer_id 包含 id_2 时的年份
  def getYearByPeer(df: DataFrame): DataFrame = {
    df.withColumn("contains_id_2", expr("instr(peer_id, id_2) > 0"))
      .filter(expr("contains_id_2 = true"))
      .withColumnRenamed("year", "min_year")
  }

  // 步骤 2: 统计每个 peer_id 在每个年份的计数
  def getCountsByYear(df: DataFrame, yearByPeer: DataFrame): DataFrame = {
    df.join(yearByPeer, Seq("peer_id"), "inner")
      .filter(expr("year <= min_year"))
      .groupBy("peer_id", "year")
      .count()
  }

  // 步骤 3: 计算结果
  def calculateOutput(countsByYear: DataFrame, givenSize: Int): DataFrame = {
    val result = countsByYear.sort(desc("year"))
      .groupBy("peer_id").agg(collect_list(struct("year", "count")).as("data"))

    result.flatMap { row =>
      val peer_id = row.getString(0)
      val dataList = row.getAs[Seq[Row]](1)

      var cumulativeCount = 0
      var outputList = List.empty[Result]
      var stop = false
```

```
        for (data <- dataList if !stop) {
            val year = data.getInt(0)
            val count = data.getLong(1).toInt
            cumulativeCount += count
            outputList = Result(peer_id, year) :: outputList
            if (cumulativeCount >= givenSize) {
                stop = true
            }
        }

        outputList.reverse
    }.toDF()
  }
}
```

单元测试代码：

```
package test.demo

import org.scalatest._
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.Row
import org.apache.spark.sql.functions._

class UnitTest extends AnyFunSuite with BeforeAndAfter {
  var spark: SparkSession = _

  before {
    spark = SparkSession.builder()
      .appName("YearlyCountUnitTest")
      .master("local[*]")
      .getOrCreate()
  }

  after {
    if (spark != null) {
      spark.stop()
    }
  }

  test("Test getYearByPeer function") {
    import spark.implicits._
```

```
    // Sample data
    val data = Seq(
      ("ABC17969(AB)", "1", "ABC17969", 2022),
      ("ABC17969(AB)", "2", "CDC52533", 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", "EG999", 2021),
      ("AE686(AE)", "10", "AE0908", 2021),
      ("AE686(AE)", "11", "QA402", 2022),
      ("AE686(AE)", "12", "OM691", 2022)
    )
    val df = data.toDF("peer_id", "id_1", "id_2", "year")
    val result = PeerYearCounter2.getYearByPeer(df)

    assert(result.map(_.toString) === expectedResult.map(_.toString))
  }
}
```