DDM Spark Project

TeamName: MatrixTeam

Members:

<Mohamad> <Kaser>, <Kaser@students.uni-marburg.de>

<Parmida> <Talebi>, < Talebip@students.uni-marburg.de >

<Anusha Nepal> <Upadhaya>, <Nepalupa@students.uni-marburg.de>

```scala
def discoverINDs(inputs: List[String], spark: SparkSession): Unit = {
  import spark.implicits._

  val columnsData = inputs.flatMap(input => {
    val df = readData(input, spark)
    df.columns.map(column => (column, df.select(column).distinct.as[String].collect.toSet))
  }).toMap
```

Here, Each column is paired with its corresponding set of distinct values. This pair is a tuple where the first element is the column name and the second is the set of distinct values. Finally, the resulting pairs from all input files are combined into a single map

```scala
val potentialINDs = columnsData.keys.flatMap { dependentCol =>
  columnsData.keys.collect {
    case referenceCol if dependentCol != referenceCol && columnsData(dependentCol).subsetOf(columnsData(referenceCol)) =>
      (dependentCol, referenceCol)
  }
}
```

Inside the flatmap, the collect method iterates over all column names and check for subset relation, For each pair of columns, it checks two conditions.

The output “potentialINDs” is a collection of tuples, indicating all potential inclusion dependencies found across the columns in the input datasets

```
val aggregatedINDs = potentialINDs
  .groupBy(_._1)
  .mapValues(_.map(_._2).toList.sorted)
  .toSeq
  .sortBy(_._1)
```

This process aggregates all the reference columns for each dependent column into a sorted list

```
  aggregatedINDs.foreach {
    case (dependent, references) =>
      println(s"$dependent < ${references.mkString(", ")}")
  }
 }
}
```

Printing the result in the desired format