Time usage

```
// Input file
z.put("timeusage", "/home/ubuntu/test_spark/timeusage/atussum.csv")
z.put("test", "/home/ubuntu/test_spark/timeusage/test.csv")
```

```
// Read RDD
val rdd = sc.textFile(z.get("timeusage").toString)
rdd.take(2)
```

```
rdd: org.apache.spark.rdd.RDD[String] = /home/ubuntu/test_spark/timeusage/atussum.csv MapPartitionsRDD[663] at textFile at <console>:156
res151: Array[String] = Array(tucaseid,gemetsta,gtmetsta,peeduca,pehspnon,ptdtrace,teage,telfs,temjot,teschenr,teschlvl,tesex,tespempnot,trchildnum,trdpftpt,trernwa,trholiday,trspftpt,trsppres,tryhhchild
```

```
val headerColumn = rdd.first.split(",").toList
```

```
headerColumn: List[String] = List(tucaseid, gemetsta, gtmetsta, peeduca, pehspnon, ptdtrace, teage, telfs, temjot, teschenr, teschlvl, tesex, tespempnot, trchildnum, trdpftpt, trernwa, trholiday, trspftp
```

```
headerColumn.tail
```

```
res38: List[String] = List(gemetsta, gtmetsta, peeduca, pehspnon, ptdtrace, teage, telfs, temjot, teschenr, teschlvl, tesex, tespempnot, trchildnum, trdpftpt, trernwa, trholiday, trspftpt, trsppres, tryh
```

```
val l1 = List(1, 2, 3, 4)
val l2 = List(5, 6)

val l3 = l1 ++ l2
val l4 = l1 ++: l2
val l5 = l1 +: l2
val l6 = l1 :+ l2
```

```
l1: List[Int] = List(1, 2, 3, 4)
l2: List[Int] = List(5, 6)
l3: List[Int] = List(1, 2, 3, 4, 5, 6)
l4: List[Int] = List(1, 2, 3, 4, 5, 6)
l5: List[Any] = List(List(1, 2, 3, 4), 5, 6)
l6: List[Any] = List(1, 2, 3, 4, List(5, 6))
```

```
// Convert header into schema
import org.apache.spark.sql._
import org.apache.spark.sql.types._

val fields = List(new StructField(headerColumn.head, StringType, false))
val fields_tail = headerColumn.tail.map(header => new StructField(header, DoubleType, false))
```

```
import org.apache.spark.sql._
import org.apache.spark.sql.types._
fields: List[org.apache.spark.sql.types.StructField] = List(StructField(tucaseid,StringType,false))
fields_tail: List[org.apache.spark.sql.types.StructField] = List(StructField(gemetsta,DoubleType,false), StructField(gtmetsta,DoubleType,false), StructField(peeduca,DoubleType,false), StructField(pehspno
```

```
val all_fields = fields ++: fields_tail
```

```
all_fields: List[org.apache.spark.sql.types.StructField] = List(StructField(tucaseid,StringType,false), StructField(gemetsta,DoubleType,false), StructField(gtmetsta,DoubleType,false), StructField(peeduca
```

```
val schema1 = new StructType(all_fields.toArray)
```

```
schema1: org.apache.spark.sql.types.StructType = StructType(StructField(tucaseid,StringType,false), StructField(gemetsta,DoubleType,false), StructField(gtmetsta,DoubleType,false), StructField(peeduca,Dou
```

```
rdd.partitions.length
```

```
res39: Int = 5
```

```
import org.apache.spark.sql._
import org.apache.spark.sql.types._

/** @return The schema of the DataFrame, assuming that the first given column has type String and all the others
    *         have type Double. None of the fields are nullable.
    * @param columnNames Column names of the DataFrame
    */
def dfSchema(columnNames: List[String]): StructType = {

    val fields = List(new StructField(columnNames.head, StringType, false)) // First column
    val rest_of_fields = columnNames.tail.map(header => new StructField(header, DoubleType, false))
    val all_fields = fields ++: rest_of_fields
    new StructType(all_fields.toArray)
}
```

```
import org.apache.spark.sql._
import org.apache.spark.sql.types._
dfSchema: (columnNames: List[String])org.apache.spark.sql.types.StructType
```

```
/** @return An RDD Row compatible with the schema produced by `dfSchema`
    * @param line Raw fields
    */
def row(line: List[String]): Row = {
    val first = List(line.head.toString)
    val rest = line.tail.map(_.toDouble)
    val first_plus_rest = first ++: rest
    Row.fromSeq(first_plus_rest.toSeq)
}
```

```
row: (line: List[String])org.apache.spark.sql.Row
```

```
val data1 = rdd
              .mapPartitionsWithIndex((i, it) => if (i==0) it.drop(1) else it)
              .map(_.split(",").toList)
              .map(row)

data1.take(1)
// // val df1 = spark.createDataFrame(data1.sample(), schema1)
// val sample = data1.sample(false, 0.0001)
// sample.count

// sample.take(1)

val df = spark.createDataFrame(data1, schema1)
df.count
```

```
<console>:171: error: not found: value schema1
       val df = spark.createDataFrame(data1, schema1)
                                             ^
```

```
/** @return The read DataFrame along with its column names. */
def read(resource: String): (List[String], DataFrame) = {
    val rdd = sc.textFile(resource)
    val headerColumns = rdd.first.split(",").toList // Get the header line as list
    val schema = dfSchema(headerColumns)  // Generate schema out of header columns

    // Convert each rdd element to Row
    val data = rdd
                 .mapPartitionsWithIndex((i, it) => if (i == 0) it.drop(1) else it) // skip header line from first partition
                 .map(line => row(line.split(",").toList)) // Convert each line into Row

    // Create DataFrame
    val df = spark.createDataFrame(data, schema)
    (headerColumns, df)
}
```

```
read: (resource: String)(List[String], org.apache.spark.sql.DataFrame)
```

```
val (columns, initDF) = read(z.get("timeusage").toString)
```

```
columns: List[String] = List(tucaseid, gemetsta, gtmetsta, peeduca, pehspnon, ptdtrace, teage, telfs, temjot, teschenr, teschlvl, tesex, tespempnot, trchildnum, trdpftpt, trernwa, trholiday, trspftpt, tr
```

```
initDF.show(2)
```

```
+----------------+--------+--------+-------+--------+--------+-----+-----+------+--------+--------+-----+----------+----------+--------+-------+---------+--------+--------+----------+----------+---------
|        tucaseid|gemetsta|gtmetsta|peeduca|pehspnon|ptdtrace|teage|telfs|temjot|teschenr|teschlvl|tesex|tespempnot|trchildnum|trdpftpt|trernwa|trholiday|trspftpt|trsppres|tryhhchild|tudiaryday| tufnwgtp
+----------------+--------+--------+-------+--------+--------+-----+-----+------+--------+--------+-----+----------+----------+--------+-------+---------+--------+--------+----------+----------+---------
|"20030100013280"|     1.0|    -1.0|   44.0|     2.0|     2.0| 60.0|  2.0|   2.0|    -1.0|    -1.0|  1.0|       2.0|       0.0|     2.0|66000.0|      0.0|    -1.0|     1.0|      -1.0|       6.0|8155463.0
|"20030100013344"|     2.0|    -1.0|   40.0|     2.0|     1.0| 41.0|  1.0|   2.0|     2.0|    -1.0|  2.0|       1.0|       2.0|     2.0|20000.0|      0.0|     1.0|     1.0|       0.0|       7.0|1735323.0
```

```
+----------------+--------+--------+-------+--------+--------+-----+-----+------+--------+--------+-----+----------+----------+--------+-------+---------+--------+--------+----------+----------+---------
only showing top 2 rows
```

```scala
/** @return The initial data frame columns partitioned in three groups: primary needs (sleeping, eating, etc.),
    *         work and other (leisure activities)
    *
    * @see https://www.kaggle.com/bls/american-time-use-survey
    *
    * The dataset contains the daily time (in minutes) people spent in various activities. For instance, the column
    * "t010101" contains the time spent sleeping, the column "t110101" contains the time spent eating and drinking, etc.
    *
    * This method groups related columns together:
    * 1. "primary needs" activities (sleeping, eating, etc.). These are the columns starting with "t01", "t03", "t11",
    *    "t1801" and "t1803".
    * 2. working activities. These are the columns starting with "t05" and "t1805".
    * 3. other activities (leisure). These are the columns starting with "t02", "t04", "t06", "t07", "t08", "t09",
    *    "t10", "t12", "t13", "t14", "t15", "t16" and "t18" (those which are not part of the previous groups only).
    */

import scala.collection.mutable.ListBuffer
  def classifiedColumns(columnNames: List[String]): (List[Column], List[Column], List[Column]) = {
      val primary = ListBuffer[Column]()
      val work = ListBuffer[Column]()
      val other = ListBuffer[Column]()
      for(c <- columnNames) {
          if (c.startsWith("t01") || c.startsWith("t03") || c.startsWith("t11") || c.startsWith("t1801") || c.startsWith("t1803")) {
              primary += new Column(c)
          } else if (c.startsWith("t05") || c.startsWith("t1805")) {
              work += new Column(c)
          } else if (c.startsWith("t02") || c.startsWith("t04") || c.startsWith("t06") || c.startsWith("t07") || c.startsWith("t08") ||
                     c.startsWith("t09") || c.startsWith("t10") || c.startsWith("t12") || c.startsWith("t13") || c.startsWith("t14") ||
                     c.startsWith("t15") || c.startsWith("t16") || (c.startsWith("t18") && !c.startsWith("t1805"))) {
              other += new Column(c)
          }
      }
      (primary.toList, work.toList, other.toList)
  }
```

```
import scala.collection.mutable.ListBuffer
classifiedColumns: (columnNames: List[String])(List[org.apache.spark.sql.Column], List[org.apache.spark.sql.Column], List[org.apache.spark.sql.Column])
```

```scala
val (primary1, work1, other1) = classifiedColumns(columns)
```

```
primary1: List[org.apache.spark.sql.Column] = List(t010101, t010102, t010199, t010201, t010299, t010301, t010399, t010401, t010499, t010501, t010599, t019999, t030101, t030102, t030103, t030104, t030105,
work1: List[org.apache.spark.sql.Column] = List(t050101, t050102, t050103, t050189, t050201, t050202, t050203, t050204, t050289, t050301, t050302, t050303, t050304, t050389, t050403, t050404, t050405, t0
```

```scala
// org.apache.spark.sql.functions.when

val working = when($"telfs".between(1, 2), "working")
              .otherwise("not working")
              .as("working")
```

```
working: org.apache.spark.sql.Column = CASE WHEN ((telfs >= 1) AND (telfs <= 2)) THEN working ELSE not working END AS `working`
```

```scala
initDF.select(working).show()
```

```
+-----------+
|    working|
+-----------+
|    working|
|    working|
|    working|
|not working|
|    working|
|    working|
|    working|
|    working|
|    working|
|    working|
|    working|
|    working|
|    working|
|not working|
|    working|
|    working|
|not working|
|    working|
|not working|
|    working|
+-----------+
only showing top 20 rows
```

```scala
val primaryNeeds1 = (primary1.reduce(_ + _) / 60).as("p")
```

```
primaryNeeds1: org.apache.spark.sql.Column = (((((((((((((((((((((((((((((((((((((((((((((((((((((((t010101 + t010102) + t010199) + t010201) + t010299) + t010301) + t010399) + t010401) + t010499) + t0105
```

```scala
initDF.select(primaryNeeds1).show
```

```
+------------------+
|                 p|
+------------------+
|             15.25|
|13.833333333333334|
|11.916666666666666|
|13.083333333333334|
|11.783333333333333|
|              17.0|
|12.783333333333333|
|               9.0|
|13.166666666666666|
| 6.683333333333334|
| 9.833333333333334|
|12.416666666666666|
|11.633333333333333|
|              14.0|
|             12.15|
|             13.75|
|             11.25|
|11.166666666666666|
|12.666666666666666|
|11.416666666666666|
+------------------+
only showing top 20 rows
```

```scala
/** @return a projection of the initial DataFrame such that all columns containing hours spent on primary needs
    *         are summed together in a single column (and same for work and leisure). The â€œteageâ€ column is also
    *         projected to three values: "young", "active", "elder".
    *
    * @param primaryNeedsColumns List of columns containing time spent on â€œprimary needsâ€
    * @param workColumns List of columns containing time spent working
    * @param otherColumns List of columns containing time spent doing other activities
    * @param df DataFrame whose schema matches the given column lists
    *
    * This methods builds an intermediate DataFrame that sums up all the columns of each group of activity into
    * a single column.
    *
    * The resulting DataFrame should have the following columns:
    * - working: value computed from the â€œtelfsâ€ column of the given DataFrame:
    *   - "working" if 1 <= telfs < 3
    *   - "not working" otherwise
    * - sex: value computed from the â€œtesexâ€ column of the given DataFrame:
    *   - "male" if tesex = 1, "female" otherwise
    * - age: value computed from the â€œteageâ€ column of the given DataFrame:
    *   - "young" if 15 <= teage <= 22,
    *   - "active" if 23 <= teage <= 55,
    *   - "elder" otherwise
    * - primaryNeeds: sum of all the `primaryNeedsColumns`, in hours
    * - work: sum of all the `workColumns`, in hours
    * - other: sum of all the `otherColumns`, in hours
    *
    * Finally, the resulting DataFrame should exclude people that are not employable (ie telfs = 5).
    *
    * Note that the initial DataFrame contains time in ''minutes''. You have to convert it into ''hours''.
    */

def timeUsageSummary(
    primaryNeedsColumns: List[Column],
    workColumns: List[Column],
    otherColumns: List[Column],
    df: DataFrame
  ): DataFrame = {
    // Transform the data from the initial dataset into data that make
    // more sense for our use case
    // Hint: you can use the `when` and `otherwise` Spark functions
    // Hint: donâ€™t forget to give your columns the expected name with the `as` method
    val workingStatusProjection: Column = when(col("telfs").between(1, 2), "working")
                                           .otherwise("not working")
                                           .as("working")
    val sexProjection: Column = when($"tesex" === 1, "male")
                                .otherwise("female")
                                .as("sex")
    val ageProjection: Column = when(col("teage").between(15, 22), "young")
                                .when(col("teage").between(23, 55), "active")
                                .otherwise("elder")
                                .as("age")

    // Create columns that sum columns of the initial dataset
    // Hint: you want to create a complex column expression that sums other columns
    //       by using the `+` operator between them
    // Hint: donâ€™t forget to convert the value to hours
    val primaryNeedsProjection: Column = (primaryNeedsColumns.reduce(_ + _) / 60).as("primaryNeeds")
    val workProjection: Column = (workColumns.reduce(_ + _) / 60).as("work")
    val otherProjection: Column = (otherColumns.reduce(_ + _) / 60).as("other")

    df
      .select(workingStatusProjection, sexProjection, ageProjection, primaryNeedsProjection, workProjection, otherProjection)
      .where($"telfs" <= 4) // Discard people who are not in labor force
  }

timeUsageSummary: (primaryNeedsColumns: List[org.apache.spark.sql.Column], workColumns: List[org.apache.spark.sql.Column], otherColumns: List[org.apache.spark.sql.Column], df: org.apache.spark.sql.DataFr

val (primaryNeedsColumns, workColumns, otherColumns) = classifiedColumns(columns)
val summaryDF = timeUsageSummary(primaryNeedsColumns, workColumns, otherColumns, initDF)

primaryNeedsColumns: List[org.apache.spark.sql.Column] = List(t010101, t010102, t010199, t010201, t010299, t010301, t010399, t010401, t010499, t010501, t010599, t019999, t030101, t030102, t030103, t03010
workColumns: List[org.apache.spark.sql.Column] = List(t050101, t050102, t050103, t050189, t050201, t050202, t050203, t050204, t050289, t050301, t050302, t050303, t050304, t050389, t050403, t050404, t0504

summaryDF.show

+-----------+------+------+------------------+------------------+------------------+
|    working|   sex|   age|      primaryNeeds|              work|             other|
+-----------+------+------+------------------+------------------+------------------+
|    working|  male| elder|             15.25|               0.0|              8.75|
|    working|female|active|13.833333333333334|               0.0|10.166666666666666|
|    working|female|active|11.916666666666666|               0.0|12.083333333333334|
|not working|female|active|13.083333333333334|               2.0| 8.916666666666666|
|    working|  male|active|11.783333333333333| 8.583333333333334|3.6333333333333333|
|    working|female|active|              17.0|               0.0|               7.0|
|    working|female|active|12.783333333333333| 8.566666666666666|              2.65|
|    working|female| young|               9.0| 9.083333333333334| 5.916666666666667|
|    working|female|active|13.166666666666666|               0.0|10.833333333333334|
|    working|female|active| 6.683333333333334|               4.5|12.816666666666666|
|    working|  male|active| 9.833333333333334|12.133333333333333| 2.033333333333333|
|    working|female|active|12.416666666666666|               0.0|11.583333333333334|
|    working|female|active|11.633333333333333| 6.333333333333333| 6.033333333333333|
|    working|female|active|             12.15|               9.0|              2.85|
|    working|female|active|             13.75|              0.75|               9.5|
|    working|female|active|11.166666666666666|1.0833333333333333|             11.75|
|    working|female| young|11.416666666666666|               0.0|12.583333333333334|
|    working|female|active|              15.8|               0.0|               8.2|
|    working|  male|active| 9.666666666666666|11.616666666666667| 2.716666666666667|
|    working|female|active|              12.1| 7.966666666666667| 3.933333333333333|
+-----------+------+------+------------------+------------------+------------------+
only showing top 20 rows

summaryDF.filter('sex === "male").show

+-------+----+------+------------------+------------------+------------------+
|working| sex|   age|      primaryNeeds|              work|             other|
+-------+----+------+------------------+------------------+------------------+
|working|male| elder|             15.25|               0.0|              8.75|
|working|male|active|11.783333333333333| 8.583333333333334|3.6333333333333333|
|working|male|active| 9.833333333333334|12.133333333333333| 2.033333333333333|
|working|male|active| 9.666666666666666|11.616666666666667| 2.716666666666667|
|working|male|active| 9.833333333333334|              9.75| 4.416666666666667|
|working|male| young|14.333333333333334|               0.0| 9.666666666666666|
|working|male|active|              17.0|              2.75|              4.25|
|working|male|active| 9.333333333333334| 9.916666666666666|              4.75|
|working|male|active| 9.183333333333334| 9.083333333333334| 5.233333333333333|
|working|male|active|              8.25|               3.0|             12.75|
|working|male|active| 9.383333333333333| 8.666666666666666| 5.633333333333334|
|working|male|active|10.833333333333334|               0.0|13.166666666666666|
|working|male|active|15.833333333333334|               0.0| 8.166666666666666|
|working|male|active|11.083333333333334|               0.0|12.916666666666666|
|working|male|active|              13.0|               0.0|              11.0|
|working|male|active|11.216666666666667|               0.0|12.783333333333333|
|working|male|active|11.583333333333334| 8.416666666666666|3.1666666666666665|
|working|male|active|              12.7|               0.0|              11.3|
|working|male|active| 7.416666666666667|15.583333333333334|               1.0|
|working|male|active| 6.666666666666667|13.166666666666666| 4.166666666666667|
+-------+----+------+------------------+------------------+------------------+
only showing top 20 rows

import org.apache.spark.sql.functions._
summaryDF.groupBy($"working", $"sex", $"age").agg(round(avg("primaryNeeds"), 1).as("primaryNeeds"), round(avg("work"), 1).as("work"), round(avg("other"), 1).as("other")).orderBy($"working", $"sex", $"age

+-----------+------+------+------------+----+-----+
|    working|   sex|   age|primaryNeeds|work|other|
+-----------+------+------+------------+----+-----+
|not working|female|active|        12.4| 0.5| 10.8|
|not working|female| elder|        10.9| 0.4| 12.4|
|not working|female| young|        12.5| 0.2| 11.1|
|not working|  male|active|        11.4| 0.9| 11.4|
|not working|  male| elder|        10.7| 0.7| 12.3|
|not working|  male| young|        11.6| 0.2| 11.9|
|    working|female|active|        11.5| 4.2|  8.1|
|    working|female| elder|        10.6| 3.9|  9.3|
|    working|female| young|        11.6| 3.3|  8.9|
|    working|  male|active|        10.8| 5.2|  7.8|
|    working|  male| elder|        10.4| 4.8|  8.6|
|    working|  male| young|        10.9| 3.7|  9.2|
+-----------+------+------+------------+----+-----+

import org.apache.spark.sql.functions._

import org.apache.spark.sql.functions._
  /** @return the average daily time (in hours) spent in primary needs, working or leisure, grouped by the different
    *         ages of life (young, active or elder), sex and working status.
    * @param summed DataFrame returned by `timeUsageSumByClass`
    *
    * The resulting DataFrame should have the following columns:
    * - working: the â€œworkingâ€ column of the `summed` DataFrame,
    * - sex: the â€œsexâ€ column of the `summed` DataFrame,
    * - age: the â€œageâ€ column of the `summed` DataFrame,
    * - primaryNeeds: the average value of the â€œprimaryNeedsâ€ columns of all the people that have the same working
    *   status, sex and age, rounded with a scale of 1 (using the `round` function),
    * - work: the average value of the â€œworkâ€ columns of all the people that have the same working status, sex
    *   and age, rounded with a scale of 1 (using the `round` function),
    * - other: the average value of the â€œotherâ€ columns all the people that have the same working status, sex and
    *   age, rounded with a scale of 1 (using the `round` function).
    *
    * Finally, the resulting DataFrame should be sorted by working status, sex and age.
    */
  def timeUsageGrouped(summed: DataFrame): DataFrame = {
    summed.groupBy($"working", $"sex", $"age").agg(round(avg("primaryNeeds"), 1).as("primaryNeeds"), round(avg("work"), 1).as("work"), round(avg("other"), 1).as("other")).orderBy($"working", $"sex", $"ag
  }
```

```
import org.apache.spark.sql.functions._
timeUsageGrouped: (summed: org.apache.spark.sql.DataFrame)org.apache.spark.sql.DataFrame
```

```
val finalDF = timeUsageGrouped(summaryDF)
finalDF.show
```

```
+-----------+------+------+------------+----+-----+
|    working|   sex|   age|primaryNeeds|work|other|
+-----------+------+------+------------+----+-----+
|not working|female|active|        12.4| 0.5| 10.8|
|not working|female| elder|        10.9| 0.4| 12.4|
|not working|female| young|        12.5| 0.2| 11.1|
|not working|  male|active|        11.4| 0.9| 11.4|
|not working|  male| elder|        10.7| 0.7| 12.3|
|not working|  male| young|        11.6| 0.2| 11.9|
|    working|female|active|        11.5| 4.2|  8.1|
|    working|female| elder|        10.6| 3.9|  9.3|
|    working|female| young|        11.6| 3.3|  8.9|
|    working|  male|active|        10.8| 5.2|  7.8|
|    working|  male| elder|        10.4| 4.8|  8.6|
|    working|  male| young|        10.9| 3.7|  9.2|
+-----------+------+------+------------+----+-----+

finalDF: org.apache.spark.sql.DataFrame = [working: string, sex: string ... 4 more fields]
```

```
finalDF.rdd.getNumPartitions
```

```
res76: Int = 12
```

```
// Bucketing
// https://jaceklaskowski.gitbooks.io/mastering-spark-sql/spark-sql-bucketing.html

import org.apache.spark.sql.SaveMode

finalDF.write
        .bucketBy(1, "working")
        .sortBy("working", "sex", "age")
        .mode(SaveMode.Overwrite)
        .saveAsTable("timeusage_final_table")
```

```
import org.apache.spark.sql.SaveMode
```

```
// List the tables

spark.catalog.listTables.show
```

```
+--------------------+--------+-----------+---------+-----------+
|                name|database|description|tableType|isTemporary|
+--------------------+--------+-----------+---------+-----------+
|timeusage_final_t...| default|       null|  MANAGED|      false|
+--------------------+--------+-----------+---------+-----------+
```

```
spark.sql("select * from timeusage_final_table").show
```

```
+-----------+------+------+------------+----+-----+
|    working|   sex|   age|primaryNeeds|work|other|
+-----------+------+------+------------+----+-----+
|    working|  male| young|        10.9| 3.7|  9.2|
|    working|female| young|        11.6| 3.3|  8.9|
|    working|female| elder|        10.6| 3.9|  9.3|
|    working|female|active|        11.5| 4.2|  8.1|
|not working|female|active|        12.4| 0.5| 10.8|
|not working|  male| elder|        10.7| 0.7| 12.3|
|not working|female| elder|        10.9| 0.4| 12.4|
|not working|  male|active|        11.4| 0.9| 11.4|
|not working|  male| young|        11.6| 0.2| 11.9|
|    working|  male| elder|        10.4| 4.8|  8.6|
|    working|  male|active|        10.8| 5.2|  7.8|
|not working|female| young|        12.5| 0.2| 11.1|
+-----------+------+------+------------+----+-----+
```

```
val tableDF = spark.table("timeusage_final_table")
```

```
tableDF: org.apache.spark.sql.DataFrame = [working: string, sex: string ... 4 more fields]
```

```
tableDF.show
```

```
+-----------+------+------+------------+----+-----+
|    working|   sex|   age|primaryNeeds|work|other|
+-----------+------+------+------------+----+-----+
|    working|  male| young|        10.9| 3.7|  9.2|
|    working|female| young|        11.6| 3.3|  8.9|
|    working|female| elder|        10.6| 3.9|  9.3|
|    working|female|active|        11.5| 4.2|  8.1|
|not working|female|active|        12.4| 0.5| 10.8|
|not working|  male| elder|        10.7| 0.7| 12.3|
|not working|female| elder|        10.9| 0.4| 12.4|
|not working|  male|active|        11.4| 0.9| 11.4|
|not working|  male| young|        11.6| 0.2| 11.9|
|    working|  male| elder|        10.4| 4.8|  8.6|
|    working|  male|active|        10.8| 5.2|  7.8|
|not working|female| young|        12.5| 0.2| 11.1|
+-----------+------+------+------------+----+-----+
```

```
tableDF.rdd.getNumPartitions
```

```
res89: Int = 1
```

```
tableDF.count
```

```
res90: Long = 12
```

```
/**
  * Models a row of the summarized data set
  * @param working Working status (either "working" or "not working")
  * @param sex Sex (either "male" or "female")
  * @param age Age (either "young", "active" or "elder")
  * @param primaryNeeds Number of daily hours spent on primary needs
  * @param work Number of daily hours spent on work
  * @param other Number of daily hours spent on other activities
  */
case class TimeUsageRow(
  working: String,
  sex: String,
  age: String,
  primaryNeeds: Double,
  work: Double,
  other: Double
)
```

```
defined class TimeUsageRow
```

```
summaryDF.printSchema
```

```
root
```

```
 |-- working: string (nullable = false)
 |-- sex: string (nullable = false)
 |-- age: string (nullable = false)
 |-- primaryNeeds: double (nullable = true)
 |-- work: double (nullable = true)
 |-- other: double (nullable = true)
```

```scala
val summaryDS = summaryDF.as[TimeUsageRow]
```

```
summaryDS: org.apache.spark.sql.Dataset[TimeUsageRow] = [working: string, sex: string ... 4 more fields]
```

```scala
summaryDS.show
```

```
+-----------+------+------+------------------+------------------+------------------+
|    working|   sex|   age|      primaryNeeds|              work|             other|
+-----------+------+------+------------------+------------------+------------------+
|    working|  male| elder|             15.25|               0.0|              8.75|
|    working|female|active|13.833333333333334|               0.0|10.166666666666666|
|    working|female|active|11.916666666666666|               0.0|12.083333333333334|
|not working|female|active|13.083333333333334|               2.0| 8.916666666666666|
|    working|  male|active|11.783333333333333| 8.583333333333334|3.6333333333333333|
|    working|female|active|              17.0|               0.0|               7.0|
|    working|female|active|12.783333333333333| 8.566666666666666|              2.65|
|    working|female| young|               9.0| 9.083333333333334| 5.916666666666667|
|    working|female|active|13.166666666666666|               0.0|10.833333333333334|
|    working|female|active| 6.683333333333334|               4.5|12.816666666666666|
|    working|  male|active| 9.833333333333334|12.133333333333333| 2.033333333333333|
|    working|female|active|12.416666666666666|               0.0|11.583333333333334|
|    working|female|active|11.633333333333333| 6.333333333333333| 6.033333333333333|
|    working|female|active|             12.15|               9.0|              2.85|
|    working|female|active|             13.75|              0.75|               9.5|
|    working|female|active|11.166666666666666|1.0833333333333333|             11.75|
|    working|female| young|11.416666666666666|               0.0|12.583333333333334|
|    working|female|active|              15.8|               0.0|               8.2|
|    working|  male|active| 9.666666666666666|11.616666666666667| 2.716666666666667|
|    working|female|active|              12.1| 7.966666666666667| 3.933333333333333|
+-----------+------+------+------------------+------------------+------------------+
only showing top 20 rows
```

```scala
import org.apache.spark.sql.expressions.scalalang.typed

summaryDS.groupByKey(s => (s.working, s.sex, s.age))
         .agg(typed.avg(_.primaryNeeds), typed.avg(_.work), typed.avg(_.other))
         .toDF("key", "primaryNeeds", "work", "other")
         .select($"key", round($"primaryNeeds", 1).as("primaryNeeds"), round($"work", 1).as("work"), round($"other", 1).as("other"))
         .orderBy($"key")
         .show
```

```
+--------------------+------------+----+-----+
|                 key|primaryNeeds|work|other|
+--------------------+------------+----+-----+
|[not working, fem...|        12.4| 0.5| 10.8|
|[not working, fem...|        10.9| 0.4| 12.4|
|[not working, fem...|        12.5| 0.2| 11.1|
|[not working, mal...|        11.4| 0.9| 11.4|
|[not working, mal...|        10.7| 0.7| 12.3|
|[not working, mal...|        11.6| 0.2| 11.9|
|[working, female,...|        11.5| 4.2|  8.1|
|[working, female,...|        10.6| 3.9|  9.3|
|[working, female,...|        11.6| 3.3|  8.9|
|[working, male, a...|        10.8| 5.2|  7.8|
|[working, male, e...|        10.4| 4.8|  8.6|
|[working, male, y...|        10.9| 3.7|  9.2|
+--------------------+------------+----+-----+

import org.apache.spark.sql.expressions.scalalang.typed
```

```scala
/**
    * @return Same as `timeUsageGrouped`, but using the typed API when possible
    * @param summed Dataset returned by the `timeUsageSummaryTyped` method
    *
    * Note that, though they have the same type (`Dataset[TimeUsageRow]`), the input
    * dataset contains one element per respondent, whereas the resulting dataset
    * contains one element per group (whose time spent on each activity kind has
    * been aggregated).
    *
    * Hint: you should use the `groupByKey` and `typed.avg` methods.
    */
  def timeUsageGroupedTyped(summed: Dataset[TimeUsageRow]): Dataset[TimeUsageRow] = {
    import org.apache.spark.sql.expressions.scalalang.typed
    ???
  }
```