Programming Assignment: Time Usage

You have not submitted. You must earn 8/10 points to pass.

Deadline Pass this assignment by May 26, 11:59 PM PDT

Instructions My submission Discussions

Note: If you have paid for the Certificate, please make sure you are submitting to the required assessment and not the optional assessment. If you mistakenly use the token from the wrong assignment, your grades will not appear

To start, first download the assignment: timeusage.zip. For this assignment, you also need to download the data (164 MB):

http://alaska.epfl.ch/~dockermoocs/bigdata/atussum.csv

and place it in the folder **src/main/resources/timeusage/** in your project directory.

The problem

The dataset is provided by Kaggle and is documented here:

https://www.kaggle.com/bls/american-time-use-survey

The file uses the comma-separated values format: the first line is a header defining the field names of each column, and every following line contains an information record, which is itself made of several columns. It contains information about how people spend their time (e.g., sleeping, eating, working, etc.).

Here are the first lines of the dataset:

```
tucaseid,gemetsta,gtmetsta,peeduca,pehspnon,ptdtrace,teage,telf
s,temjot,teschenr,teschlvl,tesex,tespempnot,trchildnum,trdpftpt
,trernwa,trholiday,trspftpt,trsppres,tryhhchild,tudiaryday,tufn
wgtp,tehruslt,tuyear,t010101,t010102,t010199,t010201,t010299,t0
10301,t010399,t010401,t010499,t010501,t010599,t019999,t020101,t
020102,t020103,t020104,t020199,t020201,t020202,t020203,t020299,
t020301,t020302,t020303,t020399,t020401,t020402,t020499,t020501
,t020502,t020599,t020681,t020699,t020701,t020799,t020801,t02089
9,t020901,t020902,t020903,t020904,t020905,t020999,t029999,t0301
01,t030102,t030103,t030104,t030105,t030108,t030109,t030110,t030
111,t030112,t030186,t030199,t030201,t030202,t030203,t030204,t03
0299,t030301,t030302,t030303,t030399,t030401,t030402,t030403,t0
30404,t030405,t030499,t030501,t030502,t030503,t030504,t030599,t
039999,t040101,t040102,t040103,t040104,t040105,t040108,t040109,
t040110,t040111,t040112,t040186,t040199,t040201,t040202,t040203
,t040204,t040299,t040301,t040302,t040303,t040399,t040401,t04040
2,t040403,t040404,t040405,t040499,t040501,t040502,t040503,t0405
04,t040505,t040506,t040507,t040508,t040599,t049999,t050101,t050
102,t050103,t050189,t050201,t050202,t050203,t050204,t050289,t05
0301,t050302,t050303,t050304,t050389,t050403,t050404,t050405,t0
50481,t050499,t059999,t060101,t060102,t060103,t060104,t060199,t
060201,t060202,t060203,t060289,t060301,t060302,t060303,t060399,
t060401,t060402,t060403,t060499,t069999,t070101,t070102,t070103
,t070104,t070105,t070199,t070201,t070299,t070301,t070399,t07999
9,t080101,t080102,t080199,t080201,t080202,t080203,t080299,t0803
01,t080302,t080399,t080401,t080402,t080403,t080499,t080501,t080
502,t080599,t080601,t080602,t080699,t080701,t080702,t080799,t08
0801,t080899,t089999,t090101,t090102,t090103,t090104,t090199,t0
90201,t090202,t090299,t090301,t090302,t090399,t090401,t090402,t
090499,t090501,t090502,t090599,t099999,t100101,t100102,t100103,
t100199,t100201,t100299,t100381,t100383,t100399,t100401,t100499
,t109999,t110101,t110199,t110281,t110289,t119999,t120101,t12019
9,t120201,t120202,t120299,t120301,t120302,t120303,t120304,t1203
05,t120306,t120307,t120308,t120309,t120310,t120311,t120312,t120
313,t120399,t120401,t120402,t120403,t120404,t120405,t120499,t12
0501,t120502,t120503,t120504,t120599,t129999,t130101,t130102,t1
30103,t130104,t130105,t130106,t130107,t130108,t130109,t130110,t
130111,t130112,t130113,t130114,t130115,t130116,t130117,t130118,
t130119,t130120,t130121,t130122,t130123,t130124,t130125,t130126
,t130127,t130128,t130129,t130130,t130131,t130132,t130133,t13013
4,t130135,t130136,t130199,t130201,t130202,t130203,t130204,t1302
05,t130206,t130207,t130208,t130209,t130210,t130211,t130212,t130
213,t130214,t130215,t130216,t130217,t130218,t130219,t130220,t13
0221,t130222,t130223,t130224,t130225,t130226,t130227,t130228,t1
30229,t130230,t130231,t130232,t130299,t130301,t130302,t130399,t
130401,t130402,t130499,t139999,t140101,t140102,t140103,t140104,
t140105,t149999,t150101,t150102,t150103,t150104,t150105,t150106
,t150199,t150201,t150202,t150203,t150204,t150299,t150301,t15030
2,t150399,t150401,t150402,t150499,t150501,t150599,t150601,t1506
02,t150699,t159989,t160101,t160102,t160103,t160104,t160105,t160
106,t160107,t160108,t169989,t180101,t180199,t180280,t180381,t18
0382,t180399,t180481,t180482,t180499,t180501,t180502,t180589,t1
80601,t180682,t180699,t180701,t180782,t180801,t180802,t180803,t
180804,t180805,t180806,t180807,t180899,t180901,t180902,t180903,
t180904,t180905,t180999,t181002,t181081,t181099,t181101,t181199
,t181201,t181202,t181204,t181283,t181299,t181301,t181302,t18139
9,t181401,t181499,t181501,t181599,t181601,t181699,t181801,t1818
99,t189999,t500101,t500103,t500104,t500105,t500106,t500107,t509
989
"20030100013280",1,-1,44,2,2,60,2,2,-1,-1,1,2,0,2,66000,0,-1,1,
-1,6,8155463,30,2003,870,0,0,40,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,325,0,0,0,0,0,0,0,
```

```
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,200,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0
3   "20030100013344",2,-1,40,2,1,41,1,2,2,-1,2,1,2,2,20000,0,1,1,0,
    7,1735323,30,2003,620,0,0,60,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,60,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,90,0,0,0,0,530,0,0,0,0,0,0,60,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0
4   "20030100013352",1,-1,41,2,1,26,2,2,2,-1,2,2,0,2,20000,0,-1,1,-
    1,7,3830528,12,2003,560,0,0,80,0,0,0,0,0,0,0,0,0,0,0,15,0,180,0
    ,60,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,60,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,75,0,0,0,0,220,0,0,0,0,0,0,120,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0,0,60,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,10,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0
```

Our goal is to identify three groups of activities:

- primary needs (sleeping and eating),
- work,
- other (leisure).

And then to observe how do people allocate their time between these three kinds of activities, and if we can see differences between men and women, employed and unemployed people, and young (less than 22 years old), active (between 22 and 55 years old) and elder people.

At the end of the assignment we will be able to answer the following questions based on the dataset:

- how much time do we spend on primary needs compared to other activities?

- do women and men spend the same amount of time in working?
- does the time spent on primary needs change when people get older?
- how much time do employed people spend on leisure compared to unemployed people?

To achieve this, we will first read the dataset with Spark, transform it into an intermediate dataset which will be easier to work with for our use case, and finally compute information that will answer the above questions.

Read-in Data

The simplest way to create a **DataFrame** consists in reading a file and letting Spark-sql infer the underlying schema. However this approach does not work well with CSV files, because the inferred column types are always **String**.

In our case, the first column contains a **String** value identifying the respondent but all the other columns contain numeric values. Since this schema will not be correctly inferred by Spark-sql, we will define it programmatically. However, the number of columns is huge. So, instead of manually enumerating all the columns we can rely on the fact that, in the CSV file, the first line contains the name of all the columns of the dataset.

Our first task consists in turning this first line into a Spark-sql **StructType**. This is the purpose of the **dfSchema** method. This method returns a **StructType** describing the schema of the CSV file, where the first column has type **StringType** and all the others have type **DoubleType**. None of these columns are nullable.

```
1   def dfSchema(columnNames: List[String]): StructType
```

The second step is to be able to effectively read the CSV file is to turn each line into a Spark-sql **Row** containing columns that match the schema returned by **dfSchema**. That's the job of the **row** method.

```
1  def row(line: List[String]): Row
```

coursera

Project

As you probably noticed, the initial dataset contains lots of information that we don't need to answer our questions, and even the columns that contain useful information are too detailed. For instance, we are not interested in the exact age of each respondent, but just whether she was "young", "active" or "elder".

Also, the time spent on each activity is very detailed (there are more than 50 reported activities). Again, we don't need this level of detail; we are only interested in three activities: primary needs, work and other.

So, with this initial dataset it would a bit hard to express the queries that would give us the answers we are looking for.

The second part of this assignment consists in transforming the initial dataset into a format that will be easier to work with.

A first step in this direction is to identify which columns are related to the same activity. Based on the description of the activity corresponding to each column (given in this document), we deduce the following rules:

- "primary needs" activities (sleeping, eating, etc.) are reported in columns starting with "t01", "t03", "t11", "t1801" and "t1803" ;
- working activities are reported in columns starting with "t05" and "t1805" ;
- other activities (leisure) are reported in columns starting with "t02", "t04", "t06", "t07", "t08", "t09", "t10", "t12", "t13", "t14", "t15", "t16" and "t18" (only those which are not part of the previous groups).

Then our work consists in implementing the **classifiedColumns** method, which classifies the given list of column names into three **Column** groups (primary needs, work or other). This method should return a triplet containing the "primary needs" columns list, the "work" columns list and the "other" columns list.

```
def classifiedColumns(columnNames: List[String]): (List[Column]
    , List[Column], List[Column])
```

The second step is to implement the **timeUsageSummary** method, which projects the detailed dataset into a summarized dataset. This summary will contain only 6 columns: the working status of the respondent, his sex, his age, the amount of daily hours spent on primary needs activities, the amount of daily hours spent on working and the amount of daily hours spent on other activities.

```
def timeUsageSummary(
  primaryNeedsColumns: List[Column],
  workColumns: List[Column],
  otherColumns: List[Column],
  df: DataFrame
): DataFrame
```

Each “activity column” will contain the sum of the columns related to the same activity of the initial dataset. Note that time amounts are given in minutes in the initial dataset, whereas in our resulting dataset we want them to be in hours.

The columns describing the work status, the sex and the age, will contain simplified information compared to the initial dataset.

Last, people that are not employable will be filtered out of the resulting dataset.

The comment on top of the **timeUsageSummary** method will give you more specific information about what is expected in each column.

Aggregate

Finally, we want to compare the *average time* spent on each activity, for all the combinations of working status, sex and age.

We will implement the **timeUsageGrouped** method which computes the average number of hours spent on each activity, grouped by working status (employed or unemployed), sex and age (young, active or elder), and also ordered by working status, sex and age. The values will be rounded to the nearest tenth.

```
def timeUsageGrouped(summed: DataFrame): DataFrame
```

Now you can run the project and see what the final **DataFrame** contains. What do you see when you compare elderly men versus elderly women's time usage? How much time elder people allocate to leisure compared to active people? How much time do active employed people spend to work?

Alternative ways to manipulate data

We can also implement the **timeUsageGrouped** method by using a plain SQL query instead of the **DataFrame** API. Note that sometimes using the programmatic API to build queries is a lot easier than writing a plain SQL query. If you do not have experience with SQL, you might find these examples useful.

```
def timeUsageGroupedSqlQuery(viewName: String): String
```

Can you think of a previous query that would have been a nightmare to write in plain SQL?

Finally, in the last part of this assignment we will explore yet another alternative way to express queries: using typed **Datasets** instead of untyped **DataFrames**.

```
def timeUsageSummaryTyped(timeUsageSummaryDf: DataFrame):
    Dataset[TimeUsageRow]
```

Implement the **timeUsageSummaryTyped** method to convert a **DataFrame** returned by **timeUsageSummary** into a **DataSet[TimeUsageRow]**. The **TimeUsageRow** is a data type that models the content of a row of a summarized dataset. To achieve the conversion you might want to use the **getAs** method of **Row**. This method retrieves a named column of the row and attempts to cast its value to a given type.

```
def timeUsageGroupedTyped(summed: Dataset[TimeUsageRow]):
    Dataset[TimeUsageRow]
```

Then, implement the **timeUsageGroupedTyped** method that performs the same query as **timeUsageGrouped** but uses typed APIs as much as possible. Note that not all the operations have a typed equivalent. **round** is an example of operations that has no typed equivalent: it will return a **Column** that you will have to turn into a **TypedColumn** by calling **.as[Double]**. Another example is **orderBy**, which has no typed equivalent. Make sure your **Dataset** has a schema because this operation requires one (column names are generally lost when using typed transformations).

How to submit

Copy the token below and run the submission script included in the assignment download. When prompted, use your email address **joel.vallone@gmail.com**.

Generate new token

Your submission token is unique to you and should not be shared with anyone. You may submit as many times as you like.