

# PROJECT BIG DATA

## Assignment 2

You will continue to work with the hue data files supplied for Assignment 1, `hue_upload.csv` and `hue_upload2.csv`. The first four columns represent the *row id*, *user id*, *event id*, and *value*. Any extra columns are irrelevant. For example, the first row of one file reads:

```
"1";"10";"lamp_change_29_mai.2015.19.08.33_984";"OFF"
```

As you can see, the event id encompasses both a description of the event ("lamp\_change") and the date/time (May 29, 7:08:33 pm). The following events are considered informative:

String	Description
<code>lamp_change</code>	Light control via app
<code>nudge_time</code>	Automatic light dim time for people in experimental group
<code>bedtime_tonight</code>	Intended bedtime (self-reported)
<code>risetime</code>	Rise time (self-reported)
<code>rise_reason</code>	Reason for rising (self-reported)
<code>adherence_importance</code>	Adherence (self-reported)
<code>fitness</code>	Fitness (self-reported)

All self-reported values are entered around noon. Records with other events may be ignored.

(60 points) The first part of this assignment is to write a Python function `read_csv_data` that reads the data into a Pandas DataFrame. The index should be a (date,user) tuple, where date is stored in `datetime.datetime` format (see the 'Tips' Section below). The columns of your Pandas DataFrame should be `bedtime`, `intended_bedtime`, `rise_time`, `rise_reason`, `fitness`, `adherence_importance` and `in_experimental_group` (note: it is important to *stick to this nomenclature*). The way to do this is by going through the csv data line by line, and parsing each line individually, following these requirements:

**bedtime** should be inferred from the `lamp_change` event. For example, from the row printed above you may infer that the person did not sleep before 7:08:33 pm. As you go through the lines in the csv file, whenever you discover new relevant information, you either update an existing record in the dataframe (if a record for that day and user exists), or you create a new record (see the 'Tips' Section below).

For example, if you encounter a line where user 10 turns the light on at 9 pm and another line where he turns it off at 10 pm (still on May 29), you update the record above to change the bedtime to 10 pm. If someone falls asleep past midnight, the bedtime should be stored in the record corresponding to the day before. Again, dates and times should be stored as `datetime.datetime`.

**intended\_bedtime** should be filled in based on the `bedtime_tonight` event. Note that 1030 probably means 10:30 in the evening. Again, dates and times should be stored as `datetime.datetime`.

**rise\_time** The value for the column in your solution should be obtained from the `risetime` event in the csv file.

**rise\_reason, fitness and adherence\_importance** values should be copied from the csv file. Note that if multiple distinct values are entered, the last should be assumed to be correct.

**in\_experimental\_group** should be boolean (True/False). The default value is False, but should be changed to True if a `nudge_time` event is encountered. If a user is in the experimental group on one day, he is on all days.

(10+20 points) The second part of this assignment is to store the contents of the DataFrame into MongoDB, and to write a function that retrieves data from MongoDB and outputs it in a user-friendly format.

1. The data should be stored in the collection “sleepdata” in the database “BigData”. Make sure to use the same column names as specified for the DataFrame, and to define the correct primary key. See below in the Section ‘Tips’ for some comments about the primary key. Add the extra columns “date”, “user”, “sleep\_duration” to facilitate sorting the data. Here, “sleep\_duration” is the difference between the risetime and the bedtime.
2. The following is an example of how the output must be presented.

date	user	bedtime	intended	risetime	reason	fitness	adh	in_exp	sleep_duration
11-06-2015	2	00:51:28	22:30:00	07:00:00	ja	-	47.0	no	22351
11-06-2015	20	00:28:10	23:00:00	07:10:00	nee	55.0	88.0	yes	33510
11-06-2015	34	19:54:10	-	-	-	-	-	yes	-

Where sleep duration is in number of seconds. Note that, in order to determine the sleep duration of day X, it is necessary to know the risetime of day X, but the bedtime of day X - 1.

You might find nonsensical values, such as 2500 (25:00) as a time of day. This is typical for real-life data. Try to use common sense to interpret such as values (for example by interpreting 25:00 as 01:00 the next day) or discard the entire data point if no interpretation makes sense. There is no single correct answer here, and you will not be graded on how such values are handled.

Another 10 points are awarded for the cleanliness of your code. A total of 100 points can be obtained. The assignment should be done in groups of two students, and must be handed in via Blackboard on June 15 by 23:59h. Your solution should use the provided template (`solution.py`). It is *crucial* that you do not alter the names and arguments of the existing functions, although adding additional functions is recommended. The template contains the file `run_solution.py` that shows how we run your file. Each group should hand in only one solution. If both students submit a solution, only the first submission will be graded. Feedback will be provided through Canvas.

## Frequently asked questions - Pandas

**Do I have to use `hue_upload.csv` or `hue_upload2.csv`, or can I merge them and use the merged file?**

Your program should be able to deal with multiple files separately. The argument `filenames` in the method `def read_csv_data(filenames)` can be a list through which you should iterate (which is a simple a for-loop through `filenames`). Do not adjust any of those two files.

**Where can I find the dates and times in the data?** For the `lamp_change` event, both the date and time are in the third column. For the `rise_time` and `bedtime_tonight` events, the date is in the third column, but the time is in the fourth column, in a different format (e.g., 2300). Note that a value of 0 and 30 correspond to respectively 0:00 and 0:30, making parsing non-trivial. Use regular expressions to find all dates and times in the third column (see the slides from lecture 1 for their usage).

**How do I create a tuple (date, user) index?** See below for an example:

```
date = datetime.datetime(2016, 1, 1, 0, 0, 0)
user_id = 10
index = (date, user_id)
```

See the second tip in the ‘Tips’ section on how to add a new document/row for an index and how to change its values.

**How do I solve the error: KeyError: "[datetime.datetime(2015, 6, 1, 0, 0) '12'] not in index"?** Your code probably includes the line:

```
insert_if_new(...)
```

which should be replaced with:

```
df = insert_if_new(...)
```

This ensures new indexes are added to your DataFrame.

Another cause for this error is that you use `df.get_value(index, ..., ...)` before you created a document for that index, meaning that you are requesting a value from your DataFrame of an index which does not exist yet. To solve this, you should consistently use the `insert_if_new` method to ensure that the index that you ask for, always exists in the DataFrame.

**How to solve the error: TypeError: '>' not supported between instances of 'datetime.datetime' and 'float'?** You get this error probably when you want to check for a `lamp_change` event if a specific date-time you found, is later than the datetime that you already stored (if so, you want to update this). This idea is correct, but the reason for this error is because all default values of a new row have been set to `float('nan')`, and Python cannot compare floats with datetimes.

To solve this, you have to check if the stored `bedtime` value is a float; if so, then you know you have to overwrite it with the datetime value you found anyway. If not, then it has to be a datetime value, and then you can do the comparison that you want.

**How do I get rid of the annoying 'FutureWarning: set\_value is deprecated (...)'?** Add the line

```
import warnings
```

at the header of your code, and the line

```
warnings.simplefilter(action='ignore', category=FutureWarning)
```

at the start of your program.

## Frequently asked questions - MongoDB

**How to solve the error: ServerSelectionTimeoutError: localhost:27017: [WinError 10061] Kan geen verbinding maken omdat de doelcomputer de verbinding actief heeft geweigerd?** For Mac, open the terminal and enter the two lines:

```
brew services start mongod
mongo
```

For Windows, go to `C:\Program Files\MongoDB\Server\3.6\bin` and open `mongod.exe`. This program has to be open to make sure you can use MongoDB.

**How do I initialize a tuple index in MongoDB?** Do this by using a dictionary where you assign a value to the default key `'_id'`. It is not needed to use the method `create_index()`, unless you want to use another name than `'_id'` for your index. Suppose you have an empty dictionary in the variable `x`, then adding a tuple index for our assignment can be done by:

```
x['_id'] = {'date':idx[0], 'user':idx[1]}
```

where `idx` is a (date,id) tuple/index from the earlier created DataFrame object. All other values ('bed-time', etc.) can be added similarly. Afterwards, it is possible to add the dictionary to the database using `sleepdata.insert_one(x)` and the entry on the `'_id'` value is automatically recognized as the index.

**How do I remove the data in my database? It keeps on adding data everytime I run it.** Use `sleepdata.delete_many({})`.

**How do I have to use the filter and sort parameter in the method `def read_mongodb(filter,sort)?`** Basically, you have pass these two parameters through to the `find()` and `sort()` method respectively, which you use on `sleepdata`. Basically, this boils down to using:

```
for doc in sleepdata.find(filter).sort(sort, pymongo.ASCENDING):
```

somewhere in the code. In the for-statement, you ensure that you print the contents of the database nicely. You do not have to do special things with the filter or sort; this is all done for you if you use those two functions.

The following could make the idea of the assignment more clear. If we will test your programs, we might for example use the following in your code:

```
df = read_csv_data(['hue_upload.csv', 'hue_upload2.csv'])
to_mongodb(df)
read_mongodb('sleep_duration': {'$gt': 40000}, '_id')
```

If we use these three lines of code in your program, this means that we will test your program on being able to read in both files, where we want to have all results printed of which the sleep duration is larger than 40000 seconds.

**How do I print the output so nicely as asked in the assignment?** You have to do this manually using the `print()` function. Using `%s`, you can print strings, and by adding a number between the `%` and `s`, you indicate how many symbols/spaces you reserve for this string. If you use less, the remainder will be filled up with whitespaces. We will give you an example where values are printed out nicely, which should be enough for you to find out how to do so for the assignment.

```
print("%10s\t%5s\t%8s" % ('test123', 'x12', 'test1'))
print("%10s\t%5s\t%8s" % ('test4567', 'x3', 'test234'))
print("%10s\t%5s\t%8s" % ('test89', 'x456', 'test56'))
```

gives output:

test123	x12	test1
test4567	x3	test234
test89	x456	test56

## Tips

In the assignments, there are some hurdles that are hard to jump without further help.

- It is important to use the `datetime.datetime` datatype instead of the (seemingly more appropriate) `datetime.date` type, as the following example shows. The following code *does not* run as expected:

```
idx = (datetime.date(2015,1,1),10)
df = df.append(pd.Series({'bedtime':None,'intended_bedtime':None}, name=idx))
if idx in df.index:
    print("surprisingly, this line does not run")
```

The reason is that pandas converts the `datetime.date` to a pandas datetime type. Since a comparison between a date and a datetime is always false, `idx` is not found in `df.index`. So, just stick to the `datetime.datetime` data type for now (we suspect it to be a bug in Pandas that will get fixed some day, but not soon enough for this course). The following code (the change is in the first line) *does run* as expected:

```
idx = (datetime.datetime(2015,1,1,0,0,0),10)
df = df.append(pd.Series({'bedtime':None,'intended_bedtime':None}, name=idx))
if idx in df.index:
    print("This gets printed (as expected)")
```

- The next hurdle is that in some configurations, calling `df.set_value` leads to the error “ValueError: could not convert string to float”. It turns out to be difficult to pinpoint the exact cause of this error. As a workaround, we advise to use the following function:

```
def insert_if_new(df,idx):
    if idx not in df.index:
        df = df.append(pd.Series({'bedtime' : float('nan'), 'intended_bedtime' : float('nan'), \
                                'risetime' : float('nan'), 'rise_reason' : float('nan'), \
                                'fitness' : float('nan'), 'adherence_importance' : float('nan'), \
                                'in_experimental_group' : False}, name=idx))

    return df
```

Mixing float with `nan` ensures that Pandas does not infer incorrect datatypes. Call this function to add a row, and use `set_value` to modify the dataframe.

- The third hurdle is setting the primary key in mongodb. As mongodb does not support tuples (just dicts and lists), the primary key has to be converted from a tuple to a dict (with strings 'date' and 'user' as keys).
- Good luck! Oh, for those in need, there is an official [Pandas cheat sheet](#).