Challenge March-2024

Analyzing Employees

A solution with Python/DT5GL by Jack Jansonius – 4 June 2024

Introduction
Without help from ChatGPT, I would not have submitted a solution to this challenge, as demonstrated in the previous submission, and then also not with my own tool, as is shown in this submission.

As for the use of a third-generation programming language, this does show my preference for Python, because in the prompt for ChatGPT, instead of Python I could just as easily have entered Java or Javascript or PHP or C++ or even a rule engine like Drools, which I did. As I expected, Python code is the most elegant, compact and understandable.

Although slightly less compact, a combination of Python with decision tables may be even easier to understand. After all, answering the questions does not require very complicated decisions. What happens in pseudocode is little more than:

```
For each employee in the JSON file do:
    Increase Total_children with Number_children of this employee
    Increase Total_salary with Salary of this employee
    If employee is Single: increase number of Singles by 1
    If salary of employee > Maximum salary:
                        Maximum salary becomes Salary of this employee
    If salary of employee < Minimum salary:
                        Minimum salary becomes Salary of this employee
    If salary of employee > 80% of Calculated Maximum:
                        - Increase number of highest paid employees by 1
                        - Add name and salary of employee
                          to list of highest paid employees
    For each location of this employee do:
        If state not in states collected:
            add this state to it
        if zipcode not in selected_zip_codes:
            add name of employee + zipcode to
            collected_employees_in_zipcode
```

In theory, one iteration by all employees and one iteration per employee by the employee's (1 or more) locations is sufficient to gather all the information to answer all the questions.

These loops are also found in ChatGPT's solution with Python, but are executed over and over again for answering some questions.

For example, the question in which states the employees live:

```python
# Question 5: States where employees have residences
residence_states = set()
for emp in employees:
    for loc in emp['locations']:
        residence_states.add(loc['state'])
```

But in another session, ChatGPT also comes up with this solution:

```
# States where employees have residences
states = set(loc['state'] for emp in employees for loc in emp['locations'])
```

which, of course, produces exactly the same result.

Very nicely, ChatGPT makes clear with these (instructive!) solutions that you can retrieve the necessary information from a JSON structure with an iteration on employees and within that with an iteration on 1 or more locations per employee.

By defining the collected states as a collection ('set'), states are included in the collection only once, i.e. uniquely.

Now if I want to retrieve all the information in one iteration across all employees I will have to convert ChatGPT's solutions to indexes, and again I can ask ChatGPT to do that:

```
# Question 5: States where employees have residences (using indexes)
residence_states = set()
for i in range(len(employees)):
    for j in range(len(employees[i]['locations'])):
        residence_states.add(employees[i]['locations'][j]['state'])
```

Based on the indexes i (employee_nr) and j (location_nr), the data is now retrieved from the JSON file.

Other examples:
The name of the first employee: employees[0]['name']
```
'Robinson'
```

The street name of the 2nd location of the last employee:
employees[11]['locations'][1]['street']
```
'Morgan Street'
```

All data on the second employee: employees[1]
```
{'age': 45,
 'children': 0,
 'gender': 'Male',
 'locations': [{'id': 'WarnerLoc',
   'state': 'NJ',
   'street': 'Maple Street',
   'zipCode': '08817'}],
 'maritalStatus': 'Married',
 'minor': False,
 'name': 'Warner',
 'salary': 150000.0}
```

And this employee's zip code: employees[1]['locations'][0]['zipCode']
```
'08817'
```

With this background info, the implementation of DT5GL in combination with Python is quite understandable.[1]

[1] For experimenting with Python, for example with suggestions from ChatGPT, I recommend installing Anaconda; see Appendix B

Implementation of the decision model in DT5GL/Python:

```
Initial_instructions:
>>: fn = read_json('Employees.json')        # read employees and selected_zip_codes[2]
>>: total_employees = len_employees()
>>: single_employees = 0
>>: total_children = 0
>>: sum_salary = 0
>>: max_salary = 0
>>: min_salary = 10000000
>>: high_salary = 0.8 * get_max_salary()[3]
>>: number_high_paid = 0
>>: names_high_paid = ""
>>: collected_states = ""
>>: collected_employees_in_zipcode = ""
End_Instructions


Table 0:
If:                                           | 0| 1|
next employee_nr in [0 - lastEmployee]        | Y| N|
Then:
Action is all_employees_handled               |  | X|
Action is update_data_employee                | X|  |
# .......

Attribute: lastEmployee  Type: Integer
Equals: total_employees - 1

rTable 1:
If:                                           | 0| 1| 2| 3| 4|
employee_status = "Single"                    | Y| -| -| -| -|
employee_salary > max_salary                  | -| Y| -| -| -|
employee_salary < min_salary                  | -| -| Y| -| -|
employee_salary >= high_salary                | -| -| -| Y| -|
Then:
Action is increase_singles                    | X|  |  |  |  |
Action is increase_max_salary                 |  | X|  |  |  |
Action is decrease_min_salary                 |  |  | X|  |  |
Action is add_to_high_paid_employees          |  |  |  | X|  |
Action is check_locations                     |  |  |  |  | X|
# .......


rTable 2:
If:                                                    | 0| 1| 2| 3|
next location_nr in [0 - lastLocation]                 | Y| Y| Y| N|
True = location_state not in collected_states          | -| Y| -| -|[4]
True = location_zipcode in get_selected_zip_codes()    | -| -| Y| -|
Then:
Act_Location is all_locations_handled                  |  |  |  | X|
Act_Location is print_location                         | X|  |  |  |
Act_Location is add_state_to_collected_states          |  | X|  |  |
Act_Location is add_employee_to_collected_names        |  |  | X|  |
# .......
```

[2] Here the python function read_json in DTFunctions.py is called; see next chapter. Nothing else is done with the returned value, so fn is a dummy variable here.

[3] For the question, How many people are inside 20% of highest paid employees? multiple interpretations are possible. For example, sort employees descending by salary and take the first 20%. But you can also take 80% of highest salary, and then select all employees who earn more than that. This interpretation has been followed here (unlike ChatGPT's interpretation in the first submission).

[4] In future versions, the addition “True = ” will no longer be necessary.

```
# rTable 1:
Attribute: employee_status
Equals: employee_attribute(employee_nr, "maritalStatus")
Attribute: employee_name       Type: Text
Equals: employee_attribute(employee_nr, "name")
Attribute: employee_salary     Type: Integer
Equals: employee_attribute(employee_nr, "salary")

Attribute: employee_children   Type: Integer
Equals: employee_attribute(employee_nr, "children")

# rTable 2:
Attribute: location_state      Type: Text
Equals: location_attribute(employee_nr, location_nr, "state")
Attribute: location_zipcode    Type: Text
Equals: location_attribute(employee_nr, location_nr, "zipCode")
Attribute: True                Type: Integer
Equals: 1

Attribute: average_children    Type: Real
Equals: total_children / total_employees

Attribute: average_salary      Type: Real
Equals: sum_salary / total_employees

Attribute: residence_states    Type: Text
Equals: collected_states[2:]

Attribute: names_in_selected_zips  Type: Text
Equals: newline() + "   " + collected_employees_in_zipcode[2:] \
        if collected_employees_in_zipcode != "" else "None"

Attribute: fn                  Type: Integer
Attribute: total_employees     Type: Integer
Attribute: high_salary         Type: Integer

Attribute: names_high_paid_print  Type: Text
Equals:    sort_names_high_paid (names_high_paid[2:]) \
           if names_high_paid != "" else "None"
# [2:] => skip first ", "

Attribute: lastLocation        Type: Integer

GOALATTRIBUTE: Action
Repeat_until: all_employees_handled
Multivalued_until: all_employees_handled[5]

Case: all_employees_handled
Print: "Ready!"
Print: "1. Total number of employees: %s"             total_employees
Print: "2. Total children: %s"                        total_children
Print: "   Average children per employee: %s"         average_children
Print: "3. Average salary: %s"                        average_salary
Print: "   Maximal salary: %s"                        max_salary
Print: "   Minimal salary: %s"                        min_salary
Print: "4. Number of single employees: %s"            single_employees
Print: "5. States where employees have residences: %s" residence_states
Print: "6. Number of high paid employees (salary > %s): %s"   high_salary
number_high_paid
Print: "   High-paid employees: %s"                   names_high_paid_print
Print: "7. Employees living in selected zip codes: %s" names_in_selected_zips
```

[5] Why here “Multivalued_until: all_employees_handled”? In the case of a multivalued goal attribute, the reasoning mechanism tries to prove all values for the goal attribute (instead of stopping at the first found value), and you don't want that if the value “all_employees_handled” is detected. For this reason, this condition is also evaluated first based on the order in the decision tables.

```
Case: update_data_employee
Print: "Employee: %s - %s - %s - %s - %s"          employee_nr employee_name
employee_salary employee_children employee_status
>>: total_children = total_children + employee_children
>>: sum_salary = sum_salary + employee_salary

Case: increase_singles
>>: single_employees = single_employees + 1

Case: increase_max_salary
>>: max_salary = employee_salary

Case: decrease_min_salary
>>: min_salary = employee_salary



Case: add_to_high_paid_employees
>>: number_high_paid = number_high_paid + 1
>>: names_high_paid = names_high_paid + ", " + employee_name + "(" +
str(employee_salary) + ")"

Case: check_locations
>>: lastLocation = get_number_of_locations_for_employee(employee_nr) - 1


GOALATTRIBUTE: Act_Location
Repeat_until: all_locations_handled
Multivalued_until: all_locations_handled

Case: all_locations_handled
Print: "#REM# - "

Case: print_location
Print: "              Location: %s - %s - %s"   location_nr location_zipcode
location_state

Case: add_state_to_collected_states
>>: collected_states = collected_states + ", " + location_state

Case: add_employee_to_collected_names
>>: collected_employees_in_zipcode = collected_employees_in_zipcode + ", " +
employee_name + "(" + location_zipcode + ")"
```

Required python code in DTFunctions.py:

```
import json

def read_json(file_path):
    global employees, selected_zip_codes
    with open(file_path, 'r') as file:
        data = json.load(file)
        employees = data['company']['employees']
        selected_zip_codes = data['company']['selectedZipCode']
    return 1

def len_employees():
    return len(employees)

def get_max_salary():
    salaries = [emp['salary'] for emp in employees]
    return max(salaries)

def employee_attribute (employee_nr, attribute):
    return employees[employee_nr][attribute]


# functions if a location decision table is used:
def get_number_of_locations_for_employee(employee_nr):
    return len(employees[employee_nr]['locations'])

def get_selected_zip_codes():
    return str(selected_zip_codes)

def location_attribute(employee_nr, location_nr, attribute):
    return employees[employee_nr]['locations'][location_nr][attribute]

def newline():
    return "\n"

import re

def sort_names_high_paid(data):
    # In: Name1(Integer1), Name2(Integer2), Name3(Integer3),...
    # Out: sorted by descending integer value
    entries = data.split(", ")
    sorted_entries = sorted(entries, key=lambda x: int(re.search(r'\((\d+)\)',
                                                x).group(1)), reverse=True)
    return (", ".join(sorted_entries))[6]
```

[6] Question to ChatGPT: In python, I have the following string: Robinson(220000), White(195000), Doe(210000) Can I sort this one?

ChatGPT: Yes, you can sort this string based on the numbers in the parentheses. To do this, you can use Python's re module to extract the numbers and then sort the string based on them. Below is an example of how to do this: [..]
Question to ChatGPT: That works, but please sort from high to low.
ChatGPT: Of course! To sort the list from high to low, simply add the reverse=True parameter to the sorted function. Here is the custom code: [...]

By the way, the above was entirely in Dutch and I am not making any effort to understand this code further either.

Output:

```
PS C:\..\..\dt5gl> .\dt.exe[7] -source:Analyze_Employees_JSON_v2[8].txt -nti
Employee: 0 - Robinson - 220000 - 2 - Married
              Location: 0 - 08831 - NJ
              Location: 1 - 33019 - FL
Employee: 1 - Warner - 150000 - 0 - Married
              Location: 0 - 08817 - NJ
Employee: 2 - Stevens - 135000 - 0 - Single
              Location: 0 - 08817 - NJ
Employee: 3 - White - 195000 - 2 - Married
              Location: 0 - 33019 - FL
Employee: 4 - Smith - 120000 - 1 - Single
              Location: 0 - 90027 - CA
Employee: 5 - Green - 140000 - 3 - Married
              Location: 0 - 33019 - FL
Employee: 6 - Brown - 65000 - 2 - Married
              Location: 0 - 33019 - FL
Employee: 7 - Klaus - 85000 - 1 - Married
              Location: 0 - 33019 - FL
Employee: 8 - Houston - 135000 - 2 - Single
              Location: 0 - 33019 - FL
Employee: 9 - Long - 40000 - 3 - Married
              Location: 0 - 33019 - FL
Employee: 10 - Short - 120000 - 0 - Single
              Location: 0 - 33019 - FL
Employee: 11 - Doe - 210000 - 1 - Single
              Location: 0 - 33019 - FL
              Location: 1 - 33020 - FL
              Location: 2 - 90027 - CA
Ready!
1. Total number of employees: 12
2. Total children: 17
   Average children per employee: 1.4166666666666667
3. Average salary: 134583.33333333334
   Maximal salary: 220000
   Minimal salary: 40000
4. Number of single employees: 5
5. States where employees have residences: NJ, FL, CA
6. Number of high paid employees (salary > 176000): 3
   High-paid employees: Robinson(220000), Doe(210000), White(195000)
7. Employees living in selected zip codes:
   Warner(08817), Stevens(08817), Smith(90027), Doe(90027)
Time elapsed: 0:00:00.061440
```

[7] DT.exe is Python code, compiled to C, and runs directly under Windows (without pre-installation of Python). Download with many sample scripts is also freely available at: https://github.com/JackJansonius/DT5GL. Possibly, a knowledge group can be started on LinkedIn for answering questions and exchanging experiences. I can be contacted for this at: https://www.linkedin.com/in/jackjansonius/

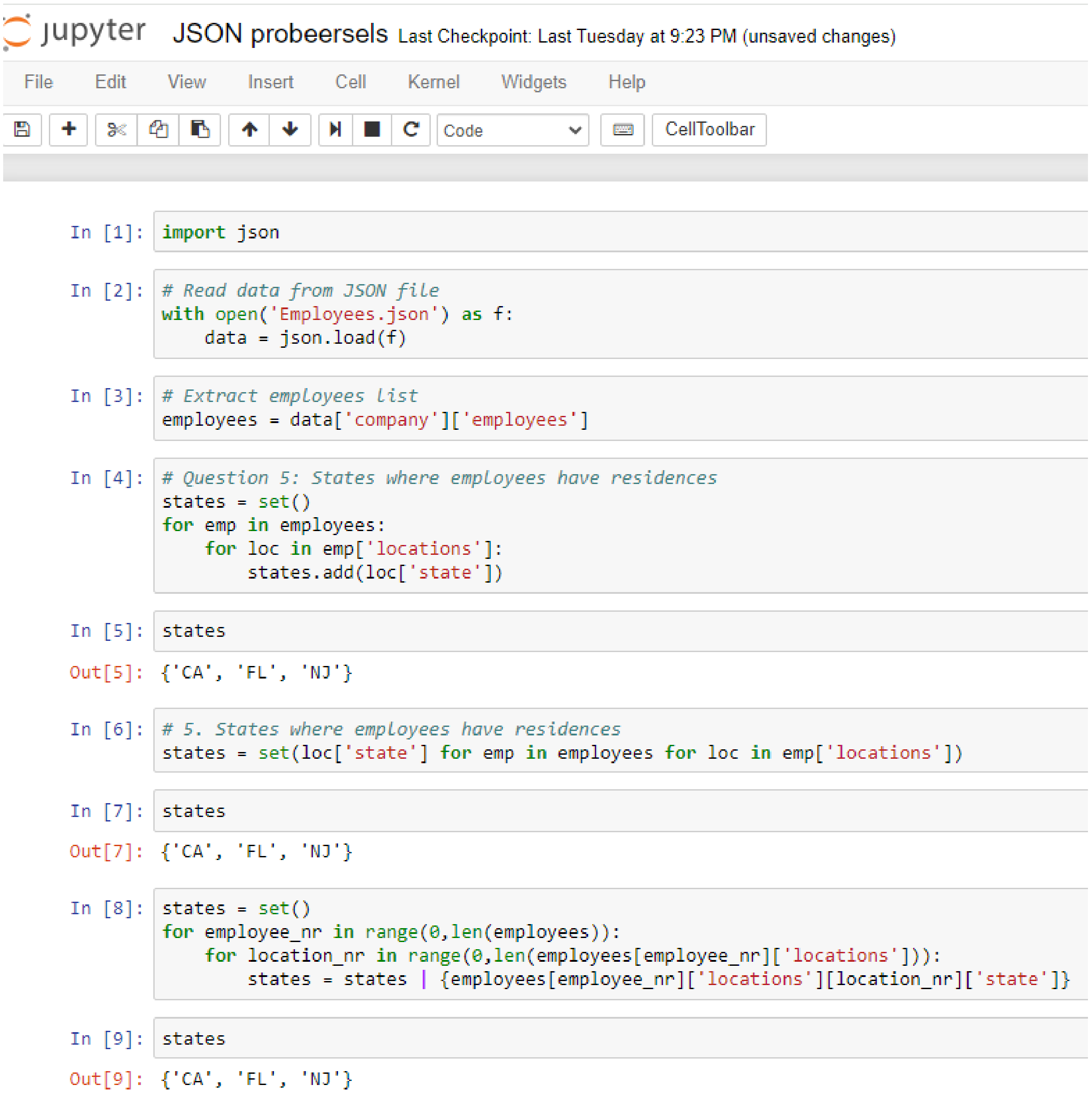
[8] There is also a version 1 where the location data (state and zipcode) is not handled via a decision table but by python functions, see: https://github.com/JackJansonius/DT5GL/blob/main/Analyze_Employees_JSON_v1.txt

Appendix A: Test file 'Employees.json'

```
{
  "company" : {
    "companyName" : "ABC",
    "employees" : [ {
      "name" : "Robinson",
      "age" : 25,
      "gender" : "Female",
      "maritalStatus" : "Married",
      "minor" : false,
      "locations" : [ {
        "id" : "RobinsonLoc1",
        "street" : "Main Str",
        "zipCode" : "08831",
        "state" : "NJ"
      }, {
        "id" : "RobinsonLoc2",
        "street" : "Ocean Drive",
        "zipCode" : "33019",
        "state" : "FL"
      } ],
      "children" : 2,
      "salary" : 220000.0
    }, {
      "name" : "Warner",
      "age" : 45,
      "gender" : "Male",
      "maritalStatus" : "Married",
      "minor" : false,
      "locations" : [ {
        "id" : "WarnerLoc",
        "street" : "Maple Street",
        "zipCode" : "08817",
        "state" : "NJ"
      } ],
      "children" : 0,
      "salary" : 150000.0
    }, {
      "name" : "Stevens",
      "age" : 24,
      "gender" : "Male",
      "maritalStatus" : "Single",
      "minor" : false,
      "locations" : [ {
        "id" : "StevensLoc",
        "street" : "Regency Drive",
        "zipCode" : "08817",
        "state" : "NJ"
      } ],
      "children" : 0,
      "salary" : 135000.0
    }, {
      "name" : "White",
      "age" : 32,
      "gender" : "Female",
      "maritalStatus" : "Married",
      "minor" : false,
      "locations" : [ {
        "id" : "WhiteLoc",
        "street" : "Green Road",
        "zipCode" : "33019",
        "state" : "FL"
      } ],
      "children" : 2,
      "salary" : 195000.0
    }, {
```

```
    "name" : "Smith",
    "age" : 46,
    "gender" : "Male",
    "maritalStatus" : "Single",
    "minor" : false,
    "locations" : [ {
      "id" : "SmithLoc",
      "street" : "Maple Street",
      "zipCode" : "90027",
      "state" : "CA"
    } ],
    "children" : 1,
    "salary" : 120000.0
  }, {
    "name" : "Green",
    "age" : 28,
    "gender" : "Female",
    "maritalStatus" : "Married",
    "minor" : false,
    "locations" : [ {
      "id" : "GreenLoc",
      "street" : "Green Road",
      "zipCode" : "33019",
      "state" : "FL"
    } ],
    "children" : 3,
    "salary" : 140000.0
  }, {
    "name" : "Brown",
    "age" : 32,
    "gender" : "Male",
    "maritalStatus" : "Married",
    "minor" : false,
    "locations" : [ {
      "id" : "BrownLoc",
      "street" : "Green Road",
      "zipCode" : "33019",
      "state" : "FL"
    } ],
    "children" : 2,
    "salary" : 65000.0
  }, {
    "name" : "Klaus",
    "age" : 54,
    "gender" : "Male",
    "maritalStatus" : "Married",
    "minor" : false,
    "locations" : [ {
      "id" : "KlausLoc",
      "street" : "Green Road",
      "zipCode" : "33019",
      "state" : "FL"
    } ],
    "children" : 1,
    "salary" : 85000.0
  }, {
    "name" : "Houston",
    "age" : 47,
    "gender" : "Female",
    "maritalStatus" : "Single",
    "minor" : false,
    "locations" : [ {
      "id" : "HoustonLoc",
      "street" : "Green Road",
      "zipCode" : "33019",
      "state" : "FL"
    } ],
    "children" : 2,
      "salary" : 135000.0
    }, {
      "name" : "Long",
      "age" : 29,
      "gender" : "Male",
      "maritalStatus" : "Married",
      "minor" : false,
      "locations" : [ {
        "id" : "LongLoc",
        "street" : "Green Road",
        "zipCode" : "33019",
        "state" : "FL"
      } ],
      "children" : 3,
      "salary" : 40000.0
    }, {
      "name" : "Short",
      "age" : 22,
      "gender" : "Male",
      "maritalStatus" : "Single",
      "minor" : false,
      "locations" : [ {
        "id" : "ShortLoc",
        "street" : "Green Road",
        "zipCode" : "33019",
        "state" : "FL"
      } ],
      "children" : 0,
      "salary" : 120000.0
    }, {
      "name" : "Doe",
      "age" : 21,
      "gender" : "Female",
      "maritalStatus" : "Single",
      "minor" : false,
      "locations" : [ {
        "id" : "DoeLoc1",
        "street" : "Green Road",
        "zipCode" : "33019",
        "state" : "FL"
      }, {
        "id" : "DoeLoc2",
        "street" : "Morgan Street",
        "zipCode" : "33020",
        "state" : "FL"
      }, {
        "id" : "DoeLoc3",
        "street" : "Lyric Ave",
        "zipCode" : "90027",
        "state" : "CA"
      } ],
      "children" : 1,
      "salary" : 210000.0
    } ],
    "selectedZipCode" : [ "08817", "90027" ]
  }
}
```

Appendix B: Jupyter Notebook (Anaconda)


jupyter
JSON probeersels Last Checkpoint: Last Tuesday at 9:23 PM (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Code
CellToolbar

```
In [1]: import json

In [2]: # Read data from JSON file
        with open('Employees.json') as f:
            data = json.load(f)

In [3]: # Extract employees list
        employees = data['company']['employees']

In [4]: # Question 5: States where employees have residences
        states = set()
        for emp in employees:
            for loc in emp['locations']:
                states.add(loc['state'])

In [5]: states

Out[5]: {'CA', 'FL', 'NJ'}

In [6]: # 5. States where employees have residences
        states = set(loc['state'] for emp in employees for loc in emp['locations'])

In [7]: states

Out[7]: {'CA', 'FL', 'NJ'}

In [8]: states = set()
        for employee_nr in range(0,len(employees)):
            for location_nr in range(0,len(employees[employee_nr]['locations'])):
                states = states | {employees[employee_nr]['locations'][location_nr]['state']}

In [9]: states

Out[9]: {'CA', 'FL', 'NJ'}
```