Data Analysis and Programming for Operations Management 2020-2021

Technology & Operations Management

Faculty of Economics & Business

University of Groningen

Individual Assignment

Summary of requirements

This is an individual assignment.

The deadline of this assignment is October 26, 09.00AM.

An oral exam is part of the assessment.

The oral exams will be scheduled in the exam weeks.

You can enroll for a time slot for the oral exam through Nestor. It will be announced through Nestor when the enrollment opens.

The maximum number of pages of your report (excluding pictures, tables, and code) is 3.

1 Introduction

In the individual final assignment for DAPOM you will help a bicycle courier company in their capacity and scheduling decisions. For the assignment, you need to write a concise report that includes the choices you made and the activities that followed from those choices. The report should not be just a collection of notes; it needs to have a proper flow with introduction, conclusions, graphs, etc. The final source code needs to be properly formatted and included in the appendix. You can upload the report on Nestor.

The assessment criteria for the assignment are the following:

- Solid reasoning explaining modelling choices
- Appropriate use of the techniques and Python code
- Use of Gurobi
- Quality of the report
- Quality of the oral defense

In the final assignment you will show that you master the various programming skills that were practiced in the previous weeks. You will work with CSV files, data structures, Elasticsearch, Gurobi, etc.. Note that

many roads lead to Rome. The techniques and libraries used during the practicals are sufficient to answer the various questions in this assignment, but you are also allowed to use additional libraries.

Please consider that helping each other is allowed, but that the assignment is individual. Hence, plagiarism is not allowed. You need to make your own choices, decisions, and visualizations, and defend them in your report and oral exam.

During the oral exam we will test your understanding about your own report and programming code, but also your understanding of the Dapom topics in general. The questions at the oral exam largely depend on your report and code. You must be able to explain the goals and functionality of the libraries you used. You are, however, not expected to memorize all parameters of Gurobi or Elasticsearch libraries. Typical examples of questions are:

- Where in your program do you determine the distance between two postcodes?
- Could you explain the code on the top part of page 8 of your report?
- How could you extend your procedure on page 4 to include weather forecasts?
- How would your Gurobi model change if it also should consider that an 8 hour shift needs a break of at least half an hour.
- <teacher showing a piece of code from the report of a fellow student>. Could you explain what this piece of code is doing?

The oral exam will take half an hour. The exam will be done with two students simultaneously to entice discussion. You are not allowed to do the oral exam at the same time slot as your current team mate.

A working webcam and microphone are mandatory during the oral exam!

2. Case description

The company you need to advice delivers meals from restaurants to home addresses in the city of Groningen. The demand for the companies' services is increasing the past years but very dynamic. For example, on Friday more orders are placed than on Tuesday, and on rainy days it is busier than on sunny days. Luckily, bicycle couriers (who are mainly students) can be called at short notice so the company is quite flexible; they can determine each day how many couriers are needed for the next day. Still, the company would like to get more grip on the number of bicycle couriers they hire and at what time during the day they are needed. The company thinks that past order data can be used to predict each day how many couriers they should hire for the next day and has made this data available for analysis.

The assignment consists of three steps:

- Step 1: Create load profiles for days. A load profile shows the cumulative hours of travelling time needed per hour.
- Step 2: Based on the load profiles, optimize the hiring of couriers.
- Step 3: Analyze to what extent combining orders could be advantageous.

3. Data

You have two files available:

- groningenPostcodeDistances.csv: the travelling time (in seconds) and distance (in meters) from each postcode in Groningen to all other postcodes in Groningen. Normally you would calculate the distance and predicted traveling time between locations on the basis of actual addresses. Still, using postcodes is considered to be accurate enough to determine realistic time and distance between locations, because each postcode on average consists of only 15 houses. Note that this still is a large file with almost 30.000.000 distances. It was generated with the Open Source Routing Machine (http://project-osrm.org/) using a bicycle profile.
- orders.csv: this contains all the bicycle courier orders received by restaurants in 2018 in Groningen. Each order line contains the date/time, the name of the restaurant, the destination address, the postcode of the restaurant, the postcode of the destination, the distance in meters, and the time to travel in seconds (which we extracted from groningenPostcodeDistances.csv). Note that, for reasons of privacy, this data is randomly generated, fictitious and not based on real data in any way.

4. Data Analysis questions

To start, look at the structure of the data files and determine the meaning of each column.

Second, import the both files in Elasticsearch. Before you start the import, specify the kinds of fields that are in the data file, to make sure that the time field is indeed indexed as time (for an example, see section 5.5 of the B-practical manual how this was done for the taxi-file). Then, use the `elasticsearch_loader` program you also used in the practical, but now specify you are importing a csv file, for example:

```
elasticsearch_loader --index dapom_orders --bulk-size 5000 --encoding "ansi"
csv "./orders.csv"
```

Adapt the path to the location where you stored the file, and execute the command for both files (make sure you give both files their own index, so you don't accidentally add the postcode file to the orders index).

The file with postcodes is quite big, so importing it will take some time. On my reasonably fast laptop it took approximately 30 minutes. If it takes long, and you don't trust whether the import is actually running, go to:

```
http://localhost:9200/insert-your-index-name-here/_count
```

After a minute or so, press F5. The count should increase. The total count for the postcode file should be around 29.000.000.

Tips:

- Map the date/time field in the order file to the elastic `date` type with format `yyyy-MM-dd HH:mm:ss`
- The files might be downloaded encoded as ANSI, so make sure to add the parameter `--encoding 'ansi'` to the `elasticsearch_loader` command.

Step 1. Clearly, on some days more orders are sent than on others, so if you determine a hiring strategy, predicting the number of orders is very important. If the hiring policy is based on overall averages, you almost always end up with too many or too few couriers. Therefore, search for patterns in the orders to identify *types of days*. You can look at the data yourself and use common sense reasoning, and use aggregations in Elasticsearch to detect patterns.

Tips:

- If you mapped the date/time of the order in Elasticsearch to a date field with the correct format, you can use aggregations in Elasticsearch on date/time oriented slices (e.g., the number of orders per month, the sum of travel times per hour, etc.). Start with aggregating the number of orders per date using Elasticsearch. Look at the results; especially look at the differences between weekdays and weekend-days, and national Dutch holidays. Weather could also have been an influence.
- You can also aggregate over days instead of dates, for example, giving back the summation of orders on all Mondays, Tuesdays, Wednesdays, etc. Because the dayOfWeek is not in the index itself, a script (small program) is needed to extract the day of the week when the query is executed. Here is an example to get the total amount of orders per day of the week:

```
search_body = {
      "aggs": {
            "total_orders_per_day_of_week": {
                  "terms": {
                        "script": {
                              "lang": "painless",
                              "source": "doc['dateTime'].value.dayOfWeek
                        }
                  }
            }
      }
}
```

Include in your report the average number of orders per day for the types of days that you distinguished.

Now create a load profile (both for the number of orders per hour and the travel time needed) for each of the types of days identified, and also include this in your report. This can be done in multiple ways, for example:

- Combine a filter query with an aggregation with buckets for each hour. The aggregation is then applied only to the records that meet the query. You can then do this for each type of day (e.g., Sundays, or Fridays, or Kings day, etc.).
- If your day types are based on an aggregation (such as the dayOfWeek example above), you can create a sub-aggregation, where you can aggregate over the hours within the dayOfWeek buckets.

Step 2. The bicycle courier company would on each day like to know how many bicycle couriers they need to call for the next day. For each courier, they can choose from two kinds of shifts: 4 hours or 8 hours. 4 hour shifts are a bit more expensive per hour, but give more flexibility. An 8 hour shift costs them E9 per hour, and a 4 hour shift costs them E10 per hour. A shift always starts at the whole hour (08.00, or 09.00, or 10.00, etc.)

Calculate *using Gurobi* the optimum number of bicycle couriers, how many of them should work respectively a 4 hour or 8 hour shift, and at what time they should start. Do this for each of the kinds of days you have determined in step 1.

Tips:

- From the load profile, you already know the time window for your delivery service, for example between 08.00 in the morning until 22.00 in the evening.
- In your model, make two decision variables *per* hour. The first decision variable indicates the number of 4 hour shifts that starts at that hour, the second indicates the number of 8 hour shifts that starts at that hour. Note that the number decision variables is less than the number of hours in your time window; if your latest order is placed at 22.45, then the last 4 hour shift starts at 19.00, and the last 8 hour shift starts at 15.00. After all, it would make no sense to start an 8 hour shift at 20.00, because that courier would have nothing to do after 22.45.
- Your constraints indicate that the number of couriers available at each hour must at least be the demand as determined in your load profile. So if in your load profile you have 12 hours of work (travelling time) starting between 18.00 and 19.00, you need at least 12 couriers (and it does not matter whether these couriers run 4 hour or 8 hour shifts, or at what time they started, as long as 18.00-19.00 is in their shift window).
- The goal is to minimize the cost, i.e., the number of 4 hour shifts x 4 x E10 + the number of 8 hour shifts x 8 x 9

You do not schedule individual routes to couriers in this assignment. Rather, you make decisions based on aggregate information. Therefore, you must make some assumptions here, for example that riding back also takes time, and that you neglect that in practice, couriers would not start a route just before their shift ends.

Include in you report the results of your model (decision variables and total cost per day type). Include the code for the model in the appendix.

Step 3. If two different restaurants both get an order at approximately the same time, and the destinations are close to each other, two couriers are sent. If the restaurants themselves are close to each other as well, however, it might be sufficient to just send one courier that delivers both meals. So, the bicycle courier company thinks that giving multiple orders to a courier could be advantageous but they do not know if enough orders would meet the requirement that time, restaurants, and destinations are close to each other.

Tip: to determine an optimal solution is challenging. Luckily, a simple non-optimal approach can already provide insights whether the idea of bundling orders is worthwhile. An easy way is to:

- Pick a time slot (e.g., a busy hour) and load all orders for that time slot in a set in Python (set A). Make sure the orders in this set are sorted on time.
- For each order from set A, get order A, and then:
    - iterate through the rest of set A, get each respective order and call it order B. Compare the time between the order A and order B. If the difference is less than x minutes (you can experiment with different values of x):
        - Lookup the traveling time between the two corresponding restaurants using Elasticsearch in groningenPostcodeDistances.
          If less than y minutes (you need to try different values of y):
            - Lookup the traveling time between the destinations
              If less than z minutes (you need to try different values of z):
                - Add order A and order B to the set with combined orders (set B), and remove order B from set A.

The orders in set B now contain all combined orders. Compare how many orders can be combined on busy versus quit hours for different values of x, y, and z, and how many bicycle couriers you would need less (if any).

Make sure that you put your code in a function in which the time slot, the maximum time between orders, the maximum travelling time between restaurants, and the maximum travelling time between destinations are parameterized, so you can perform what-if analyses for different time slots and timing settings.

Bonus question

It would be nice if you could visualize the bundled orders. This is easily possible using Google Maps by letting your Python code create Url's that you can copy paste in your browser, for example:

https://www.google.com/maps/dir/?api=1&origin=9722AD&destination=9712BE&travelmode=bicycling&waypoints=9721AH|9711KE

During the practicals, however, you worked with Smopy and/or Folium, which gives you much more control over the visualization. 5 points will be equally divided between all students who can:

- Calculate the difference in total distance travelled with the combined orders (set B and remainder of set A) compared to not combining (original set A)
- successfully show the routes of all bundled orders from set B on a Smopy or Folium map (the routes may be drawn as straight lines).

The text in your report regarding the bonus question does not count towards the maximum of 3 pages.