DSA2101 mid-term AY22/23 Sem 1

2022-10-08

Contents

Question 1 (10 marks)

This question involves using API calls to download data on *Visitor International Arrivals to Singapore* from data.gov.

When we make GET calls to an API, we are often returned result in batches. This is to avoid crashing the requester's program by sending too much data at once. The information to retrieve the *next* batch is usually contained in the response from the API.

Question 1.1 (3 marks)

1. For this particular endpoint, the final result is that you should have a dataset named `visitor_count` with 455 rows and the following 2 columns:

```
visitor_count
```

```
## # A tibble: 455 x 2
##    arr_month  total_visitors
##    <date>              <int>
##  1 1978-01-01         167016
##  2 1978-02-01         147954
##  3 1978-03-01         163199
##  4 1978-04-01         162400
##  5 1978-05-01         162667
##  6 1978-06-01         149896
##  7 1978-07-01         175968
##  8 1978-08-01         201355
##  9 1978-09-01         167980
## 10 1978-10-01         177639
## # ... with 445 more rows
```

The following code will retrieve the 455 rows containing the monthly counts of visitor arrivals to Singapore since Jan 1978. Fill in the blanks to get the code working.

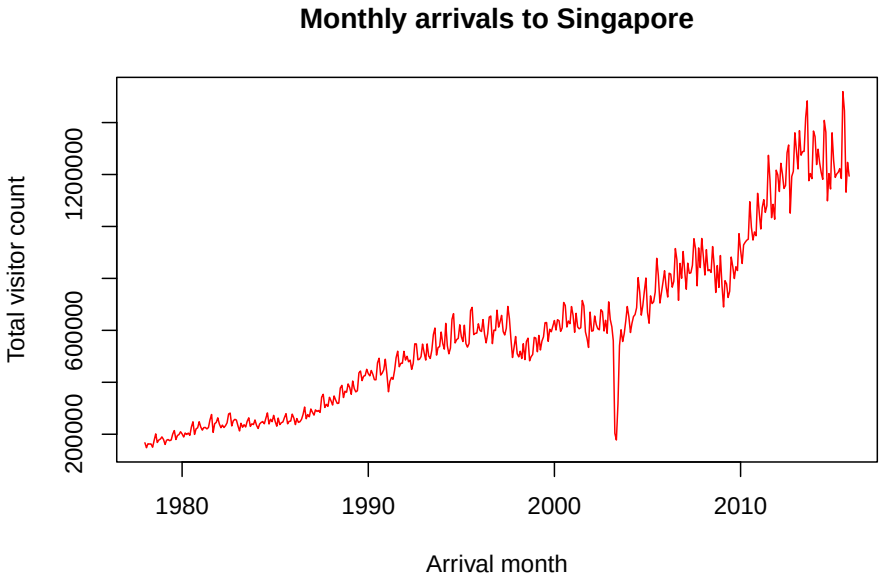
```
library(httr)

url1 <- paste0("https://data.gov.sg/api/action/datastore_search?",
               "resource_id=f8c014e4-fc08-4e28-a1b8-27a8390afd1e")
get_response <- _____(url1, verbose())
tmp_content <- _____(get_response)
visitor_count <- NULL

while(NROW(visitor_count) < _____) {
  total_visitors <- _____
  arr_month <- _____
  visitor_count <- bind_rows(visitor_count, tibble(arr_month, total_visitors))

  new_url <- _____
  get_response <- GET(new_url, verbose())
  tmp_content <- content(get_response)
}

## If you need to do any additional modifications to the tibble, you may do so
## here:
```

Question 1.2 (2 marks)

2. Is it appropriate to use a `while` loop above as opposed to `sapply` or `lapply`? Why or why not? Please explain your answer in not more than 100 words in a markdown section entitled "`## Question 1.2`"

Question 1.3 (4 marks)

3. When we create a plot of the full dataset, it should look like this:

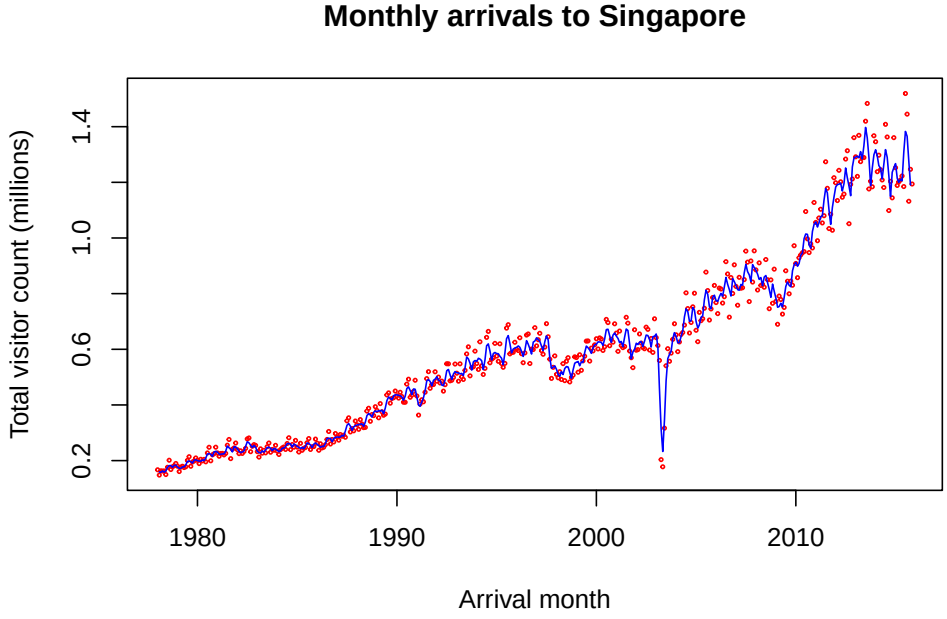


However, there is clearly a great deal of noise in the data. To smooth out noise in observations over time, we sometimes compute a moving average of the points. For a set of 3 points x_1, x_2 and x_3 observed at times 1,2 and 3, the Moving Average y_2 at time 2 is computed as

$$y_2 = \frac{x_1 + x_2 + x_3}{3}$$

Add a column containing the moving average of visitor count to `visitor_count` and re-create the following plot, **matching all labels and values.**

- *Hint:* You can use the `filter` function from the `stats` package to help you.
- *Note:* You won't be able to compute the Moving Average for the first and last points.



Question 1.4 (1 mark)

4. The Moving Average of the visitor counts can be treated as "fitted" values. Extract the months with the 10 largest residuals (in absolute value), and store them in a vector `q1_4_vec`.

Question 2 (14 marks)

The file `mid_term_q2.rdt` contains 2 tibbles that contain data on customer transactions on an online shopping website in Brazil.

The `prod_cat` tibble contains columns describing the products listed on the website:

- `product_id`: a unique identifier for the product.
- Columns 2, 3 and 4 describe the number of characters used to describe the product, and the number of photos of it, on the product page.
- Columns 5 - 8 contain the physical dimensions of the product.
- Column 9 contains a categorisation of the product.

The `orders` tibble consists of information on online orders made by customers to the website. The columns are:

- `customer_id`: a unique customer identifier.
- `customer_state`: a two-letter abbreviation of the state that the customer is from.
- Columns 4 - 6 contain information on when the customer made the purchase, when the item was actually delivered, and an estimated delivery date (which is given to the customer upon purchase).

Question 2.1 (1 mark)

1. Count the number of items in each product category and store the counts in a tibble `q2_1_tbl`. The tibble should only have two columns.

Question 2.2 (3 marks)

2. Select the product category column and all physical dimension columns of each product using three different `select` expressions. (You don't have to worry about re-ordering the columns.)

   The operators in each call must be completely disjoint.

   Store your expressions in a code chunk entitled `q2_2_code_chunk`.

Question 2.3 (4 marks)

3. From the `prod_cat` table,
   - convert all the cm columns to inches (1cm = 0.3937in),
   - then compute the area of each product (length x width) in square inches,
   - then compute the mean area for each product category. Return those *product categories* whose mean area in sq. inches is between 300 and 310 (inclusive). Store your output in a tibble named `q2_3_tbl`.

Question 2.4 (2 marks)

4. A late order is one where the customer receives the item after the estimated delivery date. What is the proportion of late orders in each state? Store your tibble, which should have two columns and be in decreasing order of proportion of late orders, as `q2_4_tbl`. There should be one row for each state in the data.

Question 2.5 (4 marks)

5. A marketing analyst claims that "In all states, purchases are equally likely to be on any day of the week". Do you agree with this claim? Justify your answer with code, a plot, and text explanations in a section entitled "`## Question 2.5`". Please try to keep your text explanation to less than 200 words.

- *Note*: In this question, you are being tested on how well you can translate a question of interest into queries.

Question 3 (6 marks)

In assignment 2, we downloaded radar scans from NEA. Such radar scans are used by meteorologists to monitor weather conditions. The colours on the scans indicate regions of high moisture (i.e. most likely clouds).

Suppose that you have access to several years of (grayscale) timestamped radar scans, and you have been tasked with studying storm patterns in the region, using these scans. This animated gif gives an indication of the kind of analysis expected of you. It is hoped that at the end, you will be able to identify storms from a scan, and even predict the path of the storm.

Each red polygon in the gif denotes a storm at a timepoint. Notice that, as time passes, storms can get bigger or smaller. They can even split and merge with one another.

Within R, each radar scan is stored as a 480x480 matrix, with each cell containing a numeric value between 0 and 1 to indicate the moisture content. Most entries in the matrix will be zero. Storms are usually identified as contiguous cells above some threshold value.

Here is an example of the data for a single scan, after reading into R. The file `mid_term_q3.rdt` contains an example of such a radar scan. It can be plotted with the following code.

```
load("../data/mid_term_q3.rdt")
plot(c(0,480), c(0, 480), type="n", asp=1)
rasterImage(radar_scan, 0, 0, 480, 480)
```

Propose an **S3 class** you would define to contain this data, and propose **two** methods you would write for it. Put your answer in a section entitled "`## Question 3`", and please keep your answer to less than 300 words.

You do not have to write any code for this question. Your task is to think about the following issues:

- For instance, would you define a `storm` class, or a `radar_scans` class?
- What inputs would go into the function that creates objects of your proposed class?
- What tasks will the constructor function have to perform to process the inputs?
- What information would the proposed class contain? Can you describe how they will be stored in terms of R data structures?
- What arguments will the methods you propose take? What output will they produce?
- How will these methods help in your work as the analyst?

Question 4 (11 marks)

Acme Bank is a small bank that operates with a single bank teller from 9am to 6pm (540 mins) every weekday. The teller works throughout lunch (12noon to 1pm), and the bank closes at 6pm sharp. Customers who are in the queue at this time will have to leave and come back another day. They are considered *unserved*.

Over a period of 100 days, data has been recorded on the arrival times, departure times and service times of customers. The data is stored in two tibbles in `mid_term_q4.rdt`.

The first tibble, `mon_arrivals`, contains information on the arrival times of customers to the bank.

- `arr_time` is the time (in minutes after 9am) that the customer arrived.
- `dep_time` is the time at which the customer departs the bank.
- `service_time` is the amount of time spent at the counter.
- `finished` is a Boolean variable indicating if the customer completed his service before the bank closed. If this is FALSE, it means that the customer did not reach the counter before 6pm; he/she had to leave and come back the next day.

- `day` denotes the day on which data was recorded.

The second tibble `mon_resources` contains information on the evolution of the situation within the bank. Suppose the first few rows in the table are as follows:

resource	time	server	queue	day
Counter	11.3	1	0	1
Counter	22.4	1	1	1
Counter	28.4	1	0	1
Counter	38.2	0	0	1

The table indicates that:

- At time 11.3, a new customer arrived, and began service. The server (i.e. the teller) now has one person being served.
- At time 22.4, another customer arrived, and joined the queue. The previous customer has not left, which is the value in the `server` column is still 1 in row 2. The `queue` column is now 1 because the new arrival will have to wait in line.
- At time 28.4, the first customer leaves. The second customer begins service, which results in the `queue` column taking the value 0. The 1 in the `server` column corresponds to the second customer being served by the teller.
- At time 38.2, this second customer completes services and departs the bank. At this point, there are no customers in the bank.

The rows in the `mon_arrivals` table will corroborate the information in the `mon_resources` table. For instance, you would see the following rows there:

name	arr_time	dep_time	service_time	finished	day
Customer0	11.3	28.4	17.10	TRUE	1
Customer1	22.4	38.2	9.82	TRUE	1

Question 4.1 (1 mark)

1. Compute the number of unserved customers for each day. Store your information in a tibble `q4_1_tbl`.

Question 4.2 (3 marks)

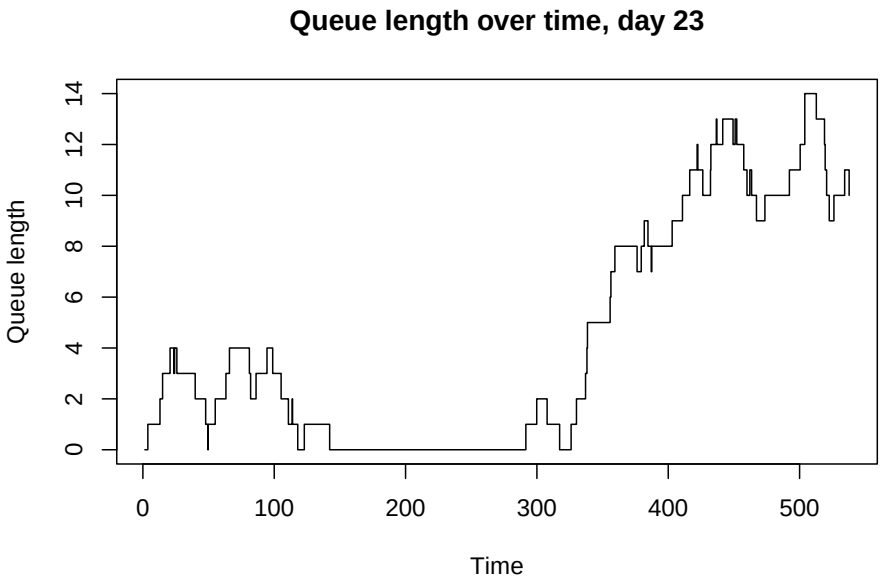
2. Suppose that the manager is considering keeping the bank open until all customers who arrived before 6pm have been served. For each day, fill in the service time for the unserved customers to be the average service time for that day. Compute the mean amount of **extra** time (in minutes) the bank would need to be kept open (beyond 6pm), over the 100 days. Store your answer as `q4_2_scalar`.

Question 4.3 (3 marks)

3. Compute the total amount of time on each day for which there were 10 or more people in the queue. Store your answer as a vector `q4_3_vec`.

Question 4.4 (2 marks)

4. Create a plot of how queue length evolves for the 23rd day. Your plot should appear similar to this.
    - *Hint*: This is known as a stairs plot. Please take a look at the help page for `plot()`.



Question 4.5 (2 marks)

5. *For each day*, compute the time-averaged queue length. The time-averaged queue length is computed as the area underneath the graph above, divided by 540 minutes. Store your answer as a vector `q4_5_vec`.

Question 5 (10 marks)

The file `bdates.csv` contains information on birthdays of 250 students. The file `mid_term_q5.rdt` contains the three candidate pmfs shown in the table below.

Question 5.1 (5 marks)

1. Clean the dataset and store it as `q5_1_tbl`. None of the rows should be removed, and each column should be in a standardised format, and in an appropriate data type.

Question 5.2 (2 marks)

2. Estimate the proportion of students born in each month. Store your answer as a vector of length 12 in `q5_2_vec`.

Question 5.3 (3 marks)

3. In tutorial 1, we assumed birthdays were uniformly distributed over the year. There is suspicion that the birthdays in this data are not uniformly distributed. Compare the observed counts in each month from the data, to expected counts from the following pmfs, decide which pmf below is most appropriate for the given data (out of the three), and then create a rootogram with the best-fitting pmf. In deciding, be quantitative even if it seems "obvious" which is the best fitting distribution.

month	pmf1	pmf2	pmf3
1	0.1864295	0.0196759	0.0830939
2	0.0812910	0.0543981	0.1801306
3	0.0656129	0.0821759	0.2028400
4	0.0584932	0.1030093	0.1827696
5	0.0548731	0.1168981	0.1429067

month	pmf1	pmf2	pmf3
6	0.0533004	0.1238426	0.0988843
7	0.0533004	0.1238426	0.0601869
8	0.0548731	0.1168981	0.0313555
9	0.0584932	0.1030093	0.0131940
10	0.0656129	0.0821759	0.0039742
11	0.0812910	0.0543981	0.0006420
12	0.1864295	0.0196759	0.0000224

Reminders

1. Has your code generated:
   - `visitor_count` (tibble), section `Question 1.2`, a plot, `q1_4_vec`,
   - `q2_1_tbl`, code chunk `q2_2_code_chunk`, `q2_3_tbl`, `q2_4_tbl`, section `Question 2.5`
   - section `Question 2.3`
   - `q4_1_tbl`, `q4_2_scalar`, `q4_3_vec`, a plot, `q4_5_vec`,
   - `q5_1_tbl`, `q5_2_vec`, a plot, section `Question 5.3`