Exercises: How profitable is your online shop?

Michael T. Gastner

1 Creating vectors

Suppose you started a small online shop last week. Table 1 shows the number of daily customers and the daily sales. You are trying to figure out your operational strategy based on these data.

(a) Create two vectors to represent the two columns: `customers` and `sales`. Implement `customers` as an integer vector.
(b) How can you confirm the class and number of elements in each vector?

2 Vector operations and subsetting

(a) Generate a new vector called `sales_per_customer` to store the average sales per customer on each day.

(b) You just double-checked your accounts, and you discovered that you incorrectly entered the sales income for Tuesday as \$137.88 instead of \$317.88. Oops! Please correct this value in the `sales` vector with vector subsetting (i.e. do not generate the whole vector again). Afterwards, re-calculate the `sales_per_customer` vector.

(c) Calculate the total number of customers during this week and the total weekly sales.

(d) Calculate the mean daily sales on the weekend (i.e. Saturday and Sunday) only.

(e) Calculate the mean daily sales on weekdays (i.e. Monday to Friday) in two different ways:

- Select 'positively' those indices that correspond to weekdays.
- Select 'negatively' those indices that correspond to weekends.

(f) Sales during the first two days were low. You are wondering whether it would have been better to pay a freelance web designer for work on the weekend. She would have charged you \$50, and you would not have sold anything that weekend. However, you expect that sales on weekdays would have been up by 10%. Would the additional sales have been bigger than the cost for the web designer?

Table 1: Daily number of customers and sales during the first week of operations.

Day	**Customers**	**Sales (in \$)**
Saturday	33	65.05
Sunday	22	80.17
Monday	41	326.53
Tuesday	39	137.88
Wednesday	38	374.34
Thursday	46	329.04
Friday	47	251.29

Table 2: Predicted additional customers per day compared to the no-advertisement scenario and the corresponding costs of advertising packages.

Package	Additional customers per day	Weekly cost (in $)
1	2	50
2	4	110
3	6	180
4	8	260
5	10	350
6	12	450

3 Writing our own function

In this section, we write a simple function to calculate the profit when implementing different advertising strategies. We begin by defining the daily number of customers and daily sales as vectors, given in the code chunk below.

```
daily_customers <- c(33L, 22L, 41L, 39L, 38L, 46L, 47L)
daily_sales <- c(65.05, 80.17, 326.53, 317.88, 374.34, 329.04, 251.29)
```

(a) Using these vectors, calculate the total number of customers during the week and the total weekly sales. These numbers should be identical to those in 2(c).

(b) Your business consultants recommend paying for online advertisement. The advertising company offers a basic package that costs $50 per week. Your consultants expect that the ads will attract two additional customers to your website each day. They also predict that these customers spend on average the same amount as other customers on the same day. Calculate the predicted weekly profit (i.e. weekly sales minus advertising cost) with this strategy.

(c) Write an R function `weekly_profit()` to calculate the weekly profit. This function should accept the following arguments:

- `d_customers`: daily number of customers in the no-advertisement scenario (vector of length 7).
- `d_sales`: daily sales income in the no-advertisement scenario (vector of length 7).
- `w_ad_cost`: advertising cost per week.
- `addl_d_customers`: predicted additional number of daily customers.

(d) Use `weekly_profit()` to calculate the weekly profit if you do not pay for advertisement. Confirm that the result matches the answer to 2(c).

(e) Use `weekly_profit()` to calculate the weekly profit if you purchase the basic package from the advertising company. Confirm that the result matches the answer to 3(b).

4 Working with sequences

It appears as if purchasing the basic advertisement package is profitable. You are now considering upgrading. The advertising company offers several packages. Your business consultants expect that, for every additional customer, you need to pay an increasing amount for advertisement because some ads may be shown to the same user multiple times. The costs for advertisement and the number of additional customers are listed in table 2.

(a) Store the number of additional customers in table 2 as a vector. Use `seq()`.

(b) The costs of the advertising packages in table 2 follow a regular pattern ($50, 50+60, \ldots, 50+60+70+80+90+100$). How can you generate this pattern as a vector with relatively little typing using `cumsum()` and `seq()`? See `?cumsum` or search the World Wide Web for help.

(c) Using the elements in the vectors from 4(a) and 4(b), calculate the weekly profit under each scenario (i.e. each row in table 2) with the function `weekly_profit()` from 3(c).

5 Finding the maximum of a vector

(a) In 4(c), you probably ran `weekly_profit()` six times following the pattern:

```
weekly_profit(
  daily_customers,
  daily_sales,
  weekly_ad_costs[1],
  addl_customers_per_day[1]
)
...
weekly_profit(
  daily_customers,
  daily_sales,
  weekly_ad_costs[6],
  addl_customers_per_day[6]
)
```

R code with many almost identical lines may do the job in a quick and dirty manner, but there are almost always better options. In this case, functional programming (i.e. a coding style in which functions are passed as arguments) can shorten the code. We learn about the functional programming tools provided by the **purrr** package in chapter 22. Base R also has adequate tools that do not require third-party packages, for example the `mapply()` function. The following code chunk uses `mapply()` to store the weekly profit in a vector `wp`. You do not need to memorize this function and its arguments. Here is a brief explanation. The first argument `weekly_profit` is the function that is applied to the remaining arguments. The first two arguments (`weekly_profit` and `weekly_ad_costs`) are inserted into `weekly_profit` in parallel (i.e. `weekly_profit[1]` and `weekly_ad_costs[1]` first, `weekly_profit[2]` and `weekly_ad_costs[2]` second etc.). The final argument `MoreArgs` lists the arguments that are to be held constant in each run of `weekly_profit()`.

```
wp <-
  mapply(
    weekly_profit,
    weekly_ad_costs,
    addl_customers_per_day,
    MoreArgs = list(
      d_customers = daily_customers,
      d_sales = daily_sales
    )
  )
```

(b) Write R code to find the maximum profit from `wp`.

(c) Write R code to find the advertising package (1-6) that maximises profit. You may find `which.max()` useful.