Electricity Consumption: ICA

Overview

It’s coding time!

1. We'll define a problem, design a solution, and code it up in R!
2. We'll solve it using functions, conditionals, and loops.

The Problem

Your electricity bills have been very high lately. You're worried about your environmental impact (not to mention the financial burden!).

To find the root cause, you’ve collected a year of electricity data. A snippet looks like this:

Month	Day	Hour	Kitchen	Laundry	Heating
11	26	18	12	27	1046
11	26	19	534	41	1058
11	26	20	534	0	1060

Each row represents an hour, specified by month, day, hour, and three types of energy usage. Measurements are reported in watt-hours (Wh).

The Problem

Your goal is: **for each month** of your annual electricity data,
find the **day and hour** during which the maximum electricity was used.

Month	Day	Hour	Kitchen	Laundry	Heating
11	26	18	12	27	1046
11	26	19	534	41	1058
11	26	20	534	0	1060

Solution Steps:

1. Algorithm
2. Pseudocode
3. Code

Coming up with an Algorithm

Step 1: Algorithm

Perhaps you've heard the word algorithm before, but what does it mean?

An **algorithm** is defined as a step-by-step procedure for converting inputs to outputs, often to solve a problem.

A canonical example of an algorithm is a recipe. It is a step-by-step process for converting raw ingredients into a food product.

What would be a good sequence of steps to find the solution to our problem?

Hint: Split-Apply-Combine

The **split-apply-combine** paradigm is *very* common in data science.

Indeed, to solve this problem, we must:

1. **Split** the data by month
2. For each month
   a. **Apply** a function to find a day and hour during which time consumption was maximal
3. **Combine** and display the results

Step 2: Pseudocode

A written description of the sequence of steps necessary to solve a problem is **pseudocode**.

Here is a first attempt at pseudocode for our algorithm:

```
for each month
    1. total the consumption (Kitchen + Laundry + Heating)
    2. scan through the total consumption across days and hours
    3. return the day and hour with maximal consumption
```

Step 2: Pseudocode

But you can also write pseudocode that looks more like real code.

For example, here is pseudocode for the function `printSign(x)`:

```
print_sign function(x):
   if x is positive:
      print "x is positive."
   else:
      print "x is negative."
```

Step 2: Pseudocode

Here is pseudocode for our problem that looks (a bit) more like real code:

```
find_max_consumption function(monthly bill):
   for each month in monthly bill:
      total = Kitchen + Laundry + Heating
      index = the index of the entry in the total column with
              the maximum value
      concatenate and print month, day, and hour at this index
```

Coding an Algorithm

Step 3: Coding Time!

Let's try to implement this function!

We'll go through the solution with you step-by-step, so no need to rush!

Download the `.csv` file, and create a new Rmd or R file in the same folder as the download.

At the start of your new file, enter (or copy and paste) this code snippet:

```
raw_data <- read.csv(file = 
'http://cs.brown.edu/courses/cs100/lectures/scripts/section4/electricity_consumption.csv')
monthly_bill <- split(raw_data, factor(raw_data$Month))
```

These two lines mean: import the data into `raw_data`, and then split `raw_data` into a vector of 12 separate data frames for each month. Store the split data in `monthly_bill`.

The `monthly_bill` data type is a **list**.

A refresher: Lists

A sequence can only hold numerics.
A vector can only hold one type of data.

A list is a collection of **components** of any type.

```
> lst <- list("Fred", "Wilma", -1, c(1,3,5,7,9))
> lst
[[1]]
[1] "Fred"
[[2]]
[1] "Wilma"
...
```

A refresher: Lists

As you can see list components are indexed with double brackets:

```
> lst[[1]]
[1] "Fred"

> lst[[4]]
[1] 1 3 5 7 9

> lst[[4]][1]
[1] 1
```

A refresher: Lists

You can loop through a list, just like you might loop through a sequence or a vector.

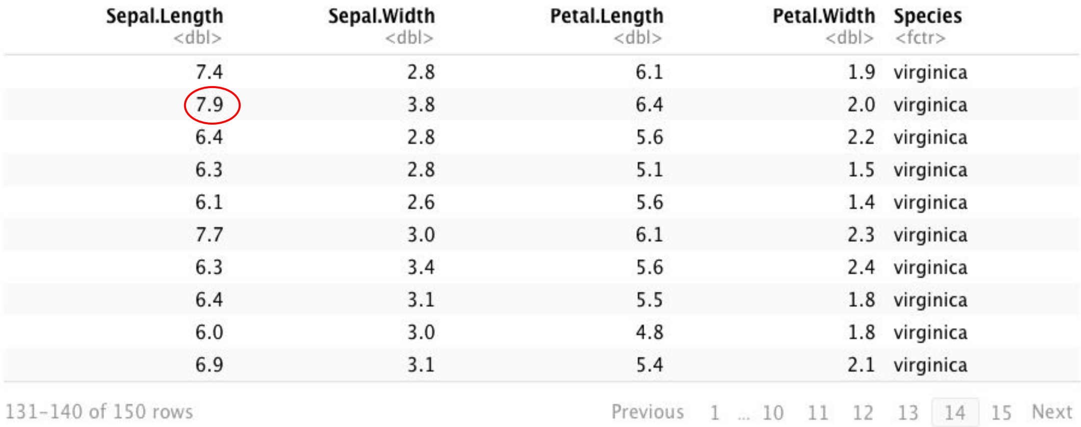
```
lst <- list("Fred", "Wilma", -1, c(1,3,5,7,9))
for (i in lst) {
   print(i)
}
```

```
Fred
Wilma
-1
1 3 5 7 9
```

which.max

Gives the **index** of the entry with the maximum value (not the value itself!)

```
max(iris$Sepal.Length) # 7.9
which.max(iris$Sepal.Length) # 32
```



Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
<dbl>	<dbl>	<dbl>	<dbl>	<fctr>
7.4	2.8	6.1	1.9	virginica
7.9	3.8	6.4	2.0	virginica
6.4	2.8	5.6	2.2	virginica
6.3	2.8	5.1	1.5	virginica
6.1	2.6	5.6	1.4	virginica
7.7	3.0	6.1	2.3	virginica
6.3	3.4	5.6	2.4	virginica
6.4	3.1	5.5	1.8	virginica
6.0	3.0	4.8	1.8	virginica
6.9	3.1	5.4	2.1	virginica

131-140 of 150 rows

Step 3: Coding Time!

Now, enter (or copy and paste) this code snippet:

```
find_max_consumption <- function(monthly_bill) {
  # TODO: insert your code here
}

find_max_consumption(monthly_bill)
```

Your task is to implement the `find_max_consumption` function. So pair with a neighbor, and start coding! Feel free to refer back to the lectures notes on loops.

Solution

Solution

```
find_max_consumption <- function(monthly_bill) {
  # loop over all months
  for (i in 1:length(monthly_bill)) {

  }
}
```

Solution

```
find_max_consumption <- function(monthly_bill) {
  # loop over all months
  for (i in 1:length(monthly_bill)) {

    # save current month
    month <- monthly_bill[[i]]

  }
}
```

Solution

```
find_max_consumption <- function(monthly_bill) {
  # loop over all months
  for (i in 1:length(monthly_bill)) {

    # save current month
    month <- monthly_bill[[i]]

    # total is a new vector that adds up Kitchen, Laundry, and Heating
    total <- month$Kitchen + month$Laundry + month$Heating

  }
}
```

Solution

```
find_max_consumption <- function(monthly_bill) {
  # loop over all months
  for (i in 1:length(monthly_bill)) {

    # save current month
    month <- monthly_bill[[i]]

    # total is a new vector that adds up Kitchen, Laundry, and Heating
    total <- month$Kitchen + month$Laundry + month$Heating

    # for each month, we want to find the index of the maximum consumption
    # instead of the max function, we use which.max
    index <- which.max(total)

  }
}
```

Solution

```
find_max_consumption <- function(monthly_bill) {
  # loop over all months
  for (i in 1:length(monthly_bill)) {

    # save current month
    month <- monthly_bill[[i]]

    # total is a new vector that adds up Kitchen, Laundry, and Heating
    total <- month$Kitchen + month$Laundry + month$Heating

    # for each month, we want to find the index of the maximum consumption
    # instead of the max function, we use which.max
    index <- which.max(total)

    # display the result in month-day-hour format
    cat(i, ”-”, month$Day[index], ”-”, month$Hour[index], "\n", sep = “”)
  }
}
```

More than just the basics

max

First, let’s write a `max` function from scratch.

1. Algorithm
2. Pseudocode
3. Code

Discuss a possible algorithm, and then write pseudocode, with your neighbors.

What's going under the hood?

In the code we just wrote, we used a built-in function to find the index of the maximal value.

```
index <- which.max(total)
```

How would you write the `which.max` function from scratch?

max

```
max function()
    max = 0
    for each item in vector:
        if item > max:
            then max = item
    return max
```

which.max

But we really want the **index** of the maximum element, not the maximum value. So now, let's write the `which.max` function from scratch.

Discuss an algorithm and pseudocode with your neighbors.

Here's the pseudocode for `max` for your reference:

```
max function()
    max = 0
    for each item in vector:
        if item > max:
            then max = item
    return max
```

max and which.max

```
max function()
    max = 0
    for each item in vector:
        if item > max:
            then max = item
    return index
```

```
argmax function()
    max = 0
    index = 0
    for each item in vector:
        if item > max:
            then max = item
            and index = the index of
                        this item
    return index
```

Wrapping up

That's it for programming basics! Good work.

Next week we'll start delving into data cleaning.

Enjoy the long weekend!

Extras

dplyr Solution

```
find_max_consumption2 <- function(data) {
    # Pipeline
    data %>%







}
```

dplyr Solution

```
find_max_consumption2 <- function(data) {
   # Pipeline
   data %>%
      # Split: group_by
      group_by(Month) %>%






}
```

dplyr Solution

```
find_max_consumption2 <- function(data) {
   # Pipeline
   data %>%
      # Split: group_by
      group_by(Month) %>%
      # Apply: new column named Total
      mutate(Total = Kitchen + Laundry + Heating) %>%



}
```

dplyr Solution

```
find_max_consumption2 <- function(data) {
   # Pipeline
   data %>%
      # Split: group_by
      group_by(Month) %>%
      # Apply: new column named Total
      mutate(Total = Kitchen + Laundry + Heating) %>%
      # Combine: top_n displays the rows corresponding
      # to the top n values in a column
      top_n(1, Total)
}
```

dplyr Solution

```
> find_max_consumption2(raw_data)
# A tibble: 12 x 7
# Groups:   Month [12]
     Day Month  Hour Kitchen Laundry Heating Total
   <int> <int> <int>   <int>   <int>   <int> <int>
 1    24     1    11    1161    2251    1042  4454
 2     7     2    14     481    2056    1046  3583
 3     8     3    11    2069      42    1024  3135
 4    12     4    20    2516      10    1112  3638
 5    10     5    16     514    2636      98  3248
 6     7     6    13     857    1088     651  2596
 7    31     7    19     408    1408     977  2793
 8    13     8    20     808    1288     755  2851
 9    12     9    22    2500      56     426  2982
10    17    10    16    1205    1686    1121  4012
11    11    11    17     594    1838    1047  3479
```