Reviewing Python and R basics

Intro to Advanced Data Science - DSC 205 - Lyon College Spring'24

Table of Contents

1. README

The best way to do this is by creating an extended example. We'll continue to develop the code in parallel - first R, then Python.

To study the basics of R, I recommend Norm Matloff's free "fasteR" tutorial (Matloff, 2023). An equivalent for Python is this tutorial by Ian Eyre (Eyre, 2024).

If you're coming back to this session after a break, run all code blocks so far with `org-babel-execute-buffer`.

2. Problem

I have a data set - test grades from two tests. What I want is:

1. to import the data (in a suitable format or formats),
2. to compute statistics (using standard measures),
3. to plot the data (in a suitable format or formats).

3. Classroom setup

1. Open Emacs
2. Open a new file `C-x C-f`
3. Save it as `review.org`

4. Add meta data (header information):

```
#+TITLE: Reviewing Python and R basics
#+AUTHOR: your name (pledged)
#+SUBTITLE: A really short practical introduction to R and Pyhon
#+DATE: Time-stamp: <>
#+PROPERTY: header-args:R :session *R* :results output
#+PROPERTY: header-args:python :session *Python* :results output
```

4. Data

The data are available from Canvas a this (sanitized) CSV file[1]: tinyurl.com/grades-csv.

To import the data:

1. `M-x eww RET tinyurl.com/grades-csv`
2. `C-x C-w RET ~/grades.csv`
3. `C-x k`
4. `C-x C-f RET ~/grades.csv`

What can you tell about the data?

1. It's a CSV file
2. There are three `numeric` columns
3. The first column is an ID column (data type `character` OK)
4. There are missing values (`NA`, `NaN`)
5. There is a `header` row
6. The data look unordered

5. Importing the data

5.1. Importing the data with base R

- Importing data from CSV files is done with the base R function `read.csv`. You should always look at the documentation (`?read.csv`).
- To see, which packages are currently loaded in your session, use `search`:

```
search()
```

- Question: can R packages have duplicate names?

  It's technically possible (they're just names for software) as long as they're distributed through different channels. For packages on CRAN (Comprehensive R Archive Network at cran.r-project.org, where your base R comes from), the rule that each package has a distinct name, is enforced (source). Since there are much fewer packages than functions in packages, one can check the CRAN package list for duplicates.

- There is one positional, mandatory, and a bunch of optional keyword parameter. The positional `file` parameter can be a URL.
- You need to specify if the file has a `header` and if you want strings to automatically be imported as `factor` values. You should also check if `numeric` values have a decimal point or a decimal comma.
- To begin with, just import the data and spit them out again:

```
## save URL as R object
url <- "http://tinyurl.com/grades-csv"
## read CSV data from URL assuming there is a header row
read.csv(file=url,
         header=TRUE)
```

```
     ID Test.1 Test.2
1  1433   4.83  10.00
2  1447  13.00  11.00
3  1421  16.33   8.50
4  1488  19.07  14.50
5  2157  16.83  12.00
6  1380  10.00   9.50
7  1466  18.00  10.33
8  1485  15.50  10.67
9   646  16.83  13.00
10 1136  17.50   9.67
11 1654  11.50  10.67
12 2130  15.83  10.33
13 1916  17.00  10.50
14 1377     NA   3.50
15 1459  16.33  10.17
16 1504  17.50   9.50
17  779  17.50  12.50
18 1329  16.74  12.00
19 1295  17.33   8.17
20  753  16.83  11.33
21 1292     NA   9.50
22 2190     NA     NA
```

5.2. Importing the data with 'Tidyverse'

- At this point, since you've already got one success, you might think about alternatives. There are always alternatives. In R, you could e.g. use `readr::read_csv`. Here is the documentation.

- Just for fun, let's see what we get with this function (you need to install and load `readr`[2]):

```
library(readr)
tb <- read_csv(file = url)
tb
```

```
`curl` package not installed, falling back to using `url()`

indexed 0B in  0s, 0B/s
indexed 1.00TB in  0s, 1.56PB/s

Rows: 22 Columns: 3
── Column specification ──────────────────────────────────────────────────────────
Delimiter: ","
dbl (3): ID, Test 1, Test 2

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
# A tibble: 22 × 3
      ID `Test 1` `Test 2`
   <dbl>    <dbl>    <dbl>
 1  1433     4.83    10
 2  1447    13       11
 3  1421    16.3      8.5
 4  1488    19.1     14.5
 5  2157    16.8     12
 6  1380    10        9.5
 7  1466    18       10.3
 8  1485    15.5     10.7
 9   646    16.8     13
10  1136    17.5      9.67
# ℹ 12 more rows
# ℹ Use `print(n = ...)` to see more rows
```

- Let's check the data structure of a `tibble`:

```
class(tb)
```

- If you're not sure anymore what the value of `url` is or if it is even defined, you can print it:

```
url
```

- The result is a "tibble", a "modern reimagining of the `data.frame`, keeping what time has proven to be effective, and throwing out what is not." (Source). If you're new to tibbles, best forget them again.

- To display the tibble without the control characters, which are generated by the R package, add the following line to your `~/.Rprofile` file:

```
options(crayon.enabled=FALSE)
options(repos = c(CRAN = "https://cloud.r-project.org"))
```

- The second line ensures that there's no pop-up in windows asking you to pick a mirror site. The `~/.Rprofile` file is run whenever you start an R session (to make sure, you can add a message to it, like:

```
message("*** ~/.Rprofile run! ***")
```

- You can source the file (and its environment changes) from within R:

```
source("~/.Rprofile") # source = run the .Rprofile commands
```

5.3. Importing the data as `DataFrame` with Python `pandas`

- To import data from CSV files in Python, you can use the function `pandas.read_csv`. Look at the documentation - it's so vast that you had better looked it up online here. Here is more useful information.
- When you decide to use a package, you must digest all of its documentation. What you skipped or did not understand, will harm you later.
- There is one positional and a bunch of keyword parameters. The positional file parameter can be a URL. One difference to R is that the positional argument cannot be named.
- Python 'infers' if there's a `header` or not but (unlike R) it assumes that there is one in the first record (line).

- Let's try it. You may have to run this code block twice.

```
from pandas import read_csv
url = "http://tinyurl.com/grades-csv"
print(read_csv(url))
```

- How can you see which packages are loaded in your Python session?

```
import sys
loaded_packages = list(sys.modules.keys())
print(loaded_packages)
```

- This is not easy to read. Instead, print the `list` as a `comprehension`, with a `for` loop integrated:

```
[print(_) for _ in loaded_packages] # as list comprehension
```

5.4. Emacs interlude

- I've just set myself up with Linux at home - finally fed up with Windows (again). So I'm repopulating Emacs with some packages that I like. You should learn how to do that, too.
- Let's start with the `org-bullets` package, which turns the `*` characters used for headlines in Org-mode into nice bullets.
- To load it, you have to enter
  1. `M-x list-packages` [this lists all available packages]
  2. `U` [this checks for updates]
  3. `C-s org-bullets` [to find the package]
  4. `i` [to mark it for install]
  5. `x` [to install it]
- Now run `M-x org-bullets-mode` in any Org-mode file with headlines. This mode toggles - that is you can switch bullets on/off.

- To have functions like these enabled at startup, you need to add a line of Lisp to your `.emacs` file: `(require 'org-bullets)`. Then it will always be 'on' unless you switch it off.

5.5. Importing the data with the Python Standard library (`urllib`)

If we want to only use the Standard Library, things get more complicated: we fetch data from the web, write them to memory, and then write the file into a dictionary, which we can convert to a data frame.

5.5.1. Fetching CSV data from the web and write them to file

1. Approach
   1. Use the `urllib.request` module to open the URL and read the data.
   2. Use the `csv` module to parse the CSV data read from the URL.
   3. Write the parsed CSV data to a file.
2. Code Example

```
import csv
import urllib.request

# URL containing the CSV data
url = "http://tinyurl.com/grades-csv"

# File path to write the CSV data
output_file_path = "grades.csv"

# Open the URL and fetch the CSV data
with urllib.request.urlopen(url) as response:
    # Assume the response is text (CSV data), read it as such
    lines = [l.decode('utf-8') for l in response.readlines()]

    # Now, write these lines to a CSV file
    with open(output_file_path, 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)
        for line in lines:
            # Parse each line as CSV
            reader = csv.reader([line])
            for row in reader:
                # Write the parsed row to the file
                writer.writerow(row)

print("CSV data has been read from the URL and written to", output_file_path)
```

3. Notes
   - This code snippet assumes the CSV data is encoded in UTF-8.
   - The `newline`"= parameter in the `open` function call ensures that the newline characters in the input are handled according to the Python CSV module's requirements, which might vary across different platforms.
   - This example reads all lines from the URL response into memory before writing them to a file. For very large CSV files, you might consider a more memory-efficient approach that processes lines one at a time.

5.5.2. Read the CSV file into a Python `dictionary`

1. Approach
   - Use the `csv.DictReader` class to read the CSV file. This class automatically reads the first row of the CSV file as fieldnames (keys of the dictionary).

   - Iterate over the rows in the `DictReader` object to access each row as a dictionary.
2. Code Example

```
import csv

# File path of the CSV file
input_file_path = "grades.csv"

# List to hold dictionaries (each row as a dictionary)
data = []

# Read the CSV file as a dictionary
with open(input_file_path, mode='r', newline='') as csvfile:
    reader = csv.DictReader(csvfile)

    # Iterate over rows in the CSV file
    for row in reader:
        # Each row is a dictionary
        data.append(row)

# print result if file exists
if data:
    [print(i) for i in data]
```

3. Notes
   - The `csv.DictReader` does not require specifying column names upfront; it uses the first row of the CSV file for that.
   - Each row accessed in the loop is a dictionary, where keys are column names from the first row of the CSV file, and values are the corresponding entries for each row.
   - This method is handy for CSV files with a header row. If your CSV file does not have a header row, you need to manually specify the fieldnames parameter when creating the `DictReader` object.

5.5.3. Convert `dictionary` data to `DataFrame` with `pandas`

- Convert the `dictionary` to a data frame:

```
import pandas as pd

df = pd.DataFrame(data)
print(df)
```

- Note that the missing 'NaN' values are not displayed in this result. However, if you check the Python console (`*Python*`), you will see them.

6. TODO Summary and glossary

Footnotes:

[1] The original file is a lot messier. We'll learn later how to handle messy CSV files, i.e. how to include or exclude columns, or how to dump the lot into an SQLite database and `SELECT` them from there.

[2] To install use `install_packages("readr")` in the R console.

Date: Time-stamp: <2024-02-09 Fri 07:23>

Author: Marcus Birkenkrahe (pledged)

Created: 2024-03-04 Mon 12:49

Validate