Reviewing Python and R basics

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

Table of Contents

1. README

- We're looking at data transformation and modeling in R and Python.
- Code along.

2. Decisions

- We can only transform the date with regard to what the data actually represent. A lot of misinterpretation is based on lack of information.
- In this case, the values are point values. Test 1 had a maximum of 20, test 2 had a maximum of 15 points. To compare results across these two tests, we need to transform the date to the same scale.
- We can also perform this last transformation when plotting the data to compare performance in test 1 vs. test 2.
- The last test subject with the ID = 2190 whose test values are missing, is not a student at all but a test user. We need to exclude him from the analysis altogether (this is a decision!).
- We also should exclude other missing values from the data analysis, because a student who got 0 points because he did not participate, should not alter the statistical averages (this is a decision!).
- In both R and Python, you can easily check the number of missing values and where they are in the data, and you can exclude them from any computation.
- We also need to decide the order in which to transform the data:
  1. remove test user data
  2. remove missing values

3. Mutability in Python ("Mutabor!")[1]

- Python distinguishes between mutable and immutable data structures. Mutable means that you can add or drop values, modify values in cells, add or remove rows, and change the index.

- Pandas are mutable, strings and tuples are immutable.
- A string example:

```
s = "hello"  # defines a string
print(s)
s[1] = 'a'  # TypeError because strings are immutable
print("Error: Strings are immutable. Aborting.")
```

- We can use `try...except` to define an exception:

```
s = "hello"
try:
    s[1] = 'a'  # TypeError because strings are immutable
except TypeError:
    print("Error: Strings are immutable. Aborting.")
```

- To change the letter, we must create a new string

```
s_new = s.replace('e','a')
# print the result as an f-string
print(f"Old: '{s}', new: '{s_new}'")
```

- How did we print the strings?
  1. with an f-string ('formatted')
  2. it has the form `print(f"...")` or `print(f'...')`
  3. inside the string, variables are added in `{ }`
  4. formatting is taken care of automatically depending on data type
- A tuple is an immutable collection of ordered elements. Once it's created, you cannot add, remove or change its elements.

```
t = (1,2,3)
try:
    t[0] = '10'  # TypeError because strings are immutable
except TypeError:
    print("Error: Tuples are immutable. Aborting.")
```

- To change a tuple, you need to create a new one, too:

```
t_new = (10,) + t[1:]
print(f"Old: '{t}', new: '{t_new}'")
```

- How did we create `t_new`?
  1. concatenate a single-item tuple `(10,)`
  2. to the two-item sub-tuple `(2,3)`

4. Remove unwanted data in Python

4.1. Listing user-defined variables (equivalent of `ls()`)

- Let's find out how to list the user-defined variables, like our data frame. In IPython, there is the `%who` "magic" command for that.
- In Python, the built-in function `globals` returns the current session's global variables. Check out the `help` for `globals`.

- `globals` is a dictionary, so it has keys and values:

```
print(type(globals().items()))
```

- We can print all `items` as a comprehension (an implicit loop):

```
[print(_) for _ in globals().items()]
```

- Without the list comprehension, this looks like:

```
for name, value in globals().items():
    print(name,value)
```

- We're only interested in user-defined variables though. All system-defined objects either start with an underscore _, or they are `callable` (if they're functions), or they are built-in.

- To only see the user-defined variables but not functions or built-in objects, run the following code block:

```
for variable_name, value in globals().items():
    if not variable_name.startswith('_') and not callable(value) and\
       not type(value).__module__ == 'builtins':
        print(f"{variable_name}: {type(value)}")
```

- As you can see, Python lists the libraries and modules that we loaded as well. If we're only interested in data frames, we can write:

```
for variable_name, value in globals().items():
    if type(value).__name__ == 'DataFrame':
        print(f"DataFrame found: {variable_name}")
```

- This checks if any of the dictionary values have the `__name__` attribute 'DataFrame'. If you enter `help(__name__)` you get all the registry information about your current session:

```
print(help(__name__))
```

4.2. Converting missing values to NaN

- R will always show the `NA` values, Python does not always - the `pandas` function `read_csv` however does convert missing values to `NaN`.

- Checking if `url` still stores the link to the CSV file:

```
print(url)
```

```
http://tinyurl.com/grades-csv
```

- The import with `read_csv` will not work without loading `pandas`: there are two ways to do this - import all of it or only the function

```
import pandas as pd  # now you can use all pandas functions
# you have to prefix them with pd.
from pandas import read_csv # now now prefix is needed
```

- Use `read_csv(url)` for CSV data stored online at `url`, or with a file name as argument:

```
df = read_csv(url)
print(df)
```

- With file name:

```
df1 = read_csv('../data/grades.csv')
print(df==df1)
```

- When comparing two DataFrames with missing values, the comparison will return `False` for those values, because `NaN` is a floating-point representation of "Not a Number" and is inherently unequal to itself:

```
import numpy as np
print(np.nan == np.nan)
print(pd.NA == pd.NA)

print(pd.isna(pd.NA)) # check for missing values with pandas
print(np.isnan(np.nan)) # check for missing values with numpy
```

```
False
<NA>
True
True
```

4.3. Removing unwanted values

- To remove the `2190` record for the test user, you can use a pandas function, or you can use the index method with a Boolean comparison.
- The index method has two parts:
    1. create a logical flag vector
    2. use the flag vector as an index vector

- Create the flag vector:

```
## compare all ID values of the DataFrame df with '2190'
print(df.ID=='2190')
```

- Unfortunately, there is no `True` value - the ID value `'2190'` is not found. What to do? The following code says why (similar to `str`):

```
print(df.info())
```

- Or if you only want to see your DataFrame data types:

```
print(df.dtypes)
```

- Notice the difference: `df.info()` is a callable function, while `df.dtypes` is an attribute:

```
print(callable(df.info))  # is df.info callable?
print(callable(df.dtypes))   # is df.dtypes callable
```

- Now, create the flag vector for real:

```
print(df.ID==2190)
```

- Use the flag vector as an index vector to get the record:

```
idx = df.ID==2190
print(df[idx])
```

- What's the equivalent of the `which` function in R (which returns the index for the `True` elements)? The solution reveals the close connection between `pandas` for data frames, and `numpy` for arrays:

```
print(np.where(df['ID']==2190)[0])
```

- Let's analyze this expression:

  1. `df['ID']` extracts the 'ID' column
  2. `df['ID'] == 2190` looks for the (numeric) value `2190` in the column
  3. `np.where` is an array of the indices for which the condition is `True` - the first element is indexed '0':

     ```
     print(np.where(df['ID']==2190))
     ```

```
print(df.dtypes)
```

- Finally! Remove the record, overwrite `df`, then check explicitly if the record is still there or not:

```
# remove record - this prints a transient copy of df
print(df[df['ID'] != 2190])

# overwrite data frame
df = df[df['ID'] != 2190] # copy non-targeted records to new df

# check if any records contain the test user ID
print(any(df['ID'] == 2190))  # output should be False
```

- Alternatively, you can use the `df.drop` function and our earlier method of getting the index value corresponding to the last row:

```
print(df.drop(df.index[np.where(idx)[0]]))
```

- Notice that this last expression does not actually change the data frame because by default, these functions create copies. To modify the original DataFrame, you need to set `inplace=True`:

```
df.drop(df.index[np.where(idx)[0]], inplace=True
```

- If you know that you want to remove the last record, the simplest way is to use the known index of the last row and check with `tail`:

```
df.drop(df.index[-1], inplace=True)
df.tail()
```

5. Remove unwanted data in R

- Begin by importing the data in a `data.frame` named `df` from the URL http://tinyurl.com/grades-csv and print it:

```
url <- 'http://tinyurl.com/grades-csv'
df <- read.csv(url, header=TRUE)
df
```

- In R, the `ls()` command lists all user-defined variables, not distinguishing between different data structures - everything in R is an object:

```
ls()
```

- Data frames are built-in, and so is the display of missing values (`NA`):

```
df
```

- Data frames are built-in, and so is the display of missing values (`NA`):

```
summary(df)
```

  - To find the flag vector make sure of data types:

```
str(df)
```

- Then use the index method to define the flag vector:

```
idx <- df$ID == 2190
idx
```

- Use the flag vector to index the dataframe, remove the record and overwrite the dataframe:

```
df$ID[idx]  # the ID value of the record
which(idx)  # the index (row number) of the record
df <- df[-which(idx),] # overwriting the data frame without the row
df # the corrected data frame
```

6. Statistical properties in Python with `describe` and in R with `summary`

- Our next project is to compute some statistical properties without being affected by missing values.
- As indicated, this is done more easily directly when plotting but we shall first do it directly, algebraically, to know what we're doing.

6.1. Python

- For Python, let's get the proper DataFrame first: we don't need the last row and we don't need the first column for statistics:

  1. import `pandas`
  2. import data from URL (http://tinyurl.com/grades-csv)
  3. remove last row and overwrite `DataFrame`
  4. remove first column of `DataFrame`
  5. print the result

```
# import pandas for data frames
import pandas as pd

# save URL in variable
url = 'http://tinyurl.com/grades-csv'

# read data from URL and store in dataframe
df = pd.read_csv(url)

# remove last row and overwrite dataframe
df = df[:-1]

# print dataframe
print(df)
```

```
      ID  Test 1  Test 2
0   1433    4.83   10.00
1   1447   13.00   11.00
2   1421   16.33    8.50
3   1488   19.07   14.50
4   2157   16.83   12.00
5   1380   10.00    9.50
6   1466   18.00   10.33
7   1485   15.50   10.67
8    646   16.83   13.00
9   1136   17.50    9.67
10  1654   11.50   10.67
11  2130   15.83   10.33
12  1916   17.00   10.50
13  1377     NaN    3.50
```

```
14  1459   16.33   10.17
15  1504   17.50    9.50
16   779   17.50   12.50
17  1329   16.74   12.00
18  1295   17.33    8.17
19   753   16.83   11.33
20  1292     NaN    9.50
```

- The 'ID' column is not statistically relevant so let's remove it by selecting only the second and third column and overwriting `df`:

```
df = df.iloc[:,[1,2]]
print(df)
```

- The `pandas` function `describe` returns some of the information of R's `summary`. Note that it is not generic (it only works for `pandas` data frames):

```
print(df.describe())
```

- Find out what happened to the `NaN` values by looking at the documentation:

> "Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values."

6.2. R

- Compare with R's `summary`:

```
## read data into dataframe
df <- read.csv(file="http://tinyurl.com/grades-csv",
               header=TRUE)
# remove last row
df <- df[-which(df$ID==2190),]
# for the summary, only use second and third column
summary(df[-1])
```

- When reading the documentation of `summary`, it is not immediately clear if `NA` are removed or not. Using the function `na.omit`, we can check if the summary is changed or not. `na.omit` removes all rows with `NA` values in them:

```
summary(df$Test.1)
summary(na.omit(df$Test.1))
```

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
4.83   15.66   16.83   15.50   17.41   19.07       2
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
4.83   15.66   16.83   15.50   17.41   19.07
```

- This settles it: `summary` in R, like `describe` in Python's `pandas`, ignores `NA` for the summary statistics computation.

7. More modeling (later)

There is more modeling that we can do here, as part of our EDA with the data. Vector data can be checked for trends, and we could use the results for the first two tests to try and predict the next test (e.g. with linear regression). We'll do that in a later session.

8. TODO Statistical properties from scratch in R

- The functions we need to rebuild the results of `summary` are `quantile`, `median`, and `mean` (since the quantiles include min and max)
- To find out how these functions treate `NA`, check their documentation. You can do this most easily by opening an R console and calling `help`.

    1. `quantile`: default is `na.rm=FALSE`
    2. `median`: default is `na.rm=FALSE`
    3. `mean`: default is `na.rm=FALSE`

- Let's run these:

```
## test 1
quantile(df$Test.1,na.rm=TRUE)
median(df$Test.1,na.rm=TRUE)
mean(df$Test.1,na.rm=TRUE)
## test 2
quantile(df$Test.2,na.rm=TRUE)
median(df$Test.2,na.rm=TRUE)
mean(df$Test.2,na.rm=TRUE)
```

```
    0%    25%    50%    75%   100%
 4.830 15.665 16.830 17.415 19.070
[1] 16.83
[1] 15.49737
   0%   25%   50%   75%  100%
 3.50  9.50 10.33 11.33 14.50
[1] 10.33
[1] 10.34952
```

- To compare the results of `summary` and the individual functions automatically, we can use `identical`, but we need to know what structure `summary` has:

```
str(summary(df))
```

```
'table' chr [1:7, 1:3] "Min.   : 646  " "1st Qu.:1295  " "Median :1433  " "Mean
- attr(*, "dimnames")=List of 2
 ..$ : chr [1:7] "" "" "" "" ...
 ..$ : chr [1:3] "      ID" "    Test.1" "    Test.2"
```

- The statistical properties are stored in a table of character values:

```
summary(df)[1,] # first row across  all columns
summary(df)[,2] # second column across all rows
summary(df)[dim(summary(df))[1],c(2,3)] # NA information in the last row
```

```
               ID            Test.1            Test.2
 "Min.   : 646  " "Min.   : 4.83  " "Min.   : 3.50  "

"Min.   : 4.83  " "1st Qu.:15.66  " "Median :16.83  " "Mean   :15.50  " "3rd Qu.:1

"Max.   :19.07  "     "NA's   :2  "
       Test.1        Test.2
"NA's   :2  "            NA
```

- The last command is worth analyzing:
  1. `dim` returns the dimensions of the `summary` object: 7,3
  2. The first column gives the number of rows for any object: `dim(summary(df))[1]` returns the number 7 (last row)
  3. The full command extracts the 7th row and the columns 2-3.
- To compare the individual function values and the `summary` table results automatically, we should write a function! We do that in a later session!

9. TODO Statistical properties from scratch in Python

To be completed!

10. TODO Summary and glossary

Footnotes:

[1] Of all Hauff's tales the most popular in English was 'Caliph Stork', which was in fact the first story in Die Karawane. Its first appearance in English was in Burns's Select Popular Tales, after which it was printed in all subsequent major selections or complete editions. It was also included in Grimms' Goblins, Andrew Lang's Green Fairy Book (1892) and no. 57 of 'Books for the Bairns'. It recounts how the Caliph of Bagdad and his Vizier acquire the means of transforming themselves into storks, but because they laugh while thus transformed they forget the magic word that will turn them back into human beings. This word is Mutabor, the Latin for 'I shall be changed'. An owl that is similarly metamorphosed advises them how to rediscover the word, but only on condition that one of them offers her his hand in marriage and so disenchants her. In this way the Caliph acquires a wife, not through any romantic attachment, but as an exchange for services rendered. Hauff's source was the story of 'König Papagei' (King Parrot) from the German translation of the Arabian Nights by Habicht, von der Hagen and Schall (Breslau, 1824). (Source)

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

Author: Marcus Birkenkrahe (pledged)

Created: 2024-03-04 Mon 12:51

Validate