Python String Array Unique Frequency, Search, Replace, and Wrap Strings

Fan Wang

2020-06-17

Contents

1 Strings

Go to the **RMD**, **PDF**, or **HTML** version of this file. Go back to Python Code Examples Repository (bookdown site) or the pyfan Package (API).

```
import numpy as np
import string as string
import random as random
import pprint
```

1.1 List of Array Count Frequency

There is a list of strings, with repeating values, count the frequence of the each unique string.

```
ls_st_status = ["success", "running", "running", "running", "finished", "pending", "pending"]
ls_freq = [ [f'{ls_st_status.count(st_status)} of {len(ls_st_status)} {st_status}']  for st_status in s
pprint.pprint(ls_freq)
```

```
## [['3 of 7 running'],
##  ['1 of 7 finished'],
##  ['1 of 7 success'],
##  ['2 of 7 pending']]
```

1.2 Get Substring

Given string, get substring after a word.

```
st_func_stack_code = 'dc_ls_combo_type = pyfan_amto_lsdcconvert.ff_ls2dc(ls_combo_type,'
st_search_break = 'ff_ls2dc('
st_string_after = st_func_stack_code.split(st_search_break)[1]
st_search_break = ','
st_string_after = st_func_stack_code.split(st_search_break)[1]
print(f'{st_string_after=}')
```

```
## st_string_after=''
```

1.3 Generate Random Strings

Generate some random strings:

```
random.seed(123)
it_word_length = 5
st_rand_word = ''.join(random.choice(string.ascii_lowercase) for i in range(it_word_length))
st_rand_word = st_rand_word.capitalize()
print(f'{st_rand_word=}')
```

```
## st_rand_word='Bicyn'
```

Generate a block or random text and then convert it to a one list of strings:

```
random.seed(123)
it_words_count = 15
it_word_length = 5
st_rand_word_block = ''.join(random.choice(string.ascii_lowercase) for ctr in range(it_word_length*it_w
ls_st_rand_word = [st_rand_word_block[ctr: ctr + it_word_length].capitalize()
                   for ctr in range(0, len(st_rand_word_block), it_word_length)]
print(f'{ls_st_rand_word=}')
```

```
## ls_st_rand_word=['Bicyn', 'Idbmr', 'Rkkbf', 'Ekrkw', 'Hfany', 'Ctmca', 'Kxodb', 'Cveez', 'Ajnsp', 'Ip
```

Reshape the array of words to a matrix:

```
mt_st_rand_word = np.reshape(ls_st_rand_word, [3,5])
print(f'{mt_st_rand_word=}')
```

```
## mt_st_rand_word=array([['Bicyn', 'Idbmr', 'Rkkbf', 'Ekrkw', 'Hfany'],
##        ['Ctmca', 'Kxodb', 'Cveez', 'Ajnsp', 'Ipbyj'],
##        ['Kqzpg', 'Tuqsz', 'Kamyu', 'Qnvru', 'Zvtpq']], dtype='<U5')
```

```
print(f'{mt_st_rand_word.shape=}')
```

```
## mt_st_rand_word.shape=(3, 5)
```

```
print(f'{type(mt_st_rand_word)=}')
```

```
## type(mt_st_rand_word)=<class 'numpy.ndarray'>
```

1.4 Paste String with Connector

Similar to the paste function in R, given a list of strings, paste them together with a connector.

```
st_separator = "_"
st_pasted = st_separator.join(filter(None, ['abc', 'efg']))
print(f'{st_pasted=}')
# empty if the strings are empty
```

```
## st_pasted='abc_efg'
```

```
st_pasted = st_separator.join(filter(None, ['', '', '']))
print(f'{st_pasted=}')
# If only one not empy output the same
```

```
## st_pasted=''
```

```
st_pasted = st_separator.join(filter(None, ['abc', '']))
print(f'{st_pasted=}')
```

```
## st_pasted='abc'
```

1.5 Add String Suffix to Numeric Array

Given an numeric array, add string, for example to generate sequencial column names with suffix c:

```
ar_st_colnames = [ 's' + str(it_col) for it_col in np.array(range(1, 3))]
print(ar_st_colnames)
```

```
## ['s1', 's2']
```

1.6 Search if Names Include Strings

Given a list of strings, loop but skip if string contains elements string list.

```
# define string
ls_st_ignore = ['abc', 'efg', 'xyz']
ls_st_loop = ['ab cefg sdf', '12345', 'xyz', 'abc xyz', 'good morning']

# zip and loop and replace
for st_loop in ls_st_loop:
  if sum([st_ignore in st_loop for st_ignore in ls_st_ignore]):
    print('skip:', st_loop)
  else:
    print('not skip:', st_loop)
```

```
## skip: ab cefg sdf
## not skip: 12345
## skip: xyz
## skip: abc xyz
## not skip: good morning
```

1.7 Replace a Set of Strings in String

Replace terms in string

```
# define string
st_full = """
abc is a great efg, probably xyz. Yes, xyz is great, like efg.
eft good, EFG capitalized, efg good again.
A B C or abc or ABC. Interesting xyz.
"""

# define new and old
ls_st_old = ['abc', 'efg', 'xyz']
ls_st_new = ['123', '456', '789']
```

```python
# zip and loop and replace
for old, new in zip(ls_st_old, ls_st_new):
  st_full = st_full.replace(old, new)

# print
print(st_full)
```

```
##
## 123 is a great 456, probably 789. Yes, 789 is great, like 456.
## eft good, EFG capitalized, 456 good again.
## A B C or 123 or ABC. Interesting 789.
```

1.8 Wrap String with Fixed Width

Given a long string, wrap it into multiple lines with fixed width.

```python
import textwrap

# A long Path
st_path = """
C:/Users/fan/Documents/Dropbox (UH-ECON)/Project Emily Minority Survey/EthLang/reg_lang_abi_cls_mino/ta
"""

# Wrap text with tight width
st_wrapped = textwrap.fill(st_path, width = 20)
print(st_wrapped)
```

```
##  C:/Users/fan/Docume
## nts/Dropbox (UH-
## ECON)/Project Emily
## Minority Survey/EthL
## ang/reg_lang_abi_cls
## _mino/tab3_fm/attain
## _m_vs_f/tab3_mand_ta
## lk_m2c_hfracle02.tex
```

Combine Strings that are wrapped and not Wrapped

```python
# Paths
st_path_a = "C:/Users/fan/Documents/Dropbox (UH-ECON)/Project Emily Minority Survey/EthLang/reg_lang_ab
st_path_b = 'C:/Users/fan/R4Econ/support/development/fs_packaging.html'

# Combine Strings and Wrap
str_dc_records = 'First Path:'.upper() + '\n' + \
                 textwrap.fill(st_path_a, width=25) + '\n\n' + \
                 'Second Path:'.upper() + '\n' + \
                 textwrap.fill(st_path_b, width=25)

# Print
print(str_dc_records)
```

```
## FIRST PATH:
## C:/Users/fan/Documents/Dr
## opbox (UH-ECON)/Project
## Emily Minority Survey/Eth
## Lang/reg_lang_abi_cls_min
```

```
## o/tab3_fm/attain_m_vs_f/t
## ab3_mand_talk_m2c_hfracle
## 02.tex
##
## SECOND PATH:
## C:/Users/fan/R4Econ/suppo
## rt/development/fs_packagi
## ng.html
```

1.9 Change Round for Lists of String Estimates

Here we have two strings in a list, with point estimates and corresponding standard errors. Estimates are separated by commas. We want to change the number of decimal points shown and set appropriate roundings. Several steps: (1) split string by comma (2) Loop over (3) extract numerical elements (4) recover

```
it_round_decimal = 1
ls_st_all_estimates = ["84.506***, 91.758***, 107.950***, 115.879***, 133.560***\n",
                       "(7.796), (4.848), (4.111), (5.044), (6.961)\n",
                       "68.180***, 47.921***, 47.127***, 51.366***, 41.764***\n",
                       "(8.986), (5.368), (4.995), (5.099), (8.637)\n"]

for st_all_estimates in ls_st_all_estimates:

    # delete linebreak at end of line
    st_all_estimates = st_all_estimates.replace("\n", "")

    # split
    ls_st_estimates = st_all_estimates.split(",")

    # Loop over each value separated by commas
    for it_esti_ctr, st_esti in enumerate(ls_st_estimates):

        # Default update is to keep current
        st_esti_update = st_esti

        # If estimates, might have stars
        st_esti_numeric = st_esti.strip()
        st_esti_numeric = st_esti_numeric.replace("*", "")
        st_esti_numeric = st_esti_numeric.replace("(", "")
        st_esti_numeric = st_esti_numeric.replace(")", "")

        # Decimal Rounding
        fl_esti_rounded = round(float(st_esti_numeric), it_round_decimal)
        st_esti_rounded = f'{fl_esti_rounded:.{it_round_decimal}f}'

        # Replace
        print(f'{st_esti=} + {st_esti_numeric=} + {st_esti_rounded=}')
        st_esti_rounded = st_esti.replace(st_esti_numeric, st_esti_rounded)

        # Update List
        ls_st_estimates[it_esti_ctr] = st_esti_rounded

    # Flatten comman
    st_text_out = ','.join(ls_st_estimates)
    print(f'\n{st_text_out=}\n')
```

```
    print()
```

```
## st_esti='84.506***' + st_esti_numeric='84.506' + st_esti_rounded='84.5'
## st_esti=' 91.758***' + st_esti_numeric='91.758' + st_esti_rounded='91.8'
## st_esti=' 107.950***' + st_esti_numeric='107.950' + st_esti_rounded='108.0'
## st_esti=' 115.879***' + st_esti_numeric='115.879' + st_esti_rounded='115.9'
## st_esti=' 133.560***' + st_esti_numeric='133.560' + st_esti_rounded='133.6'
##
## st_text_out='84.5***, 91.8***, 108.0***, 115.9***, 133.6***'
##
##
## st_esti='(7.796)' + st_esti_numeric='7.796' + st_esti_rounded='7.8'
## st_esti=' (4.848)' + st_esti_numeric='4.848' + st_esti_rounded='4.8'
## st_esti=' (4.111)' + st_esti_numeric='4.111' + st_esti_rounded='4.1'
## st_esti=' (5.044)' + st_esti_numeric='5.044' + st_esti_rounded='5.0'
## st_esti=' (6.961)' + st_esti_numeric='6.961' + st_esti_rounded='7.0'
##
## st_text_out='(7.8), (4.8), (4.1), (5.0), (7.0)'
##
##
## st_esti='68.180***' + st_esti_numeric='68.180' + st_esti_rounded='68.2'
## st_esti=' 47.921***' + st_esti_numeric='47.921' + st_esti_rounded='47.9'
## st_esti=' 47.127***' + st_esti_numeric='47.127' + st_esti_rounded='47.1'
## st_esti=' 51.366***' + st_esti_numeric='51.366' + st_esti_rounded='51.4'
## st_esti=' 41.764***' + st_esti_numeric='41.764' + st_esti_rounded='41.8'
##
## st_text_out='68.2***, 47.9***, 47.1***, 51.4***, 41.8***'
##
##
## st_esti='(8.986)' + st_esti_numeric='8.986' + st_esti_rounded='9.0'
## st_esti=' (5.368)' + st_esti_numeric='5.368' + st_esti_rounded='5.4'
## st_esti=' (4.995)' + st_esti_numeric='4.995' + st_esti_rounded='5.0'
## st_esti=' (5.099)' + st_esti_numeric='5.099' + st_esti_rounded='5.1'
## st_esti=' (8.637)' + st_esti_numeric='8.637' + st_esti_rounded='8.6'
##
## st_text_out='(9.0), (5.4), (5.0), (5.1), (8.6)'
```

1.10 Check String Composition

There are some special string structures, if a string is of this special structure, do something, if it is not, do something else. In the following example, one string structure is a string with a equality sign and than an integer after. Is a string of this nature?

The integer check uses this:

> all([st_ele in '1234567890' for st_ele in esti_top_which])

In the example below, the first and last elements are valid.

```
# examples strings to check
ls_st_exas = ["C1E126M4S3=2",
              "simu_tst/M4S3_top_json.json",
              "M4S3_top_json.json",
              "simu_tst/M4S3=_top_json.json",
              "=====",
              "==$$%%==123123",
```

```
            "$$%%=123123"]
# check
for combo_type_e in ls_st_exas:
    # split
    st_connector = "="
    ls_combo_type_e_split = combo_type_e.split(st_connector)
    # first check length
    bl_esr_json = True
    if len(ls_combo_type_e_split) == 2:
        [compesti_short_name, esti_top_which] = ls_combo_type_e_split
        # check type
        bl_first_is_str = isinstance(compesti_short_name, str)
        bl_second_is_int = all([st_ele in '1234567890' for st_ele in esti_top_which])
        if bl_first_is_str + bl_second_is_int < 2:
            bl_esr_json = False
            print(f'{bl_esr_json=}, {bl_first_is_str=}, {bl_second_is_int=}, {combo_type_e=}')
        else:
            # Print
            print(f'{bl_esr_json=}, {bl_first_is_str=}, {bl_second_is_int=}')
    else:
        bl_esr_json = False
        print(f'{bl_esr_json=}, {combo_type_e=}')
```

```
## bl_esr_json=True, bl_first_is_str=True, bl_second_is_int=True
## bl_esr_json=False, combo_type_e='simu_tst/M4S3_top_json.json'
## bl_esr_json=False, combo_type_e='M4S3_top_json.json'
## bl_esr_json=False, bl_first_is_str=True, bl_second_is_int=False, combo_type_e='simu_tst/M4S3=_top_js
## bl_esr_json=False, combo_type_e='====='
## bl_esr_json=False, combo_type_e='==$$%%==123123'
## bl_esr_json=True, bl_first_is_str=True, bl_second_is_int=True
```