PharmaSUG 2022 - Paper EP-07

Clean Messy Clinical Data Using Python

Abhinav Srivastva, Exelixis Inc

ABSTRACT

Data in clinical trials can be transmitted in various formats such as Excel, CSV, tab-delimited, ASCII files or via any Electronic Data Capture (EDC) tool. A potential problem arises when data has embedded special characters or even non-printable (control) characters which affects all downstream analysis and reporting; in addition to being non-compliant with CDISC standards. The paper explores various ways to handle these unwanted characters by leveraging the capabilities of Python language. The discussion begins with correcting these characters using their ASCII code, followed by Unicode, and later exploring normalization techniques such as NFC, NFD, NFKC, NFKD to normalize unicode characters. Often times, data cleaning will require a combination of these techniques to get the desired result.

INTRODUCTION

Python provides some useful libraries that can aid in identifying and treating unwanted characters in the text. Some of these resemble closely with SAS programming language. The paper uses regular expression and character translation available in Python for most of these tasks. Below is a summary of different ways of data cleaning techniques covered in the paper:

Create a Test String

Before looking at the techniques, let's use a clean test string and add special and control characters to it.

```
clean = 'The quick brown fox jumps over the lazy dog!'
```

Next, let's add special characters (ü and ó) and a carriage return (\r) to the test string and store in a variable called '**messy**'.

```
# add special characters (ü and ó)
sp = 'The qüick brown fóx jumps over the lazy dóg!'

# add control words (carriage return character)
messy = sp + chr(13)
messy

>>[Out] 'The qüick brown fóx jumps over the lazy dóg!\r'
```

```
messy = 'The qüick brown fóx jumps over the lazy dóg!\r'
```

Example 1: Using REPLACE and TRANSLATE string methods

We can use Python's replace and translate method to selectively exclude special and control characters that we added above as demonstrated below:

```
# Using replace()

clean_replace = messy.replace('ü', 'u').replace('ó', 'o').replace('\r', '')
clean_replace


>>[Out] 'The quick brown fox jumps over the lazy dog!'

# Using translate()

charmap = {'ü': 'u', 'ó' : 'o', '\r': None}

trans_table = messy.maketrans(charmap)
clean_trans = messy.translate(trans_table)
clean_trans

>>[Out] 'The quick brown fox jumps over the lazy dog!'
```

Both of these methods got rid of unwanted characters and matches up exactly with the original 'clean' string.

```
# Check equality, otherwise print a custom error message

assert clean_trans == clean, "Strings dont match"
```

Example 2: Using Regular Expressions

To use regular expression and perform string manipulation, 're' and 'string' are two libraries that needs to be imported. A pattern can be made to include alphabets, numbers, punctuations as shown below.

A list of punctuations can be conveniently obtained by calling `string.punctuation`

```
# Import "re" module
import re, string

# Call string.punctuation
string.punctuation


>>[Out] '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

Next, we can build a pattern as:

```
# Build a regular expression pattern
re.sub(r'[^A-Za-z0-9 !]', '', messy)

>>[Out] 'The qick brown fx jumps over the lazy dg!'
```

Notice that this method simply got rid of all unwanted characters, which is not the desired result, as the meaning of the text has also got altered. Therefore, a careful review is needed before deleting characters.

Example 3: Using ASCII

A broad classification of ASCII codes can be done as shown in Table 1: ASCII code classification.

Type	ASCII Codes
Non-printable characters	0-31, 127
ASCII extended characters (special)	128 - 255
ASCII Printable characters	32 - 126

Table 1: ASCII code classification

Knowing these ranges, we can utilize the codes to remove unwanted characters using Translate method as shown below:

```
# Using ASCII codes from Table 1
np_sp = set(map(chr, list(range(0,32)) + list(range(127,256))))
ord_dict = {ord(character):None for character in np_sp}

def filter_np_sp(text):
    return text.translate(ord_dict)

clean_ascii = filter_np_sp(messy)
clean_ascii

>>[Out] 'The qick brown fx jumps over the lazy dg!'
```

An alternative to manually listing ASCII codes, can be simply using Python's `string.printable` attribute to generate a list of readily available printable characters.

```
string.printable

>>[Out] '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"
        #$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c'
```

Some of the latter characters in the above list falls under control characters, so the list can be slightly improvised to get a clean set of printable characters.

```
print_list = string.printable
excl_list  = '\t\n\r\x0b\x0c'

if excl_list in print_list:
    print_list = print_list.replace(excl_list,'')

print_list

>>[Out] '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"
        #$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ '
```

Finally, this clean **print_list** can be used to keep only the printable character set as shown below.

```
clean_print = ''.join([x if x in print_list else '' for x in messy])
clean_print

>>[Out] 'The qick brown fx jumps over the lazy dg!'
```

Just like Example 2: Using Regular Expressions, it got rid of all unwanted results which is not the desired outcome in this case. Instead of deleting special characters, flagging for review is a better solution and is covered in Example 6: Flagging Special Characters but dropping Control characters.

Example 4: Using Unicode

Unicode characters provides a rich set of characters beyond ASCII codes. Unicodes can also be grouped as shown in this partial output in Table 2. Complete list can be looked up as pointed out in REFERENCES section.

Abbr	Long	Description
Lu	Uppercase Letter	an uppercase letter
Ll	Lowercase Letter	a lowercase letter
Lt	Titlecase Latter	a digraphic character, with first part uppercase
Cc	Control	a C0 or C1 control code

Table 2: Unicode classification

For Unicode method, we first import 'unicode' library, and can also assess the number of unicodes available on the operating system.

```
# import unicode module
import unicodedata, sys

# print # of unicodes
print("{:,}".format(sys.maxunicode))

>>[Out] 1,114,111
```

Let's first remove control characters using '**Cc**' codes as provided in Table 2.

```
all_chars = (chr(i) for i in range(sys.maxunicode))
categories = {'Cc'}
control_chars = ''.join(c for c in all_chars if unicodedata.category(c) in
                        categories)

# use regular expressions

control_char_re = re.compile('[%s]' % re.escape(control_chars))

def remove_control_chars(s):
    return control_char_re.sub('', s)

clean_unc = remove_control_chars(messy)
clean_unc

>>[Out] 'The qüick brown fóx jumps over the lazy dóg!'
```

Looking at the above output, carriage return '**\r**' is no longer present in the string. For removing special characters or keeping only printable characters, unicode list is quite large to filter out irrelevant ones. For example, '**Lu**' code for uppercase characters (Table 2) will yield a vast list of uppercase characters which is beyond English alphabet list.

```
# Exploring UPPERCASE unicode character set with "Lu" (Table 2)
all_chars = (chr(i) for i in range(sys.maxunicode))
categories = {'Lu'}
uppercase_chars = ''.join(c for c in all_chars if unicodedata.category(c) in
                          categories)
uppercase_chars


>>[Out] 'ABCDEFGHIJKLMNOPQRSTUVWXYZÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÜÝÞĀĂĄĆĈČĎĐĒĔĖ
       ĘĚĜĞĠĢĤĦĨĪĬĮİIJĴĶĹĻĽĿŁŃŅŇŊŌŎŐŒŔŖŘŚŜŞŠŢŤŦŨŪŬŮŰŲŴŶŸŹŻŽƁƂƄƆƇƉƊƋƎƏƐƑƓƔƖƗ
       ƘƜƝƠƢƤƦƧƩƬƮƯƱƲƳƵƷƸƼǄǅǇǈǊǋǍǏǑǓǕǗǙǛǞǠǢǤǦǨǪǬǮǱǲǴǶǷǸǺǼǾȀȂȄȆȈȊȌȎȐȒȔȖ
       ȘȚȜȞȠȢȤȦȨȪȬȮȰȲȺȻȽȾɁɃɄɅɆɈɊɌɎͰͲͶͿΆΈΉΊΌΎΏΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩΪΫϏϘϚϜϞϠϢ
       ϤϦϨϪϬϮϴϷϹϺϽϾϿЀЁЂЃЄЅІЇЈЉЊЋЌЍЎЏАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯѠѢѤѦѨѪѬѮѰѲ
       ѴѶѸѺѼѾҀҊҌҎҐҒҔҖҘҚҜҞҠҢҤҦҨҪҬҮҰҲҴҶҸҺҼҾӀӁӃӅӇӉӋӍӐӒӔӖӘӚӜӞӠӢӤӦӨӪӬӮӰӲӴӶӸӺӼ
       ӾԀԂԄԆԈԊԌԎԐԒԔԖԘԚԜԞԠԢԤԦԨԪԬԮԱԲԳԴԵԶԷԸԹԺԻԼԽԾԿՀՁՂՃՄՅՆՇՈՉՊՋՌՍՎՏՐՑՒՓՔՕՖ
       ჀჁჂჃჄჅჇჍᎠᎡᎢᎣᎤᎥᎦᎧᎨᎩᎪᎫᎬᎭᎮᎯᎰᎱᎲᎳᎴᎵᎶᎷᎸᎹᎺᎻᎼᎽᎾᎿᏀᏁᏂᏃᏄᏅᏆᏇᏈᏉᏊᏋᏌᏍᏎᏏᏐᏑᏒᏓᏔᏕᏖᏗᏘᏙᏚᏛᏜᏝᏞᏟᏠᏡᏢᏣᏤᏥᏦᏧᏨᏩᏪᏫᏬᏭᏮᏯᏰᏱᏲᏳᏴᏵ
        ..................... many more ............................
```

Deleting special characters using Unicode method can especially useful when data is transmitted over formats allowing non-English like characters. However, in some instances, there could be some meaningful text that should be normalized to retain relevant information instead of dropping it as covered in Example 5: Normalization.

Example 5: Normalization

Unicode characters can be normalized to extract relevant information contained in them. Unicodes can be **Composed** (merge separate characters into a single unicode) or **Decomposed** (break a single unicode into separate characters). Based on this idea, 4 methods of normalization exist as - NFC, NFD, NFKC, and NFKD. More details can be looked up on the link provided in REFERENCES section.

Applying decomposition (NFKD) to our example results in:

```
# Using Normalization NFKD method
print(ascii(unicodedata.normalize("NFKD", messy)))

>[Out] 'The qu\u0308ick brown fo\u0301x jumps over the lazy do\u0301g!\r'
```

As evident from the result, **ü** is decomposed into 'u' and '\u0308', and likewise '**ó**' is decomposed into 'o' and '\u0301'. Next, these unwanted characters '\u0308' and '\u0301' can be easily removed as demonstrated below.

```
def norm_func(data):
    return ''.join(x for x in unicodedata.normalize('NFKD', data) if x in
                   print_list)

clean_norm = norm_func(messy)
clean_norm

>>[Out] 'The quick brown fox jumps over the lazy dog!'
```

This approach turns out to be the best solution for our test string so far without impacting any of its meaning.

Example 6: Flagging Special Characters but dropping Control characters

A good starting point is to flag special characters for review and adjudicate on a case-by-case basis. However, dropping control characters is a safe option as they don't alter the meaning of the text. In this example, we will import a CSV file (sample data provided in APPENDIX) and flag unwanted characters by storing them as a Python list in a new column in the revised CSV file.

```
# Read a CSV file called 'messy_file.csv' and write to 'clean_file.csv'

mask_print = {ord(char):None for char in print_list} # mask print_list

with open('messy_file.csv', errors='ignore') as csv_f:
    reader = csv.reader(csv_f)

    with open('clean_file.csv', 'w', newline='') as wf:
        csv_writer = csv.writer(wf, delimiter=',')
        for row in reader:
            spchr=[]
            data = [i.translate(ord_dict).strip() for i in row]
                                                  # exclude control chars
            spchr = [''.join(set(i.translate(mask_print))) for i in row]
                                                  # store special chars
            spchr = list(filter(None, set(spchr))) # filter empty strings

            if len(spchr)==0:
                final_data = data + spchr
            else:
                final_data = data + [spchr]

            csv_writer.writerow(final_data)
```

However, if all unwanted characters were simply removed without review in the above example, then the results would have been very different as can be observed by running this code.

```
# remove all unwanted characters

with open('messy_file.csv', encoding="utf-8", errors='ignore') as csv_f:
    reader = csv.reader(csv_f)
    with open('excl_all.csv', 'w', newline='') as wf:
        csv_writer = csv.writer(wf, delimiter=',')
        for row in reader:
            data = [i.encode('ascii',  'ignore').decode('ascii')
                      .translate(ord_dict).strip() for i in row]
            csv_writer.writerow(data)
```

CONCLUSION

The paper presented several options which can be considered when dealing with special and control characters. Example 6: Flagging Special Characters but dropping Control characters, approach could serve as a good starting reference when processing external files (csv, excel, etc) from a vendor that requires a detailed data quality check.

REFERENCES

[1] Unicode reference : http://www.unicode.org/reports/tr44/#General_Category_Values

[2] Unicode normalization : https://en.wikipedia.org/wiki/Unicode_equivalence#Normalization

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name : Abhinav Srivastva
Enterprise : Exelixis Inc | asrivastva@exelixis.com

APPENDIX

TEXT1	TEXT2	** Special chars **
Liver toxicity when bilirubin <= 1.5 × ULN	Otherwise normal	['×']
Capable of understanding and complying	Site error can be reported§	['§']
Investigator-assessed clinical benefit which outweighs the potential risks	Recovery to baseline or <= Grade 1 CTCAE v5 from toxicities	
Lábs taken from all sources when possible	Continue at the current dóse level with supportive care	['á', 'ó']
Test to be performed every 2 weeks	Known brain metastases or cranial epidural disease	
Platelets >= 100,000/µL without transfµsion	(@ST) >=3 ULN.	['µ']
This will be false positive	This will be false negative	['\n']
Radiographic tumor assessments are to continued	CT to be performed®	['®', '\n']
Radiation therapy for bone metastasis within 2 weeks	prior to any screening assessments	