Learning_Pandas_Part_6_CharacterOperations

June 20, 2021

0.0.1 Prepared by Abhishek Kumar

0.0.2 https://www.linkedin.com/in/abhishekkumar-0311/

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# To get multiple outputs in the same cell

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

%matplotlib inline
```

0.1 List of frequently used string functions

Function	Description	MS EXCEL FUNCTION
len( )	Calculate length of string	LEN( )
mystring[:N]	Extract N number of characters from start of string.	LEFT( )
mystring[-N:]	Extract N number of characters from end of string	RIGHT( )
mystring[X:Y]	Extract characters from middle of string, starting from X position and ends with Y	MID( )
str.split(),rsplit()	Split each string with the given pattern.	-
cat(sep=' ')	Concatenates the series/index elements with given separator.	-
separator.join(str)	Concatenate Strings	CONCATENATE( )
str.replace(old_substring, new_substring)	Replace a part of text with different sub-string	REPLACE( )
str.count('sub_string')	Count occurence of pattern in string	-

Function	Description	MS EXCEL FUNCTION
strip(),lstrip(),rstrip()	Helps strip whitespace(including newline) from each string in the Series/index from both the sides.	
repeat(value)	Repeats each element with specified number of times.	-
startswith(pattern)	Returns true if the element in the Series/Index starts with the pattern.	-
endswith(pattern)	Returns true if the element in the Series/Index ends with the pattern.	-
str.contains('pattern', case=False)	Returns a Boolean value True for each element if the substring contains in the element, else False	SQL LIKE Operator
match(pattern)	Determine if each string starts with a match of a regular expression.	-
fullmatch(pattern)	Stricter matching that requires the entire string to match.	-
index(pattern),rindex()	Return lowest indexes in each string in Series/Index.	-
find(pattern),rfind()	Returns the first position of the first occurrence of the pattern.	-
findall(pattern)	Returns a list of all occurrence of the pattern.	-
str.extract(regular_expression)	Return matched values (Pandas Function)	-
str.extractall(regular_expression)	Return matched values (Pandas Function)	-
str.zfill(n)	Pad strings in the Series/Index by prepending '0' characters.	-
str.ljust(width, fillchar=' ')	Fills the right side of strings with an arbitrary character.	-
str.rjust(width, fillchar=' ')	Fills the left side of strings with an arbitrary character.	-
str.center(width, fillchar=' ')	Fills both sides of strings with an arbitrary character.	-
str.pad(width,side='left',fillchar='0')	Pad strings in the Series/Index up to width.	-

Function	Description	MS EXCEL FUNCTION
str.lower()	Convert characters to lowercase	LOWER( )
str.upper()	Convert characters to uppercase	UPPER( )
str.swapcase()	Swaps the case lower/upper.	-
str.title()	Converts first character of each word to uppercase and remaining to lowercase.	-
str.capitalize()	Converts first character to uppercase and remaining to lowercase.	-
str.isalnum()	Check whether string consists of only alphanumeric characters	-
str.isdigit()	Check whether string consists of only digit characters	-
str.isalpha()	Check whether string consists of only alphabets characters	-
str.isdecimal()	Check whether string consists of only decimals characters	-
str.isnumeric()	Check whether string consists of only numeric characters	-
str.isspace()	Check whether string consists of only whitespace characters	-
str.islower()	Check whether characters are all lower case	-
str.isupper()	Check whether characters are all upper case	-
str.istitle()	Check whether characters are all title case	-

- https://www.listendata.com/2019/06/python-string-functions.html

0.2 1. Dataframe creation

[3]:
```
import numpy as np
import pandas as pd
sample = {
'col_a':['Houston,TX', 'Dallas,TX', 'Chicago,IL', 'Phoenix,AZ',      'San␣
 ↪Diego,CA'],
'col_b':['62K-70K', '62K-70K', '69K-76K', '62K-72K', '71K-78K' ],
```

```
'col_c':['A','B','A','a','c'],
'col_d':['  1x', ' 1y', '2x  ', '1x', '1y  ']
}
df_sample = pd.DataFrame(sample)
df_sample
```

[3]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y
2	Chicago,IL	69K-76K	A	2x
3	Phoenix,AZ	62K-72K	a	1x
4	San Diego,CA	71K-78K	c	1y

[4]:
```
# Setup Data
df = df_sample.copy()
df
```

[4]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y
2	Chicago,IL	69K-76K	A	2x
3	Phoenix,AZ	62K-72K	a	1x
4	San Diego,CA	71K-78K	c	1y

[ ]:
```

```

0.3 LEN()

- LENGTH() in SAS

[5]:
```
# Setup Data
df1 = df.copy()
df1

df1['LenOfColA'], df1['LenOfColB'] = df1['col_a'].str.len() ,  df1['col_b'].str.
↪len()
df1
```

[5]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y
2	Chicago,IL	69K-76K	A	2x
3	Phoenix,AZ	62K-72K	a	1x
4	San Diego,CA	71K-78K	c	1y

[5]:

	col_a	col_b	col_c	col_d	LenOfColA	LenOfColB
0	Houston,TX	62K-70K	A	1x	10	7

```
1     Dallas,TX  62K-70K     B    1y          9          7
2    Chicago,IL  69K-76K     A  2x           10          7
3    Phoenix,AZ  62K-72K     a    1x         10          7
4  San Diego,CA  71K-78K     c  1y           12          7
```

0.4 Slicing using Index

- SUBSTR in SAS

[6]:
```
# Setup Data
df2 = df.copy()
df2

df2['substring1'] = df2['col_a'].str[-2:]
df2
```

[6]:
```
          col_a    col_b col_c col_d
0    Houston,TX  62K-70K     A    1x
1     Dallas,TX  62K-70K     B    1y
2    Chicago,IL  69K-76K     A  2x
3    Phoenix,AZ  62K-72K     a    1x
4  San Diego,CA  71K-78K     c  1y
```

[6]:
```
          col_a    col_b col_c col_d substring1
0    Houston,TX  62K-70K     A    1x         TX
1     Dallas,TX  62K-70K     B    1y         TX
2    Chicago,IL  69K-76K     A  2x           IL
3    Phoenix,AZ  62K-72K     a    1x         AZ
4  San Diego,CA  71K-78K     c  1y           CA
```

[7]:
```
df2['substring2'] = df2['col_a'].str[0:3]
df2
```

[7]:
```
          col_a    col_b col_c col_d substring1 substring2
0    Houston,TX  62K-70K     A    1x         TX        Hou
1     Dallas,TX  62K-70K     B    1y         TX        Dal
2    Chicago,IL  69K-76K     A  2x           IL        Chi
3    Phoenix,AZ  62K-72K     a    1x         AZ        Pho
4  San Diego,CA  71K-78K     c  1y           CA        San
```

[8]:
```
df2['substring3'] = df2['col_a'].str[2::-1]
df2
```

[8]:
```
          col_a    col_b col_c col_d substring1 substring2 substring3
0    Houston,TX  62K-70K     A    1x         TX        Hou        uoH
1     Dallas,TX  62K-70K     B    1y         TX        Dal        laD
2    Chicago,IL  69K-76K     A  2x           IL        Chi        ihC
```

```
3    Phoenix,AZ  62K-72K     a    1x          AZ         Pho         ohP
4  San Diego,CA  71K-78K     c   1y           CA         San         naS
```

0.5 SPLIT()

- SCAN in SAS
- .str.split(pat=None, n=- 1, expand=False)[source]
- Parameters
  - pat - string or regex to split on
  - n - number of splits
  - expand - True/False : If True, return DataFrame/MultiIndex expanding dimensionality.
- https://pandas.pydata.org/docs/reference/api/pandas.Series.str.split.html
- https://pandas.pydata.org/docs/reference/api/pandas.Series.str.rsplit.html

[9]:
```
# Setup Data
df3 = df.copy()
df3

df3['City'] = df3['col_a'].str.split(',').str[0]
df3
df3['State'] = df3['col_a'].str.split(',').str[1]
df3
df3['State2'] = df3['col_a'].str.split(',').str.get(1)
df3
```

[9]:
```
           col_a    col_b col_c col_d
0    Houston,TX  62K-70K     A    1x
1     Dallas,TX  62K-70K     B    1y
2    Chicago,IL  69K-76K     A  2x
3    Phoenix,AZ  62K-72K     a    1x
4  San Diego,CA  71K-78K     c   1y
```

[9]:
```
           col_a    col_b col_c col_d       City
0    Houston,TX  62K-70K     A    1x    Houston
1     Dallas,TX  62K-70K     B    1y     Dallas
2    Chicago,IL  69K-76K     A  2x      Chicago
3    Phoenix,AZ  62K-72K     a    1x    Phoenix
4  San Diego,CA  71K-78K     c   1y    San Diego
```

[9]:
```
           col_a    col_b col_c col_d       City State
0    Houston,TX  62K-70K     A    1x    Houston    TX
1     Dallas,TX  62K-70K     B    1y     Dallas    TX
2    Chicago,IL  69K-76K     A  2x      Chicago    IL
3    Phoenix,AZ  62K-72K     a    1x    Phoenix    AZ
4  San Diego,CA  71K-78K     c   1y    San Diego    CA
```

[9]:

	col_a	col_b	col_c	col_d	City	State	State2
0	Houston,TX	62K-70K	A	1x	Houston	TX	TX
1	Dallas,TX	62K-70K	B	1y	Dallas	TX	TX
2	Chicago,IL	69K-76K	A	2x	Chicago	IL	IL
3	Phoenix,AZ	62K-72K	a	1x	Phoenix	AZ	AZ
4	San Diego,CA	71K-78K	c	1y	San Diego	CA	CA

0.5.1 Between different symbols

- The goal is to obtain the digits between two different symbols (the dash symbol and the dollar symbol)
  - First, set the variable (i.e., betweenTwoDifferentSymbols) to obtain all the characters after the dash symbol
  - Then, set the same variable to obtain all the characters before the dollar symbol

[10]:
```
Data = {'Identifier': ['IDAA-111$AA','IDB-2222222$B','IDCCC-33$CCC']}
dfi = pd.DataFrame(Data, columns= ['Identifier'])

Data
dfi

betweenTwoDifferentSymbols = dfi['Identifier'].str.split('-').str[1]
dfi['betweenTwoDifferentSymbols'] = betweenTwoDifferentSymbols.str.split('$').
 ↪str[0]

dfi
```

[10]:
```
{'Identifier': ['IDAA-111$AA', 'IDB-2222222$B', 'IDCCC-33$CCC']}
```

[10]:

	Identifier
0	IDAA-111$AA
1	IDB-2222222$B
2	IDCCC-33$CCC

[10]:

	Identifier	betweenTwoDifferentSymbols
0	IDAA-111$AA	111
1	IDB-2222222$B	2222222
2	IDCCC-33$CCC	33

0.6 CAT()

- catx, cats in SAS
- sep = ' , '
- na_rep = '?' - It replaces the NaN with question-mark (?)

0.6.1 Concat 2 columns

- In the same way, it could be extended for more columns

– df['combined']=df['bar']+'_'+df['foo']+'_'+df['new']

[11]:
```
# Setup Data
df4 = df3.copy()
df4

df4['Location1'] = df4['City'] + '-' + df4['State']
df4
```

[11]:

	col_a	col_b	col_c	col_d	City	State	State2
0	Houston,TX	62K-70K	A	1x	Houston	TX	TX
1	Dallas,TX	62K-70K	B	1y	Dallas	TX	TX
2	Chicago,IL	69K-76K	A	2x	Chicago	IL	IL
3	Phoenix,AZ	62K-72K	a	1x	Phoenix	AZ	AZ
4	San Diego,CA	71K-78K	c	1y	San Diego	CA	CA

[11]:

	col_a	col_b	col_c	col_d	City	State	State2	Location1
0	Houston,TX	62K-70K	A	1x	Houston	TX	TX	Houston-TX
1	Dallas,TX	62K-70K	B	1y	Dallas	TX	TX	Dallas-TX
2	Chicago,IL	69K-76K	A	2x	Chicago	IL	IL	Chicago-IL
3	Phoenix,AZ	62K-72K	a	1x	Phoenix	AZ	AZ	Phoenix-AZ
4	San Diego,CA	71K-78K	c	1y	San Diego	CA	CA	San Diego-CA

0.6.2 Concat 2 columns using series.str.cat()

- sep = ' , ' - To add separator in between columns
- na_rep = '?' - It replaces the NaN with question-mark (?)
- https://pandas.pydata.org/docs/reference/api/pandas.Series.str.cat.html

[12]:
```
df4['Location2'] = df4['City'].str.strip().str.cat(df4['State'].str.strip(),␣
↪sep= ':::')
df4
```

[12]:

	col_a	col_b	col_c	col_d	City	State	State2	Location1	\
0	Houston,TX	62K-70K	A	1x	Houston	TX	TX	Houston-TX	
1	Dallas,TX	62K-70K	B	1y	Dallas	TX	TX	Dallas-TX	
2	Chicago,IL	69K-76K	A	2x	Chicago	IL	IL	Chicago-IL	
3	Phoenix,AZ	62K-72K	a	1x	Phoenix	AZ	AZ	Phoenix-AZ	
4	San Diego,CA	71K-78K	c	1y	San Diego	CA	CA	San Diego-CA	

	Location2
0	Houston:::TX
1	Dallas:::TX
2	Chicago:::IL
3	Phoenix:::AZ
4	San Diego:::CA

0.6.3 Concat 2 or more columns

- 2 or more columns can be added by providing list of values as shown below

[13]:
```
df4['Cat3'] = df4['City'].str.strip().str.cat([df4['State'],df4['col_c']], sep=␣
↪':::')
df4
```

[13]:
```
          col_a    col_b col_c col_d       City State State2     Location1  \
0    Houston,TX  62K-70K     A    1x    Houston    TX     TX    Houston-TX
1     Dallas,TX  62K-70K     B    1y     Dallas    TX     TX     Dallas-TX
2    Chicago,IL  69K-76K     A  2x      Chicago    IL     IL    Chicago-IL
3    Phoenix,AZ  62K-72K     a    1x    Phoenix    AZ     AZ    Phoenix-AZ
4  San Diego,CA  71K-78K     c  1y    San Diego    CA     CA  San Diego-CA

        Location2               Cat3
0    Houston:::TX    Houston:::TX:::A
1     Dallas:::TX     Dallas:::TX:::B
2    Chicago:::IL    Chicago:::IL:::A
3    Phoenix:::AZ    Phoenix:::AZ:::a
4  San Diego:::CA  San Diego:::CA:::c
```

0.6.4 Technique to perform concat on non-string columns

- `df['combined']=df['bar'].astype(str)+'_'+df['foo']+'_'+df['new']`
- `df['Full Date'] = df['Day'].map(str) + '-' + df['Month'].map(str) + '-' + df['Year'].map(str)`

0.7 STR.JOIN()

[14]:
```
# Setup Data
df5 = df3.copy()
df5
```

[14]:
```
          col_a    col_b col_c col_d       City State State2
0    Houston,TX  62K-70K     A    1x    Houston    TX     TX
1     Dallas,TX  62K-70K     B    1y     Dallas    TX     TX
2    Chicago,IL  69K-76K     A  2x      Chicago    IL     IL
3    Phoenix,AZ  62K-72K     a    1x    Phoenix    AZ     AZ
4  San Diego,CA  71K-78K     c  1y    San Diego    CA     CA
```

[ ]:
```

```

0.8 REPLACE()

- TRANWRD in SAS
- str.replace(old_text,new_text,case=False) is used to replace a particular character(s) or pattern with some new value or pattern.

- .str.replace(pat, repl, n=- 1, case=None, flags=0, regex=None)[source]
- https://pandas.pydata.org/docs/reference/api/pandas.Series.str.replace.html

[15]:
```
# Setup Data
df6 = df.copy()
df6

df6['NewColA1'] = df6['col_a'].str.replace('TX', 'Abhishek')
df6
df6['NewColA2'] = df6['col_a'].str.replace('tX', ' ', case= False)
df6
df6['LenNewColA2'] = df6['NewColA2'].str.len()
df6
```

[15]:
```
          col_a    col_b col_c col_d
0    Houston,TX  62K-70K     A    1x
1     Dallas,TX  62K-70K     B    1y
2    Chicago,IL  69K-76K     A  2x
3    Phoenix,AZ  62K-72K     a    1x
4  San Diego,CA  71K-78K     c  1y
```

[15]:
```
          col_a    col_b col_c col_d          NewColA1
0    Houston,TX  62K-70K     A    1x  Houston,Abhishek
1     Dallas,TX  62K-70K     B    1y   Dallas,Abhishek
2    Chicago,IL  69K-76K     A  2x          Chicago,IL
3    Phoenix,AZ  62K-72K     a    1x        Phoenix,AZ
4  San Diego,CA  71K-78K     c  1y        San Diego,CA
```

[15]:
```
          col_a    col_b col_c col_d          NewColA1      NewColA2
0    Houston,TX  62K-70K     A    1x  Houston,Abhishek     Houston,
1     Dallas,TX  62K-70K     B    1y   Dallas,Abhishek      Dallas,
2    Chicago,IL  69K-76K     A  2x          Chicago,IL    Chicago,IL
3    Phoenix,AZ  62K-72K     a    1x        Phoenix,AZ    Phoenix,AZ
4  San Diego,CA  71K-78K     c  1y        San Diego,CA  San Diego,CA
```

[15]:
```
          col_a    col_b col_c col_d          NewColA1      NewColA2  \
0    Houston,TX  62K-70K     A    1x  Houston,Abhishek     Houston,
1     Dallas,TX  62K-70K     B    1y   Dallas,Abhishek      Dallas,
2    Chicago,IL  69K-76K     A  2x          Chicago,IL    Chicago,IL
3    Phoenix,AZ  62K-72K     a    1x        Phoenix,AZ    Phoenix,AZ
4  San Diego,CA  71K-78K     c  1y        San Diego,CA  San Diego,CA

   LenNewColA2
0            9
1            8
2           10
3           10
4           12
```

[16]:
```
# Replacing 1st word of col_a with Mumbai

# df['NewColA'] = df['col_a'].str.replace(df['col_a'].str.split(',').str[0],␣
↪'Mumbai')
# df
```

0.9 COUNT( )

- It returns the count of the appearance of pattern in each element in Data-Frame like below in example it counts 'n' in each string of DataFrame and returns the total counts of 'n' in each string.
- In SAS
  - COUNT : Count characters
  - COUNTW : Count words in SAS
  - COUNTC : Count specific character in SAS

[17]:
```
# Setup Data
df7 = df.copy()
df7

import re
df7['CountA'] = df7['col_a'].str.count('c',re.I)
df7
```

[17]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y
2	Chicago,IL	69K-76K	A	2x
3	Phoenix,AZ	62K-72K	a	1x
4	San Diego,CA	71K-78K	c	1y

[17]:

	col_a	col_b	col_c	col_d	CountA
0	Houston,TX	62K-70K	A	1x	0
1	Dallas,TX	62K-70K	B	1y	0
2	Chicago,IL	69K-76K	A	2x	2
3	Phoenix,AZ	62K-72K	a	1x	0
4	San Diego,CA	71K-78K	c	1y	1

0.9.1 Count No. of Words

[18]:
```
df7['NoOfWords1'] = [len(x.split(',')) for x in df7['col_a'].tolist()]
df7
```

[18]:

	col_a	col_b	col_c	col_d	CountA	NoOfWords1
0	Houston,TX	62K-70K	A	1x	0	2
1	Dallas,TX	62K-70K	B	1y	0	2
2	Chicago,IL	69K-76K	A	2x	2	2

```
3    Phoenix,AZ  62K-72K     a    1x       0           2
4  San Diego,CA  71K-78K     c  1y         1           2
```

[19]:
```
df7['NoOfWords2'] = df7['col_a'].str.split(',').str.len()
df7
df7['NoOfWords3'] = df7['col_a'].str.split(',').apply(len)
df7
```

[19]:
```
           col_a    col_b col_c col_d  CountA  NoOfWords1  NoOfWords2
0    Houston,TX  62K-70K     A    1x       0           2           2
1     Dallas,TX  62K-70K     B    1y       0           2           2
2    Chicago,IL  69K-76K     A  2x         2           2           2
3    Phoenix,AZ  62K-72K     a    1x       0           2           2
4  San Diego,CA  71K-78K     c  1y         1           2           2
```

[19]:
```
           col_a    col_b col_c col_d  CountA  NoOfWords1  NoOfWords2  \
0    Houston,TX  62K-70K     A    1x       0           2           2
1     Dallas,TX  62K-70K     B    1y       0           2           2
2    Chicago,IL  69K-76K     A  2x         2           2           2
3    Phoenix,AZ  62K-72K     a    1x       0           2           2
4  San Diego,CA  71K-78K     c  1y         1           2           2

   NoOfWords3
0           2
1           2
2           2
3           2
4           2
```

[20]:
```
import re
df7['NoOfWords4'] = df7['col_a'].str.count('\w+')
df7
```

[20]:
```
           col_a    col_b col_c col_d  CountA  NoOfWords1  NoOfWords2  \
0    Houston,TX  62K-70K     A    1x       0           2           2
1     Dallas,TX  62K-70K     B    1y       0           2           2
2    Chicago,IL  69K-76K     A  2x         2           2           2
3    Phoenix,AZ  62K-72K     a    1x       0           2           2
4  San Diego,CA  71K-78K     c  1y         1           2           2

   NoOfWords3  NoOfWords4
0           2           2
1           2           2
2           2           2
3           2           2
4           2           3
```

[21]:
```
import re
df7['NoOfWords5'] = df7['col_a'].str.count('^[cpiad]\w+', re.I)
df7
```

[21]:

	col_a	col_b	col_c	col_d	CountA	NoOfWords1	NoOfWords2	\
0	Houston,TX	62K-70K	A	1x	0	2	2	
1	Dallas,TX	62K-70K	B	1y	0	2	2	
2	Chicago,IL	69K-76K	A	2x	2	2	2	
3	Phoenix,AZ	62K-72K	a	1x	0	2	2	
4	San Diego,CA	71K-78K	c	1y	1	2	2	

	NoOfWords3	NoOfWords4	NoOfWords5
0	2	2	0
1	2	2	1
2	2	2	1
3	2	2	1
4	2	3	0

0.10 STRIP(' ')

- strip(), lstrip(), rstrip()
- Stripping is like trimming tree branches. We can remove spaces or any other characters at the beginning or end of a string. like, strip('$') - remove dollar sign from both left and right side
- STRIP, TRIM in SAS

[22]:
```
# Setup Data
df8 = df.copy()
df8

df8['NewColB'] = df8['col_b'].str.strip('K')
df8
```

[22]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y
2	Chicago,IL	69K-76K	A	2x
3	Phoenix,AZ	62K-72K	a	1x
4	San Diego,CA	71K-78K	c	1y

[22]:

	col_a	col_b	col_c	col_d	NewColB
0	Houston,TX	62K-70K	A	1x	62K-70
1	Dallas,TX	62K-70K	B	1y	62K-70
2	Chicago,IL	69K-76K	A	2x	69K-76
3	Phoenix,AZ	62K-72K	a	1x	62K-72
4	San Diego,CA	71K-78K	c	1y	71K-78

0.11 REPEAT()

- REPEAT() in SAS

[23]:
```
# Setup Data
df9 = df.copy()
df9

df9['RepeatColD'] = df9['col_d'].str.repeat(3)
df9
```

[23]:
```
          col_a    col_b col_c col_d
0    Houston,TX  62K-70K     A    1x
1     Dallas,TX  62K-70K     B    1y
2    Chicago,IL  69K-76K     A  2x
3    Phoenix,AZ  62K-72K     a    1x
4  San Diego,CA  71K-78K     c  1y
```

[23]:
```
          col_a    col_b col_c col_d    RepeatColD
0    Houston,TX  62K-70K     A    1x    1x  1x  1x
1     Dallas,TX  62K-70K     B    1y      1y 1y 1y
2    Chicago,IL  69K-76K     A  2x    2x  2x  2x
3    Phoenix,AZ  62K-72K     a    1x        1x1x1x
4  San Diego,CA  71K-78K     c  1y    1y  1y  1y
```

0.12 STARTSWITH()

- .str.startswith(pat, na=None)
- na = False : Specifying na to be False instead of NaN.
- Using SUBSTR() in SAS
- https://pandas.pydata.org/docs/reference/api/pandas.Series.str.startswith.html

[24]:
```
# Setup Data
df10 = df.copy()
df10

df10[df10.col_a.str.startswith('C')]
```

[24]:
```
          col_a    col_b col_c col_d
0    Houston,TX  62K-70K     A    1x
1     Dallas,TX  62K-70K     B    1y
2    Chicago,IL  69K-76K     A  2x
3    Phoenix,AZ  62K-72K     a    1x
4  San Diego,CA  71K-78K     c  1y
```

[24]:
```
         col_a    col_b col_c col_d
2  Chicago,IL  69K-76K     A  2x
```

0.12.1 It's possible to pass a tuple of prefixes to startswith() method in Python.

- If the string starts with any item of the tuple, startswith() returns True. If not, it returns False

[25]:
```
df10[df10.col_a.str.startswith(('C','P'))]
```

[25]:
```
        col_a    col_b col_c col_d
2  Chicago,IL  69K-76K     A  2x
3  Phoenix,AZ  62K-72K     a    1x
```

0.13 ENDSWITH()

- .str.endswith(pat, na=None)[source]
- na = False : Specifying na to be False instead of NaN.
- Using SUBSTR() in SAS
- https://pandas.pydata.org/docs/reference/api/pandas.Series.str.endswith.html

[26]:
```
# Setup Data
df11 = df.copy()
df11

df11[df11.col_a.str.endswith('X')]
```

[26]:
```
           col_a    col_b col_c col_d
0    Houston,TX  62K-70K     A    1x
1     Dallas,TX  62K-70K     B    1y
2    Chicago,IL  69K-76K     A  2x
3    Phoenix,AZ  62K-72K     a    1x
4  San Diego,CA  71K-78K     c  1y
```

[26]:
```
         col_a    col_b col_c col_d
0  Houston,TX  62K-70K     A    1x
1   Dallas,TX  62K-70K     B    1y
```

0.14 CONTAINS()

- Using CONTAINS() in SAS
- .str.contains(pat, case=True, flags=0, na=None, regex=True)[source]
- Test if pattern or regex is contained within a string of a Series or Index.
- Return boolean Series or Index based on whether a given pattern or regex is contained within a string of a Series or Index.
- Parameters
  - pat : str : Character sequence or **regular expression**.
  - case : bool : default True If True, case sensitive.
  - flags : int : default 0 (no flags) Flags to pass through to the re module, e.g. re.IGNORECASE.

- na : scalar : optional Fill value for missing values, eg False . The default depends on dtype of the array. For object-dtype, numpy.nan is used. For StringDtype, pandas.NA is used.
- regex : bool : default True If True, assumes the pat is a regular expression.
  - If False, treats the pat as a literal string.

[27]:
```
# Setup Data
df12 = df.copy()
df12
```

[27]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y
2	Chicago,IL	69K-76K	A	2x
3	Phoenix,AZ	62K-72K	a	1x
4	San Diego,CA	71K-78K	c	1y

[28]:
```
bool = df12.col_a.str.contains('oU', flags = re.IGNORECASE)
df12[bool]
```

[28]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x

[29]:
```
bool = df12.col_a.str.contains(('tx|ca'), case = False)
df12[bool]
```

[29]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y
2	Chicago,IL	69K-76K	A	2x
4	San Diego,CA	71K-78K	c	1y

[30]:
```
bool = df12.col_b.str.contains(('^[0-6].*'), case = False)
df12[bool]
```

[30]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y
2	Chicago,IL	69K-76K	A	2x
3	Phoenix,AZ	62K-72K	a	1x

[31]:
```
bool = df12.col_a.str.contains('z', flags = re.IGNORECASE)
df12[bool]
```

[31]:

	col_a	col_b	col_c	col_d
3	Phoenix,AZ	62K-72K	a	1x

[32]:
```
# Ending with TX , ignore case

bool = df12.col_a.str.contains('tx$', flags = re.IGNORECASE, regex=True)
df12[bool]
```

[32]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y

[33]:
```
# Ending with 2 characters , ignore case

bool = df12.col_a.str.contains('\w{2}$', flags = re.IGNORECASE, regex=True)
df12[bool]
```

[33]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y
2	Chicago,IL	69K-76K	A	2x
3	Phoenix,AZ	62K-72K	a	1x
4	San Diego,CA	71K-78K	c	1y

0.15 str.MATCH

- str.match(pat, case=True, flags=0, na=None)[source]
- Determine if each string starts with a match of a regular expression.
- Parameters
  - pat : Character sequence or **regular expression**.
  - case : bool : True, if case sensitive
  - flags : Regex module flags, e.g. re.IGNORECASE.
  - na : Fill value for missing values.
- Returns : Series/array of boolean values

[34]:
```
# Setup Data
df13 = df.copy()
df13
```

[34]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	Dallas,TX	62K-70K	B	1y
2	Chicago,IL	69K-76K	A	2x
3	Phoenix,AZ	62K-72K	a	1x
4	San Diego,CA	71K-78K	c	1y

[35]:
```
bool = df13.col_a.str.match('h', flags = re.IGNORECASE)
df13[bool]
```

[35]:
```
         col_a    col_b col_c col_d
0  Houston,TX  62K-70K     A    1x
```

[36]:
```
bool = df13.col_a.str.match('h', case = False)
df13[bool]
```

[36]:
```
         col_a    col_b col_c col_d
0  Houston,TX  62K-70K     A    1x
```

[37]:
```
bool = df13.col_b.str.match('62.*k', case = False)
df13[bool]
```

[37]:
```
         col_a    col_b col_c col_d
0  Houston,TX  62K-70K     A    1x
1   Dallas,TX  62K-70K     B    1y
3  Phoenix,AZ  62K-72K     a    1x
```

0.16 str.FULLMATCH

- str.fullmatch(pat, case=True, flags=0, na=None)[source]
- Determine if each string starts with a match of a regular expression.
- Parameters
  - pat : Character sequence or **regular expression**.
  - case : bool : True, if case sensitive
  - flags : Regex module flags, e.g. re.IGNORECASE.
  - na : Fill value for missing values.
- Returns : Series/array of boolean values

[38]:
```
bool = df13.col_b.str.fullmatch('62.*')
df13[bool]
```

[38]:
```
         col_a    col_b col_c col_d
0  Houston,TX  62K-70K     A    1x
1   Dallas,TX  62K-70K     B    1y
3  Phoenix,AZ  62K-72K     a    1x
```

[39]:
```
bool = df13.col_b.str.fullmatch('62K-70k', case= False)
df13[bool]
```

[39]:
```
         col_a    col_b col_c col_d
0  Houston,TX  62K-70K     A    1x
1   Dallas,TX  62K-70K     B    1y
```

0.17 str.INDEX

- Series.str.index(sub, start=0, end=None)[source]
- Return lowest indexes in each string in Series/Index.
- Returns ValueError on failure, when the substring is not found.

- Parameters
  - sub - substring being searched
  - start - int, left edge index
  - end - int, right edge index

[40]:
```
# Setup Data
df14 = df.copy()
df14
```

[40]:
```
          col_a    col_b col_c col_d
0    Houston,TX  62K-70K     A    1x
1     Dallas,TX  62K-70K     B    1y
2    Chicago,IL  69K-76K     A  2x
3    Phoenix,AZ  62K-72K     a    1x
4  San Diego,CA  71K-78K     c  1y
```

[41]:
```
# Its not working as expected

# bool = df14.col_a.str.index('Dallas',start=0)
# df14[bool]
```

0.18 str.FIND

- Series.str.find(sub, start=0, end=None)[source]
- Return lowest indexes in each string in Series/Index.
- Return -1 on failure, unlike INDEX(), which returns ValueError
- Parameters
  - sub - substring being searched
  - start - int, left edge index
  - end - int, right edge index
- RFIND() - Return highest indexes in each strings in the Series/Index.

[42]:
```
# Setup Data
df15 = df.copy()
df15
```

[42]:
```
          col_a    col_b col_c col_d
0    Houston,TX  62K-70K     A    1x
1     Dallas,TX  62K-70K     B    1y
2    Chicago,IL  69K-76K     A  2x
3    Phoenix,AZ  62K-72K     a    1x
4  San Diego,CA  71K-78K     c  1y
```

[43]:
```
df15['isA'] = df15.col_a.str.find('Dallas',start=0, end=6)
df15
```

[43]:
```
           col_a    col_b col_c col_d  isA
0    Houston,TX  62K-70K     A    1x   -1
1     Dallas,TX  62K-70K     B    1y    0
2    Chicago,IL  69K-76K     A  2x     -1
3    Phoenix,AZ  62K-72K     a    1x   -1
4  San Diego,CA  71K-78K     c  1y     -1
```

[44]:
```
df15['isA1'] = df15.col_a.str.find('TX')
df15
```

[44]:
```
           col_a    col_b col_c col_d  isA  isA1
0    Houston,TX  62K-70K     A    1x   -1     8
1     Dallas,TX  62K-70K     B    1y    0     7
2    Chicago,IL  69K-76K     A  2x     -1    -1
3    Phoenix,AZ  62K-72K     a    1x   -1    -1
4  San Diego,CA  71K-78K     c  1y     -1    -1
```

[45]:
```
df15['isA2'] = df15.col_a.str.rfind('San')
df15
```

[45]:
```
           col_a    col_b col_c col_d  isA  isA1  isA2
0    Houston,TX  62K-70K     A    1x   -1     8    -1
1     Dallas,TX  62K-70K     B    1y    0     7    -1
2    Chicago,IL  69K-76K     A  2x     -1    -1    -1
3    Phoenix,AZ  62K-72K     a    1x   -1    -1    -1
4  San Diego,CA  71K-78K     c  1y     -1    -1     0
```

0.19 str.FINDALL

- Series.str.findall(pat, flags=0)[source]
- Find all occurrences of pattern or regular expression in the Series/Index.
- Equivalent to applying re.findall() to all the elements in the Series/Index.
- Parameters
  - pat - Pattern or **regular expression**.
  - flags - Flags from re module, e.g. re.IGNORECASE (default is 0, which means no flags).
- Return - All non-overlapping matches of pattern or regular expression in each string of this Series/Index.

[46]:
```
# Setup Data
df16 = df.copy()
df16
```

[46]:
```
           col_a    col_b col_c col_d
0    Houston,TX  62K-70K     A    1x
1     Dallas,TX  62K-70K     B    1y
2    Chicago,IL  69K-76K     A  2x
3    Phoenix,AZ  62K-72K     a    1x
```

```
4  San Diego,CA  71K-78K     c  1y
```

0.19.1 If the pattern is found more than once in the same string, then a list of multiple strings is returned:

[47]:
```
matches = df16['col_a'].str.repeat(3).str.findall('Go', flags=re.I)
type(matches)
matches
matches[2][2]
```

[47]:
```
pandas.core.series.Series
```

[47]:
```
0              []
1              []
2    [go, go, go]
3              []
4    [go, go, go]
Name: col_a, dtype: object
```

[47]:
```
'go'
```

[48]:
```
matches = df16['col_a'].str.repeat(3).str.findall('go', flags=re.I)
type(matches)
matches
matches[:][2]
```

[48]:
```
pandas.core.series.Series
```

[48]:
```
0              []
1              []
2    [go, go, go]
3              []
4    [go, go, go]
Name: col_a, dtype: object
```

[48]:
```
['go', 'go', 'go']
```

0.20 str.extract

Series.str.extract(pat, flags=0, expand=True)[source] - Extract capture groups in the regex pat as columns in a DataFrame.

- For each subject string in the Series, extract groups from the first match of regular expression pat.
- Parameters
  - patstr - Regular expression pattern with capturing groups.

  - flagsint, default 0 (no flags) - Flags from the re module, e.g. re.IGNORECASE, that modify regular expression matching for things like case, spaces, etc. For more details, see re.
  - expandbool, default True - If True, return DataFrame with one column per capture group. If False, return a Series/Index if there is one capture group or DataFrame if there are multiple capture groups.
- Returns - DataFrame or Series or Index
  - A DataFrame with one row for each subject string, and one column for each group. Any capture group names in regular expression pat will be used for column names; otherwise capture group numbers will be used. The dtype of each result column is always object, even when no match is found. If expand=False and pat has only one capture group, then return a Series (if subject is a Series) or Index (if subject is an Index).
- https://pandas.pydata.org/docs/reference/api/pandas.Series.str.extract.html

[ ]:

0.21 str.extractall

- Series.str.extractall(pat, flags=0)[source]
- Extract capture groups in the regex pat as columns in DataFrame.
- For each subject string in the Series, extract groups from all matches of regular expression pat. When each subject string in the Series has exactly one match, extractall(pat).xs(0, level='match') is the same as extract(pat).
- Parameters
  - patstr - Regular expression pattern with capturing groups.
  - flagsint, default 0 (no flags)
  - A re module flag, for example re.IGNORECASE. These allow to modify regular expression matching for things like case, spaces, etc. Multiple flags can be combined with the bitwise OR operator, for example re.IGNORECASE | re.MULTILINE.
- Returns - DataFrame
  - A DataFrame with one row for each match, and one column for each group. Its rows have a MultiIndex with first levels that come from the subject Series. The last level is named 'match' and indexes the matches in each item of the Series. Any capture group names in regular expression pat will be used for column names; otherwise capture group numbers will be used.
- https://pandas.pydata.org/docs/reference/api/pandas.Series.str.extractall.html#pandas.Series.str.extractal

[ ]:

0.22 https://pandas.pydata.org/docs/reference/api/pandas.Series.str.zfill.html

Function	Description	MS EXCEL FUNCTION
str.zfill(n)	Pad strings in the Series/Index by prepending '0' characters.	-
str.ljust(width, fillchar=' ')	Fills the right side of strings with an arbitrary character.	-
str.rjust(width, fillchar=' ')	Fills the left side of strings with an arbitrary character.	-
str.center(width, fillchar=' ')	Fills both sides of strings with an arbitrary character.	-
str.pad(width,side='left',fillchar=' ')	Pad strings in the Series/Index up to width.	-

0.22.1 str.zfill(width)[source]

- width - Minimum length of resulting string

0.22.2 str.rjust(width, fillchar=' ')[source]

- Pad left side of strings in the Series/Index.
- width - Minimum width of resulting string
- fillchar - Additional character for filling, default is whitespace.

0.22.3 str.ljust(width, fillchar=' ')[source]

- Pad right side of strings in the Series/Index.
- width - Minimum width of resulting string
- fillchar - Additional character for filling, default is whitespace.

0.22.4 str.center(width, fillchar=' ')[source]

- Pad left and right side of strings in the Series/Index.
- width - Minimum width of resulting string
- fillchar - Additional character for filling, default is whitespace.

0.22.5 .str.pad(width, side='left', fillchar=' ')[source]

- Pad strings in the Series/Index up to width.
- Parameters
  - width - int, minimum width of resulting string
  - side - left, right, both - Side from which to fill resulting string.
  - fillchar - Additional character for filling, default is whitespace.
- Series.str.pad(side='left') : Series.str.rjust
- Series.str.pad(side='right') : Series.str.ljust
- Series.str.pad(side='both') : Series.str.center
- Series.str.pad(side='left', fillchar='0') : Series.str.zfill

[ ]:

0.23 https://pandas.pydata.org/docs/reference/api/pandas.Series.str.lower.html

Function	Description	MS EXCEL FUNCTION
str.lower()	Convert characters to lowercase	LOWER( )
str.upper()	Convert characters to uppercase	UPPER( )
str.swapcase()	Swaps the case lower/upper.	-
str.title()	Converts first character of each word to uppercase and remaining to lowercase.	-
str.capitalize()	Converts first character to uppercase and remaining to lowercase.	-

[ ]:

0.24 https://pandas.pydata.org/docs/reference/api/pandas.Series.str.isalpha.html

Function	Description	MS EXCEL FUNCTION
str.isalnum()	Check whether string consists of only alphanumeric characters	-
str.isdigit()	Check whether string consists of only digit characters	-
str.isalpha()	Check whether string consists of only alphabets characters	-
str.isdecimal()	Check whether string consists of only decimals characters	-
str.isnumeric()	Check whether string consists of only numeric characters	-
str.isspace()	Check whether string consists of only whitespace characters	-
str.islower()	Check whether characters are all lower case	-
str.isupper()	Check whether characters are all upper case	-
str.istitle()	Check whether characters are all title case	-

[ ]:

1 Object vs String

Before pandas 1.0, only "object" datatype was used to store strings which cause some drawbacks because non-string data can also be stored using "object" datatype. Pandas 1.0 introduces a new datatype specific to string data which is StringDtype. As of now, we can still use object or StringDtype to store strings but in the future, we may be required to only use StringDtype. One important thing to note here is that object datatype is still the default datatype for strings. To use StringDtype, we need to explicitly state it. We can pass "string" or pd.StringDtype() argument to dtype parameter to string datatype.

[ ]:
```
# ![image.png](attachment:image.png)
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

1.1 Problem Solving

[49]:
```
# Replacing 2nd word of col_a with 1st word of col_a

dfp = df.copy()
dfp

dfp['DupA'] = dfp['col_a']
dfp

x = dfp.col_a.str.split(',').str[0]

def func(row):
    return row['DupA'].replace(row['DupA'].split(',')[1], row['DupA'].
 ↪split(',')[0])


dfp['DupA'] = dfp.apply(func, axis = 1)
```

```
dfp

d2= dfp.apply(func, axis = 1)

d2
```

[49]:
```
           col_a    col_b col_c col_d
0    Houston,TX  62K-70K     A    1x
1     Dallas,TX  62K-70K     B    1y
2    Chicago,IL  69K-76K     A  2x
3    Phoenix,AZ  62K-72K     a    1x
4  San Diego,CA  71K-78K     c  1y
```

[49]:
```
           col_a    col_b col_c col_d          DupA
0    Houston,TX  62K-70K     A    1x    Houston,TX
1     Dallas,TX  62K-70K     B    1y     Dallas,TX
2    Chicago,IL  69K-76K     A  2x      Chicago,IL
3    Phoenix,AZ  62K-72K     a    1x    Phoenix,AZ
4  San Diego,CA  71K-78K     c  1y    San Diego,CA
```

[49]:
```
           col_a    col_b col_c col_d                 DupA
0    Houston,TX  62K-70K     A    1x      Houston,Houston
1     Dallas,TX  62K-70K     B    1y        Dallas,Dallas
2    Chicago,IL  69K-76K     A  2x        Chicago,Chicago
3    Phoenix,AZ  62K-72K     a    1x      Phoenix,Phoenix
4  San Diego,CA  71K-78K     c  1y    San Diego,San Diego
```

[49]:
```
0        Houston,Houston
1          Dallas,Dallas
2        Chicago,Chicago
3        Phoenix,Phoenix
4    San Diego,San Diego
dtype: object
```

[50]:
```
# Data Setup

df = dfp.copy()
df
```

[50]:
```
           col_a    col_b col_c col_d                 DupA
0    Houston,TX  62K-70K     A    1x      Houston,Houston
1     Dallas,TX  62K-70K     B    1y        Dallas,Dallas
2    Chicago,IL  69K-76K     A  2x        Chicago,Chicago
3    Phoenix,AZ  62K-72K     a    1x      Phoenix,Phoenix
4  San Diego,CA  71K-78K     c  1y    San Diego,San Diego
```

Replacing 2nd word of col_a with 1st word of col_a

[51]:
```
def func(row):
    return row['col_a'].replace(row['col_a'].split(',')[1],row['col_a'].
 ↪split(',')[0] )

df['NewColA1'] = df.apply(func, axis=1)
df
```

[51]:

	col_a	col_b	col_c	col_d	DupA	NewColA1
0	Houston,TX	62K-70K	A	1x	Houston,Houston	Houston,Houston
1	Dallas,TX	62K-70K	B	1y	Dallas,Dallas	Dallas,Dallas
2	Chicago,IL	69K-76K	A	2x	Chicago,Chicago	Chicago,Chicago
3	Phoenix,AZ	62K-72K	a	1x	Phoenix,Phoenix	Phoenix,Phoenix
4	San Diego,CA	71K-78K	c	1y	San Diego,San Diego	San Diego,San Diego

Replacing 2nd word of col_a with a constant value '_IN'

[52]:
```
def func(row):
    return row['col_a'].replace(row['col_a'].split(',')[1],'_IN' )

df['NewColA2'] = df.apply(func, axis=1)
df
```

[52]:

	col_a	col_b	col_c	col_d	DupA	\
0	Houston,TX	62K-70K	A	1x	Houston,Houston	
1	Dallas,TX	62K-70K	B	1y	Dallas,Dallas	
2	Chicago,IL	69K-76K	A	2x	Chicago,Chicago	
3	Phoenix,AZ	62K-72K	a	1x	Phoenix,Phoenix	
4	San Diego,CA	71K-78K	c	1y	San Diego,San Diego	

	NewColA1	NewColA2
0	Houston,Houston	Houston,_IN
1	Dallas,Dallas	Dallas,_IN
2	Chicago,Chicago	Chicago,_IN
3	Phoenix,Phoenix	Phoenix,_IN
4	San Diego,San Diego	San Diego,_IN

Splitting 1st word of col_a to a new column

[53]:
```
def func(row):
    return row['col_a'].split(',')[0]

df['NewColA3'] = df.apply(func, axis=1)
df
```

[53]:

	col_a	col_b	col_c	col_d	DupA	\
0	Houston,TX	62K-70K	A	1x	Houston,Houston	
1	Dallas,TX	62K-70K	B	1y	Dallas,Dallas	
2	Chicago,IL	69K-76K	A	2x	Chicago,Chicago	

```
3    Phoenix,AZ  62K-72K     a    1x      Phoenix,Phoenix
4  San Diego,CA  71K-78K     c  1y    San Diego,San Diego

              NewColA1       NewColA2   NewColA3
0      Houston,Houston    Houston,_IN    Houston
1        Dallas,Dallas     Dallas,_IN     Dallas
2      Chicago,Chicago    Chicago,_IN    Chicago
3      Phoenix,Phoenix    Phoenix,_IN    Phoenix
4  San Diego,San Diego  San Diego,_IN  San Diego
```

Replacing 2nd word of col_a with a constant value '_IN' but NOT using REPLACE -

- **_instead use SPLIT to extract 1st word and CONCAT with the constant value '*IN*'**

[54]:
```
def func(row):
    return row['col_a'].split(',')[0] + '_IN'

df['NewColA4'] = df.apply(func, axis=1)
df
```

[54]:
```
          col_a    col_b col_c col_d                DupA  \
0    Houston,TX  62K-70K     A    1x      Houston,Houston
1     Dallas,TX  62K-70K     B    1y        Dallas,Dallas
2    Chicago,IL  69K-76K     A  2x        Chicago,Chicago
3    Phoenix,AZ  62K-72K     a    1x      Phoenix,Phoenix
4  San Diego,CA  71K-78K     c  1y    San Diego,San Diego

              NewColA1       NewColA2   NewColA3      NewColA4
0      Houston,Houston    Houston,_IN    Houston    Houston_IN
1        Dallas,Dallas     Dallas,_IN     Dallas     Dallas_IN
2      Chicago,Chicago    Chicago,_IN    Chicago    Chicago_IN
3      Phoenix,Phoenix    Phoenix,_IN    Phoenix    Phoenix_IN
4  San Diego,San Diego  San Diego,_IN  San Diego  San Diego_IN
```

[55]:
```
# Tried doing the same thing, but not with '+' operator and passing each row to␣
 ↪.apply() by axis=1

def func(row):
    print(type(row))
    return "-".join([row['col_a'].split(',')[0],'IN'])

df['NewColA6'] = df.apply(func, axis=1)
df
```

```
# It can be clearly seen that each row is passed to the ufunc as a series and␣
 ↪is accessible as String ( str )
# That is why, cat() is not working and have to use .join.
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

[55]:

	col_a	col_b	col_c	col_d	DupA	\
0	Houston,TX	62K-70K	A	1x	Houston,Houston	
1	Dallas,TX	62K-70K	B	1y	Dallas,Dallas	
2	Chicago,IL	69K-76K	A	2x	Chicago,Chicago	
3	Phoenix,AZ	62K-72K	a	1x	Phoenix,Phoenix	
4	San Diego,CA	71K-78K	c	1y	San Diego,San Diego	

	NewColA1	NewColA2	NewColA3	NewColA4	NewColA6
0	Houston,Houston	Houston,_IN	Houston	Houston_IN	Houston-IN
1	Dallas,Dallas	Dallas,_IN	Dallas	Dallas_IN	Dallas-IN
2	Chicago,Chicago	Chicago,_IN	Chicago	Chicago_IN	Chicago-IN
3	Phoenix,Phoenix	Phoenix,_IN	Phoenix	Phoenix_IN	Phoenix-IN
4	San Diego,San Diego	San Diego,_IN	San Diego	San Diego_IN	San Diego-IN

[56]:
```
def func(row):
    return "-".join([row['col_a'].split(',')[0],row['col_d']])

df['NewColA7'] = df.apply(func, axis=1)
df


# It can be clearly seen that each row is passed to the ufunc as a series and␣
 ↪is accessible as String ( str )
# That is why, cat() is not working and have to use .join.
```

[56]:

	col_a	col_b	col_c	col_d	DupA	\
0	Houston,TX	62K-70K	A	1x	Houston,Houston	
1	Dallas,TX	62K-70K	B	1y	Dallas,Dallas	
2	Chicago,IL	69K-76K	A	2x	Chicago,Chicago	
3	Phoenix,AZ	62K-72K	a	1x	Phoenix,Phoenix	
4	San Diego,CA	71K-78K	c	1y	San Diego,San Diego	

	NewColA1	NewColA2	NewColA3	NewColA4	NewColA6	\
0	Houston,Houston	Houston,_IN	Houston	Houston_IN	Houston-IN	
1	Dallas,Dallas	Dallas,_IN	Dallas	Dallas_IN	Dallas-IN	
2	Chicago,Chicago	Chicago,_IN	Chicago	Chicago_IN	Chicago-IN	
3	Phoenix,Phoenix	Phoenix,_IN	Phoenix	Phoenix_IN	Phoenix-IN	

```
4  San Diego,San Diego  San Diego,_IN  San Diego  San Diego_IN  San Diego-IN

         NewColA7
0    Houston-  1x
1      Dallas- 1y
2    Chicago-2x
3      Phoenix-1x
4  San Diego-1y
```

[57]:
```
# Tried doing the same thing, but not with '+' operator and passing each column␣
 ↪to .apply() by axis=0

def func(col):
    print(type(col))
    return "-".join([col.split(',')[0], 'IN'])

df['NewColA8'] = df['col_a'].apply(func)
df


# It can be clearly seen that for each row, one column is passed to the ufunc␣
 ↪as one cell (str) and is accessible as String ( str )
# That is why, cat() is not working and have to use .join.
```

```
<class 'str'>
<class 'str'>
<class 'str'>
<class 'str'>
<class 'str'>
```

[57]:
```
           col_a    col_b col_c col_d                DupA  \
0    Houston,TX  62K-70K     A    1x      Houston,Houston
1     Dallas,TX  62K-70K     B    1y        Dallas,Dallas
2    Chicago,IL  69K-76K     A  2x        Chicago,Chicago
3    Phoenix,AZ  62K-72K     a    1x      Phoenix,Phoenix
4  San Diego,CA  71K-78K     c  1y    San Diego,San Diego

               NewColA1       NewColA2   NewColA3      NewColA4      NewColA6  \
0      Houston,Houston    Houston,_IN    Houston    Houston_IN    Houston-IN
1        Dallas,Dallas     Dallas,_IN     Dallas     Dallas_IN     Dallas-IN
2      Chicago,Chicago    Chicago,_IN    Chicago    Chicago_IN    Chicago-IN
3      Phoenix,Phoenix    Phoenix,_IN    Phoenix    Phoenix_IN    Phoenix-IN
4  San Diego,San Diego  San Diego,_IN  San Diego  San Diego_IN  San Diego-IN

         NewColA7      NewColA8
0    Houston-  1x    Houston-IN
1      Dallas- 1y     Dallas-IN
```

```
2    Chicago-2x      Chicago-IN
3      Phoenix-1x    Phoenix-IN
4  San Diego-1y    San Diego-IN
```

[ ]: