INDUSTRIAL TRAINING DAILY DIARY
DAY 12

08 July, 2025

Topic : String manipulations in Pandas DataFrame

Objectives:

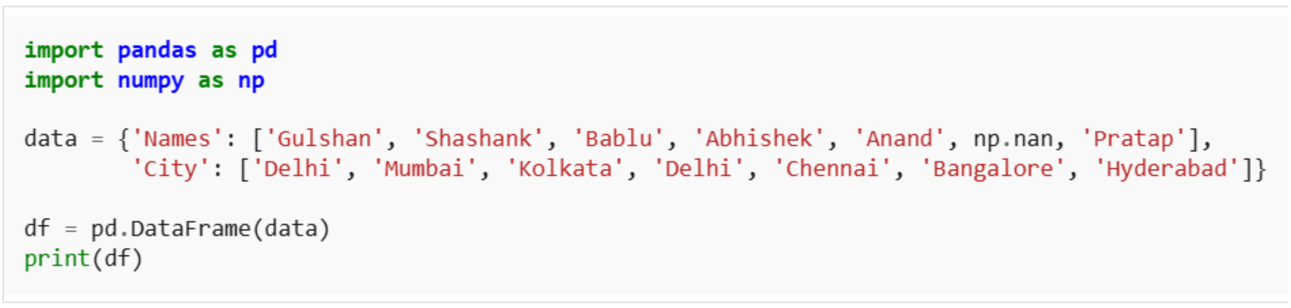
- To understand how to work with string data within a pandas DataFrame.
- To apply string functions using the .str accessor on Series.
- To perform common operations like:
    - Converting to lowercase and uppercase using .str.lower() / .str.upper()
    - Removing whitespace with .str.strip(), .str.lstrip(), and .str.rstrip()
    - Finding substrings using .str.contains() and .str.find()
    - Replacing characters or substrings using .str.replace()
    - Splitting strings with .str.split() and extracting parts using .str.get()
    - Checking for string patterns (like prefixes/suffixes) using .str.startswith() and .str.endswith()
- To clean and preprocess textual data for analysis or machine learning tasks.

String manipulations in Pandas DataFrame

String manipulation is the process of changing, parsing, splicing, pasting or analyzing strings. As we know that sometimes data in the string is not suitable for manipulating the analysis or get a description of the data. But Python is known for its ability to manipulate strings. In this article we will understand how Pandas provides us the ways to manipulate to modify and process string data-frame using some builtin functions.

Create a String Dataframe using Pandas

First of all we will know ways to create a string dataframe using Pandas.

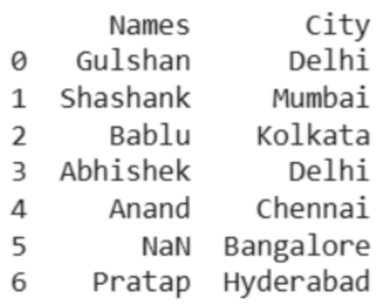


```python
import pandas as pd
import numpy as np

data = {'Names': ['Gulshan', 'Shashank', 'Bablu', 'Abhishek', 'Anand', np.nan, 'Pratap'],
        'City': ['Delhi', 'Mumbai', 'Kolkata', 'Delhi', 'Chennai', 'Bangalore', 'Hyderabad']}

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

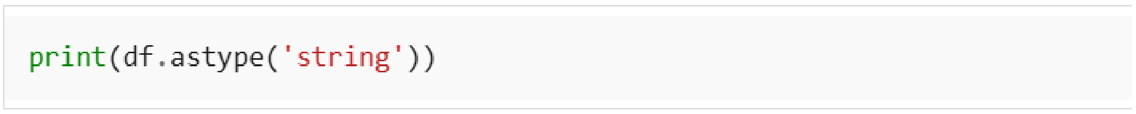
Output:



```
      Names       City
0   Gulshan      Delhi
1  Shashank     Mumbai
2     Bablu    Kolkata
3  Abhishek      Delhi
4     Anand    Chennai
5       NaN  Bangalore
6    Pratap  Hyderabad
```

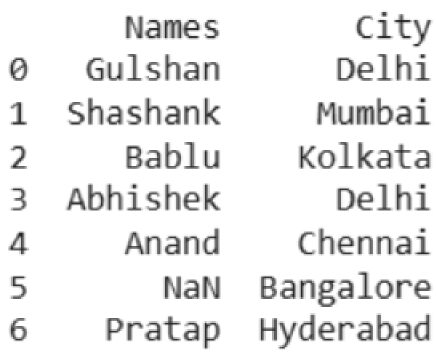
Change Column Datatype in Pandas

To change the type of the created dataframe to string type. we can do this with the help of **.astype()** . Let's have a look at them in the below example



```python
print(df.astype('string'))
```

Output:



```
      Names       City
0   Gulshan      Delhi
1  Shashank     Mumbai
2     Bablu    Kolkata
3  Abhishek      Delhi
4     Anand    Chennai
5       NaN  Bangalore
6    Pratap  Hyderabad
```

String Manipulations in Pandas

Now we see the string manipulations inside a Pandas Dataframe, so first create a Dataframe and manipulate all string operations on this single data frame below so that everyone can get to know about it easily.

Example:

```python
import pandas as pd
import numpy as np

data = {'Names': ['Gulshan', 'Shashank', 'Bablu', 'Abhishek', 'Anand', np.nan, 'Pratap'],
        'City': ['Delhi', 'Mumbai', 'Kolkata', 'Delhi', 'Chennai', 'Bangalore',
'Hyderabad']}

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

Output:

```
      Names       City
0   Gulshan      Delhi
1  Shashank     Mumbai
2     Bablu    Kolkata
3  Abhishek      Delhi
4     Anand    Chennai
5       NaN  Bangalore
6    Pratap  Hyderabad
```

Let's have a look at various methods provided by this library for string manipulations.

•**lower():** Converts all uppercase characters in strings in the DataFrame to lower case and returns the lowercase strings in the result.

```python
print(df['Names'].str.lower())
```

Output:

```
0     gulshan
1    shashank
2       bablu
3    abhishek
4       anand
5         NaN
6      pratap
Name: Names, dtype: object
```

•**upper():** Converts all lowercase characters in strings in the DataFrame to upper case and returns the uppercase strings in result.

```
print(df['Names'].str.upper())
```

Output:

```
0      GULSHAN
1     SHASHANK
2        BABLU
3     ABHISHEK
4        ANAND
5          NaN
6       PRATAP
Name: Names, dtype: object
```

•**strip()**: If there are spaces at the beginning or end of a string, we should trim the strings to eliminate spaces using strip() or remove the extra spaces contained by a string in DataFrame.

```
print(df['Names'].str.strip())
```

Output:

```
0      Gulshan
1     Shashank
2        Bablu
3     Abhishek
4        Anand
5          NaN
6       Pratap
Name: Names, dtype: object
```

•**split(' ')**: Splits each string with the given pattern. Strings are split and the new elements after the performed split operation, are stored in a list.

```
df['Split_Names'] = df['Names'].str.split('a')
print(df[['Names', 'Split_Names']])
```

Output:

```
      Names     Split_Names
0   Gulshan     [Gulsh, n]
1  Shashank  [Sh, sh, nk]
2     Bablu       [B, blu]
3  Abhishek     [Abhishek]
4     Anand       [An, nd]
5       NaN            NaN
6    Pratap     [Pr, t, p]
```

•**len()**: With the help of len() we can compute the length of each string in DataFrame & if there is empty data in DataFrame, it returns NaN.

```
print(df['Names'].str.len())
```

Output:

```
0    7.0
1    8.0
2    5.0
3    8.0
4    5.0
5    NaN
6    6.0
Name: Names, dtype: float64
```

•**cat(sep=' '):** It concatenates the data-frame index elements or each string in DataFrame with given separator.

```
print(df)

print("\nafter using cat:")
print(df['Names'].str.cat(sep=', '))
```

Output:

```
      Names       City     Split_Names
0   Gulshan      Delhi      [Gulsh, n]
1  Shashank     Mumbai    [Sh, sh, nk]
2     Bablu    Kolkata        [B, blu]
3  Abhishek      Delhi      [Abhishek]
4     Anand    Chennai       [An, nd]
5       NaN  Bangalore             NaN
6    Pratap  Hyderabad     [Pr, t, p]

after using cat:
Gulshan, Shashank, Bablu, Abhishek, Anand, Pratap
```

•**get_dummies():** It returns the DataFrame with One-Hot Encoded values like we can see that it returns boolean value 1 if it exists in relative index or 0 if not exists.

```
print(df['City'].str.get_dummies())
```

Output:

```
   Bangalore  Chennai  Delhi  Hyderabad  Kolkata  Mumbai
0          0        0      1          0        0       0
1          0        0      0          0        0       1
2          0        0      0          0        1       0
3          0        0      1          0        0       0
4          0        1      0          0        0       0
5          1        0      0          0        0       0
6          0        0      0          1        0       0
```

•**startswith(pattern):** It returns true if the element or string in the DataFrame Index starts with the pattern.

```
print(df['Names'].str.startswith('G'))
```

Output:

```
0     True
1    False
2    False
3    False
4    False
5      NaN
6    False
Name: Names, dtype: object
```

•**endswith(pattern):** It returns true if the element or string in the DataFrame Index ends with the pattern.

```
print(df['Names'].str.endswith('h'))
```

Output:

```
0    False
1    False
2    False
3    False
4    False
5      NaN
6    False
Name: Names, dtype: object
```

•**Python replace(a,b):** It replaces the value a with the value b like below in example 'Gulshan' is being replaced by 'Gaurav'.

```
print(df['Names'].str.replace('Gulshan', 'Gaurav'))
```

Output:

```
0      Gaurav
1    Shashank
2       Bablu
3    Abhishek
4       Anand
5         NaN
6      Pratap
Name: Names, dtype: object
```

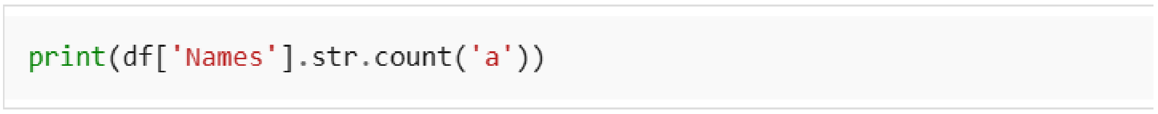
•**Python repeat(value):** It repeats each element with a given number of times like below in example, there are two appearances of each string in DataFrame.

```
print(df['Names'].str.repeat(2))
```

Output:

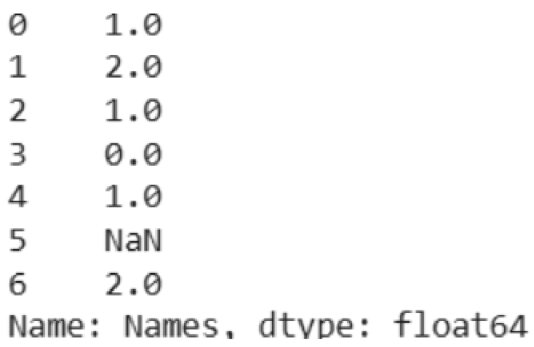
```
0        GulshanGulshan
1      ShashankShashank
2            BabluBablu
3      AbhishekAbhishek
4            AnandAnand
5                   NaN
6          PratapPratap
Name: Names, dtype: object
```

•**Python count(pattern):** 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 'a' in each string.



```
print(df['Names'].str.count('a'))
```

Output:

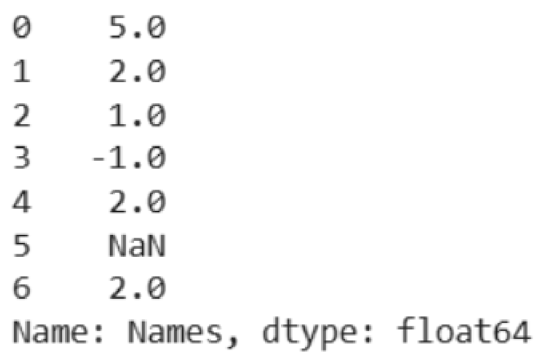


```
0     1.0
1     2.0
2     1.0
3     0.0
4     1.0
5     NaN
6     2.0
Name: Names, dtype: float64
```

•**Python find(pattern):** It returns the first position of the first occurrence of the pattern. We can see in the example below that it returns the index value of appearance of character 'a' in each string throughout the DataFrame.

```
print(df['Names'].str.find('a'))
```

Output:

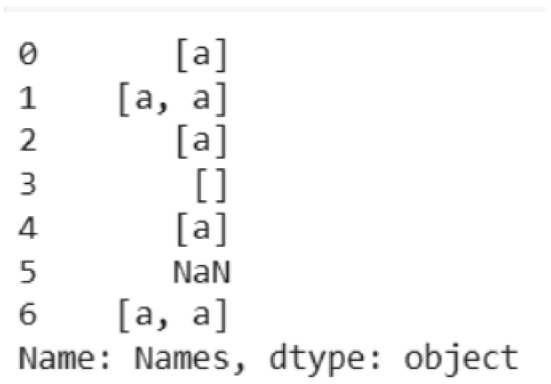


```
0    5.0
1    2.0
2    1.0
3   -1.0
4    2.0
5    NaN
6    2.0
Name: Names, dtype: float64
```

•**findall(pattern):** It returns a list of all occurrences of the pattern. As we can see in below, there is a returned list consisting n as it appears only once in the string.

```
print(df['Names'].str.findall('a'))
```

Output:

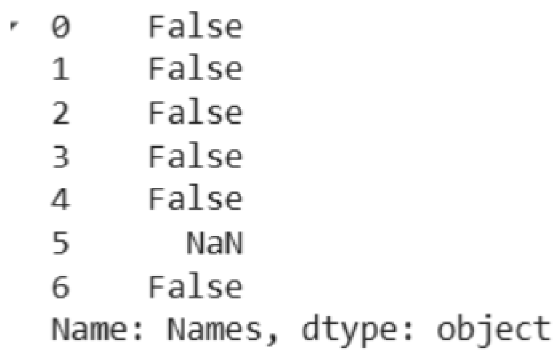


```
0       [a]
1    [a, a]
2       [a]
3        []
4       [a]
5       NaN
6    [a, a]
Name: Names, dtype: object
```

•**islower():** It checks whether all characters in each string in the Index of the Data-Frame in lower case or not, and returns a Boolean value.

```
print(df['Names'].str.islower())
```

Output:

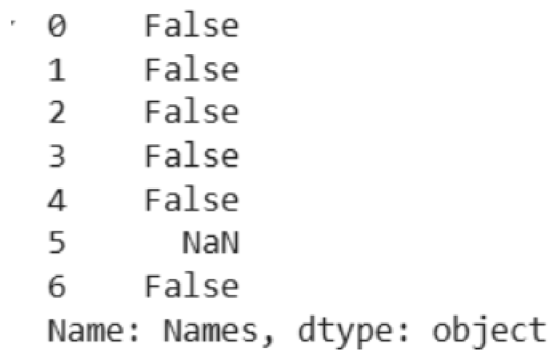


```
0    False
1    False
2    False
3    False
4    False
5      NaN
6    False
Name: Names, dtype: object
```

•**isupper():** It checks whether all characters in each string in the Index of the Data-Frame in upper case or not, and returns a Boolean value.

```
print(df['Names'].str.isupper())
```

Output:



```
0    False
1    False
2    False
3    False
4    False
5      NaN
6    False
Name: Names, dtype: object
```

•**isnumeric():** It checks whether all characters in each string in the Index of the Data-Frame are numeric or not, and returns a Boolean value.

```
print(df['Names'].str.isnumeric())
```

Output:

```
0     False
1     False
2     False
3     False
4     False
5       NaN
6     False
Name: Names, dtype: object
```

•**swapcase():** It swaps the case lower to upper and vice-versa. Like in the example below, it converts all uppercase characters in each string into lowercase and vice-versa (lowercase -> uppercase).

```
print(df['Names'].str.swapcase())
```

Output:

```
0      gULSHAN
1     sHASHANK
2        bABLU
3     aBHISHEK
4        aNAND
5          NaN
6       pRATAP
Name: Names, dtype: object
```

Pandas Exercises

** Import pandas and read in the banklist.csv file into a dataframe called banks. **

```
import pandas as pd

banks =
pd.read_csv(r"https://raw.githubusercontent.com/PrateekKumarSingh/
Python/master/Python%20for%20Finance/Python-for-Finance-Repo-
master/03-%20General%20Pandas/Pandas-Exercises/banklist.csv")
```

** Show the head of the dataframe **

```
banks.head()
```

	Bank Name	City	ST	CERT	Acquiring Institution	Closing Date	Updated Date
0	Fayette County Bank	Saint Elmo	IL	1802	United Fidelity Bank, fsb	26-May-17	1-Jun-17
1	Guaranty Bank, (d/b/a BestBank in Georgia & Mi...	Milwaukee	WI	30003	First-Citizens Bank & Trust Company	5-May-17	1-Jun-17
2	First NBC Bank	New Orleans	LA	58302	Whitney Bank	28-Apr-17	23-May-17
3	Proficio Bank	Cottonwood Heights	UT	35495	Cache Valley Bank	3-Mar-17	18-May-17
4	Seaway Bank and Trust Company	Chicago	IL	19328	State Bank of Texas	27-Jan-17	18-May-17

** What are the column names? **

```
banks.columns
```

```
Index(['Bank Name', 'City', 'ST', 'CERT', 'Acquiring Institution',
       'Closing Date', 'Updated Date'],
      dtype='object')
```

** How many States (ST) are represented in this data set? **

```
print(banks['ST'].nunique())
```

```
44
```

** Get a list or array of all the states in the data set. **

```
print(banks['ST'].unique())
```

```
['IL' 'WI' 'LA' 'UT' 'NJ' 'AR' 'GA' 'PA' 'TN' 'WA' 'CO' 'PR' 'FL' 'MN'
 'CA' 'MD' 'OK' 'OH' 'SC' 'VA' 'ID' 'TX' 'CT' 'AZ' 'NV' 'NC' 'KY' 'MO'
 'KS' 'AL' 'MI' 'IN' 'IA' 'NE' 'MS' 'NM' 'OR' 'NY' 'MA' 'SD' 'WY' 'WV'
 'NH' 'HI']
```

** What are the top 5 states with the most failed banks? **

```
bg = banks.groupby("ST").count()

bg.sort_values('Bank Name',ascending = False).iloc[:5]['Bank Name']
```

```
ST
GA    93
FL    75
IL    67
CA    41
MN    23
Name: Bank Name, dtype: int64
```

** What are the top 5 acquiring institutions? **

```
banks['Acquiring Institution'].value_counts().head(5)
```

```
Acquiring Institution
No Acquirer                            31
State Bank and Trust Company           12
First-Citizens Bank & Trust Company    11
Ameris Bank                            10
U.S. Bank N.A.                          9
Name: count, dtype: int64
```

** How many banks has the State Bank of Texas acquired? How many of them were actually in Texas?**

```
banks[(banks['Acquiring Institution'] == "State Bank of Texas") ]
```

	Bank Name	City	ST	CERT	Acquiring Institution	Closing Date	Updated Date
4	Seaway Bank and Trust Company	Chicago	IL	19328	State Bank of Texas	27-Jan-17	18-May-17
21	The National Republic Bank of Chicago	Chicago	IL	916	State Bank of Texas	24-Oct-14	6-Jan-16
450	Millennium State Bank of Texas	Dallas	TX	57667	State Bank of Texas	2-Jul-09	26-Oct-12

**** What is the most common city in California for a bank to fail in?****

```
# ** What is the most common city in California for a bank to fail in?**

ca_banks = banks[banks['ST']== 'CA']

city_count = ca_banks['City'].value_counts()
print(city_count.idxmax(),city_count.max())
```

```
Los Angeles 4
```

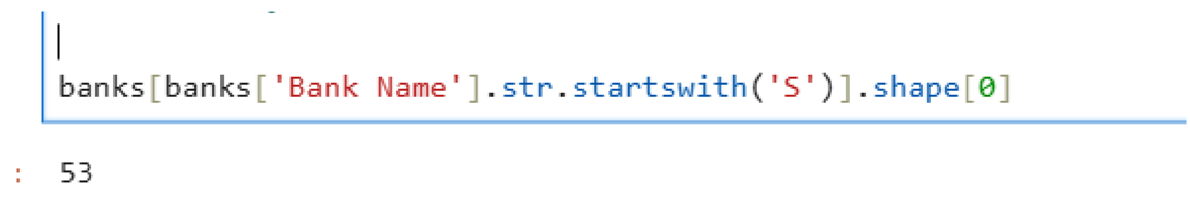
**** How many failed banks don't have the word "Bank" in their name? ****

```
# bank_names = banks['Bank Name']
# count = 0
# for i in bank_names :
#     if 'Bank' not in i  :
#         count+=1
# print(count)

banks[~banks['Bank Name'].str.contains('Bank')].shape[0]
```

```
14
```

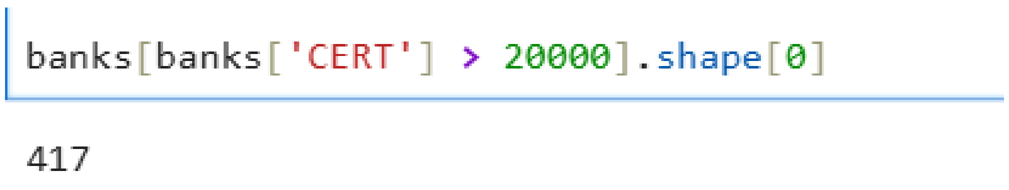
** How many bank names start with the letter 's' ? **



```
banks[banks['Bank Name'].str.startswith('S')].shape[0]
```

```
:   53
```

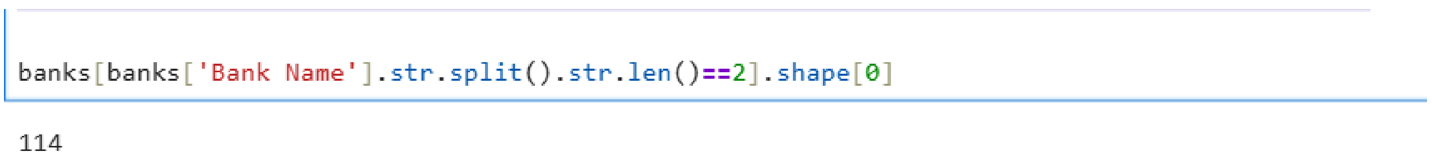
How many CERT values are above 20000 ? **



```
banks[banks['CERT'] > 20000].shape[0]
```

```
417
```

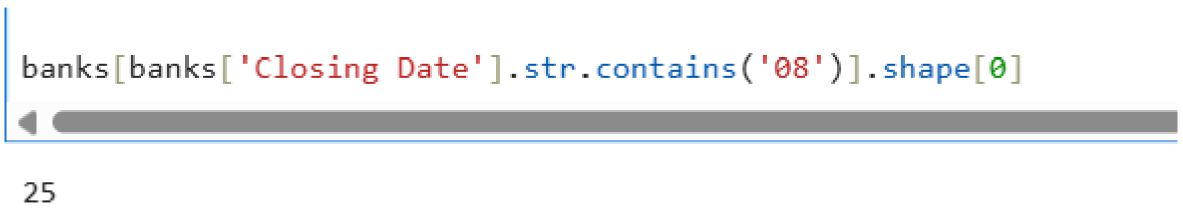
** How many bank names consist of just two words? (e.g. "First Bank" , "Bank Georgia")**



```
banks[banks['Bank Name'].str.split().str.len()==2].shape[0]
```

```
114
```

Bonus: How many banks closed in the year 2008?



```
banks[banks['Closing Date'].str.contains('08')].shape[0]
```

```
25
```