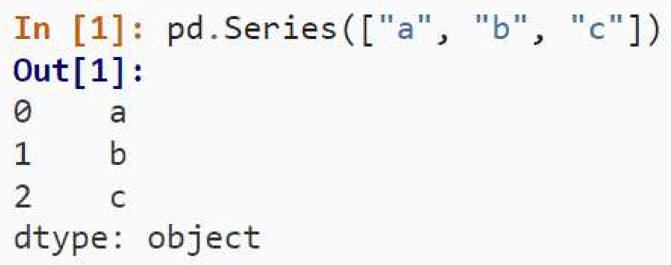
Pandas – strings

There are two ways to store text data:

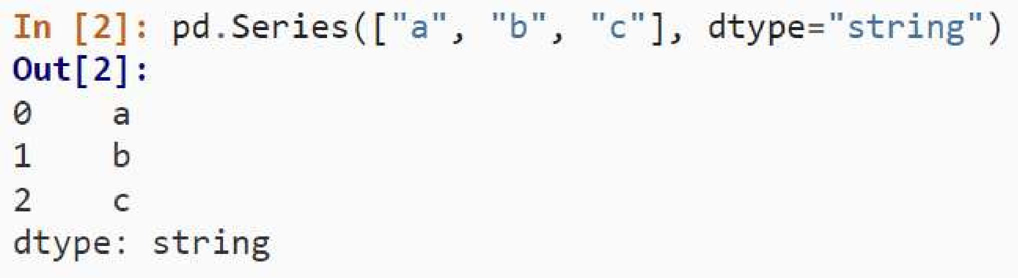
1) object dtype NumPy array
2) StringDtype extension type

object dtype is the default type (for backwards-compatibility)



```
In [1]: pd.Series(["a", "b", "c"])
Out[1]:
0    a
1    b
2    c
dtype: object
```

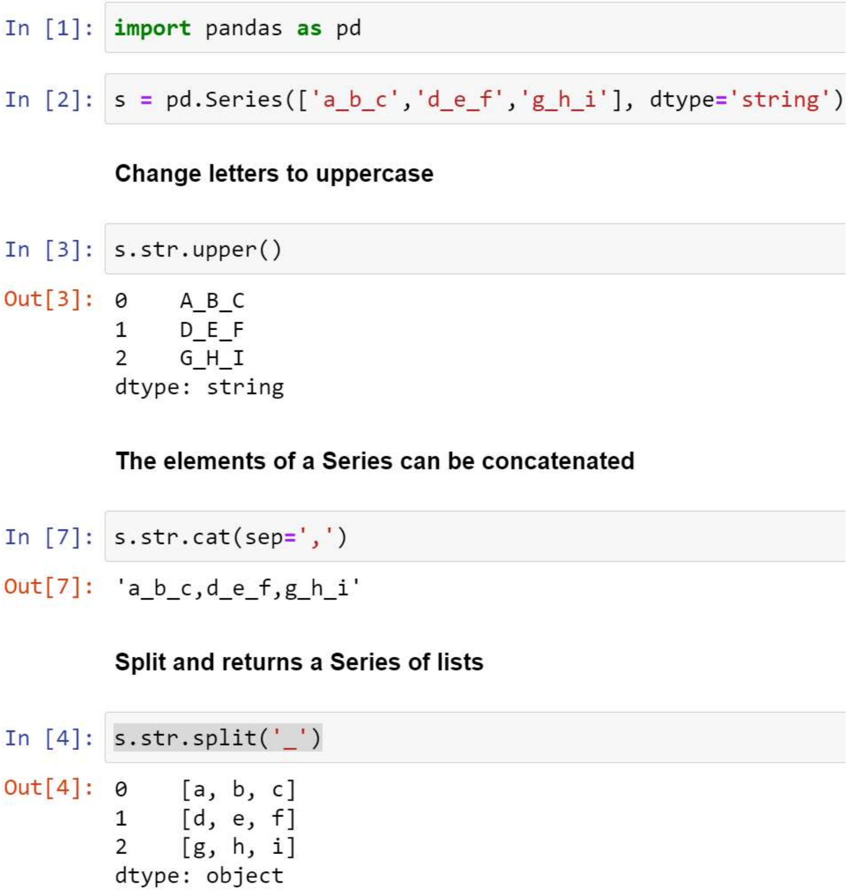
to use type string is necessary to request it explicitly



```
In [2]: pd.Series(["a", "b", "c"], dtype="string")
Out[2]:
0    a
1    b
2    c
dtype: string
```

Pandas – strings methods

Series and Index are equipped with a set of string processing methods that make it easy to operate on each element of the array. These methods exclude missing/NA values automatically. These are accessed via the **str** attribute and generally have names matching the equivalent (scalar) built-in string methods



```
In [1]: import pandas as pd

In [2]: s = pd.Series(['a_b_c','d_e_f','g_h_i'], dtype='string')
```

Change letters to uppercase

```
In [3]: s.str.upper()

Out[3]: 0    A_B_C
        1    D_E_F
        2    G_H_I
        dtype: string
```

The elements of a Series can be concatenated

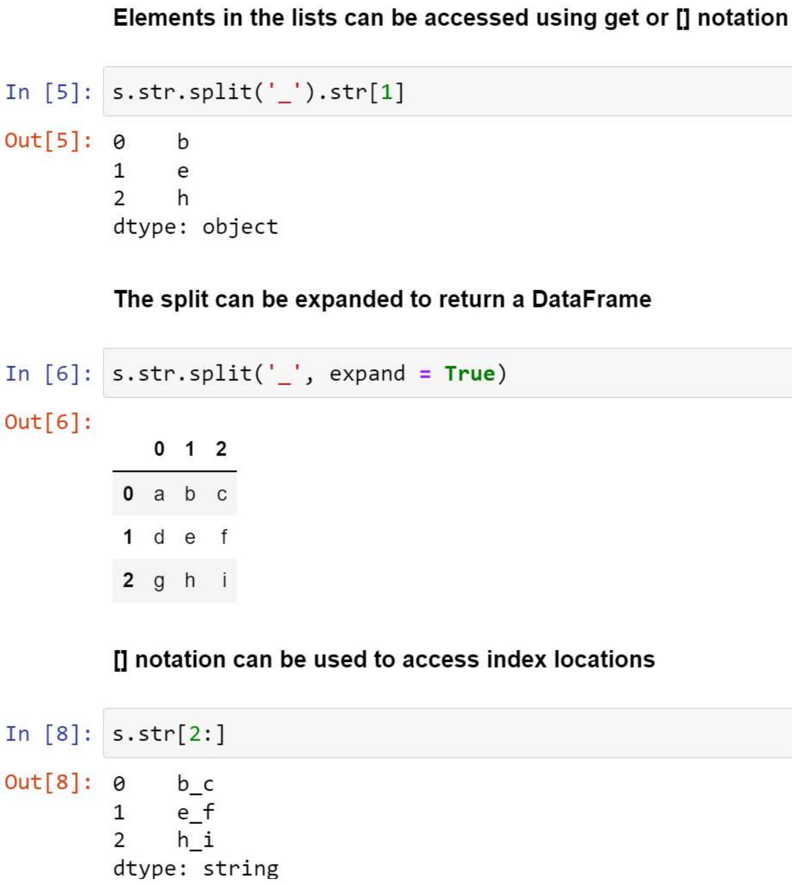
```
In [7]: s.str.cat(sep=',')

Out[7]: 'a_b_c,d_e_f,g_h_i'
```

Split and returns a Series of lists

```
In [4]: s.str.split('_')

Out[4]: 0    [a, b, c]
        1    [d, e, f]
        2    [g, h, i]
        dtype: object
```



Elements in the lists can be accessed using get or [] notation

```
In [5]: s.str.split('_').str[1]

Out[5]: 0    b
        1    e
        2    h
        dtype: object
```

The split can be expanded to return a DataFrame

```
In [6]: s.str.split('_', expand = True)

Out[6]:
```

	0	1	2
0	a	b	c
1	d	e	f
2	g	h	i

[] notation can be used to access index locations

```
In [8]: s.str[2:]

Out[8]: 0    b_c
        1    e_f
        2    h_i
        dtype: string
```

Pandas – str methods

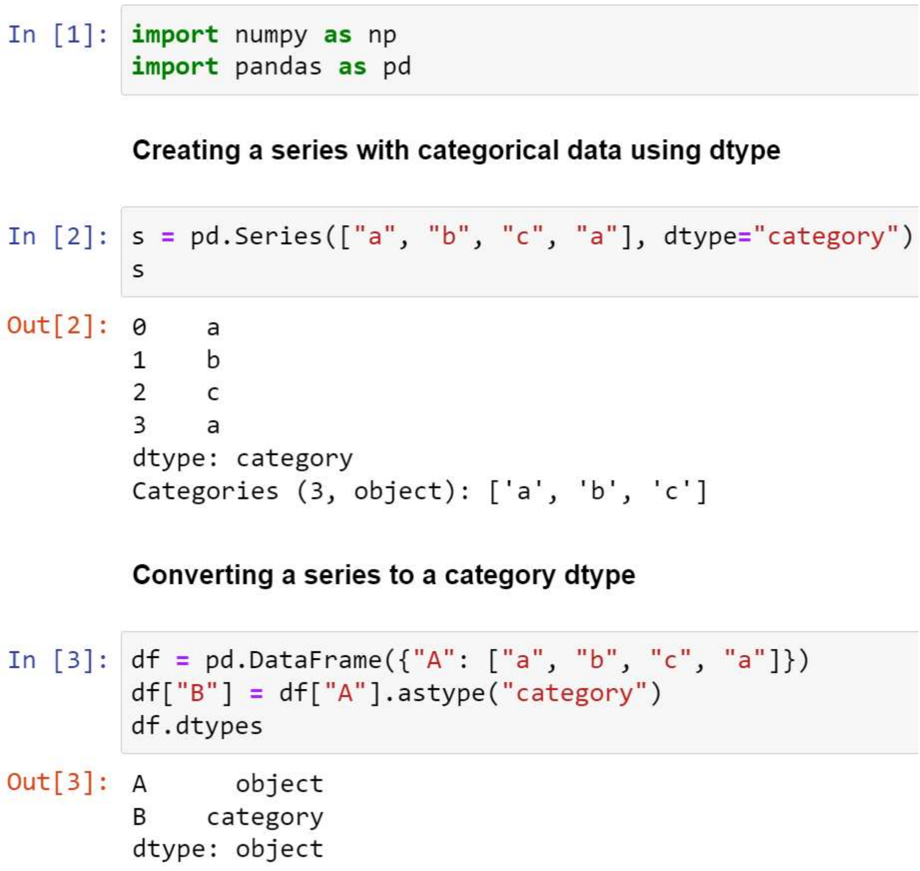
Method	Description
cat()	Concatenate strings
split()	Split strings on delimiter
rsplit()	Split strings on delimiter working from the end of the string
get()	Index into each element (retrieve i-th element)
join()	Join strings in each element of the Series with passed separator
get_dummies()	Split strings on the delimiter returning DataFrame of dummy variables
contains()	Return boolean array if each string contains pattern/regex
replace()	Replace occurrences of pattern/regex/string with some other string or the return value of a callable given the occurrence
repeat()	Duplicate values (s.str.repeat(3) equivalent to x * 3)
pad()	Add whitespace to left, right, or both sides of strings
center()	Equivalent to str.center
ljust()	Equivalent to str.ljust
rjust()	Equivalent to str.rjust
zfill()	Equivalent to str.zfill
wrap()	Split long strings into lines with length less than a given width
slice()	Slice each string in the Series
slice_replace()	Replace slice in each string with passed value
count()	Count occurrences of pattern
startswith()	Equivalent to str.startswith(pat) for each element
endswith()	Equivalent to str.endswith(pat) for each element
findall()	Compute list of all occurrences of pattern/regex for each string
match()	Call re.match on each element, returning matched groups as list
extract()	Call re.search on each element, returning DataFrame with one row for each element and one column for each regex capture group
extractall()	Call re.findall on each element, returning DataFrame with one row for each match and one column for each regex capture group
len()	Compute string lengths

Pandas – str methods

Method	Description
strip()	Equivalent to str.strip
rstrip()	Equivalent to str.rstrip
lstrip()	Equivalent to str.lstrip
partition()	Equivalent to str.partition
rpartition()	Equivalent to str.rpartition
lower()	Equivalent to str.lower
casefold()	Equivalent to str.casefold
upper()	Equivalent to str.upper
find()	Equivalent to str.find
rfind()	Equivalent to str.rfind
index()	Equivalent to str.index
rindex()	Equivalent to str.rindex
capitalize()	Equivalent to str.capitalize
swapcase()	Equivalent to str.swapcase
normalize()	Return Unicode normal form. Equivalent to unicodedata.normalize
translate()	Equivalent to str.translate
isalnum()	Equivalent to str.isalnum
isalpha()	Equivalent to str.isalpha
isdigit()	Equivalent to str.isdigit
isspace()	Equivalent to str.isspace
islower()	Equivalent to str.islower
isupper()	Equivalent to str.isupper
istitle()	Equivalent to str.istitle
isnumeric()	Equivalent to str.isnumeric
isdecimal()	Equivalent to str.isdecimal

Pandas – Categorical data

Categoricals are a pandas data type corresponding to categorical variables in statistics. A categorical variable takes on a limited, and usually fixed, number of possible values. Examples are gender, social class, blood type, country affiliation, observation time or rating via Likert scales



```
In [1]: import numpy as np
        import pandas as pd
```

Creating a series with categorical data using dtype

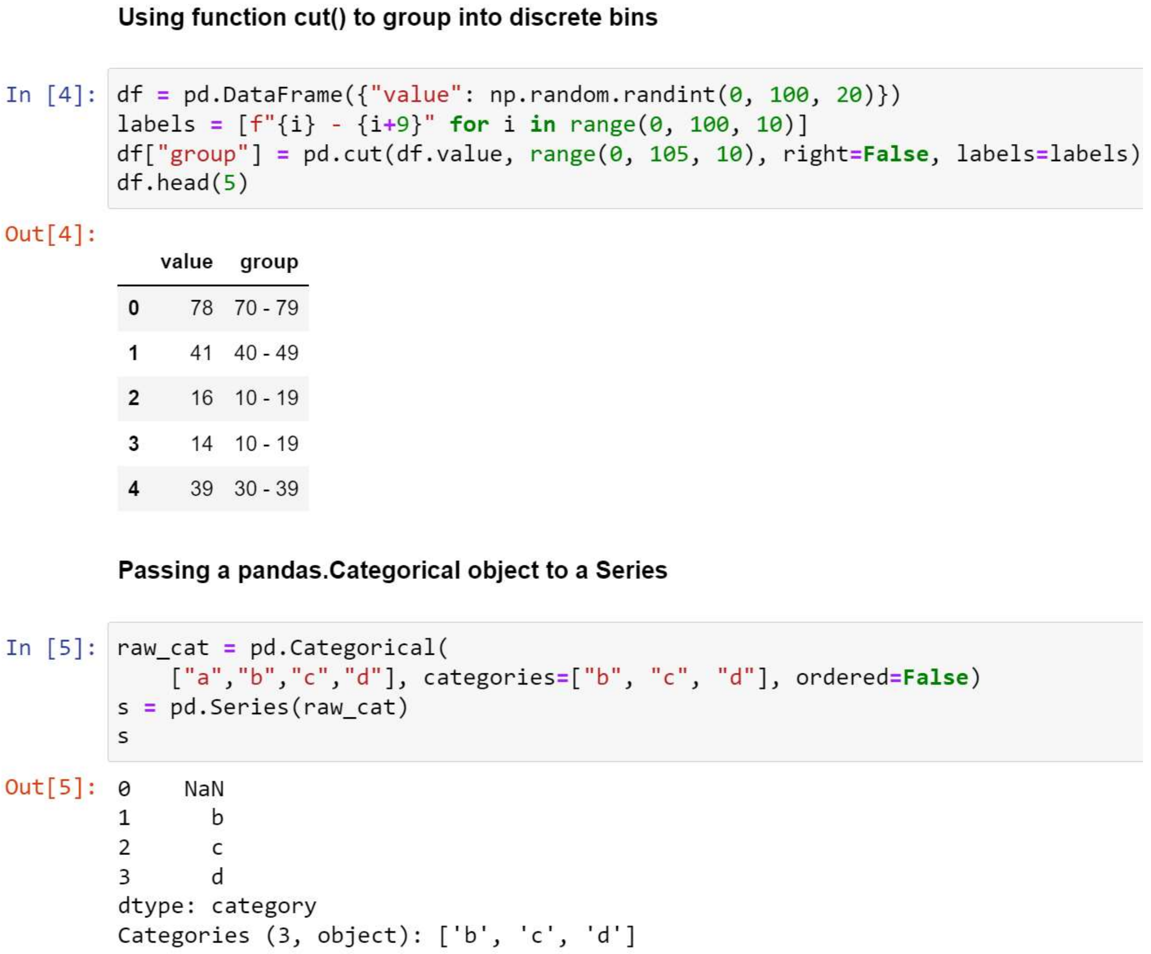
```
In [2]: s = pd.Series(["a", "b", "c", "a"], dtype="category")
        s

Out[2]: 0    a
        1    b
        2    c
        3    a
        dtype: category
        Categories (3, object): ['a', 'b', 'c']
```

Converting a series to a category dtype

```
In [3]: df = pd.DataFrame({"A": ["a", "b", "c", "a"]})
        df["B"] = df["A"].astype("category")
        df.dtypes

Out[3]: A      object
        B    category
        dtype: object
```



Using function cut() to group into discrete bins

```
In [4]: df = pd.DataFrame({"value": np.random.randint(0, 100, 20)})
        labels = [f"{i} - {i+9}" for i in range(0, 100, 10)]
        df["group"] = pd.cut(df.value, range(0, 105, 10), right=False, labels=labels)
        df.head(5)

Out[4]:
```

	value	group
0	78	70 - 79
1	41	40 - 49
2	16	10 - 19
3	14	10 - 19
4	39	30 - 39

Passing a pandas.Categorical object to a Series

```
In [5]: raw_cat = pd.Categorical(
            ["a","b","c","d"], categories=["b", "c", "d"], ordered=False)
        s = pd.Series(raw_cat)
        s

Out[5]: 0    NaN
        1      b
        2      c
        3      d
        dtype: category
        Categories (3, object): ['b', 'c', 'd']
```