Data Analysis with Python

Session-10

CLARUSWAY©
WAY TO REINVENT YOURSELF

pandas

Working with Text & Date Data

CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents

- Working with Text Data
  - Text Data Types
  - String Methods
  - Dummy Operation
- Working with Time Data
  - pd.to_datetime()
  - Series.dt()
  - Datetime Module
  - Strftime & Strptime

I've completed the **pre-class** content?

Students choose an option

Pear Deck Interactive Slide
Do not remove this bar

No Multiple Choice Response
You didn't answer this question

Working with Text Data

Text Data Types

There are two ways to store text data in pandas:

1. `object` -dtype NumPy array.
2. **`StringDtype`** extension type.

We recommend using **`StringDtype`** to store text data.

Prior to pandas 1.0, `object` dtype was the only option. This was unfortunate for many reasons:

1. You can accidentally store a *mixture* of strings and non-strings in an `object` dtype array. It's better to have a dedicated dtype.
2. `object` dtype breaks dtype-specific operations like **`DataFrame.select_dtypes()`**. There isn't a clear way to select *just* text while excluding non-text but still object-dtype columns.
3. When reading code, the contents of an `object` dtype array is less clear than `'string'`.

Working with Text Data

String Methods

Python Built-in String Methods

Pandas String 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.
- Perhaps most importantly, 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**.

Working with Text Data

String Methods

Python Built-in String Methods

Pandas String Methods

```
>>>dir(str)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__',
'__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '_formatter_field_name_split', '_formatter_parser',
'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs',
'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace',
'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

Working with Text Data

String Methods

`pandas.Series.str.string_method`

```
df["Column_name"].str.split()

df["Column_name"].str.split().str[0]

df["Column_name"].str.split().str[0].str.replace("-", "&")
```

Working with Text Data

String Methods

isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case

Returns boolean

Working with Text Data

String Methods

- endswith() => Returns true if the string ends with the specified value
- startswith() => Returns true if the string starts with the specified value
- contains() => Returns a Boolean value True for each element if the substring contains in the element, else False.

Returns boolean

Working with Text Data

String Methods

- lower() => Converts a string into lower case
- upper() => Converts a string into upper case
- capitalize() => Converts the first character to upper case
- title() => Converts the first character of each word to upper case
- swapcase() => Swaps the case lower/upper

Changing upper/lower case

Working with Text Data

String Methods

- replace()=> Returns a string where a specified value is replaced with a specified value
- split() => Splits the string at the specified separator, and returns a list
- strip() => Returns a trimmed version of the string
- find() => Searches the string for a specified value and returns the position of where it was found
- findall() => Returns a list of all occurrence of the pattern.
- join() => Converts the elements of an iterable into a string

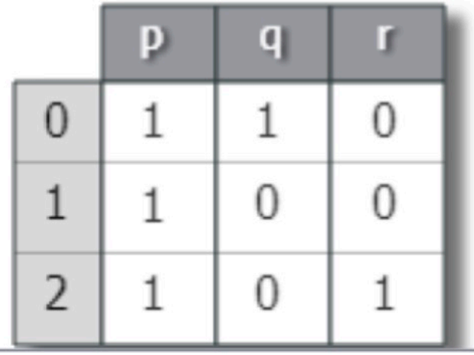
Working with Text Data

Dummy Operation

```
Series.str.get_dummies(sep = "|")
```

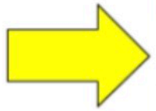
```
pd.get_dummies(array_like)
pd.get_dummies(Series)
pd.get_dummies(DataFrame)
```

pd.Series(['p|q', 'p', 'p|r']).str.get_dummies()



	p	q	r
0	1	1	0
1	1	0	0
2	1	0	1

Color
Red
Red
Yellow
Green
Yellow



Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

Working with Time Data

pandas.to_datetime

- Convert argument to datetime.

```
pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None,
exact=True, unit=None, infer_datetime_format=False, origin='unix', cache=True)
```

arg : The object to convert to a datetime.

(int, float, str, datetime, list, tuple, 1-d array, Series, DataFrame/dict-like)

Returns datetime

If parsing succeeded. Return type depends on input:

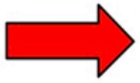
- list-like: DatetimeIndex
- Series: Series of datetime64 dtype
- scalar: Timestamp

Working with Time Data

pandas.Series.dt....

- Accessor object for datetimelike properties of the Series values.

```
 8 date['year']   = date['date'].dt.year
 9 date['month']  = date['date'].dt.month
10 date['day']    = date['date'].dt.day
11 date['hour']   = date['date'].dt.hour
12 date['minute'] = date['date'].dt.minute
```



	date	year	month	day	hour	minute
0	2020-10-28 00:00:00	2020	10	28	0	0
1	2020-10-28 01:00:00	2020	10	28	1	0
2	2020-10-28 02:00:00	2020	10	28	2	0
3	2020-10-28 03:00:00	2020	10	28	3	0
4	2020-10-28 04:00:00	2020	10	28	4	0

Working with Time Data

Datetime Module

- *class* datetime.**date**
- *class* datetime.**time**
- ★ *class* datetime.**datetime**
- ★ *class* datetime.**timedelta**
- *class* datetime.**tzinfo**
- *class* datetime.**timezone**

A combination of a date and a time. Attributes: year, month, day, hour, minute, second, microsecond, and tzinfo.

```
print(datetime.now())
```
2022-01-09 16:47:30.542100

A duration expressing the difference between two date, time, or datetime instances to microsecond resolution.

```
print(timedelta(weeks=2, hours=4))
```
14 days, 4:00:00

Working with Time Data

strftime() & strptime()

	strftime	strptime
Usage	Convert object to a string according to a given format	Parse a string into a `datetime` object given a corresponding format
Type of method	Instance method	Class method
Method of	`date`, `datetime`, `time`	`datetime`
Signature	`strftime(format)`	`strptime(date_string, format)`

```
print(current_datetime)
```

```
2022-01-09 16:46:41.117294
```

```
print(current_datetime.strftime("%m/%d/%Y, %H:%M:%S"))
```

```
01/09/2022, 16:46:41
```

```
date_string = "21 June, 2018"
date_string
```

```
'21 June, 2018'
```

```
datetime_object = datetime.strptime(date_string, "%d %B, %Y")
datetime_object
```

```
datetime.datetime(2018, 6, 21, 0, 0)
```

Data Analysis with Python

let's start the hands-on phase

No Draggable™ Response

You didn't answer this question