pandas

COGS 108 Discussion 4

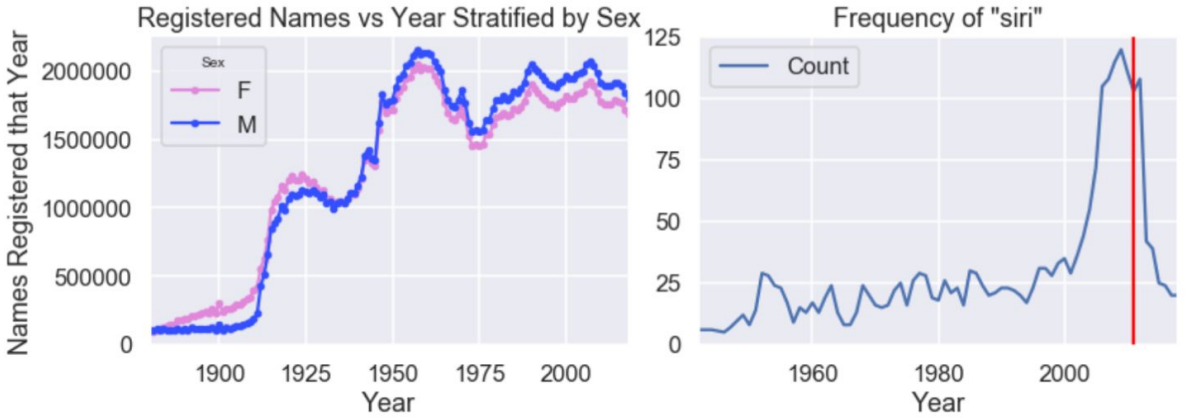
Slides adapted from Atman Patel COGS 108 Fall 2020

Pandas is really useful!

It converts python into a usable (and good!) data analysis tool.

	Name	Sex	Count	Year
0	Mary	F	7065	1880
1	Anna	F	2604	1880
2	Emma	F	2003	1880
...	...	...	...	...
1957043	Zyrie	M	5	2018
1957044	Zyron	M	5	2018
1957045	Zzyzx	M	5	2018

1957046 rows × 4 columns



Pandas has terrible error messages.

	Timestamp	Name	Sex	Age
0	10/15/2019 21:49:38	samuel	M	24
1	10/16/2019 9:07:31	aditi	F	22
2	10/16/2019 9:07:34	hanyang	M	21
...	...	...	...	...
24	10/16/2019 16:08:45	amy	F	20
25	10/16/2019 16:08:46	sheila	F	21
26	10/16/2019 16:09:15	thomas	M	23

```
students['name']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~/anaconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2656             try:
-> 2657                 return self._engine.get_loc(key)
   2658             except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'name'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-27-ae454297f350> in <module>()
----> 1 students['name']

~/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py in __getitem__(self, key)
   2925             if self.columns.nlevels > 1:
   2926                 return self._getitem_multilevel(key)
-> 2927             indexer = self.columns.get_loc(key)
   2928             if is_integer(indexer):
   2929                 indexer = [indexer]

~/anaconda3/lib/python3.7/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2657                 return self._engine.get_loc(key)
   2658             except KeyError:
-> 2659                 return self._engine.get_loc(self._maybe_cast_indexer(key))
   2660         indexer = self.get_indexer([key], method=method, tolerance=tolerance)
   2661         if indexer.ndim > 1 or indexer.size > 1:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'name'
```

Pandas has unfriendly documentation.

There are also typically many ways to do the same thing in pandas.

`DataFrame.`**`rename`**(*self, mapper=None, index=None, columns=None, axis=None, copy=True, inplace=False, level=None, errors='ignore'*) [source]

Alter axes labels.

Function / dict values must be unique (1-to-1). Labels not contained in a dict / Series will be left as-is. Extra labels listed don't throw an error.

See the user guide for more.

Parameters:

mapper : *dict-like or function*
Dict-like or functions transformations to apply to that axis' values. Use either `mapper` and `axis` to specify the axis to target with `mapper`, or `index` and `columns`.

index : *dict-like or function*
Alternative to specifying axis (`mapper, axis=0` is equivalent to `index=mapper`).

columns : *dict-like or function*
Alternative to specifying axis (`mapper, axis=1` is equivalent to `columns=mapper`).

axis : *int or str*
Axis to target with `mapper`. Can be either the axis name ('index', 'columns') or number (0, 1). The default is 'index'.

copy : *bool, default True*
Also copy underlying data.

inplace : *bool, default False*
Whether to return a new DataFrame. If True then value of copy is ignored.

level : *int or level name, default None*
In case of a MultiIndex, only rename labels in the specified level.

errors : *{'ignore', 'raise'}, default 'ignore'*

3 skills that’ll save you a bunch of time:

- Knowing the difference between a pandas Series and Data Frame.
- Knowing how to use Google effectively.
- Knowing how to read the pandas documentation.

What's a **Data Frame**?

```
dogs
```

	breed	type	size	weight
0	German Shepherd	herding	large	70.0
1	Beagle	hound	small	5.2
2	Yorkshire Terrier	toy	small	5.5
3	Golden Retriever	sporting	medium	60.0
4	Bulldog	non-sporting	medium	45.0
5	Labrador Retriever	sporting	medium	67.5
6	Boxer	working	medium	42.0
7	Pomeranian	toy	small	5.0

2D table of data.

Every row and every column has a label.

What's a **Data Frame**?

```
dogs
```

	breed	type	size	weight
0	German Shepherd	herding	large	70.0
1	Beagle	hound	small	5.2
2	Yorkshire Terrier	toy	small	5.5
3	Golden Retriever	sporting	medium	60.0
4	Bulldog	non-sporting	medium	45.0
5	Labrador Retriever	sporting	medium	67.5
6	Boxer	working	medium	42.0
7	Pomeranian	toy	small	5.0

2D table of data.

Every row and every column has a label.

We call the set of row labels the **Index** of a DataFrame.

What's a **Series**?

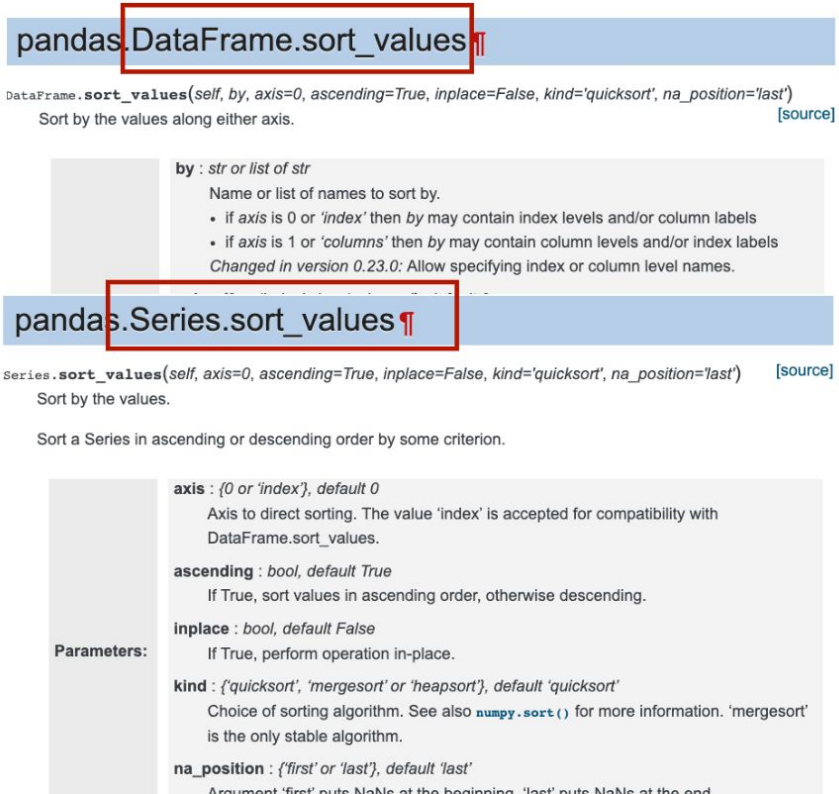
```
dogs['breed']
```

```
0       German Shepherd
1                Beagle
2     Yorkshire Terrier
3      Golden Retriever
4                Bulldog
5    Labrador Retriever
6                 Boxer
7             Pomeranian
Name: breed, dtype: object
```

1D sequence of data.

Usually created by taking a column from a Data Frame.

Why is this important?



pandas.DataFrame.sort_values¶

DataFrame.**sort_values**(*self*, *by*, *axis=0*, *ascending=True*, *inplace=False*, *kind='quicksort'*, *na_position='last'*)

Sort by the values along either axis. [source]

by : *str or list of str*
Name or list of names to sort by.
- if *axis* is 0 or *'index'* then *by* may contain index levels and/or column labels
- if *axis* is 1 or *'columns'* then *by* may contain column levels and/or index labels

Changed in version 0.23.0: Allow specifying index or column level names.

pandas.Series.sort_values¶

Series.**sort_values**(*self*, *axis=0*, *ascending=True*, *inplace=False*, *kind='quicksort'*, *na_position='last'*) [source]

Sort by the values.

Sort a Series in ascending or descending order by some criterion.

Parameters:

axis : *{0 or 'index'}, default 0*
Axis to direct sorting. The value 'index' is accepted for compatibility with DataFrame.sort_values.

ascending : *bool, default True*
If True, sort values in ascending order, otherwise descending.

inplace : *bool, default False*
If True, perform operation in-place.

kind : *{'quicksort', 'mergesort' or 'heapsort'}, default 'quicksort'*
Choice of sorting algorithm. See also `numpy.sort()` for more information. 'mergesort' is the only stable algorithm.

na_position : *{'first' or 'last'}, default 'last'*

Most pandas methods work differently between Data Frames and Series.

The documentation will tell you what type of object the method is for.

Why is this important?

pandas.DataFrame.sort_values¶

DataFrame.**sort_values**(*self, by, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last'*)

Sort by the values along either axis. [source]

by : *str or list of str*

Name or list of names to sort by.

- if *axis* is 0 or *'index'* then *by* may contain index levels and/or column labels
- if *axis* is 1 or *'columns'* then *by* may contain column levels and/or index labels

Changed in version 0.23.0: Allow specifying index or column level names.

pandas.Series.sort_values¶

Series.**sort_values**(*self, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last'*) [source]

Sort by the values.

Sort a Series in ascending or descending order by some criterion.

pandas.read_csv

pandas.**read_csv**(*filepath_or_buffer, sep=', ', delimiter=None, header='infer', names=None, index_col=None, usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, iterator=False, chunksize=None, compression='infer', thousands=None, decimal=b'.', lineterminator=None, quotechar='"', quoting=0, escapechar=None, comment=None, encoding=None, dialect=None, tupleize_cols=None, error_bad_lines=True, warn_bad_lines=True, skipfooter=0, doublequote=True, delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None*) [source]

Read CSV (comma-separated) file into DataFrame

Also supports optionally iterating or breaking of the file into chunks.

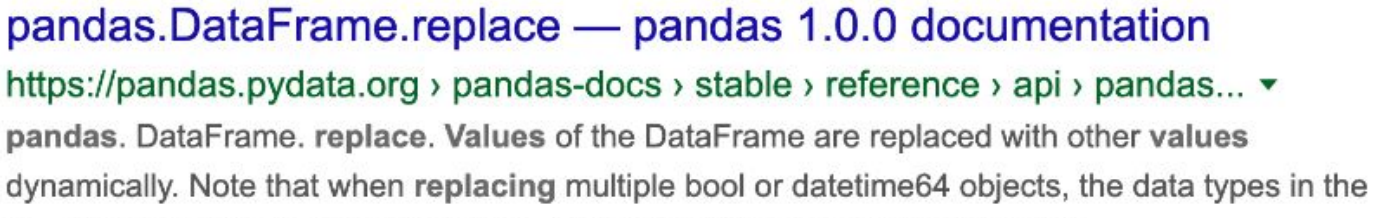
→ `df.sort_values(...)`

→ `df['names'].sort_values(...)`

→ `pd.read_csv(...)`

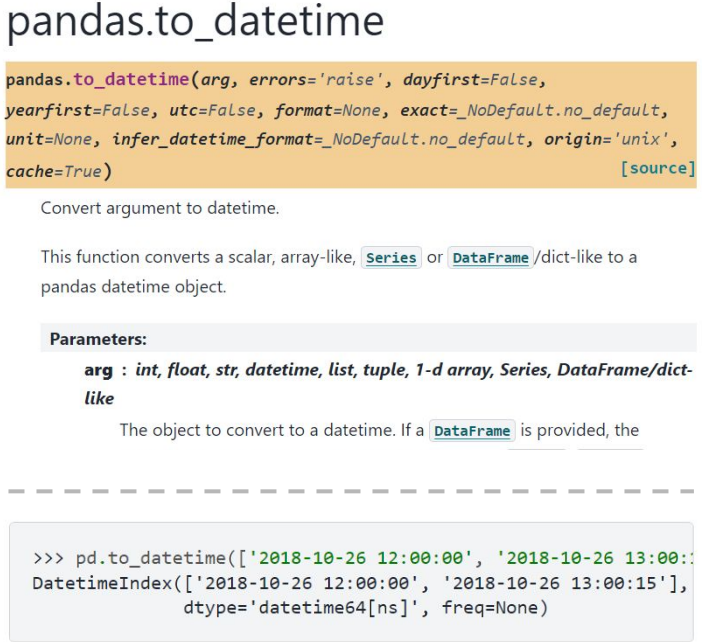
How to use Google

1. **State your task:** “I need to replace 0 with False and 1 with True”
2. **Remove question-specific details:** “replace values”
3. **Add the package name to the front:** “pandas replace values”
4. **If you already know the right method, just search it:** “pandas replace”
5. **Cheatsheets can help you find the right method.**



How to read pandas documentation

1. Skip the table of method parameters and look at the examples.
2. From here, you can even copy the example, then modify it to work for your notebook.
3. If needed, refer back to the method parameters for additional options.



pandas.to_datetime

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

Convert argument to datetime.

This function converts a scalar, array-like, `Series` or `DataFrame`/dict-like to a pandas datetime object.

Parameters:

arg : *int, float, str, datetime, list, tuple, 1-d array, Series, DataFrame/dict-like*

The object to convert to a datetime. If a `DataFrame` is provided, the

```
>>> pd.to_datetime(['2018-10-26 12:00:00', '2018-10-26 13:00:
DatetimeIndex(['2018-10-26 12:00:00', '2018-10-26 13:00:15'],
              dtype='datetime64[ns]', freq=None)
```

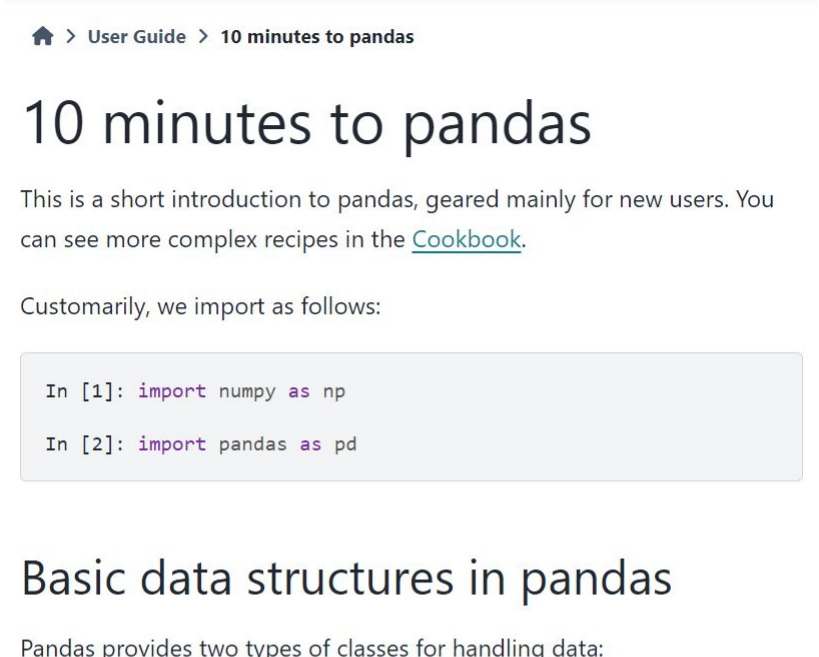
Finally: Avoid loops.

As a general rule of thumb, if you find yourself trying to write a for/while loop when working with pandas, you're almost definitely doing it wrong.

Look for the right pandas method. And ask your friend + staff for help.

Additional resource:

pandas.pydata.org/docs/user_guide/10min.html



User Guide > 10 minutes to pandas

10 minutes to pandas

This is a short introduction to pandas, geared mainly for new users. You can see more complex recipes in the Cookbook.

Customarily, we import as follows:

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

In [2]: import pandas as pd
```

Basic data structures in pandas

Pandas provides two types of classes for handling data: