Microsoft: DAT210x Programming with Python for Data Science

Bookmark

Loading Data

MOD6

0:00 / 4:15 1.0x

Start of transcript. Skip to the end.

From here on out, you're going to be using Pandas,

a high performance data analysis library for Python.

And it's what you're going to be using to do all of your inputs,

your data output, your data manipulation,

everything is gonna be taken care of with Pandas.

Download video Download transcript .srt

Once you collected your data, the next step is learning how to learn how to manipulate it efficiently. Knowing how to do some basic operations, such as slicing your dataset by conditionals, aggregating entries, and searching for values properly, will save you a lot of time when you have to look over hundreds or thousands of records. *Pandas* is one of the most vital and actively developed high-performance data analysis libraries for Python, and you'll be using it for all your data input, output, and manipulation needs. If you're already familiar with the library NumPy, Pandas will feel right at home since it's built on top of it. To get started with Pandas, import it:

```
import pandas as pd
```

There are two data structures in Pandas you need to know how to work with. The first is the series object, a one-dimensional labeled array that represents a single column in your dataset. Which of the following two, essentially equal series would you rather work with?

House Size (free form)	**House Size (sq. feet)**
Large	2950
900 sqft	900
100m^2	1076
small	499
NaN	734

1831	1831
167 square meters	1798
500	500

Clearly, the second series will be easier for you to analyze and manipulate. Since all the elements are of the same data type and the same unit, it makes sense that you would have the ability to do series-wide operations. Because of this, Pandas series must be homogenous. They're capable of storing any Python data type (integers, strings, floating point numbers, objects, etc.), but all the elements in a series **must** be of the same data type.

The second structure you need to work with is a collection of series called a dataframe. To manipulate a dataset, you first need to load it into a dataframe. Different people prefer alternative methods of storing their data, so Pandas tries to make loading data easy no matter how its stored:

```
from sqlalchemy import create_engine
engine = create_engine('sqlite:///:memory:')

sql_dataframe  = pd.read_sql_table('my_table', engine, columns=['ColA', 'ColB'])
xls_dataframe  = pd.read_excel('my_dataset.xlsx', 'Sheet1', na_values=['NA', '?'])
json_dataframe = pd.read_json('my_dataset.json', orient='columns')
csv_dataframe  = pd.read_csv('my_dataset.csv', sep=',')
table_dataframe= pd.read_html('http://page.com/with/table.html')[0]
```

Writing an existing dataframe to disk is equally as straightforward:

```
my_dataframe.to_sql('table', engine)
my_dataframe.to_excel('dataset.xlsx')
my_dataframe.to_json('dataset.json')
my_dataframe.to_csv('dataset.csv')
```

Except in certain cases, like a database table where the columns are clearly defined, the first row in your data will be used as the column headers. Therefore if your data starts from the first line and you don't actually have a header row, ensure you pass in the `names` parameter (a list of column header names) when you call the `.read_*()` method. Pandas will use the provided headers in place of your first data entry.

If you do have column titles already defined in your dataset but wish to rename them, in that case, use the `.columns` property:

```
my_dataframe.columns = ['new', 'column', 'header', 'labels']
```

There are many additional optional parameters you can tinker with, most existing on both read and write operations. Be sure to check out the Pandas API Reference to see how to make Pandas' powerful I/O work for you. With a dataset loaded into your dataframe, you can now use various methods to examine it.

A Quick Peek

To get a quick peek at your data, select its top or bottom few rows using `.head()` and `.tail()`:

```
>>> df.head(5)
id    name   age   location
```

```
0    David    10    at_home
1      HAL  3000    at_home
2  Mustafa    35      other
3     Adam    38    at_work
4   Kelsey    14  at_school

[5 rows x 3 columns]
```

Overall Summary

To see a descriptive statistical summary of your dataframe's numeric columns, use `.describe()`:

```
>>> df.describe()
              age
count     5.00000
mean    619.40000
std    1330.85341
min      10.00000
25%      14.00000
50%      35.00000
75%      38.00000
max    3000.00000
```

View Columns

`.columns` will display the name of the columns in your dataframe:

```
>>> df.columns
```

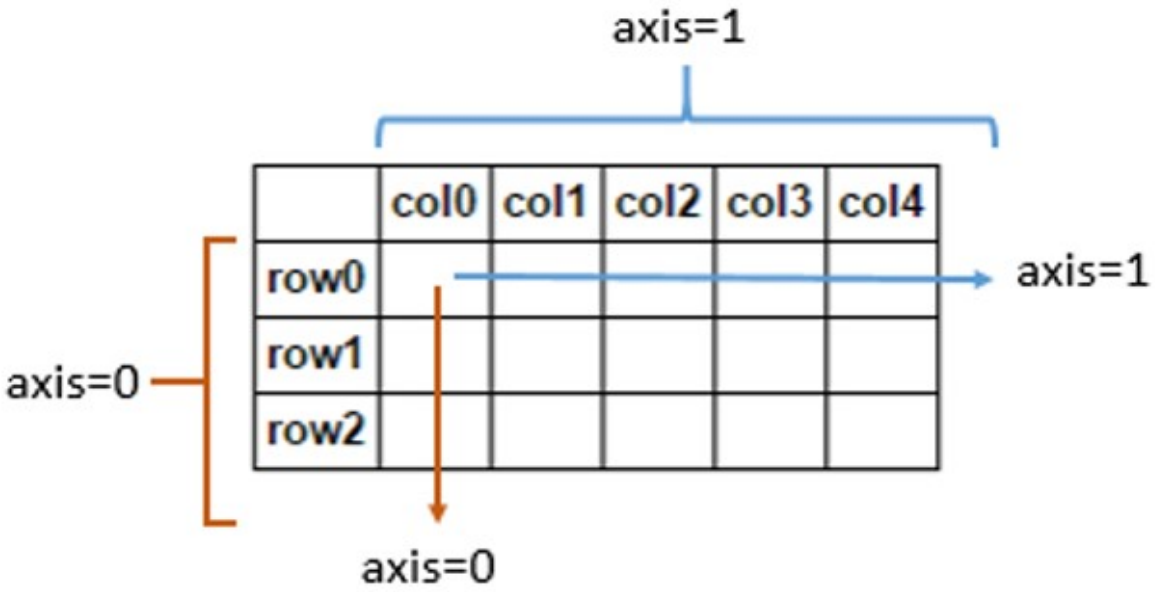
```
Index([u'name', u'age', u'location'],
      dtype='object')
```

View Indices

Finally, to display the index values, enter `.index`:

```
>>> df.index
RangeIndex(start=0, stop=649, step=1)
```

A note of caution: While generally we would say *axis* is another word for *dimension* or *feature*, Pandas uses the word differently. In Pandas, *axes* refers to the two-dimensional, matrix-like shape of your dataframe. Samples span horizontal rows and are stacked vertically on top of one another by index (axis=0). Features are vertical spans that are stacked horizontally next to each other by columns (axis=1):



In this context, if you see or hear the term axis, assume the speaker is talking about the layout of your dataframe as opposed to the dimensionality of your features.