Pandas I

Aidan Hunt, University of Washington

Learning Objectives

After this lesson, students will be able to:

- Identify the components of `DataFrame` and `Series` objects
- Import data directly as DataFrames using Pandas
- Index `DataFrame` and `Series` objects using multiple schemes.

Check-in

- Fill out mid-quarter survey
- Homework 4 due on Friday
- Homework 5 posted on Friday
- More details on final Python guide will be posted by the end of the week

---

Framing

Representing tabular data

Another type of data format we work with as engineers: spreadsheets/tables

- Data organized into rows and columns
- Columns represent different categories of data of interest, and may have different data types
  - E.g., in a course roster, column of names (strings), column of student IDs (integers), etc.
- Rows synchronize the entries of the columns
- Often convenient to import csv/excel files into

Question: How could we create a spreadsheet from existing data types?

- Dictionary of NumPy arrays!
- Dictionary "keys" are the column names
- Dictionary values are NumPy arrays that represent the row-entries in that column.

Question: What limitations might be encounter with this approach?

- As we discovered when discussing how we could represent matrices with lists of lists, this sort of representation of a spreadsheet is not suitable for common spreadsheet operations.
- Dictionary values are unlinked
- E.g., filtering, indexing row/column combinations quickly are difficult

As before, the solution is to use a specialized data structure to represent spreadsheet-like data in Python.

The Pandas Package

Pandas is another package introduces the `DataFrame` data structure.

- Built on top of the NumPy package!
- Tons of tools for working with tabular data
- We're just going to scratch the surface this week

To use Pandas:

In [32]:
```
import numpy as np
import pandas as pd
```

Motivating Example

To motivate our discussion, we will look at timeseries data from the **Skykomish River as reported by the USGS from 2020-2022**:

- Show data as retrieved from USGS website
- `sky_flow.txt` Measured flow rates (cfs)
- `sky_height.text` Measured water levels (ft)
- `sky_waterTemp.txt` Measured water temperatures (C)

Importing data

Let's quickly import the flow data into Python using `pd.read_csv()`.

In [47]:
```
# Import packages
import numpy as np
import pandas as pd

# filename = 'skykomish.txt'
# skiprows: Skip first 29 lines (data starts on line 30)
skyFlow = pd.read_csv('sky_flow.txt', skiprows=29)

sky # Display
```

Out[47]:

	USGS	12134500	2020-01-01 01:15	PST	17800	A
0	USGS	12134500	2020-01-01 01:30	PST	17900	A
1	USGS	12134500	2020-01-01 01:45	PST	18300	A
2	USGS	12134500	2020-01-01 02:00	PST	18500	A
3	USGS	12134500	2020-01-01 02:15	PST	18800	A
4	USGS	12134500	2020-01-01 02:30	PST	19000	A
...	...	...	...	...	...	...
105234	USGS	12134500	2023-01-01 23:45	PST	4970	A
105235	USGS	12134500	2023-01-02 00:00	PST	4930	A

105236	USGS	12134500	2023-01-02 00:15	PST	4910	A
105237	USGS	12134500	2023-01-02 00:30	PST	4890	A
105238	USGS	12134500	2023-01-02 00:45	PST	4910	A

105239 rows × 6 columns

Each entry is just one string...we need to specify a delimiter. Looks like it is tab delimited!

In [48]:
```
# filename = 'skykomish.txt'
# skiprows: Skip first 29 lines (data starts on line 30)
# delimiter: Columns are separated by \t (tab) characters

sky = pd.read_csv('sky_flow.txt', skiprows=29, delimiter='\t')

sky # Display
```

Out[48]:

	USGS	12134500	2020-01-01 01:15	PST	17800	A
0	USGS	12134500	2020-01-01 01:30	PST	17900	A
1	USGS	12134500	2020-01-01 01:45	PST	18300	A
2	USGS	12134500	2020-01-01 02:00	PST	18500	A
3	USGS	12134500	2020-01-01 02:15	PST	18800	A
4	USGS	12134500	2020-01-01 02:30	PST	19000	A
...	...	...	...	...	...	...
105138	USGS	12134500	2022-12-31 23:45	PST	6650	A
105139	USGS	12134500	2023-01-01 00:00	PST	6670	A
105140	USGS	12134500	2023-01-01 00:15	PST	6650	A
105141	USGS	12134500	2023-01-01 00:30	PST	6620	A
105142	USGS	12134500	2023-01-01 00:45	PST	6650	A

105143 rows × 6 columns

That looks better! The data now matches the format we saw on the USGS website.

Okay, what are our columns names?

In [35]:
```
sky.columns
```

Out[35]:
```
Index(['USGS', '12134500', '2020-01-01 01:15', 'PST', '17800', 'A'], dtype='object')
```

Looks like Pandas is using the first non-skipped row as the headers, which isn't really what we want. So let's specify our headers explicitly.

In [49]:
```
# filename = 'skykomish.txt'
# skiprows: Skip first 29 lines (data starts on line 30)
# delimiter: Columns are separated by \t (tab) characters

colNames = ['Agency', 'Site Number', 'Timestamp', 'Time zone', 'Flow', 'Code']
skyFlow = pd.read_csv('sky_flow.txt', skiprows=29, delimiter='\t', names=colNames)

skyFlow # Display
```

Out[49]:

	Agency	Site Number	Timestamp	Time zone	Flow	Code
0	USGS	12134500	2020-01-01 01:15	PST	17800	A
1	USGS	12134500	2020-01-01 01:30	PST	17900	A
2	USGS	12134500	2020-01-01 01:45	PST	18300	A
3	USGS	12134500	2020-01-01 02:00	PST	18500	A
4	USGS	12134500	2020-01-01 02:15	PST	18800	A
...	...	...	...	...	...	...
105139	USGS	12134500	2022-12-31 23:45	PST	6650	A
105140	USGS	12134500	2023-01-01 00:00	PST	6670	A
105141	USGS	12134500	2023-01-01 00:15	PST	6650	A
105142	USGS	12134500	2023-01-01 00:30	PST	6620	A
105143	USGS	12134500	2023-01-01 00:45	PST	6650	A

105144 rows × 6 columns

We now have tabular data in Python

- Columns of DataFrame correspond to the columns of the text file
- There is also an "index" column visible, which numbers the rows.

Pandas Data Structures

Pandas introduces two major new data structures: The `Series` and the `DataFrame`. These two data structures are related.

The DataFrame

The `DataFrame` is the entire spreadsheet:

- A collection of columns.
- Rows are organized by a shared index (can be anything!)

For example, maybe we want the index of our dataframe to be the timestamp, since this is what distinguishes rows of the spreadsheet from each other:

In [50]:
```
skyFlow = skyFlow.set_index('Timestamp')
skyFlow
```

Out[50]:

Timestamp	Agency	Site Number	Time zone	Flow	Code
2020-01-01 01:15	USGS	12134500	PST	17800	A
2020-01-01 01:30	USGS	12134500	PST	17900	A
2020-01-01 01:45	USGS	12134500	PST	18300	A
2020-01-01 02:00	USGS	12134500	PST	18500	A

2020-01-01 02:15	USGS	12134500	PST	18800	A
...	...	...	...	...	...
2022-12-31 23:45	USGS	12134500	PST	6650	A
2023-01-01 00:00	USGS	12134500	PST	6670	A
2023-01-01 00:15	USGS	12134500	PST	6650	A
2023-01-01 00:30	USGS	12134500	PST	6620	A
2023-01-01 00:45	USGS	12134500	PST	6650	A

105144 rows × 5 columns

The Series

A `Series` is basically an individual column of a spreadsheet

- Like a cross between a NumPy vector and a dictionary:
  - Similar to a NumPy vector: single type, ordered
  - Similar to a dictionary: indices are like keys
- A `Series` is returned when we access the columns of a `DataFrame`.
  - Note that a series retains its "name" from the DataFrame.

In [52]:
```
# Access the "flow" column
skyFlow.Flow
```

Out[52]:
```
Timestamp
2020-01-01 01:15    17800
2020-01-01 01:30    17900
2020-01-01 01:45    18300
2020-01-01 02:00    18500
2020-01-01 02:15    18800
                    ...
2022-12-31 23:45     6650
2023-01-01 00:00     6670
2023-01-01 00:15     6650
2023-01-01 00:30     6620
2023-01-01 00:45     6650
Name: Flow, Length: 105144, dtype: int64
```

In [53]:
```
# Equivalent syntax:
skyFlow['Flow']
```

Out[53]:
```
Timestamp
2020-01-01 01:15    17800
2020-01-01 01:30    17900
2020-01-01 01:45    18300
2020-01-01 02:00    18500
2020-01-01 02:15    18800
                    ...
2022-12-31 23:45     6650
2023-01-01 00:00     6670
2023-01-01 00:15     6650
2023-01-01 00:30     6620
2023-01-01 00:45     6650
Name: Flow, Length: 105144, dtype: int64
```

We can think of a `Series` as an dictionary-like object "wraps" a NumPy array of data and the corresponding indices together. We can access the index and data separately, too.

In [57]:
```
# Access index of series
skyFlow.Flow.index
```

Out[57]:
```
Index(['2020-01-01 01:15', '2020-01-01 01:30', '2020-01-01 01:45',
       '2020-01-01 02:00', '2020-01-01 02:15', '2020-01-01 02:30',
       '2020-01-01 02:45', '2020-01-01 03:00', '2020-01-01 03:15',
       '2020-01-01 03:30',
       ...
       '2022-12-31 22:30', '2022-12-31 22:45', '2022-12-31 23:00',
       '2022-12-31 23:15', '2022-12-31 23:30', '2022-12-31 23:45',
       '2023-01-01 00:00', '2023-01-01 00:15', '2023-01-01 00:30',
       '2023-01-01 00:45'],
      dtype='object', name='Timestamp', length=105144)
```

In [55]:
```
# Access the underlying data of series (returns a NumPy array)
skyFlow.Flow.values
```

Out[55]:
```
array([17800, 17900, 18300, ...,  6650,  6620,  6650], dtype=int64)
```

Indexing DataFrames

There are many many ways to extract the elements of `DataFrames`

`.loc` (Label-Based)

`.loc` is an "indexer" (think of it as a cross between basic indexing and a method) that allows for label-based indexing.

- You can use it to index based on index names, column names, or a combination.
- Label-based slicing is inclusive for both start and end labels.

In [72]:
```
# Get the data in the row corresponding to the index '2022-12-31 23:45'
skyFlow.loc['2022-12-31 23:45']
```

Out[72]:
```
Agency             USGS
Site Number    12134500
Time zone           PST
Flow                6650
Code                   A
Name: 2022-12-31 23:45, dtype: object
```

In [73]:
```
# Get the data in all rows for columns 'Agency' and 'Flow'
skyFlow.loc[:, ['Agency', 'Flow']]
```

Out[73]:

Timestamp	Agency	Flow
2020-01-01 01:15	USGS	17800
2020-01-01 01:30	USGS	17900
2020-01-01 01:45	USGS	18300
2020-01-01 02:00	USGS	18500
2020-01-01 02:15	USGS	18800
...	...	...

2022-12-31 23:45	USGS	6650	
2023-01-01 00:00	USGS	6670	
2023-01-01 00:15	USGS	6650	
2023-01-01 00:30	USGS	6620	
2023-01-01 00:45	USGS	6650	

105144 rows × 2 columns

In [74]:
```
# Get the data in rows from '2020-01-01 12:00' to '2021-01-01 12:00' for columns 'Agency
skyFlow.loc['2020-01-01 12:00':'2021-01-01 12:00', 'Agency':'Flow']
```

Out[74]:

Timestamp	Agency	Site Number	Time zone	Flow
2020-01-01 12:00	USGS	12134500	PST	18900
2020-01-01 12:15	USGS	12134500	PST	18800
2020-01-01 12:30	USGS	12134500	PST	18600
2020-01-01 12:45	USGS	12134500	PST	18500
2020-01-01 13:00	USGS	12134500	PST	18300
...	...	...	...	...
2021-01-01 11:00	USGS	12134500	PST	5160
2021-01-01 11:15	USGS	12134500	PST	5230
2021-01-01 11:30	USGS	12134500	PST	5250
2021-01-01 11:45	USGS	12134500	PST	5250
2021-01-01 12:00	USGS	12134500	PST	5320

35095 rows × 4 columns

`.iloc` (Integer-Based)

`.iloc` is similar to loc, but uses integer indices instead of labels.

- Specify row and/or column indices that we want
- For slicing, start index is inclusive, end index is exclusive

Note: This works even if we have changed our "index" to something non-numeric (the basic 0,1,2,3... index remains under the hood).

In [75]:
```
# Get data from the first five rows
skyFlow.iloc[0:5]
```

Out[75]:

Timestamp	Agency	Site Number	Time zone	Flow	Code
2020-01-01 01:15	USGS	12134500	PST	17800	A
2020-01-01 01:30	USGS	12134500	PST	17900	A

2020-01-01 01:45	USGS	12134500	PST	18300	A
2020-01-01 02:00	USGS	12134500	PST	18500	A
2020-01-01 02:15	USGS	12134500	PST	18800	A

In [76]:
```
# Get data from all rows, and the last 3 columns
skyFlow.iloc[:, 2:]
```

Out[76]:

Timestamp	Time zone	Flow	Code
2020-01-01 01:15	PST	17800	A
2020-01-01 01:30	PST	17900	A
2020-01-01 01:45	PST	18300	A
2020-01-01 02:00	PST	18500	A
2020-01-01 02:15	PST	18800	A
...	...	...	...
2022-12-31 23:45	PST	6650	A
2023-01-01 00:00	PST	6670	A
2023-01-01 00:15	PST	6650	A
2023-01-01 00:30	PST	6620	A
2023-01-01 00:45	PST	6650	A

105144 rows × 3 columns

In [ ]:
```
```

`.at` and `.iat`

`.at` and `.iat` are like `.loc` and `.iloc`, but are for extracting single elements from the DataFrame.

In [77]:
```
# Get the flow rate at midnight on New Year's day, 2022.
skyFlow.at['2022-01-01 00:00', 'Flow']
```

Out[77]:
```
2120
```

In [79]:
```
# Get the element in the 1000th row, 4th column of the DataFrame
skyFlow.iat[999, 3]
```

Out[79]:
```
6060
```

Logical indexing

We can use logical indexing to find rows that meet certain criteria, as well.

In [83]:
```
# Find all rows that correspond to flow rates greater than 6000cfs
skyFlow.loc[skyFlow.Flow > 6000, 'Flow':]
```

Out[83]:

	Flow	Code

Timestamp		
2020-01-01 01:15	17800	A
2020-01-01 01:30	17900	A
2020-01-01 01:45	18300	A
2020-01-01 02:00	18500	A
2020-01-01 02:15	18800	A
...	...	...
2022-12-31 23:45	6650	A
2023-01-01 00:00	6670	A
2023-01-01 00:15	6650	A
2023-01-01 00:30	6620	A
2023-01-01 00:45	6650	A

26595 rows × 2 columns

In [85]:
```
# Can also use brackets directly without loc, but just can't choose columns
skyFlow[skyFlow.Flow > 6000]
```

Out[85]:

Timestamp	Agency	Site Number	Time zone	Flow	Code
2020-01-01 01:15	USGS	12134500	PST	17800	A
2020-01-01 01:30	USGS	12134500	PST	17900	A
2020-01-01 01:45	USGS	12134500	PST	18300	A
2020-01-01 02:00	USGS	12134500	PST	18500	A
2020-01-01 02:15	USGS	12134500	PST	18800	A
...	...	...	...	...	...
2022-12-31 23:45	USGS	12134500	PST	6650	A
2023-01-01 00:00	USGS	12134500	PST	6670	A
2023-01-01 00:15	USGS	12134500	PST	6650	A
2023-01-01 00:30	USGS	12134500	PST	6620	A
2023-01-01 00:45	USGS	12134500	PST	6650	A

26595 rows × 5 columns

Summarize

- Pandas gives us data structures for handling tabular data that are built on top of NumPy
  - `DataFrame` : like a spreadsheet
  - `Series` : like a column of the spreadsheet

    - Both `DataFrame` and `Series` objects have an "explicit" index (like dictionary keys) and "implicit" numerical index (like NumPy arrays)
    - All columns are single type
    - Like dictionary keys, index can be anything immutable
- Very easy to import data .txt, .csv, etc. data directly as a DataFrame
- Can index `DataFrame` and `Series` objects using labels or using integer index

---

Converting Timestamps to Datetimes

Right now, our timestamps (index) look like strings

In [87]:
```
# Get the timestamp column (returns a series)
skyFlow.index.values
```

Out[87]:
```
array(['2020-01-01 01:15', '2020-01-01 01:30', '2020-01-01 01:45', ...,
       '2023-01-01 00:15', '2023-01-01 00:30', '2023-01-01 00:45'],
      dtype=object)
```

Let's convert these to a more convenient format: `datetime`, a datatype specifically designed for time-related data!

In [90]:
```
# Convert the timestamp column to datetime
pd.to_datetime(skyFlow.index)
```

Out[90]:
```
DatetimeIndex(['2020-01-01 01:15:00', '2020-01-01 01:30:00',
               '2020-01-01 01:45:00', '2020-01-01 02:00:00',
               '2020-01-01 02:15:00', '2020-01-01 02:30:00',
               '2020-01-01 02:45:00', '2020-01-01 03:00:00',
               '2020-01-01 03:15:00', '2020-01-01 03:30:00',
               ...
               '2022-12-31 22:30:00', '2022-12-31 22:45:00',
               '2022-12-31 23:00:00', '2022-12-31 23:15:00',
               '2022-12-31 23:30:00', '2022-12-31 23:45:00',
               '2023-01-01 00:00:00', '2023-01-01 00:15:00',
               '2023-01-01 00:30:00', '2023-01-01 00:45:00'],
              dtype='datetime64[ns]', name='Timestamp', length=105144, freq=None)
```

In [92]:
```
# But note that the original column remains unchanged...
skyFlow.index
```

Out[92]:
```
Index(['2020-01-01 01:15', '2020-01-01 01:30', '2020-01-01 01:45',
       '2020-01-01 02:00', '2020-01-01 02:15', '2020-01-01 02:30',
       '2020-01-01 02:45', '2020-01-01 03:00', '2020-01-01 03:15',
       '2020-01-01 03:30',
       ...
       '2022-12-31 22:30', '2022-12-31 22:45', '2022-12-31 23:00',
       '2022-12-31 23:15', '2022-12-31 23:30', '2022-12-31 23:45',
       '2023-01-01 00:00', '2023-01-01 00:15', '2023-01-01 00:30',
       '2023-01-01 00:45'],
      dtype='object', name='Timestamp', length=105144)
```

In [94]:
```
# Assign it to change it!
skyFlow.index = pd.to_datetime(skyFlow.index)
skyFlow.index
```

Out[94]:
```
DatetimeIndex(['2020-01-01 01:15:00', '2020-01-01 01:30:00',
               '2020-01-01 01:45:00', '2020-01-01 02:00:00',
               '2020-01-01 02:15:00', '2020-01-01 02:30:00',
               '2020-01-01 02:45:00', '2020-01-01 03:00:00',
```

```
 '2020-01-01 03:15:00', '2020-01-01 03:30:00',
 ...
 '2022-12-31 22:30:00', '2022-12-31 22:45:00',
 '2022-12-31 23:00:00', '2022-12-31 23:15:00',
 '2022-12-31 23:30:00', '2022-12-31 23:45:00',
 '2023-01-01 00:00:00', '2023-01-01 00:15:00',
 '2023-01-01 00:30:00', '2023-01-01 00:45:00'],
dtype='datetime64[ns]', name='Timestamp', length=105144, freq=None)
```

Why format things this way? Two main reasons:

1) Because we can easily access properties of the timestamps, like the month and year.

In [98]:
```
# Extract timestamp year
skyFlow.index.year
```

Out[98]:
```
Int64Index([2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
            ...
            2022, 2022, 2022, 2022, 2022, 2022, 2023, 2023, 2023, 2023],
           dtype='int64', name='Timestamp', length=105144)
```

In [100…
```
# Extract timestamp days
skyFlow.index.day
```

Out[100]:
```
Int64Index([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
            ...
            31, 31, 31, 31, 31, 31,  1,  1,  1,  1],
           dtype='int64', name='Timestamp', length=105144)
```

2) Because we can now index based on year or month, without having to type out the entire timestamp

In [104…
```
skyFlow.loc['2020'] # Get all data from the year 2020
```

Out[104]:

Timestamp	Agency	Site Number	Time zone	Flow	Code
2020-01-01 01:15:00	USGS	12134500	PST	17800	A
2020-01-01 01:30:00	USGS	12134500	PST	17900	A
2020-01-01 01:45:00	USGS	12134500	PST	18300	A
2020-01-01 02:00:00	USGS	12134500	PST	18500	A
2020-01-01 02:15:00	USGS	12134500	PST	18800	A
…	…	…	…	…	…
2020-12-31 22:45:00	USGS	12134500	PST	5100	A
2020-12-31 23:00:00	USGS	12134500	PST	5040	A
2020-12-31 23:15:00	USGS	12134500	PST	5040	A
2020-12-31 23:30:00	USGS	12134500	PST	5060	A
2020-12-31 23:45:00	USGS	12134500	PST	5040	A

35089 rows × 5 columns

In [105…
```
skyFlow.loc['2021-03'] # Get all data from March 2021
```

Out[105]:

Timestamp	Agency	Site Number	Time zone	Flow	Code

2021-03-01 00:00:00	USGS	12134500	PST	2940	A
2021-03-01 00:15:00	USGS	12134500	PST	2930	A
2021-03-01 00:30:00	USGS	12134500	PST	2940	A
2021-03-01 00:45:00	USGS	12134500	PST	2930	A
2021-03-01 01:00:00	USGS	12134500	PST	2940	A
...	...	...	...	...	...
2021-03-31 22:45:00	USGS	12134500	PDT	2330	A
2021-03-31 23:00:00	USGS	12134500	PDT	2300	A
2021-03-31 23:15:00	USGS	12134500	PDT	2300	A
2021-03-31 23:30:00	USGS	12134500	PDT	2290	A
2021-03-31 23:45:00	USGS	12134500	PDT	2320	A

2972 rows × 5 columns