3. Loading and Handling Pandas Data

Overview

Questions

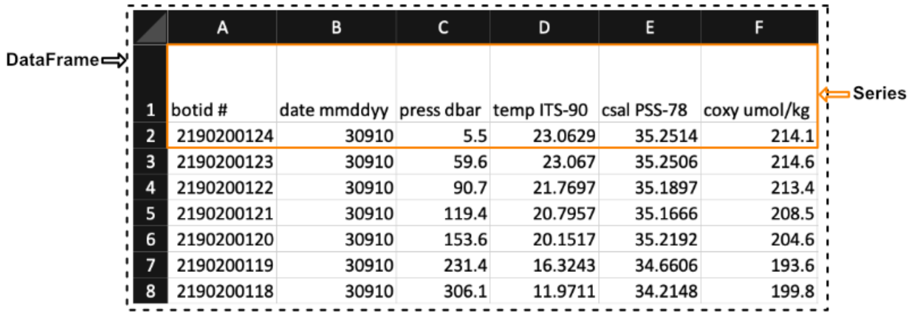
- How are Pandas data structures setup?
- How to load data into Pandas?
- How to write data from Pandas to a file.

Objectives

- Understand the usefulness of Pandas when loading data.
- Understand how to load data and deal with common issues.

Pandas Data Structures

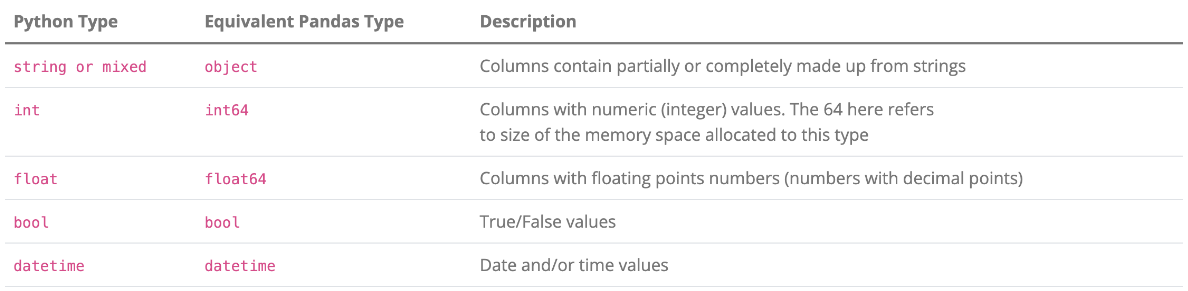
- The two primary data structures of Pandas:
- Series: an array, or list like collection of values
    - Similar to a single row or a single column in Excel.
- DataFrames: a table-like structure consisting of a collection of rows or column



	A	B	C	D	E	F
1	botid #	date mmddyy	press dbar	temp ITS-90	csal PSS-78	coxy umol/kg
2	2190200124	30910	5.5	23.0629	35.2514	214.1
3	2190200123	30910	59.6	23.067	35.2506	214.6
4	2190200122	30910	90.7	21.7697	35.1897	213.4
5	2190200121	30910	119.4	20.7957	35.1666	208.5
6	2190200120	30910	153.6	20.1517	35.2192	204.6
7	2190200119	30910	231.4	16.3243	34.6606	193.6
8	2190200118	30910	306.1	11.9711	34.2148	199.8

Pandas Data Types

- Series and DataFrames have many functions that facilitate data analysis
    - Filter or impute missing values in a Series (column)
    - Select rows from a Series or DataFrame using conidtional operations
    - Convert DataFrames across formats
    - etc.
- Pandas attempts to assign the correct type to each column based on the type of data it contains.
    - You can override the data type assigned by Pandas



Python Type	Equivalent Pandas Type	Description
`string or mixed`	`object`	Columns contain partially or completely made up from strings
`int`	`int64`	Columns with numeric (integer) values. The 64 here refers to size of the memory space allocated to this type
`float`	`float64`	Columns with floating points numbers (numbers with decimal points)
`bool`	`bool`	True/False values
`datetime`	`datetime`	Date and/or time values

About File Formats

- Dozens if not hundreds of file formats.
    - Some such as Excel's format, are binary and are not meant to be read by a human
        - Typically, we use custom formatting to delineate columns and rows.
    - Others are plain text and can be opened and edited in any text editor.
- Plain text file formats fall into one of the following two categories: Delimited and Fixed Width
    - Delimited files are organized such that columns and rows are separated by a certain character called a delimiter
    - Fixed width files are those where each entry in a column has a fixed number of characters
    - Character delimited using special tags or characters.

About File Formats - Cont’d

- Plain text are the most popular formats.
- In this workshop, we will use the ‘Comma Separated Values’ `.csv` format
- Entries delimited by commas and the rows are delimited by a new line
- The file may contian a header with column labels
- Rows may have an index

```
,column1,column2,column3,
row1,a,b,c
row2,d,e,f
row3,g,h,i
```

- The file above is in the csv (comma delimited) format, has a header with a missing first value (potentially index)

Loading and Parsing Data

The following csv data is stored in a file called `my_data.csv`

```
,column1,column2,column3,
row1,a,b,c
row2,d,e,f
row3,g,h,i
```

- The table looks as follows:

```
|      | column 1 | column 2 | column 3 |
| ---- | -------- | -------- | -------- |
| row1 | a        | b        | c        |
| row2 | d        | e        | f        |
| row3 | g        | h        | i        |
```

Import the Pandas Package

- `Series` and `DataFrames` can be created from scratch or loading their data from a file.
- Pandas supports a variety of file formats, such as comma delimited (csv), tab delimted (tsv), excel, etc.
- Loading a specific format is done using custom functions. For example:
  - read a csv using `read_csv`
  - Read an Excel file using `read_excel`
  - read a JSON file using `read_json`
  - etc..

Loading and Parsing Data

- Before loading the data we need to import the pandas package.
  - Pandas is typically imported using the `pd` alias

```
import pandas as pd
```

- To read a csv file into a variable called `df` we can write:

```
df = pd.read_csv('data/some_data.csv')
```

- Simply typing the variable name `df` will display the table in a user friendly format.

```
df
```

	botid #	date mmddyy	press dbar	temp ITS-90	csal PSS-78	coxy umol/kg	ph
0	2190200520	31010	25.5	23.3391	35.2731	213.4	Null
1	2190200519	31010	25.5	23.3389	35.2731	213.4	8.068
2	2190200518	31010	36.1	23.3381	35.2730	213.3	Null
3	2190200517	31010	45.6	23.3361	35.2728	213.8	8.064
4	2190200516	31010	59.9	23.2012	35.2685	213.9	Null
5	2190200515	31010	75.3	23.0755	35.2563	214.1	8.06
6	2190200514	31010	85.5	23.0472	35.2521	214.5	Null

Loading and Parsing Data

- Each of Pandas functions for reading a text file provides many parameters to customaize how a file is read

- Customize the field delimiter or separator (comma by default)

```
df = pd.read_csv('data/weired_format.csv', sep='|')
```

- or

```
df = pd.read_csv('data/tsv_example.tsv', sep='\t')
```

- Read in a a small subset

```
df = pd.read_csv('data/tsv_example.tsv', nrows=5)
```

Headers and Indexes

- Headers (column labels) and index (row labels) are very useful for indexing into the data
- By default, `read_csv` assumes that:
    - The first row is the table header
    - Rows are indexed using integer values from 0 to n-1, were n is the number of rows in the data.
- You can change the `read_csv` behaviour to omit the header or rename the columns
- You can change the `read_csv` behaviour to specify which column to use as the index.

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

# ```bash
# |column1|column2|column3
# row1|a|b|c
# row2|d|e|f
# row3|g|h|i
# ```

df = pd.read_csv("data/weired_format.csv", sep="|")
df
```

Out[1]:

	Unnamed: 0	column1	column2	column3
0	row1	a	b	c
1	row2	d	e	f
2	row3	g	h	i
3	row4	j	k	l
4	row5	m	n	o

In [2]:
```
# ```bash
# row1|a|b|c
# row2|d|e|f
# row3|g|h|i
# ```

df = pd.read_csv("data/weired_format_no_header.csv",
                 sep="|")
df
```

Out[2]:

	row1	a	b	c
0	row2	d	e	f
1	row3	g	h	i
2	row4	j	k	l
3	row5	m	n	o

```
In [3]: df = pd.read_csv("data/weired_format_no_header.csv",
                         sep="|", header= None)
        df
```

Out[3]:

	0	1	2	3
0	row1	a	b	c
1	row2	d	e	f
2	row3	g	h	i
3	row4	j	k	l
4	row5	m	n	o

```
In [17]: df = pd.read_csv("data/weired_format_no_header.csv",
                          sep="|", header= None, nrows=2)
         df
```

Out[17]:

	0	1	2	3
0	row1	a	b	c
1	row2	d	e	f

```
In [23]: # row1,a,b,c
         # row2,d,e,f
         # row3,g,h,i



         df = pd.read_csv("data/weired_format_no_header.csv",
                          sep="|")
         df
```

Out[23]:

	row1	a	b	c
0	row2	d	e	f
1	row3	g	h	i
2	row4	j	k	l
3	row5	m	n	o

```
In [24]: # row1,a,b,c
         # row2,d,e,f
         # row3,g,h,i




         df = pd.read_csv("data/weired_format_no_header.csv",
                                 sep="|", header=None)
         df
```

Out[24]:

	0	1	2	3
0	row1	a	b	c
1	row2	d	e	f
2	row3	g	h	i
3	row4	j	k	l
4	row5	m	n	o

```
In [27]: # row1,a,b,c
         # row2,d,e,f
         # row3,g,h,i




         df = pd.read_csv("data/weired_format_no_header.csv",
                          sep="|",
                          names=["COL_ONE", "COL_TWO", "COL_THREE"])
         df
```

Out[27]:

	COL_ONE	COL_TWO	COL_THREE
row1	a	b	c
row2	d	e	f
row3	g	h	i
row4	j	k	l
row5	m	n	o

```
In [28]: # row1,a,b,c
         # row2,d,e,f
         # row3,g,h,i

         df = pd.read_csv("data/weired_format_no_header.csv",
                          sep="|",
                          names=["COL_ONE", "COL_TWO", "COL_THREE"],
                          index_col=0)
         df
```

Out[28]:

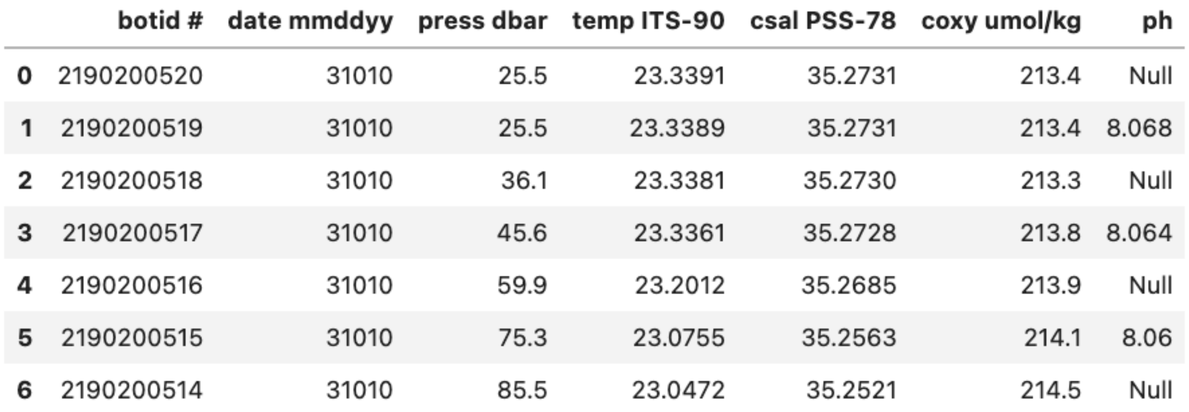
	COL_ONE	COL_TWO	COL_THREE
row1	a	b	c
row2	d	e	f
row3	g	h	i
row4	j	k	l
row5	m	n	o

Missing Values

- There are often missing values in a real-world datasets.
    - E.g. `NA`, `N.A.`, `9999`, `missing`, `''`, etc.
- Some functions depend on properly identifying missing values.
    - What is the averge of [1, 2 , 3, 'UNKNOWN']?
- Pandas identifies missing NaN (Not a Number)
    - Provides ways to handle missing values in computation values.
- `read_csv` can take as a parameter the value used to represent missing values. For example,

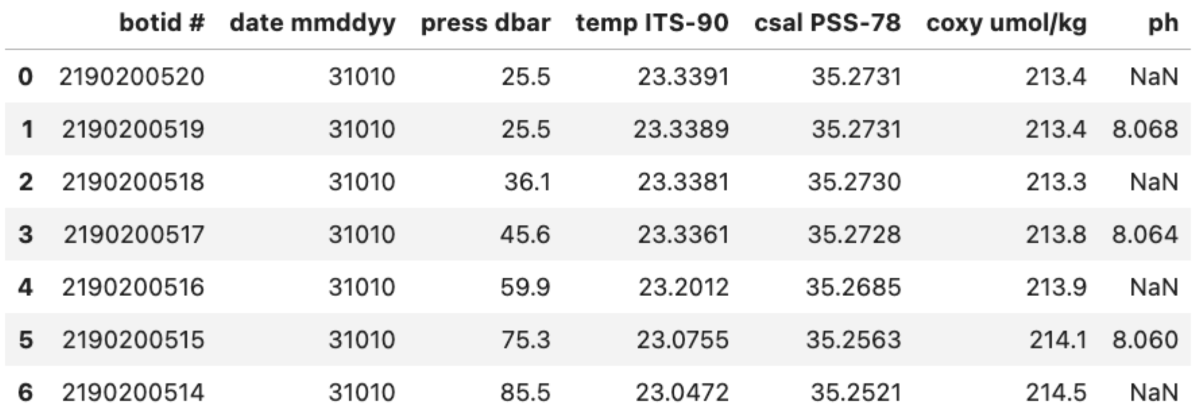
```
df = pd.read_csv("data/null_values_example.csv", na_values='Null')
df
```

Without `na_values='Null'`



	botid #	date mmddyy	press dbar	temp ITS-90	csal PSS-78	coxy umol/kg	ph
0	2190200520	31010	25.5	23.3391	35.2731	213.4	Null
1	2190200519	31010	25.5	23.3389	35.2731	213.4	8.068
2	2190200518	31010	36.1	23.3381	35.2730	213.3	Null
3	2190200517	31010	45.6	23.3361	35.2728	213.8	8.064
4	2190200516	31010	59.9	23.2012	35.2685	213.9	Null
5	2190200515	31010	75.3	23.0755	35.2563	214.1	8.06
6	2190200514	31010	85.5	23.0472	35.2521	214.5	Null

With `na_values='Null'`:



	botid #	date mmddyy	press dbar	temp ITS-90	csal PSS-78	coxy umol/kg	ph
0	2190200520	31010	25.5	23.3391	35.2731	213.4	NaN
1	2190200519	31010	25.5	23.3389	35.2731	213.4	8.068
2	2190200518	31010	36.1	23.3381	35.2730	213.3	NaN
3	2190200517	31010	45.6	23.3361	35.2728	213.8	8.064
4	2190200516	31010	59.9	23.2012	35.2685	213.9	NaN
5	2190200515	31010	75.3	23.0755	35.2563	214.1	8.060
6	2190200514	31010	85.5	23.0472	35.2521	214.5	NaN

Writing Data in Text Format

Pandas DataFrames have a collection of `to_<filetype>` methods used to write data to disk

- Example, `to_csv()` takes the parameter path and and will either create a new file or overwrite the existing file with the same name.

```
df.to_csv('data/new_file.csv')
```

- Has a number of optional parameters to change the delimiter, write the numerical automatically generated index, omit certain columns, etc.

Key Points

- Pandas contains numerous methods to help load/write data to/from files of different types.
- `read_csv` is highly customizable and can allow you to handle many issues when loading the data.

1 - Exercise: Read an Excel File

Try it yourself! Fill in the blanks to load the first 10 lines of the excel file `'20_sales_records.xlsx'` into a variable called `df` and then display the `DataFrame`.

- Instructions
    - The file is located in the `data` folder.
    - Use the `read_excel` command along with the argument you learned to parse a specified number of rows.
    - This file has `NaN` values that are not automatically detected. They are labeled as `'none'`. Have Pandas interpret these as `NaN` values upon loading of the dataset.
    - Display the results.