

5. DataFrame Analysis

Questions

- What are some common attributes of Pandas `DataFrame` s?
- What are some common methods of Pandas `DataFrame` s?
- How can you do arithmetic between two Pandas columns?

Objectives

- Learn how to access `DataFrame` attributes.
- Learn how to get statistics on a loaded `DataFrame` .
- Learn how to sum two Pandas `DataFrame` columns together.

DataFrame Attributes

- `DataFrame` are python objects
 - They have access to dozens of attributes and methods that can be used to inspect, wrangle or analyze the data.

Attribute	Description
<code>shape</code>	Returns a tuple representing the dimensionality of the <code>DataFrame</code> .
<code>size</code>	Returns an int representing the number of elements in this object.
<code>dtypes</code>	Returns the data types in the <code>DataFrame</code> .
<code>columns</code>	Returns a <code>Series</code> of the header names from the <code>DataFrame</code>

- See Pandas doc for [list of attributes and methods](#)
- In Jupyter (and most other IDEs), you can use `tab` to see which attributes and objects are available on an object
- use `help` function or the `?` character to get more details about a method or attribute

```
In [19]: import pandas as pd  
df = pd.read_csv("data/types_dataframe.csv")  
df.shape
```

```
Out[19]: (8, 7)
```

```
In [20]: df.columns
```

```
Out[20]: Index(['Sample ID', 'date mmddyy', 'press dbar', 'temp ITS-90',  
               'csal PSS-78',  
               'coxy umol/kg', 'ph'],  
              dtype='object')
```

Inspecting Data Types

- `DataFrame` types determine methods that can be used on the data
 - You can compute the mean of a numeric value but not an Object column (i.e. words).
 - Common friction point in data analysis (compute average price)
- The `dtypes` attribute is an attribute that provides access to the type of each column
 - Note the column `date mmddyy`

	Sample ID	date mmddyy	press dbar	temp ITS-90	csal PSS-78	coxy umol/kg	ph
0	Sample-1	40610	239.8	18.9625	35.0636	NaN	7.951
1	Sample-2	40610	280.7	16.1095	34.6103	192.3	NaN
2	Sample-3	40610	320.1	12.9729	34.2475	190.8	NaN
3	Sample-4	40610	341.3	11.9665	34.1884	191.3	7.780
4	Sample-5	40610	360.1	11.3636	34.1709	203.5	NaN
5	Sample-6	40610	385.0	10.4636	34.1083	193.7	NaN
6	Sample-7	40610	443.7	8.5897	34.0567	156.5	NaN
7	Sample-8	40610	497.8	7.1464	34.0424	110.7	7.496

```
In [6]: df.dtypes
```

```
Out[6]: Sample ID      object  
date mmddyy      int64  
press dbar      float64  
temp ITS-90      float64  
csal PSS-78      float64  
coxy umol/kg      float64  
ph               float64  
dtype: object
```

Data Types

- Pandas data types:

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

Data Types Inference

- Pandas attempts to figure out the column's data type when reading in the data.
 - Type is inferred purely based on the data format.
- Datatypes can be specified when loading the data or after loading the data.
- For Example, we can convert the `date mmddyy` column into a new `Series` with the `datetime` type.
 - use the `to_datetime` function with the data and the date `format` (`'%m%d%y'` in this case) as inputs.
 - `datetime` uses python [date formats](#)

```
In [7]: pd.to_datetime(df['date mmddyy'], format='%m%d%y')
```

```
Out[7]: 0    2010-04-06  
1    2010-04-06  
2    2010-04-06  
3    2010-04-06  
4    2010-04-06  
5    2010-04-06  
6    2010-04-06  
7    2010-04-06  
Name: date mmddyy, dtype: datetime64[ns]
```

```
In [9]: # Add the data a as new column (same syntax as adding to a collection i
# and remove old column
df["date"] = pd.to_datetime(df['date mmdyy'], format='%m%d%y')
df = df.drop(columns=["date mmdyy"])
df.dtypes
```

```
Out[9]: Sample ID          object
press dbar          float64
temp ITS-90         float64
csal PSS-78         float64
coxy umol/kg        float64
ph                  float64
date               datetime64[ns]
dtype: object
```

In [10]:

df

Out[10]:

	Sample ID	press dbar	temp ITS- 90	csal PSS- 78	coxy umol/kg	ph	date
0	Sample-1	239.8	18.9625	35.0636	NaN	7.951	2010-04-06
1	Sample-2	280.7	16.1095	34.6103	192.3	NaN	2010-04-06
2	Sample-3	320.1	12.9729	34.2475	190.8	NaN	2010-04-06
3	Sample-4	341.3	11.9665	34.1884	191.3	7.780	2010-04-06
4	Sample-5	360.1	11.3636	34.1709	203.5	NaN	2010-04-06
5	Sample-6	385.0	10.4636	34.1083	193.7	NaN	2010-04-06
6	Sample-7	443.7	8.5897	34.0567	156.5	NaN	2010-04-06
7	Sample-8	497.8	7.1464	34.0424	110.7	7.496	2010-04-06

DataFrame Methods

- `DataFrame` and `Series` (e.g., single column) have access to a variety of built-in methods
 - Accessible through the notation `some_var_name.method_name()`

Method	Description
<code>head()</code>	Return the first <code>n=5</code> rows by default. The value of <code>n</code> can be changed.
<code>tail()</code>	Return the last <code>n=5</code> rows by default. The value of <code>n</code> can be changed.
<code>min()</code> , <code>max()</code>	Computes the numeric or alphanumeric min, max in a <code>Series</code> or <code>DataFrame</code> .
<code>sum()</code> , <code>mean()</code> , <code>std()</code> , <code>var()</code>	Computes the sum, mean, standard deviation, and variance in a <code>Series</code> or <code>DataFrame</code> .
<code>nlargest()</code>	Return the first <code>n</code> rows of the <code>Series</code> or <code>DataFrame</code> , ordered by the specified columns in descending order.
<code>count()</code>	Returns the number of non-NaN values in a <code>Series</code> or <code>DataFrame</code> .
<code>value_counts()</code>	Returns the frequency for each value in the <code>Series</code> .
<code>describe()</code>	Computes column-wise statistics.

```
In [8]: df.mean(numeric_only=True)
```

```
/var/folders/5l/gk6s2xx10qs0mkg4_9_vyp3h0000gn/T/ipykernel_9653
7/3698961737.py:1: FutureWarning: The default value of numeric_
only in DataFrame.mean is deprecated. In a future version, it w
ill default to False. In addition, specifying 'numeric_only=Non
e' is deprecated. Select only valid columns or specify the valu
e of numeric_only to silence this warning.
  df.mean()
```

```
Out[8]: date mddyy      40610.000000
press dbar      358.562500
temp ITS-90      12.196838
csal PSS-78      34.311013
coxy umol/kg     176.971429
ph              7.742333
dtype: float64
```

```
In [9]: df.mean(numeric_only=True)
```

```
Out[9]: date mddyy      40610.000000  
        press dbar      358.562500  
        temp ITS-90      12.196838  
        csal PSS-78      34.311013  
        coxy umol/kg     176.971429  
        ph              7.742333  
        dtype: float64
```

```
In [10]: # Accessing a single column  
df['press dbar']
```

```
Out[10]: 0      239.8  
         1      280.7  
         2      320.1  
         3      341.3  
         4      360.1  
         5      385.0  
         6      443.7  
         7      497.8  
         Name: press dbar, dtype: float64
```



```
In [11]: # Accessing a single column  
df['press dbar'].mean()
```

```
Out[11]: 358.5625
```

`describe()` Method

- The `describe()` method provides a range of descriptive statistics of a dataframe
- Statistics to summarize the central tendency, dispersion, and shape of a dataset's distribution
- By default, uses only numeric columns.
 - change behavior using the `include='all'` param.

In [17]:

df

Out[17]:

	Sample ID	date mmddyy	press dbar	temp ITS- 90	csal PSS- 78	coxy umol/kg	ph
0	Sample-1	40610	239.8	18.9625	35.0636	NaN	7.951
1	Sample- 2	40610	280.7	16.1095	34.6103	192.3	NaN
2	Sample- 3	40610	320.1	12.9729	34.2475	190.8	NaN
3	Sample- 4	40610	341.3	11.9665	34.1884	191.3	7.780
4	Sample- 5	40610	360.1	11.3636	34.1709	203.5	NaN
5	Sample- 6	40610	385.0	10.4636	34.1083	193.7	NaN
6	Sample-7	40610	443.7	8.5897	34.0567	156.5	NaN
7	Sample- 8	40610	497.8	7.1464	34.0424	110.7	7.496

```
In [16]: df.describe(include='all', datetime_is_numeric=True)
```

```
Out[16]:
```

	Sample ID	date mmddyy	press dbar	temp ITS- 90	csal PSS- 78	coxy umol/kg	
count	8	8.0	8.000000	8.000000	8.000000	7.000000	3.0
unique	8	NaN	NaN	NaN	NaN	NaN	
top	Sample- 1	NaN	NaN	NaN	NaN	NaN	
freq	1	NaN	NaN	NaN	NaN	NaN	
mean	NaN	40610.0	358.562500	12.196838	34.311013	176.971429	7.7
std	NaN	0.0	83.905762	3.853666	0.353064	32.726376	0.2
min	NaN	40610.0	239.800000	7.146400	34.042400	110.700000	7.4
25%	NaN	40610.0	310.250000	9.995125	34.095400	173.650000	7.6
50%	NaN	40610.0	350.700000	11.665050	34.179650	191.300000	7.7
75%	NaN	40610.0	399.675000	13.757050	34.338200	193.000000	7.8
max	NaN	40610.0	497.800000	18.962500	35.063600	203.500000	7.9

Key points

- `DataFrame`s and `Series` have a plethora of attributes and variables to access data
 - See Pandas `doc`, or use `tab` to explore.
- Use `.dtypes` to get the types of each column in a `DataFrame`.
- To get general statistics on the `DataFrame` you can use the `describe` method.

Break

We will take a 10-minute break before we move on to the final two lessons

