Accessing Data within Pandas

Introduction

In this lesson, we're going to dig into various methods for accessing data from our Pandas Series and DataFrames.

Objectives

You will be able to:

- Use pandas methods and attributes to access information about a dataset
- Index pandas dataframes with .loc, .iloc, and column names
- Use a boolean mask to index pandas series and dataframes

Importing pandas and the data

First, let's make sure we import `pandas as pd`.

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

To show how to access data with Pandas, let's use the `wine` dataset in the scikit-learn library. Don't worry about the code below. We're essentially just making sure you have access to the `wine` dataset.

The data contained in the wine dataset are the results of a chemical analysis of wines grown in Italy. It contains the quantities of 13 wine constituents.

In [3]:
```
from sklearn.datasets import load_wine

data = load_wine()
df = pd.DataFrame(data.data, columns=data.feature_names)
```

Great! Our data set is now stored in the variable `df`. As you know, you can look at its elements by using `df` or `print(df)`.

In [4]:
```
print(df)
```

```
     alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0      14.23        1.71  2.43               15.6      127.0           2.80
1      13.20        1.78  2.14               11.2      100.0           2.65
2      13.16        2.36  2.67               18.6      101.0           2.80
3      14.37        1.95  2.50               16.8      113.0           3.85
4      13.24        2.59  2.87               21.0      118.0           2.80
..       ...         ...   ...                ...        ...            ...
173    13.71        5.65  2.45               20.5       95.0           1.68
174    13.40        3.91  2.48               23.0      102.0           1.80
175    13.27        4.28  2.26               20.0      120.0           1.59
176    13.17        2.59  2.37               20.0      120.0           1.65
177    14.13        4.10  2.74               24.5       96.0           2.05

     flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue
\
0          3.06                  0.28             2.29             5.64  1.04
1          2.76                  0.26             1.28             4.38  1.05
2          3.24                  0.30             2.81             5.68  1.03
3          3.49                  0.24             2.18             7.80  0.86
4          2.69                  0.39             1.82             4.32  1.04
..          ...                   ...              ...              ...   ...
173        0.61                  0.52             1.06             7.70  0.64
174        0.75                  0.43             1.41             7.30  0.70
175        0.69                  0.43             1.35            10.20  0.59
176        0.68                  0.53             1.46             9.30  0.60
177        0.76                  0.56             1.35             9.20  0.61

     od280/od315_of_diluted_wines  proline
0                            3.92   1065.0
1                            3.40   1050.0
2                            3.17   1185.0
3                            3.45   1480.0
4                            2.93    735.0
..                            ...      ...
173                          1.74    740.0
174                          1.56    750.0
175                          1.56    835.0
176                          1.62    840.0
177                          1.60    560.0

[178 rows x 13 columns]
```

Now, what if you want to see only a few lines of the data, based on certain constraints? You'll learn how to access data in this lesson!

Methods and attributes to access data information

It won't be a surprise that our `df` object is a Pandas DataFrame object. Let's verify this using the `type()` function:

In [7]:
```
type(df)
```

Out[7]:
```
pandas.core.frame.DataFrame
```

There are some methods and attributes associated with Pandas objects (both DataFrames *and* series!) which make retrieving information from the data particularly easy. Some commonly used methods:

- `.head()`
- `.tail()`

And attributes:

- `.index`
- `.columns`
- `.dtypes`
- `.shape`

Some methods: `.head()`, `.tail()`, and `.info()`

By using `.head()` and `.tail()`, you can select the first n rows from your dataframe. The default n is 5, but you can change this value inside the parentheses. For example:

In [11]:
```
# First 5 rows of df
df.head()
```

Out[11]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	

In [6]:
```
# Last 3 rows of df
df.tail(3)
```

Out[6]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavano
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	

To get a concise summary of the dataframe, you can use `.info()`:

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
dtypes: float64(13)
memory usage: 18.2 KB
```

Some attributes

Using `.index`, you can access the index or row labels of the DataFrame.

In [5]:
```
df.index
```

Out[5]:
```
RangeIndex(start=0, stop=178, step=1)
```

Using `.columns`, you can access the column labels of the DataFrame.

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

Out[6]:
```
Index(['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium',
       'total_phenols', 'flavanoids', 'nonflavanoid_phenols',
       'proanthocyanins', 'color_intensity', 'hue',
       'od280/od315_of_diluted_wines', 'proline'],
      dtype='object')
```

Using `.dtypes` returns the data types of all columns in the DataFrame (compare with `.info()` !)

In [7]:
```
df.dtypes
```

Out[7]:
```
alcohol                         float64
malic_acid                      float64
ash                             float64
alcalinity_of_ash               float64
magnesium                       float64
total_phenols                   float64
flavanoids                      float64
nonflavanoid_phenols            float64
proanthocyanins                 float64
color_intensity                 float64
hue                             float64
od280/od315_of_diluted_wines    float64
proline                         float64
dtype: object
```

`.shape` returns a tuple representing the dimensionality (in `(rows, columns)`) of the DataFrame.

In [8]:
```
df.shape
```

Out[8]:
```
(178, 13)
```

Selecting DataFrame information

In the previous section, we deliberately omitted 2 very important attributes:

- `.iloc`, which is a Pandas DataFrame indexer used for integer-location based indexing / selection by position
- `.loc`, which has two use cases:

```
- Selecting by label / index
- Selecting with a boolean / conditional lookup
```

`.iloc`

You can use `.iloc` to select single rows. To select the 4th row, you can use `.iloc[3]` like:

In [12]:
```
df.iloc[3]
```

Out[12]:
```
alcohol                         14.37
malic_acid                       1.95
ash                              2.50
alcalinity_of_ash               16.80
magnesium                      113.00
total_phenols                    3.85
flavanoids                       3.49
nonflavanoid_phenols             0.24
proanthocyanins                  2.18
color_intensity                  7.80
hue                              0.86
od280/od315_of_diluted_wines     3.45
proline                       1480.00
Name: 3, dtype: float64
```

You can use a colon to select several rows. Note that you'll use a structure `.iloc[a:b]` where the row with index `a` will be included in the selection and the row with index `b` is excluded.

In [13]:
```
df.iloc[5:8]
```

Out[13]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_
5	14.20	1.76	2.45	15.2	112.0	3.27	3.39	
6	14.39	1.87	2.45	14.6	96.0	2.50	2.52	
7	14.06	2.15	2.61	17.6	121.0	2.60	2.51	

Next, you can use `,` to perform *column* selections based on their index as well. The command below selects full columns 3-6:

In [14]:
```
df.iloc[:, 3:7]
```

Out[14]:

	alcalinity_of_ash	magnesium	total_phenols	flavanoids
0	15.6	127.0	2.80	3.06
1	11.2	100.0	2.65	2.76
2	18.6	101.0	2.80	3.24
3	16.8	113.0	3.85	3.49
4	21.0	118.0	2.80	2.69
5	15.2	112.0	3.27	3.39
6	14.6	96.0	2.50	2.52
7	17.6	121.0	2.60	2.51
8	14.0	97.0	2.80	2.98
9	16.0	98.0	2.98	3.15
10	18.0	105.0	2.95	3.32
11	16.8	95.0	2.20	2.43
12	16.0	89.0	2.60	2.76
13	11.4	91.0	3.10	3.69
14	12.0	102.0	3.30	3.64
15	17.2	112.0	2.85	2.91
16	20.0	120.0	2.80	3.14
17	20.0	115.0	2.95	3.40
18	16.5	108.0	3.30	3.93
19	15.2	116.0	2.70	3.03
20	16.0	126.0	3.00	3.17
21	18.6	102.0	2.41	2.41
22	16.6	101.0	2.61	2.88
23	17.8	95.0	2.48	2.37
24	20.0	96.0	2.53	2.61
25	25.0	124.0	2.63	2.68
26	16.1	93.0	2.85	2.94
27	17.0	94.0	2.40	2.19
28	19.4	107.0	2.95	2.97
29	16.0	96.0	2.65	2.33
...	...	...	...	...
148	21.5	92.0	1.93	0.76
149	21.5	113.0	1.41	1.39
150	24.0	123.0	1.40	1.57

	alcalinity_of_ash	magnesium	total_phenols	flavanoids
151	22.0	112.0	1.48	1.36
152	25.5	116.0	2.20	1.28
153	18.5	98.0	1.80	0.83
154	20.0	103.0	1.48	0.58
155	22.0	93.0	1.74	0.63
156	19.5	89.0	1.80	0.83
157	27.0	97.0	1.90	0.58
158	25.0	98.0	2.80	1.31
159	22.5	89.0	2.60	1.10
160	21.0	88.0	2.30	0.92
161	20.0	107.0	1.83	0.56
162	22.0	106.0	1.65	0.60
163	18.5	106.0	1.39	0.70
164	22.0	90.0	1.35	0.68
165	22.5	88.0	1.28	0.47
166	23.0	111.0	1.70	0.92
167	19.5	88.0	1.48	0.66
168	24.5	105.0	1.55	0.84
169	25.0	112.0	1.98	0.96
170	19.0	96.0	1.25	0.49
171	19.5	86.0	1.39	0.51
172	20.0	91.0	1.68	0.70
173	20.5	95.0	1.68	0.61
174	23.0	102.0	1.80	0.75
175	20.0	120.0	1.59	0.69
176	20.0	120.0	1.65	0.68
177	24.5	96.0	2.05	0.76

178 rows × 4 columns

Last but not least, you can perform column and row selections at once:

In [15]:
```
df.iloc[5:10, 3:9]
```

Out[15]:

	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins
5	15.2	112.0	3.27	3.39	0.34	1.97
6	14.6	96.0	2.50	2.52	0.30	1.98
7	17.6	121.0	2.60	2.51	0.31	1.25
8	14.0	97.0	2.80	2.98	0.29	1.98
9	16.0	98.0	2.98	3.15	0.22	1.85

`.loc`

a) `.loc` label-based indexing

You can `.loc` to select columns based on their (row index and) column name. Examples:

In [16]:
```
df.loc[:, 'magnesium']
```

Out[16]:
```
0      127.0
1      100.0
2      101.0
3      113.0
4      118.0
5      112.0
6       96.0
7      121.0
8       97.0
9       98.0
10     105.0
11      95.0
12      89.0
13      91.0
14     102.0
15     112.0
16     120.0
17     115.0
18     108.0
19     116.0
20     126.0
21     102.0
22     101.0
23      95.0
24      96.0
25     124.0
26      93.0
27      94.0
28     107.0
29      96.0
       ...
148     92.0
149    113.0
150    123.0
151    112.0
152    116.0
153     98.0
154    103.0
155     93.0
156     89.0
157     97.0
158     98.0
159     89.0
160     88.0
161    107.0
162    106.0
163    106.0
164     90.0
165     88.0
166    111.0
167     88.0
168    105.0
169    112.0
170     96.0
171     86.0
```

```
172     91.0
173     95.0
174    102.0
175    120.0
176    120.0
177     96.0
Name: magnesium, Length: 178, dtype: float64
```

An alternative method here is simply calling `df['magnesium']` !

In [9]:
```
df.loc[7:16, 'magnesium']
```

Out[9]:
```
7     121.0
8      97.0
9      98.0
10    105.0
11     95.0
12     89.0
13     91.0
14    102.0
15    112.0
16    120.0
Name: magnesium, dtype: float64
```

b) boolean indexing using `.loc`

Sometimes you'd like to select certain rows in your dataset based on the value for a certain variable. Imagine you'd like to create a new DataFrame that only contains the wines with an alcohol percentage below 12. This can be done as follows:

In [10]:
```
df.loc[df['alcohol'] < 12]
```

Out[10]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavano
74	11.96	1.09	2.30	21.0	101.0	3.38	2.14	
75	11.66	1.88	1.92	16.0	97.0	1.61	1.57	
77	11.84	2.89	2.23	18.0	112.0	1.72	1.32	
84	11.84	0.89	2.58	18.0	94.0	2.20	2.21	
87	11.65	1.67	2.62	26.0	88.0	1.92	1.61	
88	11.64	2.06	2.46	21.6	84.0	1.95	1.69	
94	11.62	1.99	2.28	18.0	98.0	3.02	2.26	
96	11.81	2.12	2.74	21.5	134.0	1.60	0.99	
103	11.82	1.72	1.88	19.5	86.0	2.50	1.64	
109	11.61	1.35	2.70	20.0	94.0	2.74	2.92	
110	11.46	3.74	1.82	19.5	107.0	3.18	2.58	
112	11.76	2.68	2.92	20.0	103.0	1.75	2.03	
113	11.41	0.74	2.50	21.0	88.0	2.48	2.01	
115	11.03	1.51	2.20	21.5	85.0	2.46	2.17	
116	11.82	1.47	1.99	20.8	86.0	1.98	1.60	
120	11.45	2.40	2.42	20.0	96.0	2.90	2.79	
121	11.56	2.05	3.23	28.5	119.0	3.18	5.08	
124	11.87	4.31	2.39	21.0	82.0	2.86	3.03	
127	11.79	2.13	2.78	28.5	92.0	2.13	2.24	

In [13]:
```
df[df['alcohol'] < 12]
```

Out[13]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavano
74	11.96	1.09	2.30	21.0	101.0	3.38	2.14	
75	11.66	1.88	1.92	16.0	97.0	1.61	1.57	
77	11.84	2.89	2.23	18.0	112.0	1.72	1.32	
84	11.84	0.89	2.58	18.0	94.0	2.20	2.21	
87	11.65	1.67	2.62	26.0	88.0	1.92	1.61	
88	11.64	2.06	2.46	21.6	84.0	1.95	1.69	
94	11.62	1.99	2.28	18.0	98.0	3.02	2.26	
96	11.81	2.12	2.74	21.5	134.0	1.60	0.99	
103	11.82	1.72	1.88	19.5	86.0	2.50	1.64	
109	11.61	1.35	2.70	20.0	94.0	2.74	2.92	
110	11.46	3.74	1.82	19.5	107.0	3.18	2.58	
112	11.76	2.68	2.92	20.0	103.0	1.75	2.03	
113	11.41	0.74	2.50	21.0	88.0	2.48	2.01	
115	11.03	1.51	2.20	21.5	85.0	2.46	2.17	
116	11.82	1.47	1.99	20.8	86.0	1.98	1.60	
120	11.45	2.40	2.42	20.0	96.0	2.90	2.79	
121	11.56	2.05	3.23	28.5	119.0	3.18	5.08	
124	11.87	4.31	2.39	21.0	82.0	2.86	3.03	
127	11.79	2.13	2.78	28.5	92.0	2.13	2.24	

You can verify that simply using `df[df['alcohol'] < 12]`, you can obtain the same result!

However, the `.loc` attribute is useful if you'd only want the color intensity for the wines with an alcohol percentage below 12. You can obtain the result as follows:

In [17]:
```
df.loc[df['alcohol'] < 12, ['color_intensity']]
```

Out[17]:

	color_intensity
74	3.21
75	3.80
77	2.65
84	3.05
87	2.60
88	2.80
94	3.25
96	2.50
103	2.06
109	2.65
110	2.90
112	3.80
113	3.08
115	1.90
116	1.95
120	3.25
121	6.00
124	2.80
127	3.00

Selectors for series

Until now we've only really discussed Pandas DataFrames. Most of these methods and selectors are also applicable to Pandas Series. See how you can convert a one-column DataFrame into a Pandas Series:

In [18]:
```
# Let's save our color intensity dataframe into an object col_intensity
col_intensity = df['color_intensity']
```

In [19]:
```
type(col_intensity)
```

Out[19]:
```
pandas.core.series.Series
```

Note how `col_intensity` is now a Pandas *Series*.

Many of the commands discussed before are readily applicable to series:

In [20]:
```
col_intensity[0:3]
```

Out[20]:
```
0    5.64
1    4.38
2    5.68
Name: color_intensity, dtype: float64
```

In [21]:
```
# Or col_intensity.loc[col_intensity > 8]
col_intensity[col_intensity > 8]
```

Out[21]:
```
18      8.700000
49      8.900000
144     8.210000
148     8.420000
149     9.400000
150     8.600000
151    10.000000
153    10.000000
156     9.010000
158    10.000000
159    10.000000
164     9.580000
166    10.000000
167    10.000000
168     8.660000
169     8.500000
171     9.899999
172     9.700000
175    10.000000
176     9.300000
177     9.200000
Name: color_intensity, dtype: float64
```

Changing and setting values in DataFrames and series

Changing values

Imagine that for some reason, you're not interested in the color intensity values for color intensities above 10, and simply want to set all color intensities to 10 when they are bigger than 10. You can use a selector method and then assign it a new value, just like this:

In [23]:
```
df.loc[df['color_intensity'] > 10, 'color_intensity'] = 10
```

Creating new columns

Now imagine that we want to create a new column named, "shade" which has a value, "light" when the `color_intensity` is below 7, and, "dark" when the intensity is > 7. This can be done as follows:

In [24]:
```
df.loc[df['color_intensity'] > 7, 'shade'] = 'dark'
df.loc[df['color_intensity'] <= 7, 'shade'] = 'light'
```

If you now look at the output of `df.shape`, you will notice that `df` now has 14 columns.

In [11]:
```
df.shape
```

Out[11]:
```
(178, 13)
```

Summary

We've introduced a range of techniques for accessing information in Pandas Series and DataFrames, selecting rows and columns, changing values, and creating new columns! Now, it's time for some practice! Let's start working on a lab where you will get a chance to practice some of these methods!