AN7914 Week 05 Python

February 26, 2024

1 Week 5 Python

1.1 Introduction to Pandas

Pandas is a package built on top of **NumPy**, and provides an efficient implementation of a `DataFrame`. `DataFrame`s are essentially multidimensional *arrays* with attached row and column labels, and often with heterogeneous types and/or missing data.

1.2 Installing Pandas

Installation of Pandas on your system requires NumPy to be installed. Details on this installation can be found in the Pandas documentation. Once Pandas is installed, you can import it and check the version:

`import pandas`

`pd.__version__`

We will however use an alias to call pandas. So when importing we do the following:

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

In the code above we imported `pandas` under the alias `pd`. Now let's check the version again, but this we will use the alias.

[2]:
```
pd.__version__
```

[2]:
```
'1.5.2'
```

1.3 Creating data

There are two core objects in pandas: the `DataFrame` and the `Series`.

1.4 `DataFrame`

A `DataFrame` is a table. It contains an array of individual entries, each of which has a certain value. Each entry corresponds to a *row* (or record) and a *column*.

For example, consider the following simple DataFrame:

[3]:
```
pd.DataFrame({'Yup':[50,21,32], 'Nope':[131,2,200]})
```

[3]:
```
   Yup  Nope
0   50   131
1   21     2
2   32   200
```

`DataFrame` entries are not limited to integers. For instance, here's a `DataFrame` whose values are strings:

[4]:
```python
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], 'Sue': ['Pretty good.',␣
 ↪'Bland.']})
```

[4]:
```
             Bob           Sue
0    I liked it.  Pretty good.
1  It was awful.        Bland.
```

We are using the `pd.DataFrame()` constructor to generate these DataFrame objects. The syntax for declaring a new one is a **dictionary** whose keys are the *column* names (Bob and Sue in this example), and whose *values* are a list of entries.

The dictionary-list constructor assigns values to the column labels, but just uses an ascending count from 0 (0, 1, 2, 3, ...) for the row labels. Sometimes this is OK, but oftentimes we will want to assign these labels ourselves.

[5]:
```python
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'],
              'Sue': ['Pretty good.', 'Bland.']},
             index=['Product A', 'Product B'])
```

[5]:
```
                     Bob           Sue
Product A    I liked it.  Pretty good.
Product B  It was awful.        Bland.
```

1.5 Series

A `Series`, by contrast, is a sequence of data values. If a `DataFrame` is a table, a `Series` is a **list**. And in fact you can create one with nothing more than a list:

[6]:
```python
pd.Series([1,2,3,4])
```

[6]:
```
0    1
1    2
2    3
3    4
dtype: int64
```

A `Series` is, in essence, a single **column** of a `DataFrame`. So you can assign row labels to the `Series` the same way as before, using an `index` parameter. However, a `Series` does not have a column name, it only has one overall `name`:

1.6 Importing data sets

You will need to use and import data sets from the internet or from your hard-drive. So if you want to import a **csv** file you will need to use `pd.read_csv()` command. The argument in the command could be the location where the file is stored in your computer or it could be a file store in the internet as show below.

[7]:
```
employee=pd.read_csv('/Users/sakibanwar/Downloads/updated_employee_dataset.csv')
```

We use `pd.read_csv()` to read a file stored in `'/Users/sakibanwar/Downloads/updated_employee_dataset.csv`
Then we store this dataset as a `dataframe` in `employee`.

Now lets take a look at the dataset. We can simply type `employee` the name of the `dataframe`. It will usually not going to show all columns and rows.

[8]:
```
employee
```

[8]:
```
    Employee_ID Department  Years_of_Experience  Full_Time Performance_Score  \
0             1    Finance                 1.82      False              Good
1             2         IT                 2.09      False              Poor
2             3         IT                13.24      False         Excellent
3             4    Finance                 1.55      False         Excellent
4             5  Marketing                 4.16       True           Average
5             6    Finance                 7.42       True         Excellent
6             7  Marketing                10.42       True              Good
7             8  Marketing                12.90       True              Good
8             9  Marketing                10.00       True              Poor
9            10         HR                18.74      False           Average
10           11    Finance                 6.17      False              Poor
11           12         IT                11.26       True           Average
12           13         HR                19.10       True              Good
13           14         IT                 6.46       True         Excellent
14           15    Finance                14.47       True              Good
15           16         IT                 7.40       True           Average
16           17         IT                 4.56      False              Poor
17           18         HR                12.48       True              Poor
18           19         HR                 3.77       True         Excellent
19           20         HR                 4.86       True         Excellent
20           21         IT                15.91      False           Average
21           22         IT                15.97       True              Poor
22           23  Marketing                 8.06      False         Excellent
23           24    Finance                18.83       True           Average
24           25  Marketing                11.52       True         Excellent
25           26    Finance                 8.55       True         Excellent
26           27         HR                10.74       True              Good
27           28         HR                 5.13      False           Average
28           29    Finance                 8.09      False              Good
29           30    Finance                 8.82       True         Excellent
30           31         IT                19.08       True              Good
```

31	32	Finance	2.08	False	Poor
32	33	HR	3.65	False	Poor
33	34	HR	15.79	True	Poor
34	35	Finance	15.13	True	Poor
35	36	IT	7.27	False	Average
36	37	IT	13.26	False	Poor
37	38	IT	19.42	False	Good
38	39	Finance	16.00	False	Good
39	40	HR	11.64	True	Excellent
40	41	Finance	1.12	True	Average
41	42	Marketing	14.48	False	Good
42	43	Marketing	15.61	True	Poor
43	44	HR	2.28	False	Average
44	45	Finance	11.35	True	Excellent
45	46	Finance	14.65	True	Good
46	47	IT	7.38	True	Average
47	48	IT	16.16	True	Excellent
48	49	IT	5.72	True	Average
49	50	IT	4.32	True	Good

	Salary	First_Name	Last_Name
0	54550.0	Michael	Davis
1	55225.0	Karen	Brown
2	83100.0	Joseph	Johnson
3	53875.0	David	Garcia
4	60400.0	Linda	Martinez
5	68550.0	Michael	Brown
6	76050.0	Charles	Moore
7	82250.0	David	Lopez
8	75000.0	David	Johnson
9	96850.0	Patricia	Martinez
10	65425.0	Linda	Brown
11	78150.0	John	Wilson
12	97750.0	Mary	Thomas
13	66150.0	Barbara	Anderson
14	86175.0	Jennifer	Davis
15	68500.0	Mary	Wilson
16	61400.0	James	Davis
17	81200.0	Barbara	Thomas
18	59425.0	Barbara	Gonzalez
19	62150.0	Thomas	Moore
20	89775.0	Elizabeth	Anderson
21	89925.0	Jessica	Gonzalez
22	70150.0	Joseph	Rodriguez
23	97075.0	Joseph	Anderson
24	78800.0	Charles	Gonzalez
25	71375.0	Barbara	Smith

```
26  76850.0      Karen     Miller
27  62825.0       John  Rodriguez
28  70225.0     Robert      Smith
29  72050.0    Charles      Lopez
30  97700.0    Michael      Davis
31  55200.0    William  Hernandez
32  59125.0    Michael    Jackson
33  89475.0      Sarah     Taylor
34  87825.0   Patricia      Davis
35  68175.0      Susan   Williams
36  83150.0      Sarah   Williams
37  98550.0    William      Smith
38  90000.0       Mary      Jones
39  79100.0      Karen   Martinez
40  52800.0     Joseph     Miller
41  86200.0    Michael  Rodriguez
42  89025.0    Barbara     Miller
43  55700.0      Linda  Rodriguez
44  78375.0     Joseph      Davis
45  86625.0       John      Lopez
46  68450.0      Susan    Johnson
47  90400.0     Thomas      Smith
48  64300.0   Patricia     Thomas
49  60800.0      Sarah      Jones
```

Now let's take a look at the first 15 rows.

[9]:
```
employee.head(15)
```

[9]:
```
    Employee_ID Department  Years_of_Experience  Full_Time Performance_Score  \
0             1    Finance                 1.82      False              Good
1             2         IT                 2.09      False              Poor
2             3         IT                13.24      False         Excellent
3             4    Finance                 1.55      False         Excellent
4             5  Marketing                 4.16       True           Average
5             6    Finance                 7.42       True         Excellent
6             7  Marketing                10.42       True              Good
7             8  Marketing                12.90       True              Good
8             9  Marketing                10.00       True              Poor
9            10         HR                18.74      False           Average
10           11    Finance                 6.17      False              Poor
11           12         IT                11.26       True           Average
12           13         HR                19.10       True              Good
13           14         IT                 6.46       True         Excellent
14           15    Finance                14.47       True              Good

     Salary First_Name Last_Name
```

```
0   54550.0    Michael     Davis
1   55225.0      Karen     Brown
2   83100.0     Joseph   Johnson
3   53875.0      David    Garcia
4   60400.0      Linda  Martinez
5   68550.0    Michael     Brown
6   76050.0    Charles     Moore
7   82250.0      David     Lopez
8   75000.0      David   Johnson
9   96850.0   Patricia  Martinez
10  65425.0      Linda     Brown
11  78150.0       John    Wilson
12  97750.0       Mary    Thomas
13  66150.0    Barbara  Anderson
14  86175.0   Jennifer     Davis
```

`employee.head(15)` gives us the first 15 rows of the dataframe. If we typed `employee.head(25)` it would show us the first 25 rows.

To see the last 10 rows we can use `employee.tail(10)`

[10]:
```
employee.tail(10)
```

[10]:
```
    Employee_ID Department  Years_of_Experience  Full_Time Performance_Score  \
40           41    Finance                 1.12       True           Average
41           42  Marketing                14.48      False              Good
42           43  Marketing                15.61       True              Poor
43           44         HR                 2.28      False           Average
44           45    Finance                11.35       True         Excellent
45           46    Finance                14.65       True              Good
46           47         IT                 7.38       True           Average
47           48         IT                16.16       True         Excellent
48           49         IT                 5.72       True           Average
49           50         IT                 4.32       True              Good

     Salary First_Name  Last_Name
40  52800.0     Joseph     Miller
41  86200.0    Michael  Rodriguez
42  89025.0    Barbara     Miller
43  55700.0      Linda  Rodriguez
44  78375.0     Joseph      Davis
45  86625.0       John      Lopez
46  68450.0      Susan    Johnson
47  90400.0     Thomas      Smith
48  64300.0   Patricia     Thomas
49  60800.0      Sarah      Jones
```

If you want to know exactly how many rows and columns the dataframe has we can simply type employee.shape

[11]:
```
employee.shape
```

[11]:
```
(50, 8)
```

We see that the output is (50, 8)– this means we have 50 rows and 8 columns.

If we want to see the names of the columns we use `employee.columns`

[12]:
```
employee.columns
```

[12]:
```
Index(['Employee_ID', 'Department', 'Years_of_Experience', 'Full_Time',
       'Performance_Score', 'Salary', 'First_Name', 'Last_Name'],
      dtype='object')
```

If we want to see a specific columns we can pass in a list –

`employee[['Performance_Score','Salary']]`. We passed in the list `['Performance_Score','Salary']`. This list contains the list of column names. The output is going to look a `dataframe` and not a `series`.

[13]:
```
employee[['Performance_Score','Salary']]
```

[13]:

	Performance_Score	Salary
0	Good	54550.0
1	Poor	55225.0
2	Excellent	83100.0
3	Excellent	53875.0
4	Average	60400.0
5	Excellent	68550.0
6	Good	76050.0
7	Good	82250.0
8	Poor	75000.0
9	Average	96850.0
10	Poor	65425.0
11	Average	78150.0
12	Good	97750.0
13	Excellent	66150.0
14	Good	86175.0
15	Average	68500.0
16	Poor	61400.0
17	Poor	81200.0
18	Excellent	59425.0
19	Excellent	62150.0
20	Average	89775.0
21	Poor	89925.0
22	Excellent	70150.0
23	Average	97075.0

```
24          Excellent  78800.0
25          Excellent  71375.0
26               Good  76850.0
27            Average  62825.0
28               Good  70225.0
29          Excellent  72050.0
30               Good  97700.0
31               Poor  55200.0
32               Poor  59125.0
33               Poor  89475.0
34               Poor  87825.0
35            Average  68175.0
36               Poor  83150.0
37               Good  98550.0
38               Good  90000.0
39          Excellent  79100.0
40            Average  52800.0
41               Good  86200.0
42               Poor  89025.0
43            Average  55700.0
44          Excellent  78375.0
45               Good  86625.0
46            Average  68450.0
47          Excellent  90400.0
48            Average  64300.0
49               Good  60800.0
```

1.7 Indexing in Pandas

Two types of indexing: 1. Index-based selection– `.iloc`

2. Label-based selection– `.loc`

1.8 Index-based selection– `.iloc`

`.iloc` is about selecting data based on its numerical position. Let's use `.iloc` to select the first row in the `hotel_vienna DataFrame`.

[14]:
```python
employee.iloc[0]
```

[14]:
```
Employee_ID                 1
Department            Finance
Years_of_Experience      1.82
Full_Time               False
Performance_Score        Good
Salary                54550.0
First_Name            Michael
Last_Name               Davis
```

```
Name: 0, dtype: object
```

This returned a **Series** object. If we want to get the second row. We do the following:

```
[15]: employee.iloc[1]
```

```
[15]: Employee_ID                  2
      Department                  IT
      Years_of_Experience       2.09
      Full_Time                False
      Performance_Score         Poor
      Salary                 55225.0
      First_Name               Karen
      Last_Name                Brown
      Name: 1, dtype: object
```

This returned a **Series** object. If we want to get the third row. We do the following:

```
[16]: employee.iloc[2]
```

```
[16]: Employee_ID                    3
      Department                    IT
      Years_of_Experience        13.24
      Full_Time                  False
      Performance_Score      Excellent
      Salary                   83100.0
      First_Name                Joseph
      Last_Name                Johnson
      Name: 2, dtype: object
```

We are going by row-indexes here! You get the idea! How about looking at multiple rows at once? We can slice rows

```
[17]: employee.iloc[0:3]
```

```
[17]:    Employee_ID Department  Years_of_Experience  Full_Time Performance_Score  \
      0            1    Finance                 1.82      False              Good
      1            2         IT                 2.09      False              Poor
      2            3         IT                13.24      False         Excellent

          Salary First_Name Last_Name
      0  54550.0    Michael     Davis
      1  55225.0      Karen     Brown
      2  83100.0     Joseph   Johnson
```

In the above we used `employee.iloc[0:3]` to slice the rows and get the rows whose index are numbered 0,1,2. Another example below:

```
[18]: employee.iloc[0:10]
```

```
[18]:    Employee_ID Department  Years_of_Experience  Full_Time Performance_Score  \
      0            1    Finance                 1.82      False              Good
      1            2         IT                 2.09      False              Poor
      2            3         IT                13.24      False         Excellent
      3            4    Finance                 1.55      False         Excellent
      4            5  Marketing                 4.16       True           Average
      5            6    Finance                 7.42       True         Excellent
      6            7  Marketing                10.42       True              Good
      7            8  Marketing                12.90       True              Good
      8            9  Marketing                10.00       True              Poor
      9           10         HR                18.74      False           Average

          Salary First_Name Last_Name
      0  54550.0    Michael     Davis
      1  55225.0      Karen     Brown
      2  83100.0     Joseph   Johnson
      3  53875.0      David    Garcia
      4  60400.0      Linda  Martinez
      5  68550.0    Michael     Brown
      6  76050.0    Charles     Moore
      7  82250.0      David     Lopez
      8  75000.0      David   Johnson
      9  96850.0   Patricia  Martinez
```

We can also pass in a list of rows.

```
[19]: employee.iloc[[1,2,39]]
```

```
[19]:     Employee_ID Department  Years_of_Experience  Full_Time Performance_Score  \
      1             2         IT                 2.09      False              Poor
      2             3         IT                13.24      False         Excellent
      39           40         HR                11.64       True         Excellent

           Salary First_Name Last_Name
      1   55225.0      Karen     Brown
      2   83100.0     Joseph   Johnson
      39  79100.0      Karen  Martinez
```

This particlarly useful if we want rows that are not in a sequence. For example if you want rows indexed 0,2,39.

1.8.1 Indexing both axes

You can mix the indexer types for the index and columns. With scalar integers. For example:

```
[20]: employee.iloc[0,0]
```

```
[20]: 1
```

In the above code we wanted to look at row index 0 and column index 0. In the example below are looking at row index 0 and column index 1.

[21]:
```python
employee.iloc[0,1]
```

[21]:
```
'Finance'
```

With lists of integers. Remember the first list is for row and second list is for column.

[22]:
```python
employee.iloc[[0,1,2],[0,1,2]]
```

[22]:
```
   Employee_ID Department  Years_of_Experience
0            1    Finance                 1.82
1            2         IT                 2.09
2            3         IT                13.24
```

We could also use range

[23]:
```python
employee.iloc[range(0,3),range(0,3)]
```

[23]:
```
   Employee_ID Department  Years_of_Experience
0            1    Finance                 1.82
1            2         IT                 2.09
2            3         IT                13.24
```

With slice objects.

[24]:
```python
employee.iloc[0:3,0:4]
```

[24]:
```
   Employee_ID Department  Years_of_Experience  Full_Time
0            1    Finance                 1.82      False
1            2         IT                 2.09      False
2            3         IT                13.24      False
```

The code selects the first three rows and the first four columns from the DataFrame named `employee` using integer-location based indexing.

1.9 Label-based selection– `.loc`

Access a group of rows and columns by label(s) or a boolean array. Single label. Note this returns the row as a Series. To see this clearly let's create a fake dataset

[25]:
```python
df1=pd.DataFrame([[1,2,2],[4,5,2],[7,5,2],[232,21,24]],
                 index=['cobra','viper','sidewinder','rattle_snake'],
                 columns=['max_speed','shield','windy']
)
```

[26]:
```python
df1
```

[26]:

	max_speed	shield	windy
cobra	1	2	2
viper	4	5	2
sidewinder	7	5	2
rattle_snake	232	21	24

Single label. Note this returns the row as a Series.

[27]:
```
df1.loc['viper']
```

[27]:
```
max_speed    4
shield       5
windy        2
Name: viper, dtype: int64
```

The code above selects the row(s) with the index label 'viper' from the DataFrame `df1` using label-based indexing.

[28]:
```
df1.loc[['viper','rattle_snake']]
```

[28]:

	max_speed	shield	windy
viper	4	5	2
rattle_snake	232	21	24

The code above selects the rows with index labels 'viper' and 'rattle_snake' from the DataFrame `df1` using label-based indexing. In this case, the labels provided in the square brackets(i.e. list) `['viper', 'rattle_snake']` represent the index labels of the rows that you want to select from df1.

[29]:
```
df1.loc['viper':'rattle_snake']
```

[29]:

	max_speed	shield	windy
viper	4	5	2
sidewinder	7	5	2
rattle_snake	232	21	24

The code above is used to select and retrieve a range of rows from a Pandas DataFrame named `df1` using label-based indexing with the `.loc[]` method. By specifying the range `'viper':'rattle_snake'`, it tells Pandas to include all rows starting from the row labeled 'viper' up to and including the row labeled 'rattle_snake'. This operation will return all rows within this range, along with all their columns, effectively slicing the DataFrame based on row labels.

[30]:
```
df1.loc['viper':'rattle_snake','shield':'windy']
```

[30]:

	shield	windy
viper	5	2
sidewinder	5	2
rattle_snake	21	24

The code above selects a subset of rows and columns from `df1` using label-based indexing with the `.loc[]` method. Specifically, it retrieves rows from 'viper' to 'rattle_snake' and columns from 'shield' to 'windy'. This means it includes all rows starting at the index labeled 'viper' up to and including the row labeled 'rattle_snake', and similarly, it includes all columns starting at 'shield' up to and including 'windy'. The result is a DataFrame that contains the specified slice of rows and columns, based on their labels.

1.10 Difference between `.loc` and `.iloc`.

`.loc[]` and `.iloc[]` are two indexing methods available in Pandas for selecting data from a DataFrame, but they cater to different types of indexing: label-based and integer-location based, respectively. The main difference between `.loc[]` and `.iloc[]` lies in how they interpret their arguments. `.loc[]` is used for label-based indexing, meaning it selects data based on data frame index or column names. For example, `df.loc['row_label', 'column_label']` retrieves the data at the specified row and column labels. On the other hand, `.iloc[]` is used for integer-location based indexing, meaning it selects data based on the integer positions of the rows and columns. So, `df.iloc[1, 2]` would retrieve the data located at the second row and third column, as indexing starts at 0. While `.loc[]` allows for more human-readable code by using explicit labels, `.iloc[]` offers a straightforward way to navigate through data based on numeric positions, making it particularly useful when the row or column labels are not known or when iterating through data sequentially.

```
[ ]:
```