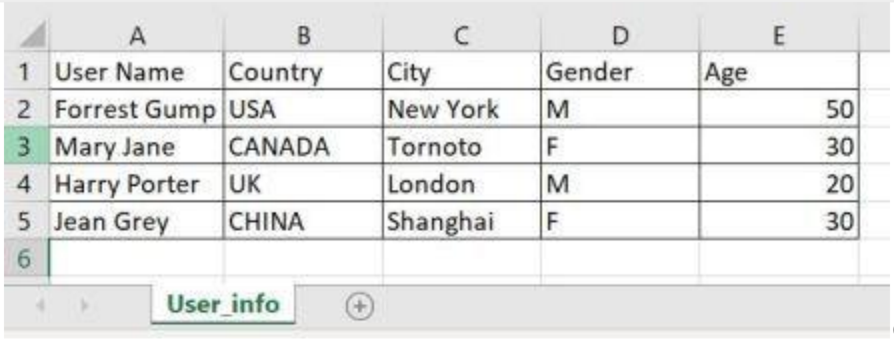
# Get values, rows and columns in pandas dataframe

AUGUST 18, 2020 JAY

This article is part of the Transition from Excel to Python series. We have walked through the data i/o (reading and saving files) part. Let's move on to something more interesting. In Excel, we can see the rows, columns, and cells. We can reference the values by using a "=" sign or within a formula. In Python, the data is stored in computer memory (i.e., not directly visible to the users), luckily the pandas library provides easy ways to get values, rows, and columns.

Let's first create a dataframe so we have something to work with. We'll use this example file from before, and we can open the Excel file on the side for reference.

```
import pandas as pd



df = pd.read_excel('users.xlsx')



>>> df

      User Name Country      City Gender  Age

0  Forrest Gump     USA  New York      M   50

1     Mary Jane  CANADA   Tornoto      F   30

2  Harry Porter      UK    London      M   20

3     Jean Grey   CHINA  Shanghai      F   30
```



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | User Name | Country | City | Gender | Age |
| 2 | Forrest Gump | USA | New York | M | 50 |
| 3 | Mary Jane | CANADA | Tornoto | F | 30 |
| 4 | Harry Porter | UK | London | M | 20 |
| 5 | Jean Grey | CHINA | Shanghai | F | 30 |
| 6 | | | | | |

User_info

excel_sheet_example

Some observations about this small table/dataframe:

- There are five columns with names: "User Name", "Country", "City", "Gender", "Age"
- There are 4 rows (excluding the header row)

`df.index` returns the list of the index, in our case, it's just integers 0, 1, 2, 3.

`df.columns` gives the list of the column (header) names.

`df.shape` shows the dimension of the dataframe, in this case it's 4 rows by 5 columns.

```
>>> df.index

RangeIndex(start=0, stop=4, step=1)



>>> df.columns

Index(['User Name', 'Country', 'City', 'Gender', 'Age'], dtype='object')



>>> df.shape

(4, 5)
```

## pandas get columns

There are several ways to get columns in pandas. Each method has its pros and cons, so I would use them differently based on the situation.

### The dot notation

We can type `df.Country` to get the "Country" column. This is a quick and easy way to get columns. However, if the column name contains space, such as "User Name". This method will not work.

```
>>> df.Country

0       USA

1    CANADA

2        UK

3     CHINA

Name: Country, dtype: object



>>> df.Age
```

```
0    50

1    30

2    20

3    30

Name: Age, dtype: int64



>>> df.User Name

SyntaxError: invalid syntax
```

## Square brackets notation

[illegible]

```
>>> df['User Name']



0    Forrest Gump

1       Mary Jane

2    Harry Porter

3       Jean Grey

Name: User Name, dtype: object





>>> df['City']



0    New York

1     Tornoto

2      London

3    Shanghai
```

```
Name: City, dtype: object
```

### Get multiple columns

The square bracket notation makes getting multiple columns easy. The syntax is similar, but instead, we pass a list of strings into the square brackets. Pay attention to the double square brackets:

**`dataframe[ [column name 1, column name 2, column name 3, ... ] ]`**

```
>>> df[['User Name', 'Age', 'Gender']]



      User Name  Age Gender

0  Forrest Gump   50      M

1     Mary Jane   30      F

2  Harry Porter   20      M

3     Jean Grey   30      F
```

## pandas get rows

We can use `.loc[]` to get rows. Note the square brackets here instead of the parenthesis (). The syntax is like this: `df.loc[row, column]`. column is optional, and if left blank, we can get the entire row. Because Python uses a zero-based index, `df.loc[0]` returns the first row of the dataframe.

### Get one row

```
>>> df.loc[0]



User Name    Forrest Gump

Country               USA

City             New York

Gender                  M

Age                    50

Name: 0, dtype: object
```

```
>>> df.loc[2]



User Name    Harry Porter

Country                UK

City               London

Gender                  M

Age                    20

Name: 2, dtype: object
```

### Get multiple rows

We'll have to use indexing/slicing to get multiple rows. In pandas, this is done similar to how to index/slice a Python list.

To get the first three rows, we can do the following:

```
>>> df.loc[0:2]



      User Name Country      City Gender  Age

0  Forrest Gump     USA  New York      M   50

1     Mary Jane  CANADA   Tornoto      F   30

2  Harry Porter      UK    London      M   20
```

## pandas get cell values

To get individual cell values, we need to use the intersection of rows and columns. Think about how we reference cells within Excel, like a cell "C10", or a range "C10:E20". The follow two approaches both follow this row & column idea.

### Square brackets notation

Using the square brackets notation, the syntax is like this: `dataframe[column name][row index]`. This is sometimes called chained indexing. An easier way to remember this notation is: `dataframe[column name]` gives a column, then adding another `[row index]` will give the specific item from that column.

Let’s say we want to get the City for Mary Jane (on row 2).

```
>>> df['City'][1]

'Tornoto'
```

To get the 2nd and the 4th row, and only the User Name, Gender and Age columns, we can pass the rows and columns as two lists like the below.

```
>>> df[['User Name', 'Age', 'Gender']].loc[[1,3]]



   User Name  Age Gender

1  Mary Jane   30      F

3  Jean Grey   30      F
```

Remember, `df[['User Name', 'Age', 'Gender']]` returns a new dataframe with only three columns. Then `.loc[[1,3]]` returns the 1st and 4th rows of that dataframe.

## .loc[] method

As previously mentioned, the syntax for .loc is `df.loc[row, column]`. Need a reminder on what are the possible values for rows (index) and columns?

```
>>> df.index

RangeIndex(start=0, stop=4, step=1)

>>> df.columns

Index(['User Name', 'Country', 'City', 'Gender', 'Age'], dtype='object')
```

Let’s try to get the country name for Harry Porter, who’s on row 3.

```
>>> df.loc[2,'Country']

'UK'
```

To get the 2nd and the 4th row, and only the User Name, Gender and Age columns, we can pass the rows and columns as two lists into the “row” and “column” positional arguments.

```
>>> df.loc[[1,3],['User Name', 'Age', 'Gender']]
```

```
   User Name  Age Gender

1  Mary Jane   30      F

3  Jean Grey   30      F
```

Source: https://pythoninoffice.com/get-values-rows-and-columns-in-pandas-dataframe/