

Using iloc & loc to select rows and columns in Pandas DataFrames

Pandas Data Selection

There are [multiple ways to select](#) and index rows and columns from [PandasDataFrames](#). I find tutorials online focusing on advanced selections of row and column choices a little complex for my requirements.

Selection Options

There's three main options to achieve the selection and indexing activities in Pandas, which can be confusing. The three selection cases and methods covered in this post are:

1. [Selecting data by row numbers \(.iloc\)](#)
2. [Selecting data by label or by a conditional statement \(.loc\)](#)
3. [Selecting in a hybrid approach \(.ix\) \(now Deprecated in Pandas 0.20.1\)](#)

Data Setup

This blog post, [inspired by other tutorials](#), describes selection activities with these operations. The tutorial is suited for the general data science situation where, typically I find myself:

1. Each row in your data frame represents a data sample.
2. Each column is a variable, and is usually named. I rarely select columns without their names.
3. I need to quickly and often select relevant rows from the data frame for modelling and visualisation activities.

For the uninitiated, the [Pandas](#) library for Python provides high-performance, easy-to-use data structures and data analysis tools for handling tabular data in “series” and in “data frames”. It's brilliant at making your data processing easier and I've written before about [grouping and summarising data](#) with Pandas.

Python Pandas Selections and Indexing

.iloc selections - position based selection

`data.iloc[<row selection>, <column selection>]`

Integer list of rows: [0,1,2]

Slice of rows: [4:7]

Single values: 1

Integer list of columns: [0,1,2]

Slice of columns: [4:7]

Single column selections: 1

loc selections - position based selection

`data.loc[<row selection>, <column selection>]`

Index/Label value: 'john'

List of labels: ['john', 'sarah']

Logical/Boolean index: data['age'] == 10

Named column: 'first_name'

List of column names: ['first_name', 'age']

Slice of columns: 'first_name':'address'

Summary of iloc and loc methods discussed in this blog post. iloc and loc are operations for retrieving data from Pandas dataframes.

Selection and Indexing Methods for Pandas DataFrames

For these explorations we'll need some sample data – I downloaded the uk-500 sample data set from www.briandunning.com. This data contains artificial names, addresses, companies and phone numbers for fictitious UK characters. To follow along, you can download the .csv file [here](#). Load the data as follows (the diagrams here come from a [Jupyter notebook](#) in the [Anaconda Python](#) install):

```
import pandas as pd
import random

# read the data from the downloaded CSV file.
data = pd.read_csv('https://s3-eu-west-1.amazonaws.com/shanebucket/downloads/uk-500.csv')

# set a numeric id for use as an index for examples.
data['id'] = [random.randint(0,1000) for x in range(data.shape[0])]
```

data.head(5)

view rawPandas Index - Loading Data.py hosted with by [GitHub](#)

In [124]: data.head(5)

Out[124]:

	first_name	last_name	company_name	address	city	county	postal	phone1	phone2	email	web
0	Aleshia	Tomkiewicz	Alan D Rosenburg Cpa Pc	14 Taylor St	St. Stephens Ward	Kent	CT2 7PP	01835- 703597	01944- 369967	atomkiewicz@hotmail.com	http://www
1	Evan	Zigomalas	Cap Gemini America	5 Binney St	Abbey Ward	Buckinghamshire	HP11 2AX	01937- 864715	01714- 737668	evan.zigomalas@gmail.com	http://www
2	France	Andrade	Elliott, John W Esq	8 Moor Place	East Southbourne and Tuckton W	Bournemouth	BH6 3BE	01347- 368222	01935- 821636	france.andrade@hotmail.com	http://www
3	Ulysses	Mcwalters	Mcmahan, Ben L	505 Exeter Rd	Hawerby cum Beesby	Lincolnshire	DN36 5RP	01912- 771311	01302- 601380	ulysses@hotmail.com	http://www
4	Tyisha	Veness	Champagne Room	5396 Forth Street	Greets Green and Lyng Ward	West Midlands	B70 9DT	01547- 429341	01290- 367248	tyisha.veness@hotmail.com	http://www

Example data loaded from CSV file.

1. Selecting pandas data using “iloc”

The [iloc](#) indexer for Pandas Dataframe is used for [integer-location based indexing / selection](#) by position.

The iloc indexer syntax is data.iloc[<row selection>, <column selection>], which is sure to be a source of confusion for R users. “iloc” in pandas is used to **select rows and columns by number**, in the order that they appear in the data frame. You can imagine that each row has a row number from 0 to the total rows (data.shape[0]) and iloc[] allows selections based on these numbers. The same applies for columns (ranging from 0 to data.shape[1])

There are two “arguments” to iloc – a row selector, and a column selector. For example:

Single selections using iloc and DataFrame

Rows:

data.iloc[0] # first row of data frame (Aleshia Tomkiewicz) - Note a Series data type output.

data.iloc[1] # second row of data frame (Evan Zigomalas)

data.iloc[-1] # last row of data frame (Mi Richan)

Columns:

data.iloc[:,0] # first column of data frame (first_name)

```
data.iloc[:,1] # second column of data frame (last_name)
```

```
data.iloc[:, -1] # last column of data frame (id)
```

[view rawPandas Index - Single iloc selections.py](#) hosted with [by GitHub](#)

Multiple columns and rows can be selected together using the .iloc indexer.

```
# Multiple row and column selections using iloc and DataFrame
```

```
data.iloc[0:5] # first five rows of dataframe
```

```
data.iloc[:, 0:2] # first two columns of data frame with all rows
```

```
data.iloc[[0,3,6,24], [0,5,6]] # 1st, 4th, 7th, 25th row + 1st 6th 7th columns.
```

```
data.iloc[0:5, 5:8] # first 5 rows and 5th, 6th, 7th columns of data frame (county -> phone1).
```

[view rawPandas Index - Multi iloc selections.py](#) hosted with [by GitHub](#)

There's two gotchas to remember when using iloc in this manner:

1. Note that .iloc returns a Pandas Series when one row is selected, and a Pandas DataFrame when multiple rows are selected, or if any column in full is selected. To counter this, pass a single-valued list if you require DataFrame output.

```
In [65]: print type(data.iloc[100])           # result of type series because only one row selected
         print type(data.iloc[[100]])         # result of type DataFrame because list selection used
         print type(data.iloc[2:10])          # result of type dataframe since there are two rows selected
         print type(data.iloc[1:2, 3])        # Series result because only one column selected
         print type(data.iloc[1:2, [3]])      # DataFrame result with one column be only one column selected
         print type(data.iloc[1:2, 3:6])      # DataFrame results because multiple columns and multiple rows.

<class 'pandas.core.series.Series'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

When using .loc, or .iloc, you can control the output format by passing lists or single values to the selectors.

2. When selecting multiple columns or multiple rows in this manner, remember that in your selection e.g.[1:5], the rows/columns selected will run from the first number to *one minus* the second number. e.g. [1:5] will go 1,2,3,4., [x,y] goes from x to y-1.

In practice, I rarely use the iloc indexer, unless I want the first (.iloc[0]) or the last (.iloc[-1]) row of the data frame.

2. Selecting pandas data using “loc”

The Pandas [loc](#) indexer can be used with DataFrames for two different use cases:

- a.) Selecting [rows by label/index](#)
- b.) Selecting rows with a [boolean / conditional lookup](#)

The loc indexer is used with the same syntax as iloc: `data.loc[<row selection>, <column selection>]`.

2a. Label-based / Index-based indexing using .loc

Selections using the loc method are based on the index of the data frame (if any). Where the index is set on a DataFrame, using `df.set_index()`, the .loc method directly selects based on index values of any rows. For example, setting the index of our test data frame to the persons “last_name”:

```
data.set_index("last_name", inplace=True)
data.head()
```

view raw [Pandas Index - Setting index for iloc.py](#) hosted with [by GitHub](#)

In [10]:	data.head()									
Out[10]:		first_name	company_name	address	city	county	postal	phone1	phone2	email
	last_name									
	Tomkiewicz	Aleshia	Alan D Rosenburg Cpa Pc	14 Taylor St	St. Stephens Ward	Kent	CT2 7PP	01835- 703597	01944- 369967	atomkiewicz@hotmail.com
	Zigomalas	Evan	Cap Gemini America	5 Binney St	Abbey Ward	Buckinghamshire	HP11 2AX	01937- 864715	01714- 737668	evan.zigomalas@gmail.com
	Andrade	France	Elliott, John W Esq	8 Moor Place	East Southbourne and Tuckton W	Bournemouth	BH6 3BE	01347- 368222	01935- 821636	france.andrade@hotmail.com
	Mcwalters	Ulysses	Mcmahan, Ben L	505 Exeter Rd	Hawerby cum Beesby	Lincolnshire	DN36 5RP	01912- 771311	01302- 601380	ulysses@hotmail.com
	Veness	Tyisha	Champagne Room	5396 Forth Street	Greets Green and Lyng Ward	West Midlands	B70 9DT	01547- 429341	01290- 367248	tyisha.veness@hotmail.com

Last Name set as Index set on sample data frameNow with the index set, we can directly select rows for different “last_name” values using `.loc[<label>]` – either singly, or in multiples. For example:

```
In [11]: data.loc['Andrade']
```

```
Out[11]: first_name          France
company_name      Elliott, John W Esq
address           8 Moor Place
city              East Southbourne and Tuckton W
county            Bournemouth
postal            BH6 3BE
phone1            01347-368222
phone2            01935-821636
email             france.andrade@hotmail.com
web               http://www.elliottjohnwesq.co.uk
id                487
Name: Andrade, dtype: object
```

```
In [12]: data.loc[['Andrade', 'Veness']]
```

```
Out[12]:
```

	first_name	company_name	address	city	county	postal	phone1	phon
last_name								
Andrade	France	Elliott, John W Esq	8 Moor Place	East Southbourne and Tuckton W	Bournemouth	BH6 3BE	01347-368222	01935-821636
Veness	Tyisha	Champagne Room	5396 Forth Street	Greets Green and Lyng Ward	West Midlands	B70 9DT	01547-429341	01293-36724

Selecting single or multiple rows using .loc index selections with pandas. Note that the first example returns a series, and the second returns a DataFrame. You can achieve a single-column DataFrame by passing a single-element list to the .loc operation.

Select columns with .loc using the names of the columns. In most of my data work, typically I have named columns, and use these named selections.

```
In [46]: data.loc[['Andrade', 'Veness'], ['first_name', 'address', 'city']]
```

```
Out[46]:
```

	first_name	address	city
last_name			
Andrade	France	8 Moor Place	East Southbourne and Tuckton W
Veness	Tyisha	5396 Forth Street	Greets Green and Lyng Ward

When using the .loc indexer, columns are referred to by names using lists of strings, or “:” slices.

You can select ranges of index labels – the selection `data.loc['Bruch':'Julio']` will return all rows in the data frame between the index entries for “Bruch” and “Julio”. The following examples should now make sense:




```

# Select rows with index values 'Andrade' and 'Veness', with all columns between 'city' and 'email'
data.loc[['Andrade', 'Veness'], 'city':'email']

# Select same rows, with just 'first_name', 'address' and 'city' columns
data.loc['Andrade':'Veness', ['first_name', 'address', 'city']]

# Change the index to be based on the 'id' column
data.set_index('id', inplace=True)

# select the row with 'id' = 487
data.loc[487]

```

[view rawPandas Index - Select rows with loc.py](#) hosted with [by GitHub](#)

Note that in the last example, `data.loc[487]` **(the row with index value 487) is not equal to** `data.iloc[487]` **(the 487th row in the data)**. The index of the DataFrame can be out of numeric order, and/or a string or multi-value.

2b. Boolean / Logical indexing using `.loc`

[Conditional selections with boolean arrays](#) using `data.loc[<selection>]` is the most common method that I use with Pandas DataFrames. With boolean indexing or logical selection, you pass an array or [Series](#) of True/False values to the `.loc` indexer to select the rows where your Series has *True* values.

In most use cases, you will make selections based on the values of different columns in your data set.

For example, the statement `data['first_name'] == 'Antonio'` produces a Pandas Series with a True/False value for every row in the 'data' DataFrame, where there are "True" values for the rows where the `first_name` is "Antonio". These type of boolean arrays can be passed directly to the `.loc` indexer as so:

```
In [26]: data.loc[data['first_name'] == 'Antonio']
```

```
Out[26]:
```

	first_name	company_name	address	city	county	postal	phone1	phone2	email	web
last_name										
Villamarin	Antonio	Combs Sheetmetal	353 Standish St #8264	Little Parndon and Hare Street	Hertfordshire	CM20 2HT	01559-403415	01388-777812	antonio.villamarin@gmail.com	http://www.com
Glasford	Antonio	Saint Thomas Creations	425 Howley St	Gaer Community	Newport	NP20 3DE	01463-409090	01242-318420	antonio_glasford@glasford.co.uk	http://www.sain
Heilig	Antonio	Radisson Suite Hotel	35 Elton St #3	Ipplepen	Devon	TQ12 5LL	01324-171614	01442-946357	antonio.heilig@gmail.com	http://www.radi

Using a boolean True/False series to select rows in a pandas data frame – all rows with first name of “Antonio” are selected.

As before, a second argument can be passed to .loc to select particular columns out of the data frame. Again, columns are referred to by name for the loc indexer and can be a single string, a list of columns, or a slice “:” operation.

```
In [32]: data.loc[data['first_name'] == 'Erasmus', ['company_name', 'email', 'phone1']]
```

```
Out[32]:
```

	company_name	email	phone1
last_name			
Talentino	Active Air Systems	erasmo.talentino@hotmail.com	01492-454455
Gath	Pan Optx	egath@hotmail.com	01445-796544
Rhea	Martin Morrissey	erasmo_rhea@hotmail.com	01507-386397

Selecting multiple columns with loc can be achieved by passing column names to the second argument of .loc[] Note that when selecting columns, if one column only is selected, the .loc operator returns a Series. For a single column DataFrame, use a one-element list to keep the DataFrame format, for example:


```
In [35]: data.loc[data['first_name'] == 'Antonio', 'email']
```

```
Out[35]: last_name  
Villamarin      antonio.villamarin@gmail.com  
Glasford        antonio_glasford@glasford.co.uk  
Heilig          antonio.heilig@gmail.com  
Name: email, dtype: object
```

**.loc[<selection>, String]
returns a Series**

```
In [37]: data.loc[data['first_name'] == 'Antonio', ['email']]
```

```
Out[37]:
```

	email
last_name	
Villamarin	antonio.villamarin@gmail.com
Glasford	antonio_glasford@glasford.co.uk
Heilig	antonio.heilig@gmail.com

**.loc[<selection>, List]
returns a DataFrame**

If selections of a single column are made as a string, a series is returned from .loc. Pass a list to get a DataFrame back.

Make sure you understand the following additional examples of .loc selections for clarity:

```
# Select rows with first name Antonio, # and all columns between 'city' and 'email'  
data.loc[data['first_name'] == 'Antonio', 'city':'email']  
  
# Select rows where the email column ends with 'hotmail.com', include all columns  
data.loc[data['email'].str.endswith("hotmail.com")]  
  
# Select rows with last_name equal to some values, all columns  
data.loc[data['first_name'].isin(['France', 'Tyisha', 'Eric'])]  
  
# Select rows with first name Antonio AND hotmail email addresses  
data.loc[data['email'].str.endswith("gmail.com") & (data['first_name'] == 'Antonio')]  
  
# select rows with id column between 100 and 200, and just return 'postal' and 'web' columns
```

```

data.loc[(data['id'] > 100) & (data['id'] <= 200), ['postal', 'web']]

# A lambda function that yields True/False values can also be used.
# Select rows where the company name has 4 words in it.
data.loc[data['company_name'].apply(lambda x: len(x.split(' ')) == 4)]

# Selections can be achieved outside of the main .loc for clarity:
# Form a separate variable with your selections:
idx = data['company_name'].apply(lambda x: len(x.split(' ')) == 4)
# Select only the True values in 'idx' and only the 3 columns specified:
data.loc[idx, ['email', 'first_name', 'company']]

```

Logical selections and boolean Series can also be passed to the generic [] indexer of a pandas DataFrame and will give the same results:
`data.loc[data['id'] == 9] == data[data['id'] == 9]` .

3. Selecting pandas data using ix

Note: The ix indexer has been deprecated in recent versions of Pandas, starting with version 0.20.1.

The [ix\[\] indexer](#) is a hybrid of .loc and .iloc. Generally, ix is label based and acts just as the .loc indexer. However, .ix also supports integer type selections (as in .iloc) where passed an integer. This only works where the index of the DataFrame is not integer based. ix will accept any of the inputs of .loc and .iloc.

Slightly more complex, I prefer to explicitly use .iloc and .loc to avoid unexpected results.

As an example:

```

# ix indexing works just the same as .loc when passed strings
data.ix[['Andrade']] == data.loc[['Andrade']]

```

```
# ix indexing works the same as .iloc when passed integers.  
data.ix[[33]] == data.iloc[[33]]  
  
# ix only works in both modes when the index of the DataFrame is NOT an integer itself.
```

Setting values in DataFrames using .loc

With a slight change of syntax, you can actually update your DataFrame in the same statement as you select and filter using .loc indexer. This particular pattern allows you to update values in columns depending on different conditions. The setting operation does not make a copy of the data frame, but edits the original data.

As an example:

```
# Change the first name of all rows with an ID greater than 2000 to "John"  
data.loc[data['id'] > 2000, "first_name"] = "John"  
  
# Change the first name of all rows with an ID greater than 2000 to "John"  
data.loc[data['id'] > 2000, "first_name"] = "John"
```