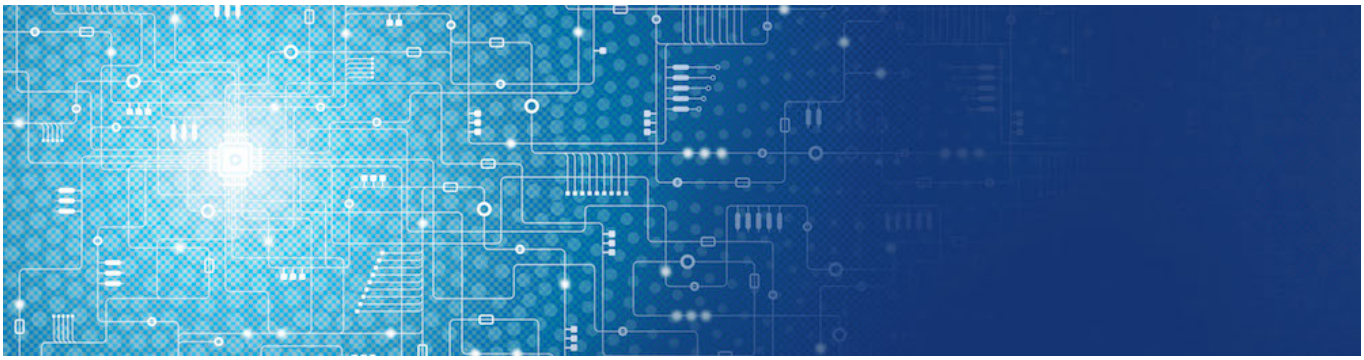
The Pandas DataFrame – loading, editing, and viewing data in Python

29 Comments / blog, data science, Data Visualisation, Pandas, python, Tutorials / By shanelynn



Starting out with Python Pandas DataFrames

If you're developing in data science, and moving from excel-based analysis to the world of Python, scripting, and automated analysis, you'll come across the incredibly popular data management library, "Pandas" in Python. Pandas development started in 2008 with main developer Wes McKinney and the library has become a standard for data analysis and management using Python. Pandas fluency is essential for any Python-based data professional, people interested in trying a Kaggle challenge, or anyone seeking to automate a data process.

The aim of this post is to help beginners get to grips with the basic data format for Pandas – the DataFrame. We will examine basic methods for creating data frames, what a DataFrame actually is, renaming and deleting data frame columns and rows, and where to go next to further your skills.

The topics in this post will enable you (hopefully) to:

1. Load your data from a file into a Python Pandas DataFrame,
2. Examine the basic statistics of the data,
3. Change some values,
4. Finally output the result to a new file.

What is a Python Pandas DataFrame?

The Pandas library documentation defines a DataFrame as a "two-dimensional, size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns)". In plain terms, think of a DataFrame as a table of data, i.e. a single set of formatted two-dimensional data, with the following characteristics:

- There can be multiple rows and columns in the data.
- Each row represents a sample of data,
- Each column contains a different variable that describes the samples (rows).

- The data in every column is usually the same type of data – e.g. numbers, strings, dates.
- Usually, unlike an excel data set, DataFrames avoid having missing values, and there are no gaps and empty values between rows or columns.

By way of example, the following data sets that would fit well in a Pandas DataFrame:

- **In a school system DataFrame** – each row could represent a single student in the school, and columns may represent the students name (string), age (number), date of birth (date), and address (string).
- **In an economics DataFrame,** each row may represent a single city or geographical area, and columns might include the the name of area (string), the population (number), the average age of the population (number), the number of households (number), the number of schools in each area (number) etc.
- **In a shop or e-commerce system DataFrame**, each row in a DataFrame may be used to represent a customer, where there are columns for the number of items purchased (number), the date of original registration (date), and the credit card number (string).

Creating Pandas DataFrames

We'll examine two methods to create a DataFrame – manually, and from comma-separated value (CSV) files.

Manually entering data

The start of every data science project will include getting useful data into an analysis environment, in this case Python. There's multiple ways to create DataFrames of data in Python, and the simplest way is through typing the data into Python manually, which obviously only works for tiny datasets.

```
In [2]:
# The convention is to import Pandas with shortcut 'pd'
import pandas as pd
import os
```

```
In [25]:
# You can create a database using a dictionary of lists.
# Each column is a dictionary key and the key becomes the column name.
# All the lists need to be the same length, and these become the rows.
new_dataframe = pd.DataFrame(
    {
        "column_1": [1,2,3,4,5],
        "another_column": ['this', 'column', 'has', 'strings', 'inside!'],
        "float_column": [0.1, 0.5, 33, 48, 42.5558],
        "binary_solo": [True, False, True, True, False]
    }
)
# you can look at your new dataframe in Jupyter, by simply typing it's name:
new_dataframe
```

Out[25]:

	another_column	binary_solo	column_1	float_column
0	this	True	1	0.1000
1	column	False	2	0.5000
2	has	True	3	33.0000
3	strings	True	4	48.0000
4	inside!	False	5	42.5558

Using Python dictionaries and lists to create DataFrames only works for small datasets that you can type out manually. There are other ways to format manually entered data which you can check out here.

Note that convention is to load the Pandas library as 'pd' (`import pandas as pd`). You'll see this notation used frequently online, and in Kaggle kernels.

Loading CSV data into Pandas

Creating DataFrames from CSV (comma-separated value) files is made extremely simple with the read_csv() function in Pandas, once you know the path to your file. A CSV file is a text file containing data in table form, where columns are separated using the ',' comma character, and rows are on separate lines (see here).

If your data is in some other form, such as an SQL database, or an Excel (XLS / XLSX) file, you can look at the other functions to read from these sources into DataFrames, namely read_xlsx, read_sql. However, for simplicity, sometimes extracting data directly to CSV and using that is preferable.

In this example, we're going to load Global Food production data from a CSV file downloaded from the Data Science competition website, Kaggle. You can download the CSV file from Kaggle, or directly from here. The data is nicely formatted, and you can open it in Excel at first to get a preview:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude	Y1961	Y1962	Y1963
2	AF	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	1928	1904	1666
3	AF	2	Afghanistan	2805	Rice (Milled Equivalent)	5142	Food	1000 tonnes	33.94	67.71	183	183	182
4	AF	2	Afghanistan	2513	Barley and products	5521	Feed	1000 tonnes	33.94	67.71	76	76	76
5	AF	2	Afghanistan	2513	Barley and products	5142	Food	1000 tonnes	33.94	67.71	237	237	237
6	AF	2	Afghanistan	2514	Maize and products	5521	Feed	1000 tonnes	33.94	67.71	210	210	214
7	AF	2	Afghanistan	2514	Maize and products	5142	Food	1000 tonnes	33.94	67.71	403	403	410
8	AF	2	Afghanistan	2517	Millet and products	5142	Food	1000 tonnes	33.94	67.71	17	18	19
9	AF	2	Afghanistan	2520	Cereals, Other	5142	Food	1000 tonnes	33.94	67.71	0	0	0
10	AF	2	Afghanistan	2531	Potatoes and products	5142	Food	1000 tonnes	33.94	67.71	111	97	103
11	AF	2	Afghanistan	2536	Sugar cane	5521	Feed	1000 tonnes	33.94	67.71	45	45	45
12	AF	2	Afghanistan	2537	Sugar beet	5521	Feed	1000 tonnes	33.94	67.71	0	0	0
13	AF	2	Afghanistan	2542	Sugar (Raw Equivalent)	5142	Food	1000 tonnes	33.94	67.71	45	41	43
14	AF	2	Afghanistan	2543	Sweeteners, Other	5142	Food	1000 tonnes	33.94	67.71	0	0	0
15	AF	2	Afghanistan	2745	Honey	5142	Food	1000 tonnes	33.94	67.71	2	2	2
16	AF	2	Afghanistan	2549	Pulses, Other and products	5521	Feed	1000 tonnes	33.94	67.71	1	1	1
17	AF	2	Afghanistan	2549	Pulses, Other and products	5142	Food	1000 tonnes	33.94	67.71	15	16	17
18	AF	2	Afghanistan	2551	Nuts and products	5142	Food	1000 tonnes	33.94	67.71	2	3	1
19	AF	2	Afghanistan	2560	Coconuts - Incl Copra	5142	Food	1000 tonnes	33.94	67.71	0	0	0
20	AF	2	Afghanistan	2561	Sesame seed	5142	Food	1000 tonnes	33.94	67.71	10	10	10
21	AF	2	Afghanistan	2563	Olives (including preserved)	5142	Food	1000 tonnes	33.94	67.71	0	0	0
22	AF	2	Afghanistan	2571	Soyabean Oil	5142	Food	1000 tonnes	33.94	67.71	0	0	0
23	AF	2	Afghanistan	2572	Groundnut Oil	5142	Food	1000 tonnes	33.94	67.71	0	0	0
24	AF	2	Afghanistan	2573	Sunflowerseed Oil	5142	Food	1000 tonnes	33.94	67.71	3	3	3
25	AF	2	Afghanistan	2574	Rape and Mustard Oil	5142	Food	1000 tonnes	33.94	67.71	0	0	0
26	AF	2	Afghanistan	2575	Cottonseed Oil	5142	Food	1000 tonnes	33.94	67.71	4	6	9
27	AF	2	Afghanistan	2577	Palm Oil	5142	Food	1000 tonnes	33.94	67.71	0	0	0
28	AF	2	Afghanistan	2579	Sesameseed Oil	5142	Food	1000 tonnes	33.94	67.71	2	2	2

The sample data for this post consists of food global production information spanning 1961 to 2013. Here the CSV file is examined in Microsoft Excel.

The sample data contains 21,478 rows of data, with each row corresponding to a food source from a specific country. The first 10 columns represent information on the sample country and food/feed type, and the remaining columns represent the food production for every year from 1963 – 2013 (63 columns in total).

If you haven't already installed Python / Pandas, I'd recommend setting up Anaconda or WinPython (these are downloadable distributions or bundles that contain Python with the top libraries pre-installed) and using Jupyter notebooks (notebooks allow you to use Python in your browser easily) for this tutorial. Some installation instructions are here.

Load the file into your Python workbook using the Pandas read_csv function like so:

```
In [17]: # Option 1: You can read data using the full path to the file you're loading.
         path_to_file = "/Users/shane/Documents/Blog/DataFrame/FAO+database.csv"
         data = pd.read_csv(path_to_file, encoding='utf-8')

         print(type(data))

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

```
In [18]: # Option 2:
         # If the file is in the same directory that you are working in - you can load it with just the filename.
         # Use os.getcwd() to 'get current working directory'.
         print("The directory we are working in is {}".format(os.getcwd()))
         data = pd.read_csv("FAO+database.csv")

         print(type(data))

         The directory we are working in is /Users/shane/Documents/Blog/DataFrame
         <class 'pandas.core.frame.DataFrame'>
```

Load CSV files into Python to create Pandas Dataframes using the read_csv function. Beginners often trip up with paths – make sure your file is in the same directory you're working in, or specify the complete path here (it'll start with C:/ if you're using Windows).

If you have path or filename issues, you'll see FileNotFoundError exceptions like this:

```
FileNotFoundError: File b'/some/directory/on/your/system/FAO+database.csv' does
 not exist
```

Preview and examine data in a Pandas DataFrame

Once you have data in Python, you'll want to see the data has loaded, and confirm that the expected columns and rows are present.

Print the data

If you're using a Jupyter notebook, outputs from simply typing in the name of the data frame will result in nicely formatted outputs. Printing is a convenient way to preview your loaded data, you can confirm that column names were imported correctly, that the data formats are as expected, and if there are missing values anywhere.

```
In [33]: data
```

Out[33]:

	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude	...	Y2004	Y2005	Y2006	Y2007	Y2008	Y2009
0	AF	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	...	3249.0	3486.0	3704.0	4164.0	4252.0	4538.0
1	AF	2	Afghanistan	2805	Rice (Milled Equivalent)	5142	Food	1000 tonnes	33.94	67.71	...	419.0	445.0	546.0	455.0	490.0	415.0
2	AF	2	Afghanistan	2513	Barley and products	5521	Feed	1000 tonnes	33.94	67.71	...	58.0	236.0	262.0	263.0	230.0	379.0
3	AF	2	Afghanistan	2513	Barley and products	5142	Food	1000 tonnes	33.94	67.71	...	185.0	43.0	44.0	48.0	62.0	55.0
4	AF	2	Afghanistan	2514	Maize and products	5521	Feed	1000 tonnes	33.94	67.71	...	120.0	208.0	233.0	249.0	247.0	195.0
5	AF	2	Afghanistan	2514	Maize and products	5142	Food	1000 tonnes	33.94	67.71	...	231.0	67.0	82.0	67.0	69.0	71.0
6	AF	2	Afghanistan	2517	Millet and products	5142	Food	1000 tonnes	33.94	67.71	...	15.0	21.0	11.0	19.0	21.0	18.0
7	AF	2	Afghanistan	2520	Cereals, Other	5142	Food	1000 tonnes	33.94	67.71	...	2.0	1.0	1.0	0.0	0.0	0.0
8	AF	2	Afghanistan	2531	Potatoes and products	5142	Food	1000 tonnes	33.94	67.71	...	276.0	294.0	294.0	260.0	242.0	250.0
9	AF	2	Afghanistan	2536	Sugar cane	5521	Feed	1000 tonnes	33.94	67.71	...	50.0	29.0	61.0	65.0	54.0	114.0
10	AF	2	Afghanistan	2537	Sugar beet	5521	Feed	1000 tonnes	33.94	67.71	...	0.0	0.0	0.0	0.0	0.0	0.0

In a Jupyter notebook, simply typing the name of a data frame will result in a neatly formatted outputs. This is an excellent way to preview data, however notes that, by default, only 100 rows will print, and 20 columns.

You'll notice that Pandas displays only 20 columns by default for wide data dataframes, and only 60 or so rows, truncating the middle section. If you'd like to change these limits, you can edit the defaults using some internal options for Pandas displays (simple use `pd.display.options.XX = value` to set these):

- pd.display.options.width – the width of the display in characters – use this if your display is wrapping rows over more than one line.
- pd.display.options.max_rows – maximum number of rows displayed.
- pd.display.options.max_columns – maximum number of columns displayed.

You can see the full set of options available in the official Pandas options and settings documentation.

DataFrame rows and columns with .shape

The shape command gives information on the data set size – 'shape' returns a tuple with the number of rows, and the

number of columns for the data in the DataFrame. Another descriptive property is the 'ndim' which gives the number of dimensions in your data, typically 2.

Get the shape of your DataFrame – the number of rows and columns using .shape, and the number of dimensions using .ndim.

Our food production data contains 21,477 rows, each with 63 columns as seen by the output of .shape. We have two dimensions – i.e. a 2D data frame with height and width. If your data had only one column, ndim would return 1. Data sets with more than two dimensions in Pandas used to be called Panels, but these formats have been deprecated. The recommended approach for multi-dimensional (>2) data is to use the Xarray Python library.

Preview DataFrames with head() and tail()

The DataFrame.head() function in Pandas, by default, shows you the top 5 rows of data in the DataFrame. The opposite is DataFrame.tail(), which gives you the last 5 rows.

Pass in a number and Pandas will print out the specified number of rows as shown in the example below. Head() and Tail() need to be core parts of your go-to Python Pandas functions for investigating your datasets.

```
In [22]: # Print out the first 5 rows of data in my dataset
         data.head()
```

Out[22]:

	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude	...	Y2004	Y2005	Y2006	Y2007	Y2008	Y2009	Y2010
0	AF	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	...	3249.0	3486.0	3704.0	4164.0	4252.0	4538.0	4605.0
1	AF	2	Afghanistan	2805	Rice (Milled Equivalent)	5142	Food	1000 tonnes	33.94	67.71	...	419.0	445.0	546.0	455.0	490.0	415.0	442.0
2	AF	2	Afghanistan	2513	Barley and products	5521	Feed	1000 tonnes	33.94	67.71	...	58.0	236.0	262.0	263.0	230.0	379.0	315.0
3	AF	2	Afghanistan	2513	Barley and products	5142	Food	1000 tonnes	33.94	67.71	...	185.0	43.0	44.0	48.0	62.0	55.0	60.0
4	AF	2	Afghanistan	2514	Maize and products	5521	Feed	1000 tonnes	33.94	67.71	...	120.0	208.0	233.0	249.0	247.0	195.0	178.0

5 rows × 63 columns

```
In [21]: # Print out 3 rows instead
         data.head(3)
```

Out[21]:

	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude	...	Y2004	Y2005	Y2006	Y2007	Y2008	Y2009	Y2010
0	AF	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	...	3249.0	3486.0	3704.0	4164.0	4252.0	4538.0	4605.0
1	AF	2	Afghanistan	2805	Rice (Milled Equivalent)	5142	Food	1000 tonnes	33.94	67.71	...	419.0	445.0	546.0	455.0	490.0	415.0	442.0
2	AF	2	Afghanistan	2513	Barley and products	5521	Feed	1000 tonnes	33.94	67.71	...	58.0	236.0	262.0	263.0	230.0	379.0	315.0

3 rows × 63 columns

```
In [20]: # Print out the last 10 rows of data in the dataset
         data.tail(10)
```

Out[20]:

	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude	...	Y2004	Y2005	Y2006	Y2007	Y2008	Y2009	Y201
21467	ZW	181	Zimbabwe	2943	Meat	5142	Food	1000 tonnes	-19.02	29.15	...	222.0	228.0	233.0	238.0	242.0	265.0	262
21468	ZW	181	Zimbabwe	2945	Offals	5142	Food	1000 tonnes	-19.02	29.15	...	20.0	20.0	21.0	21.0	21.0	21.0	21
21469	ZW	181	Zimbabwe	2946	Animal fats	5142	Food	1000 tonnes	-19.02	29.15	...	26.0	26.0	29.0	29.0	27.0	31.0	30
21470	ZW	181	Zimbabwe	2949	Eggs	5142	Food	1000 tonnes	-19.02	29.15	...	15.0	18.0	18.0	21.0	22.0	27.0	27
21471	ZW	181	Zimbabwe	2948	Milk - Excluding Butter	5521	Feed	1000 tonnes	-19.02	29.15	...	21.0	21.0	21.0	21.0	21.0	23.0	25
21472	ZW	181	Zimbabwe	2948	Milk - Excluding Butter	5142	Food	1000 tonnes	-19.02	29.15	...	373.0	357.0	359.0	356.0	341.0	385.0	418
21473	ZW	181	Zimbabwe	2960	Fish, Seafood	5521	Feed	1000 tonnes	-19.02	29.15	...	5.0	4.0	9.0	6.0	9.0	5.0	15
21474	ZW	181	Zimbabwe	2960	Fish, Seafood	5142	Food	1000 tonnes	-19.02	29.15	...	18.0	14.0	17.0	14.0	15.0	18.0	29
21475	ZW	181	Zimbabwe	2961	Aquatic Products, Other	5142	Food	1000 tonnes	-19.02	29.15	...	0.0	0.0	0.0	0.0	0.0	0.0	0

The first 5 rows of a DataFrame are shown by head(), the final 5 rows by tail(). For other numbers of rows – simply specify how many you want!

In our example here, you can see a subset of the columns in the data since there are more than 20 columns overall.

Data types (dtypes) of columns

Many DataFrames have mixed data types, that is, some columns are numbers, some are strings, and some are dates etc. Internally, CSV files do not contain information on what data types are contained in each column; all of the data is just characters. Pandas infers the data types when loading the data, e.g. if a column contains only numbers, pandas will set that column's data type to numeric: integer or float.

You can check the types of each column in our example with the '.dtypes' property of the dataframe.

```
In [15]: data.dtypes

Out[15]: Area Abbreviation     object
         Area Code              int64
         Area                  object
         Item Code              int64
         Item                  object
         Element Code           int64
         Element               object
         Unit                  object
         latitude             float64
         longitude            float64
         Y1961                float64
         Y1962                float64
         Y1963                float64
         Y1964                float64
         Y1965                float64
         Y1966                float64
         Y1967                float64
         Y1968                float64
         Y1969                float64
         Y1970                float64
         Y1971                float64
         Y1972                float64
         Y1973                float64
         Y1974                float64
         Y1975                float64
         Y1976                float64
```

See the data types of each column in your dataframe using the .dtypes property. Notes that character/string columns appear as 'object' datatypes.

In some cases, the automated inferring of data types can give unexpected results. Note that strings are loaded as 'object' datatypes, because technically, the DataFrame holds a pointer to the string data elsewhere in memory. This behaviour is expected, and can be ignored.

To change the datatype of a specific column, use the .astype() function. For example, to see the 'Item Code' column as a string, use:

```
data['Item Code'].astype(str)
```

Describing data with .describe()

Finally, to see some of the core statistics about a particular column, you can use the 'describe' function.

- For numeric columns, describe() returns basic statistics: the value count, mean, standard deviation, minimum, maximum, and 25th, 50th, and 75th quantiles for the data in a column.
- For string columns, describe() returns the value count, the number of unique entries, the most frequently occurring value ('top'), and the number of times the top value occurs ('freq')

Select a column to describe using a string inside the [] braces, and call describe() as follows:

```
In [37]: # Using Pandas to describe the 'Y2013' column - a column of numbers (integers)
         data['Y2013'].describe()

Out[37]: count     21477.000000
         mean        575.557480
         std        6218.379479
         min        -246.000000
         25%           0.000000
         50%           8.000000
         75%          90.000000
         max      489299.000000
         Name: Y2013, dtype: float64

In [35]: # Using Pandas to describe the 'Area' column - a column of strings
         data['Area'].describe()

Out[35]: count     21477
         unique      174
         top       Spain
         freq        150
         Name: Area, dtype: object
```

Use the describe() function to get basic statistics on columns in your Pandas DataFrame. Note the differences between columns with numeric datatypes, and columns of strings and characters.

Note that if describe is called on the entire DataFrame, statistics only for the columns with numeric datatypes are returned, and in DataFrame format.

```
In [34]: data.describe()
```

Out[34]:

	Area Code	Item Code	Element Code	latitude	longitude	Y1961	Y1962	Y19
count	21477.000000	21477.000000	21477.000000	21477.000000	21477.000000	17938.000000	17938.000000	17938.0000
mean	125.449411	2694.211529	5211.687154	20.450613	15.794445	195.262069	200.782250	205.4646
std	72.868149	148.973406	146.820079	24.628336	66.012104	1864.124336	1884.265591	1861.1747
min	1.000000	2511.000000	5142.000000	-40.900000	-172.100000	0.000000	0.000000	0.0000
25%	63.000000	2561.000000	5142.000000	6.430000	-11.780000	0.000000	0.000000	0.0000
50%	120.000000	2640.000000	5142.000000	20.590000	19.150000	1.000000	1.000000	1.0000
75%	188.000000	2782.000000	5142.000000	41.150000	46.870000	21.000000	22.000000	23.0000
max	276.000000	2961.000000	5521.000000	64.960000	179.410000	112227.000000	109130.000000	106356.0000

8 rows × 58 columns

Describing a full dataframe gives summary statistics for the numeric columns only, and the return format is another DataFrame.

Selecting and Manipulating Data

The data selection methods for Pandas are very flexible. In another post on this site, I've written extensively about the core selection methods in Pandas – namely iloc and loc. For detailed information and to master selection, be sure to read that post. For this example, we will look at the basic method for column and row selection.

Selecting columns

There are three main methods of selecting columns in pandas:

- using a dot notation, e.g. `data.column_name`,

- using square braces and the name of the column as a string, e.g. `data['column_name']`
- or using numeric indexing and the iloc selector `data.iloc[:, <column_number>]`

Three primary methods for selecting columns from dataframes in pandas – use the dot notation, square brackets, or iloc methods. The square brackets with column name method is the least error prone in my opinion.

When a column is selected using any of these methodologies, a pandas.Series is the resulting datatype. A pandas series is a one-dimensional set of data. It's useful to know the basic operations that can be carried out on these Series of data, including summing (`.sum()`), averaging (`.mean()`), counting (`.count()`), getting the median (`.median()`), and replacing missing values (`.fillna(new_value)`).

```
# Series summary operations.
# We are selecting the column "Y2007", and performing various calculations.
[data['Y2007'].sum(), # Total sum of the column values
 data['Y2007'].mean(), # Mean of the column values
 data['Y2007'].median(), # Median of the column values
 data['Y2007'].nunique(), # Number of unique entries
 data['Y2007'].max(), # Maximum of the column values
 data['Y2007'].min()] # Minimum of the column values

Out: [10867788.0, 508.48210358863986, 7.0, 1994, 402975.0, 0.0]
```

Selecting multiple columns at the same time extracts a new DataFrame from your existing DataFrame. For selection of multiple columns, the syntax is:

- square-brace selection with a list of column names, e.g. `data[['column_name_1', 'column_name_2']]`
- using numeric indexing with the iloc selector and a list of column numbers, e.g. `data.iloc[:, [0,1,20,22]]`

Selecting rows

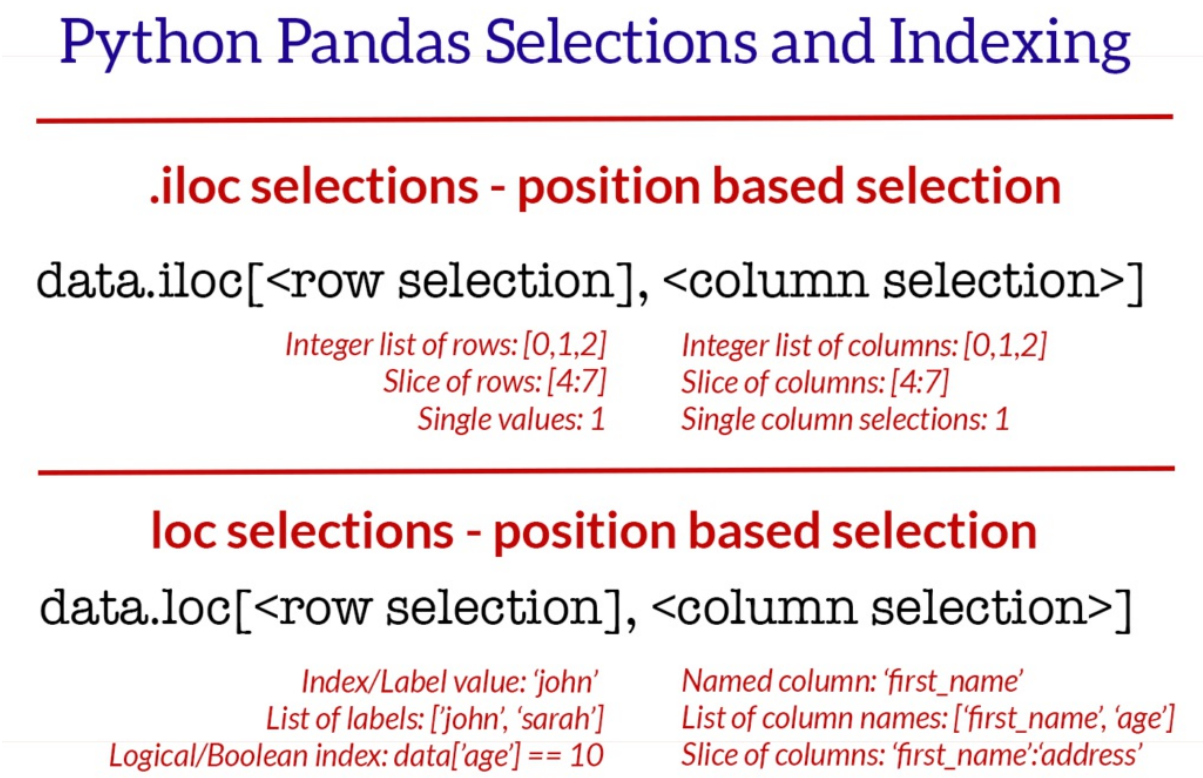
Rows in a DataFrame are selected, typically, using the iloc/loc selection methods, or using logical selectors (selecting based on the value of another column or variable).

The basic methods to get your heads around are:

- numeric row selection using the iloc selector, e.g. `data.iloc[0:10, :]` – select the first 10 rows.
- label-based row selection using the loc selector (this is only applicably if you have set an "index" on your dataframe. e.g. `data.loc[44, :]`
- logical-based row selection using evaluated statements, e.g. `data[data["Area"] == "Ireland"]` – select the

rows where Area value is ‘Ireland’.

Note that you can combine the selection methods for columns and rows in many ways to achieve the selection of your dreams. For details, please refer to the post “Using iloc, loc, and ix to select and index data“.



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

Deleting rows and columns (drop)

To delete rows and columns from DataFrames, Pandas uses the “drop” function.

To delete a column, or multiple columns, use the name of the column(s), and specify the “axis” as 1. Alternatively, as in the example below, the ‘columns’ parameter has been added in Pandas which cuts out the need for ‘axis’. The drop function returns a new DataFrame, with the columns removed. To actually edit the original DataFrame, the “inplace” parameter can be set to True, and there is no returned value.

```
# Deleting columns

# Delete the "Area" column from the dataframe
data = data.drop("Area", axis=1)

# alternatively, delete columns using the columns parameter of drop
data = data.drop(columns="area")

# Delete the Area column from the dataframe in place
# Note that the original 'data' object is changed when inplace=True
data.drop("Area", axis=1, inplace=True).

# Delete multiple columns from the dataframe
data = data.drop(["Y2001", "Y2002", "Y2003"], axis=1)
```

Rows can also be removed using the “drop” function, by specifying axis=0. Drop() removes rows based on “labels”, rather than numeric indexing. To delete rows based on their numeric position / index, use iloc to reassign the dataframe values, as in the examples below.

```
In [37]: # Preview the first 3 rows of the DataFrame
         data.head(3)
```

Out[37]:

	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude
0	AF	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71
1	AF	2	Afghanistan	2805	Rice (Milled Equivalent)	5142	Food	1000 tonnes	33.94	67.71
2	AF	2	Afghanistan	2513	Barley and products	5521	Feed	1000 tonnes	33.94	67.71

3 rows × 63 columns

```
In [38]: # Delete the rows with labels 0 & 1, and preview first 3 rows.
         data.drop([0,1], axis=0).head(3)
```

Out[38]:

	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude	...
2	AF	2	Afghanistan	2513	Barley and products	5521	Feed	1000 tonnes	33.94	67.71	...
3	AF	2	Afghanistan	2513	Barley and products	5142	Food	1000 tonnes	33.94	67.71	...
4	AF	2	Afghanistan	2514	Maize and products	5521	Feed	1000 tonnes	33.94	67.71	...

3 rows × 63 columns

The drop() function in Pandas be used to delete rows from a DataFrame, with the axis set to 0. As before, the inplace parameter can be used to alter DataFrames without reassignment.

```
# Delete the rows with labels 0,1,5
data = data.drop([0,1,2], axis=0)

# Delete the rows with label "Ireland"
# For label-based deletion, set the index first on the dataframe:
data = data.set_index("Area")
data = data.drop("Ireland", axis=0). # Delete all rows with label "Ireland"

# Delete the first five rows using iloc selector
data = data.iloc[5:,]
```

Renaming columns

Column renames are achieved easily in Pandas using the DataFrame rename function. The rename function is easy to use, and quite flexible. Rename columns in these two ways:

- Rename by mapping old names to new names using a dictionary, with form {“old_column_name”: “new_column_name”, …}
- Rename by providing a function to change the column names with. Functions are applied to every column name.

```
# Rename columns using a dictionary to map values
# Rename the Area columnn to 'place_name'
data = data.rename(columns={"Area": "place_name"})

# Again, the inplace parameter will change the dataframe without assignment
data.rename(columns={"Area": "place_name"}, inplace=True)

# Rename multiple columns in one go with a larger dictionary
data.rename(
    columns={
        "Area": "place_name",
        "Y2001": "year_2001"
    },
    inplace=True
)

# Rename all columns using a function, e.g. convert all column names to lower c
ase:
data.rename(columns=str.lower)
```

In many cases, I use a tidying function for column names to ensure a standard, camel-case format for variables names. When loading data from potentially unstructured data sets, it can be useful to remove spaces and lowercase all column names using a lambda (anonymous) function:

```
# Quickly lowercase and camelcase all column names in a DataFrame
data = pd.read_csv("/path/to/csv/file.csv")
data.rename(columns=lambda x: x.lower().replace(' ', '_'))
```

Exporting and Saving Pandas DataFrames

After manipulation or calculations, saving your data back to CSV is the next step. Data output in Pandas is as simple as loading data.

Two two functions you'll need to know are to_csv to write a DataFrame to a CSV file, and to_excel to write DataFrame information to a Microsoft Excel file.

```
# Output data to a CSV file
# Typically, I don't want row numbers in my output file, hence index=False.
# To avoid character issues, I typically use utf8 encoding for input/output.

data.to_csv("output_filename.csv", index=False, encoding='utf8')

# Output data to an Excel file.
# For the excel output to work, you may need to install the "xlsxwriter" packag
e.

data.to_csv("output_excel_file.xlsx", sheet_name="Sheet 1", index=False)
```

Additional useful functions

Grouping and aggregation of data

As soon as you load data, you'll want to group it by one value or another, and then run some calculations. There's another post on this blog – Summarising, Aggregating, and Grouping Data in Python Pandas, that goes into extensive detail on this subject.

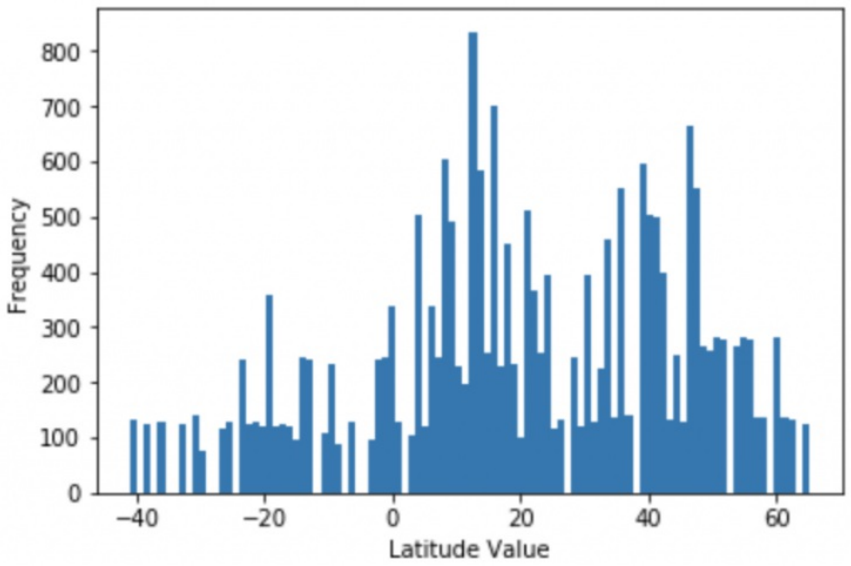
Plotting Pandas DataFrames – Bars and Lines

There's a relatively extensive plotting functionality built into Pandas that can be used for exploratory charts – especially useful in the Jupyter notebook environment for data analysis.

You'll need to have the matplotlib plotting package installed to generate graphics, and the `%matplotlib`

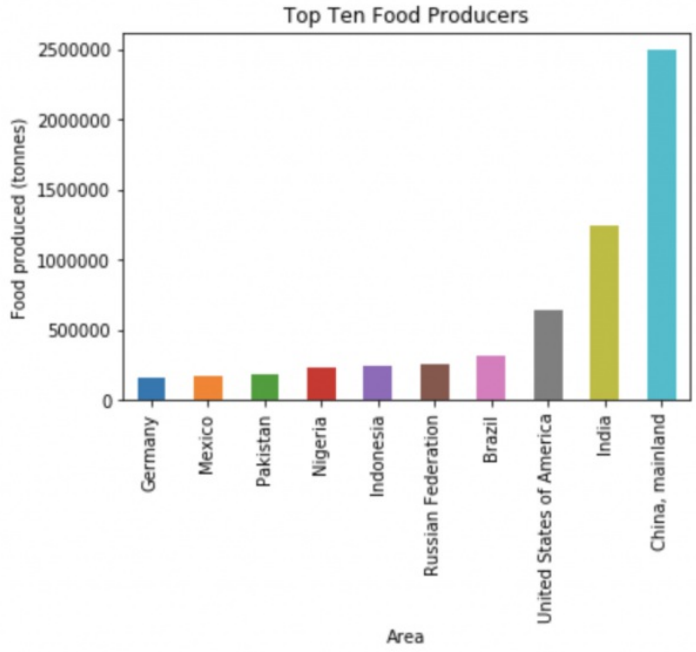
`inline` notebook 'magic' activated for inline plots. You will also need `import matplotlib.pyplot as plt` to add figure labels and axis labels to your diagrams. A huge amount of functionality is provided by the .plot() command natively by Pandas.

```
In [76]: data['latitude'].plot(kind='hist', bins=100)
         plt.xlabel('Latitude Value')

Out[76]: Text(0.5,0,'Latitude Value')
```



Create a histogram showing the distribution of latitude values in the dataset. Note that "plt" here is imported from matplotlib – 'import matplotlib.pyplot as plt'.

```
In [79]: # Select just data relevant to Food production
         plot_data = data[data['Element'] == 'Food']
         # Group the data by the Production Area and sum the values for 2013.
         plot_data = plot_data.groupby('Area')['Y2013'].sum()
         # Sort the values in ascending order, and plot
         # the last 10 values in a barchart
         plot_data.sort_values()[-10:].plot(kind='bar')
         plt.title("Top Ten Food Producers")
         plt.ylabel("Food produced (tonnes)")

Out[79]: Text(0,0.5,'Food produced (tonnes)')
```



Create a bar plot of the top food producers with a combination of data selection, data grouping, and finally plotting using the Pandas DataFrame plot command. All of this could be produced in one line, but is separated here for clarity.

With enough interest, plotting and data visualisation with Pandas is the target of a future blog post – let me know in the comments below!

For more information on visualisation with Pandas, make sure you review:

- The official Pandas documentation on plotting and data visualisation.
- Simple Graphing with Python from Practical Business Python
- Quick and Dirty Data Analysis with Pandas from Machine Learning Mastery.

Going further

As your Pandas usage increases, so will your requirements for more advance concepts such as reshaping data and merging / joining (see accompanying blog post.). To get started, I'd recommend reading the 6-part "Modern Pandas" from Tom Augspurger as an excellent blog post that looks at some of the more advanced indexing and data manipulation methods that are possible.

Related

Previous Post

Next Post

29 thoughts on “The Pandas DataFrame – loading, editing, and viewing data in Python”

JAMES
JANUARY 29, 2018 AT 3:35 PM

Another fantastic guide! Noticed a typo on the to_excel example

Reply

ADIL WARSI
FEBRUARY 12, 2018 AT 7:58 AM

very usefull tutorial, provided with pandas necessary info…….thnx

Reply

@CLARKE_RJ
MARCH 4, 2018 AT 12:07 AM

Thank you so much for a well paced and detailed overview of Pandas, kind Regards

Reply

JOHN_W
MARCH 18, 2018 AT 12:26 AM

Great tutorial. Just in case others find this, I found the encoding=”utf-8' didnt work for me but just substituted encoding=’ISO-8859-1' which worked. cheers, John.

Reply

DAN T
MAY 1, 2018 AT 3:12 AM

Extremely useful! Thank you!

Reply

TIM
MAY 20, 2018 AT 9:34 PM

This is the best thing i have read all year. Thank you. t

Reply

ABEL
JUNE 7, 2018 AT 12:57 AM

I want to store the length and the breadth of a dataset separately from shape, or any function that can make me get these. I used len(), it returns only the number of rows, but I need the function for the breadth also.

Thank you

Reply

SHANELYNN
JUNE 7, 2018 AT 8:18 AM

Hi Abel, have you looked at data.shape[0] and data.shape[1], or len(data.columns)?

Reply

ABEL
JUNE 7, 2018 AT 11:12 PM

Thank you very much. data.shape[0] returns the number of rows; data.shape[1] returns the number of columns and len(data.columns) returns the number of columns also.

It all worked

Reply

JAY
JUNE 12, 2018 AT 10:13 PM

Epic blog. This is very well detailed. This is a summary of 100's of Youtube, Datacamp, Udemy, and Stack Overflow tutorials I have taken so far on Python Pandas DataFrame for Datascience. Thank you @shanelynn.

Reply

Pingback: Python Pandas read_csv: Load Data from CSV Files | Shane Lynn

VALHERUBORN
JULY 27, 2018 AT 9:19 AM

What a guide! Thank you loads!

Reply

SHANELYNN
JULY 27, 2018 AT 10:23 AM

Thank you! Please do share with anyone you feel would be interested!

Reply

SHEAM UDDIN
AUGUST 20, 2018 AT 1:05 PM

Thanks for it

Reply

ABHISHEK SUBRAMANIAN
SEPTEMBER 6, 2018 AT 11:05 AM

This is a wonderful post to go from beginner to intermediate user level in pandas. I cannot think of any more methods

that I use regularly. Great Job !!

Reply

MAHER FAISAL
SEPTEMBER 11, 2018 AT 6:19 AM

Great job. much helping

Reply

RAMIRO RODAS
SEPTEMBER 11, 2018 AT 7:20 PM

Congratulations!!. This is quite clear post.

Reply

VISHALI
SEPTEMBER 17, 2018 AT 1:54 PM

Thanks for the post. I have a doubt

I have an excel file path, with “xlrd.open_workbook” I have opened the file and got the column names with row_values(0).
Now, I want to check whether the input file columns (have more compared to required columns) contains the required column names.

Is there anyway, that I can do with python using machine learning algorithm.

Hope to hear from you soon.

Thank you very much in advance.

Reply

SEAN ANDERSON
SEPTEMBER 17, 2018 AT 4:43 PM

Great post. Love brain dumps like this!

Reply

ASHLEY LEWIS
OCTOBER 25, 2018 AT 3:18 AM

Hi Shane,

This is such a clear and concise post. You did a really great job and it was extremely helpful. Thank you for it!

Best,

Reply

DAN
OCTOBER 26, 2018 AT 2:12 AM

Great examples and information! Thanks for helping me learn Python!

Reply

Pingback: Water data with Pandas – Part 1 | Massbalanced

LACED
NOVEMBER 24, 2018 AT 5:05 AM

Thank you so frickin much. I don't know why this has been so hard for me to find.

Reply

SAMUEL

FEBRUARY 16, 2019 AT 10:34 AM

awsome to tutorial about pandas I didn't know pandas can be so much flexible with data

Reply

DURGA SENGAR
MARCH 1, 2019 AT 12:43 PM

Its great article which help me lot to understand pandas Python lambda

Reply

SEAN GOODMAN
MARCH 9, 2019 AT 7:30 PM

This is great. Thank you Shane. As stated earlier this combines all the tutorials i have been studying on YouTube. Excellent. Thank you

Reply

KANCHI PANCHAL
MARCH 11, 2019 AT 3:01 PM

While saving dataframe I am getting an error as:
PermissionError: [Errno 13] Permission denied: 'output.csv'

how to resolve this??

Please help me out

Reply

PRIYA
APRIL 9, 2019 AT 3:25 PM

Good Post! Thank you so much for sharing this pretty post, it was so good to read and useful to improve my knowledge

as updated one, keep blogging.

Reply

WENDY
APRIL 10, 2019 AT 1:56 AM

Thanks for the sharing, that is very helpful.

Reply

Leave a Reply

Email (required) (Address never made public)

Name (required)

Website

Save my name, email, and website in this browser for the next time I comment.

Notify me of new comments via email.
Notify me of new posts via email.

Post Comment

Coffee generator:

Subscribe to Blog via Email

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Email Address

Subscribe

Categories

Select Category

Copyright © 2019 Shane Lynn | Powered by Astra