10 Reasons Working With Any Kind Of Data In Python, Using Pandas Dataframes Is Awesome

Craig Godden-Payne
Mar 10 · 8 min read ★

Pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools for the python programming language.

It works really well with heterogeneous data, and my aim is to show some examples of how to use it over a quirky dataset

The data set that I will be using is as follows:

formatted-data:

Id	Name	Age	Colour	Long	Lat	Active
1	Adrian	20	Red	53.494365	-2.166388	True
2	Brian	NULL	Blue	53.528015	-2.367448	True
3	Chris	30	Yellow	NULL	NULL	False
4	David	32	Green	53.497630	-2.227112	NULL
5	Eric	32	Purple	53.957493	-2.347803	True
6	Francis	40	Orange	53.707394	-2.035048	True
7	NULL	18	Orange	53.283746	-2.203748	False
8	Henry	NULL	Black	53.654947	-2.026495	True
9	Isla	22	White	NULL	-2.975037	True
10	Jessie	28	NULL	53.494036	-2.947930	True

data.csv:

```
Id,Name,Age,Colour,Long,Lat,Active
1,Adrian,20,Red,53.494365,-2.166388,True
2,Brian,NULL,Blue,53.528015,-2.367448,True
3,Chris,30,Yellow,NULL,NULL,False
4,David,32,Green,53.497630,-2.227112,NULL
5,Eric,32,Purple,53.957493,-2.347803,True
6,Francis,40,Orange,53.707394,-2.035048,True
7,NULL,18,Orange,53.283746,-2.203748,False
8,Henry,NULL,Black,53.654947,-2.026495,True
9,Isla,22,White,NULL,-2.975037,True
10,Jessie,28,NULL,53.494036,-2.947930,True
```

Photo by Pascal Müller on Unsplash

1. It is really easy to load a dataset into a pandas dataframe

You can load data in different ways, in my case the data was in csv format, and it can easily be loaded using the read_csv method:

```
>>> import pandas as pd
>>> pd.read_csv('./data.csv')
   Id     Name   Age  Colour       Long       Lat Active
0   1   Adrian  20.0     Red  53.494365 -2.166388   True
1   2    Brian   NaN    Blue  53.528015 -2.367448   True
2   3    Chris  30.0  Yellow        NaN       NaN  False
3   4    David  32.0   Green  53.497630 -2.227112    NaN
4   5     Eric  32.0  Purple  53.957493 -2.347803   True
5   6  Francis  40.0  Orange  53.707394 -2.035048   True
6   7      NaN  18.0  Orange  53.283746 -2.203748  False
7   8    Henry   NaN   Black  53.654947 -2.026495   True
8   9     Isla  22.0   White        NaN -2.975037   True
9  10   Jessie  28.0     NaN  53.494036 -2.947930   True
```

Photo by Chris Curry on Unsplash

2. Printing a dataframe or just output results just looks so perfect

There really is no need to format the output of the dataframe, it just seems to always get it perfect. The dataframe is outputted into a neat table which is really easy to read. This can be an issue with huge datasets printing out a lot of data, but there are functions to help out with that too

If you are working within a python shell, or conda environment, this will be printed after the statement. If you are working within a some python code, you can assign the dataframe to a variable or just print() or log it out using a logging library.

```
import pandas as pd
dataframe = pd.read_csv('./data.csv')
print(dataframe)
```

```
    Id     Name   Age  Colour       Long       Lat Active
0    1   Adrian  20.0     Red  53.494365 -2.166388   True
1    2    Brian   NaN    Blue  53.528015 -2.367448   True
2    3    Chris  30.0  Yellow        NaN       NaN  False
3    4    David  32.0   Green  53.497630 -2.227112    NaN
4    5     Eric  32.0  Purple  53.957493 -2.347803   True
5    6  Francis  40.0  Orange  53.707394 -2.035048   True
6    7      NaN  18.0  Orange  53.283746 -2.203748  False
7    8    Henry   NaN   Black  53.654947 -2.026495   True
8    9     Isla  22.0   White        NaN -2.975037   True
9   10   Jessie  28.0     NaN  53.494036 -2.947930   True
```

Photo by Michael Payne on Unsplash

3. And it does a pretty good job at assuming the types

When the dataset is loaded into a dataframe, pandas will attempt a best guess at the types, which is pretty awesome.

```
>>> df.dtypes
Id          int64
Name       object
Age       float64
Colour     object
Long      float64
```

```
Lat       float64
Active     object
```

and you can explicitly change the types if you know you data type better

```
>>> df['Name'] = df['Name'].astype('string')
>>> df.dtypes
Id          int64
Name       string
Age       float64
Colour     object
Long      float64
Lat       float64
Active     object
```

4. You can head() and tail() your dataframe

These are really handy if you only want to select the top or bottom X, or just output a smaller subset of data. You can even pass in the amount of rows to select

```
>>> import pandas as pd
>>> df = pd.read_csv('./data.csv')
>>> df.head(5)
   Id    Name   Age  Colour       Long       Lat Active
0   1  Adrian  20.0     Red  53.494365 -2.166388   True
1   2   Brian   NaN    Blue  53.528015 -2.367448   True
2   3   Chris  30.0  Yellow        NaN       NaN  False
3   4   David  32.0   Green  53.497630 -2.227112    NaN
4   5    Eric  32.0  Purple  53.957493 -2.347803   True

>>> import pandas as pd
>>> df = pd.read_csv('./data.csv')
>>> df.tail(3)
   Id    Name   Age Colour       Long       Lat Active
7   8   Henry   NaN  Black  53.654947 -2.026495   True
8   9    Isla  22.0  White        NaN -2.975037   True
9  10  Jessie  28.0    NaN  53.494036 -2.947930   True
```

Photo by Ilona Froehlich on Unsplash

5. It's really easy to select the columns you need

You can select columns from your dataframe in different ways.

- Named columns as a list
- Named columns as a range, e.g. from Name to Colour
- Column Indexes
- Column Index ranges

```
>>> import pandas as pd
>>> df = pd.read_csv('./data.csv')
>>> df[['Name', 'Age']]
      Name   Age
0   Adrian  20.0
1    Brian   NaN
2    Chris  30.0
3    David  32.0
4     Eric  32.0
5  Francis  40.0
6      NaN  18.0
7    Henry   NaN
8     Isla  22.0
9   Jessie  28.0

>>> df.loc[:, 'Name':'Colour']
      Name   Age  Colour
0   Adrian  20.0     Red
1    Brian   NaN    Blue
2    Chris  30.0  Yellow
3    David  32.0   Green
4     Eric  32.0  Purple
```

```
5  Francis  40.0  Orange
6      NaN  18.0  Orange
7    Henry   NaN   Black
8     Isla  22.0   White
9   Jessie  28.0     NaN

>>> df.iloc[:, [1,2,3]]
      Name   Age  Colour
0   Adrian  20.0     Red
1    Brian   NaN    Blue
2    Chris  30.0  Yellow
3    David  32.0   Green
4     Eric  32.0  Purple
5  Francis  40.0  Orange
6      NaN  18.0  Orange
7    Henry   NaN   Black
8     Isla  22.0   White
9   Jessie  28.0     NaN

>>> df.iloc[:, 2:6]
    Age  Colour       Long       Lat
0  20.0     Red  53.494365 -2.166388
1   NaN    Blue  53.528015 -2.367448
2  30.0  Yellow        NaN       NaN
3  32.0   Green  53.497630 -2.227112
4  32.0  Purple  53.957493 -2.347803
5  40.0  Orange  53.707394 -2.035048
6  18.0  Orange  53.283746 -2.203748
7   NaN   Black  53.654947 -2.026495
8  22.0   White        NaN -2.975037
9  28.0     NaN  53.494036 -2.947930
```

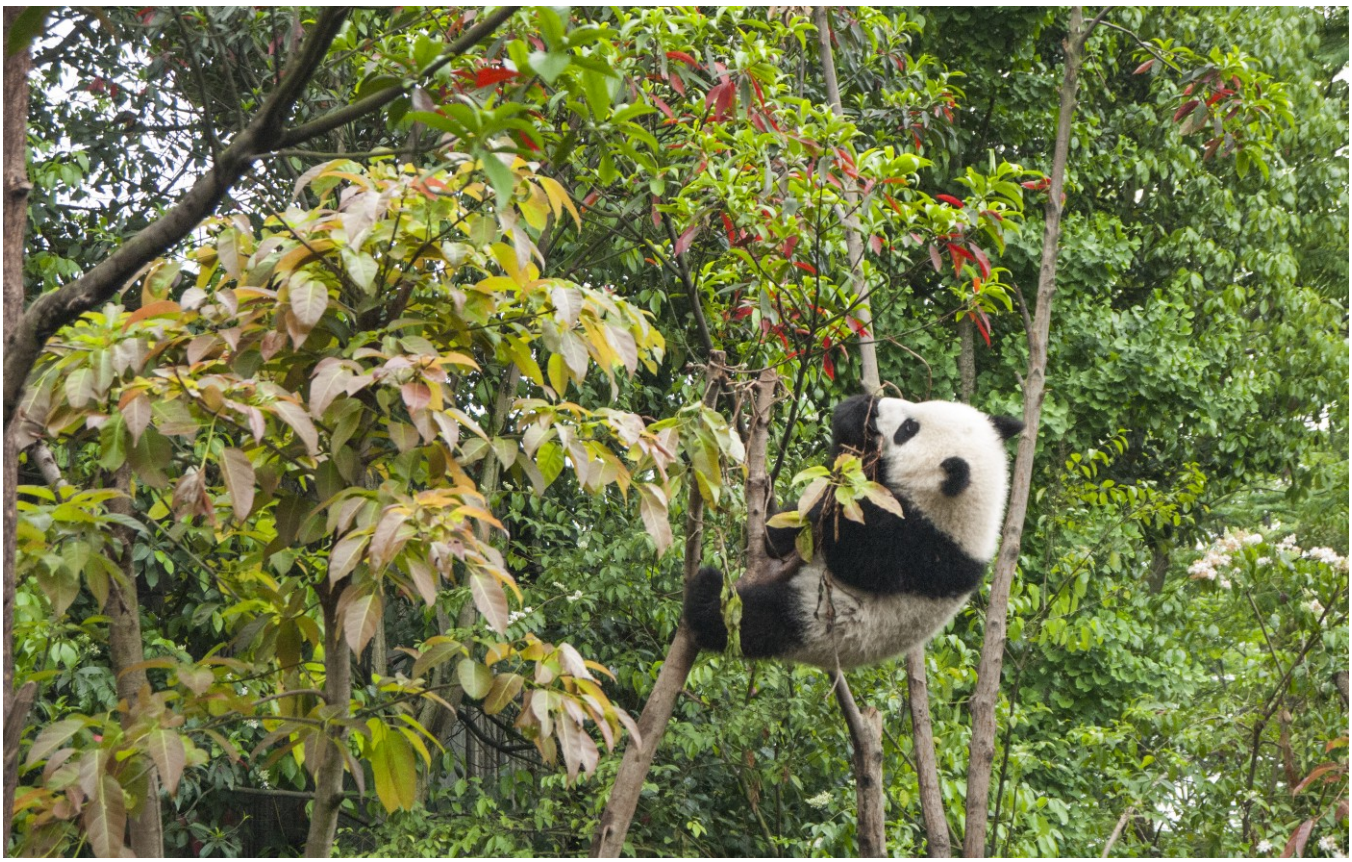

Photo by Chester Ho on Unsplash

6. And you can filter rows in a very similar manner whilst you are at it

You can use the same filters on the dataframe to select certain rows

- Use iloc with a list to return many rows
- Use iloc with a single row to return single dimension

```
>>> df.iloc[:, [1,2,3]].iloc[[6,7,8]]
    Name   Age  Colour
6    NaN  18.0  Orange
7  Henry   NaN   Black
8   Isla  22.0   White

>>> df.iloc[:, [1,2,3]].iloc[6]
Name         NaN
Age           18
Colour    Orange
```

Photo by Stan W. on Unsplash

7. Or you can apply a filter and do it that way

You don't have to filter using indexes, you could say only filter where the Age is between 20 and 30, or only select Active users

```
>>> df[df.Age.between(20,30)]
   Id    Name   Age  Colour       Long       Lat Active
0   1  Adrian  20.0     Red  53.494365 -2.166388   True
2   3   Chris  30.0  Yellow        NaN       NaN  False
8   9    Isla  22.0   White        NaN -2.975037   True
9  10  Jessie  28.0     NaN  53.494036 -2.947930   True

>>> df[df.Active==True]
   Id     Name   Age  Colour       Long       Lat Active
0   1   Adrian  20.0     Red  53.494365 -2.166388   True
1   2    Brian   NaN    Blue  53.528015 -2.367448   True
4   5     Eric  32.0  Purple  53.957493 -2.347803   True
5   6  Francis  40.0  Orange  53.707394 -2.035048   True
7   8    Henry   NaN   Black  53.654947 -2.026495   True
8   9     Isla  22.0   White        NaN -2.975037   True
9  10   Jessie  28.0     NaN  53.494036 -2.947930   True
```

Photo by Vanessa on Unsplash

8. You can convert dataframes to json pretty easily

Theres a to_json function, which allows you to print the dataframe out as json, in varying dimensions. The default uses ‘columns’ output, but I prefer to use ‘records’ as it is in a format I find most useful

I have pretty printed the outputs here, to increase readability

```
df[df.Active==True].head(2).to_json()
{
  "Id": {
    "0": 1,
    "1": 2
  },
  "Name": {
    "0": "Adrian",
    "1": "Brian"
  },
  "Age": {
    "0": 20,
    "1": null
  },
  "Colour": {
    "0": "Red",
    "1": "Blue"
  },
  "Long": {
    "0": 53.494365,
    "1": 53.528015
  },
  "Lat": {
    "0": -2.166388,
    "1": -2.367448
  },
  "Active": {
    "0": true,
    "1": true
  }
}

>>> df[df.Active==True].head(2).to_json(orient='records')
[
  {
    "Id": 1,
    "Name": "Adrian",
    "Age": 20,
    "Colour": "Red",
    "Long": 53.494365,
    "Lat": -2.166388,
    "Active": true
  },
```

```
  {
    "Id": 2,
    "Name": "Brian",
    "Age": null,
    "Colour": "Blue",
    "Long": 53.528015,
    "Lat": -2.367448,
    "Active": true
  }
]
```

Photo by chuttersnap on Unsplash

9. Or even back to CSV format

Theres a to_csv function, which allows you to print the dataframe out as csv

In this case, null values are set as empty, it's important to notice this kind of thing, as converting to different formats can cause your data to drift.

```
>>> df.head(8).to_csv()
',Id,Name,Age,Colour,Long,Lat,Active\n0,1,Adrian,20.0,Red,53.494365,
```

```
-2.166388,True\n1,2,Brian,,Blue,53.528015,-2.367448,True\n2,3,Chris,
30.0,Yellow,,,False\n3,4,David,32.0,Green,53.49763000000001,-2.22711
2,\n4,5,Eric,32.0,Purple,53.95749300000001,-2.347803,True\n5,6,Franc
is,40.0,Orange,53.707393999999994,-2.0350479999999997,True\n6,7,,18.
0,Orange,53.283745999999994,-2.203748,False\n7,8,Henry,,Black,53.654
947,-2.026495,True\n'
```

Photo by billow 926 on Unsplash

10. Handling missing or erroneous data is simple

There are features to skip missing data, or include missing data. You can work on the columnal data like below, which will return all data that is NaN or not NaN

```
>>> df.head(5)[df.Name.notna()]
__main__:1: UserWarning: Boolean Series key will be reindexed to
match DataFrame index.
   Id    Name   Age  Colour       Long       Lat Active
0   1  Adrian  20.0     Red  53.494365 -2.166388   True
1   2   Brian   NaN    Blue  53.528015 -2.367448   True
2   3   Chris  30.0  Yellow        NaN       NaN  False
3   4   David  32.0   Green  53.497630 -2.227112    NaN
4   5    Eric  32.0  Purple  53.957493 -2.347803   True

>>> df.head(10)[df.Name.isna()]
   Id Name   Age  Colour       Long       Lat Active
```

```
6   7  NaN  18.0  Orange  53.283746 -2.203748  False
```

Photo by Theodor Lundqvist on Unsplash

4.2K signups

Pandas Data Analysis Data Analytics Data Science Python