Pandas Series & DataFrame Explained

A comprehensive guide to understanding Pandas Series and DataFrame data structures

Dean McGrath Oct 9, 2020 · 4 min read ★

Photo by Eleonora Albasi on Unsplash

Introduction

To be successful as a Data Scientist one needs to be continuously learning and improving our skills across a wide range of tools. A tool synonymous with Data Science these days is Pandas. Pandas is an incredibly powerful open-source library written in Python. It offers a diverse set of tools that we as Data Scientist can use to clean, manipulate and analyse data. Today we are beginning with the fundamentals and learning two of the most common data structures in Pandas the Series and DataFrame.

Getting Started

Installing Pandas & Numpy

Pandas and Numpy are open-source libraries written in Python that you can utilise in your Python scripts. There are several ways that you can install these packages. By default, if you are using Anaconda, these packages are already installed. If you aren't using Anaconda, then the most straightforward installation option is using `pip`, which is Python's recommended installation package. You can find in-depth installation instructions for `pip` within PyPA's documentation.

Once `pip` has finished installing, you can run the following command on a terminal to install Pandas and Numpy `pip install pandas` `pip install numpy`. After installation, you can import Pandas and Numpy libraries into your scripts using the below syntax.

```
1  import pandas as pd
2  import numpy as np
```

pandas_numpy_import.py hosted with ♡ by GitHub view raw

The above Python snippet shows the syntax for importing Pandas and Numpy packages.

Pandas Series

The Pandas Series data structure is a one-dimensional labelled array. It is the primary building block for a DataFrame, making up its rows and columns. You can view the constructor for the Series below.

```
1  # pandas.Series
2  series = pd.Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)
```

pandas_series_constructor.py hosted with ♡ by GitHub view raw

The above Python snippet shows the constructor for a Pandas Series.

The `data` parameter can accept several different data types such as ndarray, dictionaries and scalar values. The `index` parameter accepts array-like objects which will allow you to label your index axis. If you don't pass an item to the `index` parameter and a dictionary is given to the `data` parameter, then Pandas will use the dictionary keys as index labels. You can set the data type for the Series by setting the `dtype` parameter. If a data type is not specified then, Pandas will make inferences to the data types the Series should be. The `name` parameter does as it suggests, allowing you to name the Series that you have created.

Creating a Series Using a Dictionary

Below we have provided with an example Python snippet which you can use to create a Series.

```python
import pandas as pd

if __name__ == '__main__':
    data = {'a': 1.0, 'b': 2.0, 'c': 3.0, 'd': 4.0}
    series = pd.Series(data=data, name='series_from_dict')
    print(series)
```

pandas_series_dict.py hosted with ♡ by GitHub view raw

The above Python snippet demonstrates how to create and name a Series using a dictionary.

```
a    1.0
b    2.0
c    3.0
d    4.0
Name: series_from_dict, dtype: float64
```

The above image shows the result of executing the Python snippet to create a Series.

From the above console output, we can see that the Series has used the dictionary keys as indexes, the Series has was named `series_from_dict`, and Pandas inferred a data type of `float64`.

Creating a Series From ndarray

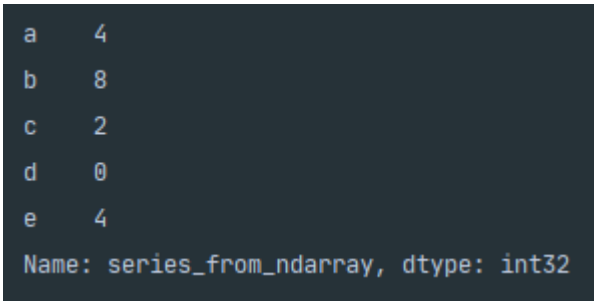
One of the quickest ways to generate a Series for testing is using NumPy's `random.randint()` which produces a ndarray populated with random integers.

```
import numpy as np
import pandas as pd

if __name__ == '__main__':
    data = np.random.randint(0, 10, 5)
    series = pd.Series(data=data,
                       index=['a', 'b', 'c', 'd', 'e'],
                       name='series_from_ndarray')
    print(series)
```

pandas_series_ndarray.py hosted with ♡ by GitHub view raw

The above Python snippet demonstrates how to create and name a Series using a ndarray.



The above image shows the result of executing the Python snippet.

Creating a Series From Scalar Values

The final method we are going to look at today is creating a Series using Scalar values. Here you can assign a single value to `data` and have it repeated for the length of the index.

```
import pandas as pd

if __name__ == '__main__':
    series = pd.Series(data=3.,
                       index=['a', 'b', 'c', 'd'],
                       name='series_from_scalar')
    print(series)
```

pandas_series_scalar.py hosted with ♡ by GitHub view raw

The above Python snippet shows how to create a Series by passing a scalar value.

```
a    3.0
b    3.0
c    3.0
d    3.0
Name: series_from_scalar, dtype: float64
```

The above image shows the result of executing the Python snippet.

Pandas DataFrame

The Pandas DataFrame is a two-dimensional data structure composed of columns and rows. You can think of the DataFrame as similar to a CSV or relational database table. Below you can see the constructor for creating a DataFrame.

```
# pandas.DataFrame
df = pd.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)
```

dataframe_constructor.py hosted with ♡ by GitHub view raw

The above Python snippet shows the constructor for a Pandas DataFrame.

The `data` parameter similar to Series can accept a broad range of data types such as a Series, a dictionary of Series, structured arrays and NumPy arrays. In addition to being able to pass index labels to `index`, the DataFrame constructor can accept column names through `columns`.

Creating a DataFrame for a Dictionary of Series

One of the easiest ways to generate a DataFrame is creating a dictionary containing Series. The dictionary keys will become the DataFrame column labels, and the Series indexes will become the DataFrame row labels. Below is a Python snippet that you can use to produce your first DataFrame.

```
import numpy as np
import pandas as pd

if __name__ == '__main__':
    data = {'column_a': pd.Series(data=np.random.randint(10, 100, 5),
```

```
                                    index=['a', 'b', 'c', 'd', 'e']),
            'column_b': pd.Series(data=np.random.randint(10, 100, 5),
                                    index=['a', 'b', 'c', 'd', 'e']),
            'column_c': pd.Series(data=np.random.randint(10, 100, 5),
                                    index=['a', 'b', 'c', 'd', 'e'])}
    df = pd.DataFrame(data=data)
    print(df)
```

dataframe_series_dict.py hosted with ♡ by GitHub view raw

The above Python snippet shows how to create a DataFrame using a dictionary of Series.

	column_a	column_b	column_c
a	47	46	30
b	72	63	16
c	89	28	85
d	95	13	22
e	83	63	23

The above image shows the result of executing the Python script.

The console output above clearly demonstrates the DataFrame data structure. The dictionary keys `column_a`, `column_b` and `column_c` now form the labelled columns and the Series indexes `a`, `b`, `c`, `d` and `e` are the DataFrame row labels.

Creating a DataFrame for a Dictionary of Lists

The Pandas DataFrame constructor can also receive a dictionary of lists. Below we have provided you with a Python snippet that you can use to achieve this. Here we have assigned the row labels through the `index` parameter within the DataFrame constructor.

```
import pandas as pd

if __name__ == '__main__':
    data = {'column_a': [1, 2, 3],
            'column_b': [4, 5, 6],
            'column_c': [7, 8, 9]}
    df = pd.DataFrame(data=data, index=['row_a', 'row_b', 'row_c'])
    print(df)
```

dataframe_list_dict.py hosted with ♡ by GitHub view raw

The above Python snippet shows how to create a DataFrame from a dictionary of lists.

	column_a	column_b	column_c
row_a	1	4	7
row_b	2	5	8
row_c	3	6	9

The above image shows the result of creating a DataFrame from a dictionary of lists.

Where to next?

Now that you have covered the fundamental building blocks of Pandas, your next steps should be learning how to navigate the DataFrame through iterating a DataFrame or diving headfirst into analysing with Pandas Profiling.

Thank you again for taking the time to read our story — we hope you have found it valuable!

Python Programming Pandas Data Science Computer Science