

✓ Pandas

The "ndarray" data structure of Numpy provides features for clean, well-organized, and many high-level operations. However, it is inflexible while working with missing data, attaching labels to the data, and attempting operations that don't require broadcasting (e.g., grouping, pivots, etc.). Pandas, built on top of NumPy, provides an efficient solution to these sorts of "data munging" tasks.

	Company	Automatic shift	Color	Number of doors
0	Ford	1	red	4
1	Ferrari	1	blue	2
2	Lamborghini	1	white	2
3	Toyota	0	white	4

In general, "Pandas" is a way to store data efficiently on a computer. In the above image, you can see a table of data storing information about 4 cars. Usually, data required for machine learning contains many more features (the columns that tell us the different attributes of the cars), and observations (the rows that represent different cars). "Pandas" helps us organize and manipulate this data. It is a popular open-source library in Python, which means that you can get it for free.

✓ Handling data using Pandas

✓ Series

If you look at one individual column, "Company," for example, this is a Series. There are four columns, which means that this table is made up of four pandas series put together. If you are wondering about the numbers 0, 1, 2, and 3 on the left, they are just indices, so we don't count them.

Alternatively, you can also look at each row as a Series. The first row where it says "Company," "Automatic shift," etc. is the equivalent of indices, we do not count them. They simply describe what the data means. If you look at it this way, there are four rows, so there are four pandas Series.

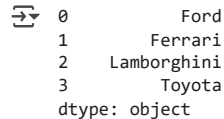
Let us create a pandas Series.

```
import pandas as pd

name_list = ['Ford', 'Ferrari', 'Lamborghini', 'Toyota']

# Create Series
names = pd.Series(name_list)

names
```

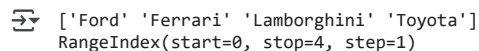


```
0      Ford
1    Ferrari
2  Lamborghini
3     Toyota
dtype: object
```

In the above code, we first made a list of the values we want to put in the Series, and then assigned it to the variable `names`. Then we created a Series object and assigned it to `names`.

We can also print the values and index of the Series.

```
print(names.values)
print(names.index)
```



```
['Ford' 'Ferrari' 'Lamborghini' 'Toyota']
RangeIndex(start=0, stop=4, step=1)
```

✓ Series - an enhanced version of NumPy

The series object looks like a one-dimensional NumPy array. However, while the index in a NumPy array is implicitly defined, the series in Pandas have explicitly defined the index. Due to this explicit definition, the series object is not limited only to the integer index, but we can

also have strings as an index.

```
data = pd.Series([5, 10, 15, 20],
                 index=['a', 'b', 'c', 'd'])
```

data

```
↗ a    5
  b   10
  c   15
  d   20
dtype: int64
```

▼ Access data from series

```
# numerical indexing/slicing as numpy
print(data[0])
print(data[0:4])
```

```
↗ 5
  a    5
  b   10
  c   15
  d   20
dtype: int64
```

```
# accessing data using the actual index
print(data['a'])
```

```
↗ 5
  a    5
  b   10
  c   15
  d   20
dtype: int64
```

▼ DataFrame

The table that we saw is called a DataFrame in pandas. It is a simple table that stores data. The "pandas" library aims to enhance data manipulation; it is good for us to put the data into a pandas DataFrame object. It allows us to do many operations effortlessly.

Let us create a DataFrame.

```
import pandas as pd
```

```
#create dataframe
df = pd.DataFrame()
```

```
print(df)
```

```
↗ Empty DataFrame
  Columns: []
  Index: []
```

As you can imagine, this DataFrame is empty. Now let us add the first column. We use the Series we described above to create this.

```
import pandas as pd
```

```
name_list = ['Ford', 'Ferrari', 'Lamborghini', 'Toyota']
```

```
# Create Series
names = pd.Series(name_list)
```

```
#create dataframe
df = pd.DataFrame()
df['Company'] = names
```

```
df
```



	Company	Automatic Shift
0	Ford	1
1	Ferrari	1
2	Lamborghini	1
3	Toyota	0

We create a new column inside the DataFrame. We do this when we write `df['Company']`. Since this is an empty DataFrame, there is no column called 'Company', so the DataFrame creates one and puts the contents in it.

This DataFrame has just one column. We need to make other columns as well.

```
import pandas as pd

name_list = ['Ford', 'Ferrari', 'Lamborghini', 'Toyota']
shift_list = [1,1,1,0]
color_list = ['red', 'blue', 'white', 'white']
door_list = [4,2,2,4]
```

```
# Create Series
names = pd.Series(name_list)
shift = pd.Series(shift_list)
color = pd.Series(color_list)
door = pd.Series(door_list)
```

```
#create dataframe
df = pd.DataFrame()

df['Company'] = names
df['Automatic shift'] = shift
df['Color'] = color
df['Number of doors'] = door
```

```
#display dataframe
df
```



	Company	Automatic shift	Color	Number of doors
0	Ford	1	red	4
1	Ferrari	1	blue	2
2	Lamborghini	1	white	2
3	Toyota	0	white	4

✓ Pandas makes it easier

We made the DataFrame taking the long route. We wanted to show you step by step. When we use "pandas" on a day to day basis, we want more comfortable ways to create these tables. "Pandas" has many different ways with which might make your job easier.

For example, instead of creating each original series first, and then "gluing" them together, Pandas lets us create DataFrames directly.

```
import pandas as pd

name_list = ['Ford', 'Ferrari', 'Lamborghini', 'Toyota']
shift_list = [1,1,1,0]
color_list = ['red', 'blue', 'white', 'white']
door_list = [4,2,2,4]
```

```
#create dataframe
df = pd.DataFrame(data={'Company':name_list,
                        'Automatic shift':shift_list,
                        'Color':color_list,
                        'Number of doors':door_list})
```

```
#display dataframe
df
```

```

Company Automatic shift Color Number of doors
0 Ford 1 red 4
1 Ferrari 1 blue 2
2 Lamborghini 1 white 2
3 Toyota 0 white 4

```

In the code above, we have first created the lists that represent the information in the columns. Then, instead of creating a Series, we put the information directly into a DataFrame. We made a dictionary, with the keys representing the names of the column names, and the values representing the information.

We can also view more information about DataFrame.

```
df.shape
```

```
(4, 4)
```

```
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
Company      4 non-null object
Automatic shift  4 non-null int64
Color        4 non-null object
Number of doors  4 non-null int64
dtypes: int64(2), object(2)
memory usage: 208.0+ bytes

```

```
df.describe()
```

```

Automatic shift Number of doors
count          4.00          4.000000
mean           0.75          3.000000
std            0.50          1.154701
min            0.00          2.000000
25%            0.75          2.000000
50%            1.00          3.000000
75%            1.00          4.000000
max            1.00          4.000000

```

We can use other ways as well. For example, we can pass lists instead of dictionaries.

```

import pandas as pd

A = ['a', 'b', 'c', 'd']
B = ['e', 'f', 'g', 'h']

#create dataframe
df = pd.DataFrame(data=[A,B])

#display dataframe
df

```

```

0 1 2 3
0 a b c d
1 e f g h

```

Here we made two lists and then fed them as a list when creating the DataFrame. Notice that the lists are row-wise and not column-wise. With our previous method, the letters a, b, c, and d would have appeared as one column, and the other letters would have been another column. When you use lists to input data, "Pandas" puts them in as individual rows.

Bear in mind that the previous approach where we used dictionaries to create the DataFrame had 'keys' in the dictionary that represented the names. This behavior might be a way to remember that lists translate into rows of the Dataframe.

If you look at the output above, you can see that the columns numbered as 0, 1, 2, and 3. You also have the option of naming these columns

```
#create dataframe
df = pd.DataFrame(data=[A,B], columns=['A','B','C','D'])
df
```



	A	B	C	D
0	a	b	c	d
1	e	f	g	h

In the code above, we have simply used the parameter `columns` to pass a list of the four names of the columns.

✓ Reading CSV file into DataFrame

Usually, in a machine learning project, the data are stored in a file on a computer. This file is usually of a popular format called CSV. It stands for Comma Separated Values. It is a text file that contains the table. A comma separates each element in the table, hence the name.

Reading from this file is made very easy in Pandas.

```
import pandas as pd
dataset_path = 'folder/my_data.csv'

df = pd.read_csv(dataset_path)
```

That's it! All we need to do is pass the path for finding the file to the `read_csv` function.

References

Articles

- [Pandas Tutorial 1: Pandas Basics](#)
- [Load CSV](#)
- [Pandas apply](#)

Tutorial

- [Tutorial Point](#)