Pandas Tutorial: Essentials of Data Science in Pandas Library

Abdishakur
May 8, 2018 · 5 min read

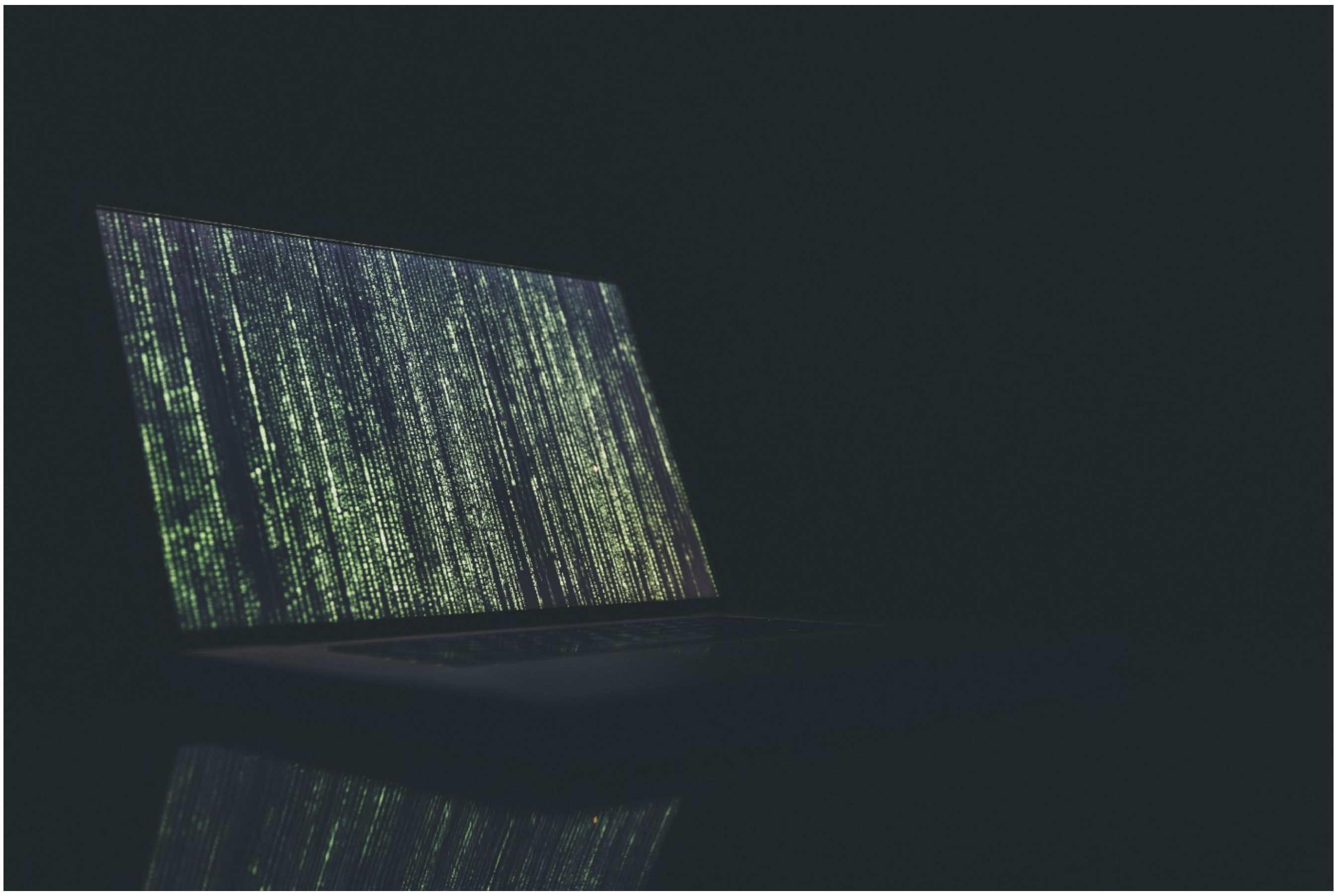


Photo by Markus Spiske on Unsplash

This is a 5-part series on learning the basic foundations of using Pandas library for data science. In this post, I will touch on the basics of using Pandas. I assume you have already installed Python and Pandas in your computer. If not, I would suggest downloading/installing Anaconda or simply typing this into your shell/terminal:

```
conda install -c anaconda python
```

Once Anaconda is installed, simply run the following to install Pandas:

```
conda install pandas
```

Before using Pandas,you need to import the library:

```
import pandas as pd
```

. . .

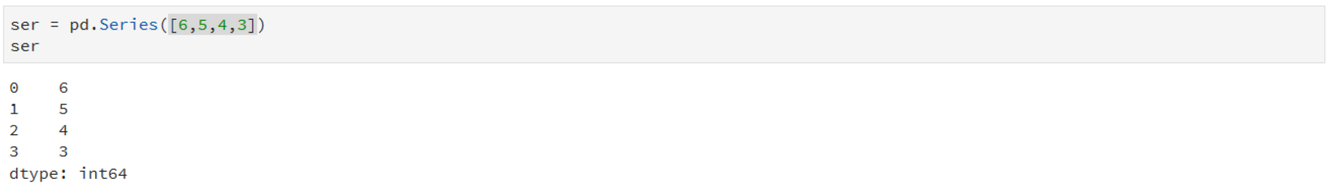
1. Data Structures

In Pandas library, there are two main data types: ***Series and DataFrames.***

Series: is a one-dimensional object that contains a sequence of values and data labels(index). The basic syntax for creating is:

```
pd.Series(data, index)
```

You can create Pandas series as the following:

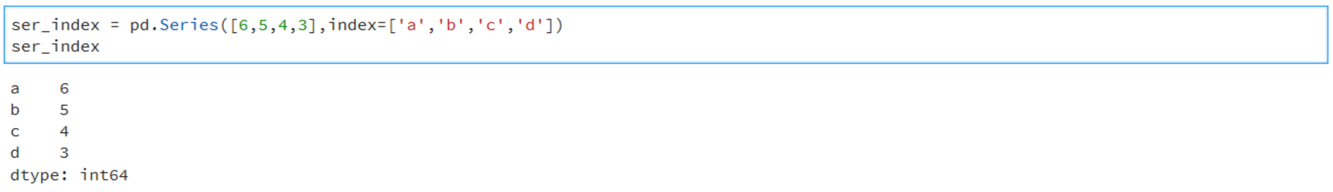


```
ser = pd.Series([6,5,4,3])
ser
```

```
0    6
1    5
2    4
3    3
dtype: int64
```

Pandas Series

In this case, notice that we have only provided the data. The index is optional to use and you can leave it. Providing index is handy when you want to provide your own index, be it numbers starting from zero or any other alphanumeric sequences.



```
ser_index = pd.Series([6,5,4,3],index=['a','b','c','d'])
ser_index
```

```
a    6
b    5
c    4
d    3
dtype: int64
```

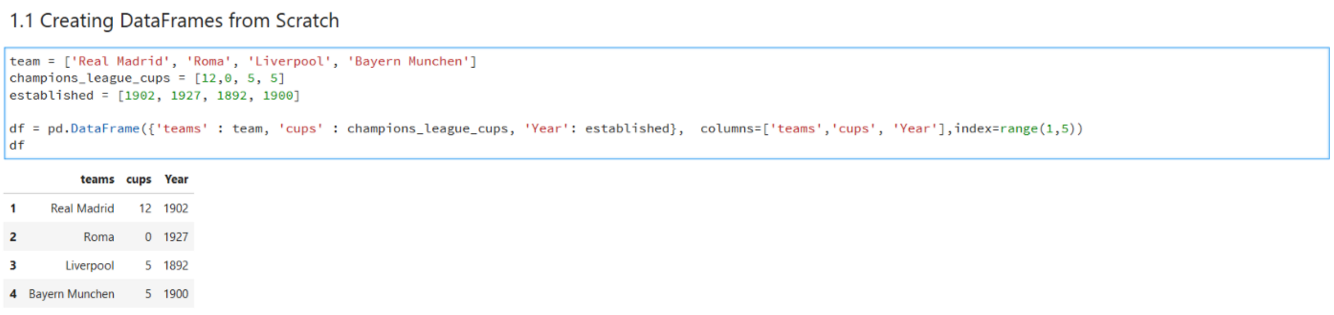
Pandas Series with Index

DataFrame: has two-dimensional data consisting of rows and columns. It is tabular data, like excel. Columsn can be of different types as we will see (Strings, numeric, datetime, etc.)

The syntax for creating DataFrame is similar to creating Series:

```
pd.(data, index)
```

To create a DataFrame, you have many options. You might read it from files (Excel, CSV, etc.), read it from Database or from the web. Let us start first with creating DataFrame from scratch to understand the basics.



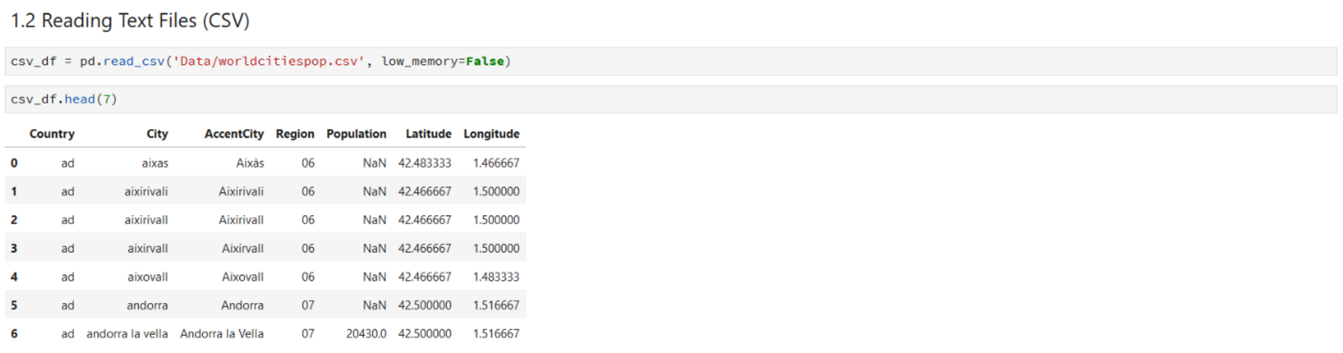
1.1 Creating DataFrames from Scratch

```
team = ['Real Madrid', 'Roma', 'Liverpool', 'Bayern Munchen']
champions_league_cups = [12,0, 5, 5]
established = [1902, 1927, 1892, 1900]

df = pd.DataFrame({'teams' : team, 'cups' : champions_league_cups, 'Year': established},  columns=['teams','cups', 'Year'],index=range(1,5))
df
```

	teams	cups	Year
1	Real Madrid	12	1902
2	Roma	0	1927
3	Liverpool	5	1892
4	Bayern Munchen	5	1900

Creating DataFrame from Scratch

However, most of the time, you might read a text file. So let us how to read different types of data in Pandas. First, we can read CSV Files in pandas by using the *pd.read_csv()*.



1.2 Reading Text Files (CSV)

```
csv_df = pd.read_csv('Data/worldcitiespop.csv', low_memory=False)
```

```
csv_df.head(7)
```

	Country	City	AccentCity	Region	Population	Latitude	Longitude
0	ad	aixas	Aixàs	06	NaN	42.483333	1.466667
1	ad	aixirivali	Aixirivali	06	NaN	42.466667	1.500000
2	ad	aixirivall	Aixirivall	06	NaN	42.466667	1.500000
3	ad	aixirvall	Aixirvall	06	NaN	42.466667	1.500000
4	ad	aixovall	Aixovall	06	NaN	42.466667	1.483333
5	ad	andorra	Andorra	07	NaN	42.500000	1.516667
6	ad	andorra la vella	Andorra la Vella	07	20430.0	42.500000	1.516667

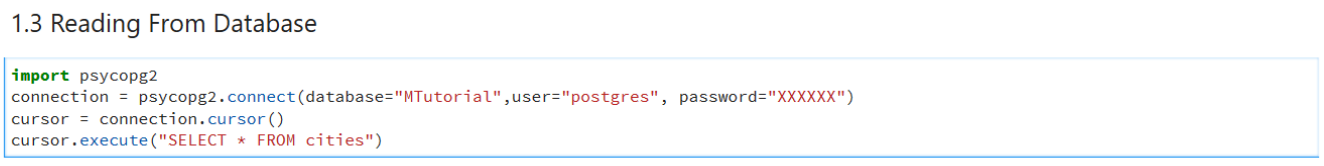
Reading CSV files

In order to successfully load the data, you need to know the directory of the data. In this case we have used this dataset of world cities and thier population.Download the data, create a directory and give it a name (in the example above, the directory is *Data*, and the name of the csv file is *worldcitiespop.csv.*

As you can see, we have saved our data as csv_df and to display your data you can simply type the variable name (csv_df), but in this case we have only showed the first 7 rows by using, csv_df.head() method.

Although CSV files are most likely the data format that you will use most often, there are at times that you might need to read data from other sources. Pandas has many options to read different data formats. You can read Excel files, JSON files, directly from HTML on the web as well as many other types. They are almost like the method we have seen above (reading CSV files). But, let us take another example of reading data from a database. I use here PostgreSQL, but other databases would work the same.

Before reading the database tables into Pandas DataFrame, you need to set up the connection. Here we use *psycog2* and create *connection* by calling psycog2.connect() with the database name, user and password. Then we create *cursor and execute* it using SQL query by selecting all attributes from a table called cities.

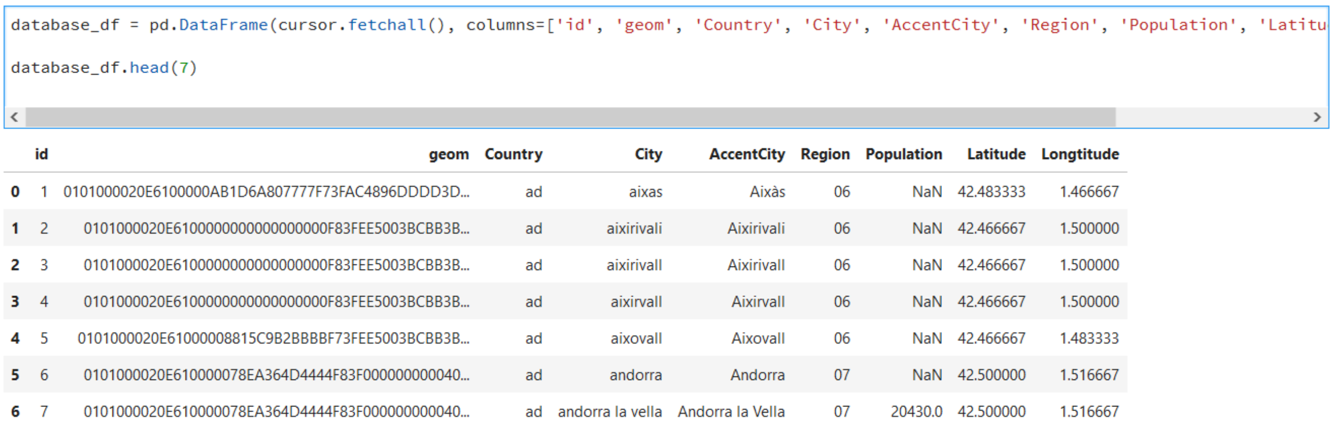


1.3 Reading From Database

```
import psycopg2
connection = psycopg2.connect(database="MTutorial",user="postgres", password="XXXXXX")
cursor = connection.cursor()
cursor.execute("SELECT * FROM cities")
```

Database Connection

You can now create Pandas DataFrame by fetching the *cursor* created above. We also add optional parameter *columns* to name our DataFrame columns.



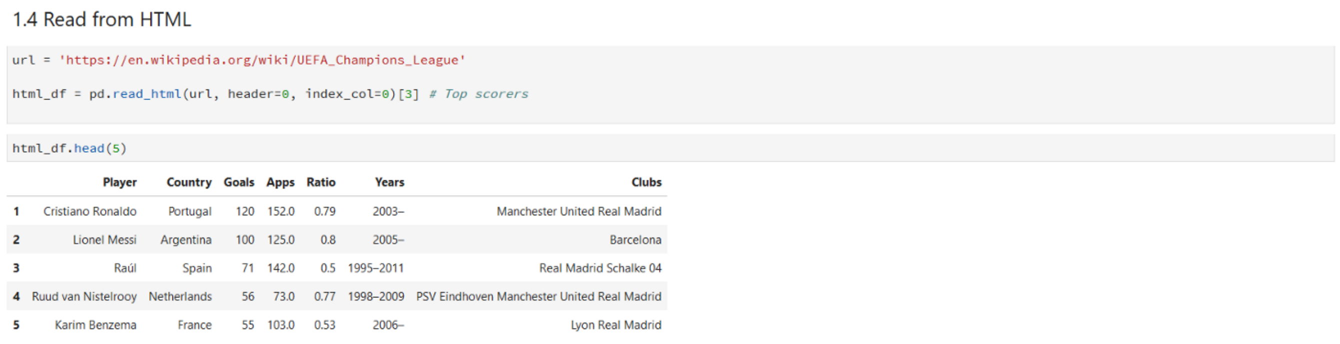
```
database_df = pd.DataFrame(cursor.fetchall(), columns=['id', 'geom', 'Country', 'City', 'AccentCity', 'Region', 'Population', 'Latitu

database_df.head(7)
```

	id	geom	Country	City	AccentCity	Region	Population	Latitude	Longtitude
0	1	0101000020E6100000AB1D6A807777F73FAC4896DDDD3D...	ad	aixas	Aixàs	06	NaN	42.483333	1.466667
1	2	0101000020E610000000000000000000F83FEE5003BCBB3B...	ad	aixirivali	Aixirivali	06	NaN	42.466667	1.500000
2	3	0101000020E610000000000000000000F83FEE5003BCBB3B...	ad	aixirivall	Aixirivall	06	NaN	42.466667	1.500000
3	4	0101000020E610000000000000000000F83FEE5003BCBB3B...	ad	aixirvall	Aixirvall	06	NaN	42.466667	1.500000
4	5	0101000020E61000008815C9B2BBBBF73FEE5003BCBB3B...	ad	aixovall	Aixovall	06	NaN	42.466667	1.483333
5	6	0101000020E610000078EA364D4444F83F000000000040...	ad	andorra	Andorra	07	NaN	42.500000	1.516667
6	7	0101000020E610000078EA364D4444F83F000000000040...	ad	andorra la vella	Andorra la Vella	07	20430.0	42.500000	1.516667

Read from Database

Lastly, reading data directly from websites is not always straightforward and you need to do some web scraping, but Pandas makes it easy to read HTML directly into

DataFrames. Here we will read data directly from Wikipedia to a DataFrame by using *pd.read_html()*



1.4 Read from HTML

```
url = 'https://en.wikipedia.org/wiki/UEFA_Champions_League'

html_df = pd.read_html(url, header=0, index_col=0)[3] # Top scorers
```

```
html_df.head(5)
```

	Player	Country	Goals	Apps	Ratio	Years	Clubs
1	Cristiano Ronaldo	Portugal	120	152.0	0.79	2003–	Manchester United Real Madrid
2	Lionel Messi	Argentina	100	125.0	0.8	2005–	Barcelona
3	Raúl	Spain	71	142.0	0.5	1995–2011	Real Madrid Schalke 04
4	Ruud van Nistelrooy	Netherlands	56	73.0	0.77	1998–2009	PSV Eindhoven Manchester United Real Madrid
5	Karim Benzema	France	55	103.0	0.53	2006–	Lyon Real Madrid

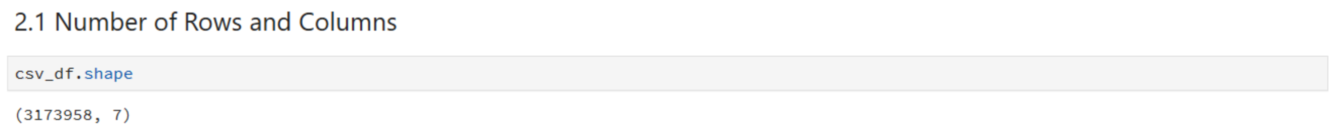
Read from HTML

• • •

2. Basic Data Exploration and Descriptive statistics

One fundamental routine in data science is an early sanity checking of the data at hand. Here are some basic functions that will allow you to have a quick look at your data and find out discrepancies.

I normally look first the number of rows and columns in my data like this:



2.1 Number of Rows and Columns

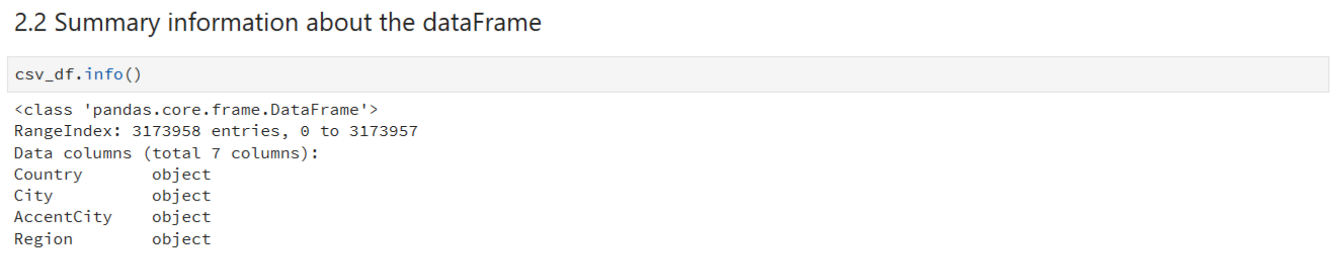
```
csv_df.shape
```

```
(3173958, 7)
```

Number of Rows and columns

csv_df.shape returns a tuple containing (rows, columns). In this example, we have 3173958 rows and 7 columns. In case you want to access only one of them, rows are index 0 and columns are at index 1.

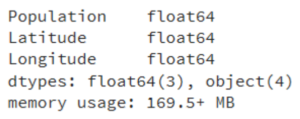
Second, you might use *csv_df.info()* command to see the types of columns and some other information about the DataFrame.



2.2 Summary information about the dataFrame

```
csv_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3173958 entries, 0 to 3173957
Data columns (total 7 columns):
Country       object
City          object
AccentCity    object
Region        object
```

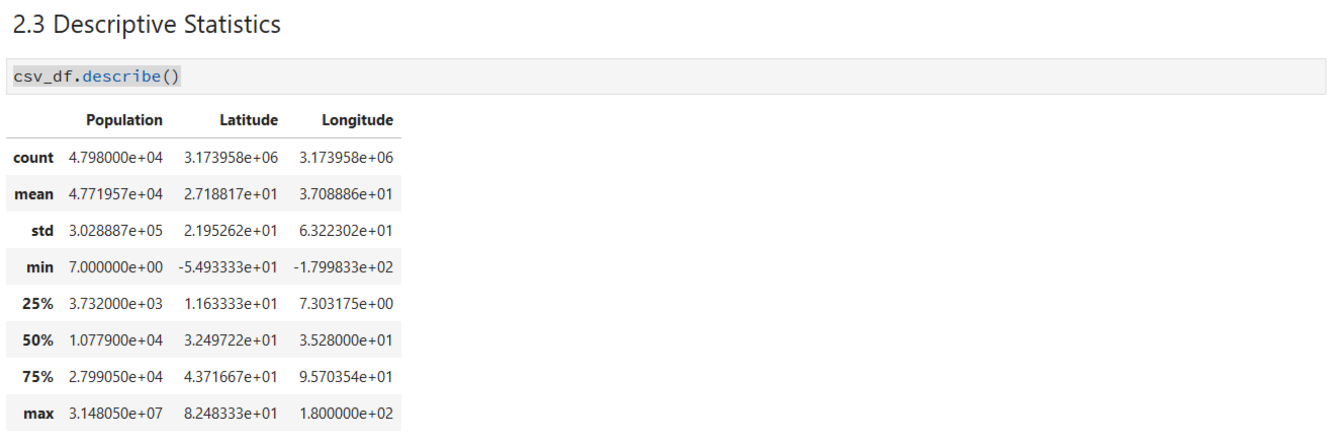


```
Population    float64
Latitude      float64
Longitude     float64
dtypes: float64(3), object(4)
memory usage: 169.5+ MB
```

DataFrame informaiton

Here, *csv_df.info()* returns information about the columns and thier types as well as range of the index. In this particular data, 3 of the 7 columns are Float type while the other 4 are objects.

If you want to have a quick look at some descriptive statistics of your data, look no further than the *.describe()* method.



2.3 Descriptive Statistics

```
csv_df.describe()
```

	Population	Latitude	Longitude
count	4.798000e+04	3.173958e+06	3.173958e+06
mean	4.771957e+04	2.718817e+01	3.708886e+01
std	3.028887e+05	2.195262e+01	6.322302e+01
min	7.000000e+00	-5.493333e+01	-1.799833e+02
25%	3.732000e+03	1.163333e+01	7.303175e+00
50%	1.077900e+04	3.249722e+01	3.528000e+01
75%	2.799050e+04	4.371667e+01	9.570354e+01
max	3.148050e+07	8.248333e+01	1.800000e+02

Descriptive statistics

Remember that we had only 3 float columns in our dataset. Well, you have some descriptive statistics about them including their count, mean, maximum, minimum, etc.

Well, you can see the maximum population of a city is 31 million, but which city is it? The next part will deal with data manipulation, transformation and cleaning. To whet your appetite, here is the code to get the city with the maximum population.

```
csv_df.loc[csv_df['Population'].argmax()][['City', 'Population']]
```

```
City                tokyo
Population    3.14805e+07
Name: 1544449, dtype: object
```

Stay tuned for the next part. You will learn data cleaning, one of the most crucial tasks in data science.

The Jupyter notebook is availabe at this link.

Data Science