2.1-Pandas Introduction

KeytoDataScience.com

Introduction to Pandas in Python

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on the top of the NumPy library. Pandas is fast and it has high-performance & productivity for users.

Table of Contents

1 Advantages

Fast and efficient for manipulating and analyzing data.

Data from different file objects can be loaded.

Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data

Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects

Data set merging and joining.

Flexible reshaping and pivoting of data sets

Provides time-series functionality.

Powerful group by functionality for performing split-apply-combine operations on data sets.

↥ back to top

2 Downloading and Installing Pandas

Pandas can be installed in multiple ways on Windows and on Linux.

Windows: Python Pandas can be installed on Windows in two ways:

- Using pip
- Using Anaconda

Install Pandas using pip: PIP is a package management system used to install and manage software packages/libraries written in Python. These files are stored in a large "on-line repository" termed as Python Package Index (PyPI).

Pandas can be installed using PIP by the use of the following command:

In [1]:
```
!pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\prateek\appdata\roaming\python\python3
9\site-packages (1.3.5)
Requirement already satisfied: pytz>=2017.3 in c:\users\prateek\appdata\local\programs\p
ython\python39\lib\site-packages (from pandas) (2021.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\prateek\appdata\local
\programs\python\python39\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\prateek\appdata\local\programs
\python\python39\lib\site-packages (from pandas) (1.21.2)
Requirement already satisfied: six>=1.5 in c:\users\prateek\appdata\local\programs\pytho
n\python39\lib\site-packages (from python-dateutil>=2.7.3->pandas) (1.16.0)
```

↥ back to top

💡💡💡💡💡 **Getting Started** 💡💡💡💡💡

After the pandas has been installed into the system, you need to import the library. This module is generally imported as –

In [2]:
```
import pandas as pd
```

Pandas generally provide two data structures for manipulating data, They are:

- Series

- DataFrame

↥ back to top

3 Series

Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index. Pandas Series is nothing but a column in an excel sheet. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

A Series is very similar to a NumPy array (in fact it is built on top of the NumPy array object). What differentiates the NumPy array from a Series, is that a Series can have axis labels, meaning it can be indexed by a label, instead of just a number location. It also doesn't need to hold numeric data, it can hold any arbitrary Python Object.

Let's explore this concept through some examples:

3.1 Creating a Series

In the real world, a Pandas Series will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas Series can be created from the lists, dictionary, and from a scalar value etc.

Example:

In [3]:
```
import pandas as pd
import numpy as np

# Creating empty series
ser = pd.Series()

print(ser)

# simple array
data = np.array(['K','e','y','t','o','D','a','t','a','S','c','i','e','n','c','e'])

ser = pd.Series(data)
print(ser)
```

```
Series([], dtype: float64)
0    K
1    e
2    y
3    t
4    o
5    D
6    a
7    t
8    a
9    S
```

```
10    c
11    i
12    e
13    n
14    c
15    e
dtype: object
```

```
C:\Users\Prateek\AppData\Local\Temp/ipykernel_13336/358935292.py:5: DeprecationWarning:
The default dtype for empty Series will be 'object' instead of 'float64' in a future ver
sion. Specify a dtype explicitly to silence this warning.
  ser = pd.Series()
```

In [4]:
```
labels = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p']
pd.Series(data=data,index=labels)
```

Out[4]:
```
a    K
b    e
c    y
d    t
e    o
f    D
g    a
h    t
i    a
j    S
k    c
l    i
m    e
n    n
o    c
p    e
dtype: object
```

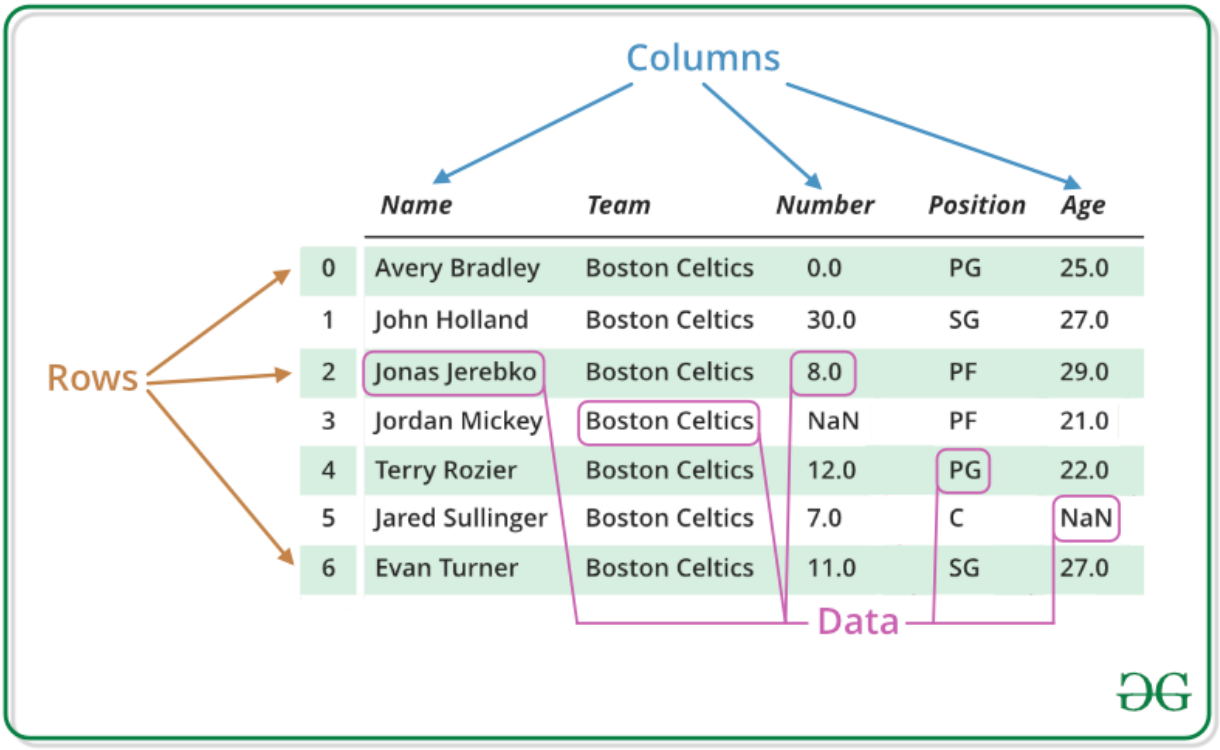
In [5]:
```
# Even functions (although unlikely that you will use this)
pd.Series([sum,print,len])
```

Out[5]:
```
0      <built-in function sum>
1    <built-in function print>
2      <built-in function len>
dtype: object
```

↥ back to top

4 DataFrame

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.



4.1 Creating a DataFrame

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc.

Read more: https://keytodatascience.com/create-pandas-dataframe-python/

In [6]:
```
import pandas as pd

# Calling DataFrame constructor
df = pd.DataFrame()
print(df)

# list of strings
lst = ['Key', 'to', 'Data', 'Science',
            'is', 'Awesome']

# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
df
```

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

Out[6]:

	0
0	Key
1	to
2	Data

	0
3	Science
4	is
5	Awesome

Creating DataFrame from dict of ndarray/lists:

To create DataFrame from dict of narray/list, all the narray must be of same length. If index is passed then the length index should be equal to the length of arrays. If no index is passed, then by default, index will be range(n) where n is the array length.

In [7]:
```
# Python code demonstrate creating
# DataFrame from dict narray / lists
# By default addresses.

import pandas as pd

# intialise data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'],
        'Age':[20, 21, 19, 18]}

# Create DataFrame
df = pd.DataFrame(data)

# Print the output.
print(df)
df
```

```
    Name  Age
0    Tom   20
1   nick   21
2  krish   19
3   jack   18
```

Out[7]:

	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

Explore various ways to create DataFrame using inputs like:

- Dictionary
- Lists
- Series
- Numpy arrays
- Empty DataFrame

↥ back to top

5 Working With Text Data

In [8]:
```
# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

df
```

Out[8]:

	Name	Age	Address	Qualification
0	Jai	27	Delhi	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannauj	Phd

5.1 Convert column text to lowercase

In [9]:
```
# converting and overwriting values in name column
df["Name"]= df["Name"].str.lower()

df
```

Out[9]:

	Name	Age	Address	Qualification
0	jai	27	Delhi	Msc
1	princi	24	Kanpur	MA
2	gaurav	22	Allahabad	MCA
3	anuj	32	Kannauj	Phd

5.2 Splitting and Replacing a Data

In order to split a data, we use str.split() this function returns a list of strings after breaking the given string by the specified separator but it can only be applied to an individual string. Pandas str.split() method can be applied to a whole series. .str has to be prefixed every time before calling this method to differentiate it from the Python's default function otherwise, it will throw an error.

In order to replace a data, we use str.replace() this function works like Python .replace() method only, but it works on Series too. Before calling .replace() on a Pandas series, .str has to be prefixed in order to differentiate it from the Python's default replace method.

In [10]:
```
# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
```

```
        'Age':[27, 24, 22, 32],
        'Address':['Nagpur', 'Kanpur', 'Allahabad', 'Una'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

df
```

Out[10]:

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Una	Phd

In [11]:

```
# dropping null value columns to avoid errors
df.dropna(inplace = True)

# new data frame with split value columns
nw= df["Address"].str.split("a", n = 1, expand = True)

# df display
nw
```

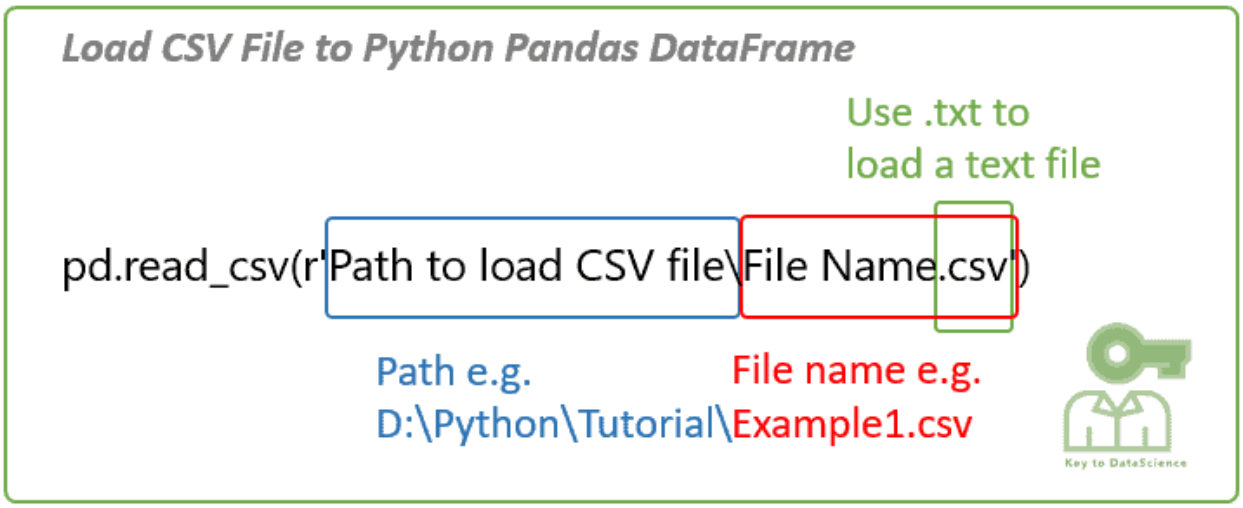
Out[11]:

	0	1
0	N	gpur
1	K	npur
2	All	habad
3	Un	

↥ back to top

6 Reading a file in Pandas

Load csv/text file using read_csv()



Learn More Here - KeytoDataScience

- Pandas read_csv() Syntax
- Read CSV file using Pandas (Example)
- Common Errors and Troubleshooting

In [12]:

```
# importing pandas module
import pandas as pd

# making data frame
df = pd.read_csv("https://media.geeksforgeeks.org/wp-content/uploads/nba.csv")

df.head(15)
```

Out[12]:

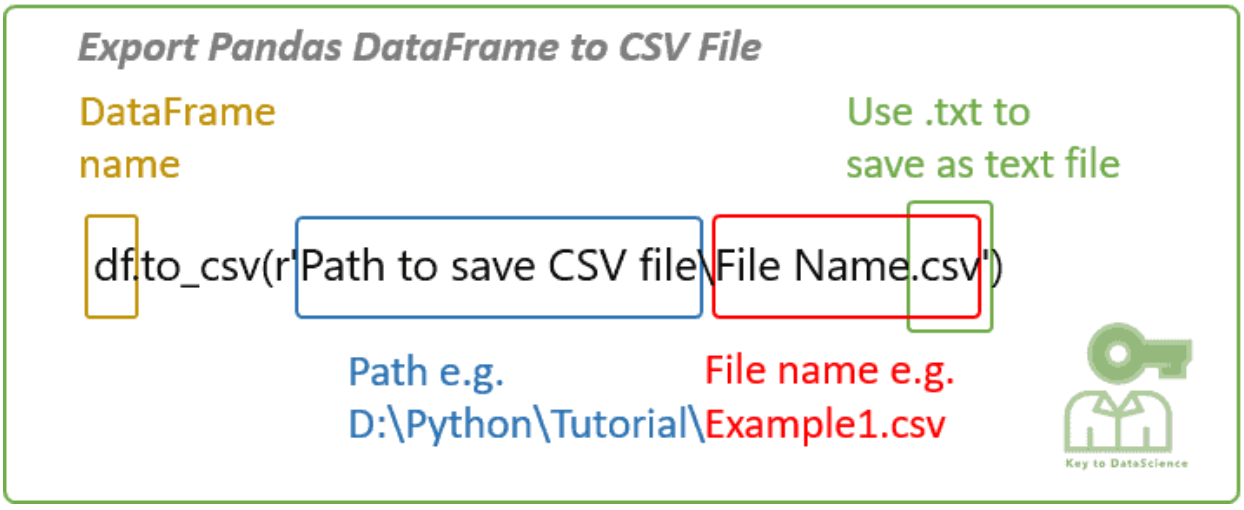
	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0
10	Jared Sullinger	Boston Celtics	7.0	C	24.0	6-9	260.0	Ohio State	2569260.0
11	Isaiah Thomas	Boston Celtics	4.0	PG	27.0	5-9	185.0	Washington	6912869.0
12	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0
13	James Young	Boston Celtics	13.0	SG	20.0	6-6	215.0	Kentucky	1749840.0
14	Tyler Zeller	Boston Celtics	44.0	C	26.0	7-0	253.0	North Carolina	2616975.0

↥ back to top

7 Saving a Pandas Dataframe as a CSV

Export/Save Pandas DataFrame to CSV File



In [29]:
```
df.to_csv('file1.csv')
```

Learn More Here - KeytoDataScience

- Pandas DataFrame to_csv() Syntax
- Write Pandas DataFrame to a CSV file (Explained)

↥ back to top

8 Reading an Excel file (.xlsx) in Pandas

To read Excel (.xlsx) file we have to install openpyxl. Run either of the following command:

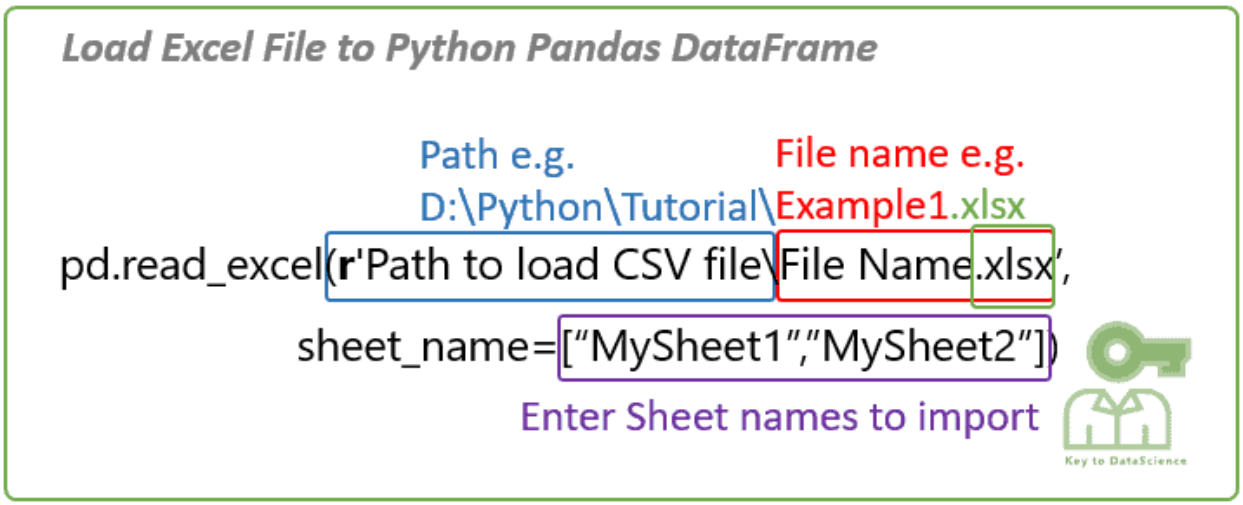
- pip install openpyxl

- conda install openpyxl

In [30]:
```
pip install openpyxl
```

```
Collecting openpyxl
  Downloading openpyxl-3.0.9-py2.py3-none-any.whl (242 kB)
Collecting et-xmlfile
  Downloading et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)
Installing collected packages: et-xmlfile, openpyxl
Successfully installed et-xmlfile-1.1.0 openpyxl-3.0.9
Note: you may need to restart the kernel to use updated packages.
```

Import/Read Excel File using Pandas



We will explore read_excel() function possibilities below:

Method #1: Read by default 1st sheet of an excel file

In [13]:
```
# read by default 1st sheet of an excel file
dataframe1 = pd.read_excel('SampleWork.xlsx')

print(dataframe1)
```

```
        Name  Age    Stream  Percentage
0      Ankit   18      Math          95
1      Rahul   19   Science          90
2    Shourya   20  Commerce          85
3  Aishwarya   18      Math          80
4   Priyanka   19   Science          75
```

Method #2: Read Specific Sheets using 'sheet_name'

In [14]:
```
# read 2nd sheet of an excel file by passing sheet name as INTEGER
dataframe2 = pd.read_excel('SampleWork.xlsx', sheet_name = 0)
# Remember: First sheet index is 0, not 1

print(dataframe2)
```

```
        Name  Age    Stream  Percentage
0      Ankit   18      Math          95
1      Rahul   19   Science          90
2    Shourya   20  Commerce          85
```

```
3  Aishwarya   18      Math          80
4   Priyanka   19   Science          75
```

In [15]:
```
# read 2nd sheet of an excel file y passing sheet name as NAME
dataframe2 = pd.read_excel('SampleWork.xlsx', sheet_name = 'Data')

print(dataframe2)
```

```
         Name  Age    Stream  Percentage
0      Ankit1   18      Math          95
1      Rahul1   19   Science          90
2    Shourya1   20  Commerce          85
3  Aishwarya1   18      Math          80
4   Priyanka1   19   Science          75
```

Method #3: Read only specific columns from an excel file

In [16]:
```
require_cols = [0, 3]

# only read specific columns from an excel file
required_df = pd.read_excel('SampleWork.xlsx', usecols = require_cols)

print(required_df)
```

```
        Name  Percentage
0      Ankit          95
1      Rahul          90
2    Shourya          85
3  Aishwarya          80
4   Priyanka          75
```

Method #4: Skip starting rows when Reading an Excel File

Using 'skiprows' parameter of read_excel() method

In [17]:
```
# read 2nd sheet of an excel file after
# skipping starting two rows
df = pd.read_excel('SampleWork.xlsx', sheet_name = 1, skiprows = 2)

print(df)
```

```
      Rahul2  19   Science    90
0   Shourya2  20  Commerce  85.0
1  Aishwara2  18      Math  80.0
2  Priyanka2  19   Science   NaN
```

Method #5: Set the header to any row and start reading from that row

Using 'header' parameter of the read_excel() method.

In [18]:
```
# setting the 3rd row as header.
df = pd.read_excel('SampleWork.xlsx', sheet_name = 1, header = 0)

print(df)
```

```
        Name  Age    Stream  Percentage
0     Ankit2   18      Math        95.0
```

```
1     Rahul2   19   Science        90.0
2   Shourya2   20  Commerce        85.0
3  Aishwara2   18      Math        80.0
4  Priyanka2   19   Science         NaN
```

Method #6: Reading Multiple Excel Sheets

Using 'sheet_name' parameter of the read_excel()method.

In [19]:
```
# read both 1st and 2nd sheet.
df = pd.read_excel('SampleWork.xlsx',  sheet_name =[0, 1])

print(df)
```

```
{0:         Name  Age    Stream  Percentage
0      Ankit   18      Math          95
1      Rahul   19   Science          90
2    Shourya   20  Commerce          85
3  Aishwarya   18      Math          80
4   Priyanka   19   Science          75, 1:         Name  Age    Stream  Percentage
0     Ankit2   18      Math        95.0
1     Rahul2   19   Science        90.0
2   Shourya2   20  Commerce        85.0
3  Aishwara2   18      Math        80.0
4  Priyanka2   19   Science         NaN}
```

In [20]:
```
# Select dataframe of first sheet
df[0]
```

Out[20]:

	Name	Age	Stream	Percentage
0	Ankit	18	Math	95
1	Rahul	19	Science	90
2	Shourya	20	Commerce	85
3	Aishwarya	18	Math	80
4	Priyanka	19	Science	75

In [21]:
```
# Select dataframe of second sheet
df[1]
```

Out[21]:

	Name	Age	Stream	Percentage
0	Ankit2	18	Math	95.0
1	Rahul2	19	Science	90.0
2	Shourya2	20	Commerce	85.0
3	Aishwara2	18	Math	80.0
4	Priyanka2	19	Science	NaN

Learn More Here - KeytoDataScience

- Python Pandas read_excel() Syntax
- Import Excel file using Python Pandas (Example)
- read_excel Important Parameters Examples
- Import Specific Excel Sheet using sheet name
- Import Multiple Excel Sheets Pandas
- Import only n Rows of Excel Sheet
- Import specific columns of Excel Sheet
- Common Errors and Troubleshooting

↥ back to top

9 View basic statistical details

In [22]:
```
# importing pandas module
import pandas as pd

# making data frame
df = pd.read_csv("https://media.geeksforgeeks.org/wp-content/uploads/nba.csv")

df.head(5)
```

Out[22]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0

9.1 describe()

Pandas **describe()** is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values. When this method is applied to a series of string, it returns a different output which is shown in the examples below.

In [23]:
```
# percentile list
perc =[.20, .40, .60, .80]

# list of dtypes to include
include =['object', 'float', 'int']

# calling describe method
desc = df.describe(percentiles = perc, include = include)
desc
```

Out[23]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
count	457	457	457.000000	457	457.000000	457	457.000000	373	4.460000e+02
unique	457	30	NaN	5	NaN	18	NaN	118	NaN
top	Avery Bradley	New Orleans Pelicans	NaN	SG	NaN	6-9	NaN	Kentucky	NaN
freq	1	19	NaN	102	NaN	59	NaN	22	NaN
mean	NaN	NaN	17.678337	NaN	26.938731	NaN	221.522976	NaN	4.842684e+06
std	NaN	NaN	15.966090	NaN	4.404016	NaN	26.368343	NaN	5.229238e+06
min	NaN	NaN	0.000000	NaN	19.000000	NaN	161.000000	NaN	3.088800e+04
20%	NaN	NaN	4.000000	NaN	23.000000	NaN	195.600000	NaN	9.472760e+05
40%	NaN	NaN	10.000000	NaN	25.000000	NaN	213.400000	NaN	1.938840e+06
50%	NaN	NaN	13.000000	NaN	26.000000	NaN	220.000000	NaN	2.839073e+06
60%	NaN	NaN	18.600000	NaN	27.000000	NaN	230.000000	NaN	3.815000e+06
80%	NaN	NaN	30.000000	NaN	30.000000	NaN	245.000000	NaN	8.042895e+06
max	NaN	NaN	99.000000	NaN	40.000000	NaN	307.000000	NaN	2.500000e+07

In [24]:
```
df['College'].describe()
```

Out[24]:
```
count          373
unique         118
top       Kentucky
freq            22
Name: College, dtype: object
```

9.2 cut()

Pandas cut() function is used to separate the array elements into different bins . The cut function is mainly used to perform statistical analysis on scalar data.

In [25]:
```
import numpy as np

df= pd.DataFrame({'number': np.random.randint(1, 100, 20)})
df['bins'] = pd.cut(x=df['number'], bins=[1, 20, 40, 60,
                                          80, 100])
print(df)


# We can check the frequency of each bin
print(df['bins'].unique())
```

```
    number       bins
0       36   (20, 40]
1       14    (1, 20]
2       44   (40, 60]
```

```
3       47   (40, 60]
4       81  (80, 100]
5       25   (20, 40]
6       91  (80, 100]
7       70   (60, 80]
8       65   (60, 80]
9       18    (1, 20]
10      81  (80, 100]
11       9    (1, 20]
12       8    (1, 20]
13      26   (20, 40]
14      99  (80, 100]
15      82  (80, 100]
16      28   (20, 40]
17      34   (20, 40]
18      41   (40, 60]
19      38   (20, 40]
[(20, 40], (1, 20], (40, 60], (80, 100], (60, 80]]
Categories (5, interval[int64, right]): [(1, 20] < (20, 40] < (40, 60] < (60, 80] < (80,
100]]
```

↥ back to top

Great Job!
