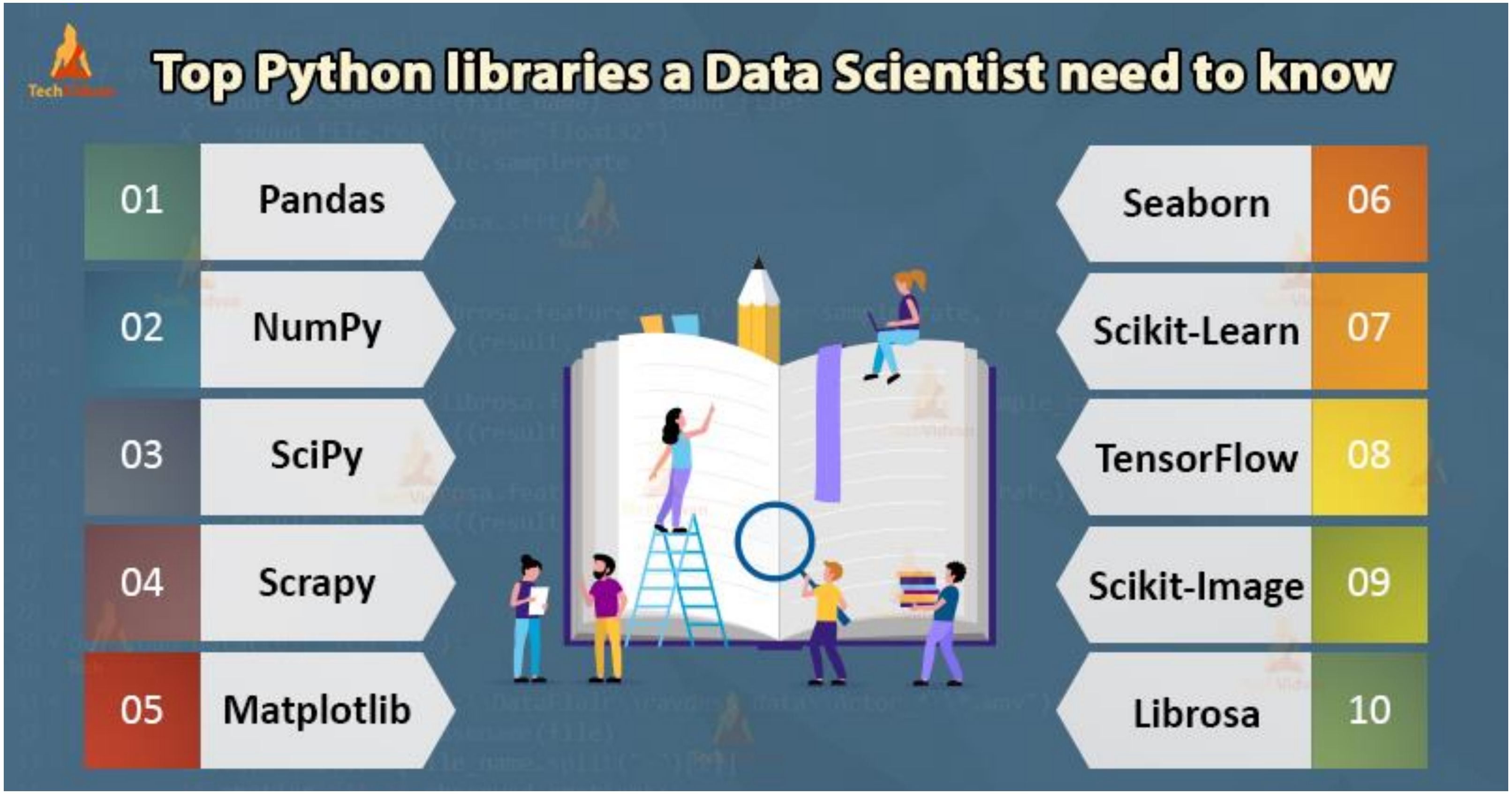
DATA ANALYTICS WITH PYTHON

Data Manipulation with Pandas

Peter Lo

Top Python Libraries for Data Science



What is Pandas?

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

□ Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Key Features of Pandas

- Fast and efficient DataFrame object with default and customized indexing.
- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of date sets.
- Label-based slicing, indexing and subsetting of large data sets.
- Columns from a data structure can be deleted or inserted.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.
- Time Series functionality.

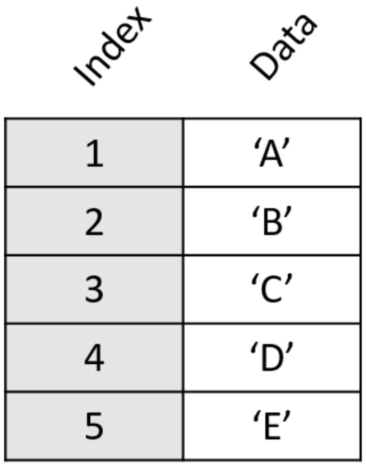
Data Structures

- Pandas deals with three data structures:

Data Structure	Dimension	Description
Series	1	1D labeled homogeneous array, size immutable
Data Frames	2	General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns
Panel	3	General 3D labeled, size-mutable array

Series

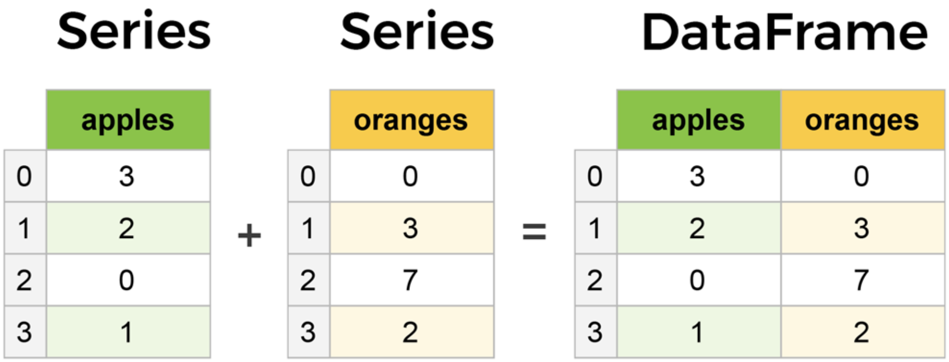
- A one-dimensional labeled array capable of holding any data type.
- A Series object has two main components: Index and Data
- Both components are one-dimensional arrays with the same length. The index should be made up of unique elements, and it is used to access individual data values



Index	Data
1	‘A’
2	‘B’
3	‘C’
4	‘D’
5	‘E’

DataFrame

- DataFrame is a 2-dimensional labeled data structure with columns of potentially different types.
- You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input.



Series

	apples
0	3
1	2
2	0
3	1

+

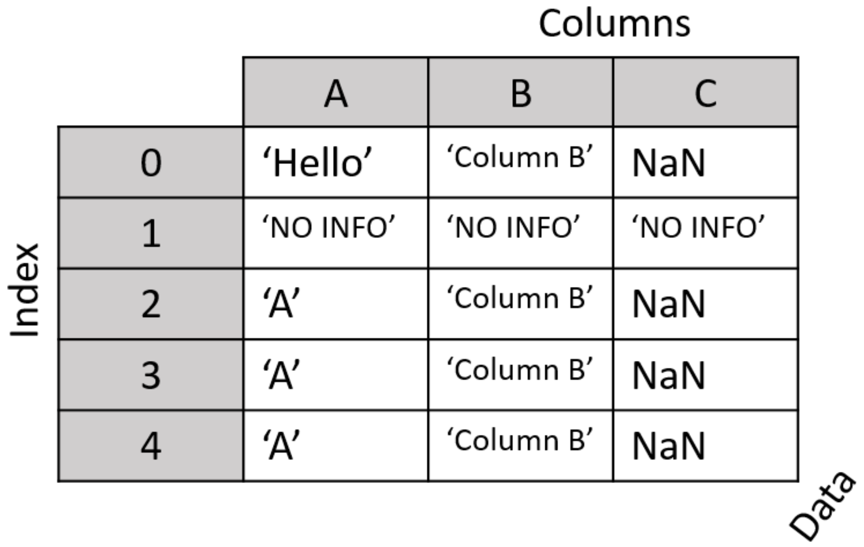
Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

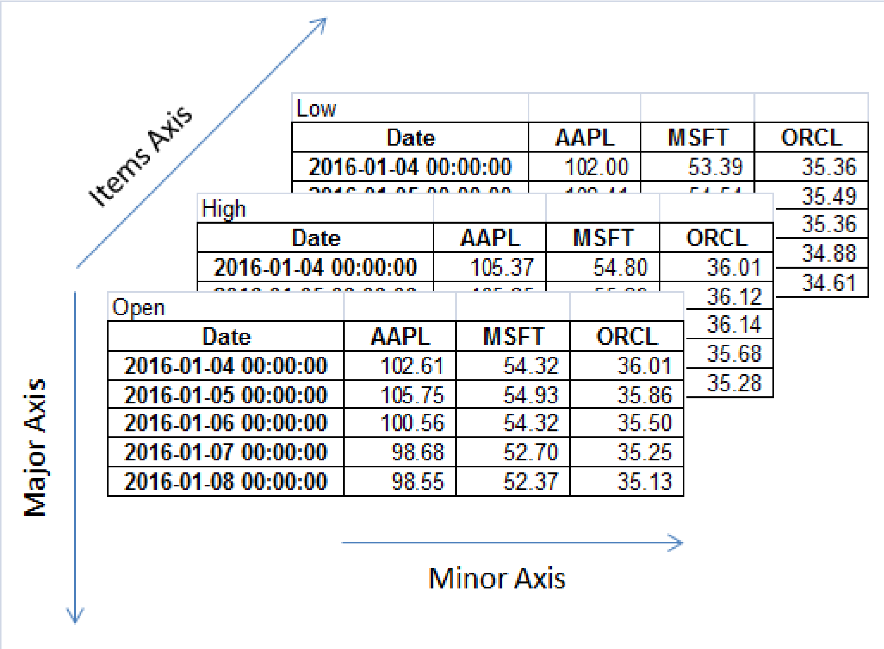
	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2



	A	B	C
0	‘Hello’	‘Column B’	NaN
1	‘NO INFO’	‘NO INFO’	‘NO INFO’
2	‘A’	‘Column B’	NaN
3	‘A’	‘Column B’	NaN
4	‘A’	‘Column B’	NaN

Panel

- □ A panel is a 3D container of data. It is the natural extension of the DataFrame and can be seen as a 3D table, or a collection of multiple DataFrames.



Low

Date	AAPL	MSFT	ORCL
2016-01-04 00:00:00	102.00	53.39	35.36
			35.49
			35.36
			34.88
			34.61

High

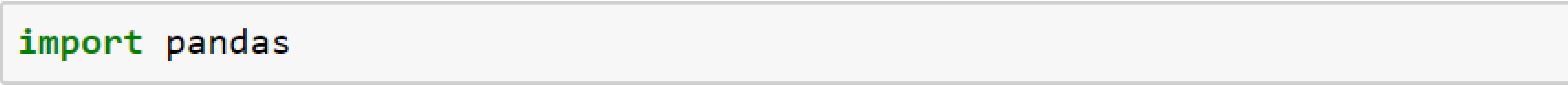
Date	AAPL	MSFT	ORCL
2016-01-04 00:00:00	105.37	54.80	36.01
			36.12
			36.14
			35.68
			35.28

Open

Date	AAPL	MSFT	ORCL
2016-01-04 00:00:00	102.61	54.32	36.01
2016-01-05 00:00:00	105.75	54.93	35.86
2016-01-06 00:00:00	100.56	54.32	35.50
2016-01-07 00:00:00	98.68	52.70	35.25
2016-01-08 00:00:00	98.55	52.37	35.13

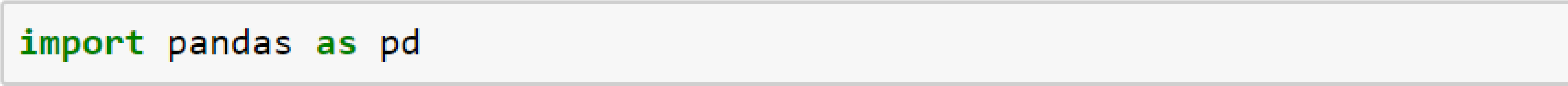
Using Pandas Module

- Before we can use Pandas we will have to import it. It has to be imported like any other module:



```
import pandas
```

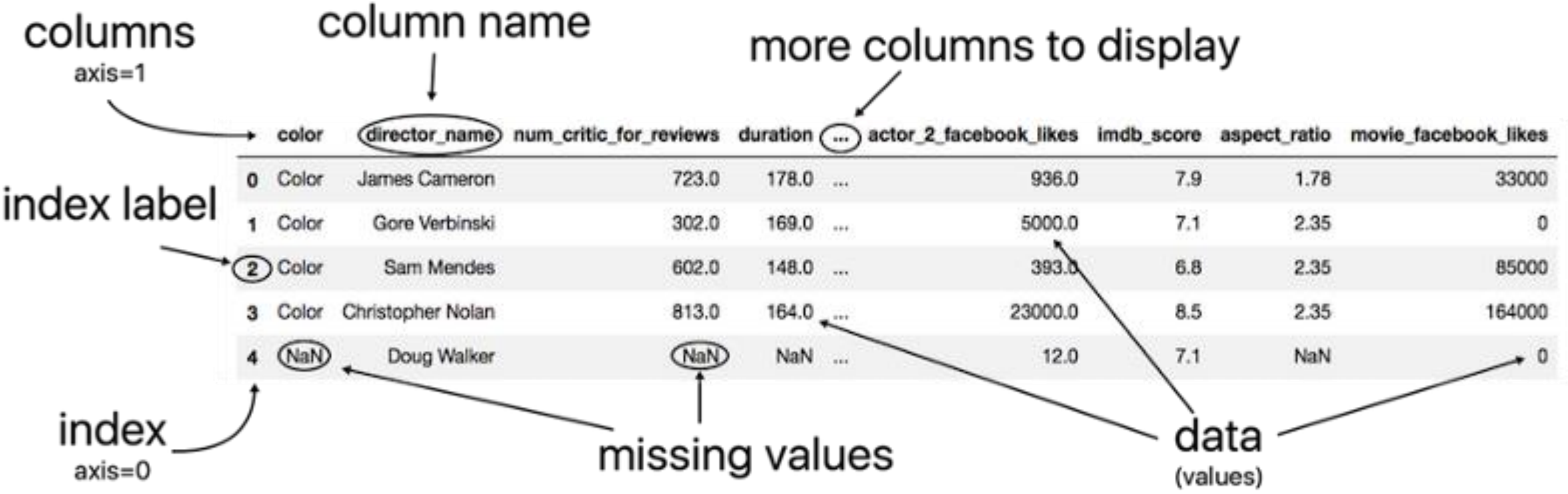
- But you will hardly ever see this. Pandas is usually renamed to pd:



```
import pandas as pd
```

Parsing CSV Files

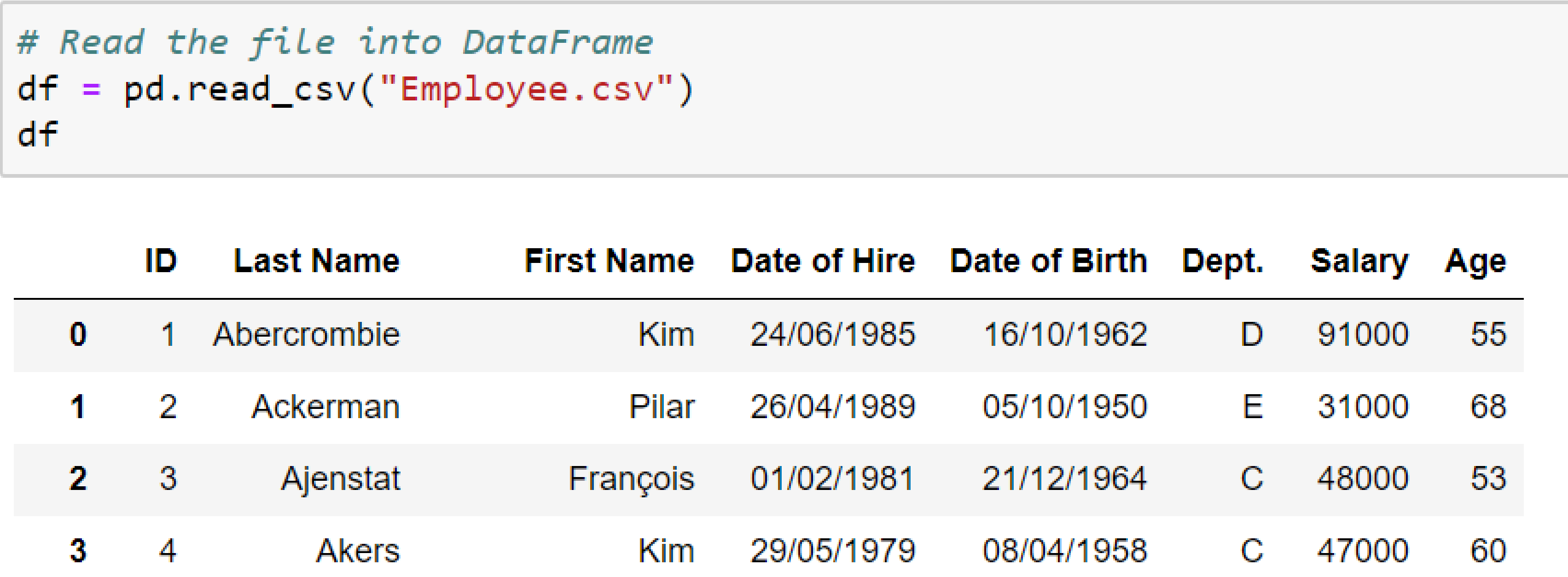
- Reading CSV files is very easy in pandas. It is highly recommended if you have a lot of data to analyze.
- pandas provides high performance data analysis tools and easy to use data structures.



	color	director_name	num_critic_for_reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
0	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
1	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
2	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
3	Color	Christopher Nolan	813.0	164.0	...	23000.0	8.5	2.35	164000
4	NaN	Doug Walker	NaN	NaN	...	12.0	7.1	NaN	0

Example

- The *read_csv()* method opens, analyzes, and reads the CSV file provided, and stores the data in a DataFrame
- pandas recognized that the first line of the CSV contained column names, and used them automatically.



```
# Read the file into DataFrame
df = pd.read_csv("Employee.csv")
df
```

	ID	Last Name	First Name	Date of Hire	Date of Birth	Dept.	Salary	Age
0	1	Abercrombie	Kim	24/06/1985	16/10/1962	D	91000	55
1	2	Ackerman	Pilar	26/04/1989	05/10/1950	E	31000	68
2	3	Ajenstat	François	01/02/1981	21/12/1964	C	48000	53
3	4	Akers	Kim	29/05/1979	08/04/1958	C	47000	60

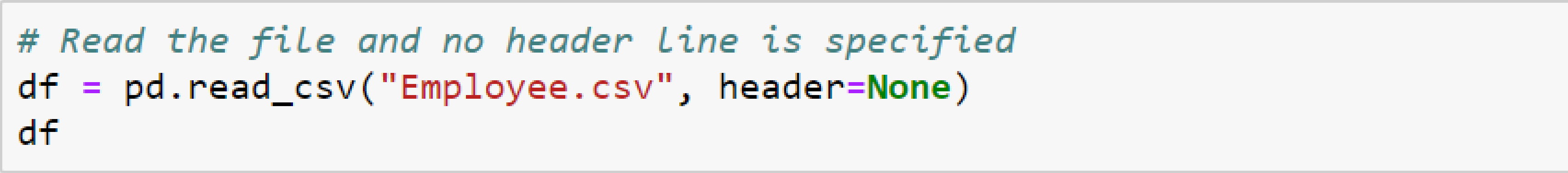
Parameters for read_csv

- There are several parameters that can be used to alter the way the data is read from file and formatted in the DataFrame.

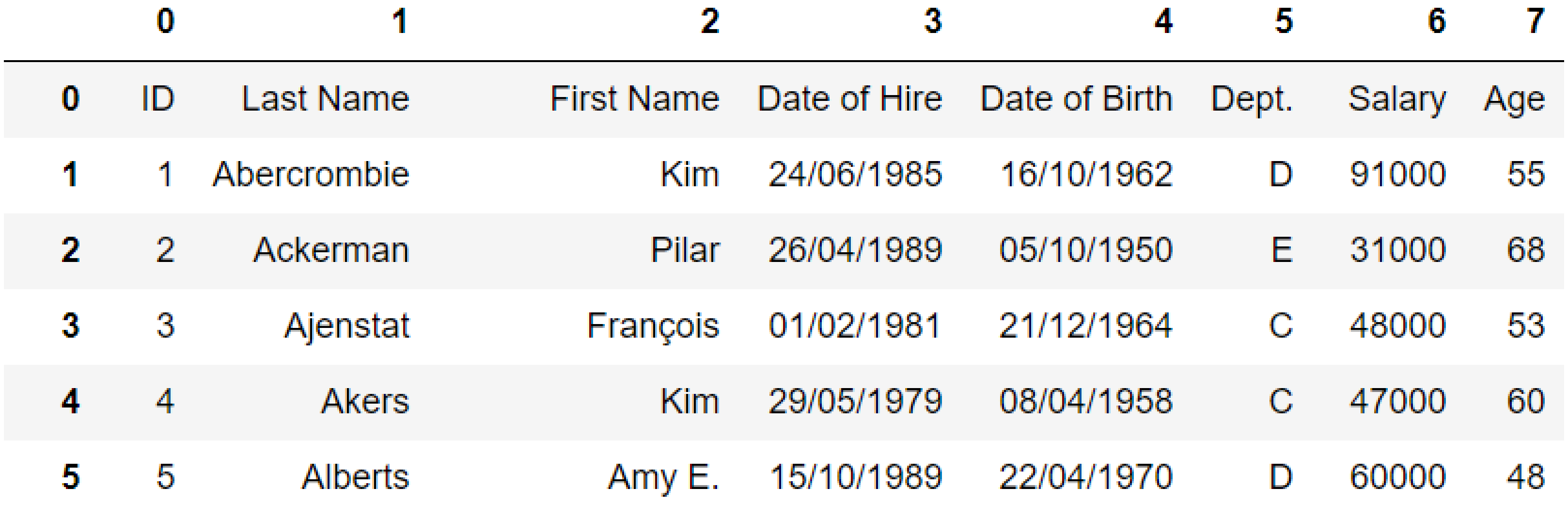
Parameter	Description
header	Row numbers to use as the column names, and the start of the data
nrows	Number of rows of file to read. Useful for reading pieces of large files
skiprows	Number of lines to skip at the start of the file.
usecols	Return a subset of the columns
Index_col	Column to use as the row labels of the DataFrame
skip_blank_lines	Skip over blank lines

The header Parameter

- If “header = None” is specified, first line also consider as data. This is useful for file without header



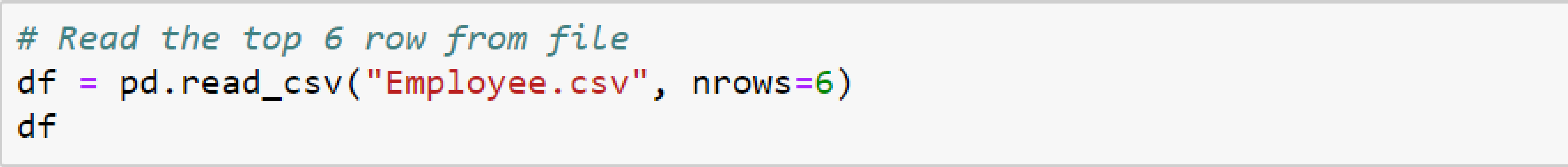
```
# Read the file and no header line is specified
df = pd.read_csv("Employee.csv", header=None)
df
```



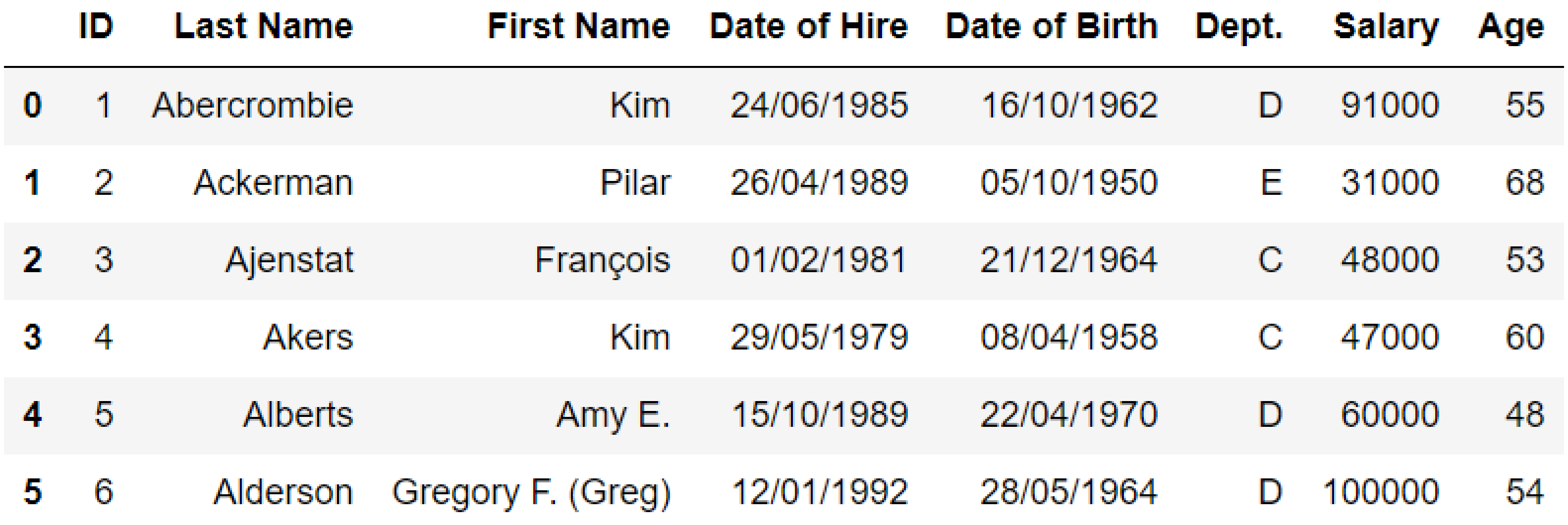
	0	1	2	3	4	5	6	7
0	ID	Last Name	First Name	Date of Hire	Date of Birth	Dept.	Salary	Age
1	1	Abercrombie	Kim	24/06/1985	16/10/1962	D	91000	55
2	2	Ackerman	Pilar	26/04/1989	05/10/1950	E	31000	68
3	3	Ajenstat	François	01/02/1981	21/12/1964	C	48000	53
4	4	Akers	Kim	29/05/1979	08/04/1958	C	47000	60
5	5	Alberts	Amy E.	15/10/1989	22/04/1970	D	60000	48

The nrow Parameter

- By adding the ***nrows*** parameter with an integer value, you can control the number of rows to be read.
- The rest of the data in the file is not imported.



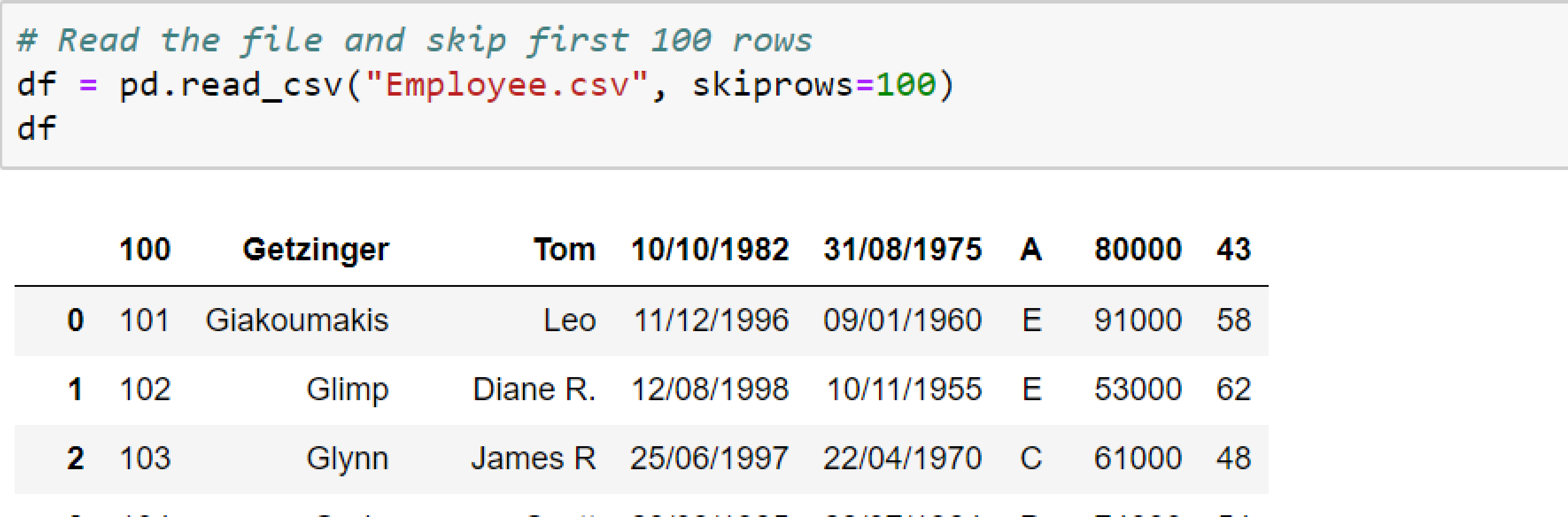
```
# Read the top 6 row from file
df = pd.read_csv("Employee.csv", nrows=6)
df
```



	ID	Last Name	First Name	Date of Hire	Date of Birth	Dept.	Salary	Age
0	1	Abercrombie	Kim	24/06/1985	16/10/1962	D	91000	55
1	2	Ackerman	Pilar	26/04/1989	05/10/1950	E	31000	68
2	3	Ajenstat	François	01/02/1981	21/12/1964	C	48000	53
3	4	Akers	Kim	29/05/1979	08/04/1958	C	47000	60
4	5	Alberts	Amy E.	15/10/1989	22/04/1970	D	60000	48
5	6	Alderson	Gregory F. (Greg)	12/01/1992	28/05/1964	D	100000	54

The skiprows Parameter

- By adding the ***skiprows*** parameter, you can skip reading the specify number of row.
- Since we skipped the header row, the new data has lost its header and the index based on the file data.
- In some cases, it may be better to slice your data in a DataFrame rather than before loading the data.

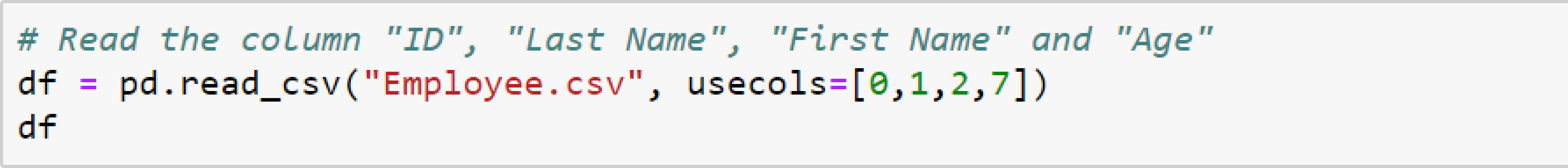


```
# Read the file and skip first 100 rows
df = pd.read_csv("Employee.csv", skiprows=100)
df
```

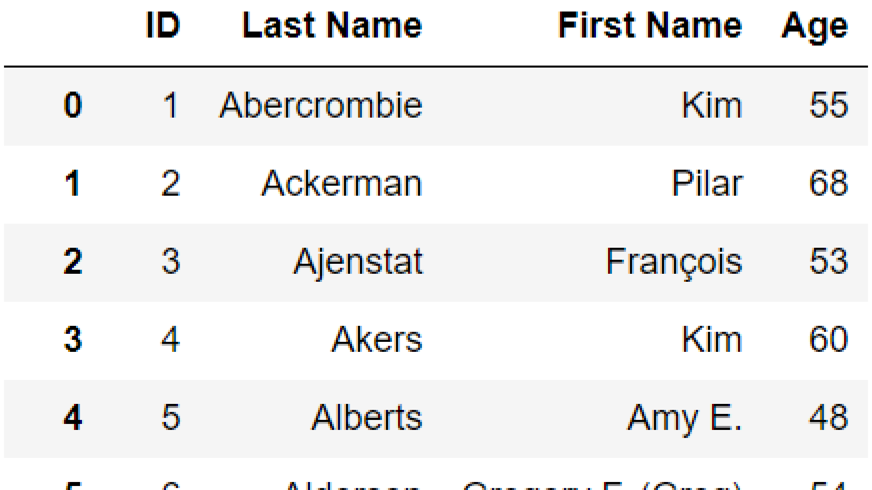
	100	Getzinger	Tom	10/10/1982	31/08/1975	A	80000	43
0	101	Giakoumakis	Leo	11/12/1996	09/01/1960	E	91000	58
1	102	Glimp	Diane R.	12/08/1998	10/11/1955	E	53000	62
2	103	Glynn	James R	25/06/1997	22/04/1970	C	61000	48

The usecols Parameter

- The ***usecols*** is a useful parameter that allows you to import only a subset of the data by column.
- It can be passed a zeroth index or a list of strings with the column names.



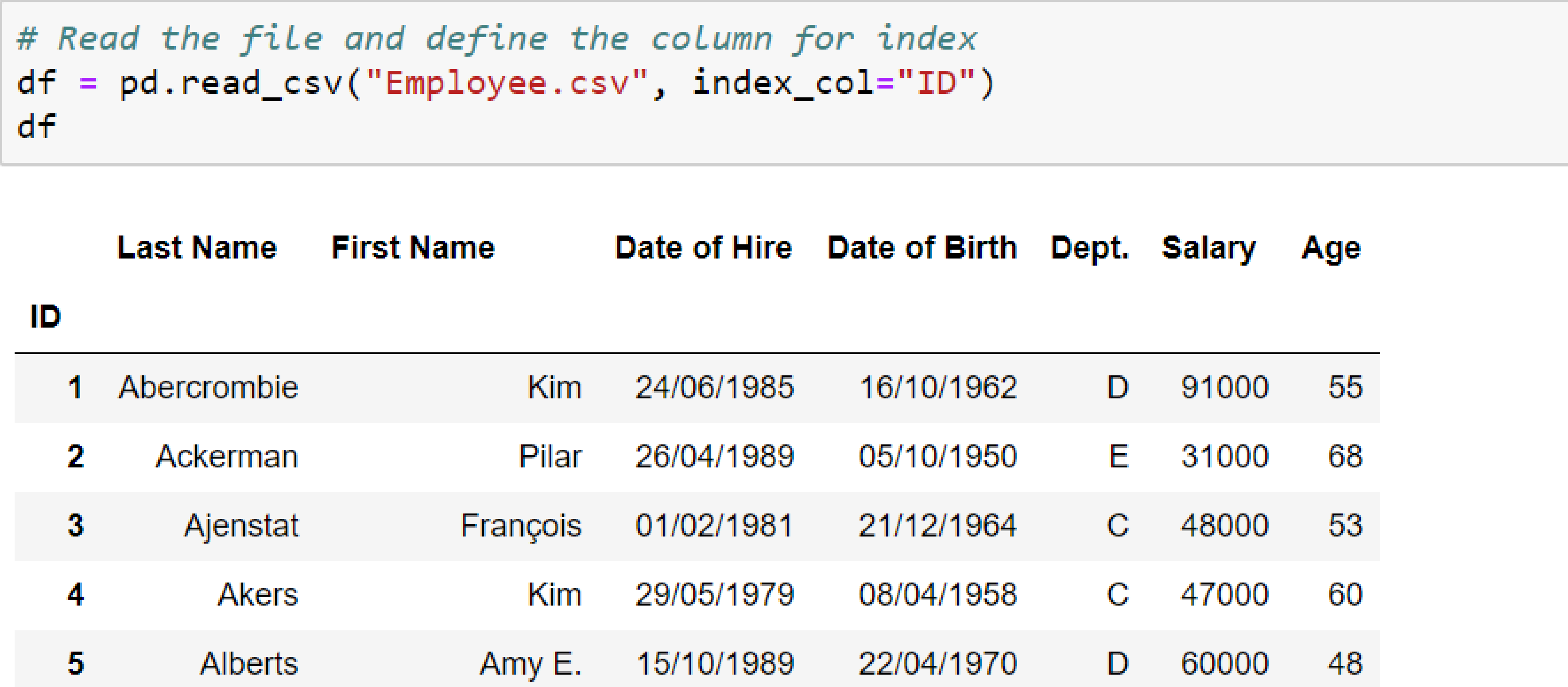
```
# Read the column "ID", "Last Name", "First Name" and "Age"
df = pd.read_csv("Employee.csv", usecols=[0,1,2,7])
df
```



	ID	Last Name	First Name	Age
0	1	Abercrombie	Kim	55
1	2	Ackerman	Pilar	68
2	3	Ajenstat	François	53
3	4	Akers	Kim	60
4	5	Alberts	Amy E.	48
5	6	Alderman	Gregory F. (Greg)	54

The index_col Paramter

- The standard behavior for ***read_csv()*** automatically create an incremental integer based index, the ***index_col*** parameter that can be used to indicate the column that holds the index.



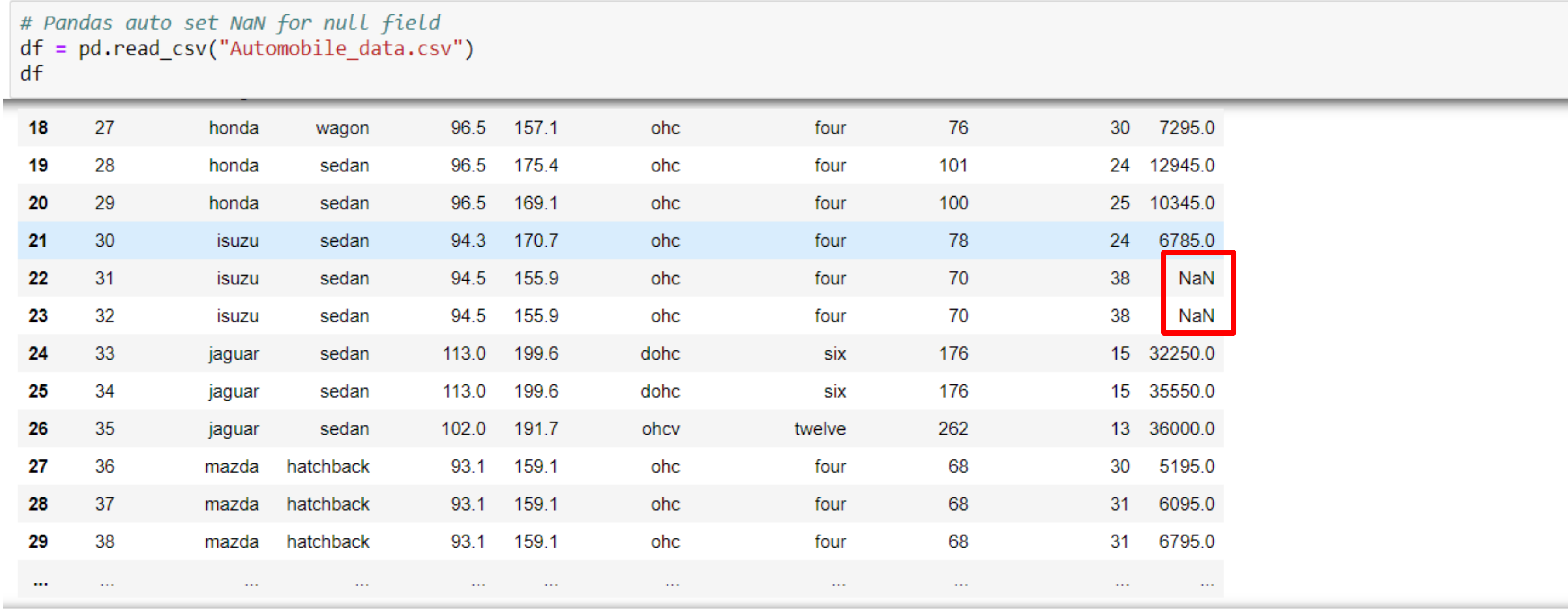
```
# Read the file and define the column for index
df = pd.read_csv("Employee.csv", index_col="ID")
df
```

	Last Name	First Name	Date of Hire	Date of Birth	Dept.	Salary	Age
ID							
1	Abercrombie	Kim	24/06/1985	16/10/1962	D	91000	55
2	Ackerman	Pilar	26/04/1989	05/10/1950	E	31000	68
3	Ajenstat	François	01/02/1981	21/12/1964	C	48000	53
4	Akers	Kim	29/05/1979	08/04/1958	C	47000	60
5	Alberts	Amy E.	15/10/1989	22/04/1970	D	60000	48

Handle Null Value

- The current handling of the na_values argument to read_csv is strangely different depending on what kind of value you pass to na_values.
- If you pass None, the default NA values are used.
- If you pass a dict mapping column names to values, then those values will be used for those columns, totally overriding the default NA values, while for columns not in the dict, the default values will be used.
- If you pass some other kind of iterable, it uses the union of the passed values and the default values as the NA values.

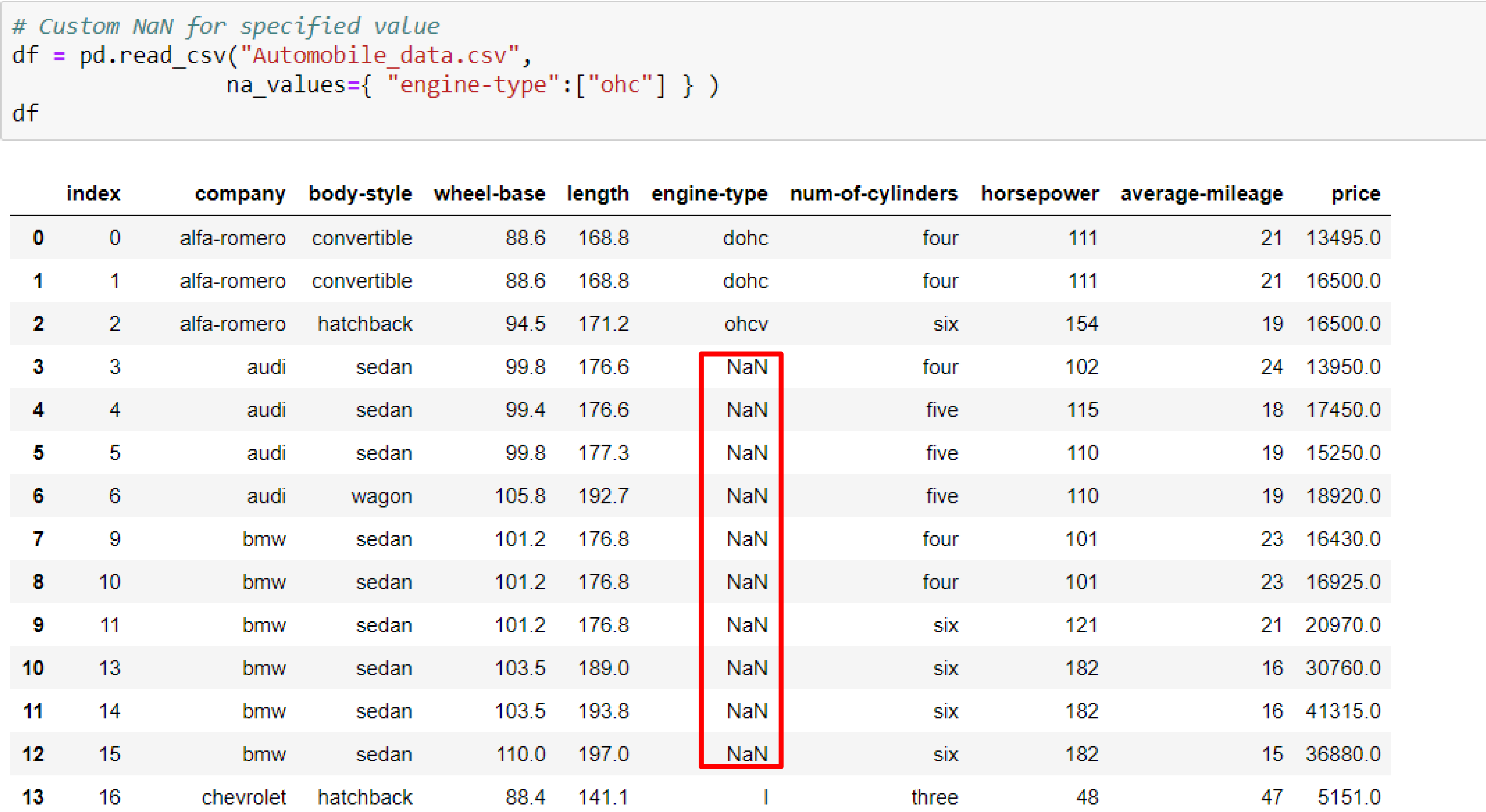
Example 1: Auto Detect Null Value



```
# Pandas auto set NaN for null field
df = pd.read_csv("Automobile_data.csv")
df
```

18	27	honda	wagon	96.5	157.1	ohc	four	76	30	7295.0
19	28	honda	sedan	96.5	175.4	ohc	four	101	24	12945.0
20	29	honda	sedan	96.5	169.1	ohc	four	100	25	10345.0
21	30	isuzu	sedan	94.3	170.7	ohc	four	78	24	6785.0
22	31	isuzu	sedan	94.5	155.9	ohc	four	70	38	NaN
23	32	isuzu	sedan	94.5	155.9	ohc	four	70	38	NaN
24	33	jaguar	sedan	113.0	199.6	dohc	six	176	15	32250.0
25	34	jaguar	sedan	113.0	199.6	dohc	six	176	15	35550.0
26	35	jaguar	sedan	102.0	191.7	ohcv	twelve	262	13	36000.0
27	36	mazda	hatchback	93.1	159.1	ohc	four	68	30	5195.0
28	37	mazda	hatchback	93.1	159.1	ohc	four	68	31	6095.0
29	38	mazda	hatchback	93.1	159.1	ohc	four	68	31	6795.0
...	...	...	...	...	...	...	...	...	...	...

Example 2: Custom NaN Value

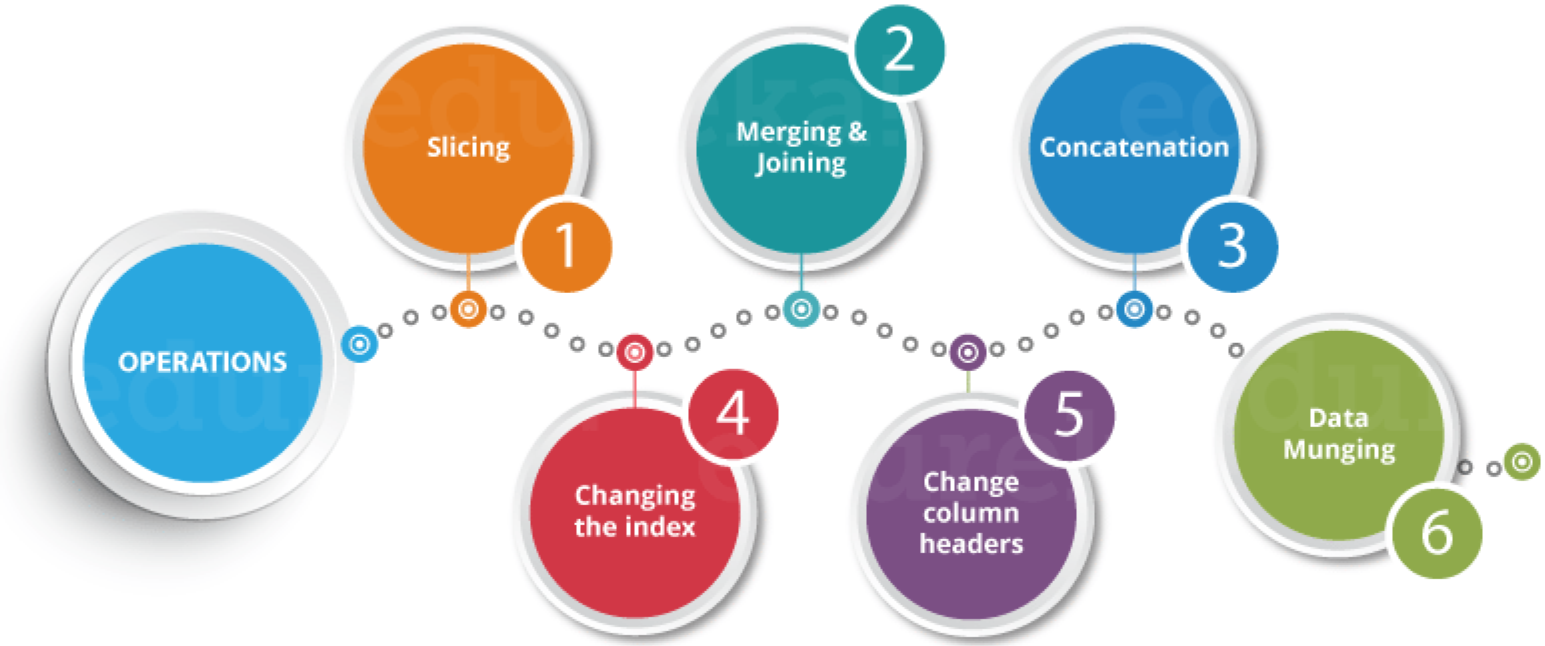


```
# Custom NaN for specified value
df = pd.read_csv("Automobile_data.csv",
                 na_values={ "engine-type":["ohc"] } )
df
```

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
0	0	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	13495.0
1	1	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	16500.0
2	2	alfa-romero	hatchback	94.5	171.2	ohcv	six	154	19	16500.0
3	3	audi	sedan	99.8	176.6	NaN	four	102	24	13950.0
4	4	audi	sedan	99.4	176.6	NaN	five	115	18	17450.0
5	5	audi	sedan	99.8	177.3	NaN	five	110	19	15250.0
6	6	audi	wagon	105.8	192.7	NaN	five	110	19	18920.0
7	9	bmw	sedan	101.2	176.8	NaN	four	101	23	16430.0
8	10	bmw	sedan	101.2	176.8	NaN	four	101	23	16925.0
9	11	bmw	sedan	101.2	176.8	NaN	six	121	21	20970.0
10	13	bmw	sedan	103.5	189.0	NaN	six	182	16	30760.0
11	14	bmw	sedan	103.5	193.8	NaN	six	182	16	41315.0
12	15	bmw	sedan	110.0	197.0	NaN	six	182	15	36880.0
13	16	chevrolet	hatchback	88.4	141.1	l	three	48	47	5151.0

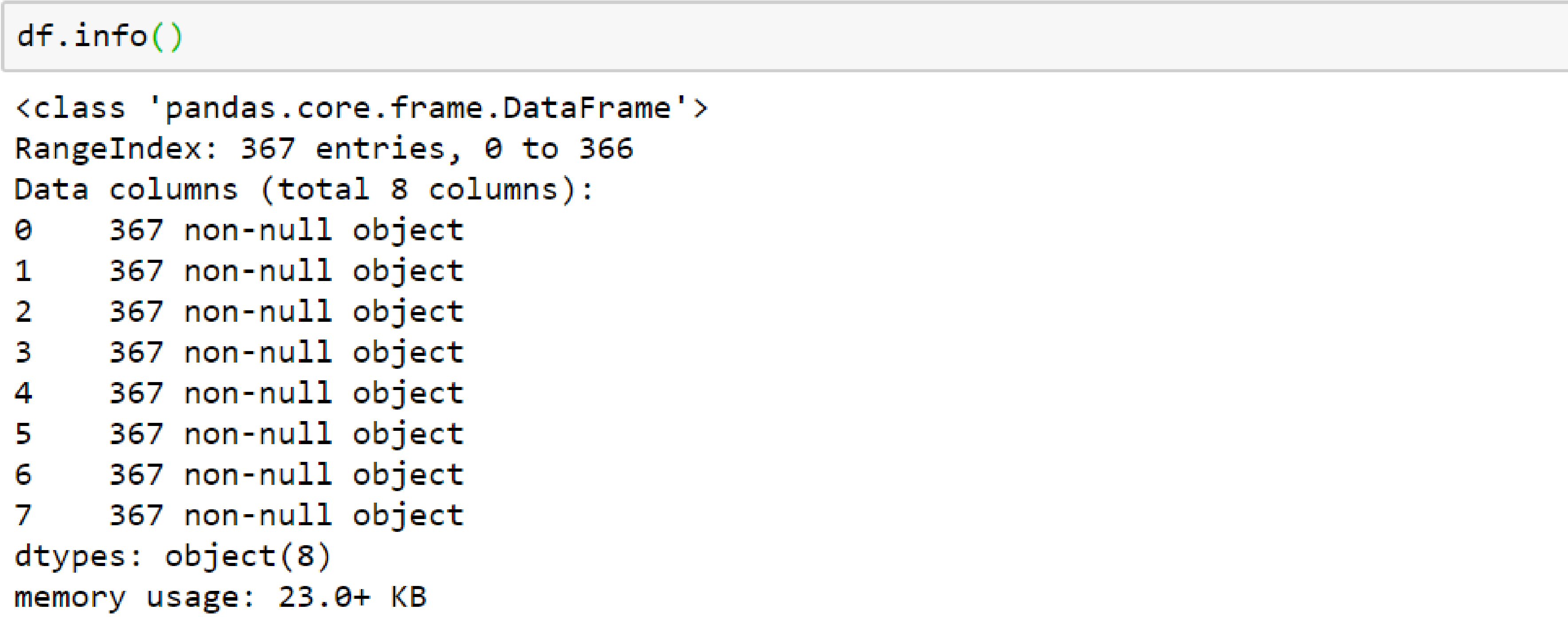
Python Pandas Operations

- Using Python pandas, you can perform a lot of operations with series, data frames, missing data, group by etc.



Obtain the Information of Data

□ **info()** provides the essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using.



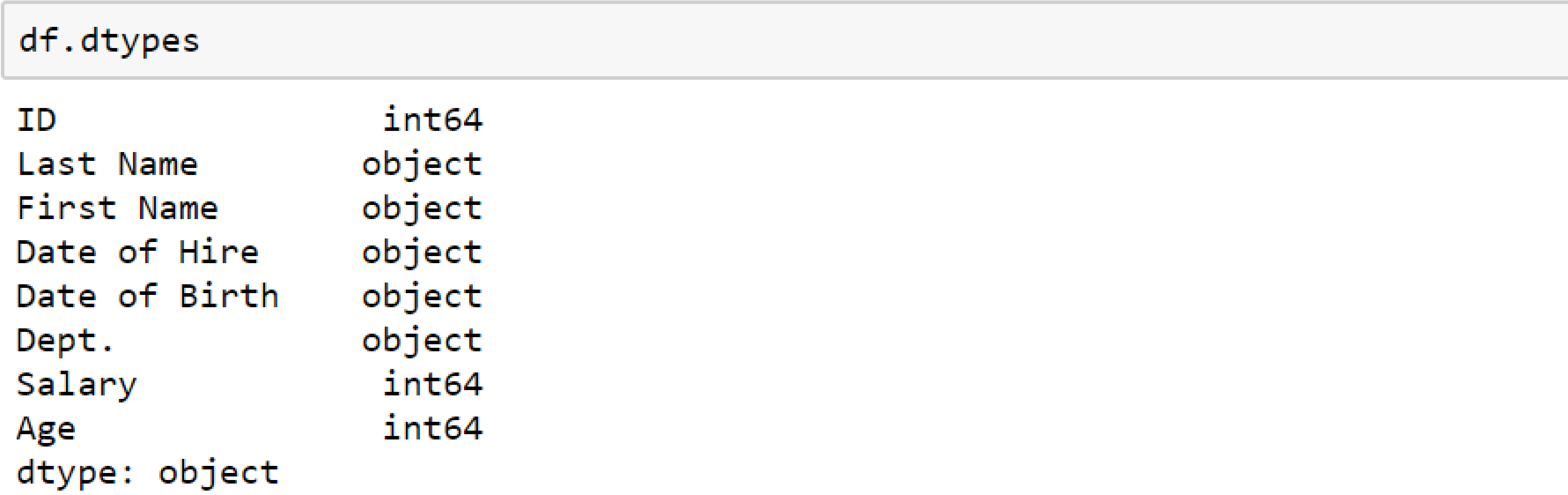
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 8 columns):
0    367 non-null object
1    367 non-null object
2    367 non-null object
3    367 non-null object
4    367 non-null object
5    367 non-null object
6    367 non-null object
7    367 non-null object
dtypes: object(8)
memory usage: 23.0+ KB
```

Data Types of Columns

- DataFrames always have mixed data types: some columns are numbers, some are strings, and some are dates etc.
- CSV files do not contain information on what data types are contained in each column; all of the data is just characters.
- Pandas infers the data types when loading the data, e.g. if a column contains only numbers, pandas will set that column’s data type to numeric: integer or float.
- You can check the types of each column in our example with the ***dtypes*** property of the dataframe.

Example

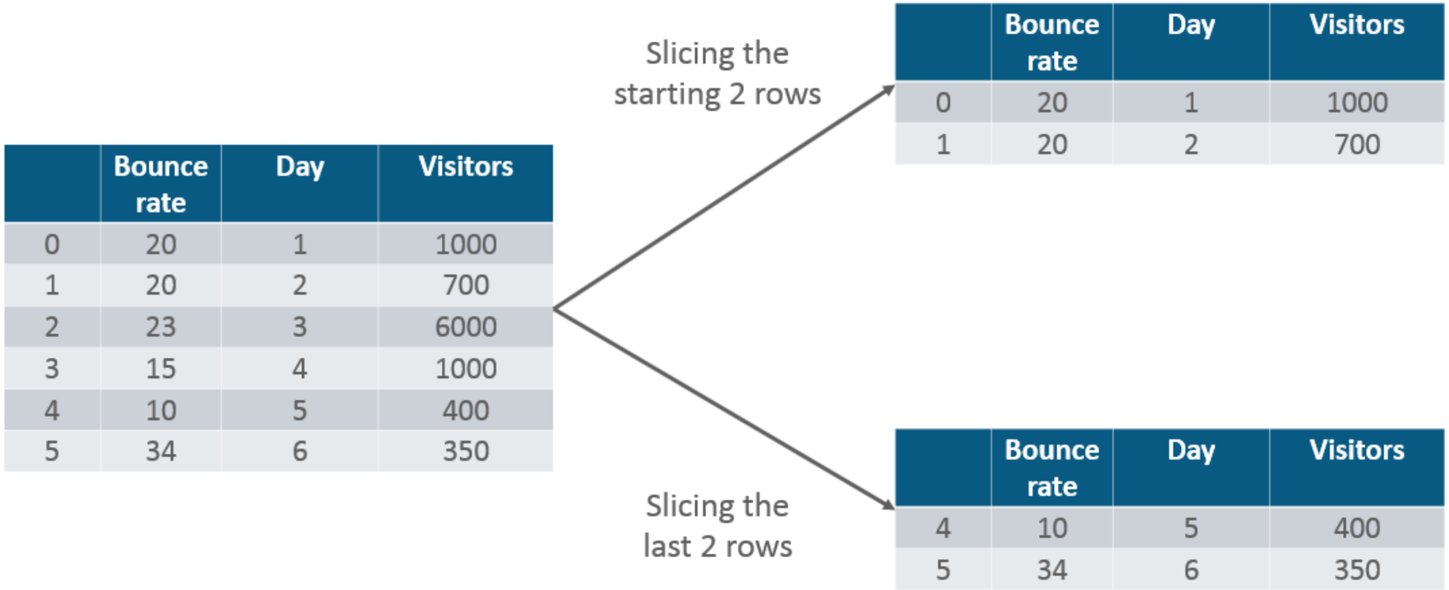


```
df.dtypes
```

```
ID                  int64
Last Name          object
First Name         object
Date of Hire       object
Date of Birth      object
Dept.              object
Salary              int64
Age                 int64
dtype: object
```

Slicing the Data Frame

- Once we read in a DataFrame, Pandas gives us two methods that make it fast to print out the data.
  - The ***head( )*** method prints the first N rows of a DataFrame. (Default 5).
  - The ***tail( )*** method prints the last N rows of a DataFrame. (Default 5).



	Bounce rate	Day	Visitors
0	20	1	1000
1	20	2	700
2	23	3	6000
3	15	4	1000
4	10	5	400
5	34	6	350

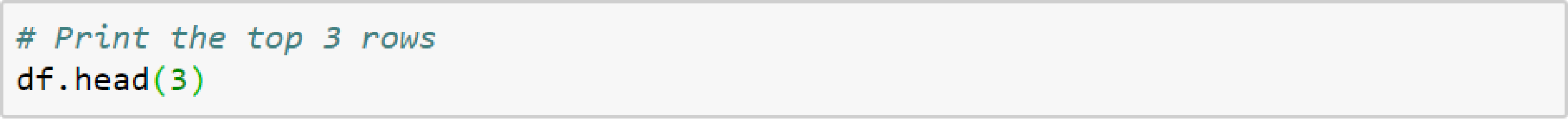
Slicing the starting 2 rows

	Bounce rate	Day	Visitors
0	20	1	1000
1	20	2	700

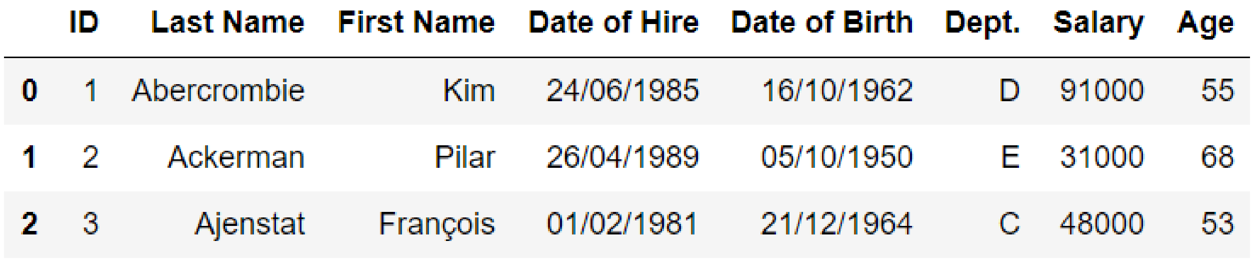
Slicing the last 2 rows

	Bounce rate	Day	Visitors
4	10	5	400
5	34	6	350

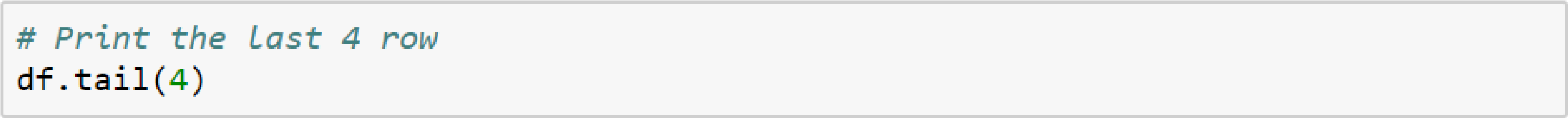
Example



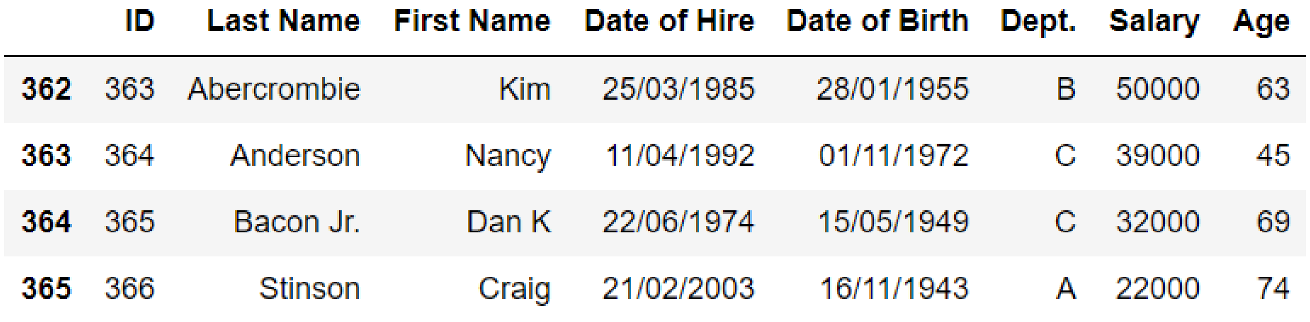
```
# Print the top 3 rows
df.head(3)
```



	ID	Last Name	First Name	Date of Hire	Date of Birth	Dept.	Salary	Age
0	1	Abercrombie	Kim	24/06/1985	16/10/1962	D	91000	55
1	2	Ackerman	Pilar	26/04/1989	05/10/1950	E	31000	68
2	3	Ajenstat	François	01/02/1981	21/12/1964	C	48000	53



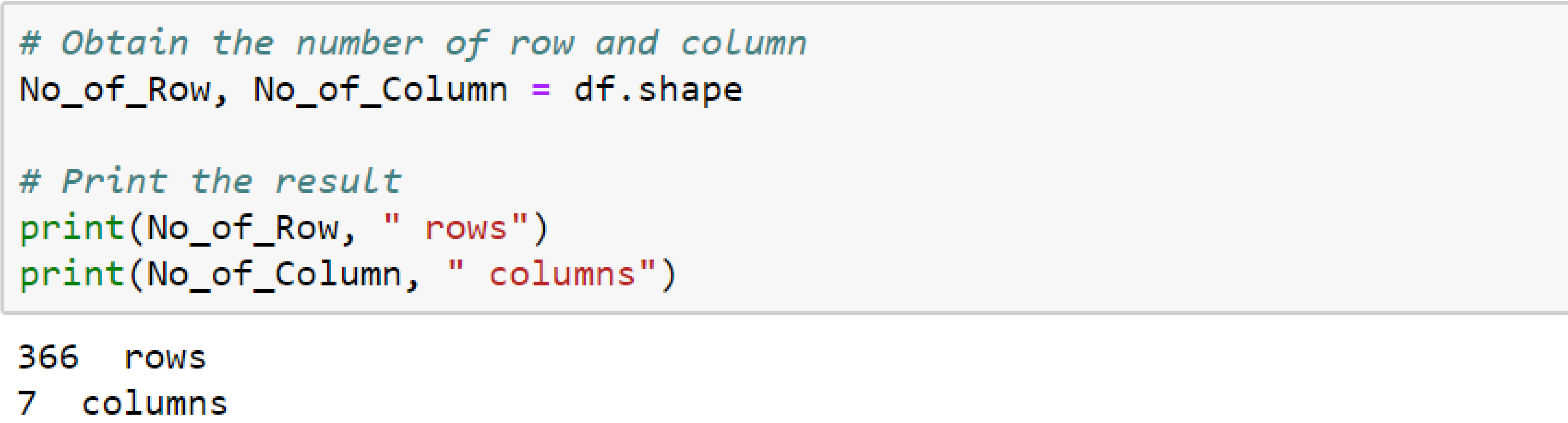
```
# Print the last 4 row
df.tail(4)
```



	ID	Last Name	First Name	Date of Hire	Date of Birth	Dept.	Salary	Age
362	363	Abercrombie	Kim	25/03/1985	28/01/1955	B	50000	63
363	364	Anderson	Nancy	11/04/1992	01/11/1972	C	39000	45
364	365	Bacon Jr.	Dan K	22/06/1974	15/05/1949	C	32000	69
365	366	Stinson	Craig	21/02/2003	16/11/1943	A	22000	74

Number of Row and Column

- The ***shape*** property show the number of rows and column in DataFrame.

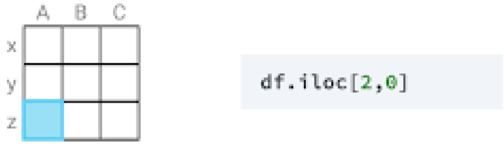


```
# Obtain the number of row and column
No_of_Row, No_of_Column = df.shape

# Print the result
print(No_of_Row, " rows")
print(No_of_Column, " columns")
```

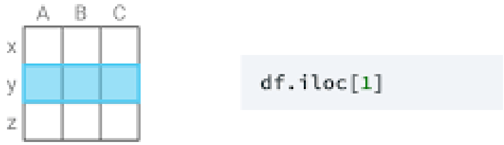
```
366  rows
7  columns
```

Indexing DataFrames

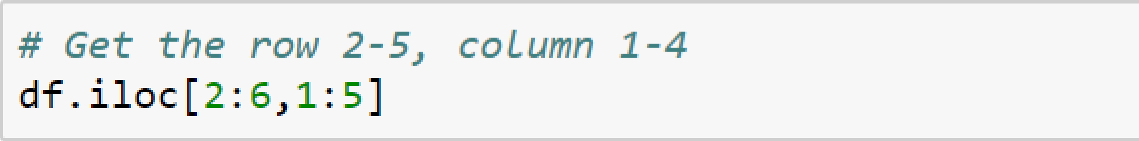
- The *iloc* method allows us to retrieve rows and columns by position.
- In order to do that, we'll need to specify the positions of the rows that we want, and the positions of the columns that we want as well.



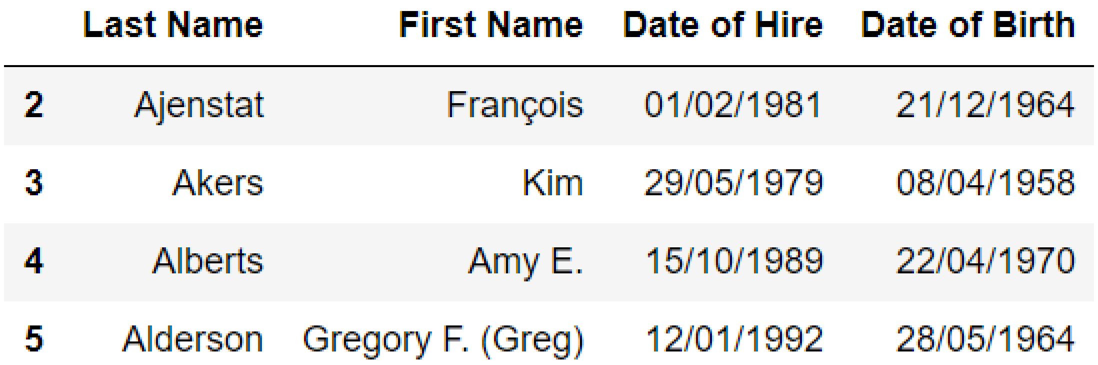
```
df.iloc[2,0]
```



```
df.iloc[1]
```



```
# Get the row 2-5, column 1-4
df.iloc[2:6,1:5]
```

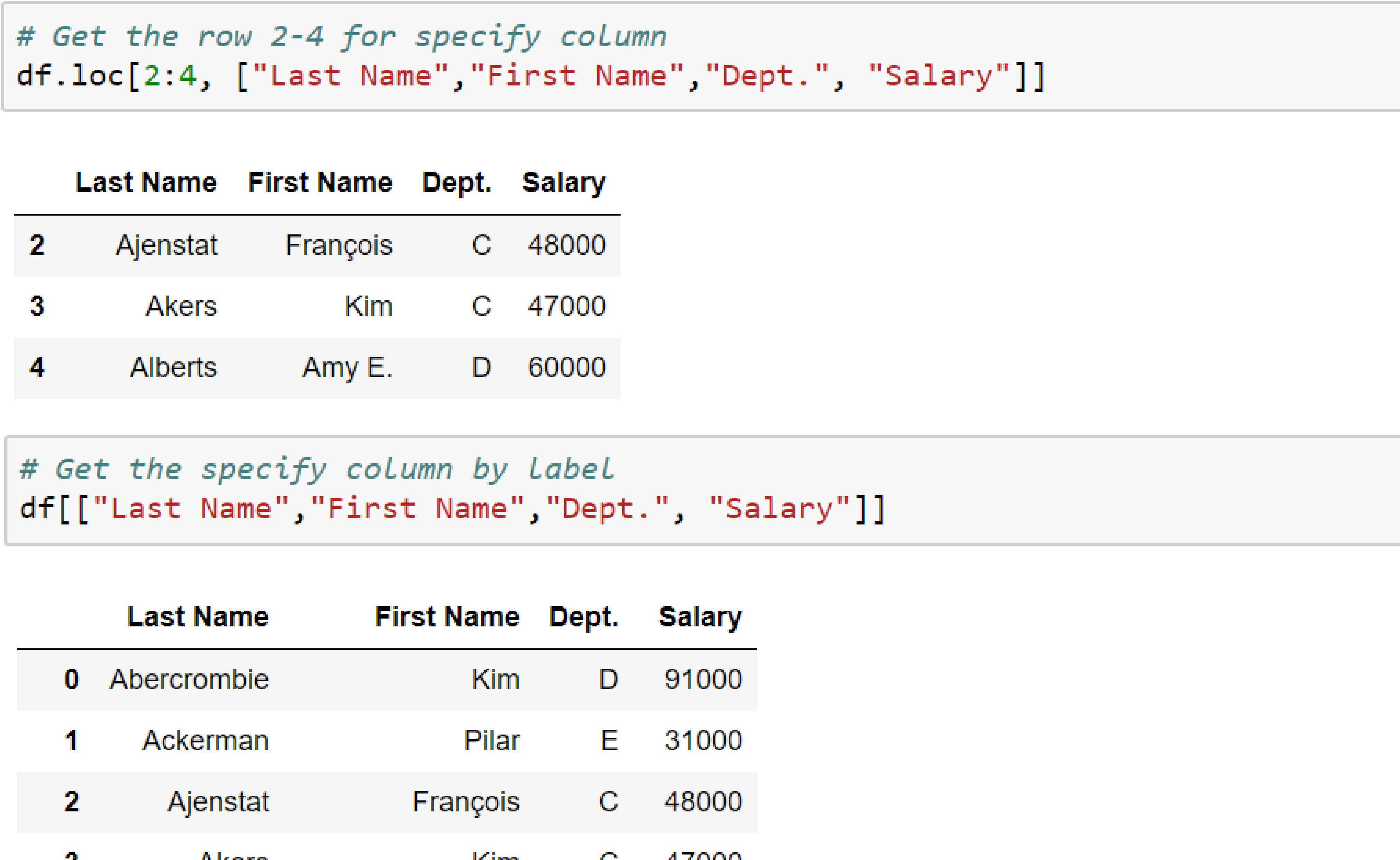


	Last Name	First Name	Date of Hire	Date of Birth
2	Ajenstat	François	01/02/1981	21/12/1964
3	Akers	Kim	29/05/1979	08/04/1958
4	Alberts	Amy E.	15/10/1989	22/04/1970
5	Alderson	Gregory F. (Greg)	12/01/1992	28/05/1964

Indexing using Labels

- □ A major advantage of Pandas over NumPy is that each of the columns and rows has a label.
- □ Working with column positions is possible, but it can be hard to keep track of which number corresponds to which column.
- □ We can work with labels using the ***loc()*** method, which allows us to index using labels instead of positions.

Example



```
# Get the row 2-4 for specify column
df.loc[2:4, ["Last Name","First Name","Dept.", "Salary"]]
```

	Last Name	First Name	Dept.	Salary
2	Ajenstat	François	C	48000
3	Akers	Kim	C	47000
4	Alberts	Amy E.	D	60000

```
# Get the specify column by label
df[["Last Name","First Name","Dept.", "Salary"]]
```

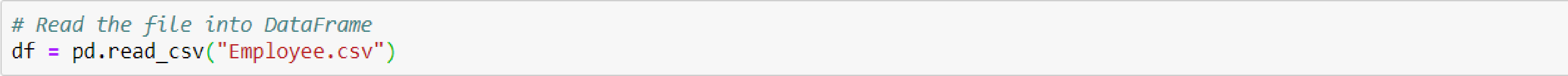
	Last Name	First Name	Dept.	Salary
0	Abercrombie	Kim	D	91000
1	Ackerman	Pilar	E	31000
2	Ajenstat	François	C	48000
3	Akers	Kim	C	47000

Sorting

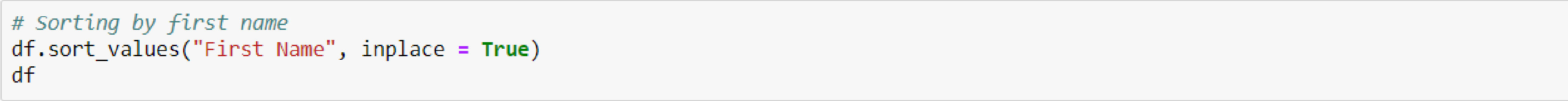
- Pandas **sort_values()** function sorts a data frame in Ascending or Descending order of passed Column.

Parameter	Description
by	Single/List of column names to sort Data Frame by.
axis	0 or 'index' for rows and 1 or 'columns' for Column.
ascending	Boolean value which sorts Data frame in ascending order if True.
inplace	Boolean value. Makes the changes in passed data frame itself if True.
kind	String which can have three inputs('quicksort', 'mergesort' or 'heapsort') of algorithm used to sort data frame.
na_position	Takes two string input 'last' or 'first' to set position of Null values. Default is 'last'.

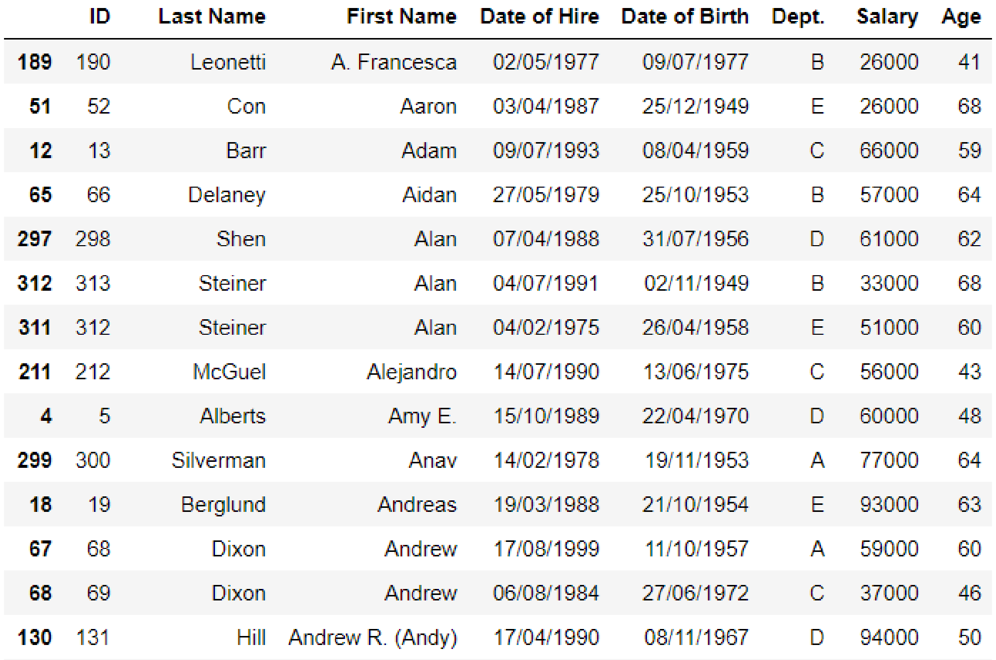
Example 1: Sort by Single Column



```
# Read the file into DataFrame
df = pd.read_csv("Employee.csv")
```

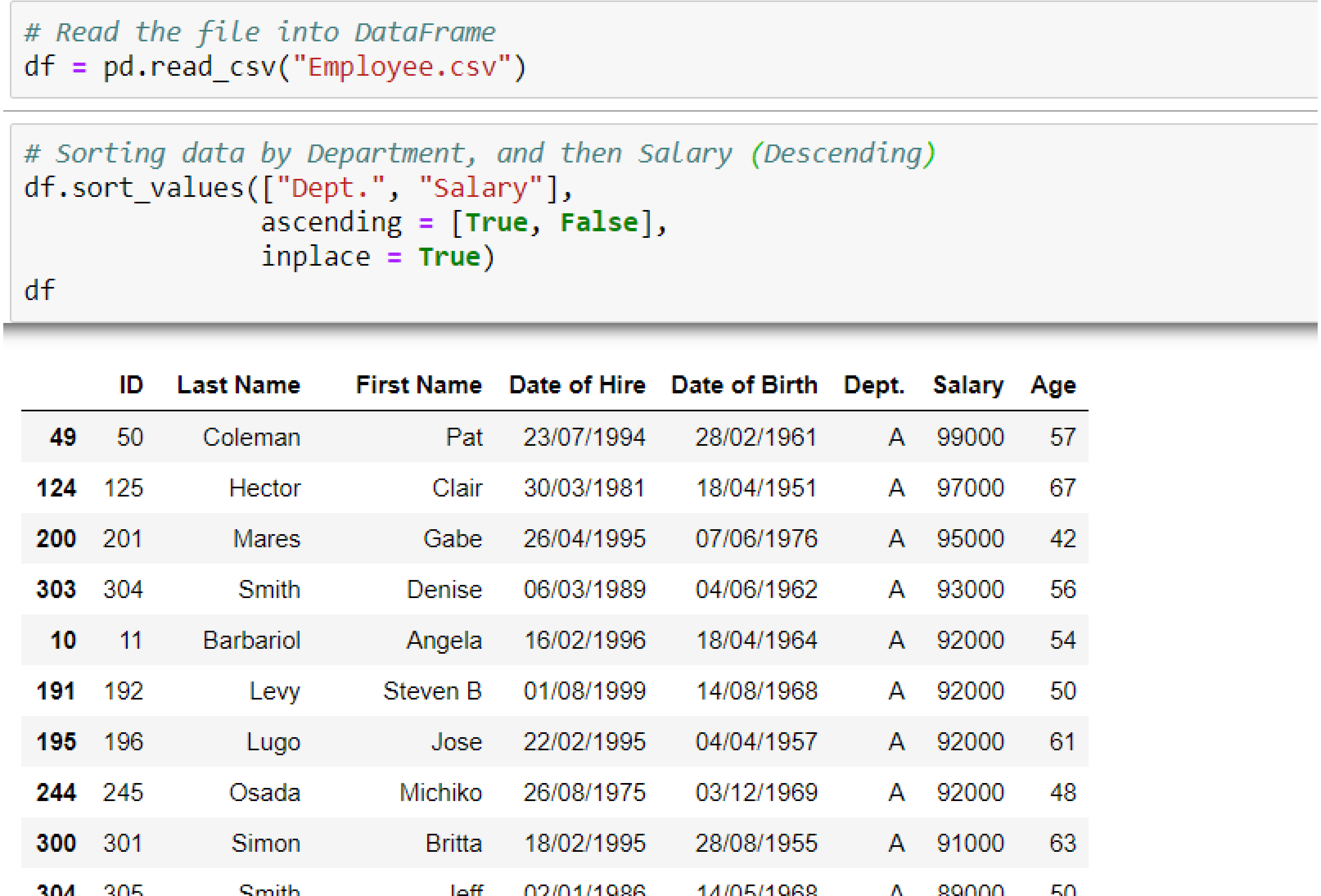


```
# Sorting by first name
df.sort_values("First Name", inplace = True)
df
```



	ID	Last Name	First Name	Date of Hire	Date of Birth	Dept.	Salary	Age
189	190	Leonetti	A. Francesca	02/05/1977	09/07/1977	B	26000	41
51	52	Con	Aaron	03/04/1987	25/12/1949	E	26000	68
12	13	Barr	Adam	09/07/1993	08/04/1959	C	66000	59
65	66	Delaney	Aidan	27/05/1979	25/10/1953	B	57000	64
297	298	Shen	Alan	07/04/1988	31/07/1956	D	61000	62
312	313	Steiner	Alan	04/07/1991	02/11/1949	B	33000	68
311	312	Steiner	Alan	04/02/1975	26/04/1958	E	51000	60
211	212	McGuel	Alejandro	14/07/1990	13/06/1975	C	56000	43
4	5	Alberts	Amy E.	15/10/1989	22/04/1970	D	60000	48
299	300	Silverman	Anav	14/02/1978	19/11/1953	A	77000	64
18	19	Berglund	Andreas	19/03/1988	21/10/1954	E	93000	63
67	68	Dixon	Andrew	17/08/1999	11/10/1957	A	59000	60
68	69	Dixon	Andrew	06/08/1984	27/06/1972	C	37000	46
130	131	Hill	Andrew R. (Andy)	17/04/1990	08/11/1967	D	94000	50

Example 2: Sort by Multiple Column



```
# Read the file into DataFrame
df = pd.read_csv("Employee.csv")
```

```
# Sorting data by Department, and then Salary (Descending)
df.sort_values(["Dept.", "Salary"],
               ascending = [True, False],
               inplace = True)
df
```

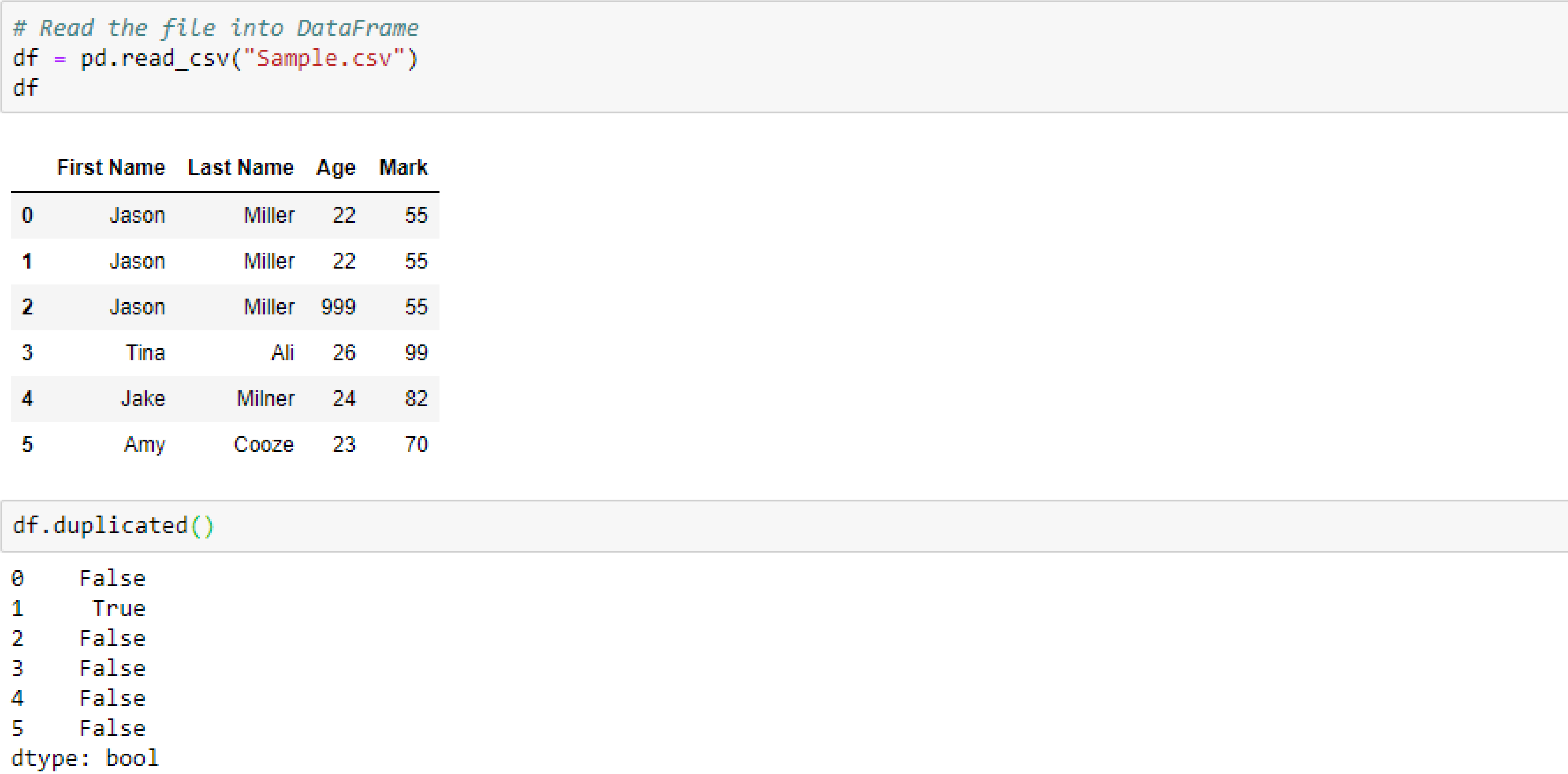
	ID	Last Name	First Name	Date of Hire	Date of Birth	Dept.	Salary	Age
49	50	Coleman	Pat	23/07/1994	28/02/1961	A	99000	57
124	125	Hector	Clair	30/03/1981	18/04/1951	A	97000	67
200	201	Mares	Gabe	26/04/1995	07/06/1976	A	95000	42
303	304	Smith	Denise	06/03/1989	04/06/1962	A	93000	56
10	11	Barbariol	Angela	16/02/1996	18/04/1964	A	92000	54
191	192	Levy	Steven B	01/08/1999	14/08/1968	A	92000	50
195	196	Lugo	Jose	22/02/1995	04/04/1957	A	92000	61
244	245	Osada	Michiko	26/08/1975	03/12/1969	A	92000	48
300	301	Simon	Britta	18/02/1995	28/08/1955	A	91000	63
304	305	Smith	Jeff	02/01/1986	14/05/1968	A	89000	50

Duplicate Handling

- The function **duplicated** is used to duplicate rows based on all columns or some specific columns in DataFrame.

Parameters	Description
subset	Single or multiple column labels which should used for duplication check. If not provides all columns will be checked for finding duplicate rows.
keep	Denotes the occurrence which should be marked as duplicate. It's value can be {'first', 'last', False}, default value is 'first'. • first : All duplicates except their first occurrence will be marked as True • last : All duplicates except their last occurrence will be marked as True • False : All duplicates except will be marked as True

Example



```
# Read the file into DataFrame
df = pd.read_csv("Sample.csv")
df
```

	First Name	Last Name	Age	Mark
0	Jason	Miller	22	55
1	Jason	Miller	22	55
2	Jason	Miller	999	55
3	Tina	Ali	26	99
4	Jake	Milner	24	82
5	Amy	Cooze	23	70

```
df.duplicated()
```

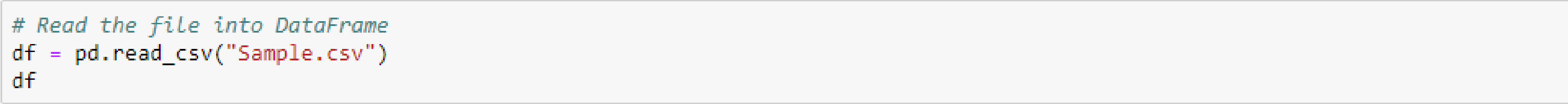
```
0    False
1     True
2    False
3    False
4    False
5    False
dtype: bool
```

Remove Duplicate Row

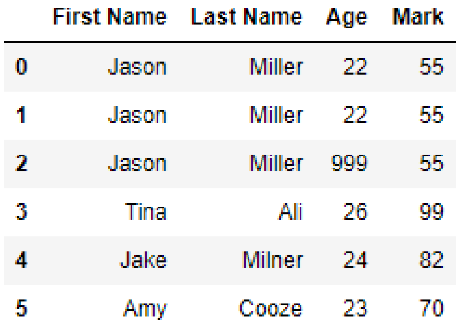
- The **drop_duplicates()** method return a copy of your DataFrame, but this time with duplicates removed.

Parameters	**Description**
subset	Subset takes a column or list of column label. It’s default value is none. After passing columns, it will consider them only for duplicates.
keep	keep is to control how to consider duplicate value. It has only three distinct value and default is ‘first’. • If ‘first’, it considers first value as unique and rest of the same values as duplicate. • If ‘last’, it considers last value as unique and rest of the same values as duplicate. • If False, it consider all of the same values as duplicates
inplace	Boolean values, removes rows with duplicates if True.
Return type	DataFrame with removed duplicate rows depending on Arguments passed.

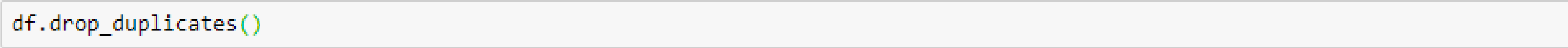
Example



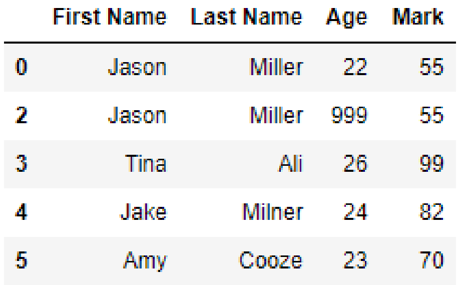
```python
# Read the file into DataFrame
df = pd.read_csv("Sample.csv")
df
```



	First Name	Last Name	Age	Mark
0	Jason	Miller	22	55
1	Jason	Miller	22	55
2	Jason	Miller	999	55
3	Tina	Ali	26	99
4	Jake	Milner	24	82
5	Amy	Cooze	23	70



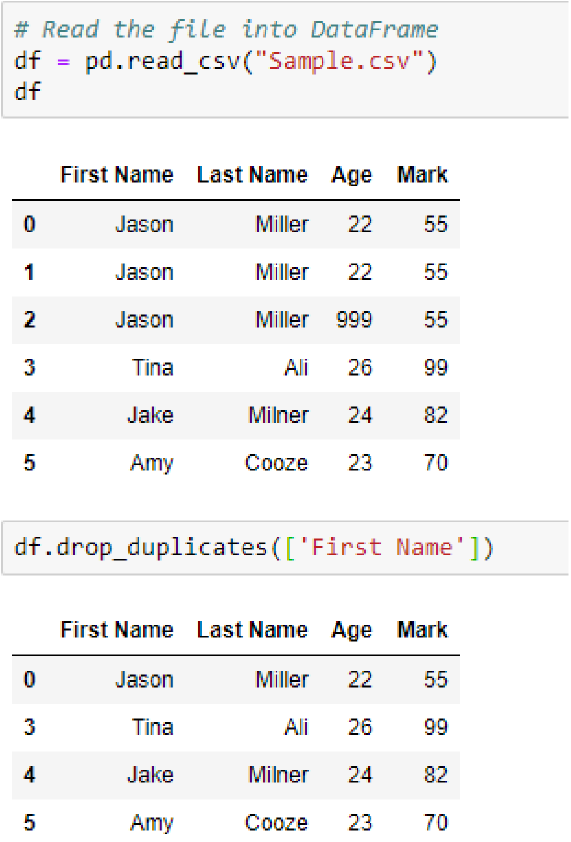
```python
df.drop_duplicates()
```



	First Name	Last Name	Age	Mark
0	Jason	Miller	22	55
2	Jason	Miller	999	55
3	Tina	Ali	26	99
4	Jake	Milner	24	82
5	Amy	Cooze	23	70

Keep First / Last Record

□ Keep First Record



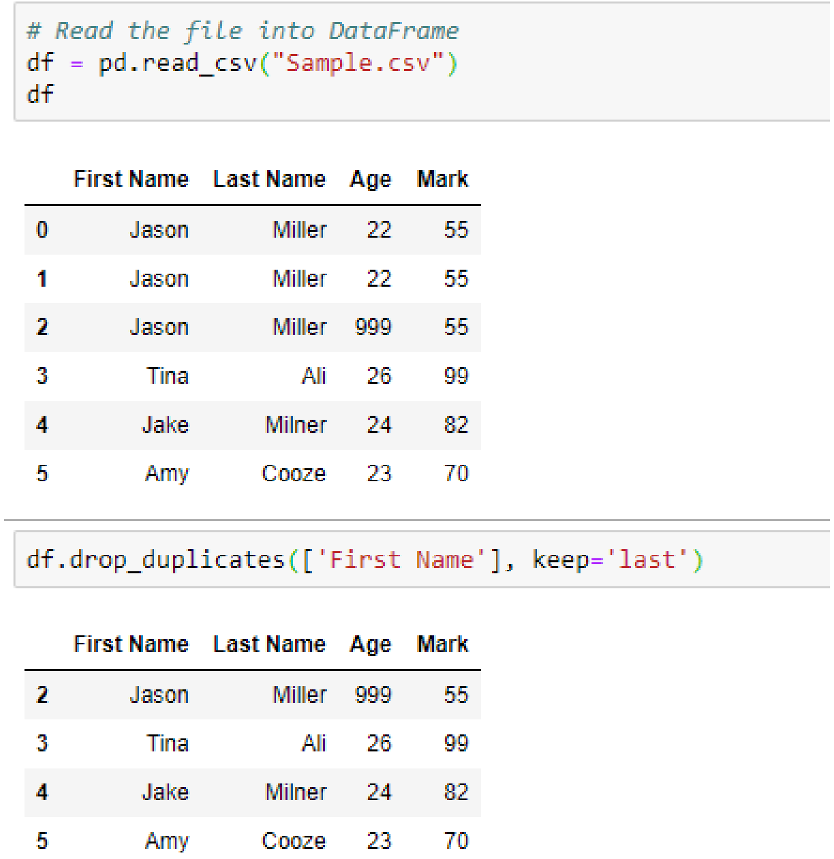
```
# Read the file into DataFrame
df = pd.read_csv("Sample.csv")
df
```

	First Name	Last Name	Age	Mark
0	Jason	Miller	22	55
1	Jason	Miller	22	55
2	Jason	Miller	999	55
3	Tina	Ali	26	99
4	Jake	Milner	24	82
5	Amy	Cooze	23	70

```
df.drop_duplicates(['First Name'])
```

	First Name	Last Name	Age	Mark
0	Jason	Miller	22	55
3	Tina	Ali	26	99
4	Jake	Milner	24	82
5	Amy	Cooze	23	70

□ Keep Last Record



```
# Read the file into DataFrame
df = pd.read_csv("Sample.csv")
df
```

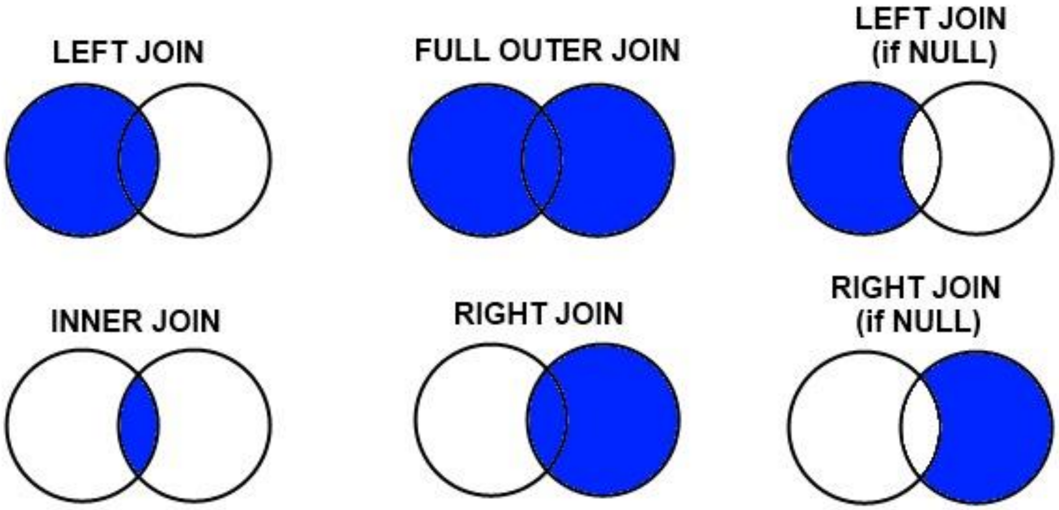
	First Name	Last Name	Age	Mark
0	Jason	Miller	22	55
1	Jason	Miller	22	55
2	Jason	Miller	999	55
3	Tina	Ali	26	99
4	Jake	Milner	24	82
5	Amy	Cooze	23	70

```
df.drop_duplicates(['First Name'], keep='last')
```

	First Name	Last Name	Age	Mark
2	Jason	Miller	999	55
3	Tina	Ali	26	99
4	Jake	Milner	24	82
5	Amy	Cooze	23	70

Merging DataFrames

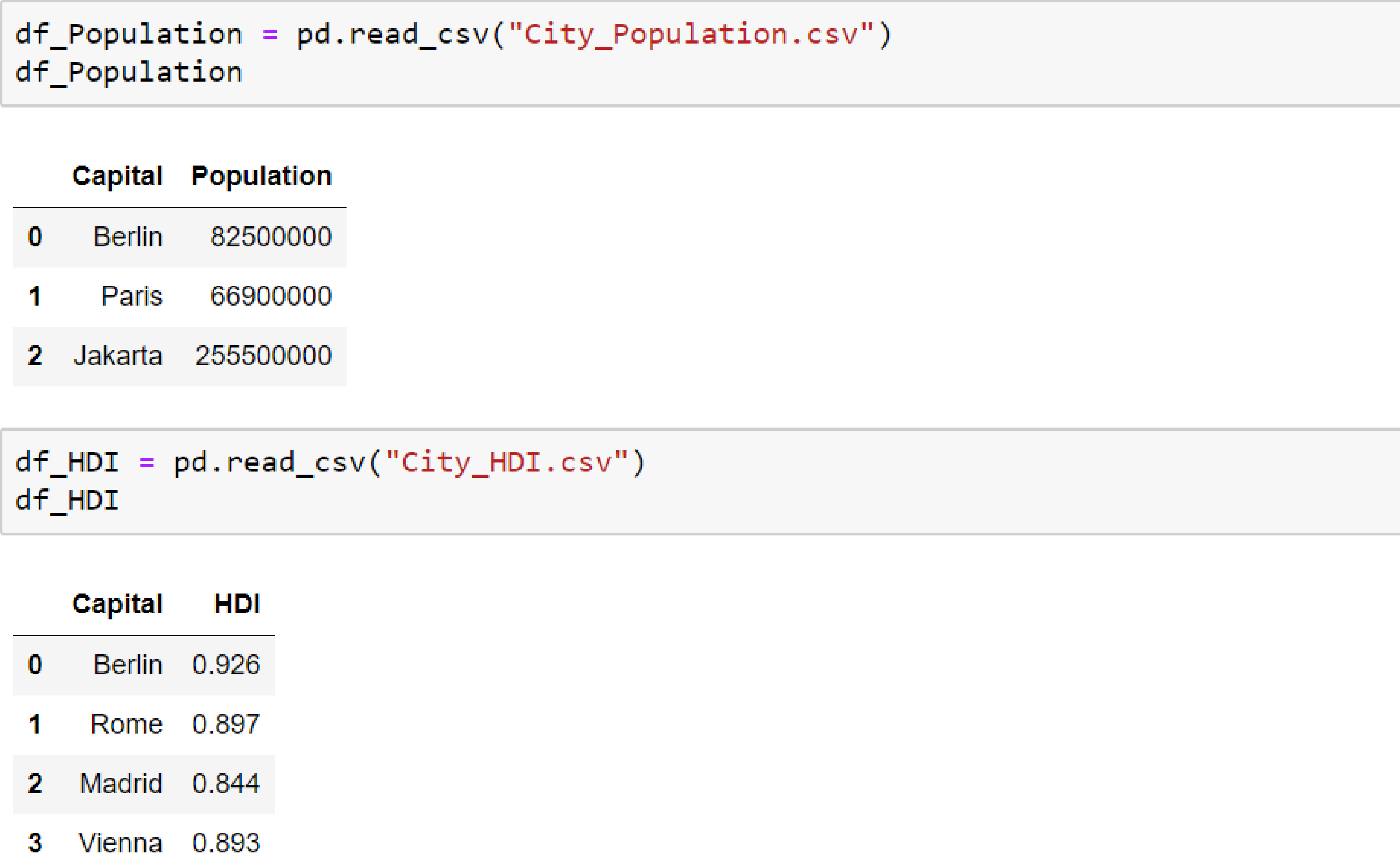
- Merging and joining DataFrames is a core process that any aspiring data analyst will need to master.
- The merge type to use is specified using the how parameter in the merge command, taking values “left”, “right”, “inner” (default), or “outer”.



Merge Type

Merge Type	Description
Inner Join	The default Pandas behavior, only keep rows where the merge “on” value exists in both the left and right DataFrames.
Left Outer Join	Keep every row in the left DataFrame. Where there are missing values of the “on” variable in the right DataFrame, add empty (NaN) values in the result.
Right Outer Join	Keep every row in the right DataFrame. Where there are missing values of the “on” variable in the left column, add empty (NaN) values in the result.
Full Outer Join	A full outer join returns all the rows from the left DataFrame, all the rows from the right DataFrame, and matches up rows where possible, with empty (NaN) elsewhere.

Example



```
df_Population = pd.read_csv("City_Population.csv")
df_Population
```

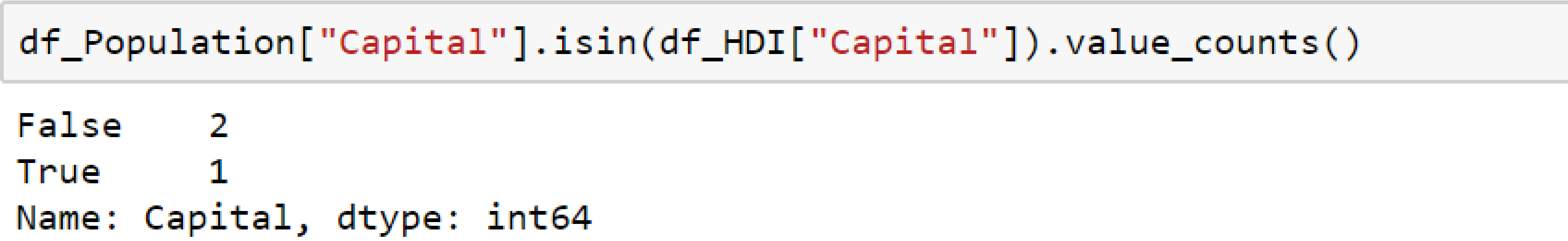
	Capital	Population
0	Berlin	82500000
1	Paris	66900000
2	Jakarta	255500000

```
df_HDI = pd.read_csv("City_HDI.csv")
df_HDI
```

	Capital	HDI
0	Berlin	0.926
1	Rome	0.897
2	Madrid	0.844
3	Vienna	0.893

Example

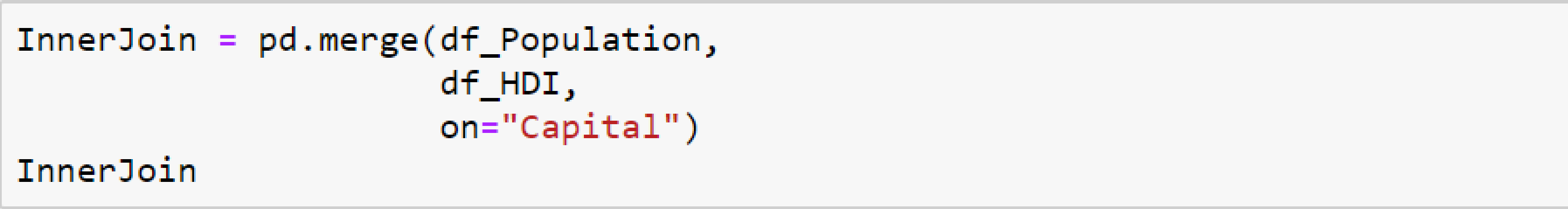
- By default, the Pandas merge operation acts with an “inner” join.
- An inner join keeps only the common values in both the left and right DataFrames for the result. We can validate this by looking at how many values are common:



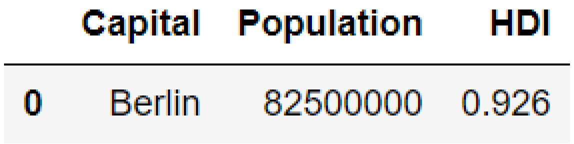
```
df_Population["Capital"].isin(df_HDI["Capital"]).value_counts()
```

```
False    2
True     1
Name: Capital, dtype: int64
```

Inner Join

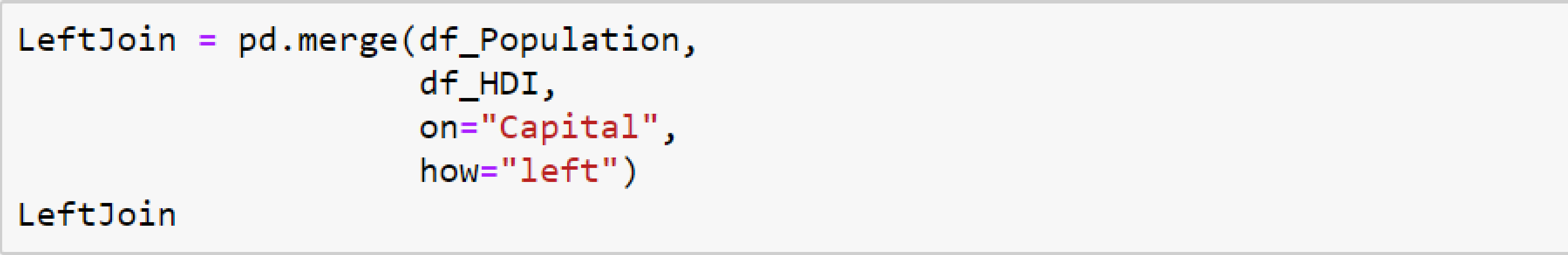


```
InnerJoin = pd.merge(df_Population,
                     df_HDI,
                     on="Capital")
InnerJoin
```

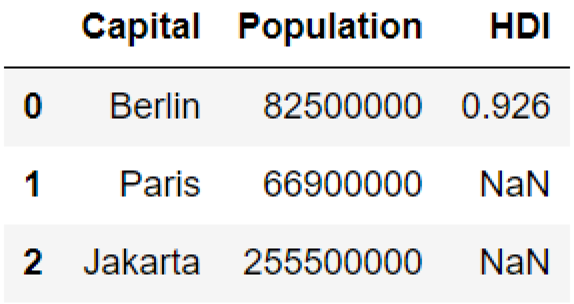


	Capital	Population	HDI
0	Berlin	82500000	0.926

Left Outer Join

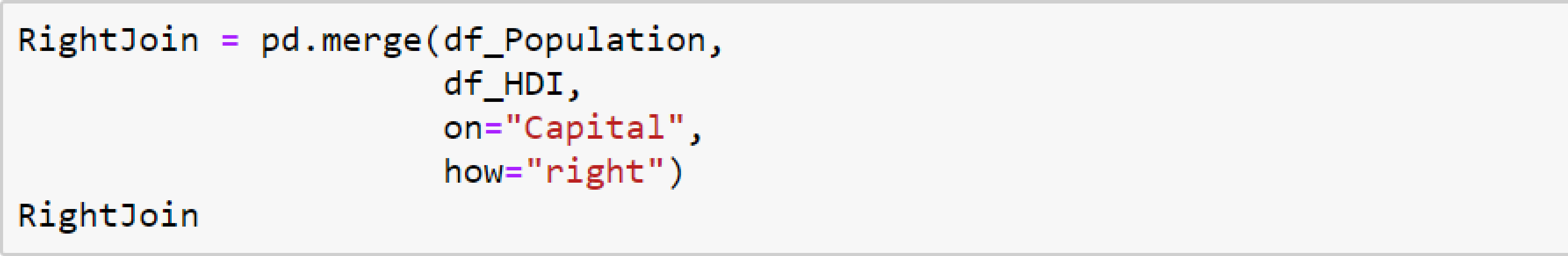


```
LeftJoin = pd.merge(df_Population,
                    df_HDI,
                    on="Capital",
                    how="left")
LeftJoin
```

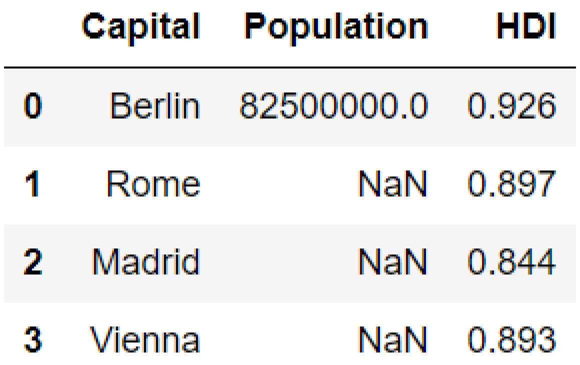


	Capital	Population	HDI
0	Berlin	82500000	0.926
1	Paris	66900000	NaN
2	Jakarta	255500000	NaN

Right Outer Join

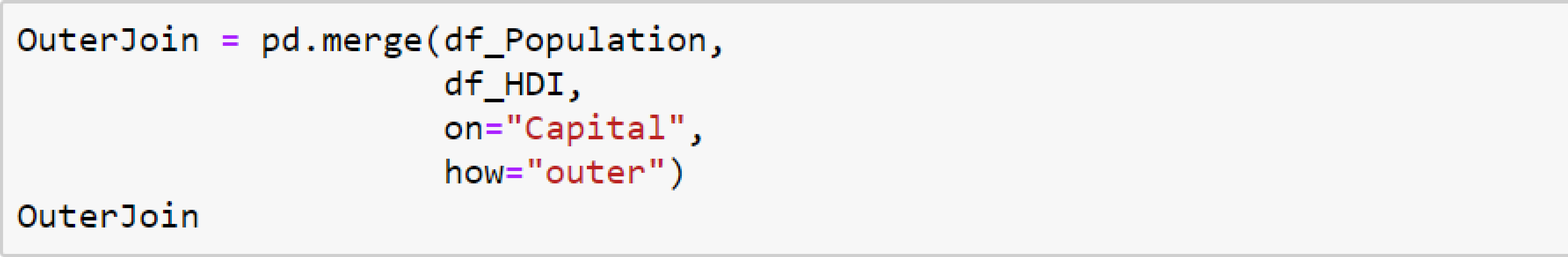


```
RightJoin = pd.merge(df_Population,
                     df_HDI,
                     on="Capital",
                     how="right")
RightJoin
```

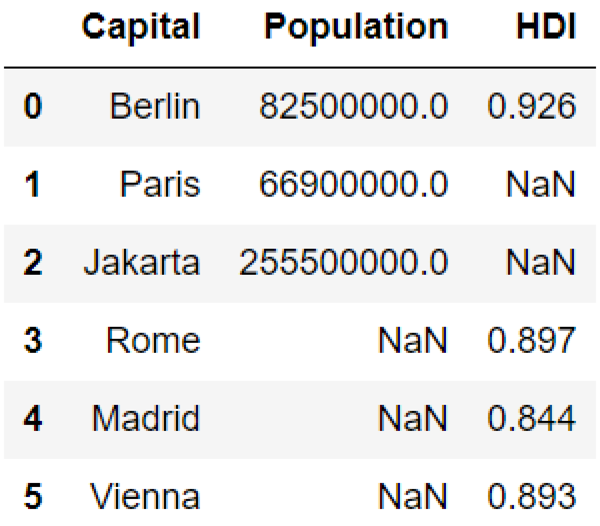


	Capital	Population	HDI
0	Berlin	82500000.0	0.926
1	Rome	NaN	0.897
2	Madrid	NaN	0.844
3	Vienna	NaN	0.893

Full Outer Join



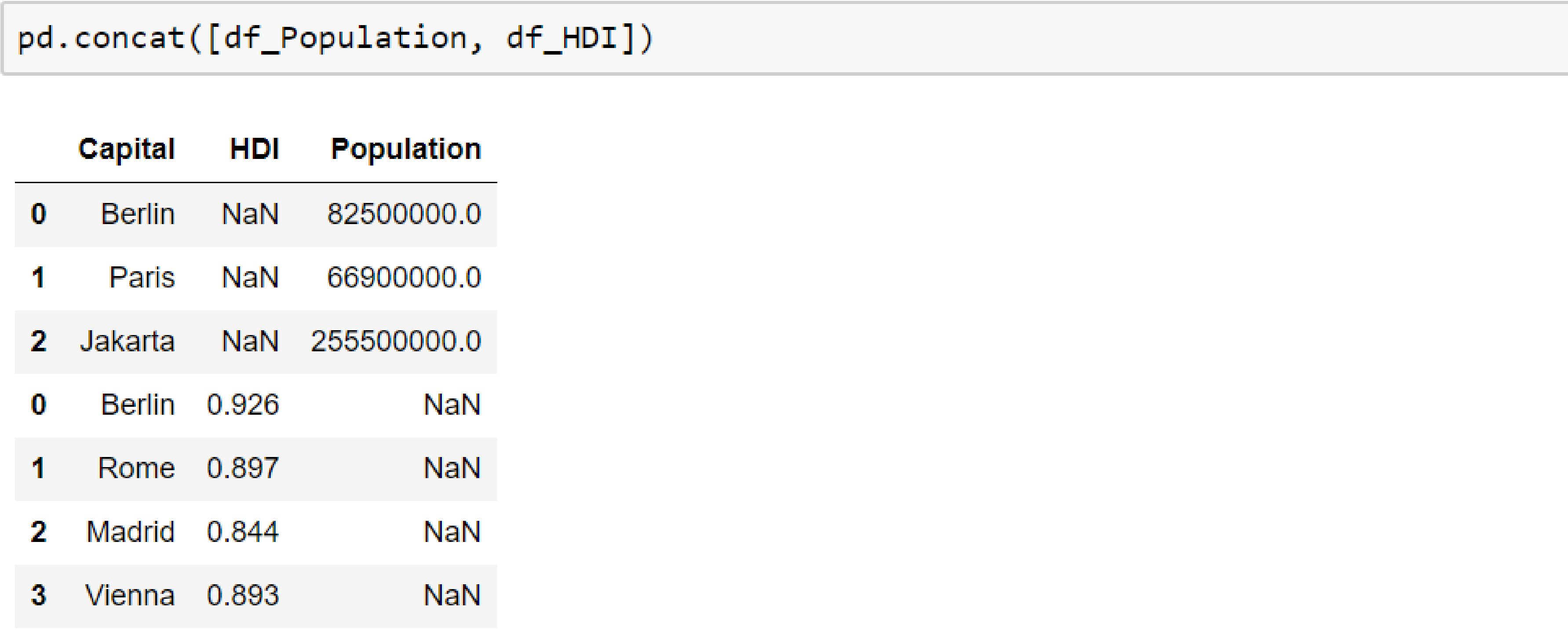
```
OuterJoin = pd.merge(df_Population,
                     df_HDI,
                     on="Capital",
                     how="outer")
OuterJoin
```



	Capital	Population	HDI
0	Berlin	82500000.0	0.926
1	Paris	66900000.0	NaN
2	Jakarta	255500000.0	NaN
3	Rome	NaN	0.897
4	Madrid	NaN	0.844
5	Vienna	NaN	0.893

Concatenate DataFrame

- The ***concat()*** function does all of the heavy lifting of performing concatenation operations along an axis while performing optional set logic of the indexes on the other axes.



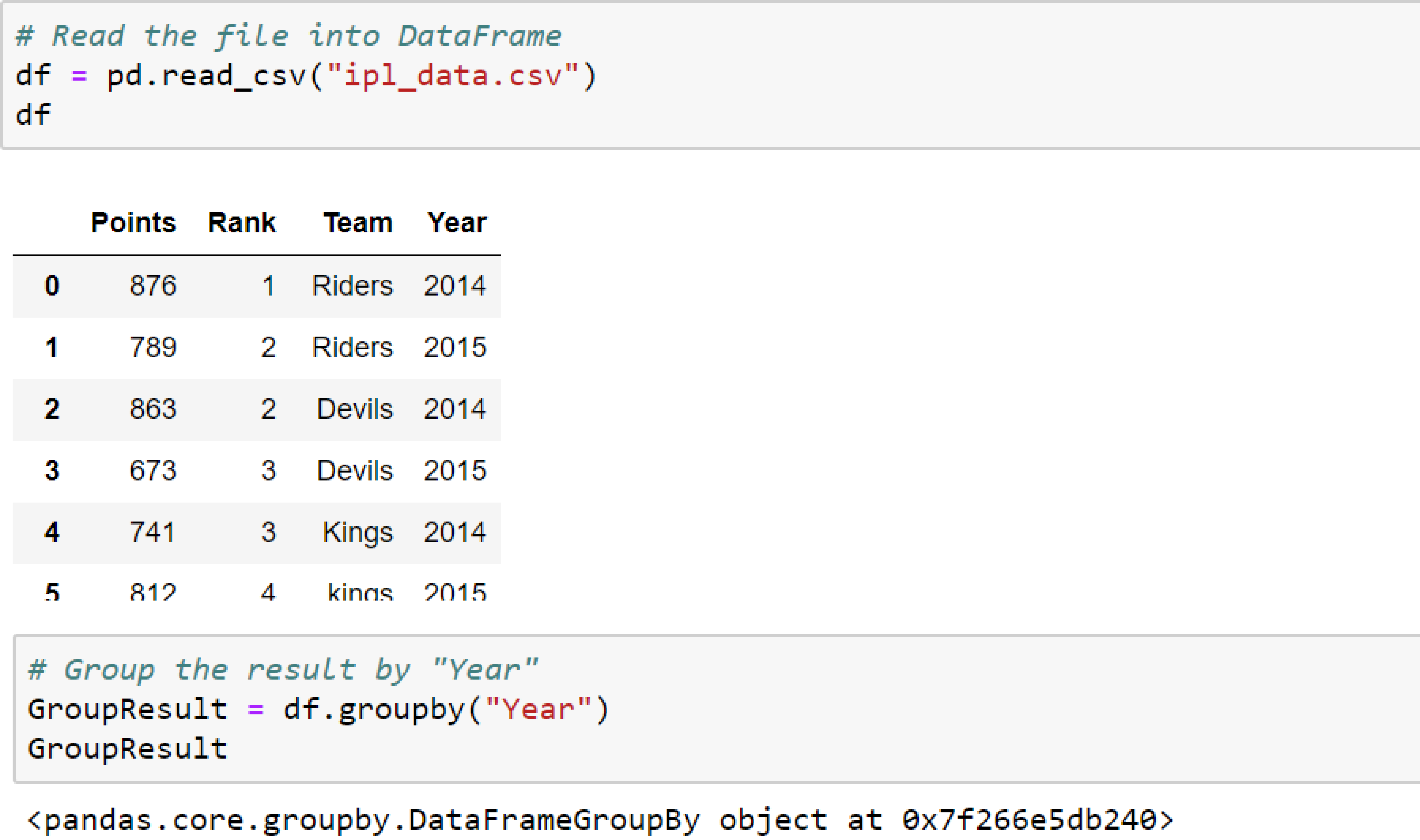
```
pd.concat([df_Population, df_HDI])
```

	Capital	HDI	Population
0	Berlin	NaN	82500000.0
1	Paris	NaN	66900000.0
2	Jakarta	NaN	255500000.0
0	Berlin	0.926	NaN
1	Rome	0.897	NaN
2	Madrid	0.844	NaN
3	Vienna	0.893	NaN

Group By

- Any groupby operation involves one of the following operations on the original object.
  - Splitting the Object
  - Applying a function
  - Combining the results
- In many situations, we split the data into sets and we apply some functionality on each subset. In the apply functionality, we can perform the following operations:
  - Aggregation − computing a summary statistic
  - Transformation − perform some group-specific operation
  - Filtration − discarding the data with some condition

Example



```
# Read the file into DataFrame
df = pd.read_csv("ipl_data.csv")
df
```

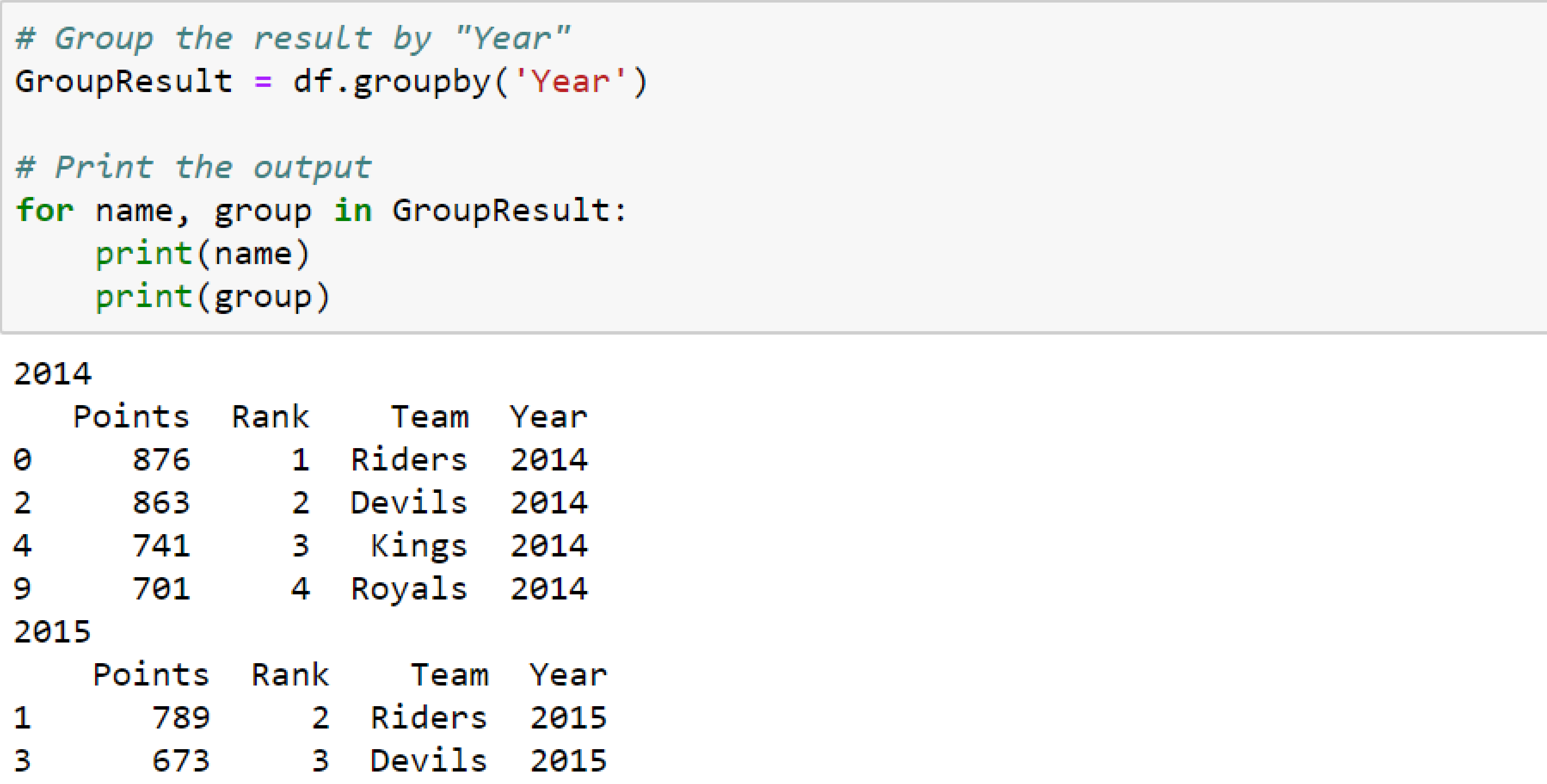
	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
2	863	2	Devils	2014
3	673	3	Devils	2015
4	741	3	Kings	2014
5	812	4	kings	2015

```
# Group the result by "Year"
GroupResult = df.groupby("Year")
GroupResult
```

```
<pandas.core.groupby.DataFrameGroupBy object at 0x7f266e5db240>
```

Iterating through Groups

□ With the ***groupby( )*** object in hand, we can iterate through the object similar to itertools.obj.

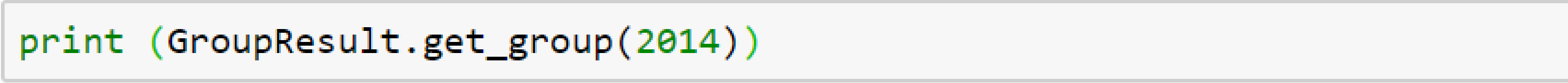


```
# Group the result by "Year"
GroupResult = df.groupby('Year')

# Print the output
for name, group in GroupResult:
    print(name)
    print(group)
```

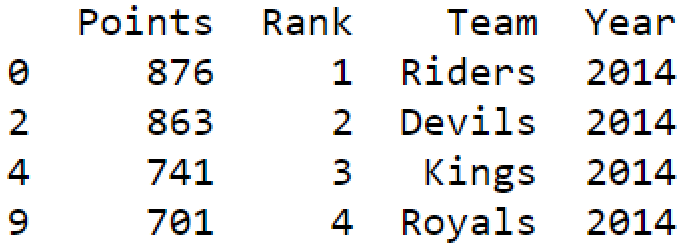
```
2014
   Points  Rank    Team  Year
0     876     1  Riders  2014
2     863     2  Devils  2014
4     741     3   Kings  2014
9     701     4  Royals  2014
2015
    Points  Rank    Team  Year
1      789     2  Riders  2015
3      673     3  Devils  2015
```

Select a Group

- Using the ***get_group( )*** method, we can select a single group.



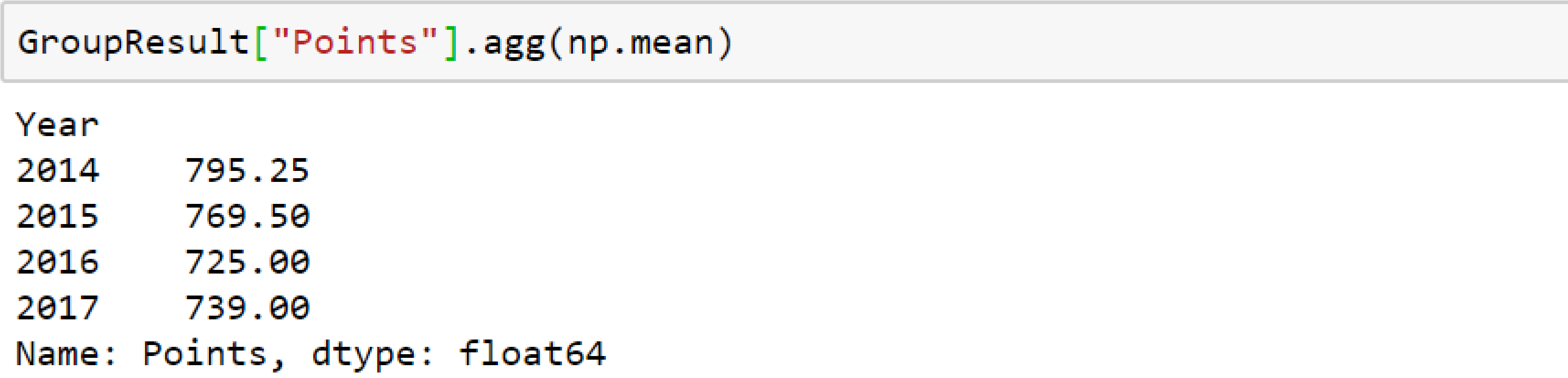
```
print (GroupResult.get_group(2014))
```



```
   Points  Rank    Team  Year
0     876     1  Riders  2014
2     863     2  Devils  2014
4     741     3   Kings  2014
9     701     4  Royals  2014
```

Aggregations

- An aggregated function returns a single aggregated value for each group. Once the group by object is created, several aggregation operations can be performed on the grouped data.
- An obvious one is aggregation via the aggregate or equivalent ***agg*** method

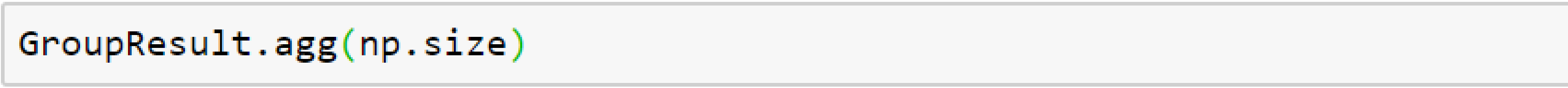


```
GroupResult["Points"].agg(np.mean)
```

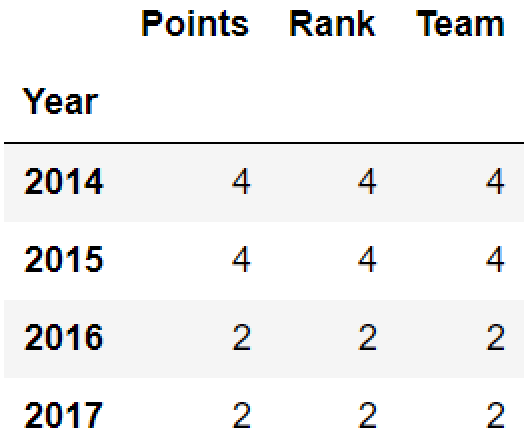
```
Year
2014    795.25
2015    769.50
2016    725.00
2017    739.00
Name: Points, dtype: float64
```

Size of Group

- To see the size of each group is by applying the ***size( )*** function



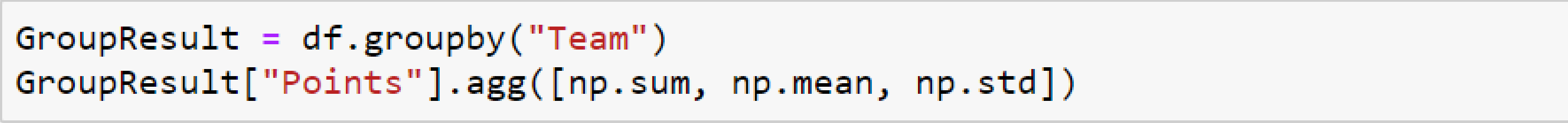
```
GroupResult.agg(np.size)
```



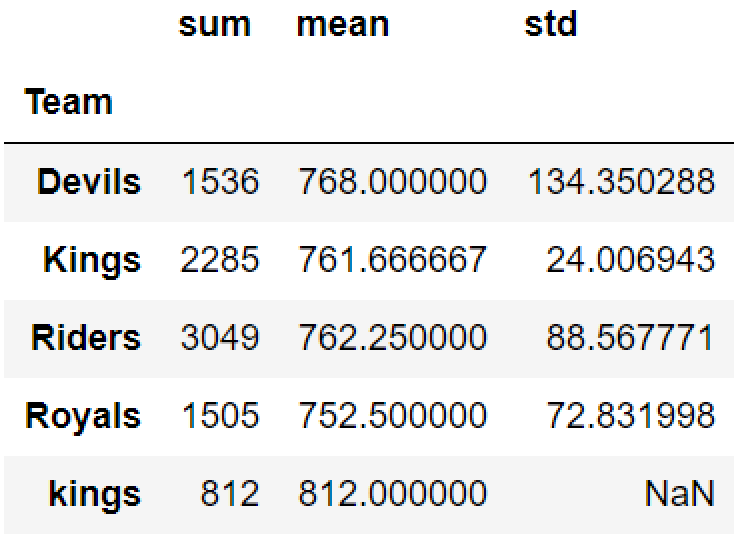
	Points	Rank	Team
Year			
2014	4	4	4
2015	4	4	4
2016	2	2	2
2017	2	2	2

Applying Multiple Aggregation Functions

- With grouped Series, you can also pass a list or dict of functions to do aggregation with, and generate DataFrame as output



```
GroupResult = df.groupby("Team")
GroupResult["Points"].agg([np.sum, np.mean, np.std])
```



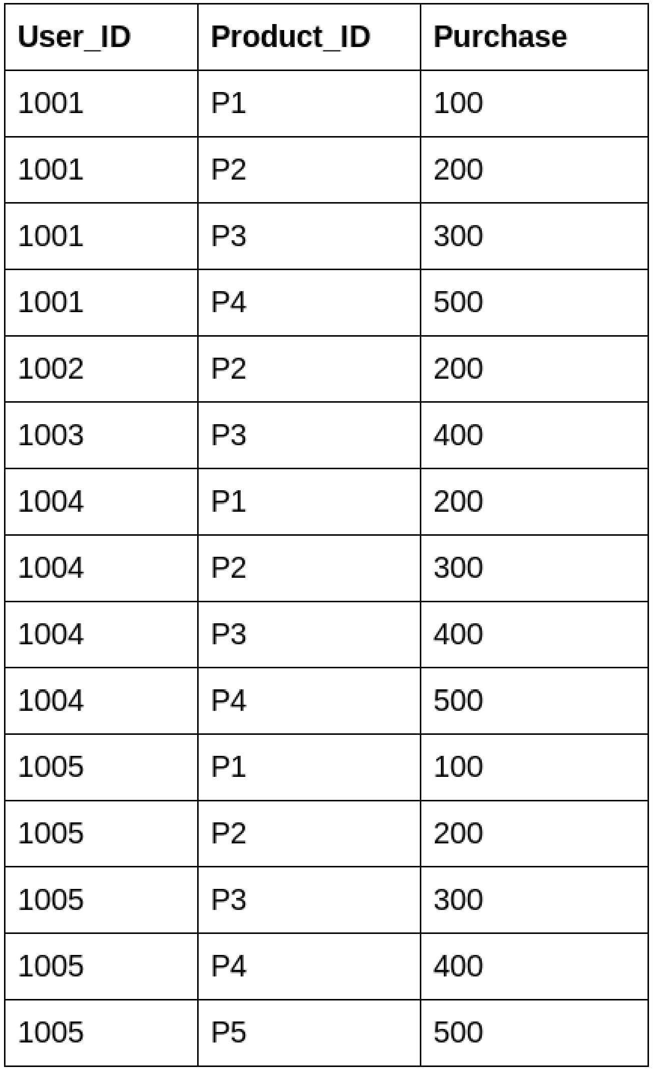
	sum	mean	std
Team			
Devils	1536	768.000000	134.350288
Kings	2285	761.666667	24.006943
Riders	3049	762.250000	88.567771
Royals	1505	752.500000	72.831998
kings	812	812.000000	NaN

Transformations

- Transformation on a group or a column returns an object that is indexed the same size of that is being grouped.
- The ***transform( )*** function is used to call function on self producing a Series with transformed values and that has the same axis length as self.
- Thus, the transform should return a result that is the same size as that of a group chunk.

Example

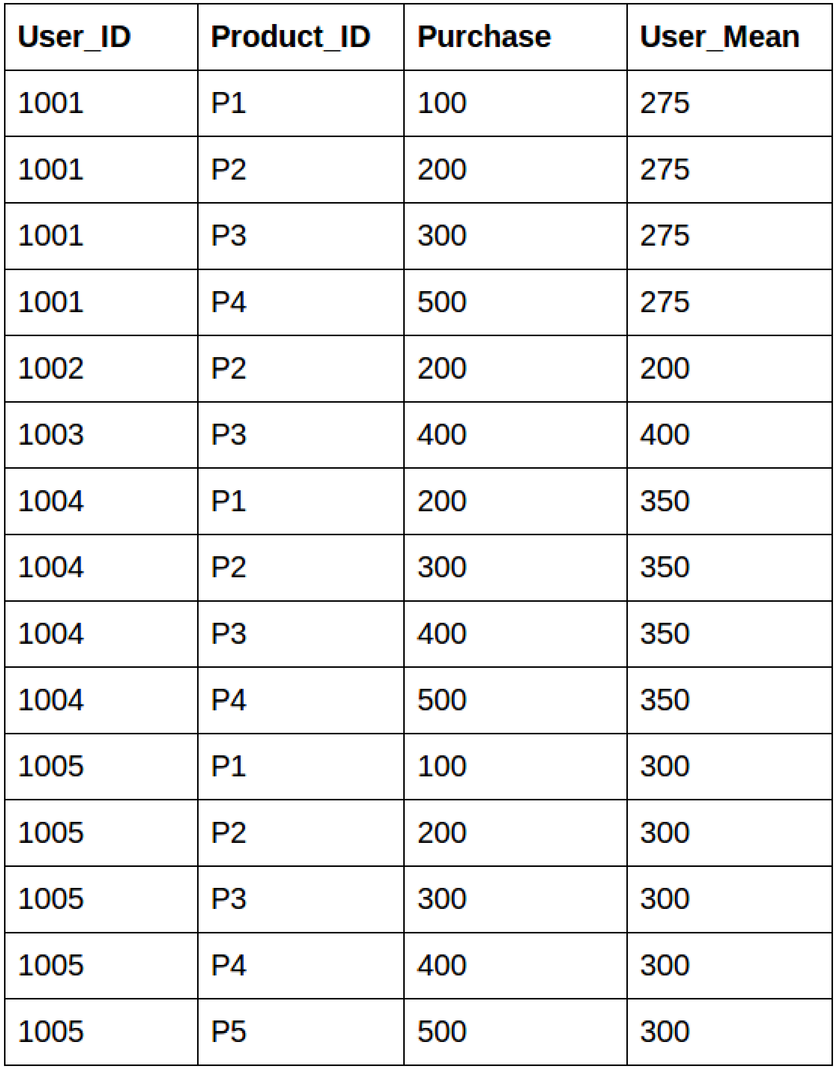
- Consider we have a dataset about a department store.
  - We can see that each user has bought multiple products with different purchase amounts.
  - We would like to know what is the mean purchase amount of each user



User_ID	Product_ID	Purchase
1001	P1	100
1001	P2	200
1001	P3	300
1001	P4	500
1002	P2	200
1003	P3	400
1004	P1	200
1004	P2	300
1004	P3	400
1004	P4	500
1005	P1	100
1005	P2	200
1005	P3	300
1005	P4	400
1005	P5	500

Example (cont.)

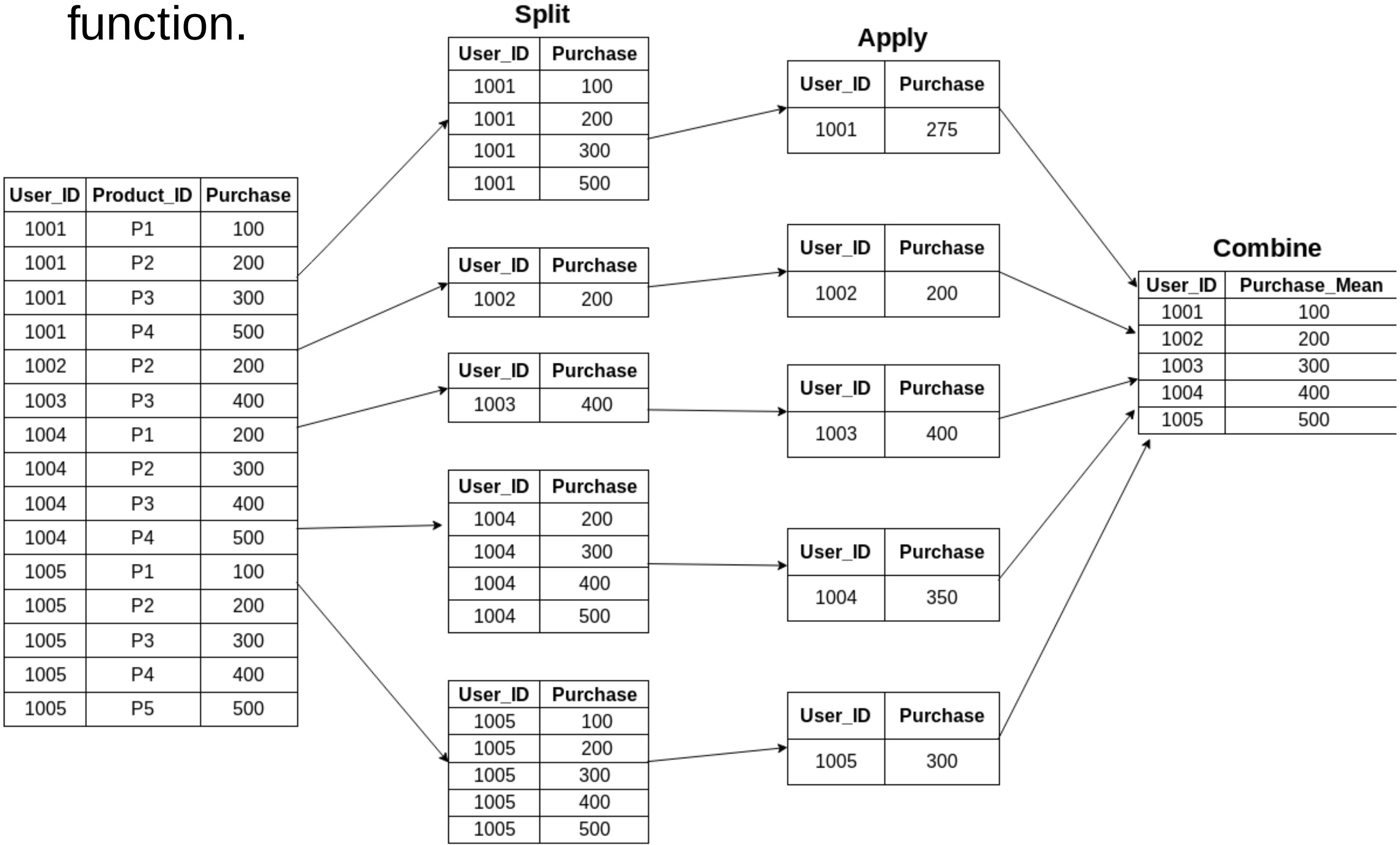
- There are multiple options to do this:
  1. Using Groupby followed by ***merge()***
  2. Transform function approach



User_ID	Product_ID	Purchase	User_Mean
1001	P1	100	275
1001	P2	200	275
1001	P3	300	275
1001	P4	500	275
1002	P2	200	200
1003	P3	400	400
1004	P1	200	350
1004	P2	300	350
1004	P3	400	350
1004	P4	500	350
1005	P1	100	300
1005	P2	200	300
1005	P3	300	300
1005	P4	400	300
1005	P5	500	300

Example: Option 1 – Merge

- The first approach is using groupby to aggregate the data, then merge this data back into the original dataframe using the merge() function.



User_ID	Product_ID	Purchase
1001	P1	100
1001	P2	200
1001	P3	300
1001	P4	500
1002	P2	200
1003	P3	400
1004	P1	200
1004	P2	300
1004	P3	400
1004	P4	500
1005	P1	100
1005	P2	200
1005	P3	300
1005	P4	400
1005	P5	500

Split

User_ID	Purchase
1001	100
1001	200
1001	300
1001	500

User_ID	Purchase
1002	200

User_ID	Purchase
1003	400

User_ID	Purchase
1004	200
1004	300
1004	400
1004	500

User_ID	Purchase
1005	100
1005	200
1005	300
1005	400
1005	500

Apply

User_ID	Purchase
1001	275

User_ID	Purchase
1002	200

User_ID	Purchase
1003	400

User_ID	Purchase
1004	350

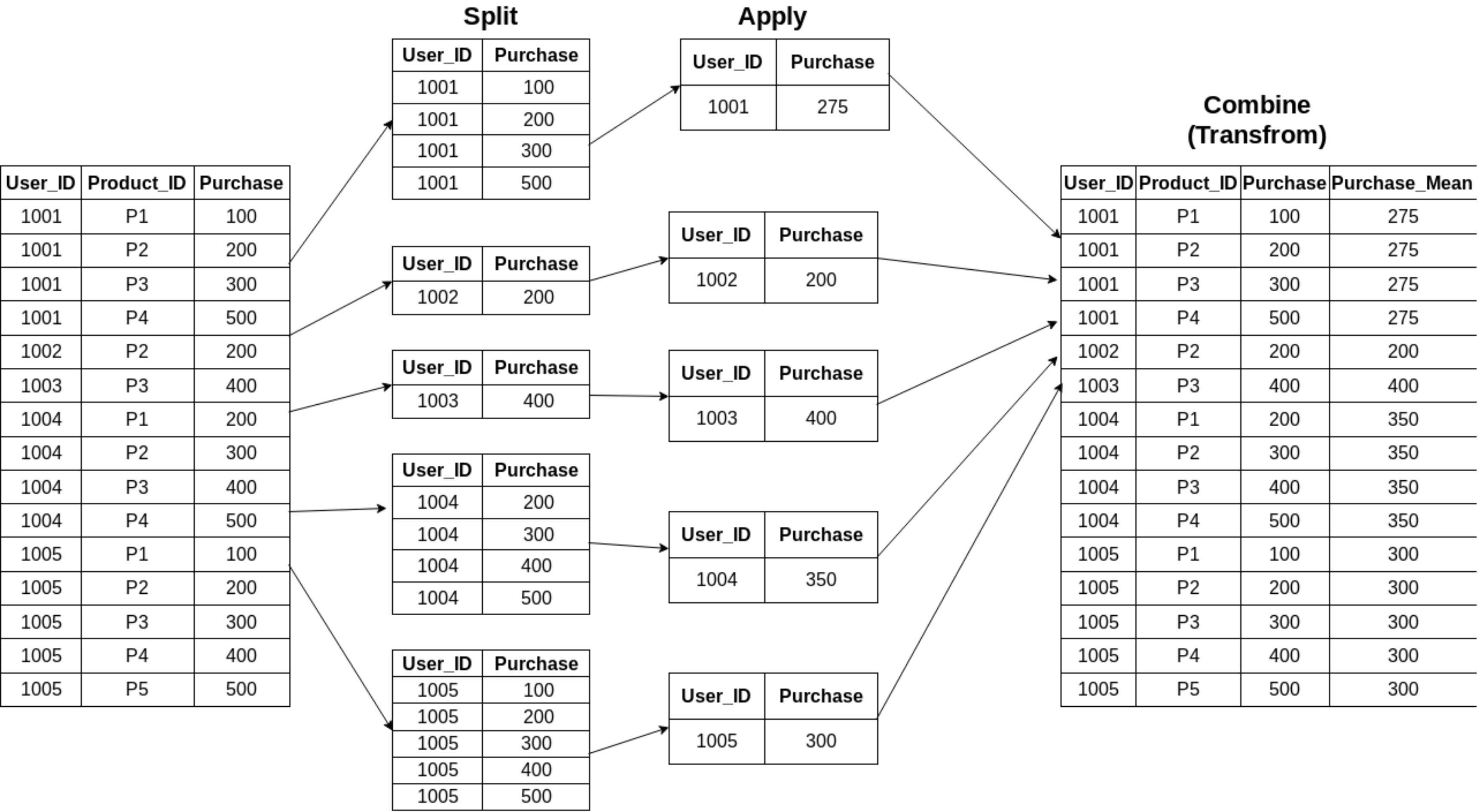
User_ID	Purchase
1005	300

Combine

User_ID	Purchase_Mean
1001	100
1002	200
1003	300
1004	400
1005	500

Example: Option 2 – Transform

- The transform function retains the same number of items as the original dataset after performing the transformation.



User_ID	Product_ID	Purchase
1001	P1	100
1001	P2	200
1001	P3	300
1001	P4	500
1002	P2	200
1003	P3	400
1004	P1	200
1004	P2	300
1004	P3	400
1004	P4	500
1005	P1	100
1005	P2	200
1005	P3	300
1005	P4	400
1005	P5	500

Split

User_ID	Purchase
1001	100
1001	200
1001	300
1001	500

User_ID	Purchase
1002	200

User_ID	Purchase
1003	400

User_ID	Purchase
1004	200
1004	300
1004	400
1004	500

User_ID	Purchase
1005	100
1005	200
1005	300
1005	400
1005	500

Apply

User_ID	Purchase
1001	275

User_ID	Purchase
1002	200

User_ID	Purchase
1003	400

User_ID	Purchase
1004	350

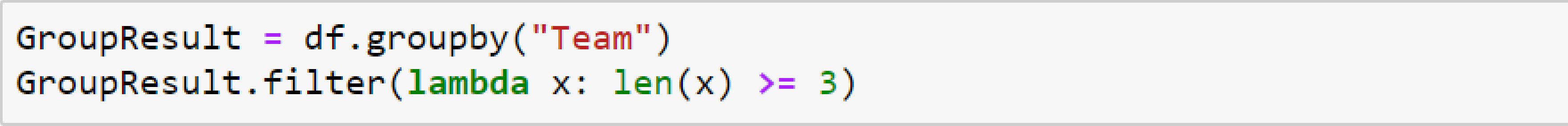
User_ID	Purchase
1005	300

Combine (Transfrom)

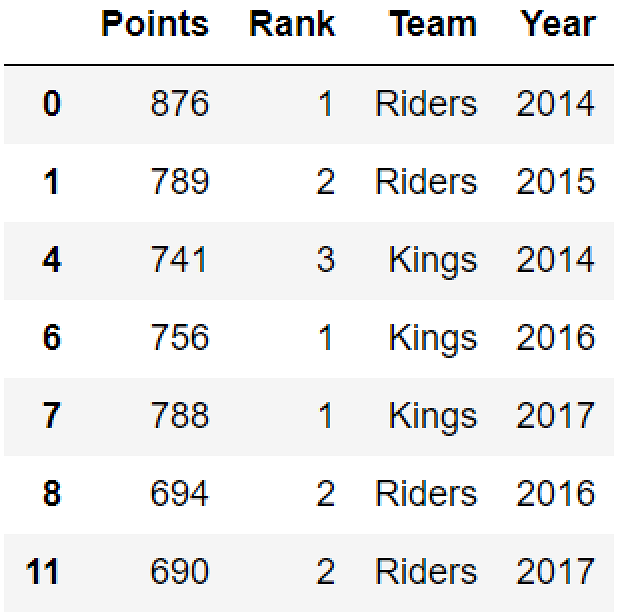
User_ID	Product_ID	Purchase	Purchase_Mean
1001	P1	100	275
1001	P2	200	275
1001	P3	300	275
1001	P4	500	275
1002	P2	200	200
1003	P3	400	400
1004	P1	200	350
1004	P2	300	350
1004	P3	400	350
1004	P4	500	350
1005	P1	100	300
1005	P2	200	300
1005	P3	300	300
1005	P4	400	300
1005	P5	500	300

Filtration

- □ Filtration filters the data on a defined criteria and returns the subset of data. The ***filter()*** function is used to filter the data



```
GroupResult = df.groupby("Team")
GroupResult.filter(lambda x: len(x) >= 3)
```

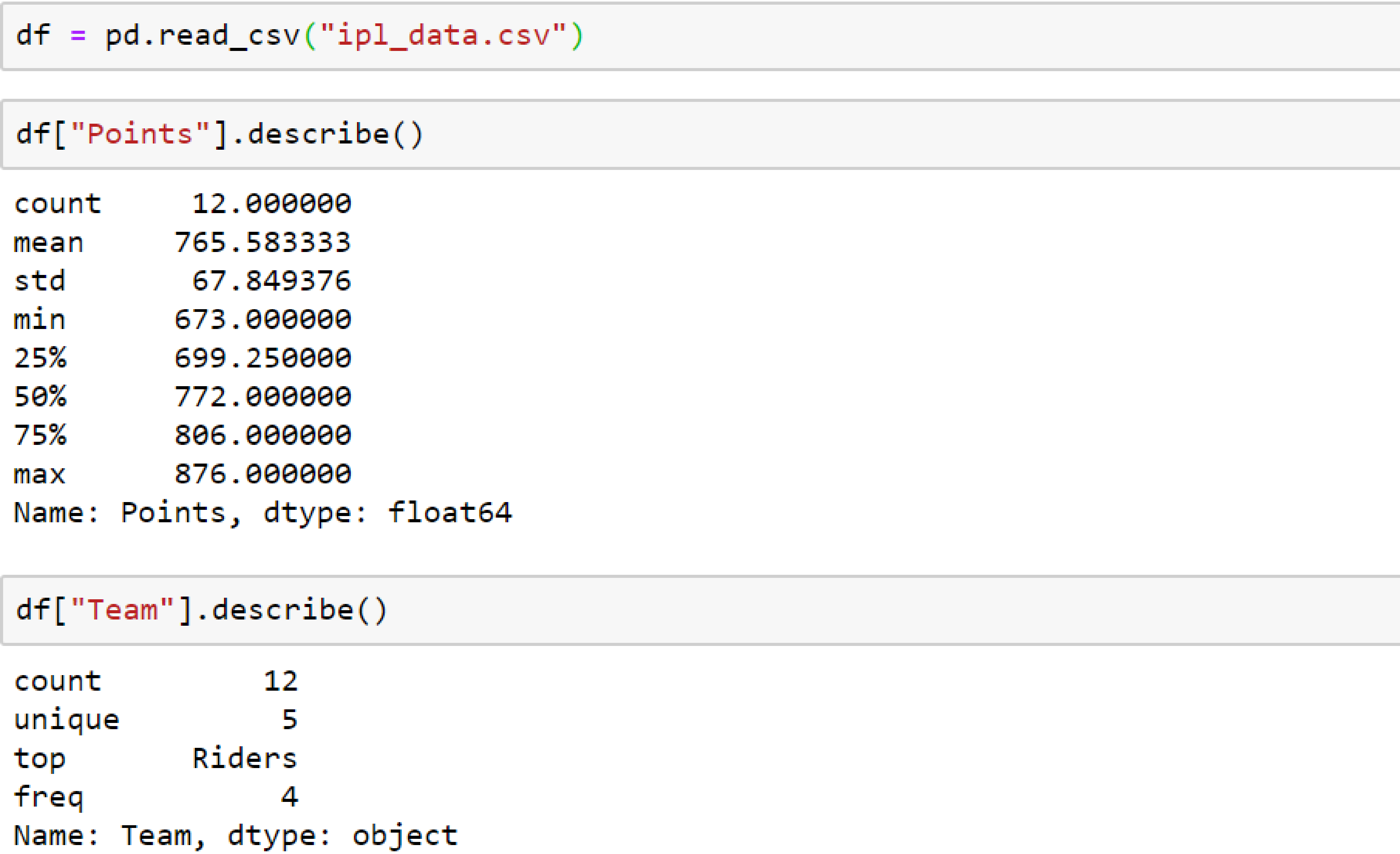


	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
4	741	3	Kings	2014
6	756	1	Kings	2016
7	788	1	Kings	2017
8	694	2	Riders	2016
11	690	2	Riders	2017

Describe Column

- To see some of the core statistics about a particular column, you can use the ***describe()*** function.
  - For numeric columns, ***describe()*** returns basic statistics: the value count, mean, standard deviation, minimum, maximum, and 25th, 50th, and 75th quantiles for the data in a column.
  - For string columns, ***describe()*** returns the value count, the number of unique entries, the most frequently occurring value (‘top’), and the number of times the top value occurs (‘freq’)

Example



```
df = pd.read_csv("ipl_data.csv")
```

```
df["Points"].describe()
```

```
count     12.000000
mean     765.583333
std       67.849376
min      673.000000
25%      699.250000
50%      772.000000
75%      806.000000
max      876.000000
Name: Points, dtype: float64
```

```
df["Team"].describe()
```

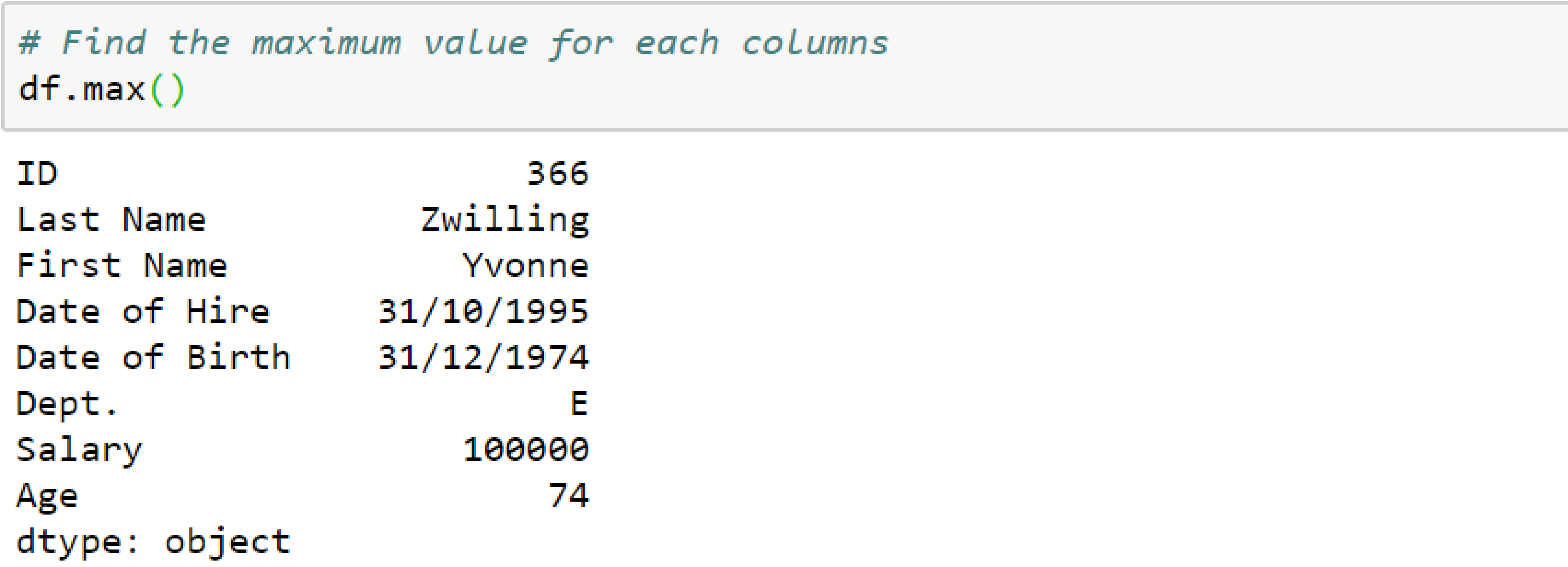
```
count          12
unique          5
top        Riders
freq            4
Name: Team, dtype: object
```

DataFrames Calculation Method

Method	Description
abs	Find the absolute value
corr	Find the correlation between columns in a DataFrame
count	Count the number of non-null values in each DataFrame column
max	Finds the highest value in each column
mean	Find the mean of each row or of each column
median	Find the median of each column.
min	Find the lowest value in each column
mode	Find the Mode
std	Find the standard deviation of each column

Maximum

- The ***max()*** method is used to find the maximum value for each column

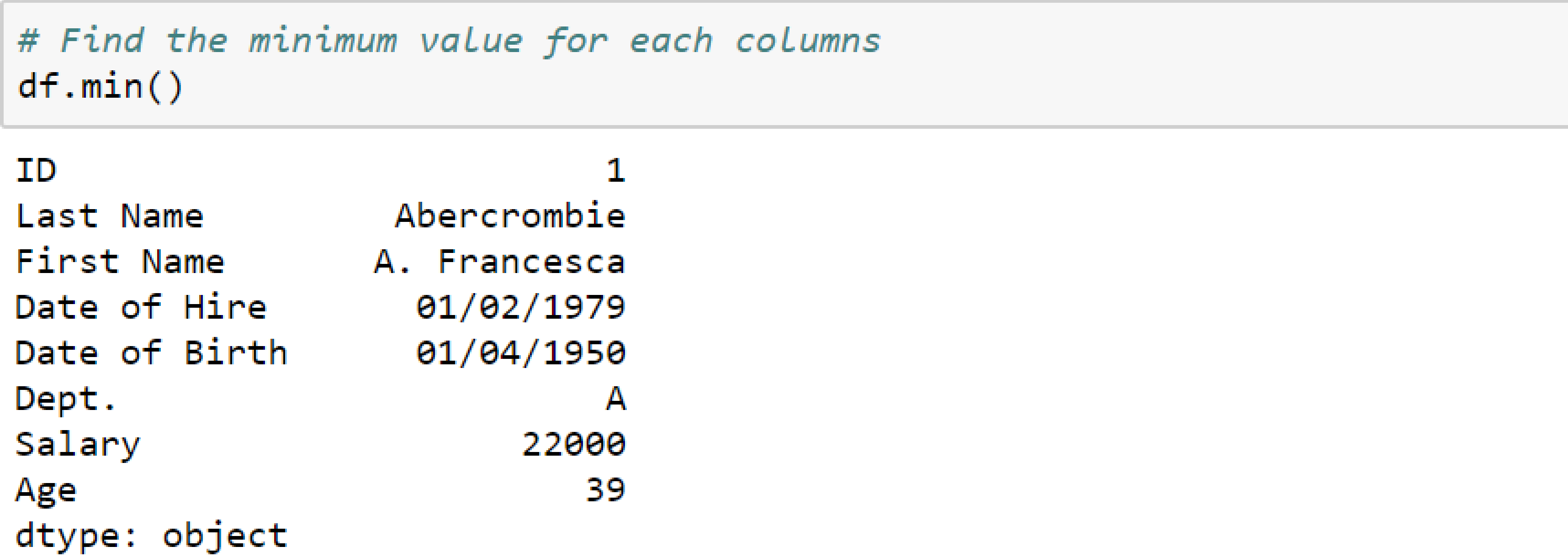


```
# Find the maximum value for each columns
df.max()
```

```
ID                       366
Last Name           Zwilling
First Name            Yvonne
Date of Hire      31/10/1995
Date of Birth     31/12/1974
Dept.                      E
Salary                100000
Age                       74
dtype: object
```

Minimum

- The *min( )* method is used to find the minimum value for each column

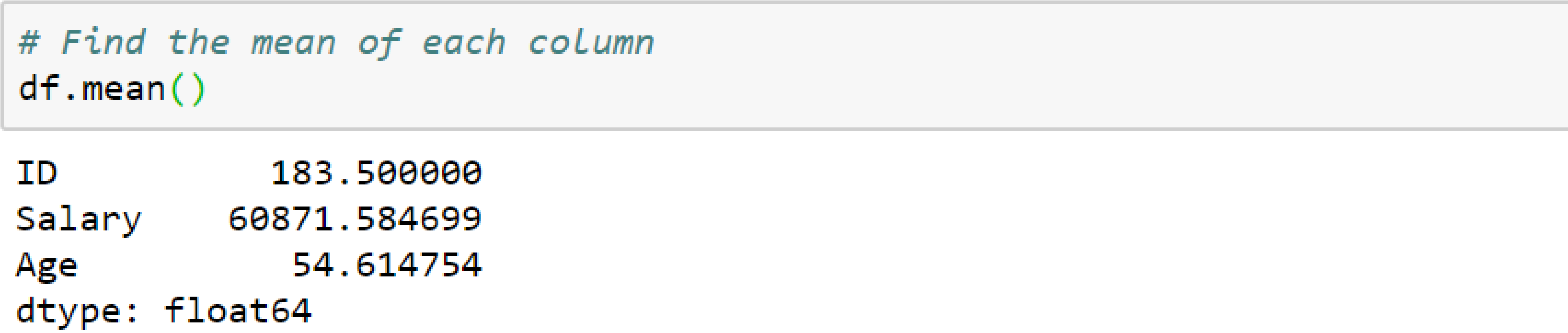


```
# Find the minimum value for each columns
df.min()
```

```
ID                          1
Last Name         Abercrombie
First Name       A. Francesca
Date of Hire       01/02/1979
Date of Birth      01/04/1950
Dept.                       A
Salary                  22000
Age                        39
dtype: object
```

Mean and Median

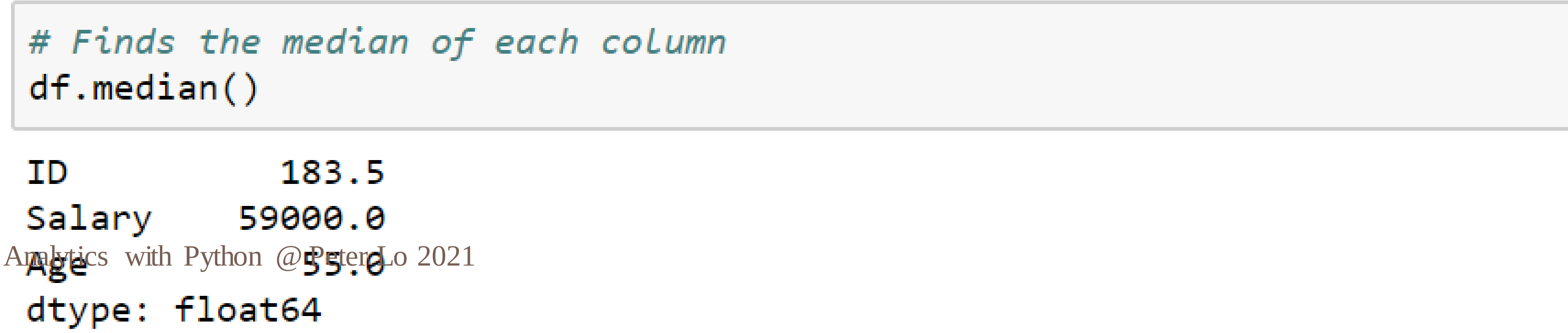
- The ***mean( )*** method is used to find the mean of each row or of each column



```
# Find the mean of each column
df.mean()
```

```
ID            183.500000
Salary      60871.584699
Age            54.614754
dtype: float64
```

- The ***median( )*** method is used to find the median of each row or of each column

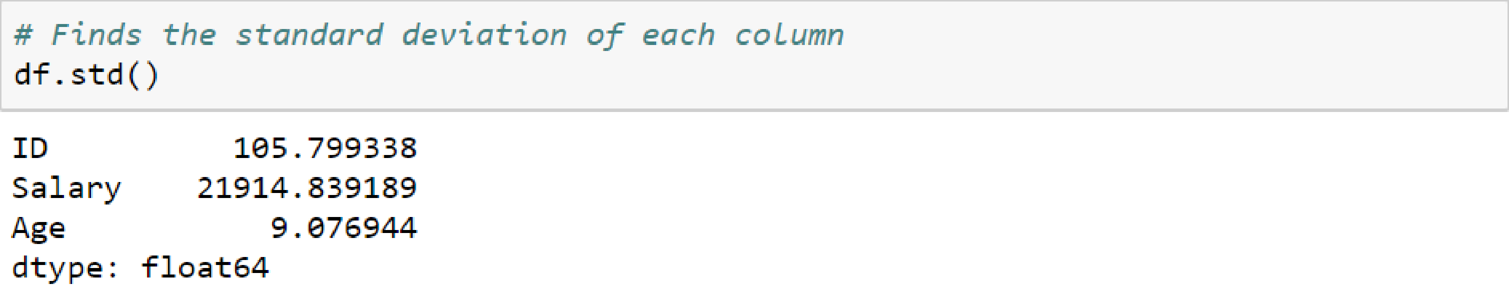


```
# Finds the median of each column
df.median()
```

```
ID          183.5
Salary    59000.0
Age          55.0
dtype: float64
```

Standard Deviation and Correlation

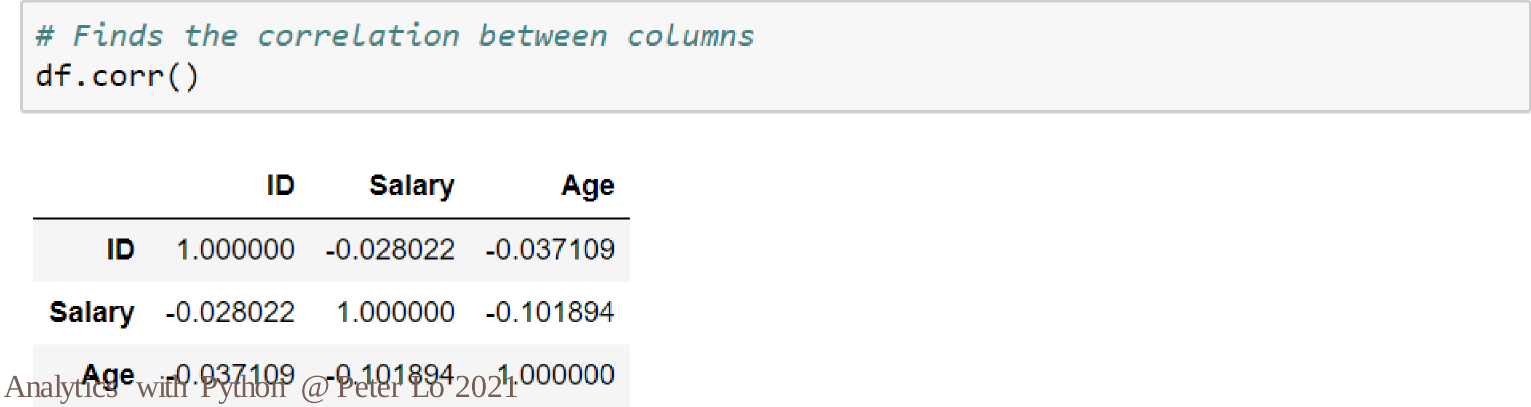
- The ***std()*** method is used to finds the standard deviation of each column



```
# Finds the standard deviation of each column
df.std()
```

```
ID            105.799338
Salary      21914.839189
Age             9.076944
dtype: float64
```

- The ***corr()*** method is used to finds the correlation between columns in a DataFrame

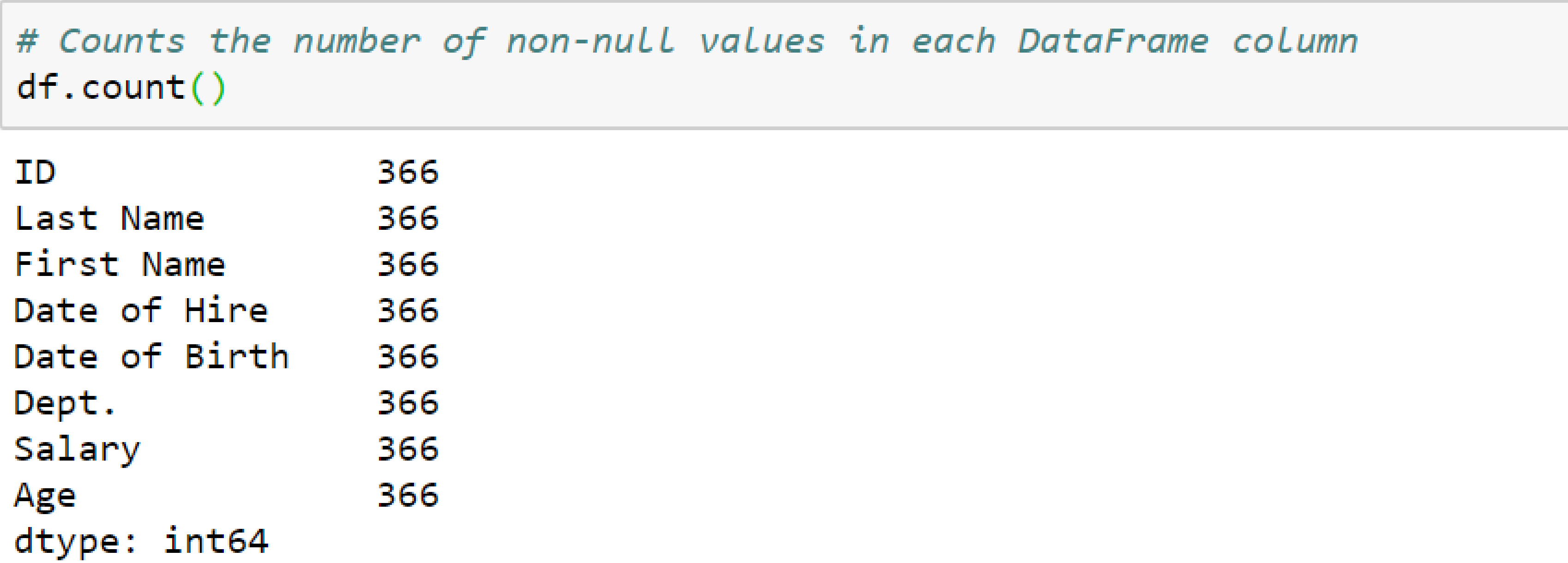


```
# Finds the correlation between columns
df.corr()
```

	ID	Salary	Age
ID	1.000000	-0.028022	-0.037109
Salary	-0.028022	1.000000	-0.101894
Age	-0.037109	-0.101894	1.000000

Counter

- The count() method is used to counts the number of non-null values in each DataFrame column

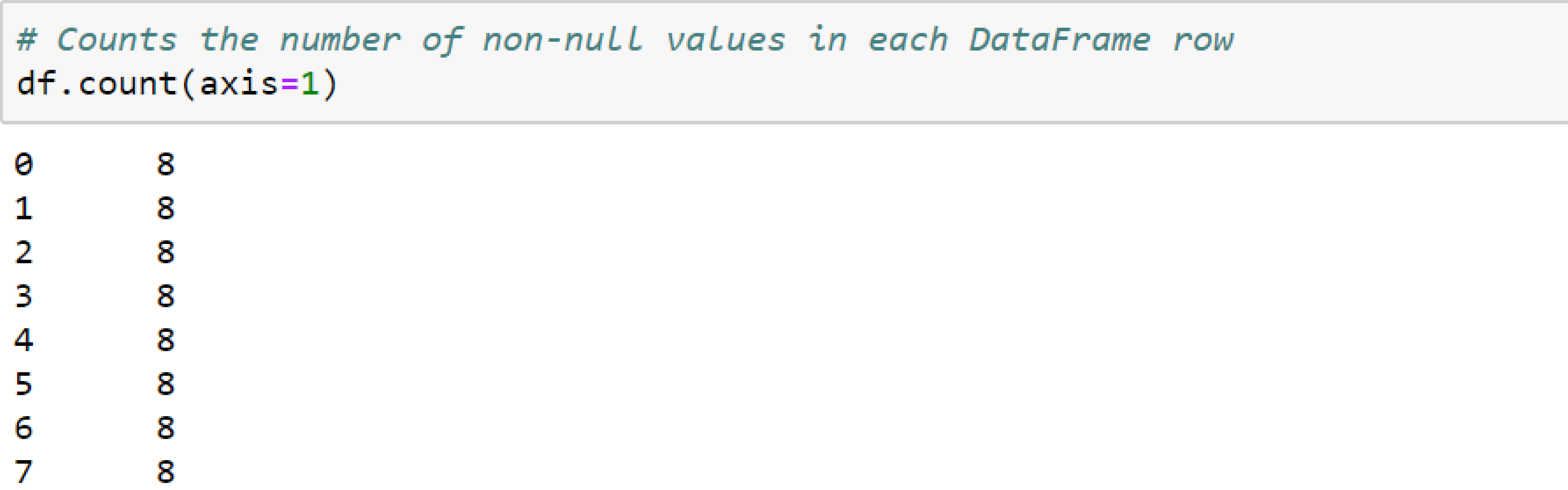


```
# Counts the number of non-null values in each DataFrame column
df.count()
```

```
ID               366
Last Name        366
First Name       366
Date of Hire     366
Date of Birth    366
Dept.            366
Salary           366
Age              366
dtype: int64
```

Calculation by Row

- We can modify the axis keyword argument to mean in order to compute the mean of each row or of each column.
- By default, axis is equal to 0, and will compute the mean of each column.
- We can also set it to 1 to compute the mean of each row



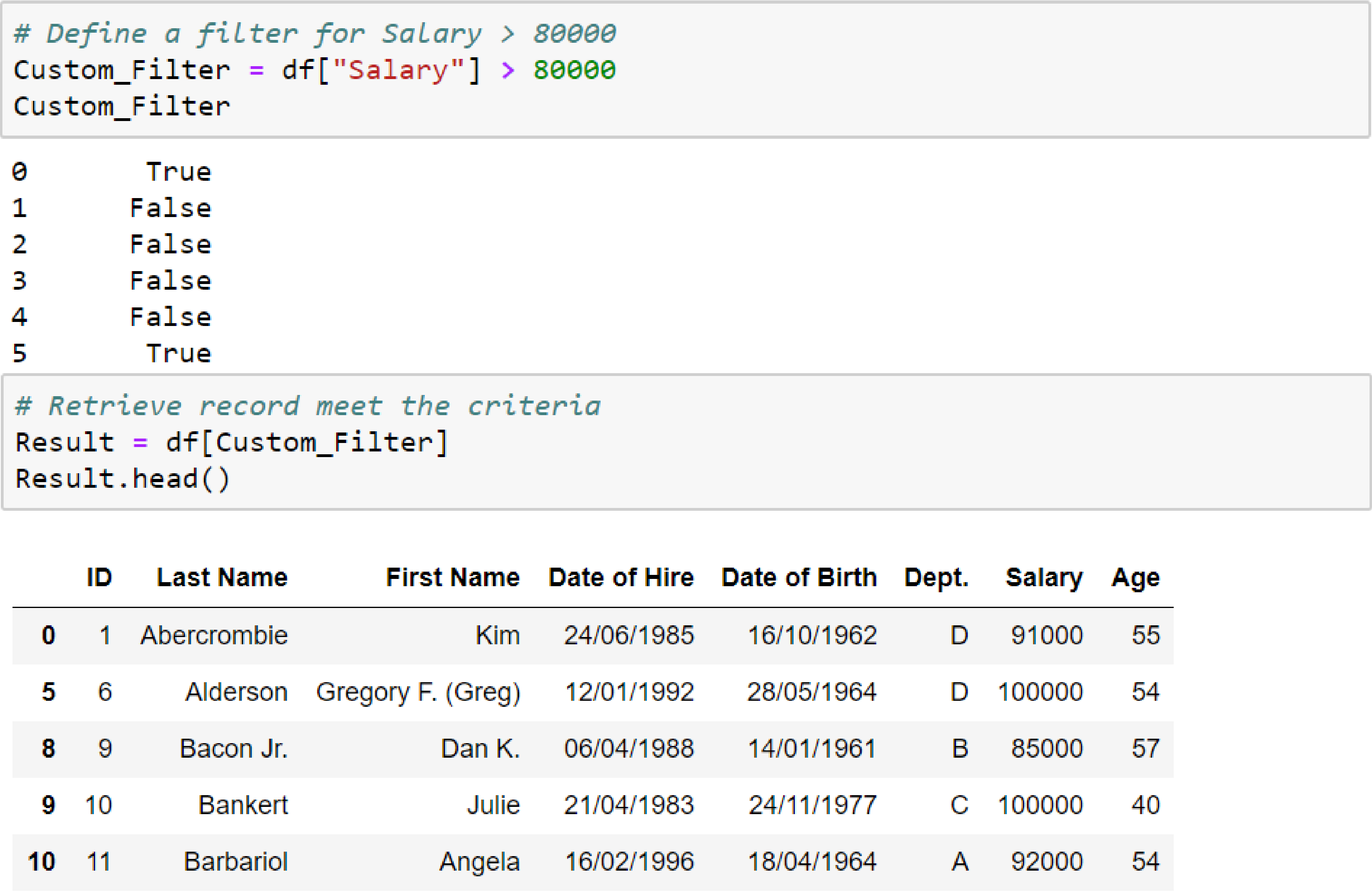
```
# Counts the number of non-null values in each DataFrame row
df.count(axis=1)
```

```
0       8
1       8
2       8
3       8
4       8
5       8
6       8
7       8
```

Boolean Indexing and Filtering

- We could start by doing a comparison.
- The comparison compares each value in a Series to a specified value, then generate a Series full of Boolean values indicating the status of the comparison.
- Once we have a Boolean Series, we can use it to select only rows in a DataFrame where the Series contains the value True

Example 1



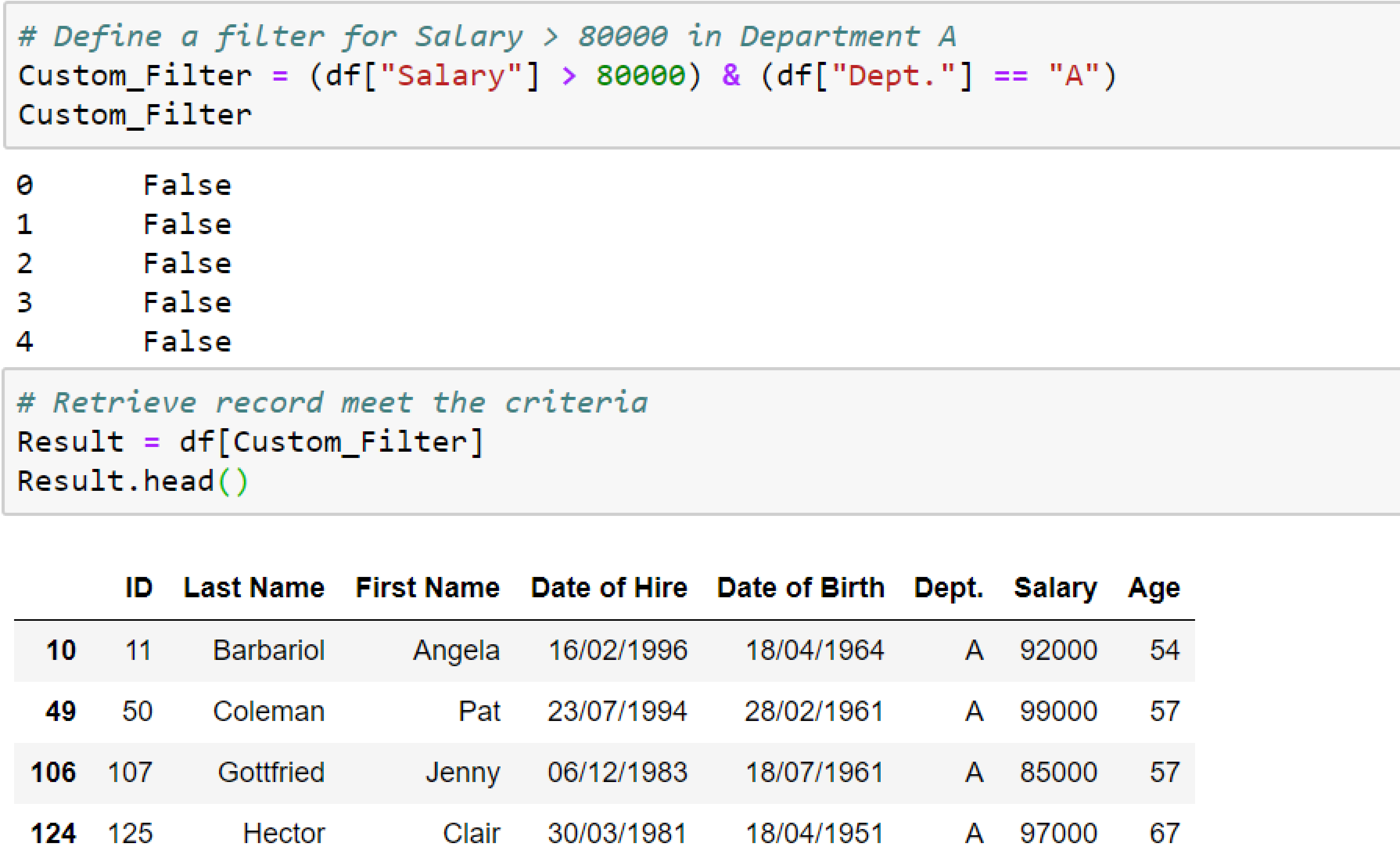
```
# Define a filter for Salary > 80000
Custom_Filter = df["Salary"] > 80000
Custom_Filter
```

```
0        True
1       False
2       False
3       False
4       False
5        True
```

```
# Retrieve record meet the criteria
Result = df[Custom_Filter]
Result.head()
```

	ID	Last Name	First Name	Date of Hire	Date of Birth	Dept.	Salary	Age
0	1	Abercrombie	Kim	24/06/1985	16/10/1962	D	91000	55
5	6	Alderson	Gregory F. (Greg)	12/01/1992	28/05/1964	D	100000	54
8	9	Bacon Jr.	Dan K.	06/04/1988	14/01/1961	B	85000	57
9	10	Bankert	Julie	21/04/1983	24/11/1977	C	100000	40
10	11	Barbariol	Angela	16/02/1996	18/04/1964	A	92000	54

Example 2



```
# Define a filter for Salary > 80000 in Department A
Custom_Filter = (df["Salary"] > 80000) & (df["Dept."] == "A")
Custom_Filter
```

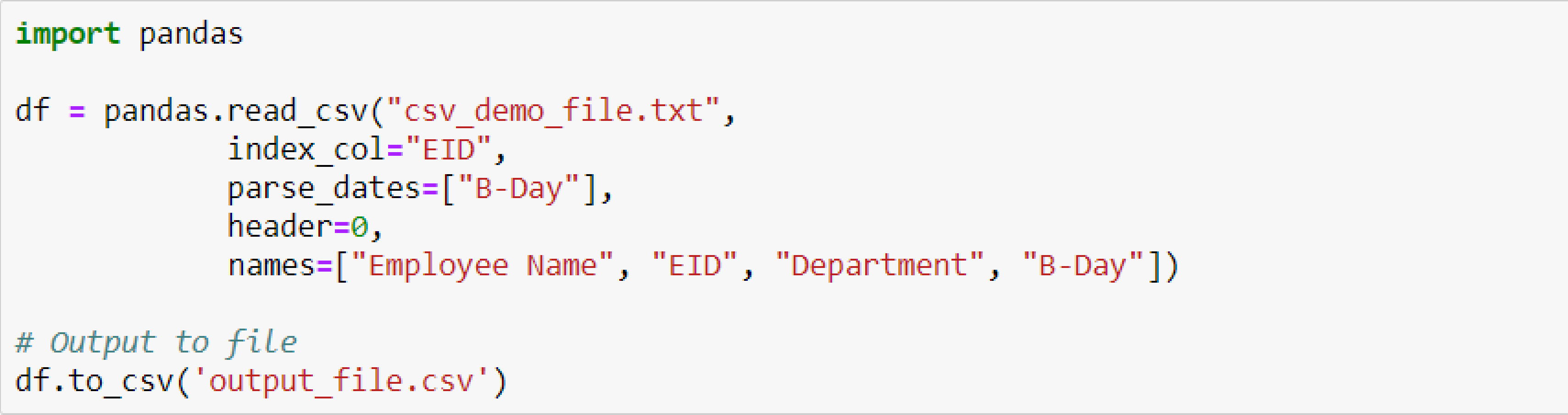
```
0        False
1        False
2        False
3        False
4        False
```

```
# Retrieve record meet the criteria
Result = df[Custom_Filter]
Result.head()
```

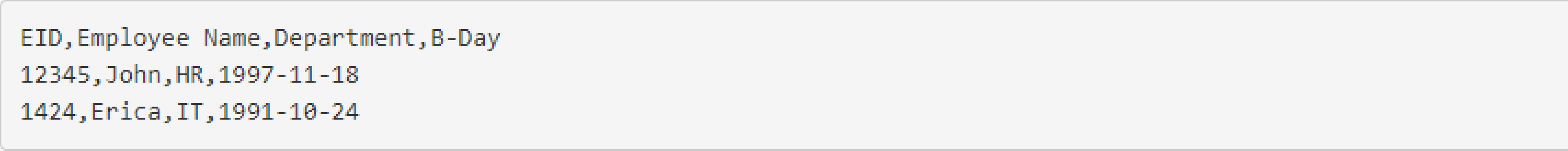
	ID	Last Name	First Name	Date of Hire	Date of Birth	Dept.	Salary	Age
10	11	Barbariol	Angela	16/02/1996	18/04/1964	A	92000	54
49	50	Coleman	Pat	23/07/1994	28/02/1961	A	99000	57
106	107	Gottfried	Jenny	06/12/1983	18/07/1961	A	85000	57
124	125	Hector	Clair	30/03/1981	18/04/1951	A	97000	67

Writing CSV Files with pandas

- The ***to_csv( )*** method save the contents of a DataFrame in a CSV.



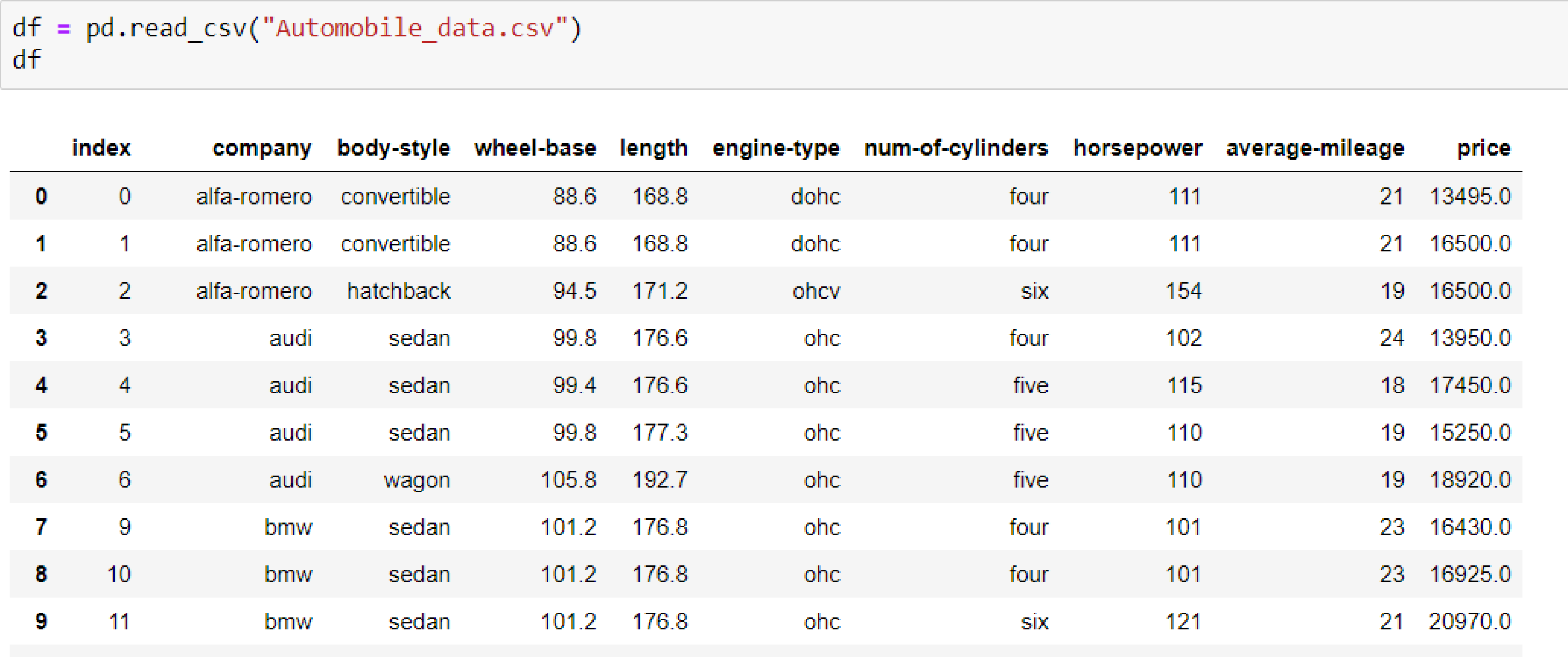
```python
import pandas

df = pandas.read_csv("csv_demo_file.txt",
             index_col="EID",
             parse_dates=["B-Day"],
             header=0,
             names=["Employee Name", "EID", "Department", "B-Day"])

# Output to file
df.to_csv('output_file.csv')
```



```
EID,Employee Name,Department,B-Day
12345,John,HR,1997-11-18
1424,Erica,IT,1991-10-24
```

Case Study

□ This automobile dataset has a different characteristic of an auto such as body-style, wheel-base, engine-type, price, mileage, horsepower and many more.

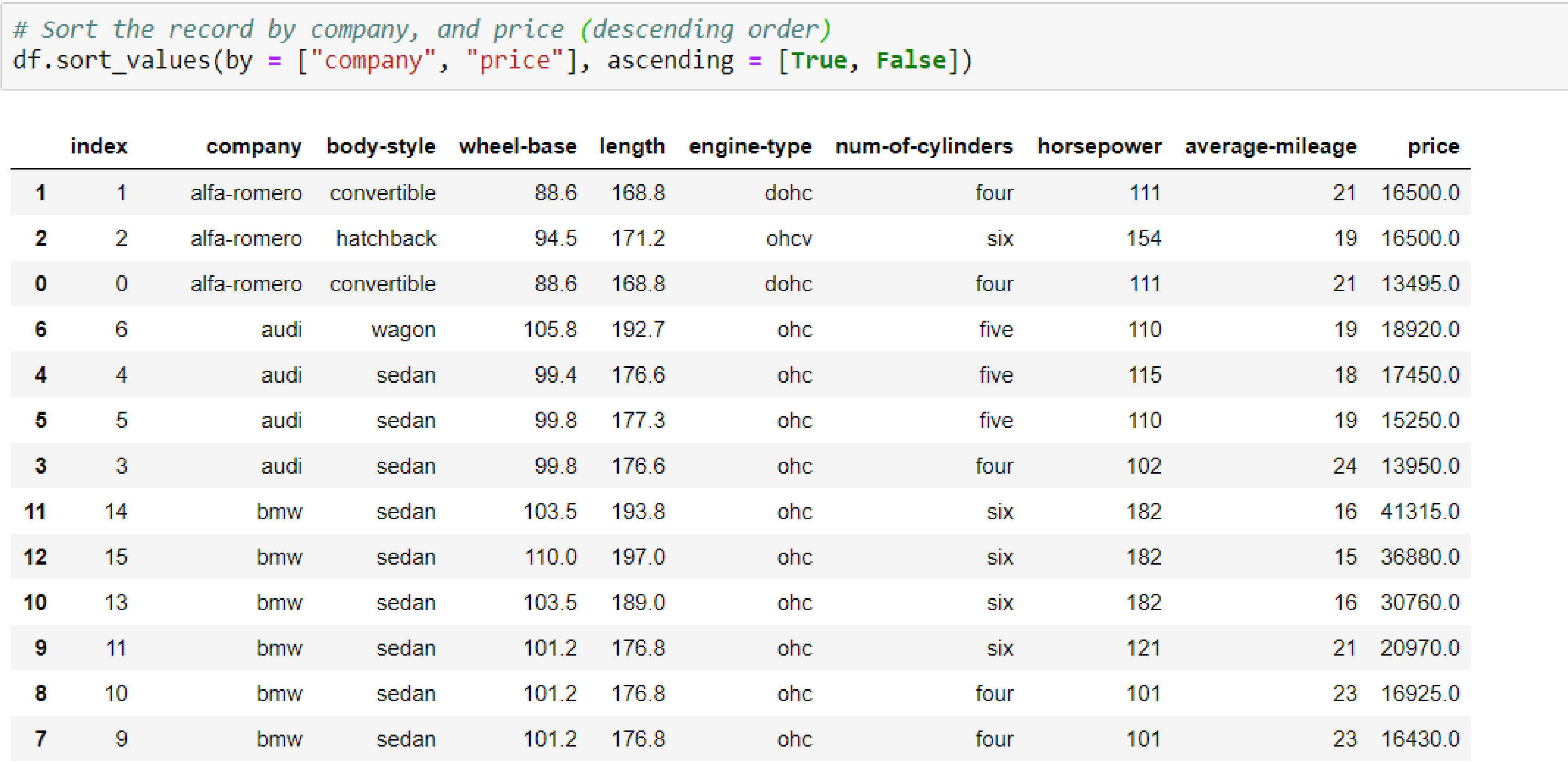


```
df = pd.read_csv("Automobile_data.csv")
df
```

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
0	0	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	13495.0
1	1	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	16500.0
2	2	alfa-romero	hatchback	94.5	171.2	ohcv	six	154	19	16500.0
3	3	audi	sedan	99.8	176.6	ohc	four	102	24	13950.0
4	4	audi	sedan	99.4	176.6	ohc	five	115	18	17450.0
5	5	audi	sedan	99.8	177.3	ohc	five	110	19	15250.0
6	6	audi	wagon	105.8	192.7	ohc	five	110	19	18920.0
7	9	bmw	sedan	101.2	176.8	ohc	four	101	23	16430.0
8	10	bmw	sedan	101.2	176.8	ohc	four	101	23	16925.0
9	11	bmw	sedan	101.2	176.8	ohc	six	121	21	20970.0

Case Study (cont.)

- □ Sort the price by company in descending order

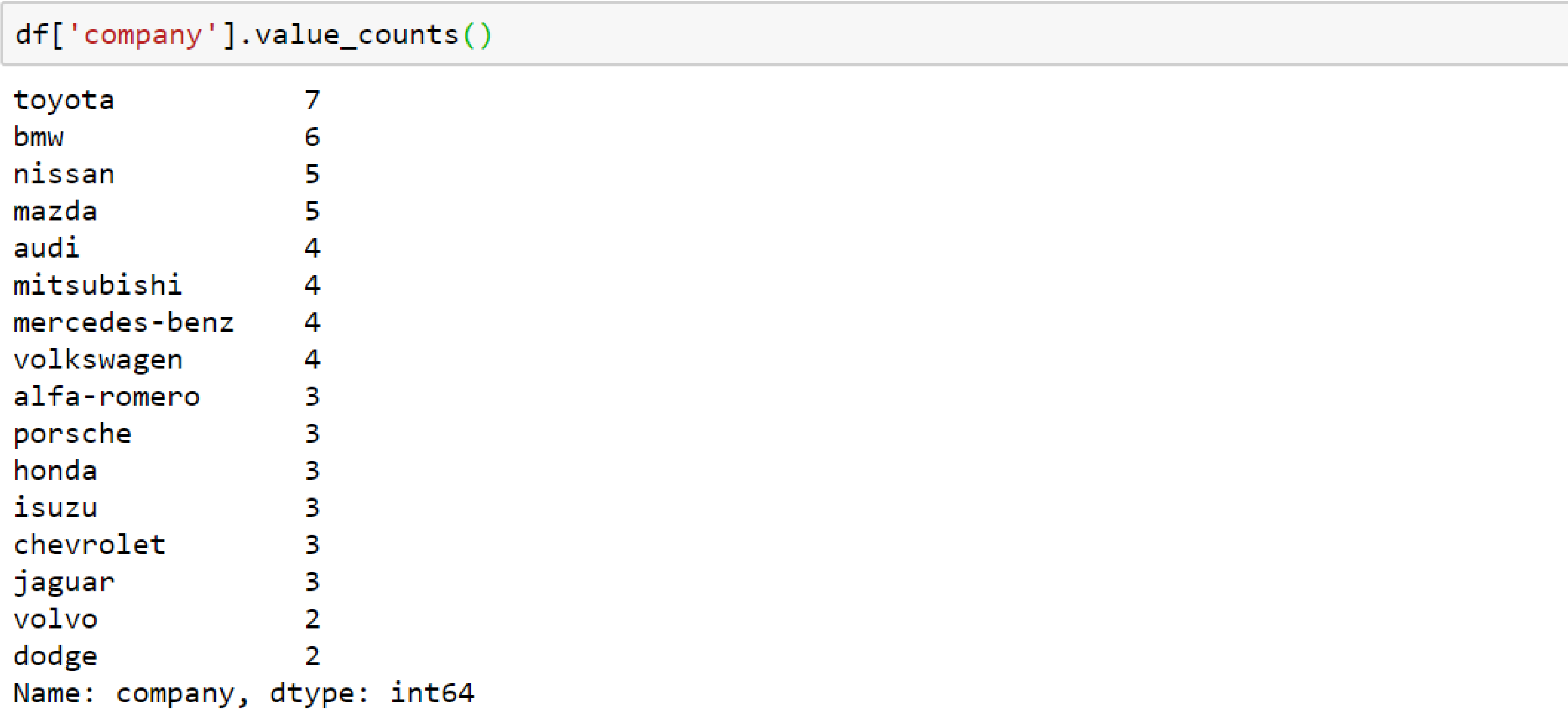


```
# Sort the record by company, and price (descending order)
df.sort_values(by = ["company", "price"], ascending = [True, False])
```

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
1	1	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	16500.0
2	2	alfa-romero	hatchback	94.5	171.2	ohcv	six	154	19	16500.0
0	0	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	13495.0
6	6	audi	wagon	105.8	192.7	ohc	five	110	19	18920.0
4	4	audi	sedan	99.4	176.6	ohc	five	115	18	17450.0
5	5	audi	sedan	99.8	177.3	ohc	five	110	19	15250.0
3	3	audi	sedan	99.8	176.6	ohc	four	102	24	13950.0
11	14	bmw	sedan	103.5	193.8	ohc	six	182	16	41315.0
12	15	bmw	sedan	110.0	197.0	ohc	six	182	15	36880.0
10	13	bmw	sedan	103.5	189.0	ohc	six	182	16	30760.0
9	11	bmw	sedan	101.2	176.8	ohc	six	121	21	20970.0
8	10	bmw	sedan	101.2	176.8	ohc	four	101	23	16925.0
7	9	bmw	sedan	101.2	176.8	ohc	four	101	23	16430.0

Case Study (cont.)

- Count the total cars per company

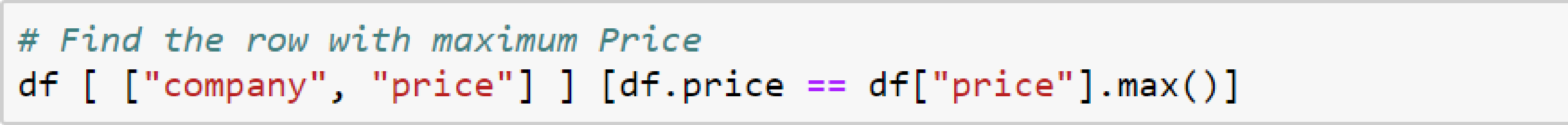


```
df['company'].value_counts()
```

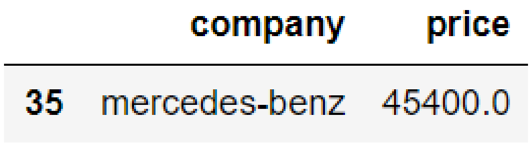
```
toyota           7
bmw              6
nissan           5
mazda            5
audi             4
mitsubishi       4
mercedes-benz    4
volkswagen       4
alfa-romero      3
porsche          3
honda            3
isuzu            3
chevrolet        3
jaguar           3
volvo            2
dodge            2
Name: company, dtype: int64
```

Case Study (cont.)

- □ Find the most expensive car with company name



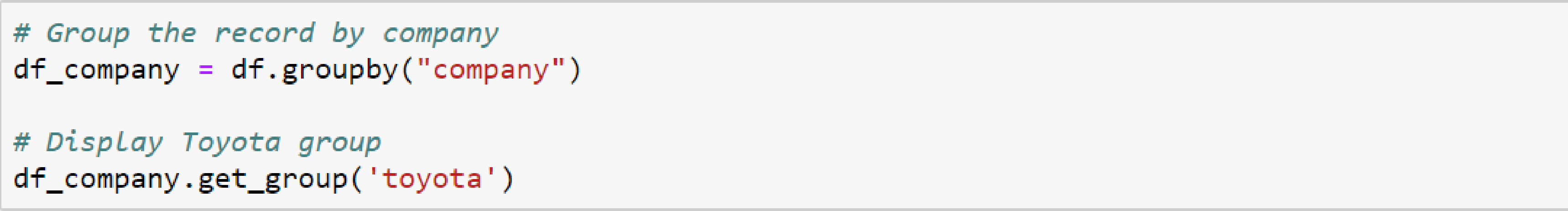
```
# Find the row with maximum Price
df [ ["company", "price"] ] [df.price == df["price"].max()]
```



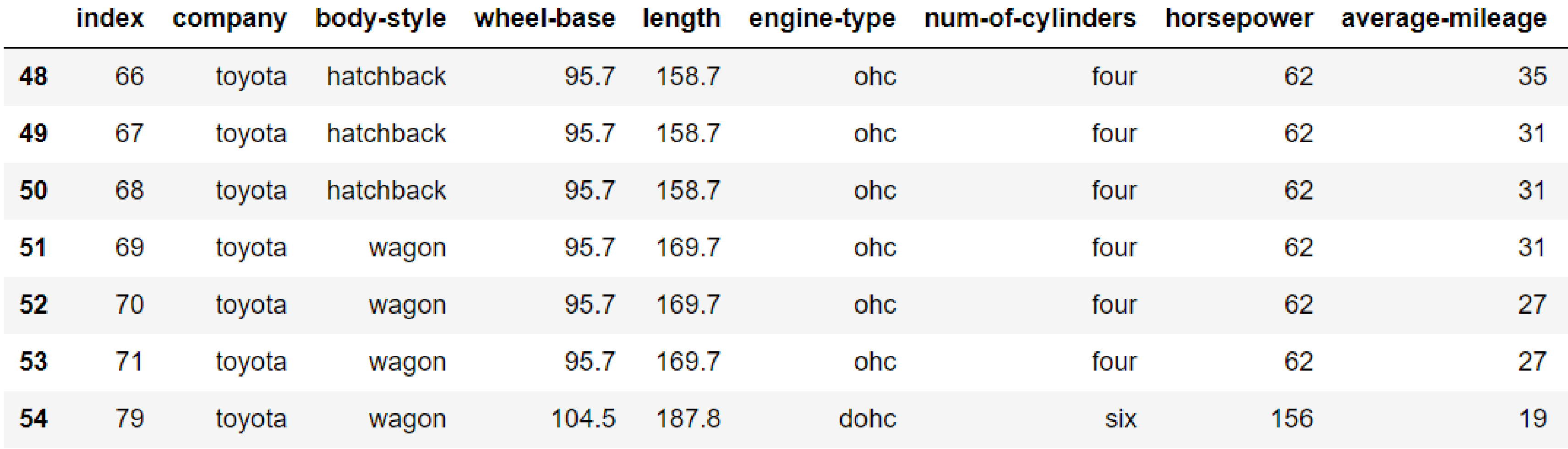
	company	price
35	mercedes-benz	45400.0

Case Study (cont.)

- Print all Toyota cars details



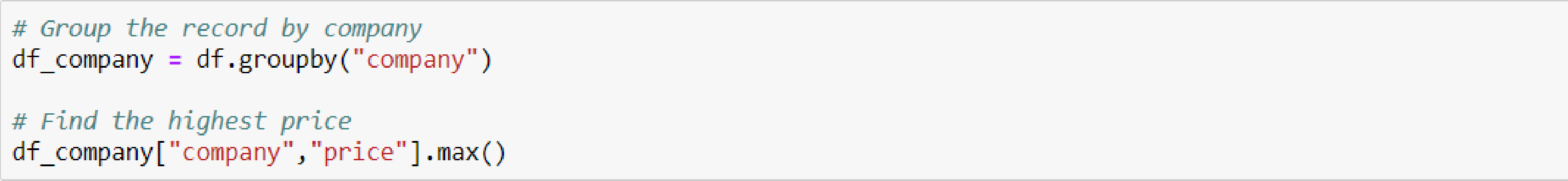
```
# Group the record by company
df_company = df.groupby("company")

# Display Toyota group
df_company.get_group('toyota')
```



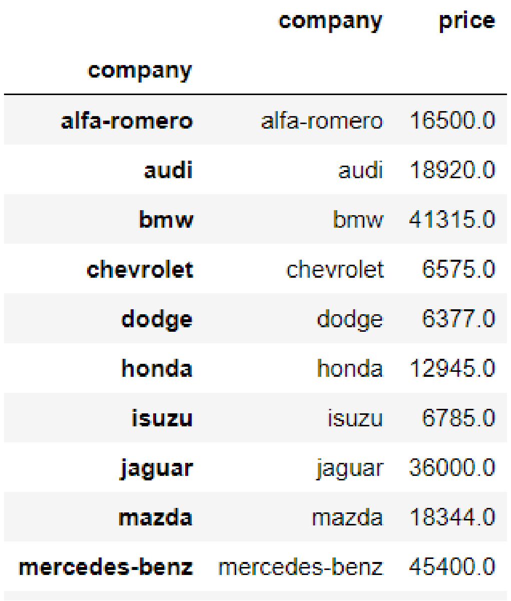
	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage
48	66	toyota	hatchback	95.7	158.7	ohc	four	62	35
49	67	toyota	hatchback	95.7	158.7	ohc	four	62	31
50	68	toyota	hatchback	95.7	158.7	ohc	four	62	31
51	69	toyota	wagon	95.7	169.7	ohc	four	62	31
52	70	toyota	wagon	95.7	169.7	ohc	four	62	27
53	71	toyota	wagon	95.7	169.7	ohc	four	62	27
54	79	toyota	wagon	104.5	187.8	dohc	six	156	19

Case Study (cont.)

- Find the most expensive car per company



```
# Group the record by company
df_company = df.groupby("company")

# Find the highest price
df_company["company","price"].max()
```



```
                    company     price
company
  alfa-romero   alfa-romero   16500.0
         audi          audi   18920.0
          bmw           bmw   41315.0
    chevrolet     chevrolet    6575.0
        dodge         dodge    6377.0
        honda         honda   12945.0
        isuzu         isuzu    6785.0
       jaguar        jaguar   36000.0
        mazda         mazda   18344.0
mercedes-benz mercedes-benz   45400.0
```