Module 3: Pandas DataFrame

Pandas Series is a one-dimensional array. Meanwhile, Pandas DataFrame is a two-dimensional, size-mutable, potentially heterogeneous tabular data structure with labelled axes.

Two-dimensional means it has rows and columns. To extract specific information we need to have both information on row and column.

In [1]:

```
import pandas as pd
```

Now we are going to read the nba.csv file.

- The file contains information about the player in the 2015/2016 season.
- NaN value means blank or null value and it is automatically assigned when it finds a missing value.
- The structure of a DataFrame is almost like a Series. It will only show the first 30 rows and last 30 rows. In between that, it will truncate the table and replace with ... symbol.
- Pandas DataFrame automatically generates indexes on the left side.
- At the bottom of the table, we can see the total number of rows and columns. For example in this table, there are 458 rows and 9 columns.

In [2]:

```
nba = pd.read_csv("data/nba.csv")
nba
```

Out[2]:

	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
...	...	...	...	...	...	...	...	...	...
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

458 rows × 9 columns

Methods and Attributes in Pandas DataFrame

Since DataFrame and Series are not the same type, there are some things which can be used in DataFrame, but not in Series and vice versa. However, they do share common methods and attributes.

.index attribute returns information about the object's index.

In [3]:

```
nba.index
```

Out[3]:

```
RangeIndex(start=0, stop=458, step=1)
```

.values attribute returns the value of the object.

In [4]:

```
nba.values
```

Out[4]:

```
array([['Avery Bradley', 'Boston Celtics', 0.0, ..., 180.0, 'Texas',
        7730337.0],
       ['Jae Crowder', 'Boston Celtics', 99.0, ..., 235.0, 'Marquette',
        6796117.0],
       ['John Holland', 'Boston Celtics', 30.0, ..., 205.0,
        'Boston University', nan],
       ...,
       ['Tibor Pleiss', 'Utah Jazz', 21.0, ..., 256.0, nan, 2900000.0],
       ['Jeff Withey', 'Utah Jazz', 24.0, ..., 231.0, 'Kansas', 947276.0],
       [nan, nan, nan, ..., nan, nan, nan]], dtype=object)
```

.shape attribute returns the number rows and columns of the object.

In [5]:

```
nba.shape
```

Out[5]:

```
(458, 9)
```

.dtypes attribues return the data type for each columns.

Hint : object means string

In [6]:

```
nba.dtypes
```

Out[6]:

```
Name          object
Team          object
Number       float64
Position      object
Age          float64
Height        object
Weight       float64
College       object
Salary       float64
dtype: object
```

.columns returns a list of column names.

In [7]:

```
nba.columns
```

Out[7]:

```
Index(['Name', 'Team', 'Number', 'Position', 'Age', 'Height', 'Weight',
       'College', 'Salary'],
      dtype='object')
```

.axes attribute returns information on our axes, which are index and column names.

In [8]:

```
nba.axes
```

Out[8]:

```
[RangeIndex(start=0, stop=458, step=1),
 Index(['Name', 'Team', 'Number', 'Position', 'Age', 'Height', 'Weight',
        'College', 'Salary'],
       dtype='object')]
```

.head() : returns first N row(s) of the DataFrame. By default, N=5.

In [9]:

```
nba.head()
```

Out[9]:

	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

.tail() : returns last N row(s) of the DataFrame. Default value of N is 5.

In [10]:

```
nba.tail(3)
```

Out[10]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

.info() method returns a summary of the DataFrame. This makes understanding the data a lot easier.

According to the DataFrame's summary, we can see that College and Salary columns have 373 and 446 rows respectively. However, the total number of rows is actually 458. This means that these columns have some missing elements.

In [11]:

```
nba.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 458 entries, 0 to 457
Data columns (total 9 columns):
Name        457 non-null object
Team        457 non-null object
Number      457 non-null float64
Position    457 non-null object
Age         457 non-null float64
Height      457 non-null object
Weight      457 non-null float64
College     373 non-null object
Salary      446 non-null float64
dtypes: float64(4), object(5)
memory usage: 32.3+ KB
```

.get_dtypes_counts returns information on our columns data type. In this example, there are 4 columns with float data type and 5 with string.

In [12]:

```
nba.get_dtype_counts()
```

```
C:\Users\Ismail\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureW
arning: `get_dtype_counts` has been deprecated and will be removed in a futu
re version. For DataFrames use `.dtypes.value_counts()
  """Entry point for launching an IPython kernel.
```

Out[12]:

```
float64    4
object     5
dtype: int64
```

.sum() method returns the summation of numerical columns' elements. We can choose either to sum horizontally or vertically.

In [13]:

```
rev = pd.read_csv("data/revenue.csv", index_col="Date")
rev.head()
```

Out[13]:

Date	New York	Los Angeles	Miami
1/1/16	985	122	499
1/2/16	738	788	534
1/3/16	14	20	933
1/4/16	730	904	885
1/5/16	114	71	253

In this example, we are adding all the rows vertically. We will get the total revenue of New York which is 5475.

In [14]:

```
rev.sum()
```

Out[14]:

```
New York       5475
Los Angeles    5134
Miami          5641
dtype: int64
```

When we use parameter axis and give argument of 1 or columns, it does summation operation over the column axis.

In [15]:

```
rev.sum(axis="columns")
```

Out[15]:

```
Date
1/1/16     1606
1/2/16     2060
1/3/16      967
1/4/16     2519
1/5/16      438
1/6/16     1935
1/7/16     1234
1/8/16     2313
1/9/16     2623
1/10/16     555
dtype: int64
```

We can also create a new column in the DataFrame to store the results of **sum()** method.

In [16]:

```
rev["Sum by day"] = rev.sum(axis=1)
```

In [17]:

```
rev.head()
```

Out[17]:

Date	New York	Los Angeles	Miami	Sum by day
1/1/16	985	122	499	1606
1/2/16	738	788	534	2060
1/3/16	14	20	933	967
1/4/16	730	904	885	2519
1/5/16	114	71	253	438

Extract One Column from DataFrame

To extract values of a specific column, you can call the column's name on the DataFrame variable. For example, **nba.Name** will return all the values in the "Name" column of "nba" DataFrame.

*Hint : Always remember the column name is **Case Sensitive**. Capital and small letters will give different results.*

In [18]:

```
nba = pd.read_csv("data/nba.csv")
nba.head(3)
```

Out[18]:

	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

In [19]:

```
nba.Name.head()
```

Out[19]:

```
0    Avery Bradley
1      Jae Crowder
2     John Holland
3      R.J. Hunter
4    Jonas Jerebko
Name: Name, dtype: object
```

Another way is to add square brackets “[ ]” around the column’s name.

In [20]:

```
nba["Name"].head()
```

Out[20]:

```
0    Avery Bradley
1      Jae Crowder
2     John Holland
3      R.J. Hunter
4    Jonas Jerebko
Name: Name, dtype: object
```

It is often better to use the second way as the first one may not work well with column names that contain spaces. For example, "Player Name".

When extracting a specific column, the result will be returned in Pandas Series. To make it return in Pandas DataFrame, we can add another "[ ]" bracket around the column name.

In [21]:

```
nba[["Name"]].head()
```

Out[21]:

	Name
0	Avery Bradley
1	Jae Crowder
2	John Holland
3	R.J. Hunter
4	Jonas Jerebko

Select Two or More Columns in Data Frame

We can extract values of more than one column using the same way by specifying **all the columns we want** inside the square brackets. For example, if we want to extract "Name" and "Salary" column.

Hint : The order of how they are written is not important

In [22]:

```
nba = pd.read_csv("data/nba.csv")
nba.head(3)
```

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

In [23]:

```
nba[["Name", "Salary"]].head()
```

Out[23]:

	Name	Salary
0	Avery Bradley	7730337.0
1	Jae Crowder	6796117.0
2	John Holland	NaN
3	R.J. Hunter	1148640.0
4	Jonas Jerebko	5000000.0

In [24]:

```
nba[[ "Team" , "Name", "College"]].head()
```

Out[24]:

	Team	Name	College
0	Boston Celtics	Avery Bradley	Texas
1	Boston Celtics	Jae Crowder	Marquette
2	Boston Celtics	John Holland	Boston University
3	Boston Celtics	R.J. Hunter	Georgia State
4	Boston Celtics	Jonas Jerebko	NaN

Add a new column to DataFrame

We can also use this way to add column which does not exist in the DataFrame and assign values into it.

For example, here we are adding a new column named "Sport" into the DataFrame and setting "Basketball" as its values.

In [25]:

```
nba = pd.read_csv("data/nba.csv")
nba.head(3)
```

Out[25]:

	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

In [26]:

```
nba["Sport"] = "Basketball"
```

In [27]:

```
nba.head()
```

Out[27]:

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

.insert() method also does the same job. However, we can be more specific on the location of our new column.

- **loc**: location of the new column
- **column** : name for the new column
- **value**: the value for the new column

In [28]:

```
nba = pd.read_csv("data/nba.csv")
```

In [29]:

```
nba.insert(loc = 3, column = "Sport" , value = "Basketball")
```

In [30]:

```
nba.head(3)
```

Out[30]:

	Name	Team	Number	Sport	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	Basketball	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	Basketball	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	Basketball	SG	27.0	6-5	205.0	Boston University	NaN

Broadcasting Operations

Pandas have methods to compute operation on every row of every column. They are useful beacuse they come with support to handle the NaN value and will not prompt error. We also do not have to use for loop to iterate every row.

- **.add()** : add each row with specified value
- **.sub()** : subtract each row with specified value
- **.mul()** : do multiplication operation for each row

In [31]:

```
nba["Age"].add(5).head()
```

Out[31]:

```
0    30.0
1    30.0
2    32.0
3    27.0
4    34.0
Name: Age, dtype: float64
```

In [32]:

```
nba["Salary"].sub(100000).head(10)
```

Out[32]:

```
0     7630337.0
1     6696117.0
2           NaN
3     1048640.0
4     4900000.0
5    11900000.0
6     1070960.0
7     2065160.0
8     1724360.0
9     3331040.0
Name: Salary, dtype: float64
```

Change the weight to Kilogram, then assign the result to a new column

Hint : 1 lbs = 0.453592 kg

In [33]:

```
nba["Weight in Kilogram"] = nba["Weight"].mul(0.453592)
```

In [34]:

```
nba.head()
```

Out[34]:

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

value_counts() method

This method returns an object containing counts of unique values. The resulting object will be in descending order so that the first element is the most frequently-occurring element. By default, it will exclude NA values.

In [35]:

```
nba = pd.read_csv("data/nba.csv")
nba.head(3)
```

Out[35]:

	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

Information we can gain from value counts are

- **SG** or Shooting Guard is the most popular position among the players.
- **C** or Center is the least popular.
- **New Orleans Pelicans** hold the most players.

In [36]:

```
nba["Position"].value_counts()
```

Out[36]:

```
SG    102
PF    100
PG     92
SF     85
C      78
Name: Position, dtype: int64
```

In [37]:

```
nba["Team"].value_counts()
```

Out[37]:

```
New Orleans Pelicans      19
Memphis Grizzlies         18
Milwaukee Bucks           16
New York Knicks           16
Toronto Raptors           15
Utah Jazz                 15
Boston Celtics            15
Chicago Bulls             15
Sacramento Kings          15
San Antonio Spurs         15
Indiana Pacers            15
Portland Trail Blazers    15
Atlanta Hawks             15
Denver Nuggets            15
Phoenix Suns              15
Los Angeles Lakers        15
Golden State Warriors     15
Los Angeles Clippers      15
Brooklyn Nets             15
Miami Heat                15
Cleveland Cavaliers       15
Oklahoma City Thunder     15
Washington Wizards        15
Detroit Pistons           15
Philadelphia 76ers        15
Charlotte Hornets         15
Dallas Mavericks          15
Houston Rockets           15
Orlando Magic             14
Minnesota Timberwolves    14
Name: Team, dtype: int64
```

Drop Rows with Null Values

.dropna() Removes row, which contains null or NaN value. The return value is DataFrame object

- **how** : the default value is "any" which means if there is at least one Null value, the row will be removed. If how = "all “, only row with all its values are null will be removed
- **inplace** : store permanent changes

- **axis** : default value is 0. If we change to 1, it will remove column that contains any null value.
- **subset** : choose which column(s) contains null values. By default it will check all the columns.

In [38]:

```
nba = pd.read_csv("data/nba.csv")
nba.tail()
```

Out[38]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [39]:

```
nba.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 458 entries, 0 to 457
Data columns (total 9 columns):
Name        457 non-null object
Team        457 non-null object
Number      457 non-null float64
Position    457 non-null object
Age         457 non-null float64
Height      457 non-null object
Weight      457 non-null float64
College     373 non-null object
Salary      446 non-null float64
dtypes: float64(4), object(5)
memory usage: 32.3+ KB
```

In [40]:

```
nba.dropna(how="all", inplace=True)
```

As you can see here, the last row which contains all Null value is removed.

In [41]:

```
nba.tail()
```

Out[41]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
452	Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0

Remove all rows that contain at least one Null value. Then, check the summary of the new data using the .info() method.

In [42]:

```
nba.dropna(inplace=True)
```

In [43]:

```
nba.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 364 entries, 0 to 456
Data columns (total 9 columns):
Name        364 non-null object
Team        364 non-null object
Number      364 non-null float64
Position    364 non-null object
Age         364 non-null float64
Height      364 non-null object
Weight      364 non-null float64
College     364 non-null object
Salary      364 non-null float64
dtypes: float64(4), object(5)
memory usage: 28.4+ KB
```

Remove column that has null value

First, let's remove rows which contain all null values. Then, use dropna() method again with parameter **axis = 1**.

As a result, we can see that **Salary and College column is dropped** because these two columns have null values.

In [44]:

```
nba = pd.read_csv("data/nba.csv")
nba.dropna(how="all", inplace=True)
nba.head(3)
```

Out[44]:

	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

In [45]:

```
nba.dropna(axis=1).head()
```

Out[45]:

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

subset parameter

In [46]:

```
nba = pd.read_csv("data/nba.csv")
nba.dropna(how="all", inplace=True)
nba.head(3)
```

Out[46]:

	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

In [47]:

```
nba.dropna(subset=["College"], inplace=True)
nba.head()
```

Out[47]:

	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
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0

We can see the index values has changed so we should reset them as follows.

In [48]:

```
nba.reset_index()
```

Out[48]:

	index	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
...	...	...	...	...	...	...	...	...	...	...
368	449	Rodney Hood	Utah Jazz	5.0	SG	23.0	6-8	206.0	Duke	1348440.0
369	451	Chris Johnson	Utah Jazz	23.0	SF	26.0	6-6	206.0	Dayton	981348.0
370	452	Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0
371	453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
372	456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0

373 rows × 10 columns

Fill in the Null Values with .fillna() method

Fill NA/NaN values using the specified method.

It is better if we select a specific column and then apply fillna() method on it. If we apply fillna() method on the whole DataFrame, the result will not be good as not all columns have the same data type.

For instance, we would rather replace the NaN value in Salary column with the mean value instead of 0. 0 seems illogical as no player is willing to play for free.

In [49]:

```
nba = pd.read_csv("data/nba.csv")
nba.dropna(how="all", inplace=True)
mask = nba["Salary"].isna()
nba[mask].head()
```

Out[49]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
46	Elton Brand	Philadelphia 76ers	42.0	PF	37.0	6-9	254.0	Duke	NaN
171	Dahntay Jones	Cleveland Cavaliers	30.0	SG	35.0	6-6	225.0	Duke	NaN
264	Jordan Farmar	Memphis Grizzlies	4.0	PG	29.0	6-2	180.0	UCLA	NaN
269	Ray McCallum	Memphis Grizzlies	5.0	PG	24.0	6-3	190.0	Detroit	NaN

In [50]:

```
mean_Salary = nba["Salary"].mean()
mean_Salary
```

Out[50]:

```
4842684.105381166
```

In [51]:

```
nba["Salary"].fillna(mean_Salary, inplace=True)
```

In [52]:

```
nba["College"].fillna("No College", inplace=True)
```

In [53]:

```
nba[mask].head()
```

Out[53]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	4.842684e+06
46	Elton Brand	Philadelphia 76ers	42.0	PF	37.0	6-9	254.0	Duke	4.842684e+06
171	Dahntay Jones	Cleveland Cavaliers	30.0	SG	35.0	6-6	225.0	Duke	4.842684e+06
264	Jordan Farmar	Memphis Grizzlies	4.0	PG	29.0	6-2	180.0	UCLA	4.842684e+06
269	Ray McCallum	Memphis Grizzlies	5.0	PG	24.0	6-3	190.0	Detroit	4.842684e+06

The .astype() method

Change a column data type. This method requires the column to be filled and has no NaN value.

Jersey Number in Number column are supposed to be integer type. It is weird to see a jersey with a point. Let's change the data type to integer.

In [54]:

```
nba = pd.read_csv("data/nba.csv")
nba.dropna(inplace=True)
nba.reset_index(inplace=True)
```

In [55]:

```
nba.head()
```

Out[55]:

	index	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
3	6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
4	7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0

In [56]:

```
nba.dtypes
nba.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 364 entries, 0 to 363
Data columns (total 10 columns):
index       364 non-null int64
Name        364 non-null object
Team        364 non-null object
Number      364 non-null float64
Position    364 non-null object
Age         364 non-null float64
Height      364 non-null object
Weight      364 non-null float64
College     364 non-null object
Salary      364 non-null float64
dtypes: float64(4), int64(1), object(5)
memory usage: 28.6+ KB
```

astype() method does not have inplace parameter. Hence, we need to reassign the nba variable.

In [57]:

```
nba["Number"] = nba["Number"].astype("int")
```

As you can see below, the data type for the Number column has changed. Besides that, the memory usage has also decreased. This is because floating point requires more storage than integer.

In [58]:

```
nba.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 364 entries, 0 to 363
Data columns (total 10 columns):
index       364 non-null int64
Name        364 non-null object
Team        364 non-null object
Number      364 non-null int32
Position    364 non-null object
Age         364 non-null float64
Height      364 non-null object
Weight      364 non-null float64
College     364 non-null object
Salary      364 non-null float64
dtypes: float64(3), int32(1), int64(1), object(5)
memory usage: 27.1+ KB
```

astype() method can also change column to category type. Using this type can save a few bytes of memory usage.

In [59]:

```
nba["Team"] = nba["Team"].astype("category")
```

In [60]:

```
nba.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 364 entries, 0 to 363
Data columns (total 10 columns):
index       364 non-null int64
Name        364 non-null object
Team        364 non-null category
Number      364 non-null int32
Position    364 non-null object
Age         364 non-null float64
Height      364 non-null object
Weight      364 non-null float64
College     364 non-null object
Salary      364 non-null float64
dtypes: category(1), float64(3), int32(1), int64(1), object(4)
memory usage: 26.1+ KB
```

Sort a DataFrame with the .sort_values() Method

Sort all columns based on a specific column.

- **by** : using column name as references.
- **ascending** : default value is True. If set to False, it will sort in descending order.
- **na_position** : default value is "last" meaning the NaN value will be placed in the last row. We can also change it to "first".

In [61]:

```
nba = pd.read_csv("data/nba.csv")
nba.head(3)
```

Out[61]:

	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

In [62]:

```
nba.sort_values(by = "Name").head()
```

Out[62]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
152	Aaron Brooks	Chicago Bulls	0.0	PG	31.0	6-0	161.0	Oregon	2250000.0
356	Aaron Gordon	Orlando Magic	0.0	PF	20.0	6-9	220.0	Arizona	4171680.0
328	Aaron Harrison	Charlotte Hornets	9.0	SG	21.0	6-6	210.0	Kentucky	525093.0
404	Adreian Payne	Minnesota Timberwolves	33.0	PF	25.0	6-10	237.0	Michigan State	1938840.0
312	Al Horford	Atlanta Hawks	15.0	C	30.0	6-10	245.0	Florida	12000000.0

Now we know, Player Kobe Bryant has the highest salary.

In [63]:

```
nba.sort_values(by = "Salary", ascending=False).head()
```

Out[63]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
109	Kobe Bryant	Los Angeles Lakers	24.0	SF	37.0	6-6	212.0	NaN	25000000.0
169	LeBron James	Cleveland Cavaliers	23.0	SF	31.0	6-8	250.0	NaN	22970500.0
33	Carmelo Anthony	New York Knicks	7.0	SF	32.0	6-8	240.0	Syracuse	22875000.0
251	Dwight Howard	Houston Rockets	12.0	C	30.0	6-11	265.0	NaN	22359364.0
339	Chris Bosh	Miami Heat	1.0	PF	32.0	6-11	235.0	Georgia Tech	22192730.0

Sort_values() with two or more column references

This can be achieved by listing all the columns' names we want inside the **by parameter**.

We can also specify how each of the columns should be sorted (ascending or descending order).

In this example, we will sort by Team name in ascending order and Salary in descending order. As a result, we can see who has the highest Salary in the first Team.

In [64]:

```
nba.sort_values(by = ["Team","Salary"], ascending=[True, False])
```

Out[64]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
315	Paul Millsap	Atlanta Hawks	4.0	PF	31.0	6-8	246.0	Louisiana Tech	18671659.0
312	Al Horford	Atlanta Hawks	15.0	C	30.0	6-10	245.0	Florida	12000000.0
321	Tiago Splitter	Atlanta Hawks	11.0	C	31.0	6-11	245.0	NaN	9756250.0
323	Jeff Teague	Atlanta Hawks	0.0	PG	27.0	6-2	186.0	Wake Forest	8000000.0
314	Kyle Korver	Atlanta Hawks	26.0	SG	35.0	6-7	212.0	Creighton	5746479.0
...	...	...	...	...	...	...	...	...	...
380	Garrett Temple	Washington Wizards	17.0	SG	30.0	6-6	195.0	LSU	1100602.0
371	Jarell Eddie	Washington Wizards	8.0	SG	24.0	6-7	218.0	Virginia Tech	561716.0
374	JJ Hickson	Washington Wizards	21.0	C	27.0	6-9	242.0	North Carolina State	273038.0
381	Marcus Thornton	Washington Wizards	15.0	SF	29.0	6-4	205.0	LSU	200600.0
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

458 rows × 9 columns

.sort_index() Method

After we use dropna() or sort_values() method, the index will not be in order. To revert the changes back to the orginal, we can use sort_index() method.

In [65]:

```
nba = pd.read_csv("data/nba.csv")
nba.sort_values(by = ["Team","Salary"], ascending=[True, False], inplace=True)
nba.head(3)
```

Out[65]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
315	Paul Millsap	Atlanta Hawks	4.0	PF	31.0	6-8	246.0	Louisiana Tech	18671659.0
312	Al Horford	Atlanta Hawks	15.0	C	30.0	6-10	245.0	Florida	12000000.0
321	Tiago Splitter	Atlanta Hawks	11.0	C	31.0	6-11	245.0	NaN	9756250.0

In [66]:

```
nba.sort_index().head(10)
```

Out[66]:

	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
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0

.rank() method

Compute data ranks (1 through n). Equal values are assigned a rank that is the average of the ranks of those values.

For example, we can create a ranking of player based on their salary. Method chaining is used to compute them in one line of code.

In [67]:

```
nba = pd.read_csv("data/nba.csv")
nba.fillna(nba["Salary"].mean(), inplace=True)
nba.head(3)
```

Out[67]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7.730337e+06
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6.796117e+06
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	4.842684e+06

In [68]:

```
nba["Salary Rank"] = nba["Salary"].rank(ascending=False).astype("int")
```

In [69]:

```
nba.head()
```

Out[69]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary	Salary Rank
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7.730337e+06	97
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6.796117e+06	110
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	4.842684e+06	156
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1.148640e+06	334
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	4.84268e+06	5.000000e+06	147

Here we can see the players' rank based on their salary.

In [70]:

```
nba.sort_values("Salary Rank")
```

Out[70]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salar
109	Kobe Bryant	Los Angeles Lakers	24.0	SF	37.0	6-6	212.0	4.84268e+06	25000000.
169	LeBron James	Cleveland Cavaliers	23.0	SF	31.0	6-8	250.0	4.84268e+06	22970500.
33	Carmelo Anthony	New York Knicks	7.0	SF	32.0	6-8	240.0	Syracuse	22875000.
251	Dwight Howard	Houston Rockets	12.0	C	30.0	6-11	265.0	4.84268e+06	22359364.
339	Chris Bosh	Miami Heat	1.0	PF	32.0	6-11	235.0	Georgia Tech	22192730.
...	...	...	...	...	...	...	...	...	.
175	Jordan McRae	Cleveland Cavaliers	12.0	SG	25.0	6-5	179.0	Tennessee	111196.
135	Alan Williams	Phoenix Suns	15.0	C	23.0	6-8	260.0	UC Santa Barbara	83397.
130	Phil Pressey	Phoenix Suns	25.0	PG	25.0	5-11	175.0	Missouri	55722.
291	Orlando Johnson	New Orleans Pelicans	0.0	SG	27.0	6-5	220.0	UC Santa Barbara	55722.
32	Thanasis Antetokounmpo	New York Knicks	43.0	SF	23.0	6-7	205.0	4.84268e+06	30888.

458 rows × 10 columns

DataFrame Part 2 : Filtering and Cleaning

In [71]:

```
import pandas as pd
```

In [72]:

```
employees = pd.read_csv("data/employees.csv")
employees.head()
```

Out[72]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

In [73]:

```
employees.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
First Name           933 non-null object
Gender               855 non-null object
Start Date           1000 non-null object
Last Login Time      1000 non-null object
Salary               1000 non-null int64
Bonus %              1000 non-null float64
Senior Management    933 non-null object
Team                 957 non-null object
dtypes: float64(1), int64(1), object(6)
memory usage: 62.6+ KB
```

Problems in the dataset

- First Name, Gender, Senior Management and Team have missing values/NaN
- Start Date and Last Login Time columns are not in datetime type
- Senior Management column consists of data which are either True or False. But the column's data type is not Boolean.
- Gender and Team columns' data type is not category.

1. Change Start Date and Last Login Time columns into the datetime data type

Pandas has **to_datetime()** method to change data type of a column into datetime type.

We simply have to reassign the result of this method to the column name so we do not have to create a new column and delete the old one. Notice the change of dtype when we call info() method.

In [74]:

```
employees["Start Date"].head()
```

Out[74]:

```
0     8/6/1993
1    3/31/1996
2    4/23/1993
3     3/4/2005
4    1/24/1998
Name: Start Date, dtype: object
```

In [75]:

```
employees["Start Date"] = pd.to_datetime(employees["Start Date"])
employees["Last Login Time"] = pd.to_datetime(employees["Last Login Time"])
```

In [76]:

```
employees.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
First Name           933 non-null object
Gender               855 non-null object
Start Date           1000 non-null datetime64[ns]
Last Login Time      1000 non-null datetime64[ns]
Salary               1000 non-null int64
Bonus %              1000 non-null float64
Senior Management    933 non-null object
Team                 957 non-null object
dtypes: datetime64[ns](2), float64(1), int64(1), object(4)
memory usage: 62.6+ KB
```

In [77]:

```
employees.head()
```

Out[77]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	1993-08-06	2019-10-31 12:42:00	97308	6.945	True	Marketing
1	Thomas	Male	1996-03-31	2019-10-31 06:53:00	61933	4.170	True	NaN
2	Maria	Female	1993-04-23	2019-10-31 11:17:00	130590	11.858	False	Finance
3	Jerry	Male	2005-03-04	2019-10-31 13:00:00	138705	9.340	True	Finance
4	Larry	Male	1998-01-24	2019-10-31 16:47:00	101004	1.389	True	Client Services

The **parse_dates()** parameter in read_csv() can automatically change columns into datetime type.

In [78]:

```
employees = pd.read_csv("data/employees.csv", parse_dates = ["Start Date" , "Last Login Tim
employees.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
First Name           933 non-null object
Gender               855 non-null object
Start Date           1000 non-null datetime64[ns]
Last Login Time      1000 non-null datetime64[ns]
Salary               1000 non-null int64
Bonus %              1000 non-null float64
Senior Management    933 non-null object
Team                 957 non-null object
dtypes: datetime64[ns](2), float64(1), int64(1), object(4)
memory usage: 62.6+ KB
```

2.Change Gender and Senior Management types

We have learned about **astype()** method which is for changing the data type of columns.

Gender only has 2 unique values. Hence, it is better to change the type to category.

In [79]:

```
employees["Gender"] = employees["Gender"].astype("category")
```

In [80]:

```
employees["Senior Management"] = employees["Senior Management"].astype("bool")
```

In [81]:

```
employees.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
First Name           933 non-null object
Gender               855 non-null category
Start Date           1000 non-null datetime64[ns]
Last Login Time      1000 non-null datetime64[ns]
Salary               1000 non-null int64
Bonus %              1000 non-null float64
Senior Management    1000 non-null bool
Team                 957 non-null object
dtypes: bool(1), category(1), datetime64[ns](2), float64(1), int64(1), objec
t(2)
memory usage: 49.0+ KB
```

Notice the memory usage changes from 62.6KB to 49KB. Imagine if we have 1 Million rows. These changes do affect memory performance.

Filter a DataFrame based on condition

In [82]:

```
df = pd.read_csv("data/employees.csv", parse_dates = ["Start Date" , "Last Login Time"])
df["Gender"] = df["Gender"].astype("category")
df["Senior Management"] = df["Senior Management"].astype("bool")
df.head()
```

Out[82]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	1993-08-06	2019-10-31 12:42:00	97308	6.945	True	Marketing
1	Thomas	Male	1996-03-31	2019-10-31 06:53:00	61933	4.170	True	NaN
2	Maria	Female	1993-04-23	2019-10-31 11:17:00	130590	11.858	False	Finance
3	Jerry	Male	2005-03-04	2019-10-31 13:00:00	138705	9.340	True	Finance
4	Larry	Male	1998-01-24	2019-10-31 16:47:00	101004	1.389	True	Client Services

Let's say we want to filter the data of only male employees. The first step we need to do is create a list of Boolean values that corresponds to each row of Gender column.

Next, pass this list to the DataFrame. Once we do that, we will get a filtered DataFrame.

In [83]:

```
mask = df["Gender"] == "Male"
mask
```

Out[83]:

```
0       True
1       True
2      False
3       True
4       True
       ...
995    False
996     True
997     True
998     True
999     True
Name: Gender, Length: 1000, dtype: bool
```

Here is the list of Male Employees.

In [84]:

```
df[mask]
```

Out[84]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	1993-08-06	2019-10-31 12:42:00	97308	6.945	True	Marketing
1	Thomas	Male	1996-03-31	2019-10-31 06:53:00	61933	4.170	True	NaN
3	Jerry	Male	2005-03-04	2019-10-31 13:00:00	138705	9.340	True	Finance
4	Larry	Male	1998-01-24	2019-10-31 16:47:00	101004	1.389	True	Client Services
5	Dennis	Male	1987-04-18	2019-10-31 01:35:00	115163	10.125	False	Legal
...	...	...	...	...	...	...	...	...
994	George	Male	2013-06-21	2019-10-31 17:47:00	98874	4.479	True	Marketing
996	Phillip	Male	1984-01-31	2019-10-31 06:30:00	42392	19.675	False	Finance
997	Russell	Male	2013-05-20	2019-10-31 12:39:00	96914	1.421	False	Product
998	Larry	Male	2013-04-20	2019-10-31 16:45:00	60500	11.985	False	Business Development
999	Albert	Male	2012-05-15	2019-10-31 18:24:00	129949	10.169	True	Sales

424 rows × 8 columns

In this second example, we want to extract Employees that are in Finance Team. As the first step, we do the same thing which is creating a list of Boolean values by checking every row with a condition.

In [85]:

```
extract = df["Team"] == "Finance"
extract
```

Out[85]:

```
0      False
1      False
2       True
3       True
4      False
       ...
995    False
996     True
997    False
998    False
999    False
Name: Team, Length: 1000, dtype: bool
```

Then, we pass the list into Pandas DataFrame.

In [86]:

```
df[extract]
```

Out[86]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
2	Maria	Female	1993-04-23	2019-10-31 11:17:00	130590	11.858	False	Finance
3	Jerry	Male	2005-03-04	2019-10-31 13:00:00	138705	9.340	True	Finance
7	NaN	Female	2015-07-20	2019-10-31 10:43:00	45906	11.598	True	Finance
14	Kimberly	Female	1999-01-14	2019-10-31 07:13:00	41426	14.543	True	Finance
46	Bruce	Male	2009-11-28	2019-10-31 22:47:00	114796	6.796	False	Finance
...	...	...	...	...	...	...	...	...
907	Elizabeth	Female	1998-07-27	2019-10-31 11:12:00	137144	10.081	False	Finance
954	Joe	Male	1980-01-19	2019-10-31 16:06:00	119667	1.148	True	Finance
987	Gloria	Female	2014-12-08	2019-10-31 05:08:00	136709	10.331	True	Finance
992	Anthony	Male	2011-10-16	2019-10-31 08:35:00	112769	11.625	True	Finance
996	Phillip	Male	1984-01-31	2019-10-31 06:30:00	42392	19.675	False	Finance

102 rows × 8 columns

We can also do it for the date column since our Start Time is datetime data type. Here, we are extracting

employees that have been working since 1990s.

In [87]:

```
extract = df["Start Date"] <= "1990-01-01"
extract
```

Out[87]:

```
0      False
1      False
2      False
3      False
4      False
       ...
995    False
996     True
997    False
998    False
999    False
Name: Start Date, Length: 1000, dtype: bool
```

In [88]:

```
df[extract]
```

Out[88]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
5	Dennis	Male	1987-04-18	2019-10-31 01:35:00	115163	10.125	False	Legal
6	Ruby	Female	1987-08-17	2019-10-31 16:20:00	65476	10.012	True	Product
10	Louise	Female	1980-08-12	2019-10-31 09:01:00	63241	15.132	True	NaN
12	Brandon	Male	1980-12-01	2019-10-31 01:08:00	112807	17.492	True	Human Resources
17	Shawn	Male	1986-12-07	2019-10-31 19:45:00	111737	6.414	False	Product
...	...	...	...	...	...	...	...	...
983	John	Male	1982-12-23	2019-10-31 22:35:00	146907	11.738	False	Engineering
985	Stephen	NaN	1983-07-10	2019-10-31 20:10:00	85668	1.909	False	Legal
986	Donna	Female	1982-11-26	2019-10-31 07:04:00	82871	17.999	False	Marketing
990	Robin	Female	1987-07-24	2019-10-31 13:35:00	100765	10.982	True	Client Services
996	Phillip	Male	1984-01-31	2019-10-31 06:30:00	42392	19.675	False	Finance

240 rows × 8 columns

Since the Senior Management column is already a boolean type, we can just pass the column to dataFrame.

In [89]:

```
df[df["Senior Management"]].head()
```

Out[89]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	1993-08-06	2019-10-31 12:42:00	97308	6.945	True	Marketing
1	Thomas	Male	1996-03-31	2019-10-31 06:53:00	61933	4.170	True	NaN
3	Jerry	Male	2005-03-04	2019-10-31 13:00:00	138705	9.340	True	Finance
4	Larry	Male	1998-01-24	2019-10-31 16:47:00	101004	1.389	True	Client Services
6	Ruby	Female	1987-08-17	2019-10-31 16:20:00	65476	10.012	True	Product

Filtering DataFrame with more than one condition

If we have two or more conditions when filtering our DataFrame, we need to combine the Boolean lists with **&** symbol for **AND** operator and **|** symbol for **OR** operator.

In [90]:

```
df = pd.read_csv("data/employees.csv", parse_dates = ["Start Date" , "Last Login Time"])
df["Gender"] = df["Gender"].astype("category")
df["Senior Management"] = df["Senior Management"].astype("bool")
df.head(3)
```

Out[90]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	1993-08-06	2019-10-31 12:42:00	97308	6.945	True	Marketing
1	Thomas	Male	1996-03-31	2019-10-31 06:53:00	61933	4.170	True	NaN
2	Maria	Female	1993-04-23	2019-10-31 11:17:00	130590	11.858	False	Finance

Create two variables for each condition. Then, combine these two variables with & symbol and pass it to the DataFrame.

In [91]:

```
mask1 = df["Gender"] == "Male"
mask2 = df["Team"] == "Finance"
df[mask1 & mask2].head()
```

Out[91]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
3	Jerry	Male	2005-03-04	2019-10-31 13:00:00	138705	9.340	True	Finance
46	Bruce	Male	2009-11-28	2019-10-31 22:47:00	114796	6.796	False	Finance
56	Carl	Male	2006-05-03	2019-10-31 17:55:00	130276	16.084	True	Finance
68	Jose	Male	2004-10-30	2019-10-31 13:39:00	84834	14.330	True	Finance
83	Shawn	Male	2005-09-23	2019-10-31 02:55:00	148115	6.539	True	Finance

We also can do it in a single line, though it is harder to read. The example below is 3 conditions filtering.

In [92]:

```
df[(df["Gender"] == "Male") &  (df["Team"] == "Finance") & (df["Start Date"] <= "1990-01-01
```

Out[92]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
200	Gary	Male	1987-08-12	2019-10-31 00:04:00	89661	8.525	False	Finance
241	Walter	Male	1983-07-06	2019-10-31 23:54:00	127813	5.961	False	Finance
286	Todd	Male	1984-02-02	2019-10-31 10:13:00	69989	10.985	True	Finance
304	Jeremy	Male	1981-08-12	2019-10-31 00:05:00	46930	18.702	True	Finance
348	Philip	Male	1989-08-02	2019-10-31 11:21:00	129968	19.897	False	Finance

Below is an example of using | operator in which we are extracting employee with a Salary more than 149,000 **OR** their First Name is Gary.

In [93]:

```
mask1  = df["Salary"] > 149000
mask2 = df["First Name"] == "Gary"

df[mask1 | mask2]
```

Out[93]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
13	Gary	Male	2008-01-27	2019-10-31 23:40:00	109831	5.831	False	Sales
160	Kathy	Female	2000-03-18	2019-10-31 19:26:00	149563	16.991	True	Finance
186	NaN	Female	2005-02-23	2019-10-31 21:50:00	149654	1.825	True	Sales
200	Gary	Male	1987-08-12	2019-10-31 00:04:00	89661	8.525	False	Finance
414	Gary	Male	2011-08-18	2019-10-31 08:12:00	49101	11.900	True	Business Development
429	Rose	Female	2015-05-28	2019-10-31 08:40:00	149903	5.630	False	Human Resources
644	Katherine	Female	1996-08-13	2019-10-31 00:21:00	149908	18.912	False	Finance
740	Russell	NaN	2009-05-09	2019-10-31 11:59:00	149456	3.533	False	Marketing
793	Andrea	Female	1999-07-22	2019-10-31 09:25:00	149105	13.707	True	Distribution
828	Cynthia	Female	2006-07-12	2019-10-31 08:55:00	149684	7.864	False	Product

The isin() method

Checks whether each element in the DataFrame contains the values we want.

For example, we want to extract employees who are either in team Finance, Sales or Product. One way to do it is as follows.

In [94]:

```
mask1 = df["Team"] == "Finance"
mask2 = df["Team"] == "Sales"
mask3 = df["Team"] == "Product"

df[mask1 | mask2 | mask3]
```

Out[94]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
2	Maria	Female	1993-04-23	2019-10-31 11:17:00	130590	11.858	False	Finance
3	Jerry	Male	2005-03-04	2019-10-31 13:00:00	138705	9.340	True	Finance
6	Ruby	Female	1987-08-17	2019-10-31 16:20:00	65476	10.012	True	Product
7	NaN	Female	2015-07-20	2019-10-31 10:43:00	45906	11.598	True	Finance
13	Gary	Male	2008-01-27	2019-10-31 23:40:00	109831	5.831	False	Sales
...	...	...	...	...	...	...	...	...
987	Gloria	Female	2014-12-08	2019-10-31 05:08:00	136709	10.331	True	Finance
992	Anthony	Male	2011-10-16	2019-10-31 08:35:00	112769	11.625	True	Finance
996	Phillip	Male	1984-01-31	2019-10-31 06:30:00	42392	19.675	False	Finance
997	Russell	Male	2013-05-20	2019-10-31 12:39:00	96914	1.421	False	Product
999	Albert	Male	2012-05-15	2019-10-31 18:24:00	129949	10.169	True	Sales

291 rows × 8 columns

However, the above way is not convenient if we have many Teams to filter. Hence, this is where the **isin() method** comes in handy.

In [95]:

```
mask_isin = df["Team"].isin(["Finance", "Product" , "Sales"])
mask_isin
```

Out[95]:

```
0      False
1      False
2       True
3       True
4      False
       ...
995    False
996     True
997     True
998    False
999     True
Name: Team, Length: 1000, dtype: bool
```

In [96]:

```
df[mask_isin]
```

Out[96]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
2	Maria	Female	1993-04-23	2019-10-31 11:17:00	130590	11.858	False	Finance
3	Jerry	Male	2005-03-04	2019-10-31 13:00:00	138705	9.340	True	Finance
6	Ruby	Female	1987-08-17	2019-10-31 16:20:00	65476	10.012	True	Product
7	NaN	Female	2015-07-20	2019-10-31 10:43:00	45906	11.598	True	Finance
13	Gary	Male	2008-01-27	2019-10-31 23:40:00	109831	5.831	False	Sales
...	...	...	...	...	...	...	...	...
987	Gloria	Female	2014-12-08	2019-10-31 05:08:00	136709	10.331	True	Finance
992	Anthony	Male	2011-10-16	2019-10-31 08:35:00	112769	11.625	True	Finance
996	Phillip	Male	1984-01-31	2019-10-31 06:30:00	42392	19.675	False	Finance
997	Russell	Male	2013-05-20	2019-10-31 12:39:00	96914	1.421	False	Product
999	Albert	Male	2012-05-15	2019-10-31 18:24:00	129949	10.169	True	Sales

291 rows × 8 columns

isnull() and notnull() methods

- **isnull()** : checks each row in the Series if the element is null/ NaN. Returns a list of Boolean values, True or False
- **notnull()** : opposite of isnull(). Checks each row if the element exists.

In [97]:

```
df = pd.read_csv("data/employees.csv", parse_dates = ["Start Date" , "Last Login Time"])
df["Gender"] = df["Gender"].astype("category")
df["Senior Management"] = df["Senior Management"].astype("bool")
df.head(3)
```

Out[97]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	1993-08-06	2019-10-31 12:42:00	97308	6.945	True	Marketing
1	Thomas	Male	1996-03-31	2019-10-31 06:53:00	61933	4.170	True	NaN
2	Maria	Female	1993-04-23	2019-10-31 11:17:00	130590	11.858	False	Finance

Filtering rows that have missing Gender values.

In [98]:

```
df[df["Gender"].isnull()]
```

Out[98]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
20	Lois	NaN	1995-04-22	2019-10-31 19:18:00	64714	4.934	True	Legal
22	Joshua	NaN	2012-03-08	2019-10-31 01:58:00	90816	18.816	True	Client Services
27	Scott	NaN	1991-07-11	2019-10-31 18:58:00	122367	5.218	False	Legal
31	Joyce	NaN	2005-02-20	2019-10-31 14:40:00	88657	12.752	False	Product
41	Christine	NaN	2015-06-28	2019-10-31 01:08:00	66582	11.308	True	Business Development
...	...	...	...	...	...	...	...	...
961	Antonio	NaN	1989-06-18	2019-10-31 21:37:00	103050	3.050	False	Legal
972	Victor	NaN	2006-07-28	2019-10-31 14:49:00	76381	11.159	True	Sales
985	Stephen	NaN	1983-07-10	2019-10-31 20:10:00	85668	1.909	False	Legal
989	Justin	NaN	1991-02-10	2019-10-31 16:58:00	38344	3.794	False	Legal
995	Henry	NaN	2014-11-23	2019-10-31 06:09:00	132483	16.655	False	Distribution

145 rows × 8 columns

On the other hand, we can get dataFrame that does not have missing values on Gender using the **notnull() method.** This method is used to detect non-missing values for an array-like object.

In [99]:

```
df[df["Gender"].notnull()]
```

Out[99]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	1993-08-06	2019-10-31 12:42:00	97308	6.945	True	Marketing
1	Thomas	Male	1996-03-31	2019-10-31 06:53:00	61933	4.170	True	NaN
2	Maria	Female	1993-04-23	2019-10-31 11:17:00	130590	11.858	False	Finance
3	Jerry	Male	2005-03-04	2019-10-31 13:00:00	138705	9.340	True	Finance
4	Larry	Male	1998-01-24	2019-10-31 16:47:00	101004	1.389	True	Client Services
...	...	...	...	...	...	...	...	...
994	George	Male	2013-06-21	2019-10-31 17:47:00	98874	4.479	True	Marketing
996	Phillip	Male	1984-01-31	2019-10-31 06:30:00	42392	19.675	False	Finance
997	Russell	Male	2013-05-20	2019-10-31 12:39:00	96914	1.421	False	Product
998	Larry	Male	2013-04-20	2019-10-31 16:45:00	60500	11.985	False	Business Development
999	Albert	Male	2012-05-15	2019-10-31 18:24:00	129949	10.169	True	Sales

855 rows × 8 columns

.between() method

This function returns a Boolean vector containing True whenever the corresponding Series element is between the boundary values left and right. NA values are treated as False.

For instance, we want to filter Salary between 50,000 to 80,000. The first variable is the left boundary and the second variable is the right boundary.

In [100]:

```
df["Salary"].between(50000,80000)
```

Out[100]:

```
0      False
1       True
2      False
3      False
4      False
       ...
995    False
996    False
997    False
998     True
999    False
Name: Salary, Length: 1000, dtype: bool
```

In [101]:

```
df[df["Salary"].between(50000,80000)]
```

Out[101]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
1	Thomas	Male	1996-03-31	2019-10-31 06:53:00	61933	4.170	True	NaN
6	Ruby	Female	1987-08-17	2019-10-31 16:20:00	65476	10.012	True	Product
10	Louise	Female	1980-08-12	2019-10-31 09:01:00	63241	15.132	True	NaN
15	Lillian	Female	2016-06-05	2019-10-31 06:09:00	59414	1.256	False	Product
20	Lois	NaN	1995-04-22	2019-10-31 19:18:00	64714	4.934	True	Legal
...	...	...	...	...	...	...	...	...
972	Victor	NaN	2006-07-28	2019-10-31 14:49:00	76381	11.159	True	Sales
974	Harry	Male	2011-08-30	2019-10-31 18:31:00	67656	16.455	True	Client Services
978	Sean	Male	1983-01-17	2019-10-31 14:23:00	66146	11.178	False	Human Resources
993	Tina	Female	1997-05-15	2019-10-31 15:53:00	56450	19.040	True	Engineering
998	Larry	Male	2013-04-20	2019-10-31 16:45:00	60500	11.985	False	Business Development

256 rows × 8 columns

This also works on datetime

Let's filter employees who started working in the year 1990.

In [102]:

```
df[df["Start Date"].between("1990-01-01", "1991-01-01")]
```

Out[102]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
52	Todd	Male	1990-02-18	2019-10-31 02:41:00	49339	1.695	True	Human Resources
64	Kathleen	NaN	1990-04-11	2019-10-31 18:46:00	77834	18.771	False	Business Development
139	NaN	Female	1990-10-03	2019-10-31 01:08:00	132373	10.527	True	NaN
141	Adam	Male	1990-12-24	2019-10-31 20:57:00	110194	14.727	True	Product
163	Terry	Male	1990-09-03	2019-10-31 21:15:00	52226	19.135	False	Client Services
198	Maria	Female	1990-12-27	2019-10-31 21:57:00	36067	9.640	True	Product
242	Robert	Male	1990-10-27	2019-10-31 19:15:00	38041	18.428	True	Engineering
251	Sharon	NaN	1990-03-01	2019-10-31 03:46:00	83658	6.513	False	Business Development
310	Harold	Male	1990-02-20	2019-10-31 23:00:00	66775	2.158	True	Legal
323	Linda	Female	1990-12-16	2019-10-31 02:20:00	115658	3.041	True	Sales
356	Judy	Female	1990-02-01	2019-10-31 15:32:00	38092	5.668	False	Distribution
404	Sarah	NaN	1990-07-20	2019-10-31 22:49:00	109980	8.860	False	Sales
415	Shirley	NaN	1990-11-16	2019-10-31 18:01:00	67811	12.699	False	Finance
440	Aaron	Male	1990-07-22	2019-10-31 14:53:00	52119	11.343	True	Client Services
581	Ernest	Male	1990-01-28	2019-10-31 00:08:00	81919	15.118	False	Marketing
599	NaN	Female	1990-10-11	2019-10-31 22:57:00	98385	10.925	True	Human Resources
619	Beverly	Female	1990-10-22	2019-10-31 14:45:00	59070	19.064	True	Engineering
632	Rebecca	Female	1990-02-23	2019-10-31 15:21:00	134673	6.878	False	Engineering
697	NaN	Male	1990-05-23	2019-10-31 02:09:00	136655	9.801	True	Distribution
805	Thomas	Male	1990-05-03	2019-10-31 09:29:00	111371	15.081	True	Engineering
846	Stephen	Male	1990-09-10	2019-10-31 22:42:00	129663	15.574	False	Human Resources
940	Andrew	Male	1990-09-28	2019-10-31 09:38:00	137386	8.611	True	Distribution

.duplicated() method

Returns boolean Series denoting duplicate rows, optionally only considering certain columns.

parameter **keep** :

- *first* : Mark duplicates as True except for the first occurrence.
- *last* : Mark duplicates as True except for the last occurrence.
- *False* : Mark all duplicates as True.

In [103]:

```
df = pd.read_csv("data/employees.csv", parse_dates = ["Start Date" , "Last Login Time"])
df["Gender"] = df["Gender"].astype("category")
df["Senior Management"] = df["Senior Management"].astype("bool")
df.sort_values("First Name", inplace=True)
df.head(10)
```

Out[103]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
101	Aaron	Male	2012-02-17	2019-10-31 10:20:00	61602	11.849	True	Marketing
327	Aaron	Male	1994-01-29	2019-10-31 18:48:00	58755	5.097	True	Marketing
440	Aaron	Male	1990-07-22	2019-10-31 14:53:00	52119	11.343	True	Client Services
937	Aaron	NaN	1986-01-22	2019-10-31 19:39:00	63126	18.424	False	Client Services
137	Adam	Male	2011-05-21	2019-10-31 01:45:00	95327	15.120	False	Distribution
141	Adam	Male	1990-12-24	2019-10-31 20:57:00	110194	14.727	True	Product
302	Adam	Male	2007-07-05	2019-10-31 11:59:00	71276	5.027	True	Human Resources
538	Adam	Male	2010-10-08	2019-10-31 21:53:00	45181	3.491	False	Human Resources
300	Alan	Male	1988-06-26	2019-10-31 03:54:00	111786	3.592	True	Engineering
53	Alan	NaN	2014-03-03	2019-10-31 13:28:00	40341	17.578	True	Finance

Notice below that the first row is returned False. Meanwhile, the 2nd until 4th-row return True indicating that they are duplicates. By default, it will identifies the first value as Non Duplicate. However, we can change the value to **last** if we want.

In [104]:

```
df["First Name"].duplicated()
```

Out[104]:

```
101    False
327     True
440     True
937     True
137    False
       ...
902     True
925     True
946     True
947     True
951     True
Name: First Name, Length: 1000, dtype: bool
```

In [105]:

```
df[df["First Name"].duplicated()].head()
```

Out[105]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
327	Aaron	Male	1994-01-29	2019-10-31 18:48:00	58755	5.097	True	Marketing
440	Aaron	Male	1990-07-22	2019-10-31 14:53:00	52119	11.343	True	Client Services
937	Aaron	NaN	1986-01-22	2019-10-31 19:39:00	63126	18.424	False	Client Services
141	Adam	Male	1990-12-24	2019-10-31 20:57:00	110194	14.727	True	Product
302	Adam	Male	2007-07-05	2019-10-31 11:59:00	71276	5.027	True	Human Resources

In [106]:

```
df[df["First Name"].duplicated(keep="last")].head()
```

Out[106]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
101	Aaron	Male	2012-02-17	2019-10-31 10:20:00	61602	11.849	True	Marketing
327	Aaron	Male	1994-01-29	2019-10-31 18:48:00	58755	5.097	True	Marketing
440	Aaron	Male	1990-07-22	2019-10-31 14:53:00	52119	11.343	True	Client Services
137	Adam	Male	2011-05-21	2019-10-31 01:45:00	95327	15.120	False	Distribution
141	Adam	Male	1990-12-24	2019-10-31 20:57:00	110194	14.727	True	Product

The difference between keep = "first" (default value) and keep = "last" is the rows they choose to keep.

Let's see how we can show unique values by ignoring the duplicated values.

In [107]:

```
df[~df["First Name"].duplicated(keep=False)]
```

Out[107]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
8	Angela	Female	2005-11-22	2019-10-31 06:29:00	95570	18.523	True	Engineering
688	Brian	Male	2007-04-07	2019-10-31 22:47:00	93901	17.821	True	Legal
190	Carol	Female	1996-03-19	2019-10-31 03:39:00	57783	9.129	False	Finance
887	David	Male	2009-12-05	2019-10-31 08:48:00	92242	15.407	False	Legal
5	Dennis	Male	1987-04-18	2019-10-31 01:35:00	115163	10.125	False	Legal
495	Eugene	Male	1984-05-24	2019-10-31 10:54:00	81077	2.117	False	Sales
33	Jean	Female	1993-12-18	2019-10-31 09:07:00	119082	16.180	False	Business Development
832	Keith	Male	2003-02-12	2019-10-31 15:02:00	120672	19.467	False	Legal
291	Tammy	Female	1984-11-11	2019-10-31 10:30:00	132839	17.463	True	Client Services

.drop_duplicated()

Returns DataFrame with duplicate rows removed, optionally only considering certain columns. Indexes, including time indexes, are ignored.

We have to be specific on the column that we are checking for the duplicates. Else, it will check the entire column to find a row that has the exact same value with the other row.

In [108]:

```
df = pd.read_csv("data/employees.csv", parse_dates = ["Start Date" , "Last Login Time"])
df["Gender"] = df["Gender"].astype("category")
df["Senior Management"] = df["Senior Management"].astype("bool")
df.sort_values("First Name", inplace=True)
df.head(10)
```

Out[108]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
101	Aaron	Male	2012-02-17	2019-10-31 10:20:00	61602	11.849	True	Marketing
327	Aaron	Male	1994-01-29	2019-10-31 18:48:00	58755	5.097	True	Marketing
440	Aaron	Male	1990-07-22	2019-10-31 14:53:00	52119	11.343	True	Client Services
937	Aaron	NaN	1986-01-22	2019-10-31 19:39:00	63126	18.424	False	Client Services
137	Adam	Male	2011-05-21	2019-10-31 01:45:00	95327	15.120	False	Distribution
141	Adam	Male	1990-12-24	2019-10-31 20:57:00	110194	14.727	True	Product
302	Adam	Male	2007-07-05	2019-10-31 11:59:00	71276	5.027	True	Human Resources
538	Adam	Male	2010-10-08	2019-10-31 21:53:00	45181	3.491	False	Human Resources
300	Alan	Male	1988-06-26	2019-10-31 03:54:00	111786	3.592	True	Engineering
53	Alan	NaN	2014-03-03	2019-10-31 13:28:00	40341	17.578	True	Finance

In [109]:

```
df.drop_duplicates(subset=["First Name"], inplace=True, keep=False)
df
```

Out[109]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
8	Angela	Female	2005-11-22	2019-10-31 06:29:00	95570	18.523	True	Engineering
688	Brian	Male	2007-04-07	2019-10-31 22:47:00	93901	17.821	True	Legal
190	Carol	Female	1996-03-19	2019-10-31 03:39:00	57783	9.129	False	Finance
887	David	Male	2009-12-05	2019-10-31 08:48:00	92242	15.407	False	Legal
5	Dennis	Male	1987-04-18	2019-10-31 01:35:00	115163	10.125	False	Legal
495	Eugene	Male	1984-05-24	2019-10-31 10:54:00	81077	2.117	False	Sales
33	Jean	Female	1993-12-18	2019-10-31 09:07:00	119082	16.180	False	Business Development
832	Keith	Male	2003-02-12	2019-10-31 15:02:00	120672	19.467	False	Legal
291	Tammy	Female	1984-11-11	2019-10-31 10:30:00	132839	17.463	True	Client Services

To check two columns for duplicate, we have to specify the two columns' names in the subset. If there are two elements with the same values, it will be dropped.

.unique() and .nunique() methods

unique() method returns unique values of Series object. Meanwhile, **nunique()** method returns the number of unique values.

In [110]:

```
df = pd.read_csv("data/employees.csv", parse_dates = ["Start Date" , "Last Login Time"])
df["Gender"] = df["Gender"].astype("category")
df["Senior Management"] = df["Senior Management"].astype("bool")
df.head(3)
```

Out[110]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	1993-08-06	2019-10-31 12:42:00	97308	6.945	True	Marketing
1	Thomas	Male	1996-03-31	2019-10-31 06:53:00	61933	4.170	True	NaN
2	Maria	Female	1993-04-23	2019-10-31 11:17:00	130590	11.858	False	Finance

unique() method counts NaN/Null as a unique value. nunique() method does NOT count NaN/null values.

In [111]:

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

Out[111]:

```
array(['Marketing', nan, 'Finance', 'Client Services', 'Legal', 'Product',
       'Engineering', 'Business Development', 'Human Resources', 'Sales',
       'Distribution'], dtype=object)
```

In [112]:

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

Out[112]:

```
10
```

In [113]:

```
df["Team"].nunique(dropna=False)
```

Out[113]:

```
11
```

DataFrame Part 3:

set_index() and reset_index()

The Index is automatically generated in numeric. However, we can specify if we want a certain column to be the index, such as Date.

In [114]:

```
import pandas as pd
```

In [115]:

```
bond = pd.read_csv("data/jamesbond.csv")
bond.head(3)
```

Out[115]:

	Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
0	Dr. No	1962	Sean Connery	Terence Young	448.8	7.0	0.6
1	From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6
2	Goldfinger	1964	Sean Connery	Guy Hamilton	820.4	18.6	3.2

In [116]:

```
bond.set_index("Film", inplace=True)
bond.head(3)
```

Out[116]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Dr. No	1962	Sean Connery	Terence Young	448.8	7.0	0.6
From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6
Goldfinger	1964	Sean Connery	Guy Hamilton	820.4	18.6	3.2

To reset the index back to numeric and create a column for the previous index, we can use the reset_index() method.

In [117]:

```
bond.reset_index(inplace=True)
bond.head(3)
```

Out[117]:

	Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
0	Dr. No	1962	Sean Connery	Terence Young	448.8	7.0	0.6
1	From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6
2	Goldfinger	1964	Sean Connery	Guy Hamilton	820.4	18.6	3.2

If we to swap to using index label, we have to reset the index first, and then use the set_index() method. If we use set_index() straight away, the index will be deleted and replaced.

In [118]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.head(3)
```

Out[118]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Dr. No	1962	Sean Connery	Terence Young	448.8	7.0	0.6
From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6
Goldfinger	1964	Sean Connery	Guy Hamilton	820.4	18.6	3.2

In [119]:

```
bond.reset_index()
bond.set_index("Year", inplace=True)
bond.head()
```

Out[119]:

Year	Actor	Director	Box Office	Budget	Bond Actor Salary
1962	Sean Connery	Terence Young	448.8	7.0	0.6
1963	Sean Connery	Terence Young	543.8	12.6	1.6
1964	Sean Connery	Guy Hamilton	820.4	18.6	3.2
1965	Sean Connery	Terence Young	848.1	41.9	4.7
1967	David Niven	Ken Hughes	315.0	85.0	NaN

Retrieve Row(s) by Index Position with iloc

Each DataFrame has Index Position Number.

In [120]:

```
bond = pd.read_csv("data/jamesbond.csv")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[120]:

	Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
0	Dr. No	1962	Sean Connery	Terence Young	448.8	7.0	0.6
1	From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6
2	Goldfinger	1964	Sean Connery	Guy Hamilton	820.4	18.6	3.2

Extract a Series from DataFrame by a specifying the Index Number.

In [121]:

```
bond.iloc[1]
```

Out[121]:

```
Film                 From Russia with Love
Year                                  1963
Actor                          Sean Connery
Director                     Terence Young
Box Office                           543.8
Budget                                12.6
Bond Actor Salary                      1.6
Name: 1, dtype: object
```

To convert the Series into DataFrame, we have to add an extra square bracket.

In [122]:

```
bond.iloc[[1]]
```

Out[122]:

	Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
1	From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6

In [123]:

```
bond.iloc[[5,10,15,20]]
```

Out[123]:

	Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
5	You Only Live Twice	1967	Sean Connery	Lewis Gilbert	514.2	59.9	4.4
10	The Spy Who Loved Me	1977	Roger Moore	Lewis Gilbert	533.0	45.1	NaN
15	A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
20	The World Is Not Enough	1999	Pierce Brosnan	Michael Apted	439.5	158.3	13.5

A range of rows can be extracted by separating the upper bound and lower bound with colon (:)

In [124]:

```
bond.iloc[3:5]
```

Out[124]:

	Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
3	Thunderball	1965	Sean Connery	Terence Young	848.1	41.9	4.7
4	Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

Retrieve row by Index Label using .loc[ ] method

.loc method enables us to extract a specific or range of rows.

In [125]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(6)
```

Out[125]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN
Diamonds Are Forever	1971	Sean Connery	Guy Hamilton	442.5	34.7	5.8
Die Another Day	2002	Pierce Brosnan	Lee Tamahori	465.4	154.2	17.9
Dr. No	1962	Sean Connery	Terence Young	448.8	7.0	0.6

We know that we can specify names as the index. If our index is string labelled, we have to write the index label inside the square bracket. It will return a Series.

In [126]:

```
bond.loc["Die Another Day"]
```

Out[126]:

```
Year                          2002
Actor                Pierce Brosnan
Director               Lee Tamahori
Box Office                    465.4
Budget                        154.2
Bond Actor Salary              17.9
Name: Die Another Day, dtype: object
```

If we want it to return a DataFrame, we just have to put **another square* *bracket.

In [127]:

```
bond.loc[["Die Another Day"]]
```

Out[127]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Die Another Day	2002	Pierce Brosnan	Lee Tamahori	465.4	154.2	17.9

Next, let's access a **range of rows**. Specify two index labels and separate them with a colon (:)

In [128]:

```
bond.loc["Casino Royale" : "Die Another Day"]
```

Out[128]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN
Diamonds Are Forever	1971	Sean Connery	Guy Hamilton	442.5	34.7	5.8
Die Another Day	2002	Pierce Brosnan	Lee Tamahori	465.4	154.2	17.9

Specific only two rows

Put the index label in a square bracket separated by a comma (,)

In [129]:

```
bond.loc[["Diamonds Are Forever" ,"A View to a Kill" ]]
```

Out[129]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Diamonds Are Forever	1971	Sean Connery	Guy Hamilton	442.5	34.7	5.8
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1

iloc() can also be used even though the index is String labelled.

In [130]:

```
bond.iloc[[0]]
```

Out[130]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1

The Catch-All .ix[ ]

Works the same as .loc[ ] and iloc[ ]. However, it is deprecated and advised to use loc[ ] or iloc[ ]

In [131]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[131]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

In [132]:

```
bond.ix[["A View to a Kill"]]
```

```
C:\Users\Ismail\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureW
arning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ix-inde
xer-is-deprecated (http://pandas.pydata.org/pandas-docs/stable/user_guide/in
dexing.html#ix-indexer-is-deprecated)
  """Entry point for launching an IPython kernel.
```

Out[132]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1

Second Arguments in .loc[ ] and iloc[ ]

The second argument indicates the column that we need. Combining the first and second argument enables us to extract very specific data.

In [133]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[133]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

In [134]:

```
bond.loc["A View to a Kill"]
```

Out[134]:

```
Year                      1985
Actor                Roger Moore
Director               John Glen
Box Office                 275.2
Budget                      54.5
Bond Actor Salary            9.1
Name: A View to a Kill, dtype: object
```

Let's extract the budget of "A View to a Kill" movie from the DataFrame.

- **loc** : The arguments can only be string type.
- **iloc** : the arguments can only be integer type.
- **ix** : Arguments can be both integer and string type. However, this method is deprecated.

In [135]:

```
bond.loc["A View to a Kill" , "Budget"]
```

Out[135]:

```
54.5
```

In [136]:

```
bond.iloc[0 , 4]
```

Out[136]:

```
54.5
```

In [137]:

```
bond.ix["A View to a Kill" , 4]
```

```
C:\Users\Ismail\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureW
arning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ix-inde
xer-is-deprecated (http://pandas.pydata.org/pandas-docs/stable/user_guide/in
dexing.html#ix-indexer-is-deprecated)
  """Entry point for launching an IPython kernel.
C:\Users\Ismail\Anaconda3\lib\site-packages\pandas\core\indexing.py:961: Fut
ureWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ix-inde
xer-is-deprecated (http://pandas.pydata.org/pandas-docs/stable/user_guide/in
dexing.html#ix-indexer-is-deprecated)
  return getattr(section, self.name)[new_key]
```

Out[137]:

```
54.5
```

Set a New Values for a Specific Cell or Row

In [138]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[138]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

Add equal sign (=) to a value, and the cell will be replaced with the new value.

In [139]:

```
bond.loc["A View to a Kill", "Year"]
```

Out[139]:

```
1985
```

In [140]:

```
bond.loc["A View to a Kill", "Year"] = 2019
bond.head(3)
```

Out[140]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	2019	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

We can also change more than one cell by calling the list of columns and then assigning them with the new values as follows.

In [141]:

```
bond.loc["A View to a Kill" , ["Year", "Budget" , "Actor"]]
```

Out[141]:

```
Year             2019
Budget           54.5
Actor     Roger Moore
Name: A View to a Kill, dtype: object
```

In [142]:

```
bond.loc["A View to a Kill" , ["Year", "Budget" , "Actor"]] = [2020, 50, "Ali bin Abu" ]
bond.head(3)
```

Out[142]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	2020	Ali bin Abu	John Glen	275.2	50.0	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

Set Multiple Values in DataFrame

For example, we want to change the Actor's name, Roger Moore to Amin Hakim. To do that, we need to extract the specific rows first.

In [143]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[143]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

In [144]:

```
mask = bond.Actor == "Roger Moore"
bond.loc[mask]
```

Out[144]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
For Your Eyes Only	1981	Roger Moore	John Glen	449.4	60.2	NaN
Live and Let Die	1973	Roger Moore	Guy Hamilton	460.3	30.8	NaN
Moonraker	1979	Roger Moore	Lewis Gilbert	535.0	91.5	NaN
Octopussy	1983	Roger Moore	John Glen	373.8	53.9	7.8
The Man with the Golden Gun	1974	Roger Moore	Guy Hamilton	334.0	27.7	NaN
The Spy Who Loved Me	1977	Roger Moore	Lewis Gilbert	533.0	45.1	NaN

Then, we set the second argument as the name of the column of the value we want to change and assign the new value by using the equal (=) sign.

In [145]:

```
bond.loc[mask, "Actor"] = "Amin Hakim"
bond.loc[mask]
```

Out[145]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Amin Hakim	John Glen	275.2	54.5	9.1
For Your Eyes Only	1981	Amin Hakim	John Glen	449.4	60.2	NaN
Live and Let Die	1973	Amin Hakim	Guy Hamilton	460.3	30.8	NaN
Moonraker	1979	Amin Hakim	Lewis Gilbert	535.0	91.5	NaN
Octopussy	1983	Amin Hakim	John Glen	373.8	53.9	7.8
The Man with the Golden Gun	1974	Amin Hakim	Guy Hamilton	334.0	27.7	NaN
The Spy Who Loved Me	1977	Amin Hakim	Lewis Gilbert	533.0	45.1	NaN

Rename Index Labels or Columns in DataFrame

In [146]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.head(3)
```

Out[146]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Dr. No	1962	Sean Connery	Terence Young	448.8	7.0	0.6
From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6
Goldfinger	1964	Sean Connery	Guy Hamilton	820.4	18.6	3.2

To rename columns, specify the column parameter with a dictionary. The dictionary key is the column name that we want to replace. Meanwhile, the corresponding value will be the new name for the column.

In [147]:

```
bond.rename(columns= {
    "Year" : "Released Date",
    "Box Office" : "Revenue"
}, inplace=True)
```

In [148]:

```
bond.head()
```

Out[148]:

Film	Released Date	Actor	Director	Revenue	Budget	Bond Actor Salary
Dr. No	1962	Sean Connery	Terence Young	448.8	7.0	0.6
From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6
Goldfinger	1964	Sean Connery	Guy Hamilton	820.4	18.6	3.2
Thunderball	1965	Sean Connery	Terence Young	848.1	41.9	4.7
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

To rename indexes, specify the index parameter with a dictionary. The dictionary key is the index label that we want to replace while the value will be the new index label.

In [149]:

```
bond.rename(index={
    "Dr. No" : "Doctor No"
}, inplace=True)
```

In [150]:

```
bond.head()
```

Out[150]:

Film	Released Date	Actor	Director	Revenue	Budget	Bond Actor Salary
Doctor No	1962	Sean Connery	Terence Young	448.8	7.0	0.6
From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6
Goldfinger	1964	Sean Connery	Guy Hamilton	820.4	18.6	3.2
Thunderball	1965	Sean Connery	Terence Young	848.1	41.9	4.7
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

Delete Rows or Columns from a DataFrame

In [151]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[151]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

Dropping only a row

In [152]:

```
bond.drop("A View to a Kill", inplace=True)
```

In [153]:

```
bond.head()
```

Out[153]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN
Diamonds Are Forever	1971	Sean Connery	Guy Hamilton	442.5	34.7	5.8
Die Another Day	2002	Pierce Brosnan	Lee Tamahori	465.4	154.2	17.9
Dr. No	1962	Sean Connery	Terence Young	448.8	7.0	0.6

Dropping multiple rows

In [154]:

```
bond.drop(["Casino Royale" , "Dr. No"], inplace=True)
bond.head()
```

Out[154]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Diamonds Are Forever	1971	Sean Connery	Guy Hamilton	442.5	34.7	5.8
Die Another Day	2002	Pierce Brosnan	Lee Tamahori	465.4	154.2	17.9
For Your Eyes Only	1981	Roger Moore	John Glen	449.4	60.2	NaN
From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6
GoldenEye	1995	Pierce Brosnan	Martin Campbell	518.5	76.9	5.1

When dropping columns, we need to include **columns parameter**. For example, we want to drop the Box Office column.

In [155]:

```
bond.drop(columns = "Box Office", inplace=True)
bond.head()
```

Out[155]:

Film	Year	Actor	Director	Budget	Bond Actor Salary
Diamonds Are Forever	1971	Sean Connery	Guy Hamilton	34.7	5.8
Die Another Day	2002	Pierce Brosnan	Lee Tamahori	154.2	17.9
For Your Eyes Only	1981	Roger Moore	John Glen	60.2	NaN
From Russia with Love	1963	Sean Connery	Terence Young	12.6	1.6
GoldenEye	1995	Pierce Brosnan	Martin Campbell	76.9	5.1

In [156]:

```
actor = bond["Actor"]
actor
```

Out[156]:

```
Film
Diamonds Are Forever                 Sean Connery
Die Another Day                    Pierce Brosnan
For Your Eyes Only                    Roger Moore
From Russia with Love                Sean Connery
GoldenEye                          Pierce Brosnan
Goldfinger                           Sean Connery
Licence to Kill                    Timothy Dalton
Live and Let Die                      Roger Moore
Moonraker                             Roger Moore
Never Say Never Again                Sean Connery
Octopussy                             Roger Moore
On Her Majesty's Secret Service    George Lazenby
Quantum of Solace                    Daniel Craig
Skyfall                              Daniel Craig
Spectre                              Daniel Craig
The Living Daylights               Timothy Dalton
The Man with the Golden Gun           Roger Moore
The Spy Who Loved Me                  Roger Moore
The World Is Not Enough            Pierce Brosnan
Thunderball                          Sean Connery
Tomorrow Never Dies                Pierce Brosnan
You Only Live Twice                  Sean Connery
Name: Actor, dtype: object
```

Using Python **del** keyword, we can quickly delete a column.

In [157]:

```
del bond["Actor"]
```

In [158]:

```
bond.head()
```

Out[158]:

Film	Year	Director	Budget	Bond Actor Salary
Diamonds Are Forever	1971	Guy Hamilton	34.7	5.8
Die Another Day	2002	Lee Tamahori	154.2	17.9
For Your Eyes Only	1981	John Glen	60.2	NaN
From Russia with Love	1963	Terence Young	12.6	1.6
GoldenEye	1995	Martin Campbell	76.9	5.1

Extract Random Sample

Random sampling is a method to choose random sample of items while ensuring non-bias selection.

In [159]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[159]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

n parameter indicates how many rows we want.

frac parameter indicates fraction number of rows we want out of the total.

In [160]:

```
bond.sample(n = 10)
```

Out[160]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Live and Let Die	1973	Roger Moore	Guy Hamilton	460.3	30.8	NaN
GoldenEye	1995	Pierce Brosnan	Martin Campbell	518.5	76.9	5.1
The World Is Not Enough	1999	Pierce Brosnan	Michael Apted	439.5	158.3	13.5
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Tomorrow Never Dies	1997	Pierce Brosnan	Roger Spottiswoode	463.2	133.9	10.0
Dr. No	1962	Sean Connery	Terence Young	448.8	7.0	0.6
Quantum of Solace	2008	Daniel Craig	Marc Forster	514.2	181.4	8.1
The Man with the Golden Gun	1974	Roger Moore	Guy Hamilton	334.0	27.7	NaN
For Your Eyes Only	1981	Roger Moore	John Glen	449.4	60.2	NaN

0.25 means 25% of the total number of rows. We have a total of 26 rows so 25% will be equal to 6 rows.

In [161]:

```
bond.sample(frac = 0.25)
```

Out[161]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Skyfall	2012	Daniel Craig	Sam Mendes	943.5	170.2	14.5
From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN
For Your Eyes Only	1981	Roger Moore	John Glen	449.4	60.2	NaN
The Man with the Golden Gun	1974	Roger Moore	Guy Hamilton	334.0	27.7	NaN
The World Is Not Enough	1999	Pierce Brosnan	Michael Apted	439.5	158.3	13.5

We can also extract samples from random column. Try re-running the following code. Notice that there are ONLY 3 random columns selected.

In [162]:

```
bond.sample(n = 3, axis = 1)
```

Out[162]:

Film	Bond Actor Salary	Box Office	Year
A View to a Kill	9.1	275.2	1985
Casino Royale	3.3	581.5	2006
Casino Royale	NaN	315.0	1967
Diamonds Are Forever	5.8	442.5	1971
Die Another Day	17.9	465.4	2002
Dr. No	0.6	448.8	1962
For Your Eyes Only	NaN	449.4	1981
From Russia with Love	1.6	543.8	1963
GoldenEye	5.1	518.5	1995
Goldfinger	3.2	820.4	1964
Licence to Kill	7.9	250.9	1989
Live and Let Die	NaN	460.3	1973
Moonraker	NaN	535.0	1979
Never Say Never Again	NaN	380.0	1983
Octopussy	7.8	373.8	1983
On Her Majesty's Secret Service	0.6	291.5	1969
Quantum of Solace	8.1	514.2	2008
Skyfall	14.5	943.5	2012
Spectre	NaN	726.7	2015
The Living Daylights	5.2	313.5	1987
The Man with the Golden Gun	NaN	334.0	1974
The Spy Who Loved Me	NaN	533.0	1977
The World Is Not Enough	13.5	439.5	1999
Thunderball	4.7	848.1	1965
Tomorrow Never Dies	10.0	463.2	1997
You Only Live Twice	4.4	514.2	1967

nsmallest() and nlargest() methods

nsmallest() returns the first n rows of a specific column which was sorted in ascending order.

In [163]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[163]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

Below is the two Smallest values based on the Box Office.

If we have already selected the column, we can directly use this method on it.

In [164]:

```
bond.nsmallest(2,columns="Box Office")
```

Out[164]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Licence to Kill	1989	Timothy Dalton	John Glen	250.9	56.7	7.9
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1

In [165]:

```
bond["Box Office"].nsmallest(2)
```

Out[165]:

```
Film
Licence to Kill     250.9
A View to a Kill    275.2
Name: Box Office, dtype: float64
```

nlargest() returns the first n rows of a specific column which was sorted in descending order. Below is the 2 Largest values based on the Box Office.

In [166]:

```
bond.nlargest(2,columns="Box Office")
```

Out[166]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
Skyfall	2012	Daniel Craig	Sam Mendes	943.5	170.2	14.5
Thunderball	1965	Sean Connery	Terence Young	848.1	41.9	4.7

In [167]:

```
bond["Box Office"].nlargest(2)
```

Out[167]:

```
Film
Skyfall        943.5
Thunderball    848.1
Name: Box Office, dtype: float64
```

Filtering with where Method

Replace value with NaN value where the condition is False.

In [168]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[168]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

In [169]:

```
mask = bond["Box Office"] > 500
bond.where(mask)
```

Out[169]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	NaN	NaN	NaN	NaN	NaN	NaN
Casino Royale	2006.0	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	NaN	NaN	NaN	NaN	NaN	NaN
Diamonds Are Forever	NaN	NaN	NaN	NaN	NaN	NaN
Die Another Day	NaN	NaN	NaN	NaN	NaN	NaN
Dr. No	NaN	NaN	NaN	NaN	NaN	NaN
For Your Eyes Only	NaN	NaN	NaN	NaN	NaN	NaN
From Russia with Love	1963.0	Sean Connery	Terence Young	543.8	12.6	1.6
GoldenEye	1995.0	Pierce Brosnan	Martin Campbell	518.5	76.9	5.1
Goldfinger	1964.0	Sean Connery	Guy Hamilton	820.4	18.6	3.2
Licence to Kill	NaN	NaN	NaN	NaN	NaN	NaN
Live and Let Die	NaN	NaN	NaN	NaN	NaN	NaN
Moonraker	1979.0	Roger Moore	Lewis Gilbert	535.0	91.5	NaN
Never Say Never Again	NaN	NaN	NaN	NaN	NaN	NaN
Octopussy	NaN	NaN	NaN	NaN	NaN	NaN
On Her Majesty's Secret Service	NaN	NaN	NaN	NaN	NaN	NaN
Quantum of Solace	2008.0	Daniel Craig	Marc Forster	514.2	181.4	8.1
Skyfall	2012.0	Daniel Craig	Sam Mendes	943.5	170.2	14.5
Spectre	2015.0	Daniel Craig	Sam Mendes	726.7	206.3	NaN
The Living Daylights	NaN	NaN	NaN	NaN	NaN	NaN
The Man with the Golden Gun	NaN	NaN	NaN	NaN	NaN	NaN
The Spy Who Loved Me	1977.0	Roger Moore	Lewis Gilbert	533.0	45.1	NaN
The World Is Not Enough	NaN	NaN	NaN	NaN	NaN	NaN
Thunderball	1965.0	Sean Connery	Terence Young	848.1	41.9	4.7
Tomorrow Never Dies	NaN	NaN	NaN	NaN	NaN	NaN
You Only Live Twice	1967.0	Sean Connery	Lewis Gilbert	514.2	59.9	4.4

The .query() Method

Query the columns of a DataFrame using Boolean expressions.

To use the query method, we first have to replace the spaces in all column names with underscore (_). This is because the query method only works if the column name doesn't have any empty spaces.

In [170]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[170]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

In [171]:

```
bond.columns = [column_name.replace(" ", "_") for column_name in bond.columns]
bond.head()
```

Out[171]:

Film	Year	Actor	Director	Box_Office	Budget	Bond_Actor_Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN
Diamonds Are Forever	1971	Sean Connery	Guy Hamilton	442.5	34.7	5.8
Die Another Day	2002	Pierce Brosnan	Lee Tamahori	465.4	154.2	17.9

In [172]:

```
bond.query("Actor == 'David Niven'")
```

Out[172]:

Film	Year	Actor	Director	Box_Office	Budget	Bond_Actor_Salary
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

Another example:

In [173]:

```
bond.query("Box_Office > 500 ")
```

Out[173]:

Film	Year	Actor	Director	Box_Office	Budget	Bond_Actor_Salary
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
From Russia with Love	1963	Sean Connery	Terence Young	543.8	12.6	1.6
GoldenEye	1995	Pierce Brosnan	Martin Campbell	518.5	76.9	5.1
Goldfinger	1964	Sean Connery	Guy Hamilton	820.4	18.6	3.2
Moonraker	1979	Roger Moore	Lewis Gilbert	535.0	91.5	NaN
Quantum of Solace	2008	Daniel Craig	Marc Forster	514.2	181.4	8.1
Skyfall	2012	Daniel Craig	Sam Mendes	943.5	170.2	14.5
Spectre	2015	Daniel Craig	Sam Mendes	726.7	206.3	NaN
The Spy Who Loved Me	1977	Roger Moore	Lewis Gilbert	533.0	45.1	NaN
Thunderball	1965	Sean Connery	Terence Young	848.1	41.9	4.7
You Only Live Twice	1967	Sean Connery	Lewis Gilbert	514.2	59.9	4.4

Example with two query conditions.

In [174]:

```
bond.query("Box_Office > 500 and Actor == 'Daniel Craig'")
```

Out[174]:

Film	Year	Actor	Director	Box_Office	Budget	Bond_Actor_Salary
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Quantum of Solace	2008	Daniel Craig	Marc Forster	514.2	181.4	8.1
Skyfall	2012	Daniel Craig	Sam Mendes	943.5	170.2	14.5
Spectre	2015	Daniel Craig	Sam Mendes	726.7	206.3	NaN

Apply() method on Single Column

apply() method enable us to apply function along an axis of the DataFrame.

For example, we want to add "Millions" at each cell in the Box Office column. Firstly, we create add_million() function and then use its name as the argument in the apply() method.

In [175]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[175]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

In [176]:

```
def add_million(number):
    return " " + str(number) + "Millions"
```

In [177]:

```
bond["Box Office"] = bond["Box Office"].apply(add_million)
```

In [178]:

```
bond.head(3)
```

Out[178]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2Millions	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5Millions	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0Millions	85.0	NaN

Apply function() for each row.

This can be done by using the axis parameter and specifying it as "columns".

In [179]:

```
def feedback_movie(row):
    actor = row[1] # Actor column
    director = row[2] #Director column
    budget = row[4] #Budget column

    if actor == "Roger Moore" :
        return "Enjoyable"
    elif director == "David Niven" and budget > 40:
        return "Exciting movie"
    else :
        return "Pretty decent"

bond.apply(feedback_movie, axis = "columns")
```

Out[179]:

```
Film
A View to a Kill                       Enjoyable
Casino Royale                     Pretty decent
Casino Royale                     Pretty decent
Diamonds Are Forever              Pretty decent
Die Another Day                   Pretty decent
Dr. No                            Pretty decent
For Your Eyes Only                     Enjoyable
From Russia with Love             Pretty decent
GoldenEye                         Pretty decent
Goldfinger                        Pretty decent
Licence to Kill                   Pretty decent
Live and Let Die                       Enjoyable
Moonraker                              Enjoyable
Never Say Never Again             Pretty decent
Octopussy                              Enjoyable
On Her Majesty's Secret Service   Pretty decent
Quantum of Solace                 Pretty decent
Skyfall                           Pretty decent
Spectre                           Pretty decent
The Living Daylights              Pretty decent
The Man with the Golden Gun            Enjoyable
The Spy Who Loved Me                   Enjoyable
The World Is Not Enough           Pretty decent
Thunderball                       Pretty decent
Tomorrow Never Dies               Pretty decent
You Only Live Twice               Pretty decent
dtype: object
```

.copy() method

- Makes a copy of an object and any changes made on the new object will not affect the original dataset.
- Preserve the original dataset

In [180]:

```
bond = pd.read_csv("data/jamesbond.csv", index_col="Film")
bond.sort_index(inplace=True)
bond.head(3)
```

Out[180]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	John Glen	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN

Here, we create a copy of the Director column. Then, we assign it to a variable named directors.

In [181]:

```
directors = bond["Director"]
directors.head(3)
```

Out[181]:

```
Film
A View to a Kill          John Glen
Casino Royale       Martin Campbell
Casino Royale            Ken Hughes
Name: Director, dtype: object
```

Let's change the name of A View to a Kill's movie director to something else.

In [182]:

```
directors["A View to a Kill"] = "Ali bin Abu"
directors.head(3)
```

```
C:\Users\Ismail\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (http://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy)
  """Entry point for launching an IPython kernel.
```

Out[182]:

```
Film
A View to a Kill        Ali bin Abu
Casino Royale       Martin Campbell
Casino Royale            Ken Hughes
Name: Director, dtype: object
```

If you look at the original dataset, there are no changes in the director's name. This is because the change was made in the copy of the dataset.

In [183]:

```
bond.head(3)
```

Out[183]:

Film	Year	Actor	Director	Box Office	Budget	Bond Actor Salary
A View to a Kill	1985	Roger Moore	Ali bin Abu	275.2	54.5	9.1
Casino Royale	2006	Daniel Craig	Martin Campbell	581.5	145.3	3.3
Casino Royale	1967	David Niven	Ken Hughes	315.0	85.0	NaN