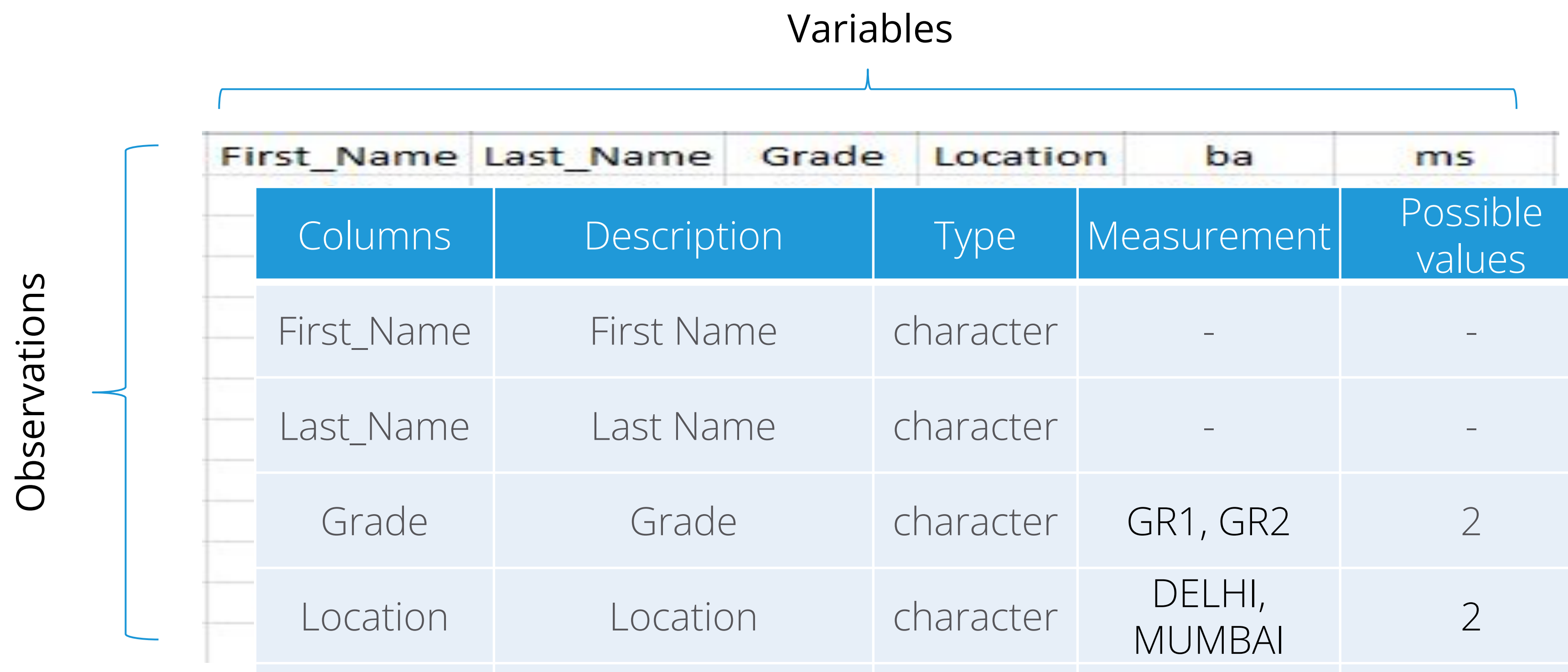
Data Management in Python –

Creating Subsets & Sorting Data

Contents

Data Snapshot

basic_salary data consist salary of each employee with it's Location & Grade.



First_Name Last_Name Grade Location ba ms

Columns	Description	Type	Measurement	Possible values
First_Name	First Name	character	-	-
Last_Name	Last Name	character	-	-
Grade	Grade	character	GR1, GR2	2
Location	Location	character	DELHI, MUMBAI	2
ba	Basic Allowance	numeric	Rs.	positive values
ms	Management Supplements	numeric	Rs.	positive values

* Here we continue to use previous data for our further analysis.

Need for Creating Subsets

- Sometimes we want to view filtered snippet, or to extract just the data we are interested in from a data frame.

- Python doesn't need any additional functions to slice its data,

Indexing & Slicing in Pandas

- axis labelling function in Python helps identify observations and variables
- Python and NumPy indexing operators [ ] and attribute operator provide quick and easy access
- Pandas support 2 types of multi-indexing, **`loc`** and **`iloc`**.
- **`loc`** is used for label based indexing whereas **`iloc`** is primarily integer position based (from 0 to length -1 of the axis).

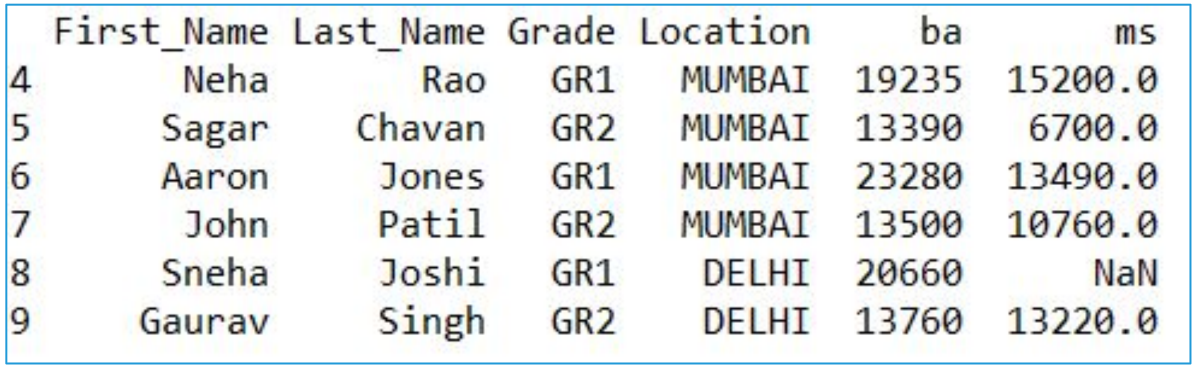
Row Subsetting

- The **loc** function is used for label based indexing so it accepts labels and integers, provided that the integers are labels and not the index itself. However, note that python follows 0 index.

```
# Import data & Display rows from 5th to 10th
```

```
import pandas as pd
salary_data_org= pd.read_csv('basic_salary.csv')
salary_data_org.loc[4:9]
```

```
# Output
```



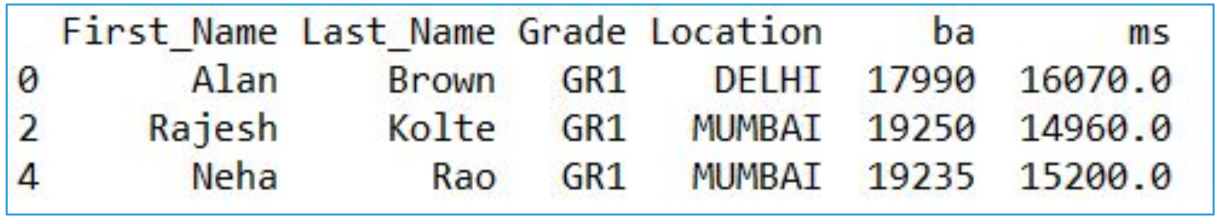
```
  First_Name Last_Name Grade Location     ba       ms
4       Neha       Rao   GR1   MUMBAI  19235  15200.0
5      Sagar    Chavan   GR2   MUMBAI  13390   6700.0
6      Aaron     Jones   GR1   MUMBAI  23280  13490.0
7       John     Patil   GR2   MUMBAI  13500  10760.0
8      Sneha     Joshi   GR1    DELHI  20660      NaN
9     Gaurav     Singh   GR2    DELHI  13760  13220.0
```

Row Subsetting

```
# Display row numbers 1,3 and 5 only
```

```
salary_data_org.loc[[0,2,4]]
```

```
# Output
```



```
  First_Name Last_Name Grade Location     ba       ms
0       Alan     Brown   GR1    DELHI  17990  16070.0
2     Rajesh     Kolte   GR1   MUMBAI  19250  14960.0
4       Neha       Rao   GR1   MUMBAI  19235  15200.0
```

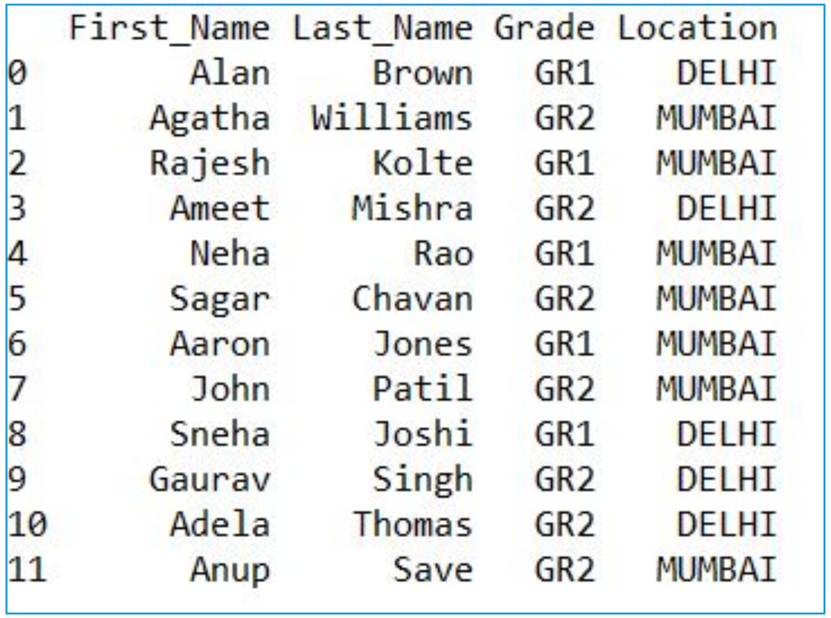
* We use .loc[[ ]] when we want specific rows only.

Column Subsetting

```
# Display columns 1 to 4
salary_data_org.iloc[:,0:4]
```

iloc helps use index by position. The row index is given first and the column index is added after a comma. Since a range of index is used here, the fact that all the rows have to be shown is denoted by the empty range.

```
# Output
```

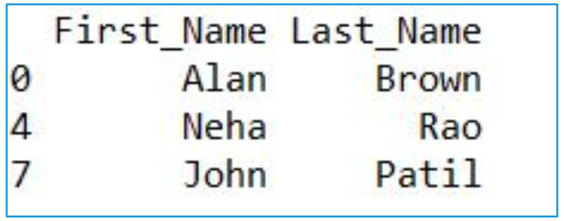


```
   First_Name Last_Name Grade Location
0        Alan     Brown   GR1    DELHI
1      Agatha  Williams   GR2   MUMBAI
2      Rajesh     Kolte   GR1   MUMBAI
3       Ameet    Mishra   GR2    DELHI
4        Neha       Rao   GR1   MUMBAI
5       Sagar    Chavan   GR2   MUMBAI
6       Aaron     Jones   GR1   MUMBAI
7        John     Patil   GR2   MUMBAI
8       Sneha     Joshi   GR1    DELHI
9      Gaurav     Singh   GR2    DELHI
10      Adela    Thomas   GR2    DELHI
11       Anup      Save   GR2   MUMBAI
```

Row-Column Subsetting

```
# Display rows 1,5,8 and columns 1 and 2
# With labels
salary_data_org.loc[[0,4,7],['First_Name','Last_Name']]
```

```
# Output
```

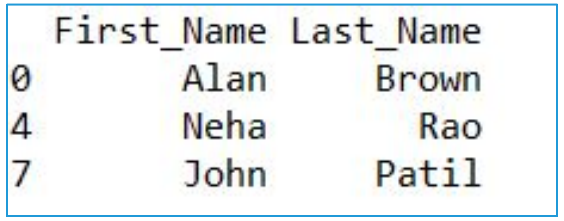


```
  First_Name Last_Name
0       Alan     Brown
4       Neha       Rao
7       John     Patil
```

```
# With Index

salary_data_org.iloc[[0,4,7],[0,1]]
```

```
# Output
```



```
  First_Name Last_Name
0       Alan     Brown
4       Neha       Rao
7       John     Patil
```

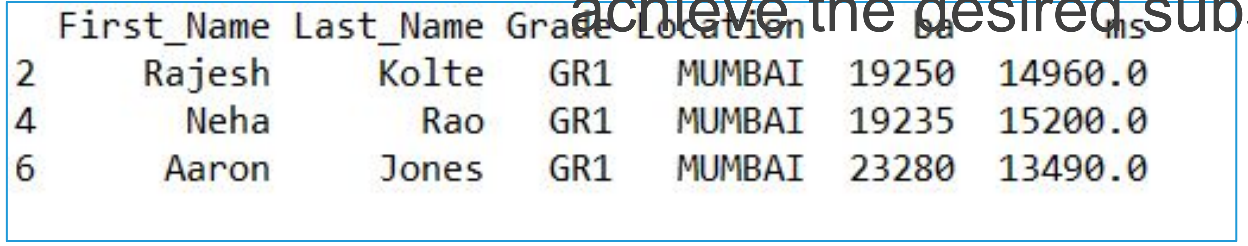
Subsetting Observations

```
# Create a subset with all details of employees of MUMBAI with ba
# more than 15000
```

```
salary_data_org[(salary_data_org.Location=='MUMBAI')
&(salary_data_org.ba>15000)]
```

There is no limit on how many conditions may be combined to achieve the desired subset.

```
# Output
```



```
  First_Name Last_Name Grade Location    ba       ms
2     Rajesh     Kolte   GR1   MUMBAI 19250  14960.0
4       Neha       Rao   GR1   MUMBAI 19235  15200.0
6      Aaron     Jones   GR1   MUMBAI 23280  13490.0
```

Subsetting Observations

```
salary_data_org[(salary_data_org.Grade!='GR1') &
(salary_data_org.Location!="MUMBAI")]
```

```
# Output
```

```
   First_Name Last_Name Grade Location     ba       ms
3       Ameet    Mishra   GR2    DELHI  14780   9300.0
9      Gaurav     Singh   GR2    DELHI  13760  13220.0
10      Adela    Thomas   GR2    DELHI  13660   6840.0
```

Not Equal To (!) operator is used to give condition.

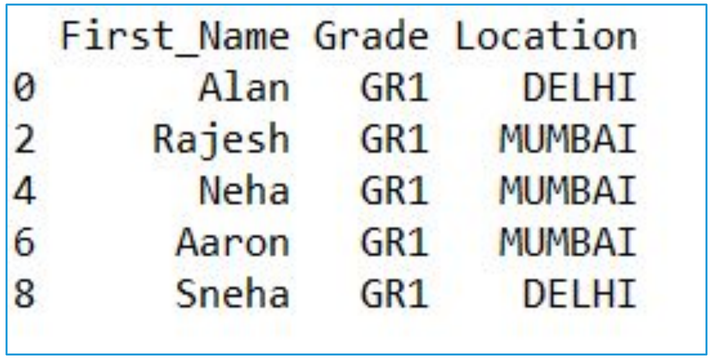
Subsetting Both Observations and Variables

We can subset observations and variables by simply combining the previous two methods of subsetting.

```
# Select First_Name, Grade and Location of employees of GR1 with ba
# more than 15000
```

```
salary_data_org.loc[(salary_data_org.Grade=='GR1') &
(salary_data_org.ba>15000), ['First_Name','Grade', 'Location']]
```

```
# Output
```



```
  First_Name Grade Location
0       Alan   GR1    DELHI
2     Rajesh   GR1   MUMBAI
4       Neha   GR1   MUMBAI
6      Aaron   GR1   MUMBAI
8      Sneha   GR1    DELHI
```

We're are combining the boolean conditions with **loc** function as we're trying to subset the dataframe by label positioning.

Quick Recap

Using loc, iloc	• Row Subsetting: By specifying the row labels using integers in []. • Column Subsetting: By specifying the column labels in []. • Row-Column Subsetting: By combining the above two methods.
Using Boolean Conditions	• Subsetting observations: By giving conditions on columns using this function. • Subsetting both observations and variables: By simply combining above two methods.

Introduction

Sorting data is one of the common activities in preparing data for analysis

Sorting is storage of data in sorted order, it can be in ascending or descending order.

```
# Import Pandas and basic_salary data
```

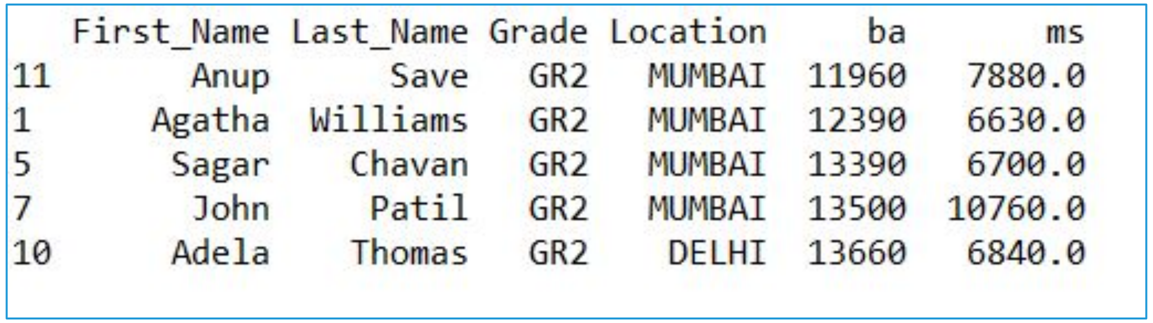
```
import pandas as pd
salary_data = pd.read_csv('basic_salary.csv')
```

Ascending Data

```
# Sort salary_data by ba in Ascending order
```

```
ba_sorted_1=salary_data.sort_values(by=['ba'])
ba_sorted_1.head()
```

```
# Output
```



```
   First_Name Last_Name Grade Location     ba       ms
11       Anup      Save   GR2   MUMBAI  11960   7880.0
1      Agatha  Williams   GR2   MUMBAI  12390   6630.0
5       Sagar    Chavan   GR2   MUMBAI  13390   6700.0
7        John     Patil   GR2   MUMBAI  13500  10760.0
10      Adela    Thomas   GR2    DELHI  13660   6840.0
```

By default, **sort_values()** sorts data in ascending order

Descending Order

```
# Sort salary_data by ba in Descending order
```

```
ba_sorted_2=salary_data.sort_values(by=['ba'], ascending = [0])
ba_sorted_2.head()
```

```
# Output
```

```
  First_Name Last_Name Grade Location     ba       ms
6      Aaron     Jones   GR1   MUMBAI  23280  13490.0
8      Sneha     Joshi   GR1    DELHI  20660      NaN
2     Rajesh     Kolte   GR1   MUMBAI  19250  14960.0
4       Neha       Rao   GR1   MUMBAI  19235  15200.0
0       Alan     Brown   GR1    DELHI  17990  16070.0
```

Here, we are defining ascending as false by passing the Boolean argument 0.

Sorting by Factor Variable

Sort data by column with characters / factors

```
# Sort salary_data by Grade
```

```
gr_sorted=salary_data.sort_values(by=['Grade'])
gr_sorted.head()
```

```
# Output
```

```
  First_Name Last_Name Grade Location     ba      ms
0       Alan     Brown   GR1    DELHI  17990  16070.0
2     Rajesh     Kolte   GR1   MUMBAI  19250  14960.0
4       Neha       Rao   GR1   MUMBAI  19235  15200.0
6      Aaron     Jones   GR1   MUMBAI  23280  13490.0
8      Sneha     Joshi   GR1    DELHI  20660      NaN
```

Note that by default even with factor variables, **sort_values()** sorts by ascending.

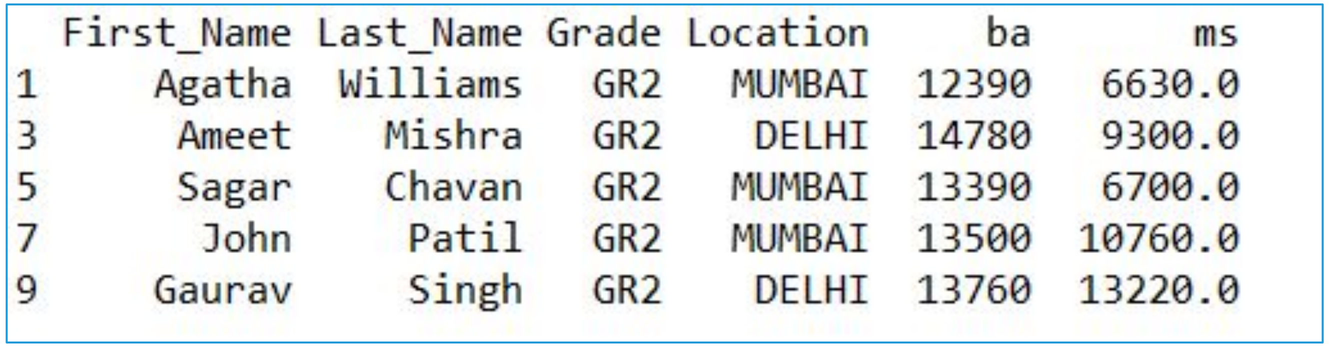
Sorting by Factor Variable

Sort data by column with characters / factors in Descending order

```
# Sort salary_data by Grade in Descending order
```

```
gr_sorted=salary_data.sort_values(by=['Grade'], ascending = [0])
gr_sorted.head()
```

```
# Output
```



```
  First_Name Last_Name Grade Location     ba       ms
1     Agatha  Williams   GR2   MUMBAI  12390   6630.0
3      Ameet    Mishra   GR2    DELHI  14780   9300.0
5      Sagar    Chavan   GR2   MUMBAI  13390   6700.0
7       John     Patil   GR2   MUMBAI  13500  10760.0
9     Gaurav     Singh   GR2    DELHI  13760  13220.0
```

Sorting Data by Multiple Variables

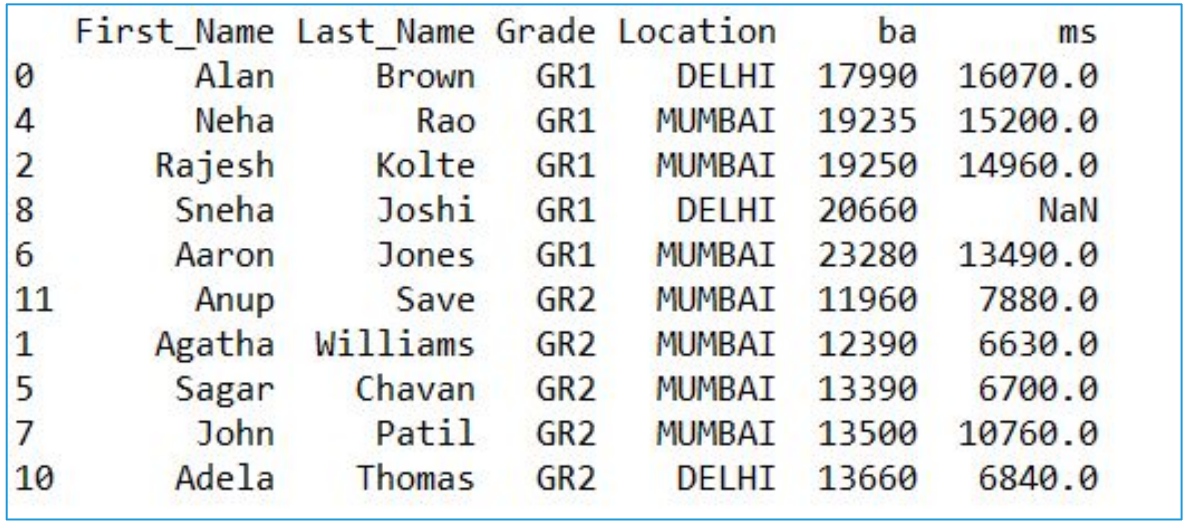
Sort data by giving multiple columns; one column with characters / factors and one with numerals

```
# Sort salary_data by Grade and ba
```

```
grba_sorted=salary_data.sort_values(by=['Grade','ba'])
grba_sorted.head(10)
```

Here, data is first sorted in increasing order of **Grade** then **ba.**

```
# Output
```



```
   First_Name Last_Name Grade Location     ba       ms
0        Alan     Brown   GR1    DELHI  17990  16070.0
4        Neha       Rao   GR1   MUMBAI  19235  15200.0
2      Rajesh     Kolte   GR1   MUMBAI  19250  14960.0
8       Sneha     Joshi   GR1    DELHI  20660      NaN
6       Aaron     Jones   GR1   MUMBAI  23280  13490.0
11       Anup      Save   GR2   MUMBAI  11960   7880.0
1      Agatha  Williams   GR2   MUMBAI  12390   6630.0
5       Sagar    Chavan   GR2   MUMBAI  13390   6700.0
7        John     Patil   GR2   MUMBAI  13500  10760.0
10      Adela    Thomas   GR2    DELHI  13660   6840.0
```

Multiple Variables & Multiple Ordering Levels

Sort data by giving multiple columns; one column with characters / factors and one with numerals and multiple ordering levels

```
# Sort salary_data by Grade in Descending order and then by ms in
# Ascending order
```

```
grms_sorted=salary_data.sort_values(by=['Grade','ms'],
            ascending=[0,1])
grms_sorted.head(10)
```

```
# Output
```

```
   First_Name Last_Name Grade Location     ba       ms
1      Agatha  Williams   GR2   MUMBAI  12390   6630.0
5       Sagar    Chavan   GR2   MUMBAI  13390   6700.0
10      Adela    Thomas   GR2    DELHI  13660   6840.0
11       Anup      Save   GR2   MUMBAI  11960   7880.0
3       Ameet    Mishra   GR2    DELHI  14780   9300.0
7        John     Patil   GR2   MUMBAI  13500  10760.0
9      Gaurav     Singh   GR2    DELHI  13760  13220.0
6       Aaron     Jones   GR1   MUMBAI  23280  13490.0
2      Rajesh     Kolte   GR1   MUMBAI  19250  14960.0
4        Neha       Rao   GR1   MUMBAI  19235  15200.0
```

- Here, data is sorted by **Grade** in descending order and **ms** in ascending order.
- By default missing values in data are put last.
- You can put it first by adding an

Quick Recap

In this session, we learnt sorting data using **`sort_values`** in various ways.

Ascending/ Descending Order	• **`sort_values`** by default sorts in ascending order. • For descending order: specify ascending**`=[0]`** for all variables.
Multiple Columns	• **`sort_values`** allows us to sort by multiple columns of different type
Multiple Columns and Multiple Ordering Levels	• **`sort_values`** provides flexibility to order by multiple columns with different ordering levels