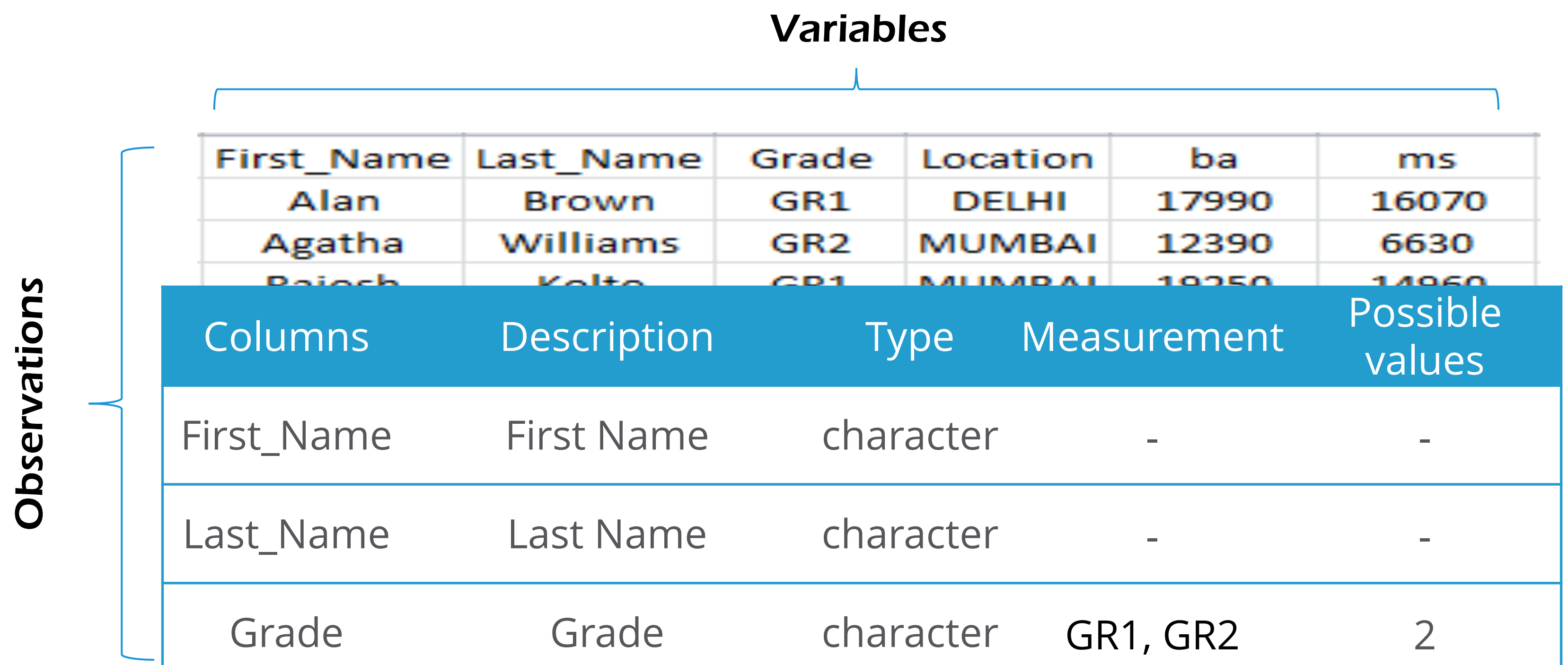
Data Management in Python –

Checking & Modifying Data

Data Snapshot

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


Variables
Observations

First_Name	Last_Name	Grade	Location	ba	ms
Alan	Brown	GR1	DELHI	17990	16070
Agatha	Williams	GR2	MUMBAI	12390	6630
Rajesh	Kolte	GR1	MUMBAI	19250	14960

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

Dimension of Data and Names of the Columns

Use the following commands to know how many rows and columns are there in our data and the names of the columns it contains:

```
# Retrieve the dimension of data
salary_data_org.shape

(12, 6)
```

- ❑ ***shape*** *gives row and column dimension of the data. This data contains 12 rows and 6 columns.*
- ❑ *Alternatively,* ***data.shape[0]*** *and* ***data.shape[1]*** *can be used separately to know no. of rows and columns respectively.*

```
# Get the Names of the columns
list(salary_data_org)
['First_Name', 'Last_Name', 'Grade', 'Location', 'ba', 'ms']
```

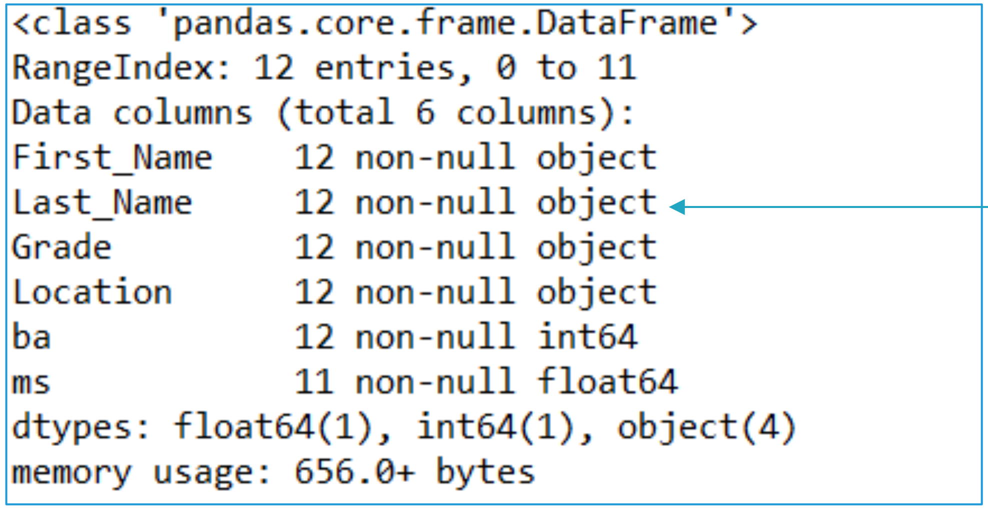
- ❑ ***list()*** *gives column names.*
- ❑ *You can also use* ***salary_data.columns*** *instead to get the column names*

Internal Structure of Data

When Python reads data, it treats different variable types in different ways. `info()` compactly displays a dataframe's internal structure:

```
salary_data_org.info()
```

```
# Output
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 6 columns):
First_Name    12 non-null object
Last_Name     12 non-null object
Grade         12 non-null object
Location      12 non-null object
ba            12 non-null int64
ms            11 non-null float64
dtypes: float64(1), int64(1), object(4)
memory usage: 656.0+ bytes
```

Character variables are entered into a dataframe as object in Python

This gives us the following information:

- Type of the variable.
- Memory usage of the data

Check Levels of a Categorical Variable

Our data has 4 object variables. A variable of data type 'object' is a categorical variable but in Python it has to be explicitly converted to the data type 'category' to be treated as one. Let's convert the variable Location to 'category' and check the number of levels it has using the `column.cat.categories` method:

```
salary_data_org['Location']=salary_data_org['Location'].astype('category')
salary_data_org['Location'].cat.categories
```

```
Index(['DELHI', 'MUMBAI'], dtype='object')
```

Check the Size of an Object

Suppose we want to know how much memory space is used to store `salary_data` object, we can use `memory_usage()` function to get an estimate in bytes.

```
salary_data_org.memory_usage()


Index           80
First_Name      96
Last_Name       96
Grade           96
Location       108
ba              96
ms              96
dtype: int64
```

Number of Missing Observations

Our data might contain some missing values or observations. In Python, missing data are usually recorded as NaN. We can check the number of missing observations like this:

```
salary_data_org.isnull().sum()

First_Name    0
Last_Name     0
Grade         0
Location      0
ba            0
ms            1
dtype: int64
```

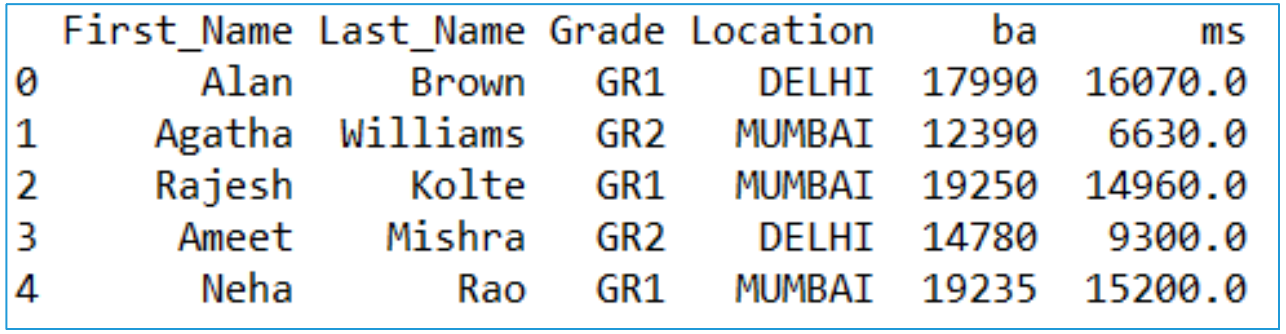
- ***isnull()*** *returns a Boolean dataframe that returns TRUE for each null value'*
- ***sum()*** *displays the sum of missing observations.*

First n Rows of Data

To check how your data looks, without revealing the entire data set, which could have millions of rows and thousands of columns, we can use `head()` to obtain first n observations.

```
salary_data_org.head()
```

```
# Output
```



```
  First_Name Last_Name Grade Location     ba       ms
0       Alan     Brown   GR1    DELHI  17990  16070.0
1     Agatha  Williams   GR2   MUMBAI  12390   6630.0
2     Rajesh     Kolte   GR1   MUMBAI  19250  14960.0
3      Ameet    Mishra   GR2    DELHI  14780   9300.0
4       Neha       Rao   GR1   MUMBAI  19235  15200.0
```

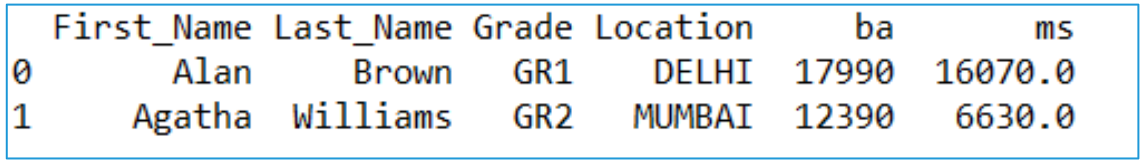
* By default, head() displays the first 5 rows

First n Rows of Data

The no. of rows to be displayed can be customised to n

```
salary_data_org.head(n=2)
```

```
# Output
```



```
  First_Name Last_Name Grade Location     ba       ms
0       Alan     Brown   GR1    DELHI  17990  16070.0
1     Agatha  Williams   GR2   MUMBAI  12390   6630.0
```

Last n Rows of Data

Now we will see the last n rows of our data using `tail()`. By default, it displays last 5 rows.

```
salary_data_org.tail()
```

```
# Output
```

```
   First_Name Last_Name Grade Location     ba       ms
7        John     Patil   GR2   MUMBAI  13500  10760.0
8       Sneha     Joshi   GR1    DELHI  20660      NaN
9      Gaurav     Singh   GR2    DELHI  13760  13220.0
10      Adela    Thomas   GR2    DELHI  13660   6840.0
11       Anup      Save   GR2   MUMBAI  11960   7880.0
```

The no. of rows to be displayed can be customised to n

```
salary_data_org.tail(n=2)
```

```
# Output
```

```
   First_Name Last_Name Grade Location     ba      ms
10      Adela    Thomas   GR2    DELHI  13660  6840.0
11       Anup      Save   GR2   MUMBAI  11960  7880.0
```

Summarising Data

We can also inspect our data using `describe()`. This function gives summary of objects including datasets, variables, linear models, etc

```
# Variables are summarised based on their type
```

```
salary_data_org.describe(include='all')
```

***describe()** is essentially applied to each column and it summarises all the columns.*
*It only provides summary of numeric variables until explicitly programmed to include factor variables using **include ='all'.***

```
    First_Name             Last_Name    Grade         Location      ba
             ms
count        12            12           12            12
             12.0          11.0
unique       12            12           2             2
             NaN           NaN
top          Rajesh        Kolte        GR2           MUMBAI
             NaN           NaN
freq         1             1            7             7
             NaN           NaN
mean         NaN           NaN          NaN           NaN
             16154.58   11004.54
std          NaN           NaN          NaN           NaN
             3739.37    3711.18
```

Change Variable Names – rename()

Our data is saved as an object named salary_data.

Suppose we want to change the name of some variable (column) and its values.

Let's rename the ‘ba’ variable to ‘basic_allowance’ -

```
salary_data = salary_data_org.rename(columns={'ba':'basic_allowance'})
list(salary_data)
```

```
['First_Name', 'Last_Name', 'Grade', 'Location', 'basic_allowance',
'ms']
```

- ***rename()*** *uses name of the data object and assign* ***{'old name':'new name'}.***
- *The result needs to be saved in an object because* ***rename()*** *doesn't modify the object directly.*
- *You can rename multiple column names like this:*
- ***salary_data=salary_data.rename(columns= {'ba':'basic_allowance', 'ms':'management_supplements'})***

Derive a New Variable

Add a new variable to `salary_data` containing values as 5% of ba. We will use the **`assign()`** function to accomplish this:

```
salary_data=salary_data.assign(newvariable=salary_data['basic_allowance']
*0.05)
salary_data.head(n=3)
```

```
# Output
```

```
    First_Name    Last_Name           Grade   Location    basic_allowance ms
newvariable
0   Alan            Brown             GR1     DELHI       17990  16070.0
            899.5
1   Agatha        Williams            GR2     MUMBAI      12390  6630.0
            619.5
2   Rajesh        Kolte   GR1     MUMBAI      19250  14960.0      962.5
```

Remove Columns from a Data Frame

Remove the column Last_Name from salary_data.

```
salary_data.drop('Last_Name',axis=1,inplace=True)
salary_data.head()
```

```
# Output
```

```
  First_Name Grade  Location  basic_allowance       ms  newvariable Category
0       Alan   GR1         2            17990  16070.0       899.50   medium
1     Agatha   GR2         1            12390   6630.0       619.50      low
2     Rajesh   GR1         1            19250  14960.0       962.50     high
3      Ameet   GR2         2            14780   9300.0       739.00   medium
4       Neha   GR1         1            19235  15200.0       961.75     high
```

Remove Rows from a Data Frame

We can remove unwanted rows from our data by using their index nos.

Suppose we want to remove rows 2, 3 and 4 (i.e index 1,2 and 3)from salary_data

then we will write the following command:

```
salary_data.drop(salary_data.index[1:4], axis=0, inplace=True)
salary_data.head(n=4)
```

```
# Output
```

```
  First_Name Grade  Location  basic_allowance       ms  newvariable Category
0       Alan   GR1         2            17990  16070.0       899.50   medium
4       Neha   GR1         1            19235  15200.0       961.75     high
5      Sagar   GR2         1            13390   6700.0       669.50      low
6      Aaron   GR1         1            23280  13490.0      1164.00     high
```

Remove Rows from a Data Frame

Remove only rows which has Location as ‘MUMBAI’ i.e. 1

```
salary_data.drop(salary_data[salary_data.Location==1].index,
inplace=True)
salary_data
```

```
# Output
   First_Name Grade  Location  basic_allowance       ms  newvariable Category
0        Alan   GR1         2            17990  16070.0        899.5   medium
8       Sneha   GR1         2            20660      NaN       1033.0     high
9      Gaurav   GR2         2            13760  13220.0        688.0      low
10      Adela   GR2         2            13660   6840.0        683.0      low
```