Data Management in Python -

Importing & Exporting Data

(CSV, TXT, XLSX, SAS, STATA, SPSS, MySQL, PostgreSQL and Oracle)

Contents

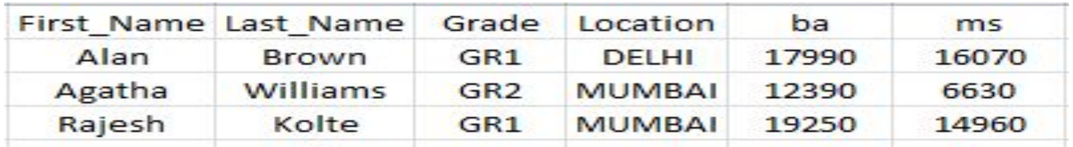
1. Importance of Checking Data
2. Know the Dimensions of Data and Variable Names
3. Display the Internal Structure of Data
4. Check the Levels of a Categorical Variable
5. Check the Size of an Object
6. Check the Number of Missing Observations
7. Display First n Rows of Data
8. Display Last n Rows of Data
9. Summarise Your Data
10. Change Variable Names and Content of Data
11. Derive a New Variable
12. Recode a Categorical Variable
13. Recode a Continuous Variable into Categorical Variable
14. Create a Decile Object
15. Remove Columns from a Data Frame
16. Remove Rows from a Data Frame

Data Snapshot

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

Variables

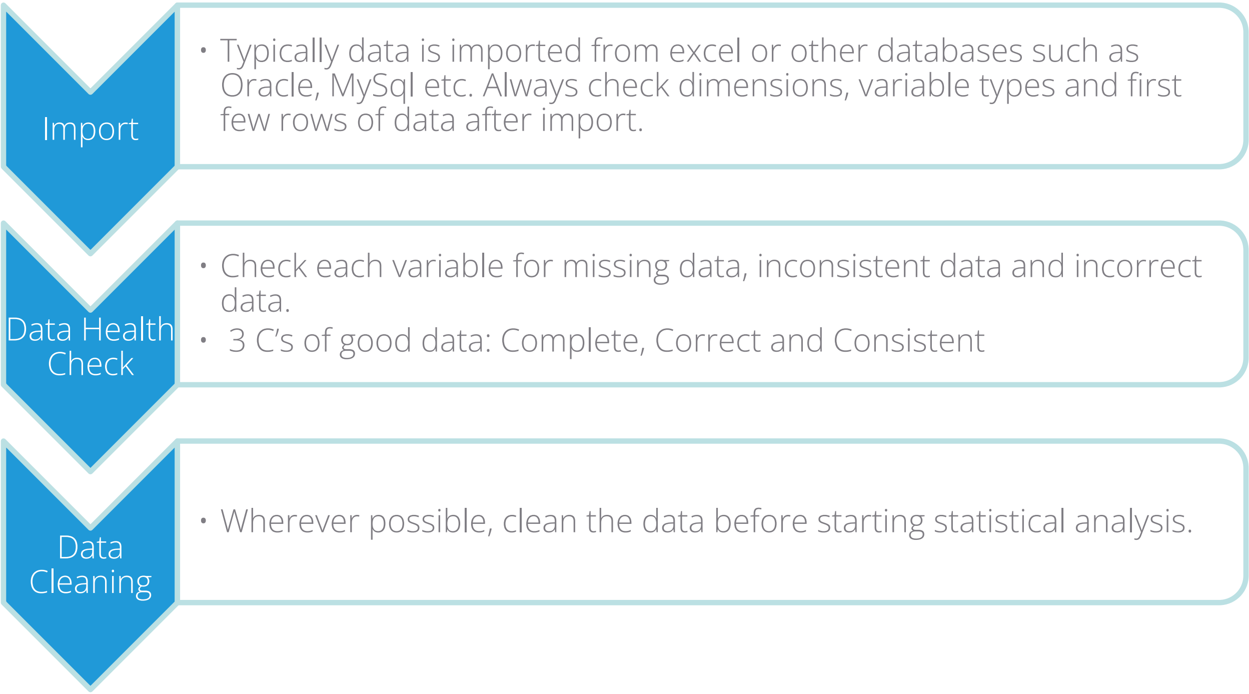
Observations



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

First Look at the Data



Import

- Typically data is imported from excel or other databases such as Oracle, MySql etc. Always check dimensions, variable types and first few rows of data after import.

Data Health Check

- Check each variable for missing data, inconsistent data and incorrect data.
- 3 C's of good data: Complete, Correct and Consistent

Data Cleaning

- Wherever possible, clean the data before starting statistical analysis.

Why Data Checking is Important?

After importing the data into Python and before analysing the data, it is very important to understand your data & check that it has maintained the correct format and also to know how your data looks like, how are the variables treated, what types of variables data contains, how many missing observations are there in your data, how many rows and columns does it contain, etc. so that you get familiar with the data with which you are working. This preliminary step is the foundation for further data analysis, it helps you make your initial inferences about the data before you can start modelling/testing it.

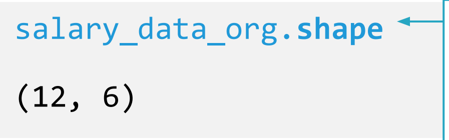
```
# Import basic_salary data
```

```
import pandas as pd
# current directory is already specified
salary_data_org= pd.read_csv('basic_salary.csv')
```

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:

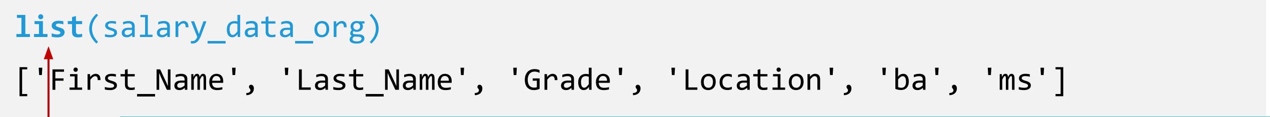
```
# Retrieve the dimension of data
```



```
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']
```

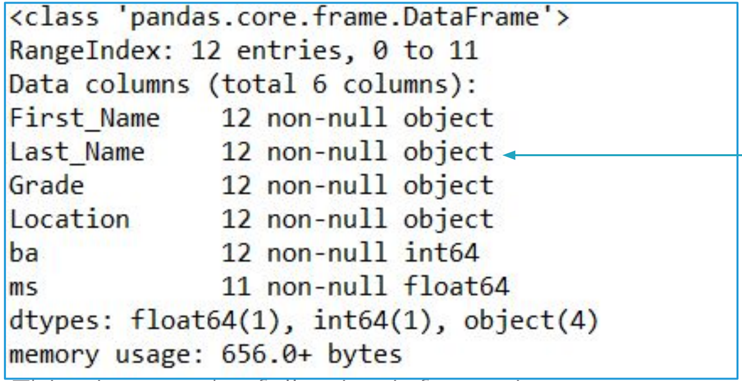
- **list()** gives column names.
- You can also use **salary_data.columns** instead to get the column names

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
```

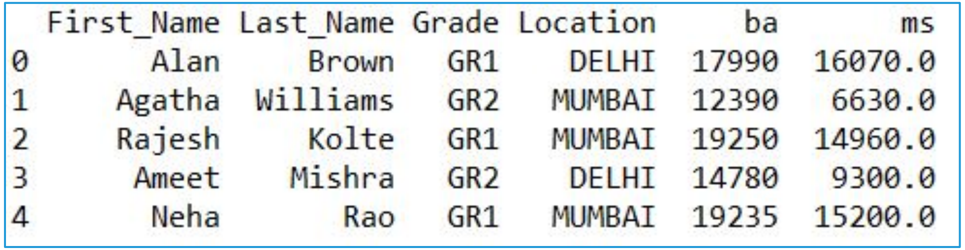
- ❑ **isnull()** returns a Boolean dataframe that returns TRUE for each null value'
- ❑ **sum()** displays the sum of missing observations.

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
```

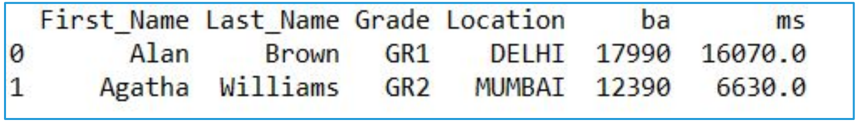
By default, `head()` displays the first 5 rows

First n Rows of Data

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

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

```
# Output
```



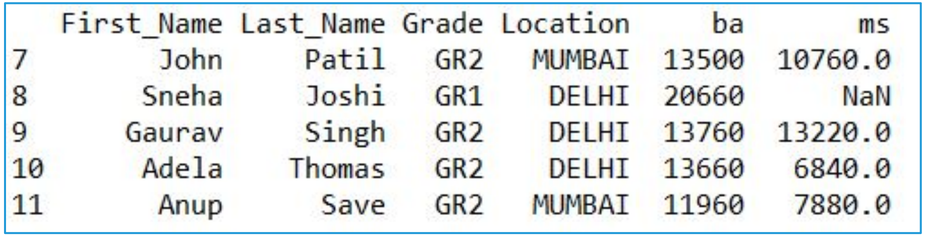
```
  First_Name Last_Name Grade Location     ba       ms
0       Alan     Brown   GR1    DELHI  17990  16070.0
1     Agatha  Williams   GR2   MUMBAI  12390   6630.0
```

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
```

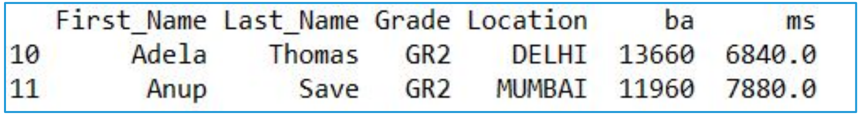


```
   First_Name Last_Name Grade Location     ba       ms
7        John     Patil   GR2   MUMBAI  13500  10760.0
8       Sneha     Joshi   GR1    DELHI  20660      NaN
9      Gaurav     Singh   GR2    DELHI  13760  13220.0
10      Adela    Thomas   GR2    DELHI  13660   6840.0
11       Anup      Save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

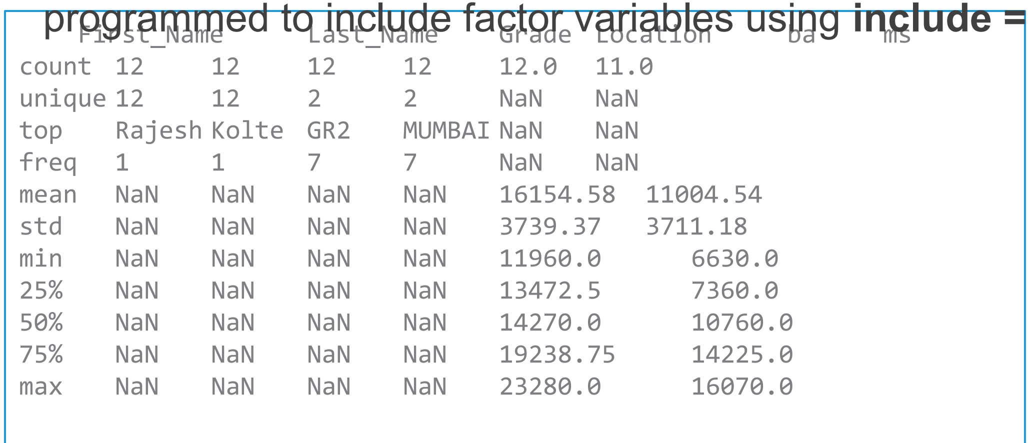
Summarising Data

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'**.

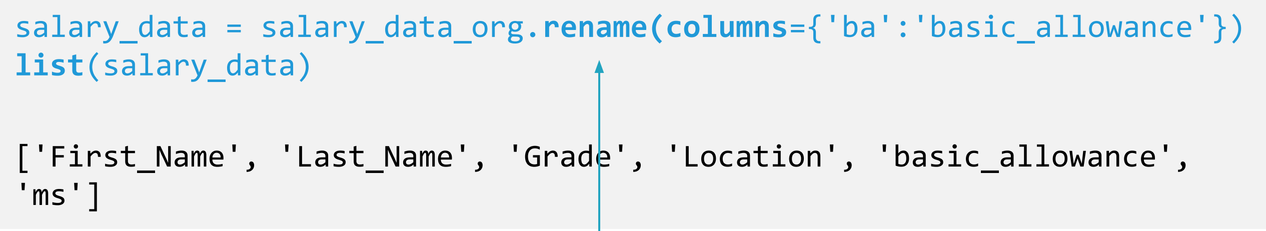


```
          First_Name      Last_Name    Grade  Location     ba     ms
count  12      12      12      12      12.0    11.0
unique 12      12      2       2       NaN     NaN
top    Rajesh  Kolte   GR2     MUMBAI  NaN     NaN
freq   1       1       7       7       NaN     NaN
mean   NaN     NaN     NaN     NaN     16154.58   11004.54
std    NaN     NaN     NaN     NaN     3739.37    3711.18
min    NaN     NaN     NaN     NaN     11960.0       6630.0
25%    NaN     NaN     NaN     NaN     13472.5       7360.0
50%    NaN     NaN     NaN     NaN     14270.0       10760.0
75%    NaN     NaN     NaN     NaN     19238.75      14225.0
max    NaN     NaN     NaN     NaN     23280.0       16070.0
```

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
```

Recode a Categorical Variable – replace()

One data manipulation task that you need to do in pretty much any data analysis is recode data. It's almost never the case that the data are set up exactly the way you need them for your analysis.

Let's recode Location 'MUMBAI' as 1 and 'DELHI' as 2.

```
salary_data.Location.replace(to_replace=['MUMBAI','DELHI'],value=[1,2],
inplace=True)
salary_data.head(n=3)
```

- ❑ **replace()** can be used to replace any part of a dataframe with another value.
- ❑ **to_replace=** takes the value(s) to be replaced.
- ❑ **value=** takes the value(s) they need to be replaced with.
- ❑ **inplace= True** modifies the data directly i.e. you don't have to save the result back into salary_data.

```
# Output
```

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

Make a note of this syntax. It's great for recoding within Python programs.

Recode a Continuous Variable into Categorical Variable – cut()

Categorise the employees on the basis of their ba in three categories, namely, Low, Medium and High. Converting a continuous variable to categorical is called binning. Pandas makes it efficient to bin variables through the `pd.cut()` function:

```
ba_labels = ['low','medium','high']
bins = [0,14000,19000,24000]
salary_data['Category'] = pd.cut(salary_data['basic_allowance'], bins,
labels=ba_labels)
salary_data.head()
```

```
# Output
```

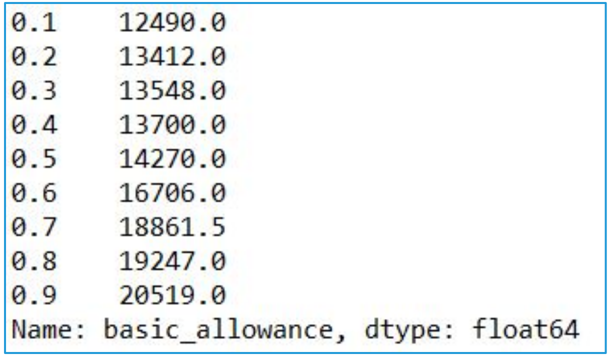
```
  First_Name Last_Name Grade  Location  basic_allowance       ms  newvariable
\
0       Alan     Brown   GR1         2            17990  16070.0       899.50
1     Agatha  Williams   GR2         1            12390   6630.0       619.50
2     Rajesh     Kolte   GR1         1            19250  14960.0       962.50
3      Ameet    Mishra   GR2         2            14780   9300.0       739.00
4       Neha       Rao   GR1         1            19235  15200.0       961.75

  Category
0   medium
1      low
2     high
3   medium
4     high
```

Create a Decile Object – quantile()

Interpret the deciles of **ba**

```
Decile_ba=salary_data['basic_allowance'].quantile(q=[0.1,0.2,0.3,0.4,0
.5,0.6,0.7,0.8,0.9])
Decile_ba
```

```
# Output
```



```
0.1     12490.0
0.2     13412.0
0.3     13548.0
0.4     13700.0
0.5     14270.0
0.6     16706.0
0.7     18861.5
0.8     19247.0
0.9     20519.0
Name: basic_allowance, dtype: float64
```

Interpretation:

- The first decile is 10%, implies that one-tenth of the ‘**basic_allowance**’ fall below or equal to 12490, and the remaining nine-tenth fall above 12490. 14270 is the median, thus half of the employees’ **‘basic_allowance’** is

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

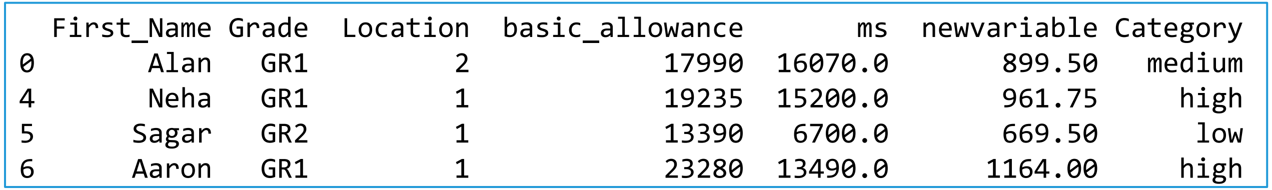
We can remove unwanted rows from our data by using their index nos.

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
```

Quick Recap

In this session, we learnt how to check data features in Python and why we should do it.

Dimensions and Variable Names

- `shape` returns the count of rows and columns
- `list(data)` returns variable names or column names.

Compact Internal Structure, Levels of Categorical Variable and Size of an Object

- `info()` returns many useful pieces of information like class of the object, data type of each column.
- `column.cat.categories()` returns the value of the levels of the object
- `memory_usage()` returns the size of an object in bytes.

Check the missing values (if any), First and Last n Rows

- `isnull()` returns a Boolean dataframe telling whether the data has missing values or not.
- `head()` returns the first n rows of data.
- `tail()` returns the last n rows of data.

Summarise Data

- `describe()` summarises the data based on the type of variable it contains.

Quick Recap

In this session, we learnt how to modify data in different ways.

Change Variable Names	• `rename()` can be used to change names of single or multiple variables
Derive a New Variable	• `assign()` adds a new variable in the dataframe
Recoding	• `replace()` or `cut()` let you change the content of data
Create a Decile Object	• `quantile()` is used for creating a decile object
Remove Variables or Rows	• `drop()` is used to remove unwanted columns or rows from the data, either by specifying the variable names or using index