In the last class, we learnt about the Pandas series. Now in this lesson, we will learn about Pandas DataFrame which is a collection of Pandas series. In other words, a Pandas DataFrame is a two-dimensional array.

```
#Creating a dataframe from list

l1=[1,2,4,6,7]

import pandas as pd
d1=pd.DataFrame(l1)
d1
```

```
S1=pd.Series(l1)
S1
```

```
l2=[[1,2,4,6,7],
    [1,2,4,6,7],
    [1,2,4,6,7],
    [1,2,4,6,7],
    [1,2,4,6,7]]

d2=pd.DataFrame(l2)
d2
```

```
#Creating a dataframe from array
import numpy as np
ar=np.ones((10,6), dtype=int)

d3=pd.DataFrame(ar)
d3
```

```
#Creating a dataframe from dictionary
di={ 'a':'gh',
    'b':'abc',
     'c':'bcd',
     'd':'gjk',
     'e': 'fgk'}
di
```

```
dp = pd.DataFrame(list(di.items()), columns=['Key', 'Value'])
dp
```

```
di={ 1:['gh', 1, 'gg', 'as'],
    2:['abc', 'bnn','okk', 0],
     3:np.nan,
     4:['gjk', 'kkk',1,4],
     5: np.nan}
di
```

```
d4=pd.DataFrame(di)
d4
```

```
d4.isnull()
```

```
d4.isnull().sum()
```

Activity 1: Loading A CSV File^^

Generally, we store data in different files such as text (`txt` format) file, comma-separated value (`csv` format) file, tab-separated value (`tsv` format) file etc. We can read the contents of these files through Python.

A comma-separated value (`csv`) file is used most commonly to store data.

To load or read the contents of a `csv` file, we can use the `read_csv()` function in Pandas. The data is read in the form of a two-dimensional array called a **Pandas DataFrame**.

As an input to the `read_csv()` function, we need to provide the full-location of the `csv` file that we wish to read in the string format. The file that we wish to read is stored in cloud storage.

Here's the link to the file.

File in G-Drive

In the above link, `exoTrain.csv` is the name of the `csv` file. We will create Pandas DataFrame using this file.

```
# Connect to the drive
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
cd/content/gdrive/My Drive
```

```
/content/gdrive/My Drive
```

```
#Read a 'csv' file using the 'read_csv()' function. Also, display the first 5 rows of the DataFrame using the 'head()' funct
# First of all we have to import the Pandas module with pd as an alias (or nickname).
import pandas as pd
df=pd.read_csv('cancer.csv')
df
```

```
import pandas as pd
```

```
# Information about a dataframe
df.describe()
```

```
#  Read the 'exoTrain.csv' file and display its first 5 rows using the 'head()' function.
df.head(20)
```

So, there are 5087 rows and 3198 columns in the `exo_train_df` DataFrame.

Activity 2: Check For The Missing Values^

In most of the cases, we do not get complete datasets. They either have some values missing from the rows and columns or they do not have standardized values.

For example: If there is a date column in a dataset, then there is a huge chance that some of the dates are entered in the `DD-MM-YYYY` format, some in the `MM-DD-YYYY` format and so on.

So, before going ahead with the analysis, it is a good idea to check whether the dataset has any missing values.

```
# Check for the missing values using the 'isnull()' function.
df.isnull()
```

To check for missing values in a DataFrame, use `isnull()` function. If a DataFrame has a missing value, then the `isnull()` function will return `True` else it will return `False`.

As you can see, the `isnull()` function has returned the DataFrame with a lot of `False` values as an output. There are $5087 \times 3198 = 16268226$ values in the DataFrame. It is not feasible to check so many values manually. So, we need a better approach to check for missing values.

We can call the `sum()` function on the `exo_train_df.isnull()` statement. It will return the sum of `True` values for every column in a DataFrame. If a column does not have any missing values, then it will return `0` else a number greater than `0`.

```
#  Use the 'sum()' function to find the total number of True values in each column.
df.isnull().sum()
```

```
d4.isnull().sum()
```

We can see that a lot of columns have `0` missing values. But still, we cannot manually see whether all the columns have missing values or not because the list of columns is too long to be seen in this notebook. There are `3198` columns to search. If there were very few columns, then we would not need to go any further to check for the missing values.

The `columns` keyword returns an array of all the columns in a DataFrame. To get a column from a DataFrame just, write the name of the column inside the square brackets in the single or double inverted quote after writing the name of the DataFrame. For example: If you want to get the values of `FLUX.1` column from the `exo_train_df`, then write `exo_train_df['FLUX.1']`.

```
# dealing with null values using dropna() and fillna()
df['FLUX.1'].mean()
```

```
m=df["FLUX.1"].mean()
```

```
df.fillna(m)
```

```
df.dropna()
```

```
df.iloc
```

```
# View all the columns in the 'exo_train_df' DataFrame.
df.columns
```

We, again, need a better approach. We will create a variable called `num_missing_values` to store the total number of values that are missing. Then, we will iterate through each column and within each column, we will iterate through each item to check for the missing values. If the `isnull()` function for a column returns `True`, then we will increase the value of the `num_missing_values` by `1` else we will not do anything.

```
# Get the values of the 'FLUX.1' column from a DataFrame.
df['LABEL']
```

```
df['FLUX.56']
```

Activity 3: Slicing A DataFrame Using The `iloc[]` Function^^^

Inside, the `iloc[]` function, the digit `0` indicates the first row (located at index `0`) of the `exo_train_df` DataFrame and the colon (`:`) symbol denotes *collect all the values from the first column till the last column*, i.e., all the columns starting from `LABEL` to `FLUX.3197` columns.

Syntax:

`dataframe_name.iloc[row_position_start : row_position_end, column_position_start : column_position_end]`

In this syntax:

- `row_position_start` denotes the position of the row in the DataFrame **starting** from whose values you want to take in the new Pandas series or DataFrame.
- `row_position_end` denotes the position of the row in the DataFrame till whose values you want to take in the new Pandas series or DataFrame.
- `column_position_start` denotes the position of the column in the DataFrame **starting** from whose values you want to take in the new Pandas series or DataFrame.
- `column_position_end` denotes the position of the column in the DataFrame till whose values you want to take in the new Pandas series or DataFrame.

You can verify manually whether we have extracted the values from the first row or not by viewing the first 5 rows of the DataFrame using the `head()` function.

Let's create a Pandas series for the first star in the `exo_train_df`. Let's store the series in a variable called `star_0`. To do this, we need to use the `iloc` function.

```
df
```

```
df.iloc[0 :1 , : ]
```

```
c1=df.iloc[ : ,0:1 ]
c1
```

```
type(c1)
```

```
r5082=df.iloc[5082 :5083 , :5 ]
```

```
r5082
```

```
r0=df.iloc[0 :1 ,  :  ]
r0
```

```
r0=df.iloc[19 : ,  :  ]
r0
```

```
:# Create a Pandas series from a Pandas DataFrame using the 'iloc[]' function.
```