Importing Pandas

In [1]:
```
import numpy as np

import pandas as pd
```

Creation of DataFrames

Creation from a python object

In [2]:
```
dict1 = {
    "name": ["Pratham", "Utkarsh", "Shreyansh"],
    "marks": [92, 90, 91],
    "city": ["Kanpur", "Sitapur", "Lucknow"]
}

df = pd.DataFrame(dict1)

df
```

Out[2]:

	name	marks	city
0	Pratham	92	Kanpur
1	Utkarsh	90	Sitapur
2	Shreyansh	91	Lucknow

Creation from CSVs

In [3]:
```
dafr = pd.read_csv('eg2.csv')
```

Exporting dataframe to CSV

In [4]:
```
df.to_csv('eg1.csv')

df.to_csv('eg3.csv', index = False)
# this one creates without any index coloumn
```

Basic Functions

In [5]:
```
dafr.head(2)
```

Out[5]:

	name	marks	city
0	Pratham	95	Kanpur
1	Utkarsh	90	Sitapur

In [6]:
```
dafr.tail(2)
```

Out[6]:

	name	marks	city
1	Utkarsh	90	Sitapur
2	Shreyansh	91	Lucknow

In [7]:
```
dafr.describe()
```

Out[7]:

	marks
count	3.000000
mean	92.000000
std	2.645751
min	90.000000
25%	90.500000
50%	91.000000
75%	93.000000
max	95.000000

In [8]:
```
# changing values at cells

dafr["marks"][0] = 96

dafr.to_csv('eg3.csv') # flushing the change into csv

dafr
```

```
C:\Users\prath\AppData\Local\Temp\ipykernel_18520\664463831.py:3: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  dafr["marks"][0] = 96
```

Out[8]:

	name	marks	city
0	Pratham	96	Kanpur
1	Utkarsh	90	Sitapur
2	Shreyansh	91	Lucknow

In [9]:
```
dafr.index = ["zeroth", "first", "second"]

dafr
```

Out[9]:

	name	marks	city
zeroth	Pratham	96	Kanpur
first	Utkarsh	90	Sitapur
second	Shreyansh	91	Lucknow

In [10]:
```
dafr.to_numpy()
# changes the pandas DataFrame to numpy array/object
```

Out[10]:
```
array([['Pratham', 96, 'Kanpur'],
       ['Utkarsh', 90, 'Sitapur'],
       ['Shreyansh', 91, 'Lucknow']], dtype=object)
```

Pandas Data Structure

Series - Series refers to one dimensional data structures. May it be a single row or a coloumn. They always contain data of similar datatype.

DataFrame - DataFrame refers to the whole two dimensional structure containing multiple rows and coloumns. Thay contain data of many different datatypes.

In [11]:
```
ser = pd.Series(np.random.rand(5))

print(type(ser))

ser
```

```
<class 'pandas.core.series.Series'>
```

Out[11]:
```
0    0.066531
1    0.674875
2    0.560481
3    0.890320
4    0.442227
dtype: float64
```

In [12]:
```
newdf = pd.DataFrame(np.random.rand(200,5), index = np.arange(200))

print(type(newdf))
# thus we see that this is a DataFrame data structure

print(type(newdf[0]))
# thus we can see that a coloumn is a series data structure

newdf
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

Out[12]:

	0	1	2	3	4
0	0.604004	0.377366	0.405540	0.237006	0.025481
1	0.302459	0.049076	0.908400	0.396762	0.455881
2	0.426805	0.865095	0.431399	0.201936	0.784481
3	0.712978	0.910919	0.934989	0.402459	0.942392
4	0.047410	0.647865	0.147949	0.999761	0.167011
...	...	...	...	...	...
195	0.436273	0.598410	0.121860	0.116263	0.049573
196	0.786359	0.477013	0.765064	0.892292	0.857685
197	0.194662	0.324571	0.144850	0.157605	0.506618
198	0.719622	0.707832	0.947345	0.128997	0.516320
199	0.822657	0.352411	0.361754	0.776716	0.580889

200 rows × 5 columns

Shallow Copy (View) & Deep Copy of a DataFrame

When we make changes to a copied (directly assigned) DataFrame, they are reflected into the original DataFrame too.

In [13]:
```
newdf2 = newdf
# making a copy

newdf2[0][0] = 12345
# changing cell value of copied DataFrame

newdf.head()
```

Out[13]:

	0	1	2	3	4
0	12345.000000	0.377366	0.405540	0.237006	0.025481
1	0.302459	0.049076	0.908400	0.396762	0.455881
2	0.426805	0.865095	0.431399	0.201936	0.784481
3	0.712978	0.910919	0.934989	0.402459	0.942392
4	0.047410	0.647865	0.147949	0.999761	0.167011

For a Deep copy we need to use the .copy() function

In [14]:
```
newdf2 = newdf.copy()

newdf2[0][0] = 45678
# changing the cell value of copied DataFrame

newdf.head()
```

Out[14]:

	0	1	2	3	4
0	12345.000000	0.377366	0.405540	0.237006	0.025481
1	0.302459	0.049076	0.908400	0.396762	0.455881
2	0.426805	0.865095	0.431399	0.201936	0.784481
3	0.712978	0.910919	0.934989	0.402459	0.942392
4	0.047410	0.647865	0.147949	0.999761	0.167011

As we see that changing the cell value like dafr[0][0] = 546 gives a warning of slicing and cannot determine when to display view or copy thus we use .loc() function to change the value of cell

In [15]:
```
newdf.loc[0, 0] = 654
# changing the value at (0, 0) cell of newdf DataFrame

newdf.head()
```

Out[15]:

	0	1	2	3	4
0	654.000000	0.377366	0.405540	0.237006	0.025481
1	0.302459	0.049076	0.908400	0.396762	0.455881
2	0.426805	0.865095	0.431399	0.201936	0.784481
3	0.712978	0.910919	0.934989	0.402459	0.942392
4	0.047410	0.647865	0.147949	0.999761	0.167011

Dropping a Column or a Row from a DataFrame

We can use the .drop() function to delete a particular column or a row from the DataFrame

In [16]:
```
newdf = newdf.drop(4, axis = 1)
# using axis = 1 for coloumns
# also axis = 0 is used for deleting rows

newdf.head(3)
```

Out[16]:

	0	1	2	3
0	654.000000	0.377366	0.405540	0.237006
1	0.302459	0.049076	0.908400	0.396762
2	0.426805	0.865095	0.431399	0.201936

Running Queries on DataFrames

We can query a DataFrame in the same way as we query a database

In [17]:
```
# selecting a particular cell
print(newdf.loc[0, 0])
print()

# selecting particular rows and particular columns

print(newdf.loc[[0,1,2], [0,1]])

print()

# selecting particular rows and all columns

print(newdf.loc[[1, 3, 5] , :])

print()

# selecting all rows and particular columns

print(newdf.loc[: , [1, 3]])
```

```
654.0

            0         1
0  654.000000  0.377366
1    0.302459  0.049076
2    0.426805  0.865095

          0         1         2         3
1  0.302459  0.049076  0.908400  0.396762
3  0.712978  0.910919  0.934989  0.402459
5  0.336489  0.930926  0.311172  0.033386

            1         3
0    0.377366  0.237006
1    0.049076  0.396762
2    0.865095  0.201936
3    0.910919  0.402459
4    0.647865  0.999761
..        ...       ...
195  0.598410  0.116263
196  0.477013  0.892292
197  0.324571  0.157605
198  0.707832  0.128997
199  0.352411  0.776716

[200 rows x 2 columns]
```

Running complex queries:

In [18]:
```
print(newdf.loc[newdf[0]<0.8])

print()

print(newdf.loc[(newdf[0]<0.8) & (newdf[1]>0.6)])
```

```
            0         1         2         3
1    0.302459  0.049076  0.908400  0.396762
2    0.426805  0.865095  0.431399  0.201936
3    0.712978  0.910919  0.934989  0.402459
4    0.047410  0.647865  0.147949  0.999761
5    0.336489  0.930926  0.311172  0.033386
..        ...       ...       ...       ...
194  0.344229  0.450295  0.093813  0.693153
195  0.436273  0.598410  0.121860  0.116263
196  0.786359  0.477013  0.765064  0.892292
197  0.194662  0.324571  0.144850  0.157605
198  0.719622  0.707832  0.947345  0.128997

[160 rows x 4 columns]

            0         1         2         3
2    0.426805  0.865095  0.431399  0.201936
3    0.712978  0.910919  0.934989  0.402459
4    0.047410  0.647865  0.147949  0.999761
5    0.336489  0.930926  0.311172  0.033386
6    0.149721  0.688935  0.065697  0.643000
..        ...       ...       ...       ...
185  0.191025  0.859240  0.199273  0.805456
189  0.336541  0.972459  0.671848  0.478052
191  0.499460  0.952349  0.329848  0.650840
192  0.473241  0.749687  0.360812  0.549596
198  0.719622  0.707832  0.947345  0.128997

[72 rows x 4 columns]
```

iloc : a special function

When we use .loc, we have to clearly specify our column/row names but while using iloc we can just use the indexing of columns or rows. for eg.

In [19]:
```
print(dafr.loc['zeroth', 'name'])

print()

print(dafr.iloc[0, 0])
```

```
Pratham

Pratham
```

inplace : a special parameter

For example if we have to delete a column from a dataframe and modify the changes into the original dataframe, we had to reassign the changed dataframe. But using the inplace attribute we can do so without reassigning the change.

In [20]:
```
newdf.drop([2, 3], axis = 1, inplace = True)
newdf
```

Out[20]:

	0	1
0	654.000000	0.377366
1	0.302459	0.049076
2	0.426805	0.865095
3	0.712978	0.910919
4	0.047410	0.647865
...	...	...
195	0.436273	0.598410
196	0.786359	0.477013
197	0.194662	0.324571
198	0.719622	0.707832
199	0.822657	0.352411

200 rows × 2 columns

In [21]:
```
newdf.drop([2, 3], axis = 0, inplace = True)
newdf.head()
```

Out[21]:

	0	1
0	654.000000	0.377366
1	0.302459	0.049076
4	0.047410	0.647865
5	0.336489	0.930926
6	0.149721	0.688935

In [22]:
```
# for resetting indices:
# we have taken 'drop = True' to avoid an uneccessary column named 'index'
newdf.reset_index(drop = True, inplace = True)

newdf.head()
```

Out[22]:

	0	1
0	654.000000	0.377366
1	0.302459	0.049076
2	0.047410	0.647865
3	0.336489	0.930926
4	0.149721	0.688935