Introduction to Pandas

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

Pandas is a core library for data scientists and data analysts. It comes very handy for data manipulation and analysis specially for large data sets as it works faster than numpy.

Using Pandas, we can:

- Load datasets from various sources
- Prepare dataset
- Transform or manipulate data
- Analyze data
- Save data to various sources

In [1]:

```
# Importing numpy and pandas
import numpy as np
import pandas as pd
```

Data Structures in Pandas

- **Series** is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index
- **DataFrame** is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object.

Pandas Series (1-D Data)

- **pd.Series()** creates an empty series
- **Series = pd.series(list)** can be used to create a pandas series
- **Series = pd.series(list, index =[i1,i2,i3....])** can be used to create a pandas series

In [2]:

```
series1 = pd.Series([2,4,5,6,10])
print(series1)
```

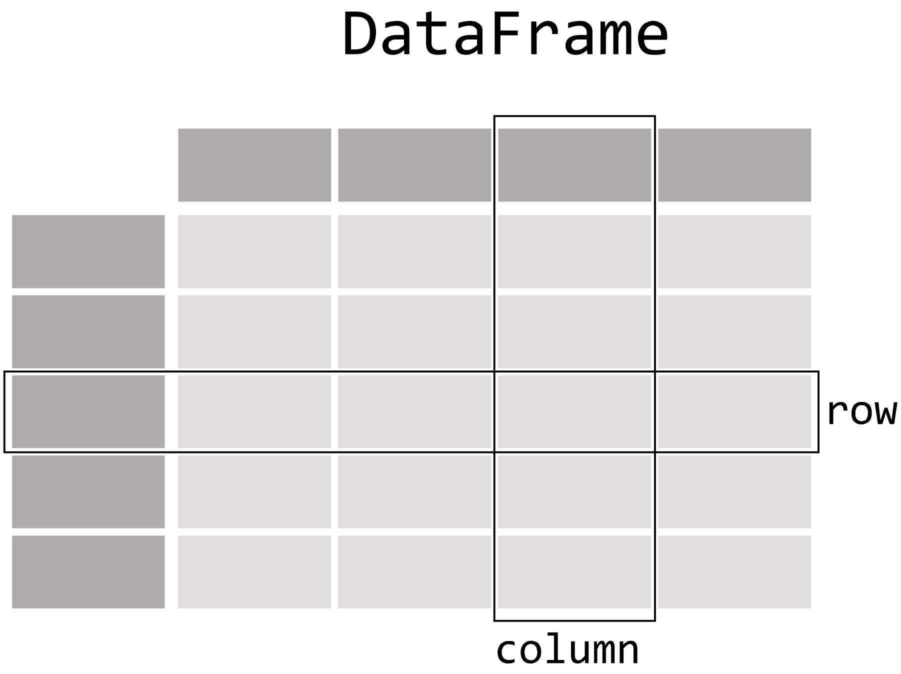
```
0     2
1     4
2     5
3     6
4    10
dtype: int64
```

```
In [3]: series2 = pd.Series([2,3,6,10,12], index=["c1", "c2", "c3", "c4", "c5"])
        print(series2)
```

```
c1     2
c2     3
c3     6
c4    10
c5    12
dtype: int64
```

Pandas DataFrame

- DataFrame is a 2 Dimensional data structure with labeled axis (rows and columns)

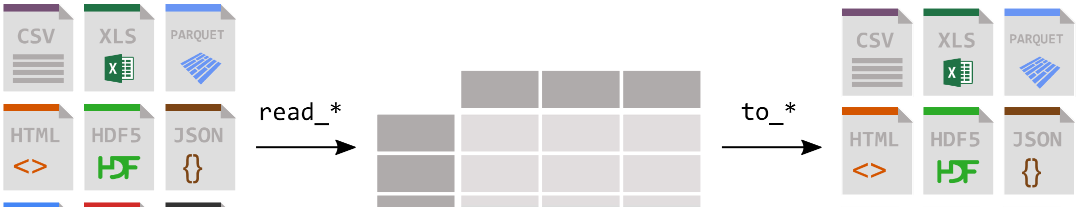


1. Creating dataframe from a numpy array
   - **pd.DataFrame(np_array, columns = [C1, C2, C3....Cn])**. Ensure that columns are equal to columns of array
2. Creating dataframe from dictionary
   - **Pd.DataFrame(dict)**
3. Import csv or other file and transform to dataframe
   - **pd.read_csv(file_location)**
   - file_location can be a URL(online location) or local location(on device)
   - path, name, extension etc can have / or \ in accessing the **local** location
   - We can give just file_name instead of complete path, in case the file is located in working directory
   - Pandas supports different file formats (csv, xls, json etc) with which read function changes

Read from File and write data to a File

- We have few datasets already uploaded here: https://github.com/mwaskom/seaborn-data (https://github.com/mwaskom/seaborn-data)
- use **read_filetype** to read in dataframe
- Use **to_datatype** to export to particular file type

Note: We can assign **None** if we need to keep the value empty



In [4]:

```
np.random.randn(8,4)
```

Out[4]:

```
array([[-0.04041735, -2.01211366,  0.36182679,  0.27376358],
       [ 0.69001521, -0.29038747,  0.30274573, -0.52566629],
       [-0.68316948,  1.3726267 ,  1.65787137,  1.40755173],
       [ 0.91913009,  0.20063835, -2.80409365,  0.0551908 ],
       [ 0.93875592, -2.5029262 ,  0.89051904, -0.96766324],
       [ 0.32206087,  0.94768801,  1.91183752, -1.5380036 ],
       [-1.34972299, -0.91602981,  0.47735877,  2.03213196],
       [ 2.31807617, -1.71763333, -0.79100795,  0.43262556]])
```

In [5]:

```
#From a numpy array
pd_dataframe = pd.DataFrame(np.random.randn(8,4), columns= ["A", "B", "C", "D"
print("dataframe from numpy array:\n", pd_dataframe)
```

```
dataframe from numpy array:
           A         B         C         D
0 -0.995028  1.244043  1.549950  0.402268
1  1.688583  0.828868  0.573770 -0.350037
2 -0.238033 -2.338364  1.996000 -0.835012
3  0.092416  1.220599 -0.602394 -0.087543
4  0.241726  0.500355 -0.747160 -1.013116
5 -0.834689 -1.481522  1.473233  0.413467
6  1.257992 -0.273358  0.667742  0.190285
7 -1.109319 -0.020809  1.663369  0.105021
```

In [6]:

```
#From a dictionary
dict1 = {"Person Name": ["Ramu", "Raju", "Ravi", "Sheela"], "Age": [34,23,None
dummy_data = pd.DataFrame(dict1)
print("\ndataframe from dict:\n", dummy_data)
```

```

dataframe from dict:
   Person Name   Age  Gender  Weight
0        Ramu    34    Male      76
1        Raju    23    Male      45
2        Ravi  None    Male      82
3      Sheela     a  Female      61
```

© Sourabh Khatri

In [7]:
```
#From read_csv
titanic = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/r
#print("\ndataframe from read file:\n", titanic)
df1 =  pd.read_csv('C:\\Users\\srbhk\\Downloads\\Python\\titanic.csv')
# we can also use pd.read_csv('titanic.csv') if the file is in working directo
#print(df1)
```

Writing to a File

dataframe.to_filetype is used to export data

- **df.to_numpy()** converts the dataframe to numpy array.
- df.to_csv("file_location", header = False, index = True)
- e.g. titanic.to_csv("titanic.csv")
- **Note:** header and index are defined as true or false if we require them or not while exporting

In [8]:
```
pd_dataframe.to_numpy() #converts to numpy array
titanic.to_csv("titanic.csv", index=False)
# titanic.to_excel('titanic.xlsx')
```

In [9]:
```
import os
os.getcwd()
```

Out[9]:
```
'C:\\Users\\srbhk\\Downloads\\Python'
```

In [10]:
```
pwd
```

Out[10]:
```
'C:\\Users\\srbhk\\Downloads\\Python'
```

Renaming Columns

There are different ways to rename/assign column names:

1. Direct renaming all columns by assigning a list: **df.columns =[c1,c2,c3....Cn]**
2. Renaming selected columns with rename function: **df.rename(columns = {'old_Col':'New_Col'},inplace = True)**
   - We put inplace = True if we want this to be saved permanently
3. Using set_axis function: **df.set_axis([newColList], axis='columns', inplace=True)**
   - axis = 'index' will rename the indexes 0,1,2... to the newColList.
4. *Making column as index:* **df.set_index("Cn")**.
   - We can still use the default indexes 0,1...
   - It changes the shape of the dataframe, as the column defined as index is not counted
5. using add_prefix() and add_suffix() functions: **df.add_prefix('prefix')** or **df.add_suffix('suffix')**
   - Eg: If col name is hello,hi... it changes them to prefixhellosuffix, prefixhisuffix...etc.

6. Replace specific texts of column names: **df.columns.str.replace function(oldCol, NewCol)**

In [11]:

```
dummy_data
```

Out[11]:

	Person Name	Age	Gender	Weight
0	Ramu	34	Male	76
1	Raju	23	Male	45
2	Ravi	None	Male	82
3	Sheela	a	Female	61

In [12]:

```
dummy_data.columns = ['person_name','age','gender','weight']
print("the shape is:",dummy_data.shape)
dummy_data
```

```
the shape is: (4, 4)
```

Out[12]:

	person_name	age	gender	weight
0	Ramu	34	Male	76
1	Raju	23	Male	45
2	Ravi	None	Male	82
3	Sheela	a	Female	61

In [13]:

```
dummy_data = dummy_data.set_index("person_name")
print("the new shape is:",dummy_data.shape)
dummy_data
```

```
the new shape is: (4, 3)
```

Out[13]:

	age	gender	weight
person_name			
Ramu	34	Male	76
Raju	23	Male	45
Ravi	None	Male	82
Sheela	a	Female	61

In [14]:

```
titanic.columns #To check all columns. Discussed later
```

Out[14]:

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'],
      dtype='object')
```

In [15]:
```
titanic.rename(columns = {'survived':'P.Survived'},inplace = True) # Changes s
titanic.rename(columns = {'P.Survived':'survived'},inplace = True) # Changes P
```

Slicing and Manipulation using loc and iloc

- **iloc:** index based search
- **loc:** label based seach on a particular row and column like searching with keywords
- The loc() function is label based data selecting method which means that we have to pass the name of the row or column which we want to select.
    - **This method includes the last element of the range passed in it, unlike iloc()**.
    - loc() can accept the boolean data unlike iloc().
- The iloc() function is an indexed-based selecting method which means that we have to pass an integer index in the method to select a specific row/column.

In [16]:
```
titanic.head()
```

Out[16]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	c
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	

- sibsp: Number of Siblings/Spouses Aboard
- parch: Number of Parents/Children Aboard

In [17]:
```
titanic.iloc[1:3,4:8]
```

Out[17]:

	sibsp	parch	fare	embarked
1	1	0	71.2833	C
2	0	0	7.9250	S

```
In [18]: titanic.iloc[0]
```

```
Out[18]: survived                 0
         pclass                   3
         sex                   male
         age                   22.0
         sibsp                    1
         parch                    0
         fare                  7.25
         embarked                 S
         class                Third
         who                    man
         adult_male            True
         deck                   NaN
         embark_town    Southampton
         alive                   no
         alone                False
         Name: 0, dtype: object
```

```
In [19]: titanic.iloc[1]
```

```
Out[19]: survived               1
         pclass                 1
         sex               female
         age                 38.0
         sibsp                  1
         parch                  0
         fare             71.2833
         embarked               C
         class              First
         who                woman
         adult_male         False
         deck                   C
         embark_town    Cherbourg
         alive                yes
         alone              False
         Name: 1, dtype: object
```

```
In [20]: titanic.iloc[2:4]
```

Out[20]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	de
2	1	3	female	26.0	0	0	7.925	S	Third	woman	False	N
3	1	1	female	35.0	1	0	53.100	S	First	woman	False	

```
In [21]: titanic.iloc[:,1:3]
```

Out[21]:

	pclass	sex
0	3	male
1	1	female
2	3	female
3	1	female
4	3	male
...	...	...
886	2	male
887	1	female
888	3	female
889	1	male
890	3	male

891 rows × 2 columns

```
In [22]: titanic.iloc[2:4, 3:6]
```

Out[22]:

	age	sibsp	parch
2	26.0	0	0
3	35.0	1	0

```
In [23]: titanic.head() #to review head data
```

Out[23]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True

Picking up values exactly from the specified row and column (based on index)

- Syntax: **df.iloc[list, list]**
- We can keep any list empty or unordered and iloc will return data accordingly

In [24]:

```
titanic.iloc[[2,4],[3,6,2]]
```

Out[24]:

	age	fare	sex
2	26.0	7.925	female
4	35.0	8.050	male

In [25]:

```
dummy_data.loc["Ramu"]
```

Out[25]:

```
age          34
gender     Male
weight       76
Name: Ramu, dtype: object
```

In [26]:

```
dummy_data.loc["Sheela"]
```

Out[26]:

```
age            a
gender    Female
weight        61
Name: Sheela, dtype: object
```

In [27]:

```
dummy_data.iloc[0:2]
```

Out[27]:

	age	gender	weight
person_name			
Ramu	34	Male	76
Raju	23	Male	45

Exploring Data

- **df.info()** method allows us to learn the shape of object types of our data. The information contains the below:

    1. **RangeIndex:** Number of rows
    2. **Data columns:** Number of columns
    3. **column labels:**, Name of each column
    4. **column data types:** could be object, int64, int32 etc.
    5. **Non-Null Count:** the number of cells in each column (non-null values).
    6. **memory usage:**, Total memory usage
- **df.describe()** method gives us summary statistics for numerical columns in our DataFrame

In [28]:

```
dummy_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, Ramu to Sheela
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   age     3 non-null      object
 1   gender  4 non-null      object
 2   weight  4 non-null      int64
dtypes: int64(1), object(2)
memory usage: 300.0+ bytes
```

In [29]:

```
print("\nThe describe method returns:\n", dummy_data.describe())
```

```

The describe method returns:
           weight
count   4.000000
mean   66.000000
std    16.552945
min    45.000000
25%    57.000000
50%    68.500000
75%    77.500000
max    82.000000
```

Checking the data individually

- **df.shape** returns the shape of the dataframe i.e. (rows, columns)
- **df.dtypes** returns the datatype of each column
    - We can check datatype of individual column as **df['column'].dtype**
- **df.isnull()** checks if any value is null and returns true corrosponding to it
    - **df.isnull().sum()** can be used to get total number of null places in each column

In [30]:

```
print("Shape is:", dummy_data.shape)
print("\nDatatypes are:\n",dummy_data.dtypes)
dummy_data['age'].dtype
```

```
Shape is: (4, 3)

Datatypes are:
 age        object
gender    object
weight     int64
dtype: object
```

Out[30]:

```
dtype('O')
```

In [31]:
```
print(dummy_data.isnull())
print("\nTotal null places:\n",dummy_data.isnull().sum())
```

```
               age  gender  weight
person_name
Ramu         False   False   False
Raju         False   False   False
Ravi          True   False   False
Sheela       False   False   False

Total null places:
 age       1
gender    0
weight    0
dtype: int64
```

Retrieve Data/Rows/Columns

df.head(num) and **df.tail(num)** return the first and last rows of the dataframe. By default it returns 5 rows, we can give num to have the required number of rows.

In [32]:
```
print("First row is:\n",dummy_data.head(1))
print("\nFirst 5 rows are:\n:",dummy_data.head())
```

```
First row is:
             age gender  weight
person_name
Ramu         34   Male      76

First 5 rows are:
:               age  gender  weight
person_name
Ramu           34    Male      76
Raju           23    Male      45
Ravi         None    Male      82
Sheela          a  Female      61
```

In [33]:
```
print("Last row is:\n",dummy_data.tail(1))
print("\nLast 5 rows are:\n:",dummy_data.tail())
```

```
Last row is:
             age  gender  weight
person_name
Sheela        a  Female      61

Last 5 rows are:
:               age  gender  weight
person_name
Ramu           34    Male      76
Raju           23    Male      45
Ravi         None    Male      82
Sheela          a  Female      61
```

- **df.columns** returns the name of all the columns

    - Slicing can be done for name of columns too. Eg. **df.columns[from: to+1]**
- **df.col1** or **df[col1]** returns a pandas series with data of col1 with indexes
- Similarly, **df[[list]]** can be used to get multiple columns, where list will have col1, col2...

In [34]:
```
titanic.columns
```

Out[34]:
```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'],
      dtype='object')
```

In [35]:
```
titanic.columns[:5]
```

Out[35]:
```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp'], dtype='object')
```

In [36]:
```
titanic.columns[2:5]
```

Out[36]:
```
Index(['sex', 'age', 'sibsp'], dtype='object')
```

In [37]:
```
series_survived = titanic.survived # saves data in survived column to series_s
#or
series_survived = titanic["survived"]
print("The data stored in pandas series is:\n",series_survived)
type(series_survived)
```

```
The data stored in pandas series is:
 0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: survived, Length: 891, dtype: int64
```

Out[37]:
```
pandas.core.series.Series
```

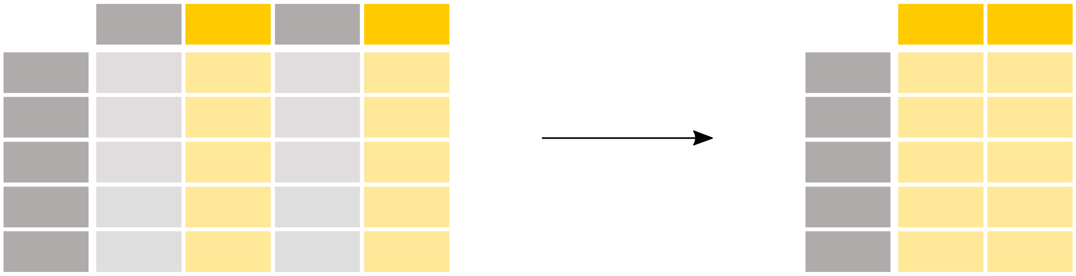
```
In [38]: titanic[['adult_male','survived']]
```

Out[38]:

	adult_male	survived
0	True	0
1	False	1
2	False	1
3	False	1
4	True	0
...	...	...
886	True	0
887	False	1
888	False	0
889	True	1
890	True	0

891 rows × 2 columns

Selecting a Subset of Columns from a Dataframe



- **df.["col"]** will return a Series
- **df.[["col"]]** will return a Dataframe
- **df.[['col1', 'col3', 'col2']]** will return a sub-Dataframe in required column order

In [39]:

```python
survived_df = titanic[["survived"]]
print(survived_df.head())
selected_df = titanic[["survived", "sex", "age", "fare", "class" ]]
selected_df.head()
```

```
   survived
0         0
1         1
2         1
3         1
4         0
```

Out[39]:

	survived	sex	age	fare	class
0	0	male	22.0	7.2500	Third
1	1	female	38.0	71.2833	First
2	1	female	26.0	7.9250	Third
3	1	female	35.0	53.1000	First
4	0	male	35.0	8.0500	Third

In [40]:

```python
# Below functions can be used to explore data as discussed above
print("Shape of original DF is: ",titanic.shape)
print("Shape of new DF is: ",selected_df.shape)
#selected_df.info()
#selected_df.dtypes
#selected_df['survived'].dtype
```

```
Shape of original DF is:  (891, 15)
Shape of new DF is:  (891, 5)
```