*Pandas :

→ used for working with data sets

→ analyzing, cleaning, exploring and manipulating data

→ Pandas – Panel data

Python data analysis

```
import Pandas as Pd

mydata = { 'cars' : [ 'BMW', 'volvo', 'Ford' ]
           'Passings' : [ 3, 7, 2 ]
         }

myvar = Pd.DataFrame (mydataset)

Print (myVar)
```

Series :

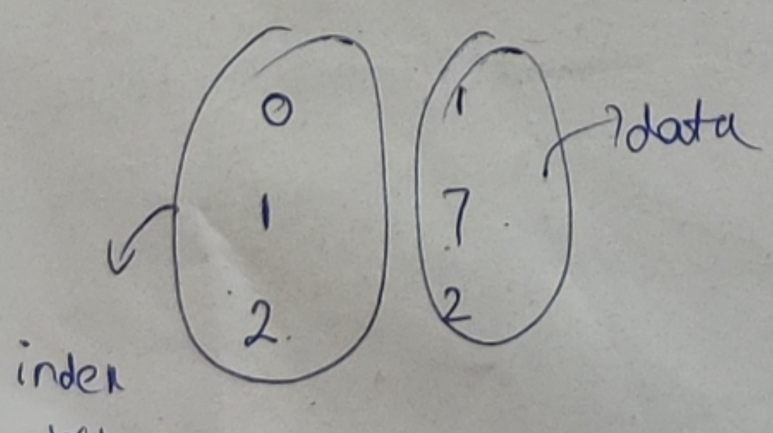
→ A Pandas series is like a column in a table.

→ It is a 1-D array holding data of any type.

```
a = [1, 7, 2]

myVar = Pd.Series (a)
```

if nothing specified the values are labelled with index number.



index argument is used to specify the labels seperated by ,

```
myVar = Pd.Series(a, index = ['x', 'y', 'z'])
```

we can access item by referring label

```
Print (myVar["y"])
```

```
myVar= Pd.series(
a = ["day1": 420, "day2": 320, "day3": 300]
```

↳ [No need of using index in myVar as they are already labelled.]

```
a = {"Fruits": ['Mango', 'Apple', 'grapes],
     "Vegetables": ['Potato', 'Tomato', 'Pumpkin']
}
myVar = Pd.DataFrame(a)
```

↳ series like a column, and it's a whole table

Print (df.loc[0]) ⟶ Prints the 1st row

↳ used to specify location of arow (can be one/more)

| we can also give index |

```
myvar = Pd.DataFrame(a, index = ["day1", "day2"])
```

```
Pd.read_csv('data.csv')
```

↳ loading a csv file into a DataFrame

```
df = Pd.read_csv ('data.csv')
Print ( df.to_string())
          ↳ to Print entire DataFrame
```

if we don't use to_string() it will only Print first 5 rows and last 5 rows.

[Print (Pd.options.display.max_rows)
↳ display max no of rows that can be shown]

to change the no of rows that can be displayed

```
Import Pandas as Pd

Pd.options.display.max_rows = 9999

df = Pd.read_csv ('data.csv')

Print df
```

CSV - comma seperated Values

JSON :

Big data sets are often stored/extracted in JSON.

```
df = Pd.read_json ('data.json')
        ↳ loading a json file
```

Viewing the data :

the head() method returns the headers and specific no of rows, starting from top.

```
Print (df.head(10))        Print (df.head())
  ↳ 10 rows                  ↳ 5 rows
```

the tail() method returns the headers and a specified no of rows

```
Print(df.tail())
```

the info() , give more information about data set.

```
Print(df.info())
```

* Cleaning Data :

Bad data
- Empty Cells
- Data in wrong format
- Wrong data
- duplicates

```
df = Pd.read_csv('data.csv')
new_df = df.Pnona()
Print(new_df.to_string())
```

→ [dropna() method return new Data frame, and will not change the original]

If we want to change original Dataframe use inplace = True

```
new_df = df.dropna(inplace = True)
```

↳ it will remove all rows containing NULL values from original DF.

To fill empty cell with a Value use

```
new_df = df.fillna(130, inPlace = True)
```

To replace Null Values from a column specify column name

```
df.fillna({"calories": 130}, inPlace = True)
```

```
df["calories"].median()  → median
df["calories"].mode()[0] → mode
```

```
x = df["calories"].mean()
df.fillna({"calories":x}, inPlace = True)
```
} replace null values in calories coloumn with mean

```
df['Date'] = pd.to_datetime(df['Date'], format = 'mixed')
```
↳ to convert the messy date into date format

```
df.dropna( subset = ['Date'], inPlace = True)
```

*Replacing Values :

```
df.loc [7, 'Duration'] = 45
```

to replace a large no of Values :

```
for x in df.index :
    if df.loc [x, "Duration"] > 120 :
        df.loc [x,"Duration"] =120
```
↳ If Value is greater than 120 replace with 120

```
for x in df.index:
    if df.loc[x,"Duration"] > 120 :
        df.drop(x, inPlace = True
```
↳ remove rows with no Value

```
df.drop_duplicates()
```
↳ remove row duplicate Values

```
df.duplicated()
```
↳ if any of the row has duplicates gives True/ false

* Working with missing data :

Pandas use different sentinal values to represent a missing value depending on data type.

`numpy.nan for Numpy datatypes`. The disadvantage is that the original datatype will be coerced to `np.float64` (or) `object`

Pd. Series([1,2], dtype = int np.int64). reindex([0,2,2])

*output :

```
0    1.0
1    2.0
2    NaN
dtype : float64
```

↳ "the i/p int64 is coerced to float64".

Pd. Series([True, False], dtype = np.bool_). reindex([0,1,2])

*output :

```
0    True
1    False
2    Nan
dtype : object
```

↳ "bool coerced to object".

NaT for Numpy np.datetime64, np.timedelta64 and PeriodDtype.

Pd. series([1,2], dtype = np.dtype("timedelta64[ns]")). reindex ([0,1,2])

```
0    0 days  00:00:00.000000001
1    0 days  00:11:11:000000002
2            NaT
```

↳ "There will be no coversion to another type"

And similar for above mentioned data types.

* NA for StringDtype, Int64Dtype and ArrowDtype

```
Pd.Series([1,2], dtype = "Int64").reindex([0,1,2])
```

O/P :

0	1
1	2
2	<NA>

```
Pd.Series([True, False], dtype = "boolean[pyarrow]").reinder
([0,1,2])
```

O/P :

0	True
1	False
2	<NA>

dtype = bool[pyarrow]

* To detect these missing values, use the isna() (or) notna() methods.

```
Ser = Pd.Series([Pd.Timestamp("2020-01-01"), Pd.NaT])
Pd.isna(ser)
```

O/P :

0	False	↗ Not null value
1	True	↘ Null value

dtype = bool

* isna() / notna() will also consider "None" a missing value.

* Equality compaisons between np.nan, NaT and NA don't act like none.

None == None

o/p : True

np.nan == np.nan

o/p : False

Pd.NaT == Pd.NaT

o/p : False

Pd.NA == Pd.NA

o/p : <NA>

* An experimental NA value (singleton) is available to represent scalar missing values.

For ex, When having missing values in a series with nullable integer dtype, it will use NA :

```
S = Pd.Series([1,2,None], dtype = "Int64")
S[2]
```

o/p : <NA>

```
S[2] is Pd.NA
```

o/p : True

* Logical operations :

for logical operations NA follows the rules of the three-valued logic.

for ex, for the logical "or" operation (|), if one of the operands is True, we already know the result is true, regardless of the other value.

True | False

o/p : True

True | Pd.NA

o/p : True

Pd.NA | True

o/p : True

*If the operands is false, the result depends on the value of the other operand. Hence Here NA Prologates.

False | True

o/p : True

False | Pd.NA

o/p : <NA>

False | False

o/p : False

*Similar thing happens in the "&" operaton.

False & True

o/p : False

False & Pd.NA

o/p : False

False & False

o/p : False

True & Pd.NA

o/p : <NA>