What is Pandas?

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

What Can Pandas Do?

Pandas gives you answers about the data. Like:

Is there a correlation between two or more columns?

What is average value?

Max value?

Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

DataFrames

Data sets in Pandas are usually multi-dimensional tables, called DataFrames.

Series is like a column, a DataFrame is the whole table.

In [9]:

```
import pandas

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pandas.DataFrame(mydataset)

print(myvar)
print()
print(myvar.loc[0])
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2

cars        BMW
passings      3
Name: 0, dtype: object
```

In [2]:

```
import pandas as pd

print(pd.__version__)
```

```
1.0.5
```

What is a Series?

A Pandas Series is like a column in a table.

It is a one-dimensional array holding data of any type.

In [3]:

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

myvar = pd.Series(a)

print(myvar)
```

```
0    1
1    7
2    2
dtype: int64
```

Labels

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

This label can be used to access a specified value.

In [4]:

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

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

```
x    1
y    7
z    2
dtype: int64
```

In [5]:

```
# Return the value of "y":
print(myvar["y"])
```

```
7
```

In [6]:

```
calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)
```

```
day1    420
day2    380
day3    390
dtype: int64
```

In [7]:

```
# Create a Series using only data from "day1" and "day2":

import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories, index = ["day1", "day2"])

print(myvar)
```

```
day1    420
day2    380
dtype: int64
```

Load Files Into a DataFrame

If your data sets are stored in a file, Pandas can load them into a DataFrame.

In [4]:

```
import pandas

mydataset = {
  'cars': ["BMW", "Volvo", "Ford", 'tesla', 'tata'],
  'passings': [3, 7, 2,4,5]
}

myvar = pandas.DataFrame(mydataset)
print(myvar)
myvar.to_csv('info.csv')
# myvar.to_csv('info.csv', index=False)  index pahije nsla tr
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
3  tesla         4
4   tata         5
```

In [5]:

```
myvar.head(2)     #fakt starting che 2 row dakhvel
```

Out[5]:

	cars	passings
0	BMW	3
1	Volvo	7

In [6]:

```
myvar.tail(2)   #fakt last che 2 row dakhvel
```

Out[6]:

	cars	passings
3	tesla	4
4	tata	5

In [7]:

```
myvar.describe()
```

Out[7]:

	passings
count	5.000000
mean	4.200000
std	1.923538
min	2.000000
25%	3.000000
50%	4.000000
75%	5.000000
max	7.000000

In [6]:

```
#read csv file
import pandas as pd
file= pd.read_csv('student.csv')
print(file)
```

```
     name  rollno  city
0   rushi       1  pune
1  keshav       2  pune
2   anuja       3  pune
3  shital       4  pune
```

In [7]:

```
file['rollno'][1]=12   #to change data in file
```

```
<ipython-input-7-cd732f05db09>:1: SettingWithCopyWarning:
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/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  file['rollno'][1]=12
```

In [8]:

```
print(file)
```

```
     name  rollno  city
0   rushi       1  pune
1  keshav      12  pune
2   anuja       3  pune
3  shital       4  pune
```

In [10]:

```
file.index=[10,11,12,13]      #change index
print(file)
```

```
      name  rollno  city
10   rushi       1  pune
11  keshav      12  pune
12   anuja       3  pune
13  shital       4  pune
```

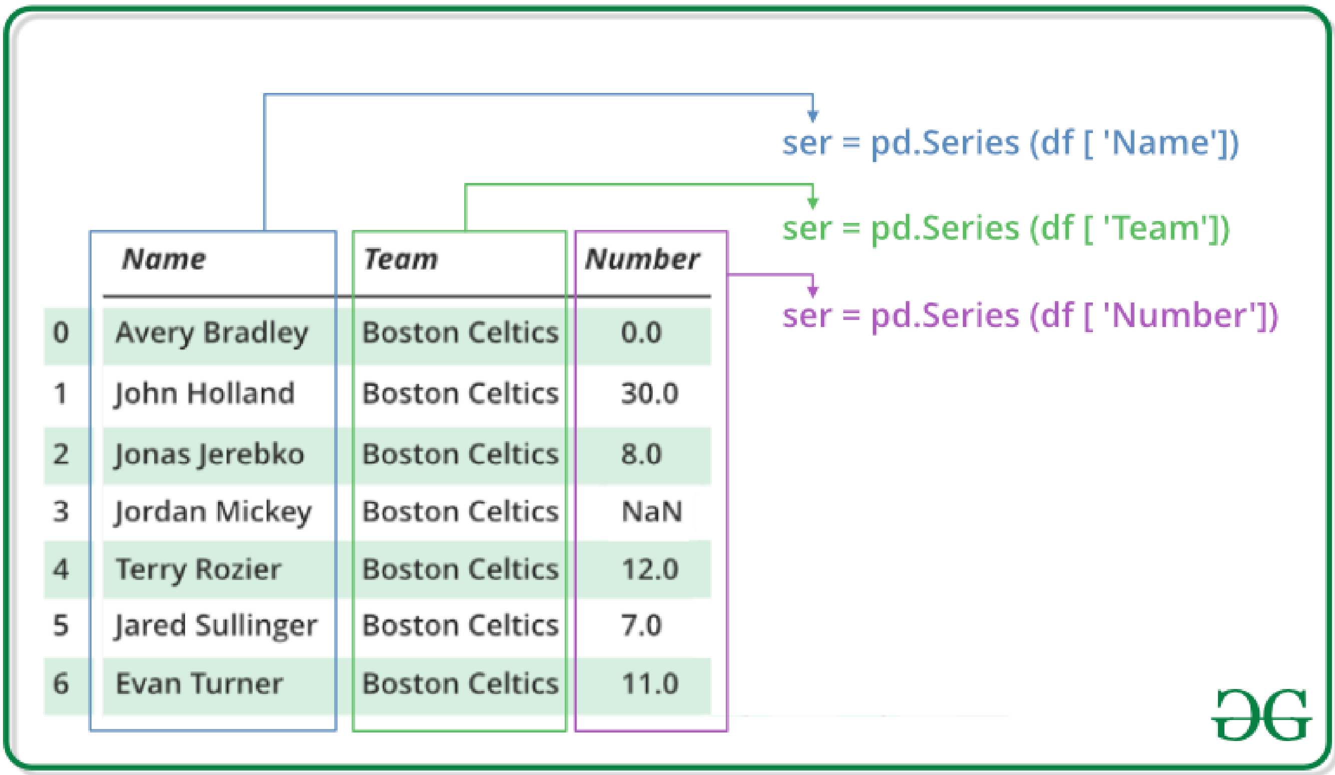
pandas data structure

--series

--dataframe

Series:

Pandas Series is a one-dimensional labelled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called indexes. Pandas Series is nothing but a column in an excel sheet. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

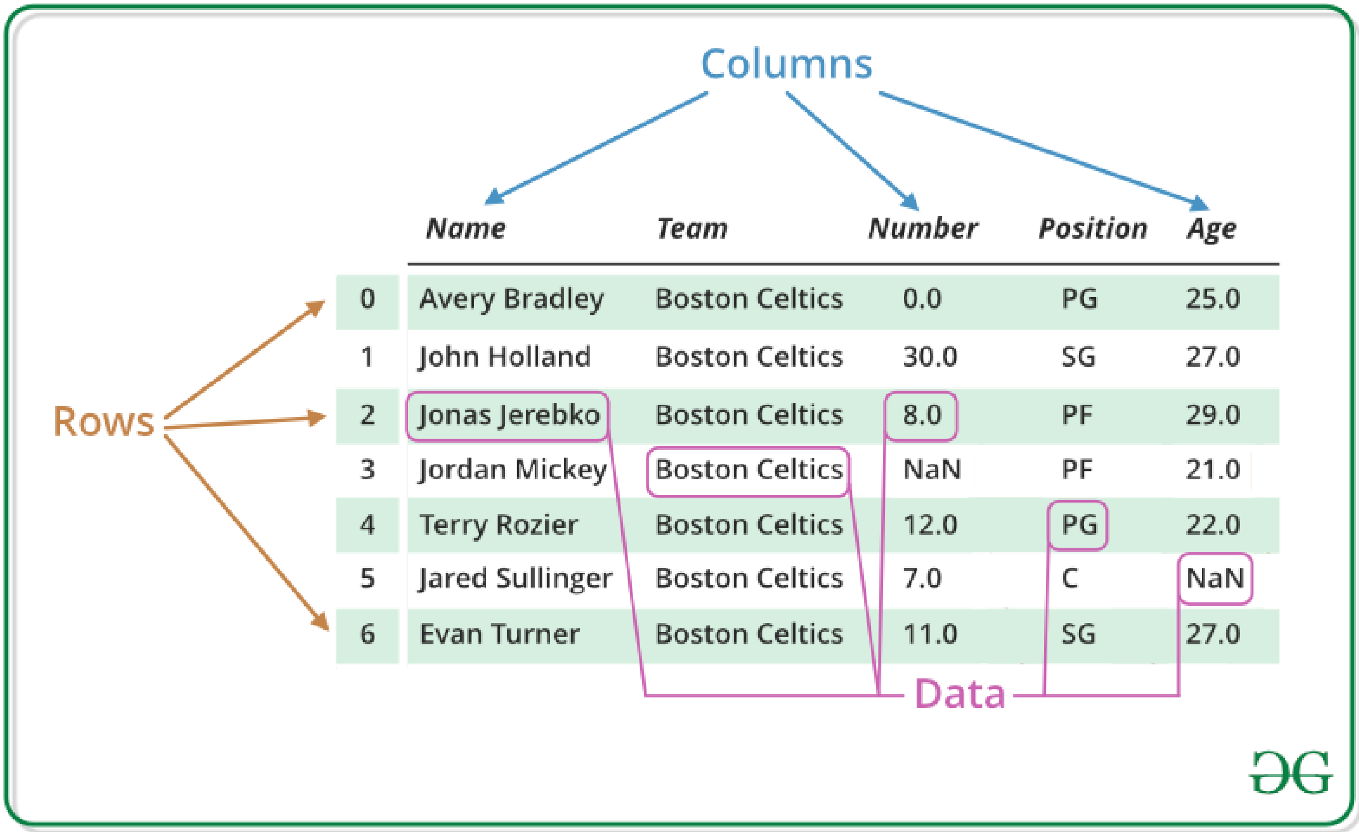


```
ser = pd.Series (df [ 'Name'])
ser = pd.Series (df [ 'Team'])
ser = pd.Series (df [ 'Number'])
```

	Name	Team	Number
0	Avery Bradley	Boston Celtics	0.0
1	John Holland	Boston Celtics	30.0
2	Jonas Jerebko	Boston Celtics	8.0
3	Jordan Mickey	Boston Celtics	NaN
4	Terry Rozier	Boston Celtics	12.0
5	Jared Sullinger	Boston Celtics	7.0
6	Evan Turner	Boston Celtics	11.0

DataFrame

Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

In [8]:

```
import numpy as np
import pandas as pd
newdf = pd.DataFrame(np.random.rand(334,5), index=np.arange(334))
newdf
```

Out[8]:

	0	1	2	3	4
0	0.475100	0.173122	0.261321	0.283373	0.710600
1	0.074084	0.895896	0.356004	0.670794	0.367154
2	0.401107	0.409255	0.629785	0.416491	0.632631
3	0.579149	0.065175	0.862828	0.453117	0.251587
4	0.139493	0.700041	0.446373	0.229521	0.774194
...	...	...	...	...	...
329	0.283477	0.349884	0.562985	0.588773	0.890677
330	0.798235	0.205785	0.410589	0.716287	0.429332
331	0.610896	0.282866	0.908540	0.695854	0.008176
332	0.544440	0.414596	0.121202	0.983670	0.750115
333	0.117343	0.038432	0.223937	0.645885	0.837589

334 rows × 5 columns

In [9]:

```
newdf.head()
```

Out[9]:

	0	1	2	3	4
0	0.475100	0.173122	0.261321	0.283373	0.710600
1	0.074084	0.895896	0.356004	0.670794	0.367154
2	0.401107	0.409255	0.629785	0.416491	0.632631
3	0.579149	0.065175	0.862828	0.453117	0.251587
4	0.139493	0.700041	0.446373	0.229521	0.774194

In [12]:

```
newdf.index
```

Out[12]:

```
Int64Index([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
            ...
            324, 325, 326, 327, 328, 329, 330, 331, 332, 333],
           dtype='int64', length=334)
```

In [13]:

```
newdf.columns
```

Out[13]:

```
RangeIndex(start=0, stop=5, step=1)
```

In [14]:

```
newdf.to_numpy()
```

Out[14]:

```
array([[0.47510047, 0.17312217, 0.26132107, 0.28337312, 0.71059975],
       [0.07408361, 0.89589573, 0.35600446, 0.67079358, 0.36715359],
       [0.4011072 , 0.4092549 , 0.62978493, 0.41649125, 0.6326309 ],
       ...,
       [0.61089631, 0.28286613, 0.90854008, 0.69585439, 0.00817552],
       [0.5444398 , 0.41459605, 0.12120208, 0.98367007, 0.75011469],
       [0.11734308, 0.03843185, 0.22393732, 0.64588467, 0.83758889]])
```

In [15]:

```
newdf.T    #row column madhe ani column row madhe convert hotil
```

Out[15]:

	0	1	2	3	4	5	6	7	8
0	0.475100	0.074084	0.401107	0.579149	0.139493	0.133298	0.768989	0.708546	0.808793
1	0.173122	0.895896	0.409255	0.065175	0.700041	0.901605	0.133485	0.111715	0.713773
2	0.261321	0.356004	0.629785	0.862828	0.446373	0.901748	0.657082	0.036265	0.008844
3	0.283373	0.670794	0.416491	0.453117	0.229521	0.895150	0.528441	0.528502	0.736985
4	0.710600	0.367154	0.632631	0.251587	0.774194	0.077758	0.024994	0.577201	0.809680

5 rows × 334 columns

In [16]:

```
newdf.sort_index(axis=0, ascending=False)     #reverse sort
```

Out[16]:

	0	1	2	3	4
333	0.117343	0.038432	0.223937	0.645885	0.837589
332	0.544440	0.414596	0.121202	0.983670	0.750115
331	0.610896	0.282866	0.908540	0.695854	0.008176
330	0.798235	0.205785	0.410589	0.716287	0.429332
329	0.283477	0.349884	0.562985	0.588773	0.890677
...	...	...	...	...	...
4	0.139493	0.700041	0.446373	0.229521	0.774194
3	0.579149	0.065175	0.862828	0.453117	0.251587
2	0.401107	0.409255	0.629785	0.416491	0.632631
1	0.074084	0.895896	0.356004	0.670794	0.367154
0	0.475100	0.173122	0.261321	0.283373	0.710600

334 rows × 5 columns

In [17]:

```
newdf[0]
```

Out[17]:

```
0      0.475100
1      0.074084
2      0.401107
3      0.579149
4      0.139493
         ...
329    0.283477
330    0.798235
331    0.610896
332    0.544440
333    0.117343
Name: 0, Length: 334, dtype: float64
```

In [18]:

```
type(newdf[0])
```

Out[18]:

```
pandas.core.series.Series
```

In [20]:

```
newdf2=newdf  # create view of newdf
newdf2
```

Out[20]:

	0	1	2	3	4
0	0.475100	0.173122	0.261321	0.283373	0.710600
1	0.074084	0.895896	0.356004	0.670794	0.367154
2	0.401107	0.409255	0.629785	0.416491	0.632631
3	0.579149	0.065175	0.862828	0.453117	0.251587
4	0.139493	0.700041	0.446373	0.229521	0.774194
...	...	...	...	...	...
329	0.283477	0.349884	0.562985	0.588773	0.890677
330	0.798235	0.205785	0.410589	0.716287	0.429332
331	0.610896	0.282866	0.908540	0.695854	0.008176
332	0.544440	0.414596	0.121202	0.983670	0.750115
333	0.117343	0.038432	0.223937	0.645885	0.837589

334 rows × 5 columns

In [24]:

```
newdf3=newdf.copy()      #create copy of newdf
newdf3
```

Out[24]:

	A	B	C	D	E
0	0.475100	0.173122	0.261321	0.283373	0.710600
1	0.074084	0.895896	0.356004	0.670794	0.367154
2	0.401107	0.409255	0.629785	0.416491	0.632631
3	0.579149	0.065175	0.862828	0.453117	0.251587
4	0.139493	0.700041	0.446373	0.229521	0.774194
...	...	...	...	...	...
329	0.283477	0.349884	0.562985	0.588773	0.890677
330	0.798235	0.205785	0.410589	0.716287	0.429332
331	0.610896	0.282866	0.908540	0.695854	0.008176
332	0.544440	0.414596	0.121202	0.983670	0.750115
333	0.117343	0.038432	0.223937	0.645885	0.837589

334 rows × 5 columns

In [23]:

```
newdf.columns=list("ABCDE") #label column name
```

In [25]:

```
newdf.loc[0,'A']=123
```

In [26]:

```
print(newdf.head())
```

```
            A         B         C         D         E
0  123.000000  0.173122  0.261321  0.283373  0.710600
1    0.074084  0.895896  0.356004  0.670794  0.367154
2    0.401107  0.409255  0.629785  0.416491  0.632631
3    0.579149  0.065175  0.862828  0.453117  0.251587
4    0.139493  0.700041  0.446373  0.229521  0.774194
```

In [28]:

```
newdf.loc[0,0]=12223     #ek aankhi column add hoil
print(newdf.head())
```

```
            A         B         C         D         E        0
0  123.000000  0.173122  0.261321  0.283373  0.710600  12223.0
1    0.074084  0.895896  0.356004  0.670794  0.367154      NaN
2    0.401107  0.409255  0.629785  0.416491  0.632631      NaN
3    0.579149  0.065175  0.862828  0.453117  0.251587      NaN
4    0.139493  0.700041  0.446373  0.229521  0.774194      NaN
```

In [31]:

```
newdf.drop(0 , axis=1)  #add zalela column delete karnyasathi.  row jr delete karaycha asla
```

Out[31]:

	A	B	C	D	E
0	123.000000	0.173122	0.261321	0.283373	0.710600
1	0.074084	0.895896	0.356004	0.670794	0.367154
2	0.401107	0.409255	0.629785	0.416491	0.632631
3	0.579149	0.065175	0.862828	0.453117	0.251587
4	0.139493	0.700041	0.446373	0.229521	0.774194
...	...	...	...	...	...
329	0.283477	0.349884	0.562985	0.588773	0.890677
330	0.798235	0.205785	0.410589	0.716287	0.429332
331	0.610896	0.282866	0.908540	0.695854	0.008176
332	0.544440	0.414596	0.121202	0.983670	0.750115
333	0.117343	0.038432	0.223937	0.645885	0.837589

334 rows × 5 columns

In [32]:

```
newdf.loc[[1,2], ["C", "D"]]     #fakt specific row ani column pahije aslyas
```

Out[32]:

	C	D
1	0.356004	0.670794
2	0.629785	0.416491

In [34]:

```
newdf.loc[:, ["C", "D"]]  #sarv row sathi
```

Out[34]:

	C	D
0	0.261321	0.283373
1	0.356004	0.670794
2	0.629785	0.416491
3	0.862828	0.453117
4	0.446373	0.229521
...	...	...
329	0.562985	0.588773
330	0.410589	0.716287
331	0.908540	0.695854
332	0.121202	0.983670
333	0.223937	0.645885

334 rows × 2 columns

In [35]:

```
newdf.loc[[1,2], :]    #sarv column sathi
```

Out[35]:

	A	B	C	D	E	0
1	0.074084	0.895896	0.356004	0.670794	0.367154	NaN
2	0.401107	0.409255	0.629785	0.416491	0.632631	NaN

In [36]:

```
newdf.loc[(newdf['A']<0.3)]   #column A madhe jya jya row madhe 0.3 peksha kami value ahe t
```

Out[36]:

	A	B	C	D	E	0
1	0.074084	0.895896	0.356004	0.670794	0.367154	NaN
4	0.139493	0.700041	0.446373	0.229521	0.774194	NaN
5	0.133298	0.901605	0.901748	0.895150	0.077758	NaN
14	0.075409	0.000197	0.514362	0.140561	0.811226	NaN
19	0.178522	0.421768	0.977016	0.097444	0.892860	NaN
...	...	...	...	...	...	...
318	0.015291	0.875448	0.185450	0.527804	0.704151	NaN
319	0.206724	0.145403	0.580917	0.082577	0.922378	NaN
327	0.275266	0.823185	0.728597	0.592526	0.550500	NaN
329	0.283477	0.349884	0.562985	0.588773	0.890677	NaN
333	0.117343	0.038432	0.223937	0.645885	0.837589	NaN

90 rows × 6 columns

In [37]:

```
newdf.loc[(newdf['A']<0.3) & (newdf['C']>0.1)]   #column A madhe jya jya row madhe 0.3 peks
```

Out[37]:

	A	B	C	D	E	0
1	0.074084	0.895896	0.356004	0.670794	0.367154	NaN
4	0.139493	0.700041	0.446373	0.229521	0.774194	NaN
5	0.133298	0.901605	0.901748	0.895150	0.077758	NaN
14	0.075409	0.000197	0.514362	0.140561	0.811226	NaN
19	0.178522	0.421768	0.977016	0.097444	0.892860	NaN
...	...	...	...	...	...	...
318	0.015291	0.875448	0.185450	0.527804	0.704151	NaN
319	0.206724	0.145403	0.580917	0.082577	0.922378	NaN
327	0.275266	0.823185	0.728597	0.592526	0.550500	NaN
329	0.283477	0.349884	0.562985	0.588773	0.890677	NaN
333	0.117343	0.038432	0.223937	0.645885	0.837589	NaN

85 rows × 6 columns

In [44]:

```python
newdf.drop(['A','B'], axis=1, inplace=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-44-b61296a29257> in <module>
----> 1 newdf.drop(['A','B'], axis=1, inplace=True)

~\anaconda3\lib\site-packages\pandas\core\frame.py in drop(self, labels, axi
s, index, columns, level, inplace, errors)
   3988                 weight  1.0     0.8
   3989         """
-> 3990         return super().drop(
   3991             labels=labels,
   3992             axis=axis,

~\anaconda3\lib\site-packages\pandas\core\generic.py in drop(self, labels, a
xis, index, columns, level, inplace, errors)
   3934         for axis, labels in axes.items():
   3935             if labels is not None:
-> 3936                 obj = obj._drop_axis(labels, axis, level=level, erro
rs=errors)
   3937 
   3938         if inplace:

~\anaconda3\lib\site-packages\pandas\core\generic.py in _drop_axis(self, lab
els, axis, level, errors)
   3968                 new_axis = axis.drop(labels, level=level, errors=err
ors)
   3969             else:
-> 3970                 new_axis = axis.drop(labels, errors=errors)
   3971             result = self.reindex(**{axis_name: new_axis})
   3972 

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in drop(self, labe
ls, errors)
   5016         if mask.any():
   5017             if errors != "ignore":
-> 5018                 raise KeyError(f"{labels[mask]} not found in axis")
   5019             indexer = indexer[~mask]
   5020         return self.delete(indexer)

KeyError: "['A' 'B'] not found in axis"
```

In [39]:

```
newdf  #note:aapn jr inplace true nhi kel tr to fakt sadyapurta change hoil manje fakt copy
# ex.aapn 0 coulun drop kela hota but to tri pn show krtoy krn aapn tevha inplace true nvte
```

Out[39]:

	C	D	E	0
0	0.261321	0.283373	0.710600	12223.0
1	0.356004	0.670794	0.367154	NaN
2	0.629785	0.416491	0.632631	NaN
3	0.862828	0.453117	0.251587	NaN
4	0.446373	0.229521	0.774194	NaN
...	...	...	...	...
329	0.562985	0.588773	0.890677	NaN
330	0.410589	0.716287	0.429332	NaN
331	0.908540	0.695854	0.008176	NaN
332	0.121202	0.983670	0.750115	NaN
333	0.223937	0.645885	0.837589	NaN

334 rows × 4 columns

In [40]:

```
newdf
```

Out[40]:

	C	D	E	0
0	0.261321	0.283373	0.710600	12223.0
1	0.356004	0.670794	0.367154	NaN
2	0.629785	0.416491	0.632631	NaN
3	0.862828	0.453117	0.251587	NaN
4	0.446373	0.229521	0.774194	NaN
...	...	...	...	...
329	0.562985	0.588773	0.890677	NaN
330	0.410589	0.716287	0.429332	NaN
331	0.908540	0.695854	0.008176	NaN
332	0.121202	0.983670	0.750115	NaN
333	0.223937	0.645885	0.837589	NaN

334 rows × 4 columns

In [45]:

```
#row delete kelya nantr index reset karnyasthi
newdf.reset_index(drop=True, inplace=True)
```

In [46]:

```
newdf.head()
```

Out[46]:

	C	D	E	0
0	0.261321	0.283373	0.710600	12223.0
1	0.356004	0.670794	0.367154	NaN
2	0.629785	0.416491	0.632631	NaN
3	0.862828	0.453117	0.251587	NaN
4	0.446373	0.229521	0.774194	NaN

In [56]:

```
df=pd.DataFrame({
  'cars': ["BMW", "Volvo", "Ford",'tata','tata'],
  'passings': [3, 7, 2,3,'NaT']
})
df
```

Out[56]:

	cars	passings
0	BMW	3
1	Volvo	7
2	Ford	2
3	tata	3
4	tata	NaT

In [66]:

```
df.dropna()
```

Out[66]:

	cars	passings
0	BMW	3
1	Volvo	7
2	Ford	2
3	tata	3
4	tata	NaT

In [64]:

```
df.drop_duplicates(subset=['cars'])
```

Out[64]:

	cars	passings
0	BMW	3
1	Volvo	7
2	Ford	2
3	tata	3

In [67]:

```
df.drop_duplicates(subset=['cars'], keep=False)
```

Out[67]:

	cars	passings
0	BMW	3
1	Volvo	7
2	Ford	2

In [70]:

```
df.drop_duplicates(subset=['cars'], keep='last')
```

Out[70]:

	cars	passings
0	BMW	3
1	Volvo	7
2	Ford	2
4	tata	NaT

In [77]:

```
df.drop_duplicates(subset=['cars'], keep='first', inplace=True)
df
```

Out[77]:

	cars	passings
0	BMW	3
1	Volvo	7
2	Ford	2
3	tata	3

In [76]:

```
df.shape               #return size of dataframe
```

Out[76]:

```
(4, 2)
```

In [78]:

```
df.info()    #return all information about your dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4 entries, 0 to 3
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   cars      4 non-null      object
 1   passings  4 non-null      object
dtypes: object(2)
memory usage: 96.0+ bytes
```

In [83]:

```
data=pd.read_excel('pandas.xlsx', sheet_name=0)
data
```

Out[83]:

	name	salary
0	rushi	50k
1	sagar	100k
2	keshav	50k
3	dipak	40k

In [92]:

```
data.loc[0,'name']='shital'
```

In [93]:

```
data
```

Out[93]:

	name	salary
0	shital	50k
1	sagar	100k
2	keshav	50k
3	dipak	40k

In [95]:

```
data.to_excel('pandas.xlsx')
```

In [ ]:

```

```