Pandas

```python Pandas is a library which helps to create the following Data Structures

1)Series - 1D array of data

2)DataFrame - 2D array of data

3)Panel - Multiple Dimension array of data

## A) Series -

Series is the 1D Data Structure

Different ways of Creating Series in pandas

1)Series With Array

2)Series With Lists

3)Series With Dictionary

# Different ways Of Creating Series

In [1]:
```
import pandas as pd
```

In [2]:
```
#Creating Series with list
l=[1,2,3,4,5]
s=pd.Series(l)
s
```

Out[2]:
```
0   1
1   2
2   3
3   4
4   5
dtype: int64
```

In [3]:
```
#we can also change the index of series by
l=[1,2,3,4,5]
s=pd.Series(l,index=['a','b','c','d','e'])
s
```

Out[3]:
```
a   1
b   2
c   3
d   4
e   5
dtype: int64
```

In [4]:
```
#Creating Series with Array(space seperated values in list is array)
import numpy as np
a=np.array([80,20,78])
s=pd.Series(a)
s
```

Out[4]:
```
0   80
1   20
2   78
dtype: int32
```

In [5]:
```
#Creating Series with Dictionary
d={'name':'miraj','roll':10,'age':15}
s=pd.Series(d)
s
```
```

Out[5]:
```
name    miraj
roll       10
age        15
dtype: object
```

B) DataFrame -

Data Frame is the 2D Data Structure (tabluar form)

Different ways of Creating Data Frame in pandas

1)By reading data from Excel File

2)By reading data from CSV File (comma seperated values)

3)BY using Dictionary

In [6]:
```
import pandas as pd
```

In [7]:
```
#making data frame by reading excel
df=pd.read_excel(r"excel_df.xlsx")
df
```

Out[7]:

	name	age	roll
0	miraj	10	76
1	apil	12	72
2	arbit	13	12
3	sabal	14	8

In [8]:
```
# making data frame by reading csv
df=pd.read_csv("new_data.csv")
df
```

Out[8]:

	sn	name	age	address
0	1	miraj	12	kalanki
1	4	ayush	18	kalanki

In [9]:
```
# making data frame by using dictionary
dict = {'name':['miraj','sabal','sarita','rashmi','sabnam'],
        'roll':[1,2,3,4,5] ,'age':[13,34,32,54,32] }

df=pd.DataFrame(dict)
df
```

Out[9]:

	name	roll	age
0	miraj	1	13
1	sabal	2	34
2	sarita	3	32
3	rashmi	4	54
4	sabnam	5	32

<!DOCTYPE html>

Attributes for Data Frame

`df.T` - Transpose the dataframe, (ROW TO COLUMN AND COLUMN TO ROW)

`df.at[1,'age']` - It only returns the single element from the data frame. It is label-based like `loc` (no slicing allowed, only returns one element).

`df.iat[1,2]` - It only returns the single element from the data frame. It is index-based like `iloc`, but the last index is inclusive (no slicing allowed, only returns one element).

`df.axes` - It returns the rows index (names) and columns index (names).

`df.index` - It only returns the rows index (names).

`df.columns` - It only returns the columns index (names).

`df.empty` - It returns a boolean value. If the data frame is empty, it gives `True`, and `False` if the data frame is not empty.

`df.ndim` - It returns the dimension of the dataframe.

`df.shape` - It returns the number of rows and number of columns (nrows x ncolumns).

`df.size` - It returns the total number of elements in the data frame.

`df.values` - It converts the data frame into a numpy array.

In [10]:
```
df #original dataframe
```

Out[10]:

	name	roll	age
0	miraj	1	13
1	sabal	2	34
2	sarita	3	32
3	rashmi	4	54
4	sabnam	5	32

In [11]:
```
df.T #transpose
```

Out[11]:

	0	1	2	3	4
name	miraj	sabal	sarita	rashmi	sabnam
roll	1	2	3	4	5
age	13	34	32	54	32

In [12]:
```
df #original dataframe
```

Out[12]:

	name	roll	age
0	miraj	1	13
1	sabal	2	34
2	sarita	3	32
3	rashmi	4	54
4	sabnam	5	32

In [13]:
```
'''Similar to loc, both need label based  lookups.
Use at if you only need to get or set a single value in a DataFrame or Series.'''
df.at[1,'age']
```

Out[13]:
```
34
```

In [14]:
```
df #original dataframe
```

Out[14]:

	name	roll	age
0	miraj	1	13
1	sabal	2	34
2	sarita	3	32
3	rashmi	4	54
4	sabnam	5	32

In [15]:
```
'''Similar to iloc, both need index based lookups.
Use at if you only need to get or set a single value in a DataFrame or Series.'''
df.iat[1,2] # the last value is inclusive
```

Out[15]:
```
34
```

In [16]:
```
df #original dataframe
```

Out[16]:

	name	roll	age
0	miraj	1	13
1	sabal	2	34
2	sarita	3	32
3	rashmi	4	54
4	sabnam	5	32

In [17]:
```
df.index #gives the row index of the dataframe
df.index=['a','b','c','d','e']#we can also change index like this
df
```

Out[17]:

	name	roll	age
a	miraj	1	13
b	sabal	2	34
c	sarita	3	32
d	rashmi	4	54
e	sabnam	5	32

In [18]:
```
df.axes # gives columns row names and columns anme
#df.axes[0]# row names
#df.axes[1]# columns names
```

Out[18]:
```
[Index(['a', 'b', 'c', 'd', 'e'], dtype='object'),
 Index(['name', 'roll', 'age'], dtype='object')]
```

In [19]:
```
df #original dataframe
```

Out[19]:

	name	roll	age
a	miraj	1	13
b	sabal	2	34
c	sarita	3	32
d	rashmi	4	54
e	sabnam	5	32

In [20]:
```
df.columns #gives coulmns names
```

Out[20]:
```
Index(['name', 'roll', 'age'], dtype='object')
```

In [21]:
```
df.dtypes #Return what types of data is stoted by each columns in data frame
```

Out[21]:
```
name    object
roll     int64
age      int64
dtype: object
```

In [22]:
```
df.empty #Indicator whether Series/DataFrame is empty.
```

Out[22]:
```
False
```

In [23]:
```
df.ndim #It returns the dimension of the dataframe.
```

Out[23]:
```
2
```

In [24]:
```
df.shape # It returns the number of rows and number of columns (nrows x ncolumns).
```

Out[24]:
```
(5, 3)
```

In [25]:
```
df.size # gives the number of elements in the data frame
```

Out[25]:
```
15
```

In [26]:
```
df #original dataframe
```

Out[26]:

	name	roll	age
a	miraj	1	13
b	sabal	2	34
c	sarita	3	32
d	rashmi	4	54
e	sabnam	5	32

In [27]:
```
df.values #converts numpy dataframe into numpy array !!! we cant do df.array to dataframe it is only working to
```

Out[27]:
```
array([['miraj', 1, 13],
       ['sabal', 2, 34],
       ['sarita', 3, 32],
       ['rashmi', 4, 54],
       ['sabnam', 5, 32]], dtype=object)
```

<!DOCTYPE html>

IMPORTANT METHODS() FOR DATAFRAME

`value_counts():` This method gives how many times a element is repeated in series. df['column'].value_count()

`to_dict():` This method converts a Series or DataFrame into a Python dictionary.

`sort_index():` This method sorts a Series or DataFrame by its index in ascending order.

`sort_values():` This method sorts a Series or DataFrame by its values in ascending order.

`set_index():` This method sets available column in df as the index of a DataFrame.

`reset_index():` This method resets the index of a DataFrame, converting the index into a regular column. It creates a new DataFrame with a default index.

`replace():` This method replaces specified values in a Series or DataFrame with other values.

`rename():` This method renames the row index or column labels of a Series or DataFrame.

`query():` This method filters a DataFrame using a Boolean expression and returns a new DataFrame containing the matching rows.

`notna():` gives boolean value true for not null values of dataframe.

`nsmallest():` This method returns the n smallest values from a Series or DataFrame

`nlargest():` This method returns the n largest values from a Series or DataFrame.

`nunique():` it counts the number of unique values present in the column.

`info():` This method provides a concise summary of a DataFrame, including its data types, non-null values, and memory usage.

`describe():` This method generates descriptive statistics of a DataFrame, including count, mean, standard deviation, minimum, maximum, and quartile values.

`count():` This method counts the number of non-null values in each column of a DataFrame.

`copy():` This method creates a deep copy of a Series or DataFrame.

In [28]:
```
data = pd.Series(['apple', 'banana', 'apple', 'orange', 'banana', 'apple'])

# Count the occurrences of each unique item
data.value_counts()
```

Out[28]:
```
apple     3
banana    2
orange    1
dtype: int64
```

In [29]:
```
# converts the dataframe into dictionary
df.to_dict()
```

Out[29]:
```
{'name': {'a': 'miraj',
  'b': 'sabal',
  'c': 'sarita',
  'd': 'rashmi',
  'e': 'sabnam'},
 'roll': {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5},
 'age': {'a': 13, 'b': 34, 'c': 32, 'd': 54, 'e': 32}}
```

In [30]:
```
# we can also convert data frame into jason
df.to_json('new_data.json')
```

In [31]:
```
# reading from json file
df=pd.read_json('new_data.json')
df
```

Out[31]:

	name	roll	age
a	miraj	1	13
b	sabal	2	34
c	sarita	3	32
d	rashmi	4	54
e	sabnam	5	32

In [32]:
```
df.index=['b','a','g','c','d']
#sorting the items on the basis of index in descending way
df.sort_index(ascending = False)
```

Out[32]:

	name	roll	age
g	sarita	3	32
d	sabnam	5	32
c	rashmi	4	54
b	miraj	1	13
a	sabal	2	34

In [33]:
```
# sort_values() this example is below in sorting topic
```

In [34]:
```
j=[i for i in range(5,10)]
#adding new column in dataframe to set index
df['s.n']=j
#setting the existing column to index
df.set_index('s.n',inplace=True)
df
```

Out[34]:

s.n	name	roll	age
5	miraj	1	13
6	sabal	2	34
7	sarita	3	32
8	rashmi	4	54
9	sabnam	5	32

In [35]:
```
# reset of index to default one
df.reset_index() #if we dont want s.n in colum we can do df.reset_index(drop=True) #if we dont want s.n in colu
```

Out[35]:

	s.n	name	roll	age
0	5	miraj	1	13
1	6	sabal	2	34
2	7	sarita	3	32
3	8	rashmi	4	54
4	9	sabnam	5	32

In [36]:
```
df.reset_index(drop=True)
```

Out[36]:

	name	roll	age
0	miraj	1	13
1	sabal	2	34
2	sarita	3	32
3	rashmi	4	54
4	sabnam	5	32

In [37]:
```
# Dataframe whhich consists the nan values
data = {'Name': ['John', 'Emma', np.nan],
        'Age': [25, np.nan, 30],
        'Height': [175, 162, np.nan]}
df = pd.DataFrame(data)
df
```

Out[37]:

	Name	Age	Height
0	John	25.0	175.0
1	Emma	NaN	162.0
2	NaN	30.0	NaN

In [38]:
```
#replacing nan values with None
df.replace(np.nan, None, inplace=True)
df
```

Out[38]:

	Name	Age	Height
0	John	25.0	175.0
1	Emma	None	162.0
2	None	30.0	None

In [39]:
```
#replacing Emma with Miraj
df.replace('Emma','Miraj',inplace=True)
df
```

Out[39]:

	Name	Age	Height
0	John	25.0	175.0
1	Miraj	NaN	162.0
2	None	30.0	NaN

In [40]:
```
#The rename() method in pandas is used to change the labels (names) of either the rows  or the columns  in a Da
df
```

Out[40]:

	Name	Age	Height
0	John	25.0	175.0
1	Miraj	NaN	162.0
2	None	30.0	NaN

In [41]:
```
#renamig the columns labels and row labes
col_corr={'Name':'NAME1','Age':'AGE1','Height':'HEIGHT1'}
row_corr={0:'a',1:'b',2:'c'}

df.rename(index=row_corr, columns=row_corr, inplace=True)# index is always row index
df
```

Out[41]:

	Name	Age	Height
a	John	25.0	175.0
b	Miraj	NaN	162.0
c	None	30.0	NaN

In [42]:
```
# performing the query() function in the dataframe

import pandas as pd

# Create a sample DataFrame
data = {'Name': ['John', 'Emma', 'Michael', 'Sophia'],
        'Age': [25, 28, 22, 30],
        'Height': [175, 162, 180, 168]}
df = pd.DataFrame(data)

# Print the original DataFrame
print("Original DataFrame:")
print(df)
#.................................see from here.....................................................................
# Perform different types of queries
filtered_df1 = df.query("Age > 25 and Height < 170")  # Numeric comparison
filtered_df2 = df.query("Name == 'John'")  # Name column ma john vako row dinxa
filtered_df3 = df.query("Name in ['John', 'Emma']")  # Select rows where Name is either 'John' or 'Emma'
filtered_df4 = df.query("not Height <= 168")  # # Select rows where Height is not lesser equal to 168
filtered_df5 = df.query("Name.str.startswith('S')")  # Select rows where Name starts with the letter 'M'

# Print the filtered DataFrame
print("\n")
print(filtered_df1)
print("\n")
print(filtered_df2)
print("\n")
print(filtered_df3)
print("\n")
print(filtered_df4)
print("\n")
print(filtered_df5)
```

```
Original DataFrame:
      Name  Age  Height
0     John   25     175
1     Emma   28     162
2  Michael   22     180
3   Sophia   30     168


     Name  Age  Height
1    Emma   28     162
3  Sophia   30     168


   Name  Age  Height
0  John   25     175


   Name  Age  Height
0  John   25     175
1  Emma   28     162


      Name  Age  Height
0     John   25     175
2  Michael   22     180


     Name  Age  Height
3  Sophia   30     168
```

In [43]:
```
# dataframe which have nan values

import pandas as pd
```

```
import numpy as np

# Create a sample DataFrame with missing values
data = {'Name': ['John', np.nan, 'Michael'],
        'Age': [25, np.nan, 22],
        'Height': [175, 162, np.nan]}
df = pd.DataFrame(data)

df
```

Out[43]:

	Name	Age	Height
0	John	25.0	175.0
1	NaN	NaN	162.0
2	Michael	22.0	NaN

In [44]:
```
df['Name'].notna() # returns the boolean value true if there is value in dataframe
```

Out[44]:
```
0     True
1    False
2     True
Name: Name, dtype: bool
```

In [45]:
```
# returns the row which have values
df[df['Name'].notna()]
```

Out[45]:

	Name	Age	Height
0	John	25.0	175.0
2	Michael	22.0	NaN

In [46]:
```
# just creating  a data frame
dict = {'name':['miraj','sabal','sarita','rashmi','sabnam'],
        'roll':[1,2,3,4,5] ,'age':[13,34,32,54,32]     }

df=pd.DataFrame(dict)
df
```

Out[46]:

	name	roll	age
0	miraj	1	13
1	sabal	2	34
2	sarita	3	32
3	rashmi	4	54
4	sabnam	5	32

In [47]:
```
s = df.nsmallest(2, 'age') #age column ko 2 ota smallest value dinxa
s
```

Out[47]:

	name	roll	age
0	miraj	1	13
2	sarita	3	32

In [48]:
```
l=df.nlargest(2, 'age') #age column ko 2 ota largest value dinxa
l
```

Out[48]:

	name	roll	age
3	rashmi	4	54
1	sabal	2	34

In [49]:
```
df # dataframe
```

Out[49]:

	name	roll	age
0	miraj	1	13
1	sabal	2	34
2	sarita	3	32
3	rashmi	4	54
4	sabnam	5	32

In [50]:
```
df.info()
#This method provides a concise summary of a DataFrame, including its data types, non-null values, and memory u
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   name    5 non-null      object
 1   roll    5 non-null      int64 
 2   age     5 non-null      int64 
dtypes: int64(2), object(1)
memory usage: 248.0+ bytes
```

In [51]:
```
df.describe()
#This method generates descriptive statistics of a DataFrame,
#including count, mean, standard deviation, minimum, maximum, and quartile values.
```

Out[51]:

	roll	age
count	5.000000	5.000000
mean	3.000000	33.000000
std	1.581139	14.525839
min	1.000000	13.000000
25%	2.000000	32.000000
50%	3.000000	32.000000
75%	4.000000	34.000000
max	5.000000	54.000000

In [52]:
```
df.count()
# count() returns the total no of non-null values in each column of a DataFrame or in the entire Series.
```

Out[52]:
```
name    5
roll    5
age     5
dtype: int64
```

In [53]:
```
#copying the dataframe into m
m=df.copy()
m
```

Out[53]:

	name	roll	age
0	miraj	1	13
1	sabal	2	34
2	sarita	3	32
3	rashmi	4	54
4	sabnam	5	32

In [54]:
```
df #original dataframe
```

Out[54]:

	name	roll	age
0	miraj	1	13
1	sabal	2	34
2	sarita	3	32
3	rashmi	4	54
4	sabnam	5	32

In [55]:
```
#kati ota unique values each column ma tyo dinxa (duplicated values lai euta consider garxa)
df.nunique() #age ma 32 dui ota repeat vako xa tei vara eutai manera value diyo
```

Out[55]:
```
name    5
roll    5
age     4
dtype: int64
```

<!DOCTYPE html>

Attributes for Series

`series.index` - Returns all index values

`series.array` - Returns all elements in a pandas array

`series.dtype` - Returns the data types of elements in the series

`series.values` - Returns elements of the series in a numpy array

`series.shape` - Returns the number of rows and number of columns (since series is 1D, it only gives the number of rows) or the total number of elements

`series.ndim` - Returns the dimension of the series

`series.size` - Returns the total number of elements present in the series

`series.empty` - Returns `true` if the series is empty and `false` if the series is not empty

`series.at` - Returns a single element from the series (label-based)

`series.iat` - Returns a single element from the series (index-based)

`series.loc` - Same as in a dataframe (label-based)

`series.iloc` - Same as in a dataframe (index-based slicing, last value exclusive)

In [56]:
```
import pandas as pd
```

In [57]:
```
s= pd.Series(['miraj','aman','apil','sabal'], index=['a','b','c','d'])
s
```

Out[57]:
```
a    miraj
b     aman
c     apil
d    sabal
dtype: object
```

In [58]:
```
#using index attributes (returns all index values)
s.index
```

Out[58]:
```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

In [59]:
```
#using array attributes (returns all data in  pandas array)
s.array
```

Out[59]:
```
<PandasArray>
['miraj', 'aman', 'apil', 'sabal']
Length: 4, dtype: object
```

In [60]:
```
#using dtype attributes  gives what is the data types of elements of this series
s.dtype
```

Out[60]:
```
dtype('O')
```

In [61]:
```
#using values attributes (returns values of series in numpy array)
s.values
```

Out[61]:
```
array(['miraj', 'aman', 'apil', 'sabal'], dtype=object)
```

In [62]:
```
#using shape attributes (for series shape 4 represents the 4 rows )
#for series s.shape also help us to find the numbers of elements in the series like s.size
s.shape
```

Out[62]:
```
(4,)
```

In [63]:
```
#using shape attributes (returns the no of dimension of the series)
s.ndim
```

Out[63]:
```
1
```

In [64]:
```
#using size attributes gives the total number of elements in the series
s.size
```

Out[64]:
```
4
```

In [65]:
```
#using empty attributes gives boolean values (if series is empty then True and if series is not empty then Fals
s.empty
```

Out[65]:
```
False
```

In [66]:
```
s
```

Out[66]:
```
a    miraj
b     aman
c     apil
d    sabal
dtype: object
```

In [67]:
```
s.at['a'] # 'a' label ma vako value dinxa series bata
#s['a']
```

Out[67]:
```
'miraj'
```

In [68]:
```
s.iat[3]# '3' index ma vako value dinxa series bata
```

Out[68]:
```
'sabal'
```

Mathematical Operations in series

In [69]:
```
import pandas as pd

s1=pd.Series([1,2,3])
s2=pd.Series([2,3,4])
```

In [70]:
```
#add
s1+s2
#s1.add(s2)
```

Out[70]:
```
0    3
1    5
2    7
dtype: int64
```

In [71]:
```
#substract
s1-s2
```

Out[71]:
```
0   -1
1   -1
2   -1
dtype: int64
```

In [72]:
```
#multiply
s1*s2
```

Out[72]:
```
0     2
1     6
2    12
dtype: int64
```

In [73]:
```
#division
s1/s2
```

Out[73]:
```
0    0.500000
1    0.666667
2    0.750000
dtype: float64
```

In [74]:
```
#power
s1**s2
```

Out[74]:
```
0     1
1     8
2    81
dtype: int64
```

In [75]:
```
#less than
s1<s2
```

Out[75]:
```
0    True
1    True
2    True
dtype: bool
```

In [76]:
```
#greater than
s1>s2
```

Out[76]:
```
0    False
1    False
2    False
dtype: bool
```

In [77]:
```
#equalto
s1==s2
```

Out[77]:
```
0    False
1    False
2    False
dtype: bool
```

Getting value from the series

In [78]:
```
import pandas as pd
s=pd.Series([1,3,4,6])
s
```

Out[78]:
```
0    1
1    3
2    4
3    6
dtype: int64
```

In [79]:
```
#getting index 3 value
s[3]
```

Out[79]:
```
6
```

In [80]:
```
#getting index 1 and 2 element
s[1:3]
```

Out[80]:
```
1    3
2    4
dtype: int64
```

In [81]:
```
#we can also set index like this in series
s.index=['a','b','c','d']
s
```

Out[81]:
```
a    1
b    3
c    4
d    6
dtype: int64
```

In [82]:
```
s['d']
```

Out[82]:
```
6
```

IMP Functions, Accessing values, Slicing in data frame

In [83]:
```
import pandas as pd
df=pd.read_excel("example.xlsx")
df
```

Out[83]:

	Name	Age	Roll	Salary	Address
0	John Doe	27	101	50000	123 Main Street
1	Jane Smith	32	102	65000	456 Elm Avenue
2	David Johnson	41	103	80000	789 Oak Lane
3	Sarah Williams	29	104	55000	321 Pine Road
4	Michael Brown	35	105	70000	654 Maple Drive
5	Emily Davis	24	106	45000	987 Cedar Street
6	James Wilson	38	107	75000	543 Birch Court
7	Jennifer Taylor	31	108	60000	876 Walnut Lane
8	Robert Anderson	26	109	48000	234 Spruce Avenue
9	Jessica Thomas	33	110	68000	567 Cherry Lane

In [84]:
```
# To set the custom index in data frame by reading excel and CSV File
df=pd.read_excel('example.xlsx',index_col = 'Roll')
df
```

Out[84]:

Roll	Name	Age	Salary	Address
101	John Doe	27	50000	123 Main Street
102	Jane Smith	32	65000	456 Elm Avenue
103	David Johnson	41	80000	789 Oak Lane
104	Sarah Williams	29	55000	321 Pine Road
105	Michael Brown	35	70000	654 Maple Drive
106	Emily Davis	24	45000	987 Cedar Street
107	James Wilson	38	75000	543 Birch Court
108	Jennifer Taylor	31	60000	876 Walnut Lane
109	Robert Anderson	26	48000	234 Spruce Avenue
110	Jessica Thomas	33	68000	567 Cherry Lane

In [85]:
```
#we can also give index like this
df=pd.read_excel('example.xlsx')
df.set_index("Name", inplace=True) #.set_index() returns new df so to modify in old data frame inplace=True
df
```

Out[85]:

Name	Age	Roll	Salary	Address
John Doe	27	101	50000	123 Main Street
Jane Smith	32	102	65000	456 Elm Avenue
David Johnson	41	103	80000	789 Oak Lane
Sarah Williams	29	104	55000	321 Pine Road
Michael Brown	35	105	70000	654 Maple Drive
Emily Davis	24	106	45000	987 Cedar Street
James Wilson	38	107	75000	543 Birch Court
Jennifer Taylor	31	108	60000	876 Walnut Lane
Robert Anderson	26	109	48000	234 Spruce Avenue
Jessica Thomas	33	110	68000	567 Cherry Lane

In [86]:
```
# To make data frame from the specific column while reading csv and excel file
df=pd.read_excel('example.xlsx',index_col = 'Roll', usecols=['Name','Age','Roll'])
df
```

Out[86]:

Roll	Name	Age
101	John Doe	27
102	Jane Smith	32
103	David Johnson	41
104	Sarah Williams	29
105	Michael Brown	35
106	Emily Davis	24
107	James Wilson	38
108	Jennifer Taylor	31
109	Robert Anderson	26
110	Jessica Thomas	33

In [87]:
```
# To get only specific rows in data frame (nrows)
df=pd.read_excel('example.xlsx',index_col = 'Roll', usecols=['Name','Age','Roll'], nrows=2)
df
```

Out[87]:

Roll	Name	Age
101	John Doe	27
102	Jane Smith	32

In [88]:
```
df=pd.read_excel('example.xlsx')
df
```

Out[88]:

	Name	Age	Roll	Salary	Address
0	John Doe	27	101	50000	123 Main Street
1	Jane Smith	32	102	65000	456 Elm Avenue
2	David Johnson	41	103	80000	789 Oak Lane
3	Sarah Williams	29	104	55000	321 Pine Road
4	Michael Brown	35	105	70000	654 Maple Drive
5	Emily Davis	24	106	45000	987 Cedar Street
6	James Wilson	38	107	75000	543 Birch Court
7	Jennifer Taylor	31	108	60000	876 Walnut Lane
8	Robert Anderson	26	109	48000	234 Spruce Avenue
9	Jessica Thomas	33	110	68000	567 Cherry Lane

In [89]:
```
df.head() #gives first 5 rows
```

Out[89]:

	Name	Age	Roll	Salary	Address
0	John Doe	27	101	50000	123 Main Street
1	Jane Smith	32	102	65000	456 Elm Avenue
2	David Johnson	41	103	80000	789 Oak Lane
3	Sarah Williams	29	104	55000	321 Pine Road
4	Michael Brown	35	105	70000	654 Maple Drive

In [90]:
```
df.tail() #gives last 5 rows
#we can also specify how much we want df.head(10), df.tail(11)
```

Out[90]:

	Name	Age	Roll	Salary	Address
5	Emily Davis	24	106	45000	987 Cedar Street
6	James Wilson	38	107	75000	543 Birch Court
7	Jennifer Taylor	31	108	60000	876 Walnut Lane
8	Robert Anderson	26	109	48000	234 Spruce Avenue
9	Jessica Thomas	33	110	68000	567 Cherry Lane

In [91]:
```
# to get particular colum form the dataframe
df['Name']
```

Out[91]:
```
0           John Doe
1         Jane Smith
2      David Johnson
3     Sarah Williams
4      Michael Brown
5        Emily Davis
6       James Wilson
7    Jennifer Taylor
8    Robert Anderson
9     Jessica Thomas
Name: Name, dtype: object
```

In [92]:
```
# to get specific value from the colum of dataframe (data frame bata series aayo series bata index bata value)
df['Name'][5]
```

Out[92]:
```
'Emily Davis'
```

In [93]:
```
# To get only 5 value from colums/series
df['Name'].head()
```

Out[93]:
```
0          John Doe
1        Jane Smith
2     David Johnson
3    Sarah Williams
4     Michael Brown
Name: Name, dtype: object
```

In [94]:
```
#if we have to access the two colums form the data frame the we have to use use nexted list
df[['Name','Age','Salary']]
```

Out[94]:

	Name	Age	Salary
0	John Doe	27	50000
1	Jane Smith	32	65000
2	David Johnson	41	80000
3	Sarah Williams	29	55000
4	Michael Brown	35	70000
5	Emily Davis	24	45000
6	James Wilson	38	75000
7	Jennifer Taylor	31	60000
8	Robert Anderson	26	48000
9	Jessica Thomas	33	68000

In [95]:
```
# slicing #default slicing is row
df[2:7]
```

Out[95]:

	Name	Age	Roll	Salary	Address
2	David Johnson	41	103	80000	789 Oak Lane
3	Sarah Williams	29	104	55000	321 Pine Road
4	Michael Brown	35	105	70000	654 Maple Drive
5	Emily Davis	24	106	45000	987 Cedar Street
6	James Wilson	38	107	75000	543 Birch Court

In [96]:
```
# now getting colum from the slicing
df[2:7]['Salary']
```

Out[96]:
```
2    80000
3    55000
4    70000
5    45000
6    75000
Name: Salary, dtype: int64
```

In [97]:
```
df[0:10:2] #with two step slicing
```

Out[97]:

	Name	Age	Roll	Salary	Address
0	John Doe	27	101	50000	123 Main Street
2	David Johnson	41	103	80000	789 Oak Lane
4	Michael Brown	35	105	70000	654 Maple Drive
6	James Wilson	38	107	75000	543 Birch Court
8	Robert Anderson	26	109	48000	234 Spruce Avenue

In [98]:
```
# slicing multiple colums in dataframe
df[['Name','Age','Salary']][2:9]
```

Out[98]:

	Name	Age	Salary
2	David Johnson	41	80000
3	Sarah Williams	29	55000
4	Michael Brown	35	70000
5	Emily Davis	24	45000
6	James Wilson	38	75000
7	Jennifer Taylor	31	60000
8	Robert Anderson	26	48000

In [99]:
```
df
```

Out[99]:

	Name	Age	Roll	Salary	Address
0	John Doe	27	101	50000	123 Main Street
1	Jane Smith	32	102	65000	456 Elm Avenue
2	David Johnson	41	103	80000	789 Oak Lane
3	Sarah Williams	29	104	55000	321 Pine Road
4	Michael Brown	35	105	70000	654 Maple Drive
5	Emily Davis	24	106	45000	987 Cedar Street
6	James Wilson	38	107	75000	543 Birch Court
7	Jennifer Taylor	31	108	60000	876 Walnut Lane
8	Robert Anderson	26	109	48000	234 Spruce Avenue
9	Jessica Thomas	33	110	68000	567 Cherry Lane

In [100…
```
# to get specific row as series
df.iloc[2] # gives index 2 row value as series
```

Out[100]:
```
Name       David Johnson
Age                   41
Roll                 103
Salary             80000
Address     789 Oak Lane
Name: 2, dtype: object
```

In [101…
```
df=pd.read_excel("example.xlsx",index_col="Name")
df
```

Out[101]:

Name	Age	Roll	Salary	Address
John Doe	27	101	50000	123 Main Street
Jane Smith	32	102	65000	456 Elm Avenue
David Johnson	41	103	80000	789 Oak Lane
Sarah Williams	29	104	55000	321 Pine Road
Michael Brown	35	105	70000	654 Maple Drive
Emily Davis	24	106	45000	987 Cedar Street
James Wilson	38	107	75000	543 Birch Court
Jennifer Taylor	31	108	60000	876 Walnut Lane
Robert Anderson	26	109	48000	234 Spruce Avenue
Jessica Thomas	33	110	68000	567 Cherry Lane

In [102…
```
# To get value of James Wilson row
df.iloc[6]
```

Out[102]:
```
Age                    38
Roll                  107
Salary              75000
Address    543 Birch Court
Name: James Wilson, dtype: object
```

In [103…
```
df.loc['James Wilson']
```

Out[103]:
```
Age                    38
Roll                  107
Salary              75000
Address    543 Birch Court
Name: James Wilson, dtype: object
```

Sorting the dataframe on the basis Values of column asscending and descending

In [104…
```
df
```

Out[104]:

Name	Age	Roll	Salary	Address
John Doe	27	101	50000	123 Main Street
Jane Smith	32	102	65000	456 Elm Avenue
David Johnson	41	103	80000	789 Oak Lane
Sarah Williams	29	104	55000	321 Pine Road
Michael Brown	35	105	70000	654 Maple Drive
Emily Davis	24	106	45000	987 Cedar Street
James Wilson	38	107	75000	543 Birch Court
Jennifer Taylor	31	108	60000	876 Walnut Lane
Robert Anderson	26	109	48000	234 Spruce Avenue
Jessica Thomas	33	110	68000	567 Cherry Lane

In [105…
```
#sorting Assending wise
df.sort_values('Salary')
```

Out[105]:

Name	Age	Roll	Salary	Address
Emily Davis	24	106	45000	987 Cedar Street
Robert Anderson	26	109	48000	234 Spruce Avenue
John Doe	27	101	50000	123 Main Street
Sarah Williams	29	104	55000	321 Pine Road
Jennifer Taylor	31	108	60000	876 Walnut Lane
Jane Smith	32	102	65000	456 Elm Avenue
Jessica Thomas	33	110	68000	567 Cherry Lane
Michael Brown	35	105	70000	654 Maple Drive
James Wilson	38	107	75000	543 Birch Court
David Johnson	41	103	80000	789 Oak Lane

In [106…
```
#sorting Descending wise
df.sort_values('Salary', ascending=False)
```

Out[106]:

Name	Age	Roll	Salary	Address
David Johnson	41	103	80000	789 Oak Lane
James Wilson	38	107	75000	543 Birch Court
Michael Brown	35	105	70000	654 Maple Drive
Jessica Thomas	33	110	68000	567 Cherry Lane
Jane Smith	32	102	65000	456 Elm Avenue
Jennifer Taylor	31	108	60000	876 Walnut Lane
Sarah Williams	29	104	55000	321 Pine Road
John Doe	27	101	50000	123 Main Street
Robert Anderson	26	109	48000	234 Spruce Avenue
Emily Davis	24	106	45000	987 Cedar Street

Adding and Dropping the Column in Data Frame

In [107…
```
df=pd.read_excel("example.xlsx")
df
```

Out[107]:

	Name	Age	Roll	Salary	Address
0	John Doe	27	101	50000	123 Main Street
1	Jane Smith	32	102	65000	456 Elm Avenue
2	David Johnson	41	103	80000	789 Oak Lane
3	Sarah Williams	29	104	55000	321 Pine Road
4	Michael Brown	35	105	70000	654 Maple Drive
5	Emily Davis	24	106	45000	987 Cedar Street
6	James Wilson	38	107	75000	543 Birch Court
7	Jennifer Taylor	31	108	60000	876 Walnut Lane
8	Robert Anderson	26	109	48000	234 Spruce Avenue
9	Jessica Thomas	33	110	68000	567 Cherry Lane

In [108…
```
# Adding new Column Gender
df['Gender']=['m','f','f','m','m','f','m','f','m','f']
df
```

Out[108]:

	Name	Age	Roll	Salary	Address	Gender
0	John Doe	27	101	50000	123 Main Street	m
1	Jane Smith	32	102	65000	456 Elm Avenue	f
2	David Johnson	41	103	80000	789 Oak Lane	f
3	Sarah Williams	29	104	55000	321 Pine Road	m
4	Michael Brown	35	105	70000	654 Maple Drive	m
5	Emily Davis	24	106	45000	987 Cedar Street	f
6	James Wilson	38	107	75000	543 Birch Court	m
7	Jennifer Taylor	31	108	60000	876 Walnut Lane	f
8	Robert Anderson	26	109	48000	234 Spruce Avenue	m
9	Jessica Thomas	33	110	68000	567 Cherry Lane	f

In [109…
```
df['roll+Age']=df['Age']+df['Roll'] # 2 ota series return hunxa ra sum hunxa ani naya column ma basnxa value
df
```

Out[109]:

	Name	Age	Roll	Salary	Address	Gender	roll+Age
0	John Doe	27	101	50000	123 Main Street	m	128
1	Jane Smith	32	102	65000	456 Elm Avenue	f	134
2	David Johnson	41	103	80000	789 Oak Lane	f	144
3	Sarah Williams	29	104	55000	321 Pine Road	m	133
4	Michael Brown	35	105	70000	654 Maple Drive	m	140
5	Emily Davis	24	106	45000	987 Cedar Street	f	130
6	James Wilson	38	107	75000	543 Birch Court	m	145
7	Jennifer Taylor	31	108	60000	876 Walnut Lane	f	139
8	Robert Anderson	26	109	48000	234 Spruce Avenue	m	135
9	Jessica Thomas	33	110	68000	567 Cherry Lane	f	143

In [110…
```
df=df.drop(columns="roll+Age") # does not delete on orignal datafrae so df=....
df
```

Out[110]:

	Name	Age	Roll	Salary	Address	Gender
0	John Doe	27	101	50000	123 Main Street	m
1	Jane Smith	32	102	65000	456 Elm Avenue	f
2	David Johnson	41	103	80000	789 Oak Lane	f
3	Sarah Williams	29	104	55000	321 Pine Road	m
4	Michael Brown	35	105	70000	654 Maple Drive	m
5	Emily Davis	24	106	45000	987 Cedar Street	f
6	James Wilson	38	107	75000	543 Birch Court	m
7	Jennifer Taylor	31	108	60000	876 Walnut Lane	f
8	Robert Anderson	26	109	48000	234 Spruce Avenue	m
9	Jessica Thomas	33	110	68000	567 Cherry Lane	f

In [111…
```
#Adding multiple Colums in data frame
```

```
df[['Height', 'Weight']] = [['5ft', '1kg'],
                            ['5.5ft', '2kg'],
                            ['6ft', '33kg'],
                            ['7ft', '43kg'],
                            ['4ft', '34kg'],
                            ['4ft', '56kg'],
                            ['3ft', '45kg'],
                            ['5ft', '76kg'],
                            ['4.5ft', '78kg'],
                            ['6ft', '90kg']]

df
```

Out[111]:

	Name	Age	Roll	Salary	Address	Gender	Height	Weight
0	John Doe	27	101	50000	123 Main Street	m	5ft	1kg
1	Jane Smith	32	102	65000	456 Elm Avenue	f	5.5ft	2kg
2	David Johnson	41	103	80000	789 Oak Lane	f	6ft	33kg
3	Sarah Williams	29	104	55000	321 Pine Road	m	7ft	43kg
4	Michael Brown	35	105	70000	654 Maple Drive	m	4ft	34kg
5	Emily Davis	24	106	45000	987 Cedar Street	f	4ft	56kg
6	James Wilson	38	107	75000	543 Birch Court	m	3ft	45kg
7	Jennifer Taylor	31	108	60000	876 Walnut Lane	f	5ft	76kg
8	Robert Anderson	26	109	48000	234 Spruce Avenue	m	4.5ft	78kg
9	Jessica Thomas	33	110	68000	567 Cherry Lane	f	6ft	90kg

In [112…
```
# droppimg multiple columns in dataframe
df=df.drop(columns=['Height','Weight','Gender'])
df
```

Out[112]:

	Name	Age	Roll	Salary	Address
0	John Doe	27	101	50000	123 Main Street
1	Jane Smith	32	102	65000	456 Elm Avenue
2	David Johnson	41	103	80000	789 Oak Lane
3	Sarah Williams	29	104	55000	321 Pine Road
4	Michael Brown	35	105	70000	654 Maple Drive
5	Emily Davis	24	106	45000	987 Cedar Street
6	James Wilson	38	107	75000	543 Birch Court
7	Jennifer Taylor	31	108	60000	876 Walnut Lane
8	Robert Anderson	26	109	48000	234 Spruce Avenue
9	Jessica Thomas	33	110	68000	567 Cherry Lane

Exporting the DataFrame into Excel, csv and txt file

In [113…
```
import pandas as pd
dict={'name':['miraj','ram','shyam','gita'],'roll':[1,2,3,4],'age':[12,32,31,54]}
df=pd.DataFrame(dict)
df
```

Out[113]:

	name	roll	age
0	miraj	1	12
1	ram	2	32
2	shyam	3	31
3	gita	4	54

In [114…
```
# Expoting the data frame into excel file
df.to_excel("export_into_excel.xlsx",index=False)
```

In [115…
```
# Expoting the data frame into csv file
df.to_csv("export_into_csv.csv",index=False)
```

In [116…
```
# Expoting the data frame into txt file
df.to_csv("export_into_txt.txt",index=False)
```

Understanding loc and iloc

In [117…
```
df=pd.read_excel('example.xlsx', index_col="Name")
df
```

Out[117]:

Name	Age	Roll	Salary	Address
John Doe	27	101	50000	123 Main Street
Jane Smith	32	102	65000	456 Elm Avenue
David Johnson	41	103	80000	789 Oak Lane
Sarah Williams	29	104	55000	321 Pine Road
Michael Brown	35	105	70000	654 Maple Drive
Emily Davis	24	106	45000	987 Cedar Street
James Wilson	38	107	75000	543 Birch Court
Jennifer Taylor	31	108	60000	876 Walnut Lane
Robert Anderson	26	109	48000	234 Spruce Avenue
Jessica Thomas	33	110	68000	567 Cherry Lane

loc----> it helps to retrive value of the basis of label df.loc[1:3,'name':'roll'](the stop is inclusive here in loc in slicing) iloc---> it helps to retrive value of the basis of index (the stop is exclusive here in iloc in slicing)

In [118…
```
#using loc for  row df.loc[row,column]
df.loc['John Doe']
```

Out[118]:
```
Age                     27
Roll                   101
Salary               50000
Address    123 Main Street
Name: John Doe, dtype: object
```

In [119…
```
#using loc to get value
df.loc['John Doe','Salary'] #df.loc[row,column]
```

Out[119]:
```
50000
```

In [120…
```
#multiple column chaiye list bhitra halne
df.loc['John Doe',['Salary','Address']]
```

Out[120]:
```
Salary               50000
Address    123 Main Street
Name: John Doe, dtype: object
```

In [121…
```
df
```

Out[121]:

Name	Age	Roll	Salary	Address
John Doe	27	101	50000	123 Main Street
Jane Smith	32	102	65000	456 Elm Avenue
David Johnson	41	103	80000	789 Oak Lane
Sarah Williams	29	104	55000	321 Pine Road
Michael Brown	35	105	70000	654 Maple Drive
Emily Davis	24	106	45000	987 Cedar Street
James Wilson	38	107	75000	543 Birch Court
Jennifer Taylor	31	108	60000	876 Walnut Lane
Robert Anderson	26	109	48000	234 Spruce Avenue
Jessica Thomas	33	110	68000	567 Cherry Lane

In [122…
```
# slicing  row

df.loc['John Doe':'Michael Brown']
```

Out[122]:

Name	Age	Roll	Salary	Address
John Doe	27	101	50000	123 Main Street
Jane Smith	32	102	65000	456 Elm Avenue
David Johnson	41	103	80000	789 Oak Lane
Sarah Williams	29	104	55000	321 Pine Road
Michael Brown	35	105	70000	654 Maple Drive

In [123…
```
# getting multiple values from the dataframe using loc
```

```
df.loc['Jane Smith':'David Johnson','Age':'Roll']
```

Out[123]:

Name	Age	Roll
Jane Smith	32	102
David Johnson	41	103

In [124…
```
#jane dekhi david samma ko age ra address matra chaiye
df.loc['Jane Smith':'David Johnson',['Age','Address']]
```

Out[124]:

Name	Age	Address
Jane Smith	32	456 Elm Avenue
David Johnson	41	789 Oak Lane

In [125…
```
df
```

Out[125]:

Name	Age	Roll	Salary	Address
John Doe	27	101	50000	123 Main Street
Jane Smith	32	102	65000	456 Elm Avenue
David Johnson	41	103	80000	789 Oak Lane
Sarah Williams	29	104	55000	321 Pine Road
Michael Brown	35	105	70000	654 Maple Drive
Emily Davis	24	106	45000	987 Cedar Street
James Wilson	38	107	75000	543 Birch Court
Jennifer Taylor	31	108	60000	876 Walnut Lane
Robert Anderson	26	109	48000	234 Spruce Avenue
Jessica Thomas	33	110	68000	567 Cherry Lane

In [126…
```
# 104 ra 105 matra chaiyo rey aba
df.loc[['Sarah Williams','Michael Brown'],'Roll']
```

Out[126]:
```
Name
Sarah Williams    104
Michael Brown     105
Name: Roll, dtype: int64
```

In [127…
```
df.iloc[3:5,1]
```

Out[127]:
```
Name
Sarah Williams    104
Michael Brown     105
Name: Roll, dtype: int64
```

In [128…
```
#we can convert any row or column into list

list(df['Roll'])
```

Out[128]:
```
[101, 102, 103, 104, 105, 106, 107, 108, 109, 110]
```

In [129…
```
list(df.loc['John Doe'])
```

Out[129]:
```
[27, 101, 50000, '123 Main Street']
```

Handeling missing data in data frame

In [130…
```
df=pd.read_excel("missing_values.xlsx")
df
```

Out[130]:

	SN	Age	Roll	Employee ID	Name
0	1.0	32	A01	100234.0	NaN
1	2.0	5	B07	102345.0	Jane Smith
2	3.0	42	C03	105678.0	David Brown
3	4.0	37	NaN	107890.0	Lisa Green
4	NaN	29	E05	109876.0	Mark Davis
5	6.0	31	F08	NaN	Anna

In [131…
```
#check if the values are null
df.isnull()
```

Out[131]:

	SN	Age	Roll	Employee ID	Name
0	False	False	False	False	True
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	True	False	False
4	True	False	False	False	False
5	False	False	False	True	False

In [132…
```
# returns the total number of null fields in each columns
df.isnull().sum()
```

Out[132]:
```
SN             1
Age            0
Roll           1
Employee ID    1
Name           1
dtype: int64
```

TO DELETE THE VALUES ROW

In [133…
```
df=pd.read_excel("missing_values.xlsx")
df
```

Out[133]:

	SN	Age	Roll	Employee ID	Name
0	1.0	32	A01	100234.0	NaN
1	2.0	5	B07	102345.0	Jane Smith
2	3.0	42	C03	105678.0	David Brown
3	4.0	37	NaN	107890.0	Lisa Green
4	NaN	29	E05	109876.0	Mark Davis
5	6.0	31	F08	NaN	Anna

In [134…
```
#to delete the row which have no value
df.dropna(inplace=True) #inplace true is for to change in original data frame
df
```

Out[134]:

	SN	Age	Roll	Employee ID	Name
1	2.0	5	B07	102345.0	Jane Smith
2	3.0	42	C03	105678.0	David Brown

In [135…
```
# we can also remove the missing values from specific colums
df=pd.read_excel("missing_values.xlsx")
df
```

Out[135]:

	SN	Age	Roll	Employee ID	Name
0	1.0	32	A01	100234.0	NaN
1	2.0	5	B07	102345.0	Jane Smith
2	3.0	42	C03	105678.0	David Brown
3	4.0	37	NaN	107890.0	Lisa Green
4	NaN	29	E05	109876.0	Mark Davis
5	6.0	31	F08	NaN	Anna

In [136…
```
df['Name'].dropna() # 0 index NAN is removed
```

Out[136]:
```
1     Jane Smith
2    David Brown
3     Lisa Green
4     Mark Davis
5           Anna
Name: Name, dtype: object
```

In [137…
```
df[['Name','Roll']].dropna() # removing the NAN values from from the name ans roll column
```

Out[137]:

	Name	Roll
1	Jane Smith	B07
2	David Brown	C03
4	Mark Davis	E05
5	Anna	F08

TO FILL THE MISSING VALUE FIELD WITH DEFAULT VALUE OR OTHER VALUES

In [138…
```
df
```

Out[138]:

	SN	Age	Roll	Employee ID	Name
0	1.0	32	A01	100234.0	NaN
1	2.0	5	B07	102345.0	Jane Smith
2	3.0	42	C03	105678.0	David Brown
3	4.0	37	NaN	107890.0	Lisa Green
4	NaN	29	E05	109876.0	Mark Davis
5	6.0	31	F08	NaN	Anna

In [139…
```
#filling all the missing filelds with 4
df.fillna(4)
```

Out[139]:

	SN	Age	Roll	Employee ID	Name
0	1.0	32	A01	100234.0	4
1	2.0	5	B07	102345.0	Jane Smith
2	3.0	42	C03	105678.0	David Brown
3	4.0	37	4	107890.0	Lisa Green
4	4.0	29	E05	109876.0	Mark Davis
5	6.0	31	F08	4.0	Anna

In [140…
```
#filling a specific column with valuie
df
```

Out[140]:

	SN	Age	Roll	Employee ID	Name
0	1.0	32	A01	100234.0	NaN
1	2.0	5	B07	102345.0	Jane Smith
2	3.0	42	C03	105678.0	David Brown
3	4.0	37	NaN	107890.0	Lisa Green
4	NaN	29	E05	109876.0	Mark Davis
5	6.0	31	F08	NaN	Anna

In [141…
```
#name column ko value na vako lai fill gardai
df['Name'].fillna('miraj',inplace=True) # inplace true is to make change in original datafraame
df
```

Out[141]:

	SN	Age	Roll	Employee ID	Name
0	1.0	32	A01	100234.0	miraj
1	2.0	5	B07	102345.0	Jane Smith
2	3.0	42	C03	105678.0	David Brown
3	4.0	37	NaN	107890.0	Lisa Green
4	NaN	29	E05	109876.0	Mark Davis
5	6.0	31	F08	NaN	Anna

In [142…
```
df['Roll'].fillna(15, inplace=True) # filling 5 in empty field of Age column
df # look change to roll up and down table
```

Out[142]:

	SN	Age	Roll	Employee ID	Name
0	1.0	32	A01	100234.0	miraj
1	2.0	5	B07	102345.0	Jane Smith
2	3.0	42	C03	105678.0	David Brown
3	4.0	37	15	107890.0	Lisa Green
4	NaN	29	E05	109876.0	Mark Davis
5	6.0	31	F08	NaN	Anna

In [143…
```
# we can also give mean vaue to empty field like this
df['SN'].fillna(df['SN'].mean()) #SN KO MEAN NIKALERA FILLNA MA AS A ARGUMENT VALUE PASS GARDINXA
```

Out[143]:
```
0    1.0
1    2.0
2    3.0
3    4.0
4    3.2
5    6.0
Name: SN, dtype: float64
```

Handeling the duplicated Values

In [144…
```
df=pd.read_excel('duplicate_values.xlsx')
df
#here the values are duplicated in data frame
```

Out[144]:

	SN	Age	Roll	Employee ID	Name
0	1	32	A01	100234.0	mmd
1	2	5	B07	102345.0	Jane Smith
2	3	42	C03	105678.0	David Brown
3	4	34	df4	23242.0	miraj
4	2	5	B07	102345.0	Jane Smith
5	6	31	F08	NaN	Anna
6	4	34	df4	23242.0	miraj

In [145…
```
# to check the values are duplicated or not in data frame
df.duplicated()
```

Out[145]:
```
0    False
1    False
2    False
3    False
4     True
5    False
6     True
dtype: bool
```

In [146…
```
# to see the actual duplicated value / we wrap with another df[] only true values comes
df[df.duplicated()]
```

Out[146]:

	SN	Age	Roll	Employee ID	Name
4	2	5	B07	102345.0	Jane Smith
6	4	34	df4	23242.0	miraj

In [147…
```
#to remove the duplicated values we can use
df.drop_duplicates(inplace=True)
df
```

Out[147]:

	SN	Age	Roll	Employee ID	Name
0	1	32	A01	100234.0	mmd
1	2	5	B07	102345.0	Jane Smith
2	3	42	C03	105678.0	David Brown
3	4	34	df4	23242.0	miraj
5	6	31	F08	NaN	Anna

In [148…
```
df # dataframe
```

Out[148]:

	SN	Age	Roll	Employee ID	Name
0	1	32	A01	100234.0	mmd
1	2	5	B07	102345.0	Jane Smith
2	3	42	C03	105678.0	David Brown
3	4	34	df4	23242.0	miraj
5	6	31	F08	NaN	Anna

In [149…
```
# to see the non repeated value we can use not(~)
~df.duplicated()
```

Out[149]:
```
0    True
1    True
2    True
3    True
5    True
dtype: bool
```

TO CHECK the number of unique values in each column

In [150…
```
df
```

Out[150]:

	SN	Age	Roll	Employee ID	Name
0	1	32	A01	100234.0	mmd
1	2	5	B07	102345.0	Jane Smith
2	3	42	C03	105678.0	David Brown
3	4	34	df4	23242.0	miraj
5	6	31	F08	NaN	Anna

In [151…
```
df.nunique() # sabai column ma kati ota unique value xa teko count dinxa
```

Out[151]:
```
SN             5
Age            5
Roll           5
Employee ID    4
Name           5
dtype: int64
```

Checking conditions in to filter data

In [152…
```
df
```

Out[152]:

	SN	Age	Roll	Employee ID	Name
0	1	32	A01	100234.0	mmd
1	2	5	B07	102345.0	Jane Smith
2	3	42	C03	105678.0	David Brown
3	4	34	df4	23242.0	miraj
5	6	31	F08	NaN	Anna

In [153…
```
#Checking if Name colum have miraj or not
df['Name']=='miraj'
```

Out[153]:
```
0    False
1    False
2    False
3     True
5    False
Name: Name, dtype: bool
```

In [154…
```
# to extract the value of row of miraj
df[df['Name']=='miraj']
```

Out[154]:

	SN	Age	Roll	Employee ID	Name
3	4	34	df4	23242.0	miraj

In [155…
```
#To check multiple conditions
df[(df['Age'] == 31) & (df['SN']>5)]
```

Out[155]:

	SN	Age	Roll	Employee ID	Name
5	6	31	F08	NaN	Anna

In [156…
```
df
```

Out[156]:

	SN	Age	Roll	Employee ID	Name
0	1	32	A01	100234.0	mmd
1	2	5	B07	102345.0	Jane Smith
2	3	42	C03	105678.0	David Brown
3	4	34	df4	23242.0	miraj
5	6	31	F08	NaN	Anna

DELETE SPECIFIC ROW AND COLUMN

In [157…
```
df.index=['1st','2nd','3rd','4th','5th']
df
```

Out[157]:

	SN	Age	Roll	Employee ID	Name
1st	1	32	A01	100234.0	mmd
2nd	2	5	B07	102345.0	Jane Smith
3rd	3	42	C03	105678.0	David Brown
4th	4	34	df4	23242.0	miraj
5th	6	31	F08	NaN	Anna

In [158…
```
#To delete the specific row
df.drop(labels='1st') # miraj vako row hatyo
```

Out[158]:

	SN	Age	Roll	Employee ID	Name
2nd	2	5	B07	102345.0	Jane Smith
3rd	3	42	C03	105678.0	David Brown
4th	4	34	df4	23242.0	miraj
5th	6	31	F08	NaN	Anna

In [159…
```
#To delete the multiple specific row
df.drop(labels=['1st','4th'])
```

Out[159]:

	SN	Age	Roll	Employee ID	Name
2nd	2	5	B07	102345.0	Jane Smith
3rd	3	42	C03	105678.0	David Brown
5th	6	31	F08	NaN	Anna

In [160…
```
df
```

Out[160]:

	SN	Age	Roll	Employee ID	Name
1st	1	32	A01	100234.0	mmd
2nd	2	5	B07	102345.0	Jane Smith
3rd	3	42	C03	105678.0	David Brown
4th	4	34	df4	23242.0	miraj
5th	6	31	F08	NaN	Anna

In [161…
```
#To delete the specific column
df.drop(columns='Name')
```

Out[161]:

	SN	Age	Roll	Employee ID
1st	1	32	A01	100234.0
2nd	2	5	B07	102345.0
3rd	3	42	C03	105678.0
4th	4	34	df4	23242.0
5th	6	31	F08	NaN

In [162…
```
df
```

Out[162]:

	SN	Age	Roll	Employee ID	Name
1st	1	32	A01	100234.0	mmd
2nd	2	5	B07	102345.0	Jane Smith
3rd	3	42	C03	105678.0	David Brown
4th	4	34	df4	23242.0	miraj
5th	6	31	F08	NaN	Anna

In [163…
```
#To delete the specific column or row by another axis methhos ( 0---> row     1-----> column)
#deleting column
df.drop('Age', axis=1)
```

Out[163]:

	SN	Roll	Employee ID	Name
1st	1	A01	100234.0	mmd
2nd	2	B07	102345.0	Jane Smith
3rd	3	C03	105678.0	David Brown
4th	4	df4	23242.0	miraj
5th	6	F08	NaN	Anna

In [164…
```
df
```

Out[164]:

	SN	Age	Roll	Employee ID	Name
1st	1	32	A01	100234.0	mmd
2nd	2	5	B07	102345.0	Jane Smith
3rd	3	42	C03	105678.0	David Brown
4th	4	34	df4	23242.0	miraj
5th	6	31	F08	NaN	Anna

In [165…
```
#deleting row
df.drop('4th', axis=0)
```

Out[165]:

	SN	Age	Roll	Employee ID	Name
1st	1	32	A01	100234.0	mmd
2nd	2	5	B07	102345.0	Jane Smith
3rd	3	42	C03	105678.0	David Brown
5th	6	31	F08	NaN	Anna

In [166…
```
df.drop(columns=['Age','Employee ID','Name'], inplace=True)

df
```

Out[166]:

	SN	Roll
1st	1	A01
2nd	2	B07
3rd	3	C03
4th	4	df4
5th	6	F08

To concat dataframes

In [170…
```
df1=pd.DataFrame({'Name':['miraj','aman'],'Roll':[10,11]})
df2=pd.DataFrame({'Name':['arbit','amir'],'Roll':[15,13]})
df3=pd.DataFrame({'Name':['june','july'],'Roll':[18,21]})

df=pd.concat([df1,df2,df3])
df
```

Out[170]:

	Name	Roll
0	miraj	10
1	aman	11
0	arbit	15
1	amir	13
0	june	18
1	july	21

In [174…
```
df=pd.concat([df1,df2,df3],keys=['group1','group2','group3'],names=['group','rowno'])
df
```

Out[174]:

		Name	Roll
group	rowno		
group1	0	miraj	10
	1	aman	11
group2	0	arbit	15
	1	amir	13
group3	0	june	18
	1	july	21

In [175…
```
df=pd.concat([df1,df2,df3],keys=['group1','group2','group3'],names=['group','rowno']).reset_index()
df
```

Out[175]:

	group	rowno	Name	Roll
0	group1	0	miraj	10
1	group1	1	aman	11
2	group2	0	arbit	15
3	group2	1	amir	13
4	group3	0	june	18
5	group3	1	july	21

String Operation in the dataframe

In [168…
```
import pandas as pd

# Create a sample DataFrame
df = pd.DataFrame({'name': ['John', 'Alice', 'Bob'],
                   'salary': ['10,000', '5,500', '8,750']})

# Remove commas from 'salary' column
#data frame ko sab element ko comma hatxa
df['salary'] = df['salary'].str.replace(',', '')

# Display the updated DataFrame
print(df)
```

```
    name salary
0   John  10000
1  Alice   5500
2    Bob   8750
```

In [ ]: