19/7/22

Python Pandas

Python pandas is python's library for data analysis. Python has derived its name from Panel Data Panda's System which is an econometrics term for multi-dimensional structured data sets.

Pandas was developed by Wes K McKinney.

Pandas is an open source, BSD library B(Berkeley Source Distribution)

We import external libraries of python in our ~~cod~~ program with the help of import Keyword.

For example :-

```
import pandas as pd
```

import ↓ Keyword — pandas ↓ name of the library — as — pd ↓ Alias name

Command to install pandas library

```
pip install pandas
```

Advantages of Pandas

(i) It can read or write data in different formats

(ii) ~~Calcul~~
It can calculate in all possible ways data is organized i.e. across rows and down the columns.

(iii) It can easily select subsets from bulky data

(iv) It supports reshaping of data.

(v) It supports advanced time series functionality.

(vi) It supports visualization by integrating matplotlib and seaborn libraries

Pandas Data Structure

Data Structure refer to specialize way of storing data so, as to apply specific type of functionality on them.

In Pandas data structures can be divided into three different types :-

(i) Series - One D

(ii) DataFrame - 2D

(iii) Panel - nD

Difference between Series and DataFrame

Series Object

Index	Data
1	'A'
2	'B'
3	'C'
4	'D'
5	'E'

~~DataFrame~~

DataFrame Object

Index	A	B	C
0	'Hello'	'Coulomn B'	NaN
1	'No INFO'	No	'No INFO'
2	'A'	'Column B'	NaN
3	'A'	'Column B'	NaN
4	'A'	'Column B'	NaN

	Series	DataFrame
(i)	1 - dimensional	2 - dimensional
(ii)	Homogeneous	Heterogeneous
(iii)	Value mutable	value mutable
(iv)	size immutable	size mutable

Series Data Structure

Series is an important data structure of pandas it represents one dimensional array of indexed data.

A series type object have two main component

(i) An array of actual data

(ii) An unassociated array of indexes and data labels

Creating Series object.

(i) import pandas as pd

```
data = [10, 20, 30, 40, 50]

s1 = pd.Series(data)
print(s1)
```

Output :-

```
0    10
1    20
2    30
3    40
4    50

dtype : int64
```

Q. Write a Python Program to create a series to store the name of your favourite fruits and print it.

=)

```
import panda as pd

a = ['Apple', 'Mango', 'Banana', 'Guava']

S1 = pd.Series(a)

print(S1)
```

Different arguments that can be passed in series.

```
import pandas as pd
import numpy as np

data = [10, 20, 30, 40, 50]

S1 = pd.Series(data, index = ['a', 'b', 'c', 'd', 'e'],
     dtype = np.float64, name = 'marks')

print(S1)
```

Output:-

```
a    10.0
b    20.0
c    30.0
d    40.0
e    50.0

Name: marks, dtype: float64
```

Q. Creating non empty series object.

* Data inside series object can be passed in the form of lists, tuples

(i) lists
(ii) tuples
(iii) Strings
(iv) Numpy Array
(v) Dictionaries
(vi) Scalen Values

1 Create the List of your hobbies in a series by a tuple

2. Store your Name in Series Data Structure

Soln 1

```
Import pandas as pd
a = ('reading', 'sleeping')
s1 = pd.Series(a)
print(s1)
```

Soln 2.

```
Import pandas as pd
a = {'Gaurav'}
s1 = pd.Series(a)
print(s1)
```

Qs. Create a series to store the name of the countries and their capitals via dictionary

Sol^n

```
import pandas as pd
a = {'India' : 'New Delhi', 'China' : 'Beijing',}
s1 = pd.Series(a)
print(s1)
```

Qs Write the output of the previous program.

Sol^n

```
India    New Delhi
China    Beijing.
dtype: object
```

Creating Non Empty Series Object

(i) Creating Series in sequences

(a) Creating series using lists

```
import pandas as pd
a = [1, 2, 3, 4, 5, 6]
s1 = pd.Series(a)
print(s1)
```

(b) Creating Series using tuple

```
import pandas as pd
b = (1, 2, 3, 4, 5)
s1 = pd.Series(b)
print(s1)
```

(c) Creating Series using String

```
import pandas as pd
c = {"a", "b", "c"}
s1 = pd.Series(c)
print(s1)
```

Specifying data as an ndarray

Numpy Array

Numpy stands for numerical python is a python's library for numerical data analysis.

Numpy stores data in homogeneous order

To import numpy, the following line is used:

→ import numpy as np

★ Differences b/w List and array

List	Array
1) List can be elements of different data type for eg - [1, 2, 3, 4, 'hello'] as @	1) All elements of an array are of the same data type eg - [1, 2, 3, 4]
2) Elements of a list are not stored configuously in memory.	2) Array elements are stored in contiguous memory location. This make operations faster than lists.

3) Lists do not support memory wise Cooperations for eg - Addition, Multiplication etc because elements may not be of same type.

3) Array support element wise operations. For eg. If A_1 is an array it is possible to say $A_1/3$ divide each element of the array by it.

4) List can contain objects of different datatypes that python must store the type information for every element along with its element value. Thus, list take more space in memory and less efficient

4) Numpy array takes up less space in memory as compared to a list because array does not require to store datatype of each element seperately.

21/7/22

Creating a Numpy Array from a list

```
import numpy as np
data = [1, 3, 5, 7, 9]
arr = np.array(data)
print(arr * 4)
```

Create a Numpy array using tuples

```
import numpy as np
data = (1, 2, 3, 4)
arr = np.array(data)
print(arr * 2)
```

Attributes of Numpy Array

(i) ~~ndarray.nd~~
Attributes gives us the info of the data object.

(i) ndarray.ndim : It will give the dimension of the array
for example :-

```
import numpy as np
data = [1, 2, 3, 4]
arr = np.array(data)
print(arr.ndim)
```

(ii) ndarray.shape : It will give the number of elements in rows and columns

Example:

```
import numpy as np
data = [1, 2, 3, 4, 5, 6]
arr = np.array(data)
print(arr)
print(arr.shape)
```

Excessing array elements

One-D

```
import numpy as np
data = [1, 2, 3, 4]
arr = np.array(data)
print(arr)
print(arr[1])
```

Two-D

```
import numpy as np
data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
arr = np.array(data)
print(arr)
print(arr[1, 2])
```

(iii) ndarray.size : It gives the total number of elements

```
import numpy as np
data = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12],
        [13, 14, 15]]
arr = np.array(data)
print("The 2D Array:", "\n", arr)
```

```
print ("No of element in 2D Array :", arr.size)

print("No of rows and cols in 2D Array :",
                                    arr.shape )

print("Dimension of the array", arr.ndim)
```

Output :

```
The 2D Array :

[[1, 2  3 ]
 [4, 5  6 ]
 [7  8  9 ]
 [10 11 12 ]
 [13 14 15 ] ]

No of element in 2D Array : 15
No of rows and cols in 2D Array : (5, 3)
Dimension of the array : 2
```

(iv) ndarray.dtype : Returns data type of an array.

:)
```
import numpy as np
data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
arr = np.array(data)
print ("Data type of the array is :", arr.dtype)
```

Output :- Data type of the array is : int64

(v) ndarray.itemsize (returns the size in bytes)
size int = 8 //

:)
```
import numpy as np
data : [[1, 2, 3], [4, 5, 6]]
arr = np.array(data)
print ("Item size :" arr.itemsize)
```

Output :- Item size : 8

Operations on Numpy Array

(i) Addition of array elements

```
import numpy as np

data = [[1, 2, 3], [4, 5, 6], [1, 2, 3]]

arr = np.array(data)

arr2 = np.array([[1, 2, 3], [1, 1, 1], [1, 1, 1]])

print(arr + arr2)
```

Output :-

```
[[2, 4, 6],
 [5, 6, 7]
 [2, 3, 4]]
```

(ii) Subtraction of array

```
import numpy as np

data = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr1 = np.array(data)
data2 = [1, 1, 1, 1]
arr2 = np.array(data2)
print(arr1 - arr2)
```

```
[[0, 1, 2, 3]
 [4, 5, 6, 7]]
```

Q. Create two 2-D arrays and perform the following operations

(i) Addition
(ii) Subtraction
(iii) Multiplication
(iv) Division
(v) Modulus

and write the output of all the operations perf

=)

```
import numpy as np

data = [[1, 2, 3], [4, 5, 6]]

arr1 = np.array(data)
arr2 = np.array [[1, 1, 1], [1, 1, 1]]
```

(i)

```
print (arr1 + arr2)
```

Output :

```
[[2 3 4]
 [5 6 7]]
```

(ii)

```
print (arr1 - arr2)
```

Output :

```
[[0 1 2]
 [3 4 5]]
```

(iii)

```
print (arr1 * arr2)
```

Output :-

```
[[1 2 3]
 [4 5 6]]
```

(iv) Division

print(arr1 / arr2)

Output :- [[1 2 3]
[4 5 6]]

(v) Modulue

print(arr1 % arr2)

Output :- [[0 0 0]
[0 0 0]]

Statistical Operations on Arrays

The following statistical functions are used to peaform the operations

(i) mean()
(ii) median()
(iii) mode()
(iv) min()
(v) max()
(vi) sum()
(vii) std()

Specifying data as ndarray

```
import numpy as np
import pandas as pd

arr1 = np.arange(2, 10, 2)
s1 = pd.Series(arr1)
print(s1)
```

Output :-

```
0    2
1    4
2    6
3    8

dtype: int64
```

Specifying data as python dictionary

```
import pandas as pd
stu = {'A': 39, 'B': 41, 'C': 42, 'D': 44}

S8 = pd.Series(stu)
print(S8)
```

Output :-

```
A    39
B    41
C    42
D    44
```

Custom index

```
import numpy as np
import pandas as pd
arr1 = np.arrange(2, 10, 2)

S1 = pd.Series(arr1, index = ['Ram', 'Alex', 'Sam',
                              'Mukesh'])

print(s1)
```

Output :-

```
Ram      2
Alex     4
Sam      6
Mukesh   8

dtype : int64
```

Custom datatype

```
import numpy as np
import pandas as pd
arr1 = np.arange(2, 10, 2)
S1 = pd.Series(arr1, index = ['Ram', 'Alex', 'Sam',
     'Mukesh'], dtype = np.float64).
```

Output :-

```
Ram      2
Alex     4
Sam      6
Mukesh   8
dtype: float64
```

20/8/22

Specifying NaN values in series (Not a Number)

```
import numpy as np
import pandas as pd
data = (10, 20, 30, 40, 50, np.NaN)

S1 = pd.Series(data)
print(S1)
```

Output

```
0    10.0
1    20.0
2    30.0
3    40.0
4    50.0
5     NaN

dtype: float64
```

Attributes of Series

Series attributes : Attributes helps us to get the information of a datastructure

Attributes of Series objects :

Attribute	Description
{Series object}.index	The index (axis labels) of the Series.
{Series object}.values	Return Series as ndarray or ndarray like depending on the dtype
{Series object}.dtype	return the dtype of the underlying data.
{Series object}.shape	return a tuple of the shape of the underlying data
{Series object}.nbytes	return the number of bytes in the underlying data
{Series object}.ndim	return the number of dimensions of the underlying data.
{Series object}.size	return the number of elements in the underlying data
{Series object}.itemsize	return the size of the datatype of the item of the underlying data
{Series object}.hasnans	Return 'True' if there are any NaN value : Otherwise returns false.
{Series Object}.empty	Return 'True' if the series object is empty otherwise returns false.
Name	Assigning name to the series object
Index.name	Returns the name of the index

Head and Tail function of Series

By default the Head and Tail funct^{n} is going to return 5 - elements if you don't pass the argument

Head function returns the top elements
tail function returns the bottom elements.

*
```
import pandas as pd
data = [10, 20, 30, 40, 50, 60, 70, 80]
S1 = pd.Series(data, name="JoJo")
Si.index.name = "KoKo"
print(S1.tail(3))
```

Output :-

KoKo	JoJo
0	10
1	20
2	30
3	40
4	50
5	60
6	70
7	80

KoKo	JoJo
5	60.0
6	70.0
7	80.0

Vector Function

- Addition of Series

```
import pandas as pd
d1 = [1, 2, 3, 4, 5]
d2 = [1, 2, 3, 4, 5]
S1 = pd.Series [d1]
S2 = pd.Series [d2]
print (S1 + S2)
```

Output →

0	2
1	4
2	6
3	8
4	10
dtype	Int64

- Addition of series with NaN

```
import numpy as np
import pandas as pd
d1 = [1, 2, 3, 4, np.NaN]
d2 = [1, 3, 3, 4, 5]
S1 = pd.Series (d1)
S2 = pd.Series (d2)
print (S1 + S2)
```

Output →

0	2
1	4
2	6
3	8
4	NaN
dtype	float64

- Addition with custom indexes

```
import pandas as pd
d1 = [1, 2, 3, 4, 5], index = [1, 2, 3, 4, 5]]
d2 = [1, 2, 3, 4, 5 index = [1, 2, 3, 4, 5]]
S1 = pd.Series (d1)
S2 = pd.Series (d2)
```

```
import pandas as pd
data = [10, 20, 30, 40]
S1 = pd.Series(data, index = ['a', 'b', 'c', 'd'])
data2 = [12, 14, 11, 10]
S2 = pd.Series(data2, index = ['d', 'a', 'b', 'c'])
print(S1 + S2)
```

Output

```
a    24
b    31
c    41
d    52

dtype: Int64
```

```
import pandas as pd
dat
```

Filtering Entries

```
import pandas as pd
data = [10, 20, 30, 40, 50, 60]
S1 = pd.Series(data, index = ['a', 'b', 'c', 'd', 'e', 'f'])
print(S1 > 20)
print(S1[S1 > 20])
```

Output :-

```
a   false
b   false
c   True
d   True
e   True
f   True

c
d
e
f
```

Sorting Series Values

```
import pandas as pd
data : [10, 30, 20, 40]
S1 = pd.Series(data, index = ['a', 'b', 'c', 'd'])
S1.sort_values(ascending = False, inplace = True)
print(S1)
```

Output

a	10	d	40
b	20	c	30
c	30	b	20
d	40	a	10

dtype: int64

If we use inplace the output will be of the same object.

Sort_index

Sorting on the basis of indexes

```
import pandas as pd
data = [10, 20, 30, 40, 50]
S1 = pd.Series(data)
S1.sort_index(ascending = False, inplace = True)
print(S1)
```

Output

```
4    50
3    40
2    30
1    20
0    10

dtype: int64
```

Reindexing of Series

```
import pandas as pd
data = [10, 20, 30, 40, 50]
S1 = pd.Series(data)
S1 = S1.reindex([4, 1, 2, 3, 0])
print(S1).
```

Dropping Entries

We can drop Rows
We can delete Columns.

```
import pandas as pd
data = [10, 20, 30, 40, 50]
S1 = pd.Series(data)
print
S1 = S1.reindex
S1.drop(index = [4], inplace = True)
print(S1)
```

Output :-

```
0    10
1    20
2    30
3    40
dtype : int64
```

27/8/22

DataFrame

Dataframe is a two-dimensional data structure that stores an ordered collection of data.

Characteristics of a dataframe

(i) It has two axes : i.e. (axis=0) and (axis=1)

Axis 0 signifies rows
Axis 1 signifies columns.

(ii) Stores data like a spreadsheet

(iii) Indexes can be of numbers, letters or strings.

(iv) Data can be of different type across the rows

(v) Values and Size is mutable.

27/8/22

Creation of dataframe using 2D dictionary

```
import pandas as pd

data = [{
    'Name' : ['Jai', 'Ritesh'],
    'Class' : ['VI', 'VII'],
    'Marks' : [99, 100]}]

df = pd.DataFrame(data, index = ['R1', 'R2',])

print(df)
```

Output

	Name	Class	Marks
R1	Jai	VI	99
R2	Ritesh	VII	100

27/8/22

Creating Dataframes from list

```
import pandas as pd
data = [
    ["Jai", 45, "10-10-1995", 40000],
    ["Bai", 50, "15-15-1996", 100000],
    ["Gang", 17, "16-16-1999", 50000)
    ]
df = pd.DataFrame(data), column = ["Name", "Age", "Date",
print(df)                                  "Salary"])
```

	Name	Age	Date	Salary
0	Jai	45	10-10-1995	40000
1	Bai	50	15-15-1996	100000
2	Gang	17	16-16-1999	50000

Creating Dataframe from Numpy Array

```
import pandas as pd
import numpy as np
arr = np.array([[1,3,3,4], [1,2,3,4], [1,2,3,4]])
df = pd.DataFrame(arr)
print(df)
```

Output :-

	0	1	2	3
0	1	2	3	4
1	1	2	3	4
2	1	2	3	4

27/8/22

Creating Dataframe from Series

```
import pandas as pd
import numpy as np
staff = pd.Series([28, 36, 44])
Salaries = pd.Series([279000, 396000, 563000])
avg = Salaries / staff
org = {'People': staff, 'Amount': salaries, 'Average': avg}
dtfs = pd.Dataframe(org)
print(dtfs)
```

	Amount	Average	People
0	279000	1395.0	20
1	396000	11022	36
2	563000	1279	44

Creating Dataframe from Another dataframe

```
import pandas as pd
import numpy as np
arr = np.array([[1,2,3,4],[1,3,3,4],[1,3,3,4]])
print(arr)
df = pd.DataFrame(arr)
df2 = pd.DataFrame(df)
print(df2)
```

	0	1	2	3
0	1	2	3	4
1	1	2	3	4
2	1	2	3	4

30/8/22

Dataframe Attributes

Attributes help us to give the information of a data structure

(i) index : The index (row labels) of the Dataframe.

(ii) columns: The column labels of the Dataframe.

(iii) axes : Returns ~~a list~~ representing both axes (axis 0 i.e. index and axis 1 i.e., columns) of the Dataframe.

(iv) dtypes : Return the dtypes of data in the Dataframe.

(v) size : Returns an int representing no of elements in this object

(vi) shape : Returns a tuple representing dimensions of the dataframe.

(vii) values : Returns a Numpy representation of the Dataframe

(viii) empty : Indicates whether a dataframe is empty

(ix) ndim. Returns an int representing number of axes/ array dimensions

(x) T Transpose index and columns.

30/8/22

(1) Index

```
import pandas as pd
data = {"name" : ['Alex', 'Mohan', 'Rohan', 'Bushan'],
        'Marks" : [20, 30, 40, 50] }

df = pd.DataFrame(data, index = ["a", "b", "c", "d"])
print(df)
print(df.index)
print(df.columns)
print(df.axes)
print(df.dtypes)
print(df.size)
print(df.shape)
print(df.values)
print(df.empty)
print(df.ndim)
print(df.T)
```

Output :-

	Name	Marks
a	Alex	20
b	Mohan	30
c	Rohan	40
d	Bushan	50

Index (["a", "b", "c", "d"], dtype : 'object')

Columns (["Name", "Marks"], dtype : 'object')

30/8/22

[Index (['a', 'b', 'c', 'd'], dtype = 'object'),
Index (['Name', 'Marks'], dtype = 'object')]

name object
Marks int64
dtype : object
8
(4, 2)

[['Alex' 20]
['Mohan' 30]
['Rohan' 40]
['Bushan' 50]]

False
2

	a	b	c	d
name	Alex	Mohan	Rohan	Bushan
Marks	20	30	40	50

★ Selecting or Accessing Data

① Selecting or accessing column

⇒
```
import pandas as pd
d = {"Name": ["Jai", "Rishi"], "Sec": ["a", "b"],
     "Class": [12, 11]}

df = pd.DataFrame(d)
```
(i) print(df["Class"])
(ii) print(df.Class)

2/9/22

	Class
0	12
1	11

	Class
0	12
1	11

② Selecting or Accessing Columns

=) import pandas as pd

d = { "Name" : ["Jai", "Rishi"], "Sec" : ["a", "b

"Class" : [12, 11] }

df = pd.DataFrame(d)

(i) print(df["class"])

	Class
0	12
1	11

② Selecting or Accessing Multiple columns

print(df[["Name", "class"]])

	Name	Class
0	Jai	12
1	Rishi	11

2/9/22

③ Selecting / Accessing a subset from a DataFrame using row / column Name

→ Syntax of using [.loc]

print (df. loc [start row : stop row, start col : stop col])

Example print(df. loc [0:1, "Item- Category": "Expenditure"])

Output

	Item_ Category	Item - name	Expenditure
0	book	IP	87
1	pen	flair	92
2	book	Physics	73

* printing a particular row [using. loc]

print (df. loc [1:1, "Item_ Category" : "Expenditure"])

On

print (df. loc [1, "Item - Category" : "Expenditure"])

∴ When start or Stop value is not given it takes the starting and ending value respectively by default.

* Using [.iloc]

print(df.iloc [0:4, 0:])

:) ending value should be given by adding (+1) to it as the stop will end the program before the value.

3/9/22

* print only the 4th row using .iloc

```
print(df[4, 0:])
```

OR

```
print(df[4:5, 0:])
```

* Selecting / Accessing individual values

```
print(df.Item_Category[0])
```

OR

```
print(df["Item_Category"][0])
```

* Difference b/w (at) and (iat)

Use

1. <Df object>.at[<row label>, <col label>]
2. <Df object>.iat[<row index number>, <col index no>]

To

1. Access a single value for a row/column label pair.
2. Access a single value for a row/column by integer position

* Selecting / Accessing Individual value using [at][iat]

at :- print(df.at[3, "Item-Category"])

iat :- print(df.iat[3,0])

3/9/22

* Selecting DataFrame Rows / Columns based on conditio

→ print (df["Expenditure"] > 50)
it will only give boolean result

→ print (df [df ["Expenditure"] > 50])

* Adding / modifying Rows / columns in a DataFrame

① Adding / modifying a Column

→ <Df name> [<Column name>] = [values]

df ["Quantity"] = [1, 2, 3, 4, 5, 6, 7, 8]

print (df) Here all the values are diff

→ df["Quantity"] = 200
print (df) : Here all the values are [200]

Modifying a Column

→ <Df name> [<column name>] = [new values]
→ df ["Expenditure"] : [1, 2, 3, 4, 5, 6, 7, 8]

4/9/22

② Adding / Modifying Rows

(i) Adding

df.loc[8, :] = ["mobile", "samsung", 198]

<Df object>.loc[New index label, :] = [values]

(ii) Modifying

<df object>.loc[<existing row index label>, :] = [value to change]

df.loc[4, :] = ["tv", "samsung", "1000"]

* Modifying a single cell

→ <df object>[Column name][Row label] = "Value"

df["Item Name"][2] = "Astrophysics"

→ <df object>.at[<Row label>, <column name>] = "Value"

df.at[2, "Item Name"] = "Astro Physics"

* Deleting or Renaming Column / Rows

(i) Deletion

Deleting Column

→ del <df object>["column name"]

del df["Expenditure"]

4/9/22

Dropping column

→ <df object>.drop [columns = [<"column name">],
axis = 1, inplace = True]

* Drop can delete both multiple rows and columns
Del can only delete column

Dropping Rows

<df object>.drop [index = [<index labels>],
axis = 0, inplace = True]

Renaming

for col → <df object>.rename (column = {"old name":
"new name"},
inplace = True)

for rows → <df object>.rename [index = { "old label":
"New label" }, inplace = True]

* Boolean Index in DataFrame

<df. object> = pd.DataFrame (data, index = [True, False])

* Filtering Data with help of Boolean Indexes
print (<df object>.loc [True, False])

* Using iter [rows]

→ For i in df.iterrows():
print (i)

6/9/22

Output:- The output will give the data of a row in a tuple form, till the no of row are present.

→ for i in df.iterrows():
print(i[0]) | print(i[1]) | print(i[1][0/1/2])

the rows and columns are stored in rows and columns.
as [0,1] index.
Iterrows prints the data row wise in the form of tuples
for example :-

(0, Item_Category → book [1][0]
element 0 of tuple
ItemName ip → [1][1]
Expenditure 87 → [1][2]
element 1 of tuple

(Name : <df name>, dtype : object)
↳ Attributes of tuple

→
```
import pandas as pd
data = {
    0 : {'a': 1 , 'b': 2},
    1 : {'a': 55 , 'b': 99}
}
df = pd.DataFrame(data)
print(df)
```

classmate Date Page 103

Output:

→

	0	1
a	1	2
b	55	99

* Inner Key will become the indexes / Row labels.

Q. i) Create a DataFrame to store the following details :
(i) Car Manufacturers (ii) Models (iii) Price (iv) Total Sales

The dictionary should have 5 minimum rows
Perform the following operation after creating dictionary

(i) Add a details of a new manufacturer
(ii) Add a column named 'Profit'
(iii) Delete the column Total Sales
(iv) Delete the row with index position 4.

```
import pandas as pd
data = {
  "Car_Mfgs" : ['Kia', 'Tata', 'Hyundai', 'Maruti',
                'Suzuki', 'SKoda'],
  "Models" : ['Centar', "Nano", "110", "800", "Rapid"],
  "Price" : [400000, 100000, 500000, 100000, 800000],
  "Total Sales" : [80, 70, 85, 60, 90] }

df = pd.DataFrame(data)
print(df)
```

8/9/22

Output →

	Car_Mfgs	Models	Price	Total Sales
0	Kia	Centar	400000	80
1	Tata	Nano	100000	70
2	Hyundai	i10	500000	85
3	Maruti Suzuki	800	100000	60
4	SKoda	Rapid	600000	90

(i) `df.loc[5, :] = ['Mahindra', 'SUV-700', 700000, 75]`

(ii) `df["Profit"] = [20, 30, 60, 10, 40, 60]`

(iii) `del df["Total Sales"]`

(iv) `df.drop(index=[4], axis=0, inplace=True)`

(v) `df.drop(columns=["Total Sales"], axis=1, inplace=True)`

	Car_mfgs	Models	Price	Profit
0	Kia	Centas	400000	20
1	Tata	Nano	100000	30
2	Hyundai	i10	300000	60
3	Maruti Suzuki	800	100000	10
4	SKoda	Rapid	600000	40
5	Mahindra	SUV 700	700000	60

Q (i) Update the car manufacturer at index position 2 to BMW.

(ii) Display the details of all the manufacturers whose profit is more than 30.

(iii) Update all the column names to C_1, C_2, C_3, C_4,

Ans (i) `df[Car_Mfgs][2] = "BMW"`

(ii) `print(df[df["Profit"] > 30])`

1/9/22

(iii) df. rename (columns = {"Car_Mfgs" : "C1", "Models" : "C2", "Price" : "C3", "Profit" : "C4" },
inplace = True)