=====================================

Pandas

======================================

Introduction to Pandas:

----------------------------------

=>Pandas is an open-source Python Library / Module providing high performance and data manipulation and Analysis Tool.

=>The word PANDAs derived from PANel DAta

=>The pandas concept developed by WES MCKinney in the year 2008.

=>The Traditional Python Programming does not contain any Module for Data Analysis and Now Python Programming uses Pandas as an anaysis tool.

=>Python Pandas can be used in wide range of fields like Finance Services, Statistics, retail maketing sectors..etc

=>pandas module developed in C and Python Languages.

-----------------------------------------------

Instalation of Pandas:

-----------------------------------------------

=>The standard python software / Distribution (CPYTHON) does not contain any module for data analysis and now we are using third party module called PANDAS and whose module name is pandas

=>Programmatically to use pandas as part of our python program, we must install panda’s module by using pip tool.

Syntax: - pip install module name

Example: - pip install pandas

-----------------------------------------------------------------------

Key Features of Pandas: -----> Series DataFrame

-----------------------------------------------------------------------

1) Fast and Efficient Data Frame with default costomized indexing

2) Tools for loading the data in in-memory data objects(objects of Series, DataFrame as Panels)

3) We can access the data from pandas by using Labeled Based Slicing and indexing.

4) Columns from in-memory data objects(objects of Series, DataFrame as Panel) can be deleted and inserted and updated

=========================================

Data Structures used in Pandas

=========================================

=>In Pandas programming, we can store the data in 2 types of Data structures. They are.

a) Series

b) DataFrame

=>The best of way of thinking of these data structures is that The higher dimensional Data Structure is a container of its lower dimensional data structure.

Examples:

--------------

=>Series is part of DataFrame.

------------------------------------

====================================

Series

====================================

=>It is a One-Dimensional Labelled Array Capable of Storing / Holding Homogeneous data of any type (Integer, String, float, .........Python objects etc).

=>The Axis Labels are collectively called Index.

=>Pandas Series is nothing but a column value in excel sheet.

=>Pandas Series Values are Mutable.

=>Pandas Series contains Homogeneous Data ( Internally even we store different types values , They are treated as object type)

-------------------------------------------------------------------- Creating a Series

-----------------------------------------------------------------------

=>A Series object can be created by using the folowing Syntax:

Syntax:-

--------------

varname=pandas.Series(object, index, dtype)

-------------------

Explanation:-

-------------------

=>Here varname is an object of <class, pandas.core.series.Series >

=>pandas are one of the pre-defined third party module name

=>Series () is pre-defined Function in pandas module and it is used for creating an object of Series class.

=>'object' can either int, float, complex, bool, str, bytes, bytearray,range, list,ndarray,dict .....etc (But not set type bcoz they are un-ordered)

=>'index' represents the position of values present Series object. The default value of Index starts from 0 to n-1, Here n represents number of values in Series object. Programatically we can give our own Index Values.

=>'dtype' represents data type (Ex:- int32, ,int64, float32, float64...etc)

-------------------------------------------------------------------------------------------------------

Examples: - Create a series for 10 20 30 40 50 60

>>> import pandas as pd

>>> import numpy as np

>>> lst=[10,20,30,40,50,60]

>>> s=pd.Series(lst)

>>> print(s,type(s))

0 10

1 20

2 30

3 40

4 50

5 60

dtype: int64 <class 'pandas.core.series.Series'>

----------------------------

>>> lst=[10,20,30,40,50,60]

>>> s=pd.Series(lst,dtype=float)

>>> print(s,type(s))

0 10.0

1 20.0

2 30.0

3 40.0

4 50.0

5 60.0

dtype: float64 <class 'pandas.core.series.Series'>

------------------------------------------------------------------

>>> lst=["Rossum","Gosling","Travis","MCKinney"]

>>> a=np.array(lst)

>>> a ------array (['Rossum', 'Gosling', 'Travis', 'MCKinney'], dtype='<U8')

>>> print(a, type(a))--['Rossum' 'Gosling' 'Travis' 'MCKinney'] <class 'numpy.ndarray'>

>>> s=pd.Series(a)

>>> print(s,type(s))

0 Rossum

1 Gosling

2 Travis

3 MCKinney

dtype: object <class 'pandas.core.series.Series'>

----------------------------------------------------------------------

>>>lst=[10,"Rossum",34.56,"Author"]

>>> s=pd.Series(lst)

>>> print(s,type(s))

0 10

1 Rossum

2 34.56

3 Author

dtype: object <class 'pandas.core.series.Series'>

--------------------------------------------------------------------

Creating an Series object with Programmer-defined Index

-----------------------------------------------------------------------

>>> lst=[10,"Rossum",34.56,"Author"]

>>> print(lst)--------[10, 'Rossum', 34.56, 'Author']

>>> s=pd.Series(lst,index=["Stno","Name","Marks","Desg"])

>>> print(s)

Stno 10

Name Rossum

Marks 34.56

Desg Author

dtype: object

>>> print(s["Stno"]) -------10

-----------------------------------------------------------------------

>>> lst=["Rossum","Gosling","Travis","MCKinney"]

>>> s=pd.Series(lst,index=[100,200,300,400])

>>> print(s,type(s))

100 Rossum

200 Gosling

300 Travis

400 MCKinney

dtype: object <class 'pandas.core.series.Series'>

-----------------------------------------------------------------------

Creating a Series object from dict

-------------------------------------------------------------------------

=>A dict object can be used for creating a series object

=>If we use dict object in Series() then keys can be taken as Indices (Or Indexes)

automatically and corresponding values of dict can be taken as Series data.

----------------

Examples:

---------------

>>> import pandas as pd

>>> d1={"sub1":"Python","sub2":"Java","sub3":"Data Science","sub4":"ML"}

>>> print(d1)--{'sub1': 'Python', 'sub2': 'Java', 'sub3': 'Data Science', 'sub4': 'ML'}

>>> s=pd.Series(d1)

>>> print(s)

sub1 Python

sub2 Java

sub3 Data Science

sub4 ML

dtype: object

>>> d2={"RS":2.3,"JG":1.2,"MCK":4.5,"TOLI":2.4}

>>> print(d2)---{'RS': 2.3, 'JG': 1.2, 'MCK': 4.5, 'TOLI': 2.4}

>>> s=pd.Series(d2)

>>> print(s)

RS 2.3

JG 1.2

MCK 4.5

TOLI 2.4

dtype: float64

===============================

Counter

===============================

Index

----------

=>What is Python Counter?

=>Why use Python Counter?

=>Counter with String

=>Counter with List

=>Counter with Dictionary

=>Counter with Tuple

=>Counter with Set

=>Accessing, Initializing and Updating Counters

=>Deleting an Element from Counter

=>Methods Available on Python Counter

=>Reassigning Counts in Python

=>Get and set the count of Elements using Counter

-----------------------------------------------

=>Python Counter is a container that will hold the count of each of the elements present in the container. The counter is a sub-class available inside the dictionary class.

-------------------------------------------------

Uses Python Counter

--------------------------

=>The Counter holds the data in an unordered collection.

=>The elements here represent the keys and the count as values.

=>It allows you to count the items in an iterable list.

=>Arithmetic operations like addition, subtraction, intersection, and union can be easily performed on a Counter.

=>A Counter can also count elements from another counter

----------------------------------------------------------

Syntax:

-------------

Varname=collections.counter(Iterable object)

=>Varname is an object of <class 'collections.Counter'>

=>Iterable object can be list, tuple, dictionary, string

---------------------------------------------------------

Example1:

from collections import Counter

list1 = ['x','y','z','x','x','x','y', 'z']

print(Counter(list1)) # Counter({'x': 4, 'y': 2, 'z': 2})

---------------------------------------------------------------------------------------

Example2:

from collections import Counter

dict1 = {'x': 4, 'y': 2, 'z': 2, 'z': 2}

print(Counter(dict1)) # Counter({'x': 4, 'y': 2, 'z': 2})

------------------------------------------------------------------------------

Example3

from collections import Counter

c=Counter("MIssisipi")

print(c) # Counter({'s': 3, 'i': 3, 'M': 1, 'I': 1, 'p': 1})

------------------------------------------------------------------------------

Example4:

from collections import Counter

tuple1 = ('x','y','z','x','x','x','y','z')

print(Counter(tuple1)) # Counter({'x': 4, 'y': 2, 'z': 2})

------------------------------------------------------------------------------

=>Methods Available on Python Counter

------------------------------------------

The methods available with Counter are:

1) elements () : This method will return you all the elements with count >0. Elements with 0 or -1 count will not be returned.

Examples:

----------

d = Counter(a = 2, b = 3, c = 6, d = 1, e = 5)

for i in d.elements():

print ( i, end = " ") # a a b b b c c c c c c d e e e e e

2) most\_common(value): This method will return you the most common elements from Counter list.

c=Counter({'x': 5, 'y': 12, 'z': -2, 'x1':0})

ce = c.most\_common(2)

print(ce) # [('y', 12), ('x', 5)]

print(common\_element)

3) subtract (): This method is used to deduct the elements from another Counter.

Examples:

from collections import Counter

c1 = Counter(A=4, B=3, C=10)

c2 = Counter(A=10, B=3, C=4)

c1.subtract(c2)

print(c1) # Counter({'c': 6, 'B': 0, 'A': -6})

4) update(): This method is used to update the elements from another Counter.

from collections import Counter

coun = Counter()

coun.update([1, 2, 3, 1, 2, 1, 1, 2])

print(coun) # Counter({1: 4, 2: 3, 3: 1})

coun.update([1, 2, 4])

print(coun) # Counter({1: 5, 2: 4, 3: 1, 4: 1})

5) total()---> This Function is used for adding all the occurences of Counter

Values.

Examples:

---------------------

s="MISSISSIPPI"

c=Counter(s)

print("content of c=",c) # content of c= Counter({'I': 4, 'S': 4, 'P': 2, 'M': 1})

t=c.total()

print("Total=",t) # Total=11

#Program for finding number of occurences of a String

#s="MISSISSIPPI" Output: M:1 I : 4 S : 4 P:2

#non-counter.py

s="MISSISSIPPI"

d={} # create an empty dict

for ch in s:

if ch not in d:

d[ch]=1

else:

d[ch]=d[ch]+1

else:

print("content of dict:",d)

print("----------------------------------")

for k,v in d.items():

print("\t{}--->{}".format(k,v))

#Program for finding number of occurences of a String

#s="MISSISSIPPI"

#counterex1.py

from collections import Counter

s="MISSISSIPPI" # here s is an object of str and it is considered as Iterable object

c=Counter(s) # here c is an object of <class, collections.Counter>

print("content of c=",c) # content of c= Counter({'I': 4, 'S': 4, 'P': 2, 'M': 1})

print("-----------------------------")

for on,ov in c.items():

print("\t{}--->{}".format(on,ov))

print("-----------------------------")

#counterex2.py

from collections import Counter as c

lst=[10,20,10,20,10,30,40,10,20,50,34,10] # Here lst is called list object

c1=c(lst)

print("content of c1=",c1)

print("------------------------------------")

for on in c1:

print("\t{}-->{}".format(on,c1.get(on)))

#counterex3.py

from collections import Counter

c1=Counter((10,20,"Python","Python",20,"Python",10))

print("content of c1=",c1)

print("------------------------------------")

for on in c1.keys():

print("\t{}-->{}".format(on,c1[on]))

#counterex4.py

from collections import Counter

c1=Counter({10:2,20:4,40:5,10:6})

print("content of c1=",c1)

print("------------------------------------")

for on in c1.keys():

print("\t{}-->{}".format(on,c1[on]))

#counterex5.py

from collections import Counter

c1=Counter(a=10,b=20,c=3,d=4)

print("content of c1=",c1)

print("------------------------------------")

for on in c1.keys():

print("\t{}-->{}".format(on,c1[on]))

#counterex6.py-----update()

from collections import Counter

s="MISSISSIPPI"

c=Counter(s)

print("content of c=",c) # content of c= Counter({'I': 4, 'S': 4, 'P': 2, 'M': 1})

print("------------------------------------")

s1="bitter"

c.update(s1)

print("content of c after update=",c) # content of c= Counter({'I': 4, 'S': 4, 'P': 2, 'M': 1})

for k,v in c.items():

print("\t{}-->{}".format(k,v))

c.popitem()

print("------------------------------------")

for k,v in c.items():

print("\t{}-->{}".format(k,v))

#counterex7.py-----most\_common(val) and total()

from collections import Counter

s="MISSISSIPPI"

c=Counter(s)

print("content of c=",c) # content of c= Counter({'I': 4, 'S': 4, 'P': 2, 'M': 1})

print("------------------------------------")

print("content of c after update=",c) # content of c= Counter({'I': 4, 'S': 4, 'P': 2, 'M': 1})

for k,v in c.items():

print("\t{}-->{}".format(k,v))

print("------------------------------------")

cm=c.most\_common(2)

print(cm) # [('I', 4), ('S', 4)]

print("------------------------------------")

t=c.total()

print("Total=",t) # Total=11

#counterex8.py

from collections import Counter

c1=Counter({10:2,20:4,40:5,10:3,25:-2,35:0})

print("content of c1=",c1)

print("------------------------------------")

for on in c1.keys():

print("\t{}-->{}".format(on,c1[on]))

print("---------------------------------------------")

for e in c1.elements():

print(e)

#counterex9.py

from collections import Counter

c=Counter() # Empty counter

print("content of c=",c)

c.update("AYAAN")

print("content of c after update=",c) # Counter({'A': 3, 'Y': 1, 'N': 1})

print('------------------------------------------------')

for v in c.elements():

print("\t{}".format(v))

c['A']=c.get('A')+3

print("content of c after update=",c) # Counter({'A': 3, 'Y': 1, 'N': 1})

#counterex10.py

from collections import Counter

c1=Counter("abaabaabc")

c2=Counter("ababcdef")

c3=c1+c2

print("counter1=",c1)

print("counter2=",c2)

print("Add of c1 and c2=",c3)

c4=c2-c1

print("Sub of c1 and c2=",c4)

c5=c1|c2 # Union Operation

print("content of c5 after Union=",c5)

c6=c1&c2

print("content of c6 after Intersection=",c6)

==========================================

DataFrame in Pandas

==========================================

=>A DataFrame is 2-Dimensional Data Structure to organize the data .

=>In Otherwords a DataFrame Organizes the data in the Tabular Format, which is nothing but Collection of Rows and Columns.

=>The Columns of DataFrame can be Different Data Types or Same Type

=>The Size of DataFrame can be mutable.

----------------------------------------------------------------------- ================================================

Number of approaches to create DataFrame

================================================

=>To create an object of DataFrame, we use pre-defined DataFrame() which is present in pandas Module and returns an object of DataFrame class.

=>We have 5 Ways to create an object of DataFrame. They are

a) By using list / tuple

b) By using dict

c) By using set type

d) By using Series

e) By using ndarray of numpy

f) By using CSV File (Comma Separated Values)

-----------------------------------------------------------------------

=>Syntax for creating an object of DataFrame in pandas:

-----------------------------------------------------------------------

varname=pandas.DataFrame(object,index,columns,dtype)

----------------

Explanation:

----------------

=>'varname' is an object of <class,'pandas.core.dataframe.DataFrame'>

=>'pandas.DataFrame()' is a pre-defined function present in pandas module and it is used to create an object of DataFrame for storing Data sets.

=>'object' represents list (or) tuple (or) dict (or) Series (or) ndarray (or) CSV file

=>'index' represents Row index and whose default indexing starts from 0,1,...n-1

where 'n' represents number of values in DataFrame object.

=>'columns' represents Column index whose default indexing starts from 0,1..n-1

where n number of columns.

=>'dtype' represents data type of values of Column Value.

=======================================================

Creating an object DataFrame by Using list / tuple

-----------------------------------------------------------------------------------

>>>import pandas as pd

>>>lst=[10,20,30,40]

>>>df=pd.DataFrame(lst)

>>>print(df)

0

0 10

1 20

2 30

3 40

------------------------------------

lst=[[10,20,30,40], ["RS","JS","MCK","TRV"]]

df=pd.DataFrame(lst)

print(df)

0 1 2 3

0 10 20 30 40

1 RS JS MCK TRV

--------------------------------------------

lst=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]

df=pd.DataFrame(lst)

print(df)

0 1

0 10 RS

1 20 JG

2 30 MCK

3 40 TRA

--------------------------------------------------

lst=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]

df=pd.DataFrame(lst, index=[1,2,3,4],columns=['Rno','Name'])

print(df)

Rno Name

1 10 RS

2 20 JG

3 30 MCK

4 40 TRA

-------------------------------------------

tpl=( ("Rossum",75), ("Gosling",85), ("Travis",65), ("Ritche",95),("MCKinney",60) )

df=pd.DataFrame(tpl, index=[1,2,3,4,5],columns=['Name','Age'])

print(df)

Name Age

1 Rossum 75

2 Gosling 85

3 Travis 65

4 Ritche 95

5 MCKinney 60

-------------------------------------------------------------------------------------

Creating an object DataFrame by Using dict object

--------------------------------------------------------------------

=>When we create an object of DataFrame by using Dict , all the keys are taken as Column Names and Values of Value are taken as Data.

-----------------

Examples:

-----------------

>>> import pandas as pd

>>> dictdata={"Names":["Rossum","Gosling","Ritche","McKinney"],"Subjects":["Python","Java","C","Pandas"],"Ages":[65,80,85,55] }

>>> df=pd.DataFrame(dictdata)

>>> print(df)

Names Subjects Ages

0 Rossum Python 65

1 Gosling Java 80

2 Ritche C 85

3 McKinney Pandas 55

>>> df=pd.DataFrame(dictdata,index=[1,2,3,4])

>>> print(df)

Names Subjects Ages

1 Rossum Python 65

2 Gosling Java 80

3 Ritche C 85

4 McKinney Pandas 55

----------------------------------------------------------------------

Creating an object DataFrame by Using Series object

----------------------------------------------------------------------

>>> import pandas as pd

>>> sdata=pd.Series([10,20,30,40])

>>> df=pd.DataFrame(sdata)

>>> print(df)

0

0 10

1 20

2 30

3 40

>>> sdata=pd.Series({"IntMarks":[10,20,30,40],"ExtMarks":[80,75,65,50]})

>>> print(sdata)

IntMarks [10, 20, 30, 40]

ExtMarks [80, 75, 65, 50]

dtype: object

>>> df=pd.DataFrame(sdata)

>>> print(df)

0

IntMarks [10, 20, 30, 40]

ExtMarks [80, 75, 65, 50]

>>> ddata={"IntMarks":[10,20,30,40],"ExtMarks":[80,75,65,50]}

>>> df=pd.DataFrame(ddata)

>>> print(df)

IntMarks ExtMarks

0 10 80

1 20 75

2 30 65

3 40 50

-----------------------------------------------------------------------------------------------

Creating an object DataFrame by Using ndarray object

----------------------------------------------------------------------------------------------

>>> import numpy as np

>>> l1=[[10,60],[20,70],[40,50]]

>>> a=np.array(l1)

>>> df=pd.DataFrame(a)

>>> print(df)

0 1

0 10 60

1 20 70

2 40 50

>>> df=pd.DataFrame(a,columns=["IntMarks","ExtMarks"])

>>> print(df)

IntMarks ExtMarks

0 10 60

1 20 70

2 40 50

------------------------------------------------------------------------------------------

e) By using CSV File(Comma Separated Values)

-----------------------------------------------------------------

import pandas as pd1

df=pd1.read\_csv("D:\KVR-JAVA\stud.csv")

print("type of df=",type(df)) #type of df= <class 'pandas.core.frame.DataFrame'>

print(df)

--------------------- OUTPUT--------------------

stno name marks

0 10 Rossum 45.67

1 20 Gosling 55.55

2 30 Ritche 66.66

3 40 Travis 77.77

4 50 KVR 11.11

-----------------------------------------------------------------------

=======================================================

Accesssing the Data of DataFrame

=======================================================

1) DataFrameobj.head(no.of rows)

2) DataFrameobj.tail(no.of rows)

3) DataFrameobj.describe()

4) DataFrameobj.shape

5) DataFrameobj [start:stop:step]

6) DataFrameobj["Col Name"]

7) DataFrameobj[ ["Col Name1","Col Name-2"...."Col Name-n"] ]

8) DataFrameobj[ ["Col Name1","Col Name-2"...."Col Name-n"]] [start:stop:step]

9) DataFrameobj.iterrows()

===================================================

Understabding loc() ----- here start and stop index Included and

Col Names can be used(but not column numbers]

--------------------------------------------------------------------------------------

1) DataFrameobj.loc[row\_number]

2) DataFrameobj.loc[row\_number,[Col Name,.........] ]

3) DataFrameobj.loc[start:stop:step]

4) DataFrameobj.loc[start:stop:step,["Col Name"] ]

5) DataFrameobj.loc[start:stop:step,["Col Name1", Col Name-2......."] ]

6) DataFrameobj.loc[start:stop:step,"Col Name1" : Col Name-n"]

------------------------------------------------------------------------------------------------------------

Understabding iloc() ----- here start index included and stop index excluded and

Col Numbers must be used(but not column names]

--------------------------------------------------------------------------------------

1) DataFrameobj.iloc[row\_number]

2) DataFrameobj.iloc[row\_number,Col Number.........]

3) DataFrameobj.iloc[row\_number,[Col Number1,Col Number2............] ]

3) DataFrameobj.iloc[row start:row stop, Col Start: Col stop]

4) DataFrameobj.iloc[row start:row stop:step, Col Start: Col stop:step]

5) DataFrameobj.iloc[row start:row stop,Col Number ]

6) DataFrameobj.iloc[ [row number1, row number-2.....] ]

7) DataFrameobj.iloc[ row start: row stop , [Col Number1,Col Number2............] ]

8) DataFrameobj.iloc[ : , [Col Number1,Col Number2............] ]

===================================================================

Adding new Column Name to Data Frame

===================================================================

1) dataframeobj['new col name']=default value

2) dataframeobj['new col name']=expression

=======================================================================

Removing Column Name from Data Frame

=======================================================================

1)dataframe.drop(columns="col name")

2)dataframe.drop(columns="col name",inplace=True)

=======================================================================

sorting the dataframe data

=======================================================================

1) dataframeobj.sort\_values(["colname"])

2) dataframeobj.sort\_values(["colname"],ascending=False)

===================================================================

knowing duplicates in dataframe data

=======================================================================

1) dataframeobj.duplicated()---------------gives boolean result

===================================================================

Removing duplicates from dataframe data

===================================================================

1) dataframeobj.drop\_duplicates()

2) dataframeobj.drop\_duplicates(inplace=True)

===================================================================

Data Filtering and Conditional Change / updations

===================================================================

1) dataframeobj.loc[ simple condition]

Ex: df.loc[ df["maths"]>75 ]

df.loc[df["maths"]>90 ,["name","maths"]]

2) dataframeobj.loc[ compound condition ]

Ex: df.loc[ (df["maths"]>60) & (df["maths"]<85) ]

Ex: df.loc[ (df["maths"]>95) & (df["maths"]<=99),["name","maths"] ]

---------------------

MOST IMP

---------------------

3) dataframeobj.loc[ (compund condition), ["Col Name"] ]=Expression

Ex: df.loc[ (df["percent"]>=60) & (df["percent"]<=80),["grade"] ]="First" # cond updattion.

-----------------------------------------------------------------------------------------------------------------------

To Export the DataFrame object data to the csv file

df.to\_csv("E:\KVR-PYTHON-7AM\PANDAS\studfinaldata.csv")

To Export the DataFrame object data to the txt file

df.to\_csv("E:\KVR-PYTHON-7AM\PANDAS\class\_10.txt")

(or)

df.to\_csv("E:\KVR-PYTHON-7AM\PANDAS\class\_10.txt",index=False)

(OR)

df.to\_csv("E:\KVR-PYTHON-7AM\PANDAS\class\_10.txt",index=False,sep="\t")

-----------------------------------------------------------------------------------------------------------------------

To read the data from EXCEL into dataframe object

dataframeobj=pandas.read\_excel("Absolute path of excel file")

Examples:

df=pd.read\_excel("D:\\KVR\\kvr.xlsx")

print(df)

--------------------------------------------------------------------------------------------------------------------------

==================================================

DataFrame--GroupBy

==================================================

=>The Group By mechanism in the Pandas provides a way to break a DataFrame into different groups or chunks based on the values of single or multiple columns.

=>Let’s understand with some examples.

=>Assume we have a DataFrame,

-----------------------------------------------------------------------

ID Name Age City Experience

-----------------------------------------------------------------------

11 Jack 44 Sydney 19

12 Riti 41 Delhi 17

13 Aadi 46 Mumbai 11

14 Mohit 45 Delhi 15

15 Veena 43 Delhi 14

16 Shaunak 42 Mumbai 17

17 Manik 42 Sydney 14

18 Vikas 42 Delhi 11

19 Samir 42 Mumbai 15

20 Shobhit 40 Sydney 12

----------------------------------------------------------------------------------------------------

=>This DataFrame has a column ‘City’ which has three unique values like, “Delhi”, “Mumbai” and “Sydney”. We want to create different groups out of this DataFrame based on the column “City” values.

=>As this column has only three unique values, so there will be three different groups.

=>Group 1 will contain all the rows for which column “City” has the value “Delhi” i.e.

----------------------------------------------------------------------------------------------------

ID Name Age City Experience

----------------------------------------------------------------------------------------------------

12 Riti 41 Delhi 17

14 Mohit 45 Delhi 15

15 Veena 43 Delhi 14

18 Vikas 42 Delhi 11

----------------------------------------------------------------------------------------------------

Group 2 will contain all the rows for which column “City” has the value “Mumbai” i.e.

----------------------------------------------------------------------------------------------------

ID Name Age City Experience

----------------------------------------------------------------------------------------------------

13 Aadi 46 Mumbai 11

16 Shaunak 42 Mumbai 17

19 Samir 42 Mumbai 15

----------------------------------------------------------------------------------------------------

Group 3 will contain all the rows for which column “City” has the value “Sydney” i.e.

ID Name Age City Experience

11 Jack 44 Sydney 19

17 Manik 42 Sydney 14

20 Shobhit 40 Sydney 12

-----------------------------------------------------------------------------------------------------------------

DataFrame.groupby() method

-----------------------------------------------------------------------------------------------------------------

DataFrame’s groupby() method accepts column names as arguments. Based on the column values, it creates several groups and returns a DataFrameGroupBy object that contains information about these groups.

For example, let’s create groups based on the column “City”,

# Create Groups based on values in column 'city'

groupObj = df.groupby('City')

print(groupObj)

-------------

Output

-------------

<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002895CA14048>

The groupby() function created three groups because column ‘City’ has three unique values. It returned a DataFrameGroupBy object with information regarding all three groups.

-----------------------------------------------------------------------Iterate over all the DataFrame Groups

-----------------------------------------------------------------------DataFrame’s groupby() function returns a DataFrameGroupBy object, which contains the information of all the groups. The DataFrameGroupBy is an iterable object. It means using a for loop, we can iterate over all the created Groups,

# Iterate over all the groups

for grpName, rows in df.groupby('City'):

print("Group Name: ", grpName)

print('Group Content: ')

print(rows)

------------------------------

Output:

-------------------------------------------------

Group Name: Delhi

Group Content:

Name Age City Experience

ID

12 Riti 41 Delhi 17

14 Mohit 45 Delhi 15

15 Veena 43 Delhi 14

18 Vikas 42 Delhi 11

Group Name: Mumbai

Group Content:

Name Age City Experience

ID

13 Aadi 46 Mumbai 11

16 Shaunak 42 Mumbai 17

19 Samir 42 Mumbai 15

Group Name: Sydney

Group Content:

Name Age City Experience

ID

11 Jack 44 Sydney 19

17 Manik 42 Sydney 14

20 Shobhit 40 Sydney 12

-------------------------------------------------------------------------------------------------------------------

Get first row of each Group

------------------------------------------------------------------------------------------------------------------

=>DataFrame’s groupby() function returns a DataFrameGroupBy object, which contains the information of all the groups. The DataFrameGroupBy object also provides a function first(), and it returns a DataFrame containing the first row of each of the Group.

-----------------------------------------------------------------------

Get nth row of each Group

-----------------------------------------------------------------------

=>The pandas.groupby().nth() function is used to get the value corresponding the nth row for each group. To get the first value in a group, pass 0 as an argument to the nth () function.

-----------------------------

For example

-----------------------------

# Get first row of each group

firstRowDf = df.groupby('City').first()

print(firstRowDf)

---------------------------------

Output:

---------------------------------

Name Age Experience

City

Delhi Riti 41 17

Mumbai Aadi 46 11

Sydney Jack 44 19

There were three unique values in the column “City”, therefore 3 groups were created. The first() function fetched the first row of each of the Group and returned a DataFrame populated with that. The returned DataFrame has a row for each of the city and it is the first row from each of the city groups.

-----------------------------------------------------------------------

Get the count of number of DataFrame Groups

------------------------------------------------------------------------------------------------------------------

The DataFrameGroupBy object also provides a function size(), and it returns the count of rows in each of the groups created by the groupby() function. For example,

# Get the size of DataFrame groups

print(df.groupby('City').size())

----------------

Output:

----------------

Delhi 4

Mumbai 3

Sydney 3

dtype: int64

As there were three unique values in the column “City”, therefore 3 groups were created by groupby() function. The size() function returned a Series containing the count of number of rows for each of the group.

#GroupByEx1.py

import pandas as pd

# List of Tuples

empoyees = [

(11, 'Jack', 44, 'Sydney',19) ,

(12, 'Riti', 41, 'Delhi' , 17) ,

(13, 'Aadi', 46, 'Mumbai', 11) ,

(14, 'Mohit', 45, 'Delhi' , 15) ,

(15, 'Veena', 43, 'Delhi' , 14) ,

(16, 'Shaunak', 42, 'Mumbai', 17 ),

(17, 'Manik', 42, 'Sydney', 14 ),

(18, 'Vikas', 42, 'Delhi', 11 ),

(19, 'Samir', 42, 'Mumbai', 15 ),

(20, 'Shobhit', 40, 'Sydney', 12) ]

# Create a DataFrame object

df = pd.DataFrame(empoyees,columns=['ID', 'Name', 'Age', 'City', 'Experience'])

df = df.set\_index('ID')

# Display the DataFrame

print("-"\*40)

print(df)

print("-"\*40)

x

#GroupByEx2.py

import pandas as pd

# List of Tuples

empoyees = [

(11, 'Jack', 44, 'Sydney',19) ,

(12, 'Riti', 41, 'Delhi' , 17) ,

(13, 'Aadi', 46, 'Mumbai', 11) ,

(14, 'Mohit', 45, 'Delhi' , 15) ,

(15, 'Veena', 43, 'Delhi' , 14) ,

(16, 'Shaunak', 42, 'Mumbai', 17 ),

(17, 'Manik', 42, 'Sydney', 14 ),

(18, 'Vikas', 42, 'Delhi', 11 ),

(19, 'Samir', 42, 'Mumbai', 15 ),

(20, 'Shobhit', 40, 'Sydney', 12) ]

# Create a DataFrame object

df = pd.DataFrame(empoyees,

columns=['ID', 'Name', 'Age', 'City', 'Experience'])

df = df.set\_index('ID')

# Display the DataFrame

print(df)

print("---------------------------------------------------------------------")

groupObj = df.groupby('City')

for grpName, rows in groupObj:

print("Group Name: ", grpName)

print('Group Content: ')

print(rows)

print("---------------------------------------------------------------------")

#GroupByEx3.py

import pandas as pd

# List of Tuples

empoyees = [

(11, 'Jack’, 44, 'Sydney',19) ,

(12, 'Riti', 41, 'Delhi' , 17) ,

(13, 'Aadi', 46, 'Mumbai', 11) ,

(14, 'Mohit', 45, 'Delhi' , 15) ,

(15, 'Veena', 43, 'Delhi' , 14) ,

(16, 'Shaunak', 42, 'Mumbai', 17 ),

(17, 'Manik', 42, 'Sydney', 14 ),

(18, 'Vikas', 42, 'Delhi', 11 ),

(19, 'Samir', 42, 'Mumbai', 15 ),

(20, 'Shobhit', 40, 'Sydney', 12) ]

# Create a DataFrame object

df = pd.DataFrame(empoyees,

columns=['ID', 'Name', 'Age', 'City', 'Experience'])

df = df.set\_index('ID')

# Display the DataFrame

print(df)

print("-"\*40)

print("---------------------------------------------------------------------")

groupObj = df.groupby('City')

for grpName, rows in groupObj:

print("Group Name: ", grpName)

print('Group Content: ')

print(rows)

print("\n------------------------------------------------------------------------------")

firstRowDf = df.groupby('City').first()

print(firstRowDf)

print("\n------------------------------------------------------------------------------")

firstRowDf = df.groupby('City').nth(1)

print(firstRowDf)

print("\n------------------------------------------------------------------------------")

print(df.groupby('City').size())

print("\n------------------------------------------------------------------------------")

specificGroup = df.groupby('City').get\_group('Mumbai')

print(specificGroup)

print("----------------------------------------------------------------------------")

#GroupByEx11.py

import pandas as pd

# List of Tuples

empoyees = [

(11, 'Jack', 44, 'Sydney',19) ,

(12, 'Riti', 41, 'Delhi' , 17) ,

(13, 'Aadi', 46, 'Mumbai', 11) ,

(14, 'Mohit', 45, 'Delhi' , 15) ,

(15, 'Veena', 43, 'Delhi' , 14) ,

(16, 'Shaunak', 42, 'Mumbai', 17 ),

(17, 'Manik', 42, 'Sydney', 14 ),

(18, 'Vikas', 42, 'Delhi', 11 ),

(19, 'Samir', 42, 'Mumbai', 15 ),

(20, 'Shobhit', 40, 'Sydney', 12) ]

# Create a DataFrame object

df = pd.DataFrame(empoyees, columns=['ID', 'Name', 'Age', 'City', 'Experience'])

df = df.set\_index('ID')

# Display the DataFrame

print("-"\*40)

print(df)

print("-"\*40)

groupObj = df.groupby('City')

for grpname,row in groupObj:

print("Group Name:{}".format(grpname))

print(row)

#GroupByEx111.py

import pandas as pd

# List of Tuples

empoyees = [

(11, 'Jack', 44, 'Sydney',19) ,

(12, 'Riti’, 41, 'Delhi' , 17) ,

(13, 'Aadi’, 46, 'Mumbai', 11) ,

(14, 'Mohit', 45, 'Delhi' , 15) ,

(15, 'Veena', 43, 'Delhi' , 14) ,

(16, 'Shaunak', 42, 'Mumbai', 17 ),

(17, 'Manik', 42, 'Sydney', 14 ),

(18, 'Vikas', 42, 'Delhi', 11 ),

(19, 'Samir', 42, 'Mumbai', 15 ),

(20, 'Shobhit', 40, 'Sydney', 12) ]

# Create a DataFrame object

df = pd.DataFrame(empoyees, columns=['ID', 'Name', 'Age', 'City', 'Experience'])

df = df.set\_index('ID')

# Display the DataFrame

print ("-"\*40)

print(df)

print ("-"\*40)

groupObj = df.groupby('City')

for grpname,row in groupObj:

print ("Group Name:{}".format(grpname))

print(row)

print("="\*50)

# Get first row of each group

firstRowDf = df.groupby('City').first()

print(firstRowDf)

print("="\*50)

firstRowDf = df.groupby('City').nth(1)

print(firstRowDf)

print("="\*50)

# Get the size of DataFrame groups

print(df.groupby('City').size())

print("="\*50)