In [ ]:
```
# VIVEK-CHAUHAN-ADVANCED-DATA-ANALYTICS-DATAFRAME
```

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

In [32]:
```
#creating a data-frame from a 2d dictionary haing values as a list/ndarray.

dic1 = {
    'student' : ['ruchika','neha','mark','gurjyot','jamal'],
    'marks' : [79,89,15,22,15]
}

print(dic1) #without data-frame it will be print in a single manner.
```

```
{'student': ['ruchika', 'neha', 'mark', 'gurjyot', 'jamal'], 'marks': [79, 89, 15, 2
2, 15]}
```

In [34]:
```
# after creating the data-frame it will look nice and easily readable and print in
#by default automatically the rows will start from 0 indexing value.

dtf1 = pd.DataFrame(dic1) #declare the data-frame and store it in a variable
print(dtf1)
```

```
   student  marks
0  ruchika     79
1     neha     89
2     mark     15
3  gurjyot     22
4    jamal     15
```

In [19]:
```
# we can give our own indexing for our easy understanding.

dtf2 = pd.DataFrame(dic1,index = ['first','second','third','fourth','fifth'])
print(dtf2)
```

```
        student  marks
first   ruchika     79
second     neha     89
third      mark     15
fourth  gurjyot     22
fifth     jamal     15
```

In [30]:
```
dic2 = {
    "section" : ['a','b','c','d'],
    "contri" : [6700,5600,5000,5200]
}

df2 = pd.DataFrame(dic2)
print(df2)
```

```
  section  contri
0       a    6700
1       b    5600
2       c    5000
3       d    5200
```

In [42]:
```
# creating data-frame from a 2d dictionary having values as dictionary objects.

people = {'sales' : {
    'name' : 'rohit',
    'age' : 24,
    'gender' : 'male'
},
    'marketing' : {
        "name" : 'neha',
        'age' : 25,
        'gender' : 'female'
    }
         }
print(people) #it will look like a conjusted without data-frame.
```

```
{'sales': {'name': 'rohit', 'age': 24, 'gender': 'male'}, 'marketing': {'name': 'neh
a', 'age': 25, 'gender': 'female'}}
```

In [46]:
```
#with the help of data-frame it will look easy,clean and readale format.

pd.DataFrame(people) #without variable store the data-frame print but not recommend
```

Out[46]:

	sales	marketing
name	rohit	neha
age	24	25
gender	male	female

In [50]:
```
# in the inner dictionary the key will become index ex like qtr1,qtr2... and first

sales = {
    'yr1':{
        'qtr1' : 34500,
        'qtr2' : 56000,
        'qtr3' : 47000,
        'qtr4' : 49000
    },
    'yr2':{
        'qtr1' : 30500,
        'qtr2' : 51000,
        'qtr3' : 97000,
        'qtr4' : 68000
    }
}

pd.DataFrame(sales)
```

Out[50]:

	yr1	yr2
qtr1	34500	30500
qtr2	56000	51000
qtr3	47000	97000
qtr4	49000	68000

In [56]:
```
collect1 = {'yr1' : 1500,'yr2':2500}
collect2 = {'yr1':2200,'nil':0}

collect = {1:collect1,2:collect2}
pd.DataFrame(collect)

# for missing values it will print the NaN.
```

Out[56]:

	1	2
yr1	1500.0	2200.0
yr2	2500.0	NaN
nil	NaN	0.0

In [62]:
```
# when you want two different data-frames in a single data-frame then this is used.

yr1 = {
    'qtr1' : 44900,
    'qtr2' : 46100,
    'q3' : 57000,
    'q4' : 59000
}

yr2 = {
    'a' : 54500,
    'b' : 51000,
    'qtr4' : 57000
}

df3 = {
    1: yr1,
    2:yr2
}
pd.DataFrame(df3)
```

Out[62]:

	1	2
qtr1	44900.0	NaN
qtr2	46100.0	NaN
q3	57000.0	NaN
q4	59000.0	NaN
a	NaN	54500.0
b	NaN	51000.0
qtr4	NaN	57000.0

In [3]:
```
toppera = {
    'rollno' : 115,
    'name' : 'pavni',
    'marks' : 97.5
}
topperb = {
    'rollno' : 236,
    'name' : 'rishi',
    'marks' : 98
}
topperc = {
    'rollno' : 387,
    'name' : 'preet',
    'marks' : 98.5
}
topperd = {
    'rollno' : 422,
    'name' : 'paula',
    'marks' : 98
}

toppers = [toppera,topperb,topperc,topperd] # here is list that's why it will consi

topdf = pd.DataFrame(toppers,index = ["toppera","topperb","topperc","topperd"]) # h
print(topdf)
```

```
         rollno   name  marks
toppera     115  pavni   97.5
topperb     236  rishi   98.0
topperc     387  preet   98.5
topperd     422  paula   98.0
```

In [102…
```
# passing 2-d list

list2 = [[10,20,30],[40,50,60],[70,80,90]] # here it will list.
pd.DataFrame(list2) #after the data-frame it will consider as array and values are
```

Out[102…

	0	1	2
0	10	20	30
1	40	50	60
2	70	80	90

In [119…

```
target = [56000,70000,75000,60000]
sales = [58000,68000,78000,61000]

zonalsales = [target,sales]

zsaledf = pd.DataFrame(zonalsales,columns = ["zonea","zoneb","zonec","zoned"],index
print(zsaledf)
```

```
        zonea  zoneb  zonec  zoned
target  56000  70000  75000  60000
sales   58000  68000  78000  61000
```

In [9]:

```
arr2 = np.array([[101,113,124],[130,140,200],[115,216,217]])
arr3 = pd.DataFrame(arr2)
print(arr3)
```

```
     0    1    2
0  101  113  124
1  130  140  200
2  115  216  217
```

In [21]:

```
#creating the dataframe object with the help of series

staff = pd.Series([20,36,44])
salaries = pd.Series([166000,246000,563000])

school = {
    "people" : staff,
    "amount" : salaries #here if you write the values in the "" quote then it will
}

df4 = pd.DataFrame(school)
print(df4)
```

```
   people  amount
0      20  166000
1      36  246000
2      44  563000
```

In [25]:

```
avg = staff / salaries

org = {
    "people" : staff,
    "amount" : salaries,
    "average" : avg
}
```

```
dtf5 = pd.DataFrame(org)
print(dtf5)
```

```
   people  amount   average
0      20  166000  0.000120
1      36  246000  0.000146
2      44  563000  0.000078
```

In [55]:
```
# retriving the various properties of the dataframe object.

print(dtf5.index)
print(dtf5.columns)
print(dtf5.axes)
print(dtf5.dtypes)
print(dtf5.size)
print(dtf5.shape)
print(dtf5.ndim)
print(dtf5.empty)
```

```
RangeIndex(start=0, stop=3, step=1)
Index(['people', 'amount', 'average'], dtype='object')
[RangeIndex(start=0, stop=3, step=1), Index(['people', 'amount', 'average'], dtype
='object')]
people       int64
amount       int64
average    float64
dtype: object
9
(3, 3)
2
False
```

In [57]:
```
# getting the number of rows in the dataframe

print(len(dtf5))
```

```
3
```

In [61]:
```
# getting count of non na values in dataframe

print(dtf5.count()) # by default it will take is 0 and count the column which is no
```

```
people     3
amount     3
average    3
dtype: int64
```

In [65]:
```
print(dtf5.count(axis = "index")) #produces the same result.
```

```
people     3
amount     3
average    3
dtype: int64
```

In [69]:
```
print(dtf5.count(1)) # it will count as row wise.
```

```
0    3
1    3
2    3
dtype: int64
```

In [73]:
```
print(dtf5.count(axis = "columns")) #produces the same result.
```

```
0    3
1    3
2    3
dtype: int64
```

In [77]:
```
# transposing a dataframe

print(dtf5.T) # it will transpose the values rows become column and column become r
```

```
                  0             1             2
people        20.00000      36.000000      44.000000
amount    166000.00000  246000.000000  563000.000000
average       0.00012       0.000146       0.000078
```

In [17]:
```
peopledata = {
    "weight" : [42,75,66],
    "name" : ["arnav","charles","guru"],
    "age" : [15,22,35]
}

df = pd.DataFrame(peopledata,index = ["people1","people2","people3"])
print(df)
```

```
         weight     name  age
people1      42    arnav   15
people2      75  charles   22
people3      66     guru   35
```

In [83]:
```
# now its time to transpose it

print(df.T)
```

```
        people1  people2 people3
weight      42       75      66
name     arnav  charles    guru
age         15       22      35
```

In [89]:
```
# lets create a dataframe using numpy value attribute.

df.values
```

Out[89]:
```
array([[42, 'arnav', 15],
       [75, 'charles', 22],
       [66, 'guru', 35]], dtype=object)
```

In [93]:
```
# selecting or the accessing data
# selecting or accessing a column
# both notation is used to print colums
```

```
print(df.name)
print(df["name"])
```

```
people1      arnav
people2    charles
people3       guru
Name: name, dtype: object
people1      arnav
people2    charles
people3       guru
Name: name, dtype: object
```

In [101…
```
# selecting or accessing a multiple columns
# for multiple column you need to give a list

print(df[["name","age"]]) # notice the double square brackets.
```

```
            name  age
people1    arnav   15
people2  charles   22
people3     guru   35
```

In [105…
```
aids = {
    "toys" : [7916,8508,7226,7617],
    "books" : [6189,8208,6149,6157],
    "uniform" : [610,508,611,457],
    "shoes" : [8810,6798,9611,6457]
}

aidsdf = pd.DataFrame(aids,index = ["andhra","odisha","m.p.","u.p"])
print(aidsdf)
```

```
        toys  books  uniform  shoes
andhra  7916   6189      610   8810
odisha  8508   8208      508   6798
m.p.    7226   6149      611   9611
u.p     7617   6157      457   6457
```

In [107…
```
print(aidsdf[["books","uniform"]])
```

```
        books  uniform
andhra   6189      610
odisha   8208      508
m.p.     6149      611
u.p      6157      457
```

In [109…
```
print(aidsdf.shoes)
```

```
andhra    8810
odisha    6798
m.p.      9611
u.p       6457
Name: shoes, dtype: int64
```

In [13]:
```
# selecting/accessing a subset from a dataframe using row/column
# , is in the bracket then the rest of columns will be print
#, is out of the bracket then this particular column will be print
```

```
df.loc["people1",:]
```

Out[13]:
```
weight       42
name       arnav
age           15
Name: people1, dtype: object
```

In [133…
```
df.loc["people1" : "people3",:]
```

Out[133…

	weight	name	age
people1	42	arnav	15
people2	75	charles	22
people3	66	guru	35

In [143…
```
# to accessing the columns

df.loc[:,"weight":"age"] # if you forget to write the : and , then only column name
```

Out[143…

	weight	name	age
people1	42	arnav	15
people2	75	charles	22
people3	66	guru	35

In [145…
```
# to access the range of rows and columns then
# to print only selective range of rows and columns then

df.loc["people1" : "people2","name" : "age"]
```

Out[145…

	name	age
people1	arnav	15
people2	charles	22

In [149…
```
print(aidsdf.loc["andhra" : "odisha", "books" : "uniform"])
```

```
        books  uniform
andhra   6189      610
odisha   8208      508
```

In [163…
```
# we cant remember and given the labels to the dataframe at that scenario the integ
# iloc means integer locations comes into the picture same like slices one is less

print(df.iloc[0:2])
```

```
         weight     name  age
people1      42    arnav   15
people2      75  charles   22
```

In [167…
```
# 1 is starting index for columns same like excel.

print(df.iloc[0:2,1:2])
```

```
           name
people1    arnav
people2  charles
```

In [169…
```
# selecting or accssing the individual values from dataframe

print(df.name["people1"])
```

```
arnav
```

In [188…
```
# slecting the dataframe rows/columns based on the boolean value it will compare ea

df["age"] > 18
```

Out[188…
```
people1    False
people2     True
people3     True
Name: age, dtype: bool
```

In [194…
```
# but in the df[the condition is passed] then it will return the values who passed

print(df[df["age"] > 18])
```

```
         weight     name  age
people2      75  charles   22
people3      66     guru   35
```

In [200…
```
# adding or modifying a column

df.age = 18 # by default it will store the 18 value 18 to age columns.
print(df)
```

```
         weight     name  age
people1      42    arnav   18
people2      75  charles   18
people3      66     guru   18
```

In [202…
```
# we can write own modify value by using list.

df.age = [18,22,15]
print(df)
```

```
         weight     name  age
people1      42    arnav   18
people2      75  charles   22
people3      66     guru   15
```

In [210…
```
# adding or modifying rows
# loc always use location of the rows and modify it values.

df.loc["people1"] = [85,"vivek",22]
print(df)
```

```
         weight     name  age
people1      85    vivek   22
people2      75  charles   22
people3      66     guru   15
```

In [29]:
```
#if you want to add new column in the existing dataframe then ["new column name"] i
#dataframe-name.newcolumn name = [] just print the elements like list not add the n

df["rollno"] = [11,12,13]
print(df)
```

```
         weight     name  age  rollno
people1      42    arnav   15      11
people2      75  charles   22      12
people3      66     guru   35      13
```

In [49]:
```
# adding rows from a dataframe

rowdf = pd.DataFrame(
    {
        "col1" : [15,13,12],
        "col2" : [14,12,10]
    }
,index = ["row1","row2","row3"])

print(rowdf)
```

```
      col1  col2
row1    15    14
row2    13    12
row3    12    10
```

In [51]:
```
#lets append the existing dataframe

newappend = rowdf.append(rowdf)
print(newappend)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_40536\3820229638.py in ?()
      1 #lets append the existing dataframe
      2
----> 3 newappend = rowdf.append(rowdf)
      4 print(newappend)

C:\ProgramData\anaconda3\Lib\site-packages\pandas\core\generic.py in ?(self, name)
   6295             and name not in self._accessors
   6296             and self._info_axis._can_hold_identifiers_and_holds_name(name)
   6297         ):
   6298             return self[name]
-> 6299         return object.__getattribute__(self, name)

AttributeError: 'DataFrame' object has no attribute 'append'
```

In [64]:
```
# modifying the single cell
```

```
rowdf.col1["row1"] = 20 #element will be updated
print(rowdf)
```

```
      col1  col2
row1    20    14
row2    13    12
row3    12    10
```

```
C:\Users\fv8.DESKTOP-N5HA3AQ\AppData\Local\Temp\ipykernel_40536\1660350734.py:3: Fut
ureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain c
ases, but when using Copy-on-Write (which will become the default behaviour in panda
s 3.0) this will never work to update the original DataFrame or Series, because the
intermediate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a si
ngle step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy

  rowdf.col1["row1"] = 20
```

In [66]:
```
# deleting rows/columns del statement is used.

del rowdf['col1']
print(rowdf) # col1 is deleted
```

```
      col2
row1    14
row2    12
row3    10
```

In [ ]:
```
# deleting rows/columns drop function is used with the help of indexes but make sur

rowdf.drop(range(1,2,1))
print(rowdf)
```

In [83]:
```
# deleting row/columns with the help of row names.

rowdf.drop(["row3"],axis = 0)
print(rowdf)
```

```
      col2
row1    14
row2    12
row3    10
```

In [120…
```
#renaming the rows/columns

rowdf.rename(index = {"row1":"r1","row2":"r2","row3":"r3"})
print(rowdf)
```

```
    col2
r1    14
r2    12
r3    10
```

In [124…
```
#after renaming the rows but it is temperory
print(rowdf)
```

```
    col2
r1    14
r2    12
r3    10
```

In [126…
```
#renaming the rows/columns permanently. just use inplace = True.makesure the t is c

newname = rowdf.rename(index = {"row1":"r1","row2":"r2","row3":"r3"},inplace = True
print(rowdf)
```

```
    col2
r1    14
r2    12
r3    10
```

In [38]:
```
#boolean indexing in the dataframe

days = ["sunday","monday"]
classes = [6,7]

daysframe = {
    "classes-days":days,
    "no of classes" : classes
}

classdf = pd.DataFrame(daysframe,index = [True,False])
print(classdf)
```

```
      classes-days  no of classes
True        sunday              6
False       monday              7
```

In [66]:
```
#number indexing in the dataframe

day = ["sunday","monday"]
classs = [6,7]

daysframe = {
    "classes-days":day,
    "no of classes" : classs
}

classdf = pd.DataFrame(daysframe,index = [0,1])
print(classdf)
```

```
  classes-days  no of classes
0       sunday              6
1       monday              7
```

In [64]:
```
# rows accessing from dataframe with boolean indexes

print(classdf.loc[True])
```

```
classes-days     sunday
no of classes         6
Name: True, dtype: object
```

In [62]:
```
#with the help of row 0,1 indexing value accessing.
print(classdf.iloc[0])
```

```
classes-days     sunday
no of classes         6
Name: True, dtype: object
```

In [60]:
```
#with the help of condition

classdf.loc[classdf['no of classes']>=6]
```

Out[60]:

	classes-days	no of classes
True	sunday	6
False	monday	7

In [ ]:

In [ ]: