py-pandas-visualization-practice-1

November 12, 2023

[ ]:
```
# !pip install pandas
```

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

Section A: 20 Marks

0.0.1 1.Answer the following questions (Marks : 4)

a. Create a numpy array with 20 equidistant point between 3 to 17. (Marks 1)

b. Convert the numpy array into a 4 * 5 matrix.(Marks 1)

c. Multiply the below matrix with the matrix created above.(Marks 1)

$$\begin{bmatrix} 1 & 3 & 0 & 1 \\ 2 & 3 & 2 & 7 \\ 1 & 4 & 2 & 1 \\ 1 & 9 & 0 & 1 \\ 6 & 4 & 6 & 7 \end{bmatrix}$$

d. Find the inverse of the matrix created after multiplication.(Marks 1)

a. Create a numpy array with 20 equidistant point between 3 to 17. (Marks 1)

[3]:
```
np.arange(3,18,0.5)  # start,stop,step
```

[3]:
```
array([ 3. ,  3.5,  4. ,  4.5,  5. ,  5.5,  6. ,  6.5,  7. ,  7.5,  8. ,
        8.5,  9. ,  9.5, 10. , 10.5, 11. , 11.5, 12. , 12.5, 13. , 13.5,
       14. , 14.5, 15. , 15.5, 16. , 16.5, 17. , 17.5])
```

[15]:
```
arr = np.linspace(3,17,20)  # start, stope, how many numbers
arr
```

[15]:
```
array([ 3.        ,  3.73684211,  4.47368421,  5.21052632,  5.94736842,
        6.68421053,  7.42105263,  8.15789474,  8.89473684,  9.63157895,
       10.36842105, 11.10526316, 11.84210526, 12.57894737, 13.31578947,
       14.05263158, 14.78947368, 15.52631579, 16.26315789, 17.        ])
```

b. Convert the numpy array into a 4 * 5 matrix.(Marks 1)

[16]:
```
arr.shape
```

[16]:
```
(20,)
```

[17]:
```
arr.ndim
```

[17]:
```
1
```

[18]:
```
arr = arr.reshape(4,5)
arr
```

[18]:
```
array([[ 3.        ,  3.73684211,  4.47368421,  5.21052632,  5.94736842],
       [ 6.68421053,  7.42105263,  8.15789474,  8.89473684,  9.63157895],
       [10.36842105, 11.10526316, 11.84210526, 12.57894737, 13.31578947],
       [14.05263158, 14.78947368, 15.52631579, 16.26315789, 17.        ]])
```

[19]:
```
arr.shape
```

[19]:
```
(4, 5)
```

[20]:
```
arr.ndim
```

[20]:
```
2
```

c. Multiply the below matrix with the matrix created above.(Marks 1)

$$\begin{bmatrix} 1 & 3 & 0 & 1 \\ 2 & 3 & 2 & 7 \\ 1 & 4 & 2 & 1 \\ 1 & 9 & 0 & 1 \\ 6 & 4 & 6 & 7 \end{bmatrix}$$

[21]:
```
arr
```

[21]:
```
array([[ 3.        ,  3.73684211,  4.47368421,  5.21052632,  5.94736842],
       [ 6.68421053,  7.42105263,  8.15789474,  8.89473684,  9.63157895],
       [10.36842105, 11.10526316, 11.84210526, 12.57894737, 13.31578947],
       [14.05263158, 14.78947368, 15.52631579, 16.26315789, 17.        ]])
```

[23]:
```
arr1 = np.array([[1,3,0,1],[2,3,2,7],[1,4,2,1],[1,9,0,1],[6,4,6,7]])
arr1
```

[23]:
```
array([[1, 3, 0, 1],
       [2, 3, 2, 7],
       [1, 4, 2, 1],
       [1, 9, 0, 1],
```

```
        [6, 4, 6, 7]])
```

[46]:
```
arr2 = np.matmul(arr,arr1)
```

[29]:
```
np.matmul(arr1,arr)
```

[29]:
```
array([[ 37.10526316,  40.78947368,  44.47368421,  48.15789474,
         51.84210526],
       [145.15789474, 155.47368421, 165.78947368, 176.10526316,
        186.42105263],
       [ 64.52631579,  70.42105263,  76.31578947,  82.21052632,
         88.10526316],
       [ 77.21052632,  85.31578947,  93.42105263, 101.52631579,
        109.63157895],
       [205.31578947, 222.26315789, 239.21052632, 256.15789474,
        273.10526316]])
```

[38]:
```
a = np.array([[1,2],[3,4]])
b = np.array([[10,20],[30,40]])
a
```

[38]:
```
array([[1, 2],
       [3, 4]])
```

[39]:
```
b
```

[39]:
```
array([[10, 20],
       [30, 40]])
```

[43]:
```
b%a
```

[43]:
```
array([[0, 0],
       [0, 0]])
```

[44]:
```
np.matmul(a,b)
```

[44]:
```
array([[ 70, 100],
       [150, 220]])
```

[45]:
```
np.dot(a,b)
```

[45]:
```
array([[ 70, 100],
       [150, 220]])
```

d. Find the inverse of the matrix created after multiplication.(Marks 1)

[47]:
```
np.linalg.inv(arr2)
```

[47]:
```
array([[-5.89362871e+13,  8.87000977e+13, -5.91333985e+11,
        -2.91724766e+13],
       [ 1.64587959e+13, -1.18266797e+13, -2.57230283e+13,
         2.10909121e+13],
       [ 4.69124961e+13, -7.03687442e+13, -0.00000000e+00,
         2.34562481e+13],
       [-1.17281240e+13,  0.00000000e+00,  3.51843721e+13,
        -2.34562481e+13]])
```

0.0.2 2. Write a program for (Marks : 16)

Score 1 :

Name | |ITP| |NPV| |SLC|

|—— | ———-|—-| |—-| |—-| |Harish| |77.0| |N/A| |57.4| |Mukesh| |84.0| |65.0| |N/A| |Ram| |N/A| |61.0| |81.0| |Senthil| |55.0| |N/A| |N/A| |Tom| |N/A| |N/A| |72.0|

Score 2 :

Name	SQL
Harish	74.0

a.Prepare two dataframes `Score1` & `Score2` (Marks 2)

b.Join `Score2` with `Score1` along the name to get the score of all the students in one dataframe. (Marks 3)

c.Prepare a table that shows count for the students who has not appeared in the corresponding subjects.(Marks 2)

d.Add a column that will show rank of the students based on the total score obtained.Do not consider students who have appeared in less than two subjects.Exclude those students from the table. (Marks 3)

e.Visualize Score in each subject corresponding to each student name .Make sure data corresponding to each student is represented by a different colour.(Marks 4)

f.Print the student name who has ranked second.(Marks 2)

a.Prepare two dataframes `Score1` & `Score2` (Marks 2)

[68]:
```
data_list = [['Harish',77,np.nan,57.4], ['Mukesh',84,65,np.nan],
             ['Ram',np.nan,61,81], ['Senthil',55,np.nan,np.nan],
             ['Tom',np.nan,np.nan,72]]
score1 = pd.DataFrame(data_list, columns = ['Name','ITP','NPV','SLC'])
print(score1)
```

```
      Name   ITP   NPV   SLC
0   Harish  77.0   NaN  57.4
1   Mukesh  84.0  65.0   NaN
```

```
2      Ram   NaN  61.0  81.0
3  Senthil  55.0   NaN   NaN
4      Tom   NaN   NaN  72.0
```

[64]:
```
score1.index
```

[64]:
```
RangeIndex(start=0, stop=5, step=1)
```

[65]:
```
score1.columns
```

[65]:
```
Index(['Name', 'ITP', 'NPV', 'SLC'], dtype='object')
```

[66]:
```
score1.values
```

[66]:
```
array([['Harish', 77.0, nan, 57.4],
       ['Mukesh', 84.0, 65.0, nan],
       ['Ram', nan, 61.0, 81.0],
       ['Senthil', 55.0, nan, nan],
       ['Tom', nan, nan, 72.0]], dtype=object)
```

[63]:
```
score2 = pd.DataFrame({'Name' : ['Harish'], 'SQL':[74.0]})
print(score2)
```

```
     Name   SQL
0  Harish  74.0
```

b.Join Score2 with Score1 along the name to get the score of all the students in one dataframe. (Marks 3)

[78]:
```
score1.columns = ['StudentName', 'ITP', 'NPV', 'SLC']
print(score1)
```

```
  StudentName   ITP   NPV   SLC
0      Harish  77.0   NaN  57.4
1      Mukesh  84.0  65.0   NaN
2         Ram   NaN  61.0  81.0
3     Senthil  55.0   NaN   NaN
4         Tom   NaN   NaN  72.0
```

[81]:
```
print(score2)
```

```
     Name   SQL
0  Harish  74.0
```

[85]:
```
# pd.merge(score1,score2, how = 'left', on='Name')
score_df = pd.merge(score1,score2, how = 'left', left_on= 'StudentName',␣
 ↪right_on='Name')
score_df.drop('Name',axis = 1, inplace = True)
```

```
score_df
```

[85]:

	StudentName	ITP	NPV	SLC	SQL
0	Harish	77.0	NaN	57.4	74.0
1	Mukesh	84.0	65.0	NaN	NaN
2	Ram	NaN	61.0	81.0	NaN
3	Senthil	55.0	NaN	NaN	NaN
4	Tom	NaN	NaN	72.0	NaN

1 Drop

[110]:
```
data_list = [['Harish',77,np.nan,57.4], ['Mukesh',84,65,np.nan],
             ['Ram',np.nan,61,81], ['Senthil',55,np.nan,np.nan],
             ['Tom',np.nan,np.nan,72]]
df = pd.DataFrame(data_list, columns = ['Name','ITP','NPV','SLC'])
df
```

[110]:

	Name	ITP	NPV	SLC
0	Harish	77.0	NaN	57.4
1	Mukesh	84.0	65.0	NaN
2	Ram	NaN	61.0	81.0
3	Senthil	55.0	NaN	NaN
4	Tom	NaN	NaN	72.0

[113]:
```
df.drop([0,2,4],axis = 0)
```

[113]:

	Name	ITP	NPV	SLC
1	Mukesh	84.0	65.0	NaN
3	Senthil	55.0	NaN	NaN

[112]:
```
df1 = df.set_index('Name')
df1
```

[112]:

Name	ITP	NPV	SLC
Harish	77.0	NaN	57.4
Mukesh	84.0	65.0	NaN
Ram	NaN	61.0	81.0
Senthil	55.0	NaN	NaN
Tom	NaN	NaN	72.0

[114]:
```
df1.drop(['Harish','Ram'],axis = 0)
```

[114]:

Name	ITP	NPV	SLC
Mukesh	84.0	65.0	NaN

```
Senthil  55.0   NaN   NaN
Tom       NaN   NaN  72.0
```

[109]:
```
df.drop([0,2,4],axis = 0)
```

[109]:
```
      Name   ITP   NPV  SLC
1   Mukesh  84.0  65.0  NaN
3  Senthil  55.0   NaN  NaN
```

[107]:
```
df.drop('Name',axis = 1)
```

[107]:
```
    ITP   NPV   SLC
0  77.0   NaN  57.4
1  84.0  65.0   NaN
2   NaN  61.0  81.0
3  55.0   NaN   NaN
4   NaN   NaN  72.0
```

[103]:
```
df.drop([3,2], axis = 0, inplace = True)
```

[104]:
```
df
```

[104]:
```
     Name   ITP   NPV   SLC
0  Harish  77.0   NaN  57.4
1  Mukesh  84.0  65.0   NaN
4     Tom   NaN   NaN  72.0
```

[93]:
```
df.drop(['SLC','NPV'], axis = 1)
```

[93]:
```
      Name   ITP
0   Harish  77.0
1   Mukesh  84.0
2      Ram   NaN
3  Senthil  55.0
4      Tom   NaN
```

[94]:
```
df
```

[94]:
```
      Name   ITP   NPV   SLC
0   Harish  77.0   NaN  57.4
1   Mukesh  84.0  65.0   NaN
2      Ram   NaN  61.0  81.0
3  Senthil  55.0   NaN   NaN
4      Tom   NaN   NaN  72.0
```

c.Prepare a table that shows count for the students who has not appeared in the corresponding subjects.(Marks 2)

[115]:
```
score_df
```

[115]:

	StudentName	ITP	NPV	SLC	SQL
0	Harish	77.0	NaN	57.4	74.0
1	Mukesh	84.0	65.0	NaN	NaN
2	Ram	NaN	61.0	81.0	NaN
3	Senthil	55.0	NaN	NaN	NaN
4	Tom	NaN	NaN	72.0	NaN

[138]:
```
count_df = pd.DataFrame(score_df[['ITP','NPV','SLC','SQL']].count()).
 ↪reset_index()
count_df.columns = ['Subject','Count of students not appeared']
print(count_df)
```

	Subject	Count of students not appeared
0	ITP	3
1	NPV	2
2	SLC	3
3	SQL	1

[132]:
```
df.set_index('ITP').reset_index()
```

[132]:

	ITP	Name	NPV	SLC
0	77.0	Harish	NaN	57.4
1	84.0	Mukesh	65.0	NaN
2	NaN	Ram	61.0	81.0
3	55.0	Senthil	NaN	NaN
4	NaN	Tom	NaN	72.0

[134]:
```
pd.DataFrame(df[['ITP','NPV','SLC']].isnull().sum())
```

[134]:

	0
ITP	2
NPV	3
SLC	2

d.Add a column that will show rank of the students based on the total score obtained.Do not consider students who have appeared in less than two subjects.Exclude those students from the table. (Marks 3)

[151]:
```
score_df['subjects appeared'] = score_df.iloc[:,1:].count(axis = 1)
score_df
```

[151]:

	StudentName	ITP	NPV	SLC	SQL	subjects appeared
0	Harish	77.0	NaN	57.4	74.0	3
1	Mukesh	84.0	65.0	NaN	NaN	2
2	Ram	NaN	61.0	81.0	NaN	2

```
3     Senthil  55.0   NaN   NaN   NaN                  1
4         Tom   NaN   NaN  72.0   NaN                  1
```

[155]:
```
score_df = score_df[score_df['subjects appeared']>=2]
score_df
```

[155]:
```
  StudentName   ITP   NPV   SLC   SQL  subjects appeared
0      Harish  77.0   NaN  57.4  74.0                  3
1      Mukesh  84.0  65.0   NaN   NaN                  2
2         Ram   NaN  61.0  81.0   NaN                  2
```

[159]:
```
score_df['total marks'] = score_df.loc[:,['ITP','NPV','SLC','SQL']].sum(axis =␣
↪1)
score_df
```

```
C:\Users\Naaaz\AppData\Local\Temp\ipykernel_1468\1956666571.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  score_df['total marks'] = score_df.loc[:,['ITP','NPV','SLC','SQL']].sum(axis =
1)
```

[159]:
```
  StudentName   ITP   NPV   SLC   SQL  subjects appeared  total marks
0      Harish  77.0   NaN  57.4  74.0                  3        208.4
1      Mukesh  84.0  65.0   NaN   NaN                  2        149.0
2         Ram   NaN  61.0  81.0   NaN                  2        142.0
```

[165]:
```
score_df['rank'] = score_df['total marks'].rank(ascending = False)
score_df
```

```
C:\Users\Naaaz\AppData\Local\Temp\ipykernel_1468\1426188528.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  score_df['rank'] = score_df['total marks'].rank(ascending = False)
```

[165]:
```
  StudentName   ITP   NPV   SLC   SQL  subjects appeared  total marks  rank
0      Harish  77.0   NaN  57.4  74.0                  3        208.4   1.0
1      Mukesh  84.0  65.0   NaN   NaN                  2        149.0   2.0
2         Ram   NaN  61.0  81.0   NaN                  2        142.0   3.0
```

e.Visualize Score in each subject corresponding to each student name .Make sure data corresponding to each student is represented by a different colour. ‘.(Marks 4)

[ ]:

f.Print the student name who has ranked second.(Marks 2)

[170]:
```
score_df[score_df['rank']==2]['StudentName'].values[0]
```

[170]: 'Mukesh'

Section B: 20 Marks

1.0.1 3. Write a program for (Marks : 20)

Score1: Name | |Subject| |Score|

|—— | ———-|—-| |—-| |Sutithi.Chakraborty_1| |ITP| |34| |Sutithi.Chakraborty_2| |ITP| |55| |Deepali.Gatade_1| |ITP| |77| |Mukul.Kumar.Singh_1| |NPV| |65|

Score2: Name | |Subject| |Score|

|—— | ———- |—-| |—-| |Deepali.Gatade_01| |SQL| |74| |Deepali.Gatade_2| |SLC| |57| |Mukul.Kumar.Singh_1| |SLC| |-| |Rakesh.Sriramula_1| |NPV| |61|

a. Create the dataframes (Marks 3)

b. Apply a mapping and modify the `Name` column excluding the’_’ and subsequent number for both the dataframes(Marks 2)

c. Combine the dataframe data to have all the records in a single dataframe (Marks 1)

d. Remove the record where the score is not available (present as `-`)(Marks 2)

e. If a student is having multiple scores for the same subject,consider only the maximum number. (Marks 3)

f. Show a table where we have differnt subject as column header and the marks of students under each subsequent column and corresponding student name. (Marks 2)

Sample output : Name | |ITP| |NPV| |SLC| |SQL|

|—— | ———-|—-| |—-| |—-| |—-| |Deepali.Gatade| |77.0| |NaN| |57.4| |74.0| |Mukul.Kumar.Singh| |NaN| |65.0| |NaN| |NaN| |Rakesh.Sriramula| |NaN| |61.0| |NaN| |NaN|

g. `Mukul.Kumar.Singh` has got 78 in `ITP` , 62 in `SLC` , `Rakesh.Sriramula` has got 59 in `SQL` .Update his score in the above table.(Marks 2)

h. Add a column that will show the total marks obtained by each student in the above prepared table.(Marks 1)

i. Split the `Name` column into `First_Name` , `Middle_Name` & `Last_Name`.For ,(Marks 4)

Mukul.Kumar.Singh Mukul is the first name, Kumar is the middle name & Singh is the last name.

Sutithi.Chakraborty Sutithi is the first name & Chakraborty is the last name

For the other it is applicable in the same way.

a.Create the dataframes (Marks 3)

[189]:
```
score1_df = pd.DataFrame([['Sutithi.Chakraborty_1','ITP',34],
            ['Sutithi.Chakraborty_2' , 'ITP', 55],
            ['Deepali.Gatade_1', 'ITP' ,77],
            ['Mukul.Kumar.Singh_1', 'NPV',65]], columns =␣
 ↪['Name','Subject','Score'])

data_list2=[['Deepali.Gatade_1','SQL',74],['Deepali.Gatade_2','SLC',57]
           ,['Mukul.kumar.Singh_1','SLC','-'],['Rakesh.Sriramula_1','NPV',61]]
score2_df=pd.DataFrame(data_list2,columns=['Name','Subject','Score'])
```

[190]:
```
print(score1_df)
print(score2_df)
```

```
                   Name Subject  Score
0  Sutithi.Chakraborty_1     ITP     34
1  Sutithi.Chakraborty_2     ITP     55
2       Deepali.Gatade_1     ITP     77
3    Mukul.Kumar.Singh_1     NPV     65
                  Name Subject Score
0     Deepali.Gatade_1     SQL    74
1     Deepali.Gatade_2     SLC    57
2  Mukul.kumar.Singh_1     SLC     -
3   Rakesh.Sriramula_1     NPV    61
```

b. Apply a mapping and modify the Name column excluding the'_' and subsequent number for both the dataframes (Marks 2)

[192]:
```
def replace_text(text):
    return text[:text.find("_")]
score1_df['Name'] = score1_df['Name'].apply(replace_text)
score1_df
```

[192]:
```
                   Name Subject  Score
0  Sutithi.Chakraborty     ITP     34
1  Sutithi.Chakraborty     ITP     55
2       Deepali.Gatade     ITP     77
3    Mukul.Kumar.Singh     NPV     65
```

[193]:
```
score2_df['Name'] = score2_df['Name'].apply(lambda text: text[:text.find("_")])
score2_df
```

[193]:

	Name	Subject	Score
0	Deepali.Gatade	SQL	74
1	Deepali.Gatade	SLC	57
2	Mukul.kumar.Singh	SLC	-
3	Rakesh.Sriramula	NPV	61

c. Combine the dataframe data to have all the records in a single dataframe(Marks 1)

[202]:
```
comb_df = pd.concat([score1_df, score2_df],axis = 0).reset_index(drop = True)
comb_df
```

[202]:

	Name	Subject	Score
0	Sutithi.Chakraborty	ITP	34
1	Sutithi.Chakraborty	ITP	55
2	Deepali.Gatade	ITP	77
3	Mukul.Kumar.Singh	NPV	65
4	Deepali.Gatade	SQL	74
5	Deepali.Gatade	SLC	57
6	Mukul.kumar.Singh	SLC	-
7	Rakesh.Sriramula	NPV	61

[205]:
```
comb_df.index = [f'R{i}' for i in comb_df.index]
comb_df
```

[205]:

	Name	Subject	Score
R0	Sutithi.Chakraborty	ITP	34
R1	Sutithi.Chakraborty	ITP	55
R2	Deepali.Gatade	ITP	77
R3	Mukul.Kumar.Singh	NPV	65
R4	Deepali.Gatade	SQL	74
R5	Deepali.Gatade	SLC	57
R6	Mukul.kumar.Singh	SLC	-
R7	Rakesh.Sriramula	NPV	61

[207]:
```
comb_df.loc['R1':'R4','Name':'Subject']
```

[207]:

	Name	Subject
R1	Sutithi.Chakraborty	ITP
R2	Deepali.Gatade	ITP
R3	Mukul.Kumar.Singh	NPV
R4	Deepali.Gatade	SQL

[209]:
```
comb_df = comb_df.reset_index(drop = True)
comb_df
```

[209]:

	Name	Subject	Score
0	Sutithi.Chakraborty	ITP	34

1	Sutithi.Chakraborty	ITP	55
2	Deepali.Gatade	ITP	77
3	Mukul.Kumar.Singh	NPV	65
4	Deepali.Gatade	SQL	74
5	Deepali.Gatade	SLC	57
6	Mukul.kumar.Singh	SLC	-
7	Rakesh.Sriramula	NPV	61

[214]:
```
for i in comb_df.values.flatten():
    print(i,end = ",")
```

```
Sutithi.Chakraborty,ITP,34,Sutithi.Chakraborty,ITP,55,Deepali.Gatade,ITP,77,Muku
l.Kumar.Singh,NPV,65,Deepali.Gatade,SQL,74,Deepali.Gatade,SLC,57,Mukul.kumar.Sin
gh,SLC,-,Rakesh.Sriramula,NPV,61,
```

[211]:
```
comb_df.to_csv('Combination.csv', index = False)
```

d. Remove the record where the score is not available (present as -)(Marks 2)

[219]:
```
comb_df = comb_df[comb_df['Score'] != "-"]
comb_df
```

[219]:

	Name	Subject	Score
0	Sutithi.Chakraborty	ITP	34
1	Sutithi.Chakraborty	ITP	55
2	Deepali.Gatade	ITP	77
3	Mukul.Kumar.Singh	NPV	65
4	Deepali.Gatade	SQL	74
5	Deepali.Gatade	SLC	57
7	Rakesh.Sriramula	NPV	61

e. If a student is having multiple scores for the same subject,consider only the maximum number.(Marks 3)

[226]:
```
comb_df.groupby(['Name', 'Subject'])['Score'].agg([min,max])
```

```
C:\Users\Naaaz\AppData\Local\Temp\ipykernel_1468\3706994344.py:1: FutureWarning:
The provided callable <built-in function min> is currently using
SeriesGroupBy.min. In a future version of pandas, the provided callable will be
used directly. To keep current behavior pass the string "min" instead.
  comb_df.groupby(['Name', 'Subject'])['Score'].agg([min,max])
C:\Users\Naaaz\AppData\Local\Temp\ipykernel_1468\3706994344.py:1: FutureWarning:
The provided callable <built-in function max> is currently using
SeriesGroupBy.max. In a future version of pandas, the provided callable will be
used directly. To keep current behavior pass the string "max" instead.
  comb_df.groupby(['Name', 'Subject'])['Score'].agg([min,max])
```

[226]:

		min	max
Name	Subject		
Deepali.Gatade	ITP	77	77
	SLC	57	57
	SQL	74	74
Mukul.Kumar.Singh	NPV	65	65
Rakesh.Sriramula	NPV	61	61
Sutithi.Chakraborty	ITP	34	55

f. Show a table where we have differnt subject as column header and the marks of students under each subsequent column and corresponding student name(Marks 2)

Sample output : Name | |ITP| |NPV| |SLC| |SQL|

|—— | ————-|—-| |—-| |—-| |—-| |Deepali.Gatade| |77.0| |NaN| |57.4| |74.0| |Mukul.Kumar.Singh| |NaN| |65.0| |NaN| |NaN| |Rakesh.Sriramula| |NaN| |61.0| |NaN| |NaN|

[ ]:

g. `Mukul.Kumar.Singh` has got 78 in `ITP` , 62 in `SLC` , `Rakesh.Sriramula` has got 59 in `SQL` .Update his score in the above table.(Marks 2)

[ ]:

h. Add a column that will show the total marks obtained by each student in the above prepared table.(Marks 1)

[ ]:

i.Split the `Name` column into `First_Name` , `Middle_Name` & `Last_Name`.For ,(Marks 4) Mukul.Kumar.Singh Mukul is the first name, Kumar is the middle name & Singh is the last name. Sutithi.Chakraborty Sutithi is the first name & Chakraborty is the last name For the other it is applicable in the same way.

[ ]:

Section C: 30 Marks

1.0.2 4. Consider the following `NPV.csv` data and provide solution for the following question (Marks : 30)

a. What are the number of rows and no. of cols & types of variables (2 Marks)

b. Using for loop display the categorical data and numerical data seperetly(3 marks)

c. Convert the following columns value type into categorical 'cp', 'fbs', 'restecg', 'keratin_type', 'Hemoglobin_level', 'Immunity_level' (2 marks)

d. Drop the `Patient I.D.` column (1 marks)

e. Write a function named `Visualize` which will have four parameters. (Marks : 17) `First parameter` would be dataset name. `Second parameter` would be column name. `Third parameter` should have the name of a column.(which should have categorical data with two categories)[students can fixed this as Survive column].`Fourth parameter` should be `Yes` or `No`, where `Yes` denotes `Survived` and `No` denotes `Passed away`

Based on the Forth parameter the diagram should be drawn. If it says `Yes` then the data corresponding to `Survived` data in Survive Column should be considered.

If it says `No` then the data corresponding to `Passed away` data in Survive Column should be considered.

The function would draw a `histogram` if the parameter contains `continuos` data.

For `histogram` there should be 20 bins

The function would draw a `bar chart` if the parameter contains categorical data with `more than 3 categories`.

For `bar chart` : Arrange the bar as per the descending order of the height of the bars.

The categories that have top 25% of the frequency should be coloured in `Green` & rest should be coloured in `Red`.

For example if a categorical column is having 8 categories the categories with highest and second highest frequencies should be coloured in `Green` and rest should be coloured in `Red`. Corresponding index for `Green` should be mentioned as `High` and the `Red` as `Low`.

Again if a categorical column is having 17 categories the categories with top 5 bar length should be coloured in `Green` and rest should be coloured in `Red`. Corresponding index for `Green` should be mentioned as `High` and the `Red` as `Low`.

The function would draw a `pie chart` if the parameter contains categorical data with `less than 4 categories`.

For `pie chart` : The colour index box for respective categories should be mentioned. The percentage of each slice should be mentioned.

In each plot a title should appear mentioning the `chart name` (histogram , distribution , bar chart or pie chart) `represents the column name` for (`Survived` or `Passed away`) data

Note : Do not consider Survive column for second parameter

f. Plot an appropriate diagram to visualise the `Age` distribution over different `blood_group` type.Comment on your observation.(Marks 3)

g. Plot an appropriate diagram to visualise the relationship between `Age` & `thalachh`.Comment on your observation.(Marks 2) import pandas as pd data = pd.read_csv('NPV.csv') data.head()

a. What are the number of rows and no. of cols & types of variables (2 Marks)

[ ]:

b. Using for loop display the categorical data and numerical data seperetly(3 marks)

[ ]:

c. Convert the following columns value type into categorical 'cp', 'fbs', 'restecg', 'keratin_type', 'Hemoglobin_level', 'Immunity_level' (2 marks)

[ ]:

d. Drop the `Patient I.D.` column (1 marks)

[ ]:

e. Write a function named `Visualize` which will have four parameters. (Marks : 17) `First parameter` would be dataset name. `Second parameter` would be column name. `Third parameter` should have the name of a column.(which should have categorical data with two categories)[students can fixed this as Survive column].`Fourth parameter` should be `Yes` or `No`, where `Yes` denotes `Survived` and `No` denotes `Passed away`

Based on the Forth parameter the diagram should be drawn. If it says `Yes` then the data corresponding to `Survived` data in Survive Column should be considered.

If it says `No` then the data corresponding to `Passed away` data in Survive Column should be considered.

The function would draw a `histogram` if the parameter contains `continuos` data.

For `histogram` there should be 20 bins

The function would draw a `bar chart` if the parameter contains categorical data with `more than 3 categories`.

For `bar chart` : Arrange the bar as per the descending order of the height of the bars.

The categories that have top 25% of the frequency should be coloured in `Green` & rest should be coloured in `Red`.

For example if a categorical column is having 8 categories the categories with highest and second highest frequencies should be coloured in `Green` and rest should be coloured in `Red`. Corresponding index for `Green` should be mentioned as `High` and the `Red` as `Low`.

Again if a categorical column is having 17 categories the categories with top 5 bar length should be coloured in `Green` and rest should be coloured in `Red`. Corresponding index for `Green` should be mentioned as `High` and the `Red` as `Low`.

The function would draw a `pie chart` if the parameter contains categorical data with `less than 4 categories`.

For `pie chart` : The colour index box for respective categories should be mentioned. The percentage of each slice should be mentioned.

In each plot a title should appear mentioning the `chart name` (histogram , distribution , bar chart or pie chart) `represents the column name` for (`Survived` or `Passed away`) data

Note : Do not consider Survive column for second parameter

Sample Test Case :

[ ]:
```
Visualize(data , 'blood_group' , 'Survive' , 'Yes' )
```

[ ]:
```
Visualize(data , 'Sex' , 'Survive' , 'No' )
```

[ ]:
```

```

f. Plot an appropriate diagram to visualise the `Age` distribution over different `blood_group` type.Comment on your observation.(Marks 3)

[ ]:
```

```

g. Plot an appropriate diagram to visualise the relationship between `Age` & `thalachh`.Comment on your observation.(Marks 2)

[ ]:
```

```