Consider the following Python dictionary data and Python list labels:

data = {'birds': ['Cranes', 'Cranes', 'plovers', 'spoonbills', 'spoonbills', 'Cranes', 'plovers', 'Cranes', 'spoonbills', 'spoonbills'], 'age': [3.5, 4, 1.5, np.nan, 6, 3, 5.5, np.nan, 8, 4], 'visits': [2, 4, 3, 4, 3, 4, 2, 2, 3, 2], 'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

In [1]:

```
import numpy as np
import pandas as pd

#Imported the necessary libraries for performing data manipulation operations for numer
ical and multidimensional dataset.
```

1. Create a DataFrame birds from this dictionary data which has the index labels.

In [2]:

```
data = {'birds': ['Cranes', 'Cranes', 'plovers', 'spoonbills', 'spoonbills', 'Cranes',
'plovers', 'Cranes', 'spoonbills', 'spoonbills'], 'age': [3.5, 4, 1.5, np.nan, 6, 3, 5.
5, np.nan, 8, 4], 'visits': [2, 4, 3, 4, 3, 4, 2, 2, 3, 2], 'priority': ['yes', 'yes',
'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

## called the pd.DataFrame function which accepts the dict dataset and the row index is
assigned with custom defined index name using the index parameter
## And then displayed both with the help of print and without using print
df_bd=pd.DataFrame(data,index=labels)
```

In [3]:

```
print(df_bd)
```

```
        birds  age  visits priority
a      Cranes  3.5       2      yes
b      Cranes  4.0       4      yes
c     plovers  1.5       3       no
d  spoonbills  NaN       4      yes
e  spoonbills  6.0       3       no
f      Cranes  3.0       4       no
g     plovers  5.5       2       no
h      Cranes  NaN       2      yes
i  spoonbills  8.0       3       no
j  spoonbills  4.0       2       no
```

In [4]:

```
df_bd
```

Out[4]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
b	Cranes	4.0	4	yes
c	plovers	1.5	3	no
d	spoonbills	NaN	4	yes
e	spoonbills	6.0	3	no
f	Cranes	3.0	4	no
g	plovers	5.5	2	no
h	Cranes	NaN	2	yes
i	spoonbills	8.0	3	no
j	spoonbills	4.0	2	no

In [5]:

```
df_bd.shape
```

Out[5]:

```
(10, 4)
```

2. Display a summary of the basic information about birds DataFrame and its data.

In [6]:

```
df_bd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   birds     10 non-null     object 
 1   age       8 non-null      float64
 2   visits    10 non-null     int64  
 3   priority  10 non-null     object 
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
```

In [7]:

```
print("Please find the below Summary details for Bird Dataframe :")
print("***********************************")
print(df_bd.info())
print("***********************************")
```

```
Please find the below Summary details for Bird Dataframe :
***********************************
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   birds     10 non-null     object 
 1   age       8 non-null      float64
 2   visits    10 non-null     int64  
 3   priority  10 non-null     object 
dtypes: float64(1), int64(1), object(2)
memory usage: 400.0+ bytes
None
***********************************
```

1. The above Output tells us df_bd belongs to the Dataframe Class.
2. It has a total of 10 Index entries and the index name which ranges from a to j
3. A total of 4 columns and detailed each columns name, datatype and Not Null count.
4. Also it summaries how many columns belongs to each data type ---> dtypes: float64(1), int64(1), object(2)

3. Print the first 2 rows of the birds dataframe

In [8]:

```
## We can use Head/tail function to display first and last set of rows.
## df_bd.head() by default displays first 5 rows, if we pass the number parameter insid
e the head() function, it will display
## that much rows. Below I have passed the n parameter as 2.
df_bd.head(2)
```

Out[8]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
b	Cranes	4.0	4	yes

4. Print all the rows with only 'birds' and 'age' columns from the dataframe

In [9]:

```
## There are multiple ways to display those information.

df_bd[['birds','age']] ## Since we are going to display all the rows, so directly mentioning the column name is also one of the possible ways
```

Out[9]:

	birds	age
a	Cranes	3.5
b	Cranes	4.0
c	plovers	1.5
d	spoonbills	NaN
e	spoonbills	6.0
f	Cranes	3.0
g	plovers	5.5
h	Cranes	NaN
i	spoonbills	8.0
j	spoonbills	4.0

In [10]:

```
df_bd.iloc[:,0:2] ## By using slicing techniques
```

Out[10]:

	birds	age
a	Cranes	3.5
b	Cranes	4.0
c	plovers	1.5
d	spoonbills	NaN
e	spoonbills	6.0
f	Cranes	3.0
g	plovers	5.5
h	Cranes	NaN
i	spoonbills	8.0
j	spoonbills	4.0

In [11]:

```
df_bd.iloc[:,[0,1]] ## Using column selection based on column index name along with sli
cing
```

Out[11]:

	birds	age
a	Cranes	3.5
b	Cranes	4.0
c	plovers	1.5
d	spoonbills	NaN
e	spoonbills	6.0
f	Cranes	3.0
g	plovers	5.5
h	Cranes	NaN
i	spoonbills	8.0
j	spoonbills	4.0

5. select [2, 3, 7] rows and in columns ['birds', 'age', 'visits']

In [12]:

```
## Let me display the dataframe again.
df_bd
```

Out[12]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
b	Cranes	4.0	4	yes
c	plovers	1.5	3	no
d	spoonbills	NaN	4	yes
e	spoonbills	6.0	3	no
f	Cranes	3.0	4	no
g	plovers	5.5	2	no
h	Cranes	NaN	2	yes
i	spoonbills	8.0	3	no
j	spoonbills	4.0	2	no

In [13]:

```
#lets try using iloc again.
#since the default indexing starts from 0 the 2nd row is denited as 1, 3rd row as 2 and
7th row as 6 and the same for column index as well
df_bd.iloc[[1,2,6],[0,1,2]]
```

Out[13]:

	birds	age	visits
b	Cranes	4.0	4
c	plovers	1.5	3
g	plovers	5.5	2

In [14]:

```
#Lets try using loc instead of using default index location.
df_bd.loc[['b','c','g'],['birds','age','visits']]
```

Out[14]:

	birds	age	visits
b	Cranes	4.0	4
c	plovers	1.5	3
g	plovers	5.5	2

6. select the rows where the number of visits is less than 4

*** Since visits column is of integer data type and there is no NaN values, so we dont have to perform any type casting or conversion. Index: 10 entries, a to j Data columns (total 4 columns): # Column Non-Null Count Dtype --- ------ -------------- ----- 0 birds 10 non-null object 1 age 8 non-null float64 2 visits 10 non-null int64 3 priority 10 non-null object dtypes: float64(1), int64(1), object(2) memory usage: 720.0+ bytes

In [15]:

```
df_bd[df_bd['visits'] < 4]
```

Out[15]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
c	plovers	1.5	3	no
e	spoonbills	6.0	3	no
g	plovers	5.5	2	no
h	Cranes	NaN	2	yes
i	spoonbills	8.0	3	no
j	spoonbills	4.0	2	no

7. select the rows with columns ['birds', 'visits'] where the age is missing i.e NaN

In [16]:

```
## will select all the columns whose rows has missing values under age column and then
 will select the appropriate columns required.

df_bd[df_bd['age'].isnull()][['birds','visits']]
```

Out[16]:

	birds	visits
d	spoonbills	4
h	Cranes	2

In [17]:

```
## The other way,now the boolean value returned is applied to both birds and visits col
umn and only rows which has 'True' to it is returned.
df_bd[['birds','visits']][df_bd['age'].isnull()]
```

Out[17]:

	birds	visits
d	spoonbills	4
h	Cranes	2

8. Select the rows where the birds is a Cranes and the age is less than 4

In [18]:

```
df_bd
```

Out[18]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
b	Cranes	4.0	4	yes
c	plovers	1.5	3	no
d	spoonbills	NaN	4	yes
e	spoonbills	6.0	3	no
f	Cranes	3.0	4	no
g	plovers	5.5	2	no
h	Cranes	NaN	2	yes
i	spoonbills	8.0	3	no
j	spoonbills	4.0	2	no

In [19]:

```
df_bd[(df_bd['age'] < 4.0) & (df_bd['birds'].str.lower() == 'cranes')]
```

Out[19]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
f	Cranes	3.0	4	no

In [20]:

```
df_bd[(df_bd['age'] < 4) & (df_bd['birds'].str.lower() == 'cranes')]
```

Out[20]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
f	Cranes	3.0	4	no

## Handled case sensitive strings using str.lower() method. Also tested how dtype float workes if we pass or test against int

9. Select the rows the age is between 2 and 4(inclusive)

In [21]:

```
df_bd[(df_bd['age'] >= 2) & (df_bd['age'] <= 4)]
```

Out[21]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
b	Cranes	4.0	4	yes
f	Cranes	3.0	4	no
j	spoonbills	4.0	2	no

10. Find the total number of visits of the bird Cranes

In [22]:

```
# Lets see total visits for each birds
df_bd.groupby(['birds']).sum()[['visits']]
```

Out[22]:

	visits
birds	
Cranes	12
plovers	5
spoonbills	12

In [23]:

```
## Tried using Groupby method.
df_bd[df_bd['birds'].str.lower() == 'cranes'].groupby(['birds']).sum()['visits']
```

Out[23]:

```
birds
Cranes    12
Name: visits, dtype: int64
```

In [24]:

```
## In a more structured way. where the result is shows as a dataframe
df_bd[df_bd['birds'].str.lower() == 'cranes'].groupby(['birds']).sum()[['visits']]
```

Out[24]:

	visits
birds	
Cranes	12

In [25]:

```
## Tried the same without groupby
df_bd[df_bd['birds'].str.lower() == 'cranes']['visits'].sum()
```

Out[25]:

```
12
```

11. Calculate the mean age for each different birds in dataframe.

In [26]:

```
## Grouped based on birds and then calculated the mean for each integer column and select age among them.
df_bd.groupby(['birds']).mean()[['age']]
```

Out[26]:

```
             age
     birds
    Cranes   3.5
   plovers   3.5
spoonbills   6.0
```

In [27]:

```
## select the desired column age, brids and then grouped them based on birds and applied mean function
df_bd[['age','birds']].groupby(['birds']).mean()
```

Out[27]:

```
             age
     birds
    Cranes   3.5
   plovers   3.5
spoonbills   6.0
```

12. Append a new row 'k' to dataframe with your choice of values for each column. Then delete that row to return the original DataFrame.

In [28]:

```
df_bd
```

Out[28]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
b	Cranes	4.0	4	yes
c	plovers	1.5	3	no
d	spoonbills	NaN	4	yes
e	spoonbills	6.0	3	no
f	Cranes	3.0	4	no
g	plovers	5.5	2	no
h	Cranes	NaN	2	yes
i	spoonbills	8.0	3	no
j	spoonbills	4.0	2	no

In [29]:

```
df_bd=df_bd.append({'birds':'HummingBird','age':1,'visits':'2','priority':'yes'},ignore
_index=True)
```

In [30]:

```
df_bd
## since append resets the custom index, we need to reassign the row index name again.
 And the same has been assigned using DataFrame index attribute.
```

Out[30]:

	birds	age	visits	priority
0	Cranes	3.5	2	yes
1	Cranes	4.0	4	yes
2	plovers	1.5	3	no
3	spoonbills	NaN	4	yes
4	spoonbills	6.0	3	no
5	Cranes	3.0	4	no
6	plovers	5.5	2	no
7	Cranes	NaN	2	yes
8	spoonbills	8.0	3	no
9	spoonbills	4.0	2	no
10	HummingBird	1.0	2	yes

In [31]:

```
df_bd.index = ['a','b','c','d','e','f','g','h','i','j','k']
```

In [32]:

```
df_bd
```

Out[32]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
b	Cranes	4.0	4	yes
c	plovers	1.5	3	no
d	spoonbills	NaN	4	yes
e	spoonbills	6.0	3	no
f	Cranes	3.0	4	no
g	plovers	5.5	2	no
h	Cranes	NaN	2	yes
i	spoonbills	8.0	3	no
j	spoonbills	4.0	2	no
k	HummingBird	1.0	2	yes

In [33]:

```
df_bd.drop('k',inplace=True) ## If inplace=True is not coded, then it is not saved to the dataframe
```

In [34]:

```
df_bd
```

Out[34]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
b	Cranes	4.0	4	yes
c	plovers	1.5	3	no
d	spoonbills	NaN	4	yes
e	spoonbills	6.0	3	no
f	Cranes	3.0	4	no
g	plovers	5.5	2	no
h	Cranes	NaN	2	yes
i	spoonbills	8.0	3	no
j	spoonbills	4.0	2	no

In [35]:

```
### Lets try using Pandas Concat method to perform row append, where we can use axis bi
undary case to indicate how to append the data.
df_bd
```

Out[35]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
b	Cranes	4.0	4	yes
c	plovers	1.5	3	no
d	spoonbills	NaN	4	yes
e	spoonbills	6.0	3	no
f	Cranes	3.0	4	no
g	plovers	5.5	2	no
h	Cranes	NaN	2	yes
i	spoonbills	8.0	3	no
j	spoonbills	4.0	2	no

In [ ]:

```

```

13. Find the number of each type of birds in dataframe (Counts)

In [36]:

```
df_bd.groupby(['birds']).count()
```

Out[36]:

	age	visits	priority
birds			
Cranes	3	4	4
plovers	2	2	2
spoonbills	3	4	4

In [37]:

```
df_bd.groupby(['birds']).size()
```

Out[37]:

```
birds
Cranes        4
plovers       2
spoonbills    4
dtype: int64
```

*** When tried with count() method it shows the count for other column values and if there are any NaN values it gets skipped. *** i came across a method called size() which tells the total number of data points including NaN, *** so i have used them after performing a groupby based on birds category and applied size function.

14. Sort dataframe (birds) first by the values in the 'age' in decending order, then by the value in the 'visits' column in ascending order.

In [39]:

```
df_bd.sort_values(by=['age','visits'],ascending=[False,True])
```

Out[39]:

	birds	age	visits	priority
i	spoonbills	8.0	3	no
e	spoonbills	6.0	3	no
g	plovers	5.5	2	no
j	spoonbills	4.0	2	no
b	Cranes	4.0	4	yes
a	Cranes	3.5	2	yes
f	Cranes	3.0	4	no
c	plovers	1.5	3	no
h	Cranes	NaN	2	yes
d	spoonbills	NaN	4	yes

In [40]:

```
## One thing I noted above is, In SQL NULL values will be treated as HIGH Values by def
ault, whereas it treats NaN values as LOW values.
```

In [41]:

```
df_bd.sort_values(by=['age','visits'],ascending=[False,True],na_position='first')
```

Out[41]:

	birds	age	visits	priority
h	Cranes	NaN	2	yes
d	spoonbills	NaN	4	yes
i	spoonbills	8.0	3	no
e	spoonbills	6.0	3	no
g	plovers	5.5	2	no
j	spoonbills	4.0	2	no
b	Cranes	4.0	4	yes
a	Cranes	3.5	2	yes
f	Cranes	3.0	4	no
c	plovers	1.5	3	no

15. Replace the priority column values with'yes' should be 1 and 'no' should be 0

In [42]:

```
df_bd.replace({'priority' : {'yes':1,'no':0}}) ## I have referred https://pandas.pydat
a.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html

## I have referred Python Documentation for DataFrames and Series for Replace. I need t
o revist this attribute functionality again.
```

Out[42]:

	birds	age	visits	priority
a	Cranes	3.5	2	1
b	Cranes	4.0	4	1
c	plovers	1.5	3	0
d	spoonbills	NaN	4	1
e	spoonbills	6.0	3	0
f	Cranes	3.0	4	0
g	plovers	5.5	2	0
h	Cranes	NaN	2	1
i	spoonbills	8.0	3	0
j	spoonbills	4.0	2	0

In [43]:

```
df_bd
```

Out[43]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
b	Cranes	4.0	4	yes
c	plovers	1.5	3	no
d	spoonbills	NaN	4	yes
e	spoonbills	6.0	3	no
f	Cranes	3.0	4	no
g	plovers	5.5	2	no
h	Cranes	NaN	2	yes
i	spoonbills	8.0	3	no
j	spoonbills	4.0	2	no

In [44]:

```
df_bd.replace({'priority' : {'yes':1,'no':0}},inplace=True)
```

In [45]:

```
df_bd
```

Out[45]:

	birds	age	visits	priority
a	Cranes	3.5	2	1
b	Cranes	4.0	4	1
c	plovers	1.5	3	0
d	spoonbills	NaN	4	1
e	spoonbills	6.0	3	0
f	Cranes	3.0	4	0
g	plovers	5.5	2	0
h	Cranes	NaN	2	1
i	spoonbills	8.0	3	0
j	spoonbills	4.0	2	0

In [ ]:

```

```

16. In the 'birds' column, change the 'Cranes' entries to 'trumpeters'.

In [46]:

```
df_bd.replace({'birds':{'Cranes'.lower() : 'trumpeters'}})
## Somehow the lower methos doesnt work for a string, might be im wrong.
```

Out[46]:

	birds	age	visits	priority
a	Cranes	3.5	2	1
b	Cranes	4.0	4	1
c	plovers	1.5	3	0
d	spoonbills	NaN	4	1
e	spoonbills	6.0	3	0
f	Cranes	3.0	4	0
g	plovers	5.5	2	0
h	Cranes	NaN	2	1
i	spoonbills	8.0	3	0
j	spoonbills	4.0	2	0

In [47]:

```
df_bd['birds'].replace('Cranes','trumpeters')
```

Out[47]:

```
a    trumpeters
b    trumpeters
c       plovers
d    spoonbills
e    spoonbills
f    trumpeters
g       plovers
h    trumpeters
i    spoonbills
j    spoonbills
Name: birds, dtype: object
```

In [51]:

```
## The above method doesnt handle case sensitive letters, so i tried the below one which converts each string in lowercase
## and then applied the replace method. I tried the same with df_bd[['birds']].str.lower().replace('cranes','trumpeters'),
## but getting error. Saying DataFrame object didnt have a attribute str, so handled them as a series and it worked.

df_bd['birds'].str.lower().replace('cranes','trumpeters')
```

Out[51]:

```
a    trumpeters
b    trumpeters
c       plovers
d    spoonbills
e    spoonbills
f    trumpeters
g       plovers
h    trumpeters
i    spoonbills
j    spoonbills
Name: birds, dtype: object
```

In [53]:

```
df_bd1 = df_bd['birds'].str.lower().replace('cranes','trumpeters')
```

In [54]:

```
df_bd1
```

Out[54]:

```
a    trumpeters
b    trumpeters
c       plovers
d    spoonbills
e    spoonbills
f    trumpeters
g       plovers
h    trumpeters
i    spoonbills
j    spoonbills
Name: birds, dtype: object
```

In [ ]:

```
## inplace doesnt work for the above code (df_bd['birds'].str.lower().replace('cranes','trumpeters'))
```