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']

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

In [0]:
```
import numpy as np # import numpy for numerical computation
import pandas as pd # import pandas for Data manipulation
```

In [0]:
```
# here creating a dictionary of data
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']
        }

# creating list of labels for indexing purpose
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

In [0]:
```
data = pd.DataFrame(data,index=labels) # Creating dataframe using method DataFrame()
```

In [8]:
```
data # See the data
```

Out[8]:

	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 [0]:
```
```

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

In [9]:
```
data.info()
# .info() shows the basic information about the Data:
# Information included:
# type
# no_of_columns
# column_name ,  Data_type , null_state_of_column,
# memory usage
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, a to j
Data columns (total 4 columns):
birds       10 non-null object
age         8 non-null float64
visits      10 non-null int64
priority    10 non-null object
dtypes: float64(1), int64(1), object(2)
memory usage: 720.0+ bytes
```

In [0]:
```
```

3. Print the first 2 rows of the birds dataframe

In [10]:
```
data.iloc[0:2] # use iloc for index_location means to access rows using their
 state(index_value)
```

Out[10]:

	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 [11]:
```
data[['birds','age']]
# to access more than one column pass list of columns
```

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]:
```
data[['birds','age','visits']].iloc[[2,3,7]]
```

Out[12]:

	birds	age	visits
c	plovers	1.5	3
d	spoonbills	NaN	4
h	Cranes	NaN	2

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

In [0]:
```
list_of_response = data['visits'] < 4 # list_of_response stores the Series of
 Boolean value
```

In [15]:
```
type(list_of_response) # let's check the type of list_of_response variable
```

Out[15]:
```
pandas.core.series.Series
```

In [16]:
```
list(list_of_response) # I converted to list just for better appearence
```

Out[16]:
```
[True, False, True, False, True, False, True, True, True, True]
```

In [17]:
```
# let's access rows using above Series
data[list_of_response]
```

Out[17]:

	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

In [25]:
```
# I can do all above task by just writting one line of code
data[ data['visits'] < 4 ]
```

Out[25]:

	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

In [26]:
```
data[ data['visits'] < 4 ][['birds','age']]
```

Out[26]:

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

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

In [33]:
```
data[data['age'].isnull()][['birds','visits']]
```

Out[33]:

	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 [34]:
```
data[ (data['birds']=='Cranes') & (data['age']<4)] # use ()'paranthesis' when
 comparing two conditions
# use '&' instead of 'and', right.
```

Out[34]:

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

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

In [36]:
```
data[ (data['age']<=4) & (data['age']>=2)]
```

Out[36]:

	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 [38]:
```
data[data['birds'] == 'Cranes']['visits'].sum()
# steps to find the result
# step1: accessing all rows where 'birds' == 'Crane'
# step2: among all selected rows, select 'visits' column, right.
# step3: sum up total value
```

Out[38]: 12

To understand above process in better way, let's break it into many steps.

In [39]:
```
# step1: accessing all rows where 'birds' == 'Crane'
# and store it in temp_data(variable)
# and show temp_data
temp_data = data[ data['birds'] == 'Cranes']
temp_data
```

Out[39]:

	birds	age	visits	priority
a	Cranes	3.5	2	yes
b	Cranes	4.0	4	yes
f	Cranes	3.0	4	no
h	Cranes	NaN	2	yes

In [41]:
```
# step2: select the column visits
# and show it
# use temp_data instead of data because our processed data is sotred in temp_d
ata
# store 'visits' column in temp_data_visits,right.
temp_data_visits = temp_data['visits']
# you can also use SQL style:    "temp_data.visits" instead of "temp_data['vis
its']"
# But SQL style is highly not recommended
temp_data_visits
```

Out[41]:
```
a    2
b    4
f    4
h    2
Name: visits, dtype: int64
```

In [43]:
```
# step3: (final step) let's sum up all values
# using .sum() method
temp_data_visits.sum()
```

Out[43]:
```
12
```

we did it in different steps, just to understand what is happening behind the scene

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

In [45]:
```
data.groupby('birds')['age'].mean()
```

Out[45]:
```
birds
Cranes        3.5
plovers       3.5
spoonbills    6.0
Name: age, dtype: float64
```

Let's break above one-line code into many steps to understand better.

In [48]:
```
# step 1: grouping The 'data' by using 'birds column'
# and save it into grouped_data_bird
# and show grouped_data_bird (grouped data)
grouped_data_bird = data.groupby('birds')
grouped_data_bird # it is an object
```

Out[48]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fedc1bb8a58>
```

In [53]:
```
# step 2: select 'age' column
grouped_data_age=grouped_data_bird['age']
grouped_data_age # it is a series object
```

Out[53]:
```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x7fedc1abc4a8>
```

In [54]:
```
# step 3: find mean of every species
grouped_data_age.mean()
```

Out[54]:
```
birds
Cranes        3.5
plovers       3.5
spoonbills    6.0
Name: age, dtype: float64
```

In [0]:

In [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 [59]:
```
data
```

Out[59]:

	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 [0]:
```
data = data.append(pd.DataFrame(data={'birds':'parrot',
                                      'age':5.4,
                                      'visits':6,
                                      'priority':'yes'},
                                      index=['k']),
                                      )
```

In [69]:
```
data
```

Out[69]:

	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	parrot	5.4	6	yes

In [70]:
```
# let's remove added row
data.drop(labels='k',axis=0)
```

Out[70]:

	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 [71]:
```
data
```

Out[71]:

	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	parrot	5.4	6	yes

In [0]:
```
# remove the row inplace
data.drop(labels='k',inplace=True)
```

In [73]:
```
data
# k is deleted now
```

Out[73]:

	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 [0]:
```
```

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

In [77]:
```
data.groupby('birds')['birds'].value_counts()
```

Out[77]:
```
birds       birds
Cranes      Cranes        4
plovers     plovers       2
spoonbills  spoonbills    4
Name: birds, dtype: int64
```

In [0]:
```
```

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 [78]:
```
data.sort_values('age',ascending=False)
```

Out[78]:

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

In [82]:
```
data.sort_values('age',ascending=False).sort_values('visits',ascending=True) # here we are making both condition true
```

Out[82]:

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

In [83]:
```
data.sort_values('visits',ascending=True) # sorting the dataset by 'visits' co
lumn in ascending order
```

Out[83]:

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

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

In [0]:
```
def changingLabel(temp):
  length = len(temp)
  for i in range(length):
    if temp[i] == 'yes':
      temp[i] = 1
    else:
      temp[i] = 0
  return temp
```

In [94]:
```
data['priority'] = changingLabel(data['priority'])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

In [95]:
```
data
```

Out[95]:

	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 [0]:
```
```

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

In [0]:
```
def valueConverter(value):
  length = len(value)
  for i in range(length):
    if value[i] == 'Cranes':
      value[i] = 'trumpeters'
  return value
```

In [100]:
```
data['birds'] = valueConverter(data['birds'])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
```

In [101]:
```
data
```

Out[101]:

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

In [0]:
```

```