Week2 myNotes

April 13, 2020

WEEK -2 My notes

The Series Data Structure

Doubts:

-What is Vectorization?

The series is one of the core data structures in pandas. You think of it a cross between a list and a dictionary. The items are all stored in an order and there's labels with which you can retrieve them. An easy way to visualize this is two columns of data. The first is the special index, a lot like the dictionary keys. While the second is your actual data. It's important to note that the data column has a label of its own and can be retrieved using the .name attribute.

The name attribute usage: https://stackoverflow.com/questions/31292140/what-is-the-nameparameter-in-pandas-series

```
[5]: import pandas as pd
     import numpy as np
     pd.Series
```

[5]: pandas.core.series.Series

```
[3]: animals=["Tiger", "Lion", "Zebra", "Monkey"]
     animals_ser=pd.Series(animals,name="Animals")
     print(animals ser)
     animals_df=pd.DataFrame(animals_ser)
     animals df
```

```
0
       Tiger
1
        Lion
```

2 Zebra

Monkey

Name: Animals, dtype: object

- [3]: Animals
 - Tiger
 - 1 Lion
 - 2 Zebra

3 Monkey

```
[4]: numbers=[1,2,3,4]
     pd.Series(numbers,name="Numbers")
[4]: 0
     1
          2
     2
           3
     3
           4
     Name: Numbers, dtype: int64
    When there is lack of datapoint, it is considered as None objectType in case of list of Strings
[5]: animals=["Tiger","Lion","Zebra",None]
     pd.Series(animals)
[5]: 0
           Tiger
           Lion
     2
           Zebra
     3
            None
     dtype: object
    When there is lack of datapoint, it is considered as Nan floatType in case of list of integers
[6]: numbers=[1,2,3,None]
     pd.Series(numbers)
[6]: 0
           1.0
     1
           2.0
     2
           3.0
     3
          NaN
     dtype: float64
    Nan and None are not the same. They are not equal in terms of Numerical value.
[7]: import numpy as np
     np.nan == None
[7]: False
    np.isnan(np.nan)
[8]: True
[9]: alphabets={'A':'a','B':'b',
                'C':'c','D':'d'}
     s= pd.Series(alphabets)
```

```
[9]: A
      В
           b
      С
            С
      D
            d
      dtype: object
[10]: s.index
[10]: Index(['A', 'B', 'C', 'D'], dtype='object')
     1.1.1 Querying a Series
     A panda. Series can be queried, either by the index position or the index label.
[11]: alphabets={'A':'a','B':'b',
                 'C':'c','D':'d'}
      s= pd.Series(alphabets)
      s
```

[11]: A a
B b
C c
D d
dtype: object

Query using index number

```
[11]: s.iloc[3]
```

[11]: 'd'

Query using index name

```
[12]: s.loc['D']
```

[12]: 'd'

Note: When the index is list of numbers, pandas cant determine whether u r intending to query by index position or index lable.

1.1.2 What else we can do with Series?

Okay, so now we know how to get data out of the series. Let's talk about working with the data. A common task is to want to consider all of the values inside of a series and want to do some sort of operation. This could be trying to find a certain number, summarizing data or transforming the data in some way.

```
[13]: s=pd.Series([100.00,120.00,101.00,3.00])
s
```

```
[13]: 0 100.0
1 120.0
2 101.0
3 3.0
dtype: float64
```

A typical programmatic approach to this would be to iterate over all the items in the series, and invoke the operation one is interested in. For instance, we could create a data frame of floating point values. Let's think of these as prices for different products. We could write a little routine which iterates over all of the items in the series and adds them together to get a total.

```
[14]: #summing without vectorization
total=0
for item in s:
    total+=item
print(total)
```

324.0

Let's think of these as prices for different products. We could write a little routine which iterates over all of the items in the series and adds them together to get a total.

This works, but it's slow. Modern computers can do many tasks simultaneously, especially, but not only, tasks involving mathematics. Pandas and the underlying NumPy libraries support a method of computation called vectorization.

```
[16]: import numpy as np
    #summing using vectorization
    total=np.sum(s)
    print(total)
```

324.0

Now both of these methods create the same value, but is one actually faster?

```
[17]: #Let's create a big series of random numbers
s=pd.Series(np.random.randint(0,1000,10000))
s
```

```
[17]: 0 990

1 422

2 209

3 166

4 238

...

9995 284
```

```
9996 681
9997 225
9998 129
9999 330
Length: 10000, dtype: int64
```

The Jupyter Notebook has a magic function which can help to compute the time taken by each bunch of code to execute.

Magic functions begin with a percentage sign. If we type this sign and then hit the Tab key, we can see a list of the available magic functions. You could write your own magic functions too, but that's a little bit outside of the scope of this course. We're actually going to use what's called a **cellular magic function.**

The function is called **timeit**

2.21 ms ± 111 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

```
[19]: \( \frac{\psi \text{timeit} -n 100}{\psi \text{With Vectorization}} \)
\text{summary=np.sum(s)}
```

```
146 \mu s \pm 14.9 \mu s per loop (mean \pm std. dev. of 7 runs, 100 loops each)
```

This is a pretty shocking difference in the speed and demonstrates why data scientists need to be aware of parallel computing features and start thinking in functional programming terms.

Related feature in Pandas and NumPy is called broadcasting. With broadcasting, you can apply an operation to every value in the series, changing the series.

For instance, if we wanted to increase every random variable by 2, we could do so quickly using the += operator directly on the series object. Here I'll just use the head operator to just print out the top five rows in the series.

The procedural way of doing this would be to iterate through all of the items in the series and increase the values directly. A quick aside here. Pandas does support iterating through a series much like a dictionary, allowing you to unpack values easily. But if you find yourself iterating through a series, you should question whether you're doing things in the best possible way.

```
[22]: %%timeit -n 1
s=pd.Series(np.random.randint(0,1000,10000))
for label,value in s.iteritems():
    s.loc[label]=value+2
    print(s.loc[label])
```

..-

. . .

...

- I I

. . .

. . . .

. _ .

J-

. . .

--

-0-

4.0

OI1

-30

_ . .

...

. - -

...

_-

0.0

-

JJZ

- - -

--

--

-111

0.45

. . .

. . .

-00

- - -

-

--

--

- --

. . .

J J Z

0 -0

-0

--

. . .

- -

.--

...

_ _ _

...

. . .

0.40

- 40

_ . _

_ _ _

__ .

. . . _

1 12

- - -

0 1 1

. .

Ι1

0 10

. . .

4.40

0 --

- --

__.

OI

- - -

4.0

. . .

--

___.

20.

4-

٥٠.

--

__.

. . .

__`

_ _

~~

1 -10

-..

. . .

4.0

- .

- - -

--

. . . .

10 1

...

0,0

.

1 691

. . .

-

__.

1.0

J1 1

- . .

- - -

103

. . .

'

-00

-

- - -

_ _ .

__

. _ _ _

- - -

. . . .

//1

_ . . .

_ . .

. . .

. _ _

. . . .

_ .

E00

_ .

2 10

...

-

. . .

. - -

- - -

. - -

25.

- - -

~~

_

. . .

- - -

. _ .

J4

_

-..

```
462
      642
      336
      122
      629
      44
      47
      22
      314
      799
      493
      648
      435
      62
      506
      567
      2.97 \text{ s} \pm 49.1 \text{ ms} per loop (mean \pm std. dev. of 7 runs, 1 loop each)
[10]: %%timeit -n 10
       s=pd.Series(np.random.randint(0,1000,10000))
```

The slowest run took 17.44 times longer than the fastest. This could mean that an intermediate result is being cached.

```
1.79 \text{ ms} \pm 2.94 \text{ ms} per loop (mean \pm std. dev. of 7 runs, 10 loops each)
```

One last note on using the indexing operators to access series data. The .loc attribute lets you not only modify data in place, but also add new data as well. If the value you pass in as the index doesn't exist, then a new entry is added. And keep in mind, indices can have mixed types. While it's important to be aware of the typing going on underneath, Pandas will automatically change the underlying NumPy types as appropriate.

```
[24]: s=[1,2,3]
    ss=pd.Series(s)
    ss.loc['Animal']='Bears'
    ss
```

```
[24]: 0 1
1 2
2 3
Animal Bears
dtype: object
```

Up until now I've shown only examples of a series where the index values were unique. I want to end this lecture by showing an example where index values are not unique, and this makes data frames different, conceptually, that a relational database might be.

```
[26]: original_sports= pd.Series({'Archery':'Bhutan','Golf':'Scotland','Sumo':'Japan'
                                   ,'Shooting':'Italy',})
      cricket_loving_countries=pd.
       →Series(['Australia','India','Pakistan','England'],index=['Cricket'
                                            ,'Cricket','Cricket','Cricket'])
      all_countries=original_sports.append(cricket_loving_countries)
[27]: original_sports
[27]: Archery
                    Bhutan
      Golf
                  Scotland
      Sumo
                      Japan
      Shooting
                      Italy
      dtype: object
[28]: cricket_loving_countries
[28]: Cricket
                 Australia
      Cricket
                     India
      Cricket
                  Pakistan
      Cricket
                   England
      dtype: object
[29]: all_countries
[29]: Archery
                      Bhutan
      Golf
                   Scotland
      Sumo
                       Japan
      Shooting
                       Italy
                  Australia
      Cricket
      Cricket
                       India
      Cricket
                   Pakistan
      Cricket
                    England
      dtype: object
```

1.2 The DataFrame Data Structure

The DataFrame data structure is the heart of the Panda's library. It's a primary object that you'll be working with in data analysis and cleaning tasks.

The DataFrame is conceptually a two-dimensional series object, where there's an index and multiple columns of content, with each column having a label. In fact, the distinction between a column and a row is really only a conceptual distinction. And you can think of the DataFrame itself as simply a two-axes labeled array.

[12]: Name Item Purchased Cost
Store 1 Chris Dog Food 22.5
Store 1 Kevyn Kitty Litter 2.5
Store 2 Vinod Bird Seed 5.0

Similar to the series, we can extract data using the iLock and Lock attributes. Because the DataFrame is two-dimensional, passing a single value to the lock indexing operator will return series if there's only one row to return

```
[31]: df.loc['Store 2']
```

[31]: Name Vinod
Item Purchased Bird Seed
Cost 5
Name: Store 2, dtype: object

It's important to remember that the indices and column names along either axes, horizontal or vertical, could be non-unique. For instance, in this example, we see two purchase records for Store 1 as different rows. If we use a single value with the DataFrame lock attribute, multiple rows of the DataFrame will return, **not as a new series**, **but as a new DataFrame**.

```
[34]: df.loc['Store 1']
```

[34]: Name Item Purchased Cost
Store 1 Chris Dog Food 22.5
Store 1 Kevyn Kitty Litter 2.5

One of the powers of the Panda's DataFrame is that you can quickly **select data based on multiple axes.**

```
[14]: df.loc['Store 1','Cost']
```

[14]: Store 1 22.5 Store 1 2.5 Name: Cost, dtype: float64 What if we just wanted to do column selection and just get a list of all of the costs?

```
[39]:
     df.T
[39]:
                        Store 1
                                                   Store 2
                                       Store 1
      Name
                          Chris
                                         Kevyn
                                                     Vinod
                       Dog Food
                                 Kitty Litter
                                                 Bird Seed
      Item Purchased
      Cost
                           22.5
                                           2.5
      df.T.loc['Cost']
[41]:
[41]: Store 1
                  22.5
      Store 1
                   2.5
      Store 2
                     5
      Name: Cost, dtype: object
```

Here's another method. As we saw, .loc does row selection, and it can take two parameters, the row index and the list of column names. .loc also supports slicing. If we wanted to select all rows, we can use a column to indicate a full slice from beginning to end. And then add the column name as the second parameter as a string. In fact, if we wanted to include multiply columns, we could do so in a list. And Pandas will bring back only the columns we have asked for.

```
[15]: df.loc[:,['Name','Cost']]
# df[['Name','Cost']]
```

[15]: Name Cost
Store 1 Chris 22.5
Store 1 Kevyn 2.5
Store 2 Vinod 5.0

It's easy to delete data in series and DataFrames, and we can use the **drop function** to do so. This function **takes a single parameter**, which is the index or roll label, to drop.

The drop function doesn't change the DataFrame by default. And instead, returns to you a copy of the DataFrame with the given rows removed. We can see that our original DataFrame is still intact.

```
[20]: print(df.drop('Store 1'))
df
```

Cost

```
Bird Seed
                                      5.0
     Store 2 Vinod
[20]:
                Name Item Purchased
                                      Cost
      Store 1
               Chris
                            Dog Food
                                      22.5
      Store 1
              Kevyn
                       Kitty Litter
                                       2.5
      Store 2
               Vinod
                          Bird Seed
                                       5.0
```

Name Item Purchased

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

```
Item Purchased Cost
Store 1 Dog Food 22.5
Store 1 Kitty Litter 2.5
Store 2 Bird Seed 5.0
```

[19]: Name Item Purchased Cost
Store 1 Chris Dog Food 22.5
Store 1 Kevyn Kitty Litter 2.5
Store 2 Vinod Bird Seed 5.0

Drop has two interesting optional parameters. The first is called **in place**, and if it's set to true, the DataFrame will be updated in place, instead of a copy being returned. The second parameter is the **axes**, which should be dropped. By default, this **value is 0**, **indicating the row axes**. But you could change it to 1 if you want to drop a column.

There is a second way to **drop a column**, however. And that's directly through the use of the indexing operator, using the **del keyword**.

```
[17]: copy_df=df.copy()
  del copy_df['Name']
  copy_df
```

[17]: Item Purchased Cost
Store 1 Dog Food 22.5
Store 1 Kitty Litter 2.5
Store 2 Bird Seed 5.0

Finally, adding a new column to the DataFrame is as easy as assigning it to some value.

```
[59]: df['Location']=None df
```

[59]: Name Item Purchased Cost Location Dog Food 22.5 Store 1 Chris None Store 1 Kevyn Kitty Litter 2.5 None Store 2 Bird Seed 5.0 Vinod None

Question:1 For the purchase records from the pet store, how would you update the DataFrame, applying a discount of 20% across all the values in the 'Cost' column?

```
[60]: df['Cost'] *= 0.8 df
```

[60]: Name Item Purchased Cost Location
Store 1 Chris Dog Food 18.0 None
Store 1 Kevyn Kitty Litter 2.0 None

Store 2 Vinod Bird Seed 4.0 None

1.3 DataFrame Indexing and Loading

The common work flow is to read your data into a DataFrame then reduce this DataFrame to the particular columns or rows that you're interested in working with. As you've seen, the Panda's toolkit tries to give you views on a DataFrame. This is much faster than copying data and much more memory efficient too.

But it does mean that if you're manipulating the data you have to be aware that any changes to the DataFrame you're working on may have an impact on the base data frame you used originally.

Here's an example using our same purchasing DataFrame from earlier. We can create a series based on just the cost category using the square brackets. Then we can increase the cost in this series using broadcasting. Now if we look at our original DataFrame, we see those costs have risen as well. This is an important consideration to watch out for. If you want to explicitly use a copy, then you should consider calling the copy method on the DataFrame for it first.

```
[21]: costs = df['Cost']
      costs
[21]: Store 1
                  22.5
      Store 1
                   2.5
      Store 2
                   5.0
      Name: Cost, dtype: float64
[22]: costs+=2
      costs
[22]: Store 1
                  24.5
      Store 1
                   4.5
                   7.0
      Store 2
      Name: Cost, dtype: float64
[23]:
     df
[23]:
                Name Item Purchased
                                       Cost
      Store 1
               Chris
                            Dog Food
                                       24.5
               Kevyn
                        Kitty Litter
                                        4.5
      Store 1
                           Bird Seed
                                        7.0
      Store 2
               Vinod
```

What happens here is that when the Jupyter notebook sees a line beginning with an exclamation mark, it sends the rest of the line to the operating system shell for evaluation.

```
[24]: !cat olympics.csv
```

```
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
,№ Summer,01 !,02 !,03 !,Total,№ Winter,01 !,02 !,03 !,Total,№ Games,01 !,02
!,03 !,Combined total
Afghanistan (AFG),13,0,0,2,2,0,0,0,0,0,13,0,0,2,2
Algeria (ALG), 12,5,2,8,15,3,0,0,0,0,15,5,2,8,15
Argentina (ARG),23,18,24,28,70,18,0,0,0,0,41,18,24,28,70
Armenia (ARM),5,1,2,9,12,6,0,0,0,0,11,1,2,9,12
Australasia (ANZ) [ANZ],2,3,4,5,12,0,0,0,0,0,2,3,4,5,12
Australia (AUS) [AUS] [Z],25,139,152,177,468,18,5,3,4,12,43,144,155,181,480
Austria (AUT), 26, 18, 33, 35, 86, 22, 59, 78, 81, 218, 48, 77, 111, 116, 304
Azerbaijan (AZE),5,6,5,15,26,5,0,0,0,0,10,6,5,15,26
Bahamas (BAH), 15, 5, 2, 5, 12, 0, 0, 0, 0, 0, 15, 5, 2, 5, 12
Bahrain (BRN),8,0,0,1,1,0,0,0,0,0,8,0,0,1,1
Barbados (BAR) [BAR],11,0,0,1,1,0,0,0,0,0,11,0,0,1,1
Belarus (BLR),5,12,24,39,75,6,6,4,5,15,11,18,28,44,90
Belgium (BEL), 25, 37, 52, 53, 142, 20, 1, 1, 3, 5, 45, 38, 53, 56, 147
Bermuda (BER),17,0,0,1,1,7,0,0,0,0,24,0,0,1,1
Bohemia (BOH) [BOH] [Z],3,0,1,3,4,0,0,0,0,0,3,0,1,3,4
Botswana (BOT),9,0,1,0,1,0,0,0,0,0,9,0,1,0,1
Brazil (BRA),21,23,30,55,108,7,0,0,0,0,28,23,30,55,108
British West Indies (BWI) [BWI],1,0,0,2,2,0,0,0,0,0,1,0,0,2,2
Bulgaria (BUL) [H], 19,51,85,78,214,19,1,2,3,6,38,52,87,81,220
Burundi (BDI),5,1,0,0,1,0,0,0,0,5,1,0,0,1
Cameroon (CMR),13,3,1,1,5,1,0,0,0,0,14,3,1,1,5
Canada (CAN), 25, 59, 99, 121, 279, 22, 62, 56, 52, 170, 47, 121, 155, 173, 449
Chile (CHI) [I],22,2,7,4,13,16,0,0,0,0,38,2,7,4,13
China (CHN) [CHN], 9, 201, 146, 126, 473, 10, 12, 22, 19, 53, 19, 213, 168, 145, 526
Colombia (COL),18,2,6,11,19,1,0,0,0,0,19,2,6,11,19
Costa Rica (CRC),14,1,1,2,4,6,0,0,0,0,20,1,1,2,4
Ivory Coast (CIV) [CIV],12,0,1,0,1,0,0,0,0,0,12,0,1,0,1
Croatia (CRO), 6, 6, 7, 10, 23, 7, 4, 6, 1, 11, 13, 10, 13, 11, 34
Cuba (CUB) [Z],19,72,67,70,209,0,0,0,0,19,72,67,70,209
Cyprus (CYP),9,0,1,0,1,10,0,0,0,0,19,0,1,0,1
Czech Republic (CZE) [CZE],5,14,15,15,44,6,7,9,8,24,11,21,24,23,68
Czechoslovakia (TCH) [TCH],16,49,49,45,143,16,2,8,15,25,32,51,57,60,168
Denmark (DEN) [Z],26,43,68,68,179,13,0,1,0,1,39,43,69,68,180
Djibouti (DJI) [B],7,0,0,1,1,0,0,0,0,0,7,0,0,1,1
Dominican Republic (DOM), 13, 3, 2, 1, 6, 0, 0, 0, 0, 0, 13, 3, 2, 1, 6
Ecuador (ECU),13,1,1,0,2,0,0,0,0,0,13,1,1,0,2
Egypt (EGY) [EGY] [Z],21,7,9,10,26,1,0,0,0,0,22,7,9,10,26
Eritrea (ERI),4,0,0,1,1,0,0,0,0,0,4,0,0,1,1
Estonia (EST), 11, 9, 9, 15, 33, 9, 4, 2, 1, 7, 20, 13, 11, 16, 40
Ethiopia (ETH),12,21,7,17,45,2,0,0,0,0,14,21,7,17,45
Finland (FIN), 24,101,84,117,302,22,42,62,57,161,46,143,146,174,463
France (FRA) [0] [P] [Z],27,202,223,246,671,22,31,31,47,109,49,233,254,293,780
Gabon (GAB),9,0,1,0,1,0,0,0,0,0,9,0,1,0,1
Georgia (GEO),5,6,5,14,25,6,0,0,0,0,11,6,5,14,25
Germany (GER) [GER] [Z],15,174,182,217,573,11,78,78,53,209,26,252,260,270,782
```

```
United Team of Germany (EUA) [EUA], 3, 28, 54, 36, 118, 3, 8, 6, 5, 19, 6, 36, 60, 41, 137
East Germany (GDR) [GDR],5,153,129,127,409,6,39,36,35,110,11,192,165,162,519
West Germany (FRG) [FRG],5,56,67,81,204,6,11,15,13,39,11,67,82,94,243
Ghana (GHA) [GHA],13,0,1,3,4,1,0,0,0,0,14,0,1,3,4
Great Britain (GBR) [GBR]
[Z],27,236,272,272,780,22,10,4,12,26,49,246,276,284,806
Greece (GRE) [Z],27,30,42,39,111,18,0,0,0,0,45,30,42,39,111
Grenada (GRN),8,1,0,0,1,0,0,0,0,0,8,1,0,0,1
Guatemala (GUA), 13, 0, 1, 0, 1, 1, 0, 0, 0, 0, 14, 0, 1, 0, 1
Guyana (GUY) [GUY],16,0,0,1,1,0,0,0,0,0,16,0,0,1,1
Haiti (HAI) [J],14,0,1,1,2,0,0,0,0,0,14,0,1,1,2
Hong Kong (HKG) [HKG], 15, 1, 1, 1, 3, 4, 0, 0, 0, 0, 19, 1, 1, 1, 3
Hungary (HUN), 25, 167, 144, 165, 476, 22, 0, 2, 4, 6, 47, 167, 146, 169, 482
Iceland (ISL), 19,0,2,2,4,17,0,0,0,0,36,0,2,2,4
India (IND) [F],23,9,6,11,26,9,0,0,0,0,32,9,6,11,26
Indonesia (INA), 14,6,10,11,27,0,0,0,0,14,6,10,11,27
Iran (IRI) [K],15,15,20,25,60,10,0,0,0,0,25,15,20,25,60
Iraq (IRQ),13,0,0,1,1,0,0,0,0,0,13,0,0,1,1
Ireland (IRL),20,9,8,12,29,6,0,0,0,0,26,9,8,12,29
Israel (ISR),15,1,1,5,7,6,0,0,0,0,21,1,1,5,7
Italy (ITA) [M] [S],26,198,166,185,549,22,37,34,43,114,48,235,200,228,663
Jamaica (JAM) [JAM],16,17,30,20,67,7,0,0,0,0,23,17,30,20,67
Japan (JPN), 21, 130, 126, 142, 398, 20, 10, 17, 18, 45, 41, 140, 143, 160, 443
Kazakhstan (KAZ),5,16,17,19,52,6,1,3,3,7,11,17,20,22,59
Kenya (KEN), 13, 25, 32, 29, 86, 3, 0, 0, 0, 0, 16, 25, 32, 29, 86
North Korea (PRK),9,14,12,21,47,8,0,1,1,2,17,14,13,22,49
South Korea (KOR), 16,81,82,80,243,17,26,17,10,53,33,107,99,90,296
Kuwait (KUW),12,0,0,2,2,0,0,0,0,0,12,0,0,2,2
Kyrgyzstan (KGZ),5,0,1,2,3,6,0,0,0,0,11,0,1,2,3
Latvia (LAT), 10, 3, 11, 5, 19, 10, 0, 4, 3, 7, 20, 3, 15, 8, 26
Lebanon (LIB),16,0,2,2,4,16,0,0,0,0,32,0,2,2,4
Liechtenstein (LIE),16,0,0,0,0,18,2,2,5,9,34,2,2,5,9
Lithuania (LTU),8,6,5,10,21,8,0,0,0,0,16,6,5,10,21
Luxembourg (LUX) [0],22,1,1,0,2,8,0,2,0,2,30,1,3,0,4
Macedonia (MKD),5,0,0,1,1,5,0,0,0,0,10,0,0,1,1
Malaysia (MAS) [MAS],12,0,3,3,6,0,0,0,0,0,12,0,3,3,6
Mauritius (MRI),8,0,0,1,1,0,0,0,0,0,8,0,0,1,1
Mexico (MEX),22,13,21,28,62,8,0,0,0,0,30,13,21,28,62
Moldova (MDA),5,0,2,5,7,6,0,0,0,0,11,0,2,5,7
Mongolia (MGL),12,2,9,13,24,13,0,0,0,0,25,2,9,13,24
Montenegro (MNE),2,0,1,0,1,2,0,0,0,0,4,0,1,0,1
Morocco (MAR), 13,6,5,11,22,6,0,0,0,0,19,6,5,11,22
Mozambique (MOZ),9,1,0,1,2,0,0,0,0,0,9,1,0,1,2
Namibia (NAM), 6, 0, 4, 0, 4, 0, 0, 0, 0, 0, 6, 0, 4, 0, 4
Netherlands (NED) [Z],25,77,85,104,266,20,37,38,35,110,45,114,123,139,376
Netherlands Antilles (AHO) [AHO] [I],13,0,1,0,1,2,0,0,0,0,15,0,1,0,1
New Zealand (NZL) [NZL],22,42,18,39,99,15,0,1,0,1,37,42,19,39,100
Niger (NIG), 11, 0, 0, 1, 1, 0, 0, 0, 0, 0, 11, 0, 0, 1, 1
```

```
Nigeria (NGR), 15, 3, 8, 12, 23, 0, 0, 0, 0, 0, 15, 3, 8, 12, 23
Norway (NOR) [Q],24,56,49,43,148,22,118,111,100,329,46,174,160,143,477
Pakistan (PAK), 16, 3, 3, 4, 10, 2, 0, 0, 0, 0, 18, 3, 3, 4, 10
Panama (PAN),16,1,0,2,3,0,0,0,0,0,16,1,0,2,3
Paraguay (PAR),11,0,1,0,1,1,0,0,0,0,12,0,1,0,1
Peru (PER) [L],17,1,3,0,4,2,0,0,0,0,19,1,3,0,4
Philippines (PHI),20,0,2,7,9,4,0,0,0,0,24,0,2,7,9
Poland (POL), 20,64,82,125,271,22,6,7,7,20,42,70,89,132,291
Portugal (POR),23,4,8,11,23,7,0,0,0,0,30,4,8,11,23
Puerto Rico (PUR),17,0,2,6,8,6,0,0,0,0,23,0,2,6,8
Qatar (QAT),8,0,0,4,4,0,0,0,0,0,8,0,0,4,4
Romania (ROU), 20,88,94,119,301,20,0,0,1,1,40,88,94,120,302
Russia (RUS) [RUS],5,132,121,142,395,6,49,40,35,124,11,181,161,177,519
Russian Empire (RU1) [RU1],3,1,4,3,8,0,0,0,0,0,3,1,4,3,8
Soviet Union (URS) [URS],9,395,319,296,1010,9,78,57,59,194,18,473,376,355,1204
Unified Team (EUN) [EUN],1,45,38,29,112,1,9,6,8,23,2,54,44,37,135
Saudi Arabia (KSA),10,0,1,2,3,0,0,0,0,0,10,0,1,2,3
Senegal (SEN),13,0,1,0,1,5,0,0,0,0,18,0,1,0,1
Serbia (SRB) [SRB],3,1,2,4,7,2,0,0,0,0,5,1,2,4,7
Serbia and Montenegro (SCG) [SCG], 3, 2, 4, 3, 9, 3, 0, 0, 0, 0, 6, 2, 4, 3, 9
Singapore (SIN), 15,0,2,2,4,0,0,0,0,15,0,2,2,4
Slovakia (SVK) [SVK],5,7,9,8,24,6,2,2,1,5,11,9,11,9,29
Slovenia (SLO), 6, 4, 6, 9, 19, 7, 2, 4, 9, 15, 13, 6, 10, 18, 34
South Africa (RSA), 18, 23, 26, 27, 76, 6, 0, 0, 0, 0, 24, 23, 26, 27, 76
Spain (ESP) [Z],22,37,59,35,131,19,1,0,1,2,41,38,59,36,133
Sri Lanka (SRI) [SRI],16,0,2,0,2,0,0,0,0,0,16,0,2,0,2
Sudan (SUD), 11, 0, 1, 0, 1, 0, 0, 0, 0, 0, 11, 0, 1, 0, 1
Suriname (SUR) [E],11,1,0,1,2,0,0,0,0,0,11,1,0,1,2
Sweden (SWE) [Z],26,143,164,176,483,22,50,40,54,144,48,193,204,230,627
Switzerland (SUI),27,47,73,65,185,22,50,40,48,138,49,97,113,113,323
Syria (SYR),12,1,1,1,3,0,0,0,0,0,12,1,1,1,3
Chinese Taipei (TPE) [TPE] [TPE2],13,2,7,12,21,11,0,0,0,0,24,2,7,12,21
Tajikistan (TJK),5,0,1,2,3,4,0,0,0,0,9,0,1,2,3
Tanzania (TAN) [TAN],12,0,2,0,2,0,0,0,0,0,12,0,2,0,2
Thailand (THA), 15, 7, 6, 11, 24, 3, 0, 0, 0, 0, 18, 7, 6, 11, 24
Togo (TOG),9,0,0,1,1,1,0,0,0,0,10,0,0,1,1
Tonga (TGA),8,0,1,0,1,1,0,0,0,0,9,0,1,0,1
Trinidad and Tobago (TRI) [TRI], 16,2,5,11,18,3,0,0,0,0,19,2,5,11,18
Tunisia (TUN),13,3,3,4,10,0,0,0,0,0,13,3,3,4,10
Turkey (TUR), 21, 39, 25, 24, 88, 16, 0, 0, 0, 0, 37, 39, 25, 24, 88
Uganda (UGA),14,2,3,2,7,0,0,0,0,0,14,2,3,2,7
Ukraine (UKR),5,33,27,55,115,6,2,1,4,7,11,35,28,59,122
United Arab Emirates (UAE),8,1,0,0,1,0,0,0,0,0,8,1,0,0,1
United States (USA) [P] [Q] [R]
[Z], 26, 976, 757, 666, 2399, 22, 96, 102, 84, 282, 48, 1072, 859, 750, 2681
Uruguay (URU), 20, 2, 2, 6, 10, 1, 0, 0, 0, 0, 21, 2, 2, 6, 10
Uzbekistan (UZB),5,5,5,10,20,6,1,0,0,1,11,6,5,10,21
Venezuela (VEN), 17, 2, 2, 8, 12, 4, 0, 0, 0, 0, 21, 2, 2, 8, 12
```

```
Vietnam (VIE),14,0,2,0,2,0,0,0,0,0,14,0,2,0,2
Virgin Islands (ISV),11,0,1,0,1,7,0,0,0,0,18,0,1,0,1
Yugoslavia (YUG) [YUG],16,26,29,28,83,14,0,3,1,4,30,26,32,29,87
Independent Olympic Participants (IOP) [IOP],1,0,1,2,3,0,0,0,0,0,1,0,1,2,3
Zambia (ZAM) [ZAM],12,0,1,1,2,0,0,0,0,0,12,0,1,1,2
Zimbabwe (ZIM) [ZIM],12,3,4,1,8,1,0,0,0,0,13,3,4,1,8
Mixed team (ZZX) [ZZX],3,8,5,4,17,0,0,0,0,0,3,8,5,4,17
Totals,27,4809,4775,5130,14714,22,959,958,948,2865,49,5768,5733,6078,17579
```

We can read this into a DataFrame by calling the read_csv function of the module. When we look at the DataFrame we see that the first cell has an NaN in it since it's an empty value, and the rows have been automatically indexed for us.

[6]:				0		1	2	3	4	5	6	7	8	\
	0			NaN	N	Summer	01 !	02!	03!	Total	♪ Winter	01 !	02!	
	1	Afgl	nanistan	(AFG)		13	0	0	2	2	0	0	0	
	2		Algeria	(ALG)		12	5	2	8	15	3	0	0	
	3	Argentina Armenia		(ARG)		23	18	24	28	70	18	0	0	
	4			(ARM)		5	1	2	9	12	6	0	0	
		Ş	9 10		11	12	13	14		1	5			
	0	03	! Total	№ Gam	es	01 !	02!	03!	Combine	ed total	1			
	1	(0		13	0	0	2		2	2			
	2	(0		15	5	2	8		1	5			
	3	(0		41	18	24	28		70	0			
	4	(0		11	1	2	9		13	2			

It seems pretty clear that the first row of data in the DataFrame is what we really want to see as the column names. It also seems like the first column in the data is the country name, which we would like to make an index.

Read csv has a number of parameters that we can use to indicate to Pandas how rows and columns should be labeled.

For instance, we can use the index col to indicate which column should be the index and we can also use the header parameter to indicate which row from the data file should be used as the header.

```
[17]: df=pd.read_csv('olympics.csv',index_col=0,skiprows=1) df
```

[17]:	№ Summer	01 !	02!	03!	\
Afghanistan (AFG)	13	0	0	2	
Algeria (ALG)	12	5	2	8	
Argentina (ARG)	23	18	24	28	
Armenia (ARM)	5	1	2	9	
Australasia (ANZ) [ANZ]	2	3	4	5	
•••		•••	•••		

```
Independent Olympic Participants (IOP) [IOP]
                                                         1
                                                                             2
Zambia (ZAM) [ZAM]
                                                        12
                                                                0
                                                                      1
                                                                             1
Zimbabwe (ZIM) [ZIM]
                                                        12
                                                                3
                                                                      4
                                                                             1
                                                         3
                                                                      5
Mixed team (ZZX) [ZZX]
                                                                8
                                                                             4
Totals
                                                        27
                                                            4809
                                                                  4775
                                                                         5130
                                                  Total № Winter
                                                                   01 !.1
                                                                             02 !.1
                                                      2
                                                                 0
Afghanistan (AFG)
                                                                          0
                                                                                  0
                                                                 3
                                                                                  0
Algeria (ALG)
                                                     15
                                                                          0
Argentina (ARG)
                                                     70
                                                                18
                                                                          0
                                                                                  0
Armenia (ARM)
                                                     12
                                                                          0
                                                                 6
                                                                                  0
Australasia (ANZ) [ANZ]
                                                     12
                                                                 0
                                                                          0
Independent Olympic Participants (IOP) [IOP]
                                                      3
                                                                 0
                                                                          0
                                                                                  0
Zambia (ZAM) [ZAM]
                                                      2
                                                                 0
                                                                                  0
                                                                          0
                                                      8
                                                                                  0
Zimbabwe (ZIM) [ZIM]
                                                                 1
                                                                          0
                                                                          0
                                                                                  0
Mixed team (ZZX) [ZZX]
                                                     17
                                                                 0
Totals
                                                  14714
                                                                22
                                                                       959
                                                                                958
                                                  03 !.1
                                                          Total.1
                                                                    J Games
Afghanistan (AFG)
                                                       0
                                                                 0
                                                                          13
Algeria (ALG)
                                                       0
                                                                 0
                                                                          15
Argentina (ARG)
                                                       0
                                                                 0
                                                                          41
Armenia (ARM)
                                                       0
                                                                 0
                                                                          11
Australasia (ANZ) [ANZ]
                                                       0
                                                                 0
                                                                           2
Independent Olympic Participants (IOP) [IOP]
                                                       0
                                                                 0
                                                                           1
Zambia (ZAM) [ZAM]
                                                       0
                                                                 0
                                                                          12
Zimbabwe (ZIM) [ZIM]
                                                       0
                                                                 0
                                                                          13
                                                       0
                                                                 0
                                                                          3
Mixed team (ZZX) [ZZX]
Totals
                                                     948
                                                              2865
                                                                          49
                                                  01 !.2
                                                          02 !.2
                                                                   03 !.2
                                                                0
Afghanistan (AFG)
                                                       0
                                                                         2
                                                       5
                                                                2
Algeria (ALG)
                                                                        8
Argentina (ARG)
                                                      18
                                                               24
                                                                       28
Armenia (ARM)
                                                       1
                                                                2
                                                                        9
Australasia (ANZ) [ANZ]
                                                       3
                                                                        5
Independent Olympic Participants (IOP) [IOP]
                                                       0
                                                                        2
                                                                1
Zambia (ZAM) [ZAM]
                                                       0
                                                                1
                                                                         1
Zimbabwe (ZIM) [ZIM]
                                                       3
                                                                4
                                                                         1
Mixed team (ZZX) [ZZX]
                                                       8
                                                                5
                                                                         4
Totals
                                                    5768
                                                            5733
                                                                     6078
                                                  Combined total
```

Afghanistan (AFG)

```
Algeria (ALG)
                                                             15
                                                             70
Argentina (ARG)
Armenia (ARM)
                                                             12
Australasia (ANZ) [ANZ]
                                                             12
Independent Olympic Participants (IOP) [IOP]
                                                              3
Zambia (ZAM) [ZAM]
                                                              2
Zimbabwe (ZIM) [ZIM]
                                                              8
Mixed team (ZZX) [ZZX]
                                                             17
Totals
                                                          17579
```

[147 rows x 15 columns]

TypeError

→last)

If we head to the page we could see that instead of running gold, silver and bronze in the pages, these nice little icons with a one, a two, and a three in them In our csv file these were represented with the strings 01!, 02!, and so on.

But this labeling isn't really as clear as it could be, so we should clean up the data file. We can of course do this just by going and editing the CSV file directly, but we can also set the column names using the Pandas name property.

```
<ipython-input-4-6b3fa8e3c965> in <module>
```

Traceback (most recent call

TypeError: 'DataFrame' object is not callable

```
[28]: (df['Gold'] - df['Gold.1']).idxmax()
```

[28]: 'Totals'

1.4 Querying a DataFrame

Before we talk about how to query data frames, we need to talk about Boolean masking. Boolean masking is the heart of fast and efficient querying in NumPy. It's analogous a bit to masking used in other computational areas.

A Boolean mask is an array which can be of one dimension like a series, or two dimensions like a data frame, where each of the values in the array are either true or false. This array is essentially overlaid on top of the data structure that we're querying. And any cell aligned with the true value will be admitted into our final result, and any sign aligned with a false value will not.

Boolean masks are created by applying operators directly to the pandas series or DataFrame objects. For instance, in our Olympics data set, you might be interested in seeing only those countries who have achieved a gold medal at the summer Olympics. To build a Boolean mask for this query, we project the gold column using the indexing operator and apply the greater than operator with a comparison value of zero. This is essentially broadcasting a comparison operator, greater than, with the results being returned as a Boolean series. The resultant series is indexed where the value of each cell is either true or false depending on whether a country has won at least one gold medal, and the index is the country name.

```
[39]: #The below is boolean mask for Gold column df['Gold']>0
```

```
[39]: Afghanistan (AFG)
                                                        False
      Algeria (ALG)
                                                          True
      Argentina (ARG)
                                                         True
      Armenia (ARM)
                                                         True
      Australasia (ANZ) [ANZ]
                                                          True
      Independent Olympic Participants (IOP) [IOP]
                                                        False
      Zambia (ZAM) [ZAM]
                                                        False
      Zimbabwe (ZIM) [ZIM]
                                                         True
      Mixed team (ZZX) [ZZX]
                                                         True
      Totals
                                                         True
      Name: Gold, Length: 147, dtype: bool
```

What we want to do next is overlay that mask on the data frame. We can do this using the where function. The where function takes a Boolean mask as a condition, applies it to the data frame or series, and returns a new data frame or series of the same shape. Let's apply this Boolean mask to our Olympics data and create a data frame of only those countries who have won a gold at a summer games.

```
[40]: #Creating a seperate df that matches our boolean mask only_gold=df.where(df['Gold']>0) only_gold.head()
```

	only_gold.head()									
[40]:	Afghanistan (AFG)		# Summer		Silver NaN				er \	
	•								.0	
	Algeria (ALG)		12.0		2.0					
	Argentina (ARG)		23.0						.0	
	Armenia (ARM)		5.0	1.0			12.0		.0	
	Australasia (ANZ)	[ANZ]	2.0	3.0	4.0	5.0	12.0	0	.0	
			Gold.1	Silver.	l Bron:	ze.1 To	otal.1	# Games	Gold.2	\
	Afghanistan (AFG)		NaN	Nal	V	NaN	NaN	NaN	NaN	
	Algeria (ALG)		0.0	0.0)	0.0	0.0	15.0	5.0	
	Argentina (ARG)		0.0	0.0)	0.0	0.0	41.0	18.0	
	Armenia (ARM)		0.0	0.0)	0.0	0.0	11.0	1.0	
	Australasia (ANZ)	[ANZ]	0.0	0.0)	0.0	0.0	2.0	3.0	
			Silver.2	Bronze	e.2 Cor	mbined t	otal			
	Afghanistan (AFG)		NaN	· I	NaN		NaN			
	Algeria (ALG)		2.0		3.0		15.0			
	Argentina (ARG)		24.0	28	3.0		70.0			
	Armenia (ARM)		2.0		9.0		12.0			
	Australasia (ANZ)	[ANZ]	4.0		5.0		12.0			

We see that the resulting data frame keeps the original indexed values, and only data from countries that met the condition are retained. All of the countries which did not meet the condition have NaN data instead. This is okay. Most statistical functions built into the data frame object ignore values of NaN.

For instance, if we call the df.count on the only gold data frame, we see that there are 100 countries which have had gold medals awarded at the summer games, while if we call count on the original data frame, we see that there are 147 countries total.

```
[41]: only_gold['Gold'].count()

[41]: 100

[42]: df['Gold'].count()
```

[42]: 147

Often we want to drop those rows which have no data. To do this, we can use the drop NA function.

You can optionally provide drop NA the axes it should be considering. Remember that the axes is just an indicator for the columns or rows and that the default is zero, which means rows.

[43]:	only_gold.dropna()						
[43]:		# Summer	Gold	Silver	Bronze	Total \	
	Algeria (ALG)	12.0	5.0	2.0	8.0	15.0	
	Argentina (ARG)	23.0	18.0	24.0	28.0	70.0	
	Armenia (ARM)	5.0	1.0	2.0	9.0	12.0	
	Australasia (ANZ) [ANZ]	2.0	3.0	4.0	5.0	12.0	
	Australia (AUS) [AUS] [Z]	25.0	139.0	152.0	177.0	468.0	
		•••		•••	•••		
	Venezuela (VEN)	17.0	2.0	2.0	8.0	12.0	
	Yugoslavia (YUG) [YUG]	16.0	26.0	29.0	28.0	83.0	
	Zimbabwe (ZIM) [ZIM]	12.0	3.0	4.0	1.0	8.0	
	Mixed team (ZZX) [ZZX]	3.0	8.0	5.0	4.0	17.0	
	Totals	27.0	4809.0	4775.0	5130.0 1	4714.0	
		# Winter	Gold.1	Silver.	1 Bronze.	1 Total.	1 \
	Algeria (ALG)	3.0	0.0	0.0	0.	0.0)
	Argentina (ARG)	18.0	0.0	0.0	0.	0.0)
	Armenia (ARM)	6.0	0.0	0.0	0.	0.0	0
	Australasia (ANZ) [ANZ]	0.0	0.0	0.0	0.	0.0	0
	Australia (AUS) [AUS] [Z]	18.0	5.0	3.0	0 4.	0 12.0)
							•
	Venezuela (VEN)	4.0		0.0			
	Yugoslavia (YUG) [YUG]	14.0		3.0			
	Zimbabwe (ZIM) [ZIM]	1.0		0.0			
	Mixed team (ZZX) [ZZX]	0.0		0.0		0.0	
	Totals	22.0	959.0	958.0	948.	0 2865.0	0
						Combine	
	Algeria (ALG)	15.0	5.0	2.0			15.0
	Argentina (ARG)	41.0	18.0	24.0	28.0)	70.0
	Armenia (ARM)	11.0	1.0	2.0)	12.0
	Australasia (ANZ) [ANZ]	2.0	3.0	4.0	5.0)	12.0
	Australia (AUS) [AUS] [Z]	43.0	144.0	155.0	181.0)	480.0
	•••	•••					
	Venezuela (VEN)	21.0	2.0	2.0	8.0		12.0
	Yugoslavia (YUG) [YUG]	30.0	26.0	32.0	29.0)	87.0
	Zimbabwe (ZIM) [ZIM]	13.0	3.0	4.0	1.0)	8.0
	Mixed team (ZZX) [ZZX]	3.0	8.0	5.0	4.0)	17.0
	Totals	49.0	5768.0	5733.0	6078.0)	17579.0

[100 rows x 15 columns]

One more thing to keep in mind if you're not used to Boolean or bit masking for data reduction. The output of two Boolean masks being compared with logical operators is another Boolean mask.

This means that you can chain together a bunch of and/or statements in order to create more complex queries, and the result is a single Boolean mask.

```
[44]: #No of Countries that have won Gold in Summmeror Winter olympics
      len(df[(df['Gold']>0) | (df['Gold.1'])]>0)
[44]: 101
[45]: #Countries that have won Gold only in Winter Olympics
      df[(df['Gold.1']>0) & (df['Gold']==0)]
[45]:
                                                                     # Winter
                            # Summer
                                      Gold
                                            Silver
                                                    Bronze
                                                             Total
      Liechtenstein (LIE)
                                  16
                                         0
                                                  0
                                                          0
                                                                  0
                                                                           18
                                                                                    2
                            Silver.1
                                      Bronze.1
                                                 Total.1
                                                          # Games
                                                                    Gold.2
                                              5
      Liechtenstein (LIE)
                                   2
                                                       9
                                                               34
                                                                         2
                                                                                   2
                            Bronze.2
                                      Combined total
      Liechtenstein (LIE)
                                   5
```

1.5 Indexing DataFrames

Both series and DataFrames can have indices applied to them. The index is essentially a row level label, and we know that rows correspond to axis zero.

Indices can either be inferred, such as when we create a new series without an index, in which case we get numeric values, or they can be set explicitly, like when we use the dictionary object to create the series, or when we loaded data from the CSV file and specified the header.

Another option for setting an index is to use the set_index function. This function takes a list of columns and promotes those columns to an index. Set index is a destructive process, it doesn't keep the current index. If you want to keep the current index, you need to manually create a new column and copy into it values from the index attribute.

```
[46]: #Making the current index 'Countries' as one of the usual column df ['country'] = df.index

#setting 'Gold' column as new index of the df df = df.set_index('Gold') df.head()
```

```
[46]:
             # Summer
                       Silver Bronze
                                          Total
                                                  # Winter Gold.1 Silver.1 Bronze.1
      Gold
      0
                    13
                              0
                                       2
                                              2
                                                          0
                                                                   0
                                                                              0
                                                                                         0
                              2
      5
                    12
                                       8
                                              15
                                                          3
                                                                   0
                                                                              0
                                                                                          0
                             24
                                      28
      18
                    23
                                              70
                                                         18
                                                                   0
                                                                              0
                                                                                          0
                                       9
                                                          6
      1
                     5
                              2
                                              12
                                                                   0
                                                                              0
                                                                                          0
      3
                     2
                                       5
                                              12
```

	Total.1	# Games	Gold.2	Silver.2	Bronze.2	Combined total	\
Gold							
0	0	13	0	0	2	2	
5	0	15	5	2	8	15	
18	0	41	18	24	28	70	
1	0	11	1	2	9	12	
3	0	2	3	4	5	12	
		С	ountry				
Gold							
0	Af	ghanistan	(AFG)				
5	Algeria (ALG)						
18		Argentina	(ARG)				
1		Armenia	(ARM)				
3	Australa	sia (ANZ)	[ANZ]				

We can get rid of the index completely by calling the function **reset_index**. This promotes the index into a column and creates a default numbered index.

```
[47]: #Getting rid of the current index
df=df.reset_index()
df.head()
```

	di	.head()															
[47]:		Gold	#	Summer	Sil	/er	Bro	nze	Tot	al	# V	/int	er	Gold.1	Silver.1	. \		
	0	0		13		0		2		2			0	0	C)		
	1	5		12		2		8		15			3	0	C)		
	2	18		23		24		28		70			18	0	C)		
	3	1		5		2		9		12			6	0	C)		
	4	3		2		4		5		12			0	0	C)		
		Bronz	е.:	l Total.	1 ‡	‡ Ga	mes	Gol	d.2	Si	lver	2.2	Br	onze.2	Combined	tota	1	\
	0		()	0		13		0			0		2			2	
	1		()	0		15		5			2		8		1	.5	
	2		()	0		41		18			24		28		7	0	
	3		()	0		11		1			2		9		1	.2	
	4		()	0		2		3			4		5		1	.2	
					cour	ntry												
	0		A	fghanista	n (1	AFG)												
	1			Algeri	a (1	ALG)												
	2			Argentin	a (<i>I</i>	ARG)												
	3			Armeni	a (<i>I</i>	ARM)												

One nice feature of pandas is that it has the option to do multi-level indexing

4 Australasia (ANZ) [ANZ]

```
[2]: #Loading Census dataset
     df=pd.read_csv('census.csv')
     df.head()
[2]:
        SUMLEV
                REGION
                         DIVISION
                                   STATE
                                           COUNTY
                                                     STNAME
                                                                     CTYNAME \
            40
                      3
                                                   Alabama
     0
                                6
                                                                     Alabama
                      3
     1
            50
                                6
                                        1
                                                1
                                                    Alabama
                                                             Autauga County
     2
            50
                      3
                                6
                                        1
                                                3
                                                    Alabama
                                                             Baldwin County
     3
                      3
                                6
            50
                                        1
                                                5
                                                    Alabama
                                                             Barbour County
            50
                      3
                                 6
                                        1
                                                    Alabama
                                                                Bibb County
        CENSUS2010POP
                        ESTIMATESBASE2010 POPESTIMATE2010
                                                                 RDOMESTICMIG2011
     0
              4779736
                                   4780127
                                                     4785161
                                                                          0.002295
     1
                54571
                                     54571
                                                       54660
                                                                          7.242091
     2
                182265
                                    182265
                                                      183193
                                                                         14.832960
     3
                                                       27341
                27457
                                     27457
                                                                         -4.728132
     4
                 22915
                                     22919
                                                       22861
                                                                         -5.527043
        RDOMESTICMIG2012
                           RDOMESTICMIG2013
                                              RDOMESTICMIG2014
                                                                 RDOMESTICMIG2015
     0
                -0.193196
                                   0.381066
                                                       0.582002
                                                                         -0.467369
     1
                -2.915927
                                   -3.012349
                                                       2.265971
                                                                         -2.530799
     2
                17.647293
                                   21.845705
                                                      19.243287
                                                                         17.197872
     3
                -2.500690
                                   -7.056824
                                                      -3.904217
                                                                        -10.543299
                -5.068871
                                   -6.201001
                                                      -0.177537
                                                                          0.177258
        RNETMIG2011 RNETMIG2012 RNETMIG2013 RNETMIG2014 RNETMIG2015
     0
                                                     1.724718
           1.030015
                         0.826644
                                       1.383282
                                                                  0.712594
     1
           7.606016
                        -2.626146
                                      -2.722002
                                                     2.592270
                                                                 -2.187333
     2
                                      22.727626
          15.844176
                        18.559627
                                                    20.317142
                                                                 18.293499
     3
          -4.874741
                        -2.758113
                                      -7.167664
                                                    -3.978583
                                                                -10.543299
          -5.088389
                        -4.363636
                                      -5.403729
                                                     0.754533
                                                                   1.107861
     [5 rows x 100 columns]
    I want to see a list of all the unique values in a given column.
[3]: #Unique values present in SUMLEV column
     df['SUMLEV'].unique()
[3]: array([40, 50])
[5]: #Retaining data only that has 'SUMLEV' equal to 50
     df=df[df['SUMLEV']==50]
     df.head()
                REGION DIVISION STATE
                                          COUNTY
                                                     STNAME
                                                                     CTYNAME \
[5]:
        SUMI.F.V
     1
            50
                      3
                                6
                                        1
                                                   Alabama
                                                            Autauga County
```

```
2
       50
                3
                                 1
                                         3 Alabama
                                                     Baldwin County
3
       50
                3
                          6
                                 1
                                         5 Alabama
                                                     Barbour County
4
       50
                3
                          6
                                 1
                                         7 Alabama
                                                         Bibb County
5
                3
       50
                                 1
                                         9 Alabama
                                                       Blount County
  CENSUS2010POP ESTIMATESBASE2010 POPESTIMATE2010 ...
                                                         RDOMESTICMIG2011 \
                                                                  7.242091
1
           54571
                              54571
                                                54660 ...
2
          182265
                             182265
                                               183193 ...
                                                                 14.832960
3
                                                27341 ...
           27457
                              27457
                                                                 -4.728132
4
                                                22861 ...
           22915
                              22919
                                                                 -5.527043
5
           57322
                              57322
                                                57373 ...
                                                                  1.807375
  RDOMESTICMIG2012 RDOMESTICMIG2013 RDOMESTICMIG2014 RDOMESTICMIG2015
                                                2.265971
          -2.915927
                            -3.012349
1
                                                                 -2.530799
2
          17.647293
                            21.845705
                                              19.243287
                                                                 17.197872
3
          -2.500690
                            -7.056824
                                              -3.904217
                                                                -10.543299
4
          -5.068871
                            -6.201001
                                               -0.177537
                                                                  0.177258
5
          -1.177622
                            -1.748766
                                               -2.062535
                                                                 -1.369970
  RNETMIG2011 RNETMIG2012 RNETMIG2013 RNETMIG2014 RNETMIG2015
1
     7.606016
                  -2.626146
                               -2.722002
                                              2.592270
                                                          -2.187333
                  18.559627
                               22.727626
2
    15.844176
                                            20.317142
                                                          18.293499
3
    -4.874741
                 -2.758113
                              -7.167664
                                            -3.978583
                                                         -10.543299
4
    -5.088389
                 -4.363636
                              -5.403729
                                              0.754533
                                                           1.107861
5
      1.859511
                  -0.848580
                               -1.402476
                                            -1.577232
                                                          -0.884411
```

[5 rows x 100 columns]

```
[6]: #Getting rows that are summaries of state level and just keep county data
     #Selected columns to be used
     columns_to_keep = ['STNAME',
                         'CTYNAME',
                         'BIRTHS2010',
                         'BIRTHS2011',
                         'BIRTHS2012'.
                         'BIRTHS2013',
                         'BIRTHS2014',
                         'BIRTHS2015',
                         'POPESTIMATE2010',
                         'POPESTIMATE2011',
                         'POPESTIMATE2012',
                         'POPESTIMATE2013',
                         'POPESTIMATE2014',
                         'POPESTIMATE2015']
     df=df[columns_to_keep]
     df.head()
```

[6]:	STNA 1 Alaba 2 Alaba 3 Alaba 4 Alaba 5 Alaba	Autauga County ama Baldwin County ama Barbour County ama Bibb County ama Blount County	151 517 70 44 183	636 2187 335 266 744	615 2092 300 245 710	BIRTHS2013 \ 574 2160 283 259 646
	1 2 3 4 5	HS2014 BIRTHS2015 623 600 2186 2240 260 269 247 253 618 603	183 27 22	2010 PUPEST 1660 3193 7341 2861 7373	IMATE2011 PO 55253 186659 27226 22733 57711	PESTIMATE2012 \
	POPES 1 2 3 4 5	5TIMATE2013 POPEST 55038 195126 26973 22512 57734	IMATE2014 PC 55290 199713 26815 22549 57658	DPESTIMATE20 553- 2037 264- 225- 576	47 09 89 83	
[7]:		<pre>Dual index et_index(['STNAME',</pre>	'CTYNAME'])			
[7]:	STNAME Alabama	CTYNAME Autauga County Baldwin County Barbour County Bibb County Blount County	BIRTHS2010 151 517 70 44 183	636 2187 335 266 744	615 2092 300 245 710	574 2160 283 259 646
	wyoming	Sweetwater County Teton County Uinta County Washakie County Weston County	167 76 73 26 26	 640 259 324 108 81 BIRTHS2015	595 230 311 90 74	657 261 316 95 93
	STNAME Alabama 	CTYNAME Autauga County Baldwin County Barbour County Bibb County Blount County	623 2186 260 247 618	600 2240 269 253 603	54 183 27 22	660

Wyomir	ng Sweetwater County	629	620	43593	
•	Teton County	249	269	21297	
	Uinta County	316	316	21102	
	Washakie County	96	90	8545	
	Weston County	77	79	7181	
	•				
		POPESTIMATE2011	POPESTIMATE2012	POPESTIMATE2013	\
STNAME	E CTYNAME				
Alaban	na Autauga County	55253	55175	55038	
	Baldwin County	186659	190396	195126	
	Barbour County	27226	27159	26973	
	Bibb County	22733	22642	22512	
	Blount County	57711	57776	57734	
•••		•••	•••	•••	
Wyomir	ng Sweetwater County	44041	45104	45162	
	Teton County	21482	21697	22347	
	Uinta County	20912	20989	21022	
	Washakie County	8469	8443	8443	
	Weston County	7114	7065	7160	
		POPESTIMATE2014	POPESTIMATE2015		
STNAME	E CTYNAME				
Alaban	na Autauga County	55290	55347		
	Baldwin County	199713	203709		
	Barbour County	26815	26489		
	Bibb County	22549	22583		
	Blount County	57658	57673		
•••		•••	•••		
Wyomir	ng Sweetwater County	44925	44626		
	Teton County	22905	23125		
	Uinta County	20903	20822		
	Washakie County	8316	8328		
	Weston County	7185	7234		

[3142 rows x 12 columns]

When you use a MultiIndex, you must provide the arguments in order by the level you wish to query. Inside of the index, each column is called a level and the outermost column is level zero

[8]: BIRTHS2010 977
BIRTHS2011 3826
BIRTHS2012 3780

```
BIRTHS2013
                           3662
     BIRTHS2014
                           3683
     BIRTHS2015
                           3709
     POPESTIMATE2010
                         345563
     POPESTIMATE2011
                         349048
     POPESTIMATE2012
                         351213
     POPESTIMATE2013
                         354289
    POPESTIMATE2014
                         357029
     POPESTIMATE2015
                         358880
     Name: (Michigan, Washtenaw County), dtype: int64
[9]: #To compare population of two different county of same states
     df.loc[ [('Michigan', 'Washtenaw County'),
              ('Michigan', 'Wayne County')] ]
[9]:
                                 BIRTHS2010 BIRTHS2011
                                                          BIRTHS2012
                                                                       BIRTHS2013
     STNAME
              CTYNAME
     Michigan Washtenaw County
                                        977
                                                    3826
                                                                 3780
                                                                             3662
              Wayne County
                                       5918
                                                   23819
                                                                23270
                                                                            23377
                                 BIRTHS2014
                                             BIRTHS2015
                                                          POPESTIMATE2010
     STNAME
              CTYNAME
     Michigan Washtenaw County
                                                    3709
                                                                    345563
                                       3683
              Wayne County
                                      23607
                                                   23586
                                                                   1815199
                                 POPESTIMATE2011
                                                   POPESTIMATE2012 POPESTIMATE2013 \
     STNAME
              CTYNAME
     Michigan Washtenaw County
                                           349048
                                                            351213
                                                                              354289
              Wayne County
                                          1801273
                                                           1792514
                                                                             1775713
                                 POPESTIMATE2014
                                                   POPESTIMATE2015
     STNAME
              CTYNAME
     Michigan Washtenaw County
                                           357029
                                                            358880
              Wayne County
                                          1766008
                                                           1759335
```

1.6 Missing Values

We've seen a preview of how Pandas handles missing values using the None type and NumPy NaN values. Missing values are pretty common in data cleaning activities. There are couple of caveats and discussion points which we should address.

One of the handy functions that Pandas has for working with missing values is the filling function, fillna. This function takes a number or parameters, for instance, you could pass in a single value which is called a scalar value to change all of the missing data to one value. This isn't really applicable in this case, but it's a pretty common use case. Next up though is the method parameter. The two common fill values are ffill and bfill. ffill is for forward filling and it updates an na value for a particular cell with the value from the previous row. It's important to note that your data needs

to be sorted in order for this to have the effect you might want. Data that comes from traditional database management systems usually has no order guarantee, just like this data. So be careful.

```
[17]: df=pd.read_csv('log.csv')
df
```

[17]:		time	user	video	plavback	position	paused	volume
	0	1469974424	cheryl	intro.html	1 3	5	False	10.0
	1	1469974454	cheryl	intro.html		6	NaN	NaN
	2	1469974544	cheryl	intro.html		9	NaN	NaN
	3	1469974574	cheryl	intro.html		10	NaN	NaN
	4	1469977514	bob	intro.html		1	NaN	NaN
	5	1469977544	bob	intro.html		1	NaN	NaN
	6	1469977574	bob	intro.html		1	NaN	NaN
	7	1469977604	bob	intro.html		1	NaN	NaN
	8	1469974604	cheryl	intro.html		11	NaN	NaN
	9	1469974694	cheryl	intro.html		14	NaN	NaN
	10	1469974724	cheryl	intro.html		15	NaN	NaN
	11	1469974454	sue	advanced.html		24	NaN	NaN
	12	1469974524	sue	advanced.html		25	NaN	NaN
	13	1469974424	sue	advanced.html		23	False	10.0
	14	1469974554	sue	advanced.html		26	NaN	NaN
	15	1469974624	sue	advanced.html		27	NaN	NaN
	16	1469974654	sue	advanced.html		28	NaN	5.0
	17	1469974724	sue	advanced.html		29	NaN	NaN
	18	1469974484	cheryl	intro.html		7	NaN	NaN
	19	1469974514	cheryl	intro.html		8	NaN	NaN
	20	1469974754	sue	advanced.html		30	NaN	NaN
	21	1469974824	sue	advanced.html		31	NaN	NaN
	22	1469974854	sue	advanced.html		32	NaN	NaN
	23	1469974924	sue	advanced.html		33	NaN	NaN
	24	1469977424	bob	intro.html		1	True	10.0
	25	1469977454	bob	intro.html		1	NaN	NaN
	26	1469977484	bob	intro.html		1	NaN	NaN
	27	1469977634	bob	intro.html		1	NaN	NaN
	28	1469977664	bob	intro.html		1	NaN	NaN
	29	1469974634	cheryl	intro.html		12	NaN	NaN
	30	1469974664	cheryl	intro.html		13	NaN	NaN
	31	1469977694	bob	intro.html		1	NaN	NaN
	32	1469977724	bob	intro.html		1	NaN	NaN

In Pandas we can sort either by index or by values. Here we'll just promote the time stamp to an index then sort on the index.

```
[15]: df=df.set_index('time')
    df=df.sort_index()
    df
```

```
[15]:
                                     video playback position paused volume
                     user
      time
                               intro.html
                                                                 False
                                                                           10.0
      1469974424
                   cheryl
                                                              5
      1469974424
                            advanced.html
                                                             23
                                                                 False
                                                                            10.0
                       sue
                   cheryl
      1469974454
                                intro.html
                                                              6
                                                                    NaN
                                                                            NaN
                            advanced.html
                                                             24
                                                                    NaN
                                                                            NaN
      1469974454
                       sue
      1469974484
                   cheryl
                               intro.html
                                                              7
                                                                    NaN
                                                                            NaN
      1469974514
                   cheryl
                                intro.html
                                                              8
                                                                    NaN
                                                                            NaN
      1469974524
                            advanced.html
                                                             25
                                                                    NaN
                                                                            NaN
                       sue
      1469974544
                   cheryl
                                intro.html
                                                              9
                                                                    NaN
                                                                            NaN
                                                             26
                                                                            NaN
      1469974554
                            advanced.html
                                                                    NaN
                       sue
                                                                    NaN
                                                                            NaN
      1469974574
                   cheryl
                                intro.html
                                                             10
                   cheryl
      1469974604
                                                             11
                                                                    NaN
                                                                            NaN
                                intro.html
                                                             27
                                                                    NaN
                                                                            NaN
      1469974624
                       sue
                            advanced.html
      1469974634
                   cheryl
                                intro.html
                                                             12
                                                                    NaN
                                                                            NaN
      1469974654
                                                             28
                                                                    NaN
                                                                             5.0
                       sue
                            advanced.html
      1469974664
                   cheryl
                               intro.html
                                                             13
                                                                    NaN
                                                                            NaN
                   cheryl
                               intro.html
                                                             14
                                                                    NaN
                                                                            NaN
      1469974694
                   cheryl
                                intro.html
                                                             15
                                                                    NaN
                                                                            NaN
      1469974724
                                                             29
      1469974724
                            advanced.html
                                                                    NaN
                                                                            NaN
                       sue
      1469974754
                       sue
                            advanced.html
                                                             30
                                                                    NaN
                                                                            NaN
                            advanced.html
                                                             31
                                                                    NaN
                                                                            NaN
      1469974824
                       sue
      1469974854
                       sue
                            advanced.html
                                                             32
                                                                    NaN
                                                                            NaN
                                                                    NaN
      1469974924
                       sue
                            advanced.html
                                                             33
                                                                            NaN
      1469977424
                                intro.html
                                                              1
                                                                   True
                                                                            10.0
                       bob
                                                                    NaN
                                                                            NaN
      1469977454
                       bob
                                intro.html
                                                              1
                                                                    NaN
                                                                            NaN
      1469977484
                       bob
                               intro.html
                                                              1
      1469977514
                       bob
                               intro.html
                                                              1
                                                                    NaN
                                                                            NaN
                                                              1
                                                                    NaN
                                                                            NaN
      1469977544
                       bob
                               intro.html
      1469977574
                       bob
                               intro.html
                                                                    NaN
                                                                             NaN
      1469977604
                       bob
                               intro.html
                                                                    NaN
                                                                             NaN
      1469977634
                       bob
                               intro.html
                                                              1
                                                                    NaN
                                                                            NaN
      1469977664
                       bob
                               intro.html
                                                              1
                                                                    NaN
                                                                            NaN
      1469977694
                               intro.html
                                                              1
                                                                    NaN
                                                                            NaN
                       bob
      1469977724
                               intro.html
                       bob
                                                              1
                                                                    NaN
                                                                            NaN
[18]: #use multi-level indexing with time and user
      df=df.set_index(['time','user'])
      df=df.sort_index()
「18]:
                                           playback position paused volume
      time
                  user
                                                               False
      1469974424 cheryl
                              intro.html
                                                                          10.0
                                                             5
                           advanced.html
                                                            23
                                                               False
                                                                          10.0
                  sue
                                                                  NaN
      1469974454 cheryl
                              intro.html
                                                             6
                                                                           NaN
                  sue
                           advanced.html
                                                            24
                                                                  NaN
                                                                           NaN
```

1469974484	cheryl	intro.html	7	NaN	NaN
1469974514	cheryl	intro.html	8	NaN	NaN
1469974524	sue	advanced.html	25	NaN	NaN
1469974544	cheryl	intro.html	9	NaN	NaN
1469974554	sue	advanced.html	26	NaN	NaN
1469974574	cheryl	intro.html	10	NaN	NaN
1469974604	cheryl	intro.html	11	NaN	NaN
1469974624	sue	advanced.html	27	NaN	NaN
1469974634	cheryl	intro.html	12	NaN	NaN
1469974654	sue	advanced.html	28	NaN	5.0
1469974664	cheryl	intro.html	13	NaN	NaN
1469974694	cheryl	intro.html	14	NaN	NaN
1469974724	cheryl	intro.html	15	NaN	NaN
	sue	advanced.html	29	NaN	NaN
1469974754	sue	advanced.html	30	NaN	NaN
1469974824	sue	advanced.html	31	NaN	NaN
1469974854	sue	advanced.html	32	NaN	NaN
1469974924	sue	advanced.html	33	NaN	NaN
1469977424	bob	intro.html	1	True	10.0
1469977454	bob	intro.html	1	NaN	NaN
1469977484	bob	intro.html	1	NaN	NaN
1469977514	bob	intro.html	1	NaN	NaN
1469977544	bob	intro.html	1	NaN	NaN
1469977574	bob	intro.html	1	NaN	NaN
1469977604	bob	intro.html	1	NaN	NaN
1469977634	bob	intro.html	1	NaN	NaN
1469977664	bob	intro.html	1	NaN	NaN
1469977694	bob	intro.html	1	NaN	NaN
1469977724	bob	intro.html	1	NaN	NaN

It's sometimes useful to use forward filling, sometimes backwards filling, and sometimes useful to just use a single number. More recently, the Pandas team introduced a method of filling missing values with a series which is the same length as your DataFrame. This makes it easy to derive values which are missing if you have the underlying to do so. For instance, if you're dealing with receipts and you have a column for final price and a column for discount but are missing information from the original price column, you can fill this automatically using fillna.

[]: