indexing

January 25, 2021

0.1 Indexing in Pandas

Pandas is a Python module that lets us work in a table-like structure called DataFrames. Pandas is the premier module for working with data in Python, and shares may structural similarites to Excel.

[13]:
```
import pandas as pd
```

[14]:
```
sales = pd.read_csv('supermarket_sales.csv')
sales.head()
```

[14]:

	Invoice ID	Branch	City	Customer type	Gender	\
0	750-67-8428	A	Yangon	Member	Female	
1	226-31-3081	C	Naypyitaw	Normal	Female	
2	631-41-3108	A	Yangon	Normal	Male	
3	123-19-1176	A	Yangon	Member	Male	
4	373-73-7910	A	Yangon	Normal	Male	

	Product line	Unit price	Quantity	Tax 5%	Total	Date	\
0	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	
1	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	
2	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	
3	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	
4	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	

	Time	Payment	cogs	gross margin percentage	gross income	Rating
0	13:08	Ewallet	522.83	4.761905	26.1415	9.1
1	10:29	Cash	76.40	4.761905	3.8200	9.6
2	13:23	Credit card	324.31	4.761905	16.2155	7.4
3	20:33	Ewallet	465.76	4.761905	23.2880	8.4
4	10:37	Ewallet	604.17	4.761905	30.2085	5.3

Let's look at the index of the dataframe.

[15]:
```
sales.index
```

[15]:
```
RangeIndex(start=0, stop=1000, step=1)
```

As you can see, the index ranges from 0 to 1000. Let's set the index to something more useful.

Notice that we have to specify the 'inplace' = True keyword to ensure that the change is made in place.

[16]:
```
sales.set_index('Invoice ID', inplace = True)
```

[17]:
```
sales.index
```

[17]:
```
Index(['750-67-8428', '226-31-3081', '631-41-3108', '123-19-1176',
       '373-73-7910', '699-14-3026', '355-53-5943', '315-22-5665',
       '665-32-9167', '692-92-5582',
       …
       '886-18-2897', '602-16-6955', '745-74-0715', '690-01-6631',
       '652-49-6720', '233-67-5758', '303-96-2227', '727-02-1313',
       '347-56-2442', '849-09-3807'],
      dtype='object', name='Invoice ID', length=1000)
```

Now the index shows invoices.

0.1.1 Using loc and iloc.

We can access specific columns through the loc and iloc accesors. Loc is used when selecting indexes by name and iloc when accessing indexes by number.

[18]:
```
## Set index to date

sales.set_index('Date', inplace = True)
```

[20]:
```
## Get all sales on Jan. 5, 2019

sales.loc['1/5/2019']
```

[20]:

Date	Branch	City	Customer type	Gender	Product line	\
1/5/2019	A	Yangon	Member	Female	Health and beauty	
1/5/2019	C	Naypyitaw	Normal	Male	Fashion accessories	
1/5/2019	A	Yangon	Member	Male	Home and lifestyle	
1/5/2019	A	Yangon	Member	Male	Home and lifestyle	
1/5/2019	B	Mandalay	Member	Female	Home and lifestyle	
1/5/2019	C	Naypyitaw	Member	Female	Fashion accessories	
1/5/2019	A	Yangon	Normal	Female	Home and lifestyle	
1/5/2019	B	Mandalay	Member	Female	Fashion accessories	
1/5/2019	B	Mandalay	Normal	Female	Food and beverages	
1/5/2019	C	Naypyitaw	Normal	Female	Fashion accessories	
1/5/2019	C	Naypyitaw	Normal	Female	Health and beauty	
1/5/2019	A	Yangon	Normal	Female	Electronic accessories	

Unit price	Quantity	Tax 5%	Total	Time	Payment	cogs	\

```
Date
1/5/2019       74.69         7  26.1415  548.9715  13:08      Ewallet  522.83
1/5/2019       27.38         6   8.2140  172.4940  20:54  Credit card  164.28
1/5/2019       62.65         4  12.5300  263.1300  11:25         Cash  250.60
1/5/2019       70.74         4  14.1480  297.1080  16:05  Credit card  282.96
1/5/2019       35.38         9  15.9210  334.3410  19:50  Credit card  318.42
1/5/2019       31.90         1   1.5950   33.4950  12:40      Ewallet   31.90
1/5/2019       42.91         5  10.7275  225.2775  17:29      Ewallet  214.55
1/5/2019       73.96         1   3.6980   77.6580  11:32  Credit card   73.96
1/5/2019       71.20         1   3.5600   74.7600  20:40  Credit card   71.20
1/5/2019       76.06         3  11.4090  239.5890  20:30  Credit card  228.18
1/5/2019       78.89         7  27.6115  579.8415  19:48      Ewallet  552.23
1/5/2019       93.88         7  32.8580  690.0180  11:51  Credit card  657.16

           gross margin percentage  gross income  Rating
Date
1/5/2019                 4.761905       26.1415     9.1
1/5/2019                 4.761905        8.2140     7.9
1/5/2019                 4.761905       12.5300     4.2
1/5/2019                 4.761905       14.1480     4.4
1/5/2019                 4.761905       15.9210     9.6
1/5/2019                 4.761905        1.5950     9.1
1/5/2019                 4.761905       10.7275     6.1
1/5/2019                 4.761905        3.6980     5.0
1/5/2019                 4.761905        3.5600     9.2
1/5/2019                 4.761905       11.4090     9.8
1/5/2019                 4.761905       27.6115     7.5
1/5/2019                 4.761905       32.8580     7.3
```

[21]:
```
## We can also get only a specific column

sales.loc['1/5/2019', 'Gender']
```

[21]:
```
Date
1/5/2019    Female
1/5/2019      Male
1/5/2019      Male
1/5/2019      Male
1/5/2019    Female
1/5/2019    Female
1/5/2019    Female
1/5/2019    Female
1/5/2019    Female
1/5/2019    Female
1/5/2019    Female
1/5/2019    Female
Name: Gender, dtype: object
```

What about iloc? Notice that won't work with the date.

[22]:
```
sales.iloc['1/5/2019']
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-22-20860a74f6c8> in <module>
----> 1 sales.iloc['1/5/2019']


/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py in␣
↪__getitem__(self, key)
   1766
   1767             maybe_callable = com.apply_if_callable(key, self.obj)
-> 1768             return self._getitem_axis(maybe_callable, axis=axis)
   1769
   1770     def _is_scalar_access(self, key: Tuple):


/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py in␣
↪_getitem_axis(self, key, axis)
   2133             key = item_from_zerodim(key)
   2134             if not is_integer(key):
-> 2135                 raise TypeError("Cannot index by location index with␣
↪a non-integer key")
   2136
   2137             # validate the location


TypeError: Cannot index by location index with a non-integer key
```

iloc uses integer indexing. Regardless of what the actual index is, iloc will get the associated rows/columns from integers.

[23]:
```
sales.iloc[3]
```

[23]:
```
Branch                             A
City                          Yangon
Customer type                 Member
Gender                          Male
Product line         Health and beauty
Unit price                     58.22
```

```
Quantity                                8
Tax 5%                             23.288
Total                             489.048
Time                                20:33
Payment                           Ewallet
cogs                               465.76
gross margin percentage            4.7619
gross income                       23.288
Rating                                8.4
Name: 1/27/2019, dtype: object
```

Let's try loc again, with an integer this time.

[24]:
```
sales.loc[3]
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-24-675e9263f29c> in <module>
----> 1 sales.loc[3]


/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py in␣
↪__getitem__(self, key)
   1766
   1767             maybe_callable = com.apply_if_callable(key, self.obj)
-> 1768             return self._getitem_axis(maybe_callable, axis=axis)
   1769
   1770     def _is_scalar_access(self, key: Tuple):


/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py in␣
↪_getitem_axis(self, key, axis)
   1962
   1963         # fall thru to straight lookup
-> 1964         self._validate_key(key, axis)
   1965         return self._get_label(key, axis=axis)
   1966


/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py in␣
↪_validate_key(self, key, axis)
   1829
```

```
   1830         if not is_list_like_indexer(key):
-> 1831             self._convert_scalar_indexer(key, axis)
   1832
   1833     def _is_scalar_access(self, key: Tuple) -> bool:


/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py in␣
↪_convert_scalar_indexer(self, key, axis)
    739         ax = self.obj._get_axis(min(axis, self.ndim - 1))
    740         # a scalar
--> 741         return ax._convert_scalar_indexer(key, kind=self.name)
    742
    743     def _convert_slice_indexer(self, key: slice, axis: int):


/opt/conda/lib/python3.7/site-packages/pandas/core/indexes/base.py in␣
↪_convert_scalar_indexer(self, key, kind)
   2885             elif kind in ["loc"] and is_integer(key):
   2886                 if not self.holds_integer():
-> 2887                     self._invalid_indexer("label", key)
   2888
   2889         return key


/opt/conda/lib/python3.7/site-packages/pandas/core/indexes/base.py in␣
↪_invalid_indexer(self, form, key)
   3074         """
   3075         raise TypeError(
-> 3076             f"cannot do {form} indexing on {type(self)} with these "
   3077             f"indexers [{key}] of {type(key)}"
   3078         )


TypeError: cannot do label indexing on <class 'pandas.core.indexes.base.
↪Index'> with these indexers [3] of <class 'int'>
```

Again, this won't work. The general rule is loc for specific index names and iloc for integer values. If we reset the index, then both loc and iloc will agree with each other.

[26]:
```
sales.reset_index(inplace = True)
sales.loc[3]
```

[26]:
```
index                           3
Date                     1/27/2019
Branch                           A
City                        Yangon
Customer type               Member
```

```
Gender                                 Male
Product line                Health and beauty
Unit price                             58.22
Quantity                                   8
Tax 5%                                23.288
Total                                489.048
Time                                   20:33
Payment                              Ewallet
cogs                                  465.76
gross margin percentage               4.7619
gross income                          23.288
Rating                                   8.4
Name: 3, dtype: object
```

Note how this looks like a 1D array (called a series in Pandas). Accessing this array is similar to accessing values in a dictionary.

[28]:
```
sales.iloc[3]['City']
```

[28]:
```
'Yangon'
```

We can also get all rows of a certain type even if the column isn't the index. Let's get all orders from the city 'Yangon'.

[30]:
```
sales.City == 'Yangon'
```

[30]:
```
0       True
1      False
2       True
3       True
4       True
       …
995    False
996    False
997     True
998     True
999     True
Name: City, Length: 1000, dtype: bool
```

Notice how this only gives us a series of boolean (True or False) values. That's not what we want. Let's use this to index into our original dataframe.

[31]:
```
sales.loc[sales.City == 'Yangon']
```

[31]:
```
     index       Date Branch    City Customer type  Gender  \
0        0   1/5/2019      A  Yangon        Member  Female
2        2   3/3/2019      A  Yangon        Normal    Male
3        3  1/27/2019      A  Yangon        Member    Male
```

```
4        4   2/8/2019      A  Yangon        Normal    Male
6        6  2/25/2019      A  Yangon        Member  Female
..     …        …    …     …           …      …
990    990  3/22/2019      A  Yangon        Normal  Female
992    992  3/10/2019      A  Yangon        Normal    Male
997    997   2/9/2019      A  Yangon        Member    Male
998    998  2/22/2019      A  Yangon        Normal    Male
999    999  2/18/2019      A  Yangon        Member  Female

               Product line  Unit price  Quantity   Tax 5%     Total   Time  \
0         Health and beauty       74.69         7  26.1415  548.9715  13:08
2        Home and lifestyle       46.33         7  16.2155  340.5255  13:23
3         Health and beauty       58.22         8  23.2880  489.0480  20:33
4         Sports and travel       86.31         7  30.2085  634.3785  10:37
6    Electronic accessories       68.84         6  20.6520  433.6920  14:36
..                      …          …       …      …       …    …
990      Food and beverages       56.56         5  14.1400  296.9400  19:06
992  Electronic accessories       58.03         2   5.8030  121.8630  20:46
997      Food and beverages       31.84         1   1.5920   33.4320  13:22
998      Home and lifestyle       65.82         1   3.2910   69.1110  15:33
999     Fashion accessories       88.34         7  30.9190  649.2990  13:28

         Payment    cogs  gross margin percentage  gross income  Rating
0        Ewallet  522.83                 4.761905       26.1415     9.1
2    Credit card  324.31                 4.761905       16.2155     7.4
3        Ewallet  465.76                 4.761905       23.2880     8.4
4        Ewallet  604.17                 4.761905       30.2085     5.3
6        Ewallet  413.04                 4.761905       20.6520     5.8
..           …     …                       …           …     …
990  Credit card  282.80                 4.761905       14.1400     4.5
992      Ewallet  116.06                 4.761905        5.8030     8.8
997         Cash   31.84                 4.761905        1.5920     7.7
998         Cash   65.82                 4.761905        3.2910     4.1
999         Cash  618.38                 4.761905       30.9190     6.6

[340 rows x 17 columns]
```

Now, let's find the sum of all taxes collected on credit card purchases of fashion accesorries. Assume a tax rate of 10%.

[35]:
```
### Get all rows that purchased fashion accesories. Store it into sales_fashion.

sales_fashion = sales[sales['Product line'] == 'Fashion accessories']
```

[37]:
```
### Get all credit card purcahses from sales_fashion.

sales_card_fashion = sales_fashion[sales_fashion['Payment'] == 'Credit card']
```

[39]:
```
sales_card_fashion.head()
```

[39]:
```
    index       Date Branch       City Customer type  Gender  \
27     27  3/10/2019      A     Yangon        Normal  Female
30     30  2/25/2019      B   Mandalay        Normal    Male
53     53  1/25/2019      C  Naypyitaw        Member    Male
76     76   1/9/2019      C  Naypyitaw        Member    Male
77     77  1/12/2019      A     Yangon        Member  Female

           Product line  Unit price  Quantity   Tax 5%     Total   Time  \
27  Fashion accessories       87.67         2   8.7670  184.1070  12:17
30  Fashion accessories       94.13         5  23.5325  494.1825  19:39
53  Fashion accessories       15.43         1   0.7715   16.2015  15:46
76  Fashion accessories       49.04         9  22.0680  463.4280  14:20
77  Fashion accessories       20.01         9   9.0045  189.0945  15:48

        Payment    cogs  gross margin percentage  gross income  Rating
27  Credit card  175.34                 4.761905        8.7670     7.7
30  Credit card  470.65                 4.761905       23.5325     4.8
53  Credit card   15.43                 4.761905        0.7715     6.1
76  Credit card  441.36                 4.761905       22.0680     8.6
77  Credit card  180.09                 4.761905        9.0045     5.7
```

[40]:
```
sales_card_fashion['Tax 10%'] = sales_card_fashion['Total'] * 0.1
```

[41]:
```
sales_card_fashion['Tax 10%'].describe()
```

[41]:
```
count     56.000000
mean      30.955481
std       28.067689
min        1.269450
25%       10.880100
50%       20.107500
75%       41.491538
max      104.265000
Name: Tax 10%, dtype: float64
```

[ ]:
```

```