

## How to select dataframe subsets from multivariate data

```
In [1]: import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100) # Show all columns when looking at dataframe
```

```
In [4]: # Download NHANES 2015-2016 data
df = pd.read_csv("nhanes_2015_2016.csv")
```

```
In [8]: df.head()
```

```
Out[8]:
```

	SEQN	ALQ101	ALQ110	ALQ130	SMQ020	RIAGENDR	RIDAGEYR	RIDRETH1	DMDCTZN	DMDEDUC2	DMDMARTL	DMDHHSIZ	WTINT2YR
0	83732	1.0	NaN	1.0	1	1	62	3	1.0	5.0	1.0	2	134.6
1	83733	1.0	NaN	6.0	1	1	53	3	2.0	3.0	3.0	1	243.2
2	83734	1.0	NaN	NaN	1	1	78	3	1.0	3.0	1.0	2	124.6
3	83735	2.0	1.0	1.0	2	2	56	3	1.0	5.0	6.0	1	102.7
4	83736	2.0	1.0	1.0	2	2	42	4	1.0	4.0	3.0	5	176.2

## Keep only body measures columns, so only columns with "BMX" in the name

```
In [9]: # get columns names
col_names = df.columns
col_names
```

```
Out[9]: Index(['SEQN', 'ALQ101', 'ALQ110', 'ALQ130', 'SMQ020', 'RIAGENDR', 'RIDAGEYR',
'RIDRETH1', 'DMDCTZN', 'DMDEDUC2', 'DMDMARTL', 'DMDHHSIZ', 'WTINT2YR',
'SDMVPSU', 'SDMVSTRA', 'INDFMPPIR', 'BPXSY1', 'BPXDIL1', 'BPXSY2',
'BPXDIL2', 'BMXWT', 'BMXHT', 'BMXBMI', 'BMXLEG', 'BMXARML', 'BMXARMC',
'BMXWAIST', 'HIQ2I0'],
dtype='object')
```

```
In [12]: # One way to get the column names we want to keep is simply by copying from the above output and storing in a
list
keep = ['BMXWT', 'BMXHT', 'BMXBMI', 'BMXLEG', 'BMXARML', 'BMXARMC',
'BMXWAIST']
print(keep)

['BMXWT', 'BMXHT', 'BMXBMI', 'BMXLEG', 'BMXARML', 'BMXARMC', 'BMXWAIST']
```

```
In [17]: # Another way to get only column names that include 'BMX' is with list comprehension
# [keep x for x in list if condition met]
[column for column in col_names if 'BMX' in column]
```

```
Out[17]: ['BMXWT', 'BMXHT', 'BMXBMI', 'BMXLEG', 'BMXARML', 'BMXARMC', 'BMXWAIST']
```

```
In [19]: keep = [column for column in col_names if 'BMX' in column]
```

```
In [20]: # use [] notation to keep columns
df_BMX = df[keep]
```

```
In [21]: df_BMX.head()
```

```
Out[21]:
```

	BMXWT	BMXHT	BMXBMI	BMXLEG	BMXARML	BMXARMC	BMXWAIST
0	94.8	184.5	27.8	43.3	43.6	35.9	101.1
1	90.4	171.4	30.8	38.0	40.0	33.2	107.9
2	83.4	170.1	28.8	35.6	37.0	31.0	116.5
3	109.8	160.9	42.4	38.5	37.7	38.3	110.1
4	55.2	164.9	20.3	37.4	36.0	27.2	80.4

There are two methods for selecting by row and column.

## link for pandas cheat sheets

- `df.loc[row labels or bool, col labels or bool]`
- `df.iloc[row int or bool, col int or bool]`

**From pandas docs (<https://pandas.pydata.org/pandas-docs/stable/indexing.html>):**

- [] column indexing
- .loc is primarily label based, but may also be used with a boolean array.
- .iloc is primarily integer position based (from 0 to length-1 of the axis), but may also be used with a boolean array.

In [22]: `df.loc[:, keep].head()`

Out[22]:

	BMXWT	BMXHT	BMXBMI	BMXLEG	BMXARML	BMXARMC	BMXWAIST
0	94.8	184.5	27.8	43.3	43.6	35.9	101.1
1	90.4	171.4	30.8	38.0	40.0	33.2	107.9
2	83.4	170.1	28.8	35.6	37.0	31.0	116.5
3	109.8	160.9	42.4	38.5	37.7	38.3	110.1
4	55.2	164.9	20.3	37.4	36.0	27.2	80.4

In [24]: `index_bool = np.isin(df.columns, keep)`

In [25]: `index_bool`

Out[25]: `array([False, False, False, False, False, False, False, False, False, False, False, False, True, True, True, True, True, True, True, False])`

In [26]: `df.iloc[:, index_bool].head() # Indexing with boolean list`

Out[26]:

	BMXWT	BMXHT	BMXBMI	BMXLEG	BMXARML	BMXARMC	BMXWAIST
0	94.8	184.5	27.8	43.3	43.6	35.9	101.1
1	90.4	171.4	30.8	38.0	40.0	33.2	107.9
2	83.4	170.1	28.8	35.6	37.0	31.0	116.5
3	109.8	160.9	42.4	38.5	37.7	38.3	110.1
4	55.2	164.9	20.3	37.4	36.0	27.2	80.4

**Selection by conditions**

In [37]: `# Lets only look at rows who 'BMXWAIST' is larger than the median  
waist_median = pd.Series.median(df_BMX['BMXWAIST']) # get the median of 'BMXWAIST'  
  
#or by another method to find median  
print(df.BMXWAIST.median())  
98.3`

In [38]: `waist_median`

Out[38]: `98.3`

In [39]: `df_BMX[df_BMX['BMXWAIST'] > waist_median].head()`

Out[39]:

	BMXWT	BMXHT	BMXBMI	BMXLEG	BMXARML	BMXARMC	BMXWAIST
0	94.8	184.5	27.8	43.3	43.6	35.9	101.1
1	90.4	171.4	30.8	38.0	40.0	33.2	107.9
2	83.4	170.1	28.8	35.6	37.0	31.0	116.5
3	109.8	160.9	42.4	38.5	37.7	38.3	110.1
9	108.3	179.4	33.6	46.0	44.1	38.5	116.0

```
In [41]: # Lets add another condition, that 'BMXLEG' must be less than 32
condition1 = df_BMX['BMXWAIST'] > waist_median
condition2 = df_BMX['BMXLEG'] < 32
df_BMX[condition1 & condition2].head() # Using [] method
# Note: can't use 'and' instead of '&'
```

Out[41]:

	BMXWT	BMXHT	BMXBMI	BMXLEG	BMXARML	BMXARMC	BMXWAIST
15	80.5	150.8	35.4	31.6	32.7	33.7	113.5
27	75.6	145.2	35.9	31.0	33.1	36.0	108.0
39	63.7	147.9	29.1	26.0	34.0	31.5	110.0
52	105.9	157.7	42.6	29.2	35.0	40.7	129.1
55	77.5	148.3	35.2	30.5	34.0	34.4	107.6

```
In [42]: df_BMX.loc[condition1 & condition2, :].head() # Using df.loc[] method
# note that the conditiona are describing the rows to keep
```

Out[42]:

	BMXWT	BMXHT	BMXBMI	BMXLEG	BMXARML	BMXARMC	BMXWAIST
15	80.5	150.8	35.4	31.6	32.7	33.7	113.5
27	75.6	145.2	35.9	31.0	33.1	36.0	108.0
39	63.7	147.9	29.1	26.0	34.0	31.5	110.0
52	105.9	157.7	42.6	29.2	35.0	40.7	129.1
55	77.5	148.3	35.2	30.5	34.0	34.4	107.6

```
In [45]: # Lets make a small dataframe and give it a new index so can more clearly see the differences between .loc and
         .iloc
tmp = df_BMX.loc[condition1 & condition2, :].head()
tmp.index = ['a', 'b', 'c', 'd', 'e'] # If you use different years than 2015-2016, this may give an error. Why?
tmp
```

Out[45]:

	BMXWT	BMXHT	BMXBMI	BMXLEG	BMXARML	BMXARMC	BMXWAIST
a	80.5	150.8	35.4	31.6	32.7	33.7	113.5
b	75.6	145.2	35.9	31.0	33.1	36.0	108.0
c	63.7	147.9	29.1	26.0	34.0	31.5	110.0
d	105.9	157.7	42.6	29.2	35.0	40.7	129.1
e	77.5	148.3	35.2	30.5	34.0	34.4	107.6

```
In [46]: tmp.loc[['a', 'b'], 'BMXLEG']
```

```
Out[46]: a    31.6
         b    31.0
         Name: BMXLEG, dtype: float64
```

```
In [47]: tmp.iloc[[0, 1], 3]
```

```
Out[47]: a    31.6
         b    31.0
         Name: BMXLEG, dtype: float64
```

### Common errors and how to read them

```
In [50]: tmp[:, 'BMXBMI']
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-50-83067c5cae7c> in <module>()
----> 1 tmp[:, 'BMXBMI']

/opt/conda/lib/python3.6/site-packages/pandas/core/frame.py in __getitem__(self, key)
   2925         if self.columns.nlevels > 1:
   2926             return self._getitem_multilevel(key)
-> 2927         indexer = self.columns.get_loc(key)
   2928         if is_integer(indexer):
   2929             indexer = [indexer]

/opt/conda/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2654         'backfill or nearest lookups')
   2655         try:
-> 2656             return self._engine.get_loc(key)
   2657         except KeyError:
   2658             return self._engine.get_loc(self._maybe_cast_indexer(key))

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

TypeError: '(slice(None, None, None), 'BMXBMI')' is an invalid key
```

### Problem

The above gives: `TypeError: unhashable type: 'slice'`

The `[]` method uses hashes to identify the columns to keep, and each column has an associated hash. A 'slice' (a subset of rows and columns) does not have an associated hash, thus causing this `TypeError`.

```
In [52]: tmp.loc[:, 'BMXBMI']
```

```
Out[52]: a    35.4
         b    35.9
         c    29.1
         d    42.6
         e    35.2
         Name: BMXBMI, dtype: float64
```

```
In [53]: tmp.loc[:, 'BMXBMI'].values
```

```
Out[53]: array([35.4, 35.9, 29.1, 42.6, 35.2])
```

```
In [56]: tmp.iloc[:, 'BMXBMI']
```

```
-----
ValueError                                Traceback (most recent call last)
/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py in _has_valid_tuple(self, key)
    222         try:
--> 223             self._validate_key(k, i)
    224         except ValueError:

/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py in _validate_key(self, key, axis)
    2083         raise ValueError("Can only index by location with "
-> 2084                          "a [{types}]".format(types=self._valid_types))
    2085

ValueError: Can only index by location with a [integer, integer slice (START point is INCLUDED, END point is E
XCLUDED), listlike of integers, boolean array]

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
<ipython-input-56-9fa39d4097e1> in <module>()
----> 1 tmp.iloc[:, 'BMXBMI']

/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py in __getitem__(self, key)
    1492         except (KeyError, IndexError, AttributeError):
    1493             pass
-> 1494         return self._getitem_tuple(key)
    1495     else:
    1496         # we by definition only have the 0th axis

/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py in _getitem_tuple(self, tup)
    2141     def _getitem_tuple(self, tup):
    2142
-> 2143         self._has_valid_tuple(tup)
    2144         try:
    2145             return self._getitem_lowerdim(tup)

/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py in _has_valid_tuple(self, key)
    225         raise ValueError("Location based indexing can only have "
    226                          "[{types}] types"
-> 227                          .format(types=self._valid_types))
    228
    229     def _is_nested_tuple_indexer(self, tup):

ValueError: Location based indexing can only have [integer, integer slice (START point is INCLUDED, END point
is EXCLUDED), listlike of integers, boolean array] types
```

### Problem

The above gives: ValueError: Location based indexing can only have [integer, integer slice (START point is INCLUDED, END point is EXCLUDED), listlike of integers, boolean array] types

'BMXBMI' is not an integer that is less than or equal number of columns -1, or a list of boolean values, so it is the wrong value type.

```
In [57]: tmp.iloc[:, 2]
```

```
Out[57]: a    35.4
         b    35.9
         c    29.1
         d    42.6
         e    35.2
         Name: BMXBMI, dtype: float64
```

```
In [58]: tmp.loc[:, 2]
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-58-a70ce725ddad> in <module>()
----> 1 tmp.loc[:, 2]

/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py in __getitem__(self, key)
   1492     except (KeyError, IndexError, AttributeError):
   1493         pass
--> 1494     return self._getitem_tuple(key)
   1495     else:
   1496         # we by definition only have the 0th axis

/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py in _getitem_tuple(self, tup)
   866     def _getitem_tuple(self, tup):
   867         try:
--> 868             return self._getitem_lowerdim(tup)
   869         except IndexingError:
   870             pass

/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py in _getitem_lowerdim(self, tup)
   986     for i, key in enumerate(tup):
   987         if is_label_like(key) or isinstance(key, tuple):
--> 988             section = self._getitem_axis(key, axis=i)
   989
   990         # we have yielded a scalar ?

/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py in _getitem_axis(self, key, axis)
   1910
   1911     # fall thru to straight lookup
--> 1912     self._validate_key(key, axis)
   1913     return self._get_label(key, axis=axis)
   1914

/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py in _validate_key(self, key, axis)
   1797
   1798     if not is_list_like_indexer(key):
--> 1799         self._convert_scalar_indexer(key, axis)
   1800
   1801     def _is_scalar_access(self, key):

/opt/conda/lib/python3.6/site-packages/pandas/core/indexing.py in _convert_scalar_indexer(self, key, axis)
   260     ax = self.obj._get_axis(min(axis, self.ndim - 1))
   261     # a scalar
--> 262     return ax._convert_scalar_indexer(key, kind=self.name)
   263
   264     def _convert_slice_indexer(self, key, axis):

/opt/conda/lib/python3.6/site-packages/pandas/core/indexes/base.py in _convert_scalar_indexer(self, key, kind)
   2878     elif kind in ['loc'] and is_integer(key):
   2879         if not self.holds_integer():
--> 2880             return self._invalid_indexer('label', key)
   2881
   2882     return key

/opt/conda/lib/python3.6/site-packages/pandas/core/indexes/base.py in _invalid_indexer(self, form, key)
   3064     "indexers [{key}] of {kind}".format(
   3065         form=form, klass=type(self), key=key,
--> 3066         kind=type(key))
   3067
   3068     # -----

TypeError: cannot do label indexing on <class 'pandas.core.indexes.base.Index'> with these indexers [2] of <cl
ass 'int'>
```

### Problem

The above code gives: `TypeError: cannot do label indexing on <class 'pandas.core.indexes.base.Index'> with these indexers [2] of <class 'int'>`

2 is not one of the labels (i.e. column names) in the dataframe

```
In [61]: # Here is another example of using a boolean list for indexing columns
tmp.loc[:, [False, True, True] + [False]*4]
```

```
Out[61]:
```

	BMXHT	BMXBMI
a	150.8	35.4
b	145.2	35.9
c	147.9	29.1
d	157.7	42.6
e	148.3	35.2

In [62]: `tmp.iloc[:, [1,2]]`

Out[62]:

	BMXHT	BMXBMI
a	150.8	35.4
b	145.2	35.9
c	147.9	29.1
d	157.7	42.6
e	148.3	35.2

In [85]: `# We can use the .loc and .iloc methods to change values within the dataframe  
tmp.iloc[0:3,2] = [0]*3  
tmp.iloc[:,2]  
#changes once made in date cannot be undone that's why output seems different`

Out[85]:

```
a    0.0
b    0.0
c    0.0
d    0.0
e    1.0
Name: BMXBMI, dtype: float64
```

In [86]: `tmp.loc['a':'c', 'BMXBMI'] = [1]*3  
tmp.loc[:, 'BMXBMI']  
#changes once made in date cannot be undone that's why output seems different`

Out[86]:

```
a    1.0
b    1.0
c    1.0
d    0.0
e    1.0
Name: BMXBMI, dtype: float64
```

In [88]: `# We can use the [] method when changing all the values of a column  
tmp['BMXBMI'] = range(0, 5)  
tmp`

Out[88]:

	BMXWT	BMXHT	BMXBMI	BMXLEG	BMXARML	BMXARMC	BMXWAIST
a	80.5	150.8	0	31.6	32.7	33.7	113.5
b	75.6	145.2	1	31.0	33.1	36.0	108.0
c	63.7	147.9	2	26.0	34.0	31.5	110.0
d	105.9	157.7	3	29.2	35.0	40.7	129.1
e	77.5	148.3	4	30.5	34.0	34.4	107.6

In [89]: `# We will get a warning when using the [] method with conditions to set new values in our dataframe  
tmp[tmp.BMXBMI > 2]['BMXBMI'] = [10]*2 # Setting new values to a copy of tmp, but not tmp itself  
tmp  
# You can see that the above code did not change our dataframe 'tmp'. This`

Out[89]:

	BMXWT	BMXHT	BMXBMI	BMXLEG	BMXARML	BMXARMC	BMXWAIST
a	80.5	150.8	0	31.6	32.7	33.7	113.5
b	75.6	145.2	1	31.0	33.1	36.0	108.0
c	63.7	147.9	2	26.0	34.0	31.5	110.0
d	105.9	157.7	3	29.2	35.0	40.7	129.1
e	77.5	148.3	4	30.5	34.0	34.4	107.6

In [90]: `# The correct way to do the above is with .loc or .iloc  
tmp.loc[tmp.BMXBMI > 2, 'BMXBMI'] = [10]*2  
tmp # Now contains the changes`

Out[90]:

	BMXWT	BMXHT	BMXBMI	BMXLEG	BMXARML	BMXARMC	BMXWAIST
a	80.5	150.8	0	31.6	32.7	33.7	113.5
b	75.6	145.2	1	31.0	33.1	36.0	108.0
c	63.7	147.9	2	26.0	34.0	31.5	110.0
d	105.9	157.7	10	29.2	35.0	40.7	129.1
e	77.5	148.3	10	30.5	34.0	34.4	107.6