# Caveats and Gotchas

Pandas follows the numpy convention of raising an error when you try to convert something to a **bool**. This happens in an **if** or **when** using the Boolean operations, and, **or**, or **not**. It is not clear what the result should be. Should it be True because it is not zerolength? False because there are False values? It is unclear, so instead, Pandas raises a **ValueError** −

Live Demo

```
import pandas as pd

if pd.Series([False, True, False]):
   print 'I am True'
```

Its **output** is as follows −

```
ValueError: The truth value of a Series is ambiguous.
Use a.empty, a.bool() a.item(),a.any() or a.all().
```

In **if** condition, it is unclear what to do with it. The error is suggestive of whether to use a **None** or **any of those**.

Live Demo

```
import pandas as pd

if pd.Series([False, True, False]).any():
   print("I am any")
```

Its **output** is as follows −

```
I am any
```

To evaluate single-element pandas objects in a Boolean context, use the method **.bool()** −

Live Demo

```
import pandas as pd

print pd.Series([True]).bool()
```

Its **output** is as follows −

```
True
```

## Bitwise Boolean

Bitwise Boolean operators like == and **!=** will return a Boolean series, which is almost always what is required anyways.

Live Demo

```
import pandas as pd
```

```
s = pd.Series(range(5))
print s==4
```

Its **output** is as follows −

```
0 False
1 False
2 False
3 False
4 True
dtype: bool
```

## isin Operation

This returns a Boolean series showing whether each element in the Series is exactly contained in the passed sequence of values.

Live Demo

```
import pandas as pd

s = pd.Series(list('abc'))
s = s.isin(['a', 'c', 'e'])
print s
```

Its **output** is as follows −

```
0 True
1 False
2 True
dtype: bool
```

## Reindexing vs ix Gotcha

Many users will find themselves using the **ix indexing capabilities** as a concise means of selecting data from a Pandas object −

Live Demo

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(6, 4), columns=['one', 'two',
'three',
'four'],index=list('abcdef'))

print df
print df.ix[['b', 'c', 'e']]
```

Its **output** is as follows −

```
          one        two      three       four
a   -1.582025   1.335773   0.961417  -1.272084
```

```
b    1.461512   0.111372  -0.072225   0.553058
c   -1.240671   0.762185   1.511936  -0.630920
d   -2.380648  -0.029981   0.196489   0.531714
e    1.846746   0.148149   0.275398  -0.244559
f   -1.842662  -0.933195   2.303949   0.677641

          one        two      three       four
b    1.461512   0.111372  -0.072225   0.553058
c   -1.240671   0.762185   1.511936  -0.630920
e    1.846746   0.148149   0.275398  -0.244559
```

This is, of course, completely equivalent in this case to using the **reindex** method −

Live Demo

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(6, 4), columns=['one', 'two',
'three',
'four'],index=list('abcdef'))

print df
print df.reindex(['b', 'c', 'e'])
```

Its **output** is as follows −

```
          one        two      three       four
a    1.639081   1.369838   0.261287  -1.662003
b   -0.173359   0.242447  -0.494384   0.346882
c   -0.106411   0.623568   0.282401  -0.916361
d   -1.078791  -0.612607  -0.897289  -1.146893
e    0.465215   1.552873  -1.841959   0.329404
f    0.966022  -0.190077   1.324247   0.678064

          one        two      three       four
b   -0.173359   0.242447  -0.494384   0.346882
c   -0.106411   0.623568   0.282401  -0.916361
e    0.465215   1.552873  -1.841959   0.329404
```

Some might conclude that **ix** and **reindex** are 100% equivalent based on this. This is true except in the case of integer indexing. For example, the above operation can alternatively be expressed as −

Live Demo

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(6, 4), columns=['one', 'two',
'three',
'four'],index=list('abcdef'))
```

```
print df
print df.ix[[1, 2, 4]]
print df.reindex([1, 2, 4])
```

Its **output** is as follows −

```
          one        two      three       four
a   -1.015695  -0.553847   1.106235  -0.784460
b   -0.527398  -0.518198  -0.710546  -0.512036
c   -0.842803  -1.050374   0.787146   0.205147
d   -1.238016  -0.749554  -0.547470  -0.029045
e   -0.056788   1.063999  -0.767220   0.212476
f    1.139714   0.036159   0.201912   0.710119

          one        two      three       four
b   -0.527398  -0.518198  -0.710546  -0.512036
c   -0.842803  -1.050374   0.787146   0.205147
e   -0.056788   1.063999  -0.767220   0.212476

    one  two  three  four
1   NaN  NaN    NaN   NaN
2   NaN  NaN    NaN   NaN
4   NaN  NaN    NaN   NaN
```

It is important to remember that **reindex is strict label indexing only**. This can lead to some potentially surprising results in pathological cases where an index contains, say, both integers and strings.

# Comparison with SQL

Since many potential Pandas users have some familiarity with SQL, this page is meant to provide some examples of how various SQL operations can be performed using pandas.

```
import pandas as pd

url = 'https://raw.github.com/pandasdev/
pandas/master/pandas/tests/data/tips.csv'

tips=pd.read_csv(url)
print tips.head()
```

Its **output** is as follows −

```
    total_bill   tip      sex  smoker  day     time  size
0        16.99  1.01   Female      No  Sun  Dinner     2
1        10.34  1.66     Male      No  Sun  Dinner     3
2        21.01  3.50     Male      No  Sun  Dinner     3
3        23.68  3.31     Male      No  Sun  Dinner     2
```

```
4         24.59  3.61    Female      No   Sun   Dinner      4
```

## SELECT

In SQL, selection is done using a comma-separated list of columns that you select (or a * to select all columns) −

```
SELECT total_bill, tip, smoker, time
FROM tips
LIMIT 5;
```

With Pandas, column selection is done by passing a list of column names to your DataFrame −

```
tips[['total_bill', 'tip', 'smoker', 'time']].head(5)
```

Let's check the full program −

```
import pandas as pd

url = 'https://raw.github.com/pandasdev/
pandas/master/pandas/tests/data/tips.csv'

tips=pd.read_csv(url)
print tips[['total_bill', 'tip', 'smoker', 'time']].head(5)
```

Its **output** is as follows −

```
   total_bill   tip  smoker     time
0       16.99  1.01      No   Dinner
1       10.34  1.66      No   Dinner
2       21.01  3.50      No   Dinner
3       23.68  3.31      No   Dinner
4       24.59  3.61      No   Dinner
```

Calling the DataFrame without the list of column names will display all columns (akin to SQL's *).

## WHERE

Filtering in SQL is done via a WHERE clause.

```
SELECT * FROM tips WHERE time = 'Dinner' LIMIT 5;
```

DataFrames can be filtered in multiple ways; the most intuitive of which is using Boolean indexing.

```
tips[tips['time'] == 'Dinner'].head(5)
```

Let's check the full program −

```
import pandas as pd

url = 'https://raw.github.com/pandasdev/
```

```
pandas/master/pandas/tests/data/tips.csv'

tips=pd.read_csv(url)
print tips[tips['time'] == 'Dinner'].head(5)
```

Its **output** is as follows −

```
   total_bill   tip      sex  smoker  day    time  size
0       16.99  1.01   Female      No  Sun  Dinner     2
1       10.34  1.66     Male      No  Sun  Dinner     3
2       21.01  3.50     Male      No  Sun  Dinner     3
3       23.68  3.31     Male      No  Sun  Dinner     2
4       24.59  3.61   Female      No  Sun  Dinner     4
```

The above statement passes a Series of True/False objects to the DataFrame, returning all rows with True.

## GroupBy

This operation fetches the count of records in each group throughout a dataset. For instance, a query fetching us the number of tips left by sex −

```
SELECT sex, count(*)
FROM tips
GROUP BY sex;
```

The Pandas equivalent would be −

```
tips.groupby('sex').size()
```

Let's check the full program −

```
import pandas as pd

url = 'https://raw.github.com/pandasdev/
pandas/master/pandas/tests/data/tips.csv'

tips=pd.read_csv(url)
print tips.groupby('sex').size()
```

Its **output** is as follows −

```
sex
Female   87
Male    157
dtype: int64
```

## Top N rows

SQL returns the **top n rows** using **LIMIT** −

```
SELECT * FROM tips
LIMIT 5 ;
```

The Pandas equivalent would be −

```
tips.head(5)
```

Let's check the full example −

```
import pandas as pd

url = 'https://raw.github.com/pandas-
dev/pandas/master/pandas/tests/data/tips.csv'

tips=pd.read_csv(url)
tips = tips[['smoker', 'day', 'time']].head(5)
print tips
```

Its **output** is as follows −

```
   smoker   day     time
0      No   Sun   Dinner
1      No   Sun   Dinner
2      No   Sun   Dinner
3      No   Sun   Dinner
4      No   Sun   Dinner
```

These are the few basic operations we compared are, which we learnt, in the previous chapters of the Pandas Library.