Boolean masking in pandas

Now that you know how to select data in pandas by referring to rows and columns, the next step is to learn how to use Boolean masks. Data professionals use Boolean masks to select data in pandas based on conditions. In this reading, you will discover Boolean masking and how to use pandas' logical operators to form multi-conditional selection statements. Understanding the fundamentals of pandas will help make your work as a data professional easier and more efficient.

Boolean masks

You know that Boolean is used to describe any binary variable whose possible values are true or false. With pandas, **Boolean masking,** also called **Boolean indexing**, is used to overlay a Boolean grid onto a dataframe's index in order to select only the values in the dataframe that align with the `True` values of the grid.

Return to the example from the video. Suppose you have a dataframe of planets, their radii, and their number of moons:

planet	radius_km	moons
Mercury	2,440	0
Venus	6,052	0
Earth	6,371	1
Mars	3,390	2
Jupiter	69,911	80
Saturn	58,232	83
Uranus	25,362	27
Neptune	24,622	14

Now suppose that you want to keep the rows of any planets that have fewer than 20 moons and filter out the rest. A Boolean mask is a pandas `Series` object indicating whether this condition is true or false for each value in the `moons` column:

	Moons < 20?
0	True
1	True
2	True
3	True
4	False
5	False
6	False
7	True

The `dtype` contained in this series is `bool`. Boolean masking effectively overlays this Boolean series onto the dataframe's index. The result is that any rows in the dataframe that are indicated as `False` in the Boolean mask get filtered out, and any rows that are indicated as `True` remain in the dataframe:

planet	radius_km	moons
Mercury	2,440	0
Venus	6,052	0
Earth	6,371	1
Mars	3,390	2
Neptune	24,622	14

Coding Boolean masks in pandas

Here is how to perform this operation in pandas.

Begin with a **`DataFrame`** object.

```
data = {'planet': ['Mercury', 'Venus', 'Earth', 'Mars',
                   'Jupiter', 'Saturn', 'Uranus', 'Neptune'],
        'radius_km': [2440, 6052, 6371, 3390, 69911, 58232,
                      25362, 24622],
        'moons': [0, 0, 1, 2, 80, 83, 27, 14]
         }
df = pd.DataFrame(data)
df
```

RunReset

Then, write a logical statement. Remember, the objective is to keep planets that have fewer than 20 moons and filter out the rest.

```
print(df['moons'] < 20)
```

RunReset

This results in a **`Series`** object of **`dtype: bool`** that consists of the row indices, where each index contains a **`True`** or **`False`** value depending on whether that row satisfies the given condition. This is the Boolean mask. To apply this mask to the dataframe, simply insert this statement into selector brackets and apply it to your dataframe:

```
print(df[df['moons'] < 20])
```

RunReset

You can also assign the Boolean mask to a named variable and then apply that to your dataframe:

```
mask = df['moons'] < 20
df[mask]
```

RunReset

Note that this doesn't permanently modify your dataframe. It only gives a filtered view of it.

```
df
```

RunReset

However, you can assign the result to a named variable:

```
mask = df['moons'] < 20
df2 = df[mask]
df2
```

RunReset

And if you want to select just the planet column as a `Series` object, you can use regular selection tools like `loc[]`:

```
mask = df['moons'] < 20
df.loc[mask, 'planet']
```

RunReset

Complex logical statements

In statements that use multiple conditions, pandas uses logical operators to indicate which data to keep and which to filter out. These operators are:

Operator	Logic
`&`	`and`
`|`	`or`
`~`	`not`

Important: Each component of a multi-conditional logical statement must be in parentheses. Otherwise, the statement will throw an error or, worse, return something that isn't what you intended.

For example, here is how to create a Boolean mask that selects all planets that have fewer than 10 moons or greater than 50 moons:

```
mask = (df['moons'] < 10) | (df['moons'] > 50)
mask
```

RunReset

Notice that each condition is self-contained in a set of parentheses, and the two conditions are separated by the logical operator, `|` (or). To apply the mask, call the dataframe and put the statement or the variable it's assigned to in selector brackets:

```
mask = (df['moons'] < 10) | (df['moons'] > 50)
df[mask]
```

RunReset

Here's an example of how to select all planets that have more than 20 moons, but not planets with 80 moons and not planets with a radius less than 50,000 km:

```
mask = (df['moons'] > 20) & ~(df['moons'] == 80) & ~(df['radius_km'] < 50000)
df[mask]
```

RunReset

Note that this returns the same result as the following:

```
mask = (df['moons'] > 20) & (df['moons'] != 80) & (df['radius_km'] >= 50000)
df[mask]
```

RunReset

Working with pandas dataframes, using their attributes and methods, and selecting data using Boolean masks are some of the core daily activities of a data professional. You'll soon be using these tools often as you progress on your journey with pandas.

Key takeaways

A Boolean mask is a method of applying a filter to a dataframe. The mask overlays a Boolean grid over your dataframe in order to select only the values in the dataframe that align with the True values of the grid. To create Boolean comparisons, pandas has its own logical operators. These operators are:

- `&` (and)
- `|` (or)

- ~ (not)

Each criterion of a multi-conditional selection statement must be enclosed in its own set of parentheses. With practice, making complex selection statements in pandas is possible and efficient.