Sorting and Filtering Rows

Pandas `.index` Attribute

The pandas DataFrame attribute `.index` displays the row labels of the DataFrame.
By default, DataFrame rows are labeled using a **RangeIndex** where the first row is labeled `0`, the second row is labeled `1`, and so on.
Indexes can also consist of text objects/strings and other values.

```
df.index
```

Pandas `.reset_index()` Method

The method `.reset_index()` replaces existing row labels for a DataFrame with the standard index: 0 for the first row, 1 for the second row, and so on.
By default, this method will keep the old labels in a new column named `'index'`.

```
df.reset_index()
```

Pandas `.sort_index()` Method

The pandas DataFrame method `.sort_index()` sorts the rows of a DataFrame by the index values.
The parameter `ascending` controls how the rows are sorted.
For numerical indexes, `.sort_index()` sorts from smallest to largest numerically. For text indexes, `.sort_index()` sorts from A to Z.
Setting `ascending` to `False` sorts the index in reverse/descending order.

```
# Sort the DataFrame by its index in ascending order
vehicles.sort_index()

# Sort the DataFrame by its index in descending order
vehicles.sort_index(ascending=False)
```

codecademy

Selecting data with `.loc[]`

Specific data in a pandas DataFrame can be accessed using

```
.loc[row_labels_list, column_l
```

Here's a sample DataFrame `vehicles` :

	id	model	year
0	12988	amg e53 4matic+ (convertible)	2022
1	689	avalanche ffv	2007
2	950	impala	2010

The code snippet in this review card performs the following selection:

	model	year
0	amg e53 4matic+ (convertible)	2022
1	avalanche ffv	2007

```
vehicles.loc[[0,1], ['model','year']]
```

codecademy

Selecting data with `.iloc[]`

Specific data in a pandas DataFrame can be accessed using

```
.iloc[row_positions, column_pc
```

By default, the position of the first row/column is 0 , the position of the second is 1 , and so on. Here's a sample DataFrame:

	id	model	year
0	12988	amg e53 4matic+ (convertible)	2022
1	689	avalanche ffv	2007
2	950	impala	2010

The code snippet in this review card performs the following selection:

	model	year
0	amg e53 4matic+ (convertible)	2022
2	impala	2010

```
vehicles.iloc[[0,2], [1,2]]
```

codecademy

Selecting Slices with `.loc[]`

	id	model	year	bes
0	12988	amg e53 4matic+ (convertible)	2022	28.
1	689	avalanche ffv	2007	21.
2	950	impala	2010	29.

Ranges of rows and columns can be selected using `.loc[]` and slice syntax:

```
df.loc[start_row:end_row, star
```

A slice `start:end` contains the row/column *labeled* `start`, the row/column *labeled* `end`, and all rows/columns between.
If either `start` or `end` is omitted, the first or last row/column is used.

```
# Select rows labeled 0 and 1
# columns labeled
'model','year','best_mpge'
vehicles[0:1, 'model':'best_mpge']

# Select all rows
# columns 'id','model', and 'year'
vehicles[:, :'year']
```

Selecting Slices with `.iloc[]`

	id	model	year	bes
0	12988	amg e53 4matic+ (convertible)	2022	28.
1	689	avalanche ffv	2007	21.
2	950	impala	2010	29.

Ranges of rows and columns can be selected using `.iloc[]` and slice syntax:

```
df.iloc[start_row:end_row, sta
```

A slice `start:end` contains the row/column *in position* `start` and all rows up to but *not including* the row/column in position `end`. Positions start at 0, and increase top-bottom (for rows) and left-right (for columns).
If either `start` or `end` is omitted, the first or last row/column is used.

```
# Select the first two rows and the last three columns
vehicles.iloc[0:2, 1:4]

# Select all rows and the first three columns
vehicles.iloc[:, :3]
```

Boolean Variables

In Python, a variable is **Boolean** if it has the value `True` or `False` (without quotes).

```
# Example Booleans
is_raining = True
is_sunny = False
```

codecademy

Python Comparison Operator `==`

The Python equal comparison operator `==` returns `True` if the variables being compared have exactly the same value, otherwise, it returns `False`.
The Python not equal comparison operator `!=` returns `True` if the variables being compared have different values, and otherwise returns `True`.

```
3 == 3
# Output: True

3 != 3
# Output: False

'auto' == 'auto'
# Output: True

'auto' != 'auto'
# Output: False
```

Relational Comparison Operators

Python has four **comparison operators** to compare sizes. These include:

- less than (`<`)
- greater than (`>`)
- less than or equal to (`<=`)
- greater than or equal to (`>=`)

These operators return `True` if their comparison is valid and `False` otherwise.

```
10 < 20
# Output: True

10 > 20
# Output: False

10 <= 20
# Output: True

10 >= 10
# Output: True
```

codecademy

Python `and` Operator

The `and` operator in Python combines two Booleans. The `and` operator is

- `True` if both Booleans are True
- `False` otherwise

```
(1 < 2) and (1 == 3)
# Output: False, because the second Boolean is False

(1 > 2) and (1 < 3)
# Output: False, because the first Boolean is False

(1 > 2) and (1 == 3)
# Output: False, because both Booleans are False

(1 < 2) and (1 < 3)
# Output: True, because both Booleans are True
```

Python `or` Operator

The Python `or` operator combines two Boolean expressions. It is

- `True` if *at least one* of the Booleans being combined is `True`
- `False` if *both* Booleans are `False`

In particular, `or` is inclusive: if both Booleans are `True` , `or` is `True` .

```
(1 < 2) or (1 == 3)
# Output: True, because the first Boolean is True

(1 > 3) or (1 < 3)
# Output: True, because the second Boolean is True

(1 < 2) or (1 < 3)
# Output: True, because both Booleans are True

(1 > 2) or (1 == 3)
# Output: False, because both Booleans are False
```

codecademy

Python `not` Operator

The `not` operator inverts the value of a Boolean:

- if the original Boolean was `True` , then placing `not` in front will make it `False`
- if the original Boolean was `False` , then placing `not` in front will make it `True`

It is good practice to place the Boolean being inverted within parentheses: `not(Boolean)` .

```
not(1 < 2)
# Output: False, because 1 < 2 is True

not(1 > 2)
# Output: True, because 1 > 2 is False
```

Pandas Boolean Masks

Performing a comparison between a DataFrame column and a value creates a **Boolean mask**: a copy of the column where each row is replaced with the value `True` if the comparison is true and `False` otherwise.

	model	year	best_mpge
0	amg e53 4matic+ (convertible)	2022	28.0
1	avalanche ffv	2007	21.0
2	impala	2010	29.0

The Boolean mask in the code snippet returns:

0	True
1	False
2	True

```
gt_25 = vehicles['best_mpge'] > 25.0
```

Filtering with Boolean Masks

Passing a Boolean mask to a DataFrame returns only the rows where the mask is `True`. Here is a sample DataFrame named `vehicles`.

	model	year	best_mpge
0	amg e53 4matic+ (convertible)	2022	28.0
1	avalanche ffv	2007	21.0
2	impala	2010	29.0

The code snippet in this review card filters `vehicles` down to models from 2010:

	model	year	best_mpge
2	impala	2010	29.0

```
# Create a Boolean mask
is_2010 = vehicles['year'] == 2010

# Filter vehicles using the mask
vehicles[is_2010]
```

Combining Boolean Masks

	year	mpge	recent & under_29	recent under_
0	2022	28.0	True	True
1	2007	21.0	False	True
2	2010	29.0	False	False

Boolean Masks can be combined using the operators

- `&`, meaning `and`
- `|`, meaning `or`

```
# Boolean mask for models newer than
2010
recent = vehicles['year'] > 2010

# Boolean mask for models under 29mpge
under_29 = vehicles['mpge'] < 29

# Boolean mask for models that are both
newer than 2010 and under 29mpge
vehicles['recent & under_29'] = recent &
under_29

# Boolean mask for models that are
either newer than 2010 or under 29mpge
vehicles['recent | under_29'] = recent |
under_29
```

codecademy

Pandas .sort_values() Method

The pandas DataFrame method `.sort_values()` sorts the rows of a DataFrame, generally using two parameters:

- `by=` to select the column to sort by
- `ascending=` to control the order of the sort (default `True`)

	model	year	best_mpge
0	amg e53 4matic+ (convertible)	2022	28.0
1	avalanche ffv	2007	21.0
2	impala	2010	29.0

Sorting by `best_mpge` from highest to lowest:

	model	year	best_mpge
2	impala	2010	29.0
0	amg e53 4matic+ (convertible)	2022	28.0
1	avalanche ffv	2007	21.0

```
# Sort vehicles by 'best_mpge' from
lowest to highest
vehicles.sort_values(by='best_mpge')

# Sort vehicles by 'best_mpge' from
highest to lowest
vehicles.sort_values(by='best_mpge',
                     ascending=False)
```

Print Share