15 rows × 7 columns

In [11]:
```
1 dogs.tail(2)
```

Out[11]:

	breed	kind	lifetime_cost	longevity	size	weight	height
40	Mastiff	working	13581.0	6.50	large	175.0	30.0
41	Saint Bernard	working	20022.0	7.78	large	155.0	26.5

- To sort by a column, use the `sort_values` method.

  `ascending=False` is a **keyword argument**, meaning you need to specify the name of the argument to use it.
  You've seen some examples of this in the `plotly` part of Homework 1.

In [13]:
```
1 # Note that the index is no longer 0, 1, 2, ...!
2 dogs.sort_values('lifetime_cost')
```

by default, sorts in ascending order

index no longer sorted!

Out[13]:

	breed	kind	lifetime_cost	longevity	size	weight	height
40	Mastiff	working	13581.0	6.50	large	175.0	30.0
38	Bloodhound	hound	13824.0	6.75	large	85.0	25.0
39	Bullmastiff	working	13936.0	7.57	large	115.0	25.5
...	...	...	...	...	...	...	...
9	German Shorthaired Pointer	sporting	25842.0	11.46	large	62.5	24.0
7	Chihuahua	toy	26250.0	16.50	small	5.5	5.0
29	Giant Schnauzer	working	26686.0	10.00	large	77.5	25.5

42 rows × 7 columns

In [34]:
```
1 dogs
```

Out[34]:

breed	kind	lifetime_cost	longevity	size	weight	height
Brittany	sporting	22589.0	12.92	medium	35.0	19.0
Cairn Terrier	terrier	21992.0	13.84	small	14.0	10.0
English Cocker Spaniel	sporting	18993.0	11.66	medium	30.0	16.0
...	...	...	...	...	...	...
Bullmastiff	working	13936.0	7.57	large	115.0	25.5
Mastiff	working	13581.0	6.50	large	175.0	30.0
Saint Bernard	working	20022.0	7.78	large	155.0	26.5

42 rows × 6 columns

In [35]:
```
1 # Returns a Series. Note the index appears again on the left!
2 dogs['height']
```

index stays

Out[35]:
```
breed
Brittany                19.0
Cairn Terrier           10.0
English Cocker Spaniel  16.0
                        ...
Bullmastiff             25.5
Mastiff                 30.0
Saint Bernard           26.5
Name: height, Length: 42, dtype: float64
```

this is a Series

In [ ]:
```
1 # Returns a DataFrame.
```

```
Mastiff                 30.0
Saint Bernard           26.5
Name: height, Length: 42, dtype: float64
```

In [37]:
```
dogs[['height', 'longevity', 'lifetime_cost']]
```

Out[37]:

breed	height	longevity	lifetime_cost
Brittany	19.0	12.92	22589.0
Cairn Terrier	10.0	13.84	21992.0
English Cocker Spaniel	16.0	11.66	18993.0
...	...	...	...
Bullmastiff	25.5	7.57	13936.0
Mastiff	30.0	6.50	13581.0
Saint Bernard	26.5	7.78	20022.0

42 rows × 3 columns

one value → Series

list → DataFrame.

In [36]:
```
# Returns a DataFrame.
dogs[['height']]
```

Out[36]:

breed	height
Brittany	19.0
Cairn Terrier	10.0
English Cocker Spaniel	16.0
...	...
Bullmastiff	25.5

```
File index.pyx:181, in pandas._

File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'breed'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[39], line 2
      1 # Breeds are stored in the index, which is not a column!
----> 2 dogs['breed']

File ~/miniforge3/envs/pds/lib/python3.10/site-packages/pandas/core/frame.py:3896, in DataFrame.__getitem__(self, key)
   3894 if self.columns.nlevels > 1:
   3895     return self._getitem_multilevel(key)
-> 3896 indexer = self.columns.get_loc(key)
   3897 if is_integer(indexer):
   3898     indexer = [indexer]

File ~/miniforge3/envs/pds/lib/python3.10/site-packages/pandas/core/indexes/base.py:3797, in Index.get_loc(self, key)
   3792     if isinstance(casted_key, slice) or (
   3793         isinstance(casted_key, abc.Iterable)
   3794         and any(isinstance(x, slice) for x in casted_key)
   3795     ):
   3796         raise InvalidIndexError(key)
-> 3797     raise KeyError(key) from err
   3798 except TypeError:
   3799     # If we have a listlike key, _check_indexing_error will raise
   3800     #  InvalidIndexError. Otherwise we fall through and re-raise
   3801     #  the TypeError.
   3802     self._check_indexing_error(key)

KeyError: 'breed'
```

look at bottom

In [ ]:
```
1 ...
```

Cairn Terrier	terrier	21992.0	13.84	small	14.0	10.0
English Cocker Spaniel	sporting	18993.0	11.66	medium	30.0	16.0
...	...	...	...	...	...	...
Bullmastiff	working	13936.0	7.57	large	115.0	25.5
Mastiff	working	13581.0	6.50	large	175.0	30.0
Saint Bernard	working	20022.0	7.78	large	155.0	26.5

42 rows × 6 columns

In [43]:
```
# What are the unique kinds of dogs?
dogs['kind'].unique()
```

Out[43]:
```
array(['sporting', 'terrier', 'herding', 'working', 'toy', 'non-sporting',
       'hound'], dtype=object)
```

In [44]:
```
# How many unique kinds of dogs are there?
dogs['kind'].nunique()
```

Out[44]:
```
7
```

In [45]:
```
# What's the distribution of kinds?
# value_counts is super useful — and I love asking exam questions about it!
dogs['kind'].value_counts()
```

Out[45]:
```
kind
sporting        12
terrier          8
working          7
toy              6
hound            5
herding          2
non-sporting     2
Name: count, dtype: int64
```

unique values

↑ frequencies

In [58]:
```
1 show_chaining_slides()
```

```
breed
Mastiff                 large
Saint Bernard           large
Newfoundland            large
                        ...
Rhodesian Ridgeback     large
Giant Schnauzer         large
Clumber Spaniel        medium
Name: size, Length: 10, dtype: object
```

```
size
large     9
medium    1
Name: count, dtype: int64
```

```
dogs.sort_values("weight", ascending=False).head(10)["size"].value_counts().index[0]
```

type: Series

Series support vectorized operations

- Series operations are vectorized, just like with arrays.

- When performing elementwise-operations involving multiple Series, `pandas` aligns the Series by their **index**.

In [61]:
```
x = pd.Series({'a': 1, 'b': 2})
x
```

Out[61]:
```
a    1
b    2
dtype: int64
```

values in a Series are shown on the right!

In [62]:
```
x * 5
```

Out[62]:
```
a     5
b    10
dtype: int64
```

In [ ]:
```
y = pd.Series({'b': 5, 'c': -1, 'a': 10})
y
```

In [ ]:
```
# If x and y were regular numpy arrays, this would error because of the size mismatch.
x + y
```

with 2D arrays.

In [94]:
```
1 dogs
```

Out[94]:

breed	kind	lifetime_cost	longevity	size	weight	height
Brittany	sporting	22589.0	12.92	medium	35.0	19.0
Cairn Terrier	terrier	21992.0	13.84	small	14.0	10.0
English Cocker Spaniel	sporting	18993.0	11.66	medium	30.0	16.0
...	...	...	...	...	...	...
Bullmastiff	working	13936.0	7.57	large	115.0	25.5
Mastiff	working	13581.0	6.50	large	175.0	30.0
Saint Bernard	working	20022.0	7.78	large	155.0	26.5

42 rows × 6 columns

In [98]:
```
1 # Try removing the iloc and see what happens!
2 dogs.iloc[:2, 2:]
```

Out[98]:

breed	longevity	size	weight	height
Brittany	12.92	medium	35.0	19.0
Cairn Terrier	13.84	small	14.0	10.0

Out[114]:
```
breed
Brittany                  False
Cairn Terrier              True
English Cocker Spaniel    False
                           ...
Bullmastiff               False
Mastiff                   False
Saint Bernard             False
Name: kind, Length: 42, dtype: bool
```

In [115]:
```
1 dogs.loc[dogs['kind'] == 'terrier']
```

Out[115]:

breed	kind	lifetime_cost	longevity	size	weight	height
Cairn Terrier	terrier	21992.0	13.84	small	14.0	10.0
Miniature Schnauzer	terrier	20087.0	11.81	small	15.5	13.0
Norfolk Terrier	terrier	24308.0	13.07	small	12.0	9.5
...	...	...	...	...	...	...
Scottish Terrier	terrier	17525.0	10.69	small	20.0	10.0
Kerry Blue Terrier	terrier	17240.0	9.40	medium	36.5	18.5
Bull Terrier	terrier	18490.0	10.21	medium	60.0	21.5

8 rows × 6 columns

keeps just the rows where Series has True

- Example: Among all breeds with `'Retriever'` in the name, which is the second tallest?

In [149]:
```
(
    dogs.loc[dogs.index.str.contains('Retriever'), 'height']
    .sort_values(ascending=False)
    .index[1]
)
```

Out[149]: 'Labrador Retriever'

In [145]:
```
# Since we're selecting both rows AND columns, we do need loc here.
(
    dogs[dogs.index.str.contains('Retriever')]
    .sort_values('height', ascending=False)
    .index[1]
)
```

Out[145]: 'Labrador Retriever'

understand the difference!

- Example: Among all breeds with `'Retriever'` in the name, which is the second tallest?

In [149]:
```
(
    dogs.loc[dogs.index.str.contains('Retriever'), 'height']
    .sort_values(ascending=False)
    .index[1]
)
```

Out[149]: 'Labrador Retriever'

In [145]:
```
# Since we're selecting both rows AND columns, we do need loc here.
(
    dogs[dogs.index.str.contains('Retriever')]
    .sort_values('height', ascending=False)
    .index[1]
)
```

Out[145]: 'Labrador Retriever'

understand the difference!

In [ ]:
```
1
```

Find the average height of medium dogs.

In [174]:
```
1 dogs.loc[dogs['size'] == 'small', 'height'].mean()
```

Out[174]: 10.883333333333333

In [172]:
```
1 dogs.loc[dogs['size'] == 'medium', 'height'].mean()
```

Out[172]: 19.0

In [175]:
```
1 dogs.loc[dogs['size'] == 'large', 'height'].mean()
```

Out[175]: 25.8

In [178]:
```
1 dogs.groupby('size')['height'].mean()
```

Out[178]:
```
size
large     25.80
medium    19.00
small     10.88
Name: height, dtype: float64
```

In [ ]:
```
1
```