localhost
12.1

The magic of `groupby` 🪄

"for each"

"for every"

- A better solution is to use the `groupby` method.

```
In [15]: # To find the overall mean 'body_mass_g':
         penguins['body_mass_g'].mean()

Out[15]: 4207.057057057057
```

```
In [16]: # To find the mean 'body_mass_g' for each 'species':
         penguins.groupby('species')['body_mass_g'].mean()

Out[16]: species
         Adelie       3706.16
         Chinstrap    3733.09
         Gentoo       5092.44
         Name: body_mass_g, dtype: float64
```

same as last slide!

- Somehow, the `groupby` method computes what we're looking for in just one line. How?

- We'll work through the internals, but remember this: **if you need to calculate something *for each group*, use `groupby`!**

	Species	Color	Weight	Age
0	dog	black	40	5.0
1	cat	golden	15	8.0
2	cat	black	20	9.0
3	dog	white	80	2.0
4	dog	golden	25	0.5
5	hamster	golden	1	3.0

	Species	Color	Weight	Age
0	dog	black	40	5.0
3	dog	white	80	2.0
4	dog	golden	25	0.5

	Weight	Age
Species		
dog	48.333333	2.5

avg

	Species	Color	Weight	Age
1	cat	golden	15	8.0
2	cat	black	20	9.0

	Weight	Age
Species		
cat	17.5	8.5

avg

	Species	Color	Weight	Age
5	hamster	golden	1	3.0

	Weight	Age
Species		
hamster	1.0	3.0

```
pets.groupby(“Species”)[[“Weight”, Age”]].mean()
```

"column independence"

	Species	Color	Weight	Age
0	dog	black	40	5.0
1	cat	golden	15	8.0
2	cat	black	20	9.0
3	dog	white	80	2.0
4	dog	golden	25	0.5
5	hamster	golden	1	3.0

	Species	Color	Weight	Age
0	dog	black	40	5.0
3	dog	white	80	2.0
4	dog	golden	25	0.5

→

	Color	Weight	Age
Species			
dog	white	80	5.0

last in dictionary

these are not from the same dog!

	Species	Color	Weight	Age
1	cat	golden	15	8.0
2	cat	black	20	9.0

→

	Color	Weight	Age
Species			
cat	golden	20	9.0

	Species	Color	Weight	Age
5	hamster	golden	1	3.0

→

	Color	Weight	Age
Species			
hamster	golden	1	3.0

```
pets.groupby("Species").max()
```

localhost
T
16 . 1

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Dream	41.3	20.3	194.0	3550.0	Male
1	Adelie	Torgersen	38.5	17.9	190.0	3325.0	Female
2	Adelie	Dream	34.0	17.1	185.0	3400.0	Female
...	...	...	...	...	...	...	...
330	Chinstrap	Dream	46.6	17.8	193.0	3800.0	Female
331	Adelie	Dream	39.7	17.9	193.0	4250.0	Male
332	Gentoo	Biscoe	45.1	14.5	207.0	5050.0	Female

333 rows × 7 columns

```
In [23]: penguins.groupby('species')['bill_length_mm'].median()

Out[23]: species
         Adelie       38.85
         Chinstrap    49.55
         Gentoo       47.40
         Name: bill_length_mm, dtype: float64

In [22]: penguins.groupby('species')['bill_length_mm'].median().idxmax()

Out[22]: 'Chinstrap'

In [30]: penguins.groupby('species')['bill_length_mm'].median().sort_values(ascending=False).index[0]

Out[30]: 'Chinstrap'

In [ ]:
```

localhost
22.1

```
Out[40]: species
         Adelie       541100.0
         Chinstrap    253850.0
         Gentoo       606000.0
         Name: body_mass_g, dtype: float64
```

```
In [45]: # Often used in conjunction with sort_values.
         # Remember this when you work on the activity in a few slides!
         penguins.groupby('species').first()
```

Out[45]:

	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
species						
Adelie	Dream	41.3	20.3	194.0	3550.0	Male
Chinstrap	Dream	45.5	17.0	196.0	3500.0	Female
Gentoo	Biscoe	46.4	15.0	216.0	4700.0	Female

```
In [46]: # Similar to value_counts, but not identical!
         penguins.groupby('species').size()
```

sorts by index in ascending order.

```
Out[46]: species
         Adelie       146
         Chinstrap     68
         Gentoo       119
         dtype: int64
```

both Series!

```
In [47]: penguins['species'].value_counts()
```

sorts by count in descending order

```
Out[47]: species
         Adelie       146
         Gentoo       119
         Chinstrap     68
         Name: count, dtype: int64
```

Reminder: Column independence

- As we've seen, within each group, the aggregation method is applied to **each column independently**.

```
In [48]: penguins.groupby('species').max()
```

Out[48]:

the last island, ALPHABETICALLY, of all Adelies

	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
species						
Adelie	Torgersen	46.0	21.5	210.0	4775.0	Male
Chinstrap	Dream	58.0	20.8	212.0	4800.0	Male
Gentoo	Biscoe	59.6	17.3	231.0	6300.0	Male

not a real penguin!

- The above result **is not** telling us that there is a `'Adelie'` penguin with a `'body_mass_g'` of `4775.0` that lived on `'Torgersen'` island.

```
In [49]: # This penguin lived on Biscoe island!
         penguins.loc[(penguins['species'] == 'Adelie') & (penguins['body_mass_g'] == 4775.0)]
```

Out[49]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
255	Adelie	Biscoe	43.2	19.0	197.0	4775.0	Male

is a real penguin.

localhost
25 . 1

Out[56]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Dream	41.3	20.3	194.0	3550.0	Male
1	Adelie	Torgersen	38.5	17.9	190.0	3325.0	Female
2	Adelie	Dream	34.0	17.1	185.0	3400.0	Female
...	...	...	...	...	...	...	...
330	Chinstrap	Dream	46.6	17.8	193.0	3800.0	Female
331	Adelie	Dream	39.7	17.9	193.0	4250.0	Male
332	Gentoo	Biscoe	45.1	14.5	207.0	5050.0	Female

333 rows × 7 columns

In [57]:

```
penguins['island'] == 'Dream'
```

Out[57]:

```
0       True
1      False
2       True
       ...
330     True
331     True
332    False
Name: island, Length: 333, dtype: bool
```

In [58]:

```
(penguins['island'] == 'Dream').mean()
```

Out[58]:

```
0.36936936936936937
```

36.9% of all penguins
live on Dream Island.

In [ ]:

In [60]:
```
(
    penguins
    .assign(lives_on_dream = (penguins['island'] == 'Dream'))
)
```

added a new column (read guide)

Out[60]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	lives_on_dream
0	Adelie	Dream	41.3	20.3	194.0	3550.0	Male	True
1	Adelie	Torgersen	38.5	17.9	190.0	3325.0	Female	False
2	Adelie	Dream	34.0	17.1	185.0	3400.0	Female	True
...	...	...	...	...	...	...	...	...
330	Chinstrap	Dream	46.6	17.8	193.0	3800.0	Female	True
331	Adelie	Dream	39.7	17.9	193.0	4250.0	Male	True
332	Gentoo	Biscoe	45.1	14.5	207.0	5050.0	Female	False

333 rows × 8 columns

conditional probabilities WITH DIFFERENT DENOMS

In [61]:
```
(
    penguins
    .assign(lives_on_dream = (penguins['island'] == 'Dream'))
    .groupby('species')
    ['lives_on_dream']
    .mean()
)
```

38% of the Adelies live on D.I.

100% of the Chinstraps live on Dream Island

Out[61]:
```
species
Adelie       0.38
Chinstrap    1.00
Gentoo       0.00
Name: lives_on_dream, dtype: float64
```

agg with a custom agg

- **Example**: What is the **second largest** recorded `'body_mass_g'` for each `'species'`?

In [71]:
```
def second_largest(s):
    '''returns the second largest value in s.'''
    return s.sort_values().iloc[-2]
```

In [72]:
```
# Here, the argument to agg is a function,
# which takes in a Series and returns a scalar.
penguins.groupby('species')['body_mass_g'].agg(second_largest)
```

Out[72]:
```
species
Adelie       4725.0
Chinstrap    4550.0
Gentoo       6050.0
Name: body_mass_g, dtype: float64
```

equivalent!

In [73]:
```
penguins.groupby('species')['body_mass_g'].agg(lambda s: s.sort_values().iloc[-2])
```

defined the aggregation method on-the-fly

Out[73]:
```
species
Adelie       4725.0
Chinstrap    4550.0
Gentoo       6050.0
Name: body_mass_g, dtype: float64
```

- **Key idea**: If you give `agg` a custom function, it should map `Series` → `number`.

agg with a custom agg

- **Example**: What is the **second largest** recorded `'body_mass_g'` for each `'species'`?

```
In [71]: def second_largest(s):
             '''returns the second largest value in s.'''
             return s.sort_values().iloc[-2]
```

```
In [72]: # Here, the argument to agg is a function,
         # which takes in a Series and returns a scalar.
         penguins.groupby('species')['body_mass_g'].agg(second_largest)
```

```
Out[72]: species
         Adelie       4725.0
         Chinstrap    4550.0
         Gentoo       6050.0
         Name: body_mass_g, dtype: float64
```

equivalent!

```
In [73]: penguins.groupby('species')['body_mass_g'].agg(lambda s: s.sort_values().iloc[-2])
```

defined the aggregation method on-the-fly

```
Out[73]: species
         Adelie       4725.0
         Chinstrap    4550.0
         Gentoo       6050.0
         Name: body_mass_g, dtype: float64
```

- **Key idea**: If you give `agg` a custom function, it should map `Series` → `number`.

- A **filter**, on the other hand, keeps **entire groups** that satisfy conditions.

- For instance, to see the **penguin** `'species'` with an *average* `'bill_length_mm'` over 47 mm, use the `filter` method after `groupby`:

In [75]:
```
(
    penguins
    .groupby('species')
    .filter(lambda df: df['bill_length_mm'].mean() > 47)
)
```

FILTER: selects groups

Out[75]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
3	Chinstrap	Dream	45.5	17.0	196.0	3500.0	Female
5	Chinstrap	Dream	50.1	17.9	190.0	3400.0	Female
7	Chinstrap	Dream	51.4	19.0	201.0	3950.0	Male
...	...	...	...	...	...	...	...
329	Chinstrap	Dream	51.3	19.9	198.0	3700.0	Male
330	Chinstrap	Dream	46.6	17.8	193.0	3800.0	Female
332	Gentoo	Biscoe	45.1	14.5	207.0	5050.0	Female

187 rows × 7 columns

shorter than 47

QUERY: selects rows

- Notice that the above DataFrame has 187 rows, fewer than the 333 in the full DataFrame. That's because there are no `'Adelie'` penguins above!

There is only one penguins 'species' with:

- At least 100 penguins.
- At least 60 'Female' penguins.

Find the 'species' using **a single expression** (i.e. no intermediate variables). Use fil

```
In [81]: (
             penguins
             .groupby('species')
             .filter(lambda df: (df.shape[0] >= 100) and (df[df['sex'] == 'Female'].shape[0] >= 60))
             ['species']
             .unique()
             [0]
         )

Out[81]: 'Adelie'

In [ ]:

In [ ]: ...
```

individual Boolean

individual Boolean (either are True OR one False)

using and instead of &.

In [85]:
```
penguins['body_mass_g']
```

Out[85]:
```
0      3550.0
1      3325.0
2      3400.0
        ...
330    3800.0
331    4250.0
332    5050.0
Name: body_mass_g, Length: 333, dtype: float64
```

In [84]:
```
penguins.groupby('species')['body_mass_g'].transform(lambda s: s - s.mean())
```

Out[84]:
```
0     -156.16
1     -381.16
2     -306.16
        ...
330     66.91
331    543.84
332    -42.44
Name: body_mass_g, Length: 333, dtype: float64
```

the means are computed separately per species

- Notice that penguin 332's transformed `'body_mass_g'` is negative, even though their actual `'body_mass_g'` is very large; this is because they have a below-average `'body_mass_g'` for their `'species'`.

> - **Key idea**: If you give `transform` a custom function, it should map `Series` → `Series`.

- **Example**: Find the two heaviest penguins per 'species'.

```
In [86]: penguins.groupby('species').apply(lambda df: df.sort_values('body_mass_g', ascending=False).head(2))
```

Out[86]:

		species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
species								
Adelie	**255**	Adelie	Biscoe	43.2	19.0	197.0	4775.0	Male
	174	Adelie	Biscoe	41.0	20.0	203.0	4725.0	Male
Chinstrap	**99**	Chinstrap	Dream	52.0	20.7	210.0	4800.0	Male
	187	Chinstrap	Dream	52.8	20.0	205.0	4550.0	Male
Gentoo	**268**	Gentoo	Biscoe	49.2	15.2	221.0	6300.0	Male
	189	Gentoo	Biscoe	59.6	17.0	230.0	6050.0	Male

sets the index to "species".

- **Example**: Find the 'flipper_length_mm' of the heaviest penguin of each 'species'.

```
In [87]: penguins.groupby('species').apply(lambda df: df.sort_values('body_mass_g', ascending=False)['flipper_length_mm'].iloc[0])
```

```
Out[87]: species
         Adelie       197.0
         Chinstrap    210.0
         Gentoo       221.0
         dtype: float64
```

- **Key idea**: If you give apply a custom function, it should map DataFrame → anything. The outputs of the custom function will be stitched together intelligently by pandas.