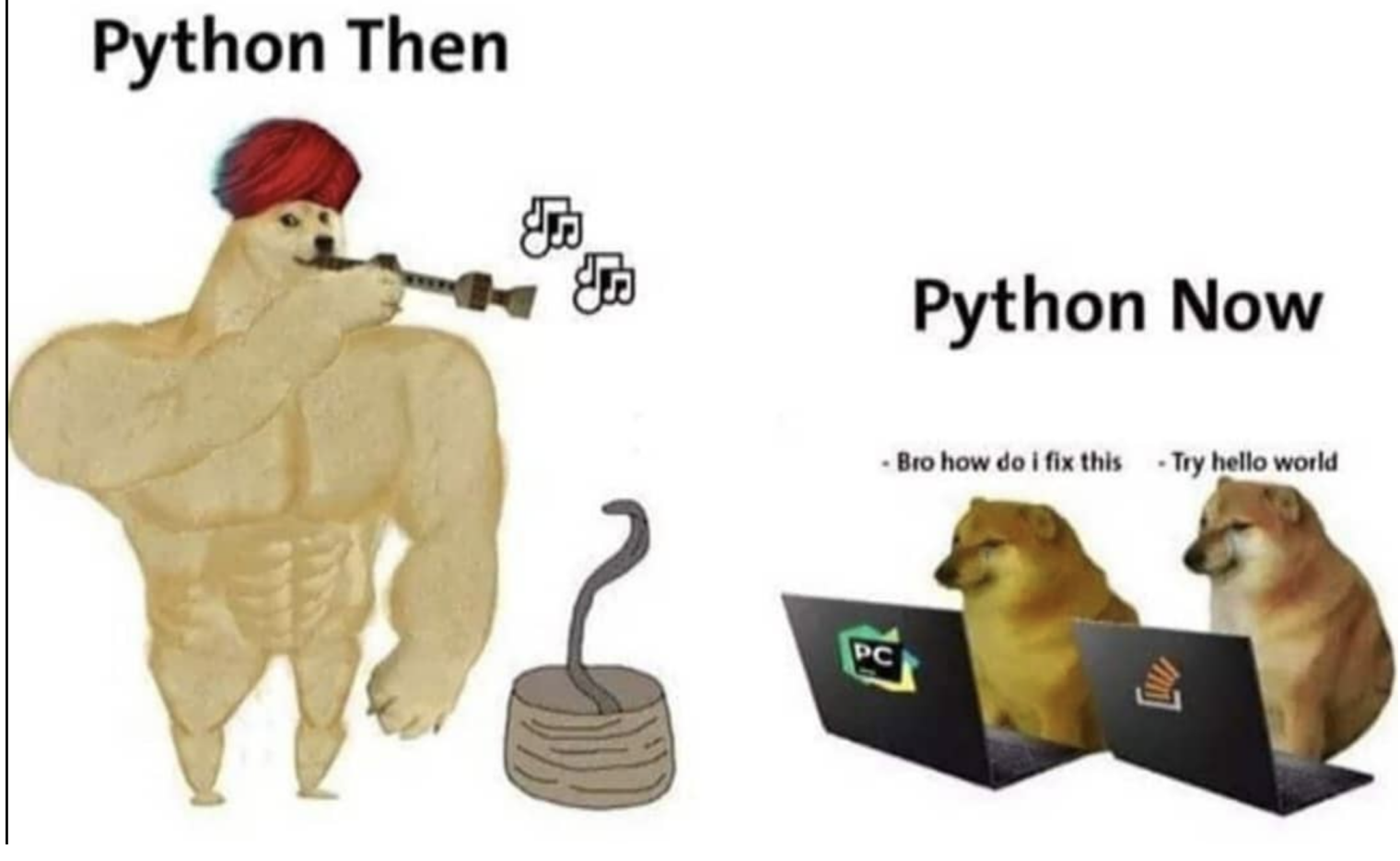
Discussion 5

DSC 20, Fall 2023

Meme of the week



Agenda

- Midterm Feedback
- **Content**
    - lambda
    - map/filter
    - HOF
    - args, kwargs, defaults
- Practice Questions
- (if we have time) project foreshadowing: machine learning

About the Midterm

- grades will come out when they do
- make sure to look through your score and submit regrade requests for sections you disagree with

reminder:

Exam Redemption Policy

The final will be split into 3 parts: midterm1, midterm2, and new material. We offer midterm redemption opportunities only for those who have taken **both** midterm exams. You could replace your midterm score (not the final exam part) with the score you earn for the counterpart on the final exam (i.e. maximum between midterm1 score (%) and final-part1 (%), maximum between midterm2 score (%) and final-part2 (%)). If you simply miss a midterm, you are **NOT** eligible for this redemption policy.

- You can't skip any part of the final regardless of your midterms score. The entire exam needs to be taken.
- You must score **at least 55%** on the final exam to pass the course. If you score lower than 55% on the final, you will receive an F in the course, regardless of your overall average.

Lambda Functions

- known as anonymous functions (their functions are so simple, they don't need a name)
- syntax: lambda (input): (some operation)
- within the scope of this course, lambda is used in conjunction with map and filter

In [4]:
```
def add_2(x):
    return x+2
add_2(1)
```

Out[4]: 3

In [3]:
```
func = lambda x: x+2
func(1)
```

Out[3]: 3

Checkpoint

Are the following 2 functions equivalent?

In [5]:
```
def strip_caps(string):
    output = ''
    for char in string:
        if not char.isupper():
            output+=char
    return output
```

In [7]:
```
lambda_strip = lambda x: x if x.isupper() else ''
```

Checkpoint Solution

nope!

In [8]:
```
example = 'AEIOUaeiou'
```

In [9]:
```
strip_caps(example)
```

Out[9]:
```
'aeiou'
```

In [14]:
```
''.join(list(map(lambda_strip, example)))
```

Out[14]:
```
'AEIOU'
```

Map

Map - **Syntax: map(function, iterable)**

- Map allows you to apply a function to all elements to an iterable input
- very common to use a lambda function as the function to apply
- returns an iterator through the iterable object, applying the function as it traverses

In [5]:
```
data = [1,2,3,4,5]
list(map(lambda x:x+2, data))
```

Out[5]:
```
[3, 4, 5, 6, 7]
```

Filter

Filter - **Syntax: filter(function, iterable)**

- Filter takes in a function that returns a boolean and only keeps elements that satisfy the function (i.e. return True).
- Very common to use a lambda function as the function to apply, but keep in mind the function **must return a boolean**.
- Returns an iterator through the iterable object that only yields values that pass the function.

In [7]:
```
data = [1,2,3,4,5]
list(filter(lambda x:x%2==0,data))
```

Out[7]:
```
[2, 4]
```

Checkpoint

Are the following 2 statements equivalent?

In [15]:
```
data = list(range(0,101))
```

In [ ]:
```
lambda_map = lambda x: x*2 if x%2==0 else 0
sum(map(lambda_map, data))
```

In [ ]:
```
lambda_filter = lambda x: x%2==0
sum(map(lambda_map, filter(lambda_filter, data)))
```

Checkpoint Solution

yep!

In [16]:
```
data = list(range(0,101))
```

In [21]:
```
lambda_map = lambda x: x*2 if x%2==0 else 0
sum(map(lambda_map, data))
```

Out[21]: 5100

In [22]:
```
lambda_filter = lambda x: x%2==0
sum(map(lambda_map, filter(lambda_filter, data)))
```

Out[22]: 5100

HOF

- Algorithm design framework to create generalized code
- Functions that either return functions or use other functions
- Helper functions!
- Many uses, including abstraction, scope protection, etc.

In [1]:

```
def summation_i(n):
    return (n*(n+1)) / 2

def summation_i2(n):
    return (n*(n+1)*(2*n+1)) / 6

def summation_formulas(n, form):
    if form=='i':
        return summation_i(n)
    if form=='i**2':
        return summation_i2(n)
    else:
        raise ValueError("unfacilitated i value")
```

HOF (cont.)

In [30]:
```
def add_1(x):
    return x + 1
def minus_1(x):
    return x - 1

def operate(op_type):
    if op_type == 'add':
        return add_1
    else:
        return minus_1
```

Checkpoint

Given the previous code, what is the result of these calls?

In [31]:
```
def add_1(x): # code from last slide
    return x + 1
def minus_1(x):
    return x - 1

def operate(op_type):
    if op_type == 'add':
        return add_1
    else:
        return minus_1
```

In [34]:
```
temp = operate('add')
```

In [ ]:
```
temp(1)
```

Checkpoint Solution

In [35]:
```
temp
```

Out[35]:
```
<function __main__.add_1(x)>
```

In [36]:
```
temp(1)
```

Out[36]:
```
2
```

*args

- Used when an unknown number of arguments will be passed into a function
- Denoted by * in the method header (IMPORTANT)
- processed in a similar manner to a list

In [10]:
```
def summation(*nums):
    return sum(nums)
print(summation())
print(summation(1,2,3,4,5))
```

```
0
15
```

Checkpoint

What is the result of this function call?

In [ ]:

```
def generate_names(*name_parts):
    output = []
    for name in name_parts:
        output.append(name*2)
    return output
generate_names('pika', 'niko', 'oro')
```

Checkpoint Solution

In [28]:
```
def generate_names(*name_parts):
    output = []
    for name in name_parts:
        output.append(name*2)
    return output
generate_names('pika', 'niko', 'oro')
```

Out[28]:
```
['pikapika', 'nikoniko', 'orooro']
```

**kwargs

- Used when an unknown number of **keyworded** arguments will be passed into a function
- Denoted by ** in the method header (IMPORTANT)
- processed in a similar manner to a dictionary

In [22]:
```
def create_dct(**entry):
    return dict(entry)
print(create_dct())
print(create_dct(marina=1, langlois=2))
```

```
{}
{'marina': 1, 'langlois': 2}
```

default_arguments

- Basically normal arguments, but with a default value
- if no value is passed, default value is set
- if a value is passed, default value is overwritten

In [23]:
```
def check_legal_age(age=18):
    return age>=21
print(check_legal_age())
print(check_legal_age(21))
```

```
False
True
```

Checkpoint

What is the result of this function call?

In [30]:
```
def filter_dict(t=2, **items_in):
    return {k:v for k,v in items_in.items() if len(v)>t}
filter_dict(0, temp=[1,2], test=[3,4,5])
```

Checkpoint Solution

In [33]:
```
def filter_dict(t=2, **items_in):
    return {k:v for k,v in items_in.items() if len(v)>t}
filter_dict(0, temp=[1,2], test=[3,4,5])
```

Out[33]:
```
{'temp': [1, 2], 'test': [3, 4, 5]}
```

note

complex argument ordering gets really messy

In [40]:
```
def func(norm, *args, darg=2, **kwargs):
    return [norm, list(args), darg, dict(kwargs)]
func(42,1,1,1,1,1,1,3,darg=4, test=1)
```

Out[40]:
```
[42, [1, 1, 1, 1, 1, 1, 3], 4, {'test': 1}]
```

In [2]:
```
def func(norm, darg=2, *args, **kwargs):
    return [norm, list(args), darg, dict(kwargs)]
func(42,4,1,1,1,1,1,1,3,test=1)
```

Out[2]:
```
[42, [1, 1, 1, 1, 1, 1, 3], 4, {'test': 1}]
```

practice questions

Time to do some practice questions! Take about 10-15 minutes to work on the questions. Feel free to flag me down if you need help/clarification.

If you finish early, feel free to head over to gradescope and complete the discussion attendance assignment

practice question solutions

Write 2 functions, one to calculate the median and another to calculate the spread of a set of numbers.

In [25]:

```
def median(vals):
    length = len(vals)
    data = sorted(vals)
    if length%2==0:
        return (data[(length-1)//2] + data[length//2])/2
    return data[length//2]

def spread(vals):
    return max(vals) - min(vals)
```

In [29]:

```
def summary_stat(operation):
    """
    Write a function that takes in a specified operation
    and returns a function that will take in a set of
    numbers and calculate the operation accordingly.

    possible operations:
        min -> finds the minimum value
        max -> finds the maximum value
        range -> finds the range of the values
        median -> finds the median of the values

    >>> med = summary_stat('median')
    >>> med([1,2,3,4,5,6])
    3.5
    >>> ran = summary_stat('range')
    >>> ran([1,2,3,4,5,6])
    5
    """
    if operation=='min':
        return min
    if operation=='max':
        return max
    if operation=='median':
        return median
    else:
        return spread
print(summary_stat('median')([1,2,3,4,5,6]))
print(summary_stat('range')([1,2,3,4,5,6]))
```

3.5
5

In [10]:

```
def count_len_lsts(*lists, counter=4):
    """
    Write a function that takes in an unknown
    number of lists and returns the sum of the
    length of the first 'counter' lists, default
    value of 4.

    Args:
        lists(args): unknown number of lists
        counter(int): number of lists length to count
    Returns:
        sum of the lengths of the first counter lists

    >>> count_len_lsts([],[1],[1],[1])
    3
    >>> count_len_lsts([],[],[1,2,3],[4,5], counter=2)
    0
    """
    return sum([len(x) for x in lists[:counter]])
print(count_len_lsts([],[1],[1],[1]))
print(count_len_lsts([],[],[1,2,3],[4,5], counter=2))
```

```
3
0
```

In [4]:
```
def query_data(database, source, quality):
    """
    Write a function that takes in a dictionary
    and returns a list of items from source that
    are at least of quality level.

    >>> data = [{'name':'a', 'quality':4, 'source':'dsc'},
    {'name':'b','quality':10, 'source':'lign'},
    {'name':'c','quality':2, 'source':'dsc'},
    {'name':'d','quality':5, 'source':'dsc'}]
    >>> query_data(data, 'dsc', 4)
    ['a','d']
    """
    data_check = lambda x: x['source']==source and x['quality']>=quali
    filtered = filter(data_check, database)
    data_yield = lambda x:x['name']
    return list(map(data_yield, filtered))
query_data(data, 'dsc', 4)
```

Out[4]:
```
['a', 'd']
```

Discussion Attendance

Take 2 minutes and head to gradescope to complete discussion attendance. The assignment is called Discussion 5 Participation.

Machine Learning

Preface

Recall from discussion 1:

"Why bother learning how to code? So that we can do cool things with the tools that people have invented!"

The start of those cool things is always foundational machine learning - solving problems and finding answers through code that can "learn".

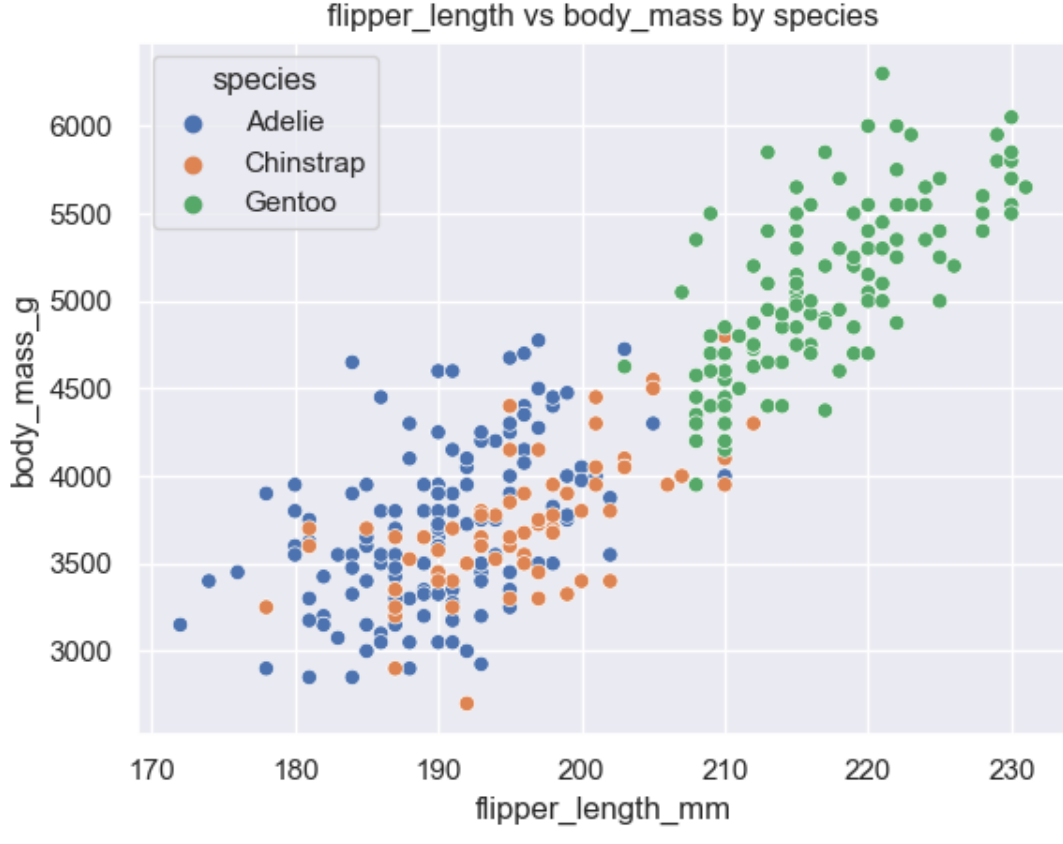
Learning always starts with data. Without having some "ground truth" to base your learning on, whatever you learn is ultimately meaningless. For today's discussion, we'll take a look at a hallmark dataset for data science, penguins:

To simplify the problem, we will not be using all features. Instead, we will only consider **'body_mass_g', 'flipper_length', and 'species'**.

	flipper_length_mm	body_mass_g	species
0	181.0	3750.0	Adelie
1	186.0	3800.0	Adelie
2	195.0	3250.0	Adelie
4	193.0	3450.0	Adelie
5	190.0	3650.0	Adelie

Classifiers

Classification is one of the major tasks in machine learning. Its goal (as its name implies) is to ingest data associated with a label and be able to predict the label of future unseen data by learning some sort of pattern. Within our penguins dataset, this would involve predicting the penguin species ('species') based on its 'flipper_length_mm' and 'body_mass_g'.



In [35]:

```
clf.fit(X_train, y_train);
training_accuracy = (clf.predict(X_train) == y_train).mean()
testing_accuracy = (clf.predict(X_test) == y_test).mean()
```

```
training accuracy: 0.745
testing accuracy: 0.6940298507462687
```

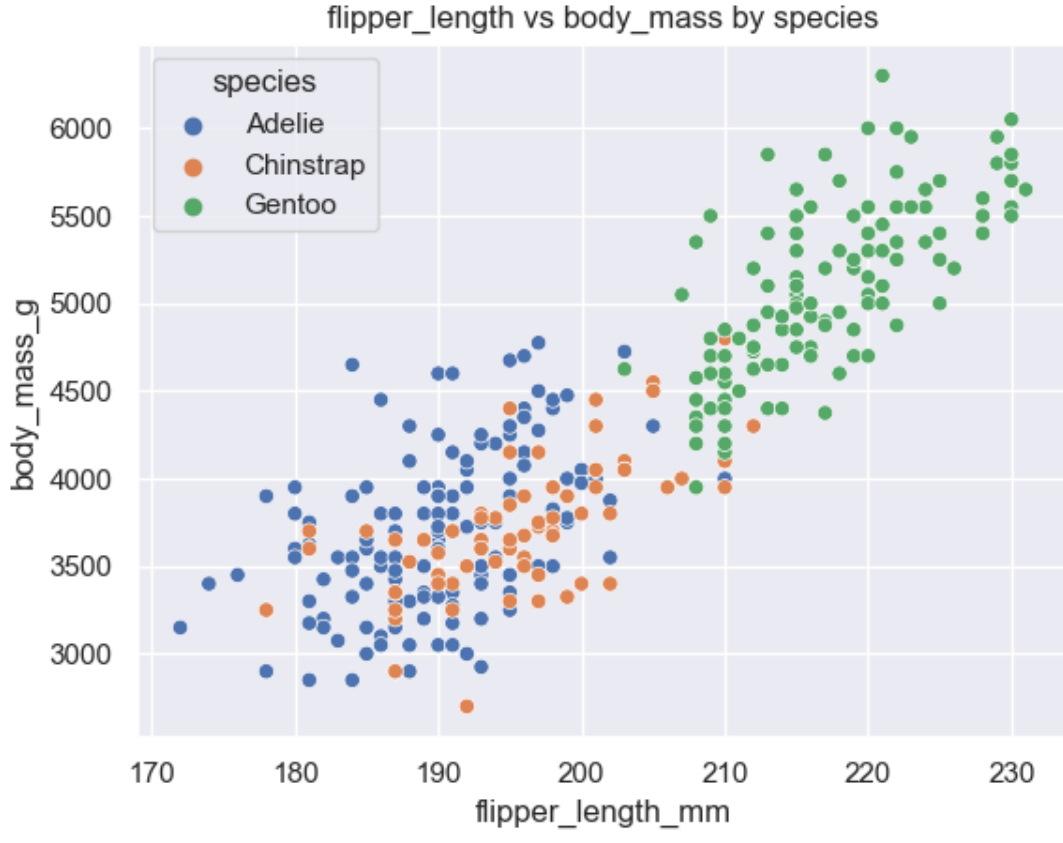
Why are we talking about this?

Foreshadowing! You may (or may not) need to apply some of this knowledge (relatively) soon! Possibly for your project

K Nearest Neighbors (KNN) for Classification

idea: Given an unseen value of flipper_length and body_mass, how can we determine what species of penguins it belongs to?

The KNN approach is simple - if I look at the k nearest points to this new value, and I take the most common species among them, then this point is **most likely** the same species as the most common species among the neighbors.


flipper_length vs body_mass by species
species
Adelie
Chinstrap
Gentoo
body_mass_g
6000
5500
5000
4500
4000
3500
3000
170
180
190
200
210
220
230
flipper_length_mm

Procedure

Step 1: Given a new point *X*, quantify the distance between all points and *X*

Step 2: Take the **k-nearest** points to *X* into consideration

Step 3: Classify *X* as the most common label among the neighbors

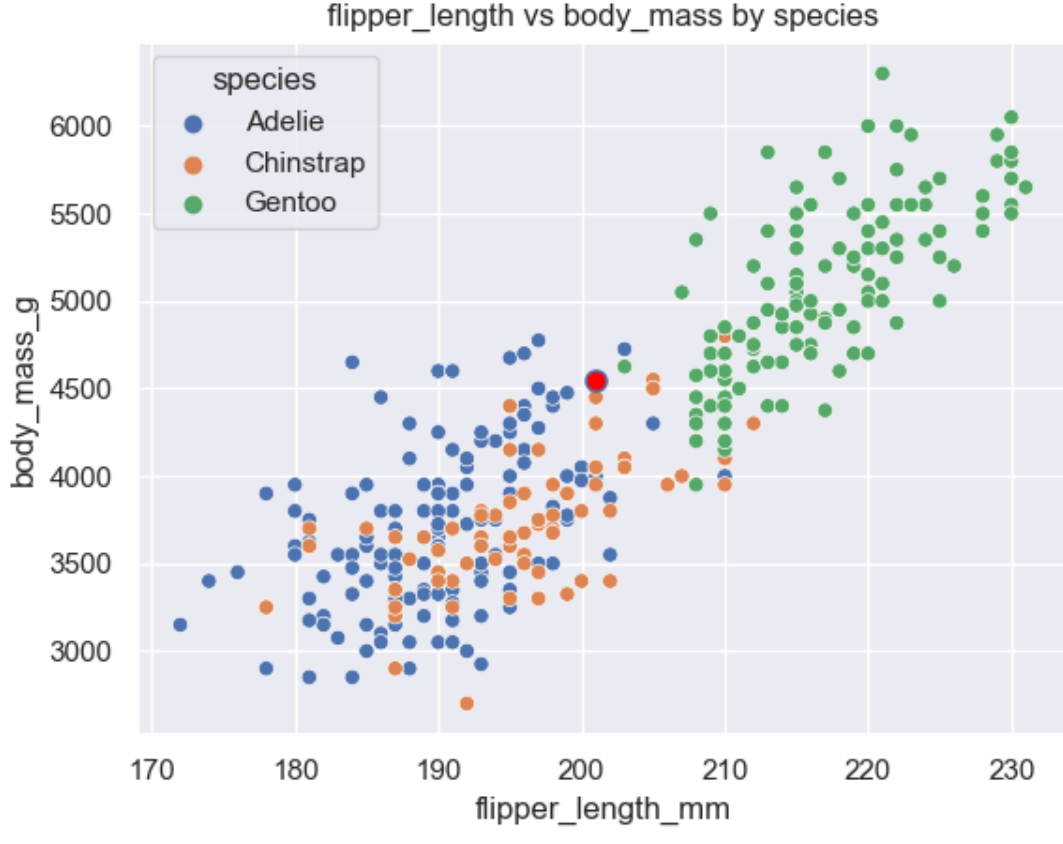
note: As the name implies, the key parts of this algorithm are **k** and **nearest**.

k is often referred to as a **hyper-parameter**, or a value that you choose independently. There are often procedures to select a good k, but this varies depending on the context.

Nearest refers to quantifying distance - one such way is to use euclidian distance (distance formula), but there are many other "distances" that can be used (ex. Manhattan Distance).

Checkpoint

Given the new data point (201, 4750), what would a KNN classifier classify the new point as for **k=1**? What about for **k=4**? What about for **k=10**?



Checkpoint Solution

for k=1: Seems like the orange point is closest according to an eye test, so Chinstrap

for k=4: The 3 closest points seem to all be from unique colors, the fourth point is hard to determine by eye. Could easily be Adelie or Chinstrap

for k=10: The bulk of the points nearby are blue, so Adelie

Thanks for coming!