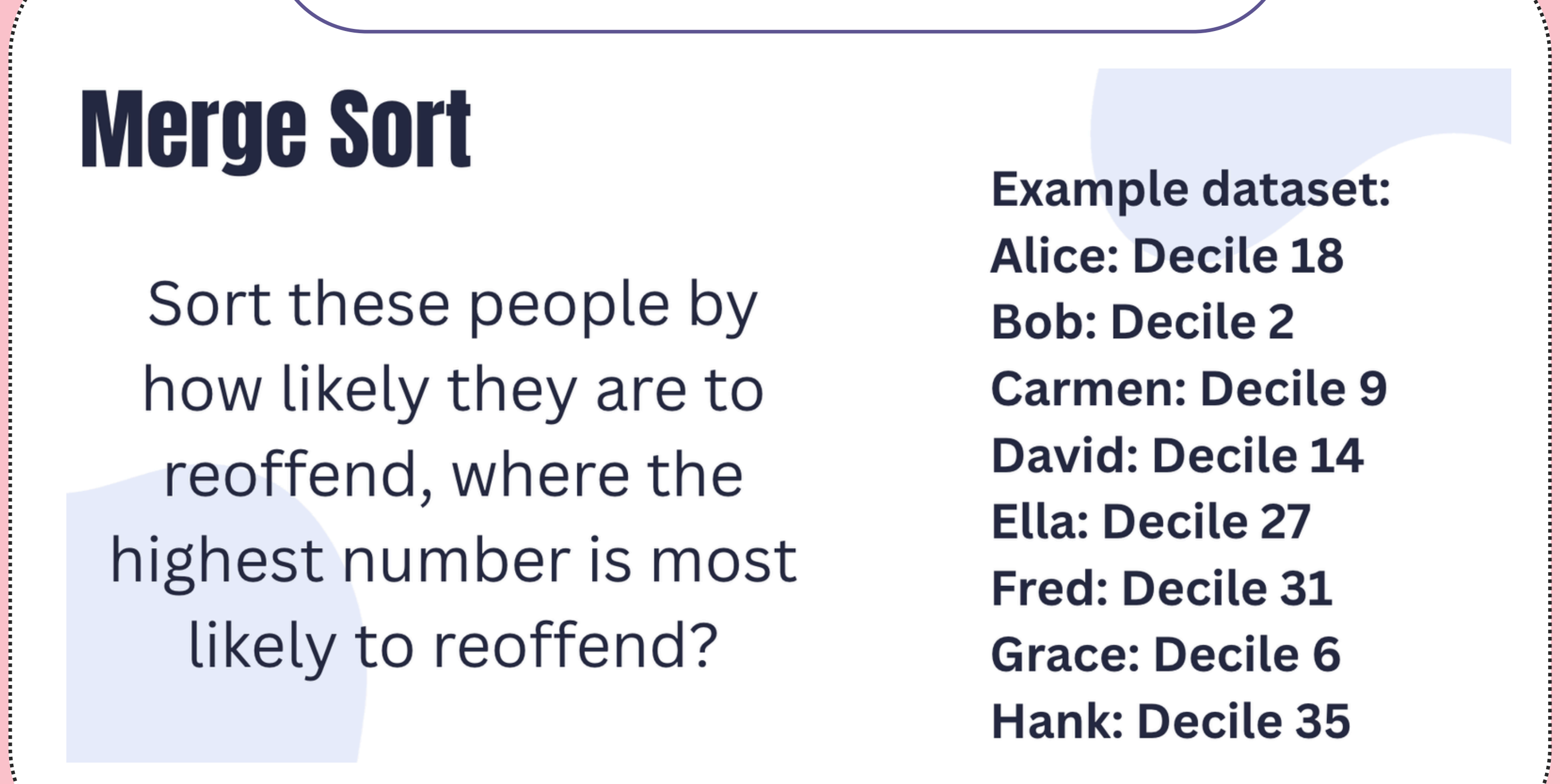
ALGORITHMS IN REAL LIFE:
WHERE THEORY MEETS THE REAL WORLD

Auiannce Euwing. Teammates: Zubeyda Shute, Emma Zhang, Chloe Xu. Advisor: Jean Salac

OVERVIEW

- How algorithmic choices can reflect or reinforce bias (e.g., risk prediction tools like COMPAS).
- How algorithms learned in class can be used in the real world

IN-CLASS SLIDES



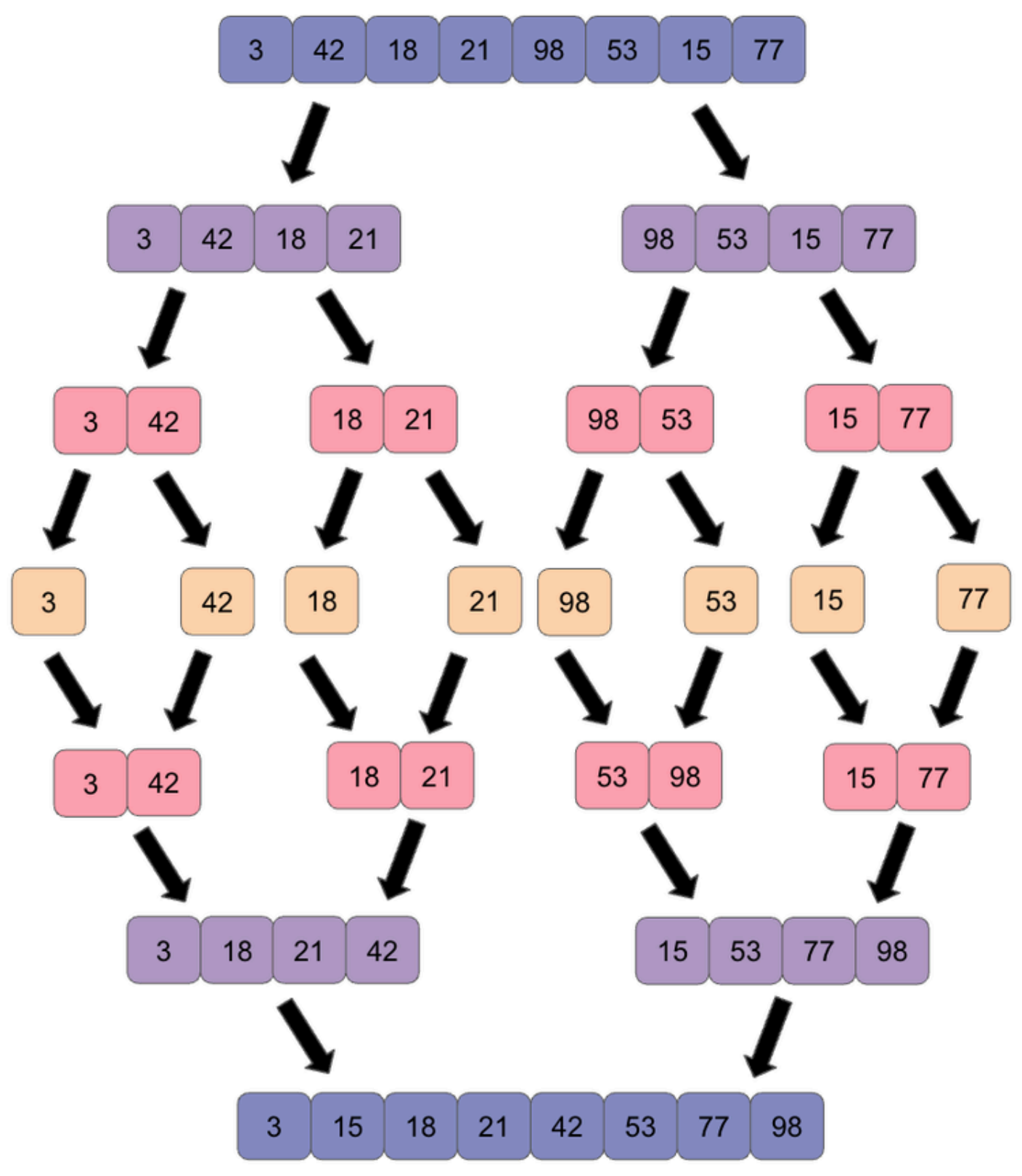
Merge Sort

Sort these people by how likely they are to reoffend, where the highest number is most likely to reoffend?

Example dataset:
Alice: Decile 18
Bob: Decile 2
Carmen: Decile 9
David: Decile 14
Ella: Decile 27
Fred: Decile 31
Grace: Decile 6
Hank: Decile 35

IN-CLASS ASSIGNMENT

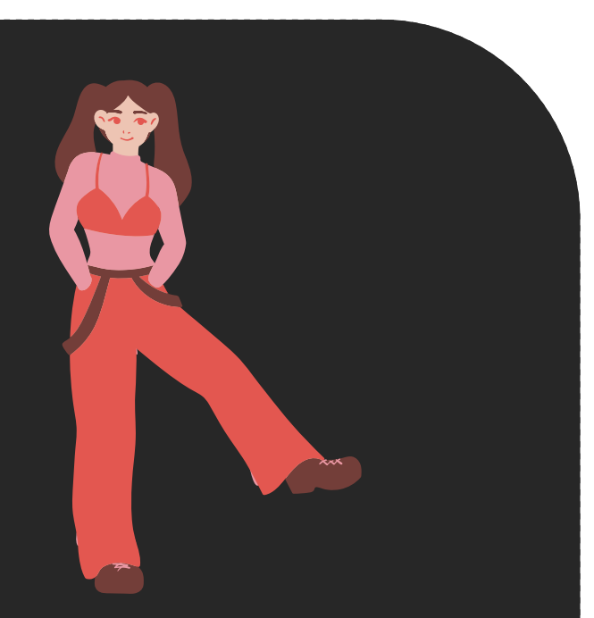
- Introduce the students to Merge Sort and Binary Search
- Explain what a decile score is
- Give an example of how one could use the Merge to order the people by how likely they are to reoffend and also search for a person who has a certain decile score.



HOMEWORK ASSIGNMENT

- Students create a new decile score using a formula given to them
- Students compare the order of the names from the decile score that the COMPAS algorithm gave and the decile scores they created using the formula.

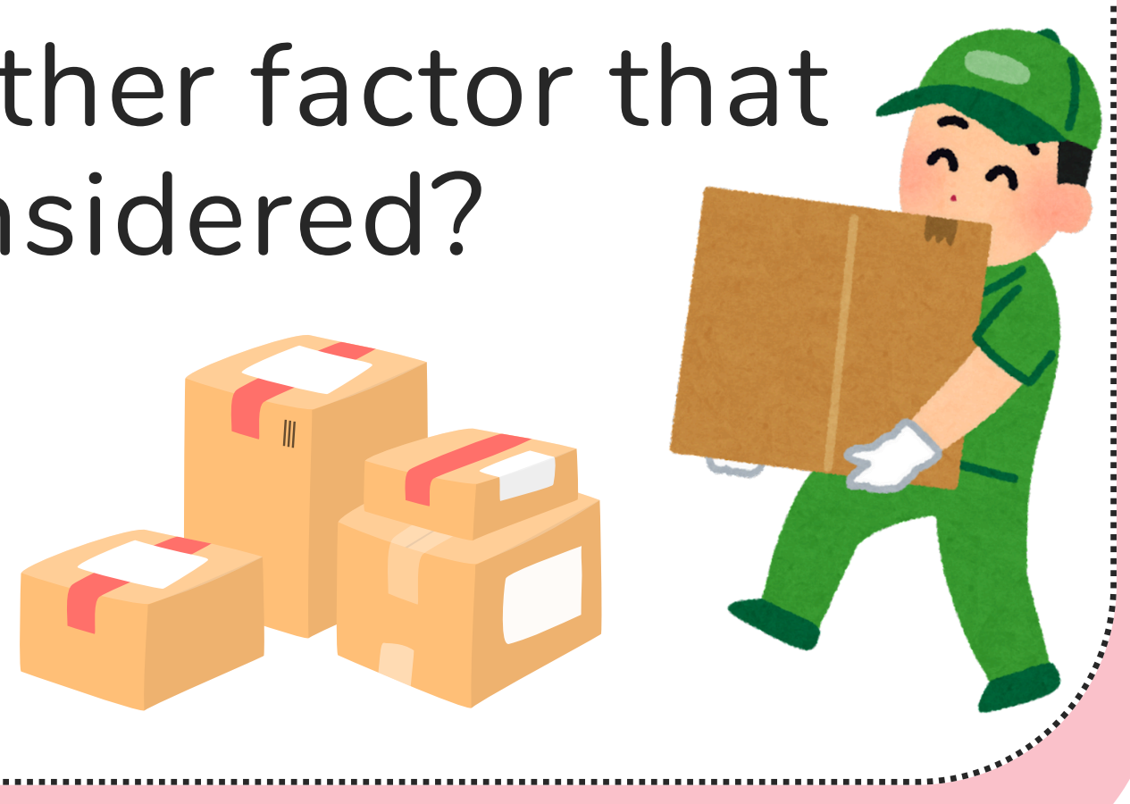
Race: Black
Days in jail: 9
Days in Custody: 1
Priors: 0
Compas Decile: 10



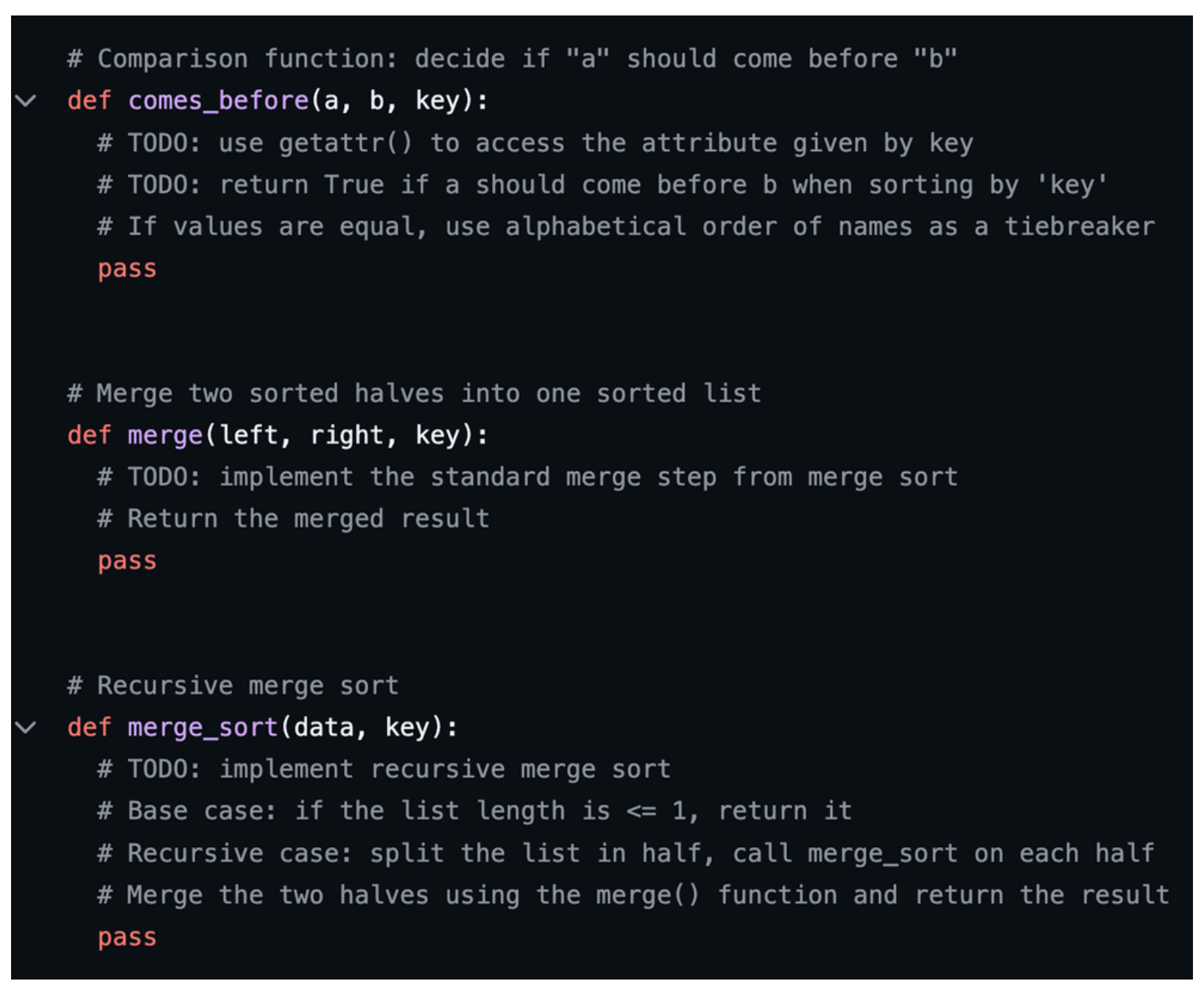
Race: White
Days in jail: 31
Days in Custody: 10
Priors: 9
Compas Decile: 2

GREEDY ALGORITHM SOLUTIONS

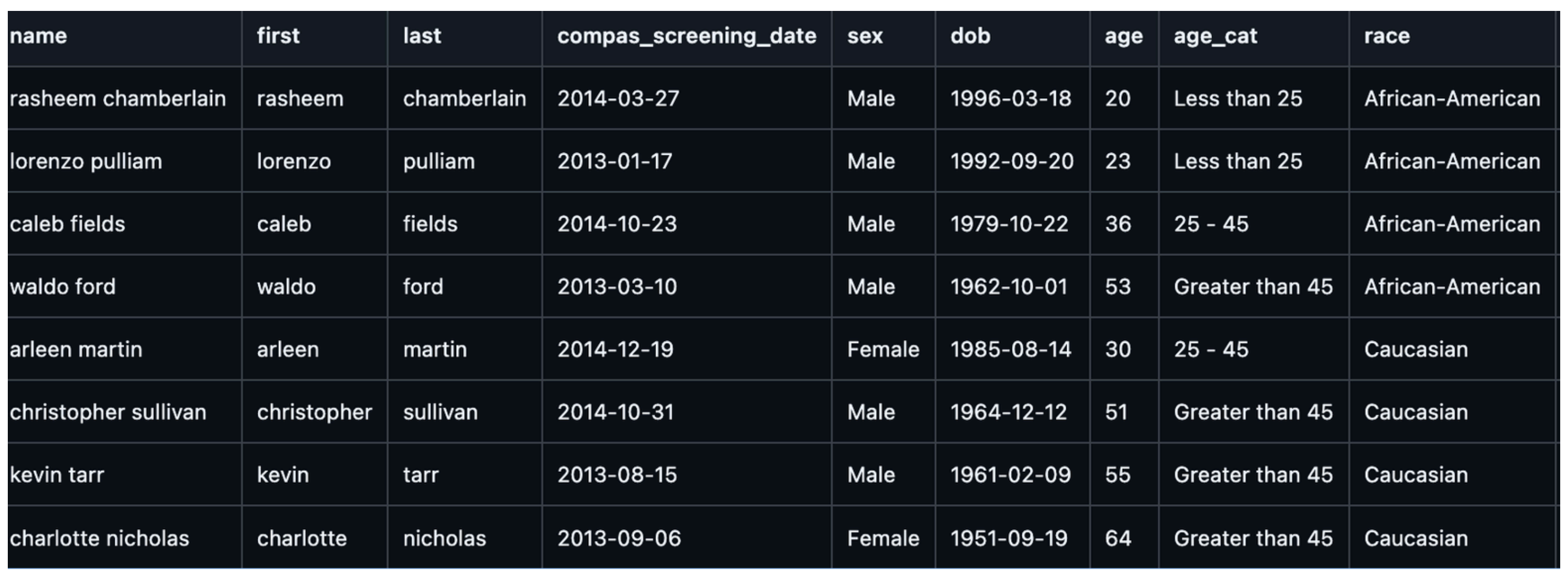
- Who are the stakeholders in each version of the algorithm, and how would the Greedy algorithm change based on the stakeholders
  - Part A: Company
  - Part B: Driver
- What is another factor that could be considered?
  - Fatigue



HOMEWORK PSEUDOCODE

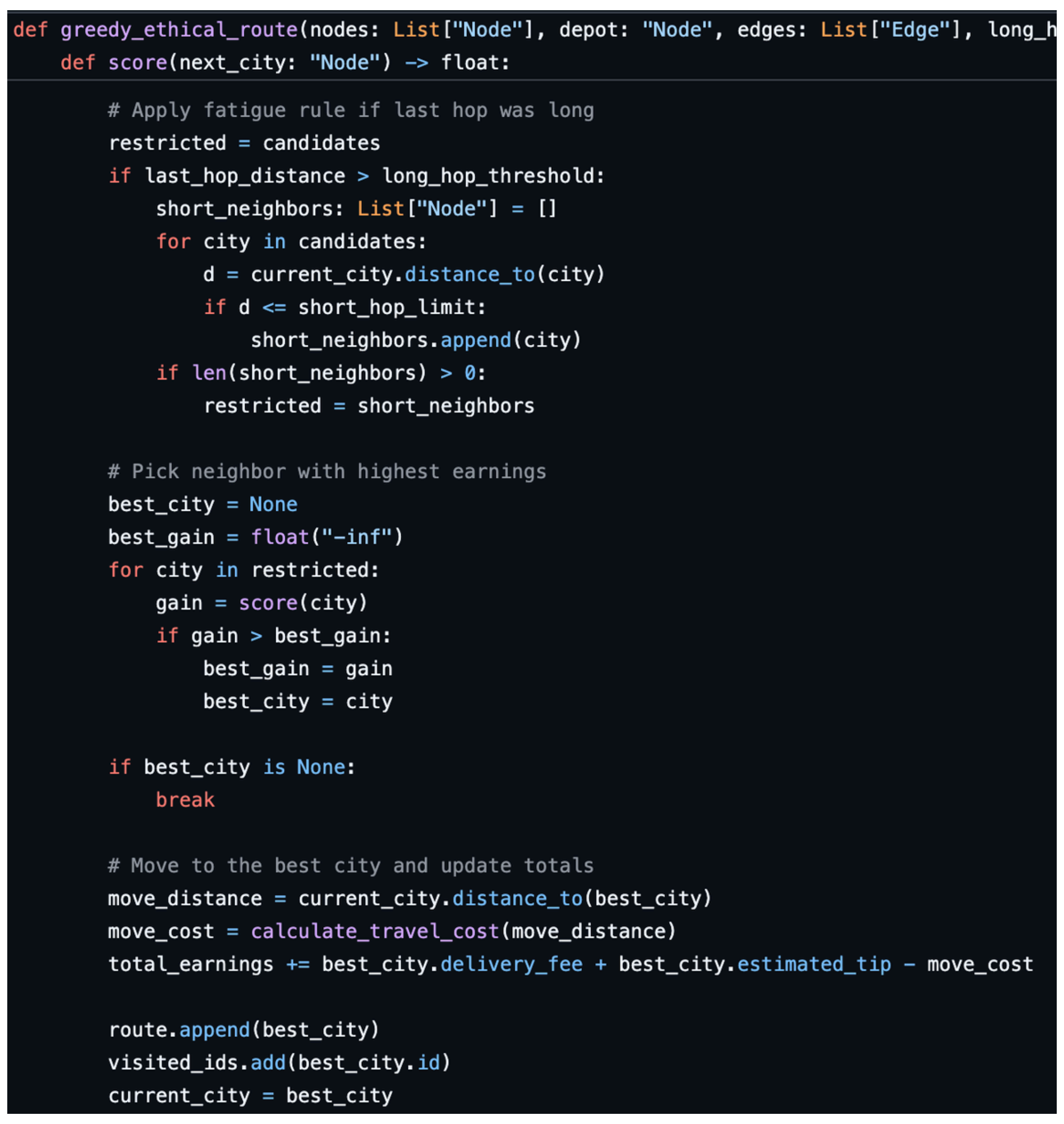


```
# Comparison function: decide if "a" should come before "b"
def comes_before(a, b, key):
  # TODO: use getattr() to access the attribute given by key
  # TODO: return True if a should come before b when sorting by 'key'
  # If values are equal, use alphabetical order of names as a tiebreaker
  pass


# Merge two sorted halves into one sorted list
def merge(left, right, key):
  # TODO: implement the standard merge step from merge sort
  # Return the merged result
  pass


# Recursive merge sort
def merge_sort(data, key):
  # TODO: implement recursive merge sort
  # Base case: if the list length is <= 1, return it
  # Recursive case: split the list in half, call merge_sort on each half
  # Merge the two halves using the merge() function and return the result
  pass
```



name	first	last	compas_screening_date	sex	dob	age	age_cat	race
rasheem chamberlain	rasheem	chamberlain	2014-03-27	Male	1996-03-18	20	Less than 25	African-American
lorenzo pulliam	lorenzo	pulliam	2013-01-17	Male	1992-09-20	23	Less than 25	African-American
caleb fields	caleb	fields	2014-10-23	Male	1979-10-22	36	25 - 45	African-American
waldo ford	waldo	ford	2013-03-10	Male	1962-10-01	53	Greater than 45	African-American
arleen martin	arleen	martin	2014-12-19	Female	1985-08-14	30	25 - 45	Caucasian
christopher sullivan	christopher	sullivan	2014-10-31	Male	1964-12-12	51	Greater than 45	Caucasian
kevin tarr	kevin	tarr	2013-08-15	Male	1961-02-09	55	Greater than 45	Caucasian
charlotte nicholas	charlotte	nicholas	2013-09-06	Female	1951-09-19	64	Greater than 45	Caucasian

SOULTION CODE



```
def greedy_ethical_route(nodes: List["Node"], depot: "Node", edges: List["Edge"], long_h
    def score(next_city: "Node") -> float:

        # Apply fatigue rule if last hop was long
        restricted = candidates
        if last_hop_distance > long_hop_threshold:
            short_neighbors: List["Node"] = []
            for city in candidates:
                d = current_city.distance_to(city)
                if d <= short_hop_limit:
                    short_neighbors.append(city)
            if len(short_neighbors) > 0:
                restricted = short_neighbors

        # Pick neighbor with highest earnings
        best_city = None
        best_gain = float("-inf")
        for city in restricted:
            gain = score(city)
            if gain > best_gain:
                best_gain = gain
                best_city = city

        if best_city is None:
            break

        # Move to the best city and update totals
        move_distance = current_city.distance_to(best_city)
        move_cost = calculate_travel_cost(move_distance)
        total_earnings += best_city.delivery_fee + best_city.estimated_tip - move_cost

        route.append(best_city)
        visited_ids.add(best_city.id)
        current_city = best_city
```

TAKEAWAYS/REFLECTION

- The lessons in the assignment take random numbers and giving them meaning my associationing them with a name and a value
- Student began to think about algorithms in a larger space

SOURCES

- Nachbar, Algorithmic Fairness, Algorithmic Discrimination
  - Argues fairness is not a number to optimize — it's a moral constraint.
- ProPublica, "How We Analyzed the COMPAS Recidivism Algorithm"
  - Showed how COMPAS misclassified Black and white defendants at different rates.
  - Demonstrated that data and algorithms can encode social bias.