CS134:
Dictionaries

Announcements & Logistics

- **Lab 6 Posted**
  - No pre-lab question, but relies on material covered Wednesday before spring break (Files) and today (Dictionaries)
  - Be sure to read through the way the data is organized before lab
    - We can go over the "**Organizing the Data**" section if you have Q's
- **Midterm will be returned on Wednesday**
  - Mostly graded, but a few up loose ends to tie up before we can return it to everyone

Do You Have Any Questions?

Last Time: Files and Plotting

- **Data science-y things:**
  - **File reading:** Files are persistent data, usable between sessions and applications!
    - Comma-Separated Values Files are a common format for data
  - Gave a template for plotting with **matplotlib**
    - matplotlib is a plotting API that we will *use* in Lab
    - you should be able to pattern match from the examples, but please feel free to refer to any documentation that would be helpful.
      - Note: Googling is OK **for matplotlib-related questions** (not OK for the computational thinking parts of the lab---that is where the computer science comes into play)

Today

- Discuss a new data structure: **dictionary**
  - "**Unordered**" and **mutable** collection, just like sets
- Dictionaries are one of the most widely use ways to organize our data in "real world" applications
  - For many problems, dictionaries are often the most efficient (i.e., fast) and most natural way to represent the relationships among data

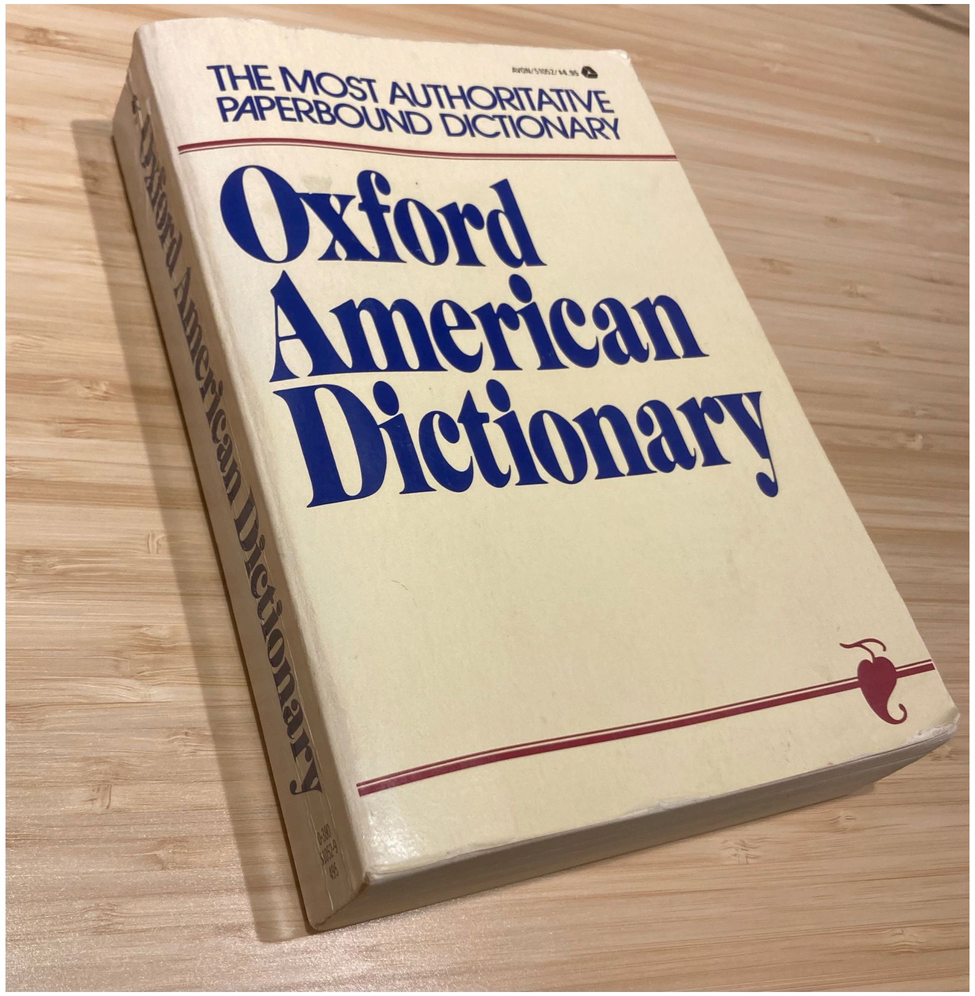
Dictionaries

Sequences vs Unordered Collections

- Sequence: a group of items that come one after the other (there is an implicit ***ordering*** of items)
  - Sequences in Python: strings, lists, ranges
- Unordered Collection: a group of things bundled together for a reason but without a specific ordering
- For some use cases, it is better to store an unordered collection
  - Maintaining order between items is not always necessary
  - Ordering items comes at a cost in terms of efficiency!
- Python has two data structures which are **unordered**:
  - **Dictionaries** and **sets:** both of them are **mutable**
  - We will discuss **dictionaries** today

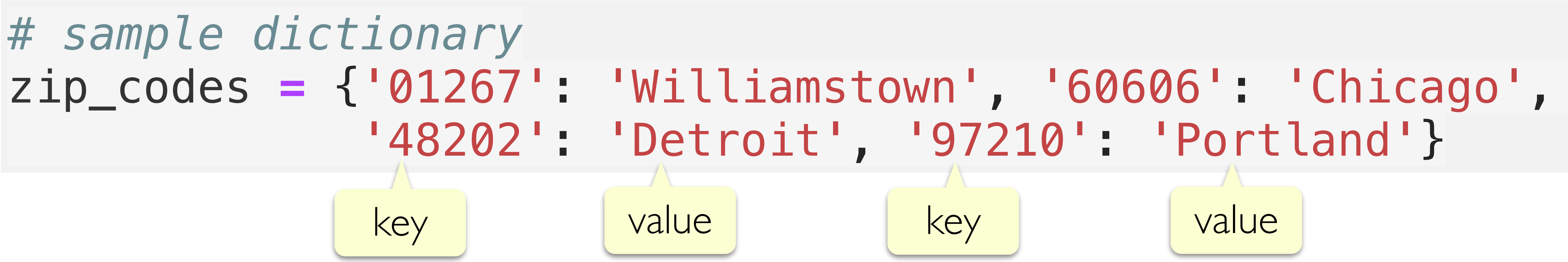
Language Dictionaries

- What does an English dictionary store?



Python Dictionaries

- A Python **dictionary** is a **mutable** collection that maps **keys** to **values**
  - Enclosed with curly brackets, and contains **comma-separated** items
  - Each item in the dictionary is a **colon-separated** ***key-value pair***
  - There is no ordering among the keys of a dictionary!



```
# sample dictionary
zip_codes = {'01267': 'Williamstown', '60606': 'Chicago',
             '48202': 'Detroit', '97210': 'Portland'}
```

- **Keys** must be an **immutable** type such as ints, strings, or tuples
  - Keys of a dictionary must also be **unique**: no duplicates allowed!
- **Values** can any Python type (ints, strings, lists, tuples, etc.)

Accessing Items in a Dictionary

- Dictionaries are **unordered** so we cannot access them by index: no notion of first or second item, etc.
- We instead lookup **`values`** in a dictionary using the corresponding **`keys`** as the subscript in `[]` notation
  - If the key exists, its corresponding value is returned
  - If the key is missing, the lookup produces a **`KeyError`**

```
>>> zip_codes = {"01267": "Williamstown", "60606": "Chicago"
                  "48202": "Detroit", "97210": "Portland"}
>>> # what US city has this zip code?
>>> zip_codes["60606"]
'Chicago'
```

key

value

value associated with key '60606'

```
>>> # what US city has this zip code?
>>> zip_codes["48202"]
'Detroit'
```

value associated with key ‘48202'

Adding a Key, Value Pair

- Dictionaries are **mutable**, so we can add, remove, and update items
- To add a new **key-value** pair, we can simply assign the key to the value using: `dict_name[key] = value`

```
>>> zip_codes["11777"] = "Port Jefferson"
>>> zip_codes
```

Add key, value pair `'11777': 'Port Jefferson'`

```
{'01267': 'Williamstown',
 '60606': 'Chicago',
 '48202': 'Detroit',
 '97210': 'Portland',
 '11777': 'Port Jefferson'}
```

- If the key already exists, an assignment operation as above will ***overwrite*** its value and associate the key with the new value

Adding a Key, Value Pair

- Dictionaries are **mutable**, so we can add items or remove items from it
- To add a new **key, value** pair, we can simply assign the key to the value using: `dict_name[key] = value`

```
    >>> zip_codes["11777"] = "Port Jefferson"
    >>> zip_codes
```

Add key, value pair `'11777': 'Port Jefferson'`

```
    {'01267': 'Williamstown',
     '60606': 'Chicago',
     '48202': 'Detroit',
     '97210': 'Portland',
     '11777': 'Port Jefferson'}
    >>> zip_codes["01267"] = "Billsville"
    >>> zip_codes
{'01267': 'Billsville', '60606': 'Chicago', '48202':
'Detroit', '97210': 'Portland', '11777': 'Port Jefferson'}
```

Operations on Dictionaries

- Just like sequences, we can use the `len()` function on dictionaries to find out the **number of keys** it contains
- To check if a **key** exists or does not exist in a dictionary, we can use the `in` or `not in` operator,' respectively

```
>>> zip_codes
{'01267': 'Williamstown',
 '60606': 'Chicago',
 '48202': 'Detroit',
 '97210': 'Portland',
 '11777': 'Port Jefferson'}
>>> len(zip_codes)
5
```

```
>>> "90210" in zip_codes
False
>>> "01267" in zip_codes
True
```

Should always check if a key exists before accessing its value in a dictionary

```
>>> "Williamstown" in zip_codes
False
```

`in` only checks the keys, not values!

Creating Dictionaries

- Direct assignment: provide key, value pairs delimited with { }
- Start with empty dict and add key, value pairs
  - Empty dict is `{}` or `dict()`
- Apply the built-in function `dict()` to a list of paired items

```
# direct assignment
scrabble_score ={'a':1, 'b':3, 'c':3, 'd':2, 'e':1,
                 'f':4, 'g':2, 'h':4, 'i':1, 'j':8,
                 'k':5, 'l':1, 'm':3, 'n':1, 'o':1,
                 'p':3, 'q':10,'r':1, 's':1, 't':1,
                 'u':1, 'v':8, 'w':4, 'x':8, 'y':4, 'z': 10}
```

Note: keys may be listed in any order, since dictionaries are unordered

Creating Dictionaries

- Direct assignment: provide key, value pairs delimited with { }
- Start with empty dict and add key, value pairs
  - Empty dict is `{}` or `dict()`
- Apply the built-in function `dict()` to a list of paired items

```
# accumulate in a dictionary
verse = "let it be,let it be,let it be,let it be,there will be an answer,let it be"
counts = {} # empty dictionary
for line in split(verse, ','):
    if line not in counts:
        counts[line] = 1 # initialize count
    else:
        counts[line] += 1 # update count
print(counts)
```

```
>>> counts
{'let it be': 5, 'there will be an answer': 1}
# use dict() function
>>> dict([['a', 5], ['b', 7], ['c', 10]])
{'a': 5, 'b': 7, 'c': 10}
```

Note: keys may be listed in any order

Example: Frequency

Example: `frequency`

- One common use of a dictionary is to store **frequencies**.
- Let's write a function `frequency()` that takes as input a list of strings `word_lst` and returns a dictionary `freq_dict` with the unique strings in `word_lst` as keys, and their number of occurrences (ints) in `word_lst` as values
- For example if `word_lst` is:

```
['hello', 'world', 'hello', 'earth', 'hello', 'earth']
```

the function should return a dictionary with the following items:

```
{'hello': 3, 'world': 1, 'earth': 2}
```

Example: `frequency`

- Let's write a function `frequency()` that takes as input a list of strings `word_lst` and returns a dictionary `freq_dict` with the unique strings in `word_lst` as keys, and their number of occurrences (ints) in `word_lst` as values
- How can we do this?

Example: `frequency`

- Let's write a function `frequency()` that takes as input a list of strings `word_lst` and returns a dictionary `freq_dict` with the unique strings in `word_lst` as keys, and their number of occurrences (ints) in `word_lst` as values
- Pseudocode:

```
# for each word in our word_lst:
    # if the word isn't already in our freq_dict, then add with count of 1
    # otherwise, update the count
# return freq_dict when done
```

Example: `frequency`

- Let's write a function `frequency()` that takes as input a list of strings `word_lst` and returns a dictionary `freq_dict` with the unique strings in `word_lst` as keys, and their number of occurrences (ints) in `word_lst` as values

```python
def frequency(word_lst):
    """Given a list of words, returns a dictionary
    of word frequencies"""
    freq_dict = {} # initialize accumulator as empty dict
    for word in word_lst:
        if word not in freq_dict:
            freq_dict[word] = 1 # add key with count 1
        else:
            freq_dict[word] += 1 # update count
    return freq_dict
```

Example:
Data Analysis w
Dictionaries of Dictionaries

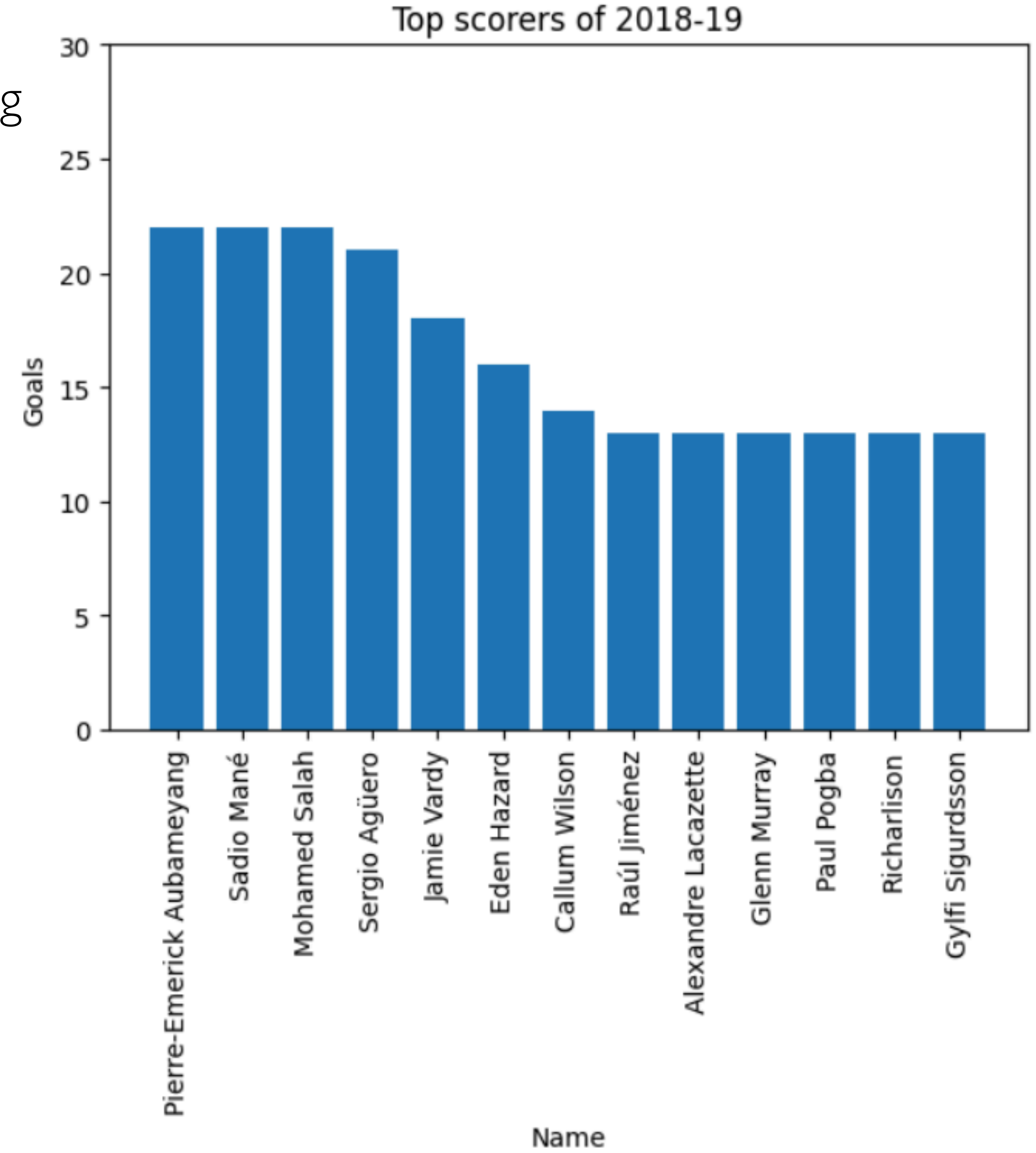
Exercise: Python code

You are a talent scout for an English football (soccer) club. The club you work for has a good defense, but a weak offense. So, you've been tasked with identifying a star striker to help score more goals!

So you decide to identify candidates in a data-driven manner.

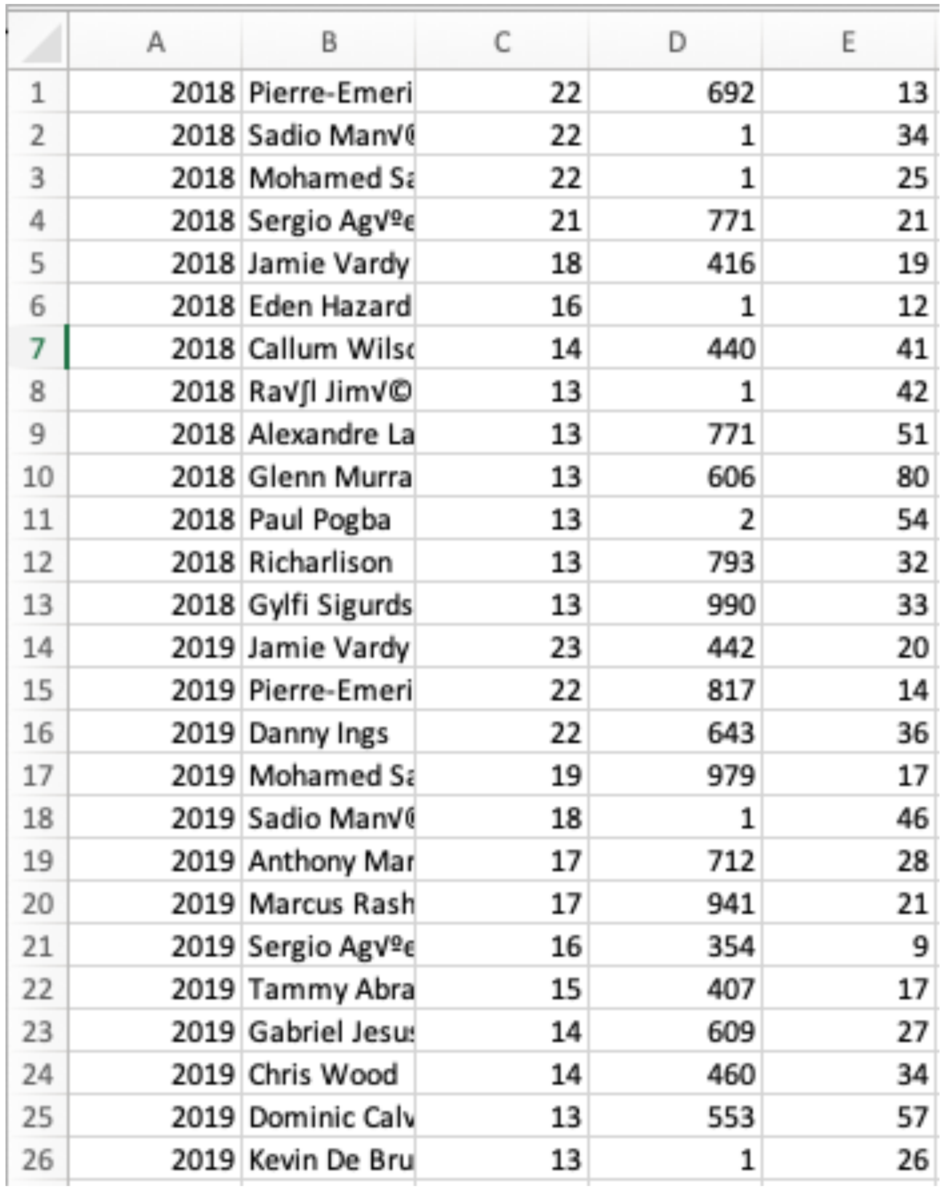
What we're aiming to produce

- We will plot bar charts showing the most frequent goal scorers in various years, and use them to determine who to try and recruit to our team



Reading-in Data from a File to Dict

- First, let's take a look at our data, `seasons2018-2022.csv`
- In a spreadsheet viewer, it looks like the screenshot on the left
- However, we'll be reading-in the data with python, so it will look more like the text on the right:



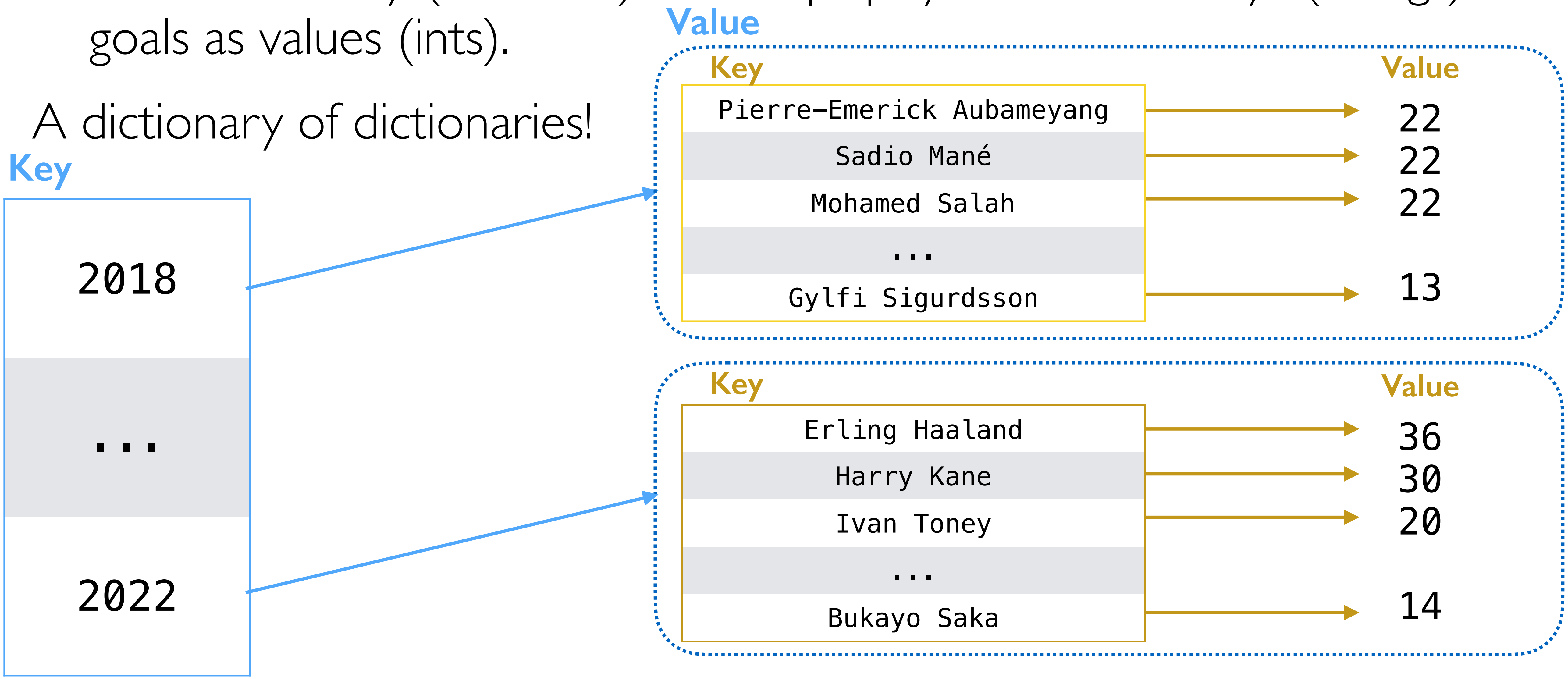
	A	B	C	D	E
1	2018	Pierre-Emeri	22	692	13
2	2018	Sadio Manv©	22	1	34
3	2018	Mohamed Sa	22	1	25
4	2018	Sergio Agvºe	21	771	21
5	2018	Jamie Vardy	18	416	19
6	2018	Eden Hazard	16	1	12
7	2018	Callum Wilso	14	440	41
8	2018	Rav∫l Jimv©	13	1	42
9	2018	Alexandre La	13	771	51
10	2018	Glenn Murra	13	606	80
11	2018	Paul Pogba	13	2	54
12	2018	Richarlison	13	793	32
13	2018	Gylfi Sigurds	13	990	33
14	2019	Jamie Vardy	23	442	20
15	2019	Pierre-Emeri	22	817	14
16	2019	Danny Ings	22	643	36
17	2019	Mohamed Sa	19	979	17
18	2019	Sadio Manv©	18	1	46
19	2019	Anthony Mar	17	712	28
20	2019	Marcus Rash	17	941	21
21	2019	Sergio Agvºe	16	354	9
22	2019	Tammy Abra	15	407	17
23	2019	Gabriel Jesus	14	609	27
24	2019	Chris Wood	14	460	34
25	2019	Dominic Calv	13	553	57
26	2019	Kevin De Bru	13	1	26

```
2018,Pierre-Emerick Aubameyang,22,692,13
2018,Sadio Mané,22,1,34
2018,Mohamed Salah,22,1,25
2018,Sergio Agüero,21,771,21
2018,Jamie Vardy,18,416,19
2018,Eden Hazard,16,1,12
2018,Callum Wilson,14,440,41
2018,Raúl Jiménez,13,1,42
2018,Alexandre Lacazette,13,771,51
2018,Glenn Murray,13,606,80
```

season,name,goals,passes,fouls

Reading-in Data from a File to Dict

- Need to write a function that reads-in this file and creates a data structure for plotting
- Want performance across seasons, names, and goals scored.
    - Outer dictionary, **`season_table`**: maps season as keys (ints) to an inner dictionary (as values) that maps player names as keys (strings) to goals as values (ints).
- A dictionary of dictionaries!



Reading-in Data from a File to Dict

- Iterate over lines, after we've parsed them...
  - dictionary stuff!

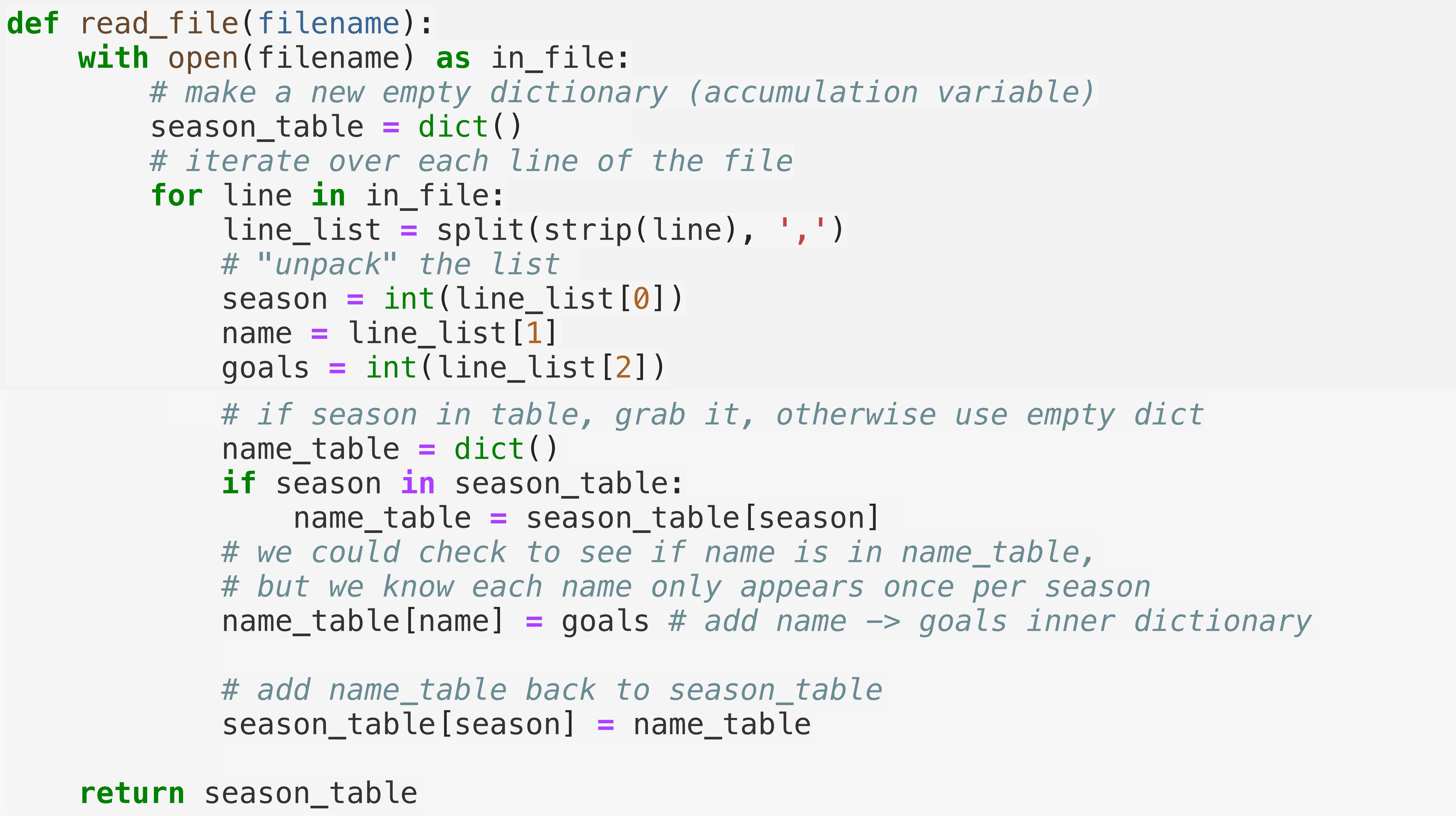
```
def read_file(filename):
    with open(filename) as in_file:


        # iterate over each line of the file
        for line in in_file:
            # remove extra newline at end
            line = strip(line)
            line_list = split(line, ',')
            # "unpack" the list
            season = int(line_list[0])
            name = line_list[1]
            goals = int(line_list[2])
            passes = line_list[3]
            fouls = line_list[4]
```

season,name,goals,passes,fouls

```
2018,Pierre-Emerick Auba
2018,Sadio Mané,22,1,34
2018,Mohamed Salah,22,1,
2018,Sergio Agüero,21,77
2018,Jamie Vardy,18,416,
2018,Eden Hazard,16,1,12
2018,Callum Wilson,14,44
2018,Raúl Jiménez,13,1,4
2018,Alexandre Lacazette
2018,Glenn Murray,13,600
```

Reading-in Data from a File to Dict

- Iterate over lines, after we've parsed them...
  - dictionary stuff!



```
def read_file(filename):
    with open(filename) as in_file:
        # make a new empty dictionary (accumulation variable)
        season_table = dict()
        # iterate over each line of the file
        for line in in_file:
            line_list = split(strip(line), ',')
            # "unpack" the list
            season = int(line_list[0])
            name = line_list[1]
            goals = int(line_list[2])
            # if season in table, grab it, otherwise use empty dict
            name_table = dict()
            if season in season_table:
                name_table = season_table[season]
            # we could check to see if name is in name_table,
            # but we know each name only appears once per season
            name_table[name] = goals # add name -> goals inner dictionary

            # add name_table back to season_table
            season_table[season] = name_table

    return season_table
```

Reading-in Data from a File to Dict

- Iterate over lines, after we've parsed them...
  - dictionary stuff!
- Can call the function, double-check output seems reasonable:

```
>>> season_table = read_file("seasonStats/seasons2018-2022.csv")
>>> print(len(season_table[2018]))
13
```

Splitting Values into X & Y lists

- Want to plot season-by-season...
- With **`matplotlib`**, we'll need a list of x and associated y values

```
selected_season = 2018 # season we'll produce list for
top_scorers2018 = []
num_goals2018 = []
if selected_season in season_table: # check it exists
    name_table = season_table[selected_season]
    for name in name_table:
        top_scorers2018 += [name]
        num_goals2018 += [name_table[name]]
```

```
>>> print(len(top_scorers2018), ':', top_scorers2018)
>>> print(len(num_goals2018), ':', num_goals2018)

13 : ['Pierre-Emerick Aubameyang', 'Sadio Mané', 'Mohamed Salah', 'Sergio
Agüero', 'Jamie Vardy', 'Eden Hazard', 'Callum Wilson', 'Raúl Jiménez',
'Alexandre Lacazette', 'Glenn Murray', 'Paul Pogba', 'Richarlison', 'Gylfi
Sigurdsson']

13 : [22, 22, 22, 21, 18, 16, 14, 13, 13, 13, 13, 13, 13]
```

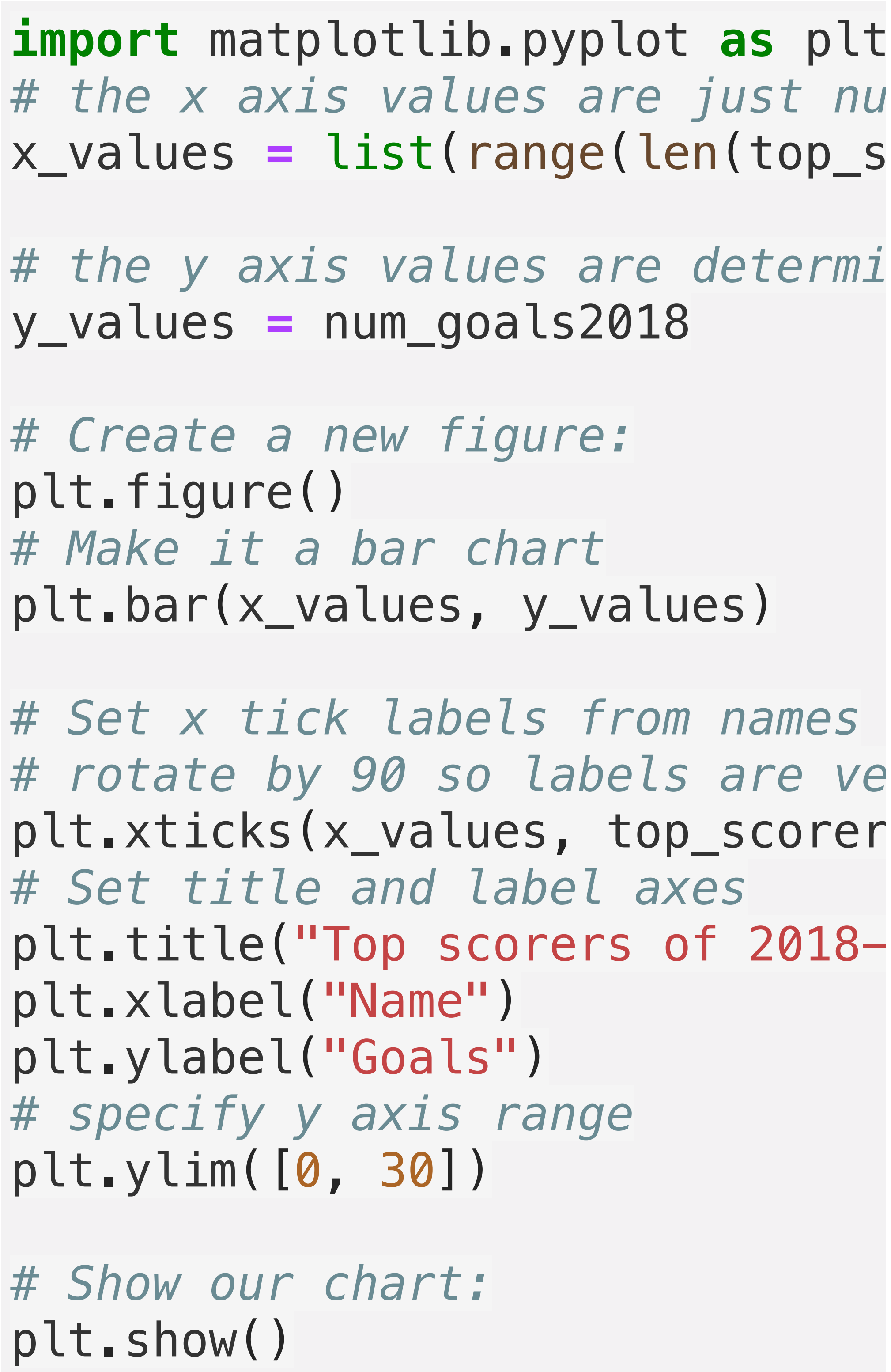
Plotting

- Now, we plot!

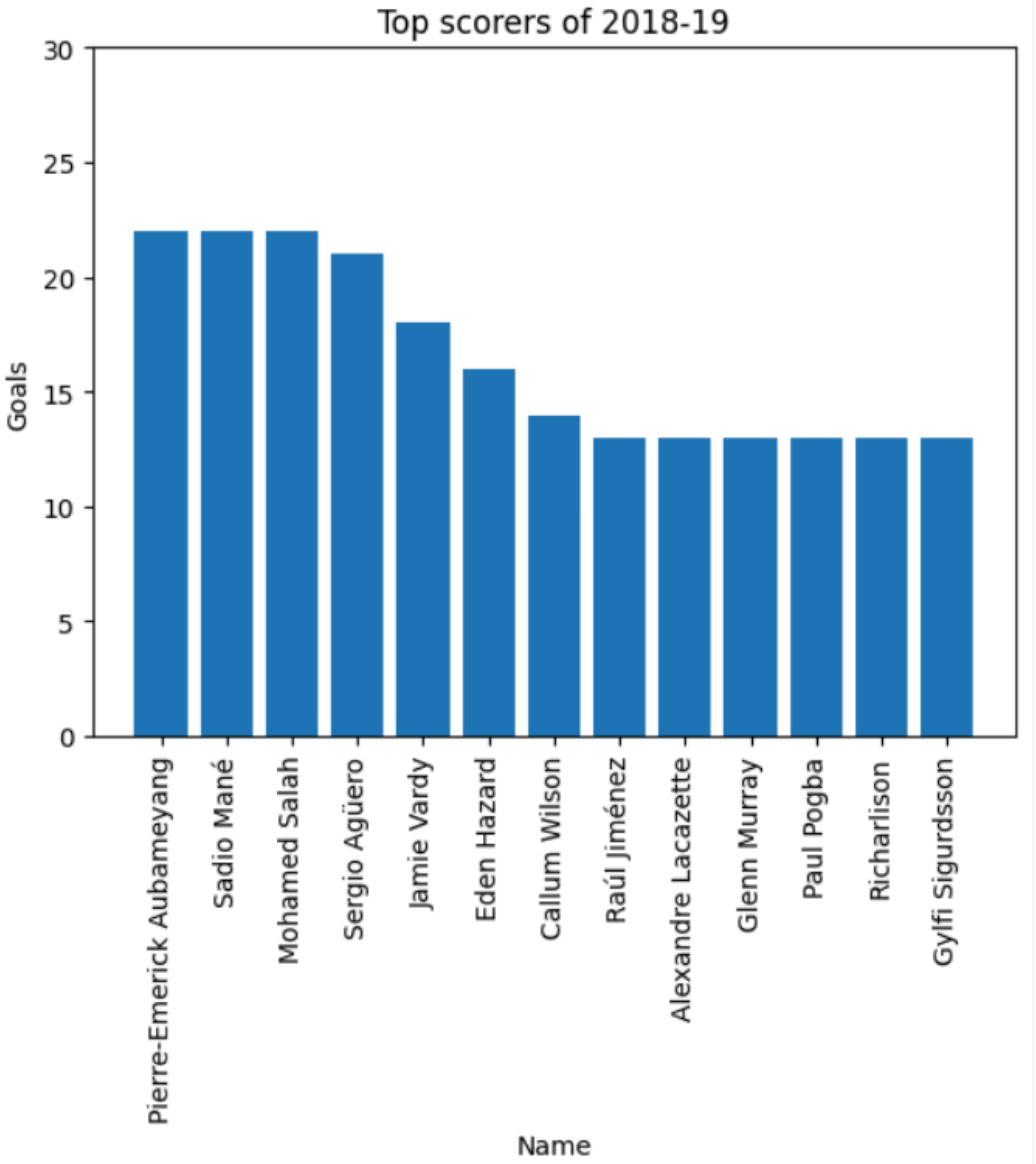
```
import matplotlib.pyplot as plt
# the x axis values are just num of names to provide even spacing for each bar
x_values = list(range(len(top_scorers2018)))

# the y axis values are determined by the number of goals scored
y_values = num_goals2018

# Create a new figure:
plt.figure()
# Make it a bar chart
plt.bar(x_values, y_values)

# Set x tick labels from names
# rotate by 90 so labels are vertical and do not overlap
plt.xticks(x_values, top_scorers2018, rotation=90)
# Set title and label axes
plt.title("Top scorers of 2018-19")
plt.xlabel("Name")
plt.ylabel("Goals")
# specify y axis range
plt.ylim([0, 30])

# Show our chart:
plt.show()
```

Plotting

- Now, we plot!



```
import matplotlib.pyplot as plt
# the x axis values are just nu
x_values = list(range(len(top_s

# the y axis values are determi
y_values = num_goals2018

# Create a new figure:
plt.figure()
# Make it a bar chart
plt.bar(x_values, y_values)

# Set x tick labels from names
# rotate by 90 so labels are ve
plt.xticks(x_values, top_scorer
# Set title and label axes
plt.title("Top scorers of 2018-
plt.xlabel("Name")
plt.ylabel("Goals")
# specify y axis range
plt.ylim([0, 30])

# Show our chart:
plt.show()
```


Top scorers of 2018-19
Goals
Pierre-Emerick Aubameyang
Sadio Mané
Mohamed Salah
Sergio Agüero
Jamie Vardy
Eden Hazard
Callum Wilson
Raúl Jiménez
Alexandre Lacazette
Glenn Murray
Paul Pogba
Richarlison
Gylfi Sigurdsson
Name