technoteach

technocamps

Llywodraeth Cymru
Welsh Government

Prifysgol Abertawe
Swansea University

Cardiff University
Prifysgol Caerdydd

Prifysgol Bangor University

Cardiff Metropolitan University
Prifysgol Metropolitan Caerdydd

University of South Wales
Prifysgol De Cymru

Cyngor Cyllido Addysg Uwch Cymru
Higher Education Funding Council for Wales
hefcw

Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Prifysgol Aberystwyth University

Prifysgol Glyndŵr Wrecsam
Wrexham Glyndŵr University

institute of CODING in wales
technocamps

technoteach
technocamps

Python – MatPlotLib

Python vs. MATLAB

Python is an open-source programming language, meaning that it is free, all the source files are freely accessible, and there is a large developer community writing new libraries and able to offer support.

MATLAB is a very powerful tool, and very popular in industry and academia. However, it is a commercial product, meaning that a license is required to download the base product (*it is not cheap*), and each additional toolkit you may require can be downloaded for an additional fee!

Python vs. MATLAB

Conveniently both MATLAB and Python have very similar syntax and structure (there are minor differences, as well as differences in how data types are handled, but they're relatively similar).

As Python is open source, there is an entire library designed to behave and display graphs and charts just as you would in MATLAB!

This is the library we are going to use today, called MatPlotLib.

MatPlotLib

To get to grips with using data structures in Python and the basic plot functions of MatPlotLib, we're going to load in some very basic data.

Our first data set has been shamefully provided by one of our delivery officers, Dan.

Download Data

The first set of data that we will use today can be downloaded here:

tc1.me/DansDiet

Once downloaded, it can be dragged into your project folder.

Load in File

To begin we will learn to load in a file:

```
dietSource = open("DansDiet.txt", "r")
diet = []

for line in dietSource:
      diet.append(line)

print(diet)
```

Load in File

Notice how when printed the output contains '\n' at the end of each line that was read into the program:

```
['Pizza Hut\n', 'Greggs\n', 'Oven Food\n', 'Greggs\n', 'Pizza Hut\n',
```

This is the newline character that tells the computer to progress to the next line!

If we want to remove this, we will have to specify it!

Load in File

Change your program to look like this:

```
dietSource = open("DansDiet.txt", "r")
diet = []

for line in dietSource:
        diet.append(line.replace("\n", "")

print(diet)
```

Notice how the '\n' has been replaced with nothing.

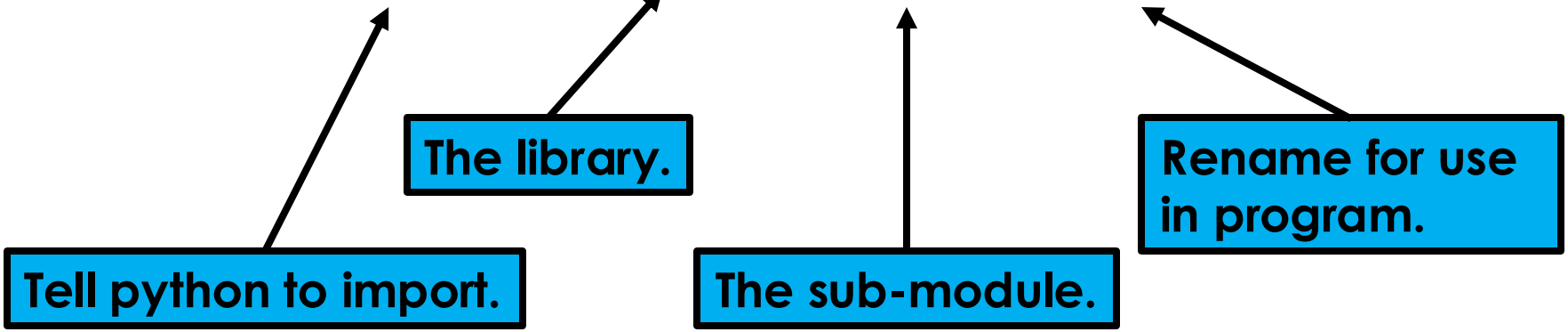
Begin Plotting

In order to plot anything we will need access to these functions!

We are going to be using the MatPlotLib library for Python, so we will need to import this for use in our program.

At the very top of our Python file, we need to add this line::

```
import matplotlib.pyplot as plot
```



Begin Plotting

To plot the data, we'll need the following code:

```
plot.figure()

fig1 = plot.bar([x[0] for x in dietCounts],
                [y[1] for y in dietCounts])

plot.show()
```

Create a window for the plot.

These are in line for loops.
The bar plot is being given two lists of values, one from column [0] of the data, the other from column [1].

Display the plot.

Adjust the Plot

Our data will automatically be plotted with a basic graph, however all elements can be adjusted before the plot is shown:

```
plot.xticks(rotation = 90)

plot.subplots_adjust(bottom = 0.25)

plot.show()
```

Rotate the x-axis labels so that they don't overlap.

Resize the bottom of the plot so that the rotated labels still fit.

The same show() as on the last slide.

Adjust the Plot

We can add titles to our chart and axes using these methods:

```
plot.xlabel("Food Options")
plot.ylabel("Number of Visits")
```

Add a label to each axis.

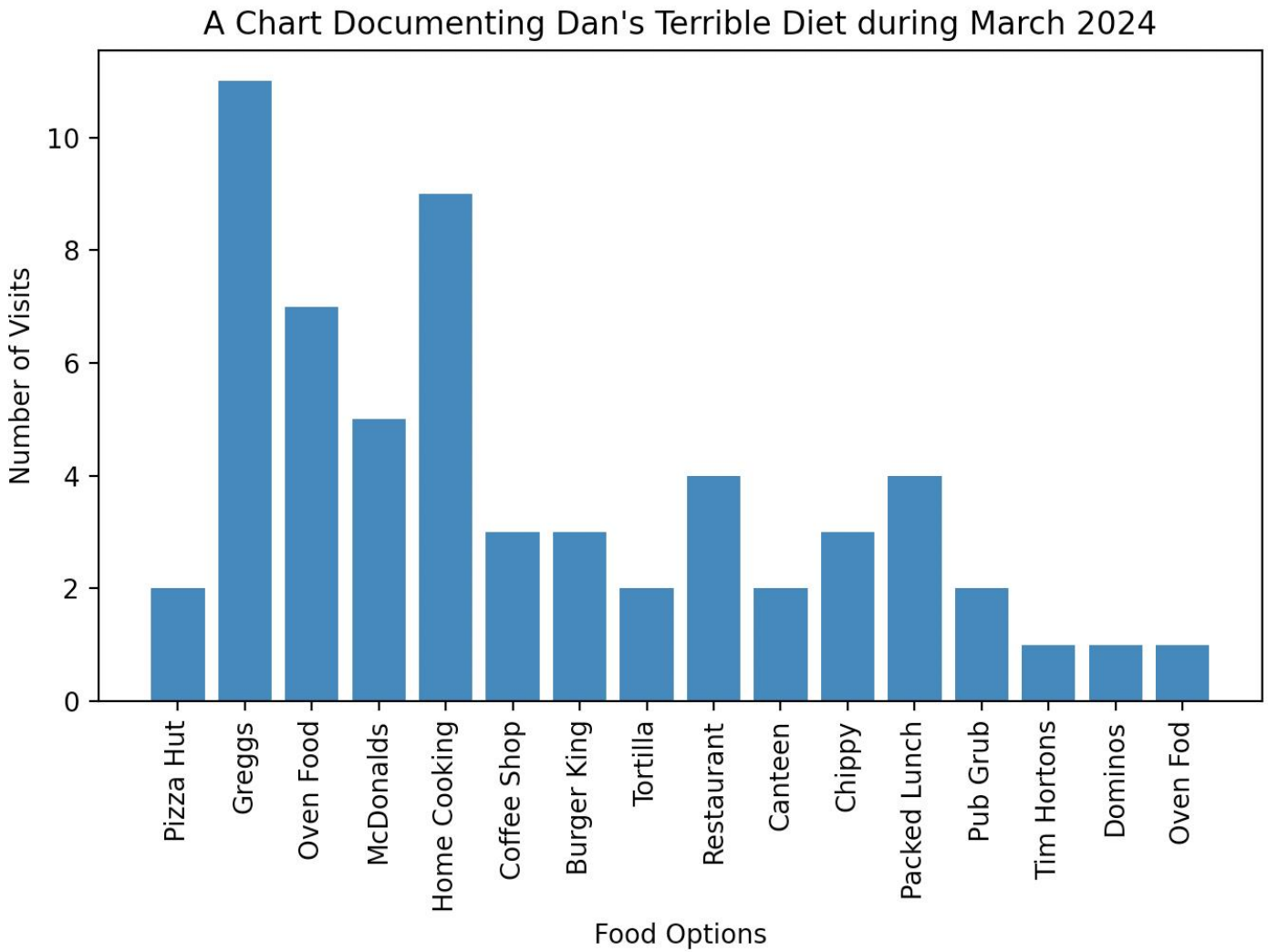
```
plot.title("A Chart Documenting Dan's Terrible
            Diet During March 2024")
```

Add a title to the top of the chart.

```
plot.show()
```

The same show() as on the last slide.

The Plot


A Chart Documenting Dan's Terrible Diet during March 2024
Number of Visits
0
2
4
6
8
10
Pizza Hut
Greggs
Oven Food
McDonalds
Home Cooking
Coffee Shop
Burger King
Tortilla
Restaurant
Canteen
Chippy
Packed Lunch
Pub Grub
Tim Hortons
Dominos
Oven Fod
Food Options

Count Categories

We can also manipulate our data to be in categories:

```
dietCategories = [["Fast Food", 0], ["Eating Out", 0],
                  [["Home Made", 0]]
for item in dietCounts:
      for item in diet:
            if ((item[0] == "Greggs") or
                  (item[0] == "McDonalds") or …):
                        dietCategories[0][1] += item[1]
            elif ((item[0] == "Pizza Hut") or
                  (item[0] == "Restaurant") or …):
                        dietCategories[1][1] += item[1]
            elif ((item[0] == "Home Cooking") or
                  (item[0] == "Oven Food") or …): :
                        dietCategories[2][1] += item[1]
```

Count Categories

```
plot.figure()
plot.title("A Pie Chart of Dan's Diet Categories")
explodeVal = (0, 0, 0.5)

fig2 = plot.pie(
       [counts[1] for counts in dietCategories],
       explode = explodeVal,
       shadow = True,
       labels = [labels[0] for labels in
                                dietCategories],
       autopct = '%1.1f%%',
       colors = ['crimson', 'orange', 'yellow'])

plot.show()
```

Count Categories

Now we will investigate importing more complex data:

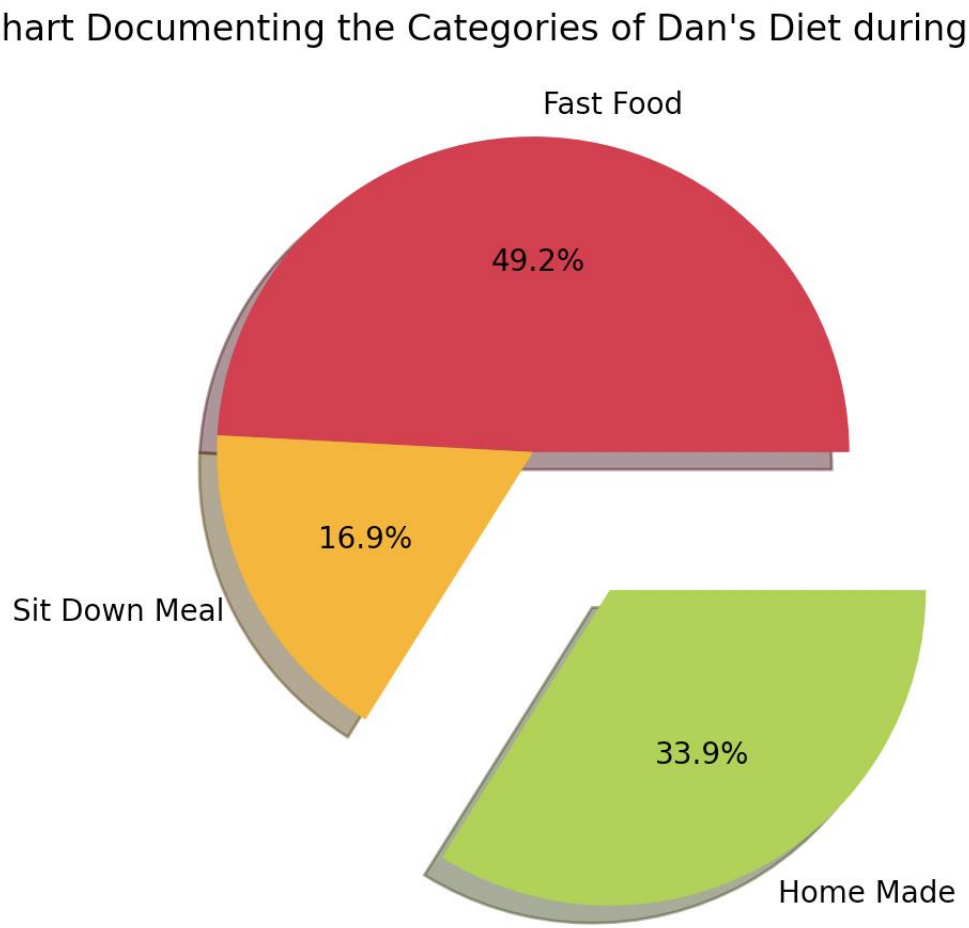
```
theUFOFile = open("UFO_Sightings.csv", "r")
allUFOData = []

for i, line in enumerate(theUFOFile):

        if i != 0:

                allUFOData.append(line.split(","))
```

Count Categories



Download UFO Data

The UFO data can be downloaded here:

tc1.me/UFOData

Once downloaded, it can be dragged into your project folder.

Open the file and take a look at the data.

UFO Sightings

We will try to plot the number of UFO reports that were received at each hour of the day, to see what times are more likely for a UFO to be spotted!

To plot this, we will have to sort the data by the timestamps, and find out how many reports were generated for each hour.

If we were to just plot the data, we would have a tick on the axis for **every timestamp**, not a tick for **each hour** of the day.

To plot this will therefore require us to manipulate the data, and extract the hour from each of the timestamps.

Load in UFO File

Change your program to look like this:

```
theUFOFile = open("UFO_Sightings.csv", "r")
allUFOData = []

for line in theUFOFile:
        line = line.replace("\n", "")
        allUFOData.append(line.split(","))

print(diet)
```

Notice how each line has been split into a list.

UFO Sightings – Sort by DateTime

To begin we will sort our data by the timestamp. This will require a new library. By first sorting the data by timestamp we can loop through the data easily:

```
from operator import itemgetter

…


allUFOData.sort(key = itemgetter(1))


timeCount = []
hourStr = "00"
count = 0
```

UFO Sightings – Extract Hours

Now that we have sorted the data by timestamp, we need to pull out each hour:

```
for entry in allUFOData:
        if entry[1][:2] == hourStr:
                count += 1
        else:
                timeStr = hourStr + ":00"
                timeCount.append([timeStr, count])
                hourStr = entry[1][:2]
                count = 1
```

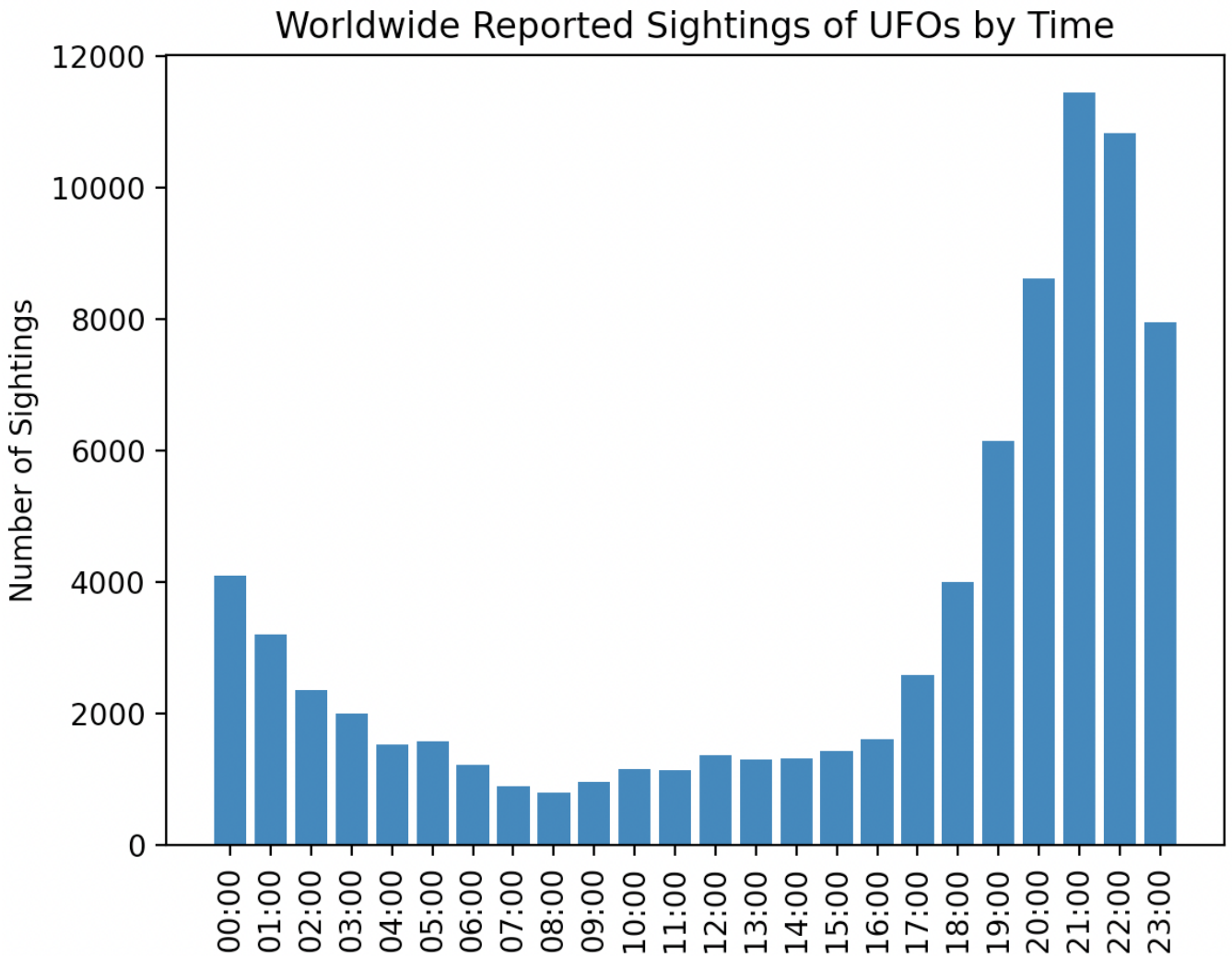
UFO Sightings - Plot

With the counts for each hour extracted from our data, we can plot:

```
plot.figure()
fig3 = plot.bar([str[0] for str in timeCount],
                [count[1] for count in timeCount])

plot.xticks(rotation = 90)
plot.xlabel("Time")
plot.ylabel("Sightings")
plot.title("Reported Sightings by Date")
```

UFO Sightings



UFO Sightings by Country

We can add the number of sightings per country with a for loop:

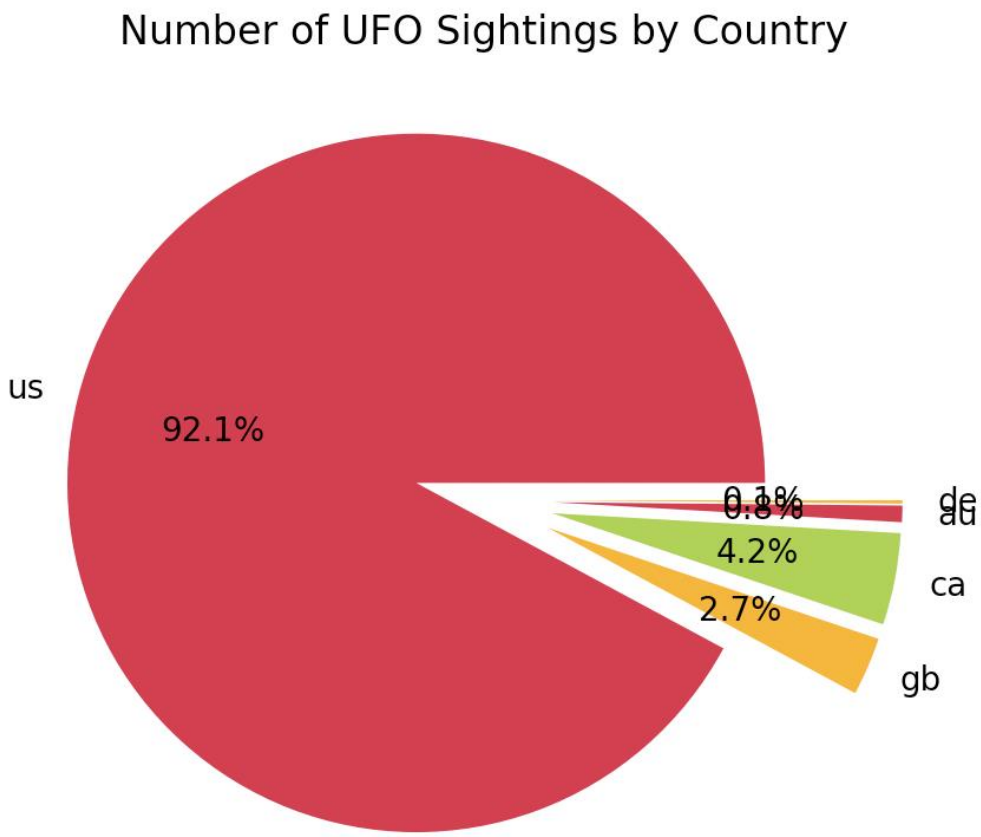
```
countryCounts = []
for entry in allUFOData:
      if entry[3] != "":
            if entry[3] not in [country[0] for
                        country in countryCounts]:
                  countryCounts.append([entry[3], 1])
            else:
                  for country in countryCounts:
                        if country[0] == entry[3]:
                              country[1] += 1
```

UFO Sightings by Country

```
plot.figure()
plot.title("Number of UFO Sightings by Country")
explodeVal = (0.2, 0.2, 0.2, 0.2, 0.2)


fig4 = plot.pie(
      [counts[1] for counts in countryCounts]),
      explode = explodeVal,
      autopct = '%1.1%%',
      colors = ['crimson', 'orange', 'yellowgreen'],
      labels = [country[0] for country in
                  countryCounts])
plot.show()
```

UFO Sightings by Country



Duration vs. Time of Day

We can make a more complex scatter graph by taking two measurements – the **duration** and **time of day** for each sighting!

To begin we need to ensure both of these lines are still present in our code:

```
from operator import itemgetter

…

allUFOData.sort(key = itemgetter(1))
```

UFO Sightings – Extract Hours

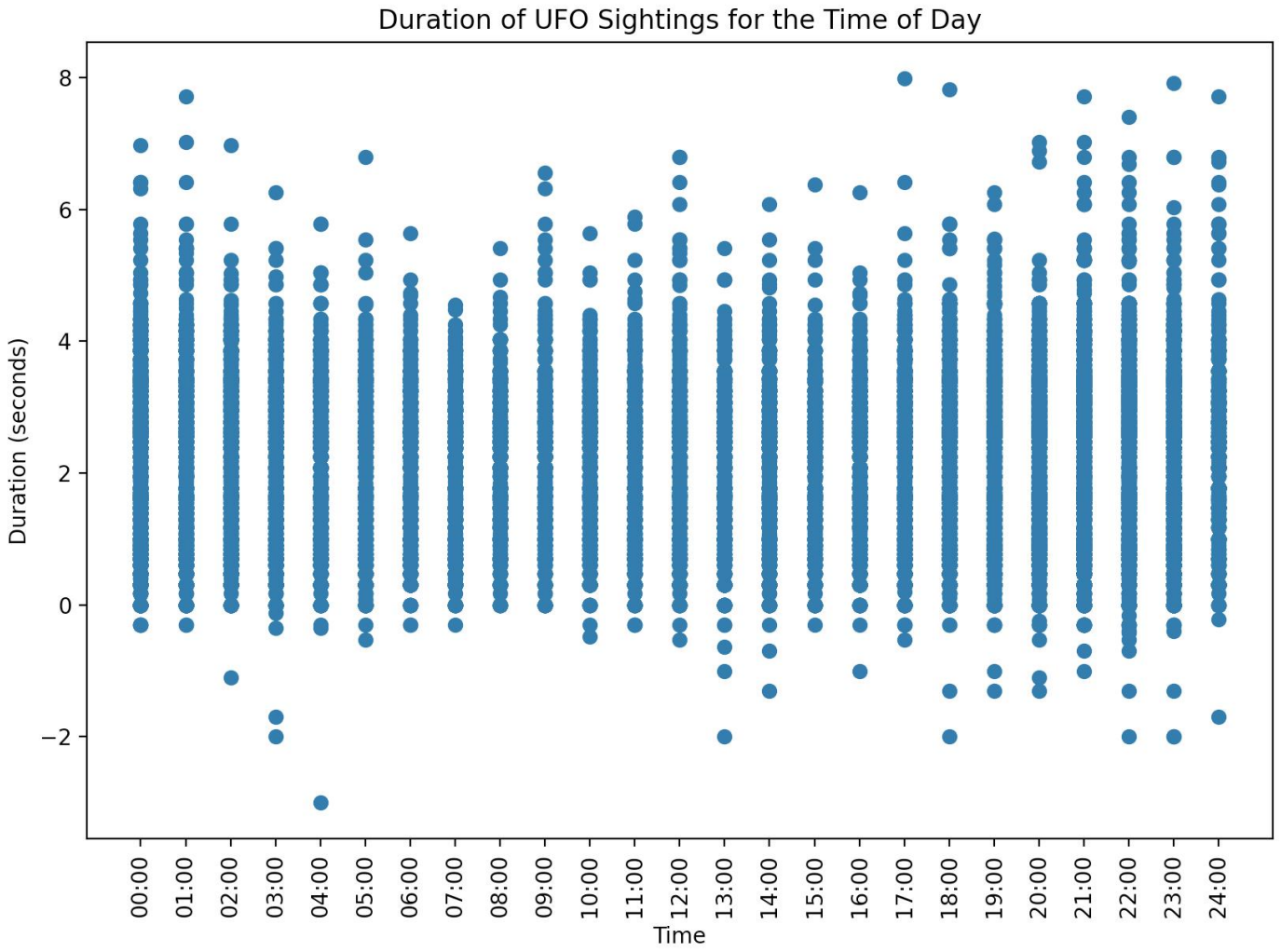
```
timeDuration = []
hourStr = "00"
timeStr = hourStr + ":00"

for entry in allUFOData:
        if entry[1][:2] == hourStr:
                timeDuration.append([timeStr,
                        float(entry[5].replace("\n", ""))])
        else:
                hourStr = entry[1][:2]
                timeStr = hourStr + ":00"
                timeDuration.append([timeStr,
                        float(entry[5].replace("\n", ""))])
```

UFO Sightings – Extract Hours

```
with plot.style.context('tableau-colorblind10'):
        fig5 = plot.scatter([time[0] for time in
                                        timeDuration],
                [math.log10(duration[1]) for duration in
                                        timeDuration])
plot.xticks(rotation = 90)
plot.title("Duration of Sighting - Time of Day")
plot.xlabel("Time")
ploat.ylabel("Duration (s)")
```

UFO Sightings by Country



technoteach
technocamps

Python – Pandas

Pandas

Pandas is a powerful data manipulation and analysis library for Python.

It provides data structures and functions that are incredibly efficient for working with complex data (tables, spreadsheets etc.)

Pandas can easily load data from various sources, clean and process it, perform complex operations like filtering or grouping, manipulate the data in many ways, and visualize the results.

It's like having a Swiss Army knife for data analysis tasks in Python.

Download Data

Today we are going to use a more complex data set, all Olympic athletes and the events they competed in since 1896!

This will allow us to demonstrate the benefits of using Pandas:

tc1.me/OlympicFile

Once downloaded, this file can be dragged into your project folder.

Load in File

To begin we will learn to load in a file with pandas:

```
import pandas as pd

athleteData = pd.read_csv("athlete_events.csv",
                  usecols = ["ID", "Sex", "Age",
                  "Height", "Weight", "NOC", "Year",
                  "Season", "Sport", "Medal"])
print(athleteData)
```

Load in File

0	1	M	24.0	180.0	...	1992	Summer	Basketball	NaN
1	2	M	23.0	170.0	...	2012	Summer	Judo	NaN
2	3	M	24.0	NaN	...	1920	Summer	Football	NaN
3	4	M	34.0	NaN	...	1900	Summer	Tug-Of-War	Gold
4	5	F	21.0	185.0	...	1988	Winter	Speed Skating	NaN
...	...	..	...	...	...	...	...	...	...
271111	135569	M	29.0	179.0	...	1976	Winter	Luge	NaN
271112	135570	M	27.0	176.0	...	2014	Winter	Ski Jumping	NaN
271113	135570	M	27.0	176.0	...	2014	Winter	Ski Jumping	NaN
271114	135571	M	30.0	185.0	...	1998	Winter	Bobsleigh	NaN
271115	135571	M	34.0	185.0	...	2002	Winter	Bobsleigh	NaN

Load in File

0		1	M	24.0	180.0	...	1992	Summer	Basketball	NaN
1		2	M	23.0	170.0	...	2012	Summer	Judo	NaN
2		3	M	24.0	NaN	...	1920	Summer	Football	NaN
3		4	M	34.0	NaN	...	1900	Summer	Tug-Of-War	Gold
4		5	F	21.0	185.0	...	1988	Winter	Speed Skating	NaN
...		...	..	...	...	...	...	...	...	...
271111	135569		M	29.0	179.0	...	1976	Winter	Luge	NaN
271112	135570		M	27.0	176.0	...	2014	Winter	Ski Jumping	NaN
271113	135570		M	27.0	176.0	...	2014	Winter	Ski Jumping	NaN
271114	135571		M	30.0	185.0	...	1998	Winter	Bobsleigh	NaN
271115	135571		M	34.0	185.0	...	2002	Winter	Bobsleigh	NaN

Same athlete, different events.

This is a pandas DataFrame.
It is a unique data structure to hold a table of information, with headers and row numbers.

Pandas Basics

For the first task we will begin to get familiar with pandas' data structures and the basic functions available!

To **filter the data** for a specific condition, we can use this format using a boolean expression:

```
goldAthletes = athleteData[
                    athleteData["Medal"] == "Gold"]
```

Pandas Basics

To get the mean result of a particular column, we can use the following function:

```
meanAge = athleteData["Age"].mean()
print(meanAge.round())
```

Pandas Basics

Now that we know the average age, we can filter by those older and younger who won a medal:

```
medalBelowMean = athleteData[
                    (athleteData["Medal"].notna()) &
                    (athleteData["Age"] < 26)]
medalAboveMean = athleteData[
                    (athleteData["Medal"].notna()) &
                    (athleteData["Age"] >= 26)]
print(f"Below: {len(medalBelowMean)}\n
            Above: {len(medalAboveMean)}")
```

Pandas Basics

We can get the oldest and youngest athletes by sorting by age, then looking at the top and bottom of the DataFrame:

```
athleteAgeSort = athleteData[athleteData["Age"]
                 .notna()].sort_values("Age",
                      ascending = False)

print(athleteAgeSort.head())
print(athleteAgeSort.tail())
```

Plot Scatter Graph

To plot a basic graph using pandas is very simple:

```
import matplotlib.pyplot as plt
...
plt.figure()
fig1 = plt.scatter(athleteData.get("Year"),
                   athleteData.get("Age"))
...
plt.show()
```

Better Scatter Graph

We can now use our previous knowledge to both filter and label:

```
fig2 = plt.scatter(
                    athleteData[athleteData["Medal"] ==
                    "Gold"].get("Year"),
                    athleteData[athleteData["Medal"] ==
                    "Gold"].get("Age"), color = "gold")
plot.xlabel("Year")
plot.ylabel("Age")
plot.title("Gold Medal Winners Age
            across all Olympic Years")
```

Grouping Data

A very similar method to our filtering is grouping, which groups our data into the discrete entries within a column.

We will begin by filtering our data by country, and adding variables to store the number of medals:

```
athletesCountry = athleteData[
                          athleteData[“NOC”] == “GBR”]

medals = [“Gold”, “Silver”, “Bronze”]
medalsCountry = [0, 0, 0]
```

Grouping Data

Now we can plot both variables:

```
medalsCountry[0] = len(athletesCountry.groupby
                   ("Medal").get_group("Gold"))

medalsCountry[1] = len(athletesCountry.groupby
                   ("Medal").get_group("Silver"))

medalsCountry[2] = len(athletesCountry.groupby
                   ("Medal").get_group("Bronze"))
```

Grouping Data

We will get the length of the grouped results as a method of counting the number of each medal:

```
plt.figure()
fig3 = plt.bar(medals, medalsCountry)


plt.title(f"Medal wins for {countryCode}")
plt.xlabel("Medals")
plt.ylabel("Number of Medals")
plt.show()
```

Plotting Multiple Data Groups

Now we'll plot the medals for a select number of years:

```
athletesCountry = athleteData.groupby("NOC")
                  .get_group("GBR")

goldMedals   = len(athletesCountry.groupby
                   ("Medal").get_group("Gold"))
silverMedals = len(athletesCountry.groupby
                   ("Medal").get_group("Silver"))
bronzeMedals = len(athletesCountry.groupby
                   ("Medal").get_group("Bronze"))
```

Plotting Multiple Data Groups

```
medalsPerYear = {"Gold": [], "Silver": [],
                 "Bronze": []}
years = [2008, 2012, 2016]

for year in years:
    medalsPerYear["Gold"].append(len(goldsCountry
                .groupby("Year").get_group(year)))
    medalsPerYear["Silver"].append(len(goldsCountry
                .groupby("Year").get_group(year)))
    medalsPerYear["Bronze"].append(len(goldsCountry
                .groupby("Year").get_group(year)))
```

Plotting Multiple Data Groups

This is a more complex plot and so requires some configuration! Don't worry about the specifics, these come with practice:

```
x = np.arange(len(years))
width = 0.25
barID = 0
fig, ax = plt.subplots(layout = "constrained")
```

Plotting Multiple Data Groups

```
for attribute, measurement in medalsPerYear.items():
        if attribute == "Gold": barColor = "gold"
        if attribute == "Silver": barColor = "silver"
        if attribute == "Bronze": barColor = "peru"

        offset = width * barID
        bar = ax.bar(x + offset, measurement, width,
                label = attribute, color = barColor)
        ax.bar_label(bar, padding = 3)
        barID += 1
```

technoteach technocamps

Plotting Multiple Data Groups

```
ax.set_ylabel("Number of Medals")
ax.set_title(f"Medal Wins for {countryCode}")

ax.legend(loc = "upper left")

ax.set_xticks(x + width, years)
ax.set_ylim(0, 250)
```

Plotting Multiple Data Groups

To instead plot the medals won by a country across all Olympic years, we will need to account for the fact that some years a country may not have won a particular medal (Gold, Silver, Bronze) or any medal at all!

This is because when grouping the data, Python won't create groups for cases with no entries, meaning when we search for that group an error will be thrown.

This will require two new keywords:

`try:` and `except ____:`

Plotting Multiple Data Groups

The previous for loop must be slightly altered to use all years that the country competed:

```
for year in athletesCountry.groupby("Year").groups:
```

Plotting Multiple Data Groups

Within the for loop this format of try and catch will be required for each of the medals:

```
try:
    medalsPerYear["Gold"].append(len(goldsCountry
                .groupby("Year").get_group(year)))

except KeyError:
    medalsPerYear["Gold"].append(0)
    print(f"No Gold for {year}")
```