Preparing your figures to share with others

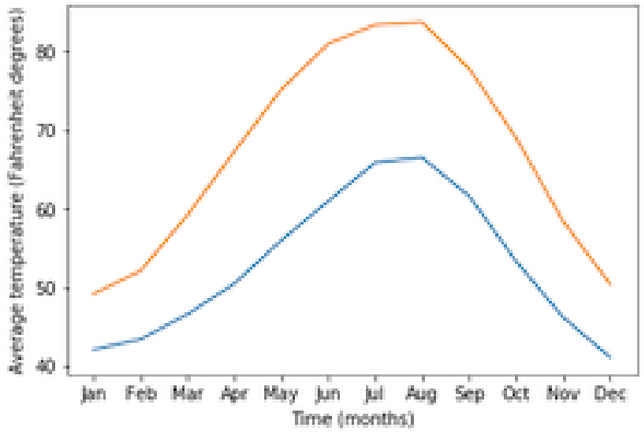
INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem
Data Scientist

datacamp

Changing plot style

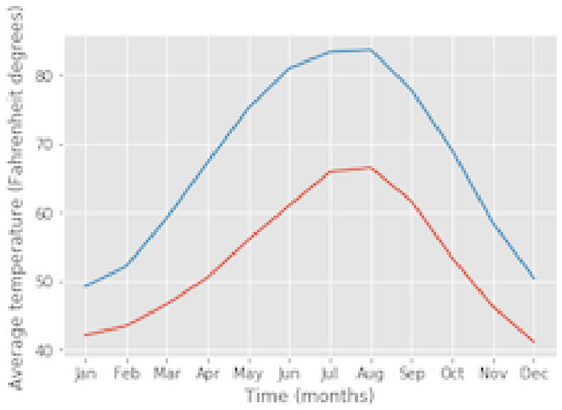
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-TAVG-NORMAL"])
ax.plot(austin_weather["MONTH"], austin_weather["MLY-TAVG-NORMAL"])
ax.set_xlabel("Time (months)")
ax.set_ylabel("Average temperature (Fahrenheit degrees)")
plt.show()
```



2. Changing plot style
Here, we'll change the overall style of the figure. To see what that means, let's look at one of the figures we created in a previous lesson. This figure shows the average temperatures in Seattle and Austin as a function of the months of the year. This is what it looks like per default.

Choosing a style

```
plt.style.use("ggplot")
fig, ax = plt.subplots()
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-TAVG-NORMAL"])
ax.plot(austin_weather["MONTH"], austin_weather["MLY-TAVG-NORMAL"])
ax.set_xlabel("Time (months)")
ax.set_ylabel("Average temperature (Fahrenheit degrees)")
plt.show()
```



3. Choosing a style
If instead, we add this line of code before the plotting code, the figure style will look completely different. The style we chose here emulates the style of the R library ggplot. Maybe you know this library and this looks familiar to you, or you can learn about ggplot in a DataCamp course devoted to this library. Either way, you will notice that the setting of the style didn't change the appearance of just one element in the figure. Rather, it changed multiple elements: the colors are different, the fonts used in the text are different, and there is an added gray background that creates a faint white grid marking the x-axis and y-axis tick locations within the plot area. Furthermore, this style will now apply to all of the figures in this session, until you change it by choosing another style.

Back to the default

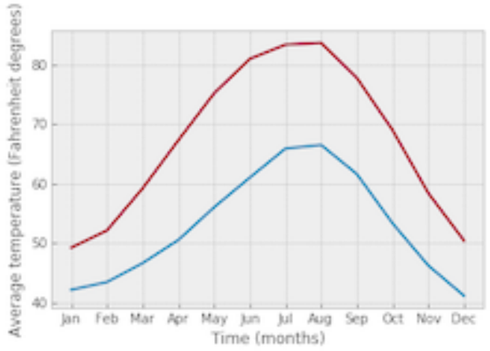
```
plt.style.use("default")
```

The available styles

https://matplotlib.org/gallery/style_sheets/style_sheets_refer

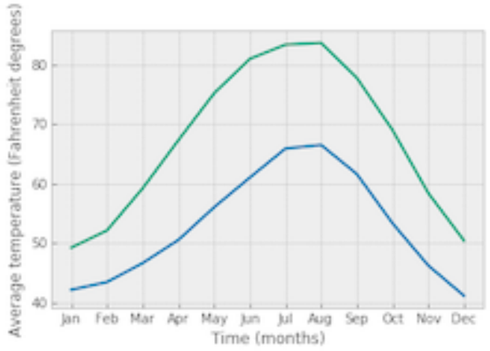
The "bmh" style

```
plt.style.use("bmh")
fig, ax = plt.subplots()
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-TAVG-NORMAL"])
ax.plot(austin_weather["MONTH"], austin_weather["MLY-TAVG-NORMAL"])
ax.set_xlabel("Time (months)")
ax.set_ylabel("Average temperature (Fahrenheit degrees)")
plt.show()
```



Seaborn styles

```
plt.style.use("seaborn-colorblind")
fig, ax = plt.subplots()
ax.plot(seattle_weather["MONTH"], seattle_weather["MLY-TAVG-NORMAL"])
ax.plot(austin_weather["MONTH"], austin_weather["MLY-TAVG-NORMAL"])
ax.set_xlabel("Time (months)")
ax.set_ylabel("Average temperature (Fahrenheit degrees)")
plt.show()
```



Guidelines for choosing plotting style

- Dark backgrounds are usually less visible
- If color is important, consider choosing colorblind-friendly options
  - "seaborn-colorblind" or "tableau-colorblind10"
- If you think that someone will want to print your figure, use less ink
- If it will be printed in black-and-white, use the "grayscale" style

Practice choosing the right style for you!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Sharing your visualizations with others

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

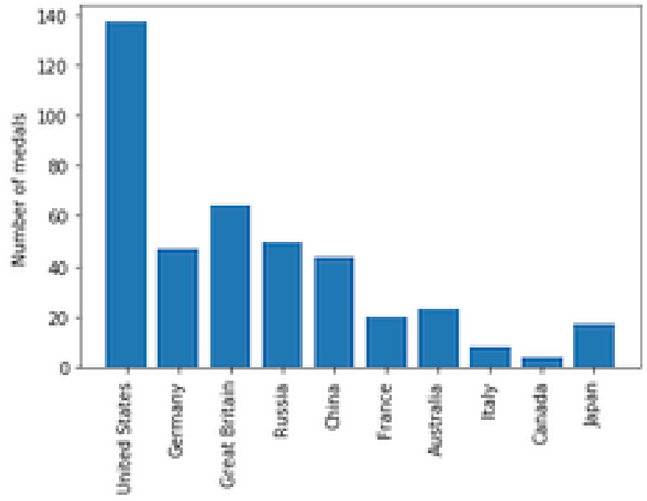
Ariel Rokem
Data Scientist

datacamp

A figure to share

```
fig, ax = plt.subplots()

ax.bar(medals.index, medals["Gold"])
ax.set_xticklabels(medals.index, rotation=90)
ax.set_ylabel("Number of medals")

plt.show()
```

2. A figure to share
Take for example this figure that you previously created to display data about the number of gold medals that each of several countries won in the 2016 Olympic Games. When you previously ran this code, it displayed the figure on your screen when you called the plt-dot-show method at the end of this code.



Saving the figure to file

```
fig, ax = plt.subplots()


ax.bar(medals.index, medals["Gold"])
ax.set_xticklabels(medals.index, rotation=90)
ax.set_ylabel("Number of medals")


fig.savefig("gold_medals.png")
```

3. Saving the figure to file
Now, we replace the call to plt-dot-show with a call to the Figure object's savefig method. We provide a file-name as input to the function. If we do this, the figure will no longer appear on our screen, but instead appear as a file on our file-system called "gold-underscore-medals-dot-png". In the interactive Python shell that we are using here, we can call the unix ls function, which gives us a listing of the files in the present working directory. In this case, only the file that we created is present. We can then share this file that now contains the visualization with others.

```
ls
```

```
gold_medals.png
```

Different file formats

```
fig.savefig("gold_medals.jpg")
```

```
fig.savefig("gold_medals.jpg", quality=50)
```

```
fig.savefig("gold_medals.svg")
```

4. Different file formats
In the previous slide, we saved the figure as a PNG file. This file format provides lossless compression of your image. That means that the image will retain high quality, but will also take up relatively large amounts of diskspace or bandwidth. You can choose other file formats, depending on your need. For example, if the image is going to be part of a website, you might want to choose the jpg format used here, instead. This format uses lossy compression, and can be used to create figures that take up less diskspace and less bandwidth. You can control how small the resulting file will be, and the degree of loss of quality, by setting the quality key-word argument. This will be a number between 1 and 100, but you should avoid values above 95, because at that point the compression is no longer effective. Choosing the svg file-format will produce a vector graphics file where different elements can be edited in detail by advanced graphics software, such as Gimp or Adobe Illustrator. If you need to edit the figure after producing it, this might be a good choice.

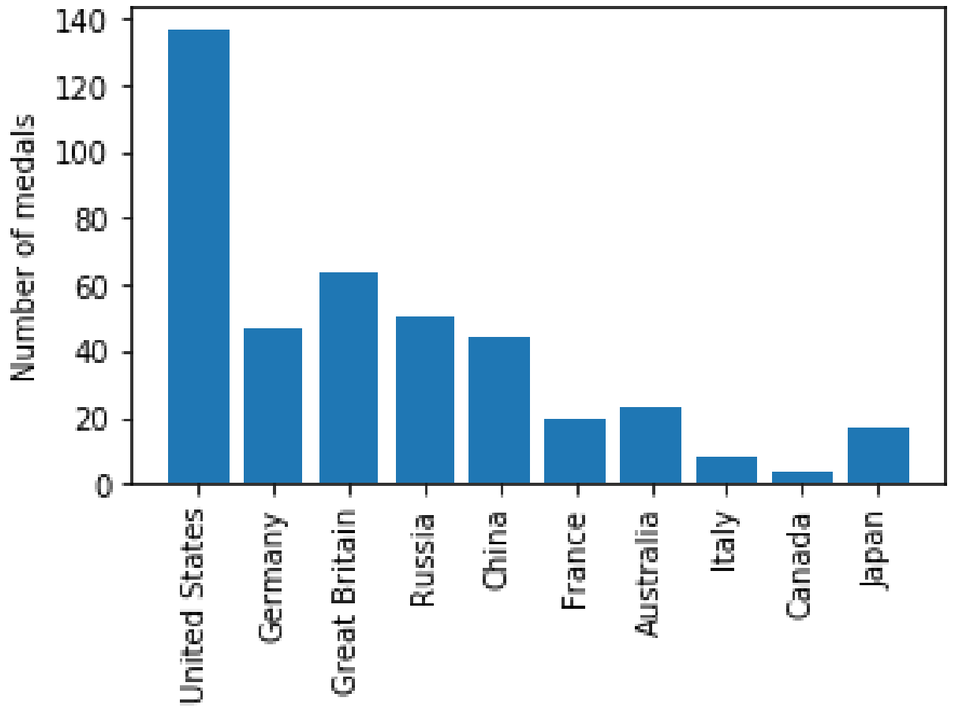
Resolution

```
fig.savefig("gold_medals.png", dpi=300)
```

5. Resolution
Another key-word that you can use to control the quality of the images that you produce is the dpi key-word argument. This stands for dots per inch. The higher this number, the more densely the image will be rendered. If you set this number to 300, for example, this will render a fairly high-quality resolution of your image to file. Of course, the higher the resolution that you ask for, the larger the file-size will be.

Size

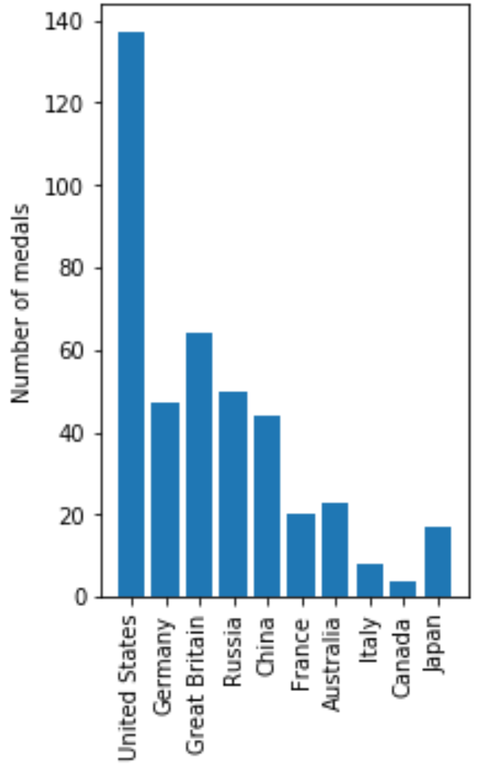
```
fig.set_size_inches([5, 3])
```



6. Size
Finally, another thing that you might want to control is the size of the figure. To control this, the Figure object also has a function called set-underscore-size-underscore-inches. This function takes a sequence of numbers. The first number sets the width of the figure on the page and the second number sets the height of the figure. So setting the size would also determine the aspect ratio of the figure. For example, you can set your figure to be wide and short

Another aspect ratio

```
fig.set_size_inches([3, 5])
```



Practice saving your visualizations!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Automating figures from data

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Ariel Rokem
Data Scientist

datacamp

Why automate?

- Ease and speed
- Flexibility
- Robustness
- Reproducibility

2. Why automate?
This means that you can write functions and programs that automatically adjust what they are doing based on the input data. Why would you want to automate figure creation based on the data? Automation makes it easier to do more. It also allows you to be faster. This is one of the major benefits of using a programming language like Python and software libraries such as Matplotlib, over tools that require you to interact with a graphical user interface every time you want to create a new figure. Inspecting the incoming data and changing the behavior of the program based on the data provides flexibility, as well as robustness. Finally, an automatic program that adjusts to the data provides reproducible behavior across different runs.

How many different kinds of data?

```
summer_2016_medals["Sport"]
```

```
ID
62              Rowing
65          Taekwondo
73           Handball
              ...
134759       Handball
135132     Volleyball
135205         Boxing
Name: Sport, Length: 976, dtype: object
```

3. How many different kinds of data?
Let's see what that means for Matplotlib. Consider the data about Olympic medal winners that we've looked at before. Until now, we always looked at two different branches of sports and compared them to each other, but what if we get a new data file, and we don't know how many different sports branches are included in the data? For example, what if we had a data-frame with hundreds of rows and a "Sport" column that indicates which branch of sport each row belongs to.

Getting unique values of a column

```
sports = summer_2016_medals["Sport"].unique()
print(sports)
['Rowing' 'Taekwondo' 'Handball' 'Wrestling'
'Gymnastics' 'Swimming' 'Basketball' 'Boxing'
'Volleyball' 'Athletics']
```

Note:

Automate your visualization
One of the main strengths of Matplotlib is that it can be automated to adapt to the data that it receives as input. For example, if you receive data that has an **unknown number of categories,** you can still create a bar plot that has bars for each category.

4. Getting unique values of a column
A column in a Pandas DataFrame is a Pandas Series object, so we can get the list of different sports present in the data by calling the unique method of that column.
This tells us that there are 10 different branches of sport here.

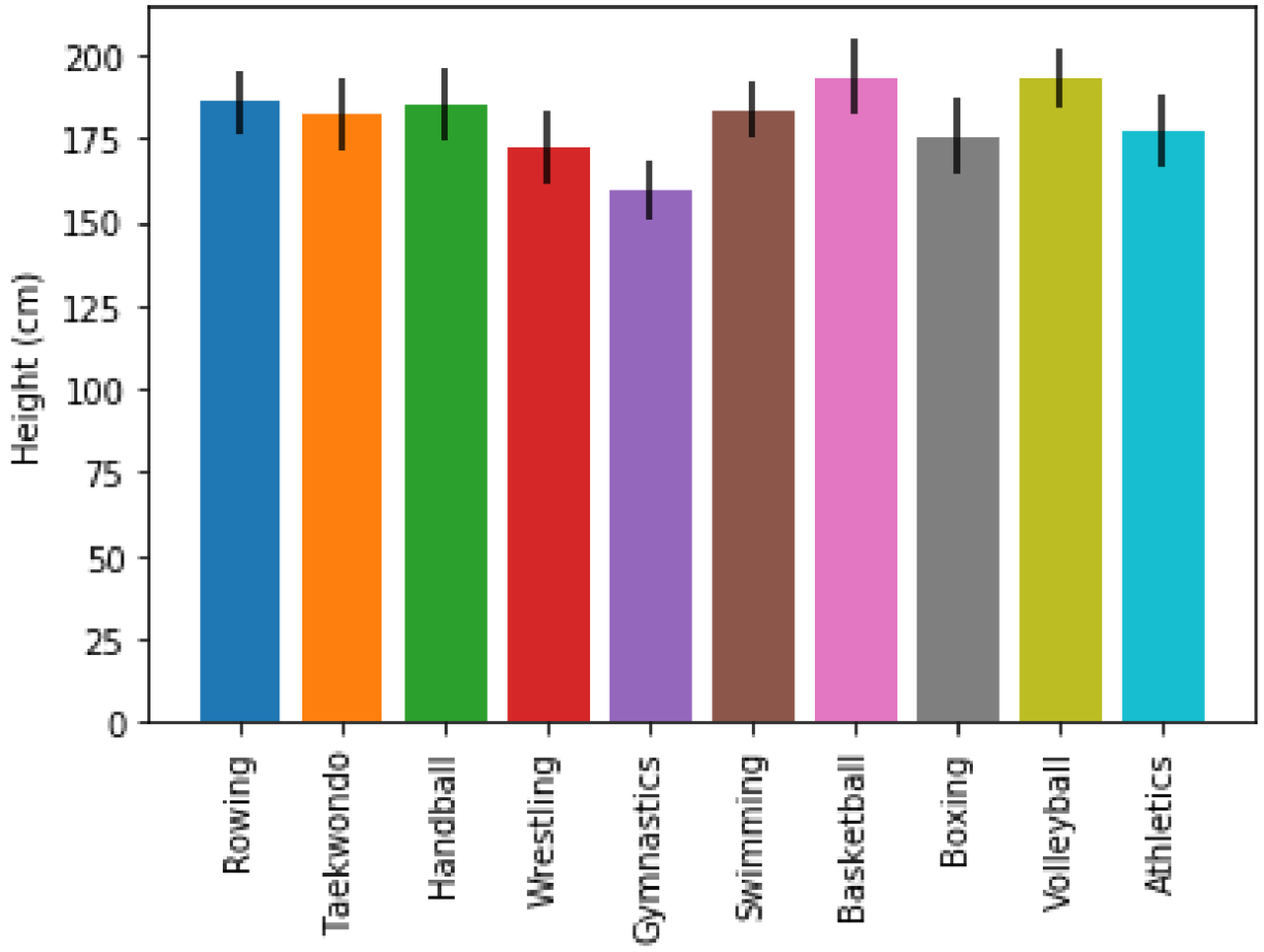
Bar-chart of heights for all sports

```
fig, ax = plt.subplots()

for sport in sports:
  sport_df = summer_2016_medals[summer_2016_medals["Sport"] == sport]
  ax.bar(sport, sport_df["Height"].mean(),
         yerr=sport_df["Height"].std())
ax.set_ylabel("Height (cm)")
ax.set_xticklabels(sports, rotation=90)
plt.show()
```

5. Bar-chart of heights for all sports

Let's say that we would like to visualize the height of athletes in each one of the sports, with a standard deviation error bar. Given that we don't know in advance how many sports there are in the DataFrame, once we've extracted the unique values, we can loop over them. In each iteration through, we set a loop variable called sport to be equal to one of these unique values. We then create a smaller DataFrame, that we call sport-underscore-d-f, by selecting the rows in which the "Sport" column is equal to the sport selected in this iteration. We can call the bar method of the Axes we created for this plot. As before, it is called with the string that holds the name of the sport as the first argument, the mean method of the "Height" column is set to be the height of the bar and an error bar is set to be equal to the standard deviation of the values in the column. After iterating over all of the sports, we exit the loop. We can then set the y-label to indicate the meaning of the height of each bar and we can set the x-axis tick labels to be equal to the names of the sports.

Figure derived automatically from the data



6. Figure derived automatically from the data
This is what this figure would look like.
Importantly, at no point during the creation of this figure did we need to know how many different sports are recorded in the DataFrame. Our code would automatically add bars or reduce the number of bars, depending on the input data.

Practice automating visualizations!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

Where to go next

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB

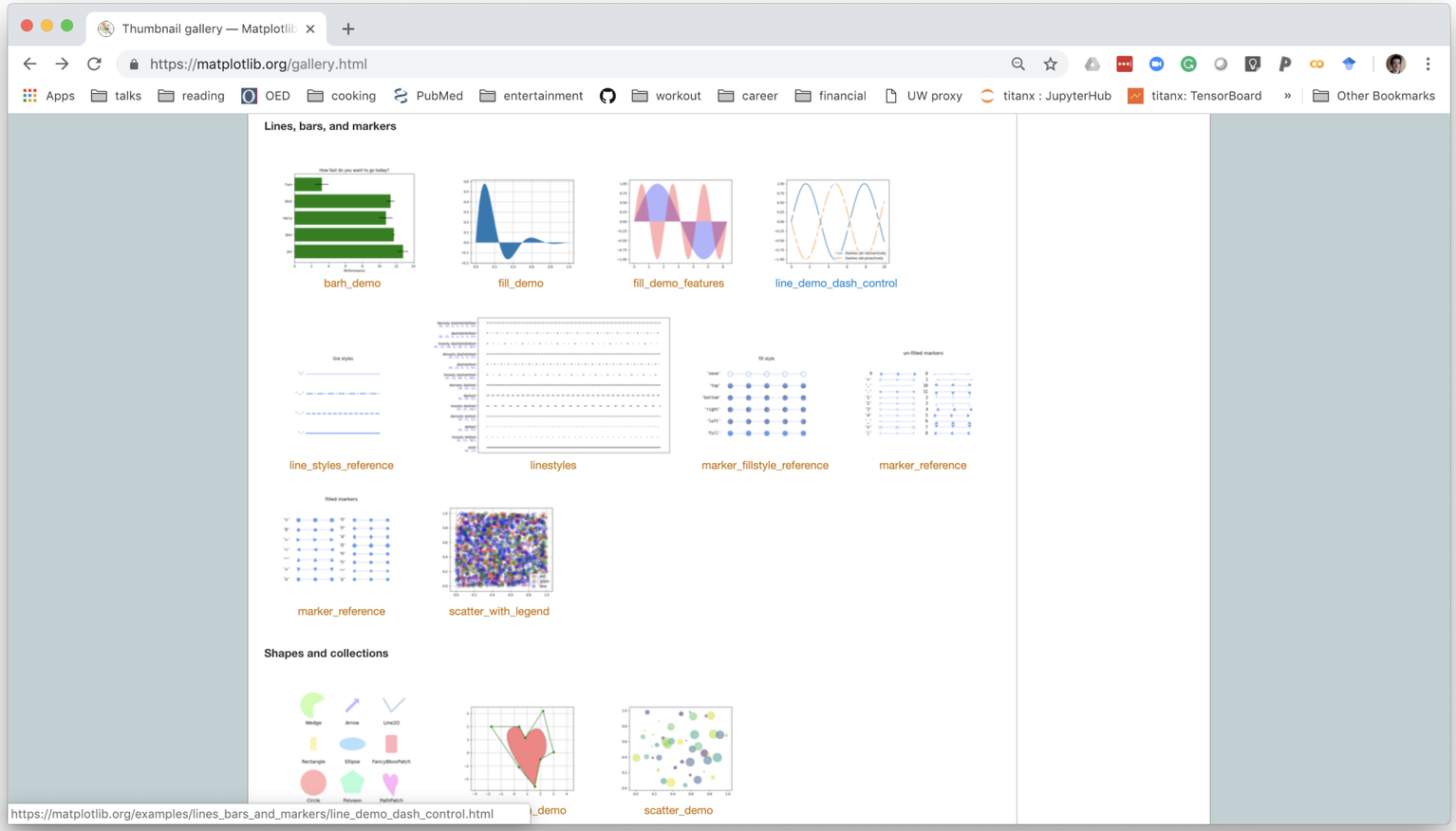
Ariel Rokem
Data Scientist

datacamp

The Matplotlib gallery

https://matplotlib.org/gallery.html

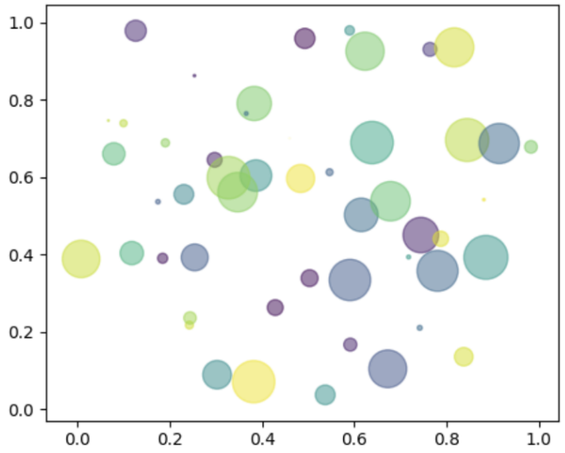
Gallery of examples



Example page with code

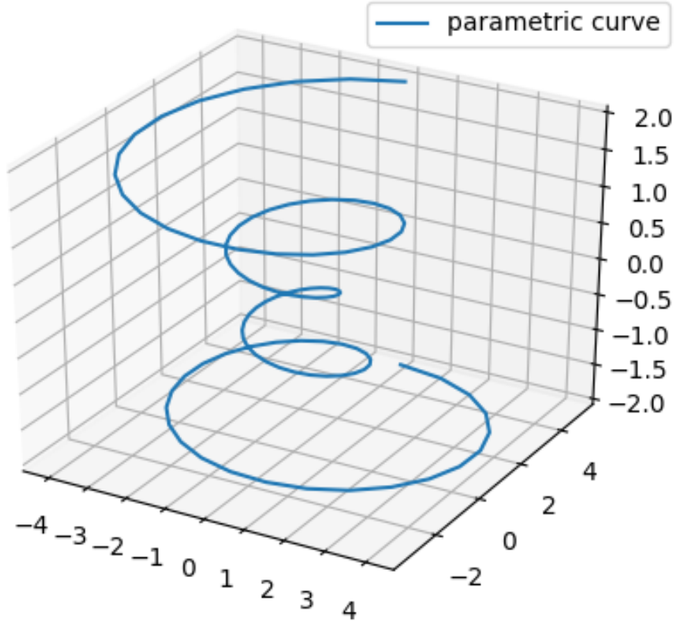
shapes_and_collections example code: scatter_demo.py

(Source code, png, pdf)



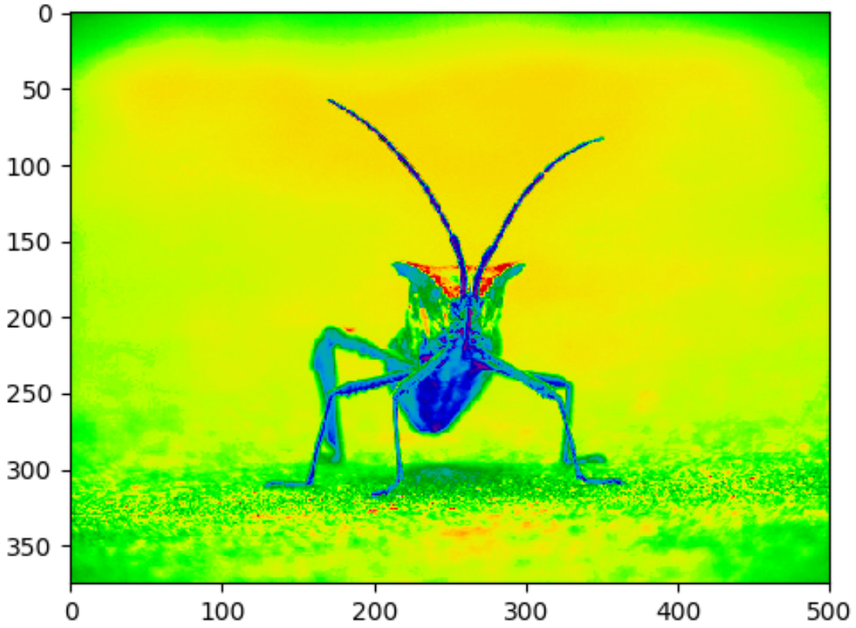

```
"""
Simple demo of a scatter plot.
"""
import numpy as np
import matplotlib.pyplot as plt


N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (15 * np.random.rand(N))**2  # 0 to 15 point radii

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```

Plotting data in 3D



https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html

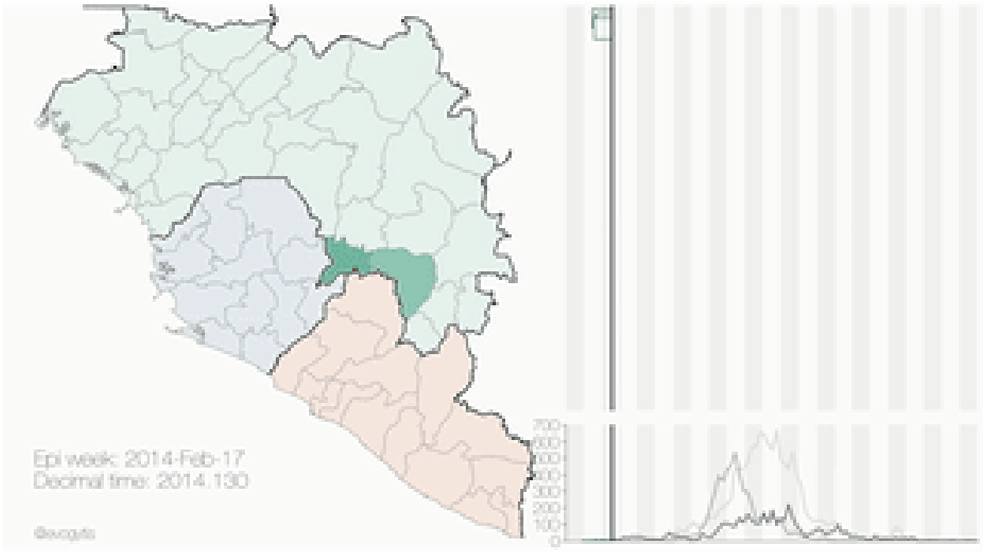
Visualizing images with pseudo-color



6. Visualizing images with pseudo-color
Another capability of Matplotlib is visualizing data from images. For example, here is an image visualized using pseudo-color, where each value in the image is translated into a color. You can learn more about working with images in this URL.

https://matplotlib.org/users/image_tutorial.html

Animations

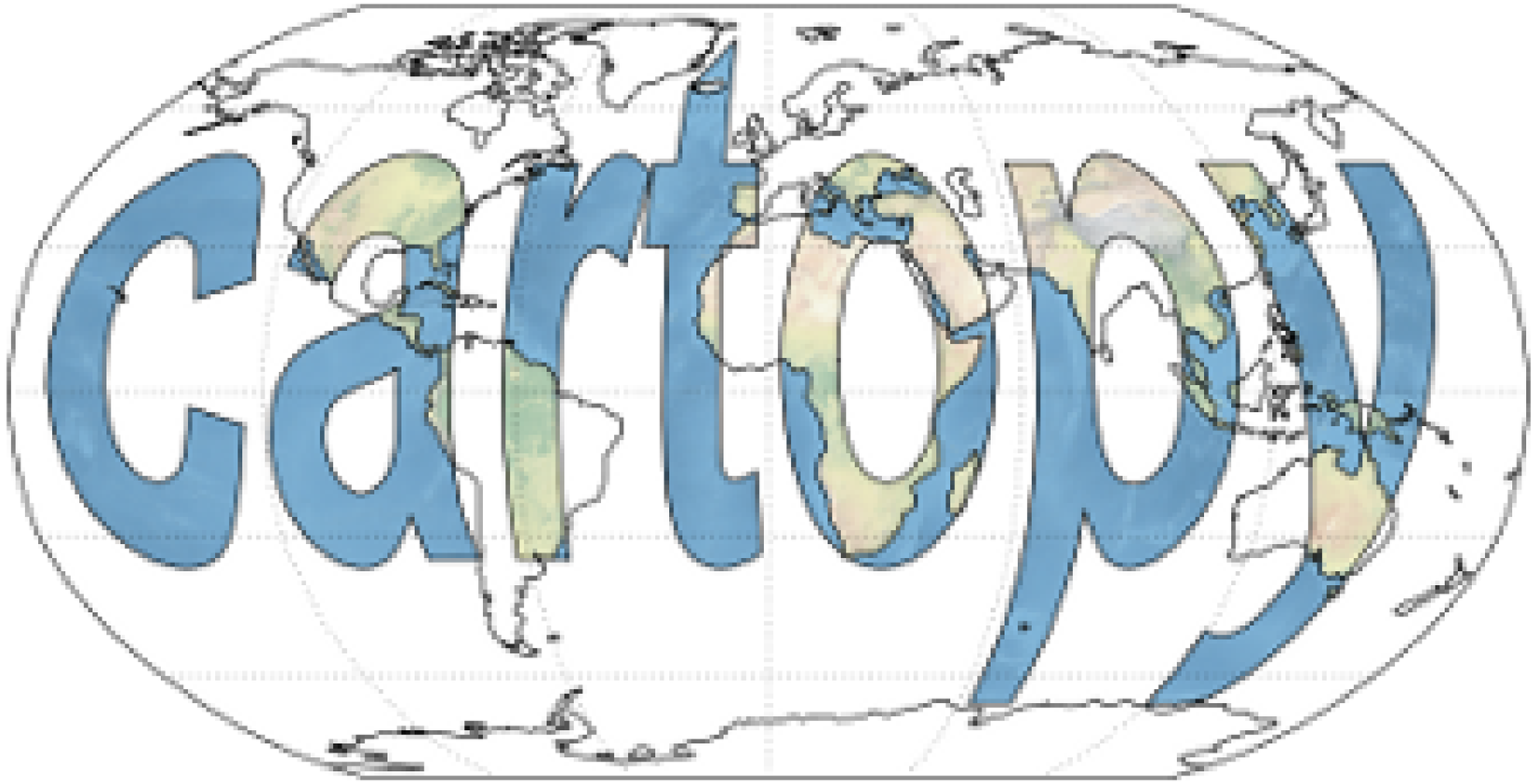


7. Animations
You might remember this visualization that I showed you in the very first lesson of this course. It used one more dimension, time, by varying the display through animation. You can create animations by creating multiple frames of the movie, each as its own visualization, and then stitching them together into a movie using tools such as Quicktime, but Matplotlib also has its own interface for creating animations. You can learn about this interface at this URL.

Image credit: **Gytis Dudas** and **Andrew Rambaut**

https://matplotlib.org/api/animation_api.html

Using Matplotlib for geospatial data

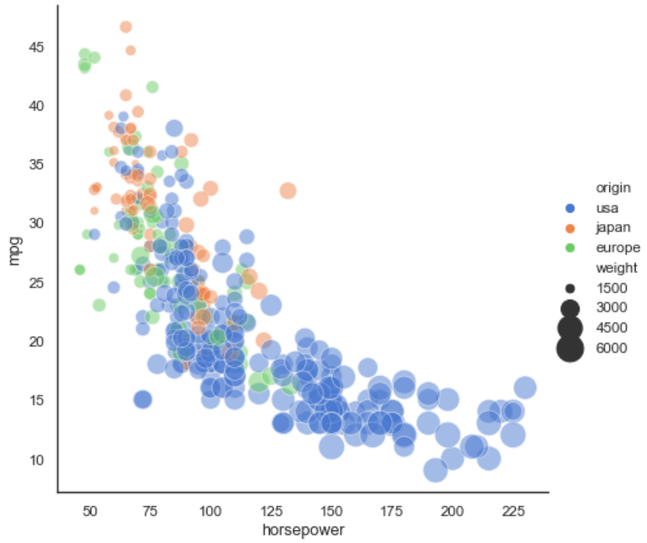


8. Using Matplotlib for geospatial data
There are multiple software packages that extend Matplotlib's capability to a variety of different kinds of data. For example, Cartopy extends Matplotlib to be used with geospatial data, such as maps.

https://scitools.org.uk/cartopy/docs/latest/

Pandas + Matplotlib = Seaborn

```
seaborn.relplot(x="horsepower", y="mpg", hue="origin", size="weight",
                sizes=(40, 400), alpha=.5, palette="muted",
                height=6, data=mpg)
```



9. Pandas + Matplotlib = Seaborn
Another library that extends Matplotlib is Seaborn. This library creates very sophisticated statistical visualizations from Pandas data structures, such as DataFrames. The nice thing about Seaborn is that you can create elegant and sophisticated visualizations of your data with very little code. For example, this code would create this visualization that encodes the fuel efficiency of cars as a function of their horsepower, but also encodes the country in which the car was manufactured, using the color of the bubbles, as well as their weight, using the size of each bubble.

Seaborn example gallery

https://seaborn.pydata.org/examples/index.html

Good luck visualizing your data!

INTRODUCTION TO DATA VISUALIZATION WITH MATPLOTLIB