Data Visualization, matplotlib & Simple Data Analysis

CS11 Computer Programming
Department of Computer Science
Wellesley College

Visualizing data is important!
What careers do Wellesley graduates choose based on their major?

Can we generate this visualization with cs1graphics?
NO. We need a tool that works with data.

https://www.wellesley.edu/admission/why/after

Plotting with matplotlib
A plot is a graphical technique for representing a data set, usually as a graph showing the relationship between two or more variables.

In CS11, we’ll be using Python’s matplotlib library to make plots/graphs/charts.

import matplotlib.pyplot as plt
import numpy as np

In all our examples, we will need to import the numpy and matplotlib.pyplot modules.
These come with Canopy – no need to download anything.

Resources
matplotlib examples: http://matplotlib.org/examples
pyplot documentation: http://matplotlib.org/api/pyplot_summary.html

matplotlib can make these plots

see more at http://matplotlib.org/gallery.html
... and these

Starting simple: \texttt{plot} function

When a single value (e.g. 10) is provided, it's considered the y-value of a data point and is assigned x to 0. We can specify how the marker for the point displays.

\begin{verbatim}
plt.figure(figsize=(3, 3))
plt.plot(10, color='red', marker='s')
plt.show()
\end{verbatim}

The function \texttt{plot} specifies many named parameters (or keyword arguments) to control how a data point is displayed.

\begin{verbatim}
plt.figure(figsize=(3, 3))
plt.plot(10, color='red', marker='o', markersize=60, markeredgecolor='blue', markeredgewidth=10)
plt.show()
\end{verbatim}

Plotting a line

A list of two values is displayed as line. Again, values are y-coordinates, and by default x is set to [0, 1]. Some arguments have default values, for example, line color is blue by default and the style is solid (see second plot).

\begin{verbatim}
plt.figure(figsize=(3, 3))
plt.plot([10, 20], linestyle='dashdot', linewidth=5)
plt.show()
\end{verbatim}

But we can specify two lists: one with the x-coordinates and one with the y-coordinates for the data points. Then they are automatically connected by lines.

\begin{verbatim}
plt.figure(figsize=(3, 3))
plt.plot([1, 3, 5, 7, 9], [1, 5, 1, 5, 1])
plt.show()
\end{verbatim}

The \texttt{plot} function

Accessing the documentation of \texttt{plot} via: \texttt{help(plt.plot)} reveals its versatility:

In [46]: help(plt.plot)
Help on function plot in module matplotlib.pyplot:

plot(*args, **kwargs)
   Plot lines and/or markers to the
   :class:`Axes`. *args* is a variable length argument, allowing for multiple *x*, *y* pairs with an
   optional format string. For example, each of the following is legal:

   - plot(x, y)  # plot x and y using default line style and color
   - plot(x, y, 'bo')  # plot x and y using blue circle markers
   - plot(y)  # plot y using x as index array 0..N-1
   - plot(y, 'r+')  # ditto, but with red plusses

Notice string argument values such as 'bo', 'r+' and 'g.' for the marker/line appearance (first letter is the color, second character the marker or line style).

Also, many common keyword arguments have shorter names: color is c, markersize is ms, linewidth is lw, linestyle is ls, etc.
Controlling axes

\[ x = [1, 3, 5, 7, 9] \] and \[ y = [1, 5, 1, 5, 1] \]

The axes automatically fit the provided range values. We can control them in different ways with the functions:

\[
\text{axis}([\text{xmin}, \text{xmax}, \text{ymin}, \text{ymax}]) \quad \# \text{ also, 'off', and 'equal'}
\]

\[
\text{xlim}(\text{xmin}, \text{xmax})
\]

\[
\text{ylim}(\text{ymin}, \text{ymax})
\]

```python
plt.plot(x, y,'r-.', lw=5)
plt.axis([0, 10, 0, 6])
plt.plot(x, y,'m:', lw=4)
plt.xlim((-5, 15))
plt.plot(x, y,'g--', lw=3)
plt.ylim((-10, 10))
```

Decorating the Plot

The functions \text{xticks} and \text{yticks} control the location and values of ticks on axes. We can supply text labels for the axes with \text{xlabel} and \text{ylabel} and a \text{title} for the plot.

```python
x = [1, 3, 5, 7, 9]
y = [1, 5, 1, 5, 1]
plt.figure(figsize=(4, 4))
plt.plot(x, y,'r-.', lw=5)
plt.axis([0, 10, 0, 6])
plt.xticks([1, 5, 9],['x:p1', 'x:p3', 'x:p5'])
plt.yticks([1, 5],['y:p1,p3,p4', 'y:p2,p4'], rotation=90)
plt.xlabel("width for letter M")
plt.ylabel("height for letter M")
plt.title("Plotting the letter M", fontsize=14)
plt.show()
```

More commands and plot types

Before \text{plt.show}, you can use \text{plt.savefig("filename")} to save the plot in a file. Use .png as file ending. Don’t put savefig after show, it saves an empty file.

This is a \text{scatter} plot with random points and colors. Notice \text{alpha} to make colors half-transparent. Uses the powerful \text{numpy} library.

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

Plotting Overview

1. Plots are 2-dimensional.
2. We need to provide x and y coordinates for the points to be drawn.
3. We will create plots by using functions that expect two lists of values for x and y.
4. Usually we have a set of points x and we will apply a math function (square, sin, etc.) to get the corresponding y values. Thus, we need the \text{mapping} pattern.
5. These functions take keyword arguments that control the appearance of the data points: color, marker shape, line type, line width, etc.
6. The \text{pyplot} library (\text{plt} for short) and its functions create and manipulate a single figure object that it’s updated every time we invoke a new function.
7. This object is implicit, we don’t access it directly. This is why we can write several statements without an explicit mention of this object:

\[
\text{plt.plot(x, y)}
\]

\[
\text{plt.xlabel("some label")}
\]

\[
\text{plt.xlim([10, 50])}
\]

\[
\text{plt.show()}
\]

\text{plt} is not an object, it’s a module.

Verify it with \text{type(plt)}.

You can see the list of all available functions and classes with \text{dir(plt)}. 
Mapping & Filtering with List Comprehension

A shorthand way of implementing the mapping and filtering patterns.

THE OLD WAY

```python
squares = []
for x in range(5):
    squares.append(x**2)
```

NEW with list comprehensions

```python
squares = [x**2 for x in range(5)]
```

How to write list comprehensions?

THE OLD WAY

```python
evens = []
for x in range(10):
    if x%2 == 0:
        evens.append(x)
```

NEW with list comprehensions

```python
evens = [x for x in range(10) if x%2 == 0]
```

Plotting (with list comprehensions)

```python
# Preparing data for plotting
xvals = range(-5, 11)
# [-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10]
yvals = [x**2 for x in xvals]
# [25,16,9,4,1,0,1,4,9,16,25,36,49,64,81,100]
```

```python
plt.style.use('ggplot')
plt.figure(figsize=(5, 5))
plt.plot(xvals, yvals, 'go-', lw=3)
plt.axis([-8, 12, -10, 110])
plt.title("A square relation")
plt.show()
```

More plotting (with list comprehensions)

```python
xvals = range(-5, 11)
# [-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10]
y = x**2 - 4*x
```

```python
plt.style.use('ggplot')
plt.plot(xvals, y, 'm:D')
plt.show()
```

Given the mapping function:

```python
y = x**2 - 4*x
```

To make it easier for you, we're showing you how it should look like.

The string value for the line/marker style is "m:D" short for “magenta, dotted line, diamond marker”
Remember the list of English words from PS5. Here are some fun exercises to see the power of list comprehension.

```python
from vocabulary import englishwords

Ex 1: Generate the list of lengths of all words:
wordLengths = [
]

Ex 2: Generate the list of all words shown as uppercase (use the string method upper).
upperWords = [
]

Ex 3: Make use of isVowel to create the list of all words that start with a vowel.
(start there are 12119 such words in the list). Use filtering!
startsWithVowel = [
]

Ex 4: Generate the list of all words with length of 5.
(there are 4684 such words, e.g., brick, comet, genes, etc.)
wordLength5 = [
]

Ex 5: Generate the list of all words that start and end with a vowel.
(there are 1788 such words, e.g., able, above, absence, etc.)
vowelWords = [
]

Ex 6: Generate the list of all words that start and end with the same letter.
(there are 4450 such words, e.g. yearly, suntans, scripts, etc.)
sameStartEnd = [
]
```

### Analyzing English words [1]

**Problem:** Find the frequency of words starting with a vowel.

**Solution 1:** Combine a dictionary with a list comprehension.

```python
vDct = {}
for v in "aeiou":
    vDct[v] = len([w for w in englishwords if w[0] == v])
```

**Solution 2:** Only use a dictionary.

```python
vDct = {}
for w in englishwords:
    if isVowel(w[0]):
        vDct[w[0]] = vDct.get(w[0], 0) + 1
```

In []: vDct
Out[]: `{‘a’: 3683, ‘e’: 2783, ‘i’: 2809, ‘o’: 1632, ‘u’: 1212}`

### Analyzing English words [2]

**Problem:** Find the frequency of words starting with each letter of the alphabet.

**Solution 1:** Combine a dictionary with a list comprehension.

```python
freqDct = {}
from string import lowercase
for c in lowercase:
    freqDct[c] = len([w for w in englishwords if w[0] == c])
```

**Solution 2:** Only use a dictionary.

```python
freqDct = {}
```
Inefficiency of nested loops

Measuring execution time. In the notebook we show two snippets of code to measure the length of execution for the two solutions of Slide 20.

```python
import time
t1 = time.time()  # starting moment
# some statements to execute
t2 = time.time()  # end moment
print t2 - t1  # elapsed time between two moments
```

Solution 1 is about 10 times slower than Solution 2. The reason is that in Solution 1 we have a hidden nested loop (within the list comprehension), thus, we are iterating 26 times over 66230 words. In Solution 2, we are only doing one single iteration over all the words.

When possible, change your solution to contain a single loop. But, there are cases in which you cannot avoid nested loops.

---

Plotting the results

**Task**: Generate a bar chart with the frequency of words starting with each letter.

**Code for the Bar Chart**

```python
from numpy import arange
from string import lowercase

# get the values in alphabetical order
counts = [frDct[char] for char in lowercase]

plt.figure(figsize=(10, 6))  # make figure of certain size
plt.bar(range(26), counts)  # bar chart needs indices in x
plt.xlim((0,26))  # limit the x axis to a certain range
plt.xticks(arange(0.5, 26.5), list(lowercase))  # set labels for axes and figure title
plt.xlabel('letters')
plt.ylabel('word counts')
plt.title('Frequency of English words by letter', fontsize=16)
```

---

**Challenge: Cumulative Distribution**

Changing the statement:

```python
plt.bar(range(26), counts)
```

to

```python
plt.step(range(26), counts)
```

creates a stepwise chart similar to the one below. However, this plot depicts a cumulative distribution, which is different from a histogram.

**Your task**: Figure out how to calculate the y-values.

**Hint**: you can use a function that we studied in Lecture 9 (about list patterns). If you plug-in the new values for `counts` in the code from slide 11-23 and change the axis labels, you'll get this plot.