PLOTTING
(download slides and .py files to follow along)

6.100L Lecture 25

Ana Bell
WHY PLOTTING?

- Sooner or later, everyone needs to produce plots
  - Helps us visualize data to see trends, pose computational questions to probe
  - If you join 6.100B, you will make extensive use of them
  - For those of you leaving us after next week, this is a valuable way to visualize data

- Example of leveraging an existing library, rather than writing procedures from scratch

- Python provides libraries for:
  - Plotting
  - Numerical computation
  - Stochastic computation
  - Many others
MATPLOTLIB

- Can **import library** into computing environment
  
  ```
  import matplotlib.pyplot as plt
  ```
  
  - Allows **code to reference library** procedures as `plt.<processName>`

- Provides access to existing set of graphing/plotting procedures

- Today will just show some simple examples; lots of additional information available in documentation associated with `matplotlib`

- Will see many other examples and details of these ideas if you take 6.100B
A SIMPLE EXAMPLE

- Idea – create different functions of a variable (n), and visualize their differences

```python
nVals = []
linear = []
quadratic = []
cubic = []
exponential = []

for n in range(0, 30):
    nVals.append(n)
    linear.append(n)
    quadratic.append(n**2)
    cubic.append(n**3)
    exponential.append(1.5**n)
```

Lists of values of functions of variable

List of values of variable

Used 1.5 to keep displays visible, more common value for order of growth example would be 2
To generate a plot:

```
plt.plot(<x values>, <y values>)
```

- Arguments are lists (or sequences) of numbers
  - Lists must be of the same length
  - Generates a sequence of <x, y> values on a Cartesian grid
  - Plotted in order, then connected with lines

- Can change iPython console to **generate plots in a new window** through Preferences
  - Inline in the console
  - In a new window
EXAMPLE

```python
plt.plot(nVals, linear)
```

Note how `matplotlib` automatically fits plot within frame.
ORDER OF POINTS MATTERS

- Suppose I create a set of values for $n$ and for $n^2$, but in arbitrary order
- Python plots using the order of the points and connecting consecutive points
UNORDERED EXAMPLE

testSamples = [0, 5, 3, 6, 15, 2, 1, 4, 25, 20, 7, 21, 22, 23, 9, 8, 24, 10, 12, 11]
testValues = [0, 25, 9, 36, 225, 4, 1, 16, 625, 400, 49, 441, 484, 529, 81, 64, 576, 100, 144, 121]

## plot connects the points
plt.plot(testSamples, testValues)
SCATTER PLOT DOES NOT CONNECT DATA POINTS

testSamples = [0, 5, 3, 6, 15, 2, 1, 4, 25, 20, 7, 21, 22, 23, 9, 8, 24, 10, 12, 11]  
tenValues = [0, 25, 9, 36, 225, 4, 1, 16, 625, 400, 49, 441, 484, 529, 81, 64, 576, 100, 144, 121]  
## scatter plot does not connect the points  
plt.scatter(testSamples, testValues)
SHOWING ALL DATA ON ONE PLOT

```python
plt.plot(nVals, linear)
plt.plot(nVals, quadratic)
plt.plot(nVals, cubic)
plt.plot(nVals, exponential)
```

Impossible to see linear graph, or even quadratic graph

Problem is that scales are very different
PRODUCING MULTIPLE PLOTS

- Let’s graph each one in separate frame/window
- Call
  ```python
  plt.figure(<arg>)
  ```
  - Creates a new display with that name if one does not already exist
  - If a display with that name exists, reopens it for additional processing

  gives a name to this figure; allows us to reference for future use
EXAMPLE CODE

```python
plt.figure('expo')
plt.plot(nVals, exponential)
plt.figure('lin')
plt.plot(nVals, linear)
plt.figure('quad')
plt.plot(nVals, quadratic)
plt.figure('cube')
plt.plot(nVals, cubic)
newExpo = []
for i in range(30):
    newExpo.append(1.6**i)
plt.figure('expo')
plt.plot(nVals, newExpo)
```

New figure with name expo
Plot inside that figure
New figure with name lin
Plot inside that figure
Make another exponential function
Go back to expo
Add another plot to that figure
DISPLAY OF quad
DISPLAY OF cube
DISPLAY OF $\text{lin}$
Note how matplotlib automatically scales to fit both plots within frame.
A “REAL” EXAMPLE

```python
months = range(1, 13, 1)
temps = [28, 32, 39, 48, 59, 68, 75, 73, 66, 54, 45, 34]
plt.plot(months, temps)
```

matplotlib has automatically selected x and y scales to best fit data

But what is this trying to tell us? Suppose I just showed you the graph; how do you know its meaning?
A "REAL" EXAMPLE

months = range(1, 13, 1)
temps = [28, 32, 39, 48, 59, 68, 75, 73, 66, 54, 45, 34]
plt.plot(months, temps)

plt.title('Ave. Temperature in Boston')
plt.xlabel('Month')
plt.ylabel('Degrees F')

Still a bit weird looking
A “REAL” EXAMPLE

```python
months = range(1, 13, 1)
temps = [28, 32, 39, 48, 59, 68, 75, 73, 66, 54, 45, 34]
plt.plot(months, temps)
plt.title('Ave. Temperature in Boston')
plt.xlabel('Month')
plt.ylabel('Degrees F')
plt.xlim(1, 12)
```

This sets limits on display for x axis

Ave. Temperature In Boston

Suppose I want to see each month on x-axis?
A "REAL" EXAMPLE

months = range(1, 13, 1)
temps = [28, 32, 39, 48, 59, 68, 75, 73, 66, 54, 45, 34]
plt.plot(months, temps)
plt.title('Ave. Temperature in Boston')
plt.xlabel('Month')
plt.ylabel('Degrees F')
plt.xticks((1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12))

This specifies which x values to mark

But what about those who can’t map numbers to months?
A \textit{“REAL” EXAMPLE}

\begin{verbatim}
plt.xticks((1,2,3,4,5,6,7,8,9,10,11,12),
           ('Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'))
\end{verbatim}
ADDING GRID LINES

Can toggle grid lines on/off with `plt.grid()`
 LET’S ADD ANOTHER CITY

```python
months = range(1, 13, 1)
boston = [28, 32, 39, 48, 59, 68, 75, 73, 66, 54, 45, 34]
plt.plot(months, boston)
phoenix = [54, 57, 61, 68, 77, 86, 91, 90, 84, 73, 61, 54]
plt.plot(months, phoenix)
# Add labels and title
plt.title('Ave. Temperatures')
plt.xlabel('Month')
plt.ylabel('Degrees F')
```
BUT WHERE AM I?
LET’S ADD ANOTHER CITY

```python
months = range(1, 13, 1)
boston = [28, 32, 39, 48, 59, 68, 75, 73, 66, 54, 45, 34]
plt.plot(months, boston, label = 'Boston')
phoenix = [54, 57, 61, 68, 77, 86, 91, 90, 84, 73, 61, 54]
plt.plot(months, phoenix, label = 'Phoenix')
# Add labels and title
plt.title('Ave. Temperatures')
plt.xlabel('Month')
plt.ylabel('Degrees F')
plt.legend(loc = 'best')
```
PLOT WITH TWO CURVES

Note: Python picked different colors for each plot; we could specify if we wanted.
CONTROLLING PARAMETERS

- Suppose we want to control **details of the displays**
- Examples:
  - Changing **color** or style of data sets
  - Changing **width** of lines or displays
  - Using **subplots**
- Can provide a “format” argument to plot
  - “marker”, “line”, “color”
  - Can skip any of these choices, plot takes default
  - Order doesn’t matter, as no confusion between symbols
CONTROLLING COLOR AND STYLE

```python
months = range(1, 13, 1)
boston = [28, 32, 39, 48, 59, 68, 75, 73, 66, 54, 45, 34]
plt.plot(months, boston, 'b-', label = 'Boston')
phoenix = [54, 57, 61, 68, 77, 86, 91, 90, 84, 73, 61, 54]
plt.plot(months, phoenix, 'r--', label = 'Phoenix')
msp = [16, 19, 34, 48, 59, 70, 75, 73, 64, 60, 37, 21]
plt.plot(months, msp, 'g-.', label = 'Minneapolis')
plt.legend(loc = 'best', fontsize=20)
```
CONTROLLING COLOR AND STYLE

Ave. Temperatures

- Boston
- Phoenix
- Minneapolis

Degrees F

Month

Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
USING KEYWORDS

```python
months = range(1, 13, 1)
boston = [28, 32, 39, 48, 59, 68, 75, 73, 66, 54, 45, 34]
plt.plot(months, boston, label = 'Boston',
        color = 'b', linestyle = '-')
phoenix = [54, 57, 61, 68, 77, 86, 91, 90, 84, 73, 61, 54]
plt.plot(months, phoenix, label = 'Phoenix',
        color = 'r', linestyle = '--')
msp = [16, 19, 34, 48, 59, 70, 75, 73, 64, 60, 37, 21]
plt.plot(months, msp, label = 'Minneapolis',
        color = 'g', linestyle = '-.')
plt.legend(loc = 'best', fontsize=20)
plt.title('Ave. Temperatures')
plt.xlabel('Month')
plt.ylabel('Degrees F')
plt.xticks((1,2,3,4,5,6,7,8,9,10,11,12),
            ('Jan','Feb','Mar','Apr','May','Jun',
             'Jul','Aug','Sep','Oct','Nov','Dec'))
```
CONTROLLING COLOR AND STYLE
LINE, COLOR, MARKER OPTIONS

- **Line Style**
  - `--` solid line
  - `--` dashed line
  - `--` dash dot line
  - `:` dotted line

- **Color Options (plus many more)**
  - `b` blue
  - `g` green
  - `r` red
  - `c` cyan
  - `m` magenta
  - `y` yellow
  - `k` black
  - `w` white

- **Marker Options (plus many more)**
  - `.` point
  - `o` circle
  - `v` triangle down
  - `^` triangle up
  - `*` star
months = range(1, 13, 1)
boston = [28,32,39,48,59,68,75,73,66,54,45,34]
plt.plot(months, boston, '.b-', label = 'Boston')
phoenix = [54,57,61,68,77,86,91,90,84,73,61,54]
plt.plot(months, phoenix, 'or-', label = 'Phoenix')
msp = [16,19,34,48,59,70,75,73,64,60,37,21]
plt.plot(months, msp, '*g-.', label = 'Minneapolis')
plt.legend(loc = 'best', fontsize=20)
WITH MARKERS

Note how actual points being plotted are now marked.
CONTROLLING LINE WIDTH

```python
months = range(1, 13, 1)
boston = [28, 32, 39, 48, 59, 68, 75, 73, 66, 54, 45, 34]
plt.plot(months, boston, label = 'Boston',
        color = 'b', linestyle = '-', linewidth = 2)
phoenix = [54, 57, 61, 68, 77, 86, 91, 90, 84, 73, 61, 54]
plt.plot(months, phoenix, label = 'Phoenix',
        color = 'r', linestyle = '--', linewidth = 10)
msp = [16, 19, 34, 48, 59, 70, 75, 73, 64, 60, 37, 21]
plt.plot(months, msp, label = 'Minneapolis',
        color = 'g', linestyle = '-.', linewidth = 20)
plt.legend(loc = 'best', fontsize=20)
```
MANY OTHER OPTIONS

- Using the linewidth keyword (in pixels)
PLOTS WITHIN PLOTS

months = range(1, 13, 1)
boston = [28, 32, 39, 48, 59, 68, 75, 73, 66, 54, 45, 34]
plt.subplot(2,1,1)
plt.plot(months, boston, 'b-')
plt.ylabel('Degrees F')
plt.title('Boston vs. Phoenix')
plt.xticks((1,2,3,4,5,6,7,8,9,10,11,12),
           ('Jan','Feb','Mar','Apr','May','Jun',
            'Jul','Aug','Sep','Oct','Nov','Dec'))

phoenix = [54,57,61,68,77,86,91,90,84,73,61,54]
plt.subplot(2,1,2)
plt.plot(months, phoenix, 'r--')
plt.ylabel('Degrees F')
plt.xticks((1,2,3,4,5,6,7,8,9,10,11,12),
           ('Jan','Feb','Mar','Apr','May','Jun',
            'Jul','Aug','Sep','Oct','Nov','Dec'))
AND THE PLOT THICKENS

But this can be misleading?

Y scales are different!
PLOTS WITHIN PLOTS

```python
months = range(1, 13, 1)
boston = [28, 32, 39, 48, 59, 68, 75, 73, 66, 54, 45, 34]
plt.subplot(2,1,1)
plt.ylim(0, 100)
plt.plot(months, boston, 'b-')
plt.ylabel('Degrees F')
plt.title('Boston vs. Phoenix')
plt.xticks((1,2,3,4,5,6,7,8,9,10,11,12),
             ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
              'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
phoenix = [54, 57, 61, 68, 77, 86, 91, 90, 84, 73, 61, 54]
plt.subplot(2,1,2)
plt.ylim(0, 100)
plt.plot(months, phoenix, 'r--')
plt.ylabel('Degrees F')
plt.xticks((1,2,3,4,5,6,7,8,9,10,11,12),
             ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
              'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
```

Fix y axis so plots are similar.
AND THE PLOT THICKENS
LOTS OF SUBPLOTS

```python
boston = [28,32,39,48,59,68,75,73,66,54,45,34]
plt.subplot(2,2,1)
plt.ylim(0, 100)
plt.plot(months, boston, 'b-')
plt.ylabel('Degrees F')
plt.title('Boston')
plt.xticks((1,3,5,7,9,11),('Jan', 'Mar', 'May', 'Jul', 'Sep', 'Nov'))

phoenix = [54,57,61,68,77,86,91,90,84,73,61,54]
plt.subplot(2,2,2)
plt.ylim(0, 100)
plt.plot(months, phoenix, 'r--')
plt.ylabel('')
plt.title('Phoenix')
plt.xticks((1,3,5,7,9,11),('Jan', 'Mar', 'May', 'Jul', 'Sep', 'Nov'))

msp = [16,19,34,48,59,70,75,73,64,60,37,21]
plt.subplot(2,2,3)
plt.ylim(0, 100)
plt.plot(months, msp, 'g-.')
plt.ylabel('Degrees F')
plt.title('Minneapolis')
plt.xticks((1,3,5,7,9,11),('Jan', 'Mar', 'May', 'Jul', 'Sep', 'Nov'))
```

6.100L Lecture 25
AND THE PLOT THICKENS

Boston vs. Phoenix vs. Minneapolis

Degrees F

Jan  Mar  May  Jul  Sep  Nov

Degrees F

Jan  Mar  May  Jul  Sep  Nov

6.100L Lecture 25
US POPULATION EXAMPLE
A MORE INTERESTING EXAMPLE

- Let’s try plotting some more complicated data
- We have provided a file with the US population recorded every 10 years for four centuries
- Would like to use plotting to examine that data
  - Use plotting to help visualize trends in the data
  - Use plotting to raise questions that might be tested computationally (you’ll see much more of this if you take 6.100B)
THE INPUT FILE
USPopulation.txt

1610 350
1620 2,302
1630 4,646
1640 26,634
1650 50,368
1660 75,058
1670 111,935
1680 151,507
1690 210,372
1700 250,888
1710 331,711
1720 466,185
1730 629,445
1740 905,563
...
1960 179,323,175
1970 203,211,926
1980 226,545,805
1990 248,709,873
2000 281,421,906
2010 308,745,538
PLOTTING THE DATA

```python
def getUSPop(fileName):
    inFile = open(fileName, 'r')
    dates, pops = [], []
    for l in inFile:
        line = ''
        for c in l:
            if c in '0123456789 ':  # Remove commas for each line
                line += c
        line = line.split(' ')  # Split into date and population
        dates.append(int(line[0]))
        pops.append(int(line[1]))
    return dates, pops

dates, pops = getUSPop('lec25_USPopulation.txt')
plt.plot(dates, pops)
plt.title('Population in What Is Now U.S.\n\n  (Native Am. Excluded Before 1860)')
plt.xlabel('Year')
plt.ylabel('Population')
```

- Remove commas for each line
- Split into date and population
- Convert to ints, and add to lists
POPULATION GROWTH


Visualizing data can expose things not easily seen in raw data

What’s going on in the early years? Could I visualize this differently?

Impact of WWII
Impact of Civil War
CHANGING THE SCALING

```
dates, pops = getUSPop('USPopulation.txt')
plt.plot(dates, pops)
plt.title('Population in What Is Now U.S\n' +
          '(Native Am. Excluded Before 1860)')
plt.xlabel('Year')
plt.ylabel('Population')
plt.semilogy()
```

Use log scale on y axis

Log scale means each increment along axis corresponds to exponential increase in size; while in normal scale each increment corresponds to linear increase in size.
POPULATION GROWTH

What does linear growth on a log scale mean?


Can now see that there was growth early on, actually at a faster rate than later years.
WHICH DO YOU FIND MORE INFORMATIVE?

Visualization can raise questions: for ex. by eye, it appears that there are three different exponential growth periods.

Changing visualization can help expose trends in data not seen with standard plotting.

Visualization can raise questions: for ex. by eye, it appears that there are three different exponential growth periods.
COUNTRY POPULATION EXAMPLE
Interested in analyzing the population numbers. Don’t care about rank, country, or year.

<table>
<thead>
<tr>
<th></th>
<th>Country</th>
<th>Population</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>China</td>
<td>1,379,302,771</td>
<td>July 2017 est.</td>
</tr>
<tr>
<td>2</td>
<td>India</td>
<td>1,281,935,911</td>
<td>July 2017 est.</td>
</tr>
<tr>
<td>3</td>
<td>United States</td>
<td>326,625,791</td>
<td>July 2017 est.</td>
</tr>
<tr>
<td>4</td>
<td>Indonesia</td>
<td>260,580,739</td>
<td>July 2017 est.</td>
</tr>
<tr>
<td>5</td>
<td>Brazil</td>
<td>207,353,391</td>
<td>July 2017 est.</td>
</tr>
<tr>
<td>6</td>
<td>Pakistan</td>
<td>204,924,861</td>
<td>July 2017 est.</td>
</tr>
<tr>
<td>7</td>
<td>Nigeria</td>
<td>190,632,261</td>
<td>July 2017 est.</td>
</tr>
<tr>
<td>8</td>
<td>Bangladesh</td>
<td>157,826,578</td>
<td>July 2017 est.</td>
</tr>
<tr>
<td>9</td>
<td>Russia</td>
<td>142,257,519</td>
<td>July 2017 est.</td>
</tr>
<tr>
<td>10</td>
<td>Japan</td>
<td>126,451,398</td>
<td>July 2017 est.</td>
</tr>
</tbody>
</table>

...
def getCountryPops(fileName):
    inFile = open(fileName, 'r')
    pops = []
    for l in inFile:
        line = l.split('	')
        l = line[2]
        pop = ''
        for c in l:
            if c in '0123456789':
                pop += c
        pops.append(int(pop))
    return pops

pops = getCountryPops('lec25_countryPops.txt')

plt.plot(pops)
plt.title('Population Size of Countries July 2017')
plt.ylabel('Population')
plt.xlabel('Country Rank Based on Size')
plt.semilogy()
POPULATION SIZES

Population Size of Countries July 2017

- Y-axis: Population (log scale)
- X-axis: Country Rank Based on Size (log scale)

The graph shows a decreasing trend in population size as the country rank increases.
STRANGE INVESTIGATION: FIRST DIGITS

pops = getCountryPops('lec25_countryPops.txt')
firstDigits = []
for p in pops:
    firstDigits.append(int(str(p)[0]))

### Plot the first digits, as found in order in the file
plt.plot(firstDigits)

Why the saw tooth pattern?
Countries are in order of biggest pop to smallest pop.
First digit pattern is: 9,8,7,...,2,1,9,8,7,6,5,...
Benford’s Law

\[ P(d) = \log_{10}(1 + \frac{1}{d}) \]

Many datasets follow this:
- # social media followers
- Stock values
- Grocery prices
- Sports stats
- Building heights
- Taxes paid
COMPARING CITIES
EXAMPLE
AN EXTENDED EXAMPLE

- Let’s use another example to examine how plotting allows us to explore data in different ways, and how it provides a valuable way to visualize that data
- Won’t be looking at the code in detail
- Example data set
  - Mean daily temperature for each day for 55 years for 21 different US cities
  - Want to explore variations across years, and across cities
## THE DATA FILE

temperatures.csv

<table>
<thead>
<tr>
<th>CITY</th>
<th>TEMP</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEATTLE</td>
<td>3.1</td>
<td>19610101</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>0.55</td>
<td>19610102</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>0</td>
<td>19610103</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>4.45</td>
<td>19610104</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>8.35</td>
<td>19610105</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>6.7</td>
<td>19610106</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>9.7</td>
<td>19610107</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>7.2</td>
<td>19610108</td>
</tr>
<tr>
<td>SEATTLE</td>
<td>9.45</td>
<td>19610109</td>
</tr>
</tbody>
</table>

...
EXTRACTING DATA

This will return a list of temperatures (in F) and a corresponding list of dates for a specific city.

```python
def CtoF(c):
    return (c * 9/5) + 32

def getTempsForCity(city):
    inFile = open('temperatures.csv')
    temps = []
    dates = []
    for l in inFile:
        data = l.split(',' ',
        c = data[0]
        tem = data[1]
        date = data[2]
        if c == city:
            temps.append(CtoF(float(tem)))
            dates.append(date)
    return temps, dates
```
This will calculate the average temp over every day for 55 years, for every city.

def getAverageTemps():
cities = getCities()[1:]
xPts = range(len(cities))
aveTemp = []
cityLabels = []
for c in cities:
    temps, dates = getTempsForCity(c)
    aveTemp.append(sum(temps)/len(temps))
    cityLabels.append(c[0:2])
    print(c[0:2], sum(temps)/len(temps))

plt.figure('Temps')
plt.scatter(xPts, aveTemp)
plt.title('Ave. Temperatures')
plt.xlabel('City')
plt.ylabel(('Degrees F'))
plt.xticks(xPts, cityLabels)
AND THE TEMPERATURE IS ...

Ave. Temperatures

San Juan, Miami, Phoenix

Detroit, Chicago, Boston
BUT MORE INTERESTING TO LOOK AT CHANGE OVER TIME

For one city, calculate the average temperature over each year.

```python
def getTempsForYear(tem, dat, y):
    yearlyTemps = []
    for i in range(len(tem)):
        if y == dat[i][:4]:
            yearlyTemps.append(tem[i])
    return sum(yearlyTemps)/len(yearlyTemps), y

def getTempsByYearForCity(city):
    temps, dates = getTempsForCity(city)
    averages = []
    years = []
    for y in range(1961, 2016):
        tem = getTempsForYear(temps, dates, str(y))[0]
        averages.append(tem)
        years.append(str(y))
    return averages, years
```
BUT MORE INTERESTING TO LOOK AT CHANGE OVER TIME

Pick some cities to plot 55 temps (avg temp over each year)

```python
if True:
    plt.close()
    for c in ('BOSTON', 'PHOENIX', 'MIAMI', 'SAN DIEGO'):
        av, yr = getTempsByYearForCity(c)
        xPts = range(len(yr))
        plt.figure('Temps by City')
        plt.plot(xPts, av, label = c)
        plt.title('Ave. Temperatures')
        plt.xlabel('Years since 1961')
        plt.ylabel(('Degrees F'))
        plt.legend(loc = 'best')
```
BABY IT’S COLD OUTSIDE!

Ave. Temperatures

Degrees F

Years since 1961

BOSTON
PHOENIX
MIAMI
SAN DIEGO
BUT WHAT IS VARIATION?
high, low, avg temps by year

def getTempsForYearRange(tem, dat, y):
    yearly = []
    for i in range(len(tem)):
        if y == dat[i][4:8]:
            yearly.append(tem[i])
    return sum(yearly)/len(yearly), max(yearly), min(yearly), y

def getTempsByYearForCityRange(city):
    tems, dates = getTempsForCity(city)
    averages = []
    maxes = []
    mins = []
    years = []
    for y in range(1961, 2000):
        tem, mx, mn, y = getTempsForYearRange(temps, dates, str(y))
        averages.append(tem)
        maxes.append(mx)
        mins.append(mn)
        years.append(str(y))
    return averages, maxes, mins, years
BUT WHAT IS VARIATION?
high, low, avg temps by year

```python
if True:
    plt.close()
    for c in ('BOSTON',):  # try for BOSTON, SAN DIEGO, MIAMI
        av, mx, mn, yr = getTempsByYearForCityRange(c)
        xPts = range(len(yr))
        plt.figure('Temps by City')
        plt.plot(xPts, av, label = 'mean')
        plt.plot(xPts, mx, label = 'max')
        plt.plot(xPts, mn, label = 'min')
        plt.title('Temperature Range: ' + c)
        plt.xlabel('Years since 1961')
        plt.ylabel(('Degrees F'))
        plt.legend(loc = 'best')
```
SOME CITY EXAMPLES

- Can see range for each city
- Not helpful for comparison between cities
  - Y axis for Boston is 0 to 80
  - Y axis for Miami is 40 to 90
  - Y axis for San Diego is 50 to 90
USE SAME Y RANGE FOR ALL PLOTS

```python
if True:
    plt.close()
    for c in ('MIAMI',):  # try for BOSTON, SAN DIEGO, MIAMI
        av, mx, mn, yr = getTempsByYearForCityRange(c)
        xPts = range(len(yr))
        plt.figure('Temps by City')
        plt.ylim(0, 100)
        plt.plot(xPts, av, label = 'mean')
        plt.plot(xPts, mx, label = 'max')
        plt.plot(xPts, mn, label = 'min')
        plt.title('Temperature Range: ' + c)
        plt.xlabel('Years since 1961')
        plt.ylabel(('Degrees F'))
        plt.legend(loc = 'best')
```
BETTER CITY COMPARISON

- One reason to plot is to visualize data
- Can see that range of variation is quite different for Boston, compared to Miami or San Diego
- Can also see that mean for Miami much closer to max than min. Different from Boston and San Diego
HOW MANY DAYS AT A TEMP in 1961?

Set up a list of 100 elements, making a histogram-like structure.
- Index 0 stores how many days had a temp of 0
- Index 1 stores how many days had a temp of 1
- Index 99 stores how many days had a temp of 99.

```python
def getDayDistributionForCity(city, year):
    # assume a range of temperatures from 0 to 100
    temps, dates = getTempsForCity(city)
    newTemps = []
    for i in range(len(dates)):
        if year == dates[i][4:]:
            newTemps.append(temps[i])
    # want to map temp to number of occurrences
    d = [0] * 100
    for t in newTemps:
        tRound = round(t)
        d[tRound] += 1
    return d
```

Create a list of temperatures for a specific year
Count number of days of a particular year for which a specific temperature was the daily average
HOW MANY DAYS AT A TEMP IN 1961?

```python
if True:
    plt.close()
    for c in ('BOSTON',):  # try for BOSTON, SAN DIEGO, MIAMI
        ans = getDayDistributionForCity(c, '1961')
        temps = []
        for i in range(100):
            temps.append(i)
        plt.figure('Distribution of Temps by City')
        plt.bar(temps, ans)
        plt.title('Temperature Distribution: ' + c)
        plt.xlabel('Temperature')
        plt.ylabel(('Number of days'))
```
SAN DIEGO IS BORING?

Could we fit a curve to parts of this data? Uniform? Gaussian (aka bell)?
CHANGE OVER TIME?

Plot two distributions, one for 1961 and one for 2015

```python
if True:
    plt.close()
    for c in ('BOSTON',):  # try for BOSTON, SAN DIEGO
        plt.figure('Distribution of Temps by City')
        for y in ('1961', '2015'):
            ans = getDayDistributionForCity(c, y)
            temps = []
            for i in range(100):
                temps.append(i)
            if y == '1961':
                plt.bar(temps, ans, color = 'blue', label = y)
            else:
                plt.bar(temps, ans, color = 'red', label = y)
        plt.title('Temperature Distribution: ' + c)
        plt.xlabel('Temperature')
        plt.ylabel(('Number of days'))
        plt.legend(loc = 'best')
```
OVERLAY BAR CHARTS

Temperature Distribution: BOSTON

Number of days

Temperature

1961
2015
OR CAN PLOT SEPARATELY
CAN CONTROL LOTS OF OTHER THINGS

- Size of
  - Markers
  - Lines
  - Title
  - Labels
  - x and y ticks
- Scales of both axes
- Subplots
- Text boxes
- Kind of plot
  - Scatter plots
  - Bar plots
  - Histograms
  - ...

Scratched the surface today!