Lab 12: Plotting data (matplotlib)

Mar 8, 2018

Main Event

1. Practice plots

The objective of this lab is to reproduce various figures using Matplotlib with data from tips.csv. There is a large amount of documentation for Matplotlib online; two good references are:

- PyPlot API
- Matplotlib tutorial
- "Matplotlib Tutorial Series—Graphing in Python"

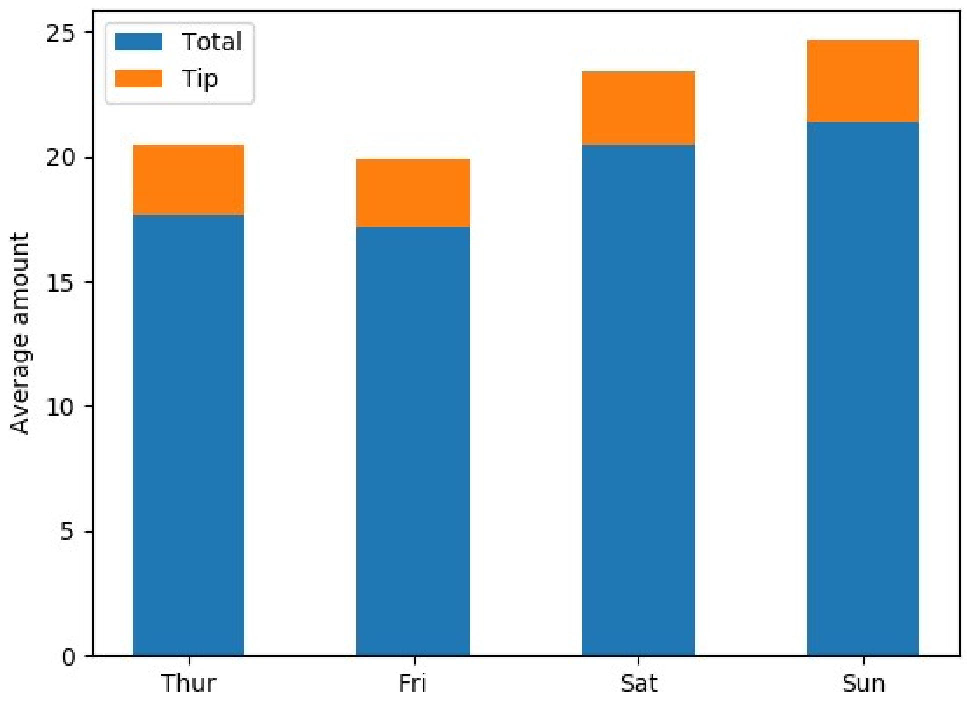
One of the best ways to learn, however, is by example. The Matplotlib gallery is full of them.

Solution:

```
# Values to help make later exercises easier. This step was not
# required! Solutions presented assume this file is called 'intro.py'

# List of days, formatted and in order
days = [ 'Thur', 'Fri', 'Sat', 'Sun' ]

# Function that reads the tips file and returns a list containing
# dictionaries of restaurant transactions.
def tip_aggregator():
    data = []
    numeric = [
        'total_bill',
        'tip',
        'size',
    ]
    fp = open('tips.csv')
    header = fp.readline().strip().split(',')

    for line in fp:
        row = line.strip().split(',')

        d = {}
        for i in range(len(header)):
            key = header[i]
            value = row[i]
            if key in numeric:
                value = float(value)
            d[key] = value

        data.append(d)
    fp.close()

    return data
```

2. Bar chart

1. Draw a “stacked” bar plot consisting of the average total bill and the average tip for given days of the week:

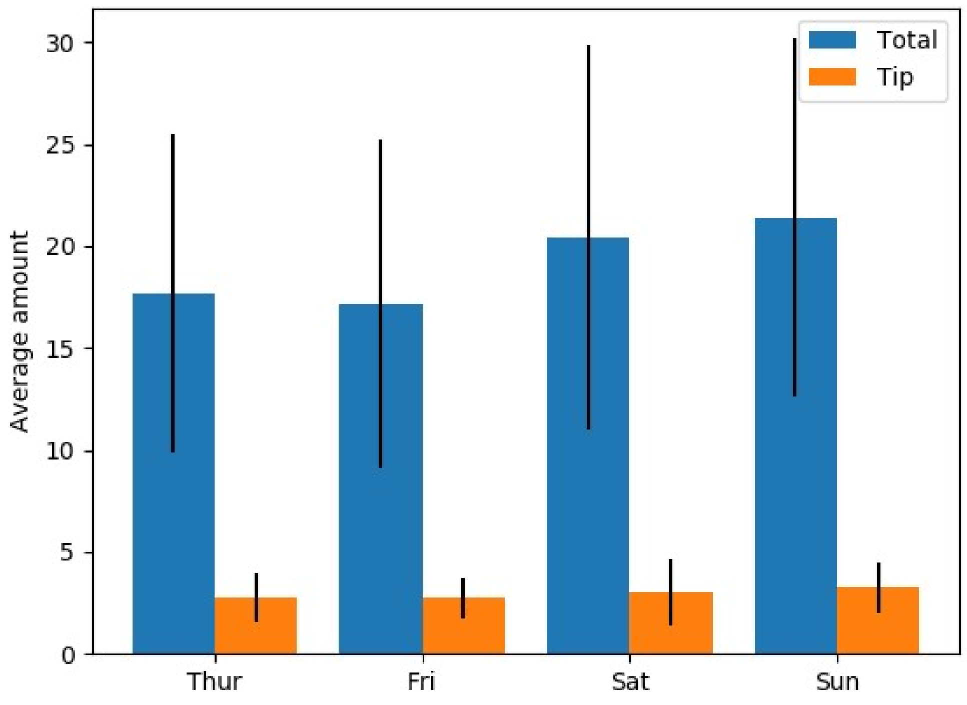


You will first have to obtain the average values for each column, for each day. You can use numpy to find the mean:

```
>>> import numpy as np
>>> np.mean([1, 2, 3, 4])
2.5
```

You will then have to draw the figure—the Matplotlib documentation is a good place to start in learning to do this. Make sure to include the legend!

2. Create a new figure (a new Python file) based on the previous bar chart. This time, however, place the respective averages side-by-side and include error bars:



See Matplotlib's example. Again, numpy can help:

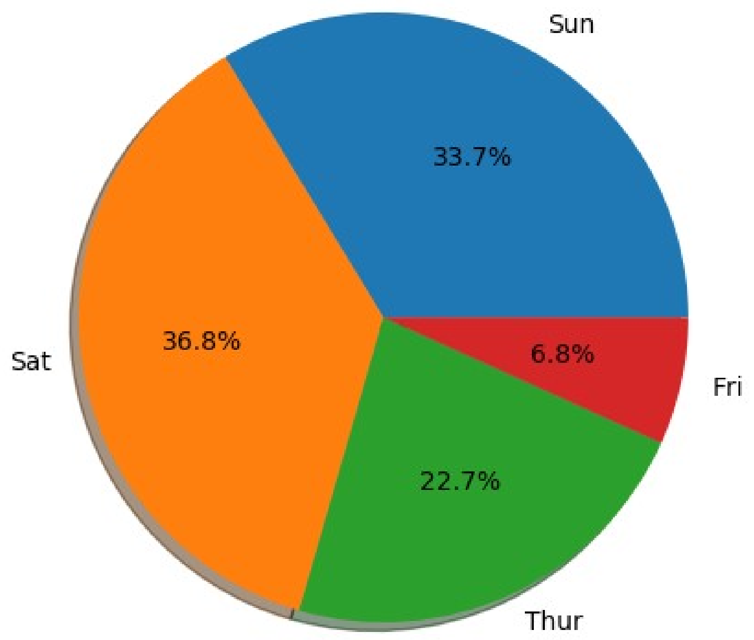
```
>>> # assuming numpy was previously imported...
>>> np.std([1, 2, 3, 4])
1.1180339887498949
```

Solution:

```
import numpy as np
import matplotlib.pyplot as plt

import intro

# aggregate total bill and tips
total = {}
tip = {}

for i in intro.tip_aggregator():
    d = i['day'].title()

    if d not in total:
        total[d] = []
    total[d].append(i['total_bill'])

    if d not in tip:
        tip[d] = []
    tip[d].append(i['tip'])

#
# 1. Stacked
#

# calculate averages
total_means = []
tip_means = []

for i in intro.days:
    total_means.append(np.mean(total[i]))
    tip_means.append(np.mean(tip[i]))

# plot the data
left = list(range(len(intro.days)))
width = 0.5

p1 = plt.bar(left, total_means, width, align='center')
p2 = plt.bar(left, tip_means, width, align='center', bottom=total_mear

plt.xticks(left, intro.days)
plt.ylabel('Average amount')
plt.legend((p1[0], p2[0]), ('Total', 'Tip'))

plt.savefig('bar-1.png')

#
# 2. side-by-side with error bars
#
plt.clf() # clear the canvas

# calculate the deviations
total_deviations = []
tip_deviations = []

for i in intro.days:
    total_deviations.append(np.std(total[i]))
    tip_deviations.append(np.std(tip[i]))

# plot the data
width = 0.4
right = []
for i in left:
    right.append(i + width)

p1 = plt.bar(left, total_means, width, yerr=total_deviations)
p2 = plt.bar(right, tip_means, width, yerr=tip_deviations)

xticks = []
for i in left:
    xticks.append(i + width / 2)
plt.xticks(xticks, intro.days)
plt.ylabel('Average amount')
plt.legend((p1[0], p2[0]), ('Total', 'Tip')) # better with loc='upper

plt.savefig('bar-2.png')
```

3. Pie chart

Create a pie chart in which each slice is the percentage of the respective days bills to the overall total:



In this case, to get the respective slices labeled with the percentage, use:

```
autopct='%1.1f%%'
```

in the `plt.pie` function. Further, in order to get the chart to appear as a circle, instead of an ellipse, use

```
>>> plt.axis('equal')
```

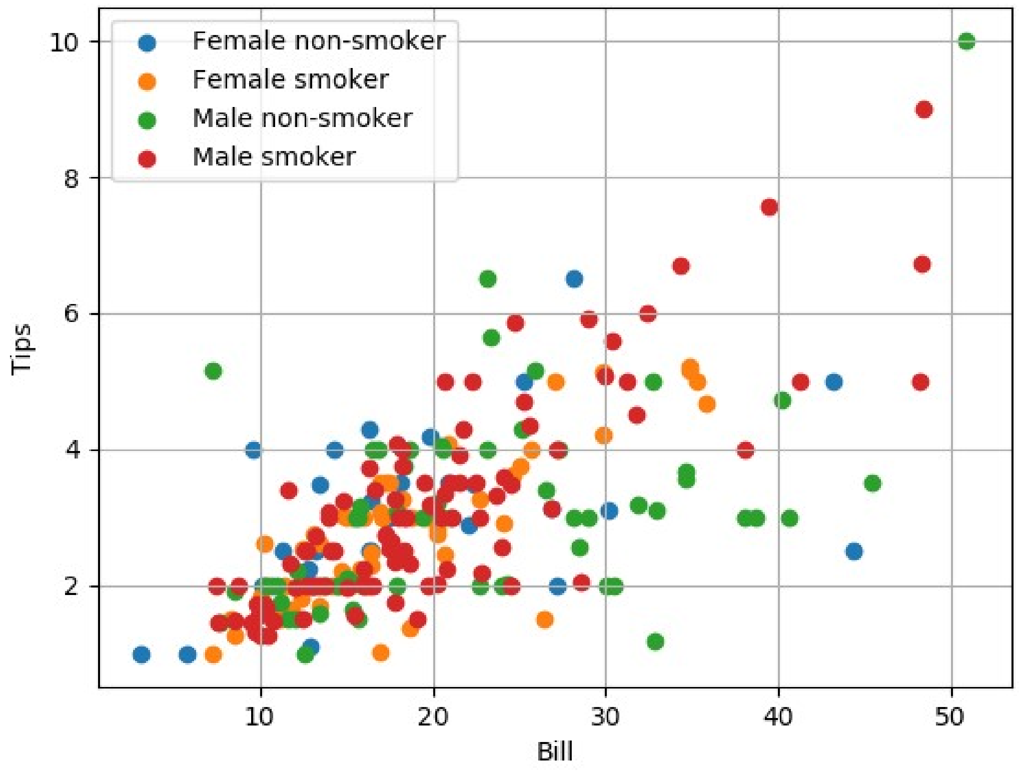
Solution:

```
import matplotlib.pyplot as plt

import intro

# aggregate total bills
bills = {}
for i in intro.tip_aggregator():
    day = i['day'].title()
    if day not in bills:
        bills[day] = 0
    bills[day] += i['total_bill']
```

```
# put days and their totals into two lists whose indices correspond
data = []
labels = []
for i in bills:
    data.append(bills[i])
    labels.append(i)

plt.pie(data, labels=labels, shadow=True, autopct='%1.1f%%')
plt.axis('equal')
plt.savefig('pie.png')
```

4. Scatter plot

A scatter plot can be a good way to recognize clusters in your data. In this case, the total bill versus the amount tipped for various patrons:



Colors represent an individuals gender, as well as their smoking preference.

It is recommended that you develop this in phases:

1. Create a scatter plot that does not contain a legend. To do this, create three lists—one for total bill, another for tipped amount, and a third for colors—in

which each column corresponds to an individual patron. This will allow you to make one call to `plt.scatter`.

2. Adding a legend requires separate calls to `plt.scatter` for each "categorzation" of the data. That is, for each set representing male/female and smoker/non-smoker, make separate calls to `plt.scatter`. See Stack Overflow for an example.

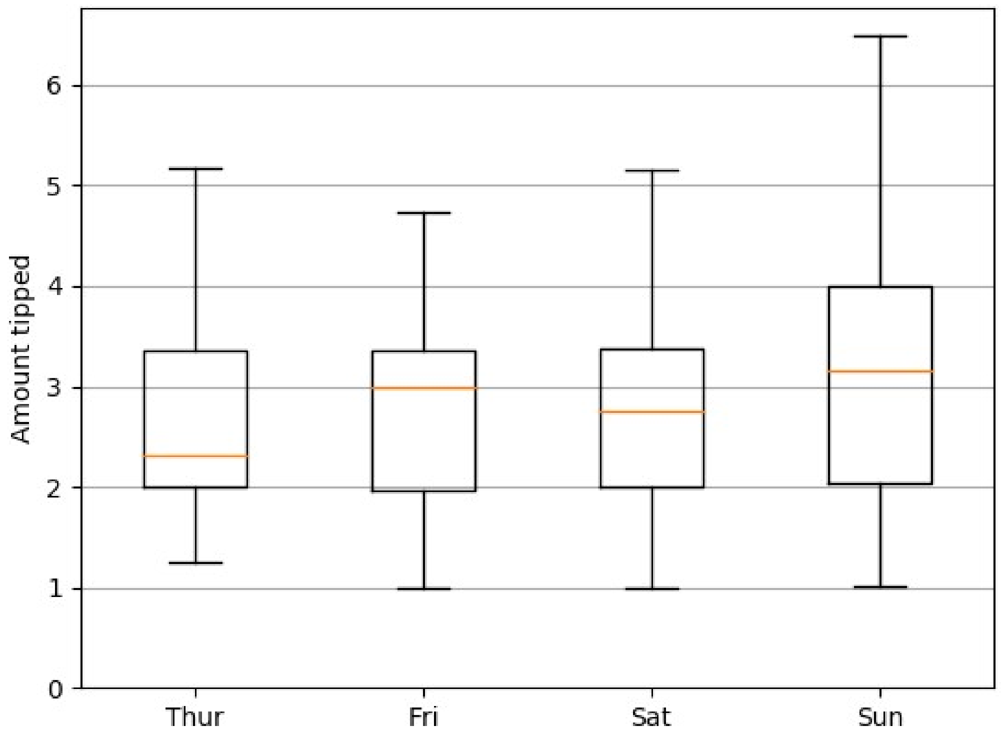
Solution:

```python
import matplotlib.pyplot as plt

import intro

tips = {}
totals = {}
legend = []

for i in intro.tip_aggregator():
    # "factors" are the dimensions in which we want to partition the
    # data. In this case by gender and smoking preference (2-by-2,
    # four in total)
    factor = i['sex'].title()
    smoker = 'smoker'
    if i['smoker'].lower() == 'yes':
        smoker = 'non-' + smoker
    factor += ' ' + smoker

    # each dimension will appear in the legend
    if factor not in legend:
        legend.append(factor)

    # tips per factor
    if factor not in tips:
        tips[factor] = []
    tips[factor].append(i['tip'])

    # total bills per factor
    if factor not in totals:
        totals[factor] = []
    totals[factor].append(i['total_bill'])

legend.sort() # so dimensions appear "together"
```

```
plots = []
for i in legend:
    p = plt.scatter(totals[i], tips[i], marker='o', edgecolors='face')
    plots.append(p)

plt.legend(plots, legend, loc='upper left')
plt.grid()
plt.xlabel('Bill')
plt.ylabel('Tips')
plt.savefig('scatter.png')
```

5. Boxplot

A boxplot is a good way to see the distribution of data. Here, the amount tipped each day:



The horizontal line in each box represents the average tip for the respective day.

The Pyplot documentation shows several examples of how to create boxplots (in the image above, `showfliers` is `False`), while Knowledge Stockpile provides a simple example that might be easier to follow.

The most difficult part is getting your data into a format the function understands: a list consisting of lists of data values. Specifically, if *d* were a list, *d*'s zeroth element would be a list of all tips on a given day. Keep in mind that sublists do not necessarily need to correspond (to day of the week, or time, for example), as the average and outlier calculation is not dependent on order.

Solution:

```python
import matplotlib.pyplot as plt

import intro

tips = {}

for i in intro.tip_aggregator():
    day = i['day'].lower()
    if day not in tips:
        tips[day] = []
    tips[day].append(i['tip'])

# put the tips into an order that corresponds to the order in which
# the days will be displayed
data = []
for i in intro.days:
    data.append(tips[i.lower()])

plt.boxplot(data, labels=intro.days, showfliers=False)
plt.ylim(ymin=0)
plt.ylabel('Amount tipped')
plt.grid(which='both', axis='y')
plt.savefig('boxplot.png')
```