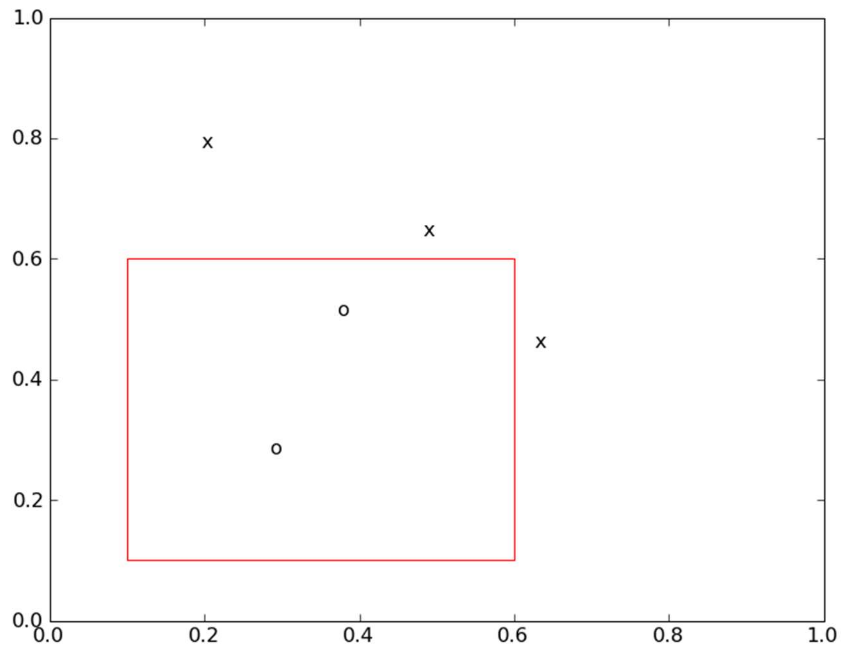
COMP 361/651 Data Analytics Techniques in Python Assignment 8

Possible points: 10

There are six problems. Do Problem 1 in Jupyter and the rest as .py files (probably in Spyder). Zip all files into one to submit. For Problem 6, use the Tkinter backend. (You might use it for Problems 2-5 as well.)

1 (2 pts.). Do this in Jupyter. Draw a red rectangle (a Patch) as shown in the figure below. The lower left corner is at (0.1, 0.1), and the upper right corner is at (0.6, 0.6). Define a handler for button-press events so that an '`x`' is added to the figure at the point where the mouse cursor is located when the button is clicked if the cursor is outside the rectangle. If the mouse cursor is inside the rectangle, an '`o`' is added at that location. (You can get the coordinates of the click event, and you have the x and y coordinates of the rectangle. Think of using global variables for the limits of the rectangle) If the mouse cursor is outside the axes when the button is clicked, nothing is added. The figure shows a situation where the button has been clicked three times with the mouse cursor outside the rectangle and twice with the mouse cursor inside. When you are using events (as here), the first line of your code should be

```
%matplotlib notebook
```



2 (2 pts.). Create two subplots in a row, as in the figure below. In both, the x-axis extends from -2π to 2π, and the y-axis extends from -2.5 to 2.5. The first subplot plots sin(*x*) (solid blue line) and 2sin(*x*) (dashed red line) from -2π to 2π using 40 equally spaced values for x. The second subplot plots cos(*x*) and 2cos(*x*) similarly. Include legends, and label the x-axis and y-axis as shown, sharing the y-axis. Both subplots have a grid and both have the x-axis and y-axis drawn as 1-point black lines. Save the figure to file. To fit both subplots conveniently in one row, use the following command to adjust the width of the figure.

```
plt.figure(2, figsize=(14, 4))
```

or, if you use the **`subplot()`** function,

```
  fig.set_size_inches(14,4)
```

Trying to execute something like

```
  plt.legend([p11, p12], ('sin(x)', '2 sin(x)'))
```

in (for this case) the first subplot (where **`p11`** and **`p12`** are the two curves) resulted for me in a warning. So, you might use the following (which result in the patches, not lines, shown in the figure below).

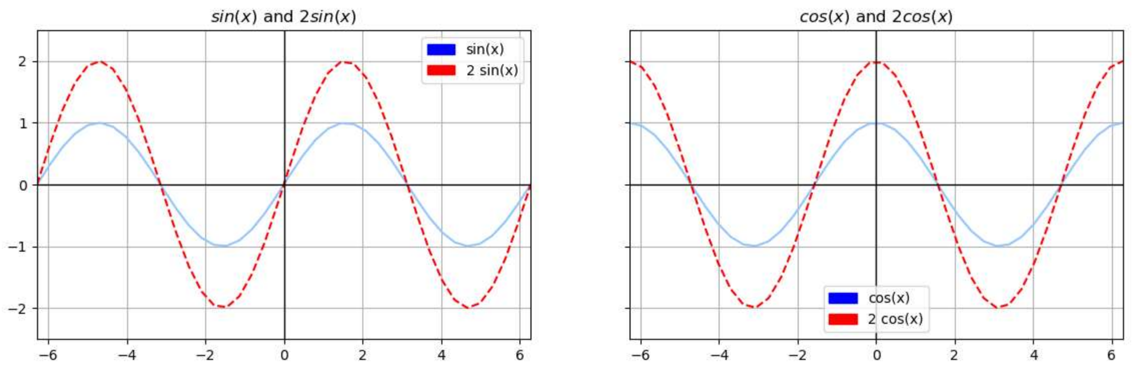
```
  import matplotlib.patches as mpatches
  blue_patch1 = mpatches.Patch(color='blue', label='sin(x)')
  red_patch1 = mpatches.Patch(color='red', label='2 sin(x)')
  plt.legend(handles=[blue_patch1, red_patch1])
```

If you use the **`subplot()`** function, replace the last of these three lines with

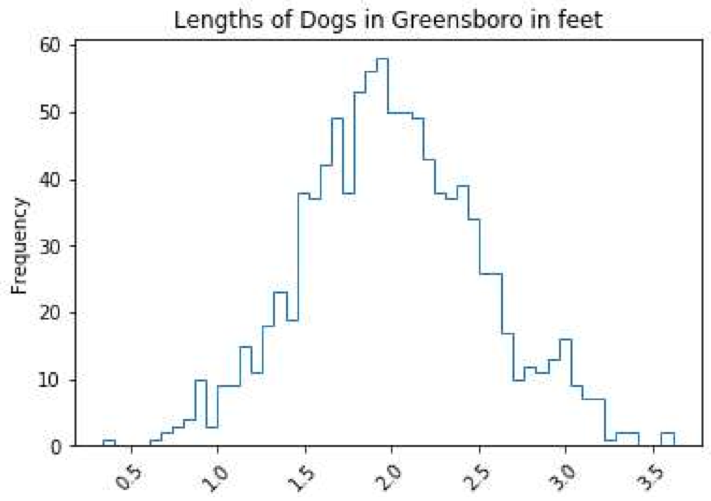
```
  axs[0].legend(handles=[blue_patch1, red_patch1])
```

You can handle the legend in the second subplot similarly.

Give each of the subplots titles as shown. Use LaTeX to get convincing renderings of the mathematical notation.



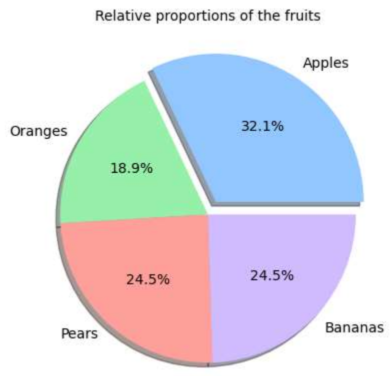
3 (1.5 pts.). Generate a list of 1000 numbers from a normal distribution (see **`np.random.normal()`**) with μ = 2 and σ = 0.5 and plot them in a step-style histogram (use **`histtype='step'`**) with 50 bins as shown in the figure below. Rotate the x-axis tick labels 45 degrees as shown (**`rotation=45`**). Include the *y* label and title shown. Save the figure to file.



4 (2 pts.). The contents of file **`fruits.txt`** (available for download from the assignment page) are shown below. Each row is an order for various numbers of fruits. The first column is apples, the second is oranges, the third is pears, and the fourth is bananas.

```
40 25 20 30
20 15 20 20
25 10 25 15
```

Read in this file and form a sum of each of the four kinds of fruit. Suppose that these numbers are in list **`cnts`** in the same order as the columns (i.e., apples, oranges, pears, bananas). From these numbers, you produce the pie chart shown at right.



To produce the pie chart, get an **`Axes`** instance,

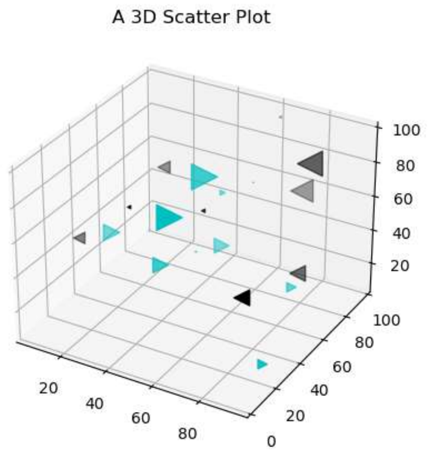
```
fig1, ax1 = plt.subplots()
```

Then call **`ax1.pie()`**, passing it **`cnts`** (which will determine the size of each wedge) and (for the **`labels`** keyword argument) a list of the names of the fruits in the same order as their counts occur in **`cnts`**. We also want a shadow under the lower part of the pie, so set **`shadow=True`**, and we want the percent appearing on each wedge to be a floating-point number with one digit after the decimal point, so set **`autopct`** accordingly. To get the Apples wedge exploded, set keyword argument **`exploded`** to a length-four list with all zeros except at the position for Apples, which should have 0.1.

After calling **`ax1.pie()`**, set the title and save the figure to a file.

5 (1.5 pts.) For this problem, you are provided on the assignment page files **`data1.txt`** and **`data2.txt`**, each with 10 lines of the form illustrated by the following.

```
9.44 54.10 34.16 6.00
```



The first three values are *x*, *y*, and *z* coordinates, and the next value is the radius. Each line specifies an extended 'dot' in a 3D scatter plot at the given coordinates with the given radius. If the 'dot' is given on **`data1.txt`**, it appears as a cyan right-pointing triangle, while if it is given on **`data2.txt`**, it appears as a black left-pointing triangle. The area is computed as πr^2 even though the 'dot' is a triangle. (Be sure to replace each value *r* with πr^2.) The figure at right is the resulting scatter plot. (You don't see 20 'dots' since some are occluded and some are too small.)

To avoid repeating code, the easiest way to approach this is to define a function that is passed the name of the file (**`data1.txt`** or **`data2.txt`**), the Axes, say, **`ax`**, on which the plot is draw, the marker ('**`>`**' or '**`<`**'), and the color ('**`c`**' or '**`b`**'). Before calling this function twice, be sure to execute

```
ax = plt.axes(projection="3d")
```

6 (1 pt.). Open the animated GIF file **`oscillator.gif`**, which you can download from the page for this assignment. You will see a small blue square moving counterclockwise around a circle of radius 1 centered on the origin. You are to write code to produce a GIF file just like this by modifying the program on slide 127 of the matplotlib slides.

The most fundamental change is to the following two lines in the **`animate(i)`** function.

```
x = np.linspace(0, 4, 1000)
y = np.sin(2 * np.pi * (x - 0.01 * i))
```

The parameter **`i`** is what changes (by increments of 1) from call to call and produces the perceived motion. In this case, the sine wave is progressively displaced to the right. To get a chunk of a curve to move, we slice it, and have the slice a function of **`i`**. To get circular motion on the unit circle, we note that, for any value of θ, the point (sin θ, cos θ) is on this circle. So, to get the desired animation, replace the above two lines with three lines, the first of which could be

```
theta = np.linspace(0, 4, 1000)
```

One of the other two can be just like the second line above except that, in place of **`(x - 0.01 * i)`**, you will have a slice of **`theta`** as a function of **`i`**. I had the slice four elements long and advancing four elements for each call to **`animate()`**. The remaining of the three lines is like this last but with **`cos()`** instead of **`sin()`**.

Set the dimensions of the **`Axes`** instance from -2 to 2 in both dimensions (leaving plenty of space for the unit circle). Increase the line width to, say, 10.

Save the figure to file.

For this, set the backend to **`Tkinter`** and restart the kernel. (See slide 109 of the matplotlib slides.)